

## **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.

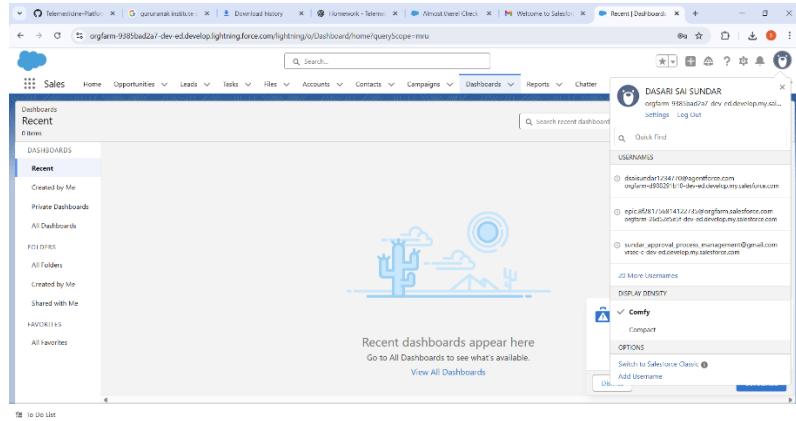
### **Objectives:**

- 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**

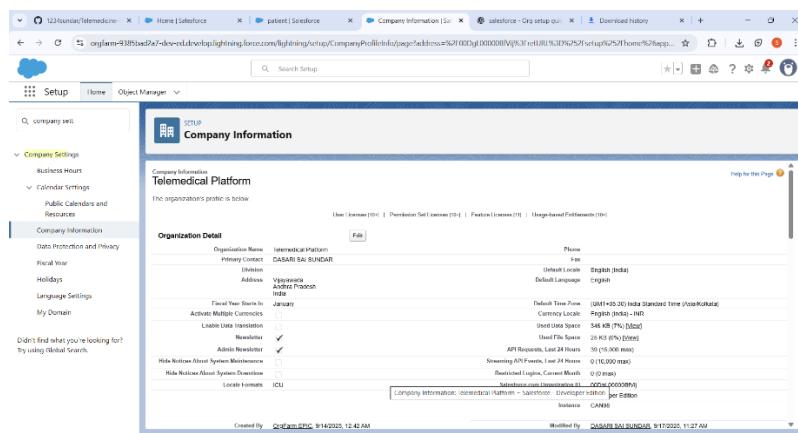
## Phase-2: Salesforce Org Setup & Configuration:

### 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.

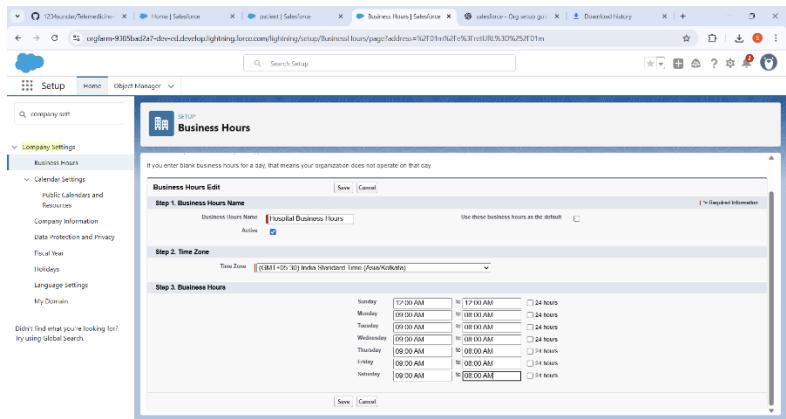
### 2. Company Profile Setup:



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

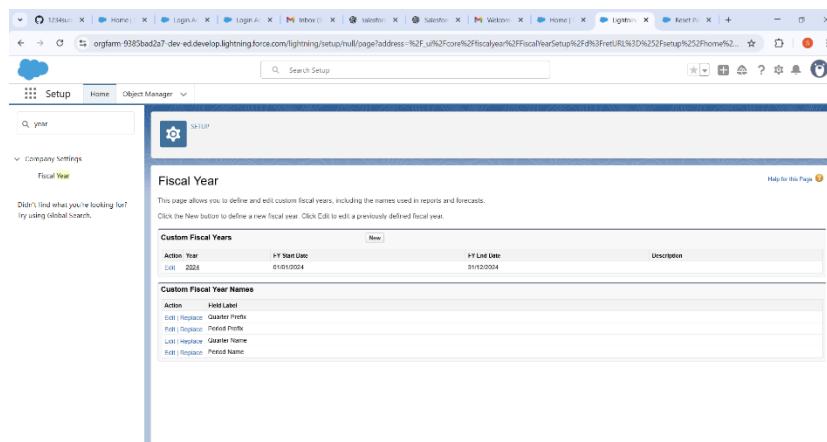
registered address. The local time zone was set to **IST (India Standard Time)** to align with working hours. The default currency was set to **INR**, and the default language was set to **English**. This ensured that appointment schedules, billing, and reporting were accurate for the region. All settings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

The screenshot shows the Salesforce Setup interface with the 'Users' section selected. A specific user record is displayed under 'User Detail'. The user's name is 'Admin User', and their email is 'dsaisundar1234@gmail.com'. They are assigned the 'System Administrator' license. Other details like title, company, department, division, address, time zone, locale, language, and delegated approver are listed. The 'Sharing' tab is selected at the top of the page.

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

The screenshot shows the Salesforce Setup interface with the 'Users' section selected. A specific user record is displayed under 'User Detail'. The user's name is 'Dr. Shyam', and their email is 'ganesbabu@gmail.com'. They are assigned the 'Standard User' license. Other details like title, company, department, division, address, time zone, locale, language, and delegated approver are listed. The 'Sharing' tab is selected at the top of the page.

**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.

The screenshot shows the Salesforce Setup interface with the 'Profiles' tab selected. A new profile named 'Patient Profile' is being created. The 'Profile Detail' section includes fields for Name (Patient Profile), User License (Salesforce), Description (DASAAR SAL BUNDAB), and Created By (DASAAR SAL BUNDAB). The 'Page Layouts' section lists various standard object layouts and location group assignments. The 'Standard Object Layouts' include Global (Global List View Assignment), Email Application (Not Assigned), Home Page Layout (Home Page Detail View Assignment), Account (Account List View Assignment), and Alternative Payment Method (Alternative Payment Method List View Assignment). The 'Location Group Assignment' section includes Global (Global Assignment), Macro (Macro Assignment), Object Whichever (Object Whichever List View Assignment), Operating Hours (Operating Hours List View Assignment), Opportunity (Opportunity List View Assignment), and Opportunity Line Item (Opportunity Line Item List View Assignment).

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:

The screenshot shows the Salesforce Setup interface with the 'Roles' tab selected. A new role hierarchy is being built. The 'Your Organization's Role Hierarchy' tree includes the following roles and their assignments:

- Telemedical Platform** (Add Role)
  - Admin** (Edit | Del | Assign)
    - Doctor** (Edit | Del | Assign)
      - Patient** (Edit | Del | Assign)
- CEO** (Edit | Del | Assign)
  - CFO** (Edit | Del | Assign)
    - COO** (Edit | Del | Assign)
      - SVP, Customer Service & Support** (Edit | Del | Assign)
        - Customer Support - International** (Edit | Del | Assign)
          - Customer Support - North America** (Edit | Del | Assign)

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:

The top screenshot shows the 'Permission Sets' page with a specific permission set named 'Prescription Edit Access'. It displays details like API Name (Prescription\_Edit\_Access), Description (Allows users to edit prescription records), and Object Settings (Edit access to Prescription object). The bottom screenshot shows the 'Assignment Summary' page after assigning the permission set to a user named 'Dr. Shyam'. The status indicates 1 assignments were successful.

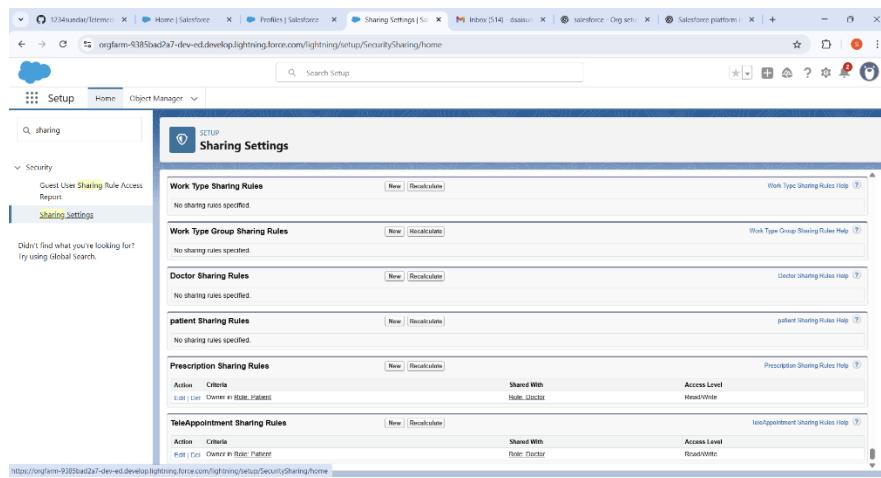
**Permission Sets** were created to provide extra access where required. For example, a permission set was created to allow doctors to access Reports and Dashboards without modifying their base profile. Patients were not assigned any permission sets as their access was limited by design. This approach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

## 9. Organization-Wide Defaults (OWD):

This screenshot shows the 'User Detail' page for a user named 'Patient Isaac'. The page includes fields for Name, Alias, Email, Username, Nickname, Title, Department, and various OWD settings such as 'Default Home Page', 'Default Record Type', 'Default Language', 'Default Time Zone', and 'Default Profile'. A note at the bottom states 'Default Approval Request From: Only if I am an approver'.

**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:



The screenshot shows the 'Sharing Settings' page in the Salesforce Setup. It lists several sharing rule categories:

- Work Type Sharing Rules:** No sharing rules specified.
- Work Type Group Sharing Rules:** No sharing rules specified.
- Doctor Sharing Rules:** No sharing rules specified.
- Patient Sharing Rules:** No sharing rules specified.
- Prescription Sharing Rules:** Action: Criteria: Owner in Role: Patient; Shared With: Role: Doctor; Access Level: Read/Write.
- TeleAppointment Sharing Rules:** Action: Criteria: Owner in Role: Patient; Shared With: Role: Doctor; Access Level: Read/Write.

**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 **24x7** 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:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the file `Order_Detail__object-meta.xml` selected.
- Terminal:** Displays the command `sfdx force:source:deploy -u [username]`.
- Salesforce Trail Project:** Shows the deployment status of the file, indicating it is being deployed to the developer org.
- Output Panel:** Shows deployment logs, including connection information and deployment progress.
- Right Sidebar:** Welcome to Copilot panel with deployment-related commands.

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the file `Order_Detail__object-meta.xml` selected.
- Terminal:** Displays the command `sfdx force:source:deploy -u [username]`.
- Salesforce CLI Integration:** Shows the deployment status of the file, indicating it is being deployed to the developer org.
- Output Panel:** Shows deployment logs, including connection information and deployment progress.
- Right Sidebar:** Welcome to Copilot panel with deployment-related commands.

- **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**

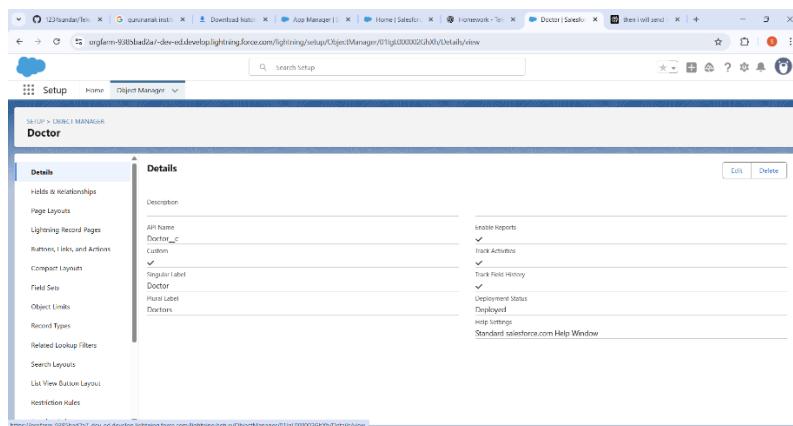
## Phase 3: Data Modeling & Relationships

### 1. Custom Objects:

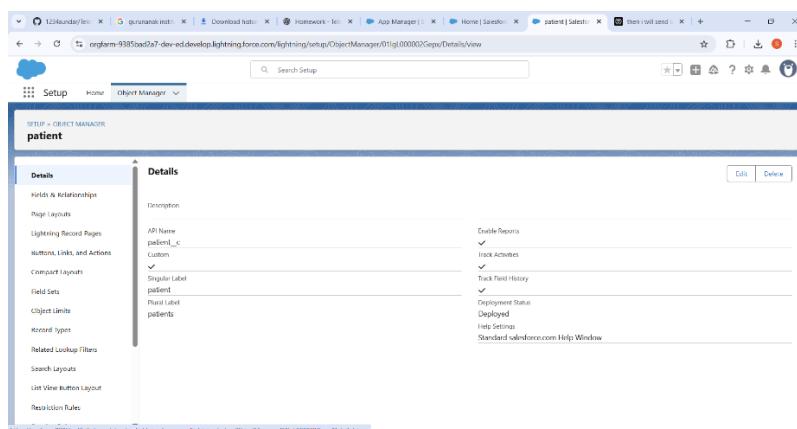
Created Custom Objects to support our use case:

- **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.

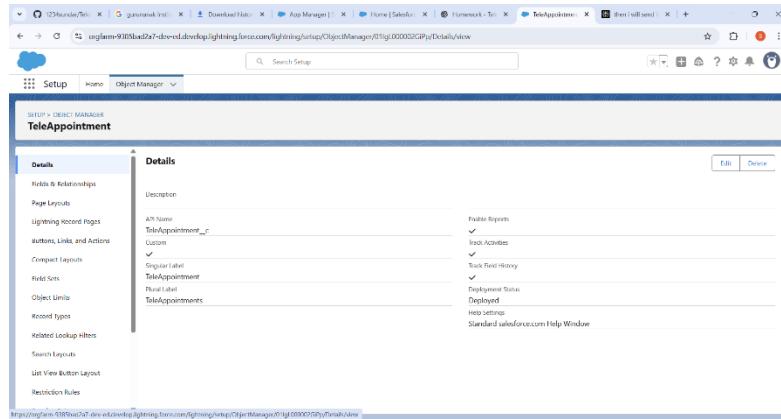
Doctor Object:



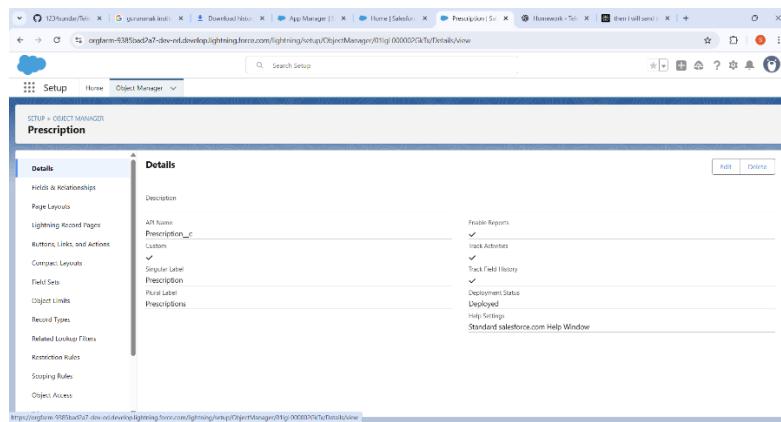
Patient Object:



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:

The image consists of six separate screenshots of the Salesforce Object Manager interface, each showing a different field configuration for the Doctor object. The fields shown are:

- Name**: A custom field definition for the Name field, set to a Text type with a maximum length of 255 characters.
- Availability**: A custom field definition for the Availability field, set to a Picklist type with options like Available, Busy, and Out of Office.
- Contact Number**: A custom field definition for the Contact Number field, set to a Text type with a maximum length of 255 characters.
- Specialty**: A custom field definition for the Specialty field, set to a Picklist type with options like Primary, Secondary, and Tertiary.
- Telemedicine Profile**: A custom field definition for the Telemedicine Profile field, set to a Picklist type with options like Enabled, Pending, and Disabled.
- Doctor Profile**: A profile page for the Doctor object, showing various field configurations and layout settings.

## b. Fields for the Patient Object are:

The image contains four separate screenshots of the Salesforce Setup interface, each showing a different step in the creation of fields for the Patient object.

- Screenshot 1:** Shows the "Fields & Relationships" tab for the Patient object. A new field is being defined under "Custom Field Definition Detail". The field name is "Contact Number", and its type is "Text". Other details like "Field Label" and "Data Type" are also visible.
- Screenshot 2:** Shows the continuation of the field creation process, with the "Custom Field Definition Detail" section still active. The field "Contact Number" is being configured with specific validation rules and field-level security.
- Screenshot 3:** Shows another view of the "Fields & Relationships" tab for the Patient object, where a new field "DOB" is being created. This screenshot shows the "Field Information" and "General Options" sections.
- Screenshot 4:** Shows the final configuration of the "DOB" field, including its "Field Label" ("Date of Birth") and "Data Type" ("Text"). It also shows the "Validation Rules" and "Field-Level Security" sections.

## c. Fields for the TeleAppointment Object are:

This screenshot shows the "Fields & Relationships" tab for the TeleAppointment object. It lists various fields such as Doctor, Duration\_Minutes\_c, Last\_Modified\_By, Location\_\_c, Notes, Owner, patient, Room\_Number\_c, Status, and Teleappointment\_Name. Each field is associated with a data type and a description.

Field Name	Type	Description
Doctor	Lookup(Doctor)	
Duration_Minutes_c	Number(0, 0)	
Last_Modified_By	Lookup(User)	
Location__c	Text(255)	
Notes	Long Text Area(1000)	
Owner	Lookup(Org Group)	
patient	Lookup(patient)	
Room_Number_c	Text(40)	
Status	Picklist	
Teleappointment_Name	Name	Auto Number

Like wise several fields are created for the TeleAppointment Object.

Fields for the Prescription Object are:

The screenshot shows the 'Object Manager' interface for the 'Prescription' object. Under the 'Fields & Relationships' tab, a table displays the following fields:

Field	Type	Description
Appointment	Appointment_c	Lookup(TeleAppointment)
Created By	CreatedById	Lookup(User)
Doctor	Doctor_c	Lookup(Doctor)
Last Modified By	LastModifiedById	Lookup(User)
Medicines	Medicines_c	Long Text Area(500)
Notes	Notes_c	Long Text Area(500)
patient	patient_c	Lookup(patient)
Prescription ID	Prescription_ID_c	Auto Number
Prescription name	Name	Text(50)
TeleAppointment	TeleAppointment_c	Master-Detail(TeleAppointment)

### 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

The screenshot shows the 'Setup' interface under the 'Page Layouts' tab for the 'TeleAppointment' object. It displays the 'TeleAppointment - In Person' layout. The layout configuration includes sections for Buttons, Details, and Fields & Relationships. The 'Fields & Relationships' section shows fields such as Appointment ID, Last Modified By, and patient.

#### B. Tele appointment Online:

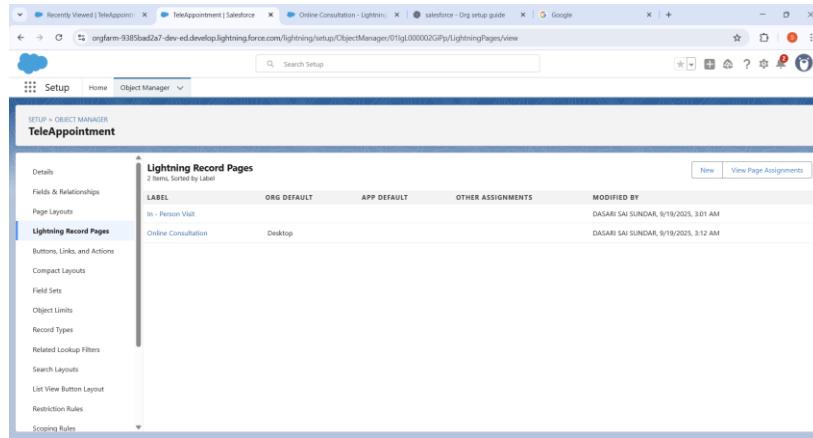
The screenshot shows the 'Setup' interface under the 'Page Layouts' tab for the 'TeleAppointment' object. It displays the 'TeleAppointment Detail' layout. The layout configuration includes sections for Buttons, Details, and Fields & Relationships. The 'Fields & Relationships' section shows fields such as Appointment ID, Last Modified By, and patient.

#### 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.

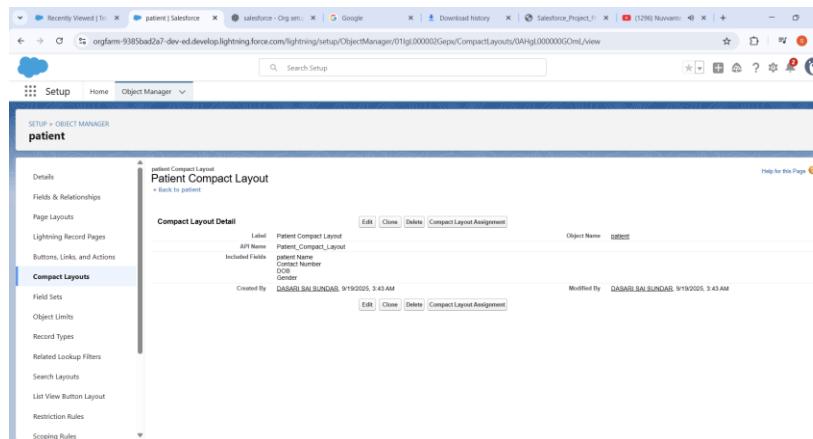


The screenshot shows the 'TeleAppointment' object in the Object Manager. Under the 'Lightning Record Pages' tab, there is one page assignment listed:

Label	Org Default	App Default	Other Assignments	Modified By
In - Person Visit	Online Consultation	Desktop		DASARI SAI SUNDAR, 9/19/2025, 3:01 AM

#### 5. Compact Layout of the Project:

##### a. Patient Compact Layout:



The screenshot shows the 'patient' object in the Object Manager. Under the 'Compact Layouts' tab, a new compact layout named 'Patient Compact Layout' is being edited. The layout detail shows the following configuration:

Label	API Name	Object Name
Patient Compact Layout	Patient_Compact_Layout	patient

Included Fields: First Name, Last Name, Contact Number, DOB, Gender.

Created By: DASARI SAI SUNDAR, 9/19/2025, 3:43 AM

Modified By: DASARI SAI SUNDAR, 9/19/2025, 3:43 AM

### b. Prescription Compact Layout:

The screenshot shows the Salesforce Setup interface with the Object Manager selected. Under the Prescription object, the Compact Layouts section is highlighted. The table displays two items:

LABEL	API NAME	PRIMARY	MODIFIED BY	LAST MODIFIED
Prescription Compact Layout	Prescription_Compact_Layout		DASARI SAI SUNDAR	9/19/2025, 3:49 AM
System Default	SYSTEM			

### c. Doctor Compact Layout:

The screenshot shows the Salesforce Setup interface with the Object Manager selected. Under the Doctor object, the Compact Layouts section is highlighted. The table displays two items:

LABEL	API NAME	PRIMARY	MODIFIED BY	LAST MODIFIED
Doctor Compact Layout	Doctor_Compact_Layout		DASARI SAI SUNDAR	9/19/2025, 3:46 AM
System Default	SYSTEM			

### d. TeleAppointment Compact Layout:

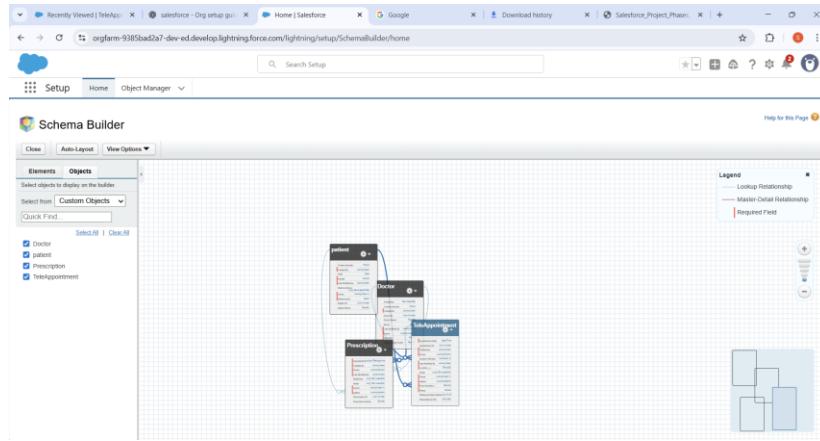
The screenshot shows the Salesforce Setup interface with the Object Manager selected. Under the TeleAppointment object, the Compact Layouts section is highlighted. The table displays two items:

LABEL	API NAME	PRIMARY	MODIFIED BY	LAST MODIFIED
System Default	SYSTEM			
TeleAppointment Compact Layout	TeleAppointment_Compact_Layout		DASARI SAI SUNDAR	9/19/2025, 3:48 AM

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:



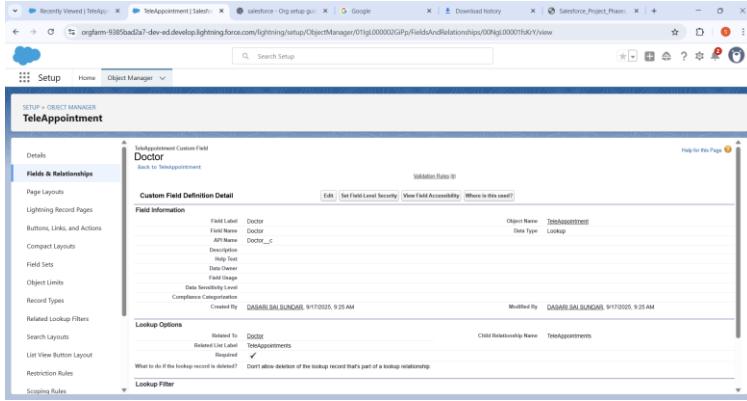
We used **Schema Builder** to visualize the entire data model. Mapped the relationships between Doctor, Patient, Appointment, and Medical Record .Ensured correct lookup and master-detail dependencies. Adjusted field placements for clarity.

## 7.Relationships:

### 1. The TeleAppointment Lookup relation with the Patient Object

This screenshot shows the 'Object Manager' for the 'TeleAppointment' object. On the left, there's a sidebar with various tabs like 'Details', 'Fields & Relationships', 'Page Layouts', etc. Under 'Fields & Relationships', the 'patient' field is selected. The right panel shows the 'Custom Field Definition Detail' for the 'patient' field. It includes sections for 'Field Information' (Field Label: patient, Field Name: patient, API Name: patient\_\_c, Description: null, Help Text: null, Field Usage: null), 'Validation Rules' (none listed), and 'Lookup Options'. In the 'Lookup Options' section, it shows 'Related To' as 'patient', 'Related List Label' as 'TeleAppointments', 'Required' checked, and a note: 'What do if the lookup record is deleted? Don't allow deletion of the lookup record that's part of a lookup relationship.' The page also shows standard metadata like 'Object Name: TeleAppointment', 'Data Type: Lookup', and 'Created By: DASARI SAI SUNDARA, 9/17/2025, 9:29 AM'.

### 2. Lookup relation between the TeleAppointment to the Doctor.



SETUP - OBJECT MANAGER

**TeleAppointment**

Custom Field Definition Detail

Field Information

Object Name: **TeleAppointment**

API Name: **Doctor**

Description: **Doctor**

Created By: **DABANI SAI SUNDAR** 9/17/2020, 9:25 AM

Modified By: **DABANI SAI SUNDAR** 9/17/2020, 9:25 AM

Lookup Options

Related To: **TeleAppointment**

Child Relationship Name: **TeleAppointments**

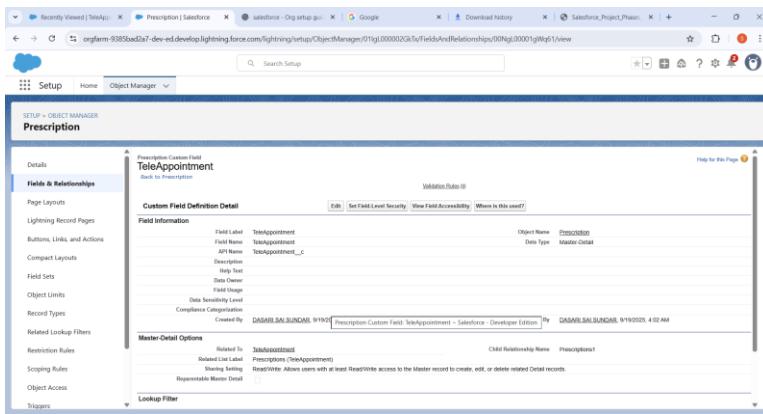
Master-Detail Options

Related To: **TeleAppointment**

Child Relationship Name: **Prescriptions**

Sharing Setting: **ReadWrite**: Allows users with at least Read/Write access to the Master record to create, edit, or delete related Detail records.

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



SETUP - OBJECT MANAGER

**Prescription**

Custom Field Definition Detail

Field Information

Object Name: **Prescription**

API Name: **TeleAppointment\_\_c**

Description: **TeleAppointment**

Created By: **DABANI SAI SUNDAR** 9/19/2020

Modified By: **DABANI SAI SUNDAR** 9/19/2020, 4:02 AM

Master-Detail Options

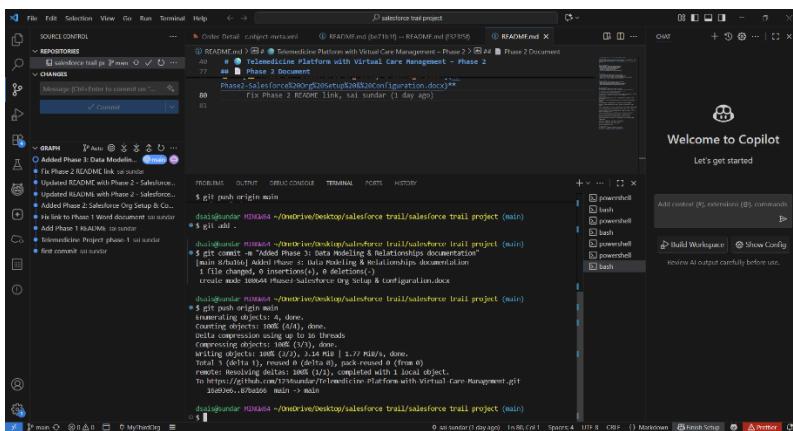
Related To: **TeleAppointment**

Child Relationship Name: **Prescriptions**

Sharing Setting: **ReadWrite**: Allows users with at least Read/Write access to the Master record to create, edit, or delete related Detail records.

## 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.



```
git push origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Writing objects: 100% (4/4), done.
total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/DabaniSaiSundar/Salesforce-Platform-with-Virtual-Care-Management.git
   3e3a4d9..6a344f3  Phase 2 README <--> Phase 2 README
```

**Salesforce IDE Project**

The screenshot shows two instances of the Salesforce IDE interface. Both windows have the title "salesforce trial project".

**Left Window (Top):**

- File Explorer:** Shows the project structure under "MyHerding". It includes "Order\_Detail\_\_cobject-met.xml", "README.md", and "Phase 2 Document".
- Terminal:** Displays the command-line output for "sfdx force:source:retrieve -u salesforcetrial -r Phase 2 Document".
- Output:** Shows retrieved sources for "Account" and "Appointment".
- Right Panel:** "Welcome to Copilot" sidebar with "Let's get started" and "Add context (F1) extensions (G), commands" buttons.

**Bottom Status Bar:**

- File: more, Open, Save, Undo, Redo, MyHerding
- Editor: 1 file, 1 tab, 1 day ago
- Terminal: 1 file, 1 tab, 1 day ago
- Output: 1 file, 1 tab, 1 day ago
- Search: 4 results, 4 tabs, 4 files
- Code: 4 tabs, 4 files
- Markdown: 1 file, 1 tab, 1 day ago
- Preview: 1 file, 1 tab, 1 day ago

**Right Window (Bottom):**

- File Explorer:** Shows "REPOSITORY", "CHANGES", and "GRAIN".
- Terminal:** Displays the command-line output for "sfdx force:source:deploy -u salesforcetrial -m Phase 2 Document".
- Output:** Shows deployment logs for "Phase 2 Document".
- Right Panel:** "Welcome to Copilot" sidebar with "Let's get started" and "Add context (F1) extensions (G), commands" buttons.

**Bottom Status Bar:**

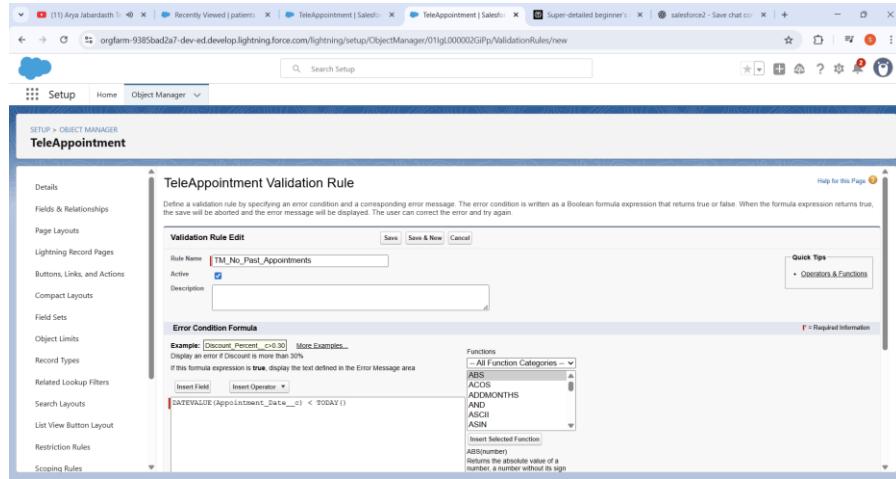
- File: more, Open, Save, Undo, Redo, MyHerding
- Editor: 1 file, 1 tab, 1 day ago
- Terminal: 1 file, 1 tab, 1 day ago
- Output: 1 file, 1 tab, 1 day ago
- Search: 4 results, 4 tabs, 4 files
- Code: 4 tabs, 4 files
- Markdown: 1 file, 1 tab, 1 day ago
- Preview: 1 file, 1 tab, 1 day ago

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

# Phase 4: Process Automation (Admin)

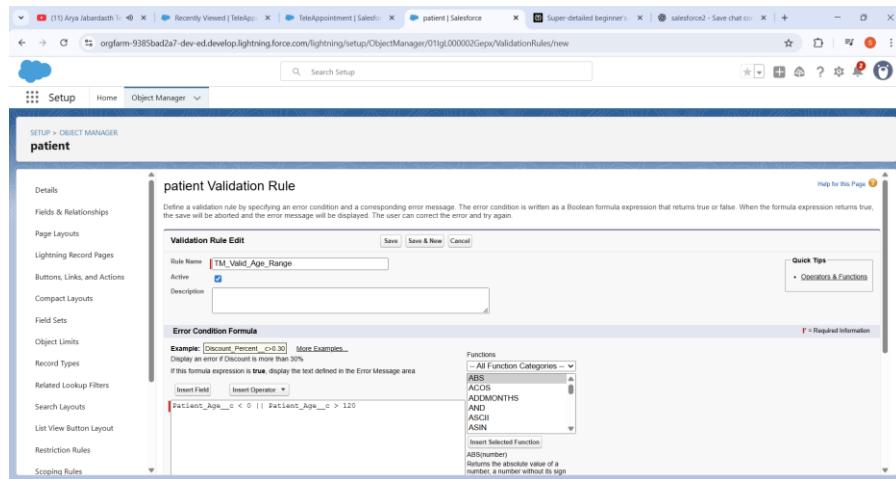
## 1. Validation rule:

### a) No Past Appointments:



The screenshot shows the 'TeleAppointment Validation Rule' configuration screen. The 'Validation Rule Edit' section includes the rule name 'TM\_No\_Past\_Appointments', which is active. The 'Error Condition Formula' field contains the formula `DATEVALUE(Appointment_Date_c) < TODAY()`. A tooltip for the formula indicates it returns the absolute value of a number, a number without its sign.

### b) Patient Valid Age Range:



The screenshot shows the 'patient Validation Rule' configuration screen. The 'Validation Rule Edit' section includes the rule name 'TM\_Valid\_Age\_Range', which is active. The 'Error Condition Formula' field contains the formula `Patient_Age__c < 0 || Patient_Age__c > 120`. A tooltip for the formula indicates it returns the absolute value of a number, a number without its sign.

## 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.

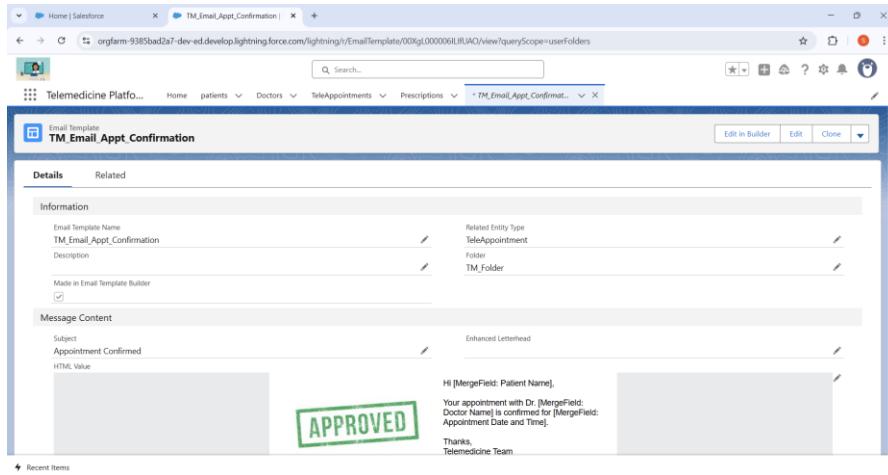
- **TM\_No\_Past\_Appointments:** This rule prevents a user from creating a TeleAppointment with a date that is in the past. It uses the formula

`DATEVALUE(Appointment_Date_c) < TODAY()`.

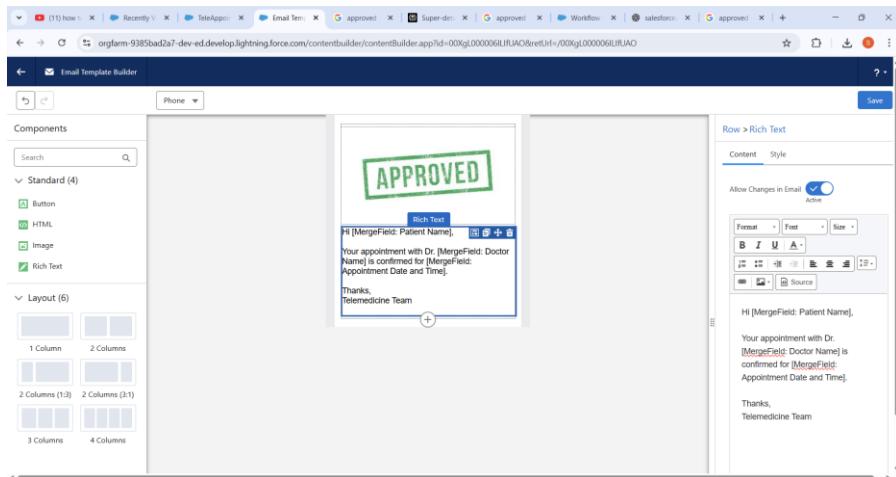
- **TM\_Valid\_Age\_Range:** This rule ensures a Patient's age is within a valid range. It uses the formula  $\text{Patient\_Age\_c} < 0 \parallel \text{Patient\_Age\_c} > 120$ .

## 2.Lightning Email Template:

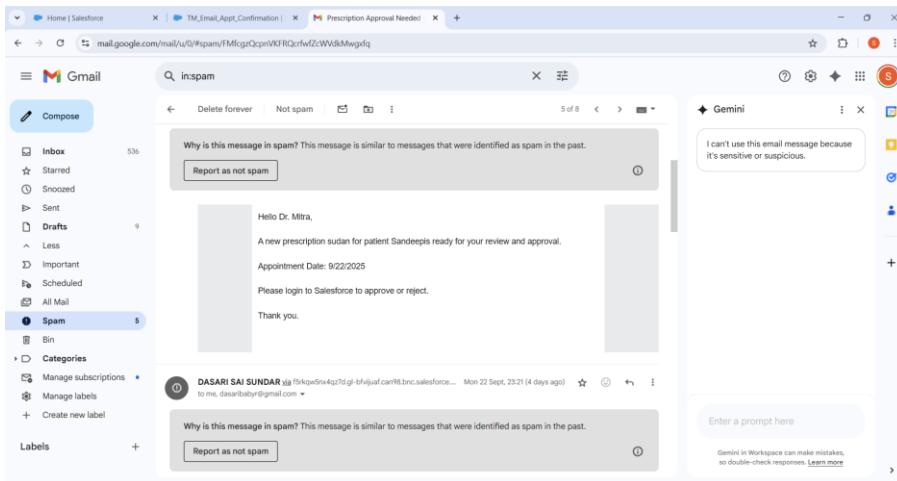
- a) Created lightning Email Template → Email Appointment confirmation:



- b) Design the Email Template with the builder:



### c) Test Case1:

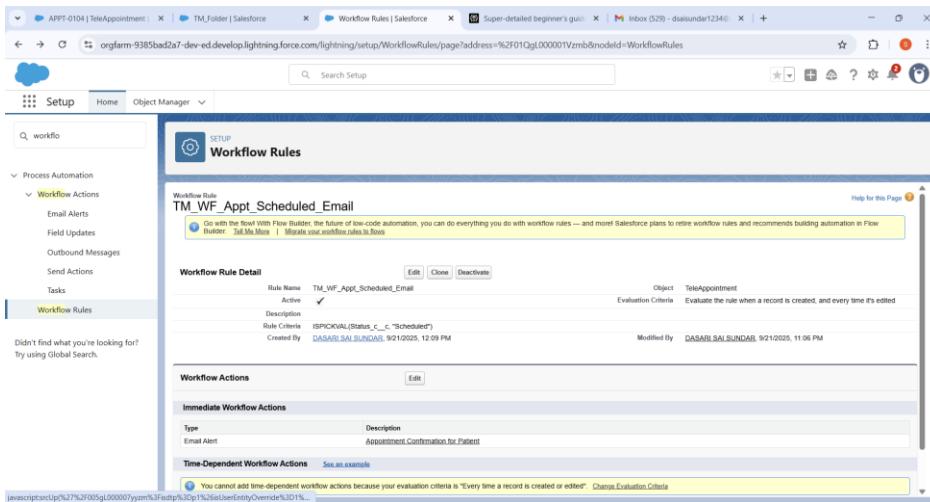


The email template is created with the above structure.

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.

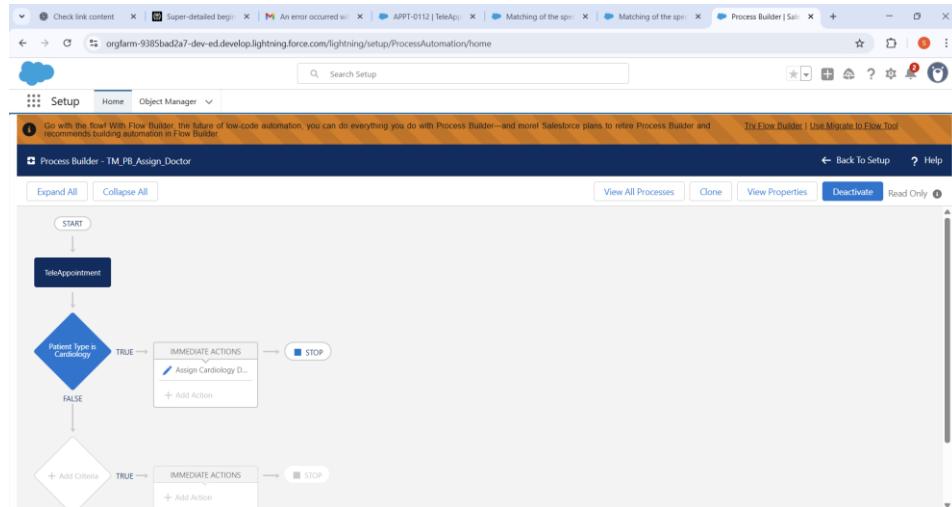
- **TM\_WF\_Appt\_Scheduled\_Email:** This rule automatically sends an email to the patient when a TeleAppointment's status is set to "Scheduled."

- The email that gets sent uses a

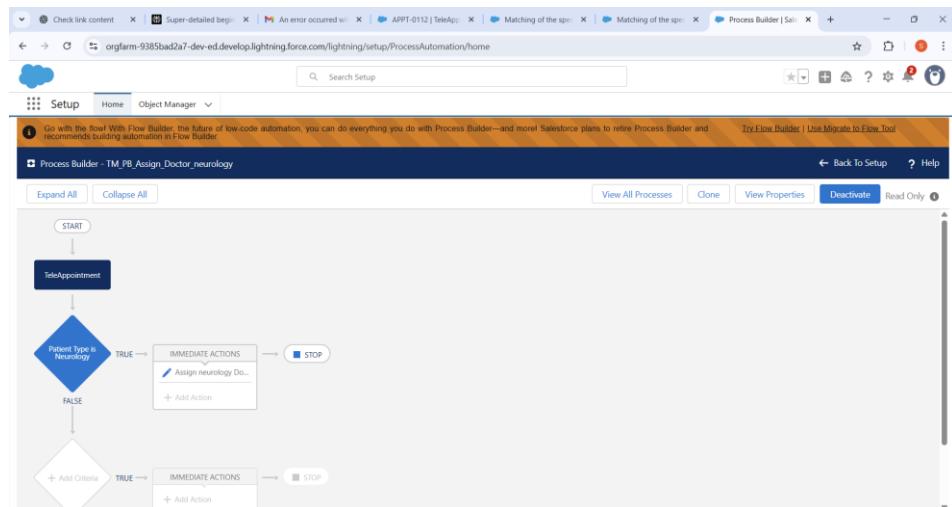
Lightning Email Template which includes an "APPROVED" stamp and merge fields for the Patient Name, Doctor Name, and Appointment Date and Time.

## 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.

The screenshot shows the Salesforce Process Builder home page. At the top, there's a banner with the text "Go with the flow! With Flow Builder, the future of low-code automation, you can do everything you do with Process Builder—and more! Salesforce plans to retire Process Builder and recommends building automation in Flow Builder." Below the banner, there are links to "Try Flow Builder" and "Use Migrate to Flow Tool". The main area is titled "My Processes" and contains a table with four rows. The columns are labeled: PROCESS, DESCRIPTION, OBJECT, PROCESS TYPE, LAST MODIFIED, STATUS, and ACTIONS. The processes listed are:

PROCESS	DESCRIPTION	OBJECT	PROCESS TYPE	LAST MODIFIED	STATUS	ACTIONS
TM_PB_ASSIGN_DOCTOR	Auto-assigns a doctor to TeleAppointment based on ...	TeleAppointment	Record Change	9/22/2025	Active	
TM_PB_ASSIGN_DOCTOR_GENERAL		TeleAppointment	Record Change	9/22/2025	Active	
TM_PB_ASSIGN_DOCTOR_NEUROLOGY		TeleAppointment	Record Change	9/22/2025	Active	
TM_PB_ASSIGN_DOCTOR_PEDIATRICS		TeleAppointment	Record Change	9/22/2025	Active	

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:

- Created the Approval Process:

The screenshot shows the Approval Processes setup screen for a process named "TM\_Prescription\_Approval". The "Process Definition Detail" section includes the following information:

- Process Name: TM\_Prescription\_Approval
- Unique Name: TM\_Prescription\_Approval
- Description: Prescription Approval
- Criteria: ORIS\_Restricted\_c = TRUE, Total\_Cost\_\_c > 5000
- Administrators ONLY
- Record Editability: Active
- Allow Submitters to Recall Approval Requests: Unchecked
- Approval Assignment Email Template: Initial Submitter
- Initial Submitter: TeleAppointment Owner
- Created By: DASARI SAI SUNDAR, 9/22/2025, 3:35 AM
- Modified By: DASARI SAI SUNDAR, 9/22/2025, 3:35 AM

The "Initial Submission Actions" section contains an action: Record Lock, with a description: Lock the record from being edited. The "Approval Steps" section is currently empty.

b) Initial Submission Action:

- Email Alert alert is created with the name Please Approve

The screenshot shows the 'Email Alerts' setup page. A specific alert named 'Please approve' is selected. The alert details include:

- Description:** Please approve
- Unique Name:** Approval\_Request\_of\_the\_prescription
- From:** Doctor's email address
- To:** Admin, User: DASARI SAI SUNDAR
- Additional Emails:** dasaraisb@gmail.com
- Created By:** DASARI SAI SUNDAR, 9/22/2025, 3:37 AM
- Modified By:** DASARI SAI SUNDAR, 9/22/2025, 3:37 AM

Below the alert detail, there are sections for 'Rules Using This Email Alert' and 'Approval Processes Using This Email Alert'. The 'Approval Processes Using This Email Alert' section lists two processes:

Action	Approval Process Name	Description	Type	Status
Edit   Del	TM_Prescription_Approval		Prescription	Obsolete
Edit   Del	TM_Prescription_Approval_1		Prescription	Active

c) Approval Steps:

- Two approval steps created with names 1.Doctor Review ,2.Department head Approval
- Formula-1: Prescription: Approval Status equals Approved by Doctor , else Approve
- Formula-2: Prescription: Total Cost GREATER OR EQUAL 5000

The screenshot shows the 'Approval Processes' setup page. It displays two approval steps:

- Step 1: Doctor Review**
  - Description:** Prescription: Approval status equals Approved by Doctor , else Approve
  - Assigned Approver:** User: Dr. Shyam
  - Reject Behavior:** Final Rejection
- Step 2: Department Head Approval**
  - Description:** Prescription: Total Cost GREATER OR EQUAL 5000
  - Assigned Approver:** User: DASARI SAI SUNDAR
  - Reject Behavior:** Final Rejection

Each step has sections for 'Approval Actions' and 'Rejection Actions', both currently empty.

d) Final Approval Action:

- Created the Please Approve

The screenshot shows the Salesforce Setup interface with the search bar set to 'approval'. Under 'Email Alerts', a new alert named 'Please approve' is being created. The alert is triggered by 'Approval\_Request' and has the unique name 'Approval\_Request\_\_EmailAlert'. It uses the 'Approval\_Status' object and the 'Prescription' email template. The alert's description is 'Please approve Approval\_Status\_\_r.prescription'. The 'From Email Address' is 'Current User's email address'. Recipients are 'User: DASARI SAI SUNDAR' and 'Additional Email: dasari.sai.sundar@luminous.com'. The alert was created by 'DASARI SAI SUNDAR' on 9/22/2025 at 3:37 AM and modified by the same user on the same date and time. Below the alert detail, there is a section for 'Rules Using This Email Alert' which is currently empty.

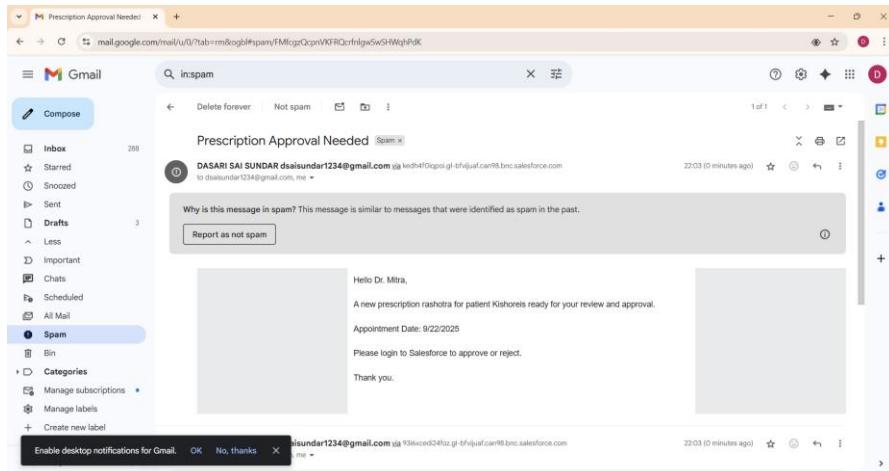
- A field update created in the Final Approval process

The screenshot shows the Salesforce Setup interface with the search bar set to 'approval'. Under 'Field Updates', a new update named 'Approve\_S' is being created. The update is triggered by 'Approval\_Request' and has the unique name 'Approval\_Request\_\_FieldUpdate'. It uses the 'Prescription' object and the 'Prescription\_Approval\_Status' field. The update's description is 'Prescription\_Approval\_Status: PONR'. The 'From Field' is 'Field' and the 'To Field' is 'Field'. The 'Re-evaluates Workflow Rules after Field Change' checkbox is checked. The new field value is 'Submitted'. Below the field update detail, there is a section for 'Rules Using This Field Update' which is currently empty.

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

The screenshot shows the Salesforce Setup interface with the search bar set to 'approval'. Under 'Approval Processes', a new process is being created. The process has two steps: 'Step 1: Doctor Review' and 'Step 2: Department Head Approval'. Step 1 has the criteria 'TelAppointments: Status equals Completed, else Approve'. Step 2 has the criteria 'Prescription: Total Cost greater or equal 3000'. Both steps are assigned to 'User: D. Shyam'. The process also includes 'Final Approval Actions' (Record Lock, Email Exit, Field Update) and 'Final Rejection Actions' (Record Lock, Email Exit, Field Update). The 'Reject Behavior' for both steps is 'Final Rejection'. Below the process detail, there is a section for 'Recall Actions' (Record Lock).

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

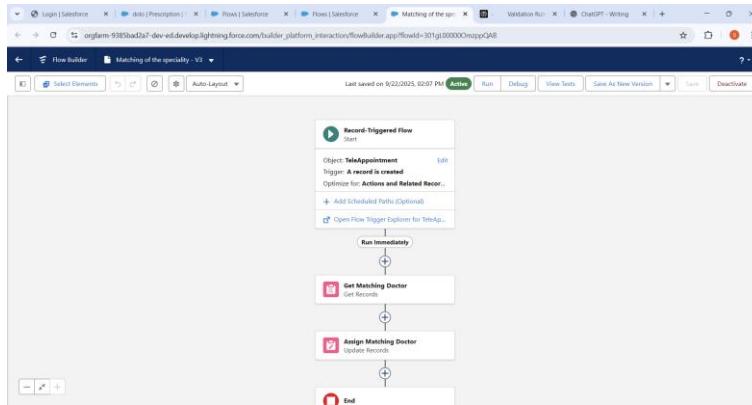


## Approval Process

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

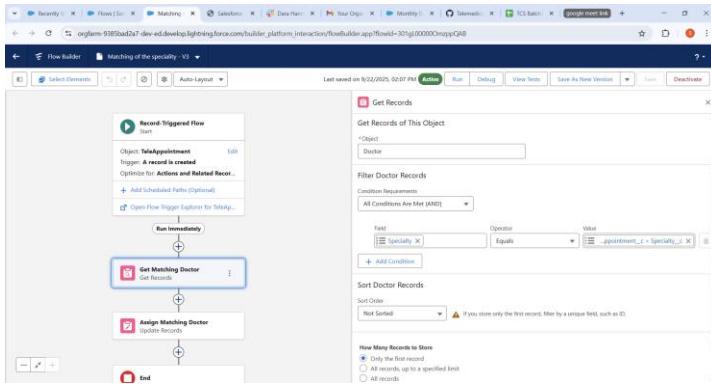
- **TM\_Prescription\_Approval:** This process is for approving prescriptions. It's triggered when a prescription is restricted or the total cost is over \$5,000.
- The approval process has two steps: a "Doctor Review" and a "Department Head Approval."
- When a record is submitted for approval, an email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

## 8.Flow:

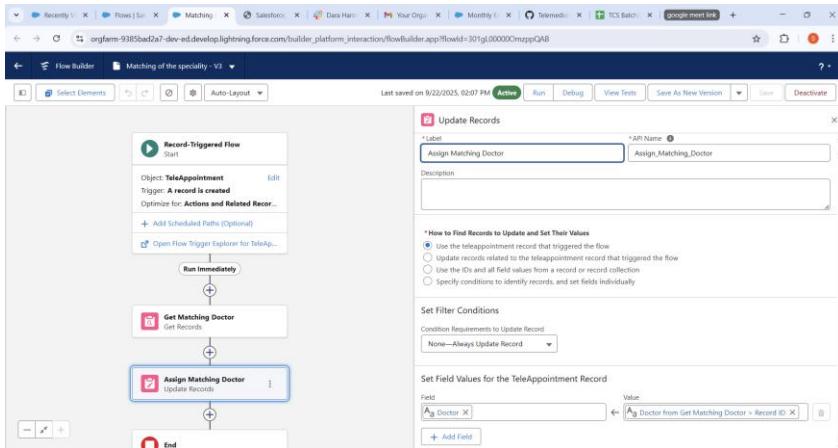


Flow is a visual automation tool that can perform everything a Process Builder or Workflow Rule can, and much more.

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

patient Name: Govardhan

Patient ID: PAT-007

Contact Number: 289995

DOB: 9/25/2025

Gender: Male

Medical History:

Patient email: bethapudi.govardhan@swaneshbgl.com

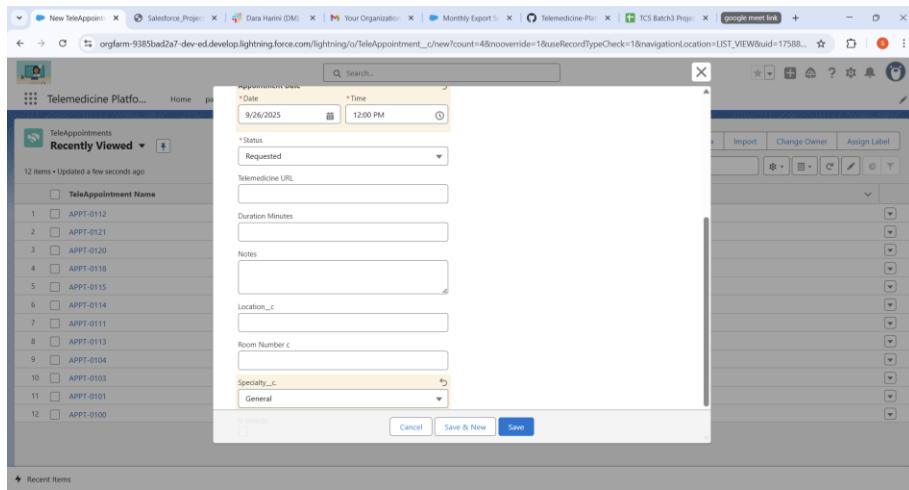
Patient Age:

Created By: DASARI SAI SUNDAR, 9/25/2025, 1:39 AM

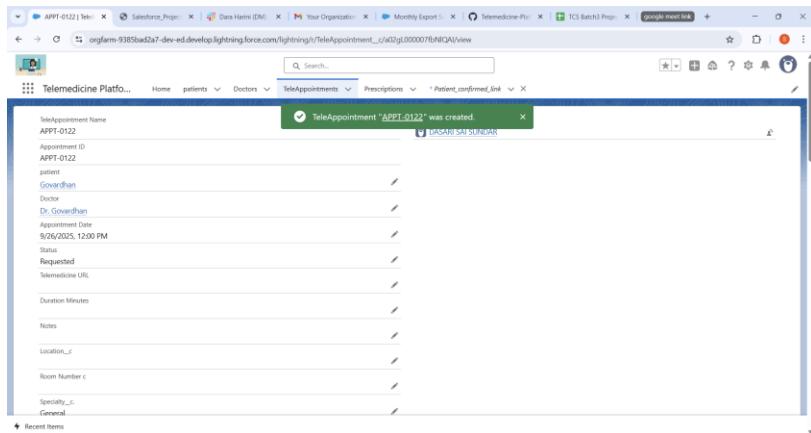
Last Modified By: DASARI SAI SUNDAR, 9/25/2025, 1:39 AM

Medical History: No history available

- Just selected the Specialization not doctor

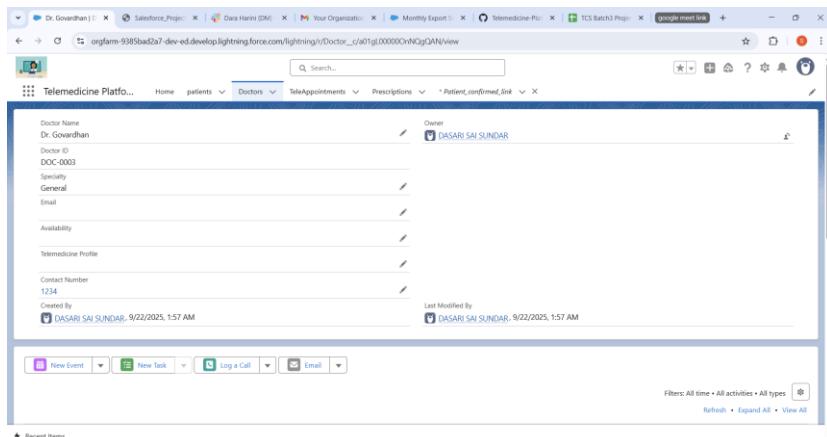


- After saving Directly Assigned



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:

The screenshot shows the Salesforce Setup Apex Classes page. The search bar at the top has "apex" entered. The left sidebar shows categories like Email, Custom Code, Apex Classes, Apex Settings, Apex Test History, Apex Triggers, Environments, and Jobs. The "Apex Classes" section is selected. The main area displays the Apex Class Detail for "PatientService.cls". The code editor contains the following Apex code:

```

1  public class PatientService {
2      public static void createPatientRecord(Patient__c patient) {
3          Patient__c p = new Patient__c();
4          p.Name = patient.Name;
5          p.DOB__c = Date.newInstance(1990, 5, 12);
6          p.Medical_History__c = 'Diabetes';
7          p.Gender__c = 'Male';
8          p.Patient_email__c = 'raviltest.com';
9          p.Id = Database.insert(p);
10         if (p.Id != null) {
11             try {
12                 Database.insert(p);
13             } catch (DatabaseException e) {
14                 System.debug('Error creating patient record: ' + e.getMessage());
15             }
16         }
17     }
18 }

```

Apex Class → PatientService.cls Created

The screenshots show the Salesforce Developer Console interface. The top bar includes tabs for File, Edit, Debug, Test, Workbench, Help, and Log executeAnonymous (9/23/2025, 3:10:27 PM). The main area is titled "Execution Log".

**Screenshot 1:** Shows the "Enter Apex Code" dialog with the following Apex code:

```

1 Patient__c testPatient = new Patient__c();
2 testPatient.Name = 'Ravi';
3 testPatient.DOB__c = Date.newInstance(1990, 5, 12);
4 testPatient.Medical_History__c = 'Diabetes';
5 testPatient.Gender__c = 'Male';
6 testPatient.Patient_email__c = 'raviltest.com';
7
8 Id newPatientId = PatientService.createPatientRecord(t);
9 System.debug('New Patient Id: ' + newPatientId);
10

```

**Screenshot 2:** Shows the "Execution Log" table with the following data:

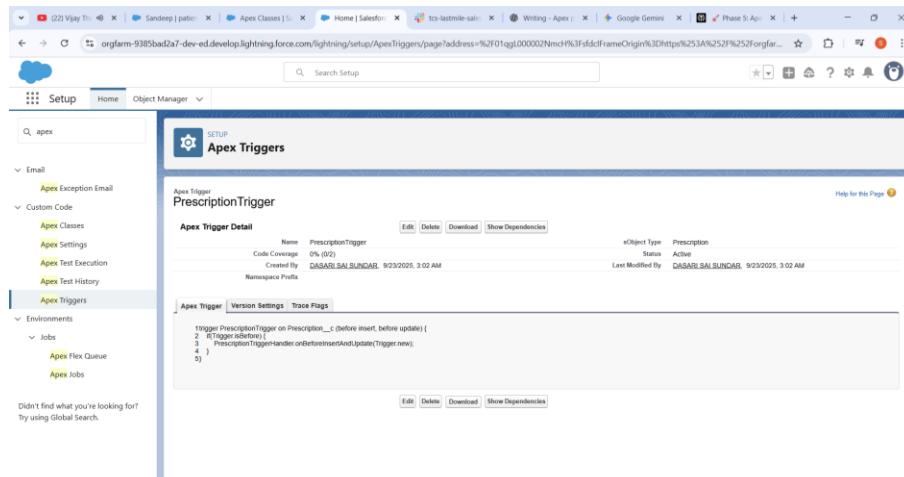
User	Application	Operation	Time	Status	Read	Size
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:10:27 PM	Success	Unread	5.97 KB
DASARI SAI SUNDAR	Browser	/amp/build/testApexClass.apexp	9/23/2025, 3:10:27 PM	Success	Unread	1.05 KB
DASARI SAI SUNDAR	Browser	/amp/testApexClass.apexp	9/23/2025, 3:10:27 PM	Success	Unread	0.84 KB
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:06:51 PM	Failed to create patient record: Requires... Unread	10.49 KB	
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:06:51 PM	Failed to create patient record: Requires... Unread	10.7 KB	
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:05:20 PM	Failed to create patient record: Requires... Unread	10.18 KB	

**Screenshot 3:** Shows the "Execution Log" table with the following data:

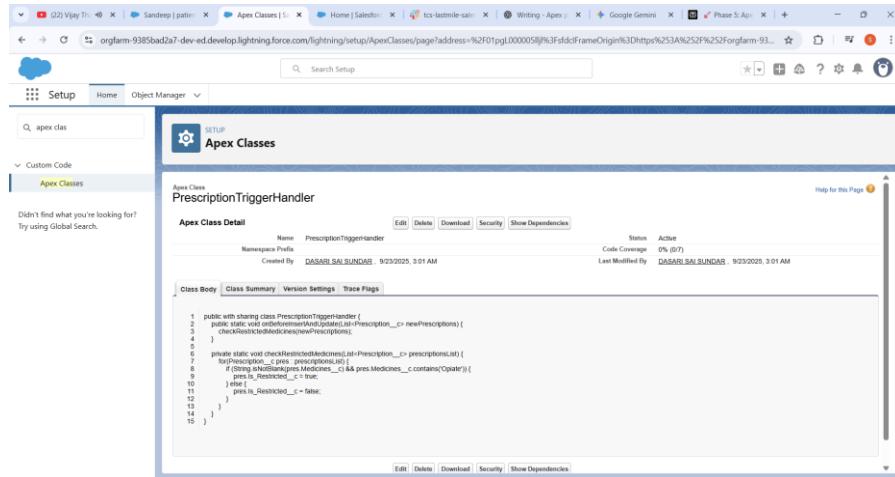
User	Application	Operation	Time	Status	Read	Size
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:10:27 PM	Success	Unread	5.97 KB
DASARI SAI SUNDAR	Browser	/amp/build/testApexClass.apexp	9/23/2025, 3:10:27 PM	Success	Unread	1.05 KB
DASARI SAI SUNDAR	Browser	/amp/testApexClass.apexp	9/23/2025, 3:10:27 PM	Success	Unread	0.84 KB
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:08:01 PM	Failed to create patient record: Requires... Unread	10.49 KB	
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:06:53 PM	Failed to create patient record: Requires... Unread	10.7 KB	
DASARI SAI SUNDAR	Unknown	/services/data/v44.0/tooling/queryAll	9/23/2025, 3:05:20 PM	Failed to create patient record: Requires... Unread	10.18 KB	

- **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.

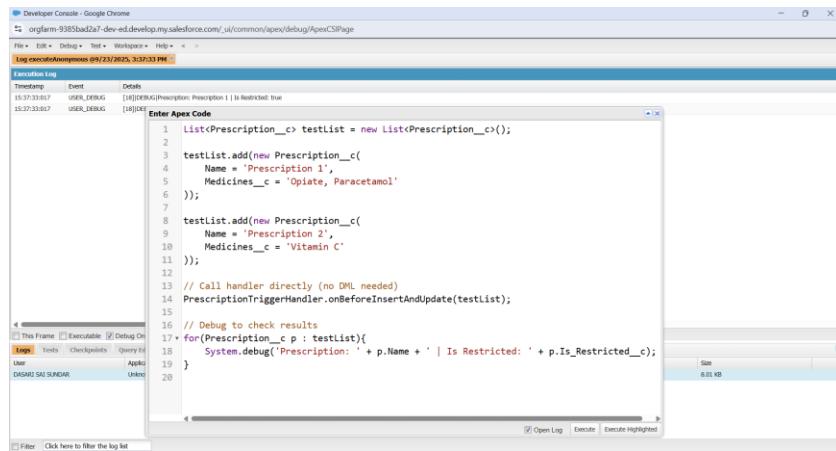
## 2. Apex Trigger and Handler:



## Prescription Trigger → Apex Trigger for the prescription



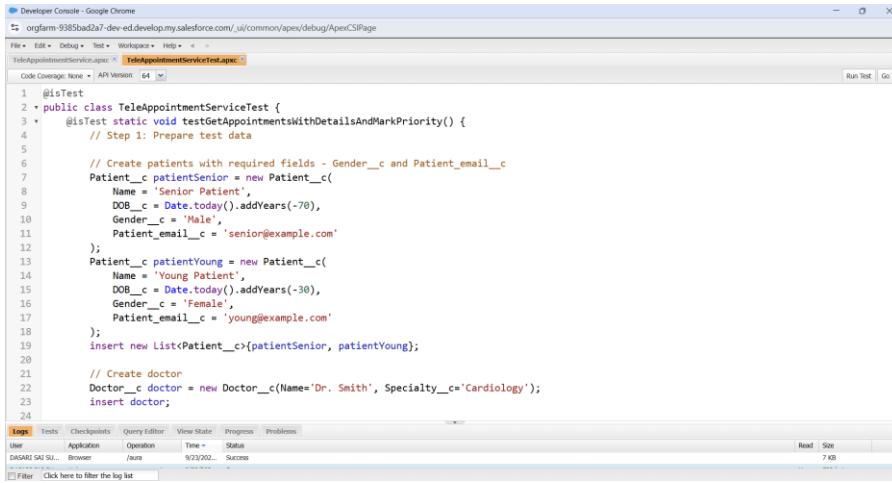
## Prescription TriggerHandler → Apex class for the Trigger Handling



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 to `PrescriptionTriggerHandler.onBeforeInsertAndUpdate(Trigger.new)`.
- Handler loops through each record in `Trigger.new` and checks the `Medicines__c` field.
- If the medicines contain 'Oipate', 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:

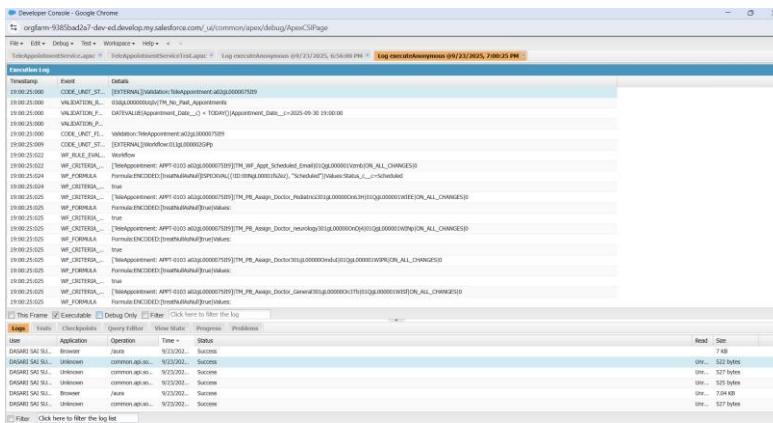


```

1 @isTest
2 public class TeleAppointmentServiceTest {
3     @isTest static void testGetAppointmentsWithDetailsAndMarkPriority() {
4         // Step 1: Prepare test data
5
6         // Create patients with required fields - Gender__c and Patient_email__c
7         Patient__c patientSenior = new Patient__c(
8             Name = 'Senior Patient',
9             DOB__c = Date.today().addYears(-70),
10            Gender__c = 'Male',
11            Patient_email__c = 'senior@example.com'
12        );
13        Patient__c patientYoung = new Patient__c(
14            Name = 'Young Patient',
15            DOB__c = Date.today().addYears(-30),
16            Gender__c = 'Female',
17            Patient_email__c = 'young@example.com'
18        );
19        insert new List<Patient__c>{patientSenior, patientYoung};
20
21        // Create doctor
22        Doctor__c doctor = new Doctor__c(Name='Dr. Smith', Specialty__c='Cardiology');
23        insert doctor;
24
25    }
26 }
```

The screenshot shows the Salesforce Developer Console interface. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a dropdown for 'API Version' set to 'v4'. Below the navigation is a tabs section with 'TeleAppointmentServiceTest.apc' selected. The main area contains the Apex code for the test class. At the bottom, there's a log viewer tab titled 'Logs' showing a single entry for a successful test run.

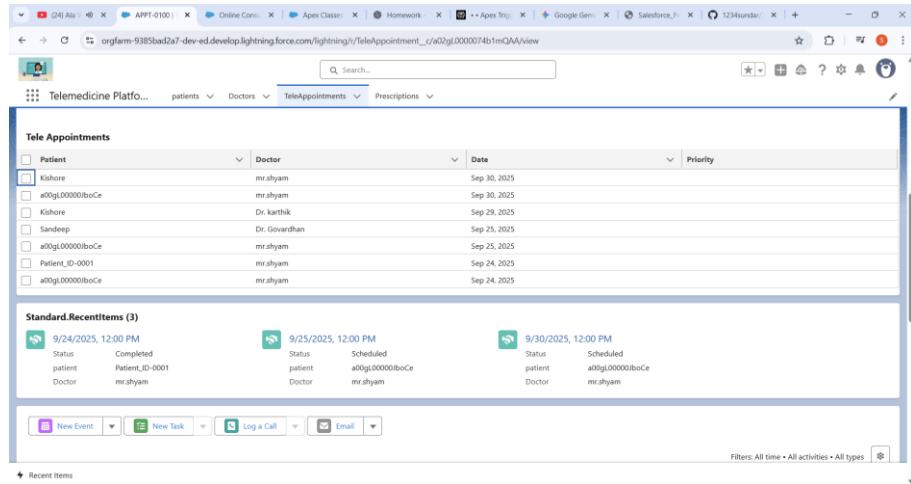
Teleappointment Service and Service test is created



The screenshot shows the Salesforce Developer Console interface with the 'Logs' tab selected. The log output displays several entries related to the execution of the test class, including API calls, validation rules, and scheduled apex triggers. The log ends with a success message for the test run.

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

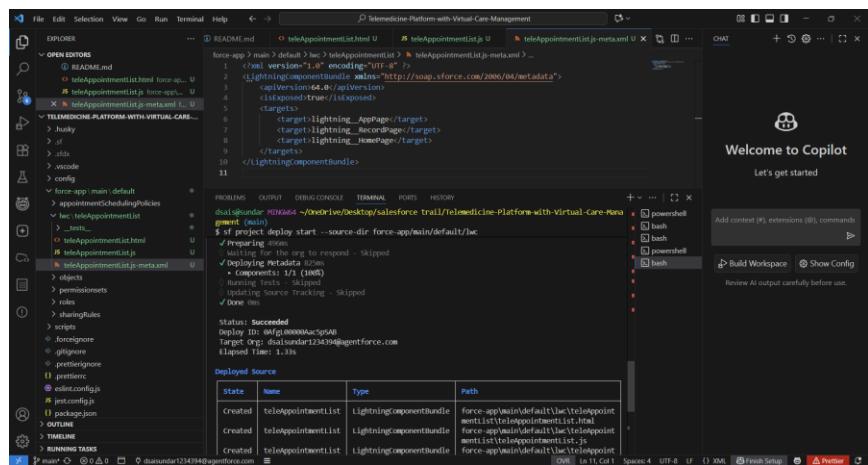
## 4. Lightning Web Component (LWC):



**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.

When 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

The screenshot shows the Salesforce Developer Console in Google Chrome. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a back arrow. The main area has tabs for 'AppointmentManager.apex' and 'Log execution log (9/24/2025, 12:05:09 AM)'. The 'Execution Log' tab displays a table of log entries with columns for Timestamp, Event, and Details. The 'Test' tab shows a table of test results with columns for User, Application, Operation, Time, Status, Read, and Size. One test, 'D4640 SAI SUNDAR', failed with the message 'Assertion Failed: Doctor\_\_r must be p...'.

Timestamp	Event	Details
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled
00:05:50:667	USER_DEBUG	[14]DEB[0]Apartment a102.00000000181pqk Status: Scheduled

User	Application	Operation	Time	Status	Read	Size
D4640 SAI SUNDAR	Unknown	ApptScheduler	9/23/2025, 12:05:09 AM	Success	<1.13 KB	28.02 KB
D4640 SAI SUNDAR	Unknown	ApptScheduler	9/23/2025, 11:51:00 PM	Assertion Failed: Doctor__r must be p...	Unread	28.02 KB

## 4. Asynchronous Apex:

### a) Create a Queueable Apex Class with name: ProcessPatientJob

The screenshot shows the Salesforce Developer Console in Google Chrome. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a back arrow. The main area has tabs for 'ProcessPatientJob.apex' and 'Log execution log (9/24/2025, 12:05:09 AM)'. The code editor shows the implementation of the Queueable interface. The 'Test' tab shows a table of test results with columns for User, Application, Operation, Time, Status, Read, and Size. One test, 'D4640 SAI SUNDAR', failed with the message 'Assertion Failed: Doctor\_\_r must be p...'.

```

1 public class ProcessPatientJob implements Queueable {
2     private List<Patient__c> patients;
3
4     public ProcessPatientJob(List<Patient__c> patientList) {
5         this.patients = patientList;
6     }
7
8     public void execute(QueueableContext ctx) {
9         try {
10             for(Patient__c p : patients) {
11                 // Example: Update medical history asynchronously
12                 p.Medical_History__c = p.Medical_History__c + ' - Updated asynchronously';
13             }
14             update patients;
15         } catch(Exception e) {
16             System.debug('Error in ProcessPatientJob: ' + e.getMessage());
17         }
18     }
19 }

```

User	Application	Operation	Time	Status	Read	Size
D4640 SAI SUNDAR	Unknown	ApptScheduler	9/23/2025, 11:51:00 PM	Assertion Failed: Doctor__r must be p...	Unread	28.02 KB

### b) Create a Queueable Apex Class with name: ProcessPrescription

The screenshot shows the Salesforce Developer Console in Google Chrome. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a back arrow. The main area has tabs for 'ProcessPrescriptionJob.apex' and 'Log execution log (9/24/2025, 12:05:09 AM)'. The code editor shows the implementation of the Queueable interface. The 'Test' tab shows a table of test results with columns for User, Application, Operation, Time, Status, Read, and Size. One test, 'D4640 SAI SUNDAR', failed with the message 'Assertion Failed: Doctor\_\_r must be p...'.

```

1 public class ProcessPrescriptionJob implements Queueable {
2     private List<Prescription__c> prescriptions;
3
4     public ProcessPrescriptionJob(List<Prescription__c> prescList) {
5         this.prescriptions = prescList;
6     }
7
8     public void execute(QueueableContext ctx) {
9         try {
10             for(Prescription__c presc : prescriptions) {
11                 // Check if medicine is restricted and update accordingly
12                 if(presc.Medicines__c != null && presc.Medicines__c.contains('Opiate')) {
13                     presc.Is_Restricted__c = true;
14                 } else {
15                     presc.Is_Restricted__c = false;
16                 }
17             }
18             update prescriptions;
19         } catch(Exception e) {
20             System.debug('Error in ProcessPrescriptionJob: ' + e.getMessage());
21         }
22     }
23 }

```

User	Application	Operation	Time	Status	Read	Size
D4640 SAI SUNDAR	Unknown	ApptScheduler	9/23/2025, 11:51:00 PM	Assertion Failed: Doctor__r must be p...	Unread	28.02 KB

### c) Modify The Existing PatientService Class:

The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-938bad2a7-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSPage. The tabs at the top are ProcessPatientJob.apc, ProcessPrescriptionJob.apc, and PatientsService.apc (which is the active tab). The code editor contains the following Apex code:

```
1 * public class PatientsService {
2
3     // This method processes a list of Patient__c records asynchronously if large, else sync
4     public static void processPatientsBulkAsync(List<Patient__c> patients) {
5         if(patients.size() > 50) {
6             // Use async processing for large datasets
7             System.enqueueJob(new ProcessPatientJob(patients));
8         } else {
9             // Process synchronously for small datasets
10            update patients;
11        }
12    }
13 }
```

The bottom navigation bar has tabs for Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Problems tab is selected, showing no errors.

### d) Modify Your PrescriptionTriggerHandler:

The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-938bad2a7-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSPage. The tabs at the top are ProcessPatientJob.apc, ProcessPrescriptionJob.apc, and PrescriptionTriggerHandler.apc (which is the active tab). The code editor contains the following Apex code:

```
1 * public class PrescriptionTriggerHandler {
2
3     public static void onBeforeInsertAndUpdate(List<Prescription__c> prescriptions) {
4         if(prescriptions.size() > 50) {
5             // For large batches, use asynchronous processing
6             System.enqueueJob(new ProcessPrescriptionJob(prescriptions));
7         } else {
8             // For small batches, process synchronously
9             for(Prescription__c presc : prescriptions) {
10                 if(presc.Medicines__c != null && presc.Medicines__c.contains('Opiate')) {
11                     presc.Is_Restricted__c = true;
12                 } else {
13                     presc.Is_Restricted__c = false;
14                 }
15             }
16         }
17     }
18 }
```

The bottom navigation bar has tabs for Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Problems tab is selected, showing no errors.

### e) Create Test Classes with the AsyncProcessingTests:

The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-938bad2a7-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSPage. The tabs at the top are ProcessPatientJob.apc, ProcessPrescriptionJob.apc, and PrescriptionTriggerHandler.apc, followed by AsyncProcessingTests.apc (which is the active tab). The code editor contains the following Apex code:

```
1 @isTest
2 private class AsyncProcessingTests {
3
4     @isTest static void testPatientAsyncProcessing() {
5         // Create test patients
6         List<Patient__c> testPatients = new List<Patient__c>();
7         for(Integer i = 0; i < 5; i++) {
8             testPatients.add(new Patient__c(
9                 Name = 'Test Patient ' + i,
10                DOB__c = Date.today().addYears(-30),
11                Gender__c = 'Male',
12                Patient_email__c = 'test' + i + '@example.com',
13                Medical_History__c = 'Initial history'
14            ));
15        }
16        insert testPatients;
17
18        Test.startTest();
19        System.enqueueJob(new ProcessPatientJob(testPatients));
20        Test.stopTest();
21    }
22 }
```

The bottom navigation bar has tabs for Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Problems tab is selected, showing no errors.

## 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.

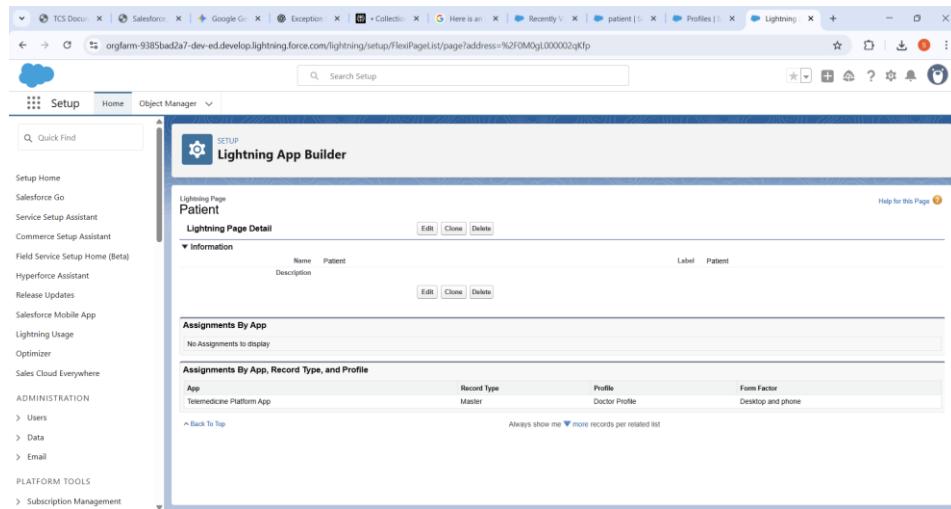
Together, these fit the best practices for asynchronous Apex in your Salesforce project and have been tailored specifically for your custom objects (Patient\_\_c and Prescription\_\_c).

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

# User interface Process

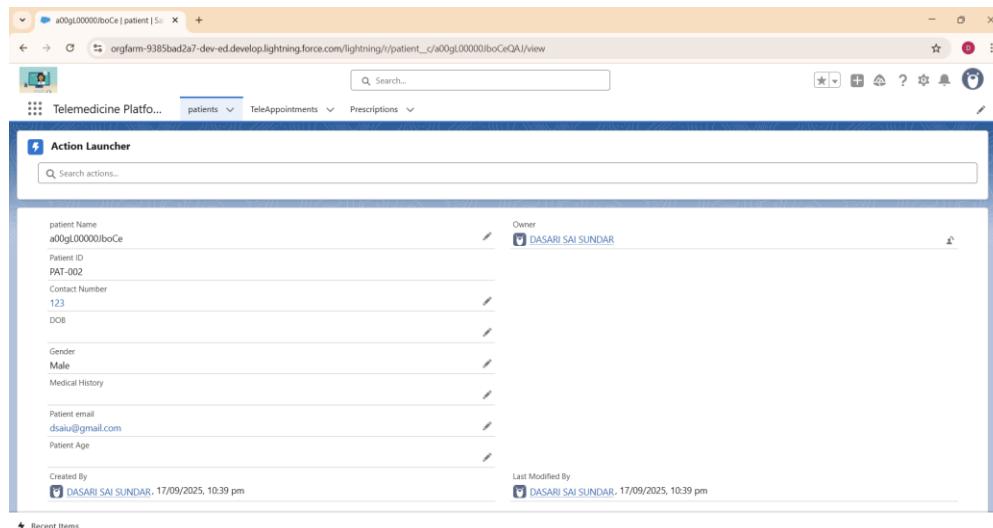
## 1. Create Custom Lightning Record Pages:

Create the patient lightning record pages:



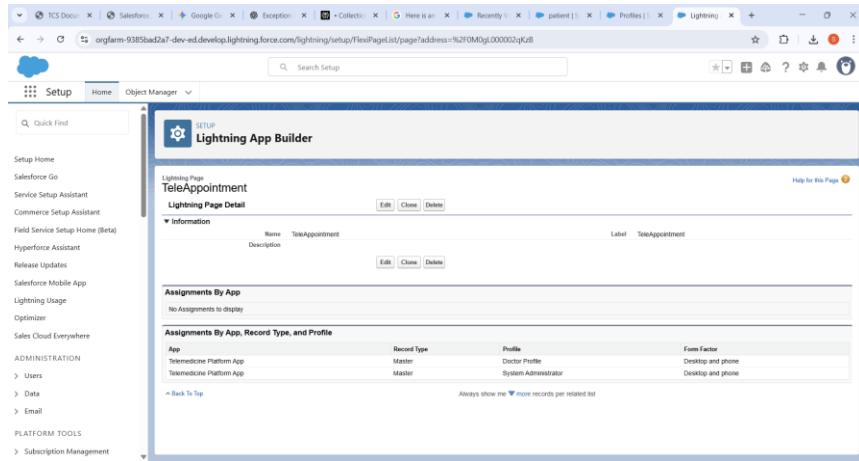
The screenshot shows the Salesforce Lightning App Builder interface. The main window displays a 'Lightning Page Detail' page for a 'Patient'. The page has sections for 'Information' (with fields for Name, Patient, Description, Label, and Patient), 'Assignments By App' (listing 'Telemedicine Platform App' with Record Type 'Master' and Profile 'Doctor Profile'), and 'Assignments By App, Record Type, and Profile' (listing the same information). On the left, a sidebar shows the navigation menu under 'SETUP'.

Check at the doctor Profile:

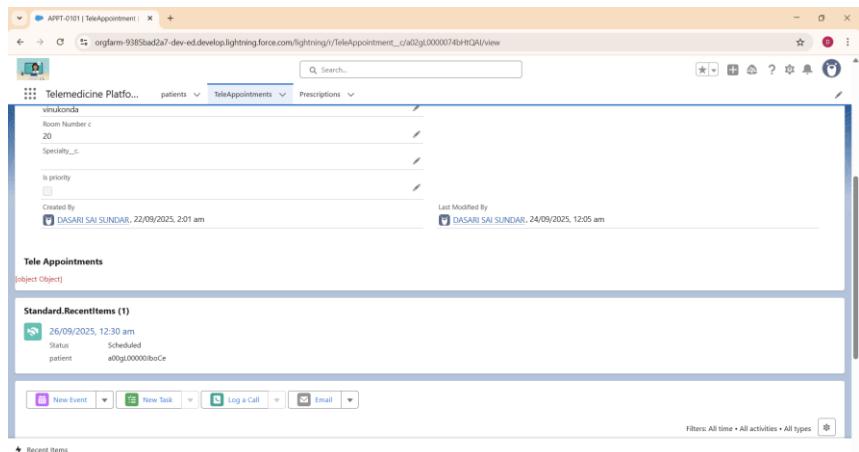


The screenshot shows a patient record page in the Salesforce Lightning UI. The page displays the following details for a patient:  
Patient Name: a00gl.00000boCe  
Patient ID: PAT-002  
Contact Number: 123  
DOB:  
Gender: Male  
Medical History:  
Patient email: dsau@gmail.com  
Patient Age:  
Created By: DASARI SAI SUNDAR, 17/09/2025, 10:39 pm  
Last Modified By: DASARI SAI SUNDAR, 17/09/2025, 10:39 pm

Create the TeleAppointment lightning record pages:



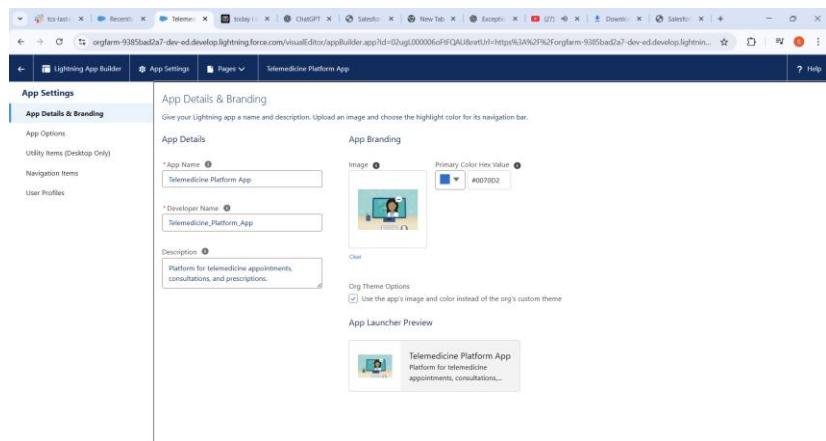
## Check at the doctor Profile:



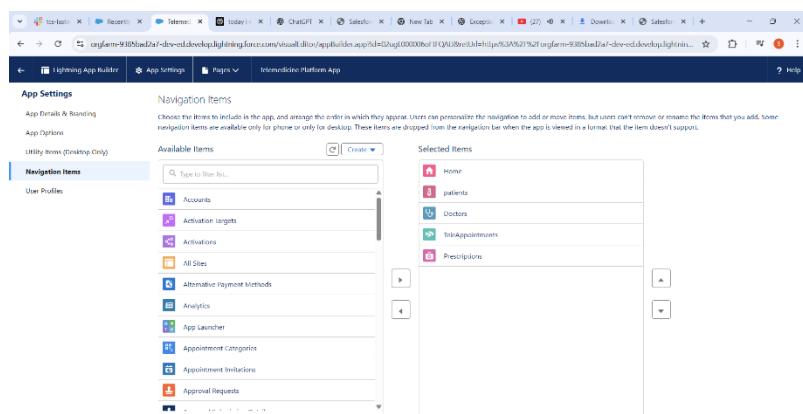
The Patient Lightning Record Page and TeleAppointment Lightning Record Page are customized to display and manage crucial patient and appointment details respectively within the Telemedicine Platform. These pages are assigned to the Doctor Profile to ensure doctors have seamless access to relevant patient 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:

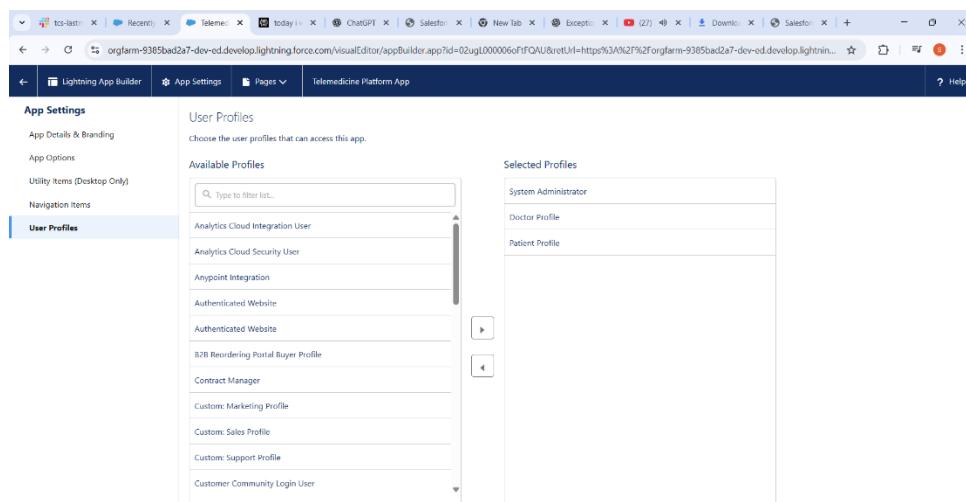
Created The Lightning App:



The Navigation Items Selected:

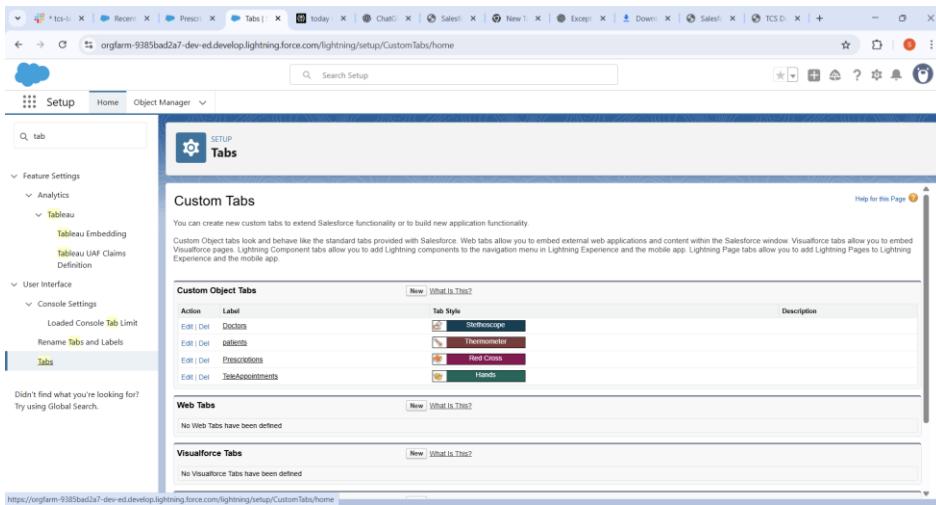


User Profiles Have Selected: As I Have 3 Main Users I have selected Three.



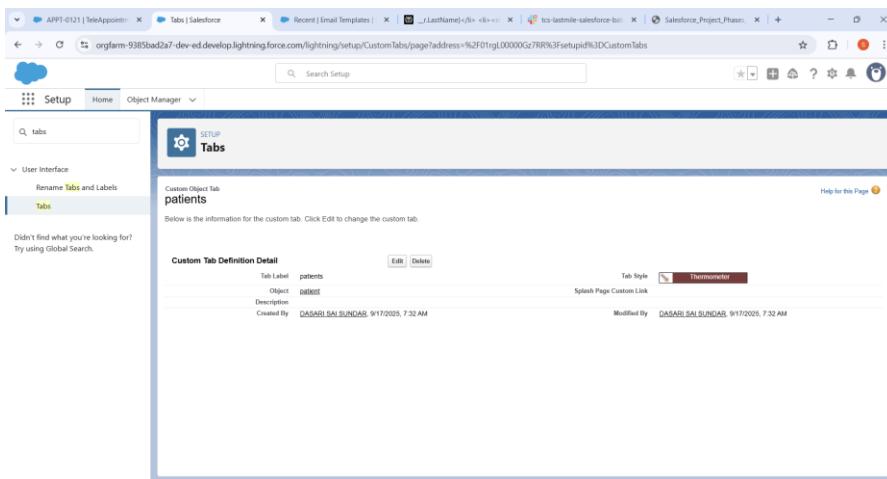
The Lightning App Framework for the Telemedicine Platform App is set up with custom branding, a description, and a highlight color for easy recognition. Key navigation items—Home, Patients, Doctors, TeleAppointments, and Prescriptions—are added to streamline access to core features. Specific profiles 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:

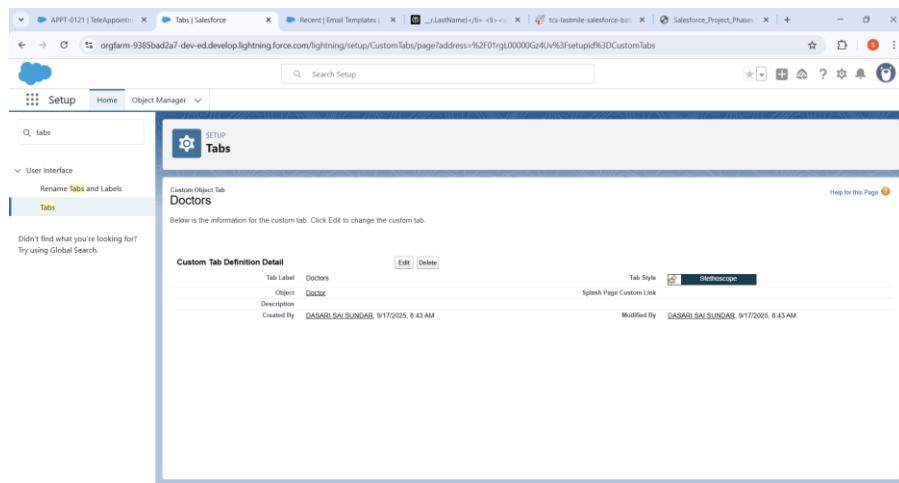


These tabs help users quickly access relevant records and features in the telemedicine platform with distinct icons for better visual identification.

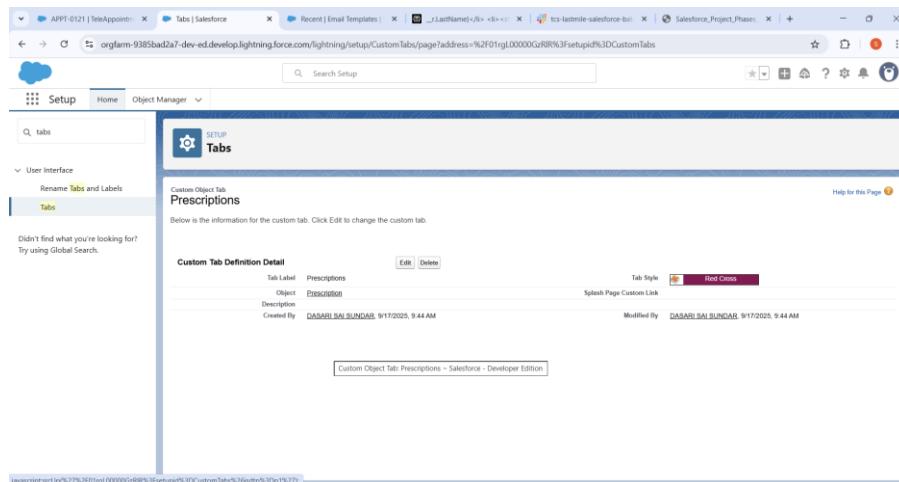
Patient Tab Created:



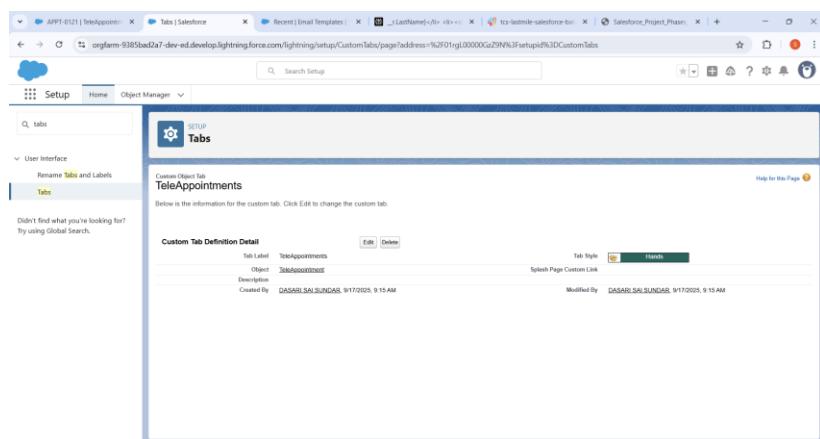
## Doctor Tab Created:



## Prescription Tab created:



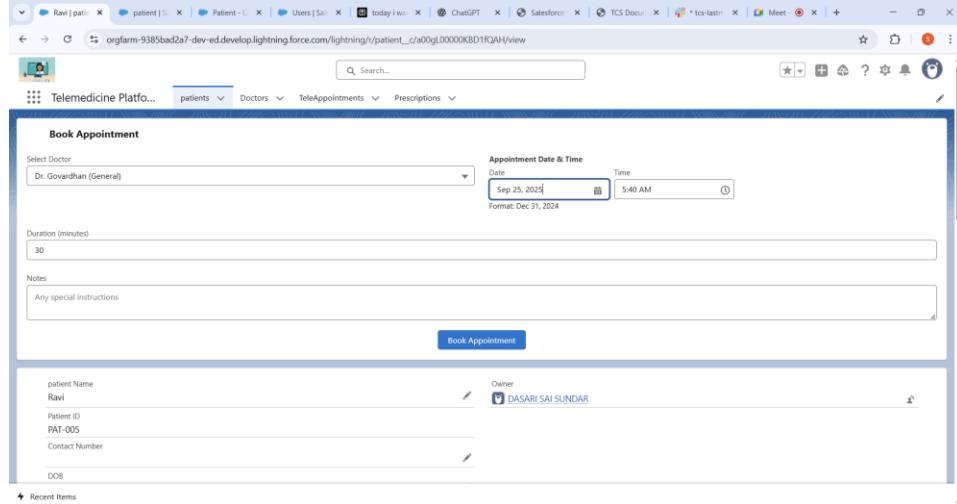
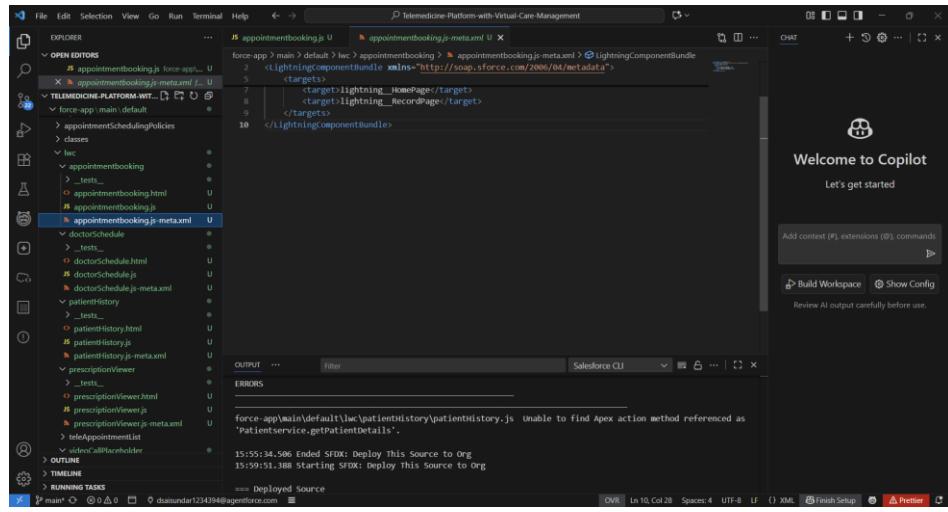
## TeleAppointment Tab Created:



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.

```

<force-app> > main > default > lwc > doctorSchedule > doctorSchedule.html ...
1  <template>
2    <lightning-card title="Today's Schedule" icon-name="standard:calendar">
3      <div class="slds-m-around_medium">
4        <template if:true={appointments}>
5          <div key={appointment.id} class="slds-box slds-m-bottom_small">
6            <strong>Time:</strong> {appointment.appointment_time_c}</p>
7            <strong>Patient:</strong> {appointment.patientName}</p>
8            <strong>Reason:</strong> {appointment.reason}</p>
9            <strong>Doctor:</strong> {appointment.doctorName}</p>
10           </div>
11         </template>
12         <template if:false={appointments}>
13           <p>No appointments for today.</p>
14         </template>
15       </div>
16     </lightning-card>
17   </template>
18 </lightning-card>
19 </template>
20 </force-app>

```

Output pane shows changes:

- Changed doctorschedule LightningComponentBundle force-app/main/default/lwc/doctorSchedule.html
- Changed doctorschedule LightningComponentBundle force-app/main/default/lwc/doctorSchedule/doctorschedule.js
- Changed doctorschedule LightningComponentBundle force-app/main/default/lwc/doctorSchedule/doctorschedule.js-meta.xml

Log pane shows:

- 18:30:11.765 Ended SFDX: Deploy This Source to Org
- 18:46:36.983 Starting SFDX: Deploy This Source to Org

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

Created the PatientHistory LWC:

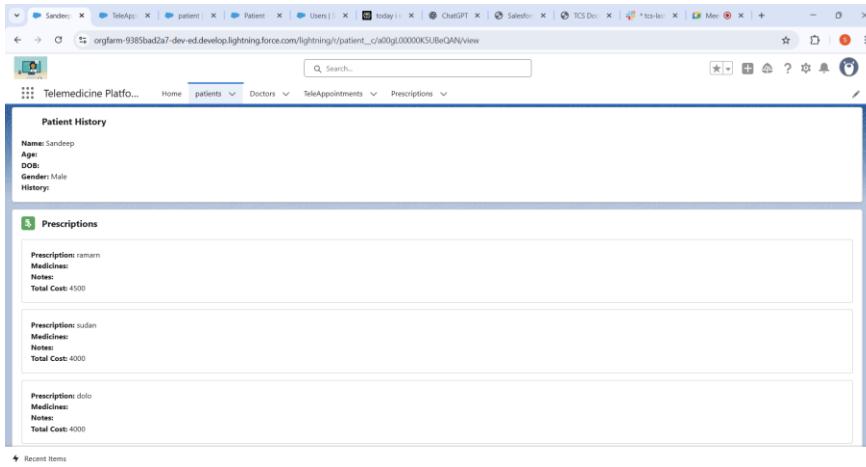
```

<force-app> > main > default > patientHistory > patientHistory.js ...
1  import { LightningElement, api, track, wire } from 'lwc';
2  import getPatientDetails from '@salesforce/apex/PatientsService/getPatientDetails';
3
4  export default class PatientHistory extends LightningElement {
5    @api recordId;
6    @track patientData;
7    @track error;
8
9    @wire(getPatientDetails, { patientId: '$recordId' })
10   wired({ data, error }) {
11     if (data) {
12       this.patientData = data;
13       this.error = undefined;
14     } else if (error) {
15       this.patientData = undefined;
16       this.error = error;
17       console.error('Error loading patient details:', error);
18     }
19   }
20 }

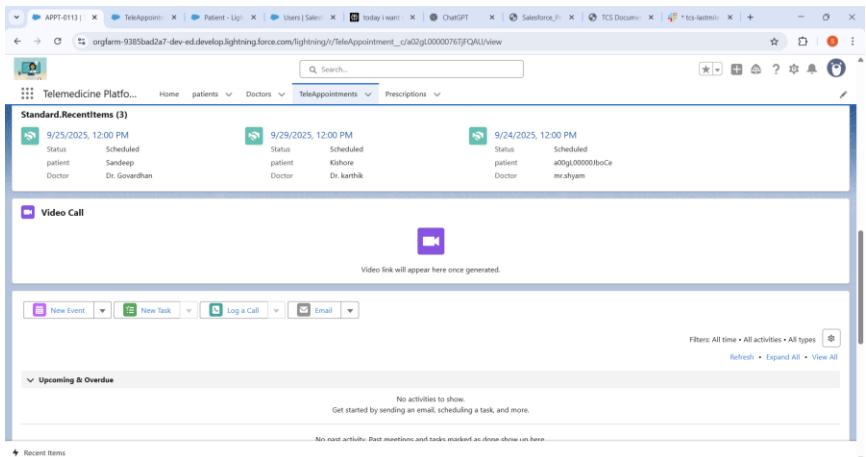
```

Output pane shows:

- Updated reading file 132:13:00
- Updated reading file 132:13:00
- Initial commit with Salesforce and updated...
- Add Phase 3: Data Modeling & Relations...
- Add Phase 3: Data Modeling & Relations...
- Fix Phase 2 README link so it points to...
- Updated README with Phase 2: Salesforce...
- Updated README with Phase 2: Salesforce...
- Added Phase 2: Salesforce Org Setup & Co...
- Fix file to Phase 1 README so it points to...
- Add Phase 1 README so it points to...
- First commit so it points to...



In the above image we can see the patient is displaying

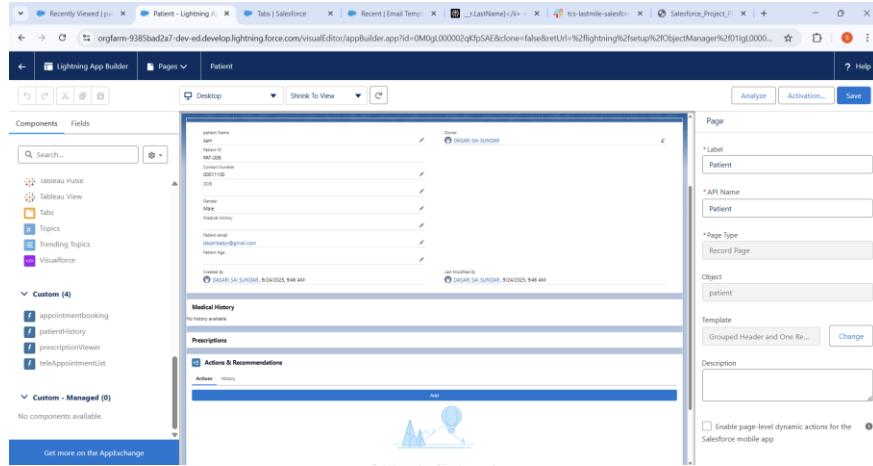


Google meet Video Call lwc is created as we go to phase the 7 we can do Background work of link generation

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

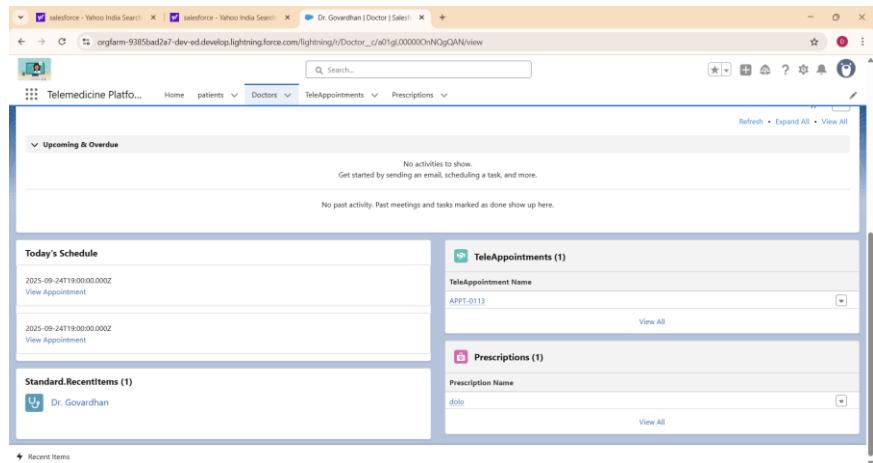
Lightning Web Components (LWCs) like appointment booking, doctor schedule, patient history, and prescription viewer enable streamlined telemedicine workflows by providing interactive and real-time data access for patients and doctors. Integrated features such as Google Meet video call support enhance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

## 5.lightning Record Page:

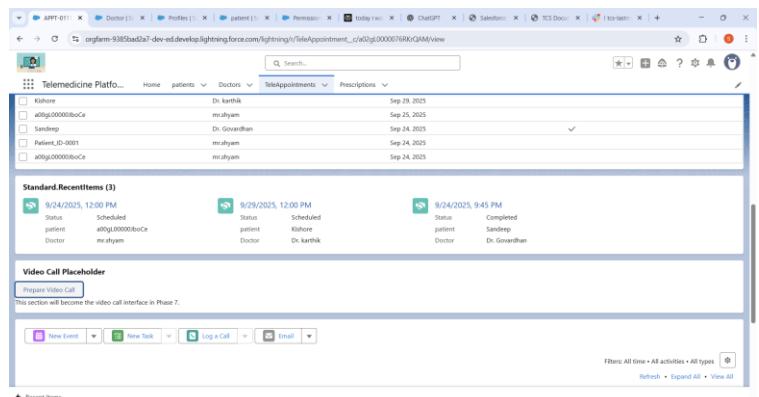


From the above we can see the Appopintbooking ,PatientHistory, Prescription Viewer and the Placevideocall custom lightning apps.We just drag them into the lightning page.

## 6.Navigation Buttons:



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



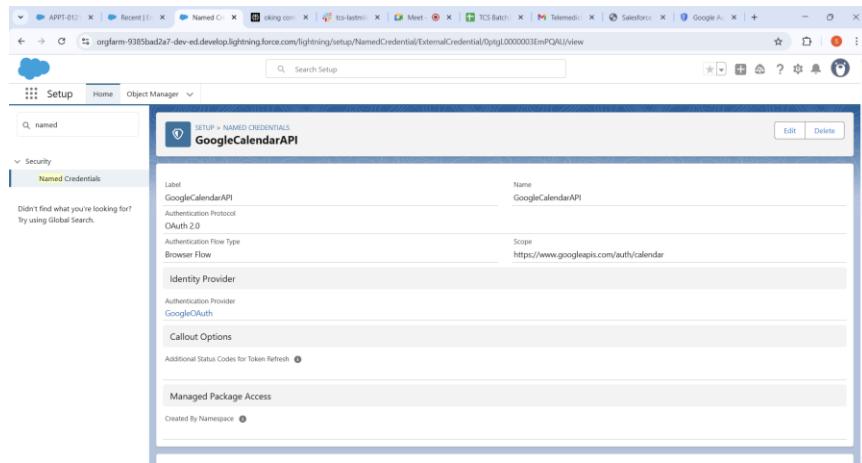
By clicking the Prepare video we can directly goes to the google meet with the patient

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.

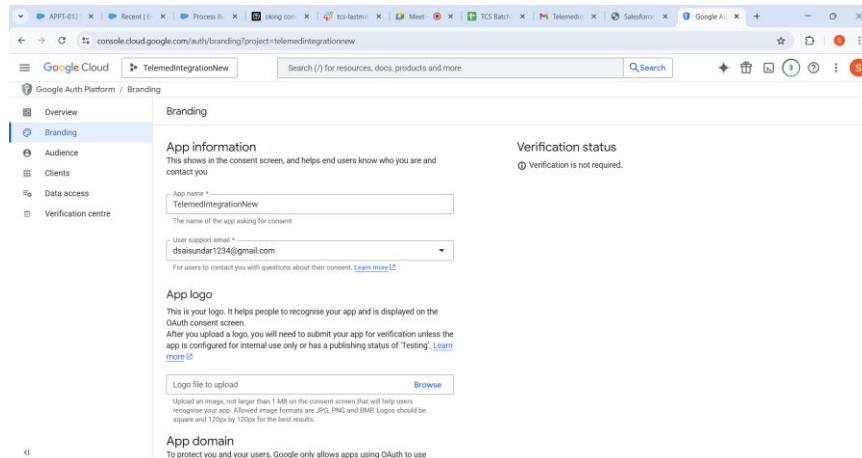
## **Phase 7: Integration & External Access**

# Phase7- Integration Access Google meet Link formation and process flows

## 1.Named Credentials:

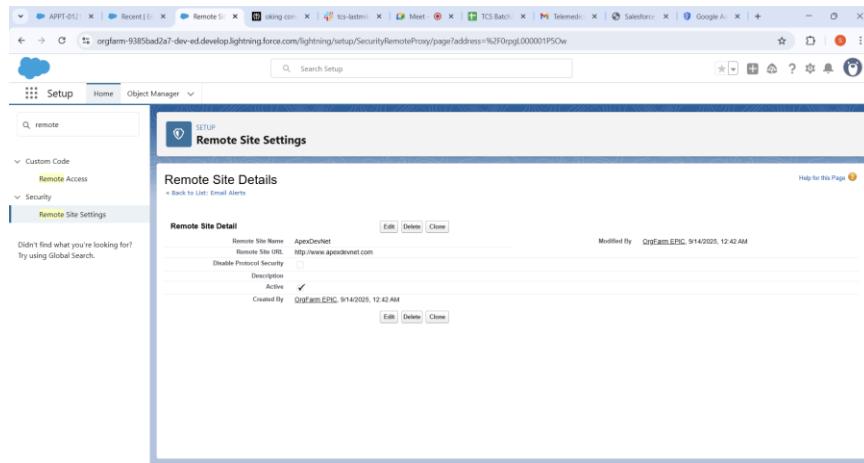


Created the GoogleAPI in the Named Credential for the access for the GoogleCloud



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.

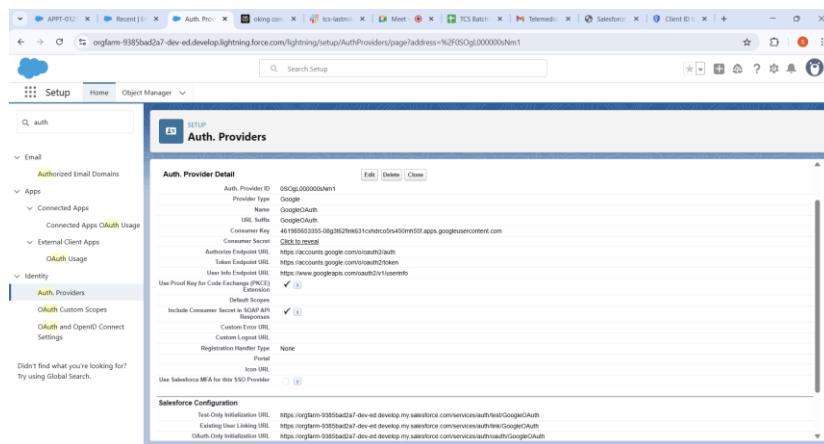
## 2. Remote Site Settings:



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) },
        }
        // Add other event fields as required
    }
}
```

```

'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');
}

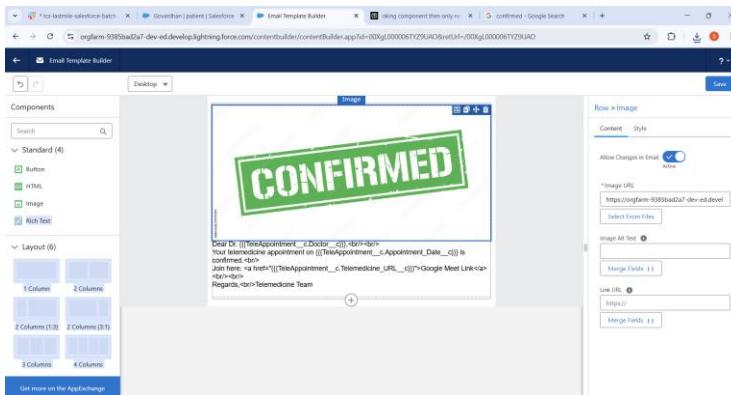
private static String getDoctorEmail(TeleAppointment__c appt) {
    if(appt.Doctor__c == null) return null;
    User doctorUser = [SELECT Email FROM User WHERE Id = :appt.Doctor__c LIMIT 1];
    return doctorUser != null ? doctorUser.Email : null;
}

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;
}
}

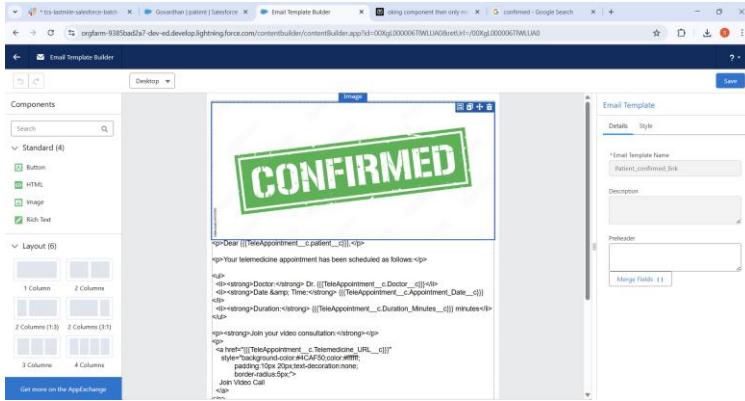
```

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:



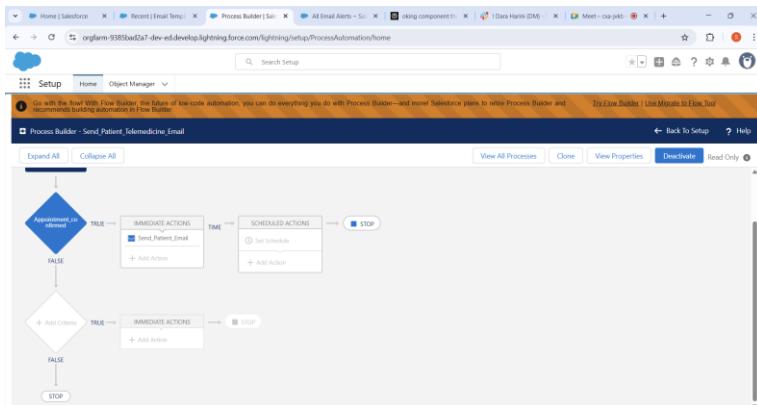
This is For the doctor for appointment is confirmed successfully and link is send successfully



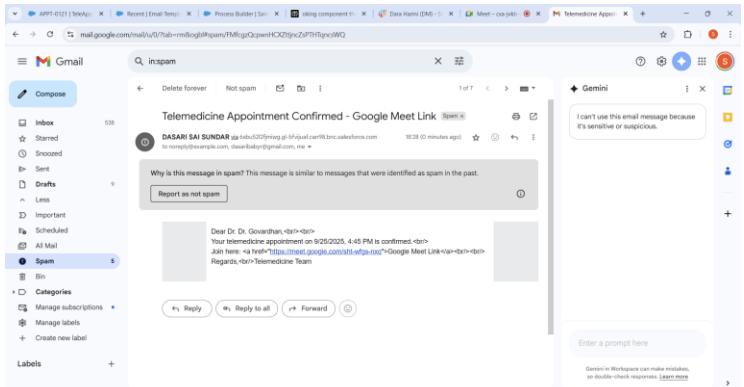
This is For the Patient for appointment is confirmed successfully and Google meet link is send successfully

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.

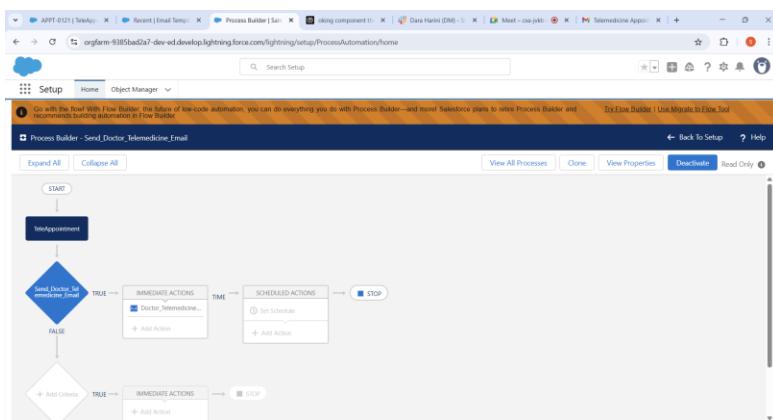
## 6.Process Builder Screenshot



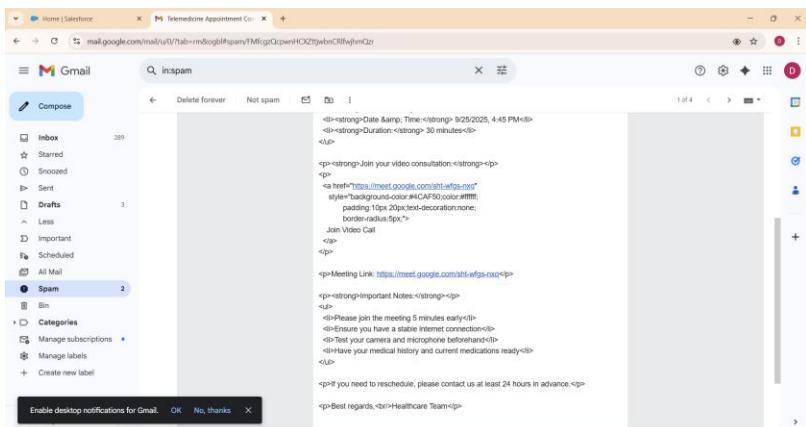
Process Builder is created for the patient link.



Received the mail at the patient email box

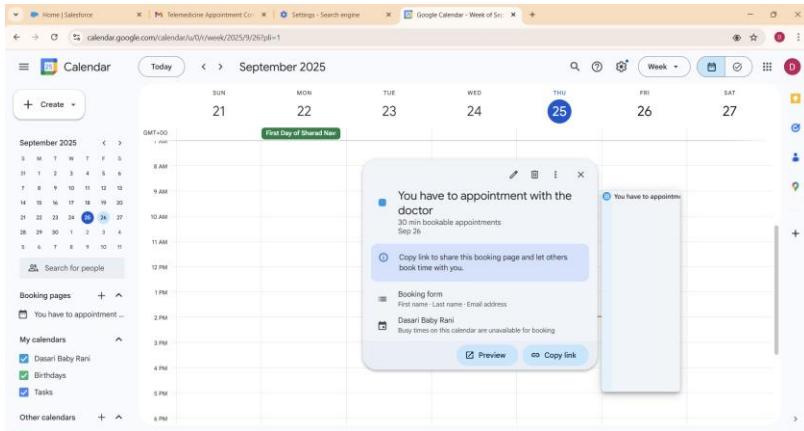


Process Builder is created for the Doctor link.



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.



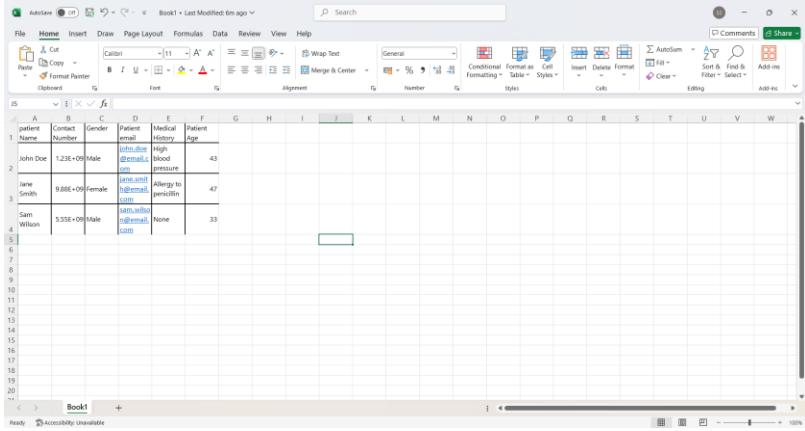
Google Calendar integration is successfully implemented using Named Credentials, OAuth authentication, and Apex REST callouts. Remote Site Settings ensure secure API access. Automated email notifications are triggered via Process Builder using Lightning Email Templates to notify users with Google Meet 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:

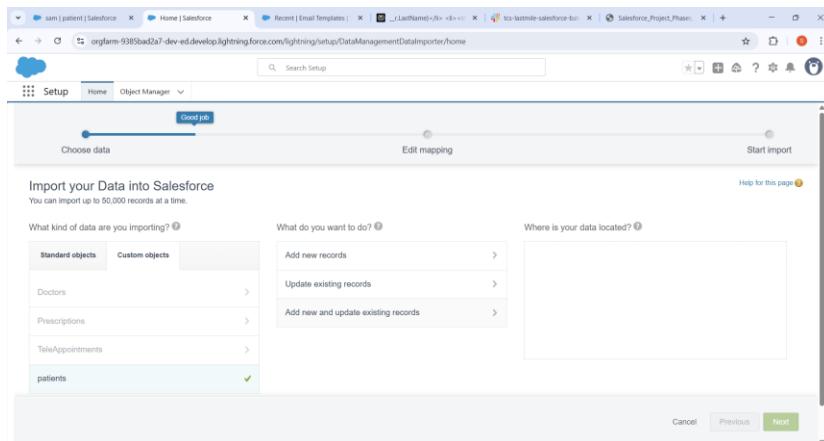
### 1. Data Import Wizard:

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



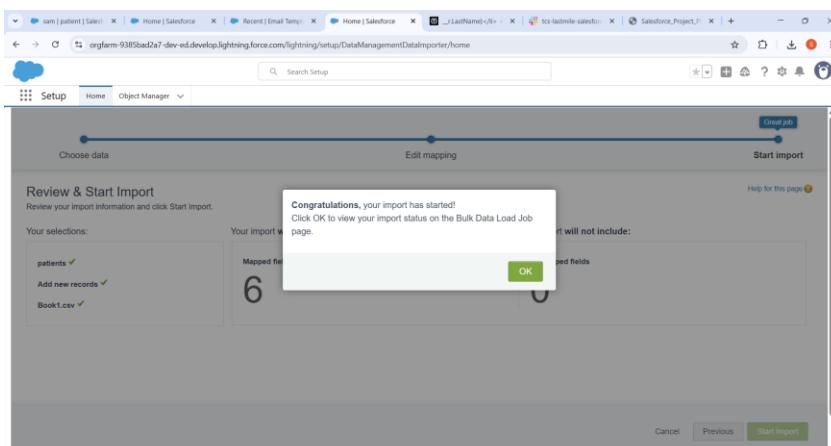
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Name	Contact Number	Gender	Patient email	Normal blood pressure	Patient age																	
2	John Doe	1.23E+09	Male	john.doe@gmail.com	High blood pressure	43																	
3	Jane Smith	8.88E+09	Female	jane.smith@gmail.com	Allergy to penicillin	47																	
4	Sam Wilson	5.55E+09	Male	sam.wilson@gmail.com	None	33																	

- b) Create and Updating the setting in the Import Data
- With select the Patient as the source object → Locate our .csv



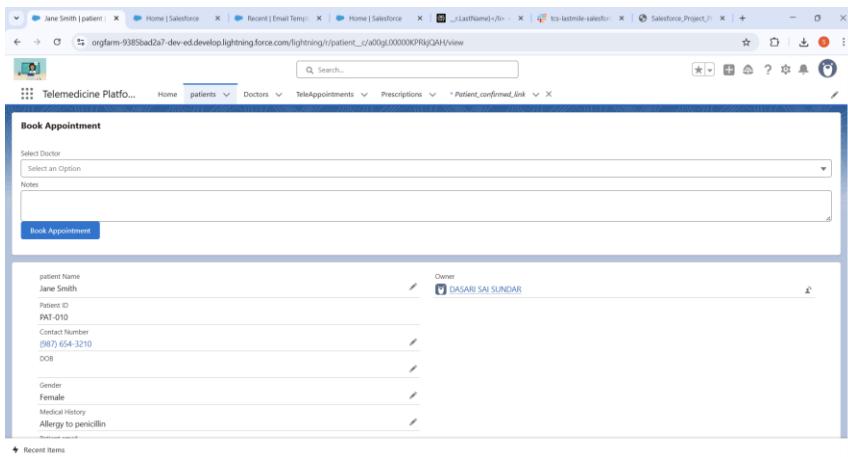
The screenshot shows the 'Choose data' step of the Data Import Wizard. It displays a progress bar with three steps: 'Good job', 'Choose data', 'Edit mapping', and 'Start import'. Below the progress bar, there's a section titled 'Import your Data into Salesforce' with the note 'You can import up to 50,000 records at a time.' It includes three dropdown menus: 'What kind of data are you importing?' (Standard objects, Custom objects), 'What do you want to do?' (Add new records, Update existing records, Add new and update existing records), and 'Where is your data located?' (with a text input field). At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

- c) Successfully saved the Import function



The screenshot shows the 'Review & Start Import' step of the Data Import Wizard. It displays a progress bar with three steps: 'Good job', 'Choose data', 'Edit mapping', and 'Start import'. A modal window is open in the center, displaying the message 'Congratulations, your import has started! Click OK to view your import status on the Bulk Data Load Job page.' Below the modal, there's a list of 'Your selections:' including 'patients' (selected), 'Add new records' (selected), and 'Book1.csv' (selected). At the bottom right are 'Cancel', 'Previous', and 'Start import' buttons.

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

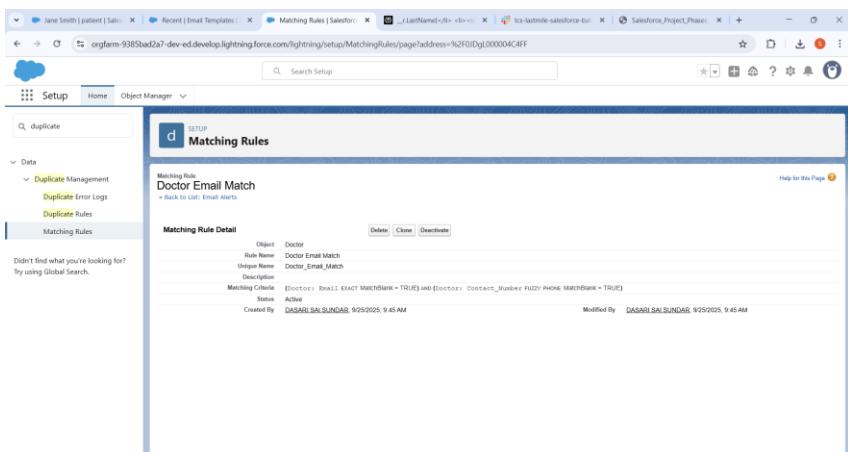


- Used Salesforce Data Import Wizard to upload patient information like names, contact numbers, date of birth, gender, email, medical history, and patient age from a well-prepared CSV file.
- Mapped CSV columns to corresponding Salesforce Patient fields to ensure correct data placement.
- Imported data 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.

The screenshot shows the 'Duplicate Rules' setup page. A specific rule named 'Prevent Doctor Duplication' is selected. The rule details are as follows:

- Rule Name:** Prevent Doctor Duplication
- Object:** Doctor
- Action On Create:** Allow
- Action On Edit:** Allow
- Alert Test:** Active
- Matching Rule:** Doctor\_Email\_Match (Mapped)
- Matching Criteria:** (Doctor\_\_Email\_\_c!=null AND Doctor\_\_Email\_\_c!= '') AND ((Contact\_\_Name\_\_c=FuzzyPhoneMatch||Contact\_\_Phone\_\_c=FuzzyPhoneMatch))

The rule is set to 'Active' and has an 'Alert' and 'Report' checkbox checked under 'Operations On Create'.

## c) Test case1:

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

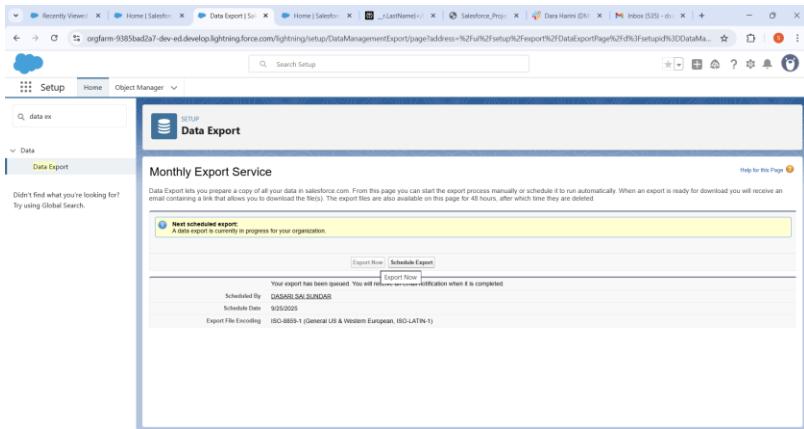
The screenshot shows the 'New Doctor' creation screen. A modal window titled 'Information' is open, displaying fields for Doctor Name (Dr. Test1), Doctor ID, Speciality (General), and Email (miraj@gmail.com). A warning message at the bottom of the modal states: 'Similar Records Exist' and 'This record looks like an existing record. Make sure to check any potential duplicate records before saving. View Duplicates'. The 'Save & New' button is visible at the bottom of the modal.

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

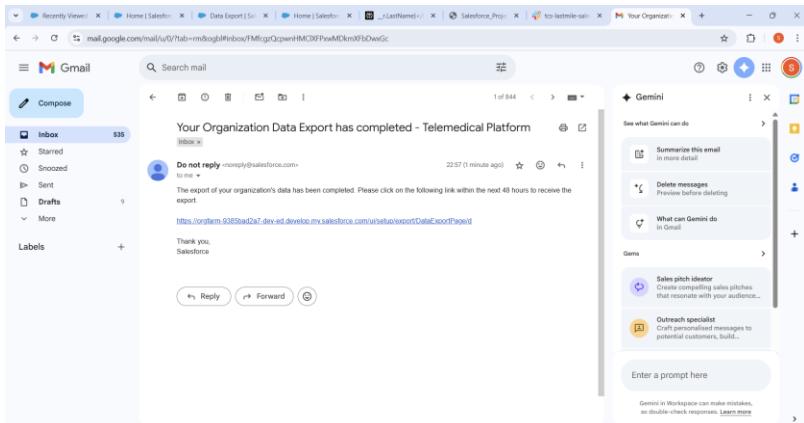
- Matching Rules can be defined to identify duplicates based on patient identifiers such as email, phone number, patient ID, as well as for doctors using license numbers or email addresses. This helps flag potential duplicate records when users enter or import data.
- Duplicate Rules enforce policies 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

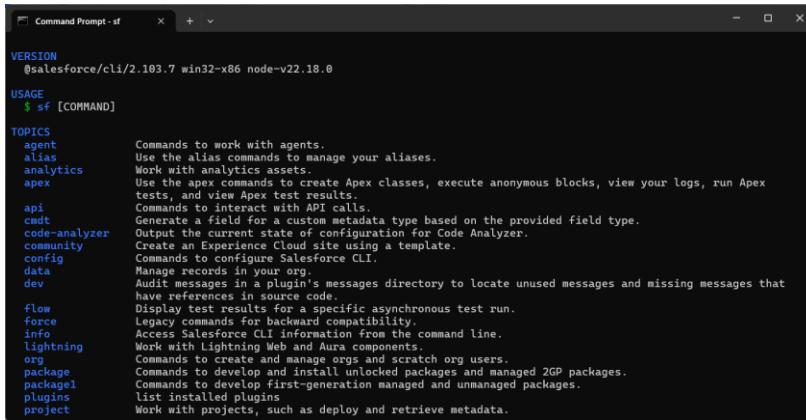
A screenshot of Microsoft Excel displaying a CSV file named 'Doctor\_c.csv'. The data consists of five rows of information, each representing a record. The columns include 'Id', 'Name', 'Lastname', 'Email', 'Phone', 'Address', 'City', 'State', 'Zip', 'Country', 'Industry', 'Category', 'LastActivityDate', 'LastActivityTime', 'Email\_c', 'Availability', 'Telemedic\_Contact\_Number\_c', and 'DOC-0001 Cardiology', 'DOC-0002 Neurology', 'DOC-0004 Pediatrics', and 'DOC-0003 General'. Row 5 shows a value '1234' in the 'Telemedic\_Contact\_Number\_c' column.

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:

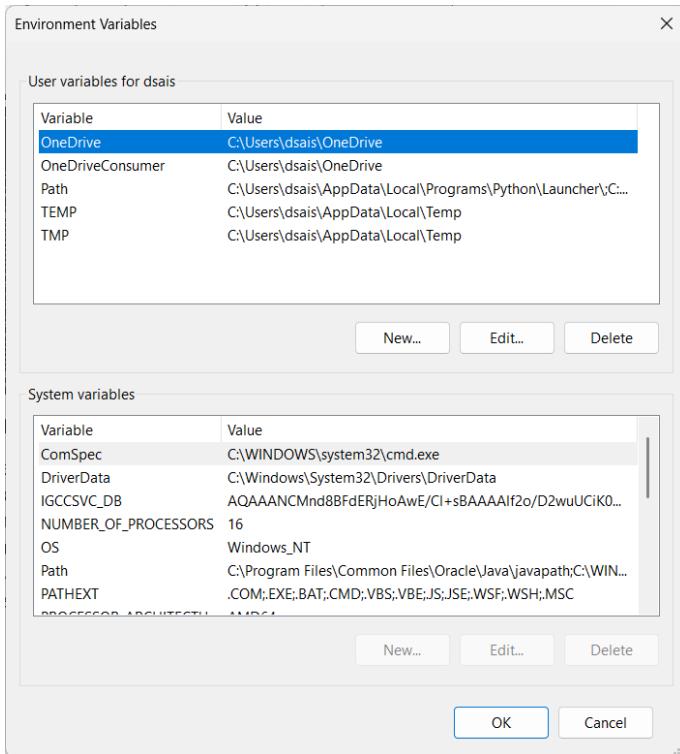
- This is the sf command to check salesforce CLI is installed or not.



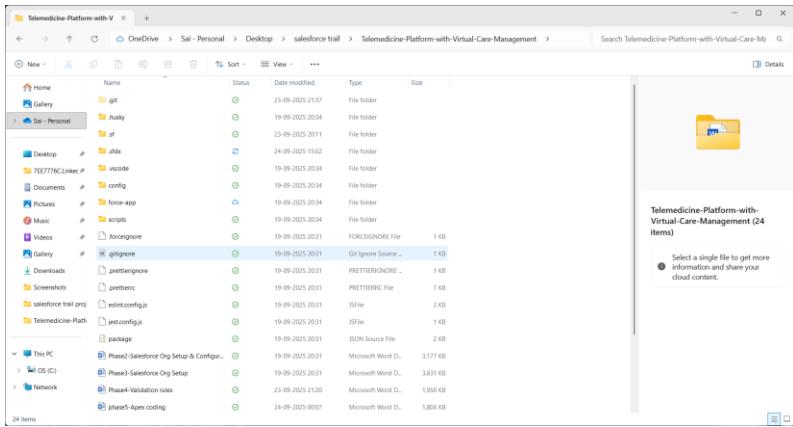
```
Command Prompt - sf
VERSION
@salesforce/cli/2.103.7 win32-x86 node-v22.18.0
USAGE
$ sf [COMMAND]

TOPICS
agent      Commands to work with agents.
alias      Use the alias commands to manage your aliases.
analytics   Work with analytics assets.
apex       Use the apex commands to create Apex classes, execute anonymous blocks, view your logs, run Apex tests, and view Apex test results.
api        Commands to interact with API calls.
cadt      Generate a field for a custom metadata type based on the provided field type.
code-analyzer   Output the current state of configuration for Code Analyzer.
community   Create an Experience Cloud site using a template.
config      Commands to configure Salesforce CLI.
data       Manage files and objects in your org.
dev        Audit messages in a plugin's messages directory to locate unused messages and missing messages that have references in source code.
flow       Display test results for a specific asynchronous test run.
force      Legacy commands for backward compatibility.
info       Access Salesforce CLI information from the command line.
lightning   Work with Lightning Web and Aura components.
org        Commands to create and manage orgs and scratch org users.
package    Commands to develop and install unlocked packages and managed 2GP packages.
package1   Commands to develop first-generation managed and unmanaged packages.
plugins    List installed plugins.
project   Work with projects, such as deploy and retrieve metadata.
```

- Check correct environment is set or not



### c) SFDX Folder



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

```

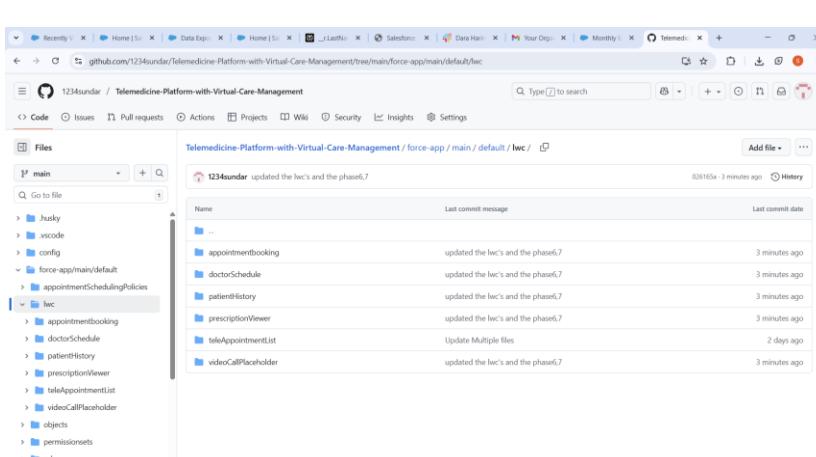
//git init
//git add .
//git commit -m "updated the github repo"
// git Push

```

```

//git init
//git add .
//git commit -m "updated the github repo"
// git Push

```



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

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

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

The screenshot shows the Report Builder interface with a report titled "New patients Report". The report has a single group row for "patient" and a single column for "Gender". The data table contains 9 rows of patient information, each with a unique ID, name, contact number, age, and gender.

patient	Contact Number	Patient Age	Gender
a0dg.00000j0sCe	123	-	Male
Patent_ID-0001	7999	-	Male
Kishore	-	-	Male
Sandeep	2341234	-	Male
Raj	-	-	Male
sem	00011100	-	Male
Govardhan	78999	-	Male
Jagan	99999999	-	Male
	0		

- Create the new summary report In TeleAppointment: Teleappointment Performance Report
  - Add the TeleAppointment Name for the summary report → group rows add Doctor, patient, Status

The screenshot shows the Report Builder interface with a report titled "New TeleAppointments Report". The report uses "GROUP ROWS" to group by "Doctor", "patient", and "Status". The "Columns" section includes "TeleAppointment" and "TeleAppointment Name". The report displays a hierarchical breakdown of appointments by doctor, patient, and status, with subtotals for each category.

Doctor	patient	Status	TeleAppointment	TeleAppointment Name
Dr. Govardhan (3)	Govardhan (1)	Requested (1)	APPT-0122	
		Subtotal		
		Subtotal		
Patent_ID-0001 (1)		Scheduled (1)	APPT-0121	
		Subtotal		
	Sandeep (1)	Completed (1)	APPT-0113	
		Subtotal		
Dr. Kishore (1)	Kishore (1)	Scheduled (1)	APPT-0112	
		Subtotal		
		Subtotal		
		Row Counts	Detail Rows	Subtotal
				Grand Total

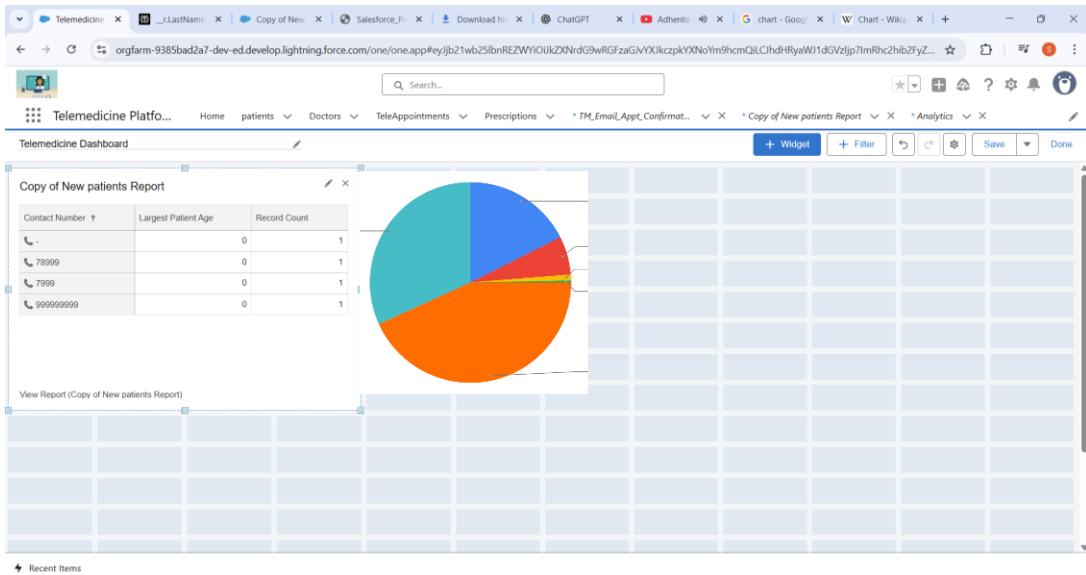
- Successfully run and the report

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

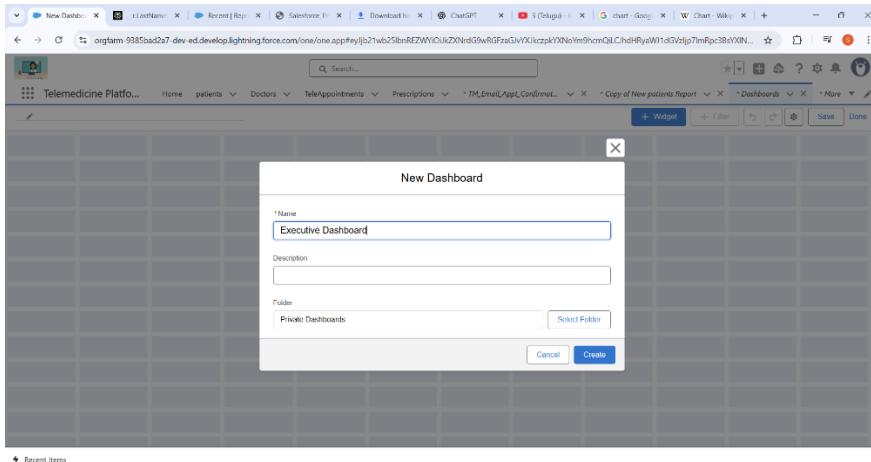
## 2. Telemedicine Dashboard: Dashboard of Patient



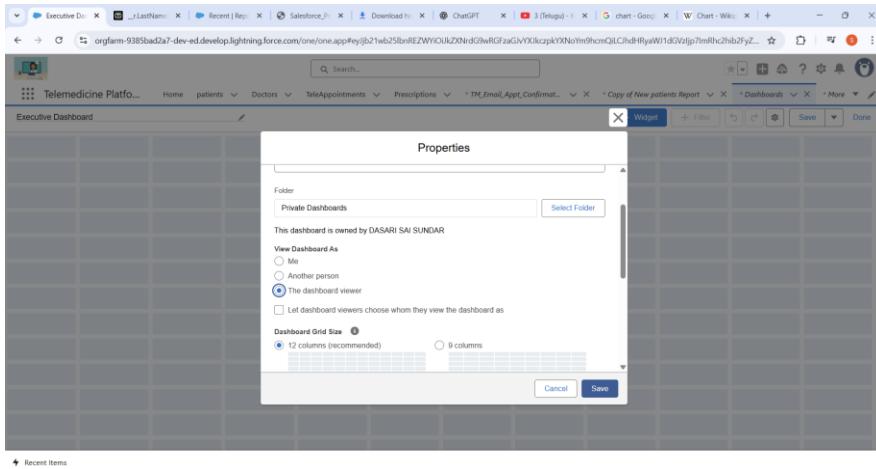
1. Click + Component, select your “Patient Demographics Report,” choose a chart type pie chart and add to the dashboard.

### 3. Dynamic Dashboard:

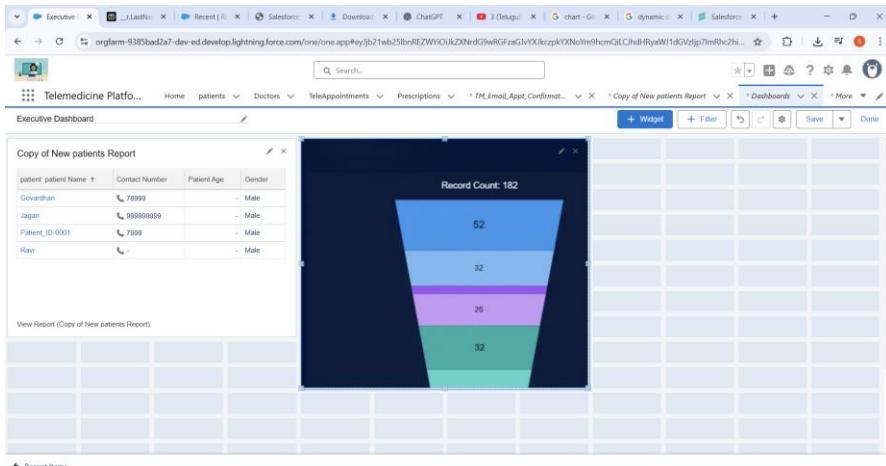
- Created the dashboard with NameExecutive Dashboard:



- Set the properties to the Dashboard Viewer:

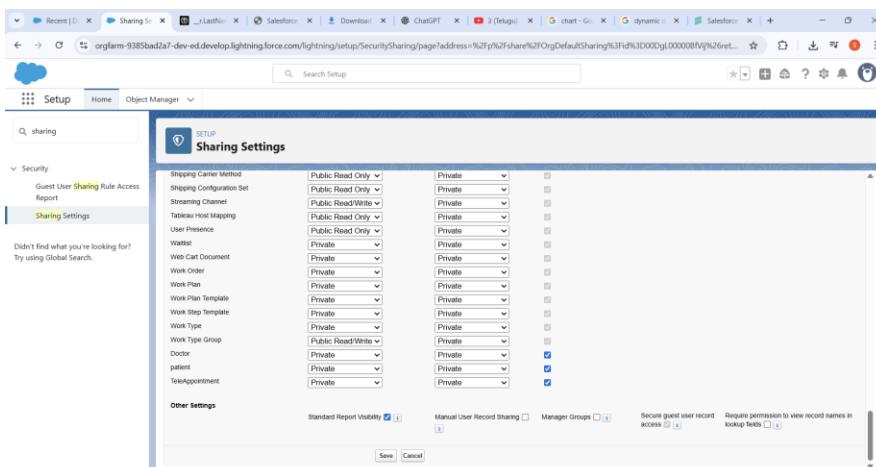


- Successfully add the Graph plot in the Dynamic Dashboard

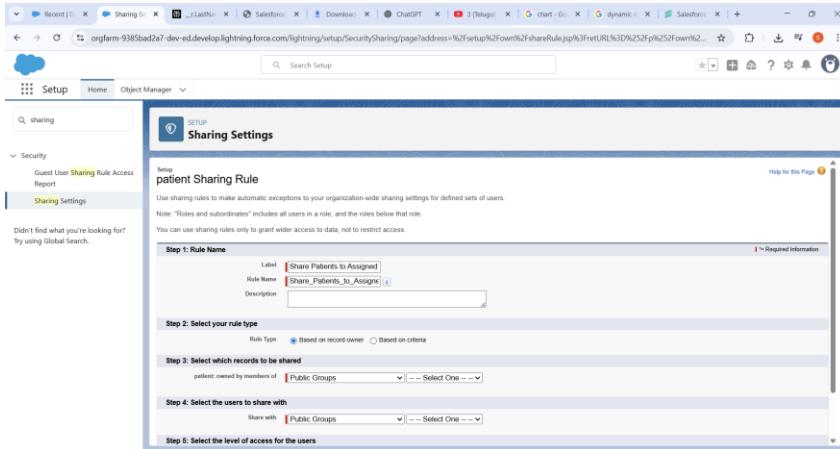


#### 4. Sharing settings:

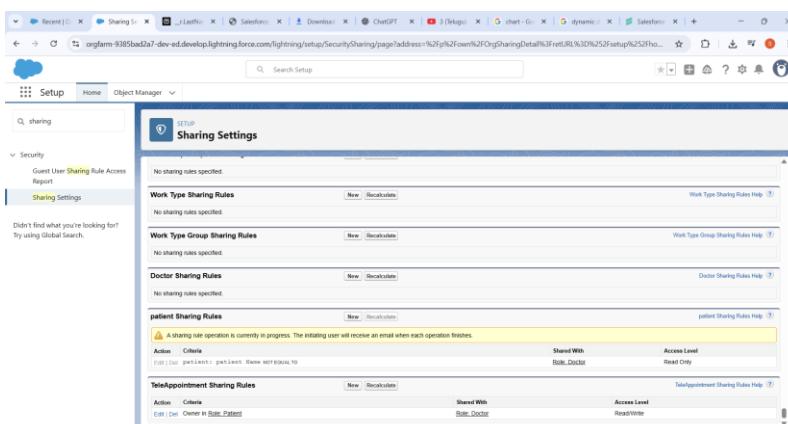
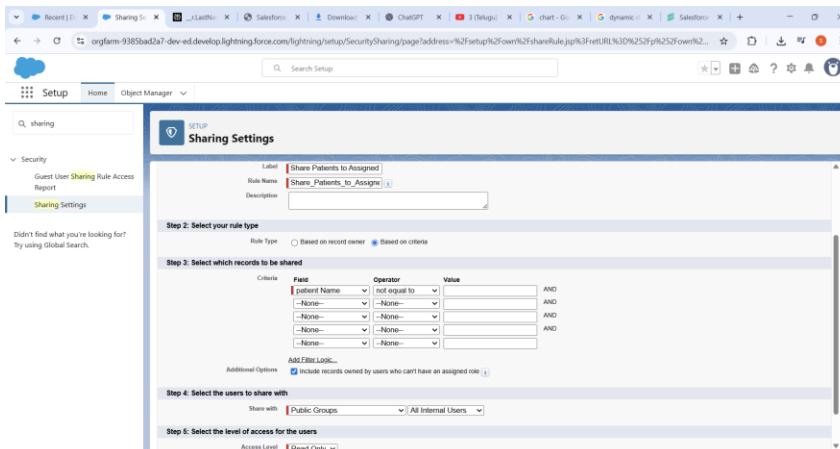
- For the patient and Teleappointment objects, set Default access to Private. This ensures only record owners and users above them in the role hierarchy can view or edit records by default.



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



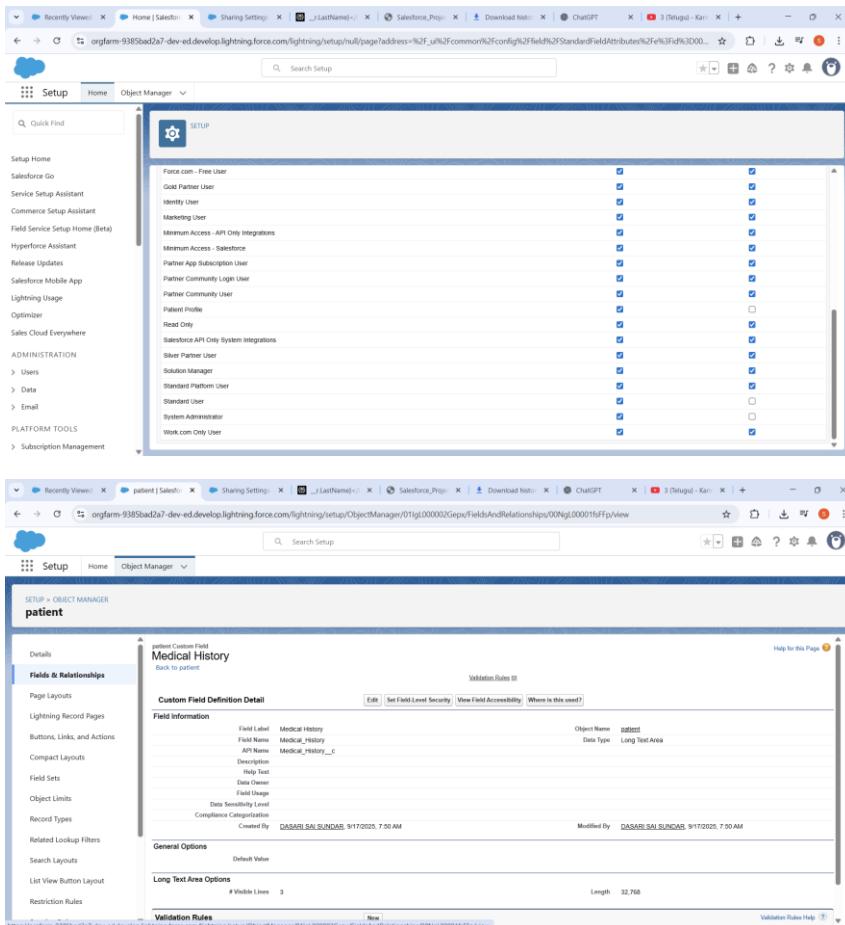
- Here the given criteria that the patient can be assigned to the only selected specialization. Patient → Assigned Doctor ≠ blank



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:

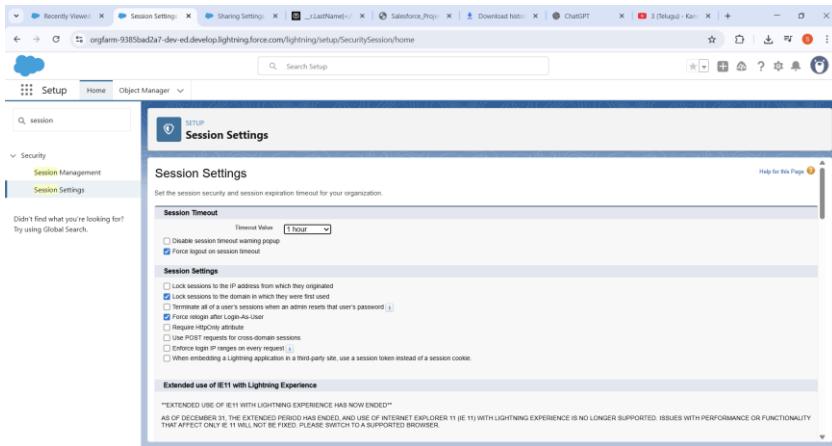
- For profiles like Standard User or Guest User, uncheck Visible to completely hide the field.
- For profiles like Doctor or System Administrator, leave Visible checked.
- If certain profiles should see but not edit the field, check Visible and Read-Only.



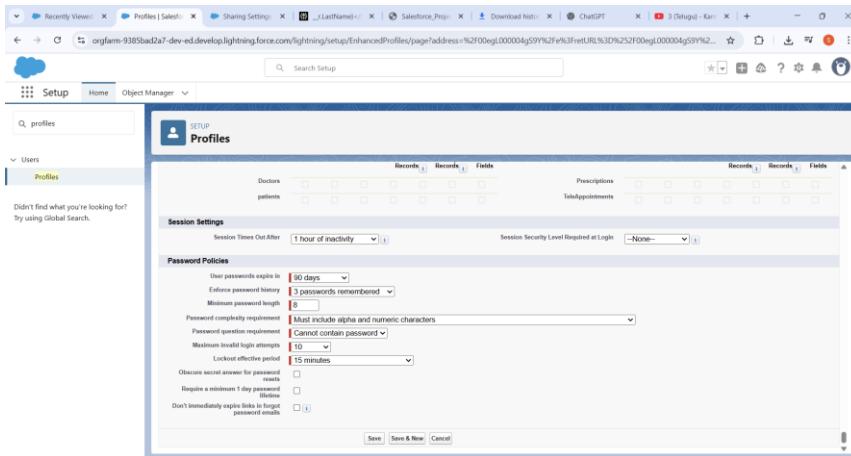
## 6. Session Settings and Login IP Ranges:

### 1. Session Settings:

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

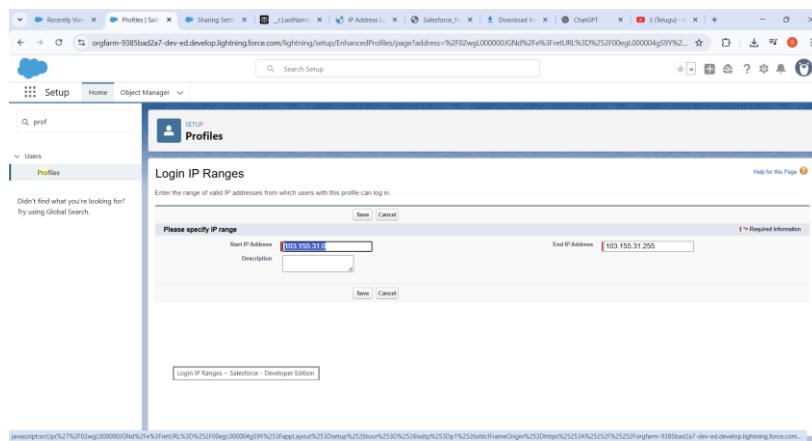


- In the profile settings session timings set to one hour

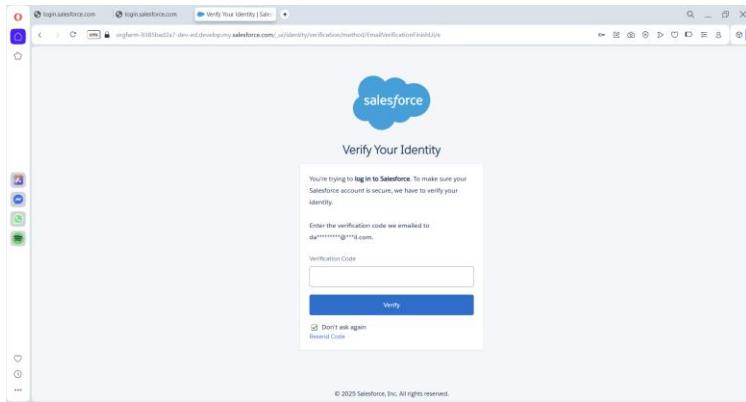


## 2. Login IP Ranges:

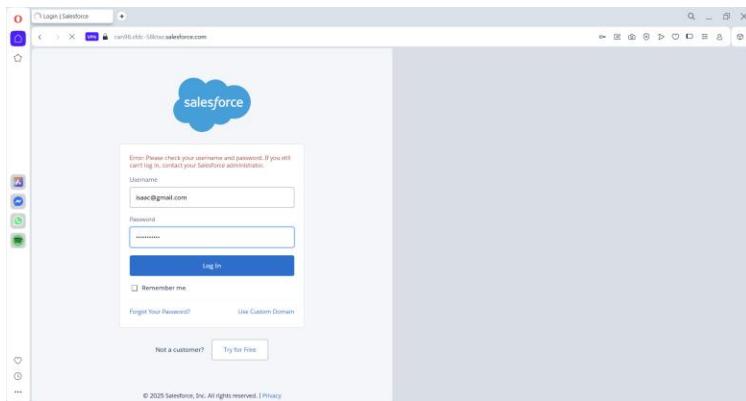
- Enter a Start IP Address and an End IP Address that define your organization's trusted network range covers all IP addresses from 103.155.31.0 up to 103.155.31.255.



- 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:

The screenshot shows the Salesforce Object Manager for the 'patient' object. In the 'Fields & Relationships' section, there is a table listing various fields. The columns include FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. Key fields listed include Contact Number, Created By, DOB, Gender, Last Modified By, and Medical History.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Contact Number	Contact_Number__c	Phone		
Created By	CreatedById	Lookup(User)		
DOB	DOB__c	Date		
Gender	Gender__c	Picklist		
Last Modified By	LastModifiedById	Lookup(User)		
Medical History	Medical_History__c	Long Text Area(32768)		

This is how it looks like

The screenshot shows the 'New patient' form. It has a header 'New patient' and a note '\* = Required Information'. The 'Information' section contains fields for patient Name (required), Patient ID, Contact Number, DOB, Gender (set to '--None--'), and Medical History. The form includes standard buttons at the bottom: Cancel, Save & New, and Save.

For the doctor object:

The screenshot shows the 'New Doctor' form. It has a header 'New Doctor' and a note '\* = Required Information'. The 'Information' section contains fields for Doctor Name (required), Doctor ID, Specialty (set to '--None--'), Email, Availability, and Telemedicine Profile. The form includes standard buttons at the bottom: Cancel, Save & New, and Save.

## For teleappointment object:

New TeleAppointment

\* = Required Information

Information	
TeleAppointment Name	Owner  DASARI SAI SUNDAR
Appointment ID	
* patient	<input type="text" value="Search patients..."/> 
Doctor	<input type="text" value="Search Doctors..."/> 
Appointment Date	
* Date	* Time
<input type="text"/> 	<input type="text"/> 
* Status	
<input type="text" value="Requested"/> 	
Telemedicine URL	
<input type="button" value="Cancel"/> <input type="button" value="Save &amp; New"/> <input type="button" value="Save"/>	

## For the Prescriptions:

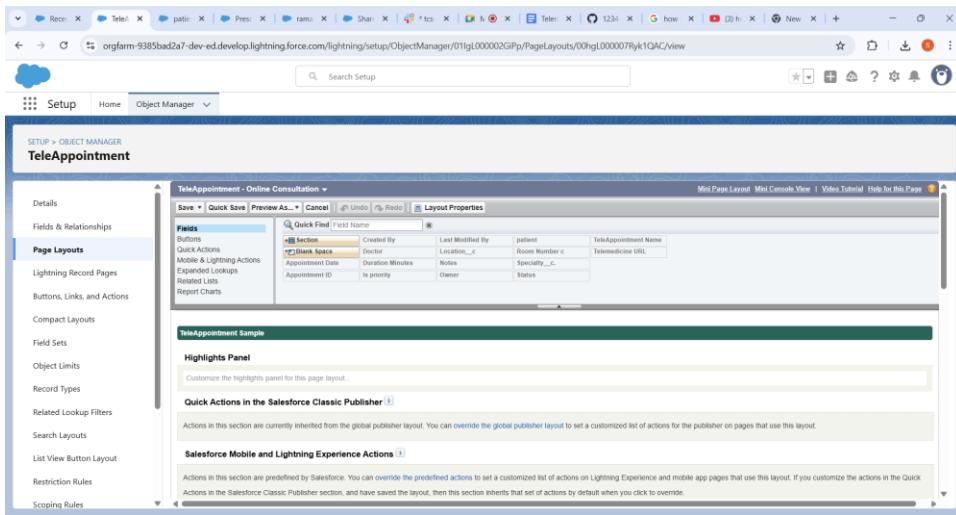
New Prescription

\* = Required Information

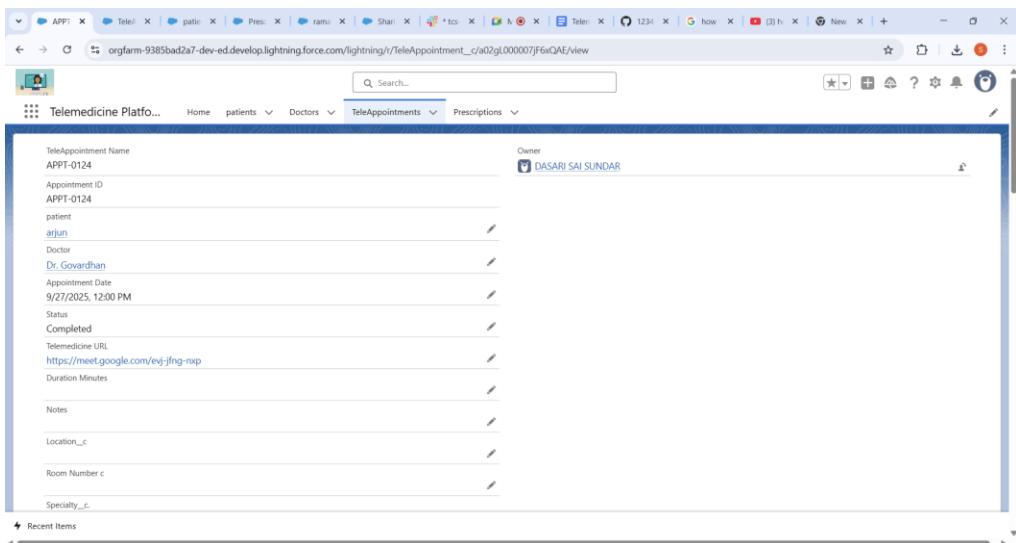
Information	
* Prescription Name	
Prescription ID	
* Appointment	
<input type="text" value="Search TeleAppointments..."/> 	
* patient	<input type="text" value="Search patients..."/> 
* Doctor	<input type="text" value="Search Doctors..."/> 
Medicines	
<input type="text"/> 	
<input type="button" value="Cancel"/> <input type="button" value="Save &amp; New"/> <input type="button" value="Save"/>	

Every field created is successfully added in the Platform objects.

## 8. Record Types: The is the custom Online Consultation

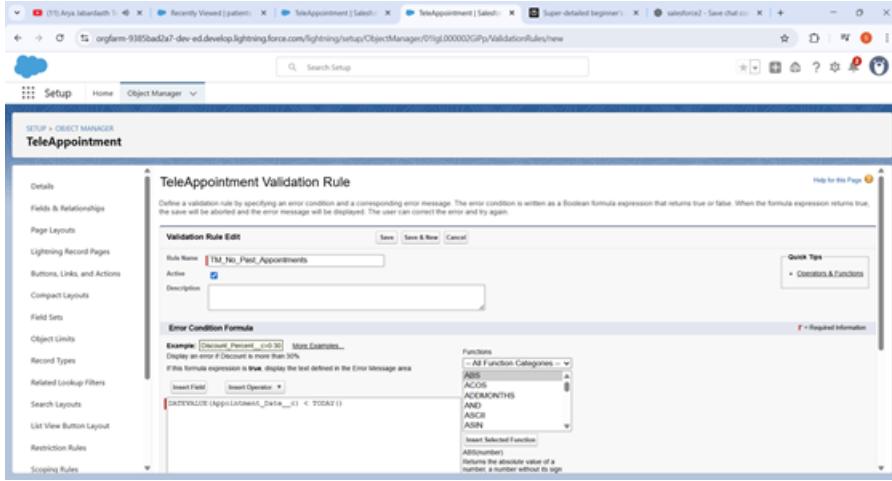


This is Appear on the screen

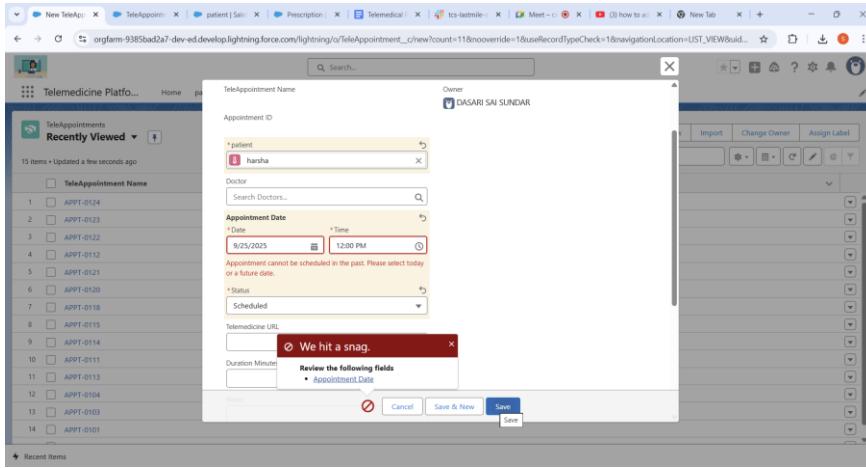


## 9. Process Automation:

**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

The screenshot shows a web browser window with a green success message at the top: "TeleAppointment 'APPT-0125' was created." Below the message, the appointment details are listed: Appointment Name (APPT-0125), Appointment ID (APPT-0125), patient (harsha), Doctor (Dr. karthik), Appointment Date (9/27/2025, 12:00 PM), Status (Scheduled), and Duration Minutes. The URL is [https://orgfarm-9385bad2a7-dev-ed.develop.lightning.force.com/lightning/r/TeleAppointment\\_c/a02g000000j5hQAx/view](https://orgfarm-9385bad2a7-dev-ed.develop.lightning.force.com/lightning/r/TeleAppointment_c/a02g000000j5hQAx/view).

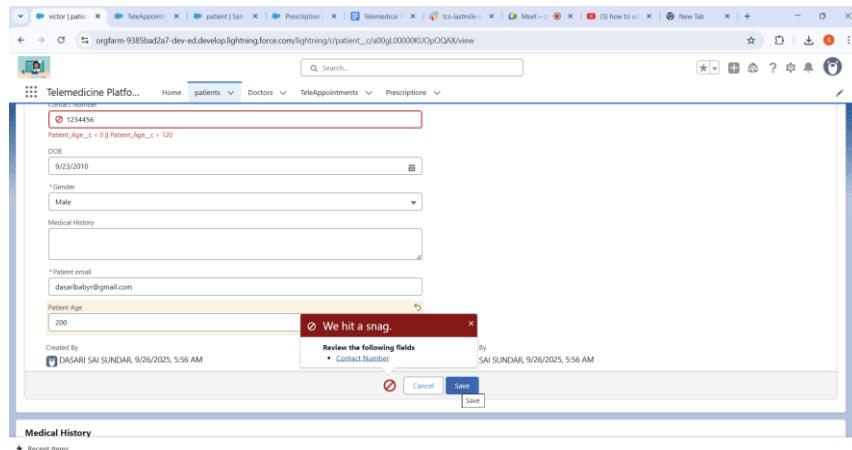
**b. Age Must be between 1-120:**

The screenshot shows the Salesforce Setup interface under the Object Manager for the 'patient' object. A validation rule named 'IM\_Valt\_Age\_Range' is being edited. The formula for the error condition is: `Patient_Age__c < 0 || Patient_Age__c > 120`. The formula builder interface is visible, showing functions like ABS, AND, ANDMONTHS, AND, ASCII, and ABS.

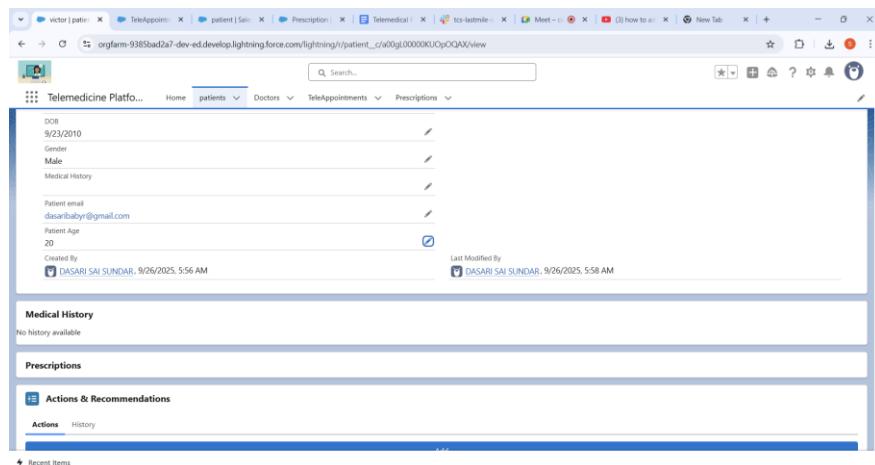
**Process Automation → Test Case1:** I give the patient age in 129:

The screenshot shows a web browser window for a patient record. The 'Patient\_Age\_\_c' field contains the value '129'. A red validation error message box appears, stating: "Patient\_Age\_\_c < 0 || Patient\_Age\_\_c > 120". Below the form, a modal dialog box says "We hit a snag." and lists the fields to review: Contact Number (DASARI SAI SUNDAR) and Date (9/26/2025, 5:56 AM). Buttons for "Cancel" and "Save" are shown at the bottom.

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

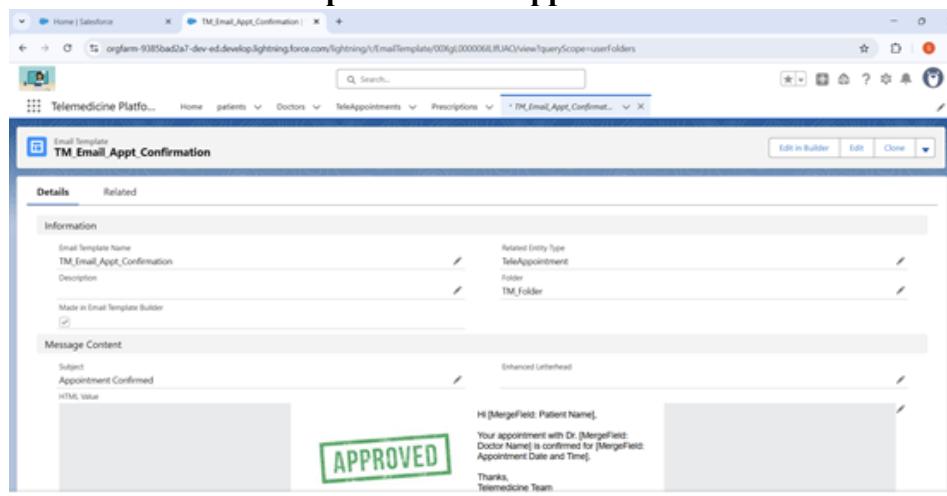


## Process Automation → Test Case3: Given 20 as the age and Testcase Passed:

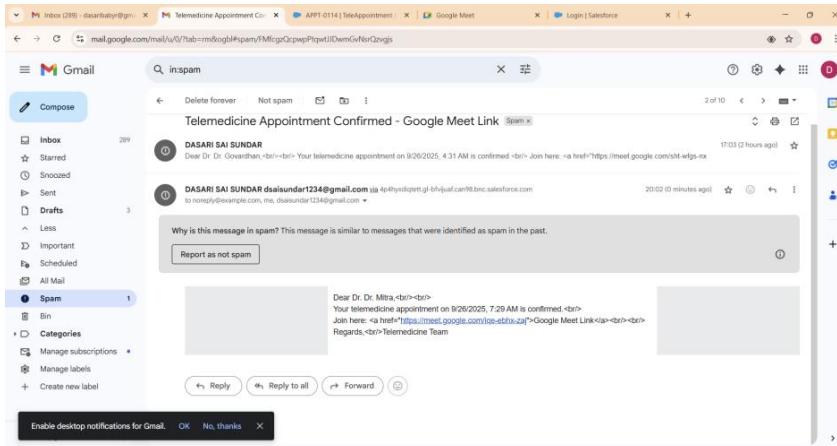


## 10. Lightning Email Template:

### Created the Email Template For the Appointment Confirmation:



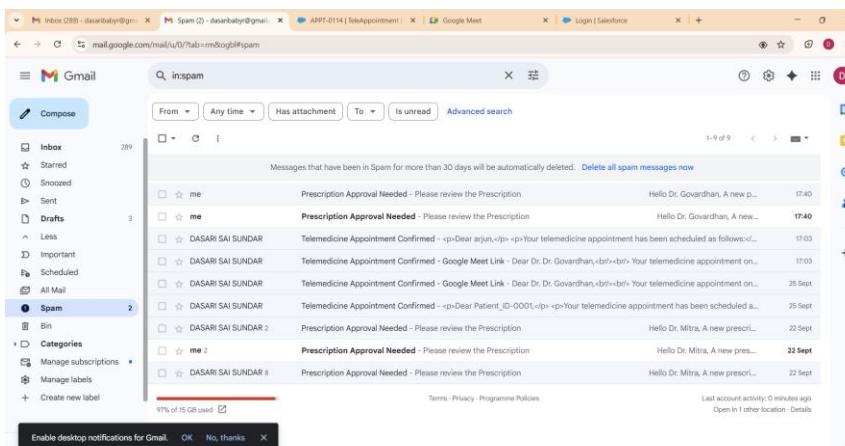
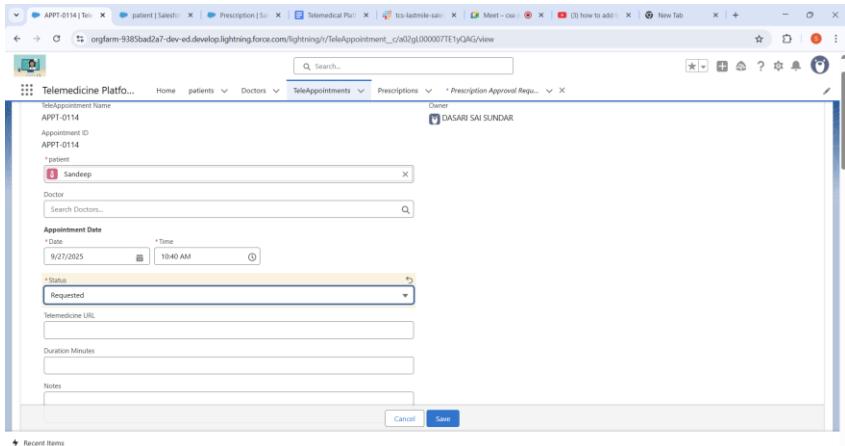
## Lightning Email Template → Test case1:



## 11. Workflow Rules Validation:

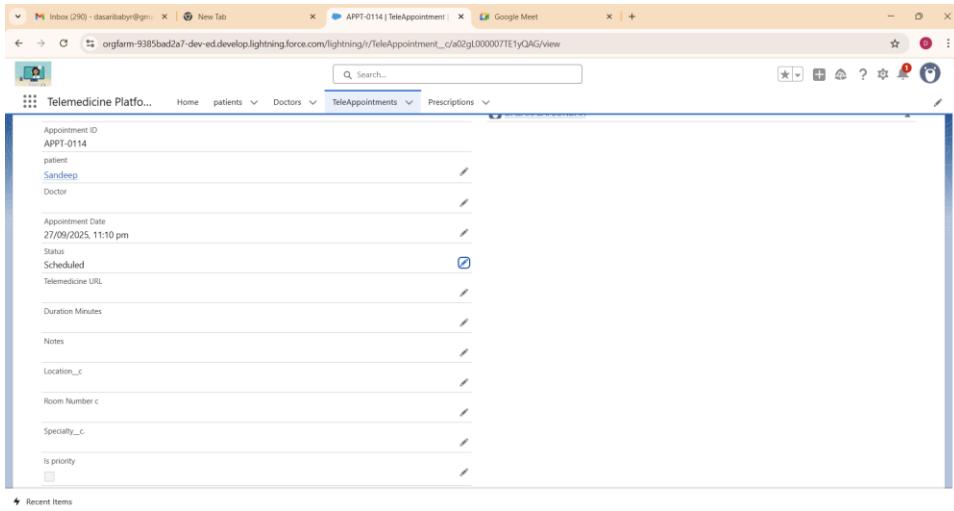
This work flow uses the Above Email Template. The doctor can make the approval in his org account then the email Approval confirmed.

## Workflow Rules Validation → Test case1:

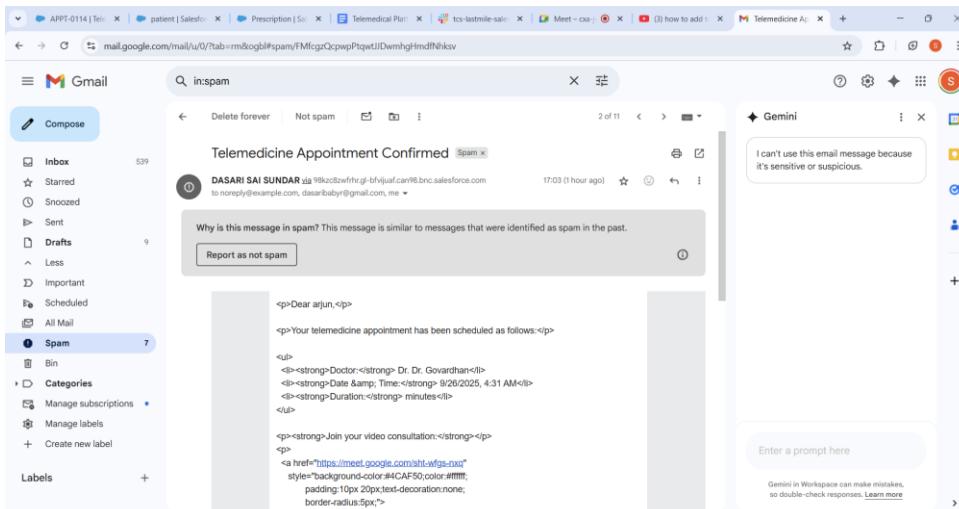


No mail can be updated in the patient mail.

## Workflow Rules Validation → Test Case2:

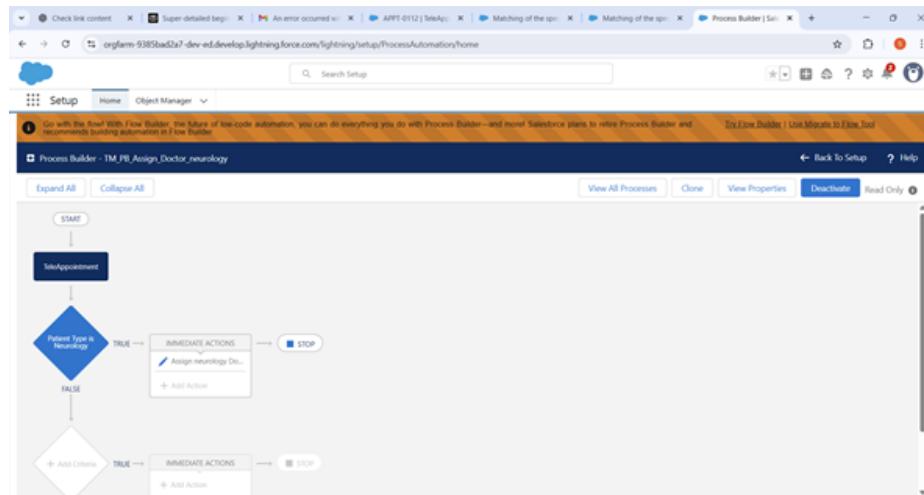


As the status is set as Scheduled and then got the mail.



## 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.

The screenshot shows the Salesforce Lightning Experience. A new "TeleAppointment" record is being created. The "Specialty\_c" field is set to "General". Other fields visible include "Status" (set to "Requested"), "Telemedicine URL", "Duration Minutes", "Notes", "Location\_c", "Room Number c", and "Is priority".

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

TeleAppointment Name: APPT-0126

Appointment ID: APPT-0126

patient: victor

Doctor: Dr. Govardhan

Appointment Date: 9/26/2025, 12:00 PM

Status: Requested

Telemedicine URL:

Duration Minutes:

Notes:

Location\_c:

Room Number c:

Specialty\_c: General

Recent Items

## Process Builder → Test Case2:

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

New TeleAppointment

Information

TelAppointment Name: APPT-0126

Owner: DASARI SAI SUNDAR

\*patient: sam

Doctor: Search Doctors...

Appointment Date: 9/27/2025 12:00 PM

\*Status: Requested

Telemedicine URL:

Save & New Save

Information

Status: Requested

Telemedicine URL:

Duration Minutes:

Notes:

Location\_c:

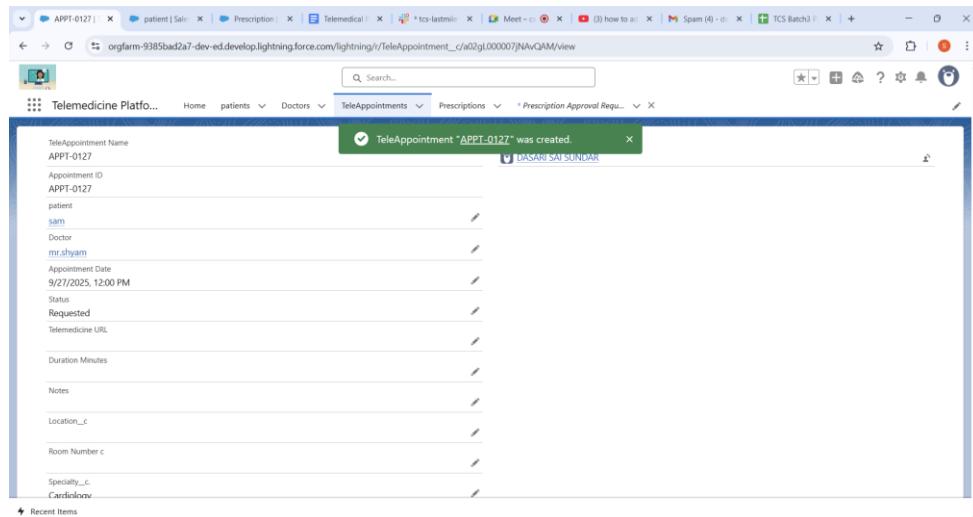
Room Number c:

Specially\_c: Cardiology

Is priority:

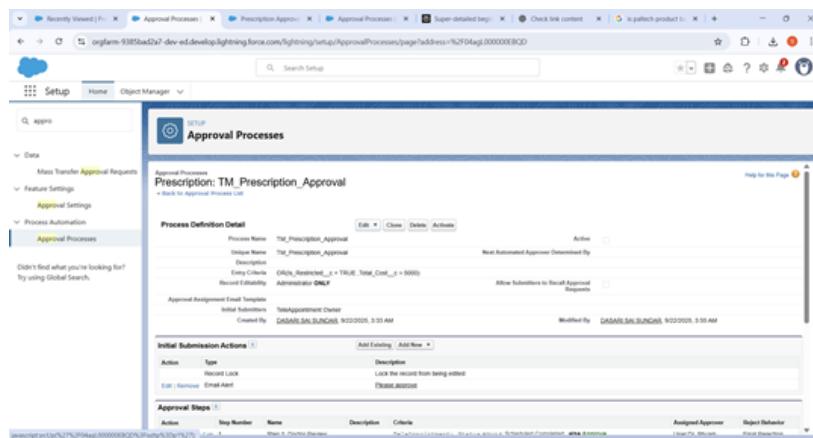
Save & New Save

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



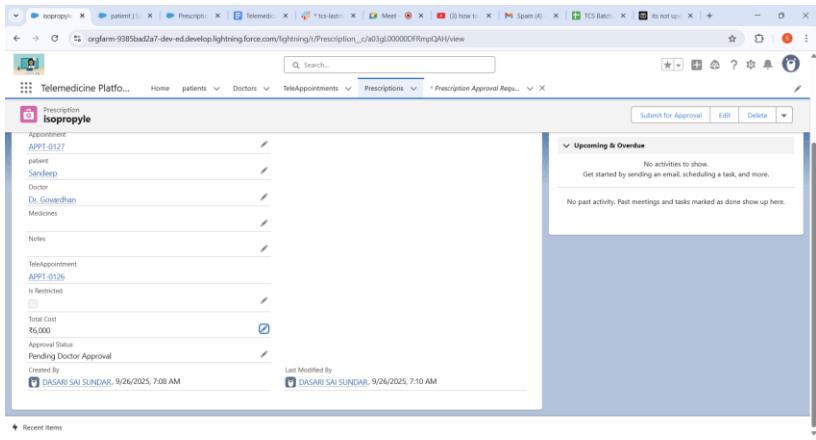
### 13. Approval Process:

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

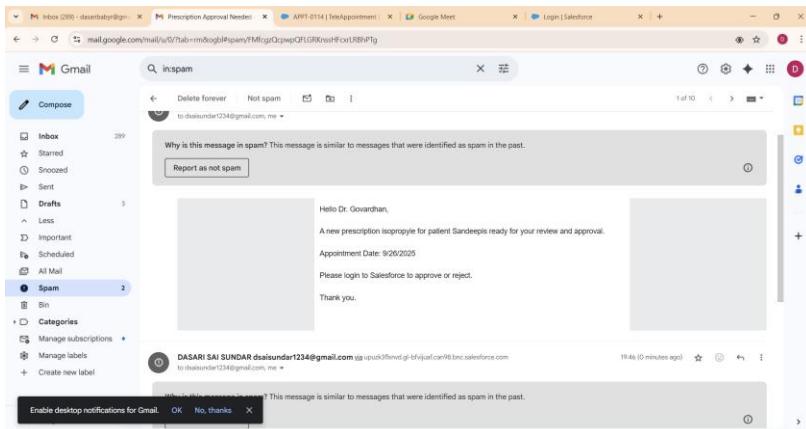


Created the Email templated and condition to send mail is total cost is more than the 5000.As the condition is achieved we should click the button **SUBMIT FOR APPROVAL**.

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



In the email box of the Doctor:



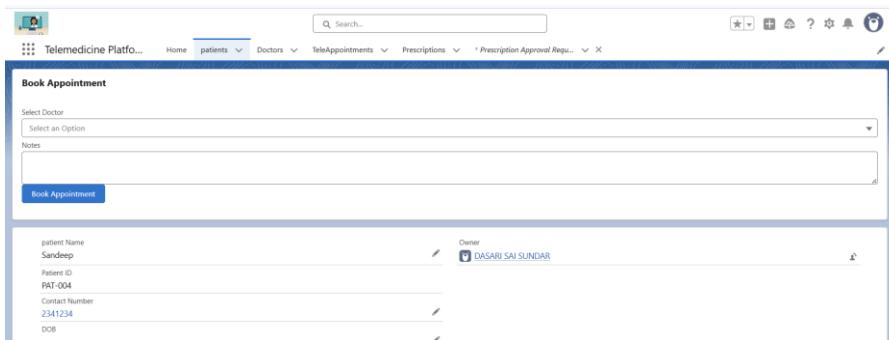
## 14. Lightning Web Component:

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

The screenshot shows the Salesforce Dev Console interface. In the top right, there's a message from Copilot: "Welcome to Copilot. Let's get started". Below it are buttons for "Add content (P) or resources (E) commands" and "Build Workspace". A status bar at the bottom indicates "Review All output carefully before use". The main area shows the deployment log with the following entries:

```
15:55:34.586 Ended SFDX: Deploy This Source to Org
15:55:34.588 Starting SFDX: Deploy This Source to Org
```

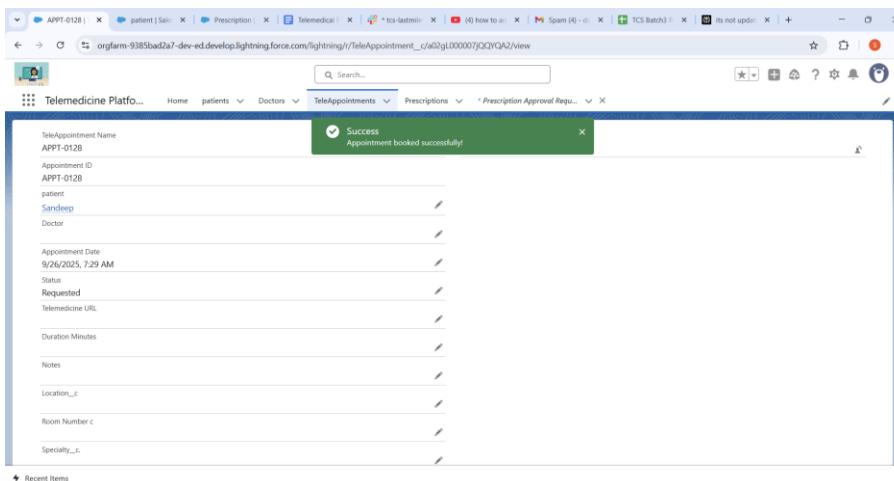
For our First Case: **AppointmentBooking** LWC is created and deployed successfully.



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:

The screenshot shows a Gmail inbox with the search term "in:spam" applied. The first result is an email from "DASARI SAI SUNDAR" with the subject "Telemedicine Appointment Confirmed - Google Meet Link". The email body contains the following text:

```
Dear Dr. Dr. Mitra,  
Your telemedicine appointment on 9/26/2025, 7:29 AM is confirmed.  
Join here: <a href="https://meet.google.com/ht-wfgs-nx">https://meet.google.com/ht-wfgs-nx</a>  
Regards,  
Telemedicine Team
```

Below the email, there is a message box asking "Why is this message in spam? This message is similar to messages that were identified as spam in the past." with a "Report as not spam" button.

Here is the mail for the patient (Sandeepr):

The screenshot shows a Gmail inbox with the search term "in:spam" applied. The first result is an email template with the following body:

```
<p>Dear Sandeep,</p>
<p>Your telemedicine appointment has been scheduled as follows:</p>
<ul>
<li><strong>Doctor:</strong> Dr. Dr. Mitra</li>
<li><strong>Date &amp; Time:</strong> 9/26/2025, 7:29 AM</li>
<li><strong>Duration:</strong> minutes</li>
</ul>
<p><strong>Join your video consultation:</strong></p>
<p><a href="https://meet.google.com/ig-cbbh-zai" style="background-color:#4CAF50;color:white;padding:10px 20px;text-decoration:none; border-radius:5px;">Join Video Call</a></p>
<p>Meeting Link: <a href="https://meet.google.com/ig-cbbh-zai">https://meet.google.com/ig-cbbh-zai</a></p>
<p><strong>Important Notes:</strong></p>
<ul>
<li>Please join the meeting 5 minutes early</li>
</ul>
```

## 15. Import the Data from the Excel:

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

The screenshot shows a Microsoft Excel spreadsheet titled "Book1" with data in the first sheet named "Sheet1". The data is organized in a table with the following columns: A (Patient Name), B (Contact Number), C (Gender), D (Patient Address), E (Medical History), and F (Patient Age). The data entries are as follows:

	A	B	C	D	E	F
1	Patient Name	Contact Number	Gender	Patient Address	Medical History	Patient Age
2	John Doe	1234567890	Male	123 Main Street	Hypertension, Blood pressure	43
3	Jane Smith	9876543210	Female	456 Elm Street	Allergy to penicillin	47
4	Sam Wilson	5555555555	Male	789 Oak Street	Name	33

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

The screenshot shows the 'Book Appointment' page. At the top, there's a dropdown menu 'Select Doctor' with 'Select an Option'. Below it is a 'Notes' input field. A large blue button at the bottom right says 'Book Appointment'. Below this button, patient details are listed: Name (Jane Smith), Patient ID (PAT-010), Contact Number (0987 654-3210), DOB, Gender (Female), Medical History (Allergy to penicillin), and a 'Recent Items' section.

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

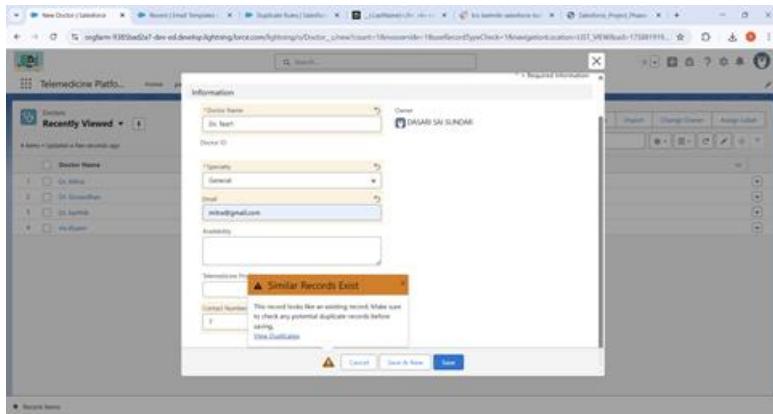
The screenshot shows the patient profile for John Doe. It includes fields for Name (John Doe), Patient ID (PAT-009), Contact Number (1234 456-7890), DOB, Gender (Male), Medical History (High blood pressure), Patient email (john.doe@email.com), and Patient Age (43). The 'Last Modified By' field shows 'DASARI SAI SUNDAR, 9/25/2025, 9:20 AM'. Below the profile, there's a 'Medical History' section with a note 'No history available'.

## 16. Duplicate Rules:

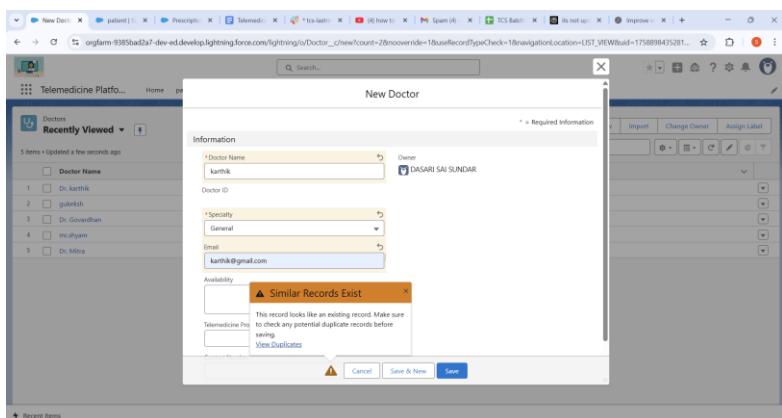
Created the duplicate rule with the Matching rule with Email.

The screenshot shows the 'Duplicate Rules' configuration screen. Under 'Data > Duplicate Management > Duplicate Rules', a rule named 'Prevent Doctor Duplicates' is selected. The 'Rule Name' is 'Doctor Duplicate Rule'. The 'Object' is 'Doctor'. The 'Matching Rule' is 'Doctor\_Email\_Match'. The 'Matching Criteria' is 'Email: Email match MatchValue = TRUE and Doctor\_email\_Matches = TRUE'. The 'Modified By' field shows 'DASARI SAI SUNDAR, 9/25/2025, 9:48 AM'.

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

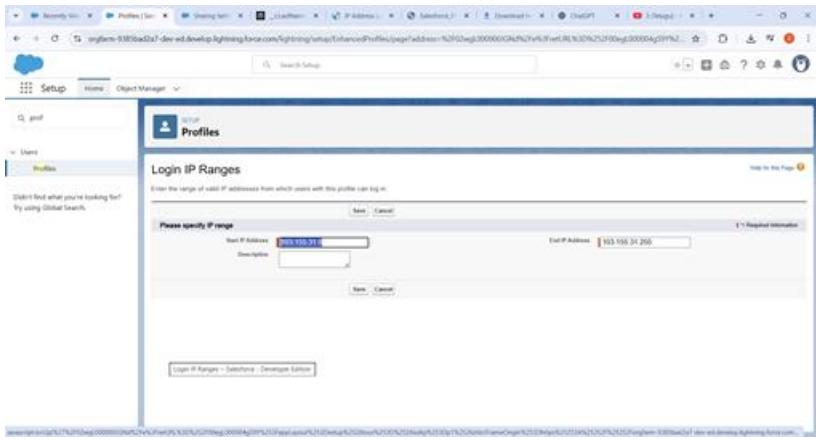


**Duplicate Rules → Test Case2**→As we same email is given then the **Warning** may appear as the same mail.

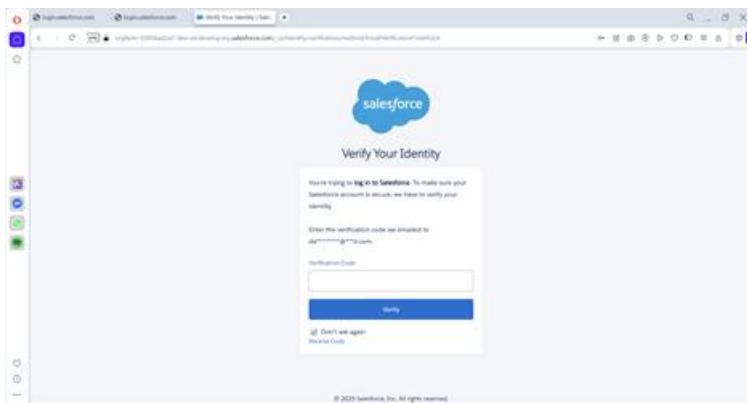


## 17. Login IP Ranges:

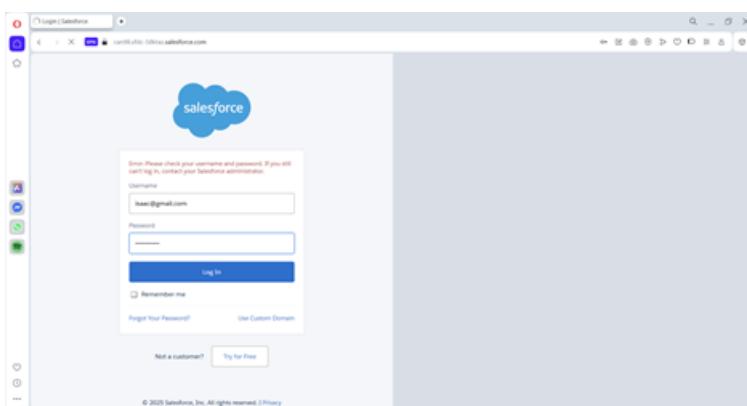
Enter a Start IP Address and an End IP Address that define your organization's trusted network range covers all IP addresses from 103.155.31.0 up to 103.155.31.255.



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



- From my friends home its not opening



**THE END**