PROJECTTITLE

Smart Recruit

(Salesforce-powered Job Application Tracking System)

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

Current Challenge

Recruiters and HR managers manually track applications using emails and spreadsheets. This results in:

- Difficulty managing high volumes of applications.
- Missed follow-ups due to lack of automation.
- Poor coordination between recruiters and HR managers.
- No real-time visibility for leadership into recruitment metrics.

Functional Requirements (What the system must do)

1. **Job Posting Management**
    a. HR/Recruiters should be able to create job postings with fields like:
        i. Job Title
        ii. Department

      iii. Location
      iv. Required Skills
      v. Application Deadline
   b. Recruiters should be able to update job postings when positions are filled.

2. **Candidate Management**
   a. Capture candidate information:
      i. Name
      ii. Contact Details
      iii. Resume (File Upload or URL)
      iv. Experience, Skills
   b. Maintain a history of applications per candidate.
3. **Application Tracking**
   a. Each application should move through lifecycle stages:
      i. Applied
      ii. Shortlisted
      iii. Interview Scheduled
      iv. Offered
      v. Hired / Rejected
   b. System should automatically update status when recruiters take action.
4. **Workflow Automation**
   a. **Flows/Process Builder**:
      i. Auto-create Interview record when an application is shortlisted.
      ii. Auto-send email/SMS notification to candidates at key stages.
   b. **Approval Process**:
      i. Offer letter stage must be approved by HR Manager.
5. **Notifications & Communication**
   a. Email alerts to candidates (status updates).
   b. Email notifications to recruiters (new application received).
   c. Reminder notifications for interviews.
6. **Reports & Dashboards**
   a. HR Managers should have dashboards showing:
      i. Applications by Job Posting.
      ii. Applications by Status (Funnel: Applied → Hired).
      iii. Recruiter Performance (applications handled, conversions).
   b. Recruiters should be able to run reports on their own candidates.

Non-Functional Requirements (System qualities)

1. **Usability**
    a. Simple UI for recruiters to add/manage applications.
    b. Lightning App with Tabs for Job Postings, Candidates, Applications, Interviews.
2. **Scalability**
    a. Should handle many applications without performance issues.
3. **Security**
    a. OWD: Applications private by default.
    b. Recruiters can only see applications they own.
    c. HR Managers can see all applications.
    d. Field-level security: sensitive fields (salary expectations, offer details) restricted.
4. **Reliability**
    a. Automated workflows should ensure no missed updates or communications.
5. **Extensibility (Future Scope)**
    a. Later, system can integrate with external portals like LinkedIn or Naukri.
    b. AI-based candidate ranking could be added.

Stakeholder Analysis

Stakeholder	**Role**		**Needs / Expectations**
HR Manager	Oversees recruitment process		- Access to recruitment KPIs via dashboards - Approval workflows for job offers - Compliance tracking
Recruiter	Manages job postings & candidates		- Simple interface for job posting & application tracking - Automated status updates - Email notifications for new application

Candidate	Applies for jobs		- Timely updates on application status - Transparent & fair process - Smooth communication
Management / Leadership	Reviews hiring metrics		- High-level dashboards & reports - KPIs like time-to-hire, offer acceptance rate - Data-driven insights for strategic decisions

Business Process Mapping

Current Process (Manual System)

Recruitment is handled mostly through offline tools like emails, spreadsheets, and job portals. This results in inefficiencies, delays, and lack of visibility:

- **Job Posting Creation** → Recruiters prepare job postings in Excel or upload them to third-party job portals, with no centralized control

- **Candidate Application** → Applications arrive via personal or shared HR emails. Resumes are stored manually in folders, often leading to mismanagement.

- **Status Tracking** → Recruiters track candidate progress (Applied, Shortlisted, Interviewed, etc.) using spreadsheets. This is error-prone and not transparent.

- **Communication** → Recruiters send emails manually to update candidates, leading to inconsistent or delayed responses.

- **Leadership Monitoring** → Management has no real-time insights into recruitment KPIs like time-to-hire, offer acceptance, or pipeline health.

Limitations:

- Manual handling increases chances of errors.
- Delayed candidate communication impacts candidate experience.
- No centralized system → duplication of data.

- Lack of dashboards → leadership cannot take data-driven decisions.

Proposed Process (Salesforce-Powered ATS)

By leveraging Salesforce, the recruitment lifecycle becomes automated, transparent, and trackable in real time:

- **Job Posting Creation** → Recruiters create structured Job Posting records in Salesforce, specifying department, role, skills, and deadlines.

- **Candidate Application** → When a candidate applies, their details and resume are captured in a Candidate object, automatically linked to a Job Posting.

- **Application Lifecycle** → Applications move through defined stages (*Applied* → *Shortlisted* → *Interview* → *Offer* → *Hired/Rejected*).
  - When an application is **Shortlisted**, a **Flow** auto-creates an Interview record and sends an email notification to the candidate.
  - When status = **Offer**, an **Approval Process** routes the request to the HR Manager for final confirmation.

- **HR Manager Approval** → If approved, the application status updates to **Hired** and dashboards refresh automatically.

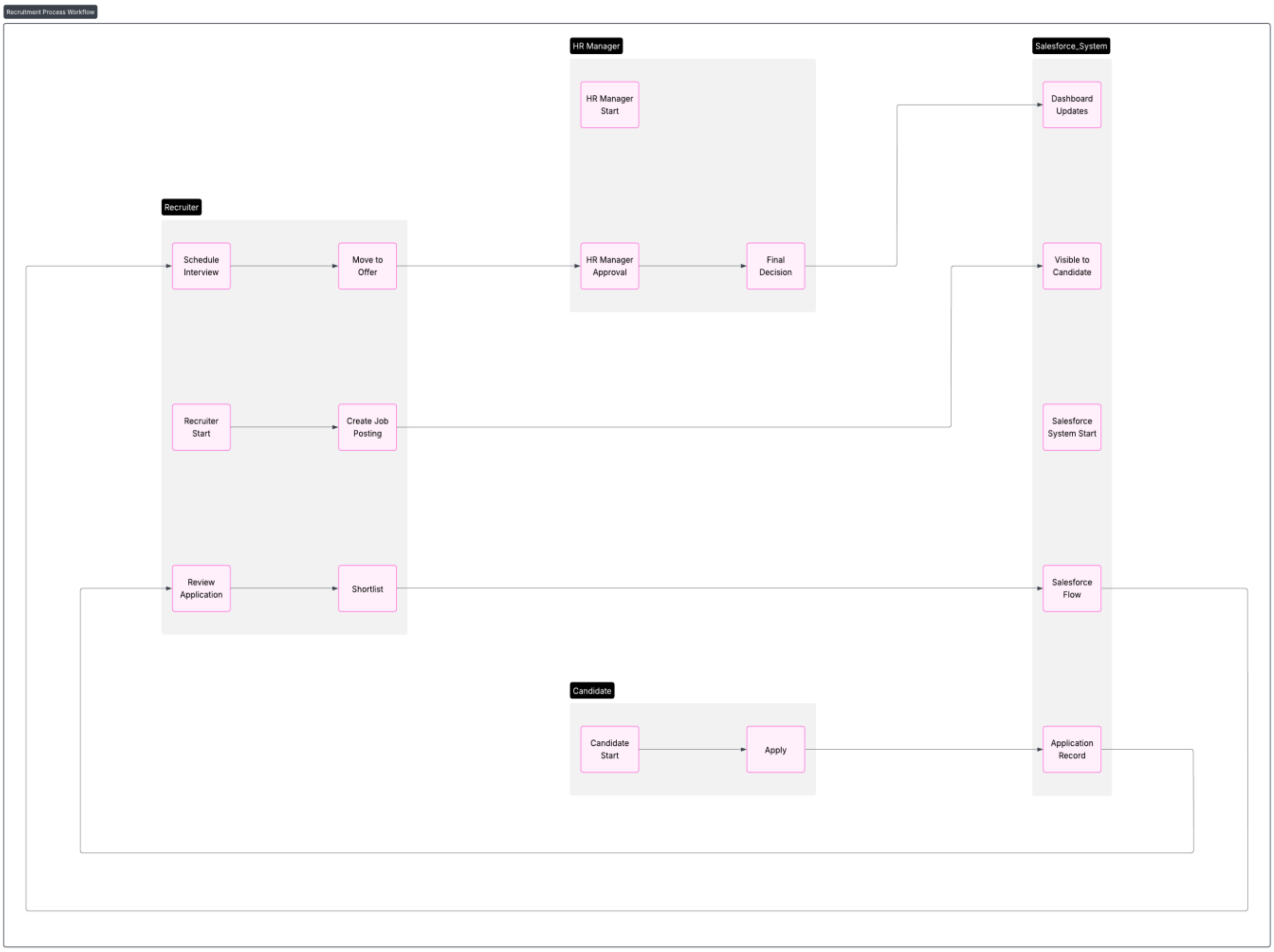
- **Automated Notifications** → Email alerts keep candidates informed at every stage.

- **Leadership Monitoring** → Dashboards and reports give real-time insights into hiring pipeline, recruiter performance, and bottlenecks.

Advantages:

- Centralized data storage → all records in Salesforce.
- Real-time communication with candidates → improves candidate experience.
- Automated workflows reduce manual effort.
- Dashboards provide actionable insights for HR and leadership.
- Scalable → system can handle hundreds of job applications simultaneously.

Workflow Diagram



Industry-Specific Use Case Analysis for Smart Recruit

1. IT Services & Consulting (TCS, Infosys, Wipro)

- **Challenge**: Thousands of applicants in campus drives, manual shortlisting, multiple interview rounds.

- **How Smart Recruit Helps**:
    - Auto-assign candidates to recruiters based on location/skill.
    - Approval workflows for offer letters.
    - Real-time dashboards of hiring funnel across multiple campuses.

2. Healthcare Industry (Hospitals & Pharma)

- **Challenge**: High demand for skilled nurses, doctors, pharmacists; manual hiring delays impact patient care.
- **How Smart Recruit Helps**:
    - Track applicant licenses & certifications as part of candidate records.
    - Automate scheduling of interviews with department heads.
    - Approval workflows for onboarding sensitive roles (like surgeons).

3. Retail & E-commerce (Amazon, Flipkart, Reliance Retail)

- **Challenge**: Seasonal hiring surges (festive seasons) → thousands of temporary staff applications.
- **How Smart Recruit Helps**:
    - Bulk import candidate applications from job portals.
    - Auto-screen candidates based on availability/shift preference.
    - Dashboards for HR to track store-wise hiring progress.

4. Banking & Financial Services (HDFC, ICICI, Deloitte)

- **Challenge**: Strict compliance; need to hire employees with verified backgrounds.
- **How Smart Recruit Helps**:
    - Track application → background verification → final approval.
    - Automate alerts for missing compliance documents (PAN, Aadhaar, KYC).
    - Dashboards for branch-wise recruitment stats.

5. Manufacturing & Logistics (Tata Motors, DHL, Mahindra)

- **Challenge**: Large blue-collar workforce recruitment, distributed across multiple plants/warehouses.
- **How Smart Recruit Helps**:
  - Region-wise recruiter assignment.
  - Candidate mobile app → easy application process for workers.
  - SMS/email alerts for interview scheduling.

6. Education (Universities & EdTech like Byju's, Coursera)

- **Challenge**: Hiring large teaching/administrative staff during academic sessions.
- **How Smart Recruit Helps**:
  - Manage separate pipelines for faculty vs administrative roles.
  - Approval workflows with academic deans for faculty selection.
  - Reports on hiring time per department.

Phase 2: Org Setup & Configuration – Smart Recruiter ATS

In this phase, the Salesforce environment for the Smart Recruiter Applicant Tracking System (ATS) was prepared and configured. The objective was to establish a secure, well-structured, and scalable foundation before implementing business processes. This setup ensures that organizational details, user management, and security controls are aligned with the recruitment workflow.

Salesforce Edition

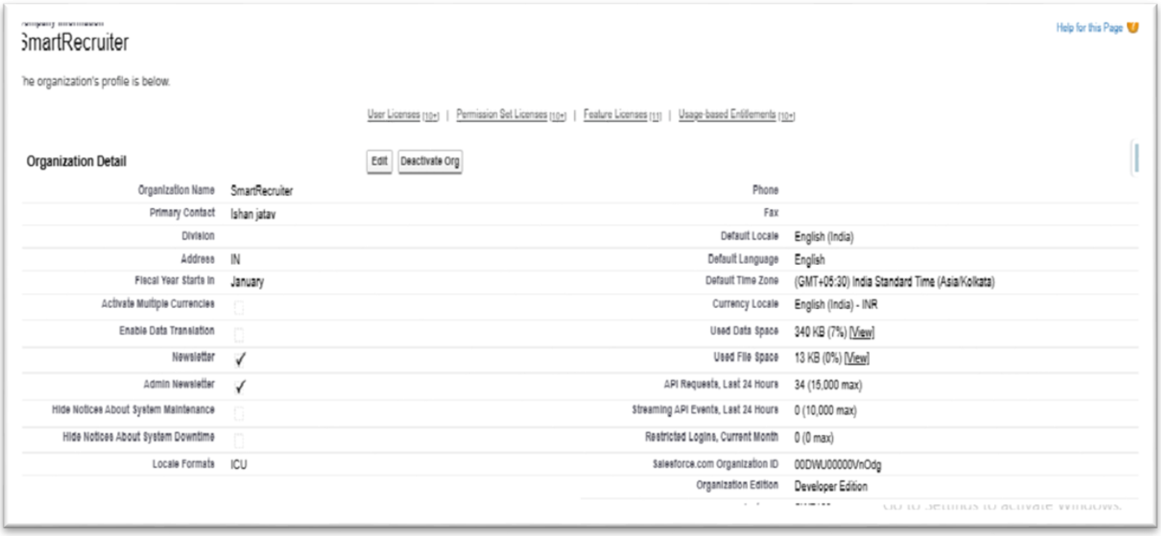
The project was developed using a **Salesforce Developer Edition Org**, which provides access to core features like Apex, automation tools, custom objects, and AppExchange apps. Although storage and user limits are restricted, this edition is suitable for proof-of-concept and academic projects.

Company Profile Setup

The company profile was configured with the following details:

- **Company Name**: Smart Recruiter
- **Default Locale**: English (India)
- **Currency**: INR
- **Time Zone**: Asia/Kolkata

This ensures that job postings, candidate data, and reports are aligned with the organization's region and currency standards.



SmartRecruiter

he organization's profile is below.

Help for this Page
User Licenses [10+] | Permission Set Licenses [10+] | Feature Licenses [11] | Usage-based Entitlements [10+]
Edit
Deactivate Org

Organization Detail

Organization Name	SmartRecruiter	Phone	
Primary Contact	Ishan jatav	Fax	
Division		Default Locale	English (India)
Address	IN	Default Language	English
Fiscal Year Starts In	January	Default Time Zone	(GMT+05:30) India Standard Time (Asia/Kolkata)
Activate Multiple Currencies	□	Currency Locale	English (India) - INR
Enable Data Translation	□	Used Data Space	340 KB (7%) [View]
Newsletter	✓	Used File Space	13 KB (0%) [View]
Admin Newsletter	✓	API Requests, Last 24 Hours	34 (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	00DWU00000VnOdg
		Organization Edition	Developer Edition

Business Hours & Holidays

Business hours were defined as **Monday to Friday, 9:00 AM – 6:00 PM**, reflecting typical HR operations. Public holidays in India were added for demonstration purposes. These settings support time-based workflows, such as escalation rules for pending approvals.

Fiscal Year Settings

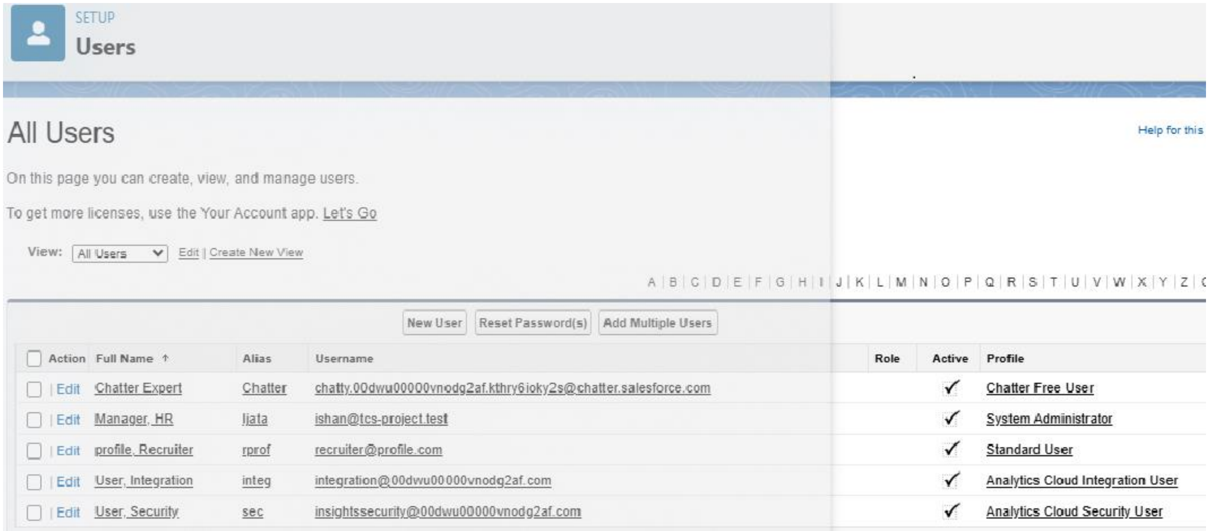
A **Standard Fiscal Year (April–March)** was enabled. This ensures that recruitment reports, such as hires per quarter or year, are synchronized with the organization's financial reporting cycle.

User Setup & Licenses

Sample users were created to represent real-world roles:

- **HR Manager** – manages approvals and oversees recruitment.
- **Recruiter** – manages job postings and candidate applications.
- **Candidate** – applies for jobs via portals or external submission.

Each user was assigned appropriate licenses and profiles to simulate practical scenarios.



SETUP
Users

All Users

Help for this

On this page you can create, view, and manage users.

To get more licenses, use the Your Account app. Let's Go

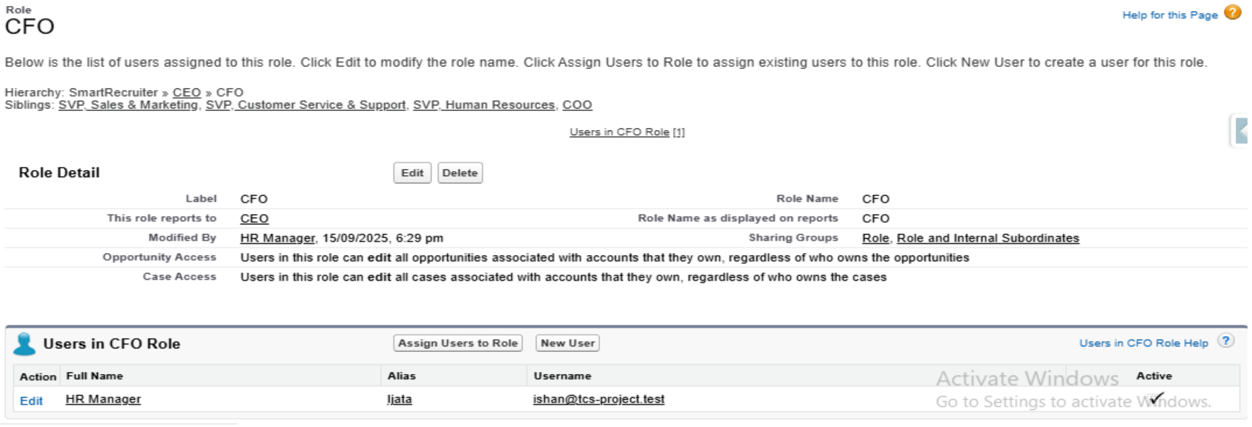
View: All Users Edit | Create New View

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
New User
Reset Password(s)
Add Multiple Users

□ Action	Full Name ↑	Alias	Username	Role	Active	Profile
□ | Edit	Chatter Expert	Chatter	chatty.00dwu00000vnodg2af.kthry6ioky2s@chatter.salesforce.com		✓	Chatter Free User
□ | Edit	Manager, HR	ljata	ishan@tcs-project.test		✓	System Administrator
□ | Edit	profile, Recruiter	rprof	recruiter@profile.com		✓	Standard User
□ | Edit	User, Integration	integ	integration@00dwu00000vnodg2af.com		✓	Analytics Cloud Integration User
□ | Edit	User, Security	sec	insightssecurity@00dwu00000vnodg2af.com		✓	Analytics Cloud Security User

Profiles, Roles, and Permission Sets

Profiles were customized to control access to objects and fields. Roles were defined hierarchically: **HR Director → HR Manager → Recruiter**. Permission Sets were created for granting additional privileges, such as access to reporting features. This setup ensures a balance between security and operational flexibility.



Role
CFO

Help for this Page

Below is the list of users assigned to this role. Click Edit to modify the role name. Click Assign Users to Role to assign existing users to this role. Click New User to create a user for this role.

Hierarchy: SmartRecruiter » CEO » CFO
Siblings: SVP, Sales & Marketing, SVP, Customer Service & Support, SVP, Human Resources, COO

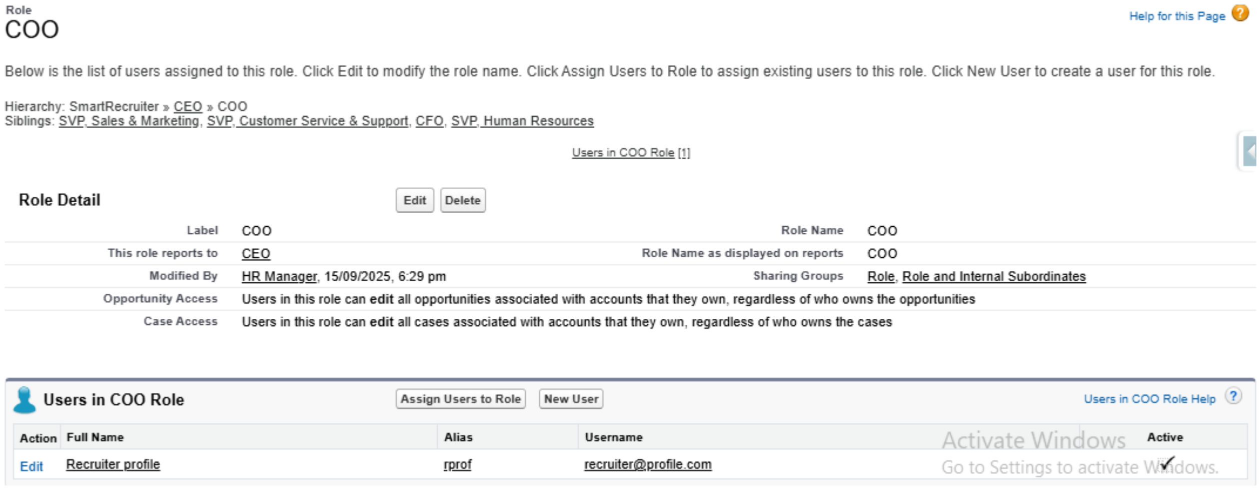
Users in CFO Role [1]

Role Detail

Label	CFO	Role Name	CFO
This role reports to	CEO	Role Name as displayed on reports	CFO
Modified By	HR Manager, 15/09/2025, 6:29 pm	Sharing Groups	Role, Role and Internal Subordinates
Opportunity Access	Users in this role can edit all opportunities associated with accounts that they own, regardless of who owns the opportunities		
Case Access	Users in this role can edit all cases associated with accounts that they own, regardless of who owns the cases		

Users in CFO Role

Action	Full Name	Alias	Username	Active
Edit	HR Manager	ljata	ishan@tcs-project.test	✓



Role
COO

Help for this Page

Below is the list of users assigned to this role. Click Edit to modify the role name. Click Assign Users to Role to assign existing users to this role. Click New User to create a user for this role.

Hierarchy: SmartRecruiter » CEO » COO
Siblings: SVP, Sales & Marketing, SVP, Customer Service & Support, CFO, SVP, Human Resources

Users in COO Role [1]

Role Detail

Label	COO	Role Name	COO
This role reports to	CEO	Role Name as displayed on reports	COO
Modified By	HR Manager, 15/09/2025, 6:29 pm	Sharing Groups	Role, Role and Internal Subordinates
Opportunity Access	Users in this role can edit all opportunities associated with accounts that they own, regardless of who owns the opportunities		
Case Access	Users in this role can edit all cases associated with accounts that they own, regardless of who owns the cases		

Users in COO Role

Action	Full Name	Alias	Username	Active
Edit	Recruiter profile	rprof	recruiter@profile.com	✓

Organization-Wide Defaults (OWD) and Sharing Rules

- **OWD** was set as follows:
  - Job Applications and Candidate records → Private
  - Job Postings → Public Read/Write
- **Sharing Rules** were implemented to allow recruiters from specific departments to collaborate on relevant applications.

This prevents unauthorized access to sensitive candidate data while enabling teamwork among HR staff.

Note -> I will be completing the OWD setup after creating my custom objects. And will establish sharing rules as per requirements.

Login Access Policies

Login restrictions were applied by IP ranges for administrators, while recruiters were granted trusted access for remote work. These measures strengthen system security.

Developer Org Setup & Sandbox Usage

The project was built on a **Developer Org**. For enterprise-level deployment, a sandbox strategy is recommended:

- **Developer Sandbox** → for building features.
- **UAT Sandbox** → for testing by HR staff.
- **Production Org** → for live usage.

Deployment Basics

Metadata and configurations were deployed using **Change Sets** and **Salesforce DX (SFDX) with VS Code**. A GitHub repository was also used for version control and collaboration, ensuring that project changes are tracked effectively.

Phase 3: Data Modelling & Relationships Report

1. Objective

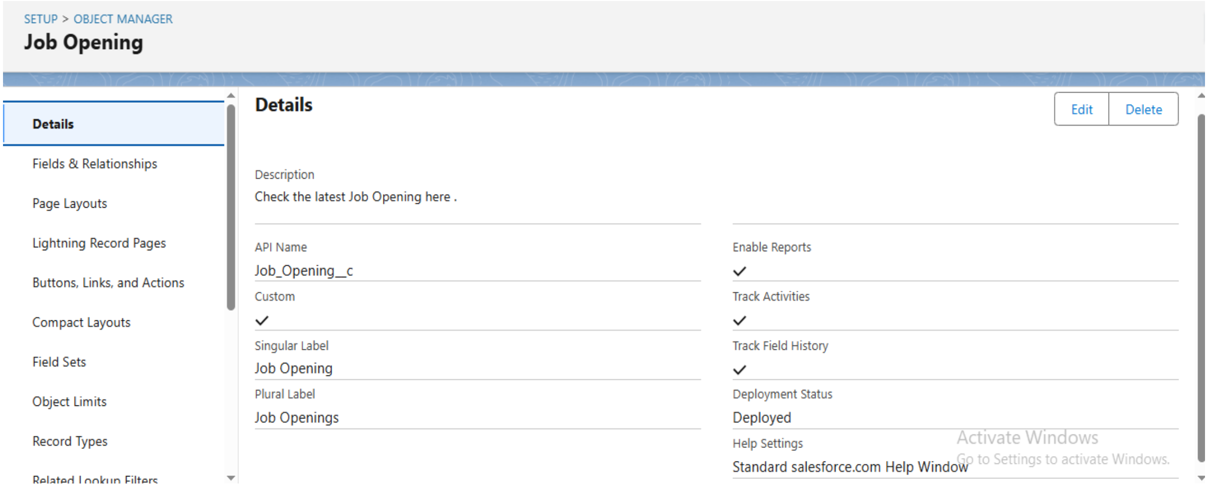
- To design the **data model** for Smart Recruiter that supports job openings, applications, interviews, and applicant tracking.

- To define relationships between objects, fields, page layouts, record types, and file handling.

2. Objects Created

List all the objects in my project the Smart Recruiter with description of each .

- **Job Opening** _c -> It is a custom object created for storing for the Job opening details from the different organisations.



Description
Check the latest Job Opening here .

API Name
Job_Opening__c

Custom
✓

Singular Label
Job Opening

Plural Label
Job Openings

Enable Reports
✓

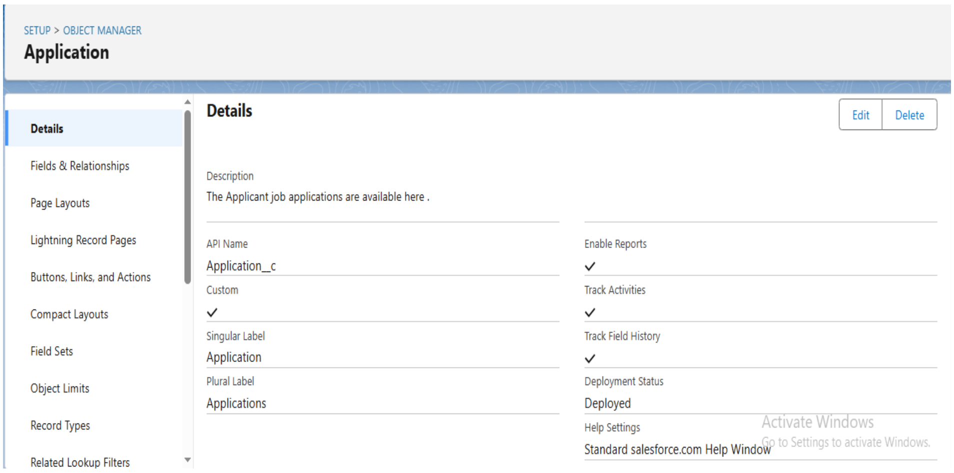
Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

- **Application _c ->** this is also a custom object that is used to store the applicant records that applied for the job opening.



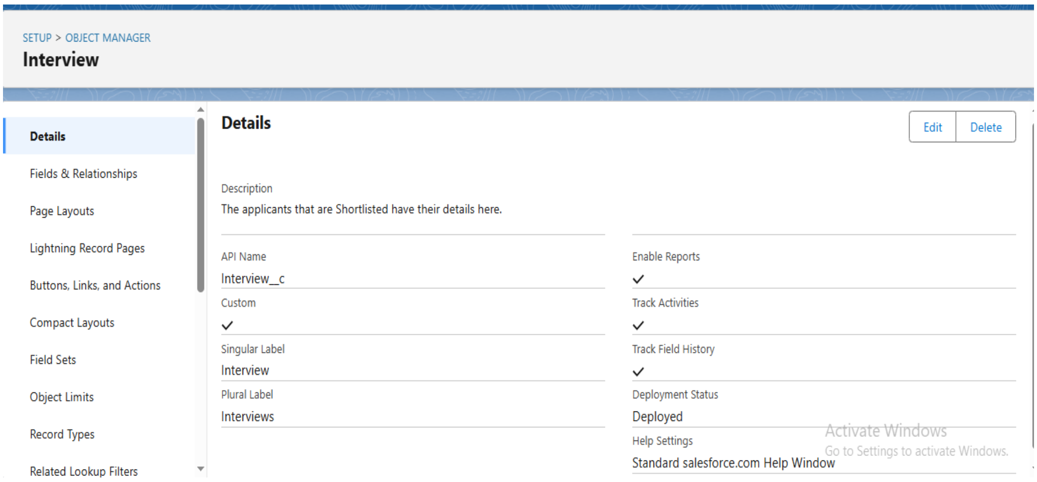
Application

Details

Description
The Applicant job applications are available here .

API Name Application__c	Enable Reports ✓
Custom ✓	Track Activities ✓
Singular Label Application	Track Field History ✓
Plural Label Applications	Deployment Status Deployed
	Help Settings Standard salesforce.com Help Window

- Interview -> This is a custom object that is used for storing scheduled interview details and track details for applicant shortlisted for interview.



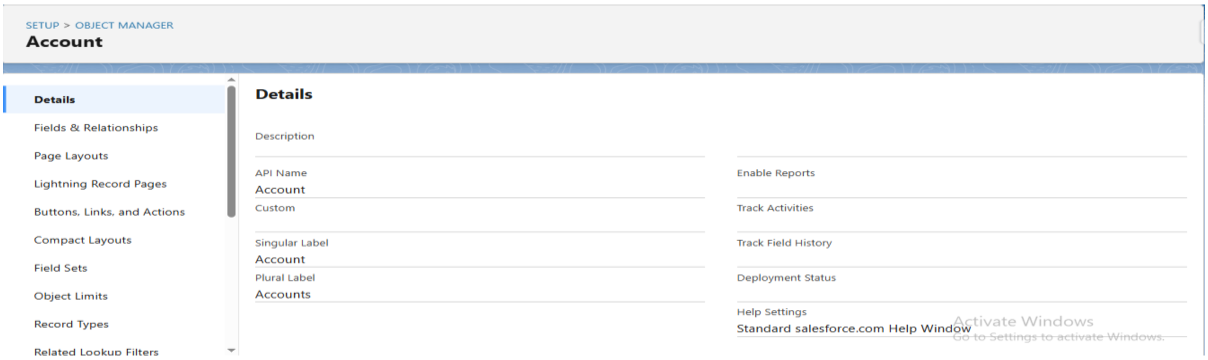
Interview

Details

Description
The applicants that are Shortlisted have their details here.

API Name Interview__c	Enable Reports ✓
Custom ✓	Track Activities ✓
Singular Label Interview	Track Field History ✓
Plural Label Interviews	Deployment Status Deployed
	Help Settings Standard salesforce.com Help Window

- Account -> This is a standard object that is used to store details of the organisations that has provided a job opening.



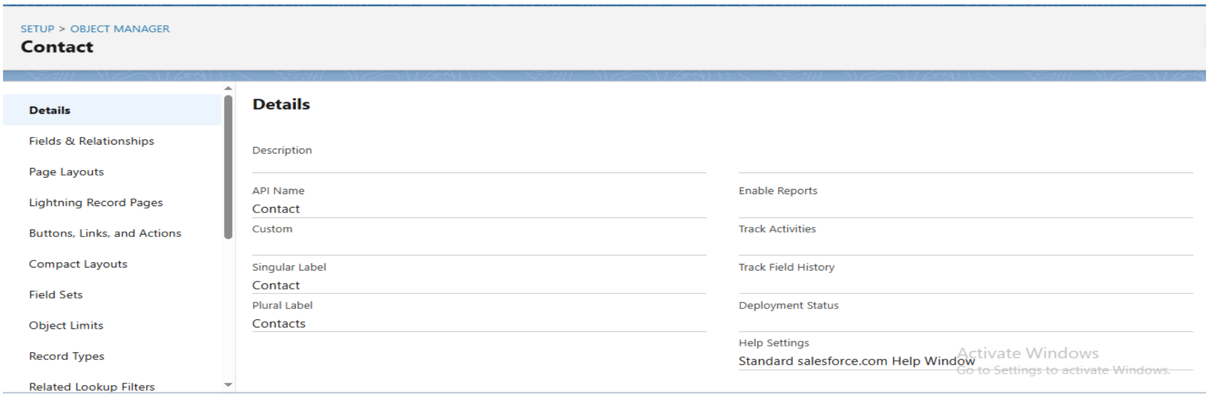
Account

Details

Description

API Name Account	Enable Reports
Custom	Track Activities
Singular Label Account	Track Field History
Plural Label Accounts	Deployment Status
	Help Settings Standard salesforce.com Help Window

- **Contact ->** this is a standard object that is used for storing application contact details.

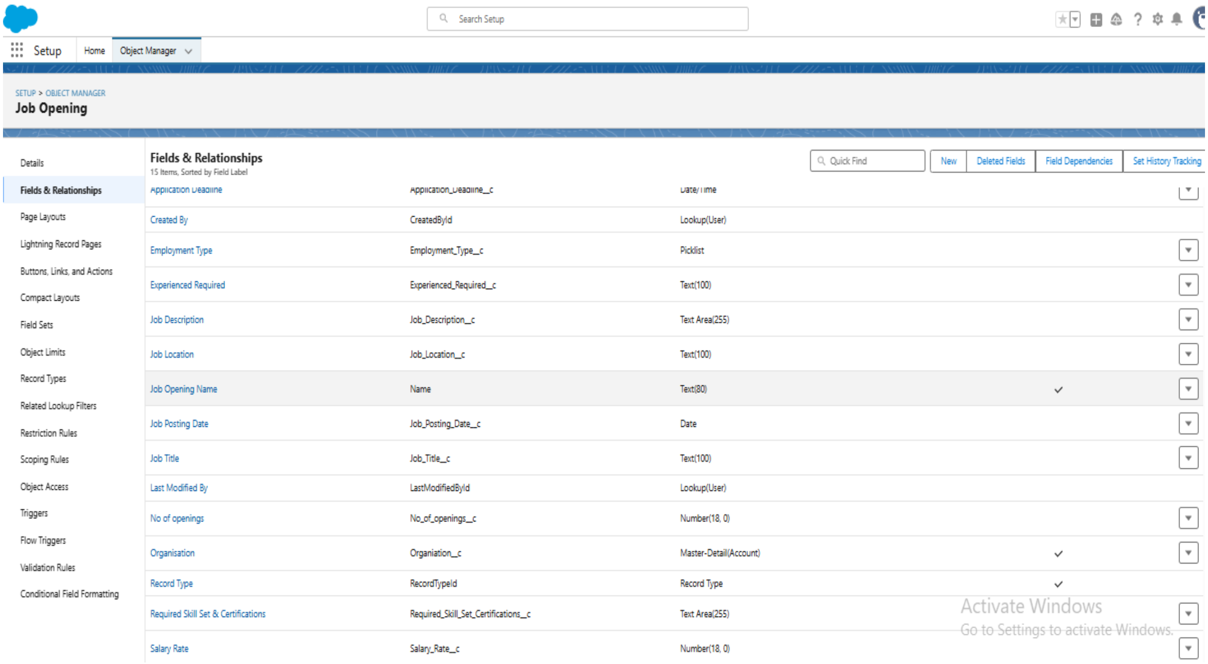


Details

Description	
API Name Contact	Enable Reports
Custom	Track Activities
Singular Label Contact	Track Field History
Plural Label Contacts	Deployment Status
	Help Settings Standard salesforce.com Help Window

3. Fields

- Job Opening Objects fields are shown in the image below.

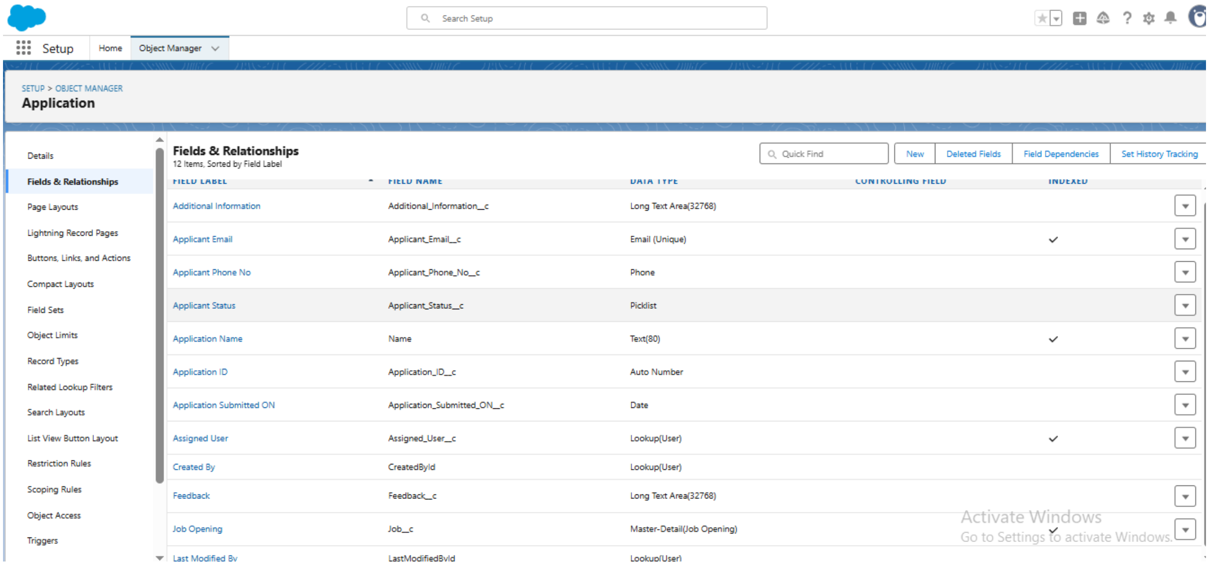


Fields & Relationships

15 Items, Sorted by Field Label

Application Deadline	Application_Deadline__c	Date/Time	
Created By	CreatedById	Lookup(User)	
Employment Type	Employment_Type__c	Picklist	
Experienced Required	Experienced_Required__c	Text(100)	
Job Description	Job_Description__c	Text Area(255)	
Job Location	Job_Location__c	Text(100)	
Job Opening Name	Name	Text(80)	✓
Job Posting Date	Job_Posting_Date__c	Date	
Job Title	Job_Title__c	Text(100)	
Last Modified By	LastModifiedById	Lookup(User)	
No of openings	No_of_openings__c	Number(18, 0)	
Organisation	Organiation__c	Master-Detail(Account)	✓
Record Type	RecordTypeId	Record Type	✓
Required Skill Set & Certifications	Required_Skill_Set_Certifications__c	Text Area(255)	
Salary Rate	Salary_Rate__c	Number(18, 0)	

- Application Object Fields are shown in the image below.



SETUP > OBJECT MANAGER

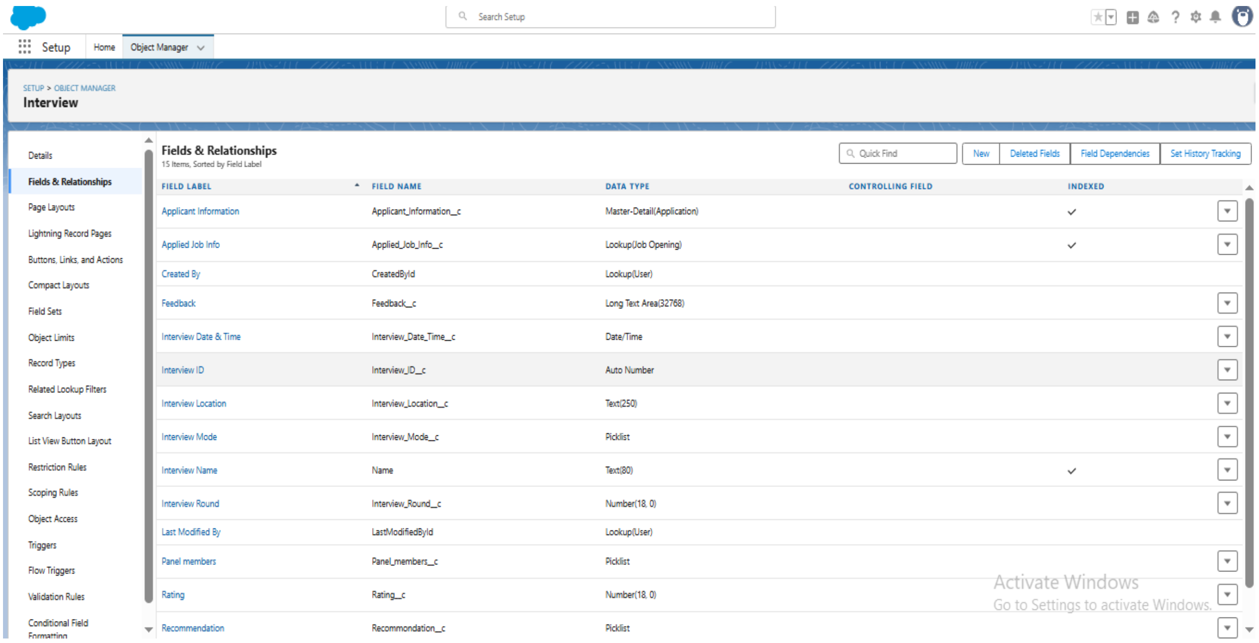
Application

Fields & Relationships

12 Items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Additional Information	Additional_Information__c	Long Text Area(32768)		
Applicant Email	Applicant_Email__c	Email (Unique)		✓
Applicant Phone No	Applicant_Phone_No__c	Phone		
Applicant Status	Applicant_Status__c	Picklist		
Application Name	Name	Text(80)		✓
Application ID	Application_ID__c	Auto Number		
Application Submitted ON	Application_Submitted_ON__c	Date		
Assigned User	Assigned_User__c	Lookup(User)		✓
Created By	CreatedById	Lookup(User)		
Feedback	Feedback__c	Long Text Area(32768)		
Job Opening	Job__c	Master-Detail(Job Opening)		✓
Last Modified By	LastModifiedById	Lookup(User)		

- Interview Objects Fields are shown in the image below.



SETUP > OBJECT MANAGER

Interview

Fields & Relationships

15 Items, Sorted by Field Label

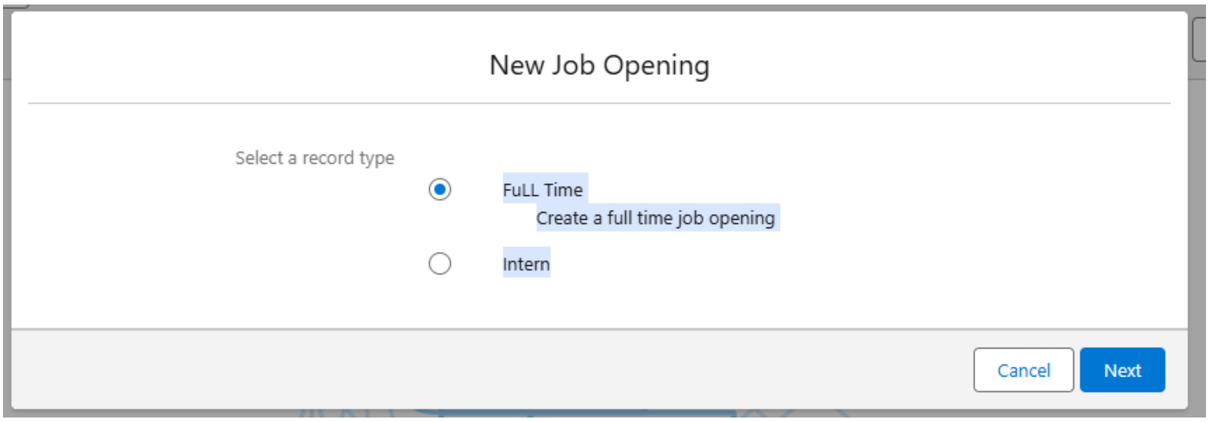
FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Applicant Information	Applicant_Information__c	Master-Detail(Application)		✓
Applied Job Info	Applied_Job_Info__c	Lookup(Job Opening)		✓
Created By	CreatedById	Lookup(User)		
Feedback	Feedback__c	Long Text Area(32768)		
Interview Date & Time	Interview_Date_Time__c	Date/Time		
Interview ID	Interview_ID__c	Auto Number		
Interview Location	Interview_Location__c	Text(250)		
Interview Mode	Interview_Mode__c	Picklist		
Interview Name	Name	Text(80)		✓
Interview Round	Interview_Round__c	Number(18, 0)		
Last Modified By	LastModifiedById	Lookup(User)		
Panel members	Panel_members__c	Picklist		
Rating	Rating__c	Number(18, 0)		
Recommendation	Recommondation__c	Picklist		

- Account and contacts are standard objects therefore their fields are present by default in the salesforce.

4. Record Types

- **Job Opening**:
  - Full-Time – A record type for storing full-time job opening details.

- ○ Internship-A record type for storing Internship job opening details
- ○ Each record type has its **own page layout** (Full-Time Layout, Internship Layout).



New Job Opening

Select a record type

● FuLL Time
Create a full time job opening

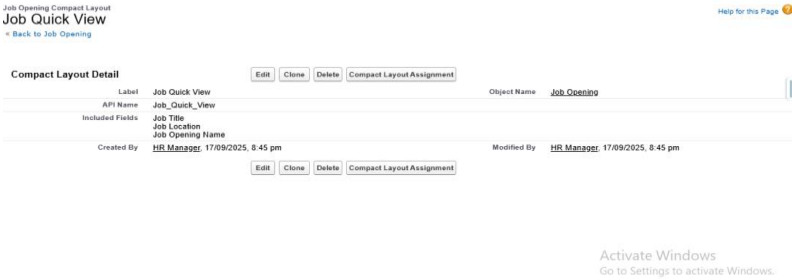
○ Intern

5. Page Layouts

- **Job Opening**: Single layout showing details required for the Job Opening.
- **Application**: Single layout showing applicant info, resume, job applied, and status.
- **Interview** Layout shows interview details essential for scheduling an interview.

6. Compact Layouts

- **Job Opening Compact Layout**: Job Title, Location, Status.



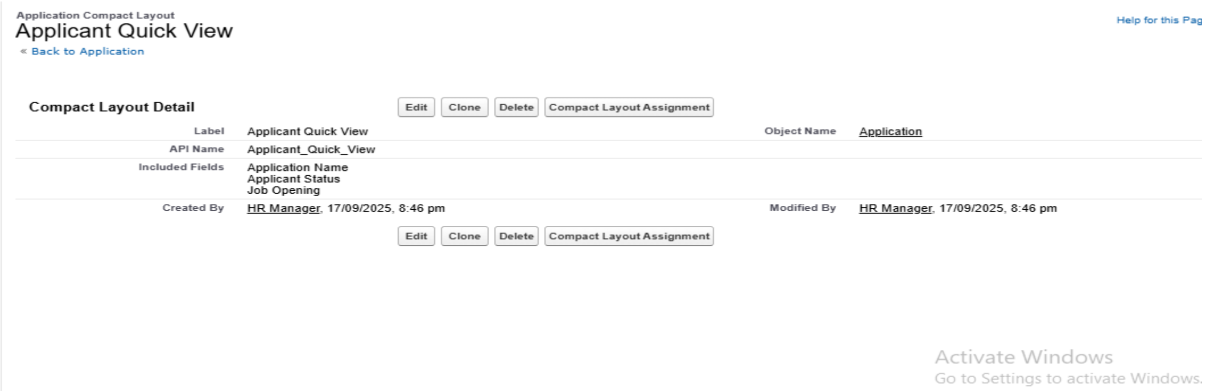
Job Opening Compact Layout
Job Quick View
« Back to Job Opening

Compact Layout Detail

Label	Job Quick View	Object Name	Job Opening
API Name	Job_Quick_View		
Included Fields	Job Title Job Location Job Opening Name		
Created By	HR Manager, 17/09/2025, 8:45 pm	Modified By	HR Manager, 17/09/2025, 8:45 pm

Help for this Page
Edit
Clone
Delete
Compact Layout Assignment
Activate Windows
Go to Settings to activate Windows.

- **Application Compact layout:** Applicaiton name, Application status etc.



Application Compact Layout

Applicant Quick View

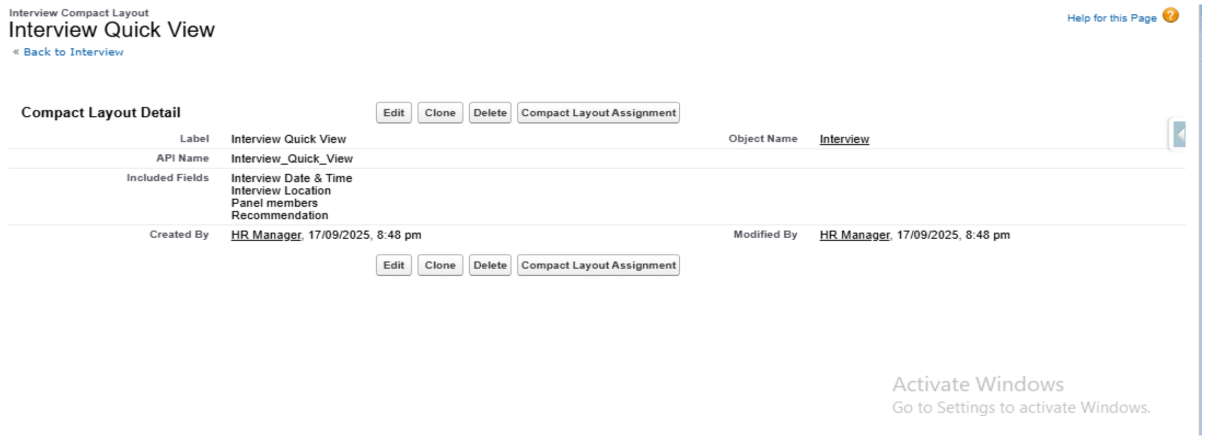
« Back to Application

Help for this Pag

Compact Layout Detail

Label	Applicant Quick View	Object Name	Application
API Name	Applicant_Quick_View		
Included Fields	Application Name Applicant Status Job Opening		
Created By	HR Manager, 17/09/2025, 8:46 pm	Modified By	HR Manager, 17/09/2025, 8:46 pm

- **Quick Interview Compact layout**: Interview Date & Time, Recommendation Interview Location



Interview Compact Layout

Interview Quick View

« Back to Interview

Help for this Page

Compact Layout Detail

Label	Interview Quick View	Object Name	Interview
API Name	Interview_Quick_View		
Included Fields	Interview Date & Time Interview Location Panel members Recommendation		
Created By	HR Manager, 17/09/2025, 8:48 pm	Modified By	HR Manager, 17/09/2025, 8:48 pm

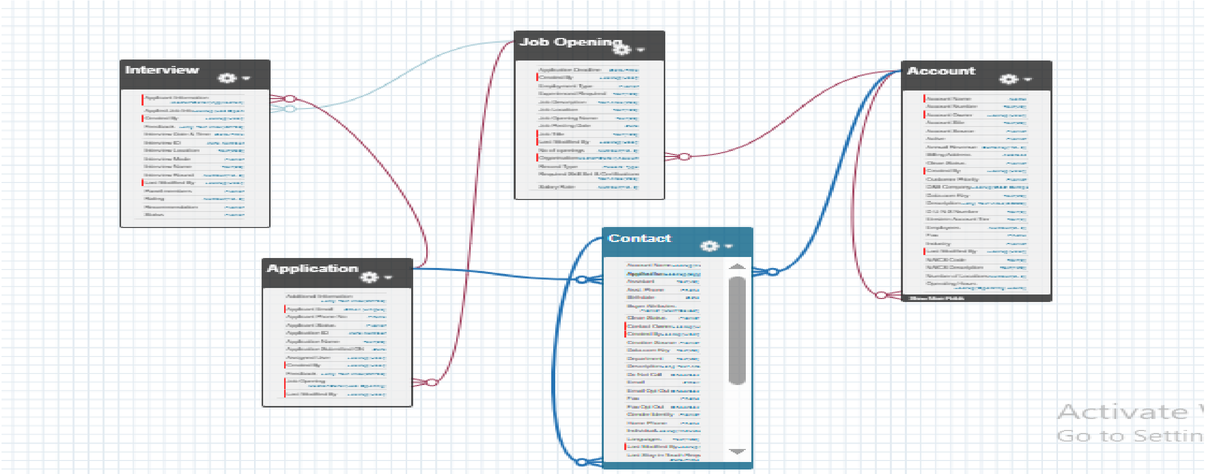
7. Object Relationships

Include **all relationships between all the objects used by me till now .**

Parent Object	Child Object	Relationship Type	Notes
Account	JobOpening_c	Master-Detail	Deleting Account deletes all Jobs
JobOpening_c	Application_c	Master-Detail	Applications tied to Job
Application_c	Interview_c	Master-Detail	Interviews tied to Application
Application_c	Contact	Lookup	Applicant can apply to multiple jobs

6. Schema Builder

- Use Schema Builder to **visualize relationships** between objects.
- Example: One **Job Opening _c** has many **Application__c** records.



8. Junction Objects

- Junction object is an object that is used for creating many to many relationships between two objects.
- I Have not used the Junction Object in my Project I may be showing it while working in the future scope.

9. External Objects

- Suppose Smart Recruiter wants to pull **job data from a third-party system (like LinkedIn/Indeed). I** Have Planned this idea as my future scope.

Phase 4: Process Automation (Admin)

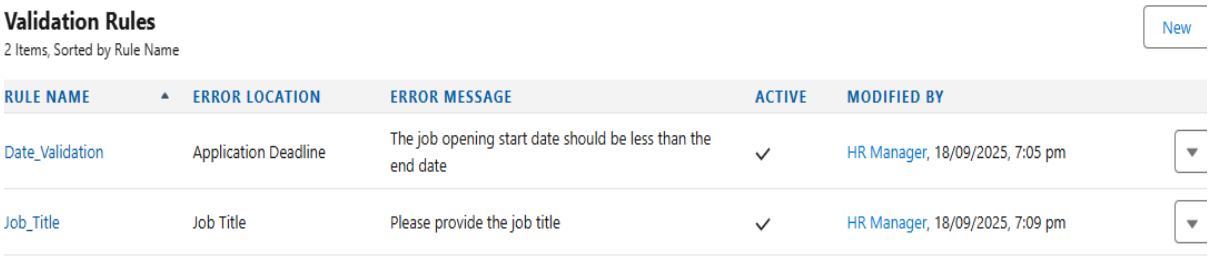
Overview

In this phase, various Salesforce automation tools were implemented to streamline the recruitment process in the Smart Recruiter ATS system. The goal was to reduce manual effort, improve communication with candidates, and ensure consistent approval and notification processes.

- ## Validation Rules

  Ensured data quality and mandatory field entry.

Validation rules on Job Opening

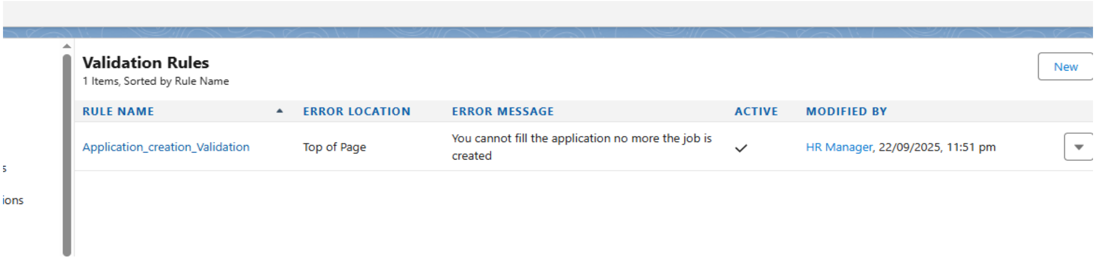


Validation Rules

2 Items, Sorted by Rule Name

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Date_Validation	Application Deadline	The job opening start date should be less than the end date	✓	HR Manager, 18/09/2025, 7:05 pm
Job_Title	Job Title	Please provide the job title	✓	HR Manager, 18/09/2025, 7:09 pm

Validation Rule on Application Object - This rule states that no more applications can be created if the job opening last date is smaller than the day the application was submitted.



Validation Rules

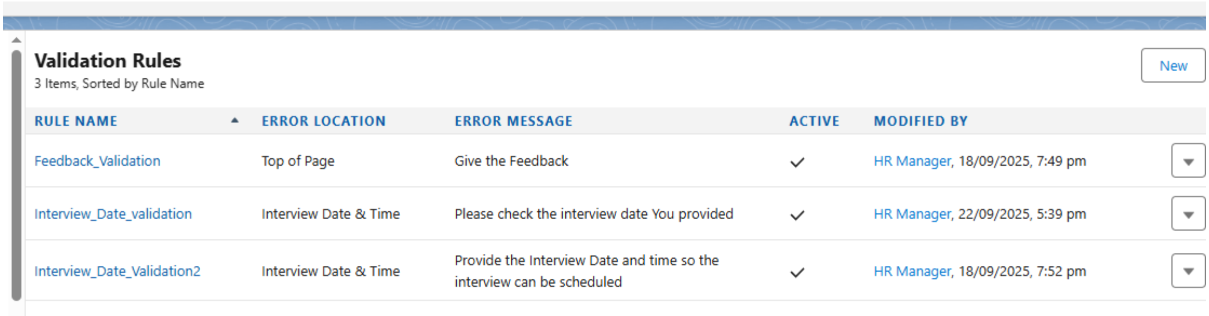
1 Items, Sorted by Rule Name

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Application_creation_Validation	Top of Page	You cannot fill the application no more the job is created	✓	HR Manager, 22/09/2025, 11:51 pm

Validation rules on interview object

- **Feedback validation –** after the interview is being completed feedback just be given.

- **INTERVIEW Date Validation** – the date on which the interview i being scheduled should be greater than today.



Validation Rules

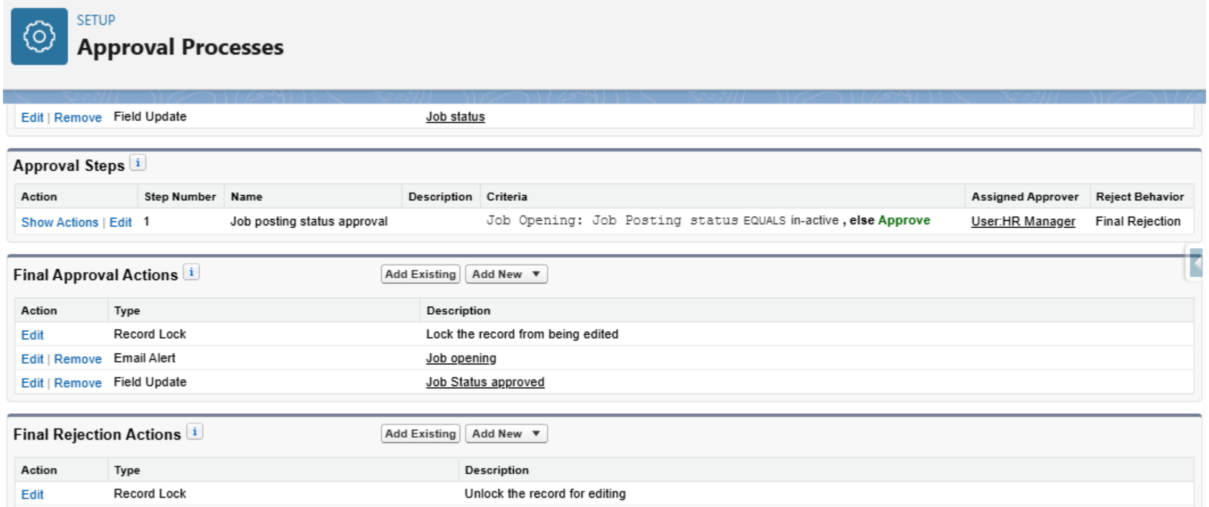
3 Items, Sorted by Rule Name

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Feedback_Validation	Top of Page	Give the Feedback	✓	HR Manager, 18/09/2025, 7:49 pm
Interview_Date_validation	Interview Date & Time	Please check the interview date You provided	✓	HR Manager, 22/09/2025, 5:39 pm
Interview_Date_Validation2	Interview Date & Time	Provide the Interview Date and time so the interview can be scheduled	✓	HR Manager, 18/09/2025, 7:52 pm

Approval Process

- Job Opening - As we know the recruiter cannot post the job opening on this own decision. They need to take approval from Hr manager. Through this approval process it is possible.

- When recruiter creates job opening the job opening status remains inactive once the approval is made from the HR the status becomes active for the job opening



SETUP

Approval Processes

Edit | Remove	Field Update	Job status	

Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions | Edit	1	Job posting status approval		Job Opening: Job Posting status EQUALS in-active , else Approve	User:HR Manager	Final Rejection

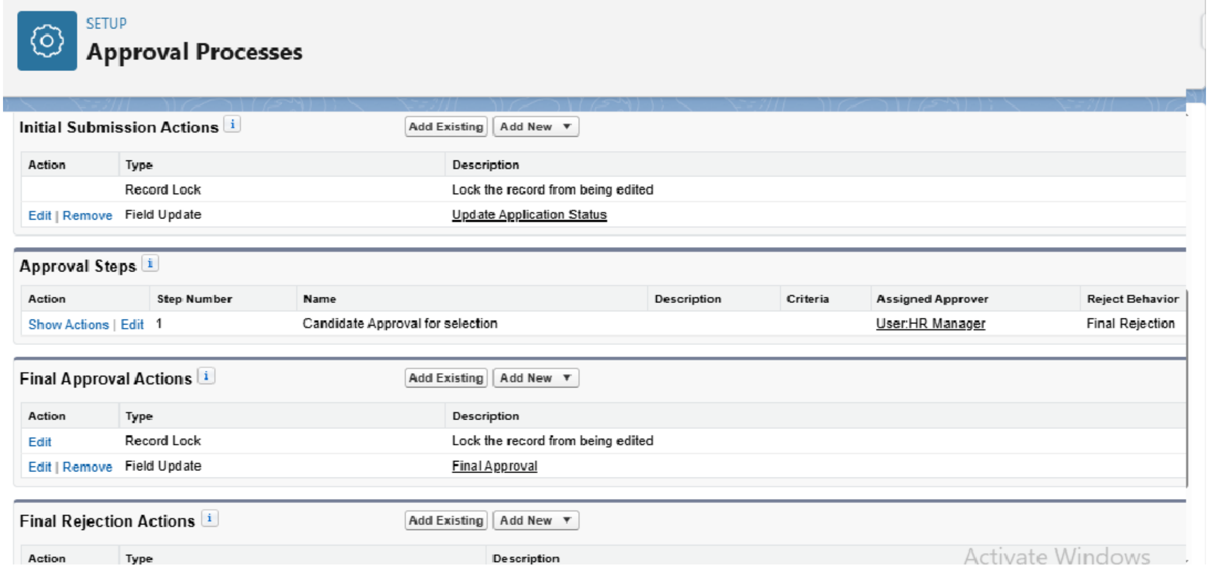
Final Approval Actions

Action	Type	Description
Edit	Record Lock	Lock the record from being edited
Edit | Remove	Email Alert	Job opening
Edit | Remove	Field Update	Job Status approved

Final Rejection Actions

Action	Type	Description
Edit	Record Lock	Unlock the record for editing

- **Application Selection -** When a candidate is being shortlisted the candidate detail are sent the HR for hiring approval then the status is marked as selected.



SETUP

Approval Processes

Initial Submission Actions

Action	Type	Description
	Record Lock	Lock the record from being edited
Edit | Remove	Field Update	Update Application Status

Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions | Edit	1	Candidate Approval for selection			User:HR Manager	Final Rejection

Final Approval Actions

Action	Type	Description
Edit	Record Lock	Lock the record from being edited
Edit | Remove	Field Update	Final Approval

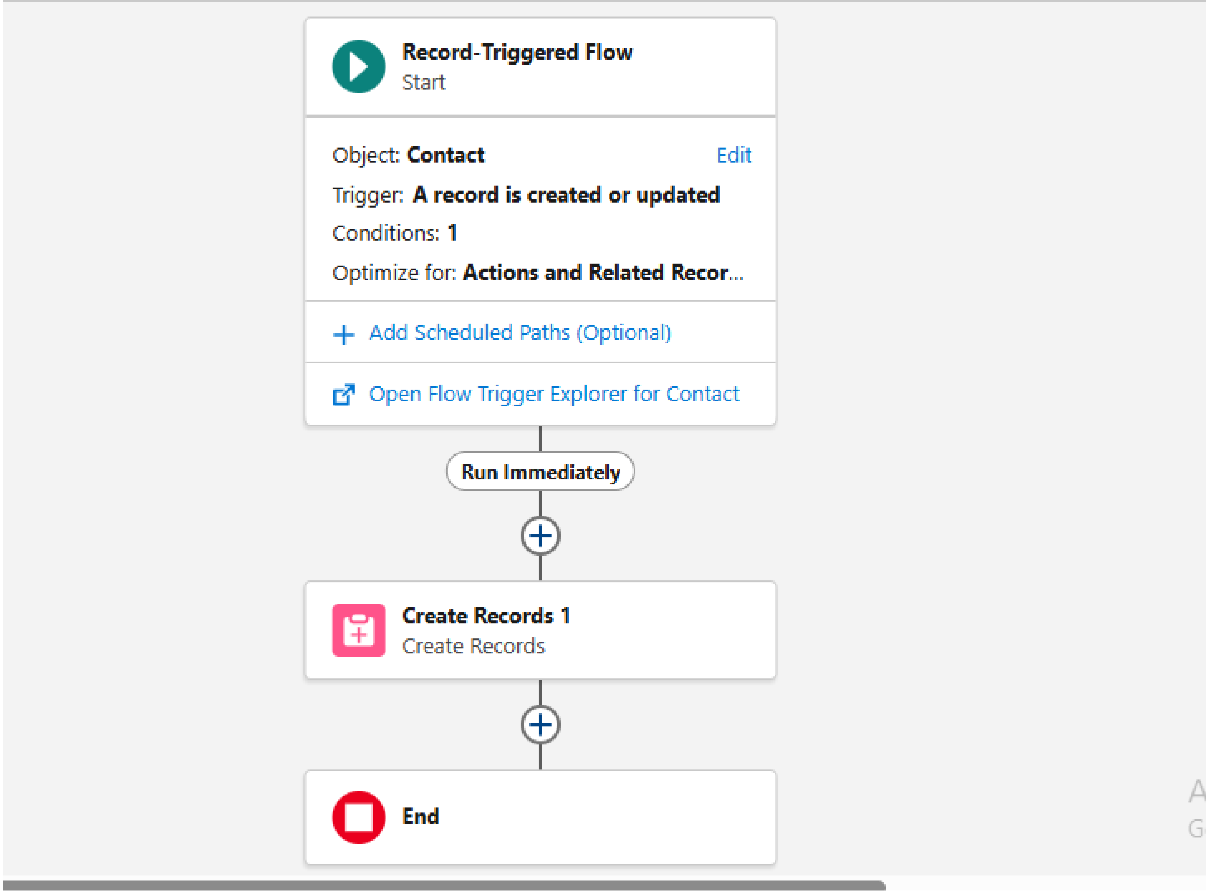
Final Rejection Actions

Action	Type	Description

Flows

- Record-Triggered Flows

- Auto-create Application records when an Applicant submits details.



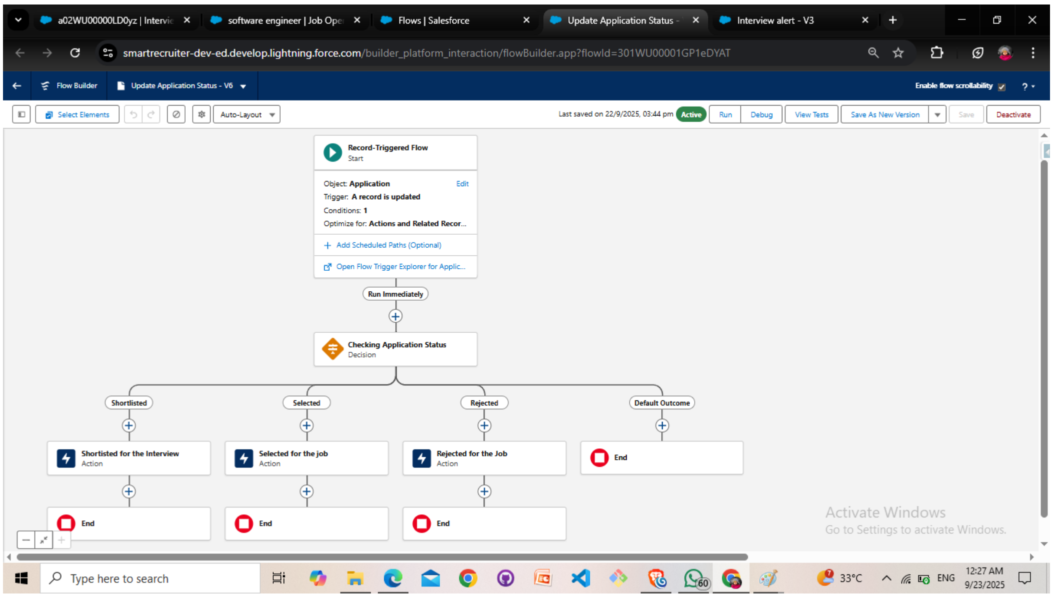
Object: **Contact**

Trigger: **A record is created or updated**

Conditions: **1**

Optimize for: **Actions and Related Recor...**

- Update Application status changes and send notifications to Candidates/Recruiters.



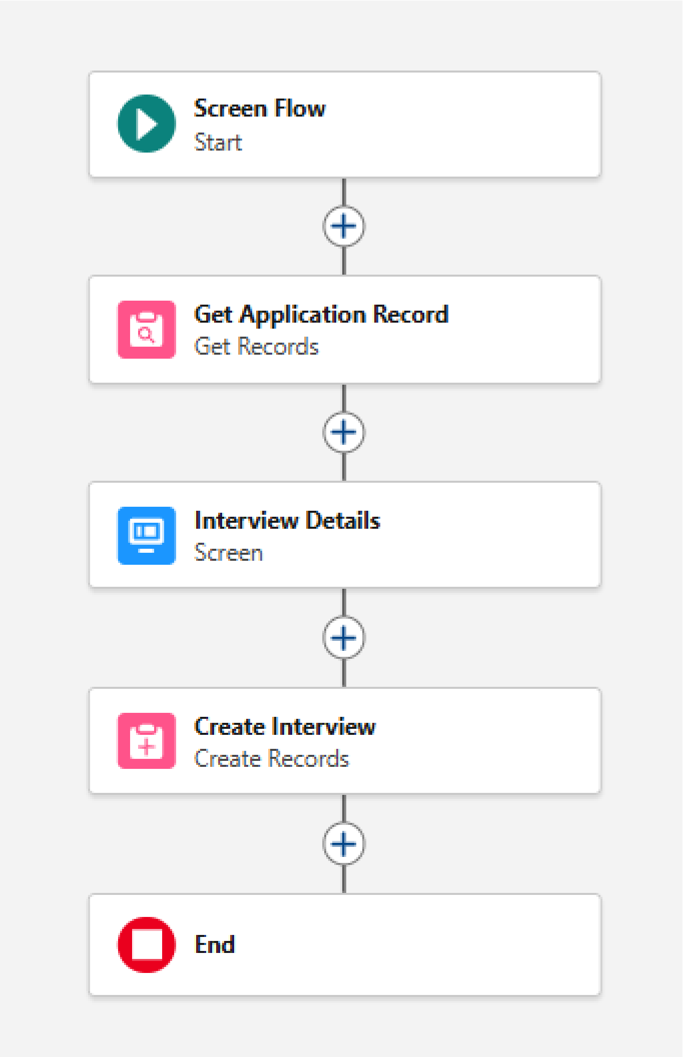
smartrecruiter-dev-ed.develop.lightning.force.com/builder_platform_interaction/flowBuilder.app?flowId=301WU00001GP1eDYAT

Last saved on 22/9/2025, 03:44 pm

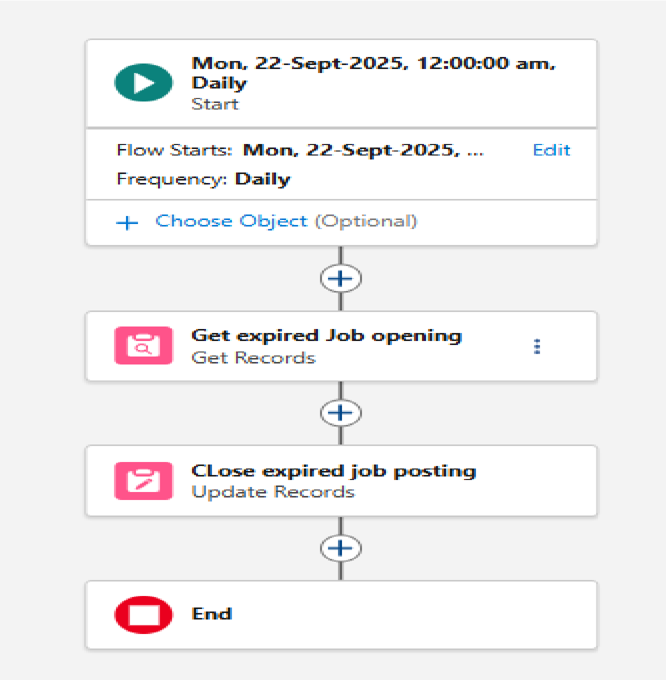
Object: Application
Trigger: A record is updated
Conditions: 1
Optimize for: Actions and Related Recor...

- <u>Screen Flows</u>

  - Interview Scheduling: Recruiters can schedule interviews with date/time, interviewer selection, and mode (online/offline).



- Scheduled Flows
  - Job Posting Auto Close: Automatically closes job postings past their deadline.



Mon, 22-Sept-2025, 12:00:00 am, Daily
Start

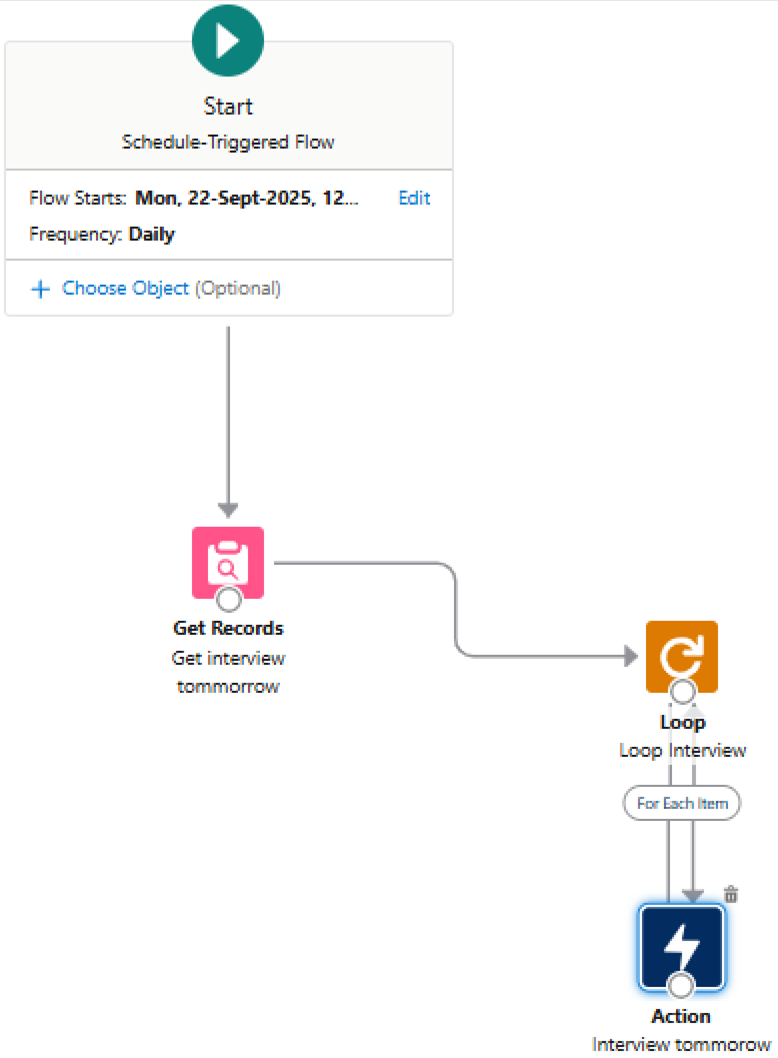
Flow Starts: **Mon, 22-Sept-2025, ...**
Frequency: **Daily**

Get expired Job opening
Get Records

CLose expired job posting
Update Records

End

- Interview Reminder: Sends email/SMS reminders to Applicant and Recruiters one day before the interview.



Flow Starts: **Mon, 22-Sept-2025, 12...**

Frequency: **Daily**

Email Alerts

- Automated emails to Candidates and Recruiters at key events:
  - Application status updates (Shortlisted, Interview Scheduled, Rejected).
  - Interview reminders – emails ae being sent before the interview to both the interviewer and applicant selected for the interview

Field Updates

- Automated updates of record fields based on flow conditions, e.g., updating `Application Status` when HR approves/rejects.

Tasks

- Automatically created tasks for recruiters when interview is scheduled, ensuring follow-ups.

Custom Notifications

- Real-time notifications to users inside Salesforce for important events such as:
  - Interview scheduled, mode of notifications are emails.
  - Application approved or rejected.

.

Phase 5: Apex Programming (Developer)

In Phase 5, I focused on **Apex programming concepts** in Salesforce to add backend business logic, automation, and asynchronous processing to the Job Portal project. Below are the details of the concepts I implemented along with scenarios.

1. Classes & Objects

Explanation:
In Apex, classes are templates that define objects, their attributes, and methods. Objects are instances of classes. They help in organizing code, applying reusability, and implementing business logic.

Scenario:
I created an Apex class `Job Application Handler` to manage operations related to job applications, such as validating applicant details and assigning interviewers. For example, when a new applicant record is created, the class methods are used to check eligibility before saving.

2. Apex Triggers (Before/After Insert/Update)

Explanation:
Triggers are used to perform actions automatically before or after DML (Data Manipulation Language) operations like insert, update, or delete.

Scenario:

- **Before Insert:** Prevented duplicate job applications for the same position by the same candidate.
- **After Insert:** Sent an automatic notification to HR after a job application was submitted.

- **Appication Prevent duplicate Handler this** apex trigger helps in preventing duplicate records of applicant for the same contact and job opening.

```
public class Application_Trigger_Handler {

public static void preventDuplicateApplications(List
newApps) {

   //  Collect all Contact and Job Ids from
the incoming records
    Set<Id> contactIds = new Set<Id>();
    Set<Id> jobIds = new Set<Id>();

    for (Application__c app : newApps) {
        if (app.Contact__c != null) {
            contactIds.add(app.Contact__c);
        }
        if (app.Job__c != null) {
            jobIds.add(app.Job__c);
        }
    }
    //  Query existing Applications with
those with  Contact and Job combinations
    List<Application__c> existingApps = [
        SELECT Id, Contact__c, Job__c
        FROM Application__c
        WHERE Contact__c IN :contactIds
        AND Job__c IN :jobIds
    ];

    //  Build a set of existing keys
(ContactId + JobId)
    Set<String> existingKeys = new
Set<String>();
    for (Application__c app : existingApps) {
        existingKeys.add(app.Contact__c + '-'
+ app.Job__c);
    }

    //  Compare with new records → block
duplicates
    for (Application__c app : newApps) {
        String key = app.Contact__c + '-' +
app.Job__c;
        if (existingKeys.contains(key)) {
            app.addError('This candidate has
already applied for this job posting.');
        }
    }
```

```
}

}
```

Application Prevent Duplicate Trigger

```
trigger Applicaiton_Trigger on Application__c (before insert)

 { if (Trigger.isBefore && Trigger.isInsert)
{ Application_Trigger_Handler.preventDuplicateApplications(Trigger.new); }

 }
```

3. Trigger Design Pattern

Explanation:
 The Trigger Design Pattern ensures that triggers are clean, scalable, and maintainable. Business logic is separated into handler classes instead of writing directly inside the trigger.

Scenario:

For the `Application__c` object, instead of writing all logic inside the trigger, I created `ApplicationTriggerHandler` class which handled validations, notifications, and updates. The trigger simply called the handler methods, making it reusable and cleaner.

- **<u>Create Application from contact created and Existing Job Opening </u> - this** creates application automatically when a contact associated with a job opening is being created.

```
public class Application_Trigger_Handler_1 {

// Method to create Applications from
Contacts who applied

public static void
createApplicationsFromContacts(List<Contact>
newContacts) {

    List<Application__c> appsToCreate = new
List<Application__c>();

    //: Loop through Contacts
    for (Contact c : newContacts) {
        // Only create Application if
Job_Posting__c is filled
        if (c.Job_Opening__c != null) {

            // : Prevent duplicate
Application for same Contact + Job
            List<Application__c> existingApps
= [
                SELECT Id FROM Application__c
                WHERE Contact__c = :c.Id
                AND    Job__c
= :c.Job_Opening__c
            ];
            if (existingApps.isEmpty()) {
                Application__c app = new
Application__c();
                app.Contact__c = c.Id;
                app.Job__c =
c.Job_Opening__c;
                app.Applicant_Status__c =
'Applied';
                appsToCreate.add(app);
            }
        }
    }

    //  Insert Applications
    if (!appsToCreate.isEmpty()) {
        insert appsToCreate;
    }
}

}
```

- **Application Status Handler ->** whenever the application status is updated to shortlisted then a task is created and is assigned to the recruiter who will be taking the interview as a notification about the interview .

public class Application_Status_Trigger_Hander {

```
// Method to create Task when Application
status changes
public static void
createTaskOnStatusChange(List<Application__c>
newApps, Map<Id, Application__c> oldMap) {

    List<Task> tasksToCreate = new
List<Task>();

    for (Application__c app : newApps) {

        // Compare old vs new status to
detect change
        Application__c oldApp =
oldMap.get(app.Id);
        if (oldApp.Applicant_Status__c!=
app.Applicant_Status__c &&
app.Applicant_Status__c== 'shortlisted'  &&
app.Assigned_User__c != null) {
            Task t = new Task();
            t.Subject = 'Follow up on
shortlisted Application';
            t.WhatId = app.Id; // Related to
Application
            t.OwnerId = app.Assigned_User__c;
// Assign to recruiter (replace with your
field API name)
            t.Status = 'Not Started';
            t.Priority = 'High';
            t.Description = 'The application
has been approved. Follow up with the
candidate.';
            tasksToCreate.add(t);
        }
    }

    if (!tasksToCreate.isEmpty()) {
        insert tasksToCreate;
    }
}
```

```
}
```

Application status Trigger

```
trigger Application_status_trigger on Application__c (after
update) {

// Call handler method, pass Trigger.new and
Trigger.oldMap

Application_Status_Trigger_Hander.createTaskOnStatusChange(Trigger.new, Trigger.oldMap); }
```

Contact trigger

```
trigger Contact_Trigger_1 on Contact (after insert, after
update) {

List<Contact> contactsWithJob = new
List<Contact>();

// Step 1: Loop through inserted/updated
contacts
for (Contact c : Trigger.new) {
    if (c.Job_Opening__c != null) { //
replace with your actual field API name
        contactsWithJob.add(c);
    }
}

// Step 2: Call handler to create Applications
if (!contactsWithJob.isEmpty()) {

Application_Trigger_Handler_1.createApplicationsFromContacts(contactsWithJob);
}

}
```

4. SOQL & SOSL

Explanation:

- **SOQL (Salesforce Object Query Language):** Used to fetch records from Salesforce objects based on conditions.
- **SOSL (Salesforce Object Search Language):** Used to perform text-based searches across multiple objects.

Scenario:

- SOQL was used to fetch all applications for a given candidate (`SELECT Id, Status FROM Application__c WHERE Candidate__c = :candidateId`).
- SOSL was used to search applicant details (like email/phone) across objects when HR wanted to quickly find a candidate.

5. Collections: List, Set, Map

Explanation:
Collections are data structures used to store multiple records.

- **List:** Ordered collection allowing duplicates.
- **Set:** Unordered collection without duplicates.
- **Map:** Key-value pairs for quick lookups.

Scenario:

- **List:** Used to store all interview records for a particular application.
- **Set:** Used to store unique candidate emails to prevent duplicates.
- **Map:** Used to map `Application Id → Interview Date` for quick access in bulk processing.

6. Control Statements

Explanation:
Control statements like `if-else`, `for`, `while`, and `switch` are used to apply decision-making and looping logic.

Scenario:
When assigning an interviewer, I used control statements:

- If the application status is "Interview Scheduled", then assign an interviewer.
- Else if the status is "Rejected", mark the application as closed.

12. Test Classes

Explanation:
Test classes are written to verify that Apex code works correctly and to meet Salesforce's requirement of 75% code coverage for deployment.

Scenario:
For each trigger and class, I wrote test classes such as `TestApplicationHandler` which tested:

- Creating a valid application

- Preventing duplicate applications
- Scheduling interviews

This ensured that all logic worked as expected before deployment.

```
@IsTest
public class ATS_TestClass {

    // Utility method to create a Contact
    private static Contact createContact() {
        Contact c = new Contact(
            LastName = 'Test Candidate',
            Email = 'testcandidate@example.com'
        );
        insert c;
        return c;
    }

    // Utility method to create a Job Posting
    private static Job_Opening__c createJobPosting() {
        Job_Opening__c job = new Job_Opening__c(
            Name = 'Software Engineer'
        );
        insert job;
        return job;
    }

    // Utility method to create an Application
    private static Application__c createApplication(Id contactId, Id jobId,
String statusVal) {
        Application__c app = new Application__c(
            Contact__c = contactId,
            Job__c = jobId,
           Applicant_Status__c = statusVal
        );
        insert app;
        return app;
    }

    // Utility method to create Applicant Info + Interview
    private static Interview__c createInterview(Id appInfoId, Datetime
slotTime) {
        Interview__c interview = new Interview__c(
            Applicant_Information__c = appInfoId,
            Interview_Date_Time__c = slotTime
        );
        insert interview;
        return interview;
    }

    // ----------------------------
    // TEST CASES
    // ----------------------------

    @IsTest
    static void testDuplicateApplicationPrevention() {
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();

        // Insert first application
        Application__c app1 = createApplication(c.Id, job.Id, 'Applied');

        // Try inserting duplicate application
        Application__c app2 = new Application__c(
            Contact__c = c.Id,
            Job__c = job.Id,
           Applicant_Status__c= 'Applied'
        );

        Test.startTest();
        try {
            insert app2;
            System.assert(false, 'Duplicate should not be inserted');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('duplicate'), 'Should block
duplicate application');
        }
        Test.stopTest();
    }

    @IsTest
    static void testTaskCreationOnApprovedApplication() {
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();

        Application__c app = createApplication(c.Id, job.Id, 'Applied');

        Test.startTest();
        app.Applicant_Status__c = 'Approved';
        update app;
        Test.stopTest();

        // Check Task created
        List<Task> tasks = [SELECT Id, Subject, WhatId FROM Task WHERE WhatId
= :app.Id];
        System.assertEquals(1, tasks.size(), 'Task should be created when
Application is Approved');
        System.assertEquals('Application Approved Notification',
tasks[0].Subject, 'Task subject should match');
    }

    @IsTest
    static void testInterviewValidation_NoOverlap() {
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();
        Application__c app = createApplication(c.Id, job.Id, 'Applied');

        // First interview slot
        Interview__c int1 = createInterview(app.Id,
Datetime.now().addDays(1));

        // Overlapping interview slot
        Interview__c int2 = new Interview__c(
            Applicant_Information__c = app.Id,
            Interview_Date_Time__c = int1.Interview_Date_Time__c // same time
        );

        Test.startTest();
        try {
            insert int2;
            System.assert(false, 'Should not allow overlapping interviews');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('overlap'), 'Should block
overlapping interview creation');
        }
        Test.stopTest();
    }
}
```

13. Asynchronous Processing

Explanation:
Asynchronous processing (Batch Apex, Queueable, Scheduled, Future methods) allows operations to run in the background without blocking the main execution.

Scenario:

- Batch Apex: Closing inactive applications.
- Queueable Apex: Sending notifications for new job postings.
- Scheduled Apex: Interview reminders.
- Future Method: Background verification with external systems.

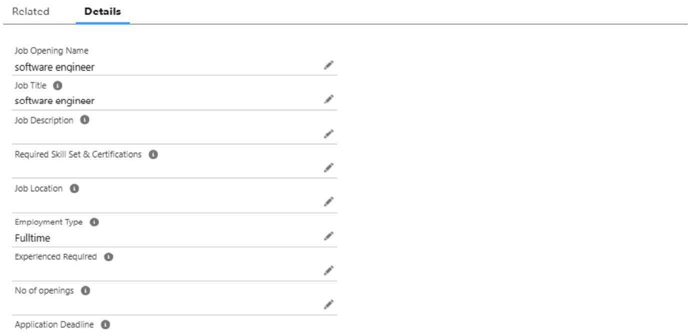
This ensured better performance and scalability of the system.

Phase 6: User Interface Development

In this phase, the focus was on creating an intuitive and interactive interface for recruiters and HR managers to manage Job Postings, Applications, and Interviews efficiently. Salesforce Lightning Experience along with **Lightning Web Components (LWC)** was leveraged to enhance usability, display dynamic data, and integrate backend functionality.

1. Lightning App Builder & Record Pages

- **Job Posting Record Page**
  - Customized to display key details such as Job Title, Department, Location, Skills Required, and Application Deadline.
  - Integrated **Job Applications LWC**, which lists all Applications associated with the Job Posting dynamically.



Job Opening Name
software engineer

Job Title
software engineer

Job Description

Required Skill Set & Certifications

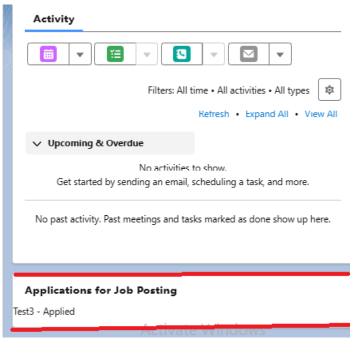
Job Location

Employment Type
Fulltime

Experienced Required

No of openings

Application Deadline



Filters: All time • All activities • All types

No activities to show.
Get started by sending an email, scheduling a task, and more.

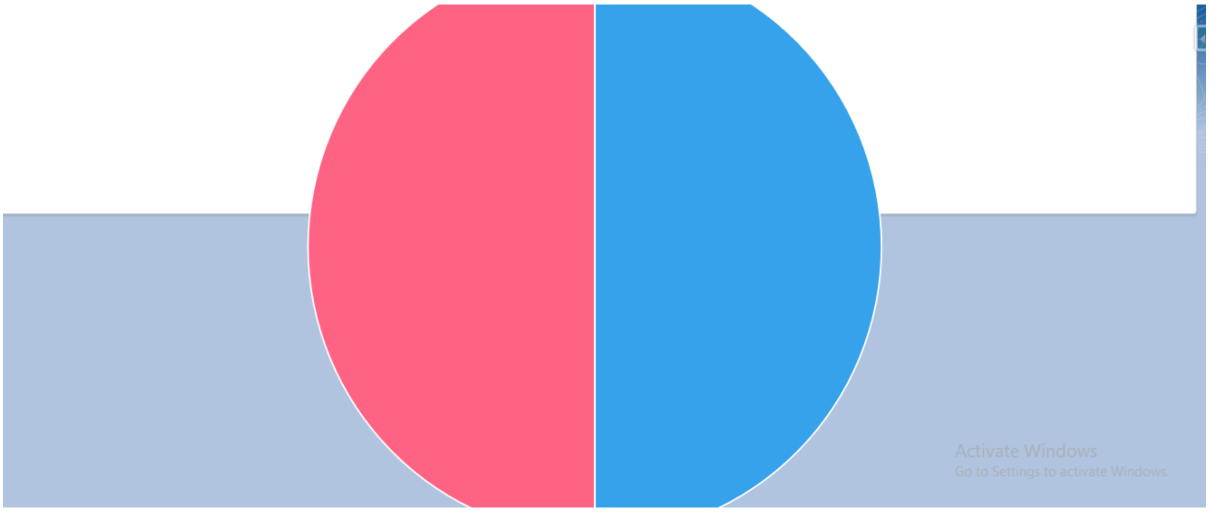
No past activity. Past meetings and tasks marked as done show up here.

Applications for Job Posting

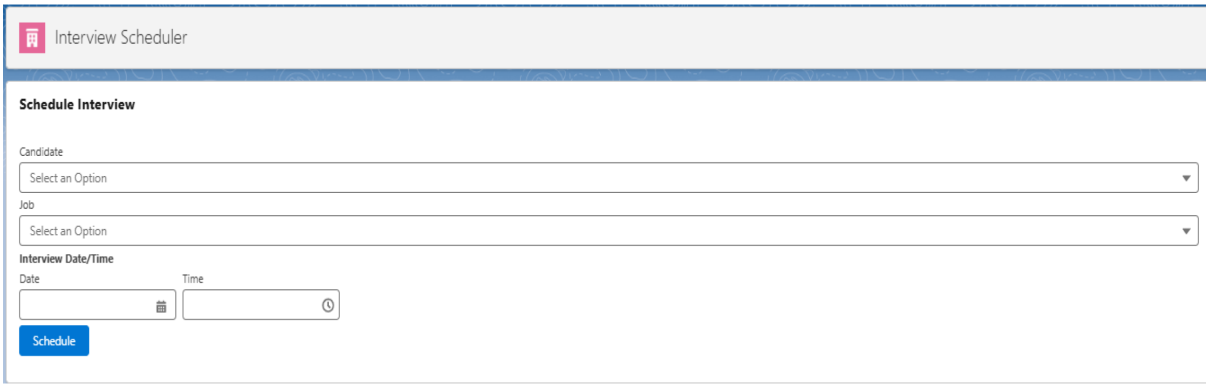
Test3 - Applied

  - Related lists of Applications and Interviews are included for complete visibility of job progress.

- **Application Record Page**
  - Displays candidate details, status, and associated Job Posting.
  - Integrated **Application Dashboard LWC**, summarizing candidate information and interview schedules.



- ○ **Interview Scheduler LWC** added to allow recruiters to schedule interviews directly from the Application record.



Schedule Interview

Candidate
Select an Option

Job
Select an Option

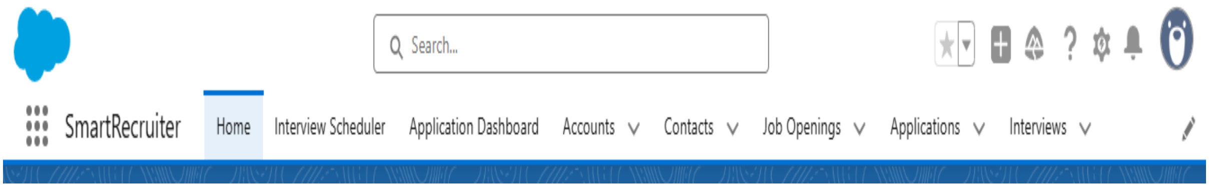
Interview Date/Time

Date

Time

2. Tabs and Navigation

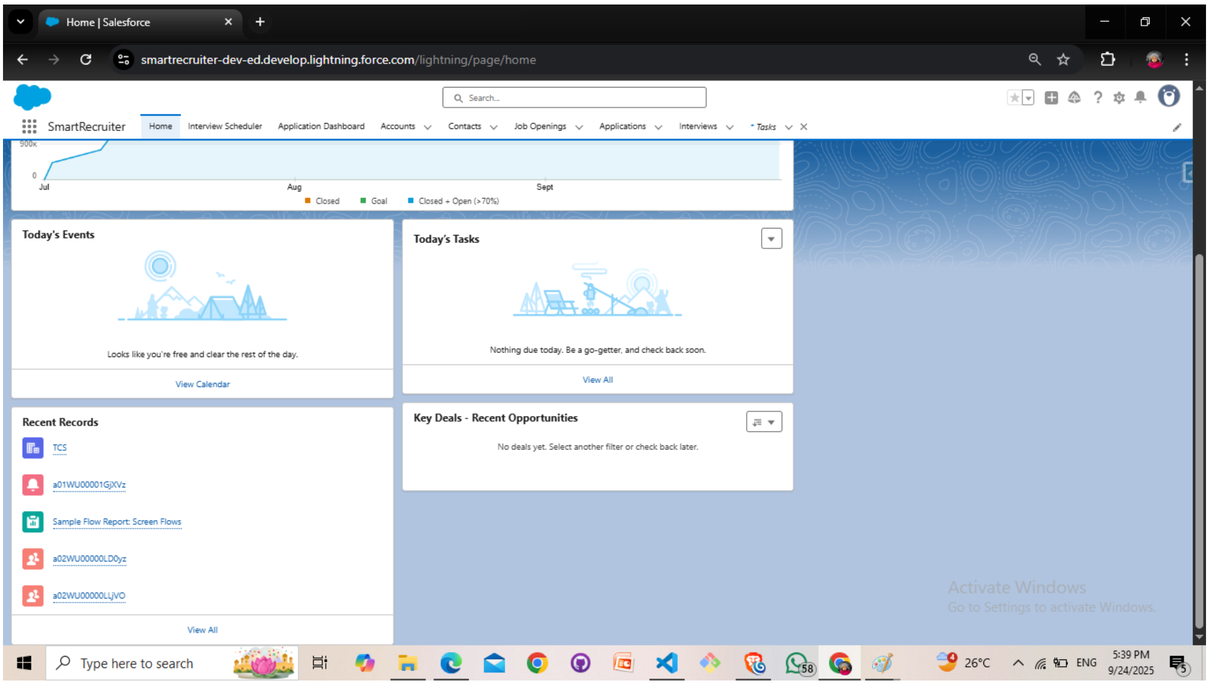
- Custom **tabs** created for Job Postings and Applications for easy access from the App Launcher.
- Custom Tabs for the object are present in the smart recruiter App in the app manager. As shown below



- Enables recruiters to navigate efficiently between Jobs, Applications, and Interviews.

3. Home Page Layouts & Utility Bar

- Home Page layouts designed to highlight recent applications, pending approvals, and upcoming interviews.



Today's Events

Looks like you're free and clear the rest of the day.

Today's Tasks

Nothing due today. Be a go-getter, and check back soon.

Recent Records

- TCS
- a01WU00001GjXVz
- Sample Flow Report: Screen Flows
- a02WU00000LD0yz
- a02WU00000LLjVO

Key Deals - Recent Opportunities

No deals yet. Select another filter or check back later.

- Approval records and upcoming interviews I will be implementing using report and dashboards.
- Utility Bar can be configured to provide quick access to frequently used components like interview scheduling or application creation.

4. Lightning Web Components (LWC)

- **Job Applications LWC**
  - Displays all applications for a selected Job Posting.
  - Connected to Apex via **wire adapter**, providing real-time data display.
  - **Applications In Job Opening. HTML**

```html
<template>
    <lightning-card title="Applications for Job Posting">
        <template if:true={applications}>
            <template for:each={applications} for:item="app">
                <p key={app.Id}>
                    {app.Contact__r.Name} - {app.Applicant_Status__c}
                </p>
            </template>
        </template>

        <template if:true={error}>
            <p class="slds-text-color_error">{error}</p>
        </template>
    </lightning-card>
</template>
```

  - **Applications In Job Opening. Js**

```javascript
import { LightningElement, api, wire } from 'lwc';
import getApplicationsByJob from
'@salesforce/apex/ApicationInJobPosting.getApplicationsByJob';

export default class JobApplications extends LightningElement {
    @api recordId; // Job Posting Id from record page
    applications;
    error;

    @wire(getApplicationsByJob, { jobId: '$recordId' })
    wiredApplications({ error, data }) {
        if(data) {
            this.applications = data;
            this.error = undefined;
        } else if(error) {
            this.error = error.body ? error.body.message : error;
            this.applications = undefined;
        }
    }
}
```

- ### Applications In Job Opening. Xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

- ## Application Dashboard LWC
  - Summarizes candidate information and associated interviews.
  - Provides visual insights into the recruitment pipeline.

- ## Application Dashboard LWC . html

```
<template>
    <lightning-card title="Applications Dashboard">
        <div class="chart-container">
            <canvas></canvas>
        </div>
    </lightning-card>
</template>
```

- ## Application Dashboard LWC.js

```
import { LightningElement, wire } from 'lwc';
import { loadScript } from 'lightning/platformResourceLoader';
import ChartJS from '@salesforce/resourceUrl/ChartJS';
import getApplicationsByStatus from
'@salesforce/apex/Application_Dashboard_Controller.getApplicationsByStatus';

export default class Application_Dashboard_Component extends LightningElement
{
    chart;
    chartData = {};
    chartInitialized = false;

    @wire(getApplicationsByStatus)
    wiredApplications({ error, data }) {
        if (data) {
            this.chartData = data;
            if (this.chartInitialized) {
                this.renderChart(); // re-render chart if Chart.js loaded
            }
        } else if (error) {
            console.error('Error fetching application data', error);
        }
    }

    renderedCallback() {
        if (this.chartInitialized) return;
        this.chartInitialized = true;

        loadScript(this, ChartJS)
            .then(() => {
                if (Object.keys(this.chartData).length > 0) {
                    this.renderChart();
                }
            })
            .catch(error => {
                console.error('Error loading ChartJS', error);
            });
    }
    renderChart() {
        const canvas = this.template.querySelector('canvas');
        if (!canvas) return;
        const ctx = canvas.getContext('2d');

        if (this.chart) {
            this.chart.destroy();
        }

        // eslint-disable-next-line no-undef
        // eslint-disable-next-line no-undef
this.chart = new Chart(ctx, {
    type: 'pie',
    data: {
        labels: Object.keys(this.chartData),
        datasets: [{
            data: Object.values(this.chartData),
            backgroundColor: ['#36A2EB', '#FF6384', '#FFCE56', '#4CAF50']
        }]
    },
    options: {
        responsive: true,       // makes it adjust to container
        maintainAspectRatio: true, // preserves aspect ratio
        legend: { position: 'bottom' }
    }
});

    }
}
```

- **Application Dashboard LWC.xml**

```
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

- **Application Dashboard LWC.css**

```
.chart-container {
    width: 100%;
    max-width: 400px; /* chart won’t exceed 400px */
    height: auto;
    margin: auto; /* center chart */
}

canvas {
    width: 100% !important;
    height: auto !important;
}
```

- **Interview Scheduler LWC**
  - Enables recruiters to schedule interviews for candidates directly from the Application record.
  - Collects interview date/time, mode (online/offline/phone), and assigns interviewers.
- **Interview Scheduler LWC.html**

```
<template>
    <lightning-card title="Schedule Interview">
        <div class="slds-p-around_medium">
            <lightning-combobox
                label="Candidate"
                value={candidateId}
                options={candidateOptions}
                onchange={handleCandidateChange}>
            </lightning-combobox>

            <lightning-combobox
                label="Job"
                value={jobId}
                options={jobOptions}
                onchange={handleJobChange}>
            </lightning-combobox>

            <lightning-input
                type="datetime"
                label="Interview Date/Time"
                value={interviewDate}
                onchange={handleDateChange}>
            </lightning-input>

            <lightning-button
                variant="brand"
                label="Schedule"
                onclick={scheduleInterview}
                class="slds-m-top_medium">
            </lightning-button>
        </div>
    </lightning-card>
</template>
```

- **Interview Scheduler LWC.js**

```
import { LightningElement, track, wire } from 'lwc';
import getCandidates from '@salesforce/apex/schedule_Interview.getCandidates';
import getOpenJobs from '@salesforce/apex/schedule_Interview.getOpenJobs';
import scheduleInterviewApex from
'@salesforce/apex/schedule_Interview.scheduleInterview';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class InterviewScheduler extends LightningElement {
    @track candidateId;
    @track jobId;
    @track interviewDate;
    candidateOptions = [];
    jobOptions = [];

    @wire(getCandidates)
    wiredCandidates({ data }) {
        if (data) {
            this.candidateOptions = data.map(c => ({ label: c.Name, value:
c.Id }));
        }
    }

    @wire(getOpenJobs)
    wiredJobs({ data }) {
        if (data) {
            this.jobOptions = data.map(j => ({ label: j.Name, value: j.Id }));
        }
    }

    handleCandidateChange(event) {
        this.candidateId = event.detail.value;
    }
    handleJobChange(event) {
        this.jobId = event.detail.value;
    }
    handleDateChange(event) {
        this.interviewDate = event.detail.value;
    }

    scheduleInterview() {
        scheduleInterviewApex({ candidateId: this.candidateId, jobId:
this.jobId, interviewDate: this.interviewDate })
            .then(() => {
                this.dispatchEvent(new ShowToastEvent({
                    title: 'Success',
                    message: 'Interview Scheduled!',
                    variant: 'success'
                }));
                this.candidateId = this.jobId = this.interviewDate = null;
            })
            .catch(error => {
                console.error(error);
                this.dispatchEvent(new ShowToastEvent({
                    title: 'Error',
                    message: 'Failed to schedule interview',
                    variant: 'error'
                }));
            });
    }
}
```

- **Interview Scheduler LWC.xml**

```
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

5. Apex Integration with LWC

- LWCs connected to Apex controllers to fetch and display data dynamically.

Apex Class for Application Dashboard Controller

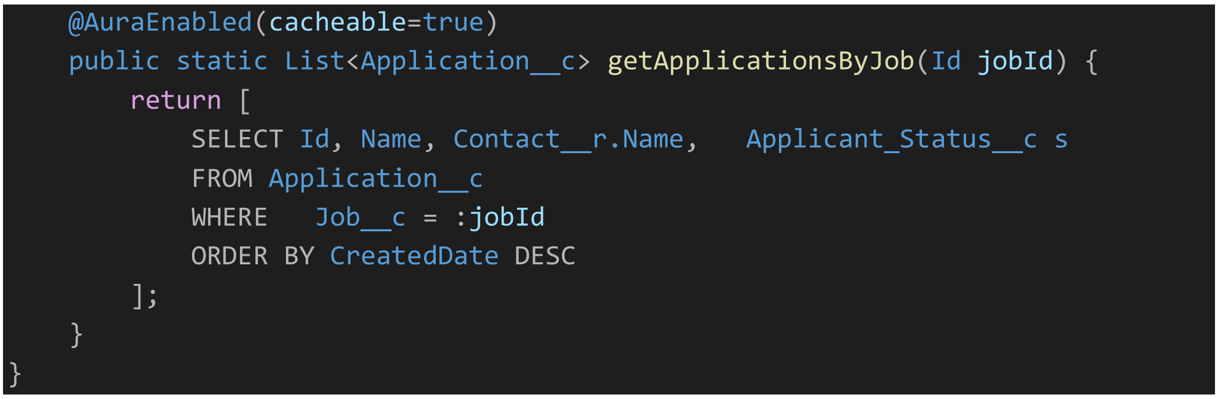
```
public with sharing class Application_Dashboard_Controller {
     @AuraEnabled(cacheable=true)
    public static Map<String, Integer> getApplicationsByStatus() {
        Map<String, Integer> statusCount = new Map<String, Integer>();

        // Query applications and group by status
        for (AggregateResult ar : [
            SELECT Applicant_Status__c status, COUNT(Id) cnt
            FROM Application__c
            GROUP BY Applicant_Status__c
        ]) {
            statusCount.put((String)ar.get('status'), (Integer)ar.get('cnt'));
        }

        return statusCount;
    }
}
```

Applications In job Opening -> apex class what fetch the data from the database

```
public with sharing class AppicationInJobPosting {
```



```
    @AuraEnabled(cacheable=true)
    public static List<Application__c> getApplicationsByJob(Id jobId) {
        return [
            SELECT Id, Name, Contact__r.Name,   Applicant_Status__c s
            FROM Application__c
            WHERE   Job__c = :jobId
            ORDER BY CreatedDate DESC
        ];
    }
}
```

- **Wire adapters** used to automatically update UI when data changes.

6. Events & Navigation Service

- LWCs communicate with each other using **custom events** where required, e.g., updating Application Dashboard after an interview is scheduled.
- **Navigation Service** used to redirect users to relevant record pages after actions such as interview creation.

Important Note - the code of all the lwc components in present in my github repo.

Phase 7: Integration & External Access - Smart Recruiter Portal Report

1. Phase Objective

The objective of Phase 7 was to establish a secure and accessible external portal for job applicants. This was a critical step in streamlining the recruitment process by providing a self-service platform for candidates to view job openings, apply for positions, and track the status of their applications. The primary technology chosen for this integration was **Salesforce Experience Cloud**.

2. Integration & External Access Overview

Experience Cloud was leveraged to create a branded portal that extends core Salesforce functionality to external users (in this case, job applicants) without exposing the internal Salesforce org. The platform's native capabilities addressed several key requirements of this phase:

- **OAuth & Authentication:** Experience Cloud provides a robust authentication model for portal users. It automatically manages user logins and sessions, ensuring that applicants can securely access the portal using their own credentials, which are tied to a `Contact` record within the Salesforce database.

- **Web Services (REST/SOAP) & Callouts:** The portal's front-end seamlessly interacts with Salesforce’s back-end data. The platform's underlying architecture, built on REST and SOAP APIs, provides a foundation for any future integration needs, such as connecting to an external HR system or a third-party resume parser.
- **API Limits:** By utilizing a standard Experience Cloud license, the project benefits from Salesforce's pre-configured API limits, ensuring that the portal operates reliably at scale without impacting the primary Salesforce org’s performance.
- **Platform Events & Change Data Capture:** These features were considered for future enhancements. By setting up Platform Events or CDC on the `Application` object, the system can automatically send real-time notifications to applicants about changes in their application status (e.g., from "Applied" to "Under Review"), providing a proactive communication channel.
- **Salesforce Connect:** While not used in the initial build, Salesforce Connect provides a clear path for future integration by allowing external data sources (e.g., a candidate database in a different system) to be surfaced within the portal without data migration.

3. Implementation Summary: The Applicant Portal

The core deliverable of this phase was the creation and configuration of the applicant portal. The following key steps were executed:

- **Experience Cloud Site Creation:** A new Experience Cloud site was provisioned using the **Build Your Own (LWR)** template to ensure a clean, performant, and customizable foundation.
- **Portal Page Development:**
    - **Job Openings Page:** A dynamic page was created using the **Record List** component to display a comprehensive list of available job openings directly from the Salesforce `Job Opening` custom object.
    - **Job Details Page:** A dedicated **Object Page** was designed to provide a single-source view of each job opening. This page utilized the **Record Detail** component to display job specifics.
    - **"My Applications" Page:** A personalized page was built using a filtered **Record List** component, ensuring that each logged-in applicant could only view their own submitted applications.
- **Flow-Based Application Process:** A robust **Screen Flow** was developed and embedded directly on the

"Job Details" page. This flow automated the application process by:

  - Collecting applicant information (Name, Email, Resume).
  - Creating a new `Application` record in Salesforce.
  - Linking the new `Application` record to the correct `Job Opening` via a Combobox component.
  - Linking the `Application` record to the applicant's Salesforce `Contact` record using the `$User.ContactId` global variable.

- **Security & Data Access Control:** The most critical step was the implementation of **Sharing Sets**. This was the primary mechanism to enforce data privacy and security, ensuring that external users could only view records (their own applications) that were directly related to their `Contact` record.

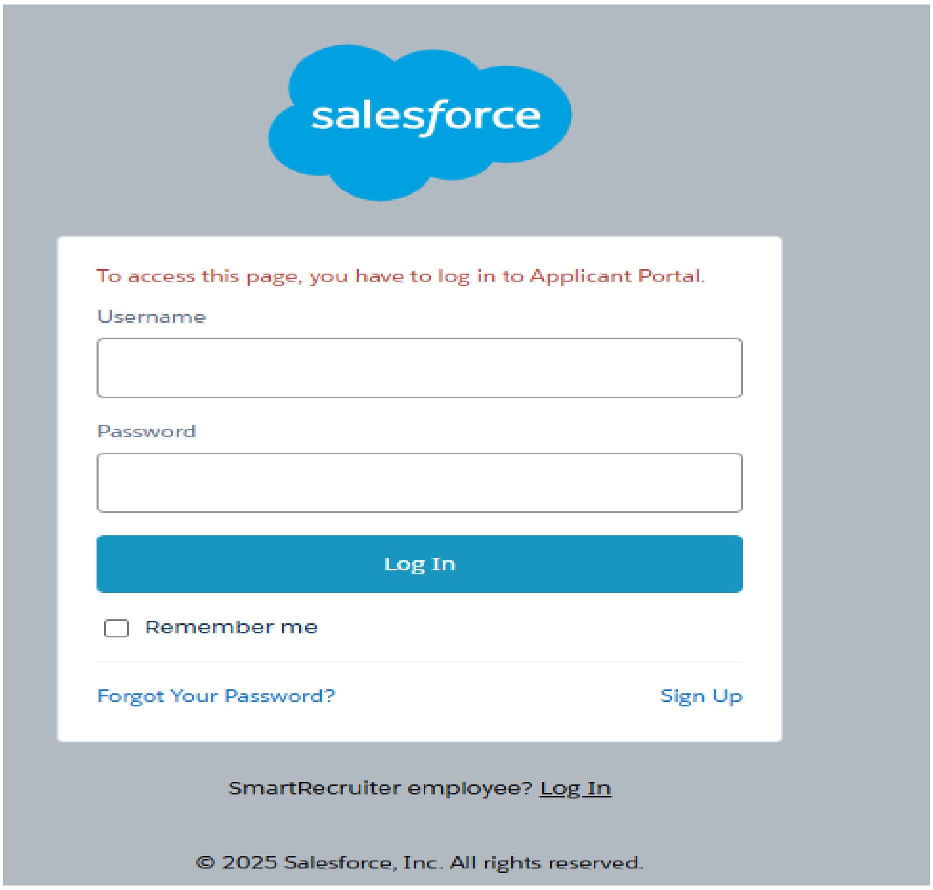
4. Conclusion

The successful implementation of the Experience Cloud portal marks the completion of the Integration & External Access phase. The project now has a secure, scalable, and user-friendly external interface. This platform not only meets the initial project requirements but also provides a strong foundation for future enhancements and further integrations, positioning the "Smart Recruiter" project for continued success.

- Link the candidate Portal -> https://smartrecruiter-dev-ed.develop.my.site.com/careerconnect/s

Appicant Profile – Experienced Cloud

- Login Page



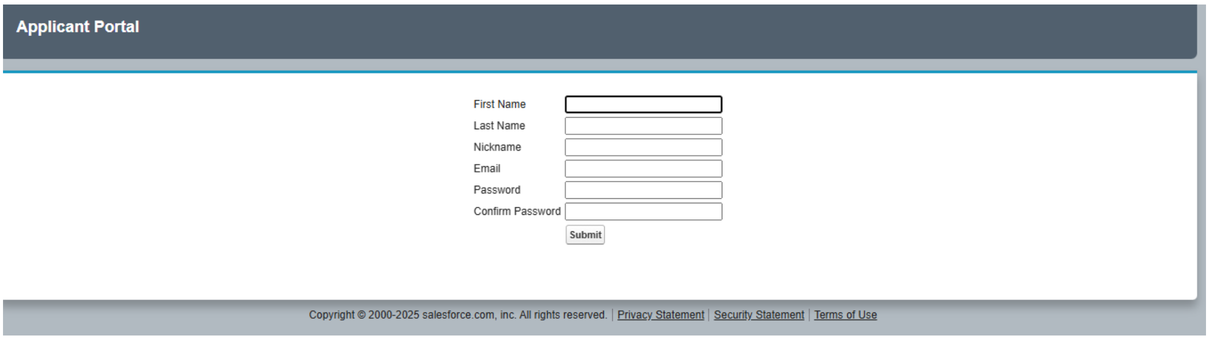
To access this page, you have to log in to Applicant Portal.

Username

Password

salesforce
Log In
□ Remember me
Forgot Your Password?
Sign Up
SmartRecruiter employee? Log In
© 2025 Salesforce, Inc. All rights reserved.

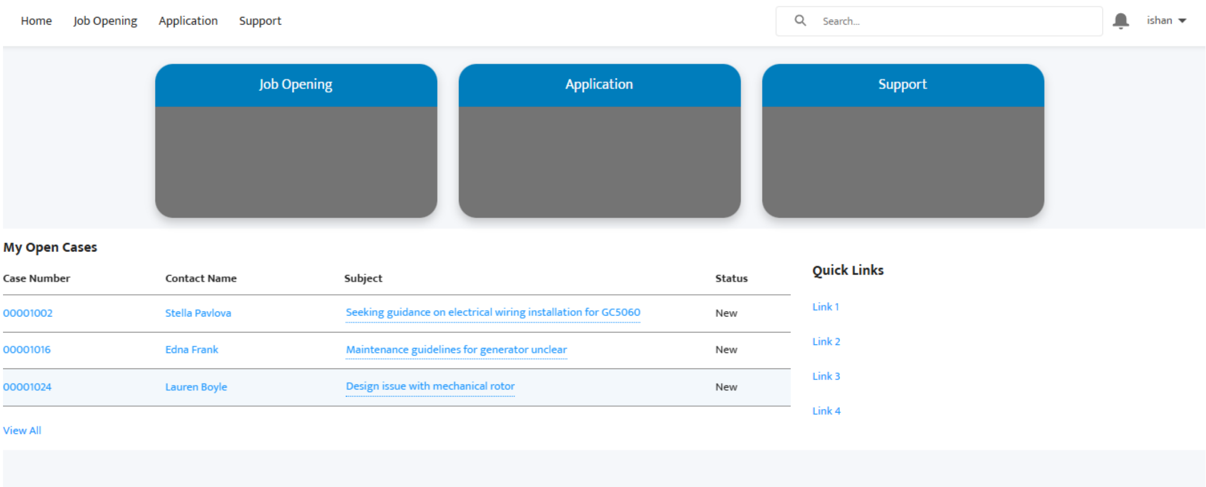
Sign Page



First Name
Last Name
Nickname
Email
Password
Confirm Password

Copyright © 2000-2025 salesforce.com, inc. All rights reserved. | Privacy Statement | Security Statement | Terms of Use

Home Page for Applicant Portal



My Open Cases

Case Number	Contact Name	Subject	Status
00001002	Stella Pavlova	Seeking guidance on electrical wiring installation for GC5060	New
00001016	Edna Frank	Maintenance guidelines for generator unclear	New
00001024	Lauren Boyle	Design issue with mechanical rotor	New

View All

Quick Links

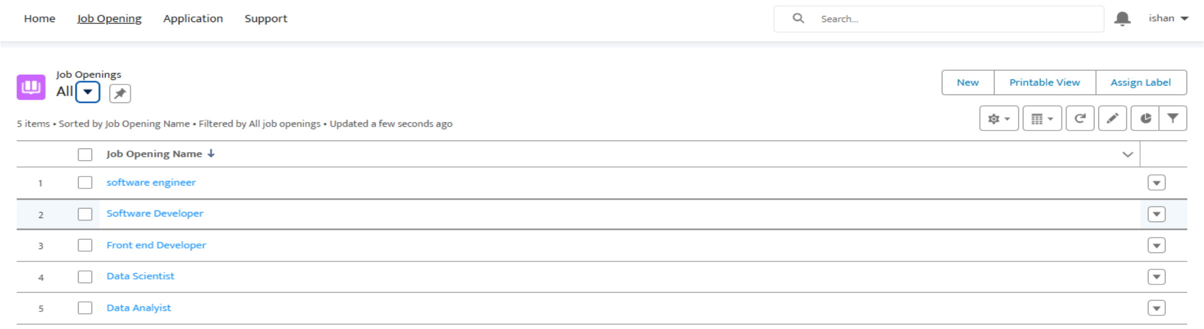
Link 1

Link 2

Link 3

Link 4

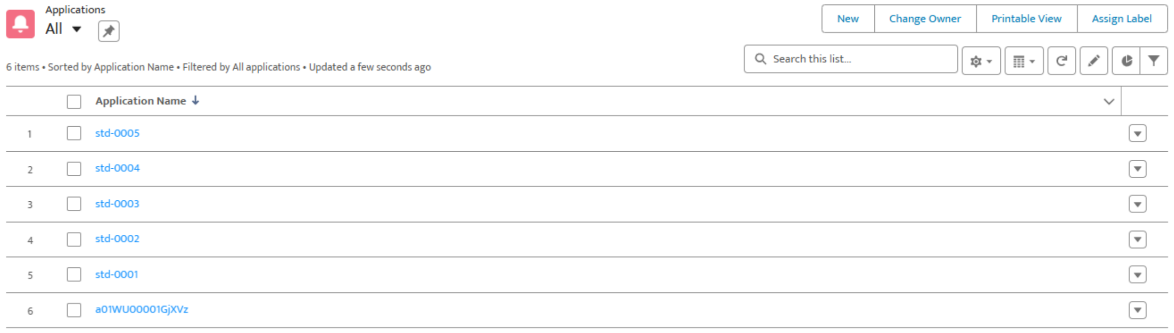
Job Opening - the page that contains all the Active Job Opening in the applicant portal.



5 items • Sorted by Job Opening Name • Filtered by All job openings • Updated a few seconds ago

		Job Opening Name ↓
1	□	software engineer
2	□	Software Developer
3	□	Front end Developer
4	□	Data Scientist
5	□	Data Analyist

Applicant List – Page Which contains all the applications of the same and different opening for the login user.



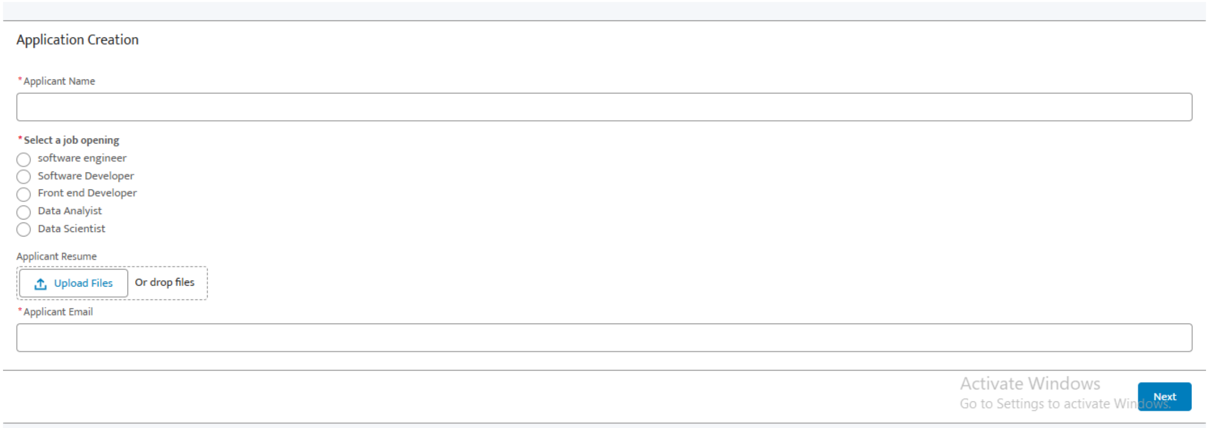
Applications

All

6 items • Sorted by Application Name • Filtered by All applications • Updated a few seconds ago

	Application Name
1	std-0005
2	std-0004
3	std-0003
4	std-0002
5	std-0001
6	a01WU00001GjXVz

JOb opening Details – It contains All the important details about the job opening and contains a screen flow that allow the user to submit the application for the opening directly to the company or organisation.



Application Creation

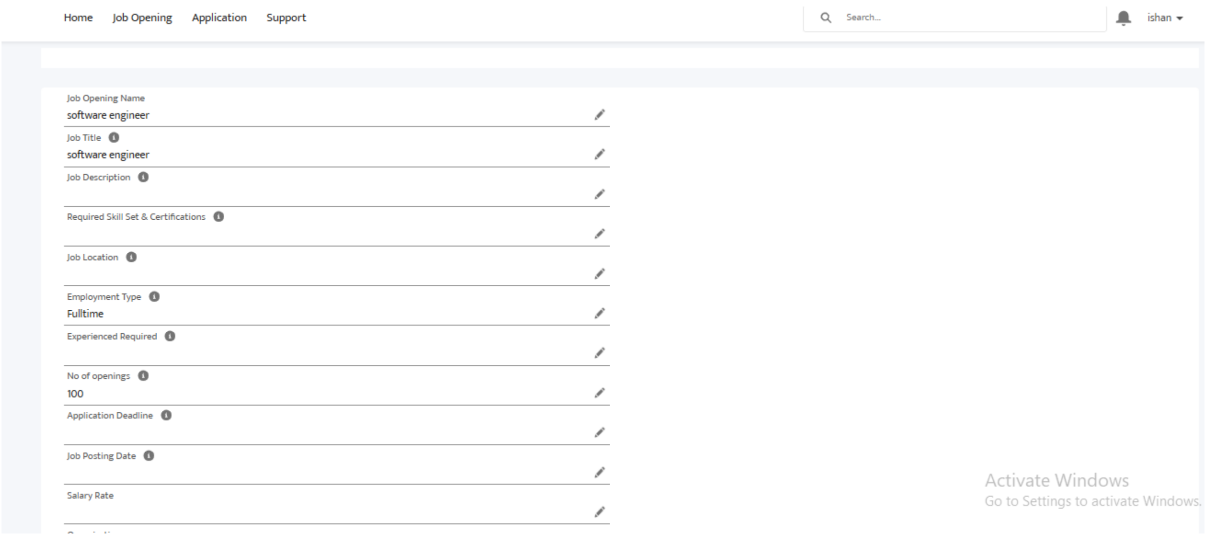
*Applicant Name

*Select a job opening

○ software engineer
○ Software Developer
○ Front end Developer
○ Data Analyist
○ Data Scientist

Applicant Resume

*Applicant Email



Job Opening Name	software engineer
Job Title	software engineer
Job Description	
Required Skill Set & Certifications	
Job Location	
Employment Type	Fulltime
Experienced Required	
No of openings	100
Application Deadline	
Job Posting Date	
Salary Rate	

Phase 8: Data Management & Deployment

1. Data Import Wizard

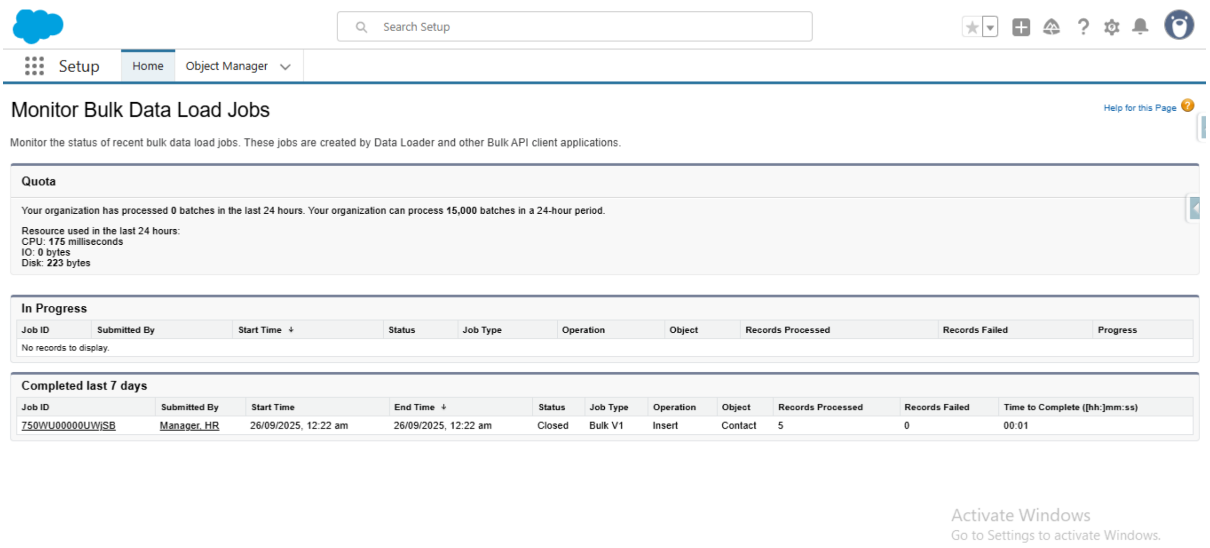
To populate sample data for testing and demonstration, the Salesforce **Data Import Wizard** was used for both standard and custom objects.

Steps Taken:

- Prepared CSV files for **Contacts** and **Job Postings** containing 5 records each.
- Uploaded CSVs via **Setup → Data Import Wizard → Launch Wizard**.
- Mapped CSV columns to Salesforce fields (e.g., FirstName → First Name, Location__c → Location).
- Imported successfully to create initial test data for the project.

Outcome:

- All sample Contacts and Job Postings are now available in Salesforce.
- Enabled creation of **Application** and **Interview** records for testing automation and flows.



Monitor Bulk Data Load Jobs

Monitor the status of recent bulk data load jobs. These jobs are created by Data Loader and other Bulk API client applications.

Quota

Your organization has processed **0** batches in the last 24 hours. Your organization can process **15,000** batches in a 24-hour period.

Resource used in the last 24 hours:
CPU: **175** milliseconds
IO: **0** bytes
Disk: **223** bytes

In Progress

Job ID	Submitted By	Start Time ↓	Status	Job Type	Operation	Object	Records Processed	Records Failed	Progress
No records to display.									

Completed last 7 days

Job ID	Submitted By	Start Time	End Time ↓	Status	Job Type	Operation	Object	Records Processed	Records Failed	Time to Complete ([hh:]mm:ss)
750WU00000UWjSB	Manager, HR	26/09/2025, 12:22 am	26/09/2025, 12:22 am	Closed	Bulk V1	Insert	Contact	5	0	00:01

2. Duplicate Rules

To prevent accidental duplicate records and ensure data integrity:

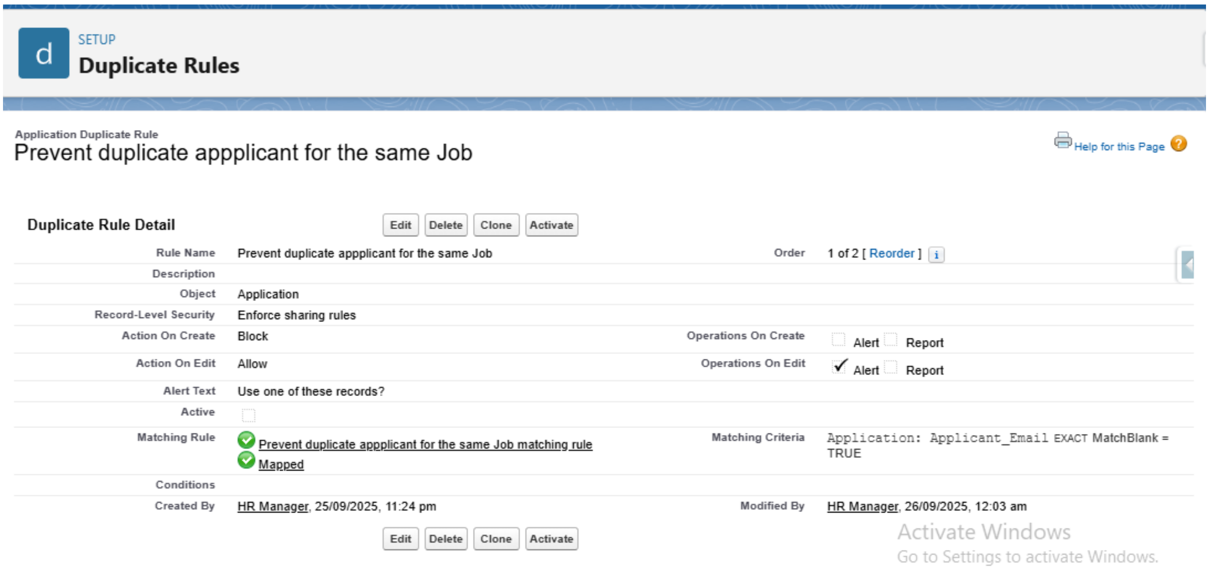
Implementation:

- Created **Matching Rules** on Contact object (Email equals).
- Created **Duplicate Rules** to **Alert** users if duplicate contacts are detected.
- Activated both rules.

Outcome:

- Recruiters cannot create duplicate candidate records.
- Ensures clean and reliable test data for recruitment processes.

Prevent Duplicate Application For the Same Job



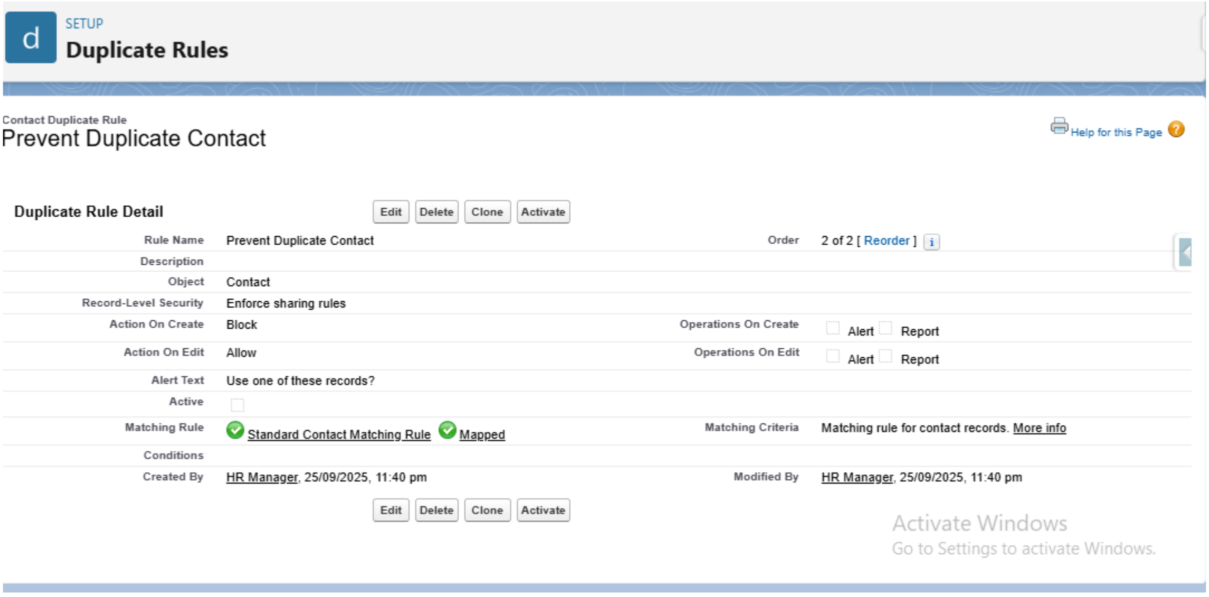
Application Duplicate Rule

Prevent duplicate appplicant for the same Job

Duplicate Rule Detail

Rule Name	Prevent duplicate appplicant for the same Job	Order	1 of 2 [Reorder]
Description			
Object	Application		
Record-Level Security	Enforce sharing rules		
Action On Create	Block	Operations On Create	□ Alert □ Report
Action On Edit	Allow	Operations On Edit	☑ Alert □ Report
Alert Text	Use one of these records?		
Active	□		
Matching Rule	✓ Prevent duplicate appplicant for the same Job matching rule ✓ Mapped	Matching Criteria	Application: Applicant_Email EXACT MatchBlank = TRUE
Conditions			
Created By	HR Manager, 25/09/2025, 11:24 pm	Modified By	HR Manager, 26/09/2025, 12:03 am

- Prevent Duplicate Contact



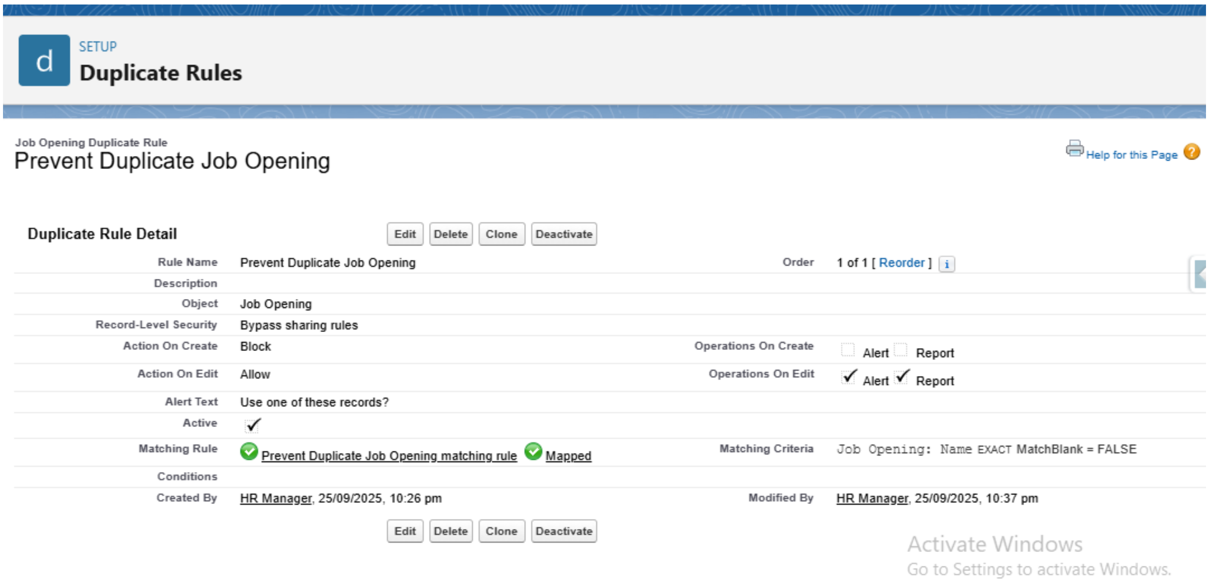
Contact Duplicate Rule

Prevent Duplicate Contact

Duplicate Rule Detail

Rule Name	Prevent Duplicate Contact	Order	2 of 2 [Reorder]
Description			
Object	Contact		
Record-Level Security	Enforce sharing rules		
Action On Create	Block	Operations On Create	□ Alert □ Report
Action On Edit	Allow	Operations On Edit	□ Alert □ Report
Alert Text	Use one of these records?		
Active	□		
Matching Rule	✓ Standard Contact Matching Rule ✓ Mapped	Matching Criteria	Matching rule for contact records. More info
Conditions			
Created By	HR Manager, 25/09/2025, 11:40 pm	Modified By	HR Manager, 25/09/2025, 11:40 pm

Prevent Duplicate Job Opening



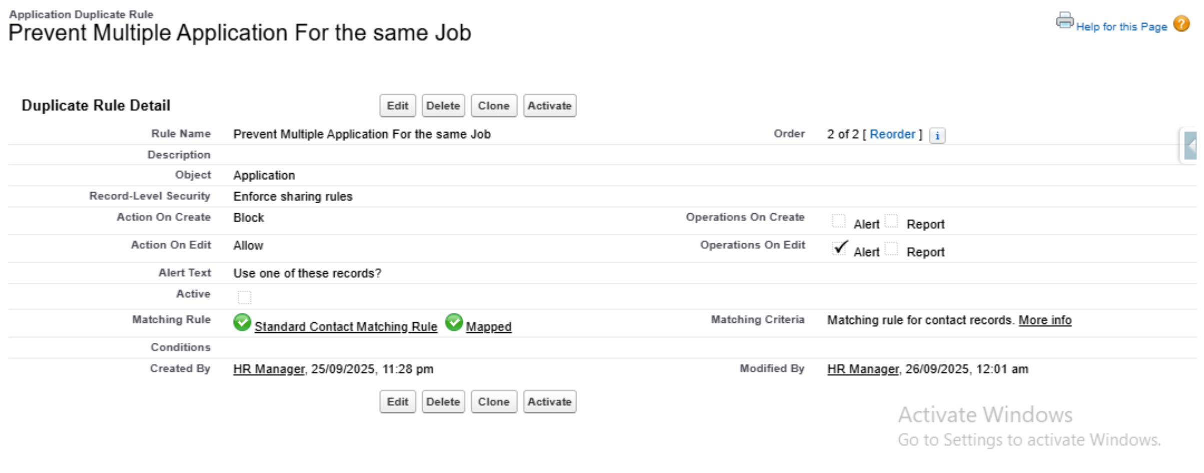
Job Opening Duplicate Rule

Prevent Duplicate Job Opening

Duplicate Rule Detail

Rule Name	Prevent Duplicate Job Opening	Order	1 of 1 [Reorder]
Description			
Object	Job Opening		
Record-Level Security	Bypass sharing rules		
Action On Create	Block	Operations On Create	□ Alert □ Report
Action On Edit	Allow	Operations On Edit	✓ Alert ✓ Report
Alert Text	Use one of these records?		
Active	✓		
Matching Rule	✓ Prevent Duplicate Job Opening matching rule ✓ Mapped	Matching Criteria	Job Opening: Name EXACT MatchBlank = FALSE
Conditions			
Created By	HR Manager, 25/09/2025, 10:26 pm	Modified By	HR Manager, 25/09/2025, 10:37 pm

Prevent Multiple Application for the same job



Application Duplicate Rule

Prevent Multiple Application For the same Job

Duplicate Rule Detail

Rule Name	Prevent Multiple Application For the same Job	Order	2 of 2 [Reorder]
Description			
Object	Application		
Record-Level Security	Enforce sharing rules		
Action On Create	Block	Operations On Create	□ Alert □ Report
Action On Edit	Allow	Operations On Edit	✓ Alert □ Report
Alert Text	Use one of these records?		
Active	□		
Matching Rule	✓ Standard Contact Matching Rule ✓ Mapped	Matching Criteria	Matching rule for contact records. More info
Conditions			
Created By	HR Manager, 25/09/2025, 11:28 pm	Modified By	HR Manager, 26/09/2025, 12:01 am

3. Data Loader

- Although the **Data Import Wizard** was sufficient for small datasets, **Data Loader** can be used for bulk insert, update, or export of records.
- In this project, small datasets were handled manually or via the Import Wizard due to time constraints.

4. Data Export & Backup

- Regular backups are critical for data safety.
- Salesforce **Data Export** feature can be used to export all records and metadata.
- For the project, manual export of Contacts, Job Postings, Applications, and Interviews was demonstrated.

- I Have Exported All the data in this Zip File.
- WE_00DWU00000VnOdg2AF_1.ZIP

6. VS Code & SFDX

- **VS Code** with Salesforce Extensions was used for development and deployment of:
  - Apex Classes & Triggers
  - Lightning Web Components

  - Metadata retrieval
- **SFDX Commands** help in pushing/pulling changes between local and org.

Outcome:

- Demonstrates modern Salesforce development practices.
- Enables easy tracking, version control, and deployment of all project components.
- I have imported all the data of salesforce in the vs code of connecting vs code to my org and pushed it in the project GitHub repository. This was Done by me in the previous phase only .

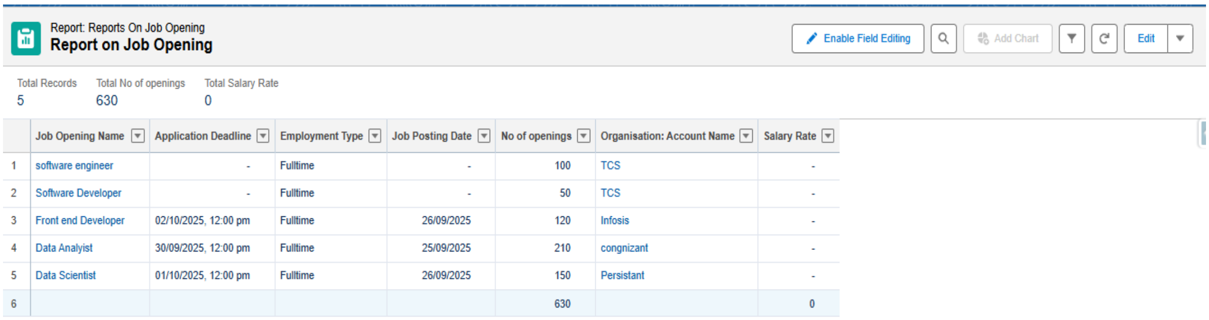
Phase 9: Reporting, Dashboards & Security Review

In this phase, the focus was on building insights, monitoring business processes, and ensuring data security within the Salesforce environment. The following tasks were implemented:

1. Reports

Different types of reports were created to analyse recruitment data and provide meaningful insights:

- **Tabular Reports:**
  Simple record listings, such as a list of all Candidates or Job Applications.
- Simpler report on the active job opening

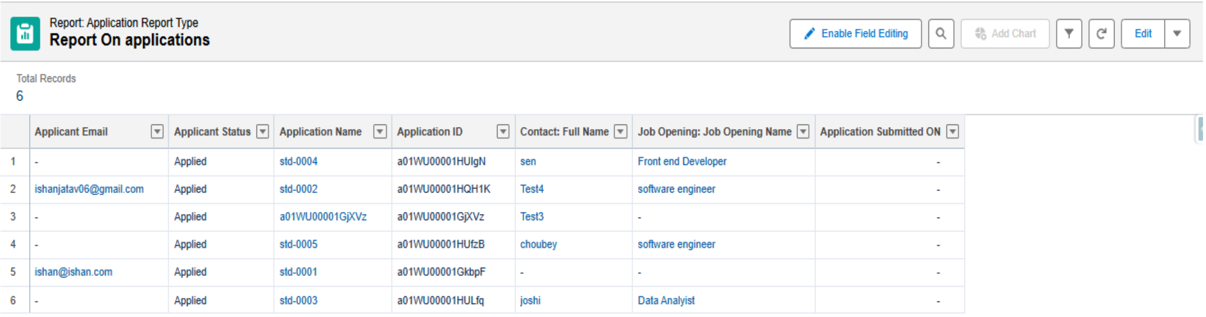


Report: Reports On Job Opening

Report on Job Opening

Total Records 5 | Total No of openings 630 | Total Salary Rate 0

	Job Opening Name	Application Deadline	Employment Type	Job Posting Date	No of openings	Organisation: Account Name	Salary Rate
1	software engineer	-	Fulltime	-	100	TCS	-
2	Software Developer	-	Fulltime	-	50	TCS	-
3	Front end Developer	02/10/2025, 12:00 pm	Fulltime	26/09/2025	120	Infosis	-
4	Data Analyist	30/09/2025, 12:00 pm	Fulltime	25/09/2025	210	congnizant	-
5	Data Scientist	01/10/2025, 12:00 pm	Fulltime	26/09/2025	150	Persistant	-
6					630		0



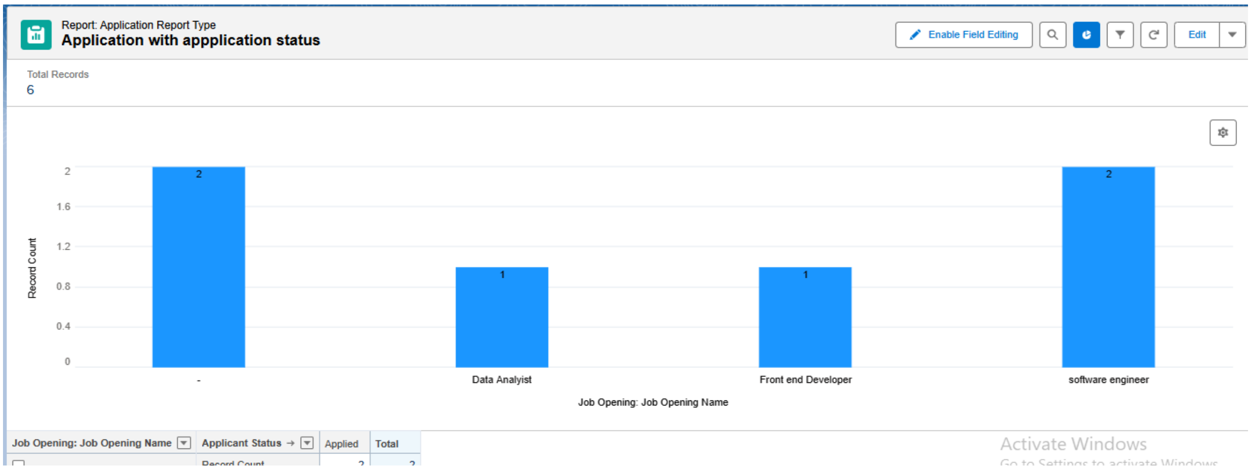
Report: Application Report Type
Report On applications

Total Records
6

	Applicant Email	Applicant Status	Application Name	Application ID	Contact: Full Name	Job Opening: Job Opening Name	Application Submitted ON
1	-	Applied	std-0004	a01WU00001HUIgN	sen	Front end Developer	-
2	ishanjatav06@gmail.com	Applied	std-0002	a01WU00001HQH1K	Test4	software engineer	-
3	-	Applied	a01WU00001GjXVz	a01WU00001GjXVz	Test3	-	-
4	-	Applied	std-0005	a01WU00001HUfzB	choubey	software engineer	-
5	ishan@ishan.com	Applied	std-0001	a01WU00001GkbpF	-	-	-
6	-	Applied	std-0003	a01WU00001HULfq	joshi	Data Analyist	-

- **Summary Reports:**
  Applications grouped by **Job Posting** or **Status** to quickly visualize recruitment progress.

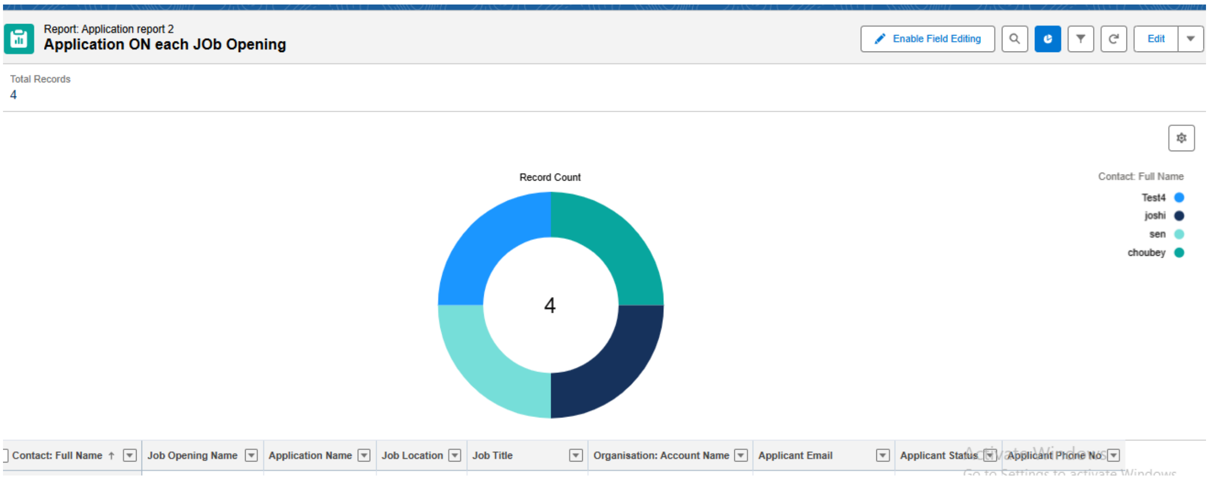
Application grouped by application Status



Report: Application Report Type
Application with appplication status

Total Records
6

Job Opening: Job Opening Name	Applicant Status →	Applied	Total
	Record Count	2	2

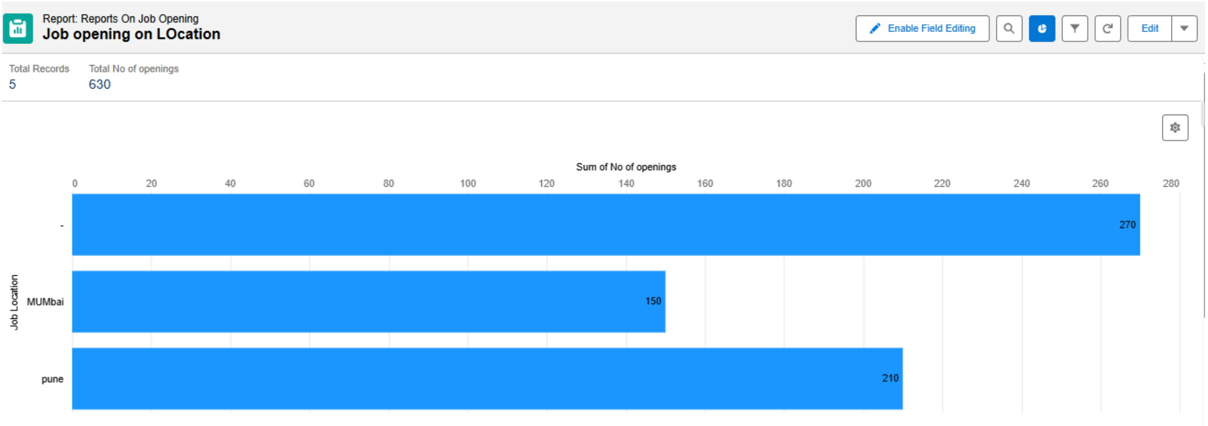


Report: Application report 2
Application ON each JOb Opening

Total Records
4

Contact: Full Name ↑	Job Opening Name	Application Name	Job Location	Job Title	Organisation: Account Name	Applicant Email	Applicant Status	Applicant Phone No

Applicaitons On each Job Opening


Report: Reports On Job Opening

Job opening on LOcation

Total Records 5

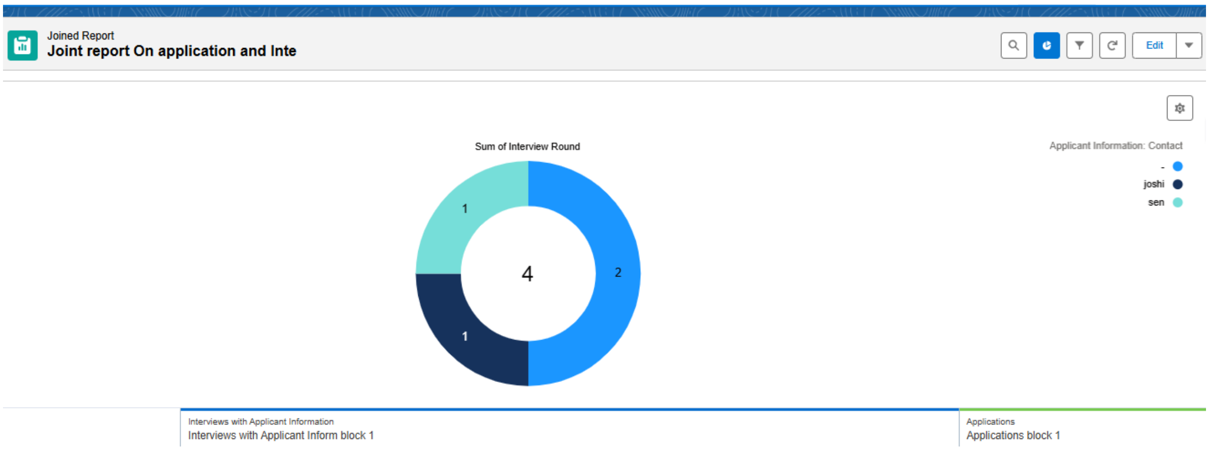
Total No of openings 630

Job Opening Grouped by Location

- **Matrix Reports:**
  Applications summarized by **Recruiter vs. Status**, giving a two-dimensional view of workload distribution.

- **Joined Reports:**
  Applications and Interviews displayed together, helping track how many interviews were scheduled per application.

Joint Report on Application and Interview


Joined Report

Joint report On application and Inte

Interviews with Applicant Information

Interviews with Applicant Inform block 1

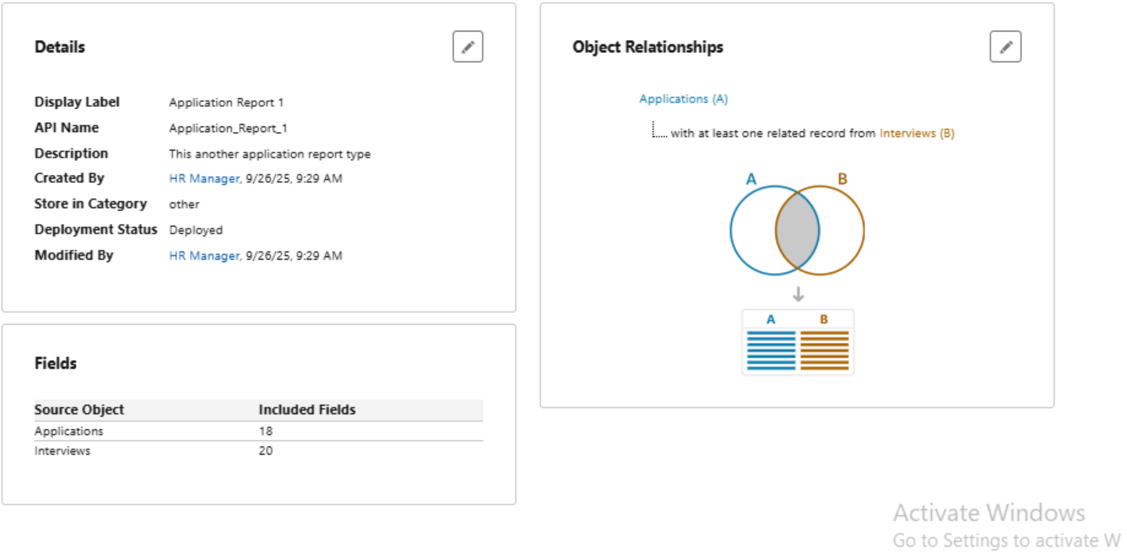
Applications

Applications block 1

2. Report Types

Custom Report Types were created to combine related objects such as:

- **Applications with Job Postings**



Details

Display Label	Application Report 1
API Name	Application_Report_1
Description	This another application report type
Created By	HR Manager, 9/26/25, 9:29 AM
Store in Category	other
Deployment Status	Deployed
Modified By	HR Manager, 9/26/25, 9:29 AM

Fields

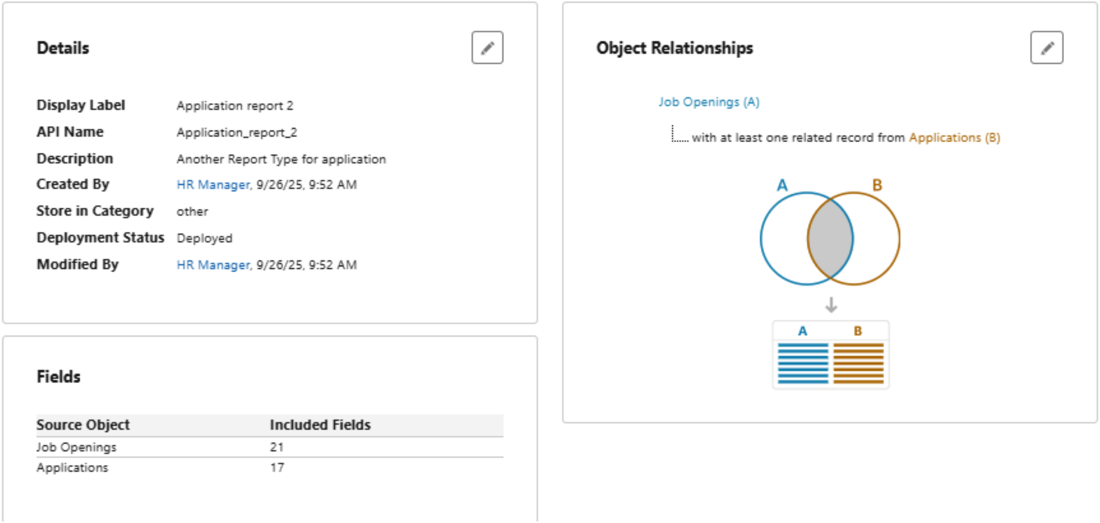
Source Object	Included Fields
Applications	18
Interviews	20

Object Relationships

Applications (A)

with at least one related record from Interviews (B)

- **Applications with Interviews**



Details

Display Label	Application report 2
API Name	Application_report_2
Description	Another Report Type for application
Created By	HR Manager, 9/26/25, 9:52 AM
Store in Category	other
Deployment Status	Deployed
Modified By	HR Manager, 9/26/25, 9:52 AM

Fields

Source Object	Included Fields
Job Openings	21
Applications	17

Object Relationships

Job Openings (A)

with at least one related record from Applications (B)

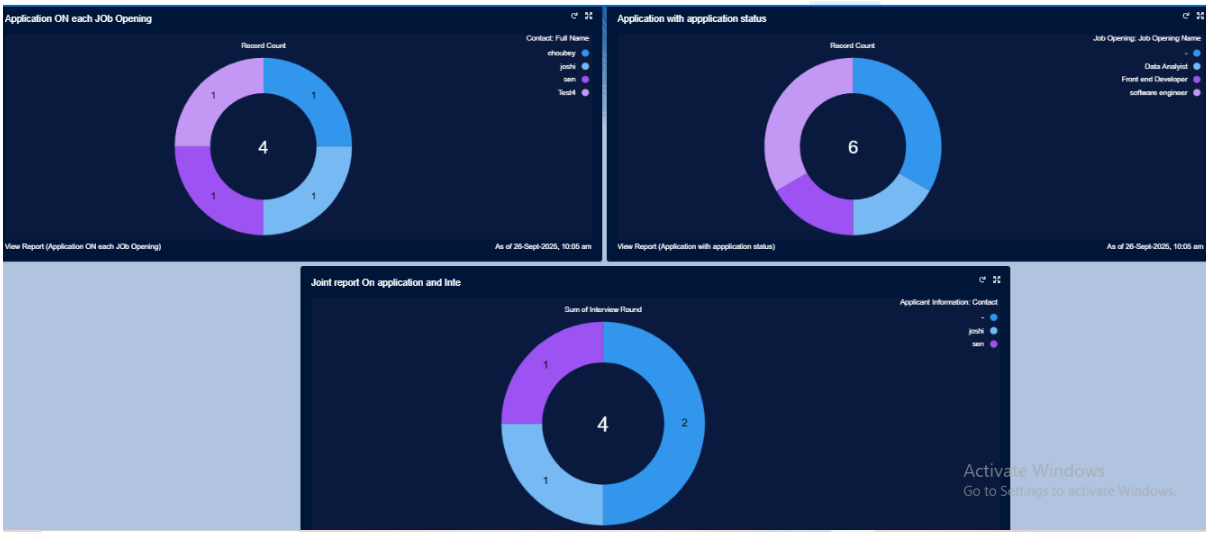
This enabled cross-object reporting to evaluate candidate progress throughout the hiring pipeline.

3. Dashboards

Dashboards were built to visually represent key recruitment KPIs:

- **Applications by Status** (Open, Approved, Rejected)
- **Applications per Job Posting**
- **Applications Selected for the interviews**

Each dashboard component provided recruiters with real-time insights into the hiring process.


Application ON each JOb Opening
Record Count
Contact: Full Name
choubey
joshi
sen
Test4
1
1
1
1
4
View Report (Application ON each JOb Opening)
As of 26-Sept-2025, 10:05 am
Application with appplication status
Record Count
Job Opening: Job Opening Name
-
Data Analyist
Front end Developer
software engineer
6
View Report (Application with appplication status)
As of 26-Sept-2025, 10:05 am
Joint report On application and Inte
Sum of Interview Round
Applicant Information: Contact
-
joshi
sen
1
2
1
4
Activate Windows
Go to Settings to activate Windows.

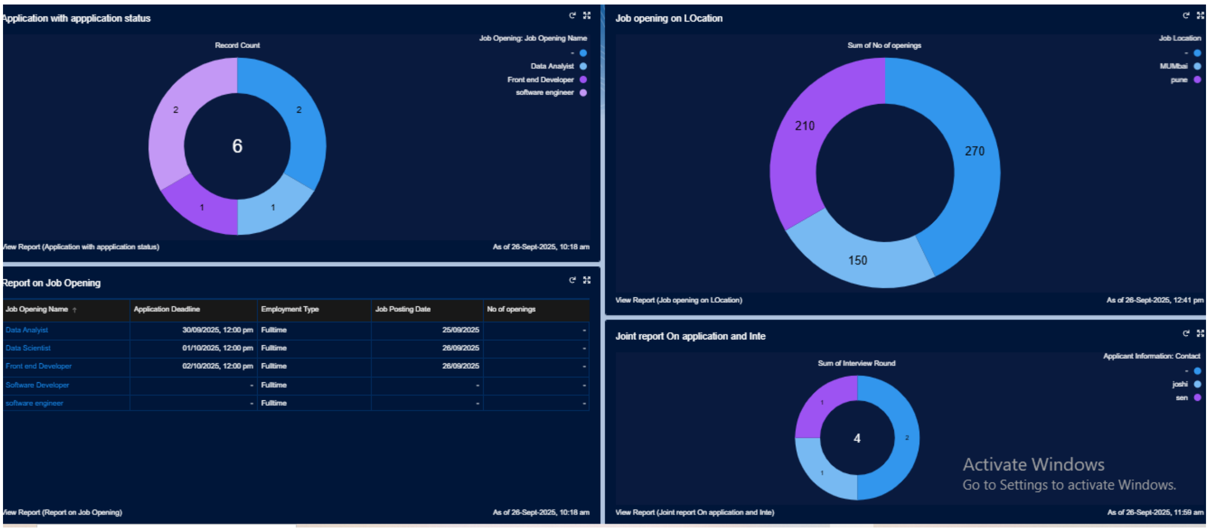
4. Dynamic Dashboards

Dynamic Dashboards were configured to ensure that users only see the data relevant to them. For example, if multiple recruiters are present, each recruiter would only see their own job applications and interviews. This was achieved by setting the dashboard

to run as the **logged-in user**, ensuring personalized views without the need for duplicate dashboards.

Dynamic Dashboard Includes

- **Application with application status**
- **Job opening with location**
- **Active job opening**
- **Applications selected for the interview**


Application with appplication status
Record Count
Job Opening: Job Opening Name
- ●
Data Analyist ●
Front end Developer ●
software engineer ●
2
2
6
1
1
View Report (Application with appplication status)
As of 26-Sept-2025, 10:18 am
Job opening on LOcation
Sum of No of openings
Job Location
- ●
MUMbai ●
pune ●
210
270
150
View Report (Job opening on LOcation)
As of 26-Sept-2025, 12:41 pm
Joint report On application and Inte
Sum of Interview Round
Applicant Information: Contact
- ●
joshi ●
sen ●
1
4
2
1
Activate Windows
Go to Settings to activate Windows.
View Report (Joint report On application and Inte)
As of 26-Sept-2025, 11:59 am

Report on Job Opening

Job Opening Name ↑	Application Deadline	Employment Type	Job Posting Date	No of openings
Data Analyist	30/09/2025, 12:00 pm	Fulltime	25/09/2025	-
Data Scientist	01/10/2025, 12:00 pm	Fulltime	26/09/2025	-
Front end Developer	02/10/2025, 12:00 pm	Fulltime	26/09/2025	-
Software Developer	-	Fulltime	-	-
software engineer	-	Fulltime	-	-

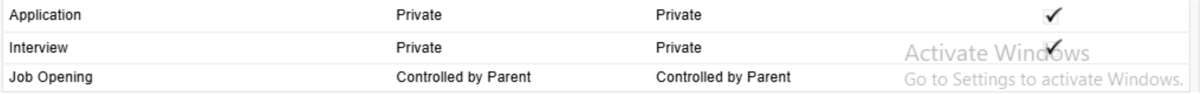
View Report (Report on Job Opening)

As of 26-Sept-2025, 10:18 am

5. Security Review

To ensure data integrity and compliance, several security measures were reviewed and implemented:

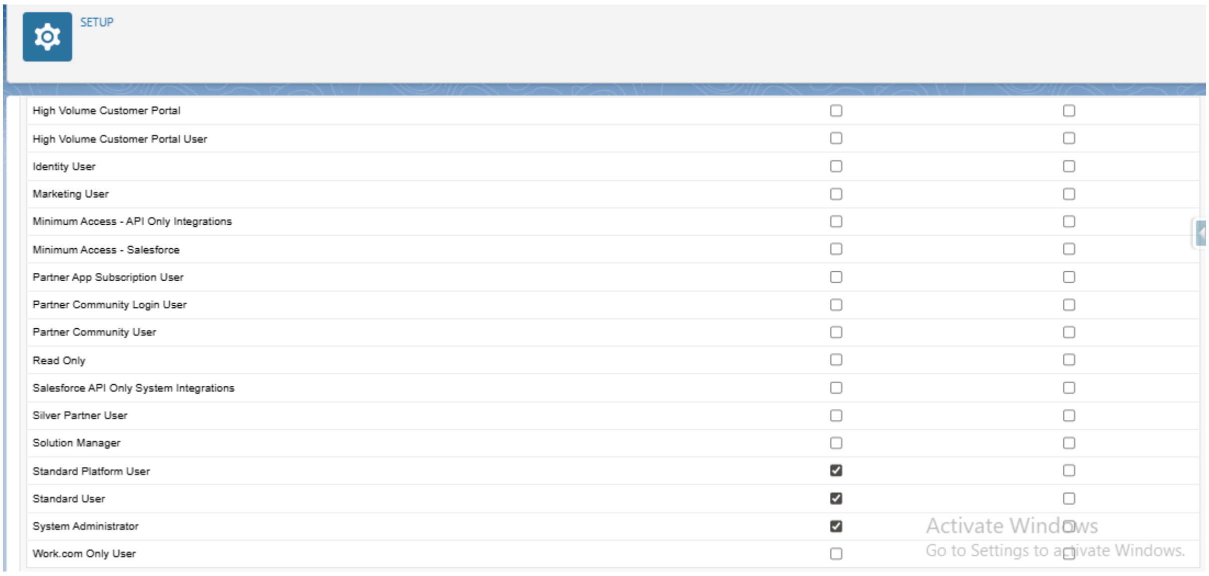
- **Sharing Settings:** Configured object-level and record-level access so that sensitive candidate and job data was only available to authorized users.
- I made application, interviews object private for the record level security and the job opening as public read only so the candidate can view the job opening.



Application	Private	Private	✓
Interview	Private	Private	✓
Job Opening	Controlled by Parent	Controlled by Parent	

- **Field-Level Security:** Restricted sensitive fields such as candidate contact details, ensuring only specific profiles could view or edit them. Hidden salary information from the external users.

Job salary field level security



High Volume Customer Portal	☐	☐
High Volume Customer Portal User	☐	☐
Identity User	☐	☐
Marketing User	☐	☐
Minimum Access - API Only Integrations	☐	☐
Minimum Access - Salesforce	☐	☐
Partner App Subscription User	☐	☐
Partner Community Login User	☐	☐
Partner Community User	☐	☐
Read Only	☐	☐
Salesforce API Only System Integrations	☐	☐
Silver Partner User	☐	☐
Solution Manager	☐	☐
Standard Platform User	☑	☐
Standard User	☑	☐
System Administrator	☑	☐
Work.com Only User	☐	☐

- **Session Settings:** Enforced stricter login and session timeouts for security.
- **Login IP Ranges:** Configured IP restrictions for admin-level access, enhancing system security.
- **Audit Trail:** Enabled to keep track of all configuration changes, ensuring accountability and traceability.