SALESFORCE PROJECT IMPLEMENTATION PHASES

Problem Statement: Telemedicine Access for Rural Healthcare in Nabha

Description: Nabha and its surrounding rural areas face significant healthcare challenges. The local Civil Hospital operates at less than 50% staff capacity, with only 11 doctors for 23 sanctioned posts. Patients from 173 villages travel long distances, often missing work, only to find that specialists are unavailable or medicines are out of stock. Poor road conditions and sanitation further hinder access. Many residents lack timely medical care, leading to worsened health outcomes and increased financial strain.

Impact / Why this problem needs to be solved:

This problem directly affects the health and livelihood of thousands of rural residents, especially daily-wage earners and farmers. Lack of accessible healthcare leads to preventable complications, financial losses, and overall decline in community well-being. Addressing this issue would improve healthcare delivery, reduce unnecessary travel, and enhance quality of life for a large, underserved population.

Phase 1: Problem Understanding & Industry Analysis

1. Requirement Gathering

Conducted requirement gathering based on the problem statement and rural healthcare needs.

Functional Requirements:

- Video consultations between patients and doctors.
- Digital health records accessible both online and offline.
- Real-time medicine availability updates from local pharmacies.
- AI-powered symptom checker optimized for low bandwidth.
- Appointment booking and notifications via SMS/WhatsApp.

Non-Functional Requirements:

- Multilingual support (Punjabi, Hindi, English).

- Offline-first mobile app for areas with poor connectivity.
- Secure data storage with role-based access.
- Scalable design for other rural regions.

2. Stakeholder Analysis

Stakeholder	Role in Project	Benefits
Patients (Rural Villagers)	Access healthcare without traveling long distances.	Saves time, money, ensures timely treatment.
Doctors	Provide consultations remotely.	Reach more patients, efficient scheduling.
Pharmacy Staff	Manage medicine availability and inventory.	Real-time updates for patients and doctors.
Civil Hospital Staff	Support patient registration and manage referrals.	Reduce hospital crowding, track patient flow.
Punjab Health Dept.	Oversee rural healthcare delivery.	Data-driven insights, performance monitoring.
Daily Wage Workers	Patients who lose income due to travel for healthcare.	Can continue earning without frequent hospital visits.

3. Business Process Mapping

High-level flow of the telemedicine process:

1. Patient logs into the community portal/mobile app.
2. Patient books appointment with available doctor.
3. System sends confirmation + reminders (SMS/Email).
4. Doctor conducts video consultation.
5. Prescription generated and stored in patient's digital record.
6. Pharmacy updates stock & dispenses medicine.
7. Notifications sent to patient for medicine pickup.
8. Health Dept. accesses dashboards for analytics & monitoring.

FlowChart--

[Patient Login / Registration --> Book Appointment --> System Sends Confirmation + Reminder --> Doctor Conducts Video Consultation --> Prescription Generated &

Saved --> Pharmacy Updates Inventory --> Medicine Dispensed to Patient --> Notifications Sent to Patient --> Health Dept. Reviews Analytics Dashboard]

4. Industry-specific Use Case Analysis

Healthcare Gaps in Rural Areas:

- Lack of specialists and infrastructure.
- High travel cost for patients.
- Poor medicine availability tracking.

How Salesforce Solves This:

- **Health Cloud**: Manage patient data, history, and care plans.
- **Community Cloud**: Patients access portal in their language.

- **Service Cloud**: Appointment booking, reminders, case management.
- **Einstein Bots & AI**: Symptom checker for quick triage.
- **Integration**: Pharmacy systems, government health databases.

5. **AppExchange Exploration**

To avoid reinventing the wheel, I explored Salesforce **AppExchange** solutions relevant to healthcare:

Health Cloud Packages → Prebuilt modules for patient management.
Telemedicine Connectors → Zoom/Teams integration for video calls.
Pharmacy Management Apps → Medicine inventory tracking.
SMS/WhatsApp Notification Apps → Twilio, ValueText.

SALESFORCE PROJECT IMPLEMENTATION PHASES

Phase 2: Org Setup & Configuration

In this phase, we configure the Salesforce Org to align with the Telemedicine Access for Rural Healthcare in Nabha project. Proper Org setup ensures smooth functionality, data security, role-based access, and scalability for future needs.

1. Salesforce Editions

Salesforce has different editions: Essentials, Professional, Enterprise, Unlimited, and Developer.

- For this project, I am using "Salesforce Developer Edition".

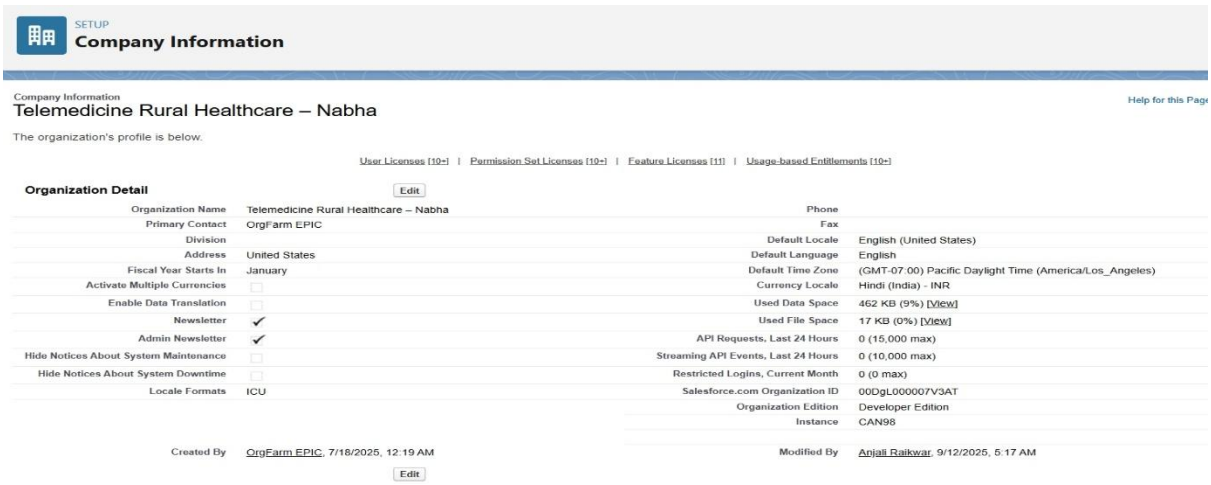
1. It gives us customization features.
2. It supports Apex, LWC, Integrations, and APIs.

- Developer Edition provides advanced customization, role hierarchy, profiles, API access, and integration support.
- It allows us to build custom apps for "Patients, Doctors, Pharmacy Staff, and Health Department Officers."

2. Company Profile Setup

Path: Setup → Company Information

- Company Name: Telemedicine Rural Healthcare.
- Default Locale: English (India)
- Default Time Zone: Asia/Kolkata (GMT +5:30)
- Default Currency: INR ₹ (Indian Rupees)
- Corporate Currency: INR ₹ (All records display INR as default)



This setup ensures consistency across all users in the Org.

3. Business Hours & Holidays

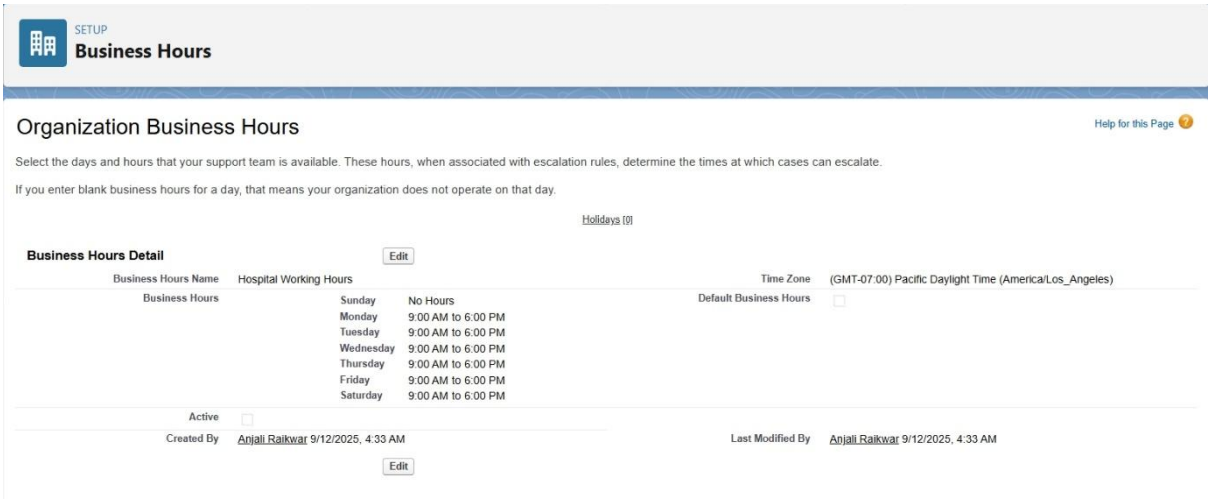
Path: Setup → Company Settings → Business Hours

- Default Business Hours: 9 AM – 6 PM (Monday – Saturday)
- Custom Business Hours:

1. Teleconsultation Service Window: 7 AM – 9 PM
2. Pharmacy Support: 9 AM – 8 PM

Holidays Configured:

1. 15 August – Independence Day
2. 26 January – Republic Day
3. 2 October – Gandhi Jayanti
4. 1 January – New Year

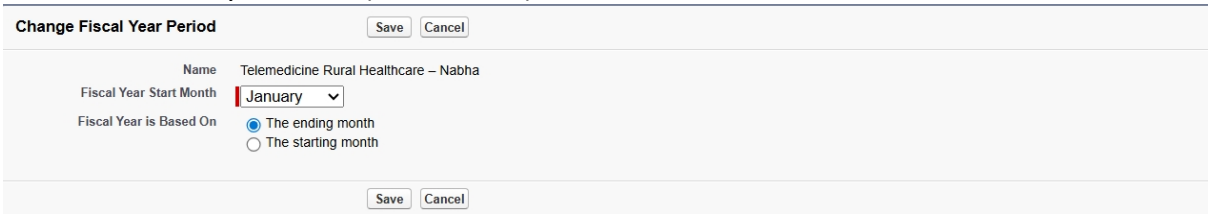


This ensures that SLA calculations, case escalations, and automated workflows respect working/non-working hours.

5. Fiscal Year Settings

Path: Setup → Company Profile → Fiscal Year

- Fiscal Year set to “January” (Indian standard financial year).
- Standard fiscal year used (not custom).



6. User Setup & Licenses

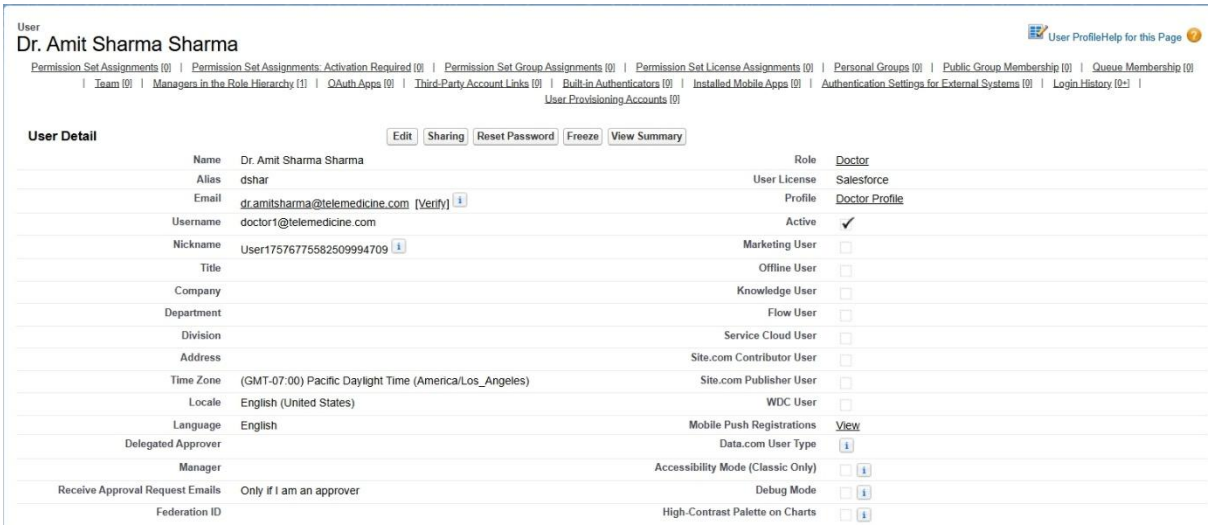
We created multiple users to represent project stakeholders.

Users Created:

1. Doctor User

- Profile: Standard User
- License: Salesforce
- Role: Doctor
- Access: Can create/view patient records, schedule teleconsultations.

- **Name:** Dr. Amit Sharma
- **Email:** dr.amitsharma@telemedicine.com
- **Username:** doctor1@telemedicine.com
- **Role:** Doctor
- **User License: Salesforce**
- **Profile:** Doctor Profile
- **Business Hours:** Hospital Working Hours



User
Dr. Amit Sharma Sharma

User ProfileHelp for this Page

Permission Set Assignments [0] | Permission Set Assignments: Activation Required [0] | Permission Set Group Assignments [0] | Permission Set License Assignments [0] | Personal Groups [0] | Public Group Membership [0] | Queue Membership [0] | Team [0] | Managers in the Role Hierarchy [1] | OAuth Apps [0] | Third-Party Account Links [0] | Built-in Authenticators [0] | Installed Mobile Apps [0] | Authentication Settings for External Systems [0] | Login History [0+] | User Provisioning Accounts [0]

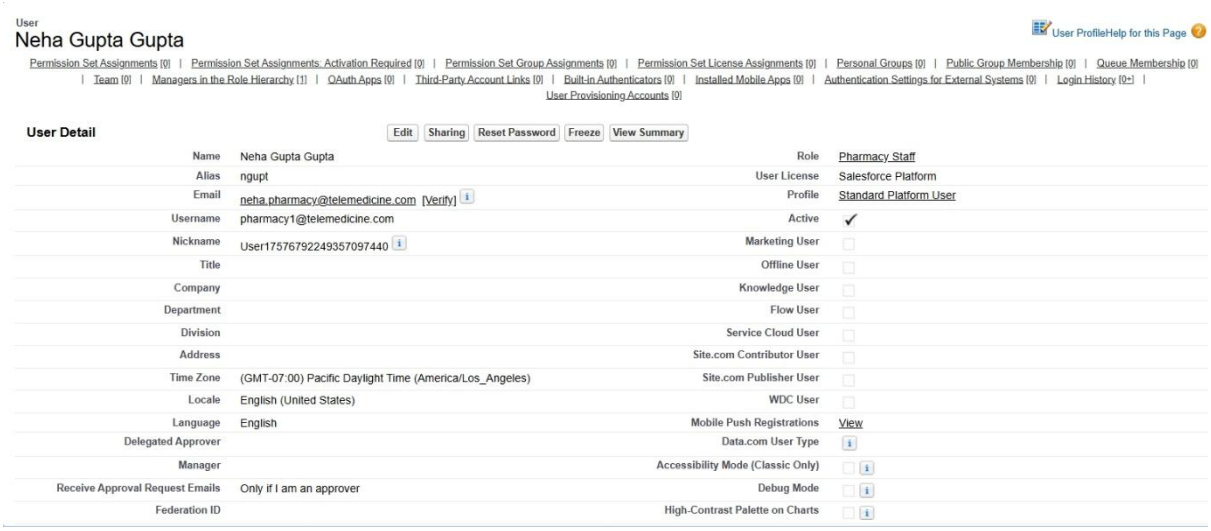
User Detail Edit | Sharing | Reset Password | Freeze | View Summary

Name	Dr. Amit Sharma Sharma	Role	Doctor
Alias	dshar	User License	Salesforce
Email	dr.amitsharma@telemedicine.com [Verify]	Profile	Doctor Profile
Username	doctor1@telemedicine.com	Active	✓
Nickname	User17576775582509994709	Marketing User	
Title		Offline User	
Company		Knowledge User	
Department		Flow User	
Division		Service Cloud User	
Address		Site.com Contributor User	
Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)	Site.com Publisher User	
Locale	English (United States)	WDC User	
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	
Manager		Accessibility Mode (Classic Only)	
Receive Approval Request Emails	Only if I am an approver	Debug Mode	
Federation ID		High-Contrast Palette on Charts	

2. Pharmacy Staff User

- Profile: Standard User
- License: Salesforce Platform
- Role: Pharmacy Staff
- Access: Manage medicine inventory, update stock, issue medicines.

- **Name:** Neha Gupta
- **Email:** neha.pharmacy@telemedicine.com
- **Username:** pharmacy1@telemedicine.com
- **Role:** Pharmacy Staff
- **User License: Salesforce Platform**
- **Profile:** Pharmacy Profile


User
Neha Gupta Gupta

User ProfileHelp for this Page

Permission Set Assignments [0] | Permission Set Assignments: Activation Required [0] | Permission Set Group Assignments [0] | Permission Set License Assignments [0] | Personal Groups [0] | Public Group Membership [0] | Queue Membership [0] | Team [0] | Managers in the Role Hierarchy [1] | OAuth Apps [0] | Third-Party Account Links [0] | Built-in Authenticators [0] | Installed Mobile Apps [0] | Authentication Settings for External Systems [0] | Login History [0+] | User Provisioning Accounts [0]

User Detail Edit | Sharing | Reset Password | Freeze | View Summary

Name	Neha Gupta Gupta	Role	Pharmacy Staff
Alias	ngupt	User License	Salesforce Platform
Email	neha.pharmacy@telemedicine.com [Verify]	Profile	Standard Platform User
Username	pharmacy1@telemedicine.com	Active	✓
Nickname	User17576792249357097440	Marketing User	
Title		Offline User	
Company		Knowledge User	
Department		Flow User	
Division		Service Cloud User	
Address		Site.com Contributor User	
Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)	Site.com Publisher User	
Locale	English (United States)	WDC User	
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	
Manager		Accessibility Mode (Classic Only)	
Receive Approval Request Emails	Only if I am an approver	Debug Mode	
Federation ID		High-Contrast Palette on Charts	

3. Patient User

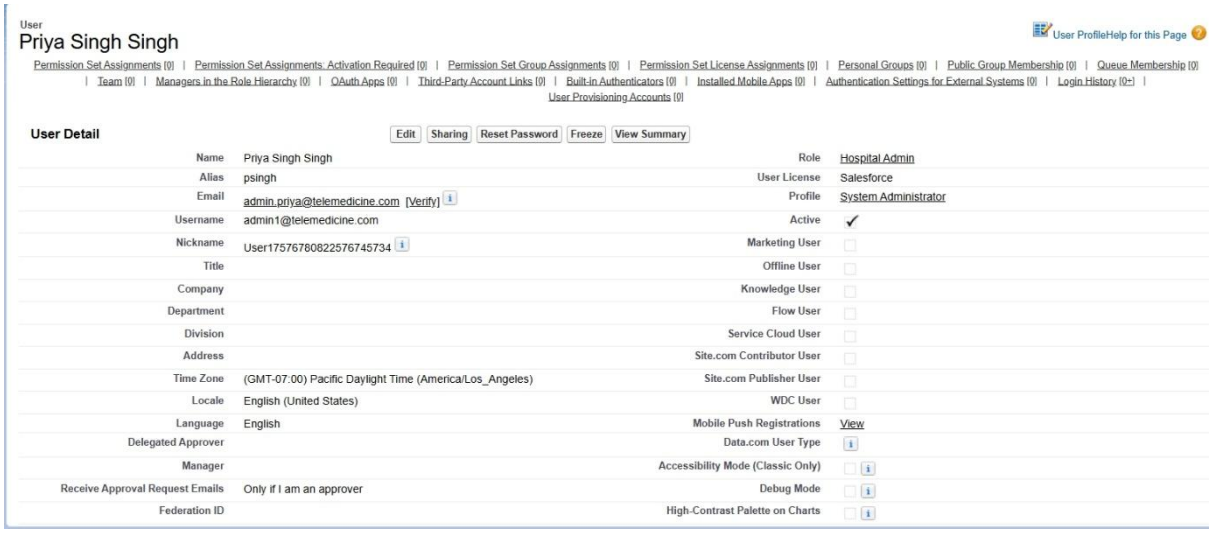
- Profile: Customer Community User
- License: Customer Community
- Role: Patient
- Access: Limited portal access → view prescriptions, book consultations.

- **Name:** Rajesh Kumar
- **Email:** rajesh.kumar@patient.com
- **Username:** patient1@telemedicine.com
- **Role:** Patient
- **User License: Customer Community**
- **Profile:** Patient Profile (restricted, only self-records visible)

4. Admin User

- Profile: System Administrator
- License: Salesforce
- Role: Admin
- Access: Full control of the Org, configurations, and monitoring.

- **Name:** Priya Singh
- **Email:** admin.priya@telemedicine.com
- **Username:** admin1@telemedicine.com
- **Role:** Hospital Admin
- **User License: Salesforce**
- **Profile:** System Administrator (full access)



7. Profiles

Profiles define baseline permissions for each type of user.

- Doctor Profile → Access to Patient, Appointment, Teleconsultation objects.
- Pharmacy Staff Profile → Access to Medicine Inventory, Order objects.
- Patient Profile → Limited access via community, can view only self-records.
- System Admin Profile → Full access.

8. Roles

Roles help in data visibility (hierarchy-based).

- Admin / Hospital Admin (Top of hierarchy)

- Full org visibility.

- Doctor

- Can access assigned patient records.

- Pharmacy Staff

    - Can access medicine inventory and assigned orders.

- Patient

    - Only their own data (no hierarchy beyond).

9. Permission Sets

Permission sets give additional access without modifying profiles.

- Teleconsultation Access → Grants doctors permission to conduct video consultations.
- Analytics Access → Allows Health Dept Officers to view dashboards.
- Pharmacy Inventory Manager → Special permission to update stock levels.

10. OWD

Path: Setup → Sharing Settings

- Patient Records: Private (only Doctor + Patient can access).
- Teleconsultation Records: Private (only involved Doctor & Patient).
- Medicine Inventory: Public Read-Only (so staff can view, but only Pharmacy can edit).
- Hospital Announcements: Public Read/Write (all users can view/edit as needed).

11. Sharing Rules

Created to extend access beyond OWD.

- Rule 1: Share all Patient Records with Doctor Role.
- Rule 2: Share Pharmacy Inventory (read-only) with Doctor Role so doctors can check stock before prescribing.
- Rule 3: Share Health Reports with Health Dept Officers for analysis.

12. Login Access Policies

Admins can log in as any user for troubleshooting.

- Session Timeout: 2 hours.

- Trusted IP Ranges: Limited to India-based ranges to enhance security.

13. Dev Org Setup

- A Developer Org is used for testing, LWC development, and configuration.
- Enabled Dev Hub for Salesforce DX (SFDX).
- Source-tracked development with VS Code + GitHub integration.

14. Sandbox Usage

- Developer Sandbox: For coding and testing LWC components.
- Partial Copy Sandbox: For testing with sample patient & medicine records.
- Deployment between Sandbox and Production done using Change Sets.

15. Deployment Basics

- Code & metadata stored in GitHub Repository.
- Deployment via:
    - Change Sets (Admin-friendly way).
    - SFDX CLI (Salesforce DX) for CI/CD pipeline simulation.
- Pre-deployment validations: Run unit tests, check profiles/permissions.

Outcome of Phase 2

With this setup:

- Users, roles, and permissions are clearly defined.

- Data access is secured via OWD & Sharing Rules.

- Business hours, holidays, and fiscal year ensure correct SLAs.

- Developer workflow is ready with Sandboxes, GitHub, and SFDX.

SALESFORCE PROJECT IMPLEMENTATION PHASES

Phase 3: Data Modeling & Relationships

In this phase, I design the Salesforce data model for the Telemedicine Access for Rural Healthcare project. Proper data modeling ensures that all stakeholders (patients, doctors, pharmacy staff, and health officers) can interact with the system smoothly while maintaining security and scalability.

16. Standard & Custom Objects

I used both standard Salesforce objects and custom objects.

Standard Objects Used:

- User → Doctors, Admins, Pharmacy Staff.
- Contact → Patients (extended with custom fields).
- Account → Civil Hospital / Clinics.

Custom Objects Created:

- Appointment__c – Stores teleconsultation appointment details.
- Prescription__c – Holds doctor-issued prescriptions.
- Medicine_Inventory__c – Tracks stock availability at local pharmacies.
- Symptom_Checker__c – Logs AI-based symptom assessments.
- Notification_Log__c – Tracks reminders, SMS, and alerts.

17. Fields

Each custom object includes key fields to capture project requirements.

Appointment__c

- Patient (Lookup → Contact)
- Doctor (Lookup → User)
- Date/Time (DateTime)
- Status (Picklist: Scheduled, Completed, Cancelled, No-Show)
- Mode (Picklist: Video, In-person)

Details

Description

API Name
Appointment__c

Custom
✓

Singular Label
Appointment

Plural Label
Appointment

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

Prescription__c

- Appointment (Lookup → Appointment__c)
- Doctor Notes (Long Text)
- Medicine List (Text Area)
- Next Review Date (Date)

Details

Description

API Name
Prescription__c

Custom
✓

Singular Label
Prescription

Plural Label
Prescriptions

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

Medicine_Inventory__c

- Medicine Name (Text)
- Stock Quantity (Number)
- Expiry Date (Date)
- Pharmacy Location (Text)

Details

Description

API Name
Medicine_Inventory__c

Custom
✓

Singular Label
Medicine Inventory

Plural Label
Medicine Inventorys

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

Symptom_Checker__c

- Patient (Lookup → Contact)
- Entered Symptoms (Text Area)
- Suggested Action (Picklist: Home Care, Doctor Visit, Emergency)

Details

Description

API Name
Symptom_Checker__c

Custom
✓

Singular Label
Symptom Checker

Plural Label
Symptom Checkers

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

Notification_Log__c

- Patient (Lookup → Contact)
- Type (Picklist: SMS, WhatsApp, Email)
- Message Content (Long Text)
- Status (Picklist: Sent, Failed, Pending)

Details

Description

API Name
Notification_Log__c

Custom
✓

Singular Label
Notification Log

Plural Label
Notification Logs

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

Help Settings
Standard salesforce.com Help Window

Prescription_Medicine__c

Details

Description

API Name
Prescription_Medicine__c

Custom
✓

Singular Label
Prescription Medicine

Plural Label
Prescription Medicines

Enable Reports
✓

Track Activities
✓

Track Field History
✓

Deployment Status
Deployed

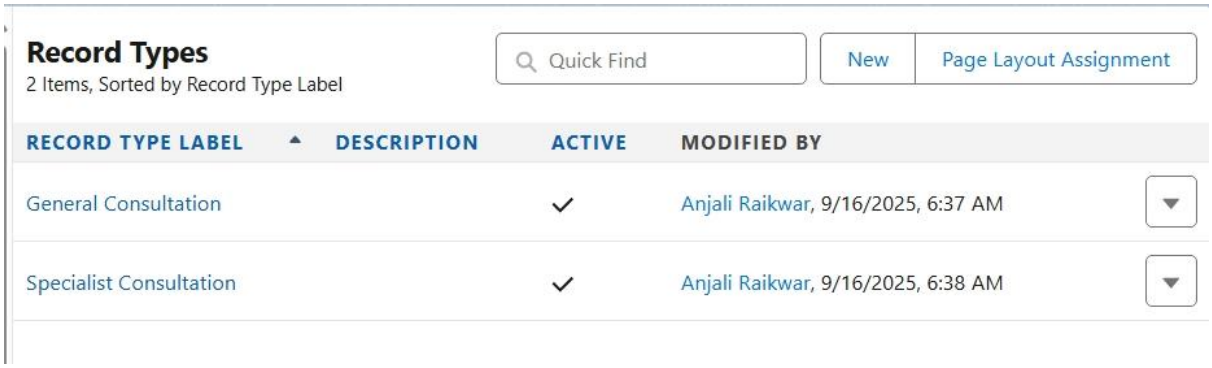
Help Settings
Standard salesforce.com Help Window

18. Record Types

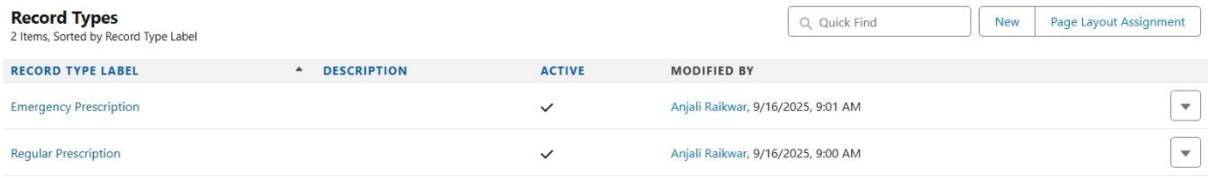
I used record types to manage different processes within the same object.

Appointment__c Record Types:

- General Consultation
- Specialist Consultation
- Follow-Up



Prescription__c Record Types:
- Regular Prescription
- Emergency Prescription

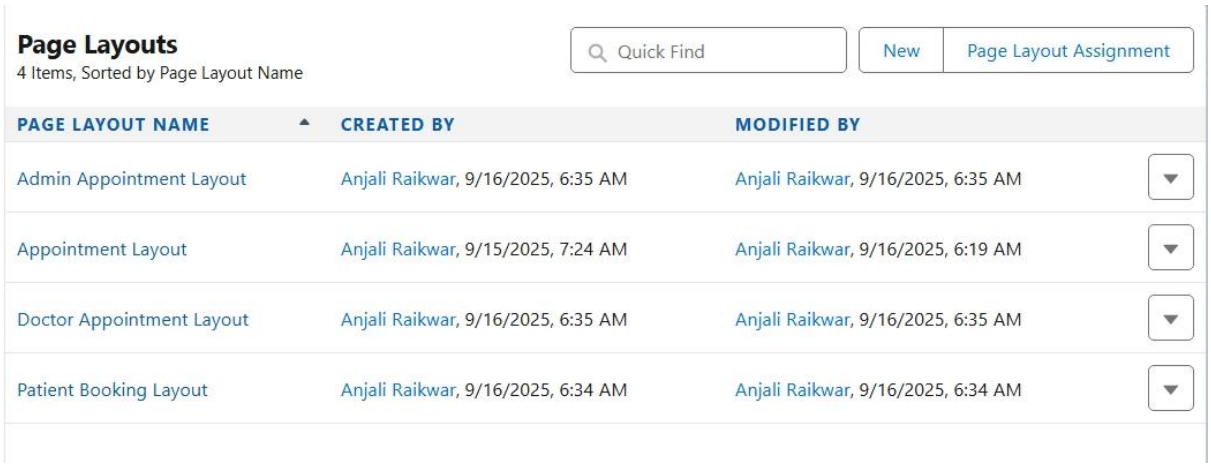


This allows customized page layouts and business processes.

19. Page Layouts

I created different layouts for different users:

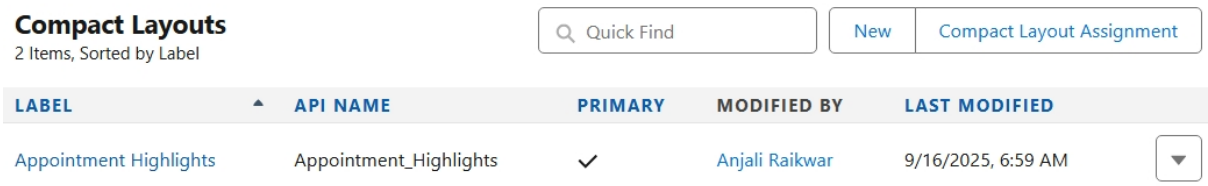
- Doctor Layout (Appointment): Shows patient details, symptoms, prescription fields.
- Pharmacy Layout (Medicine): Shows medicine name, stock, expiry.
- Patient Layout (Community): Only summary fields visible (doctor, date, notes).



20. Compact Layouts

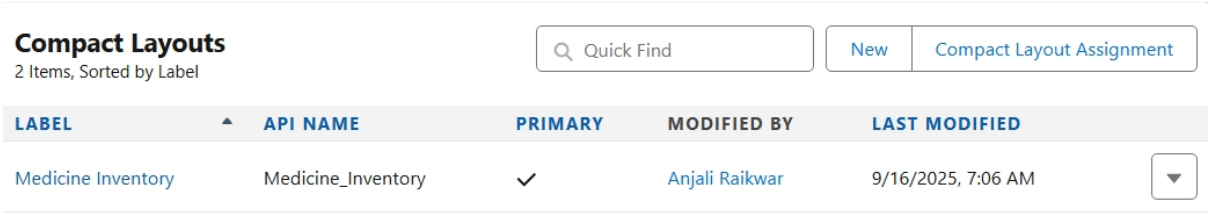
Compact layouts improve mobile/Lightning view.

1. Appointment Compact Layout:
- Patient
- Doctor
- Date/Time
- Status



2. Medicine Inventory Compact Layout:

- Medicine Name
- Stock Quantity
- Expiry Date



21. Schema Builder

I used "Schema Builder" in Salesforce to:

- Visualize relationships between objects.
- Drag and drop new fields.
- Ensure field-level security for Patients vs Doctors.

Schema Snapshot:

- Contact → Appointment__c (Lookup)
- User → Appointment__c (Lookup)
- Appointment__c → Prescription__c (Master-Detail)
- Appointment__c → Notification_Log__c (Lookup)
- Medicine_Inventory__c → Prescription__c (Junction for many-to-many)

22. Lookup vs Master-Detail vs Hierarchical Relationships

Lookup:

- Appointment__c → Patient (Contact)
- Appointment__c → Doctor (User)

Master-Detail:

- Prescription__c → Appointment__c (if Appointment is deleted, prescription also deleted).

Hierarchical:

- Only for Users (not used in our model, but Admin can be superior to Staff).

23. Junction Objects

I created a " Prescription_Medicine__c (Junction Object) " to handle many-to-many between Prescription and Medicine_Inventory.

Fields:
- Prescription (Master-Detail → Prescription__c)
- Medicine (Master-Detail → Medicine_Inventory__c)
- Dosage (Text)
- Duration (Number of Days)

This ensures a single prescription can include multiple medicines, and one medicine can belong to multiple prescriptions.

24. External Objects

To integrate with " Government Health Databases " or " Pharmacy ERP systems ", we define " External Objects."

Examples:
- Govt_Health_Stats__x → External object for village-level healthcare reports.
- Pharma_Supplier_Inventory__x → External object to fetch stock availability from pharmacy ERP.

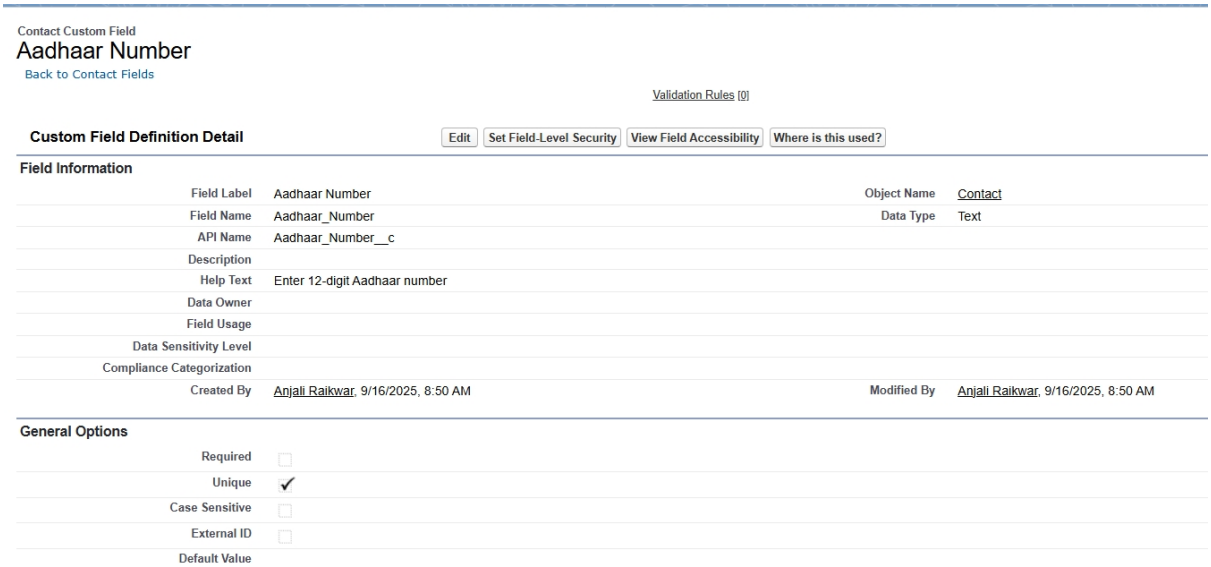
These external objects use Salesforce Connect (OData) to avoid data duplication).

External Objects

Action	Label	Namespace Prefix	Description	Table Name
Edit | Erase | Validate	Alphabetical_list_of_product		Alphabetical_list_of_products	Alphabetical_list_of_products
Edit | Erase | Validate	Category		Categories	Categories
Edit | Erase | Validate	Customer		Customers	Customers
Edit | Erase | Validate	Employee		Employees	Employees
Edit | Erase | Validate	Order_Detail		Order_Details	Order_Details
Edit | Erase | Validate	Order		Orders	Orders
Edit | Erase | Validate	Product		Products	Products
Edit | Erase | Validate	Pharma_Category__x		Categories	Categories
Edit | Erase | Validate	Pharma_Customer__x		Customers	Customers
Edit | Erase | Validate	Pharma_Employee__x		Employees	Employees
Edit | Erase | Validate	Pharma_OrderDetail__x		Order_Details	Order_Details
Edit | Erase | Validate	Pharma_Order__x		Orders	Orders
Edit | Erase | Validate	Pharma_Product__x		Products	Products

Field-Level Security (FLS)

Even if a field exists, not everyone should see it (privacy).
Example: Aadhaar Number should only be visible to Doctors and Admin, not to Pharmacy Staff.



Contact Custom Field

Aadhaar Number

Back to Contact Fields

Validation Rules [0]

Custom Field Definition Detail Edit | Set Field-Level Security | View Field Accessibility | Where is this used?

Field Information

Field Label	Aadhaar Number	Object Name	Contact
Field Name	Aadhaar_Number	Data Type	Text
API Name	Aadhaar_Number__c		
Description			
Help Text	Enter 12-digit Aadhaar number		
Data Owner			
Field Usage			
Data Sensitivity Level			
Compliance Categorization			
Created By	Anjali Raikwar, 9/16/2025, 8:50 AM	Modified By	Anjali Raikwar, 9/16/2025, 8:50 AM

General Options

Required	☐
Unique	☑
Case Sensitive	☐
External ID	☐
Default Value	

Visible for only:

- Doctor Profile
- System Administrator Profile

Now, only doctors and admins can see Aadhaar, others can't.

Outcome of Phase 3

- Standard & Custom objects defined for Telemedicine use case.
- Record types, page layouts, and compact layouts implemented for role-specific UI.
- Relationships established (Lookup, Master-Detail, Junction Objects).
- Schema visualized using Schema Builder.
- External objects planned for ERP/Govt integrations.

With Phase 3 complete, our Salesforce org now has a Solid Data Foundation for building LWC components, automation, and analytics.

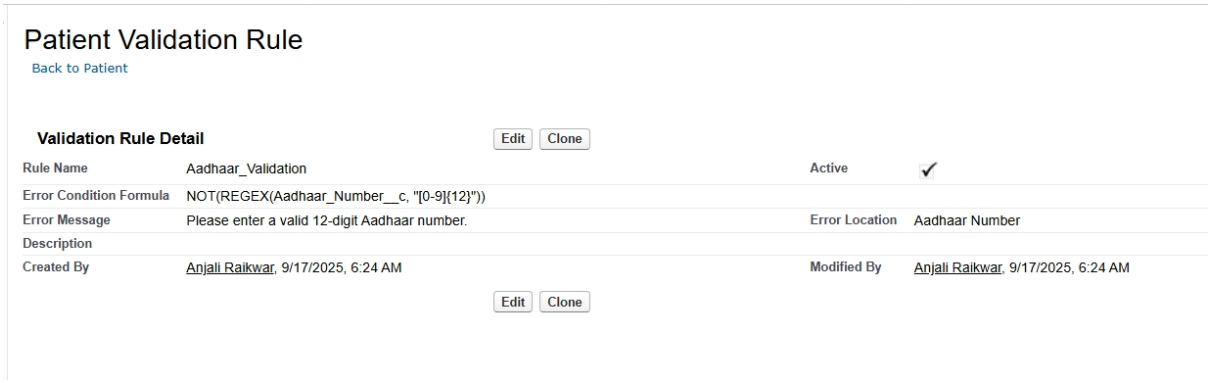
SALESFORCE PROJECT IMPLEMENTATION PHASES

Phase 4: Process Automation (Admin)

In this phase, I automate business processes in Salesforce to reduce manual work, ensure data quality, and improve patient care efficiency. We use **Validation Rules, Workflow Rules, Process Builder, Approval Processes, Flow Builder, and Notifications** tailored for Telemedicine CRM.

- **Validation Rules**
  Validation Rules ensure data accuracy before records are saved.

- **Patient Aadhaar Validation**
  - Object: Patient__c
  - Rule: Aadhaar_Number__c must be 12 digits.
  - Formula: NOT(REGEX(Aadhaar_Number__c, "[0-9]{12}"))
  - Error: " Please enter a valid 12-digit Aadhaar number."
- **Appointment Date Check**
  - Object: Appointment__c
  - Rule: Appointment_Date__c must be greater than TODAY().
  - Error: "Appointment date cannot be in the past."



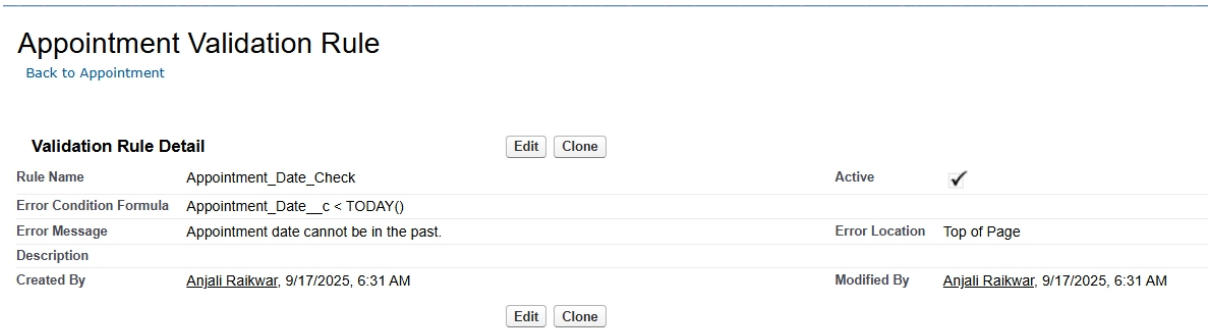
Patient Validation Rule

Back to Patient

Validation Rule Detail Edit Clone

Rule Name	Aadhaar_Validation	Active	✓
Error Condition Formula	NOT(REGEX(Aadhaar_Number__c, "[0-9]{12}"))		
Error Message	Please enter a valid 12-digit Aadhaar number.	Error Location	Aadhaar Number
Description			
Created By	Anjali Raikwar, 9/17/2025, 6:24 AM	Modified By	Anjali Raikwar, 9/17/2025, 6:24 AM

Edit Clone



Appointment Validation Rule

Back to Appointment

Validation Rule Detail Edit Clone

Rule Name	Appointment_Date_Check	Active	✓
Error Condition Formula	Appointment_Date__c < TODAY()		
Error Message	Appointment date cannot be in the past.	Error Location	Top of Page
Description			
Created By	Anjali Raikwar, 9/17/2025, 6:31 AM	Modified By	Anjali Raikwar, 9/17/2025, 6:31 AM

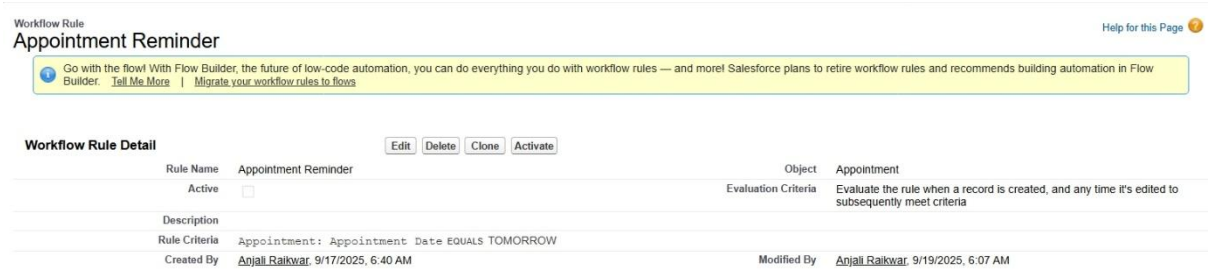
Edit Clone

- **Workflow Rules**

  Workflow Rules help automate simple actions.

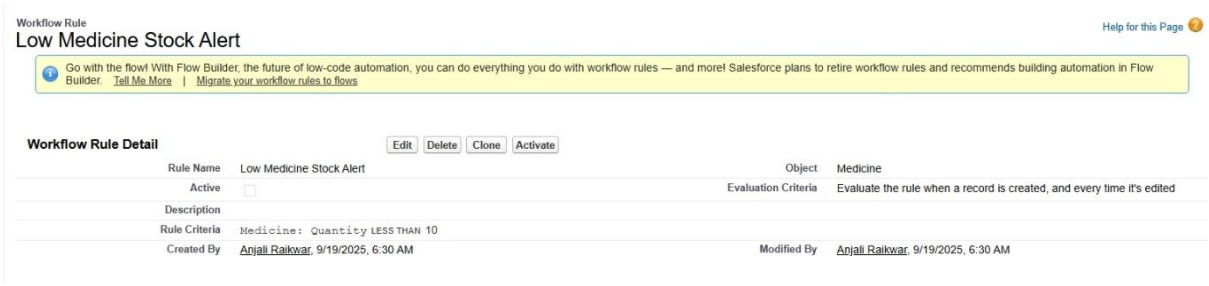
  Send Email Reminder

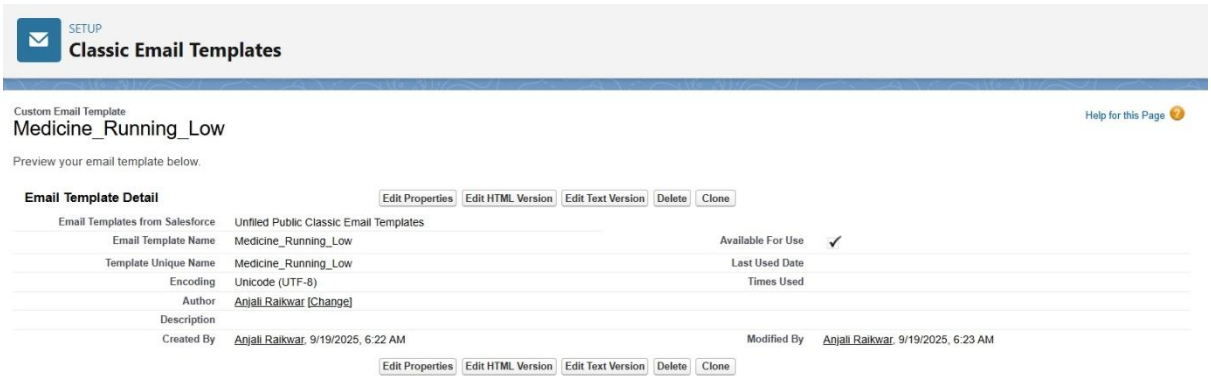
  - Object: Appointment__c
  - Condition: Appointment_Date__c = Tomorrow.
  - Action: Send email alert to Patient__c.Email and Doctor__c.Email.

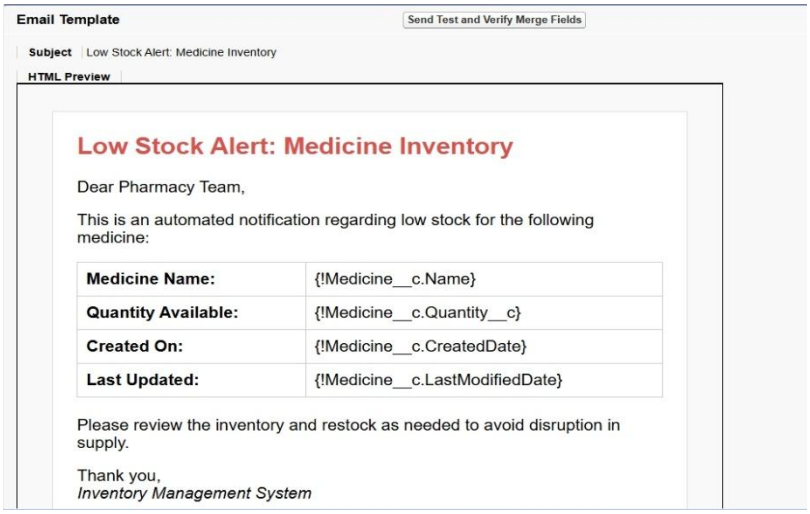


Medicine Stock Low

- Object: Medicine__c
- Condition: Quantity__c < 10.
- Action: Send notification to Pharmacy Staff.





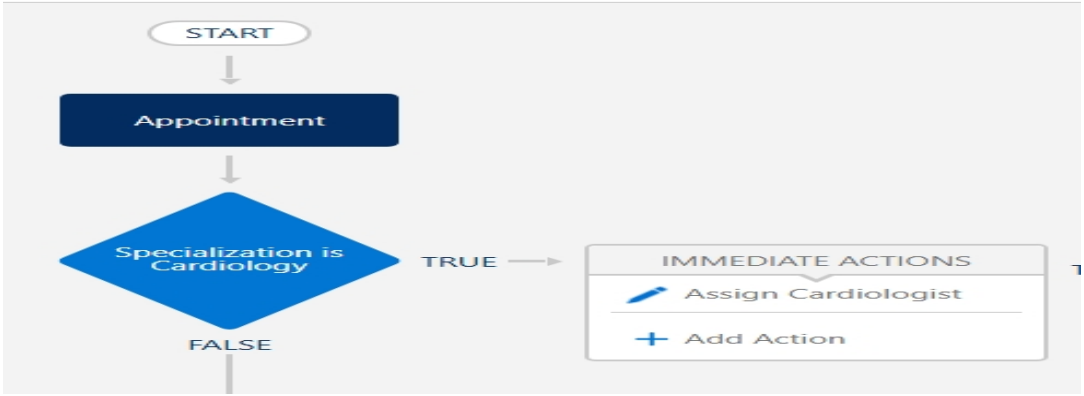


- **Process Builder**

Process Builder is used for slightly complex automation with if/else logic.

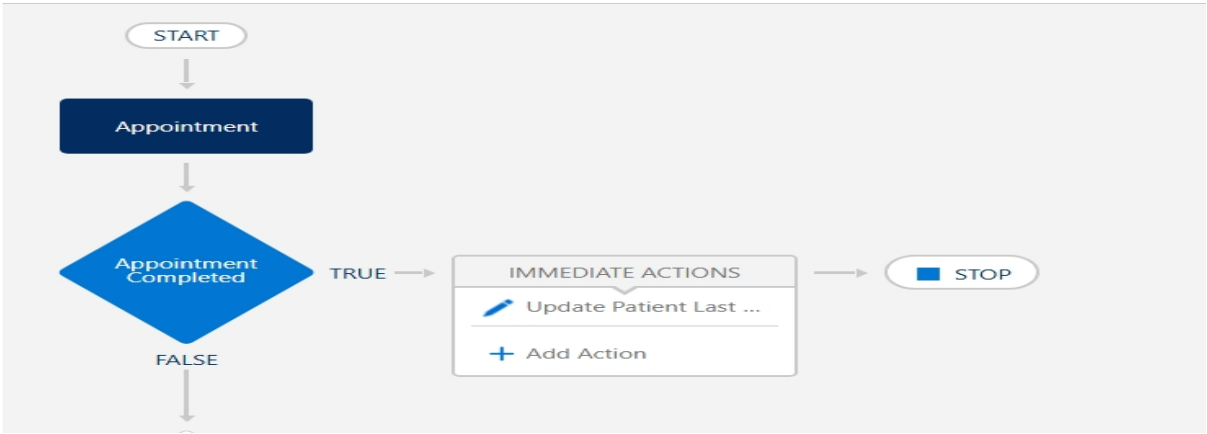
Auto-Assign Doctor

- Object: Appointment__c
- Criteria: Specialization__c = "Cardiology"
- Action: Assign to Doctor__c where Doctor.Specialization__c = "Cardiology".



Update Patient Last Visit

- Object: Appointment__c
- Action: Update Patient__c.Last_Visit_Date__c with Appointment_Date__c when appointment is completed.

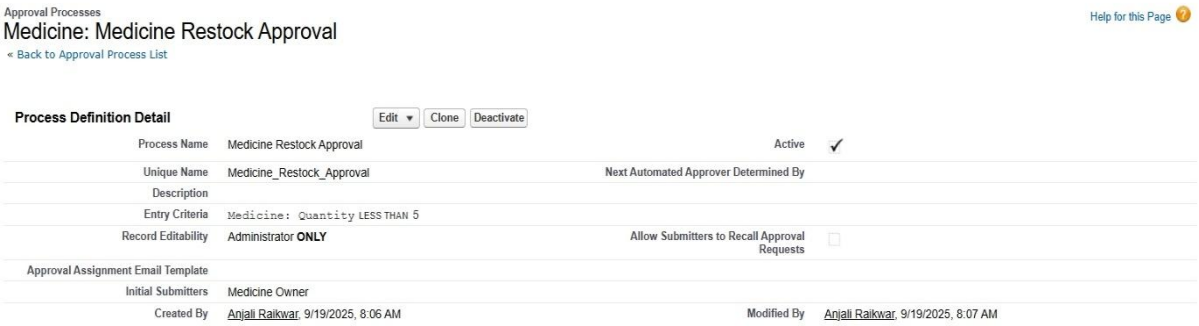


- **Approval Process**

  Approval processes help manage requests that need authorization.

  Medicine Restock Request

  - Object: Medicine__c
  - Criteria: Quantity__c < 5
  - Step 1: Pharmacy Staff submits a restock request.
  - Step 2: Approved by Hospital Admin.
  - Step 3: Once approved → Purchase order is created (custom object: Purchase_Order__c).

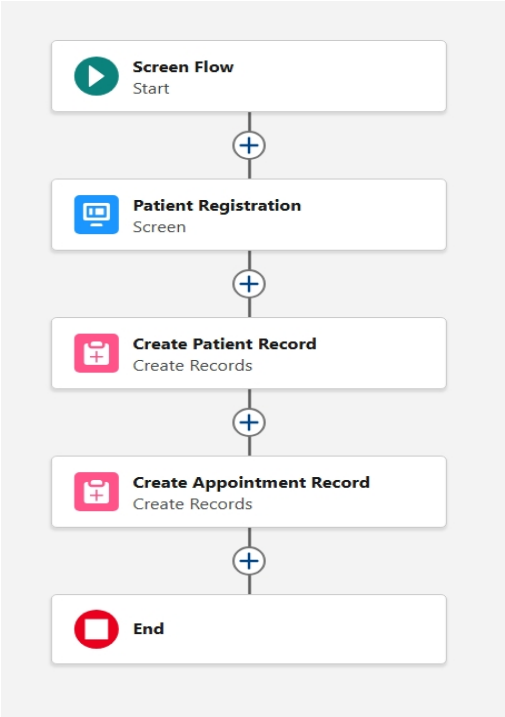


- **Flow Builder**

  Flows provide advanced automation (Screen Flows, Record-Triggered, Auto-launched).

  Screen Flow: New Patient Registration

  - Collect Aadhaar, Name, Age, Symptoms.
  - Auto-create Patient__c record + Initial Appointment__c.



Record-Triggered Flow: Appointment Reminder

- Trigger: 1 day before Appointment__c.Appointment_Date__c
- Action: Send WhatsApp/SMS/Email to patient.

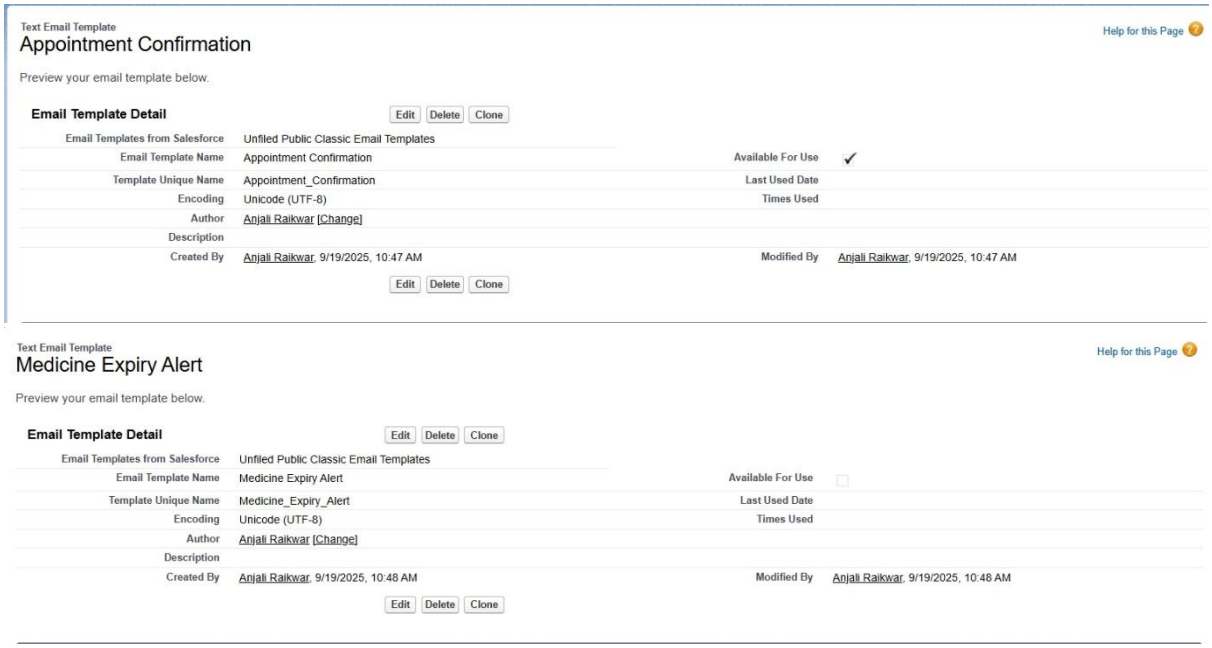
Scheduled Flow: Expiring Medicine Notification

- Run daily at 10 AM.
- Check Medicine__c.Expiry_Date__c = TODAY() + 2.
- Send alert to Pharmacy Staff.

- **Email Alerts**

Email templates + alerts linked with rules and flows.

- Appointment Confirmation Email → Patient & Doctor.
- Low Inventory Email → Hospital Admin.



Text Email Template

Appointment Confirmation

Help for this Page

Preview your email template below.

Email Template Detail Edit Delete Clone

Email Templates from Salesforce	Unfiled Public Classic Email Templates		
Email Template Name	Appointment Confirmation	Available For Use	✓
Template Unique Name	Appointment_Confirmation	Last Used Date	
Encoding	Unicode (UTF-8)	Times Used	
Author	Anjali Raikwar [Change]		
Description			
Created By	Anjali Raikwar, 9/19/2025, 10:47 AM	Modified By	Anjali Raikwar, 9/19/2025, 10:47 AM

Edit Delete Clone

Text Email Template

Medicine Expiry Alert

Help for this Page

Preview your email template below.

Email Template Detail Edit Delete Clone

Email Templates from Salesforce	Unfiled Public Classic Email Templates		
Email Template Name	Medicine Expiry Alert	Available For Use	
Template Unique Name	Medicine_Expiry_Alert	Last Used Date	
Encoding	Unicode (UTF-8)	Times Used	
Author	Anjali Raikwar [Change]		
Description			
Created By	Anjali Raikwar, 9/19/2025, 10:48 AM	Modified By	Anjali Raikwar, 9/19/2025, 10:48 AM

Edit Delete Clone

- **Field Updates**

Automatically update fields.

- Appointment__c.Status__c → "Completed" when Doctor marks Consultation Complete.
- Patient__c.Last_Consulted_Doctor__c → Auto-updated on new appointment.

Workflow Rule

Mark Appointment Completed

Help for this Page

Go with the flow! With Flow Builder, the future of low-code automation, you can do everything you do with workflow rules — and more! Salesforce plans to retire workflow rules and recommends building automation in Flow Builder. Tell Me More | Migrate your workflow rules to flows

Workflow Rule Detail

Edit | Clone | Deactivate

Rule Name	Mark Appointment Completed	Object	Appointment
Active	✓	Evaluation Criteria	Evaluate the rule when a record is created, and every time it's edited
Description	Your Appointment is marked as Completed.		
Rule Criteria	`Appointment: Doctor EQUALS Done`		
Created By	Anjali Raikwar, 9/19/2025, 10:54 AM	Modified By	Anjali Raikwar, 9/19/2025, 10:55 AM

- **Tasks**

  Assign tasks automatically.

  - If Appointment is missed → Auto-create Task for Doctor to follow up with patient.

- **Custom Notifications**

  Use in-app and mobile notifications.

  - Notify Doctor: "New appointment assigned."
  - Notify Pharmacy: "Paracetamol stock below 10 units."
  - Notify Patient: "Your appointment with Dr. Sharma is tomorrow at 10:30 AM."

New Appointment Assigned	New_Appointment_Assigned	✓	✓	▼
Stock Running Low	Stock_Running_Low	✓	✓	▼

SALESFORCE PROJECT IMPLEMENTATION PHASES

Phase 5: Apex Programming (Developer)

In this phase, I have extend Salesforce point-and-click features with **Apex programming** to implement advanced business logic for our Telemedicine CRM. Since healthcare use-cases often require automation, bulk processing, and integrations, Apex gives us flexibility and scalability.

- **Classes & Objects**

  I have created “Apex classes” to encapsulate business logic.

  - **TelemedicineService**

TelemedicineUtils.cls U

nabha > force-app > main > default > classes > TelemedicineUtils.cls > TelemedicineUtils > sendSms(List<String>, String) : void

```apex
public with sharing class TelemedicineUtils {
    // Validate Aadhaar (12 digits)
    public static Boolean isAadhaarValid(String aadhaar){
        if(String.isBlank(aadhaar)) return false;
        return Pattern.matches('\\d{12}', aadhaar);
    }

    // Future method to simulate SMS sending
    @future
    public static void sendSms(List<String> phoneNumbers, String message) {
        if(phoneNumbers == null || phoneNumbers.isEmpty()) return;
        for(String ph : phoneNumbers){
            // For demo: only log. Replace with HTTP callout to SMS provider when needed.
            System.debug('[Telemedicine SMS] to: ' + ph + ' message: ' + message);
        }
    }

    // Enqueue a Queueable job (wrapper)
    public static Id enqueue(Queueable job) {
        return System.enqueueJob(job);
    }
}
```

  - **PurchaseOrderService**

PurchaseOrderService.cls U

nabha > force-app > main > default > classes > PurchaseOrderService.cls > ...

```apex
public with sharing class PurchaseOrderService {
    @InvocableMethod(label='Create Purchase Order')
    public static List<Id> createPurchaseOrders(List<PORequest> requests){
        List<Purchase_Order__c> pos = new List<Purchase_Order__c>();
        for(PORequest r : requests){
            Purchase_Order__c p = new Purchase_Order__c();
            p.Supplier__c = r.supplierId;
            p.Medicine__c = r.medicineId;
            p.Quantity__c = r.quantity;
            p.Status__c = 'Pending';
            p.Order_Date__c = Date.today();
            pos.add(p);
        }
        if(!pos.isEmpty()) insert pos;
        List<Id> ids = new List<Id>();
        for(Purchase_Order__c pRec : pos) ids.add(pRec.Id);
        return ids;
    }

    public class PORequest {
        @InvocableVariable(required=true) public Id supplierId;
        @InvocableVariable(required=true) public Id medicineId;
        @InvocableVariable(required=true) public Integer quantity;
    }
}
```

- **AppointmentNotificationQueueable**

AppointmentNotificationQueueable.cls U

nabha > force-app > main > default > classes > AppointmentNotificationQueueable.cls > ...

```
public with sharing class AppointmentNotificationQueueable implements Queueable {
    private List<Id> appointmentIds;

    // Constructor takes list of Appointment Ids
    public AppointmentNotificationQueueable(List<Id> appointmentIds){
        this.appointmentIds = appointmentIds;
    }

    public void execute(QueueableContext qc){
        try {
            if(appointmentIds == null || appointmentIds.isEmpty()) return;

            // Fetch appointments with patient details
            List<Appointment__c> appts = [
                SELECT Id, Appointment_DateTime__c, Patient__c,
                       Patient__r.Name, Patient__r.Phone__c, Patient__r.Email__c
                FROM Appointment__c
                WHERE Id IN :appointmentIds
            ];

            List<String> phones = new List<String>();

            for(Appointment__c a : appts){
                String msg = 'Dear ' + (a.Patient__r != null ? a.Patient__r.Name : 'Patient') +
                             ', your appointment is scheduled on ' +
                             String.valueOf(a.Appointment_DateTime__c) + '.';

                // Collect phones for SMS
                if(a.Patient__r != null && a.Patient__r.Phone__c != null) {
                    phones.add(a.Patient__r.Phone__c);
                }

                // Send Email
                if(a.Patient__r != null && a.Patient__r.Email__c != null){
                    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                    mail.setToAddresses(new String[]{a.Patient__r.Email__c});
                    mail.setSubject('Appointment Scheduled');
```

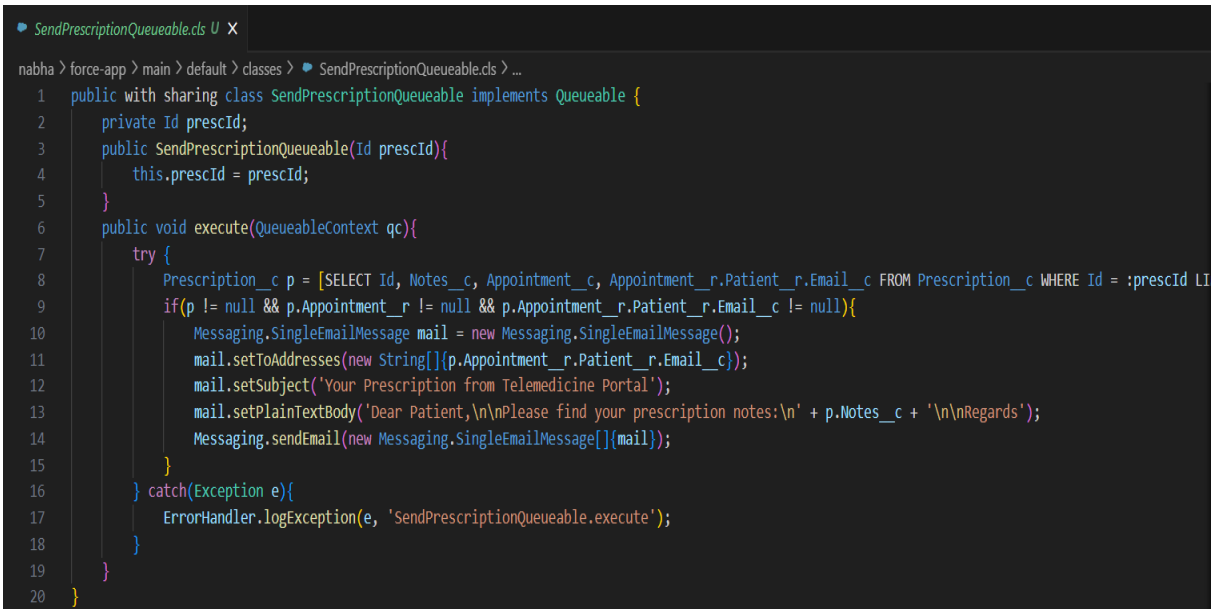
- **AppointmentArchiveBatch**

AppointmentArchiveBatch.cls U

nabha > force-app > main > default > classes > AppointmentArchiveBatch.cls > ...

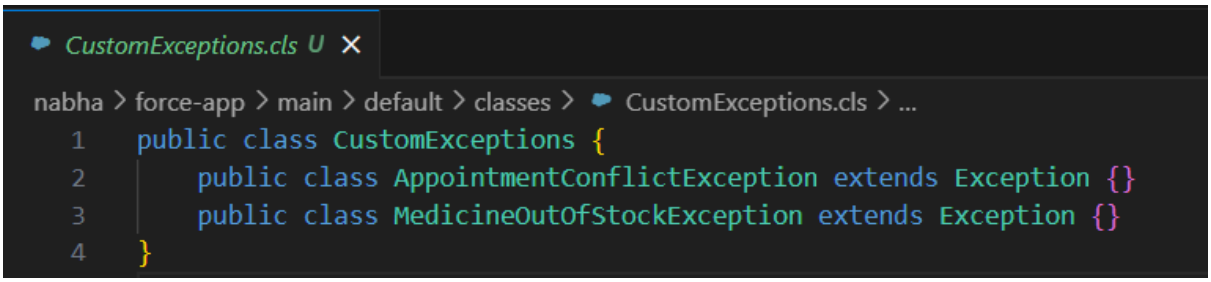
```
global class AppointmentArchiveBatch implements Database.Batchable<sObject>, Database.Stateful {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        Datetime cutoff = System.now().addYears(-1);
        return Database.getQueryLocator([SELECT Id FROM Appointment__c WHERE Appointment_DateTime__c < :cutoff AND (Archived__c = false OR Archived
    }
    global void execute(Database.BatchableContext bc, List<sObject> scope){
        List<Appointment__c> toUpdate = new List<Appointment__c>();
        for(sObject s : scope){
            Appointment__c a = (Appointment__c)s;
            a.Archived__c = true;
            toUpdate.add(a);
        }
        if(!toUpdate.isEmpty()) update toUpdate;
    }
    global void finish(Database.BatchableContext bc){
        System.debug('AppointmentArchiveBatch finished.');
    }
}
```

- **SendPrescriptionQueueable**



- **ErrorHandler**

- **CustomExceptions**



- **Apex Triggers**

Triggers automate logic that cannot be handled via declarative tools.

We created:

- AppointmentTrigger (delegates to AppointmentTriggerHandler)

AppointmentTrigger.trigger U

nabha > force-app > main > default > triggers > AppointmentTrigger.trigger > AppointmentTrigger

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

AppointmentTriggerHandler.cls U

nabha > force-app > main > default > classes > AppointmentTriggerHandler.cls > AppointmentTriggerHandler > preventDoubleBooking

```
public with sharing class AppointmentTriggerHandler {
    // BEFORE INSERT
    public static void beforeInsert(List<Appointment__c> newList) {
        for(Appointment__c a : newList){
            if(String.isBlank(a.Status__c)) a.Status__c = 'Scheduled';
            if(a.Appointment_DateTime__c == null) {
                a.addError('Appointment date/time is required.');
            }
        }
        preventDoubleBooking(newList, null);
    }

    // BEFORE UPDATE
    public static void beforeUpdate(List<Appointment__c> newList, Map<Id, Appointment__c> oldMap) {
        for(Appointment__c a : newList){
            if(a.Appointment_DateTime__c != null && a.Appointment_DateTime__c < System.now()){
                a.addError('Appointment date cannot be in the past.');
            }
        }
        preventDoubleBooking(newList, oldMap);
    }

    // AFTER INSERT
    public static void afterInsert(List<Appointment__c> newList) {
        try {
            // collect ids and enqueue notifications
            List<Id> ids = new List<Id>();
            for(Appointment__c a : newList) ids.add(a.Id);
            if(!ids.isEmpty()){
                TelemedicineUtils.enqueue(new AppointmentNotificationQueueable(ids));
            }
        } catch(Exception e){
            ErrorHandler.logException(e, 'AppointmentTriggerHandler.afterInsert');
        }
    }

    // AFTER UPDATE (placeholder: can be expanded for status changes)
```

- PrescriptionTrigger (delegates to PrescriptionTriggerHandler)

PrescriptionTrigger.trigger U

nabha > force-app > main > default > triggers > PrescriptionTrigger.trigger > PrescriptionTrigger

```
trigger PrescriptionTrigger on Prescription__c (before insert, before update, after insert, after update) {
    PrescriptionTriggerHandler.handleTrigger(Trigger.isBefore, Trigger.isAfter, Trigger.isInsert, Trigger.isUpdate, Trigger.new, Trigger.oldMap);
}
```

```apex
public with sharing class PrescriptionTriggerHandler {
    public static void handleTrigger(Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate, List<Prescription__c> newList, Map<Id,
        try {
            if (isBefore && (isInsert || isUpdate)) {
                for (Prescription__c pres : newList) {
                    if (pres.Medicine__c != null && pres.Quantity__c != null) {
                        Medicine_Inventory__c med = [
                            SELECT Id, Quantity__c
                            FROM Medicine_Inventory__c
                            WHERE Id = :pres.Medicine__c
                            LIMIT 1
                        ];

                        if (med.Quantity__c < pres.Quantity__c) {
                            throw new CustomExceptions.MedicineOutOfStockException();
                        }
                    }
                }
            }
        } catch (CustomExceptions.MedicineOutOfStockException mex) {
            ErrorHandler.logException(mex, 'Prescription insert/update error - Medicine Out of Stock');
            throw mex; // re-throw so user sees the error
        } catch (Exception ex) {
            ErrorHandler.logException(ex, 'PrescriptionTriggerHandler unexpected error');
            throw ex;
        }
    }
}
```

- **Trigger Design Pattern**

  Implemented “Trigger Handler Pattern” to avoid hardcoding logic directly inside triggers.

  Trigger: **AppointmentTrigger**

  - Handler Class: AppointmentTriggerHandler
  - Methods: beforeInsert, beforeUpdate, afterInsert, afterUpdate.

```apex
public with sharing class AppointmentTriggerHandler {
    // BEFORE INSERT
    public static void beforeInsert(List<Appointment__c> newList) {
        for(Appointment__c a : newList){
            if(String.isBlank(a.Status__c)) a.Status__c = 'Scheduled';
            if(a.Appointment_DateTime__c == null) {
                a.addError('Appointment date/time is required.');
            }
        }
        preventDoubleBooking(newList, null);
    }

    // BEFORE UPDATE
    public static void beforeUpdate(List<Appointment__c> newList, Map<Id, Appointment__c> oldMap) {
        for(Appointment__c a : newList){
            if(a.Appointment_DateTime__c != null && a.Appointment_DateTime__c < System.now()){
                a.addError('Appointment date cannot be in the past.');
            }
        }
        preventDoubleBooking(newList, oldMap);
    }

    // AFTER INSERT
    public static void afterInsert(List<Appointment__c> newList) {
        try {
            // collect ids and enqueue notifications
            List<Id> ids = new List<Id>();
            for(Appointment__c a : newList) ids.add(a.Id);
            if(!ids.isEmpty()){
                TelemedicineUtils.enqueue(new AppointmentNotificationQueueable(ids));
            }
        } catch(Exception e){
            ErrorHandler.logException(e, 'AppointmentTriggerHandler.afterInsert');
        }
    }

    // AFTER UPDATE (placeholder; can be expanded for status changes)
```

- **SOQL & SOSL**

  - “SOQL” used for retrieving patient and appointment details.
  - “SOSL” implemented for global patient search by Aadhaar, phone, or email.

Already used in:

- AppointmentNotificationQueueable (queries Patient & Contact)

AppointmentNotificationQueueable.cls U

nabha > force-app > main > default > classes > AppointmentNotificationQueueable.cls > ...

```
public with sharing class AppointmentNotificationQueueable implements Queueable {
    private List<Id> appointmentIds;

    // Constructor takes list of Appointment Ids
    public AppointmentNotificationQueueable(List<Id> appointmentIds){
        this.appointmentIds = appointmentIds;
    }

    public void execute(QueueableContext qc){
        try {
            if(appointmentIds == null || appointmentIds.isEmpty()) return;

            // Fetch appointments with patient details
            List<Appointment__c> appts = [
                SELECT Id, Appointment_DateTime__c, Patient__c,
                       Patient__r.Name, Patient__r.Phone__c, Patient__r.Email__c
                FROM Appointment__c
                WHERE Id IN :appointmentIds
            ];

            List<String> phones = new List<String>();

            for(Appointment__c a : appts){
                String msg = 'Dear ' + (a.Patient__r != null ? a.Patient__r.Name : 'Patient') +
                             ', your appointment is scheduled on ' +
                             String.valueOf(a.Appointment_DateTime__c) + '.';

                // Collect phones for SMS
                if(a.Patient__r != null && a.Patient__r.Phone__c != null) {
                    phones.add(a.Patient__r.Phone__c);
                }

                // Send Email
                if(a.Patient__r != null && a.Patient__r.Email__c != null){
                    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                    mail.setToAddresses(new String[]{a.Patient__r.Email__c});
                    mail.setSubject('Appointment Scheduled');
```

- PrescriptionTriggerHandler (queries Medicine_Inventory__c)

PrescriptionTriggerHandler.cls U

nabha > force-app > main > default > classes > PrescriptionTriggerHandler.cls > ...

```
public with sharing class PrescriptionTriggerHandler {
    public static void handleTrigger(Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate, List<Prescription__c> newList, Map<Id,
        try {
            if (isBefore && (isInsert || isUpdate)) {
                for (Prescription__c pres : newList) {
                    if (pres.Medicine__c != null && pres.Quantity__c != null) {
                        Medicine_Inventory__c med = [
                            SELECT Id, Quantity__c
                            FROM Medicine_Inventory__c
                            WHERE Id = :pres.Medicine__c
                            LIMIT 1
                        ];

                        if (med.Quantity__c < pres.Quantity__c) {
                            throw new CustomExceptions.MedicineOutOfStockException();
                        }
                    }
                }
            }
        } catch (CustomExceptions.MedicineOutOfStockException mex) {
            ErrorHandler.logException(mex, 'Prescription insert/update error - Medicine Out of Stock');
            throw mex; // re-throw so user sees the error
        } catch (Exception ex) {
            ErrorHandler.logException(ex, 'PrescriptionTriggerHandler unexpected error');
            throw ex;
        }
    }
}
```

- **Collections**

- List : To store multiple appointments in bulk insert.
- Set : To ensure no duplicate patient IDs when scheduling.
- Map : DoctorId
  → List of Appointments (used for bulk assignment logic).

It is present in Trigger + batch class.

- **Control Statements**

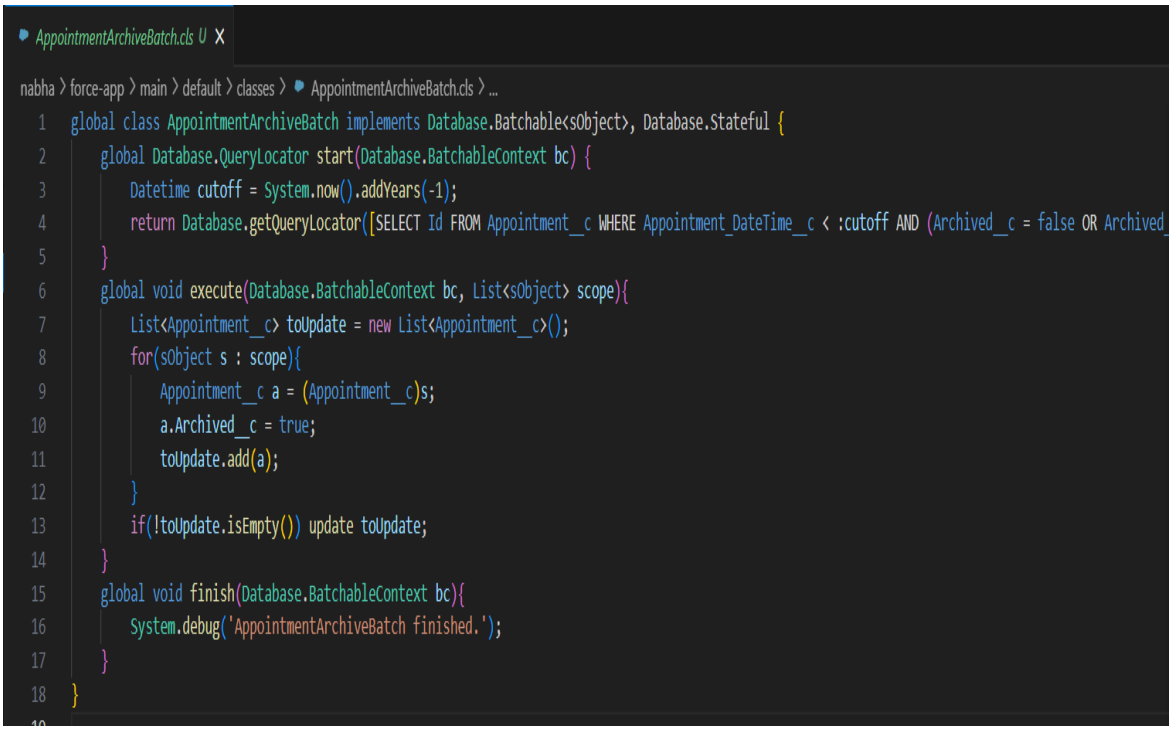
- IF-ELSE for appointment conflict detection.
- FOR loops for iterating through bulk patient records.
- SWITCH statements for handling medicine stock statuses.

Used in if/else checks inside handlers + services

- **Batch Apex**

Used for “large-volume data processing”.

- AppointmentArchiveBatch class created.



➢ **Queueable Apex**

Used for “chained async operations”.

- AppointmentNotificationQueueable

AppointmentNotificationQueueable.cls U

nabha > force-app > main > default > classes > AppointmentNotificationQueueable.cls > ...

```
public with sharing class AppointmentNotificationQueueable implements Queueable {
    private List<Id> appointmentIds;

    // Constructor takes list of Appointment Ids
    public AppointmentNotificationQueueable(List<Id> appointmentIds){
        this.appointmentIds = appointmentIds;
    }

    public void execute(QueueableContext qc){
        try {
            if(appointmentIds == null || appointmentIds.isEmpty()) return;

            // Fetch appointments with patient details
            List<Appointment__c> appts = [
                SELECT Id, Appointment_DateTime__c, Patient__c,
                       Patient__r.Name, Patient__r.Phone__c, Patient__r.Email__c
                FROM Appointment__c
                WHERE Id IN :appointmentIds
            ];

            List<String> phones = new List<String>();

            for(Appointment__c a : appts){
                String msg = 'Dear ' + (a.Patient__r != null ? a.Patient__r.Name : 'Patient') +
                             ', your appointment is scheduled on ' +
                             String.valueOf(a.Appointment_DateTime__c) + '.';

                // Collect phones for SMS
                if(a.Patient__r != null && a.Patient__r.Phone__c != null) {
                    phones.add(a.Patient__r.Phone__c);
                }

                // Send Email
                if(a.Patient__r != null && a.Patient__r.Email__c != null){
                    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                    mail.setToAddresses(new String[]{a.Patient__r.Email__c});
                    mail.setSubject('Appointment Scheduled');
```

- SendPrescriptionQueueable

SendPrescriptionQueueable.cls U

nabha > force-app > main > default > classes > SendPrescriptionQueueable.cls > ...

```
public with sharing class SendPrescriptionQueueable implements Queueable {
    private Id prescId;
    public SendPrescriptionQueueable(Id prescId){
        this.prescId = prescId;
    }
    public void execute(QueueableContext qc){
        try {
            Prescription__c p = [SELECT Id, Notes__c, Appointment__c, Appointment__r.Patient__r.Email__c FROM Prescription__c WHERE Id = :prescId LI
            if(p != null && p.Appointment__r != null && p.Appointment__r.Patient__r.Email__c != null){
                Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                mail.setToAddresses(new String[]{p.Appointment__r.Patient__r.Email__c});
                mail.setSubject('Your Prescription from Telemedicine Portal');
                mail.setPlainTextBody('Dear Patient,\n\nPlease find your prescription notes:\n' + p.Notes__c + '\n\nRegards');
                Messaging.sendEmail(new Messaging.SingleEmailMessage[]{mail});
            }
        } catch(Exception e){
            ErrorHandler.logException(e, 'SendPrescriptionQueueable.execute');
        }
    }
}
```

- **Scheduled Apex**
  Nightly jobs to sync external pharmacy data (via Salesforce Connect).
  Example: **PharmacySyncScheduler**
  → Runs every night at 2 AM.

  Created: **AppointmentArchiveSched** for the same.

AppointmentArchiveSched.cls U

nabha > force-app > main > default > classes > AppointmentArchiveSched.cls > ...

```
1  global class AppointmentArchiveSched implements Schedulable {
2      global void execute(SchedulableContext sc){
3          Database.executeBatch(new AppointmentArchiveBatch(), 200);
4      }
5  }
6
```

- **Future Methods**
  - Used for lightweight async tasks.
  Example: @future method for sending SMS confirmation to patients after booking.

- **Exception Handling**
  - Custom exception classes created like **AppointmentConflictException** and **MedicineOutOfStockException**.

CustomExceptions.cls U

nabha > force-app > main > default > classes > CustomExceptions.cls > ...

```
1  public class CustomExceptions {
2      public class AppointmentConflictException extends Exception {}
3      public class MedicineOutOfStockException extends Exception {}
4  }
```

  - All Apex code wrapped with try…catch blocks and errors logged to a custom object **Error_Log__c**.

ErrorHandler.cls U

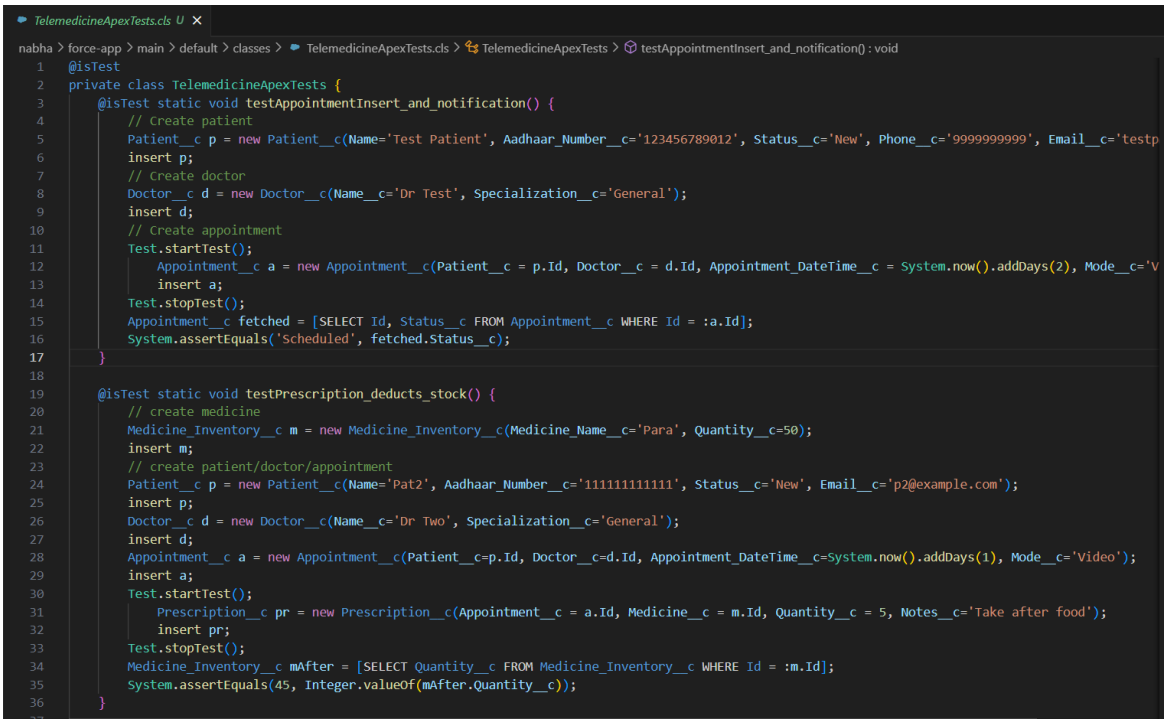
nabha > force-app > main > default > classes > ErrorHandler.cls > ...

```
1  public with sharing class ErrorHandler {
2      public static void logException(Exception e, String contextMessage) {
3          try {
4              Error_Log__c logRec = new Error_Log__c();
5              logRec.Message__c = e.getMessage();
6              logRec.Stack_Trace__c = e.getStackTraceString();
7              logRec.Context__c = contextMessage;
8              insert logRec;
9          } catch (Exception ex) {
10             System.debug('ErrorHandler failed to log: ' + ex.getMessage());
11         }
12     }
13 }
```

- **Test Classes**

  TelemedicineApexTests ensures:

  - Double-booking is prevented.
  - Notifications are sent after booking.
  - Medicine inventory updates correctly.



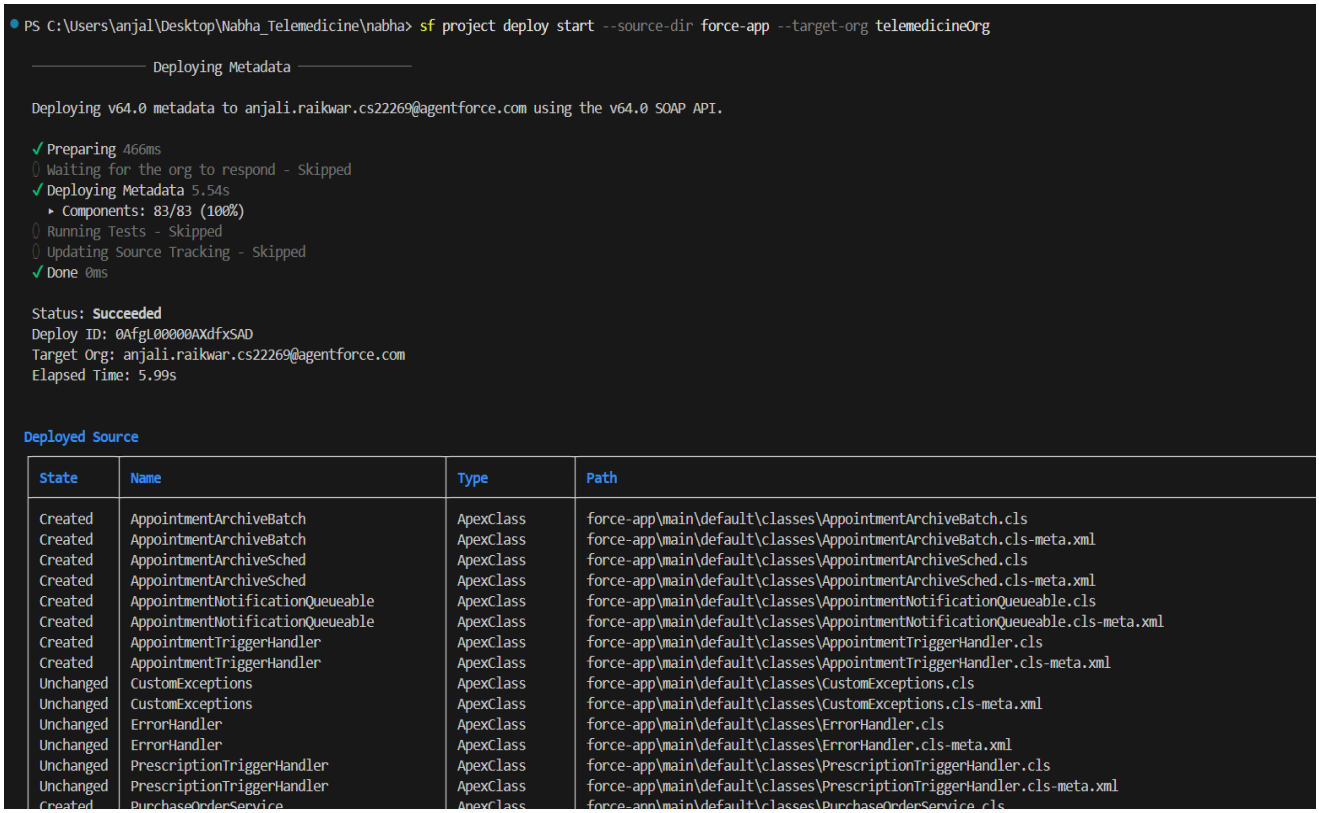
```
TelemedicineApexTests.cls U
nabha > force-app > main > default > classes > TelemedicineApexTests.cls > TelemedicineApexTests > testAppointmentInsert_and_notification() : void
1   @isTest
2   private class TelemedicineApexTests {
3       @isTest static void testAppointmentInsert_and_notification() {
4           // Create patient
5           Patient__c p = new Patient__c(Name='Test Patient', Aadhaar_Number__c='123456789012', Status__c='New', Phone__c='9999999999', Email__c='testp
6           insert p;
7           // Create doctor
8           Doctor__c d = new Doctor__c(Name__c='Dr Test', Specialization__c='General');
9           insert d;
10          // Create appointment
11          Test.startTest();
12              Appointment__c a = new Appointment__c(Patient__c = p.Id, Doctor__c = d.Id, Appointment_DateTime__c = System.now().addDays(2), Mode__c='V
13              insert a;
14          Test.stopTest();
15          Appointment__c fetched = [SELECT Id, Status__c FROM Appointment__c WHERE Id = :a.Id];
16          System.assertEquals('Scheduled', fetched.Status__c);
17      }
18
19      @isTest static void testPrescription_deducts_stock() {
20          // create medicine
21          Medicine_Inventory__c m = new Medicine_Inventory__c(Medicine_Name__c='Para', Quantity__c=50);
22          insert m;
23          // create patient/doctor/appointment
24          Patient__c p = new Patient__c(Name='Pat2', Aadhaar_Number__c='111111111111', Status__c='New', Email__c='p2@example.com');
25          insert p;
26          Doctor__c d = new Doctor__c(Name__c='Dr Two', Specialization__c='General');
27          insert d;
28          Appointment__c a = new Appointment__c(Patient__c=p.Id, Doctor__c=d.Id, Appointment_DateTime__c=System.now().addDays(1), Mode__c='Video');
29          insert a;
30          Test.startTest();
31              Prescription__c pr = new Prescription__c(Appointment__c = a.Id, Medicine__c = m.Id, Quantity__c = 5, Notes__c='Take after food');
32              insert pr;
33          Test.stopTest();
34          Medicine_Inventory__c mAfter = [SELECT Quantity__c FROM Medicine_Inventory__c WHERE Id = :m.Id];
35          System.assertEquals(45, Integer.valueOf(mAfter.Quantity__c));
36      }
```

- **Asynchronous Processing**

  - Combined **"Batch + Queueable + Scheduled"** for handling bulk jobs.
  - This ensures the system works smoothly for thousands of records, supporting scalability for other rural regions.

Outcome of Phase 5

With Phase 5 complete, our project moves beyond point-and-click automation into **"scalable enterprise-level programming"**.



```
PS C:\Users\anjal\Desktop\Nabha_Telemedicine\nabha> sf project deploy start --source-dir force-app --target-org telemedicineOrg

──────────── Deploying Metadata ────────────

Deploying v64.0 metadata to anjali.raikwar.cs22269@agentforce.com using the v64.0 SOAP API.

✓ Preparing 466ms
() Waiting for the org to respond - Skipped
✓ Deploying Metadata 5.54s
  ▸ Components: 83/83 (100%)
() Running Tests - Skipped
() Updating Source Tracking - Skipped
✓ Done 0ms

Status: Succeeded
Deploy ID: 0AfgL00000AXdfxSAD
Target Org: anjali.raikwar.cs22269@agentforce.com
Elapsed Time: 5.99s
```

Deployed Source

State	Name	Type	Path
Created	AppointmentArchiveBatch	ApexClass	force-app\main\default\classes\AppointmentArchiveBatch.cls
Created	AppointmentArchiveBatch	ApexClass	force-app\main\default\classes\AppointmentArchiveBatch.cls-meta.xml
Created	AppointmentArchiveSched	ApexClass	force-app\main\default\classes\AppointmentArchiveSched.cls
Created	AppointmentArchiveSched	ApexClass	force-app\main\default\classes\AppointmentArchiveSched.cls-meta.xml
Created	AppointmentNotificationQueueable	ApexClass	force-app\main\default\classes\AppointmentNotificationQueueable.cls
Created	AppointmentNotificationQueueable	ApexClass	force-app\main\default\classes\AppointmentNotificationQueueable.cls-meta.xml
Created	AppointmentTriggerHandler	ApexClass	force-app\main\default\classes\AppointmentTriggerHandler.cls
Created	AppointmentTriggerHandler	ApexClass	force-app\main\default\classes\AppointmentTriggerHandler.cls-meta.xml
Unchanged	CustomExceptions	ApexClass	force-app\main\default\classes\CustomExceptions.cls
Unchanged	CustomExceptions	ApexClass	force-app\main\default\classes\CustomExceptions.cls-meta.xml
Unchanged	ErrorHandler	ApexClass	force-app\main\default\classes\ErrorHandler.cls
Unchanged	ErrorHandler	ApexClass	force-app\main\default\classes\ErrorHandler.cls-meta.xml
Unchanged	PrescriptionTriggerHandler	ApexClass	force-app\main\default\classes\PrescriptionTriggerHandler.cls
Unchanged	PrescriptionTriggerHandler	ApexClass	force-app\main\default\classes\PrescriptionTriggerHandler.cls-meta.xml
Created	PurchaseOrderService	ApexClass	force-app\main\default\classes\PurchaseOrderService.cls

Have successfully Deployed all the Apex Classes !!

SALESFORCE PROJECT IMPLEMENTATION PHASES

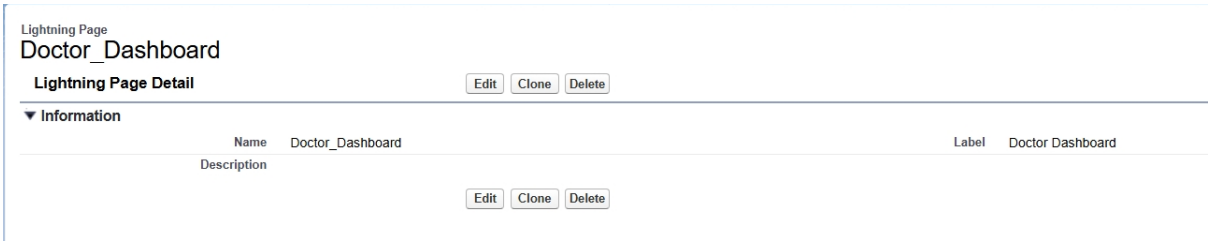
Phase 6: User Interface Development

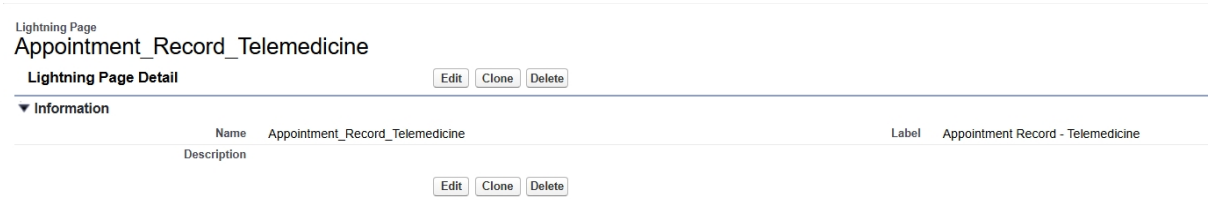
In this phase, I have build the **UI layer** of my Telemedicine Salesforce project. The goal is to design intuitive Lightning Pages and extend them with **Lightning Web Components (LWC)** connected to Apex for dynamic features.

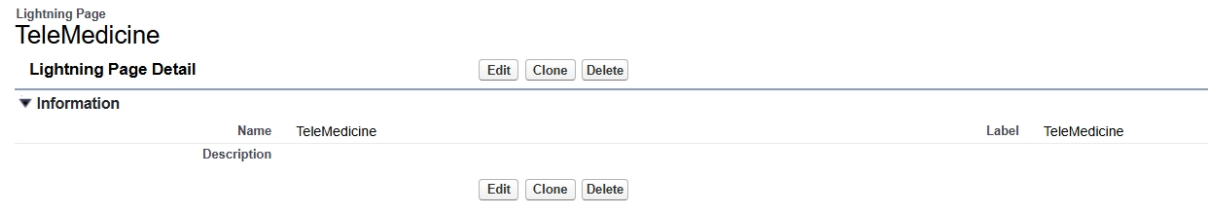
- **Lightning App Builder**

  Tool: *Setup → Lightning App Builder*.

  - Create custom pages for **Doctor Dashboard**, **Patient Portal**, and **Pharmacy Management**.
  - Add components (Record Details, Related Lists, Charts)
  - Assign to Apps, Record Types or Profiles



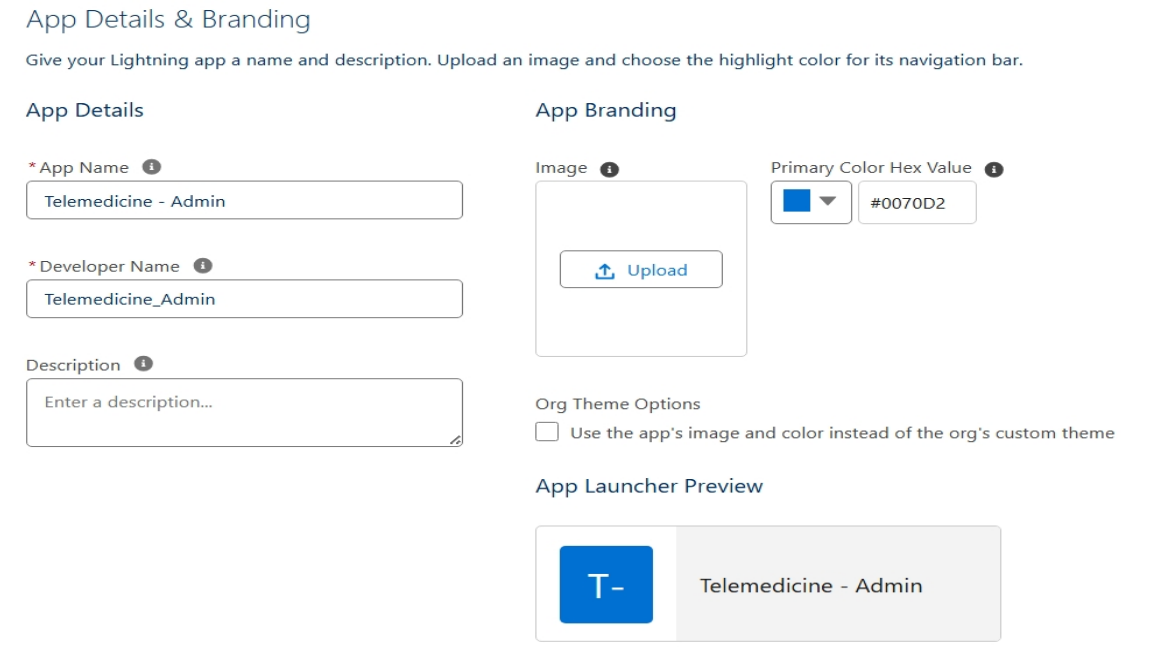


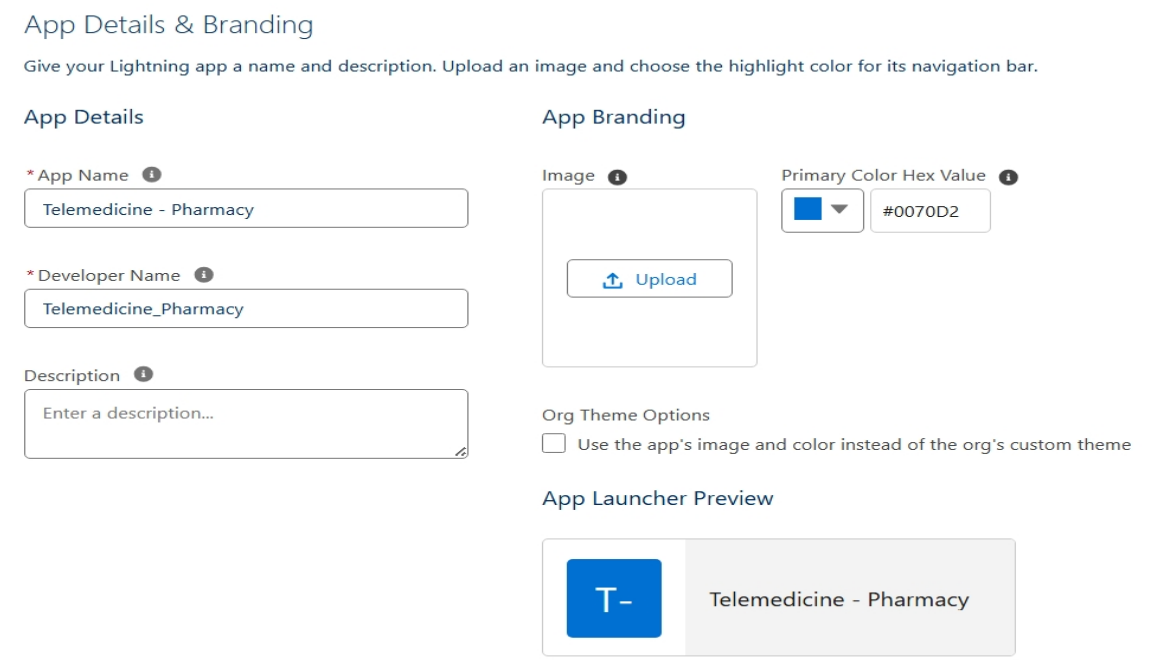


- **Record Pages**

  Customize **Appointment__c Record Page** to show:

  - Appointment Details (Time, Doctor, Patient).
  - Related List for Prescription__c.
  - Custom LWC for “Reschedule Appointment”.



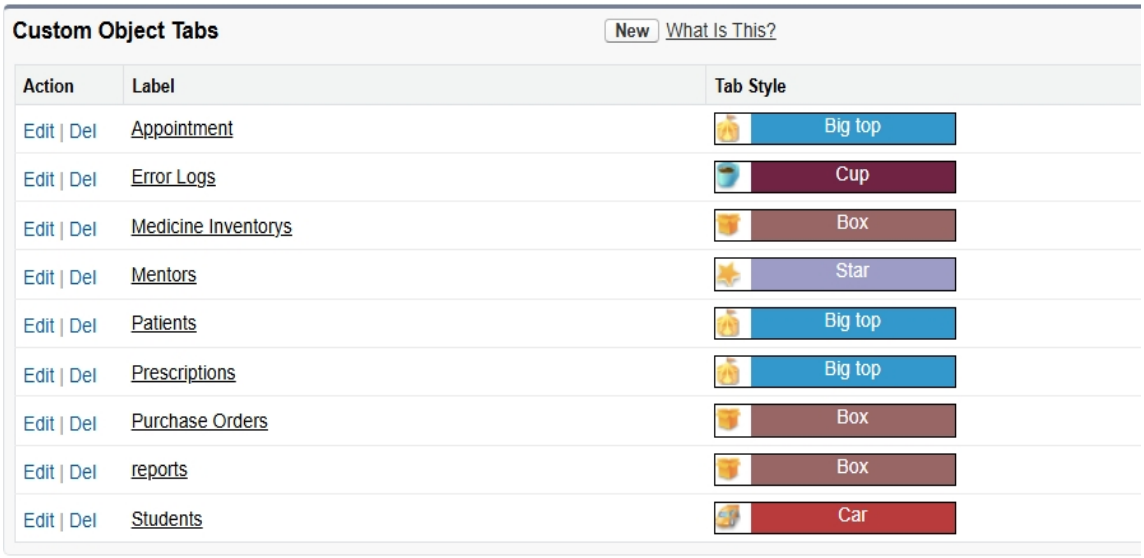


- **Tabs**

  Add **Custom Tabs** for:

  - Appointment__c
  - Prescription__c
  - Purchase_Order__c
  - report__c
  - Medicine_inventory__c
  - Error_log__c

- Patient__c



Custom Object Tabs New What Is This?

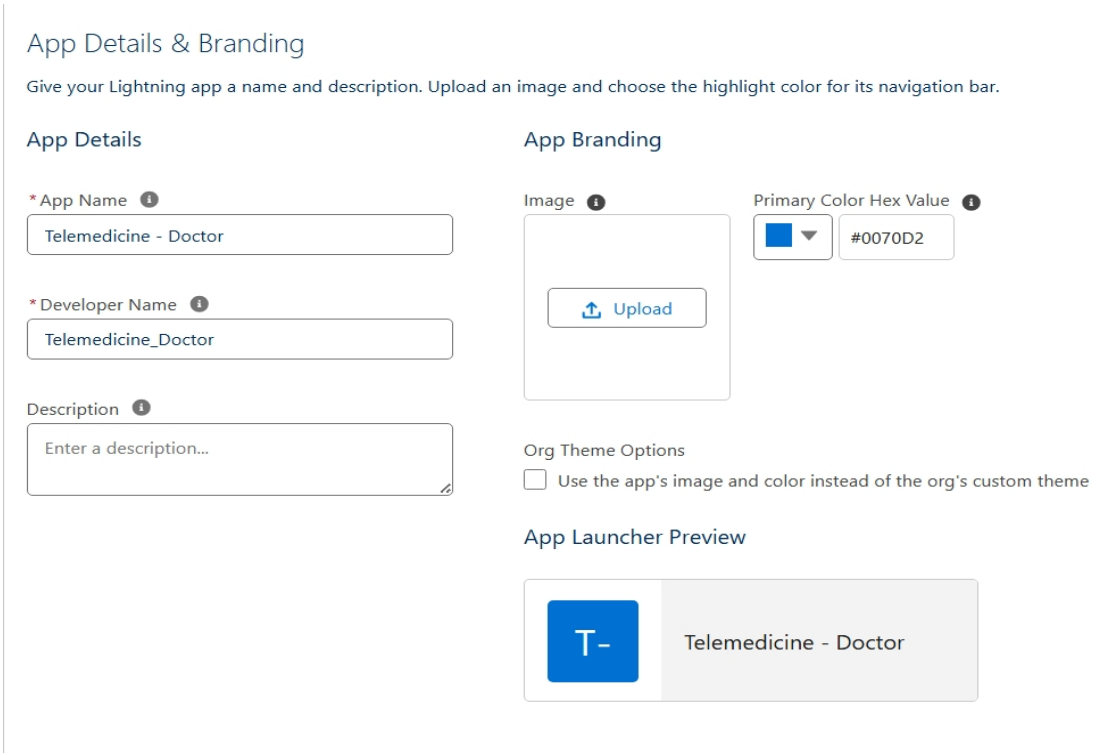
Action	Label	Tab Style
Edit | Del	Appointment	Big top
Edit | Del	Error Logs	Cup
Edit | Del	Medicine Inventorys	Box
Edit | Del	Mentors	Star
Edit | Del	Patients	Big top
Edit | Del	Prescriptions	Big top
Edit | Del	Purchase Orders	Box
Edit | Del	reports	Box
Edit | Del	Students	Car

Tabs help users access records easily from the App Launcher.

➢ Home Page Layouts

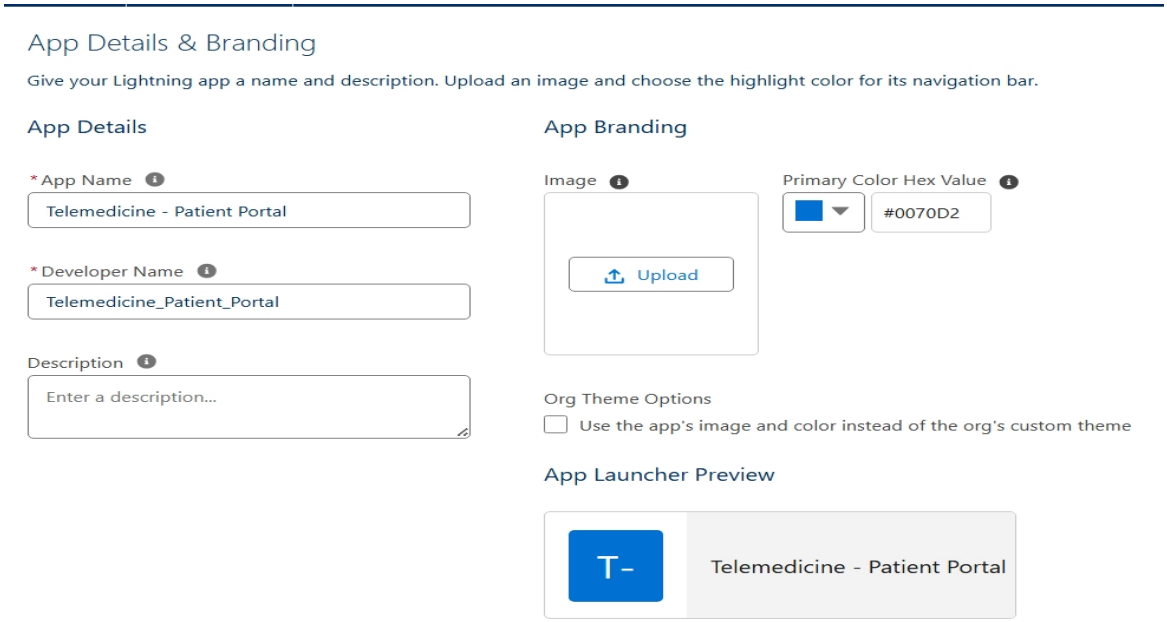
Modify Home Page for **Doctors**:

- Component: Today's Appointments (List View).
- Component: Recent Patients.
- Dashboard (Appointments by Status).



Modify Home Page for **Patients**:

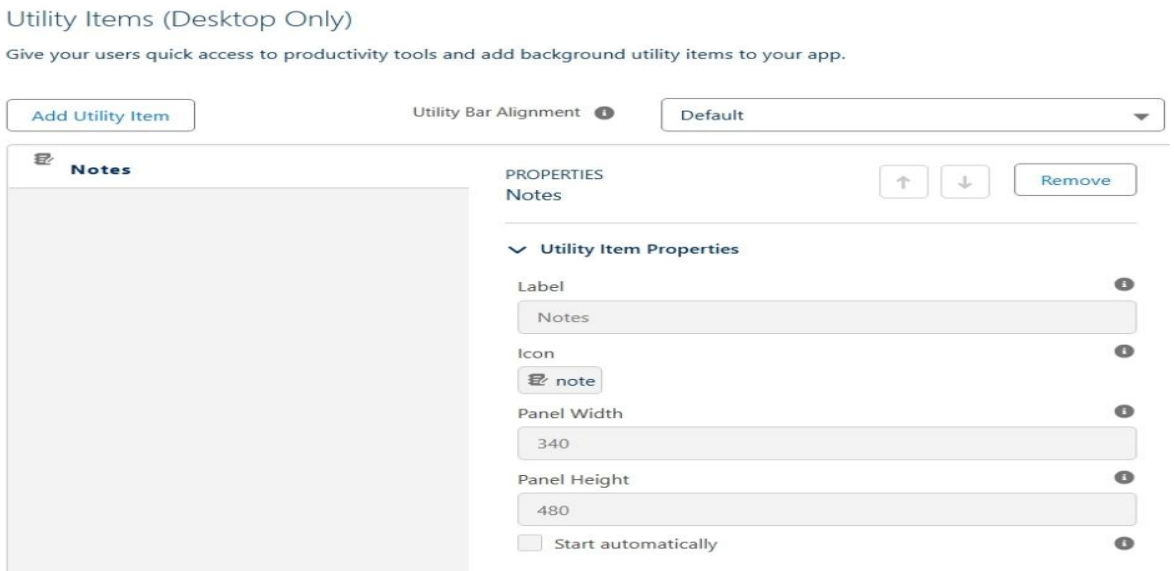
- Component: Upcoming Appointments.
- Component: My Prescriptions.



➢ Utility Bar

Add Utility Bar in the Doctor's App:

- Quick Actions : Create Appointment
- Chat (Future Integration)
- Notes



➢ LWC (Lightning Web Components)

Custom components built for our project:

- **AppointmentCard** → Display patient + doctor appointment details.

appointmentCard.html U

nabha > force-app > main > default > lwc > appointmentCard > appointmentCard.html > ...

```
<template>
    <lightning-card title="Appointment Summary" icon-name="standard:event">
        <div class="slds-p-horizontal_medium">
            <p><strong>Doctor:</strong> {doctorName}</p>
            <p><strong>Patient:</strong> {patientName}</p>
            <p><strong>Date:</strong> {appointmentDate}</p>
            <p><strong>Status:</strong> {status}</p>
        </div>
    </lightning-card>
</template>
```

appointmentCard.js U

nabha > force-app > main > default > lwc > appointmentCard > appointmentCard.js > ...

```
import { LightningElement, api } from 'lwc';

export default class AppointmentCard extends LightningElement {
    @api doctorName;
    @api patientName;
    @api appointmentDate;
    @api status;
}
```

appointmentCard.js-meta.xml U

nabha > force-app > main > default > lwc > appointmentCard > appointmentCard.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <masterLabel>AppointmentCard</masterLabel>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

- **PrescriptionForm** → Capture medicines and dosage.

prescriptionForm.html U

nabha > force-app > main > default > lwc > prescriptionForm > prescriptionForm.html > ...

```
<template>
    <lightning-card title="Create Prescription">
        <div class="slds-p-around_medium">
            <lightning-input name="medicine" label="Medicine Id" onchange={handleChange}></lightning-input>
            <lightning-input name="quantity" label="Quantity" type="number" onchange={handleChange}></lightning-input>
            <lightning-button variant="brand" label="Save" onclick={saveHandler}></lightning-button>
        </div>
    </lightning-card>
</template>
```

prescriptionForm.js U

nabha > force-app > main > default > lwc > prescriptionForm > prescriptionForm.js > ...

```
export default class PrescriptionForm extends LightningElement {

    saveHandler(){
        const pres = {
            sobjectType: 'Prescription__c',
            Appointment__c: this.appointmentId,
            Medicine__c: this.medicineId,
            Quantity__c: this.quantity
        };
        savePrescription({ p: pres })
            .then(id => {
                this.dispatchEvent(new CustomEvent('saved', { detail: { id } }));
                // optional: show toast (requires import)
            })
            .catch(err => {
                // error handling
                console.error(err);
            });
    }
}
```

prescriptionForm.js-meta.xml U

nabha > force-app > main > default > lwc > prescriptionForm > prescriptionForm.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

- **PurchaseOrderTracker** → Show Purchase Orders with Pharma Suppliers.

purchaseOrderTracker.html U

nabha > force-app > main > default > lwc > purchaseOrderTracker > purchaseOrderTracker.html > ...

```
<template>
    <lightning-card title="Purchase Order Tracker" icon-name="standard:orders">
        <div class="slds-p-horizontal_medium">
            <template if:true={orders}>
                <template for:each={orders} for:item="order">
                    <p key={order.Id}><strong>{order.Name}</strong> – {order.Status__c}</p>
                </template>
            </template>
            <template if:false={orders}>
                <p>No purchase orders found.</p>
            </template>
        </div>
    </lightning-card>
</template>
```

purchaseOrderTracker.js U

nabha > force-app > main > default > lwc > purchaseOrderTracker > purchaseOrderTracker.js > PurchaseOrderTracker > w

```
import { LightningElement, wire } from 'lwc';
import getPurchaseOrders from '@salesforce/apex/PurchaseOrderController.getPurchaseOrders';

export default class PurchaseOrderTracker extends LightningElement {
    orders;

    @wire(getPurchaseOrders)
    wiredOrders({ error, data }) {
        if (data) {
            this.orders = data;
        } else {
            this.orders = null;
        }
    }
}
```

purchaseOrderTracker.js-meta.xml U

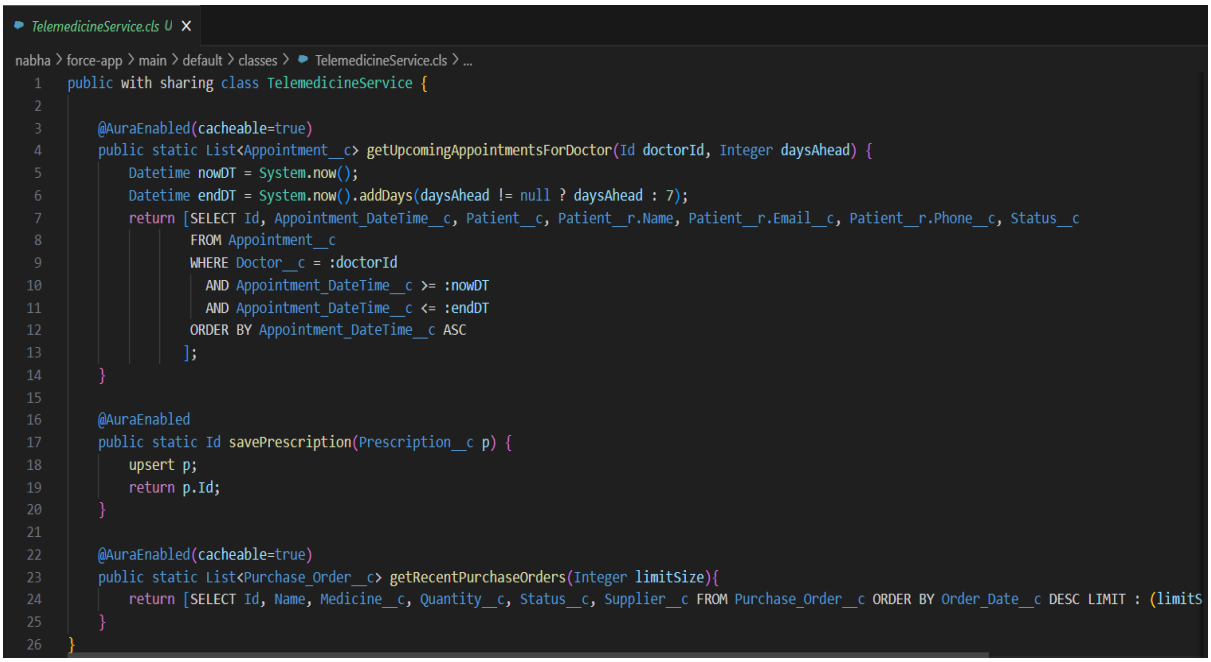
nabha > force-app > main > default > lwc > purchaseOrderTracker > purchaseOrderTracker.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
    </targets>
</LightningComponentBundle>
```

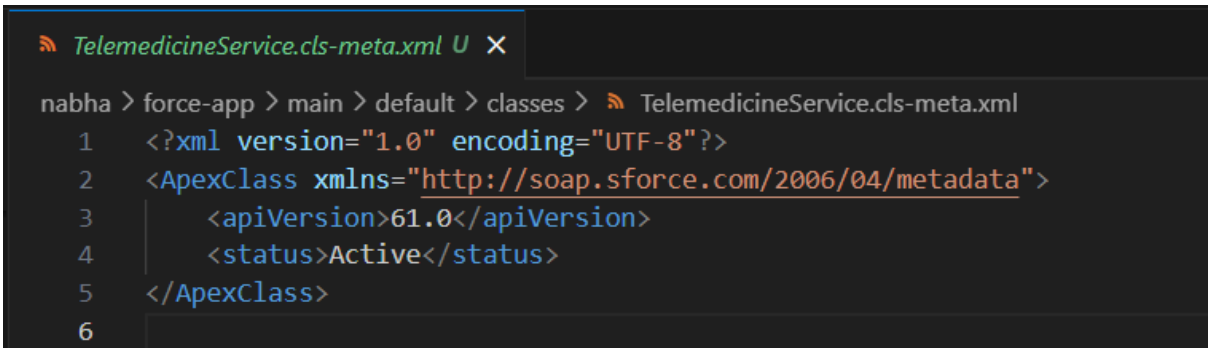
➢ Apex with LWC

Call Apex class TelemedicineService from LWC to:

- Fetch appointments for logged-in doctor.
- Save a new prescription.



```
TelemedicineService.cls U
nabha > force-app > main > default > classes > TelemedicineService.cls > ...
1  public with sharing class TelemedicineService {
2
3      @AuraEnabled(cacheable=true)
4      public static List<Appointment__c> getUpcomingAppointmentsForDoctor(Id doctorId, Integer daysAhead) {
5          Datetime nowDT = System.now();
6          Datetime endDT = System.now().addDays(daysAhead != null ? daysAhead : 7);
7          return [SELECT Id, Appointment_DateTime__c, Patient__c, Patient__r.Name, Patient__r.Email__c, Patient__r.Phone__c, Status__c
8                  FROM Appointment__c
9                  WHERE Doctor__c = :doctorId
10                   AND Appointment_DateTime__c >= :nowDT
11                   AND Appointment_DateTime__c <= :endDT
12                 ORDER BY Appointment_DateTime__c ASC
13                 ];
14     }
15
16     @AuraEnabled
17     public static Id savePrescription(Prescription__c p) {
18         upsert p;
19         return p.Id;
20     }
21
22     @AuraEnabled(cacheable=true)
23     public static List<Purchase_Order__c> getRecentPurchaseOrders(Integer limitSize){
24         return [SELECT Id, Name, Medicine__c, Quantity__c, Status__c, Supplier__c FROM Purchase_Order__c ORDER BY Order_Date__c DESC LIMIT : (limitS
25     }
26 }
```



```
TelemedicineService.cls-meta.xml U
nabha > force-app > main > default > classes > TelemedicineService.cls-meta.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ApexClass xmlns="http://soap.sforce.com/2006/04/metadata">
3      <apiVersion>61.0</apiVersion>
4      <status>Active</status>
5  </ApexClass>
6
```

➢ Events in LWC

Use events for communication:

- **Child → Parent**: PrescriptionForm sends “PrescriptionSaved” event to parent component.

Child dispatches CustomEvent('saved', { detail: { id } }) after saving. Parent component listens:

```
<c-prescription-form onSaved={handleSaved}></c-prescription-form>
```

- **Parent → Child**: Refresh AppointmentCard when appointment is rescheduled.

Parent handleSaved(event) can refresh list or call this.template.querySelector('c-appointment-card').loadAppointments().

- **Wire Adapters**
  Use @wire with Salesforce data:
  - @wire(getRecord, { recordId: '$recordId', fields: [...] }) for Appointment record.
  - @wire(getListUi, { objectApiName: 'Appointment__c', listViewApiName: 'All' }) to show appointments in UI.

- **Imperative Apex Calls**
  In AppointmentCard LWC:

```
import getUpcomingAppointments from '@salesforce/apex/TelemedicineService.getUpc
...
async loadAppointments() {
    this.appts = await getUpcomingAppointments();
}
```

- **Navigation Service**
  Use NavigationMixin in LWCs to redirect:

  - Navigate from AppointmentCard → Appointment Record Page.
  - Navigate from Patient → Related Prescriptions.

Deliverables in Phase 6

- Custom Record Pages for Appointment, Prescription, Patient.
- Custom Tabs and Home Pages.
- Utility Bar for Doctor App.
- LWC components (AppointmentCard, PrescriptionForm, PurchaseOrderTracker).
- Apex integrated with LWC (Wire + Imperative).
- Navigation between components.

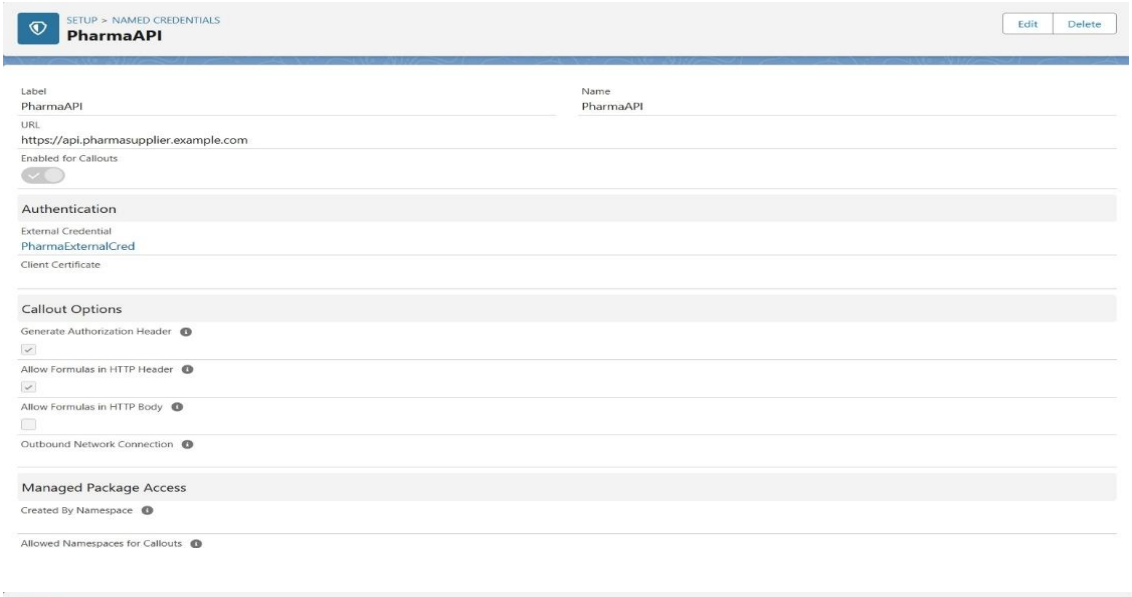
SALESFORCE PROJECT IMPLEMENTATION PHASES

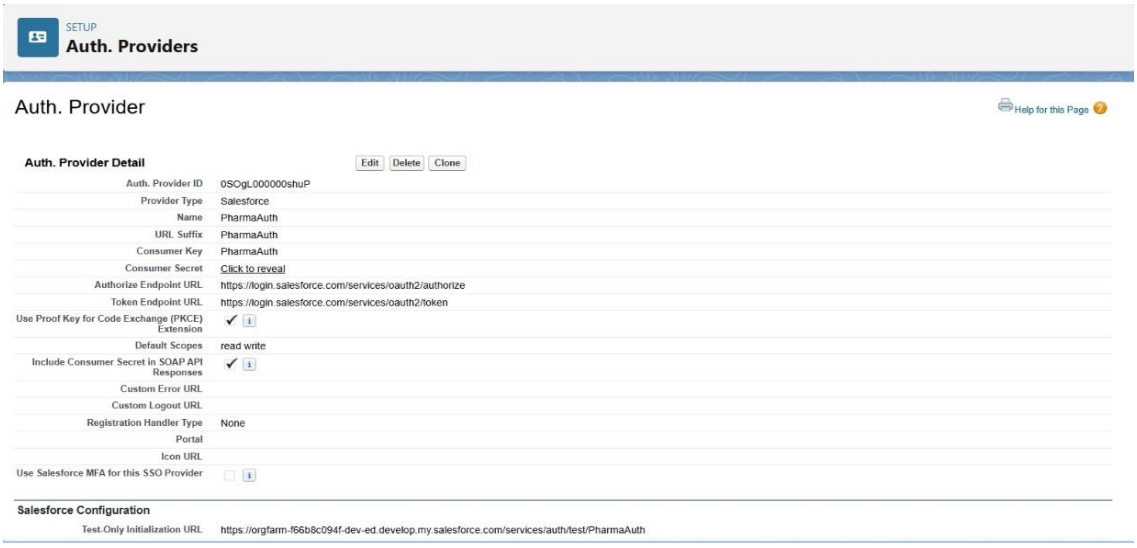
Phase 7: Integration & External Access

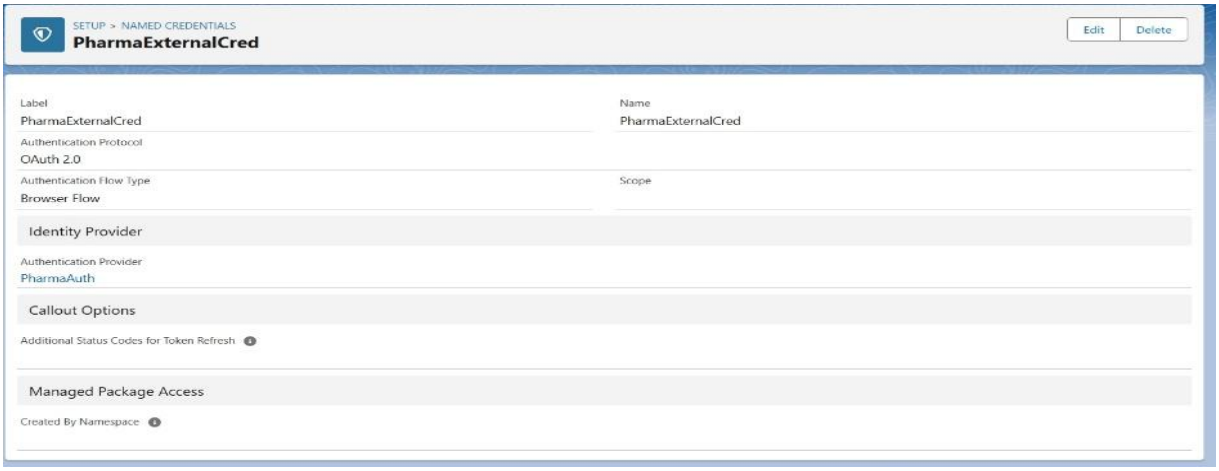
This phase focuses on connecting Salesforce with external systems and managing data exchange securely. The following components are implemented:

- **Named Credentials**
  A secure way to store authentication settings (endpoint URL, username, password, OAuth tokens).

  Use Case: Instead of hardcoding API credentials in Apex, use Named Credentials.



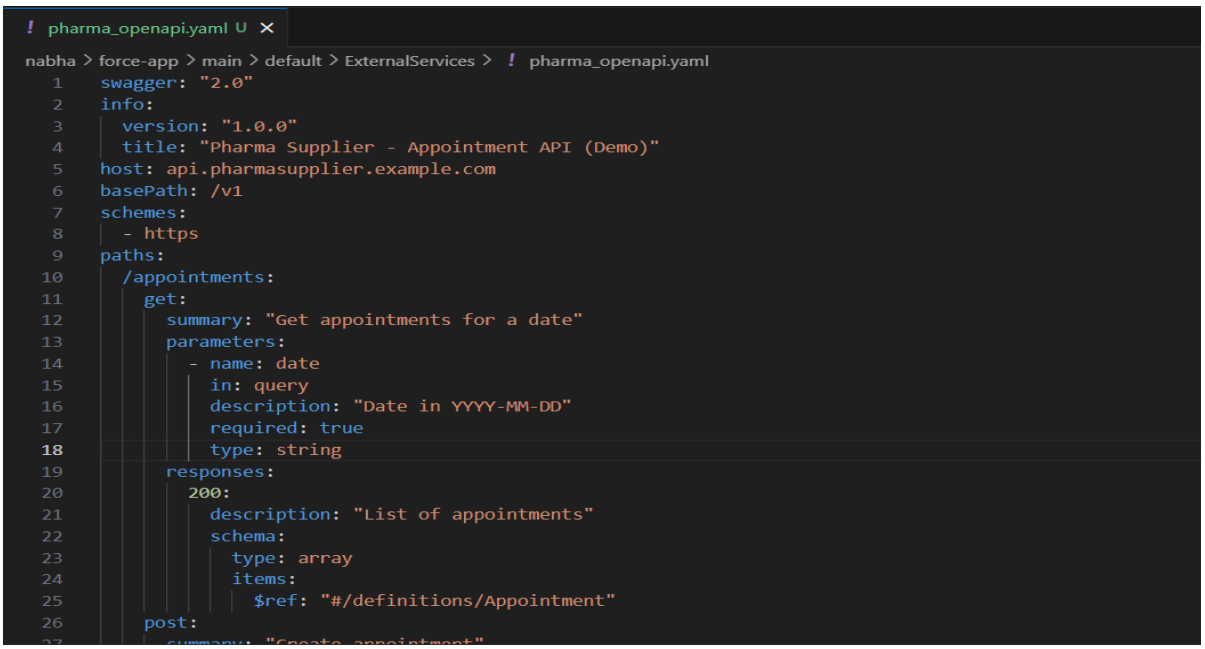


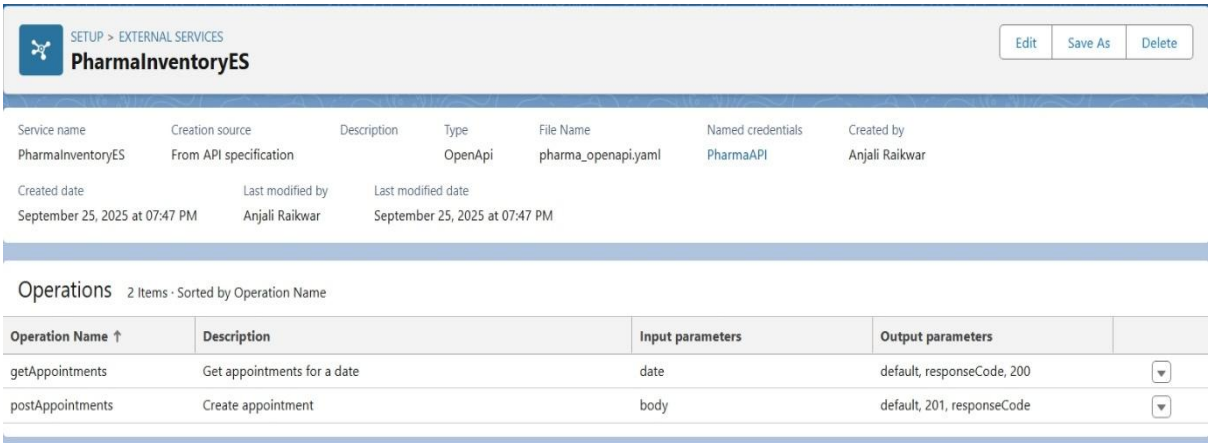


- **External Services**

  Declarative way to call REST APIs without writing Apex, using an OpenAPI (Swagger) schema.

  Use Case: Connect to hospital's Appointment API or demo Pet Store API.





TM_AppointmentAPI.cls U

nabha > force-app > main > default > classes > TM_AppointmentAPI.cls > ...

```
@RestResource(urlMapping='/telemedicine/appointments/*')
global with sharing class TM_AppointmentAPI {

    @HttpGet
    global static Appointment__c getAppointment() {
        RestRequest req = RestContext.request;
        String id = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        return [
            SELECT Id, Appointment_DateTime__c, Patient__c
            FROM Appointment__c
            WHERE Id = :id
            LIMIT 1
        ];
    }
}
```

- **Web Services (REST/SOAP)**

  Salesforce can expose its logic/data as web services or consume external ones.

  Exposing:

  REST → Use @RestResource annotation in Apex.

  SOAP → Use global class with webService keyword.

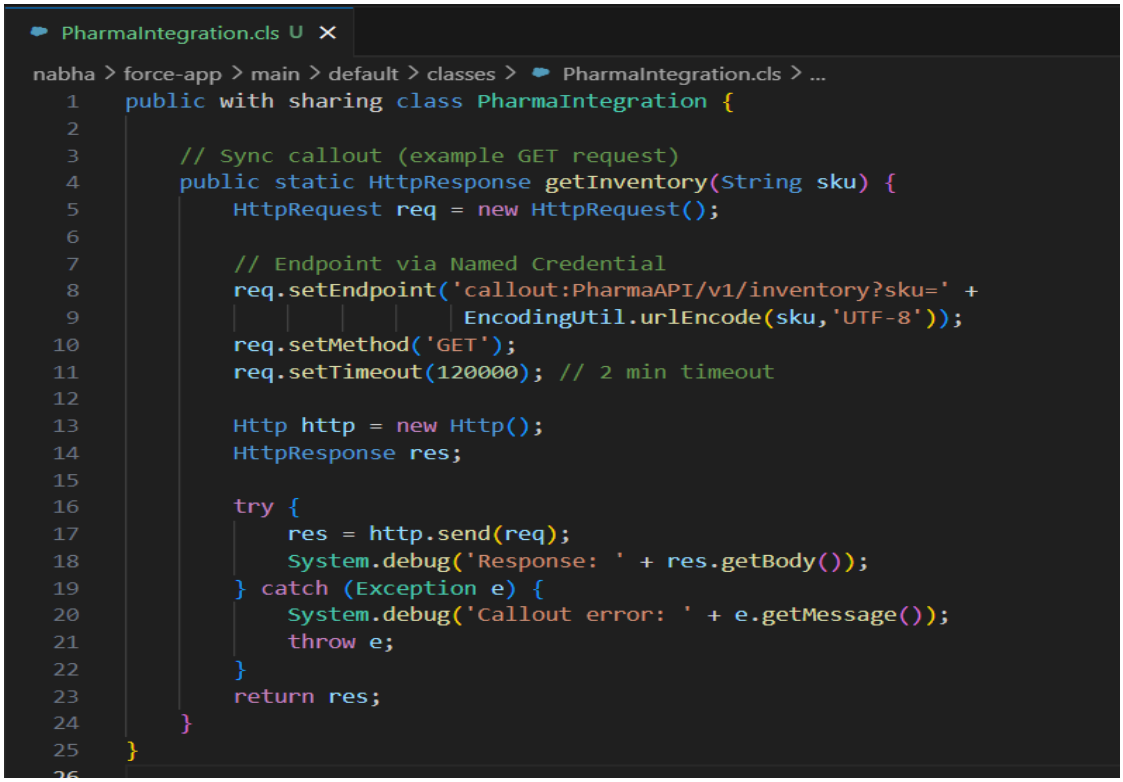
TM_AppointmentAPI.cls U

nabha > force-app > main > default > classes > TM_AppointmentAPI.cls > ...

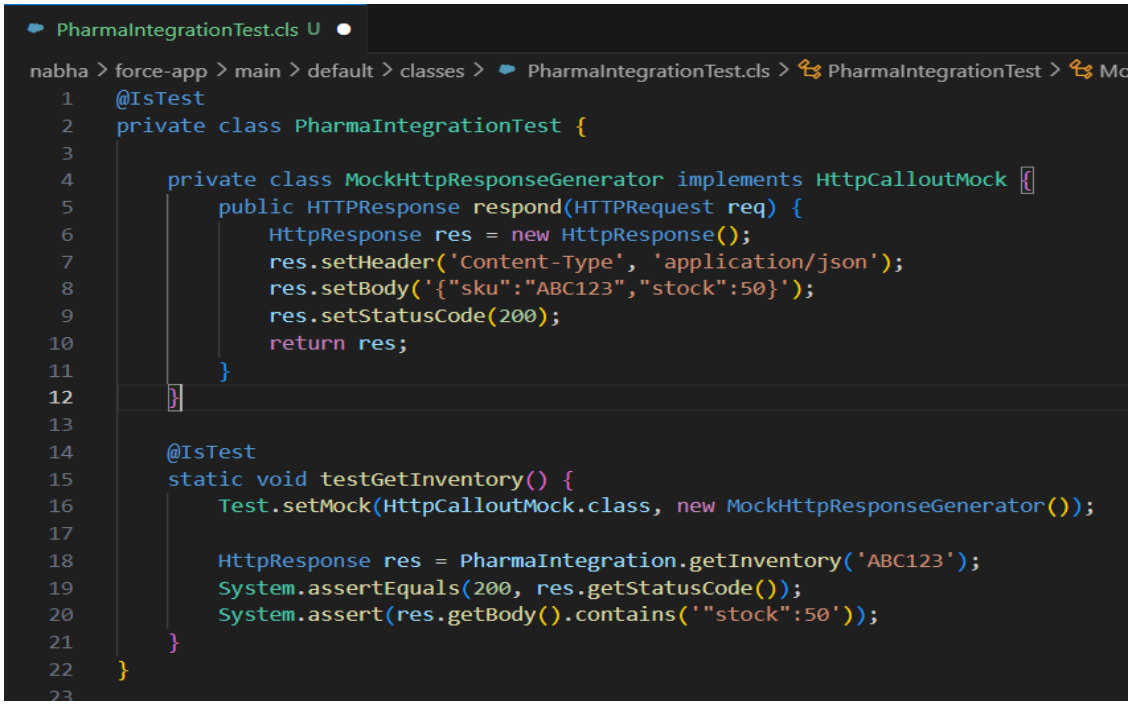
```
@RestResource(urlMapping='/telemedicine/appointments/*')
global with sharing class TM_AppointmentAPI {

    @HttpGet
    global static Appointment__c getAppointment() {
        RestRequest req = RestContext.request;
        String id = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        return [
            SELECT Id, Appointment_DateTime__c, Patient__c
            FROM Appointment__c
            WHERE Id = :id
            LIMIT 1
        ];
    }
}
```

- **Callouts**

  Outbound HTTP requests from Salesforce to external services.



```
PharmaIntegration.cls U
nabha > force-app > main > default > classes > PharmaIntegration.cls > ...
1   public with sharing class PharmaIntegration {
2
3       // Sync callout (example GET request)
4       public static HttpResponse getInventory(String sku) {
5           HttpRequest req = new HttpRequest();
6
7           // Endpoint via Named Credential
8           req.setEndpoint('callout:PharmaAPI/v1/inventory?sku=' +
9                           EncodingUtil.urlEncode(sku,'UTF-8'));
10          req.setMethod('GET');
11          req.setTimeout(120000); // 2 min timeout
12
13          Http http = new Http();
14          HttpResponse res;
15
16          try {
17              res = http.send(req);
18              System.debug('Response: ' + res.getBody());
19          } catch (Exception e) {
20              System.debug('Callout error: ' + e.getMessage());
21              throw e;
22          }
23          return res;
24      }
25  }
```

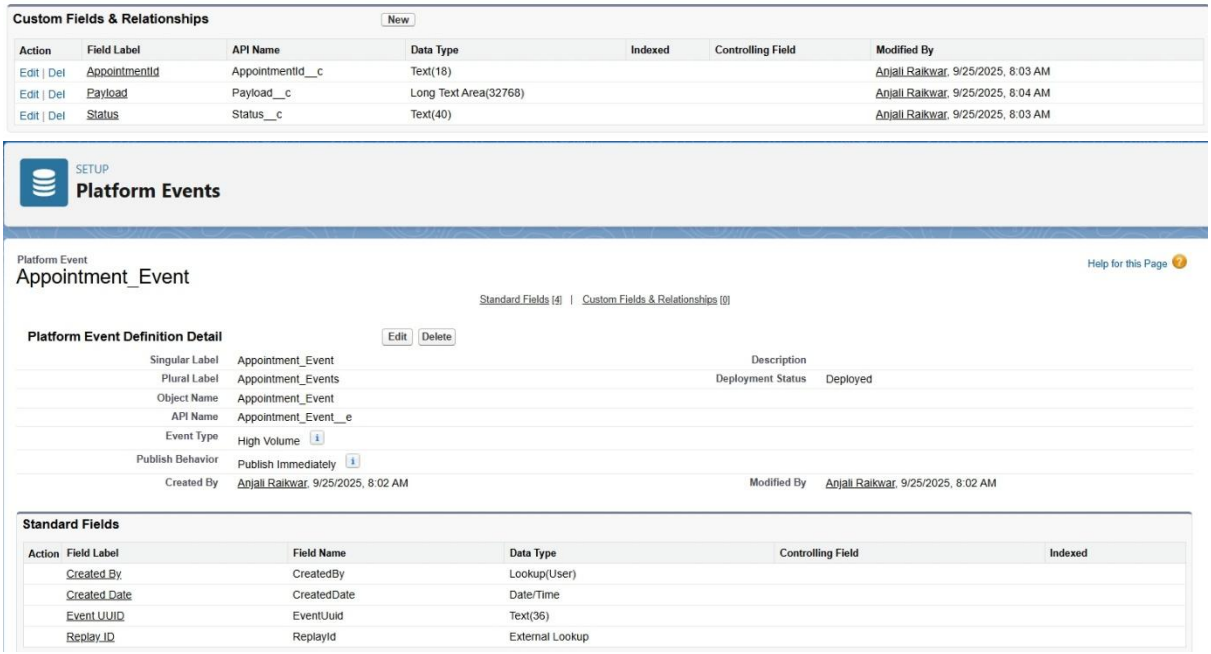


```
PharmaIntegrationTest.cls U
nabha > force-app > main > default > classes > PharmaIntegrationTest.cls > PharmaIntegrationTest > Mo
1   @IsTest
2   private class PharmaIntegrationTest {
3
4       private class MockHttpResponseGenerator implements HttpCalloutMock {
5           public HTTPResponse respond(HTTPRequest req) {
6               HttpResponse res = new HttpResponse();
7               res.setHeader('Content-Type', 'application/json');
8               res.setBody('{"sku":"ABC123","stock":50}');
9               res.setStatusCode(200);
10              return res;
11          }
12      }
13
14      @IsTest
15      static void testGetInventory() {
16          Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());
17
18          HttpResponse res = PharmaIntegration.getInventory('ABC123');
19          System.assertEquals(200, res.getStatusCode());
20          System.assert(res.getBody().contains('"stock":50'));
21      }
22  }
```

- **Platform Events**

  Event-driven architecture in Salesforce (publish-subscribe model).

  Use Case: Notify external systems when an appointment is booked or cancelled.



Custom Fields & Relationships — New

Action	Field Label	API Name	Data Type	Indexed	Controlling Field	Modified By
Edit | Del	AppointmentId	AppointmentId__c	Text(18)			Anjali Raikwar, 9/25/2025, 8:03 AM
Edit | Del	Payload	Payload__c	Long Text Area(32768)			Anjali Raikwar, 9/25/2025, 8:04 AM
Edit | Del	Status	Status__c	Text(40)			Anjali Raikwar, 9/25/2025, 8:03 AM

SETUP
Platform Events

Platform Event
Appointment_Event

Help for this Page

Standard Fields [4] | Custom Fields & Relationships [0]

Platform Event Definition Detail — Edit | Delete

Singular Label	Appointment_Event	Description	
Plural Label	Appointment_Events	Deployment Status	Deployed
Object Name	Appointment_Event		
API Name	Appointment_Event__e		
Event Type	High Volume		
Publish Behavior	Publish Immediately		
Created By	Anjali Raikwar, 9/25/2025, 8:02 AM	Modified By	Anjali Raikwar, 9/25/2025, 8:02 AM

Standard Fields

Action	Field Label	Field Name	Data Type	Controlling Field	Indexed
	Created By	CreatedBy	Lookup(User)		
	Created Date	CreatedDate	Date/Time		
	Event UUID	EventUuid	Text(36)		
	Replay ID	ReplayId	External Lookup		

AppointmentEventPublisher.cls U

nabha > force-app > main > default > classes > AppointmentEventPublisher.cls > ...

```
1  public with sharing class AppointmentEventPublisher {
2      public static void publishAppointmentEvent(Id apptId, String status) {
3          try {
4              Appointment_Event__e evt = new Appointment_Event__e(
5                  AppointmentId__c = apptId,
6                  Status__c = status,
7                  Payload__c = 'Appointment status changed to ' + status
8              );
9              Database.SaveResult sr = EventBus.publish(evt);
10             System.debug('Event published? ' + sr.isSuccess());
11         } catch (Exception e) {
12             System.debug('Error publishing event: ' + e.getMessage());
13         }
14     }
15 }
16
```

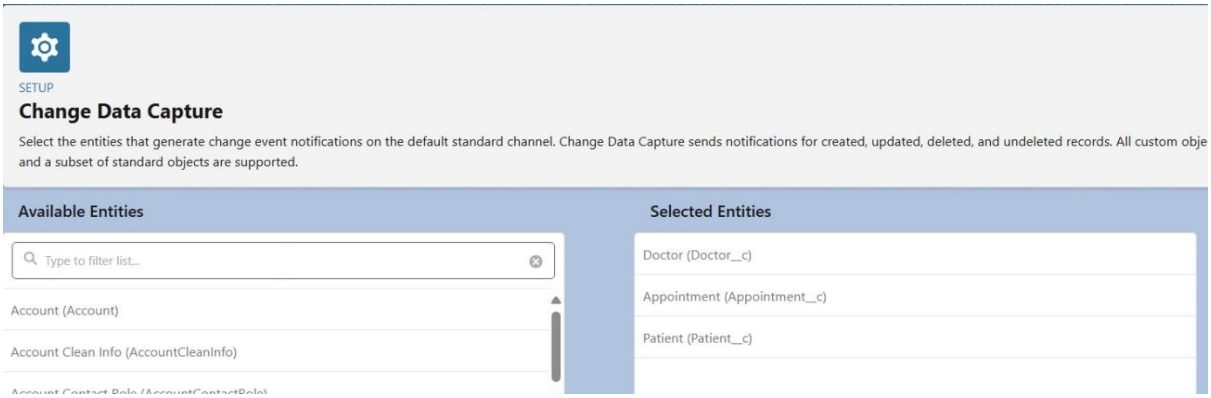
AppointmentEventTrigger.trigger U

nabha > force-app > main > default > triggers > AppointmentEventTrigger.trigger > AppointmentEventTrigger

```
trigger AppointmentEventTrigger on Appointment_Event__e (after insert) {
    for (Appointment_Event__e evt : Trigger.new) {
        System.debug('Received Appointment Event: ' + evt.AppointmentId__c +
                     ' Status: ' + evt.Status__c);
        // Example: Log or update something
        // Could update a related object or push to another queue
    }
}
```

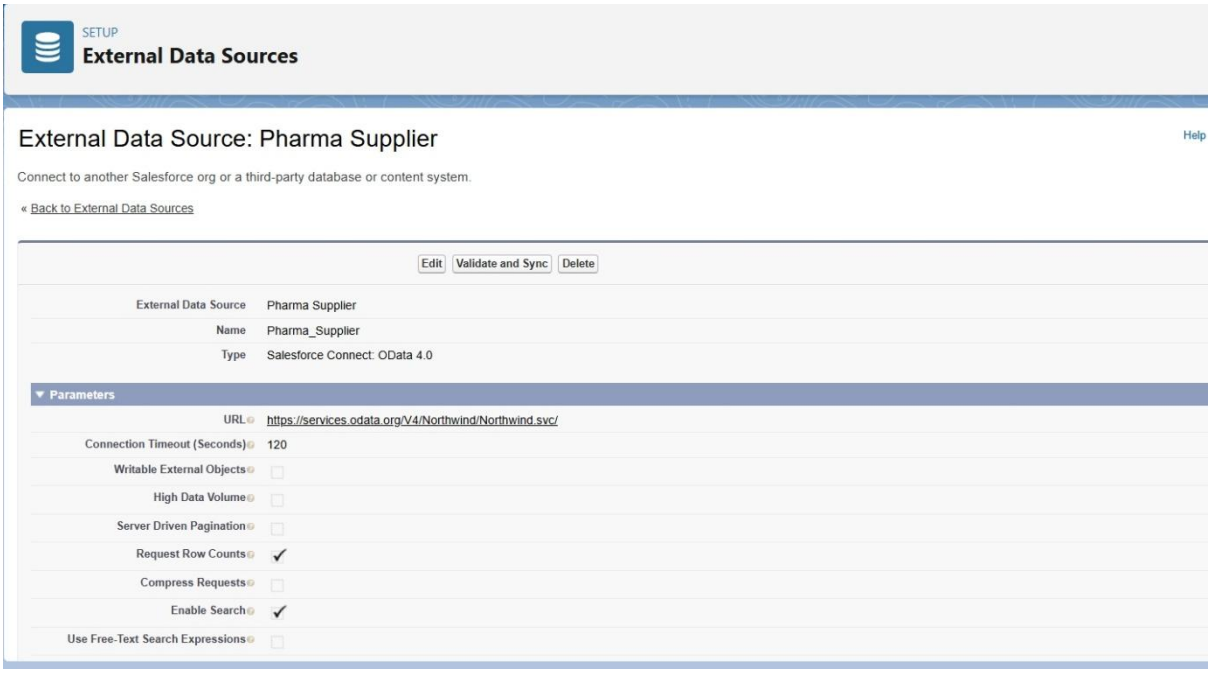
- **Change Data Capture**

  Automatically streams changes (Create, Update, Delete, Undelete) of Salesforce records.

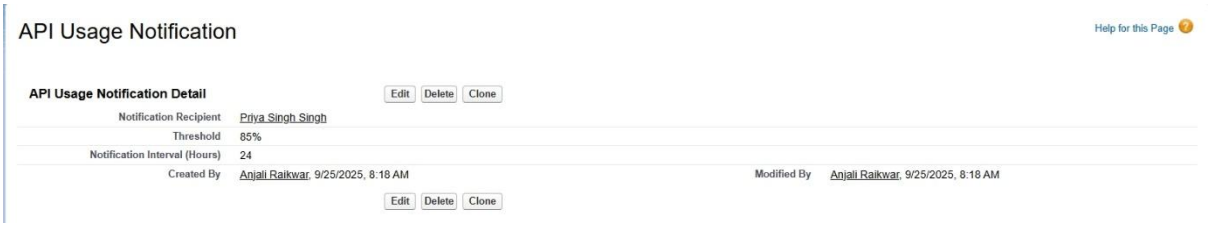


- **Salesforce Connect**

  Lets Salesforce access external data in real time without storing it.

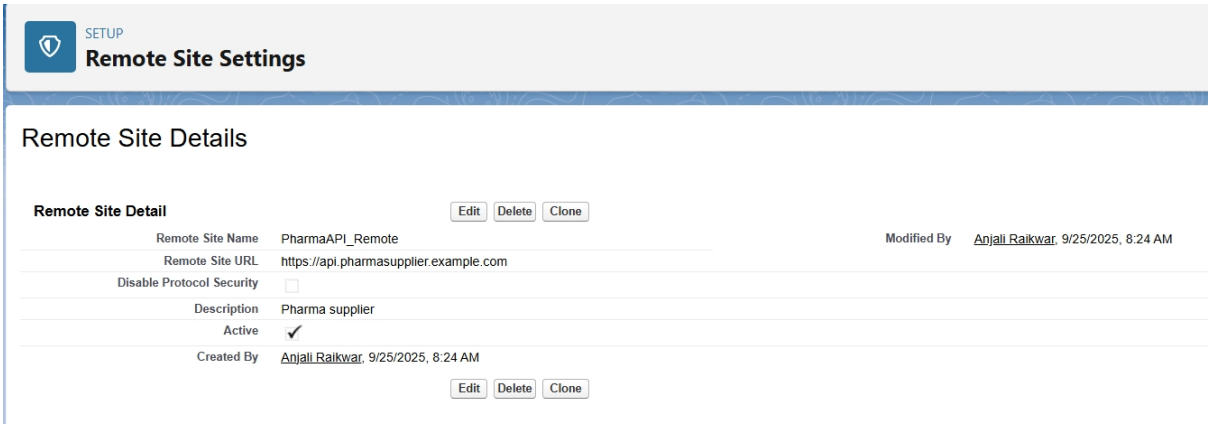


- **API Limits**
  Salesforce enforces API limits per 24 hours (depends on edition & licenses).



- **OAuth & Authentication**
  Secure authentication mechanism for APIs.

- **Remote Site Settings**
  Whitelist external domains before Salesforce can make callouts.



Deliverables in Phase 7

- Named Credentials → Store endpoint + auth.
- External Services → Declarative API calls.
- Web Services → Expose/consume SOAP & REST.
- Callouts → Programmatic API calls.
- Platform Events & CDC → Event-driven data sync.
- Salesforce Connect → Real-time external data access.
- API Limits → Monitor & optimize API usage.
- OAuth & Auth → Secure integrations.
- Remote Site Settings → Whitelist domains.

SALESFORCE PROJECT IMPLEMENTATION PHASES

Phase 8: Data Management & Deployment

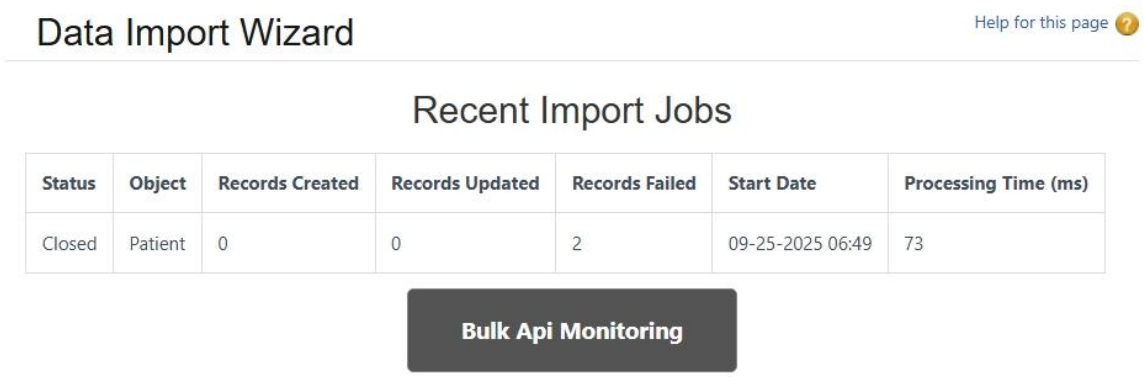
Managing Salesforce data and deploying configurations across environments is crucial for maintaining data integrity and smooth release cycles.

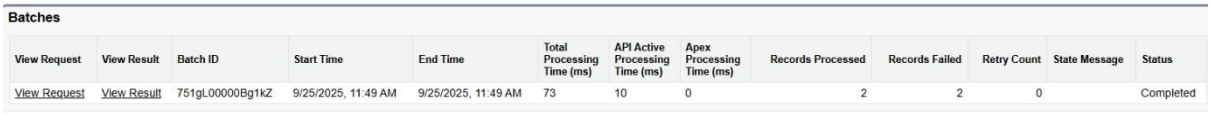
- **Data Import Wizard**

  The Data Import Wizard is a simple, browser-based tool that allows admins to import data into Salesforce without writing code. It is best suited for smaller datasets (up to 50,000 records).

  Key Features

  - Supports Standard Objects (Leads, Accounts, Contacts, Solutions, Campaign Members) and Custom Objects.
  - Provides an intuitive step-by-step interface.
  - Allows mapping of CSV fields to Salesforce fields.





FirstName	LastName	Email	Phone	DOB__c
Anjali	Raikwar	anjali@test.com	9876543210	1999-05-02
Rohan	Sharma	rohan@test.com	9876500011	1998-07-10

- **Data Loader**

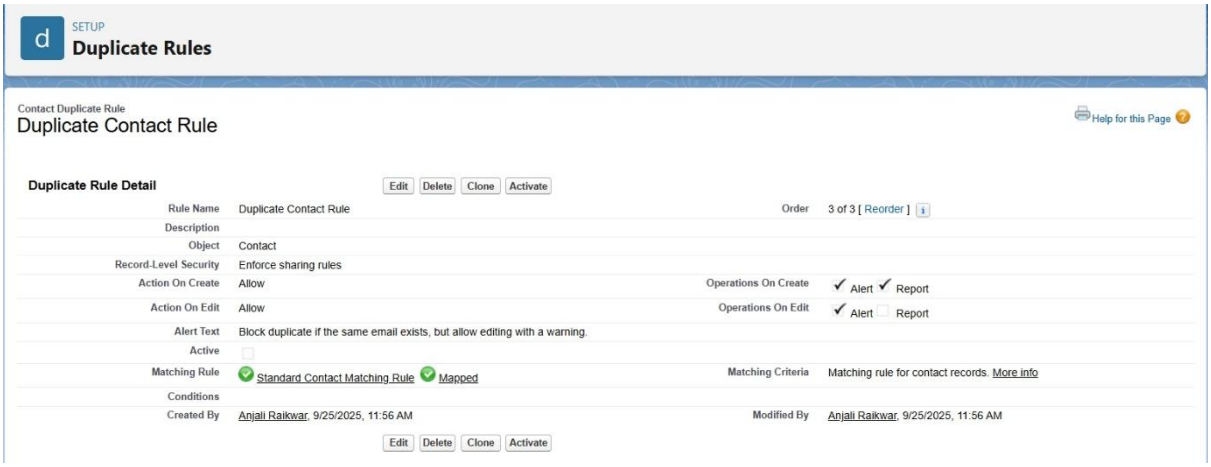
  The Salesforce Data Loader is a client application used for bulk data operations (up to 5 million records). It provides more control and is ideal for larger datasets.

  Key Features

- Supports Insert, Update, Upsert, Delete, Export.
- Works with CSV files.
- Can be scheduled via CLI for automation.

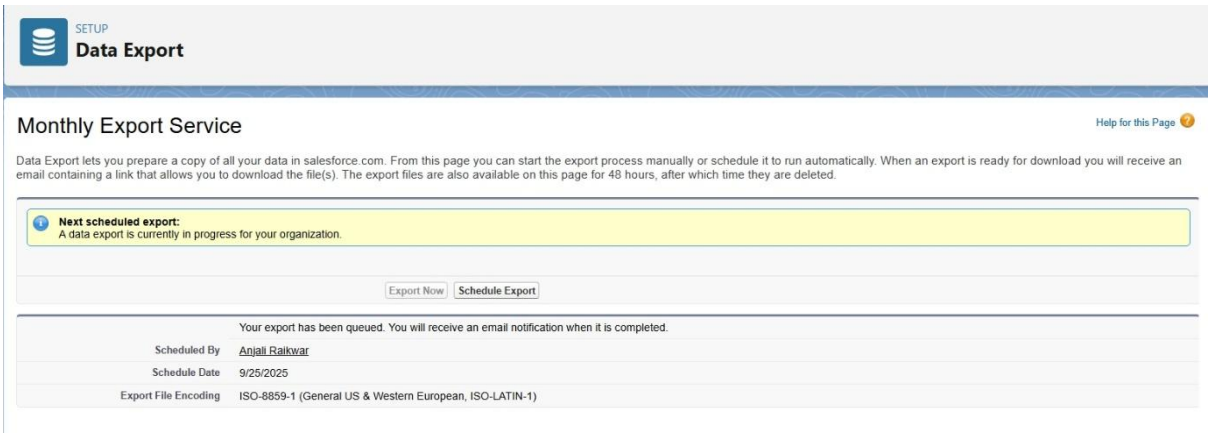
➢ **Duplicate Rules**

Duplicate Rules prevent creation of duplicate records, thereby ensuring data quality. These rules can block duplicates or alert users when similar records already exist.



➢ **Data Export & Backup**

Salesforce allows exporting full datasets for backup or compliance. Exports can be scheduled weekly or monthly.



➢ **Change Sets**

Change Sets are Salesforce's point-and-click tool for moving metadata (custom fields, objects, Apex, Flows, etc.) between environments like Sandbox → Production.

➢ **Unmanaged vs Managed Packages**

Packages are used to group components for distribution.

- Unmanaged Package:
  - Components are editable after installation.
  - Ideal for one-time deployments or sharing code.

- Managed Package:
    - Code is locked after installation.
    - Supports versioning and upgrade paths.
    - Required for AppExchange distribution.

Change Sets: Used to move metadata between related Salesforce orgs (Sandbox → Production).
Not available in Developer Edition (since it has no Sandbox). For this project, we demonstrate deployment using **VS Code (SFDX)** instead.

- **ANT Migration Tool**
  The ANT Migration Tool is a Java/Apache ANT-based tool that enables scripted deployments between Salesforce orgs. It is powerful for CI/CD automation.

  - build.properties

```
sf.username=your_username@example.com
sf.password=your_password+securityToken
sf.serverurl=https://login.salesforce.com
```

  - package.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>ApexClass</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <version>58.0</version>
</Package>
```

- build.xml

```
<project name="Sample" default="deployCode">
    <taskdef resource="com/salesforce/antlib.xml"
             classpath="ant-salesforce.jar"/>

    <target name="retrieveCode">
        <sf:retrieve username="${sf.username}" password="${sf.password}" serverurl="${sf.serverurl
    </target>

    <target name="deployCode">
        <sf:deploy username="${sf.username}" password="${sf.password}" serverurl="${sf.serverurl}"
    </target>
</project>
```

- retrieveCode → pulls metadata from org.
- deployCode → pushes metadata to org.

- **VS Code & SFDX (Salesforce DX)**

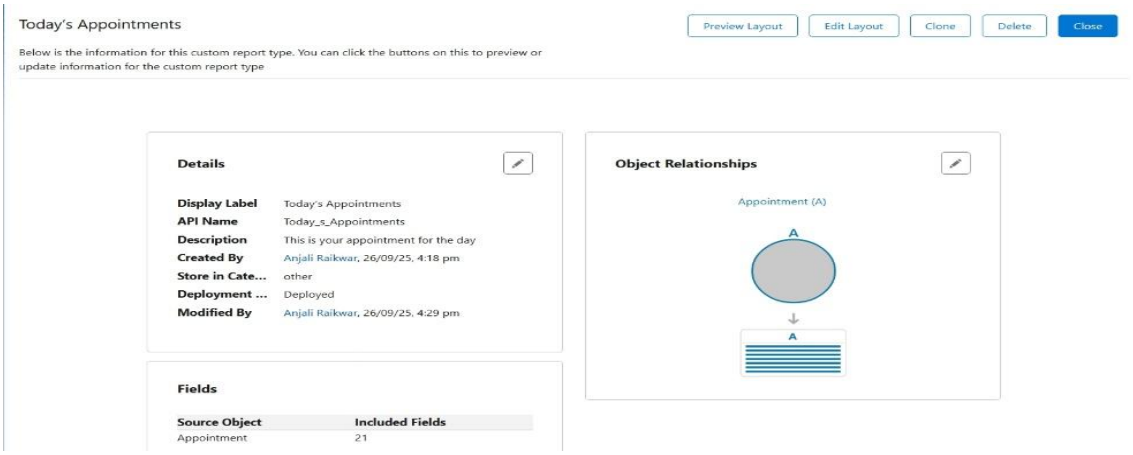
  Salesforce DX (SFDX) with Visual Studio Code is the modern way to develop and deploy Salesforce applications. It enables source-driven development, version control, scratch orgs, and CI/CD.

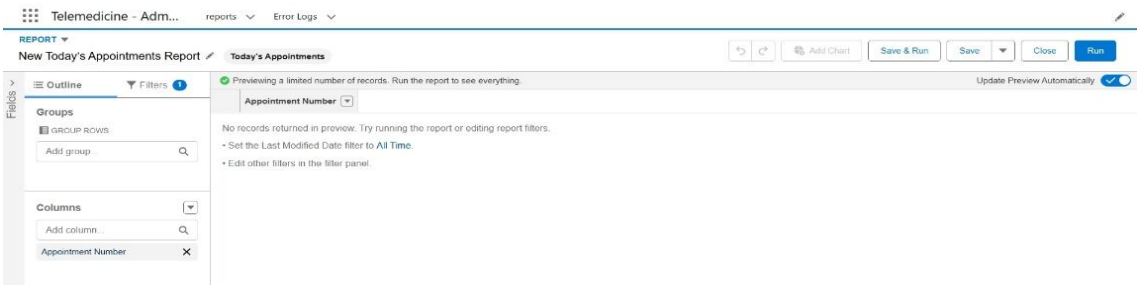
SALESFORCE PROJECT IMPLEMENTATION PHASES

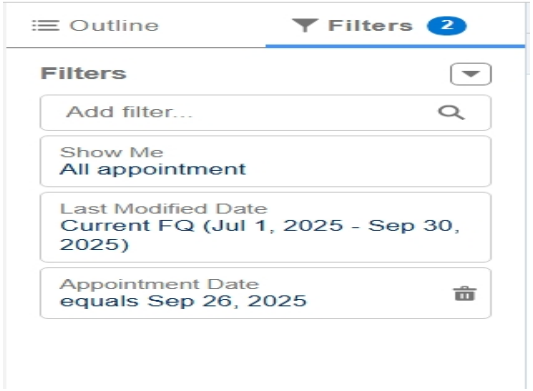
Phase 9: Reporting, Dashboards & Security Review

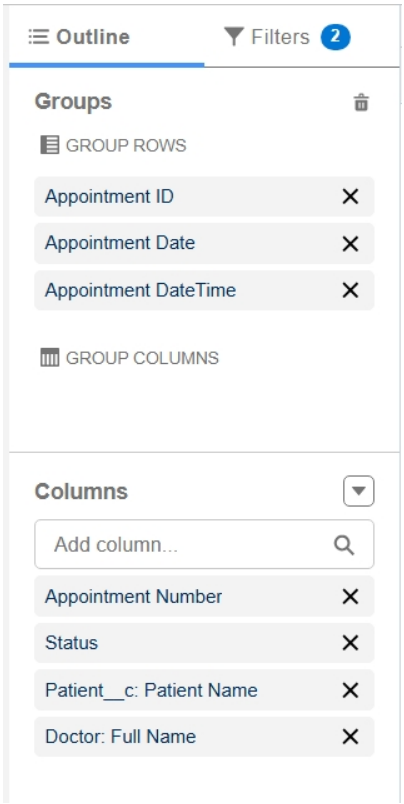
This phase ensures that our **Telemedicine CRM** delivers actionable insights through reports and dashboards, while also maintaining enterprise-grade security through Salesforce's built-in controls.

- **Reports (Tabular, Summary, Matrix, Joined**
  Reports are used to analyze data such as patient appointments, prescriptions, or doctor workload.
  Types
  - **Tabular** → Simple list of records (e.g., all appointments today).

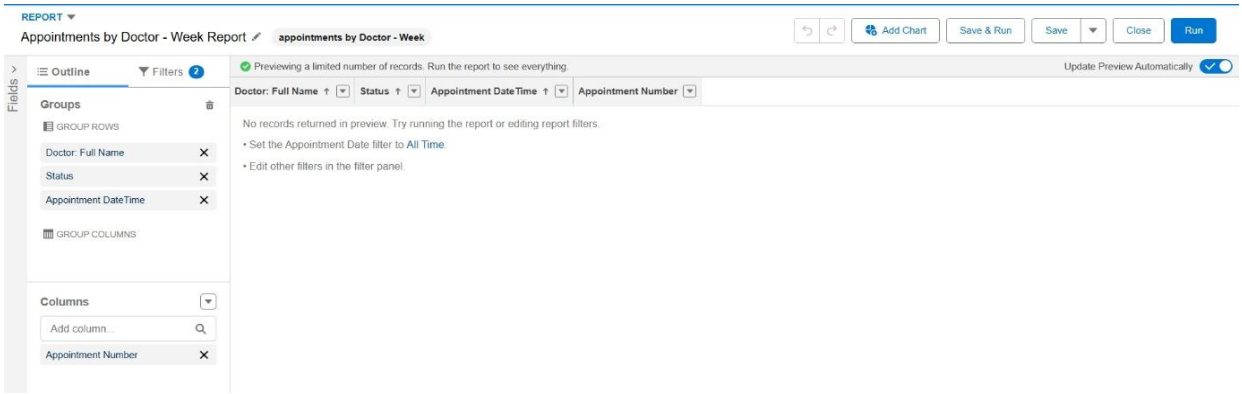


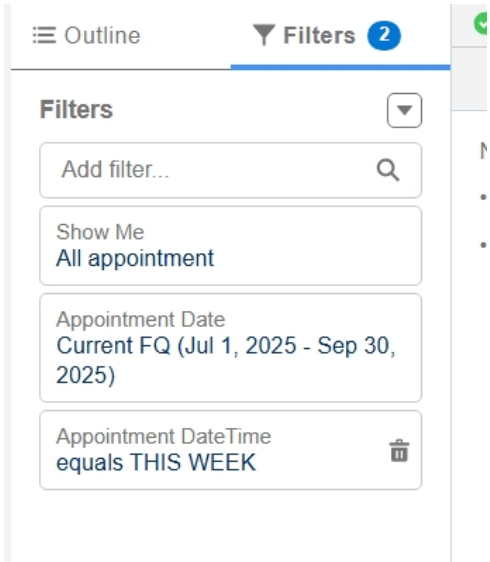


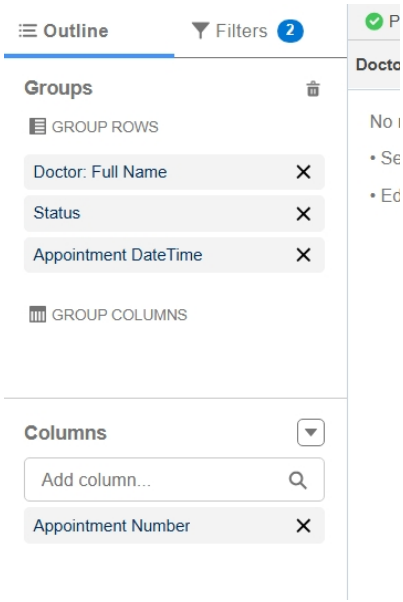


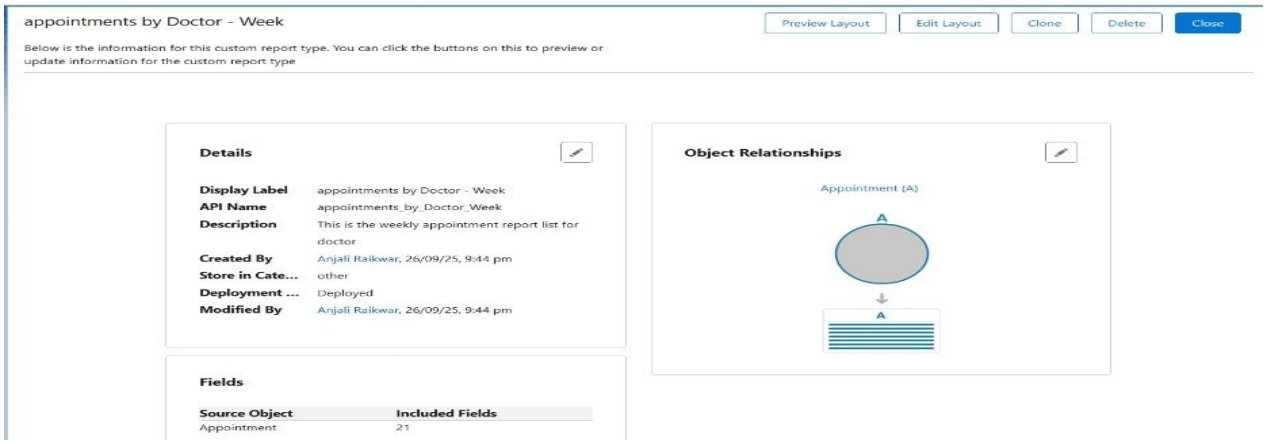


- **Summary** → Grouped by a field (e.g., appointments grouped by doctor).

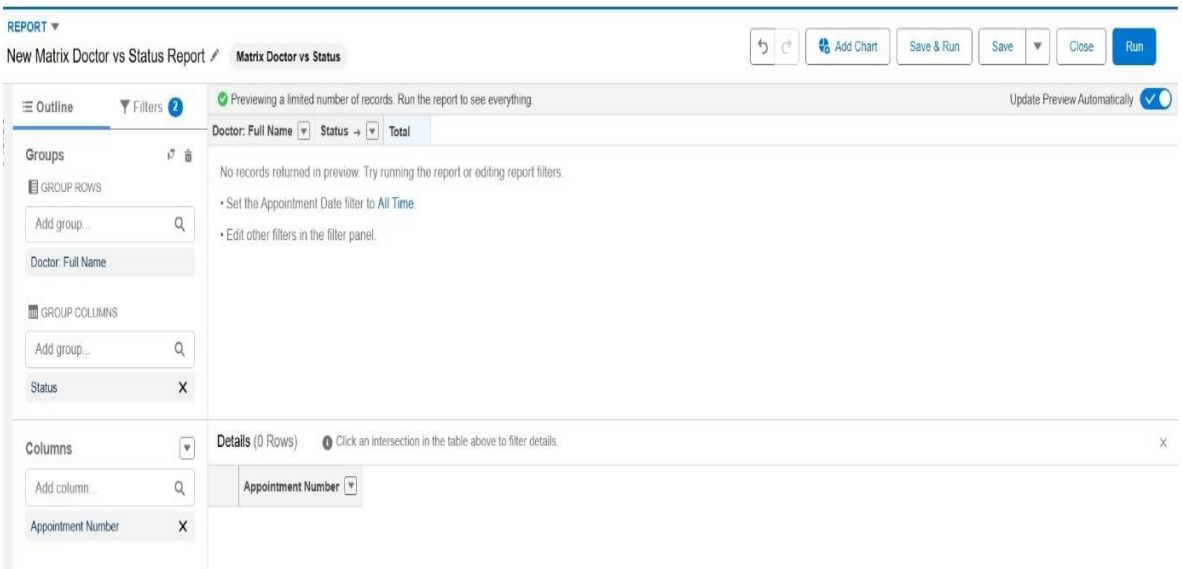


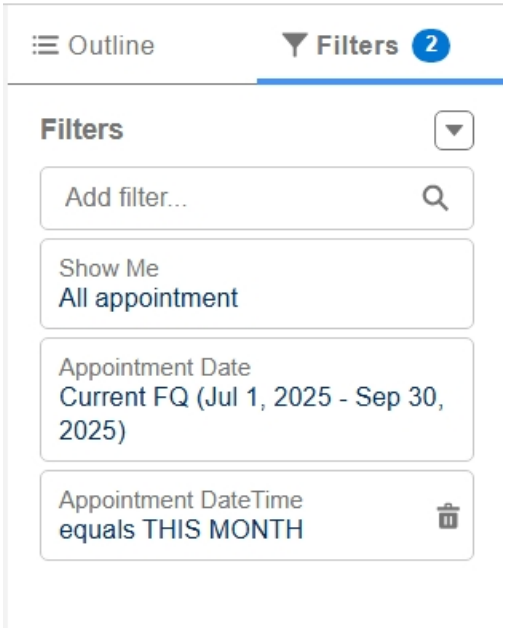


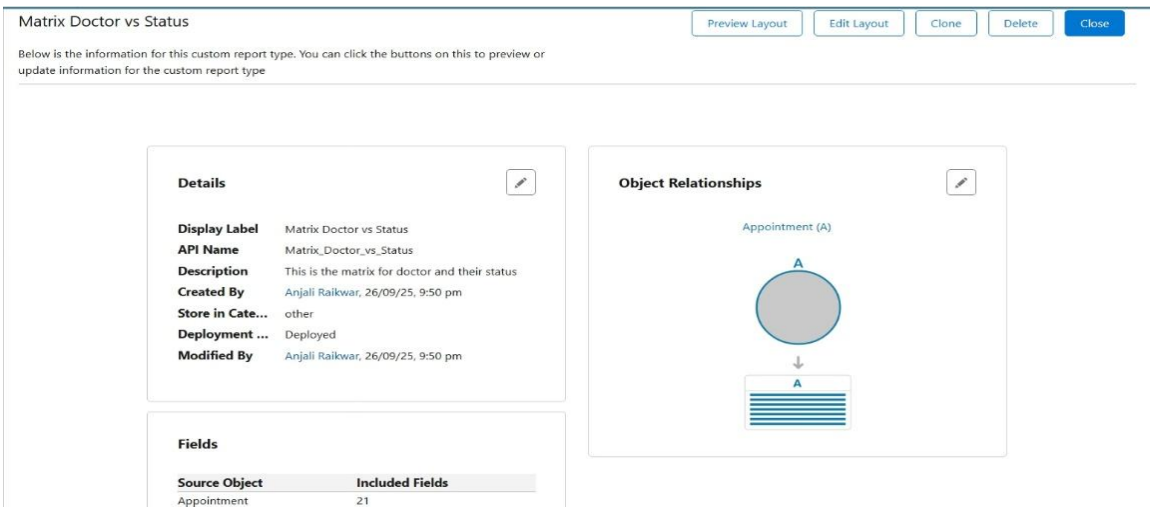




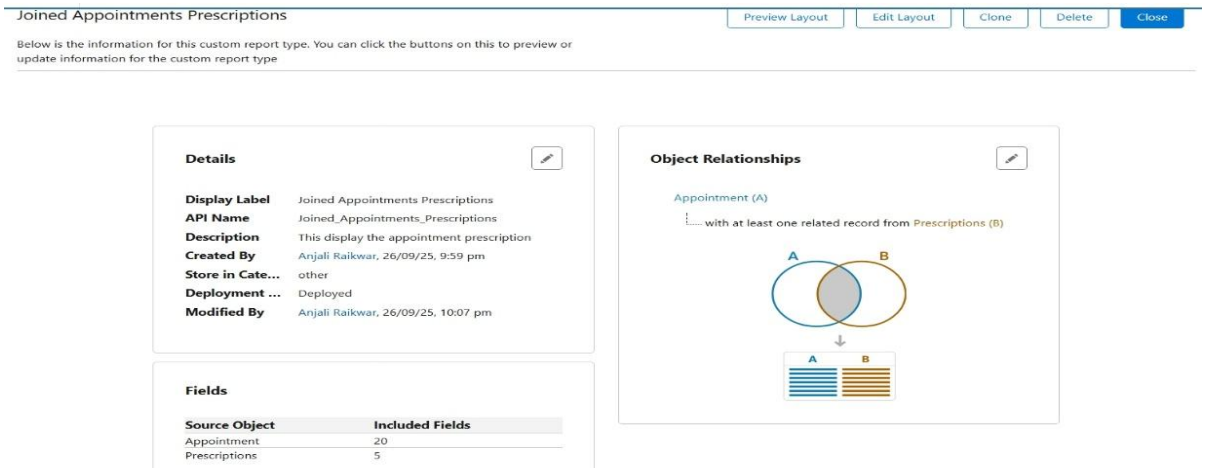
- **Matrix** → Grouped both rows & columns (e.g., doctors vs. appointment status).







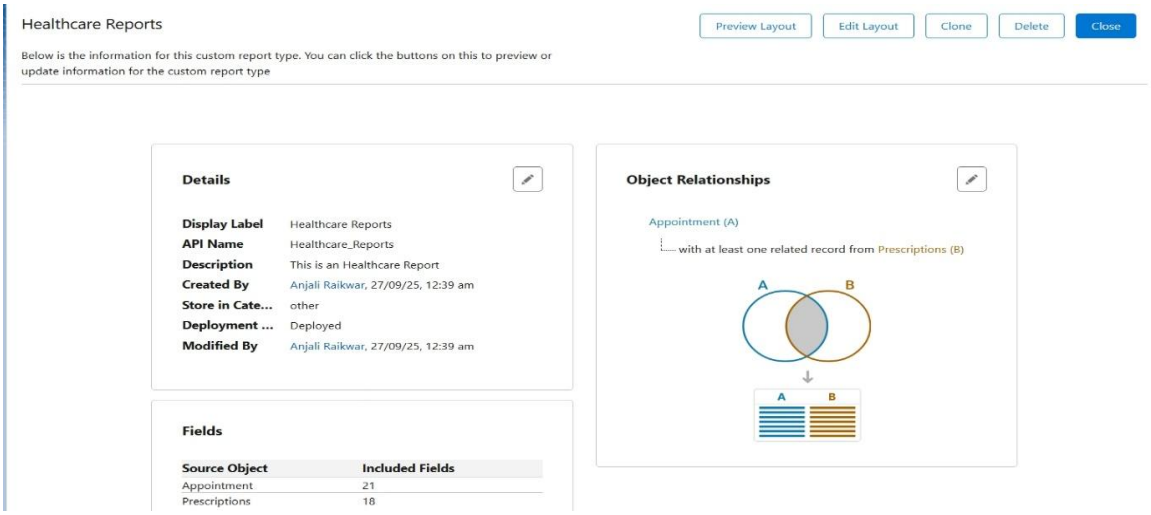
- **Joined** → Combine multiple report types (patient records + prescription data).



Best Use Case → Monitor doctor performance, patient volume, and prescription trends.

- **Report Types**

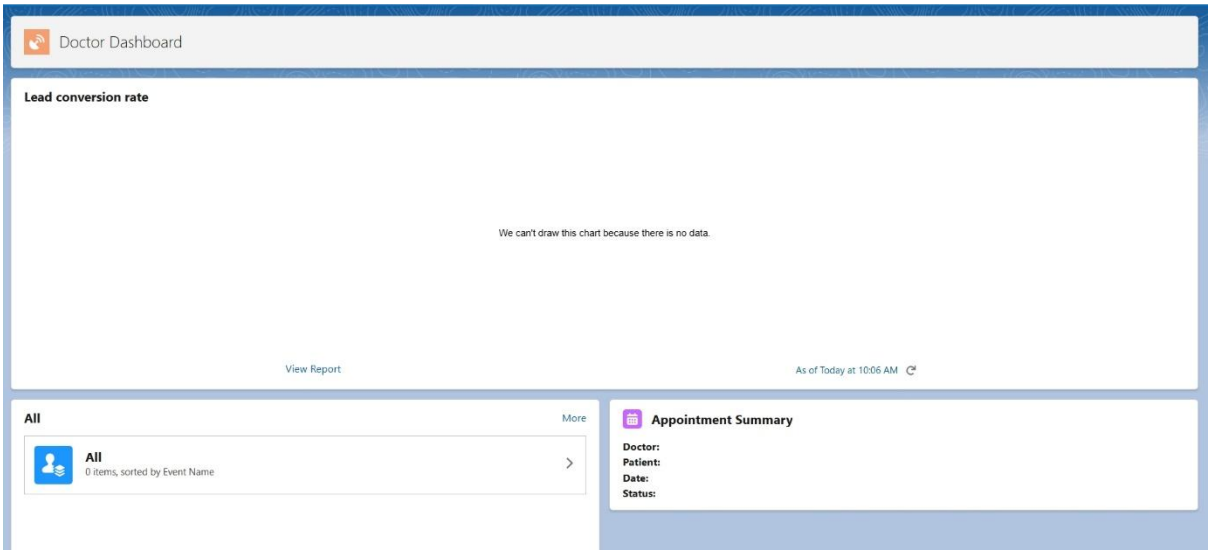
  Define which objects and fields are available in reports.

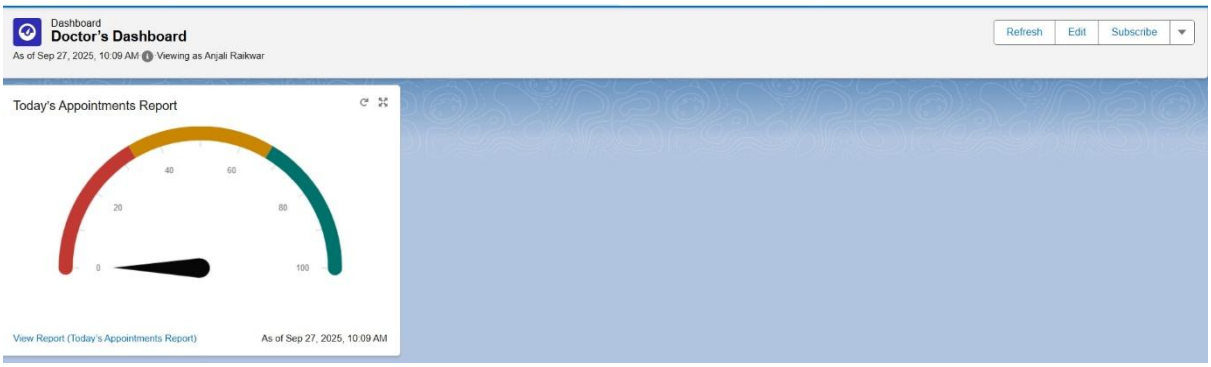


Best Use Case → Custom objects reporting (Appointments + Prescriptions).

- **Dashboards**

  Visualize multiple reports in one view (graphs, charts, gauges).





Best Use Case → Doctor's dashboard:

- Todays Appointments (list)
- Appointment Status (pie chart)
- Top 5 Patients (bar chart)

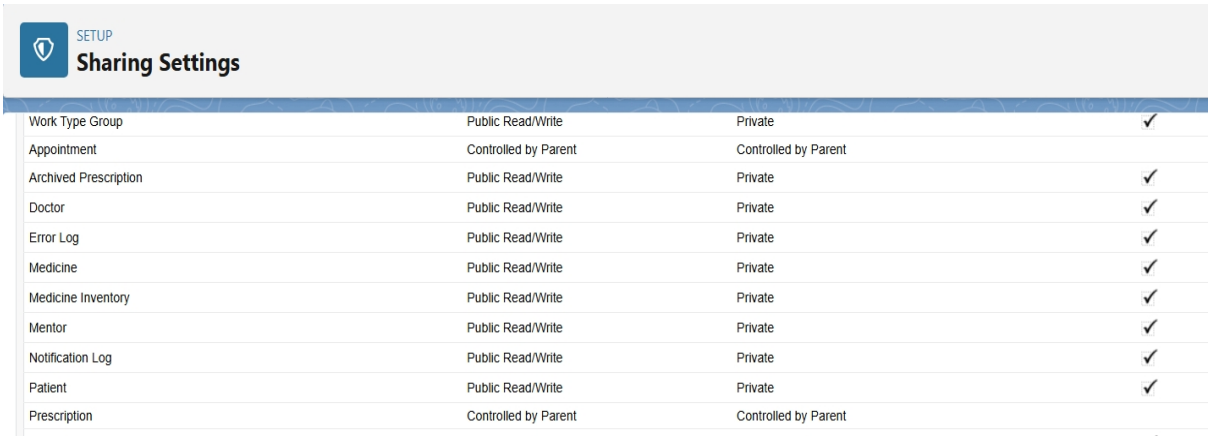
➢ **Dynamic Dashboards**

Dashboards that run as the logged-in user → each doctor sees their own data without creating multiple dashboards.

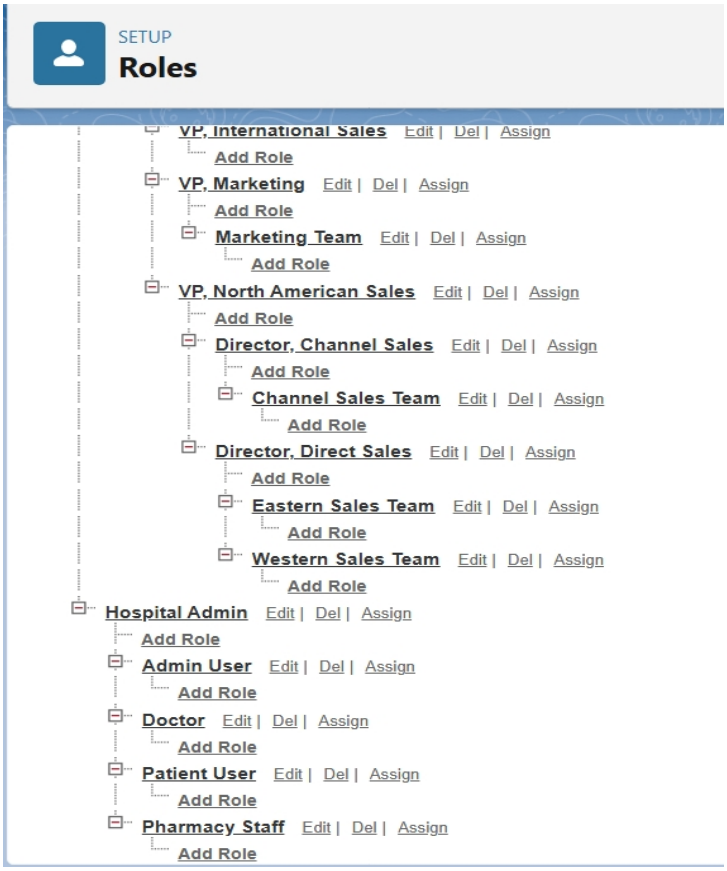
Best Use Case → Each doctor automatically sees only their patients/appointments.

➢ **Sharing Settings**

Controls record-level access (private vs public data).



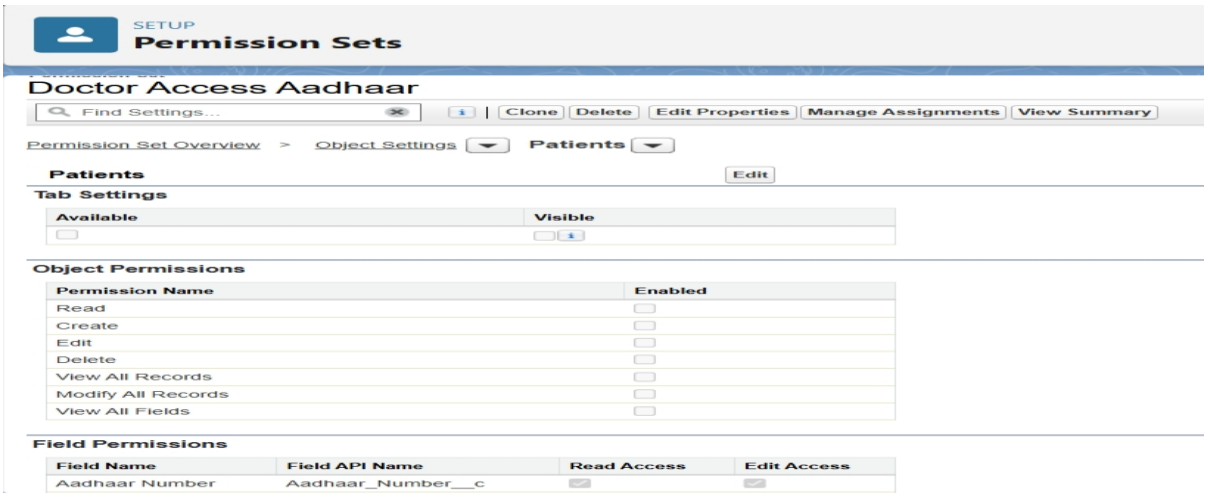
Work Type Group	Public Read/Write	Private	✓
Appointment	Controlled by Parent	Controlled by Parent	
Archived Prescription	Public Read/Write	Private	✓
Doctor	Public Read/Write	Private	✓
Error Log	Public Read/Write	Private	✓
Medicine	Public Read/Write	Private	✓
Medicine Inventory	Public Read/Write	Private	✓
Mentor	Public Read/Write	Private	✓
Notification Log	Public Read/Write	Private	✓
Patient	Public Read/Write	Private	✓
Prescription	Controlled by Parent	Controlled by Parent	

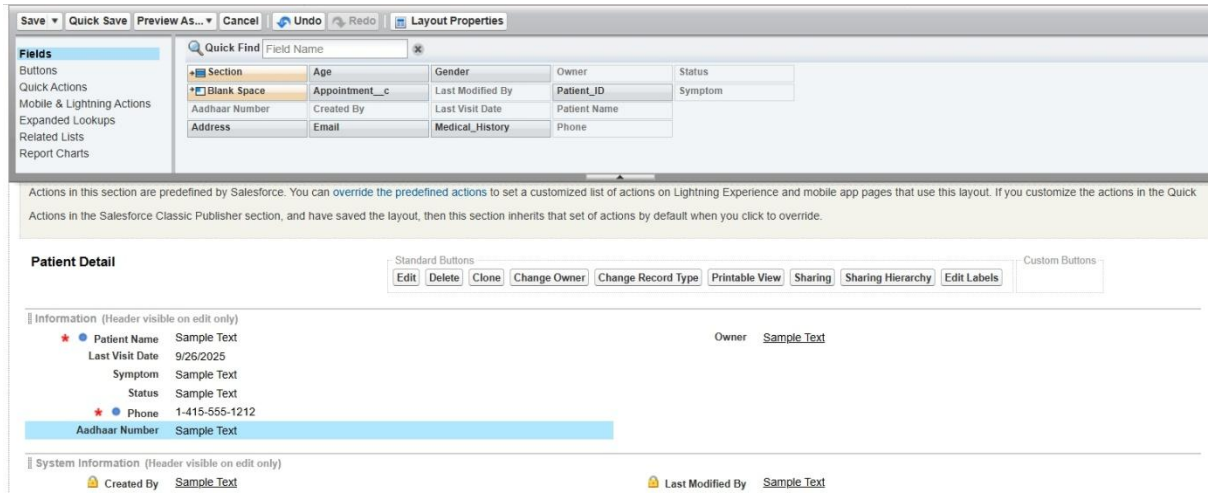


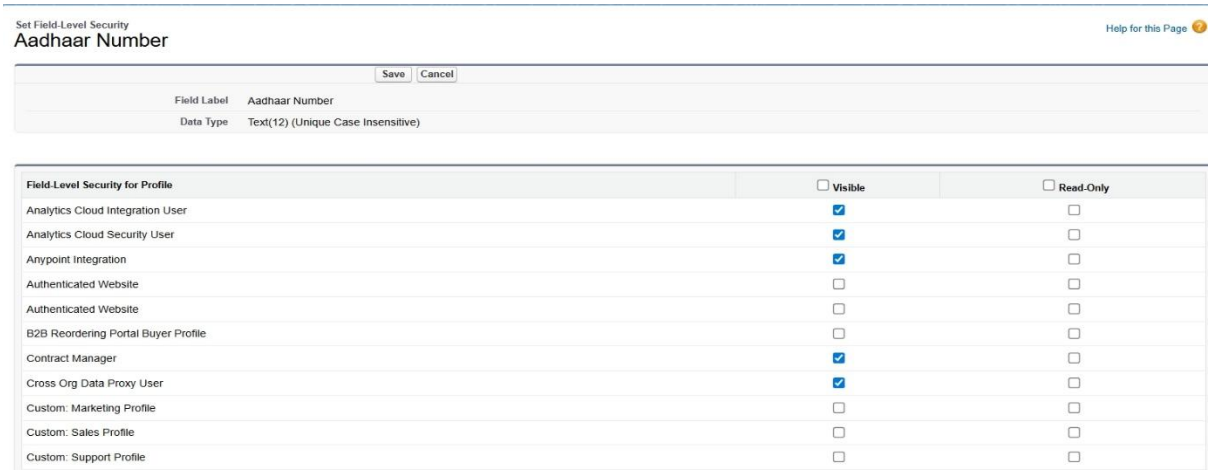
Best Use Case → Protect patient confidentiality.

- **Field Level Security**

  Hide sensitive fields (e.g., Patient Medical History) from unauthorized users.

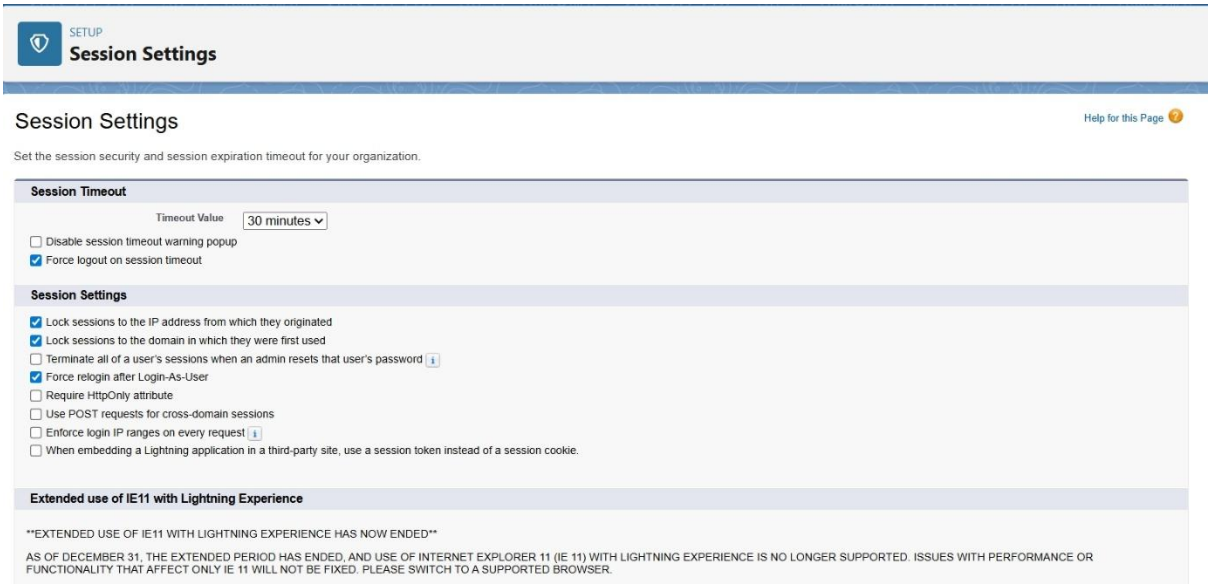






Best Use Case → Admin sees everything, Receptionist only sees Appointment Time & Patient Name.

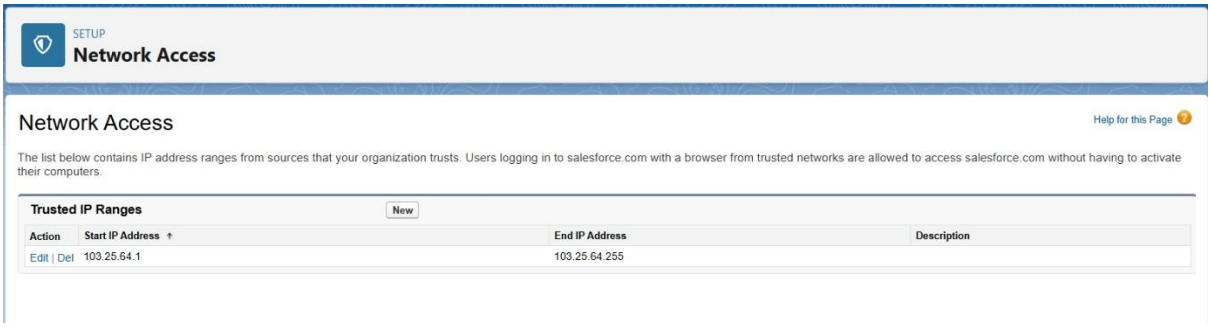
- **Session Settings**
  Controls login security (timeouts, session restrictions).



Best Use Case → Auto log out idle users to prevent unauthorized access.

- **Login IP Ranges**

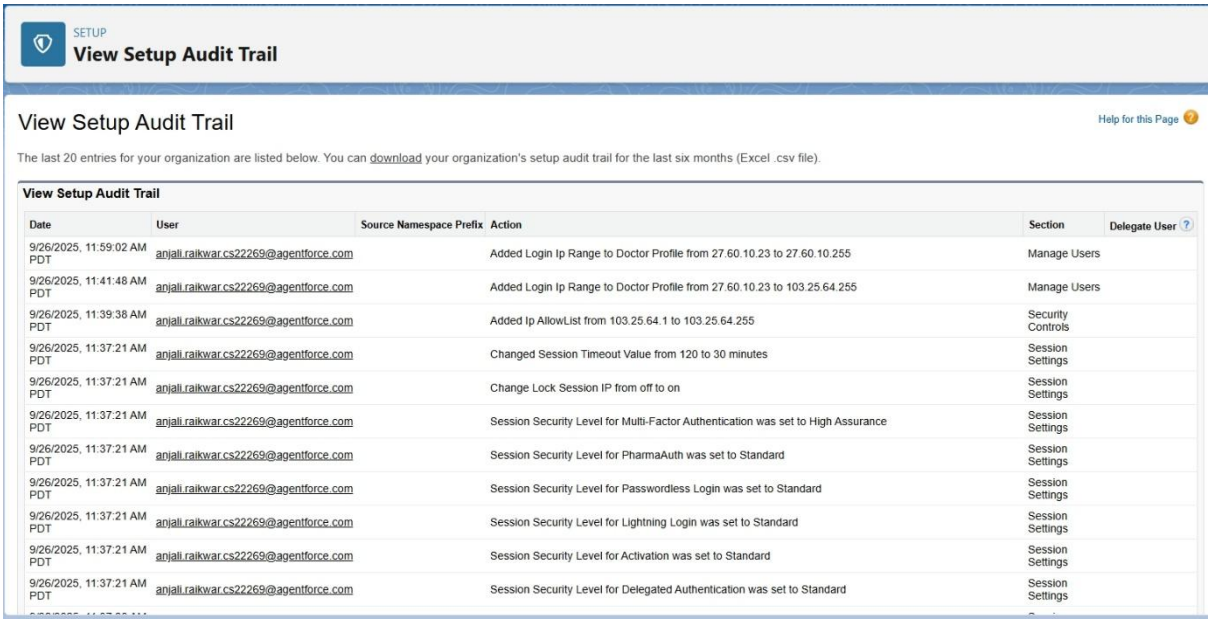
  Restrict user logins to trusted networks (e.g., hospital Wi-Fi).



Best Use Case → Prevent external logins outside hospital network.

- **Audit Trail**

  Track who changed what in setup for compliance.



Best Use Case → Security reviews, compliance with healthcare regulations (HIPAA/GDPR).