أسس
OSOS

OSOS DOCUMENTATION

Build Apps

A section of the OSOS Documentation and Help Center

English edition

Platform release 1.10
Published 23 November 2026

Guides · How-to · Tutorials

Contents

Build apps — start here 5

Guides

My First Application

My First Application: Team Tasks 7
Plan your app 9
Before you start 11

Build the app

Build the app 12
Create the module 13
Create the Tasks entity 15
Create the queries 18
Build the views 20
Add a dashboard 23
Notify the assignee 24
Send a weekly report 26
Build the home page 28

Try it as an end user

Try it as an end user 31
Try it on the web 32
Try it on mobile 34
Change your app safely 35
Builder concepts and glossary 38
Create and design an entity 45
Modules 50

Choose a field type

Choose a field type 52
Settings every field shares 55
Text field 61
Long-text field 63
Localized Text field 65
Number field 66
Date field 68
Time field 70
Date-time field 71
Month-year field 72
Year field 73
Month field 74
Block field 75
Boolean field 77
Category field 78
Document field 80
File field 81
Icon field 83
Image field 84
Map field 85
Password field 86
Phone field 87
Reference field 88
Sequence Field 90

Switch field 93
User field 94
Video field 96
Mapped Field 97
SVG Icon field (internal) 99
Application Reference field 100
Page Reference field (internal) 101
Data Query Selector field (internal) 102
Number Enum field (internal) 103
JSON field (internal) 104
Link entities 105
Virtual entities 108
Data Views & Faces 111
Pages & Widgets 122
Queries & filters 126
Reports 132
Summaries, dashboards & counters 135
Totals & grouping 139
Workflows & approvals 143
Reactions & automation 150
Buttons 159
Export templates 164
Export, import and install apps 169
Counters and record numbers 176
Timeline & History 178
Share an entity's records across spaces 182
Protect a field 184
Preview Mode 188
The Admin Bar 191
Lists in Administration 193

How-to

Quick videos

Quick videos 196
Publish a service with an approval workflow 197
Change an entity safely 201

Build recipes

Build recipes 204
Creating a module 205
Editing a module 206
Creating an entity 207
Editing an entity 209

Tutorials

Tutorials 213

Customer support system

Customer support system 214
Understand the business need 216
Plan the system 217
The build steps 219

Build the system

Build the system 220
Create the module 221
Create the entity 224
Build the workflow 228

Add the notifications 233
Create the queries 236
Build the views 238
Add the status chart 245
Create the monthly report 248
Test the system 253

Building a Recruitment System

Building a Recruitment System 261
System Explanation 262
What does the system need? 264
Building Steps 273

Building the System

Building the System 275
Creating the Recruitment Module 276
Job Application Service 279
Creating Queries 326

Entities

Entities 329
Employees Entity 330
Receptions Entity 337
Jobs Entity 341

Views

Views 367
Building the Receptions Views 368
Building the Jobs Views 373
Building the Job Application Views 379
Dashboards 383
Creating a Report 388
Building the Home Page 393
Testing the Recruitment System 408
Build apps FAQ 427

Build apps — start here

This section is for **builders**: people who design applications in OSOS without writing code. If you can open **Administration** and you have **Developer** access to a module (ask your tenant administrator if you don't), you can build a complete business application here — its data, its forms, its approval process, its automation and its dashboards.

> ### Integrating via the API? Start here instead
>
> If you are connecting another system to OSOS — calling the HTTP API with a token, receiving webhooks, or writing functions — this is not your starting point. Go to Integrate & extend — start here [Integrate & Extend], which walks you from your first authenticated call to reading and writing records.

Where the builder lives

Everything you build is designed in the **Administration** area of the portal:

You want to…	Open
Create a module, design entities, build views, pages and dashboards	**Administration › Applications** (**Modules**, **Data Entities**, **Lists**, **Pages**, **Views**, **Dashboards**, **Counters**)
Publish a service that people request from the portal	**Administration › Services › Services Catalog**
Look after shared look-and-feel and document templates	**Administration › Interfaces**
Move an application to another environment	**Administration › Packages**

Each entity has its own designer, with tabs such as **Details**, **Structure** and **Workflows**. Most of the guides below start there.

How an application fits together

An application is built from a handful of pieces, each resting on the one before it. Take a small **Asset Request** application as an example:

1. **Module** — the container for the whole application: *Asset Management*.
2. **Entities and fields** — one entity per kind of record, each with its own fields: an *Asset Request* entity with *Asset type*, *Reason*, *Needed by*.
3. **Views and forms** — how people list, open, add and edit records: a *My asset requests* list and the request form.
4. **Services and workflows** — the process a record goes through: the requester's manager approves, then the IT team fulfils it. Publish the entity as a **service** so employees can request it from the portal.
5. **Reactions** — automatic jobs that run when something happens: email the requester when the request is approved.
6. **Queries, reports and dashboards** — find and summarise records: *open requests by department*, a weekly report to the IT manager.
7. **Pages** — put it all in front of people: an *IT services* home page with a summary widget and links.

Information-structure concepts explains each piece in more depth, along with the shared ideas you will meet everywhere (identifiers, normalized names, pointers such as **@me**, core entities and attachments).

Recommended learning path

1. **My First Application** — build a small task-management app end to end: module, entity, views, notifications, a page, then use it as an end user.
2. **Publish a service with an approval workflow** — the Asset Request example above: a request form, a two-step approval (the requester's manager, then the IT team), published to the portal.
3. **Build recipes** — short, step-by-step procedures for creating and editing modules and entities.
4. **Tutorials** — larger worked systems once you are comfortable with the basics.
5. The **goal-based guides** below, whenever you need one feature in depth.

Find a guide by goal

I want to…	Guide
Store records and decide what each record holds	Entities · Field types · Relations
Group my app's pieces together	Modules
Show, add and edit records	Data views · Preview mode
Find the right records	Queries · Aggregates and grouping
Route a record for approval	Workflows & approvals · Publish a service with an approval workflow
Automate work (emails, updates, reminders)	Reactions & automation · Buttons
Summarise and report	Dashboards & summaries · Counters · Reports · Grafana dashboards [Integrate & Extend › Grafana Dashboards]
Produce documents	Templates
Build pages and a home page	Pages & widgets · Timeline
Protect personal data in a field	Sensitive fields
Move my app to another environment	Packaging [Integrate & Extend › Packaging & distribution]
Get a quick answer	Development FAQ [Integrate & Extend › Integration FAQ]

ⓘ Go deeper

For integrators: OSOS as a Backend [Integrate & Extend] · HTTP API [Integrate & Extend › Calling the HTTP API] · Webhooks & integrations [Integrate & Extend] · Functions (FaaS) [Integrate & Extend › Functions: run your own code]

My First Application: Team Tasks

This walkthrough is for **builders** who are new to OSOS. Step by step, you will build a small **Team Tasks** application: managers create tasks and assign them to employees, employees see and update their own tasks, everyone is told when a task is assigned to them, and managers get a weekly summary. You need no code.

Time: about 90 minutes, in short steps you can do one at a time.

You need:

- Access to **Administration** with **Developer** rights. If you cannot see **Administration › Applications › Modules**, ask your tenant administrator.
- A few test accounts and two groups. Before you start lists exactly what to ask for.

What you will build

Piece	In Team Tasks
Module	*Task Management* — the container for the whole app
Entity	*Tasks*, with a title, description, status, assignee and due date
Queries	*My open tasks* and *Open tasks*
Views	*My Tasks* for employees and *Manage Tasks* for managers
Dashboard	*Tasks by status*, a chart on the *Manage Tasks* view
Reactions	An in-app notification and an email to the assignee
Report	*Open tasks*, emailed to managers every Sunday
Page	A *Team Tasks* home page with links, a task list and the chart

Follow the steps in order

Each step uses something the step before it created.

1. Plan your app — the scenario, the people, and the exact names you will use.
2. Before you start — accounts, groups and access.
3. Create the module
4. Create the Tasks entity — fields and form layout.
5. Create the queries — *My open tasks* and *Open tasks*.
6. Build the views — *My Tasks* and *Manage Tasks*.
7. Add a dashboard — a *Tasks by status* chart.
8. Notify the assignee — in the app and by email.
9. Send a weekly report
10. Build the home page
11. Try it as an end user — on the web and on mobile.
12. Change your app safely — add a field, sub-tasks and a counter without breaking existing tasks.

After this walkthrough

- Add an approval step: Publish a service with an approval workflow builds a request that a manager approves.
- Look up a single procedure: Build recipes.
- Build something bigger: Tutorials.

Plan your app

Step 1 of 12. Before you open the builder, decide what the app must do and what each piece will be called. This page gives you the plan for **Team Tasks**; every later step uses the names in the tables below, so keep this page open.

The scenario

A team's work is tracked in email and spreadsheets. Tasks get lost, nobody knows who is doing what, and managers spend Monday mornings chasing updates. The team wants one place where:

- a manager creates a task, gives it a due date and assigns it to an employee;
- the employee is told straight away — in the app and by email;
- each employee sees their own open tasks and moves them from *To Do* to *In Progress* to *Completed*;
- managers see every open task, a chart of tasks by status, and a weekly report.

Who uses it

Person	Group	What they do
Manager	*Task Managers*	Creates and assigns tasks, follows the team's progress, receives the weekly report.
Employee	*Employees*	Sees their own tasks and updates their status.

The plan

The Tasks entity

One entity, *Tasks*, holds every task. Each task has these fields:

Field	Type	Required	Notes
Title	Text	Yes	
Description	Long-text	No	
Status	Category	Yes	Options **To Do**, **In Progress**, **Completed**. Default **To Do**.
Assigned To	User	Yes	The employee doing the task.
Due Date	Date	No	

These three status values are the only ones used anywhere in this walkthrough.

Everything else

Piece	Name	What it does	Built in step
Module	*Task Management*	Holds everything below and gives the app its place in the sidebar.	3
Query	*My open tasks*	Tasks assigned to the person looking, not yet *Completed*.	5
Query	*Open tasks*	All tasks not yet *Completed*.	5
View	*My Tasks*	Employees see and update their own open tasks.	6

Piece	Name	What it does	Built in step
View	*Manage Tasks*	Managers add, assign and edit every task.	6
Summary	*Tasks by status*	A chart counting tasks per status, shown on *Manage Tasks*.	7
Reaction	*Notify assignee*	An in-app notification to the assignee when a task is created.	8
Reaction	*Email assignee*	The same message by email.	8
Report	*Open tasks report*	A list of open tasks, emailed to *Task Managers* every Sunday.	9
Page	*Team Tasks*	The app's home page.	10

Why plan names first?

Names show up everywhere — in the sidebar, in notifications, in reports. Each piece also gets a **normalized name** from its English name when you create it, and that internal name should not change afterwards. Deciding names up front saves renaming later.

Next step

Before you start — get the accounts, groups and access the walkthrough needs.

Before you start

Step 2 of 12. Building the app needs a few things that a **tenant administrator** sets up, not a builder. Ask for them now so you are not blocked halfway through.

What to ask your tenant administrator for

1. **Developer access for you.** You must be able to open **Administration › Applications › Modules** and create a module. Creating modules and entities is part of a tenant's system configuration, so your administrator may need to grant it.
2. **Three test accounts**, with real email addresses you can read:
    - *Manager One* — a manager.
    - *Employee One* and *Employee Two* — employees.
3. **Two groups:**
    - *Task Managers*, containing *Manager One*.
    - *Employees*, containing *Employee One* and *Employee Two*.
4. **Outgoing email working** in your space, so the email notification and the weekly report in steps 8 and 9 can be delivered.

You can send this list as it is. Administrators create users in **Administration › Organization › Users** and groups in **Administration › Organization › Groups**; see Managing users [Administration] and Managing groups [Administration].

Check before moving on

- ☐ You can open **Administration › Applications › Modules**.
- ☐ You can sign in as *Manager One* and as *Employee One* (for example in a private browser window), and each lands on the portal.
- ☐ The groups *Task Managers* and *Employees* exist with the right members.

Next step

Create the module.

Build the app

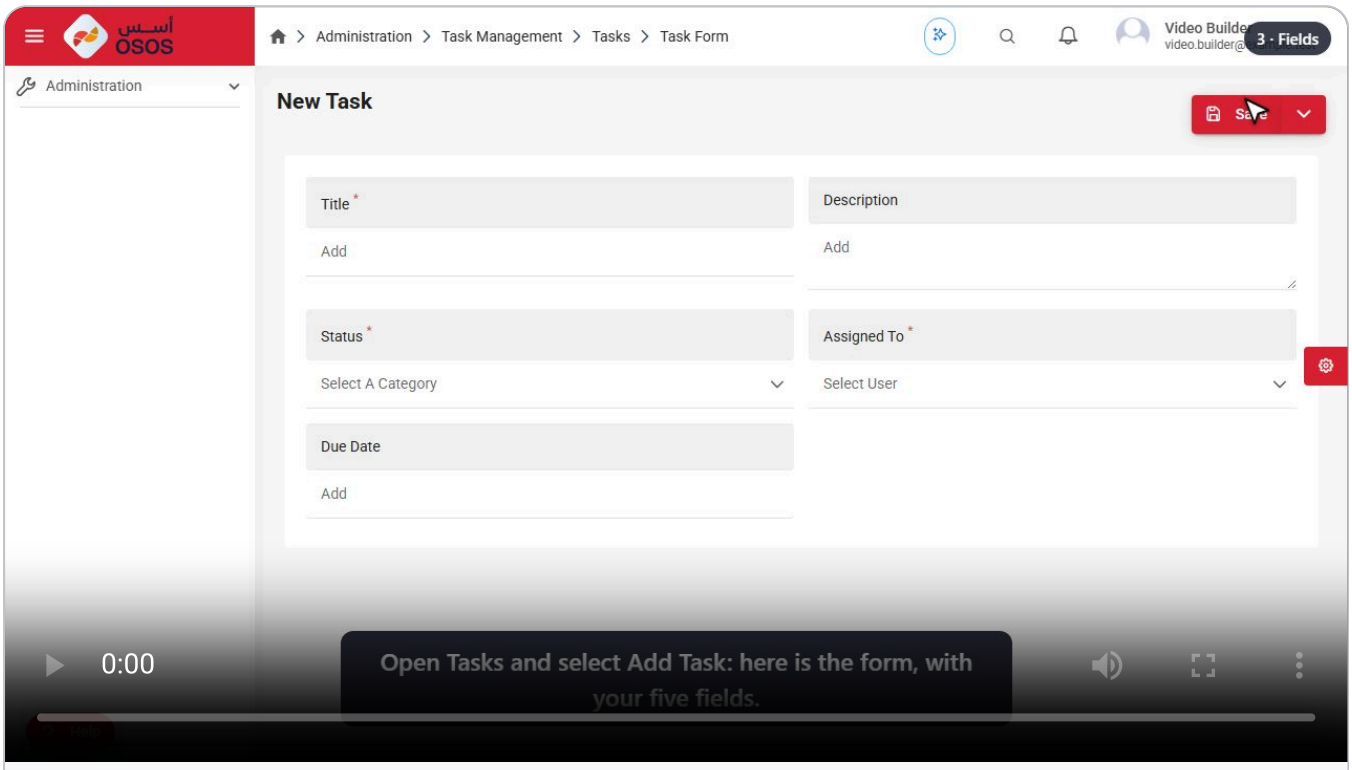
These are steps 3 to 10 of My First Application. Do them in this order — each one uses something the one before it created.

3. Create the module — *Task Management*.
4. Create the Tasks entity — its fields and form layout.
5. Create the queries — *My open tasks* and *Open tasks*.
6. Build the views — *My Tasks* and *Manage Tasks*.
7. Add a dashboard — a *Tasks by status* chart.
8. Notify the assignee — in the app and by email.
9. Send a weekly report — to the managers, every Sunday.
10. Build the home page — *Team Tasks*.

Then try it as an end user.

Create the module

Step 3 of 12. A **module** is the container for one application: its entities, views, pages and dashboard all live inside it, and it gets its own entry in the sidebar. Here you create the *Task Management* module.



TUTORIAL · PART 1 OF 3 **The module and the Tasks entity**

Create the Task Management module and its Tasks entity · 2:06 · Recorded on OSOS 1.11.0-alpha.1

Steps

1. Open **Administration › Applications › Modules**.
2. Select **Add Module**. The **New Module** dialog opens.
3. Fill in:
    - **Name** — *Task Management*, in Arabic and English.
    - **Normalized name** — filled in for you from the English name (for example *task_management*). Keep it: it is set when the module is created and does not change afterwards.
    - **Sharing** — leave it as it is.
    - **Icon** — pick one, for example a checklist icon.
    - **Description** — optional, for example *Team task tracking*.
    - **Administration Access** — leave empty for now. You can let other builders work on the module later.
4. Select **Done**. The module appears in the list.

> **Note**
>
> A new module comes with a default **landing page** and a **dashboard**, so it appears in the sidebar straight away. You will replace the landing page with your own home page in step 10.

The full procedure, with every option, is in the recipe Creating a module.

Check before moving on

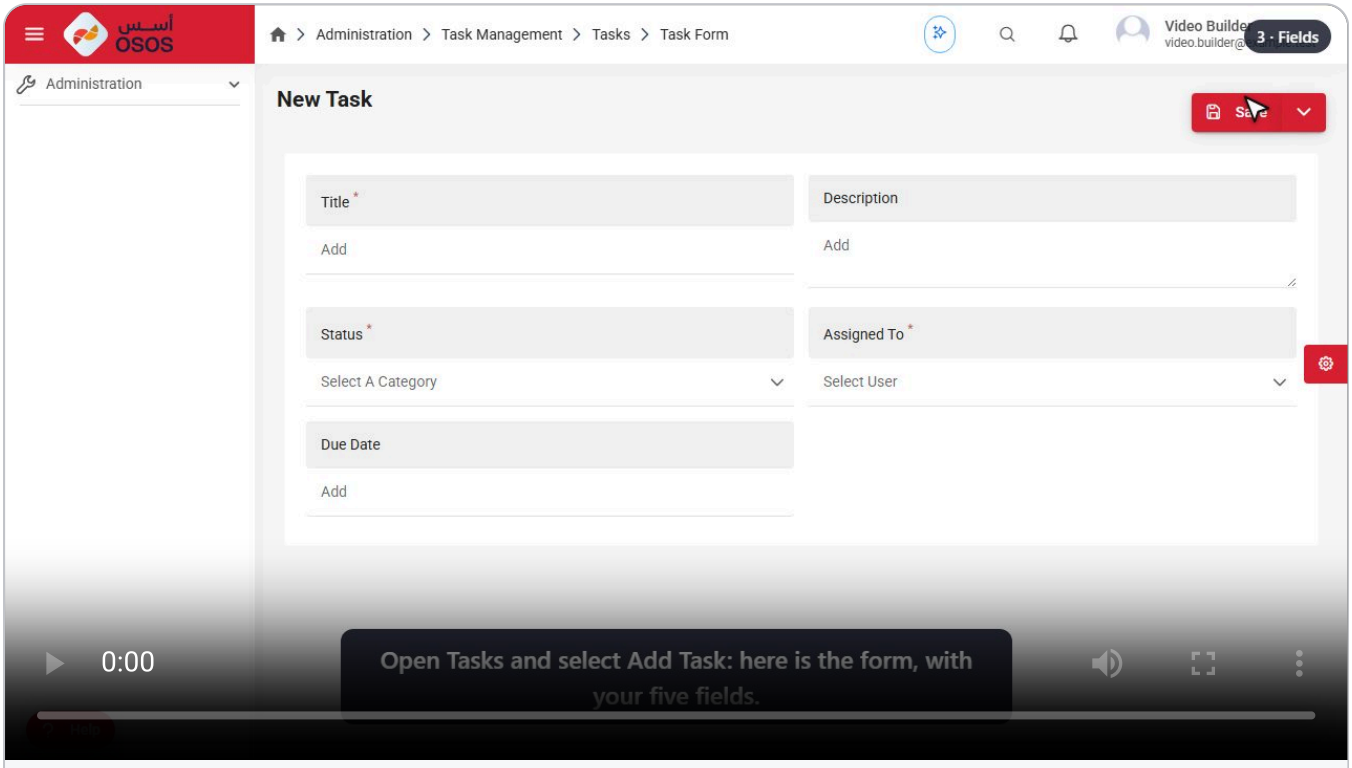
- [ ] *Task Management* is in the **Modules** list.

Next step

Create the Tasks entity.

Create the Tasks entity

Step 4 of 12. An **entity** is one kind of record — here, a task. It defines the fields every task has; each task people create is a **record** (the screens also call it an *entry*). You create the *Tasks* entity, add its fields, and lay out its form.



TUTORIAL · PART 1 OF 3 The module and the Tasks entity

Create the Task Management module and its Tasks entity · 2:06 · Recorded on OSOS 1.11.0-alpha.1

Create the entity

1. Open **Administration › Applications › Modules** and open *Task Management*.
2. On the **Entities** tab, select **Add Data Entity**. The **Add New Data Entity** dialog opens.
3. Fill in:
   - **Name** — *Tasks*, in Arabic and English.
   - **Normalized name** — filled in from the English name (*tasks*). Keep it: queries, reactions and packages refer to the entity by this name, so treat it as permanent.
   - **Select Icon** — for example *check-square*.
   - **Choose Module** — already set to *Task Management*.
   - **Data Entity Type** — keep **Normal Entity**, the type for records you define yourself.
   - Under **Field Settings**, set **Single Name** to *Task* and **Plural Name** to *Tasks*. The portal uses them on buttons and lists — for example **Add Task**.
4. Select **Create**. The entity's designer opens, with the tabs **Details**, **Structure**, **Workflows**, **Timeline** and **Indexes**.

Add the fields

1. To come back to the designer later, open the entity, open the menu next to **Add Task** at the top and select **Edit Entity**.

2. Select the **Structure** tab. The **Fields** palette lists the field types.

3. Drag each field type from the palette onto the empty area, then type its name in the row:

Drag this type	Name it	Then
Text	*Title*	Tick **Required**.
Long-text	*Description*	
Category	*Status*	Tick **Required**. Open the field's settings (the gear icon on its row) and, under **Add options**, add **To Do**, **In Progress** and **Completed**, each with a colour. **Default Value** lists the options only after the entity has been saved with them: once you have saved the entity (below), reopen the field's settings and set it to **To Do**.
User	*Assigned To*	Tick **Required**.
Date	*Due Date*	

4. Select **Save**, then confirm with **Save Changes**.

> ### Other field options
>
> A field's settings also let you make it unique, searchable, hidden or locked, give it a calculated value, or mark it **Sensitive** so only some people can read it. The full list of field types is in Field types; protected fields are covered in Sensitive fields.

Lay out the form in two columns

1. On the **Structure** tab, open **Form Settings** (the settings icon at the top right of the field list).
2. In **Choose Display Layout**, pick a two-column layout.
3. Select **Save**, then **Save Changes**.

Your entity is ready

There is no separate publish step for an ordinary entity: once it is saved, people with access can create tasks in it. (A *service* — an entity people request from the portal's catalog — is different: it must be activated. See Publish a service with an approval workflow.)

To check, open the entity's first tab, **Tasks**, select **Add Task**, and create a test task. Delete or archive it afterwards if you like.

Check before moving on

- ☐ *Tasks* has the five fields above, and *Status* has exactly the three options **To Do**, **In Progress** and **Completed**.

- ☐ You can add a task from the **Tasks** tab.

Next step

Create the queries. The general procedure is in the recipe Creating an entity.

Create the queries

Step 5 of 12. A **query** is a saved filter on an entity. Views, dashboards, reports and counters use queries to decide which records to show. You create two:

Query	Conditions	Used by
My open tasks	*Assigned To* is the person looking, and *Status* is not *Completed*	The *My Tasks* view (step 6) and the home page (step 10)
Open tasks	*Status* is not *Completed*	The weekly report (step 9)

The first query uses the pointer **@me**, which stands for whoever is looking at the list. One saved query therefore shows each employee their own tasks.

Create “My open tasks”

1. Open **Administration › Applications › Modules**, open *Task Management*, and open the *Tasks* entity.
2. Select the **Queries** tab, then select **+** (**New Query**).
3. Keep the group set to **All conditions are met**, so a task must meet every condition.
4. Select **Condition** and set:
    - **Field** — *Assigned To*.
    - **Operator** — **=**.
    - Value source — **Pointer**, then pick **@me**.
5. Select **Condition** again and set:
    - **Field** — *Status*.
    - **Operator** — **!=** (does not equal).
    - Value source — **Fixed**, then pick **Completed**.
6. Under **Query Settings**, enter the **Query Name** *My open tasks* in Arabic and English.
7. Select **Save Query**.

Create “Open tasks”

Repeat the steps with a single condition — *Status* **!= Fixed** *Completed* — and name the query *Open tasks*.

Check before moving on

- ☐ Both queries are listed on the **Queries** tab.
- ☐ Add a test task assigned to yourself with status **To Do**. On the **Queries** tab, select **Apply query** next to *My open tasks*: your test task is listed. Change its status to **Completed**, apply the query again, and it is gone.

More you can do with queries

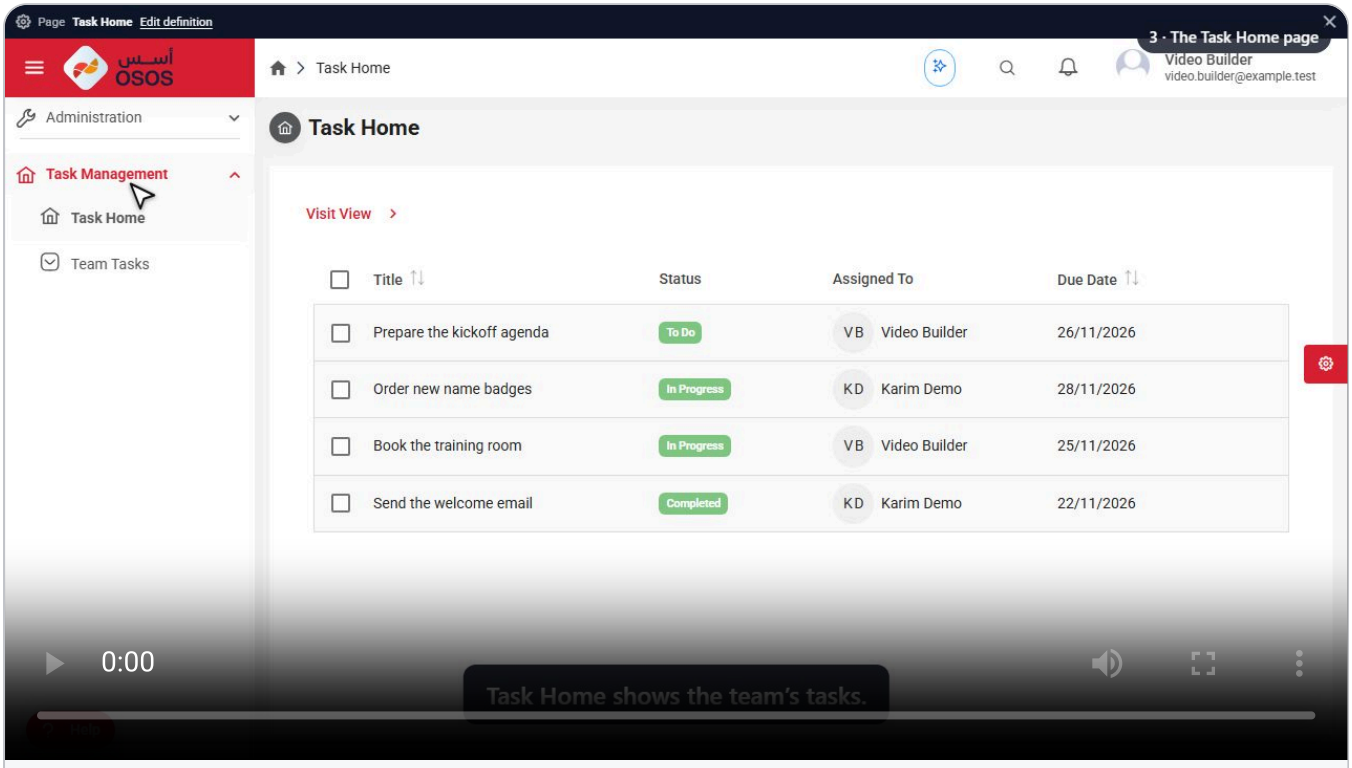
Queries can also ask the person for a value when they run, become one-click **quick filters**, or colour matching records with **tags**. See Queries.

Next step

Build the views.

Build the views

Step 6 of 12. A **view** is how people see and work with records in the portal: a list (table, kanban board, calendar…), the record's details, and the buttons to add or edit. One entity can have several views for different people. You build two:



TUTORIAL · PART 2 OF 3 The Team Tasks view and the Task Home page

A table and a Kanban board, opened to a group, on a home page · 3:06 · Recorded on OSOS 1.11.0-alpha.1

View	For	Shows	People can
My Tasks	*Employees*	*My open tasks*	Open a task and change its status
Manage Tasks	*Task Managers*	Every task	Add, edit and archive tasks

Build "My Tasks"

1. Open the *Tasks* entity (**Administration › Applications › Modules ›** *Task Management* **›** *Tasks*) and select the **Views** tab.
2. Select **New View**. The view designer opens on its **Settings** tab.
3. Under **View information**, enter the **Name** *My Tasks* in Arabic and English and pick an **Icon**.
4. Under **Type & source**, set **Source view** to **Queries**, then choose the **Query** *My open tasks*. Under **Fields**, add all five fields: a view can show only the fields you add here.
5. Under **Appearance**, turn on **Pin to sidebar** so the view appears under *Task Management* in the sidebar.
6. Under **Access Permission**, turn on **Authenticated**, then pick *Employees* in **Select groups**. Only signed-in members of *Employees* can open the view.

7. Turn on **Entries list** and, under **Displays**, turn on:
    - **Table view** — choose the fields *Title*, *Status* and *Due Date*.
    - **Kanban view** — set **Main title** to *Title* and **Category** to *Status*, so tasks appear in **To Do**, **In Progress** and **Completed** columns.
8. Turn on **Entries details** and choose all five fields, so a task opens with everything on it.
9. Turn on **Editing entries** and choose the field *Status*, so employees can move their tasks along without changing anything else. Under **After Save**, choose **Go to entries list**.
10. Select **Save**. It stays unavailable, with no message, until every required setting is filled in: check the **Fields**, the Kanban **Main title**, and an **After Save** choice for every operation you turned on.
11. Back on the **Views** tab, select **Publish** on the *My Tasks* card. A view is not visible to anyone until it is published.

Build “Manage Tasks”

Repeat the steps with these differences:

Setting	Value
Name	*Manage Tasks*
Source view	**Entries** (every task)
Select groups	*Task Managers*
Table view fields	All five fields
Turn on	**Creating new entries** and **Editing entries** (all fields), **Archive entries**, and **Archived entries** (so managers can list archived tasks); give **Creating new entries**, **Editing entries** and **Archive entries** an **After Save** choice each, such as **Go to entries list**

Select **Save**, then **Publish** the view.

Who can open a view

Access Permission offers two switches:

- **Authenticated** — people who are signed in. Pick users in **Select Users** or groups in **Select groups** to narrow it down; leave both empty to allow everyone who is signed in.
- **Anonymous** — anyone with the link, signed in or not. Use it only for information meant for the public, such as a visitor page. Never use it for a personal list like *My Tasks*: **@me** needs a signed-in person, and the records would be exposed to anyone who finds the link.

A view controls *how* people see records. What they are allowed to do with the entity’s data overall is set by permissions — see Permissions [Administration › Permissions & access].

Other view options

Views can also show records as cards, a calendar, a tree or a timeline; ask people to fill in a filter before anything loads; offer one-click quick filters; and preview the layout on a phone before publishing. See Data views and Preview mode.

Check before moving on

- [ ] Both views are published.
- [ ] Signed in as *Employee One*, you see **My Tasks** under *Task Management* in the sidebar, and not **Manage Tasks**.

Next step

Add a dashboard.

Add a dashboard

Step 7 of 12. A **summary** turns records into a number, a short list or a chart. Summaries are collected on the entity's **dashboard**, and a view can show that dashboard above its list. You create a *Tasks by status* chart and show it to managers in *Manage Tasks*.

Create the summary

1. Open the *Tasks* entity and select the **Dashboard** tab.
2. Select **Add Summary**. The **Summary Information** form opens.
3. Fill in:
   - **Title** — *Tasks by status*, in Arabic and English.
   - **Icon** — for example *chevron-down*.
   - **Show as** — **Chart**, with **Chart type Doughnut**.
   - **Source** — **All entries**.
   - **Field** — *Title*, and **Function** — **Count**. This counts tasks.
   - **Group by** — *Status*, so there is one segment per status.
   - **Order by** — **Ascending**.
4. Select **Done**. The chart appears on the **Dashboard** tab.

Show it in "Manage Tasks"

1. On the **Views** tab, open *Manage Tasks* for editing.
2. Turn on **Dashboard** and set **Summaries to show** to **Show all summaries**.
3. Turn on **Click to browse**, so a manager can click a segment — say **In Progress** — to see the tasks behind it.
4. Optionally, under **Behavior** › **Default landing screen**, drag **Dashboard** to the top of the list, so managers see the chart first when they open the view.
5. Select **Save**.

> ### Other kinds of summary
>
> A summary can also add up, average or find the minimum or maximum of a number field, and group by a date — tasks created per week, for example. See Summaries, dashboards & counters.

Check before moving on

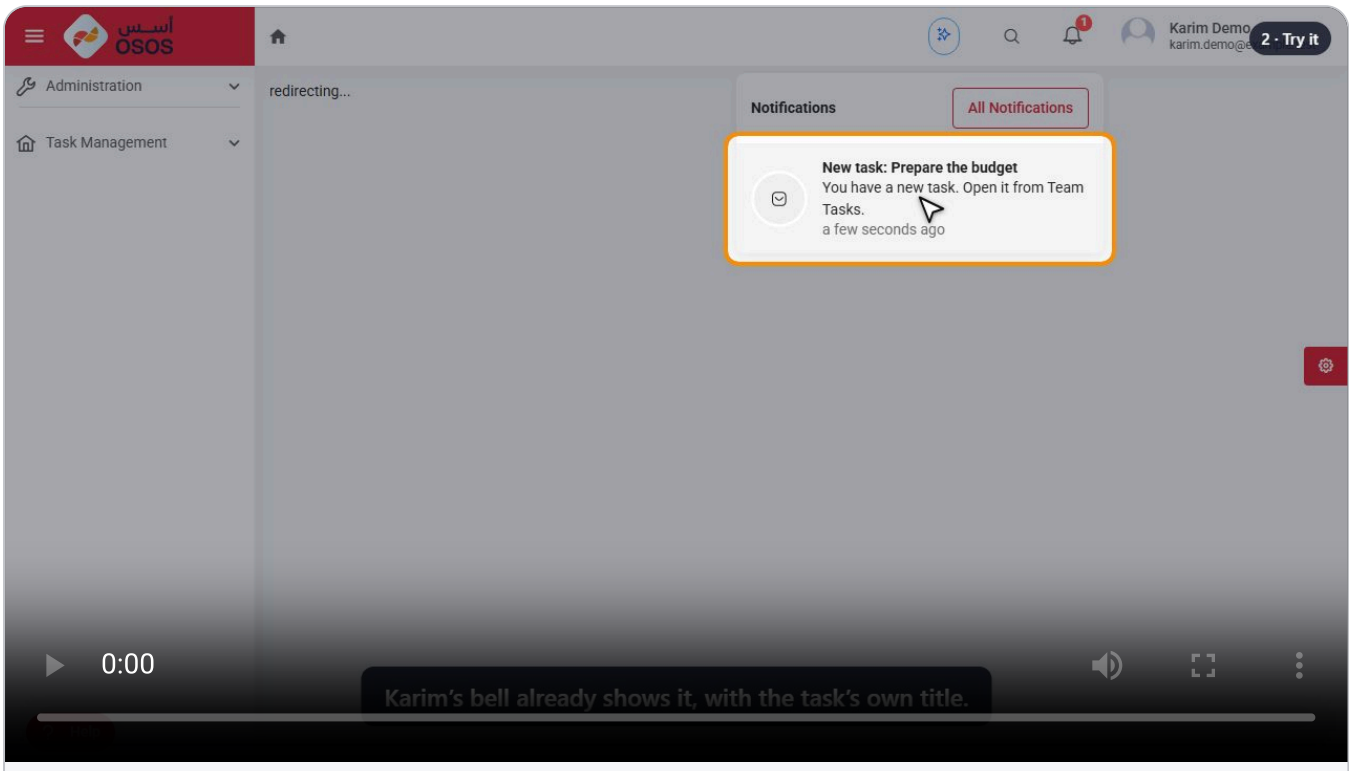
- ☐ Signed in as *Manager One*, open **Manage Tasks**: the *Tasks by status* chart shows one segment per status that has tasks.

Next step

Notify the assignee.

Notify the assignee

Step 8 of 12. A **reaction** is a rule that runs by itself: *when* something happens to a record (the **event**), *do* something (the **action**). You add two reactions to *Tasks*. When a task is created, the person in *Assigned To* gets a notification in the portal and an email.



TUTORIAL · PART 3 OF 3 Notify the assignee

A reaction tells each person when a task is assigned to them · 2:00 · Recorded on OSOS 1.11.0-alpha.1

Each reaction does one thing, so a notification and an email are two reactions on the same event.

Send an in-app notification

1. Open the *Tasks* entity and select the **Reactions** tab.
2. Select **+** (or **Add Reaction** if the list is empty). The **New Reaction** dialog opens.
3. Enter the **Name** *Notify assignee*.
4. Under **Event**, choose **On Entry Created**. Leave **Filter** empty — the reaction runs for every new task.
5. Under **Action**, set **Action Type** to **Send Notification**, and fill in:
   - **Name** — *Send notification*.
   - **Module** is filled in for you; change the **Icon** if you like.
   - **Channel Type** — **In App**.
   - **Title** — `New task: {{task}}`
   - **Content** — `You have a new task. Open it from My Tasks.`

6. Under **Custom Variables**, connect the placeholder to a field: in the row for **Variable** *task*, set **Type** to **Field** and **Map to** to *Title*. Each name you write in double braces gets a row here.
7. Under **Recipients**, set **Source** to **User**, **Type** to **Field**, and **Value** to *Assigned To*.
8. Select **Save**. The reaction appears in the list.

Use the translate icon at the end of **Title** and **Content** to enter the text in both languages.

Send an email

Add a second reaction the same way, with these differences:

Setting	Value
Name	*Email assignee*
Event	**On Entry Created**
Action Type	**Send Email**
To	**Source User**, **Type Field**, **Value** *Assigned To*
Subject	`New task: {{task}}`
Body	`You have been assigned a new task. Use the link below to open it.`
Custom Variables	*task* → **Field** → *Title*
ADD Entry Link	Turned on, so the email links straight to the task

Leave **CC**, **BCC** and attachments empty. Select **Save**.

Check before moving on

- ☐ Signed in as *Manager One*, add a task in **Manage Tasks** and assign it to *Employee One*.
- ☐ Signed in as *Employee One*, the bell in the top bar shows *New task: …*, and the email arrives with a link to the task.

If nothing arrives, open the reaction from the **Reactions** tab: the list shows whether each reaction ran. Email also needs outgoing mail working in your space — see Before you start.

What else reactions can do

Besides notifications and email, a reaction can send an SMS, create or update another record, move a workflow on, call another system, or run on a schedule. It can also react to workflow events, such as a request being approved. See Reactions & automation.

Next step

Send a weekly report.

Send a weekly report

Step 9 of 12. A **report** is a printable table built from a saved query: the query picks the rows, and the fields you choose are the columns. People can open it in the portal, export it to PDF or Excel, and receive it by email on a schedule. You build an *Open tasks report* and email it to the managers every Sunday at midnight.

Create the report

1. Open the *Tasks* entity, select the **Reports** tab, then **New Report**.
2. Under the report information, fill in:
   - **Title** — *Open tasks report*.
   - **Query** — *Open tasks* (from step 5).
   - **Icon** — any.
   - **Fields** — *Title*, *Assigned To*, *Status*, *Due Date*, in that order. The order is the column order.
3. Under **Report Design**, keep **Page header** and **Organization Header** on, turn on **Report Header**, choose **Text** and enter *Weekly open tasks*. Set the **Layout** to **Landscape**. The preview beside the form shows the result.
4. Under **Access Permission**, choose the group *Task Managers*, so only managers can open the report.

Schedule the email

Under **Sending Setting**:

1. **Timer** — set **Interval type** to **Repeat** and enter the **Cron Expression** `0 0 * * 0` (every Sunday at midnight). Leave **Start Date** and **End Date** empty to keep sending until you stop it, or set them to limit the period.
2. **Channel** — keep **Channel Type Send Email**. In **To**, choose **Group**, **Fixed**, *Task Managers*. Enter the **Subject** *Open tasks this week* and a short **Body**, for example *Here are this week's open tasks.*
3. **Sending Report Method** — tick **Send report attachment file** and choose **PDF**. Tick **Send report link** too, so managers can open the live report.
4. Select **Done**.

Publish it

On the **Reports** tab, select **Publish** on the report's card. The schedule only runs while the report is published; **Unpublish** stops it.

> ⚠ **Reports with @me are personal**
>
> Do not schedule an attached report whose query uses **@me**. The attachment is produced once for everyone on the list, so it cannot show each recipient their own tasks. That is why this report uses *Open tasks*, not *My open tasks*.

Technical details: reading a cron expression

A cron expression has five parts: minute, hour, day of the month, month, day of the week (0 is Sunday). `*` means “every”.

Expression	Runs
`0 0 * * 0`	Sundays at 00:00
`0 8 * * 1`	Mondays at 08:00
`0 8 * * 1-5`	Weekdays at 08:00
`30 17 * * 4`	Thursdays at 17:30
`0 8 1 * *`	The 1st of each month at 08:00

The schedule follows the platform server’s clock. If reports arrive at an unexpected hour, ask your operator which time zone the server uses.

Check before moving on

- ☐ The report is published, and *Manager One* can open it from the sidebar under *Task Management*.
- ☐ To test the email without waiting for Sunday, set a cron expression a few minutes ahead (for example `15 10 * * *` for 10:15 today), wait for the email, then set it back to `0 0 * * 0`.

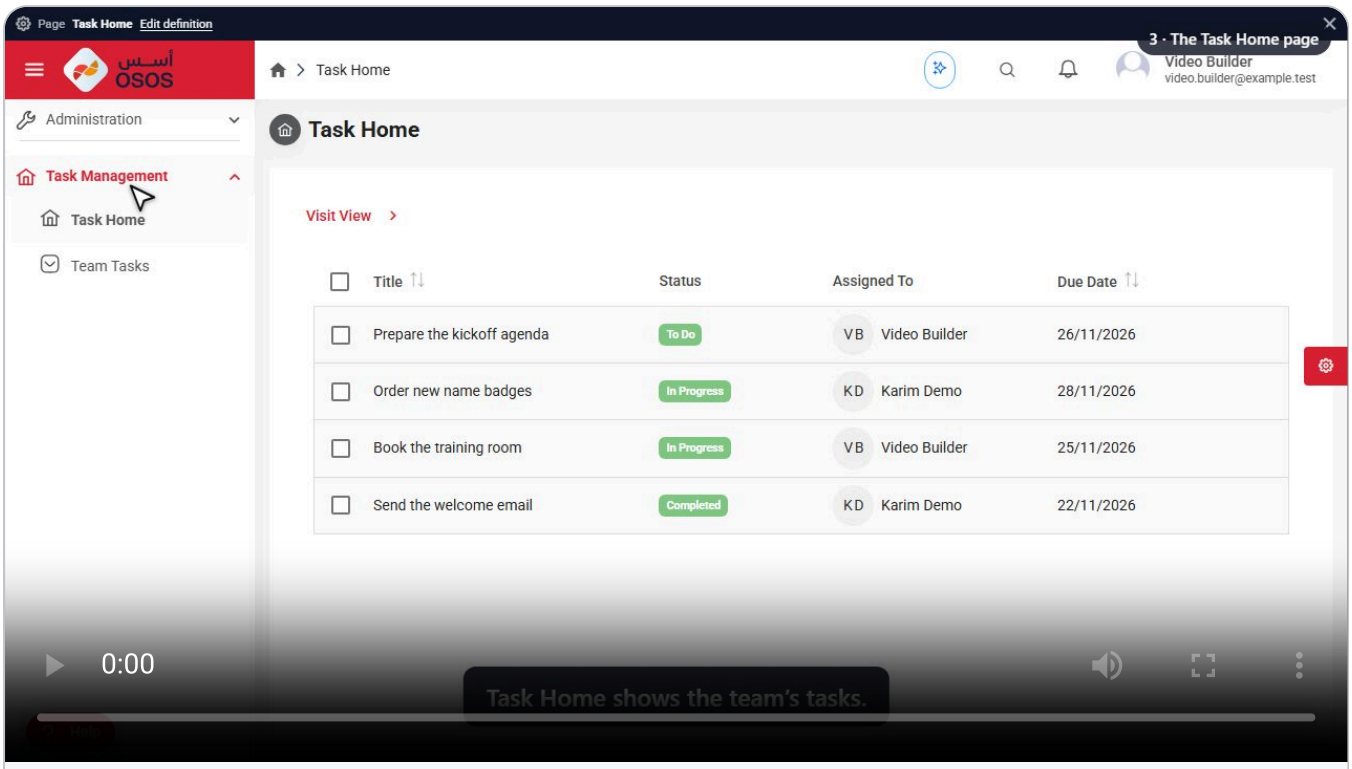
More options — one-off sends, Excel attachments, footers, column limits in PDF — are in Reports.

Next step

Build the home page.

Build the home page

Step 10 of 12. A **page** puts several things on one screen — text, links, a list from a view, a chart — using building blocks called **widgets**. You build *Team Tasks*, the page people land on when they open the app, and optionally a public visitor page.



TUTORIAL · PART 2 OF 3 The Team Tasks view and the Task Home page

A table and a Kanban board, opened to a group, on a home page · 3:06 · Recorded on OSOS 1.11.0-alpha.1

Create the page

1. Open **Administration › Applications › Modules**, open *Task Management*, and select the **Pages** tab.
2. Select **Add Page**. The page designer opens on its **Details** tab.
3. Under **Page Information**, enter the **Name** *Team Tasks* in Arabic and English, and pick an **Icon**.
4. Under **Sidebar Configurations**, turn on **Pin to Sidebar**.
5. Under **Access Permissions**, turn on **Authenticated** and tick **Default Page**, so everyone who is signed in lands on this page. Leave **Select Users** and **Select groups** empty: a page limited to particular people cannot be the default page. Each widget still shows only what the person is allowed to see.

Add the widgets

Open the **Content** tab. Add widgets from the widget list and drag them into place; each widget can take the full width or share a row.

Widget	Settings
Text	A welcome message, for example *Welcome to Team Tasks. Your open tasks are below.*
Link	Title *Open My Tasks*, pointing to the *My Tasks* view.
View	The *My Tasks* view of *Tasks*, showing 5 entries.
Summary	Module *Task Management*, entity *Tasks*, summary *Tasks by status*.

Select **Done** to save the page. A live mobile preview next to the canvas shows how the page will look on a phone.

Publish it

On the module's **Pages** tab, select **Publish** on the *Team Tasks* card. A page is not visible to anyone until it is published.

Default pages

A module can have one default page for signed-in people and one for visitors:

- **Authenticated** with **Default Page** ticked — where signed-in people land when they open the module.
- **Anonymous** with **Default Page** ticked — what visitors see before they sign in, and after they sign out.

To change the default later, open another page's **Details** tab and tick **Default Page** there, then untick **Default Page** on the page that had it. A new module starts with no pages, so the first page you add is the only candidate.

Optional: a visitor page

To give visitors a public welcome before they sign in, add a second page:

1. **Add Page**, **Name** *Welcome*.
2. Under **Access Permissions**, turn on **Anonymous** and tick **Default Page**. Leave **Authenticated** off.
3. On **Content**, add only public information — a **Text** widget with a short introduction and a **Media** widget with an image. Never put a view of tasks on an anonymous page.
4. Select **Done**, then **Publish**.

> **Note**
>
> Your platform may offer more widgets, depending on the widget packs installed — for example a *Welcome* block that greets the signed-in person by name, an image slider, or an *About us* block for your mission, vision and values. Ask your administrator if you need one you cannot find.

A page that belongs to no module — a portal-wide home page, for example — is created under **Administration › Applications › Pages** instead of a module's **Pages** tab.

More about widgets, layout and page options is in Pages & widgets.

Check before moving on

- ☐ Signed in as *Employee One*, open *Task Management*: *Team Tasks* opens, with your open tasks and the chart.

Next step

Try it as an end user.

Try it as an end user

Step 11 of 12. Before you hand the app over, use it the way your users will. Sign in with the test accounts from Before you start — a private browser window makes switching easy — and walk through a task's life: a manager creates it, the employee is notified, works on it and completes it, and the manager sees the result.

1. On the web — the full walk-through, as *Manager One* and *Employee One*.
2. On mobile — the same app in the OSOS mobile app.

When everything behaves as expected, go on to Change your app safely.

Try it on the web

This walk-through checks every piece you built, in the order your users meet them. It uses the two test accounts *Manager One* and *Employee One*. How each screen works for end users is explained in the portal guides linked at each step — this page only tells you what to check.

1. Create and assign tasks — as Manager One

1. Sign in as *Manager One*. *Team Tasks* opens as the home page.
2. In the sidebar, open *Task Management* › **Manage Tasks**. The *Tasks by status* chart is at the top.
3. Select **Add Task** and fill in:
    - *Title* — *Prepare the monthly budget*
    - *Description* — *Use last month's figures.*
    - *Status* — **To Do**
    - *Assigned To* — *Employee One*
    - *Due Date* — any date next week
4. Save the task. Add two more: one for *Employee One* with status **In Progress**, one for *Employee Two* with status **To Do**.

Check: the chart now shows **To Do** and **In Progress** segments. Adding and editing records is described for end users in Create and edit records [Using OSOS › Create and Edit Records].

2. Get notified and work on the task — as Employee One

1. Sign in as *Employee One*. The bell in the top bar shows *New task: Prepare the monthly budget*, and the email has arrived with a link to the task.
2. Open *Task Management* › **My Tasks**. Only *Employee One*'s two tasks are listed — not *Employee Two*'s.
3. Switch the list to **Kanban view**: the tasks sit in the **To Do** and **In Progress** columns. Switching displays is described in Browse a list and change how it looks [Using OSOS › Browse a List and Change How It Looks].

3. Change a task — as Employee One

1. Open *Prepare the monthly budget* and edit it.
2. Change *Status* to **In Progress** and save. Only *Status* can be changed in this view — that is how you set up *My Tasks*.
3. Edit it again, set *Status* to **Completed**, and save.

Check: the task disappears from **My Tasks**, because *My open tasks* leaves out completed tasks.

4. Follow up and archive — as Manager One

1. Sign in as *Manager One* and open **Manage Tasks**. The chart now has a **Completed** segment. Select it: the list shows the completed task.
2. Open the completed task, open the actions menu in its header, and select **Archive**. Confirm.

5. See archived tasks — as Manager One

1. In **Manage Tasks**, open the **Status** menu next to the search box and select **Archived**. The archived task is listed.
2. Select **Active** to go back.

Archiving and restoring are described for end users in Archive and restore records [Using OSOS › Archive and Restore Records].

You can also work from Administration

Builders can see every task on the entity's first tab in Administration (**Administration › Applications › Modules ›** *Task Management* **›** *Tasks*), add test tasks there, and switch the list between table, cards, calendar and board. Use it for testing; your users work in the views.

If something is missing

What you see	What to check
Task Management is not in the sidebar	The views are not published, or **Pin to sidebar** is off (step 5).
Employee One sees no tasks in **My Tasks**	The tasks are assigned to someone else, or their status is **Completed**.
No notification or email	See step 8.
Archived is not in the **Status** menu	**Archived entries** is off in *Manage Tasks*.

Next

Try it on mobile, or go straight to Change your app safely.

Try it on mobile

Your app works in the OSOS mobile app without any extra building: the same module, views, pages and notifications appear there. This page checks Team Tasks on a phone, as *Employee One*.

Install the app and sign in

1. Install the mobile app from the App Store or Google Play, open it, and choose your language.
2. Enter your organization's address, then sign in as *Employee One*.

The full first-run steps, including guest access and password reset, are in Get started with the mobile app [Using OSOS › Web vs Mobile].

Check Team Tasks on the phone

1. **Home** opens on *Team Tasks*, the default page you built in step 10.
2. Open **Sidebar** and select *Task Management* › **My Tasks**. Only *Employee One*'s open tasks are listed.
3. Change how the list looks — for example to the **Kanban view** — from the list's settings, next to the search box.
4. Search for a task by name, or filter and sort the list from the same settings.
5. Open a task, edit it, and change *Status* to **In Progress**.
6. Ask *Manager One* to assign you a new task from the web. The notification appears under **Notifications** in the bottom bar (and as a push notification, if you allowed them).

How end users find their way around the app is described in Find your way around the app [Using OSOS › Web vs Mobile].

What differs from the web

The mobile app covers the everyday work of your users — lists, records, workflow steps, comments, dashboards and notifications. A few things are web-only, and two of them matter for Team Tasks:

- **Reports** do not open on mobile. Managers read the weekly *Open tasks report* on the web or as the emailed attachment.
- **Building** — everything you did in steps 3 to 10 — is done on the web only.

The full comparison is in Web vs Mobile [Using OSOS].

Next step

Change your app safely.

Change your app safely

Step 12 of 12. Apps change after people start using them. This step teaches the one rule that keeps changes safe, then makes three realistic changes to Team Tasks:

1. Add a required *Priority* to every task — a **breaking** change, and how to repair existing tasks afterwards.
2. Let a task have sub-tasks, shown as a tree and as a *Subtasks* tab.
3. Add a counter that shows each person how many tasks are waiting.

The rule: safe and breaking changes

Every task remembers the version of the entity it was saved with. Most changes leave existing tasks alone. A **breaking** change — one existing tasks may not satisfy — creates a new version of the entity, and tasks saved before it become **Obsolete** until someone upgrades them.

Safe	Breaking
Adding an optional field	Adding a **required** field, or making a field required
Renaming a field, changing labels and layout	Changing a field's **type**
Deleting a field	Turning a field's **repeatable** or **multiple** option on or off
	Changing the **workflow** steps

When you save a breaking change on an entity that already has records, the **Confirm Structure Changes** dialog lists each affected field and why, so you can still back out. The full list is in Editing an entity.

1. Add a required Priority

1. Open the *Tasks* designer (**Edit Entity**) and the **Structure** tab.
2. Drag a **Category** field after *Status*, name it *Priority*, and add the options **Low**, **Normal** and **High**. Tick **Required** and set **Default Value** to **Normal**.
3. Select **Save**. The **Confirm Structure Changes** dialog lists *Priority* as an added required field. Select **Save**.

New tasks get *Priority* **Normal** unless someone picks another. The tasks that already existed are now **Obsolete**: they have no priority yet. Upgrade them:

4. On the entity's **Tasks** tab, filter the list to **Obsolete entries**.
5. Select all the listed tasks, then **Upgrade (Entries N)** from the selection menu.
6. Under **Required changes**, set *Priority* to **Default** — every task gets **Normal**.
7. Select **Upgrade**. The tasks are current again.

To upgrade a single task instead, open it and select **Upgrade Entry**.

> **Tip**
>
> Make breaking changes in one go. Each breaking save is a new version, and each version is another round of upgrades.

2. Add sub-tasks

Add a Parent Task field

1. In the *Tasks* designer, drag a **Reference** field after *Description* and name it *Parent Task*.
2. In its data source settings, choose the entity *Tasks* itself, so a task can point at another task.
3. Leave **Required** off — top-level tasks have no parent. Because the field is optional, this change is safe and no task becomes obsolete.
4. Select **Save**.

Show tasks as a tree

1. Open *Manage Tasks* for editing (the entity's **Views** tab).
2. Add *Parent Task* to the **Table view** fields.
3. Under **Displays**, turn on **Tree view** and set:
    - **Tree reference** — *Parent Task*
    - **Main title** — *Title*
    - **Sub title** — *Description*
4. Tick **Default** next to **Tree view** if managers should see the tree first.
5. Select **Save**.

Add a Subtasks tab

A **relation** shows the link from the other side: on a parent task, a *Subtasks* tab lists every task whose *Parent Task* points at it.

1. Open the *Tasks* entity and select its **Relations** tab.
2. Select **Add Relation**. Enter the name *Subtasks* and pick an icon.
3. In **Relation Type**, choose **Incoming**, then choose the entity *Tasks* and the field *Parent Task*.
4. Save.
5. Open *Manage Tasks* for editing, turn on **Relations**, choose *Subtasks*, and select the view to show them in (*Manage Tasks*). Select **Save**.

Check: create a task, then a second task with *Parent Task* set to the first. Open the first task: a *Subtasks* tab lists the second. The tree view shows the second task under the first.

More about reference fields and relations is in Link entities.

3. Add a counter

A **counter** is a badge in the top bar — *3 tasks* — that opens the matching records when selected.

1. Open *My Tasks* for editing and turn on **Counter view** under **Displays**, with **Main title** *Title*, **Sub title** *Description* and **Date** *Due Date*. A counter can only count a view that has this display. Select **Save**.
2. Open **Administration › Applications › Counters** and select **New Counter**.
3. Enter the **Name** *My tasks* and pick an icon.
4. In **View**, choose *My Tasks*. Because *My Tasks* already shows only your open tasks, leave **Counting Criteria** empty.

5. Save. Each employee now sees their own count of open tasks in the top bar.

See Counters and record numbers.

> **Looking for the query steps?**
>
> Earlier versions of this walkthrough created a query here. Queries are now created before the views that use them, in step 5: Create the queries.

You have built your first app

You now know the whole cycle: model the data, filter it, show it, automate it, report on it, put it on a page, and change it without breaking it. Where to go next:

- Add an approval step: Publish a service with an approval workflow.
- Build a larger system: Customer support tutorial.
- Move the app to another environment: Packaging [Integrate & Extend › Packaging & distribution].

Builder concepts and glossary

This page is for **builders** who want the vocabulary of OSOS in one place. Everything you build without code is assembled from a small set of building blocks that fit together in layers: containers that separate data, a data model that describes it, tools that work with it, processes that move it, surfaces that present it, and an identity model that controls who sees what. Each term below says what it means in plain words and links to the guide that shows you how to use it.

How the pieces fit together

The concepts stack in layers. A **space** isolates a tenant's data; inside it, **modules** group **entities** (tables), and each entity is described by **fields** and filled with **entries** (rows). **Views**, **queries**, and **pages** let people read and work with those entries; **workflows** and **reactions** move and automate them; **dashboards**, **reports**, and **timelines** present and analyse them. Cutting across all of it, an identity model of **users**, **groups**, **roles**, and **permissions** decides who can do what.

Layer	Concepts
Containers and isolation	Spaces, modules, stores, sharing levels
Data model	Entities, fields, entries, relations
Working with data	Views, queries, lists
Process and automation	Workflows, reactions, buttons, services, time monitoring
Presentation and analytics	Pages, dashboards, counters, reports, templates, timeline
Identity and access	Users, roles, organizational structure, groups, permissions, delegation

Containers and isolation

Spaces

A **space** [Administration › Spaces & domains] is the platform's multi-tenancy isolation boundary: it scopes data-entities, modules, groups, views, and the users who may access them. Every persisted record is stamped with its space, and reads and writes are automatically filtered by it, so a user cannot reach a space they don't belong to. A well-known **Root Space** holds system-level records. Spaces pair with **domains** — a space answers "which data does this request see?", while a domain answers "which login experience, branding, and identity providers apply?"; a working tenant needs both covering the same hosts.

Modules

A **module** is a self-contained container that groups related entities and functions into one unit — for example a task-management module. Modules are independent and do not depend directly on one another, though they can interact through virtual entities. The system provides a **Core module** automatically for the entities it needs to run; users cannot add entities to it. A module's **normalized name** (English, standardized) is fixed at creation and drives export and search.

Stores and sharing levels

Data stores are the units that actually hold an entity's entries, categorized by sharing scope. A **public** and a **private** store are created automatically with every entity, and a **shared** store can be defined for a chosen set of spaces; a

weight field sets which store takes priority for new inputs. The **sharing level** of a module or application (Public, Private, or Shared) controls how widely it is visible across spaces, mirroring the same three scopes.

The data model

Entities

An **entity** is the primary element for storing data — analogous to a table in a database. Besides **normal** entities, the platform offers **virtual entities** (Proxy, Joiner, and Aggregator), which are read-only and computed from a source entity rather than entered manually. See Virtual Entities. An entity's **normalized field** and **type** are fixed at creation, it moves through **Draft** → **Active** → **Archived**, and a change that existing records may not satisfy — a new required field, a type change, a workflow change — creates a new **entity version** (see Change an entity safely).

Fields

Fields are the units that hold data in each entry. Every field rests on one of about 15 storage primitives (text, number, date, boolean, entity reference, JSON, and so on), and the designer picks from roughly 35 UI field types on top of them. Field properties — required, unique, default value, repeatable, visibility, locked, multiple, and **sensitive** — shape how data is entered and protected. Special kinds include the **block** (a group of fields under one frame), the **reference** field (a link to another entry), and the **category** field (a colored pick-list of statuses).

Entries

Entries [Using OSOS › Create and Edit Records] are the actual data rows users create and act on. An entry moves through **Draft** → **Active** → **Archived** and becomes **Obsolete** when the entity's structure changes in a way the entry no longer satisfies, until it is upgraded. Draft entries skip full validation, fire no reactions, and are visible only to their creator until published. Each entry carries **metadata** (who and when), a **discussion** thread of comments, and a **history** log of every change.

Relations

Relations connect a parent entity to a child entity through a **reference field**, letting the parent contain many child entries and enabling nested, hierarchical structures. A reference remembers *which record* was picked, not its text, so it stays correct even if the target's title later changes. Relations are either **outgoing** (this entity points out) or **incoming** (another entity points here).

Working with data

Views

A **view** is the main point of interaction between a user and a single entity — listing, displaying, adding, and editing entries within the permissions the administrator sets. Views come in three access types (**Authenticated**, **Anonymous**, **Restricted**) and support seven display formats, called **faces** (table, card, calendar, Kanban, tree, timeline, badge), plus the counter view; a view can enable several faces and let users switch between them. Views also offer filter-first loading, quick filters, saved personal queries, and signed **share links** for public access. A view's entries can be exported as an **XLSX** file, or selected entries exported through a template; sensitive-field permissions are respected in the output.

Queries

Queries retrieve entries matching defined criteria. Each condition pairs a **field**, an **operator**, and a **value**; conditions join with AND/OR logic and nest into groups. Values can be fixed, dynamic (from another field), a context **pointer** (like `@me` or `@today`), an expression, or a later-supplied **parameter**. Queries can drill through reference fields into related entities, filter on workflow metadata, and be saved for reuse in views, reports, and lists. A **statistic query** aggregates entries into a number, chart, or list — the building block of dashboard summaries.

Lists

A **list** [Using OSOS › Lists: Keep Records Together] is a collection of values sharing one property, either **Flat** or **Tree** in shape. Its members come from a **Store** (an entity's entries), another **Collection**, or a **Query** — so a list can hold fixed custom values or draw live from the system's data. Lists differ from **groups**: a group organizes *users* for permissions, while a list organizes *values* for reuse.

Process and automation

Workflows

A **workflow** describes the ordered sequence of steps an entry follows to complete a task. Every workflow runs from an initial step to a final step, with your own steps in between (and, at run time, extra ad-hoc steps an owner may add); their order is what "next" and "previous" mean. Each step has an **owner** — a person, a group, a list, someone named in the record, or the manager of an earlier step's owner — with fallback owners in case the first cannot be found. Actions carry **conditional routing** to targets, and authorized users can steer entries manually (move, return, reassign, add ad-hoc, reset, close). Steps can also be bound to a **time limit** (SLA); see Time monitoring.

Reactions and automation

A **reaction** automates work on entries and follows an **Event** → **Condition** → **Action** model. Events include entry created/modified/archived/restored, reaching or leaving a workflow step, an action executed, an SLA breach, a timer, and custom or external events. More than twenty action types are available — entry and block-item operations, workflow moves, user activation, notifications/email/SMS, API calls, custom functions, and file generation from a template. Execution controls cover disabling, executor identity (system or manager), loop protection, space scope, and reaction chaining.

Buttons

Custom **buttons** are interactive elements on an entity that, when clicked, **fire an action/reaction** and/or **open a pre-filled create form**. A button's behavior is set by its action type (Custom Event, Interface Event with an "Ask User" dialog, or Navigation) and its scope (single entry, multiple entries, or a data collection). Each seeded field is filled by a value mapping — dynamic, manual, not-set, pointer, or expression — and a button's visibility can be restricted to entries matching a saved query or filter.

Services and requests

At its core a **service** [Using OSOS › Services & Submitting Requests] is a published request template; submitting it creates a **request** — a workflow-driven record that routes to owners for action. Services are surfaced in the **Service Catalog** and can target authenticated or anonymous audiences (pair public forms with human verification). Users track work through the **Inbox** (requests awaiting their action), **My Requests** (their outgoing requests), and **Request Monitoring** (an observer view).

A request shows people a status — **Draft**, **Open**, **Completed** or **OnHold** — and, once finished, a **result**: Succeeded, Failed, Rejected, Timedout, Cancelled or Obsolete.

Time monitoring

Time monitoring [Administration › Working hours & holidays] binds a workflow step to a service-level agreement. The limit can be fixed, read from a field, or computed by an expression, measured in seconds/minutes/hours, and turned into a due date against the official **working hours** and holidays. Each monitored step tracks a compliance status automatically — **Pending**, **Achieved**, or **Overdue** — and an overdue step raises an event that can trigger a reaction (an alert or escalation).

Presentation and analytics

Pages

A **page** (internally a *Generic View*) is a composable, entity-agnostic surface built by drag-and-drop: you place **widgets** into groups on a 12-column grid to design landing pages, dashboards, and portals that pull from many entities at once. The widget library includes data-view, statistic-query, timeline, text/heading, link and link-list, navigation, and media/document widgets, each with its own visibility and permission settings. Pages support redirection rules and can be exported as a **PDF** (portrait or landscape).

Dashboards and counters

A **dashboard** presents a set of cards (summaries) organized into groups. Each card is built on a **statistic query** and rendered as a **Number**, **List**, or **Chart** (column, bar, line, pie, or doughnut), with drill-down, enlarge, auto-refresh, and time filters. Dashboards exist at entity, module, or system level. A **counter widget** is a live count of the records in a view, optionally narrowed by a saved query, shown as a badge in the top bar. It is different from a **sequence field**, which gives each new record a running number.

Reports

A **report** is built from an entity, with the administrator choosing the fields and header/footer/body sections. Reports render in **Portrait or Landscape**, export as **XLSX or PDF** through the templates engine, and can be **scheduled** and emailed to recipients as a link or attachment. Report output respects sensitive-field masking. Reports are a **web-only** feature — they are not available in the mobile app, where report links are treated as unsupported.

Templates

Templates organize how entries are laid out in exported files. Supported formats are **DOCX, HTML, and XLSX**, with conversion from DOCX/HTML to **PDF**. Values are filled through tokens and expressions (JavaScript and custom functions allowed) resolved against the entry at export time, and a template sets its export level (single or multiple entries) and run mode (foreground UI or queued background). Templates power three things: a view's export, the "Generate File From Template" reaction action, and report generation. Sensitive fields the user cannot access appear masked in the output.

Timeline

The **timeline** displays **posts** in chronological order. Posts are built automatically from the entries of entities where timeline contribution is enabled, and are archived or deleted when their source entry is. Where permitted, you can publish/unpublish, schedule, and pin posts, and a workflow can drive publishing. The timeline adds a social layer — emoji **reactions** (one per user) and **comments** with @mentions — which are distinct from automation reactions.

Identity and access

Users, roles, and organizational structure

A **user** [Administration › Managing users] is a person with a profile and an account. Signing in — including two-factor verification, passkeys and single sign-on with your organization's identity provider — is handled by the platform's sign-in service, separately from what the user may do once inside. A **role** [Administration › Managing roles] expresses a job position and is bound to a user through an **Assignment** record; every role has a virtual role-group of its holders. The **organizational structure** [Administration › Organization units] arranges **organizational units** into a tree, each with a manager (updated automatically from assignments) and members, and drives manager-based permissions and relative step ownership.

Groups

A **group** [Administration › Managing groups] is the **primary unit permissions are granted to** — assign permissions to a group and members inherit them. Groups come in several types: **regular**, **role group** and **organizational-unit group** (virtual, created automatically), **query group** (dynamic membership from conditions), and **virtual** (system-managed). When a user's own grants and their group grants differ, the **highest** effective level applies.

Permissions and sensitive fields

Permissions [Administration › Permissions & access] are an ordered ladder — **None** → **Restricted** → **View** → **Developer** → **Edit** → **Admin** — applied at several scopes (system, module, entity, view, page, or report). The highest grant wins, except an explicit **None**, which is a hard deny; **Developer** is a builder/designer sibling of Edit, not a super-set, and viewing an entity's entries list needs an Edit-tier grant. **Access permissions** (authenticated / anonymous / restricted) on views, reports, and services work alongside the ladder. Beyond it, **sensitive fields** mask a field's value for anyone without an explicit grant — a separate dimension that **not even an admin bypasses**.

Delegation

Delegation [Administration] lets one user act on behalf of another for an authorized window while keeping an audit trail of who really acted. A delegate can view the delegator's data, create and edit entries, add comments, and execute workflow actions; the system records each action in the delegate's name **on behalf of** the delegator, and delegation identifiers travel with every workflow and comment request so the backend evaluates permissions accordingly.

Cross-cutting concepts

Identifiers and normalized names

Every created object receives a unique, random **identifier** used to distinguish objects and to link tables and build relationships; it lives at the entry level and is not preserved on import. The **normalized field** is a unique, standardized name (English) generated by the system — or set by you, as long as it stays unique — that *is* exported and preserved on import, serving as an object's primary reference.

Pointers and expressions

Pointers are context variables that resolve against the entry — `@me` (the current user), `@myOrgUnit` (the current user's organizational unit), and date pointers such as `@now`, `@today`, `@StartofDay`, `@StartofWeek`, `@StartofMonth`, and `@StartofYear` — commonly used in events, queries, and filters. **Programming expressions** are an optional JavaScript environment for customizing functions not exposed through the interface; they are never required to build an

application, but add flexibility for users with programming expertise or those who need functions the interface does not expose.

References

Reference fields link one entry to one or more entries in another entity, letting you model relationships across your data. See Relations.

Core entities

Core entities are structural entities the platform creates automatically to manage its own operations — for example the Posts, Users, and Groups entities, plus others such as comments, roles, services, reports, and notifications. Most cannot be modified directly, though administrators can extend some of them with additional fields, properties, reports, and queries. A few are worth knowing in more depth:

- **Posts entity.** Holds all posts added to the system, displayed as a timeline and used for system-wide statistics and reports. Entries are added dynamically through external display pages with timeline views. Key fields include title, description, publication status, timeline settings, and interactions.
- **Users entity.** Holds all users in the system. An administrator adds a user, which triggers an email to that user to create their password. Fields cover basic information (first name, last name, email, organizational affiliation), account information (username and role assignments), and the user's direct manager.
- **Groups entity.** Lists all system groups, including user groups and those created by organizational units or roles. It is used when setting permissions and generating reports. Fields include group name, description, symbol, users, and group type.

Basic lists

Basic lists are default value sets reused across the system. Some are essential to operations, such as the service-category list required to create services.

Attachments

Attachments are files uploaded as part of an entry's details, including within its discussion thread. The permitted file types and sizes are configurable — per file field, and per view for discussion attachments — up to 20 MB per file by default, rather than a fixed set.

Notifications, settings, and links

Notifications [Using OSOS › Receiving Notifications] are event-driven alerts delivered over four channels — in-app, push, email, and SMS — with template-and-token content. **Settings** [Administration › Organization settings] tune the platform at space, organization, and user levels (themes, fonts, languages, timezone, formats); OSOS is bilingual by design, storing every name in both Arabic and English. **Links** come in two kinds: sidebar links that organize navigation, and tokenized **System Links** that share a view, page, report, or module through an anonymous or authenticated shareable URL.

Not yet implemented

Gamification and HR-profile features — points, badges, achievements, and skills — exist only as an early idea. They are **not** part of the product, so do not plan an application around them.

Related guides

- Data Modeling: Entities, Field Types, Relations, and Virtual Entities

- Data Views, Queries, and Pages & Widgets
- Workflows, Reactions & Automation, and Buttons
- Dashboards & Summaries, Counters, Reports, Templates, and Timeline
- Build apps — start here for where the builder lives and a recommended learning path

Create and design an entity

This guide is for **builders** who decide what an application stores. By the end you will be able to create an entity, design the fields and form of its records, choose how its records are titled, speed up large lists with indexes, organise its quick filters, export its records, move it to another module, and archive or delete it. Changing an entity that already holds records is covered in its own how-to, Change an entity safely.

How entities work

An **entity** is where your application keeps one kind of record — for example *Asset Requests*, *Employees* or *Invoices*. You decide which **fields** each record has (*Asset type*, *Reason*, *Needed by*), how the form looks, and who can see it. Each record you or your users add is an **entry** of that entity.

- Every entity lives inside a module, the container for one application.
- Entities can point at each other: an *Asset Request* can refer to the *Employee* who asked for it. See Link entities.
- Besides ordinary entities you can build **virtual entities**, whose records are calculated from other entities. See Virtual entities.
- Each entity has a **name** people see, which you can change at any time, and a **normalized name** — a short English machine name that queries, reactions and packages use to find it. The normalized name is fixed once the entity is created.

Create an entity

1. Open **Administration › Applications › Data Entities**, or open your module and go to its **Entities** tab.
2. Select **Add Data Entity**.
3. Enter a **Name** in Arabic and English. It must be unique within the module.
4. Check the **Normalized name**. It is generated from the English name; you can change it now, but not after the entity is created. Use English letters and avoid programming words such as *new*, *class*, *default*, *if* or *return*, which are not allowed.
5. Choose an icon with **Select Icon**.
6. In **Choose Module**, pick the module. When you start from a module's **Entities** tab, it is filled in for you.
7. In **Data Entity Type**, keep **Normal Entity**. (The options under **Virtual Entity** create a virtual entity.)
8. Under **Field Settings**, enter the **Single Name** and **Plural Name** — for example *Request* and *Requests*. The product uses them in buttons and messages such as *Add Request*.
9. Optionally, under **Administration Access**, choose the users and groups who may manage this entity.
10. Confirm. The entity opens, ready for you to design its structure.

Design the structure

1. Open the entity's designer: select the **three dots** on the entity card, then **Edit Entity**. (Opening the designer requires **Developer** access to the module.)
2. On the **Structure** tab, drag a field type from the palette onto the form. The field types guide explains what each type is for.
3. Select the field to open its settings and configure it (below).

4. Repeat for every field, then select **Save** at the top of the designer.

To stop colleagues from changing the entity while you work on it, select **Lock Entity**; select **Unlock Entity** when you are done.

Configure a field

Each field has its own settings panel. Every field has a **name** (Arabic and English) and a **normalized name**, unique within the entity. The other settings depend on the field type — for example whether the field is required, unique, repeatable or allows several values, its default value, and its validation rules. A field with many settings has a section list and a search box at the top of its panel: type part of a setting's name to jump straight to it.

Group related fields in a **block** — for example an *Address* block with *Street*, *City* and *Postcode*. Make the block repeatable when a record can hold several of them, such as several addresses.

Choose the title of each record

Records are shown in lists, cards and pickers by their **Main title** — usually the name or reference number — with an optional **Sub title** under it. Choose a field that identifies the record. Some rules apply:

- The main title cannot be a field that holds several values or repeats.
- The main title cannot be a sensitive or encrypted field, because titles are shown to everyone who can see the record. A sensitive field can be the sub title; it is masked there.
- If the main title is a reference to another entity, that entity's own title cannot itself be a reference.

You can also give fields other display roles, such as the picture shown on cards, so card and calendar views show the right field without extra layout work.

Lay out the form

In **Form Settings** on the **Structure** tab, choose how many fields the form shows per row (one to four) and adjust grouping and order. The same layout is used when people add and open records.

Other tabs in the designer

Tab	Use it to
Workflows	Add the approval process a record goes through. See Workflows & approvals.
Timeline	Decide whether and how the entity's records post to the timeline.
RTC	Let several people edit the same record together.
Indexes	Speed up large lists, or make a combination of fields unique. See below.

Reactions, views, queries, buttons, reports and templates are managed from the entity's own page, each on its own tab.

Find your way around an entity's page

When you open an entity, its page has tabs for everything built on it — **Entries**, **Queries**, **Views**, **Reactions**, **Buttons**, **Lists**, **Dashboard**, **Reports**, **Relations**, **Templates**, **Stores**, **Permissions** and more. The tab you are on is part of the page address, so you can **bookmark** a tab, **share** it with a colleague, **refresh** without losing your place, and use the browser's **Back** button to return to the previous tab.

Speed up large lists with an index `Since 1.10`

The platform indexes the fields it needs automatically. When a large entity is filtered or sorted all the time by fields those automatic indexes do not cover, add a custom index. You can also use an index to make a combination of fields unique — for example *one booking per room per day*.

1. Open **Edit Entity** and select the **Indexes** tab. It lists the **System indexes (managed automatically)**, which you cannot change, and your **Custom indexes** — up to 10.
2. Select **Add index** and enter a **Name**.
3. Select **Add field** for each field to index (one to eight), and choose **Ascending** or **Descending**. Put the field you filter on most first; use **Move field up** and **Move field down** to reorder.
4. Optionally tick **Unique** and fill **Partial filter expression** (see the recipe below).
5. Select **Save** at the top of the designer. The index shows **Building…**, then **Ready**. If it shows **Failed**, read the message — usually existing records already break the uniqueness rule — fix the data and select **Retry**.

Recipe: make an optional field unique. A unique index must ignore records where the field is empty, otherwise the second record left empty is refused. For a text field with the normalized name *code*, tick **Unique** and type this into **Partial filter expression (optional, MongoDB JSON)**:

```
{ "value.code": { "$type": "string" } }
```

The **Indexes** tab is not offered for core-module, request or virtual entities, and it is read-only on an entity installed from a package.

Group quick filters into tag groups `Since 1.10`

Saved queries marked as tags appear to people as one-click filters. When an entity has many of them, put them in groups — for example *By status* and *By region*:

1. Open the entity's **Queries** tab and select the **Tag groups** tab inside it.
2. Select **New tag group**, enter the **Tag group name** and add the tags that belong to it. A tag can be in only one group.
3. When you create or edit a tag query, you can also pick its group directly, or create one with **New group**.

Group names must be unique within the entity. Tag groups travel with the entity when you package it.

Export an entity's records `Since 1.10`

1. Open the entity and, from its header menu, select **Export Entries**.
2. Choose **Excel (XLSX)** or **CSV**.
3. The file has one column per field (blocks are split into their fields) and contains the records the entity's admin list shows. Above the export limit (10,000 records by default) the file is prepared in the background and sent to you when ready.

This quick export and the quick export from a view are the only exports that offer CSV. Exports through a document template produce whatever the template defines.

Move an entity to another module `Since 1.10`

Only a **system administrator** can move entities.

1. Select the **three dots** on the entity card (or open the entity's header menu) and select **Move to module**.
2. In **Target module**, choose the module. You can create a new one from the list.
3. Select **Move**.

To move several at once, open the module and select **Move entities** from its header menu, select up to 50 entities, choose the target module and confirm.

What moves with the entity: its views, lists, sensitive-field settings and sidebar links. What does **not**: summaries and widgets on the **old** module's pages and dashboard that show this entity are removed — the move dialog warns you. Rebuild them in the new module afterwards. Each moved entity keeps the permissions granted directly on it and gains the permissions of its new module.

Core-module entities, entities installed from a package and archived entities cannot be moved.

Archive or delete an entity

Archive an entity you no longer use: select the **three dots** on its card, select **Archive** and confirm. Archiving keeps the entity and its records, so you can bring it back later.

Delete an entity permanently `Since 1.10`

Permanent delete removes the entity and everything it owns — its records, views, pages, dashboard, reports, buttons, functions and grants — and **cannot be undone**. Only a **system administrator** can do it.

1. Select the **three dots** on the entity card (or open its header menu) and select **Delete**.
2. The **Delete entity** dialog lists everything elsewhere that still points at this entity — reference fields, mapped fields, virtual entities, reactions, views, pages and so on. Select **Open** on each, remove the link, and refresh the dialog. **Delete** stays disabled until nothing is left.
3. Tick **I understand that this entity and all its entries will be permanently deleted.** and select **Delete**.

If the delete is interrupted, the entity stays archived and marked for deletion; select **Delete** again to finish. To delete a whole module with its entities, use the module's own delete — see Modules.

Only ordinary, request and virtual entities can be deleted this way — not the platform's own entities, core-module entities or entities installed from a package.

Good to know

- **Quick edit follows the same rules.** When someone changes one field straight from a list, the entity's own validation rules — such as *only one manager per unit* — still apply, and the rule's own message is shown if it fails.
- **Deleting a field cleans up after itself.** Views that showed the field drop it when they are next saved, and indexes built on it are removed.
- **A view controls what a document template sees.** A template exported from a view prints blank for any field the view does not show. Make sure the view displays every field the template prints.
- **Records of virtual entities follow their source.** Permanently deleting records also removes the copies that virtual entities made of them.
- **Two people editing at once.** If two people save the same record at the same moment, the second is told the record changed and must reload it.

Troubleshooting

Problem	What to check
The normalized name is refused	It is already used by another entity, or it is a reserved programming word.
You cannot choose a field as the main title	It holds several values, repeats, or is sensitive or encrypted.
Saving the structure shows **Confirm Structure Changes**	Your change affects existing records. See Change an entity safely.
An index shows **Failed**	Existing records break the index's uniqueness rule. Fix them and select **Retry**.
Delete stays disabled	Something outside the entity still points at it. Open and remove each link the dialog lists.
Move to module or **Delete** is missing	Only system administrators see them, and not on core-module or package-installed entities.

Go deeper

For integrators: Entity model and API reference [Integrate & Extend] — the stored model, value shapes, entity types, lifecycle endpoints and refusals, versioning, indexes, tag groups and export routes

Modules

This guide is for **builders**. A **module** is the container for one application — *Asset Management*, *Human Resources*, *Customer Support*. By the end you will be able to create a module, decide who may use it and which spaces see it, find your way around it, and delete it when it is no longer needed.

How modules work

A module groups the entities of one application, and through them their views, pages and dashboards. It is what people see as an application in the sidebar, and the unit you package to move an application to another environment.

- Each module gets its own **dashboard** and **landing page** when you create it.
- Modules are independent of each other. When one application needs another's data, use a proxy entity rather than linking the two directly.
- The platform has a **core module** for its own entities. You cannot change it or add entities to it.
- Like entities, a module has a **name** you can change at any time and a **normalized name** — a short English machine name — that is fixed once the module is created.

Create a module

1. Open **Administration › Applications › Modules**.
2. Select **Add Module**.
3. Enter a **Name** in Arabic and English.
4. Check the **Normalized name**, generated from the English name. You can change it now, but not after the module is created.
5. Under **Sharing**, choose the **Sharing level**: **Public** (every space sees the module), **Private** or **Shared** (only the spaces you choose see it), and pick the spaces where needed.
6. Choose an **Icon** and, optionally, add a **Description**.
7. Under **Administration Access**, choose the users and groups who may work on the module. You can change module permissions later on its **Permissions** tab — see Permissions [Administration › Permissions & access].
8. Confirm, and check for the success message.

Creating a module requires system **Developer** access.

Find your way around a module

Open **Administration › Applications › Modules** and select a module. Its page has four tabs:

Tab	What you find there
Dashboard	Summaries and charts for the module's entities. See Dashboards & summaries.
Entities	The module's entities. Add, open, move, archive or delete them here.
Pages	The module's pages, including its landing page. See Pages & widgets.
Permissions	Who may view and edit the module. See Permissions [Administration › Permissions & access].

The **Modules** list, the system **Data Entities** list and each module's **Entities** tab have a **Sort** control: **Newest created** (the default), **Last modified** or **Name A–Z**. Each list remembers your choice until you sign out.

People who can open even a single entity in a module see the module in their navigation, so they can reach that entity; they still need a permission on the module to change it.

Edit a module

1. In the **Modules** list, select the **three dots** next to the module and select **Edit**.
2. Change the name, icon, description or sharing, and confirm.

Delete a module `Since 1.10`

Deleting a module removes it **with everything in it** — every entity and its records, pages, the dashboard and functions — and **cannot be undone**. Only a **system administrator** can delete a module, and not the core module or a module installed from a package.

1. In the **Modules** list (or the module's own header menu), select the **three dots** and select **Delete**.
2. The **Delete module** dialog lists everything **outside** the module that still points at it or at something in it — a reference field in another module, a page, a counter widget. Open each one and remove the link. **Delete** stays disabled until nothing is left.
3. Tick **I understand that this module, all its entities and their entries will be permanently deleted.** and select **Delete**.
4. The delete runs in the background. The module shows **Deleting…** with the number of entities left, and disappears when it is done. If it stops, the module shows why; select **Delete again** to carry on from where it stopped.

While a module is being deleted, nobody can edit it, add entities or pages to it, or move entities into it.

To remove a single entity instead, see Delete an entity permanently.

Troubleshooting

Problem	What to check
Add Module is missing	Creating modules needs system **Developer** access.
The normalized name is refused	Another module already uses it.
Delete is missing	Only system administrators see it, and never on the core module or a package-installed module.
Delete stays disabled	Something outside the module still points at it. Remove each link the dialog lists.
The delete stopped part-way	Read the reason shown on the module, fix it, and select **Delete again**.
A space does not see the module	The module's **Sharing level** is **Private** or **Shared** and that space is not selected.

Go deeper

For integrators: Module model and API reference [Integrate & Extend] — the module model and sharing, routes, the background delete and its refusals, authorization, the sidebar endpoint and end-user view discovery

Choose a field type

This page is for **builders** designing an entity. Every piece of information a record holds is a **field**, and every field has a **type**. The type decides what people can enter, how it is checked, and how it is shown. Use this page to pick the right type for each thing you need to capture, then follow the link to the type's own page.

How it works

You add fields on the entity's **Structure** tab: drag a type from the **Fields** list onto the canvas (or double-click it), name the field, then open its settings. Most settings — default value, lock, hide, required, unique, display — are the same for every type and are described once on Settings every field shares.

Take an **Asset Request** entity, where an employee asks for a laptop or a phone. It needs a request number, the type of asset, a reason, a date it is needed by, the person asking, an estimated cost, a quotation document, and the delivery address. Each of those is a different field type — the table below shows which.

Choose by what you need to capture

You need to capture…	Use	Example
A short piece of text: a name, a title, a code	Text	*Asset model*
An email address or a web link	Text with an **Email address** or **URL** format	*Supplier website*
A paragraph, a description, notes, formatted text	Long-text	*Reason*
Text in both Arabic and English	Localized Text (one line) or Localized Long Text (several lines)	*Asset name* shown in each language
A secret such as a PIN	Password	*Device PIN*
A phone number	Phone	*Contact number*
A quantity, an amount of money, a percentage	Number	*Estimated cost* in SAR
A number the system assigns in order, or a formatted reference	Sequence Field	*Request number* AR-2026-0001
A date, or a period from one date to another	Date	*Needed by*
A time of day	Time	*Preferred delivery time*
A date and a time together	Date-time	*Handed over at*
A month, a month of a year, or a year	Month, Month-year, Year	*Budget year*
Yes or no	Boolean or Switch	*Replacement for a broken device?*
One or more options from a list you define	Category	*Asset type*: Laptop, Phone, Monitor
A few fixed answers shown as a checkbox, toggle or radio buttons	Switch	*Priority*: Normal, Urgent
A record from another entity or a list	Reference	*Department*
A person	User	*Requested by*

You need to capture…	Use	Example
An application, service, page, view or report in the platform	Application Reference	*Related service*
A value copied from a linked record	Mapped Field	The *Department manager*, taken from the chosen department
A file of any kind	File	*Quotation*
A picture	Image	*Photo of the broken device*
A document to view in the page	Document	*Signed approval form*
A video	Video	*Installation walkthrough*
An icon from the platform's icon set	Icon	*Asset type icon*
A location on a map	Map	*Delivery address*
Several fields that belong together, or rows of them	Block	*Items*: one row per device, each with model and quantity

Text, Long-text, Localized Text, Localized Long Text, File, Image, Document and Video can also be added as a **Display only field**: the field shows content you set — an instruction, a banner image — instead of collecting a value.

What each type can do

Not every setting applies to every type. ✓ means the type offers the setting.

Type	Searchable	Unique	Multiple	Sensitive	Encrypt
Text, Long-text, Localized Text, Localized Long Text	✓	✓		✓	✓
Password				✓	Always hashed
Phone	✓	✓		✓	✓
Number, Year	✓	✓		✓	✓
Sequence Field	✓	Always unique		✓	✓
Date, Time, Date-time, Month, Month-year	✓	✓		✓	✓
Boolean				✓	✓
Switch				✓	
Category	✓		✓	✓	
File, Image, Document, Video			✓	✓	✓
Icon, Map				✓	✓
Reference, User	✓	✓	✓		
Application Reference					
Mapped Field	Follows the source field			From the source field	From the source field
Block					

Every type except Sequence Field can also be **Repeatable** — a repeatable Block repeats its whole group of fields. See Multiple and Repeatable.

Good to know

- **Pick the type before records exist.** Some changes to a field on a published entity — such as turning **Multiple** on — mark existing records as out of date until they are upgraded. See Entities.
- **Calculated values.** Any type with a default value can work it out with an expression instead. See Give a field a starting value.
- **Protecting a value.** A field can be **Sensitive** (masked except for people you grant) or **Encrypt** (stored encrypted), never both. See Protect a field.
- **Filtering.** A type that is not searchable can still often be used in filters and queries — the **Queryable** setting decides that. See Filter with queries.
- **Showing a value.** Each type has its own way of showing a stored value in record details, tables, cards and badges — a number as currency, a date as *3 days ago*. Where a field is editable, people can also edit it in place: select the value (or press Enter), press Esc to cancel, Ctrl+S to save, or click outside the field to save.
- Some types exist only for the platform's own entities and are not in the **Fields** list — JSON, SVG Icon, Page Reference, Number Enum and the Data Query Selector.

Go deeper

For integrators: Field configuration reference [Integrate & Extend] — storage types behind each field type, capability flags, the calculated-value payload and server rules, encrypted fields · Internal field types [Integrate & Extend]

Settings every field shares

This page is for **builders** who add fields to an entity. Most field types share the same settings — a name, a starting value, whether people can change it, whether it is required, how it is searched and shown. They are described once, here. Each field-type page then covers only what is special about that type.

By the end you will be able to open a field's settings, give it a default or a calculated value, lock, hide or require it on conditions, protect it, make it searchable or unique, let it hold more than one value, add validation, and decide how it is shown in lists.

How field settings work

You add fields on the entity's **Structure** tab (open **Administration › Applications › Modules**, open the module, open the entity, then select **Structure**). Each field is a row on the canvas. Select the settings icon on a field's row to open its **Field Settings** panel. A list down the side of the panel jumps to each section and lets you search for a setting by name.

The panel is divided into sections. Which ones you see depends on the field type:

Section	What it holds	Shown for
Read Only Settings	Whether the field is a **Value field** or a **Display only field** (set when you add it)	Text, Long-text, Localized Text, Localized Long Text, File, Image, Document, Video
Basic settings	**Name**, **Normalized name**, **Description**, **Field type**	Every field
Type-specific sections	For example **Data Source**, **Add options**, **Answers options**, **Auto increment settings**	See the field-type page
Default Value	A fixed starting value, or a calculated one (**Advanced**)	Most types
Field Attributes	**Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**	Most types; each attribute only where it applies
Type / Formatting	How the value is entered or formatted	Long-text, Text, Number, File, Image, Document, Video
Value Settings	**Multiple**, **Repeatable**, **Custom Validation**, **Min & Max**, file sizes	Most types
Value Validators	Reusable validation rules	Most types
Display	Showing the field in lists, ranges, display roles such as **Main title**, column span	Every field

Your changes take effect when you select **Save** at the top of the entity designer. To remove a field, select **Delete Field** at the bottom of its settings panel.

A worked example

The examples on this page use an **Asset Request** entity: an employee asks for a laptop or a phone. It has a *Request number*, an *Asset type*, a *Reason*, a *Needed by* date, the *Requested by* user, and an *Estimated cost*.

Name a field

Under **Basic settings**:

1. Enter the **Name** in Arabic and English. This is the label people see on forms and in lists.
2. Check the **Normalized name**. It is filled in for you from the English name. It must start with an English letter and contain only English letters and numbers, and it must be unique within the entity.
3. Optionally, add a **Description**.

Field type shows the type you picked when you added the field.

The normalized name is how expressions, queries, reports and integrations refer to the field. Choose it before people start relying on it: renaming it later breaks anything that still uses the old name.

Give a field a starting value

The **Default Value** section sets what a new record shows in this field before anyone types. You have two choices.

A fixed value. Enter or pick the value in **Default Value**. Every new record starts with it, and people can change it unless the field is locked.

A calculated value. Tick **Advanced** to work the value out with an expression — for example, today's date, or the cost multiplied by the quantity.

1. In **Default Value**, tick **Advanced**.
2. In **Type expression**, write the expression that produces the value.
3. If the value should be kept up to date by the system, turn on **Lock Field** (under **Field Attributes**). **Execution Triggers** then appears in the **Default Value** section. Choose when the expression runs:
   - **On page load** — when the form opens (new records only).
   - **On save** — when the record is saved.
   - **On entry change** — whenever the record changes.
   - **On field change** — when one of the fields you pick in **Watched fields** changes.
4. Choose the **Execution Scope** — which records the calculation applies to: **New entries**, **Existing entries** or **All entries**. Pick the trigger first: changing the trigger resets the scope to **New entries**.
5. Select **Save**.

For a **Reference** or **User** field, you can instead turn on **First value** to pre-select the first option the field offers.

Example — *Estimated cost*. The *Asset Request* has *Unit price* and *Quantity*. Make *Estimated cost* a locked Number field with an **Advanced** default value `value.unitPrice * value.quantity`, trigger **On field change** watching *Unit price* and *Quantity*, and scope **All entries**. The cost now follows the two fields, and nobody can type over it.

Some types have no default value: File, Image, Document, Video, Icon, Sequence and Mapped fields.

Technical details

- The expression is worked out twice: live in the form as the triggers fire, and again on the server when the record is saved. A record saved by an automation or an integration therefore gets the same value.
- On save, the server applies the calculated value when the field is locked or hidden (a value sent for it is replaced), or when no value was sent and there is no default.
- Leave **Execution Scope** empty and the server never recalculates the field on save. **On page load** never recalculates on update.

> **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — the stored form of calculated values, trigger and scope values, and exactly when the server recalculates

Lock a field

A locked field shows its value but nobody can change it on the form. Use it for values the system fills in — a calculated total, a reference number, the person who created the request.

1. Under **Field Attributes**, turn on **Lock Field**.
2. To lock the field only in some situations, enter a **Conditional expression**. The field is locked while the expression is true. Leave it empty to lock it always.
3. Give the field a value to show: a fixed **Default Value**, or an **Advanced** calculated one. A field that is both locked and always required needs a default value; with **Advanced** on, the expression is mandatory.

Example. A Boolean field *Completed* should be ticked and frozen once the *Status* field is *Closed*. Turn on **Lock Field** with a conditional expression that checks for *Closed*, and give *Completed* a calculated default that returns true in the same case.

The lock is not just on the form. A value sent for a locked field by an automation, a script or an integration is ignored too, and the field keeps its stored value. The field's own calculated value still applies, which is why **Lock Field** plus an **Advanced** default value is the usual way to build a field the system maintains.

Block fields cannot be locked; lock the fields inside the block instead.

Hide a field

Hide Field removes the field from the form, always or on a condition.

1. Under **Field Attributes**, turn on **Hide Field**.
2. Enter a **Conditional expression**. The field is hidden while the expression is true.

Example. A form has a *Subscription* switch and a *Subscription title* text field. Hide *Subscription title* while the switch is off, so people only see it when it applies.

A field that is always hidden can't be required — the form shows *The hidden field can't be set to required*. A required field inside a hidden **Block** is not enforced while the block is hidden.

Make a field required

1. Under **Field Attributes**, turn on **Required**.
2. To require the field only in some situations, enter a **Conditional expression**. The field is required while the expression is true.

You can also tick **Required** directly on the field's row on the **Structure** tab. For multilingual fields you can say which languages must be filled in — see Localized Text. A Sequence field can never be required (the platform fills it in), and a Block can't be required — require the fields inside it.

Protect a field: Sensitive or Encrypt

Two settings under **Field Attributes** protect a value. A field can have one of them, not both: turning one on turns the other off.

Sensitive

The value is masked for everyone except the users and groups you grant. Choose them with **Select Users** and **Select groups**; **Show Inherited** lists those who have access through broader permissions. Use it when most people may see that a value exists but only a few may read it.

Encrypt `Since 1.10`

The value is stored encrypted. Only users with access to this sensitive field can decrypt and view the real value, one value at a time. Use it for values that must be unreadable even in storage, such as a national ID number.

Which to use

	Sensitive	Encrypt
Not offered for	Reference, User, Application Reference, Block, Mapped	Reference, User, Application Reference, Block, Mapped, Switch, Category
Searchable	Turned off	Turned off
Can be the **Main title**	No	No

An encrypted value can't be read without decrypting it, so an Encrypted field also can't be used in filters, query conditions, reaction conditions, summaries or reports.

A Mapped field copies both settings from the field it is mapped from. A Password field is always stored as a one-way hash, so it never shows a decrypt action. Turning **Encrypt** on or off for a field that already has records starts an upgrade of those records.

Who may see a protected value is managed by the tenant administrator — see Sensitive field grants [Administration › Sensitive Permissions]. The full behaviour — masking, revealing, the decrypt action and the record upgrade — is on Sensitive Fields.

Let people search and filter on a field

- **Searchable** — the field's value is included when people type in a list's search box. For example, make *Reason* searchable so searching for *broken screen* finds the request.
- **Queryable** — the field can be used in filters and queries.

Not every type offers both. A Sensitive or Encrypted field is never searchable.

Prevent duplicates

Unique stops two records from having the same value in this field. When someone saves a duplicate, the platform refuses it and tells them the value must be unique.

1. Under **Field Attributes**, turn on **Unique**.
2. Select **Save**.

Example. Turn on **Unique** for *Employee ID* so no two employees can be saved with the same ID. Don't use it for values people legitimately share, such as a job title.

Unique is offered for Text, Long-text, Localized Text, Localized Long Text, Phone, Number, Date, Time, Date-time, Month, Month-year, Year, Reference and User fields. A Sequence field is unique automatically. To keep an *optional* field unique only when it is filled in, use a custom index on the entity's **Indexes** tab — see Entities.

Allow more than one value: Multiple and Repeatable

Under **Value Settings**, two settings let a field hold more than one value:

- **Multiple** — one field, several picks. People choose more than one option in the same field: several approvers, several asset categories, several attachments. Offered for Reference, User, Category, File, Image, Document and Video fields. For most of them you can set a **Min** and **Max** number of picks.
- **Repeatable** — the field repeats. People add the field again, each time with its own value — for example, one *Phone* row per contact number. Set **Min** and **Max** to limit the rows; leave them empty for no limit.

To repeat a group of fields together (a name *and* an ID number per team member), make a repeatable Block.

A field that holds several values can't be the entity's **Main title**, **Main category** or **Sub category**. A Sequence field can't be repeatable.

Check what people enter

Under **Value Settings** and **Value Validators**:

- **Min & Max** — the smallest and largest allowed value (for numbers and dates) or **Min length** and **Max length** (for text). Tick **Advanced** to work the limits out with **Min Expression** and **Max Expression** instead.
- **Custom Validation** — select **Add Custom Validation**, write an **Expression** that must be true, and the **Error Message** people see when it is not. Use **Enable** to switch a rule off without deleting it.
- **Value Validators** — pick reusable validation rules with **Select Value Validators**, or create one with **Add Value Validator**.

Example. On *Needed by*, add a custom validation that the date is not in the past, with the message *Choose a date from today onwards*.

Decide how a field is shown

Under **Display**:

- **Display on table** — show the field as a column in the entity's lists, so people see it without opening each record. (The label reads *Not display on table* while it is off.)
- **Is ranged** — for Date, Time, Date-time, Month-year and Year fields: people pick a start and an end in the same field, such as a leave period.
- **Set Field As** — give the field a role in how every record is presented. Each role can be held by one field per entity:

Role	Typical field	Where it shows
Main title	Text, Long-text, Localized Text, Phone, Category, Reference, User, or a date or time that is not ranged	The record's title wherever it is listed or linked
Sub title, **Caption**	Text-type fields	Under the title
Main number	Number, Sequence	Highlighted number on the record
Main date	Date, Month, Month-year	The record's main date
Main phone	Phone	The record's contact number
Main picture "use as entry picture"	Image, Icon	The record's picture
Main category, **Sub category**	Category	The record's category badges
Main reference	Reference, User	The record's main link

- **Span** — how many columns the field takes on the form.

Good to know

- Settings an installed package locks can't be changed; you can still add your own fields.
- Changing a field's type-related settings on a published entity — for example turning **Multiple** on — can be a breaking change: existing records are marked out of date until they are upgraded. See Entities.
- **Help & documentation** at the end of the panel lets you write help text that people see next to the field.

Go deeper

For integrators: Field configuration reference [Integrate & Extend] — how a field and its settings are stored, calculated-value rules, capability flags per type

Text field

Use a **Text** field for a short, single-line value: a name, a title, a model number, an email address, a web link. This page covers what is special about Text fields — formats, masks and patterns. The settings every field has are on Settings every field shares.

For several lines or formatted text, use Long-text. For text that must be entered in both Arabic and English, use Localized Text.

Add a Text field

1. On the entity's **Structure** tab, drag **Text** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Formatting**, choose how the value is checked (see below).
4. Set any shared settings you need — for example **Required** or **Unique**.
5. Select **Save**.

Check what people type

Under **Formatting**, pick a format from the list:

- **Unformatted** — any text. This is the default.
- **Email address** — the value must be a valid email address.
- **URL** — the value must be a valid web address.
- **CUSTOM** — the value must match a pattern you write in **Regular expression**. For example, `^[A-Z]{3}-[0-9]{4}$` accepts asset tags such as *LAP-0042*.

When a value does not match, the form shows **Invalid value** under the field and the record can't be saved until it is corrected.

Example. On an *Asset Request*, a *Supplier website* field with the **URL** format rejects *supplier dot com* but accepts *https://supplier.example*.

Guide what people type with a mask

A mask shows the expected shape of the value while people type — useful for IDs, license plates or account numbers with a fixed layout.

1. Under **Formatting**, turn on **Masked input**.
2. In **Input mask**, enter the mask.
3. In **Save value**, choose **Raw** to store only what people typed, or **Masked** to store it with the mask's symbols included. This also decides how the value is matched when people search and filter.

Other input options

- **Barcode Reader** — people can fill the field by scanning a barcode instead of typing it, for example an asset's serial number.
- **Speech input** — people can dictate the value. This option appears only when speech input is available on your platform.

Settings it shares

A Text field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, validation (**Min length** and **Max length**, **Custom Validation**), **Display on table**, and the display roles **Main title**, **Sub title** and **Caption**. Each is described on Settings every field shares.

A Text field can also be added as a **Display only field**, to show fixed text such as an instruction on the form.

Good to know

- Turn on **Unique** for values that identify a record, such as *Employee ID* — not for values people share, such as a job title.

Long-text field

Use a **Long-text** field for text that runs over several lines: a description, the reason for a request, meeting notes, an article. This page covers what is special about Long-text fields — choosing plain or formatted text. The settings every field has are on Settings every field shares.

For one line of text, use Text. For long text in both Arabic and English, use **Localized Long Text** — see Localized Text.

Add a Long-text field

1. On the entity's **Structure** tab, drag **Long-text** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Type**, choose how people write in it (see below).
4. Select **Save**.

Choose plain or formatted text

The **Type** setting decides the editor people get:

Type	People get	Use it for
Normal	A plain multi-line box	Notes and reasons where formatting does not matter
Markdown	A light editor that stores Markdown	Simple formatting: bold, lists, headings
Rich Text	A full editor with a formatting toolbar	Content people read on screen, such as an announcement
Rich with code	The full editor, plus the option to edit the underlying source	Content that needs precise layout

The formatting toolbar offers bold, italic, underline, strikethrough, headings, numbered and bulleted lists, quotes, code, tables and clear formatting (Markdown has no underline). Formatted text is cleaned of anything unsafe before it is shown.

Example. An *Asset Request* uses a **Normal** Long-text field for *Reason*. A *Service announcement* entity uses **Rich Text** for its *Body*, so the IT team can add headings and a bulleted list of affected systems.

Settings it shares

A Long-text field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, validation (**Min length**, **Max length**, **Custom Validation**), **Display on table**, and the display roles **Main title**, **Sub title** and **Caption**. Each is described on Settings every field shares.

A Long-text field can also be added as a **Display only field**, to show a block of fixed text — instructions, terms and conditions — on the form.

Go deeper

For integrators: Field configuration reference [Integrate & Extend] — how each **Type** is stored and how formatted text is rendered

Localized Text field

Use a **Localized Text** field when a value must be written in more than one language — an asset's name in Arabic and English, a service title that each user sees in their own language. People enter one value per language, and each reader sees the value for the language they are using. This page covers what is special about localized fields. The settings every field has are on Settings every field shares.

There are two localized types in the **Fields** list:

- **Localized Text** — one line per language, like Text.
- **Localized Long Text** — several lines per language, written in Markdown, like Long-text.

Add a localized field

1. On the entity's **Structure** tab, drag **Localized Text** or **Localized Long Text** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. If the field must be filled in, set **Required** and choose the languages (see below).
4. Select **Save**.

Decide which languages are required

When a localized field is required, you also decide *which* languages people must fill in. Under **Value Settings**, in **Required language**, choose a **Validation Mode**:

- **Required value for all languages** — every language must be filled in.
- **Required value for at least one language** — any one language is enough.
- **Required value for the following languages** — only the languages you pick in **Languages**.

Example. A *Service* entity's *Title* must exist in Arabic, while English is welcome but optional. Make the field **Required**, then choose **Required value for the following languages** and pick Arabic.

Settings it shares

Localized fields also offer a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, validation (**Min length**, **Max length**, **Custom Validation**), **Display on table**, and the display roles **Main title**, **Sub title** and **Caption**. Each is described on Settings every field shares.

Both localized types can also be added as a **Display only field**, to show fixed text in each language on the form.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — how localized long text is stored and rendered

Number field

Use a **Number** field for anything people count or measure: a quantity, an age, a cost, a score, a completion percentage. This page covers what is special about Number fields — how a number is formatted and shown. The settings every field has are on Settings every field shares.

For a number the system assigns in order (a request number), use Sequence Field. For a year, use Year.

Add a Number field

1. On the entity's **Structure** tab, drag **Number** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Formatting**, choose a **Format** (see below).
4. To limit the allowed values, set **Min & Max** under **Value Settings**.
5. Select **Save**.

Choose how the number is formatted

Under **Formatting**, the **Format** list offers three formats.

General

The number is shown as entered. This is the default and suits counts, ages and scores.

Percentage

The number is a percentage: a **%** sign appears next to the box while people type.

1. Set **Format** to **Percentage**.
2. Choose a **Display Format**:
   - **Percentage** — the number with a % sign (the default).
   - **Progress bar** — a bar filled to that percentage.
   - **Progress bar with percentage** — the bar with the number on it.
3. Choose the **Precision**: how many decimal places to show, from 0 to 3 (the default is 0).

Example. A *Project task* has a *Completion* field set to **Progress bar with percentage**. When someone enters *75*, the task shows a bar three-quarters full, labelled 75%.

Precision applies to calculated values too `Since 1.8`

The **Precision** you choose applies to every value the field shows, including values written by a calculation, an automation or a script — not only values people type. A calculated *33.3333…* shows as *33%* with a precision of 0.

Currency

The number is an amount of money.

1. Set **Format** to **Currency**.
2. In **Type**, choose the currency.

3. Set how amounts are written:
    - **Decimal Places** — how many digits after the decimal point.
    - **Decimal separator** — **Comma and period** (1,000,000.00), **Period and comma** (1.000.000,00), **Space and period** (1 000 000.00) or **Space and comma** (1 000 000,00).
    - **Large Number Abbreviation** — **None**, **Thousand**, **Million** or **Billion**, to shorten large amounts.
    - **Allow negative number** — turn on if the amount can be below zero, such as a refund.

Example. An *Asset Request* has an *Estimated cost* field: **Currency**, type Saudi Riyal, 2 decimal places, abbreviation **Million**, negative numbers allowed. An entry of *-1500000* is shown as *-1.5M*.

Settings it shares

A Number field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, validation (**Min & Max**, **Custom Validation**), **Display on table**, and the display role **Main number**. Each is described on Settings every field shares.

A locked Number field with a calculated default is the usual way to show a total — see the *Estimated cost* example in Give a field a starting value.

Date field

Use a **Date** field for a calendar day: a due date, a date of birth, the day an asset is needed by — or, as a range, a period such as a leave request from one date to another. People pick the date from a small calendar. This page covers what is special about Date fields. The settings every field has are on Settings every field shares.

Other date and time types: Time, Date-time, Month, Month-year and Year.

Add a Date field

1. On the entity's **Structure** tab, drag **Date** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Capture a period: a date range

A ranged Date field holds a start date and an end date in one field.

1. Under **Display**, in **Ranged Date**, tick **Is ranged**.
2. Select **Save**.

People then pick two dates in the same calendar; the first must be earlier than the second.

Example. A *Leave request* has one ranged *Leave period* field instead of two separate *From* and *To* fields.

A ranged date can't be the record's **Main title**.

Make it the record's main date

Under **Display › Set Field As**, tick **Main date** to make this the date that represents the record — for example the *Needed by* date of an asset request. One field per entity can hold this role.

Default to today

To start every new record with today's date, tick **Advanced** in **Default Value** and enter an expression that returns today's date, such as `scope.today()`. See Give a field a starting value.

Filter on dates relative to today

In a query or a filter you can compare a date with a moving point in time instead of a fixed date, so the filter stays right without being edited. Choose **Pointer** as the source and pick one of:

Pointer	Means
@now	This moment
@today	Today
@StartOfDay	The start of today

Pointer	Means
@StartOfWeek	The start of this week
@StartOfMonth	The start of this month
@StartOfYear	The start of this year

Example. A *Requests this month* list filters on *Requested on* being on or after **@StartOfMonth**. On the first of next month it starts again, with nobody editing the filter. See Filter with queries.

A date can also be shown relative to today — *3 days ago*, *in 2 weeks* — instead of as a calendar date.

Settings it shares

A Date field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, validation (**Min & Max**, **Custom Validation**), **Display on table**, and the display roles **Main date** and (when not ranged) **Main title**. Each is described on Settings every field shares.

> ### Go deeper
>
> For integrators: Field configuration reference [Integrate & Extend] — date pointers and the relative-period display

Time field

Use a **Time** field for a time of day without a date: an opening time, a preferred delivery time — or, as a range, a span such as a shift from 08:00 to 16:00. People pick the time in a small picker with up and down arrows for hours and minutes. This page covers what is special about Time fields. The settings every field has are on Settings every field shares.

For a date and a time together, use Date-time.

Add a Time field

1. On the entity's **Structure** tab, drag **Time** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Capture a span: a time range

1. Under **Display**, in **Ranged Date**, tick **Is ranged**.
2. Select **Save**.

People then pick a start time and an end time in the same field.

Example. A *Meeting room booking* has one ranged *Time slot* field, from 10:00 to 11:30.

A ranged time can't be the record's **Main title**.

Settings it shares

A Time field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation**, **Display on table**, and (when not ranged) the display role **Main title**. Each is described on Settings every field shares.

Date-time field

Use a **Date-time** field when both the day and the time matter: when an asset was handed over, when an incident started — or, as a range, a period such as a maintenance window from Friday 22:00 to Saturday 06:00. People pick the date and the time in one picker. This page covers what is special about Date-time fields. The settings every field has are on Settings every field shares.

For a date alone, use Date; for a time alone, use Time.

Add a Date-time field

1. On the entity's **Structure** tab, drag **Date-time** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Capture a period: a date-time range

1. Under **Display**, in **Ranged Date**, tick **Is ranged**.
2. Select **Save**.

People then pick a start date and time and an end date and time in the same field.

Example. A *Planned outage* has one ranged *Outage window* field, so the start and end can never be stored apart.

A ranged date-time can't be the record's **Main title**.

Relative dates

Date-time fields can be filtered against moving points in time such as **@now** or **@StartOfWeek**, exactly like Date fields — see Filter on dates relative to today.

Settings it shares

A Date-time field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation**, **Display on table**, and (when not ranged) the display role **Main title**. Each is described on Settings every field shares.

Month-year field

Use a **Month-year** field for a month in a particular year, when the exact day does not matter: a reporting period (*March 2026*), a target delivery month, a budget month — or, as a range, a span of months. This page covers what is special about Month-year fields. The settings every field has are on Settings every field shares.

Add a Month-year field

1. On the entity's **Structure** tab, drag **Month-year** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Capture a span of months

1. Under **Display**, in **Ranged Date**, tick **Is ranged**.
2. Select **Save**.

People then pick a start month and an end month in the same field.

Example. A *Training plan* has one ranged *Plan period* field, from *January 2026* to *June 2026*.

Make it the record's main date

Under **Display › Set Field As**, tick **Main date** to make this the date that represents the record. A ranged field can't be the **Main title**.

Settings it shares

A Month-year field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

Year field

Use a **Year** field when only the year matters: a budget year, an academic year, a vehicle's model year — or, as a range, a span such as a three-year contract. People pick the year from a list. This page covers what is special about Year fields. The settings every field has are on Settings every field shares.

Add a Year field

1. On the entity's **Structure** tab, drag **Year** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Capture a span of years

1. Under **Display**, in **Ranged Date**, tick **Is ranged**.
2. Select **Save**.

People then pick a start year and an end year in the same field.

Example. A *Contract* has one ranged *Contract term* field, from 2026 to 2028.

Good to know

A year is stored as a number, so you can limit it with **Min & Max** under **Value Settings** — for example, no earlier than 2020.

Settings it shares

A Year field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

Month field

Use a **Month** field when only the month of the year matters, not the day or the year: a billing month, the month a seasonal campaign runs, a birth month. People pick the month from a list. This page covers what is special about Month fields. The settings every field has are on Settings every field shares.

For a month in a particular year, use Month-year.

Add a Month field

1. On the entity's **Structure** tab, drag **Month** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set any of the options below, and the shared settings you need.
4. Select **Save**.

Limit the allowed months

Under **Value Settings**, set **Min & Max** to the first and last month allowed, as month numbers from 1 (January) to 12 (December).

Example. A *Summer programme* accepts June to August only: set **Min** to 6 and **Max** to 8.

Make it the record's main date

Under **Display › Set Field As**, tick **Main date** to make this month the date that represents the record.

Good to know

A Month field holds a single month; it can't be ranged. For a span of months, use a ranged Month-year field.

Settings it shares

A Month field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation**, **Display on table**, and the display roles **Main date** and **Main title**. Each is described on Settings every field shares.

Block field

Use a **Block** field to group fields that belong together — an address made of street, city and postcode, or the details of a vehicle. Make the block **Repeatable** and people can add as many copies as they need, each with its own values: one row per requested item, per team member, per emergency contact. This page covers what is special about Block fields. The settings every field has are on Settings every field shares.

Add a Block field

1. On the entity's **Structure** tab, drag **Block** from the **Fields** list onto the canvas.
2. Name the block, then drag the fields it should contain into it.
3. Select the block's settings icon to open **Field Settings** and set the options below.
4. Select **Save**.

Repeat the block: rows of fields

1. Under **Value Settings**, turn on **Repeatable**.
2. Optionally, set **Min** and **Max** — the fewest and most rows. Leave them empty for no limit.
3. Select **Save**.

Example. An *Asset Request* has an *Items* block with *Asset type*, *Model* and *Quantity*. With **Repeatable** on, an employee can ask for a laptop, a monitor and a docking station in one request — three rows, each with its own values — instead of filing three requests.

Rows keep their identity `Since 1.10`

When someone deletes or reorders rows, each row keeps its own values — including locked and protected ones. The platform tells rows apart by an identity of their own, not by their position.

Lay out the fields

Under **Value Settings**, **Display Layouts** sets how the block's fields are laid out on the record: **Key Value View** (each field's name next to its value), or **Two Columns View**, **Three Columns View** or **Four Columns View**.

Example. A *Product details* block with name, description and price uses **Three Columns View**, so the three values sit side by side.

Show the block as its own tab

Under **Display**, tick **Display as tab** to show the block's fields on a separate tab of the record instead of with the other fields. (The label reads *Not display as tab* while it is off.)

Example. A project's *Task information* block is long; showing it as a tab keeps the main view short.

Rules for fields inside a block

- **Lock, protect and require the fields, not the block.** A Block field itself can't be locked, **Required**, **Sensitive** or **Encrypted**; set those on the fields inside it.
- **Hiding a block suspends its required fields.** While a block is hidden — by its own **Hide Field** condition or by a workflow step — the required fields inside it are not enforced. In a repeatable block this is decided row by row, so a hidden row doesn't block the save while a visible row still must be complete.
- **Revealing an encrypted value is per row.** In a repeatable block, a person with access reveals an **Encrypted** field one row at a time.
- **Simple calculated fields** inside a block stop it from being made repeatable; turn their calculation off first.

Editing rows through a view `Since 1.4`

A view may let people edit only some of a block's fields. When someone saves a repeatable block through such a view, the platform keeps the values of the fields the view does not show, row by row — they are never wiped by the save. Rows the person removes in the view are removed. (A view that shows none of the block's fields never changes the block.)

Settings it shares

A Block field also offers **Hide Field**, **Repeatable** and the **Display** settings above. A Block itself can't be made **Required** — make the fields inside it required instead. Each is described on Settings every field shares.

> **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — row ids, how rows are matched on save and through a view, and per-row decryption

Boolean field

Use a **Boolean** field for a simple yes-or-no value: *Completed*, *Urgent*, *Replacement for a broken device?*. People tick or untick a single checkbox; the value is either true or false. This page covers what is special about Boolean fields. The settings every field has are on Settings every field shares.

To show yes/no as a toggle or as radio buttons with your own labels, use Switch instead.

Add a Boolean field

1. On the entity's **Structure** tab, drag **Boolean** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Default Value**, tick the box if new records should start as *yes*; leave it unticked for *no*.
4. Select **Save**.

Examples

- **Start unticked.** A *Completed* field on a task starts unticked: leave the default box empty, and every new task starts as not completed.
- **Set it automatically.** A *Completed* field should be ticked and frozen once the task's *Status* is *Closed*. Turn on **Lock Field** with a conditional expression that checks for *Closed*, and give the field an **Advanced** default value that returns true in the same case — see Lock a field.
- **Show more fields when ticked.** Use the Boolean in another field's **Hide Field** condition: when *Subscription* is ticked, a *Discount code* field appears; when it is unticked, the field is hidden.

Settings it shares

A Boolean field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Queryable**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

Category field

Use a **Category** field when people pick from a list of options that you define: an asset type (*Laptop, Phone, Monitor*), a request status, a region. Each option can have its own colour, so records stand out at a glance in lists and boards. This page covers what is special about Category fields. The settings every field has are on Settings every field shares.

For two or three fixed answers shown as radio buttons or a toggle, a Switch may be simpler. To pick a record from another entity rather than a fixed option, use Reference.

Add a Category field and its options

1. On the entity's **Structure** tab, drag **Category** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Add options**, type the option's name, select the translation icon at the end of the box to add its Arabic and English text, then select **Add**.
4. Choose a colour for the option.
5. Repeat for each option.
6. Optionally, pick a **Default Value** from the options.
7. Select **Save**.

Copy option id copies an option's internal identifier, for use in an expression or an integration that needs to refer to that option.

Let people pick several options `Since 1.10`

A Category field can hold several options in one record — for example, the *Skills* of an employee.

1. Under **Value Settings**, turn on **Multiple**.
2. Optionally, set **Min** and **Max**: the fewest and most options people may pick.
3. Select **Save**.

People see their picks as removable chips, on the web and in the mobile app, and tables show every value. A minimum of 1 does not make the field required — use **Required** for that.

Before you turn **Multiple** on, know that:

- **It is a breaking change.** Existing records are marked out of date until they are upgraded; each one's single value becomes a one-item list. See Entities.
- **Turning it off keeps only the first pick** of each record.
- **It is refused while the field is in use** as a Kanban board's columns, a grouping, a sort, a dashboard or a timeline in a view of the same entity. The message names those places — remove the field from them first.
- **Some roles need a single value.** A field with **Multiple** can't be the entity's **Main category** or **Sub category**, a Kanban column field, a default sort, a dashboard grouping or the timeline's mapping.

Let people add and edit options while they work `Since 1.10`

Normally only builders change a Category field's options. You can let end users add an option, or rename one, from a record's create or edit card — useful when the list grows with the business, such as a list of suppliers.

1. In the Category field's settings, turn on **Allow users to manage options**. (This is not offered when the field's options are mapped from another field.)
2. In the view that people use, open its card settings and turn on **Add category options** and/or **Edit category options** for the create card and the edit card. **Link create and edit card settings** keeps the two cards the same.
3. Save the entity and the view.

Both settings must be on. If someone else changes the same option at the same moment, the second person is asked to try again. How people use this is described in Adding and editing category options while you fill in a record [Using OSOS › Create and Edit Records].

Make it the record's category

Under **Display › Set Field As**, tick **Main category** to show this field's value as the record's category badge, and **Sub category** for a second, more detailed one. A Category field can also be the **Main title**.

Settings it shares

A Category field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Searchable**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

A Category field can't be **Encrypted**.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — option changes sent with an entry, permission checks and error codes · multiple values [Integrate & Extend › Field configuration reference] — stored shapes and counting rules · option display roles [Integrate & Extend › Field configuration reference]

Document field

Use a **Document** field for documents people attach to a record and read: a signed form, a policy, a contract. People can upload plain-text, PDF and Word files. This page covers what is special about Document fields. The settings every field has are on Settings every field shares.

For any other kind of file, use File.

Add a Document field

1. On the entity's **Structure** tab, drag **Document** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the limits below.
4. Select **Save**.

Limit what people upload

Under **Value Settings**:

- **Max file size** — the largest single file, in MB. It starts at 20 MB, which is also the most you can set.
- **Total files size** — when the field is **Multiple** or **Repeatable**, the most all the files together may add up to.

Files that are too large are refused as soon as people pick them, before anything is uploaded.

Let people upload several documents

Under **Value Settings**, turn on **Multiple** and, optionally, set **Min** and **Max** — the fewest and most documents per record.

Example. A *Policy* entity has a *Policy document* field for the signed PDF.

Show a fixed document on the form

Added as a **Display only field**, a Document field shows a document you upload once — such as terms to read — and does not ask people for anything.

Settings it shares

A Document field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. A Document field has no default value.

File field

Use a **File** field to let people attach a file to a record: a quotation, an invoice, a spreadsheet, a certificate. People browse to the file on their device and upload it. This page covers what is special about File fields. The settings every field has are on Settings every field shares.

For pictures, use Image; for documents people read in the page, Document; for videos, Video.

Add a File field

1. On the entity's **Structure** tab, drag **File** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the limits below.
4. Under **Formatting**, choose a **Format**: **File** shows the file's name and an icon, **Thumbnail** shows a small preview.
5. Select **Save**.

Limit what people upload

Under **Value Settings**:

- **Max file size** — the largest single file, in MB. It starts at 20 MB, which is also the most you can set.
- **Total files size** — when the field is **Multiple** or **Repeatable**, the most all the files together may add up to.
- **Acceptable File Types** — the file types people may upload, such as PDF or Excel. People can then only pick files of those types.

Files that are too large are refused as soon as people pick them, before anything is uploaded.

Let people upload several files

Under **Value Settings**, turn on **Multiple** and, optionally, set **Min** and **Max** — the fewest and most files per record.

Example. An *Asset Request* has a *Quotation* File field that accepts PDF only, up to 5 MB.

Uploaded files show an icon for their type, the file name and the size.

Show a fixed file on the form

Added as a **Display only field**, a File field shows a file you upload once — such as a form to download — and does not ask people for anything.

Settings it shares

A File field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. A File field has no default value.

Go deeper

For integrators: Field configuration reference [Integrate & Extend] — how file limits are stored and checked

Icon field

Use an **Icon** field to let people pick an icon from the platform's built-in icon set — for example, an icon for each asset type or each service in a catalogue. People select the field and choose from the list of icons. This page covers what is special about Icon fields. The settings every field has are on Settings every field shares.

Add an Icon field

1. On the entity's **Structure** tab, drag **Icon** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the shared settings you need.
4. Select **Save**.

Make it the record's picture

Under **Display › Set Field As**, tick **Main picture "use as entry picture"** so the icon represents the record wherever it is listed.

Example. An *Asset type* entity has an *Icon* field set as the main picture, so *Laptop*, *Phone* and *Monitor* each show their own icon in lists.

Settings it shares

An Icon field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. An Icon field has no default value.

Image field

Use an **Image** field to let people upload a picture: a photo of a damaged device, a product image, an ID photo. People can upload JPG, JPEG, PNG, GIF and WebP images. This page covers what is special about Image fields. The settings every field has are on Settings every field shares.

Add an Image field

1. On the entity's **Structure** tab, drag **Image** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the limits below.
4. Select **Save**.

Limit what people upload

Under **Value Settings**:

- **Max file size** — the largest single file, in MB. It starts at 20 MB, which is also the most you can set.
- **Total files size** — when the field is **Multiple** or **Repeatable**, the most all the files together may add up to.

Files that are too large are refused as soon as people pick them, before anything is uploaded.

Let people upload several images

Under **Value Settings**, turn on **Multiple** and, optionally, set **Min** and **Max** — the fewest and most images per record.

Make it the record's picture

Under **Display › Set Field As**, tick **Main picture "use as entry picture"** so this image represents the record wherever it is listed — for example, each employee's photo on their card.

Show fixed images on the form

Added as a **Display only field**, an Image field shows an image you upload once — such as a banner — and does not ask people for anything. Under **Read Only Settings**, **Display Size** sets how large it appears: **Full Width**, **Original Dimensions**, or a **Small**, **Medium** or **Large Thumbnail** (320, 640 or 1024 pixels wide).

Settings it shares

An Image field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. An Image field has no default value.

Map field

Use a **Map** field when a record needs a location: a delivery address, the site of an incident, a branch office. People see a map and place a marker by clicking on it, or search for a place by name and pick it from the results. This page covers what is special about Map fields. The settings every field has are on Settings every field shares.

Add a Map field

1. On the entity's **Structure** tab, drag **Map** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Optionally, set a default location: under **Default Value**, select the add button to open the map and pick the location.
4. Select **Save**.

How people pick a location

- **Click the map** to place the marker.
- **Search by name**: type a place or an address; matching places appear as you type, and choosing one moves the marker there.

Example. An *Asset Request* has a *Delivery address* Map field defaulting to the head office, so most requesters only need to confirm it.

If the field is **Encrypt**ed, a person with access who reveals the value sees the full map, at full width.

Settings it shares

A Map field also offers a calculated **Default Value** (as well as the fixed one above), **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — the map and search services and how locations are stored

Password field

Use a **Password** field for a secret that must never be shown again: a device PIN, an access code, a one-time passphrase. What people type is hidden as they type it, and once the record is saved nobody — not even an administrator — can read the value back. This page covers what is special about Password fields. The settings every field has are on Settings every field shares.

Add a Password field

1. On the entity's **Structure** tab, drag **Password** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the shared settings you need — usually **Required**.
4. Select **Save**.

How a password value behaves

- **Hidden while typing.** The characters are masked as people enter them.
- **Never shown again.** In record details, tables, cards and badges the field always shows `********`, including in hover tooltips. There is no reveal or decrypt button and no lock icon, because there is nothing to reveal: the platform keeps only a one-way fingerprint of the value, not the value itself. This holds even if **Encrypt** is turned on for the field.
- **Changing it means replacing it.** When someone edits the record, the box is empty; to change the password they type a new one.

Example. An *Equipment loan* record has a *Locker PIN* Password field. The employee sets the PIN when they borrow the equipment; the IT team can see that a PIN exists but never what it is.

Settings it shares

A Password field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. A Password field can't be searchable or unique, and it can't be the record's title.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — how password values are stored and masked

Phone field

Use a **Phone** field for a phone number: a contact number, a mobile, an emergency contact. People pick the country from a searchable list — which fills in the dialling code — and then type the number. This page covers what is special about Phone fields. The settings every field has are on Settings every field shares.

Add a Phone field

1. On the entity's **Structure** tab, drag **Phone** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the shared settings you need — for example **Required**, or **Unique** so two contacts can't share a number.
4. Select **Save**.

How a phone number behaves

- **Country first.** The country list shows each country's flag and dialling code, and you can search it by name. It starts on the United Arab Emirates.
- **Shown the same way everywhere.** In record details, tables and cards, numbers are shown in international format with spacing, such as *+971 50 123 4567*, whatever way people typed them.

Example. A *Visitor* entity has a *Mobile* Phone field set as **Main phone**, so the number appears on every visitor's card and can be used as their contact number.

Make it the record's main phone

Under **Display › Set Field As**, tick **Main phone** to make this the number that represents the record. A Phone field can also be the **Main title**, **Sub title** or **Caption**.

Settings it shares

A Phone field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Searchable**, **Queryable**, **Unique**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares.

To store several numbers for one record, make the field **Repeatable**.

> ### Go deeper
>
> For integrators: Field configuration reference [Integrate & Extend] — the phone formats and the default country setting

Reference field

Use a **Reference** field to link a record to another record: an asset request to the *Department* that pays for it, a task to its *Project*, an employee to their *Manager*. People pick from a list of records in another entity, or from a list you maintain. Linked records can then show each other's details, and Mapped fields can copy values across. This page covers what is special about Reference fields. The settings every field has are on Settings every field shares.

To pick a person, use User. To point at an application, service or page of the platform, use Application Reference. For relationships between entities more generally, see Relations.

Add a Reference field and choose its source

1. On the entity's **Structure** tab, drag **Reference** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Data Source**, choose where the options come from:
    - **Entity Source** — records of an entity. In **Choose data entity**, pick the entity. To offer records from more than one entity, select **Add source**.
    - **List Source** — the items of a list. In **Choose List Item**, pick the list.
4. Optionally, narrow an entity source to the records a saved query returns: pick the query in **Entries source**. If the query takes parameters, fill them in — for example, from another field on the form.
5. Select **Save**.

Example. An *Asset Request* has a *Department* Reference field whose source is the *Departments* entity, narrowed by the saved query *Active departments*, so closed departments can't be picked.

Build a cascading pick-list

A cascade narrows one Reference field's options by what was chosen in another.

1. In the dependent field's **Data Source**, on its entity source, turn on **Cascade**.
2. In **Cascade based on**, pick the other Reference field on this entity that drives the choice.
3. In **Relation**, pick the relation between the two entities that links them.
4. Select **Save**.

Example. A task has two Reference fields to users: *Direct manager* and *Employee*. With a cascade on *Employee* based on *Direct manager*, once a manager is chosen the *Employee* list shows only the people that manager manages — not everyone in the organisation.

If the panel says *There are no valid fields to enable the cascade*, add the other Reference field first.

Link records of the same entity without loops

When a Reference field points at its own entity — a parent category, an employee's manager, a parent task — turn on **Prohibit loops**. The platform then refuses a choice that would make a record its own ancestor.

Example. In a *Category* entity with a *Parent category* field, *Laptops* is under *Computers*. With **Prohibit loops** on, nobody can then set *Computers*'s parent to *Laptops*.

Show the choices as a tree

When the source entity is a hierarchy — it supports a tree structure, or it is this same entity with **Prohibit loops** on — a **Display as** section appears. In **Display format**, choose **Flat** for a plain list or **Tree** to show the options nested under their parents. **Flat** is the default.

Let people pick several records

Under **Value Settings**, turn on **Multiple** to let people choose more than one record — several affected assets, several approving departments. Optionally, set **Min** and **Max**.

A Reference field used to split a Kanban board into columns can't be made **Multiple** or **Repeatable**; choose another column field for the board first.

Keep the link up to date

Under **Display**, **Auto-resync on change** re-links the field automatically when data in the fields you list in **Fields causing resyncing** changes (leave it empty to react to any field). Choose **Preserve value** to keep the current choice, or **Allow overwrite** to replace it with the newly matched record.

Defaults and display roles

- Under **Default Value**, turn on **First value** to pre-select the first option, or tick **Advanced** to work the value out with an expression.
- Under **Display › Set Field As**, tick **Main reference** to make this link the one that represents the record. A Reference field can also be the **Main title**.

Settings it shares

A Reference field also offers **Lock Field**, **Hide Field**, **Required**, **Searchable**, **Unique**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares. A Reference field can't be **Sensitive** or **Encrypted**.

Good to know

- Filters on the source can use moving values such as the current user (**@me**) or their organization unit (**@myOrgUnit**), so each person sees the options that apply to them.
- A record that was linked before a filter changed keeps its link when it is edited, even if it would no longer be offered.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — data sources, pointers and parameterised queries

Sequence Field

Use a **Sequence Field** to number records automatically: 1000, 1001, 1002… for employees, or a running number per year for requests. The platform assigns the number when a record is created; nobody types it and nobody can change it. With a second field you can turn the number into a formatted reference such as *LC-2026-0001* that restarts every year. This page covers what is special about sequence fields. The settings every field has are on Settings every field shares.

Add a Sequence Field

1. On the entity's **Structure** tab, drag **Sequence Field** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Auto increment settings**, enter the **Starting value** (the first number) and **Increment by** (how much each new number adds).
4. Select **Save**.

Example. An *Employee* entity has an *Employee number* sequence starting at 1000 and incrementing by 1. The first employee gets 1000, the next 1001, and so on.

Keep separate counters

By default there is one counter for the whole entity. **Conditional Sequencing** keeps a separate counter for each value of one or more **keys** — one counter per year, per branch, or per branch and year.

1. Under **Auto increment settings**, turn on **Conditional Sequencing**.
2. Select **Add Key**.
3. In **Key**, give the key a name, such as *year* or *branch*.
4. In **Value**, write the expression that gives the key's value for a record — for example `new Date().getFullYear()` for the current year, or `value.branch` for the record's *Branch* field.
5. Add more keys if the counter should depend on several values.
6. Select **Save**.

Each new combination of key values starts its own counter at the **Starting value**: the first request of 2027 gets number 1 again, with no maintenance.

Decide the keys before real records exist. Once the counter has been used, the keys can no longer be changed.

Recipe: a yearly reference such as LC-2026-0001

A reference that combines a prefix, the year and a four-digit number that restarts every year is built from two fields: a sequence that counts per year, and a locked text field that composes the reference.

Step 1 — the yearly counter.

1. Add a **Sequence Field** named *Number in year*, and check that its **Normalized name** is `seq`.
2. Under **Auto increment settings**, set **Starting value** to 1 and **Increment by** to 1.

3. Turn on **Conditional Sequencing**, select **Add Key**, and enter the **Key** *year* with the **Value** `new Date().getFullYear()`.

Step 2 — the reference.

1. Add a **Text** field named *Reference*, and check that its **Normalized name** is `ref`.
2. Under **Field Attributes**, turn on **Lock Field**.
3. Under **Default Value**, tick **Advanced** and enter this in **Type expression**:

```
value.ref ? value.ref : (value.seq ? 'LC-' + new Date().getFullYear() + '-' + String(value.seq).padStart(4, '0') : null)
```

4. Set **Execution Triggers** to **On save**, then **Execution Scope** to **All entries** (choose the trigger first — changing it resets the scope).
5. Select **Save**.

Result: *LC-2026-0001*, *LC-2026-0002*, … and, from the first record of the new year, *LC-2027-0001*.

How the expression behaves:

- **An issued reference never changes.** The first part (`value.ref ? value.ref`) keeps the reference once it exists, so a record created in 2026 and edited in 2027 keeps *LC-2026-…*.
- **No broken references.** The second part leaves the field empty until the number exists, instead of producing *LC-2026-undefined*.
- **The reference appears on the next save.** The platform assigns the sequence number when a record is created, *after* calculated values are worked out. So a new record's *Reference* is empty at first and is filled in the next time the record is saved. This is also true for records created by integrations.

If the year in the reference must always match the counter's year, take both from the same date on the record (for example a *Requested on* field) instead of from today's date.

Good to know

- A Sequence Field is always locked, and it is not shown on the record's form — the number appears once the record is saved.
- It can never be **Required**: the platform fills it in, so a required check could never be satisfied. It also has no default value, and it can't be **Repeatable**.
- Numbers are unique automatically. With **Conditional Sequencing** they are unique within each key value (one *1* per year), which the counter itself guarantees.
- Under **Display › Set Field As**, tick **Main number** to make the number the one that represents the record.

Settings it shares

A Sequence Field also offers **Hide Field**, **Sensitive**, **Encrypt**, **Searchable** and **Display on table**. Each is described on Settings every field shares.

Go deeper

For integrators: Field configuration reference [Integrate & Extend] — the stored auto-increment and calculated-value settings for this recipe, the counter's behaviour, timing, and entries created through the API

Switch field

Use a **Switch** field when people choose between a few fixed answers that you label yourself, shown as a checkbox, a toggle or a set of radio buttons — for example *Priority: Normal / Urgent*, or *Delivery: Pick up / Deliver to desk*. This page covers what is special about Switch fields. The settings every field has are on Settings every field shares.

For a plain yes/no tick box, Boolean is simpler. For a longer list of options, colours, or more than one pick, use Category.

Add a Switch field

1. On the entity's **Structure** tab, drag **Switch** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Answers options**, choose the **Select Switch Type**:
    - **Checkbox** — a tick box.
    - **Toggle** — a sliding on/off switch.
    - **Radio** — one radio button per option; people pick one.
4. Under **Add options**, type an option's label, add its Arabic and English text with the translation icon, then select **Add**. Repeat for each option.
5. Drag options to reorder them, or delete one you no longer need.
6. Optionally, set a **Default Value** by picking one of the options.
7. Select **Save**.

Example. A *Delivery method* Switch of type **Radio** with the options *Pick up from IT desk* and *Deliver to my office*, defaulting to *Pick up from IT desk*.

Copy option id copies an option's internal identifier, for use in an expression or an integration that needs to refer to that option.

Settings it shares

A Switch field also offers a fixed or calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Repeatable**, **Custom Validation**, **Display on table**, and — as **Radio** — the display role **Main title**. Each is described on Settings every field shares.

A Switch field can't be **Encrypted**.

User field

Use a **User** field to link a record to a person in your organization: who asked for it, who is assigned to it, who must approve it. People pick from a list of users. This page covers what is special about User fields. The settings every field has are on Settings every field shares.

To link to a record of another entity, use Reference.

Add a User field and choose who can be picked

1. On the entity's **Structure** tab, drag **User** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Under **Data Source**, choose who appears in the list:
    - **All users** — everyone (the default). Optionally pick a saved query in **Entries source** to narrow the list.
    - **Group** — members of the groups you pick in **Select groups**. Tick **My groups** to offer the members of the current person's own groups.
    - **My organization unit** — the people in the current person's organization unit.
4. Select **Save**.

Example. A *Leave request* has an *Approver* User field with the source **My organization unit**, so each employee picks an approver from their own unit.

Fill in the current person automatically

To record who created a request, without letting them change it:

1. Under **Default Value**, tick **Advanced**.
2. In **Type expression**, enter `scope.me()` — the person filling in the form.
3. Under **Field Attributes**, turn on **Lock Field**.
4. Back in **Default Value**, set **Execution Triggers** to **On page load** and **Execution Scope** to **New entries**, so the person is filled in once, when the record is created.
5. Select **Save**.

The *Requested by* field now always shows the person who created the record.

To pre-select the first person the list offers instead, turn on **First value** under **Default Value**.

Let people pick several users

Under **Value Settings**, turn on **Multiple** to let people choose more than one person — for example, the *Reviewers* of a document.

Display roles

Under **Display › Set Field As**, tick **Main reference** to make this person the link that represents the record. A User field can also be the **Main title**. **Auto-resync on change** works as described for Reference fields.

Settings it shares

A User field also offers **Lock Field**, **Hide Field**, **Required**, **Searchable**, **Unique**, **Repeatable**, **Custom Validation** and **Display on table**. Each is described on Settings every field shares. A User field can't be **Sensitive** or **Encrypted**.

Good to know

In filters on a User field you can use moving values: **@me** (the person viewing the list) and **@myGroupsUsers** (the people in that person's groups). A *My requests* list, for example, filters on *Requested by* equals **@me**.

> ### Go deeper
>
> For integrators: Field configuration reference [Integrate & Extend] — user pointers

Video field

Use a **Video** field to let people upload a video: an installation walkthrough, a site inspection recording, a training clip. People can upload MP4, MOV, AVI, WMV, FLV, MKV and WebM files. This page covers what is special about Video fields. The settings every field has are on Settings every field shares.

Add a Video field

1. On the entity's **Structure** tab, drag **Video** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the limits below.
4. Select **Save**.

Limit what people upload

Under **Value Settings**:

- **Max file size** — the largest single file, in MB. It starts at 20 MB, which is also the most you can set.
- **Total files size** — when the field is **Multiple** or **Repeatable**, the most all the files together may add up to.

Files that are too large are refused as soon as people pick them, before anything is uploaded.

Let people upload several videos

Under **Value Settings**, turn on **Multiple** and, optionally, set **Min** and **Max** — the fewest and most videos per record.

Show fixed videos on the form

Added as a **Display only field**, a Video field shows a video you upload once — such as a how-to clip — and does not ask people for anything. Under **Read Only Settings**, **Display Size** sets how large it appears: **Full Width**, **Original Dimensions**, or a **Small**, **Medium** or **Large Thumbnail** (320, 640 or 1024 pixels wide).

Settings it shares

A Video field also offers **Lock Field**, **Hide Field**, **Required**, **Sensitive**, **Encrypt**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. A Video field has no default value.

Mapped Field

Use a **Mapped Field** to show, on this record, a value that lives on a linked record. When someone picks a *Department* in an asset request, a mapped field can fill in that department's *Cost centre*; when they pick an *Employee*, mapped fields can show the employee's *Job title* and *Phone*. You decide whether the value is copied once, keeps following the source, can be changed locally, or is kept in sync in both directions. This page covers what is special about mapped fields. The settings every field has are on Settings every field shares.

A mapped field always works through a Reference field on the same entity: add the Reference field first.

Add a mapped field

1. On the entity's **Structure** tab, make sure the entity has a Reference field to the record you want to copy from — for example *Department*.
2. Drag **Mapped Field** from the **Fields** list onto the canvas, then select the settings icon to open **Field Settings**.
3. Under **Basic settings**:
    - In **Reference Field**, pick the Reference field — *Department*.
    - In **Field Source**, pick the field on the linked record to show — *Cost centre*.
    - In **Update Mode**, choose how the value is kept (see below).
4. Name the field. Its type follows the source field.
5. Select **Save**.

Choose how the value is kept

Update Mode	What happens	Choose it when
Local Update (the default)	The value is filled in automatically and people can change it here; the change stays on this record and is not written back.	The source is a suggestion: a *Delivery address* that starts as the employee's office address but can be changed per request.
Never	The value is filled in the first time only and the field is locked. Later changes to the source are not copied.	You need a snapshot of the value at the time: the *Unit price* when the order was placed.
From Parent Only	The field is locked and follows the source: when the value changes on the linked record, it changes here too.	The value belongs to the linked record: the department's *Cost centre*.
2 way	The value can be changed in either place, and a change on one side is copied to the other.	Both records own the value together.

What each mode allows:

- With **Never** and **From Parent Only**, **Lock Field** is switched on and can't be turned off. With **Local Update** and **2 way**, locking is up to you.
- **2 way** is only offered when this field's **Required** setting matches the source field's.
- When the mapped field is inside a repeatable Block but its Reference field is not, only **Never** and **From Parent Only** are offered.

Find the linked record by typing

With **2 way**, when the source field is searchable, you can turn on **Look Up**. People can then type a value in the mapped field to find the linked record by it. In **Look Up Match**, choose how the typed value must match: **Full Match**, **Partial Match** or **Starts Match**.

Formatting and limits

Depending on the source field's type, a mapped field also offers that type's value format — for text, none, email address or URL; for numbers, general or percentage with a display format and precision — and **Min & Max** under **Value Settings** (a value range for numbers, a length range for text).

Settings it shares

A mapped field also offers **Lock Field** (see above), **Hide Field**, **Required**, **Searchable**, **Unique**, **Display on table**, and the display roles **Main title**, **Sub title** and **Caption**. Each is described on Settings every field shares.

A mapped field has no default value of its own, and it can't be **Multiple** or **Repeatable**. It takes **Sensitive** and **Encrypt** from its source field: if the source is protected, so is the mapped value.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — update modes as stored, and the rules the server enforces

SVG Icon field (internal)

SVG Icon is a field type the platform uses for its own entities and screens — it holds custom vector artwork. It is not in the **Fields** list of the entity designer, so you can't add it to an entity you build. You may still see it in an entity the platform provides.

To let people pick an icon, use an Icon field; for custom pictures, use an Image field.

Go deeper

For integrators: Internal field types [Integrate & Extend] — what this type stores and the settings it carries

Application Reference field

Use an **Application Reference** field to link a record to a part of the platform itself rather than to another record: an application (module), a service, a view, a page or a report. People pick it from the platform's navigation. When the record is shown, the link displays the target's name and the application it belongs to, and opens it. This page covers what is special about Application Reference fields. The settings every field has are on Settings every field shares.

To link to a record of an entity, use Reference.

Add an Application Reference field

1. On the entity's **Structure** tab, drag **Application Reference** from the **Fields** list onto the canvas.
2. Type the field's name on its row, then select the settings icon to open **Field Settings**.
3. Set the shared settings you need.
4. Select **Save**.

Example. A *Help article* entity has a *Related service* Application Reference field. Each article links to the service it explains, and readers open the service straight from the article.

Settings it shares

An Application Reference field also offers a calculated **Default Value**, **Lock Field**, **Hide Field**, **Required**, **Repeatable** and **Display on table**. Each is described on Settings every field shares. It can't be **Sensitive**, **Encrypted**, searchable or unique.

> ⓘ **Go deeper**
>
> For integrators: Field configuration reference [Integrate & Extend] — what is resolved when the value is displayed

Page Reference field (internal)

Page Reference is a field type the platform uses for its own entities and screens — it points to a page of the platform. It is not in the **Fields** list of the entity designer, so you can't add it to an entity you build. You may still see it in an entity the platform provides.

To link a record to a page, service, view or application, use an Application Reference field.

Go deeper

For integrators: Internal field types [Integrate & Extend] — what this type stores and the settings it carries

Data Query Selector field (internal)

Data Query Selector is a field type the platform uses for its own entities and screens — it points to a saved query, for the platform's own configuration screens. It is not in the **Fields** list of the entity designer, so you can't add it to an entity you build. You may still see it in an entity the platform provides.

To narrow the options of a link to another entity, use a saved query as the **Entries source** of a Reference field.

Go deeper

For integrators: Internal field types [Integrate & Extend] — what this type stores and the settings it carries

Number Enum field (internal)

Number Enum is a field type the platform uses for its own entities and screens — it offers a fixed list of numbered choices. It is not in the **Fields** list of the entity designer, so you can't add it to an entity you build. You may still see it in an entity the platform provides.

For a list of choices you define, use a Category or a Switch field.

Go deeper

For integrators: Internal field types [Integrate & Extend] — what this type stores and the settings it carries

JSON field (internal)

JSON is a field type the platform uses for its own entities and screens — it stores a free-form structured value, such as data returned by an integration. It is not in the **Fields** list of the entity designer, so you can't add it to an entity you build. You may still see it in an entity the platform provides.

To capture structured data yourself, use a Block — its fields can be searched and filtered.

Go deeper

For integrators: Internal field types [Integrate & Extend] — what this type stores and the settings it carries

Link entities

This guide is for **builders** whose application has more than one kind of record and needs them connected — a request that names an employee, a project that lists its tasks, an order whose lines show the product price. By the end you will be able to add a reference field, show linked records with a relation, make one pick-list depend on another, and copy values from a linked record with a mapped field.

How linking works

Take an *Asset Management* application with three entities: *Employees*, *Assets* and *Asset Requests*.

- A **reference field** on *Asset Requests* called *Requested by* points at one record in *Employees*. When people fill the form, they pick an employee from a list. The link stays correct even if the employee's name changes later, because the field stores which record was picked, not the text.
- A **relation** is the same link seen from the other side. On *Employees*, a relation *Requests* lists every asset request whose *Requested by* points at that employee. You define relations on the entity's **Relations** tab, and they power related-record lists, joiner entities and reactions that fire on linked records.
- A **mapped field** copies a value from the linked record into this one — for example the employee's *Department* shown on the request.

Relations also let you build hierarchies, such as an organization whose child records are departments, whose child records are teams.

Add a reference field

1. Open the entity's designer (**three dots** on the entity card › **Edit Entity**) and drag a **Reference** field onto the **Structure** tab.
2. Give it a name, for example *Requested by*.
3. In its data source settings, choose where the options come from — the records of an **entity** (optionally narrowed by a saved query or a view), or the items of a **list**.
4. Decide whether people may pick one record or several, and choose the **Display format**: **Flat** for a simple list, or **Tree** for a hierarchy.
5. In **Display as**, choose how the field appears in the form: **Select list**, **Select + Details** (the picked record's details shown under the list) or **Details + Search**.
6. Optionally narrow the options to the person filling the form, with filters such as *me*, *my groups*, *my roles* or *my organizational unit*.
7. Select **Save**.

The Reference field page describes every setting.

Make one pick-list depend on another

A **cascade** narrows a reference field's options by the value of another reference field on the same form. For example, after someone picks a *Building*, the *Room* list shows only rooms in that building.

1. Make sure the *Room* entity has a reference to *Building*, and that a relation links the two.
2. In the *Room* field's source settings, turn on **Cascade**.

3. In **Cascade based on**, choose the *Building* field, and in **Relation**, choose the relation that links rooms to buildings.

Prevent circular links

On a reference field that points back at its own entity — a *Manager* field on *Employees*, say — you can turn on the restriction against circular references, so a record can never end up referring to itself through a chain. Turning it on for an entity that already has records is a breaking change; see Change an entity safely.

Show linked records with a relation

1. Open the entity that should show the linked records — *Employees* in the example — and select its **Relations** tab. Both kinds of relation are listed there as cards: **Incoming** (other entities point at this one) and **Outgoing** (this entity's own reference fields point elsewhere).
2. Select **Add Relation**.
3. Enter the relation's name and icon, for example *Requests*.
4. In **Relation Type**, choose **Incoming**, then choose the entity on the other end (*Asset Requests*) and its reference field that points here (*Requested by*).
5. Save.

To change a relation, select its card, edit the details and select **Finish**. To remove it, select the delete icon next to it and confirm; this cannot be undone.

Outgoing relations are created for you from the entity's reference fields — you do not add them by hand. Relations the platform creates for its own entities are locked and cannot be edited or deleted.

Copy values with a mapped field

A mapped field shows a value from the record that a reference field points at. On *Asset Requests*, a mapped field *Department* can take the department of the employee picked in *Requested by*.

1. In the entity's designer, add a **Mapped** field.
2. Choose the reference field it follows (*Requested by*) and the **source field** on the linked entity (*Department*).
3. Choose the **Update Mode** (below), give the field a name, and select **Save**.

Choose an update mode

The update mode decides what happens when the linked record changes, and whether people can edit the copied value.

Update Mode	When the record is filled	Can people edit it?	When the linked record changes later
Local Update	Copied from the linked record	Yes; their edit stays on this record only	Not refreshed — unless someone picks a different linked record, which copies the new value
Never	Copied once, the first time	No	Not refreshed
From Parent Only	Copied from the linked record	No	Refreshed automatically
2 way	Copied from the linked record	Yes; the edit is also written back to the linked record	Refreshed automatically

Choose **From Parent Only** for values that must always match the source, such as a product's current price on a quote that is still open. Choose **Never** for a value that must stay as it was when the record was created, such as the price on a signed order. Choose **Local Update** when the value is a sensible starting point that people may adjust. Choose **2 way** only when both records should always carry the same value and either side may change it.

2 way is not offered when the mapped field and its source differ in whether they are required. Inside a repeatable block that is not the reference field's own block, only **Never** and **From Parent Only** are offered.

Good to know about mapped fields

- If the source field is sensitive, the mapped field is sensitive too, with the same people allowed to see it.
- A mapped field never becomes unique just because its source is.
- A field whose value is not updating is often doing what its mode says — check the update mode before reporting a problem.

Sidebar and shareable links

Two other kinds of link appear in the product:

- **Sidebar links** organise the navigation menu. The platform keeps links for modules, views and reports up to date automatically; you can also add your own entries — with a type, icon, name, parent, order and who may see them — through the Links entity. A link to a **report** opens only on the web, not in the mobile app.
- **System links** are shareable links to a view, page, report or module. Each can require sign-in or allow anonymous visitors, has a validity period, and can be disabled at any time. Anonymous links to public views add a human-verification step; use Preview mode to check what visitors will see before you share.

Troubleshooting

Problem	What to check
A reference field's list is empty	Its source entity has no records the person may see, or its query or cascade filters everything out.
A cascaded list does not narrow	**Cascade based on** and **Relation** must point at the parent field and the relation that links the two entities.
A mapped value does not change when the source changes	Its update mode is **Local Update** or **Never**.
A relation cannot be edited or deleted	It is a platform relation, or an outgoing relation that comes from a reference field — change the field instead.
A reference field cannot be deleted	A mapped field still depends on it. Delete or change the mapped field first.

Go deeper

For integrators: Relations, references and mapped fields: model and API [Integrate & Extend] — the relation model, the Relations and related-entries routes, composite relation ids, update-mode internals and navigation links

Virtual entities

This guide is for **builders** who need the same data seen in a different shape — another module's records offered in yours, related records combined into one list, or records assembled from several sources. By the end you will be able to choose the right kind of virtual entity, create it, and control what people may do through it.

How virtual entities work

A **virtual entity** has no records of its own to fill in. Its records are calculated from one or more **source** entities, and the platform keeps them up to date as the sources change. Once created, you use a virtual entity like any other: build views on it, point reference fields at it, filter it, and use it in queries, summaries and reports.

Kind	In **Data Entity Type**	What it gives you	Example
Proxy	**Virtual Proxy Entity**	One source entity, re-exposed as a separate entity — the same records, possibly fewer of them, under another name or in another module	The HR module's *Employees*, offered as *Staff directory* in a Facilities module
Joiner	**Joiner Virtual Entity**	A source entity combined with entities linked to it, one row carrying fields from all of them	Each *Asset Request* together with its requester's *Department* and the *Asset*'s *Category*
Aggregate	**Aggregate Entity**	Records assembled from several sources, with each field mapped from a source field, a fixed value or a calculation	A single *All open items* entity drawing from *Incidents*, *Requests* and *Changes*

Updates reach a virtual entity shortly after the source changes, not in the same instant — expect a brief delay.

Create a proxy entity

1. Open **Administration › Applications › Data Entities** and select **Add Data Entity**.
2. Enter the **Name**, **Normalized name**, icon and module as for any entity (see Create an entity).
3. In **Data Entity Type**, under **Virtual Entity**, choose **Virtual Proxy Entity**.
4. In **Choose Entity**, pick the source entity. You can also expand it and pick one of its **proxy views** — the view that decides which records and fields the proxy offers (see below). If you pick only the entity, the form tells you that a read-only proxy view will be created inside it for you, covering every field.
5. Save. The platform builds the proxy's records and keeps them in step with the source.

To offer only some of the source's records — for example only *active* employees — pick (or build) a proxy view on the source entity whose data comes from a query that selects just those records. A proxy view that shows the whole source offers every record.

Control what a proxy entity allows `Since 1.10`

A proxy entity is governed by its **proxy view** — a special view on the **source** entity. The proxy view sets the most the proxy may ever do: which fields it shows, which screens it offers, and whether people may **create**, **edit** or **archive** records through it, or start drafts.

- A new proxy is **read-only**. To let people add or change records through it, open the proxy view on the source entity and turn on the operations you want, with the fields each may change.

- Records created or changed through the proxy are really saved on the **source** entity, so the source's own rules, workflow and history apply. A change that would take a record outside the proxy view's scope is refused.
- Views you build on the proxy entity can show less than the proxy view allows, never more. Narrowing the proxy view later takes effect everywhere at once.
- A proxy view has no permissions or sharing of its own; access to the proxy is controlled by the proxy entity's permissions.
- Once a proxy is linked to a proxy view, it stays linked. You can switch it to another proxy view on the same source, but not remove the link.

Joiner and aggregate records cannot be created or archived through the virtual entity; change their sources instead.

Create a joiner entity

1. Start as above, and in **Data Entity Type** choose **Joiner Virtual Entity**.
2. In **Choose Entity**, pick the **primary** source — the entity each row starts from, such as *Asset Requests*.
3. Under **Additional Sources**, add each linked entity and the relation that joins it to the primary source (or to another additional source).
4. For each additional source, decide whether it is **Mandatory**. A mandatory source keeps only rows that have a match there; otherwise rows are kept even without a match.
5. Choose the fields the joiner shows, and save.

A joiner needs at least one field, and the links between the sources come from relations, so set those up first.

Create an aggregate entity

1. Start as above, and in **Data Entity Type** choose **Aggregate Entity**.
2. Pick the primary source, then use **Add source** for each additional source.
3. Design the aggregate's own fields on its **Structure** tab.
4. On the **Mapping** tab, for each source, say where each field's value comes from: a field of that source (**Source field**, **Map to field**), a fixed value, a calculated expression, or nothing.
5. If a category field uses different options in the source than in the aggregate — *Open* versus *In progress*, say — use **Map source category options** to pair them up.
6. Save.

Good to know

- **Workflows stay on the source.** A virtual entity has no workflow of its own. An aggregate or proxy record cannot be edited once its source record's workflow has finished, and a proxy record's fields lock where the source's current step locks them.
- **Field changes on the source reach the proxy.** When a source field becomes required or changes type, the proxy's copy of the field follows. Display settings on the proxy stay yours.
- **Archiving follows the source.** Archiving or restoring a source record archives or restores its virtual copies, and permanently deleting it removes them.
- **Admin operations.** Virtual entities can be moved to another module and deleted permanently like any other entity, but they cannot have custom indexes. Document templates work on them too.

- **Cross-module use.** Modules stay independent; a proxy is the recommended way to use one module's data in another.

Troubleshooting

Problem	What to check
People cannot add or edit records through a proxy	The proxy view on the source entity has those operations turned off — a new proxy is read-only.
A proxy shows fewer records than expected	The proxy view's query narrows them.
A proxy shows *This entity's source proxy view was not found*	The proxy view was deleted. Link the proxy to another proxy view on the same source.
A joiner has fewer rows than the primary source	An additional source is **Mandatory**, so rows without a match are left out.
An aggregate record shows no **Edit** action	Its source record's workflow has finished.
Changes in the source appear late	Virtual records are rebuilt shortly after each change; wait a moment and refresh.

Go deeper

For integrators: Virtual entities: model, API and internals [Integrate & Extend] — type options, capability ceilings, create/update routes, write-through, scoping and how virtual records are built

Data Views & Faces

This guide is for **builders** who decide how people work with an entity's records. By the end you will be able to create a view, choose how its records are shown (as a table, cards, a calendar, a kanban board, a timeline, a tree or counters), set up searching, filtering, totals and export, design the record's details screen and its create and edit forms, add a dashboard, and send people to the right place with redirection rules.

How a view works

A **view** shows the records of one entity. People never see an entity directly — they always work through a view, and each view decides what they can see and do. An entity can have many views, each for a different audience or task.

A view is made of screens that you switch on and design one by one:

- **Entries list** — the list of records, shown through one or more **faces**.
- **Entries details** — one record: its fields, discussion, history, workflow and related records.
- **Forms** — creating, editing and archiving records.
- **Dashboard** — summaries and charts for the entity.

A **face** is one way to show the list. A view can offer several faces; one is the **Default** face it opens on, and people switch to the others while they work.

Example: a *Leave Requests* entity has a *My requests* view (a table and a calendar, people see only their own requests, they can create and edit) and a *Team board* view for managers (a kanban board split by status, no create form, a dashboard of requests per month).

> **Proxy views · Since 1.10**
>
> A **Proxy View** is a special kind of view that controls what a Proxy virtual entity may read and write. It has only a few settings (identity, source, fields and the create, edit and archive switches), and a view's kind cannot be changed once saved. The rest of this page describes ordinary views.

A view is also what a **Grafana** dashboard reads from `Since 1.6`: to show an entity's data in Grafana, create a view for it first. See Grafana Dashboards [Integrate & Extend].

Create a view

1. Open the entity: **Administration › Applications › Data Entities**, then select the entity.
2. Select the **Views** tab, then **New View** (or **Add View** if there are none).
3. Under **View information**, enter the **Name**, **Icon** and an optional **Description**.
4. Under **Type & source**:
    - **View type** — **Normal View** for a view you design. (Service views are created by the platform when you publish a service.)
    - **Query** — leave it empty to show all records, or choose a saved query to show only matching records — for example *my requests*. If the query asks for parameters, **Query parameters** appears so you can supply them.

    - **Fields** — the fields this view may use anywhere: in faces, details and forms. A field you do not add here cannot be shown.
5. Switch on and design the screens you need — the sections below.
6. Check your work in the **Preview** panel beside the settings. See Preview Mode.
7. **Save** the view, then publish it from the entity's **Views** tab (**Publish View**). An unpublished view is not available to anyone.

A view cannot be published until its permissions are set — who may open it and what they may do. See Permissions [Administration › Permissions & access].

Choose the faces

Turn on **Entries list**, then, under **Displays**, switch on each face you want and mark one as **Default**.

Face (in the designer)	Best for	Mobile app
Table view	Dense data and admin work: columns, inline edit, quick add, bulk actions.	Yes
Card view	Compact cards or a gallery.	Yes
Calendar view	Records with dates: month, week, day and list layouts.	Yes
Kanban view	Status-driven work: cards in columns, dragged from column to column.	Yes
Timeline view	Posts and news: pinned first, then by date.	Yes
Tree view	Hierarchies such as an org chart.	Yes
Counter view (under **Other options**)	A compact card list with counters.	No — use another face on mobile

Each face shows fields in **slots** — **Main title**, **Sub title**, **Image/file**, **Date** and so on. Only fields you added under **Fields** can fill a slot.

Table view

- **Table columns** — the fields shown as columns; choose at least one. A field must allow being shown in a table (a setting on the field) to be a column.
- **Limit displayed values** — how many values a cell shows for a multi-value or repeated field.
- Whether people may reorder columns and choose which columns they see.

Under **Behavior**, **Open entry in** decides what clicking a row does: **System Default Behavior**, **View Entry Details** or **Edit Entry** `Since 1.6`.

Card view

Map **Main title** (required), **Sub title** and **Image/file**. **Metadata** shows details such as the creation and update dates. On mobile, people can switch between a card and a gallery layout.

Calendar view

Map **Main title** (required), **Sub title**, **Image/file**, and the **Date** fields to plot (required) — each must be a date or date-and-time field.

Kanban view

Map **Main title** (required) and **Category** (required) — the field whose values become the board's columns. Optionally map **Sub title**, **Image/file**, **Main reference**, **Sub category**, **Date**, **Main number** and **Metadata**.

The column field may sit inside a block, even a repeatable one `Since 1.8`. Dragging a card to another column changes its value; reordering cards within a column needs **Reorderable** (under **Other options**).

Column field and columns `Since 1.10`

Which fields can split a board. A single-value **category** field, or a single-value **reference** field — to another entity's records or to a list. A multi-value or repeatable field cannot be a column field, nor can a reference to the view's own entity. Once a kanban view uses a reference field, that field cannot be made multi-value or repeatable until you change the view.

Which values become columns. Under **Kanban columns**:

Option	Columns shown
All columns	Every current value — and any value added later.
Used columns	Only values at least one card has. Reference fields only; it is the only choice when the field's options come from a query with parameters.
Or select specific columns	The values you tick.

Turn on **Allow users to change this setting** to let people pick their own columns from the board's view settings (for their session only).

Good to know:

- Cards with no value sit in a **Not Set** column.
- A board shows at most **20 columns**; beyond that people see *"Some columns are hidden!"* and should use another face.
- A column whose record was deleted stays, marked **Deleted record**, so its cards remain reachable.

When a card cannot move `Since 1.10`

On a board split by a reference field, a card is refused — it snaps back with a message — when the target column is not one of the field's allowed values, when the target record is archived, or when the card's own field would not offer that value (for example because the field is filtered by another field of the same record). Moving a card to **Not Set**, or within its own column, is always allowed. If someone else edited the record at the same moment, the move fails and the person should reload.

Timeline view

The entity needs a timeline first. Options: **Publish post** lets people with access to the view publish or unpublish posts; **Pin post** lets them pin posts to the top.

Tree view

The entity needs a reference field that points to the same entity, with loop prevention turned on — for example *Reports to* on an *Employee* entity.

- **Tree reference** (required) — that field.
- **Main title** (required), **Sub title**, **Image/file** and **Main reference**.
- **Track parents line** `Since 1.2` — show each record's line of parents up to the top.

A tree view must show all records or a query (not a list), and cannot be combined with **Filter first**. When people filter only by status (archived, obsolete…), the tree opens up to show matches deep in the hierarchy `Since 1.8`.

Counter view

Map **Main title** (required), **Sub title**, **Image/file** and **Date**. The counter view is not available in the mobile app.

Design the entries list

Search and filters

Under **Search & filter**, choose the **Search & filter style** — **Search**, **Filter**, or **Search & filter** — then which extra filters people get:

- **Metadata filter** (created by, created at, updated by, updated at, entry ID) and **Details metadata filters**.
- **Current user filter** and **Group's current user filter**.
- **Current step filters** and **Previous step filters** — the record's workflow step, its owner, when it arrived, how long it took.
- **Archived entries**, **Obsolete entries**, **Invalid entries** and **Entries invalid requirements** — let people find records in those states.
- **Save query** — people can save their filters.

Sorting

Set the default order with **Sort** › **Add Sorting** (record fields or metadata, ascending or descending). **Change the sorting** lets people change it.

Quick filters `Since 1.10`

Quick filters are one-click filters built from the entity's quick queries. A view offers only the ones you list under **Quick filters**. For each one:

- **Placement** — **In status list** (inside the list's status menu, the default) or **Standalone toggle** (its own switch next to it).
- **Default** — applied when the list opens. Several can be on by default. A quick filter with a parameter cannot be a default, because it asks for its value every time it is turned on.
- Drag the rows to set the order.

People see each quick query's **Quick filter name**. Quick queries can also be gathered into **groups** on the entity's **Queries** tab (see Queries); a group can be shown in the status menu or as a **Standalone dropdown**, and turning a group on applies all its members. The mobile app follows the same settings.

Totals and grouping `Since 1.10`

- **Aggregators** — tick the entity's aggregates (totals, averages…) this view should show. A view shows **only the ones you tick**: a new aggregate added to the entity does not appear until you tick it here. A total is drawn only under a column the view displays — if an expected total is missing, check the view's columns first.
- **Grouping** — **Add Group By** to group records by one or more of the view's fields (drag to order: the first is the outer group). **Collapse groups by default** and **User change grouping** are off unless you turn them on.

Totals cover everything the list matches, including its filters and search. How aggregates are defined is in Entity Aggregates & Entry Grouping.

Filter first

With **Filter first**, the list stays empty until the person fills in a filter form — useful for large entities or public look-ups. Choose up to four fields (**First filter** to **Fourth filter**) and the **Query type**; **Auto fill for add new entry** copies those values into a new record. Anonymous visitors are asked to prove they are human. **Filter first** cannot be combined with the tree view or the dashboard. When the view's query has parameters, filter first is switched on for you and cannot be turned off.

Classification styles and other options

- **Entry classification styles** — colour records by a classification field (**Background color**, **Border color**), choose the style applied on open, and **Allow Changing** so people can switch between the available styles.
- **Other options** — **Create list** (people build their own lists of records), **Reorderable** (drag records to reorder them; table, card and kanban faces), **Allow export** (see below) and **Counter view**.

Export

People can export a view's records in three ways:

1. **Excel export** of the view's records, with the view's columns (web and mobile).
2. **Export templates** — formatted Word, PDF or Excel documents you design; see Export templates.
3. **Export Entries** — the quick export described below.

CSV is available only from **Export Entries**. Sensitive fields follow each person's access in every export.

Export Entries (quick export) `Since 1.10`

Turn on **Allow export** to add an **Export Entries** button to the list (it only works while **Entries list** is on). People choose Excel or CSV, and the file holds **every record matching the list as they see it** — search, filters, quick filters, filter first and grouping order — in the table's current columns.

- Up to 10,000 records: the file downloads straight away.
- More, for a signed-in person: the export runs in the background and they are notified when the file is ready (up to 100,000 rows).
- More, for an anonymous visitor: only the first 10,000 can be exported.

The file holds records only — no group headings or totals. The mobile app has no quick export. Administrators have the same export on the entity page (**Export Entries**).

Design the record's details

Turn on **Entries details** (it needs **Entries list**: people open a record from the list) and, in **Choose fields**, pick the fields to show. Then add the panels you need:

- **Default sidebar tab** — which side tab opens first; **Quick pagination through entries** lets people step to the next record without going back to the list.
- **Relations** — show linked records. For each relation choose **Display as** (**Tab**, **Sub View** or **Field**), the **View** to show them with, and **Open entry as** (**Sidebar view**, **Full view**, **New tab** or **New window**).

- **Discussion** — comments on the record: **Message posting**, **Mentions** (and the **Mentions list scope**: **All Users**, **Based On View Permission** or a **Custom Mention** list), **Attachments** with the allowed **Attachments type** and **Max size** (about 1 MB by default), **Workflow comments**, and **Limit posted messages to this view**.
- **Entry history** — what people see of the record's history; see Choose what a view shows.
- **Workflow path** and **Workflow history** — the record's workflow, and **Display step owner** (**All** or **Owned by User**).
- **Emails** — **UI Integration Emails** and/or **Background Integration Emails**, for **All emails**, **Sent by user** or **Received by user**.
- **Information** — a tab with the record's **Meta data** (created and updated, by whom) and **Attachments**.

Which workflow tab opens first `Since 1.8`

When both **Workflow path** and **Workflow history** are on, **Default workflow view** chooses which one the workflow tab opens on. It applies only when **Workflow history** is on; otherwise the tab always opens on the path. Web and mobile follow the same setting. How people use the tab is in Executing workflow actions [Using OSOS › Acting on Workflow Steps].

Design the forms

Turn on the operations people may perform, and for each pick its fields in **Choose fields** (at least one) and what happens **After Save** — **Show Message**, **Go to entries list**, **Reset Form** (for another record), **Go to view entry details**, **Open in Edit Mode** or **Go to URL**.

Operation	Lets people…
Creating new entries	Add records. **Draft entries** lets them save unfinished drafts. **Quick entry creation** adds a record straight from the table.
Entry duplication	Copy a record from the table.
Editing entries	Change records. **Inline edit** edits in the table; **Workflow actions** lets people act on the workflow while editing; **Sidebar in edit** keeps the side panel open.
Archive entries	Archive records.
Allow invalid entries	Save a record that does not yet meet every rule.
Ask user for	Fill in fields that a button's custom action asks for.

For an entity with a workflow, **Administrative privileges** let selected people act as administrators in this view: **Edit entry as admin**, **View entry as admin**, **Execute action on behalf step owner**, **Close workflow**, **Move to step**, **Return to previous step**, **Reset workflow**, and **Change step owner temporary** or **permanently**.

Reorder repeated values `Since 1.8`

Repeatable reorder is set per operation — so a field can be reorderable on the create form and fixed on the edit form. It becomes available once the operation's fields include a repeatable field or repeatable block. People then get a drag handle on each repetition (when there are at least two) on the web and in the mobile app.

Add and edit category options from the form `Since 1.10`

People can add or rename a category field's options while they fill in a record, without asking an administrator. Three settings must line up:

1. On the entity, the category field has **Allow users to manage options**.
2. In the view, the create or edit operation has **Add category options** and/or **Edit category options** (both off by default), and the field is editable there.
3. **Link create and edit card settings** (on by default) keeps the create and edit forms in step.

The new options are saved with the record. If someone else changed the options in the meantime, the save is refused and the person reloads and tries again.

A category field can also hold several values. A multi-value category field cannot be a kanban column or sub-category, a default sort, a dashboard group-by, a timeline category or the entity's main category; see Multi-value category fields.

Add a dashboard

Turn on **Dashboard** to show the entity's summaries in the view; you can make it the view's **Default landing screen**.

- **Summaries to show** — **Show all summaries**, or **Show only summaries that match the view source**.
- **Apply view's source query to summaries that have all entries as a source** and **Append view's source query to summaries that have different query as a source** — narrow the summaries to the view's own records. Use with care: a summary whose source does not fit the view can give misleading figures.
- **Dashboard overview** and **Click to browse** — clicking a chart opens the matching records (needs **Entries list**).

The dashboard cannot be combined with **Filter first**. Summaries themselves are described in Dashboards & summaries.

Buttons, appearance and behaviour

- **Buttons** — choose which buttons the view shows and drag them into order; see Show the button in a view. **Buttons display layout** is **Default**, **Horizontal** or **Vertical** `Since 1.8`. With **Horizontal** or **Vertical**, **Apply to workflow actions only** lays out only the workflow actions of the current step and moves the other buttons under a **More** menu.
- **Pin to sidebar** (under **Appearance**) lists the view in the portal's sidebar.
- **Display options** — **Transparent background**, **Full width page**, **Content padding**, **Page header**, **Contained layout**, a **QR code** for each record (pointing at its link), and a **Swagger document** that describes the view for developers (see Generating a client [Integrate & Extend]).
- **Behavior** — **Default landing screen** (the list, the create form, a record's details or the dashboard) `Since 1.4`, and **Open entry in** (above).

The platform's own buttons

View, **Edit**, **Create**, **Duplicate** and **Archive** appear only when the matching screen or operation is on. If you turn an operation off, its button is removed from the view automatically when you save `Since 1.8`. **Export Entries** appears with **Allow export**. Your own buttons and export templates are never removed this way.

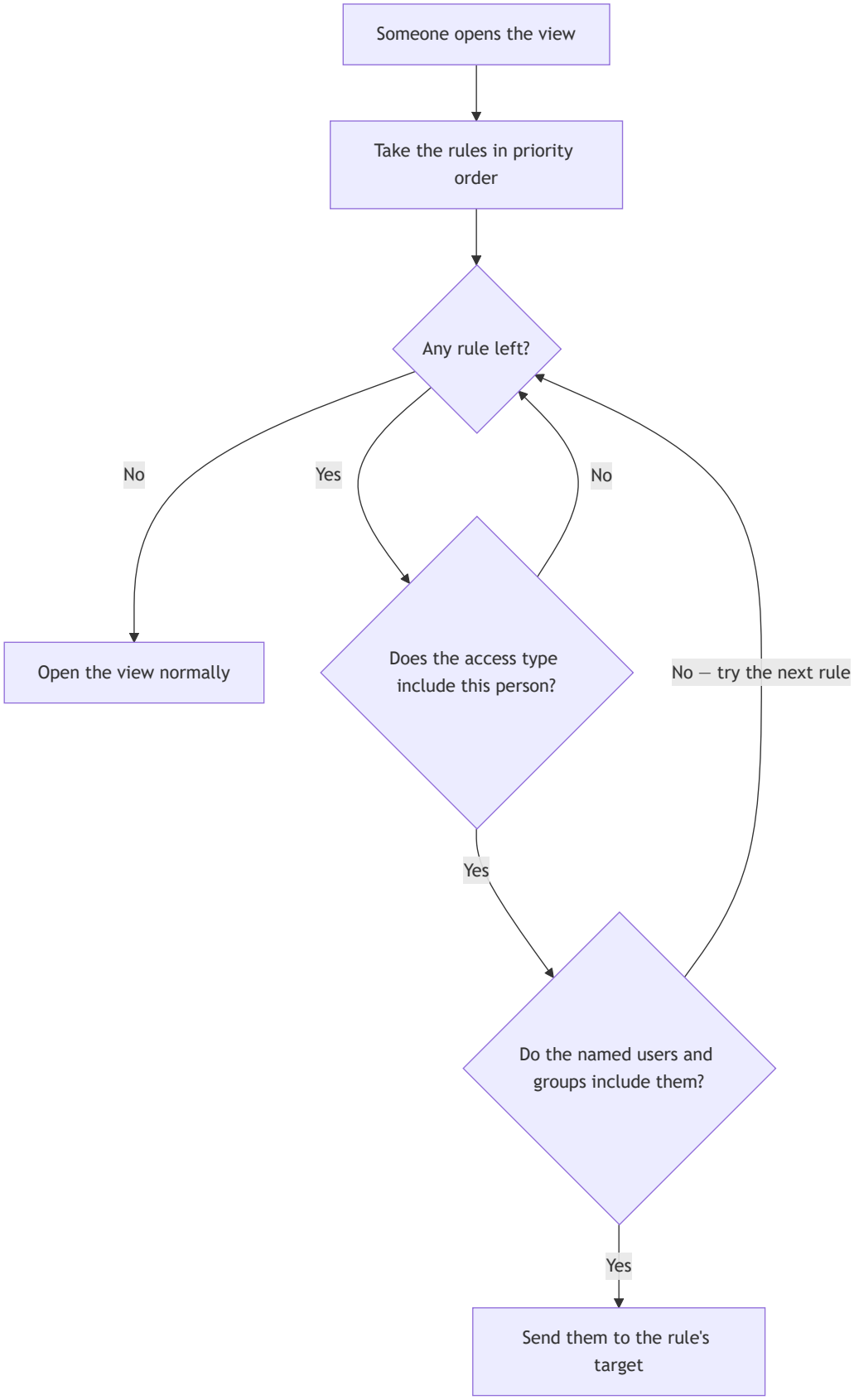
Redirection rules

Redirection rules send someone straight to another place — a web address, another view, a page or a report — instead of opening this view. Use them, for example, to send managers to a team view and everyone else to *My requests*.

Each rule has:

- an **Access type** — **Anonymous** (not signed in), **Authenticated** (signed in), **Any permission or all** (everyone) or **Restricted** (only the users and groups you name);
- optionally, the users and groups it applies to;
- an **Application type** — **Static link**, **Gallery** (a view), **Page** or **Report** — and the target;
- a **Priority** — rules are checked from the lowest number up, and the first rule that matches wins.

A rule matches only when **both** its access type and its named users or groups include the person. A **Restricted** rule that names nobody matches nobody. If no rule matches, the view opens normally.



Rules are written in the **Redirection rules** box; the designer shows an **Example of a redirection rule** to start from. Pages use the same rules — see Default pages and redirection.

Styling, help and scripts

Custom styles

Under **Custom style**, add CSS classes to the **View style**, **Fields style**, **Dashboard style**, **Workflow step style** and **Button dialog style**, and theme overrides for the mobile app. This is separate from classification styles, which colour records by a field. Check custom styles in light and dark themes and on a phone before you publish.

Help for people using the view

Under **Help & documentation**, write help in markdown for the view as a whole and for each screen — **Entries list**, **Entry details**, **Create entry**, **Edit entry**, **Dashboard**, **Ask user** — and per field (**Fields help & documentation**). It appears in a help panel beside the view. Where you write no help here, the field's own help is shown.

Scripts

On the **Web settings** tab you can add JavaScript that runs in people's browsers when they open, view or edit a record — first the view's script, then the entity's.

> ⚠ **Caution**
>
> A script runs as trusted code in the browser of everyone who uses the view. Only let trusted administrators write scripts.

When the same view appears more than once on a page, each copy's script affects only its own copy `Since 1.8`.

Good to know

- **Two people saving at once.** If someone else saved the view after you opened it, your save is refused; reload and make your change again `Since 1.10`.
- If **Editing entries** is on, at least one operation must be allowed.
- A view that neither anonymous visitors nor signed-in people may open is valid (a Proxy View is stored that way) — it simply cannot be opened.
- When a view grants only some columns of a repeatable block, saving a record keeps the values of the columns the view does not show `Since 1.4`.

Views installed from a package `Since 1.10`

A view that came with an installed package [Integrate & Extend › Packaging & distribution] is read-only. You can still publish or hide it, rename it, change its description and icon, set its order and change who can open it — and those changes survive an update of the package. Relation settings and scripts stay editable. To change anything else, fork the installation; see Marketplace and store [Integrate & Extend › Packages, Marketplace & the In-App Store].

A view on the *Layout builder* display template links to shared layouts rather than copying them, so a layout change reaches every view that uses it. No released client uses this yet.

Troubleshooting

Problem	What to check
A field cannot be chosen for a face, a column or the details	Add it under **Fields** first. For a table column, the field must also allow being shown in a table.
The kanban board shows no columns	Is **Category** mapped, and does **Kanban columns** include values that exist? With **Used columns**, a column appears only once a record has that value.
"Some columns are hidden!"	The board has more than 20 columns. Pick **Or select specific columns**, or use another face.
A card snaps back when dragged	The target value is not allowed for that record, or its record is archived. To reorder within a column, turn on **Reorderable**.
"You can't enable filter first and dashboard together"	Choose one of the two.
An expected total is missing	Tick the aggregate under **Aggregators**, and show its field as a column.
A quick filter cannot be made a default	It has a parameter; people must enter its value each time.
The counter view is empty on mobile	The counter view is web-only. Use another face on mobile.
People are sent somewhere else	A redirection rule matches them. Check the rules' priority and audience.
"Sorry, you can't publish this view, until configuring the view permission"	Set the view's permissions first.

Go deeper

For integrators: Data views API reference [Integrate & Extend] — kanban routes, column and move errors, quick-filter and export routes and limits, view save conflicts and refusals, installed-view writes, layout references, and the version notes · Field paths in view criteria: Criteria and field-path dialects [Integrate & Extend]

Pages & Widgets

This guide is for **builders** who want to put information in front of people on one screen: a team home page, a services portal, a dashboard of open requests. By the end you will be able to create a page, lay out widgets on it, decide who sees the page and each widget, make it the page people land on, publish it, and check it on a phone.

How pages work

A **page** is a screen you compose from **widgets** — blocks of text, links, images, documents, a summary chart, a list of records from a view, a timeline of posts. Unlike a view, which shows the records of one entity, a page can bring together content from many entities and modules.

Widgets sit on a **12-column grid**: a widget is 4, 6, 8 or 12 columns wide, and widgets fill rows from left to right. **Section** and **Column** widgets group other widgets together.

Example: an *IT Services* home page has a **Title** and a **Text** welcome, a **Summary** widget with *open requests by department* as a column chart, a **View** widget listing the person's latest requests, and a **Link list** to the request forms. It is the default page for signed-in employees, and its **TimeLine** widget shows IT announcements.

Create a page

1. Open the page list — either **Administration › Applications › Pages**, or a module's **Pages** tab for a page that belongs to that module.
2. Select **New Page** (or **Add Page** if there are none yet). The page opens with two tabs, **Details** and **Content**.
3. On **Details**, fill in **Page Information** — see Page settings.
4. On **Content**, add widgets — see Add widgets.
5. **Save** the page, then publish it.

A home page is created automatically for each new module and space.

Page settings

On the **Details** tab:

- **Page Information** — the **Name** (in each language), the **Icon**, an optional **Description**, and the **Page Type**:
  - **Normal** — a page you build from widgets.
  - **Embedded** — a page that shows another web page; enter its **URL**. An embedded page has no **Content** tab.
- **Default Page for** — the spaces in which this page is the default page; see Default pages.
- **Sidebar Configurations** — turn on **Pin to Sidebar** to list the page in the portal's sidebar, and choose **Display Sidebar**: **Preserve** (leave the sidebar as it was), **Open** or **Close** when the page opens.
- **Appearance** — **Transparent background**, **Full width page**, **Content padding**, **Page header** (show the page title) and **Contained layout**. Turn on **Allow Exporting as PDF** to let people download the page as a PDF.
- **General Settings** — **Auto Refresh** reloads the page's data at an interval you set.
- **Access Permissions** — who may open the page; see Who can see the page.

- **Additional CSS** — your own styling for this page. **Mobile theme override (JSON)** adjusts its colours in the mobile app.
- **Help & documentation** — help text people can open from the page.

Add widgets

1. Open the **Content** tab. The **Widgets** panel lists every widget you can use; search it if the list is long.
2. Click a widget to add it to the page, or drag it to where it belongs. **Drop widget here** marks the places it can go.
3. Fill in the widget's settings and select **DONE**. Every widget has:
    - **General** — an optional **Title**, **Description** and **Icon**; **Custom Visibility** to choose who sees this widget (see Show a widget to some people only); and **Custom CSS Classes** for styling.
    - **Content** — the settings of that kind of widget (see the widget list).
4. Arrange the layout:
    - **Edit width** — 4, 6, 8 or 12 columns.
    - **Vertical Alignment** — **Stretch**, **Top**, **Middle** or **Bottom**.
    - **New row** — start the widget on a new row.
    - **Edit height** — for text and other widgets whose height you can set.
    - Drag a widget to move it, or use **Move to** to put it in a section or column; **Ungroup** takes widgets out of one.
5. From a widget's actions you can also **Edit Widget**, **Duplicate Widget** or **Delete Widget**.
6. **Save** the page.

The widgets

Widget	Shows	Main settings
Section, **Column**	Containers that group other widgets.	—
Title	A heading.	**Heading 1** to **Heading 6**.
Text	A block of text.	**Text type**: **Multiline Text** or **Multiline Markdown Text**.
Link, **Link list**	One link, or a list of links.	**External** (a web address) or **Internal** — a **System View**, a **Module View** or a **Data View**. Links can fill in values automatically.
Media, **Media List**	An image or video, or a gallery.	Upload the files.
Document, **Document List**	A document to download, or a list of them.	Upload the files.
Summary	A chart or figure from a summary.	**Choose module**, **Choose entity**, **Choose summary**.
View	Records from a view.	The view, its **Face**, the **Count of entries** (3, 5 or 10, default 5 — or **Unlimited** for a face with paging and search), **Search**, and which **Tags** and **Button**s to show.
TimeLine	Posts from a timeline.	**Source**, **Pagination Type**, **Posts Per Page**, **Display Pinned Posts**.
Navigation	A grid of shortcuts to pages, views and reports.	**Columns**, **Height** (**Auto Height** or a fixed number of rows), **Grouping** (**No Grouping**, **Group by Module**, **Group by Type**) and a **Filter** by type (**Pages**, **Data Entity Views**, **Reports**).

Your platform may offer more widgets — quick links, sliders, QR codes, counters and others — if optional widget packs are installed. Which ones you see depends on your installation; adding a new kind of widget is a development task.

The View widget

A **View** widget follows the view's own design: its faces, sorting and paging. With the **Kanban** face it shows the board with the view's columns and a *load more* per column. Sorting a column of its table reorders the records, and clearing the sort returns to the view's default sorting.

Show a widget to some people only

By default a widget is seen by everyone who can open the page. To narrow it:

1. Open the widget's settings and, under **General**, turn on **Custom Visibility**.
2. Choose who sees it — anonymous visitors, signed-in people, or selected users and groups. A widget can only be narrower than its page: it cannot show to anonymous visitors if the page does not.

If you later change the page's own access, the platform warns that widget visibility customizations will be reset.

Who can see the page

Under **Access Permissions**:

- **Authenticated** — everyone who signs in can open the page.
- Selected users and groups (**Select Users**, **Select groups**) — only they can open it. A page limited this way cannot also be a default page.
- **Anonymous** — anyone can open the page through its share link, without signing in. You can add **Human Verification** to stop automated visitors.

Anonymous viewing and limiting access to selected people cannot be combined.

Default pages and redirection

When a module has more than one page, you can make one of them the default: under the page's **Access Permissions**, turn on **Default Page** for anonymous visitors, for signed-in people, or both. The page list shows it as the **Default View**.

- An **anonymous** default page is where visitors land before signing in or after signing out.
- An **authenticated** default page opens straight after signing in.

For more control, **Redirection rules** send people to a different page depending on who they are — for example managers to a *Team overview* page. Rules are checked in order and the first rule that matches wins. They are written in the same format as for views; see Redirection rules.

Check it on a phone `Since 1.8`

On the **Content** tab, select **Show preview** to open a phone frame beside the canvas. It runs the real mobile app and follows each change you make, without saving — use it to check widths, order and alignment on a narrow screen. It shows your **real records**, not sample data. See Preview a page on a phone.

Publish and export

A saved page is not visible to anyone until it is published:

1. In the page list, open the page's actions and select **Publish**, then confirm.
2. To take it down again, select **Unpublish**.

People reach a published page through its link, the sidebar (if **Pin to Sidebar** is on) or a **Navigation** or **Link** widget. If **Allow Exporting as PDF** is on, they can download the page as a PDF, in portrait or landscape. The PDF is made in the browser, so very complex pages may not export perfectly.

Pages on mobile

Pages appear in the mobile app with their widgets in the same order as on the web. **Reports are web-only**: a report link can appear in the sidebar, but reports do not open in the mobile app.

Pages installed from a package `Since 1.10`

A page that came with an installed package [Integrate & Extend › Packaging & distribution] is read-only. You can still publish or unpublish it, rename it, change its description and icon, set its order in lists, and change who can see it — and those changes are kept when the package is updated. A page that does not belong to a module (a *global* page) can itself be added to a package you build.

Good to know

- If two people save the same page at the same time, the second save is refused and that person must reload the page.
- An entity's views can also be pinned to the sidebar; modules with nothing visible in them do not appear there.
- A link in the sidebar is shown only to the people the page or view allows.

Troubleshooting

Problem	What to check
People cannot see the page	Is it published? Do its **Access Permissions** include them?
A widget is missing for some people	The widget's **Custom Visibility**, and the access of the view or summary it shows.
"Anonymous viewing and limiting access are not possible"	A page cannot be both anonymous and limited to selected users. Choose one.
The **Content** tab is disabled	The **Page Type** is **Embedded**.
The page does not appear in the sidebar	Turn on **Pin to Sidebar**, publish the page, and check its access.

Go deeper

For integrators: Pages, sidebar and share links API reference [Integrate & Extend] — the stored page and widget model, validation, the page endpoints and concurrency, installed and global pages in packages, the sidebar link tree and its API, and anonymous share tokens

Queries & filters

This guide is for **builders** who want people to see exactly the records that matter to them — *my open tasks*, *requests from my department this month*, *invoices over 10,000*. By the end you will be able to build a saved query on an entity, make it follow the person using it or today's date, ask for a value when it runs, offer it as a one-click quick filter or a coloured tag, and reuse it in views, reports, reference fields and dashboards.

How a query works

A **query** is a saved filter over the records of one entity. It is made of **conditions**, and each condition has three parts:

Part	What it is	Example
Field	What to test	*Status*
Operator	How to compare	**=**
Value	What to compare with	*Open*

Conditions are combined in a group: **All conditions are met** (every condition must hold) or **Some conditions are met** (one is enough). A group can contain further groups, so you can write "*Status* = Open **and** (*Priority* = High **or** *Due date* < today)".

Take a **Tasks** entity. *My open tasks* is two conditions — *Assigned to* **= @me** and *Status* **=** *Open* — under **All conditions are met**. Because **@me** stands for whoever is looking, one saved query shows each person their own tasks.

Create a query

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Queries** tab. It has three sub-tabs: **Queries**, **Tag groups** and **Quick filter groups**.
3. On **Queries**, select **+** (**New Query**).
4. Build the conditions — see Build the conditions. Select **Condition** to add a condition and **Group** to add a nested group; **Clear** starts again.
5. Under **Query Settings**, enter the **Query Name** (in each language, 3 to 128 characters).
6. Optionally tick **Quick Filter** to offer it as a quick filter (see Quick filters), or **Save Tag** to colour matching records (see Tags).
7. Select **Save Query**.

In the list, each query is marked **Normal Filter**, **Quick Filter** or **Parameterized**. Select **+** on one of those group headings to start a new query of that kind.

`Since 1.10` the query dialog always saves. To try out conditions without saving them, use the **Filter** panel on the entity's entries list instead.

Build the conditions

The field

You can test:

- **The record's own fields** — for example *Task title* or *Assigned to*.
- **The record's details** — *Created by*, *Creation date*, *Updated by*, *Update date*.
- **Fields of a related record** — pick a reference field and drill into the record it points to, for example *Department › Region = West*. Each level adds work, so keep chains short.
- **Workflow details** — on an entity with a workflow, the current step, its owner, previous steps and their owners. This is how you filter for "open" requests: *current step* is not the final step.
- **Calculated (virtual) fields** — filtered and sorted on their calculated value. If the calculation fails for a record, the list still loads and a warning is shown.

The contents of a **JSON** field cannot be filtered or searched. If you need to filter on a piece of data, make it a real field.

Protected fields in a condition `Since 1.10`

- An **encrypted** field cannot be used in a condition at all — its stored value is scrambled, so nothing could ever match.
- A **sensitive** field can be used in a query you build here in Administration, even if you may not read its values. People using the query through a view still need permission to read the field, and the results still show the value masked to anyone without it. See Sensitive fields.

The operator

The operators offered depend on the field's type:

On screen	Means	For
= · **!=**	equals · does not equal	every type
< · **<=** · **>** · **>=**	before/after, less/more than	numbers and dates
contains · **doesn't contain**	has the text anywhere (case does not matter)	text
starts with · **doesn't start with**	begins with the text	text
is like · **isn't like**	matches a text pattern	text
in · **not In**	is one of a list	references, categories, users
is empty · **is filled**	has no value · has a value	every type
under	sits below a record in a tree (hierarchy) `Since 1.2`	tree entities

Fields with several values — a multi-value field, or a field inside a repeatable block — can first be reduced to one value: **Count of values**, **Sum**, **Average**, **Min**, **Max**, **First Value** or **Last Value**. For example, *Items* **Count of values** **>** 3 finds records with more than three item rows.

Without one, the condition uses **Any Value**: it matches when any of the values meets it. `Since 1.8` a condition saved without a choice here is treated as **Any Value**, in queries and in reaction filters alike.

"is empty" and "is filled" `Since 1.8`

is filled matches only a record where the field is present *and* has a value; **is empty** matches a record where the field is missing, blank or an empty list. Queries, reaction filters and reference-field lists all agree on this.

One limit to plan around: a condition looks at a field across **all** rows of a repeatable block at once — it cannot ask "does at least one row have this filled". A block with one empty row and one filled row counts as empty. If you need row-by-row logic, add a separate marker field on the record.

The value

Choose where the value comes from:

Value source	Use it for	Example
Fixed	A value you choose now	*Status* = *Open*
Field	Another field of the same record	*Assigned to* = *Updated by*
Pointer	A value that depends on who is looking, or when	*Assigned to* = **@me**
Expression	A formula, for cases the others cannot express	—
Parameter	A value supplied when the query is used	see below

The **Pointers** offered depend on the field:

- **Users and groups** — **@me** (the person looking), **@myGroups** (their groups), **@myRoles** (their roles), **@myOrgUnit** (their organizational unit), and more for group membership.
- **Dates** — **@today**, **@now** (today with the time), **@StartOfDay**, **@StartOfWeek**, **@StartOfMonth**, **@StartOfYear**.

With **Expression**, the hint under the box says what the formula must return — for example a `yyyy-MM-dd` date for a date field.

Ask for a value when the query runs: parameters

A **parameter** is a named blank in a condition that is filled in later — for example *Department* **=** the parameter *dept*.

1. In a condition, choose **Parameter** as the value source and enter a parameter name.
2. Save the query. It is marked **Parameterized**.

Where the value comes from depends on where the query is used:

- **As a view's source or in a reference field** — you set the values in that view's or field's own settings.
- **As a quick filter** `Since 1.10` — when someone turns it on, a **Query parameters** dialog asks them for each value. The filter then runs with those values; if they save it, it is saved as a new query of their own.

Good to know:

- The same parameter name can be reused in several conditions only on fields of the same type.
- Removing every parameter from a query turns it back into a normal one. A normal query cannot be turned into a parameterized one later — create a new query.
- A quick filter with a parameter cannot be switched on by default in a view, and is left out of template exports (the view says so). The quick export applies it with the values the person entered.
- `Since 1.10` a **Parameterized** badge marks these queries wherever you pick a query in Administration.

Offer one-click quick filters

A **quick filter** is a query people can switch on from an entries list with one click; its conditions are added to whatever they have already filtered.

1. In the query, tick **Quick Filter**.
2. Enter the **Name** people will see (left blank, it is taken from the query name).
3. Optionally choose a **Group** — or create one with **New group**.
4. Select **Save Query**.

Which quick filters a list offers, where they appear and whether one is on by default is set **per view** — see Quick filters, groups and ordering. People reach a quick filter only through a view that lists it.

Quick filter groups `Since 1.10`

A **quick filter group** collects related quick filters — for example *By region* with *North*, *South* and *West* — so a view can offer the whole set at once.

1. On the **Queries** tab, open **Quick filter groups** and select **New quick filter group**.
2. Enter the **Quick filter group name** in each language.
3. Use **Add existing filter** or **Create quick filter** to fill it.

A quick filter belongs to at most one group; adding it to another group moves it (the dialog shows **Moves from …**). Group names must be unique on the entity. Deleting a group keeps its quick filters — they just become ungrouped.

Colour matching records: tags

A **tag** marks the records a query matches, with a colour people see in the list.

1. In the query, tick **Save Tag**.
2. Enter the tag's **Name**, and choose its **Color** and **Icon**.
3. Choose a **Tag Group**, or create one with **New group**.
4. Select **Save Query**.

People apply tags from the entries list's tag options, as a coloured frame or background. Tag groups are managed on the **Tag groups** sub-tab, in the same way as quick filter groups.

Use a saved query elsewhere

A saved query can be the source of:

- **Views** — the view shows only matching records.
- **Reference fields** — the field offers only matching records to choose from.
- **Reports** (web only) — the report's rows.
- **Summaries and dashboards** — the records a summary counts or totals.
- **Reactions** — which records a reaction runs for.
- **Grafana panels** `Since 1.6` — a panel can run a saved query, and its parameters become dashboard variables. See Grafana dashboards [Integrate & Extend › Grafana Dashboards].

Apply, change or delete a query

On the **Queries** sub-tab, each query has:

- **Apply query** — shows the matching records on the entity's entries list.
- **Edit** — change it and select **Save Query**.
- **View Dependencies** — see where the query is used before you change it.
- **Delete This Query** — removes it permanently. A query created by the platform, or one used by a virtual, joiner or aggregator entity, cannot be deleted.

Find a query in a long list `Since 1.10`

The list's toolbar can **search** query names and their conditions, **filter** by type, tag, field, who created or changed it and when, **sort** by last modified, created date or name, and **group** by type, tag or not at all.

Recipes

You want	Conditions
Only my records	*Created by* **= @me**
Records assigned to my team	*Assigned group* **in @myGroups**
Created this month	*Creation date* **>= @StartOfMonth**
Due in the past and not done	*Due date* **< @today**, and *Status* **!=** *Done*
Has more than three items	*Items* **Count of values >** 3
Department chosen when used	*Department* **= Parameter** *dept*

Searching, sorting and entry state on a list

These belong to the entries list rather than to a saved query:

- **Search** looks across every searchable field. A field is searchable when a view enables search on it, or when the field's own search option is on.
- **Sorting** is newest-changed first by default. People can pick a field and a direction from the sort menu, or use a column's sort icon.
- **Entry state** — people can include **archived** or **obsolete** records (records that no longer match the entity's current design), when the view allows it.

To count or total records rather than list them, use a **summary** — see Summaries & dashboards. To add totals under a list, see Totals & grouping.

Troubleshooting

Problem	What to check
A field is missing from the field list	Encrypted fields and JSON contents cannot be filtered.
A date condition matches nothing	Compare date-only fields with **@today** or **@StartOf...**, not **@now**.
A repeatable-block condition looks wrong	A condition reads all rows at once; see is empty and is filled.

Problem	What to check
The query cannot be saved: name already used	Query names must be unique on the entity.
People are asked for a value every time	It is a parameterized quick filter — that is by design.
A query cannot be deleted	It is used by a virtual, joiner or aggregator entity, or was created by the platform. Check **View Dependencies**.

Go deeper

For integrators: Query model and API reference [Integrate & Extend] — the condition operators by engine name, quick-filter parameters, quick-filter groups and tag routes · Criteria dialects [Integrate & Extend › Criteria and field-path dialects] · Reading data [Integrate & Extend › Reading Data] · For platform engineers: Engine: queries & aggregation

Reports

This guide is for **builders** who need a formal, printable list — *open purchase orders this month*, *overdue maintenance tickets*, *new hires by department* — that people can open, export to PDF or Excel, and receive by email on a schedule. By the end you will be able to create a report, lay out its PDF, decide who may open it, schedule its delivery, and publish it.

How a report works

A **report** is a table built from one saved query on one entity: the query decides the **rows**, and the fields you pick are the **columns**, in the order you pick them.

Take a **Purchase Orders** entity. A query *Open orders this month* selects the rows; the report shows *Order number*, *Supplier*, *Amount* and *Requested by*. The finance team opens it from the portal, exports it to Excel, and the finance manager receives it as a PDF every Monday morning.

Good to know before you start:

- Reports work in the **web** portal only — the mobile app does not open them.
- A report exports to **PDF** or **Excel (XLSX)**. There is no CSV export of a report.
- Protected (sensitive) fields are masked (`****`) for anyone not allowed to see them — on screen, in the exported file and in the emailed attachment.

Create a report

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Reports** tab, then **New Report**.
3. Fill in the sections below, then select **Done**.
4. **Publish** the report from its card on the **Reports** tab when it is ready — see Publish.

Report information

- **Title** — the report's name, in each language. Two reports on the same entity cannot share a title.
- **Query** — the saved query that selects the rows. It must belong to this entity.
- **Description** — optional; printed in the PDF heading under the title.
- **Icon** — optional.
- **Fields** — the columns, in order. The order you choose is the column order on screen and in every export.
- **Change fields** — lets people choose which of the report's columns to include when they export. They can only remove columns, never add one the report does not have.

> ⚠ **PDF shows only the first 7 columns**
>
> A PDF export prints at most **7 columns** — any column after the seventh is left out. Put the most important columns first, choose **Landscape** for wide reports, or export to Excel, which has no column limit.

Report design

Under **Report Design**, choose what the PDF prints around the table. **Preview** shows how the parts fit together.

Option	Prints
Page header	The report's title and description at the top of each page. On by default.
Organization Header	Your organization's header image. On by default.
Report Header	A **Text** or image heading of your own.
Page Footer	Page numbers ("Page 1 of 3"). On by default.
Organization Footer	Your organization's footer image.
Report Footer	A **Text** or image footer of your own.
Layout	**Portrait** or **Landscape**.

Organization header and footer images come from your organization's settings. Arabic text is printed right to left. Long header and footer texts are cut to about three lines.

Who can open it

Under **Access Permission**, choose the users and groups who may open the report. If you restrict the report, you must choose at least one user or group before you can publish it. On the **Reports** tab, a report opened to chosen users or groups is listed under **Restricted**.

Email it on a schedule

Under **Sending Setting**:

1. **Timer** — set the **Interval type**:
    - **None** — the report is not sent.
    - **One Time** — sent once at the **Timer Date**. The date cannot be in the past when the report is published.
    - **Repeat** — sent on a schedule written as a **Cron Expression**, between an optional **Start Date** and **End Date**.
2. **Channel** — **Channel Type Send Email** (the only channel today). Choose the recipients in **To**, **CC** and **BCC** — people, groups, lists or typed addresses — and write the **Subject** and **Body**.
3. **Sending Report Method** — tick **Send report link** to put a link in the email, **Send report attachment file** to attach the report, or both. For an attachment, choose the **Attachment type**: **PDF** or **XLSX**.

Each scheduled email is sent only while the report is **published**. If the attachment cannot be produced, the email is still sent — without the attachment.

Help & documentation lets you write help text that people see with the report.

Publish

On the **Reports** tab, select **Publish** on the report's card. Publishing makes the report available to the people you chose, and starts its schedule. **Unpublish** takes it back.

How people open and export a report

A published report can be opened:

- **From the portal** — in the sidebar, open the module, then the report.
- **From Administration** — on the entity's **Reports** tab.
- **From an email** — through the report link.

To download it, people open the report's actions menu and choose **Export to Excel** or **Export to PDF**. Both exports show the same rows and values as the screen. What a reader can do with a report is described in Dashboards & reports [Using OSOS › Dashboards, Counters & Reports].

Troubleshooting

Problem	What to check
The PDF is missing columns	PDF prints the first 7 columns only. Reorder the **Fields**, or use Excel.
You cannot save: the query is rejected	Choose a saved query of this entity. Reports only take a query as their source.
You cannot save: field not found	A chosen field was removed from the entity. Pick the fields again.
You cannot save: name already used	Another report on this entity has the same title in one of the languages.
You cannot publish	A restricted report needs at least one user or group, and a one-time send must be in the future.
Scheduled emails are not arriving	Check the report is published, the **Interval type** is not **None**, and the recipients are right.
A column shows `****`	The reader may not see that protected field.
People on mobile cannot open the link	Reports open in the web portal only.

Go deeper

For integrators: Report model and API reference [Integrate & Extend] — the stored model, validation and error codes, the execute and export endpoints, and how exports are rendered · Exported values use the same rendering as export templates.

Summaries, dashboards & counters

This guide is for **builders** who want to show people numbers instead of lists — *open requests by department*, *total spend this quarter*, *tickets created per week*. By the end you will be able to create summaries, arrange them on a dashboard, show a dashboard inside a view or a page, and add counter tiles that show how many records are waiting.

How it works

There are three building blocks:

- A **summary** is one card — a single **number**, a short **list**, or a **chart** — calculated from an entity's records: *Count of requests grouped by Department*, *Sum of Amount*, *Average of Resolution time by Priority*.
- A **dashboard** is a set of summary cards, arranged in named groups. Every entity and every module has one, and there is one for the whole system.
- A **counter** is a small tile that shows how many records a view has — *12 tasks waiting* — and opens them when clicked.

Take an **IT Requests** entity. A summary *Open requests by department* counts the requests that have not reached the final step, grouped by *Department*, as a column chart. It sits on the entity's dashboard, appears in the *My department's requests* view, and is placed on the IT team's start page. The worked example builds it.

Everything here is OSOS's own dashboarding. If your audience works in Grafana, or you need one dashboard across several systems, the same summaries can feed Grafana panels `Since 1.6` — see Grafana dashboards [Integrate & Extend › Grafana Dashboards].

Create a summary

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Dashboard** tab, then **Add Summary**.
3. Under **Summary Information**, enter the **Title** and, optionally, an **Icon** and a **Description**.
4. Choose how to show it in **Show as**: **Number**, **List** or **Chart**. For a chart, choose the **Chart type**: **Column**, **Bar**, **Line**, **Pie** or **Doughnut**.
5. Choose the **Source**: **All entries**, or **Queries** and then the **Query** that selects the records.
6. Choose the **Field** and the **Function**: **Count**, **Sum**, **Min**, **Max** or **Average**.
7. For a list or a chart, choose what to **Group by** — a field, or a detail such as the creation date. Dates can be grouped by year, quarter, month, week, day and so on. Then choose the **Order by**.
8. Optionally pick an **Additional Filter**, the **Select colors** for the chart, and **Display in overview tab** (see Show it in a view).
9. Save the summary.

The rules the form follows:

- A **Number** has no grouping — it is one figure.
- A **List** or a **Chart** needs a **Group by**.
- A **Chart** needs a **Chart type**.

- Two summaries on the same entity cannot share a title.

A summary always leaves out archived and obsolete records. When a grouping has very many values, the chart shows the largest ones and gathers the rest under **Other Values**.

Colours `Since 1.10`

Select colors holds up to **20** colours, used in order for the chart's segments. Hover over a colour to copy or remove it; **copy all** copies the whole list. Press **Ctrl+V** to paste colours copied from another summary or a design tool — any `#RGB` or `#RRGGBB` values are added until the list is full; anything else is ignored.

Duplicate a summary `Since 1.10`

Select **Duplicate** on a summary card (next to **Enlarge**). The summary form opens as a new summary with everything copied — source, function, grouping, display, colours and access policy — and the name followed by "Copy" in your current language. The original is untouched. You can duplicate only a summary you can edit.

What people can do with a card

- **Click to browse** — click a number or a chart segment to see the records behind it (when the view allows it).
- **Enlarge** — open the card in a larger window.
- Narrow the card to *this week*, *this month* or a date range.
- Cards refresh themselves when the numbers change.

In preview mode, cards show made-up figures, so a preview does not match the live numbers.

Protected (sensitive) fields `Since 1.10`

If the entity has a sensitive field, the summary form shows a **Sensitive Data Access Policy** — the same setting reactions and export templates have:

- **Privileged** — the summary always uses the real values.
- **Prohibited** — the summary may not use protected values.
- **User-Based** — the summary uses what the person looking may see. **Auto** is not offered for summaries.

A card the policy does not allow for a reader shows a lock and **You do not have permission to view this summary** instead of a figure. A reader who is allowed to see the values, but has not unmasked sensitive fields yet, is asked to unmask them first. Summaries made before this setting existed keep working as before.

Arrange a dashboard

Dashboards exist at three levels:

Dashboard	Where you build it	What it can hold
Entity	The entity's **Dashboard** tab	The entity's own summaries
Module	The module's **Dashboard** tab	Summaries from any entity in the module — choose the entity, then the summary
System	**Administration › Applications › Dashboards**	Summaries from any module

On a dashboard:

- **Add Summary** adds a card; on a module or system dashboard you **Choose module**, **Choose entity** and **Choose summary**.
- **Add group** creates a named block; drag cards into it, and drag groups and cards to reorder them.
- A summary can appear only once on a dashboard.

Every new entity and module gets an empty dashboard automatically. Removing a card from an **entity** dashboard also deletes the summary behind it, unless it is still used elsewhere on that dashboard.

Who can edit a dashboard. The system dashboard needs system admin or developer access; a module dashboard, edit access to the module; an entity dashboard, edit access to the entity or its module.

Show summaries in a view or on a page

In a view. In the view's settings, turn on **Dashboard**, then:

- **Summaries to show** — **Show all summaries**, or only the summaries whose source matches the view's own source.
- **Click to browse** — let readers open the records behind a figure.
- **Apply view's source query to summaries that have all entries as a source** — so an "All entries" summary counts only what the view shows.
- **Append view's source query to summaries that have different query as a source** `Since 1.6` — so a summary with its own query is also narrowed to the view. A summary whose query differs from the view's cannot offer **Click to browse**.

Summaries marked **Display in overview tab** appear in the view's overview tab when the view enables it. When the view narrows its summaries with its own query, the "N Entries" caption under a card counts the same narrowed records you see when you click through `Since 1.4`.

On a page. Add a **Summary** widget to the page and choose the module, the entity and the summary — see Pages & widgets.

Add a counter

A **counter** shows how many records a view has, optionally narrowed by a query, and opens them when clicked.

1. Go to **Administration › Applications › Counters**, then **New Counter**.
2. Enter the **Name** and, optionally, an **Icon** and a **Description**.
3. Choose the **View** whose records it counts.
4. Optionally choose the **Counting Criteria** — a query that marks which of the view's records count.
5. Save.

The view must have its entries list turned on and must offer the **Counter view** display; otherwise the counter cannot be saved. People see a counter only for views they can open in the current space, ordered by the counters' display order. When a counter has counting criteria, its list still shows all the view's records, marking those that match.

Summary or total?

	Summary	**Total (aggregator)**
What it is	A card of its own — a number, a list or a chart	A figure under an entries list

	Summary	Total (aggregator)
Where you make it	The entity's **Dashboard** tab	The entity's **Aggregators** tab
Grouping	Its own **Group by**	The list's grouping
Where it appears	Dashboards, views, pages, Grafana	The views that switch it on

Choose a summary when the figure should stand on its own; choose a total when it belongs to a list the reader is already looking at.

Worked example: open requests by department

1. **The query.** On the *IT Requests* entity, create a query *Open requests*: the current workflow step is not the final step. See Queries & filters.
2. **The summary.** On the **Dashboard** tab, **Add Summary**: **Title** *Open requests by department*; **Show as Chart**, **Chart type Column**; **Source Queries**, **Query** *Open requests*; **Field** any field (for example the request number) with **Function Count**; **Group by** *Department*; **Order by** descending. Save.
3. **The view.** In the *My department's requests* view, turn on **Dashboard** and **Click to browse**.
4. **The page.** On the IT team's start page, add a **Summary** widget and choose this summary. Publish the page.
5. **Check it.** Open the page as a member of the IT team, click a column, and confirm the list shows that department's open requests.

Troubleshooting

Problem	What to check
Group by is missing	**Show as** is **Number**. Choose **List** or **Chart**.
The chart shows one bar with no label	The grouping field is empty on those records.
Figures differ from the list	The summary skips archived and obsolete records; check also its query and the view's **Apply/Append view's source query** settings.
Clicking a figure does nothing	Turn on **Click to browse** in the view; a summary with a different query than the view cannot offer it.
A card shows a lock	The summary's **Sensitive Data Access Policy** does not allow this reader.
You cannot add the same summary twice	A summary can appear only once per dashboard.
A counter cannot be saved	The view needs its entries list on and the **Counter view** display.
Preview numbers look wrong	Preview uses made-up figures.

Go deeper

For integrators: Summary and dashboard API reference [Integrate & Extend] — the summary (statistic query), dashboard and counter models, validation, the execute and batch-load endpoints, view-scoped execution and the enums · For platform engineers: Engine: queries & aggregation

Totals & grouping

> **Availability**
>
> **Since 1.10.**

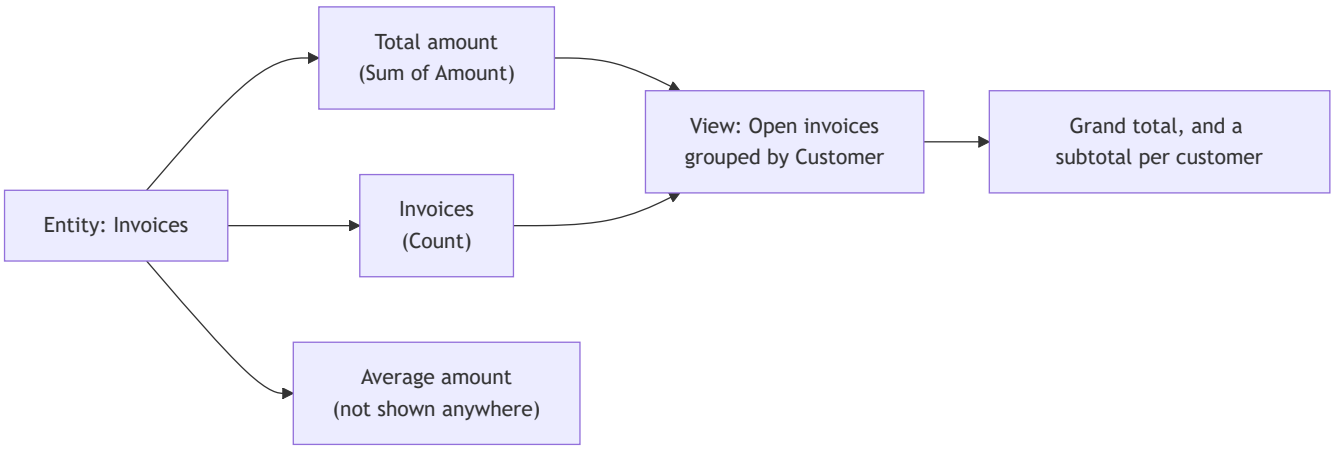
This guide is for **builders** who want an entries list to answer "how much?" as well as "which ones?" — the total value of open invoices, the number of requests per department, the average resolution time per priority. By the end you will be able to define totals on an entity, show them in the views that need them, and group a list so each group carries its own subtotals.

How it works

There are two separate settings, set in two places:

What	Where	What it decides
An **aggregator** — a field plus a calculation, such as *Sum of Amount*	The entity's **Aggregators** tab	Which totals *can* be shown
Which aggregators a view shows, and how it groups	The view's entries-list settings	What *this* view shows

Take an **Invoices** entity. You define two aggregators — *Total amount* (Sum of *Amount*) and *Invoices* (Count). The *Open invoices* view switches both on and groups by *Customer*: readers see one block per customer with its own total, and a grand total for the whole list.



An aggregator takes effect as soon as you save it — there is no publish step. It travels with the entity when you package it.

Define an aggregator

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Aggregators** tab, then **New Aggregator**.
3. Choose the **Field** and the **Operation**.
4. Enter a **Display Name** — readers see it next to the figure.

5. Save the dialog.

Copy opens a new aggregator pre-filled from an existing one (named "… - Copy"); **Edit** and **Delete** do what they say. Deleting an aggregator removes it from every view that showed it, without an error — those views simply stop showing that figure.

Which operations a field allows

The field is…	You can choose
A **Number** field	**Count**, **Sum**, **Min**, **Max**, **Average**
Any other type (text, date, reference…)	**Count** only
An **encrypted** field	Not offered

Count counts records that have a value in the chosen field, not every record in the list `Since 1.10`. Count *Assigned to* tells you how many invoices have someone assigned.

Fields inside a block can be used (each field of the block is offered, not the block itself). There is no limit on how many aggregators an entity has, and one field can have several — *Sum of Amount* and *Average of Amount* appear stacked in the same column. Name each one, so they can be told apart.

Show totals in a view

1. Open the view's settings and go to its entries-list settings.
2. Under **Aggregators**, tick the aggregators this view should show.
3. Save the view.

A view shows **only** the aggregators ticked on it. A new aggregator does not appear anywhere until you tick it in each view that should have it — there is no "all views" switch. This is deliberate: a published or public view never starts showing a new figure without your decision. On the entity's own entries list in Administration (with no view), every aggregator is shown.

Group a list

In the same entries-list settings, under **Grouping**:

1. Select **Add Group By** and choose a field. Add more levels if you need them — the first is the outer group; drag levels to reorder them.
2. Optionally turn on **Collapse groups by default** so every group starts closed.
3. Optionally turn on **User change grouping** to let readers choose their own grouping from the list's view settings.
4. Save the view.

A grouping level must be one of the fields the view lists — you cannot group by a column you have hidden.

Fields you can group by

You can group by text, translated text, number, date, category, switch, a single-value reference, a mapped field, and any field inside a non-repeating block.

You **cannot** group by a field that can hold more than one value for a record — a multi-value field, a repeatable field, a field inside a repeatable block — because the record would belong in several groups at once and every count would be wrong. Lookup fields, ranged (from/to) fields and the record's details (*Created by* and so on) cannot be grouped either.

- **Multi-value categories** `Since 1.10`. A category field set to hold several values cannot be a grouping level — nor a default sort, a dashboard group-by, a Kanban column, a timeline category, or the entity's main category. If you switch a category to multi-value while one of those uses it, the change is refused until you remove the usage; and a view that grouped by it before simply stops grouping.
- **References** — each group is one related record, titled with its main title. If that record was deleted, archived or is out of the reader's reach, the group keeps its entries and shows no title.
- **Translated text** groups on the language the reader is using, so readers in different languages can see different groups.

Read the figures

- The **grand total** covers **every record the list matches** — after filters, search and quick filters — not just the page on screen. Moving to page 4 does not change it.
- Each **group** shows the same aggregators again, as a **subtotal** for that group.
- Subtotals do not always add up to the grand total — group *sums* do, group *averages* do not.
- When the list has exactly one group, the grand-total row is left out (it would repeat that group).

What a cell shows:

You see	It means
-	There was nothing to total — no records, or none with a value.
0	A real zero.
********	The figure exists, but the field is protected and this reader may not see it.

Whose grouping wins

The view's grouping is what everyone sees when they open it. If you allowed **User change grouping**, a reader's own choice lasts for their session only:

When the reader…	Is their grouping kept?
Reloads the page, or moves around the portal and comes back	Yes
Opens a new tab, signs out, or uses another device	No
— and you change the view's grouping	No — your change replaces it

Order of the groups. Unsorted, groups follow the field's own order — the option order of a category or switch, or the title of a referenced record. When a reader sorts, groups are ordered by their highest (or, ascending, lowest) value in the sorted column, counting only the rows the reader can see.

Protected fields

See Sensitive fields for how a field becomes sensitive or encrypted.

- **An aggregator over a sensitive field** stays visible under its name, with the value shown as ******** to readers who may not see the field.

- **An encrypted field** cannot be used for an aggregator at all, and is not offered for grouping.
- **Grouping by a sensitive field** puts every record into **one** group titled ********, for everyone — even people allowed to see the field. Records with no value join that same group, so nothing about the field leaks. The setting is allowed, but useless; remove it if you see it.
- **On a public view**, an aggregator over a sensitive field is left out entirely.

Good to know

- In preview mode, totals show **-** and lists are not grouped — preview never invents figures.
- **Exports carry no totals or group headings.** Grouping only puts each group's rows next to each other in the file. If a total must leave the platform, read it from the list or a summary.
- **Aggregators or summaries?** An aggregator is a figure on a list the reader is already looking at. A summary is a card of its own — a number, list or chart — that can sit on a dashboard.

Troubleshooting

Problem	What to check
A new aggregator does not show	Tick it under **Aggregators** in each view that should show it.
Only **Count** is offered	The field is not a Number field.
A field is missing from **Add Group By**	It can hold several values, is encrypted, or is not listed in the view.
A figure shows **-**	Nothing matched, or no matching record has a value.
Everything is in one ******** group	The grouping field is sensitive. Group by another field.
A reader's grouping disappeared	Their grouping lasts for the session, and resets when you change the view's grouping.

Go deeper

For integrators: Aggregate and grouping reference [Integrate & Extend] — the design rules and their reasons, the aggregate and summary routes, and what the end-user view load returns · For platform engineers: Engine: queries & aggregation

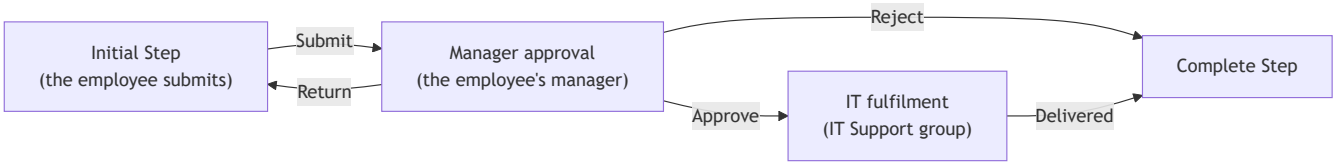
Workflows & approvals

This guide is for **builders** who want records to go through a process — a request that a manager approves, a ticket that moves from triage to resolution, a document that is reviewed before it is published. By the end you will be able to add a workflow to an entity, decide who acts at each step, route records with actions, send them back for changes, control which fields people see at each step, add time limits, and test the result.

How a workflow works

A **workflow** is the path a record follows from the moment it is submitted until the work is done. It is made of **steps**; each step has an **owner** (the person or team who must act) and one or more **actions** (the buttons the owner sees, such as **Approve** or **Reject**). Choosing an action moves the record to another step.

Take an **Asset Request**: an employee asks for a laptop. Their line manager must agree, then the IT team hands the laptop over.



A few things are always true:

- Every workflow starts at an **initial step** and ends at a **final step**. When you add a workflow, both are created for you (named *Initial Step* and *Complete Step*); you add your own steps between them. The initial step is always owned by the person who created the record.
- Steps are in order. That order is what "next step", "previous step", "first step" and "last step" mean when you route an action.
- The usual route to the goal is the **main path**. Any other route — a rejection, a return for changes, a conditional detour — is a **sub-path**.
- An entity can have more than one workflow; each appears as its own tab on the entity's **Workflows** tab.

> **Note**
>
> To let people request the record from the portal's **Services Catalog** — rather than adding it from a view — publish the entity as a service. See Publish a service with an approval workflow.

Add a workflow to an entity

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Workflows** tab.
3. Select **Add A Workflow**. A workflow named *Main workflow* is created with its initial and final steps.
4. To rename it, open **Workflow Settings** and change **Workflow Name**.
5. Select **Save** at the top of the designer.

To add a second workflow, select the **+** (**Add Workflow**) next to the workflow tabs. **Clone Workflow** copies an existing one; **Delete This Workflow** removes it.

Add a step

1. On the **Workflows** tab, select **Create Step** at the point in the flow where the new step belongs. The **Create New Step** dialog opens.
2. Under **Step Information**, enter the step's **Name** (in Arabic and English; 3 to 128 characters) and, optionally, a **Description**.
3. Under **Step Type**, keep **Normal** for an ordinary review or approval step. (**Sub-Workflow Step** and **External Step** are described below.)
4. In **Main Action**, choose where the step's main action sends the record — usually **Next Step** under **Relative Steps**, or a specific step under **Existing Steps**.
5. Select **Return** to add a *Return* action that sends the record back to the first step, and **Reject** to add a *Reject* action that ends the workflow. See Return for changes, or reject.
6. Under **Step Owner**, choose who acts at this step — see Choose who acts at a step.
7. Optionally turn on **Timing** to give the step a time limit — see Time limits and escalation.
8. Select **Done**, then **Save**.

To change a step later, select it in the flow; its **Step Settings** open beside the flow. Use **Duplicate** to copy a step and **Remove Step** (or **Delete This Step**) to delete it. The initial and final steps cannot be deleted, but you can rename them and set their field rules. You cannot delete a step that another step's action targets directly — change that action first.

Choose who acts at a step

Every step except the first and last needs an owner. In **Step Owner**, first choose the kind of owner — **User**, **Group** or **List** — and then where that owner comes from.

You want the step to go to…	Set it up like this
One specific person (for example the HR manager)	**User**, **User Type Fixed**, then pick the person in **User Name**.
The submitter's manager	**User**, **User Type Previous Step Owner**, **Step Name** = the initial step (whose owner is the submitter). Turn on **Relative**, choose **Relation Manager** and **Relation Grade 1**. Use grade **2** for the manager's manager.
A person named in the record (for example an *Assigned to* field)	**User**, **User Type Field**, then pick the field in **Field Name**.
Whoever acted at an earlier step	**User**, **User Type Previous Step Owner**, then pick that step in **Step Name**.
A team, such as the IT Support group	**Group**, **Group Type Fixed**, pick the group in **Group Name**, then choose how the team shares the work (below).
The people in a list	**List**, **List Type Fixed**, pick the list in **List Name**. The first person in the list is used.
Someone calculated by a formula	**Expression** as the source type. This is for advanced builders; the expression must return a user, group or list.

When a team owns the step, **Select User Type** decides how the work is shared:

- **Single User** — one member handles it. In **Choose Selection Type** pick:
    - **First one** — the first member of the group is assigned automatically.

    - **Take Ownership** — nobody is assigned; any member opens the record and selects **Take Ownership** to claim it. **Return to** decides who gets the step if the record comes back to it later: the **Group** again, or the **Last owner**.
    - **Org Unit Manager** — the manager of the group's organizational unit is assigned.

  Access Permission decides whether only the assigned person or every member can see the step.

- **All Users** — every member must act. **Multi Assignment Type** chooses **Sequential** (one after another) or **Parallel** (all at once). Then set the group decision:

    - **Main Action Execution Condition All users** — the record moves on with the main action only if everyone chose it.
    - **Some Users** — the record moves on once enough members chose the main action: set **Proceed Threshold Type** (**Fixed Number**) and **Proceed Threshold** (for example 2 of 3). **Action On Failure** chooses what happens when the threshold is not reached — for example **Most Chosen Secondary Action**, **Return** or **Reject**.

Fallback Assignment. If the owner might not be found — the submitter has no manager recorded, a field is empty, a group has no active members — the platform tries a backup. Whenever the owner is anything other than one fixed person, the designer adds **Fallback Assignment 1** for you and requires it. Fallbacks are tried in order; end the chain with one specific person (**User**, **Fixed**).

Two more options sit under the owner:

- **Enable Reassign** lets the current owner hand the step to someone else.
- **Allow Adhoc step** lets the owner insert an extra step on the fly — for example to ask a colleague for an additional approval or more information. The record waits there, then returns to this step automatically.

Route the record with actions

Each step needs at least one action (except the final step). The main action is created with the step; to add more, select **Create Action** on the step. In **Action Settings**:

- **Name** — the button label people see, such as *Approve* or *Send to IT*.
- **Default Target** — where the record goes: a relative step (**First Step**, **Previous Step**, **Current Step**, **Next Step**, **Last Step**) or a specific step from **Existing Steps**.
- **Expression based target** — route to different steps depending on the record, for example send requests over a certain amount to a director. Select **Add Conditional Step**, write a condition and pick its **Target Step**; conditions are checked in order and the first one that is true wins. If none is true, the default target is used.
- **Main Action** — marks the action as the step's main (happy-path) action. A step can have only one.
- **Show Confirm Action Dialog** — asks the owner to confirm before the action runs. **Action comment status** sets the comment box in that dialog to **Optional**, **Required** or **Hidden**; **Custom dialog title** and **Custom dialog message** change its wording.
- **Show Message** — shows a message after the action runs.
- **Conditional Visibility** — shows the action only when a condition on the record is true.

Return for changes, or reject

Most approval steps need a way to say "not yet" and a way to say "no":

- The **Return** checkbox in **Create New Step** adds a *Return* action that sends the record back to the **first step**, so the submitter can correct it and submit again. To send it back one step instead, open the action and set **Default Target** to **Previous Step**.
- The **Reject** checkbox adds a *Reject* action that sends the record to the **final step**, ending the workflow.

Rename either action — for example *Return for changes* — and set its **Action comment status** to **Required** so the owner must explain why.

Ask for identity confirmation before an action `Since 1.10`

For sensitive decisions — approving a payment, signing off a contract — you can make the owner confirm who they are again before the action runs. In **Action Settings**, turn on **Require step-up verification before execution**. Leave the expression empty to always ask, or write a condition to ask only for some records (for example only above a certain amount).

Good to know:

- The setting is per action, so *Approve payment* can ask while *Request more information* does not.
- People confirm with a code from their authenticator app, or a one-time code by text message or email; the account password is used only when none of those is set up. **A passkey cannot be used for this** — tell your users so.
- If several records are selected and the action is run on all of them, one confirmation covers the whole selection.
- If your condition has an error, the platform **asks for confirmation** rather than skipping it. Test the condition before you publish.
- Administrators acting from an admin page and automatic actions (reactions) are not asked.

What the person acting sees is described in Verifying your identity before an action runs [Using OSOS › Acting on Workflow Steps].

Control the fields at each step

At every step you decide how each field behaves. Open the step's **Requirements**, and for each field choose a **View Option**:

View Option	Effect at this step
Hidden	The field is not shown.
Locked	The field is shown but cannot be changed.
Editable	The field can be changed. This is the default.
Mandatory	The field must be filled before the record can move on.

For the Asset Request: at the initial step *Asset type* and *Reason* are **Mandatory**; at *Manager approval* they are **Locked** and *Manager comment* is **Editable**; at *IT fulfilment Serial number* is **Mandatory**.

Good to know:

- The final step and sub-workflow steps have no field rules of their own. A field that is **Hidden** at the steps a record passed through stays hidden after it is rejected or completed.
- Each person sees the rules of the step they own, or of the last step they acted on; anyone else sees the rules of the record's current step.

- A view set to **Edit entry as admin** or **View entry as admin** lifts the field rules only for people who take no part in the current step. The step's owner still gets the step's rules there.

Time limits and escalation

To make sure a step does not wait forever, give it a time limit (an SLA):

1. Open the step and turn on **Timing**.
2. In **Timing Schedule**, choose the working-hours schedule to count against, so weekends and holidays are not counted. Schedules and holidays are set up by your tenant administrator — see Timezones, workshifts & time monitoring [Administration › Working hours & holidays].
3. Choose the **Time Unit** (**Minutes**, **Hours** or **Seconds**).
4. In **Timing method**, choose **Fixed Value** and enter the **Value**, or take the limit from a field (**Field Value**) or an **Expression** — for example a shorter limit for urgent requests. **Fallback Value** is used if the field or expression gives no number.
5. Select **Done**, then **Save**.

Each monitored step is tracked as *Pending*, *Achieved* or *Overdue*. To **escalate** a breach, add a reaction on the entity's **Reactions** tab with the event **On Time Exceeded** for that step — for example, notify the owner's manager and re-notify every few hours. See Reactions & automation and Reacting to breaches [Administration › Working hours & holidays].

You cannot turn **Timing** off while a reaction uses it; remove or change the reaction first.

Reuse a process: sub-workflows

If several processes share the same stage — a standard budget approval, say — build it once as its own workflow on the entity and call it from a step:

1. Create the shared workflow on the same entity (it needs its own initial and final steps).
2. In the main workflow, add a step with **Step Type Sub-Workflow Step**.
3. In **Target Workflow**, choose the shared workflow, and in **Merge Step**, choose where the main workflow continues when it finishes.

A sub-workflow step has no actions of its own: the record moves on automatically when the shared workflow reaches its final step. Its first step is owned by whoever owned the sub-workflow step. If you set a time limit on the sub-workflow step, it covers the whole time the record spends inside the shared workflow.

Hand a step to an external system `Since 1.8`

An **External Step** sends the owner to another system — an e-signature service, a payment page, a partner portal — and the record moves on only when that system sends the person back. The owner cannot complete the step by clicking a button in the portal.

In the step, set **Step Type** to **External Step**, then fill in **URL**, **Method** and **Open As**. The owner must be **one specific person**. Put the return link in the URL with `{{@returnUrl}}` (or `{{@returnUrl:actionId}}` when the step has several actions; each action shows its **Action Id** to copy). The external system's developers will need the external step contract [Integrate & Extend › Workflow model and API reference].

How the external page opens

Open As decides how the owner reaches the external page: **Full Screen**, **Iframe** (inside the record), **Sidebar**, **Dialog**, or **Popup Window**. The mobile app supports only **Full Screen** and **Iframe** and opens the others full screen, so choose one of those two if the step must look the same on both.

Test your workflow

Before you share the application:

1. **Save** the entity. An entity has no publish step; the views people use to reach its records must be published.
2. Sign in as a test user — or ask a colleague — and create a record.
3. Check that the record lands with the owner you expected, and that the owner sees the actions you configured in their **Inbox** or on the record.
4. Try every route: approve, return, reject, and any conditional route.
5. Check the field rules at each step: hidden fields are hidden, mandatory ones are enforced.
6. If you added a time limit, use a short value (a few minutes) while testing and watch the step become *Overdue*; restore the real value afterwards.

Preview mode lets you check forms and views without creating real records. What owners do at run time is described in Executing workflow actions [Using OSOS › Acting on Workflow Steps].

Good to know

- **Rules checked when you save:** a workflow has at least two steps, starts with the initial step and ends with the final step; every step except the final step and sub-workflow steps needs at least one action; a step has at most one main action; a sub-workflow step needs a target workflow and a merge step, and no actions; step names and workflow names must be unique.
- **Changing a workflow** that already has records in progress can require those records to be upgraded. The designer tells you when a change is breaking.
- **Two people acting at once.** If two people act on the same record at the same moment, only the first action counts; the second person is told the record changed and should reload it. Reactions run once.
- If an owner cannot be found and no fallback works, the record cannot move to that step. Always end the fallback chain with a specific person.

Troubleshooting

Problem	What to check
The workflow is marked **This workflow is invalid**	A step has no main action (**Main action is missing**), a sub-workflow step has no target or merge step, or a name is duplicated.
A step cannot be deleted	Another step's action targets it directly. Point that action somewhere else first.
The record goes to the wrong person	Check the owner settings and the **Fallback Assignment** chain. For "the submitter's manager", check that the submitter's manager is recorded in the organization structure.
Nobody sees the step	With **Take Ownership**, the step is unassigned until a member claims it — check the group's members and **Access Permission**.
Timing cannot be turned off	A reaction uses this step's time limit. Remove or change that reaction first.

Problem	What to check
People are asked to confirm their identity unexpectedly	The action's step-up condition may contain an error; a broken condition always asks.

Worked example: a two-step expense approval

1. **Initial Step — "Submitted".** Owned by the person who submits the expense. Main action *Submit* → **Next Step**.
2. **"Manager review".** Owner: **User**, **Previous Step Owner** = *Submitted*, **Relative** on, **Relation Manager**, **Relation Grade 1**; fallback: the finance manager. Actions: *Approve* (main action) → **Next Step**; *Return* → **First Step**, with **Action comment status Required**. **Timing**: 2 working days (**Time Unit Hours**, **Value** 16 against an 8-hour working schedule).
3. **Complete Step — "Approved".** No owner needed.
4. **Escalation.** A reaction on **On Time Exceeded** for *Manager review* notifies the manager's manager (or reassigns the step to them).
5. **Several reviewers instead of one (optional).** Make *Manager review* a **Group** step with **All Users**, **Parallel**, **Main Action Execution Condition Some Users** and **Proceed Threshold** 2, with **Action On Failure Most Chosen Secondary Action**.

Go deeper

For integrators: Workflow model and API reference [Integrate & Extend] — stored model and enums, external-step return contract, step-up client contract, concurrency and the workflow endpoints · For platform engineers: Engine: Workflow & Owner Resolution

Reactions & automation

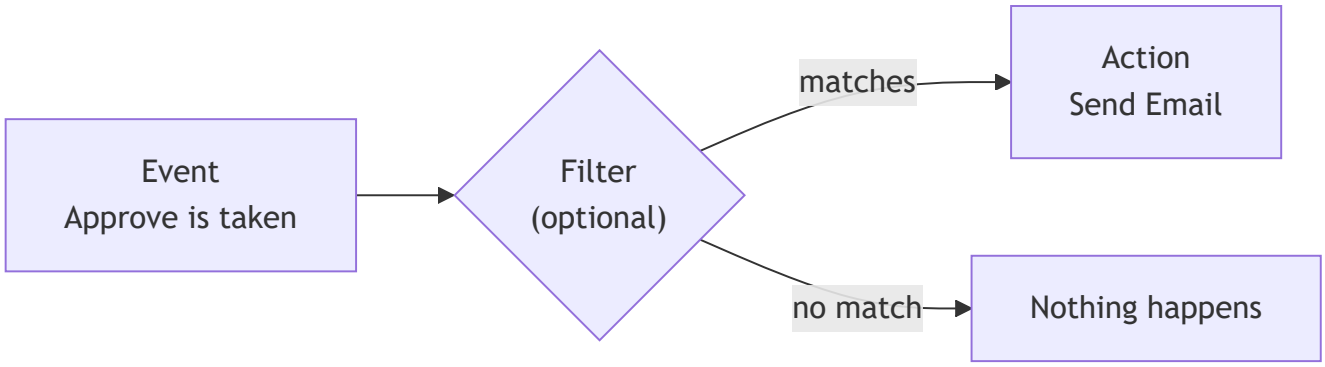
This guide is for **builders** who want an application to do routine work on its own — email the requester when a request is approved, stamp a date on a related record, remind an owner before a deadline, escalate a step that is overdue. By the end you will be able to add a reaction to an entity, choose what sets it off, narrow it to the records that matter, pick what it does, chain several reactions together, and check that each one ran.

How a reaction works

A **reaction** is a rule that runs by itself. It has three parts:

1. An **event** — *when* it runs: a record is created, a workflow action such as **Approve** is taken, a date arrives, someone presses a button.
2. A **filter** (optional) — *which records* it runs for: only requests over a certain amount, only records in a certain status.
3. An **action** — *what* it does: send an email, update a record, move the workflow on, call another system.

Take an **Asset Request**. When the IT manager approves a request, two things should happen: the requester gets an email, and the related *Asset* record is marked as *Assigned*. That is two reactions on the Asset Request entity, both fired by the **Approve** action — one sends the email, the other updates the asset. The worked example builds them step by step.



A few things are always true:

- **One reaction does one thing.** Each reaction has exactly one action. To do two things, add two reactions on the same event, or chain them when the order matters.
- **Reactions live on an entity**, on its **Reactions** tab, and run for that entity's records.
- **Reactions run in the background.** The person whose click set one off does not wait for it and is not told if it fails — see Check that it ran.

Add a reaction

You need **Developer** access or higher on the entity to add, change or delete its reactions.

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.

2. Select the **Reactions** tab. It has two sub-tabs: **New Reaction** for ordinary reactions, and **Time Monitoring** for reactions to an overdue workflow step.
3. On the sub-tab you need, select **+** (or **Add Reaction** if the list is empty). The **New Reaction** dialog opens.
4. Enter a **Name** and, optionally, a **Description**. Turn on **Deactivated** if you want to save the reaction without running it yet.
5. Under **Event**, choose what sets it off — see Choose the event. Depending on the event, more settings appear below it, such as the workflow step or the fields to watch.
6. Choose the **Execution Spaces** and **Event Executor Type** — see Where it runs and who it acts as. The defaults suit most reactions.
7. Optionally build a **Filter** — see Run it only for some records.
8. Under **Action**, choose the **Action Type** and fill in its settings — see Choose what it does.
9. Confirm the dialog. The reaction is saved straight away and appears in the list.

To change a reaction, open it from the list. To pause one, edit it and turn on **Deactivated**. **Duplicate** copies a reaction into a new dialog; you can change the copy's event before saving, unless you started it from an event group's **+**, which keeps that event.

The Reactions tab `Since 1.10`

Each row shows the reaction's event and action at a glance; expand a row to see its description and, for actions that have one, its scope. The toolbar lets you search by name, filter by who created or changed a reaction and when, filter by status — including **Executed** and **Not executed** — sort, and group the list by event, action type or filter status.

Choose the event

Pick the event that matches the moment you care about.

You want to act when...	Choose this **Event**
A record is created	**On Entry Created**
A record is changed (optionally, only when certain fields change)	**On Entry Updated**
A record is archived, or restored from the archive	**On Entry Archived** · **On Entry Restored** `Since 1.8`
Someone takes a workflow action — for example **Approve** or **Reject**	**On Workflow Action Executed**
A record arrives at a workflow step, or leaves one	**On Workflow Step Reached** · **On Workflow Step Exited**
A step is handed to someone else, reassigned or taken over	**On Owner Changed** `Since 1.8`
A date or a repeating schedule arrives	**Timer**
Someone presses a button, or another reaction calls this one	**Custom Action**
A file is produced from an export template	**Export File by template**
Another reaction finished, or failed	**Reaction Succeeded** · **Reaction Failed** `Since 1.8`
Another reaction announced something	**On Event Announced** `Since 1.8`
A workflow step is overdue (on the **Time Monitoring** sub-tab)	**On Time Exceeded**
A Zapier integration sends an event	**Zapier Event**

Approvals are workflow events, not updates. When a record moves from one workflow step to another, **On Entry Updated** does **not** run. To react to an approval, use **On Workflow Action Executed** (for "when someone chose

Approve") or **On Workflow Step Reached** (for "when the record arrives at *Approved*"). On an entity with a workflow, creating a record runs both **On Entry Created** and **On Workflow Step Reached** for the first step.

Event Source Type. By default a reaction runs for the record the event happened to (**Self**). Choose **Relation** to run it when the event happens on a *related* record — for example, react on *Employee* when their *Department* changes. Choose **Absolute Entity** with **On Event Announced**, to run over all of this entity's records that match the filter.

Run on a schedule

With **Timer**, set the **Interval type**:

- **One Time** — runs once at the **Timer Date**.
- **Repeat** — runs on a schedule written as a **Cron Expression**, between an optional **Start Date** and **End Date**.

At each run, the reaction goes through every record that matches its filter — for example "every morning, find requests still waiting after three days and remind their owner". Large sets are worked through in batches, so a run over thousands of records takes a while to finish.

Run from a button: Custom Action

Custom Action reactions run on demand. Attach one to a button (see Buttons), or call it from another reaction with Execute Custom Action.

Custom action level decides who may call it:

- **UI action** — run by a button. Only UI actions can be attached to a button.
- **Local action** — called by another reaction in the **same module**.
- **External action** — called by a reaction in **any module**.

Contextual (on by default) means the action runs *on a record*. Turn it off for an action that simply runs once, with no record — for example "recalculate this month's totals".

For a **UI action** you can also set the **Execution Indicator** people see while it runs — **Blocking** (they wait, with an **On execution started** and an **On execution succeeded** message), **Inform** (they see progress but can carry on), or **None**.

To ask the person for input first, turn on **Ask User For** (called **Parameters** for Local and External actions), give the dialog a **Dialog Name**, and choose the **Fields Source**: an **Inline Entity** you describe, or a **View** whose fields appear in the dialog. The values they enter can be used in the action.

React to an owner change `Since 1.8`

On Owner Changed runs when a step's owner changes. Narrow it with **Owner Change Type** — a hand-off to someone else, someone taking ownership, or a reassignment — and **Choose Step**. Leave either empty to react to every change or every step.

Run it only for some records

The event decides *when*; the filter decides *for which records*. Without a filter, the reaction runs every time the event happens.

- **Watch specific fields. On Entry Updated** lets you pick the fields to watch; it then runs only when one of them changes.

- **Pick steps and actions.** The workflow events let you pick the steps (and, for action events, the actions) they apply to. Leave the selection empty to cover all of them.
- **Add conditions.** Under **Filter**, reuse a **Saved Query** of the entity, add conditions of your own, or both — when you use both, a record must match both. Conditions are built exactly like a query, and can reach into related records.

Good to know:

- For **date-only** fields, compare against **@today** (with an offset such as `+1d` or `-3d`; the sign is required). **@now** includes the time of day, so it never equals a date-only value.
- A full-text search condition cannot be used in a reaction's filter.
- If the filter refers to a saved query that has since been deleted, the reaction is skipped rather than run for every record.

Choose what it does

The **Action Type** list, grouped by what you want to achieve:

Goal	Action types
Change data	**Create Data Entry** · **Update Data Entry** · **Archive Data Entry** · **Delete Archived Entry** `Since 1.8` · **Create / Update / Delete Repeatable Block Record**
Move the workflow	**Move Workflow To Step** · **Execute Workflow Step Action** · **Change Current Step Owner**
Tell people	**Send Email** · **Send Notification** (in-app) · **Send SMS** · **Send WhatsApp** `Since 1.10`
Produce a file	**Generate File** (from an export template)
Manage users and access	**Activate/Deactivate User** · **Set Permission** / **Remove Permission** `Since 1.10`
Connect and extend	**Call API** · **Execute Function** · **Execute Custom Action** `Since 1.8` · **Announce Event** `Since 1.8` · **Notify Zapier**

Filling in values. Wherever an action writes a field or fills a message, choose where each value comes from:

- **Manual** — a fixed value you type, such as *Assigned*.
- **Field** — a field of the record that set the reaction off, such as its *Requester*.
- **Pointer** — **This Entry** (the record that set it off) or **Current User** (the person whose action set it off).
- **Expression** — a formula, for advanced builders.

The sections below cover the actions most builders use. Every option of every action is listed in the reaction reference [Integrate & Extend › Reaction model and API reference].

Update the record, or a related one

Update Data Entry changes field values. **Choose update mode** decides *which* record:

- **Self** — the record that set the reaction off.
- **Reference** — the record a reference field points to. For the Asset Request, choose the *Asset* reference field to update that asset.
- **Relation** — every record reached through a relation, for example all *Tasks* of a *Project*.

When the update can touch several records, **Action Success Criteria** decides when the action counts as a success: **Successful execution on at least one entry** (the default) or **Successful execution on all entries**.

Create Data Entry adds a new record — to this entity or another — and fills its fields the same way. The repeatable-block actions add, change or delete rows of a repeatable block; **Record Selector** picks which rows (by **Number**, **Field**, **Expression**, **Last Record** or **All Records**) and **On Record Selection Failure** says what to do when none match.

Notify people

- **Send Email** — choose the **Template** (or write a **Subject** and **Body**), the recipients in **To**, **CC** and **BCC** (people, groups, lists, org units or a typed address), and **From**: **System**, **Reaction Executor** (the person who set it off) or **Custom**. **ADD Entry Link** adds a link back to the record. You can attach files from the record or a fixed file.
- **Send Notification** — an in-app notification to people, groups, lists or org units, optionally opening a view of the record.
- **Send SMS** — a text message to phone numbers taken from the record or typed in.

Write `{{name}}` placeholders in a message and map each one to a value. Tick **Redact in history** on a value — such as a one-time code — to keep it out of the reaction's run history; the real value is still sent.

Send a WhatsApp message `Since 1.10`

Send WhatsApp sends one of your organization's **approved** WhatsApp message templates.

1. Make sure WhatsApp is set up for the space — your administrator fills in the WhatsApp section of System Settings (see WhatsApp settings [Administration › Organization settings]).
2. In the action, choose the recipients, the same way as for **Send SMS**.
3. Under **Template**, pick an approved template from the list.
4. Under **Template variables**, give a value for each numbered variable in the template, in order.

Good to know:

- You need system-level access to see the template list. If WhatsApp is not set up, the action says so instead of listing templates.
- The template is not checked against WhatsApp when you save. Pick it from the list and fill in exactly as many variables as it has.
- Empty values are sent as blanks; line breaks and long runs of spaces are turned into single spaces; a protected value you may not see is sent masked. If no phone number is found, nothing is sent and the action still counts as a success.

Move the workflow on

- **Execute Workflow Step Action** "clicks" a workflow action for you — for example close a request that has sat in *Resolved* for a week. It continues through the following steps for as long as you have chosen an action for them. It cannot act on the workflow's last step.
- **Move Workflow To Step** jumps the record to a step you choose.
- **Change Current Step Owner** hands the current step to someone else — the usual answer to an overdue step.

Grant or remove access `Since 1.10`

Set Permission gives people or groups access to a resource; **Remove Permission** takes it away. Typical uses: give a request's assignee access to a report when a step is reached, or remove someone's access to a module when their record is archived.

1. Choose the **Resource Type** — **System**, **Module**, **Entity**, **View**, **Page** or **Report** — and the resource itself.

2. Choose **User or Group**, and the **Principal Source**: a **Fixed Value**, a **Reference Field** of the record, an **Expression** or a **Pointer**.
3. For **Set Permission**, choose the **Level**. Views, pages and reports can only be granted view access; modules and entities up to edit; the system up to admin.

Good to know:

- **You cannot grant more than you have.** When you save, each permission action is checked against your own access: to grant a level you must hold it, and to remove access you need admin access on the resource.
- **It never lowers access.** Someone who already has a higher level keeps it.
- A person who has no access to the system yet is given the minimum system access first, so a module or entity grant can take effect.

Call another system, or your own code

- **Call API** sends a request to an external web address and can use values from the record. For safety it only reaches public HTTP(S) addresses, must finish within 30 seconds, and cannot carry a masked protected value.
- **Execute Function** runs a function [Integrate & Extend › Functions: run your own code] your team wrote. `Since 1.10` reactions that run after it can use what it returned.
- **Announce Event** broadcasts a named announcement that **On Event Announced** reactions — on any entity — pick up.

Execute Custom Action `Since 1.8` runs a **Local** or **External** Custom Action — for example, call *Recalculate budget* in the Finance module whenever a purchase is approved. Choose the **Custom Action** (grouped as **Local Actions** and **External Actions**), then, if it is contextual, the **Scope** that finds the record to run it on:

- **Self** — this record (only when the custom action belongs to this entity).
- **Relation** — the records reached through a relation.
- **Expression** or **Pointer** — a formula, or a value such as the current user.

If the custom action asks for **Parameters**, give each one a value; one you leave empty is passed empty. Each target record is handled separately, and finding no target is not an error — the action simply does nothing. A UI action cannot be called this way; it only runs from a button.

Do several things in order: chain reactions

Because each reaction does one thing, a sequence is built from several reactions:

1. Add the first reaction as usual — for example *Create the contract record*.
2. Add a second reaction with the event **Reaction Succeeded** (or **Reaction Failed**, to handle a failure), and choose the entity and the first reaction it follows. It can use what the first reaction produced.
3. Set its **Execution Spaces** to **Triggering Space**.

Announce Event is the other way to connect reactions: one reaction announces, and any number of **On Event Announced** reactions respond — even on other entities.

Good to know:

- The follow-up reaction must exist, and be active, **before** the first one runs. It can use the first reaction's data only for about ten minutes after that run.

- A follow-up reaction may occasionally run **twice** for one run of the first reaction (for example after a server restart). Design it so a second run does no harm — check whether the record it would create already exists.
- **Reaction Failed** runs only when the first reaction's action failed.

Where it runs and who it acts as

Execution Spaces — for organizations with several spaces, where the reaction runs: **Root Space**, **Triggering Space** (where the event happened), **Root Space and Triggering Space**, **All Spaces Except Root**, or **All Spaces**. A chained reaction should always use **Triggering Space**.

Event Executor Type — whose authority the action uses: **System** (the platform's own system user, with full rights), **Manager**, or **Executor** (the person whose action set it off).

Protected (sensitive) fields `Since 1.10`

If the entity has a sensitive field, the dialog shows a **Sensitive Data Access Policy**. It decides what the reaction's action sees of those fields:

Policy	What the reaction sees
Privileged	Always the real value. This is the behaviour when nothing is chosen.
Prohibited	Always the mask (`****`). The action still runs; it just never sees the value.
User-Based	What the person who set it off is allowed to see.
Auto	**Privileged** when the system sets it off (a timer, an overdue step), **User-Based** when a person does.

User-Based cannot be chosen when nothing personal sets the reaction off — a **System** executor, a **Timer** or **On Time Exceeded**. Choose **Auto** there. Under a masking policy, an action that would write a mask over a real value, or send one out by email or **Call API**, fails instead.

Good to know

- **Loops are stopped.** A reaction that updates a record can set off another reaction — or itself. To stop endless loops, each action runs only once per chain of events. Raise **Loop Threshold** (it appears once the reaction has a filter) only if a reaction really must run again within the same chain, and make sure its filter eventually stops matching.
- **No order between records.** When one action updates many records, they may be finished in any order. The order *between* reactions in a chain is kept.
- **A restart can cut a run short.** If the platform restarts mid-run, what already happened is recorded, but the remaining work is not redone. Design follow-up reactions to cope with an earlier one not having happened.
- **A reaction only touches its own entity's records** `Since 1.8`. Records of another entity — for example through a reference field that accepts several entities — are skipped.
- **Fixed files in a reaction** `Since 1.10`, such as an email attachment you uploaded, are kept for as long as the reaction exists. If the upload expired before you saved, upload it again.
- **Secrets stay out of the run history** `Since 1.6`. Values from sensitive fields, values you marked **Redact in history**, and anything that looks like a password or access key are masked in the history. The action itself still gets the real value.

Check that it ran

A failed reaction does not show an error to the person who set it off, so check here:

- **On the Reactions tab** `Since 1.10`, each reaction shows how many runs succeeded and how many failed since it was last changed. System administrators can open either number to list the runs.
- **Reaction Logs** `Since 1.10` — system administrators see every run in **Administration › Monitoring › Reaction Logs**, with each action's input, output and error. See Reaction Logs [Administration › Monitoring: logs & audit].
- **On the record** — a view can show reaction runs in a record's history. In the view's **Entry history** settings, turn on **Reaction history** (and **Reaction history details** to include each action's input, output and error). Both are off by default.

Problem	What to check
The reaction never runs when a record is approved	You used **On Entry Updated**. Use **On Workflow Action Executed** or **On Workflow Step Reached** — see Choose the event.
It runs for some records but not others	The filter does not match those records. A run skipped by the filter is not logged at all.
It ran once but not the second time in the same chain	The loop protection skipped it — see **Loop Threshold** above.
A follow-up reaction never runs	Check it names the right entity and reaction, is active, and uses **Triggering Space**; and that the first reaction actually succeeded.
An action fails with a path error	A **Field** value points to a field that does not exist in the record that set it off. Pick the field again.
You cannot save a reaction	Check you have Developer access on the entity, that a permission action grants no more than your own access, and that required settings of the action are filled in.

Worked example: on approval, email the requester and update the asset

An **Asset Request** has a reference field *Asset* and a workflow whose *IT approval* step has the action **Approve**. When IT approves, the requester should get an email and the asset should be marked *Assigned*.

Reaction 1 — email the requester.

1. On the Asset Request entity, open **Reactions**, stay on **New Reaction**, and select **+**.
2. **Name**: *Email requester on approval*.
3. **Event**: **On Workflow Action Executed**; choose the *IT approval* step and its **Approve** action.
4. **Action Type**: **Send Email**. **To**: the record's *Requester* field. Choose your *Request approved* template, and turn on **ADD Entry Link** so the email links back to the request.
5. Confirm the dialog.

Reaction 2 — mark the asset as assigned.

1. Select **+** again. **Name**: *Mark asset assigned on approval*.
2. **Event**: the same — **On Workflow Action Executed**, *IT approval*, **Approve**.
3. **Action Type**: **Update Data Entry**. **Choose update mode**: **Reference**, then the *Asset* field.
4. Set *Status* to **Manual** *Assigned*, and *Assigned to* to **Field** *Requester*.
5. Confirm the dialog.

Test it. Submit a request as a test user, approve it as the IT owner, then check the inbox and the asset record. If either did not happen, open the reaction's counters or **Reaction Logs**.

If the asset must be updated only *after* the email went out, give Reaction 2 the event **Reaction Succeeded** pointing at Reaction 1 instead — see Chain reactions.

Go deeper

For integrators: Reaction model and API reference [Integrate & Extend] — event and action enums, every action's options, chaining contracts, run history and the reactions endpoints · For operators: Configuration reference · For platform engineers: Engine: Reactions

Buttons

This guide is for **builders** who want people to act on a record in one click. By the end you will be able to add a button to an entity, decide what it does — run an automation, open a pre-filled form or another page, or run a short script — ask for confirmation first, limit which records it is available on, and show it in the views people use.

How buttons work

A **button** belongs to one entity and appears on that entity's records, next to the buttons the platform provides (**View**, **Edit**, **Create**, **Duplicate**, **Archive**). You design it once on the entity's **Buttons** tab, publish it, and then choose in each view whether it is shown.

Every button does one of three things, chosen in its **Action**:

Action	What happens when someone clicks it	Example
A **Custom Action** (listed by name)	Runs one of the entity's custom actions — an automation you built on the **Reactions** tab. It can first ask the person for some input, and afterwards take them to another page.	*Close request*: sets the status, records the closing date and emails the requester.
Navigation	Takes the person somewhere: a list, a record, a create or edit form (optionally pre-filled), or a web address. Nothing is saved by the click itself.	*Create follow-up task*: opens a new *Task* form already linked to this request and assigned to the person who clicked.
Script	Runs a short script in the person's browser.	Show a calculated warning before someone continues.

A button's **Scope** decides whether it works on **one record** (it appears on the record and on each row), on **several selected records** at once, or both.

Take a **Service Request** entity: a *Close request* button runs a custom action on one request or on a whole selection; a *Create follow-up task* button opens the *Task* form pre-filled from the request; both appear only on requests whose status is *Open*.

Before you start: build the custom action

A button that runs automation needs a **custom action** first. On the entity's **Reactions** tab, add a reaction with the **Custom Action** event and keep its **Custom action level** at **UI action** — only UI actions can be attached to a button. If the action needs input from the person clicking (a reason, a date), turn on **Ask User For** in the custom action. See Custom action type and Contextual.

Navigation and script buttons need no reaction.

Add a button

1. Open the entity: **Administration › Applications › Data Entities**, then select the entity. (You can also reach it from its module under **Administration › Applications › Modules**.)
2. Select the **Buttons** tab, then **New Button** (or **Add Button** if the entity has none yet).
3. Fill in the dialog:
    - **Name** — the label people see, in Arabic and English. It must be unique among this entity's buttons.
    - **Icon** — the icon shown on the button.

- **Description** — optional.
- **Action** — under **Custom Action**, pick one of the entity's UI custom actions; under **UI Action**, pick **Navigation** or **Script**. The sections below explain each.
- **Scope** — **Single Entry**, **Multiple Entries**, or both.

4. Select **Save**.
5. Back on the **Buttons** tab, select **Publish Button** on the button and confirm. Until it is published, a button cannot be used.
6. Show the button in a view — see Show the button in a view.

To change a button later, select it on the **Buttons** tab. **Unpublish Button** takes it out of use without deleting it; **Delete Button** removes it permanently.

Run a custom action

1. In **Action**, pick the custom action under **Custom Action**.
2. Optionally open **Customize Navigation For Single Entry** to decide where the person goes afterwards. Set **Navigation On Success** and **Navigation On Failure** separately; for each, choose a **Navigation Type** (see the table in Open a form, a list or a link).

When someone clicks the button, the custom action runs on the record — or on every selected record — and a message tells them when it has finished. If the custom action asks for input, a dialog collects it first. The record's history shows that the button was clicked.

Open a form, a list or a link

1. In **Action**, pick **Navigation** under **UI Action**.
2. Choose the **Navigation Type**, then fill in what it asks for:

Navigation Type	Takes the person to	You also choose
Same View	Somewhere in the current view	**Navigation Action**: **Entries List**, **Create Entry**, **Show Entry Details** or **Update Entry**
Relation	Records linked to this one through a relation	**Relation**, then a **Navigation Action**
Entity	A new record of another entity	**Entity**; the **Navigation Action** is **Create Entry**
Custom URL	A web address	**URL** — it can contain variables, see below

3. When the navigation opens a create or edit form, **Advanced Options** lets you pre-fill it — see Pre-fill the form.

Put record values in a link `Since 1.8`

The **URL** of a **Navigation** button can contain variables in double braces, such as `https://example.com/profile?id={{userId}}&dept={{dept}}`. Each variable appears under **Custom Variables**; for each one choose a **Type**, then what it maps to:

Type	The variable becomes
Fixed	A value you type.

Type	The variable becomes
Field	The value of a field on the record.
Expression	The result of an expression on the record.
Pointer	Something about the person clicking, such as **@me**, their groups, roles or organizational unit.

The variables are filled in at the moment of the click. If one cannot be filled — for example its field was deleted — the link opens without that value instead of showing an error, so test the link after you change the entity.

Pre-fill the form

When a button opens a create or edit form, it can fill some fields for the person. Open **Advanced Options** and add rows under **Fields Mapping**. For each field, choose how its value is set:

Value type	The field gets
Field	The value of a field on the record the button was clicked on.
Manual	A value you type.
Not Set	Nothing — the field is left empty.
Pointer	Something about the person clicking, such as **@me** or their groups.
Expression	The result of an expression, for example a date three days from now.

Turn on **Override locked value** to lock the pre-filled fields so the person cannot change them in the form.

Example — *Create follow-up task*. On a *Service Request*, a **Navigation** button with **Navigation Type Relation** (the request's *Tasks*) and **Navigation Action Create Entry**, mapping *Related request* = **Field** (this request), *Assignee* = **Pointer** (**@me**), *Status* = **Manual** (*Open*) and *Due date* = **Expression** (three days from now).

Keep expressions short and test them before you publish the button.

Run a script `Since 1.8`

A script button runs a few lines of JavaScript in the person's browser — useful for a check, a calculation or a message that no other option covers.

1. In **Action**, pick **Script** under **UI Action**.
2. Choose the **Scope**.
3. Write the script in **Script Code**. The script receives the current entity, view and record; for a **Multiple Entries** button it receives every selected record.
4. **Save**, then publish the button.

A script button runs in the browser only — it saves nothing on its own and does not appear in the record's history. If you need the platform to change data, use a custom action instead.

Technical details

The script is called with one argument, `context`, which exposes `entity`, `view`, `entry`, `entries` and `popupManager`. For a multi-entry button, `context.entries` holds every selected row.

Ask for confirmation first `Since 1.8`

1. Under **Before Execution**, turn on **Show confirmation dialog before execution**.
2. Fill in **Dialog title** and **Dialog content**, and optionally the **Proceed button label** and **Cancel button label**.
3. To show values from the record in the dialog — *"Close request {{requestNumber}}?"* — put variables in double braces in the title or content and map each one under **Custom Variables**, the same way as for a link (above).

Variables in the dialog are available only when the **Scope** is **Single Entry** alone. With **Multiple Entries** selected you can still show a dialog, but its text is fixed.

Limit which records the button is available on

Under **Filter**, choose a saved query, or write conditions directly, or both (a record must then match both). Then choose what happens on records that do not match in **For other entries**:

- **Disable button** — the button stays visible but cannot be clicked (the default).
- **Hide button** — the button is not shown.

With several records selected, the button works only when **every** selected record matches. To learn how to write a query, see Queries.

Show the button in a view

A published button appears only in the views you add it to:

1. Open the entity's **Views** tab and open the view.
2. In the view's **Buttons** section, select the button. The list also holds the platform's own buttons and the entity's export templates; drag the rows to set the order.
3. For each selected button you can set a **Display name** for this view, a **Conditional display** expression, and — for buttons that open another entity — the **View** to open it in.
4. **Save** the view.

A button whose **Scope** is **Single Entry** needs the view's entry details screen; a **Multiple Entries** button needs its entries list.

Good to know

- Only published buttons appear and run.
- A custom action button can only run a **UI action**. If a button was attached to a local or external custom action — for example because that action's level was changed later — clicking it is refused. Local and external custom actions are run by other reactions through **Execute Custom Action**, never by a click.
- A custom action runs as the person who clicked unless the reaction says otherwise.
- In preview mode the view shows its real buttons, but they cannot be run against the sample records.

Troubleshooting

Problem	What to check
The custom action is not in the **Action** list	It must be a **Custom Action** reaction on the same entity with **Custom action level UI action**, and it must be enabled and valid. **There's no custom events** means the entity has none.
The button does not appear on a record	Is it published? Is it selected in this view's **Buttons** section? Does the record match the **Filter** (with **Hide button**, it is hidden on non-matching records)?
The button is greyed out	The record — or one of the selected records — does not match the button's **Filter**.
"You can't execute this action on all selected entries"	Some of the selected records do not match the filter. Narrow the selection.
The dialog shows `{{…}}` instead of a value	Dialog variables work only with **Single Entry** scope, and each variable must be mapped under **Custom Variables**.
"The button name is unique on the entity level"	Another button on this entity already has that name in one of the languages.

Go deeper

For integrators: Buttons API reference [Integrate & Extend] — stored model and enums, validation rules, the read, availability and execute endpoints, and execution behaviour

Export templates

This guide is for **builders** who want people to download records as a formatted document — a purchase order as a PDF, a contract in Word, a monthly list in Excel. By the end you will be able to write a template file, add it to an entity, make it available in a view or an automation, limit it to the right records, and keep large Excel exports fast.

How export templates work

An **export template** is a document you prepare in advance — a Word file, an HTML page or an Excel sheet — with placeholders where record values go. You upload it to an entity. When someone exports with it, OSOS fills the placeholders with the record's values and hands them the finished file.

Take a **Purchase Request** entity. A *Purchase Order* template in Word holds the company letterhead, a table of items and a signature block. A buyer opens an approved request, selects **Purchase Order**, and downloads a PDF with the request's supplier, items and total filled in.

Each template has a **Level**:

- **Single Entry** — one document for one record (the purchase order).
- **Multiple Entries** — one document for several records at once (a monthly list of all requests).

Templates are used in three places:

- **In a view** — people export from the records list or from a record's details. See Make the template available.
- **In an automation** — the **Generate File** action of a reaction produces the file automatically, for example to attach it to an email.
- Template exports produce Word, PDF or Excel files only. For a quick CSV or Excel dump of a list, use the view's quick export instead (see Data Views).

Choose the type

Type	You upload	People receive
Word	A Word file (`.docx`)	A Word document
Pdf	A Word file (`.docx`) or an HTML file (`.html`)	A PDF
Excel	An Excel file (`.xlsx`)	An Excel workbook

Prepare the entity first: the fields your template uses must exist, and the names in the template must match the fields' identifiers exactly (for example `title`).

Write the template file

In every template, `entries` is the list of records being exported and `entry.value.<field>` is one field of one record. A **Single Entry** export still passes a list — with one record in it.

HTML (for PDF)

Use `{{% … %}}` for logic, such as a loop, and `{{%= … %}}` to print a value:

```
<!DOCTYPE html>
<html>
<head>
    <title>Requests</title>
</head>
<body lang="en-US" dir="ltr">

    {{% for(entry of entries){ %}}
        <h1>Title</h1>
        <p>{{%=entry.value.title %}}</p>
    {{% }; %}}

</body>
</html>
```

HTML is the only type that can export records containing decorated (formatted) fields.

Word

Define a short alias for a field, then print it inside a loop:

```
<{{ALIAS title INS $entry.value.title}}>

<{{FOR entry in entries}}
{{* title}}
{{END-FOR entry}}>
```

- `ALIAS title INS $entry.value.title` names the value `title` for use in the loop.
- `{{FOR entry in entries}}` … `{{END-FOR entry}}` repeats its content once per record.

Excel

Put a placeholder in a cell; the column is filled with that field from every record, one row per record:

```
<${table:entries.value.title}>
```

A value that starts with `=` is written as text, not as a formula. To write a real formula, build it with `formula()` (see the functions below).

Functions you can use

These functions are available in every template and in variables (below):

Function	Gives you
`today()`	Today's date in your timezone; pass an amount to shift it.
`date(value, format)`	A date formatted the way you want, for example `date(entry.value.needed_by, 'dd/MM/yyyy')`.
`loc(value)`	The text of a two-language value in the reader's language.
`mdToText(value)`	Formatted (markdown) text as plain text.
`formula(text)`	An Excel formula cell (Excel templates).
`fetchEntry(ref)`, `fetchEntriesById(entity, ids)`, `fetchEntries(entity, filter)`	Related records, to print their values.

Function	Gives you
`getStepName(entry, stepId)`, `getCurrentStepOwner(entry)`, `getStepOwner(entry, stepId)`, `getStepReceivedAt(entry, stepId)`	Workflow information: a step's name, who owns the current step, who owned a step, when a step was reached.
`getStepActions(entry, stepId)`, `getStepActionName(entry, stepId, actionId)`, `getStepHistory(entry, stepId)`, `getStepLastHistory(entry, stepId)`	The actions and the history of a workflow step.
`getEntryComments(entry)`, `getEntryEmails(entry)`	The record's comments and emails — **Single Entry** templates only.
`getImage(…)` (Word), `getImageSrc(…)` (HTML)	An image field, embedded in the document.
`mdToHtml(value)` (Word and HTML)	Formatted (markdown) text as formatted document text.

Add the template to the entity

1. Open the entity: **Administration › Applications › Data Entities**, then select the entity.
2. Select the **Templates** tab, then **New Template** (or **Add Template** if the entity has none yet).
3. Fill in the form:
    - **Name** — also the default name of the downloaded file. It must be unique among this entity's templates.
    - **Level** — **Single Entry** or **Multiple Entries**.
    - **Type** — **Excel**, **Pdf** or **Word**.
    - **Sensitive Data Access Policy** — shown only if the entity has sensitive fields; see Sensitive fields in exports.
    - **File** — select **Choose file** and upload the template you prepared.
    - **Export Type** — **UI Export** builds the file straight away and downloads it; **Background Export** builds it in the background and tells the person when it is ready. Choose **Background Export** for large exports.
4. Optionally, for a **Single Entry** template, set a condition.
5. Optionally, open **Advanced Options** to add variables.
6. Select **Save**.

Make the template available

A template on the entity is not yet offered to anyone. To let people export with it from a view:

1. Open the entity's **Views** tab and open the view.
2. Under **Templates** (in the entries-list settings for **Multiple Entries** templates, and in the entry-details settings for **Single Entry** templates), tick the templates people may use in this view.
3. In the view's **Buttons** section, the templates appear alongside the other buttons — drag them to set their order.
4. **Save** the view.

To produce the file automatically instead, add a **Generate File** action to a reaction and pick the template.

Export only some records `Since 1.10`

A **Single Entry** template can carry a condition, so that only records that meet it can be exported with it — for example, a *Purchase Order* only for approved requests.

1. In the template form, under **Filter**, choose a saved query, write conditions, or both.

2. In **For other entries**, choose what people see on a record that does not meet the condition:
    - **Disable export option** (the default) — the template is listed but greyed out, with *"This entry doesn't meet the conditions to export using this template"* on hover.
    - **Hide export option** — the template is not listed for that record.
3. **Save**.

Multiple Entries templates cannot have a condition. A conditioned template can only be used on records people have selected, not on a whole list. The **Generate File** reaction action does not check template conditions.

Variables

Variables let you calculate a value once and use it in the template like a field. Open **Advanced Options** and write them in **Variables** as a list of names and expressions:

```
{
  "full_name": "entry.value.first_name + ' ' + entry.value.last_name",
  "due": "date(entry.value.needed_by, 'dd/MM/yyyy')"
}
```

Each expression is worked out for every record, and the result is available as `entry.value.full_name`, `entry.value.due` and so on. Expressions can use the functions above. You can also wrap an expression in `{{%= … %}}`; the wrapper is ignored.

Technical details

The name `extended` is reserved. Setting `"extended": "true"` switches the template to an *extended* export that also receives the entity's own design — its module, reactions, notifications, workflows, lists, queries, views and relations — for producing documentation of an entity's configuration. An extended export always covers the whole source and always runs immediately.

Large Excel exports: keep the sheet simple `Since 1.4`

A large Excel export is much faster — and the file much smaller — when the template is simple enough to be written a page of records at a time. Your template qualifies when **all** of these hold:

- It has exactly one sheet.
- Every placeholder fills a whole cell, is a plain `${table:entries.…}` placeholder, and all of them sit on the same row, with nothing else on that row.
- Nothing below that row has a value, and no merged cells reach that row or below.
- The sheet has no formulas, images, Excel tables, conditional formatting, data validation, filters or named ranges.

A template that does not qualify still works; it is just slower on very large exports. If an export of many thousands of records is slow, simplify the sheet (move headings and totals above the table row) or ask your operator which rule the template breaks.

Templates on virtual entities `Since 1.6`

The **Templates** tab is also available on virtual entities, such as Groups, when the entity may be edited. A template export over a virtual entity includes the rows the person selected.

Sensitive fields in exports

If the entity has sensitive fields, choose a **Sensitive Data Access Policy** for the template:

Policy	Sensitive values in the document
Privileged	Always shown. This is what a template without a policy does.
Prohibited	Never shown — replaced by `****`.
User-Based	Shown only if the person exporting may see them.
Auto	Shown when an automation generates the file; decided by the person's own access when a person exports.

The template's own policy applies even when a reaction generates the file. Fields set to **Encrypt** are never shown in a document, whatever the policy. See Sensitive-data access policy.

Good to know

- **Exporting from a view uses the view's columns.** A template field that the view does not display can come out blank. Add every field the template uses to the view (you can hide columns people do not need on the list face), or export from a view that shows them.
- The downloaded file is named after the template unless the export gives it another name.
- Comments and emails can be printed only in **Single Entry** templates.

Troubleshooting

Problem	What to check
A value is blank in the document	The field identifier in the template must match the entity exactly. If you exported from a view, the view must display that field.
The template is not offered in a view	Tick it under the view's **Templates**, and check its **Level**: Single Entry templates appear on a record, Multiple Entries templates on the list.
The template is greyed out on a record	The record does not meet the template's condition.
"The template name is unique on the entity level"	Another template on this entity already uses that name.
Sensitive values show `****`	The template's **Sensitive Data Access Policy** masks them, or the field is Encrypted.
A PDF template produces no file	A PDF template must be uploaded as `.docx` or `.html`.

Go deeper

For integrators: Export templates API reference [Integrate & Extend] — the stored template model, rendering engines, display-template parameters, the export and status endpoints and their errors, background and extended export, and the version notes

Export, import and install apps

This guide is for **builders** who need to move an application they built — for example from a test environment to production — and for the administrators who install it at the other end. By the end you will be able to export an app as a file and import it elsewhere, turn it into a versioned package, send a version for review and publish it to the store, and install, update, fork or uninstall apps.

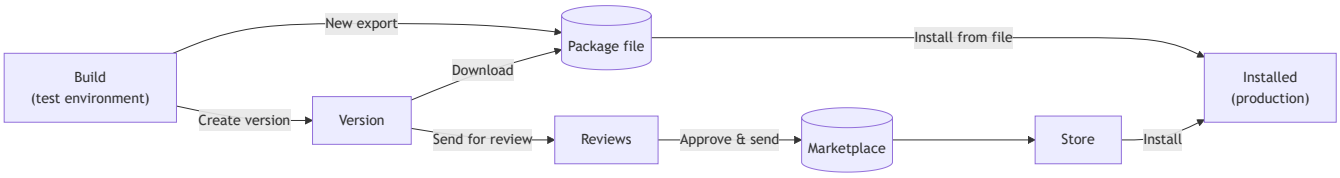
How it works

You build an *Asset Management* app in your test environment. When it is ready, you need the same modules, entities, views, workflows and pages in production — without rebuilding them by hand. There are two ways to do that:

	A file (export and import)	**A package with versions** `Since 1.10`
Good for	A one-off move, or copying sample content	An app you will keep improving and ship again
What you get in the target	Editable copies, with a note of where they came from	An installed app that can be **updated** to later versions; its parts are locked so updates apply cleanly
How it travels	You download a file and import it	You download a version file, or publish it to the **store** so other environments install it from there

Both are managed in **Administration › Packages**, which has four screens:

Screen	**What it is for**	**Who sees it**
Store	Browse apps published to the marketplace and install them	When your operator has turned the marketplace on, and you may install apps
Installed	Everything that came into this environment — store installs, package files and imported exports — and **Install from file**	System administrators
Build	Your packages (**Packages** tab) and your exports (**Exports** tab)	Developers of a module, and marketplace publishers
Reviews	Versions and store pages waiting for your organisation's approval	Marketplace reviewers, when the marketplace is on



What travels, and what does not

An export or a version carries the **design** of your app: modules, entities and their fields, views, queries, relations, reactions, reports, pages, buttons, lists, summaries, counters, functions, announcements, aggregates, and quick-filter and tag groups. It also carries what those parts depend on, and a copy of the records your settings point to — a workflow step's owner group, for example.

It does **not** carry the records people entered, archived entities, anything personal to one user, or anything the platform generates for itself. Users, groups and organization units are not copied either: in the target you match each one to an existing person or group (or let the import create it).

A function's **secret values** never travel. The package carries the secret names only; an administrator of the target fills in the values.

Before you start

To...	You need
Export, create packages and versions	**Developer** access to the module
Import a file, see **Installed**, fork an installed app	A system administrator in that space
Update or uninstall an installed app	A system administrator, working in the root space (updating also needs **Marketplace installers**)
Install from the **Store**	The marketplace turned on by your operator, the root space, and membership of **Marketplace installers** (system administrators can always install)
Send a version or store page for review	Membership of **Marketplace publishers**, and a plan that includes publishing to the marketplace
Approve or return a review request	Membership of **Marketplace reviewers**

Your tenant administrator manages these groups in **Administration › Organization › Users Groups**. Publishing and reviewing always need the group — being a system administrator is not enough.

Move an app as a file

Export

1. Open **Administration › Packages › Build**, open the **Exports** tab, and select **New export**. The **Export resources** page opens.
2. In the left panel, tick what to export. Ticking a module or an entity takes everything inside it; you can also tick single views, queries, pages and so on. Items installed from another package are marked as locked — they cannot be exported as a copy.
3. Check **External dependencies** in the summary on the right. These are things your selection uses but does not include, such as an entity in another module. Select **+ Add** next to one, or **Add all dependencies**, to take them along. Anything you leave out has to be matched to something in the target.
4. Enter a **Name**, for example *Asset Management — release 3*. It is shown wherever this file is imported.
5. Select **Export All**. When the status reads **Export ready**, select **Download file**.

Every export is listed on the **Exports** tab, where you can open it again and export the same selection once more. Deleting an export removes it from the list only. The downloadable file is temporary, so download it soon after the export finishes.

> **Tip**
>
> To export one module quickly, open the module and choose **Export a copy...** from its actions. You can also start a new export from an earlier one by dropping its export configuration file into the **Export resources** page.

Import

1. In the target environment, open **Administration › Packages › Installed** and select **Install from file**.

2. Drag the file onto the page, or select **Browse files**. The file is checked and the page tells you what it is:

    - a **plain export** — everything it adds stays editable in this space;
    - a **version of a published package** — it is installed, stays updatable, and its parts are locked;
    - a **newer version of an installed app** — running it updates that app.

3. Review the list of resources. Each one is marked **New** (it will be created) or **Existing** (something with the same name is already here).

4. Choose a **Strategy**:

    - **Manual** — new resources are created; you decide for each existing one. Tick an existing resource to overwrite it with the version in the file.
    - **Merge (new only)** — only add what is missing; existing resources are kept as they are.
    - **Overwrite all** — everything in the file wins; existing resources are replaced.

5. Resolve what the file needs but does not contain. For each missing item, choose an existing resource in this environment to **Map to**. You cannot run the import until every required one is mapped.

6. Decide what happens to the **records the file refers to** — users, groups, organization units and other referenced records:

    - **Keep** — leave the reference as it is (safe when the same record exists here);
    - **Remap** — point it at a different record in this environment;
    - **Clear** — remove the reference;
    - **Auto Create** — create the record here from the data in the file.

    Select **Review suggested matches** to accept the exact matches the platform found. A reference marked **Profile** was restored from what you chose the last time you imported from the same source.

7. If an entity has a **mapped field** whose source field cannot be found here, it is tagged **Choose a field**. Choose the item the field's parent maps to first, then pick the field it should copy from, or choose **Don't copy**. Decided rows show **Your choice**; rows the platform matched by name show **Matched**. If the parent cannot be read, the row shows **Couldn't read the entity** — try again in a moment.

8. Select **Execute Import** (for a package version the button is **Install**). A progress bar shows each resource as it is processed.

9. Read the result. **Import Complete** means everything landed. **Import Completed with Errors** lists what failed: fix the cause and select **Retry**, which re-attempts only what did not land, or **Fix Mappings** to change your choices first.

> ⚠ **Caution**
>
> **Overwrite** replaces the existing resource. Anything in it that is not in the file is removed.

Technical details

- An import is kept for seven days; after that, start again from the file.
- A resource skipped because something it depends on was skipped or failed is shown as **Cascade Skipped**.
- **Edit JSON** on a resource lets you change its definition before it is imported (up to 2 MB). It is meant for integrators; a package version cannot be edited this way.

Package your app with versions `Since 1.10`

A **package** is a named, permanent home for one or more modules (and, if you like, pages that belong to no module). Each **version** is a numbered snapshot of it that other environments install and later update to.

Create a package

1. Open **Administration › Packages › Build**, stay on the **Packages** tab, and select **Create package**. (From a module, **Add to package…** does the same.)
2. Give it a name and choose its **Modules** and **Pages**. A module can belong to only one package.
3. Open the package. Its **Contents** tab lists what the next version will hold: **Modules**, **Referenced records**, **Pages** and **Depends on** — with what changed since the last version.
4. Edit the name and description in place, and write the **README** — the long description shown in the store. Add the README **before** you create a version: a version keeps the README it was created with.

If your app needs another package — a shared *Employees* app, say — add that package as a dependency rather than copying its modules. Anyone installing yours must then install that one first.

Create a version

1. On the package page, select **Create version**.
2. Pick the kind of change — **Small fix**, **New features** or **Big change** — and the platform suggests the next number. You can type your own, as long as it is higher than every earlier version.
3. Write the **Release notes** (required, in at least one language).
4. Optionally tick **Pre-release** (never offered as the latest version) and raise the **Minimum OSOS version** — it cannot be lower than this environment's.
5. Read any **External dependencies** listed: these are resources outside the package that whoever installs it must map to something of their own. To avoid that, move the resource into one of the package's modules, or into a package this one depends on.
6. Select **Create version** and keep the dialog open until it finishes.

A version is fixed once created. To change anything, create a new version. You can **Delete version** only while it has never been sent to the store or installed, and its number is never reused. **Compare with previous** shows what changed between versions.

To install the version somewhere without the store, select **Download** and use **Install from file** in the other environment.

Publish a version to the store `Since 1.10`

When the marketplace is turned on, a version can be published to the **store**, where other environments find and install it. Nothing reaches the store without a second person in your organisation approving it, and then the marketplace team. The package page shows where you are: **Contents**, **Version**, **Store page**, **Organisation review**, **Marketplace review**, **In the store**.

Set up publishing (once per environment)

On the **Build** page, the **Set up publishing** card asks for the **publisher name** (shown on every app you publish) and an optional **store address**. Both are public. A system administrator or a marketplace publisher can do this; after that, only a system administrator can rename the publisher or change the address, and the address is locked once the store has approved your first app.

Fill in the store page

Open the package's **Store page** and fill in:

- **Name in the store** — follows the package name until you first save it; it cannot change after the store approves it.
- **Description** — the package description, up to 120 characters.
- **Website**, and up to 10 **Topics** that help people find the app.
- **Icon** — a square PNG, JPEG or WebP, 256 to 1024 pixels, up to 2 MB.
- **Screenshots** — up to 8, 16:9, at least 1280×720, up to 2 MB each, with captions.
- **Under maintenance** and a **Note for users**, when you need them.

Preview shows how the app will appear in the store as you type.

Send it for review

1. Select **Send for review**. The first time, the version and its store page are sent together and reviewed in one step.
2. Your organisation's reviewers are notified. Until someone decides, you can **Withdraw from review**.
3. If a reviewer **returns** it, you get their note. Make the change and send it again.
4. Once approved, it goes to the marketplace team. You are notified — in the app and by email — when they list or reject it. A rejected version cannot be sent again: create a new version.

Unpublish takes the app out of the store; environments that installed it keep it. **Republish** brings it back.

Review a request (reviewers)

1. Open **Administration › Packages › Reviews**. Each row shows the package, version, what was sent, who sent it and when.
2. Open a request. **What changed** compares it with the previous version; names link to the original modules, entities and pages in this environment.
3. Select **Approve & send** to pass it to the marketplace team, or **Return** and say what needs to change. The first time you approve, you are asked to accept the marketplace terms.

You may approve a request you sent yourself; the history records it.

Install an app from the store `Since 1.10`

1. Switch to the root space and open **Administration › Packages › Store**. Search, or filter by **Publisher** or **Topic**.
2. Open an app. Its page shows an overview, **What's inside**, **Screenshots**, **Details** and the **Changelog**. **Needs setup during install** lists the things you will match to items in this environment.
3. Select **Install**. The first time, the **Connect to the marketplace** dialog shows what connecting shares; select **Connect and install**. Connecting publishes nothing.
4. If the install needs choices that only an administrator can make, the page lists them. A system administrator completes them on the import page and selects **Install**.

If the app cannot be installed, the page says why: it needs a newer OSOS version (**Not compatible**), something in this environment already has the same name as one of its items (rename or delete it, then **Check again**), or the app was built here (**Published here**). An app can be installed only once per environment.

Update an installed app

When a newer version is listed, the app shows **Update available**.

1. Select **Update** on the app's store page or its installation page.
2. Read the preview: **What's new**, **What changes**, **Your data**, **Your customisations**, **Needs your choice** and **Requirements**.
3. Tick **I've read what changes for our data and our own work**, then select **Update to** the new version.

Your fields, hidden items and your own names for views and pages are kept. If the update changes an entity so that its existing records must be upgraded, they are listed under **Entries to upgrade**; select **Upgrade records** (this resets their workflow). An update that stopped part-way continues from where it stopped when you select **Retry update**.

Work with an installed app

An installed app's parts are **locked** so that updates apply cleanly. Its modules, entities, views and pages show *Part of …. Its parts are locked so updates apply cleanly.* and a **View installation** link. You can still:

- add your own fields to its entities;
- hide or turn off what you do not use;
- give its views and pages your own name, description and icon — kept on every update.

The installation page

Open an app from **Administration › Packages › Installed** (filter by **Status** and **Source**). Its page lists the **Installed resources** — each **Unchanged**, **Changed** or **Missing** — what it **Depends on** and what it is **Used by**, and its history.

- **Fork…** makes the app's modules your own and fully editable. In exchange, it stops receiving updates for good, can never be published or exported, and the fork cannot be undone. Type the package name to confirm. You cannot fork while another installed app depends on it, or while an update is running.
- **Uninstall** deletes the app's modules, entities and their records. The dialog lists what is deleted, your own work that will break, and every link from outside the app; **Uninstall** stays disabled until those links are removed. Tick the confirmation and select **Uninstall**. If it stops, select **Retry**.

A failed install

The installation page shows how to recover: **Open import session** to retry while the import still exists, or **Upload the package file again**. When a newer compatible version exists, updating to it finishes the installation instead. For a store app, **Report to *publisher*** sends the error and the failed resources to the app's publisher, with an optional note `Since 1.10`.

Good to know

- **One package per module.** A module already in a package cannot join another. A module installed from a package cannot be added to yours; add that package as a dependency instead.
- **A plain export is a copy.** Importing it does not create an updatable app; to ship updates, use a package and versions.
- **Archived names cannot be reused.** If an item in the file has the name of an archived resource in this space, it cannot be imported there.
- **Installed parts cannot be overwritten by an import.** Fork the app first if you need to change them that way.
- **Your plan's record limit applies.** If the import would create more records than your plan allows, it is refused before anything is written. Archive or delete records, or upgrade the plan, and try again.

Troubleshooting

You see	What to do
This import was cancelled and cannot run again.	Start a new import from the file.
Choose an action for every referenced record that needs one…	Give each highlighted referenced record an action (Keep, Remap, Clear or Auto Create).
A referenced record already exists in this space, so Auto Create would duplicate it.	Remap it to the existing record.
A resource in this import was installed from package …	That resource belongs to an installed app. Skip it, or fork the app first.
Choose what each mapped field copies from before installing.	Find the rows tagged **Choose a field** and decide each one.
This file isn't a valid package archive.	Check that you chose the right file, then upload it again.
This import was changed in another tab	Someone else saved changes to the same import. Select **Reload** and continue.
Lost connection to the import	The import is still running. Select **Reload** to see its progress.
Validation succeeded, but the reference mappings could not be loaded.	Reload the page to see them; your import is intact.

Go deeper

For integrators: Packaging & distribution [Integrate & Extend] · The import API [Integrate & Extend] · The package format [Integrate & Extend › The Package Format] · Packages, marketplace & the store API [Integrate & Extend › Packages, Marketplace & the In-App Store] · For operators: Turning on the marketplace connection

Counters and record numbers

This guide is for **builders**. The platform uses the word *counter* for two different things, and this page covers both:

- A **counter widget** shows people a live count in the top bar — *12 open requests*, *3 tasks due today* — and lists the records behind it.
- A **sequence field** gives every new record a running number — a request number, an invoice number, a ticket ID.

	Counter widget	Sequence field
Answers	How many records match right now?	What is this record's number?
Built from	A view, optionally narrowed by a saved query	A field on the entity
Changes	Goes up and down as records change	Given once, when the record is created

Show a live count with a counter widget

A counter widget is a badge in the top bar, next to notifications. Selecting it shows the count and the matching records; selecting a record opens it in the view the counter is built on. Counters need no programming and keep themselves up to date.

Before you start: the view the counter counts must have the **Counter view** display turned on in its settings (see Data views). If you want to count only some records — *open* requests, say — save a query on the entity first.

1. Open **Administration › Applications › Counters**.
2. Select **New Counter**.
3. Enter a **Name** (Arabic and English), choose an icon and, optionally, add a **Description**.
4. In **View**, choose the view whose records are counted.
5. Optionally, in **Counting Criteria**, choose a saved query of that entity. The counter then counts only the records that match it.
6. Save. The counter appears in the top bar. Who sees it follows the rules in Counter widgets.

To change a counter, edit it in the list and save. To change the order of counters in the top bar, drag them in the list; by default they are in the order they were created. To remove one, select it and use **Delete Counter**; this cannot be undone.

Counts also appear as **Number** cards on dashboards and as counter widgets on pages — see Dashboards & summaries and Pages & widgets.

Number records automatically with a sequence field

Add a **Sequence** field to an entity to give each new record the next number in line.

1. Open the entity's designer and drag a **Sequence** field onto the **Structure** tab.
2. Give it a name, for example *Request number*.
3. In **Auto increment settings**, set the **Starting value** (for example 1000) and **Increment by** (usually 1).
4. Select **Save**.

The first record gets the starting value, and each new record after it gets the previous number plus the increment: 1000, 1001, 1002…

Restart the numbering per year, department or branch

Turn on **Conditional Sequencing** when each group of records should have its own run of numbers — a fresh run each year, or a separate run per branch. Add a **Key** with a **Type** and a **Value** that works out the group from the record — for example the year the record was created. Each different key value starts its own run at the starting value.

With conditional sequencing the same number can appear once in each group, so the field is no longer required to be unique across all records.

For a reference such as *LC-2026-0001* that includes the year and leading zeros, combine a year-keyed sequence with a second field that builds the text — see Formatted references.

Good to know

- **Numbers are given when a record is created**, never when it is edited. Two people creating records at the same moment never get the same number.
- **Decide before you go live.** Once the first number has been given, the starting value, the increment and the conditional-sequencing keys are fixed. To change them, add a new sequence field.
- **A number is just a number.** Prefixes and leading zeros come from a second field, as in the formatted-reference recipe above.
- **If a number cannot be given**, the record is still saved, with the sequence field empty. Check the field's settings if new records arrive without a number.

Troubleshooting

Problem	What to check
A counter does not work with a view	The view must have its **Counter view** display turned on.
A counter shows a different number than the view	The counter's **Counting Criteria** narrows it, or the person's permissions hide some records from them.
New records have no number	The sequence field's settings are incomplete or invalid; the record is saved without a number rather than refused.
The starting value cannot be changed	Numbers have already been given from this sequence.

> **Go deeper**
>
> For integrators: Auto-increment sequences and counter widgets: internals and API [Integrate & Extend] — the counter and sequence model, allocation, save-time rules and the counters endpoint

Timeline & History

This guide is for **builders** who decide what people can see about the past of a record, or who want to share records as a news-style feed. By the end you will be able to choose, view by view, how much of a record's history people see, and to turn an entity's records into **timeline posts** that are published, scheduled or pinned and shown on a page.

How it works

Two features answer "what happened, and when?":

	Record history	**Timeline**
What it is	The activity log of one record: who changed what, workflow actions, button clicks, automations that ran.	A feed of **posts** made from an entity's records — announcements, news, events.
Where people see it	The **History** tab on a record's details.	A **TimeLine** widget on a page.
What you set up	Nothing to record it — every record has a history. You choose, per view, how much of it people see.	You add a timeline to the entity, decide how a record becomes a post, and who sees the posts.

Example: an *HR Announcements* entity has a timeline. Each announcement an HR officer creates becomes a post; it goes live when the announcement's workflow reaches *Approved* and its publish date arrives, and it appears on the intranet home page. On the record itself, the **History** tab shows the edits and approvals that led there.

Record history

What the history records

Every record keeps a history of:

- **Field changes** — the old and the new value of each changed field. Saving without changing anything adds nothing.
- **Workflow actions** — each action taken at each step, including steps of sub-workflows, and changes of step owner.
- **Button clicks** — when someone clicked one of the entity's custom-action buttons.
- **Automations** — each reaction that ran on the record, or that changed it from another record.
- **Creation and archiving** of the record.
- **Missed time limits** — when a workflow step went overdue.

Each item shows when it happened and who did it — including "X acting for Y" when someone acts during a delegation [Administration › Delegation].

Choose what a view shows

What people see of a record's history depends on the view they open it from:

1. Open the entity's **Views** tab and open the view.
2. In the view's entry-details settings, turn on **Entry history**. If it is off, the record has no history for people using this view.

3. Choose the detail:
    - **Changes** — show *what* changed. Without it, people see that the record was changed, but not the values.
    - **Changer** — show *who* made each change. Without it, only the time is shown.
    - **Reaction history** — include the automations that ran on the record, and **Reaction history details** to show their input, output and errors.
4. For workflow history, turn on **Workflow history** (and **Workflow path** for the step-by-step path). **Display step owner** decides whose steps show their owners: **All**, or **Owned by User** — only steps the person (or one of their groups) owned or acted on. **Default workflow view** decides which of the two opens first.
5. **Save** the view.

Only the fields the view shows appear in the history; fields hidden by the view — including fields hidden by a condition — are left out of it.

Changes to multilingual fields `Since 1.8`

When someone edits a field that has a value per language, the history shows one line per language that changed — `label: old → new` — in the space's language order. Each value is shown in its own language, so the same edit reads the same whatever language the reader uses. A language that was added shows only the new value; a language that was cleared shows as removed.

Good to know

- **Sensitive values are masked** in the history for anyone who may not see that field — permanently, including the values recorded before. See Sensitive fields.
- Creation and archiving appear only in views that allow archived records, or that open records as an administrator.
- History is written a moment after a save, so a change can take a second to appear.
- Deleting a record permanently deletes its history.

Timeline posts

Add a timeline to an entity

1. Open the entity's designer: **Administration › Applications › Modules**, open the module, open the entity, and edit it.
2. Select the **Timeline** tab, then select the empty area to add a timeline.
3. Under **Timeline Information**, set the timeline's **Title** and **Icon**. Both start from the entity's own.
4. Under **Post Configuration**, choose the **Post Template**:
    - **General Post** — a title, a description and attachments.
    - **Calender Post** — also a start and end date and time, a cover image and a location (a place, or a link for an online event).
5. Map each part of the post. For each one choose a **Type** and a **Value**:
    - **Field** — take it from a field of the record (for example the announcement's *Subject* as the post's title).
    - **Fixed** — the same value for every post.
    - **Hidden** — leave it out. Required parts cannot be hidden.
6. Under **Publish**, choose how posts go live — see Publish posts.

7. Under **Permissions**, choose who sees the posts: **Anonymous** (anyone, even without signing in), **Authenticated** (everyone signed in), or only selected users and groups.
8. **Save** the entity.

From then on, posts follow the records: a new record creates a post, editing the record updates its post, and archiving or deleting the record removes it.

Publish posts

A post starts as a **Draft** and becomes **Published**, then possibly **Unpublished**. Under **Publish**, choose who or what publishes it:

Way	Set it up	What happens
By hand	Leave **Steps to publish at** empty.	People with edit rights on the entity select **Publish** (or **Unpublish**) on the post and confirm.
At a workflow step	In **Steps to publish at**, choose one or more steps of the entity's workflow.	The post is published when its record reaches one of those steps. The entity needs a workflow with at least one step of its own.

Then, optionally, tie publishing to dates in the record: choose a date field in **Publish Date** and, if posts should come down again, in **Unpublish Date**. A post with a publish date goes live only when it has been published (by hand or by its workflow step) **and** that date has passed; it is unpublished when its unpublish date passes.

If you later remove the workflow, or the step a timeline publishes at, the designer tells you and switches publishing back to by hand.

Pin posts

To keep an important post at the top of the feed, select **Pin Post** on it; **Unpin Post** releases it. Pinning needs edit rights on the entity. The **TimeLine** widget shows pinned posts only if **Display Pinned Posts** is on.

Show the timeline on a page

1. Open the page in the page designer and add a **TimeLine** widget.
2. Choose its **Source**: **Full** (every timeline you can see), **Module** (the timelines of one module) or **Entity** (one entity's timeline — pick the **Module**, then the entity).
3. Choose the **Pagination Type** — **Auto Load**, **Manual Load** or **Pagination** — and the number of **Posts Per Page**.
4. Turn **Display Pinned Posts** on or off.
5. Publish the page. See Pages & widgets.

People see only the posts that are published and whose **Permissions** include them. They can react to a post with an emoji and comment on it, mentioning colleagues with **@**. These reactions are social reactions; they have nothing to do with automation reactions.

Post categories `Since 1.10`

Every tenant comes with the two post categories behind the templates — General and Calendar — set up automatically, including tenants created on a shared platform. You do not need to create them.

Good to know

- The **General Post** template is the one fully displayed on the web; the calendar template does not yet have its own display.
- In the mobile app, people can react to posts only with a *like*.
- A post is itself a record, so it has a history of its own.
- In preview mode, a view shows a short sample feed instead of real posts.
- If a post looks out of date right after a change to its record, refresh: posts are updated a moment after the record is saved.

Troubleshooting

Problem	What to check
The **History** tab is empty	**Entry history** is off in this view.
History shows that something changed, but not what	**Changes** is off in this view — or the field is not in the view, or it is sensitive.
Steps to publish at is disabled	The entity has no workflow with a step of its own (*"You must add at least one step to the workflow…"*).
Publish does not appear on a post	The timeline publishes at a workflow step, not by hand, or you lack edit rights on the entity.
A post is not shown to someone	Is it published, has its publish date passed, and do the timeline's **Permissions** include them?

Go deeper

For integrators: History and timeline API reference [Integrate & Extend] — the history record model and numeric types, how history is recorded, the v1 and paginated v2 history endpoints, view filtering and masking rules, trigger history, and the post feed, pin and publish endpoints

Share an entity's records across spaces

This guide is for **builders** of applications used in more than one space [Administration › Spaces & domains] — for example one space per branch or per agency. By the end you will know where an entity's records are kept, how to let a chosen group of spaces share records, and how to decide where new records are saved.

If your organization works in a single space, you do not need anything on this page: every record is visible in that space, subject to the usual permissions.

How stores work

Every record is kept in exactly one **store**, and the store decides which spaces can see the record. Each entity has three kinds:

Store	Visible in	Created
Public	Every space	Automatically, with the entity
Private	Only the space the record was created in (each space has its own)	Automatically, with the entity
Shared	Only the spaces you choose — at least two	By you, when needed

People in a space see records from every store that covers their space: the public store, their own space's private store, and any shared store that includes their space. Permissions still apply on top.

Where new records go. Each store has a **Weight**. When someone creates a record, it is saved in the store with the highest weight among those that cover their space. Out of the box the public store has weight 100 and the private store 50, so **new records are public** — visible in every space.

Take a *Contracts* entity used by three regional spaces. North and South work together and should see each other's contracts; West should not see them.

- Create a shared store *North–South* covering the North and South spaces, with weight 150. New contracts created in North or South now go into it.
- West still saves to the public store. To keep West's contracts to West, raise the private store's weight above the public one (for example to 120).

Create a shared store

1. Open the entity and select its **Stores** tab. It lists the public, private and shared stores of the entity.
2. Select **New Shared Store**.
3. Enter a **Name** and, optionally, a **Description**.
4. In **Spaces**, choose the spaces that share the records — at least two.
5. Enter a **Weight**. Use a number higher than 100 if new records created in those spaces should go into this store. Each store of an entity needs a different weight.
6. Save.

Change where new records are saved

1. On the entity's **Stores** tab, select the store you want to change — public, private or shared.
2. Change its **Weight**: the higher the number, the more it is preferred.
3. Save.

Only the weight of the public and private stores can be changed. For a shared store you can also change its name, description and spaces.

Good to know

- **Weights affect new records only.** Records already saved stay in their store.
- **Records follow their space.** The same record list shows different records in different spaces; that is expected.
- **Large lists.** The platform indexes stores for you so that long lists load quickly; there is nothing to set up.

Troubleshooting

Problem	What to check
The store weight is unique on the entity level	Another store of this entity already uses that weight. Pick a different number.
You must select at least 2 spaces.	A shared store needs two or more spaces.
Records created in a space are visible everywhere	They went to the public store. Give a shared or private store a higher weight than the public store.
A record is missing in one space	It is in a store that does not cover that space — for example another space's private store.

Go deeper

For integrators: Data stores: model, routing and API [Integrate & Extend] — store rules, write routing and read gating, the Stores API and its errors

Protect a field

This guide is for **builders** whose entities hold information that only a few people should see — salaries, national IDs, bank accounts, medical notes. By the end you will be able to mark a field **Sensitive** or **Encrypt** it, choose who may see its value, and decide what reactions, document templates and summaries may reveal when they use it.

How field protection works

Ordinary permissions decide who can open a record. Field protection goes one step further: it hides **one field's value** from everyone who has not been given access to that field — even people who can open and edit the record, and even system administrators.

Take an *Employees* entity with a *Salary* field. The HR team must see salaries; line managers can open their staff's records but must not see the salary.

	Sensitive `Since 1.8`	**Encrypt** `Since 1.10`
What it does	Shows `****` instead of the value to anyone without access	Stores the value encrypted, and shows it only on request to people with access
How people with access see it	They select **Unmask** and confirm who they are; values stay visible for the time they chose	They select the value's reveal icon and confirm who they are; one value is shown at a time
Can the field be filtered, searched, summed or grouped?	Filtered and summed only by people with access; never searched	No — the stored value is unreadable to the database
Use it for	Most confidential data people still need to work with	Data that must stay encrypted even in the database, such as identity numbers

A field can be **Sensitive** or **Encrypted**, never both. In both cases, access to the field is given to people and groups — by you when you design the field, or by a tenant administrator in **Administration › Access Management › Sensitive Permissions**.

Mark a field sensitive

1. Open the entity's designer (**three dots** on the entity card › **Edit Entity**) and select the field on the **Structure** tab.
2. In the field's settings, turn on **Sensitive**.
3. In **Select Users** and **Select groups**, choose who may see the value — for example the *HR* group.
4. Select **Save**.

From then on, everyone else sees `****` in that field — in lists, on the record, in its history and audit trail, and in reports and exports.

People who already have access through a wider grant — for example to every sensitive field of the entity or the module — are not listed in the field's own pickers. Select **Show Inherited** next to the **Sensitive** toggle to open **Inherited Permissions** and see who they are. Those grants are managed in Sensitive Permissions [Administration], where a tenant administrator can also give access for a limited period.

Encrypt a field `Since 1.10`

1. Select the field in the designer and turn on **Encrypt**. **Sensitive** turns off by itself — the two cannot be on together — and so does **Searchable**.
2. Choose who may see the value in **Select Users** and **Select groups** under the **Encrypt** toggle. These are the same grants as for a sensitive field.
3. Select **Save**.

If the entity already has records, turning **Encrypt** on or off is a structure change that marks those records obsolete until they are upgraded; upgrading encrypts (or decrypts) their stored values. See Change an entity safely.

Your operator must have set up the encryption keys for this to work — see Field encryption keys and providers.

What an encrypted field cannot do

Because the database cannot read the value, an encrypted field cannot be used:

- in searches, filters or query conditions — not even by people with access;
- in reaction conditions;
- in summary totals, averages and other aggregates;
- as the entity's main title;
- in reports;
- in notification and email text (it comes out empty).

Document templates show it masked. A view grouped by a field that later becomes encrypted (or sensitive) does not break: all its records fall into one group headed `****`.

Which fields can be protected

- **Reference** fields and **blocks** cannot be marked Sensitive or Encrypted. The individual fields inside a block can.
- **Switch** and **Category** fields can be Sensitive but not Encrypted.
- The entity's **main title** cannot be protected, because titles are shown to everyone who can see the record. The **sub title** can be — it is masked where it appears.
- A mapped field takes its protection from its source field: if the source is sensitive, so is the mapped field, with the same people allowed to see it.

Decide what automations may reveal `Since 1.10`

A reaction, a document template or a summary often runs for someone else — an email sent by a reaction, a file generated from a template, a dashboard tile. When the entity has a protected field, these forms show a **Sensitive Data Access Policy** setting that decides what *that element* may reveal:

Policy	Effect
Privileged	Always reveals the value. This is the default, and how these elements always behaved.
Prohibited	Never reveals it, whoever runs or opens the element.
User-Based	Reveals it only if the person who triggered or opened the element has access to the field.

Policy	Effect
Auto	Acts as **Privileged** when the platform runs it automatically, and as **User-Based** when a person triggers it. Not available on summaries.

Things to know:

- **User-Based** cannot be chosen for a reaction with an automatic trigger — a timer, a time-limit breach, or one run by the system — because no person's access can be checked.
- A summary that the policy does not allow is not shown at all; the tile says *You do not have permission to view this summary*.
- A reaction that must **use** a hidden value — to update a field, call another system — fails rather than writing `****`. Sending a notification, email or SMS, generating a file and announcing an event are allowed to pass the masked value on.
- **Reports** do not have this setting: each reader sees only the fields they have access to.

Good to know

- **No one sees protected values by default.** Holding access is not enough on its own: people reveal sensitive values with **Unmask**, confirming who they are, for a period they choose; encrypted values are revealed one at a time. What they see is described in Account security & sensitive data [Using OSOS › Account Security & Revealing Protected Values].
- **Filtering on a sensitive field** is allowed only for people with access to it. In **Administration**, designers may name a sensitive field in a query, filter or reaction condition even without access, so they can build them for others; the values stay hidden.
- **Editing without seeing.** People who can edit the record but cannot see the field can still type a new value into it; they never see the current one.
- **Every reveal is audited** — who revealed which fields of which records, never the value itself.
- **Access can expire.** A tenant administrator can give access for a limited period; it ends automatically.

Troubleshooting

Problem	What to check
Sensitive cannot be turned on	The field is a reference field or a block, or it is the entity's main title — choose another main title first.
Encrypt turns itself off after saving	The field is a reference, block, switch or category field, which cannot be encrypted.
Someone with access still sees `****`	They have not unmasked yet, or their unmask period has ended.
A query or reaction condition is refused	It uses an encrypted field, or a sensitive field the person does not have access to.
A summary tile shows *You do not have permission to view this summary*	Its **Sensitive Data Access Policy** blocks the viewer — check the policy and the viewer's access.
A reaction fails on a protected field	Its policy hides a value the reaction needs to act on. Use **Privileged**, or remove the field from the action.

Go deeper

For integrators: Sensitive and encrypted fields: model, API and enforcement [Integrate & Extend] — scopes, the grants API, masking signals, step-up sessions and proofs, Decrypt, write and query enforcement · For operators: Field encryption keys and providers

Preview Mode

This guide is for **builders** who want to see their work before other people do. By the end you will know how to preview a view with demo data on the web and on a phone, how to watch a page render on a phone while you edit it, and which questions a preview can and cannot answer.

> **Availability**
>
> Preview Mode is **Since 1.8** — both the view preview and the live mobile preview of pages.

How preview works

There are two previews, and they show different data:

You are designing…	The preview shows	Why
A **view** (in the view designer)	**Demo data** — realistic sample records generated for your entity	A view is often designed before anyone has entered a record. Demo data lets you judge columns, cards, faces and buttons straight away.
A **page** (in the page designer)	**Your real records**, in the page layout you are editing — unsaved changes included	A page is assembled from views and widgets that already exist, so it shows what is actually there.

In both cases **nothing is saved**: previewing never creates, changes or deletes a record, a comment, a post or a workflow step.

Preview a view

1. Open the entity's **Views** tab and open the view.
2. The **Preview** panel sits beside the view's settings. On a smaller screen, select **Preview** to open it in a dialog; **Close preview** hides it again and **Expand preview** enlarges it.
3. Choose **Web** or **Mobile** at the top of the panel.
4. Choose the screen to check — for example **Entries list**, **Table view**, **Kanban view**, **Entry details**, **Create entry**, **Edit entry** or **Dashboard**.
5. Change a setting and watch the preview follow. Open a sample record, switch between the view's faces, and check the tabs on the record's details.
6. When it looks right, **Save** the view.

If the panel says **Enable a view display to preview it here**, turn on at least one display (such as the entries list) in the view first.

Preview a page on a phone `Since 1.8`

1. Open the page in the page designer and go to its **Content** tab.
2. Select **Show preview**. A phone frame opens beside the canvas, running the real mobile app.
3. Edit the page — add a widget, change its width or order. The phone follows each change within a moment, without saving or reloading, and keeps its scroll position.

4. Select **Expand preview** for a larger frame, or **Hide preview** to close it.

Use it to check that widths, ordering and alignment still work on a narrow screen before you publish the page.

What the demo data looks like

Demo records are made to look real enough to judge a layout:

- Each field gets a value that fits its type — a number, a recent date, a sentence in each language, a point on a map, an email address or phone number when the field's name suggests one. File fields stay empty.
- **Your own settings are used wherever possible.** Category options, tags and kanban columns come from the entity's real configuration; the view's buttons and saved queries are the real ones. Only the records themselves are invented.
- **The data is the same every time.** A sample row and the details screen you open from it always agree, and the preview does not reshuffle after an unrelated edit.
- Repeatable blocks get their minimum number of rows, each with different values.
- A record's workflow status follows your entity's real first workflow, but no step has an owner, so nothing in the preview looks actionable.
- Relation tabs show the real relation names; the records under them are samples. Pickers for related records and users offer a short list of sample records.
- A dashboard with no widgets yet shows two demo widgets; a dashboard you have already built keeps its layout with sample numbers. Timelines show a short sample feed, and comments a short sample thread.

What a preview cannot tell you

- **It is not a permission check.** A preview runs with *your* access, and it cannot show you data you are not allowed to see. To find out what a restricted person will see, publish the view and sign in as (or ask) someone with that person's access. See Permissions [Administration › Permissions & access].
- **Demo records cannot be acted on.** Buttons, saved queries and records in a view preview are not connected to real records, so clicking them does nothing to your data.
- **The phone preview is one screen, not the whole app.** The app bar is drawn but does not navigate, there is no help button, and the page's redirection rules do not fire (they would take the preview away from the page). Tabs on a record's details — discussion, workflow, history, information — open inside the preview.
- **Reports are web-only.** They do not render in the mobile app, so check them on the web.
- **Half-finished settings are skipped, not shown broken.** A widget that is not fully configured is left out while the rest of the page renders; a sort rule with a field but no direction yet is ignored until you pick one.

Good to know

- Check light and dark themes and the phone layout: custom styling can look different in each.
- The phone preview shows the portal's current main colour, so it matches your branding.

Troubleshooting

Problem	What to check
Mobile preview host is not configured	Your platform has no mobile preview installed. Ask your operator; the web preview still works.
Enable a view display to preview it here	The view has no display turned on yet. Enable the entries list, entry details or another screen.
Preview for this screen is coming soon	That screen has no preview yet. Check it in the portal after saving.

Go deeper

For integrators: Preview mode internals [Integrate & Extend] — the preview request header and what it relaxes, how demo data is generated per field type, the builder-to-app bridge, and the version notes

The Admin Bar

> **Availability**
>
> **Since 1.10.**

This page is for builders and administrators who move between using an application and changing it. The admin bar is a thin strip above the portal's top bar. While you look at a page, list, report or collection, it gives you a one-click shortcut to that object's definition in **Administration**, so you don't have to find it in the admin menu.

How it works

When you open something in the portal that you're allowed to change, the bar shows a chip with the object's type (**Data View**, **Report**, **Collection** or **Page**), its name, and an **Edit definition** link. Select the link to open the definition in **Administration**.

An example. Mariam builds the *Assets* application. While testing the *Laptops* list as a user would, she notices a column in the wrong place. The admin bar above the page shows **Data View** · *Laptops* · **Edit definition**. She selects the link, moves the column in the view designer, and goes back to the list to check.

People who use the portal without building anything never see the bar.

When the bar appears

The bar shows only when all of these hold:

- you're signed in;
- your organization's license includes Administration;
- you have the access the object needs (see the table below); and
- the page you're on is one of the objects the bar supports.

If any of these doesn't hold, there is no bar and no empty space where it would be. The bar never appears inside the designer's own preview, or when a page is embedded in another site.

You're looking at	The link opens	You need
A list (data view)	The view's definition under its entity	**Developer** access on the view's entity
A report	The report's definition under its entity	**Developer** access on the report's entity. Being able to read the report isn't enough.
A personal or shared collection	The collection's definition in **Administration** (the **Lists** area)	**View** access on the collection's entity
A page	The page's definition under its module, or under **Pages** if it has no module	**Developer** access on the module

You see one chip at a time. A collection opens inside its list, but the bar shows only the collection.

Hide or show the bar

- **Hide it:** select **Hide the admin bar** (the close button on the bar). The bar stays hidden in this browser, on every page, until you show it again.
- **Show it again:** select your picture at the top of the portal, then **Show admin bar**. The same item reads **Hide the admin bar** while the bar is showing.

The menu item only appears on a page where the bar has something to link to, and only for people with at least **View** access to Administration.

Technical details

One service computes what the bar shows and one component renders it, both in the portal's main layout. The bar reads what the current page declares itself to be and cross-checks it against the object the session has already loaded, so it never shows a stale or mismatched link. The hidden/shown choice is stored in the browser.

The routes it links to are `/admin/{module}/{entity}/views/{id}` (from `/v/{id}`), `/admin/{module}/{entity}/reports/{id}` (from `/reports/{id}`), `/admin/collections/{id}` (from `/v/{id}/c/{collectionId}`) and `/admin/{module}/generic-views/{id}` or `/admin/pages/generic-views/{id}` (from a page).

The mechanism declares 14 kinds of administrable object, each with a link rule. Only the four above are wired to a portal page today. The other ten (data entity, data query, data relation, data module, data reaction, system button, statistic query, function, counter widget and announcement) are groundwork: no page declares itself as one of them, so the bar never shows a chip for them.

Lists in Administration

This page is for builders who set up applications in **Administration**. It shows you how to create a list of records or of values, fill it, arrange its values as a hierarchy, and remove it when it is no longer used. End users keep their own lists of records from a view; that is covered in Lists: Keep Records Together [Using OSOS].

How it works

A **list** groups data under one name so that other parts of an application can use it: the choices of a field, the columns of a Kanban board, a category tree. A list holds one kind of content, chosen when you create it:

Data Type	The list holds
Entity	Records of one entity. A list never mixes records from different entities, and a record appears in it at most once.
Text	Text values you enter.
Number	Numbers you enter.
Date	Dates picked from a calendar.
Time	Times, which adjust to the time zone.
DateTime	Combined date and time values.

A list is either **flat**, a plain set of values, or a **hierarchy** (a tree with levels), which suits reference lists such as a category tree. Lists of values can be hierarchies; entity lists are flat.

Lists you create here belong to the entity as a whole, so everyone who can see the entity can see them. They are different from the personal lists end users create inside a view, which belong to that user and view.

An example. A builder of an asset application creates a **Text** list called *Asset condition* with the values *New*, *Good*, *Worn* and *Out of service*, and uses it for the *Condition* field. Later the asset view gets a Kanban board split by that field, with one column per value.

Create a list

You need developer access to create lists.

1. Open **Administration › Lists**, or open an entity and go to its **Lists** tab.
2. Select **New List**.
3. Fill in the dialog:
    - **Name** (required) and **Description**, in each of your organization's languages;
    - **Icon**;
    - **Data Type**: **Entity**, or one of the value types;
    - **Choose data entity**, for an entity list;
    - **Viewing Level**: **Organization**, **Private** or **Public**.
4. Save. The list opens so you can add its content.

The name must be unique among the lists of the same entity.

Add and remove content

1. Open the list in **Administration › Lists**.
2. Select **Edit Entries**.
3. For an **Entity** list, add records from the entity or remove the ones you no longer want. For a value list, select **Add**, enter a value of the list's type, or remove a value.
4. Save.

To find a list, use the search on the **Lists** page. Each list shows as a card with its name, icon, number of entries, data type and viewing level.

End users can also add records to a list from a view, with **Add to List**, when the view allows user lists.

Build a hierarchy

In a value list, give each value a level and, below the top level, a parent value. When you save, the list is checked so that it always displays as a proper tree. The save is refused if:

- a value has no value text or no level;
- two values under the same parent clash in order;
- a value points to a parent that doesn't exist;
- a value is its own parent.

Values the platform adds itself

Some values are placed in a list by the platform rather than by you. They are protected: an edit that would change or drop one is refused, so system-maintained values can't be lost by accident. Lists that the platform maintains itself can't be deleted, and their structure can't be changed.

Use a list

- **As a field's choices.** A category field draws its options from a list. People filling in a record pick from it.
- **To split a Kanban board.** A field that points at a list item can split a board into one column per item. A card can only be dropped on an item that is still in the list. An item deleted from the list keeps its column, marked as deleted, so its cards stay reachable. See Data Views & Faces: Kanban.
- **In queries.** A query condition can take its values from a list.

Edit, export and delete a list

Open the list. Its header offers **Edit** (name, icon, description), **Export**, **Edit List** and **Delete**.

A list that is used by an entity can't be deleted: you see *"Sorry, you can't delete this list, since it is associated with data entities"*. Remove the list from those entities first. Otherwise, confirm the prompt to delete the list. Deleting a list removes the grouping only, never the records in it.

Good to know

- **Two builders editing at once.** If someone else saved the list after you opened it, your save is refused rather than overwriting their change. Reload the list and try again.

- **Installed lists.** Lists that came from an installed package can't be edited or deleted here.

Technical details

- A list is stored as a *collection* of the entity. Its structure type is **Flat** or **Tree**; two further types, **Reference** and **Multi**, are defined but reserved and not validated today.
- Each list gets a short machine-friendly name generated from its name, unless you supply one; a supplied one must be unique (since 1.6).
- Every list carries an update token that changes on each save; a save that started from an older token is refused as a conflict.

> **Go deeper**
>
> For integrators: Records and lists: behaviour reference [Integrate & Extend › Records and Lists: Behaviour Reference].

Quick videos

Short videos of the main building flows are being recorded. Until a video is published, the written page below walks you through the same flow, step by step, with the labels you see on screen. Each video will be added to its page when it is ready.

Flow	Follow these steps now
Create a module and an entity	Creating a module · Creating an entity
Add fields and lay out the form	Create the Tasks entity
Build a table and a kanban view	Build the views
Design a page with a chart	Build the home page · Add a dashboard
Set up an approval workflow	Publish a service with an approval workflow
Send an email automatically	Notify the assignee

For end users and administrators:

- Submit and track a service request [Using OSOS › Video script: Submit & Track a Request]
- Manage permissions, including sensitive fields [Administration › Video: Manage permissions (incl. sensitive fields)]

Publish a service with an approval workflow

This how-to is for **builders**. You will publish an **Asset Request** service: employees open it from the portal's **Services Catalog**, ask for a laptop or a monitor, their manager approves, and the IT team fulfils the request. The same recipe works for any request that needs approval — leave, purchases, access requests.

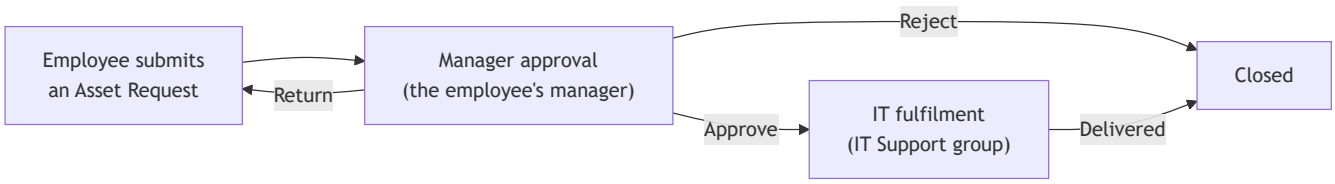
Time: about 30 minutes.

You need:

- **Developer** access to **Administration** and to the module the service will belong to (here, an *Asset Management* module — see Creating a module).
- A group for the IT team (for example *IT Support*) with its members. Your tenant administrator manages groups in **Administration › Organization › Groups**.
- Managers recorded for the employees who will submit requests, in the organization structure. "The requester's manager" is looked up from there.
- At least one service category. Categories come from the service-category list; ask your tenant administrator if none exist.
- Two test accounts: an employee who has a manager, and a member of *IT Support*.

How it works

A **service** is an entity whose records are **requests**. You design it like any other entity — fields, form, workflow — and then **activate** it. Once active, the people in its **audience** see it in **Services › Services Catalog**, open it, and select **New Request**. The request then follows the service's workflow: each step's owner finds it in their **Inbox**, and the requester follows it in their **Outbox**.



1. Create the service

1. Open **Administration › Services › Services Catalog**. The list of existing services opens.
2. Select **Add Service**. The **Add New Data Entity** dialog opens.
3. Fill in:
    - **Name** — *Asset Request*, in Arabic and English. The **Normalized name** is filled in for you; it cannot be changed later.
    - **Select Icon** — pick an icon for the catalog.
    - **Choose Module** — *Asset Management*.
    - Under **Field Settings**, set **Single Name** (*Asset request*) and **Plural Name** (*Asset requests*). The portal uses these names on buttons and lists.
    - Leave **Administration Access** as it is unless other builders need access.
4. Select **Create**. The service is created and its designer opens.

2. Design the request form

1. In the designer, open the **Structure** tab.
2. Drag these field types from the palette and set their options:

Field	Type	Notes
Asset type	Category	Options: Laptop, Monitor, Phone, Other.
Reason	Text	Turn on multiple lines.
Needed by	Date	
Serial number	Text	Filled in by IT at the last step.

3. Select **Save**.

Field-by-field options are covered in Field types.

3. Add the two-step approval workflow

1. Open the **Workflows** tab and select **Add A Workflow**. The workflow starts with an *Initial Step* (owned by whoever submits the request) and a *Complete Step*.
2. Select the *Initial Step* and rename its action to *Submit* in **Action Settings**, so requesters see a clear button.

Step 1: the requester's manager

1. Select **Create Step** after the *Initial Step*.
2. In **Step Information**, set **Name** to *Manager approval*.
3. Keep **Step Type Normal**. Set **Main Action** to **Next Step**, and tick **Return** and **Reject**.
4. Under **Step Owner**:
    1. Choose **User**.
    2. In **User Type**, choose **Previous Step Owner**, and in **Step Name** choose *Initial Step* — the person who submitted the request.
    3. Turn on **Relative**, set **Relation** to **Manager** and **Relation Grade** to **1**.
5. **Fallback Assignment 1** appears: this is who gets the request if the requester has no manager recorded. Choose **User**, **Fixed**, and pick a person — for example the IT manager.
6. Select **Done**.
7. Open the new *Main Action* and rename it *Approve*. Open *Return*, rename it *Return for changes*, and set **Action comment status** to **Required** so the manager explains what to fix.

Step 2: the IT team

1. Select **Create Step** after *Manager approval*.
2. Set **Name** to *IT fulfilment*, keep **Step Type Normal**, and set **Main Action** to **Next Step**. Leave **Return** and **Reject** unticked.
3. Under **Step Owner**:
    1. Choose **Group**, then **Group Type Fixed**, and pick *IT Support* in **Group Name**.
    2. In **Select User Type**, choose **Single User**.

3. In **Choose Selection Type**, choose **Take Ownership**, so any IT member can pick the request up. (Choose **First one** instead if one person should always get it.)
4. In **Fallback Assignment 1**, choose **User**, **Fixed**, and the IT manager.
5. Select **Done**, then rename the *Main Action* to *Delivered*.

Field rules for each step

Open each step's **Requirements** and set the **View Option** of each field:

Field	Initial Step	Manager approval	IT fulfilment
Asset type, *Reason*, *Needed by*	**Mandatory**	**Locked**	**Locked**
Serial number	**Hidden**	**Hidden**	**Mandatory**

Select **Save**.

More owner options — a group vote, a person named in a field, time limits and escalation — are in Workflows & approvals.

4. Choose who can submit it

1. Open the **Details** tab. Because this is a service, it shows the service settings.
2. In **Category**, choose a category (for example *IT services*). The catalog groups services by category.
3. Under **Audience**:
   - Turn on **Authenticated** to offer the service to signed-in users.
   - To offer it only to some people, pick them in **Select Users** or pick teams in **Select groups**. Leave both empty to offer it to everyone who signs in.
   - **Anonymous** offers the service to people who are not signed in — not what you want for an internal asset request. It cannot be combined with picking specific users or groups. If you do use it, consider **Human Verification**.
4. Optionally, under **Watchers**, add people or groups who should follow every request without acting on it (for example the IT manager).
5. Select **Save**.

5. Activate the service

1. Open the designer's actions menu and select **Save and Activate**.
2. Confirm with **Save and Activate**.

Save and Activate is available only when the service has a workflow, a **Category**, and an **Audience** (**Authenticated** or **Anonymous** turned on). To take the service out of the catalog later, select **Deactivate**.

6. Test it as an end user

1. Sign in as the test employee.
2. Open **Services › Services Catalog** and select *Asset Request*. The service's page opens.
3. Select **New Request**, fill in the form, and submit it. If the request opens on its first step with a *Submit* button, select it to send it on.

4. Sign in as the employee's manager. Open **Services › Inbox**, open the request, and select *Approve*. Try *Return for changes* on a second request and check that the employee gets it back.
5. Sign in as a member of *IT Support*. Open **Services › Inbox**, open the request, select **Take Ownership**, fill in *Serial number*, and select *Delivered*.
6. Sign in as the employee again and open **Services › Outbox**: the request is complete.

End users' side of this — submitting, tracking, acting on a step — is described in Submit and track a request [Using OSOS › Submit & Track a Request] and Executing workflow actions [Using OSOS › Acting on Workflow Steps].

Troubleshooting

Problem	What to check
Save and Activate is greyed out	The service needs a workflow, a **Category** and an **Audience** with **Authenticated** or **Anonymous** turned on. Also fix any validation error in the designer first.
The service is not in the employee's **Services Catalog**	The service is not active, or the employee is not in the audience (check **Select Users** / **Select groups**). An anonymous-only service is hidden from signed-in users.
The manager never gets the request	The employee has no manager recorded, so the request went to **Fallback Assignment 1**. Ask your tenant administrator to record the manager.
No IT member sees the request	With **Take Ownership** the request waits unassigned until a member claims it; check that *IT Support* has active members.
The category list is empty	No service categories exist yet; ask your tenant administrator to add one.

Next steps

- Email the requester when the request is approved: add a reaction on **On Workflow Action Executed** — see Reactions & automation.
- Add a time limit to *Manager approval* and escalate when it is overdue — see Time limits and escalation.
- Show open requests on a page — see Dashboards & summaries and Pages & widgets.

ⓘ Go deeper

For integrators: Workflow model and API reference [Integrate & Extend] · OSOS as a Backend [Integrate & Extend]

Change an entity safely

This how-to is for **builders** who need to change an entity that people are already using — add a required field, change a field's type, rework the approval workflow. You will learn which changes are safe, what happens to existing records when a change is not, and how to bring those records up to date.

You need: Developer access to the entity's module, and permission to edit its records if you will upgrade them yourself.

How it works

Every record remembers the version of the entity's structure it was saved with. Most changes leave that version alone. A **breaking change** — one that existing records may not satisfy — creates a new version of the structure. Records saved before it are then marked **Obsolete** until someone upgrades them.

Take an *Asset Request* entity with 200 requests. Adding an optional *Cost centre* field is safe: old requests simply have it empty. Making *Cost centre* **required** is breaking: the 200 old requests do not have one, so they become obsolete until you give them a value.

Safe — no new version	Breaking — new version, existing records become obsolete
Adding an optional field	Adding a **required** field, or making an existing field required
Renaming a field or the entity	Changing a field's **type**
Changing labels, help text, layout, icons	Turning a field's **repeatable** or **multiple** option on or off
Deleting a field	Turning **Encrypt** on or off for a field `Since 1.10`
Changing which entity a reference field points at	Turning on the restriction against circular references on a reference field
	Adding, removing, reordering or changing **workflow** steps, or deleting a workflow
	Re-adding a deleted field with a different type

An entity with no records yet never creates a new version, whatever you change.

Change the structure

1. Open the entity's designer (**three dots** on the entity card › **Edit Entity**) and make your changes.
2. Select **Save**.
3. If the change is breaking and the entity has records, the **Confirm Structure Changes** dialog lists each affected field and the reason — for example *Changed to required* or *Field type changed*. Read it carefully.
4. Select **Save** to apply the change, or **Cancel** to go back and rethink it.

> **Tip**
>
> Make breaking changes in one go. Every breaking save creates a new version, and each version is another set of records to upgrade.

Upgrade obsolete records

After a breaking change, existing records show as **Obsolete**. They can still be opened, but they may not display correctly, and they cannot go on through their workflow until they are upgraded.

1. Open the entity's **Entries** tab and filter the list to **Obsolete entries**.
2. Select the records to upgrade (or select all of them).
3. From the selection menu, select **Upgrade (Entries N)**.
4. The dialog lists **Required changes** — fields every record must get a value for, such as the new required field — and **Additional changes** you may also make. For each field, choose where the value comes from:

Option	Use it when
Fixed	Every record should get the same value, such as a default cost centre.
Expression	The value must be calculated, for example from other fields of the record.
Auto Repair	The record's current value should be kept and converted to the field's new form — for example a number stored as text becomes a number, and a category that no longer allows several values keeps the first one picked.
Default	The field's default value is right. Offered only when the field has a default.
Not Set	The field may stay empty. Not offered for required fields.

5. If the change touched the **workflow**, tick **Reset workflow if needed**. Records whose workflow changed cannot be upgraded without it; they restart their workflow from the first step.
6. Select **Upgrade**.

To upgrade a single record, open it and select **Upgrade Entry**.

To change values on many **current** records at once — not obsolete ones — select them and use **Bulk Edit (Entries N)**.

Good to know

- **Archived records** that were obsolete stay obsolete when you restore them. Upgrade them after restoring.
- **Encrypt changes** are upgraded the same way. When you turn Encrypt **off**, **Auto Repair** is not offered for that field, because the stored value is still encrypted: use **Fixed** (after revealing the value) or **Expression** instead. See Protect a field.
- **Several versions behind.** A record can be more than one version behind if several breaking changes were saved. The upgrade takes it straight to the latest version.

Troubleshooting

Problem	What to check
Upgrade is not offered	Only obsolete records can be upgraded. Filter the list to **Obsolete entries** first.
Some records fail with *Workflow reset required*	The workflow changed. Run the upgrade again with **Reset workflow if needed** ticked.

Problem	What to check
An obsolete record cannot move on in its workflow	Upgrade it first.
Auto Repair is missing for a field	Its Encrypt option is being turned off. Pick another option.

Go deeper

For integrators: Versioning and breaking changes [Integrate & Extend › Entity model and API reference] — how versions are counted, the full list of breaking reasons, and the version-diff endpoints

Build recipes

These are short procedures for **builders** who already know what they want to do. Each recipe covers one task in **Administration**, with the labels you see on screen. If you are new to building, start with My First Application, which uses these same steps in a worked example.

Modules and entities

I want to…	Recipe
Create a container for a new application	Creating a module
Rename, re-share, sort or delete a module	Editing a module
Create a new kind of record	Creating an entity
Change fields, handle breaking changes, add an index, export, move, archive or delete an entity	Editing an entity

Every other building task

Each of these is covered step by step in its guide:

I want to…	Where
Save a filter, such as *my open tasks*	Queries
Show records in a list, board or calendar	Data views
Show counts and charts	Summaries, dashboards & counters
Notify people or update records automatically	Reactions & automation
Route a record for approval	Workflows & approvals
Publish a request service with an approval	Publish a service with an approval workflow
Email a report on a schedule	Reports
Build a home page	Pages & widgets
Add a button that runs an action	Buttons
Produce a document from a template	Templates
Link one entity to another	Relations
Move an application to another environment	Packaging [Integrate & Extend › Packaging & distribution]

Creating a module

This recipe is for **builders**. A **module** is the container for one application: it holds the app's entities, pages and dashboard, and it has its own entry in the sidebar. Create one module per application.

You need: access to **Administration** with rights to create modules.

Steps

1. Open **Administration › Applications › Modules**.
2. Select **Add Module**. The **New Module** dialog opens.
3. Fill in:
    - **Name** — required, in Arabic and English.
    - **Normalized name** — filled in from the English name. It must be unique and is fixed once the module is created, so check it now.
    - **Sharing** — choose the **Sharing level**: **Public** (every space sees the module), or **Private** or **Shared** (only the spaces you pick see it).
    - **Icon** — the icon shown in the sidebar.
    - **Description** — optional.
    - **Administration Access** — optionally, pick the users (**Select Users**) or groups (**Select groups**) who may work on the module in Administration.
4. Select **Done**. A success message appears and the module is listed.

A new module comes with a default **landing page** and **dashboard**, so it appears in the sidebar straight away.

Next step

Create the entities that hold your data, or move existing entities into the new module.

For a guided example, see step 3 of My First Application. More about modules — sharing, ordering, the sidebar — is in Modules.

Editing a module

This recipe is for **builders** who need to change an existing module, find it in a long list, or remove it.

Edit a module

1. Open **Administration › Applications › Modules**.
2. Select the **three dots** next to the module, then **Edit**. The **Edit Module** dialog opens.
3. Change the **Name**, **Icon**, **Description**, **Sharing** or access as needed. The **Normalized name** cannot be changed: queries, reactions and packages rely on it.
4. Select **Done** and check for the success message.

Open a module

Select the module's name in the list. Its page has four tabs: **Dashboard**, **Entities**, **Pages** and **Permissions**.

Sort the Modules and Entities lists `Since 1.10`

1. On the **Modules** list, the system **Data Entities** list, or a module's **Entities** tab, open **Sort**.
2. Choose **Newest created**, **Last modified** or **Name A–Z**, and toggle the direction. Use **Add Sorting** to add a second level.
3. Select **Reset** to go back to the default, newest created first.

Each list remembers its own sort until you sign out.

Delete a module `Since 1.10`

Delete removes the module **and everything in it** — its entities and their records, pages, dashboard and functions — and cannot be undone. Only a **system administrator** sees **Delete**. It is not offered for the platform's own core module, for modules the platform creates itself, or for modules installed from a package.

1. In the **Modules** list, select the **three dots** next to the module, then **Delete** (or use **Delete** in the open module's header menu).
2. The **Delete module** dialog lists everything **outside** the module that still points at it or at something in it — grouped by the module, an entity, a page, the dashboard, a function or a counter widget. Select **Open** on each one and remove the link. **Delete** stays unavailable until none remain.
3. Tick **I understand that this module, all its entities and their entries will be permanently deleted.** and select **Delete**.
4. The delete runs in the background. The module shows **Deleting…** with the number of entities left, and disappears when it is done.
5. If it shows **Delete failed**, open **Delete** again and choose **Delete again**; the delete carries on where it stopped.

While a module is being deleted, nobody can edit it, add entities or pages to it, or move entities into it.

More about deleting a module, and what blocks a delete, is in Modules.

Creating an entity

This recipe is for **builders**. An **entity** is one kind of record — a task, a request, an asset. It defines the fields each record has, and it belongs to a module.

You need: Developer access to the module.

Create the entity

1. Start the dialog in one of two ways:
   - Open the module (**Administration › Applications › Modules** › your module) and, on its **Entities** tab, select **Add Data Entity**; or
   - Open **Administration › Applications › Data Entities** and select **Add Data Entity**.
2. The **Add New Data Entity** dialog opens. Fill in:
   - **Name** — required, in Arabic and English, and unique within the module.
   - **Normalized name** — filled in from the English name. Queries, reactions and packages refer to the entity by this name, so check it now and treat it as permanent. You can change the display **Name** at any time.
   - **Select Icon** — the entity's icon.
   - **Choose Module** — already set when you started from a module; otherwise pick the module.
   - **Data Entity Type** — keep **Normal Entity** for records you define yourself. The other choices create virtual entities — see below.
   - Under **Field Settings**, set the **Single Name** and **Plural Name** — for example *Task* and *Tasks*. The portal uses them on buttons and lists, such as **Add Task**.
   - **Administration Access** — optionally, the users or groups who may work on the entity in Administration.
3. Select **Create**. The entity opens.

Add the fields

1. Open the entity's designer: select the menu next to the **Add** button at the top of the entity (for example **Add Task**), then **Edit Entity**.
2. On the **Structure** tab, drag field types from the **Fields** palette onto the empty area, name each field, and set its options.
3. Select **Save**, then confirm with **Save Changes**.

The entity is ready to use as soon as it is saved — there is no separate publish step. (A **service**, which people request from the portal's catalog, does need activating: see Publish a service with an approval workflow.)

Field types and their options are described in Field types. To put the entity in another module later, move it.

Entity types

Most entities you create are **Normal Entity**. The **Data Entity Type** list also offers **virtual entities** — **Virtual Proxy Entity**, **Joiner Virtual Entity** and **Aggregate Entity** — whose records are worked out from other entities and cannot be edited directly; see Virtual entities. You will also see entities the platform provides itself, such as users and groups, which you do not create.

For a guided example, see step 4 of My First Application.

Editing an entity

This recipe is for **builders** who need to change an entity that already exists: its details and fields, its indexes, where it lives, or whether it exists at all.

You need: Developer access to the entity. Moving and permanently deleting entities needs a **system administrator**.

Open the designer

Open the entity's designer in one of two ways:

- On the module's **Entities** tab, select the **three dots** on the entity card, then **Edit Entity**; or
- Open the entity, select the menu next to its **Add** button (for example **Add Task**), then **Edit Entity**.

Change the fields

1. In the designer, select the **Structure** tab.
2. Drag new field types from the **Fields** palette, or open an existing field's settings (the gear icon on its row) to change it.
3. To change the form layout — for example two columns — open **Form Settings** and choose a layout in **Choose Display Layout**.
4. Select **Save**, then **Save Changes**.

If the change is breaking, a dialog asks you to confirm it first — see Breaking changes below.

The designer's tabs

Tab	What you change there
Details	Name, icon, description, singular and plural names.
Structure	Fields and form layout.
Workflows	The steps a record goes through — see Workflows & approvals.
Timeline	How the entity's records contribute posts to the Timeline — see Timeline.
Realtime	Whether several people can edit the same record together, live.
Indexes `Since 1.10`	Extra indexes that speed up large lists — see below.

Reactions, queries, views, the dashboard and reports are not in the designer; they are tabs on the entity's own page.

If the entity fails to load, the designer offers to try again instead of showing an empty page.

Breaking changes

Every record remembers the version of the entity it was saved with. A **breaking** change — one that existing records may not satisfy — creates a new version, and records saved before it become **Obsolete** until they are upgraded.

Safe	Breaking
Adding an optional field	Adding a **required** field, or making a field required

Safe	Breaking
Renaming fields, changing labels, help text and layout	Changing a field's **type**
Deleting a field	Turning a field's **repeatable** or **multiple** option on or off
Changing which entity a reference field points at	Turning on the restriction against circular references on a reference field
	Adding, removing, reordering or changing **workflow** steps, or deleting a workflow
	Re-adding a deleted field with a different type

An entity with no records yet never creates a new version.

1. When you save a breaking change on an entity with records, the **Confirm Structure Changes** dialog lists each affected field and the reason. Select **Save** to go ahead, or **Cancel**.
2. Afterwards, on the entity's records tab, filter to **Obsolete entries**, select the records, and choose **Upgrade (Entries N)**. For each required change, choose where the value comes from — **Fixed**, **Expression**, **Auto Repair** or **Default** — and select **Upgrade**.
3. To upgrade a single record, open it and select **Upgrade Entry**.

A worked example — adding a required *Priority* and upgrading existing tasks — is in Change your app safely.

Add a custom index `Since 1.10`

Use the **Indexes** tab when people filter or sort a large entity by fields that the automatic indexes do not cover, and the list is slow. The tab is not offered for the core module's entities, for request entities or for virtual entities, and it is read-only on an entity installed from a package.

1. In the designer, select **Indexes**. It lists **System indexes (managed automatically)**, which you cannot change, and your **Custom indexes**, with how many of the 10 allowed you have used.
2. Select **Add index**.
3. Enter a **Name**, then **Add field** for each field to index (1 to 8 different fields), choosing **Ascending** or **Descending** for each. Use **Move field up** and **Move field down** (or drag) to set the order.
4. Optionally tick **Unique** so no two records can share the same values. A unique index must also leave out records where the field is empty, using a **Partial filter expression** — see the technical details below. Until it has one, **Save** stays unavailable.
5. Select **Save** at the top of the designer. Until then the index shows **Not saved yet**.
6. The index shows **Building...**, then **Ready**. If it shows **Failed** — usually because existing records break a unique index — fix the data and select **Retry**, or edit or delete the index.

Deleting an index removes it when you save the entity.

Technical details: partial filter expressions

A **Partial filter expression (optional, MongoDB JSON)** limits the index to some records. Only `$eq`, `$exists: true`, `$gt`, `$gte`, `$lt`, `$lte` and `$type` are accepted. For a unique index on a field with the normalized name *code*, leave out empty values with:

```
{"value.code": {"$type": "string"}}
```

The rules behind each check are in Entities.

Export an entity's records `Since 1.10`

1. Open the entity, open its header menu, and select **Export Entries**.
2. In the **Export entries** dialog, choose **Excel (XLSX)** or **CSV**.
3. The file downloads with every field as a column (each block split into its fields), covering the same records as the entity's list in Administration. If there are more than the export limit — 10,000 records by default — the file is prepared in the background instead.

This quick export, and the quick export from a gallery view, are the only exports that offer CSV; template exports and reports export to XLSX or PDF. See Quick export of an entity's entries.

Move an entity to another module `Since 1.10`

Only a **system administrator** can move entities. Entities of the core module, entities the platform creates itself, archived entities and entities installed from a package cannot be moved.

One entity

1. Select the **three dots** on the entity card and **Move to module**, or open the entity and select **Move to module** in its header menu.
2. In **Target module**, choose the module (or create one from the list).
3. Read the warning: summaries and widgets in the old module that use this entity are deleted.
4. Select **Move** and check for the success message.

Several entities at once

1. Open the module and, in its header menu, select **Move entities**.
2. Select the entities to move — your selection is kept while you page through the list. You can move up to **50** at a time.
3. Choose the target module and review the list. Each entity keeps the permissions given directly to it and also gets the target module's permissions.
4. Select **Move N entities**. Entities that cannot be moved are listed with the reason; the rest are moved.

What a move repairs — views, pages, dashboards, protected-field access, sidebar links — is described in Admin move and bulk move.

Archive an entity

1. Select the **three dots** on the entity card, then **Archive**.
2. Confirm. The entity moves to the archive.

Archiving keeps the entity and its records. To remove them for good, delete the entity permanently.

Delete an entity permanently `Since 1.10`

Permanent delete removes the entity and everything it owns — records, views, pages, dashboard, reports, buttons, functions and permissions — and **cannot be undone**. Only a **system administrator** sees **Delete**, and only on ordinary, request and virtual entities that the platform did not create itself, that are not in the core module, and that were not installed from a package.

1. Select the **three dots** on the entity card, or open the entity's header menu, and select **Delete**.
2. The **Delete entity** dialog lists everything elsewhere that still points at this entity — reference fields, mapped fields, virtual entities, reactions, views, pages, dashboards, queries, summaries, reports, buttons, lists and relations — grouped by kind. Select **Open** on each one, remove the link, then refresh the dialog. **Delete** stays unavailable until nothing is left.
3. Tick **I understand that this entity and all its entries will be permanently deleted.**
4. Select **Delete** and check for the success message.

If the delete is interrupted, the entity stays archived and marked for deletion; choosing **Delete** again finishes the job. What blocks a delete is described in Permanent delete.

Tutorials

Each tutorial builds a complete system from a business need to a tested app. They are for **builders** who have finished My First Application and are comfortable creating entities, queries and views. You need the same **Developer** access, plus a few test accounts and groups that each tutorial lists up front.

Tutorial	You will learn	Level	Time
Customer support system	A five-step ticket workflow with a group, a person named on the ticket and a manager approval; seven notification reactions tied to workflow steps; a status chart; a monthly report; testing by role	Intermediate	About 3 hours
Recruitment system	Several related entities, request services that applicants submit from the portal, lists, pages and reports	Intermediate	About 4 hours

> **Note**
>
> Parts of the recruitment tutorial are still in Arabic while the English edition is being completed.

For a shorter, focused example of a request with a two-step approval, see Publish a service with an approval workflow.

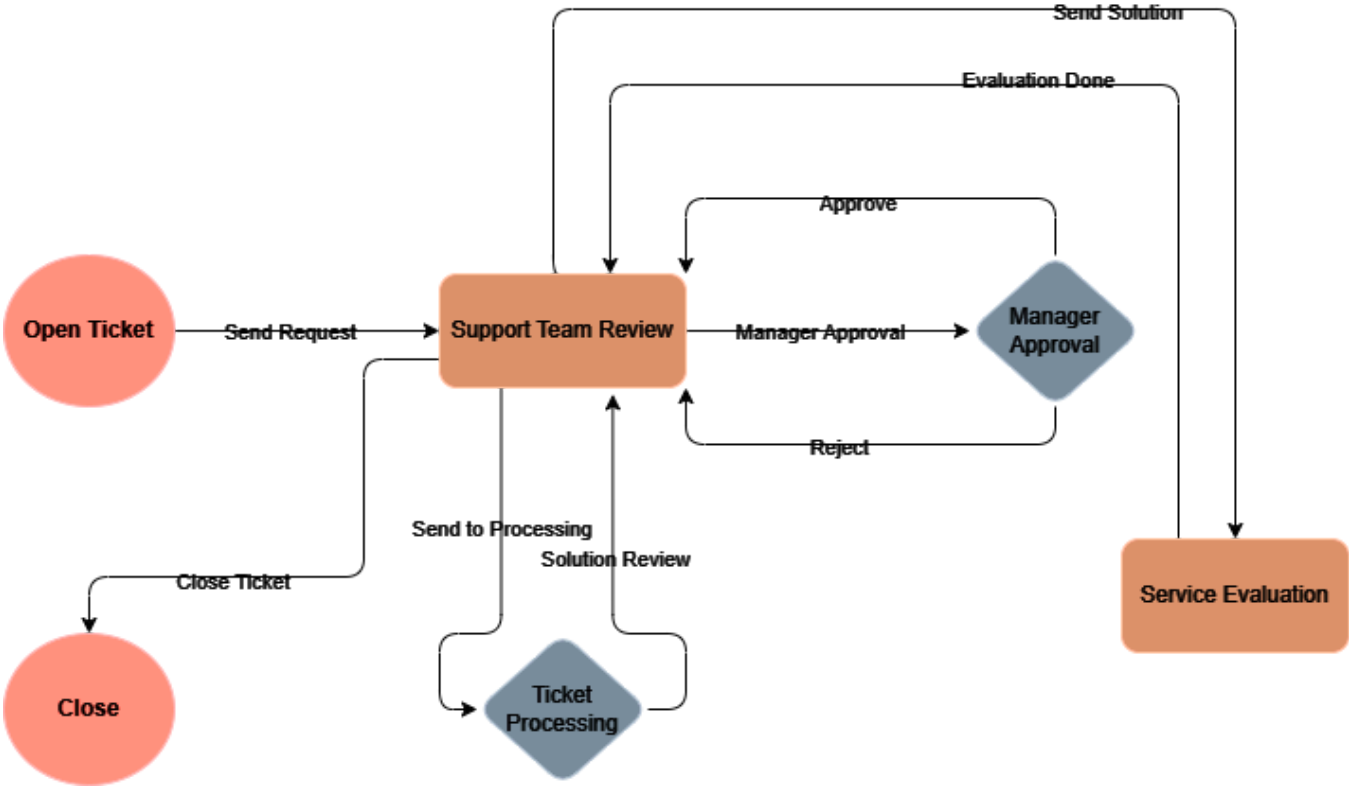
Customer support system

In this tutorial you build a complete **support-ticket system**. It is for **builders** who have finished My First Application. Where My First Application had no process, this system routes every ticket through a five-step **workflow**, with notifications at each hand-over.

Time: about 3 hours. **You need: Developer** access, and the test accounts and group listed in Plan the system.

What you will build

A customer opens a ticket. The **support desk** — a team — picks it up and either closes it, answers it directly, or assigns it to a **support representative**. The representative works out a solution and sends it back to the desk for review. If needed, a **manager** approves the answer. The desk then sends the solution to the customer, who rates the service, and the desk closes the ticket. Everyone is notified when a ticket reaches them, and managers get a status chart and a monthly report.



The people

Role	In the system	What they do
Customer	Any signed-in user	Opens tickets, receives the solution, rates the service
Support desk	The *Technical Support Office* group	Routes tickets, reviews solutions, sends them, closes tickets
Support representative	The person in *Assigned to*	Works out the solution
Manager	A fixed person	Approves or rejects a solution when the desk asks

Steps

Follow the build steps in order, starting with Understand the business need.

Understand the business need

Before building, be clear about what the business needs. This page describes the problem the support system solves; Plan the system turns it into the pieces you will build.

The problem

Customers report problems by phone and email. Requests get lost, nobody can say who is handling what, customers do not know where their request stands, and managers have no figures on how quickly problems are solved.

The goals

- **One place for every request.** Customers open a support ticket themselves, and every ticket gets a number.
- **Clear routing.** The support desk sees every new ticket and passes it to the right representative.
- **Quality control.** The desk reviews each solution before the customer sees it, and can ask a manager to approve it.
- **Customers kept informed.** Customers are told when their ticket is received and when it is solved, and are asked to rate the service.
- **Figures for managers.** A chart of tickets by status, and a monthly report.

Who uses it

Who	Needs
Customers — people or organizations who use the company's products	A simple way to open a ticket and follow it.
Support teams — the support desk and support representatives	A clear queue of tickets to handle, and the details of each ticket.
Managers	Control over important answers, and figures on team performance.

What else the business asked for

These requirements shape how you set up the system, rather than adding steps of their own:

- **Permissions.** Each person sees and changes only what their role needs. You handle this with the workflow's step owners and field rules, and with view access.
- **Mobile.** Customers can use the system on a phone. The mobile app shows the same views and workflow without extra work.
- **Volume.** The system must handle many tickets without slowing down. Queries and views keep each person's list short.

Next step

Plan the system.

Plan the system

This page turns the business need into the exact pieces you will build. Every later page uses the names below.

The Support Tickets entity

Field	Type	Required	Notes
Ticket number	Sequence	—	Numbered automatically.
Ticket title	Localized Text	Yes	
Ticket description	Localized Text	Yes	
Ticket status	Category	Yes	Options **Open**, **In Progress**, **Responded**, **Duplicate**, **Closed**. Default **Open**.
Assigned to	User	No	The support representative. Filled in by the support desk, not by the customer.
Start date	Date-time	Yes	When the ticket was opened.
End date	Date-time	No	Filled in when the ticket is closed.
Solution	Block	—	Holds one field, *Solution* (Localized Text).
Service evaluation	Block	—	Holds *Evaluation score* (Category, options **1** to **5**) and *Comment* (Long-text).

The workflow

Five steps, from the customer's ticket to the customer's rating:

Step	Owner	Actions
Open Ticket	The customer who opens the ticket	**Send Request** → *Help Desk*
Help Desk	The *Technical Support Office* group	**Send to Processing** → *Processing Ticket* · **Send Solution** → *Service Evaluation* · **Manager Approval** → *Manager Approval* · **Close Ticket** → end
Manager Approval	The support manager	**Approve** → *Help Desk* · **Reject** → *Help Desk*
Processing Ticket	The person in *Assigned to*	**Review Solution** → *Help Desk*
Service Evaluation	The customer (owner of *Open Ticket*)	**Done** → *Help Desk*

The notifications

Seven reactions keep everyone informed. They are listed with their full settings on Add the notifications.

Notification	Goes to	When
New Ticket	The support desk	A ticket reaches *Help Desk*
Reply to customer	The customer	The ticket reaches *Processing Ticket* — it has been received and assigned
New ticket assigned to you	The representative	The ticket reaches *Processing Ticket*
Review Solution	The support desk	The representative sends the solution for review

Notification	Goes to	When
The solution has been reviewed	The representative	The desk sends the solution to the customer
Ticket Resolved	The customer	The desk sends the solution to the customer
Service Evaluation	The support desk	The customer has rated the service

Queries, views, summary and report

Piece	Name	Purpose
Query	*Closed tickets*	Tickets whose status is **Closed**.
Query	*All tickets*	Every ticket; the source of the monthly report.
View	*Ticket Request*	Where customers open and follow their tickets.
View	*Incoming Requests*	Where the support desk and representatives handle tickets, with the status chart.
Summary	*Ticket Status*	A doughnut chart counting tickets by status.
Report	*Performance Report*	Every ticket with its status, emailed to the manager each month.

Prepare accounts and groups

Ask your tenant administrator for these before you start.

Add users

Test accounts with email addresses you can read:

- *Customer One* — opens tickets.
- *Desk Agent* — works on the support desk.
- *Representative One* — solves tickets.
- *Support Manager* — approves solutions.

Administrators add users as described in Managing users [Administration].

Add the support desk group

A group named *Technical Support Office*, containing *Desk Agent*. Groups are managed as described in Managing groups [Administration].

Next step

The build steps.

The build steps

Build the system in this order. Each step uses what the step before it created.

1. **Add users** and **the support desk group** — ask your tenant administrator for the test accounts and the *Technical Support Office* group.
2. **Create the module** — *Customer Support System*.
3. **Create the entity** — *Support Tickets* and its fields.
4. **Build the workflow** — the five steps, their owners, actions and field rules.
5. **Add the notifications** — seven reactions, one for each hand-over.
6. **Create the queries** — *Closed tickets* and *All tickets*.
7. **Build the views** — *Ticket Request* for customers and *Incoming Requests* for the support team.
8. **Add the status chart** — *Ticket Status*, shown in *Incoming Requests*.
9. **Create the monthly report** — *Performance Report*.
10. **Test the system** — one ticket from opening to closing, as each role.

Build the system

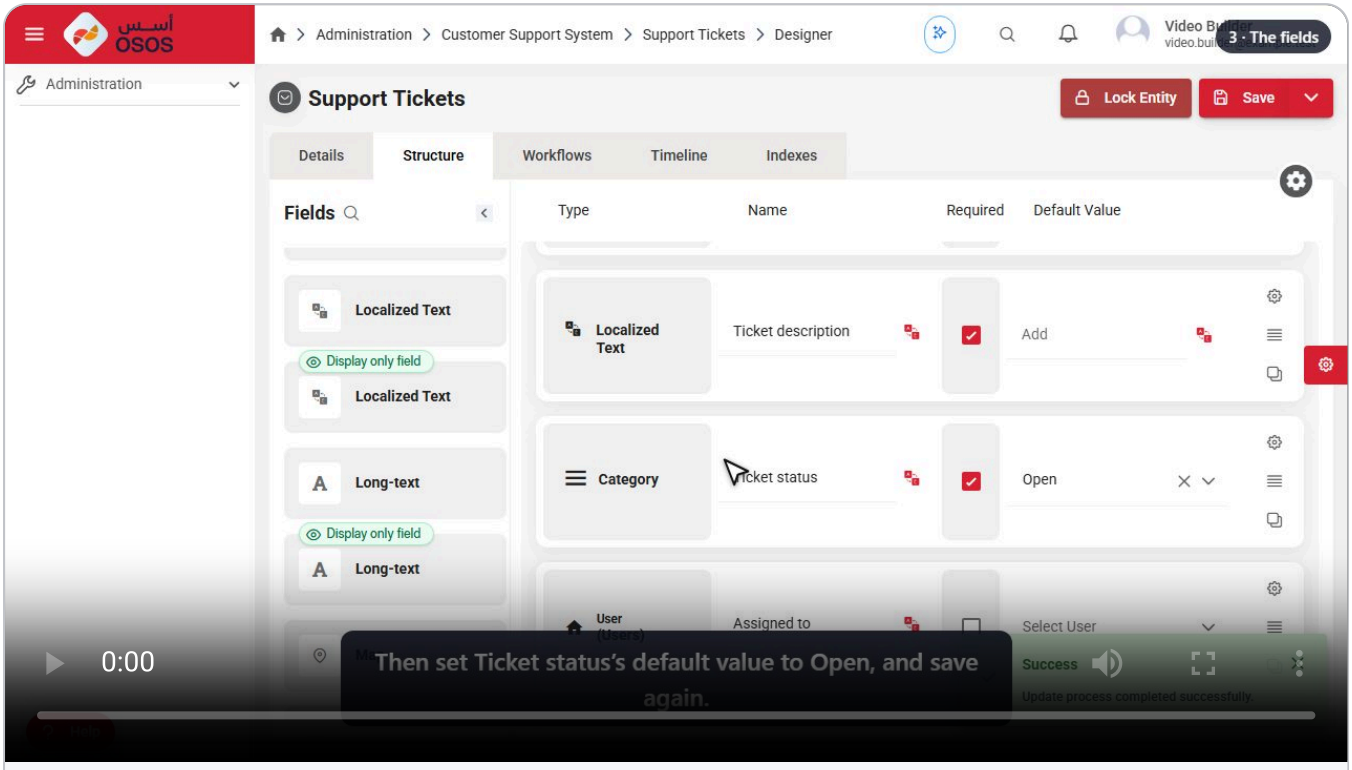
Do these in order; each one uses what the one before it created.

1. Create the module
2. Create the entity
3. Build the workflow
4. Add the notifications
5. Create the queries
6. Build the views
7. Add the status chart
8. Create the monthly report

Then test the system.

Create the module

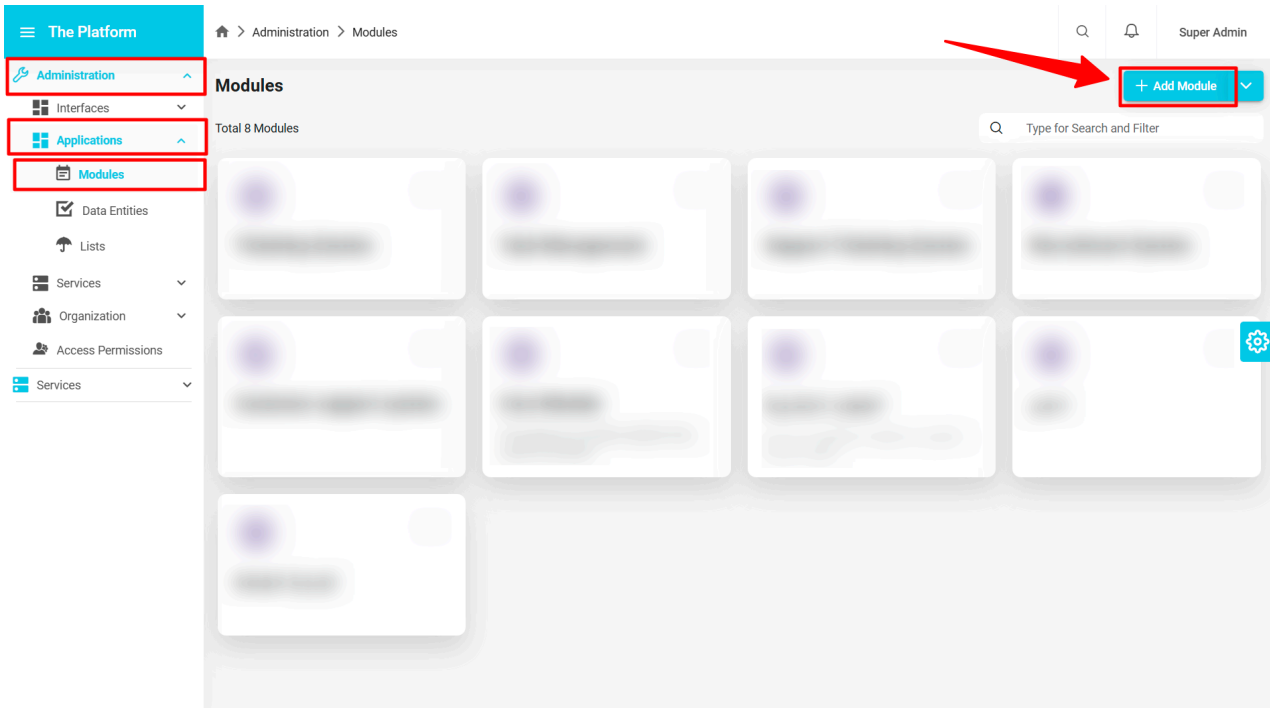
The module holds everything in the support system and gives it its entry in the sidebar.



TUTORIAL · PART 1 OF 8 The module and the ticket

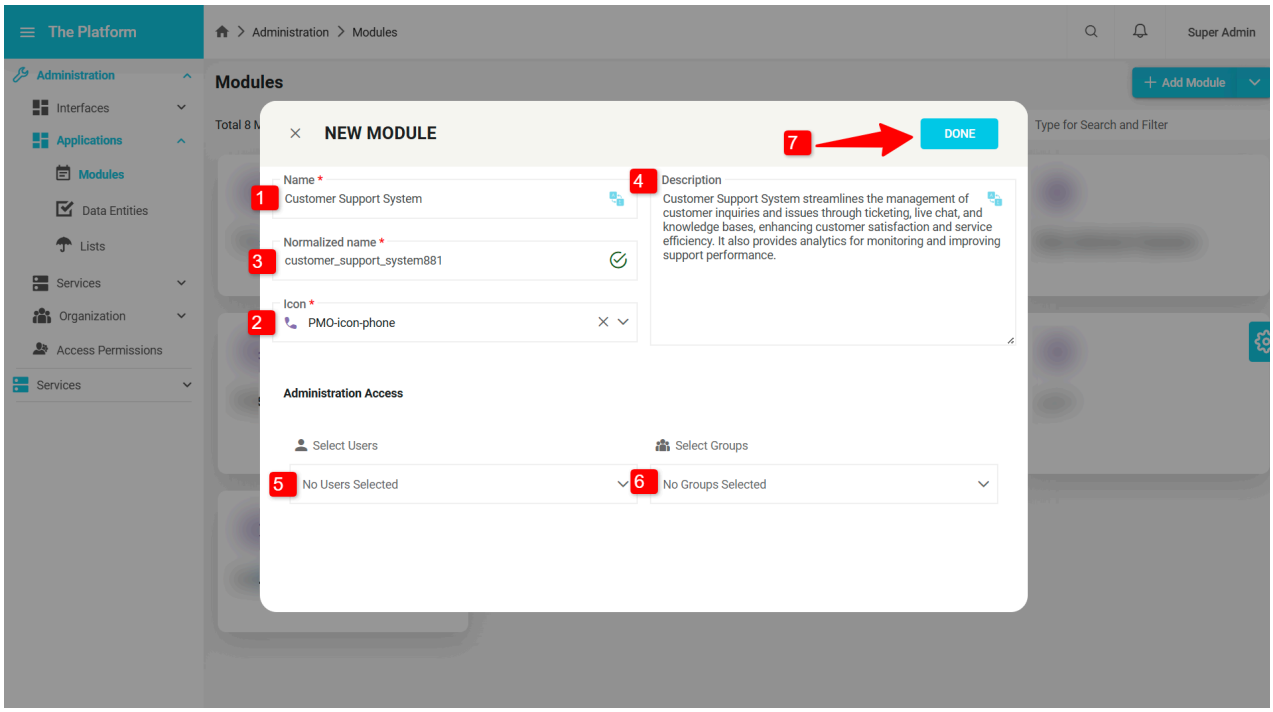
The Customer Support System module and its Support Tickets entity · 3:14 · Recorded on OSOS 1.11.0-alpha.1

1. Open **Administration › Applications › Modules** and select **Add Module**.

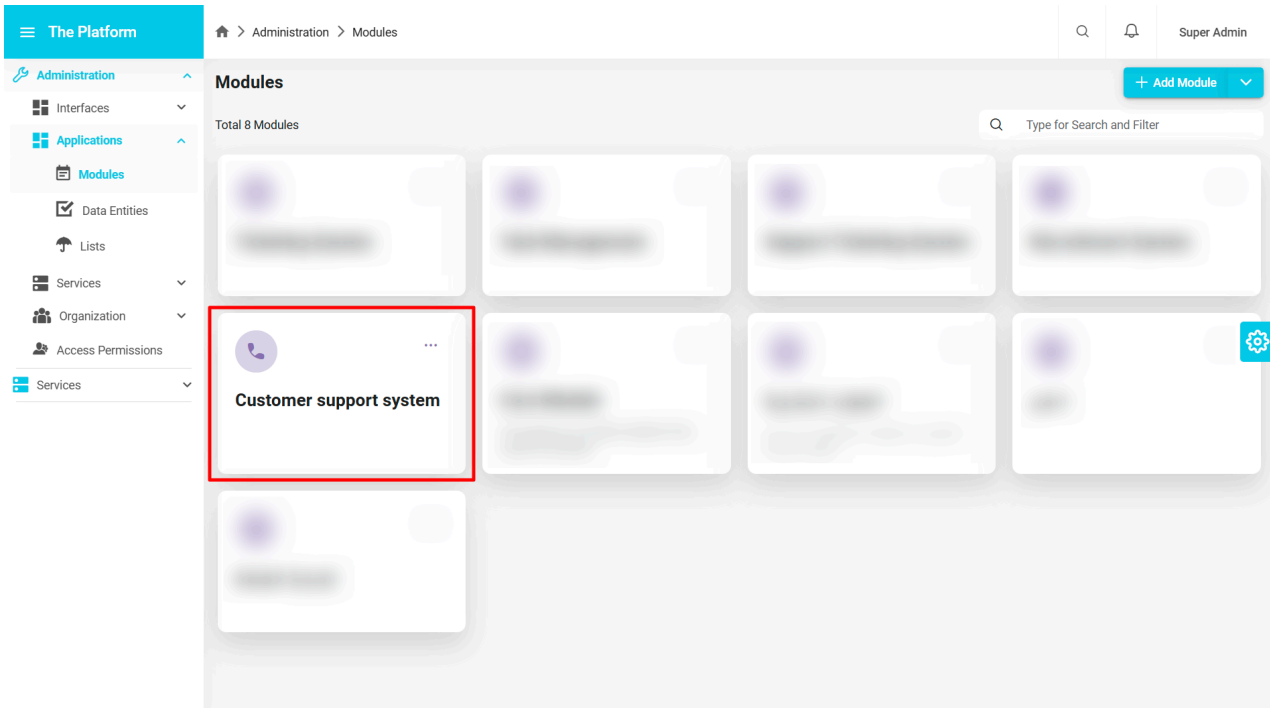


2. In the **New Module** dialog, fill in:

   - **Name** — *Customer Support System*, in Arabic and English.
   - **Normalized name** — keep the one filled in for you. It cannot be changed after the module is created.
   - **Icon** — *phone*.
   - **Description** — optional.
   - **Administration Access** — leave empty unless other builders will work on the module.



3. Select **Done**. The module is listed.

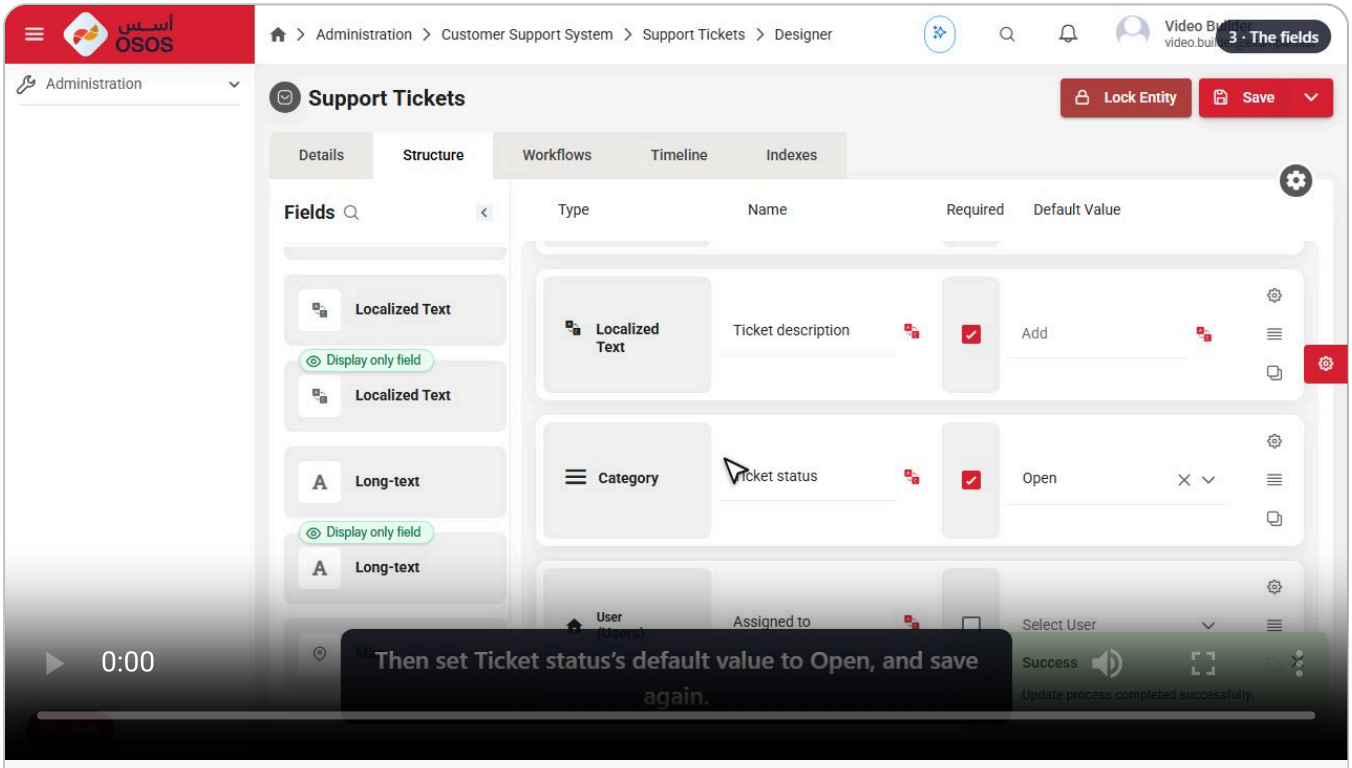


Every option is described in the recipe Creating a module.

Next: Create the entity.

Create the entity

Every ticket is a record of the *Support Tickets* entity. On this page you create the entity and its fields. The workflow and notifications come next, on their own pages.

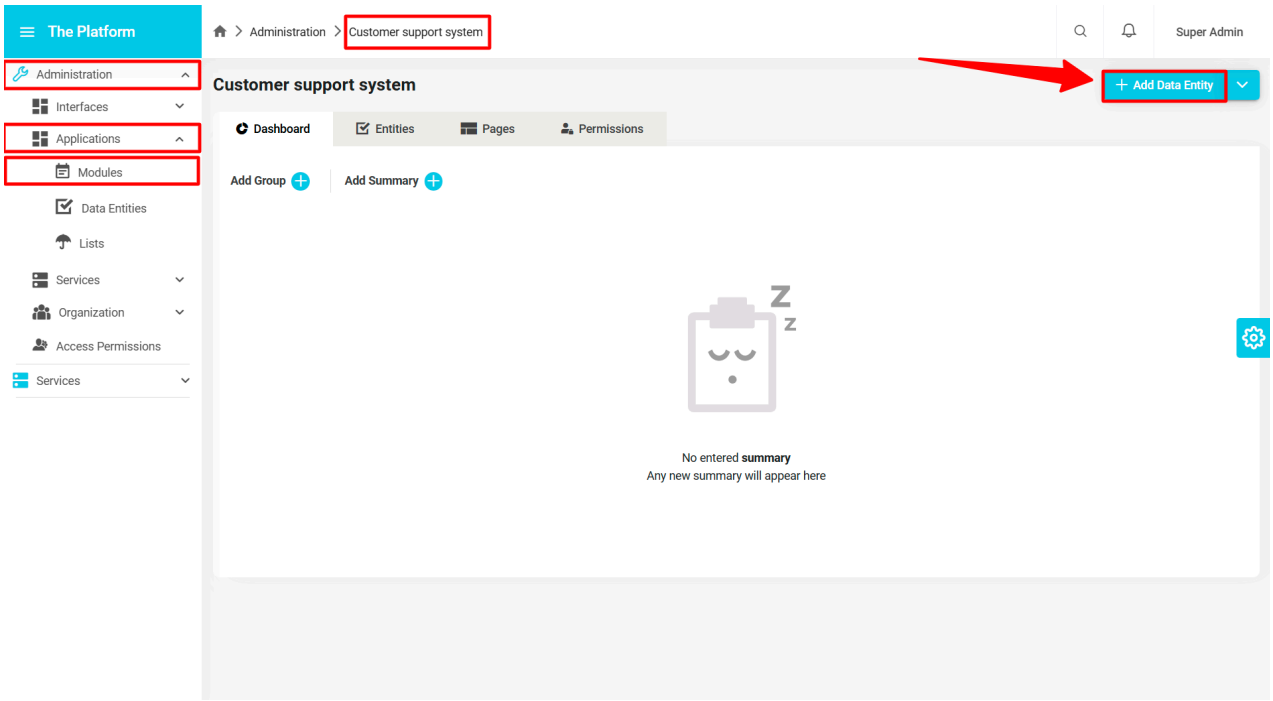


TUTORIAL · PART 1 OF 8 **The module and the ticket**

The Customer Support System module and its Support Tickets entity · 3:14 · Recorded on OSOS 1.11.0-alpha.1

Create Support Tickets

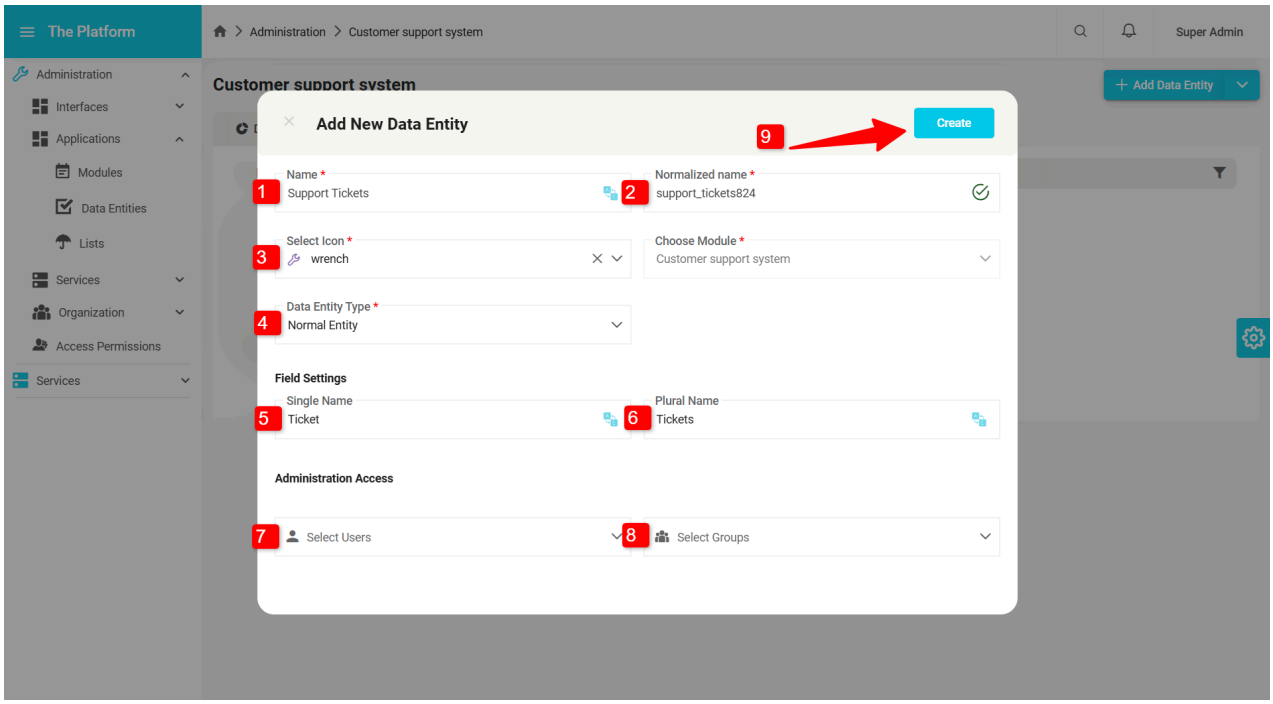
1. Open the *Customer Support System* module. On its **Entities** tab, select **Add Data Entity**.



2. In the **Add New Data Entity** dialog, fill in:

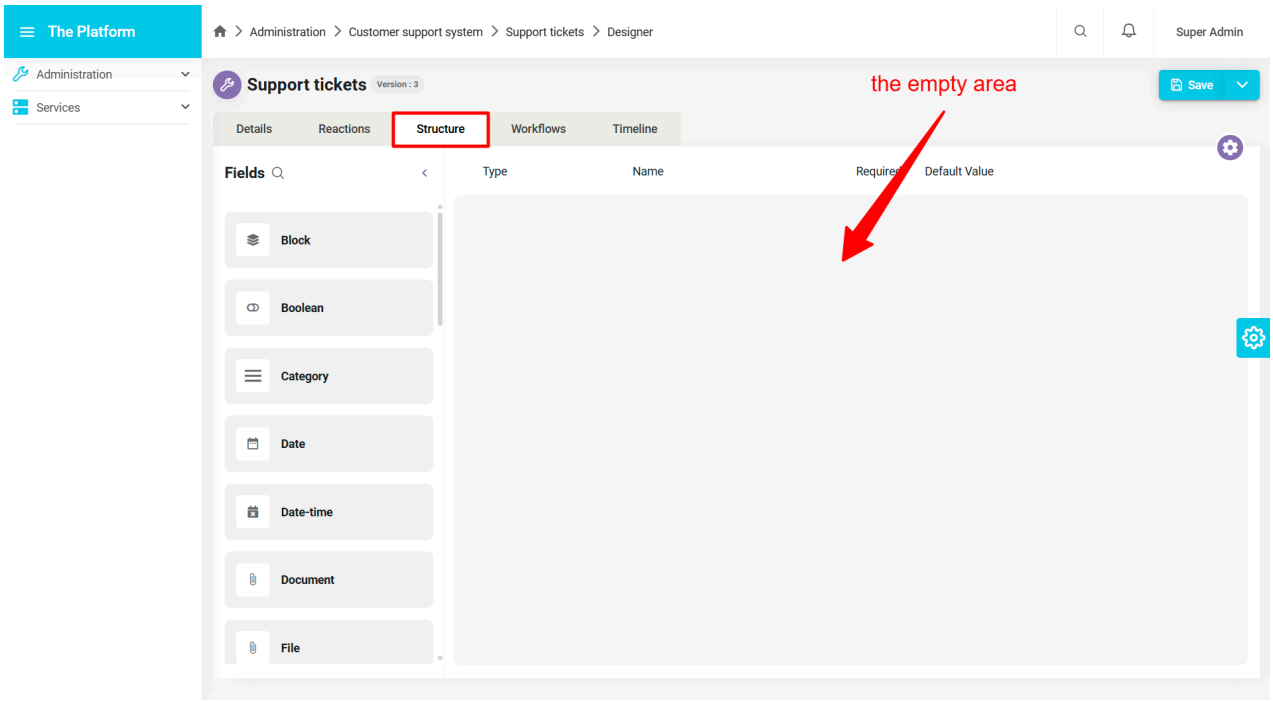
   - **Name** — *Support Tickets*, in Arabic and English.
   - **Normalized name** — keep the one filled in for you.
   - **Select Icon** — *wrench*.
   - **Choose Module** — already set to *Customer Support System*.
   - **Data Entity Type** — **Normal Entity**.
   - Under **Field Settings**: **Single Name** *Support Ticket*, **Plural Name** *Support Tickets*, in Arabic and English.

3. Select **Create**.



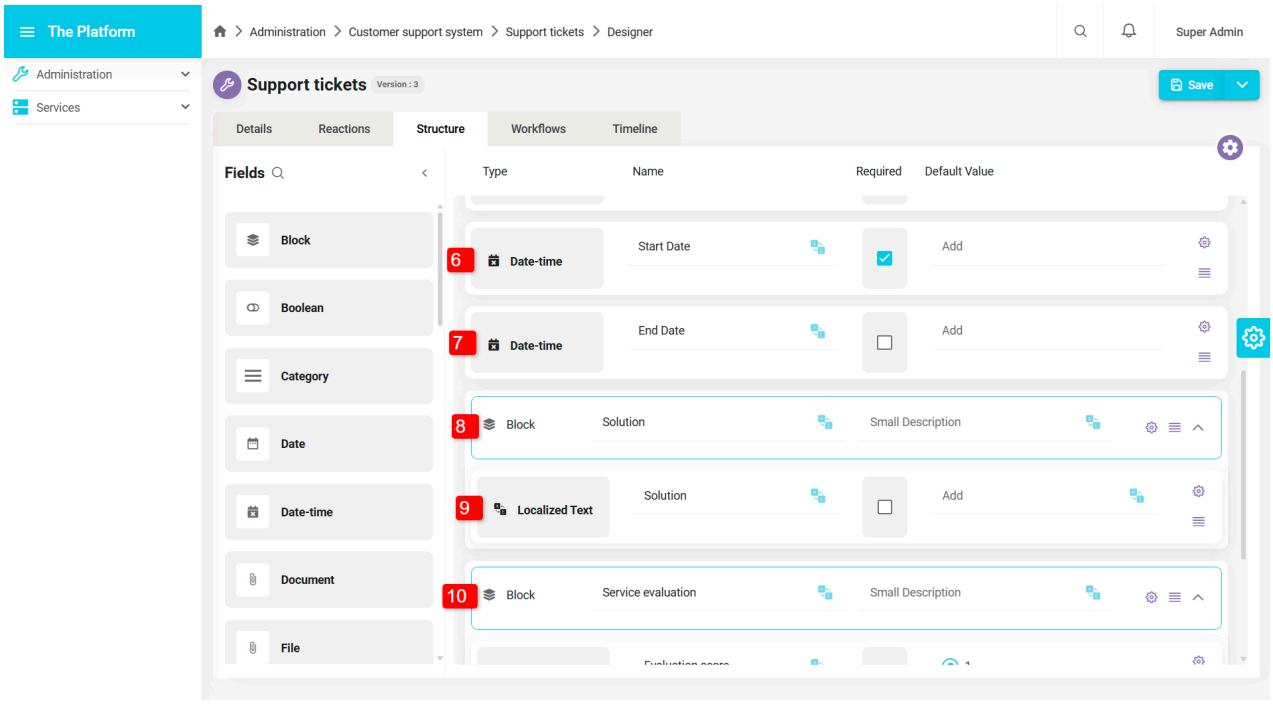
Add the fields

1. Open the entity's designer (**Edit Entity**) and select the **Structure** tab.

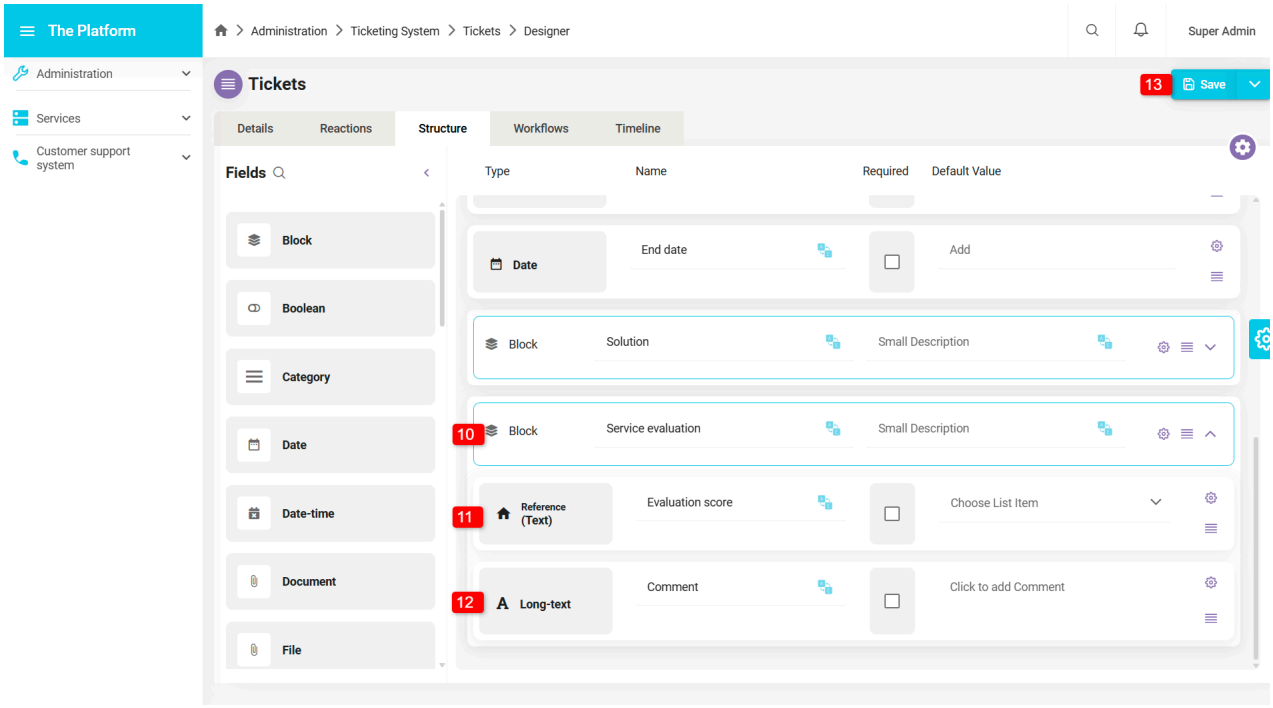


2. Drag each field type from the **Fields** palette onto the empty area and name it:

Drag this type	Name it	Then
Sequence	*Ticket number*	Numbers tickets automatically.
Localized Text	*Ticket title*	Tick **Required**.
Localized Text	*Ticket description*	Tick **Required**.
Category	*Ticket status*	Tick **Required**. Add the options **Open**, **In Progress**, **Responded**, **Duplicate** and **Closed**; set **Default Value** to **Open**.
User	*Assigned to*	Leave **Required** off — the support desk fills it in.
Date-time	*Start date*	Tick **Required**.
Date-time	*End date*	Leave **Required** off — it is filled in when the ticket is closed.
Block	*Solution*	Drag a **Localized Text** field into the block and name it *Solution*.
Block	*Service evaluation*	Drag a **Category** field into the block, name it *Evaluation score*, and add the options **1**, **2**, **3**, **4**, **5**. Then drag a **Long-text** field into the block and name it *Comment*.



3. Select **Save**, then **Save Changes**.



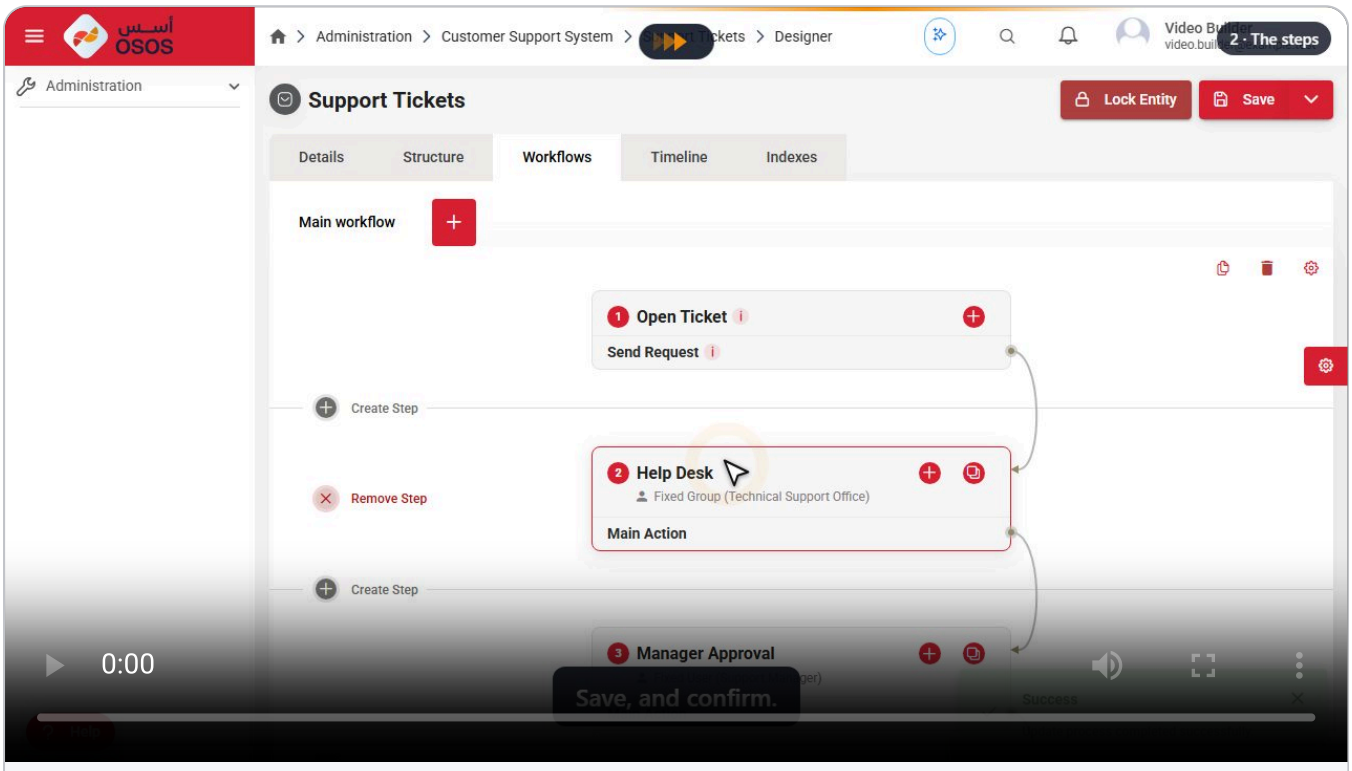
A **block** groups fields that belong together. Here the blocks keep the representative's solution and the customer's rating apart, so the workflow can show or hide each block as a whole at each step.

Field types and their options are described in Field types; entities in general in Entities.

Next: Build the workflow.

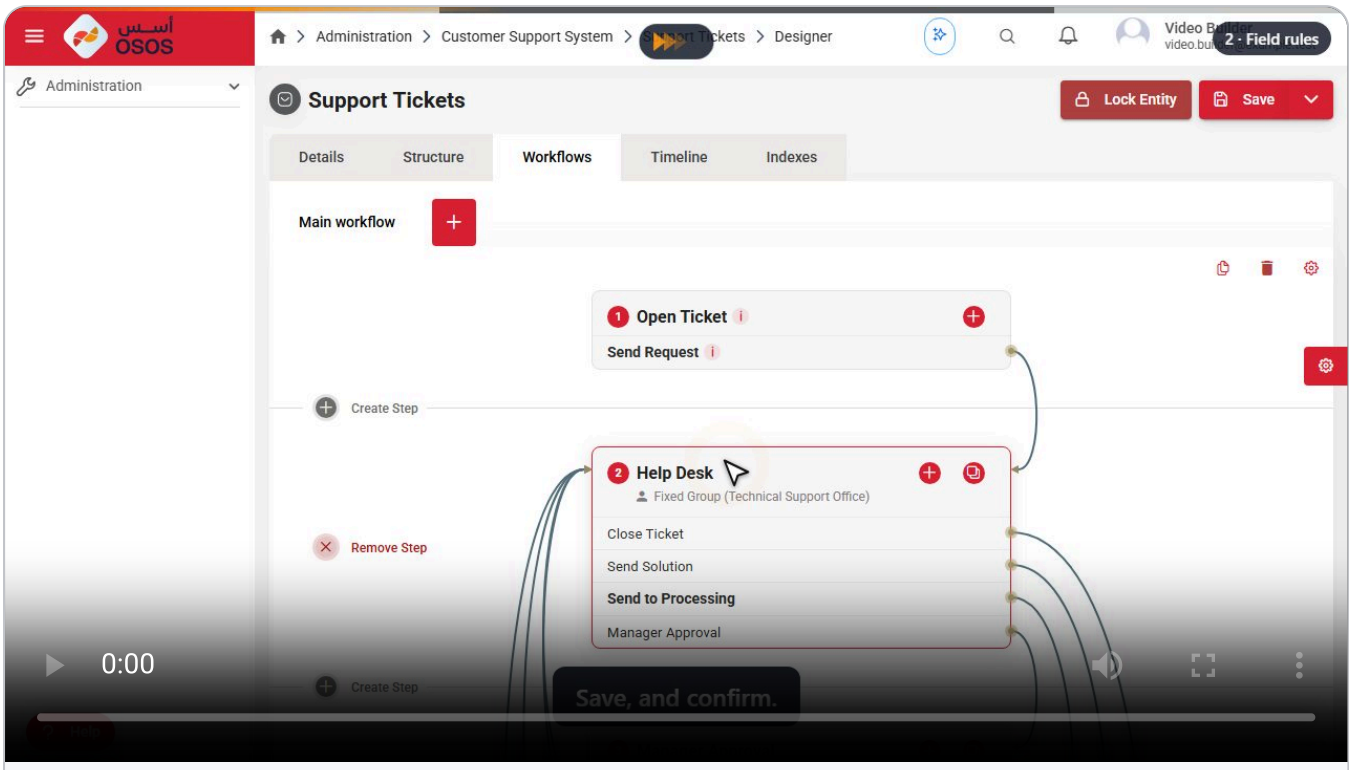
Build the workflow

The workflow is the route a ticket takes from the customer to the support desk, the representative, a manager if needed, and back to the customer. On this page you build it in four passes: create the workflow, add the steps, connect the steps with actions, and set the field rules.



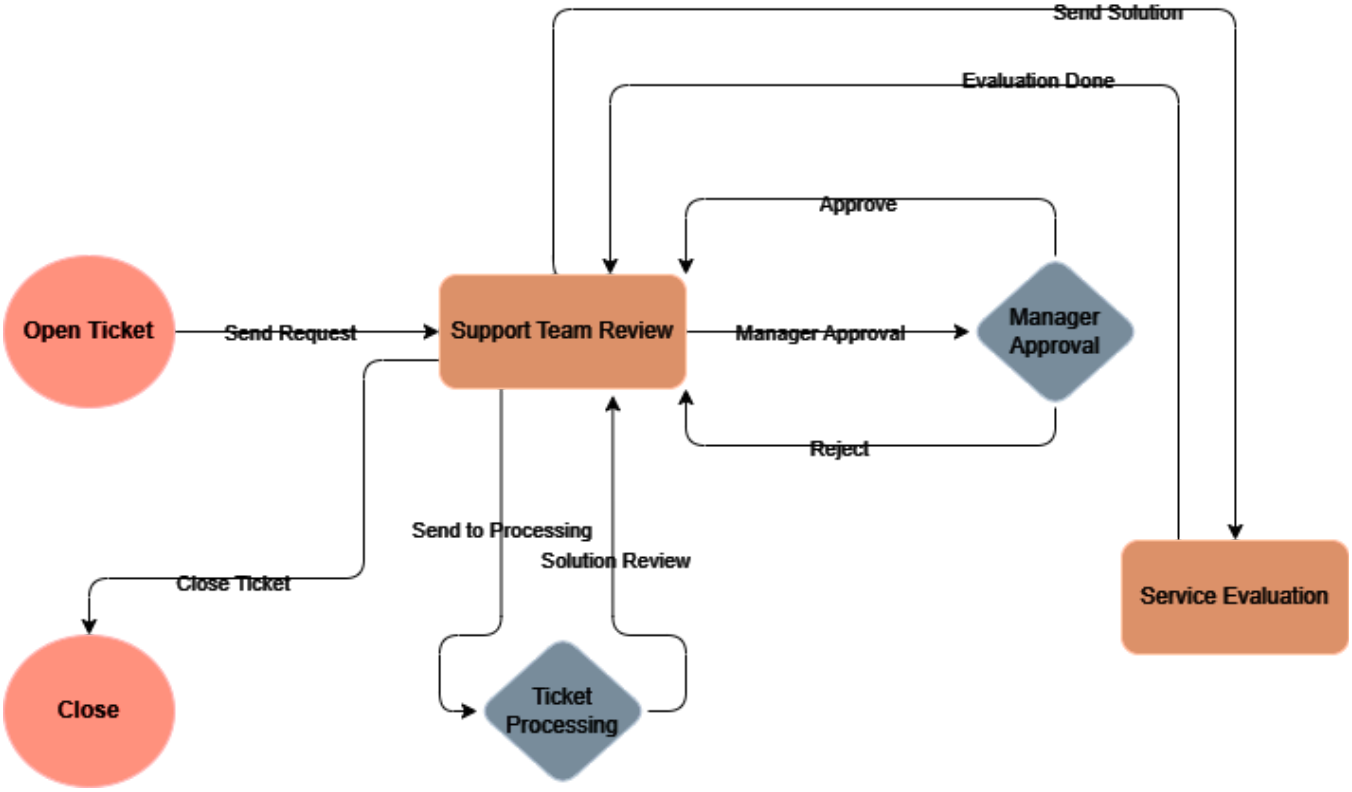
TUTORIAL · PART 2 OF 8 **Five steps, three kinds of owner**

The ticket workflow: its steps and who owns each one · 3:20 · Recorded on OSOS 1.11.0-alpha.1



TUTORIAL · PART 3 OF 8 Actions and field rules

Connect the steps, and decide what each step may change · 3:17 · Recorded on OSOS 1.11.0-alpha.1



In the diagram the *Help Desk* step is labelled *Support Team Review*, and the *Done* action *Evaluation Done*; use the names in the tables below.

1. Create the workflow

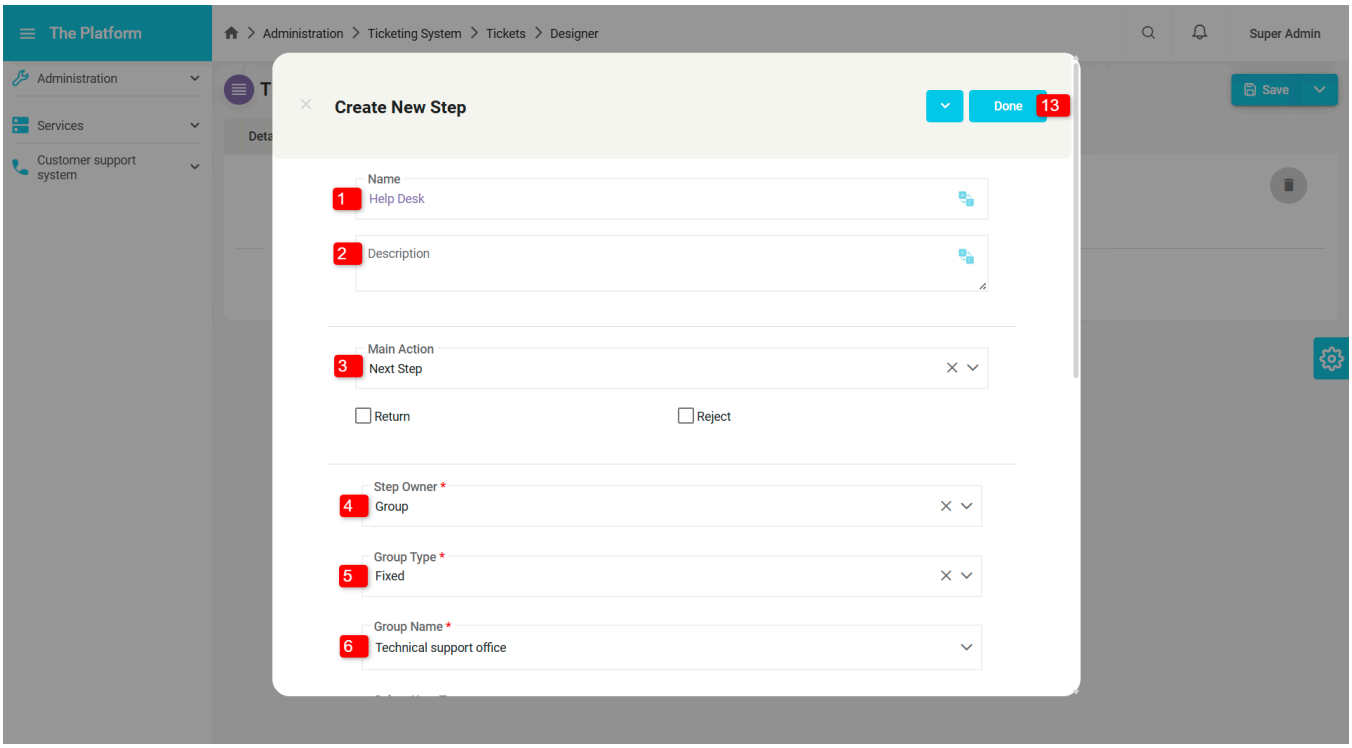
1. Open the *Support Tickets* designer (**Edit Entity**) and select the **Workflows** tab.
2. In the empty tab, select **No Workflow Exist! Tap To Enable Workflow**. The workflow starts with an *Initial Step* and a *Complete Step*.
3. Select the *Initial Step* and rename it *Open Ticket*. Select its action and, in **Action Settings**, rename it *Send Request*. The initial step always belongs to the person who opens the ticket.

2. Add the steps

Select **Create Step** below *Open Ticket* and create each step in this order, so they sit between *Open Ticket* and *Complete Step*. In the **Create New Step** dialog, enter the **Name**, keep **Step Type Normal**, leave **Return** and **Reject** unticked, set the owner, set **Main Action** to **Next Step** (you point each action at its real step in section 3), and select **Done**. **Done** stays unavailable until every required setting is filled in.

Step	Step Owner	Then set
Help Desk	**Group**	**Group Type Fixed**, **Group Name** *Technical Support Office*; **Select User Type Single User**; **Choose Selection Type Take Ownership**, so any desk agent can claim the ticket; **Access Permission All Users** (required with Take Ownership).
Manager Approval	**User**	**User Type Fixed**, **User Name** *Support Manager*.
Processing Ticket	**User**	**User Type Field**, **Field Name** *Assigned to* — the representative the desk picks.
Service Evaluation	**User**	**User Type Previous Step Owner**, **Step Name** *Open Ticket* — the customer.

Whenever the owner is not one fixed person, the dialog also asks for **Fallback Assignment 1** — who gets the ticket if the owner cannot be found, for example when *Assigned to* is empty. Choose **User**, **Fixed**, *Support Manager*.

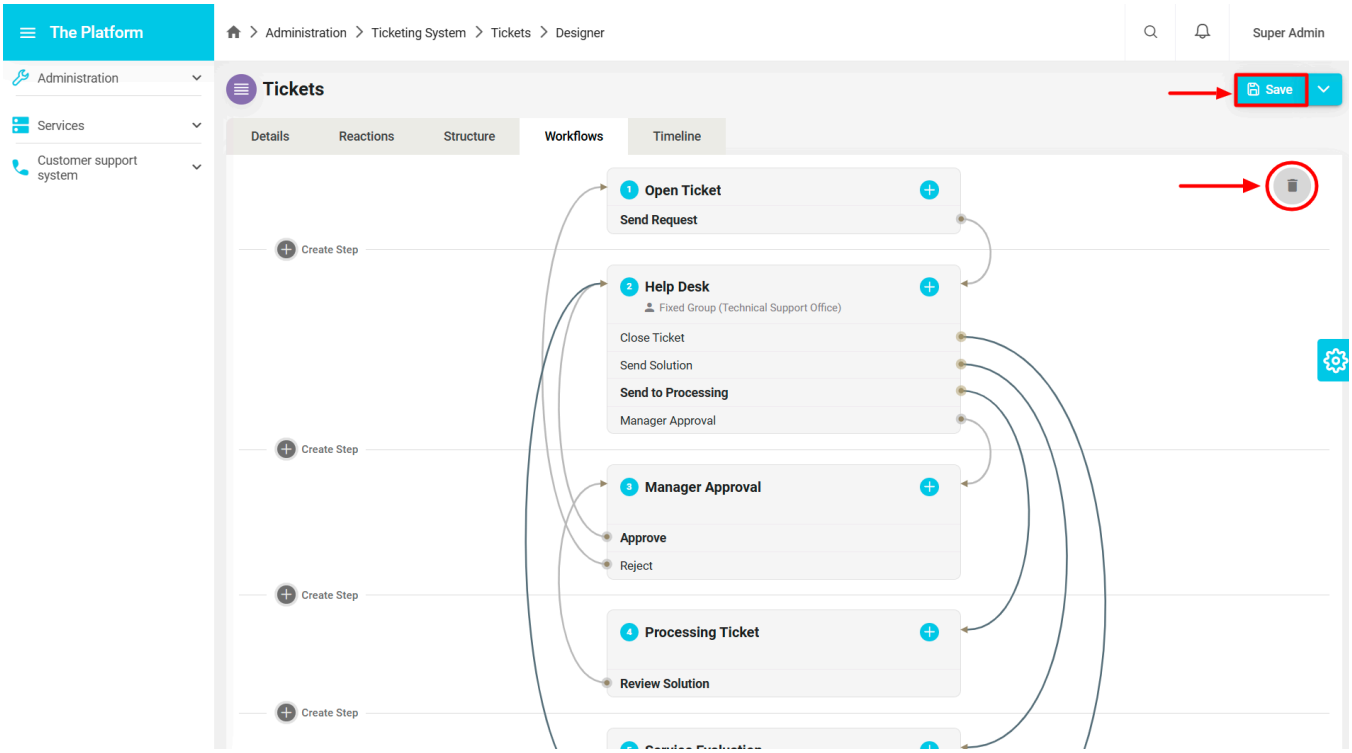


3. Connect the steps with actions

Each step was created with one main action. Rename it and point it at the right step, then add the other actions with **Create Action** on the step. In **Action Settings**, set the **Name** and the **Default Target** (pick the step under **Existing Steps**). Leave **Main Action** on only for the first action in each row.

Step	Action	Default Target
Open Ticket	*Send Request* (main)	*Help Desk*
Help Desk	*Send to Processing* (main)	*Processing Ticket*
	Manager Approval	*Manager Approval*
	Send Solution	*Service Evaluation*
	Close Ticket	*Complete Step*
Manager Approval	*Approve* (main)	*Help Desk*
	Reject	*Help Desk*
Processing Ticket	*Review Solution* (main)	*Help Desk*
Service Evaluation	*Done* (main)	*Help Desk*

So the desk is always in control: every route comes back to *Help Desk*, and only the desk closes a ticket.

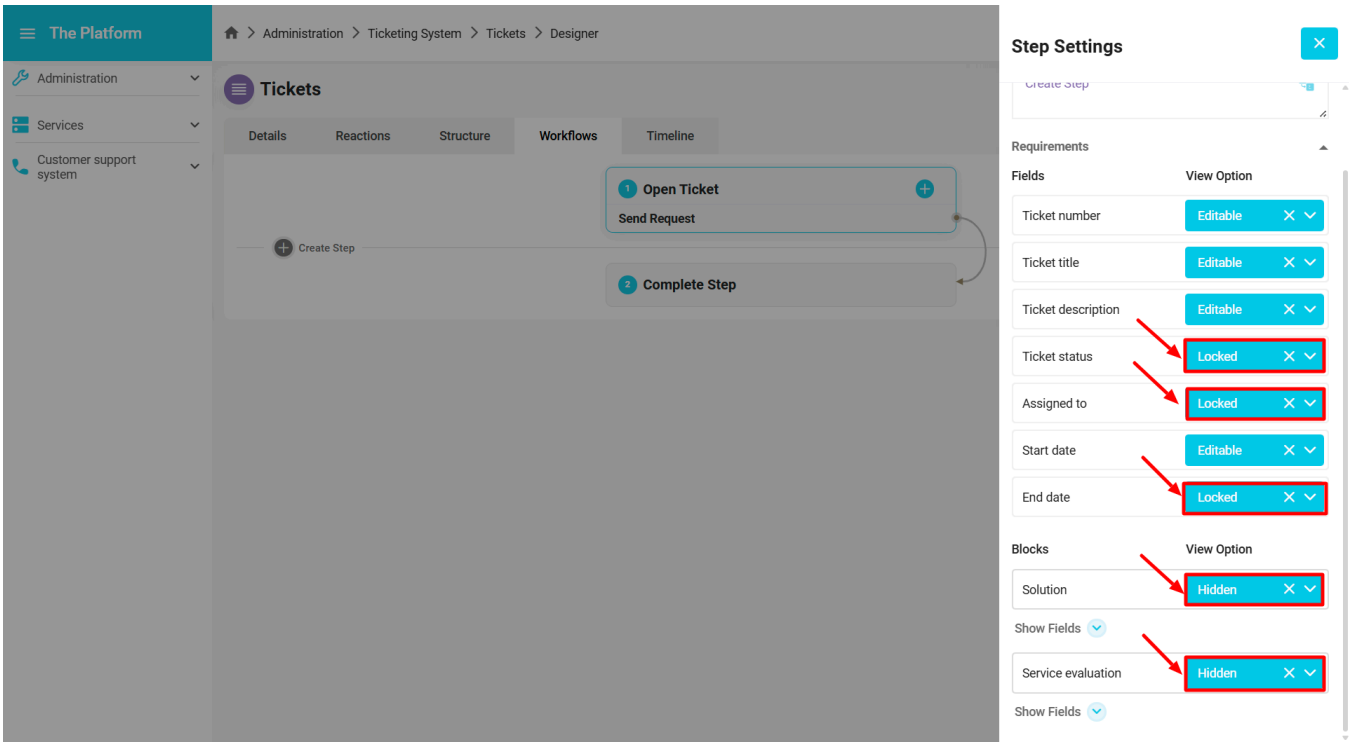


4. Set the field rules for each step

Select a step to open its **Step Settings**, open **Requirements**, and set each field's **View Option**. Blocks are listed under **Blocks** with their own **View Option**; select **Show Fields** under a block to set its fields one by one.

Field	Open Ticket	Help Desk	Manager Approval	Processing Ticket	Service Evaluation
Ticket title, Ticket description	**Mandatory**	**Locked**	**Locked**	**Locked**	**Locked**
Ticket status	**Locked**	**Editable**	**Locked**	**Editable**	**Locked**

Field	Open Ticket	Help Desk	Manager Approval	Processing Ticket	Service Evaluation
Assigned to	**Locked**	**Editable**	**Locked**	**Locked**	**Locked**
Start date	**Mandatory**	**Locked**	**Locked**	**Locked**	**Locked**
End date	**Locked**	**Editable**	**Locked**	**Locked**	**Locked**
Solution (block)	**Hidden**	**Locked**	**Locked**	**Editable**	**Locked**
Service evaluation (block)	**Hidden**	**Locked**	**Hidden**	**Hidden**	**Editable**



Select **Save** at the top of the designer, then **Save Changes**.

Time limits

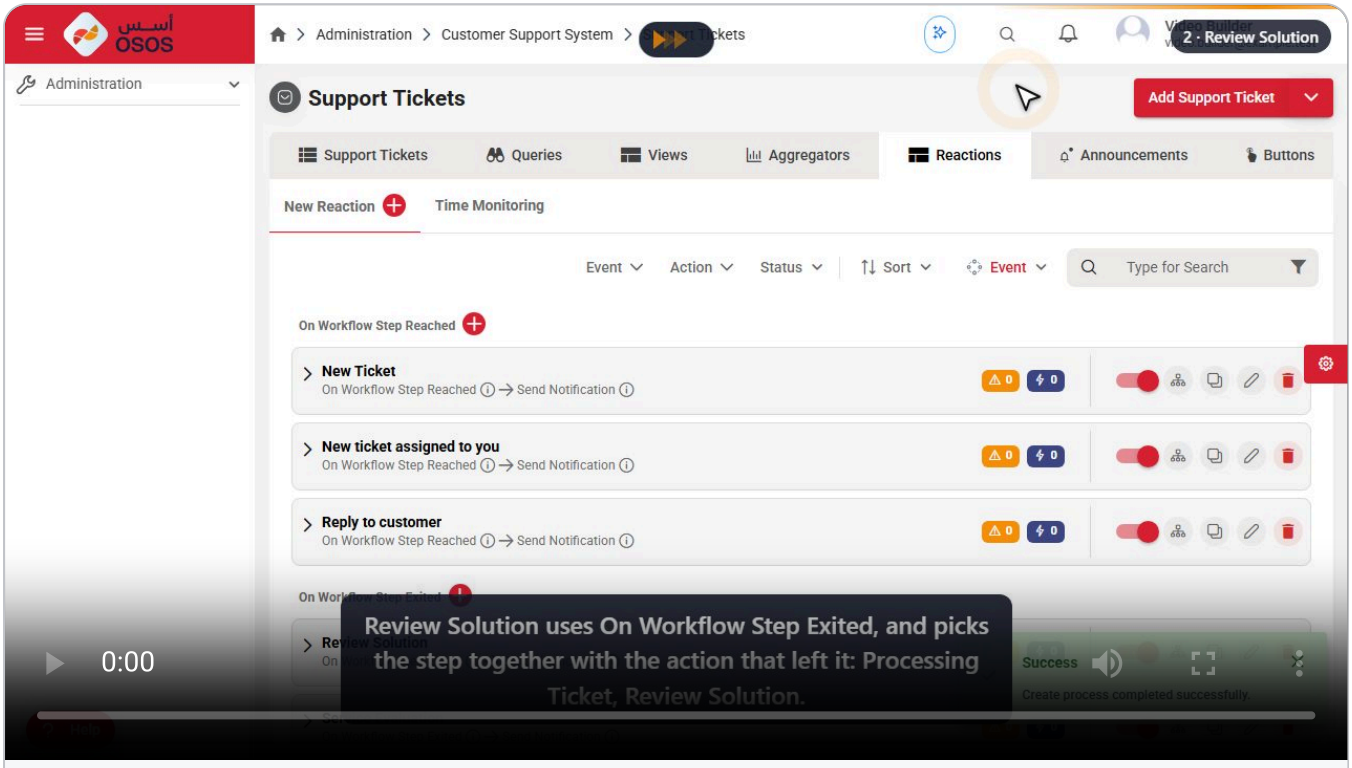
You can give a step a time limit — for example two working days for *Processing Ticket* — counted in working hours and skipping public holidays. A ticket that passes the limit becomes overdue, and a reaction on the **Time Monitoring** sub-tab can escalate it. See Time limits and escalation.

Every workflow option — group voting, conditional routes, returning for changes — is described in Workflows & approvals.

Next: Add the notifications.

Add the notifications

Every hand-over in the workflow should tell the next person that a ticket is waiting for them. You add seven **reactions**, all of the same kind: when the ticket reaches or leaves a workflow step, send an in-app notification.

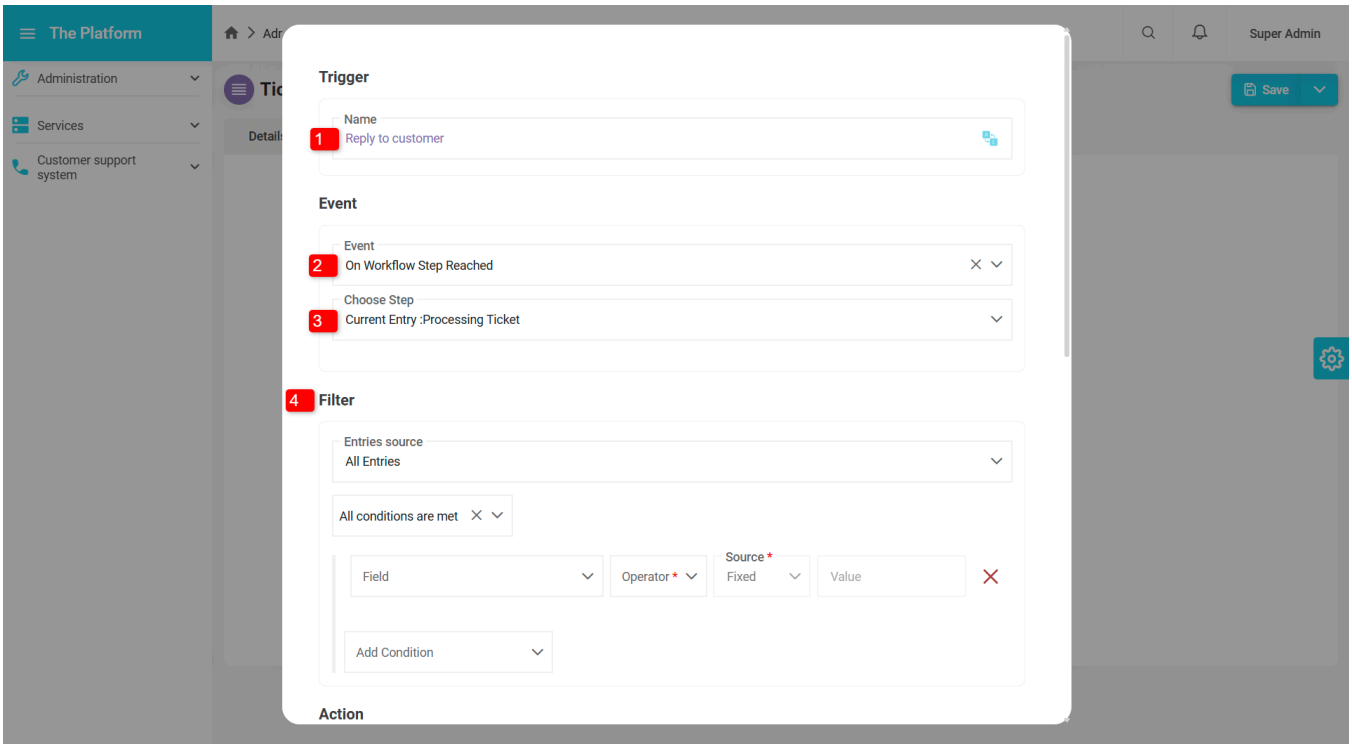


TUTORIAL · PART 4 OF 8 Seven notifications

Tell each person when a ticket reaches them · 2:08 · Recorded on OSOS 1.11.0-alpha.1

How to add each reaction

1. Open the *Support Tickets* entity and select the **Reactions** tab.
2. Select **+** (or **Add Reaction**). The **New Reaction** dialog opens.
3. Fill in the settings from the reaction's table below:
   - **Name**.
   - **Event** — **On Workflow Step Reached** or **On Workflow Step Exited** — and, in **Choose Step**, the step. For **On Workflow Step Exited**, pick the step together with the action it was left by, for example *Processing Ticket (Review Solution)*.
   - Leave **Filter** empty.
   - **Action Type Send Notification**, with **Channel Type In App**, the **Title** and the **Content**.
   - Under **Custom Variables**, add the variable *id* with **Type Field** and **Map to** *Ticket number*, so `{{id}}` in the title shows the ticket's number.
   - Under **Recipients**, the **Source**, **Type** and **Value**.
4. Select **Save**.



The seven reactions

All seven use **Action Type Send Notification**, **Channel Type In App** and the **Title** `Ticket Number {{id}}`.

1. New Ticket — tell the desk a ticket has arrived

Setting	Value
Event	**On Workflow Step Reached**
Choose Step	*Help Desk*
Content	*New ticket arrived*
Recipients	**Group** · **Fixed** · *Technical Support Office*

2. Reply to customer — confirm the ticket was received

Setting	Value
Event	**On Workflow Step Reached**
Choose Step	*Processing Ticket*
Content	*Your request has been received*
Recipients	**User** · **Field** · *Created by*

3. New ticket assigned to you — tell the representative

Setting	Value
Event	**On Workflow Step Reached**
Choose Step	*Processing Ticket*
Content	*Ticket assigned to you*
Recipients	**User** · **Field** · *Assigned to*

4. Review Solution — ask the desk to review the answer

Setting	Value
Event	**On Workflow Step Exited**
Choose Step	*Processing Ticket (Review Solution)*
Content	*Please review the solution*
Recipients	**Group** · **Fixed** · *Technical Support Office*

5. The solution has been reviewed — tell the representative

Setting	Value
Event	**On Workflow Step Exited**
Choose Step	*Help Desk (Send Solution)*
Content	*Solution has been reviewed*
Recipients	**User** · **Field** · *Assigned to*

6. Ticket Resolved — send the customer to the solution

Setting	Value
Event	**On Workflow Step Exited**
Choose Step	*Help Desk (Send Solution)*
Content	*Your request has been resolved, please review it*
Recipients	**User** · **Field** · *Created by*

7. Service Evaluation — tell the desk the customer has rated

Setting	Value
Event	**On Workflow Step Exited**
Choose Step	*Service Evaluation (Done)*
Content	*Service evaluation is done, click to see it*
Recipients	**Group** · **Fixed** · *Technical Support Office*

Good to know

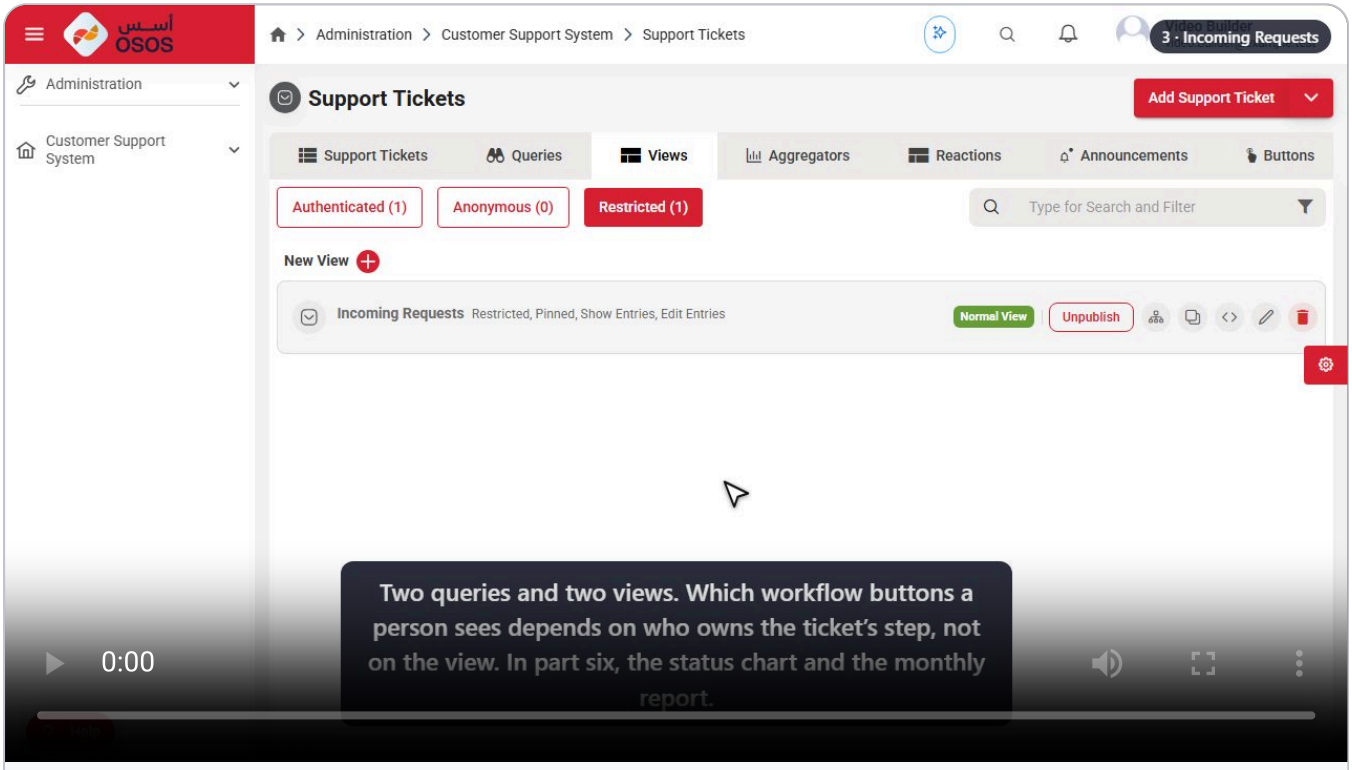
- **Reached or exited?** Use **On Workflow Step Reached** to tell the person who must act next. Use **On Workflow Step Exited** with a specific action when the message depends on the decision taken — reactions 5 and 6 fire only when the desk chose *Send Solution*, not when it closed the ticket.
- **Moving a ticket to another step is not an update.** A reaction on **On Entry Updated** does not run when the ticket moves through the workflow; use the workflow events.
- **One reaction, one action.** Reactions 5 and 6 share an event but send to different people with different messages, so they are two reactions.

Reactions can also send email or SMS, update the ticket or another record, or call another system — see Reactions & automation.

Next: Create the queries.

Create the queries

A **query** is a saved filter. You create two on *Support Tickets*:



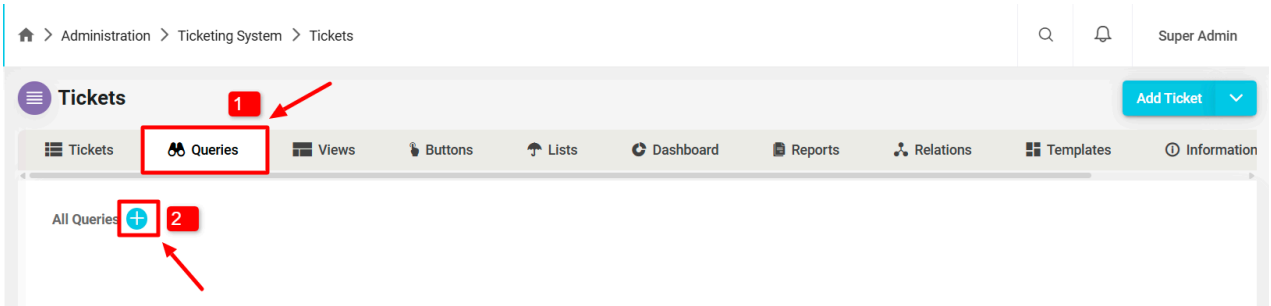
TUTORIAL · PART 5 OF 8 Queries and the two views

A saved filter, and the screens customers and the desk work in · 3:10 · Recorded on OSOS 1.11.0-alpha.1

Query	Condition	Used by
Closed tickets	*Ticket status* **Equal Closed**	Looking up finished tickets; a quick check of the workflow's end state
All tickets	*Ticket status* **Is filled** — every ticket has a status, so this matches every ticket	The monthly Performance Report, which must be based on a query

Steps

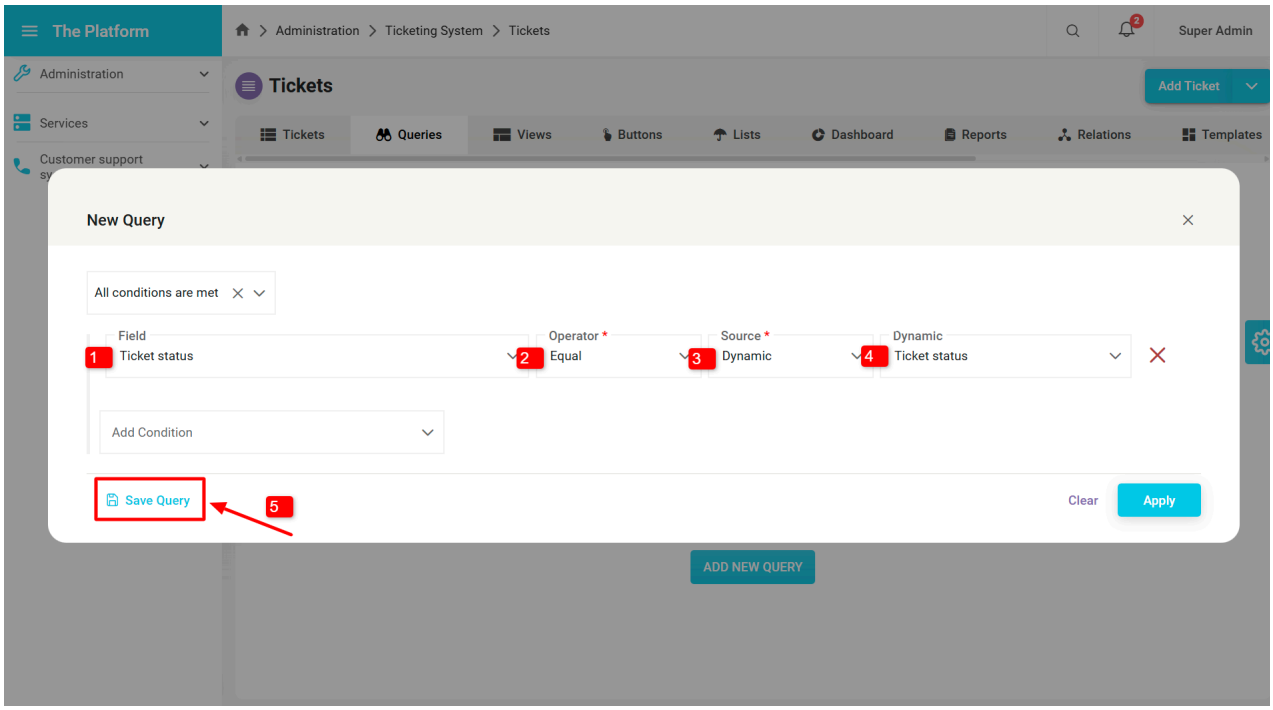
1. Open the *Support Tickets* entity, select the **Queries** tab, and select **+** (**New Query**).



2. Keep **All conditions are met**. Select **Condition** and set:
    - **Field** — *Ticket status*.
    - **Operator** — **Equal**.
    - Value — **Fixed**, **Closed**.

    In the field lists, a block appears as its inner fields: *Service evaluation* as *Evaluation score* and *Comment*, and *Solution* as *Solution*.

3. Under **Query Settings**, enter the **Query Name** *Closed tickets*, then select **Save Query**.



4. Repeat for *All tickets*, with the condition *Ticket status* **Is filled** (it needs no value).

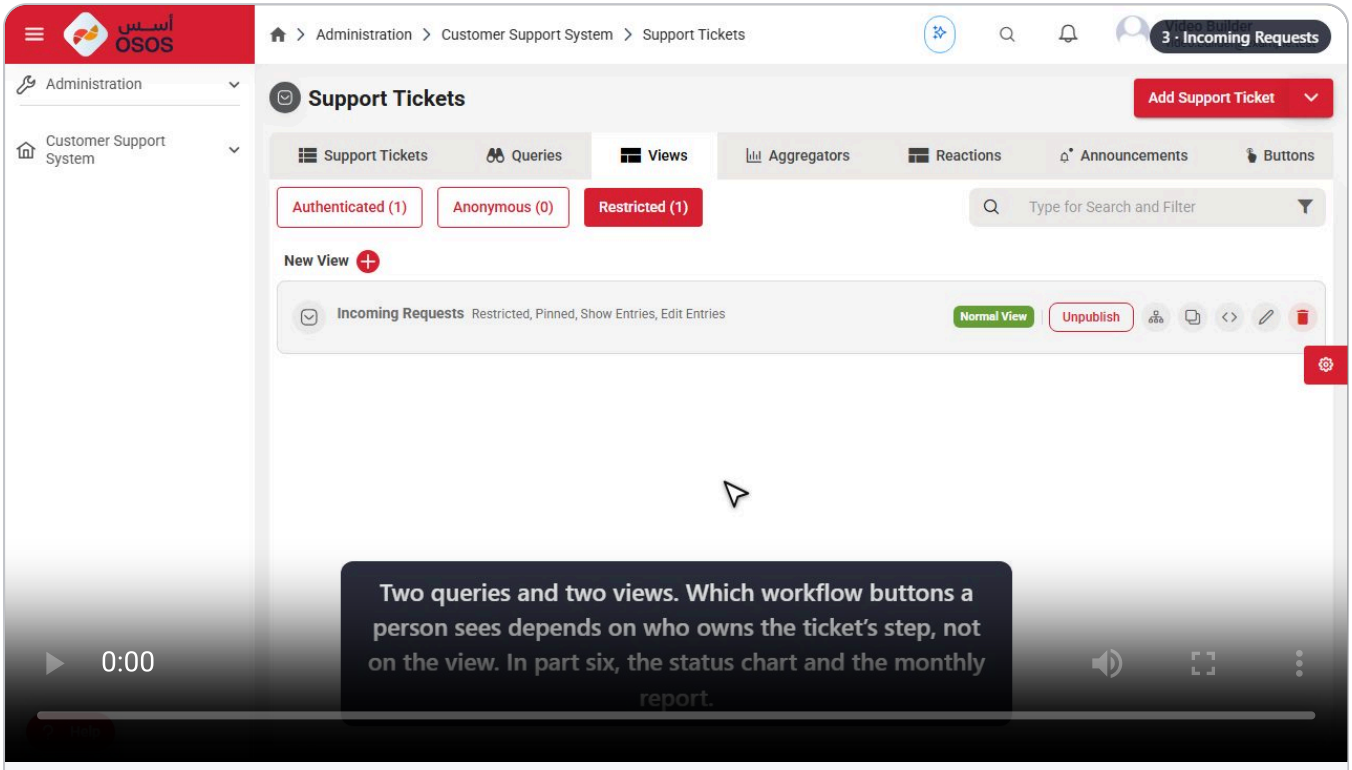
Check: on the **Queries** tab, select **Apply query** next to each query and compare the lists.

Queries can also follow the person looking (**@me**), ask for a value when they run, or become quick filters — see Queries.

Next: Build the views.

Build the views

Two views put the tickets in front of people:



TUTORIAL · PART 5 OF 8 Queries and the two views

A saved filter, and the screens customers and the desk work in · 3:10 · Recorded on OSOS 1.11.0-alpha.1

View	For	People can
Ticket Request	Every signed-in user (customers and staff)	Open a ticket, follow it, discuss it
Incoming Requests	The support team and the manager	Work on tickets, see the status chart

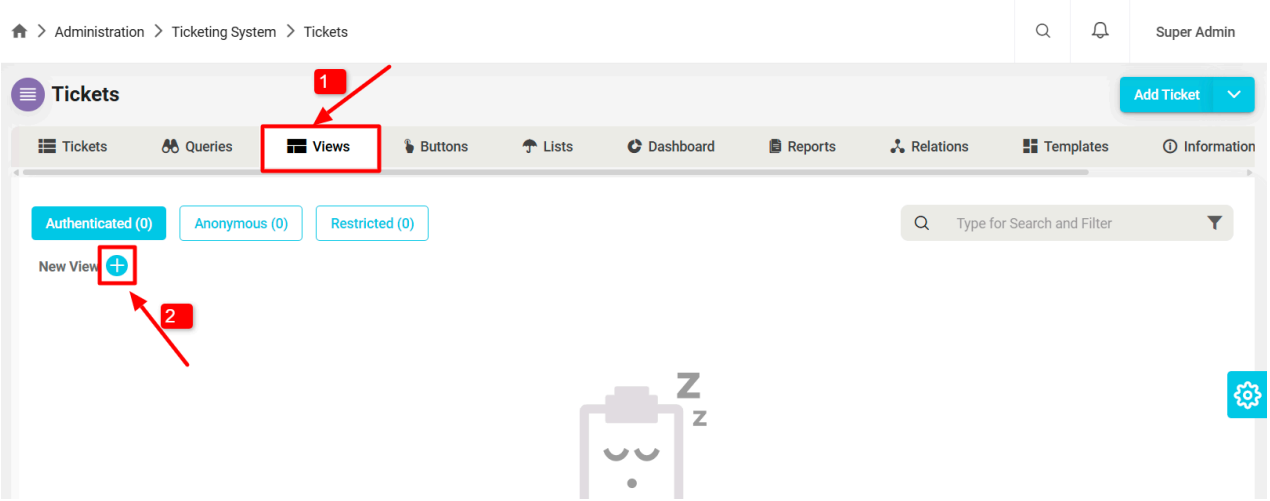
Which workflow buttons a person sees — *Send to Processing*, *Review Solution* and so on — depends on who owns the ticket's current step, not on the view.

Preview before publishing

While you design a view, **preview mode** shows it with sample records, and the mobile preview shows it on a phone. See Preview mode.

1. Build "Ticket Request"

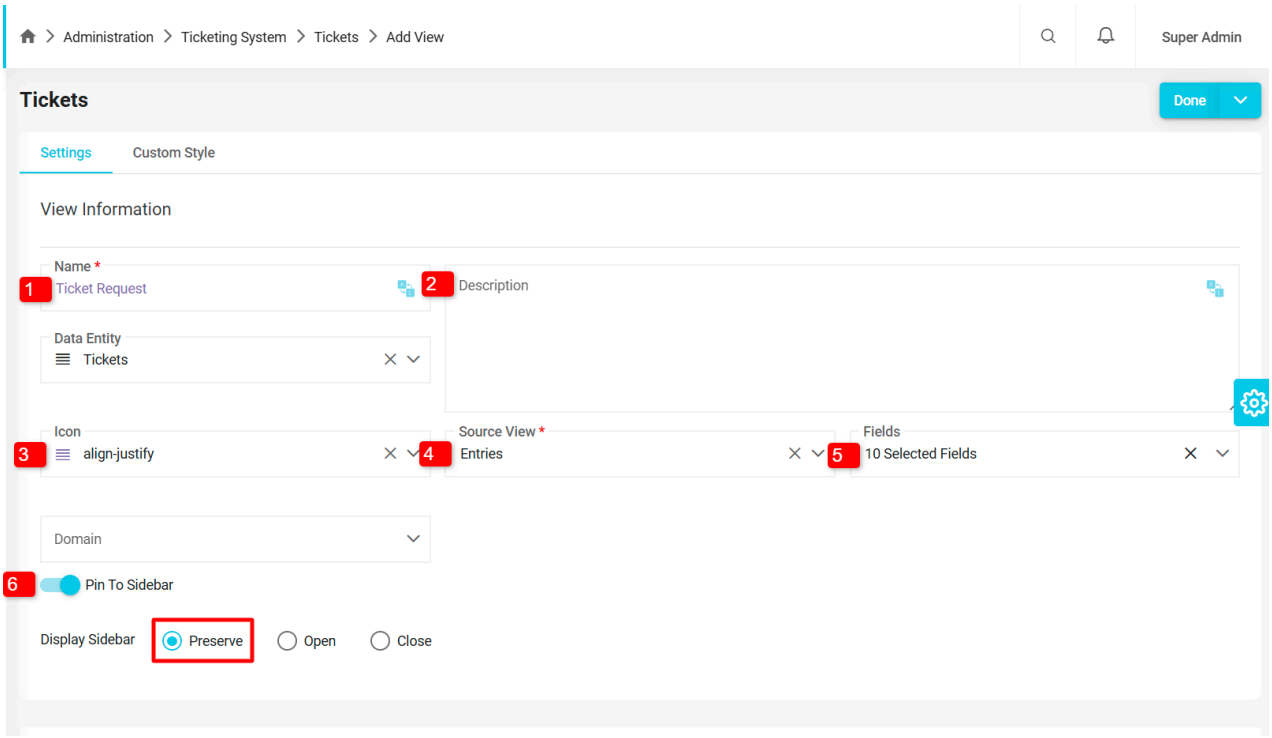
1. Open the *Support Tickets* entity, select the **Views** tab, and select **New View**.



2. Under **View information**, enter the **Name** *Ticket Request* in Arabic and English and pick the **Icon** *megaphone*.

3. Under **Type & source**, set **Source view** to **Entries**.

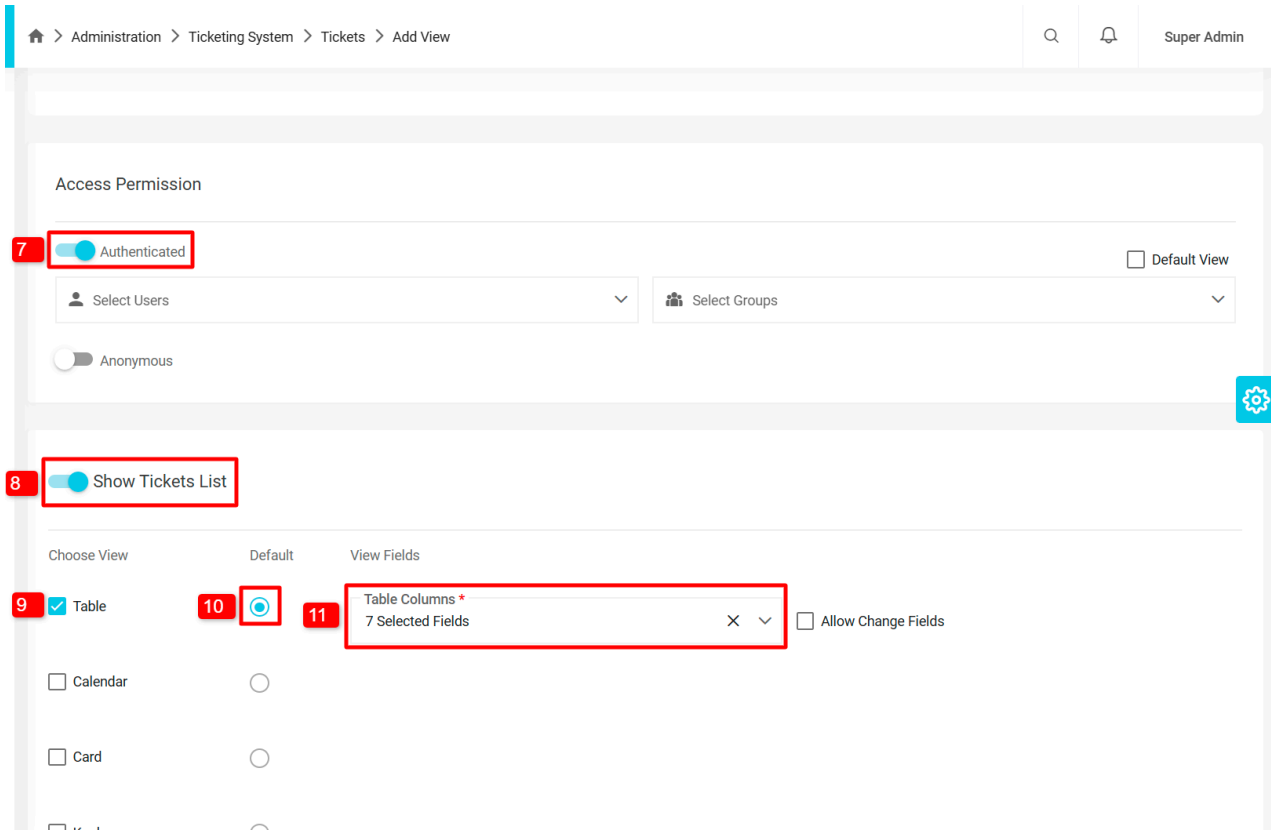
4. Under **Appearance**, turn on **Pin to sidebar**.



5. Under **Access Permission**, turn on **Authenticated** and leave **Select Users** and **Select groups** empty, so every signed-in person can open tickets.

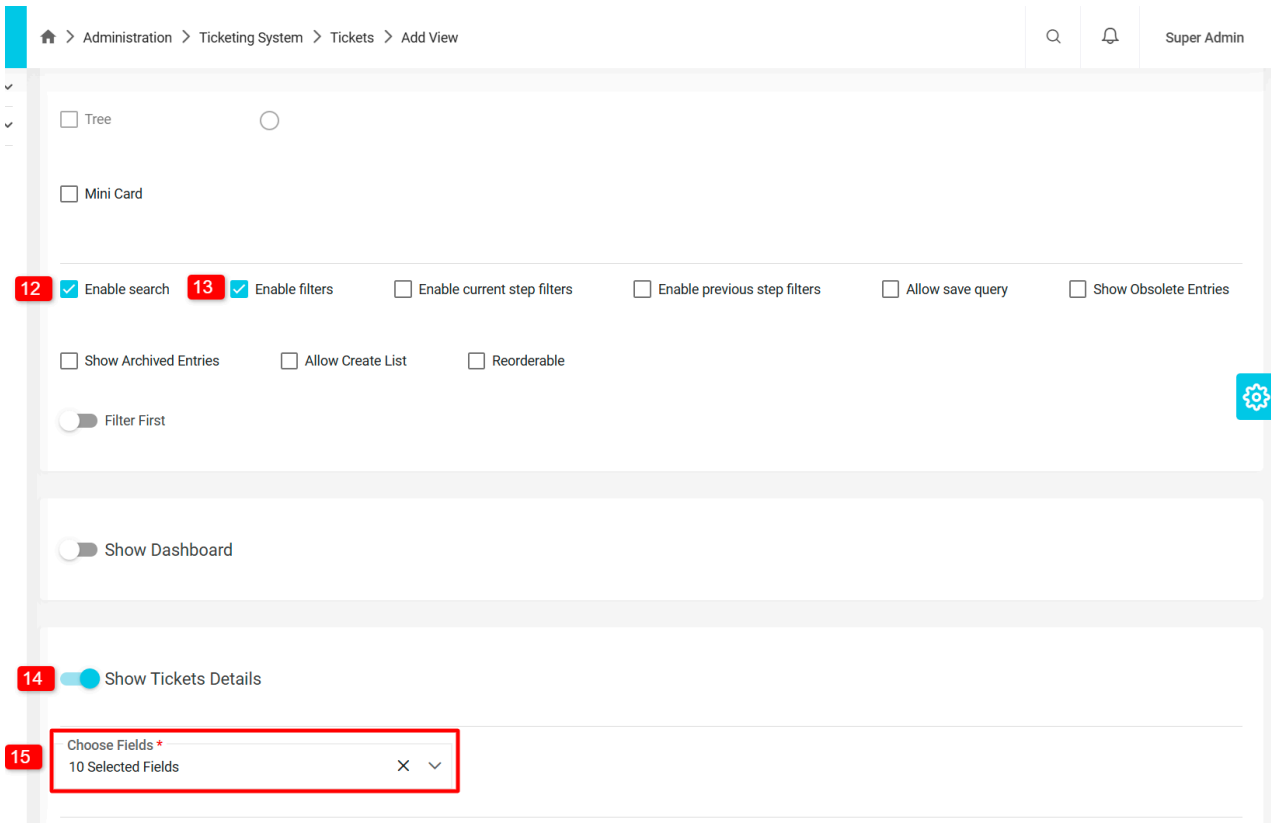
6. Turn on **Entries list**. Under **Displays**, turn on **Table view** with the fields *Ticket number*, *Ticket title*, *Ticket description*, *Start date* and *Service evaluation*, and make it the **Default**. Turn on **Search & filter**.

   Every field list in the view designer shows a block as its inner fields: for *Service evaluation*, choose *Evaluation score* and *Comment*; for *Solution*, choose *Solution*. This applies to all the field choices on this page.



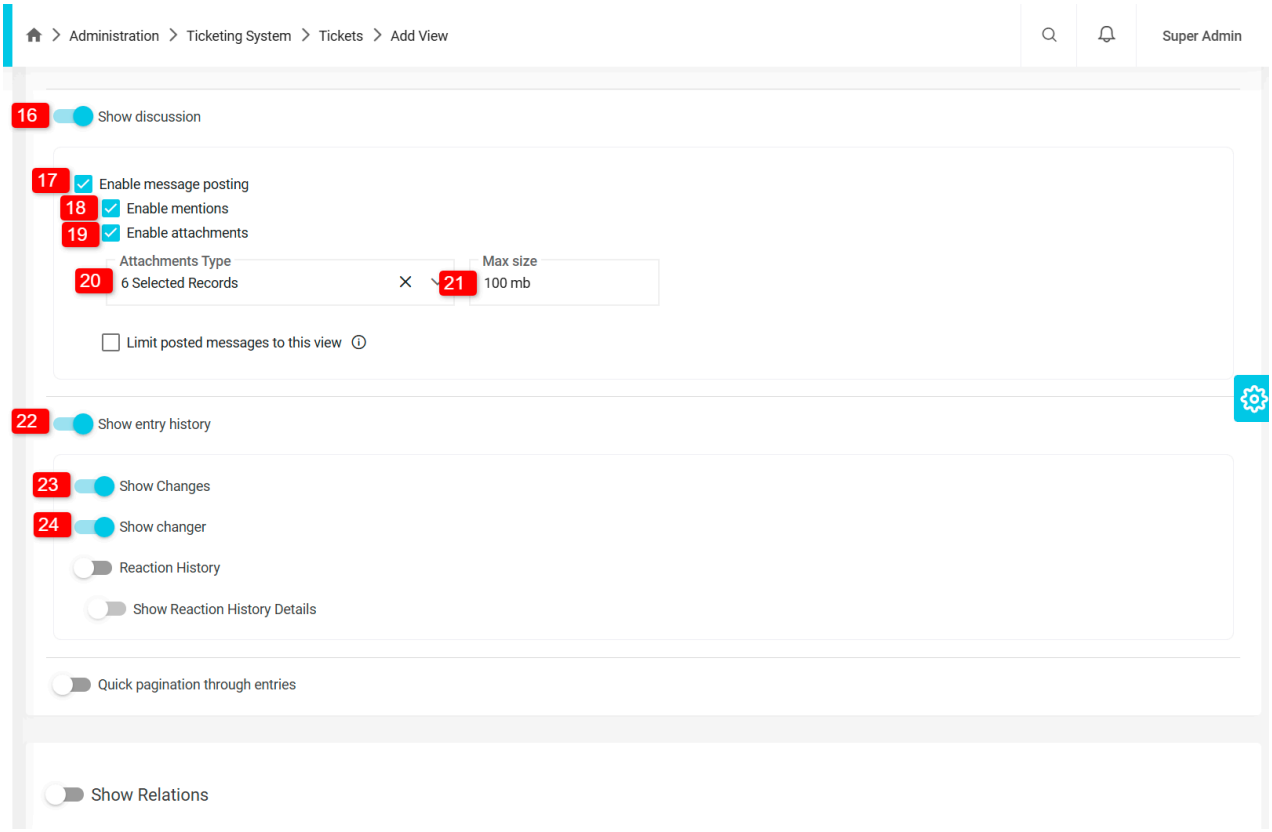
7. Leave **Dashboard** off.

8. Turn on **Entries details** and choose the same fields.



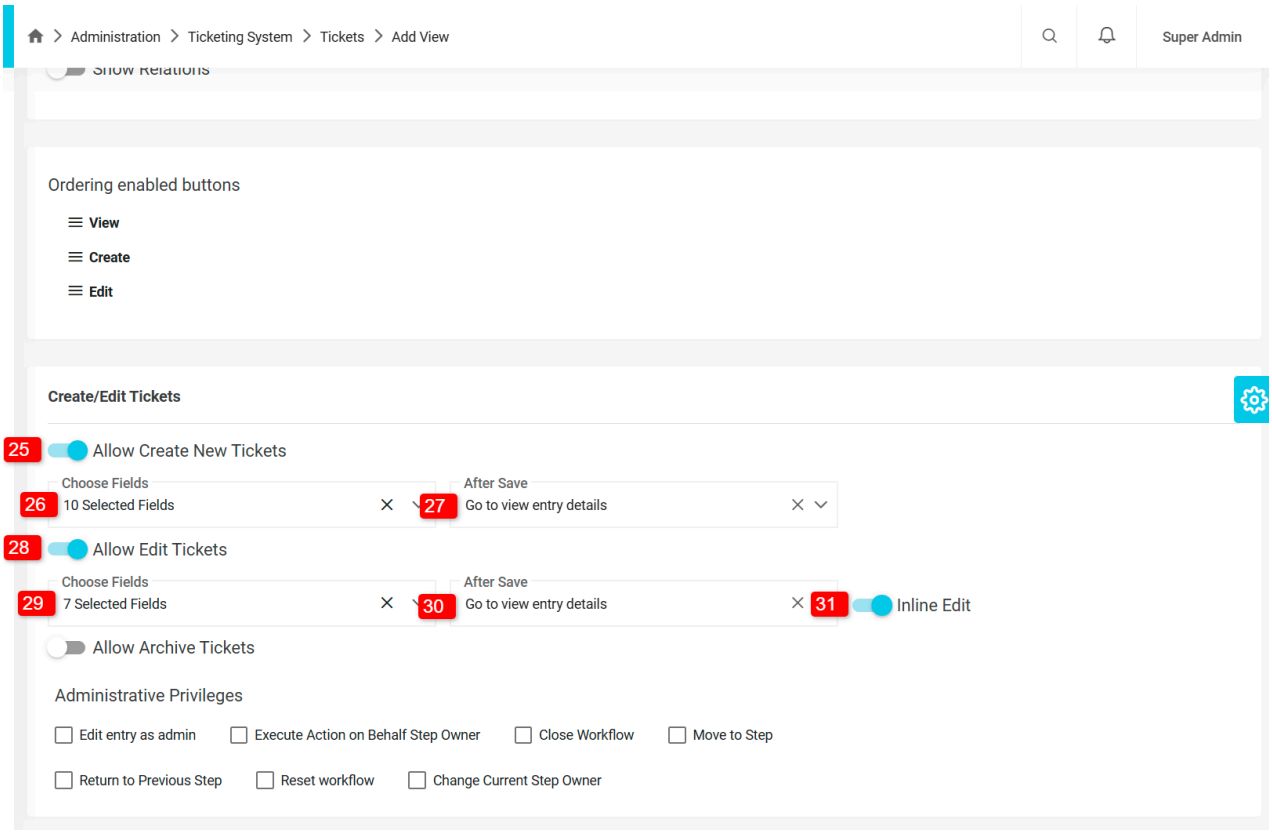
9. Turn on **Discussion** with **Workflow comments** (so each step's note appears there), **Message posting** and **Mentions**. Leave **Attachments** off: turning it on makes the allowed file extensions and a maximum size

required, and **Save** stays unavailable until they are filled in. Turn on **Entry history** with **Changes** and **Changer**, so people can see what changed and who changed it, and **Workflow path**, so people can follow a ticket's route on its **WorkFlow** tab.

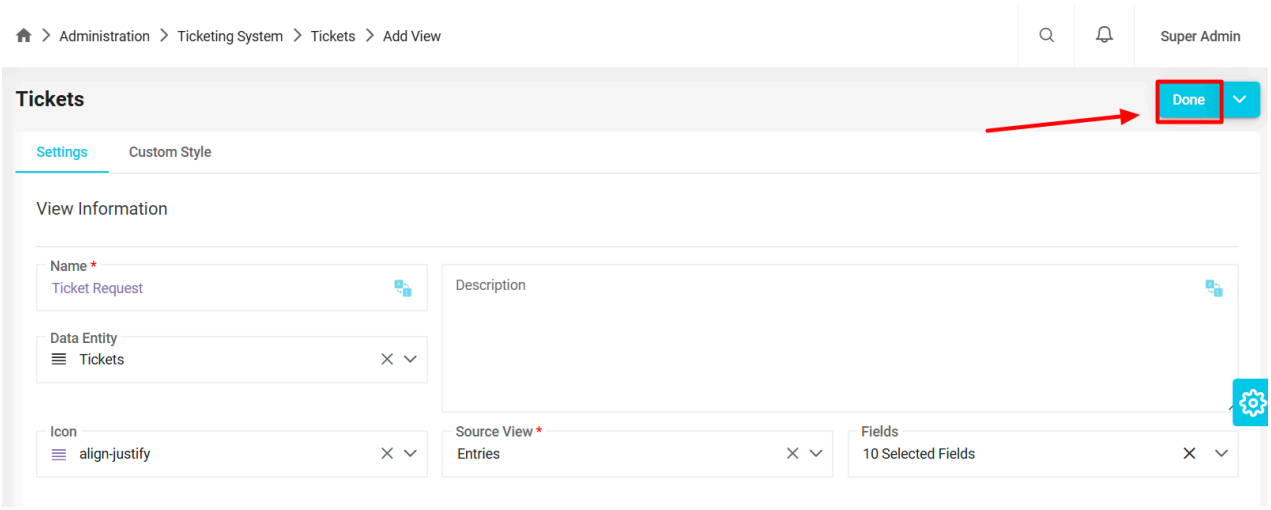


10. Leave **Relations** off.

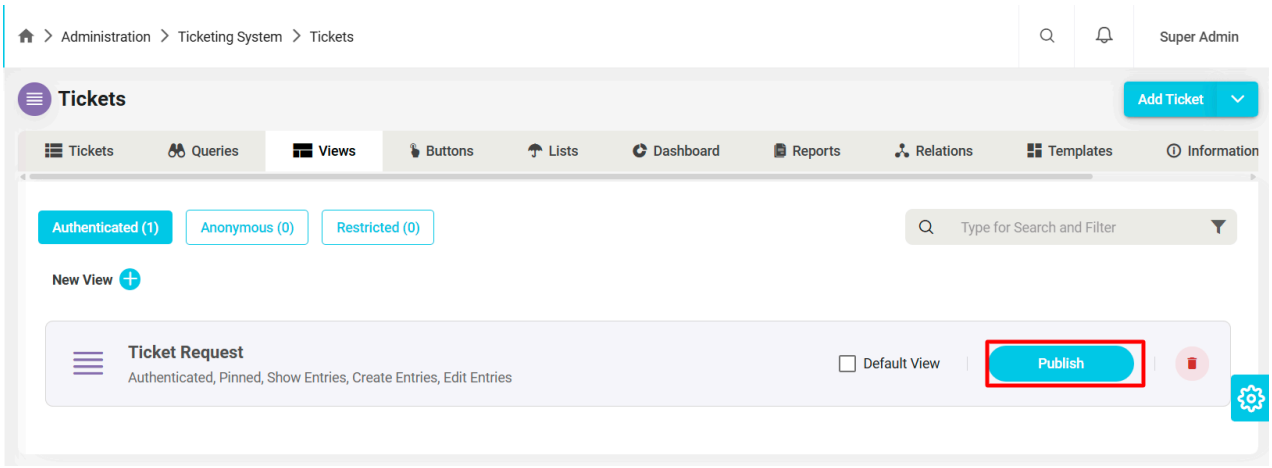
11. Turn on **Creating new entries** with the same fields plus *Ticket status*. The add form must hold every required field; otherwise the view is not saved, and the screen shows no clear error. Set **After Save** to **Go to view entry details** (offered because **Entries details** is on), so the ticket's details open after saving. Turn on **Editing entries** with the same fields.



12. Select **Save**.



13. On the **Views** tab, select **Publish** on the *Ticket Request* card.

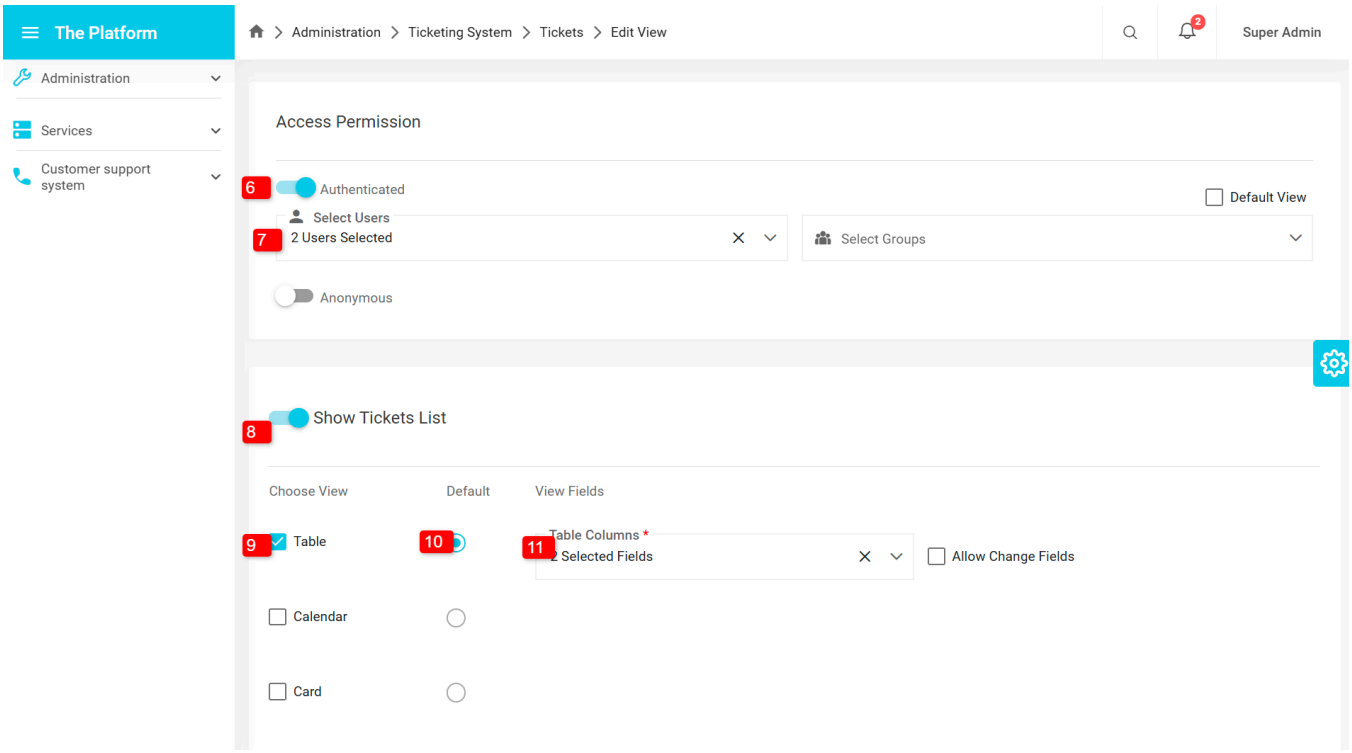


2. Build “Incoming Requests”

Repeat the steps with these differences:

Setting	Value
Name	*Incoming Requests*
Access Permission	**Authenticated**, with *Desk Agent*, *Representative One* and *Support Manager* in **Select Users** (or the *Technical Support Office* group in **Select groups**)
Table view fields	Add *Ticket status* and *Assigned to*
Editing entries fields	Add *Ticket status*, *Assigned to*, *End date* and *Solution* (the field inside the *Solution* block)
Creating new entries	Off — tickets are opened from *Ticket Request*

The step rules from the workflow still decide which of these fields each person can change at each step.



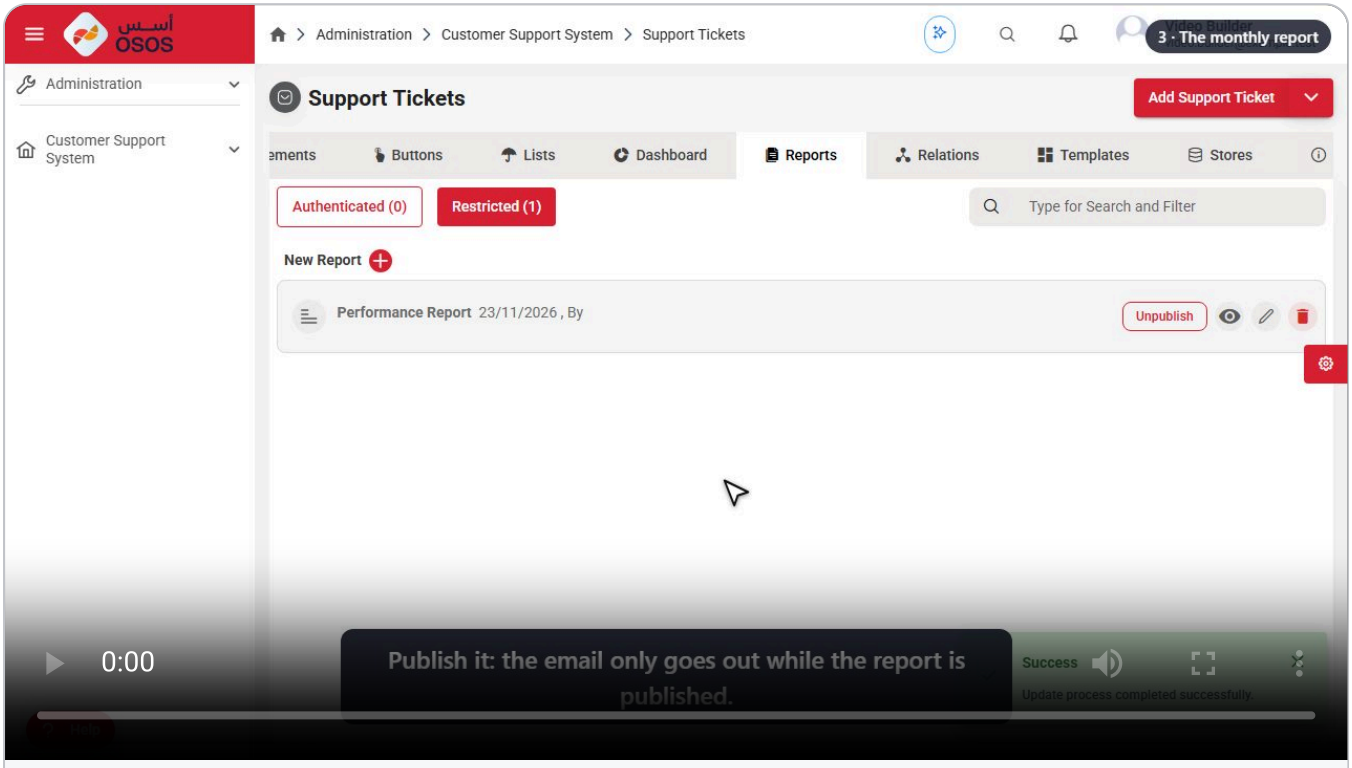
Select **Save**, then **Publish** the view. You will add the status chart to it in the next step.

More view options are described in Data views.

Next: Add the status chart.

Add the status chart

A **summary** turns tickets into a number or a chart. You create *Ticket Status*, a doughnut chart that counts tickets by status, and show it to the support team in *Incoming Requests*.

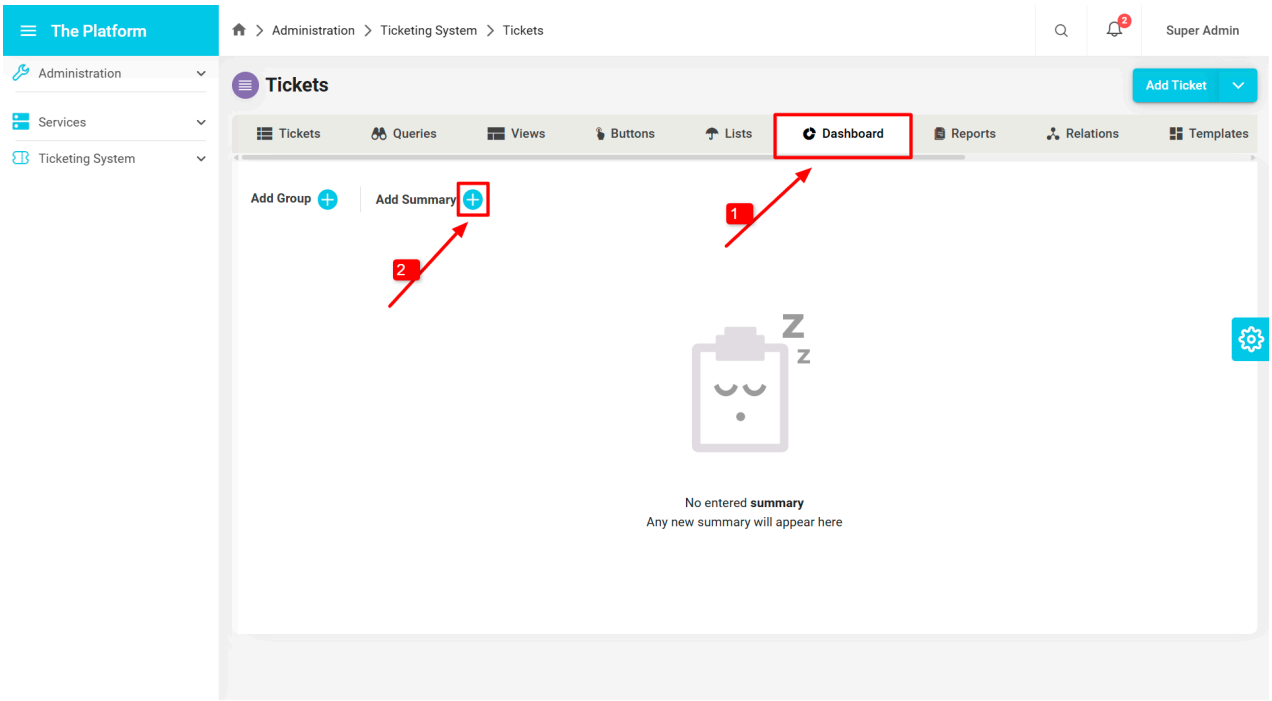


TUTORIAL · PART 6 OF 8 **The status chart and the monthly report**

A chart of tickets by status for the desk, and a PDF report for the manager · 2:51 · Recorded on OSOS 1.11.0-alpha.1

Create the summary

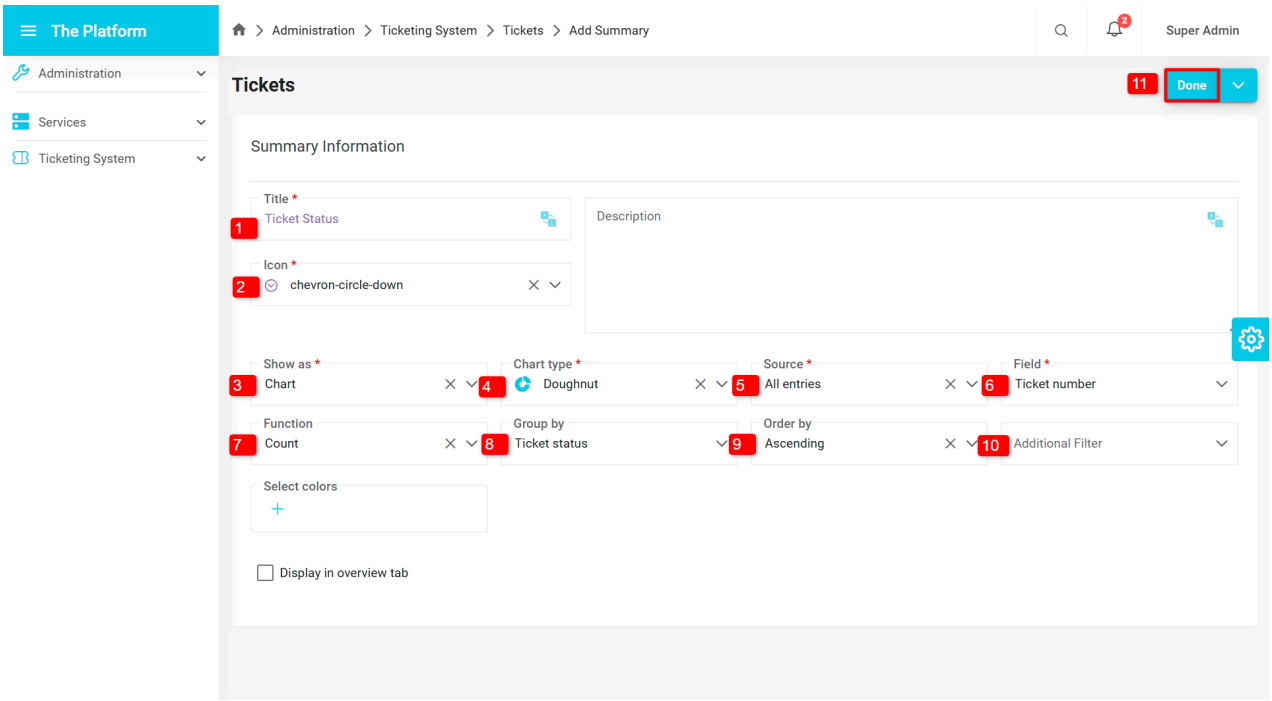
1. Open the *Support Tickets* entity, select the **Dashboard** tab, and select **Add Summary**.



2. The **Summary Information** form opens. Fill in:

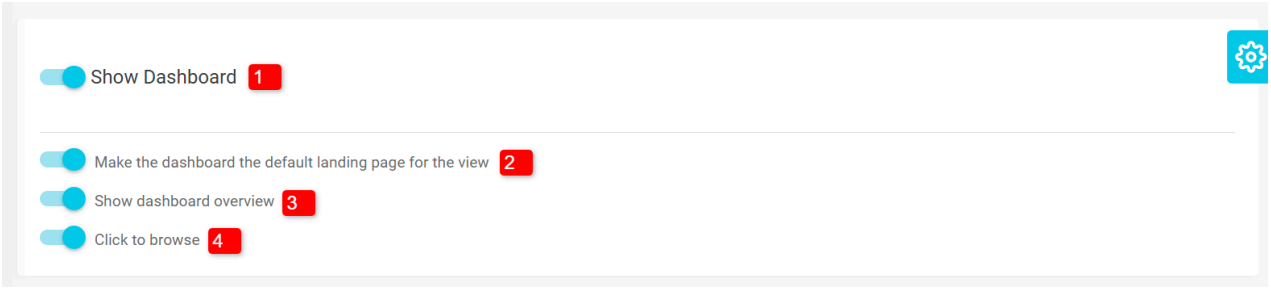
    - **Title** — *Ticket Status*, in Arabic and English.
    - **Icon** — *chevron-down*.
    - **Show as** — **Chart**.
    - **Chart type** — **Doughnut**.
    - **Source** — **All entries**.
    - **Field** — *Ticket number*.
    - **Function** — **Count**.
    - **Group by** — *Ticket status*.
    - **Order by** — **Ascending**.

3. Select **Done**, and confirm with **Save Changes** if asked.



Show it in “Incoming Requests”

1. On the **Views** tab, open *Incoming Requests* for editing.
2. Turn on **Dashboard**, then turn on **Dashboard overview** and **Click to browse**.
3. Under **Behavior** › **Default landing screen**, drag **Dashboard** to the top of the list, so the view opens on the chart.
4. Select **Save**.



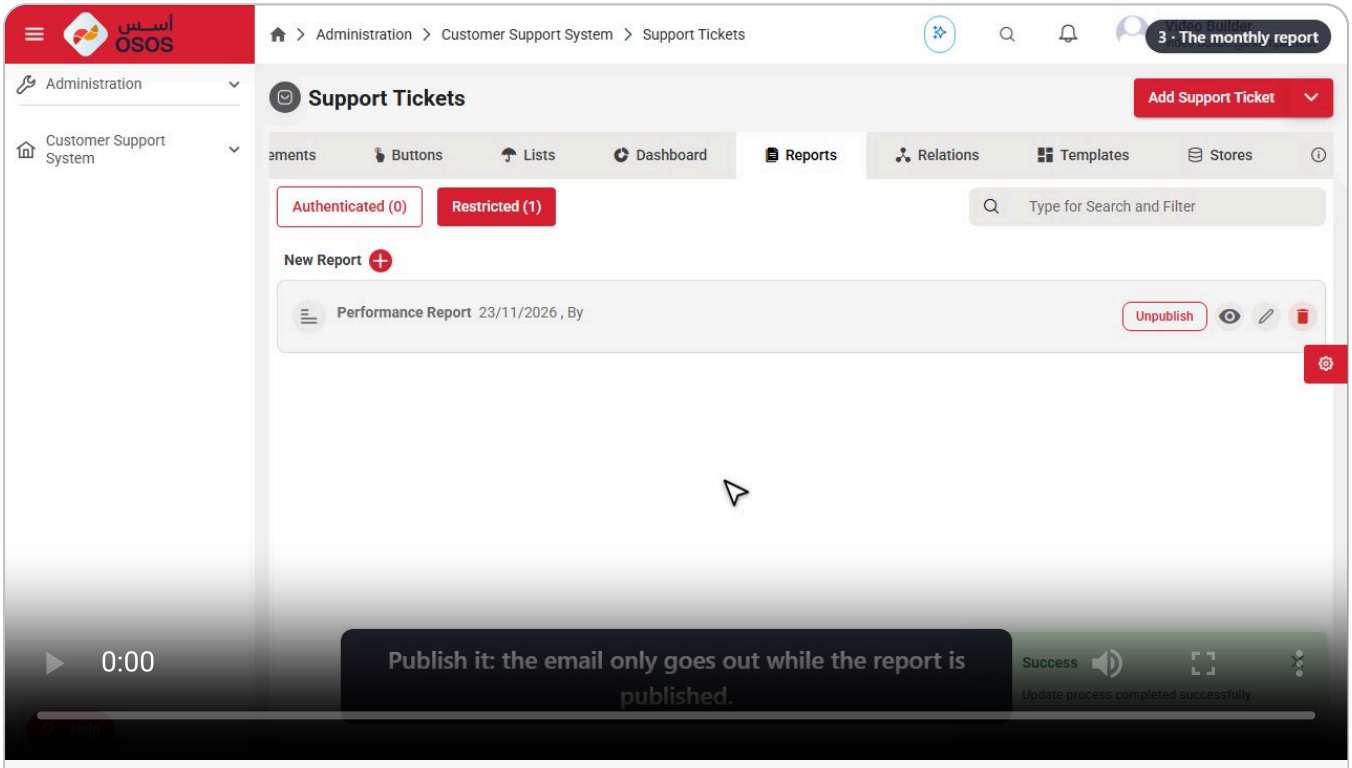
The team now lands on the chart when they open *Incoming Requests*, and can select a segment — say **In Progress** — to see those tickets.

More about summaries and dashboards is in Summaries, dashboards & counters.

Next: Create the monthly report.

Create the monthly report

The *Performance Report* lists every ticket with its status, so the manager can see how many tickets are open and closed. It is emailed as a PDF on the 28th of each month.

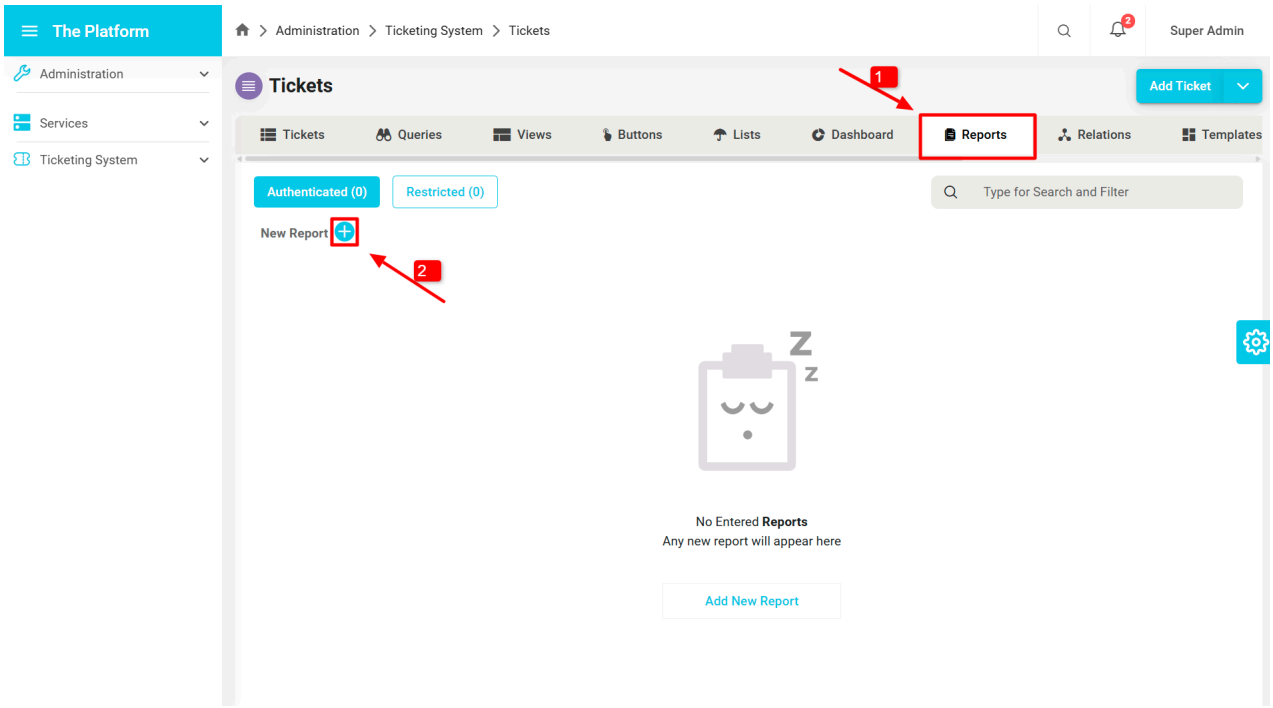


TUTORIAL · PART 6 OF 8 **The status chart and the monthly report**

A chart of tickets by status for the desk, and a PDF report for the manager · 2:51 · Recorded on OSOS 1.11.0-alpha.1

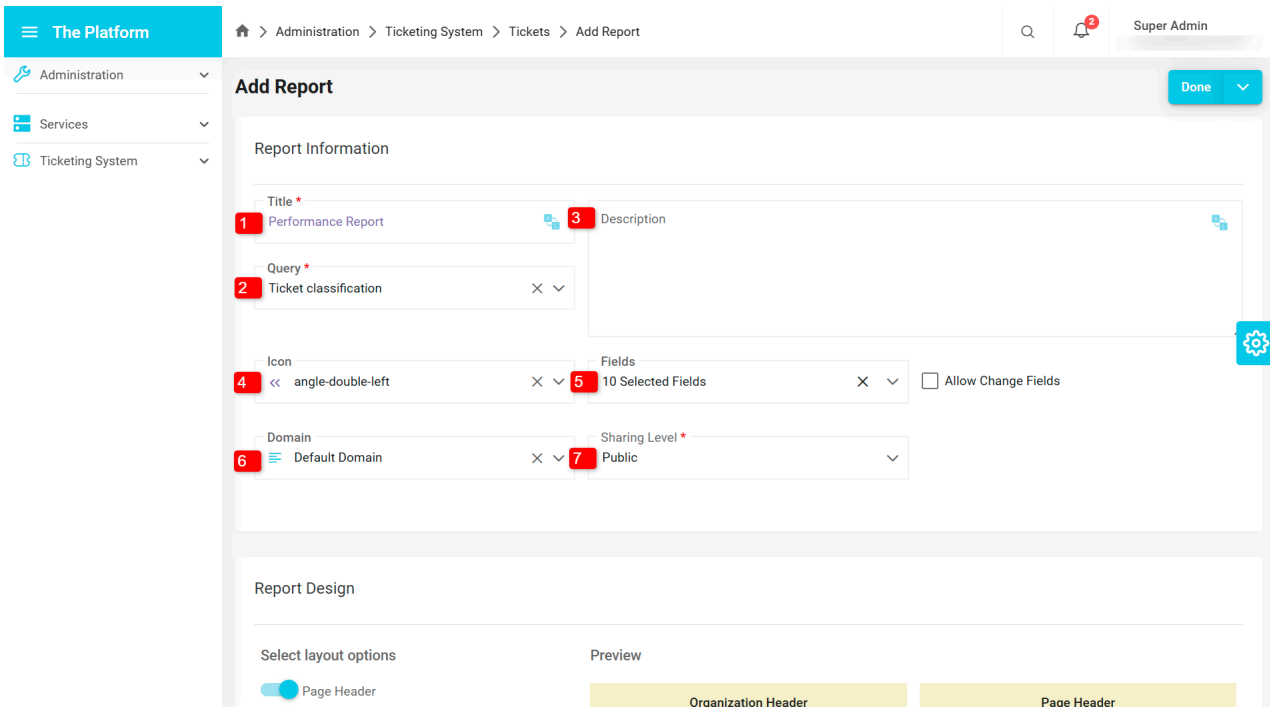
Create the report

1. Open the *Support Tickets* entity, select the **Reports** tab, and select **New Report**.

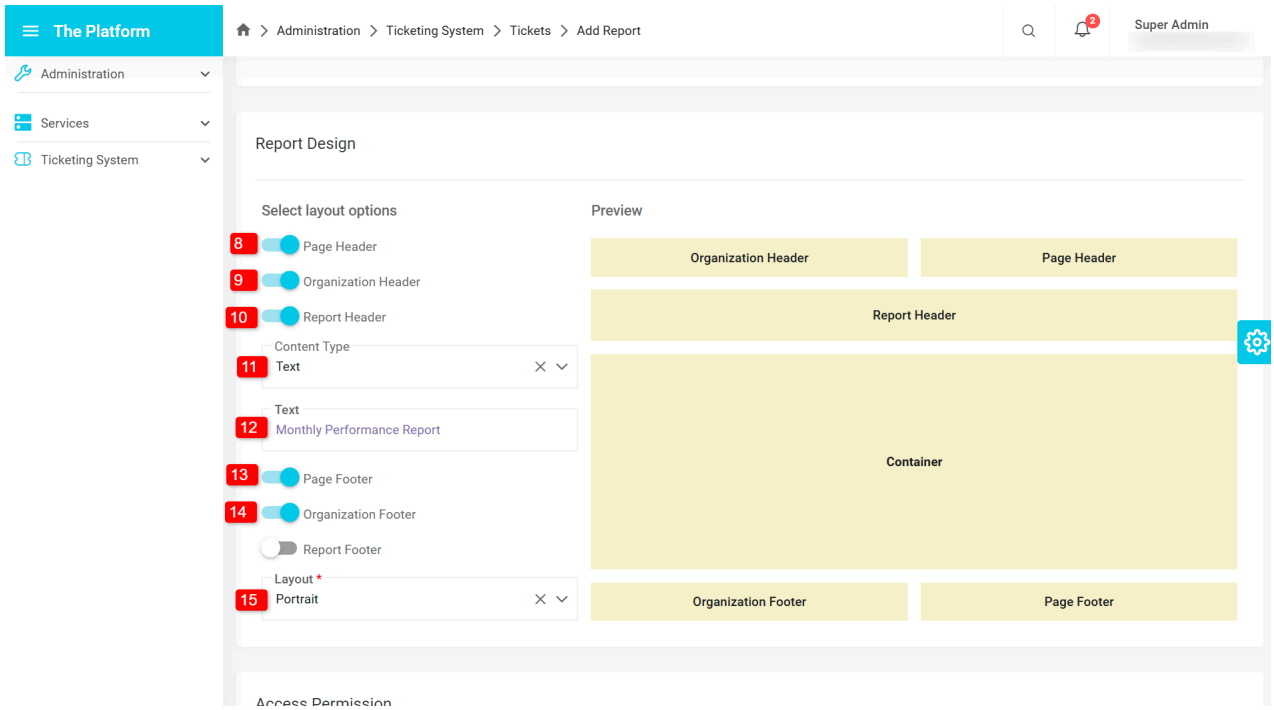


2. Fill in the report information:

   - **Title** — *Performance Report*.
   - **Query** — *All tickets*, from the queries step.
   - **Icon** — *align-right*.
   - **Fields** — *Ticket number*, *Ticket title*, *Ticket status*, *Assigned to*, *Start date*, *End date*. A PDF shows at most seven columns, so keep the list short.



3. Under **Report Design**, turn on **Page header**, **Organization Header**, **Report Header** (choose **Text** and enter *Monthly Performance Report*), **Page Footer** and **Organization Footer**. Set the **Layout** to **Landscape**.

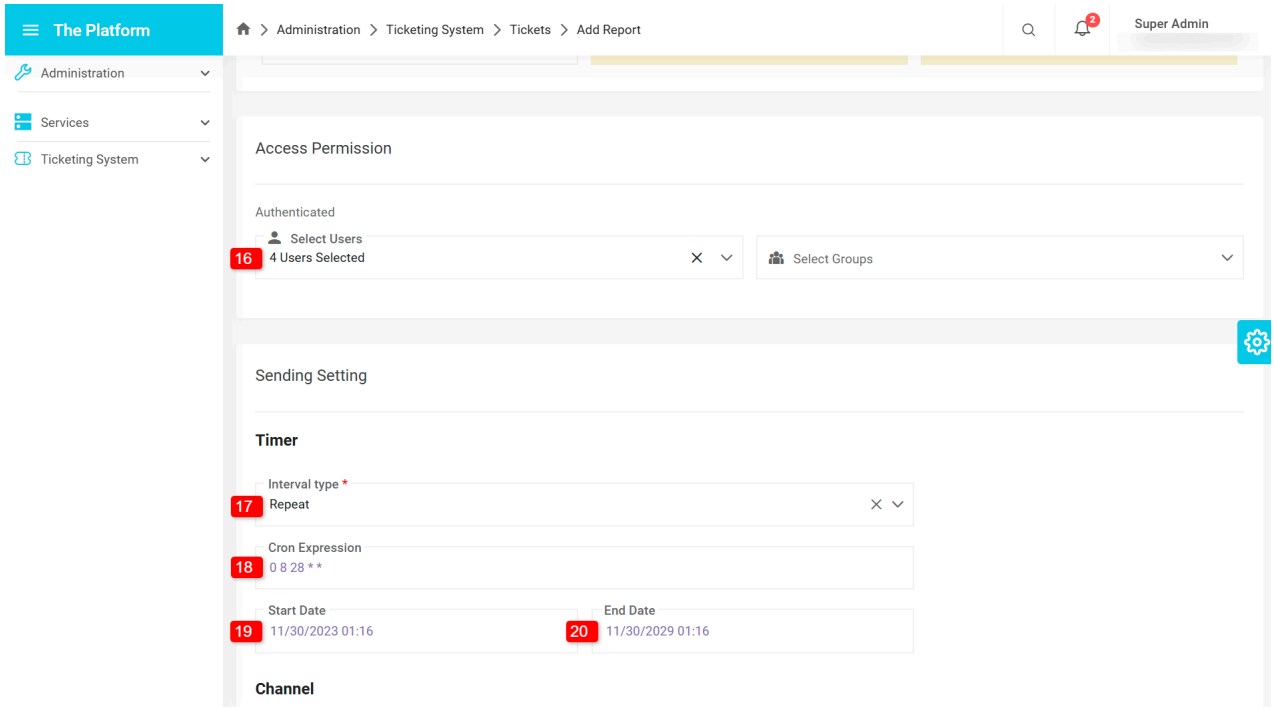


4. Under **Access Permission**, choose *Support Manager* and the *Technical Support Office* group.

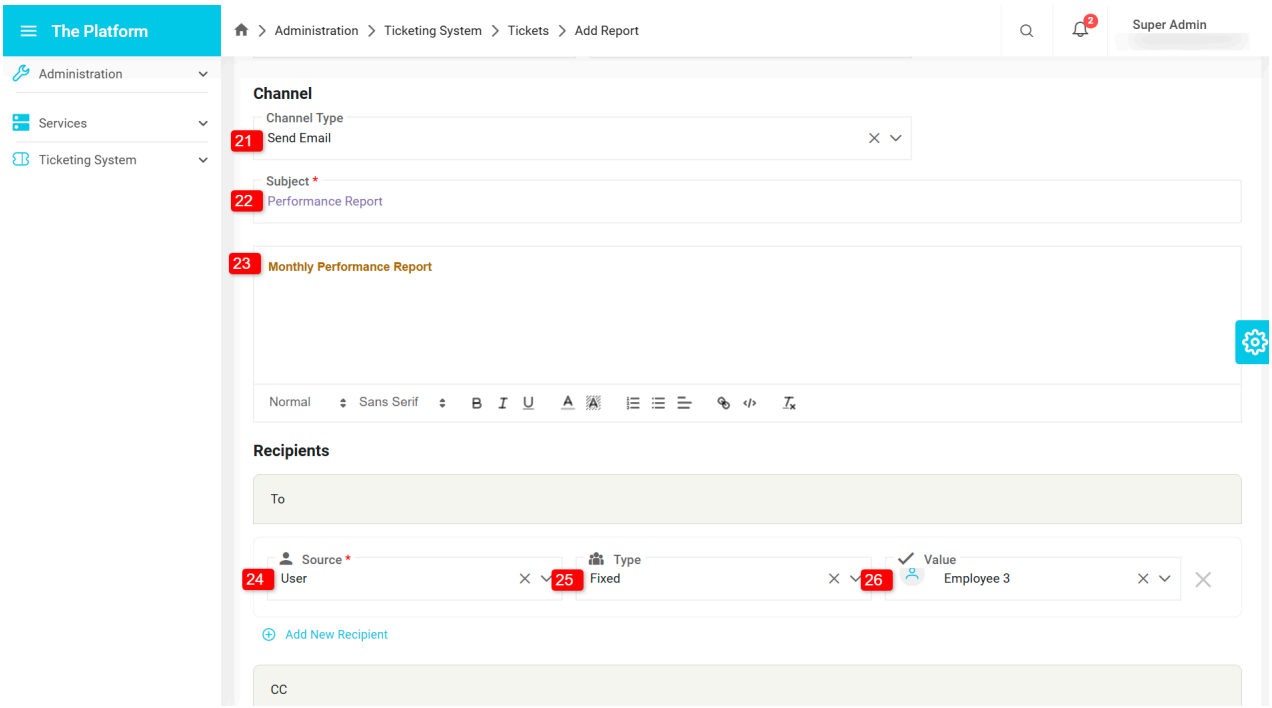
Schedule the email

Under **Sending Setting**:

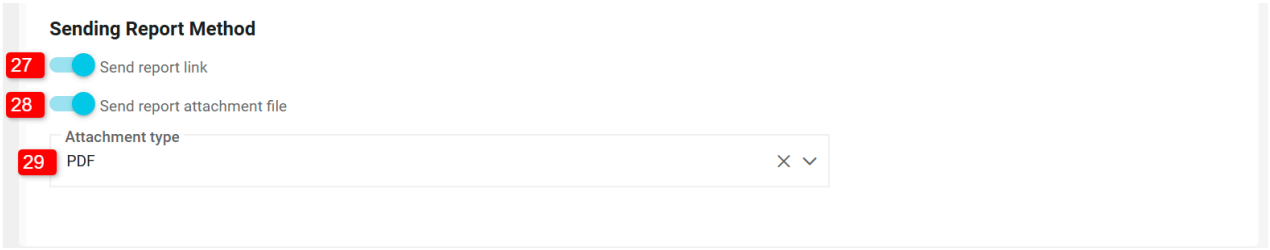
1. **Timer** — **Interval type Repeat**, **Cron Expression** `0 8 28 * *` (08:00 on the 28th of every month, a few days before the month ends). Set a **Start Date**, and an **End Date** if the report should stop.



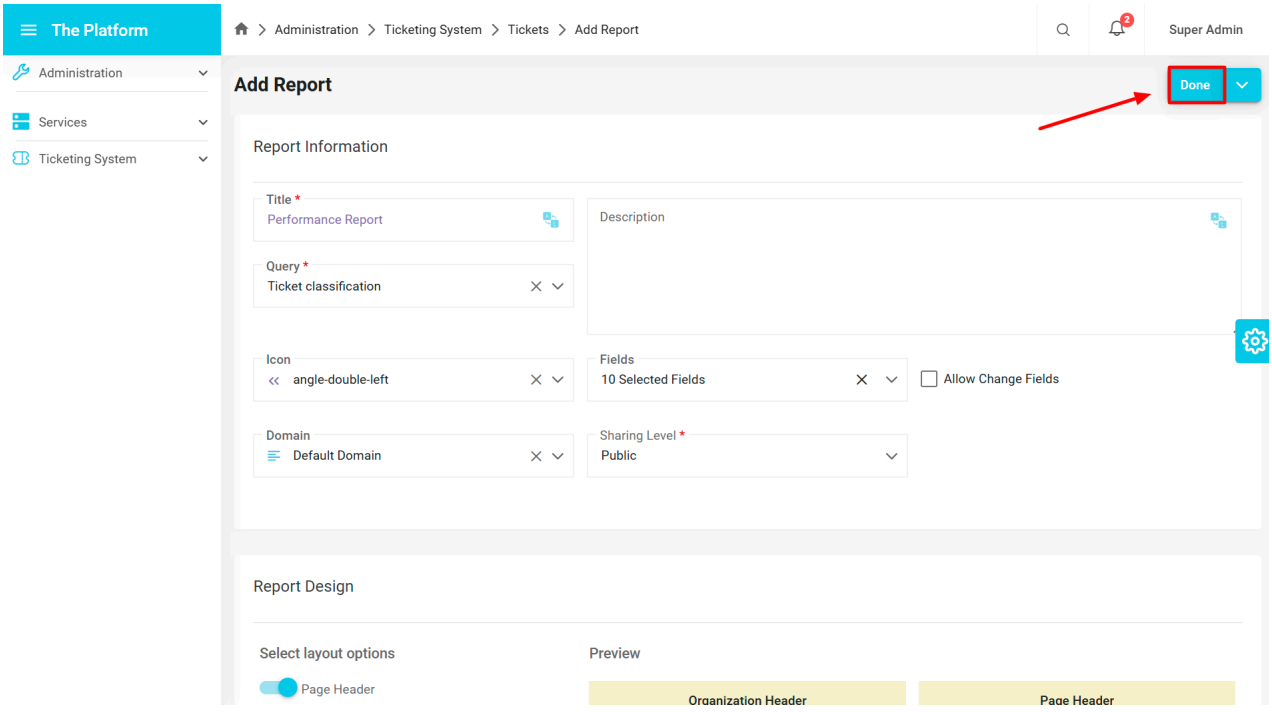
2. **Channel** — **Channel Type Send Email**. In **To**, choose **User**, **Fixed**, *Support Manager*. **Subject** *Performance Report*; **Body** *Monthly Performance Report*.



3. **Sending Report Method** — tick **Send report link**, and tick **Send report attachment file** with **Attachment type PDF**.

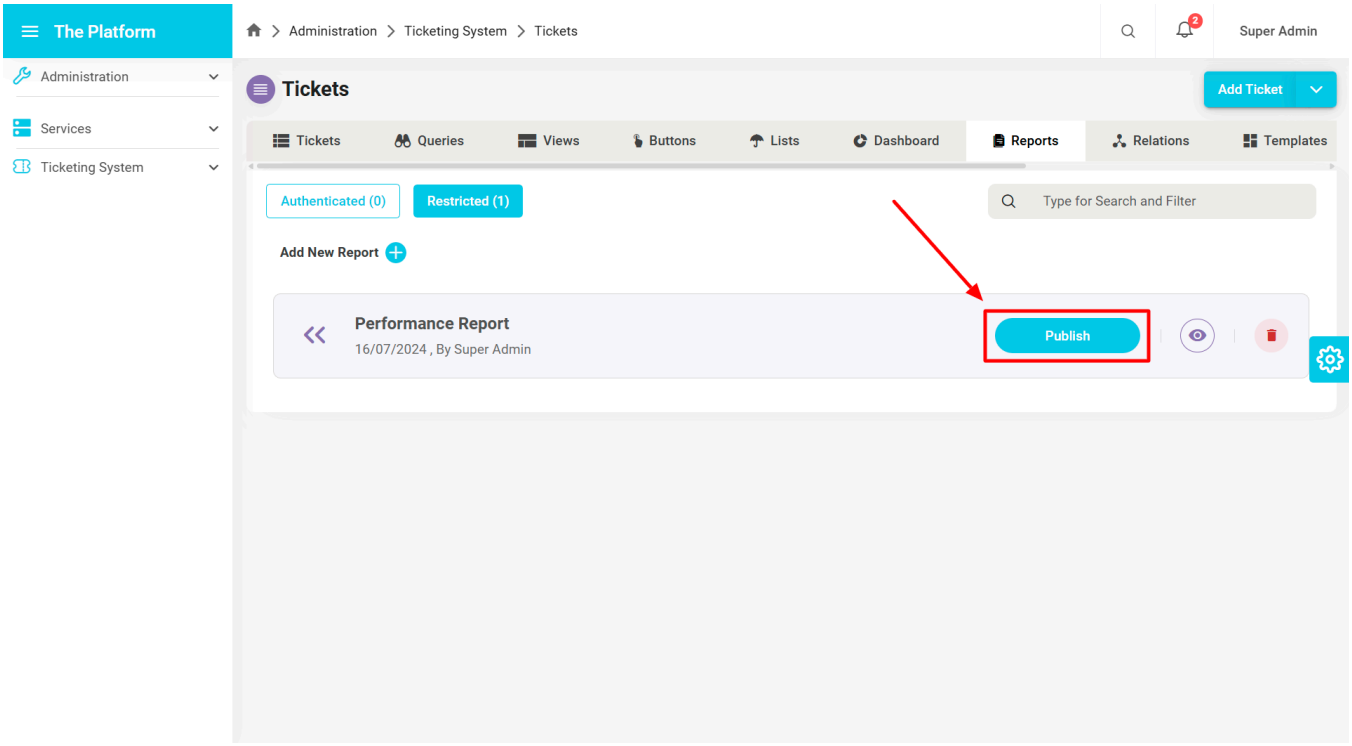


4. Select **Done**.



Publish it

On the **Reports** tab, select **Publish** on the report's card and confirm. The email is only sent while the report is published.



How cron expressions work, with common examples, is shown in Send a weekly report. All report options are in Reports.

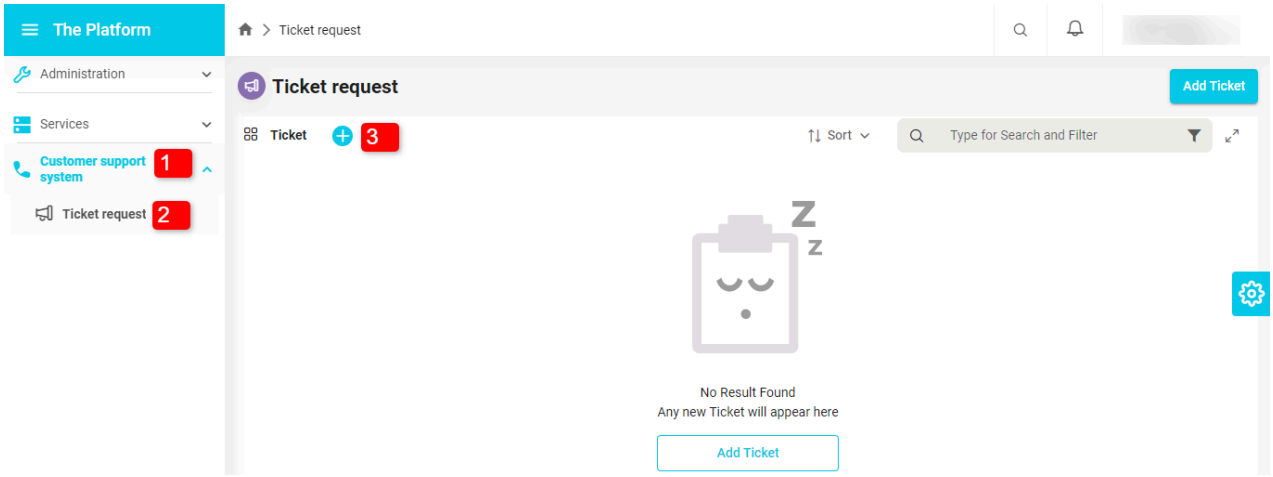
Next: Test the system.

Test the system

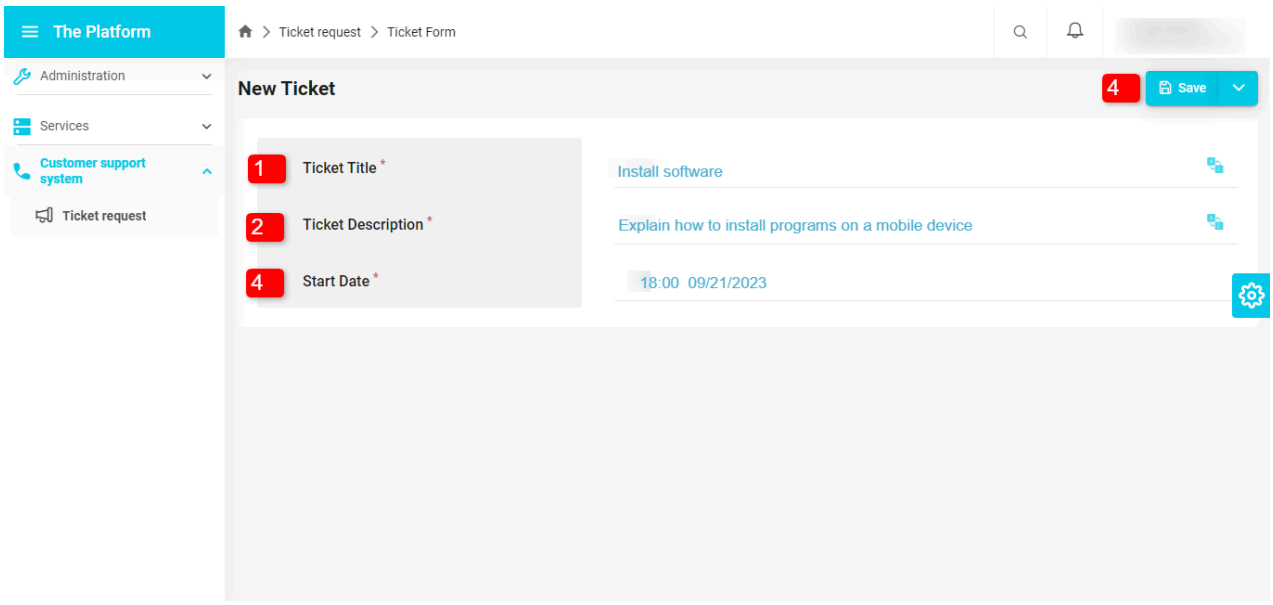
Walk one ticket through the whole workflow, signing in as each test account in turn. At each step, check that the right person is notified and sees the right buttons. A private browser window makes switching accounts easy.

1. Open a ticket — as Customer One

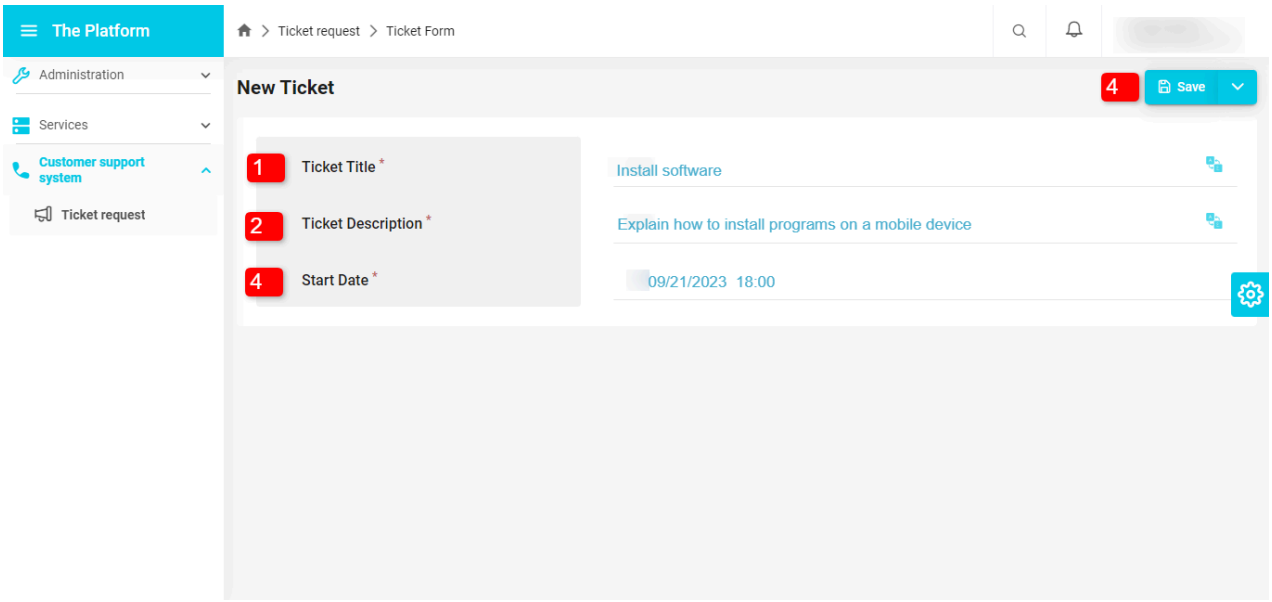
1. In the sidebar, open *Customer Support System* › **Ticket Request**, and select **+** to add a ticket.

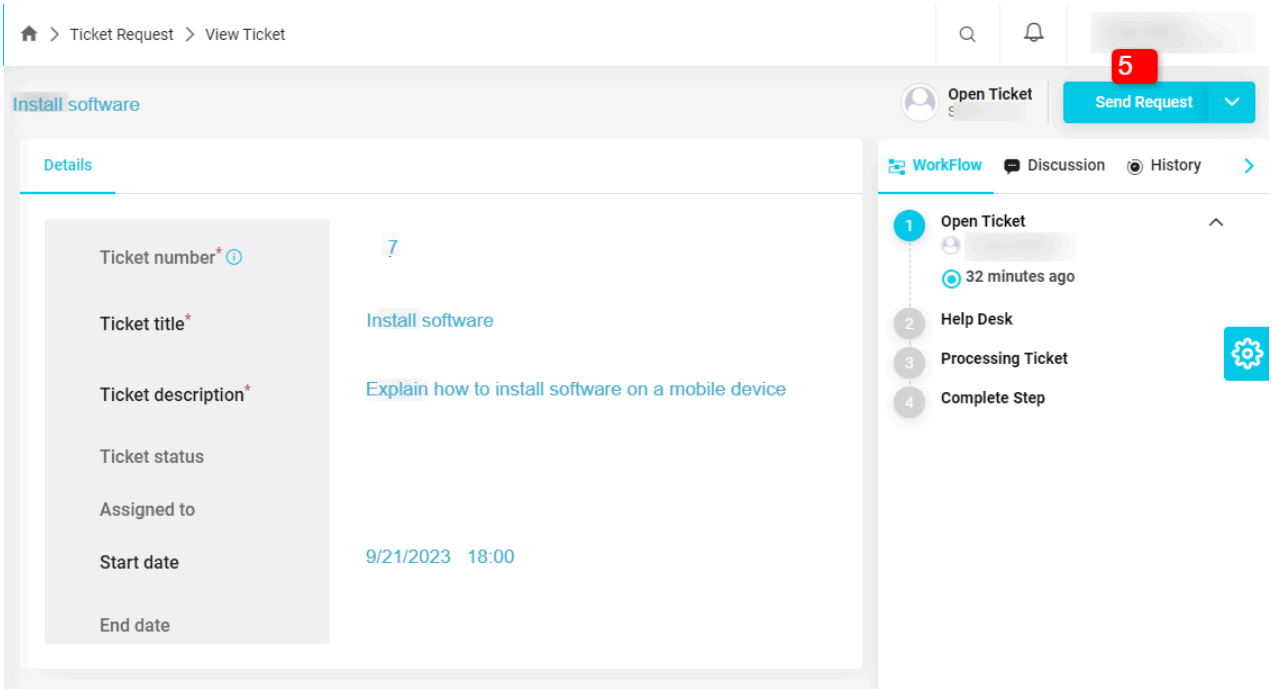


2. Fill in *Ticket title* (*Software installation*), *Ticket description* (*How do I install the software on my phone?*) and *Start date*, then select **Save** and confirm with **Save Changes**.



3. The ticket opens, with its workflow on the side. Select **Send Request**. In the confirmation you can add a note for the desk; confirm with **Send Request**.





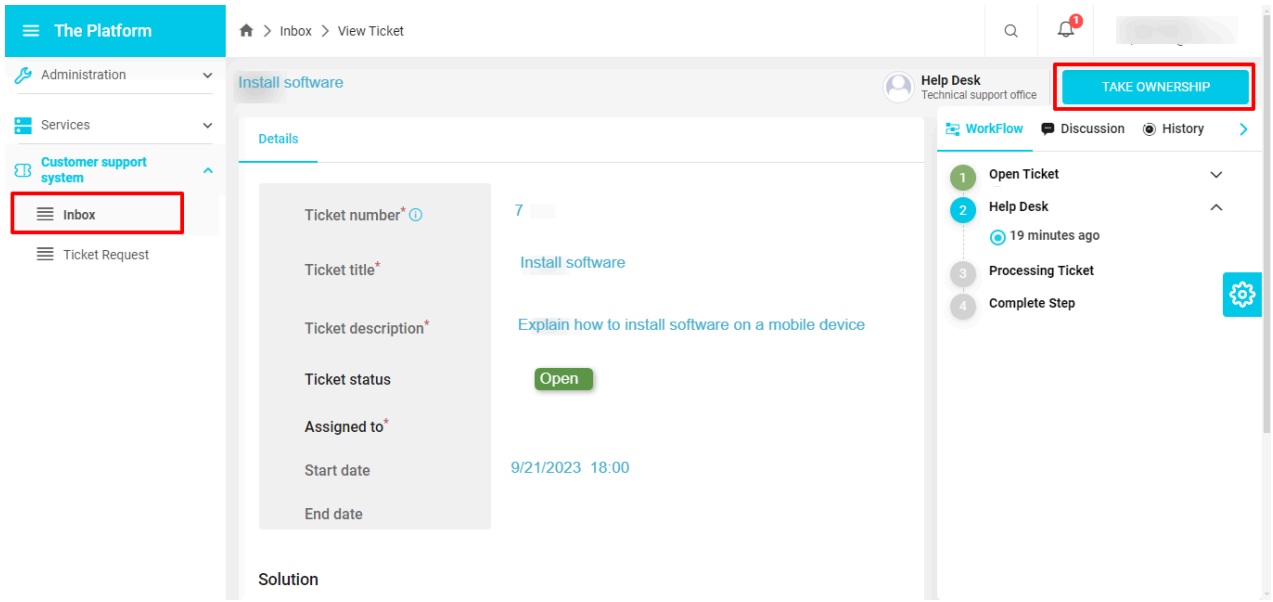
Check: the ticket has a number, its status is **Open**, and it has moved to *Help Desk*.

2. Route the ticket — as Desk Agent

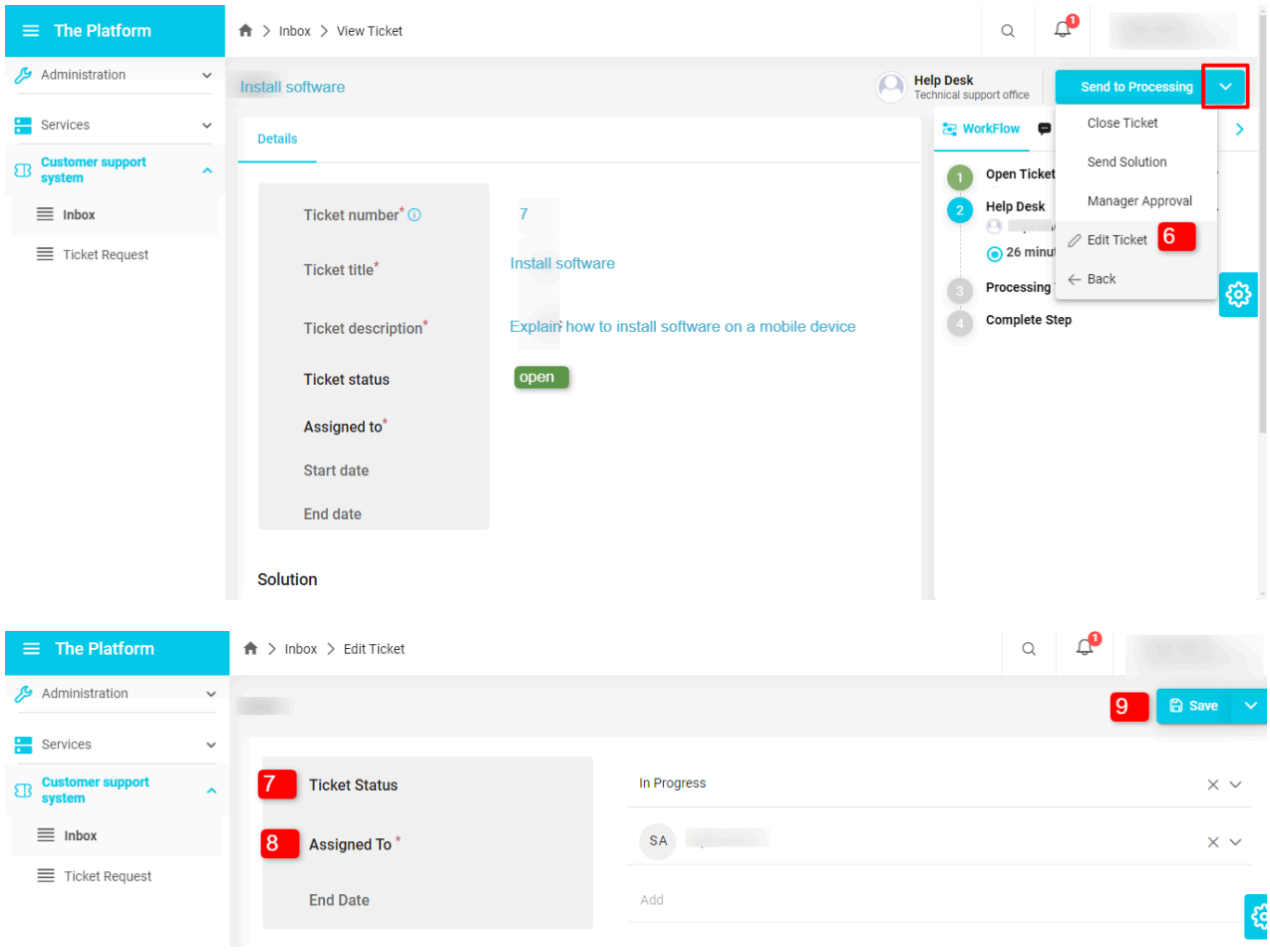
1. The bell shows *New ticket arrived*. Select the notification to open the ticket.



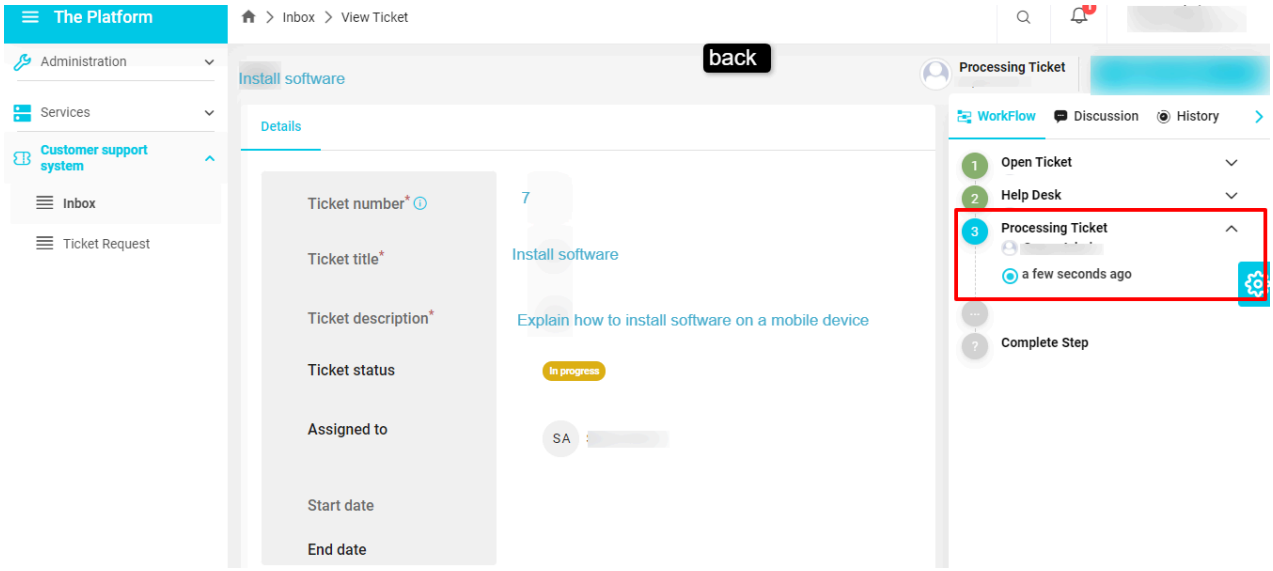
2. Select **Take Ownership** and confirm, so the ticket is yours.



3. Open the menu next to the action buttons and choose to edit the ticket. Set *Ticket status* to **In Progress** and *Assigned to* to *Representative One*, then save and confirm.



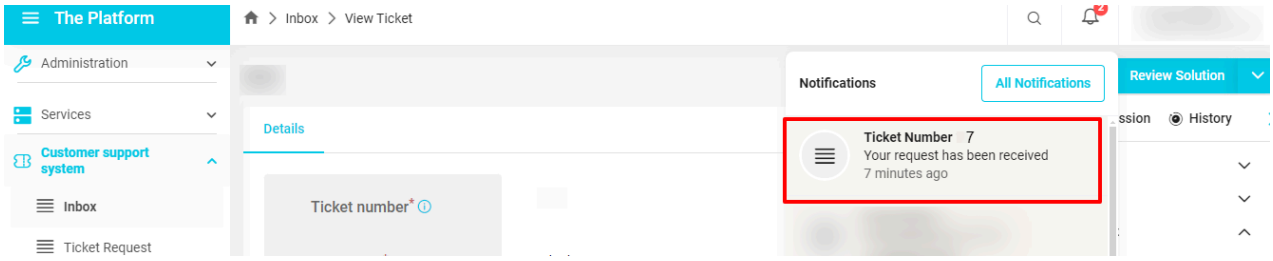
4. Select **Send to Processing** and confirm.



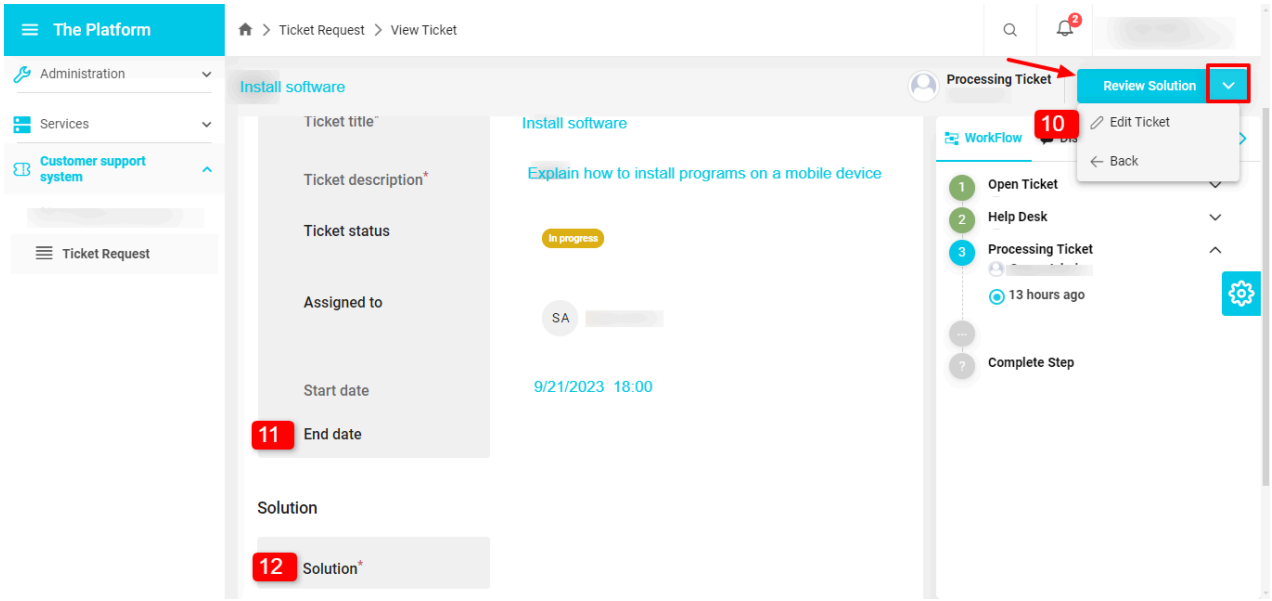
Check: *Customer One* gets *Your request has been received*.

3. Solve it — as Representative One

1. The bell shows *Ticket assigned to you*. Open the ticket (or find it in **Incoming Requests**).



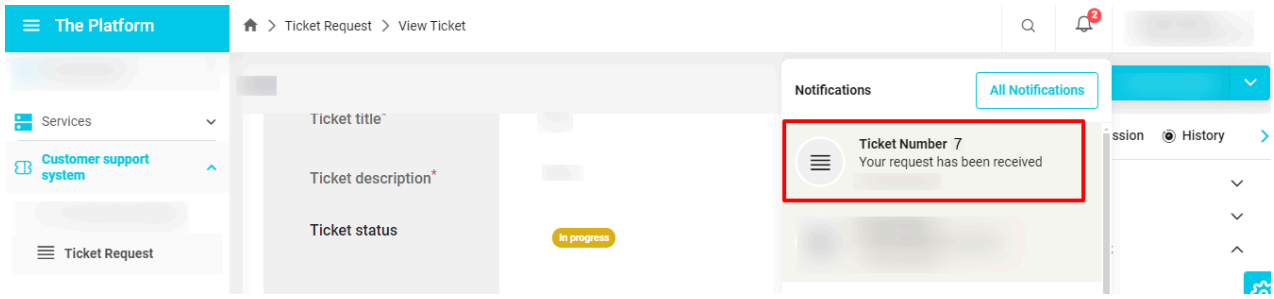
2. Edit the ticket, enter the *Solution*, save, then select **Review Solution**.



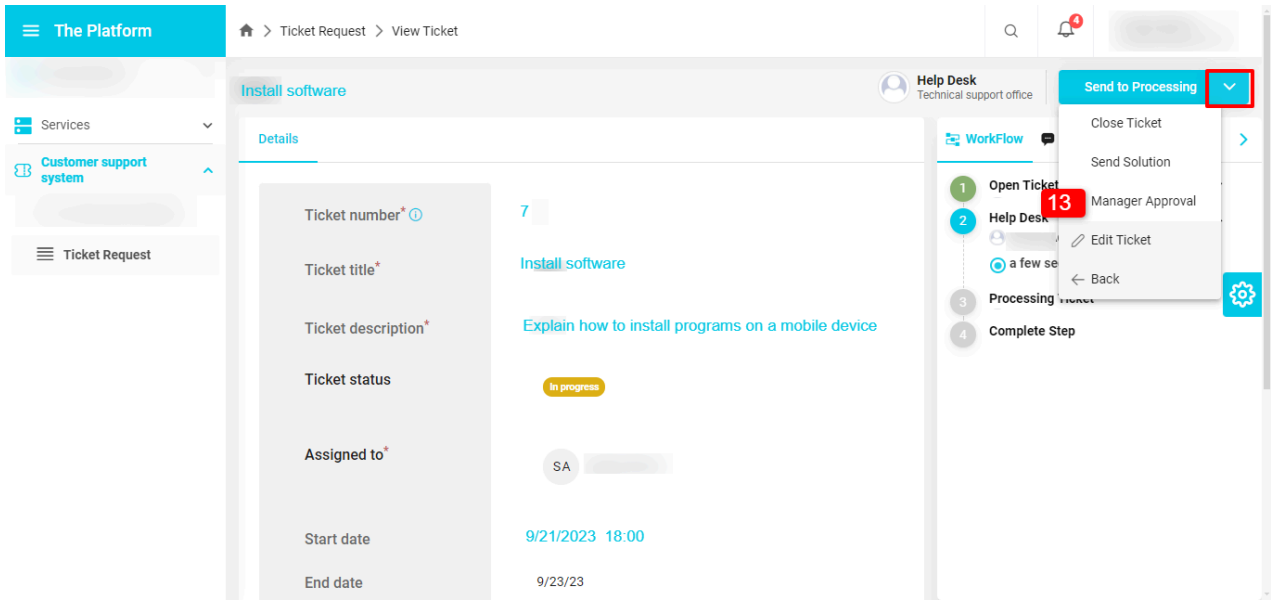
Check: the ticket is back at *Help Desk*, and the desk gets *Please review the solution*.

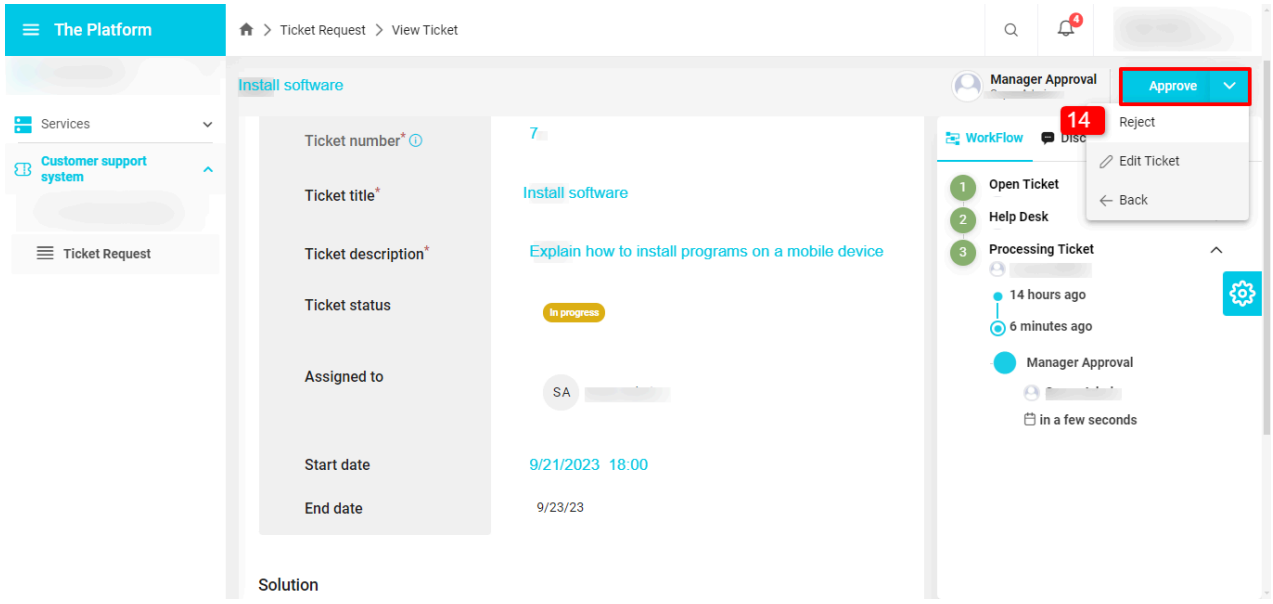
4. Review the answer — as Desk Agent, and Support Manager if needed

1. Open the ticket from the notification and read the solution.

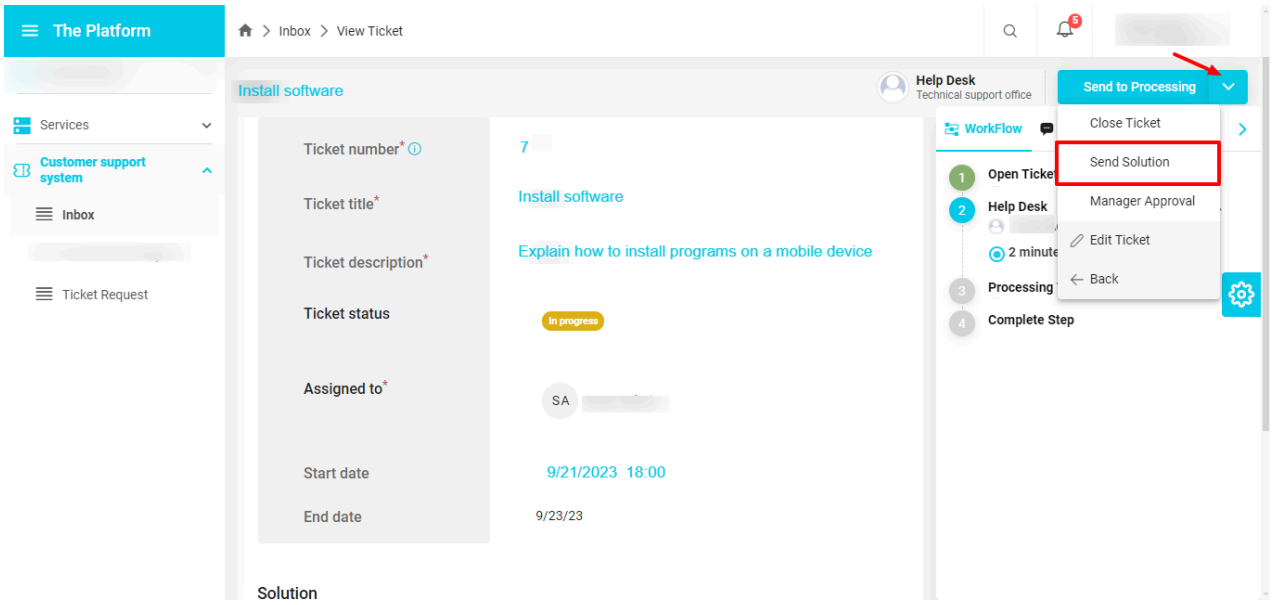


2. If the answer needs a manager's approval, select **Manager Approval**. Then, as *Support Manager*, open the ticket from **Incoming Requests** and select **Approve** (or **Reject**). Either way the ticket returns to *Help Desk*.





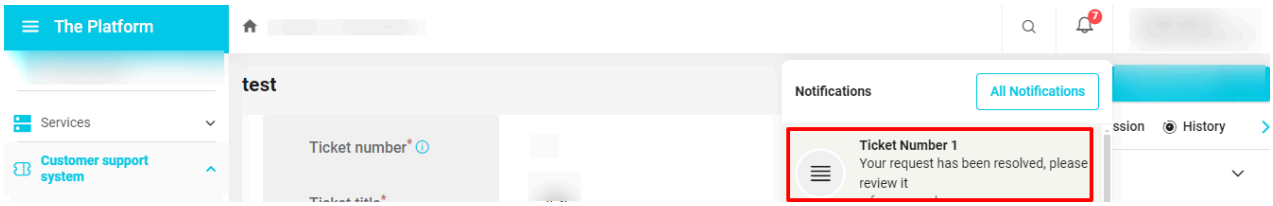
3. As *Desk Agent*, select **Send Solution**.



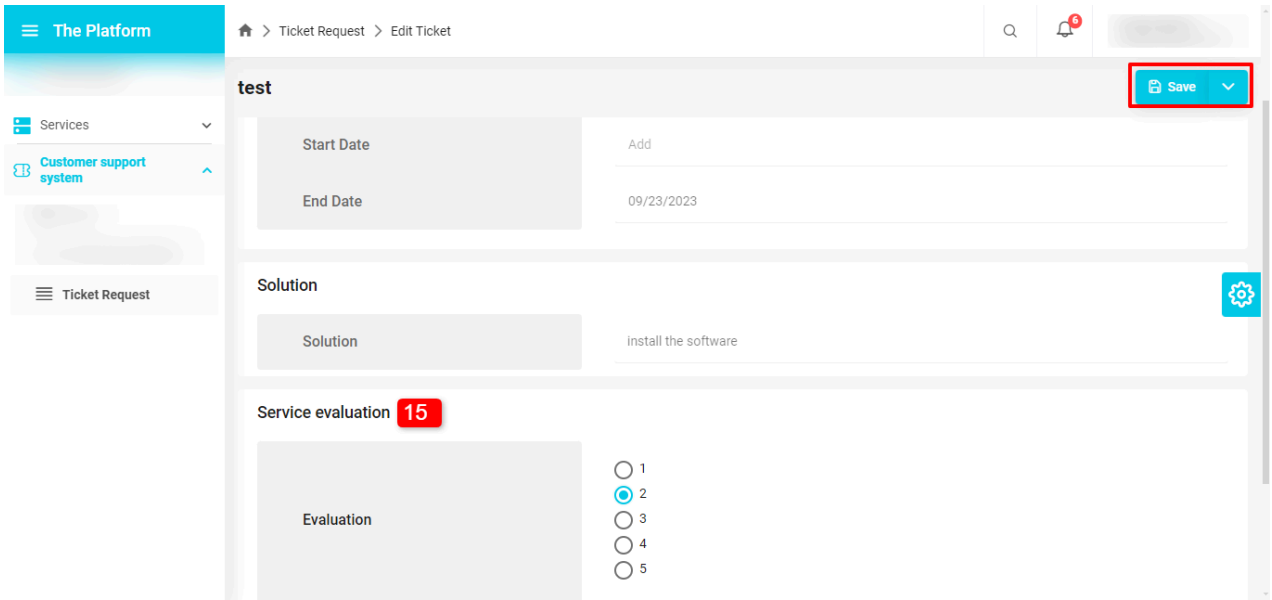
Check: *Customer One* gets *Your request has been resolved, please review it*, and *Representative One* gets *Solution has been reviewed*.

5. Rate the service — as Customer One

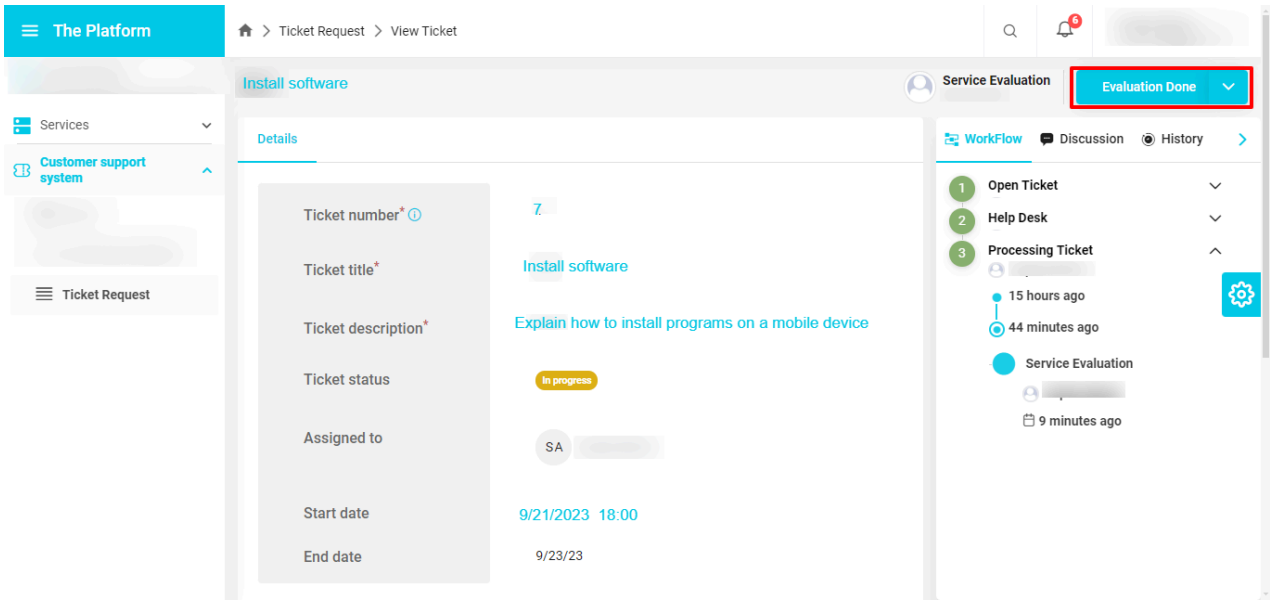
1. Open the ticket from the notification.



2. Edit the ticket, fill in the *Service evaluation* — an *Evaluation score* and a *Comment* — and save.



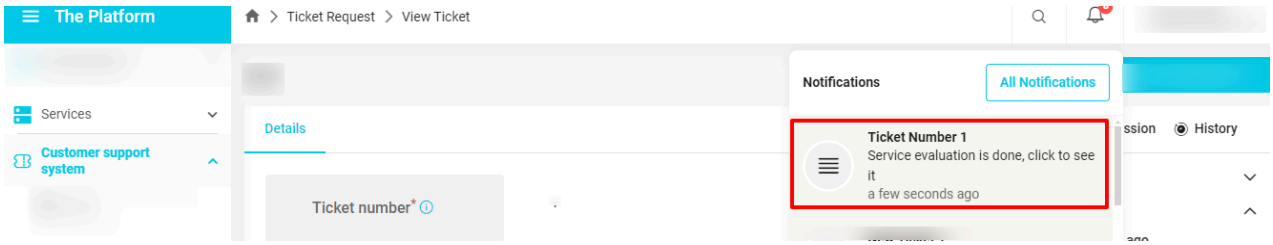
3. Select **Done**.



Check: the desk gets *Service evaluation is done, click to see it*.

6. Close the ticket — as Desk Agent

1. Open the ticket from the notification.



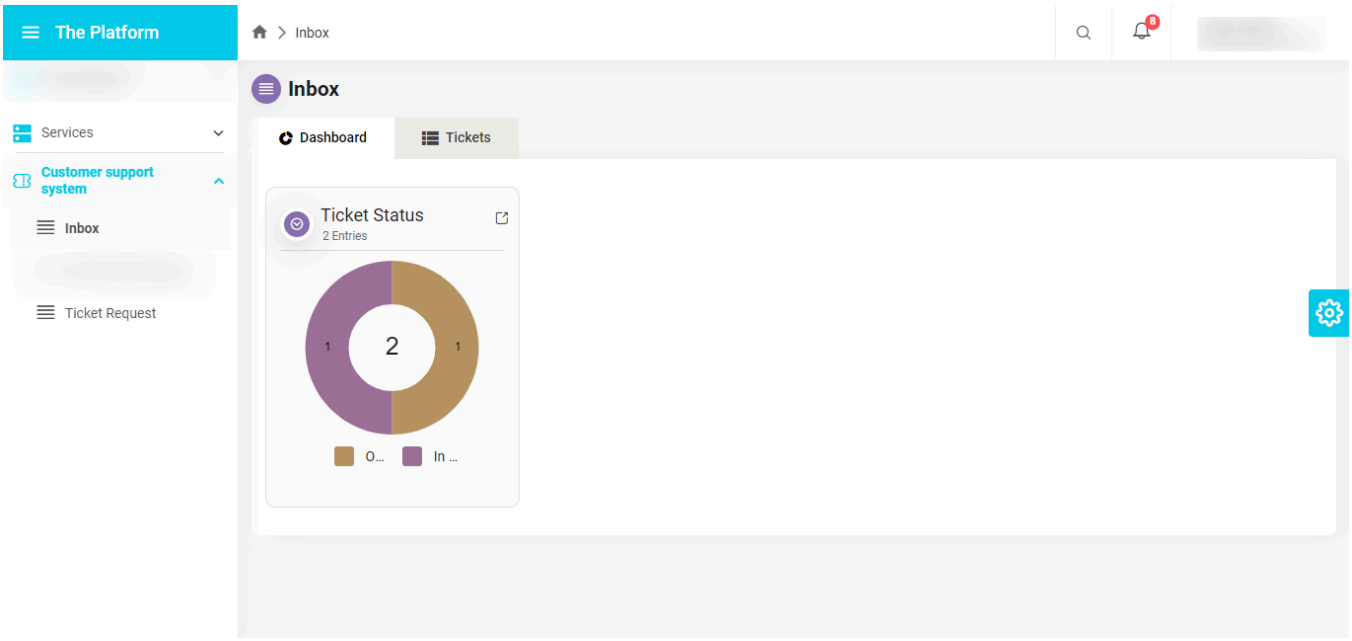
2. Edit it: set *Ticket status* to **Closed** and fill in *End date*. Save.

3. Select **Close Ticket**.

Check: the ticket's workflow is complete, and the ticket appears when you apply the *Closed tickets* query.

7. See the figures — as Support Manager

Open **Incoming Requests**. The *Ticket Status* chart opens first; select a segment to see the tickets behind it.



If something does not happen

What you see	What to check
A notification does not arrive	The reaction's step and action on Add the notifications; and the **Reactions** tab, which shows whether each reaction ran.
The desk agent has no **Take Ownership** button	*Desk Agent* is not in *Technical Support Office*.
The ticket went to *Support Manager* instead of the representative	*Assigned to* was empty, so **Fallback Assignment 1** took over.
A field cannot be changed	That field is **Locked** at the current step — see the field rules on Build the workflow.

Move the system to production

When the system works, move it from your test environment to production — or share a copy — with **Packaging**. A package carries the entities, views, queries, reactions, workflows, reports and pages, and when you import it you map its users and groups to the target environment's. See Packaging apps.

Building a Recruitment System

This tutorial is for **builders** who already know the basics from My First Application. You will build a complete recruitment system: HR publishes vacancies, candidates apply without an account, the application moves through screening, interviews, tasks and a job offer, and the new hire is added as an employee and welcomed on the company timeline.

Work through the pages in order: start with **System Explanation** and **System Requirements**, follow **Building Steps**, then test everything in **Testing the System**.

> ### About the screenshots
>
> The screenshots were taken in the Arabic interface of an earlier release. Every step names the button or field as it appears in the English interface today. Where a screen has changed since, follow the text.

System Explanation
→

System Requirements
→

Building Steps
→

Building the System
→

Testing the System
→

System Explanation

The recruitment system helps an organization hire. It takes a vacancy from the moment a manager drafts it, through publication, applications, interviews and assessment tasks, to a signed offer and a new employee. HR runs the process, managers and interview panels take part in it, and candidates apply and follow up without needing an account.

Goals of the system

1. Make it easy to create and publish vacancies, and to make them visible to potential candidates.
2. Let candidates apply online and submit their details and documents.
3. Let HR follow each application and approve, reject, return it, or make an offer.
4. Support interviews and the assessment tasks given to each candidate.
5. Make it easy to send a job offer and follow the candidate's answer.
6. Provide reports and figures that help the organization make better hiring decisions.

Who uses the system

The system serves everyone in the organization who takes part in hiring:

1. **Candidates:**
    - People looking for a job with the organization. They apply, follow their application, take part in interviews, and send the documents they are asked for.
2. **HR teams:**
    - They run the whole process: create vacancies, review applications, schedule interviews, and make hiring decisions.
3. **Managers and decision makers:**
    - They watch how hiring performs and use the system's figures to improve the process and make sure the right people are hired.
4. **Interview panels:**
    - They see each candidate's details, assess candidates against agreed criteria, and take part in hiring decisions.

Business requirements

Interface requirements

- An easy-to-use interface to create and browse vacancies and applications.
- A detailed view of each vacancy: its requirements, tasks, required experience and expected pay.
- Online applications, with uploads such as a CV.

- A dashboard for managers and staff to follow and manage applications.

Performance requirements

- Applications are processed quickly and candidates get timely answers.
- Many applications can be handled at the same time without delay.

Security and control

- Candidates' personal data and other sensitive information are protected.
- Fine-grained control over who can see which information.

Reports and analytics

- Periodic reports on hiring, interviews, approvals and rejections.
- Reports that support recommendations for improving hiring and selection.

Mobile support

- A responsive interface so candidates can apply and follow their application from a phone.

Users and permissions

- Different users, each with permissions that match their role and responsibilities.
- Administrators can change users' permissions easily.

What does the system need?

This page is the blueprint of the recruitment system. Read it once before you start: the build pages refer back to these field lists, workflows and reactions. The last section lists what must exist in your workspace before you build.

System components

Entities

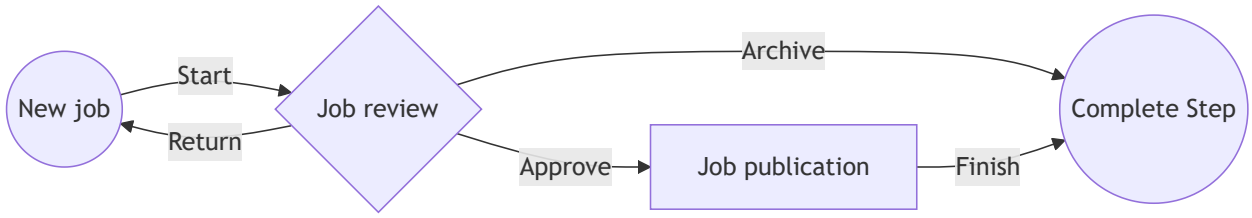
Jobs

- Lets managers create vacancies, and HR review and publish them.

1. **Structure**

Field	Required	Type
Job title	:white_check_mark:	Localized Text
Description	:white_check_mark:	Localized Text
Department	:white_check_mark:	Reference (organization units)
Required qualifications	:white_check_mark:	Reference (the *Competencies* list)
Educational qualifications	:white_check_mark:	Reference (the *Educational qualifications* list)
Years of experience required	:white_check_mark:	Category
Location	:white_check_mark:	Category
Working hours per day		Number
Registration deadline	:white_check_mark:	Date
Job status		Category
HR comments		Localized Text

2. **Workflow**



3. **Reactions**

    1. **Notify HR when a new job is created**

        - Runs when a job is created.

    2. **Notify the job's creator when the job is approved**

- Runs when the job is approved at the *Job review* step.

3. **Notify the job's creator when the job is archived**
   - Runs when the job is archived at the *Job review* step.
4. **Notify the job's creator when the job is returned**
   - Runs when the job is returned to its creator.
5. **Set the job status to *Open***
   - Runs when the job is approved.
6. **Set the job status to *Archived***
   - Runs when the job is archived.
7. **Set the job status to *Closed***
   - Runs when the job publication is finished.
8. **Send a new job to review**
   - Runs when a job is created, and moves it to the *Job review* step.
9. **Close the job on its registration deadline**
   - Runs every day, and moves to the end of the workflow every job whose registration deadline is today.

4. **Queries**
   - **Open jobs**
     - The jobs that are still open, that is not yet closed or archived.

5. **Views**
   - **Available jobs**
     - Public, not pinned to the sidebar, shows the list of entries.
   - **All jobs**
     - Authenticated, pinned to the sidebar, lets people view, create and edit entries.

6. **Dashboard summaries**
   - **Open jobs**
   - **Jobs by status**

7. **Reports**

- **Annual jobs report**

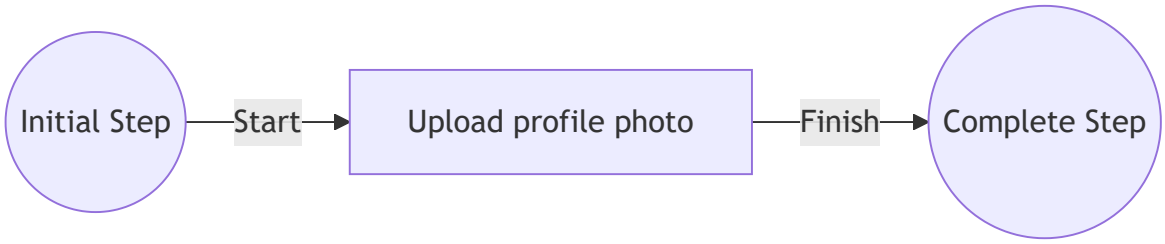
Employees

- Holds the record of each new employee.

1. **Structure**

Field	Required	Type
Full name	:white_check_mark:	Localized Text
Date of birth		Date
Email	:white_check_mark:	Text
Phone number	:white_check_mark:	Phone
Nationality	:white_check_mark:	Reference (the *Nationalities* list)
Residence		Reference (the *Residence* list)
Education		Reference (the *Educational qualifications* list)
Company email		Text
Profile photo		Image
Signed contract		File
Passport / ID card		File

2. **Workflow**



3. **Reactions**

    1. **New employee notification**

        - Runs when an employee record is created, and notifies the HR group.

    2. **Start the workflow**

        - Runs when an employee record is created, and moves the workflow to step 2.

    3. **Tell the rest of the team about the new employee**

        - Notifies the team that a new colleague has joined.

4. **Queries**

5. **Views**

Receptions

- A news entity that announces the latest people to join the company.

1. **Structure**

Field	Required	Type
Title	:white_check_mark:	Text (rich text)
Description	:white_check_mark:	Text (rich text, multiple lines)
Welcome image	:white_check_mark:	Image
Publish date	:white_check_mark:	Date

2. **Reactions**

    1. **Request a profile photo**
        - Runs when a reception is created, and notifies the HR group to ask for a profile photo.

3. **Queries**

4. **Views**

    1. **Create receptions**
        - Authenticated, pinned to the sidebar, lets people view, create and edit entries.
    2. **View receptions**
        - Authenticated, pinned to the sidebar, shows the list of entries.

Services

Job applications

- Lets candidates apply for the open vacancies.

1. **Structure**

Candidate information block:

Field	Required	Type
Full name	:white_check_mark:	Localized Text
Date of birth		Date
Email	:white_check_mark:	Text
Personal photo		Image
Phone number	:white_check_mark:	Phone
Nationality	:white_check_mark:	Reference (the *Nationalities* list)
Residence		Reference (the *Residence* list)
Education		Reference (the *Educational qualifications* list)
Marital status		Category (Married, Single)
Vacancy	:white_check_mark:	Reference (the *Jobs* entity)

Field	Required	Type
CV	:white_check_mark:	File
Cover letter		File
Expected salary		Text
Available from	:white_check_mark:	Date
Key skills	:white_check_mark:	Reference (the *Competencies* list)

Initial evaluation block:

Field	Required	Type
Evaluation score		Category (1 to 10)
Comments		Localized Text, rich text and multiple lines
Candidate status		Reference (the *Candidate status* list)
Contact person		User

Interview L1 block:

Field	Required	Type
Virtual interview?		Boolean
Interview location		Map
Interview link		Localized Text, rich text and multiple lines
Interview date		Date-time
Interviewer		User
Attendees		User, with **Multiple** turned on
Personal impression		Localized Text, rich text and multiple lines
Interview review comments		Localized Text, rich text and multiple lines
Candidate status		Reference (the *Candidate status* list)

First task details block:

Field	Required	Type
Task brief		File
Assigned on		Date
Due date		Date
Task answer		File
Submitted on		Date
Evaluation score		Category (1 to 10)
Comments		Localized Text, rich text and multiple lines

Interview L2 block:

Field	Required	Type
Virtual interview?		Boolean
Interview location		Map
Interview link		Localized Text, rich text and multiple lines
Interview date		Date-time
Interviewer		User
Attendees		User, with **Multiple** turned on
Personal impression		Localized Text, rich text and multiple lines
Interview review comments		Localized Text, rich text and multiple lines
Candidate status		Reference (the *Candidate status* list)

Second task details block:

Field	Required	Type
Task brief		File
Assigned on		Date
Due date		Date
Task answer		File
Submitted on		Date
Evaluation score		Category (1 to 10)
Comments		Localized Text, rich text and multiple lines

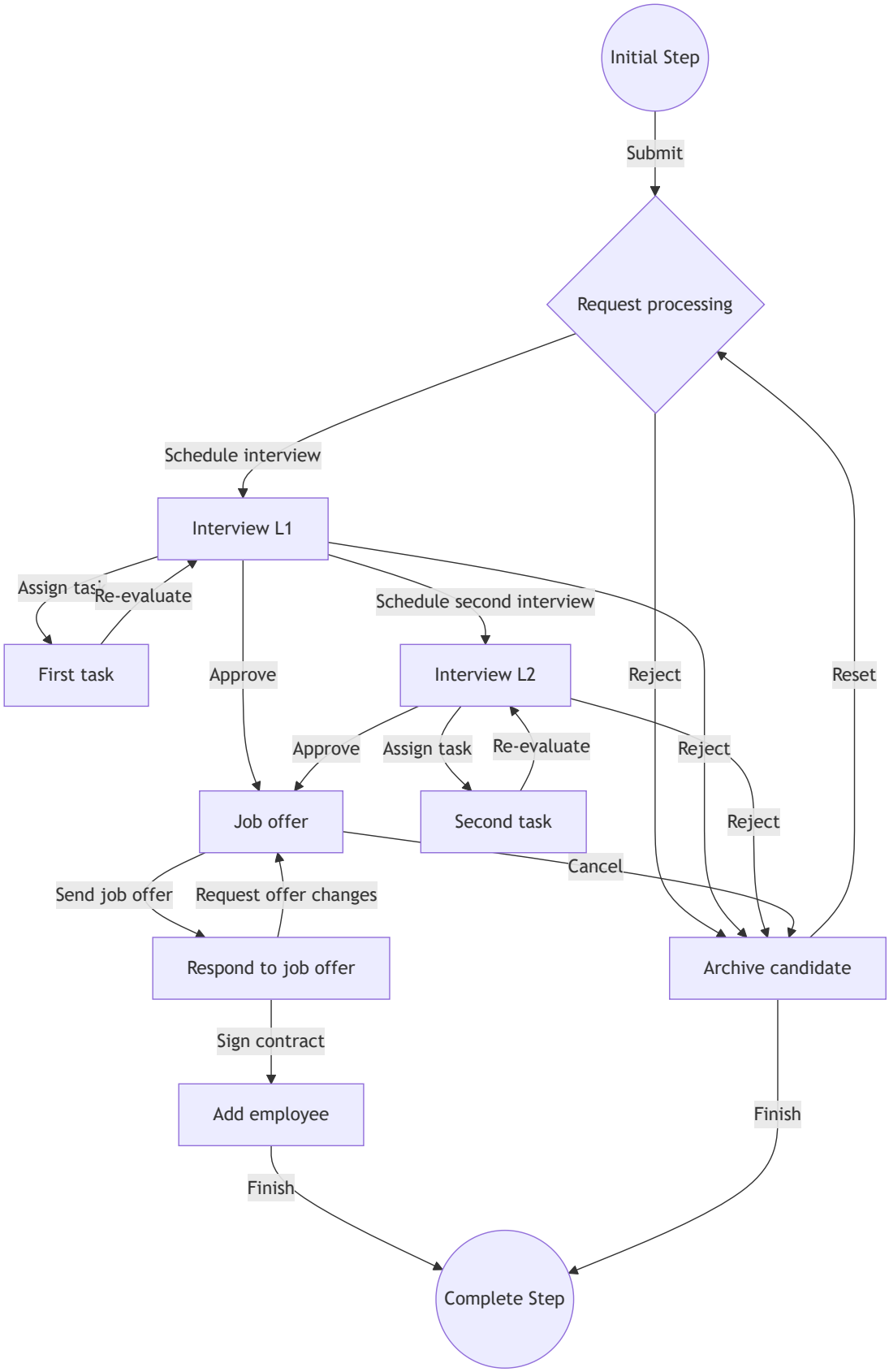
Job offer details block:

Field	Required	Type
Offered contract		File
Signed contract		File
ID / passport		File
Candidate comments		Localized Text, rich text and multiple lines
Company comments		Localized Text, rich text and multiple lines

Add employee block:

Field	Required	Type
Employee		User

2. **Workflow**



3. **Reactions**

    1. **New job application**

        - When a new application is submitted.

2. **Start the workflow**
    - When a new application is submitted.
3. **Job offer**
    - When the application reaches the *Job offer* step.
4. **Add a new employee**
    - When the application reaches the *Add employee* step.
5. **Reject the candidate**
    - When a reject action is taken at the *Request processing*, *Interview L1* or *Interview L2* step.
6. **Notify the candidate of the L1 interview**
    - When *Schedule interview* is selected at the *Request processing* step.
7. **Notify the L1 interviewer**
    - When *Schedule interview* is selected at the *Request processing* step.
8. **Invite the attendees to the L1 interview**
    - When *Schedule interview* is selected at the *Request processing* step.
9. **Assign the first task**
    - When *Assign task* is selected at the *Interview L1* step.
10. **Review the first task**
    - When *Re-evaluate* is selected at the *First task* step.
11. **Job offer response review**
    - When *Request offer changes* is selected at the *Respond to job offer* step.

4. **Views**
    - **HR view**
        - Restricted to HR, not pinned to the sidebar, lets HR view, edit and archive entries.

Preparations before you build

Before you build the system, prepare the following.

Add users and set their permissions

Create the users who will work with the system, and give each one the permissions their role needs.

How to add users is described in Users [Administration › Managing users].

Add the user groups

Organize users into groups by department or job. The system needs at least an *HR* group and an *IT* group.

How to add groups is described in Groups [Administration › Managing groups].

Add the organization units

- Add the organization units that own vacancies and run hiring and interviews. For this system, two are enough:
  - Human Resources
  - Information Technology

How to add organization units is described in Organization units [Administration].

Add the lists

- Create the following lists, each with text values:
  - Nationalities
  - Residence
  - Competencies
  - Educational qualifications
  - Job benefits
  - Candidate status

Lists are managed in **Administration › Applications › Lists**; see Collections & Lists [Using OSOS › Lists: Keep Records Together].

Building Steps

Build the recruitment system in this order. Each step relies on the ones before it: the entities need the lists and groups, the views and dashboards need the queries, and the home page links to the views.

1. **Add users**
    - Add the people who will use the system, and give each one the permissions their role needs.
2. **Add user groups**
    - Create the *HR* and *IT* groups. Workflow steps and notifications are assigned to these groups.
3. **Add organization units**
    - Add the Human Resources unit, which owns several steps of the hiring process, and the Information Technology unit.
4. **Create the lists**
    - Create these lists of text values:
        - Nationalities
        - Residence
        - Competencies
        - Educational qualifications
        - Job benefits
        - Candidate status
5. **Create the Recruitment System module**
    - The module holds every part of the system.
6. **Create the entities**
    - Define the Jobs, Employees and Receptions entities, and the Job Application service.

    1. **Design the structure:**
        - Add the fields each entity holds.
    2. **Design the workflows:**
        - Define the workflow of the Jobs entity and of the Job Application service.
    3. **Add the reactions:**
        - Add each entity's reactions, such as notifications sent to people when something happens.

7. **Create the queries**
    - Create the saved filters that let people find jobs quickly.
8. **Create the views**
    - Create the screens people use to browse jobs and apply for one.
9. **Create the dashboards**
    - Add summaries that give people the key figures at a glance.
10. **Create the report**
    - Create the report that is used to follow and evaluate hiring.
11. **Build the home page**
    - Build the system’s home page: quick links, key figures and the jobs timeline.
12. **Test the system**
    - Run the whole process end to end to check that the system works.

Building the System

These pages build the recruitment system in the order given in Building Steps: the module first, then the entities and the job application service, then the queries, views, dashboards, report and home page that sit on top of them.

Module
→

Job Application Service
→

Queries
→

Entities
→

Views
→

Dashboards
→

Report
→

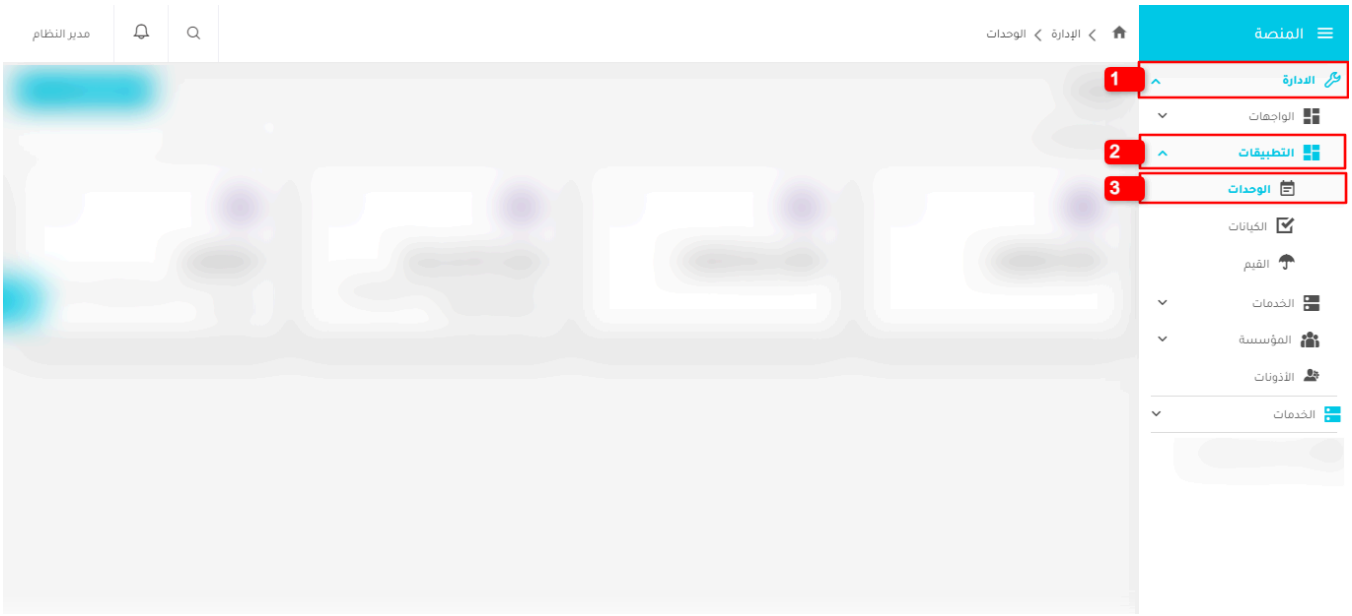
Home Page
→

Creating the Recruitment Module

Everything you build in this tutorial lives in one module, *Recruitment System*. Create it first. For more about modules, see Modules.

Open the Modules page

1. In the sidebar, open **Administration**.
2. Open **Applications**.
3. Select **Modules**.
4. The **Modules** page opens with the list of existing modules.



Add a new module

1. On the **Modules** page, select **Add Module**.

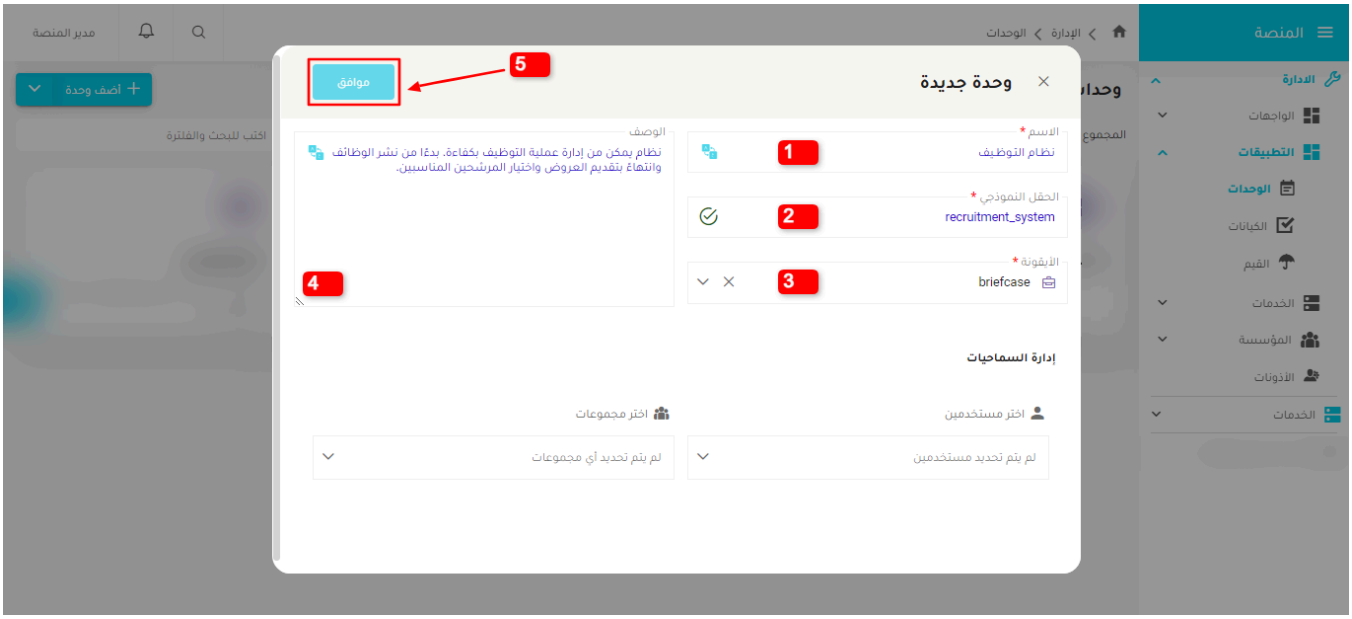


Fill in the module dialog

1. The **NEW MODULE** dialog opens.
2. In **Name**, enter *Recruitment System*, in Arabic and English. The **Normalized name** is filled in for you.
3. In **Icon**, pick an icon for the module.
4. In **Description**, enter: *A system that runs hiring efficiently, from publishing vacancies to making offers and choosing the right candidates.*

Save the module

1. Select **DONE**.



2. The new *Recruitment System* module appears in the list of modules.

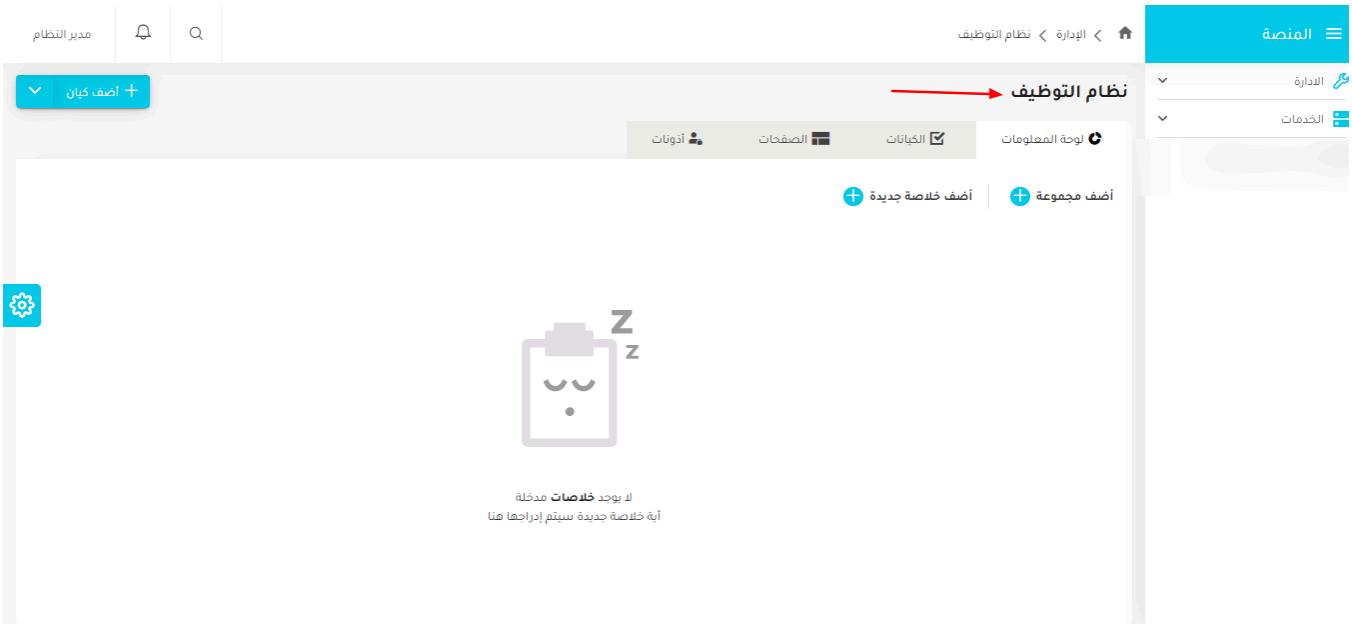
You have created the Recruitment System module.

Open the module

1. On the **Modules** page, use the search bar at the top.
2. Type the name of the module you created, *Recruitment System*.



3. The module appears in the results.
4. Select the module's name to open it. You add the entities on the module's **Entities** tab in the next pages.


المنصة
الإدارة
نظام التوظيف
الادارة
الخدمات
مدير النظام
نظام التوظيف
أضف كيان
لوحة المعلومات
الكيانات
الصفحات
أذونات
أضف مجموعة
أضف خلاصة جديدة
لا يوجد خلاصات مدخلة
أية خلاصة جديدة سيتم إدراجها هنا

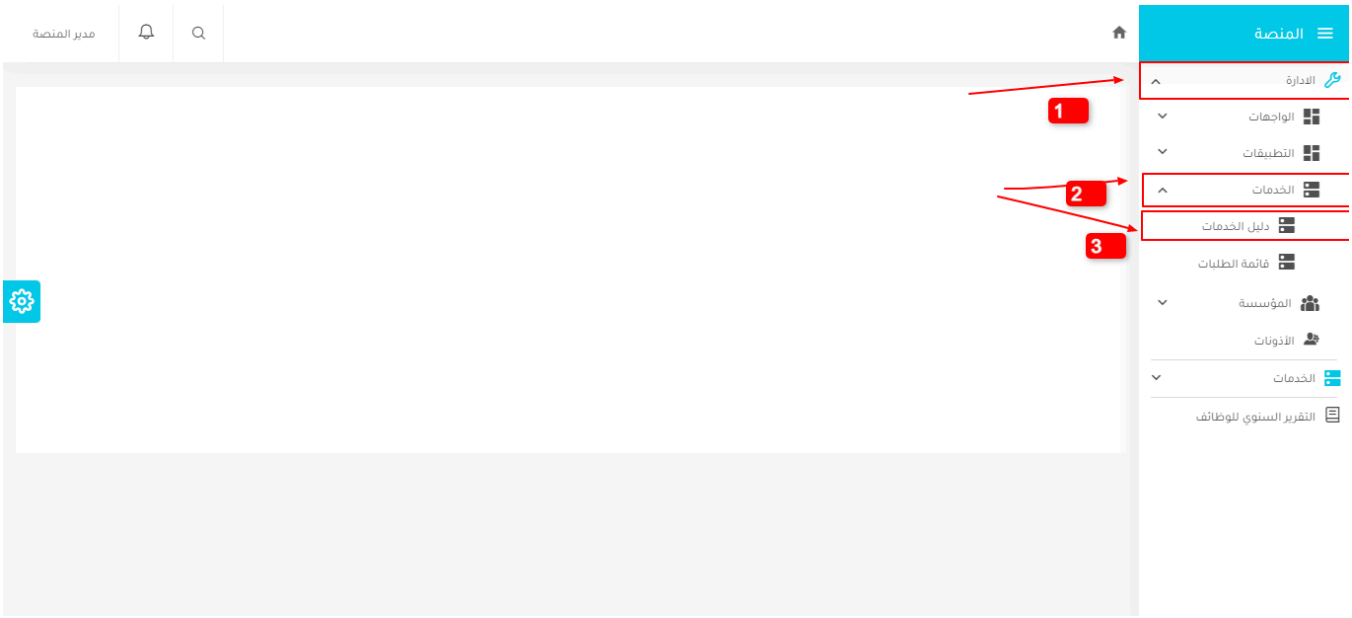
Job Application Service

Candidates use this service to apply for a vacancy. Each application is a request that follows the hiring workflow: screening, one or two interviews, optional assessment tasks, a job offer, and finally a new employee record. This page builds the service and opens it to candidates.

Create the service

Step 1: Open the services catalog

1. In the sidebar, open **Administration**.
2. Open **Services**.
3. Select **Services Catalog**.
4. The list of existing services opens.



Step 2: Add a new service

1. Select **Add Service**.



Step 3: Fill in the service dialog

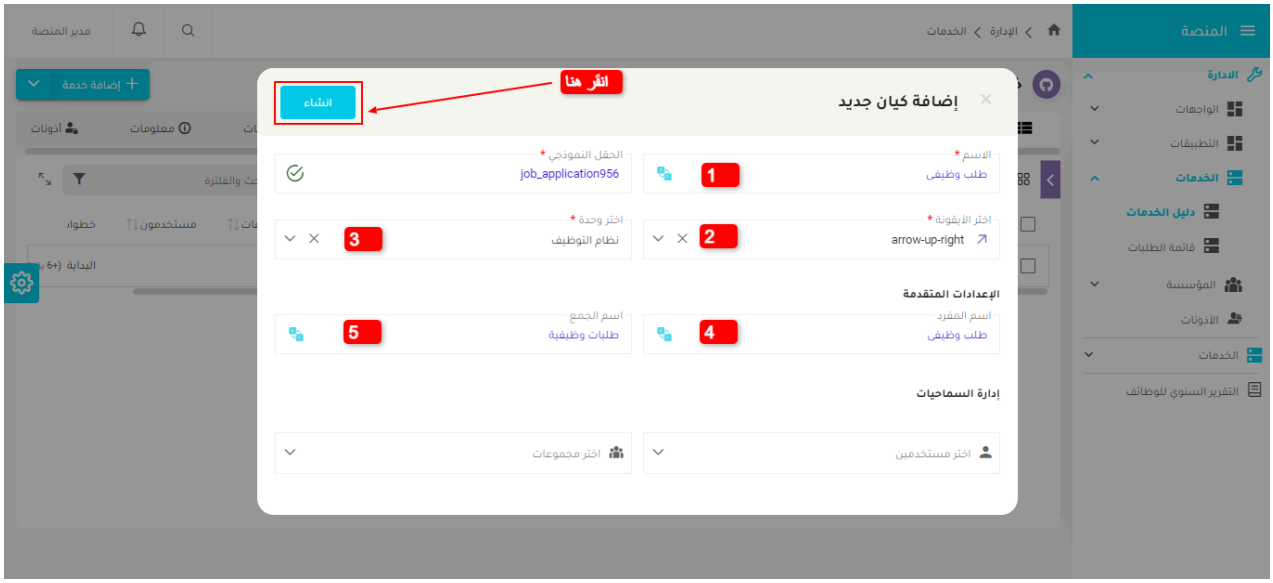
1. The **Add New Data Entity** dialog opens.

2. Fill in:

   - **Name**: *Job Application*, in Arabic and English.

- **Select Icon**: pick an icon for the service.
- **Choose Module**: *Recruitment System*.
- Under **Field Settings**: **Single Name** *Job application* and **Plural Name** *Job applications*.
- **Administration Access**: leave it as it is.

3. Select **Create**. The *Job Application* service is created.



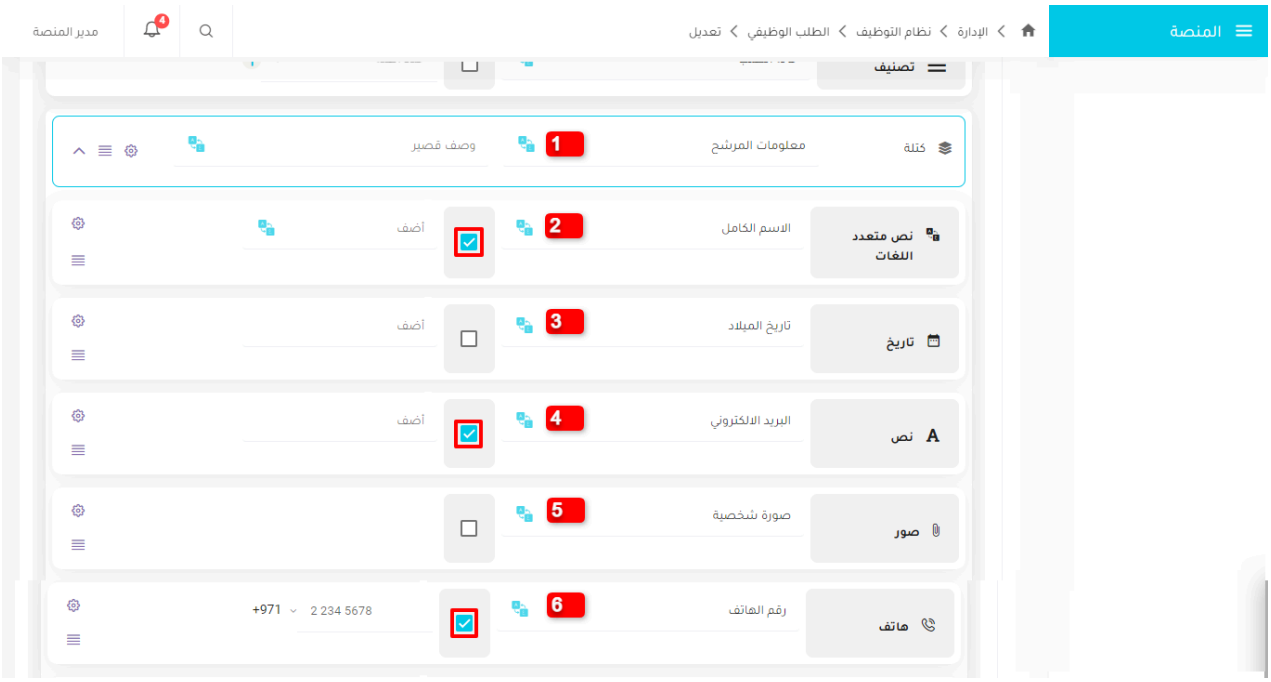
Note: the rest of this page gives only what is specific to the Job Application service: its structure, workflow and reactions. For the step-by-step screens, follow the Jobs entity page.

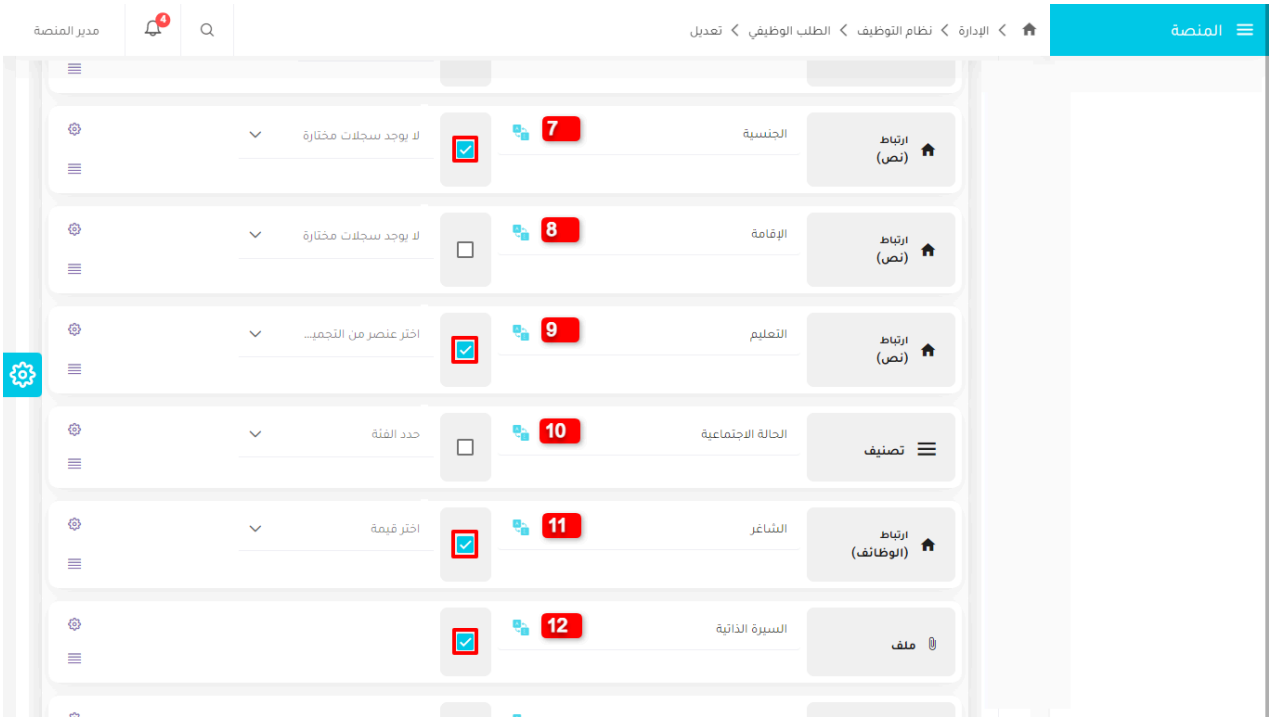
Structure

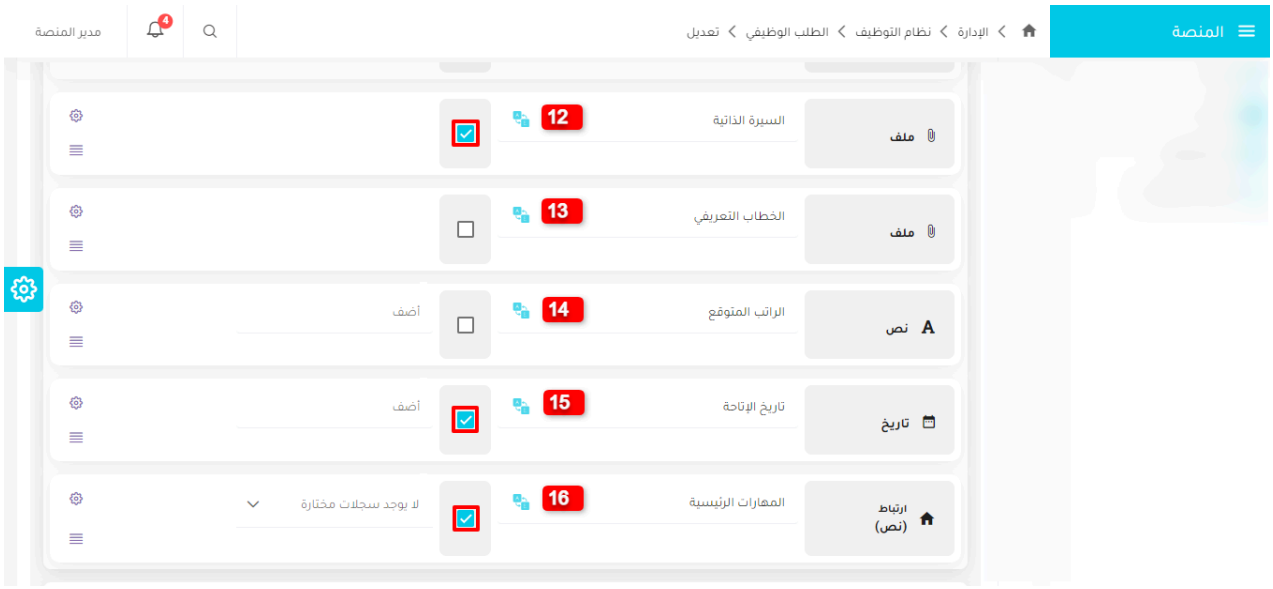
Enter every field name in both Arabic and English.

1. **Candidate information block:**

- **Full name**: required, **Localized Text**.
- **Date of birth**: optional, **Date**.
- **Email**: required, **Text**.
- **Personal photo**: optional, **Image**.
- **Phone number**: required, **Phone**.
- **Nationality**: required, **Reference** to the *Nationalities* list.
- **Residence**: optional, **Reference** to the *Residence* list.
- **Education**: optional, **Reference** to the *Educational qualifications* list.
- **Marital status**: optional, **Category** with the options *Married* and *Single*.
- **Vacancy**: required, **Reference** to the *Jobs* entity.
- **CV**: required, **File**.
- **Cover letter**: optional, **File**.
- **Expected salary**: optional, **Text**.
- **Available from**: required, **Date**.
- **Key skills**: required, **Reference** to the *Competencies* list.


المنصة
الإدارة
نظام التوظيف
الطلب الوظيفي
تعديل
مدير المنصة
تصنيف
كتلة
معلومات المرشح
وصف قصير
1
نص متعدد اللغات
الاسم الكامل
أضف
2
تاريخ
تاريخ الميلاد
3
نص
البريد الالكتروني
4
صور
صورة شخصية
5
هاتف
رقم الهاتف
+971
2 234 5678
6


المنصة
الإدارة
نظام التوظيف
الطلب الوظيفي
تعديل
مدير المنصة
ارتباط (نص)
الجنسية
لا يوجد سجلات مختارة
7
ارتباط (نص)
الإقامة
8
ارتباط (نص)
التعليم
اختر عنصر من التجمي...
9
تصنيف
الحالة الاجتماعية
حدد الفئة
10
ارتباط (الوظائف)
الشاغر
اختر قيمة
11
ملف
السيرة الذاتية
12



2. **Initial evaluation block:**

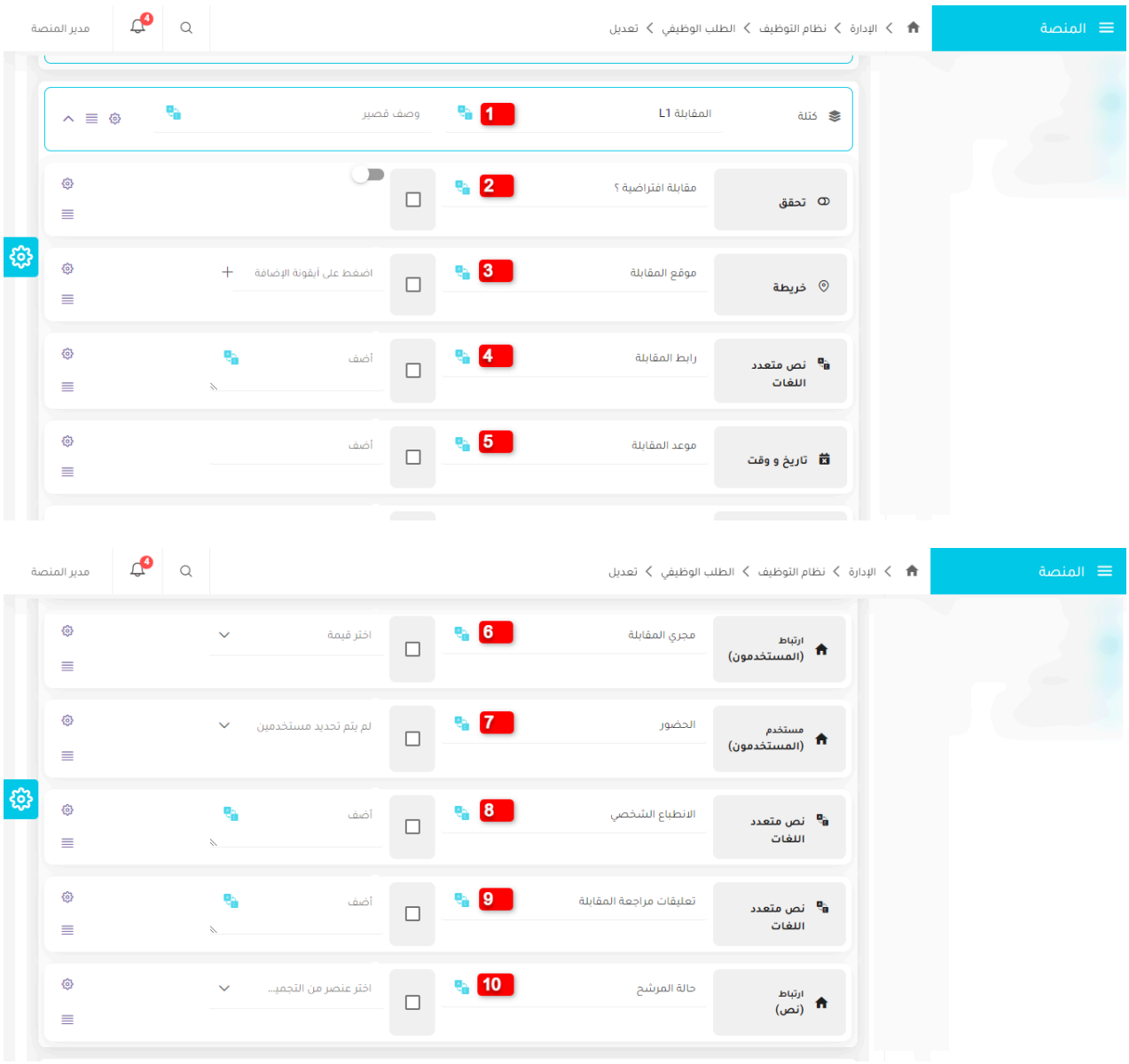
   - **Evaluation score**: optional, **Category** with the options 1 to 10.
   - **Comments**: optional, **Localized Text**; in its settings, turn on rich text and multiple lines.
   - **Candidate status**: optional, **Reference** to the *Candidate status* list.
   - **Contact person**: optional, **User**. A reaction fills it in with the HR member who handles the application.



3. **Interview L1 block:**

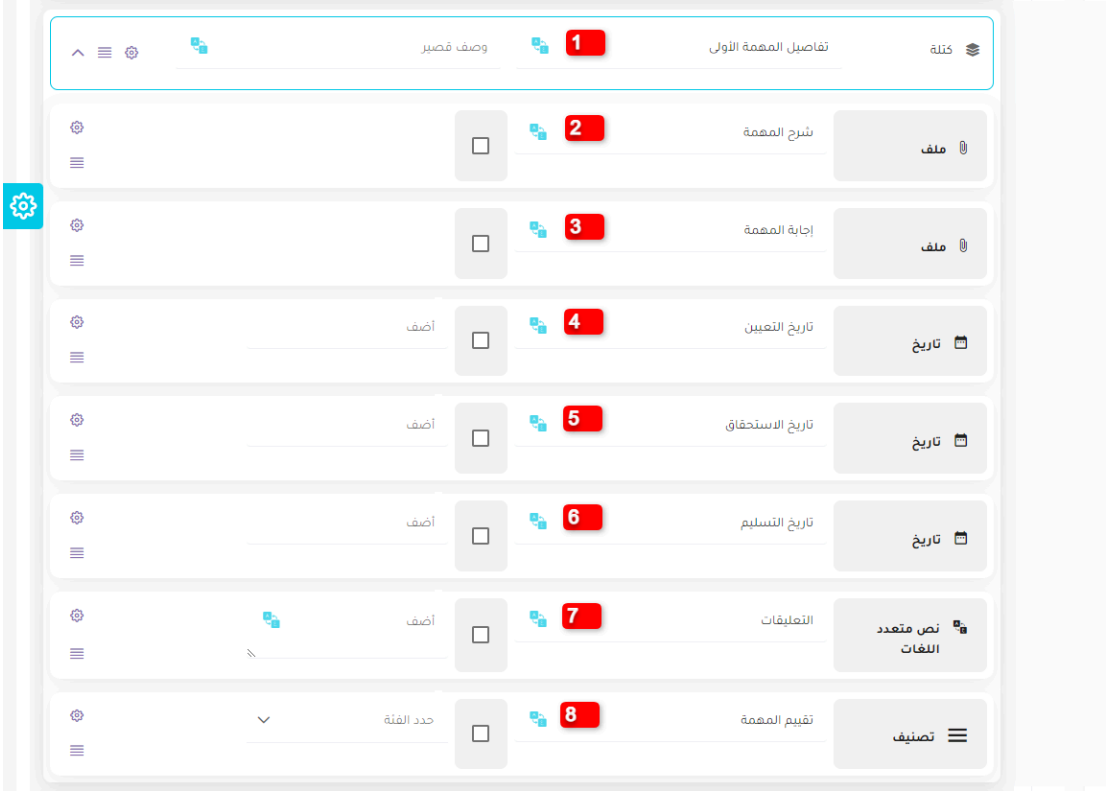
   - **Virtual interview?**: optional, **Boolean**.
   - **Interview location**: optional, **Map**.
   - **Interview link**: optional, **Localized Text**, with rich text and multiple lines.
   - **Interview date**: optional, **Date-time**.
   - **Interviewer**: optional, **User**.
   - **Attendees**: optional, **User**; in its settings, turn on **Multiple**.
   - **Personal impression**: optional, **Localized Text**, with rich text and multiple lines.

- **Interview review comments**: optional, **Localized Text**, with rich text and multiple lines.
- **Candidate status**: optional, **Reference** to the *Candidate status* list.

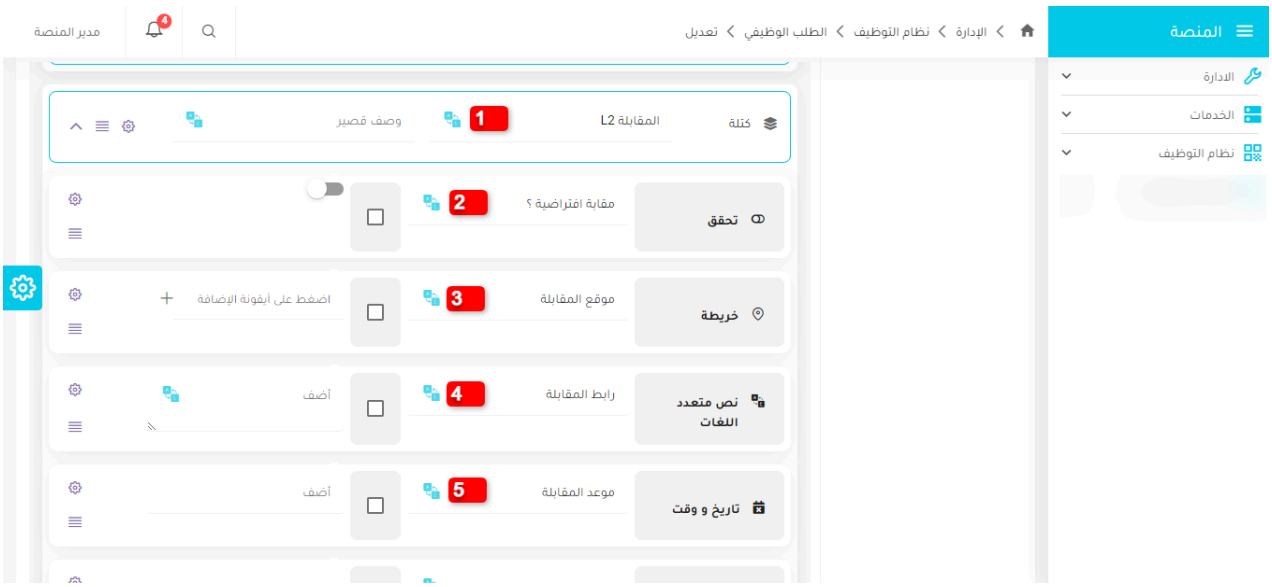


4. **First task details block:**

- **Task brief**: optional, **File**.
- **Assigned on**: optional, **Date**.
- **Due date**: optional, **Date**.
- **Task answer**: optional, **File**.
- **Submitted on**: optional, **Date**.
- **Evaluation score**: optional, **Category** with the options 1 to 10.
- **Comments**: optional, **Localized Text**, with rich text and multiple lines.

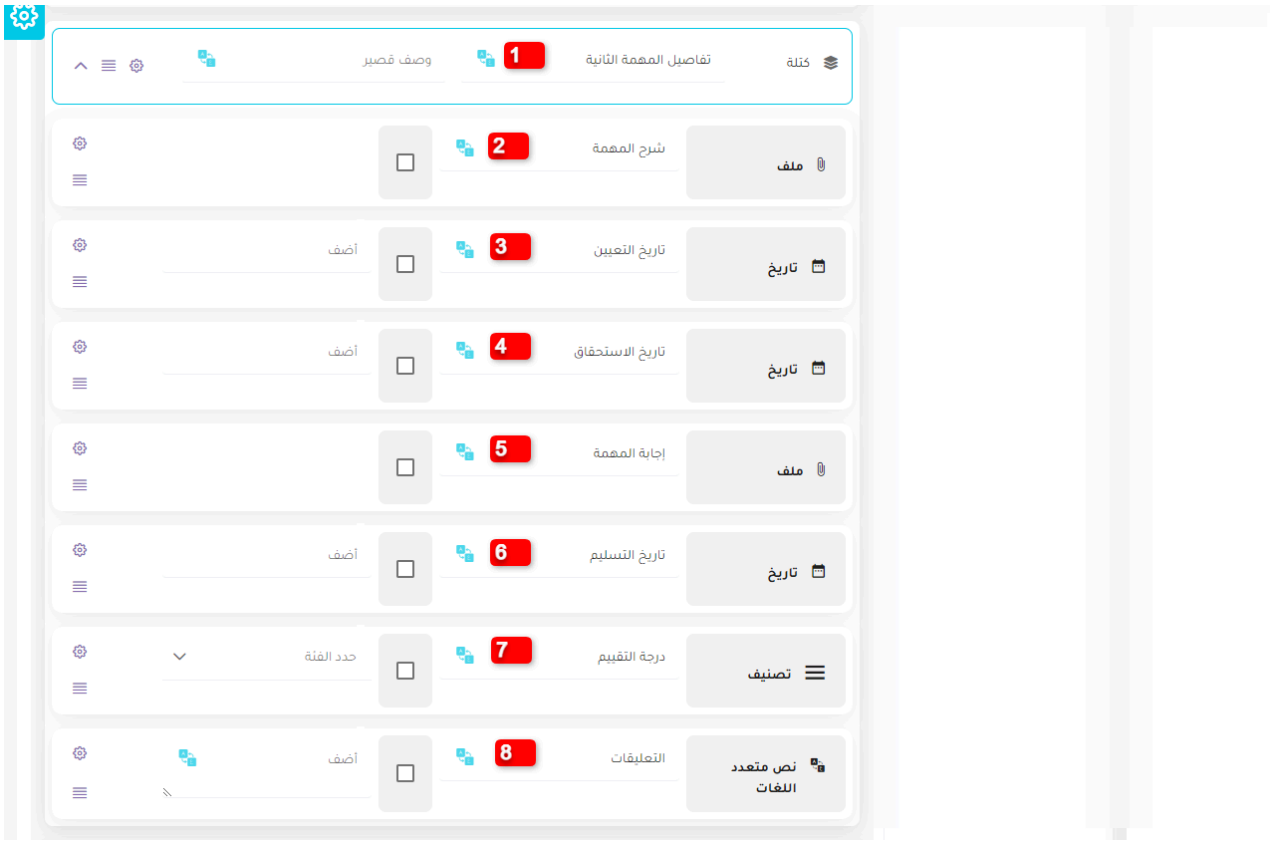


5. **Interview L2 block:** the same fields as the Interview L1 block.



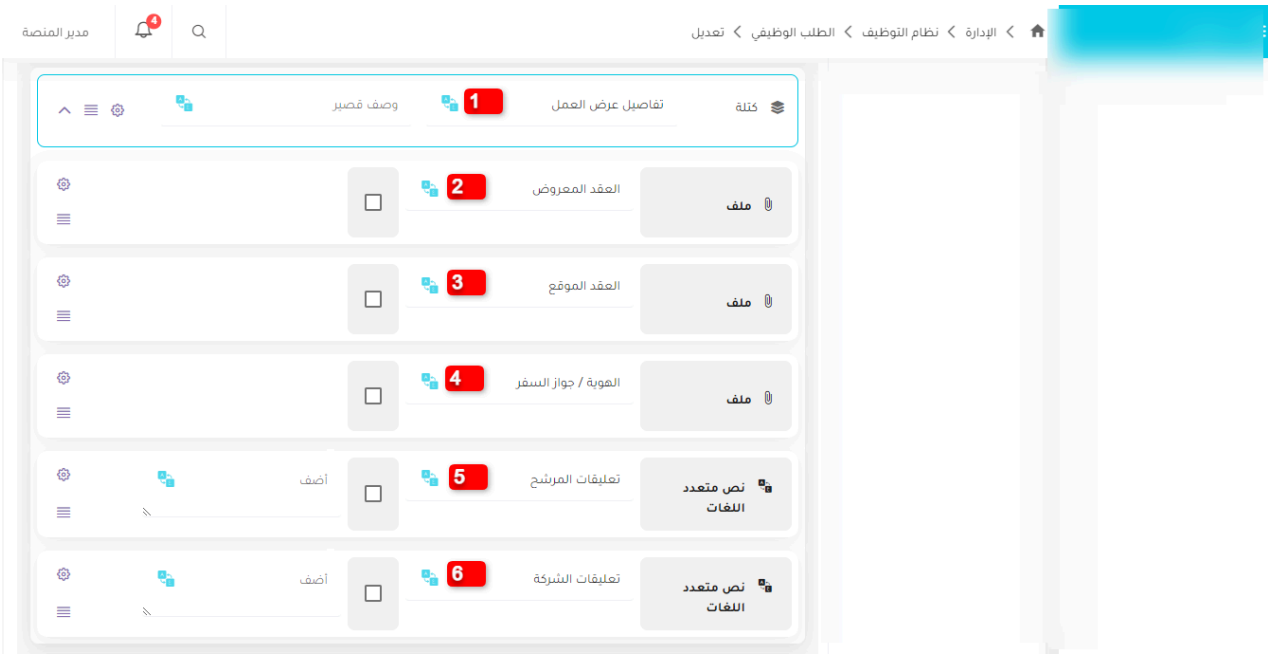


6. **Second task details block:** the same fields as the First task details block.



7. **Job offer details block:**

   - **Offered contract**: optional, **File**.
   - **Signed contract**: optional, **File**.
   - **ID / passport**: optional, **File**.
   - **Candidate comments**: optional, **Localized Text**, with rich text and multiple lines.
   - **Company comments**: optional, **Localized Text**, with rich text and multiple lines.



8. **Add employee block:**
   - **Employee**: optional, **User**.



9. In each block's settings, set the block to show its fields in three columns and turn on **Display as tab**, so every block becomes a tab of the form.



10. Select **Save** at the top of the page.

Sensitive fields

A job application holds the candidate's personal data: email, phone, expected salary, ID or passport. You can mark these fields as **Sensitive**, so their value is masked (****) for anyone who has not been granted access. See Sensitive fields.

Workflow

Build the workflow on the **Workflows** tab, as for the Jobs entity. For each step: set its **Step Owner**, add and rename its actions (point each action at its next step with **Target Step** in **Action Settings**), and set its **Requirements**: the **View Option** of each block and field at that step.

The pattern for the requirements is the same at every step: blocks the step owner only needs to read are **Locked**, blocks that do not concern the step are **Hidden**, and the block the step works on is set to **None** so that you can set each of its fields (**Mandatory**, **Editable**, **Locked** or **Hidden**).

Step 1: Initial Step

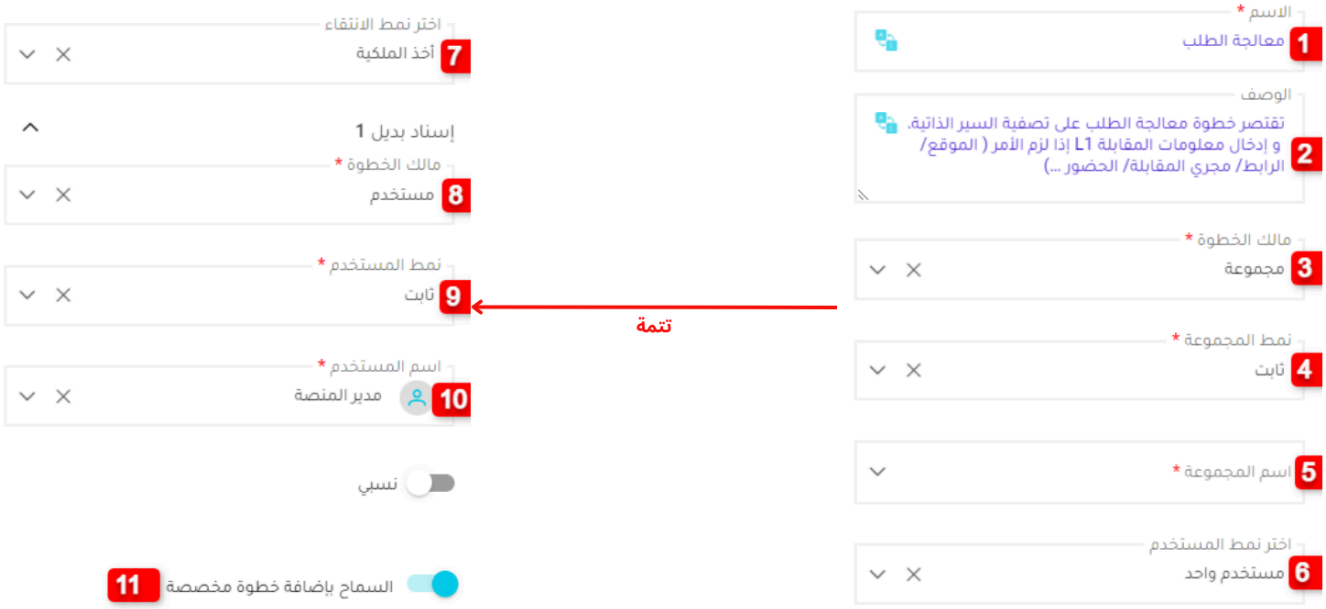
- **Name**: *Initial Step*
- **Owner**: the applicant
- **Actions**:
  - *Submit* (to *Request processing*)

The applicant fills in the *Candidate information* block. If the form shows HR's blocks to applicants, hide them at this step.



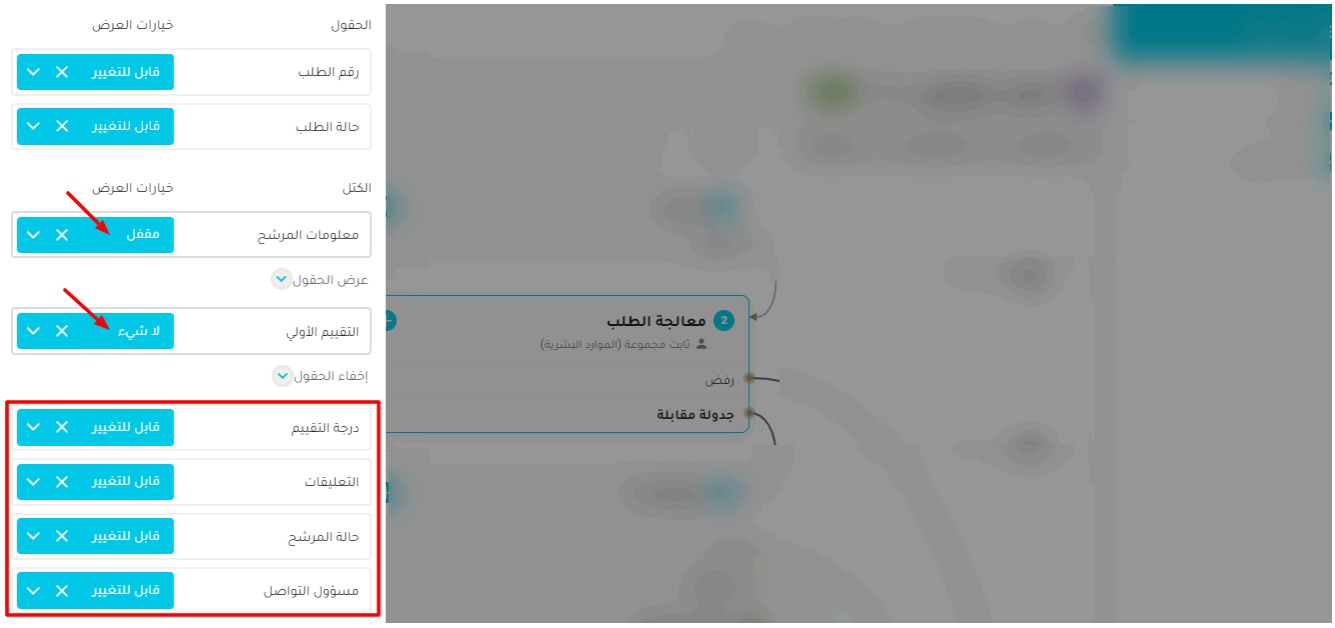
Step 2: Request processing

- **Name**: *Request processing*. Description: *Screen the CVs and, if needed, enter the Interview L1 details (location or link, interviewer, attendees).*
- **Owner**: an HR member: **Group**, **Fixed**, *HR*, **Single User**, **Take Ownership**. Fallback: a fixed user. Turn on **Allow Adhoc step**.
- **Actions**:
  - *Schedule interview* (to *Interview L1*)
  - *Reject* (to *Archive candidate*)



- **Requirements**:

Block	View Option	Fields
Candidate information	**Locked**	
Initial evaluation	**None**	All fields **Editable**
Interview L1	**None**	*Virtual interview?*, *Interview date*, *Interviewer*, *Attendees*: **Mandatory**. *Interview location*, *Interview link*: **Editable**. *Personal impression*, *Interview review comments*, *Candidate status*: **Hidden**
All other blocks	**Hidden**	





Step 3: Interview L1

- **Name**: *Interview L1*
- **Owner**: an HR member, as in step 2.
- **Actions**:
  - *Reject* (to *Archive candidate*)
  - *Approve* (to *Job offer*)
  - *Schedule second interview* (to *Interview L2*)
  - *Assign task* (to *First task*)



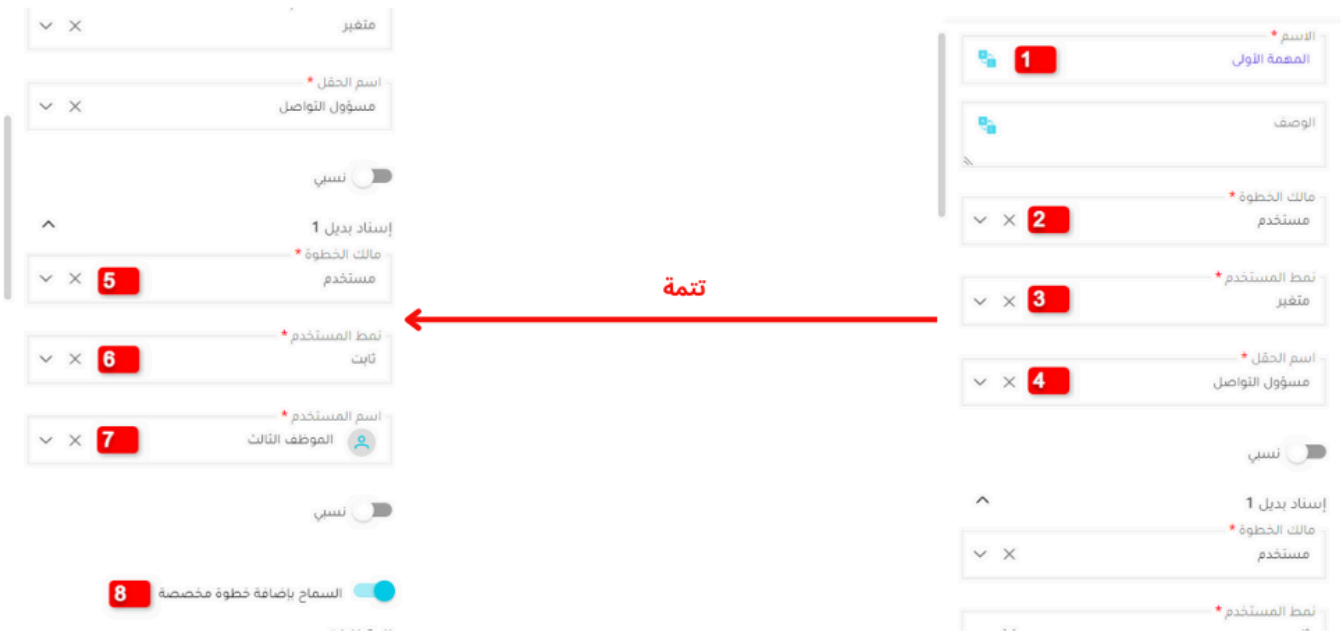
- **Requirements**:

Block	View Option	Fields
Candidate information, Initial evaluation	**Locked**	
Interview L1	**None**	*Interview date*, *Interviewer*, *Attendees*: **Locked**. *Personal impression*, *Candidate status*: **Mandatory**. *Interview review comments*: **Editable**. The rest: **Hidden**
First task details	**None**	*Task brief*, *Assigned on*, *Due date*, *Evaluation score*, *Comments*: **Editable**. *Task answer*, *Submitted on*: **Hidden**
Interview L2	**None**	Interview arrangements **Editable**; *Personal impression*, *Interview review comments*, *Candidate status*: **Hidden**
Second task details	**Editable**	
Job offer details, Add employee	**Hidden**	



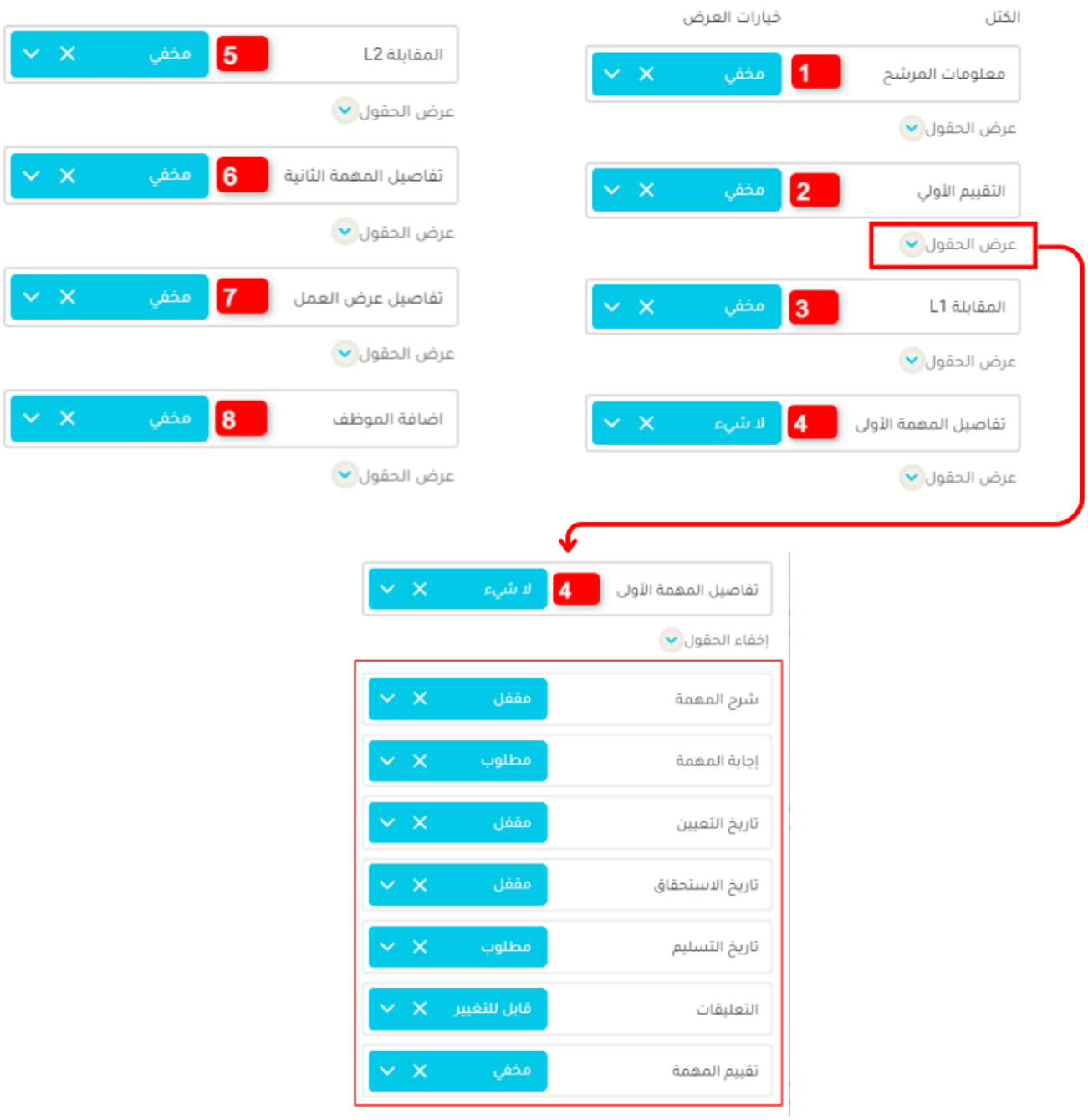
Step 4: First task

- **Name**: *First task*
- **Owner**: the owner of the previous step: **User**, **Previous Step Owner**, *Interview L1*.
- **Actions**:
  - *Re-evaluate* (back to *Interview L1*)



- **Requirements**:

Block	View Option	Fields
First task details	**None**	*Task brief*, *Assigned on*, *Due date*: **Locked**. *Task answer*, *Submitted on*: **Mandatory**. *Comments*: **Editable**. *Evaluation score*: **Hidden**
All other blocks	**Hidden**	



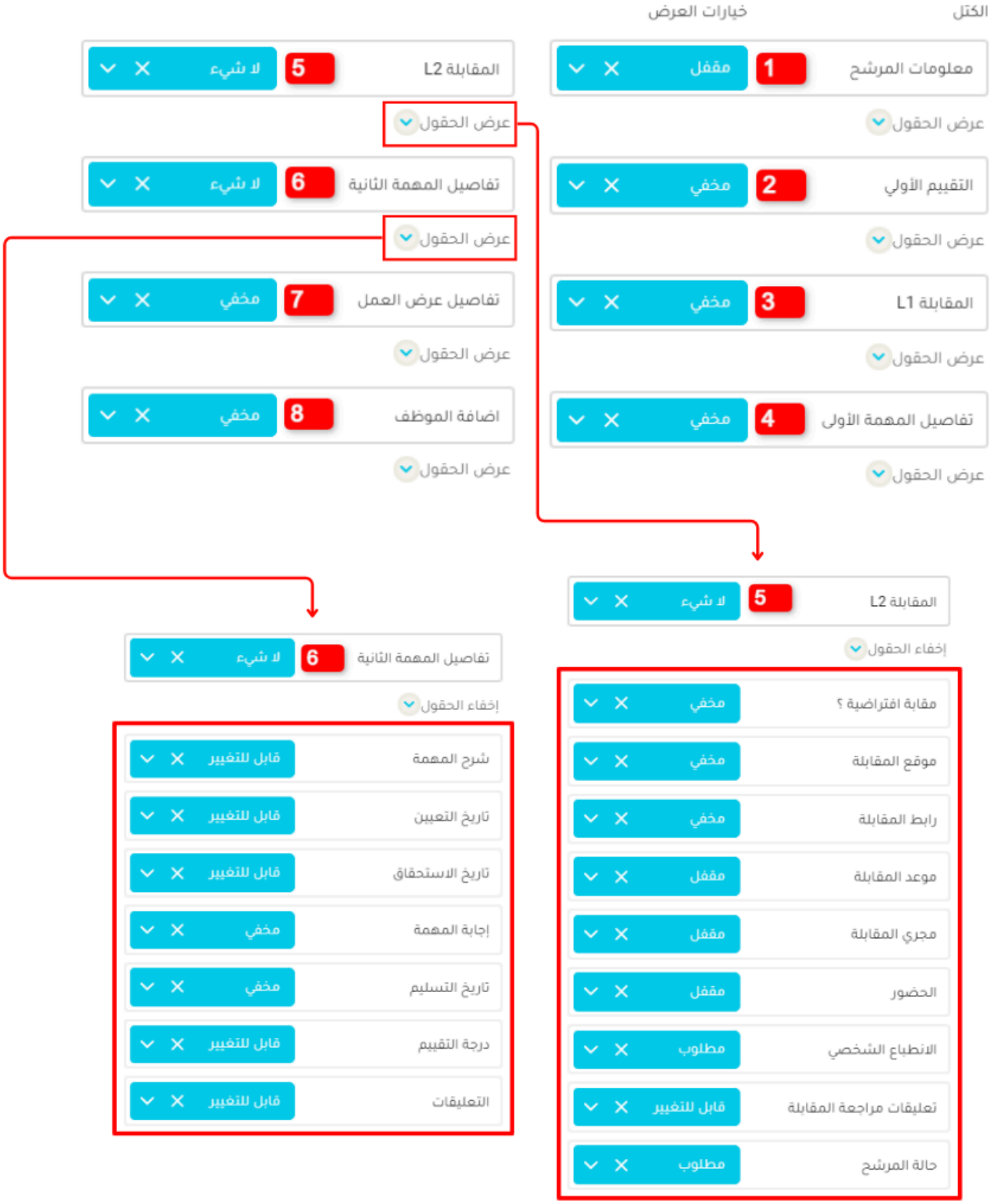
Step 5: Interview L2

- **Name**: *Interview L2*
- **Owner**: an HR member, as in step 2.
- **Actions**:
  - *Reject* (to *Archive candidate*)
  - *Approve* (to *Job offer*)
  - *Assign task* (to *Second task*)



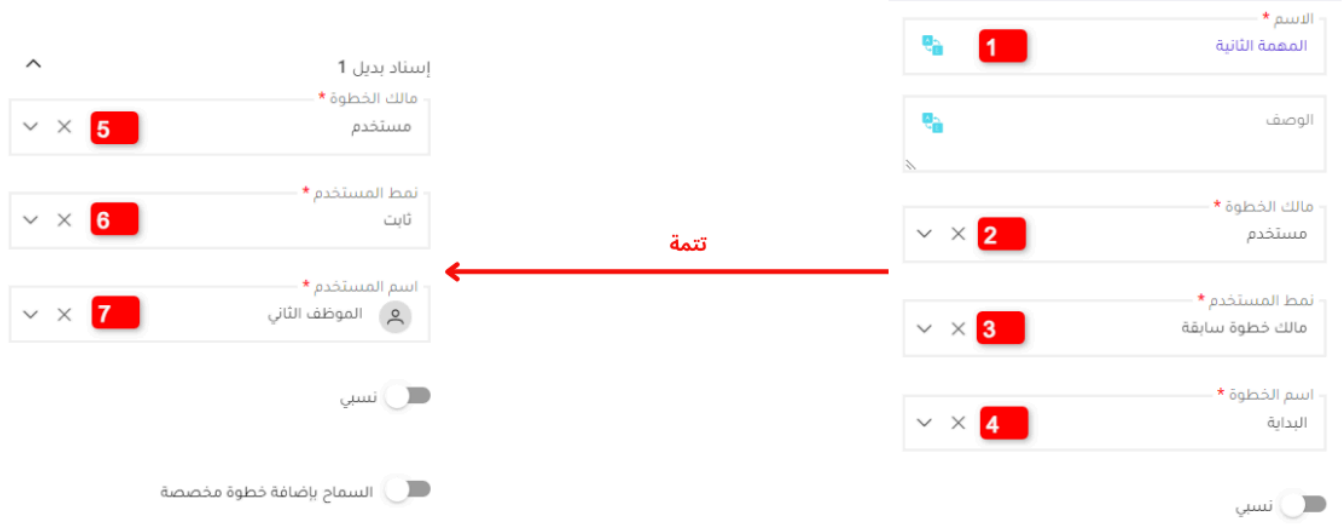
- **Requirements**:

Block	View Option	Fields
Candidate information	**Locked**	
Interview L2	**None**	*Interview date*, *Interviewer*, *Attendees*: **Locked**. *Personal impression*, *Candidate status*: **Mandatory**. *Interview review comments*: **Editable**. The rest: **Hidden**
Second task details	**None**	*Task brief*, *Assigned on*, *Due date*, *Evaluation score*, *Comments*: **Editable**. *Task answer*, *Submitted on*: **Hidden**
All other blocks	**Hidden**	



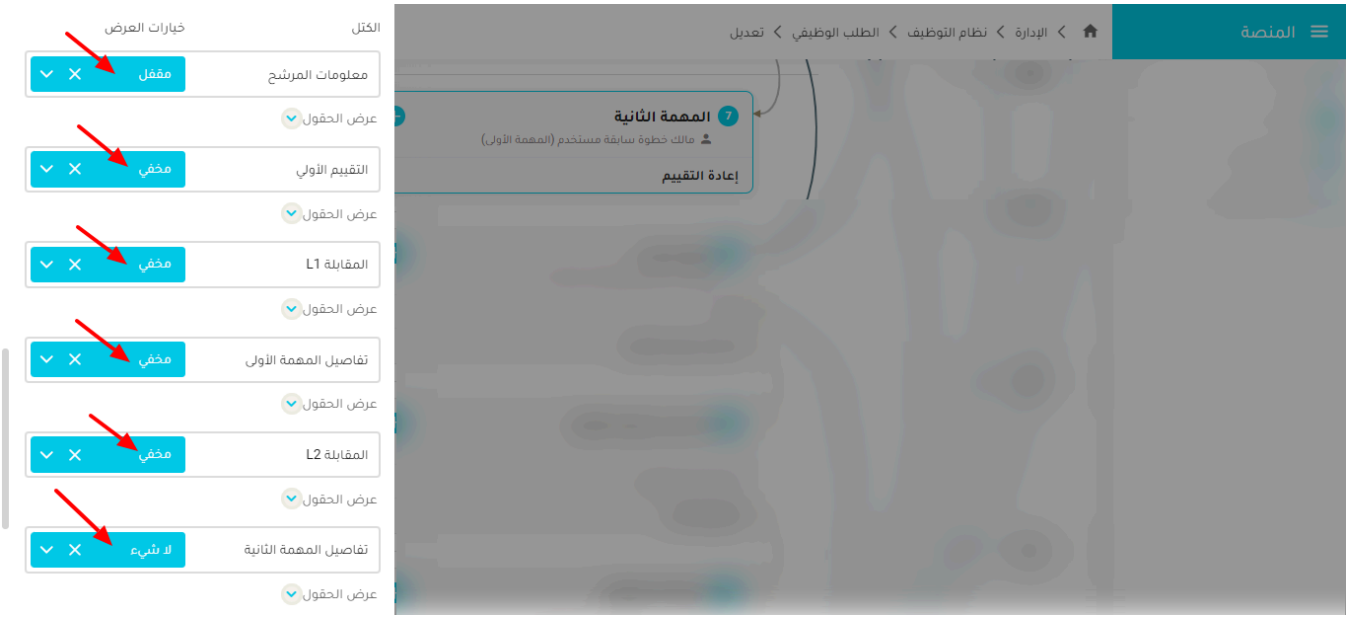
Step 6: Second task

- **Name**: *Second task*
- **Owner**: the owner of the previous step: **User**, **Previous Step Owner**.
- **Actions**:
  - *Re-evaluate* (back to *Interview L2*)



- **Requirements**:

Block	View Option	Fields
Candidate information	**Locked**	
Second task details	**None**	*Task brief*, *Assigned on*, *Due date*: **Locked**. *Task answer*, *Submitted on*: **Mandatory**. *Comments*: **Editable**. *Evaluation score*: **Hidden**
All other blocks	**Hidden**	





Step 7: Job offer

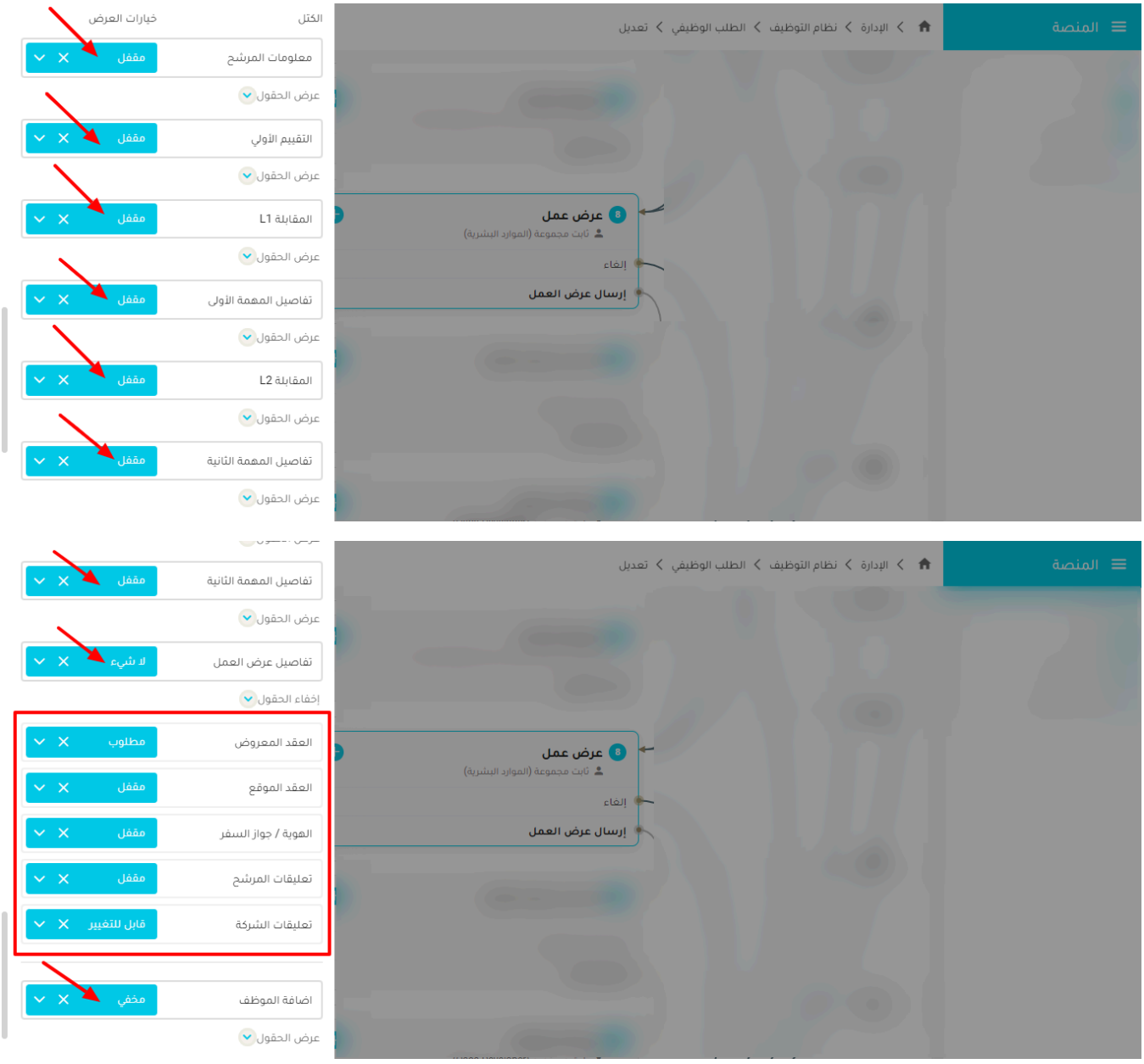
- **Name**: *Job offer*
- **Owner**: the HR manager: **Group**, **Fixed**, *HR*, **Single User**, and **Choose Selection Type Org Unit Manager**. Fallback: a fixed user. Turn on **Allow Adhoc step**.
- **Actions**:
  - *Send job offer* (to *Respond to job offer*)
  - *Cancel* (to *Archive candidate*, if the candidate turns the offer down)



- **Requirements**:

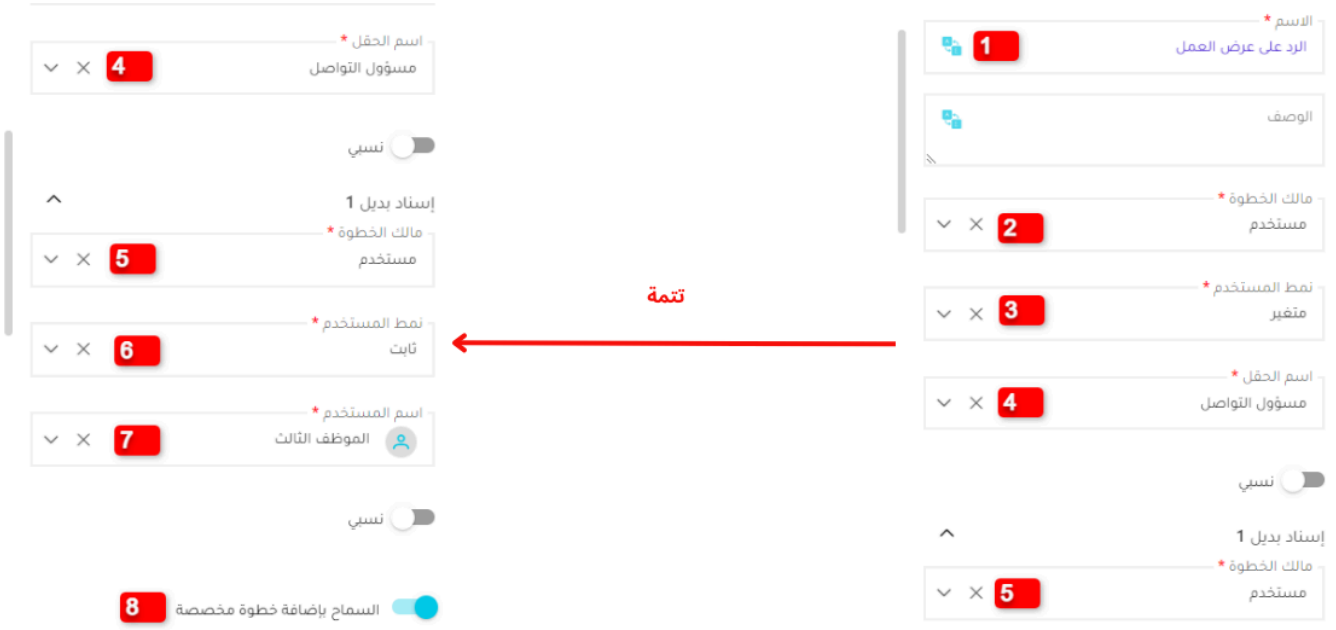
Block	View Option	Fields
Candidate information, Initial evaluation, Interview L1, First task details, Interview L2, Second task details	**Locked**	

Block	View Option	Fields
Job offer details	**None**	*Offered contract*: **Mandatory**. *Company comments*: **Editable**. *Signed contract, ID / passport, Candidate comments*: **Locked**
Add employee	**Hidden**	



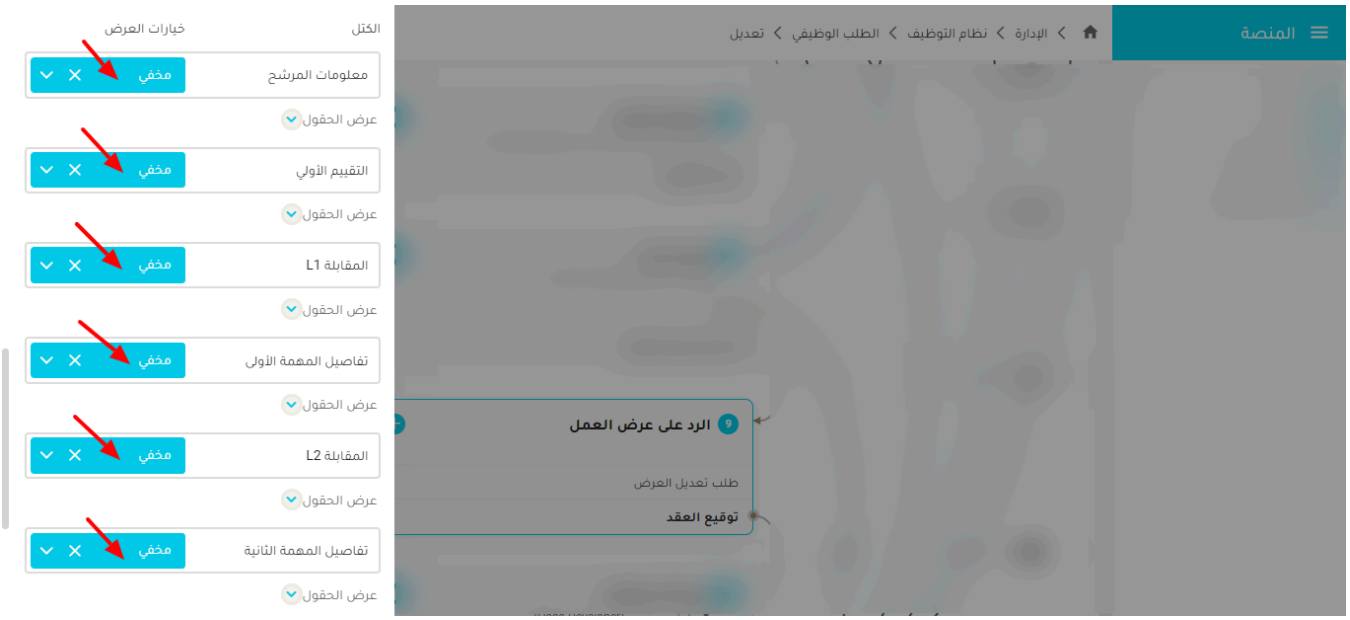
Step 8: Respond to job offer

- **Name**: *Respond to job offer*
- **Owner**: the owner of the first step (the applicant): **User**, **Previous Step Owner**, *Initial Step*. Because candidates apply without an account, in practice the contact person records the candidate's answer; see Testing the system.
- **Actions**:
  - *Request offer changes* (back to *Job offer*)
  - *Sign contract* (to *Add employee*)



- **Requirements**:

Block	View Option	Fields
Job offer details	**None**	*Signed contract*, *ID / passport*, *Candidate comments*: **Editable**. *Offered contract*, *Company comments*: **Locked**
All other blocks	**Hidden**	





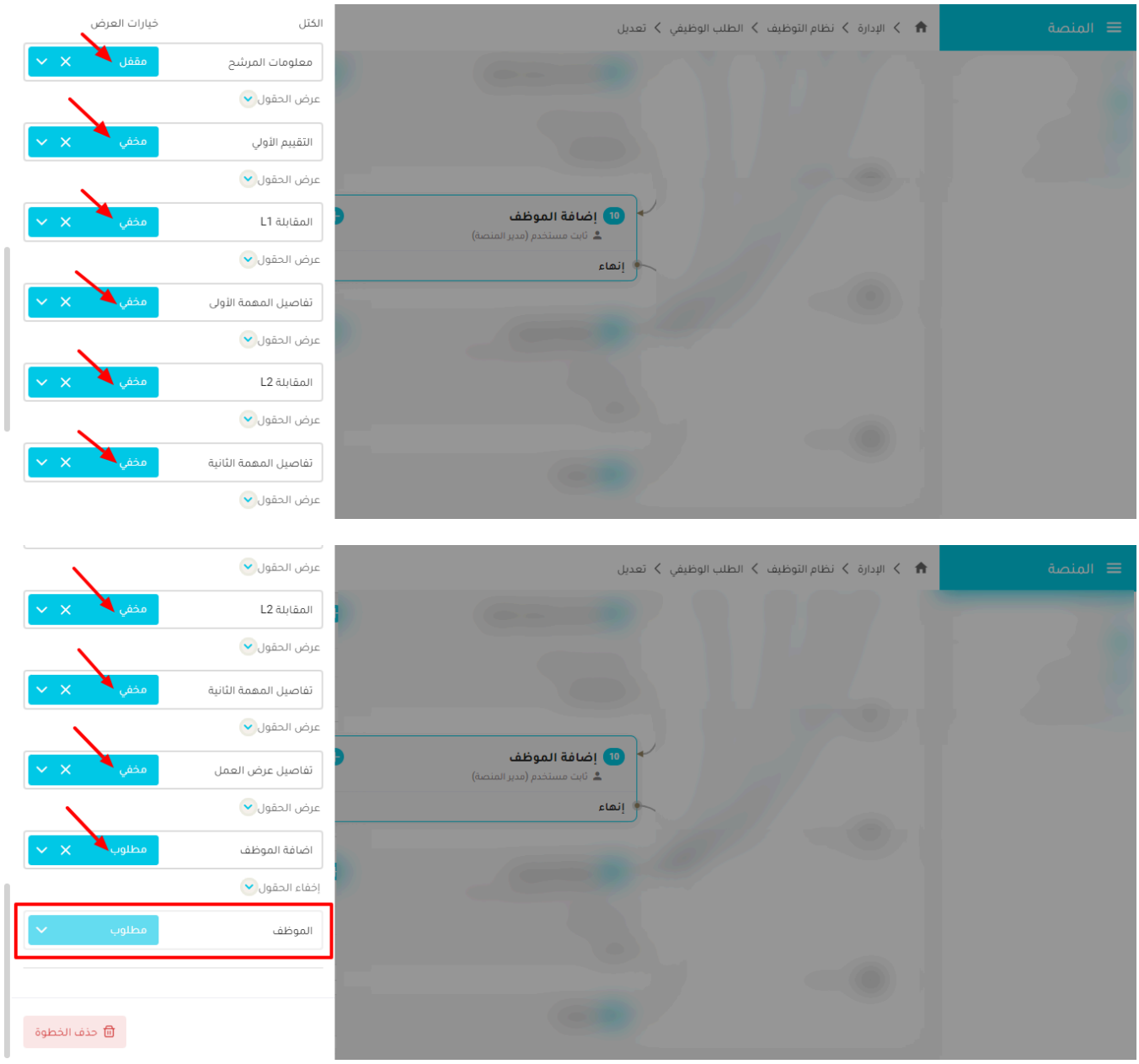
Step 9: Add employee

- **Name**: *Add employee*
- **Owner**: a fixed user who can create user accounts: **User**, **Fixed**. The tutorial uses the platform administrator.
- **Actions**:
  - *Finish* (to the final step)



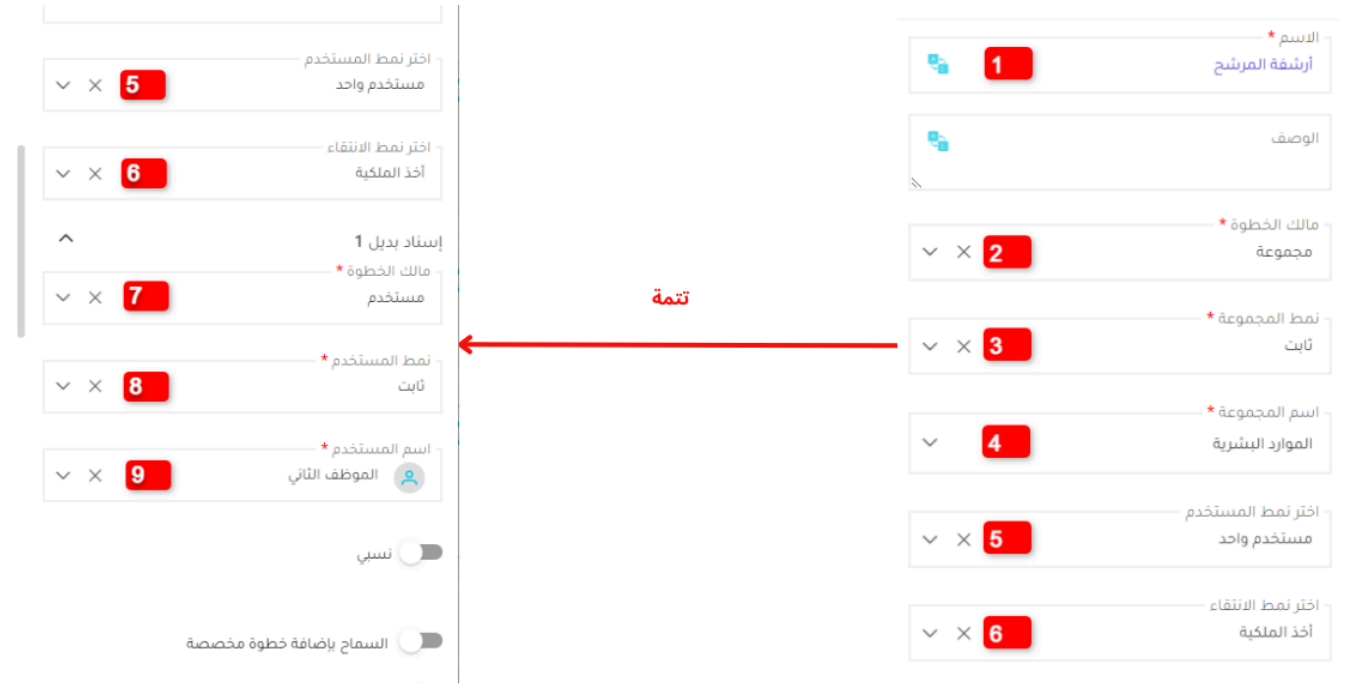
- **Requirements**:

Block	View Option	Fields
Candidate information	**Locked**	
Add employee	**Mandatory**	*Employee*: **Mandatory**
All other blocks	**Hidden**	



Step 10: Archive candidate

- **Name**: *Archive candidate*
- **Owner**: an HR member, as in step 2.
- **Actions**:
  - *Reset* (back to *Request processing*, to evaluate the candidate again if needed)
  - *Finish* (to the final step)



- **Requirements**: *Candidate information* **Locked**; every other block **Hidden**.



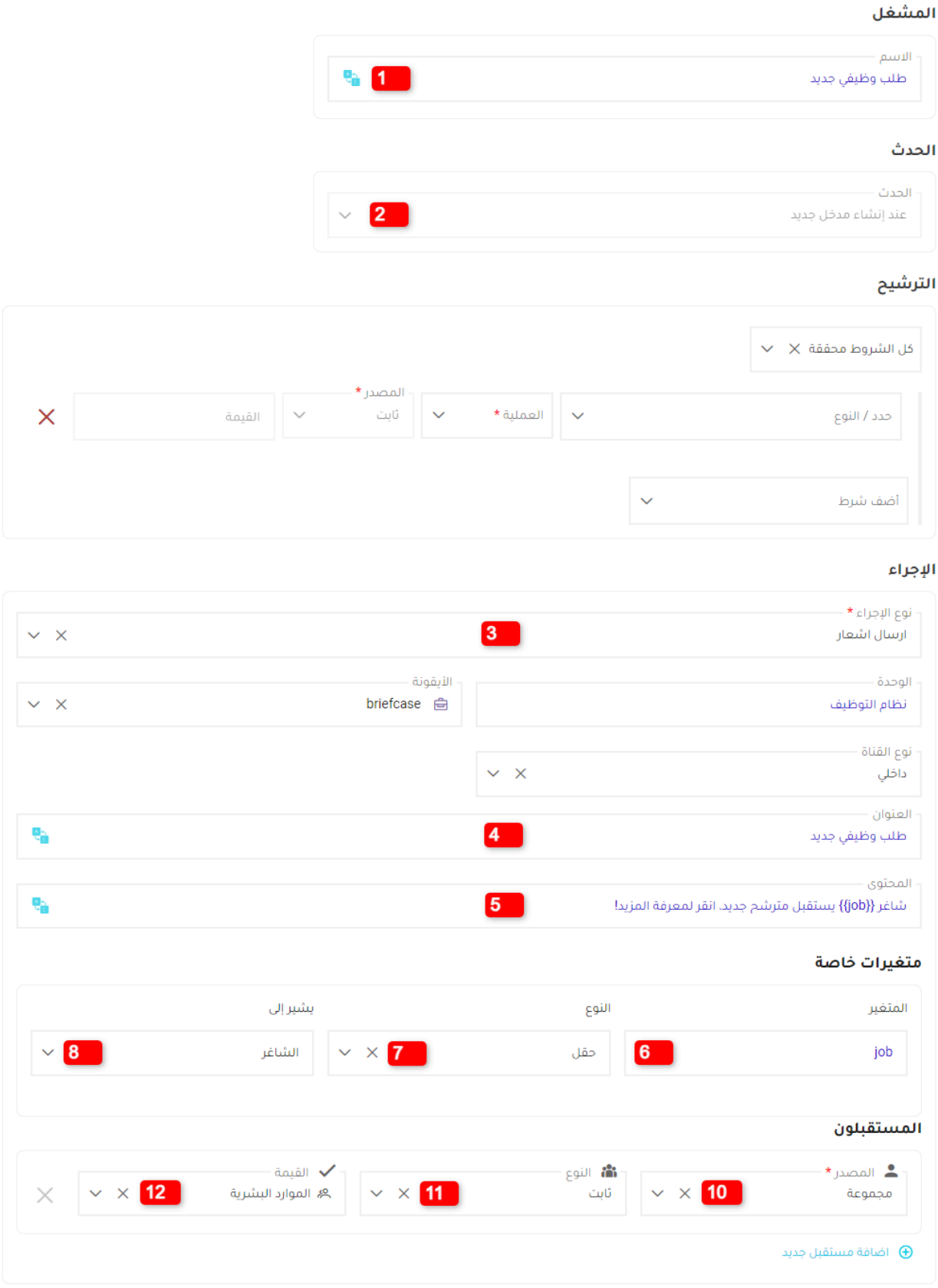


Step 11: Complete Step

Reactions

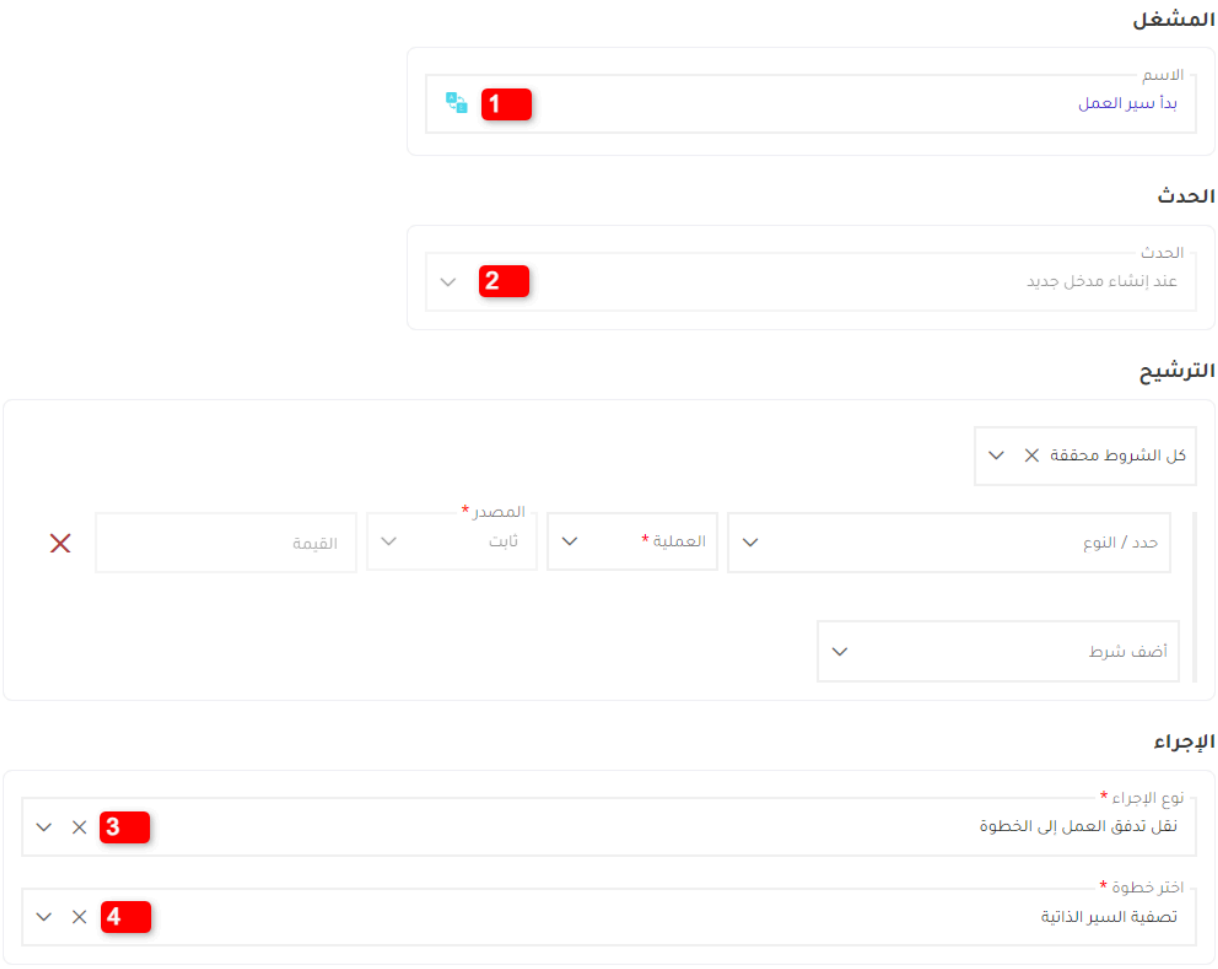
Add these on the service's **Reactions** tab, as for the Jobs entity.

1. **New job application**
    - **Event**: **On Entry Created**
    - **Action**: **Send Notification**
    - **Recipients**: the *HR* group



2. **Start the workflow**

   - **Event**: **On Entry Created**
   - **Action**: **Move Workflow To Step**, *Request processing*



3. **Job offer**

   - **Event**: **On Workflow Step Reached**, *Job offer*
   - **Action**: **Send Email**. **Subject** *Job offer details*. Body, for example: *Dear {{candidate}}, we are pleased to offer you the {{job}} position…*
   - **Custom Variables**: *candidate*, **Field**, *Full name*; *job*, **Field**, *Vacancy*.
   - **Recipients**: **Source Static**, **Type Field**, **Value** the *Email* field

المشغل

الاسم
عرض العمل

1

الحدث

الحدث
عند الوصول لخطوة في مسار العمل

2

اختر خطوة
المدخل الحالي :عرض عمل

3

الترشيح

كل الشروط محققة

حدد / النوع | العملية * | المصدر * ثابت | القيمة

أضف شرط

الإجراء

نوع الإجراء *
إرسال البريد الإلكتروني

4

الموضوع
تفاصيل عرض الوظيفة

5

6

عزيزي {{candidat}}،

نحن سعداء بأن نقدم لك عرض الوظيفة لشغل وظيفة {{job}} لديدا. يشرفنا دعوتك للانضمام إلى فريق عملنا المميز.

نود أن نشير إلى أنك قد قطعت شوطًا طويلًا في عملية التوظيف وأبدعت في كل مرحلة. يعكس اختيارك لقبول هذا العرض ثقتنا الكاملة في قدراتك وإمكانياتك. نحن متحمسون للعمل معك ونعتقد أنك ستكون إضافة قيمة لمؤسستنا.

Normal Sans Serif B I U

متغيرات خاصة

المتغير	النوع	يشير إلى
candidat 7	حقل 8	الاسم الكامل 9
job 10	حقل 11	الشاغر 12

المستقبلون

المستقبل

المصدر * ثابت 13 | النوع متغير 14 | القيمة البريد الالكتروني 15



4. **Add a new employee**

   - **Event**: **On Workflow Step Reached**, *Add employee*
   - **Action**: **Create Data Entry** in the *Employees* entity. Fill its fields from the application with **Value type Field**: *Full name*, *Email*, *Phone number* and *Nationality*.



5. **Reject the candidate**

- **Event**: **On Workflow Step Exited**, with the *Reject* action of *Request processing*, *Interview L1* or *Interview L2*
- **Action**: **Send Email**
- **Recipients**: **Static**, **Field**, the *Email* field

المشغل

الاسم
1 رفض المترشح

الحدث

الحدث
2 عند الخروج من خطوة تدفق عمل

اختر خطوة
3 المدخل الحالي :تصفية السير الذاتية (رفض), المقابلة L1 (رفض), المقابلة L2 (رفض), عرض عمل (إ...

جميع إجراءات الرفض

الترشيح

كل الشروط محققة

حدد / النوع | العملية * | المصدر * ثابت | القيمة

أضف شرط

الإجراء

نوع الإجراء *
4 إرسال البريد الإلكتروني

الموضوع
5 تحديث حول طلبك لوظيفي

6
عزيزي المترشح،

نأمل أن يجد هذا البريد صدى طيب لديك. نود أن نعبر عن امتناننا لاهتمامك بالانضمام إلى فريقنا وعلى الجهد الذي بذلته عند تقديم طلبك لشغل وظيفة {{job}} لدينا. نحن نقدر حقًا جهودك والفرصة التي أعطيتنا للتعرف عليك أكثر.

بعد النظر الدقيق، **نأسف أن نعلمك أن طلبك لشغل الوظيفة لم يتقدم إلى المرحلة التالية** من عملية التوظيف لدينا. **على الرغم من أن مؤهلاتك وخبرتك تبدوان مثيرتين بالفعل**، إلا أننا قررنا المضي قدمًا مع مرشحين آخرين يتداسبون بشكل أفضل مع متطلبات هذا الدور الخاص.

Normal | Sans Serif | B I U

متغيرات خاصة

المتغير	النوع	يشير إلى
7 job	8 حقل	9 الشاغر

المستقبلون

المستقبل

المصدر *	النوع	القيمة
10 ثابت	11 متغير	12 البريد الالكتروني

اضافة مستقبل جديد



6. **Notify the candidate of the L1 interview**
    - **Event**: **On Workflow Step Exited**, *Request processing* with *Schedule interview*
    - **Action**: **Send Email**
    - **Recipients**: **Static**, **Field**, the *Email* field

المشغل

الاسم
1 إشعار المترشح بالمقابلة L1

الحدث

الحدث
2 عند الخروج من خطوة تدفق عمل

اختر خطوة
3 المدخل الحالي :تصفية السير الذاتية (جدولة مقابلة)

الترشيح

كل الشروط محققة

حدد / النوع | العملية * | المصدر * ثابت | القيمة

أضف شرط

الإجراء

نوع الإجراء *
4 إرسال البريد الإلكتروني

الموضوع
5 دعوة للمقابلة الشخصية

6
عزيزي المرشح،

نحن نسعد بإعلامكم بأنك قد **تم اختيارك للمرحلة التالية في عملية التوظيف لشغل وظيفة {{job}} لدينا**. تهانينا على تقديمك لطلب وتميزك في المرحلة الأولى.

سيكون لديك فرصة للمقابلة الشخصية معنا والتعرف على فريق العمل لدينا. إليك تفاصيل المقابلة:

- **تاريخ المقابلة: {{date}}**

Normal Sans Serif B I U

متغيرات خاصة

المتغير	النوع	يشير إلى
7 job	حقل	الشاغر
8 date	حقل	موعد المقابلة
9 link	حقل	رابط المقابلة

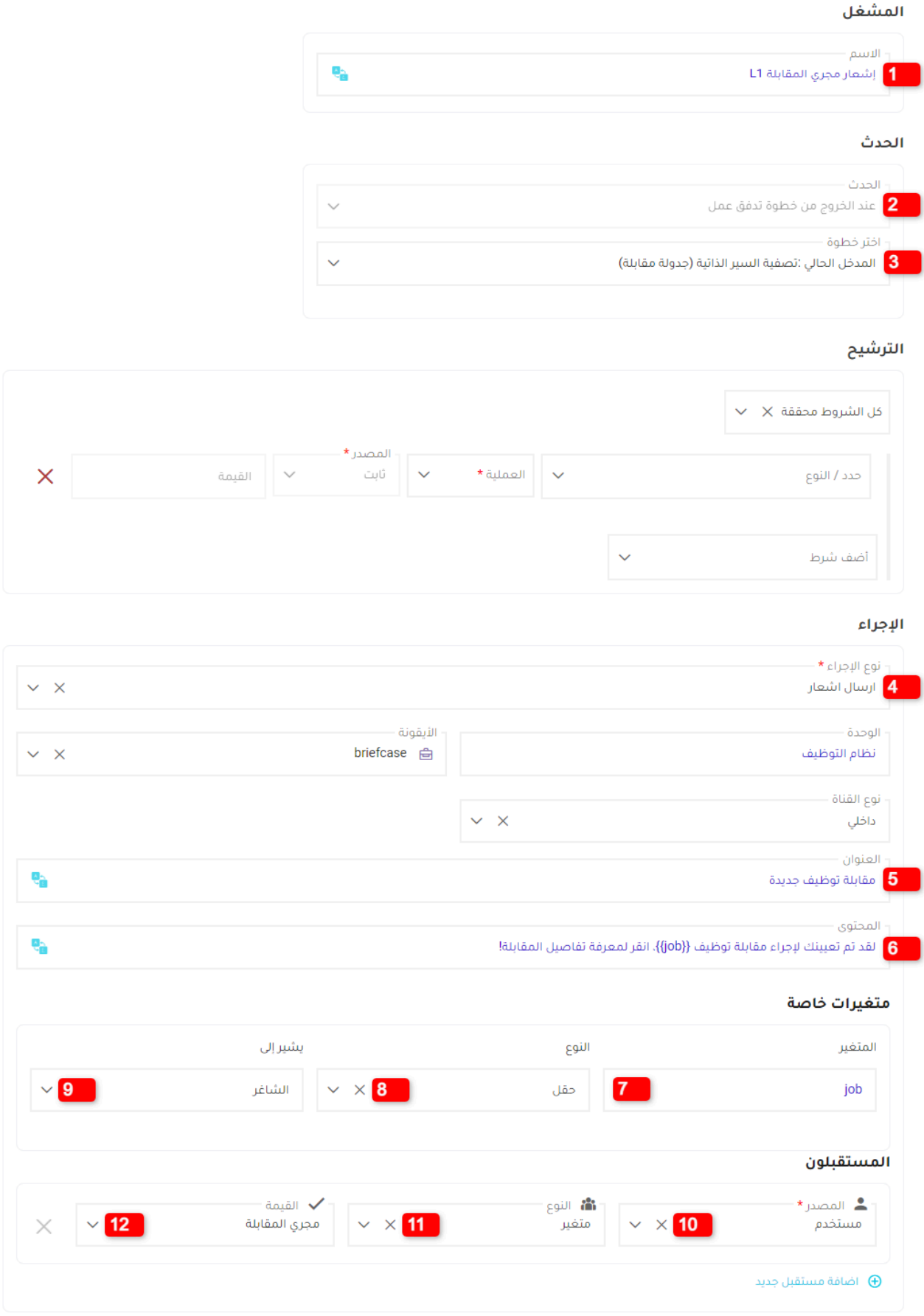
المستقبلون

المستقبل

المصدر * | النوع | القيمة



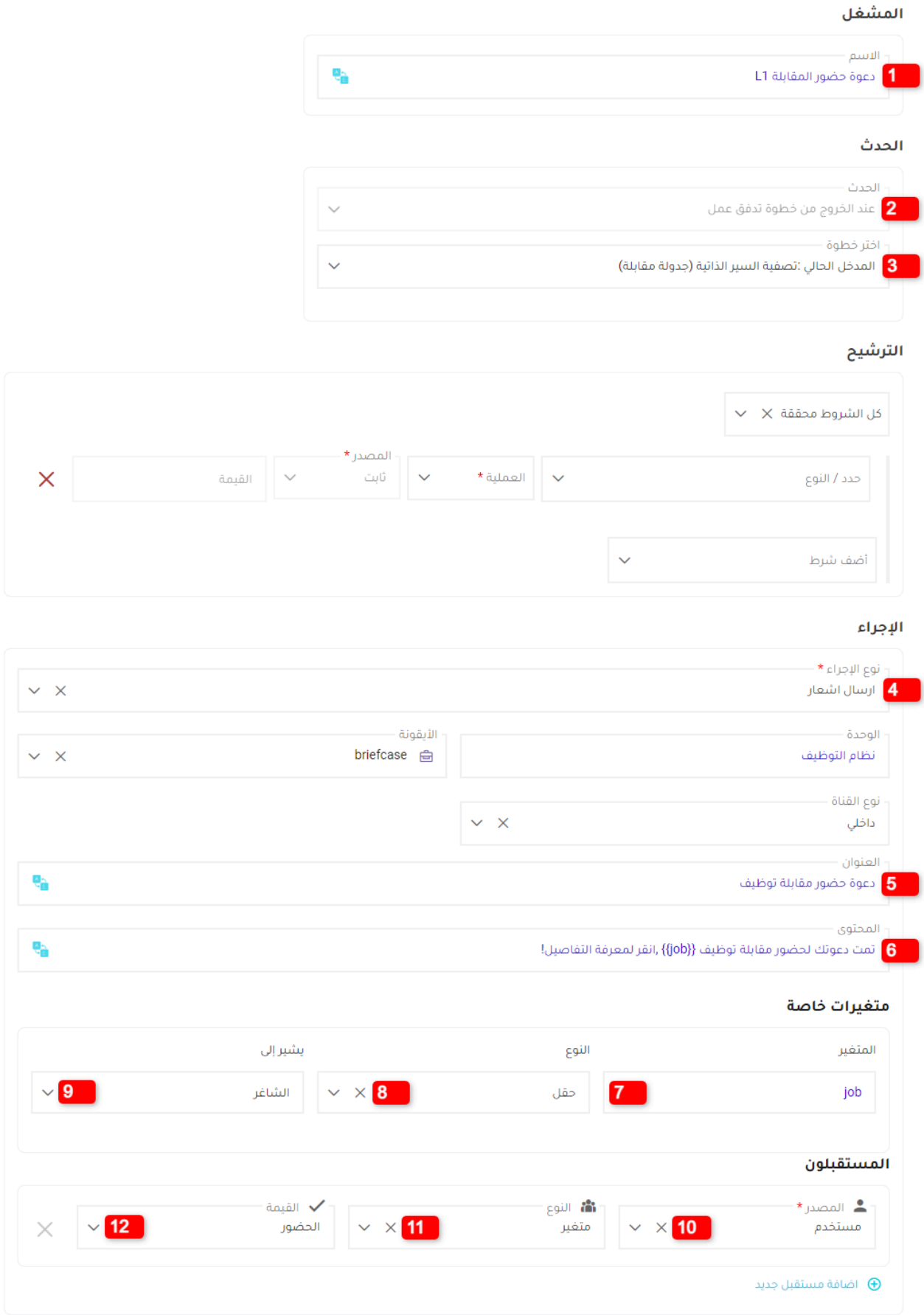
7. **Notify the L1 interviewer**
    - **Event**: **On Workflow Step Exited**, *Request processing* with *Schedule interview*
    - **Action**: **Send Notification**
    - **Recipients**: **Source User**, **Type Field**, **Value** the *Interviewer* field



8. **Invite the attendees to the L1 interview**

   - **Event**: **On Workflow Step Exited**, *Request processing* with *Schedule interview*

- **Action**: **Send Notification**
- **Recipients**: **User**, **Field**, the *Attendees* field



9. **Assign the first task**
    - **Event**: **On Workflow Step Exited**, *Interview L1* with *Assign task*
    - **Action**: **Send Email**
    - **Recipients**: **Static**, **Field**, the *Email* field

المشغل

الاسم
1 تعيين مهمة أولى

الحدث

الحدث
2 عند الخروج من خطوة تدفق عمل

اختر خطوة
3 المدخل الحالي :المقابلة L1 (تعيين مهمة)

الترشيح

كل الشروط محققة

حدد / النوع | العملية * | المصدر * ثابت | القيمة

أضف شرط

الإجراء

نوع الإجراء *
4 إرسال البريد الإلكتروني

الموضوع
5 مهمة الوظيفة الأولى

6
عزيزي المرشح،

نحن نتطلع إلى البدء في عملية التوظيف معك، ونسعد بإعلامك بمهمتك الأولى كجزء من عملية التقييم.

يرجى النقر على الرابط أدناه لعرض تفاصيل المهمة وتحميل الملفات المطلوبة

من خلال الرابط، يمكنك مشاهدة تفاصيل المهمة وقراءة المتطلبات بعناية. بمجرد الانتهاء من تنفيذ المهمة، يمكنك تحميل الملفات المطلوبة وإكمالها.

Normal Sans Serif B I U

متغيرات خاصة

المتغير	النوع	يشير إلى
7 deadline	8 حقل	9 تاريخ الاستحقاق

المستقبلون

المستقبل

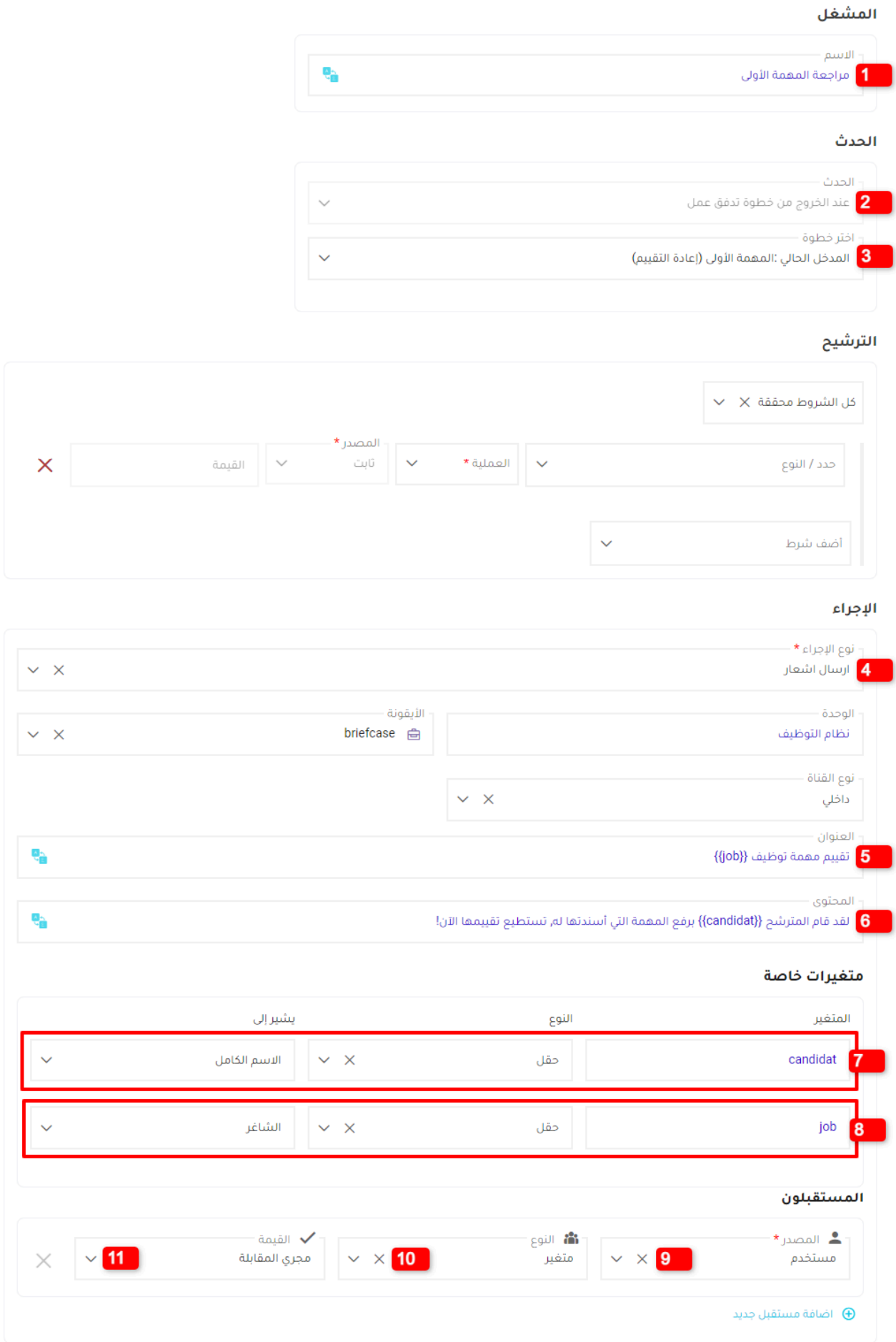
المصدر * ثابت 10 | النوع متغير 11 | القيمة البريد الالكتروني 12

اضافة مستقبل جديد

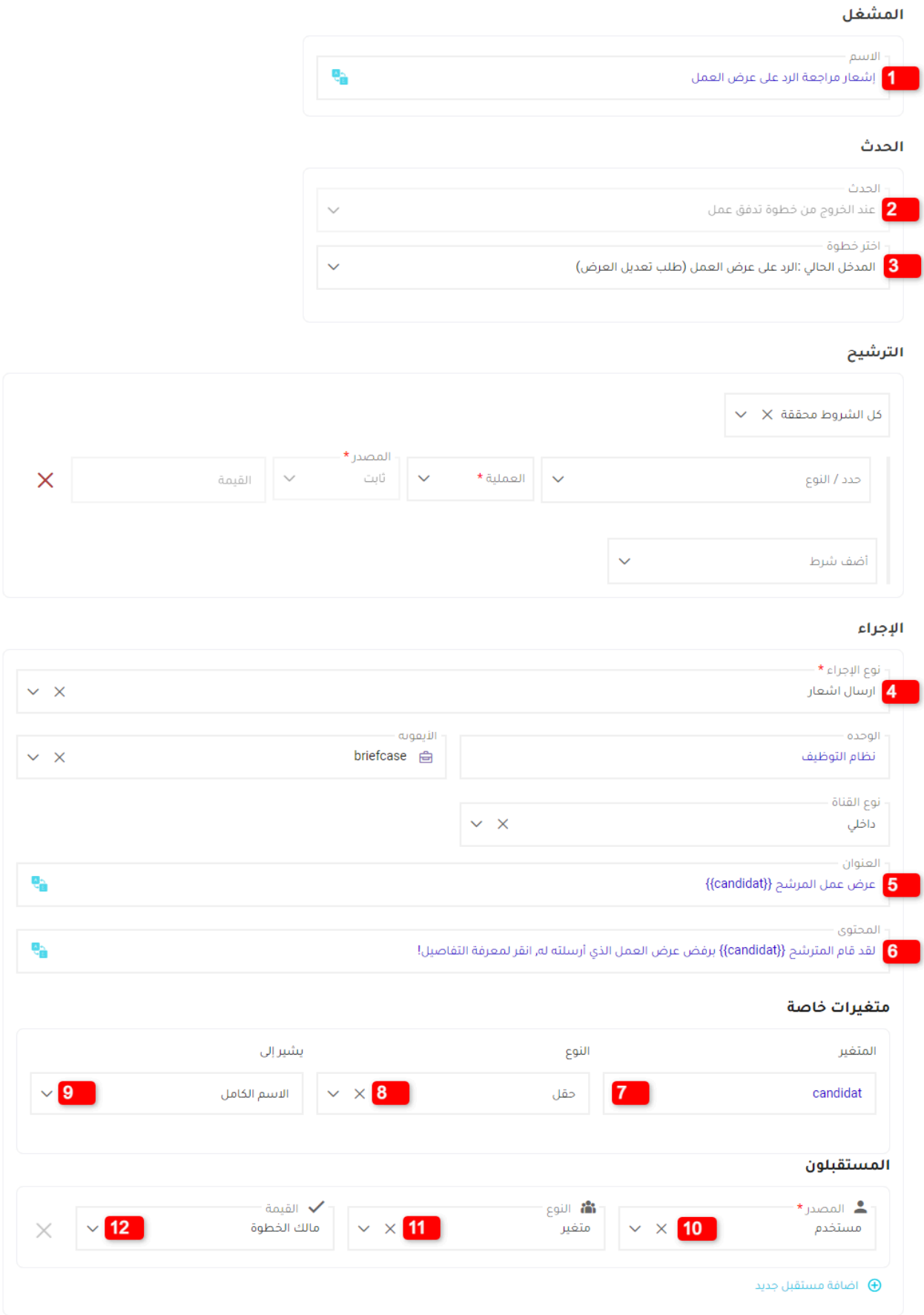
نسخة إلى



10. **Review the first task**
    - **Event**: **On Workflow Step Exited**, *First task* with *Re-evaluate*
    - **Action**: **Send Notification**
    - **Recipients**: **User**, **Field**, the *Interviewer* field


المشغل
الاسم
مراجعة المهمة الأولى
1
الحدث
الحدث
عند الخروج من خطوة تدفق عمل
2
اختر خطوة
المدخل الحالي :المهمة الأولى (إعادة التقييم)
3
الترشيح
كل الشروط محققة
حدد / النوع
العملية *
المصدر *
ثابت
القيمة
أضف شرط
الإجراء
نوع الإجراء *
ارسال اشعار
4
الوحدة
نظام التوظيف
الأيقونة
briefcase
نوع القناة
داخلي
العنوان
تقييم مهمة توظيف {{job}}
5
المحتوى
لقد قام المترشح {{candidat}} برفع المهمة التي أسندتها له, تستطيع تقييمها الآن!
6
متغيرات خاصة
المتغير
النوع
يشير إلى
candidat
7
حقل
الاسم الكامل
job
8
حقل
الشاغر
المستقبلون
المصدر *
مستخدم
9
النوع
متغير
10
القيمة
مجري المقابلة
11
اضافة مستقبل جديد

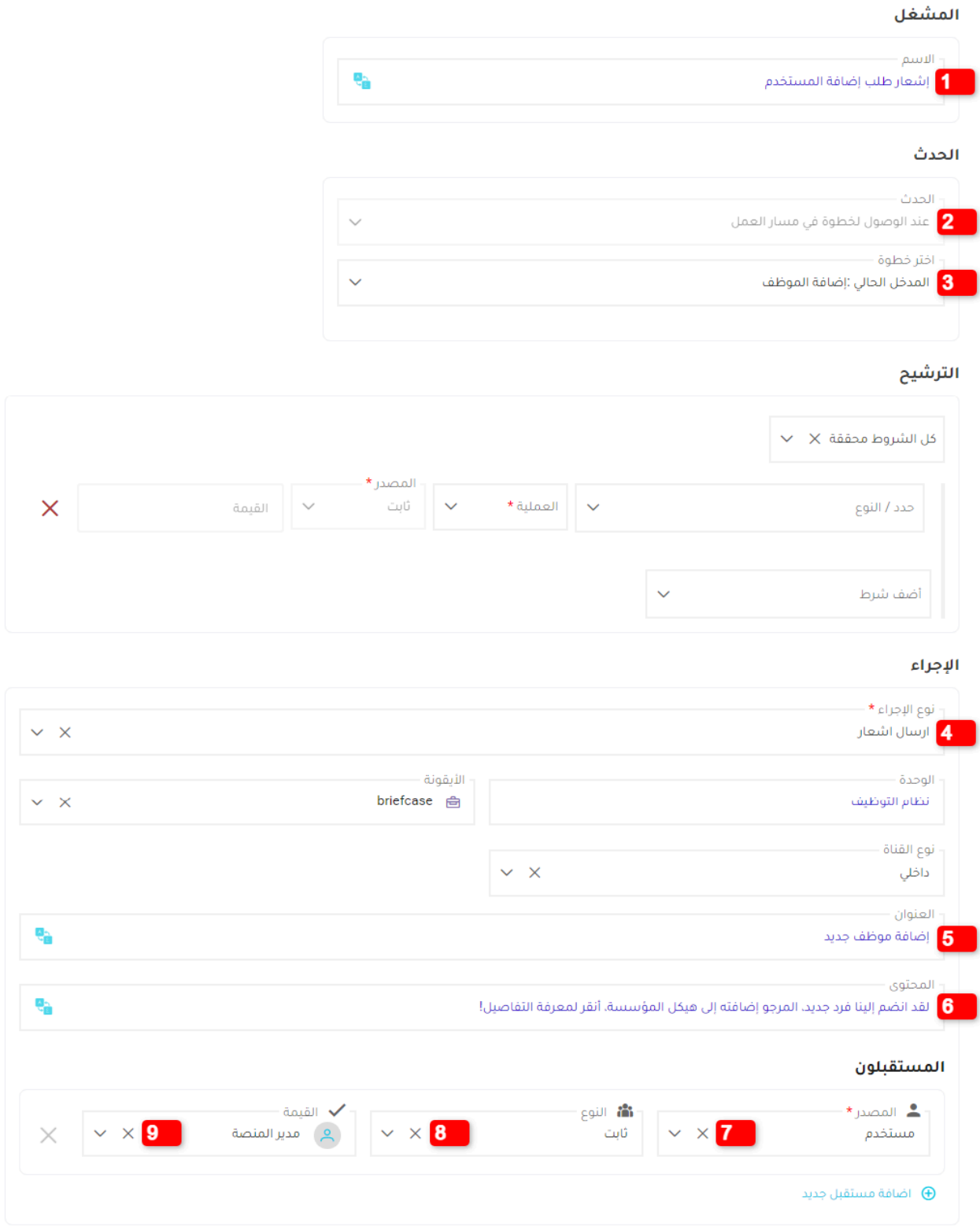
11. **Job offer response review**
    - **Event**: **On Workflow Step Exited**, *Respond to job offer* with *Request offer changes*
    - **Action**: **Send Notification**
    - **Recipients**: **User**, **Field**, the step owner



12. **Add employee notification**

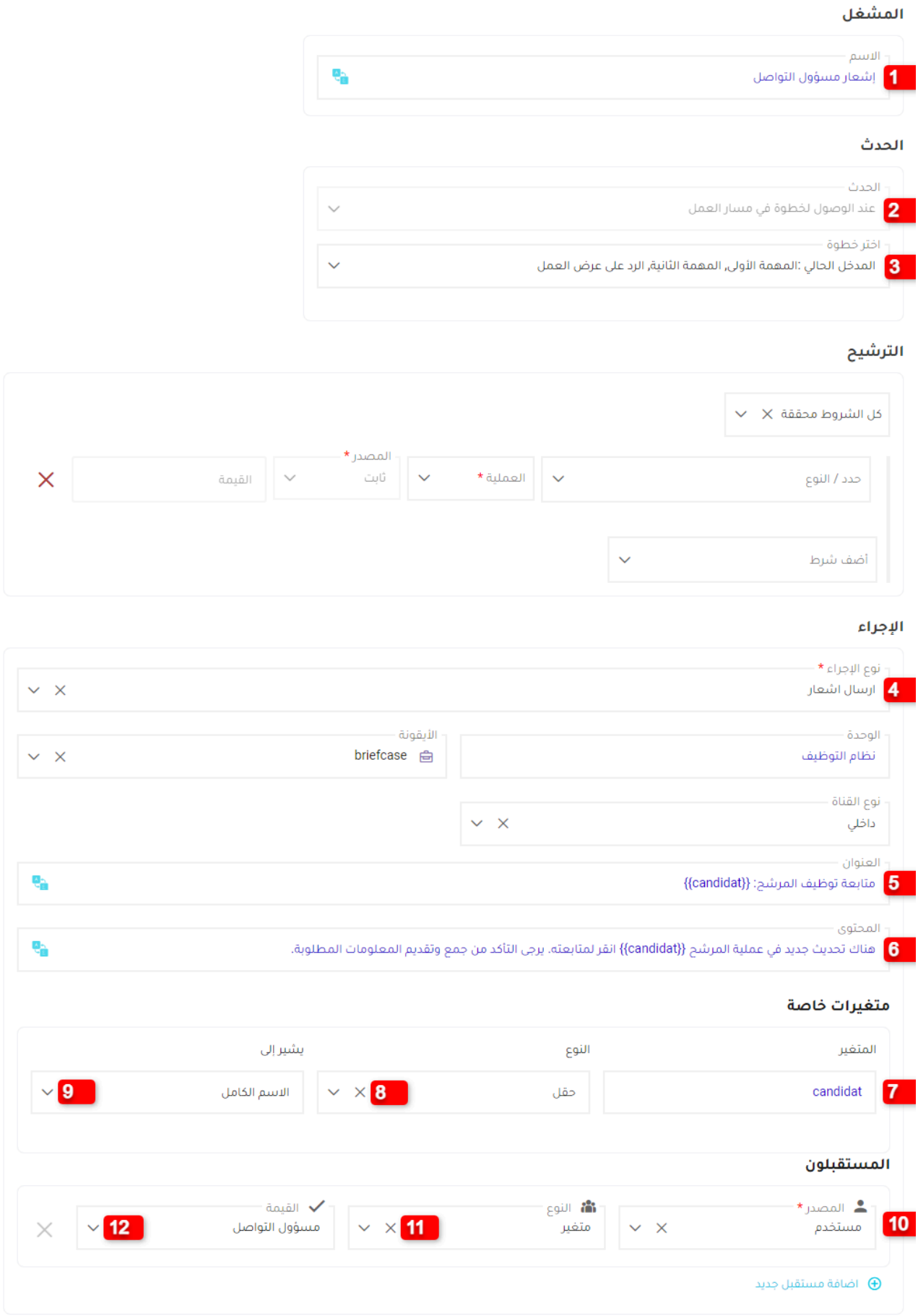
   - **Event**: **On Workflow Step Reached**, *Add employee*

- **Action**: **Send Notification**
- **Recipients**: **User**, **Fixed**, the platform administrator



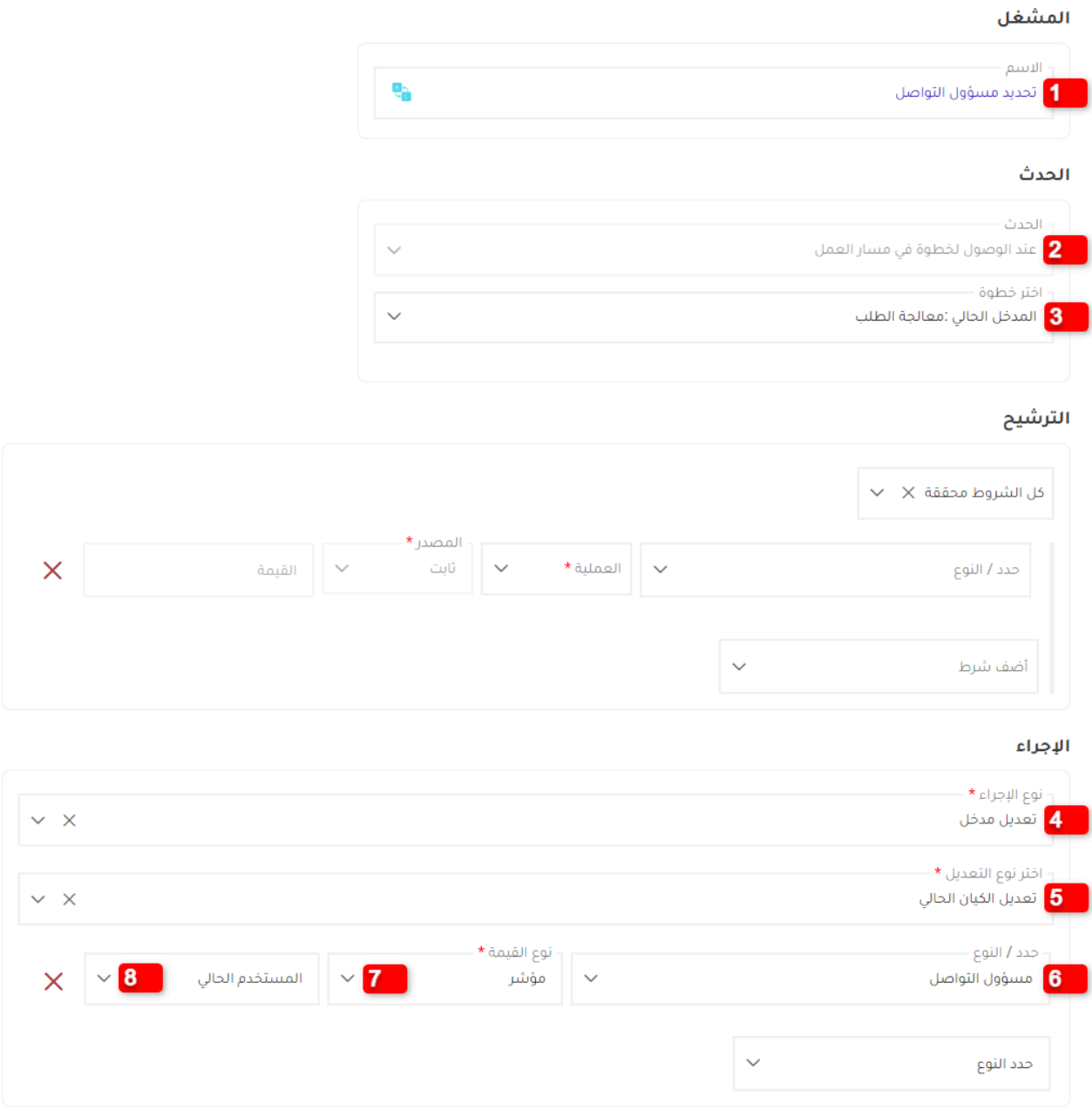
13. **Set the contact person**

- **Event**: **On Workflow Step Reached**, *Request processing*
- **Action**: **Update Data Entry**, **Choose update mode Self**: set *Contact person* with **Value type Pointer** and **Current User**



14. **Notify the contact person**

- **Event**: **On Workflow Step Reached**, *First task*, *Second task* and *Respond to job offer*
- **Action**: **Send Notification**. **Title** *Candidate follow-up: {{candidate}}*; **Content** *There is an update on {{candidate}}'s application. Select it to follow up and collect what is needed.*
- **Recipients**: **User**, **Field**, the *Contact person* field



Notes:

- Repeat the reactions of *Interview L1* and *First task* for *Interview L2* and *Second task*.
- Copy the contact person on every email sent to the candidate: add the *Contact person* field under **CC**, as shown below.



Save and activate the service

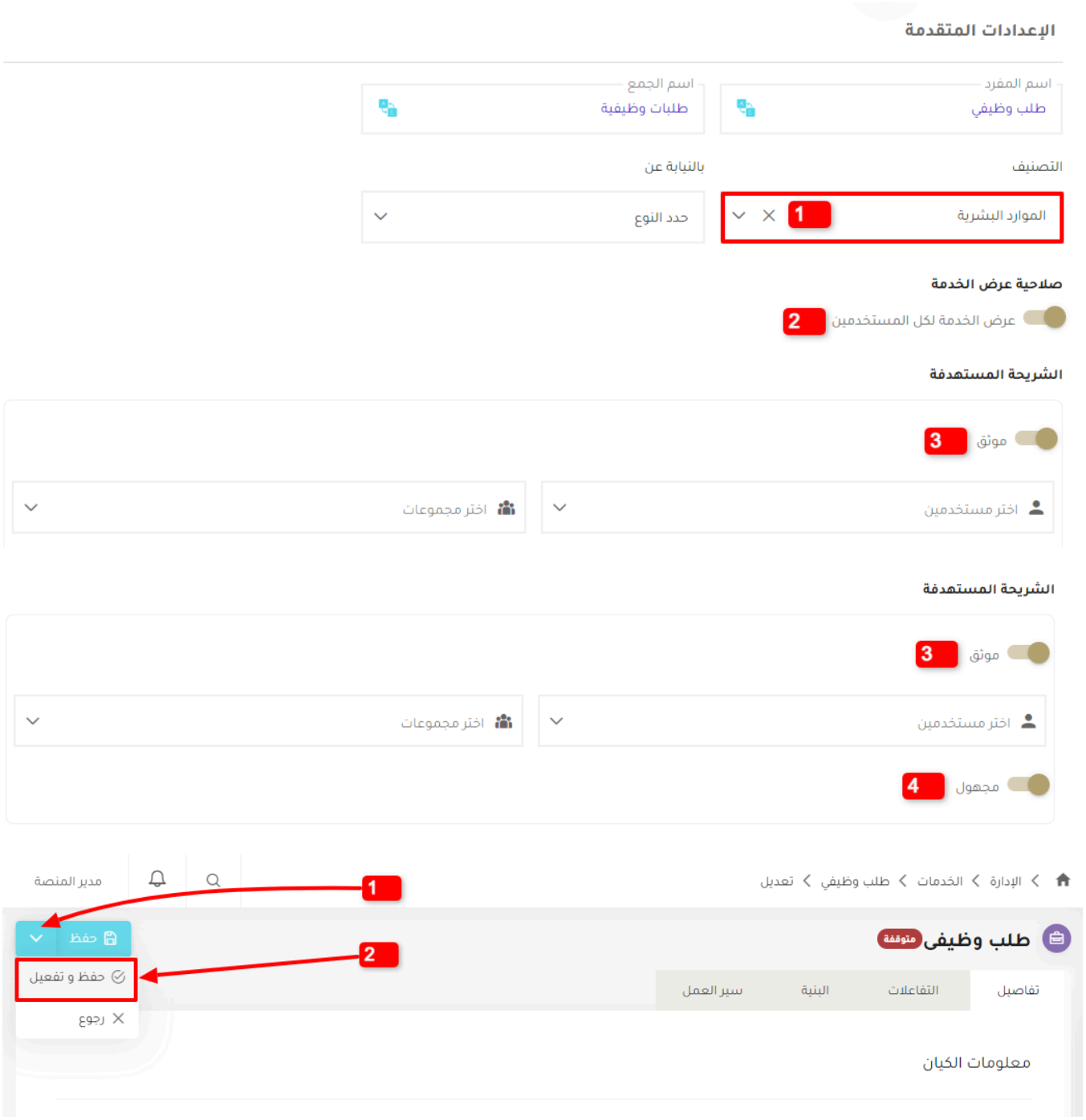
To open the service to candidates:

1. Open the **Details** tab of the service designer.
2. In **Category**, choose a service category. A service cannot be activated without one; ask your tenant administrator if the list is empty.
3. Under **Audience**, turn on **Anonymous**, so candidates can apply without an account. **Anonymous** cannot be combined with picking specific users or groups.
4. Open the actions menu next to **Save** at the top of the page.
5. Select **Save and Activate**, and confirm.

The service is saved and active.

Protect a public form

Because anyone can submit this service without signing in, turn on **Human Verification** for its form to keep out automated and spam submissions. See Human verification [Administration › Human verification (bot protection)]. For public views, the Filter-first option makes visitors fill in a search form before any results load.



Go deeper

The general recipe, with every owner and audience option: Publish a service with an approval workflow · Workflows & approvals · For integrators: Workflow model and API reference [Integrate & Extend]

Creating Queries

A query is a saved filter. The views, dashboard summaries and report of the recruitment system are built on the two queries you create here.

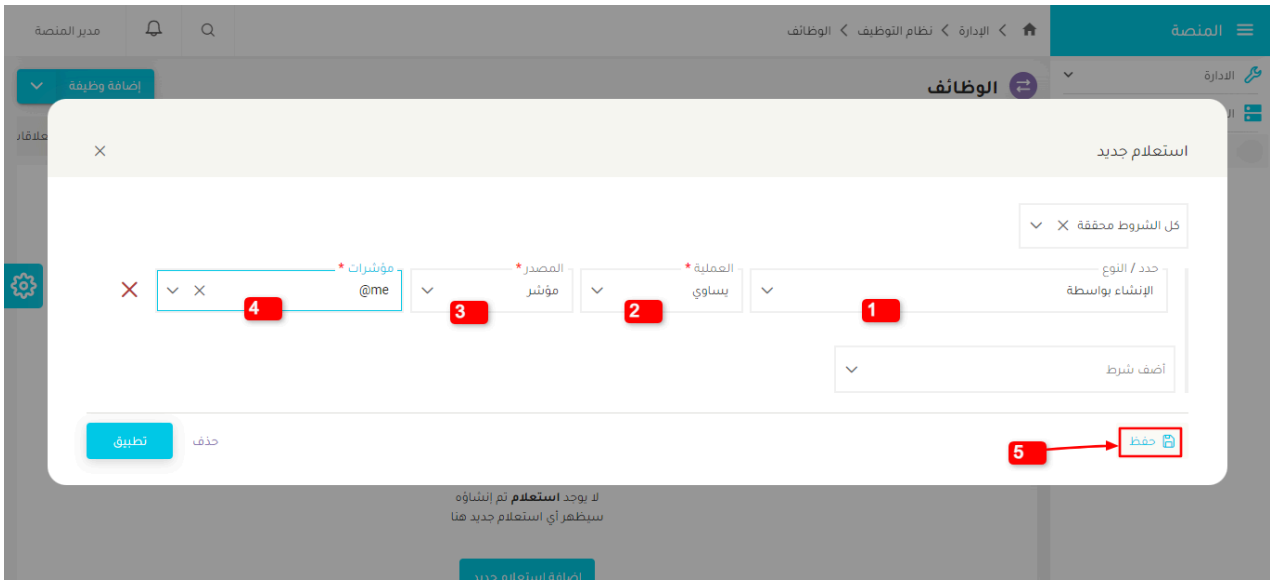
For more about queries, see Queries.

1. Open the *Jobs* entity and its **Queries** tab.

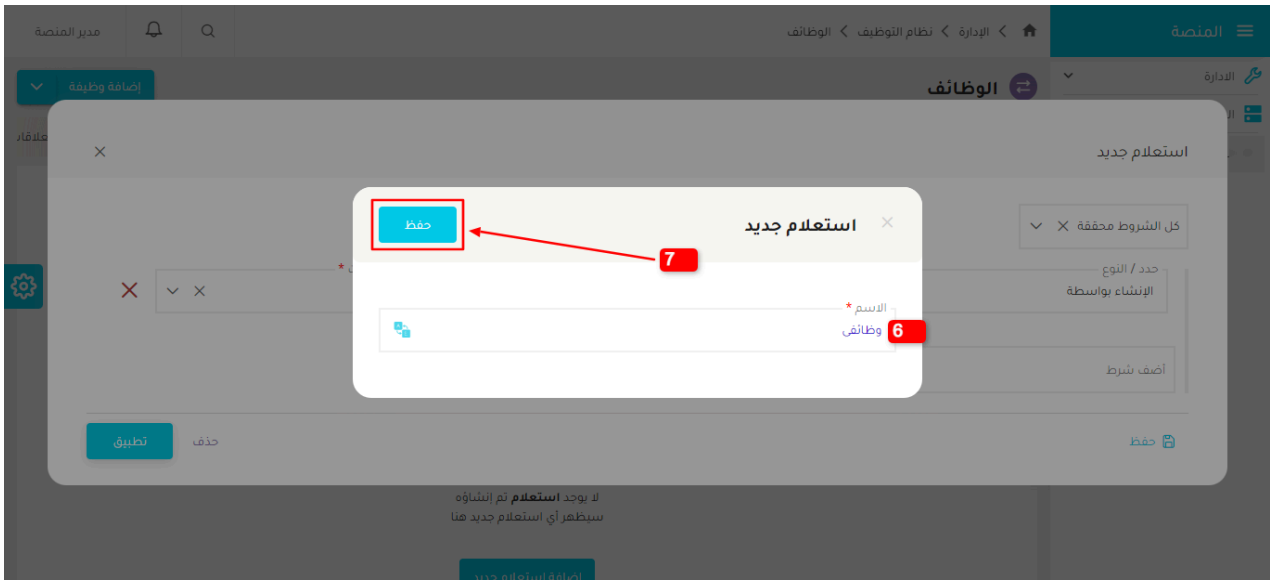
2. Select the **+** next to **New Query**.



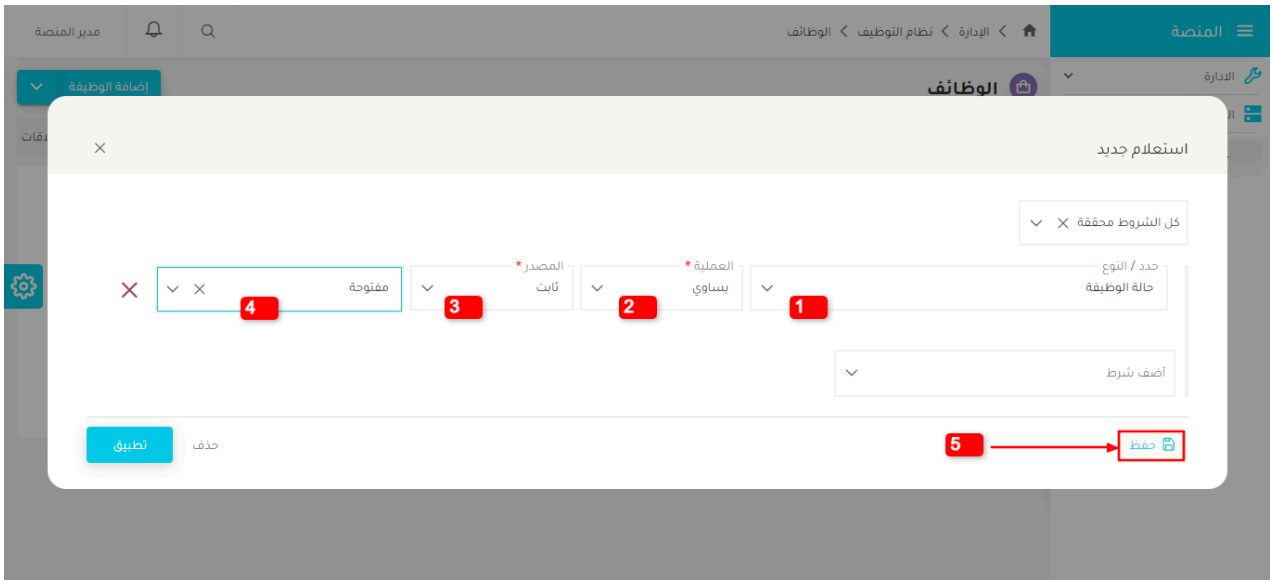
- The **New Query** dialog opens. Fill in the condition:
- **My jobs** (the jobs the signed-in user created)
  - Field: choose *Created by* from the entry's attributes.
  - **Operator**: **Equal**.
  - **Source**: **Pointer**.
  - **Pointers**: `@me`, the signed-in user.

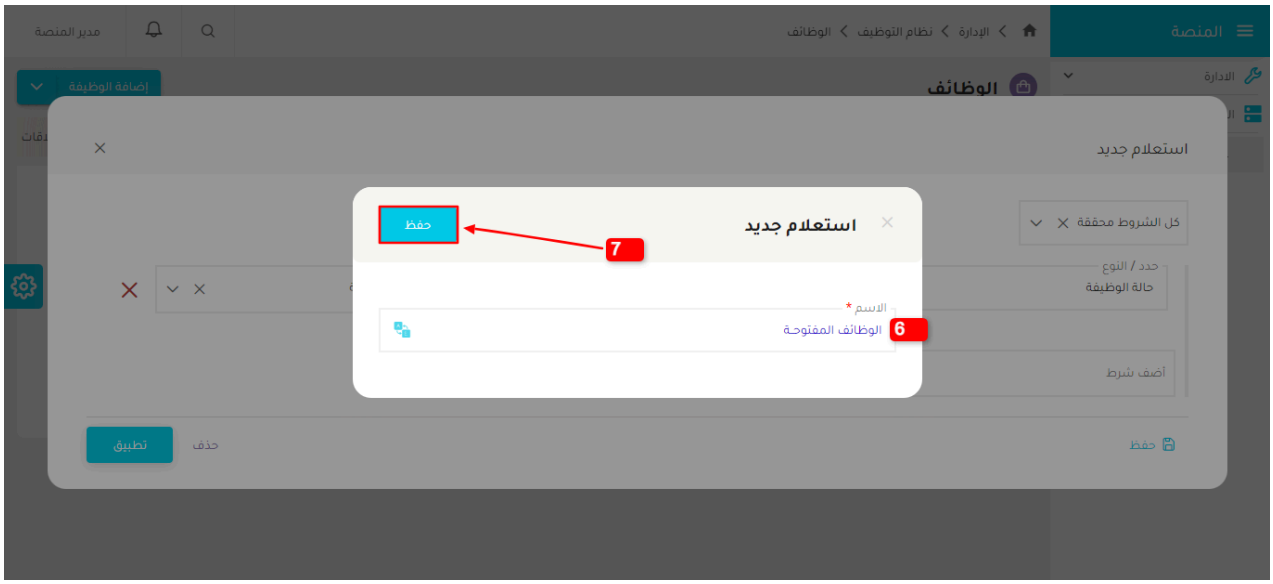


- Select **Save**. A second dialog asks for the query's name: enter *My jobs*.
- Select **Save**. The query is saved.



- Add the second query the same way:
- **Open jobs** (the jobs that are still open)
  - Field: *Job status*. **Operator**: **Equal**. **Source**: **Fixed**. Value: *Open*.
  - Name it *Open jobs*.


المنصة
الإدارة
نظام التوظيف
الوظائف
مدير المنصة
إضافة الوظيفة
الوظائف
استعلام جديد
كل الشروط محققة
حدد / النوع
حالة الوظيفة
العملية *
يساوي
المصدر *
ثابت
مفتوحة
أضف شرط
تطبيق
حذف
حفظ
1
2
3
4
5


المنصة
الإدارة
نظام التوظيف
الوظائف
مدير المنصة
إضافة الوظيفة
الوظائف
استعلام جديد
كل الشروط محققة
حدد / النوع
حالة الوظيفة
أضف شرط
تطبيق
حذف
حفظ
استعلام جديد
حفظ
الاسم *
الوظائف المفتوحة
6
7

Entities

The recruitment system stores its data in three entities. Build **Jobs** first: its page walks through every screen in detail, and the **Employees** and **Receptions** pages reuse the same steps.

Employees Entity
→

Receptions Entity
→

Jobs Entity
→

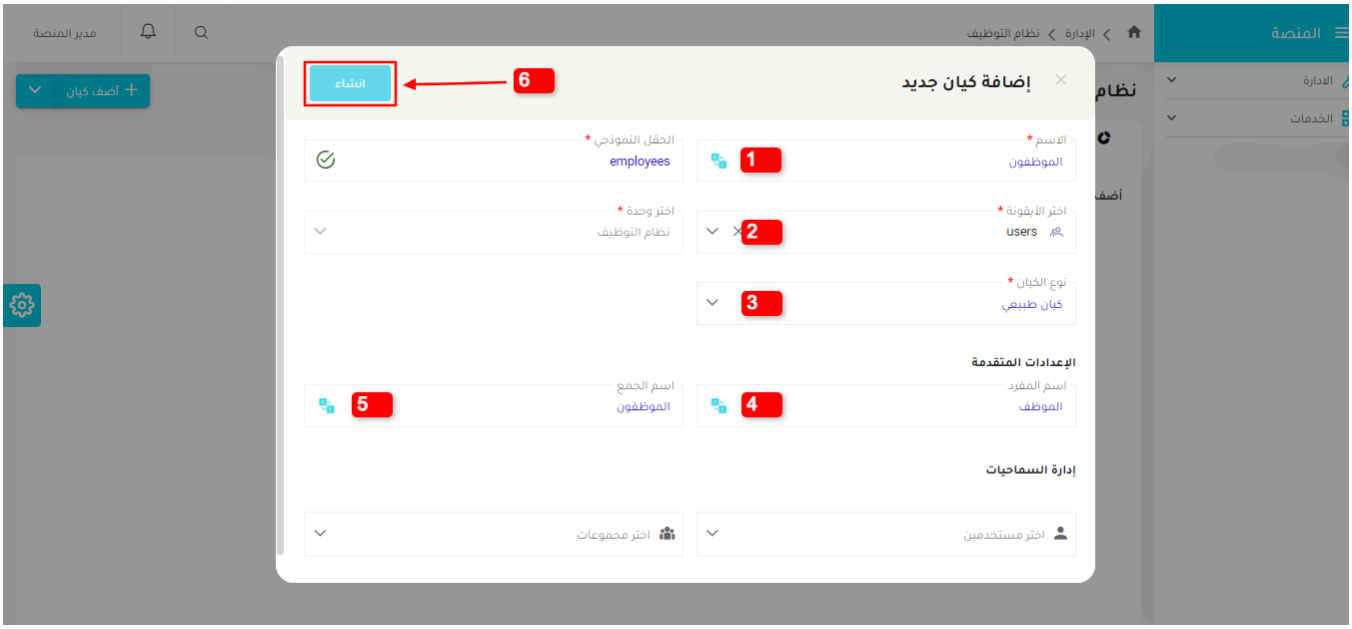
Employees Entity

The Employees entity keeps the record of each new employee. The job application service creates the record for you when a candidate signs their contract.

- **Note:** this page gives only what is specific to the Employees entity: its settings, structure, workflow and reactions. For the step-by-step screens, follow the Jobs entity page.

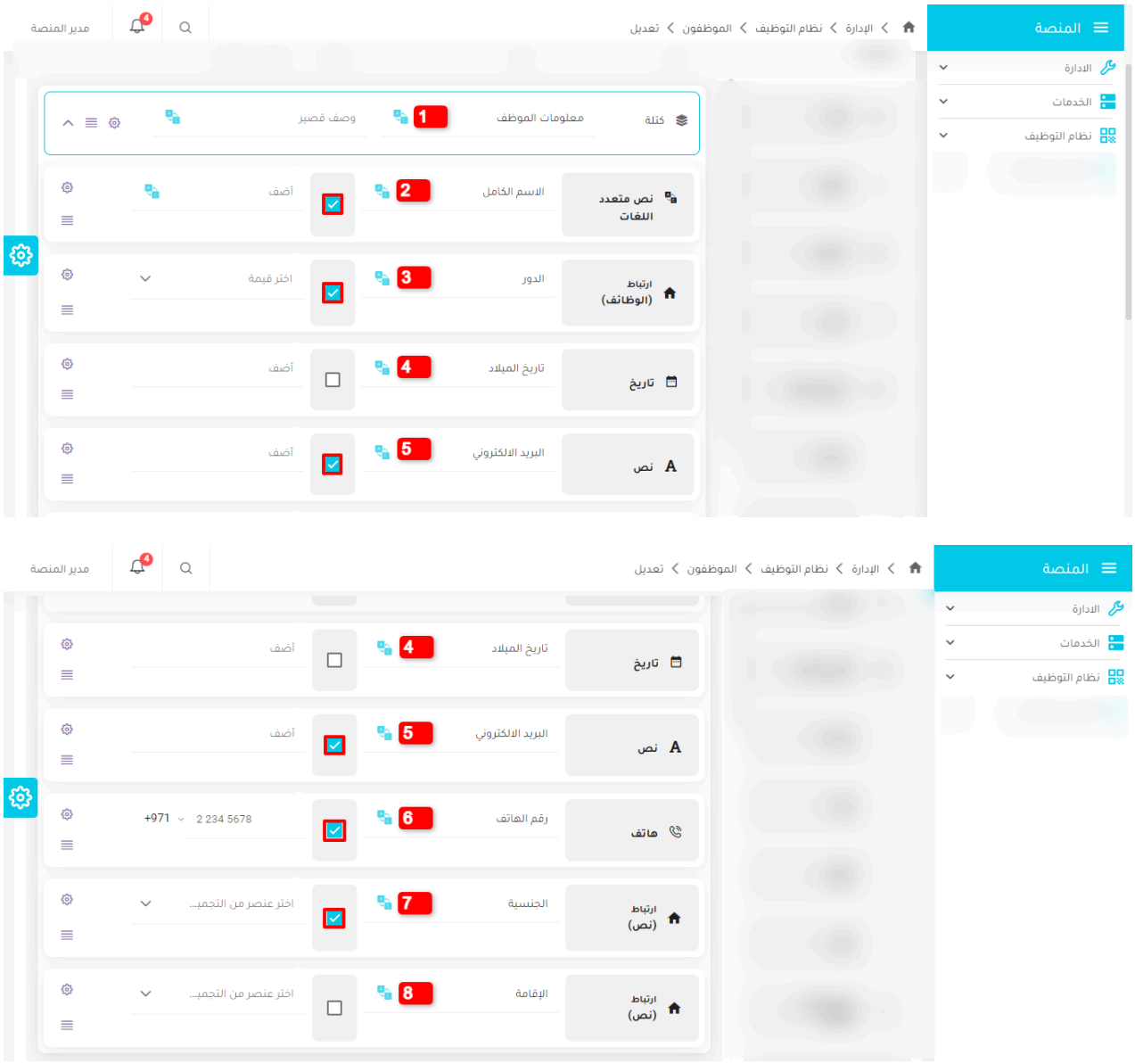
Create the Employees entity

- **Name**: *Employees*, in Arabic and English.
- **Select Icon**: pick an icon for the entity.
- **Data Entity Type**: **Normal Entity**.
- Under **Field Settings**: **Single Name** *Employee* and **Plural Name** *Employees*.



Structure

1. **Employee information block:**
    - **Full name**: required, **Localized Text**.
    - **Date of birth**: optional, **Date**.
    - **Email**: required, **Text**.
    - **Phone number**: required, **Phone**.
    - **Nationality**: required, **Reference** to the *Nationalities* list.
    - **Residence**: optional, **Reference** to the *Residence* list.
    - **Education**: optional, **Reference** to the *Educational qualifications* list.

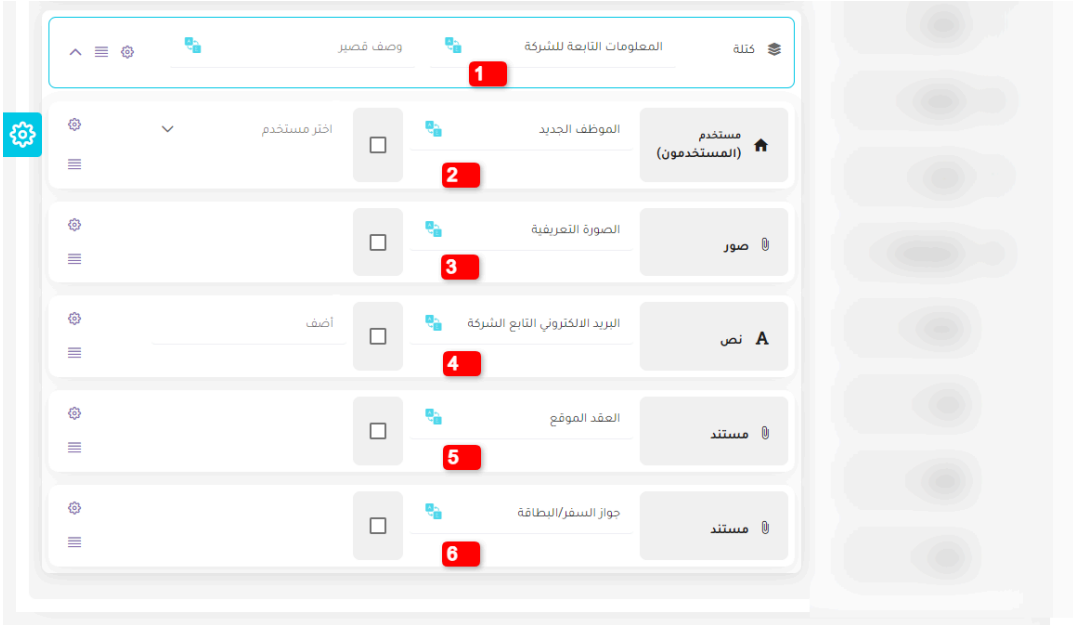


- Open the block's settings, set it to show its fields in two columns, and turn on **Display as tab**:



2. **Company information block:**

- **Company email**: optional, **Text**.
- **Profile photo**: optional, **Image**.
- **Signed contract**: optional, **File**.
- **Passport / ID card**: optional, **File**.



- Open the block's settings, set it to show its fields in two columns, and turn on **Display as tab**:



3. Select **Save** at the top of the page to keep your changes.

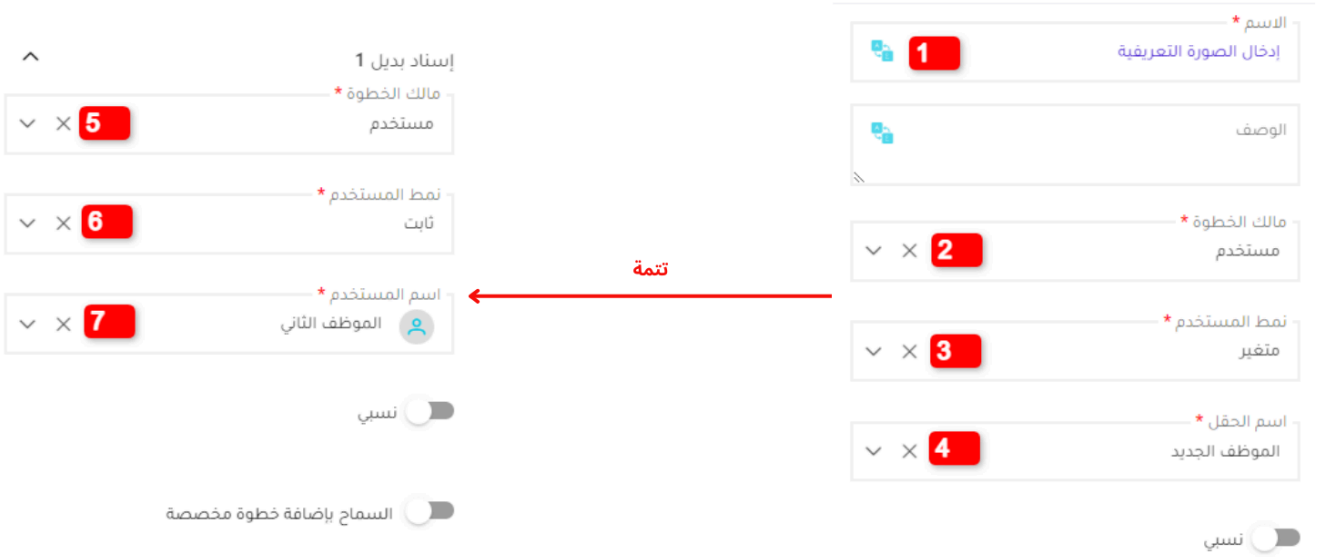
Workflow

Step 1: Initial Step

- **Name**: *Initial Step*
- **Actions**:
  - *Start* (to the next step)

Step 2: Upload profile photo

- **Name**: *Upload profile photo*
- **Owner**: the new employee. The screenshot sets this with **Step Owner User**, **User Type Field**, and a user field named *New employee*, with a fixed fallback user. The structure above has no such field: add a **User** field for the employee's account if you want this step to go to them.
- **Actions**:
  - *Finish* (to the final step)



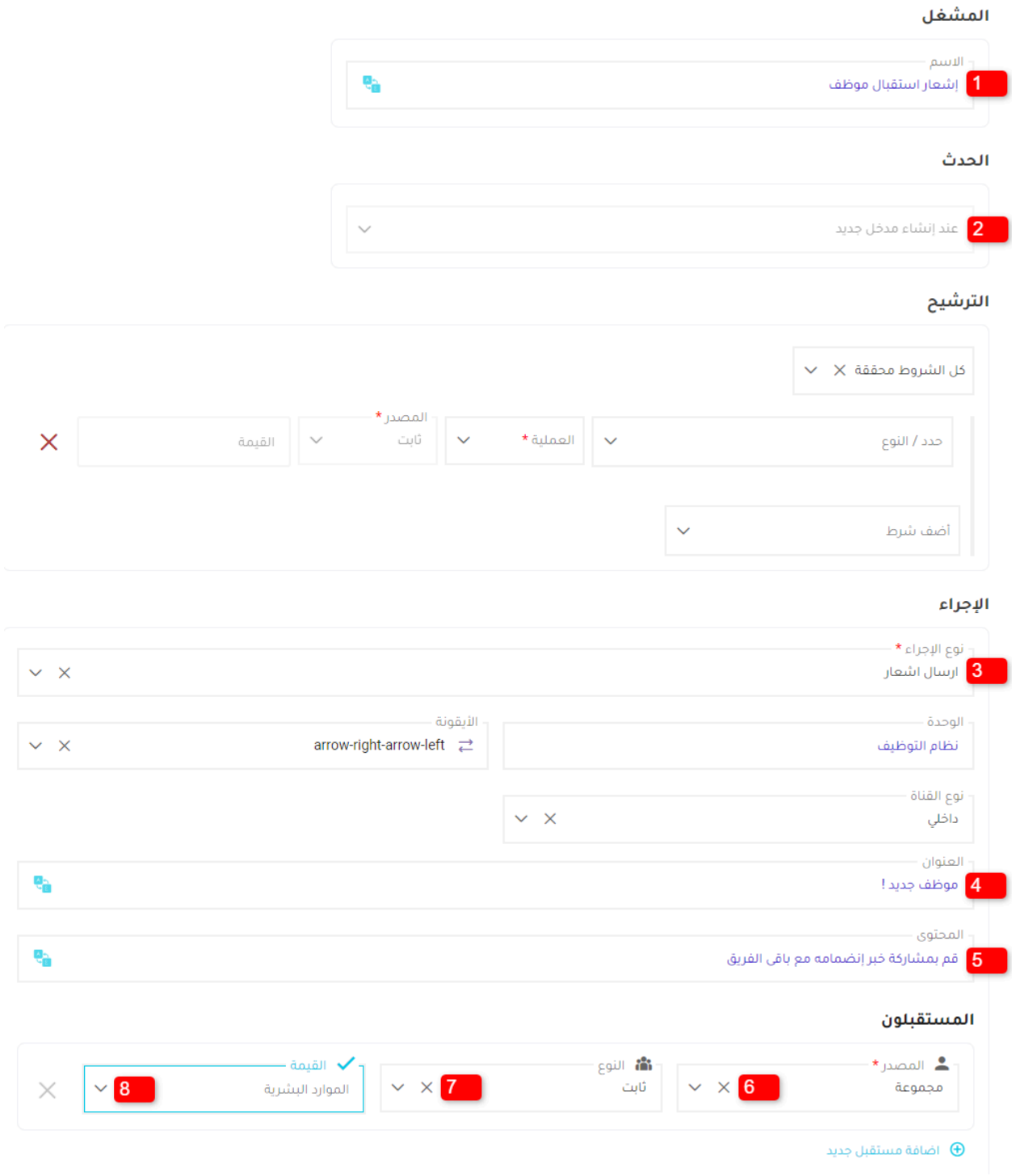


Step 3: Complete Step

Reactions

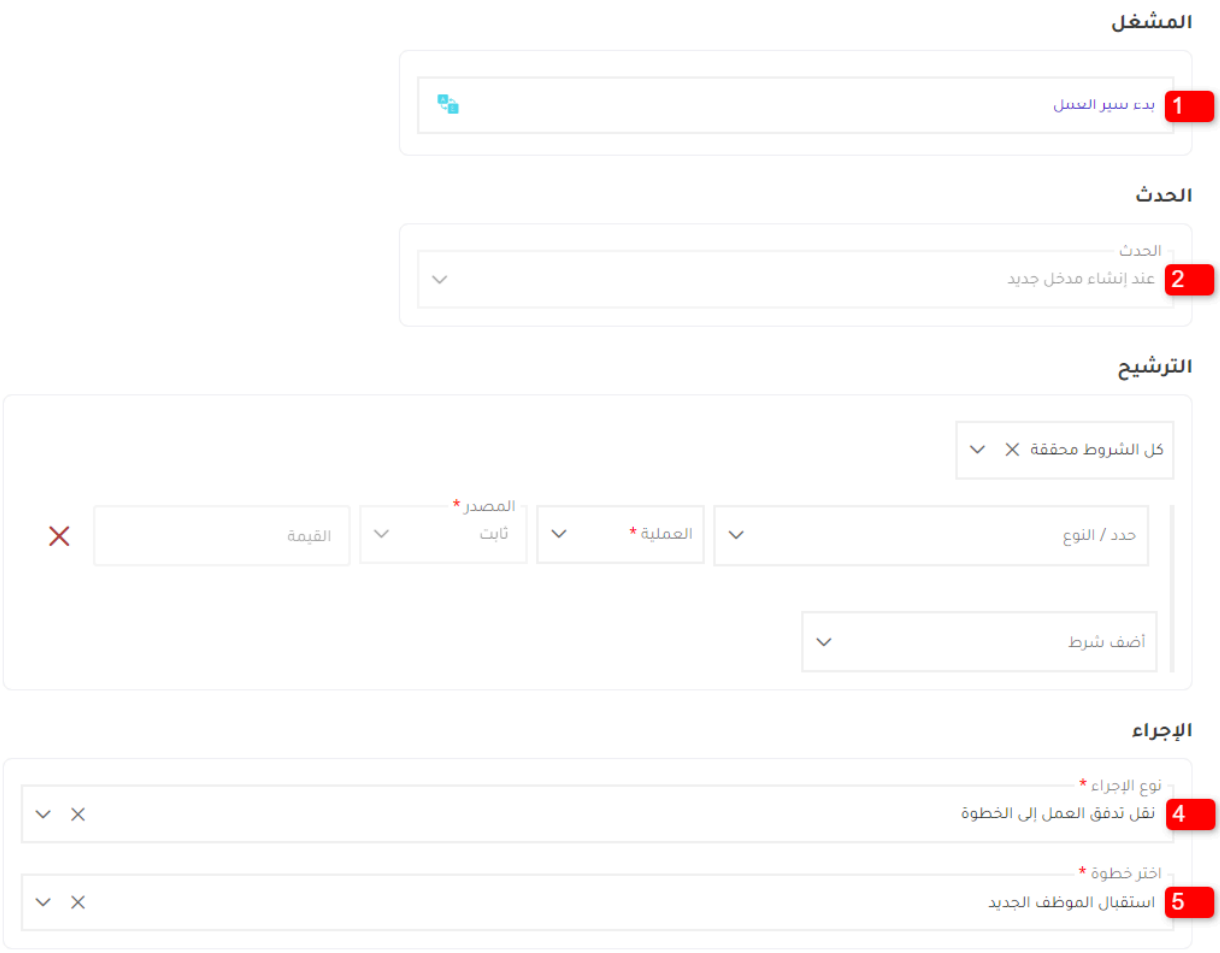
1. **New employee notification**
   - **Event**: **On Entry Created**
   - **Action**: **Send Notification**

- **Recipients**: the *HR* group



2. **Start the workflow**

- **Event**: **On Entry Created**
- **Action**: **Move Workflow To Step**
- **Step**: step 2, *Upload profile photo*



3. **Tell the rest of the team about the new employee**

   - **Event**: **On Entry Created**
   - **Action**: **Send Notification**, for example *A new colleague has joined us!*
   - **Recipients**: the *HR* group. The screenshot shows a variant that runs when the profile-photo step is finished and notifies a list of all users.

المشغل

الاسم
1 إعلام باقي الفريق بانضمام الموظف الجديد

الحدث

الحدث
2 عند الخروج من خطوة تدفق عمل

اختر خطوة
3 المدخل الحالي :استقبال الموظف الجديد (طلب الصورة التعريفية)

الترشيح

كل الشروط محققة

حدد / النوع | العملية * | المصدر * ثابت | القيمة

أضف شرط

الإجراء

نوع الإجراء *
4 ارسال اشعار

الوحدة
نظام التوظيف

الأيقونة
arrow-right-arrow-left

نوع القناة
داخلي

العنوان
5 فرد جديد انضم إلينا!

المحتوى
6 ينضم إلينا فرد جديد داخل الشركة، قم بتهنئة زميلك الجديد

المستقبلون

المصدر * قائمة 7 | النوع ثابت 8 | القيمة المستخدمون 9

اضافة مستقبل جديد

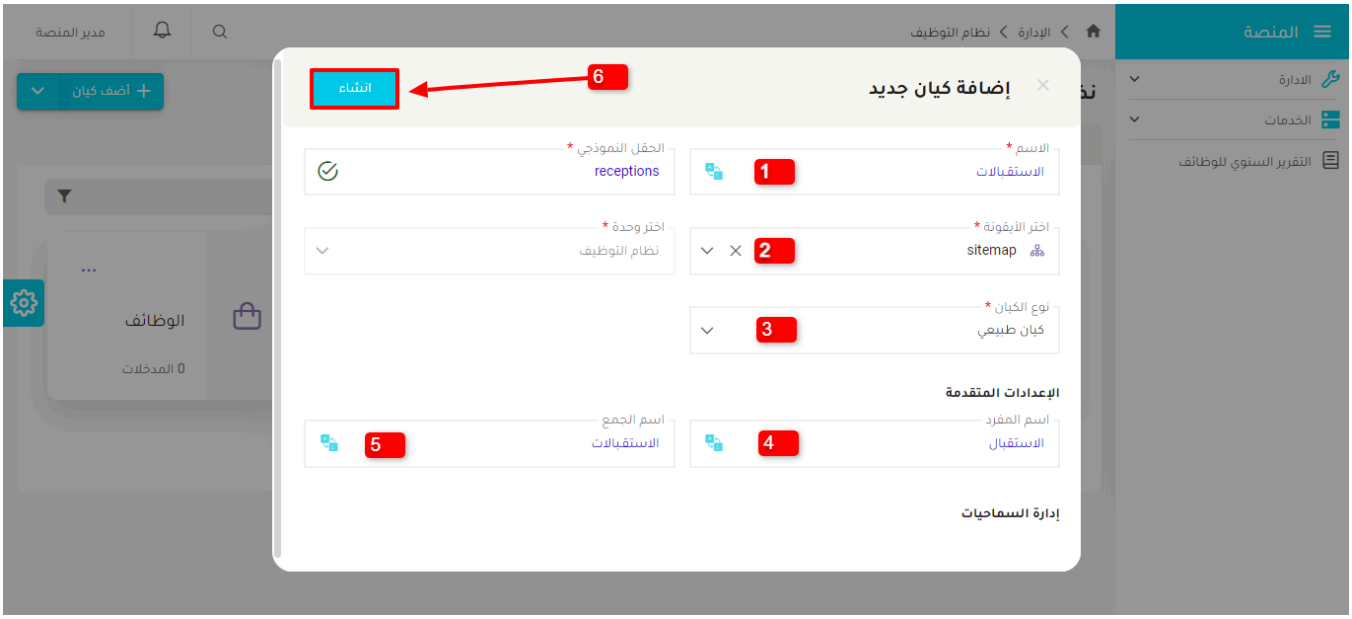
Receptions Entity

The Receptions entity holds welcome posts that introduce new employees to the company. Each reception is published on the company timeline.

Note: this page gives only what is specific to the Receptions entity: its settings, structure, reactions and timeline. For the step-by-step screens, follow the Jobs entity page.

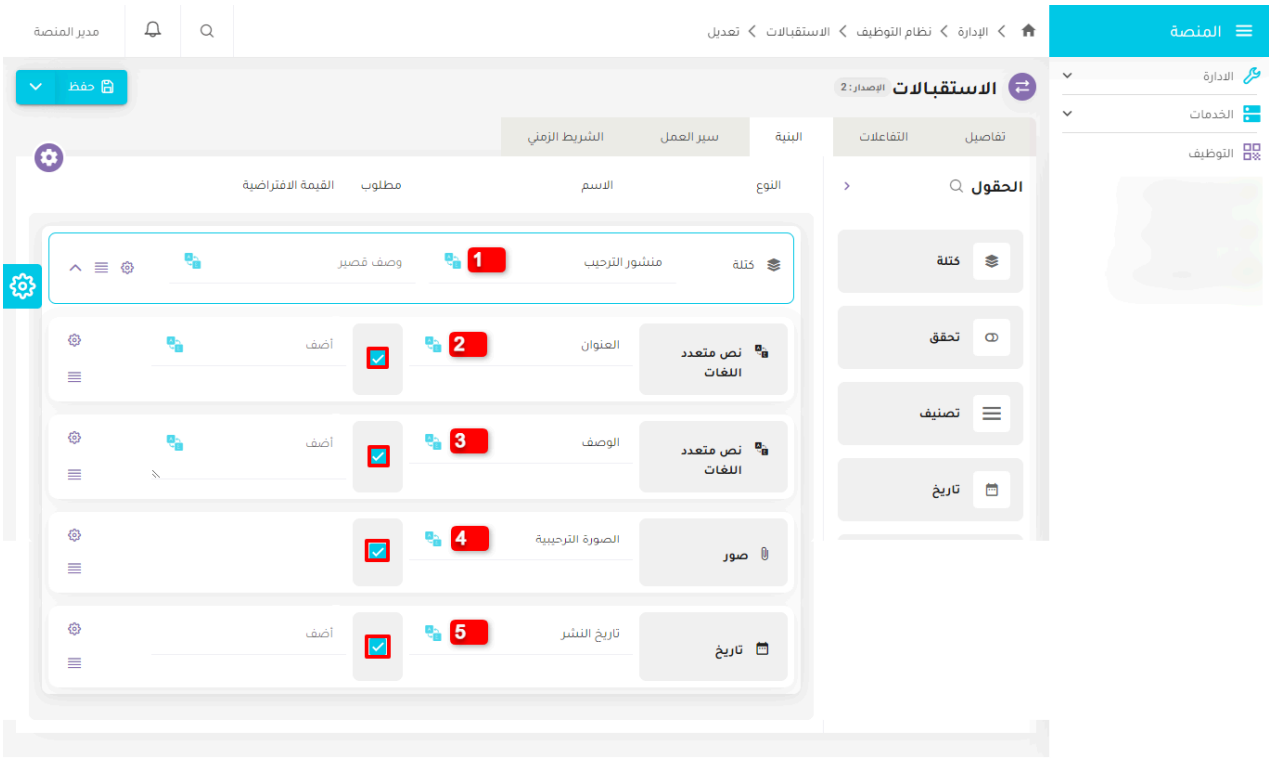
Create the Receptions entity

- **Name**: *Receptions*, in Arabic and English.
- **Select Icon**: pick an icon for the entity.
- **Data Entity Type**: **Normal Entity**.
- Under **Field Settings**: **Single Name** *Reception* and **Plural Name** *Receptions*.



Structure

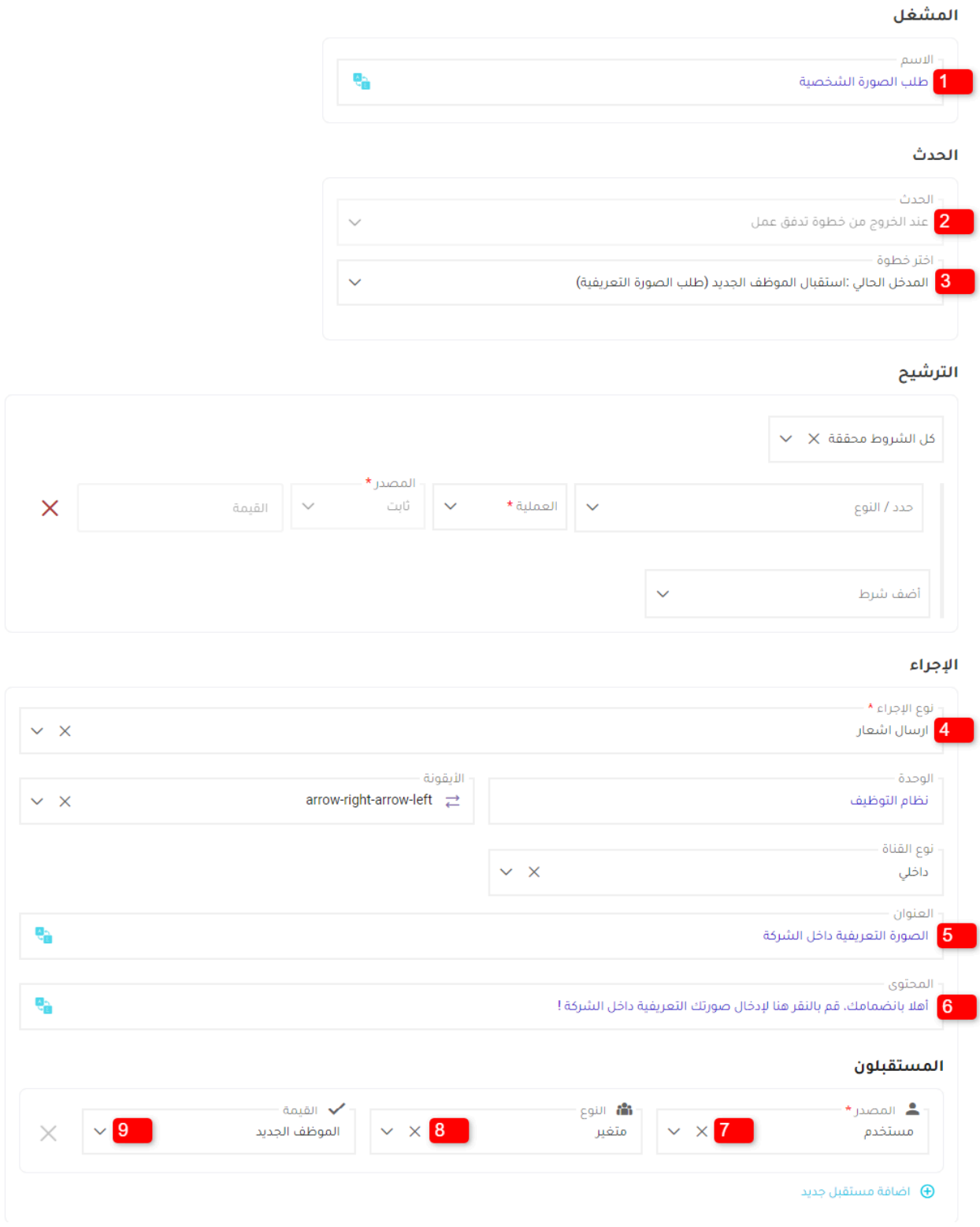
1. **Welcome post block**
    - **Title**: **Text**, with rich text turned on.
    - **Description**: **Text**, with rich text and multiple lines turned on.
    - **Welcome image**: **Image**.
    - **Publish date**: **Date**, the day the welcome post goes out.



- Select **Save** at the top of the page to keep your changes.

Reactions

1. **Request a profile photo**
    - **Event**: **On Entry Created**
    - **Action**: **Send Notification**
    - **Recipients**: the *HR* group



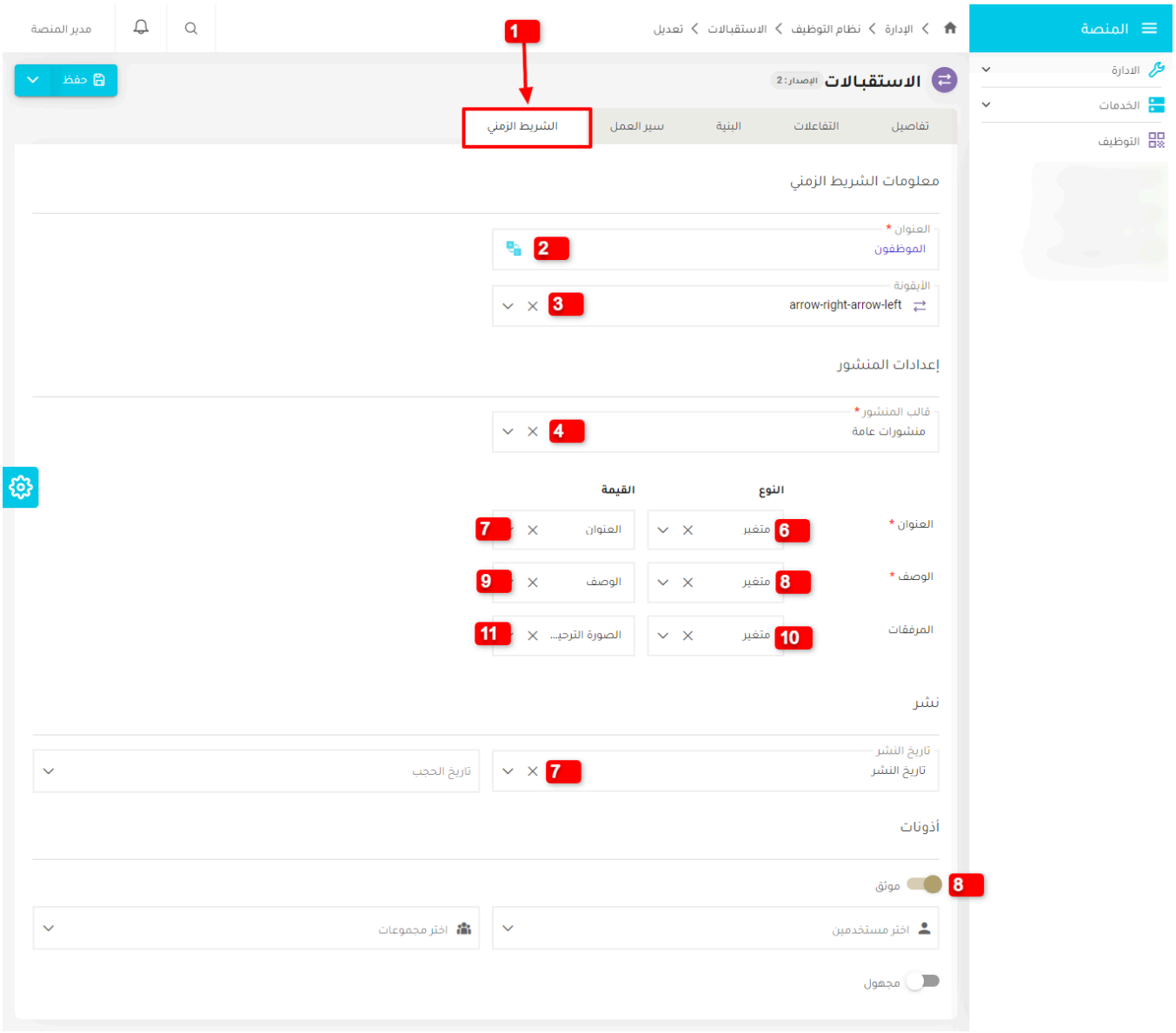
Timeline

In the entity designer, open the **Timeline** tab and add a timeline:

- **Timeline Information**: a **Title** (for example *Employees*) and an **Icon**.
- **Post Configuration**: **Post Template** *General Post*; the post's title from **Field** › *Title*, its description from **Field** › *Description*, and its attachments from **Field** › *Welcome image*.
- **Publish**: **Publish Date** › *Publish date*.

- **Permissions**: turn on **Authenticated**, so signed-in employees see the posts.

Select **Save** at the top of the page.



Jobs Entity

The Jobs entity holds the vacancies: a manager drafts a job, HR reviews it, and once approved it is published on the company timeline until its registration deadline. This page walks through every screen in detail; the other entity pages reuse these steps.

For more about entities, see Entities.

Create the Jobs entity

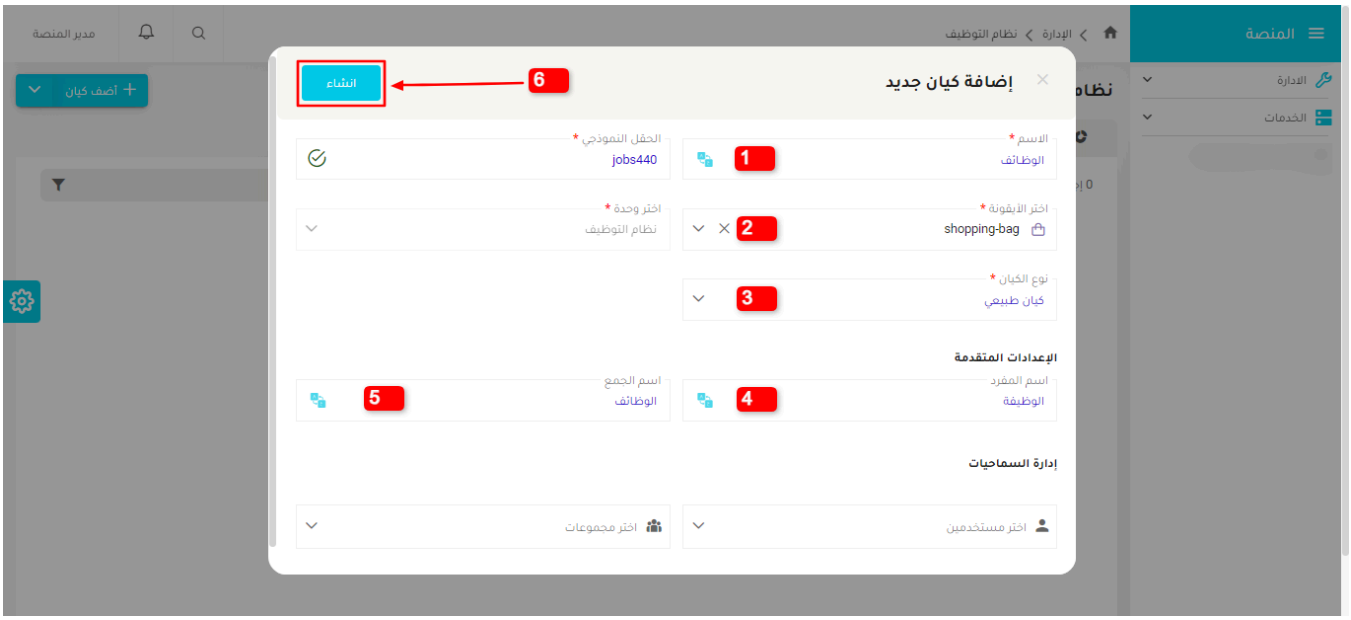
1. Open the *Recruitment System* module and, on its **Entities** tab, select **Add Data Entity**.



2. The **Add New Data Entity** dialog opens. Fill in:

   - **Name**: *Jobs*, in Arabic and English.
   - **Select Icon**: pick an icon for the entity.
   - **Data Entity Type**: **Normal Entity**.
   - Under **Field Settings**: **Single Name** *Job* and **Plural Name** *Jobs*.

3. Select **Create**. The *Jobs* entity is created.



Build the structure

1. Open the *Recruitment System* module: in the sidebar, open **Administration › Applications › Modules**, search for *Recruitment System* and select it.

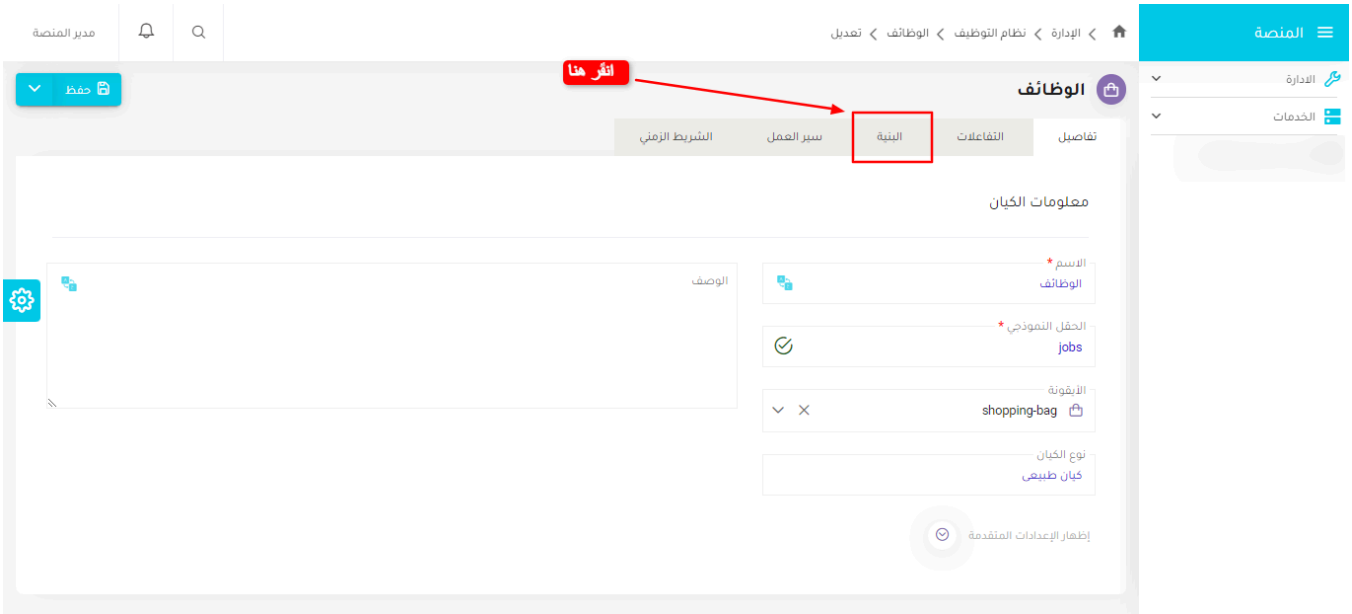
2. In the module, open the **Entities** tab.

3. Find *Jobs* and open the menu on its card.

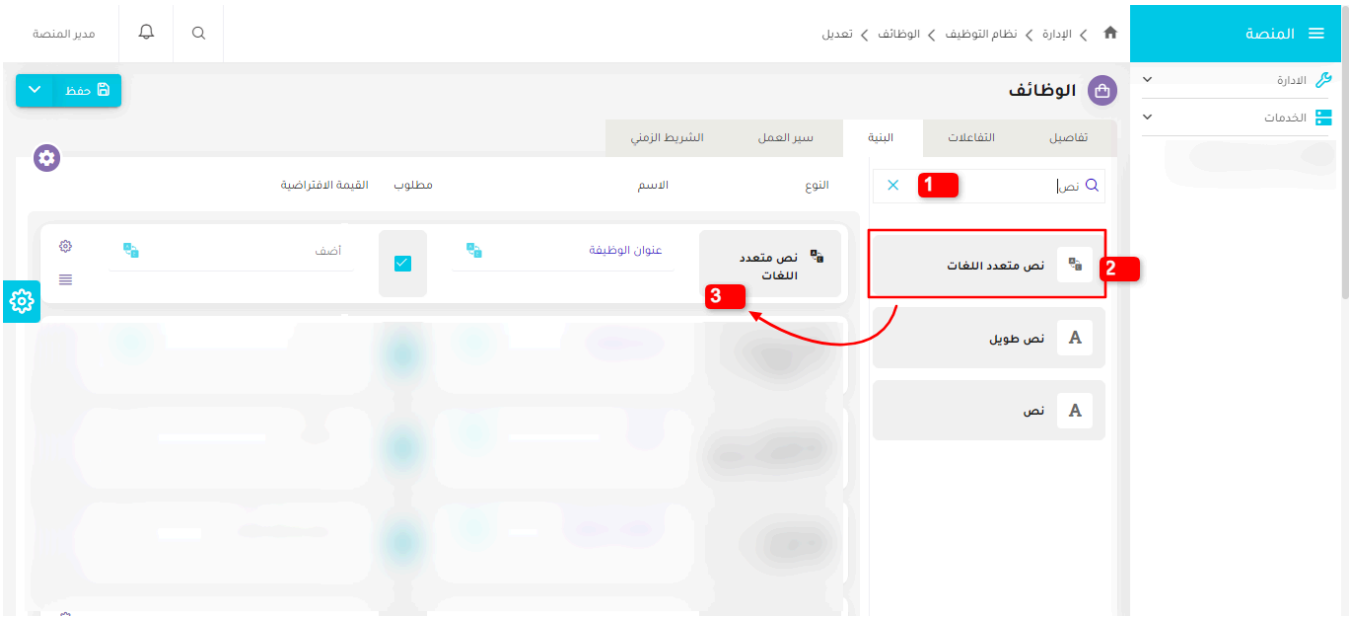
4. Select **Edit Entity**. The entity designer opens.



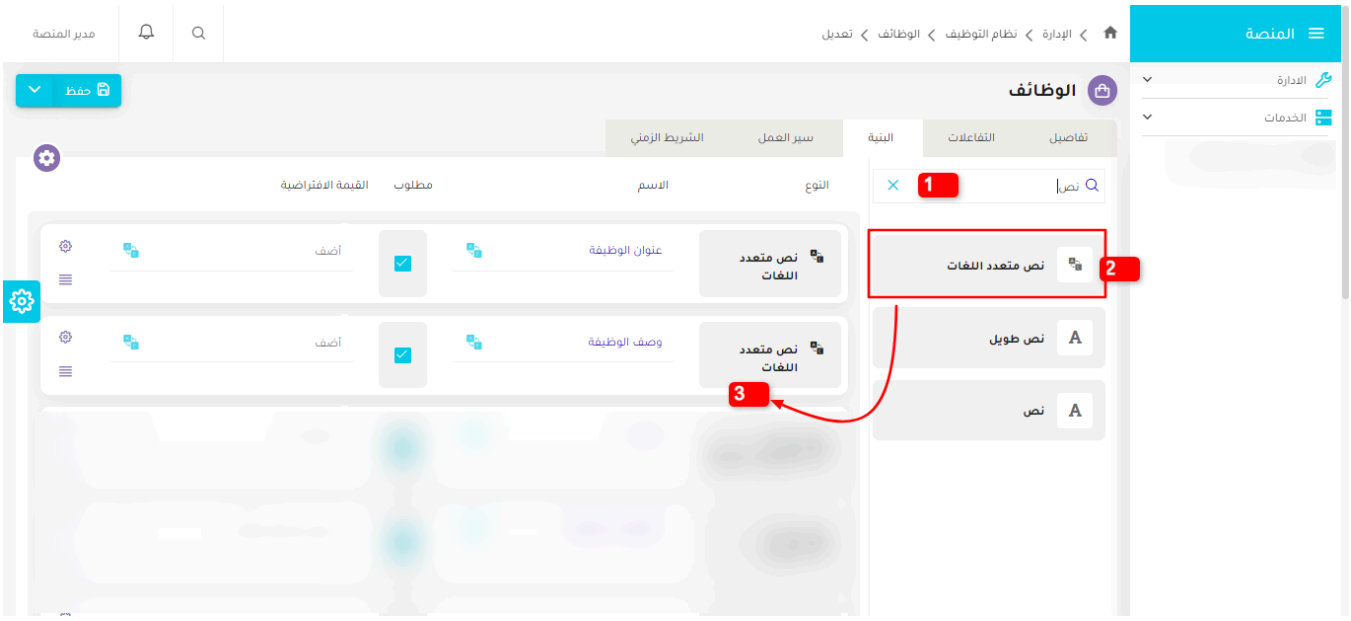
5. Open the **Structure** tab.



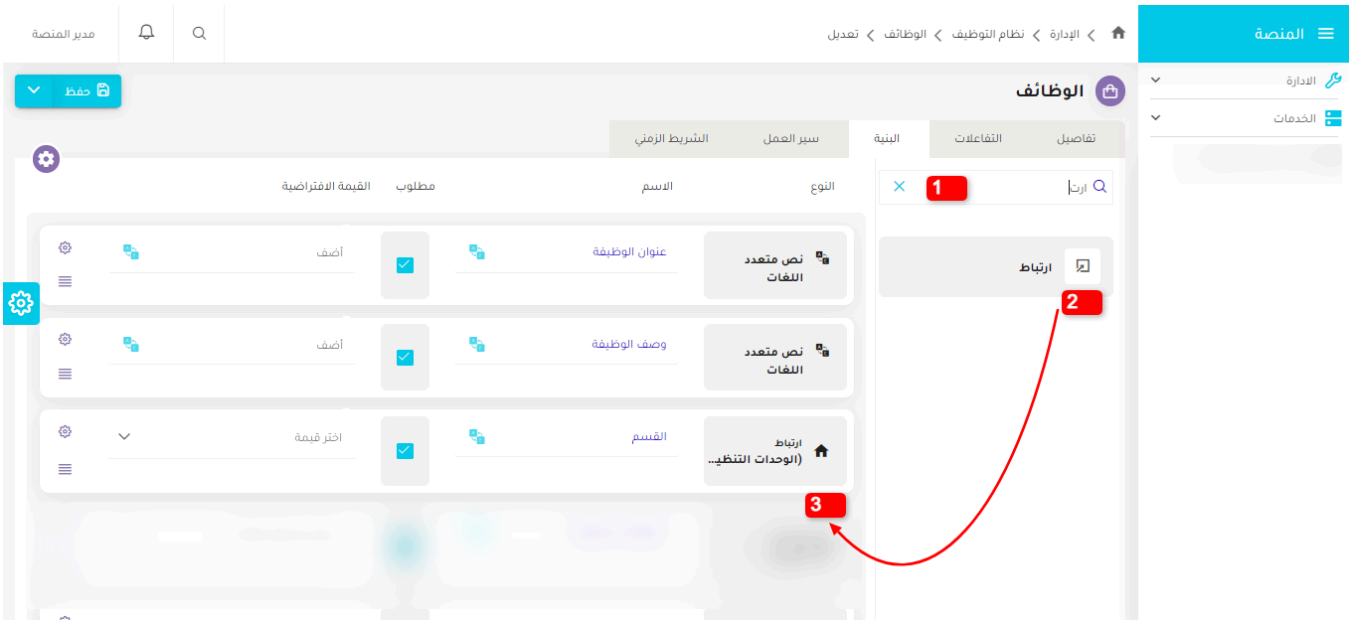
6. In the field palette, find **Localized Text**, drag it onto the structure area and drop it. Name the field *Job title* (in Arabic and English) and mark it **Required**.



7. Add a *Description* field the same way.

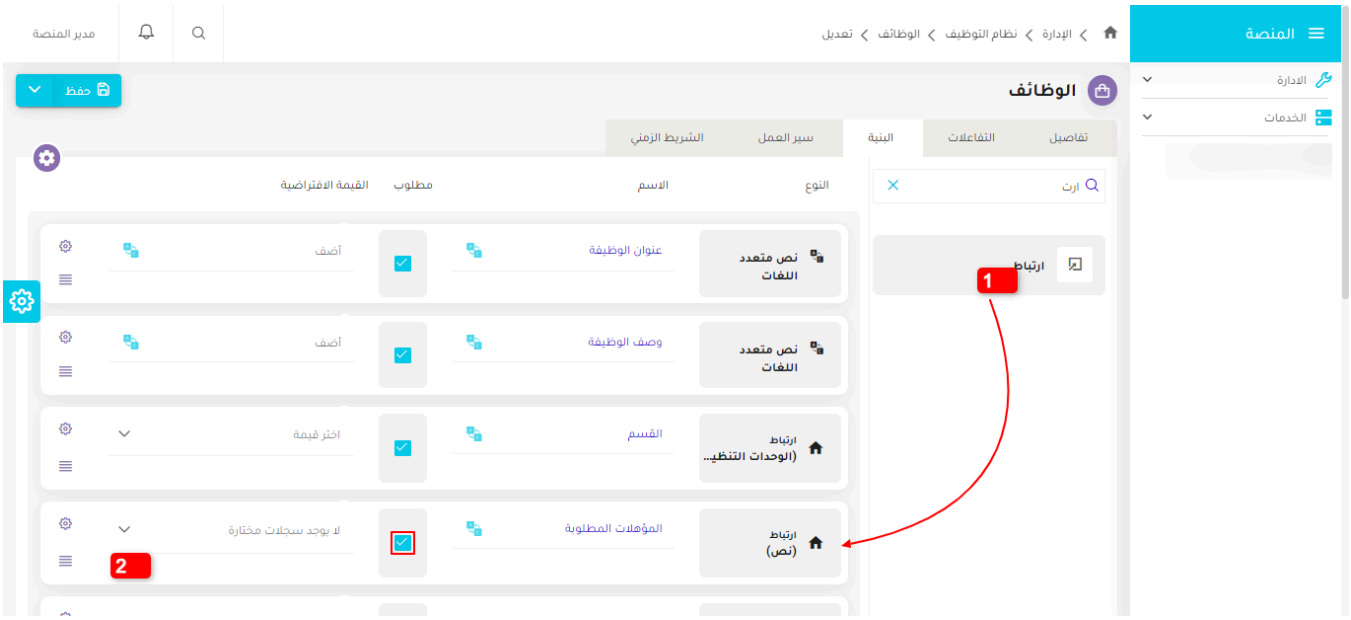


8. Drag a **Reference** field onto the structure area. A dialog asks what the field links to: search for the organization units entity, select it, and select **Save**. Name the field *Department* (in Arabic and English) and mark it **Required**.

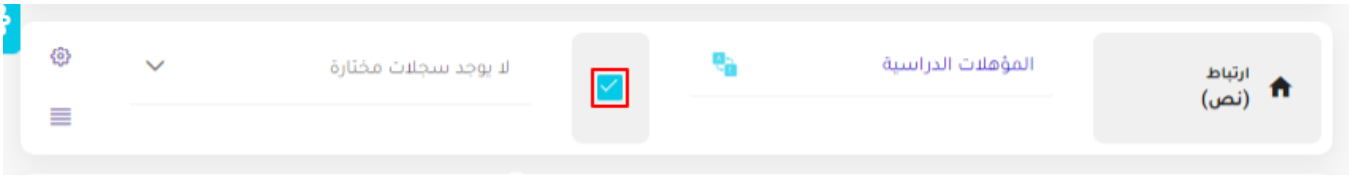




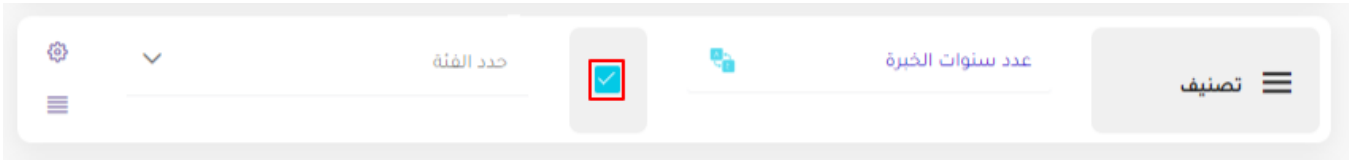
9. Add another **Reference** field. This time choose **Lists**, then the *Competencies* list you created earlier, and select **Save**. Name the field *Required qualifications* and mark it **Required**.



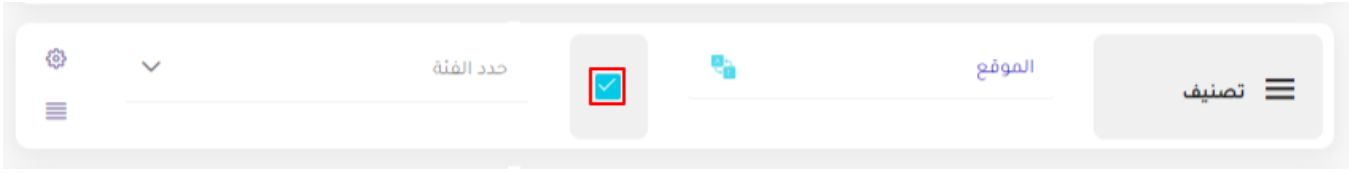
10. Add another **Reference** field linked to the *Educational qualifications* list. Name it *Educational qualifications* and mark it **Required**.



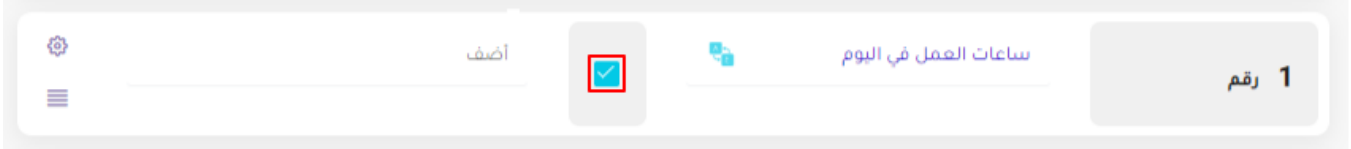
11. Drag a **Category** field onto the structure area. A dialog asks for its options. Enter the option *Less than 1 year* with its translation, pick a colour for it, and select **Add**. Repeat for *1 year*, *2–5 years*, *5–10 years* and *More than 10 years*. Name the field *Years of experience required* (in Arabic and English) and mark it **Required**.



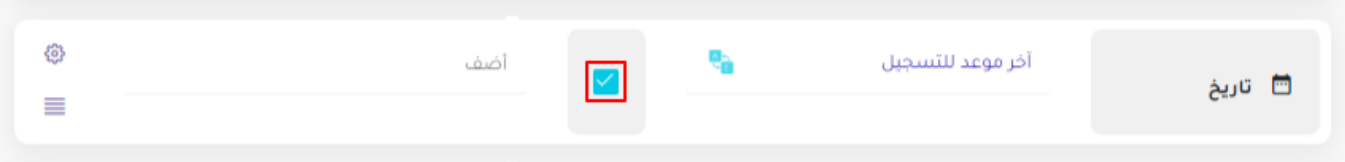
12. Add another **Category** field with the options *Remote*, *Hybrid* and *On-site*. Name it *Location* and mark it **Required**.



13. Drag a **Number** field onto the structure area and name it *Working hours per day*.



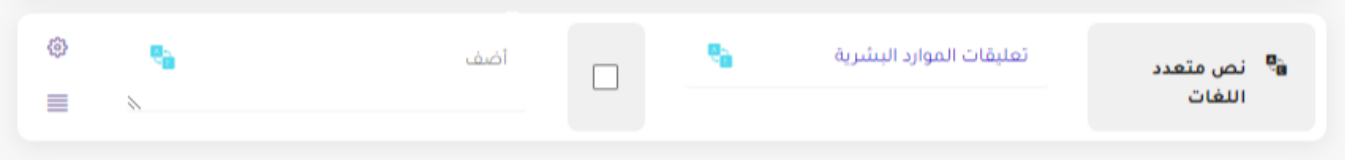
14. Drag a **Date** field onto the structure area, name it *Registration deadline*, and mark it **Required**.



15. Add a **Category** field with the options *Open*, *Closed* and *Archived*, and name it *Job status*. The reactions below set this field for you.



16. Add a **Localized Text** field named *HR comments*.



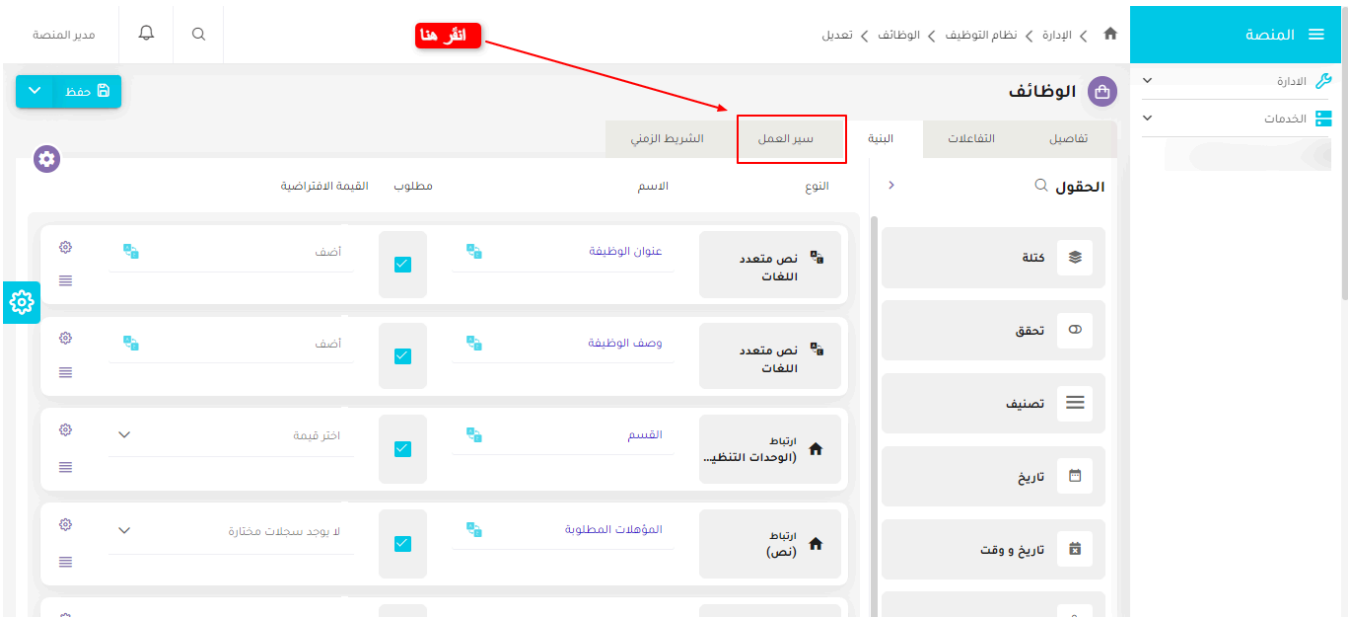
17. When each field's options are set, select **Save** in its dialog.

18. Select **Save** at the top of the page.

Build the workflow

The workflow sends every new job to HR for review. HR approves it (it is then published), returns it to its creator for changes, or archives it.

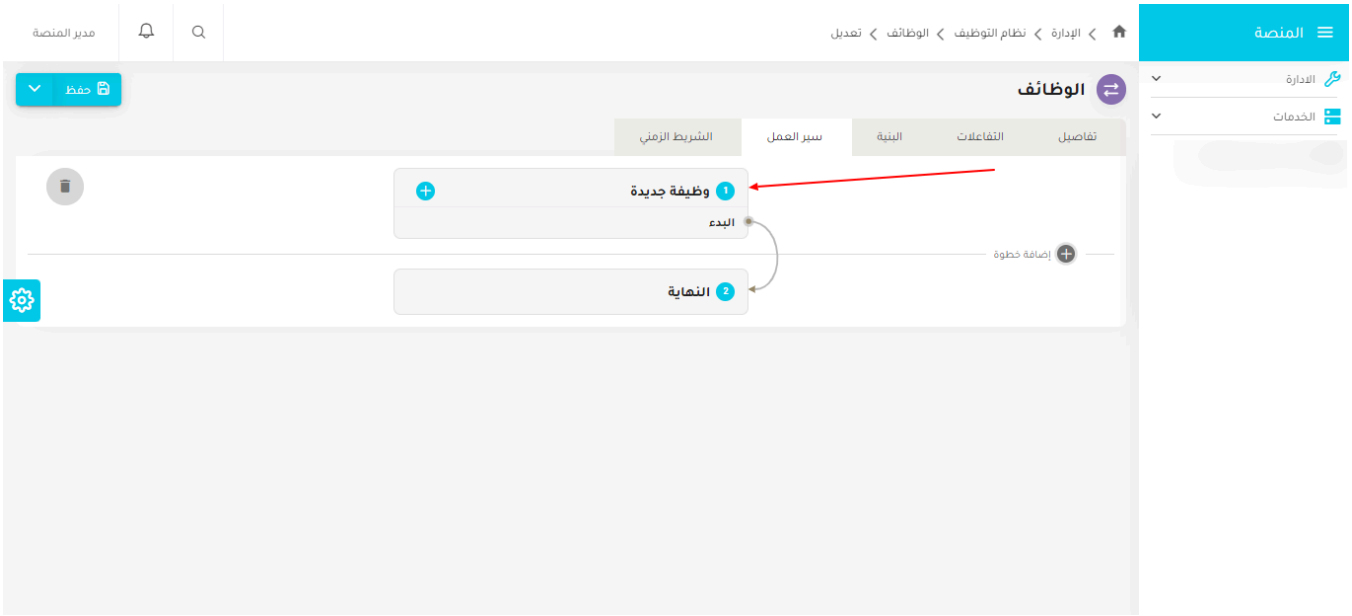
1. Open the **Workflows** tab of the same designer.



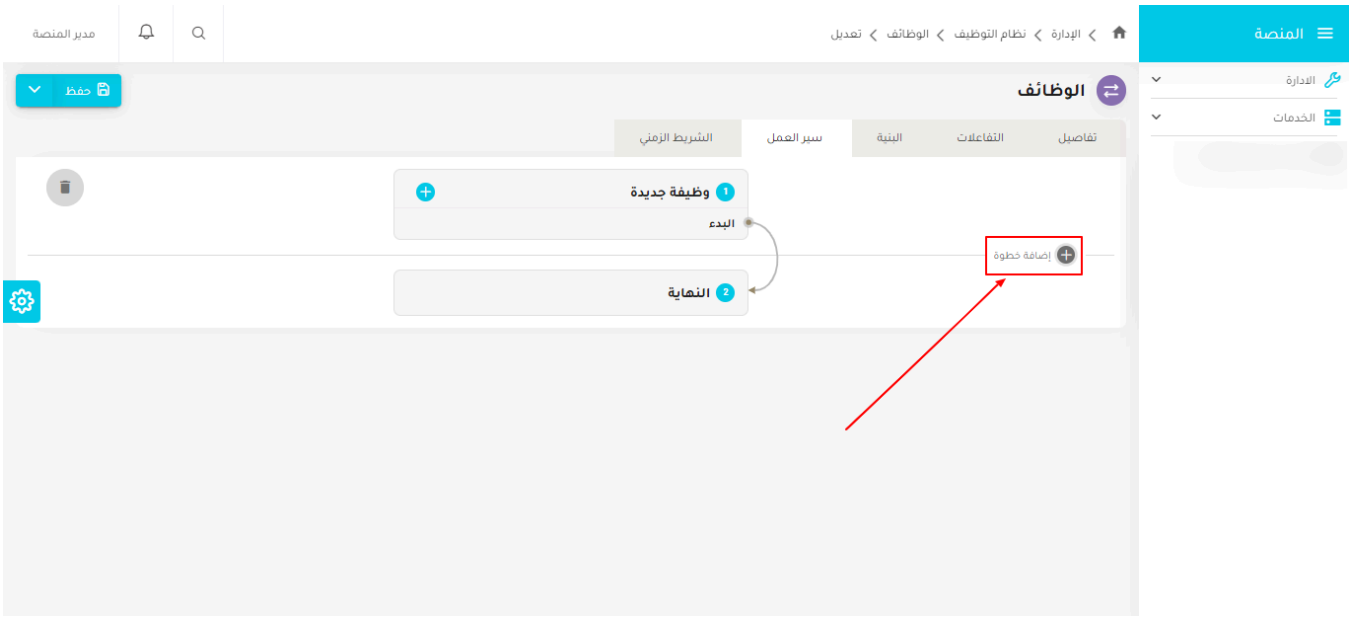
2. Select **Add A Workflow**. A workflow is created with two steps, *Initial Step* and *Complete Step*.



3. Rename *Initial Step* to *New job*: select the step's name, change **Name** in the step settings, then close the settings to return to the workflow.



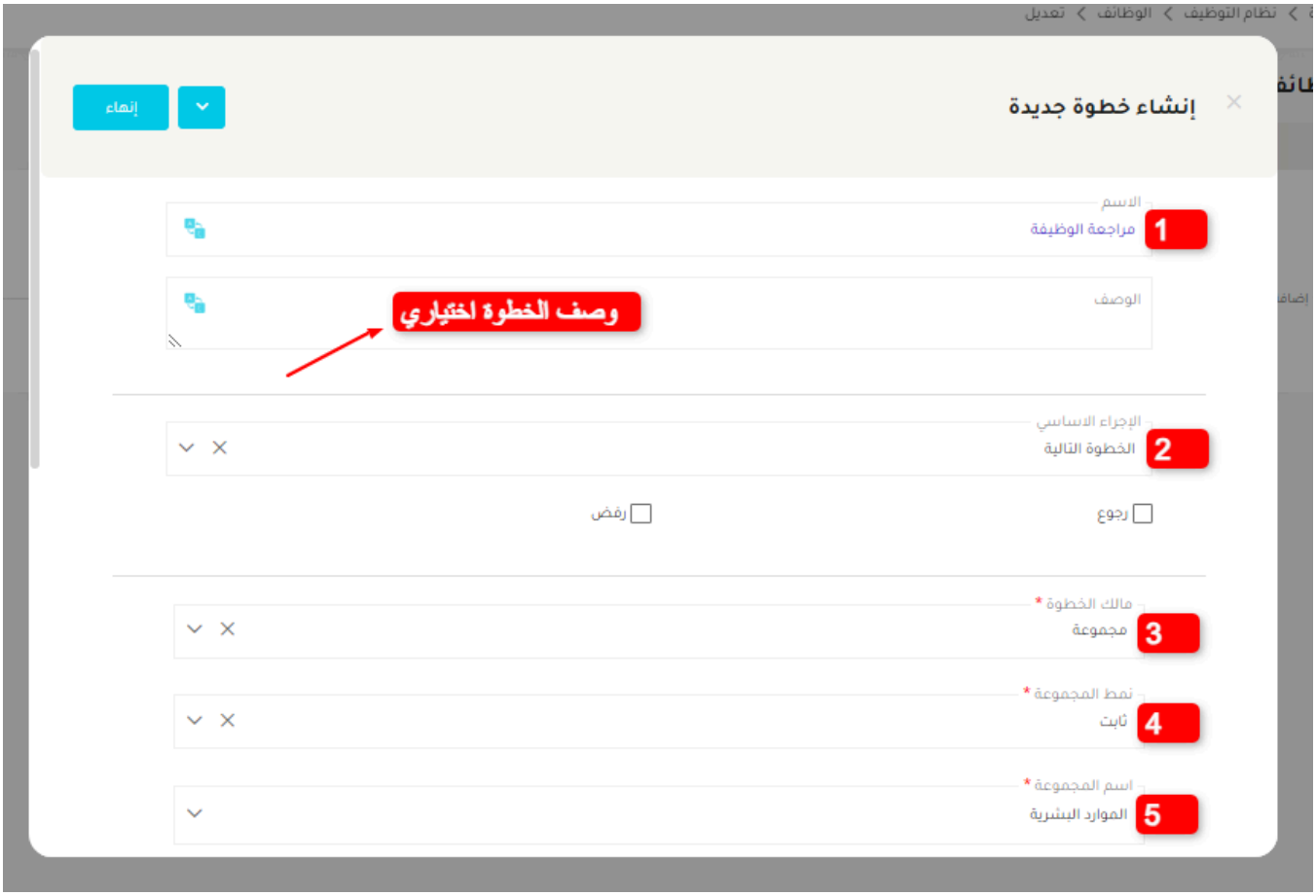
4. Select **Create Step** after *New job*.

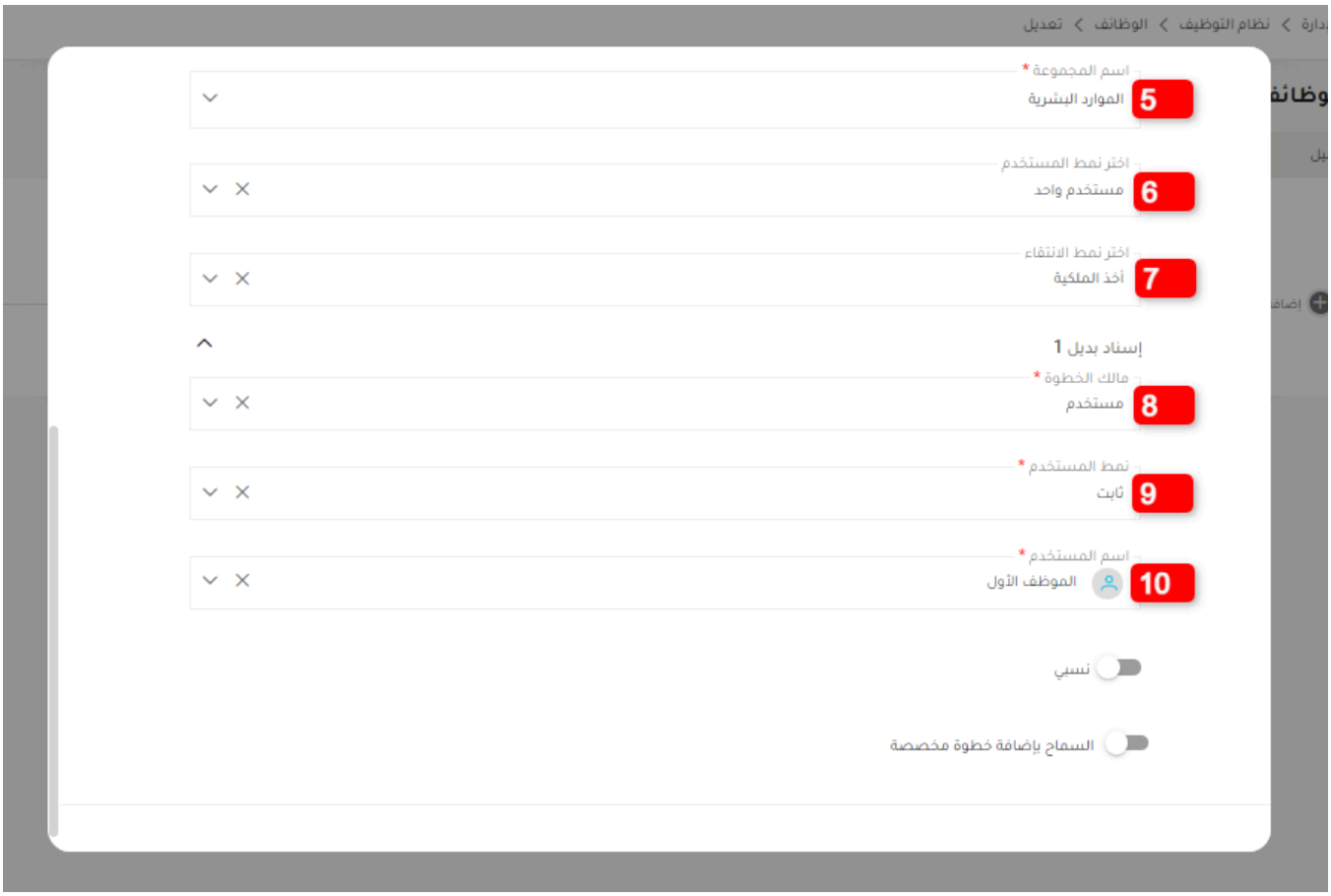


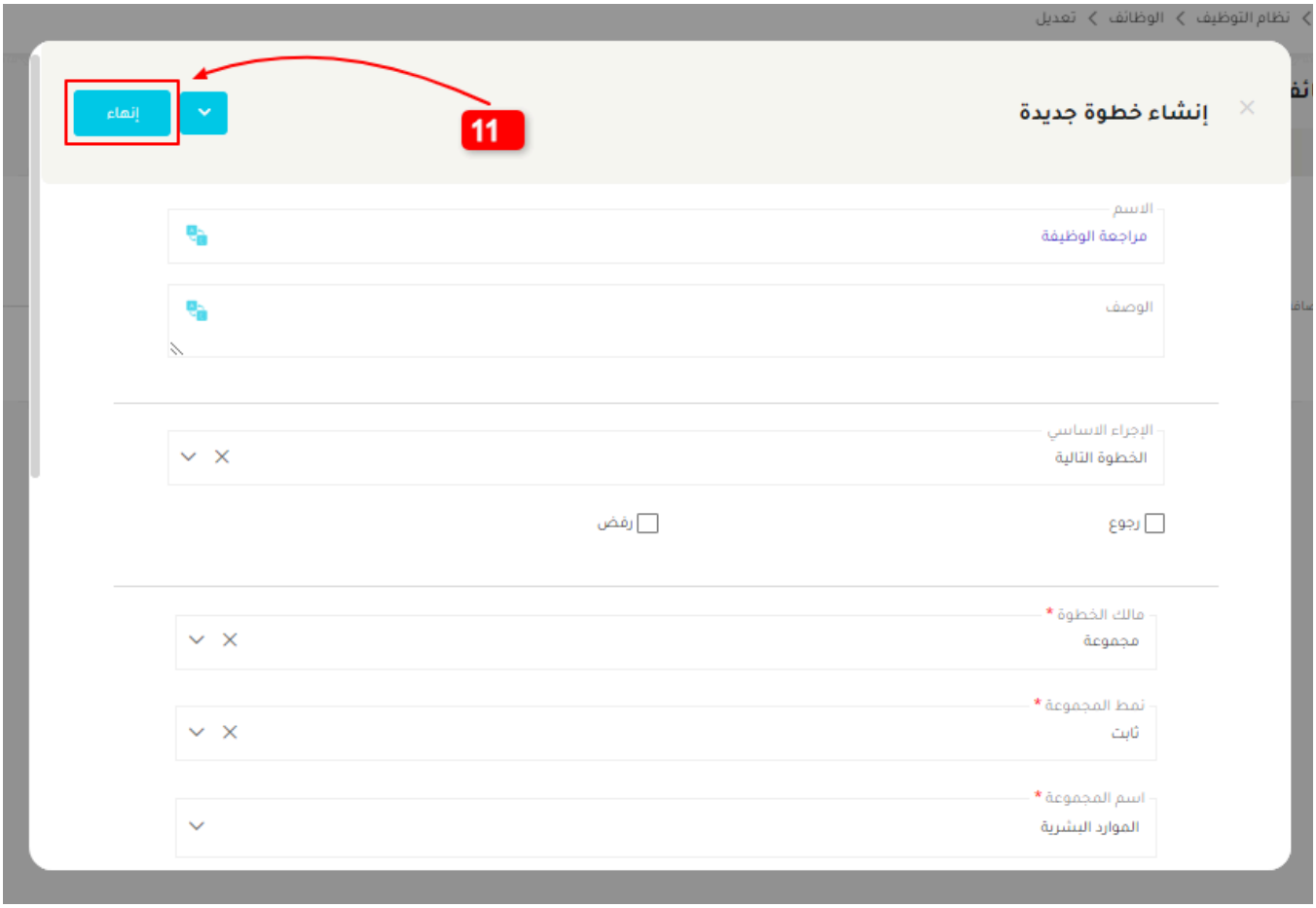
5. The **Create New Step** dialog opens. Fill it in:

    - **Name**: *Job review*. The description is optional.
    - **Main Action**: **Next Step**. Tick **Return** and **Reject** as well: HR needs them to send a job back or archive it.
    - **Step Owner**: **Group**, **Group Type Fixed**, **Group Name** *HR*.
    - **Select User Type**: **Single User**, and **Choose Selection Type**: **Take Ownership**, so any HR member can pick the job up.
    - Under **Fallback Assignment 1**: **Step Owner User**, **User Type Fixed**, and in **User Name** the person who should get the job if nobody in HR can.

    Then select **Done**.


إنشاء خطوة جديدة
إنهاء
الاسم
مراجعة الوظيفة
1
الوصف
وصف الخطوة اختياري
الإجراء الاساسي
الخطوة التالية
2
رجوع
رفض
مالك الخطوة *
مجموعة
3
نمط المجموعة *
ثابت
4
اسم المجموعة *
الموارد البشرية
5

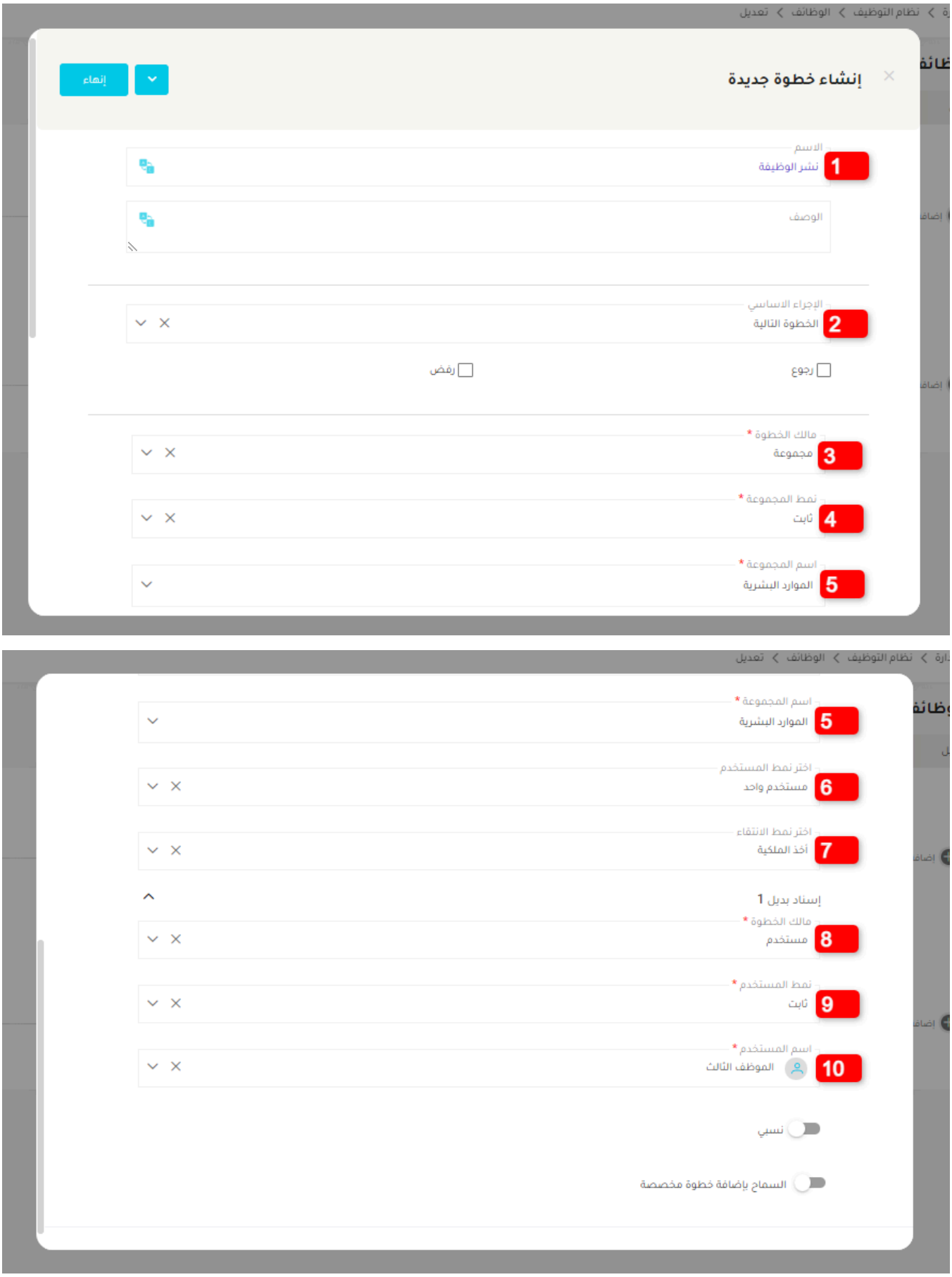

اسم المجموعة *
الموارد البشرية
5
اختر نمط المستخدم
مستخدم واحد
6
اختر نمط الانتقاء
أخذ الملكية
7
إسناد بديل 1
مالك الخطوة *
مستخدم
8
نمط المستخدم *
ثابت
9
اسم المستخدم *
الموظف الأول
10
نسي
السماح بإضافة خطوة مخصصة



6. Rename the step's main action to *Approve*, and its reject action to *Archive*.

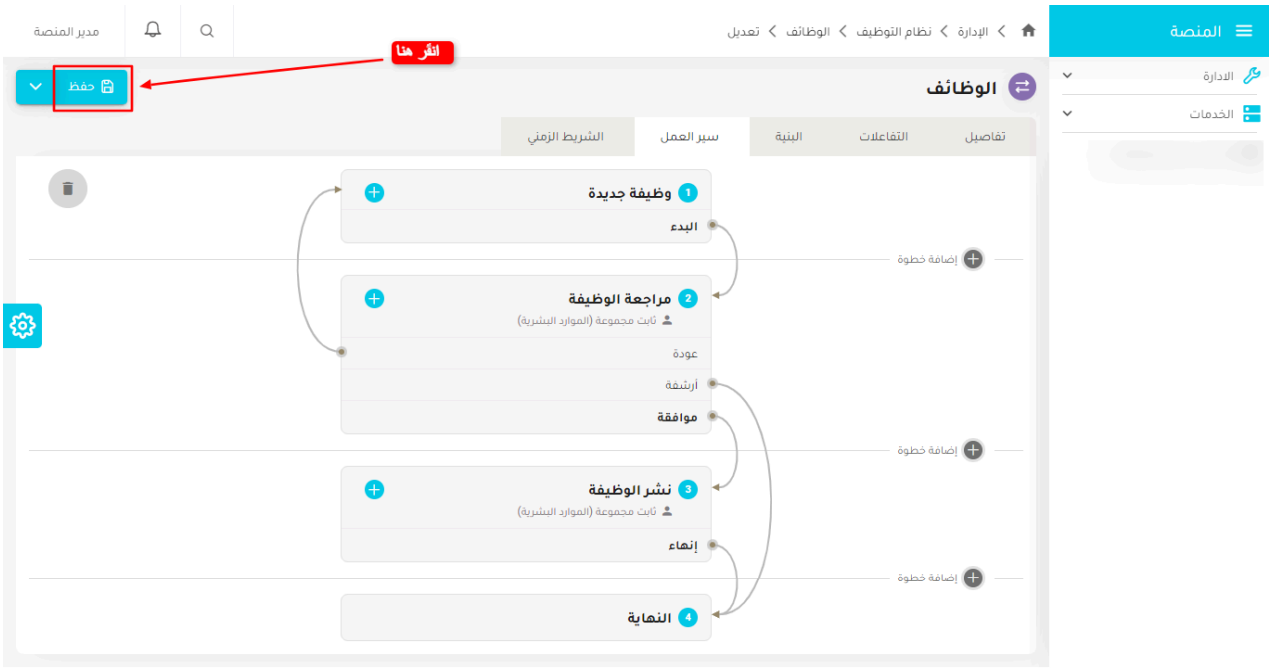


7. Add a *Job publication* step after *Job review* the same way: **Main Action Next Step** (no **Return** or **Reject**), owned by the *HR* group with **Take Ownership**, and a fixed fallback user.



8. Rename the main action of *Job publication* to *Finish*.

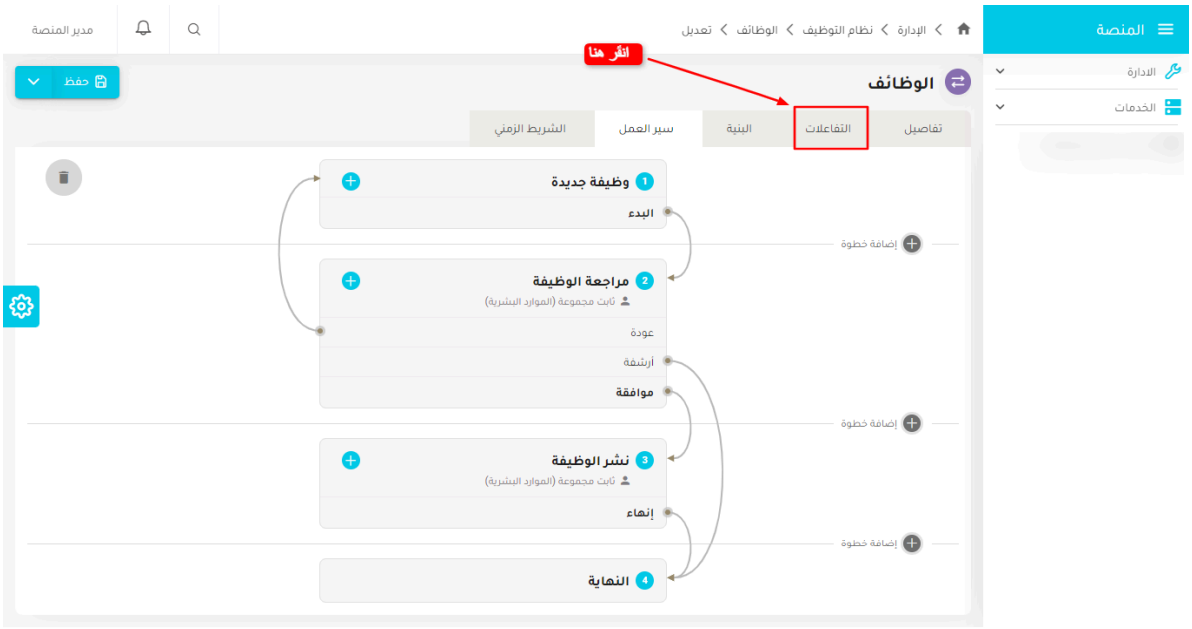
9. Select **Save** at the top of the page to save the entity, workflow included.



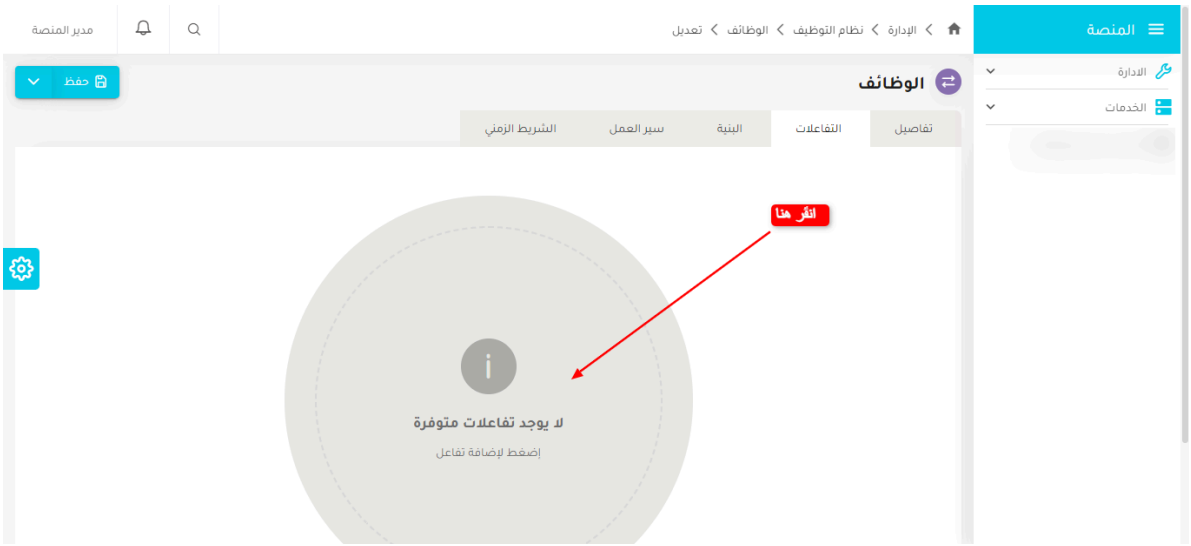
Add the reactions

- Reactions do the routine work for you: they notify people, set the job status and move the job through its workflow. Reactions are added on the entity's own page, not in the designer.

1. Open the *Jobs* entity (select its name on the module's **Entities** tab) and open the **Reactions** tab.

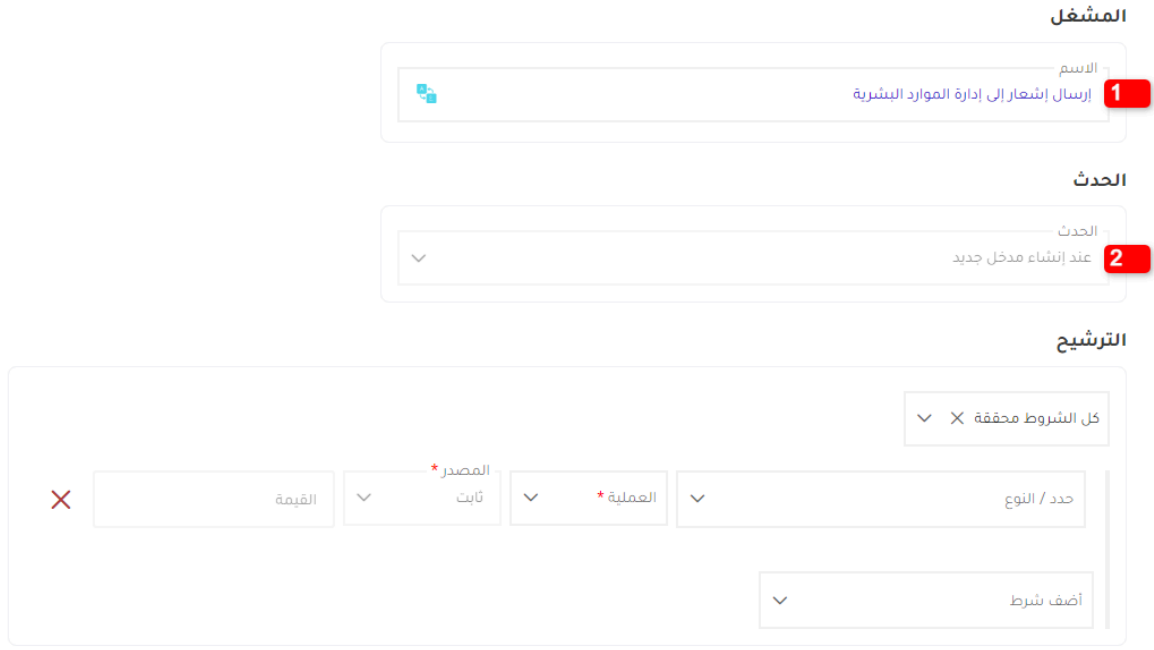


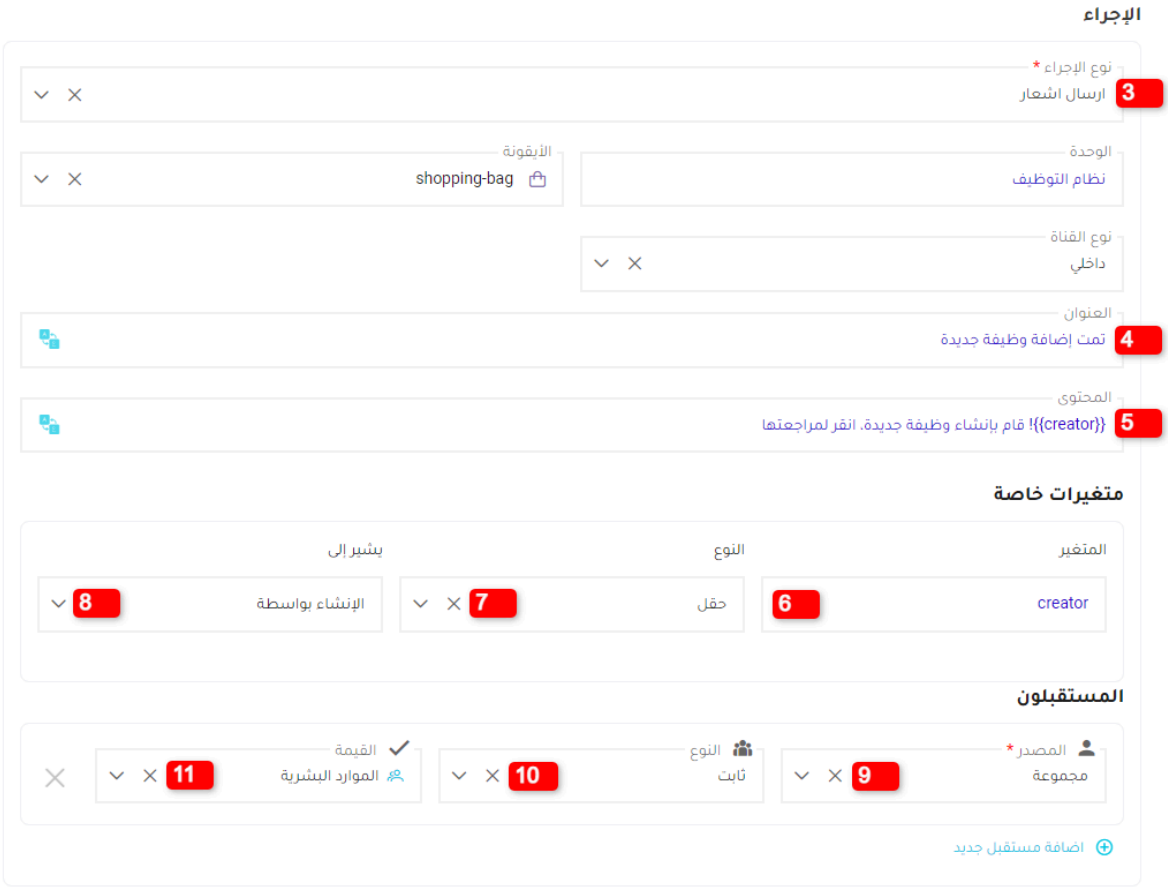
2. Select **Add Reaction**.



3. Reaction: notify HR when a new job is created

    1. Fill in the reaction:
        - **Trigger**: **Name** *Notify HR of a new job.*
        - **Event**: **On Entry Created**. Leave **Filter** empty.
        - **Action**: **Action Type Send Notification**, **Channel Type In App**, and an **Icon**.
        - **Title**: *A new job was added*. **Content**: *{{creator}} created a new job. Select it to review it.*
        - **Custom Variables**: **Variable** *creator*, **Type Field**, **Map to** the job's *Created by* attribute.
        - **Recipients**: **Source Group**, **Type Fixed**, **Value** *HR*.





2. Add the reaction to the list.



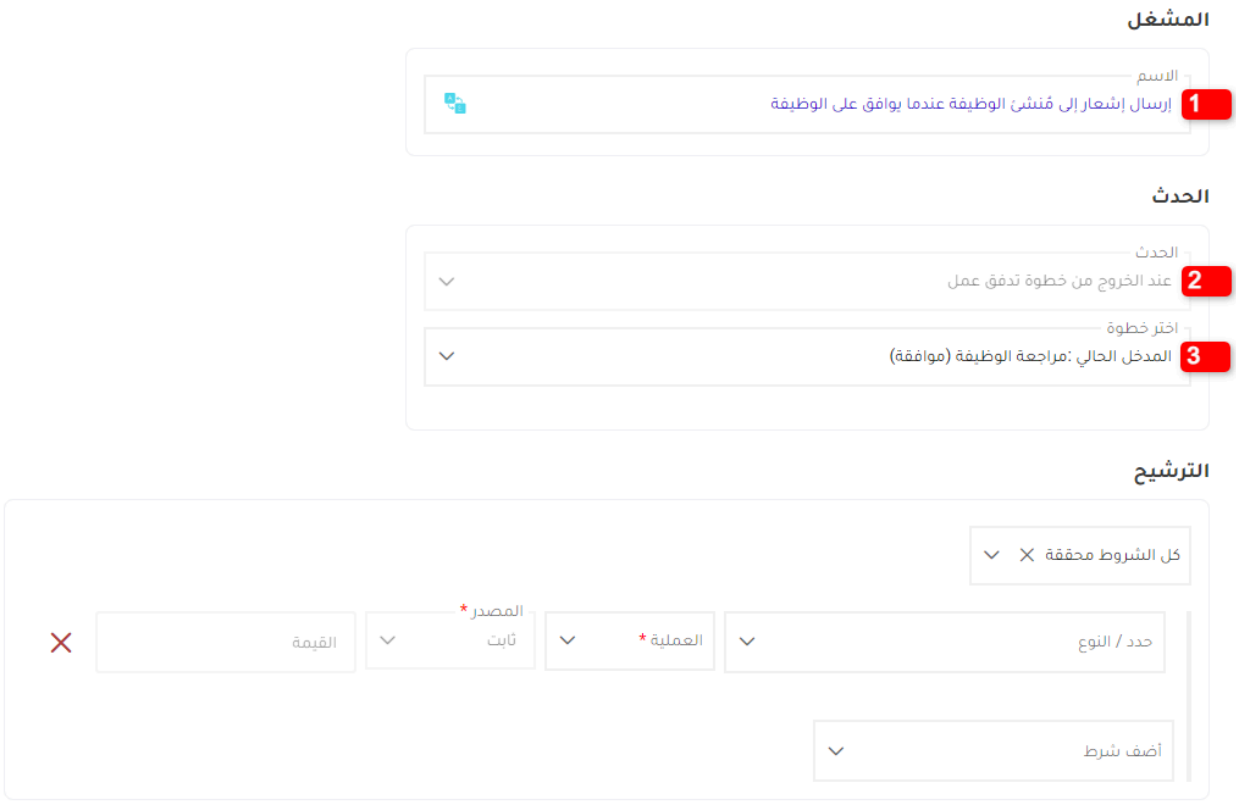
3. Select **Save** to save the reaction.

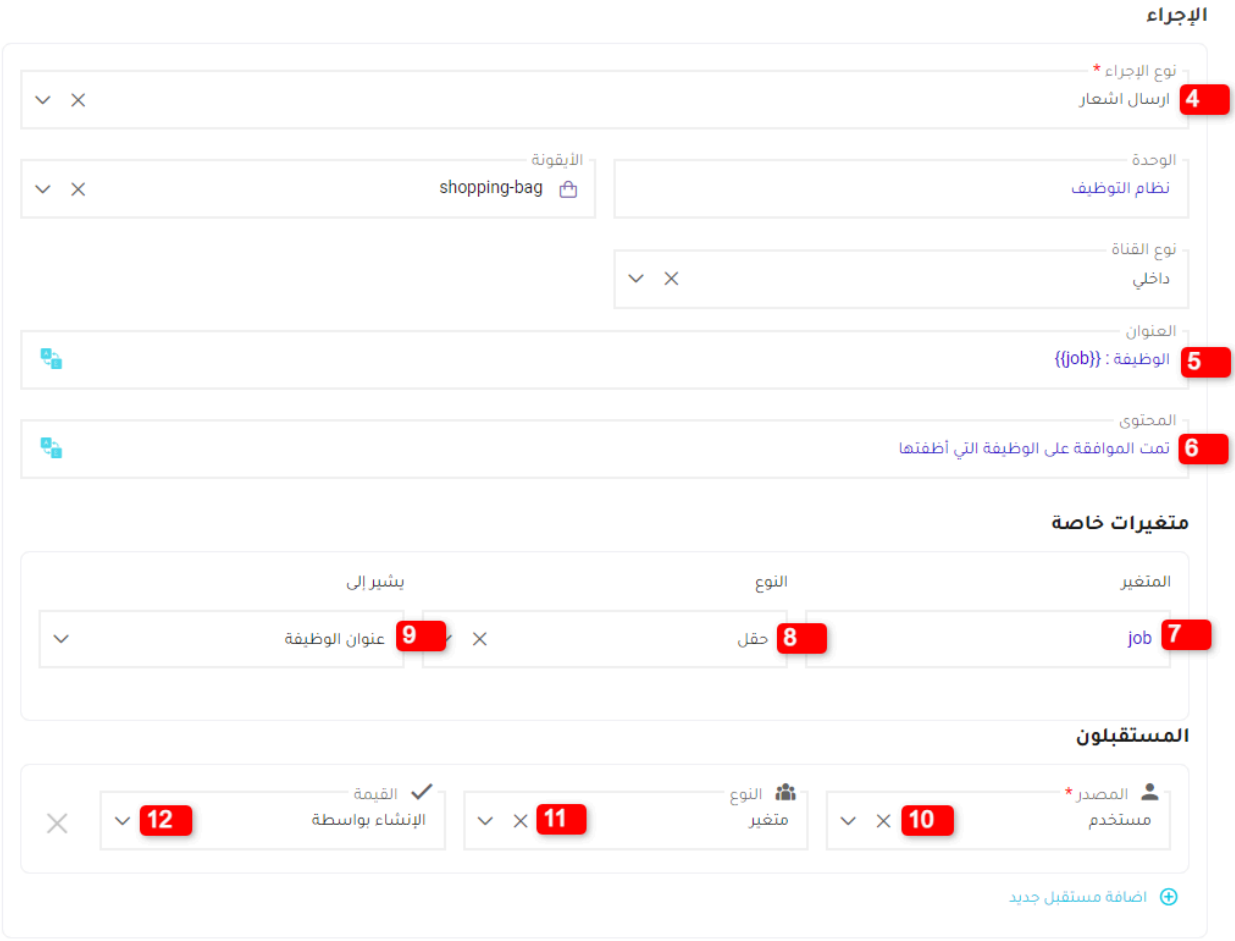


- Add the remaining reactions the same way:

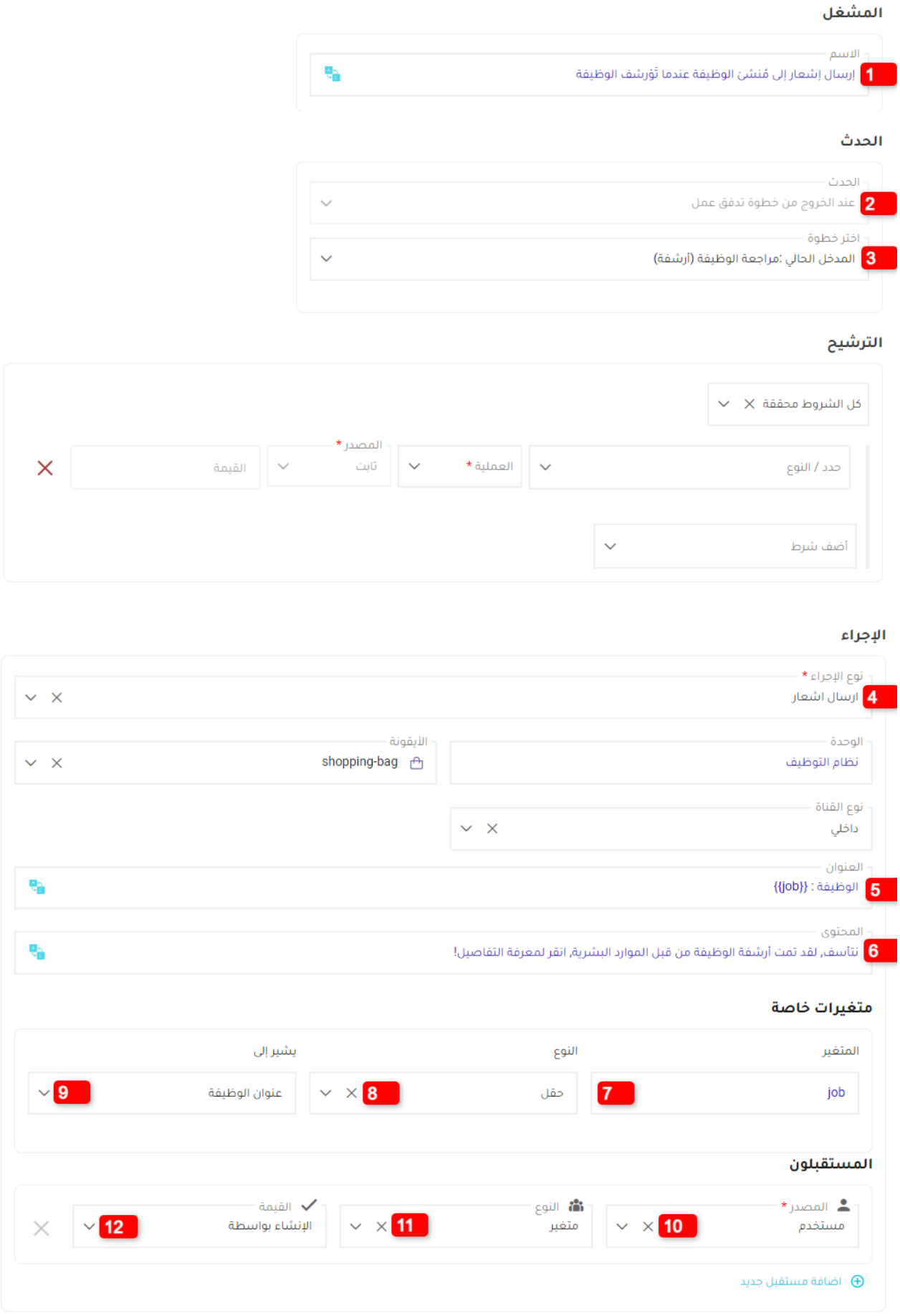
4. Reaction: notify the job's creator when the job is approved
    - **Event**: **On Workflow Step Exited**; in **Select Step**, choose *Job review* with its *Approve* action.

- **Action**: **Send Notification**, **In App**. **Title** *Job: {{job}}*; **Content** *The job you created has been approved.*
- **Custom Variables**: *job*, **Field**, mapped to *Job title*.
- **Recipients**: **Source User**, **Type Field**, **Value** *Created by*.



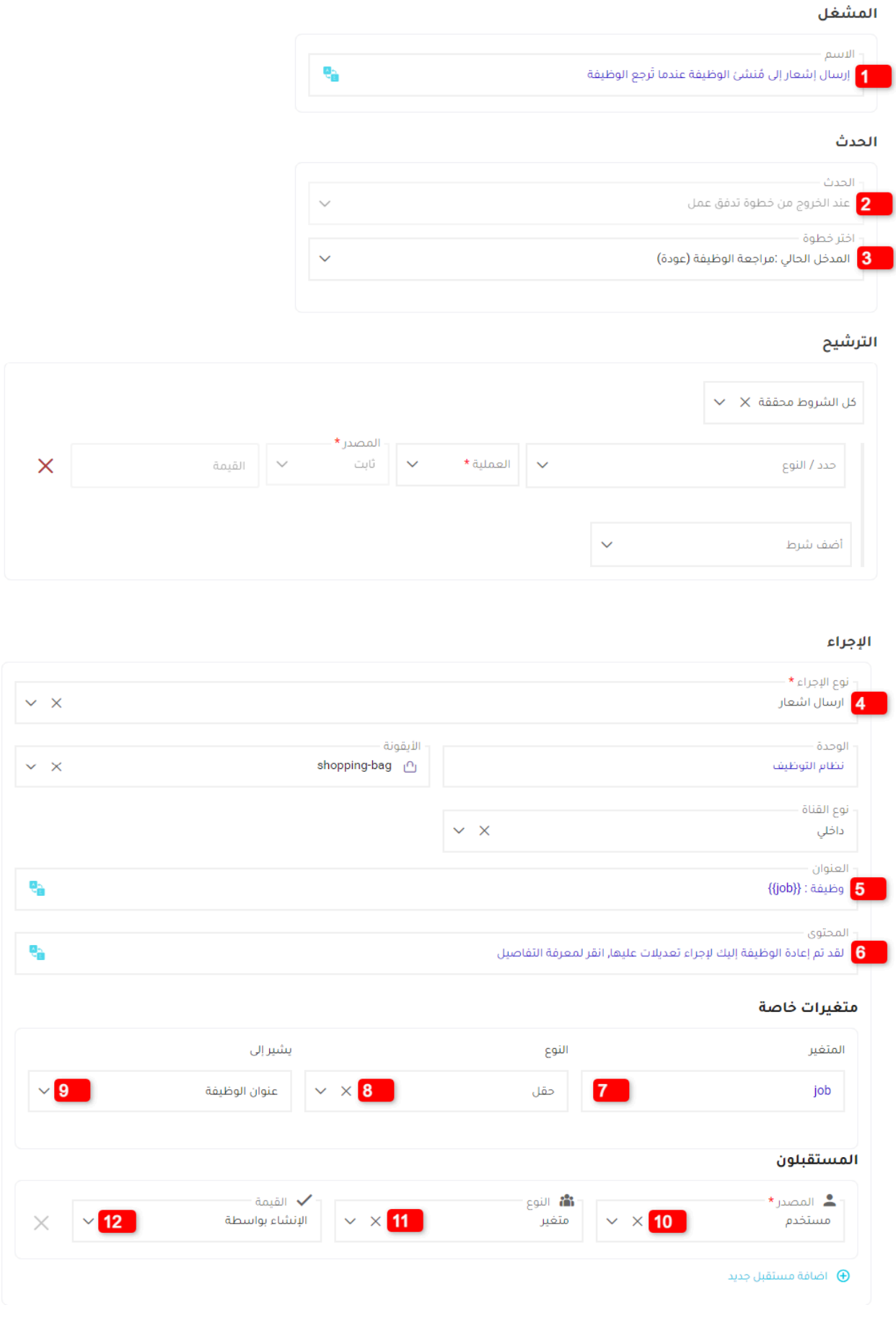


5. Reaction: notify the job's creator when the job is archived
    - As above, but in **Select Step** choose *Job review* with its *Archive* action. **Content**: *Sorry, HR has archived the job. Select it for the details.*

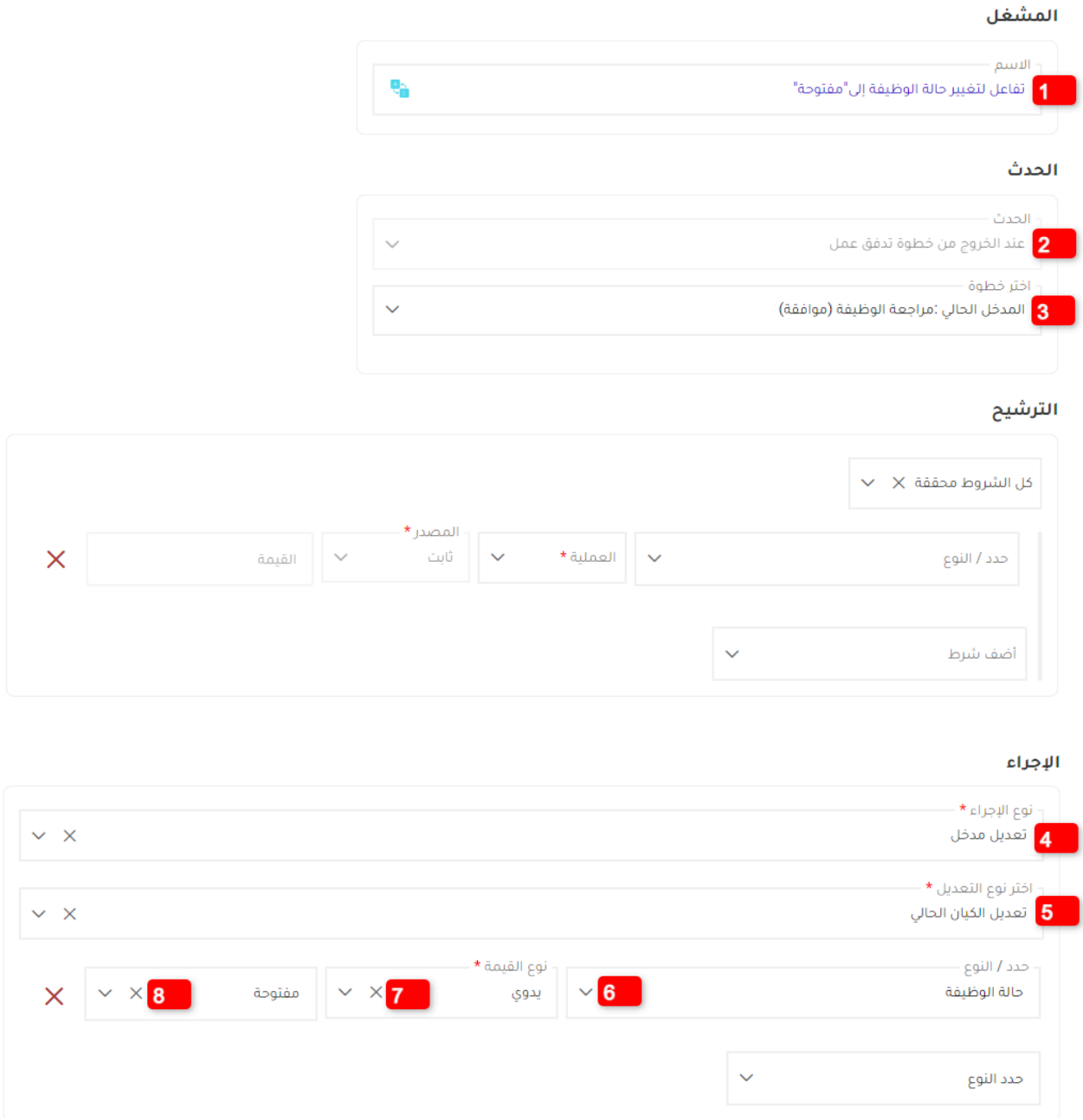


6. Reaction: notify the job's creator when the job is returned

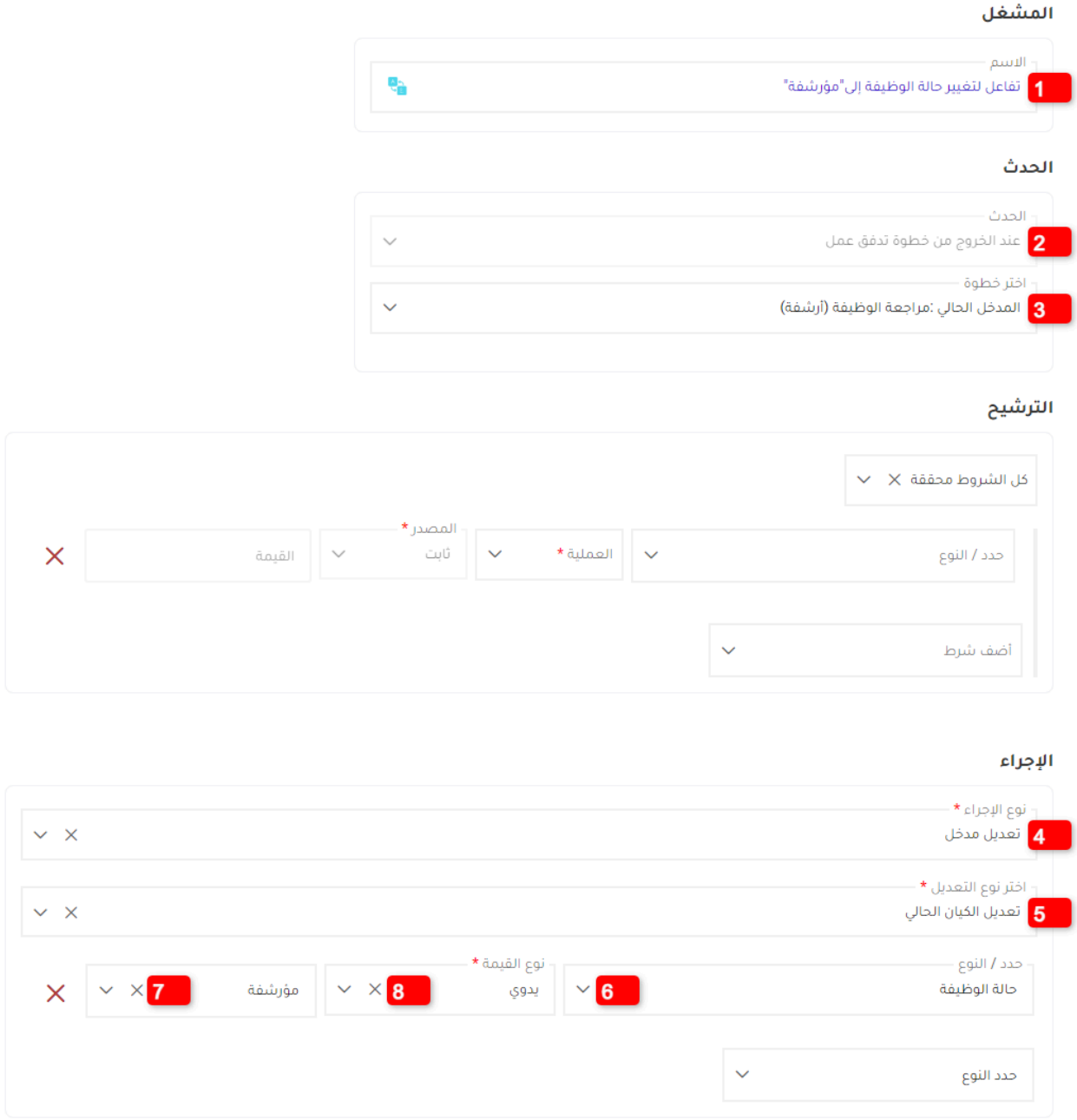
- As above, with *Job review* and its *Return* action. **Content**: *The job has been returned to you for changes. Select it for the details.*



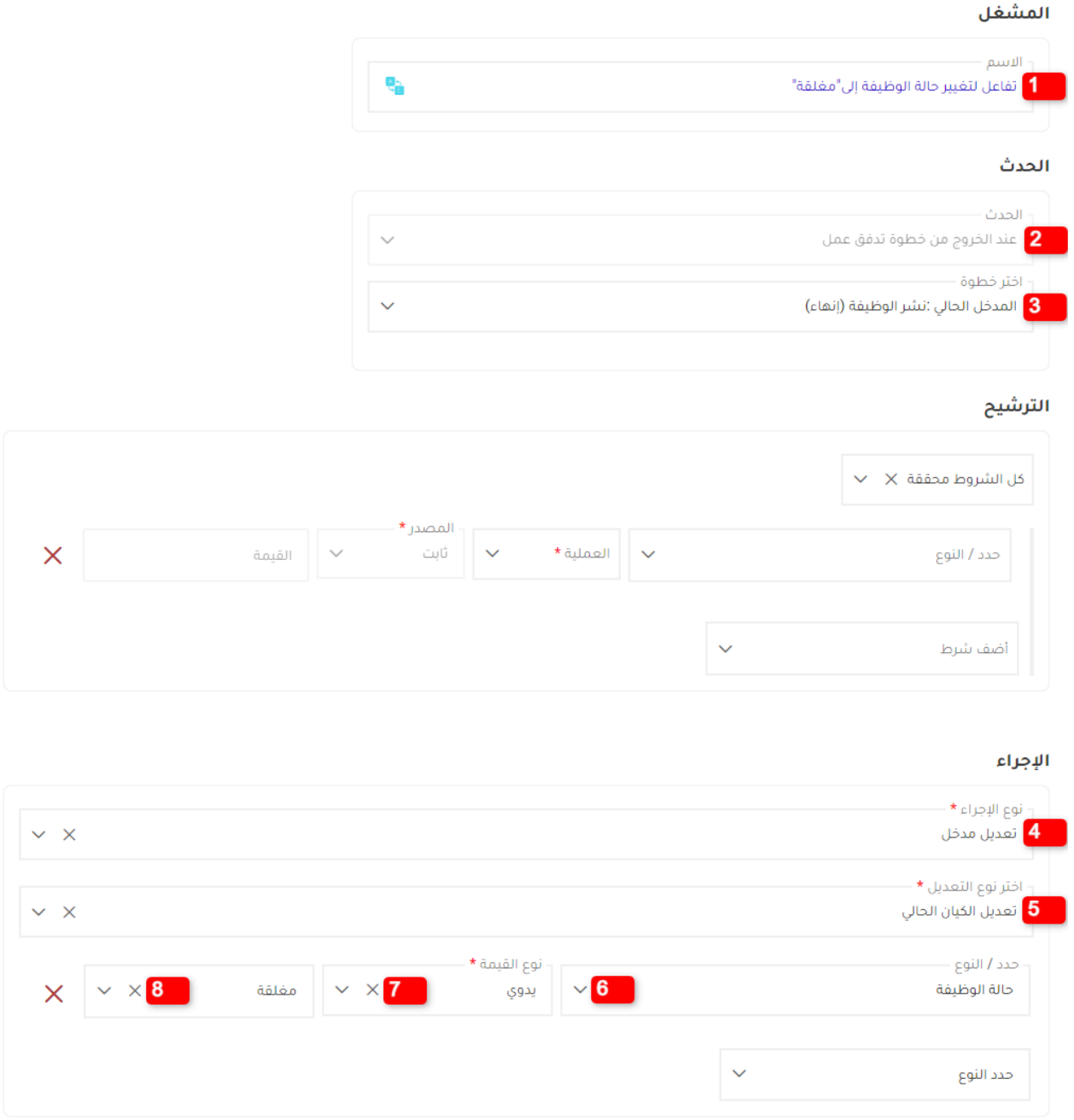
7. Reaction: set the job status to *Open*
    - **Event**: **On Workflow Step Exited**, *Job review* with *Approve*.
    - **Action**: **Update Data Entry**. In **Choose update mode**, choose **Self**; then choose the *Job status* field, **Value type Manual**, and the value *Open*.



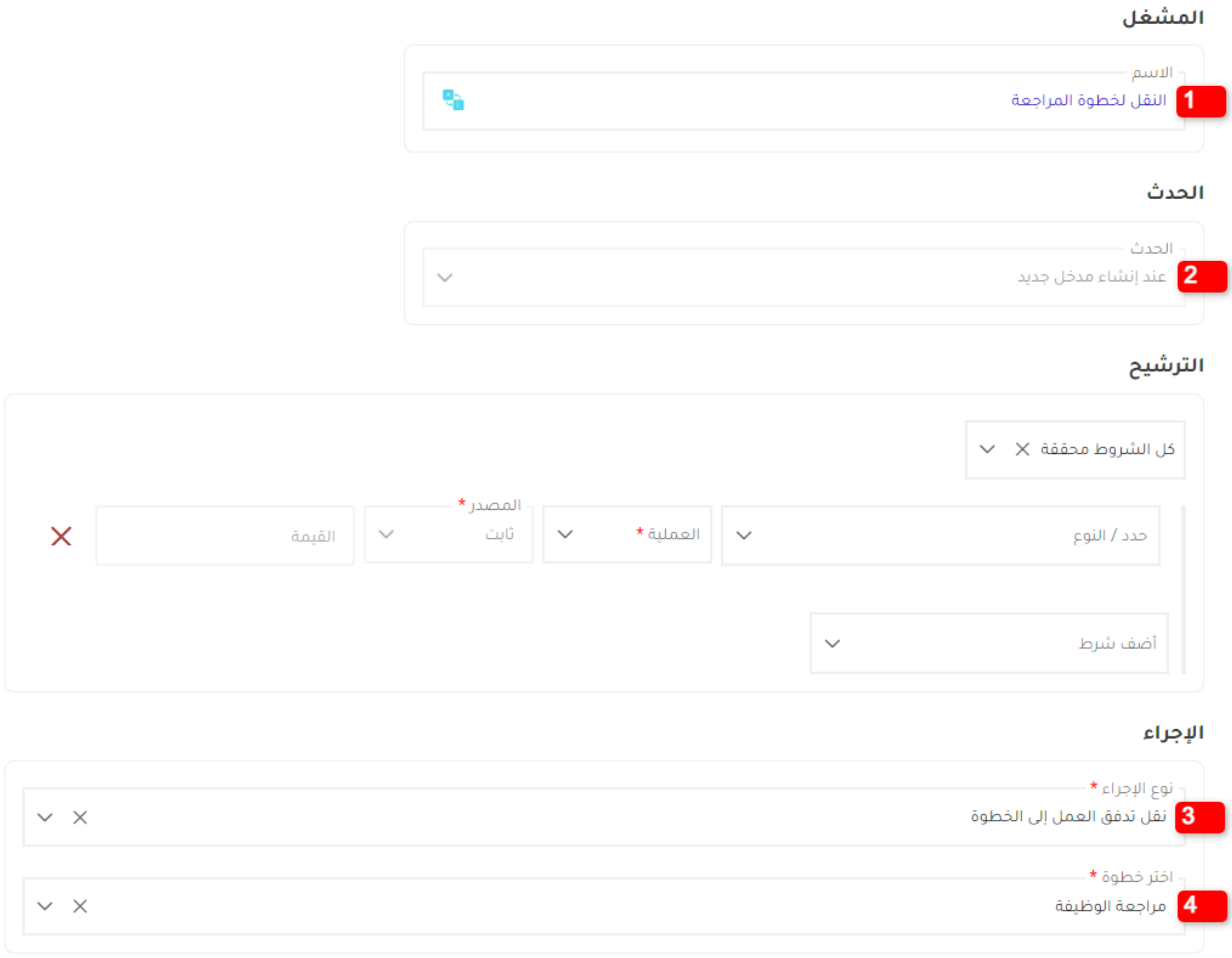
8. Reaction: set the job status to *Archived*
    - As above, with *Job review* and *Archive*, and the value *Archived*.



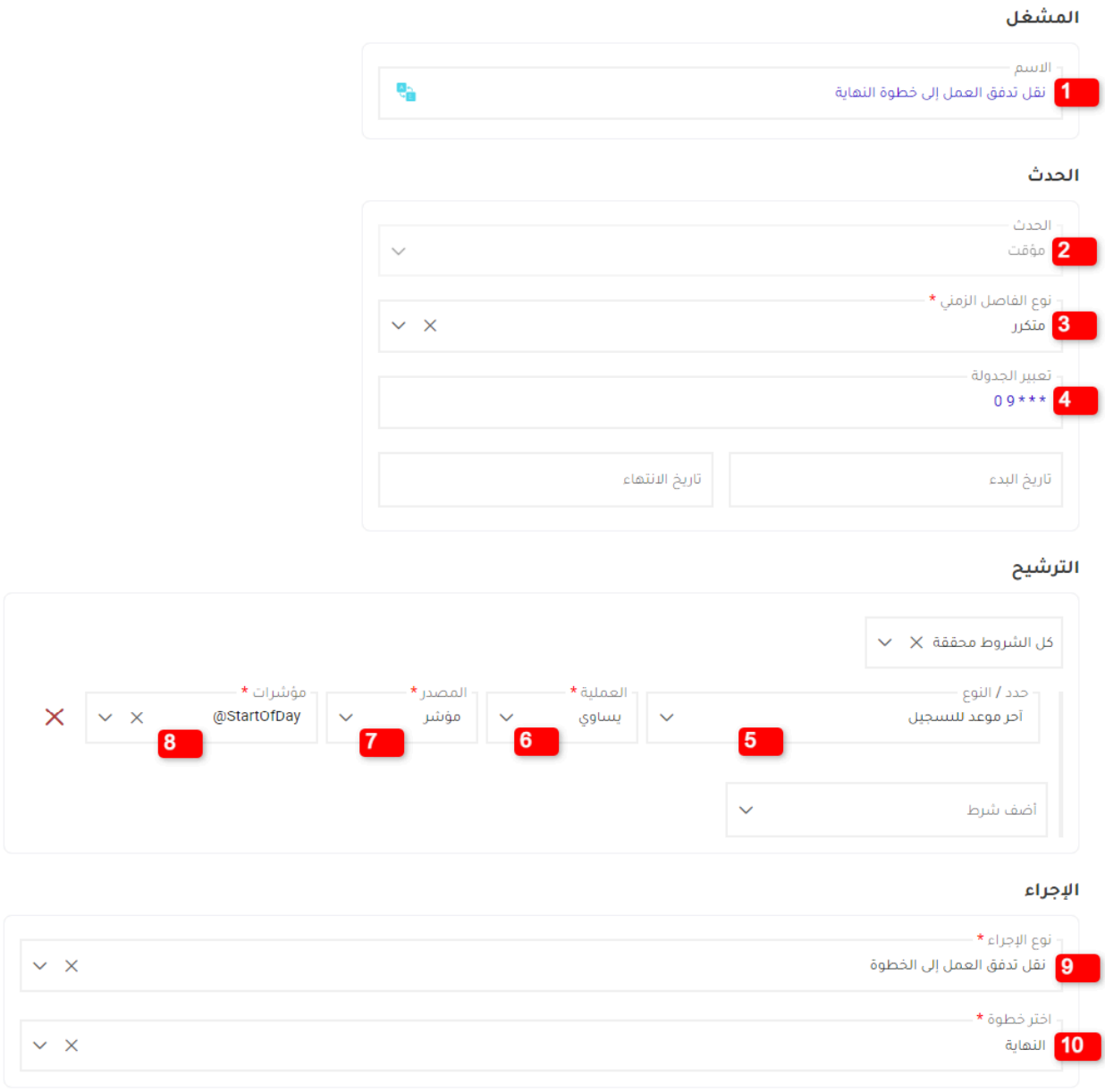
9. Reaction: set the job status to *Closed*
    - As above, with *Job publication* and its *Finish* action, and the value *Closed*.



10. Reaction: send a new job to review
    - **Event**: **On Entry Created**.
    - **Action**: **Move Workflow To Step**, and in **Select Step** choose *Job review*.



11. Reaction: close the job on its registration deadline
    - **Event**: **On Timer Reached**. **Interval type Repeat**, and **Cron Expression** `0 9 * * *` (every day at 09:00). Leave **Start Date** and **End Date** empty.
    - **Filter**: *Registration deadline* **Equal**, **Source Pointer**, and the pointer `@StartOfDay` (today).
    - **Action**: **Move Workflow To Step**, and in **Select Step** choose *Complete Step*.



12. Select **Save** again to make sure every reaction is saved.

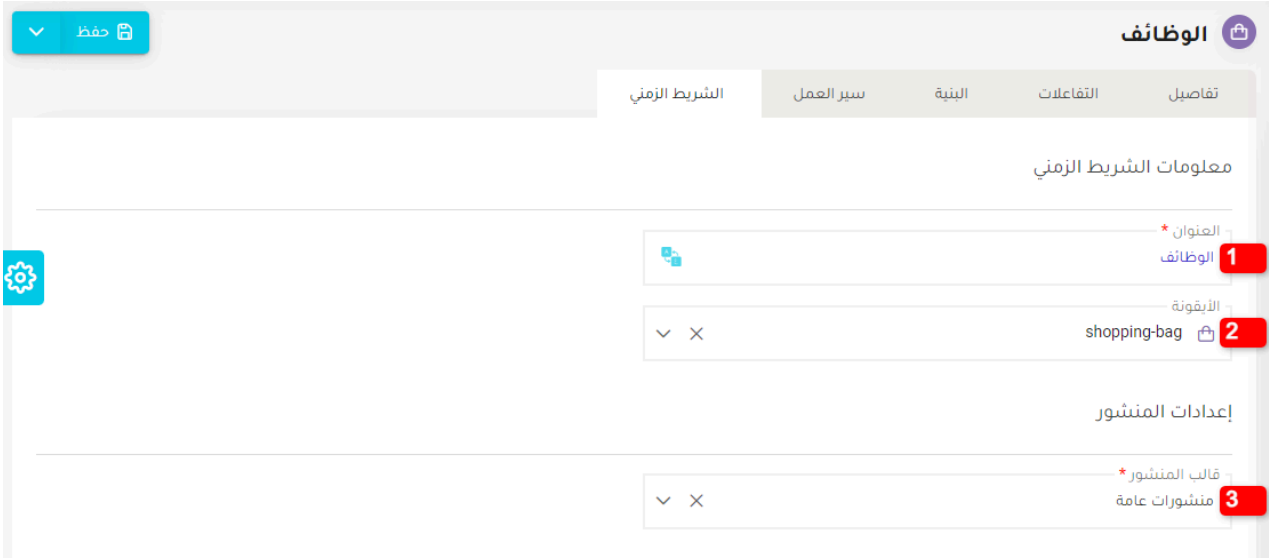
Add the timeline

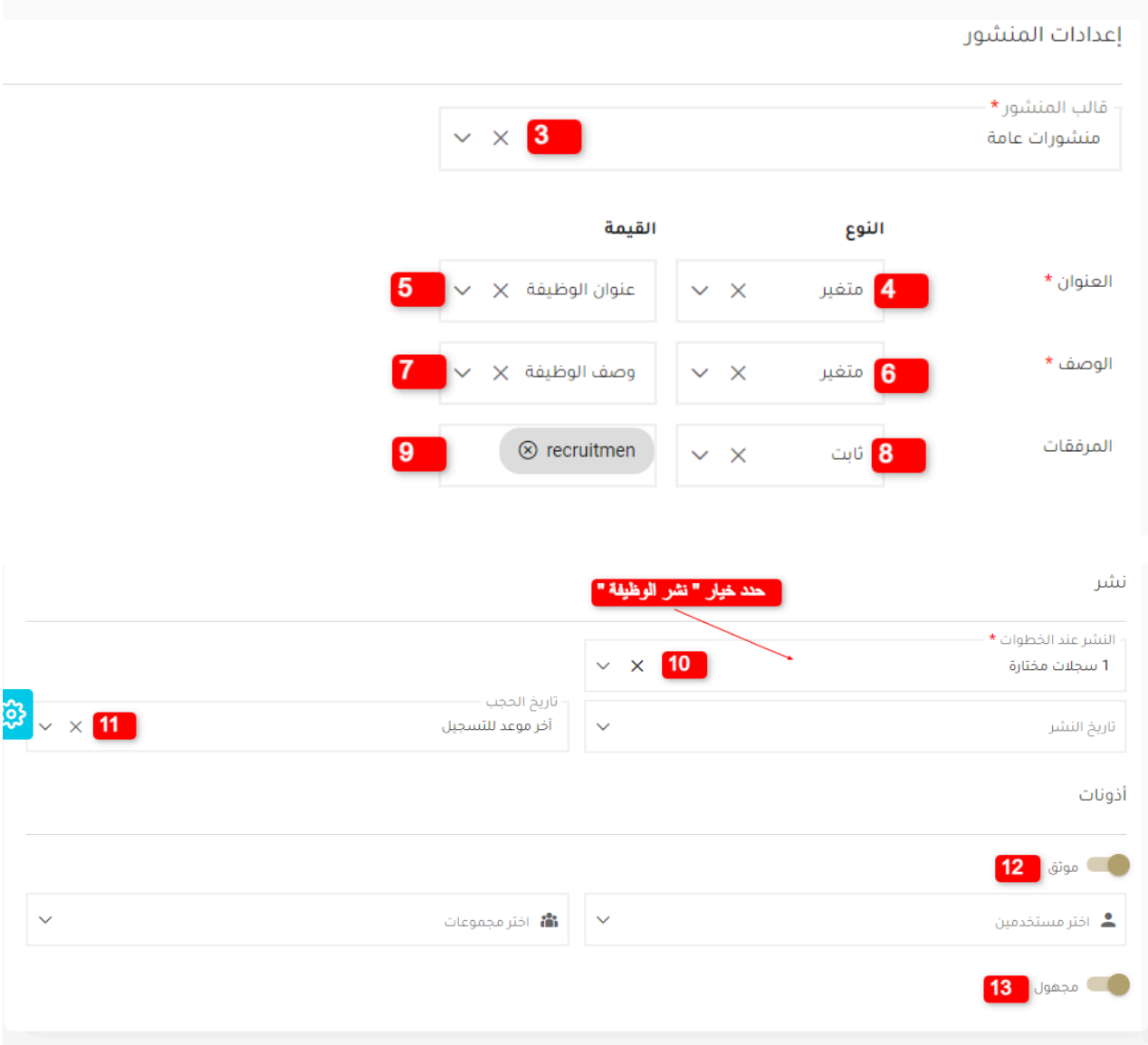
- The timeline publishes open jobs as posts and hides them again when they close. Set it up as follows:

  1. In the *Jobs* entity designer, open the **Timeline** tab and select the middle of the page to add a timeline.



2. Fill in:
   - **Timeline Information**: **Title** *Jobs* and an **Icon**.
   - **Post Configuration**: **Post Template** *General Post*. Set the post's title to **Field** › *Job title*, its description to **Field** › *Description*, and its attachments to **Fixed** with an image.
   - **Publish**: in **Steps to publish at**, choose *Job publication*; in **Unpublish Date**, choose *Registration deadline*.
   - **Permissions**: turn on **Authenticated** and **Anonymous**, so both employees and visitors see the jobs.





- For the attachment image, use any picture that suits job posts, for example:



3. Select **Save** at the top of the page to save the timeline.

Go deeper

More about each part: Workflows & approvals · Reactions & automation · Timeline

Views

A view is the screen people use to browse and enter an entity's data. These pages create the views of the recruitment system, one page per entity.

Receptions Views
→

Jobs Views
→

Job Application Views
→

Building the Receptions Views

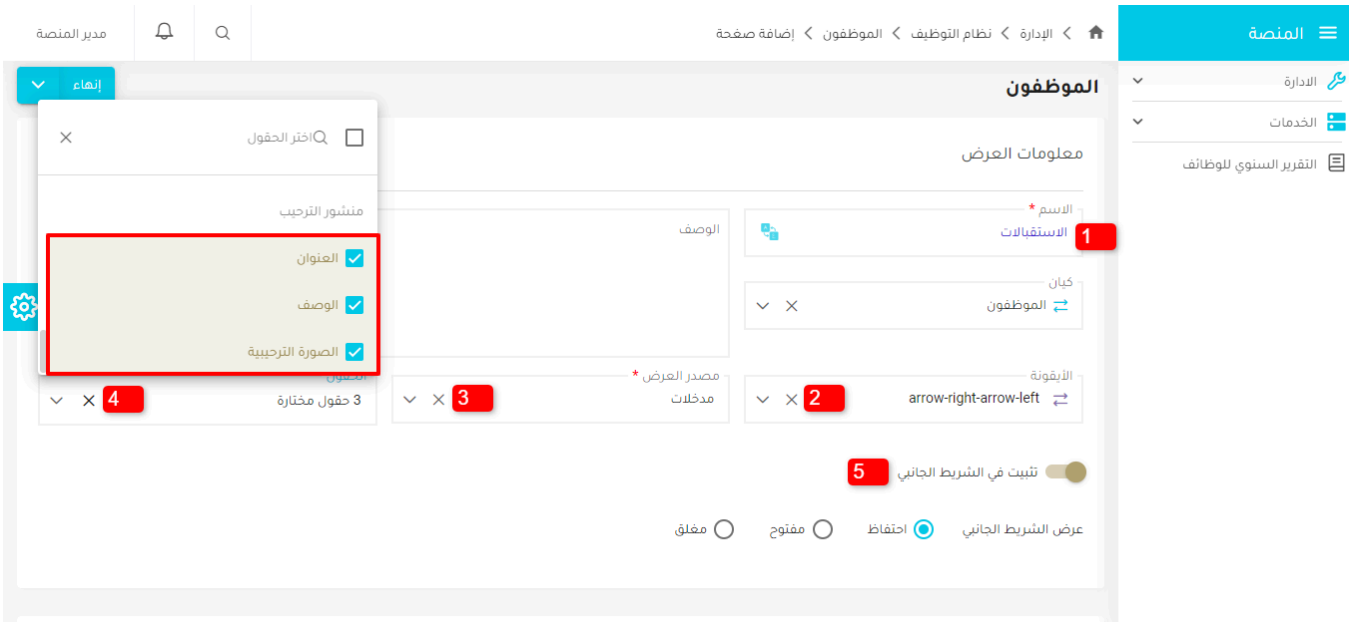
The Receptions entity gets two views: **View receptions**, where employees read the welcome posts, and **Create receptions**, where HR writes them. For more about views, see Views.

Build the “View receptions” view

- Open the *Receptions* entity.
- Open the **Views** tab.
- Select the **+** next to **New View**.



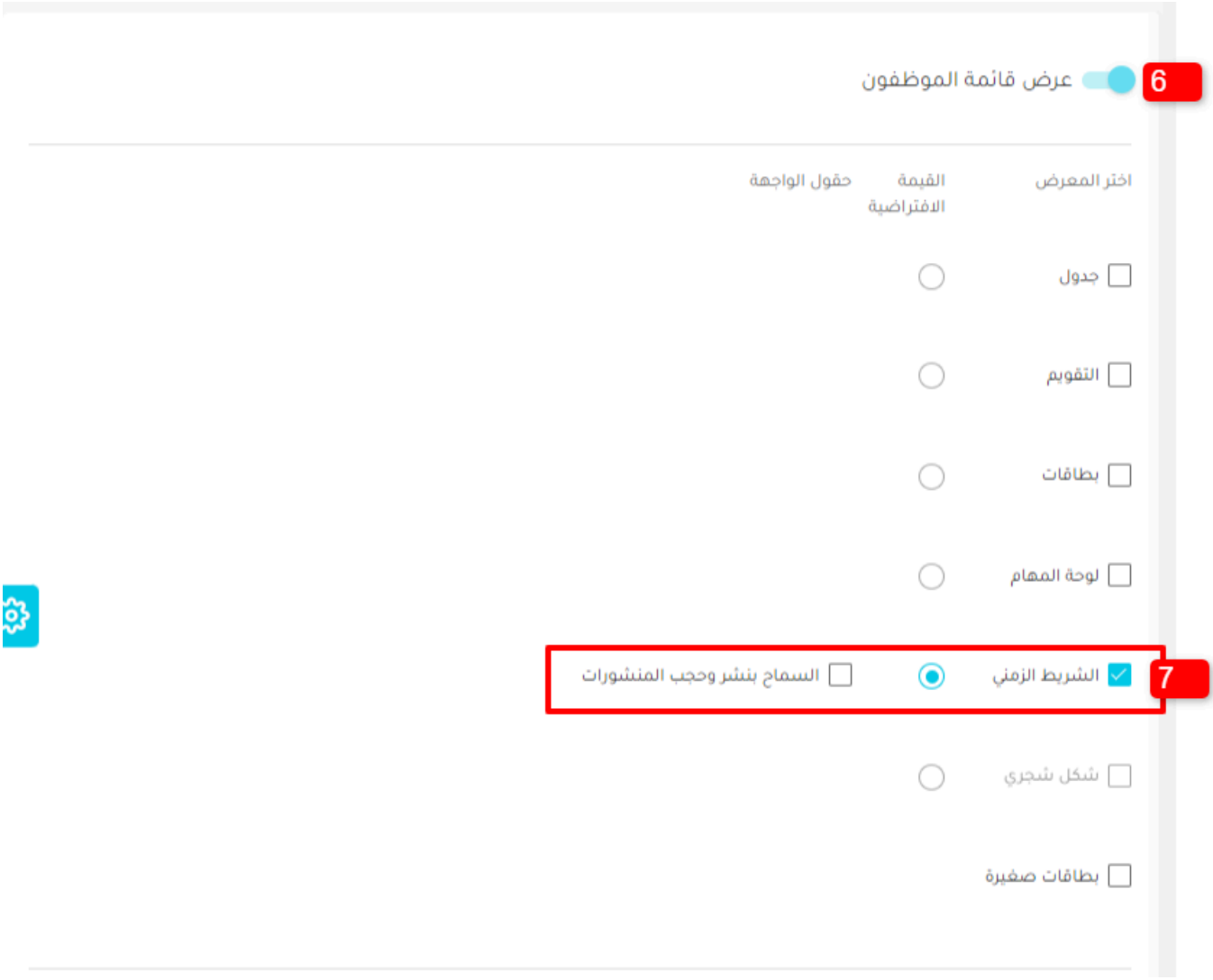
- Fill in **View information** with the details shown in the screenshot: a name (*View receptions*), an icon, **Source view Entries**, and all fields.



Entries list

Turn on **Entries list**. Under **Displays**:

- Select **Timeline view** and make it the **Default**.



Entries details

- Turn on **Entries details**, which shows a reception's full details, and select all fields.

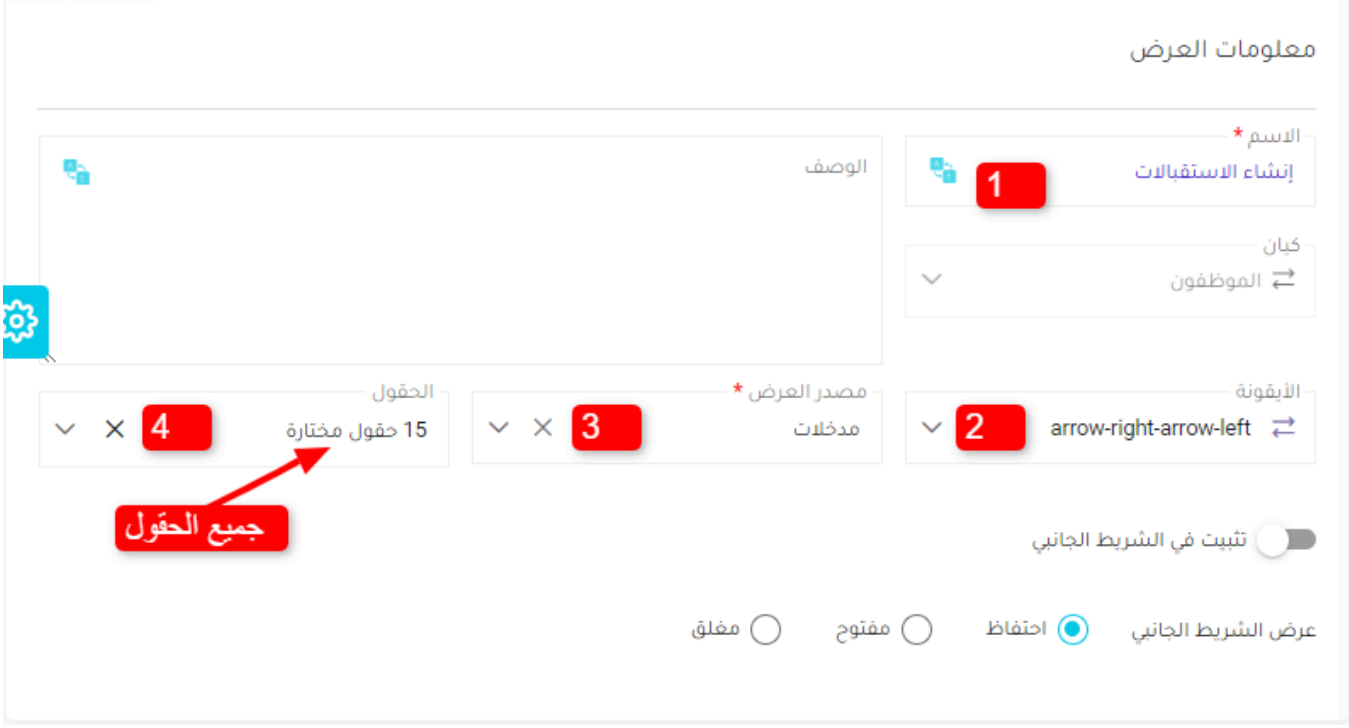


Save and publish the view

- Select **Save** at the top of the view designer.
- Select **Publish** on the view to publish it.

Build the “Create receptions” view

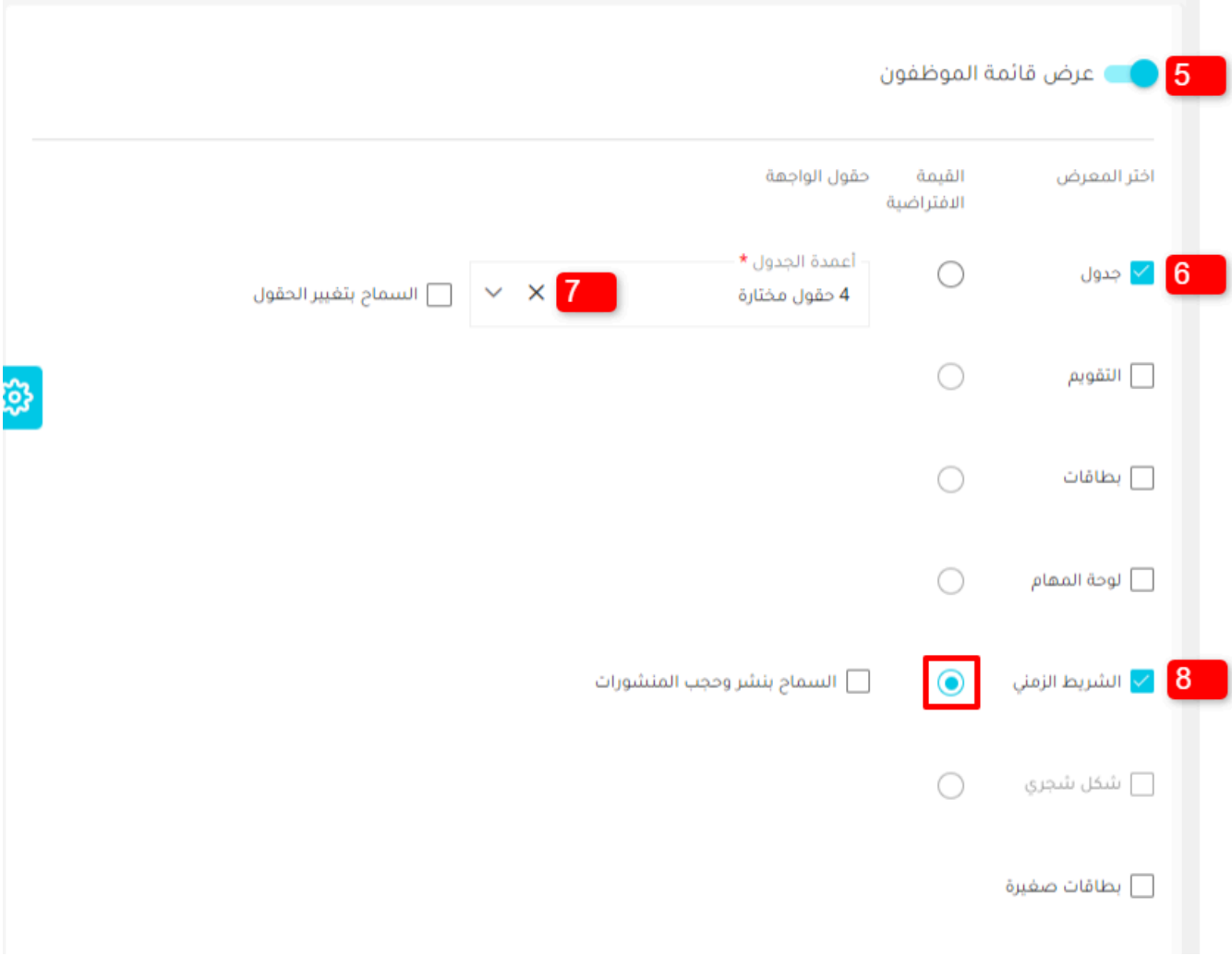
- Create a second view the same way and fill in **View information** with the details shown in the screenshot (*Create receptions*).



Entries list

Turn on **Entries list**. Under **Displays**:

- Select **Timeline view** and make it the **Default**.
- Also select **Table view**.



Adding and editing receptions

- Turn on **Creating new entries** and **Editing entries**, with the settings shown in the screenshot.



Save and publish the view

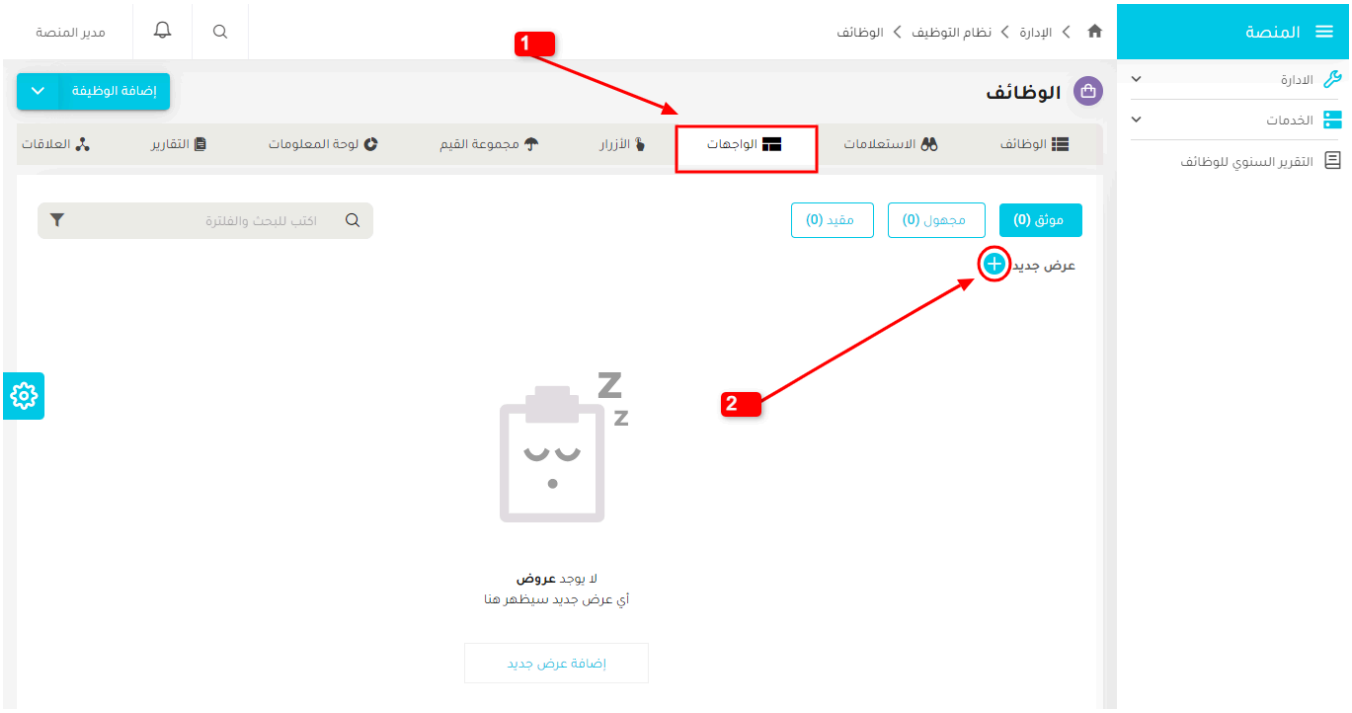
- Select **Save** at the top of the view designer.
- Select **Publish** on the view to publish it.

Building the Jobs Views

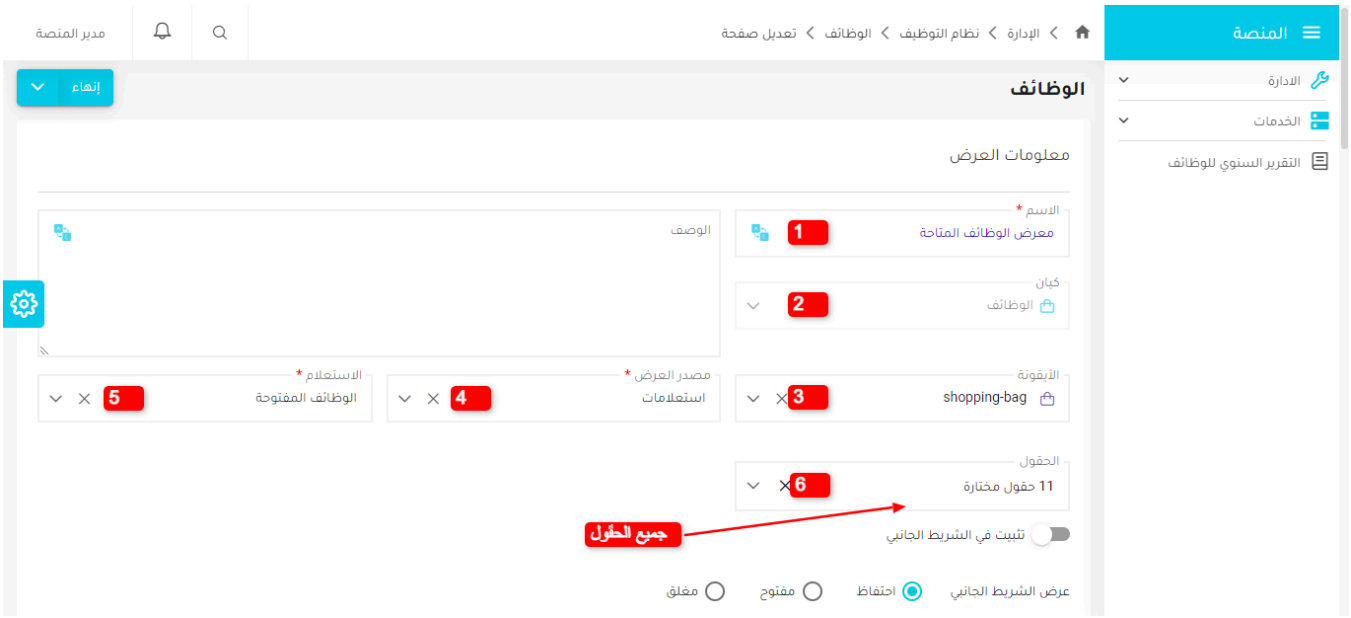
The Jobs entity gets two views: **Available jobs**, a public timeline of the open vacancies that anyone can see without signing in, and **All jobs**, where staff create and manage jobs. Both are created on the *Jobs* entity. For more about views, see Views.

Build the “Available jobs” view

- Open the *Jobs* entity.
- Open the **Views** tab.
- Select the **+** next to **New View**.



- In **View information**, fill in:
  - **Name**: *Available jobs*.
  - **Icon**: pick an icon.
  - **Source view**: **Queries**, and in **Query** choose *Open jobs*.
  - **Fields**: select all fields.
  - Leave **Pin to sidebar** off.



Access Permission

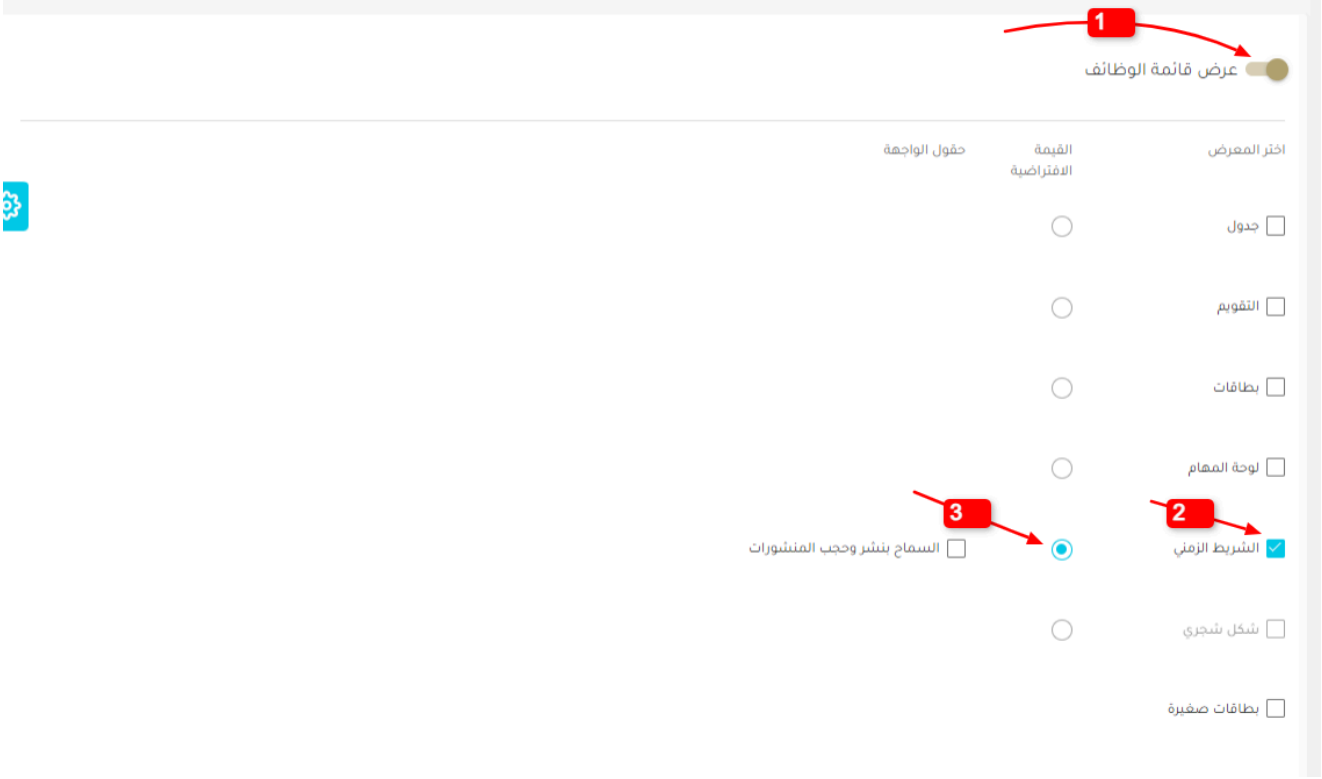
- Turn on **Anonymous**, so the view needs no sign-in.



Entries list

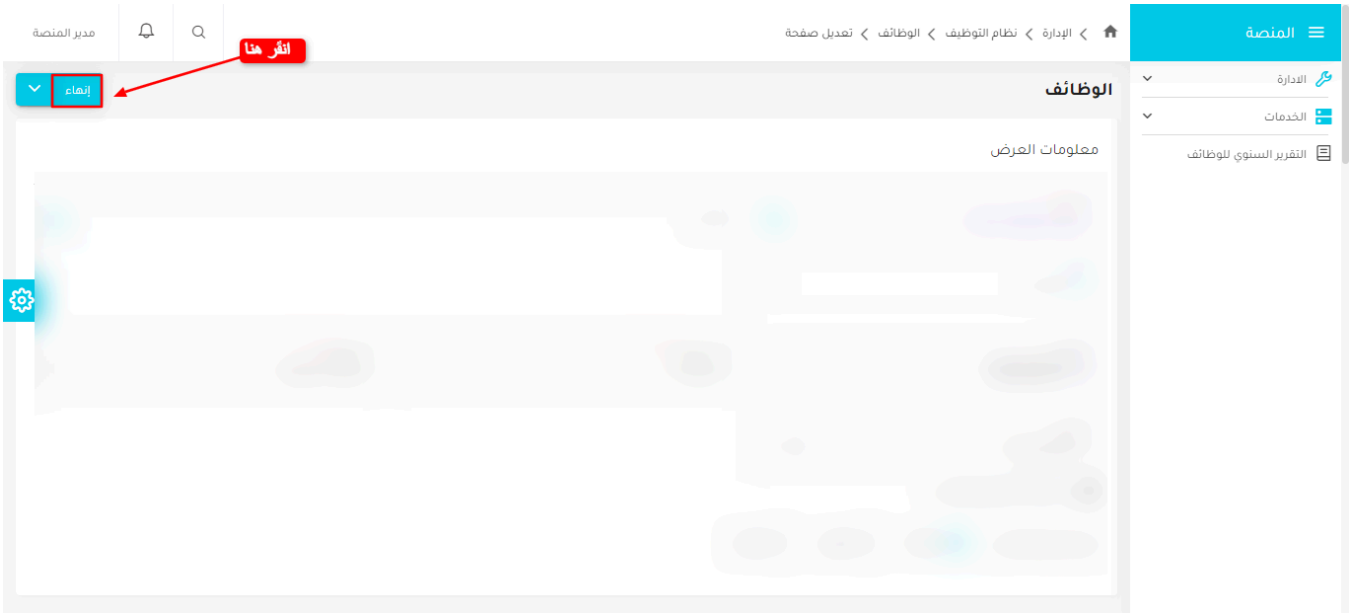
Turn on **Entries list**. This section sets how the list of entries is displayed:

- Under **Displays**, select **Timeline view** and make it the **Default**.

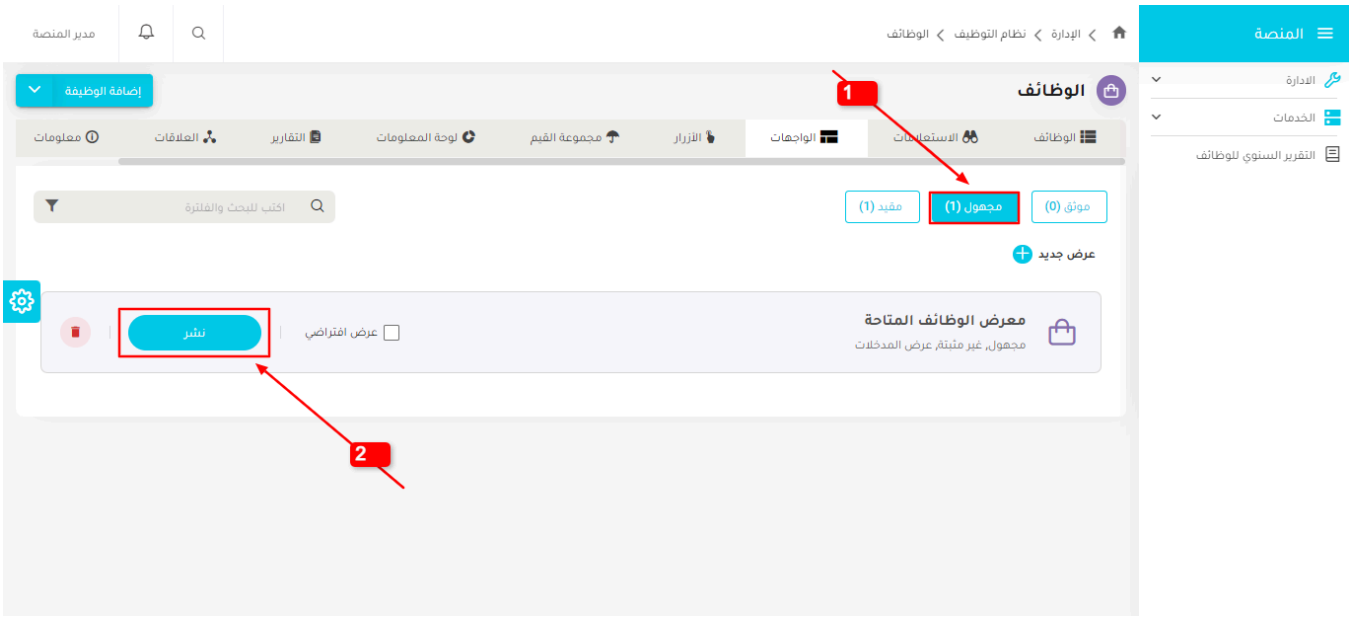


Save the view

- Select **Save** at the top of the view designer.

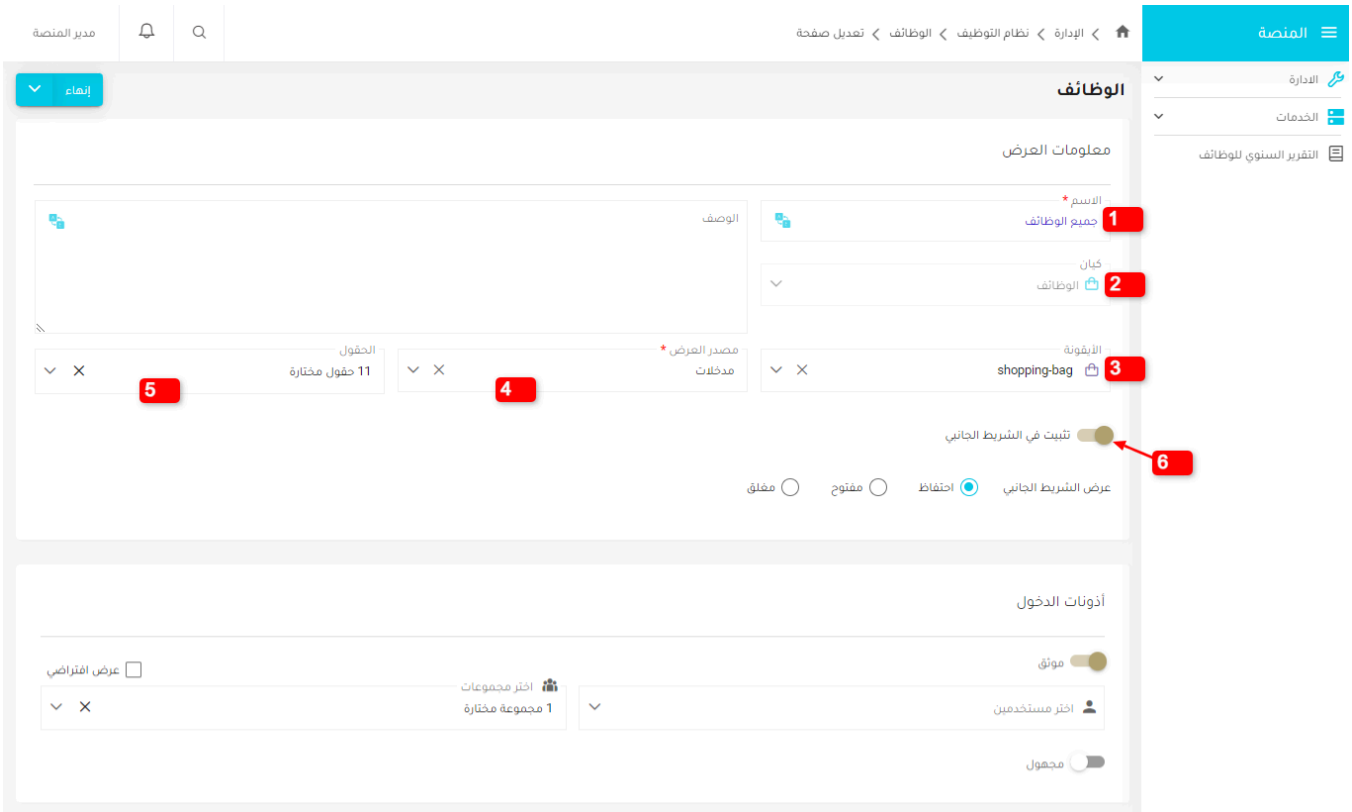


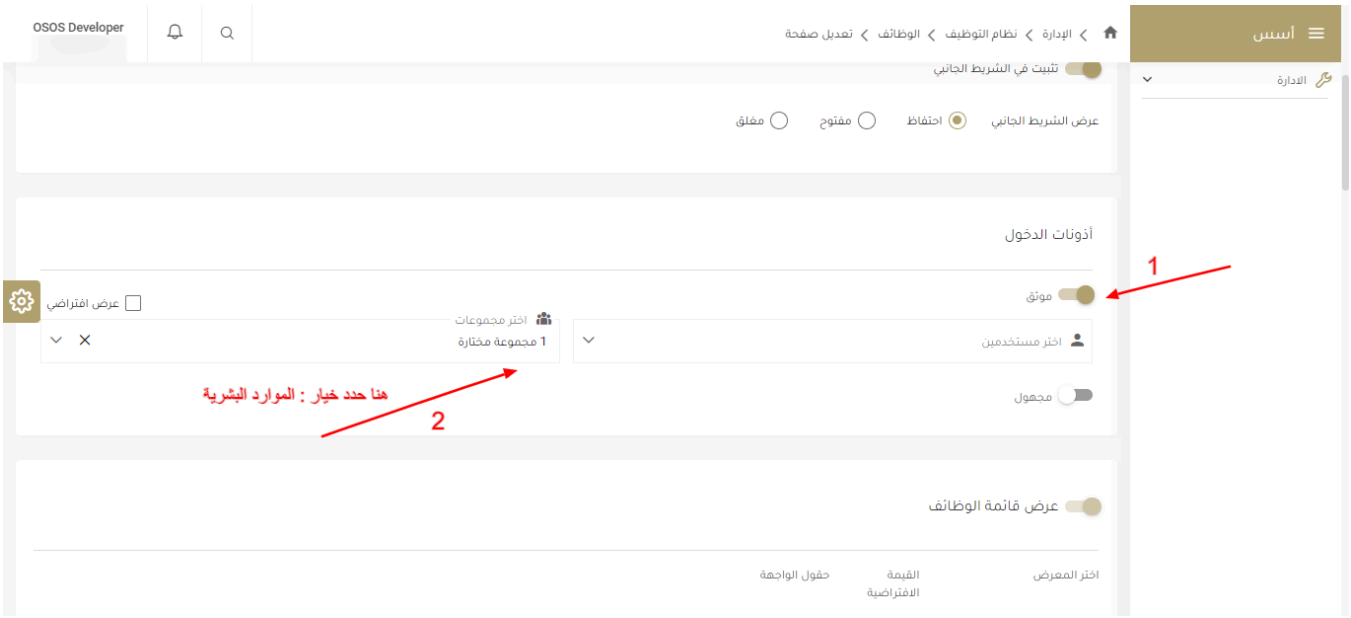
- Back in the list of views, select **Publish** on the view so people can open it.

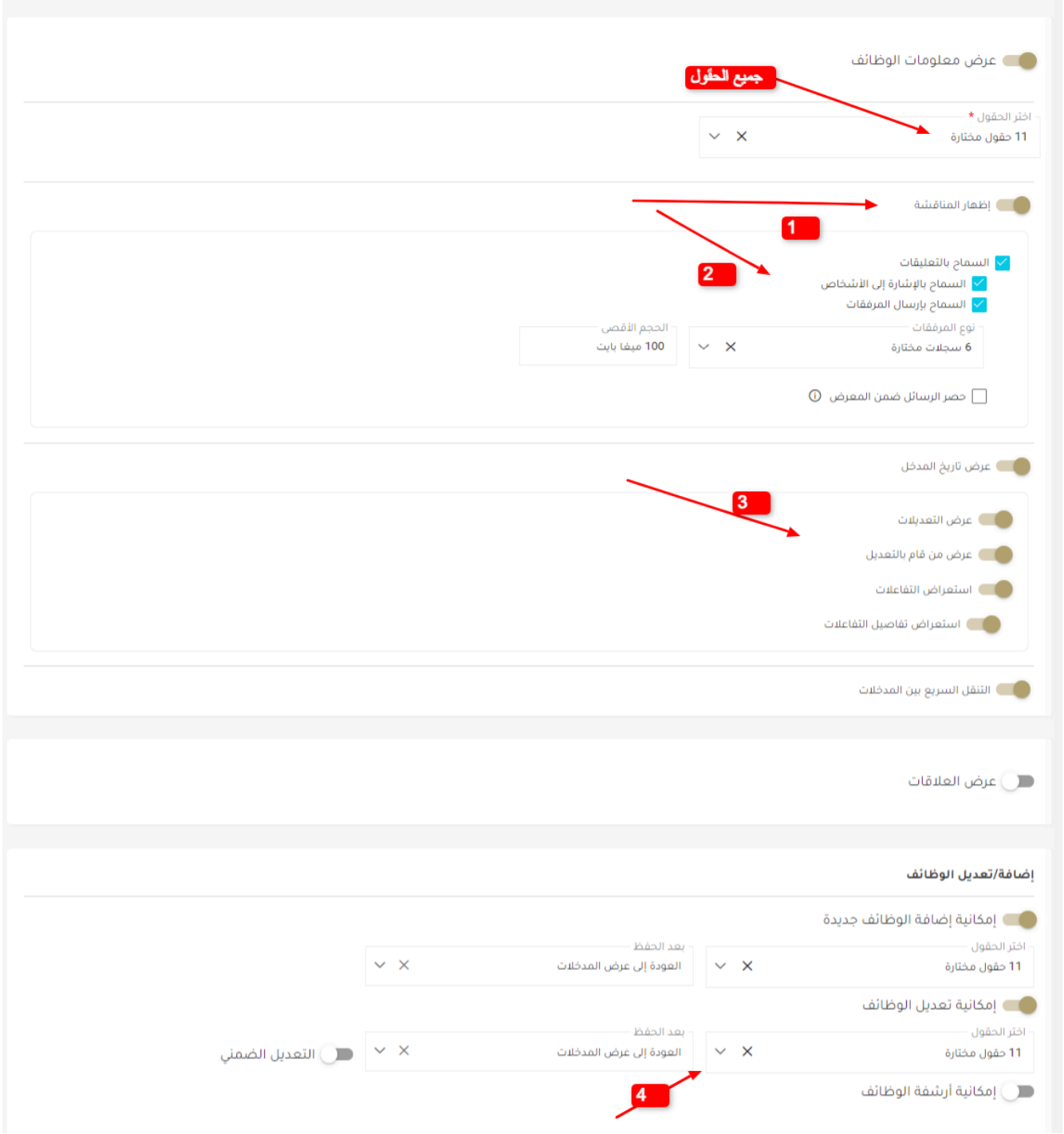


Build the “All jobs” view

- Create a second view the same way, with these settings:
  - **Name**: *All jobs*. **Source view**: **Entries**. **Fields**: all fields. Turn on **Pin to sidebar**.
  - **Access Permission**: turn on **Authenticated** and, in **Select groups**, choose the groups that create and manage jobs: *HR*, and *IT* (the home page’s *Create a new job* link opens this view for the IT team).
  - Turn on **Entries details** and select all fields. You can also turn on **Discussion** (comments, mentions and attachments) and **Entry history**.
  - In the section for adding and editing entries, turn on **Creating new entries** and **Editing entries**, choose all fields for each, and set **After Save** to **Go to entries list**. Leave archiving off.
- Save and publish the view.




OSOS Developer
أسس
الإدارة
نظام التوظيف
الوظائف
تعديل صفحة
الادارة
تثبيت في الشريط الجانبي
عرض الشريط الجانبي
احتفاظ
مفتوح
مغلق
أذونات الدخول
موثق
1
عرض افتراضي
اختر مجموعات
1 مجموعة مختارة
اختر مستخدمين
مجهول
هنا حدد خيار : الموارد البشرية
2
عرض قائمة الوظائف
اختر المعرض
القيمة الافتراضية
حقول الواجهة


عرض معلومات الوظائف
جميع الحقول
اختر الحقول *
11 حقول مختارة
إظهار المناقشة
1
2
السماح بالتعليقات
السماح بالإشارة إلى الأشخاص
السماح بإرسال المرفقات
نوع المرفقات
6 سجلات مختارة
الحجم الأقصى
100 ميغا بايت
حصر الرسائل ضمن المعرض
عرض تاريخ المدخل
3
عرض التعديلات
عرض من قام بالتعديل
استعراض التفاعلات
استعراض تفاصيل التفاعلات
التنقل السريع بين المدخلات
عرض العلاقات
إضافة/تعديل الوظائف
إمكانية إضافة الوظائف جديدة
اختر الحقول
11 حقول مختارة
بعد الحفظ
العودة إلى عرض المدخلات
إمكانية تعديل الوظائف
اختر الحقول
11 حقول مختارة
بعد الحفظ
العودة إلى عرض المدخلات
التعديل الضمني
4
إمكانية أرشفة الوظائف

Building the Job Application Views

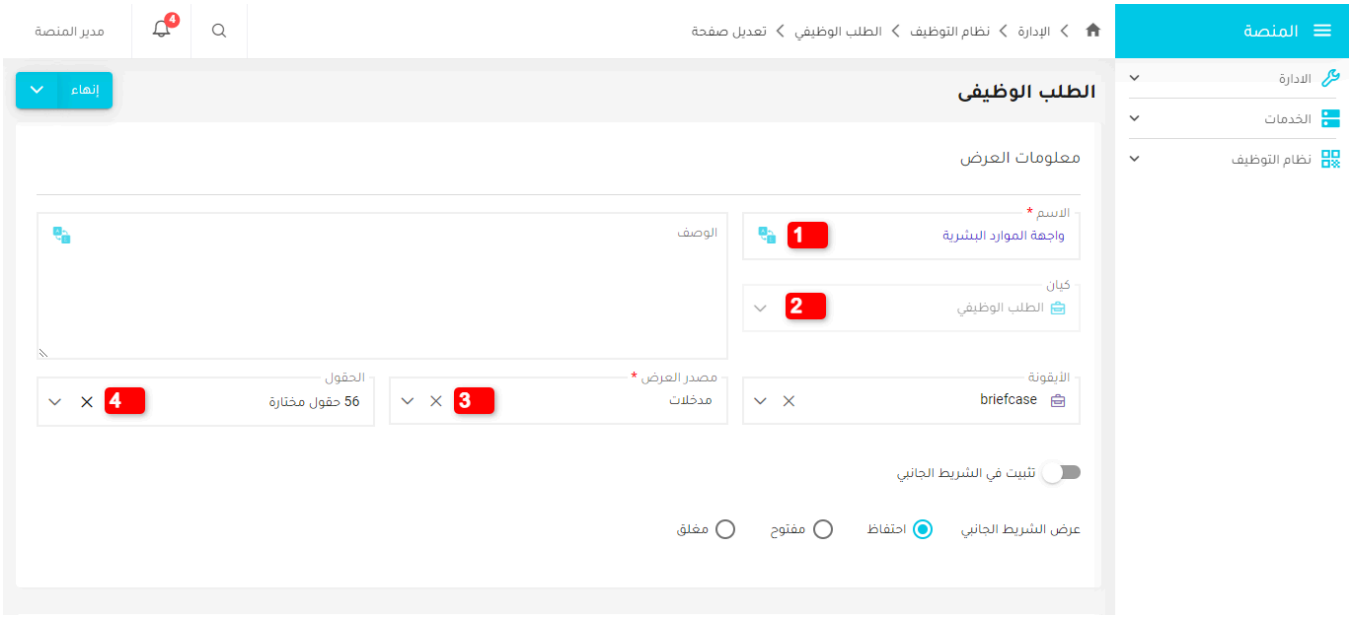
HR works on applications in a view of the *Job Application* service. Because it shows candidates' personal data, this view is open to HR only. For more about views, see Views.

Build the "HR view"

- Open the *Job Application* service as an entity (from the module's **Entities** tab).
- Open the **Views** tab.
- Select the **+** next to **New View**.

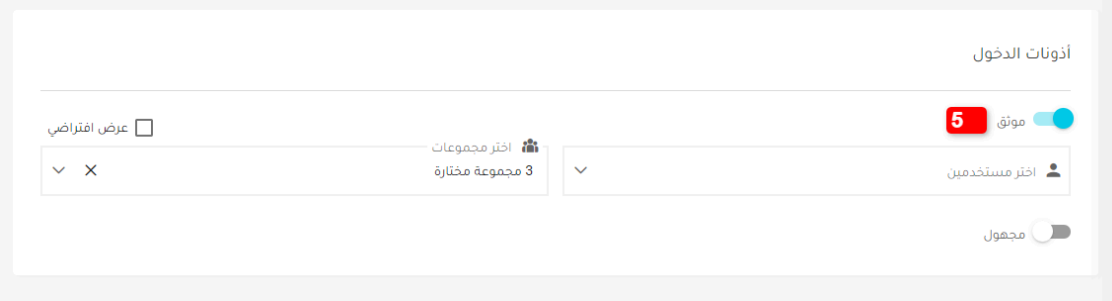


- In **View information**, fill in:
  - **Name**: *HR view*.
  - **Icon**: pick an icon.
  - **Source view**: **Entries**.
  - **Fields**: select all fields.



Access Permission

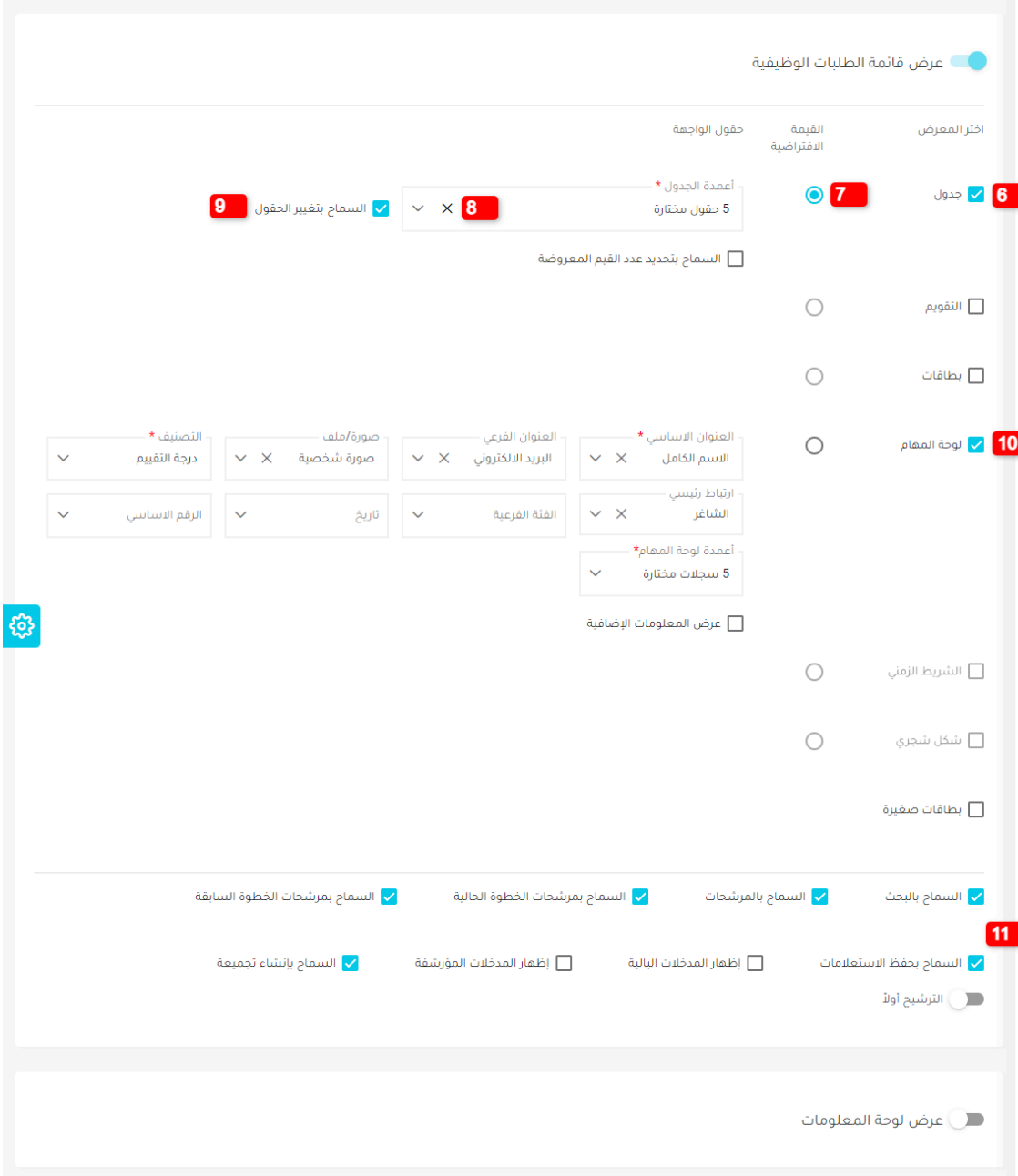
- Turn on **Authenticated** and, in **Select groups**, choose *HR* and any other group that handles applications. Leave **Anonymous** off: this view shows candidates' personal data.



Entries list

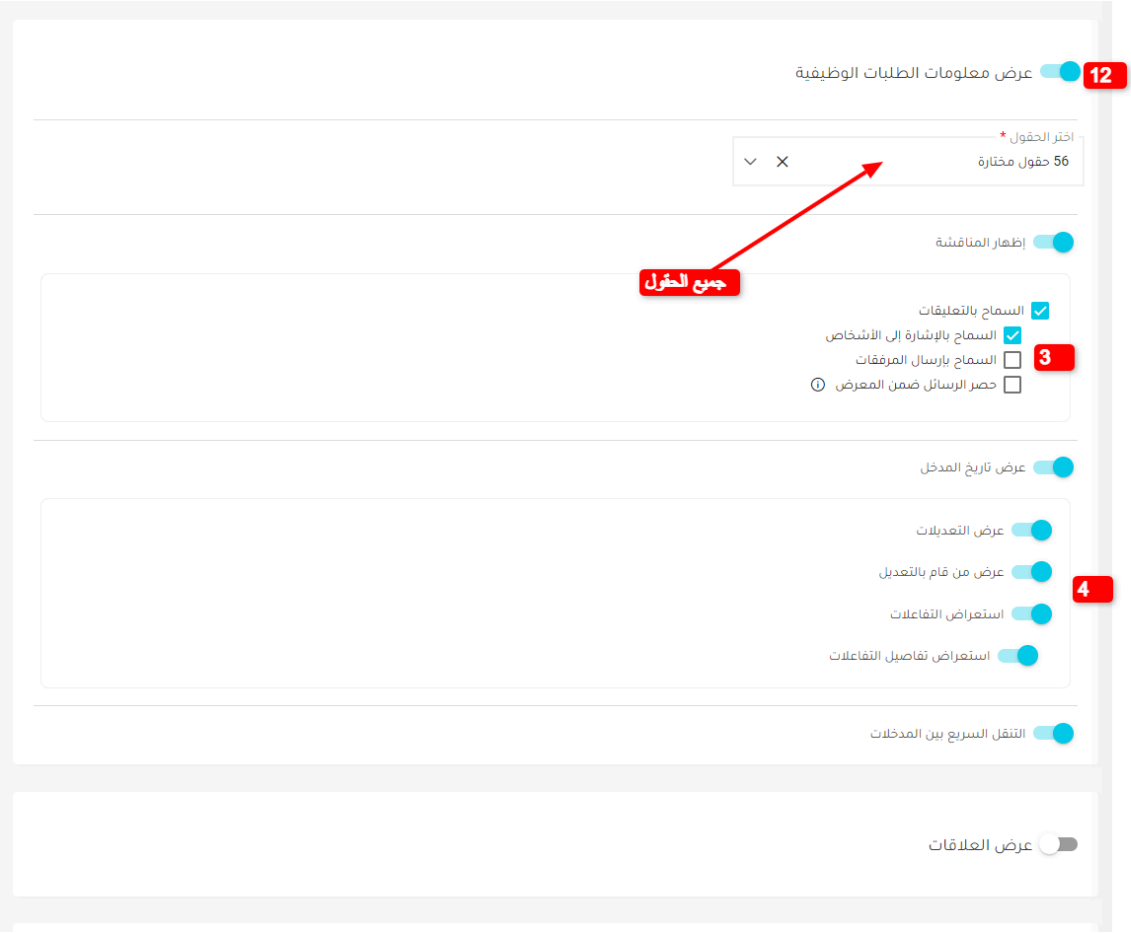
Turn on **Entries list**. Under **Displays**:

- Select **Table view** and make it the **Default**.
- Also select **Kanban view**.



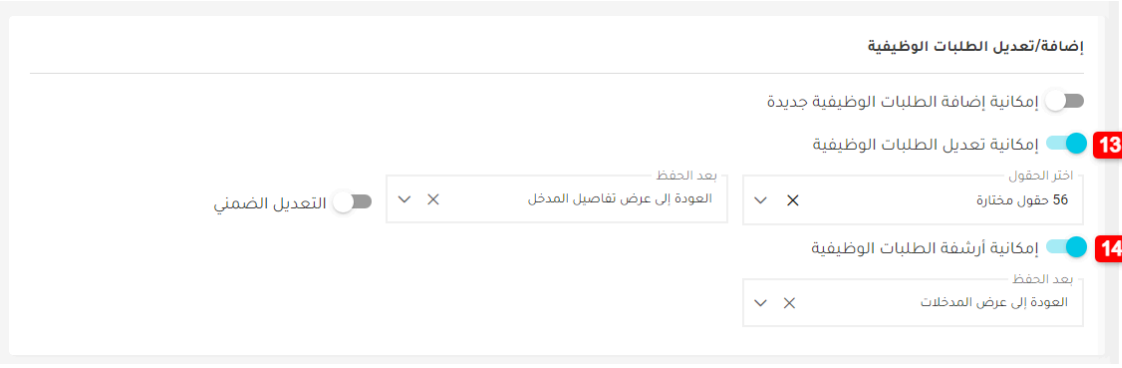
Entries details

- Turn on **Entries details**, which shows an application's full details, and select all fields.



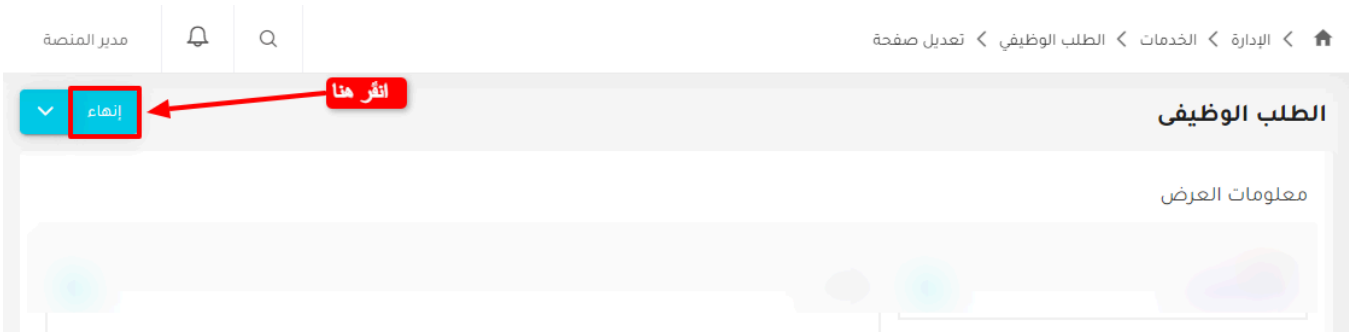
Adding and editing applications

- Turn on **Creating new entries** and **Editing entries**, with the settings shown in the screenshot.



Save the view

- Select **Save** at the top of the view designer.


مدير المنصة
الإدارة > الخدمات > الطلب الوظيفي > تعديل صفحة
انقر هنا
إنهاء
الطلب الوظيفى
معلومات العرض

Dashboards

Summaries give HR the key figures at a glance. You add them on the **Dashboard** tab of an entity, and later show some of them on the system's home page. For more about dashboards and summaries, see Dashboards & summaries.

Jobs dashboard

1. Open the *Jobs* entity.
2. Open the **Dashboard** tab.
3. Select **Add Summary**.



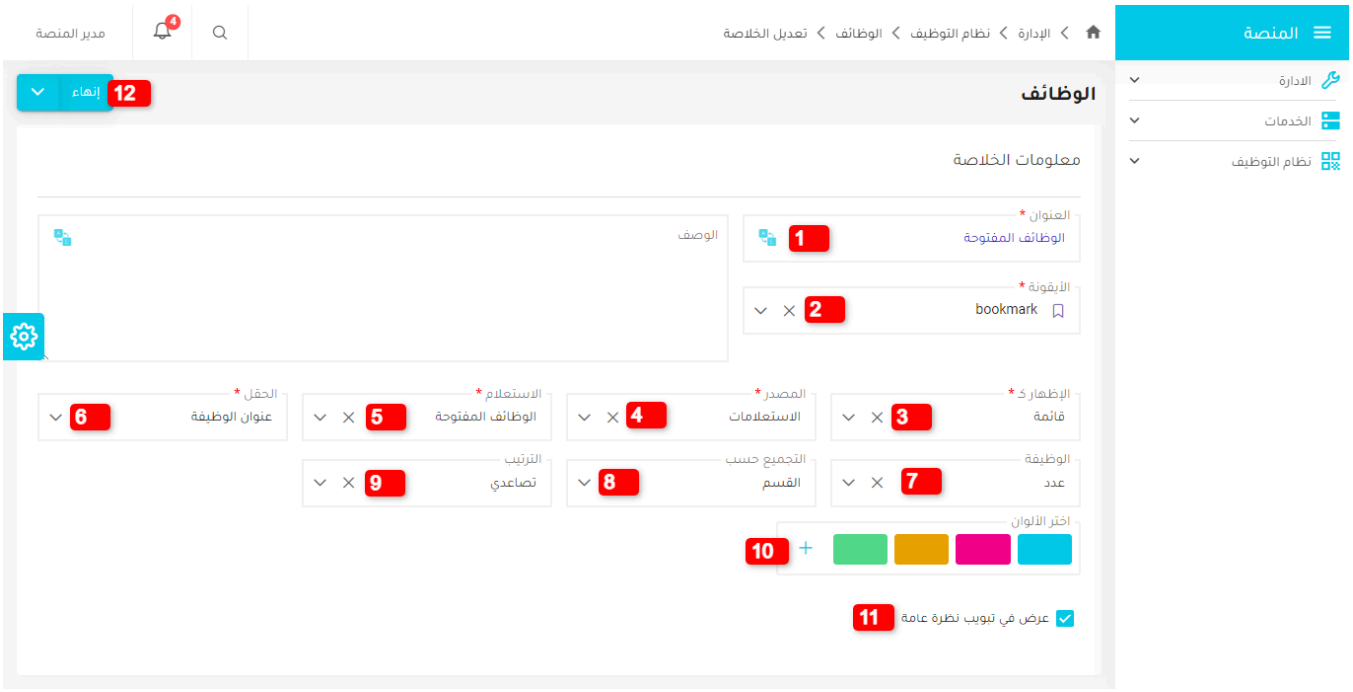
4. The **Summary Information** form opens.

Open jobs list

5. Fill in the summary:

- **Title** (required): *Open jobs*.
- **Icon** (required): *bookmark*.
- **Show as** (required): **List**.
- **Source** (required): **Queries**.
- **Query** (required): *Open jobs*.
- **Field** (required): *Job title*.
- **Function**: **Count**.
- **Group by**: *Department*.
- **Order by**: **Ascending**.
- **Select colors**: pick the colours you like.
- Turn on **Display in overview tab**.

6. Select **Done**, then confirm to save the summary.

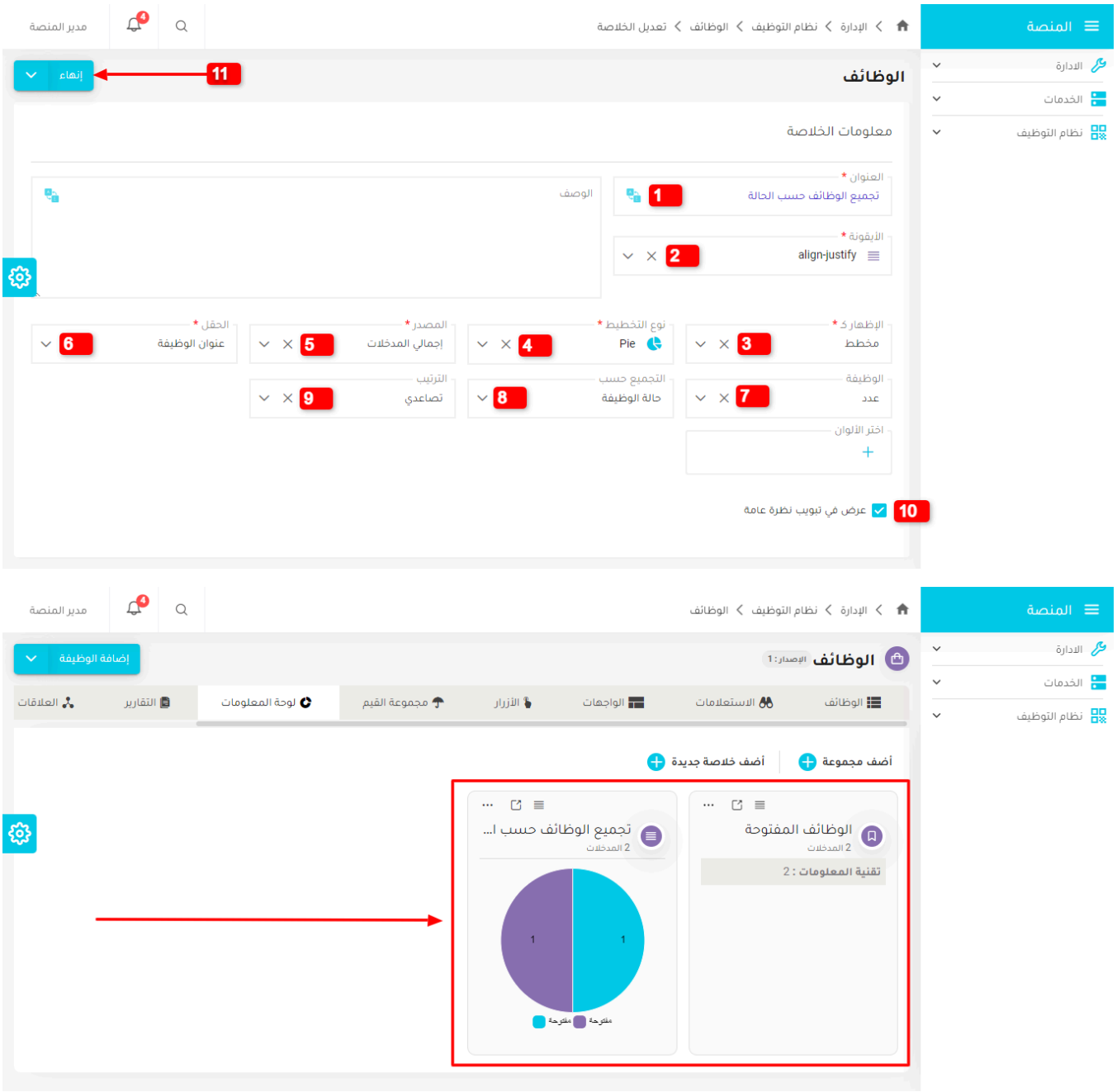


Jobs by status chart

5. Add a second summary:

- **Title** (required): *Jobs by status*.
- **Icon** (required): *align-justify*.
- **Show as** (required): **Chart**.
- **Chart type** (required): **Doughnut**.
- **Source** (required): **All entries**.
- **Field** (required): *Job title*.
- **Function**: **Count**.
- **Group by**: *Job status*.
- **Order by**: **Ascending**.
- **Select colors**: pick the colours you like.
- Turn on **Display in overview tab**.

6. Select **Done**, then confirm to save the summary.



Job Application dashboard

1. Open the *Job Application* service as an entity.
2. Open the **Dashboard** tab.
3. Select **Add Summary**.
4. The **Summary Information** form opens.

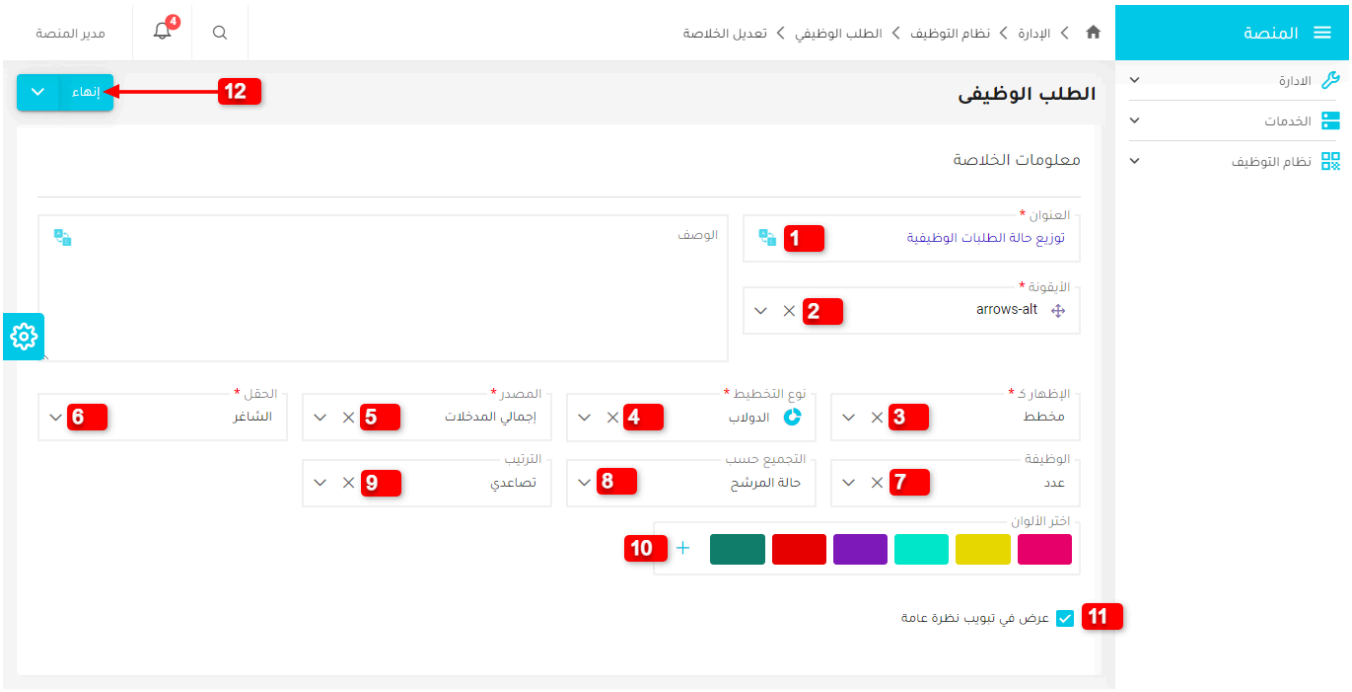
Applications by status chart

5. Fill in the summary:

- **Title** (required): *Applications by status*.
- **Icon** (required): *arrows-alt*.
- **Show as** (required): **Chart**.
- **Chart type** (required): **Doughnut**.

- **Source** (required): **All entries**.
- **Field** (required): *Vacancy*, from the *Candidate information* block.
- **Function**: **Count**.
- **Group by**: *Candidate status*.
- **Order by**: **Ascending**.
- **Select colors**: pick the colours you like.
- Turn on **Display in overview tab**.

6. Select **Done**, then confirm to save the summary.

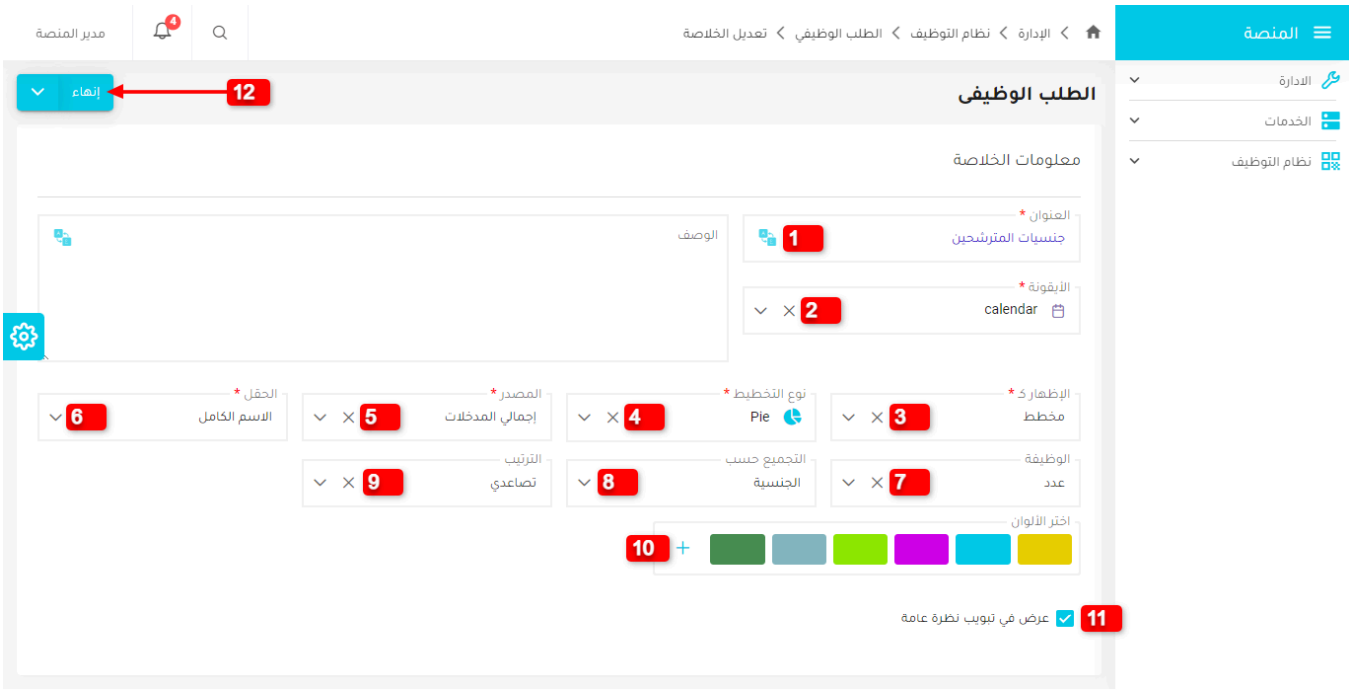


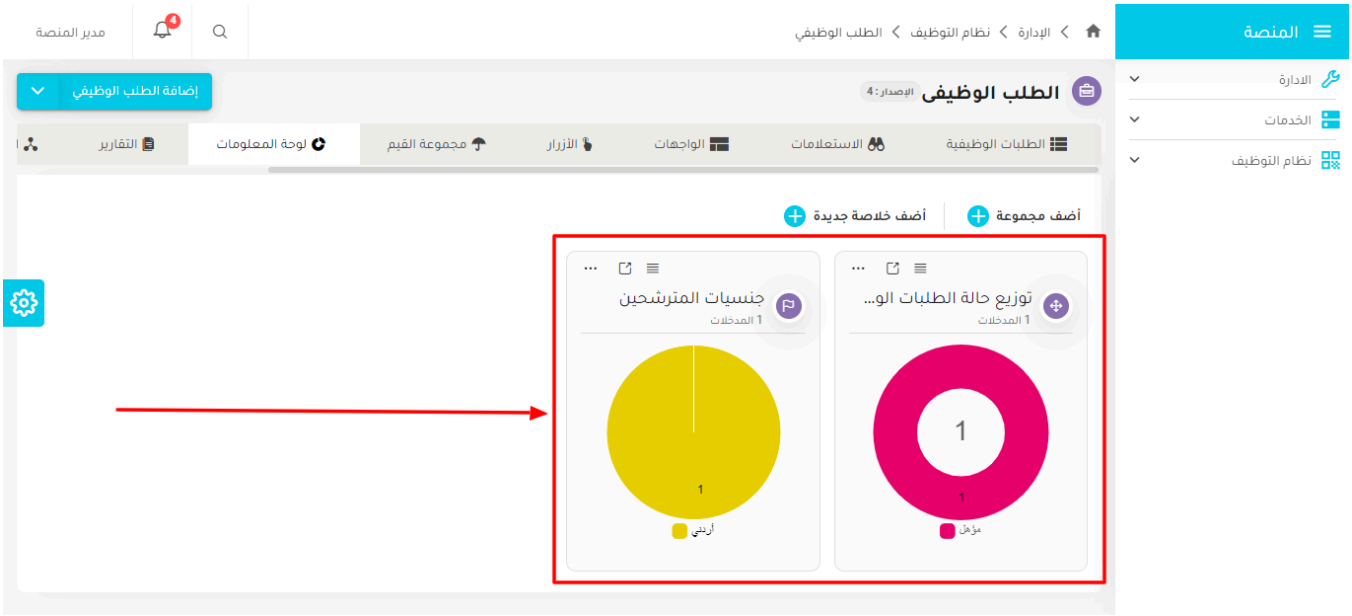
Candidates by nationality chart

5. Add a second summary:

- **Title** (required): *Candidate nationalities*.
- **Icon** (required): *flag*.
- **Show as** (required): **Chart**.
- **Chart type** (required): **Pie**.
- **Source** (required): **All entries**.
- **Field** (required): *Full name*, from the *Candidate information* block.
- **Function**: **Count**.
- **Group by**: *Nationality*.
- **Order by**: **Ascending**.
- **Select colors**: pick the colours you like.
- Turn on **Display in overview tab**.

6. Select **Done**, then confirm to save the summary.


المنصة
الادارة
الخدمات
نظام التوظيف
الإدارة > نظام التوظيف > الطلب الوظيفي > تعديل الخلاصة
مدير المنصة
الطلب الوظيفي
إنهاء
12
معلومات الخلاصة
العنوان *
جنسيات المترشحين
1
الوصف
الأيقونة *
calendar
2
الإظهار ك *
مخطط
3
نوع التخطيط *
Pie
4
المصدر *
إجمالي المدخلات
5
الحقل *
الاسم الكامل
6
الوظيفة
عدد
7
التجميع حسب
الجنسية
8
الترتيب
تصاعدي
9
اختر الألوان
10
عرض في تبويب نظرة عامة
11


المنصة
الادارة
الخدمات
نظام التوظيف
الإدارة > نظام التوظيف > الطلب الوظيفي
مدير المنصة
الطلب الوظيفي
الإصدار: 4
إضافة الطلب الوظيفي
الطلبات الوظيفية
الاستعلامات
الواجهات
الأزرار
مجموعة القيم
لوحة المعلومات
التقارير
أضف مجموعة
أضف خلاصة جديدة
توزيع حالة الطلبات الو...
1 المدخلات
1
مؤهل
جنسيات المترشحين
1 المدخلات
1
أردني

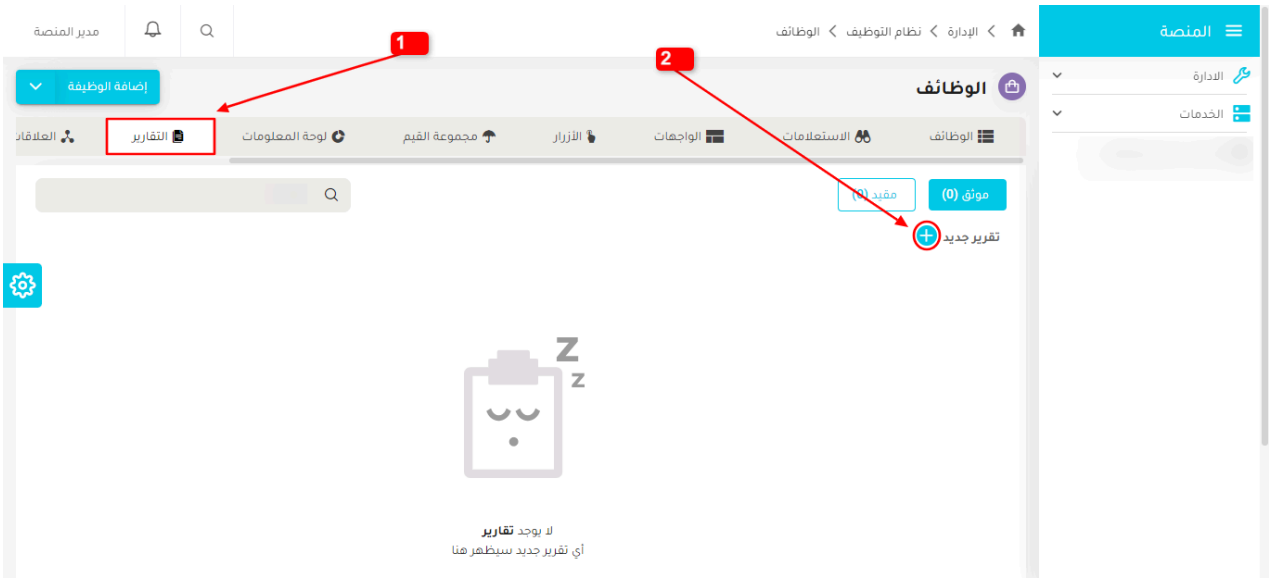
Creating a Report

A report turns a query into a formatted document, and can be sent on a schedule. Here you create a yearly report of the vacancies and email it to HR as a PDF.

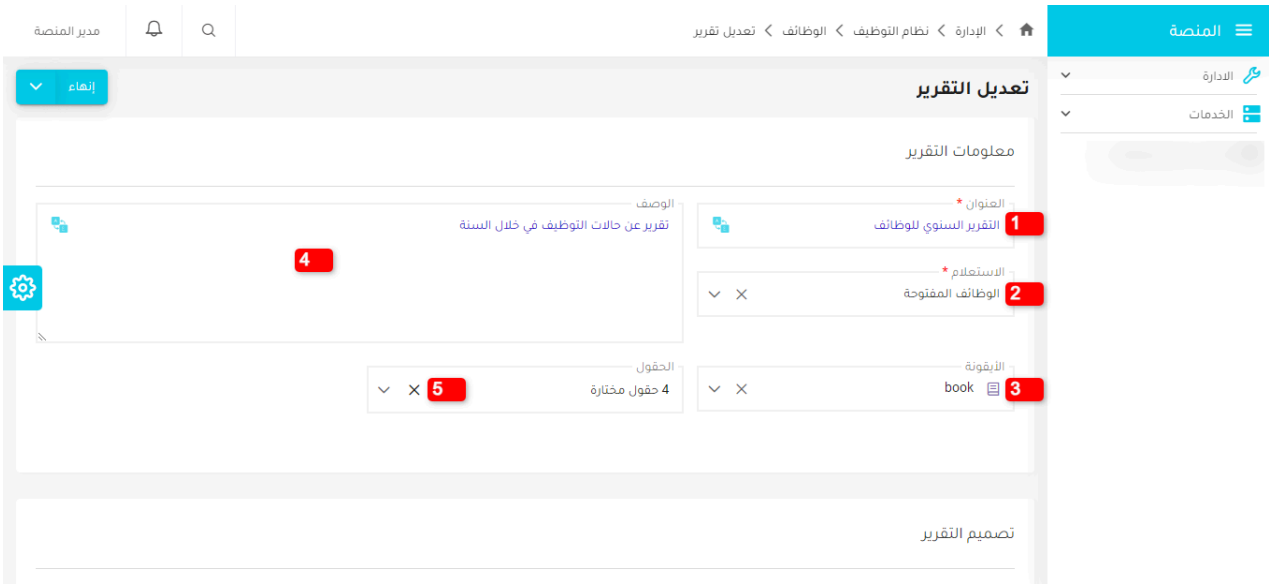
Annual jobs report

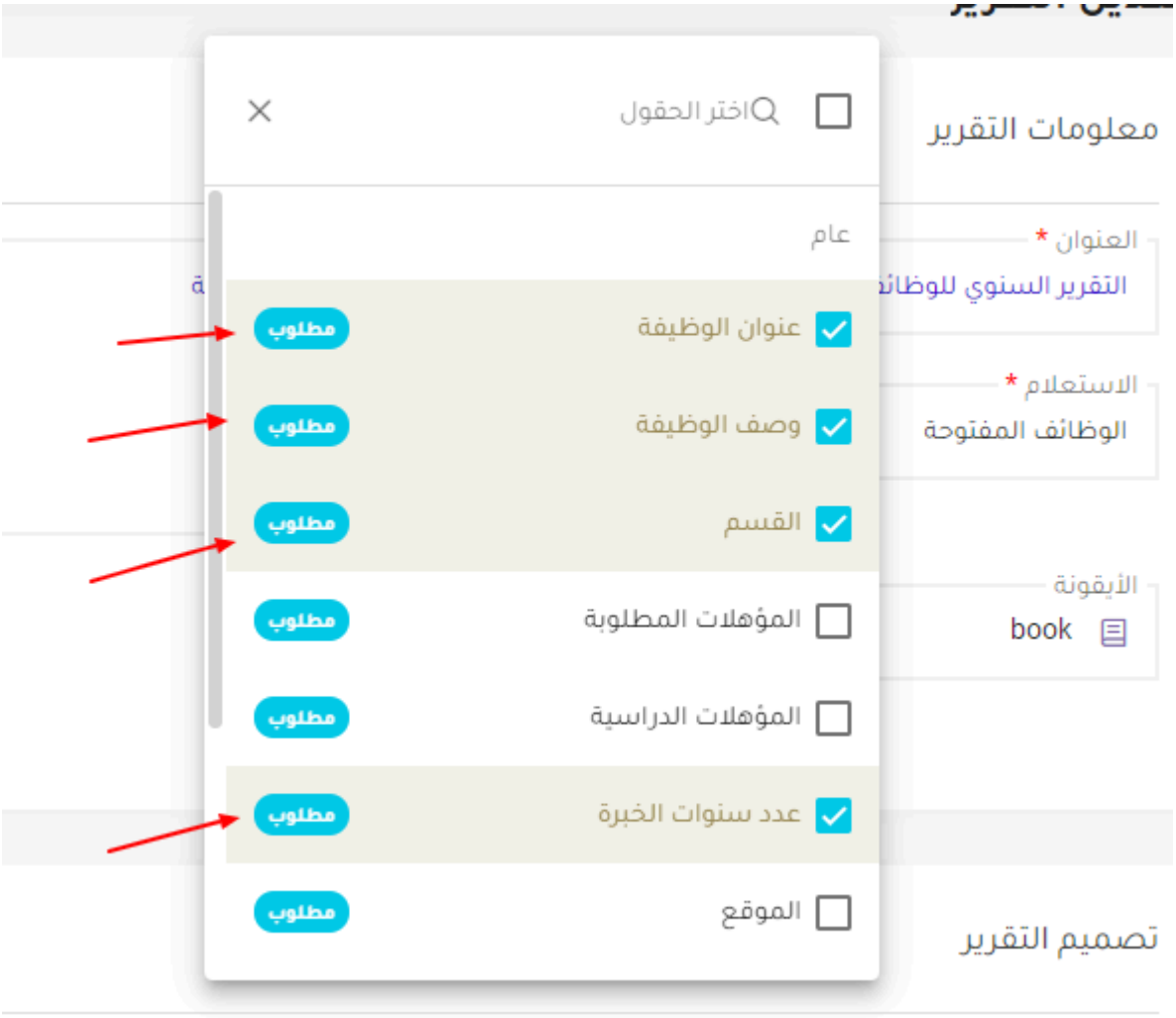
For more about reports, see Reports.

1. Create the report
    - Open the *Jobs* entity and its **Reports** tab.
    - Select **+**.



    - The report page opens. In **Report information**, fill in:
        - **Title**: *Annual jobs report*.
        - **Query**: *Open jobs*.
        - **Icon**: *book*.
        - **Description**: *A report of the year's hiring.*
        - **Fields**: *Job title*, *Description*, *Department* and *Years of experience required*.



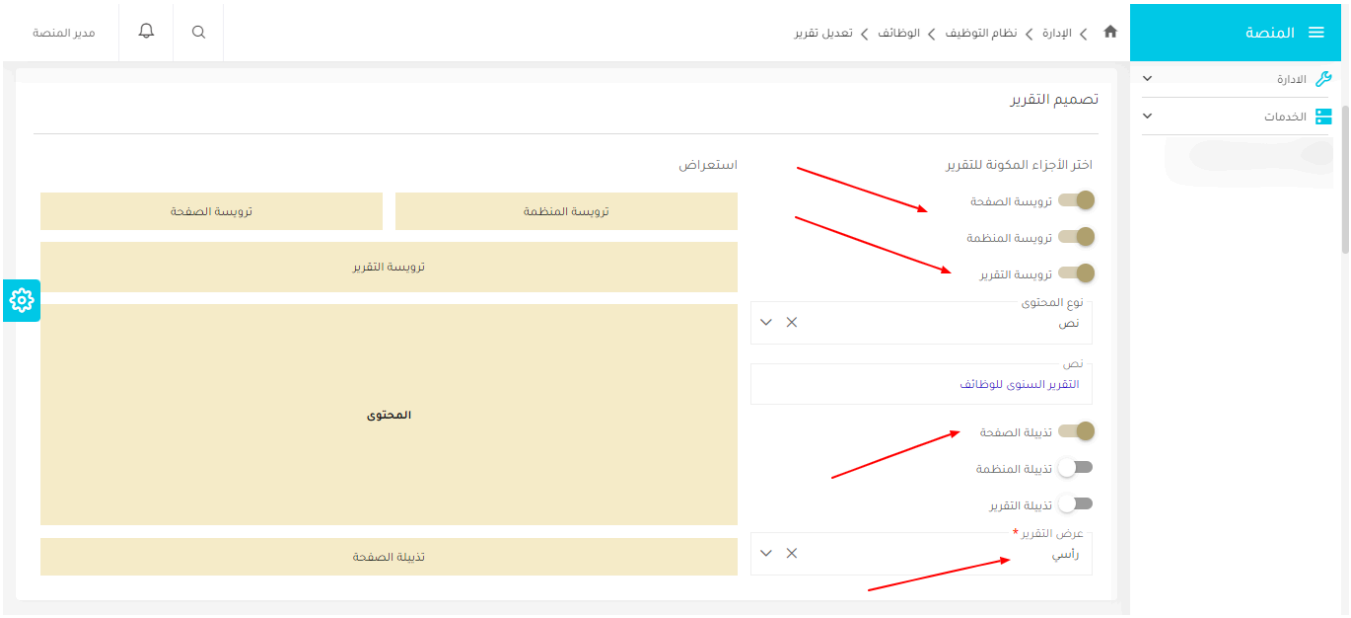


- Under **Report Design**, turn on **Page header**.
- Turn on **Organization Header**.
- Turn on **Report Header**. Two fields appear:
  - **Content Type**: **Text**.

        - The text: *Annual jobs report*.

    - Turn on **Page Footer**.

    - **Layout**: **Portrait**.



- Under the sending settings, set **Interval type** to **Repeat**.

- **Cron Expression**: `0 0 1 1 *`, which sends the report at midnight on 1 January every year.



- **Channel Type**: email.

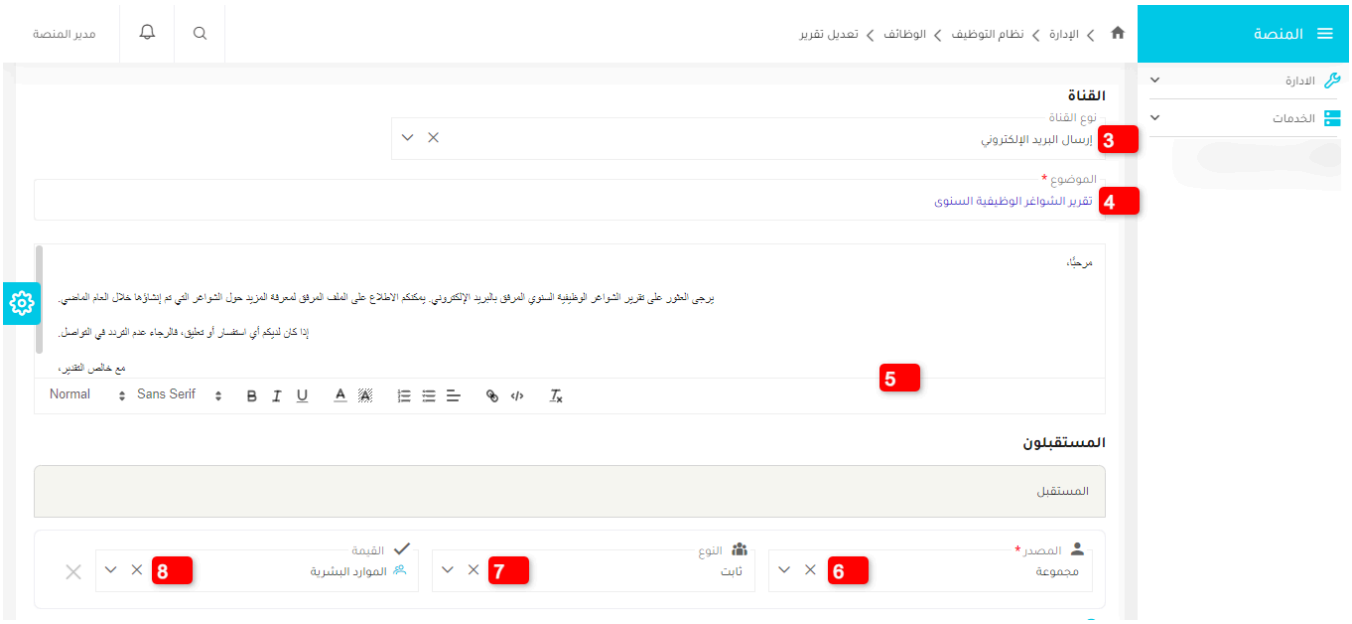
- **Subject**: *Annual vacancies report*.

- Message body, for example:

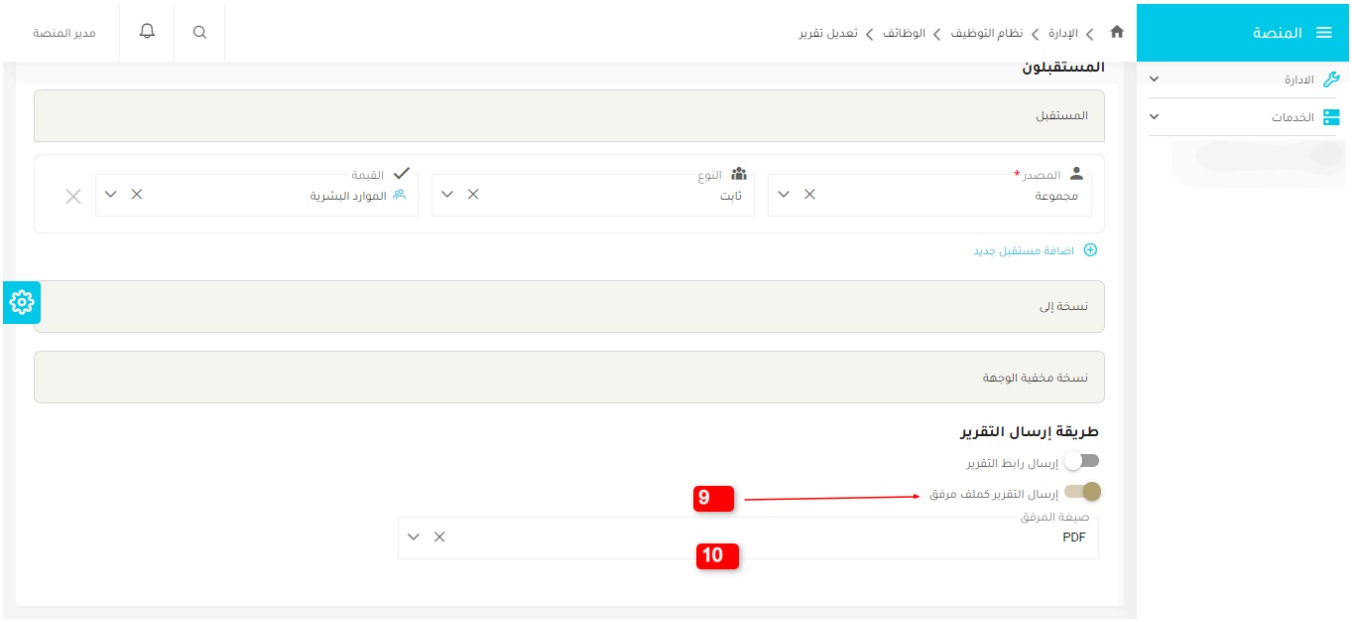
Hello, Please find attached the annual vacancies report, with the vacancies created over the past year. If you have any questions or comments, please get in touch.

Kind regards,

- Recipients: **Source Group**, **Type Fixed**, **Value** *HR*.



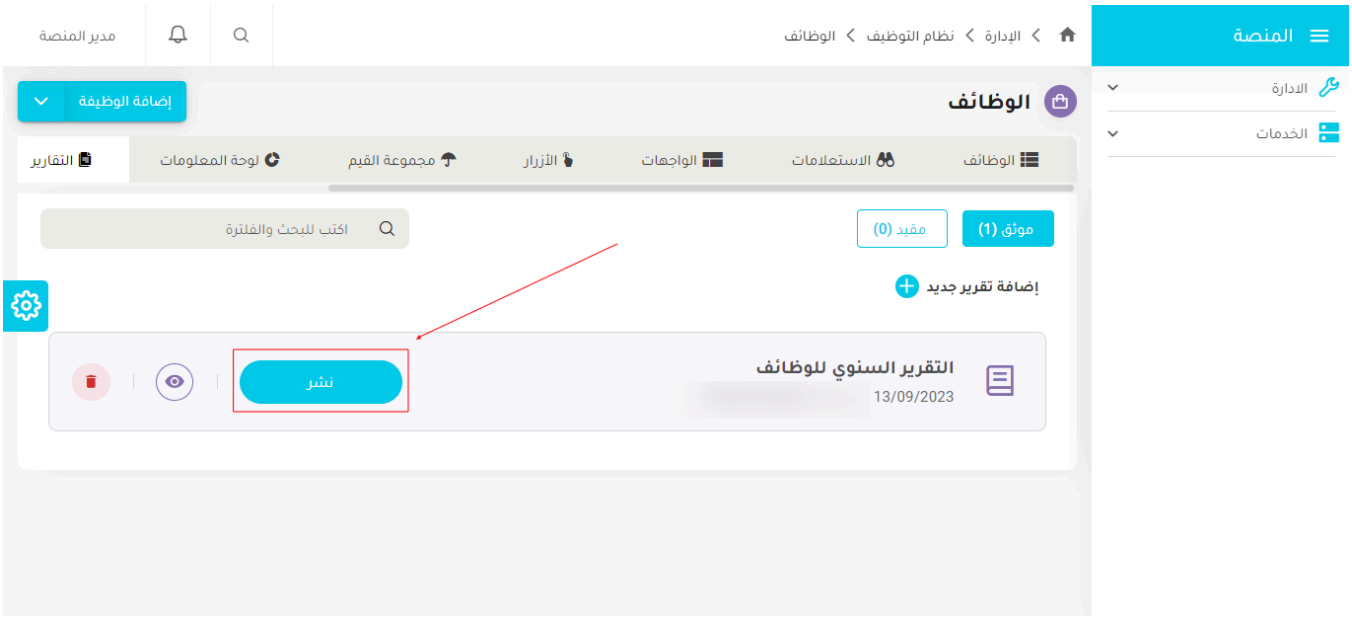
- Turn on **Send report attachment file**, and in **Attachment type** choose PDF.



- Select **Done** to save the report.



- Select **Publish** on the report so it is sent on its schedule.



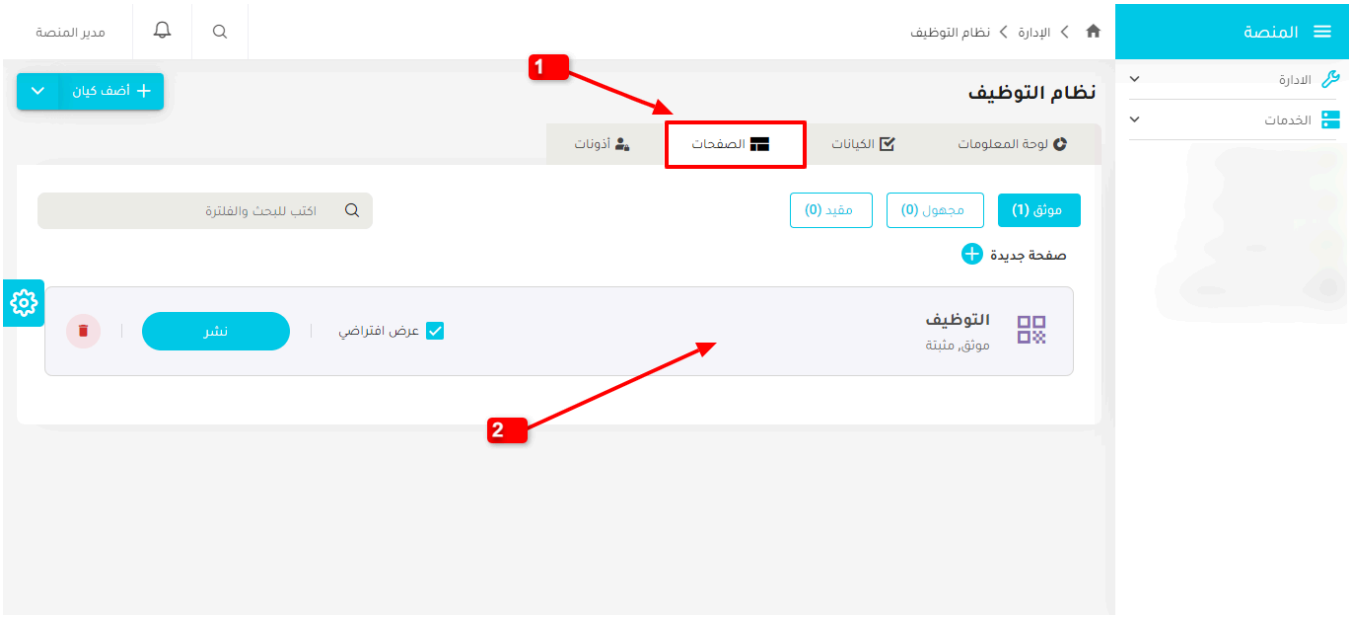
Technical details

Cron Expression takes the standard five fields: minute, hour, day of the month, month, day of the week. `0 0 1 1 *` means minute 0, hour 0, day 1, month 1 (January), any weekday. A `*` in the minute field would repeat the send every minute of that hour.

Building the Home Page

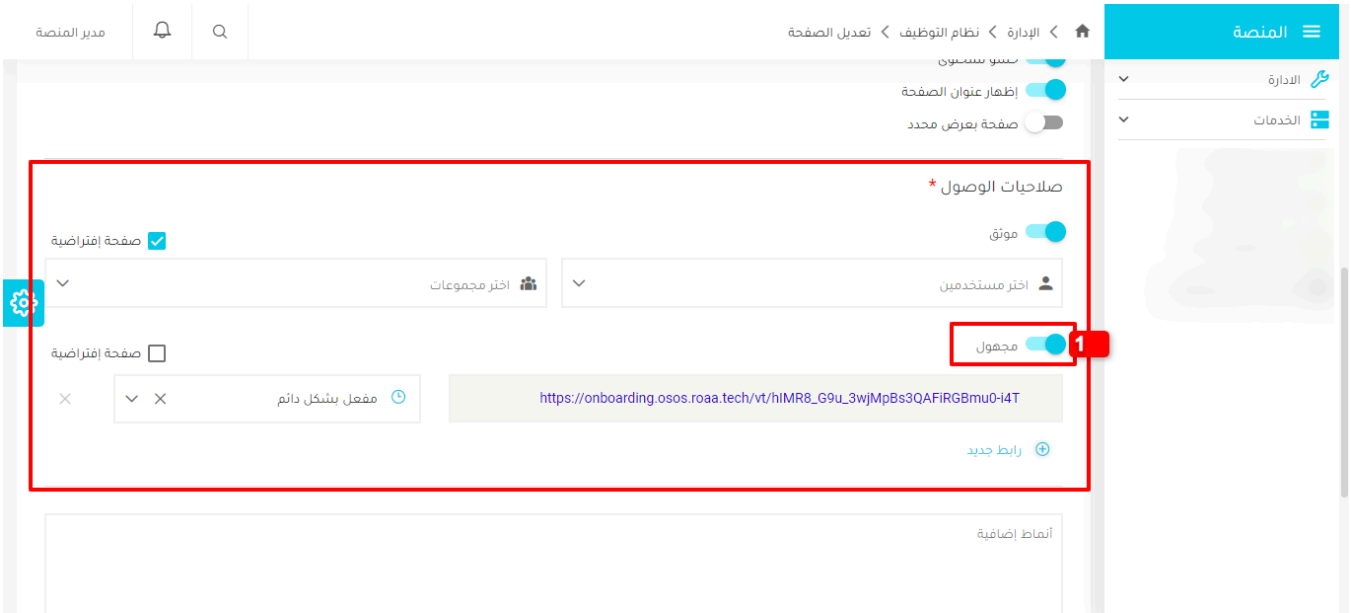
The home page is what people see when they open the recruitment system. Visitors see the open jobs; HR and IT also see the links and figures they need. Each part of the page can be shown to a different audience.

Step 1: Open the *Recruitment System* module. **Step 2:** Open its **Pages** tab.

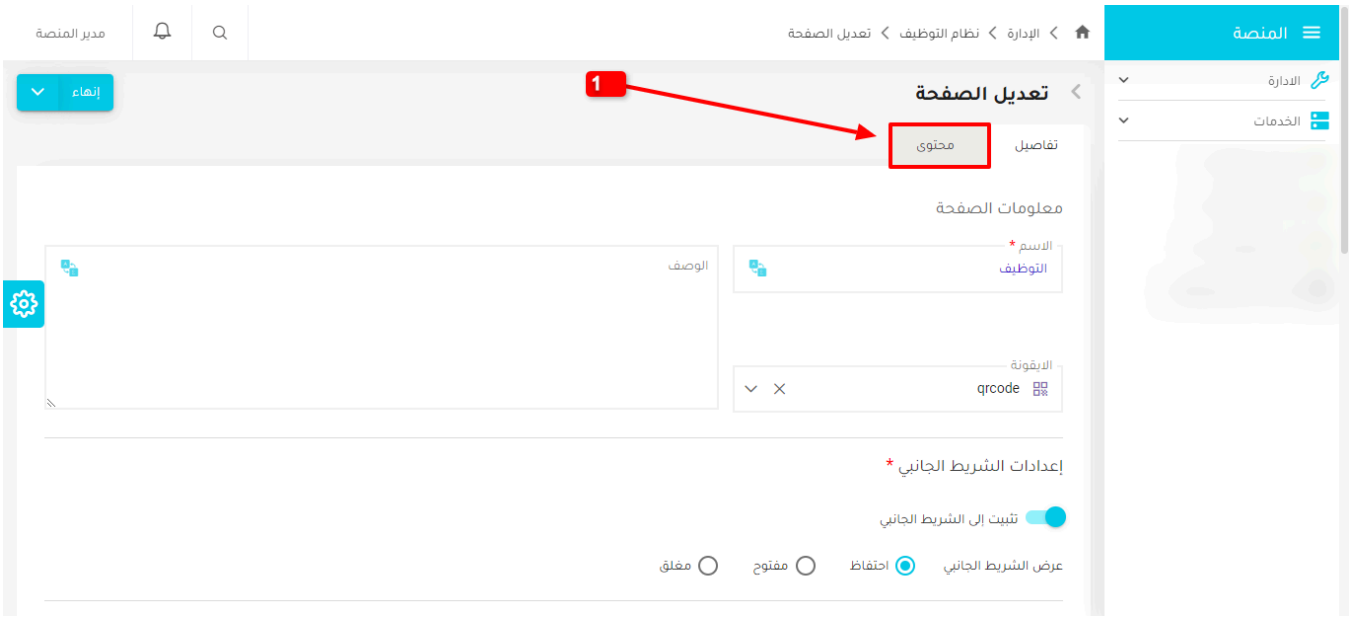


Step 3: Select the *Recruitment System* page that was created with the module. The page details open.

- In the page details, scroll to the permissions and turn on **Anonymous**, so visitors can open the page without signing in.



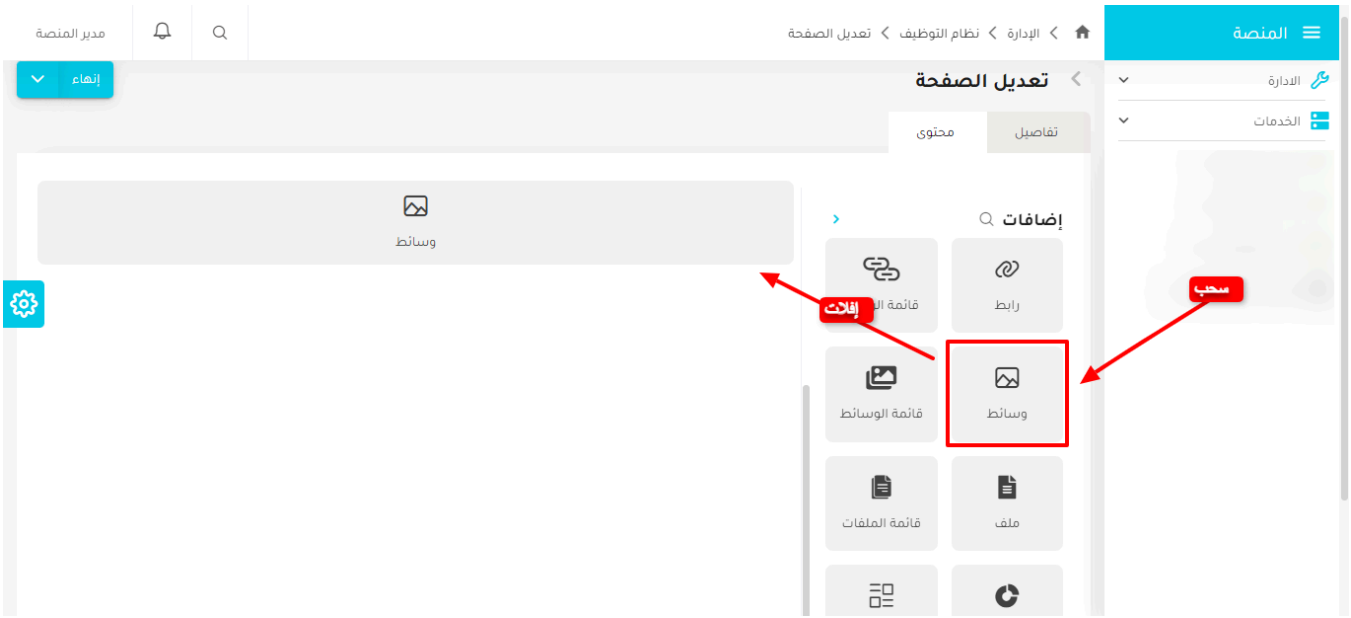
Step 4: Open the **Content** tab.



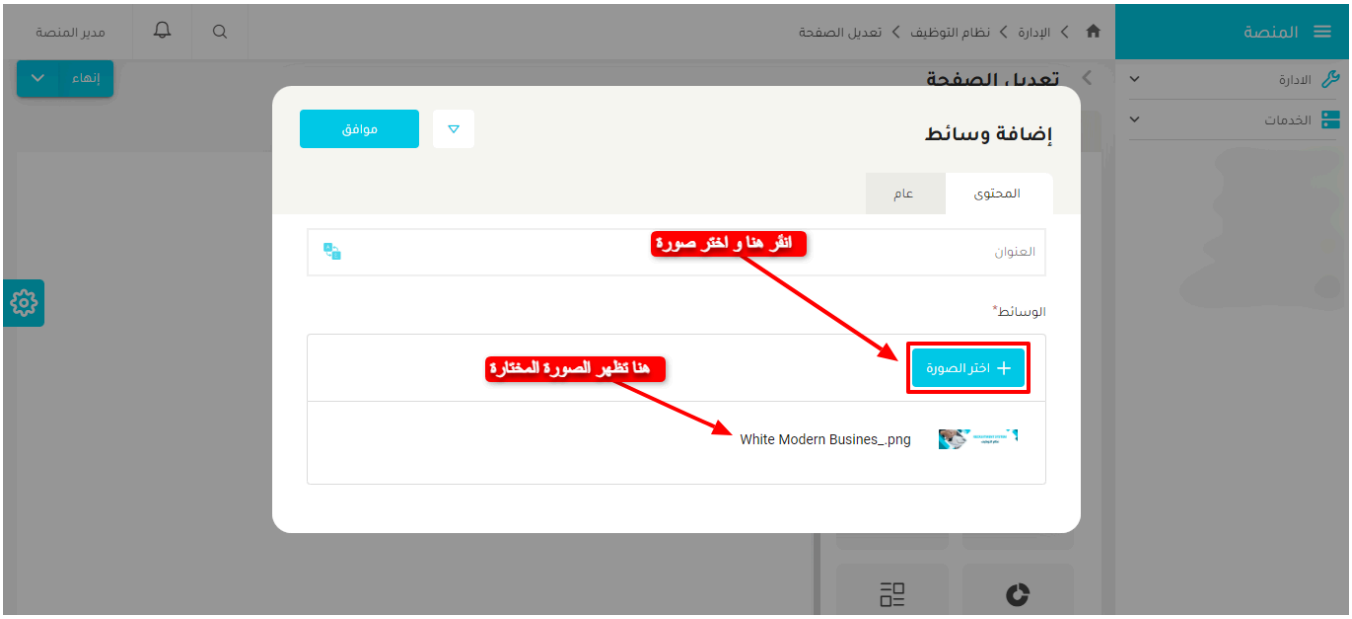
Build the page with the **Widgets** panel at the side of the page:

1. Add the logo:

- From the widgets, drag **Media** onto the page and drop it.



- In the dialog that opens, choose your image.

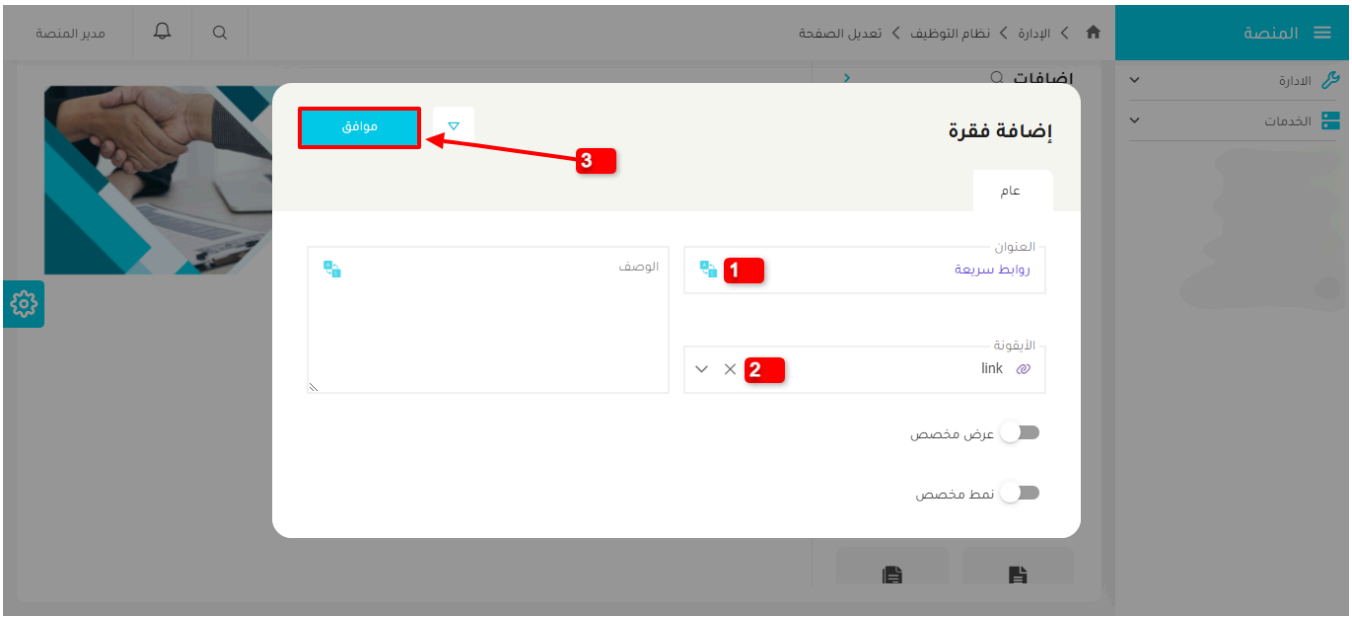


- Set the image's width to 12, the full width of the page.



2. Quick links:

- Add a **Section** widget to the page, with the details shown in the screenshot.

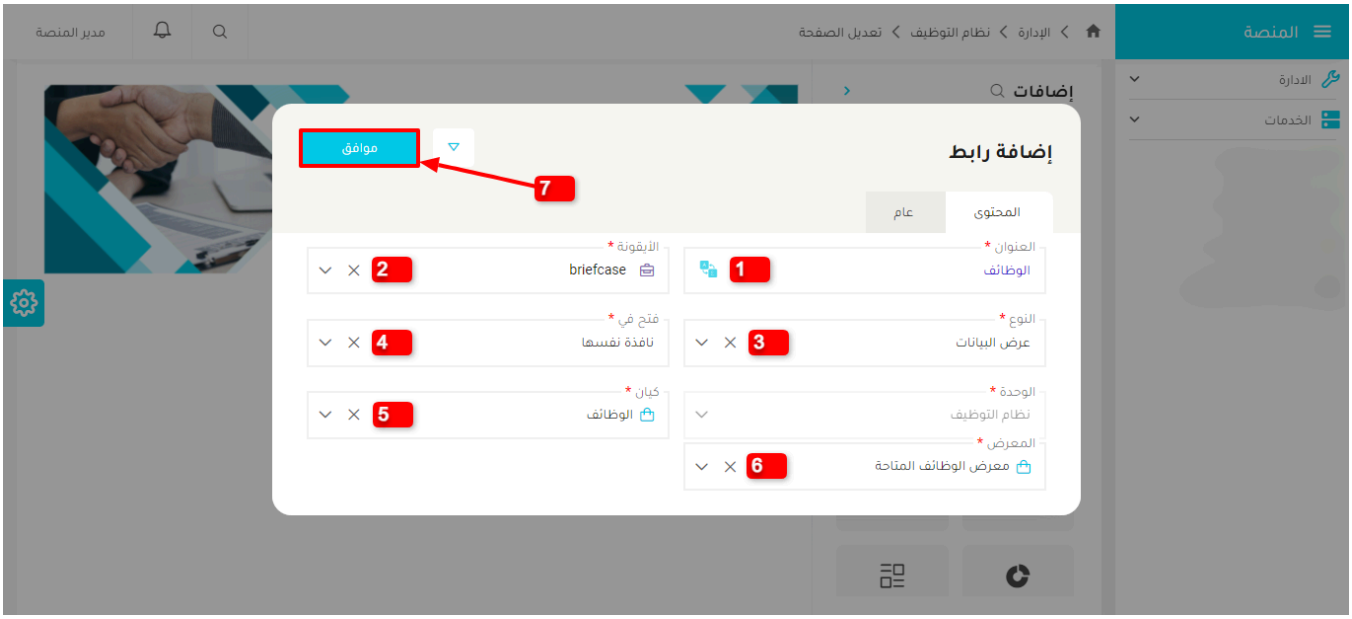


- Add a **Link** widget inside the section.



2.1. Jobs:

- The link dialog opens. Fill in: **Title** *Jobs*, an **Icon**, **Type Data View**, **Open in Same Window**, **Module** *Recruitment System*, **Entity** *Jobs*, and **View** *Available jobs*. Confirm with **OK**.



- Set the section's height to 2.

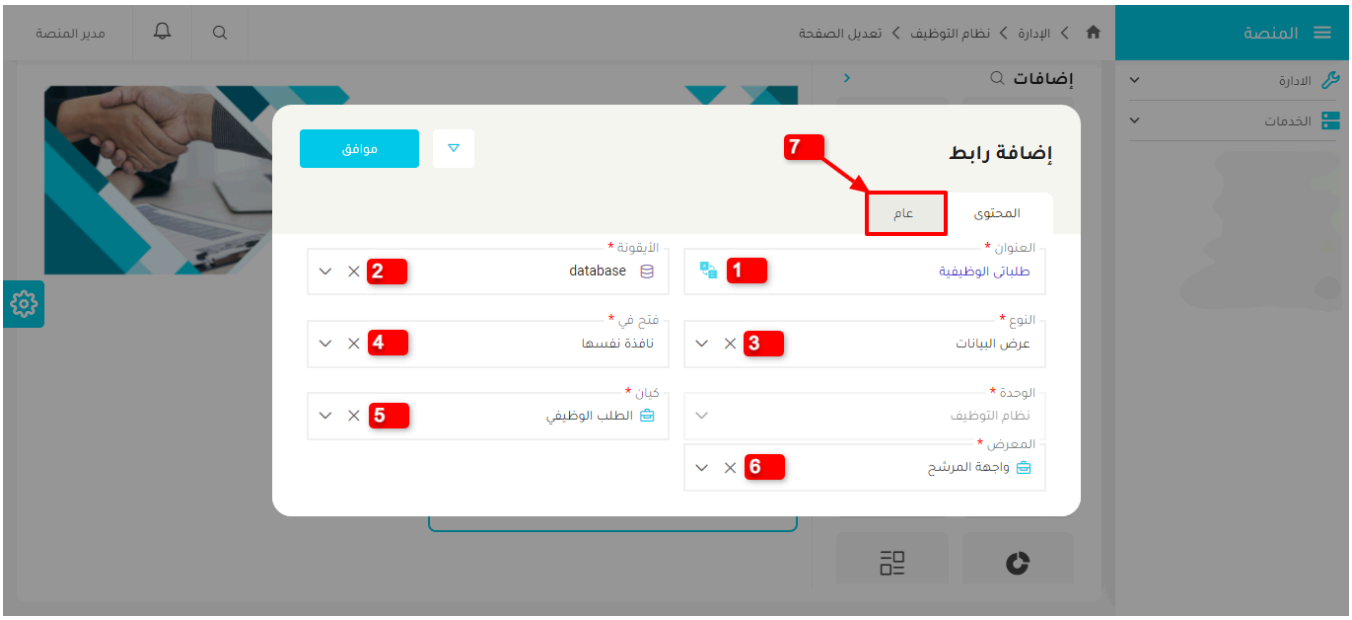


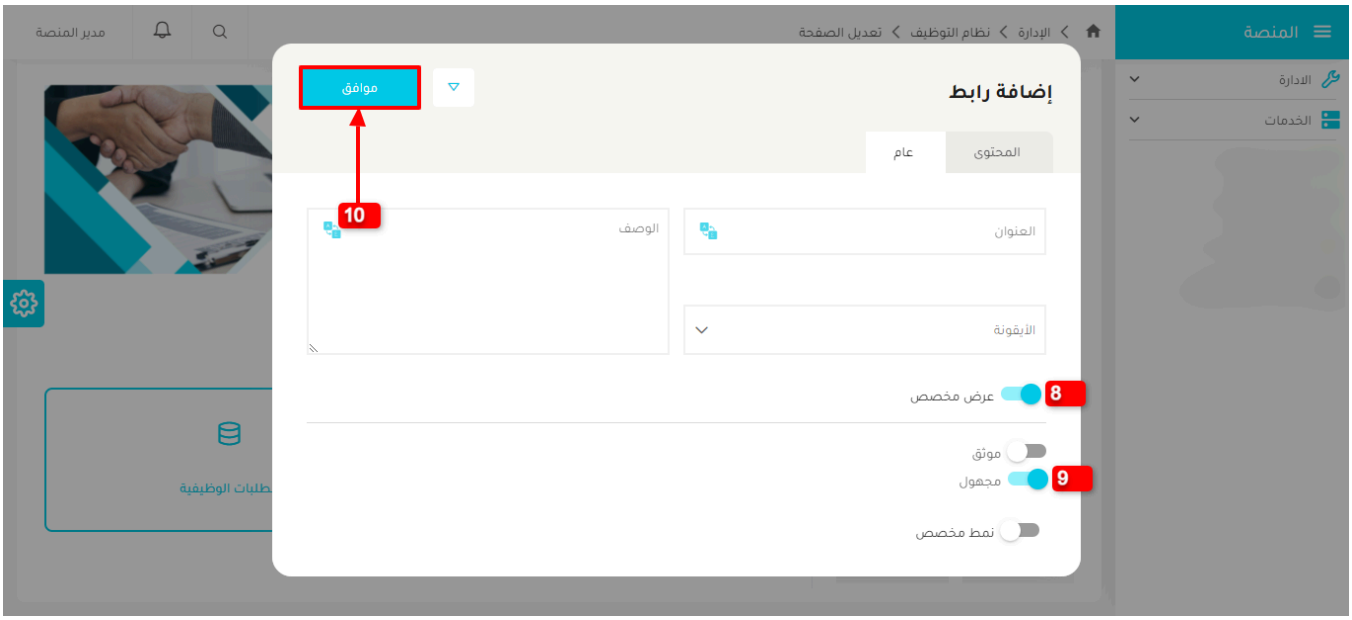
- Set the link's width to 6.



2.2. My job applications:

- Add a link to the *Job Application* service. On the dialog's **General** tab, turn on **Custom Visibility** and **Anonymous**, so only visitors see it.
- The screenshot points this link at a *Candidate view* that this tutorial does not create. Visitors who are not signed in cannot be told apart, so never open a view of job applications to **Anonymous**: point the link at a page that is safe for everyone, or leave it out.

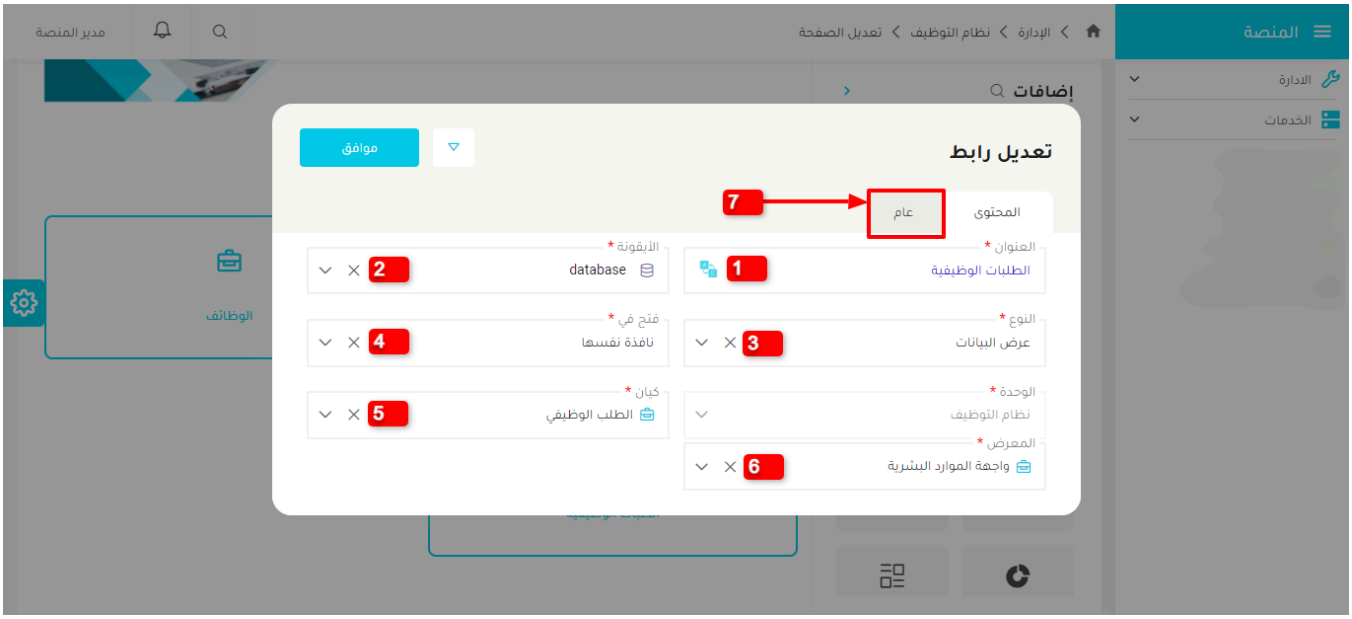




- Set the link's width to 6.

2.3. Job applications:

- Add a link to the *HR view* of the *Job Application* service. On the **General** tab, turn on **Custom Visibility** and show it only to the *HR* group.

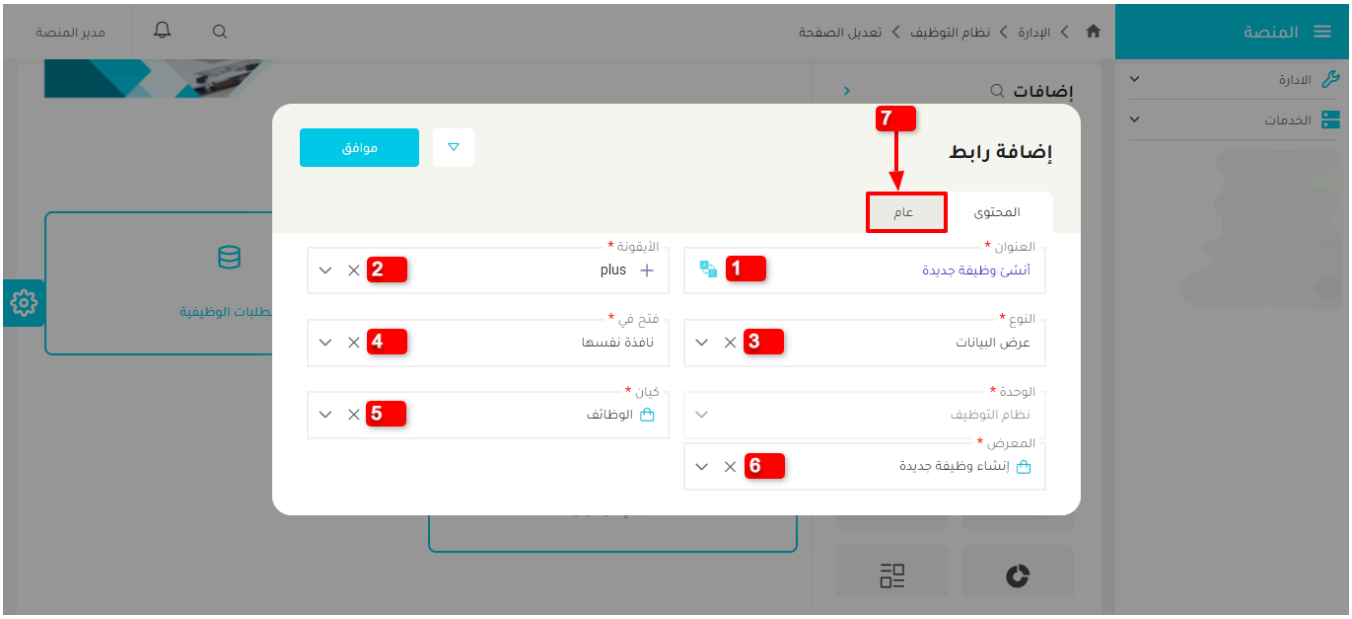




- Set the link's width to 6.

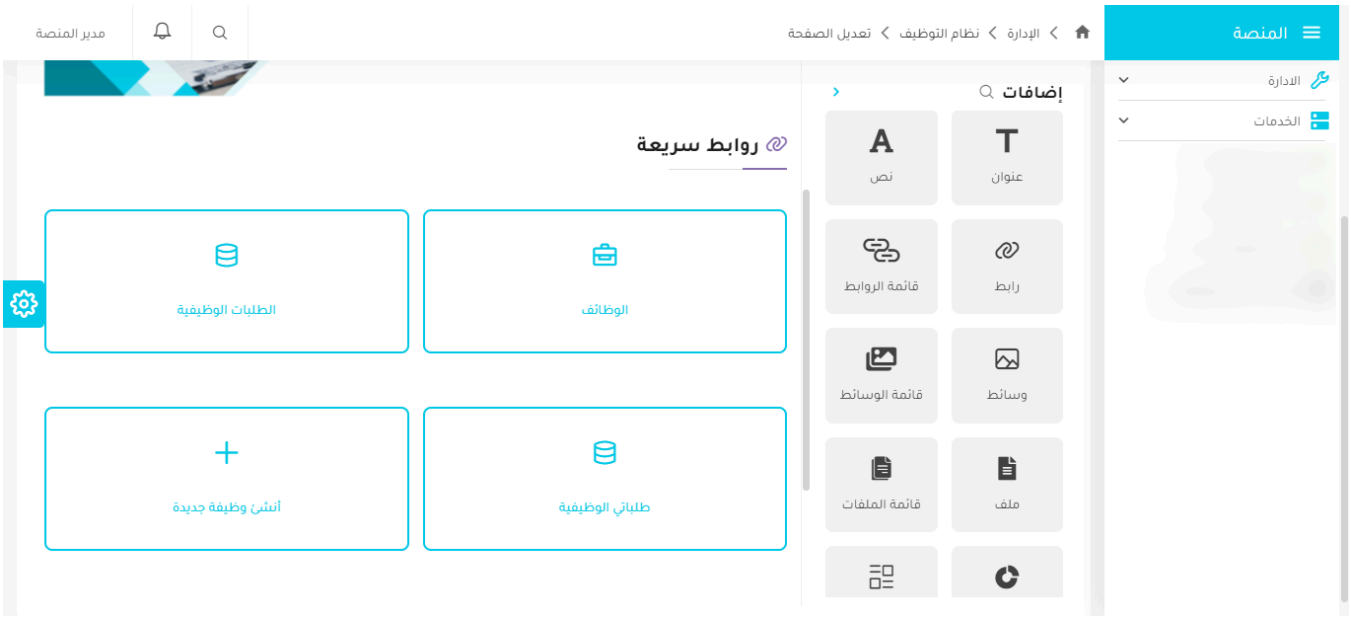
2.4. Create a new job:

- Add a link titled *Create a new job* to the *All jobs* view of the *Jobs* entity. On the **General** tab, turn on **Custom Visibility** and show it only to the *IT* group.





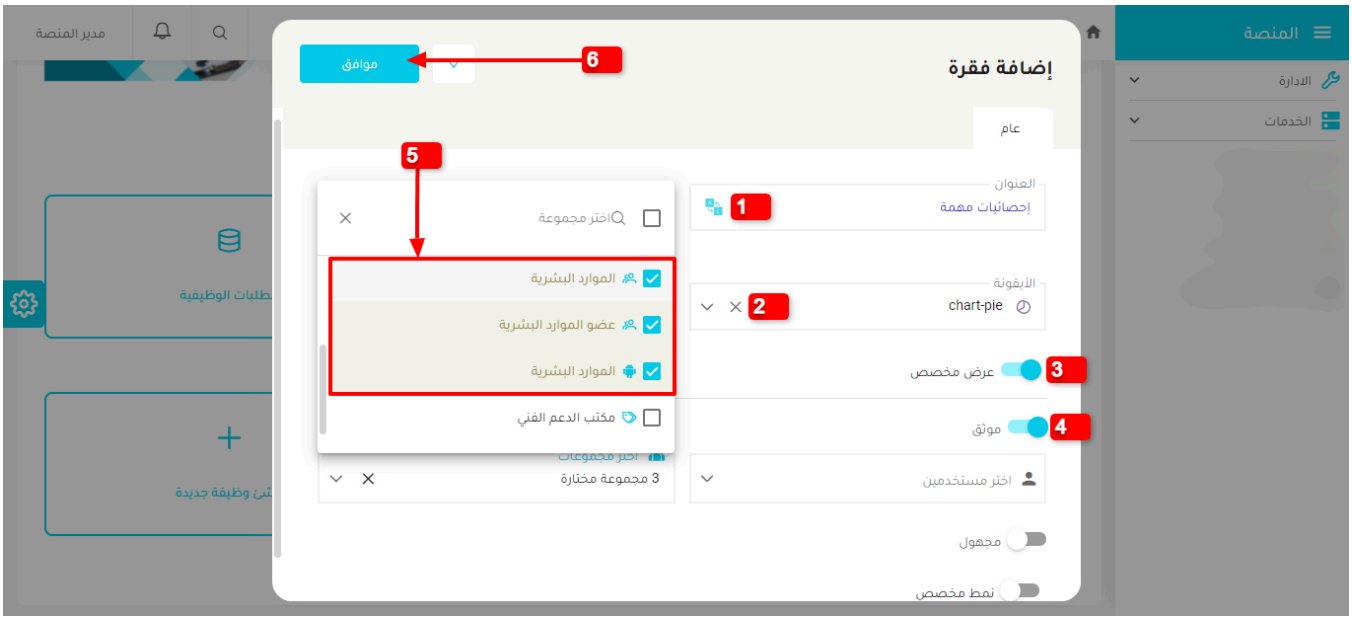
- Set the link's width to 6.



3. Key figures section

Show HR the most important figures.

- Add a **Section** widget titled *Key figures*, and make it visible only to the *HR* group, as shown in the screenshot.



- Add a **Summary** widget inside the section.



- In the summary dialog, choose one of the summaries you created on the dashboards, with the details shown in the screenshot.



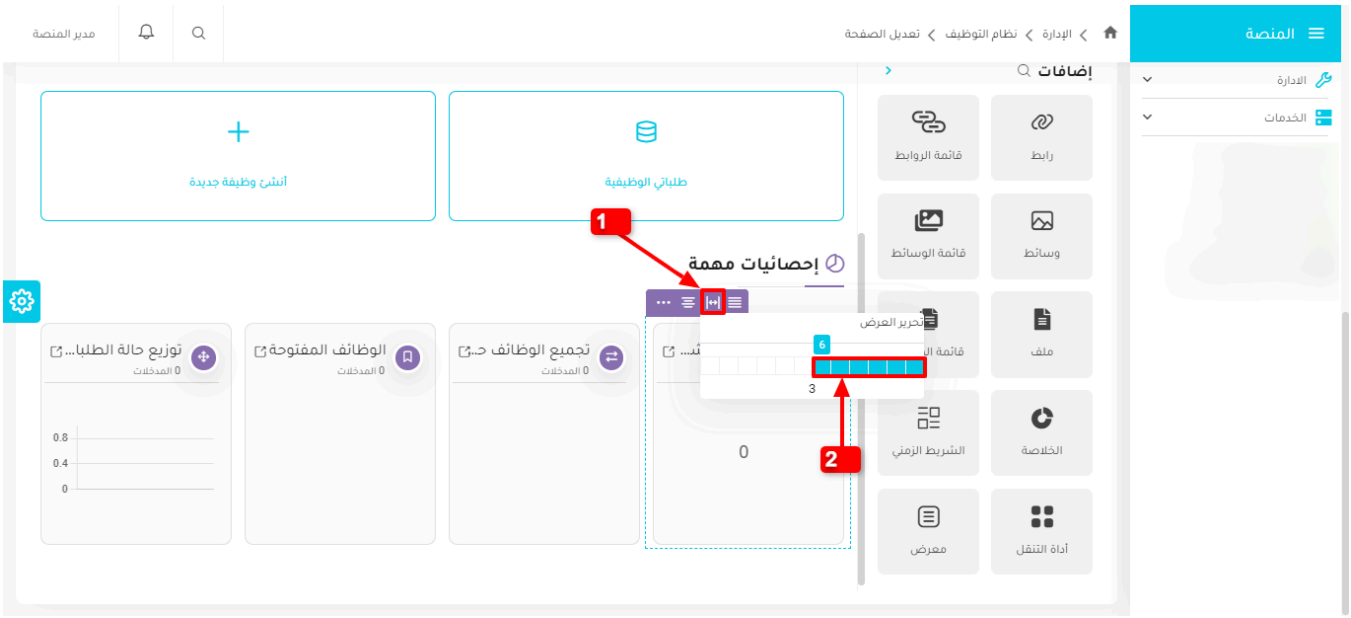
- Repeat for the other summaries.

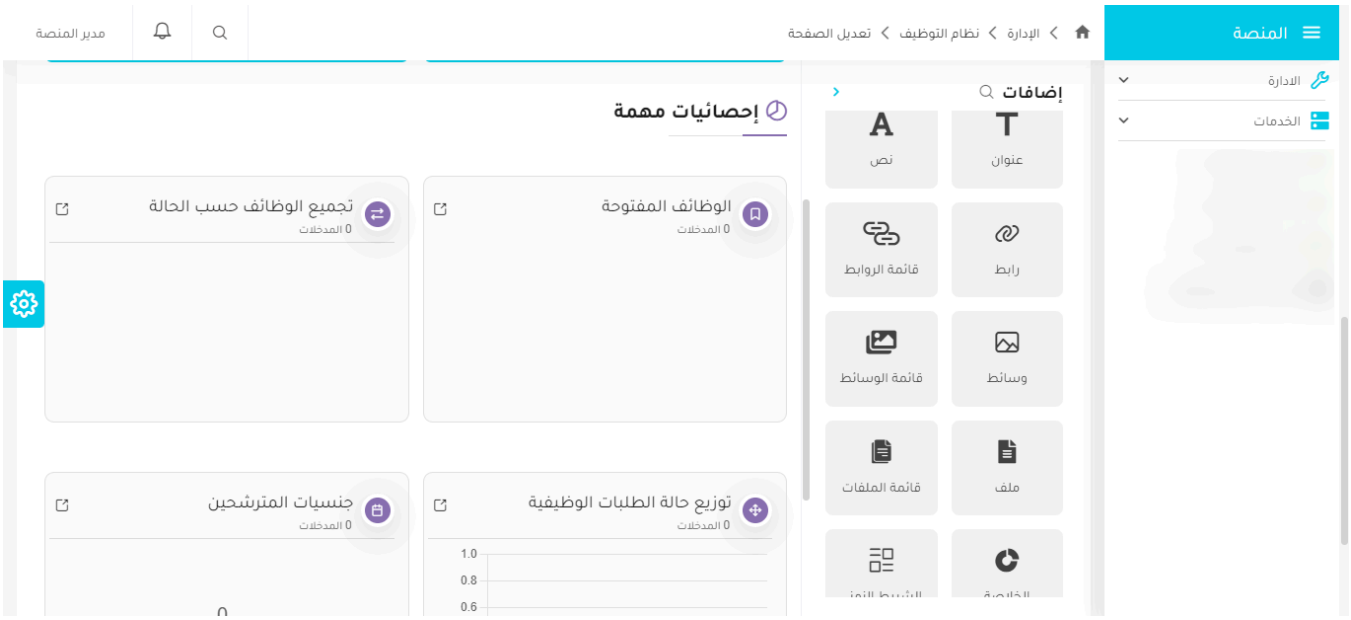






- Set each summary's width to 6.

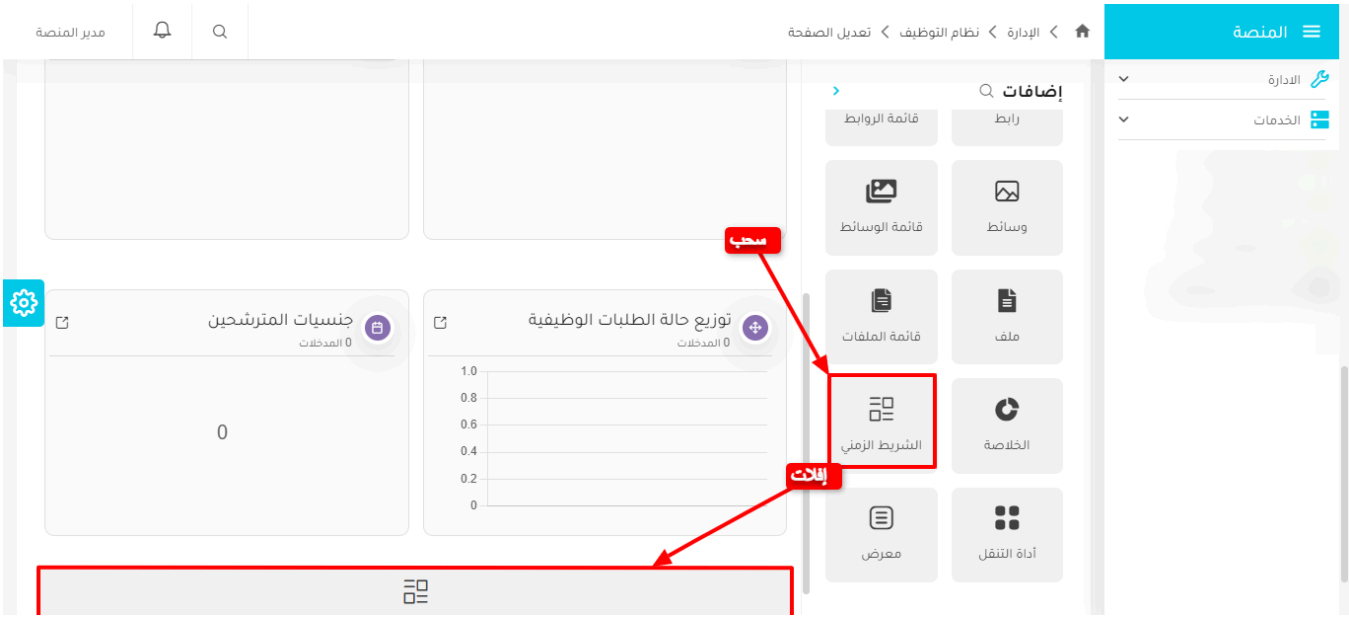




4. The timeline:

The timeline brings together the recruitment system's timelines. Visitors who are not signed in see the company's job posts; signed-in employees also see the welcome posts of the Receptions entity.

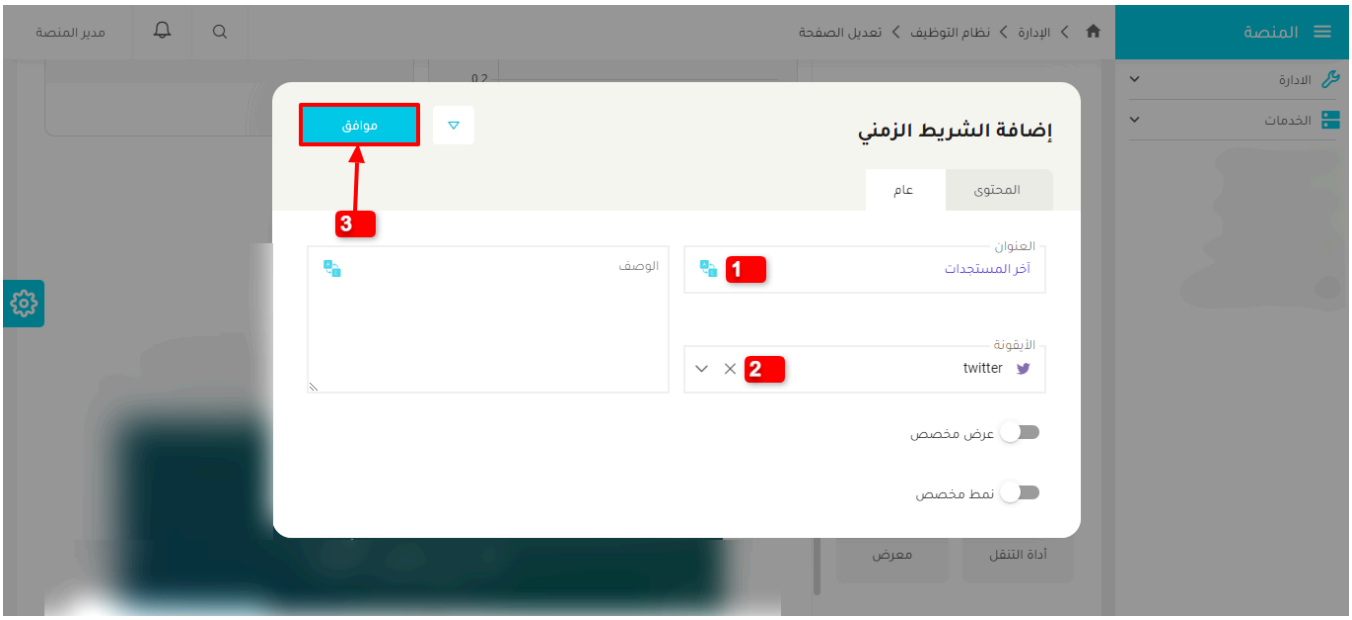
- Add a **TimeLine** widget to the page.

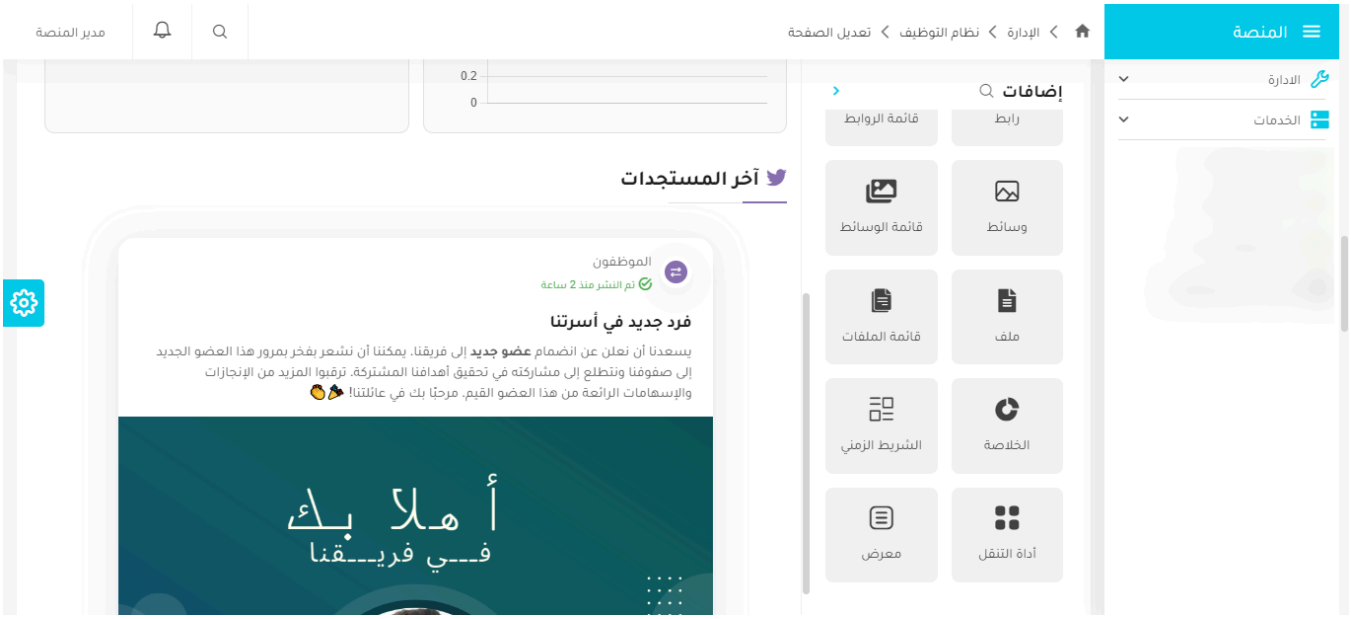


- The timeline dialog opens. Fill it in as shown in the screenshot.



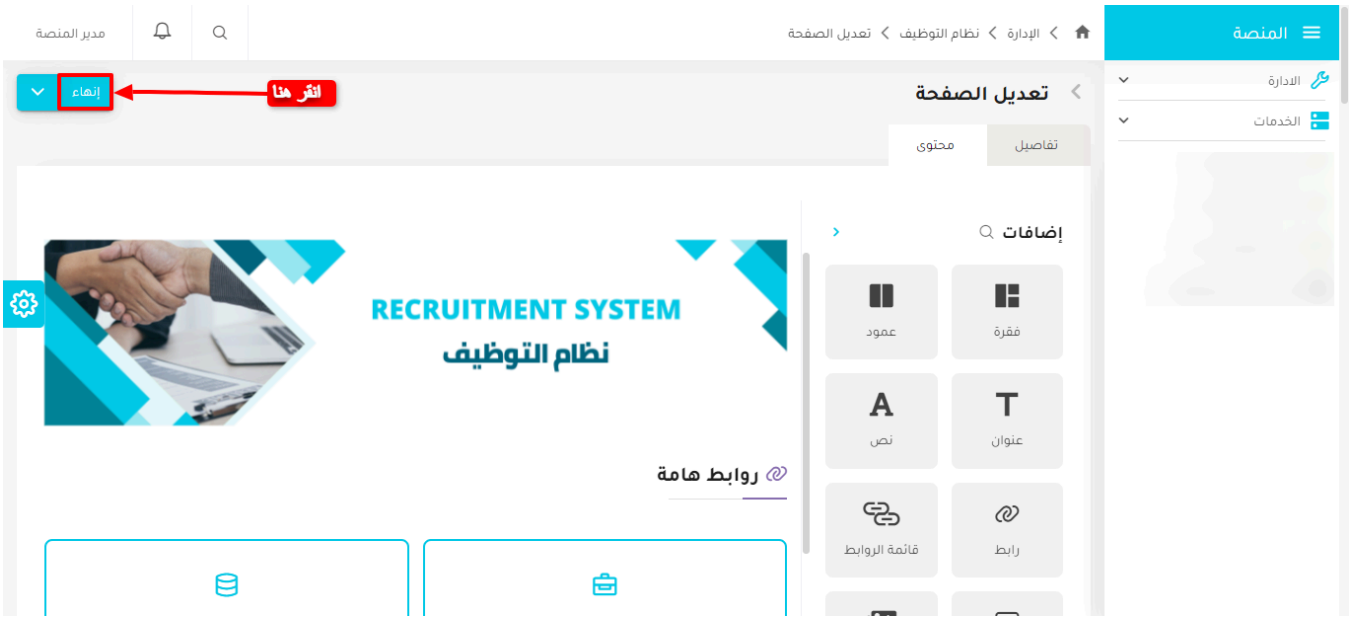
- On the **General** tab, enter the timeline's title and icon, as shown below.



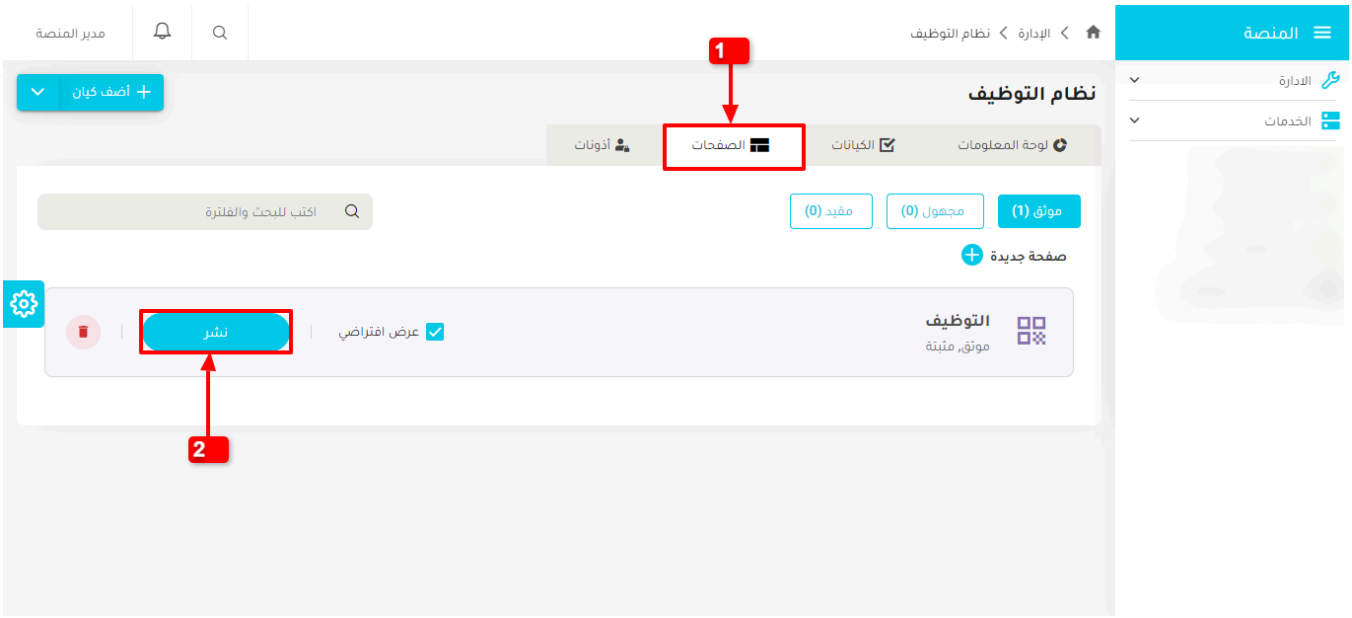


5. Save and publish the page:

- When the page is ready, select **Save**. The list of pages opens.



- Select **Publish** on the *Recruitment System* page.



Go deeper

Every widget and its options: Pages & widgets

Testing the Recruitment System

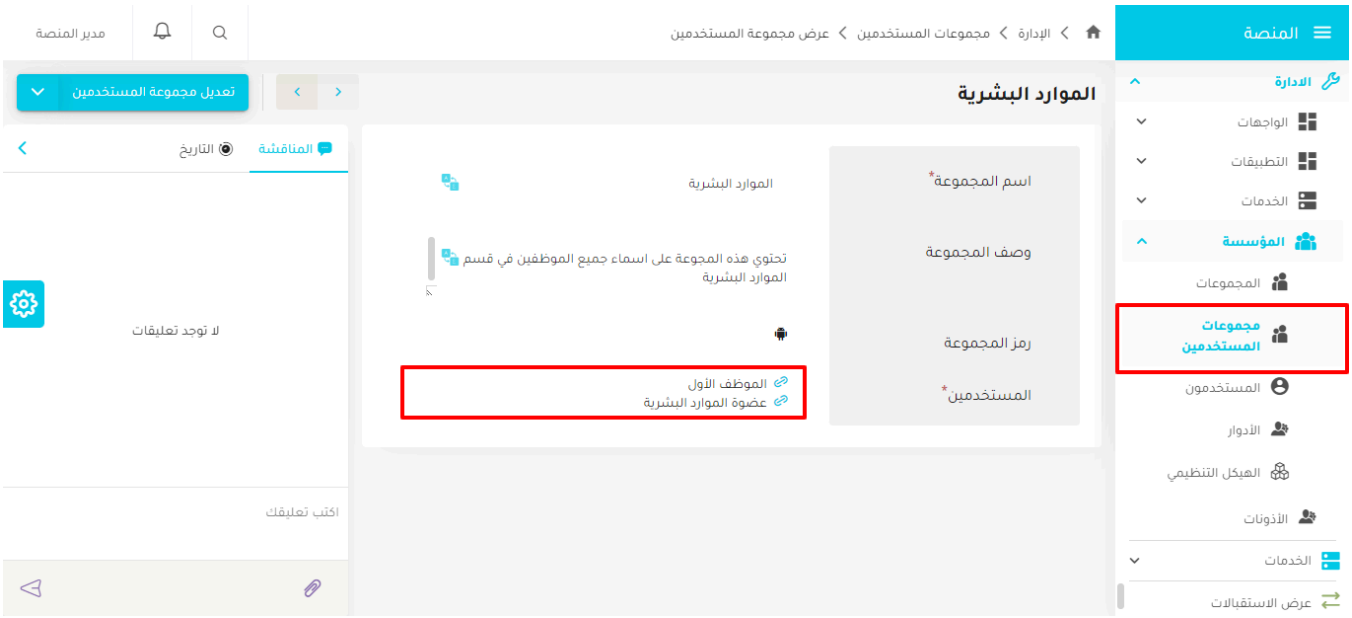
This test checks that every part of the recruitment system works together. You play each role in turn, in separate browser windows: an IT member who drafts a vacancy, an HR member who reviews and processes it, a visitor who applies, and the platform administrator who creates the new employee's account.

Create a new vacancy

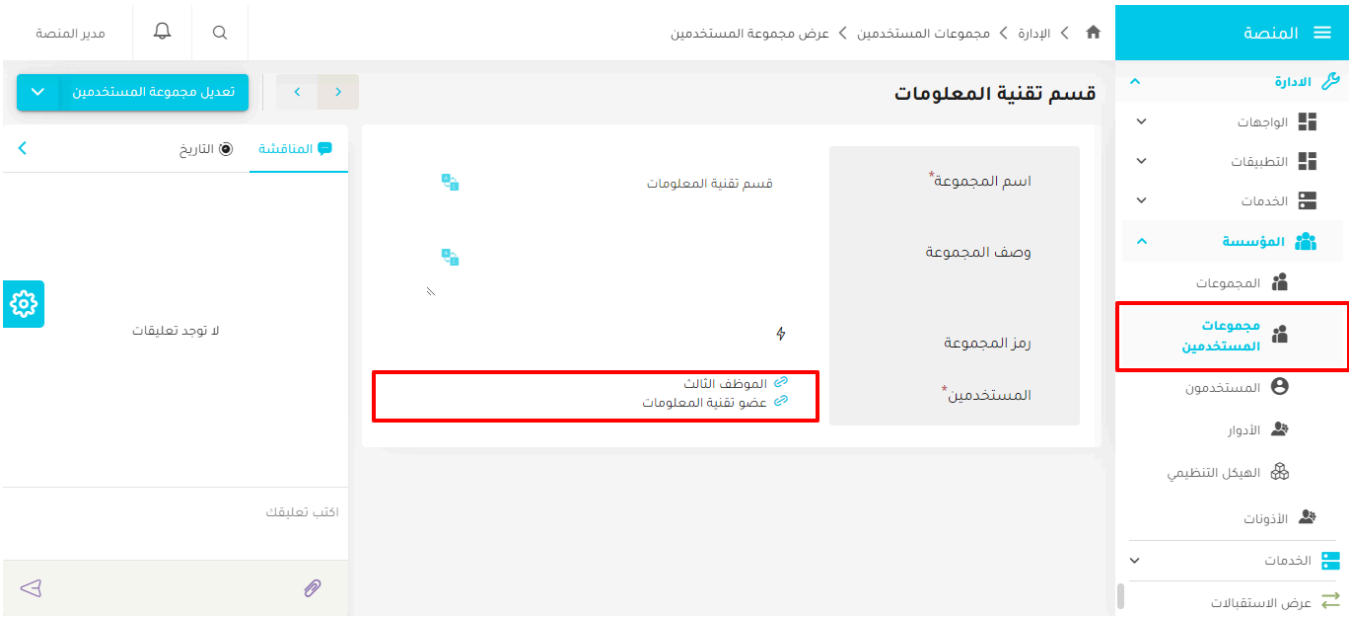
- This part involves two roles:
  - an HR member
  - an IT member

1. Create the test users:

- Create an *HR member* user and add it to the *HR* group.

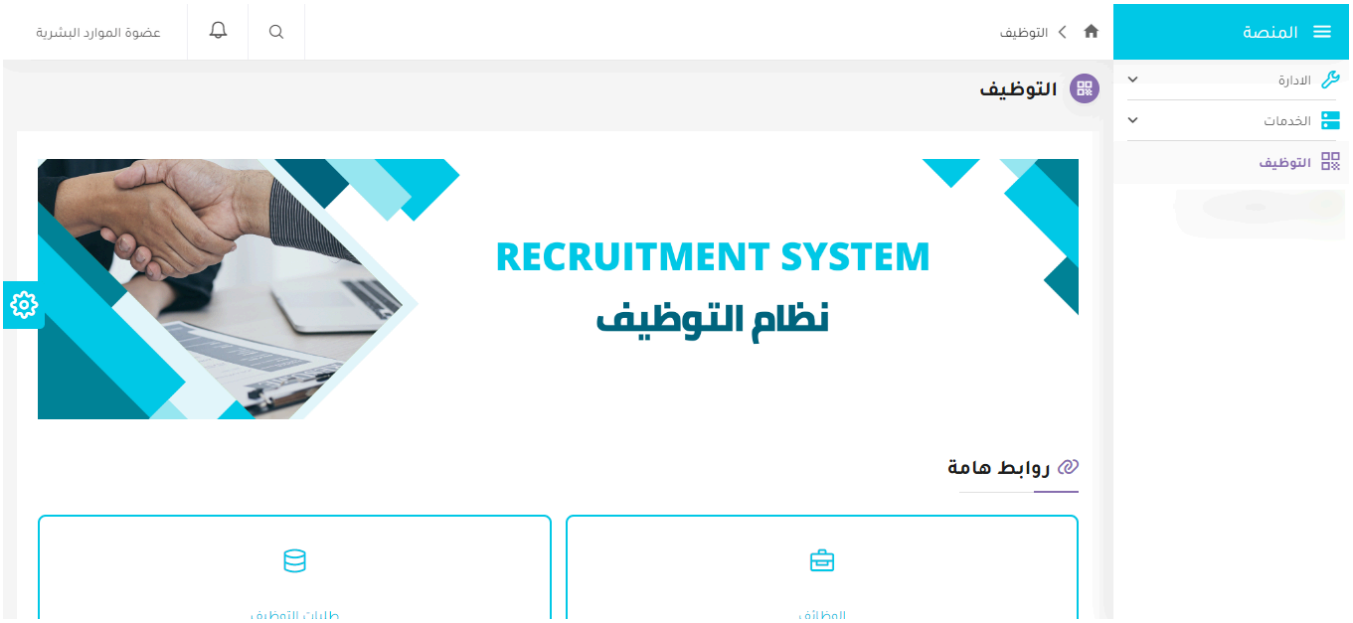


- Create an *IT member* user and add it to the *IT* group.

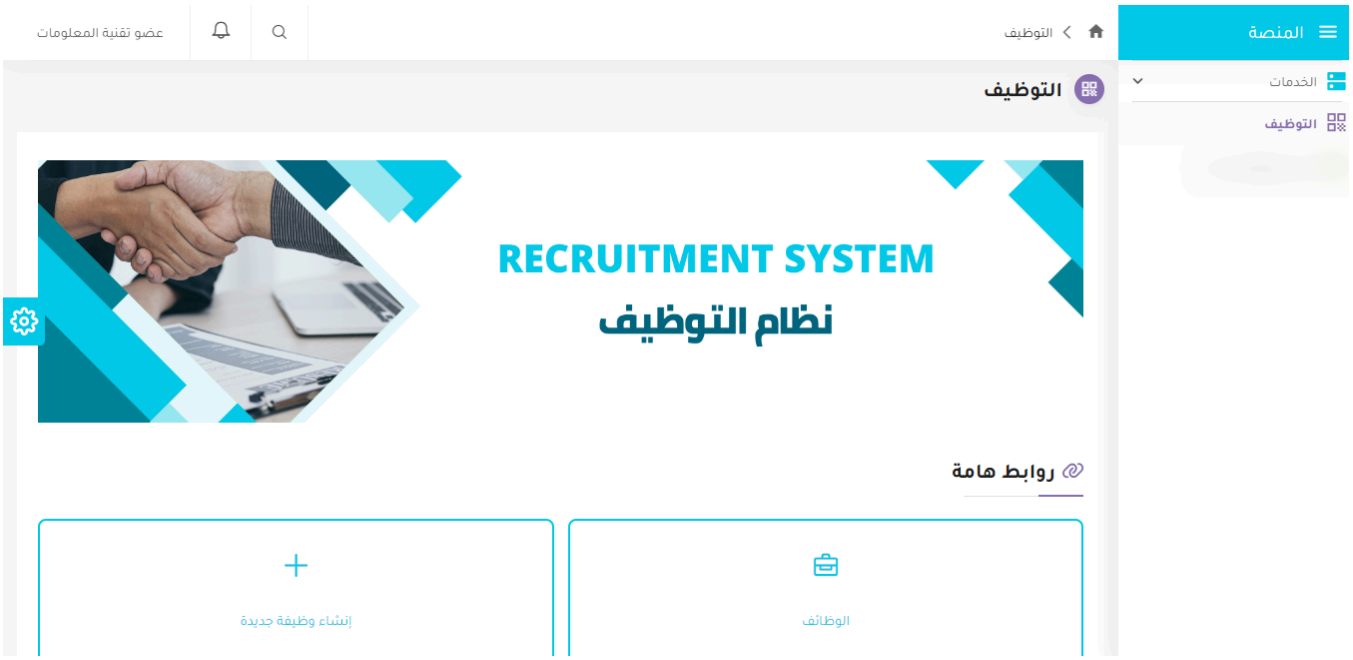


2. Sign in:

- Open a browser and sign in as the HR member.



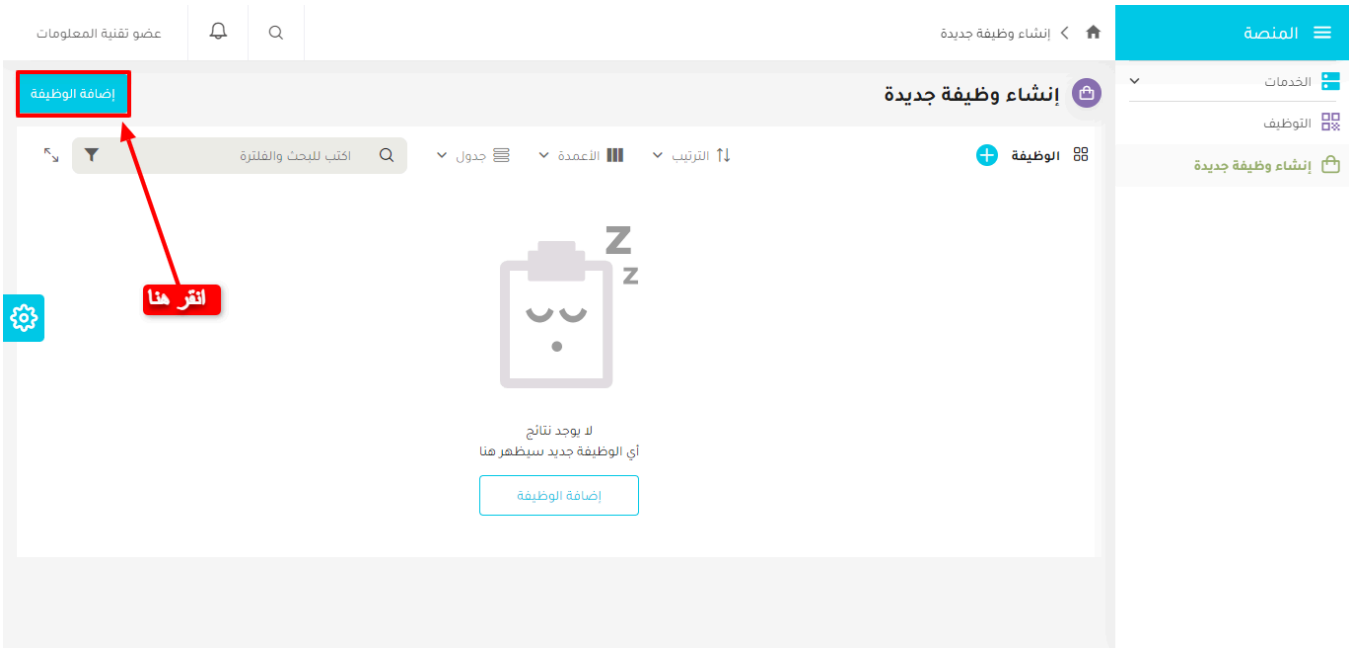
- Open a second browser and sign in as the IT member.



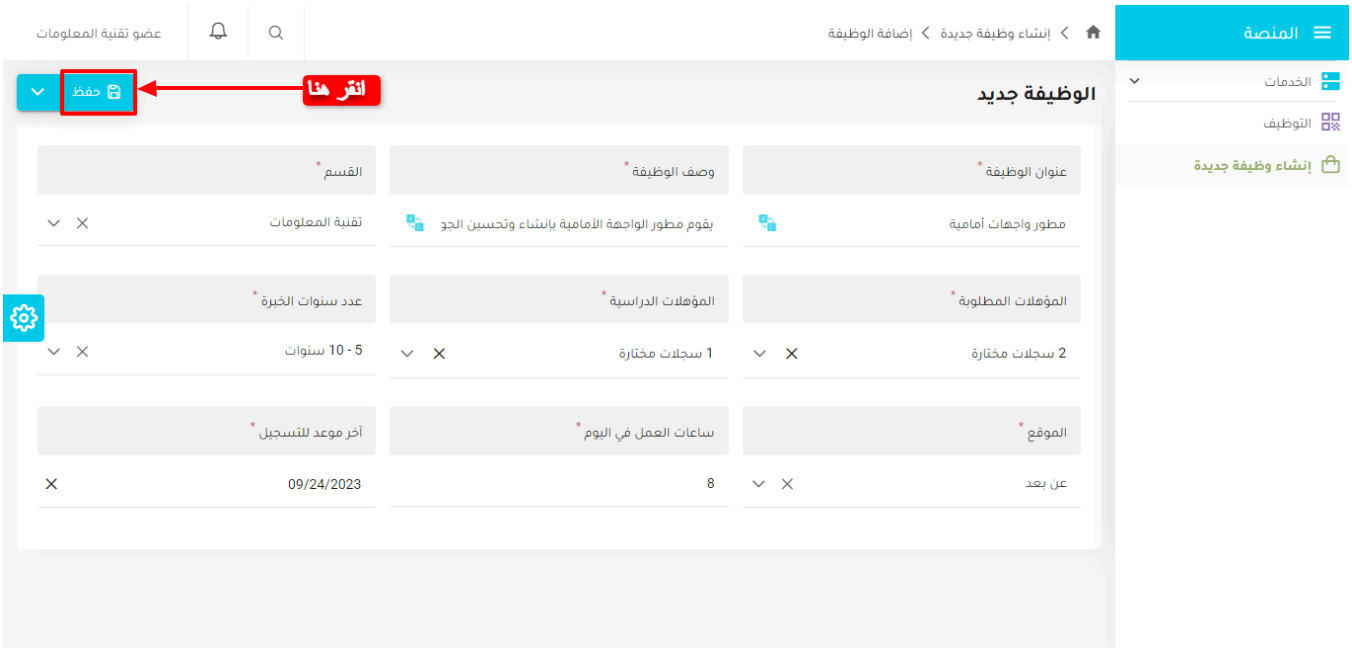
3. Create a new job:

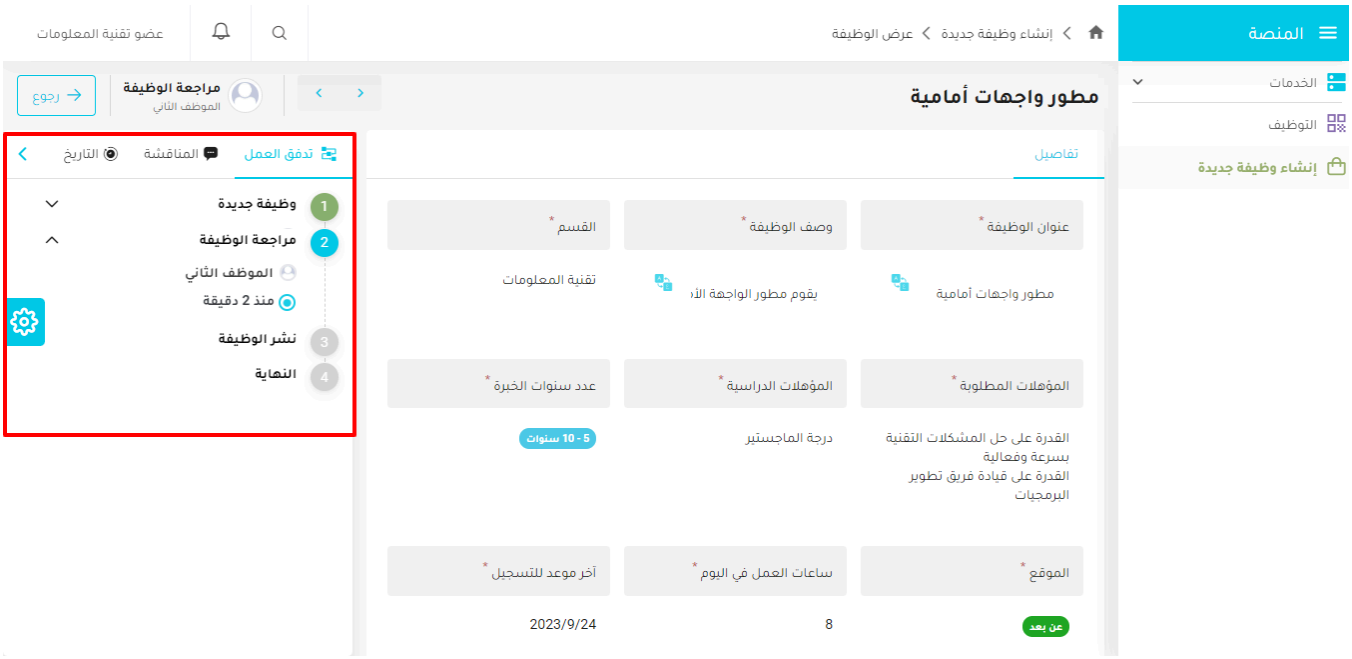
- In the IT member's browser, open the *Recruitment System* page from the sidebar.
- In the quick links, select *Create a new job*.





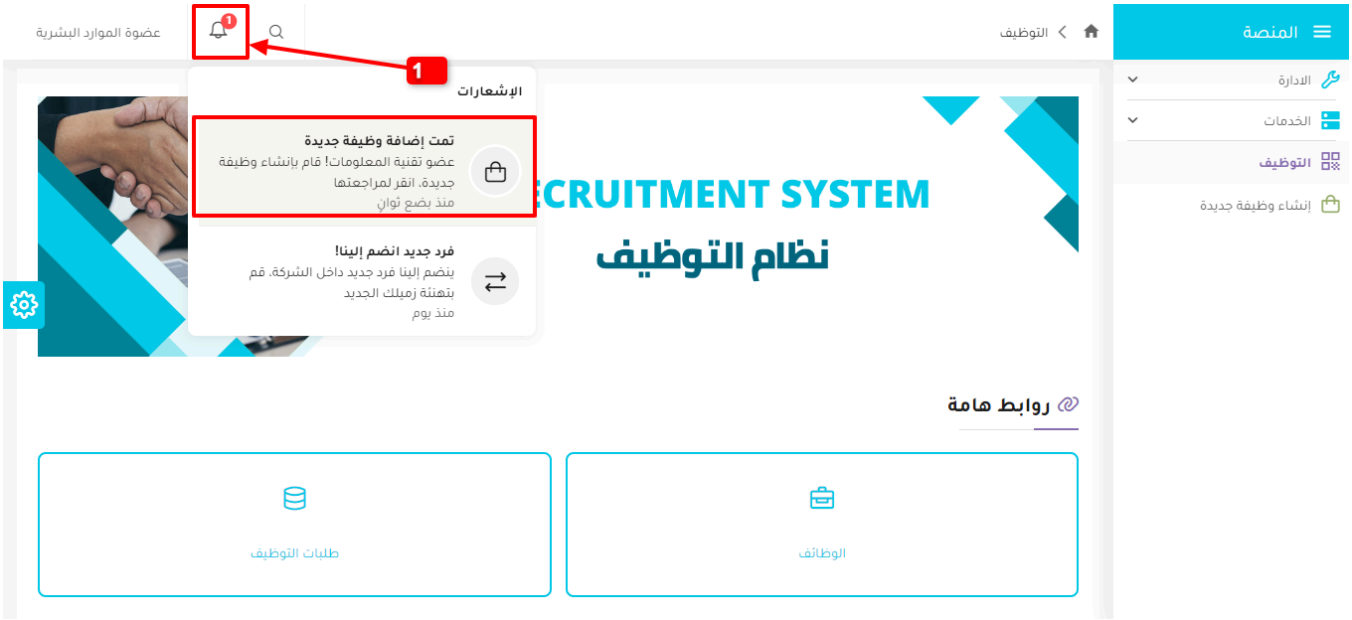
- Fill in the form and submit it.



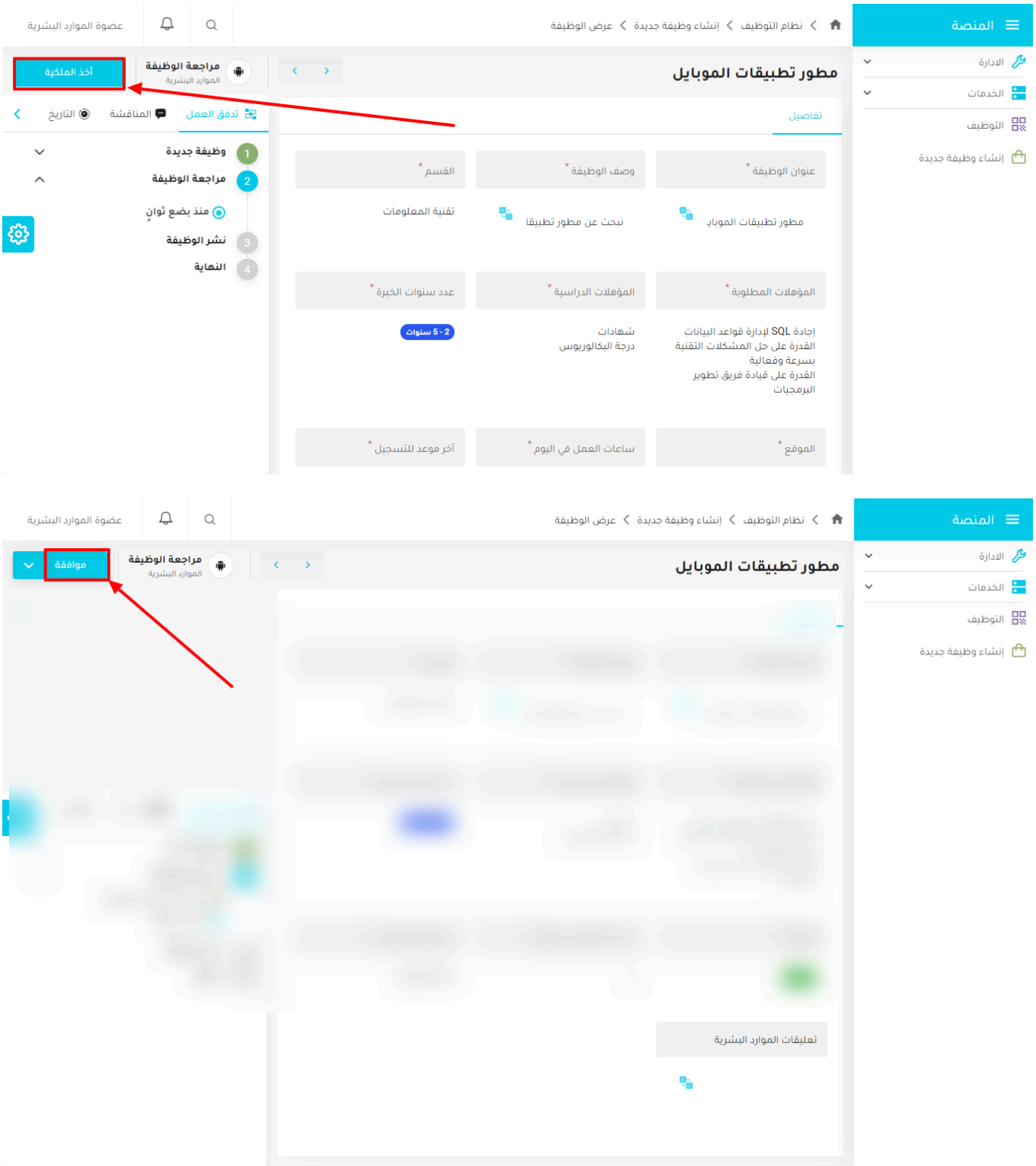


4. Approve the job:

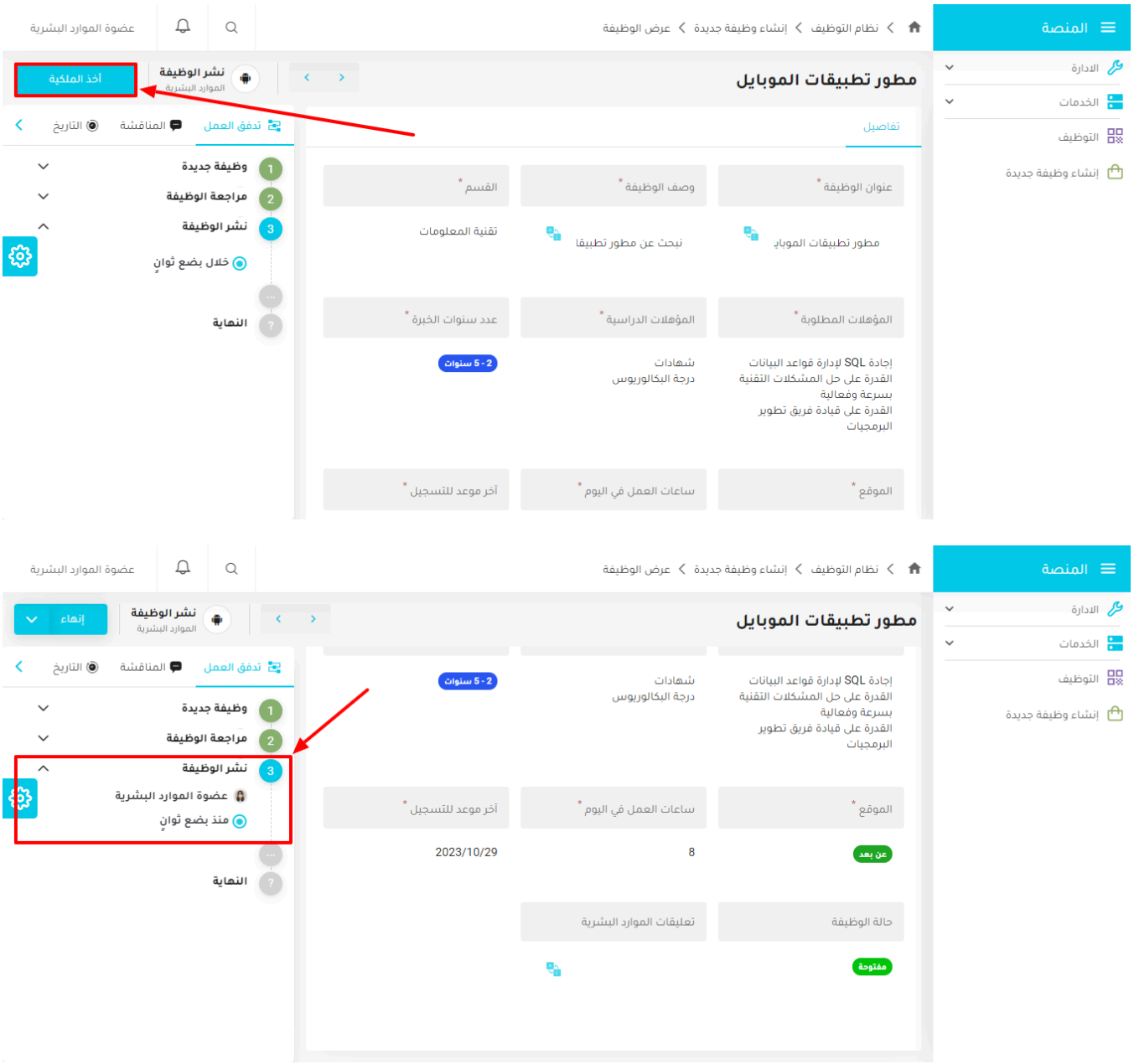
- Go to the HR member's browser.
- A notification tells you that a new job has been created.



- Select the notification and approve the job. If the job is waiting for someone in HR to pick it up, select **Take Ownership** first.



- Publish the job.



5. Check the job:

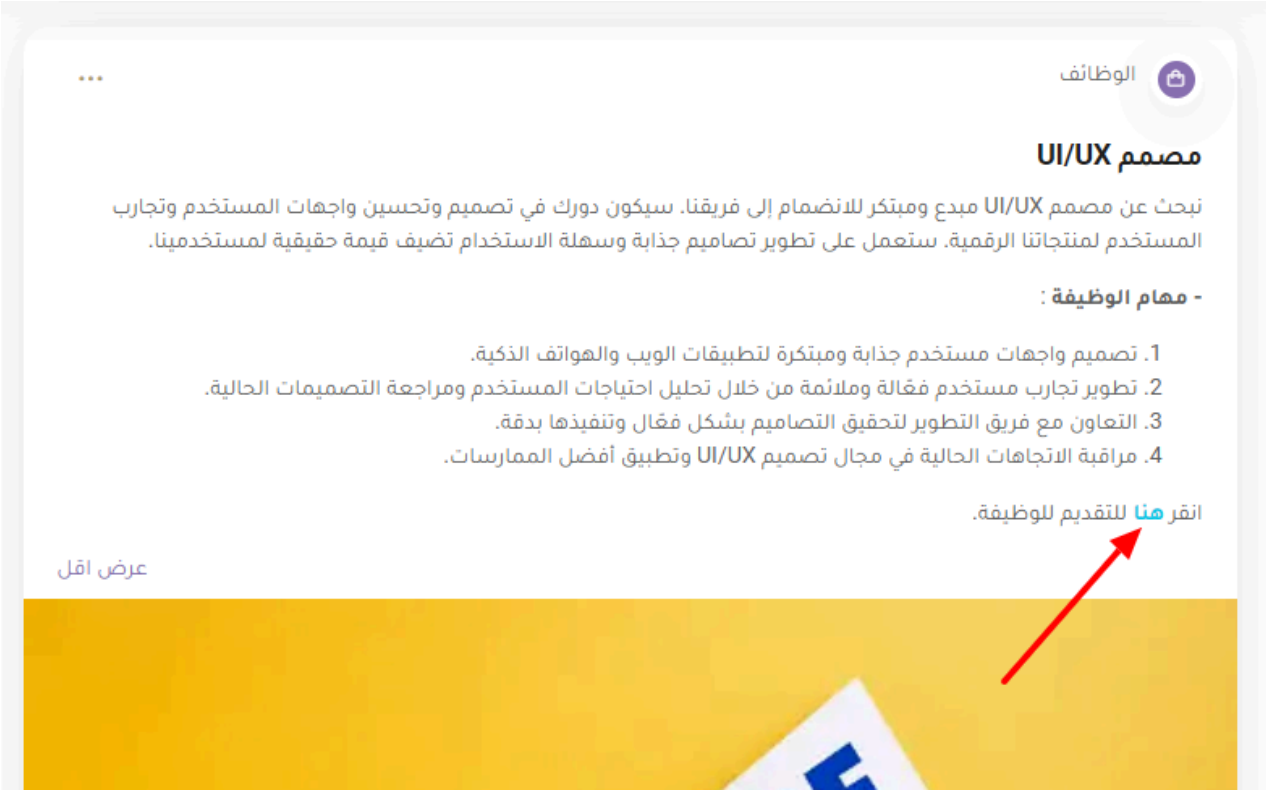
- Open a private browser window, where you are not signed in, and open the recruitment system's home page or the *Available jobs* view.
- The new job should appear in the timeline. Check that it shows correctly.

Apply for a job and follow the application

- This part involves three roles:
  - an HR member
  - a visitor who is not signed in. The email address they enter in the application is the one the candidate's emails go to.
  - the HR member who becomes the candidate's contact person

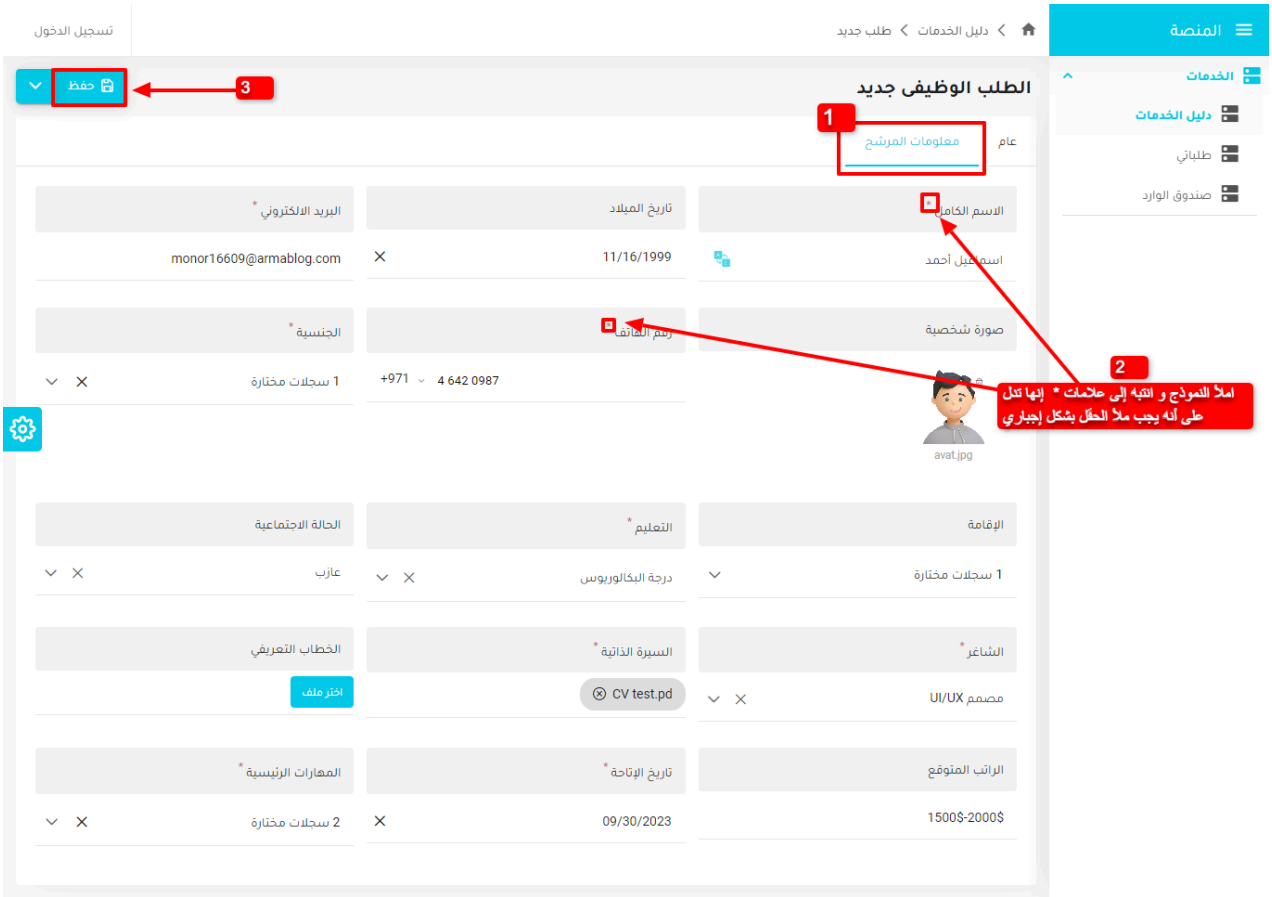
6. Apply for the job:

- In the private window, select the link in the post of the job you created.



- A new job application form opens.

- Fill in the form and submit it.

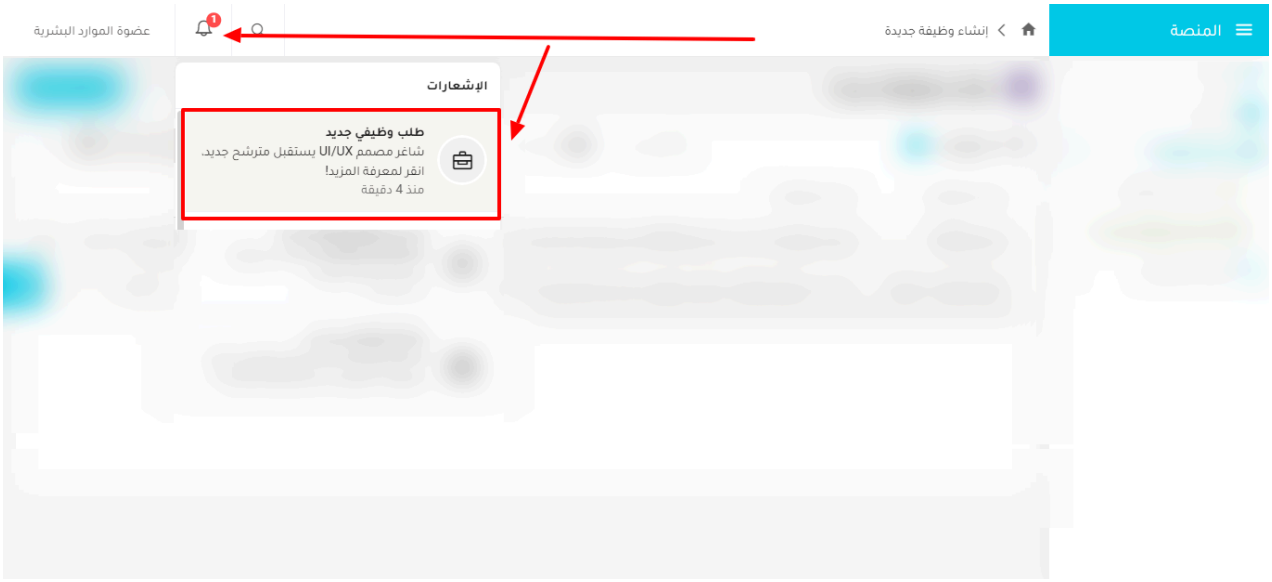


- This message confirms that the application was saved:

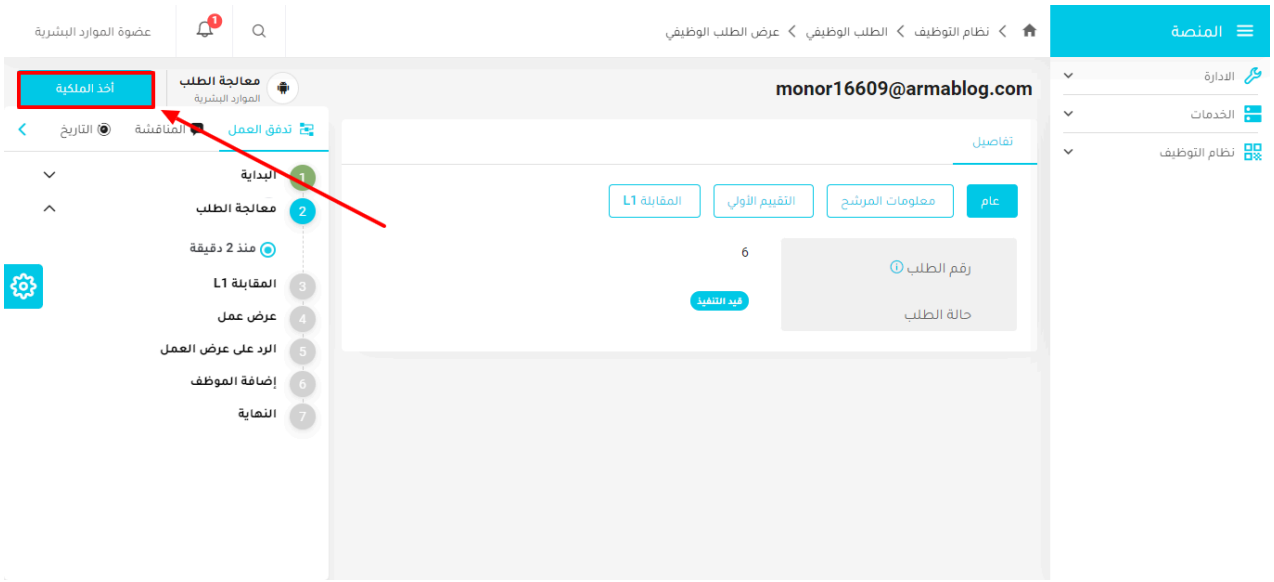


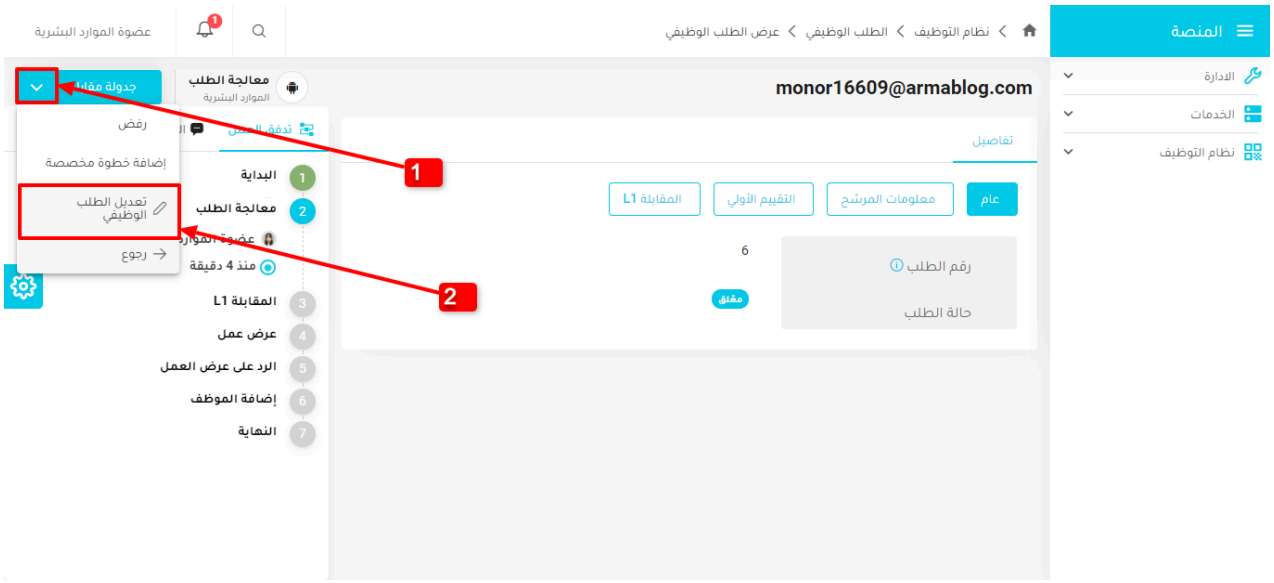
7. Review the application:

- Go to the HR member's browser.
- A notification tells you that a new application has arrived.



- Select the notification to open the application.
- The application is at the *Request processing* step.





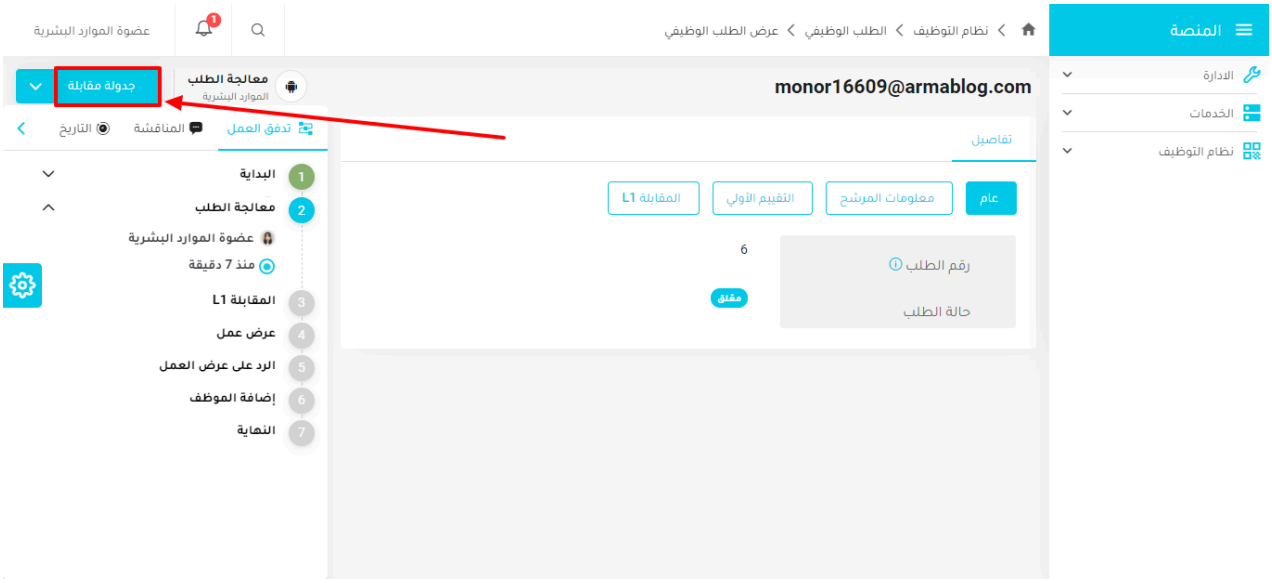
- Edit the application and fill in the initial evaluation.



- If the candidate qualifies, choose the contact person and enter the details of the first interview.

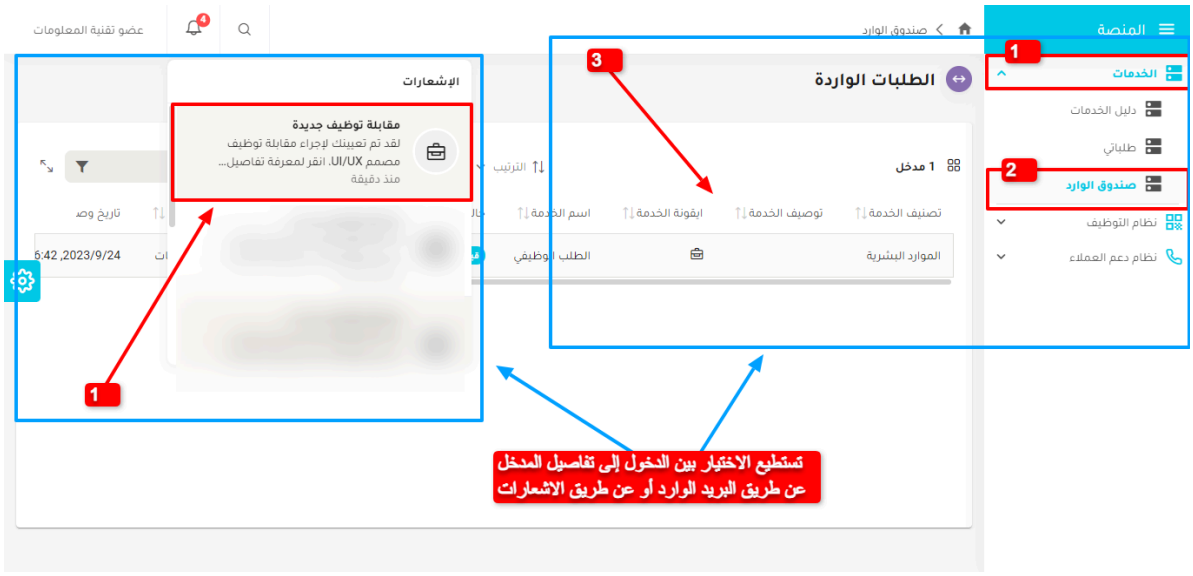


- Select *Schedule interview* to move the application to the next step.

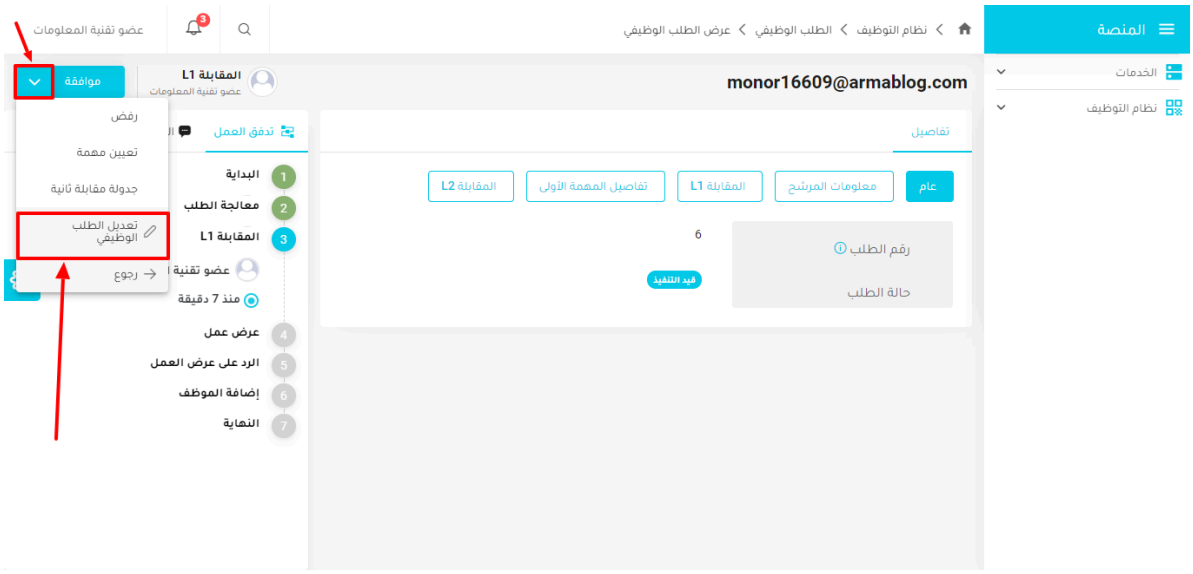


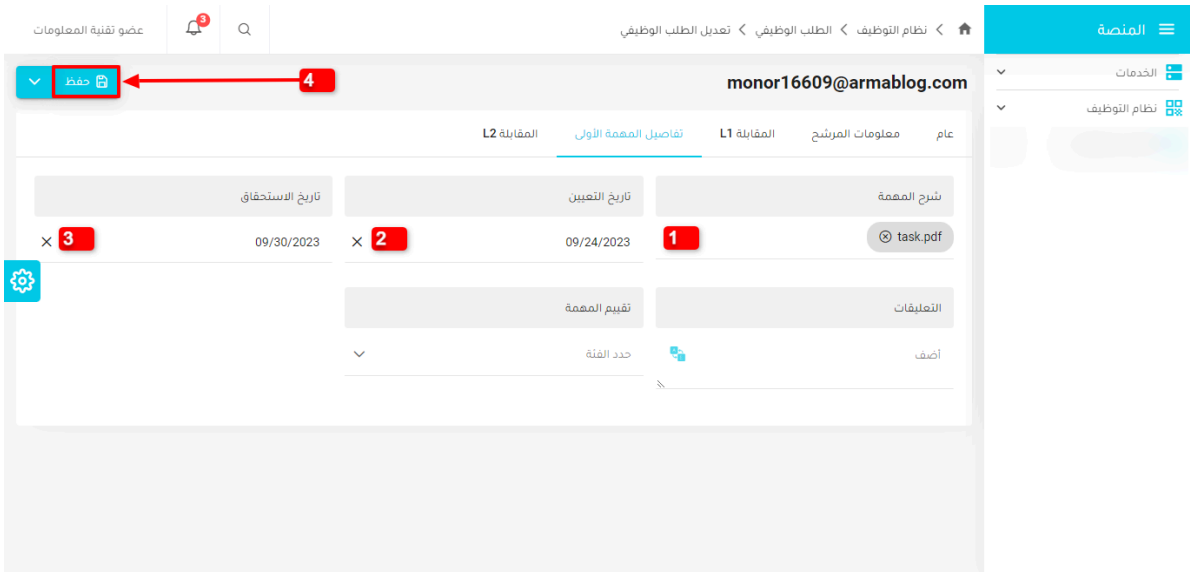
8. Interview L1:

- An email with the interview details reaches the address entered in the application (the candidate's email), and the interviewer and attendees get invitations.
- Assume the interview went well. Go to the HR member's browser.
- Open **Services › Inbox**.
- Find the application, or open it directly from the notification you received.

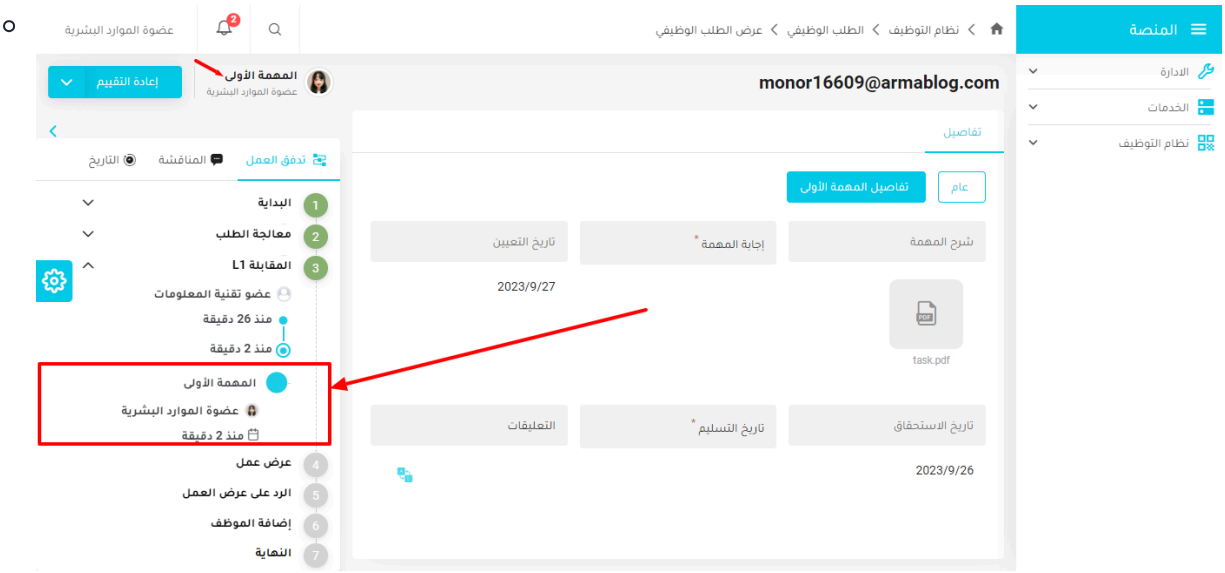


- Edit the application again and enter the first task's details.





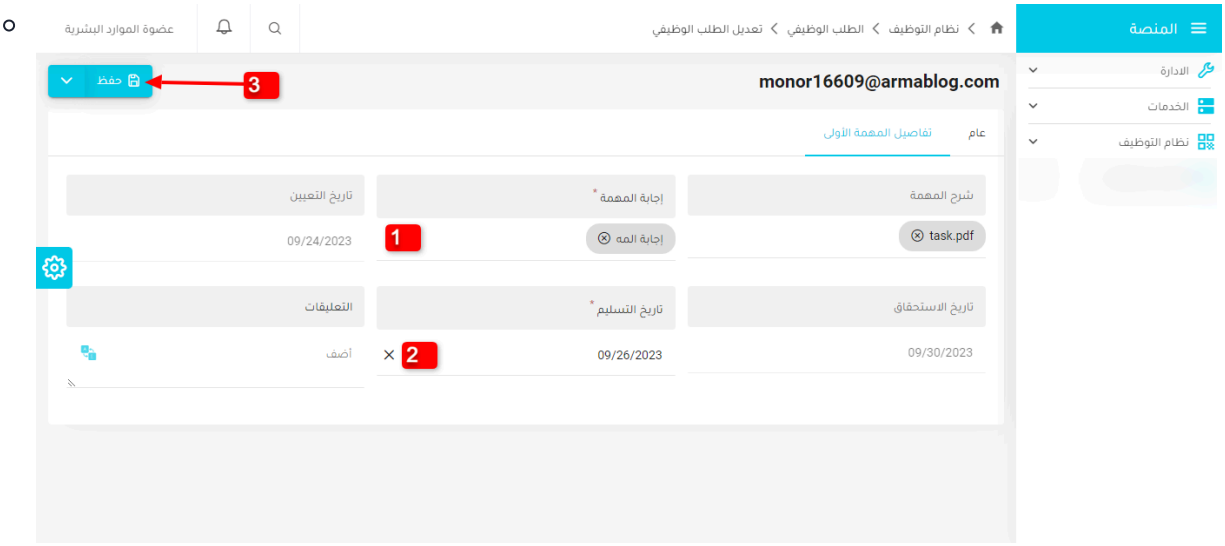
- Save the application, then select *Assign task*. The application moves to the *First task* step, with the contact person.

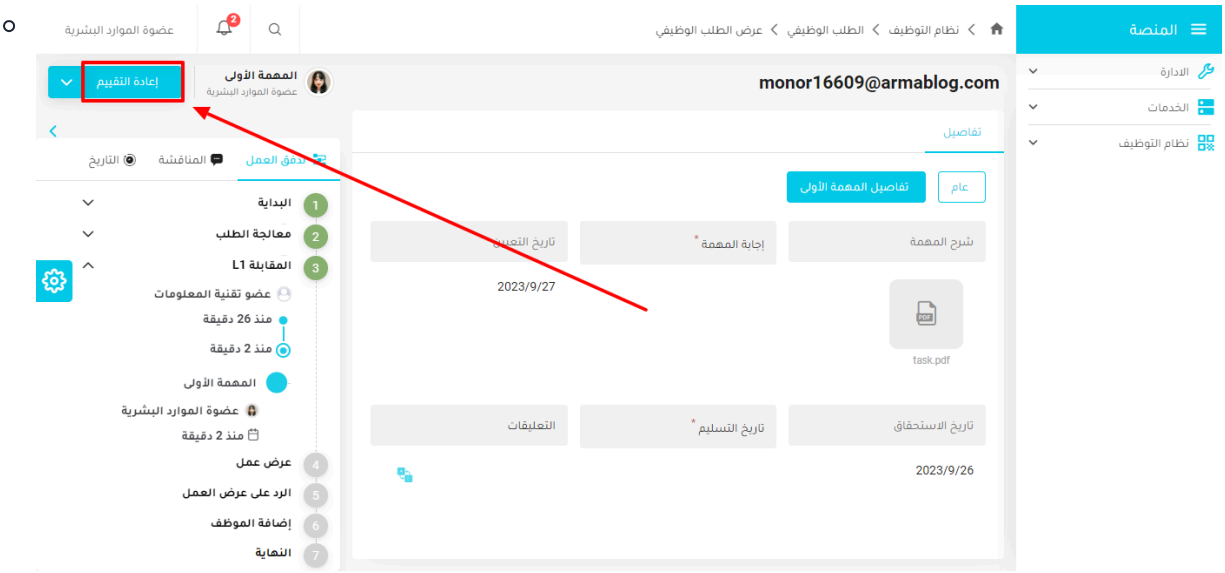


9. First task:

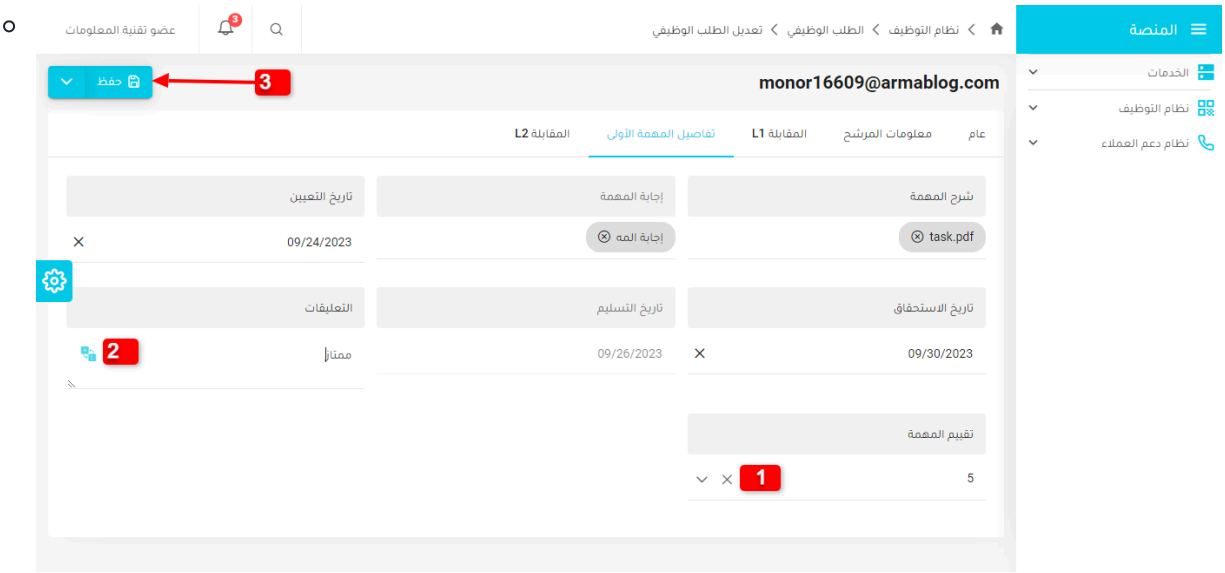
- The candidate receives an email with the task details and the contact person's email address, for answers and questions.

- When the contact person receives the answers, they enter them in the application and select *Re-evaluate* to move it on.

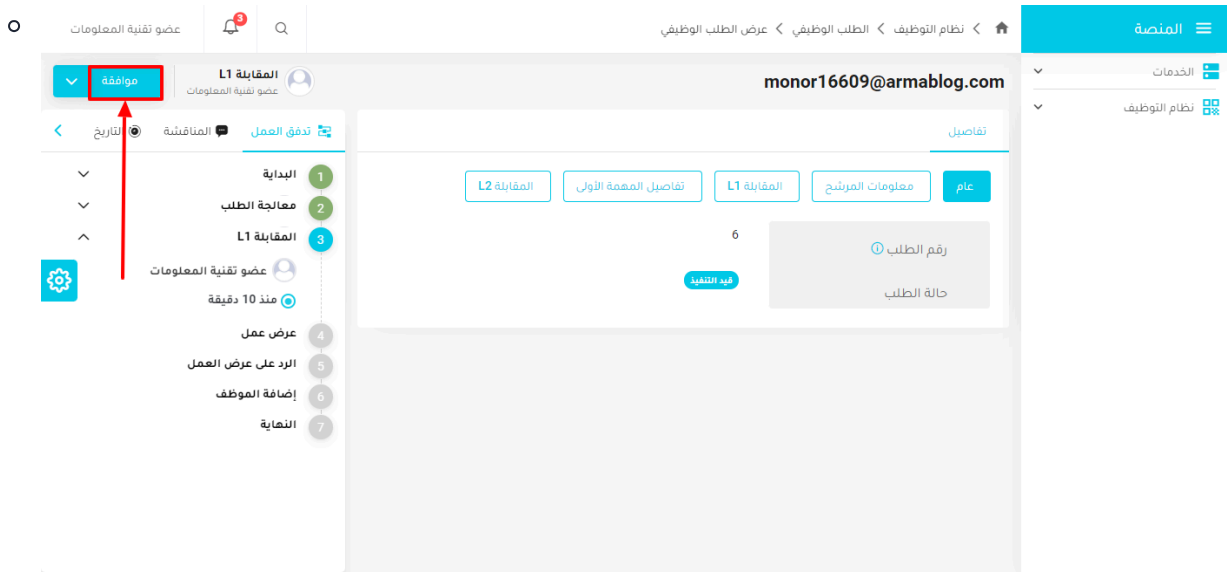




- The application returns to *Interview L1*. Assume the candidate is accepted and needs no second interview.
- Go back to the browser of the HR member who owns the interview step.
- Edit the application, enter the candidate's evaluation, and save.

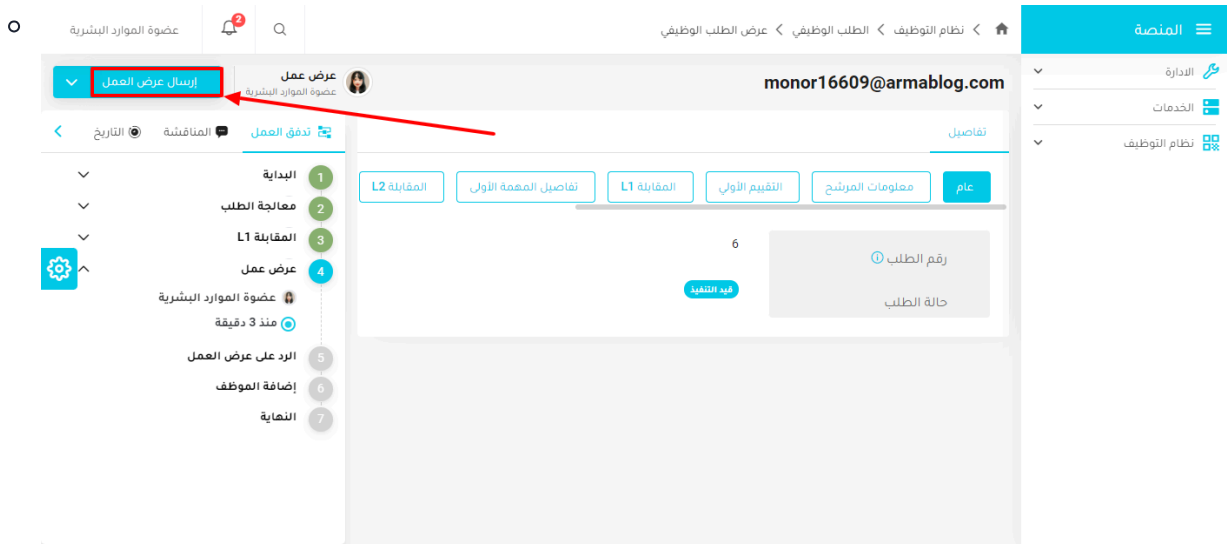


- Select *Approve* in the actions to move the application to the *Job offer* step.

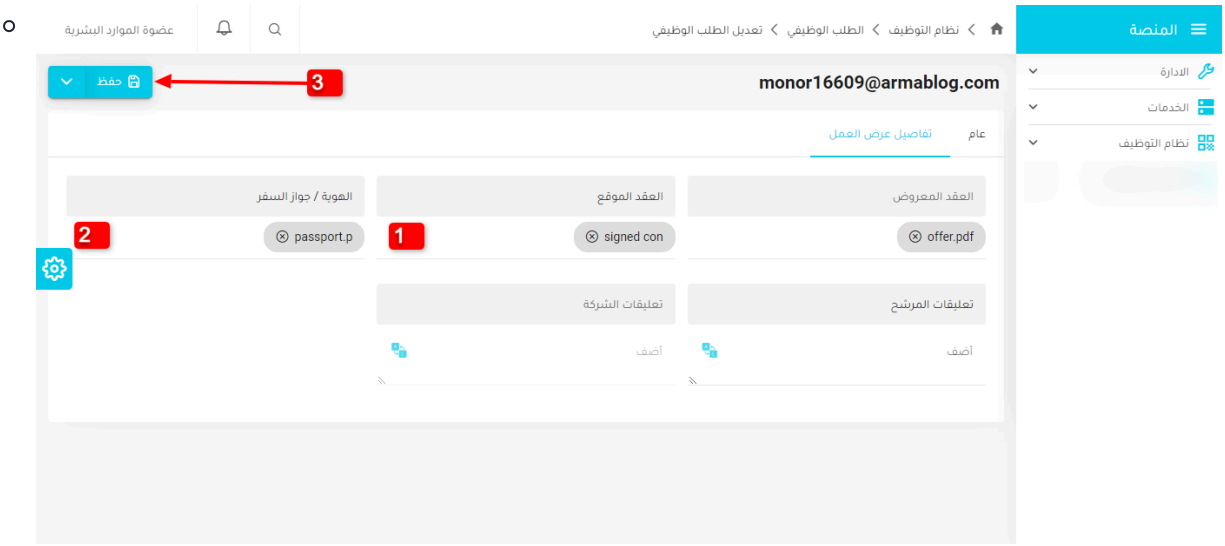


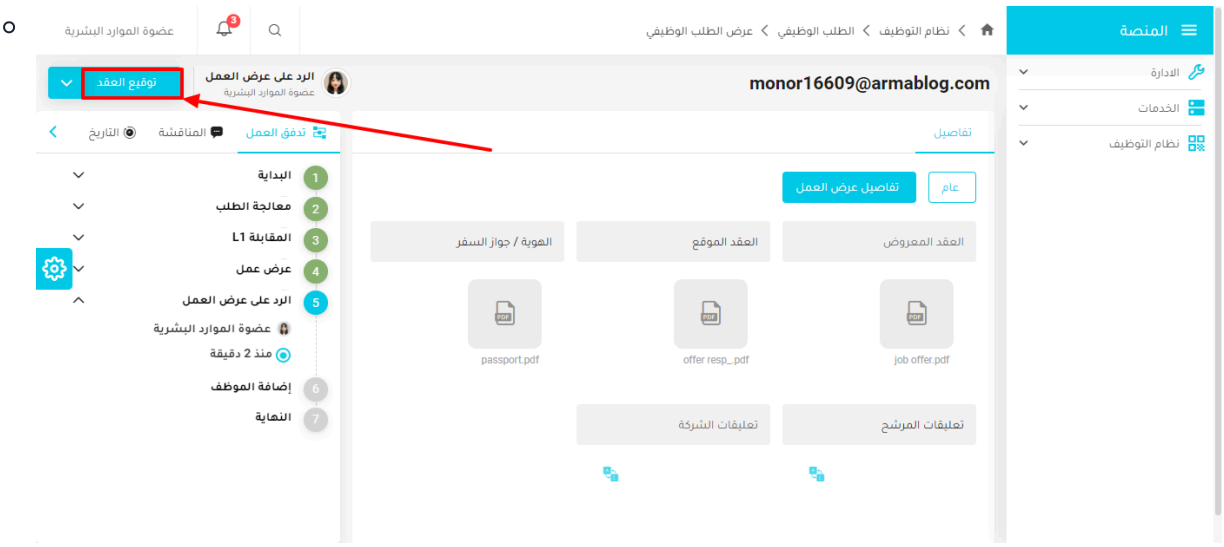
10. Job offer:

- In the HR member's browser, open the application from **Services › Inbox**.
- Select **Take Ownership**.
- Edit the application, attach the contract on the job offer tab, and save.
- Select *Send job offer*.



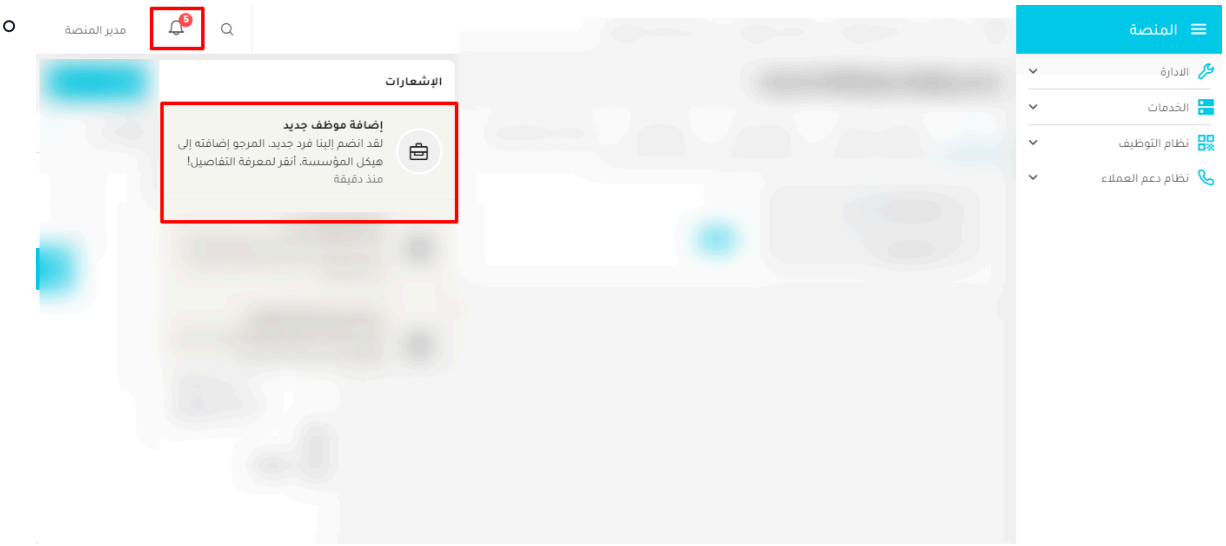
- The candidate receives an email with the offer details, and the contact person's email address for their answer. When the contact person receives the answer, they attach the files on the offer tab and select *Sign contract* to move the application on.



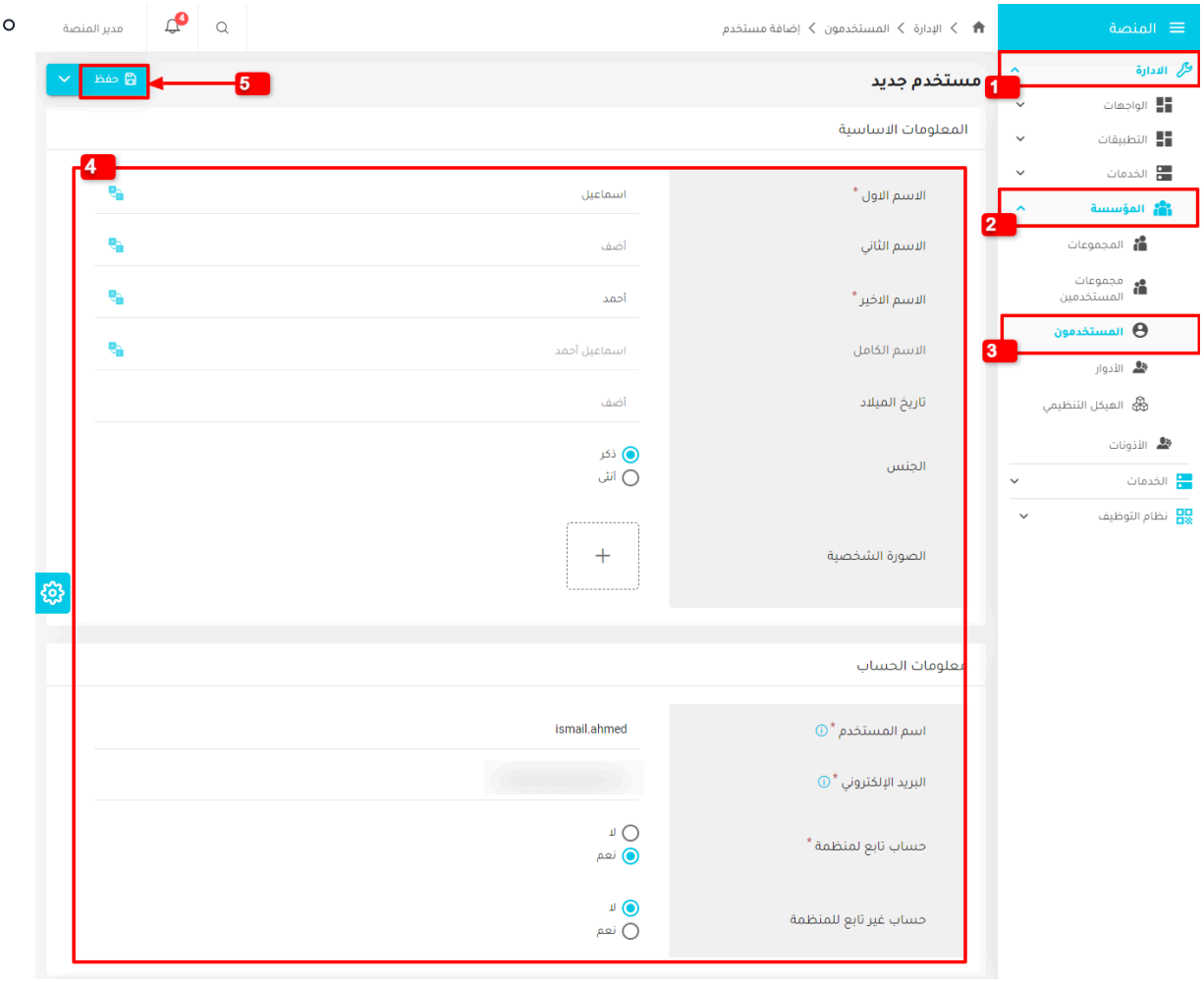


11. Add the employee:

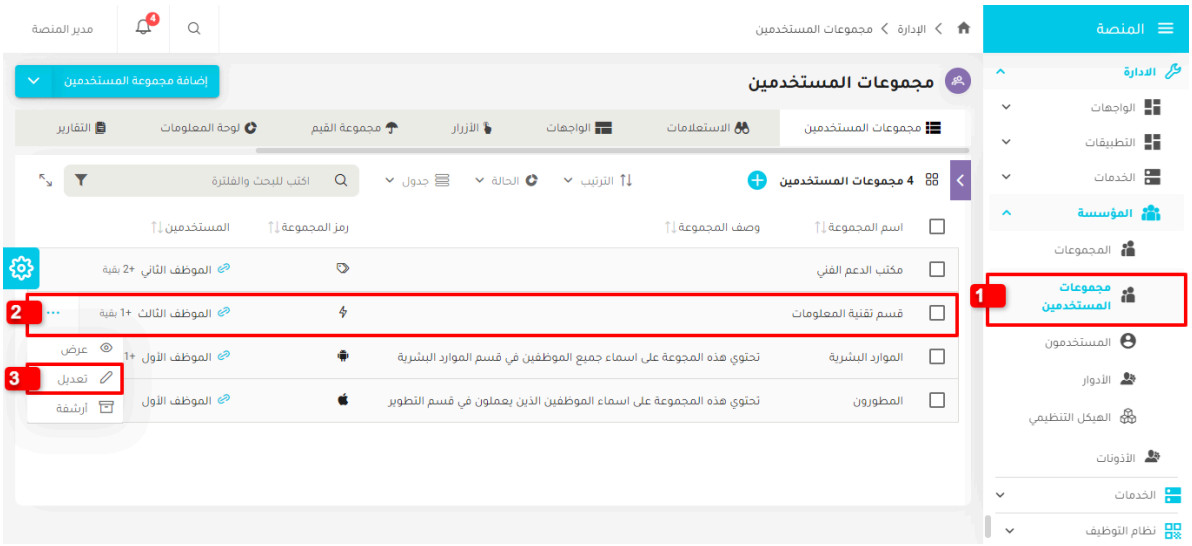
- The platform administrator receives a notification to create the user.
- In another browser, sign in as the platform administrator.
- Select the notification to see the details of the employee to add.

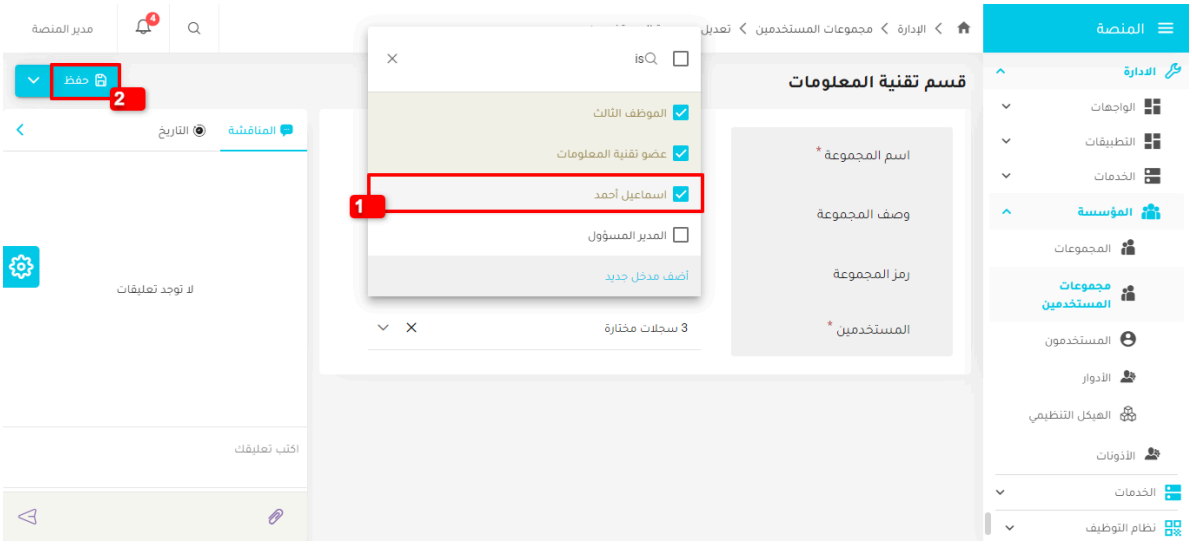


- Open **Administration › Organization › Users** and add a new user.

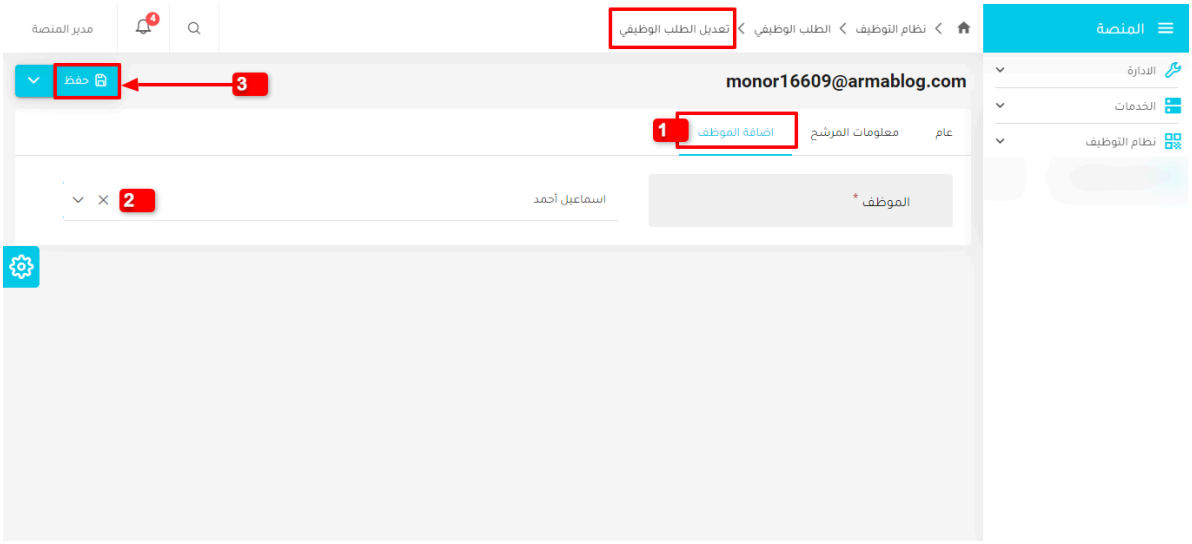


- Open **Administration › Organization › Groups** and add the user to the *IT* group.

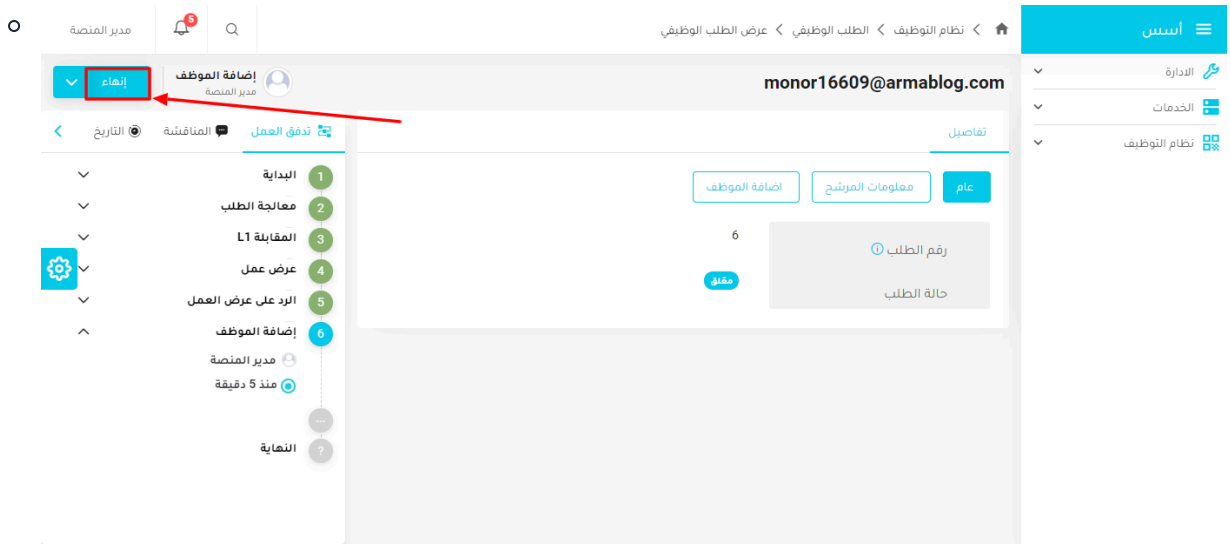


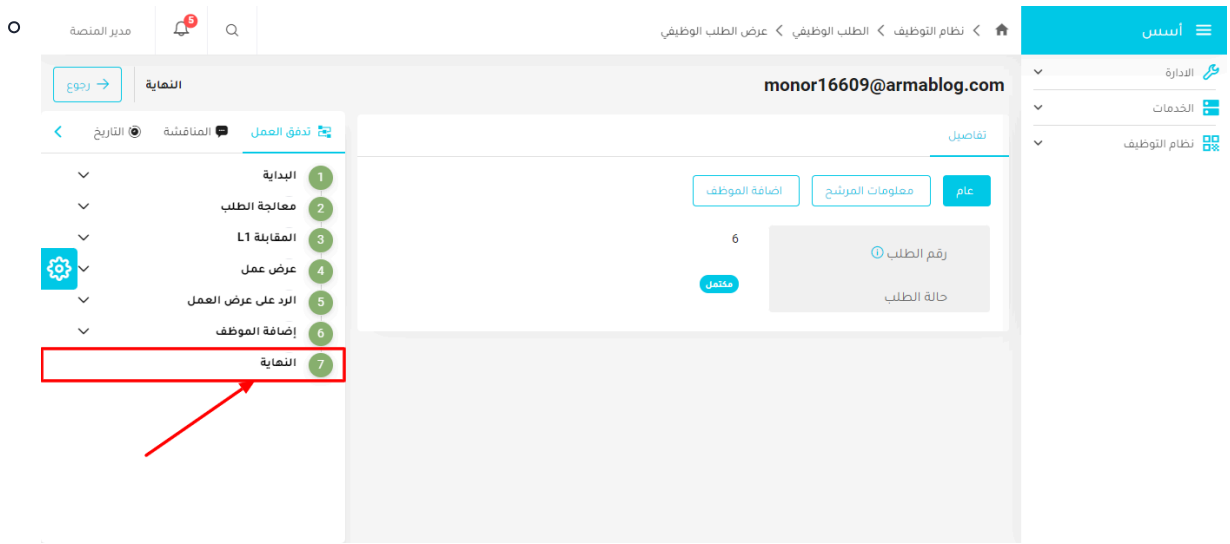


- Go back to the application and pick the new user in the *Employee* field of the add-employee tab.



- Finish the workflow.





Welcome the new employee

- Sign in as the HR member.
- A notification asks you to add a reception for the new employee.
- From the sidebar, open *Recruitment System › Create receptions*.
- Create a new reception.

Move the system between environments

Once the system works, you can move it from the development environment to production, or share a copy, with **Packaging**. A package carries the entities, services, views, queries, reactions, workflows and pages together, and is imported into another workspace, where its users and groups are matched to the ones there. See Packaging & distribution [Integrate & Extend].

Build apps FAQ

Common questions from **builders** — people who design applications in **Administration** without writing code: modeling data, designing views and forms, services and workflows, automation, pages and dashboards, and moving an app to another environment. Questions are grouped by topic; select a question to expand its answer. Each answer links to the guide that covers the topic in full.

> **Integrating through the API?**
>
> Questions about tokens, endpoints, error codes, webhooks, functions and package formats are in the Integration FAQ [Integrate & Extend].

Data and information modeling

How do I create databases on the platform?

There is no standalone "database" concept in OSOS. Instead you work with modules and entities, which play the role of databases and their tables. A module groups related entities into one navigable, shareable unit (think of it as an "app"), and each entity defines the fields its records carry. Follow the linked guides to learn how to create them.

What is a module, and how does it relate to entities?

A module is the top-level container that organizes the platform — it groups related entities (and, through them, their views and pages) into a single unit, and it is the unit of navigation in the sidebar and of packaging across spaces. A module does not embed its entities; each entity instead carries a reference back to its module. Modules are independent of one another; where they need to interact, they do so through a virtual (proxy) entity rather than a direct dependency.

Can I change an entity's normalized name or type after I create it?

No. An entity's **normalized name** (its machine name) and its **entity type** are fixed at creation and cannot be changed afterward — they are part of the entity's permanent identity, and the machine name is what queries, reactions, and import/export bind to. You can still freely change the localized **display** name, icon, singular/plural entry names, description, fields, workflows, and reactions later.

Can I move an entity to another module?

Yes, if you are a system administrator. You can move one entity, or several at once, into another module. Its views and lists move with it, and its normalized name and type stay the same. Virtual entities can be moved too. An entity cannot be moved into the protected core module, or into a module that is being permanently deleted. See Admin move, bulk move and what a move repairs. *(Since 1.10)*

What are virtual entities, and when should I use one?

A virtual entity is a **read-only** entity whose entries are computed automatically from one or more other ("source") entities rather than entered by hand. Use one when you need a combined, reshaped, or rolled-up view of existing data that you can then browse, filter, reference, and report on like any normal entity. You never create, edit, or

delete its entries directly — you maintain the source data and the platform keeps the virtual entries in sync (with a short propagation delay, since the refresh is event-driven).

What's the difference between Proxy, Joiner, and Aggregator entities?

They are the three kinds of virtual entity:

- **Proxy** — a 1:1 re-exposure of a single source entity, so the same data can appear under a different name, module, or filtered subset without being duplicated.
- **Joiner** — starts from a primary source and joins related entities through relations, presenting fields from all of them together as one flattened entity.
- **Aggregator** — assembles a purpose-built entity from several sources with explicit field mappings, and can compute aggregated values (count, sum, min/max, average).

Are there limits on how much data or how many records I can store?

Individual field types keep their own size limits. Since 1.10, your organization's plan can also limit how many **records** it holds: only entities you build count, and an archived record counts as a quarter. On a shared installation, system administrators are warned at 80 % of the limit and again at 100 %. New records are refused from 10 % above it, and reading, editing and exporting never are. Installing an app that would pass that number is refused before it starts. See Plan limits [Administration › Your license & plan limits]. *(Since 1.10)*

Can I link straight to a specific tab of the entity designer?

Yes. The designer's active tab is reflected in the URL as a named `?tab=<slug>` query parameter, so a tab is an addressable place rather than transient UI state. You can bookmark or share a tab, refresh without losing your place, and step **Back** and **Forward** through the tabs you opened instead of jumping straight out of the entity — opening a tab adds a browser history entry.

The slugs are `entries`, `queries`, `views`, `reactions`, `announcements`, `buttons`, `lists`, `dashboard`, `reports`, `relations`, `templates`, `stores`, `information`, `lock-history`, `functions`, `permissions`, and `scripts`. A URL with no `?tab=` opens on **Entries**; an unrecognised slug is ignored and the designer opens on the tab it last remembered. Older **numeric** deep links (`?tab=1`, `?tab=3`, …) are no longer recognised — replace any you have kept with the named slug. See Entities. *(Since 1.8)*

Why does quick edit now reject an edit it used to accept?

Because inline (quick) edit is applied as a **patch**, and its validation used to be narrower than the entry form's: a patch was blocked only when a validation error named a field the patch actually touched, or when a Standalone/Value **function** validator rejected the change. An **entity-level custom validator** — the business and uniqueness rules an entity enforces across its fields — raises an error that is tied to no single field and is not a function validator, so those errors were silently dropped and a quick edit could save even though the rule had failed. The full create and update paths never had this gap.

Any error raised by an entity-level custom validator now **blocks the quick edit**, and the validator's own message is returned instead of being flattened into the generic list of field-validation errors, so the user sees the rule's real explanation. For example, quick-editing a member's role to *Manager* on an org unit that already has one used to save quietly; it now blocks and surfaces the "multiple manager role not allowed" message. See Entities. *(Since 1.8)*

Can I permanently delete an entity or a whole module?

Yes, since 1.10, if you are a system administrator. A custom, request or virtual entity can be deleted permanently together with everything it owns, and a whole module together with everything in it. This cannot be undone. Unlink first: the delete is refused while anything outside still points at the entity or module, and for anything installed from a package. See Permanent delete and Modules — Permanent delete. *(Since 1.10)*

Can I add my own database indexes to an entity?

Yes. The entity designer's **Indexes** tab lets an administrator add up to **10** indexes per entity — 1 to 8 fields each, optionally **Unique**, optionally limited by a filter — to speed up a frequent filter or sort, or to make a combination of fields unique. A unique index needs a filter that leaves out empty values; otherwise every record with the fields empty counts as a duplicate. The tab is not offered for core-module, request or virtual entities. See Entity indexes (Indexes tab). *(Since 1.10)*

Does permanently deleting entries also delete their copies in a virtual entity?

Yes — and since 1.10 that also holds for a **bulk** permanent delete. A proxy, aggregator or joiner entry is a derived copy of a source entry, linked to it by an index. Only the single-entry delete used to remove those copies; a bulk delete left them behind, orphaned. Now each of the three cascades runs for every entry in a bulk delete, a failure makes the whole delete retry, a retried cascade finishes every copy, and a stale index row no longer stops the cascade. This matters for files too: once the source's files are released, file clean-up could otherwise remove files an orphaned copy still shows. See Derived entries and entry permanent delete. *(Since 1.10)*

Fields and data types

What types of data can I store?

A wide range of field types is supported, covering text, numbers, dates and times, multimedia, and composite data sets such as files and maps. See Field Types for the full list.

Can I add custom data fields?

Yes. Custom fields are added with drag-and-drop, and many field types are available. See Field Types for the full list.

What is the difference between a storage data type and a field kind?

Every field rests on one of about 15 **storage data types** (text, number, date, boolean, entity reference, JSON, and so on) that describe how the value is stored and cast. On top of those sit roughly 35 **field kinds** — the semantic field a builder actually chooses (plain text, long text, password, phone, image, reference, category, switch, map, …). A single storage type backs many kinds: text, for example, backs plain text, long text, password, and phone, each distinguished by its edit component. That is why two fields that store the same underlying value can render and behave very differently.

Can I make a field required, unique, or multi-value?

Yes. Field properties include required, unique, default value, multi-value (an array of the field's own values), repeatable (the field or block repeats), visibility, locked, and sensitive. Note that not every kind supports every option — a JSON field, for instance, cannot be marked unique or searchable, while a sequence field is always auto-increment and read-only. Multiple values on a **Category** field are new in 1.10, and turning them on is a breaking change — see Multiple values.

Why can't I search, filter, or sort on a JSON field?

The contents of a **JSON** field are not queryable or searchable — the platform treats the value as an opaque blob. If you need to filter, search, or sort on a piece of data, model it as a real field of the appropriate type rather than nesting it inside a JSON field.

What is a sequence (auto-increment) field, and can I reformat its numbers?

A sequence field stamps a new running integer onto each entry when it is created (for example a request or invoice number). The number is a plain integer — there is no built-in prefix, suffix, or zero-padding on the counter itself. Values are allocated only on **create** (never re-allocated on edit), and the numbering is effectively locked once the first value has been issued: after that you cannot change the counter key, and with conditional (multi-stream) numbering you cannot change the key set either. You can also run independent streams — for example resetting numbering per year or per department — using a key expression.

Can a field's value be calculated automatically?

Yes. A field can be **calculated** from an expression rather than typed in, with recalculation timed to form open, save, entry change, or specific field changes. The platform has a built-in, sandboxed expression and token engine for this — you do not need to deploy a function for computed fields, dynamic values, or default values.

Why did a calculated percentage render with more decimals than I configured?

Because the precision you set used to be applied only as the user *typed* a number — the edit component rounded on input, and a value written by a calculation or an expression went straight to display with its full floating-point precision. A percentage field configured for 0 or 1 decimal places could therefore render something like `33.333333333333336%`.

Display now rounds **every** percentage value to the field's configured precision, calculated values included. Both percentage display styles are covered — the plain percentage label and the *Progress Bar with Percentage* label; the progress bar's own fill is unaffected. See Number fields. *(Since 1.8)*

Should I mark a field Sensitive or Encrypted?

Pick one — they are mutually exclusive, and the field editor silently switches the other off. **Sensitive** stores the value normally and masks it on the way out, so it can still be filtered and searched by a grant holder, can still be aggregated (the figure comes back as `****`), and can still be the entity's main title. **Encrypted** stores ciphertext, so the value is protected even from a caller running with authorization skipped — but it can never be searched, filtered, sorted or aggregated by anyone, it cannot be the entity's main title, and turning Encrypt on force-clears **Searchable**. Reach for Encrypt when the stored bytes themselves must be protected; otherwise Sensitive costs you far less. See Sensitive Fields. *(Since 1.10)*

I turned Encrypt on and nothing seems to have changed. What did I miss?

There is **no data backfill** behind the toggle. Changing `encrypt` is treated as a versioned breaking schema change: a *Field encryption changed* reason is added to the new entity version, every affected field is stamped with the transition version, and existing entries below it become **Obsolete** until an administrator resolves them through the ordinary **Upgrade Records** flow. For the whole of that window the field is excluded from every builder surface. One trap in the other direction: **Auto Repair is refused while Encrypt is being turned off** — the stored value is still real ciphertext and Auto Repair would promote it to the entry's real value, so call Decrypt on the entry (which stays

reachable precisely for this case) and supply the plaintext as the dialog's `Fixed` resolution instead. See Toggling Encrypt. *(Since 1.10)*

I set Encrypt on a Switch field and it saved, but the field is not encrypted. Why?

Because four kinds of field can never be encrypted, and the server **silently normalizes** `encrypt` **back to** `false` rather than rejecting the save: reference/relation fields, block containers, **Switch** and **Category**. The two exclusion lists are deliberately different shapes — Sensitive excludes only reference and block fields, so a Switch or Category field *can* be sensitive but never encrypted, while a **mapped** field is hidden from the Sensitive toggle but not from the Encrypt one. Everything else is encryptable. Separately, an encrypted (or mid-transition) field cannot be the entity's **main title**, though it is still allowed as the sub title. See Which field types can be encrypted. *(Since 1.10)*

Can a Category field hold several values, and what happens when I turn that on?

Yes, since 1.10: turn on **Multiple** in the field's value settings, optionally with a minimum and maximum number of selections. Turning it on is a **breaking change** — the entity's version goes up and existing records are converted when they are upgraded. You cannot turn it on while a Kanban board, grouping, sort, dashboard or timeline uses the field, and a multi-value category can never be the main category, a Kanban column field, a default sort or a dashboard group-by. See Multiple values. *(Since 1.10)*

Can end users add or edit a Category field's options themselves?

Yes, from a record's create or edit form, when two settings line up: **Allow users to manage options** is on for the Category field (not available when its options come from another field), and the view's create or edit operation allows **Add category options** or **Edit category options**. The changes are saved together with the record; if someone else changed the options at the same moment, the user is asked to reload. See Letting users manage options. *(Since 1.10)*

Relationships and references

Can relationships be created between entities?

Yes. Entities can be related to one another, which lets you traverse related entries, show them as sub-lists, build hierarchical structures, and drive relation-based automation. See Relations & References for details.

What is the difference between a reference field and a data relation?

They are two related mechanisms for linking data:

- A **reference field** is a field of data type *Entity* on entity A that holds the actual pointer to an entry in entity B. The link is stored as a stable pointer (entity id + entry id), not as text, so it survives even if the target's display title later changes.
- A **data relation** is a first-class, named relationship between two entities, managed from the entity's **Relations** tab. Relations are what power "related entries" tabs, sub-lists, Joiner virtual entities, and relation-sourced reactions.

What do "incoming" and "outgoing" relations mean?

Direction describes which side owns the reference field. An **outgoing** relation is computed from the current entity's own reference fields (this entity points out — it acts as the "child"). An **incoming** relation is a stored definition describing entries in another entity that point *into* this one (this entity acts as the "parent"). Both are viewed from

the entity's Relations tab; outgoing relations are ephemeral (computed on the fly) and cannot be edited or deleted directly.

Why isn't my mapped field updating when the source changes?

Mapped fields copy values from a referenced entry, and each mapping has an **update mode** that decides who wins when the source and local values disagree. A value "not updating" is often the intended behavior: **Never** keeps a defined local value and never overwrites it; **Local** keeps the local value unless the reference itself changes; **Parent only** always takes the source; and **Two-way** lets a child edit propagate back. Also, only fields whose mapping is explicitly *enabled* are written at all.

Can I prevent circular (looping) relations?

Yes. A reference field can enable circular-relation denial, which stops relations that would cause infinite references. This is also a prerequisite for a self-referencing **Tree** view, where a field points back at the same entity.

Working with data — views, faces, and entries

Can data be deleted?

The default action is to **archive** a record (a soft delete) rather than remove it permanently. Archiving preserves the change history and lets you restore the record later. Archived records can then be permanently purged by an administrator or through automation (the "Delete Archived Entry" reaction action) when you genuinely need to remove them. Separately, since 1.10 a system administrator can permanently delete a whole entity or module, once nothing outside it still references it — see Permanent delete (purge).

Can data be filtered?

Yes. Data can be filtered from the filter icon, provided the administrator has enabled it, and you can filter by entry fields, entry metadata (created/updated by and dates), and — for entities with a workflow — the current or previous step and its owner. See Queries for the full filter builder.

Can data be exported, and to which formats?

Yes. Since 1.10 a view's **Allow export** setting adds an **Export Entries** button that exports every entry matching the list to **Excel (XLSX) or CSV**, and administrators get the same choice from **Export Entries** on an entity's page — CSV exists only in these two quick exports. The Table face's older export stays XLSX, a template export produces whatever the template defines (optionally converted to PDF), and reports stay XLSX and PDF; the mobile app has no quick export. Large exports run in the background with a *file ready* notification, and sensitive-field permissions are respected in the output. See Gallery quick export. *(Since 1.10)*

What display methods (faces) can a view present entries in?

A data view offers seven faces (display methods): **Table**, **Kanban**, **Calendar**, **Timeline**, **Tree**, **Badge**, and **Mini-card**. A single view can enable more than one, mark one as the default, and let the end user switch between them at runtime.

Which faces are unavailable on the mobile app?

The **Mini-card** face is not available on the mobile app — selecting it there renders an empty area, so use another face on mobile. Record reordering is available on Table, Badge, and Kanban; on mobile you can drag a Kanban card

to another column (it is saved), but reordering whole Kanban columns is not yet supported.

What is filter-first (search-first) loading?

Filter-first sets a view so that **no entries load until the user fills a filter form** — useful for large data sets or public lookup screens. You configure up to four filter fields (and can auto-fill those values into new entries), and it adds a human-verification step for anonymous visitors. Filter-first is mutually exclusive with the Tree face and with dashboards.

Can end users save their own filters or build personal lists?

Yes, where the view allows it. End users can save their filters as personal queries (requires filters enabled) and build their own entry lists (collections) based on their permissions. Administrators can also expose quick-filter chips and classification styles that end users apply on demand.

Can I preview a view before publishing it?

Yes. The builder's preview mode renders a view or page against deterministic **sample (fake) data**, so you can check layout, columns, cards, and action buttons without any real entries. A mobile-preview dialog shows how it looks on a phone. Preview is **not** a permission bypass: the request is still authenticated and authorized, and the only relaxation is narrow — because sample data uses fabricated ids, a genuine *not found* on that lookup is allowed through in preview instead of surfacing as a permission error. Even so, preview is a rendering aid, not an inspection tool — do **not** use it to judge what a restricted end user will actually be allowed to see.

How do I let end users reorder repeated values?

Turn on **Repeatable reorder** for the view's create or edit operation. It is set per operation, not on the field, so the same field can be reorderable on Create and fixed on Edit, or reorderable in one view and not in another. It becomes available once the operation includes at least one repeatable field or block, and switches itself off if you remove the last one. Users then get a drag handle on each repetition (when there is more than one), in the web portal and in the mobile app. See Data views. *(Since 1.8)*

I created an aggregate on the entity and no total appears anywhere. Why?

Because the definition and the decision to show it are two separate layers. An aggregate is defined once on the entity's **Aggregators** tab, and then has to be **enabled per view**, on each view's entries-list card. A view that enables none gets none: it does not inherit the entity's set, and it does not acquire an aggregate later because somebody added one to the entity. There is no "all views" switch, and that intersection is a deliberate security boundary — the server intersects what a caller asks for with what the view enabled, so a published or anonymous-facing view cannot start disclosing a new total behind your back. (Browsing the entity's own entries in the admin section, with no view bound, does compute every aggregate.) See Defined once, enabled per view. *(Since 1.10)*

Can a Kanban board be split by a reference field, and which columns does it show?

Yes, since 1.10 the column field can be a single-value reference field (to another entity's records, a List of records or a List of values) as well as a category field. The designer picks a column mode: **All** (every current value, including ones added later), **Used** (only values a card holds — reference fields only) or **Specific** (a hand-picked set); a view saved before 1.10 keeps rendering exactly as it did. A board loads at most 20 columns, and a multi-value or repeatable field cannot split a board. See Column field and column modes. *(Since 1.10)*

Can I rename a view or page that was installed from a package, or change its icon?

Yes. A view or page installed from a package is locked, but you can change its **name, description and icon** in the editor (an implicit view's name stays fixed). The editor shows the package's value beside yours, and your version is kept when the package is updated. Its publishing state, display order, audience and access rights stay yours too. Anything else needs the app to be forked. See Installed views and Installed pages. *(Since 1.10)*

Queries, search, and sorting

What is a query, and where can I reuse a saved one?

A query is a saved, reusable filter over one entity's entries — a set of conditions (each a field, an operator, and a value) joined with AND/OR and nestable into groups. You can run it once and discard it, or save it and reuse it as a source in **reference fields, views, reports** (web only), and **summaries/dashboards**. No programming is required — it is built visually.

What value types can a query condition use?

A condition's value can be **fixed** (typed manually), **variable/dynamic** (taken from another field, so it compares field-to-field), an **indicator (pointer)** that keeps the query relative to the current user or date (`@me` , `myGroups` , `myRoles` , `myOrgUnit` , `@today` , …), a **code expression** (JavaScript, for logic the other types can't express), or a **parameter** — a named placeholder supplied later when the query is used.

Why does an *Is not empty* condition no longer match an entry whose repeatable block has no rows?

Because **Is empty** (`IsEmpty`) and **Is not empty** (`IsNotEmpty`) no longer test only whether the stored value is null or an empty list — the predicate now also tests **presence**. *Is not empty* matches only when the field exists on the entry *and* holds a real value; *Is empty* matches an absent field just as it matches a null or empty-list value. The data query engine and the reaction query engine compile both operators from the same shared builder, so a saved query and a reaction criterion always agree on what "empty" means.

The case this corrects is a **reference field inside a repeatable block**: with no block instances the reference is stored as an empty set, and traversing that empty set used to make the "not equal to null" test succeed, so the entry wrongly passed an *Is not empty* condition.

Stated limitation — a condition cannot be scoped to a single repeatable-block instance. These operators receive a field path, not a block instance, so "at least one instance of the block has this field filled" is inexpressible, and a block mixing an empty instance with a filled one reads as **empty** as a whole. That is **pre-existing behaviour, not introduced by this change** — use a separate marker field on the entry if you need per-instance semantics. See Queries. *(Since 1.8)*

Why did a condition with no aggregate operator used to fail?

Because an unset aggregate operator is persisted as an empty object — the codebase's convention for every other empty value — and the engine's guard treated that as a value that should name a member of the operator enum, rejecting the criterion as an unsupported expression. Only a non-empty string can now name an operator; anything else falls through to the default **Any** behaviour, where the condition matches when any of the field's values satisfies it. An operator name the engine genuinely does not recognise is still rejected with an error.

This mattered most for **reaction criteria**, which the same engine evaluates: a criterion with no aggregate operator used to be treated as "not matched", so the reaction silently never fired. See Queries. *(Since 1.8)*

Can I filter entries by values on a related entity?

Yes. When you pick a reference/relation field in the query builder you can drill into the related entity's fields and add a condition there — for example filter tasks where *department → region = "West"* without leaving the Task query. Relations can be traversed more than one level deep, but because each level adds a join, keep the chains reasonably shallow.

What is a statistic query, and how is it different from a normal query?

Where a normal query returns a **list** of matching entries, a **statistic query** aggregates entries into a **number, chart, or list** — for example "count of requests by department". It reuses the same filter builder plus an aggregation function (Count, Sum, Min, Max, Average), a group-by field, and a display choice. Statistic queries always exclude archived and obsolete entries, and they are the building blocks of dashboards, summary cards, and counters.

How are entries sorted by default, and can I change it?

By default, an entity's entries are shown sorted **descending by modification date**. You can change this with the sort dropdown (choose a direction and a sort-by field) or by clicking the sort icon in a column header. A view can also fix its own default sorting that is applied when it opens.

Can every query be deleted?

Most can, but two kinds cannot: **implicit** (system-generated) queries, and any query currently in use by a virtual, joiner, or aggregator entity. Deleting an ordinary saved query is permanent.

How does a parameterized quick query get its values?

Each quick query brings its own values. When people apply a parameterized quick query, they get a **Query parameters** dialog to fill in. A parameterized quick query can never be applied by default. See Parameterized quick queries. *(Since 1.10)*

Why does my condition with an Expression value match nothing?

Usually because the expression returns the wrong shape: it is evaluated with no entry in scope and its raw result is compared without any casting by field type. A number field needs `10`, not `'10'`; a date needs a string in the stored format such as `yyyy-MM-dd`, not a `Date` object; a reference needs `{dataEntity, dataEntry}`, such as `scope.me()`, not a bare id. The editor only **warns** about a mismatch and never blocks Apply, so a wrong expression saves and then silently matches nothing. See The Expression value source [Integrate & Extend › Criteria and field-path dialects]. *(Since 1.10)*

Dashboards, summaries, counters, and reports

What tools are available for analysis and statistics?

Statistics are generated from queries and conditions and surfaced through dashboards and summaries, counters, and reports. Note that reports are **web-only** — they are not available in the mobile app.

What is the difference between a dashboard, a summary, and a counter?

They build on the same statistic-query foundation at different sizes:

- A **summary** is a single card built on one statistic query, rendered as a Number, List, or Chart.

- A **dashboard** is an ordered collection of summary cards, optionally arranged into named groups, attached to an entity, a module, or the whole system.
- A **counter** is the single-value form — one statistic query reduced to a live count that surfaces as a top-bar badge, a Number card, or a quick-counter widget on a page.

What chart types can a summary display?

A summary can render as a **Number** (single KPI), a **List** (top values), or a **Chart** — Column, Bar, Line, Pie, or Doughnut. Charts use theme-aware colors and apply a segment limit with an "Other values" bucket for high-cardinality group-bys.

At what levels can I build a dashboard?

At three scopes: **entity level** (an entity's own summaries, which appear automatically in its Dashboard tab once you add them), **module level** (aggregating summaries from the different entities in the module), and **system level** (built through Pages, choosing the module, entity, and dashboard in the page's settings).

Why can't I open a report on the mobile app?

Reports are a **web-only** feature. A report link opens only in the web application; on mobile it resolves to an unsupported link and cannot be navigated to. Preview and use reports on the web.

Can a report be scheduled and emailed automatically?

Yes. A report can be scheduled to be generated and emailed to specific recipients on a recurring or one-time basis, delivered either as a link or as a file attachment (Excel or PDF). Report output respects sensitive-field masking, so a recipient without access to a field sees it masked.

Why does my exported PDF report show only some of the columns?

The **PDF** export has a hard cap of **7 columns** — any fields beyond the seventh are dropped from the PDF. The **XLSX** export has no such cap, so choose Excel when you need all columns, or trim the report's field list for PDF.

Why can I define an aggregate over a sensitive field but not over an encrypted one?

Because there is a real number behind a sensitive mask and none behind ciphertext. An aggregate over a **sensitive** field is computed and then withheld — it still appears under its display name with `****` as the value, since removing it would itself disclose that the field exists and is protected (on a view an anonymous visitor can open it is dropped whole instead). An aggregate over an **encrypted** field is refused when you save the definition, and the aggregate picker hides encrypted fields altogether: there is nothing to mask, only nonsense to compute. Note also which functions you get — **Count, Sum, Min, Max and Average**, all five on a Number field and **Count only** on any other type. See Protected fields. *(Since 1.10)*

Services and workflows

What is the difference between an entity and a service?

A service must be governed by a workflow, whereas an entity can exist without one. Requests submitted through a service appear under "My Requests" for the authenticated user, and the service also has an inbox where requests waiting on you as the step owner appear, each requiring a specific action.

Can I stop a service at any time?

Yes. You can deactivate and reactivate a service whenever you like.

Can I tailor services to specific users?

Yes. Use special permissions and audience targeting to make a service available to a particular set of users.

Can I create views for services?

Yes. You can build views for a service just as you would for any regular entity.

What is service categorization, and how do I customize it?

Categorization lets you sort and organize services by section or purpose — for example, general services versus HR services. Categories are added and edited from the main “Service Categories” list.

Can a service be linked to other services?

Yes. Services can be linked by enabling relationships between them and through reactions.

Where is a workflow stored, and what step types can it have?

A workflow is stored **inside** its entity (an entity owns an array of workflow definitions) — it is not a standalone document. Its steps are an ordered array of types: **Entry** (start, owned by the entry’s creator), **Intermediate** (a review/approval stage), **Exit** (terminal), **Sub-Workflow** (delegates to a nested workflow), and runtime **Ad-hoc** steps. A workflow must begin with an Entry step and end with an Exit step, and every non-Exit step needs at least one action.

How is a workflow step's owner decided?

Each step has an owner configuration with a **type** (user, group, org-unit, or list), a **source** (fixed, dynamic from a field, the previous step’s owner, or an expression), and options such as manual “take ownership”, climbing to the owner’s manager, or assigning to a whole group for parallel/voting decisions. If the primary owner can’t be resolved, ordered **fallback owners** are tried in turn. An Entry step is always owned by the entry’s creator.

What is an SLA on a workflow step?

Each step can carry a **time limit (SLA)** — fixed, read from a field, or computed by an expression — after which the step is considered overdue. The due date is calculated against the configured official working hours and holidays, so the countdown excludes non-working time. An overdue step raises an event you can react to (an alert or an escalation, such as reassigning the owner to their manager).

What is a sub-workflow, and when would I use one?

A **Sub-Workflow step** hands an entry off to a separate, nested workflow defined on the same entity, then resumes the parent flow (at a configured “merge step”) when the nested workflow finishes. Use it to reuse a common process — for example a standard “budget approval” — inside many larger workflows without rebuilding it each time. A sub-workflow step has no actions of its own; the transition happens automatically when the nested flow completes.

How do I hand a workflow step off to an external system?

Add an **External Step** to the workflow. Set its **URL**, its **Method** (GET or POST) and **Open As** — how the external page opens for the step’s owner. A POST step can also carry a **Custom Payload**; by default it sends the record’s

details and the step's actions. Put the return-link token `{{@returnUrl}}` in the URL or the payload so the external system can hand the record back (use `{{@returnUrl:actionId}}` when the step has more than one action). The step's owner must be one specific person, and the step completes only when the external system calls back. See Hand a step to an external system. *(Since 1.8)*

What are the External Step display modes, and are they the same on mobile?

There are **five** `Open As` modes — Full Screen, Iframe, Sidebar, Dialog, and Popup Window — and the web client supports all of them. The mobile client supports only **Full Screen** and **Iframe**; Sidebar, Dialog, and Popup Window fall back to Full Screen there, so a step designed around a side panel or a modal will still run on mobile but will not look the way you designed it. Design for the fallback if mobile matters. See Workflows. *(Since 1.8)*

A Move Workflow Step reaction reported success but the entry stayed on its old step — why?

Because an entry save that had read the entry **before** the move wrote the old step back. That is fixed: a save that may not override workflow state now leaves the stored workflow fields as they are, and a step move starts from the entry's stored workflow state rather than from the triggering event's snapshot. An ad-hoc step still computes from its snapshot, and fails with a conflict rather than reverting. See Workflows. *(Since 1.4)*

Reactions and automation

How do I make the platform react automatically when data changes?

Use a reaction (called a *trigger* in the API). Automation is event-driven and follows an **Event → Condition → Action** model: a reaction binds to one event on an entity, optionally narrows down with conditions, and then runs an ordered list of actions. You define reactions per entity from its **Reactions** tab — no code required.

What events can trigger a reaction?

A reaction can fire on an entry being created, updated, archived, or restored; a workflow step being reached, exited, or an action executed; a timer/schedule; a custom event from a button; a file being generated; another reaction succeeding or failing (for chaining); an announcement; an inbound integration event; or an SLA time being exceeded. More than twenty action types are then available — creating/updating/archiving entries, driving workflows, sending notifications/email/SMS, calling external APIs, running custom functions, and generating files.

Why did my reaction run only once, or skip an expected step?

Because data actions re-emit data events, a reaction could re-trigger itself, so there is **loop protection**. Each action's re-execution count is capped by the reaction's loop threshold, which **defaults to 0 — meaning the action runs once per trace**. If an expected second-pass action was silently skipped, the loop guard is the usual cause; raise the threshold deliberately to allow a bounded number of self-re-executions.

Can one reaction trigger another?

Yes — reactions **chain**. A reaction can react to another reaction succeeding or failing, an action can publish an announcement that other reactions consume, and data-modifying actions emit the ordinary created/updated/archived events that in turn fire other reactions. This is powerful, but it is also the easiest way to create loops — see the loop-protection note above.

Can a reaction call an external system or run my own code?

Yes. The **Call API** action sends a request to another system and captures its response, and the **Execute Function** action runs a function [Integrate & Extend › Functions: run your own code] your developers have deployed. Use them when you need to reach an external system or run custom logic that the built-in actions do not cover.

Can an automation run when a workflow step's owner changes?

Yes — the `WorkflowStepOwnerChanged` trigger event fires whenever the owner of a workflow step on an entry changes. It is published right after the new owner has been persisted, so a bound reaction always sees committed state. The event says *which kind* of change it was — `ChangeOwner`, `TakeOwnership`, or `Reassign` — and the builder exposes two optional filters: an **Owner Change Type** multiselect (leave it empty to fire on any kind) and a **Choose Step** selector matched on the workflow and step id as a pair, so a step id reused across two workflows cannot cross-fire. Use the change-type filter to react to a genuine hand-off without also firing when someone simply picks up an unowned step. See Reactions and automation. *(Since 1.8)*

What replaced the "UI Event" toggle on a custom action trigger?

The **Custom Action** trigger now has a **Custom action level** instead of the old *UI Event* toggle:

- **UI action** — run from a user interface, such as a button. This is what *UI Event* switched on used to mean.
- **External action** — can be run by another reaction's **Execute Custom Action**, from any module.
- **Local action** — can be run by **Execute Custom Action** only inside its own module.

It also has a **Contextual** toggle (on by default), which is a marker only — it does not change how the reaction runs today. The **Show to user** options (the execution indicator and its start and success messages) apply only to UI actions. Existing custom actions were all converted to **UI action** with Contextual on, and the change is one-way, so review any that should be External or Local. See Custom action type and Contextual. *(Since 1.8)*

What is the "Execute Custom Action" trigger action, and when would I use Self vs Reference scope?

Execute Custom Action lets one reaction run a **Module** custom action on another entry — the automation equivalent of a user pressing a button, except a trigger drives it rather than a click. You pick the custom action from a searchable dropdown of the module's Module-type custom actions (UI and Internal ones are not offered), then choose a scope:

- **Self** — run it on the **current entry**. Offered only when the custom action's entity is the same as the trigger's entity; saving a Self-scope action across entities is rejected.
- **Reference** — follow a **reference field** on the current entry to the target entry in the custom action's entity. A reference-field selector appears so you can choose which field to follow; saving without a reference target is rejected.

When it runs, the action resolves the target entries and raises the same custom-action event a system button raises, so the target custom action runs **once per target entry, each in its own isolated context** — it sees only that entry's data, not the calling reaction's context. Note that resolving **no** target entries is not an error: the action succeeds and dispatches nothing, so check the dispatched count in the execution history rather than looking for a failure. See Execute Custom Action. *(Since 1.8)*

Can a reaction run against an entry from a different entity?

No. A reaction always runs as its own entity. A record that belongs to a different entity is skipped rather than processed with the wrong fields, and when a reference field accepts several entities, each reaction receives only the records of its own entity. A skipped record raises no error and does not appear in the reaction's history, so if a reaction seems not to fire for some records, ask your operator to look for skip warnings in the platform logs. See Entity isolation. *(Since 1.8)*

Can a reaction grant or remove access permissions?

Yes, since 1.10, with the **Set Permission** and **Remove Permission** actions, on the system, a module, an entity, a view, a page or a report, for users or groups. *Restricted* and *None* can never be granted, and Set never lowers a higher grant that already exists. A reaction cannot grant more than the person who saves it: each permission action is checked against your own access when you save. See Set / Remove Permission actions. *(Since 1.10)*

Can a later action use what my custom function returned?

Yes, since 1.10. **Execute Function** exposes `success`, `message` and `result` as outputs, so later actions in the same reaction — and reactions chained onto its success or failure — can bind them, including a key below `result` such as `result.customer.id`. Keep the function's return shape stable, because those bindings are paths into it. See Execute Function output in chained reactions. *(Since 1.10)*

Why is a reaction with a Create or Update Data Entity action refused, or the action skipped?

Because those two action types never had a handler and were formally retired in 1.10. The builder no longer offers them, creating or updating a reaction that carries either is refused with **422**, and one already stored in an older reaction is skipped when the reaction fires — recorded as a failed action, while the actions after it still run. Remove the action from the reaction. See Action types. *(Since 1.10)*

Why was I refused when saving a reaction?

Creating, changing or deleting a reaction needs **Developer** access (or higher) on its entity. Ask for Developer access on the entity or its module, or ask someone who has it to save the reaction. See Reactions & Automation. *(Since 1.8)*

Can a reaction send a WhatsApp message?

Yes, with the **Send WhatsApp** action, which sends one of your space's **approved WhatsApp message templates**. It works once your space's WhatsApp business account has been set up — in 1.10 that is done in the system settings, not on a dedicated screen, so ask your administrator — and each message's delivery status is recorded in the WhatsApp message log. See Reactions & Automation. For integrators: WhatsApp [Integrate & Extend › Calling the HTTP API]. *(Since 1.10)*

Why can't my Call API action reach an internal server?

For security, **Call API** refuses private and internal addresses — including a public name that points to one, and any redirect to one. Only the operator of your deployment can allow a particular internal destination, so ask them. Every call is also limited to HTTP or HTTPS, 30 seconds for the whole exchange, and a 10 MB response. See Call API: what the platform allows. *(Since 1.8)*

Why does my chained reaction run twice?

Because its space scope is wider than the **Triggering Space**. A `ReactionSucceeded` / `ReactionFailed` reaction runs on the stored context and entry of the run it watches, and those belong to the space that run ran in. On 1.10 the scope is expanded as for any other reaction, so the chained reaction runs once per resolved space, each time with the original space's entry: *Primary and Triggering Scopes* fires it twice for a run in a non-root space, and *All Scopes* once per space. Give it the Triggering Space only. On the development line the platform runs it once, in the watched run's space, whatever scope it stored, and refuses to save one with any other scope. See Chaining reactions. *(1.10 behaviour; enforcement Since dev)*

Buttons

What can a custom button do?

A button is an action button rendered against an entity's records. When clicked it can **fire an action/reaction** (a custom-event reaction), **open a pre-filled create/update form**, **redirect** the user somewhere, or **run a custom script**. Its scope can be a single entry, multiple selected entries, or a data collection.

Can a button open a form that's already filled in?

Yes. Each field the button seeds is filled by a value mapping: **Dynamic** (pull from the current entry), **Manual** (a fixed value you type), **Not Set** (leave empty), **Pointer** (a context token such as Me, My Groups, My Roles, My Org Unit), or **Expression** (a JavaScript expression, e.g. `now + 3 days`). So a "Create follow-up task" button can open the new-task form already populated from the record you clicked on.

Can I show a button only on certain records?

Yes. A button's availability can be restricted to entries that satisfy a saved query or an inline filter (merged with AND when both are set). When selected entries don't match, the button is either **disabled** (default) or **hidden**, depending on its non-match setting.

Why did a button disappear when I saved a view?

The **View**, **Edit**, **Archive**, **Create** and **Duplicate** buttons are generated from the view's own settings rather than stored. If you switch an operation off — editing, say — the matching button is removed when you save, instead of the save failing. Buttons you created yourself and template buttons are never removed this way. See Data views. *(Since 1.8)*

Templates and file exports

What document formats can templates produce?

Templates let you export entity data as richly formatted documents. Supported outputs are **DOCX, HTML, and XLSX**, with conversion from DOCX/HTML to **PDF**. Values are filled through tokens and expressions resolved against the entry at export time. Sensitive fields the user cannot access appear masked in the output.

Can I use logic or JavaScript inside a template?

Yes. HTML templates are rendered with EJS (`{{% … %}}` for logic, `{{%= … %}}` for output), and Word/Excel templates use their own token/loop syntax. JavaScript and a set of custom helper functions (date formatting,

resolving references, workflow-step lookups, and more) can be used inside templates, and expressions are evaluated per entry at export time.

Where are templates used?

In three places: a **view's export** (to export a view's entries), the **"Generate File From Template"** reaction action, and **reports**, which render through the same engine. Reports are web-only.

Can an export template be offered only for some entries?

Yes, since 1.10, for a **Single Entry** template: its **Filter** section sets a saved query or criteria the entry must meet, and **For other entries** chooses whether the export option is disabled (the default) or hidden on entries that do not. A Multiple Entries template cannot carry a condition, and a template without one behaves as before. See Conditional export. *(Since 1.10)*

Timeline and history

Does every record keep a history of who changed what?

Yes. Every entry accumulates a history feed assembled from field changes, workflow actions, owner reassignments, SLA breaches, automation runs, button clicks, and create/archive events — each stamped with who did it and when (including "acting on behalf of" for delegated actions). A view can control how much of this history end users see, and sensitive-field values stay masked in history for anyone without the grant. Recording is asynchronous, so expect brief eventual consistency after a write.

Why does the entry history now show only the languages that changed?

Because a multilingual field's history entry is rendered per language — `label: old → new` — for each language that actually changed. Previously both sides of the diff were resolved to whichever language the UI was set to before being displayed, so editing only the Arabic value and then reading the history with the interface in English collapsed both sides to the same unchanged English text and the change looked empty. The stored record always carried the full per-language before/after values; only the rendering was wrong.

What the card shows now: only the languages that actually changed (a language whose value is identical on both sides is skipped, and blank counts as the same as missing); languages in the space's configured language order, with anything outside that list appended; the language's **label** in the current UI language while each **value** is shown in its own language, untranslated, so the same edit reads identically whatever language you view it in; and a language added during the edit rendered as a clean addition (new value only, no arrow) with a cleared language rendered as a removal, following each language's own before/after state rather than the field's overall change type. See Timeline & history. *(Since 1.8)*

What is the timeline, and how do posts get created?

The timeline is a feed of **posts**. You don't create posts by hand — you enable "timeline contribution" on an entity and map its fields, and the platform then generates and syncs posts automatically from that entity's entries. Posts can be pinned and published/unpublished (manually, on a schedule, or by a workflow) and carry a social layer of reactions and comments.

Is a "reaction" on a timeline post the same as an automation reaction?

No — they are entirely different. A timeline **reaction** is a *social* emoji reaction on a post (one per user), whereas a **reaction** in the automation sense is an event-driven trigger with conditions and actions. The two share a name but are unrelated.

Permissions, sharing, and sensitive data

Can data be protected and access permissions controlled?

Yes. Data is protected through the platform's security and permission settings. You can also mask the values of specific fields from unauthorized viewers using the Sensitive Fields feature.

What are the access-permission levels?

Permissions are an ordered ladder — **None** → **Restricted** → **View** → **Developer** → **Edit** → **Admin** — applied at several scopes (system, module, entity, view, page, or report). The highest grant wins, except an explicit **None**, which is a hard deny. Permissions are usually granted to **groups**, and members inherit them. Alongside the ladder, **access permissions** (authenticated / anonymous / restricted) apply to views, reports, and services.

What are sensitive fields, and can an administrator bypass the masking?

Sensitive fields mask a field's value (for example a salary or national ID) for everyone who has not been granted explicit access to that specific field — on live entries, in history, in reports, and in exports. It is a **separate dimension** from ordinary permissions, and it is **not** bypassed by an administrator or Edit access: even a full admin sees the mask unless they hold the sensitive-field grant. Reference and block-container fields cannot be marked sensitive.

What is a space?

A **space** is the platform's multi-tenancy isolation boundary: it scopes entities, modules, groups, views, and the users who may access them. Every stored record is stamped with its space, and reads and writes are automatically filtered by it, so a user cannot reach a space they don't belong to.

What are data stores, and how do they decide who sees an entry?

A data store is the partition an entity's entries are written into, categorized by sharing scope: **Public** (all spaces), **Private** (one space), and **Shared** (a chosen set of spaces). Every entity gets a Public and a Private store automatically; you can add a Shared store spanning at least two spaces. A new entry is written to the single highest-**weight** store available to the current space, while reads aggregate across all stores that space can see — which is what scopes an entry's visibility across spaces.

What does a sensitive field return to a caller who lacks access?

The plain display mask `****`. No internal marker crosses the API boundary — the mask is substituted in place on read, descending into block sub-fields inside repeatable blocks, and a value is revealed only when the caller both holds a grant for the scope *and* has an active unmask session covering it. On the write side, protection is enforced **by permission**, not by expecting a round-trippable sentinel back: whatever a caller submits for a sensitive field they cannot read is discarded and the stored value restored (on create the field is dropped entirely), regardless of the literal value submitted. So a submitted mask can never overwrite the stored value. See Sensitive fields. *(Since 1.8)*

Why can't I grant a permission above my own level?

Unless you are a system administrator, you cannot grant anyone a higher level on an item than your own effective level on it, and you cannot change or remove a grant that is already above your level. Your effective level counts your system grants, your own and your groups' grants on the item, and what you inherit from its module or entity; under a delegation, the delegation's permissions count. The same ceiling applies to reactions that set permissions. See A grant cannot exceed your own level [Administration › Permissions & access]. *(Since 1.8)*

Does the OpenFGA authorization check change who can access my data?

No. It is a **shadow** check: 1.10 can keep a second copy of each tenant's access rights in OpenFGA and compare, for sampled requests, what OSOS decided with what OpenFGA says. It never changes a decision, never blocks a request and never writes to your data, and it compares system, module and entity decisions only — not sensitive-field grants. It is **off** unless the deployment includes OpenFGA and the tenant is opted in, so there is nothing to design around. See Authorization shadow check [Administration › Permissions & access]. *(Since 1.10)*

Pages and interfaces

What tools are available for customizing page designs?

Pages are composed from many components, including the Title, Text, Media, Statistics, and View components, among others. For the full set of page elements, see the Pages guide.

Can I add buttons and links to navigate between pages or sections?

Yes. Buttons and links can be added for navigation. See the pages section of the Pages guide.

Can pages or interfaces be protected from unauthorized access?

Yes, by setting access permissions on them.

Can I create multiple kinds of pages, such as a homepage and product or service pages?

Yes. You can create many pages and choose the content of each one individually.

Can elements have titles, and can those titles be shown or hidden?

Yes. You can add a title to an element and control whether it is shown or hidden.

Can elements be shown or hidden per user?

Yes. The administrator can show or hide elements for targeted user segments through the custom-display settings in the element's settings.

What is the maximum number of pages or interfaces I can create on OSOS?

There is no limit on the number of pages you can create.

Can a page be laid out in columns and rows?

Yes. Use the paragraph and column elements to divide a page into columns and rows.

Connecting to other systems

Do I need to write code to build an application on the platform?

No. OSOS is designed to build information systems through an interactive visual interface — modeling data, wiring services and workflows, and composing pages — without writing complex code. Programming expressions and deployed functions are an **optional** layer for logic the interface doesn't expose; they are never required to build an application.

Can platform data be integrated with external applications?

Yes. Data can be exchanged with external applications through webhooks and integrations [Integrate & Extend › Webhooks & integrations] and outbound API calls from reactions, with access secured by API tokens [Integrate & Extend › Use an API token].

Can data be imported from other sources?

Yes. Data can be brought in from other sources by setting up an integration.

When should I use a Function instead of a reaction?

Use a no-code reaction to respond to data or workflow events; use the built-in sandboxed expression engine for computed fields, dynamic values, and template tokens; and reach for a deployed Function [Integrate & Extend › Functions: run your own code] when you need custom code, external network calls, or heavier computation. A reaction can call a function when it needs that extra logic.

Can I send data to external tools when something happens?

Yes. Outbound webhooks are configured as reaction actions rather than a separate subscription screen: the **Call API** action is the general-purpose outbound HTTP call, and **Notify Zapier** pushes events to subscribed Zapier hooks. See Webhooks & Integrations [Integrate & Extend › Webhooks & integrations] for the full picture, including inbound flows and the parts that are deliberately incomplete.

Preview and entity versions

What does an entity's "version" number mean, and what counts as a breaking change?

An entity carries a numeric **version** that is *not* a per-save counter — it is bumped only when a change would break existing data **and** entries already exist at the current version. Breaking changes include adding a required field, making a field required, changing a field's type, toggling repeatable/multi-value, and adding/removing/reordering workflow steps; additive changes like a new optional field are safe. When a breaking change bumps a published entity's version, existing entries are shown as **obsolete** until you upgrade them, and you can **preview breaking changes** before saving.

What is preview mode, and can I trust its permissions?

Preview mode renders a view, page, or entry against deterministic **sample data** so you can design layout without real records. It is **not** a permission bypass: the request is still authenticated, and the same authorization voters that guard the real endpoints still run — access levels, restricted views, and sensitive-field rules all stay in force. There is exactly one relaxation, and it is narrow: sample data is addressed by fabricated ids that deliberately do not exist, so in preview a **genuine not-found** on that lookup is allowed through rather than surfacing as a permission error. Any

other failure still fails. Even so, preview is a rendering aid, not an inspection tool — do not use it to judge what a restricted end user will actually see. Verify real permissions with a normal (non-preview) view once published.

Can I preview a page as it will look on a phone?

Yes. The page builder's **Content** tab hosts a **live mobile preview**: the actual mobile client, embedded in the builder, rendering the page you are editing right now — including unsaved changes. Every edit is pushed to it (debounced, and re-applied in place rather than remounted, so your scroll position and local form state survive), so the phone follows the canvas without a save or a reload. Because it is the real client, you get the real mobile rendering — same layout engine, same widget implementations, same theme. Three deliberate differences: the app bar is drawn but its navigation is inert, there is no help button, and redirection rules never fire. Parsing is tolerant per component, so a malformed widget is dropped rather than blanking the page, and an unparseable push leaves the last good configuration on screen. The view builder has an equivalent mobile-preview tab beside the web preview. Remember that some surfaces are web-only — reports, for instance — so preview those on the web. See Preview mode. *(Since 1.8)*

Moving, packaging and publishing apps

How do I move an application between environments (dev → staging → production)?

Package it. Export the app as a file from **Administration › Packages › Build** and import it in the other environment with **Install from file** — or, for an app you will keep improving, create a **package** with numbered versions that the other environment installs and later updates. The platform takes along what your selection depends on and matches each item to what already exists in the target. Check the **External dependencies** before you export. See Export, import and install apps.

What is left out when I export a package?

Archived entities (with everything under them), anything personal to one user, and anything the platform generates for itself are left out, even if you select them. Users, groups, organization units and working hours are not copied: in the target you match each one to an existing record, and the platform remembers your choices for the next import from the same source. Items your app only optionally points to may be cleared in the target, so check the warnings before you export. See What travels, and what does not.

Can the same user or group be mapped differently for different resources in one import?

Yes. When the same user, group or organization unit is referenced by more than one resource in an import — the owner of a workflow step and the user a view is restricted to, say — you can choose a different target under each resource. The first target you choose becomes the default for that person or group, and choosing differently under another resource overrides it there only. **Auto Create** always applies everywhere that person or group appears. See Import. *(Since 1.8)*

Can I publish a template to the marketplace and install apps from an in-app store?

Yes, since 1.10, once your operator has turned the marketplace on. Group your modules into a **package**, create a **version**, fill in its store page and **Send for review**: a reviewer in your organization approves it, then the marketplace team lists it. Other environments install it from **Packages › Store** and update it in place when you publish a newer version. Sending needs membership of **Marketplace publishers**, approving needs **Marketplace reviewers** — even for a system administrator — and your plan must include publishing. See Publish a version to the store. *(Since 1.10)*

Why can't I edit or export a resource that was installed from a package?

An installed app is locked so that updates apply cleanly — for everyone, system administrators included. You can still add your own fields, hide what you do not use, and change the name, description and icon of its views and pages; their publishing state, display order, audience and access rights stay yours. To change anything else, **Fork** the app: its parts become yours, but it stops receiving updates for good and can never be published or exported. See Work with an installed app. *(Since 1.10)*

Who may export a package, and who may import one?

Exporting needs **Developer** access to the modules or entities you export (or a system-wide Developer grant). Importing needs a **system administrator**. See Before you start. *(Since 1.10)*

Do I need a marketplace connection to browse the in-app store?

No. You can browse the store without connecting. Only installing needs a connection: the first time you select **Install**, the **Connect to the marketplace** dialog shows what connecting shares, and **Connect and install** connects and installs. Connecting publishes nothing. The store still needs the marketplace turned on by your operator, the root space, and the right to install. See Install an app from the store. *(Since 1.10)*

We are connected to the marketplace — why is sending a version for review refused?

Because connecting does not make you a publisher. Someone must first **Set up publishing** on the **Build** page — choosing the publisher name and, optionally, a store address — and that needs a system administrator or a marketplace publisher. Sending is also refused when your plan does not include publishing, or when another publisher owns the app in the marketplace. See Set up publishing. *(Since 1.10)*

Why is ***Send for review*** disabled, with a "publishing revoked" warning?

The marketplace has stopped your publisher from publishing, without ending the connection. The package page shows the date and the marketplace's reason. New versions and store-page changes cannot be sent until the marketplace restores publishing; installing and updating apps still work. Only the marketplace team can lift it. Available from release 1.10.2. *(Since 1.10)*

I am a system administrator — why can't I send a package for review or approve one?

Publishing and reviewing come from group membership only. Add yourself to **Marketplace publishers** to send, or **Marketplace reviewers** to approve, in **Administration › Organization › Users Groups**; it takes effect straight away. The marketplace terms are accepted by the reviewer when approving, not when sending. *(Since 1.10)*

How do I move an installed app to a newer version?

Update it in place — no uninstall and reinstall. When a newer version is listed, select **Update** on its installation page or its store page, read the preview of what changes, and confirm. You need to be a system administrator in the root space and a member of **Marketplace installers**. A forked app cannot be updated. See Update an installed app. *(Since 1.10)*

An install failed — how do I recover without leaving half an app behind?

What a failed install created stays locked, so nothing is left unprotected. Its installation page shows how to recover: retry from its import session while that still exists, or upload the same package file again. When a compatible newer version is available, **Update** finishes the installation instead, without an uninstall. You can also uninstall a failed

installation. Before installing, the store checks whether a version would install now and lists any names that collide. See A failed install. *(Since 1.10)*

How do I tell a publisher that their app failed to install?

On the installation page, select **Report to *publisher***. You see exactly what will be sent — the package and version, the error and the resources that failed — and you can add a note and choose whether to share your name and email. You need to be a system administrator in the root space, with a marketplace connection. If the error changed since you opened the report, you are asked to review it again. Available from release 1.10.1. *(Since 1.10)*

Can I uninstall an installation I have forked?

Yes. The uninstall dialog warns that your own changes to the forked parts are deleted with them and — as for any app — lists what is deleted, the records people created, and the links from outside the app. **Uninstall** stays disabled until those links are removed. A module that still holds entities your users created is kept. Uninstalling needs a system administrator in the root space. See The installation page. *(Since 1.10)*

How do I find failed or uninstalled installations?

Open **Administration › Packages › Installed** and use the **Status** filter (*Installed*, *Failed*, *Uninstalled*) and the **Source** filter beside the search. Both start empty, so everything shows until you pick one. Each row shows who installed the app and, for a store install, its publisher. *(Since 1.10)*

Why does an import stop at *Choose a field* for a mapped field?

A mapped field copies a value from a parent entity, and the import could not find the source field on the matching entity in this environment by name. Choose the field it should copy from — only fields of the same type and compatible settings are offered — or **Don’t copy**. The import will not run until every such field is decided. See Import. *(Since 1.10)*

Can a package hold pages that belong to no module?

Yes. A page that belongs to no module or entity (a global page) can be part of a package, alone or with modules — so a package may hold only pages. A page that later moves into a module or entity cannot be published, and neither can one that was installed or forked from another package. A summary widget whose data is outside the package installs unconfigured instead of failing. See Global pages in packages. *(Since 1.10)*

Do a function's secrets travel in a package?

No. A packaged function carries its secret names only; the values stay in the environment they were set in. Set them in each environment after installing. *(Since dev)*