Phase-1:Telemedicine Platform with Virtual Care Management

Problem Statement:

Healthcare access is still a serious challenge, especially for people living in rural or remote areas. Many patients struggle with long travel distances, waiting times, and the lack of nearby specialists. During situations like the COVID-19 pandemic, face-to-face consultations became even harder, leaving many without proper medical support.

While telemedicine platforms exist, most are either too costly, lack flexibility, or do not integrate well with secure patient record management. What is needed is a simple, secure, and scalable system that allows patients and doctors to connect virtually, manage appointments, and maintain medical history — all in one place.

Project Idea:

This project proposes the development of a Telemedicine Platform with Virtual Care Management. The system will let patients book online appointments, doctors manage their schedules, and both parties connect through virtual consultations. Doctors can also upload and share digital prescriptions, while patient history and records are safely stored.

By building the platform on a reliable CRM base, the solution takes advantage of security, scalability, and structured record management, making it easier to handle patient-doctor interactions in a structured and reliable way.

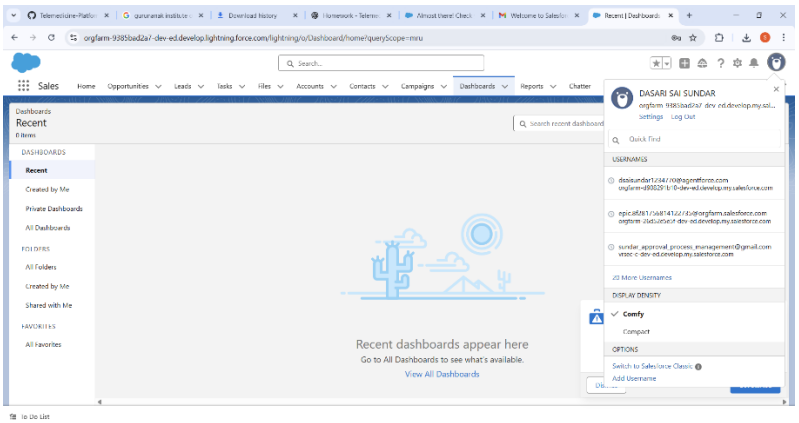
Objectives:

- Appointment Management – Allow patients to book, reschedule, or cancel doctor appointments.
- Doctor Dashboard – Provide doctors with a clear view of their schedule and patient details.
- Virtual Consultations – Generate secure online meeting links for remote consultations.
- Digital Prescriptions – Enable doctors to create and share prescriptions securely with patients.
- Patient Records – Maintain complete medical history, including past visits and prescriptions.
- Role-Based Security – Ensure data privacy with controlled access (patients see only their records, doctors see their assigned patients, admins manage users).
- User-Friendly Interface – Design intuitive components for both patients and doctors.

Phase 2: Org Setup & Configuration

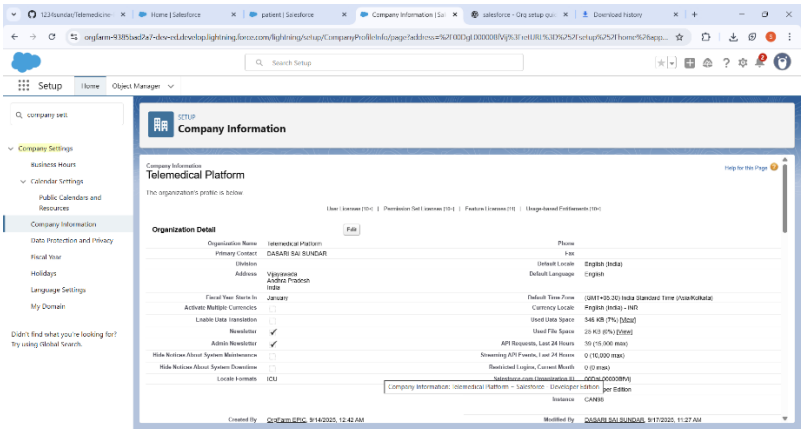
Phase-2: Salesforce Org Setup & Configuration:

1. Salesforce Editions:



We selected the **Developer Edition Org** for this project. A free developer org was registered from Salesforce, which provided all standard CRM features along with advanced tools like APEX, Lightning Web Components (LWC), APIs, and AppExchange access. This org was chosen because it is free, permanent, and includes all features required for development. The Developer Edition was used as the base environment for the telemedicine system. It was set up with administrator access and configured for customization.

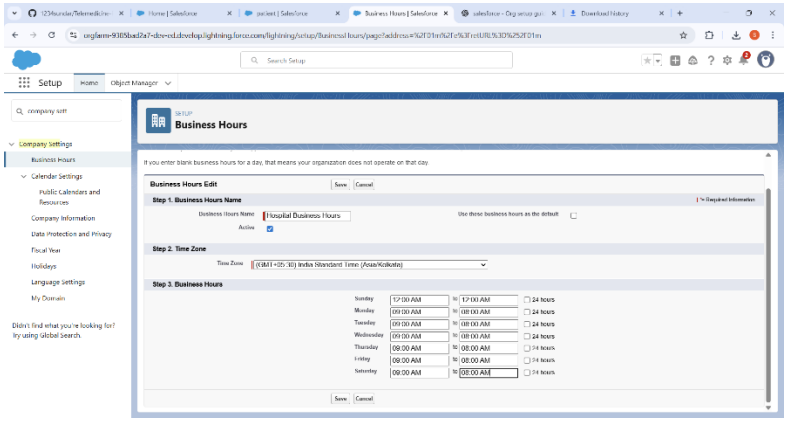
2. Company Profile Setup:



The **Company Profile** was configured under Setup → Company Settings → Company Information. The organization name *TeleMedCare Pvt Ltd* was added along with the

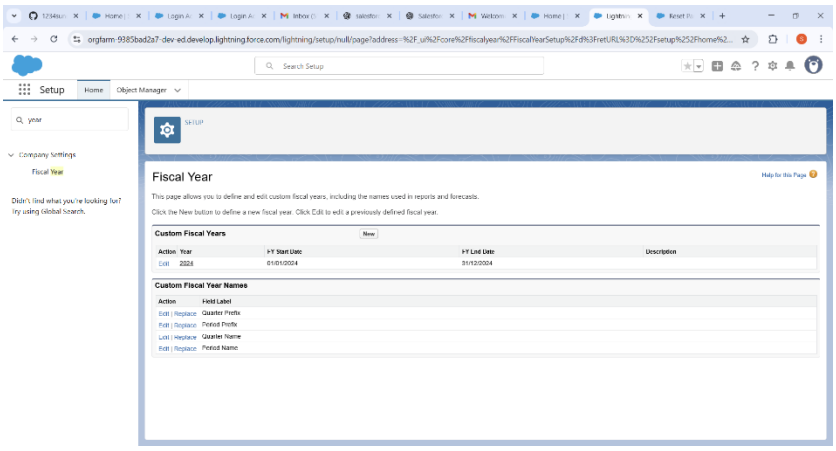
registered address. The local time zone was set to **IST (India Standard Time)** to align with working hours. The default currency was set to **INR**, and the default language was set to **English**. This ensured that appointment schedules, billing, and reporting were accurate for the region. All settings were saved and verified.

3. Business Hours & Holidays:



The **Business Hours** were created for doctors with working timings of **9 AM – 8 PM, Monday to Saturday**. Sundays were left as non-working days. Public holidays were also added to the system, preventing patients from booking appointments on those days. These business hours were linked to the service setup, ensuring that appointments and prescriptions could only be managed during defined working times. This step ensured realistic scheduling aligned with doctor availability.

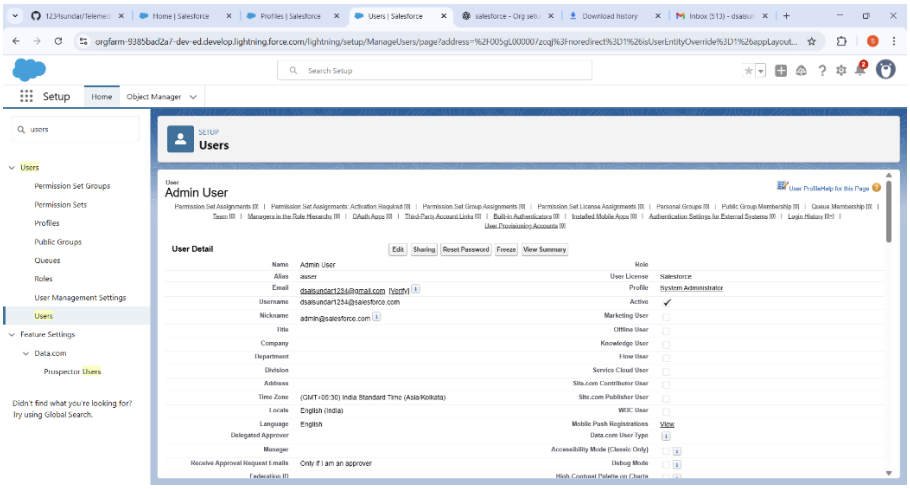
4. Fiscal Year Settings:



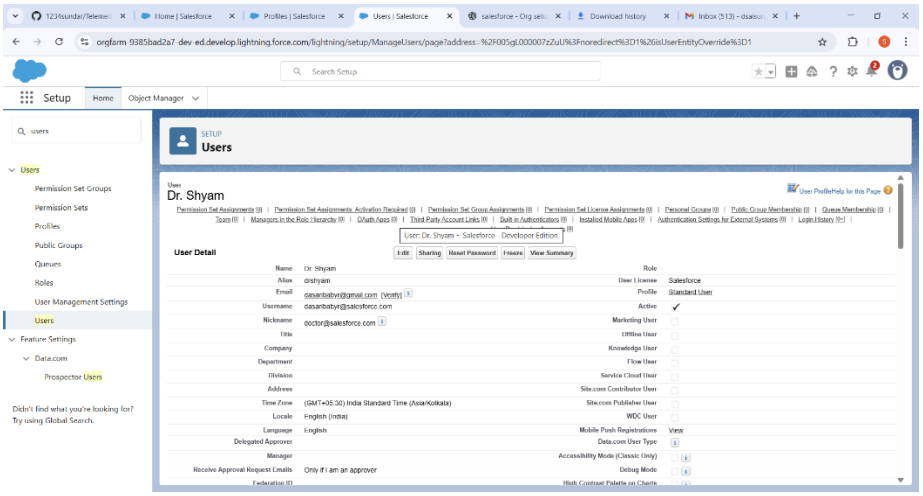
The **Standard Fiscal Year (January – December)** was enabled. A custom fiscal year was not required for this project since healthcare reporting could be handled using the standard model. This configuration ensured that all revenue, appointment, and patient

tracking reports aligned with the calendar year. The default Salesforce fiscal settings were left unchanged as they suited the project's needs.

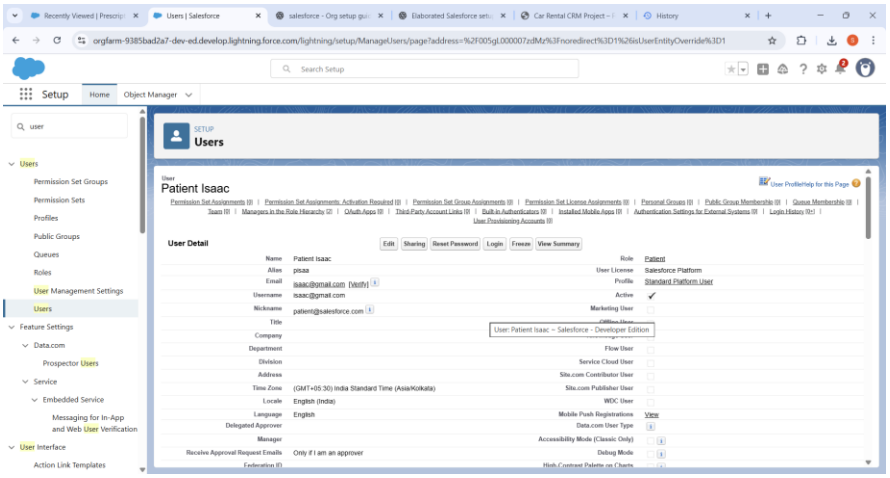
5. User Setup & Licenses:



Multiple users were created in the org. An **Admin User** was added with the System Administrator license.



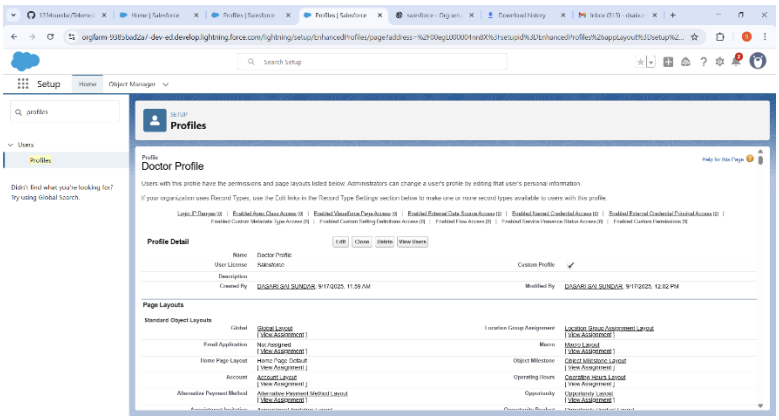
Doctor Users were created with Salesforce licenses and assigned to the doctor profile.



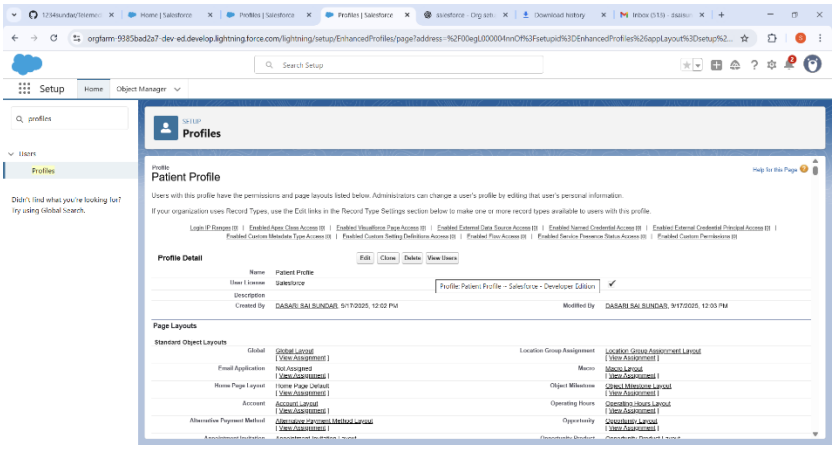
Patient Users were also created with Salesforce licenses but given restricted access. Each user was assigned an appropriate role and profile during creation. Email verification was completed for all users, and login access was tested.

6. Profiles:

Three major profiles were configured. The **Admin Profile** retained full system access and configuration rights.

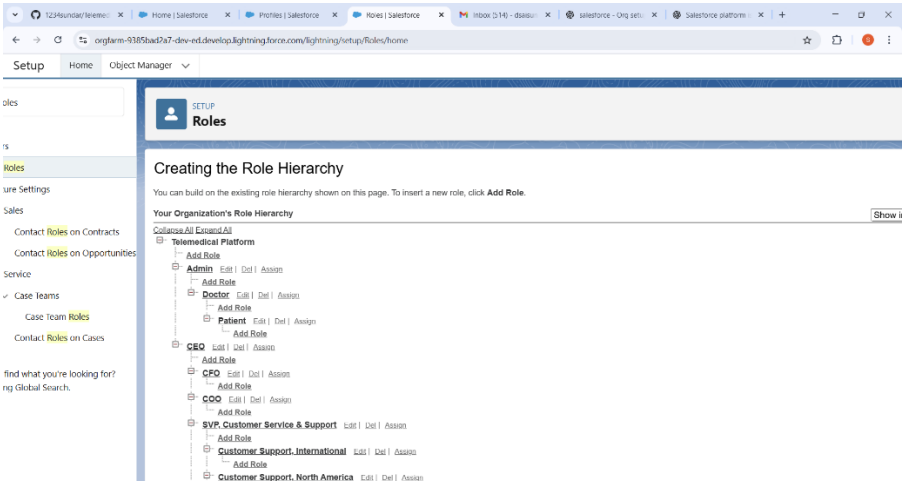


A custom **Doctor Profile** was created to allow doctors to access and update patient records, manage appointments, and create prescriptions.



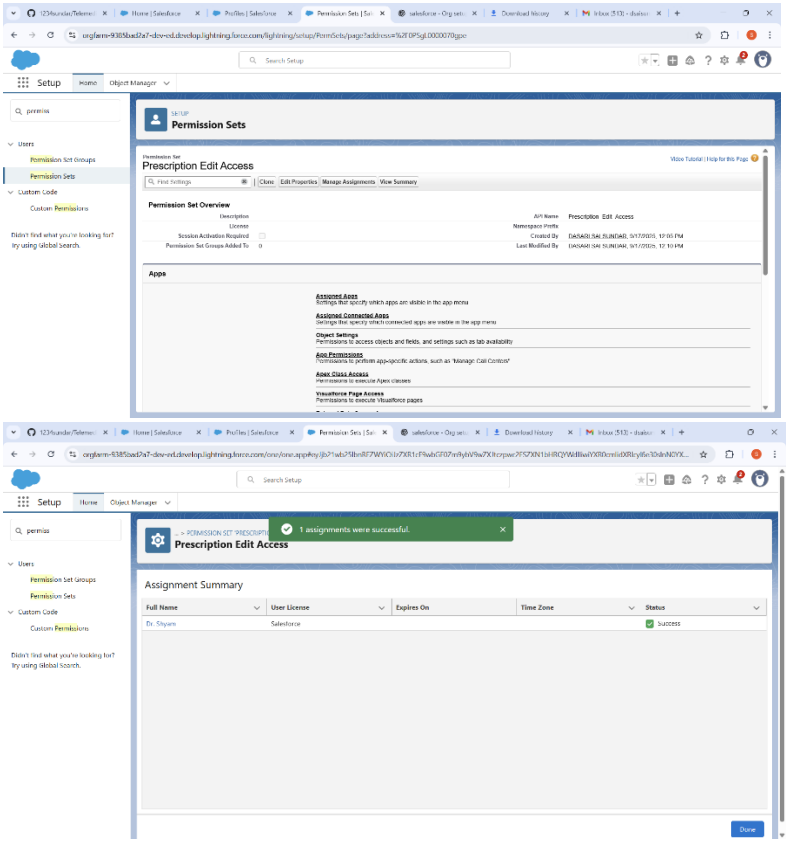
A **Patient Profile** was created with restricted access so that patients could only view their own records and bookings. Each user was assigned the correct profile during setup. This step enforced proper security controls across different roles in the system.

7. Roles:



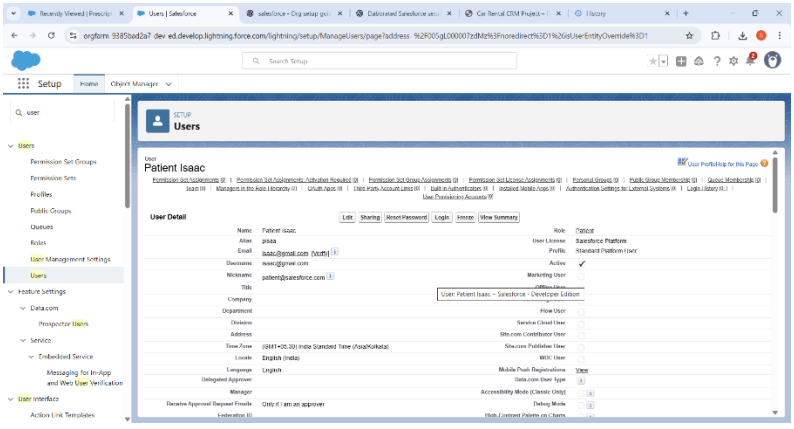
A role hierarchy was created. At the top of the hierarchy, the **Admin Role** was assigned, followed by the **Doctor Role**, and then the **Patient Role**. Doctors were assigned to the Doctor Role, and patients were assigned to the Patient Role. This ensured that data visibility flowed upwards: Admin could see all records, doctors could see their patients' data, and patients only had access to their own data.

8. Permission Sets:



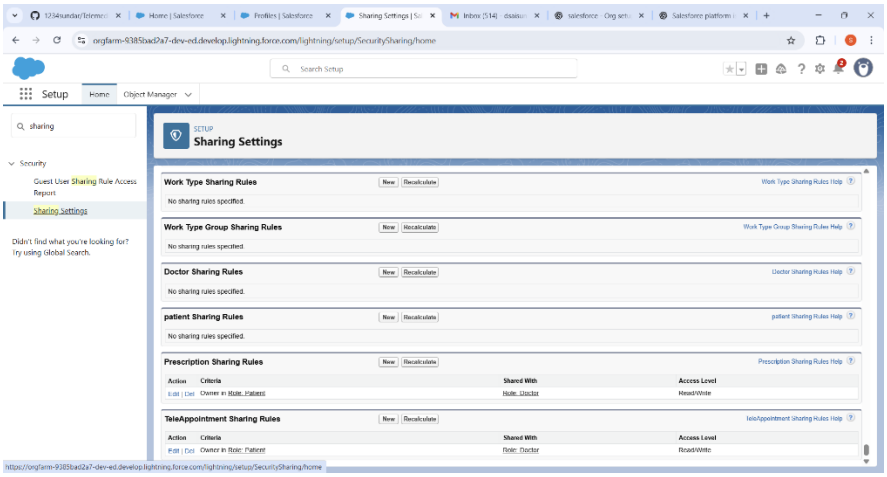
Permission Sets were created to provide extra access where required. For example, a permission set was created to allow doctors to access Reports and Dashboards without modifying their base profile. Patients were not assigned any permission sets as their access was limited by design. This approach ensured flexibility in managing exceptions while keeping profiles simple and secure.

9. Organization-Wide Defaults (OWD):



Organization-Wide Defaults (OWD) were configured to secure sensitive healthcare data. Patient__c was set to **Private**, meaning patients could only view their own records. TeleAppointment__c and Prescription__c were also set to **Private**, ensuring only the assigned doctor and patient could access them. Doctor__c was set to **Public Read Only**, so that doctor details such as names and availability were visible to all users. These settings established the baseline data security for the project.

10. Sharing Rules:

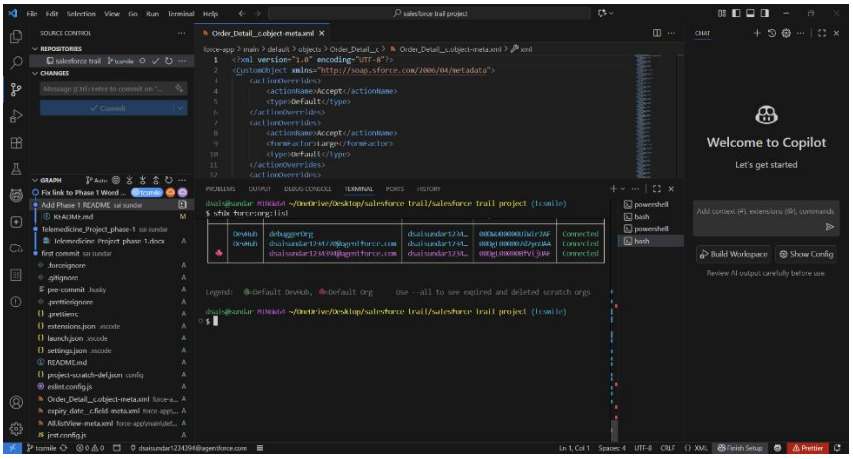


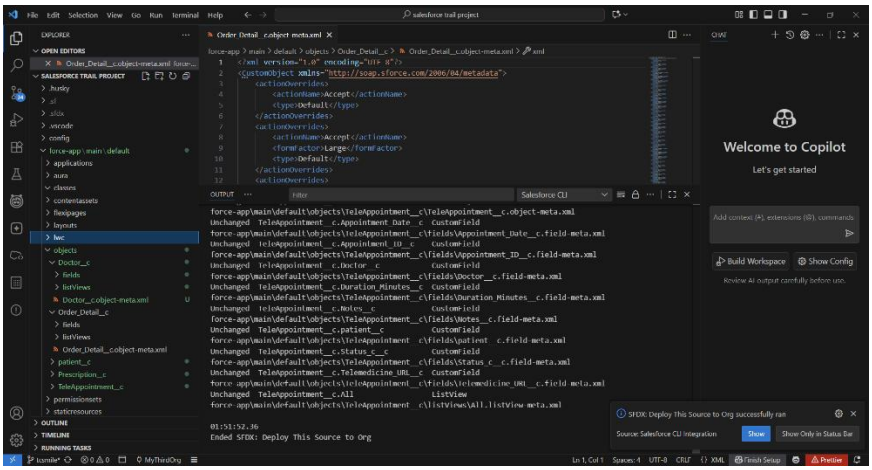
Sharing Rules were created to grant exceptions to OWD restrictions. TeleAppointment__c was shared with the Doctor role with **Read/Write access**. Prescription__c was also shared with the Doctor role with **Read/Write access**. These rules ensured that doctors had full access to appointments and prescriptions assigned to them while keeping patient data secure from unauthorized users.

11. Login Access Policies

Login Hours were configured for different users. Doctors were restricted to log in only between **9 AM – 8 PM**, matching their business hours. Patients were allowed 24×7 login access to book appointments at any time. **Two-Factor Authentication (2FA)** was enabled for all users to increase data security. These policies provided secure access management while supporting flexibility for patient bookings.

12. Deployment Basics:





- **Patient__c** – Custom object deployed through VS Code (SFDX) to store patient details, OWD set to Private.
- **Doctor__c** – Custom object deployed through VS Code (SFDX) to store doctor details, OWD set to Public Read Only.
- **TeleAppointment__c** – Custom object deployed through VS Code (SFDX) to manage appointment bookings, OWD set to Private.
- **Prescription__c** – Custom object deployed through VS Code (SFDX) to manage prescriptions, OWD set to Private.

Deployment methods were finalized. **Change Sets** were configured for point-and-click deployments between Sandbox and Developer Org. In addition, **Salesforce DX (SFDX) with VS Code** was set up for advanced CLI-based deployments. Both methods were tested to confirm that metadata, objects, and configuration changes could be moved successfully. This completed the setup of a secure, structured deployment process for the Telemedicine project.

Phase 3: Data Modeling & Relationships

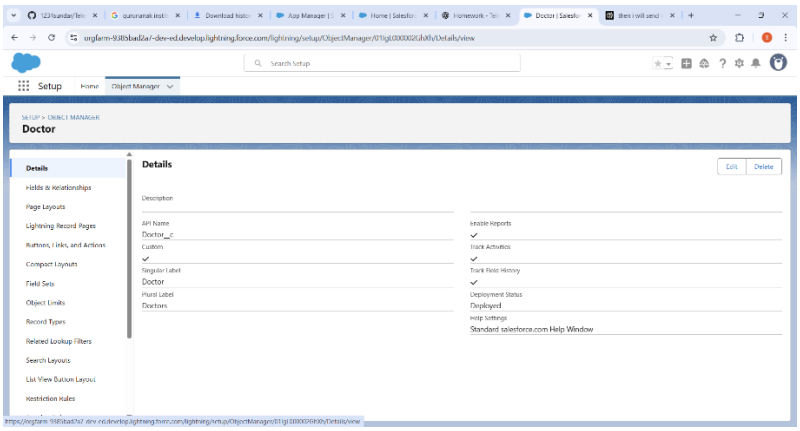
Phase 3: Data Modeling & Relationships

1.Custom Objects:

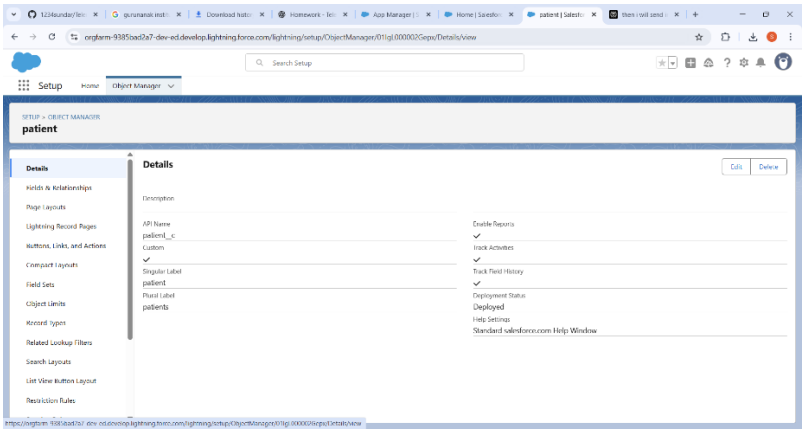
Created **Custom Objects** to support our use case:

- **Doctor** → Stores doctor-specific data (specialization, availability, license info).
- **Patient**→ Holds patient details (age, gender, medical history).
- **Appointment** → Connects patients with doctors and stores consultation details.
- **Prescription** → Stores prescriptions, reports, and treatment history.

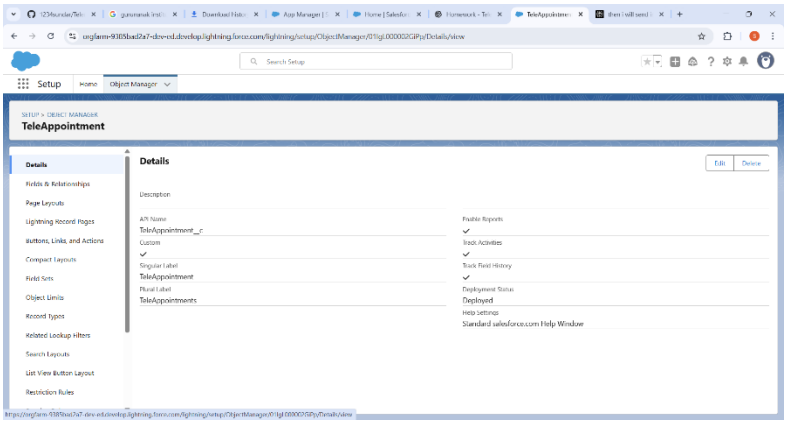
Doctor Object:



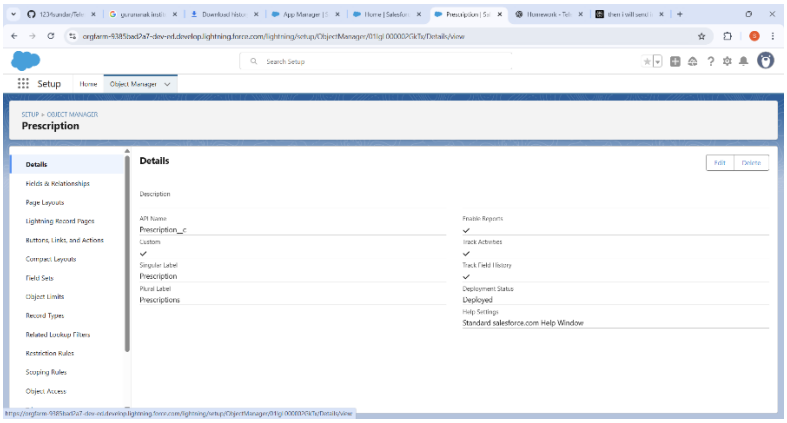
Patient Object:



TeleAppointment Object:



Prescription Object:

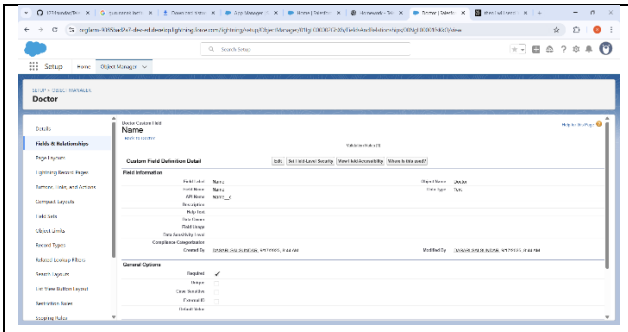


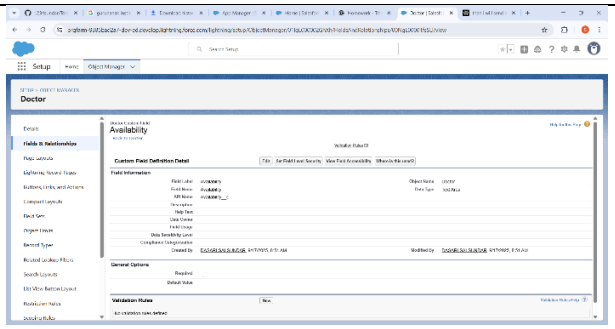
2.Fields:

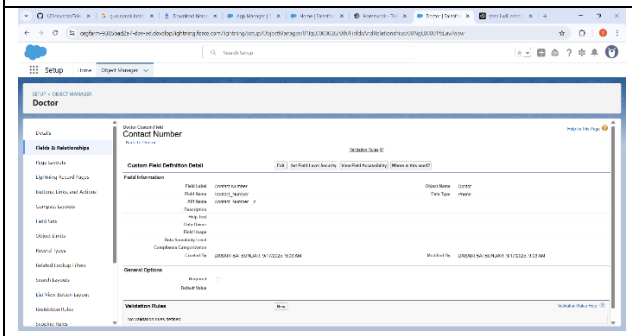
For each custom object, created fields to store critical information:

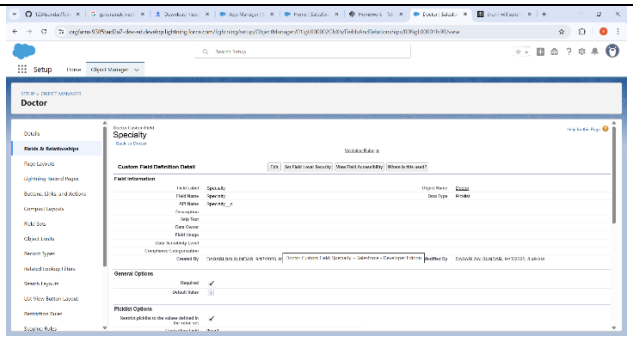
- **Doctor**→ Specialization, Consultation Fee, Experience, Availability (Picklist).
- **Patient** → Age, Gender, Contact Number, Health Condition (Text/Number/Picklist).
- **Appointment**→ Appointment Date/Time, Mode of Consultation (Virtual/Physical), Status (Scheduled/Completed/Cancelled).
- **MedicalRecord** → Diagnosis, Prescription (Rich Text), Report Upload (File).

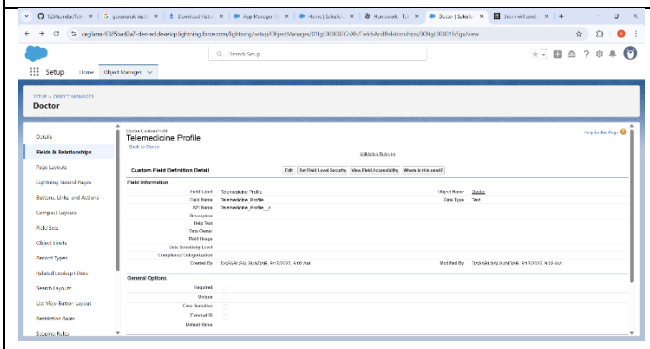
a. Fields for the Doctor Object are:

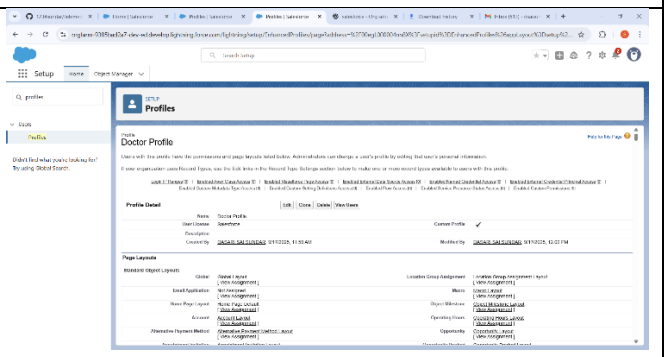




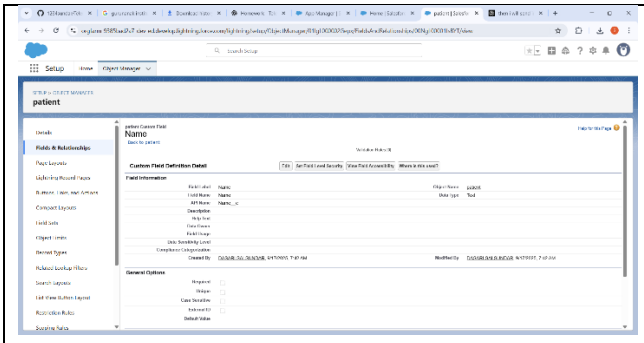


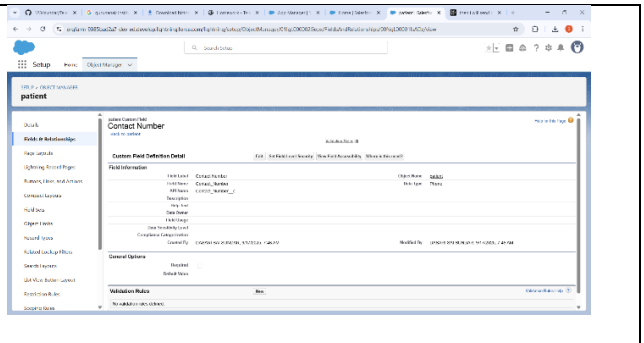


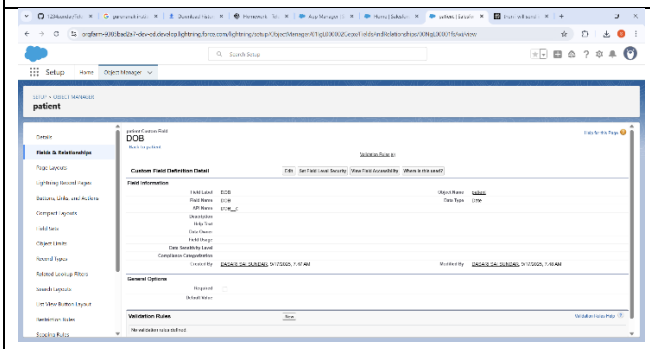


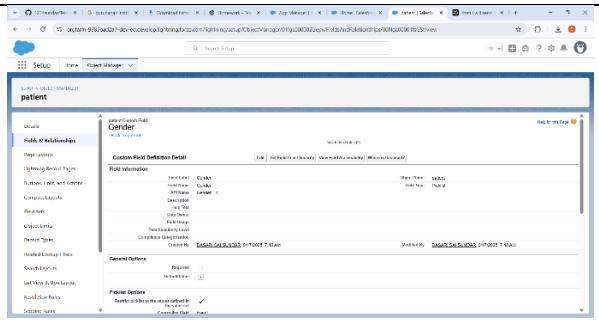


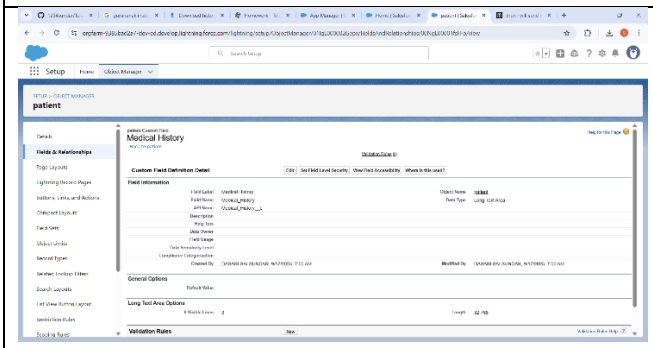
b. Fields for the Patient Object are:

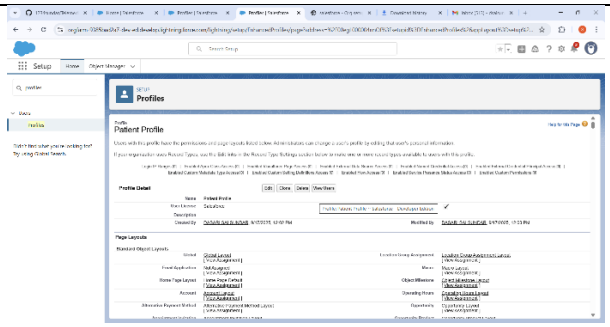




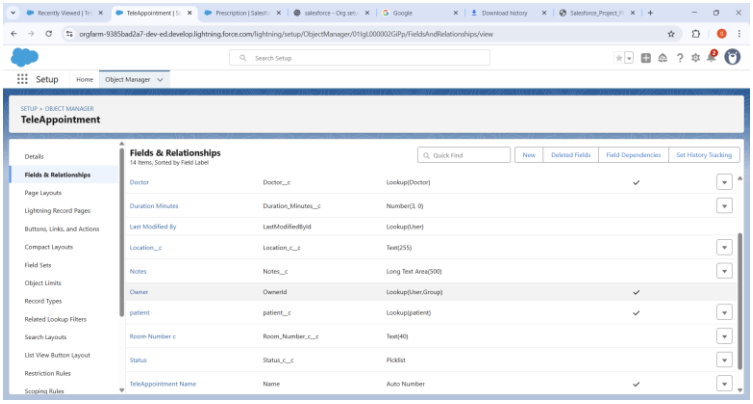






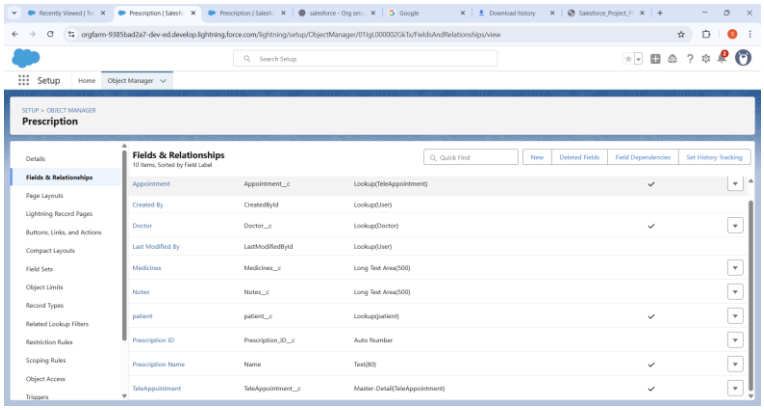


c. Fields for the TeleAppointment Object are:



Like wise several fields are created for the TeleAppointment Object.

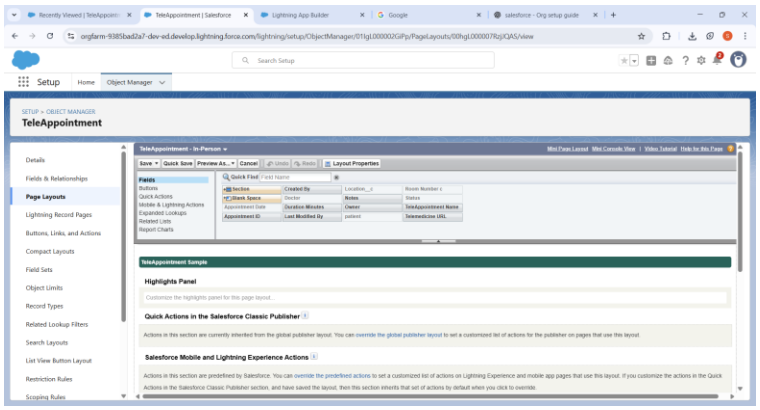
Fields for the Prescription Object are:



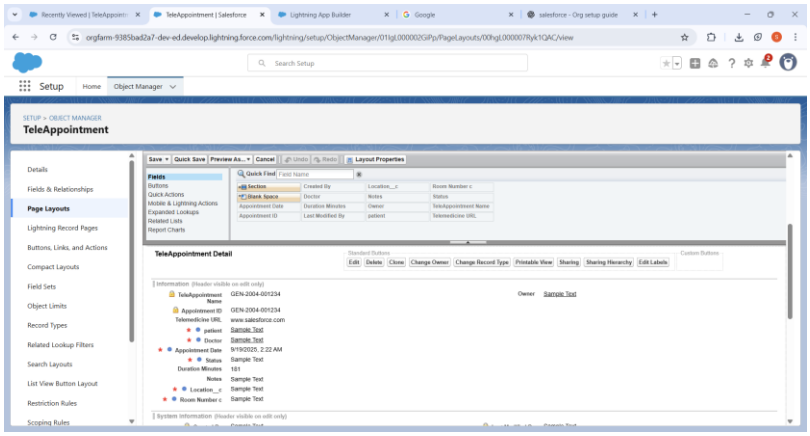
3.Create the page Layouts:

Page layouts were configured for each object to make the interface intuitive and user-friendly. Compact layouts were designed to highlight the most important information for quick access, allowing users to see key details without opening the full record.

A. Tele appointment in- person



B. Tele appointment Online:

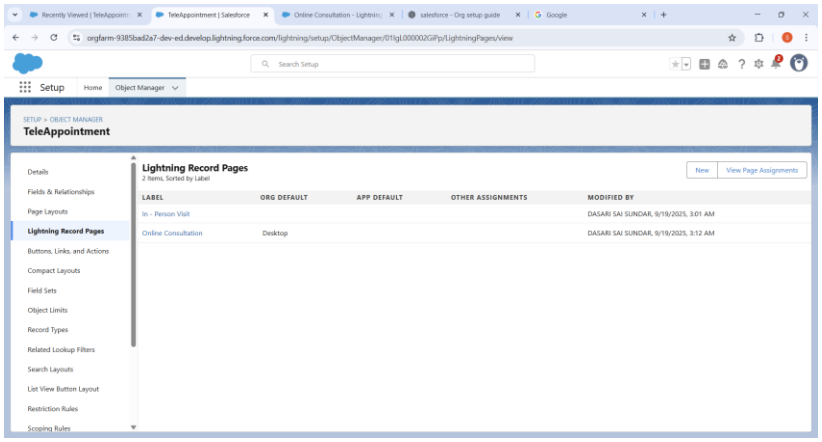


4. Create Record type:

Go to Setup → Object Manager → Select Object → Record Types → New.

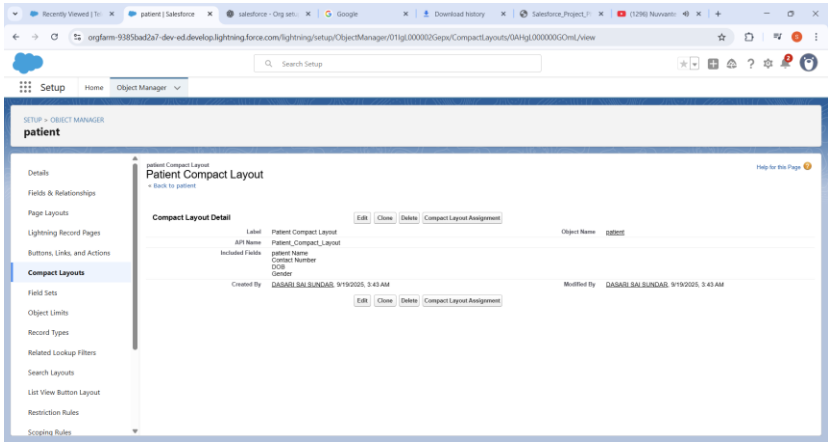
Enter the record type name and description.

Select a profile or multiple profiles that will have access to this record type**.**

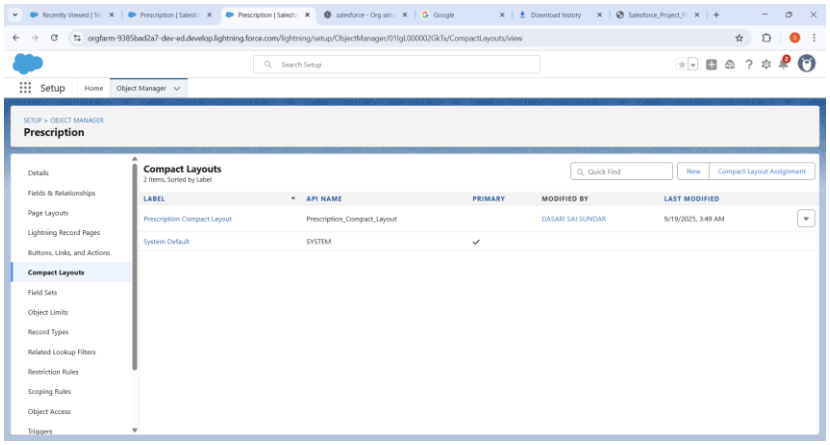


5. Compact Layout of the Project:

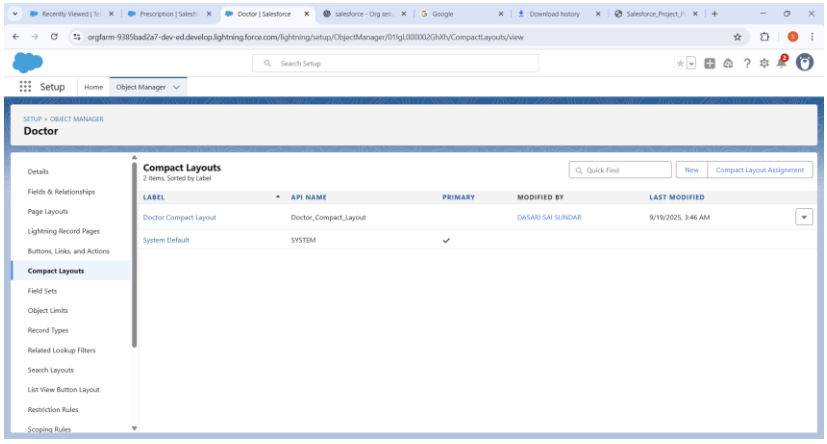
a. Patient Compact Layout:



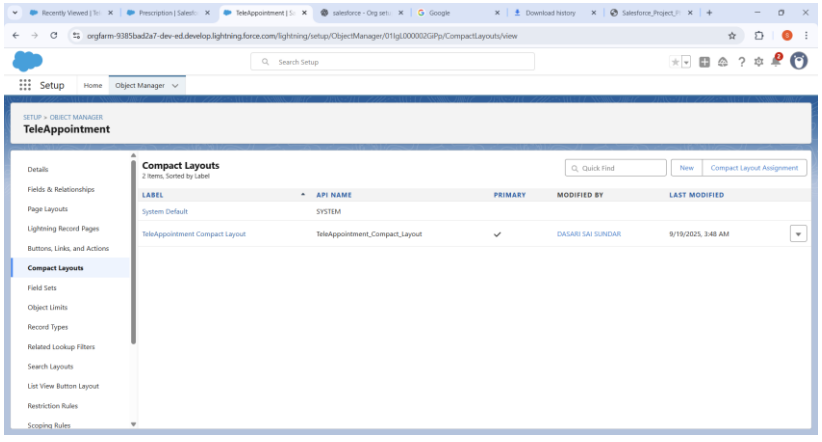
b. Prescription Compact Layout:



c. Doctor Compact Layout:



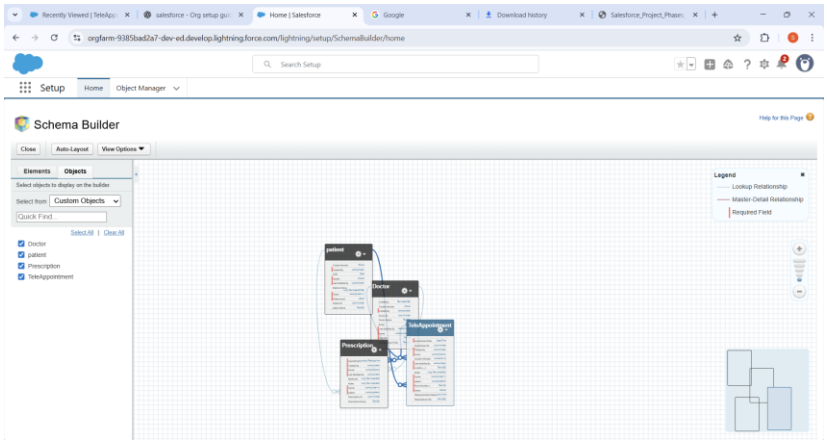
d. TeleAppointment Compact Layout:



We designed Compact Layouts to highlight the most important details at the top of the record page:

- Doctor → Name, Specialization, Availability.
- Patient → Name, Age, Gender, Health Condition.
- Appointment → Appointment Date, Time, Status.

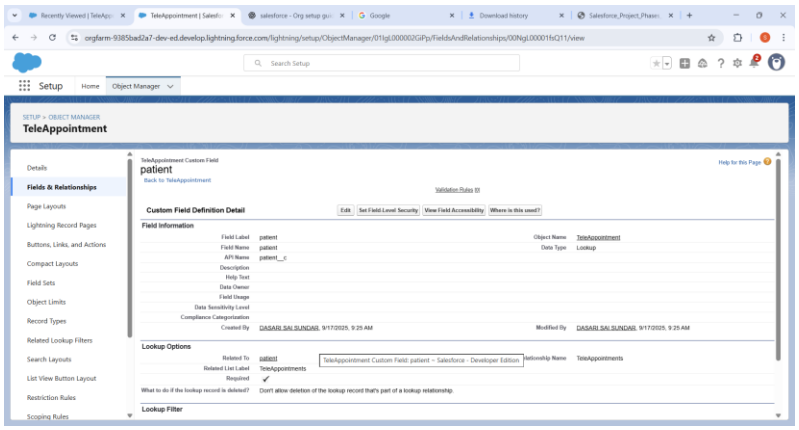
6.Schema Builder:



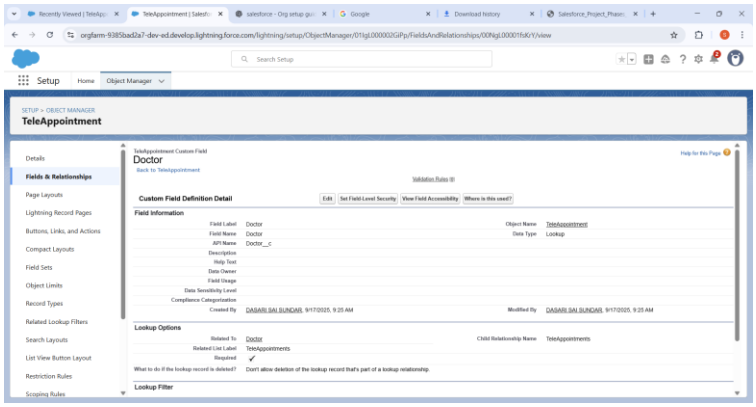
We used **Schema Builder** to visualize the entire data model. Mapped the relationships between Doctor, Patient, Appointment, and Medical Record .Ensured correct lookup and master-detail dependencies. Adjusted field placements for clarity.

7.Relationships:

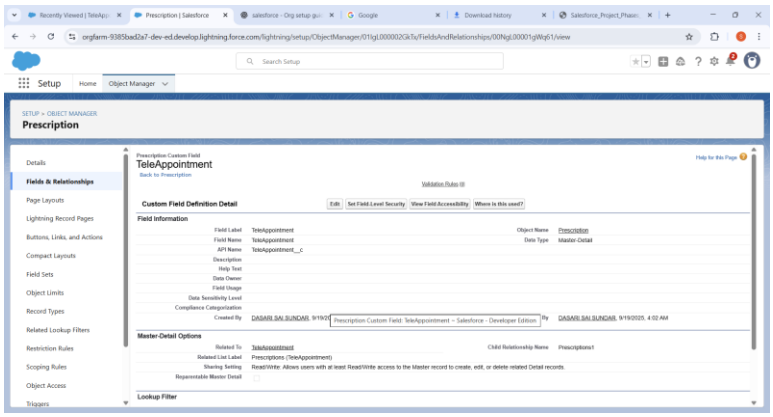
1. The TeleAppointment Lookup relation with the Patient Object



2. Lookup relation between the TeleAppointment to the Doctor.

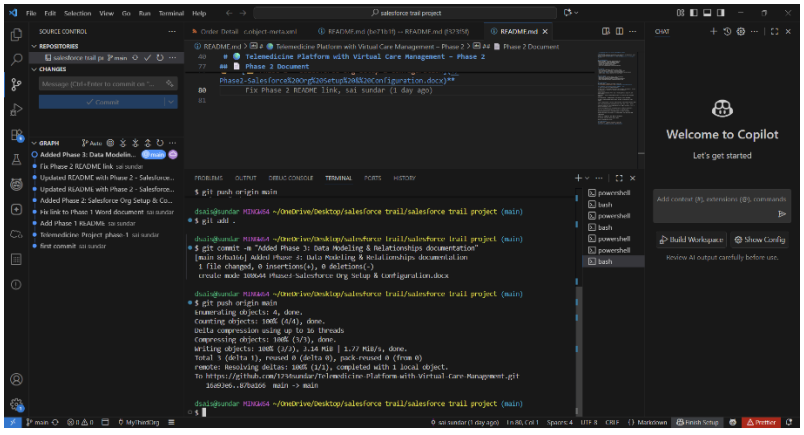


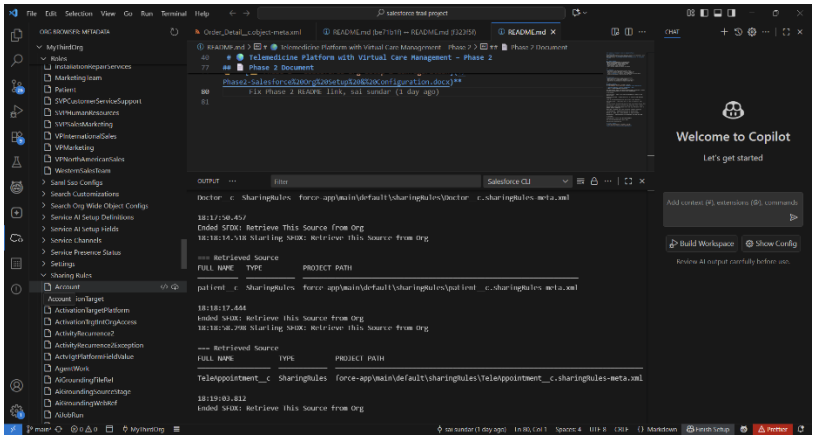
3. Master Detail Relation between the Prescription to the Teleappointment.

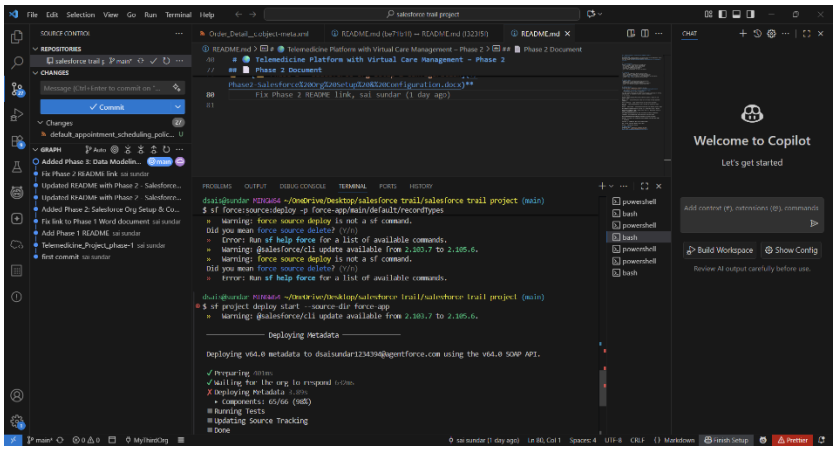


8. Deployment of the file in the Git hub using the vscode:

All Phase 3 metadata, including objects, fields, record types, layouts, and relationships, have been deployed to the Salesforce org.Documentation phase to is uploaded to the github repository successfully.




Welcome to Copilot
Let's get started
Build Workspace
Show Config

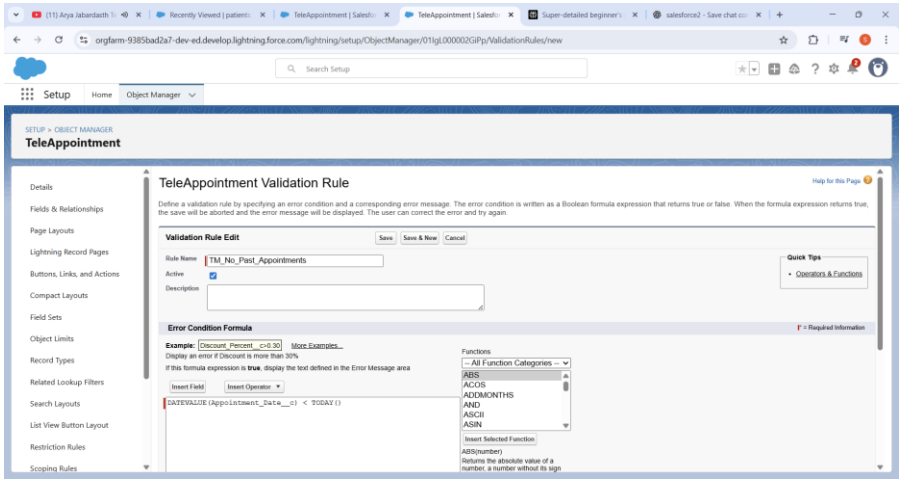

Welcome to Copilot
Let's get started
Build Workspace
Show Config

Phase 4: Process Automation (Admin)

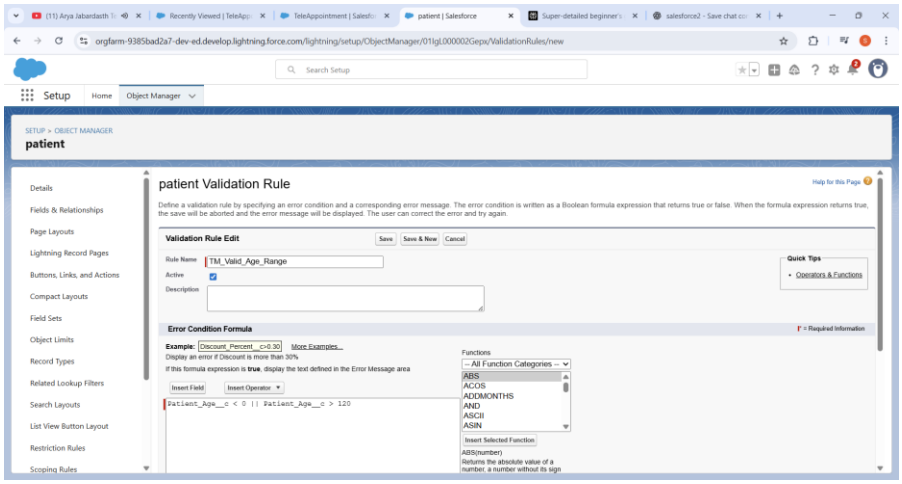
Phase 4: Process Automation (Admin)

1.Validation rule:

a) No Past Appointments:



b) Patient Valid Age Range:



Validation Rules

A validation rule is a simple way to **prevent bad data from being saved**. It checks a record before it's saved and, if the rule is true, it displays an error message.

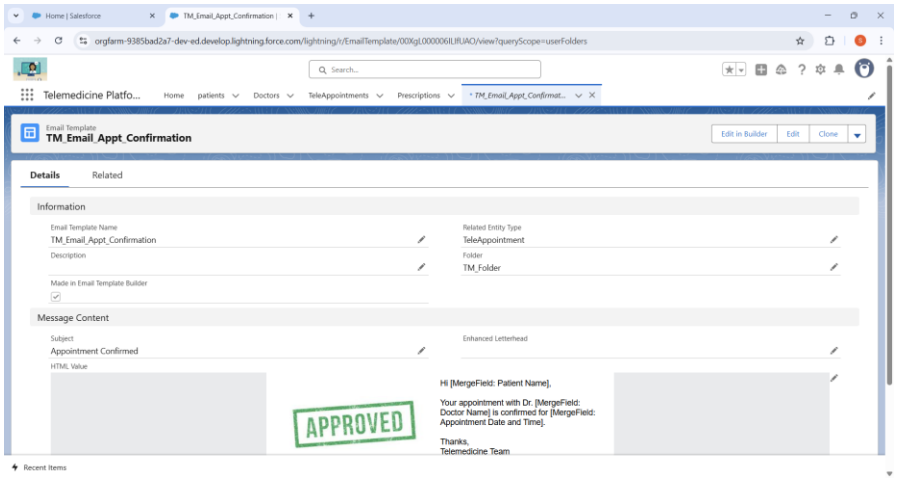
- **TM_No_Past_Appointments**: This rule prevents a user from creating a TeleAppointment with a date that is in the past. It uses the formula

DATEVALUE(Appointment_Date__c) < TODAY().

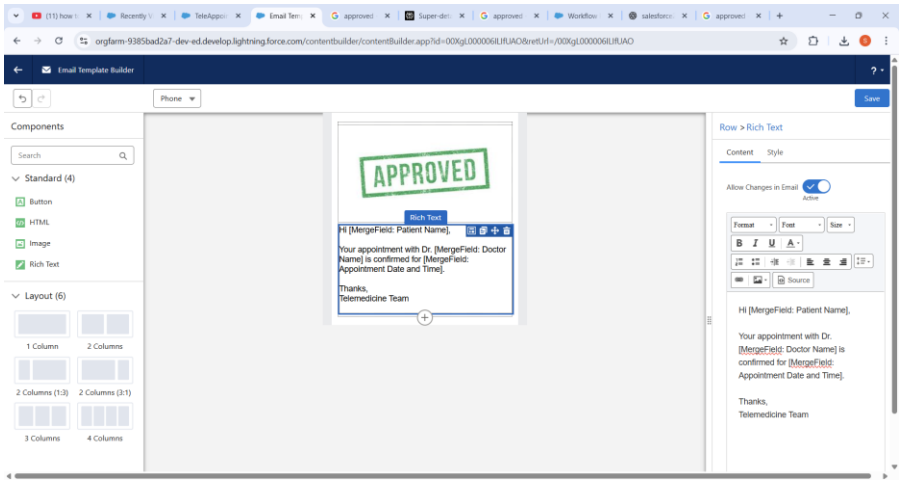
- **TM_Valid_Age_Range**: This rule ensures a Patient's age is within a valid range. It uses the formula Patient_Age__c < 0 || Patient_Age__c > 120.

2.Lightning Email Template:

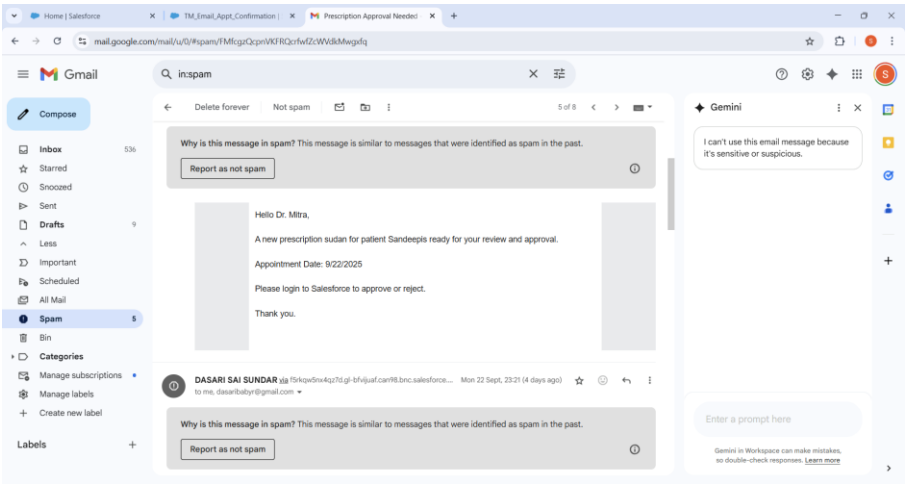
a) Created lightning Email Template→Email Appointment confirmation:



b) Design the Email Template with the builder:



c) Test Case1:

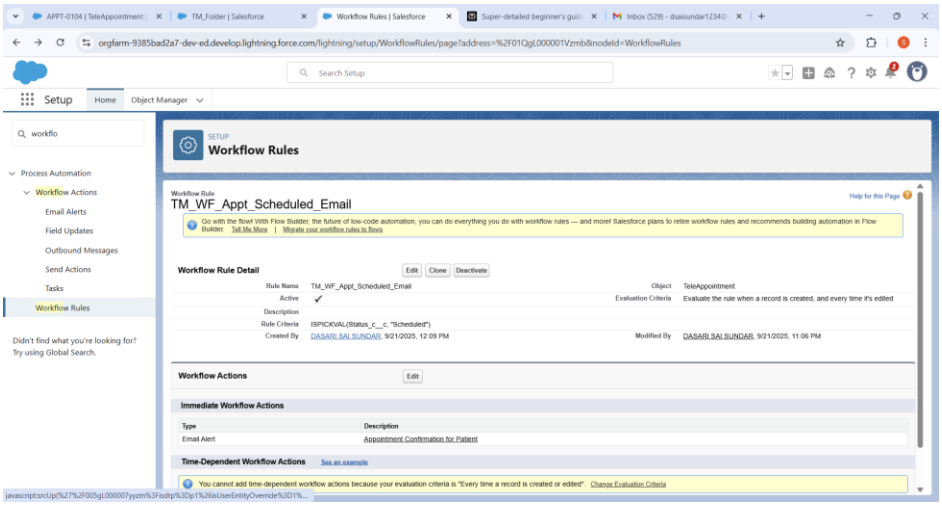


The email template is created with the above structure.

The email that gets sent uses a

Lightning Email Template, which includes an "APPROVED" stamp and merge fields for the Patient Name, Doctor Name, and Appointment Date and Time.

4.Workflow Rules Validation:



Created The TM_WF_Appt_Scheduled_Email

A Workflow Rule is a tool that performs an action when a record meets a specific criteria.

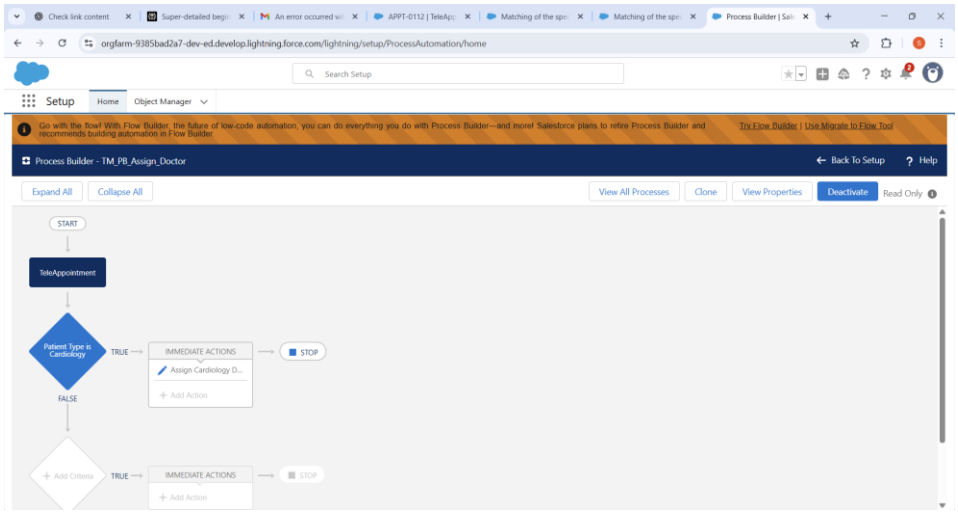
- TM_WF_Appt_Scheduled_Email: This rule automatically sends an email to the patient when a TeleAppointment's status is set to "Scheduled."

- The email that gets sent uses a

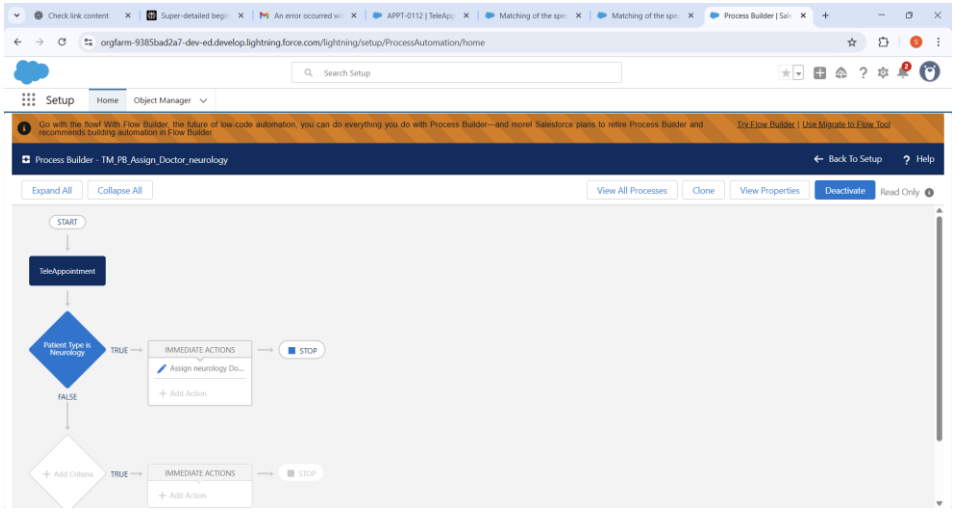
Lightning Email Template which includes an "APPROVED" stamp and merge fields for the Patient Name, Doctor Name, and Appointment Date and Time.

6.Process Builder:

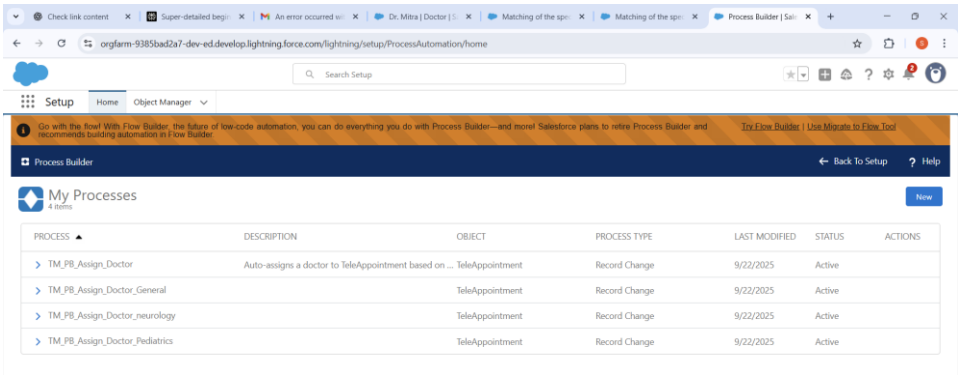
In TeleAppointment Object when we select speciality we can get assigned the doctor according to the specialization.



Here for the cardiology assigned doctor can be allocated.



Here for the cardiology assigned doctor can be allocated.



Same as the General and Pediatrics doctor can be allocated accordingly.

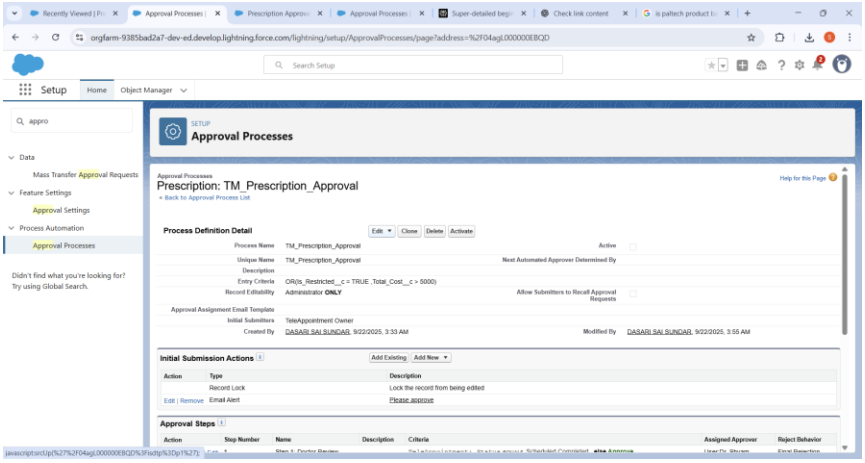
Process Builder is a powerful tool for automating business processes using a visual interface. It works by checking criteria and then performing immediate actions.

- **Assigning a Doctor**: You have created a Process Builder that automatically assigns a doctor to a TeleAppointment record based on the patient's selected specialty.
- For example, if the

Patient Type is 'Cardiology', a specific Cardiology doctor is assigned. Similarly, doctors are allocated for 'Neurology', 'General', and 'Pediatrics' specialties.

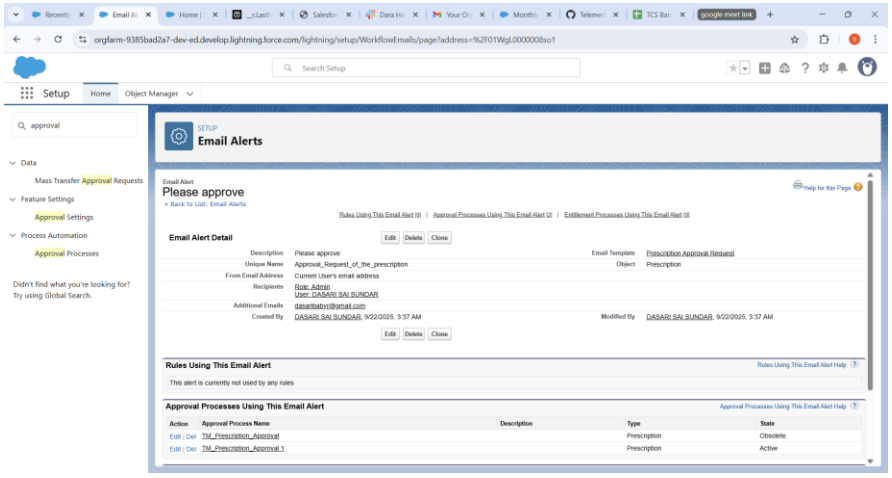
7.Approval Process:

a) Created the Approval Process:



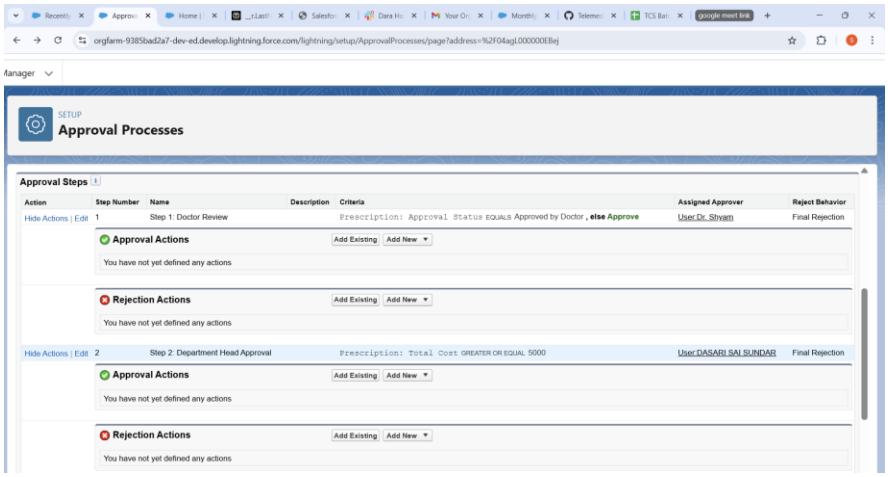
b) Initial Submission Action:

- Email Alert alert is created with the name Please Approve



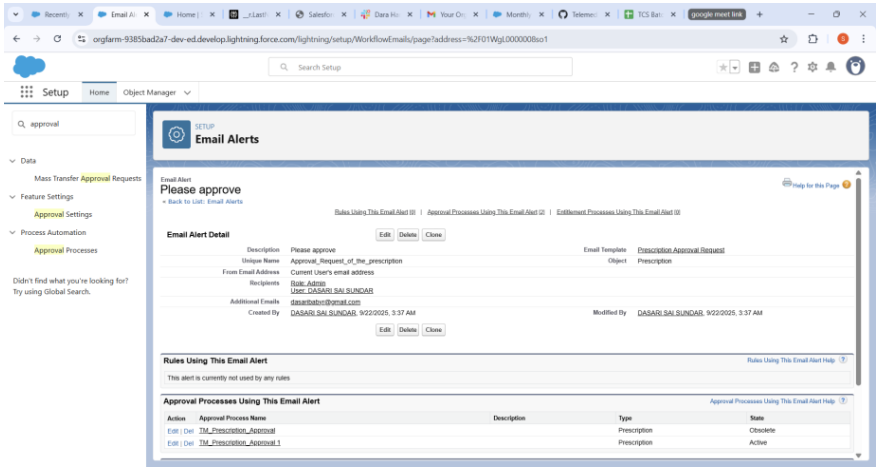
c) Approval Steps:

- Two approval steps created with names 1.Doctor Review ,2.Department head Approval
  Formula-1: Prescription: Approval StatusequalsApproved by Doctor **, else Approve**
  Formula-2: Prescription: Total CostGREATER OR EQUAL5000

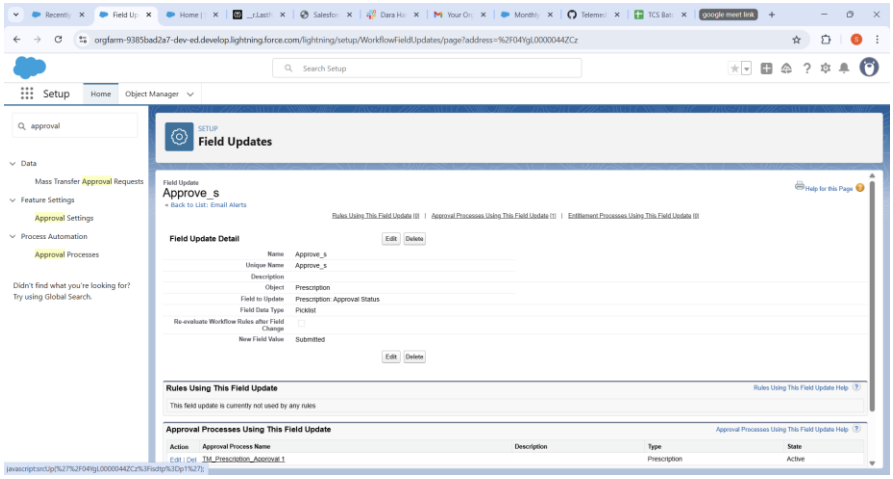


d) Final Approval Action:

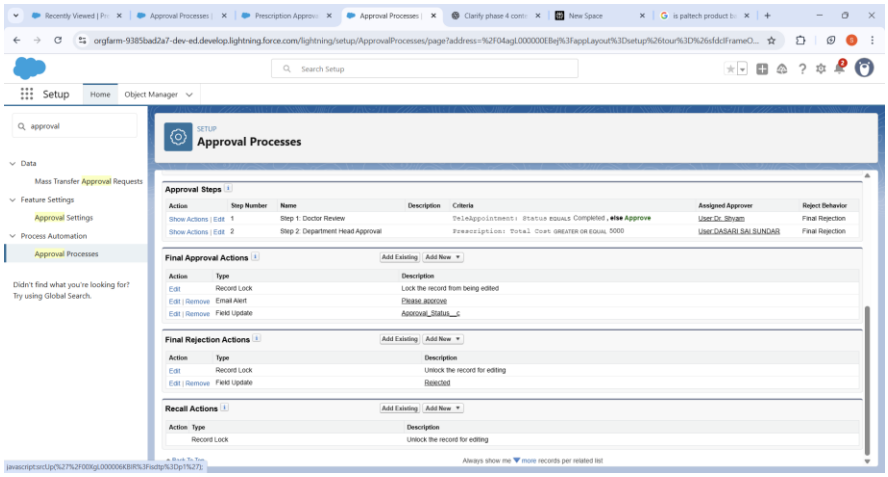
- Created the Please Approve



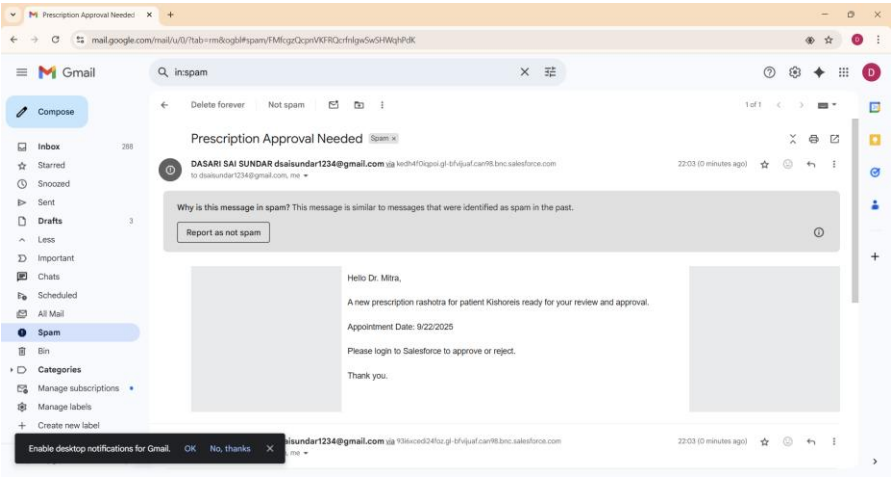
- A field update created in the Final Approval process



As same final Rejection also Created with the Rejected Named Field Update.



e) Finally check the Approval process in the doctor page and approved the new prescription is Directly sent to mail.

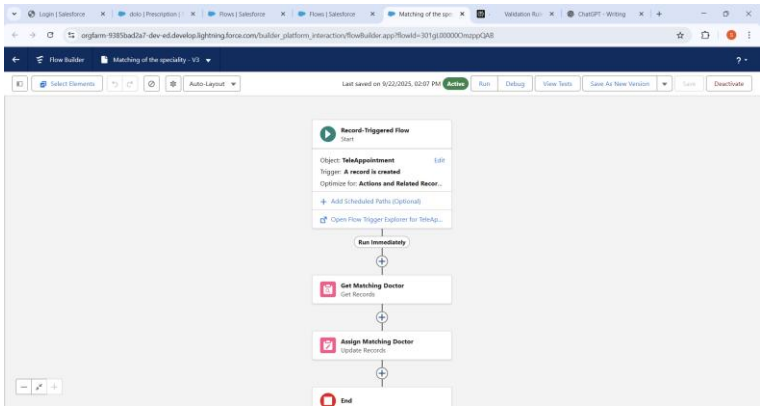


Approval Process

An Approval Process is used when you need to get formal permission before a record is finalized.

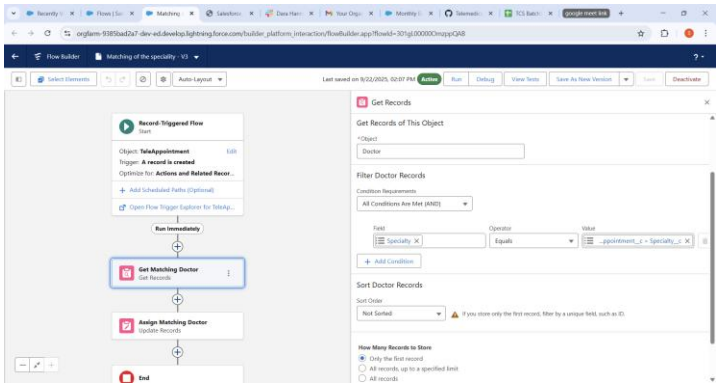
- **TM_Prescription_Approval**: This process is for approving prescriptions. It's triggered when a prescription is restricted or the total cost is over $5,000.
- The approval process has two steps: a "Doctor Review" and a "Department Head Approval."
- When a record is submitted for approval, an email is sent to the assigned approvers to notify them.

8.Flows:

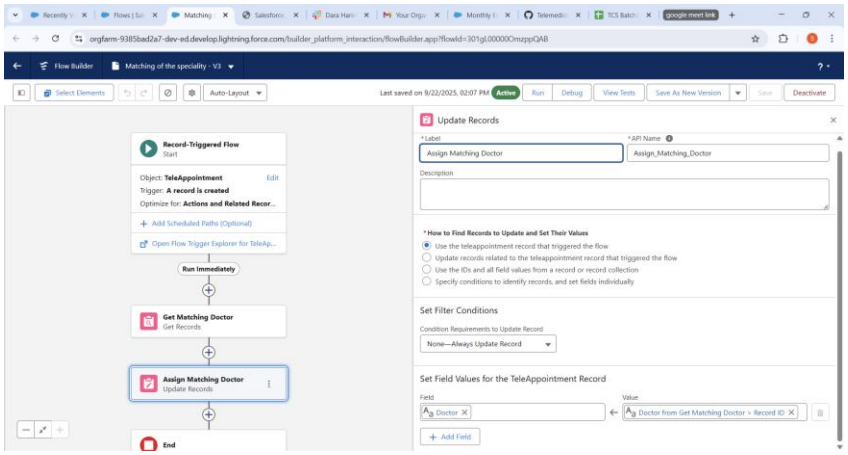


Flow is a visual automation tool that can perform everything a Process Builder or Workflow Rule can, and much more.

a) Selected the Get Record and used the given filter

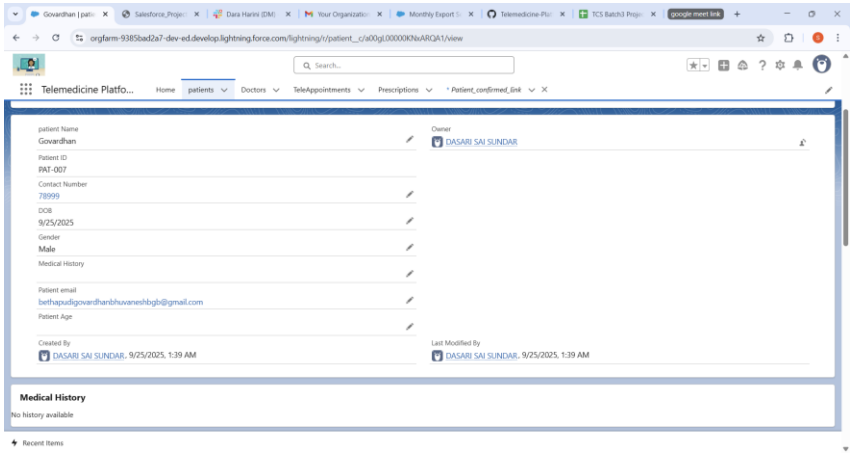


b) Update the record with the name Assigning Matching Doctor:

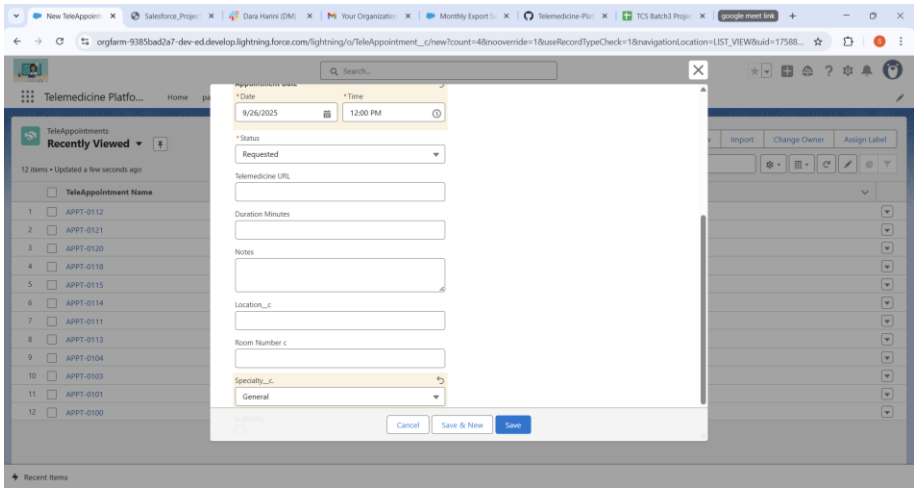


c) Test in the Platform:

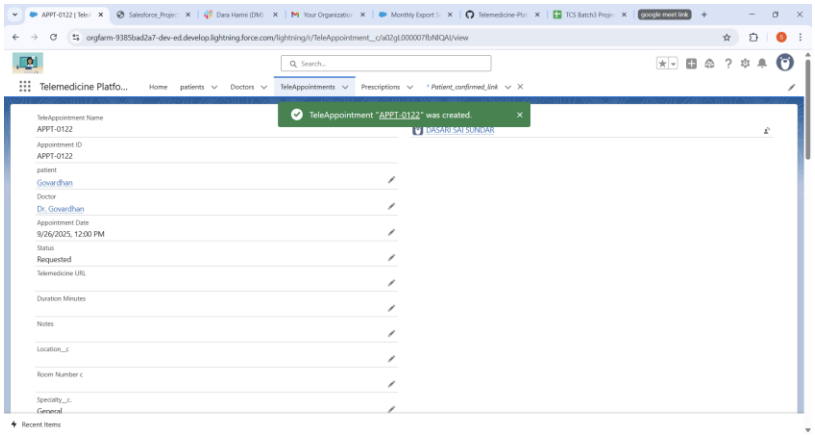
- Let us consider Govardhan



- Just selected the Specialization not doctor

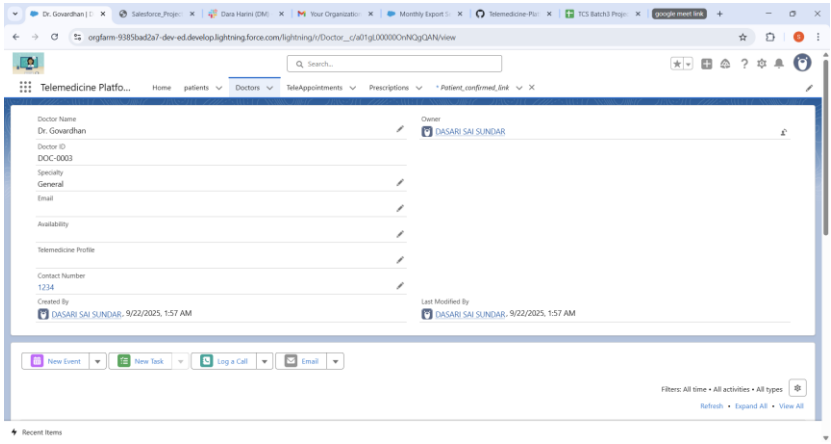


- After saving Directly Assigned



As we see there a general doctor Govardhan is assigned to him

Page of Dr. Govardhan



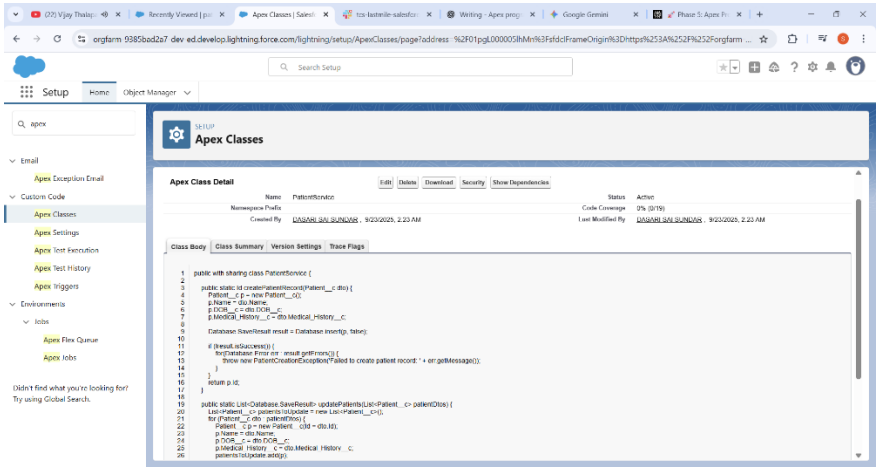
Record-Triggered Flow on the TeleAppointment object that runs when a new record is created.

- This Flow gets a matching doctor record from the database and assigns them to the appointment, which is a modern way to perform the same task as your Process Builder.

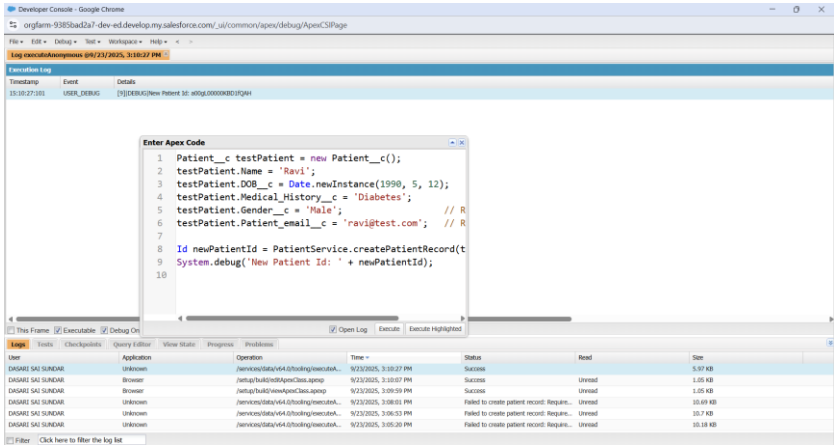
Process Phase 5: Apex Programming (Developer)(Admin)

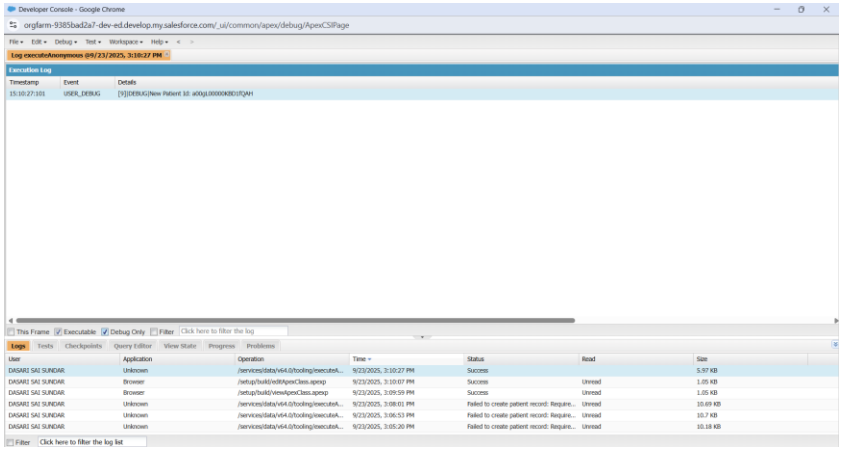
Phase-5:Apex Classes And triggers:

1.Apex classes:



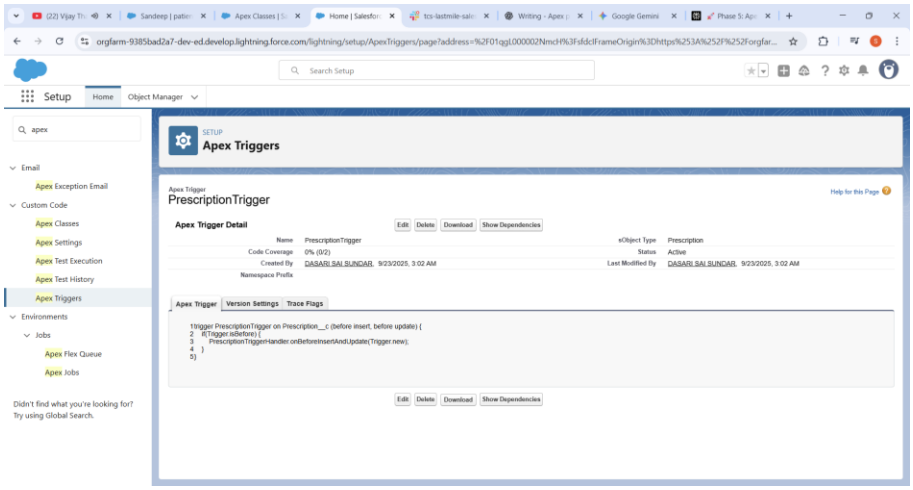
Apex Class →PatientService.cls Created



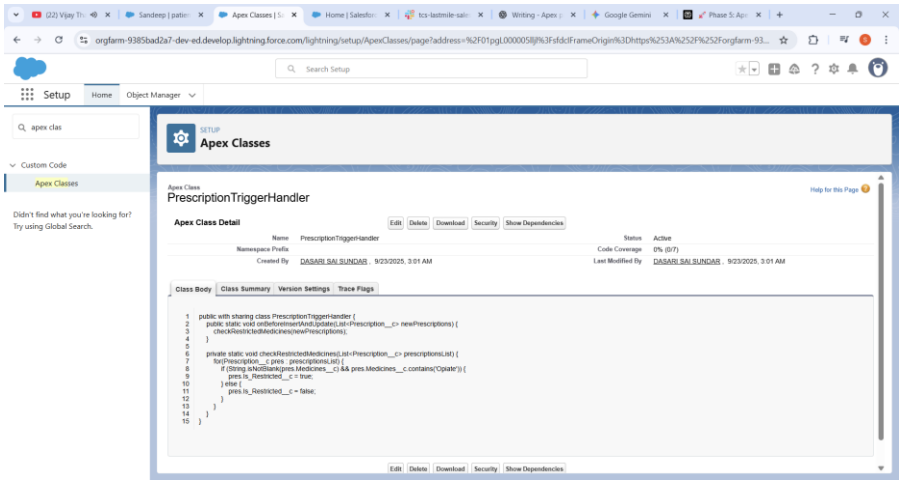


- **Create a new Patient__c record:**
  - Patient__c testPatient = new Patient__c();
  - Initializes a new instance of the custom object Patient__c.
- **Set basic patient details:**
  - Name, DOB__c, and Medical_History__c fields are assigned appropriate values.
  - These fields store the patient's name, date of birth, and medical history.
- **Provide all required fields:**
  - Gender__c and Patient_email__c are mandatory fields in Salesforce.
  - The code sets default values for these fields to ensure the record can be successfully created.
- **Insert the record using the service class:**
  - Id newPatientId = PatientService.createPatientRecord(testPatient);
  - The createPatientRecord method handles the database insert operation.
  - It also performs error handling: if the record cannot be inserted, a custom exception is thrown.
- **Log the new patient ID:**
  - System.debug('New Patient Id: ' + newPatientId);
  - Prints the Salesforce-generated record Id to the debug log, confirming the record was created successfully.

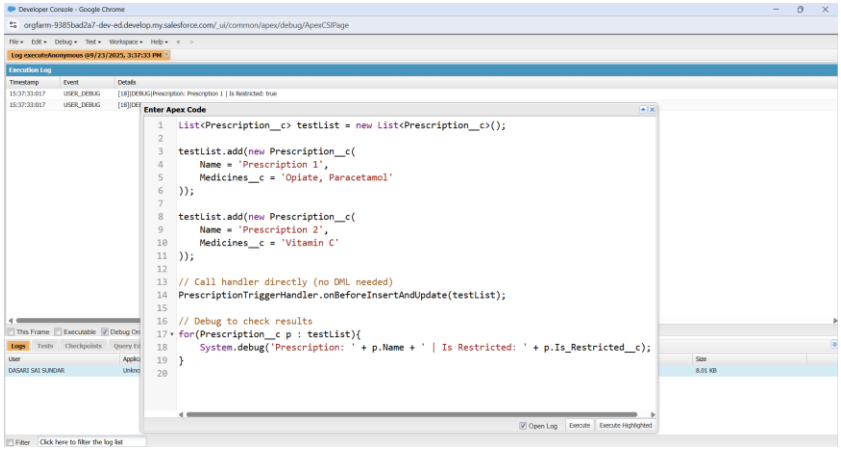
2.Apex Trigger and Handler:



Prescription Trigger→Apex Trigger for the prescription



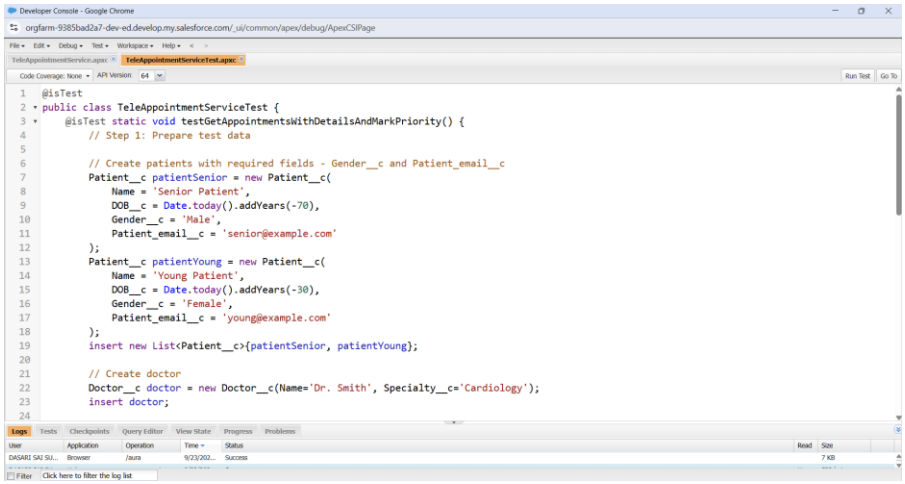
Prescription TriggerHandler → Apex class for the Trigger Handling



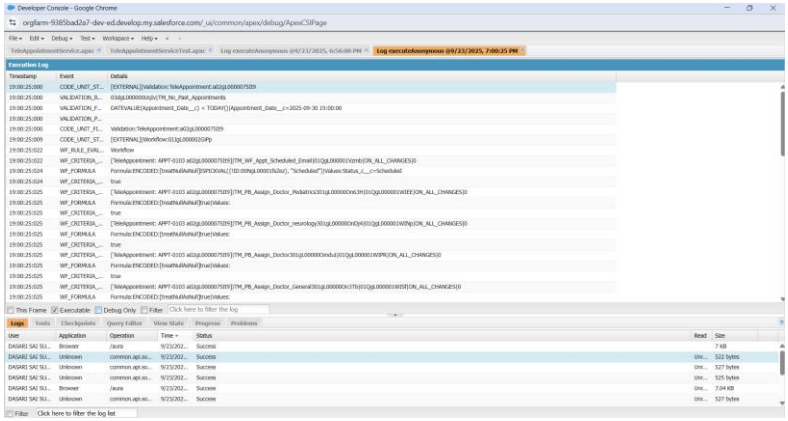
Apex triggers run automatically when records are inserted, updated, or deleted. They delegate logic to a handler class, which processes all records in Trigger.new safely in bulk. The handler applies business rules like marking restricted prescriptions before or after save. Unit tests verify the trigger works correctly for all scenarios.

- Trigger fires automatically when a Prescription__c record is inserted or updated.
- Trigger delegates control toPrescriptionTriggerHandler.onBeforeInsertAndUpdate(Trigger.new).
- Handler loops through each record in Trigger.new and checks the Medicines__c field.
- If the medicines contain 'Opiate', it sets Is_Restricted__c = true; otherwise, it sets it to false.
- Records are saved to the database with the updated Is_Restricted__c value

3.SOQL for the salesforce:

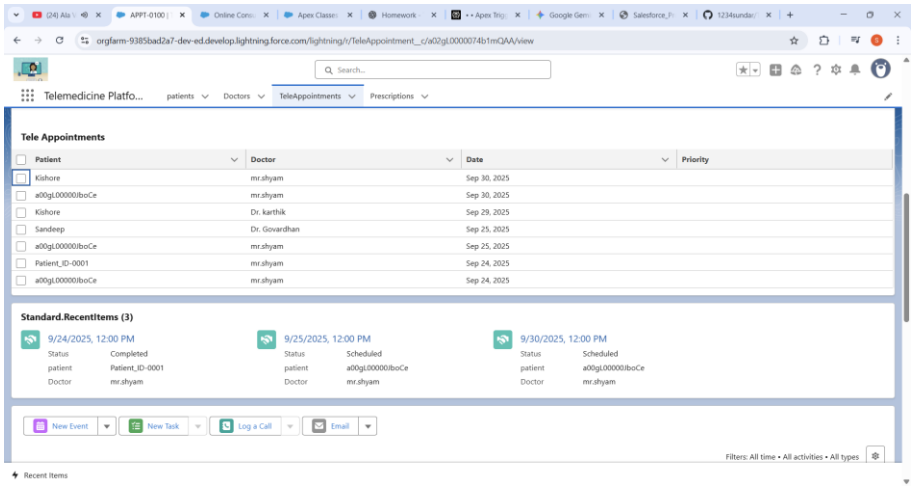


Teleappointment Service and Service test is created



1. Created test patients, doctors, and future appointments with all required info.
2. Queried appointments with related doctor and patient details for full context.
3. Calculated patient ages to flag senior patients' appointments as priority.
4. Updated appointments to mark priority status based on age condition.
5. Ran assertions verifying data integrity and priority logic worked perfectly in tests.

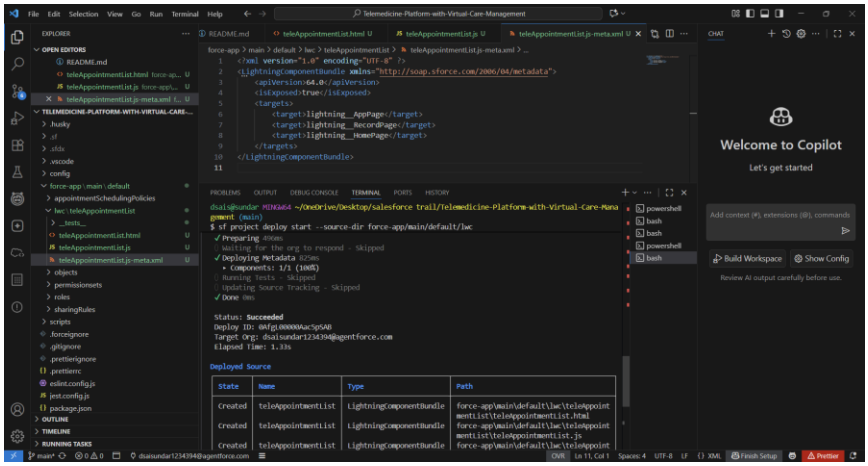
4. Lightning Web Component (LWC):



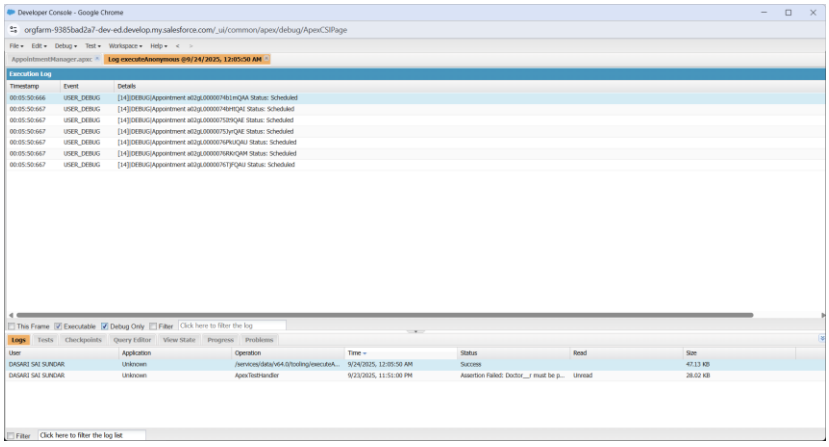
Lightning Web Component (LWC) is a dynamic dashboard for your appointments. It works by having three parts that connect and talk to each other.

1. The HTML: This file is the visual blueprint of your component. It uses a <lightning-datatable> to show the data on the screen.
2. The JavaScript: This is the logic that gets the data. It uses the @wire decorator to automatically call your Apex controller when the component loads.
3. The Apex Controller: This is the data fetcher. It runs a SOQL query on your database to get the latest appointment records and sends them back to the JavaScript.

your component loads on a page, the JavaScript tells the Apex controller to get the data, and the HTML displays it in a table for the user. It's an automated process that gives you a real-time view of your appointments.

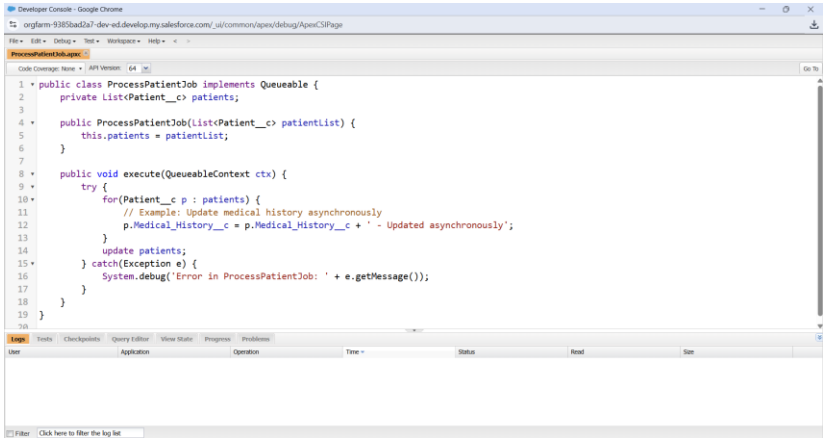


The lightning web Component is created by the help of the vscode and deploy using the vs code



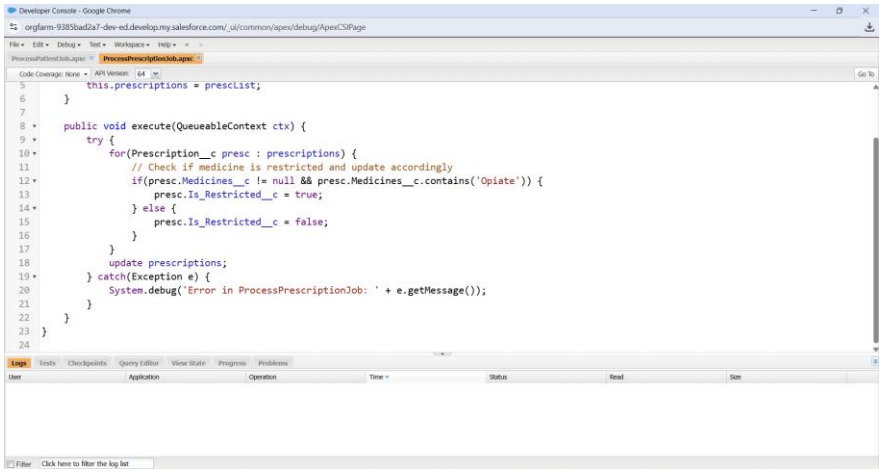
4. Asynchonous Apex:

a) Create a Queueable Apex Class with name: ProcessPatientJob



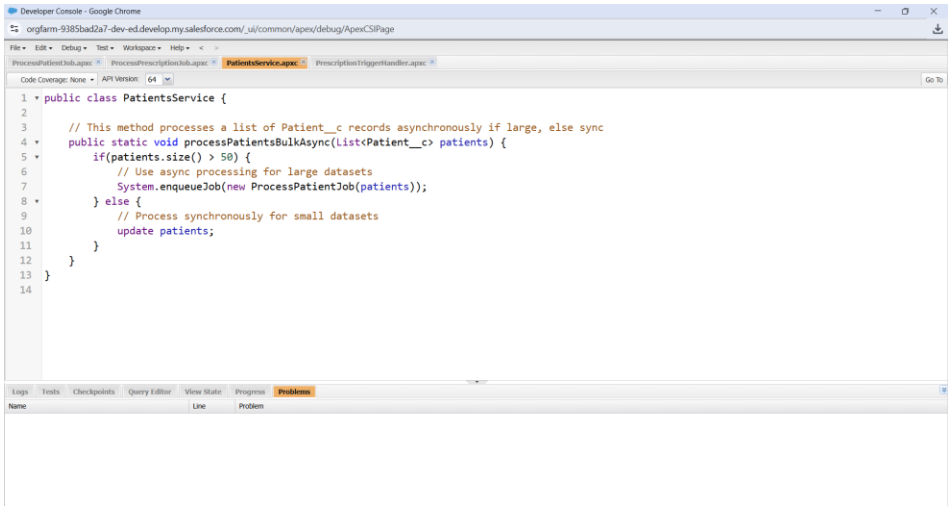
```
public class ProcessPatientJob implements Queueable {
    private List<Patient__c> patients;

    public ProcessPatientJob(List<Patient__c> patientList) {
        this.patients = patientList;
    }

    public void execute(QueueableContext ctx) {
        try {
            for(Patient__c p : patients) {
                // Example: Update medical history asynchronously
                p.Medical_History__c = p.Medical_History__c + ' - Updated asynchronously';
            }
            update patients;
        } catch(Exception e) {
            System.debug('Error in ProcessPatientJob: ' + e.getMessage());
        }
    }
}
```

b) Create a Queueable Apex Class with name: ProcessPrescription



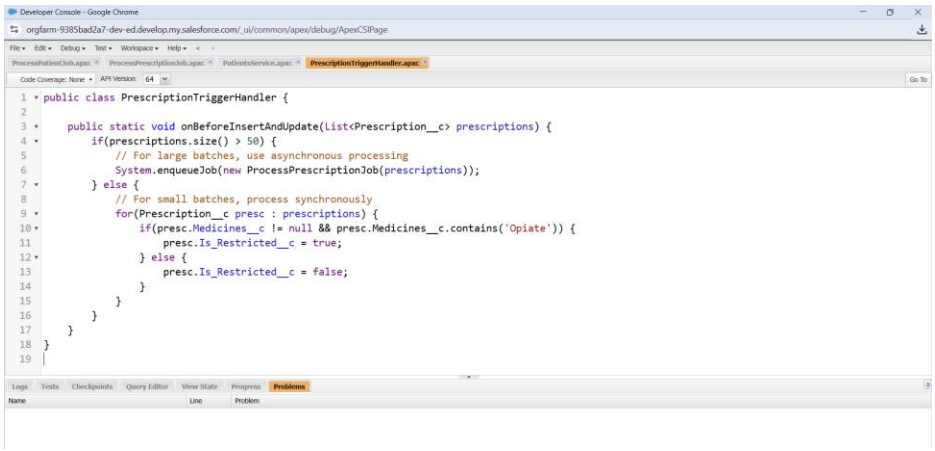
```
        this.prescriptions = prescList;
    }

    public void execute(QueueableContext ctx) {
        try {
            for(Prescription__c presc : prescriptions) {
                // Check if medicine is restricted and update accordingly
                if(presc.Medicines__c != null && presc.Medicines__c.contains('Opiate')) {
                    presc.Is_Restricted__c = true;
                } else {
                    presc.Is_Restricted__c = false;
                }
            }
            update prescriptions;
        } catch(Exception e) {
            System.debug('Error in ProcessPrescriptionJob: ' + e.getMessage());
        }
    }
}
```

c) Modify The Existing PatientService Class:



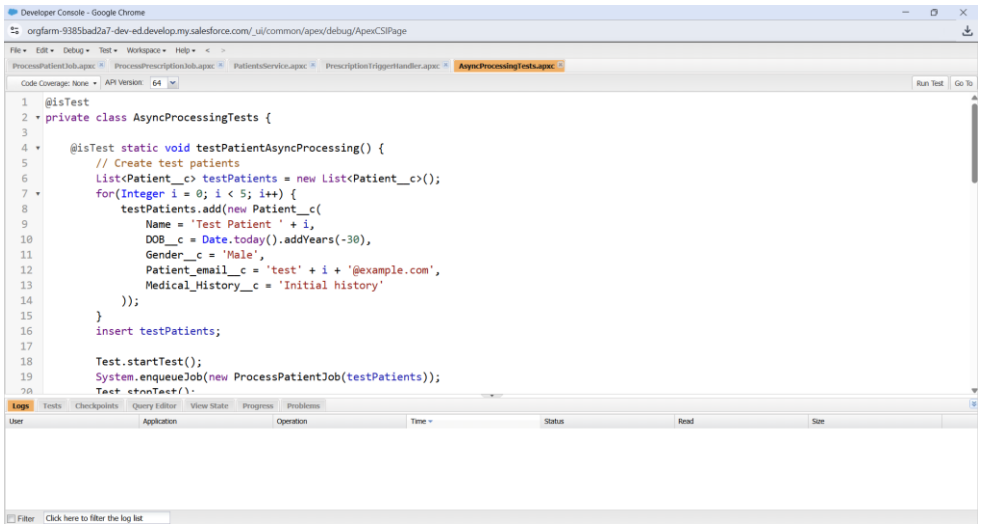
```
public class PatientsService {

    // This method processes a list of Patient__c records asynchronously if large, else sync
    public static void processPatientsBulkAsync(List<Patient__c> patients) {
        if(patients.size() > 50) {
            // Use async processing for large datasets
            System.enqueueJob(new ProcessPatientJob(patients));
        } else {
            // Process synchronously for small datasets
            update patients;
        }
    }
}
```

d) Modify Your PrescriptionTriggerHandler:



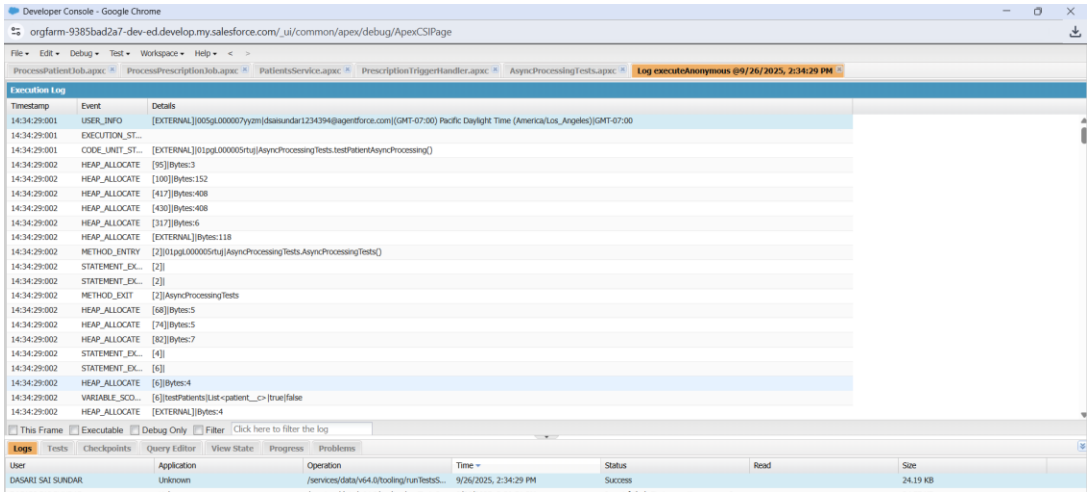
```
public class PrescriptionTriggerHandler {

    public static void onBeforeInsertAndUpdate(List<Prescription__c> prescriptions) {
        if(prescriptions.size() > 50) {
            // For large batches, use asynchronous processing
            System.enqueueJob(new ProcessPrescriptionJob(prescriptions));
        } else {
            // For small batches, process synchronously
            for(Prescription__c presc : prescriptions) {
                if(presc.Medicines__c != null && presc.Medicines__c.contains('Opiate')) {
                    presc.Is_Restricted__c = true;
                } else {
                    presc.Is_Restricted__c = false;
                }
            }
        }
    }
}
```

e) Create Test Classes with the AsyncProcessingTests:



```
@isTest
private class AsyncProcessingTests {

    @isTest static void testPatientAsyncProcessing() {
        // Create test patients
        List<Patient__c> testPatients = new List<Patient__c>();
        for(Integer i = 0; i < 5; i++) {
            testPatients.add(new Patient__c(
                Name = 'Test Patient ' + i,
                DOB__c = Date.today().addYears(-30),
                Gender__c = 'Male',
                Patient_email__c = 'test' + i + '@example.com',
                Medical_History__c = 'Initial history'
            ));
        }
        insert testPatients;

        Test.startTest();
        System.enqueueJob(new ProcessPatientJob(testPatients));
        Test.stopTest();
```

Finally run the selected with the AsyncProcessingTest



The asynchronous Apex implementation you requested is complete with these components:

- The Queueable Apex classes (ProcessPatientJob and ProcessPrescriptionJob) for running async logic.
- The service method in PatientsService managing sync vs async processing (processPatientsBulkAsync).
- The trigger handler method (onBeforeInsertAndUpdate in PrescriptionTriggerHandler) using asynchronous processing for bulk handling.
- The test class (AsyncProcessingTests) to validate asynchronous processing works correctly.

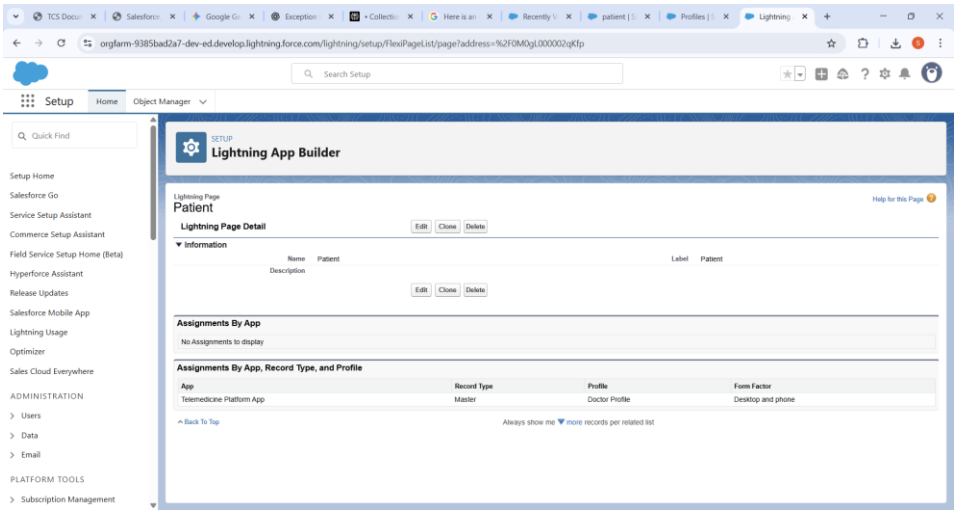
Together, these fit the best practices for asynchronous Apex in your Salesforce project and have been tailored specifically for your custom objects (Patient__c and Prescription__c).

Phase 6: User Interface Development

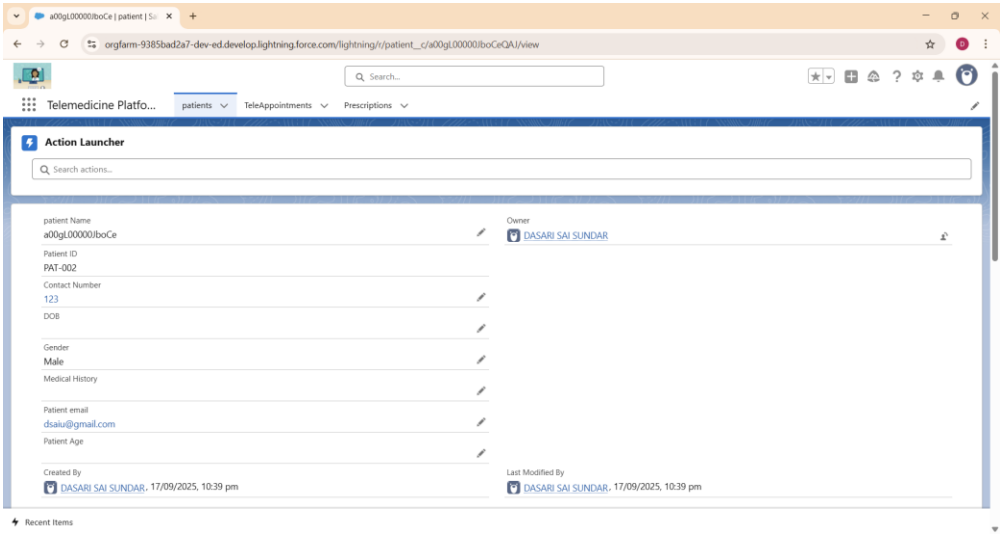
User interface Process

1. Create Custom Lightning Record Pages:

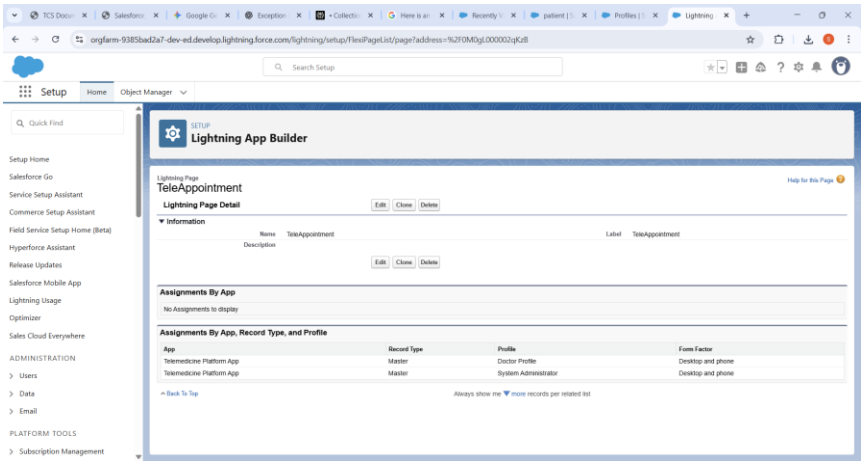
Create the patient lightning record pages:



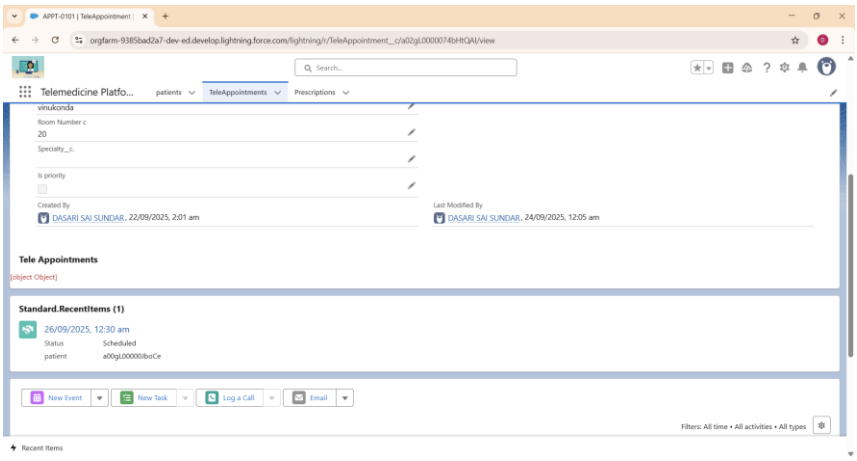
Check at the doctor Profile:



Create the TeleAppointment lightning record pages:



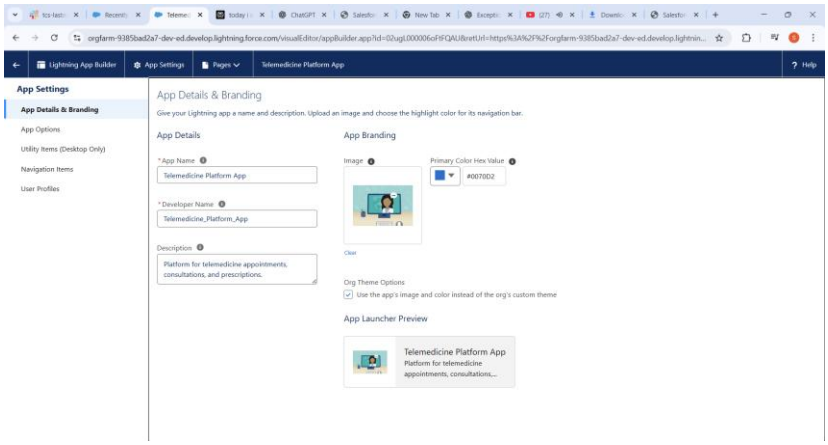
Check at the doctor Profile:



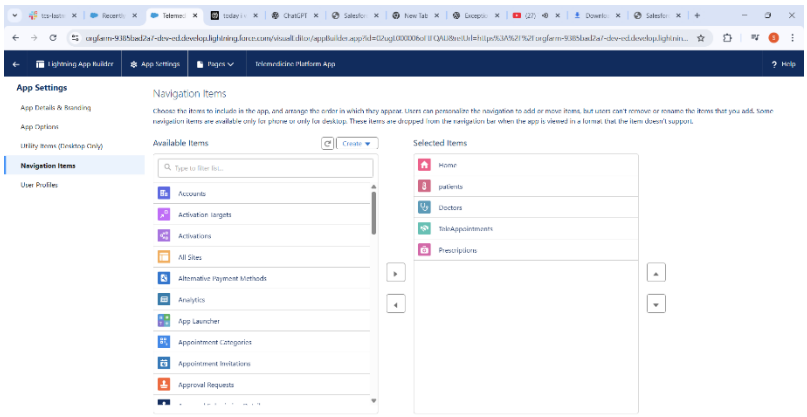
The Patient Lightning Record Page and TeleAppointment Lightning Record Page are customized to display and manage crucial patient and appointment details respectively within the Telemedicine Platform. These pages are assigned to the Doctor Profile to ensure doctors have seamless access to relevant patient information and appointment management tools on both desktop and mobile. This setup enhances the efficiency of medical consultations and telehealth service delivery.

2.Create Your Lightning App Framework:

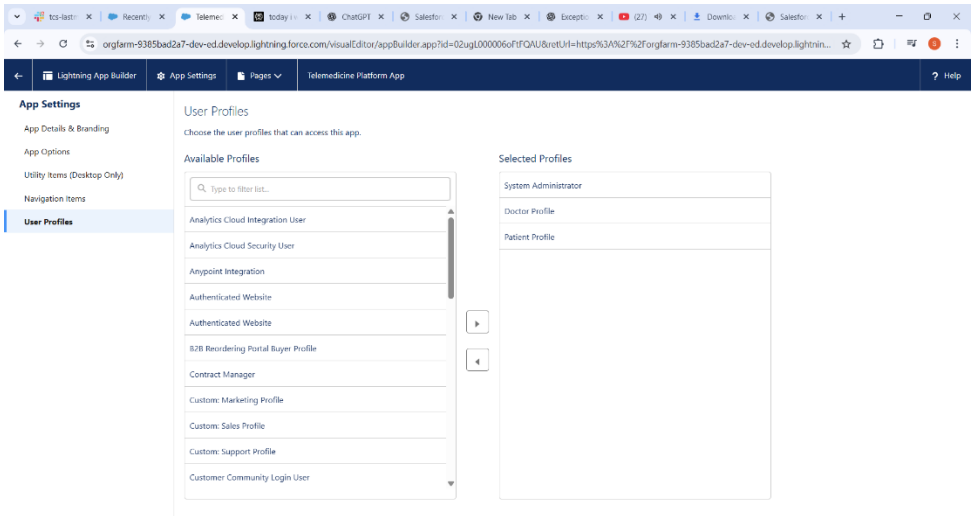
Created The Lightning App:



The Navigation Items Selected:

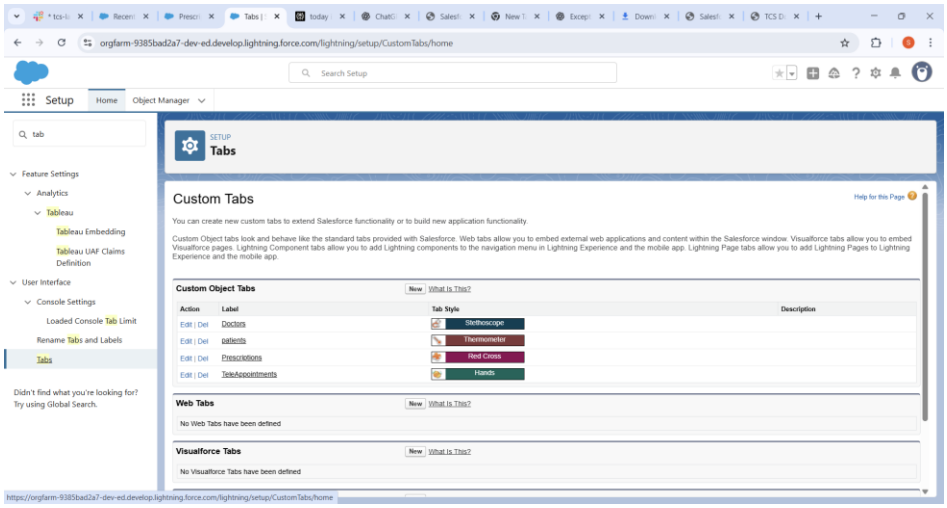


User Profiles Have Selected: As I Have 3 Main Users I have selected Three.



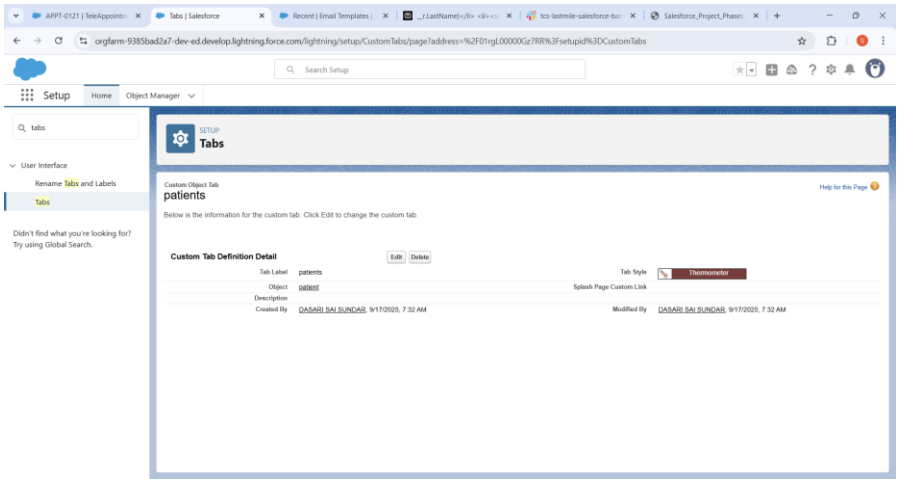
The Lightning App Framework for the Telemedicine Platform App is set up with custom branding, a description, and a highlight color for easy recognition. Key navigation items—Home, Patients, Doctors, TeleAppointments, and Prescriptions—are added to streamline access to core features. Specific profiles such as System Administrator, Doctor Profile, and Patient Profile are granted access, ensuring each user role can interact with the app as required.

3. Tabs:

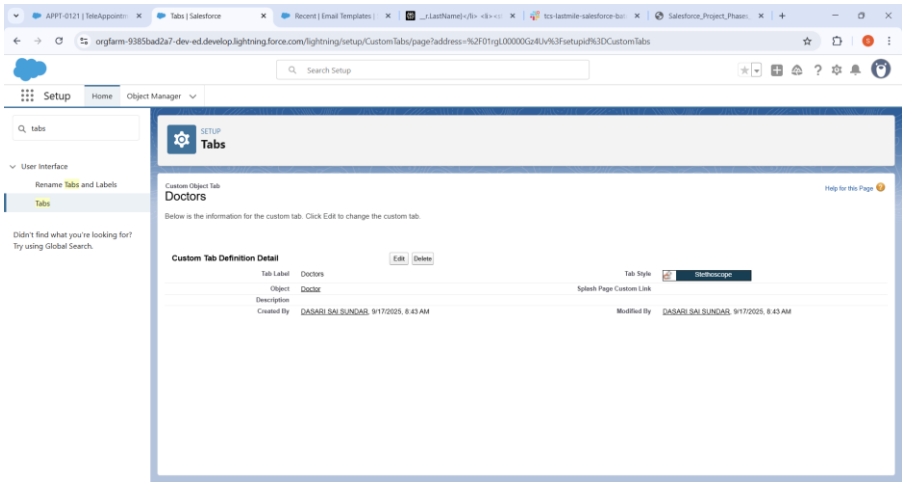


These tabs help users quickly access relevant records and features in the telemedicine platform with distinct icons for better visual identification.

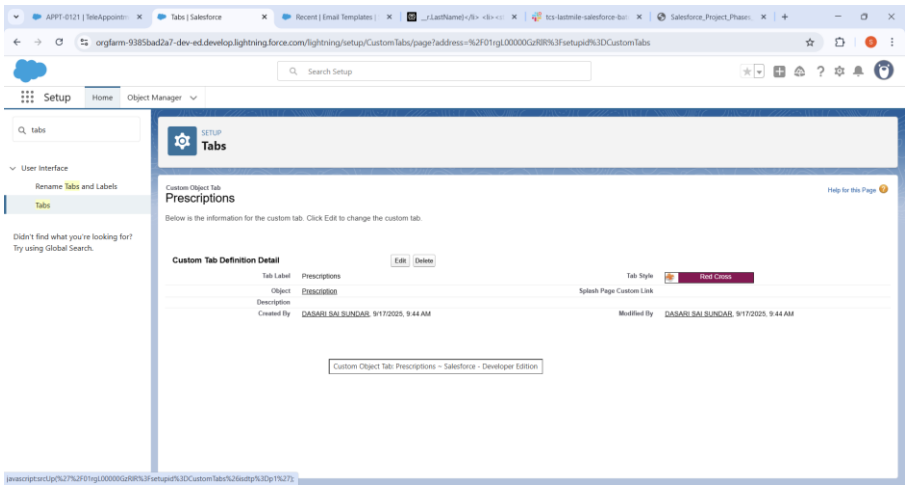
Patient Tab Created:



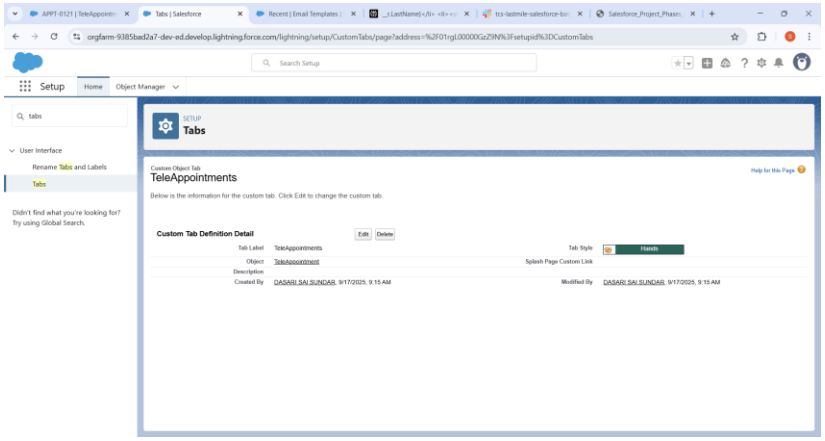
Doctor Tab Created:



Prescription Tab created:



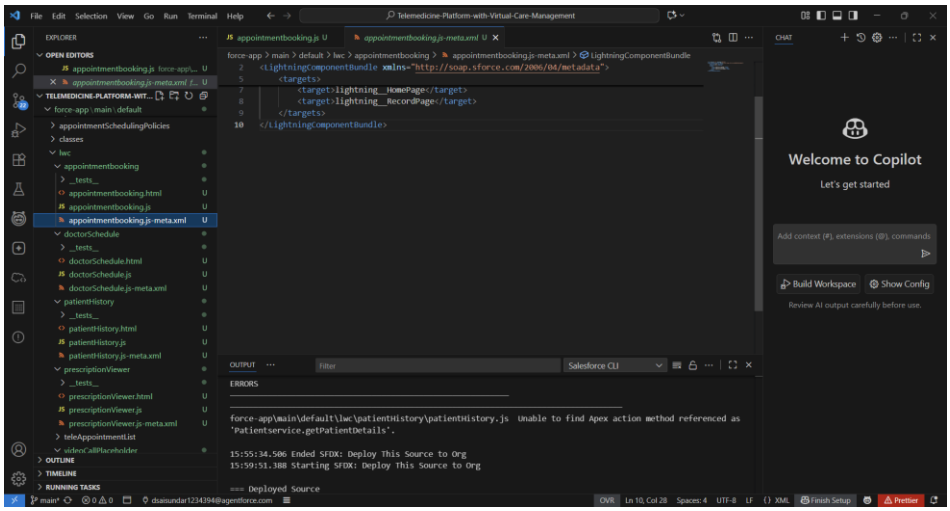
TeleAppointment Tab Created:

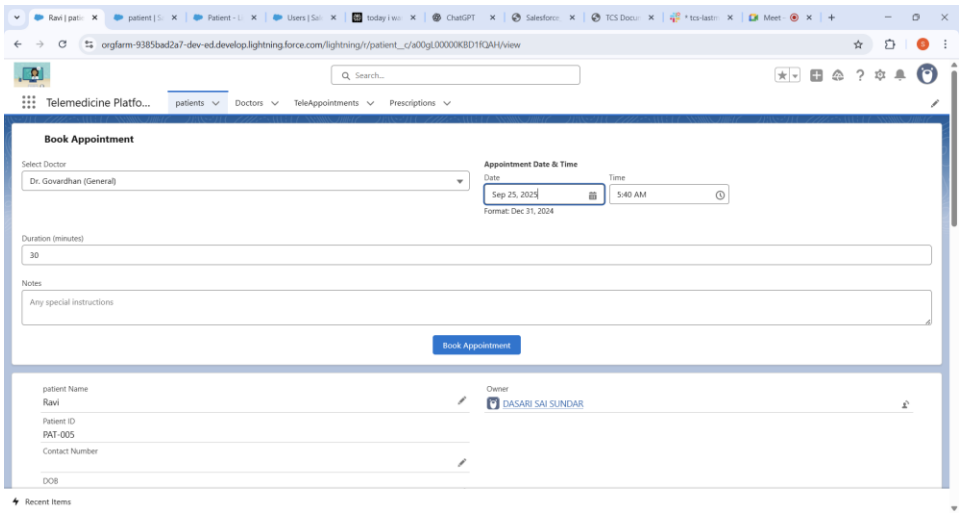


The created Tabs are Patients, Doctors, TeleAppointment, Prescriptions for easy navigation within Salesforce. These tabs help users quickly access relevant records and features in the telemedicine platform with distinct icons for better visual identification. This setup ensures streamlined workflow and improved user experience.

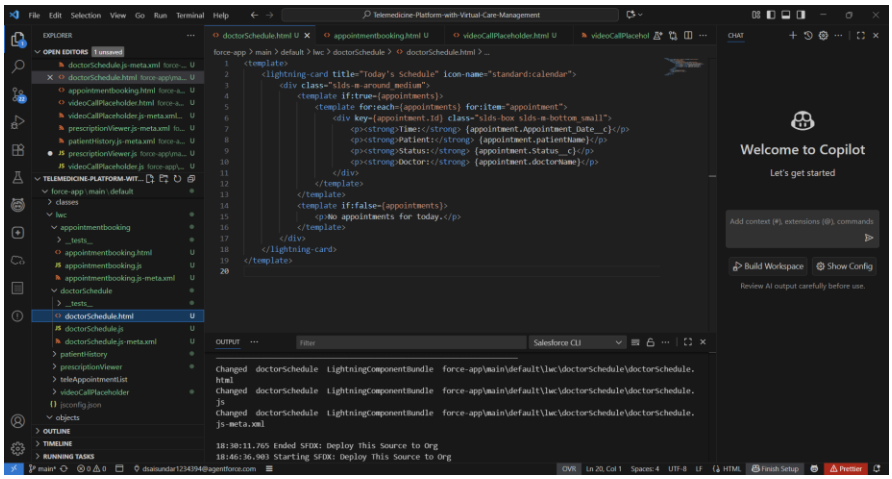
4.Lightning Web Component:

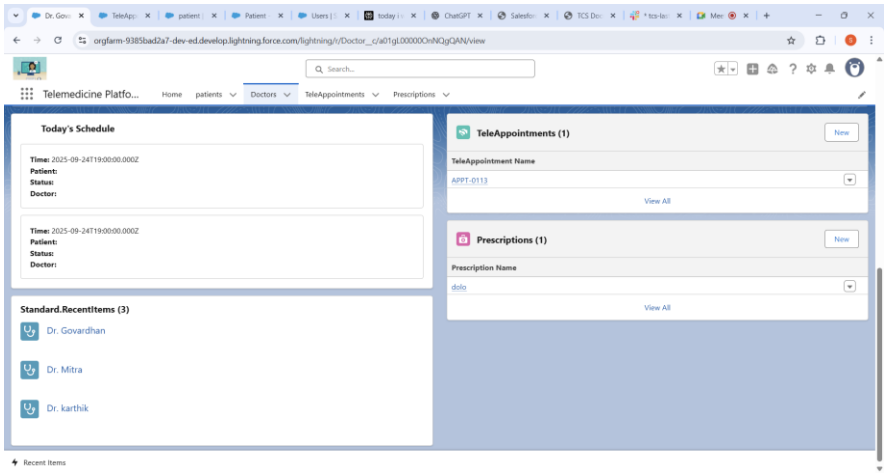
Different lwc are created like appointment booking , Doctor Schedule, Patient history, Prescription Viewer.





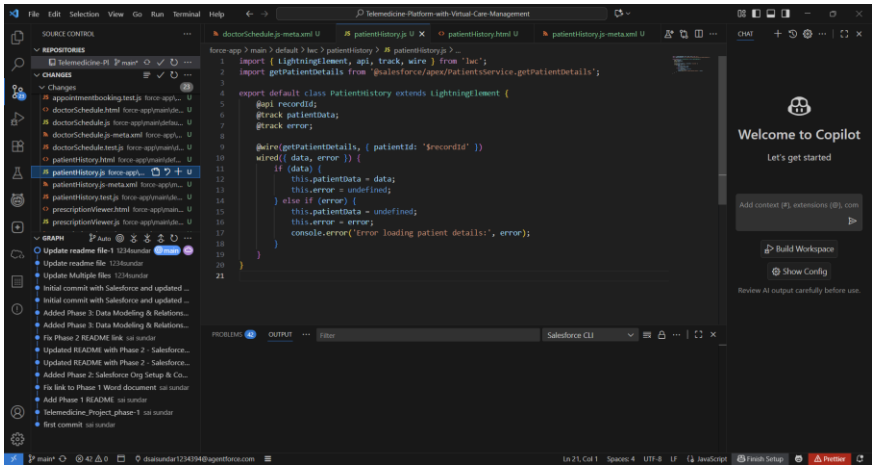
This is creating the Book Appointment to the Patient accordingly.

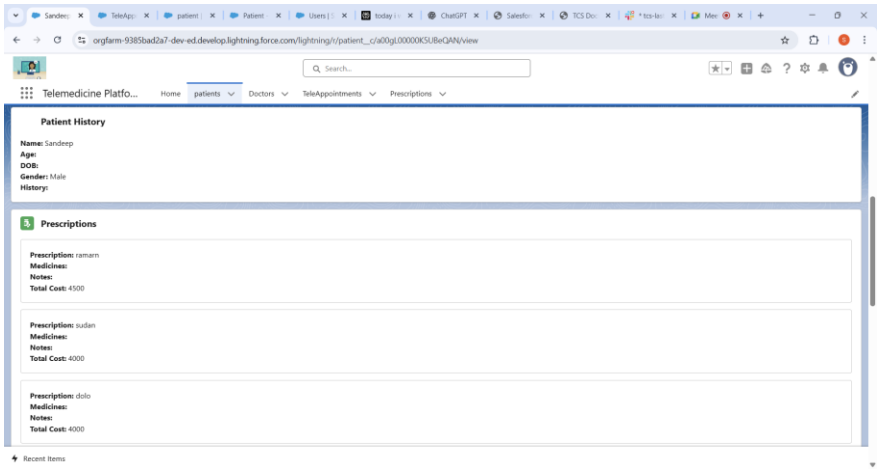




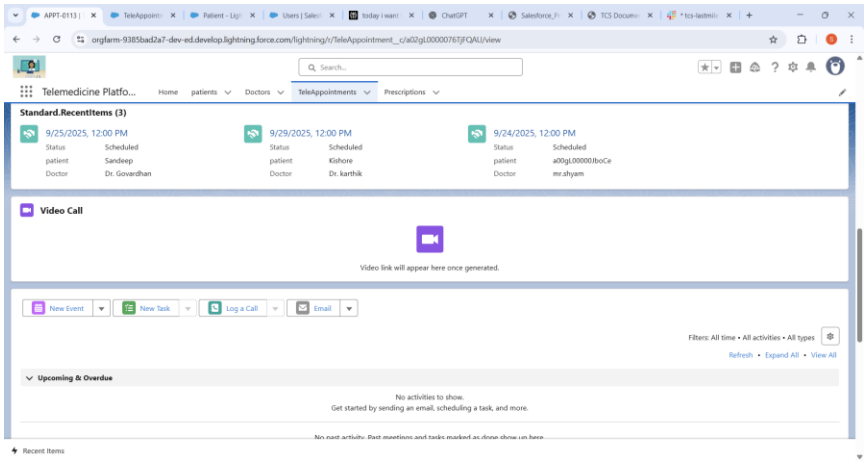
Doctor Schedule is created as he can see how many schedules are there that day.

Created the PatientHistory LWC:





In the above image we can see the patient is displaying

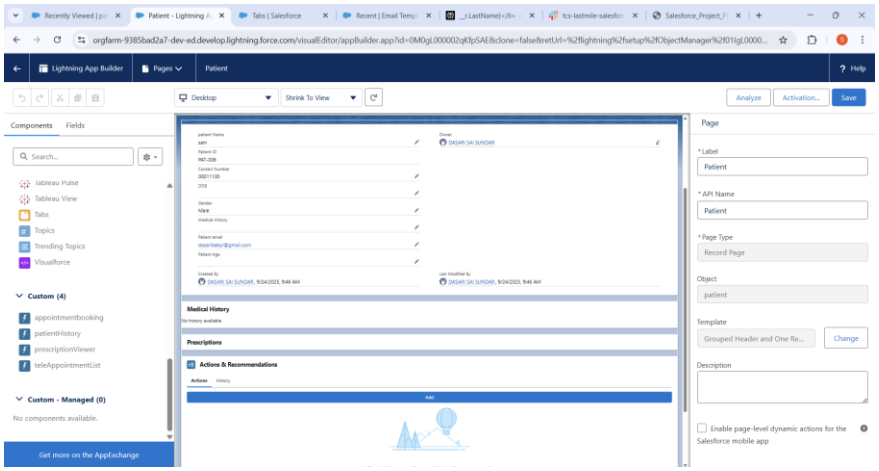


Google meet Video Call lwc is created as we go to phase the 7 we can do Background work of link generation

This is the patient Object Record Page layout with the custom Lwc's like the patient history and the prescription.

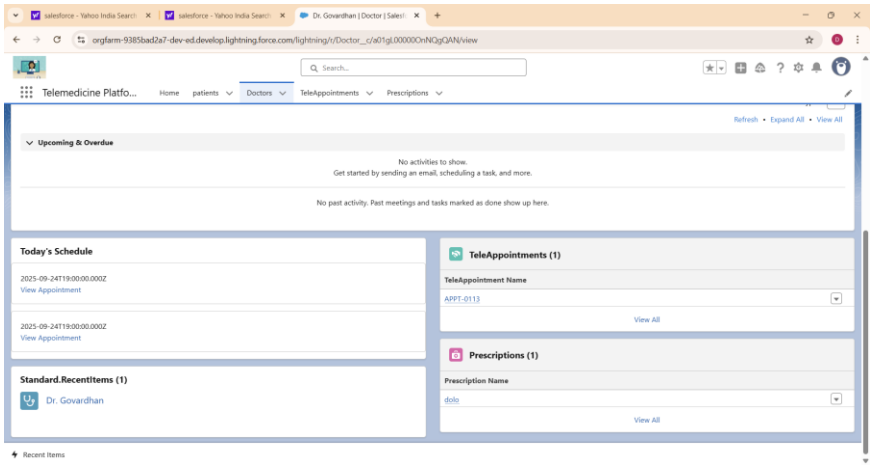
Lightning Web Components (LWCs) like appointment booking, doctor schedule, patient history, and prescription viewer enable streamlined telemedicine workflows by providing interactive and real-time data access for patients and doctors. Integrated features such as Google Meet video call support enhance virtual consultations, while the custom LWCs deployed on patient record pages deliver a unified, efficient healthcare management interface.

5.lightning Record Page:

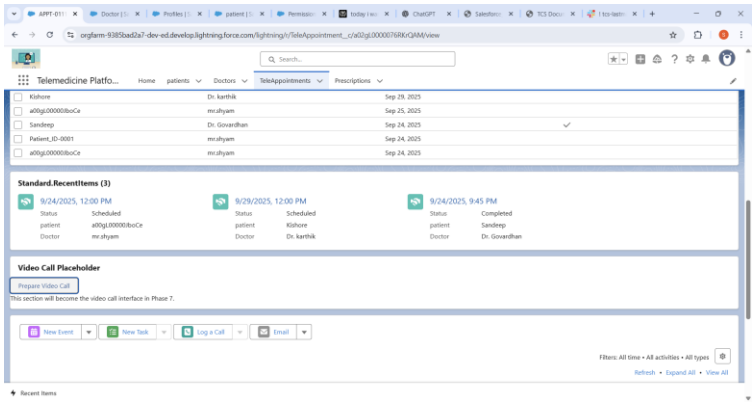


From the above we can see the Appopintbooking ,PatientHistory, Prescription Viewer and the Placevideocall custom lightning apps.We just drag them into the lightning page.

6.Navigation Buttons:



By clicking the view appointment can directly takes Doctor to the Teleappointment.



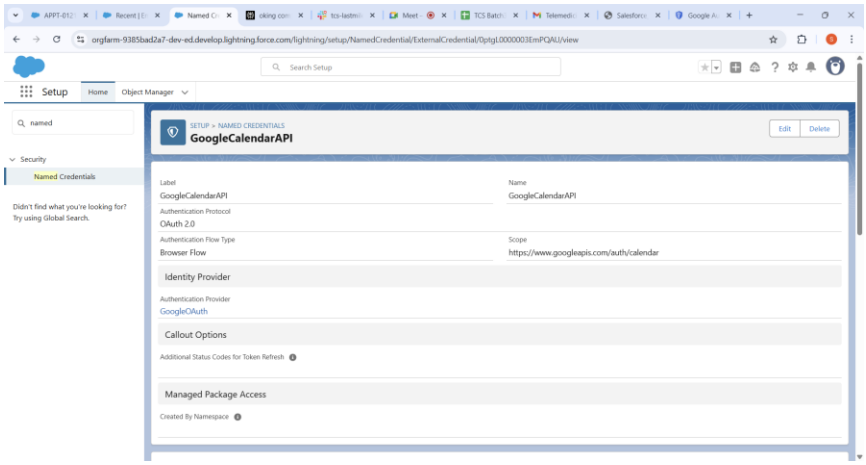
By clicking the Prepare video we can directly goes to the google meet with the patient

Navigation buttons in the telemedicine platform enhance usability by allowing doctors to instantly access detailed appointment information and initiate video consultations via Google Meet with a single click. This seamless integration improves workflow efficiency and ensures timely interactions between doctors and patients.

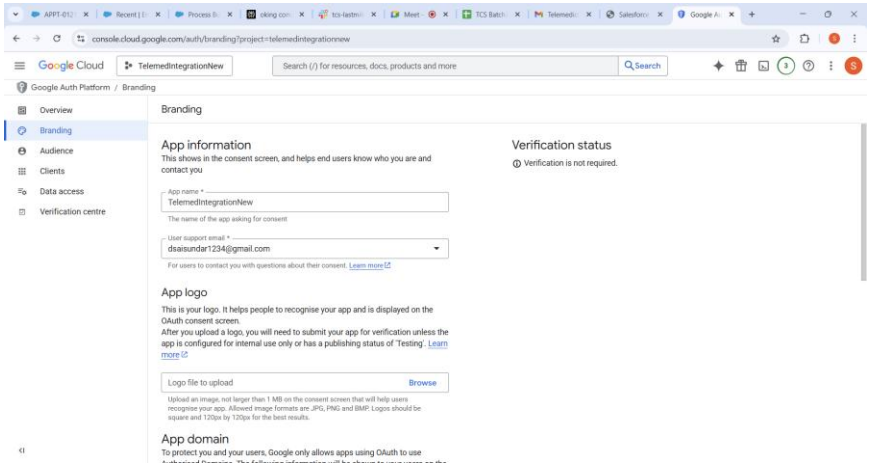
Phase 7: Integration & External Access

Phase7- Integration Access Google meet Link formation and process flows

1.Named Credentials:

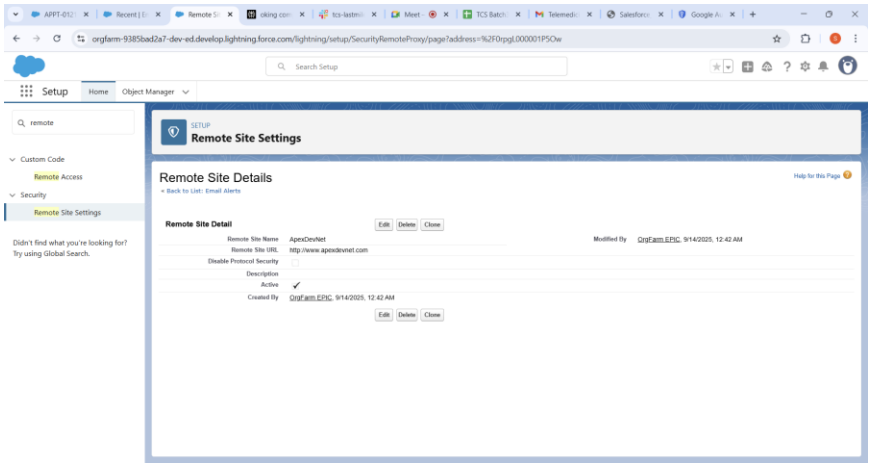


Created the GoogleAPI in the Named Credential for the access for the GoogleCloud



Named Credential configured in Salesforce to authenticate and securely connect to the Google Calendar API. The Named Credential includes the endpoint URL, OAuth 2.0 settings, and references the Google Auth Provider for seamless token management.

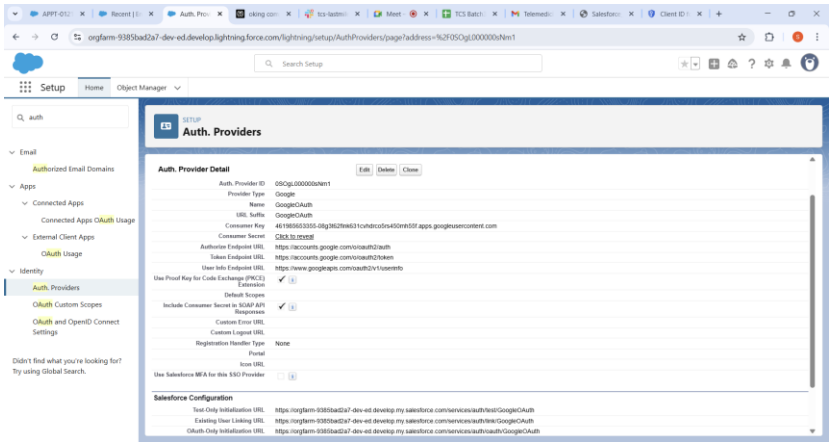
2.**Remote Site Settings:**



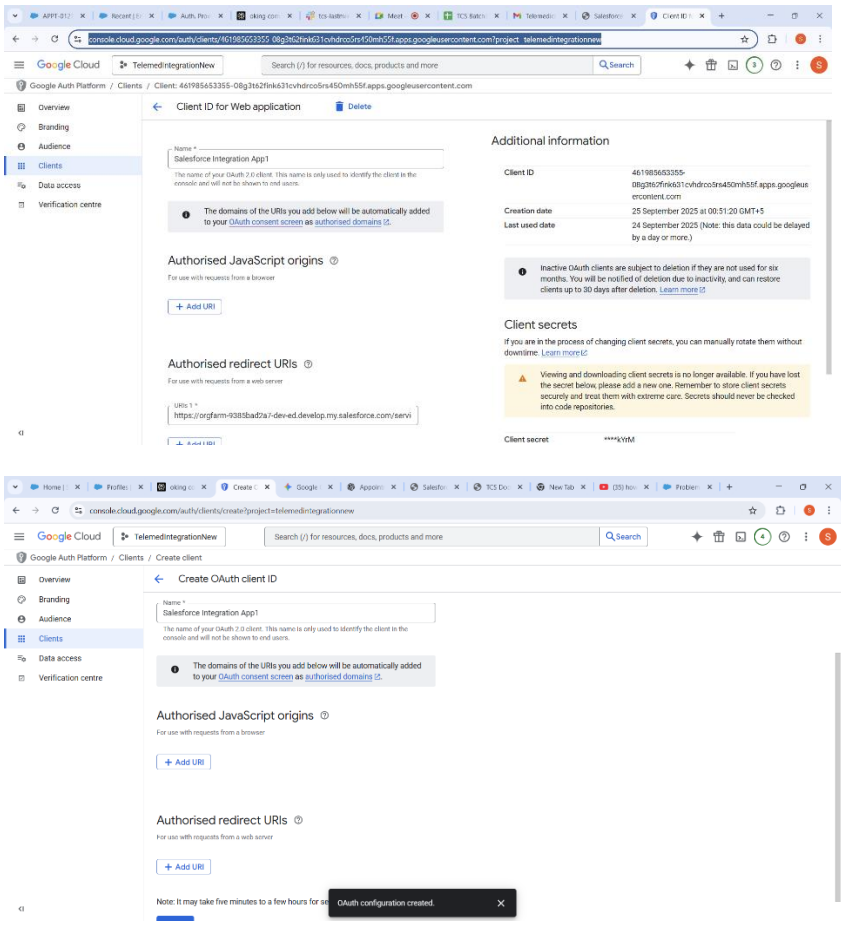
This image lists the Remote Site Settings currently registered in Salesforce, including the Google API URL. This setting allows Salesforce Apex code to make HTTP callouts to external endpoints like Google's calendar API, ensuring secure communication without Salesforce callout restrictions.

- List page showing all configured remote sites
- The remote site URL you added to allow callouts

3.**OAuth & Authentication Provider:**



Here is the Authentication Provider setup screen showing the Google OAuth 2.0 configuration.

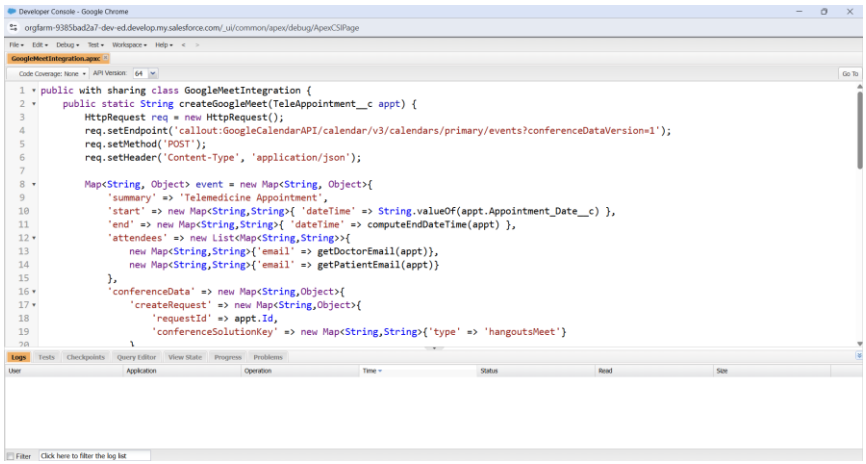


This provider facilitates OAuth authentication between Salesforce and Google, specifying Client ID, Client Secret, and the Salesforce callback URLs to handle OAuth tokens for accessing Google services.

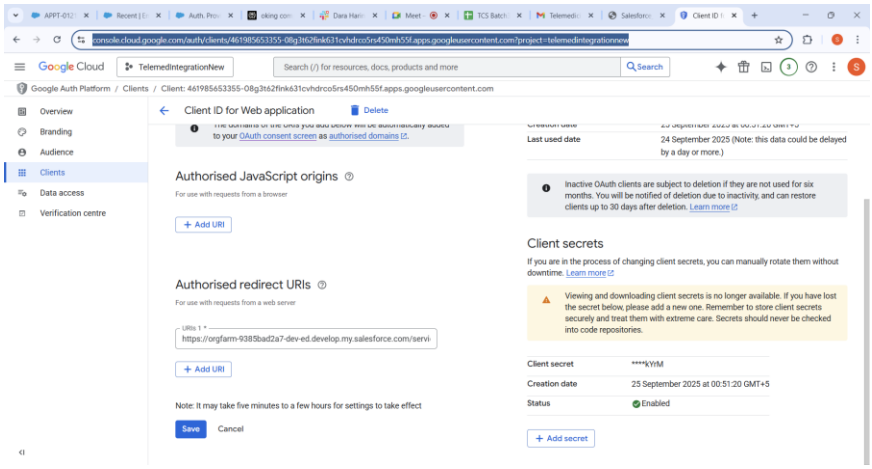
4.Apex Callout Code:

```
public with sharing class GoogleMeetIntegration {

    public static String createGoogleMeet(TeleAppointment__c appt) {

        HttpRequest req = new HttpRequest();
req.setEndpoint('callout:GoogleCalendarAPI/calendar/v3/calendars/primary/events?conferenceDataVersion=1');

        req.setMethod('POST');

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

        Map<String, Object> event = new Map<String, Object>{

            'summary' => 'Telemedicine Appointment',

            'start' => new Map<String,String>{ 'dateTime' => String.valueOf(appt.Appointment_Date__c) },
```

```
            'end' => new Map<String,String>{ 'dateTime' => computeEndDateTime(appt) },
            'attendees' => new List<Map<String,String>>{
                new Map<String,String>{'email' => getDoctorEmail(appt)},
                new Map<String,String>{'email' => getPatientEmail(appt)}
            },
#Some more Code is there just not added
    }
    private static String computeEndDateTime(TeleAppointment__c appt) {
        if(appt.Appointment_Date__c == null) return null;
        Integer duration = appt.Duration_Minutes__c != null ? appt.Duration_Minutes__c.intValue() : 30;
        return appt.Appointment_Date__c.addMinutes(duration).formatGMT('yyyy-MM-
dd\'T\'HH:mm:ss\'Z\'');
    }
    private static String getDoctorEmail(TeleAppointment__c appt) {
        if(appt.Doctor__c == null) return null;
        User doctorUser = [SELECT Email FROM User WHERE Id = :appt.Doctor__c LIMIT 1];
        return doctorUser != null ? doctorUser.Email : null;
    }
    private static String getPatientEmail(TeleAppointment__c appt) {
        if(appt.patient__c == null) return null;
        Patient__c patient = [SELECT Patient_Email__c FROM Patient__c WHERE Id = :appt.patient__c
LIMIT 1];
        return patient != null ? patient.Patient_Email__c : null;
    }
}
```

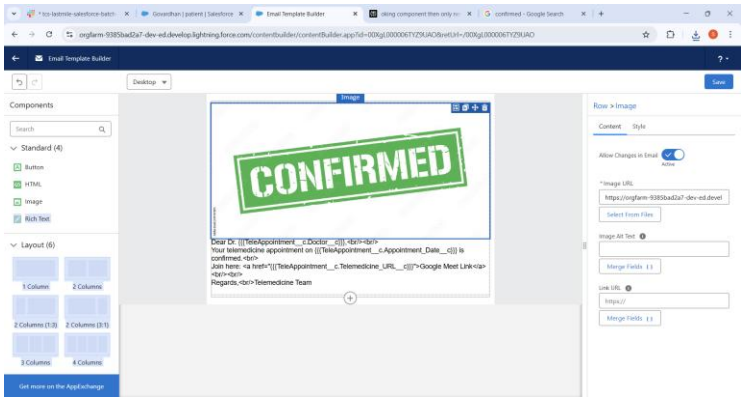


```
public with sharing class GoogleMeetIntegration {
    public static String createGoogleMeet(TeleAppointment__c appt) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:GoogleCalendarAPI/calendar/v3/calendars/primary/events?conferenceDataVersion=1');
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/json');

        Map<String, Object> event = new Map<String, Object>{
            'summary' => 'Telemedicine Appointment',
            'start' => new Map<String,String>{ 'dateTime' => String.valueOf(appt.Appointment_Date__c) },
            'end' => new Map<String,String>{ 'dateTime' => computeEndDateTime(appt) },
            'attendees' => new List<Map<String,String>>{
                new Map<String,String>{'email' => getDoctorEmail(appt)},
                new Map<String,String>{'email' => getPatientEmail(appt)}
            },
            'conferenceData' => new Map<String,Object>{
                'createRequest' => new Map<String,Object>{
                    'requestId' => appt.Id,
                    'conferenceSolutionKey' => new Map<String,String>{'type' => 'hangoutsMeet'}
```

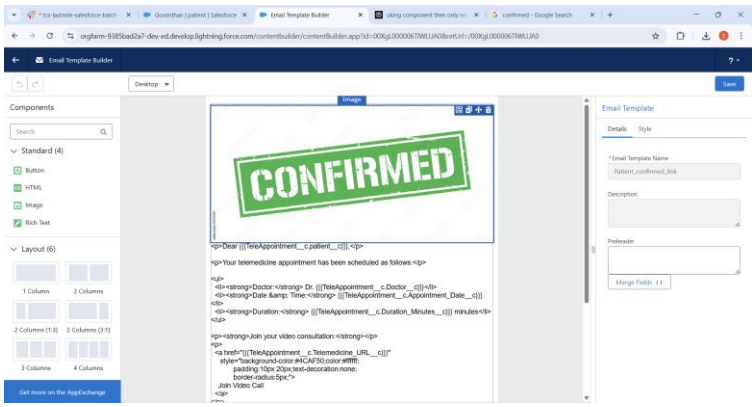


The screenshot captures the Salesforce Apex class method responsible for creating Google Meet links via HTTP REST callouts to the Google Calendar API. It demonstrates setting request endpoints, headers, body payloads, and processing JSON responses within Salesforce.

5.Email Template for confirming the Meeting and Give the meet link:



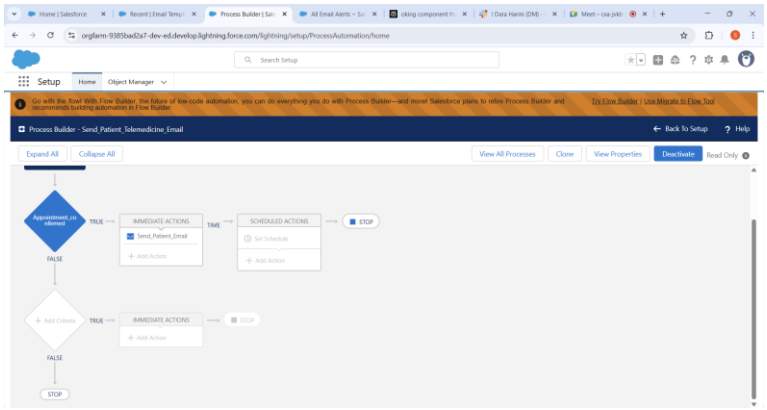
This is For the doctor for appointment is confirmed successfully and link is send successfully



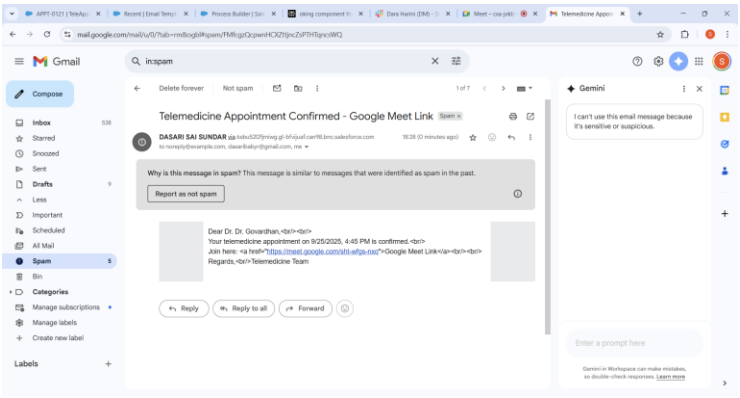
This is For the Patient for appointment is confirmed successfully and Google meet link is send successfully

This section displays the Lightning Email Template design used to send appointment confirmation emails containing Google Meet links to patients and doctors. The template utilizes merge fields from the TeleAppointment custom object for personalized content.

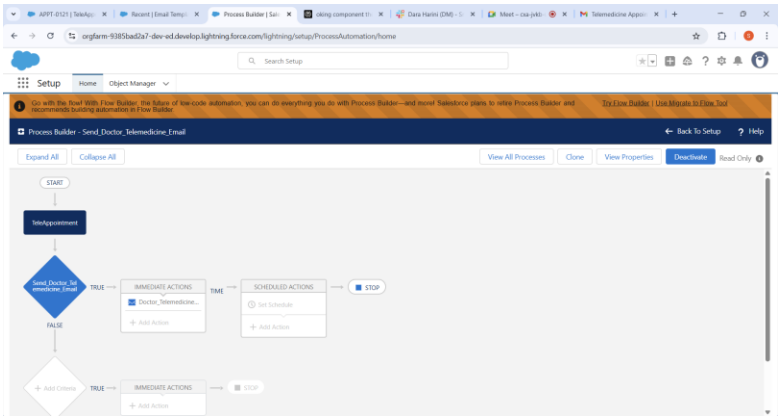
6.Process Builder Screenshot



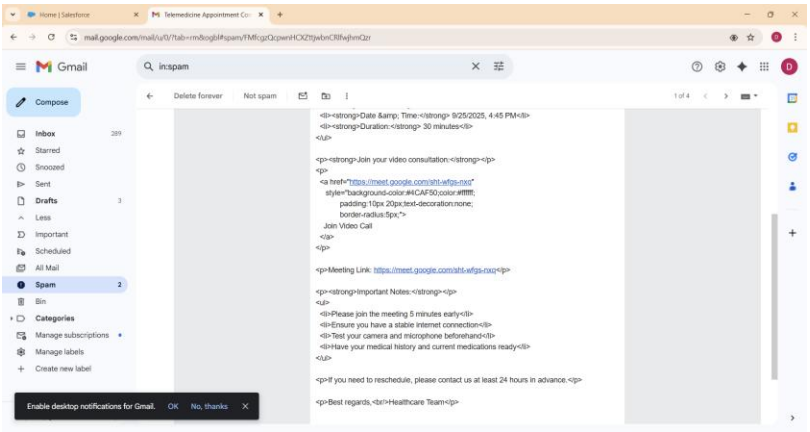
Process Builder is created for the patient link.



Received the mail at the patient email box

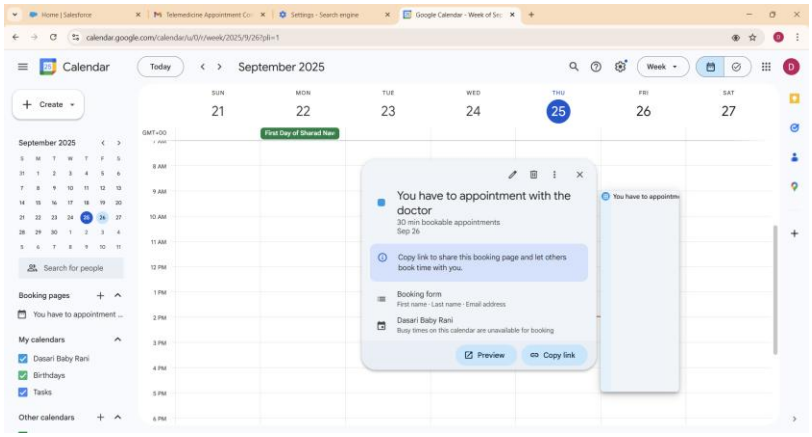


Process Builder is created for the Doctor link.



Received the email in the Doctor mail

This shows the Process Builder configuration used to automate sending of confirmation emails. It monitors the TeleAppointment object for status changes to 'Confirmed' and presence of the Google Meet link, triggering an email alert action to notify the corresponding user.



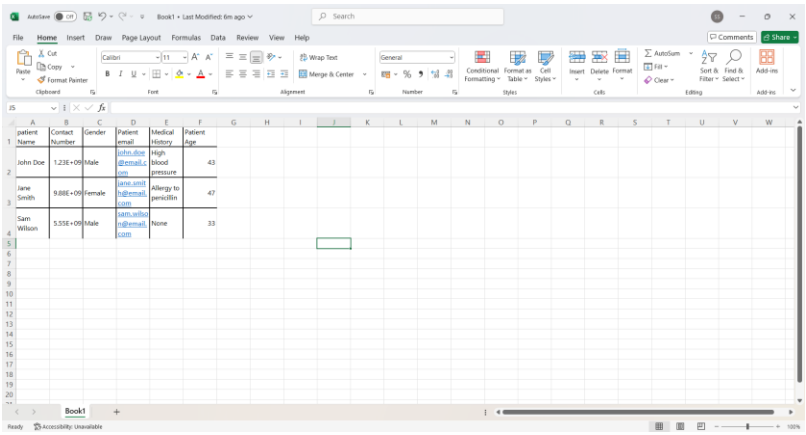
Google Calendar integration is successfully implemented using Named Credentials, OAuth authentication, and Apex REST callouts. Remote Site Settings ensure secure API access. Automated email notifications are triggered via Process Builder using Lightning Email Templates to notify users with Google Meet links. The system is now ready to streamline telemedicine appointments with seamless video conferencing features.

Phase 8: Data Management & Deployment

Phase 8: Data Management & Deployment:

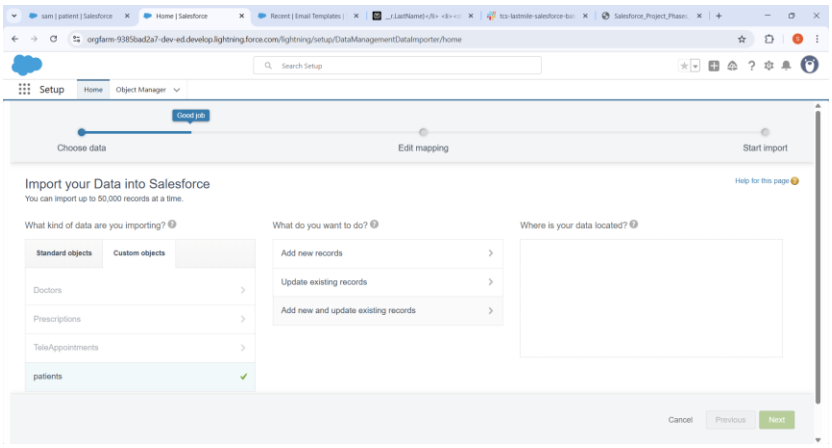
1.Data Import Wizard:

a) Created the Data of a Patient in the Word and save it in the .csv

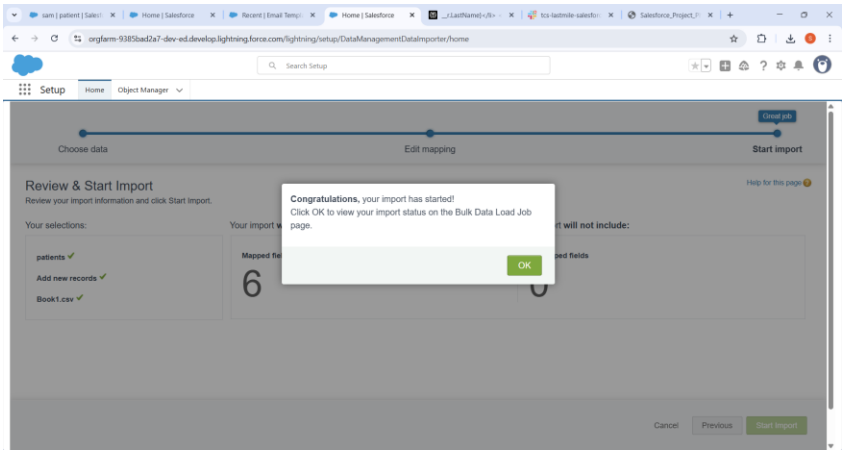


b) Create and Updating the setting in the Import Data

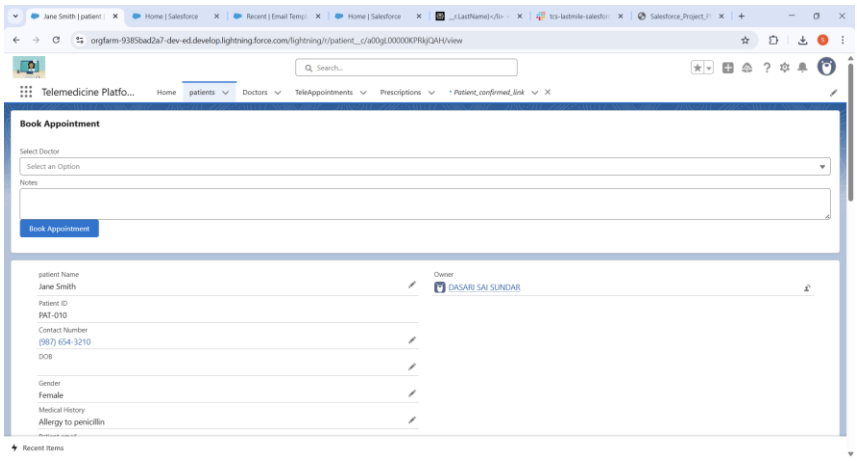
- With select the Patient as the source object→Locate our .csv



c) Successfully saved the Import function



d) Test1-Success The data Fetched to the Telemedical platform to the Patient

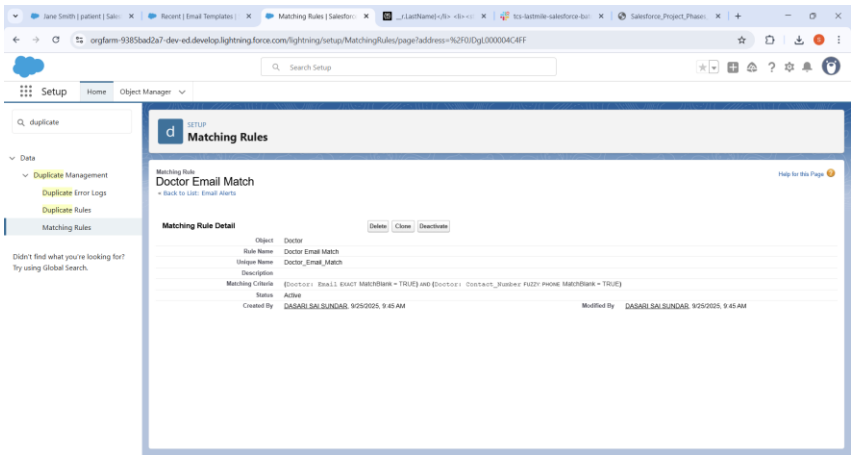


- Used Salesforce Data Import Wizard to upload patient information like names, contact numbers, date of birth, gender, email, medical history, and patient age from a well-prepared CSV file.
- Mapped CSV columns to corresponding Salesforce Patient fields to ensure correct data placement.
- Imported data as new patient records while preventing duplicates using matching and duplicate rules based on unique patient identifiers such as email and contact number.
- Validated the imported data by reviewing patient records for completeness and accuracy within Salesforce.
- Leveraged the imported patient data in appointment scheduling, teleconsultation workflows, and reporting modules to enable efficient telemedicine operations.

2. Duplicate Rules:

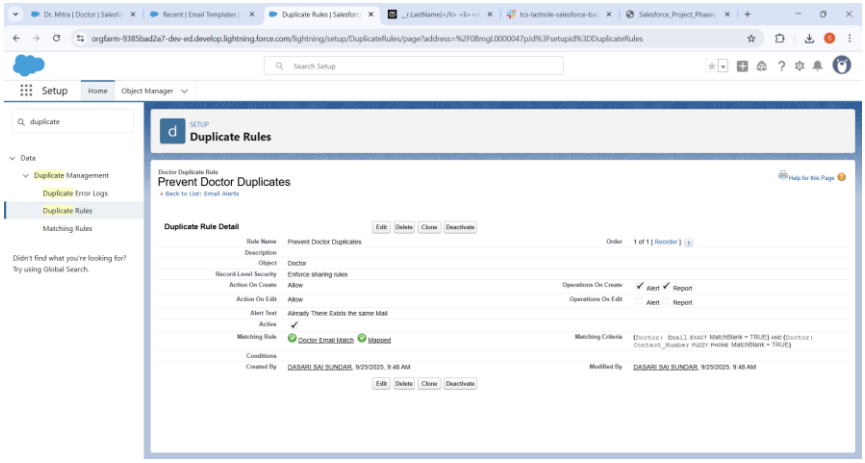
a) Matching Rules:

Created the Matching rule with Email.



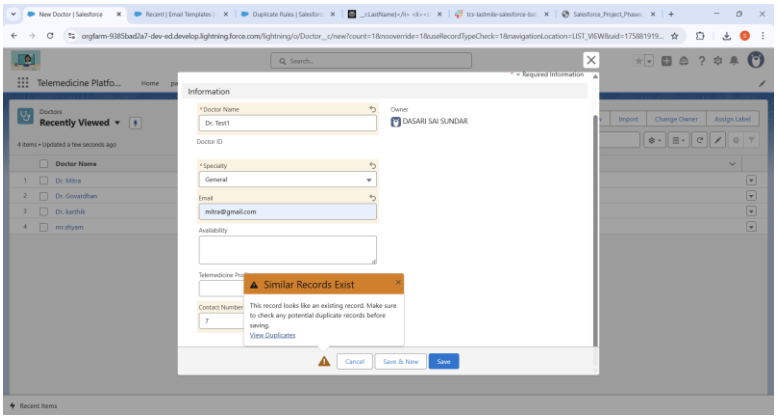
b) Duplicates Rules:

Created the duplicate rule with the above Matching rule.



c) Test case1:

Tested with same email but found The Error Arises same record Exists

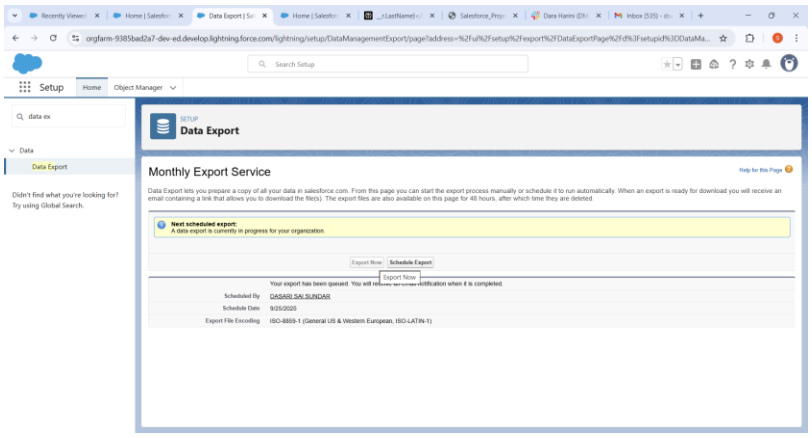


Duplicate Management in Salesforce is vital to ensure patient and doctor data remains accurate and reliable.

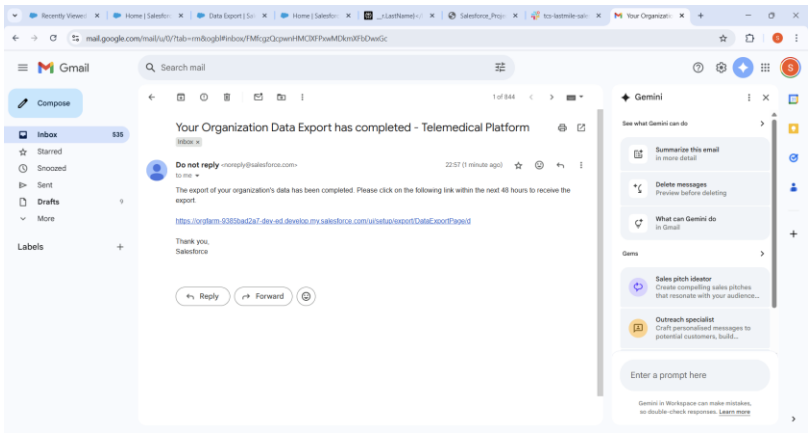
- Matching Rules can be defined to identify duplicates based on patient identifiers such as email, phone number, patient ID, as well as for doctors using license numbers or email addresses. This helps flag potential duplicate records when users enter or import data.
- Duplicate Rules enforce policies to either block duplicate patient or doctor record creation, alert users for manual review, or allow duplicates but log them for audit. This prevents issues like double bookings, duplicate billing, or inconsistent patient histories, ensuring smooth operation of telemedicine consultations and trustworthy data for healthcare decisions.

3.Data Export:

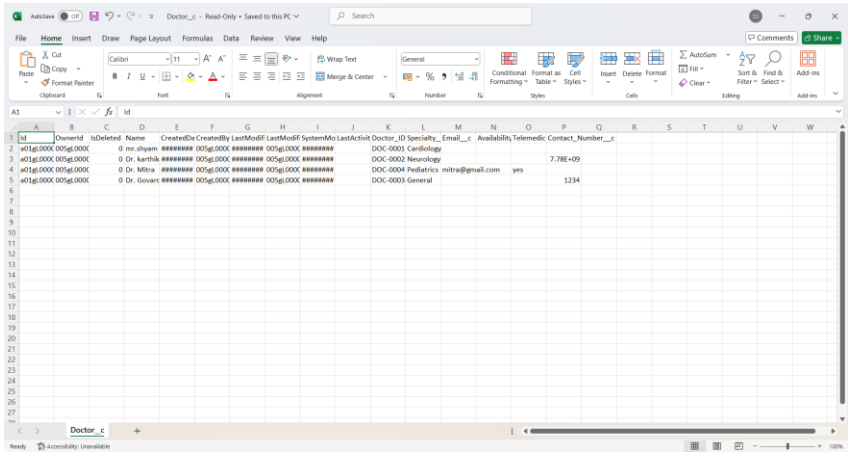
a) Add the object what to be export



b) Got a mail with download link



c) Downloaded the zip file and open the data in the excel

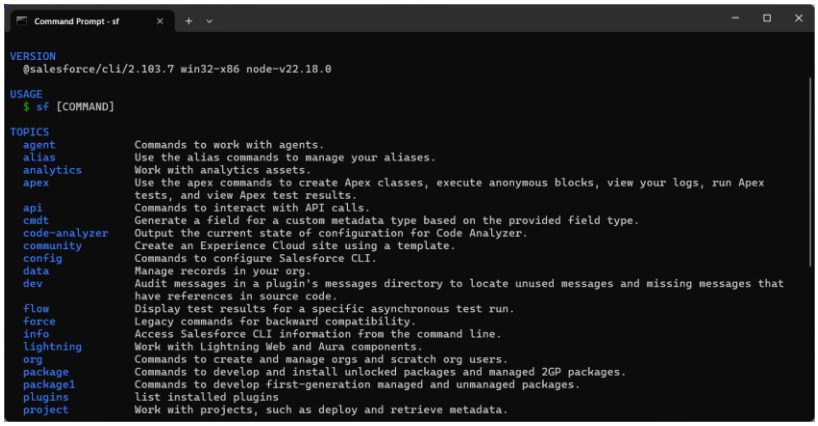


Salesforce Data Loader is a client application designed for bulk import, export, update, and deletion of Salesforce records using CSV files, suitable for processing large datasets efficiently. It

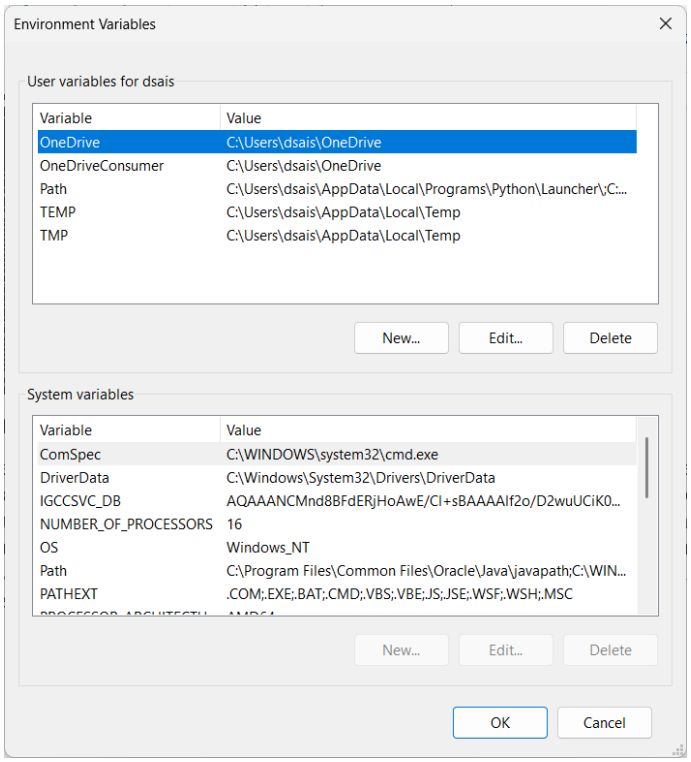
supports an easy-to-use wizard interface, drag-and-drop field mapping, and works across Windows and Mac platforms, offering detailed success and error logs for effective data management.

4.Deployment in vs code and SF dx:

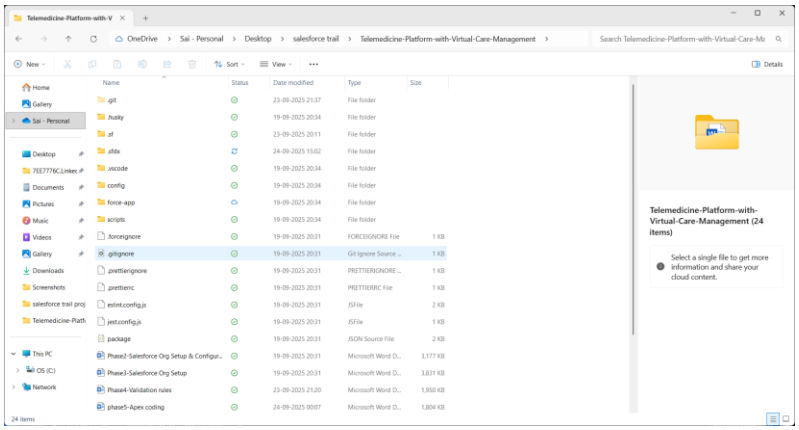
a) This is the sf command to check salesforce CLI is installed or not.



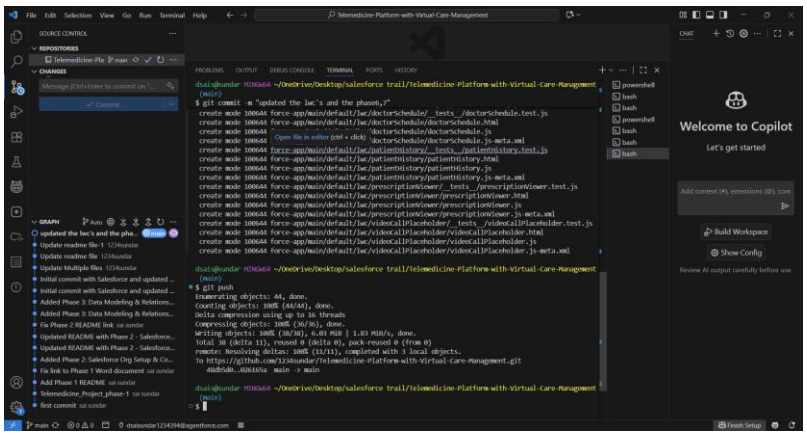
b) Check correct environment is set or not



c) SFDX Folder



d) Successfully push the all lwc to the github repo using the commands

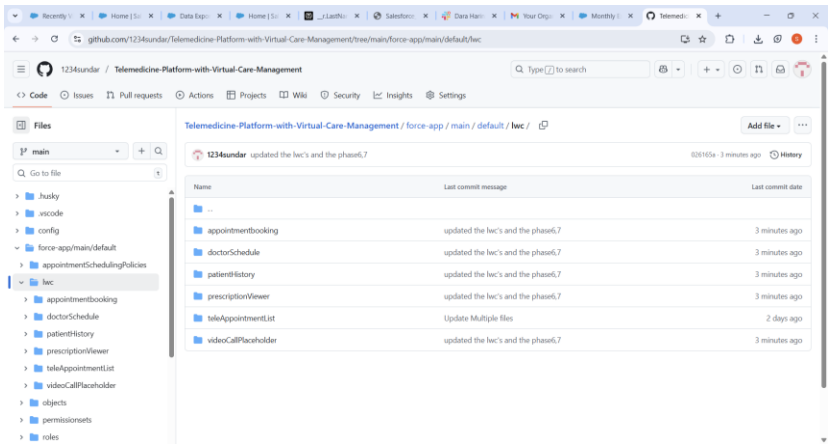


//git init

//git add .

//git commit -m “updated the github repo”

// git Push



For the deployment and retrival of the source is done by right click it and just tap deploy or retrive.

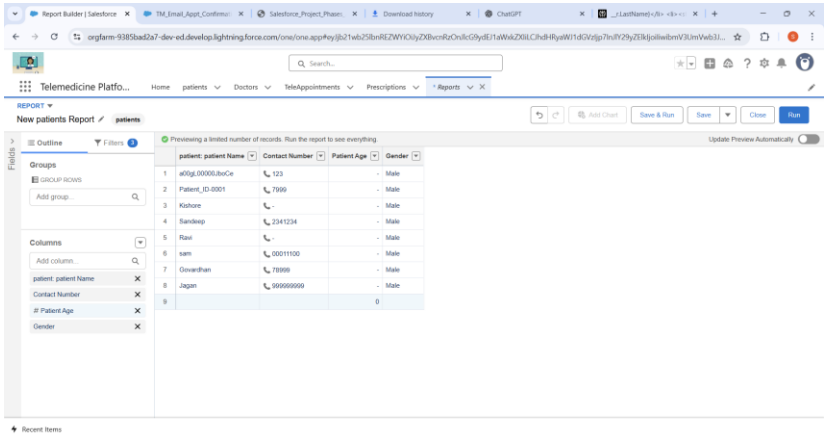
For your telemedicine project, Salesforce DX (SFDX) was used to efficiently manage and deploy custom objects like Patients and Appointments. Metadata and components were retrieved from your development org, enhanced locally in Visual Studio Code, and redeployed via SFDX commands. This method ensured controlled, versioned updates and smooth deployment of telemedicine features. Rigorous testing was performed to confirm workflows and UI elements operated as intended in your telemedicine system.

Phase 9: Reporting, Dashboards & Security Review

Phase9-Report and Dashboards

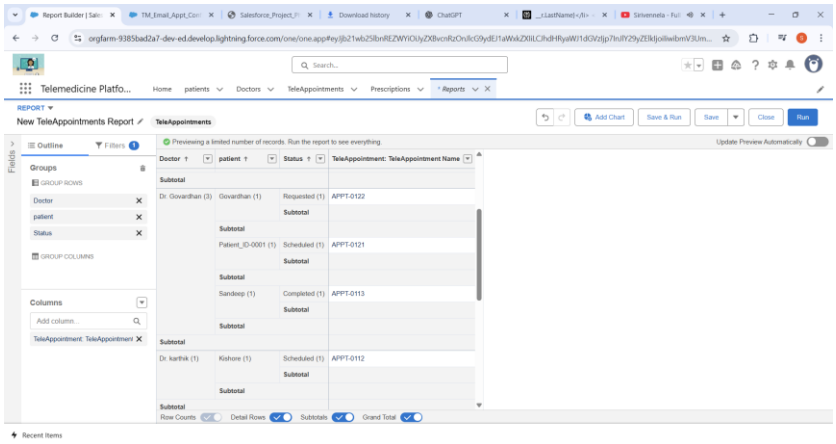
1. Reporting Components: Patient Performance Report

a. Create the new Tabular report Then Add the Patient Name ,Patient Number ,Pateint Age ,Patient Gender(Tabular Report)

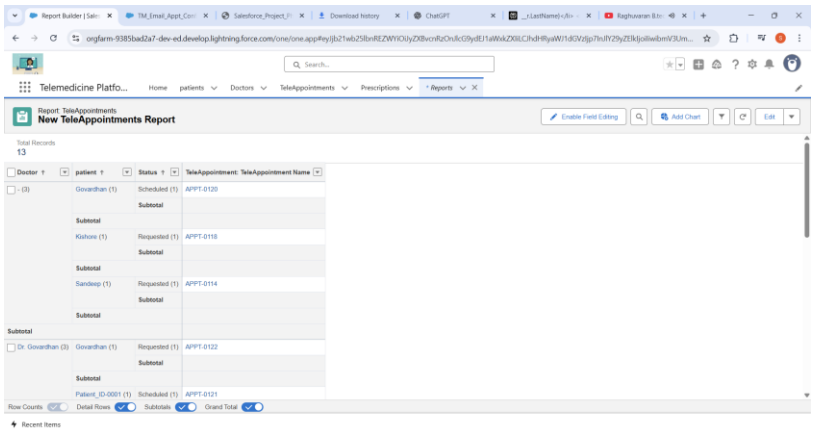


b. Create the new summary report In TeleAppointment: Teleappointment Performance Report

- Add the TeleAppointment Name for the summary report→group rows add Doctor, patient, Status

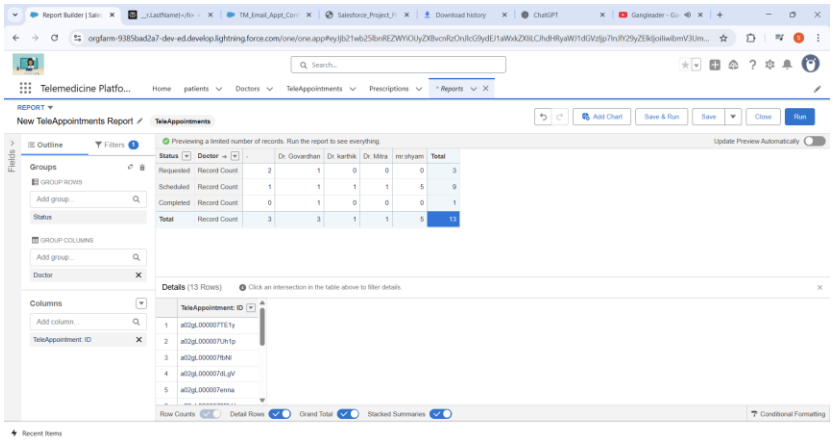


- **Successfully run and the report**

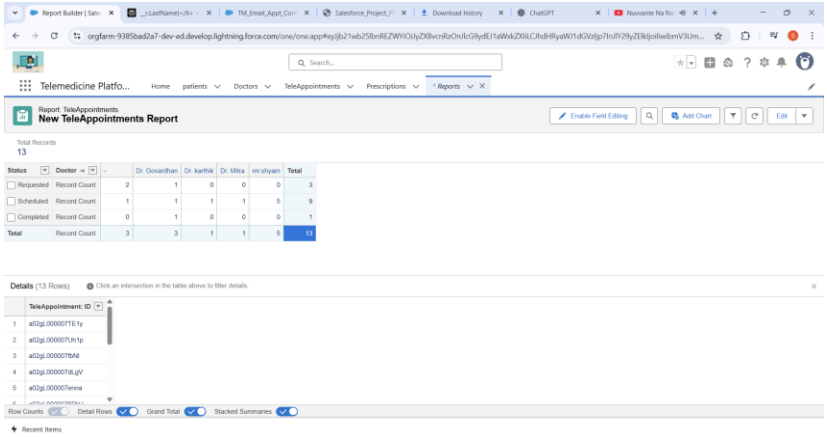


c. Create the new summary report In Doctor: Doctor Performance Report

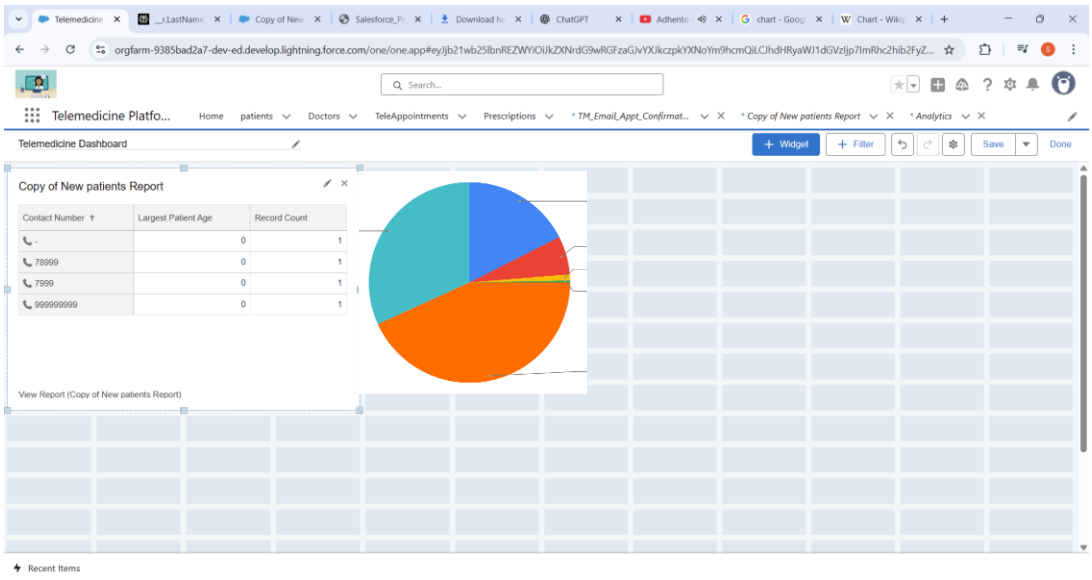
- Add the TeleAppointment Name for the Matrix report→ Drag Doctor into the Group Columns area and Status into Group Rows.



- **Successfully run the report and the total records are 13**



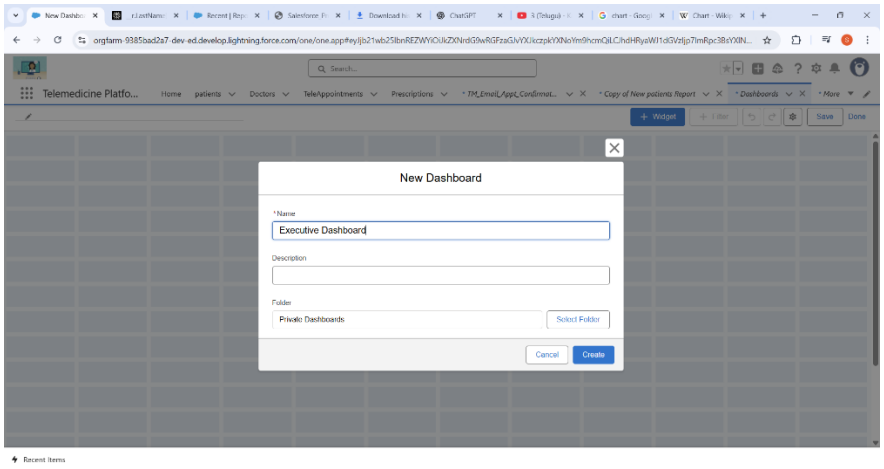
2. Telemedicine Dashboard: Dashboard of Patient



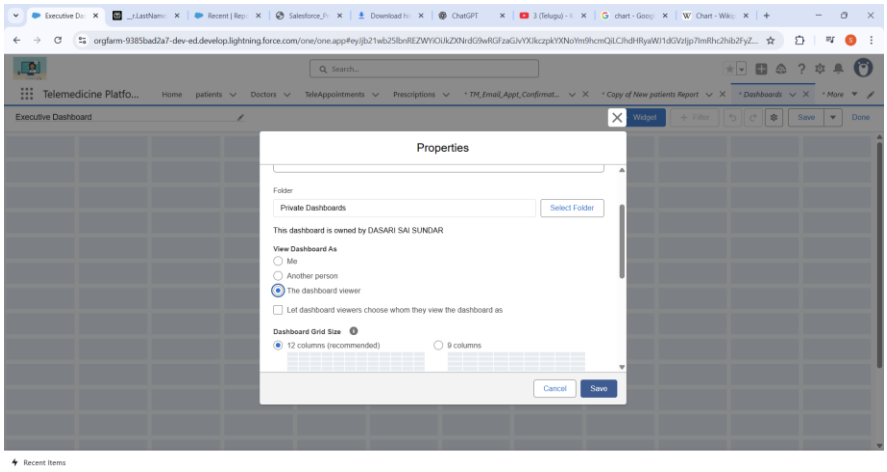
1. Click + Component, select your “Patient Demographics Report,” choose a chart type pie chart and add to the dashboard.

3. Dynamic Dashboard:

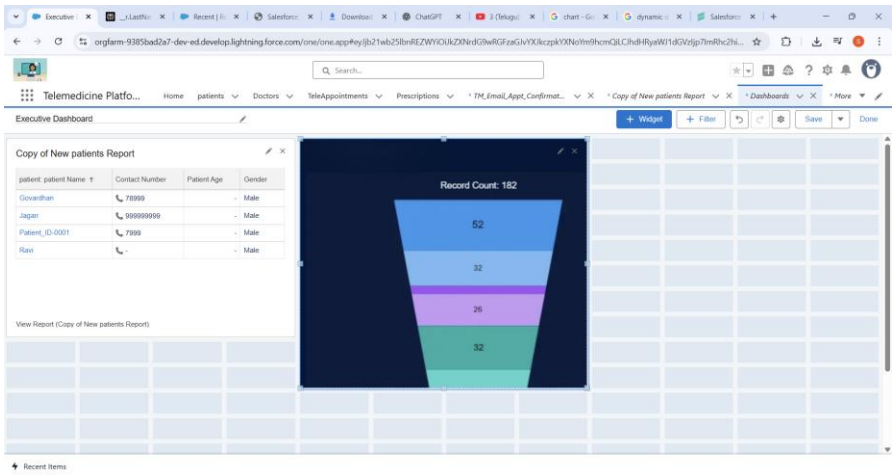
- Created the dashboard with NameExecutive Dashboard:



- Set the properties to the Dashboard Viewer:

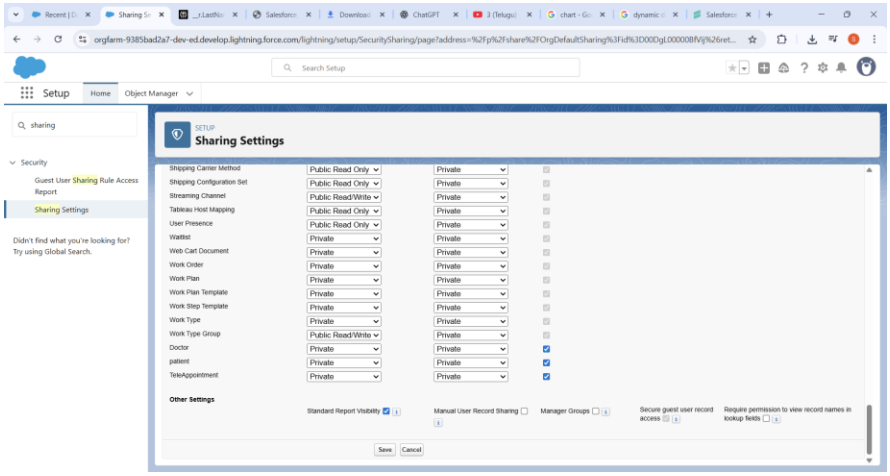


- Successfully add the Graph plot in the Dynamic Dashboard

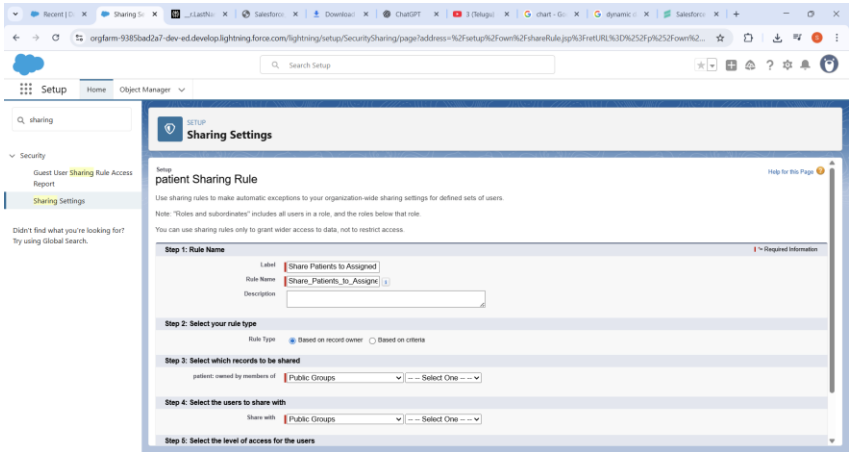


4. Sharing settings:

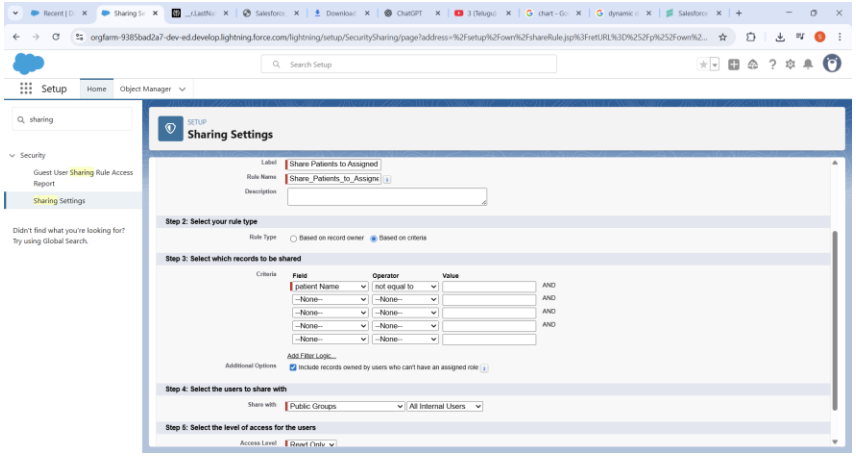
- For the patient and Teleappointment objects, set Default access to Private. This ensures only record owners and users above them in the role hierarchy can view or edit records by default.

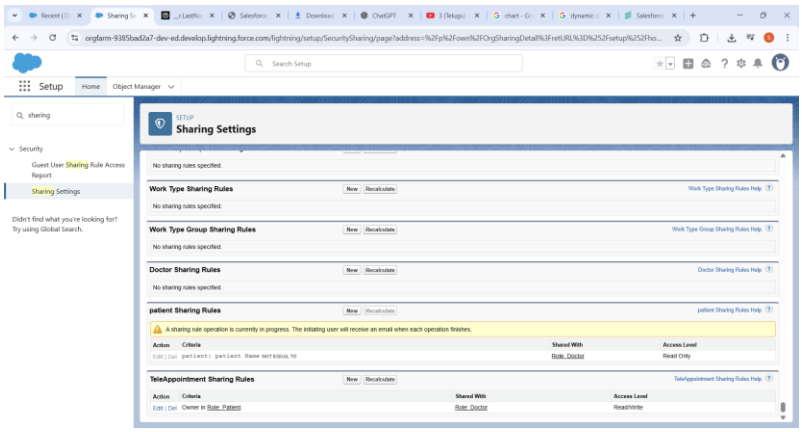


- Created the patient sharing rule: that ensure the Particular Patient assigned to the that Specialization Doctor



- Here the given criteria that the patient can be assigned to the only selected specialization. Patient → Assigned Doctor ≠ blank

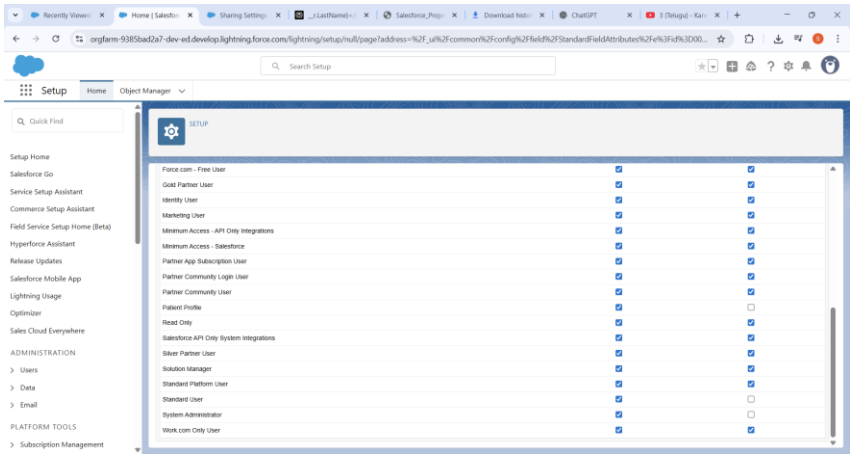


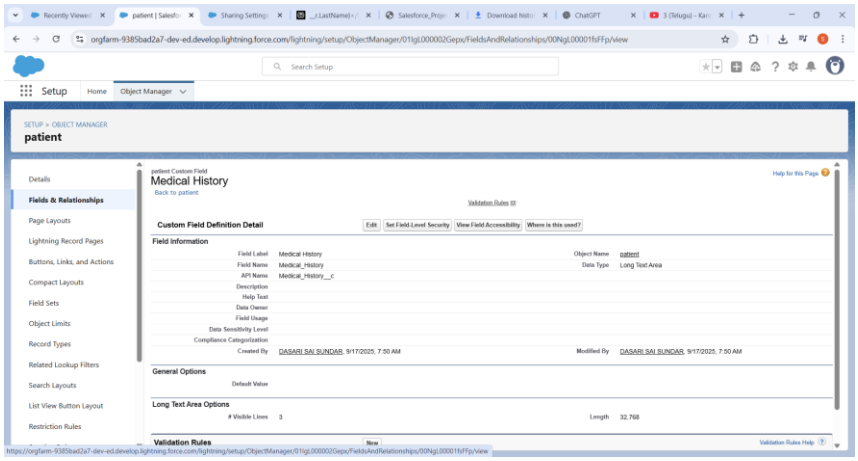


For the Patient and Appointment objects, set Default Internal Access to Private. This ensures only record owners and users above them in the role hierarchy can view or edit records by default.

5. Field Level Security:

- For profiles like Standard User or Guest User, uncheck Visible to completely hide the field.
- For profiles like Doctor or System Administrator, leave Visible checked.
- If certain profiles should see but not edit the field, check Visible and Read-Only.

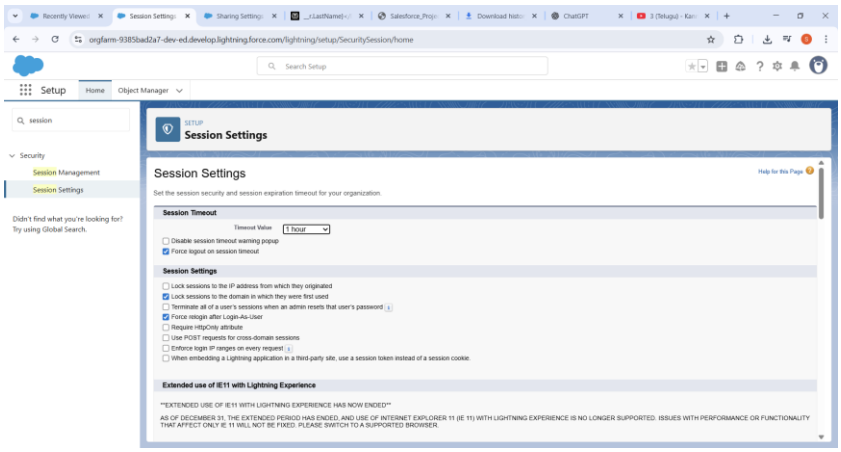




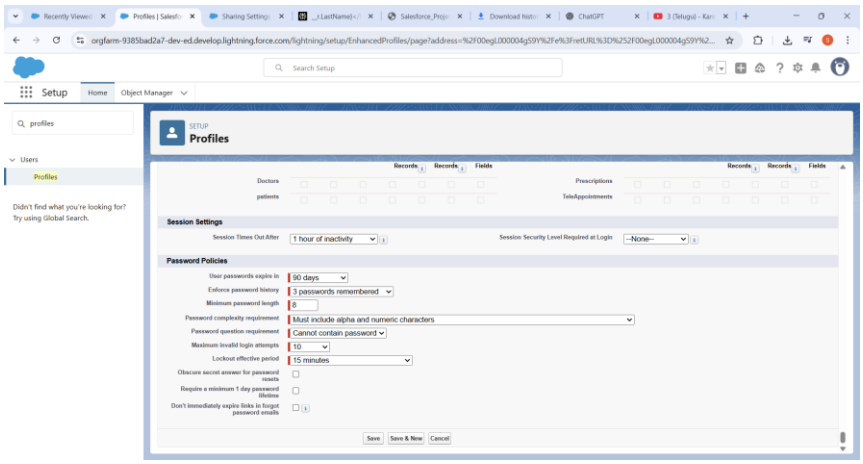
6. Session Settings and Login IP Ranges:

1. Session Settings:

- Under Session Timeout, select your desired idle timeout to one Hour.

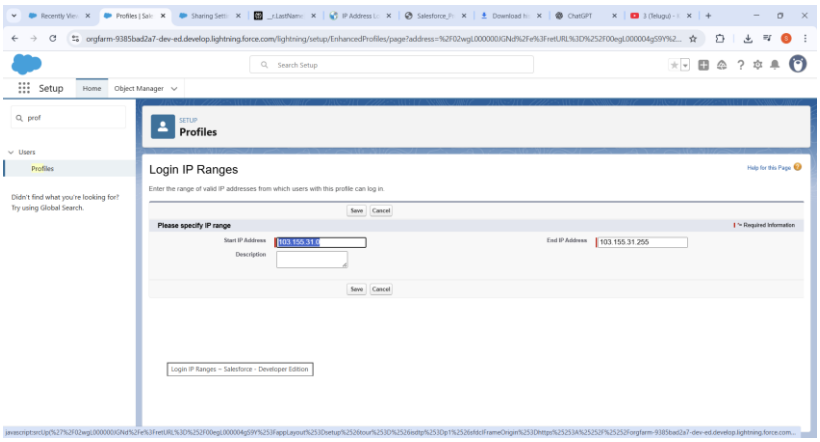


- In the profile settings session timings set to one hour

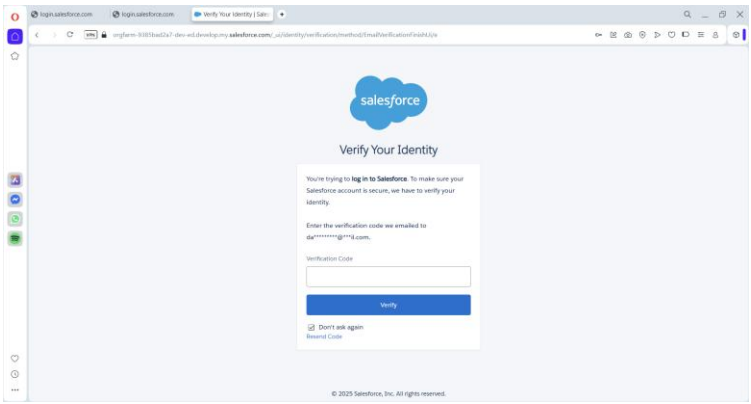


2. Login IP Ranges:

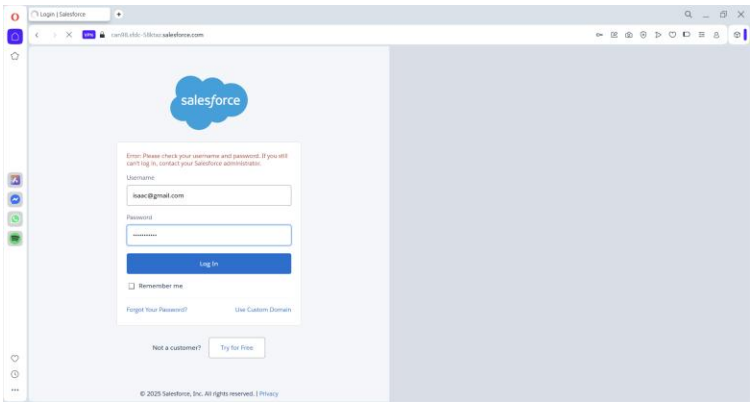
- Enter a Start IP Address and an End IP Address that define your organization's trusted network range covers all IP addresses from 103.155.31.0 up to 103.155.31.255.



- Testcase1:from my hostel no problem to login



- From my friends home its not opening

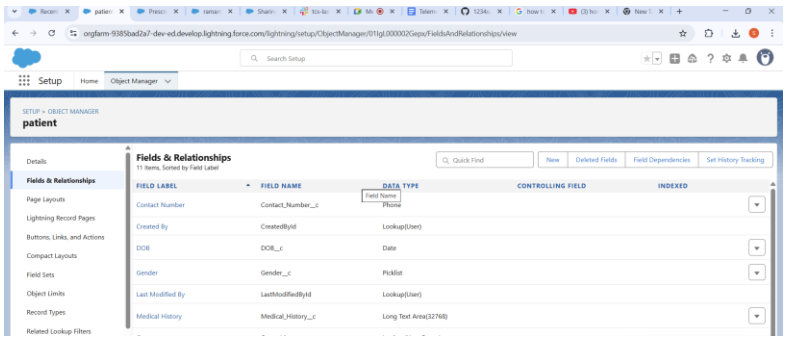


Phase 10: quality assurance

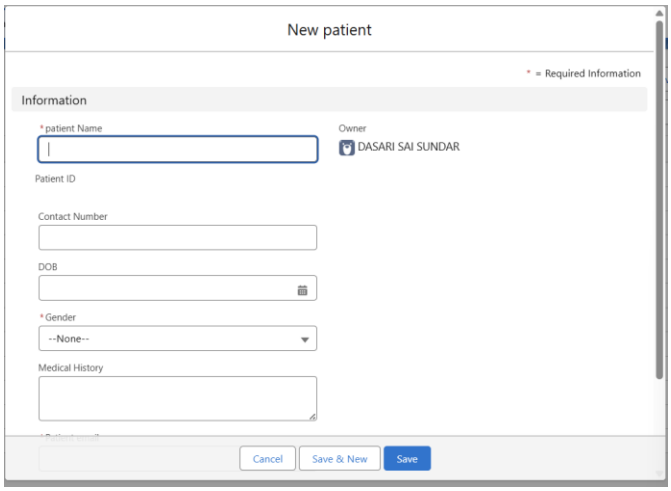
Phase10-Quality Assurance Test

7. Created Objects:

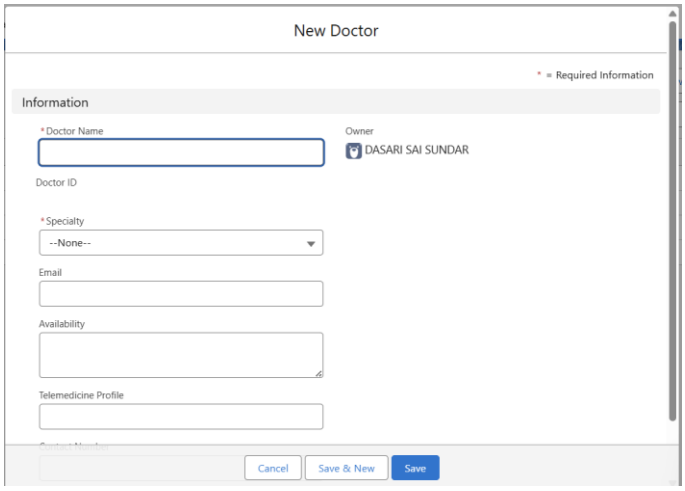
For the patient object:



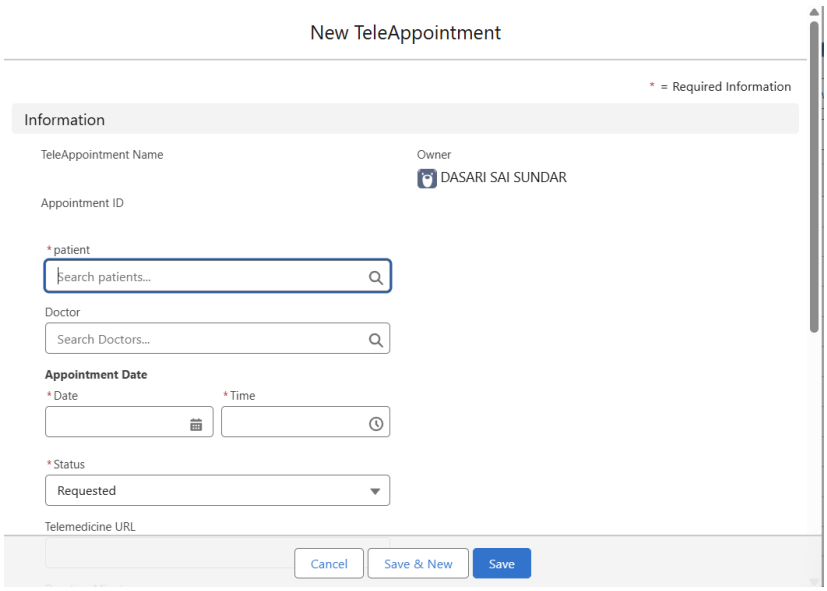
This is how it looks like



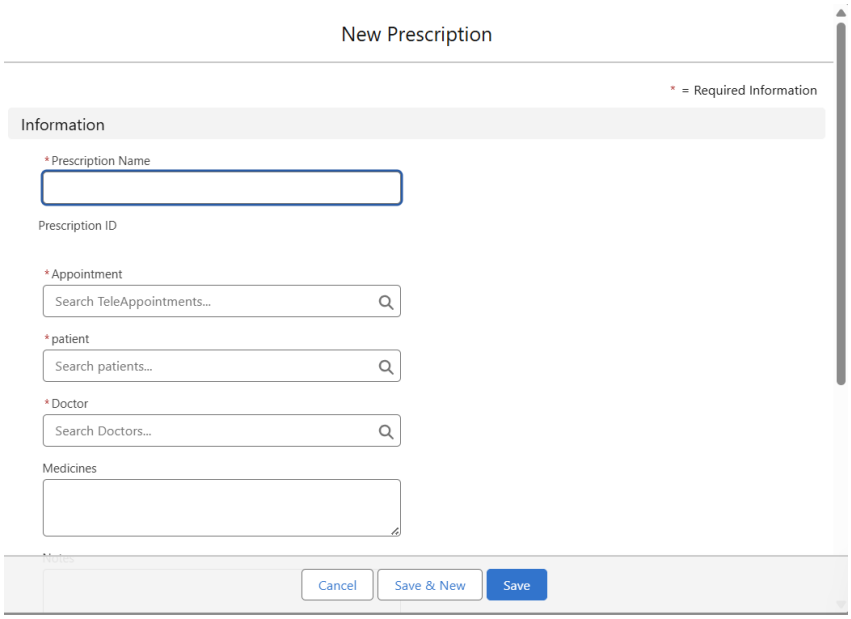
For the doctor object:



For teleappointment object:

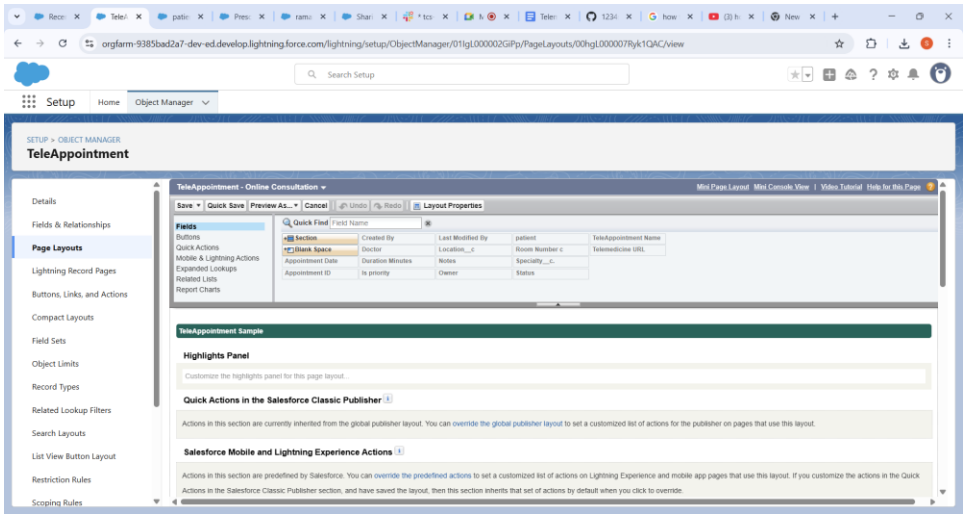


For the Prescriptions:

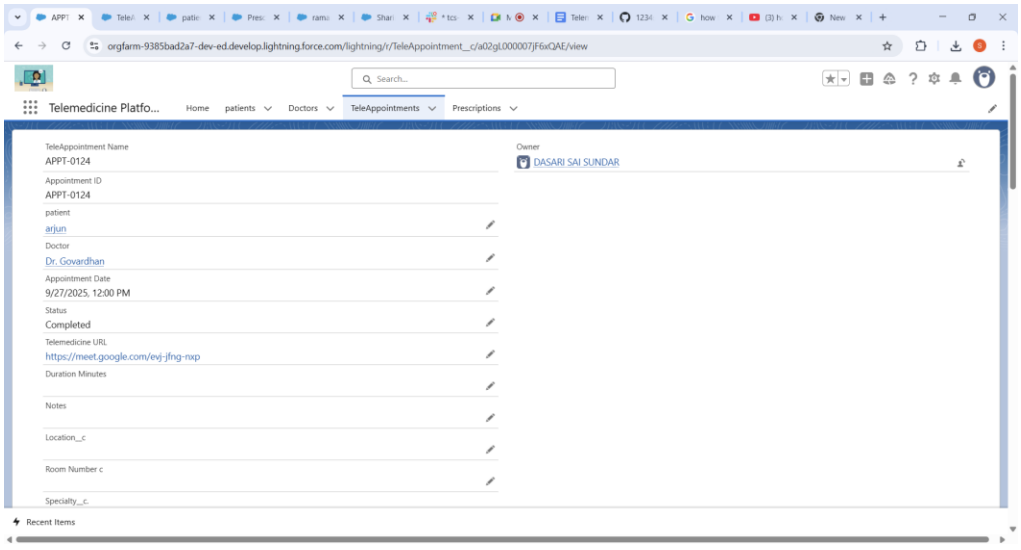


Every field created is successfully added in the Platform objects.

8. Record Types: The is the custom Online Consultation

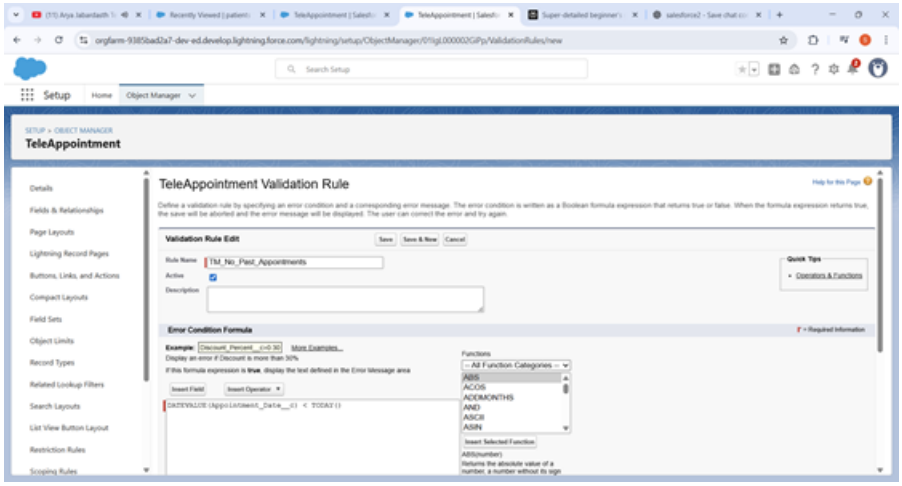


This is Appear on the screen

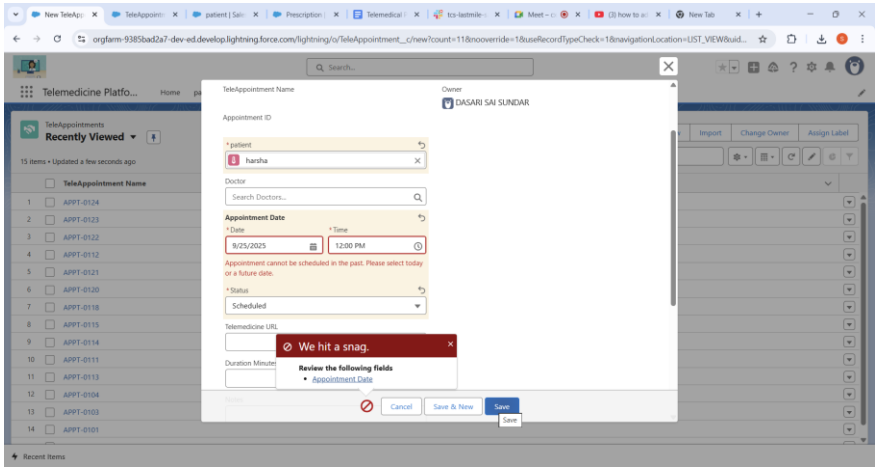


9. **Process Automation:**

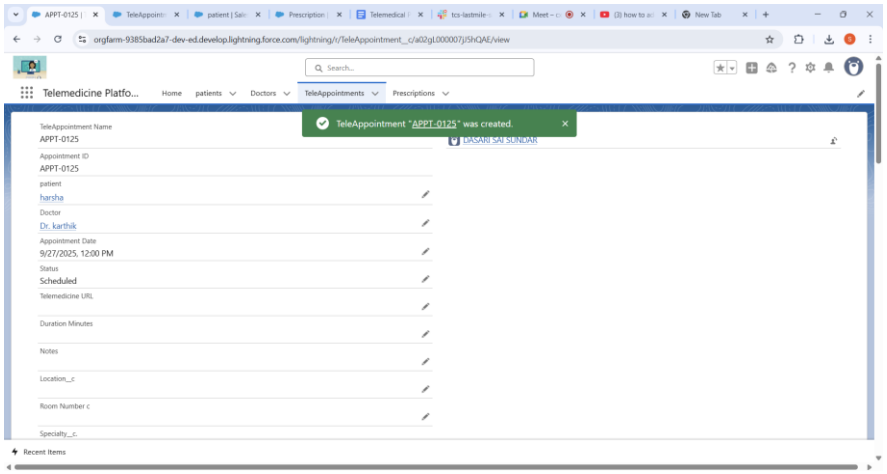
a.Validation Rule: TM_No_Past_Appointments: This rule prevents a user from creating a TeleAppointment with a date that is in the past. It uses the formula.



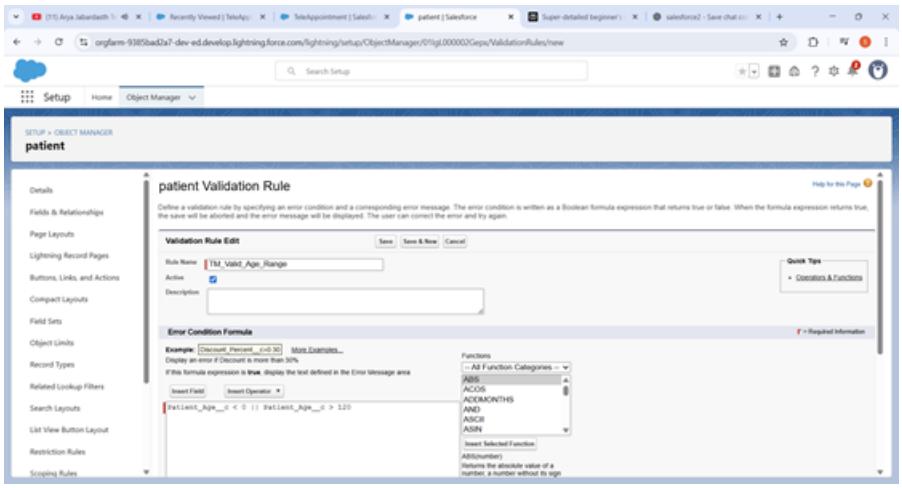
Process Automation→Testcase1: I have created the appointment in past means yesterday.



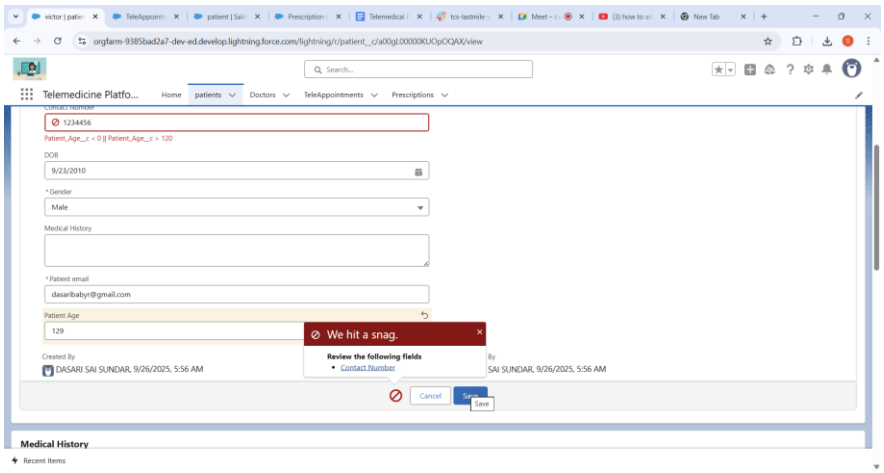
Process Automation →Testcase2:Give the date as the 2 tomorrow its saved successfully



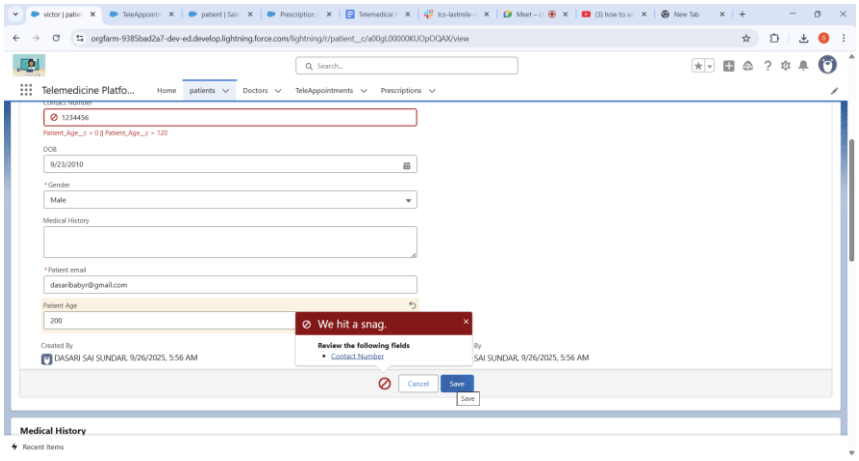
b. Age Must be between 1-120:



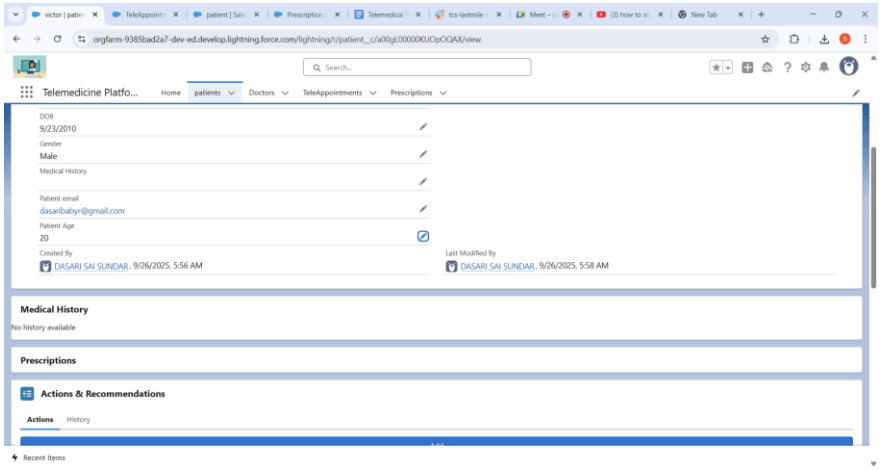
Process Automation →Test Case1:I give the patient age in 129:



Process Automation →Test case2:I have given the patient age in 200:

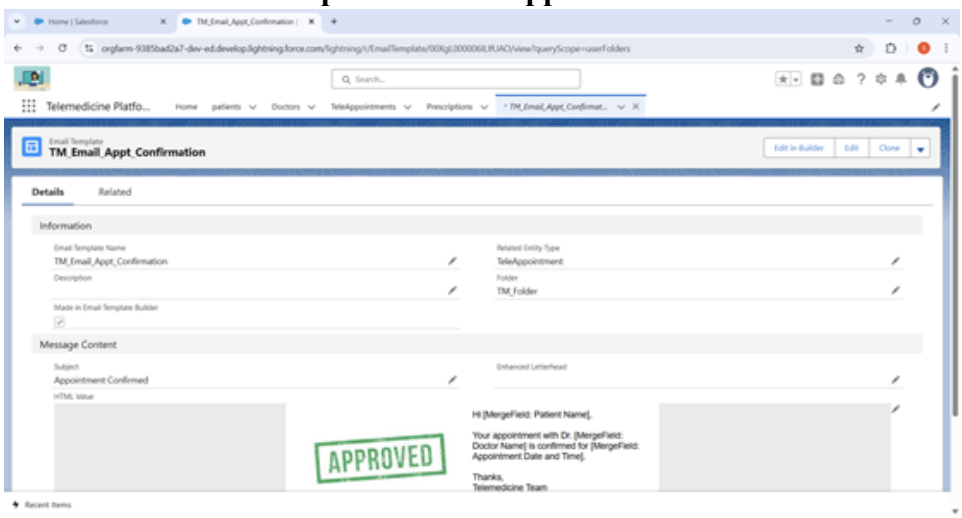


Process Automation →Test Case3: Given 20 as the age and Testcase Passed:

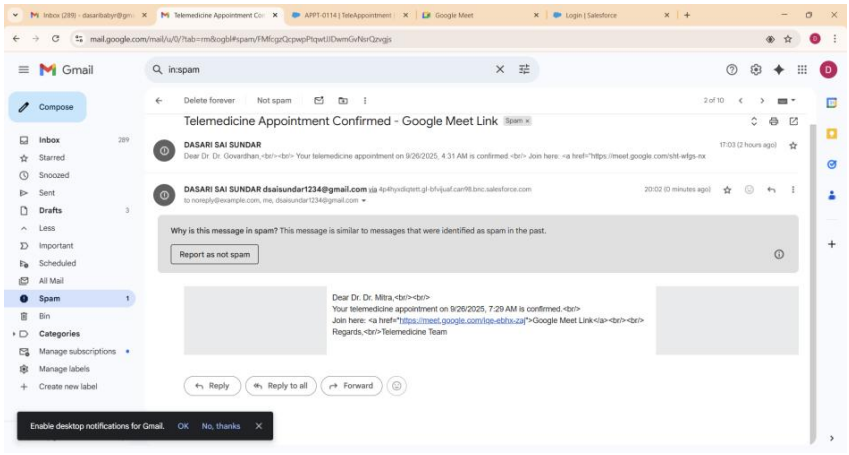


10. Lightning Email Template:

Created the Email Template For the Appointment Confirmation:



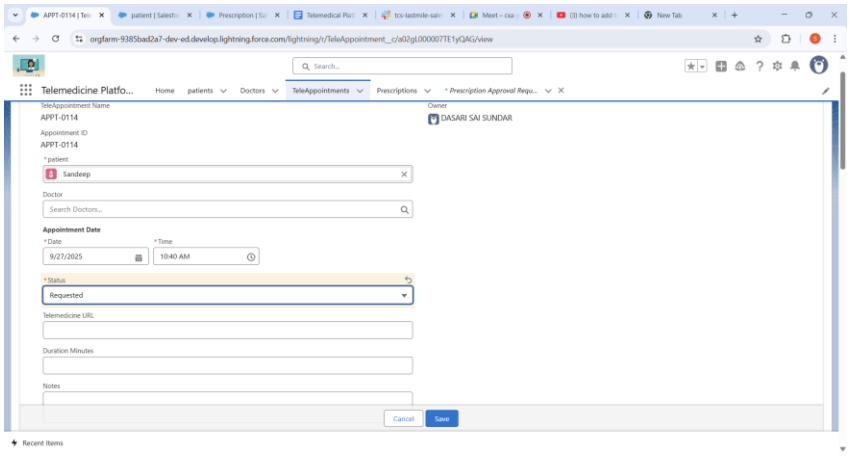
Lightning Email Template →Test case1:

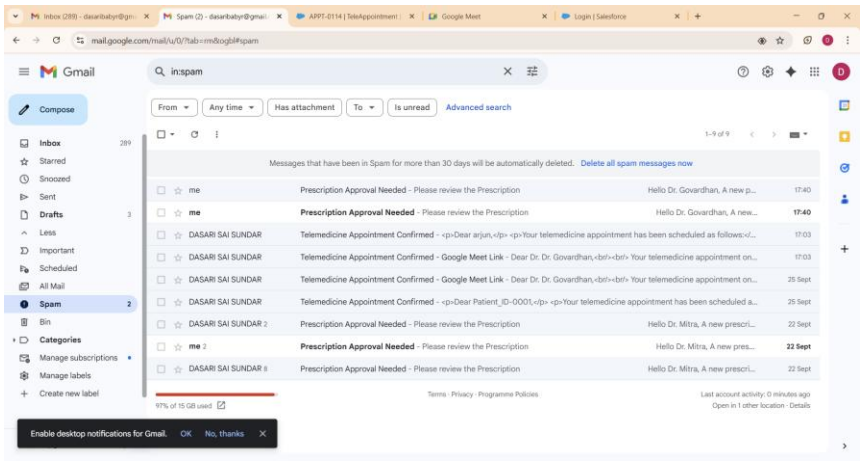


11. Workflow Rules Validation:

This work flow uses the Above Email Template. The doctor can make the approval in his org account then the email Approval confirmed.

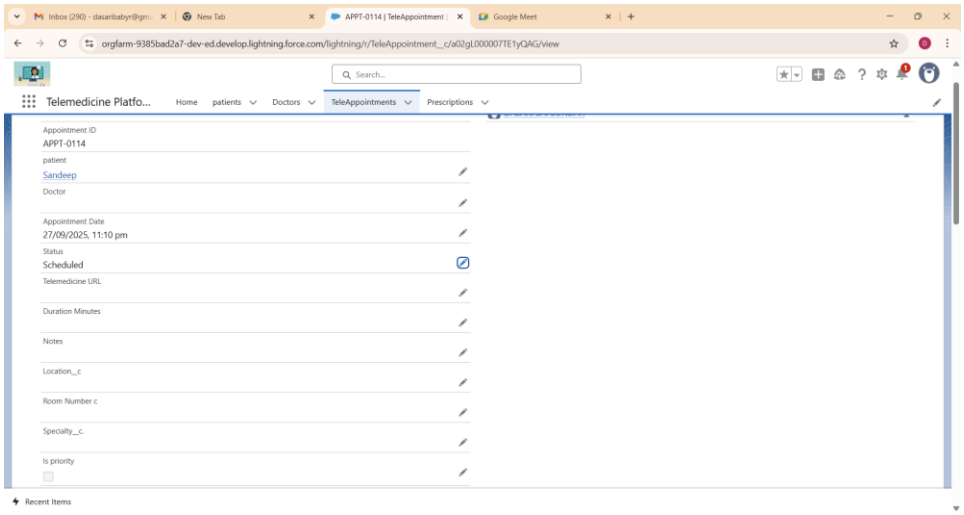
Workflow Rules Validation →Test case1:



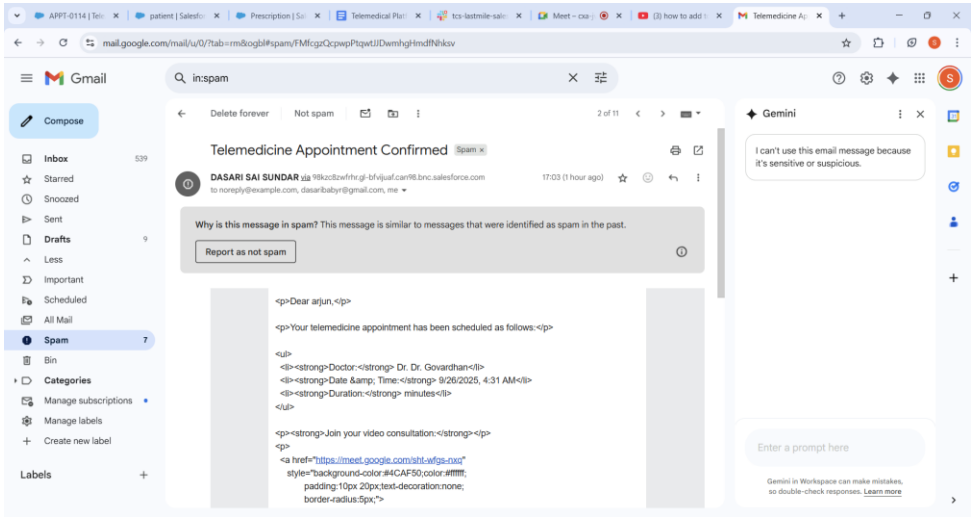


No mail can be updated in the patient mail.

Workflow Rules Validation →Test Case2:

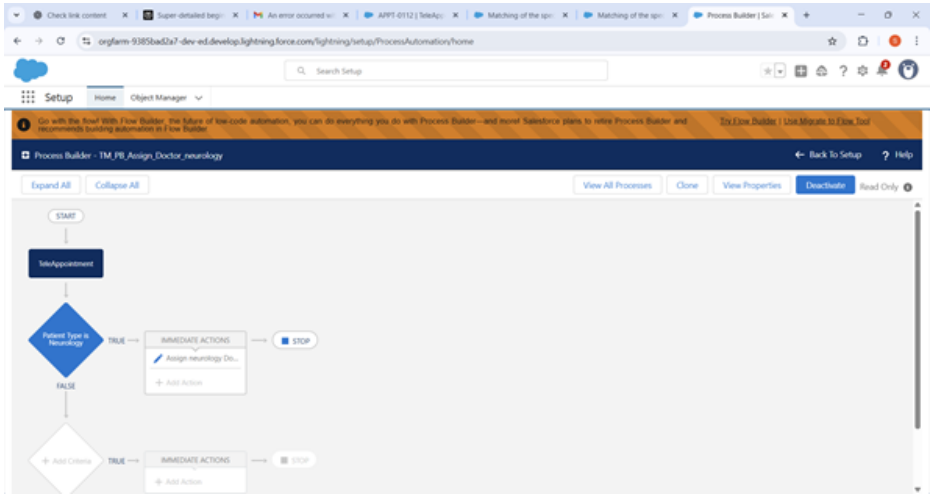


As the status is set as Scheduled and then got the mail.



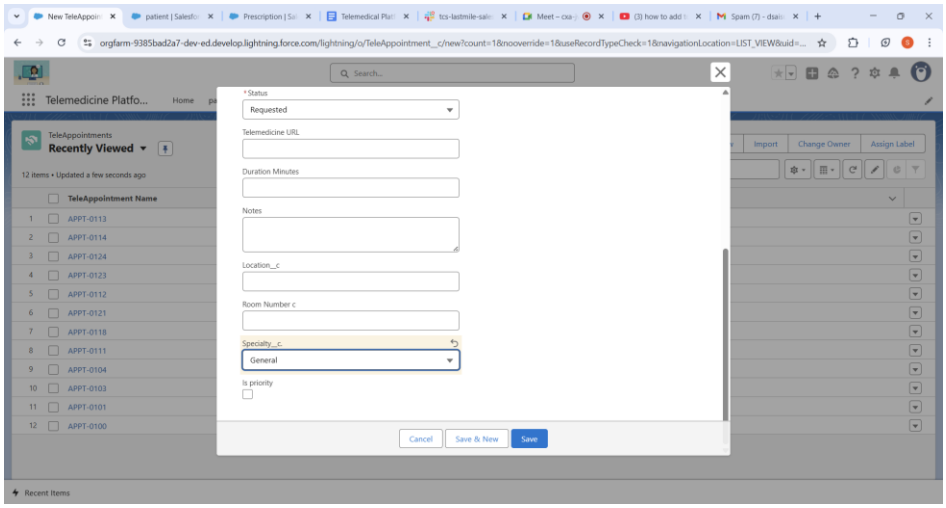
12. Process Builder:

In TeleAppointment Object when we select speciality we can get assigned the doctor according to the specialization.

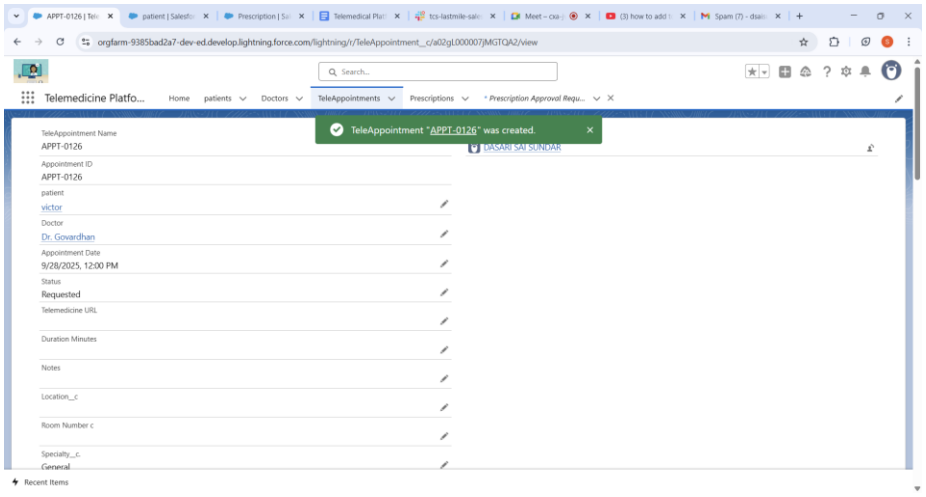


Process Builder →Test case1:

Just give the specially of the problem.

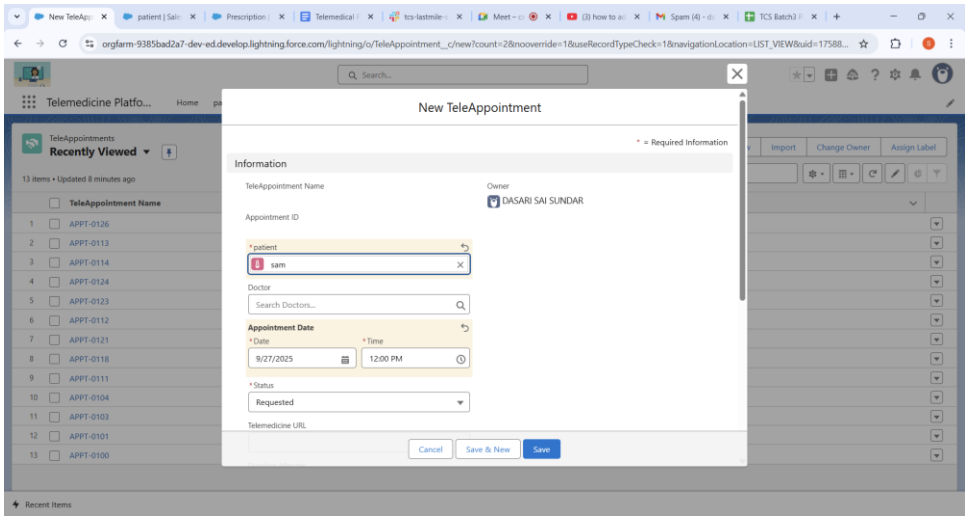


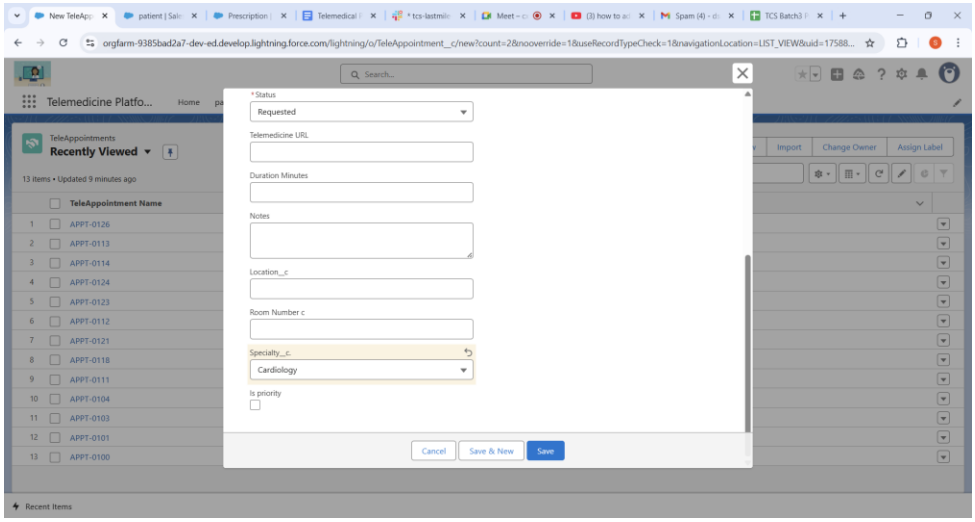
Got the doctor Assigned directly as I have given the general. Govardhan is assigned because his specialization is the General.



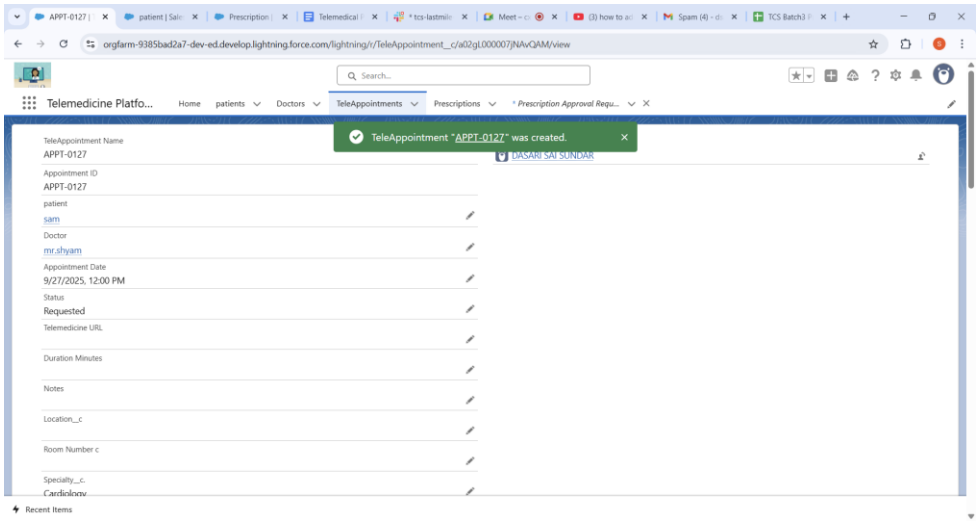
Process Builder →Test Case2:

Just given the specially as the **CARDIOLOGY. Not Assigned the Doctor.**



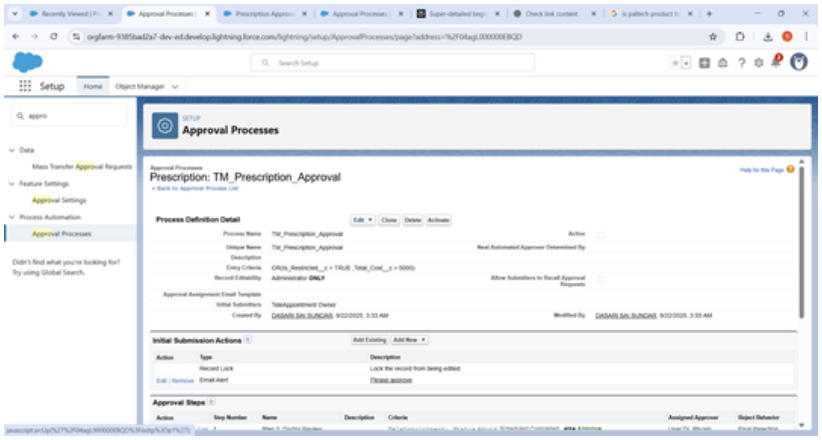


There we see the Mr. Shyam is Assigned successfully.(As the Shyam is Cardiology)



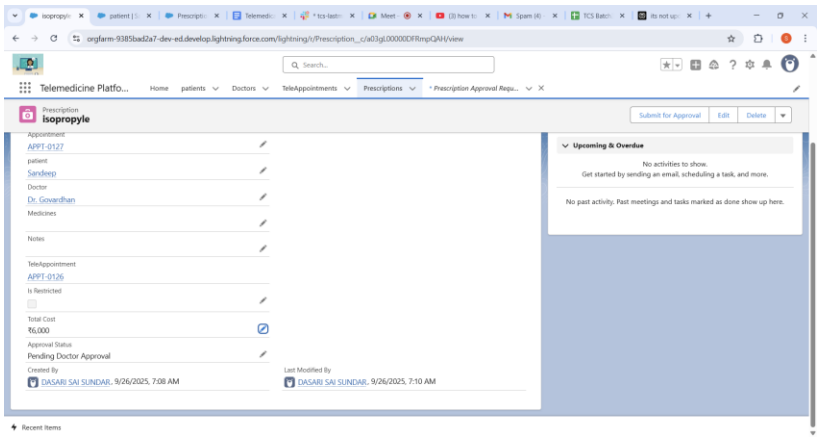
13. Approval Process:

This is the approval process for the **prescription** approval→it can be only **Approved** by the **Doctor** user.

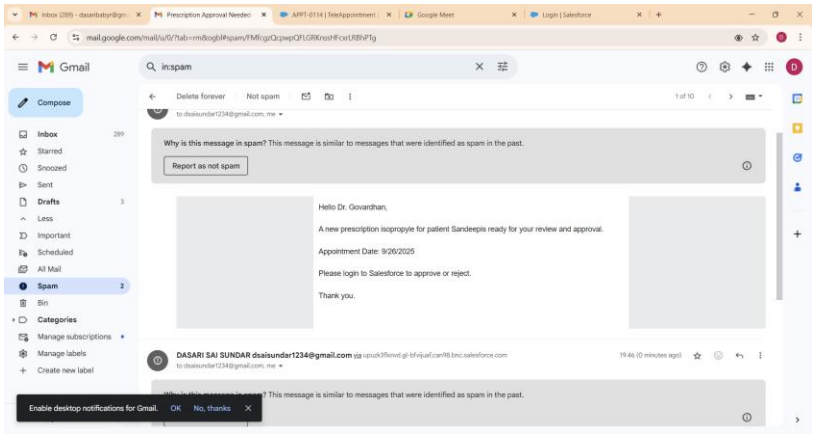


Created the Email templated and condition to send mail is total cost is more than the 5000.As the condition is achieved we should click the button **SUBMIT FOR APPROVAL.**

Test Case1: In this case mail to Dr.govardhan as to approve the prescription.

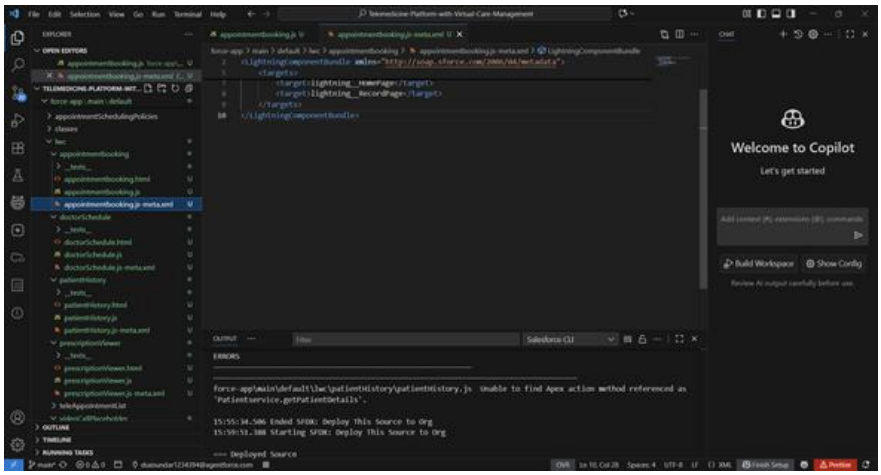


In the email box of the Doctor:

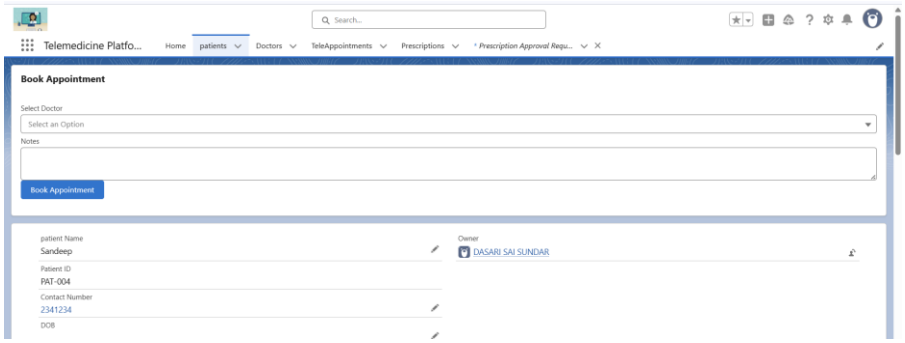


14. Lightning Web Component:

Here I have created AppointmentBooking,DoctorSchdule,PatientHistory,PrescriptionViewer etc..



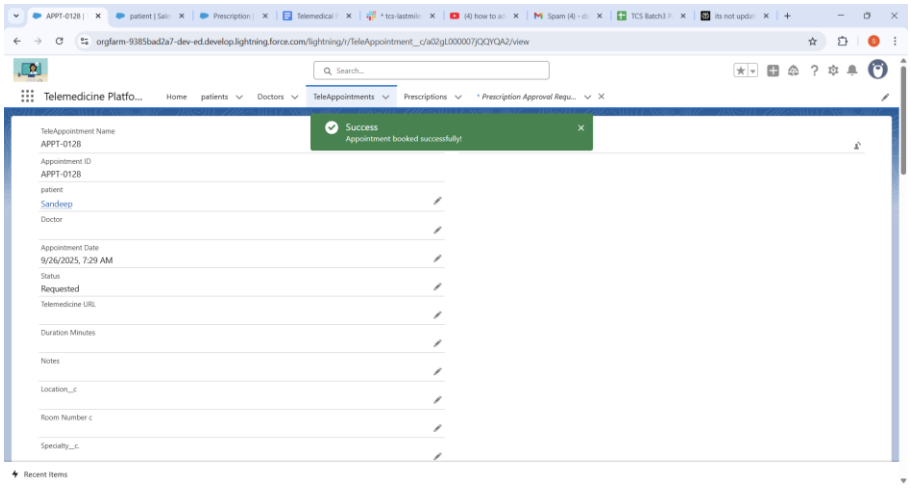
For our First Case: **AppointmentBooking** LWC is created and deployed successfully.



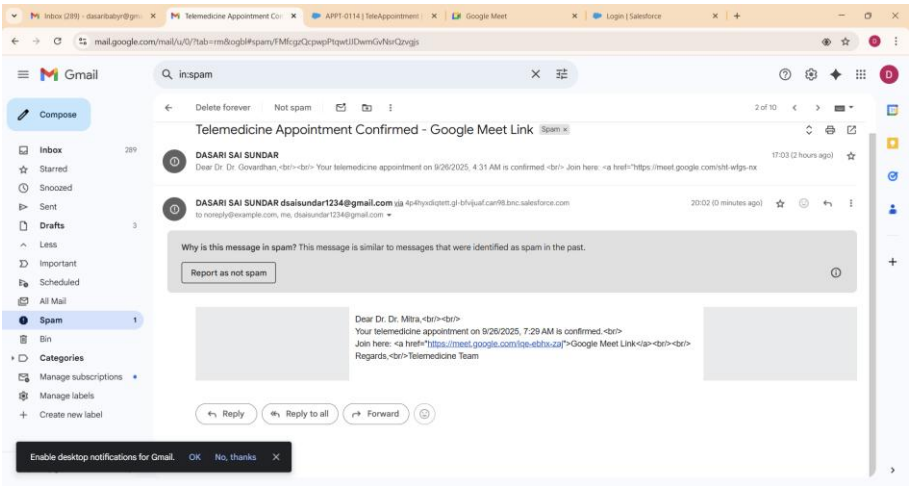
Whenever opened the patient Object in the platform we can see the Appointment Booking

Lwc with the Doctor name specialization and after clicking we can directly navigate to the Teleappointment and the Mail is send to the Both Doctor and patient.

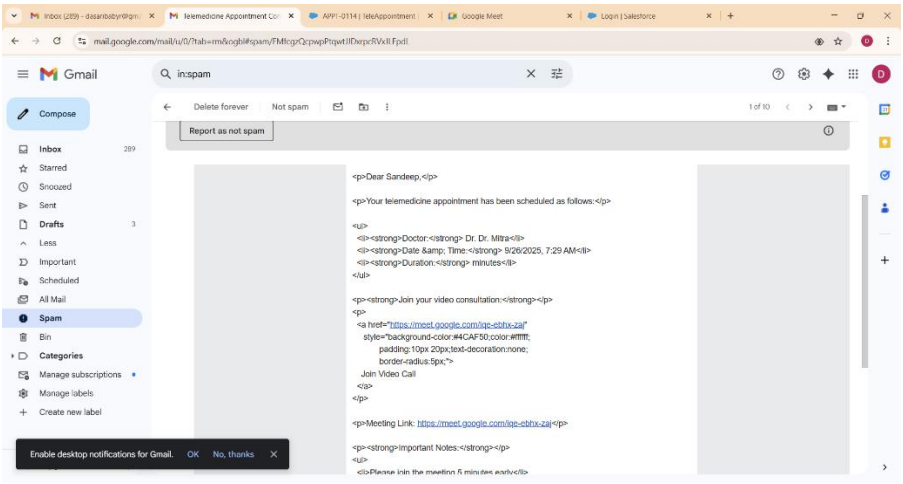
Test case1:Appointment successful with the Sandeep and doctor Mithra.



Here is the **mail** to be sent to Dr. **Mitra**:

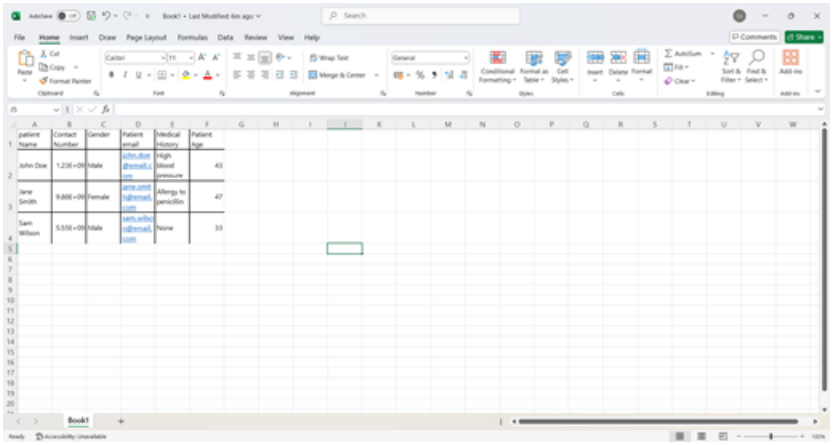


Here is the mail for the patient (**Sandeep**):

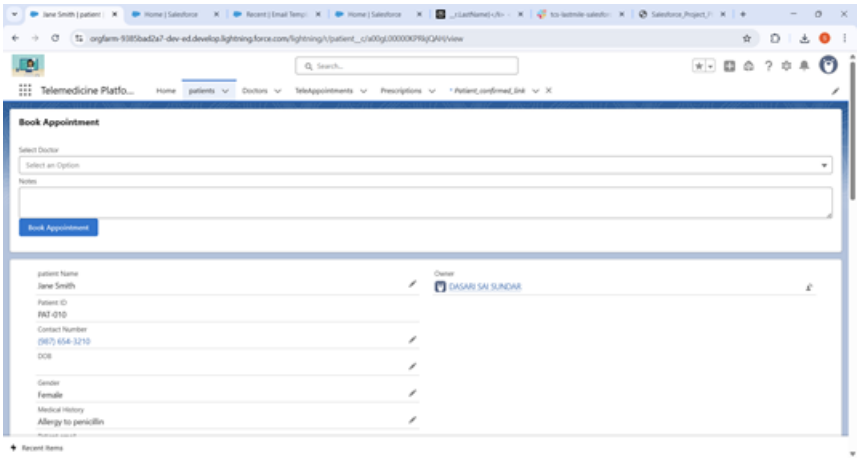


15. Import the Data from the Excel:

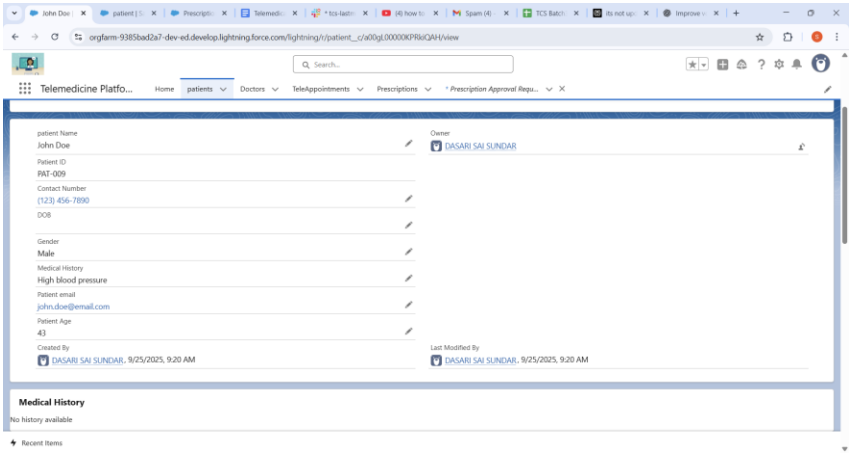
Created the Data of a Patient in the Word and save it in the .csv



Test Case1-Success Jane Smith Fetched to the Telemedical platform to the Patient.

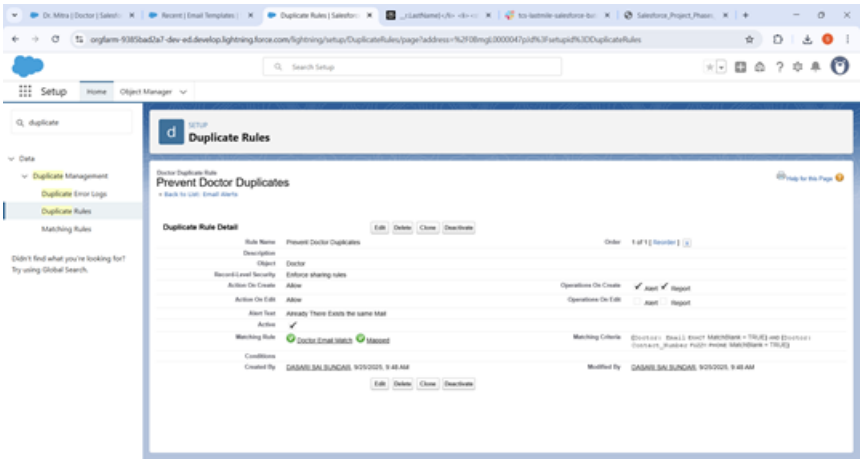


Test Case2→John Doe is fetched to the Patients data.

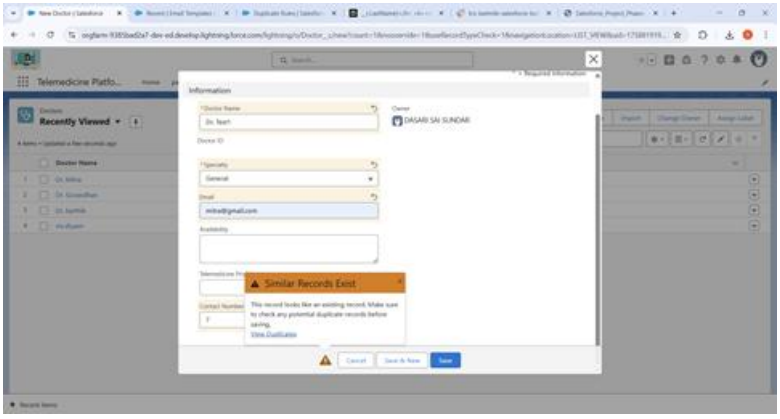


16. Duplicate Rules:

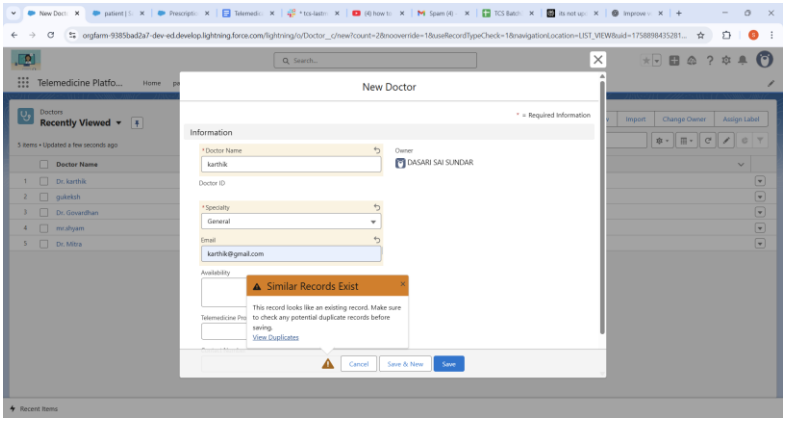
Created the duplicate rule with the Matching rule with Email.



Duplicate Rules →Test Case1→As we same email is given then the **Warning** may appear as the same mail.

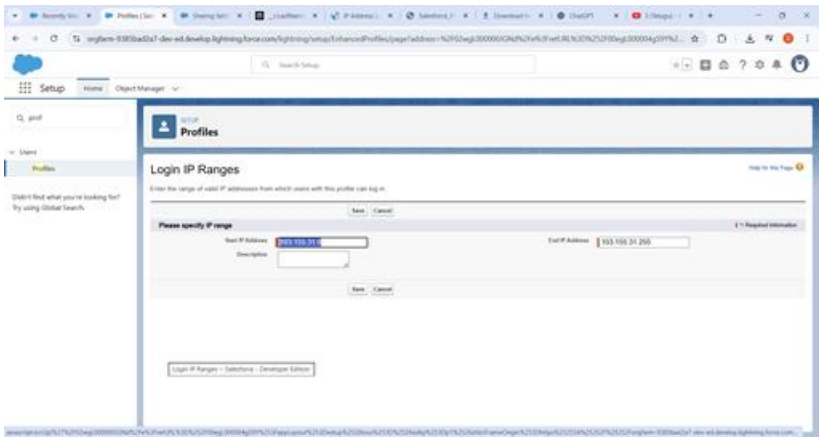


Duplicate Rules →Test Case2→As we same email is given then the **Warning** may appear as the same mail.

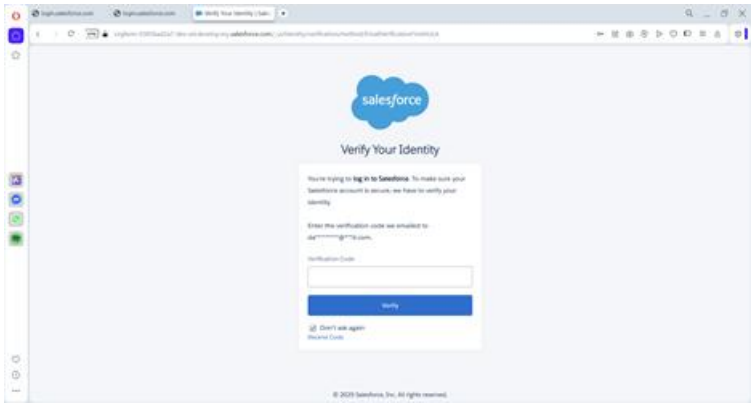


17. Login IP Ranges:

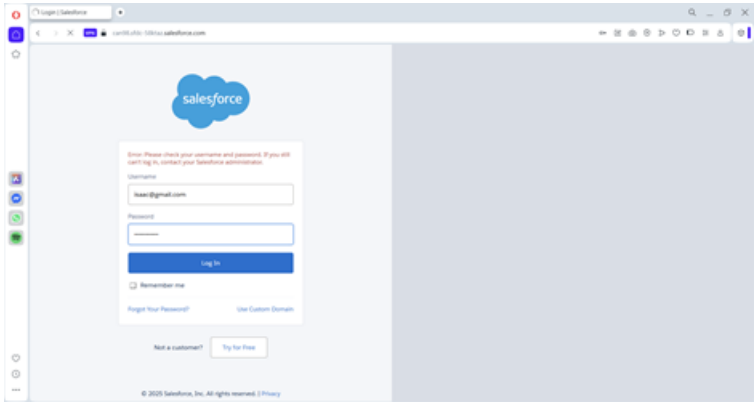
Enter a Start IP Address and an End IP Address that define your organization's trusted network range covers all IP addresses from 103.155.31.0 up to 103.155.31.255.



Login IP Ranges→ Testcase1:from my hostel no problem to login



- From my friends home its not opening



THE END