**Project Title : Healthcare Telecommunication Systems**

* ***Salesforce Project***
* **Project Title :** “Healthcare Teleconsultation System”
* **Industry :** Healthcare / Telemedicine
* **Type :** B2C & B2B Salesforce Health Cloud Implementation
* **Target Users :** Patients, Doctors, Clinic/Hospital Admins, and Support Staff
* ***Problem Statement***

Many patients—especially in rural or remote areas—struggle to access timely medical care. Appointments are booked manually, medical records are scattered, and video consultations lack a secure, unified platform. Clinics face difficulties tracking patient history, managing doctor schedules, and complying with healthcare data-privacy regulations.

A Salesforce-based Healthcare Teleconsultation System will:

* Centralize patient and doctor data in a single CRM.
* Automate appointment booking, reminders, and follow-ups.
* Provide secure, encrypted video consultations and e-prescriptions.
* Offer dashboards for doctors and administrators to track consultations and revenue.
* ***Phase 1: Problem Understanding & Industry Analysis***

• Requirement Gathering

• Stakeholder Analysis

• Business Process Mapping

• Industry-specific Use Case Analysis

• AppExchange Exploration

* **Goal :** Clearly define the healthcare access problem and capture functional/non-functional requirements before configuring Salesforce.

***1. Requirement Gathering***

Discussions with patients, doctors, hospital administrators, and IT staff revealed key pain points:

* Manual scheduling leads to missed appointments and poor follow-ups.
* Patient health records are scattered across multiple clinics.
* Secure, compliant video consultations and e-prescriptions are essential.
* Automation is needed for reminders, payment processing, and follow-up tracking.

***Key Business Requirements:***

* Patient registration with a unified health profile.
* Doctor availability management and integrated appointment calendar.
* Secure video integration (Zoom/Twilio) and encrypted chat.
* Automated SMS/email notifications and payment gateway.
* Dashboards to track patient volume, revenue, and doctor utilization.

1. ***Stakeholder Analysis***

* Patients -----> Book appointments, consult via video, and access medical history.
* Doctors / Healthcare Providers ------> Manage schedules, view complete patient data, and issue e-prescriptions.
* Clinic / Hospital Administrators ------->Oversee operations, billing, and compliance.
* IT & Support Staff ------>Maintain the system and assist users.
* Regulatory Authorities ------->Ensure adherence to data-privacy and security standards.

#### *3. Business Process Mapping*

**Current Manual Workflow**  
Patient calls hospital → Receptionist checks doctor availability → Appointment scheduled manually → Separate tools used for video calls and record storage → No centralized patient data.

**Proposed Salesforce Workflow**  
Patient registers on portal → Books doctor slot → Automated confirmation and reminders → Secure Salesforce-integrated video session → Doctor records notes and uploads prescription → Payment processed → Health records updated for future visits.

1. ***Industry-Specific Use Case Analysis (Healthcare)***

* **Telehealth Growth :** Rapid global expansion (~20% CAGR) post-COVID.
* **Compliance Needs :** HIPAA, GDPR, and India’s ABDM demand encryption, consent management, and audit trails.
* **Opportunities :** Integrating telemedicine with Salesforce Health Cloud provides a scalable, secure platform that many competitors lack.

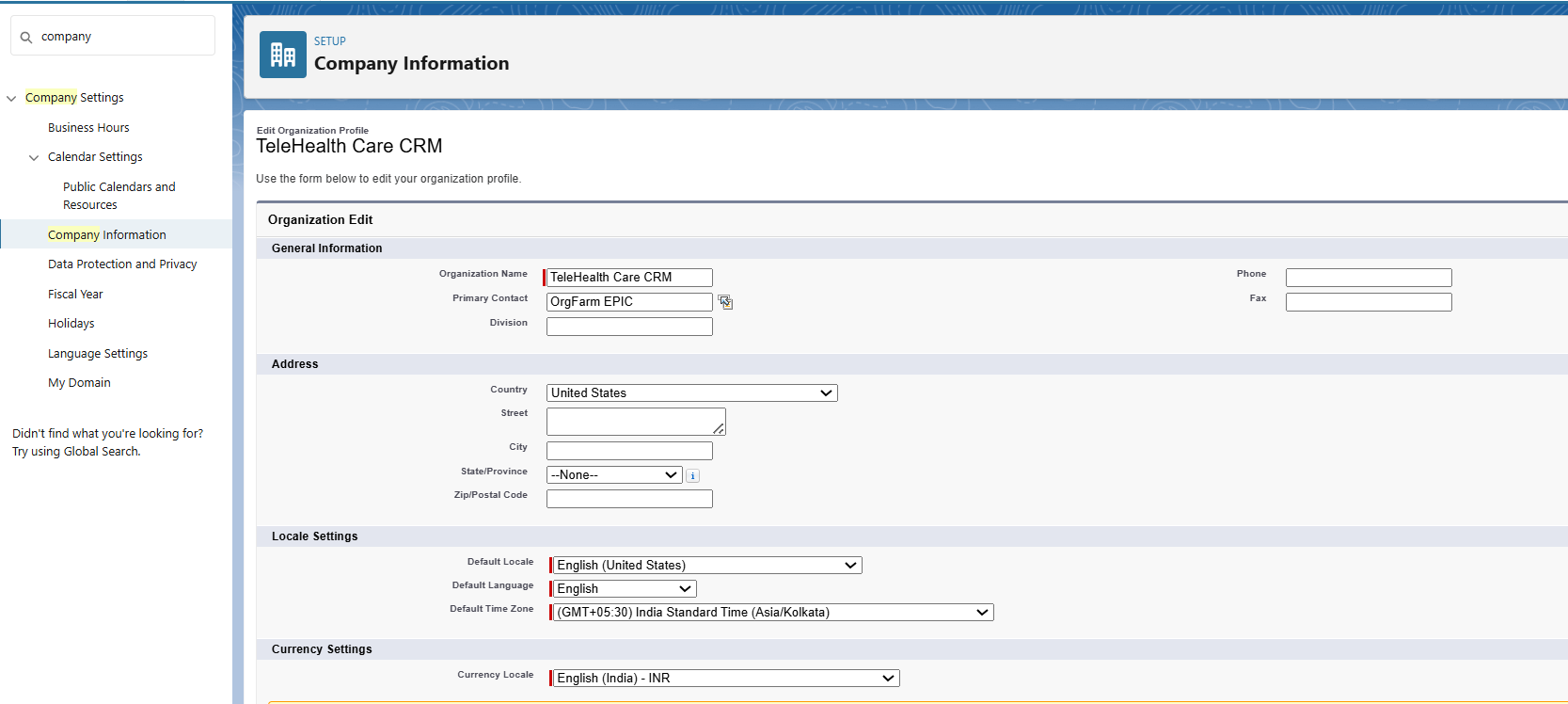
1. ***AppExchange Exploration***

Potential add-ons to enhance Health Cloud:

* **Telehealth/Video Apps** (Zoom, Vonage, Twilio).
* **E-Prescription & Pharmacy Integration.**
* **SMS/WhatsApp Connectors** for appointment reminders.
* **Survey Tools** for patient feedback.
* ***Phase 2: Org Setup & Configuration***
* ***Org Setup Settings :***

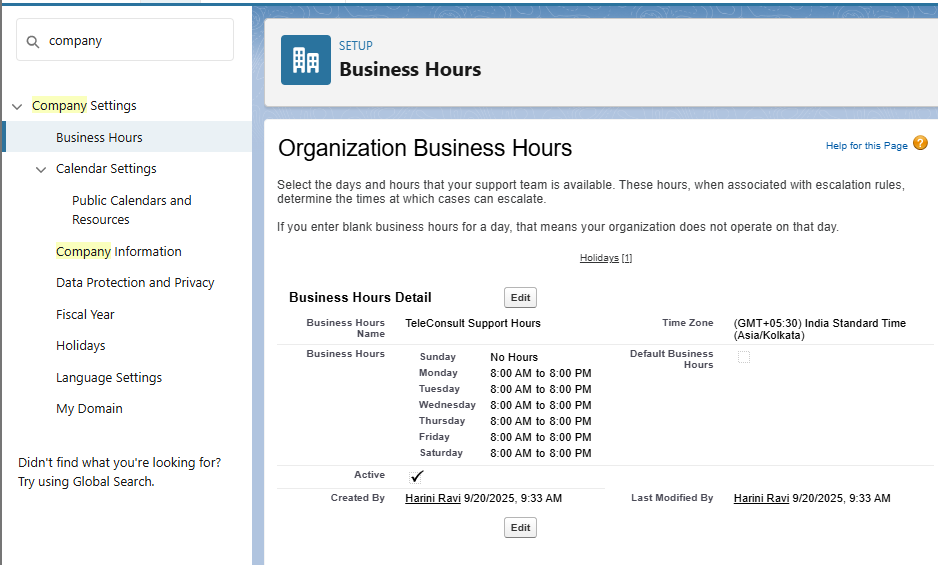
1. ***Company Information***

Setup ----> Company Settings ----> Company information ----> Edit Organization Name ----> Update Default Locale, Default Time Zone, Currency Locale



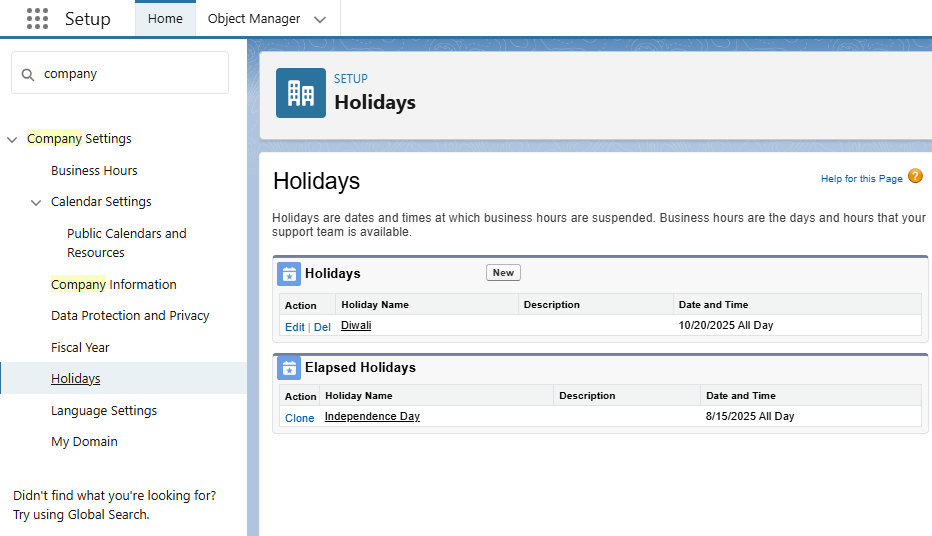
1. ***Business Hours & Holidays***

Setup ----> Business Hours ----> New ----> Create SLA support Hours



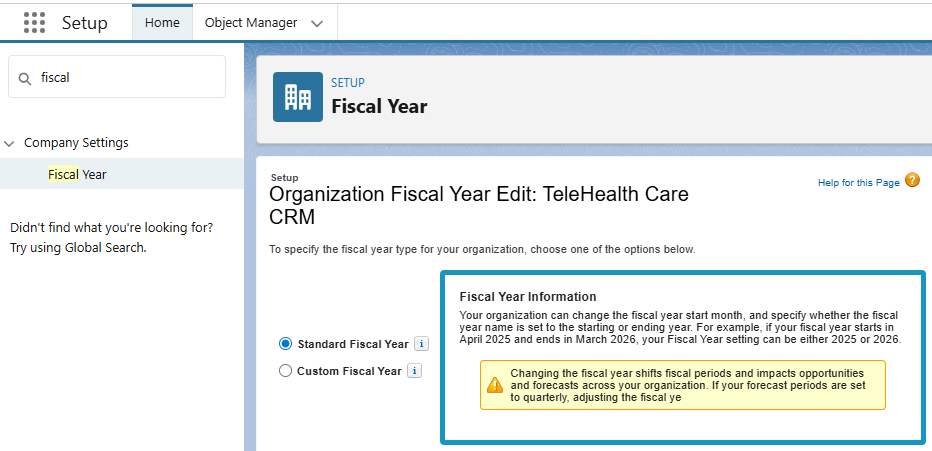
1. ***Holidays***

Setup ----> Holidays ----> Add Holidays for Independence day, Diwali.



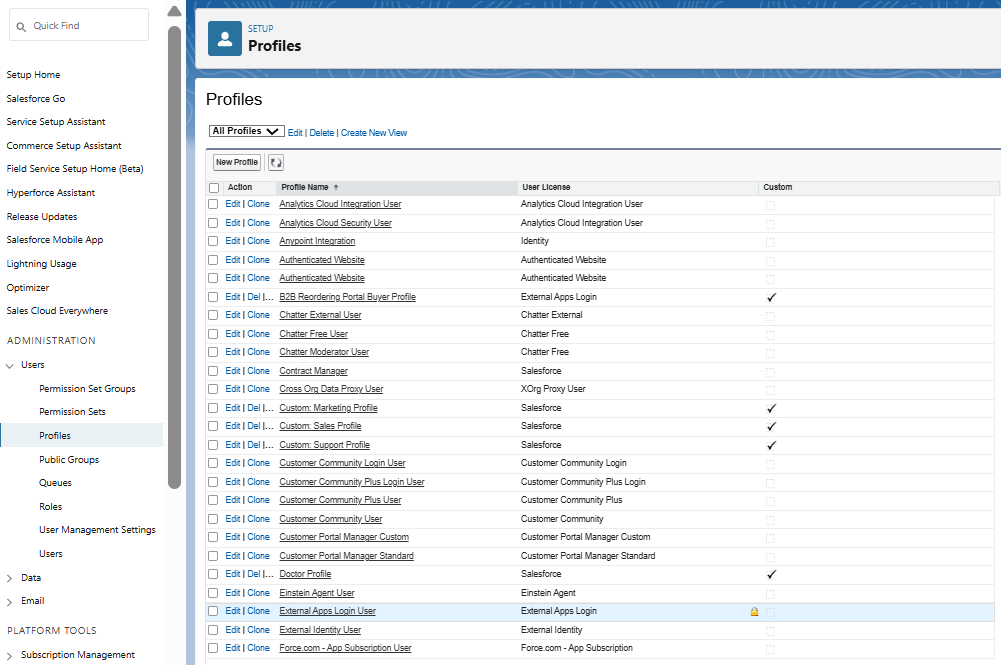
1. ***Fiscal Year Settings***

Setup ----> Fiscal Year ----> Edit Standard Fiscal year



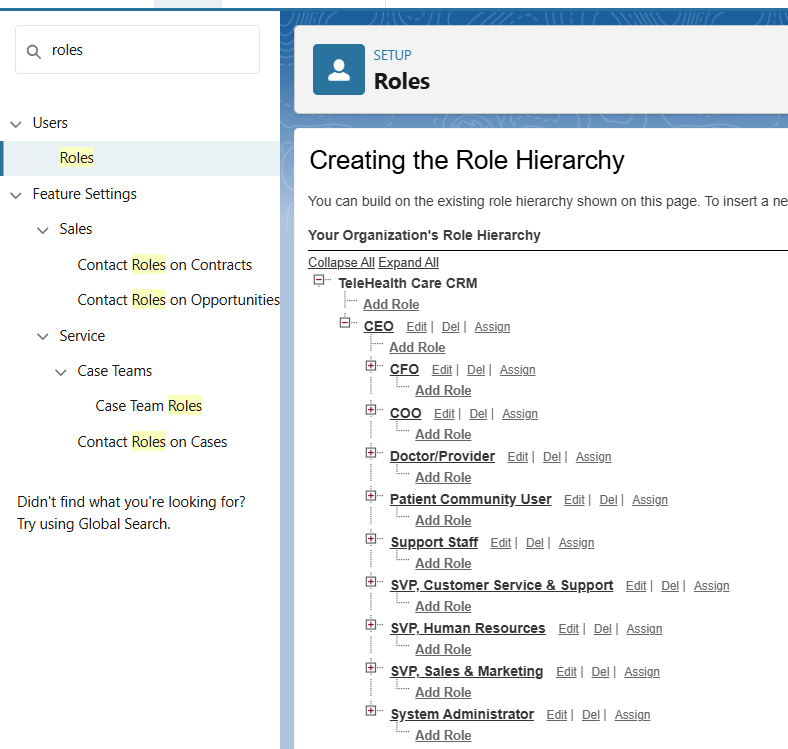
1. ***Profiles***

Setup ----> Users ----> profiles ----> New profile ----> Create Doctor profile, Support Staff, Patient Community



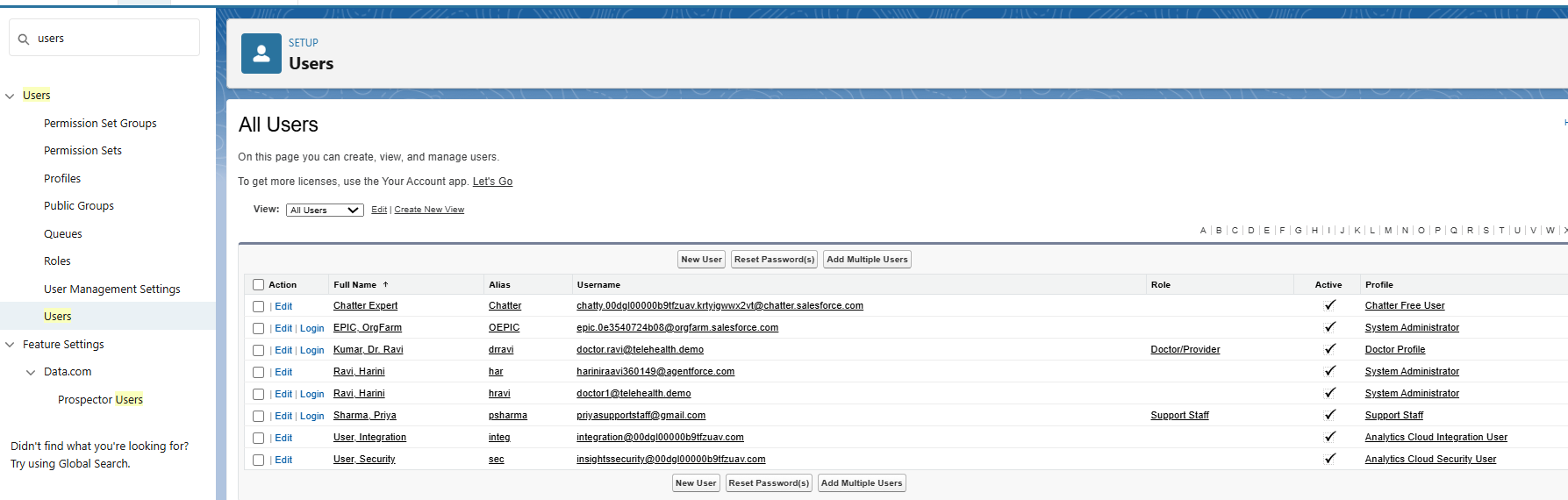
1. ***Roles***

Setup ----> Users ----> Roles ----> Creating the Role Hierarchy ---->Create Add Role ----> Doctor/Provider, Patient community User, Support staff.



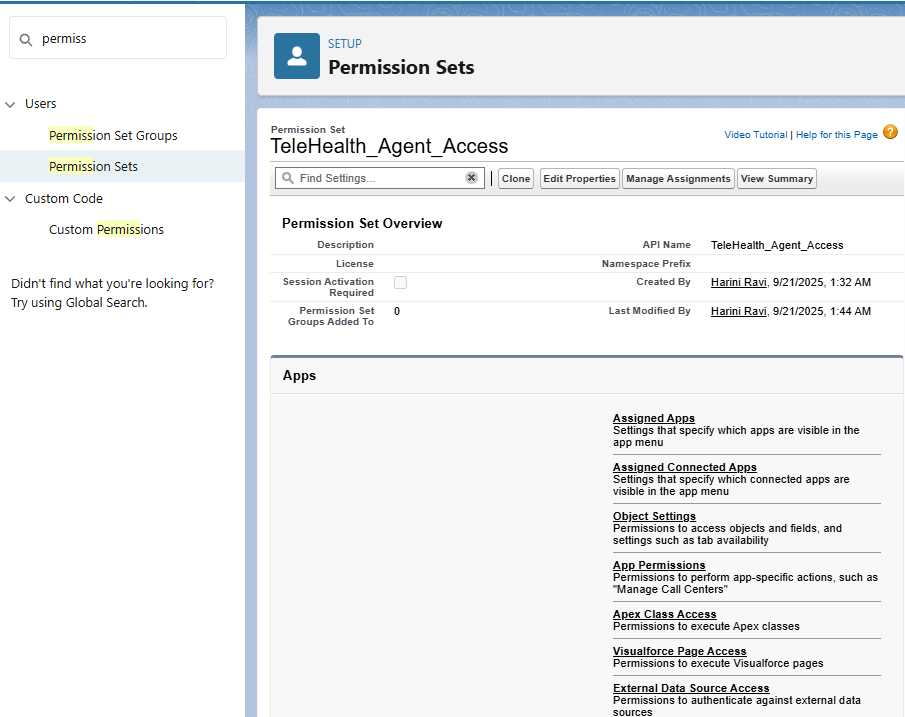
1. ***Create Users***

Setup ----> Users ----> New user ----> Create a test user for each role.



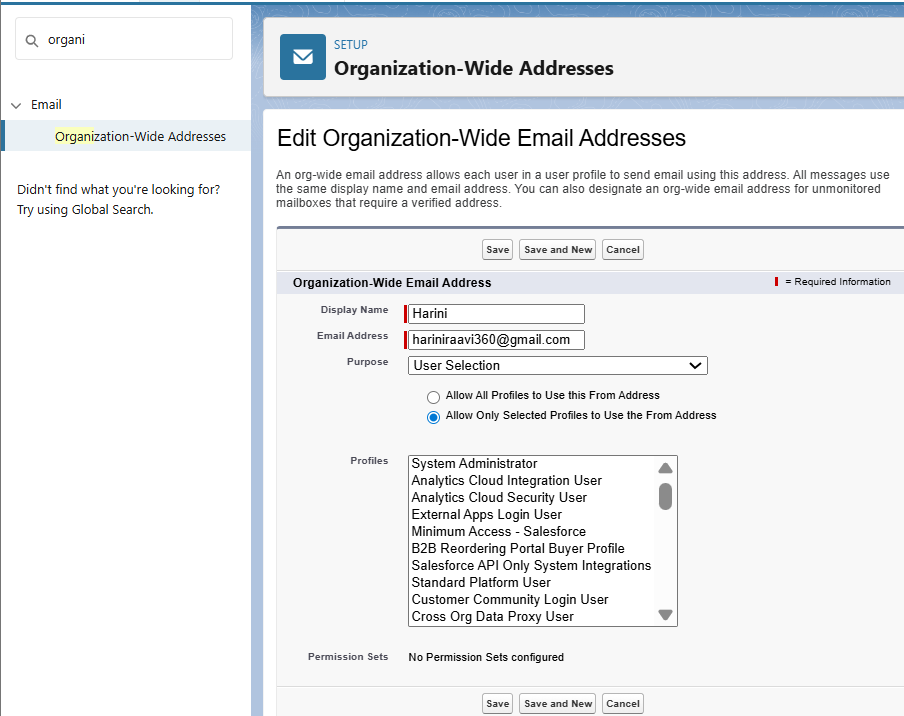
1. ***Permission Sets***

Setup ----> Permission Sets ----> Create new Permission Setup ----> TeleHealth Agent Access



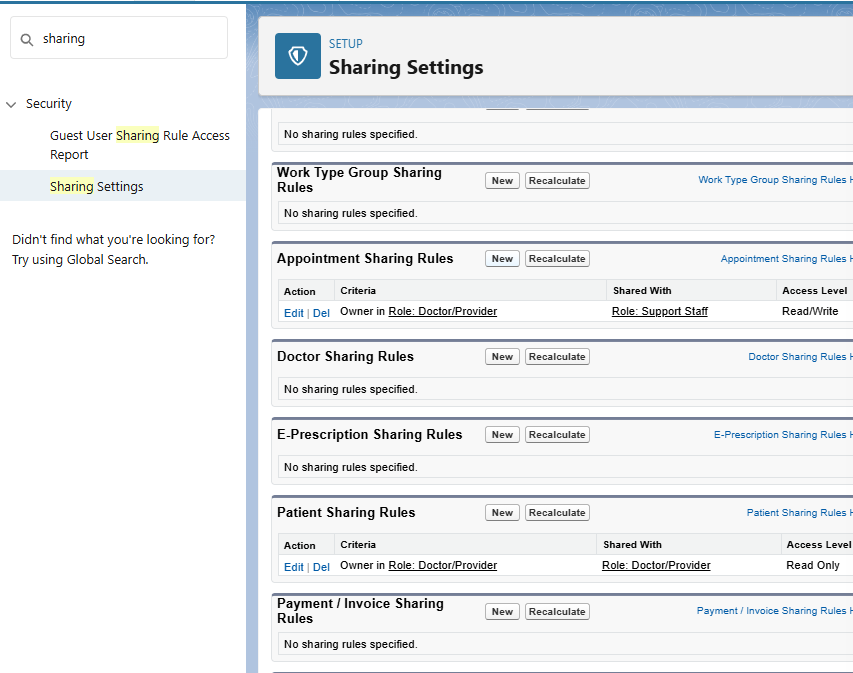
1. ***Organization-Wide Email Addresses(OWD)***

Setup ----> Email ----> OWD ----> Add new Email.



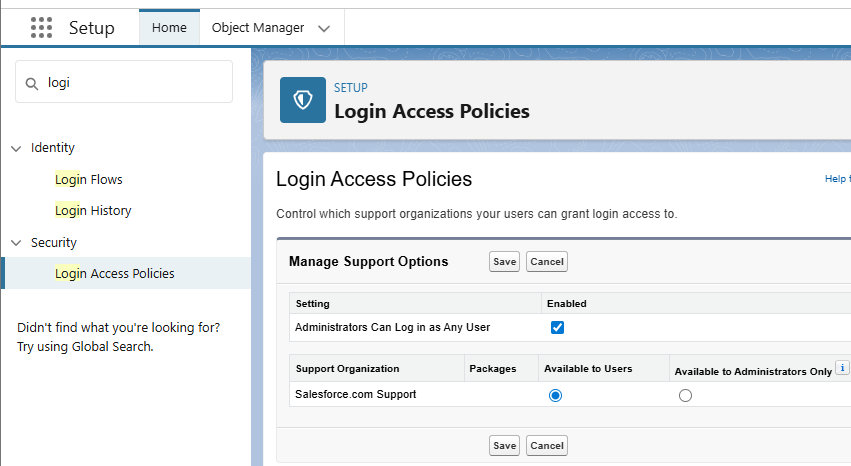
1. ***Sharing rules***

Setup ----> Sharing Settings ----> Create Sharing rules according to each profile type



1. ***Login Access Policies***

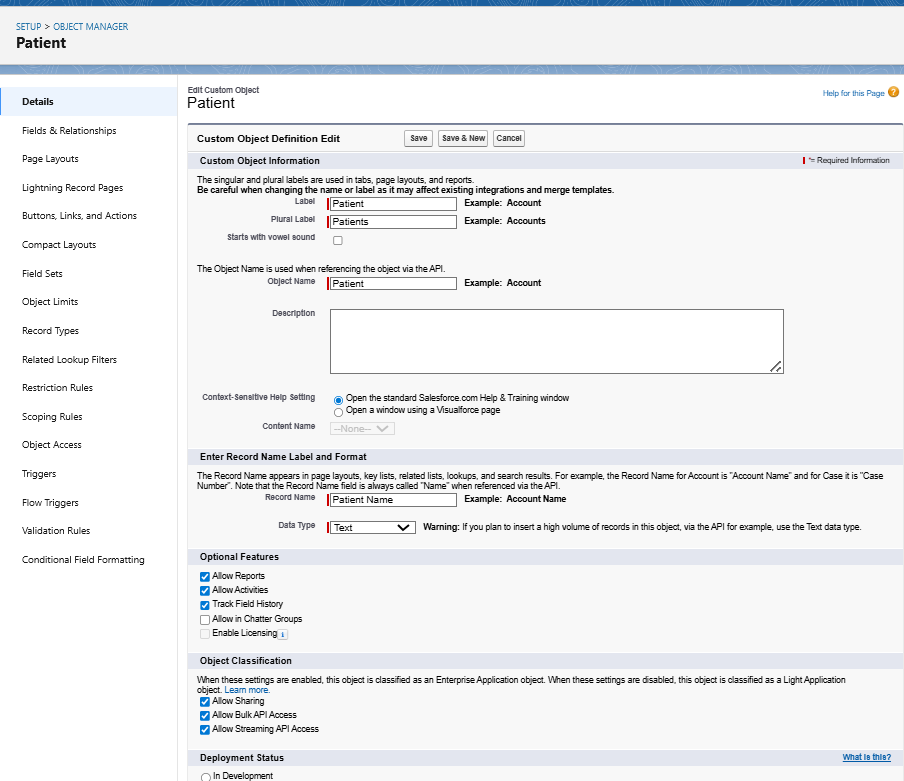
Setup ----> Security ----> Login Access Policies ----> edit Mange Support options.

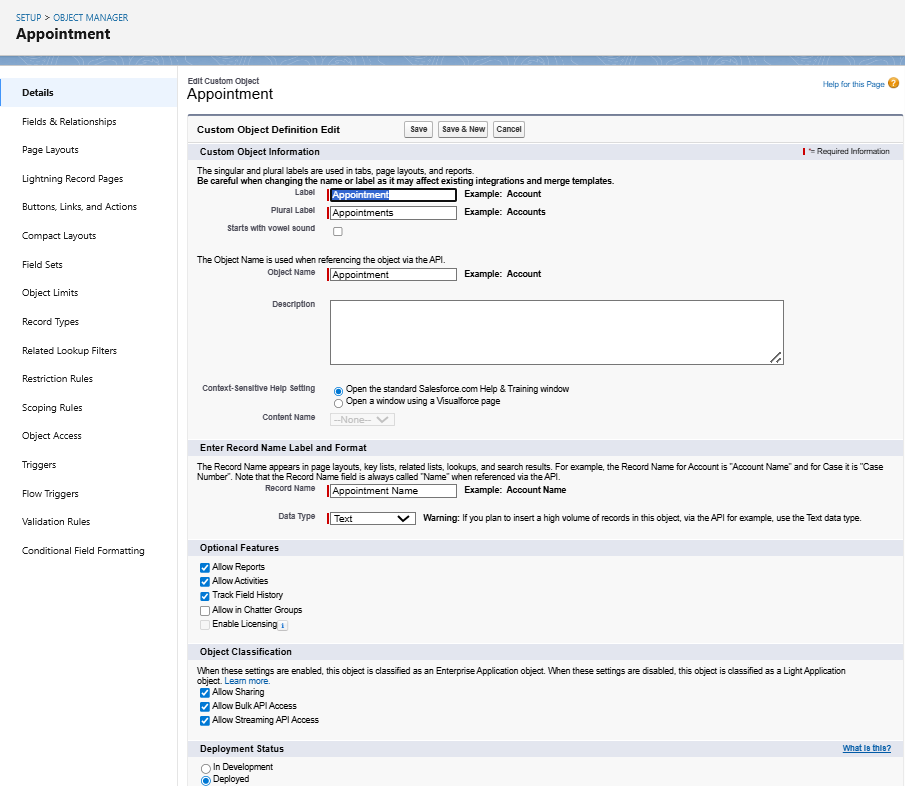


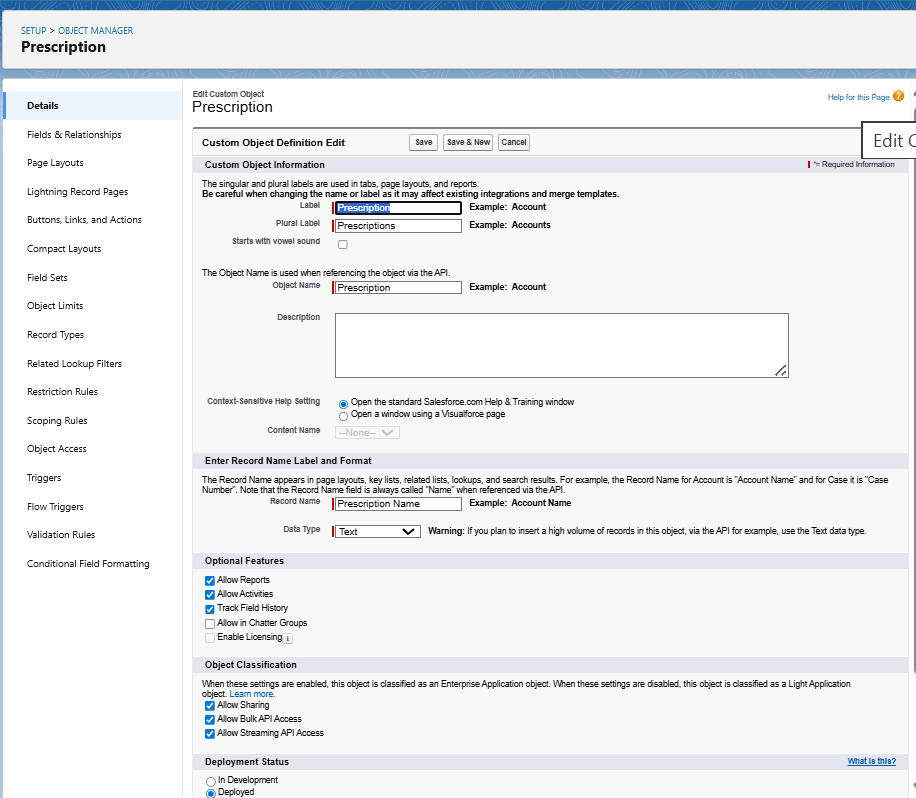
* ***Phase 3: Data Modeling & Relationships***

1. ***Standard & Custom Objects***

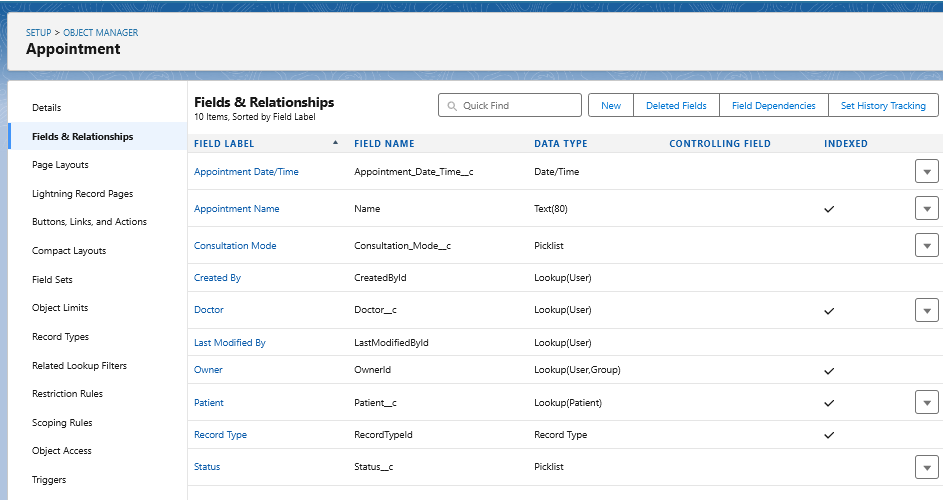
1. Setup → Object Manager → Create → Custom Object.
2. Label ---> Patient, Appointment, Prescription.
3. Plural Label ---> Patients, Appointments, Prescriptions.
4. Check Allow Reports, Allow Activities, Track Field History.

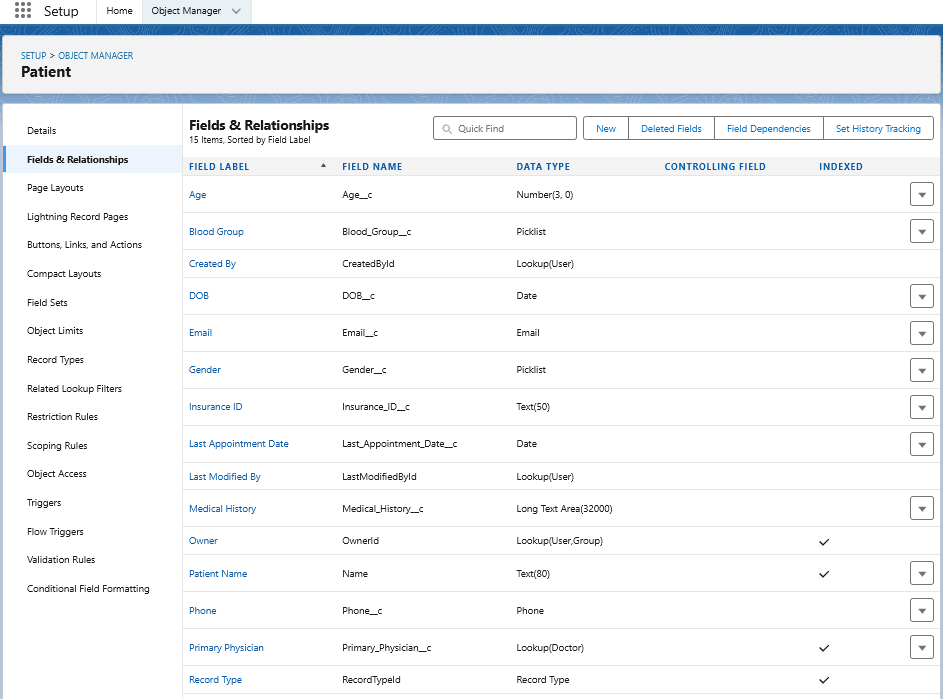


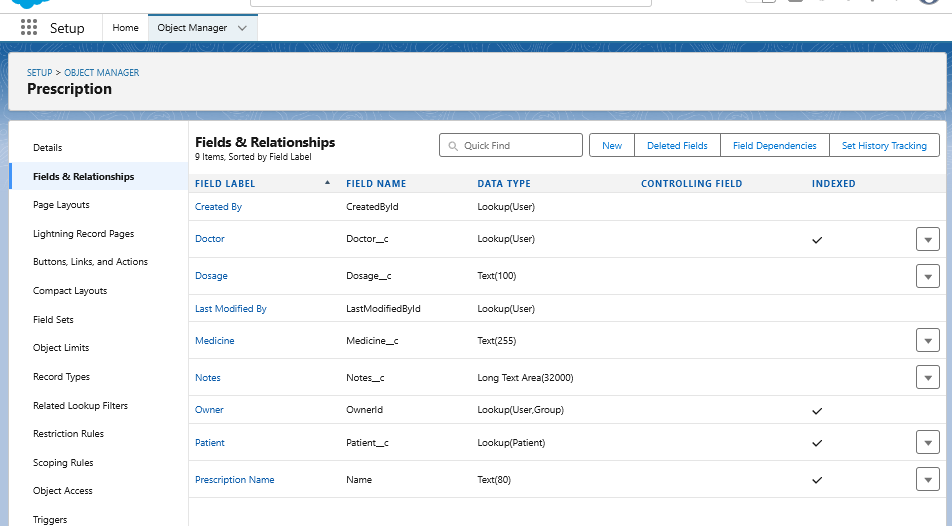




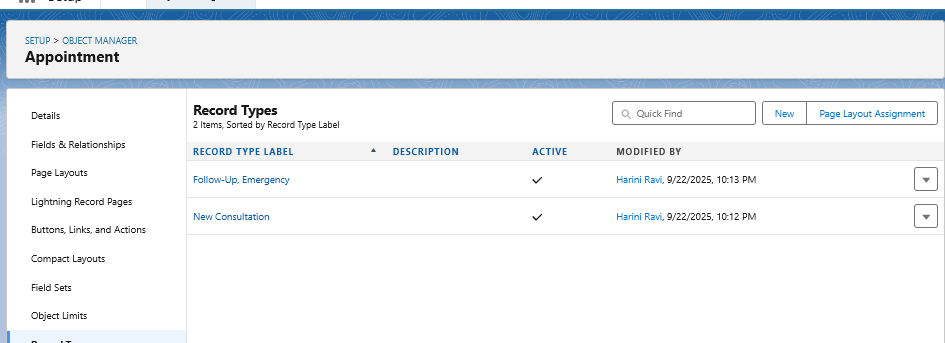
1. ***Fields***
2. Setup → Object Manager → choose object → Fields & Relationships → New.
3. For Patient\_c we use field types are Age, Gender, Phone, Email, Medical History.
4. For Appointment\_c we use fields are Appointment Date/Time , Patient, Doctor, Status.
5. For Prescription\_c we use fields are Patient,Doctor,Medicine,Dosage,NotesMedicine,Dosage,Notes





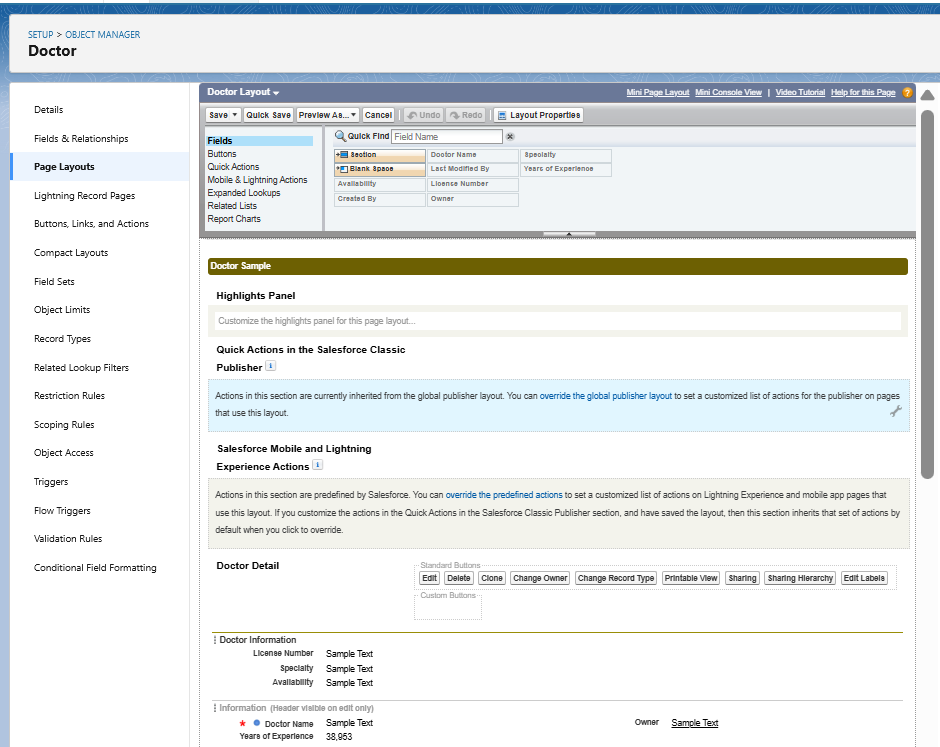


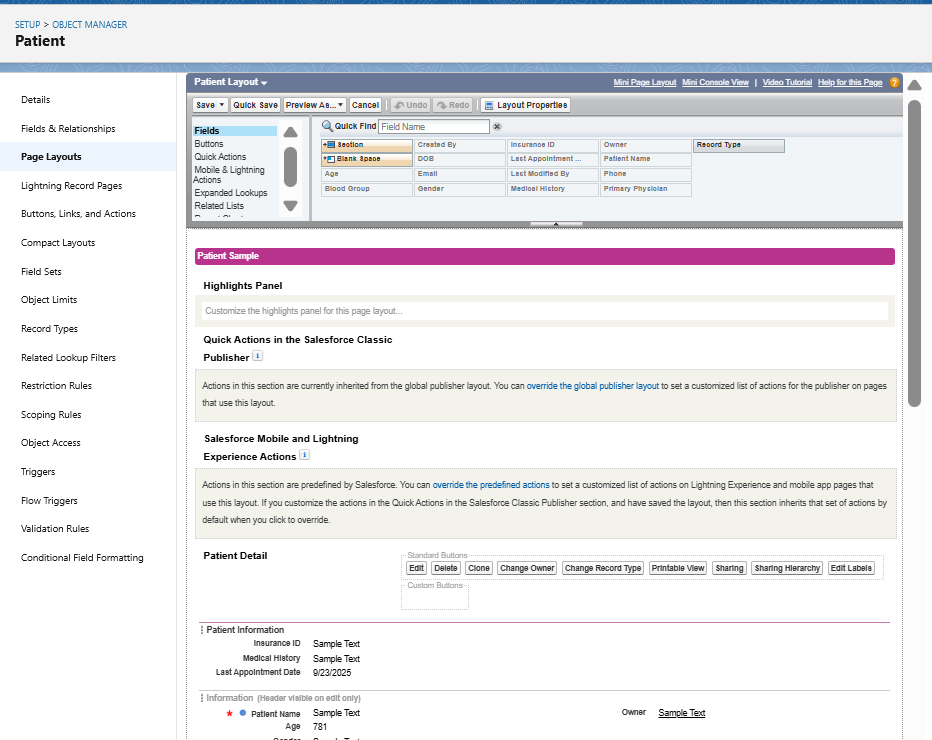
1. ***Record Types***
2. Setup → Object Manager → choose object → Record Types → New.
3. For Appointment create record type as New Consultation, Follow-up , Emergency.
4. For Patient Create record type as Adult, Child, Senior.



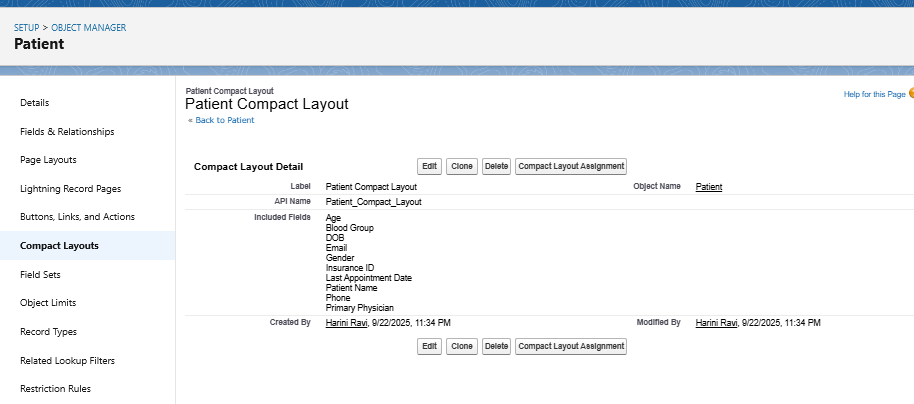


1. ***Page Layouts***
2. Setup ---> Object Manager ---> Select the object ---> page layouts
3. Drag-and-drop fields, sections, related lists, and buttons onto the layout.
4. Doctor Layout → include License Number, Specialty, Availability.
5. Patient Layout → include Insurance Details, Medical History, Last Appointment Date.

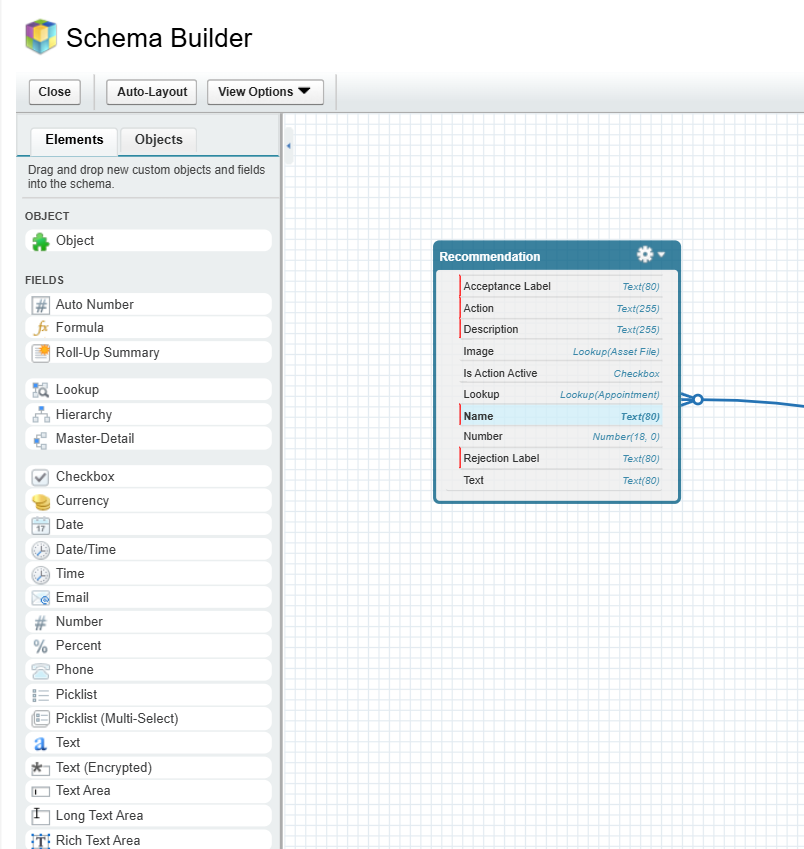




1. ***Compact Layouts***
2. Setup ---> Object Manager ---> Select the object ---> Compact Layouts
3. Create Patient Compact Layout and Add fields.



1. ***Schema Builder***
2. Setup ---> Schema builder ---> Check the objects Patient, Doctor, Appointment.
3. Drag field type --> Text, Number, Lookup.



1. ***Lookup vs Master-Detail vs Hierarchical Relationships***
2. **Lookup**

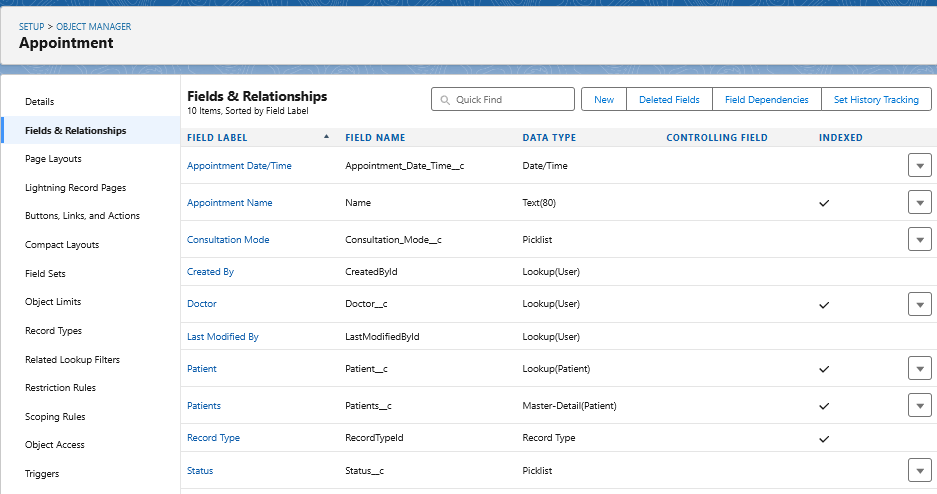
Setup --> Object Manager --> Choose the Object(Appointment) --> Fields and Relationships --> Lookup Relationship--> Doctor.

1. **Master-Detail**

Setup --> Object Manager --> Choose the Object(Appointment) --> Fields and Relationships --> Master-Detail Relationship--> Patient.

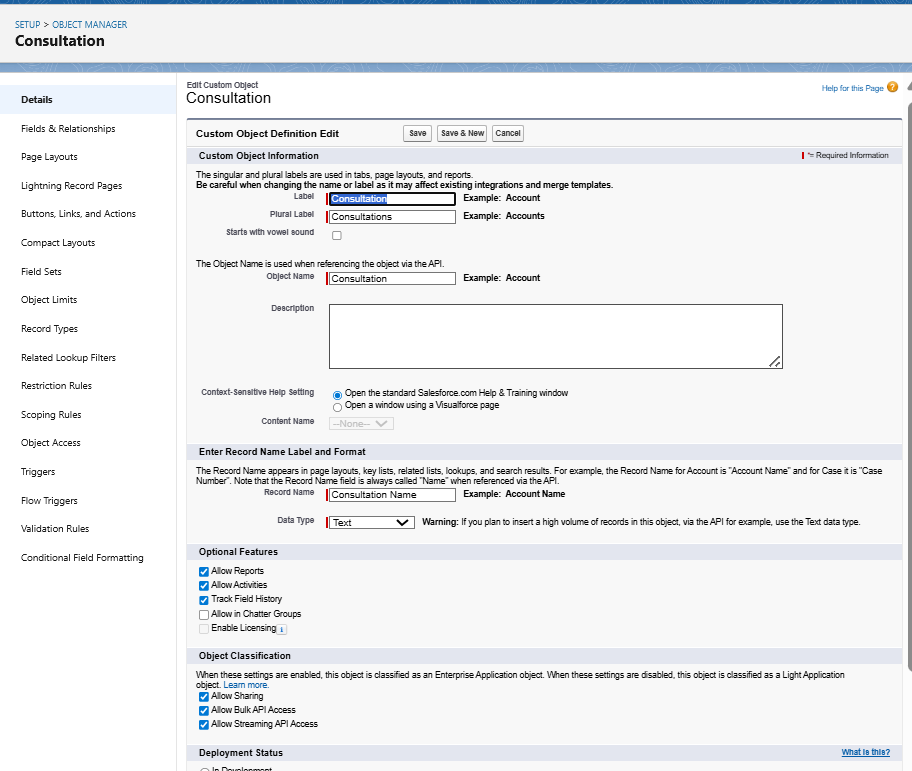
1. **Hierarchical Relationship**

Setup --> Object Manager --> User --> Fields and Relationships --> Hierarchical Relationship--> User.

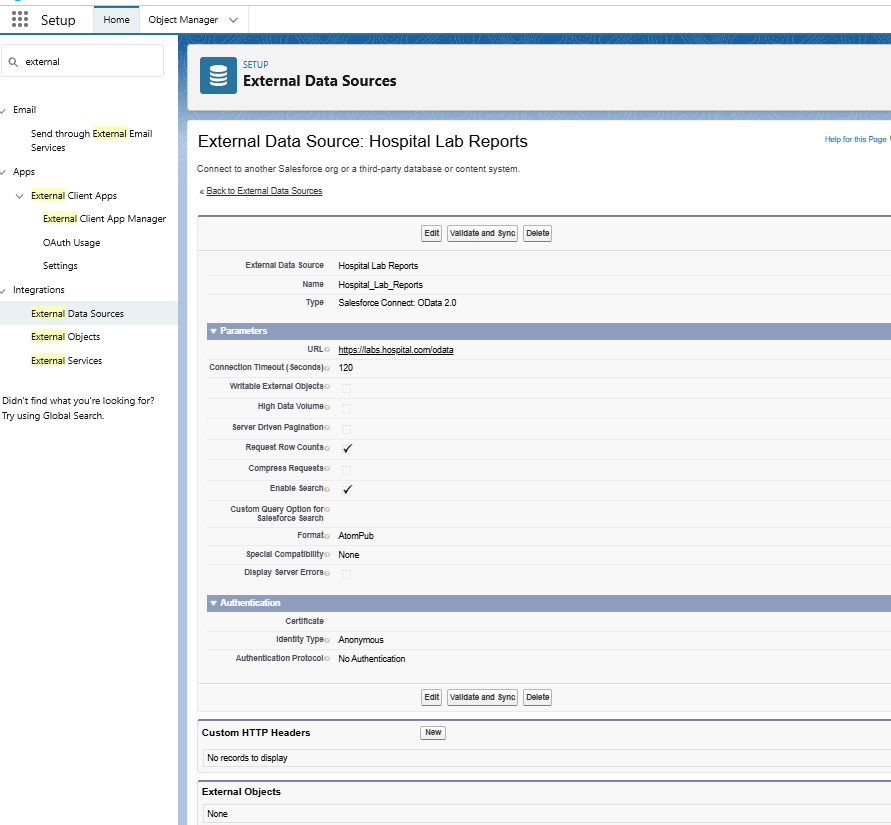


1. ***Junction Objects***

Setup --> object manager --> Create Custom object --> Consultation.



1. ***External Objects***
2. Setup --> External Data Sources --> New External Data Source
3. Name --> Hospital Lab Reports

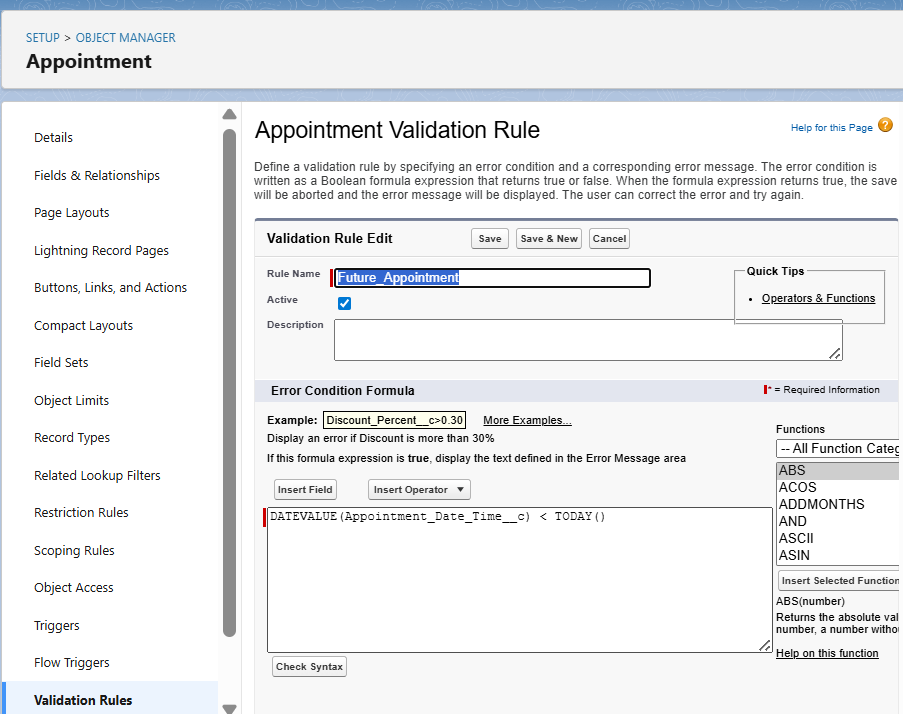


* ***Phase 4 : Process Automation (Admin)***

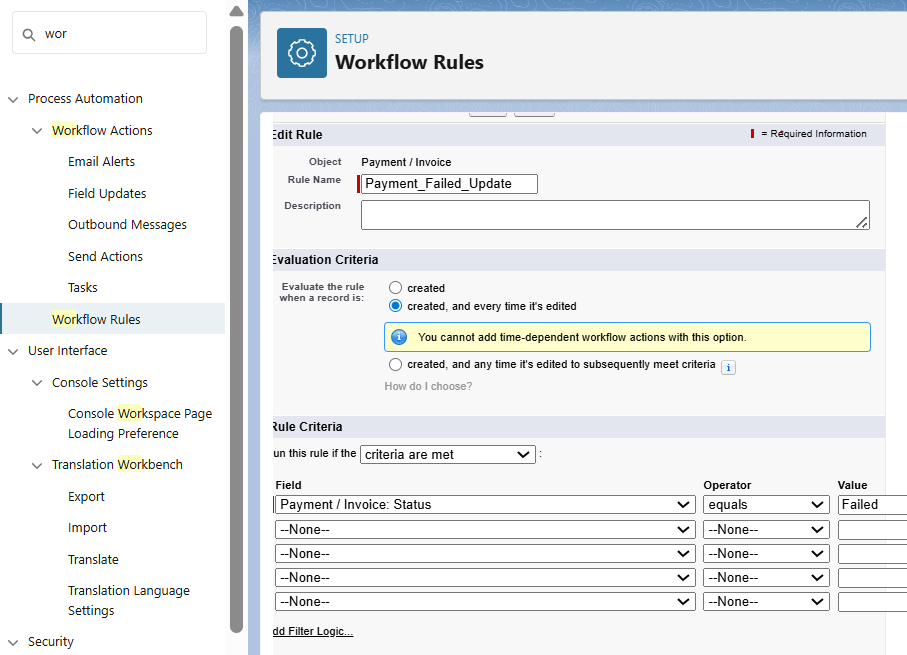
1. ***Validation Rules***
2. Setup → Object Manager → Appointment\_\_c → Validation Rules → New.
3. **Formula :**

Appointment\_Date\_\_c < TODAY()

1. Error Message: “Appointment date cannot be in the past.”
2. Set Error Location (field or page).



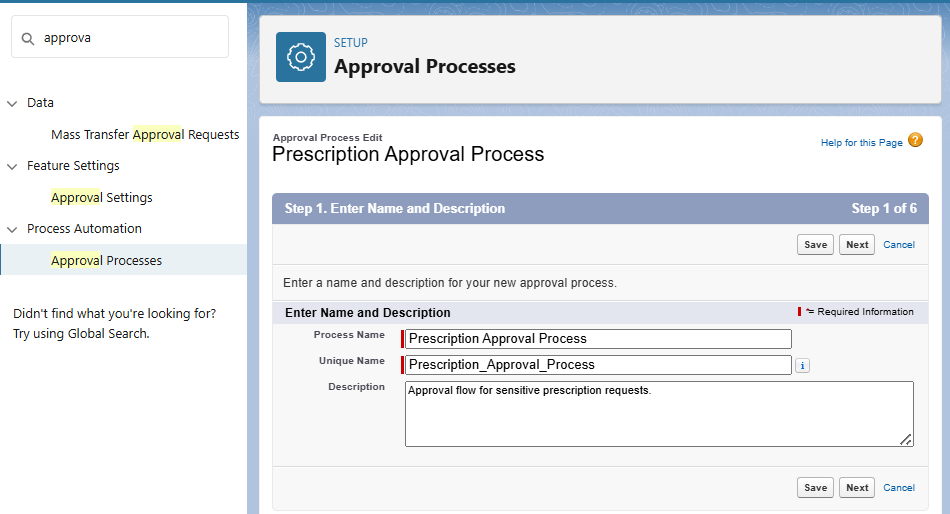
1. ***Workflow Rules***
2. Setup → Workflow Rules → New Rule.
3. Select Object.
4. Set Evaluation Criteria → “created, and every time it’s edited.”
5. Add Rule Criteria.
6. Add Action → Email Alert / Field Update / Task.
7. Save → Activate.



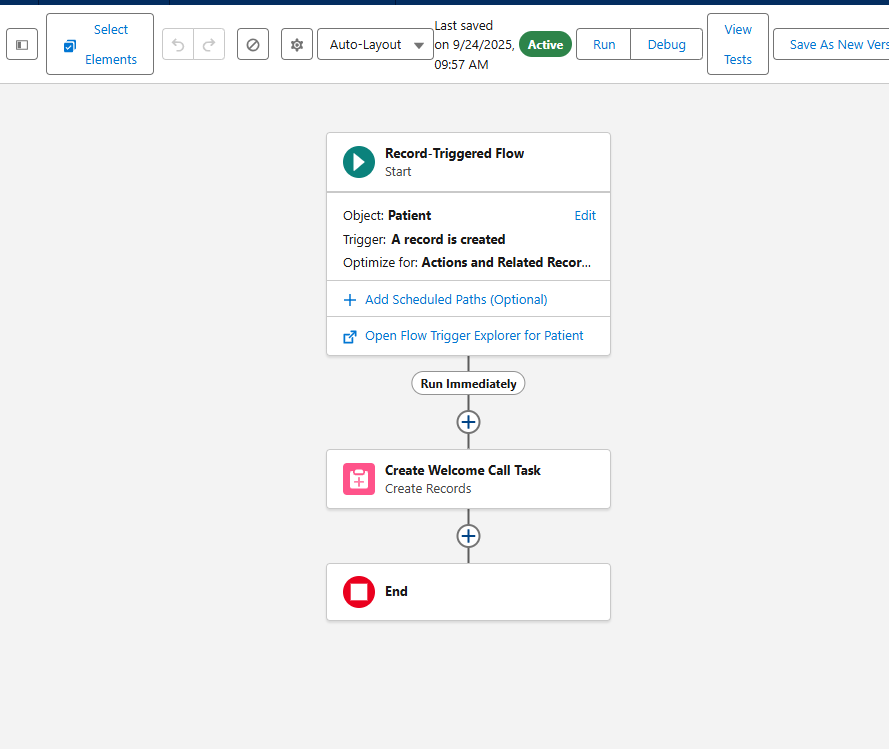
1. ***Process Builder***
2. Setup → Process Builder → New***.***
3. Enter Process Name → Choose Object
4. Define Trigger → “when record is created or edited.”
5. Add Criteria (e.g., always true).
6. Add Immediate Action → Update Records.
7. Save → Activate.



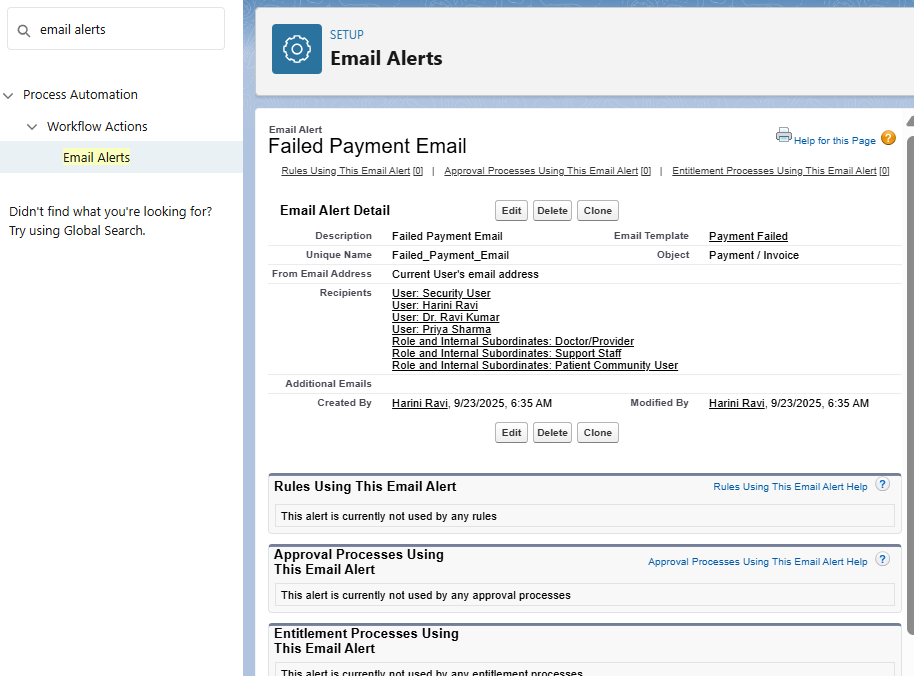
1. ***Approval Process***
2. Setup → Approval Processes → Choose Object (e.g., Payment\_\_c).
3. Create New Approval Process → Use Standard Setup Wizard.
4. Set Entry Criteria
5. Define Approver(s) (specific user, role, or queue).
6. Configure Initial Submission Actions (lock record, send email).
7. Configure Final Approval/Final Rejection Actions.
8. Save → **Activate**.



1. ***Flow Builder***
2. Setup → Flows → New Flow.
3. Choose Flow Type:
4. **Screen Flow:** interactive UI.
5. **Record-Triggered Flow:** runs on create/update/delete.
6. **Scheduled Flow:** runs on schedule.
7. **Auto-launched Flow:** runs in background.
8. Save → **Activate**.

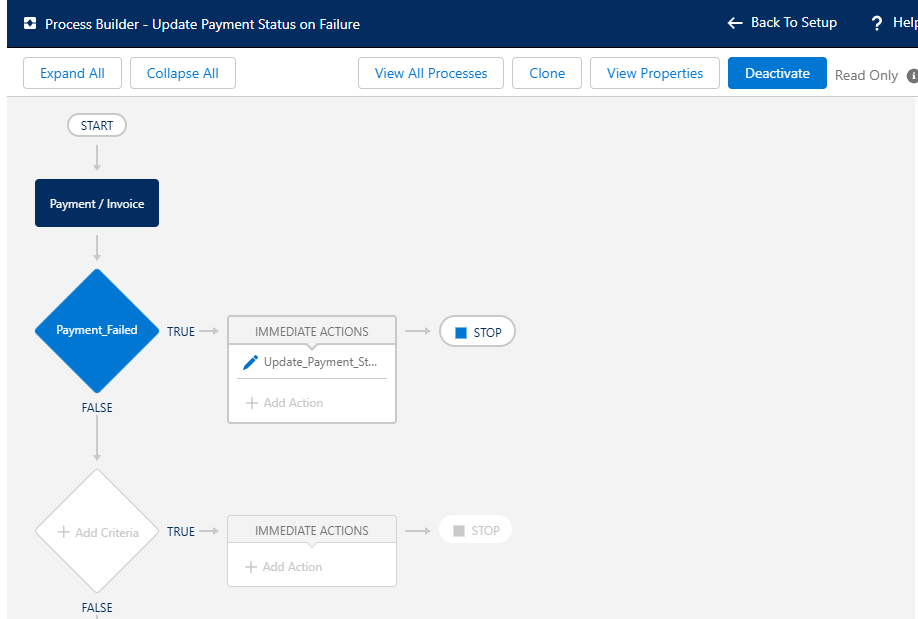


1. ***Email Alerts***
2. Setup → Email Alerts → New.
3. Select Object.
4. Choose Email Template + Recipient field.
5. Attach this to Workflow / Process Builder / Flow.
6. Save.

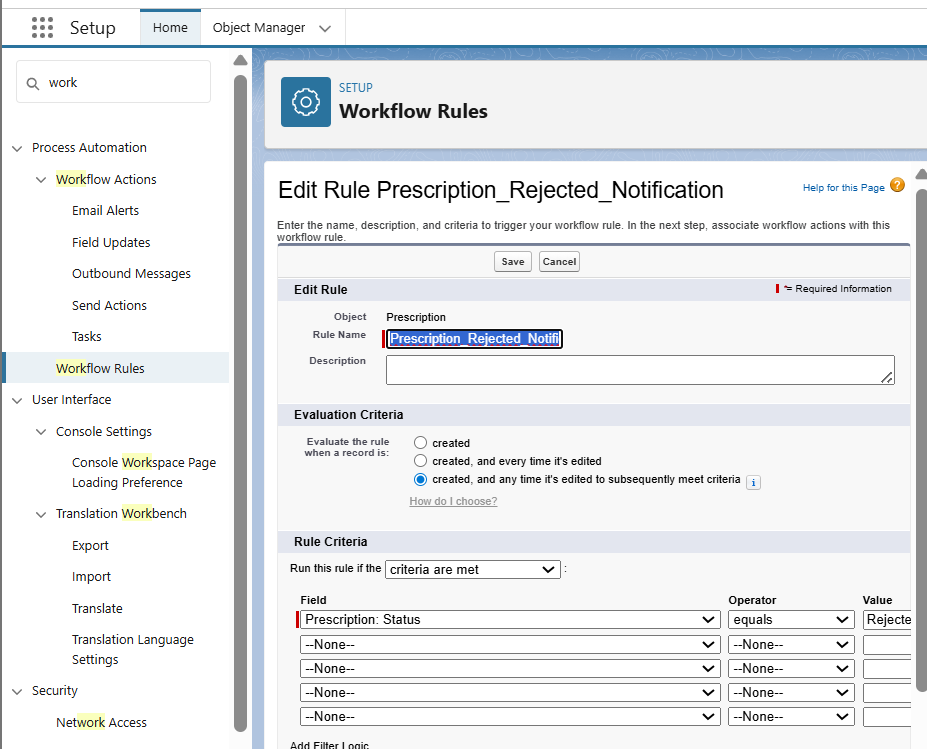


1. ***Field Updates***
2. Inside Workflow/Process Builder/Flow → Add Action → Field Update.
3. Choose Target Field.
4. Set New Value.
5. Save.



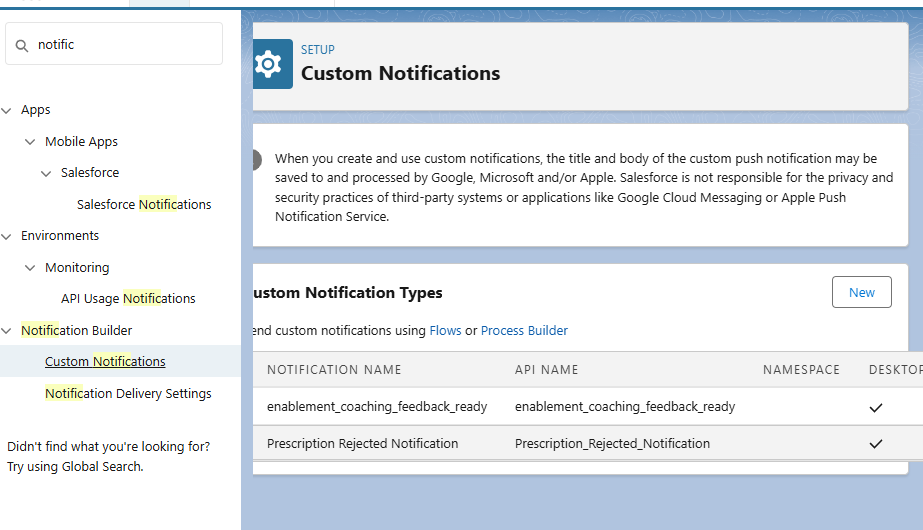


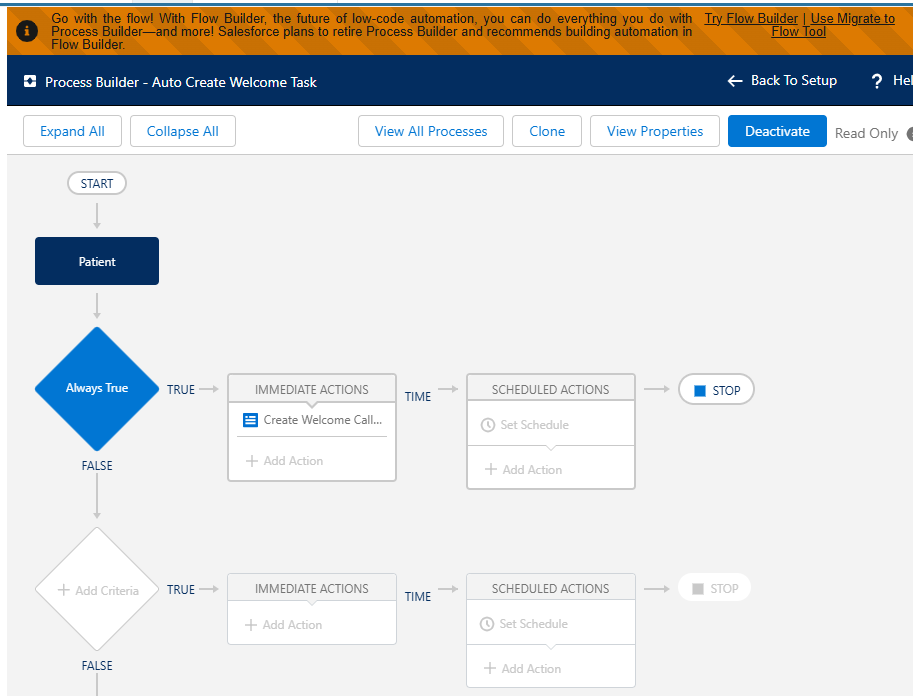
1. ***Tasks***
2. Inside Workflow/Process Builder/Flow → Add Action → Create Task.
3. Enter Subject.
4. Assign To (User/Queue).
5. Set Due Date.
6. Save.





1. ***Custom Notifications***
2. Setup → Notification Builder → Custom Notifications → New.
3. Enter Name + API Name.
4. In Flow → Add Action → Send Custom Notification.
5. Select Recipient.
6. Save → Activate.





* ***Phase 5: Apex Programming (Developer)***

1. ***Classes & Objects + Apex Classes***

* **Appex classes**

1. Create the patient helper class (PatientHelper)
2. In developer console ---> File ----->New ---> Apex Classs.

**Code :**

public class PatientHelper {

public static void assignCareCoordinator(List<Patient\_\_c> patientList) {

if (patientList == null || patientList.isEmpty()) {

return;

}

User doctor = [

SELECT Id

FROM User

WHERE Username = 'dr.ravikumar@example.com'

LIMIT 1

];

for (Patient\_\_c p : patientList) {

if (p.Region\_\_c == 'East') {

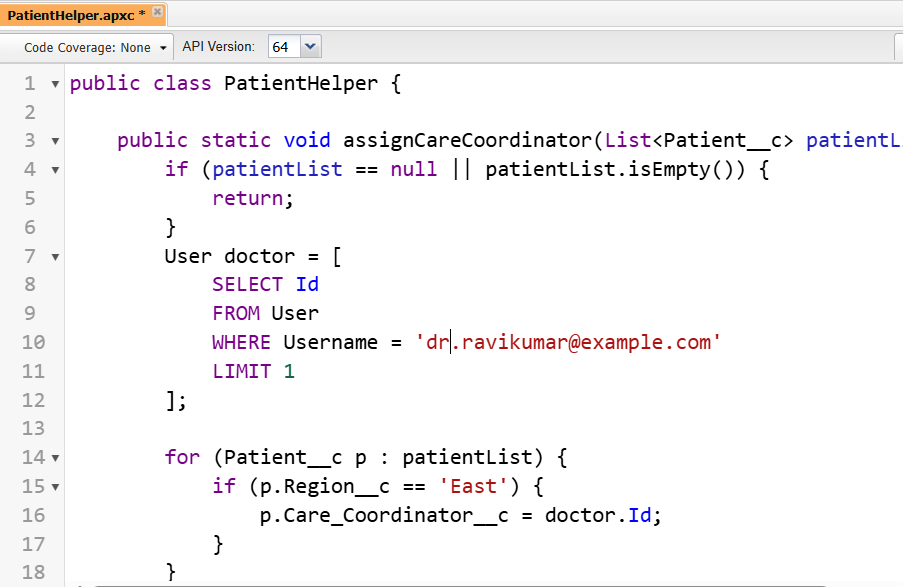
p.Care\_Coordinator\_\_c = doctor.Id;

}

}

}

}



* Save & test using Developer Console → Execute Anonymous:

**Apex Code :**

List<Patient\_\_c> patients = [

SELECT Id, Region\_\_c

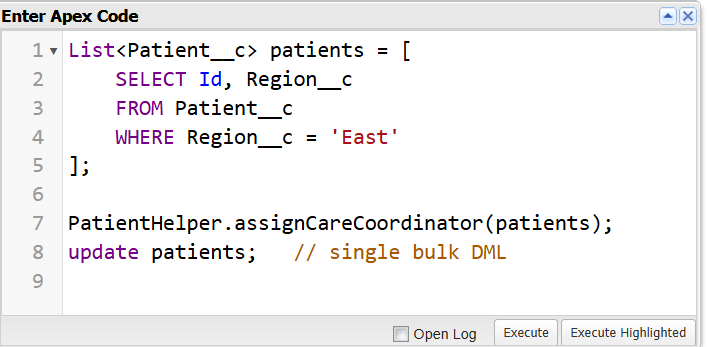
FROM Patient\_\_c

WHERE Region\_\_c = 'East'

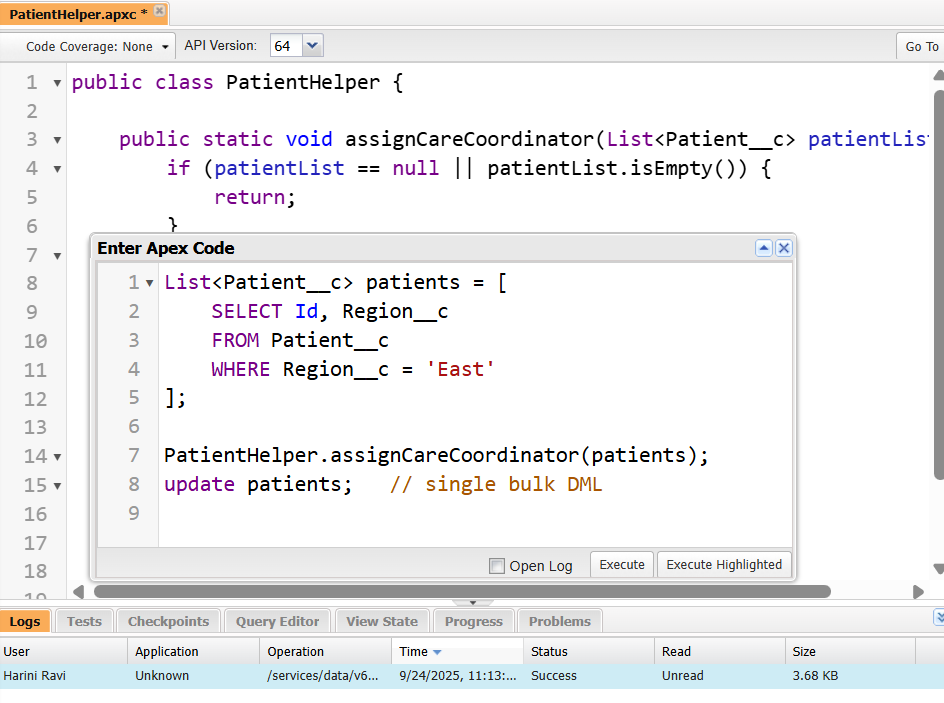
];

PatientHelper.assignCareCoordinator(patients);

update patients; // single bulk DML



***Result :***



1. ***Create the trigger (Patient)***

In Developer console --> Apex triggers --> New --> Patient Trigger

***Code :***

trigger PatientTrigger on Patient\_\_c (before insert, before update) {

List<Patient\_\_c> patientsToAssign = new List<Patient\_\_c>();

for (Patient\_\_c p : Trigger.new) {

if (p.Care\_Coordinator\_\_c != null) continue;

if (p.Region\_\_c == 'East' &&

(Trigger.isInsert ||

(Trigger.isUpdate &&

p.Region\_\_c != Trigger.oldMap.get(p.Id).Region\_\_c))) {

patientsToAssign.add(p);

}

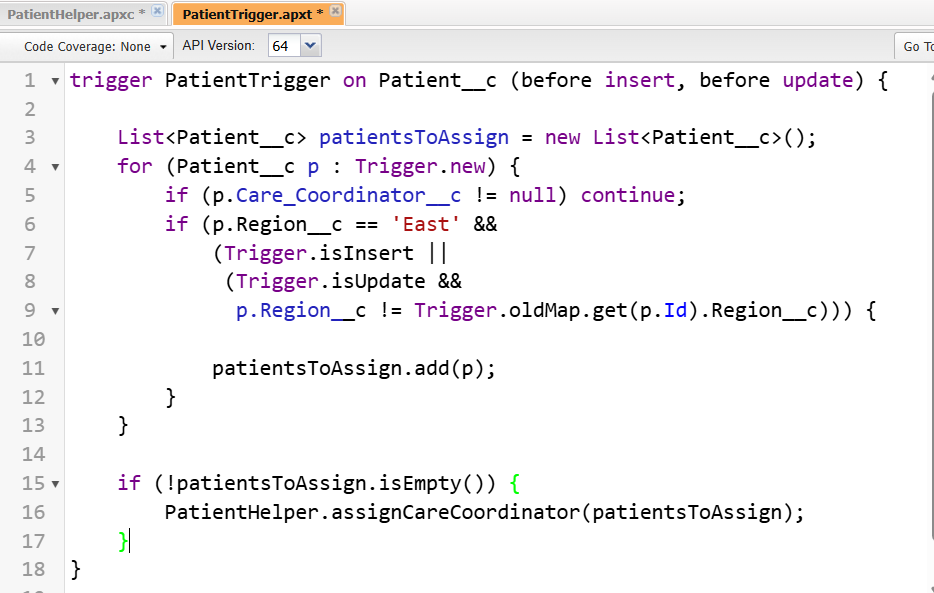
}

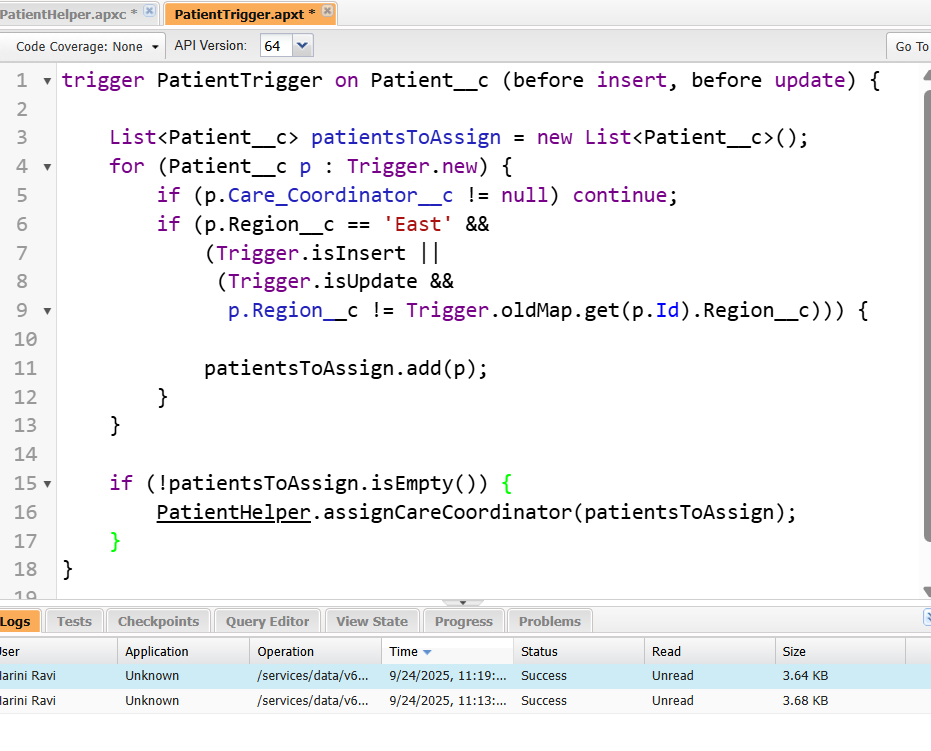
if (!patientsToAssign.isEmpty()) {

PatientHelper.assignCareCoordinator(patientsToAssign);

}

}





1. ***Test Classes***

In developer console---> File----->New--->Apex Classs.

***Name :*** PatientTriggerTest

***Code :***

@isTest

private class PatientTriggerTest {

// Create a test user with FirstName='Dr.' and LastName='Ravi Kumar'

private static User createDoctor() {

// Find a profile to use (Standard User if available, otherwise first profile)

Profile prof;

List<Profile> profs = [SELECT Id FROM Profile WHERE Name = 'Standard User' LIMIT 1];

if (profs.isEmpty()) {

prof = [SELECT Id FROM Profile LIMIT 1];

} else {

prof = profs[0];

}

// Make username/email unique for test run

String uniq = String.valueOf(DateTime.now().getTime());

User u = new User(

FirstName = 'Dr.',

LastName = 'Ravi Kumar',

Alias = 'drk' + uniq.substring(uniq.length()-3),

Email = 'dr.ravikumar.' + uniq + '@test.com',

Username = 'dr.ravikumar.' + uniq + '@test.com',

ProfileId = prof.Id,

TimeZoneSidKey = 'Asia/Kolkata',

LocaleSidKey = 'en\_US',

EmailEncodingKey = 'UTF-8',

LanguageLocaleKey = 'en\_US'

);

insert u;

return u;

}

@isTest

static void testAssignCareCoordinator() {

// Arrange

User doctor = createDoctor();

// Prepare patients

Patient\_\_c eastPatient = new Patient\_\_c(

Name = 'East Patient',

Region\_\_c = 'East'

);

Patient\_\_c westPatient = new Patient\_\_c(

Name = 'West Patient',

Region\_\_c = 'West'

);

Test.startTest();

insert new List<Patient\_\_c>{ eastPatient, westPatient };

Test.stopTest();

// Reload and assert

eastPatient = [SELECT Id, Care\_Coordinator\_\_c, Region\_\_c FROM Patient\_\_c WHERE Id = :eastPatient.Id];

westPatient = [SELECT Id, Care\_Coordinator\_\_c, Region\_\_c FROM Patient\_\_c WHERE Id = :westPatient.Id];

System.assertEquals(doctor.Id, eastPatient.Care\_Coordinator\_\_c,

'East patient should be assigned to the doctor created in test.');

System.assertEquals(null, westPatient.Care\_Coordinator\_\_c,

'West patient should not be assigned on insert.');

// Now update West -> East and verify assignment on update

westPatient.Region\_\_c = 'East';

Test.startTest();

update westPatient;

Test.stopTest();

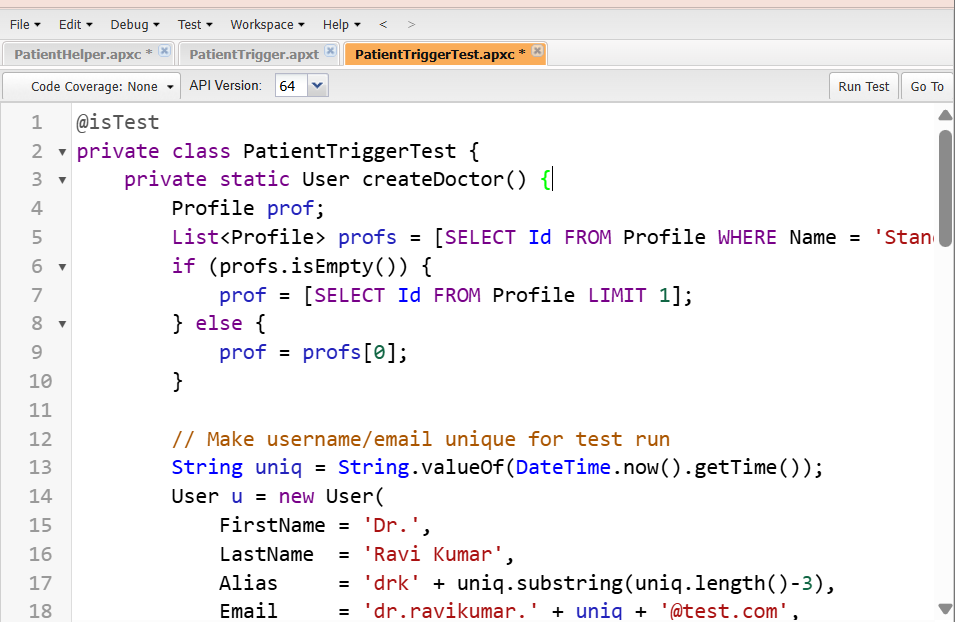
westPatient = [SELECT Id, Care\_Coordinator\_\_c, Region\_\_c FROM Patient\_\_c WHERE Id = :westPatient.Id];

System.assertEquals(doctor.Id, westPatient.Care\_Coordinator\_\_c,

'After changing Region to East, the patient should be assigned the doctor.');

}

}





1. ***SOQL & SOSL***

* **Using SOQL in Helper Classes**

In developer console---> File----->New--->Apex Classs.

**Name :** AppointmentHelper

**Code :**

public with sharing class AppointmentHelper {

public static void assignDefaultDoctor(List<Appointment\_\_c> apptList) {

if (apptList == null || apptList.isEmpty()) return;

User defaultDoc = [

SELECT Id

FROM User

WHERE FirstName = 'Dr.' AND LastName = 'Ravi Kumar'

LIMIT 1

];

for (Appointment\_\_c a : apptList) {

if (a.Doctor\_\_c == null) { // only if not already assigned

a.Doctor\_\_c = defaultDoc.Id; // assign the queried Id

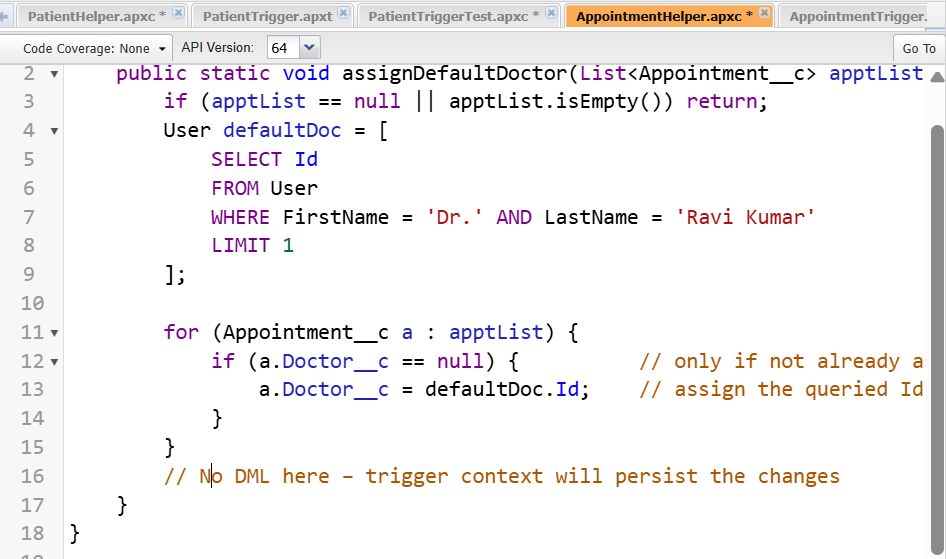
}

}

// No DML here – trigger context will persist the changes

}

}



* ***Apex Triggers***

***Code :***

trigger AppointmentTrigger on Appointment\_\_c (before insert, before update) {

List<Appointment\_\_c> toProcess = new List<Appointment\_\_c>();

for (Appointment\_\_c a : Trigger.new) {

if (Trigger.isInsert) {

toProcess.add(a);

} else if (Trigger.isUpdate &&

a.Status\_\_c == 'Confirmed' &&

a.Status\_\_c != Trigger.oldMap.get(a.Id).Status\_\_c) {

toProcess.add(a);

}

}

if (!toProcess.isEmpty()) {

AppointmentHelper.assignDefaultDoctor(toProcess);

}

}



* **Using SOSL in Helper Classes**

In developer console---> File----->New--->Apex Classs.

***Name :***SearchHealper

***Code :***

public with sharing class SearchHelper {

/\*\*

\* Performs a SOSL search for a keyword across Patient\_\_c and Appointment\_\_c.

\* Returns a Map with SObject lists.

\*

\* @param keyword Text to search

\* @return Map<String, List<SObject>> Keys: 'Patient', 'Appointment'

\*/

public static Map<String, List<SObject>> searchPatientsAndAppointments(String keyword) {

Map<String, List<SObject>> resultMap = new Map<String, List<SObject>>{

'Patient' => new List<SObject>(),

'Appointment' => new List<SObject>()

};

if (String.isBlank(keyword)) return resultMap;

// Perform SOSL query

List<List<SObject>> searchResults = [

FIND :keyword IN ALL FIELDS

RETURNING

Patient\_\_c(Id, Name, Region\_\_c),

Appointment\_\_c(Id, Name, Status\_\_c)

];

// Assign results to map

if (!searchResults.isEmpty()) {

// Patient\_\_c results

if (searchResults.size() > 0 && searchResults[0] != null) {

resultMap.put('Patient', searchResults[0]);

}

// Appointment\_\_c results

if (searchResults.size() > 1 && searchResults[1] != null) {

resultMap.put('Appointment', searchResults[1]);

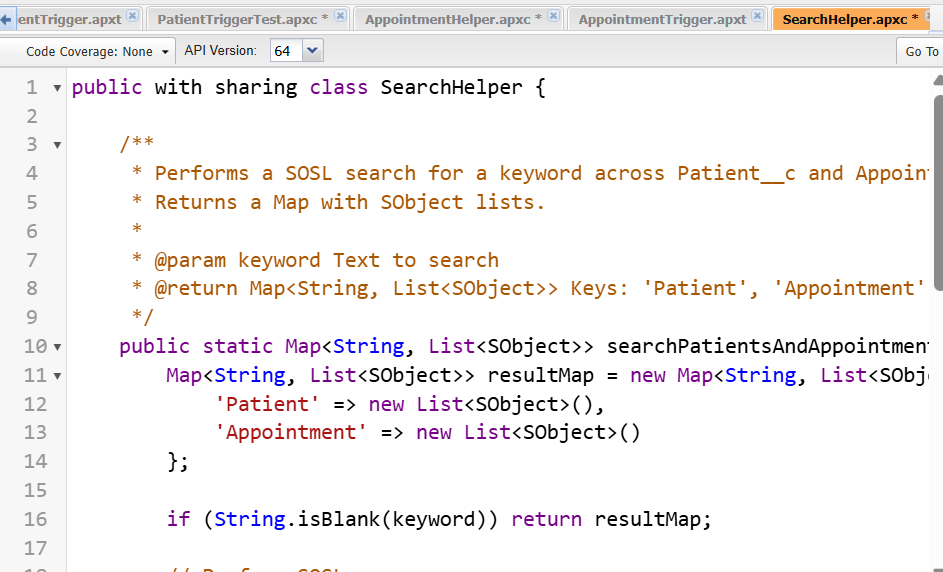
}

}

return resultMap;

}

}



1. ***Collections: List, Set, Map***

* **List:** Ordered collection.
* **Set:** Unique items, no duplicates.
* **Map:** Key-value pair, fast lookup.

***Code :***

List<Patient\_\_c> patients = new List<Patient\_\_c>();

Set<Id> patientIds = new Set<Id>();

Map<Id, Patient\_\_c> patientMap = new Map<Id, Patient\_\_c>();

1. ***Batch Apex***

In developer console---> File----->New--->Apex Classs.

* **Purpose** : Process large data asynchronously in batches.
* **Name :** AppointmentReminderBatch

***Code :***

global class AppointmentReminderBatch implements Database.Batchable<SObject>, Database.Stateful {

// Optional: track total processed count

global Integer totalProcessed = 0;

// Query to select appointments to process

global Database.QueryLocator start(Database.BatchableContext bc) {

return Database.getQueryLocator([

SELECT Id, Name, Status\_\_c, Region\_\_c, Reminder\_Sent\_\_c

FROM Appointment\_\_c

WHERE Status\_\_c = 'Scheduled' AND Region\_\_c = 'East'

]);

}

// Process each batch of records

global void execute(Database.BatchableContext bc, List<Appointment\_\_c> scope) {

for (Appointment\_\_c appt : scope) {

appt.Reminder\_Sent\_\_c = true; // Update the reminder field

}

update scope; // Bulk-safe DML

totalProcessed += scope.size();

}

// Optional: finish method for post-processing

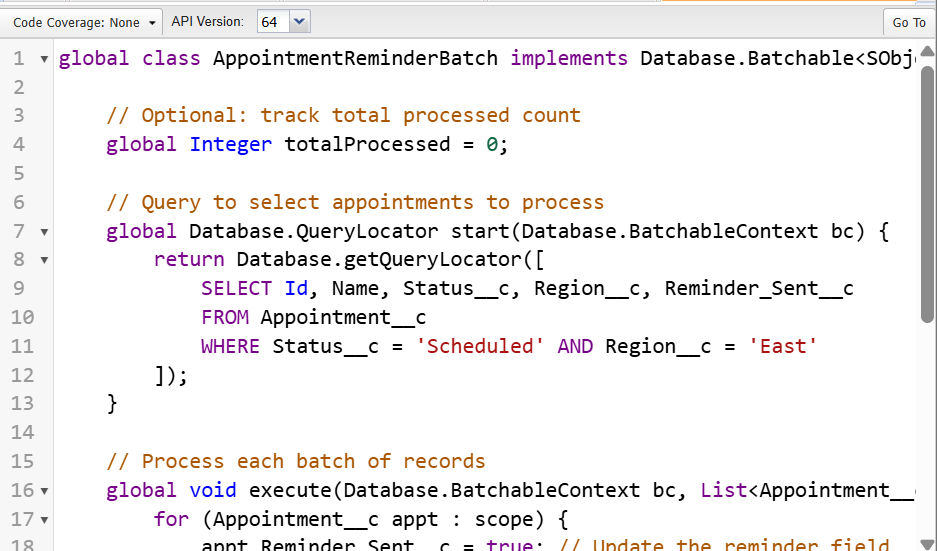
global void finish(Database.BatchableContext bc) {

System.debug('Total Appointments processed: ' + totalProcessed);

// You could also send an email notification to admin here if needed

}

}



1. ***Queueable Apex***

* **Purpose :** Asynchronous processing, can chain jobs.

***Code :***

public class AppointmentReminderQueueable implements Queueable {

private String region; // Region parameter

// Constructor to pass region dynamically

public AppointmentReminderQueueable(String region) {

this.region = region;

}

// Execute method runs asynchronously

public void execute(QueueableContext context) {

// Query appointments that are Scheduled and match the region

List<Appointment\_\_c> apptsToUpdate = [

SELECT Id, Name, Status\_\_c, Region\_\_c, Reminder\_Sent\_\_c

FROM Appointment\_\_c

WHERE Status\_\_c = 'Scheduled' AND Region\_\_c = :region

];

// Update Reminder\_Sent\_\_c field

for (Appointment\_\_c appt : apptsToUpdate) {

appt.Reminder\_Sent\_\_c = true;

}

if (!apptsToUpdate.isEmpty()) {

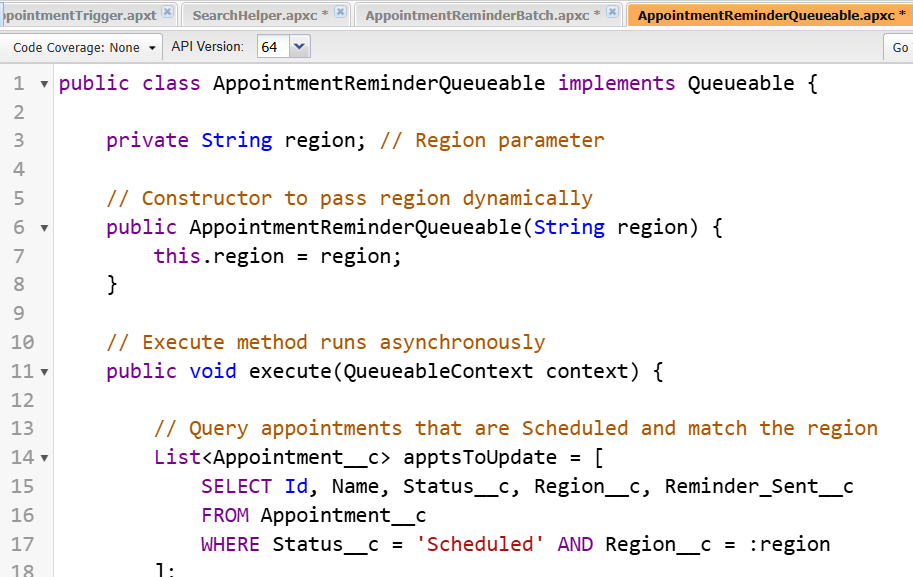
update apptsToUpdate;

}

System.debug('Queueable completed. Total records updated: ' + apptsToUpdate.size());

}

}



***8. Scheduled Apex***

* **Purpose :** Run logic at a scheduled time.

In developer console---> File----->New--->Apex Classs.

***Code :***

global class ScheduledPatientJob implements Schedulable {

// This method executes at the scheduled time

global void execute(SchedulableContext sc) {

// Example logic: fetch patients without Care Coordinator and assign one

List<Patient\_\_c> patients = [SELECT Id, Region\_\_c FROM Patient\_\_c WHERE Care\_Coordinator\_\_c = NULL];

User doctor = [SELECT Id FROM User WHERE Name='Dr. Ravi Kumar' LIMIT 1];

for(Patient\_\_c p : patients) {

if(p.Region\_\_c == 'East') {

p.Care\_Coordinator\_\_c = doctor.Id;

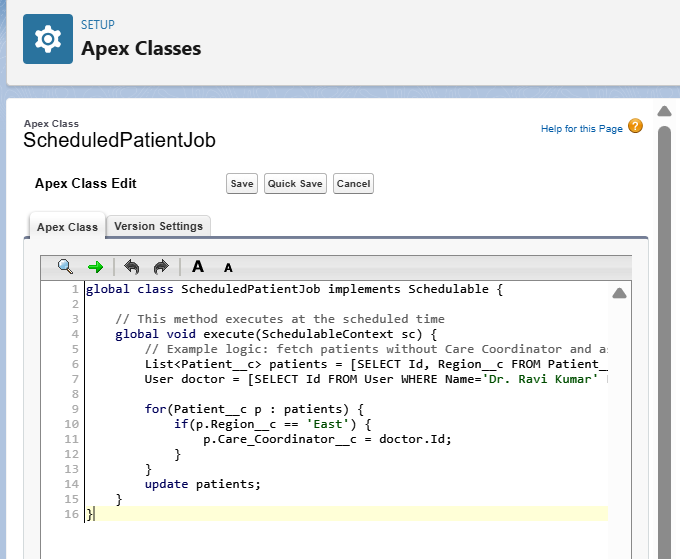
}

}

update patients;

}

}



* ***Test the Scheduled Apex***

**Name :** ScheduledPatientJobTest

**Code :**

@isTest

public class ScheduledPatientJobTest {

@isTest static void testScheduledJob() {

// Create test data

User doctor = [SELECT Id FROM User WHERE Name='Dr. Ravi Kumar' LIMIT 1];

Patient\_\_c p = new Patient\_\_c(Name='Test Patient', Region\_\_c='East');

insert p;

Test.startTest();

String cronExp = '0 0 2 \* \* ?';

System.schedule('Test Job', cronExp, new ScheduledPatientJob());

Test.stopTest();

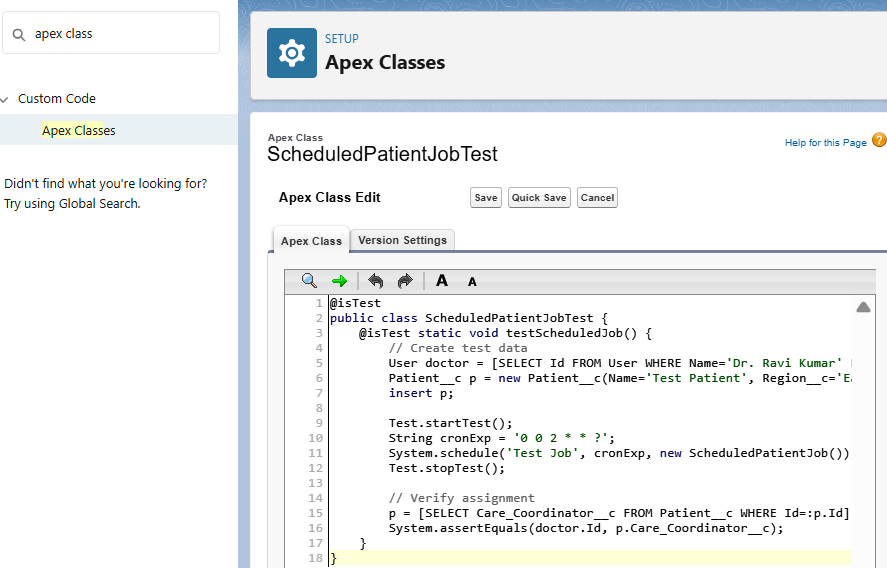
// Verify assignment

p = [SELECT Care\_Coordinator\_\_c FROM Patient\_\_c WHERE Id=:p.Id];

System.assertEquals(doctor.Id, p.Care\_Coordinator\_\_c);

}

}

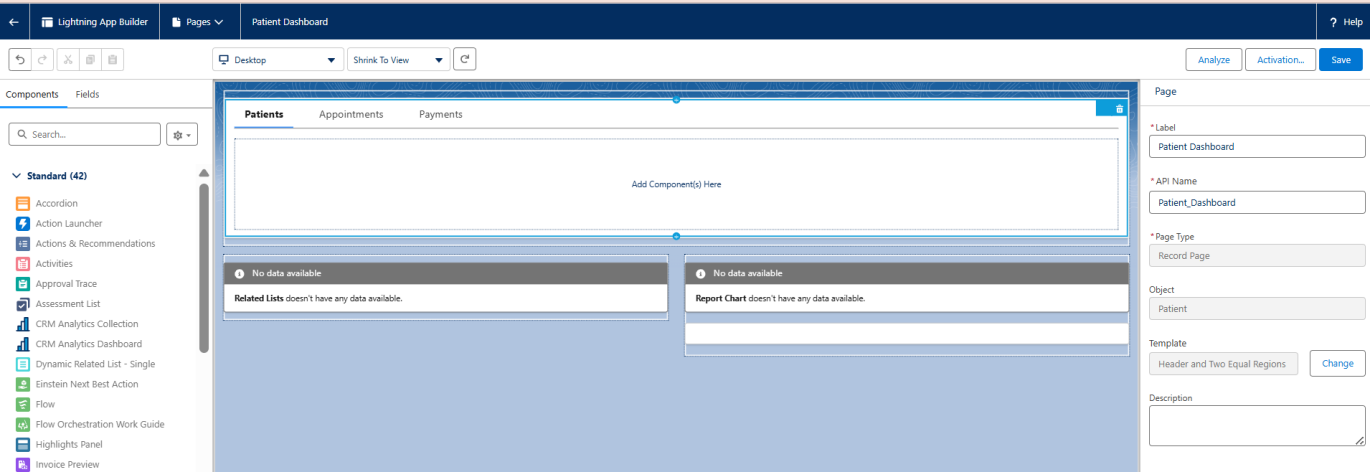


* ***Phase 6: User Interface Development***

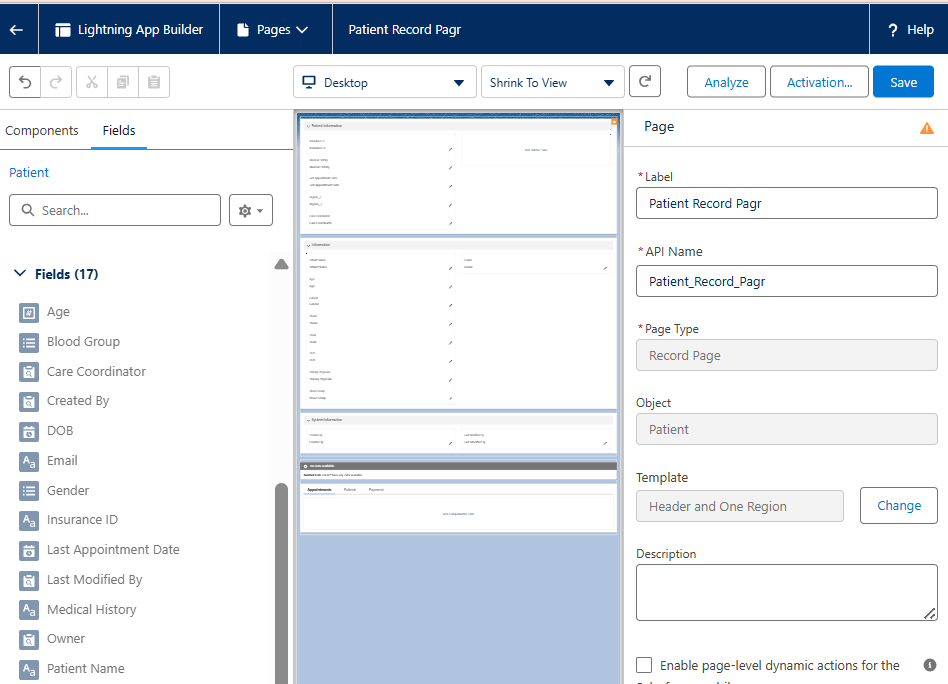
1. ***Lightning App Builder***
2. Lightning App Builder — Create the App Open App Manager.
3. Go to Setup in Salesforce ---> In the Quick Find box, type App Manager ---> Click App Manager.
4. Click New Lightning App ---> Name the App as HealthCare App



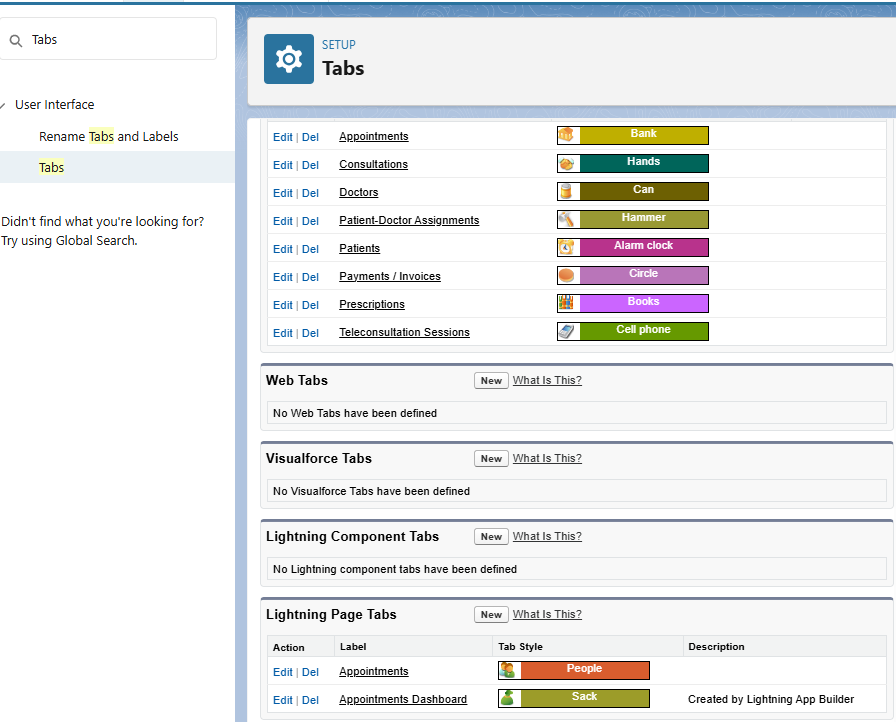
1. Open Lightning App Builder.
2. Go to Setup → Lightning App Builder --> Click New.



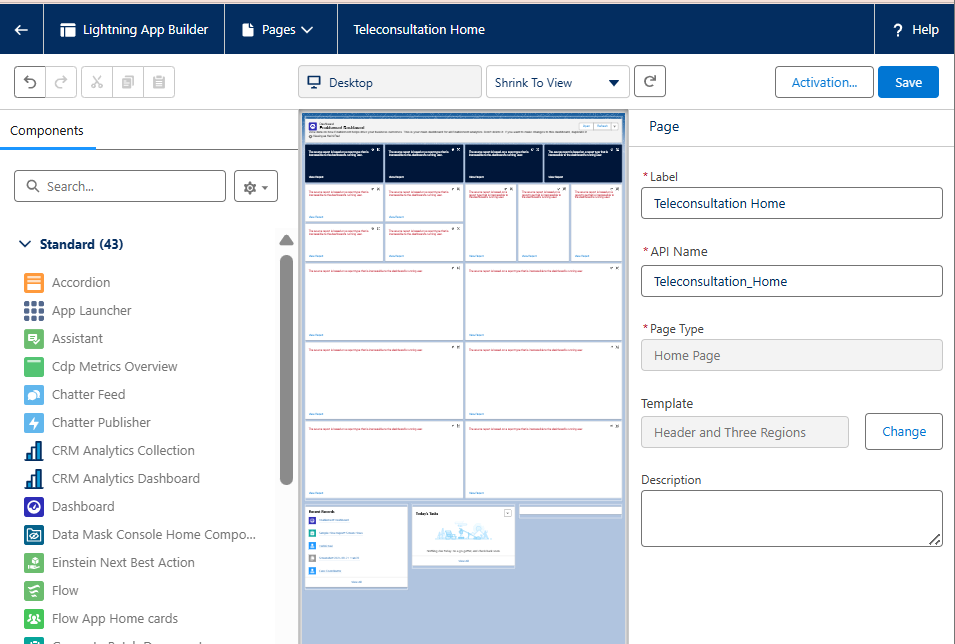
1. ***Home Page Layouts***
2. Go to **Setup → Object Manager → [Object] → Lightning Record Pages → New.**
3. Choose a **Template** (Header + Tabs or One Column).
4. Drag **Standard and Custom Components** onto the page.
5. Add **Related Lists, Tabs, Quick Actions**.
6. Save & activate for **Org Default / App / Profile**.



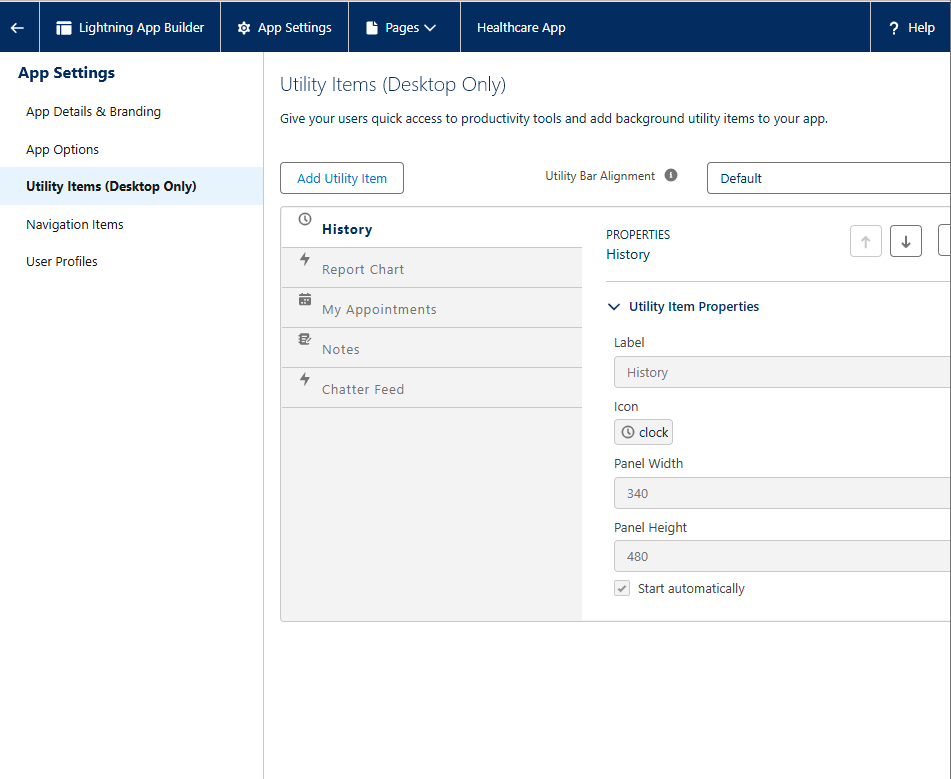
1. ***Tabs***
2. Go to **Setup → Tabs → Lightning Page Tabs**.
3. Click **New** and choose the **Object or Lightning Page**.
4. Name the tab (e.g., Appointments) and assign an **icon/color**.
5. Add tabs to **Navigation Menu** in Lightning App Builder.



1. ***Home Page Layouts***
2. Go to **Setup → Lightning App Builder → Home Page → New**.
3. Choose a **Template** (Standard Home Page or Console Home).
4. Add **Components**: Dashboard charts, Recent Records, Tasks, News.
5. Assign **Activation**: Org Default, App Default, Profile Default.



1. ***Utility Bar***
2. Go to **Setup**
3. In **Quick Find**, type **App Manager** → Click **App Manager**.
4. Find your **Healthcare Teleconsultation Lightning App**.
5. Click the dropdown → **Edit**.
6. Choose Utility Item Type ---> Report chart, My Appointments, Notes, Chatter Feed.



1. ***Custom LWC***
2. Create Apex Controller for creating LWC
3. **Code :**

public with sharing class AppointmentController {

@AuraEnabled(cacheable=true)

public static List<Appointment\_\_c> getTodaysAppointments() {

Date today = Date.today();

Id currentUserId = UserInfo.getUserId();

return [SELECT Id, Name, Appointment\_Date\_\_c, Patient\_\_r.Name

FROM Appointment\_\_c

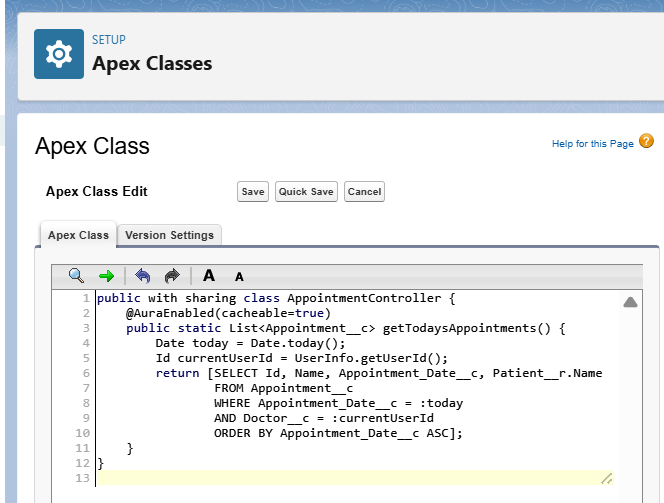
WHERE Appointment\_Date\_\_c = :today

AND Doctor\_\_c = :currentUserId

ORDER BY Appointment\_Date\_\_c ASC];

}

}



1. ***Create LWC***
2. Open a terminal inside your Salesforce DX project folder and run:

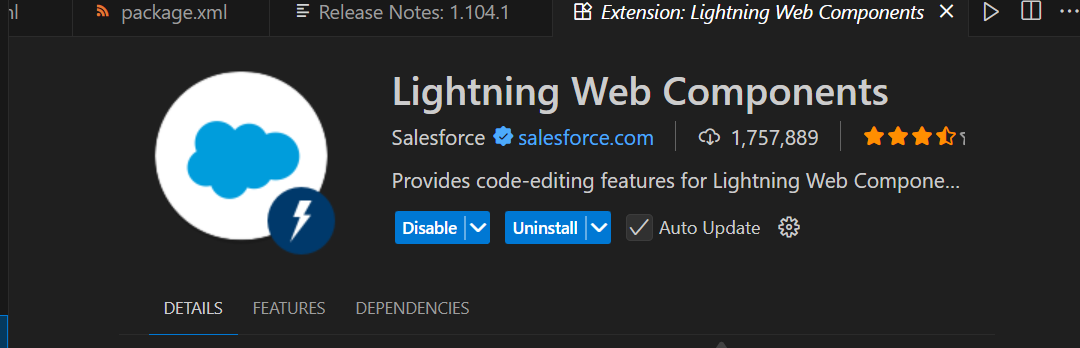
**Code :**

sfdx force:lightning:component:create \

--type lwc \

--componentname patientCard \

--outputdir force-app/main/default/lwc



1. **This generates a folder:**

force-app/main/default/lwc/patientCard/

├── patientCard.html

├── patientCard.js

├── patientCard.js-meta.xml

└── patientCard.css (optional, create if needed)

***todayAppointments.html***

<template>

<lightning-card title="Today’s Appointments">

<template if:true={appointments.data}>

<ul>

<template for:each={appointments.data} for:item="appt">

<li key={appt.Id}>

{appt.Appointment\_Date\_\_c} - {appt.Patient\_\_r.Name}

</li>

</template>

</ul>

</template>

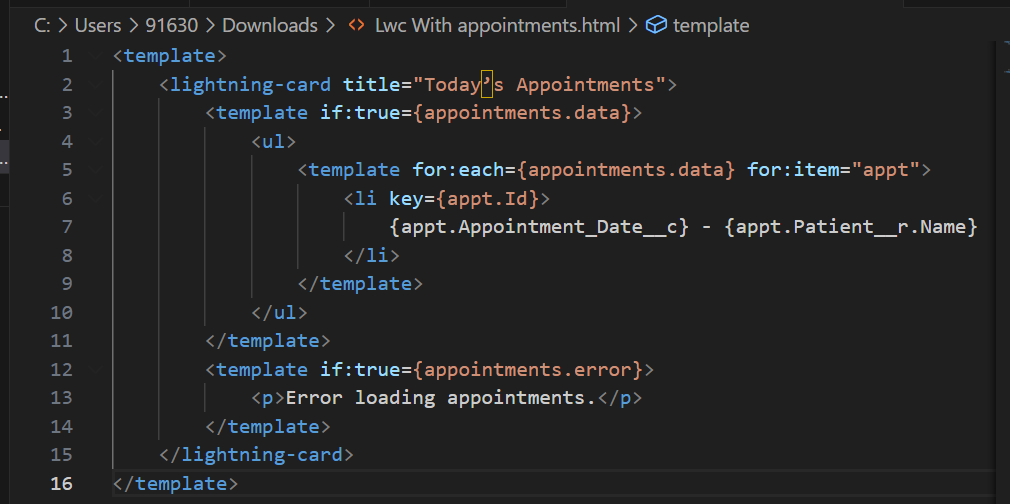
<template if:true={appointments.error}>

<p>Error loading appointments.</p>

</template>

</lightning-card>

</template>





1. ***Apex with LWC***

Create an **Apex class** annotated with @AuraEnabled(cacheable=true) if it returns data.

**Name :** PatientController

**Code :**

public with sharing class PatientController {

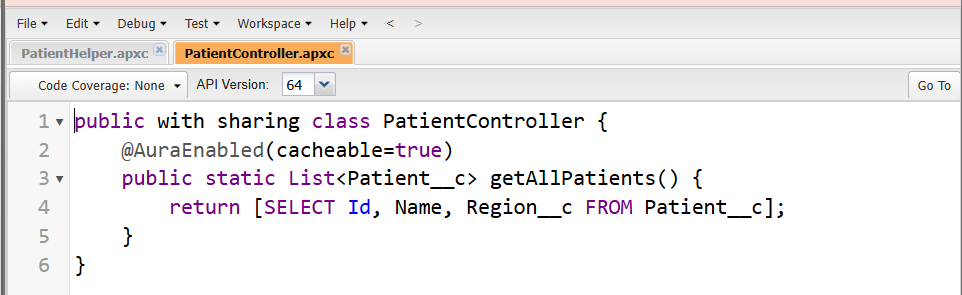
@AuraEnabled(cacheable=true)

public static List<Patient\_\_c> getAllPatients() {

return [SELECT Id, Name, Region\_\_c FROM Patient\_\_c];

}

}



1. ***Events in LWC***

Sibling or Cross-Component Communication (Lightning Message Service – LMS)

1. Create a Message Channel:
2. Setup → Lightning Message Channels → New → PatientSelectionChannel.
3. Add a single field: e.g., selectedPatientId

* **Publish Message from Component A:**

import { LightningElement } from 'lwc';

import { publish, MessageContext } from 'lightning/messageService';

import PATIENT\_CHANNEL from '@salesforce/messageChannel/PatientSelectionChannel\_\_c';

export default class ComponentA extends LightningElement {

@wire(MessageContext)

messageContext;

handleSelect(patientId) {

const message = { selectedPatientId: patientId };

publish(this.messageContext, PATIENT\_CHANNEL, message);

}

}

* **Subscribe in Component B:**

import { LightningElement, wire } from 'lwc';

import { subscribe, MessageContext } from 'lightning/messageService';

import PATIENT\_CHANNEL from '@salesforce/messageChannel/PatientSelectionChannel\_\_c';

export default class ComponentB extends LightningElement {

@wire(MessageContext)

messageContext;

connectedCallback() {

this.subscription = subscribe(this.messageContext, PATIENT\_CHANNEL, (message) => {

this.selectedPatientId = message.selectedPatientId;

// Handle selected patient data

});

}

}

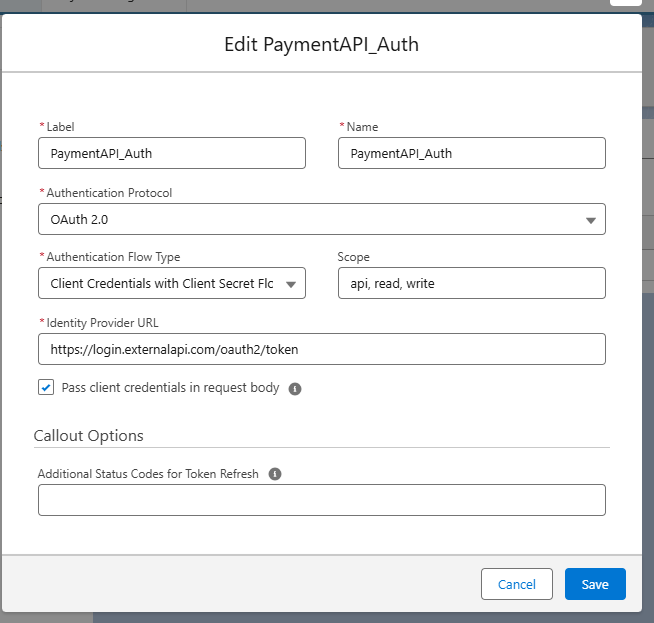
* ***Phase 7: Integration & External Access***

1. ***Named Credentials***

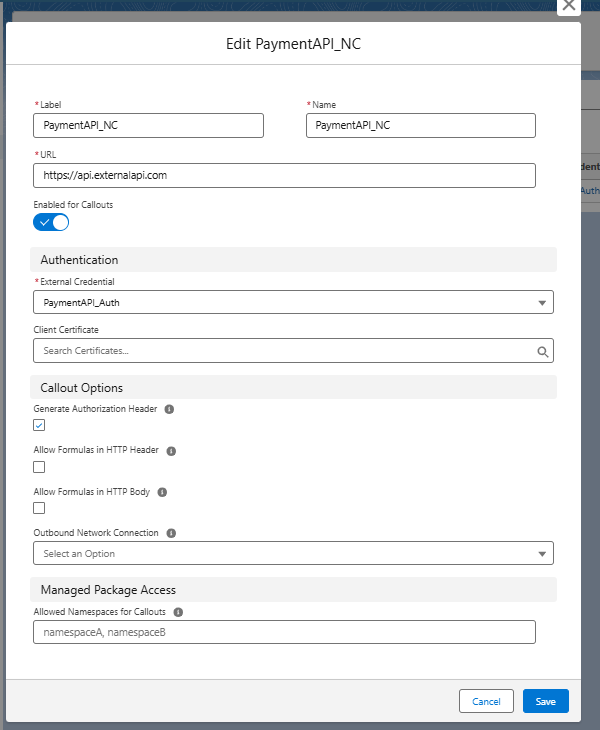
**Purpose :** Store an endpoint URL and authentication in one place for secure callouts.

1. Setup → Named Credentials → New Named Credential
2. Enter: Label & Name
3. URL of the external system.
4. Identity Type: Named Principal (one login for all users) or Per User.
5. Authentication Protocol: Password Authentication or OAuth 2.0.
6. Save and use in Apex callouts as callout—no hard-coded URLs.

* ***External Credentials***



* ***Named Credentials***



* ***Using Apex Class (Recommended for reuse)***

1. Go to Setup → Apex Classes → New
2. Enter a class name, e.g., PaymentAPIService
3. Paste the GET/POST code inside a method. Example:
4. Code :

public with sharing class PaymentAPIServiceAlt {

// GET Payment Details - returns response body

public static String getPayments() {

Http http = new Http();

HttpRequest req = new HttpRequest();

req.setEndpoint('callout:PaymentAPI\_NC/v1/payments');

req.setMethod('GET');

try {

HttpResponse res = http.send(req);

System.debug('GET Status: ' + res.getStatusCode());

System.debug('GET Body: ' + res.getBody());

return res.getBody();

} catch (CalloutException e) {

System.debug('GET Callout Error: ' + e.getMessage());

return null;

}

}

// POST a Payment - returns response body

public static String createPayment(Decimal amount, String currencyCode) {

Http http = new Http();

HttpRequest req = new HttpRequest();

req.setEndpoint('callout:PaymentAPI\_NC/v1/payments');

req.setMethod('POST');

req.setHeader('Content-Type', 'application/json');

// JSON body safely formatted

String requestBody = '{"amount":' + String.valueOf(amount) + ',"currency":"' + String.escapeSingleQuotes(currencyCode) + '"}';

req.setBody(requestBody);

try {

HttpResponse res = http.send(req);

System.debug('POST Status: ' + res.getStatusCode());

System.debug('POST Body: ' + res.getBody());

return res.getBody();

} catch (CalloutException e) {

System.debug('POST Callout Error: ' + e.getMessage());

return null;

}

}

// Optional: parse JSON response into a Map

public static Map<String, Object> parseJsonResponse(String jsonResponse) {

if(String.isBlank(jsonResponse)) return null;

try {

return (Map<String, Object>) JSON.deserializeUntyped(jsonResponse);

} catch (Exception e) {

System.debug('JSON Parse Error: ' + e.getMessage());

return null;

}

}

}

* Open **Developer Console → Execute Anonymous**, then run:

***Code :***

// GET payments

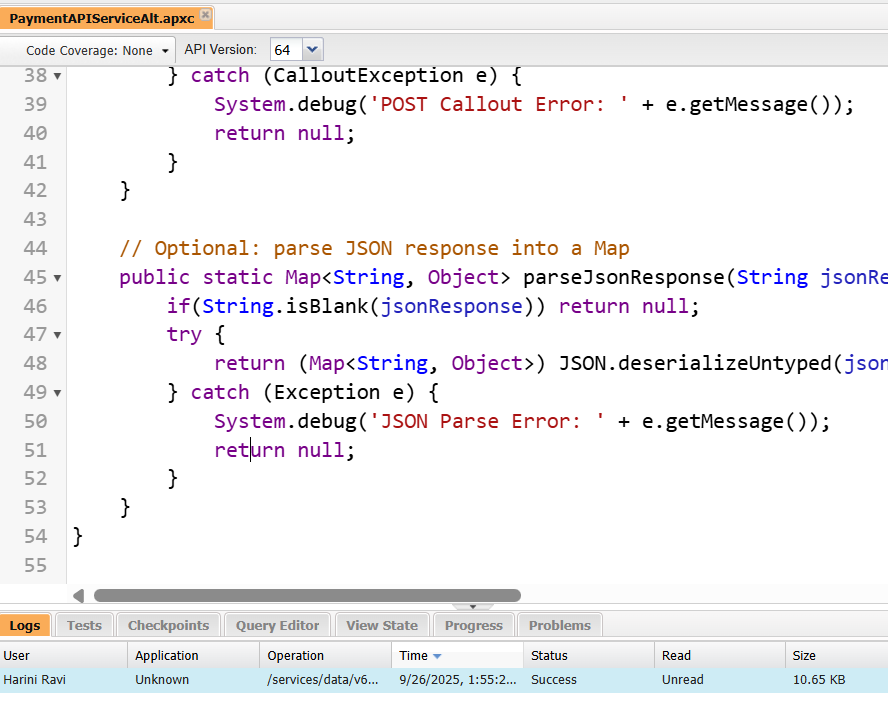
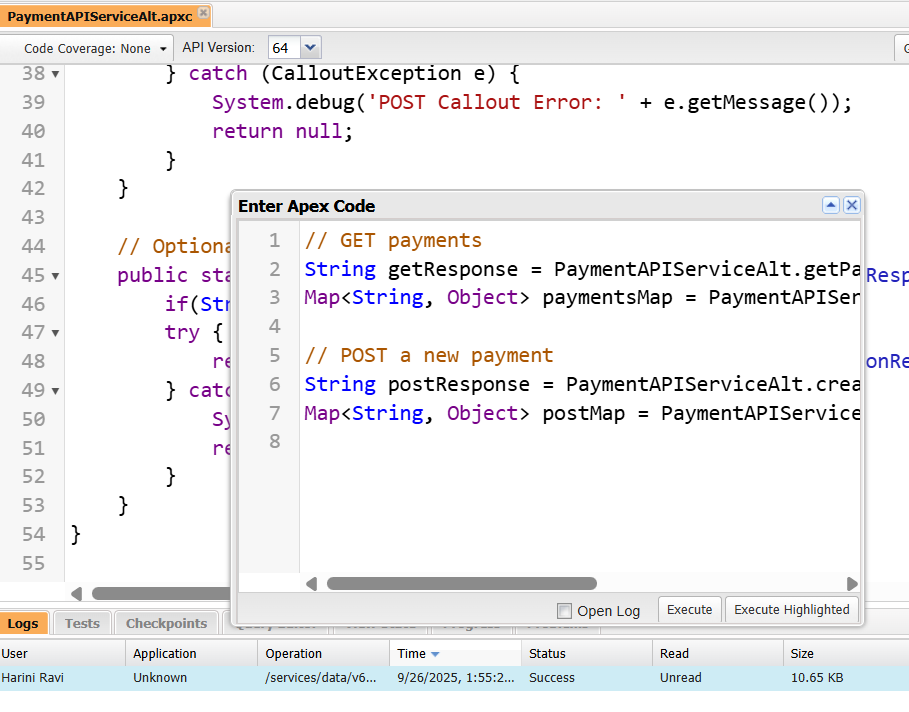
String getResponse = PaymentAPIServiceAlt.getPayments();

Map<String, Object> paymentsMap = PaymentAPIServiceAlt.parseJsonResponse(getResponse);

// POST a new payment

String postResponse = PaymentAPIServiceAlt.createPayment(200.75, 'USD');

Map<String, Object> postMap = PaymentAPIServiceAlt.parseJsonResponse(postResponse);



1. ***External Services***

**Purpose :** Declaratively connect Salesforce to REST APIs using Flows.

Steps:

1. Prepare a Swagger/OpenAPI spec of the external service.
2. Setup → External Services → New External Service
3. Fill in:

Name

Named Credential (from Step 1)

Swagger URL or file

1. Salesforce generates Apex actions for use in Flow Builder.
2. Usage: Drag actions into a Flow to call the external service without Apex code.
3. ***Web Services (REST/SOAP)***
4. Open Developer Console and Create Apex Class
5. Setup → Developer Console
6. File → New → Apex Class
7. Name it PaymentStatusAPI (or any valid name).

**Code :**

@RestResource(urlMapping='/PaymentStatus/\*')

global with sharing class PaymentStatusAPI {

@HttpGet

global static Payment\_\_c getPayment() {

RestRequest req = RestContext.request;

String paymentId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

Payment\_\_c p = [SELECT Id, Name, Status\_\_c, Amount\_\_c, Appointment\_\_r.Name

FROM Payment\_\_c WHERE Id = :paymentId LIMIT 1];

return p;

}

@HttpPost

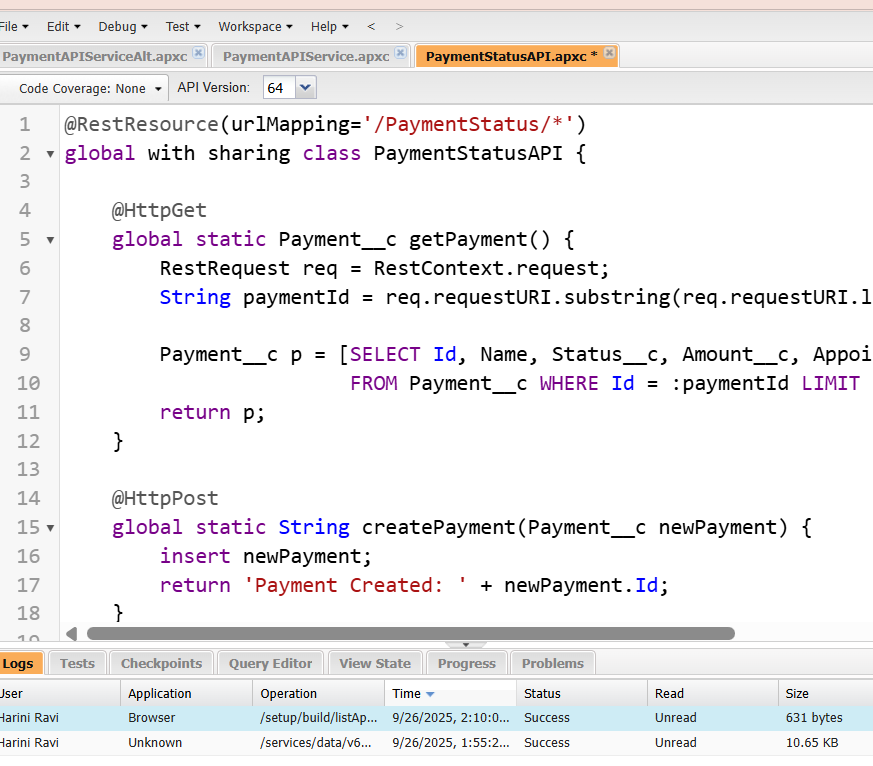
global static String createPayment(Payment\_\_c newPayment) {

insert newPayment;

return 'Payment Created: ' + newPayment.Id;

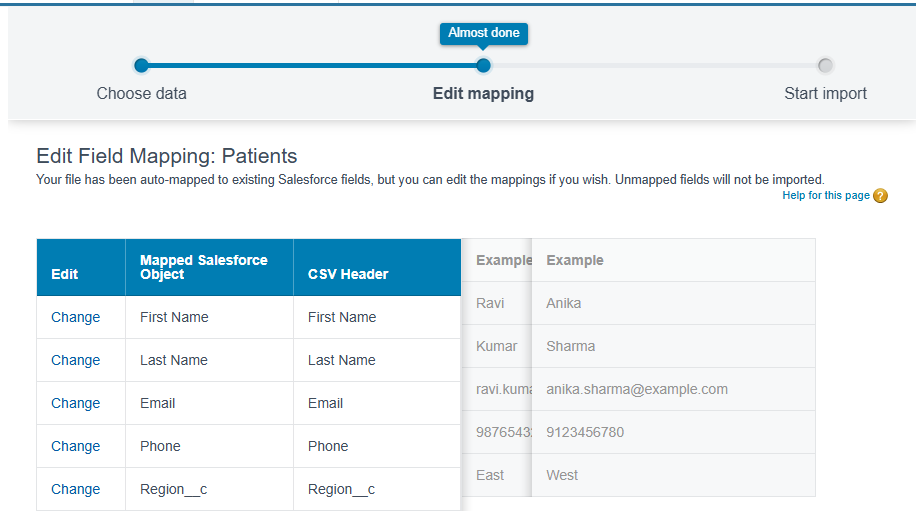
}

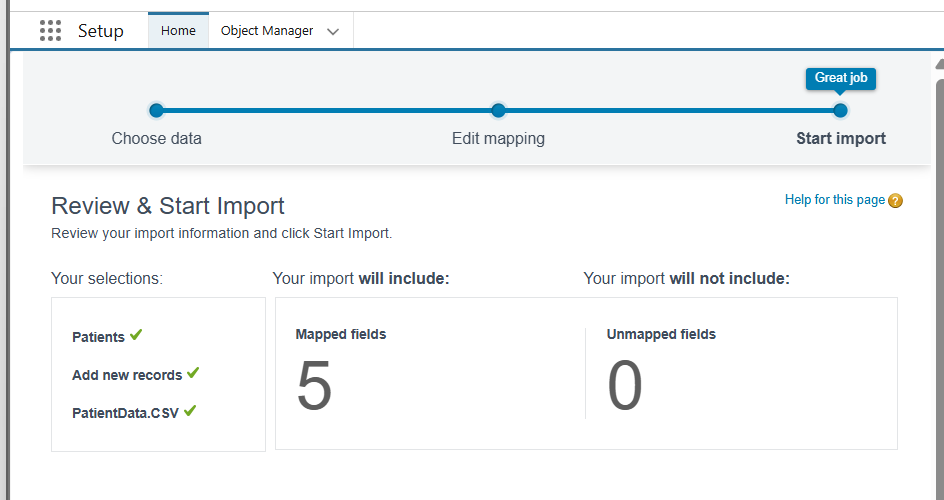
}



* ***Phase 8: Data Management & Deployment***

1. ***Data Import Wizard***
2. Setup → **Data Import Wizard** → **Launch Wizard**.
3. Choose the object (e.g., Patient\_\_c, Appointment\_\_c).
4. Upload a CSV file and map columns to Salesforce fields.
5. Click **Start Import** and monitor progress in the Bulk Data Load Jobs page.

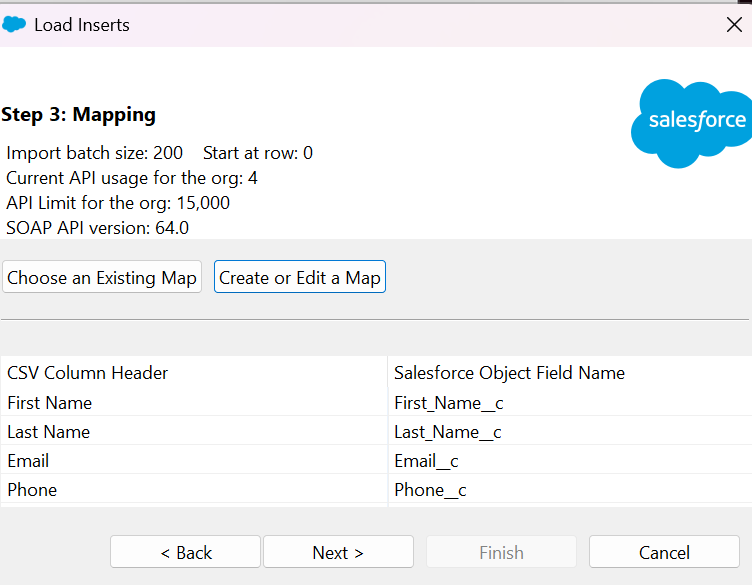


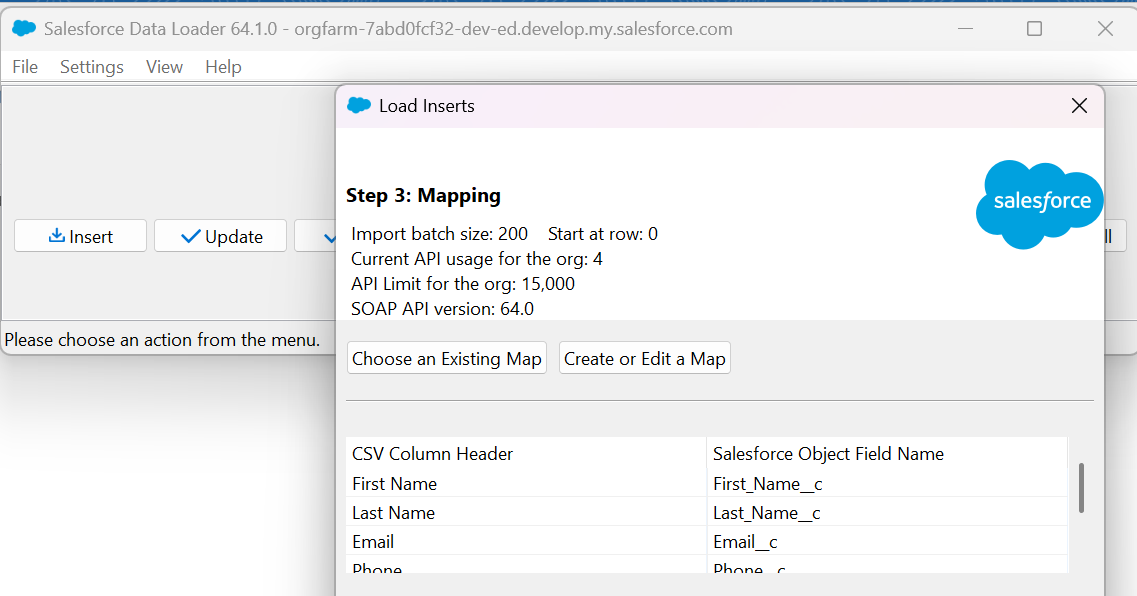


1. ***Data Loader***

**Purpose :** Desktop client for large or complex data operations (insert, update, upsert, delete, export).

1. Install Salesforce Data Loader.
2. Log in with your org credentials and security token.
3. Choose operation (Insert, Update, Upsert, Delete, Export).
4. Select object and CSV file, map fields, and run.

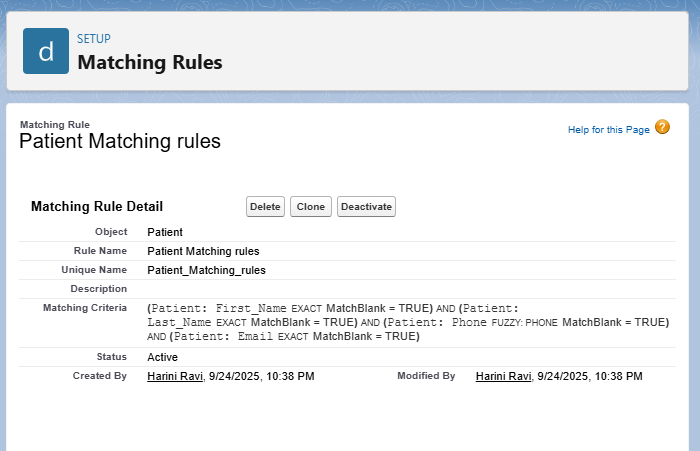


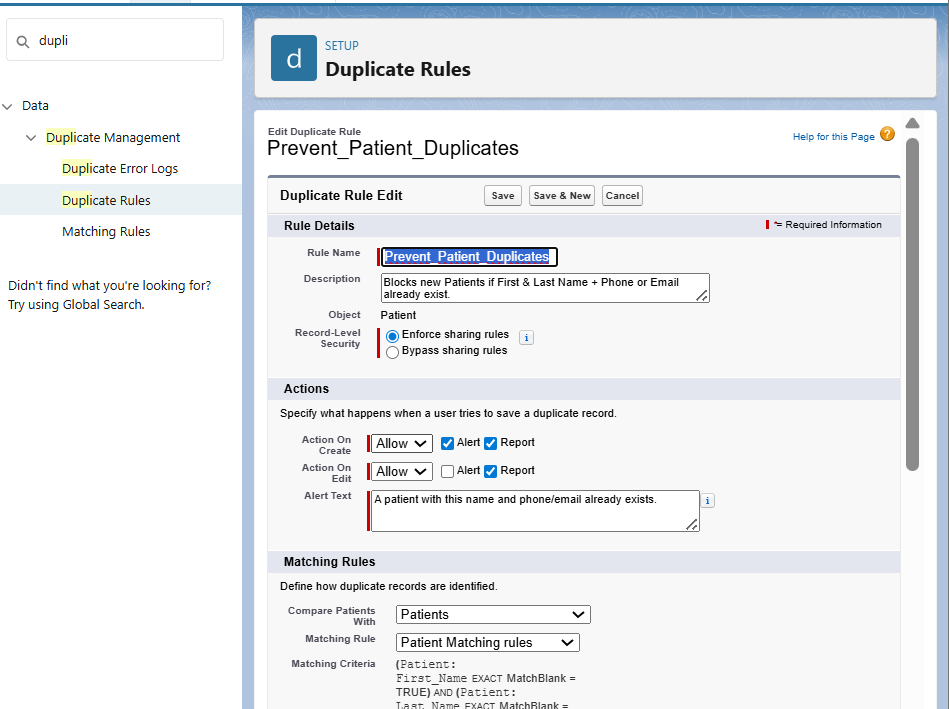


1. ***Duplicate Rules***

**Purpose :** Prevent or warn users about duplicate records (important for Patients or Doctors).

1. Setup → Duplicate Rules → New Rule.
2. Choose an object (e.g., Patient\_\_c).
3. Define matching criteria (name + phone or email).
4. Choose Block or Allow & Report.
5. Activate the rule.

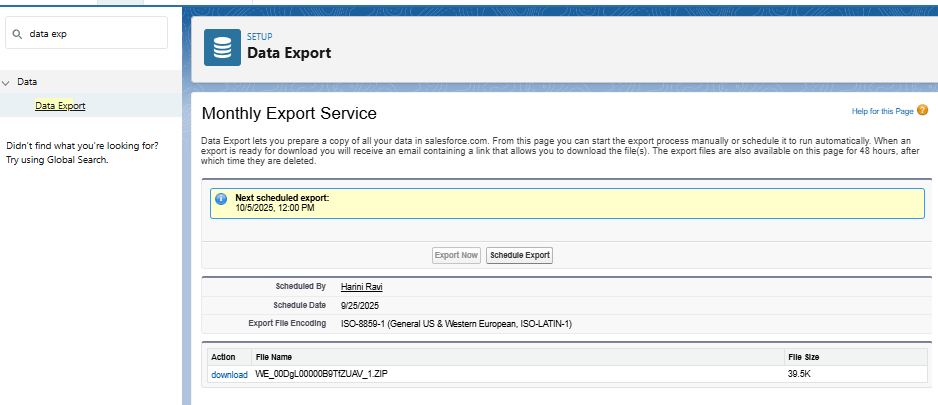


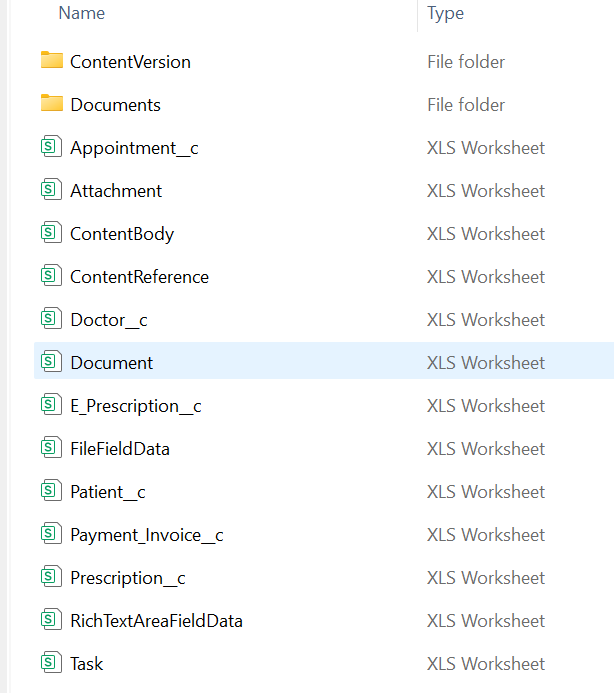


1. ***Data Export & Backup***

**Purpose :** Scheduled or manual backups of all org data.

1. Setup → Data Export
2. Choose Manual or Scheduled Export
3. Select critical objects (Patient\_\_c, Doctor\_\_c, Appointment\_\_c, Payments\_\_c, Tasks, Events)
4. Include optional attachments/files if needed
5. Start Export or Save Schedule
6. Wait for notification
7. Download the .zip file
8. Store securely and verify contents





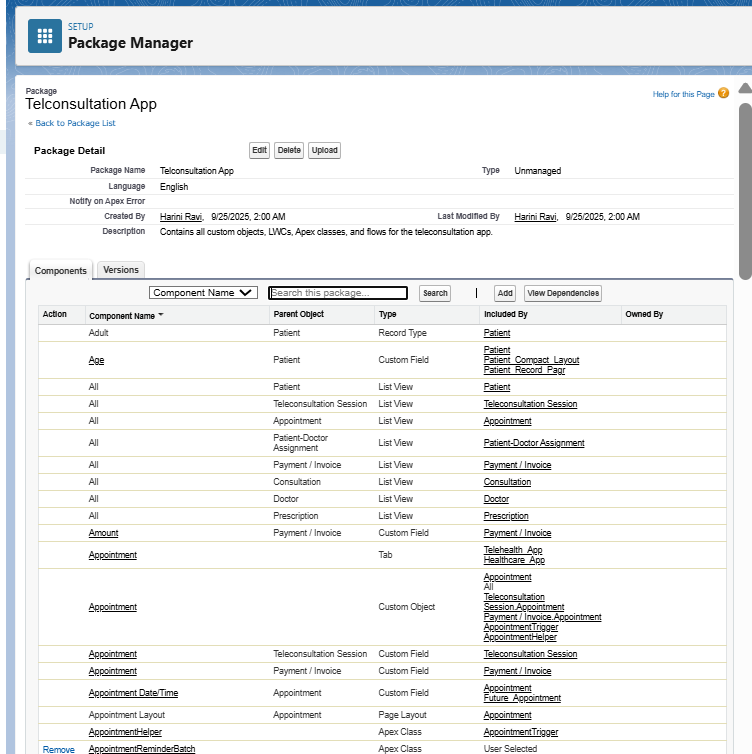
1. ***Change Sets***

**Purpose :** Deploy metadata (not data) between related Salesforce orgs.

1. Setup → Outbound Change Sets → New.
2. Add components (Objects, Fields, Apex Classes, LWCs, Profiles, etc.).
3. Upload to the target org.
4. In Target Org (Production): Setup → Inbound Change Sets → Deploy.
5. ***Unmanaged vs Managed Packages***

| **Type** | **Use Case** | **Features** |
| --- | --- | --- |
| Unmanaged | Open-source / template sharing | Editable code & metadata in subscriber org |
| Managed | Commercial apps / AppExchange | Versioning, upgrades, IP protection |

1. Go to **Setup → Quick Find → Packages → Packages → New**.
2. Enter **Package Name** and **Description**.
3. Choose the package type:
4. **Unmanaged** → for templates / sandbox sharing.
5. **Managed** → for commercial apps / production deployment.
6. Click **Save**.



1. ***ANT Migration Tool***

**Purpose :** Script-based deployments using the Metadata API (for CI/CD).

1. Install Java JDK and Apache Ant.
2. Download the Salesforce ANT Migration Tool from Setup.
3. Create build.properties (username, password+token) and package.xml.
4. Commands:

ant retrieve – pull metadata.

ant deploy – push metadata.

1. Confirm success in Deployment Status of target org.

* **Create package.xml**

**Code :**

<?xml version="1.0" encoding="UTF-8"?>

<Package xmlns="http://soap.sforce.com/2006/04/metadata">

<types>

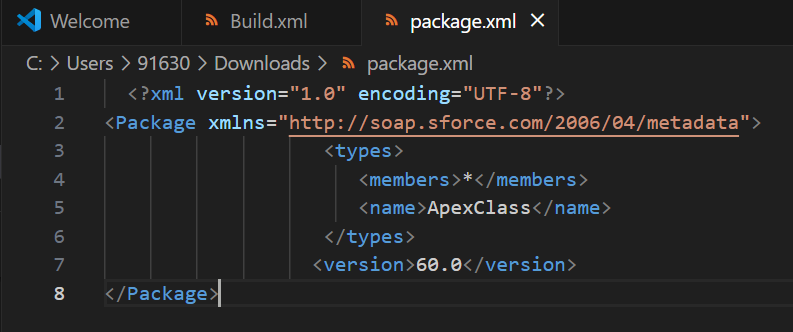
<members>\*</members>

<name>ApexClass</name>

</types>

<version>60.0</version>

</Package>



* ***Set Up build.xml***

**Code :**

<project name="SalesforceDeploy" default="deploy" basedir=".">

<property file="build.properties"/>

<taskdef resource="com/salesforce/antlib.xml" uri="antlib:com.salesforce">

<classpath>

<pathelement location="ant-salesforce.jar"/>

</classpath>

</taskdef>

<target name="retrieve">

<sf:retrieve unpackaged="package.xml" retrieveTarget="src"/>

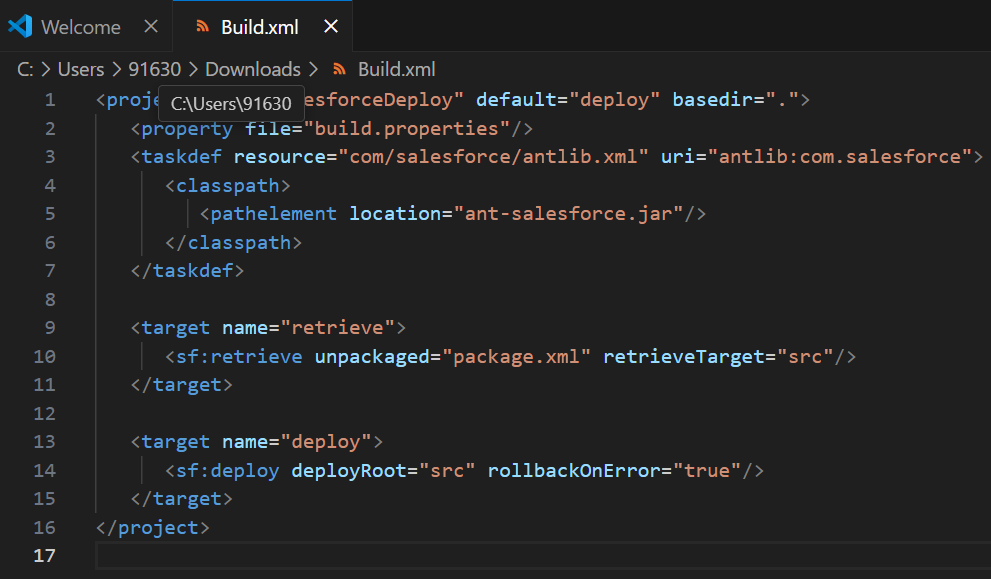
</target>

<target name="deploy">

<sf:deploy deployRoot="src" rollbackOnError="true"/>

</target>

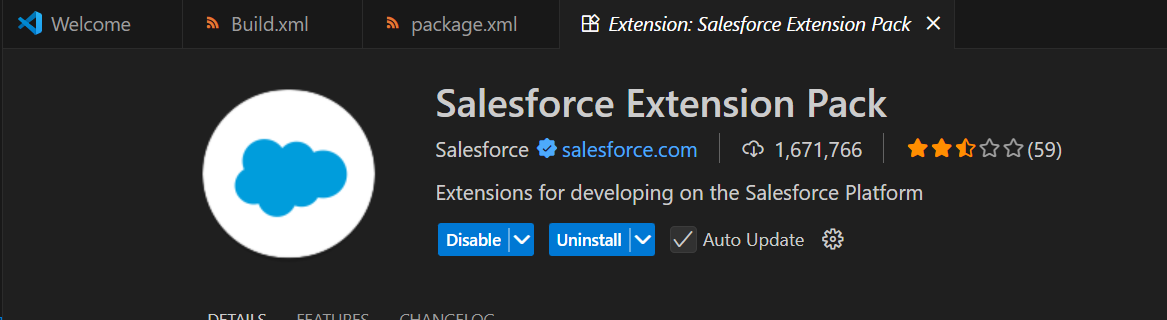
</project>

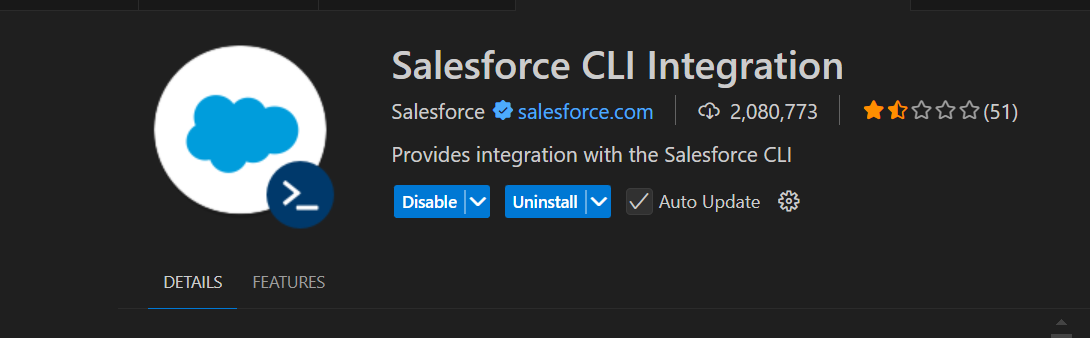


1. ***VS Code & SFDX (Salesforce CLI)***

**Purpose :** Modern development and CI/CD with scratch orgs.

1. Install Visual Studio Code + Salesforce Extensions Pack.
2. Install Salesforce CLI (SFDX).





* ***Phase 9: Reporting, Dashboards & Security Review***

1. ***Reports (Tabular, Summary, Matrix, Joined)***

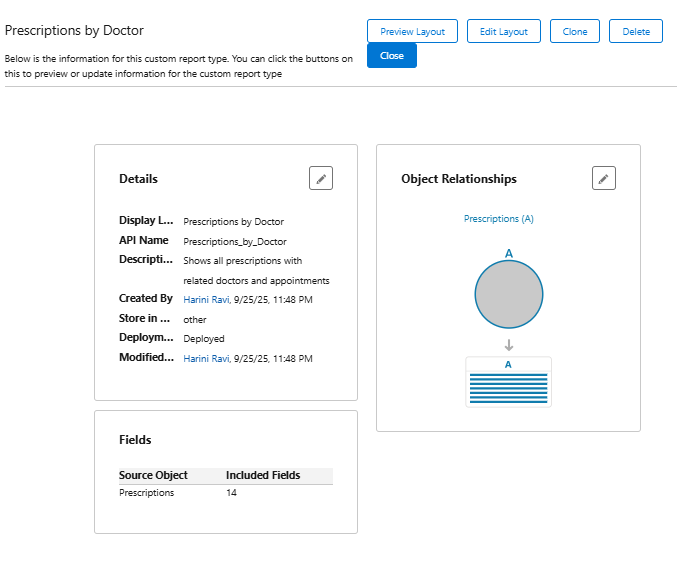
1. Navigate: App Launcher → Reports → New Report Select object or report type (e.g., Patients, Appointments, Prescriptions, Payments).
2. Choose report format:

Tabular: Simple list

Summary: Grouped rows, subtotals

Matrix: Rows & columns grouped, ideal for comparisons

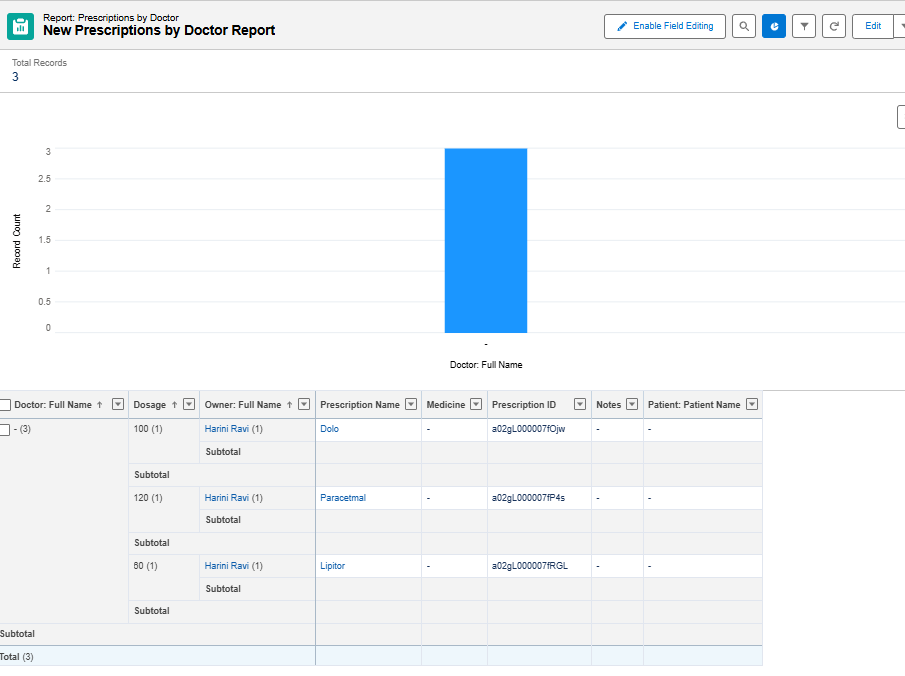
1. Joined: Combine multiple reports into one
2. Apply filters.
3. Group data by fields if needed.
4. Save report with descriptive name.



1. ***Report Types***

**Purpose :** Define which objects and relationships are available for reporting.

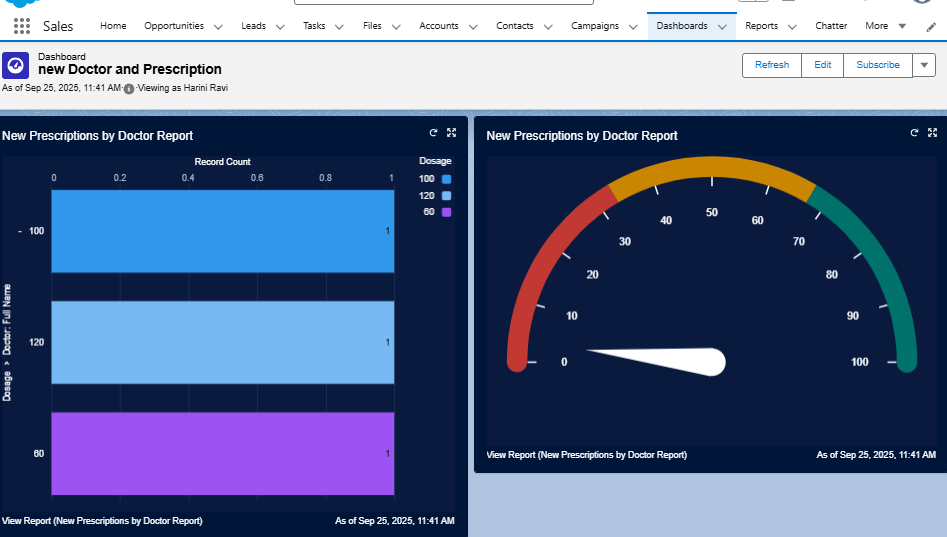
1. Setup → Report Types → New Custom Report Type
2. Select Primary Object (e.g., Prescription)
3. Add Related Objects
4. Define label, name, and description
5. Save.



1. ***Dashboards***

**Purpose :** Visualize reports in charts and graphs.

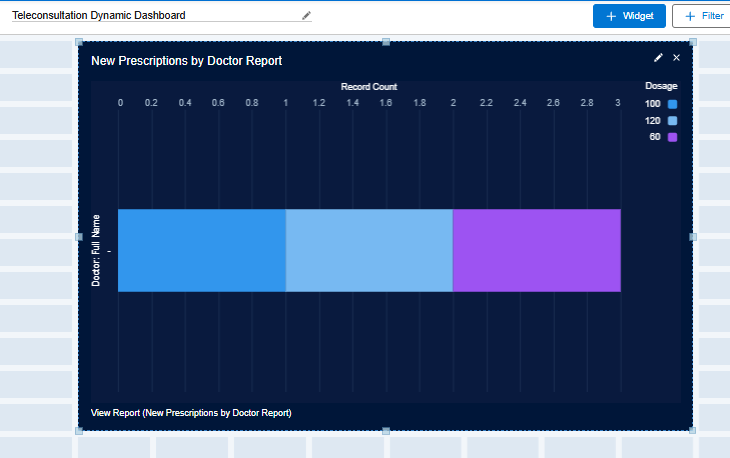
1. App Launcher → Dashboards → New Dashboard
2. Choose folder and name
3. Add components (charts, tables) based on reports
4. Select component type (Bar, Pie, Line, Metric)
5. Configure filters and data source (report)
6. Save and run dashboard



1. ***Dynamic Dashboards***

**Purpose :** Show data based on the logged-in user.

1. In the dashboard, click View Dashboard As → Run as Logged-in User
2. Save dashboard
3. Verify:
4. Doctor → sees only their prescriptions/appointments
5. Admin → sees all data
6. Support Staff → sees only records they can access.



1. ***Sharing Settings***

**Purpose :** Control record-level access.

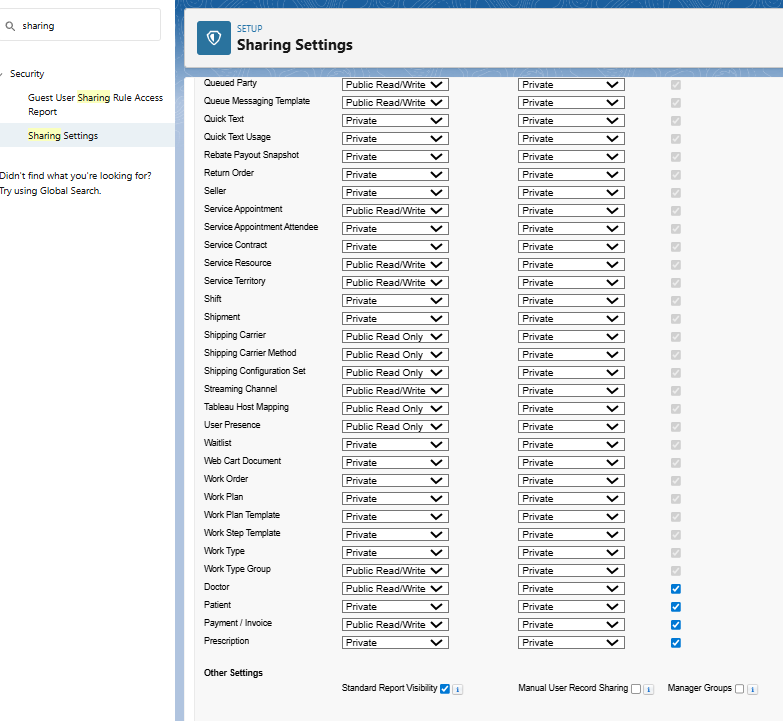
1. Setup → Sharing Settings
2. Set Org-Wide Defaults (OWD):

Patient → Private

Appointment → Controlled by Parent

Prescription → Private

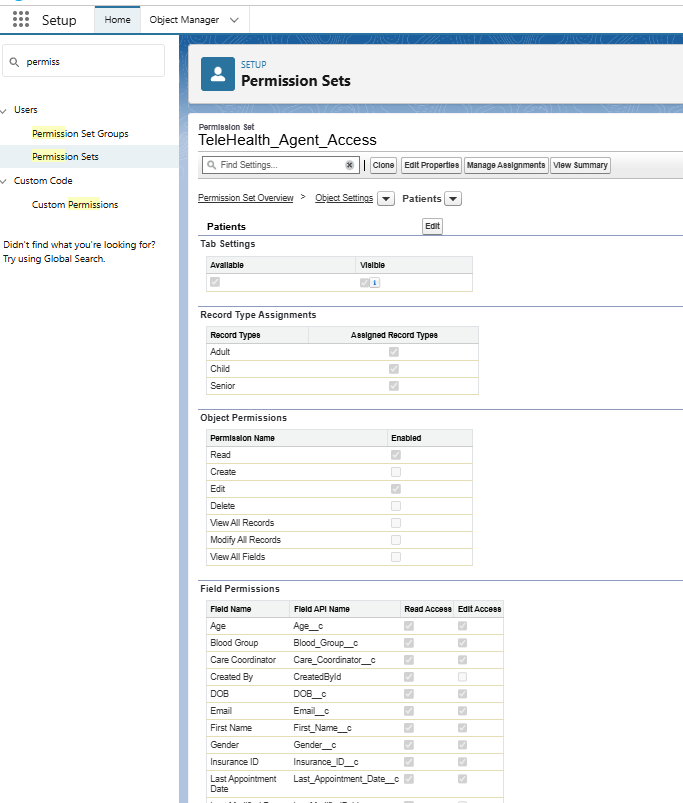
1. Create Sharing Rules to give access to roles/groups:
2. Patient → Doctors → Read/Write
3. Appointment → Support Staff → Read Only



1. ***Field Level Security (FLS)***

**Purpose :** Control field visibility per profile or permission set.

1. Setup → Object Manager → [Object] → Fields & Relationships → Field-Level Security
2. Select profiles to hide/edit fields
3. Use Permission Sets to grant extra access without changing profiles.

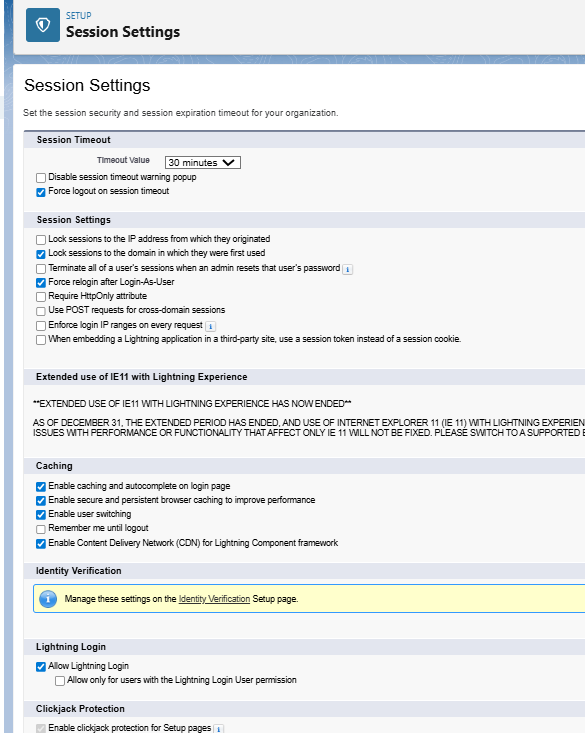


1. ***Session Settings***

**Purpose :** Control login behavior and security policies.

1. Setup → Session Settings
2. Configure:

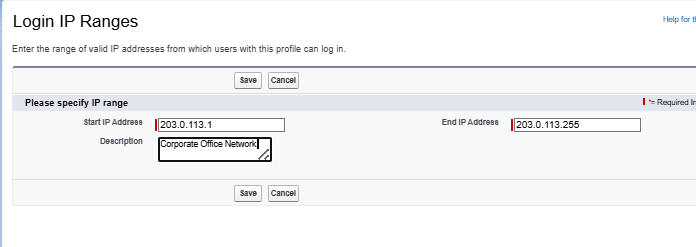
* Session timeout.
* Force logout on timeout
* Concurrent login restrictions
* High-risk session options if needed



1. ***Login IP Ranges***

**Purpose :** Restrict login access to trusted networks.

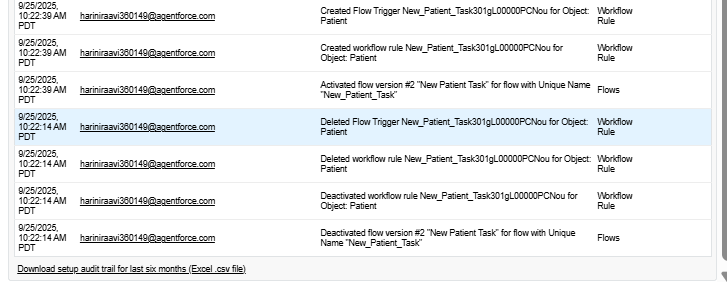
1. Setup → Profiles → [Profile] → Login IP Ranges
2. Click New → set Start IP and End IP
3. Save → only users from these IPs can log in



1. ***Audit Trail***

**Purpose :** Track administrative changes for compliance and troubleshooting.

1. Setup → View Setup Audit Trail
2. Review changes: objects, fields, profiles, permission sets
3. Download CSV to see up to 180 days of history
4. Use for compliance, troubleshooting, or documenting changes



1. I downloaded the file.
2. To see older history (up to 180 days):
3. Click Download to export a CSV file.
4. Open the CSV in Excel/Sheets to sort or filter by date, user, or action.

