Internship/Job Application Tracking System Project

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

- **Core Needs Identified:**
    1. Manage candidate records (personal details, resume, skills).
    2. Manage job postings (title, department, description, location, number of openings).
    3. Track applications (status updates such as Submitted, Shortlisted, Rejected, Hired).
    4. Record interview details (date, interviewer, mode, result).
    5. Automate communication (confirmation emails, interview schedule, result notification).
    6. Approval workflow for HR manager to finalize hiring decisions.
    7. Generate dashboards & reports for recruitment insights.

  Expected Outcome: A clear list of business requirements before any development begins.

Stakeholder Analysis

- **HR Manager** – Final decision-maker for hiring, needs approval workflows and dashboards.
- **Recruiter** – Manages job postings, shortlists candidates, schedules interviews.
- **Interviewer** – Conducts interviews, records feedback, updates results.
- **Candidate (External User)** – Applies for jobs and receives communication.
- **System Administrator** – Manages configurations, security, and maintenance of the system.

  Expected Outcome: A stakeholder responsibility matrix to ensure all user needs are captured before development.

Business Process Mapping

1. Candidate applies for a job → Recruiter receives application (currently via email/spreadsheets).
2. Recruiter reviews application manually → Decides if candidate should move forward.
3. Interview scheduled manually → Candidate informed via phone/email.
4. Interview results stored separately (spreadsheet, manual notes).
5. HR Manager makes final decision → Not formally tracked, approval often verbal.
6. Candidate informed manually → No proper history maintained.

Future State with Salesforce:

- Each step becomes a record-driven process in Salesforce (Application → Interview → Approval).
- Communication automated with email alerts & flows.
- Decisions stored in the system with audit trail.

  Expected Outcome: A clear workflow showing how the system will work once implemented.

Industry-specific Use Case Analysis

- **Industry:** Human Resources / Recruitment / Talent Management.
- **Current Challenges in Industry:**
    - Recruitment is time-consuming due to manual tracking.
    - Lack of structured candidate data storage.
    - Difficulty in generating insights (e.g., how many candidates applied or were hired).
    - Communication gaps between HR, interviewers, and candidates.
- **Salesforce as a Solution:**
    - Custom objects to store structured recruitment data.
    - Automation to reduce manual work.
    - Dashboards for HR insights.
    - Approval processes to standardize hiring decisions.

  Expected Outcome: Clear justification that Salesforce is suitable to solve the problem.

AppExchange Exploration

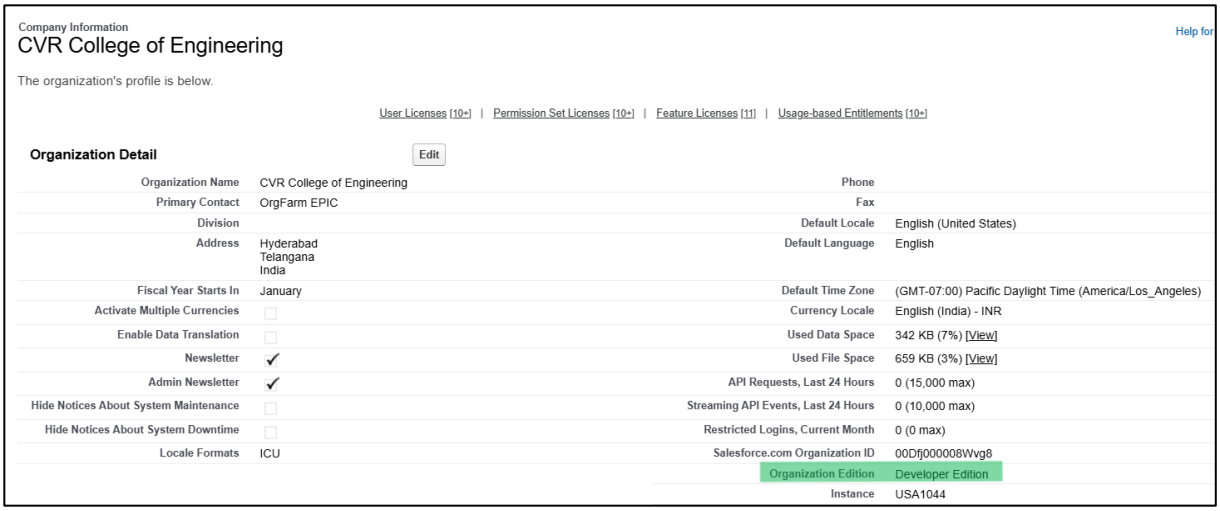
- **Gap Identified:** These existing tools are either too expensive or too complex for a simple internship/job tracking use case in an academic or small company scenario.
- **Decision:** Build a custom, lightweight solution on Salesforce with only essential features:
    - Candidate, Job, Application, Interview objects.
    - Validation rules, Flows, Triggers, Approvals.
    - Email templates for communication.
    - Dashboards for HR.

  Expected Outcome: Justification for creating a custom Salesforce solution rather than using AppExchange products.

Phase 2: Org Setup & Configuration

1. Salesforce Edition

Salesforce Developer Edition is a free Salesforce org with full features but limited storage and users, designed for learning, experimentation, and app development, not for production use.



Company Information

CVR College of Engineering

Help for

The organization's profile is below.

User Licenses [10+] | Permission Set Licenses [10+] | Feature Licenses [11] | Usage-based Entitlements [10+]

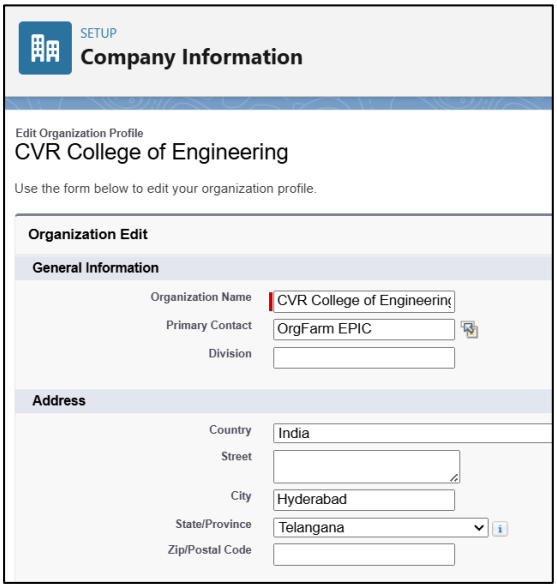
Organization Detail Edit

Organization Name	CVR College of Engineering	Phone	
Primary Contact	OrgFarm EPIC	Fax	
Division		Default Locale	English (United States)
Address	Hyderabad Telangana India	Default Language	English
Fiscal Year Starts In	January	Default Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)
Activate Multiple Currencies	☐	Currency Locale	English (India) - INR
Enable Data Translation	☐	Used Data Space	342 KB (7%) [View]
Newsletter	✓	Used File Space	659 KB (3%) [View]
Admin Newsletter	✓	API Requests, Last 24 Hours	0 (15,000 max)
Hide Notices About System Maintenance	☐	Streaming API Events, Last 24 Hours	0 (10,000 max)
Hide Notices About System Downtime	☐	Restricted Logins, Current Month	0 (0 max)
Locale Formats	ICU	Salesforce.com Organization ID	00Dfj000008Wvg8
		Organization Edition	Developer Edition
		Instance	USA1044

2. Setup company info:

Setup → Company Settings → Company Information → fill Company details.

- Company Information provides key details about your org:
  - such as the organization name, default locale, language, currency, fiscal year, and time zone.
- It also shows edition, licenses, usage limits, and org ID, helping admins manage identity, settings, and capacity of the org.



SETUP

Company Information

Edit Organization Profile

CVR College of Engineering

Use the form below to edit your organization profile.

Organization Edit

General Information

Organization Name	CVR College of Engineering
Primary Contact	OrgFarm EPIC
Division	

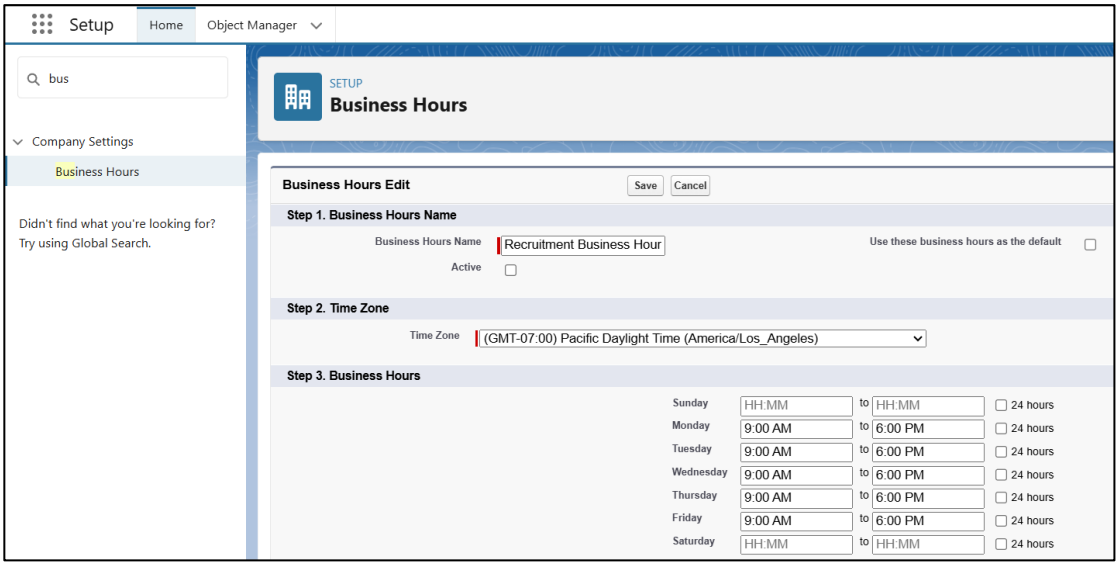
Address

Country	India
Street	
City	Hyderabad
State/Province	Telangana
Zip/Postal Code	

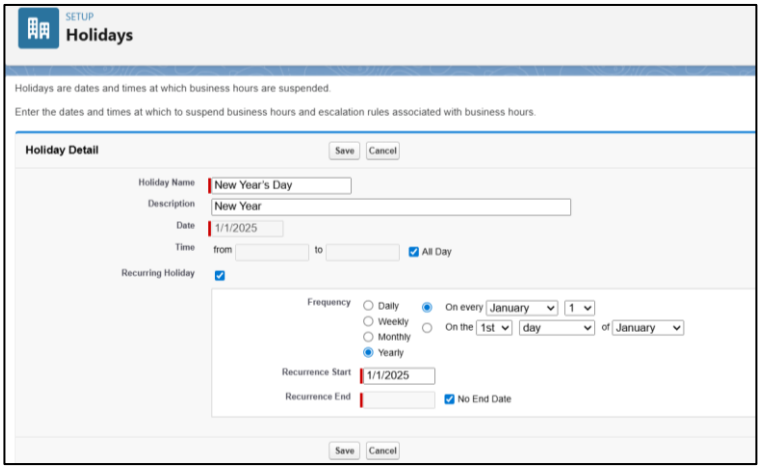
3. Business Hours

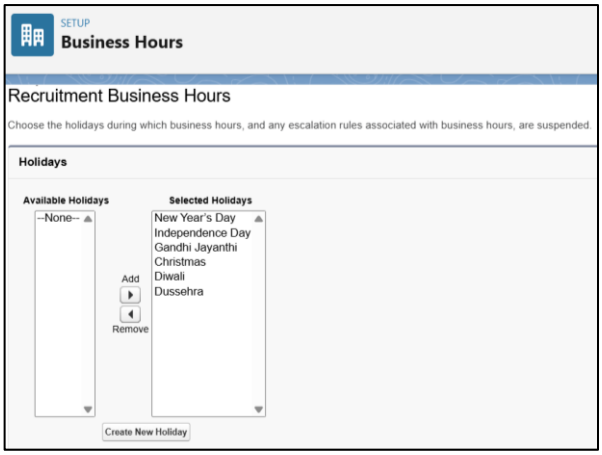
Business Hours define the working days and working hours of your company

1. Go to Setup → Business Hours → Create business hours (useful for scheduling interviews).
2. Click New Business Hours.
3. Enter details:
    - Name: Recruitment Business Hours
    - Time Zone: (GMT-07:00) Pacific Daylight Time (America/Los_Angeles)
    - Start Time: 09:00 AM
    - End Time: 06:00 PM
    - Days Active: Monday–Friday
    - Saturday & Sunday: Unchecked (non-business days)
4. Save.



4. Business Holidays





2. Click **New Holiday**.
3. Add Holiday dates:
    - Name: New Year's Day, Date: 01-Jan-2025
    - Name: Republic Day, Date: 26-Jan-2025
    - Name: Independence Day, Date: 15-Aug-2025
    - Name: Gandhi Jayanthi, Date: 02-Oct-2025
    - Name: Christmas, Date: 25-Dec-2025
4. Save each holiday.

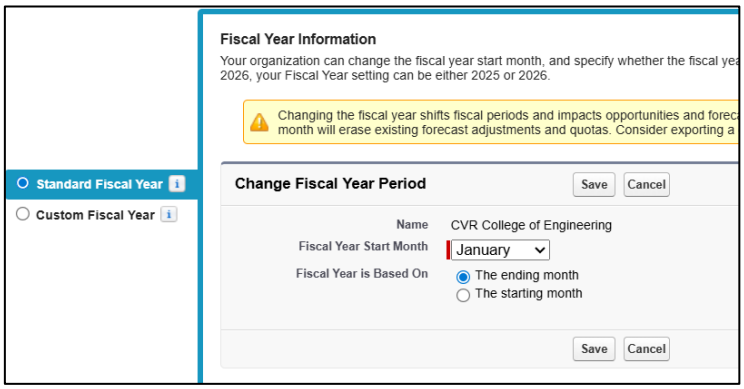
👉 These settings ensure interview scheduling avoids weekends/holidays.

6. Fiscal Year Setup

Fiscal year defines how your company tracks financial reporting periods (months, quarters, years). It impacts reports, forecasts, and dashboards.

Navigate to Fiscal Year Setup

1. Go to Setup → Company Information → Fiscal Year.
2. Click Edit.



Choose Fiscal Year Type

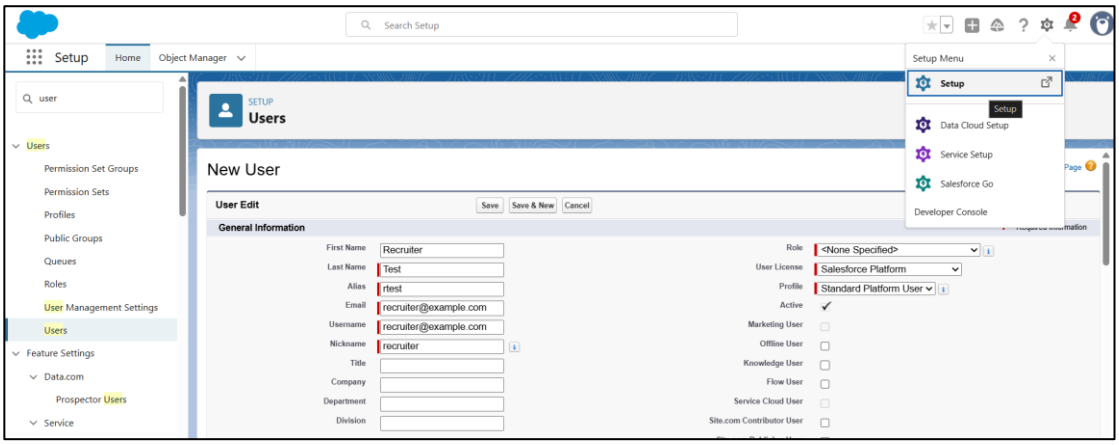
- Standard Fiscal Year: Aligns with calendar months.
- Choose start month (e.g., January).
- Set Start Month & Save

7. User Setup & Licenses

Users are individual people who log in to Salesforce. Each user needs a license which defines what Salesforce features they can access.

- Create Users (Setup → Users → Users → New User) – create three test users with different profiles:

- *recruiter@example.com* (Profile: Standard Platform User or custom Recruiter)
- *hrmanager@example.com* (Profile: Standard Platform User or custom HR Manager)
- *interviewer@example.com* (Profile: Standard Platform User or custom Interviewer)

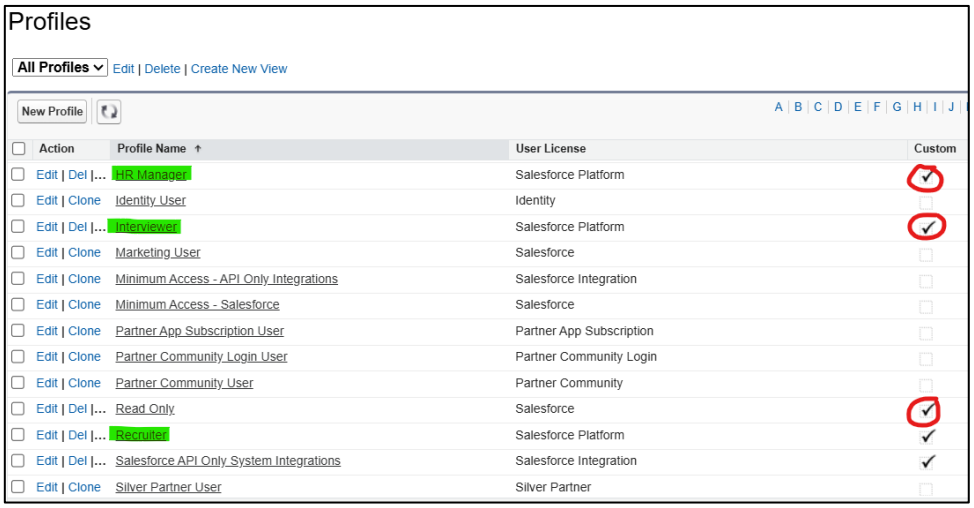


8. Profiles

- Profiles define what a user can do in Salesforce.
- Profiles define baseline permissions (object access, field access, tab visibility, app access).

Steps:

1. Setup → Profiles.
2. Clone a standard profile (e.g., Standard Platform User).
3. Rename (e.g., Recruiter Profile, HR Manager Profile).
4. Configure object permissions, tab visibility, system permissions.
5. Save.

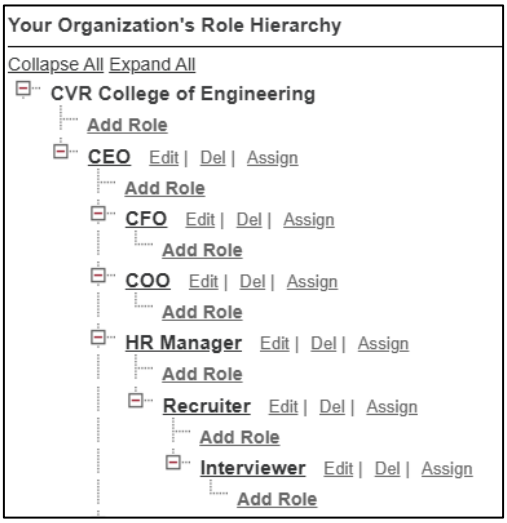


9. Roles

- Roles define where a user sits in the role hierarchy, which controls record-level visibility. Higher roles can see records owned by lower roles.
- Roles control record-level access via hierarchy.

Steps:

1. Setup → Roles → Set Up Roles.
2. Define hierarchy: CEO → HR Manager → Recruiter → Interviewer.
3. Assign users to roles.

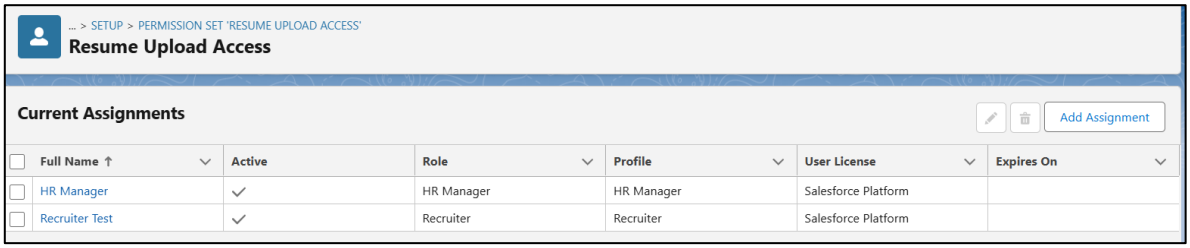


10. Permission Sets

- Permission Sets extend permissions beyond the profile without needing to create multiple profiles.
- Extend permissions beyond profiles without creating new profiles.

Steps:

1. Setup → Permission Sets → New.
2. Label: Resume Upload Access.
3. Configure object/system permissions.
4. Assign to Recruiter and HR users.



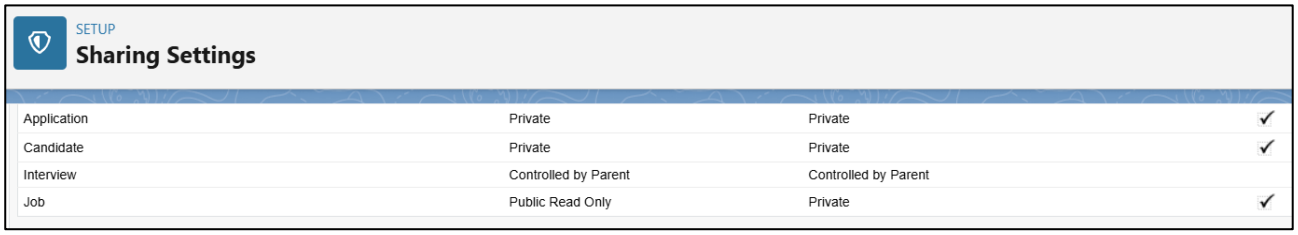
11.Organization-Wide Defaults (OWD)

OWD sets the baseline level of access for all records of an object across the org.

- Candidate, Application, Interview → Private (each user only sees their own).
- Job → Public Read Only (so all can see open jobs, but only owners edit).

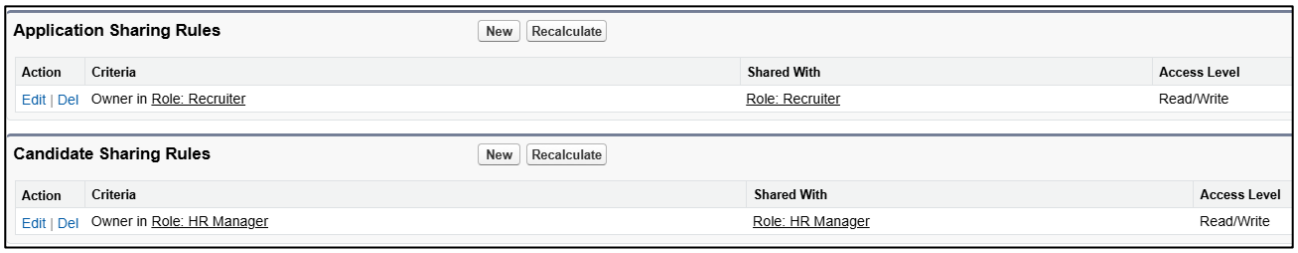
Steps:

1. Setup → Sharing Settings.
2. Candidate = Private.
3. Application = Private.
4. Job = Public Read Only.
5. Interview = Private.
6. Save.



12.Sharing Rules

- Sharing Rules extend access beyond OWD to specific users, roles, or groups.
- Use them to give Recruiters and HR Managers access to Candidate and Application records while keeping OWD restrictive.
  - Create a Public Group or use Roles (Recruiter, HR Manager) to target who should see records.
  - Create sharing rules on the Candidate and Application objects to grant appropriate access (Read Only or Read/Write).

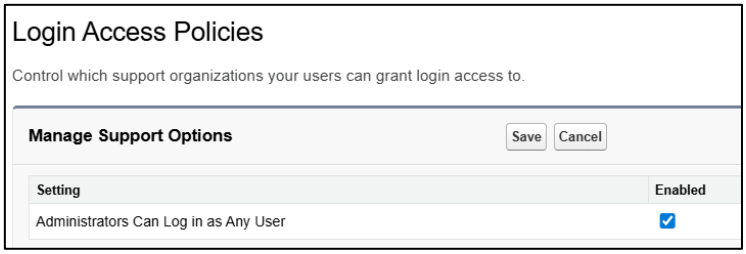


13. Login Access Policies

Controls whether Salesforce support and administrators can log in as other users for troubleshooting.

Steps:

1. Setup → Login Access Policies.
2. Enable "Administrators Can Log in as Any User."
3. Save.



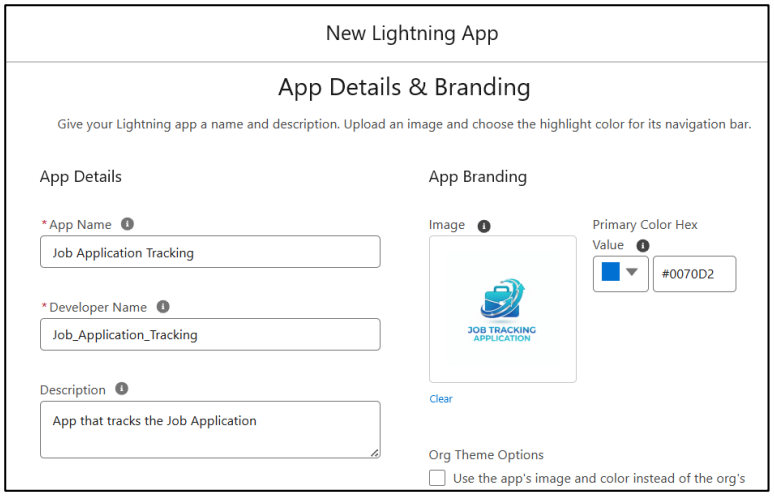
For testing/demo: enable this so you (as admin) can log in as Recruiter or Interviewer to verify role-based access, OWD, and sharing rules are working properly.

14. Dev Org Setup

A Developer Org is a free Salesforce environment provided by Salesforce for learning, testing, and building apps. It comes with preloaded features and a limited amount of storage.

Steps to Set Up:

- Go to developer.salesforce.com.
- Click Sign Up → Fill details (name, email, username).
- Activate account from the verification email.
- Login to your Dev Org → you'll have access to Salesforce Lightning Platform.



Phase 3: Data Modeling & Relationships

4.1 Custom Objects and Fields:

Setup → Object Manager → Create → Custom Object

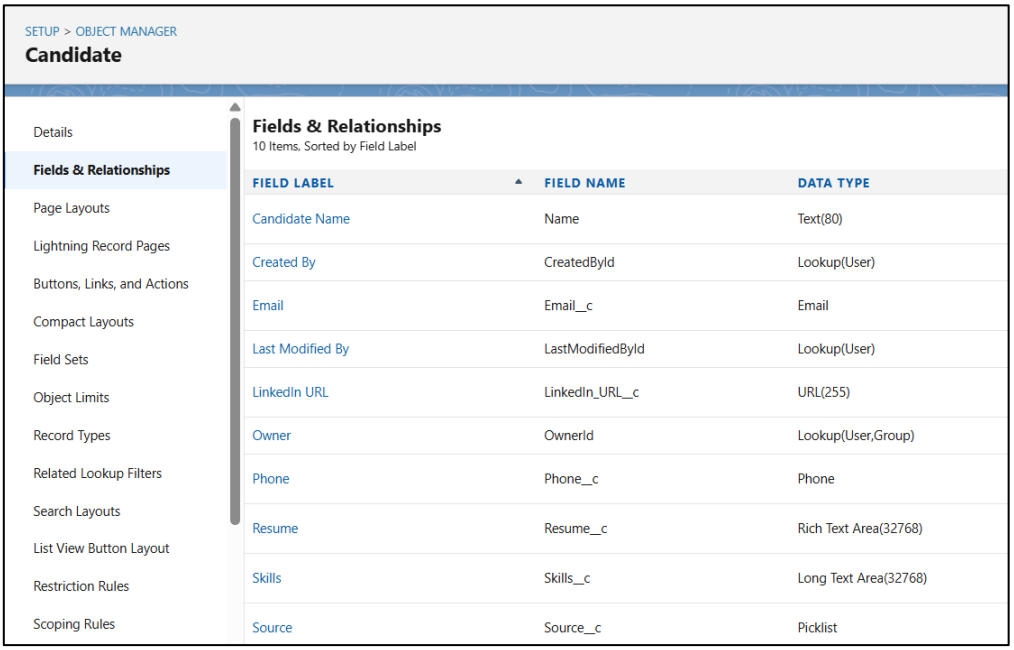
Candidate

- Label: Candidate
- Plural Label: Candidates
- Record Name: Candidate Name (Text)
- Allow Reports: ✅, Allow Activities: ✅, Track Field History: ✅

Save

Fields (Fields & Relationships → New):

1. Email (Data Type: Email) — API: Email__c — Required: yes
2. Phone (Phone) — Phone__c
3. Skills (Long Text Area) — Skills__c
4. Resume (File) — use "Files" via Related List.
5. Source (Picklist) — values: LinkedIn, Campus, Referral, Other — Source__c

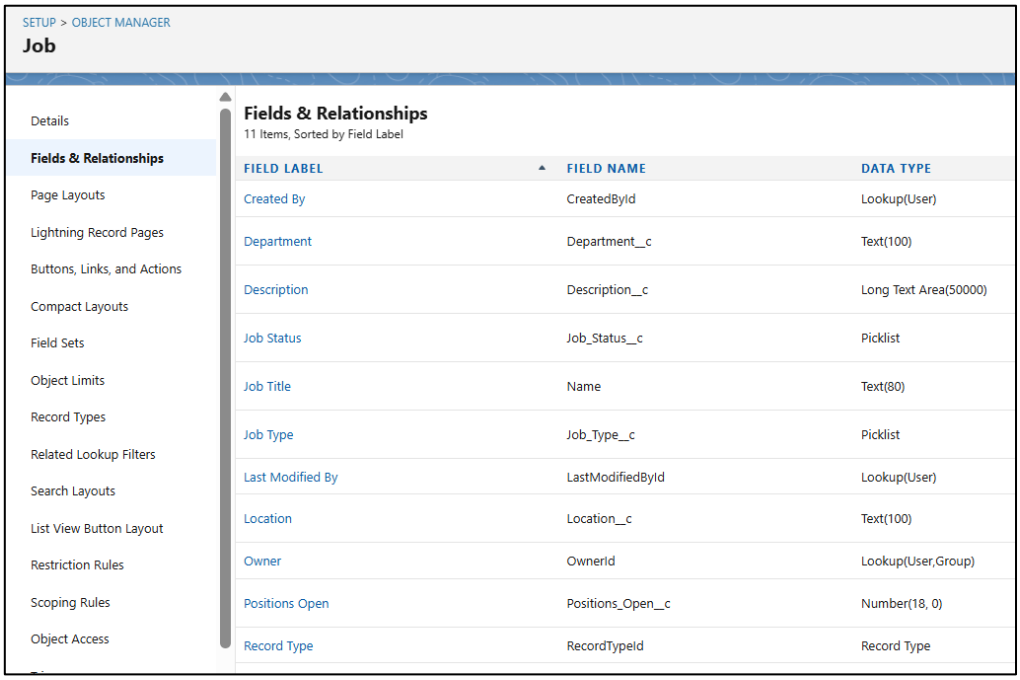


Job

- Label: Job
- Plural: Jobs
- Record Name: Job Title (Text)
- Save

Fields:

1. Department (Text)
2. Location (Text)
3. Positions_Open (Number) — Positions_Open__c
4. Job_Type (Picklist)— Job_Type__c
5. Status (Picklist) — values: Open, Closed — Status__c
6. Description (Long Text Area) — Description__c



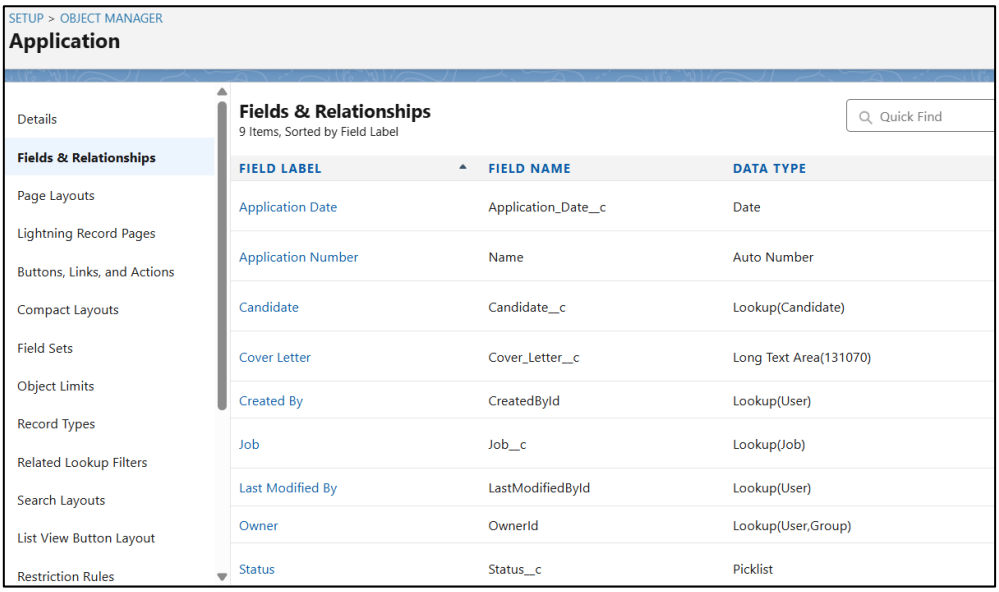
Application

- Label: Application
- Plural: Applications
- Record Name: Application Number (Auto Number) — Format: APP- {0000} — good for tracking.

Fields:

1. Candidate (Lookup → Candidate) — Candidate__c — Required
2. Job (Lookup → Job) — Job__c — Required
3. Application_Date (Date) — Application_Date__c — default = Today () if you want
4. Status (Picklist) — Submitted, Under Review, Shortlisted, Interview Scheduled, Rejected, hired — Status__c (default: Submitted)
5. Score (Number) — Score__c (optional)
6. Cover_Letter (Long Text Area) — Cover_Letter__c

➢ create lookup relationships Candidate__c and Job__c now.

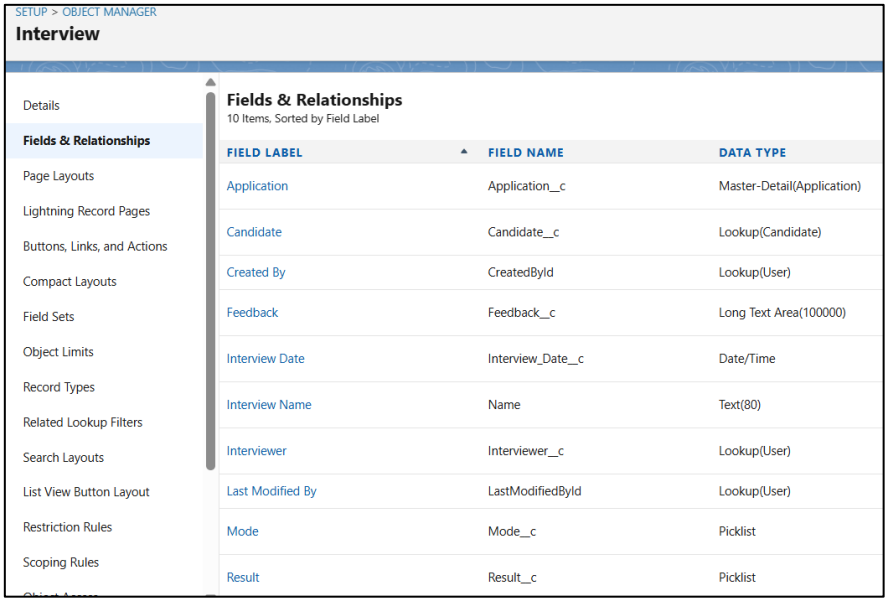


Interview

- Label: Interview
- Plural: Interviews
- Record Name: Interview (Auto Number or Text)

Fields:

1. Application (Master-Detail → Application) — Application__c — Required (master-detail recommended so interview belongs to application)
2. Interview_Date (DateTime) — Interview_Date__c — Required
3. Interviewer (Lookup → User) — Interviewer__c — or Lookup to Contact if external
4. Mode (Picklist) — values: Online, Offline — Mode__c
5. Result (Picklist) — Pending, Selected, Rejected — Result__c
6. Feedback (Long Text Area) — Feedback__c



4.2 Record Types

Use Record Types when you need different business processes/layouts/picklist values for an object (e.g., Job: Internship vs Full-Time).

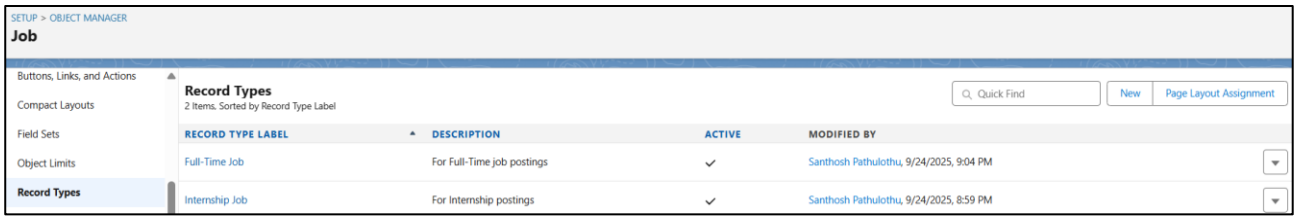
Steps to create Record Types for Job:

1. Setup → Object Manager → Job → Record Types → New.
2. Enter Record Type Label: Internship Job
    - Record Type Name: Internship_Job
    - Description: For Internship postings
    - Make Active: check
    - Select page layout: choose existing or Default
3. Choose which Profiles get access to this Record Type (select Recruiter, HR Manager)
4. Click Save.

Create a second Record Type: Full-Time Job — follow same steps. Then customize picklist values per record type if needed (e.g., Job_Type default).

Use case: Recruiter sees fields relevant to internships; HR sees fields for full-time.

- Job record types: Internship / Full-Time



4.3 Page Layouts

Design the UI users see when viewing/creating records.

Steps — Create Candidate Page Layout:

1. Setup → Object Manager → Candidate → Page Layouts → Click New.
2. Use the layout editor:
    - Drag fields (Email, Phone, Skills) into the layout sections.
    - Add Related Lists: Applications, Interviews, Files (Resumes).
    - Add Quick Actions (e.g., New Application).
3. Click **Save**.

Assign Layouts to Record Types/Profile:

1. In Page Layouts list click Page Layout Assignment → Edit Assignment.
2. Choose which Page Layout each Profile uses for each Record Type.

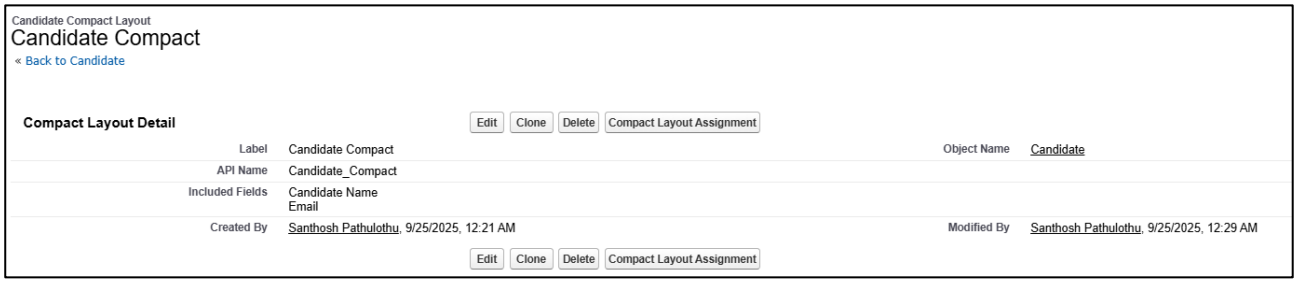
4.4 Compact Layouts

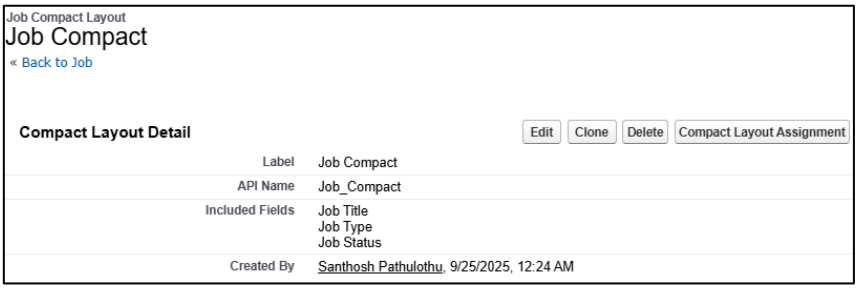
Show the most important fields in the record header (highlights panel) so users see key info at a glance on Candidate / Job / Application records.

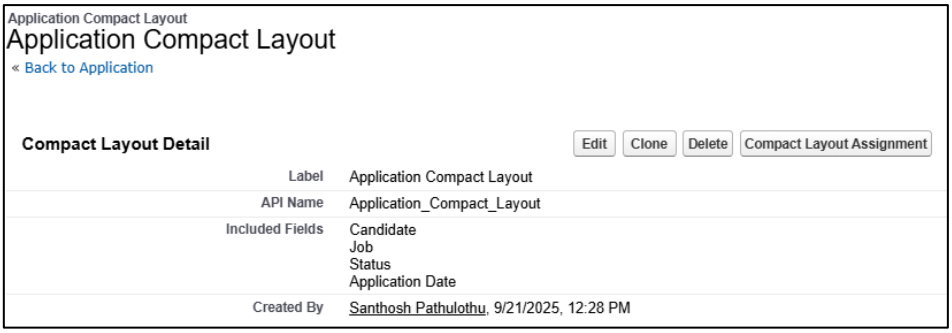
Steps:

1. Setup → Object Manager → Candidate → Compact Layouts → New.
    - Label: Candidate — Compact
    - Fields to add (order matters): Candidate Name, Email__c, Phone__c.
    - Click Save.
2. After save click Compact Layout Assignment → Edit Assignment → set Candidate — Compact as the Primary Compact Layout → Save.
3. Repeat for Job (fields: Job_Title__c, Status__c) and

    Application (fields: Application_Number__c, Status__c).





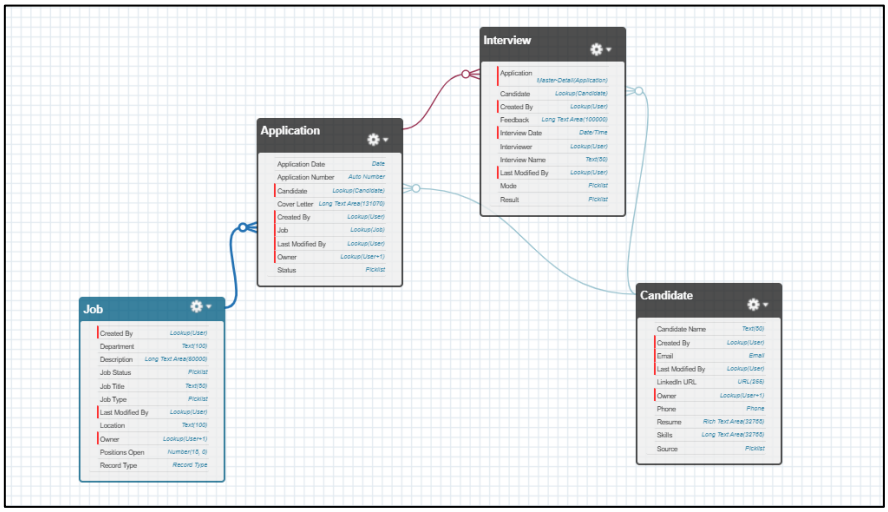


4.5 Schema Builder (visual)

- Use Schema Builder to visually create objects, fields, and relationships for Candidate / Job / Application.
- Schema Builder is excellent for visualizing relationships and creating multiple fields quickly.

Steps:

1. Setup → Quick Find → Schema Builder → Open.
2. In the right-hand panel, select only custom objects or use the search box to locate Candidate, Job, Application.
    - Schema Builder canvas showing Job, Candidate, Application and relationship lines



4.6 Lookup vs Master-Detail Relationships

Lookup Relationship: loosely links two objects. Child record can exist without parent. No ownership transfer; no automatic cascade delete; no roll-up summary on parent.

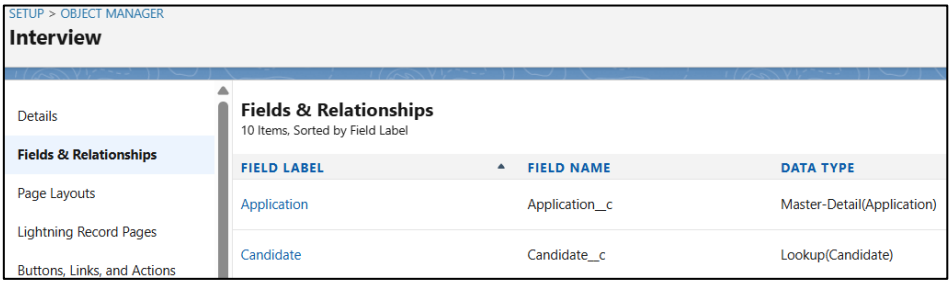
Master-Detail Relationship: tight link. Child record inherits sharing & owner from parent; if parent deleted, child is deleted (cascade); parents can have Roll-Up Summary fields counting/summing child records.

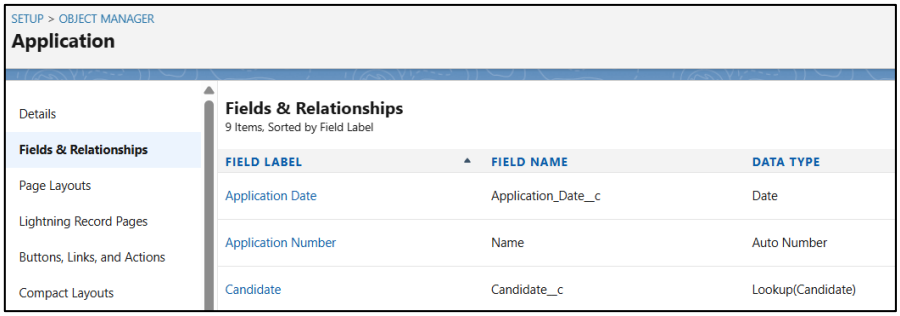
- Ownership: Master-Detail → parent owns child. Lookup → child has own owner.
- Cascade Delete: Master-Detail → yes. Lookup → no (unless "Delete related records" option used).
- Roll-Up Summary: Master-Detail → yes (on parent). Lookup → no.
- Required: Master-Detail → child record requires parent (cannot be null). Lookup → optional.

Relationships

Application = Junction (Master-Detail) between Candidate and Job

Interview → Candidate = Lookup - If an Interview must always belong to a Candidate.



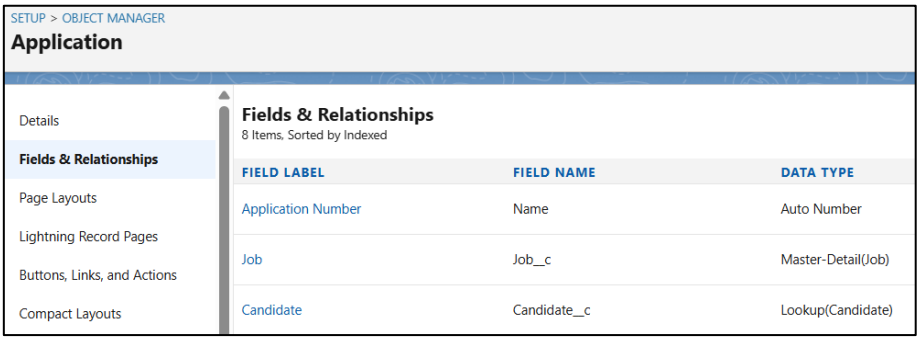


4.7 Junction Objects

Implement many-to-many between Candidate and Job using Application as junction.

Create two Master-Detail fields on Application:

- Fields & Relationships → New → Master-Detail Relationship → Related To: Job → Label Job → Save.
- Again, create Master-Detail Relationship → Related To: Candidate → Label Candidate → Save.



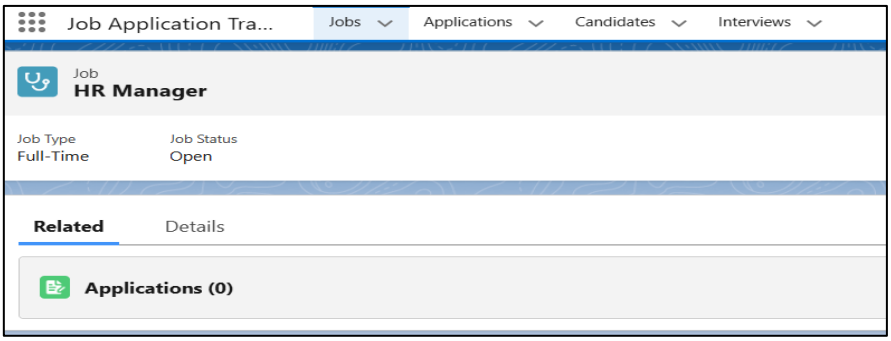
Add Applications related list to Job and Candidate page layouts:

- Object Manager → Job → Page Layouts → edit layout → Related Lists → drag Applications → Save.
- Repeat for Candidate

1. Job Object

Related List: Applications

This related list displays all applications submitted for a particular job. Each record shows which candidate has applied, the application status, and key details. It demonstrates the many-to-many relationship between Jobs and Candidates via the Application junction object.



2. Candidate Object

Related List: Applications

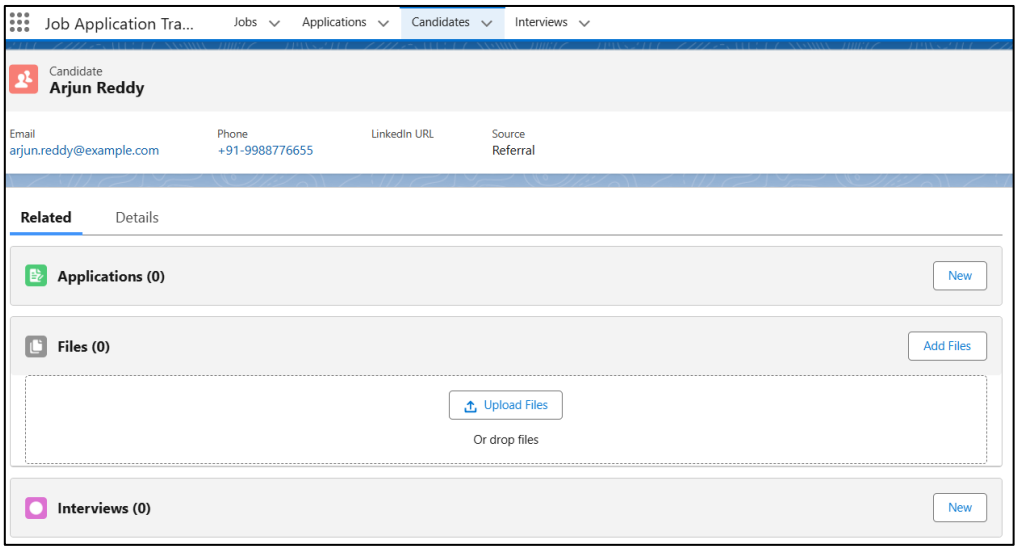
This related list shows all jobs a candidate has applied for. It helps track the candidate's application history and status across multiple job openings.

Related List: Files

This related list stores documents uploaded for the candidate, such as resumes, cover letters, and certifications. It ensures all important files are centralized for each candidate.

Related List: Interviews

This related list records scheduled or completed interviews for the candidate. It provides an overview of the recruitment process and helps track candidate progress through the hiring pipeline.



Phase 4: Process Automation (Admin)

1. Validation Rules

When to use: Block invalid data entry (simple business rules).

- Prevent scheduling an Interview before Application date.

Steps

1. Setup → Object Manager → select the object (e.g., Interview).
2. Left menu → Validation Rules → New.
3. Fill:
   - Rule Name: Interview_Date_Check
   - Description: Prevent interview before application date
   - Error Condition Formula:

Interview_Date__c < DATETIMEVALUE (Application__r.Application_Date__c)

   - Error Message: Interview date must be on or after the application date.
   - Error Location: Field → select Interview Date.
4. Click Save.
5. Test: Try to create an Interview with an earlier date — you must get the error.

2. Workflow Rules

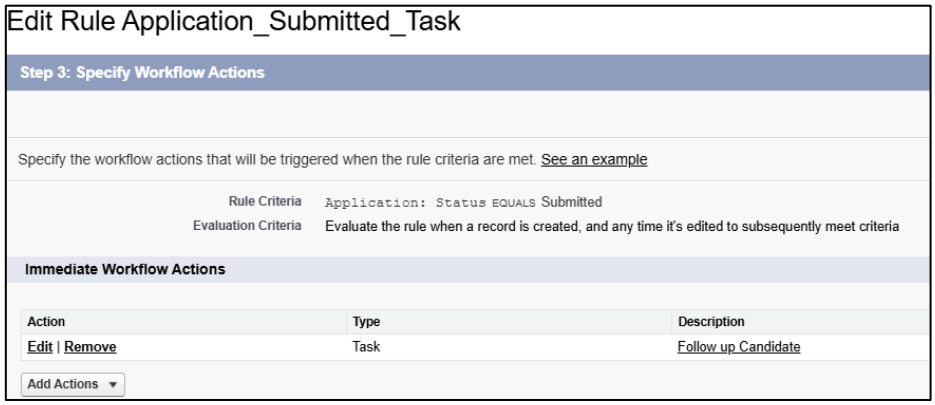
When to use: Simple email/field-updates/tasks that must run on a record create/edit.

- When Application.Status = Submitted, create a Task for Recruiter and send confirmation email.

Steps

1. Setup → in Quick Find type Workflow Rules → click Workflow Rules.
2. Click New Rule.
3. Choose Object → Application → Next.
4. Rule Name: Application_Submitted_Task
5. Evaluation Criteria: created, and every time it's edited to meet the criteria
6. Rule Criteria: Status EQUALS Submitted (use picklist selector) → Save & Next.

7. Add Workflow Actions:
    - New Task → set Subject = Follow up Candidate, Assigned To = Owner, Due Date = TODAY () +2, Priority = Normal → Save.
8. Click Done → Activate the workflow rule.



3. Approval Process

When to use: Formal approvals (HR approves hiring).

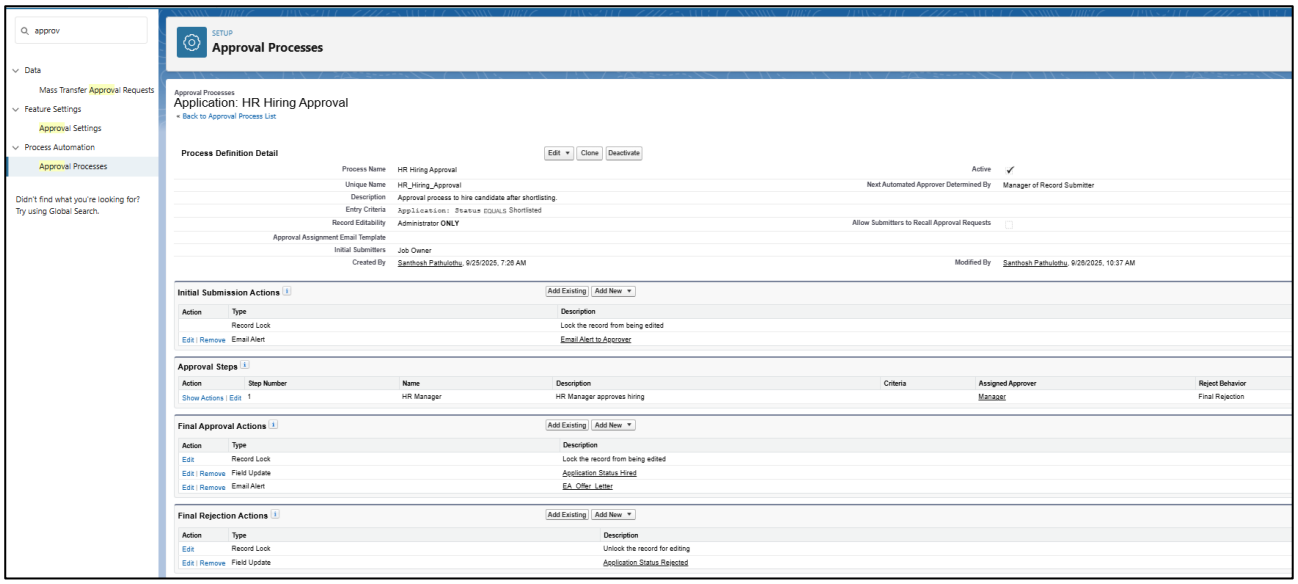
- Approve Application to move to Hired.

Steps (Wizard)

1. Setup → Approval Processes → Create New Approval Process → Use Standard Setup Wizard.
2. Choose Object = Application.
3. Name = HR_Hiring_Approval. Next.
4. Entry Criteria = Status EQUALS Shortlisted (or add Interview.Result = Selected). Next.
5. Specify Approver (e.g., Manually choose approver or use a field like HR_Manager__c).
6. Initial Submission Actions: (optional) Lock Record; Email Alert to approver.
7. Final Approval Actions:

    Field Update set Application.Status = Hired; Email Alert (Offer).
8. Final Rejection Actions:

    Field Update set Application.Status = Rejected; Email Alert (Rejection).
9. Save & Activate.



4. Flow Builder

Record-Triggered Flow

Step 1: Create a New Flow

Select Record-Triggered Flow → click Create.

Step 2: Define the Trigger

1. Object: Choose Application.
2. Trigger: When a record is Created.
3. Condition Requirements → All Conditions Are Met (AND).
    - Field: Status
    - Operator: Equals
    - Value: Submitted
4. Optimize the Flow for: Actions and Related Records.
5. Click Done.

Step 3: Add Email Alert (Confirmation to Candidate)

1. In the Flow canvas, click + → choose Action.
2. Search Email Alert.
3. If you've already created the Application Confirmation Email Alert (with template), select it.
    - Name it: Send_Application_Confirmation.
    - This will send to Candidate Email automatically.
4. Click Done.

Step 4: Add Task (For Recruiter)

1. Click + again → choose Create Records.

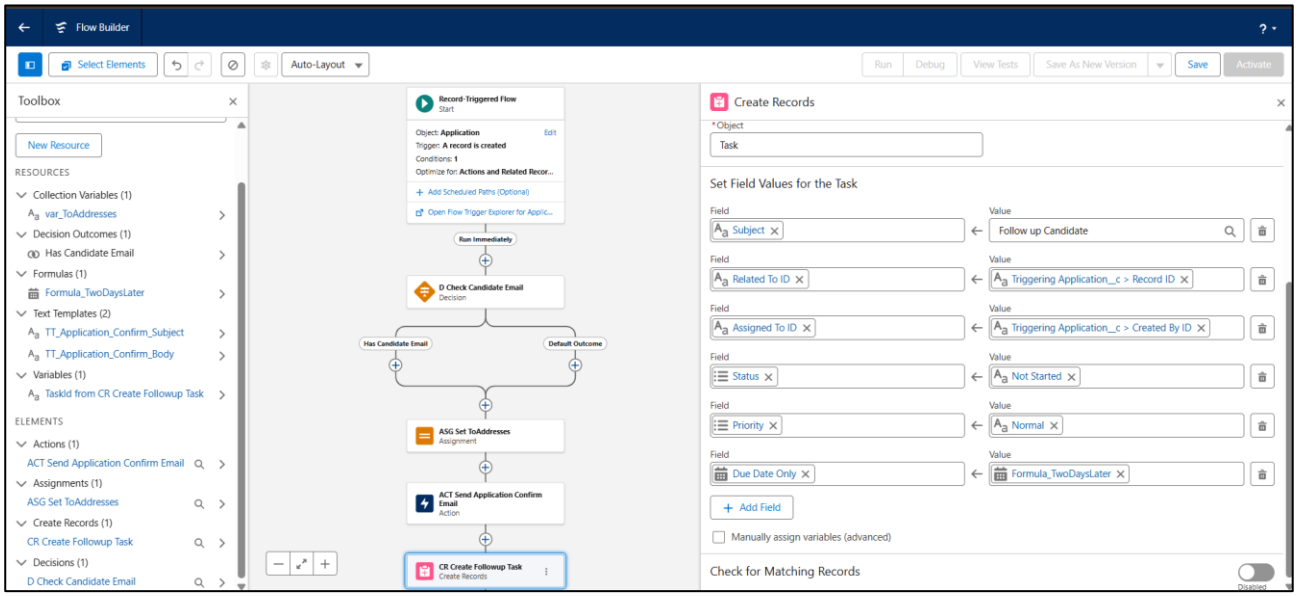
2. Label: Create Recruiter Task.
3. How Many Records: One.
4. How to Set Fields: Use separate resources and literal values.
5. Object: Task.
6. Set field values:
    - Subject: "Follow up with Candidate"
    - Assigned To: Application.OwnerId (Recruiter who owns the Application).
    - Due Date: TODAY () + 2
    - Priority: Normal. Click Done

Step 5: Save & Activate

1. Click Save → give it a Name:

   FL Application OnCreate SendConfirm CreateTask .

2. Click Activate.



Build the Screen Flow

1) Open Flow Builder

1. Setup → Quick Find → Flows → Flows.
2. Click New Flow.
3. Choose Screen Flow → Click Create.

2) Create *recordId* input variable

1. In Flow Builder, open the Manager tab (left side).
2. Click New Resource.
    - Resource Type: Variable
    - API Name: recordId

- Data Type: Text
- Check Available for input → Done (Save resource).

This is required so Salesforce passes the current Interview record Id to the Flow when launched from the record page.

3) Add the Screen (interviewer form)

1. Click the + below Start → choose Screen.
2. Label the screen: Interviewer Feedback.
3. Add components (drag from left palette):

a. Radio Buttons (Result)

- Component: Radio Buttons
- Label: Result
- API Name: result
- Click New Choice → add choices:
  - Label Pending, Value Pending → Save choice
  - Label Selected, Value Selected → Save choice
  - Label Rejected, Value Rejected → Save choice
- Under Store Output Value In, click New Resource → create a Variable:
  - Resource Type: Variable
  - API Name: varResult
  - Data Type: Text
  - Leave Available for input unchecked → Create.
- (Optional) Check Required.

b. Long Text Area (Feedback)

- Component: Text Area (or Long Text Area)
- Label: Feedback
- API Name: feedback

Click Done to save the Screen.

4) Get the Interview record (so we can update it and use fields)

1. Click + after the Screen → Get Records.
2. Label: Get_Interview_Record.
3. Object: Interview.
4. Filter: Id | Operator Equals | Value {!recordId} .

5. How Many Records to Store: Only the first record.
6. Store: All fields (so you can reference Interview.Interviewer__c etc.).
7. Click Done.

Alternative (shortcut): you could skip Get Records and use Update Records with condition Id = {!recordId}. But Get Records makes it easy to reference interviewer Id.

5) Update the Interview record with screen inputs

1. Click + → Update Records.
2. Label: Update_Interview_With_Feedback.
3. Choose Use the Interview record that was found earlier (select the Get_Interview_Record resource).
4. Under Set Field Values for the Interview Record set:
    - Result__c = {!varResult}
    - Feedback__c = {!varFeedback}
    - Next_Steps__c = {!varNextSteps}
5. Click Done.

6) Create a Task for follow-up

1. Click + → Create Records.
2. Label: Create_Interview_Task.
3. How Many Records to Create: One.
4. Record Choice: Use separate resources, and literal values.
5. Object: Task.
6. Set Field Values (example mapping):
    - Subject = Interview follow-up
    - OwnerId = {!Get_Interview_Record.Interviewer__c} ← assigns to the Interviewer user (if Interviewer__c is a User lookup)
    - WhatId = {!Get_Interview_Record.Id} ← links task to the Interview record
    - ActivityDate = {!varNextSteps} (date/time value; if null, do not set)
    - Status = Not Started
    - Priority = Normal
7. Click Done.

Note: If your Interviewer field is not a User lookup, set OwnerId to another appropriate user or the record owner: {!Get_Interview_Record.OwnerId}.

7) Add a Success Screen

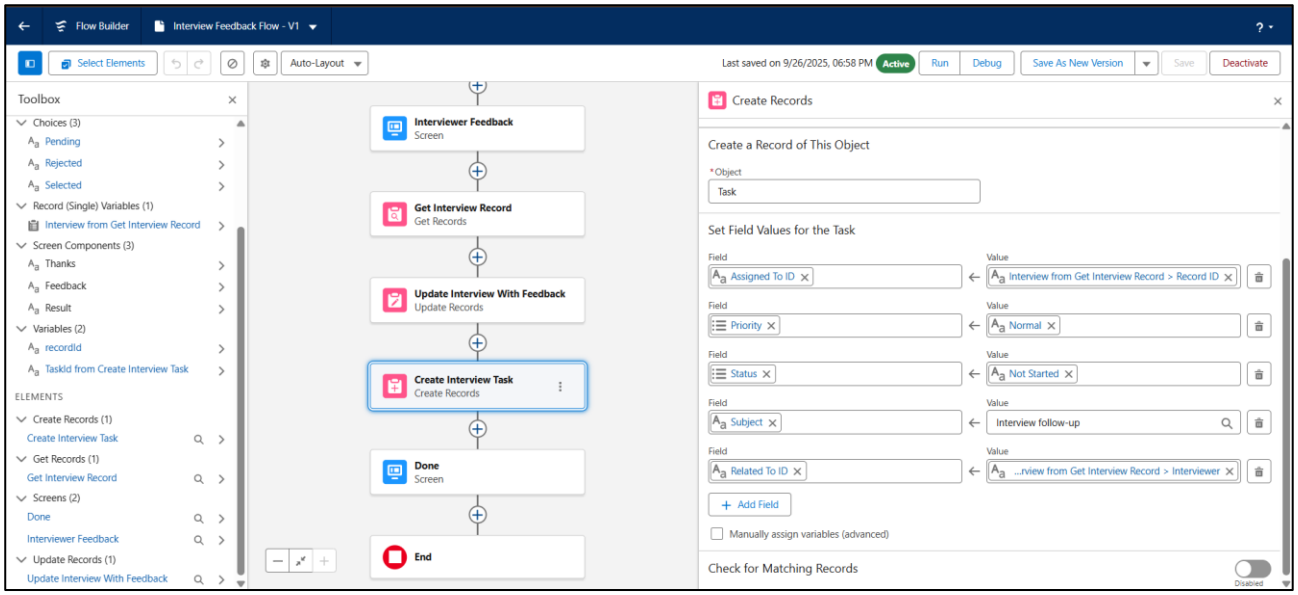
1. Click + → Screen.
2. Label: Done
3. Add a Display Text component: “Thanks — feedback saved and task created.”
4. Click Done.

8) Connect the elements

- Ensure flow path is: Start → Interviewer Feedback (Screen) → Get_Interview_Record → Update_Interview_With_Feedback → Create_Interview_Task → Done (or End).

9) Save & Activate

1. Click Save.
    - Flow Label: **Interview_Feedback_Flow**
    - API Name auto fills.
    - Description: Screen flow for interviewer to submit feedback; updates Interview and creates Task.
2. Click Activate.



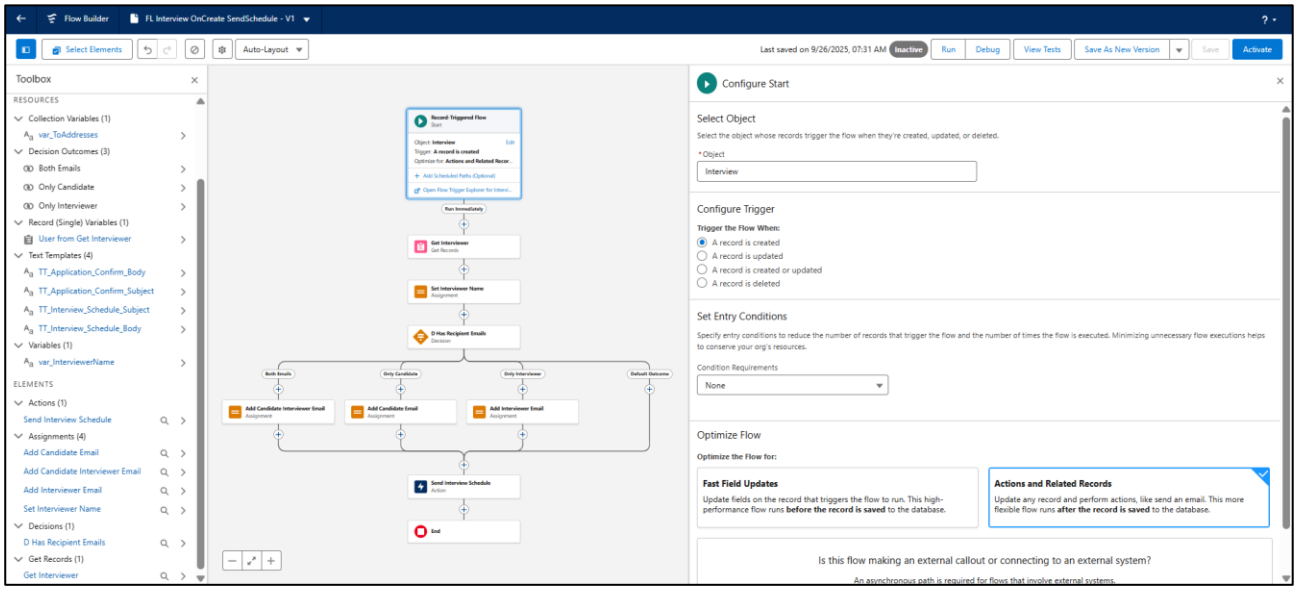
Build Record-Triggered Flow: Application Created → Send Confirmation Email + Create Task

Use Record-Triggered Flow (After Save).

Steps:

1. Setup → Flow → New Flow → Record-Triggered Flow → Create.
2. Object: Application. Trigger: A record is created.
3. Condition Requirements: Status__c EQUALS Submitted.
4. Run the Flow: After Save (so email can access related record data). Click Done.

5. On canvas, click + → choose Action → search Send Email (core action) OR select Email Alerts action (if you created Email Alert).
    - If using Send Email:
        - To Addresses: set to $Record.Candidate__r.Email__c (click the field chooser to insert this).
        - Subject: create a Text Template resource or type Application Received: {!$Record.Job__r.Job_Title__c}.
        - Body: create a Text Template resource and reference fields.
    - If using Email Alert: pick the Email Alert EA_Application_Confirmation.
6. After the send email action, click + → Create Records → create a Task:
    - Label: Create Task - Follow up Candidate
    - How Many Records to Create: One
    - Object: Task
    - Set fields:
        - Subject = Follow up Candidate
        - WhatId = $Record.Id (or link to Application record)
        - OwnerId = $Record.OwnerId (assign to app owner)
        - ActivityDate = Formula TODAY() + 2 (or a Date resource)
        - Status = Not Started
    - Save element.
7. Save the Flow → give name **FL_Application_OnCreate_SendConfirm_CreateTask** → Activate.



Record-Triggered Flow: sends the Interview Schedule email to both the Candidate and the Interviewer

Create Text Templates

Subject Text Template

- Label: TT_Interview_Schedule_Subject
- API Name: TT_Interview_Schedule_Subject

Interview Scheduled: {!$Record.Application__r.Job__r.Name}

Body Text Template

- Label: TT_Interview_Schedule_Body
- API Name: TT_Interview_Schedule_Body

Create the Text Collection Variable for recipients

Steps:

1. Open Flow → Flows → New Flow
2. In Flow Builder left panel → New Resource:
    - Resource Type: Variable
    - API Name: var_ToAddresses
    - Label: To Address Collection
    - Data Type: Text
    - Allow multiple values (collection): Checked
    - Leave Available for Input/Output unchecked.
    - Save.

Create the Record-Triggered Flow (Start)

1. Setup → Quick Find → Flow → Flows → New Flow.
2. Choose Record-Triggered Flow → Create.
3. Configure the Start:
    - Object: Interview (select Interview__c)
    - Trigger: A record is created
    - Condition Requirements: None
    - Run the Flow: After the record is saved
    - Click Done.

Add Decision element — decide which emails to send (both/only candidate/only interviewer)

1. Click + under Start → Decision.
    - Label: D_Has_Recipient_Emails

- API Name: D_Has_Recipient_Emails

Create outcomes:

- Outcome 1:
  - Label: Both_Emails
  - API Name: Both_Emails
  - Conditions (All conditions are true):

    $Record.Application__r.Candidate__r.Email__c Is Null False

    $Record.Interviewer__r.Email Is Null False
- Outcome 2:
  - Label: Only_Candidate
  - API Name: Only_Candidate
  - Conditions:

    $Record.Application__r.Candidate__r.Email__c Is Null False

    $Record.Interviewer__r.Email Is Null True
- Outcome 3:
  - Label: Only_Interviewer
  - API Name: Only_Interviewer
  - Conditions:
    - $Record.Application__r.Candidate__r.Email__c Is Null True
    - $Record.Interviewer__r.Email Is Null False

Add Assignment(s) — populate the var_ToAddresses collection

Path: Both_Emails

1. Click + on the Both_Emails path → Assignment.
   - Label: ASG_Add_Candidate_Email (first assignment)
   - API Name: ASG_Add_Candidate_Email
   - Set Variable: var_ToAddresses Operator = Add Value = $Record.Application__r.Candidate__r.Email__c
2. Add another Assignment (or in same assignment add another line):
   - Label: ASG_Add_Interviewer_Email
   - API Name: ASG_Add_Interviewer_Email
   - Set Variable: var_ToAddresses Operator = Add Value = $Record.Interviewer__r.Email

Path: Only_Candidate

- Add Assignment: add Candidate email to var_ToAddresses.

Path: Only_Interviewer

- Add Assignment: add Interviewer email to var_ToAddresses.

Add the Send Email Action

After each Assignment (for each path), click + → Action.

In Search Actions & Resources, type Send Email → choose Salesforce core Send Email action.

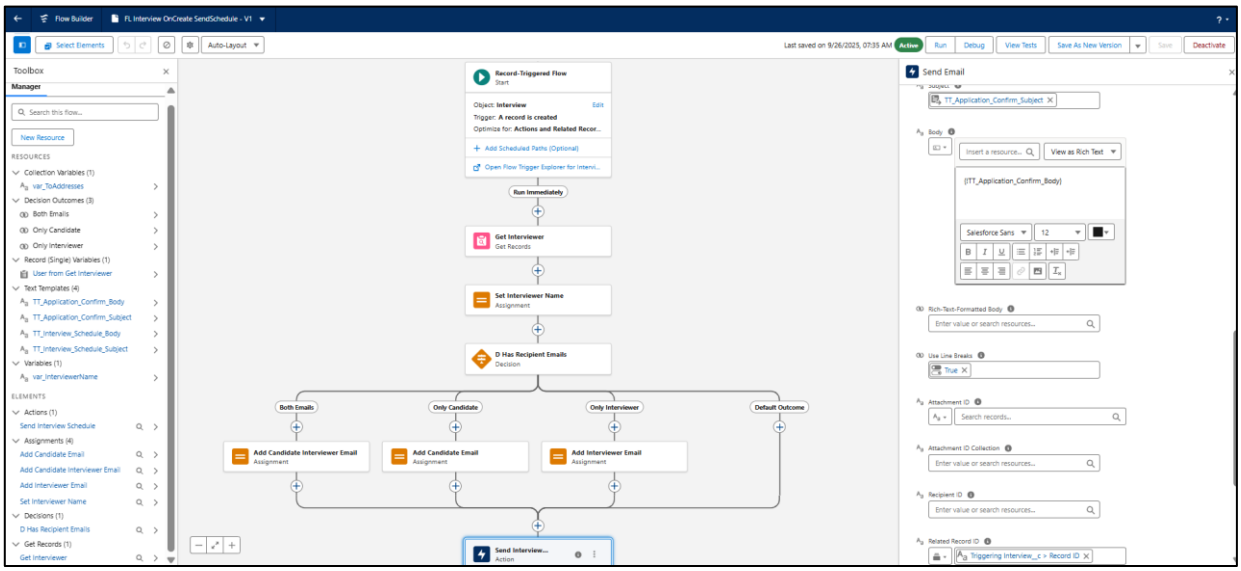
Configure the action:

- Label: ACT_Send_Interview_Schedule_Email
- API Name: ACT_Send_Interview_Schedule_Email
- Subject: select TT_Interview_Schedule_Subject (Text Template)
- Body: select TT_Interview_Schedule_Body (Text Template)
- Recipient Address Collection: select var_ToAddresses
- Sender Type: Default Workflow User (or Org-Wide Email Address if you set one)
- Use Line Breaks: True
- Rich-Text-Formatted Body: unchecked if using plain text templates
- Related Record ID: $Record.Id (links email to the Interview record)

Click Done.

Connect elements & Save Flow

- Connect Start → Decision → respective Assignments → Send Email action → End.
- Save the Flow:
    - Label: **FL_Interview_OnCreate_SendSchedule**
    - API Name: FL_Interview_OnCreate_SendSchedule
    - Description: Send interview schedule email to candidate and interviewer when Interview record is created.
- Click Activate.

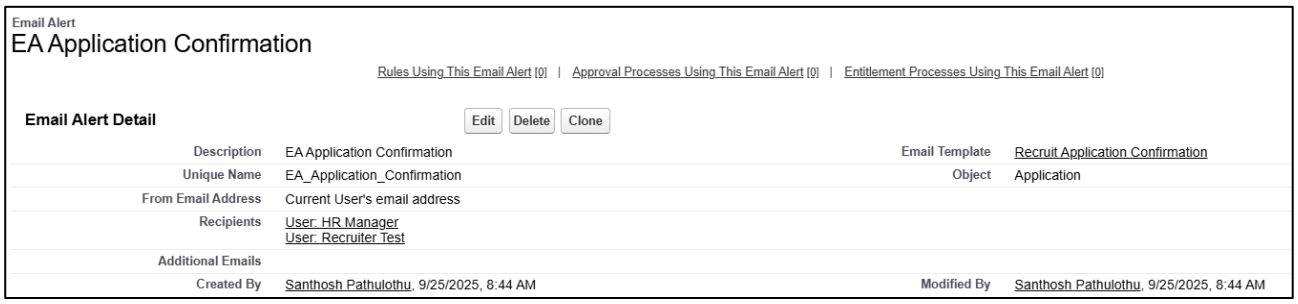


4. Email Alerts

1. EA Application Confirmation

1. Setup → Search Email Alerts → New Email Alert.
2. Gave it the label: *EA Application Confirmation*.
3. Chose the object: Application__c.
4. Selected the email template: *Recruit Application Confirmation*.
5. Set recipient type = Email Field → Candidate Email.

When a candidate applies for a job (new Application record is created), this email is automatically sent confirming their application submission.


Email Alert
EA Application Confirmation

Rules Using This Email Alert [0] | Approval Processes Using This Email Alert [0] | Entitlement Processes Using This Email Alert [0]

Email Alert Detail Edit Delete Clone

Description	EA Application Confirmation	Email Template	Recruit Application Confirmation
Unique Name	EA_Application_Confirmation	Object	Application
From Email Address	Current User's email address		
Recipients	User: HR Manager User: Recruiter Test		
Additional Emails			
Created By	Santhosh Pathulothu, 9/25/2025, 8:44 AM	Modified By	Santhosh Pathulothu, 9/25/2025, 8:44 AM

2. EA Interview Schedule

1. Setup → **Email Alerts** → New.
2. Label: *EA Interview Schedule*.
3. Object: **Interview__c**.
4. Email Template: *Recruit Interview Schedule*.
5. Recipients: Candidate Email + Interviewer (User lookup).

When an interview is scheduled (Interview__c created), this sends an email with the interview date, time, mode, and candidate details to both Candidate & Interviewer.


Email Alert
EA Interview Schedule

Rules Using This Email Alert [0] | Approval Processes Using This Email Alert [0] | Entitlement Processes Using This Email Alert [0]

Email Alert Detail Edit Delete Clone

Description	EA Interview Schedule	Email Template	Recruit_Interview_Schedule
Unique Name	EA_Interview_Schedule	Object	Interview
From Email Address	Current User's email address		
Recipients	Related User: Interviewer		

3. EA Offer Letter

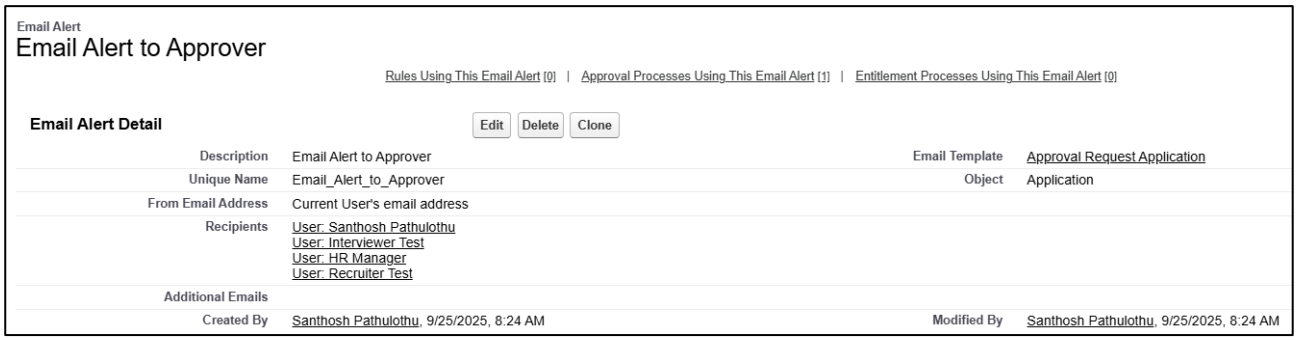
1. Setup → Email Alerts → New.
2. Label: *EA Offer Letter*.
3. Object: Application__c.
4. Email Template: *Recruit Offer Letter*.
5. Recipient: Candidate Email.

When HR approves a shortlisted candidate's application, the system sends an Offer Letter email automatically.

4. Email Alert to Approver

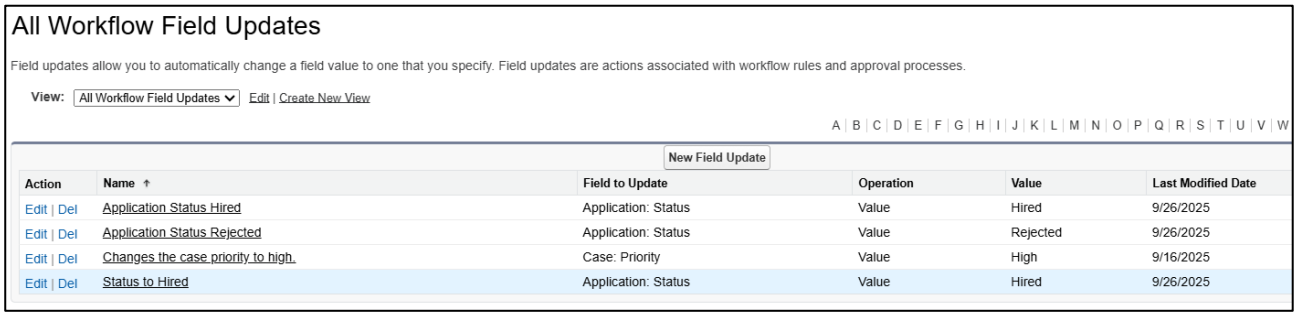
1. Setup → Email Alerts → New.
2. Label: *Email Alert to Approver*.
3. Object: Application__c.
4. Email Template: *Approval Request Application*.
5. Recipient Type: Related User → Manager/HR approver.

When an Application record enters the Approval Process, this email notifies the assigned approver (HR Manager) to review and take action (approve/reject).



5. Field Updates

When to use: Change field values automatically (via Workflow, Process Builder, Flow, Approval Process).



Application Status Hired

Setup → Workflow Actions → Field Updates → New Field Update.

1. Action Name: *Application Status Hired*.
2. Object: Application__c.
3. Field to Update: Status__c.
4. Operation: Set Value = Hired.
   Whenever a workflow/flow triggers this action (e.g., after final HR approval), the Application's status automatically changes to "Hired".

Application Status Rejected

1. Setup → Field Updates → New.
2. Action Name: *Application Status Rejected*.

3. Object: Application__c.
4. Field: Status__c.
5. Operation: Set Value = Rejected.
   If a candidate fails interviews or gets rejected, automation updates the Application's status to "Rejected".

Changes the Case Priority to High

This is a sample field update that exists in Salesforce Cases object.

- Object: Case.
- Field: Priority.
- Value: High.

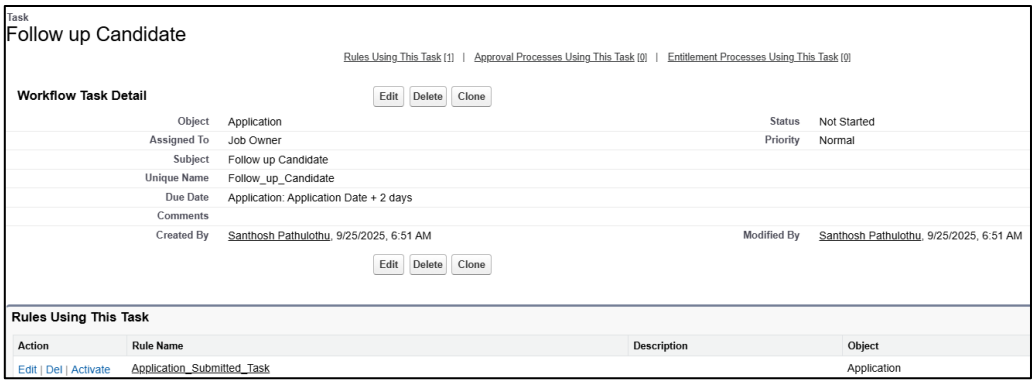
Used for Service/Support use cases. Not directly part of your Job Application project, but it automatically changes a Case's Priority to High based on conditions (e.g., escalation workflow).

6. Tasks

Create follow-up reminders for recruiters/HR.

Steps (Flow or Workflow)

1. Add Create Records element in Flow or New Task in Workflow/Process Builder.
2. Fields to set:
   - Subject: Interview follow-up
   - WhoId: candidate contact (if using Contact), otherwise leave null
   - WhatId: related job/application id (use recordId)
   - OwnerId: assign to user (e.g., Recruiter)
   - Priority: Normal/High
   - Status: Not Started
   - ActivityDate: date due
3. Save → Activate.

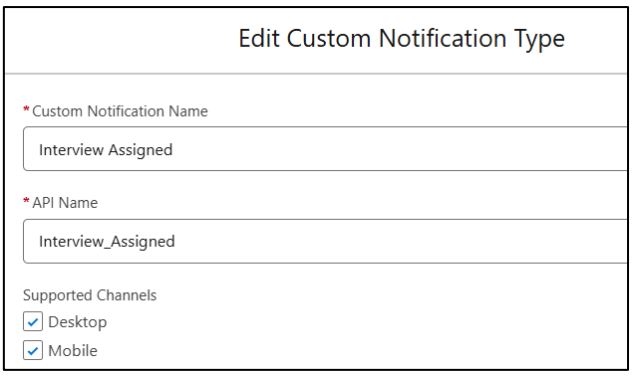


7. Custom Notifications

Send in-app push/desktop notifications to users (better UX than email for timely alerts)

Steps

1. Setup → Notification Builder → Custom Notifications → New.
    - Name: Interview_Assigned
    - Supported Channels: Desktop, Mobile (select both). Save.
2. Add a Flow Action to Send Notification:
    - In Flow Builder add Action → Choose Send Custom Notification (system action).
    - Notification Type: pick Interview_Assigned.
    - Title/Body: use Text Template with merge fields.
    - Recipient(s): set a User or User Id collection (e.g., Interviewer user lookup).
3. Save & Activate Flow.



Phase 5: Apex Programming (Developer)

1. Apex Classes & Objects — Service classes you should build

Keep business logic out of triggers. Use classes for reusability and unit testing.

Classes to create for project:

- ApplicationService — create application, duplicate check, business validations.
- InterviewService — schedule interview, send notifications (calls Flow or Queueable).
- TriggerHandlerBase — base class for standard trigger handler pattern.

Steps:

1. Setup → Developer Console → File → New → Apex Class.
2. Name it (e.g., ApplicationService) → write the code → Save.

ApplicationService.apxc

```
public with sharing class ApplicationService {
    public static void preventDuplicateApplications(List<Application__c> newApps) {
        if (newApps == null || newApps.isEmpty()) return;
        // collect candidate and job Ids
        Set<Id> candidateIds = new Set<Id>();
        Set<Id> jobIds = new Set<Id>();
        for (Application__c a : newApps) {
            if (a.Candidate__c != null) candidateIds.add(a.Candidate__c);
            if (a.Job__c != null) jobIds.add(a.Job__c);
        }
        if (candidateIds.isEmpty() || jobIds.isEmpty()) return;
        // query existing applications with those candidates and jobs
        List<Application__c> existing = [
            SELECT Id, Candidate__c, Job__c
            FROM Application__c
            WHERE Candidate__c IN :candidateIds
            AND Job__c IN :jobIds
        ];
        // build set of existing keys
        Set<String> existingKeys = new Set<String>();
        for (Application__c e : existing) {
            existingKeys.add(String.valueOf(e.Candidate__c) + '-' + String.valueOf(e.Job__c));
        }
        // mark errors on duplicates in the incoming set
        for (Application__c a : newApps) {
            if (a.Candidate__c == null || a.Job__c == null) continue;
            String key = String.valueOf(a.Candidate__c) + '-' + String.valueOf(a.Job__c);
            if (existingKeys.contains(key)) {
                a.addError('This candidate has already applied for this job.');
            }
        }
    }
    // small helper used by tests to ensure ApplicationService is callable
    public static void someUtilityMethod() {
        System.debug('ApplicationService.someUtilityMethod executed');
    }
}
```

InterviewService.apxc

```
public with sharing class InterviewService {
    public static void validateInterviewDates(List<Interview__c> newRecords) {
        if (newRecords == null) return;
        for (Interview__c i : newRecords) {
            // Use Interview_Date__c as earlier logs indicated that's the field API name
            if (i.Interview_Date__c != null && i.Interview_Date__c < Date.today()) {
                i.addError('Interview date cannot be in the past.');
            }
        }
    }

    // small helper used by tests to ensure InterviewService is callable
    public static void someMethod() {
        System.debug('InterviewService.someMethod executed');
    }
}
```

TriggerHandlerBase.apxc

```
public abstract class TriggerHandlerBase {
    /**
     * Use this static flag from tests to skip trigger/handler logic
     * when creating test data in @TestSetup. Tests should set it
     * to true during setup and set to false when they want trigger
     * behavior to run.
     */
    public static Boolean SKIP_TRIGGERS = false;
}
```

2. Apex Triggers (before/after insert/update/delete)

Respond to DML events (create/update/delete) on records.

Create one trigger per object — e.g., ApplicationTrigger, InterviewTrigger.

Steps (Developer Console):

1. Setup → Developer Console → File → New → Apex Trigger.
2. Select Object → Name, e.g., ApplicationTrigger → Save.
3. In trigger body call handler methods.

ApplicationTrigger.apxt

Code Coverage: None | API Version: 64

```
trigger ApplicationTrigger on Application__c (
    before insert,
    before update,
    after insert,
    after update
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) {
            ApplicationTriggerHandler.beforeInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            ApplicationTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) {
            ApplicationTriggerHandler.afterInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            ApplicationTriggerHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        }
    }
}
```

InterviewTrigger.apxt

Code Coverage: None | API Version: 64

```
trigger InterviewTrigger on Interview__c (
    before insert,
    before update,
    after insert
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) {
            InterviewTriggerHandler.beforeInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            InterviewTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }

    if (Trigger.isAfter) {
        if (Trigger.isInsert) {
            InterviewTriggerHandler.afterInsert(Trigger.new);
        }
    }
}
```

3. SOQL & SOSL — queries & searches

SOQL (Salesforce Object Query Language) — use to fetch records.
SOSL (Salesforce Object Search Language) — use to search text across fields (like resume text).

SOQL — get applications for candidate Ids:

```
if (candidateIds.isEmpty() || jobIds.isEmpty()) return;
// query existing applications with those candidates and jobs
List<Application__c> existing = [
    SELECT Id, Candidate__c, Job__c
    FROM Application__c
    WHERE Candidate__c IN :candidateIds
    AND Job__c IN :jobIds
];
```

4. Collections: List, Set, Map

Efficient processing and to avoid duplicates.
Examples:

Set<Id> changedRecIds when candidate or job changed, re-check duplicates.
List<Application__c> appsToUpdate to batch update records.
Map<Id, Job__c> jobById to map parent records for quick lookup.

```
// If candidate or job changed, re-check duplicates
Set<Id> changedRecIds = new Set<Id>();
for (Application__c a : newList) {
    Application__c oldA = oldMap.get(a.Id);
    if (oldA == null) continue;
    if (a.Candidate__c != oldA.Candidate__c || a.Job__c != oldA.Job__c) changedRecIds.add(a.Id);
}
if (!changedRecIds.isEmpty()) {
    List<Application__c> changed = new List<Application__c>();
    for (Application__c a : newList) if (changedRecIds.contains(a.Id)) changed.add(a);
    ApplicationService.preventDuplicateApplications(changed);
}
isExecuting = false;
```

5. Batch Apex:

Batch Apex — processing large data sets

- Bulk update Applications (e.g., mark old applications as Expired)
- Mass email or data cleanup (when >50,000 records)

Steps to create Batch Apex:

1. Create an Apex class implementing Database.Batchable<sObject>.
2. Implement start, execute, finish.
3. Optionally implement Database.Stateful to preserve state across execute batches.

```
global class ExpireOldApplicationsBatch implements Database.Batchable<sObject>, Database.AllowsCallouts {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        String q = 'SELECT Id, Status__c, Application_Date__c FROM Application__c WHERE Status__c IN (\'Submitted\', \'Under Review\')';
        return Database.getQueryLocator(q);
    }

    global void execute(Database.BatchableContext bc, List<sObject> scope) {
        List<Application__c> updates = new List<Application__c>();
        Date today = Date.today();
        for (sObject s : scope) {
            Application__c a = (Application__c)s;
            if (a.Application_Date__c != null && a.Application_Date__c.addDays(60) < today) {
                Application__c up = new Application__c(Id = a.Id, Status__c = 'Expired');
                updates.add(up);
            }
        }
        if (!updates.isEmpty()) {
            update updates;
        }
    }

    global void finish(Database.BatchableContext bc) {
        // Optionally enqueue follow-up queueable or send email summary
    }
}
```

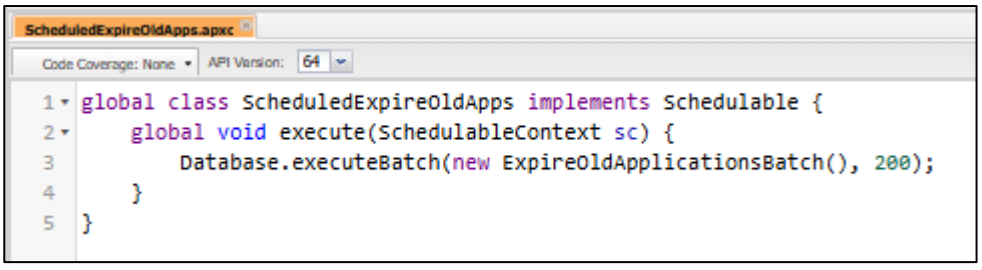
6. Queueable Apex

Process resume attachments (heavy CPU or callout)
Chain jobs (e.g., after batch finish enqueue further processing)
Replace some use-cases of future methods with more control

```
@IsTest
static void testInterviewQueueable() {
    // Create Candidate + Job + Application first
    Candidate__c cand = new Candidate__c(Name='Cand Q', Email__c='q@app.com');
    insert cand;

    Job__c job = new Job__c(Name='Dev Intern');
    insert job;

    Application__c app = new Application__c(
        Candidate__c=cand.Id,
        Job__c=job.Id,
        Status__c='Submitted',
        Application_Date__c = Date.today()
    );
    insert app;
```

7. Schedule Apex

- Run Batch Apex daily/weekly to expire applications, refresh counters.
- Update dashboard data at night.

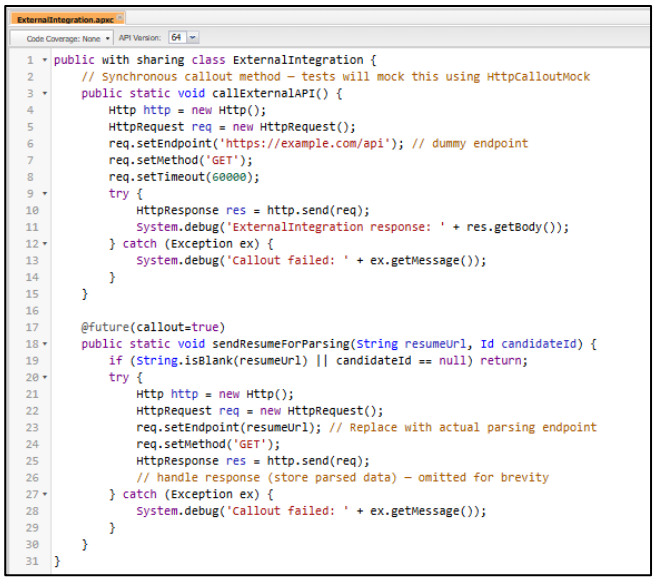


```
global class ScheduledExpireOldApps implements Schedulable {
    global void execute(SchedulableContext sc) {
        Database.executeBatch(new ExpireOldApplicationsBatch(), 200);
    }
}
```

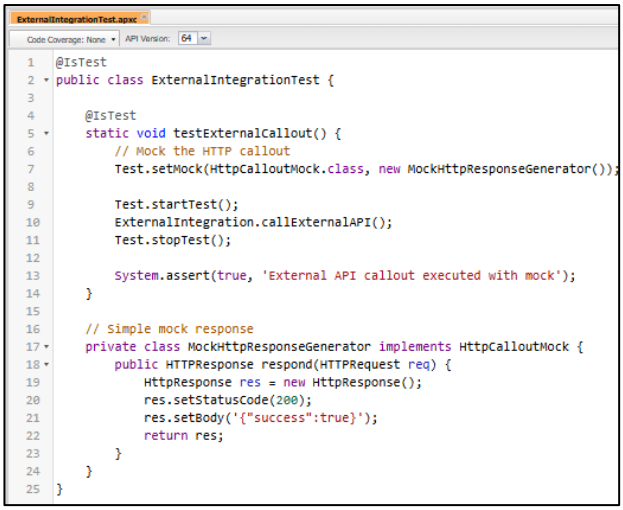
```
@IsTest
static void testScheduledExpireOldApps() {
    Test.startTest();
    String jobId = System.schedule(
        'TestSched',
        '0 0 12 * * ?',
        new ScheduledExpireOldApps()
    );
    Test.stopTest();
    System.assertNotEquals(null, jobId, 'Scheduled job should run');
}
```

8. @future Methods (Callouts) — for external integrations

- Call external resume parsing API after interview creation.
- Send data to third-party ATS.



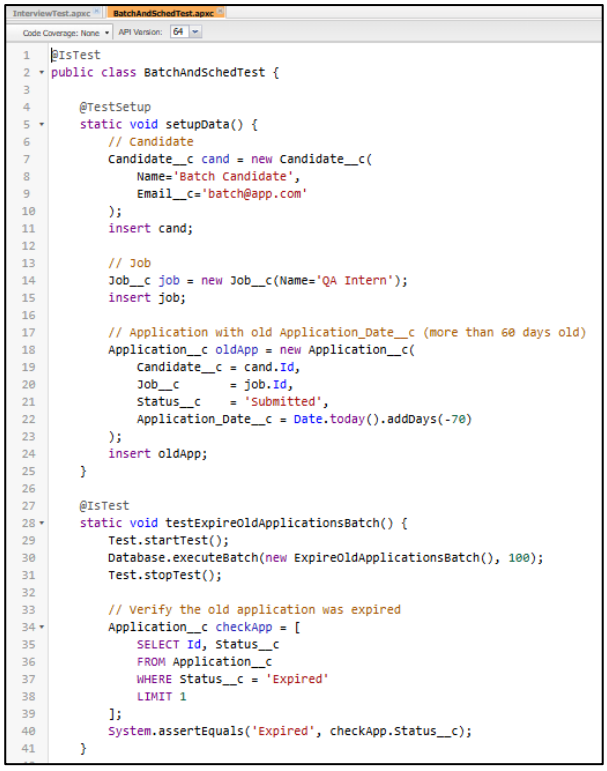
```apex
public with sharing class ExternalIntegration {
    // Synchronous callout method – tests will mock this using HttpCalloutMock
    public static void callExternalAPI() {
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://example.com/api'); // dummy endpoint
        req.setMethod('GET');
        req.setTimeout(60000);
        try {
            HttpResponse res = http.send(req);
            System.debug('ExternalIntegration response: ' + res.getBody());
        } catch (Exception ex) {
            System.debug('Callout failed: ' + ex.getMessage());
        }
    }

    @future(callout=true)
    public static void sendResumeForParsing(String resumeUrl, Id candidateId) {
        if (String.isBlank(resumeUrl) || candidateId == null) return;
        try {
            Http http = new Http();
            HttpRequest req = new HttpRequest();
            req.setEndpoint(resumeUrl); // Replace with actual parsing endpoint
            req.setMethod('GET');
            HttpResponse res = http.send(req);
            // handle response (store parsed data) – omitted for brevity
        } catch (Exception ex) {
            System.debug('Callout failed: ' + ex.getMessage());
        }
    }
}
```



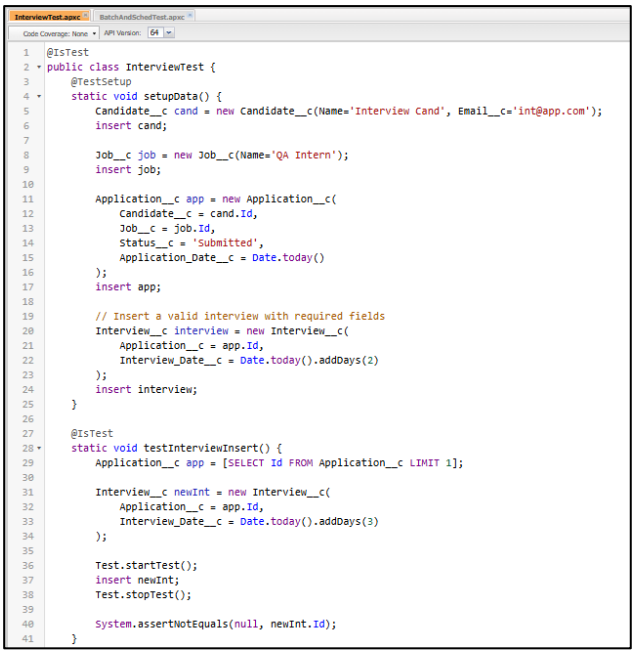
```apex
@IsTest
public class ExternalIntegrationTest {

    @IsTest
    static void testExternalCallout() {
        // Mock the HTTP callout
        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());

        Test.startTest();
        ExternalIntegration.callExternalAPI();
        Test.stopTest();

        System.assert(true, 'External API callout executed with mock');
    }

    // Simple mock response
    private class MockHttpResponseGenerator implements HttpCalloutMock {
        public HTTPResponse respond(HTTPRequest req) {
            HttpResponse res = new HttpResponse();
            res.setStatusCode(200);
            res.setBody('{"success":true}');
            return res;
        }
    }
}
```

9. Test Classes

- Use `@isTest` classes.
- Do not use `SeeAllData=true` (create all test data in test).
- Use `Test.startTest()` and `Test.stopTest()` to run async jobs, batch, queueable, future.
- Assert expected behavior using `System.assert`.
- Cover positive and negative cases (duplicates, invalid dates, bulk insert).



```apex
@IsTest
public class BatchAndSchedTest {

    @TestSetup
    static void setupData() {
        // Candidate
        Candidate__c cand = new Candidate__c(
            Name='Batch Candidate',
            Email__c='batch@app.com'
        );
        insert cand;

        // Job
        Job__c job = new Job__c(Name='QA Intern');
        insert job;

        // Application with old Application_Date__c (more than 60 days old)
        Application__c oldApp = new Application__c(
            Candidate__c = cand.Id,
            Job__c       = job.Id,
            Status__c    = 'Submitted',
            Application_Date__c = Date.today().addDays(-70)
        );
        insert oldApp;
    }

    @IsTest
    static void testExpireOldApplicationsBatch() {
        Test.startTest();
        Database.executeBatch(new ExpireOldApplicationsBatch(), 100);
        Test.stopTest();

        // Verify the old application was expired
        Application__c checkApp = [
            SELECT Id, Status__c
            FROM Application__c
            WHERE Status__c = 'Expired'
            LIMIT 1
        ];
        System.assertEquals('Expired', checkApp.Status__c);
    }
```



```apex
@IsTest
public class InterviewTest {
    @TestSetup
    static void setupData() {
        Candidate__c cand = new Candidate__c(Name='Interview Cand', Email__c='int@app.com');
        insert cand;

        Job__c job = new Job__c(Name='QA Intern');
        insert job;

        Application__c app = new Application__c(
            Candidate__c = cand.Id,
            Job__c = job.Id,
            Status__c = 'Submitted',
            Application_Date__c = Date.today()
        );
        insert app;

        // Insert a valid interview with required fields
        Interview__c interview = new Interview__c(
            Application__c = app.Id,
            Interview_Date__c = Date.today().addDays(2)
        );
        insert interview;
    }

    @IsTest
    static void testInterviewInsert() {
        Application__c app = [SELECT Id FROM Application__c LIMIT 1];

        Interview__c newInt = new Interview__c(
            Application__c = app.Id,
            Interview_Date__c = Date.today().addDays(3)
        );

        Test.startTest();
        insert newInt;
        Test.stopTest();

        System.assertNotEquals(null, newInt.Id);
    }
```

10.Asynchronous Processing

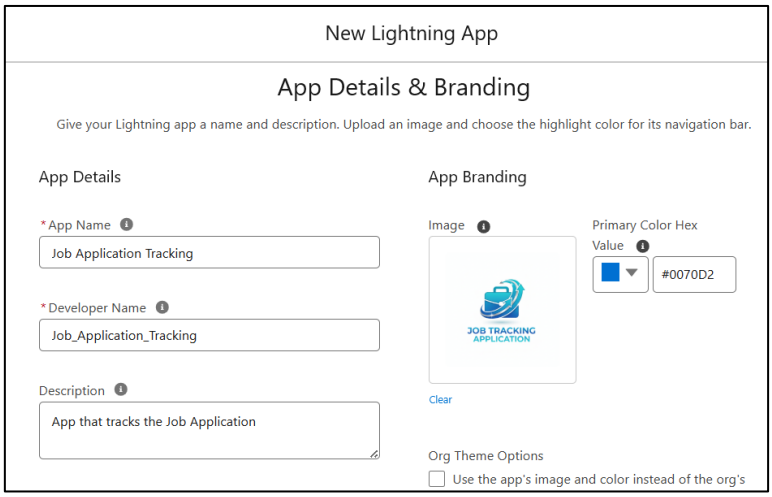
Batch Apex — use when you need to process many Application records (>50k) or heavy updates. Example: monthly cleanup, mass status changes.
Queueable — for medium-sized chaining tasks and complex processing per record (resume processing, 3rd party calls).
Scheduled Apex — to kick off Batch or Queueable at scheduled times.
Future — for simple fire-and-forget callouts (legacy; Queueable preferred now).

Phase 6 — User Interface Development

1. Create a Lightning App (Job Application Tracker)

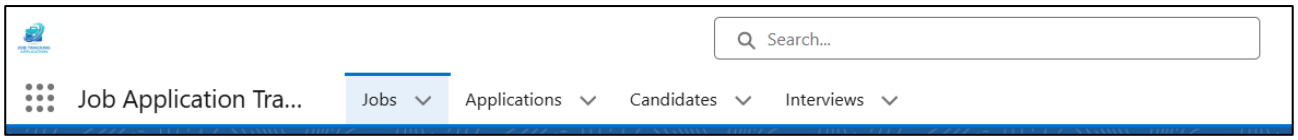
Create a custom app that contains the tabs your users will use.

1. Setup (⚙) → **App Manager**.
2. Click **New Lightning App** (top right).
3. App Details:
   - **App Name:** Job Application Tracker
   - **Description:** Recruiting & internship application tracking
   - **Developer Name:** auto filled
   - Click **Next**.
4. Branding (optional):
   - Choose an **App Logo** or upload a small logo, pick an **Accent Color**; these appear in the header.
   - Click **Next**.



5. Navigation Items:
   - Move these items from Available to Selected in this order:
     Candidates (object tab), Jobs (object tab), Applications (object tab), Interviews (object tab), Reports, Dashboards.
   - Click **Next**.
6. Utility Bar (we'll also configure in more detail later) — skip here or leave defaults → **Next**.
7. Assign to profiles: choose which profiles can see this app (Recruiter, HR Manager, Interviewer).
8. Click **Save**.

Test: App Launcher (grid icon) → search Job Application Tracker → open it. You should see tabs in the left nav.

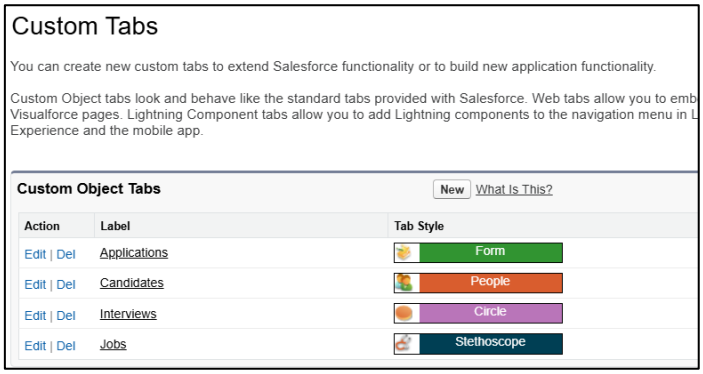


2. Create Custom Object Tabs

Setup → **Tabs** → **New** under **Custom Object Tabs**.

1. For **Object:** choose Candidate.
    - Tab Style: choose an icon.
    - Click **Next** → choose which profiles get it → Save.
2. Repeat for Job, Application, Interview.

Then add these tabs to your Lightning App (App Manager → Edit App → Navigation Items) if not already present.



3. Lightning Record Pages

Create a custom Record Page for each of the 4 main objects. I'll give the Candidate page and Application page as detailed templates — repeat the approach for Job & Interview.

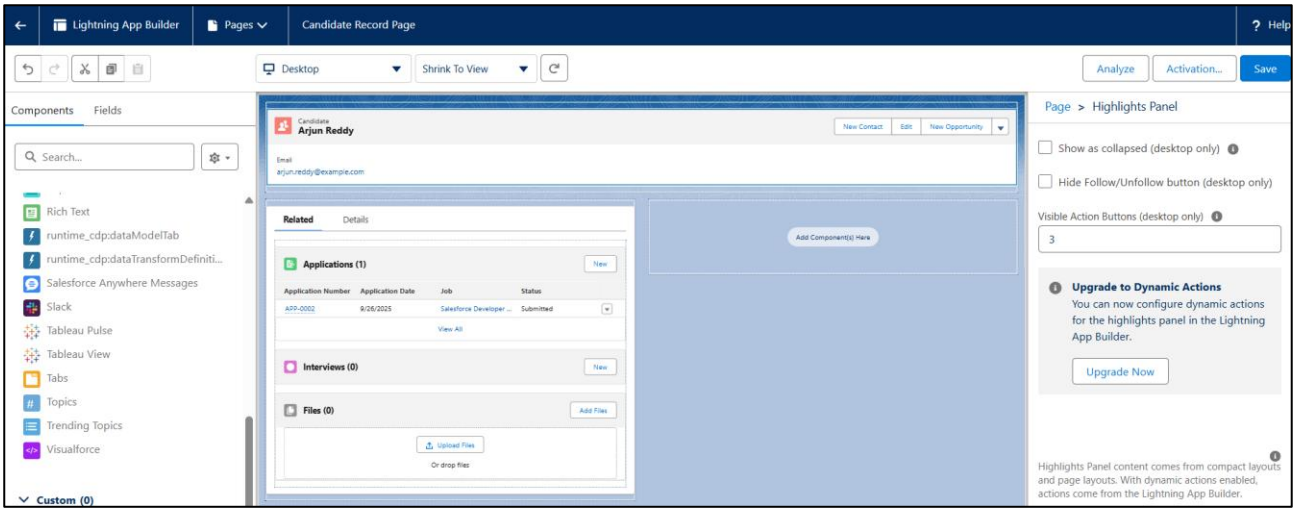
Candidate Record Page (show resume, apps, quick actions)

1. Setup → Object Manager → **Candidate** → **Lightning Record Pages**.
2. Click **New** → Choose **Record Page** → **Next**.
3. Template: pick a standard template (e.g., Header, Left Sidebar, Right Sidebar) or Two Regions — click **Finish**.
4. You're in Lightning App Builder canvas. Do this:
    - At the top-left, click the **Highlights Panel** component (or add it) — this shows record title, key fields and quick actions.
        - In Highlights Panel properties set **Fields** to use the Candidate Compact Layout (ensure Candidate compact layout shows Name, Email, Phone).

- Drag **Tabs** component into main area (if you used a template without tabs). Create tabs: Details, Related, Activities.
- In **Details** tab drop **Record Detail** component (shows full fields).
- In **Related** tab add:
  - **Related Lists** (the standard one) OR **Related List - Single** for specific lists:
    - Drag **Related List - Single**, set **Related List** = Applications → check **Show action bar** if you want New application directly.
    - Add another **Related List - Single** → set to Interviews.
    - Add **Related List - Single** → set to Files (we'll relabel as Resumes; see step below).

5. Add **Quick Actions** to Highlights:
   - Edit the Page → click the Highlights Panel → ensure the action layout includes Create Application.
   - If you don't see the action, ensure it's added to the page layout's Mobile & Lightning Actions (Object Manager → Candidate → Page Layouts → Edit → Mobile & Lightning Actions).
6. Save → Click **Activation**.
   - Choose **App Default** (select Job Application Tracker) and **Make default for this app and for Desktop**.
   - Optionally set for profiles/record types (useful if Recruiter and HR need different pages).
7. Click **Back** → **Save**.

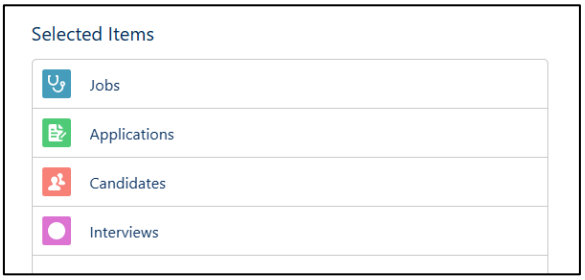
Test: Open the App → Candidates → open a Candidate record → verify Resumes related list, Create Application action in highlights, Applications and Interviews related lists show correct items.



4 — Customize Tabs & Navigation (make app nav friendly)

You already added nav items when creating the app. To reorder/change:

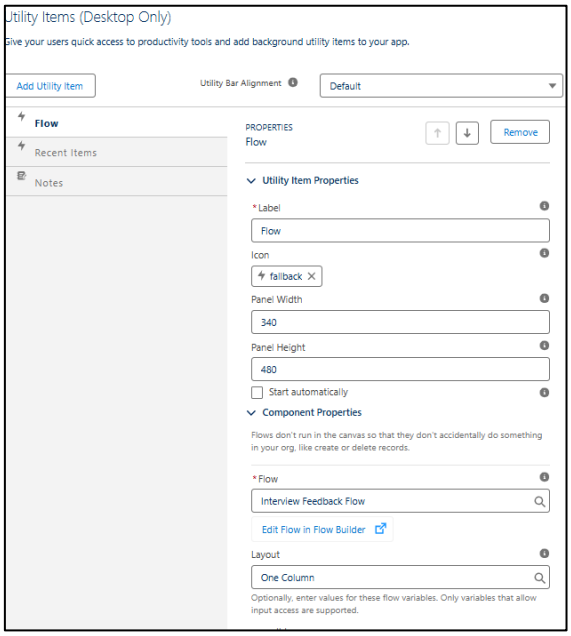
1. Setup → **App Manager** → find Job Application Tracker → click the dropdown → **Edit**.
2. Choose **Navigation Items** → add/remove/reorder tabs (Candidates, Jobs, Applications, Interviews, Reports, Dashboards).
3. Save.



6 — Utility Bar (quick access tools at bottom of app)

Add useful tools: Notes, New Application quick action, Recent Items, Global Search.

1. Setup → **App Manager** → find Job Application Tracker → click dropdown → **Edit**.
2. Left side: click **Utility Items** (or **Utility Bar**) → **Add Utility Item**.
3. Recommended items to add (in this order):
    - **Notes** — quick jot for recruiters.
    - **Recent Items** (if available) — quick jump to recently viewed Applications.
    - **New Action** → select the **Create Application** action (so you can quickly start an Application anywhere).
    - **Macros/Flow** — If you built a Screen Flow (e.g., Interview Feedback), add the Flow as a utility item so users can open it as a floating panel.



4. For each Utility Item, set:
    - **Label** (short), **Icon**, **Panel Width/Height** (defaults fine).
    - For the Create Application action, set default behavior (open as modal).
5. Save → Finish and **Save** the app.

7 — Assign Record Pages & Home Page to Profiles / Record Types

Activation choices let you show different pages to different profiles:

1. When you click **Activate** on a Lightning Record Page, choose one of:
    - **Org Default** (all users)
    - **App Default** (only for your App)
    - **App, Record Type, and Profile** (choose this to assign the page to specific Profiles and Record Types).
2. For Home Page Activation you can assign to specific Profiles and Apps.

Phase 7 — Integration & External Access

1. Named Credentials

- **What it is:** A secure way to store authentication details (like API endpoints, usernames, passwords, OAuth tokens).
- **In your project:** If you integrate with an **external Resume Parsing API** or a **Job Portal API (e.g., LinkedIn, Naukri, Indeed)**, you would use Named Credentials so Apex callouts don't need hard-coded credentials.

    Resume_Parsing_Credential to securely call an external resume parsing service.

2. External Services

- **What it is:** Lets Salesforce connect to external APIs using declarative (no-code) setup via Schema/Swagger files.
- **In your project:** Suppose you want to connect to a **Background Verification service** or **Credit Check API** for candidates. Instead of writing Apex callouts, you can register it as an **External Service** and call it from **Flow**.

3. Web Services (REST/SOAP)

- **What it is:** Salesforce can consume external REST/SOAP APIs or expose its own.
- **In your project:**
    - **Consume:** Use REST to fetch **job postings** from a partner portal.
    - **Expose:** Allow third-party portals (like your company's careers site) to submit **Applications** directly into Salesforce via a REST API you create.

4. Callouts

- **What it is:** Outbound requests from Salesforce to external systems.
- **In your project:**
    - Call an **AI resume parser** when a candidate uploads a resume.
    - Call an **email verification API** to validate candidate email addresses.

5. Platform Events

- **What it is:** Salesforce event-driven architecture (publish-subscribe).
- **In your project:**
    - When an **Application Status = Hired**, publish an event → HR system subscribes and auto-generates an Offer Letter.
    - When **Interview Scheduled**, trigger a platform event that notifies external calendar systems (Google Calendar, Outlook).

6. Change Data Capture (CDC)

- **What it is:** Real-time streaming of record changes (insert, update, delete).
- **In your project:**
    - If a candidate's application status changes (e.g., *Shortlisted → Hired*), CDC can stream this event to external systems (like HR payroll system or Slack notifications).

7. Salesforce Connect

- **What it is:** Lets you connect external databases/data sources without storing them in Salesforce (data stays external but is viewable in Salesforce).
- **In your project:**
    - Connect to an external **HRMS database** or **Employee Onboarding system** so recruiters can view onboarding details **inside Salesforce** without duplication.

8. API Limits

- **What it is:** Salesforce has daily API call limits (depends on edition).
- **In your project:**
    - If resumes are parsed via an external API for every application, you must design with limits in mind (e.g., batch processing, Queueable jobs).
    - Avoid hitting limits if multiple job portals are integrated.

9. OAuth & Authentication

- **What it is:** Secure authentication standard for integrations.
- **In your project:**
    - If a **job portal** or **HR app** wants to access Salesforce data (like candidate status), they authenticate via OAuth.
    - If you expose Salesforce REST API, you'd require OAuth login for security.

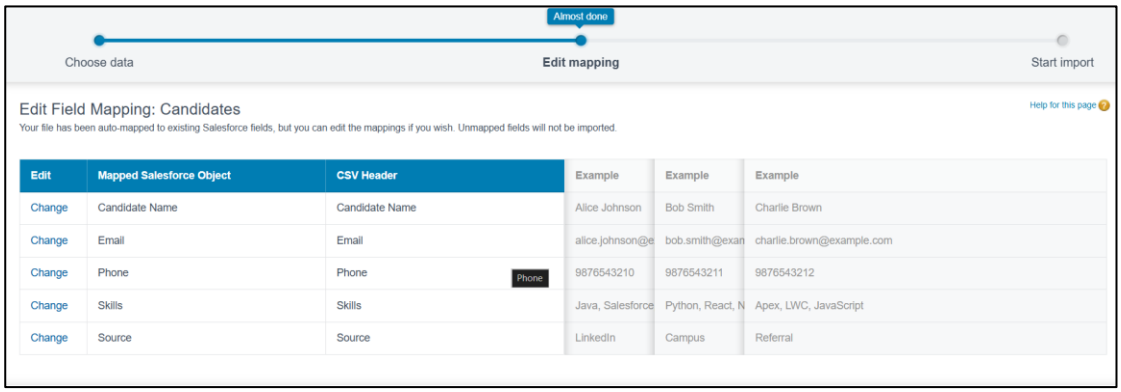
10. Remote Site Settings

- **What it is:** Whitelist external endpoints before making callouts.
- **In your project:**
    - If you make callouts to https://resumeparser.com/api, you must add it to **Remote Site Settings**.
    - Without it, Salesforce will block the callout for security reasons.

Phase 8 — Data Management & Deployment

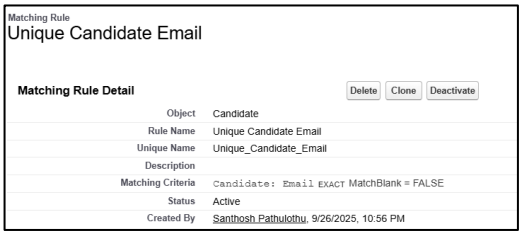
1. Data Import Wizard

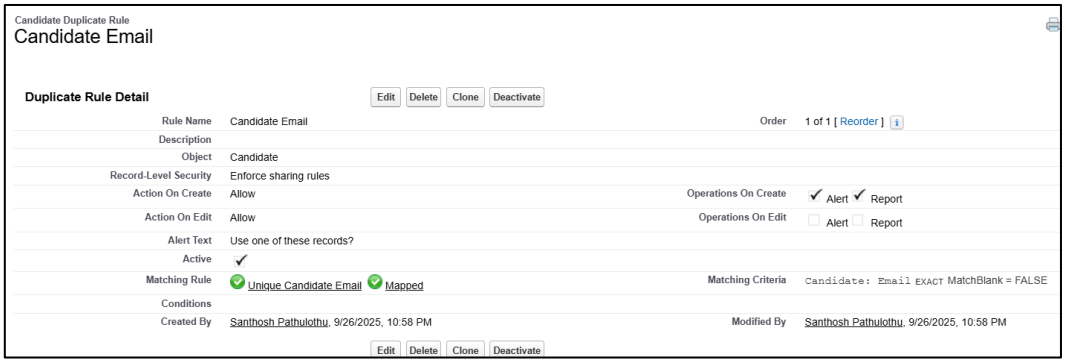
- **Use case in your project:** Import sample **Candidates, Jobs, Applications, Interviews** (CSV).
- **Steps:**
    1. Go to **Setup → Data Import Wizard**.
    2. Select Object (e.g., Candidate).
    3. Upload CSV (your test dataset).
    4. Map fields (e.g., CSV Email → Salesforce Email__c).
    5. Run Import → Check records in Salesforce.



Duplicate Rules

- **Use case:** Prevent duplicate **Candidate emails** or duplicate **Job IDs**.
- **Steps:**
    1. Setup → Duplicate Rules → New Rule.
    2. Object = Candidate.
    3. Matching Rule = Email must be unique.
    4. Action = Block or Allow with Alert.

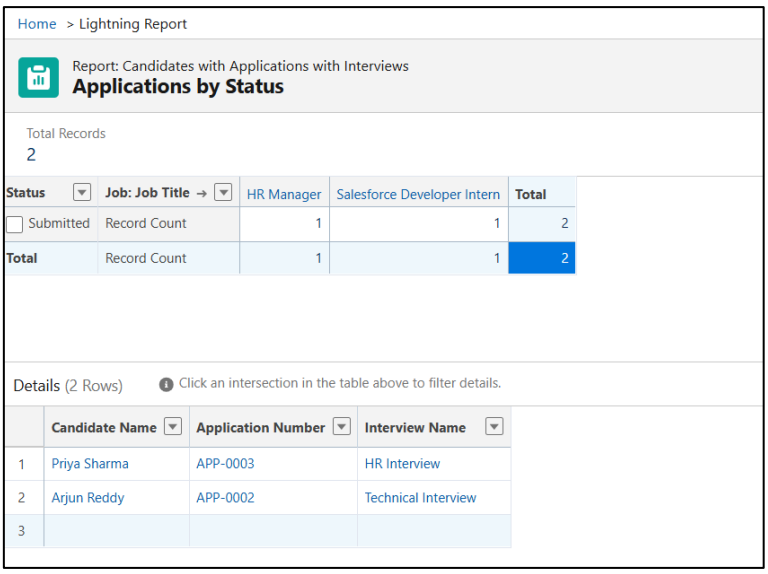




Phase 9 — Reporting, Dashboards

Step 1: Create a Report (Applications Pipeline)

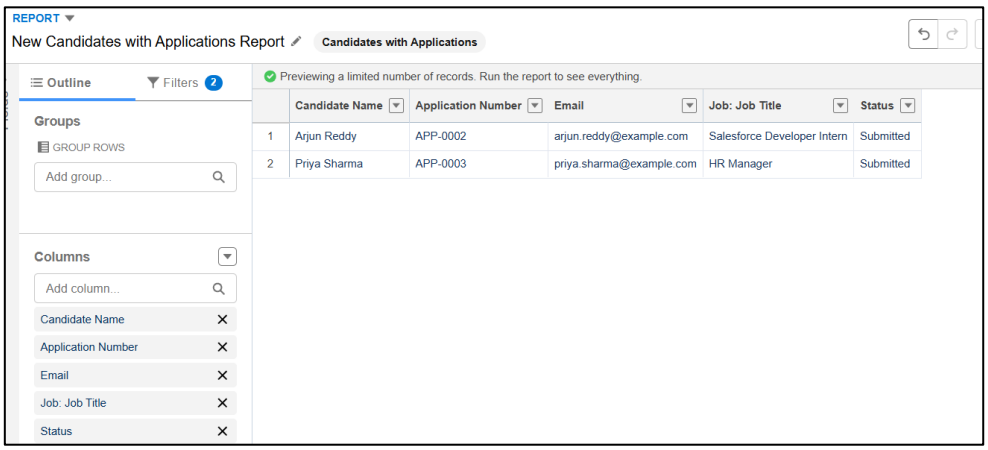
1. In Salesforce, go to the App Launcher (grid icon) → search Reports → click Reports.
2. Click New Report (top right).
3. In the report type search bar, type Applications → select Applications → click Continue.
4. By default, a table with some fields will appear.
5. Choose the format:
    - Tabular: Leave as default (just rows).
    - Summary: Drag Status into the "Group Rows" section → now Applications are grouped by Status.
    - Matrix: Drag Status to "Group Rows" and Job__c to "Group Columns".
6. Click Save & Run.
    - Name = “Applications by Status”.
    - Folder = Public Reports (so others can see).



Step 2: Create a Custom Report Type (Candidate + Applications)

1. Setup → Quick Find → Report Types → New Custom Report Type.
2. Primary Object = Candidate.
3. Related Object = Applications (Each candidate may or may not have applications).
4. Save.
5. Now go back to Reports → New Report → Choose Custom Report Type → Candidate with Applications.
6. Add fields: Candidate Name, Email, Job Title, Application Status.

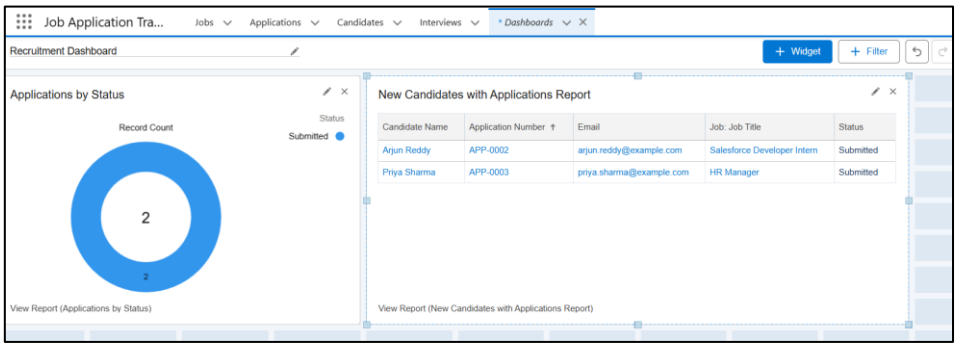
7. Save as “Candidates with Applications”.



Step 3: Create a Dashboard

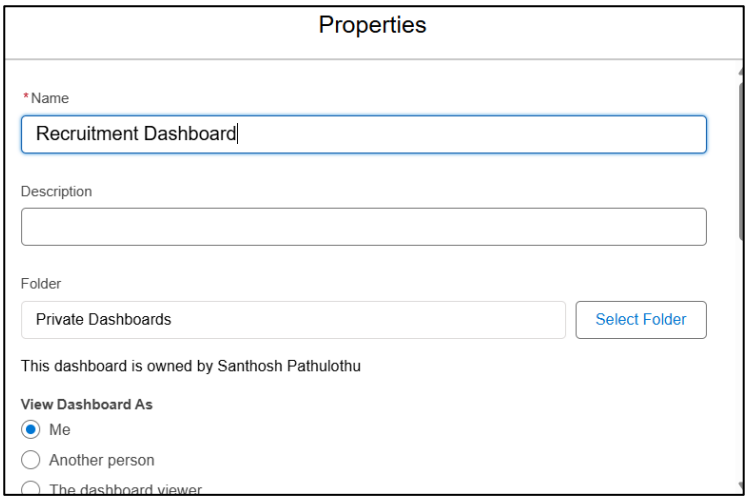
1. Go to App Launcher → search Dashboards → click Dashboards.
2. Click New Dashboard.
   - Name = “Recruitment Dashboard”.
   - Folder = Public Dashboards.
3. Click + Component.
   - Choose the Applications by Status report.
   - Display as Pie Chart → Show percentage of Submitted/Shortlisted/Hired/Rejected.
4. Click + Component again.
   - Choose Applications by Job report.
   - Display as Bar Chart → Applications count per Job.

   Save → Done.



Step 4: Make it Dynamic

1. Open the Dashboard you created → click Edit.
2. In Dashboard Settings (top right), set:
   - View Dashboard As → Logged-in User (instead of a fixed user).
3. Save.



📌 What this does:

- A Recruiter sees only their candidates.
- An HR Manager sees all applications.