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.

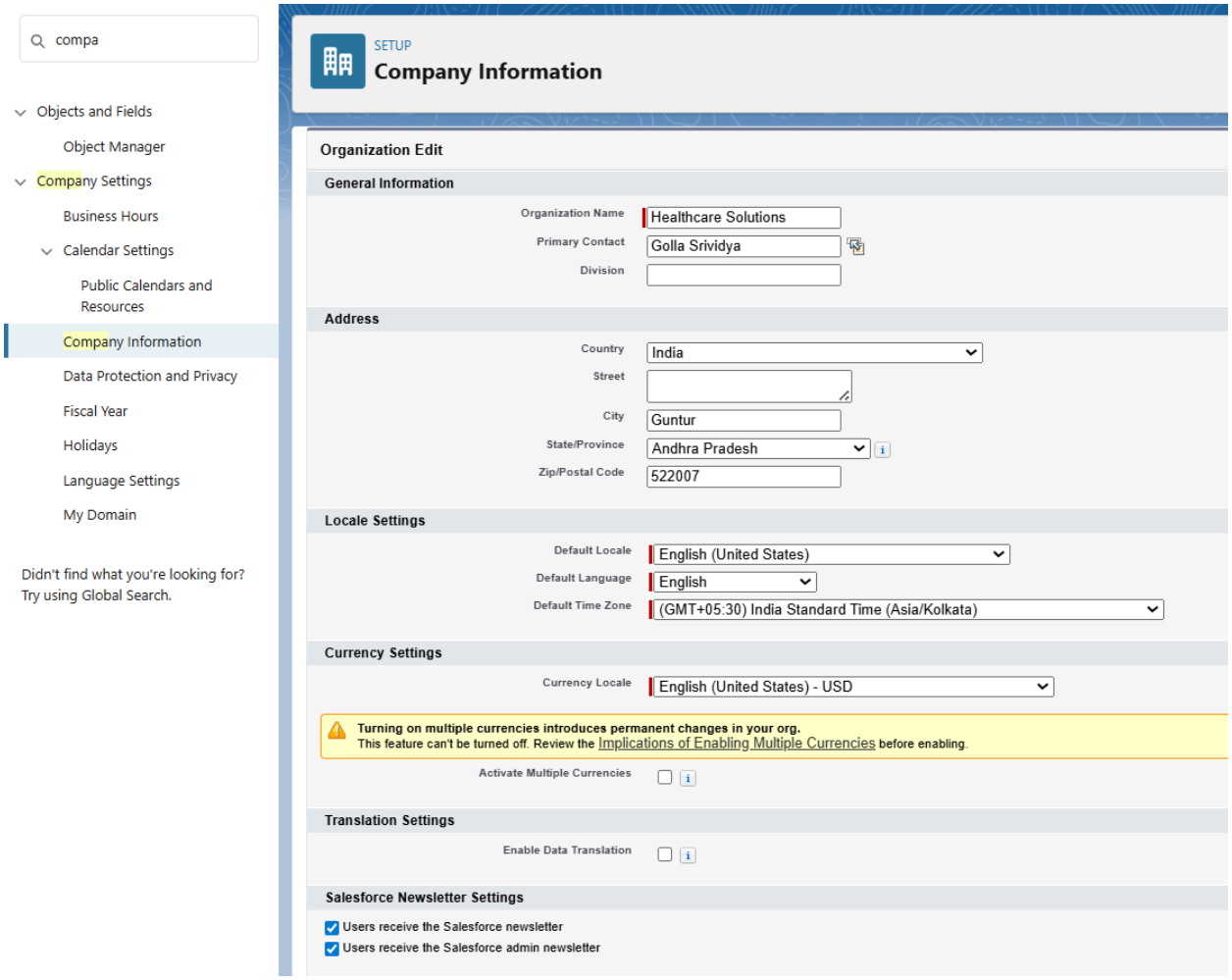
Phase 2: Org Setup & Configuration

1)Salesforce Editions:

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

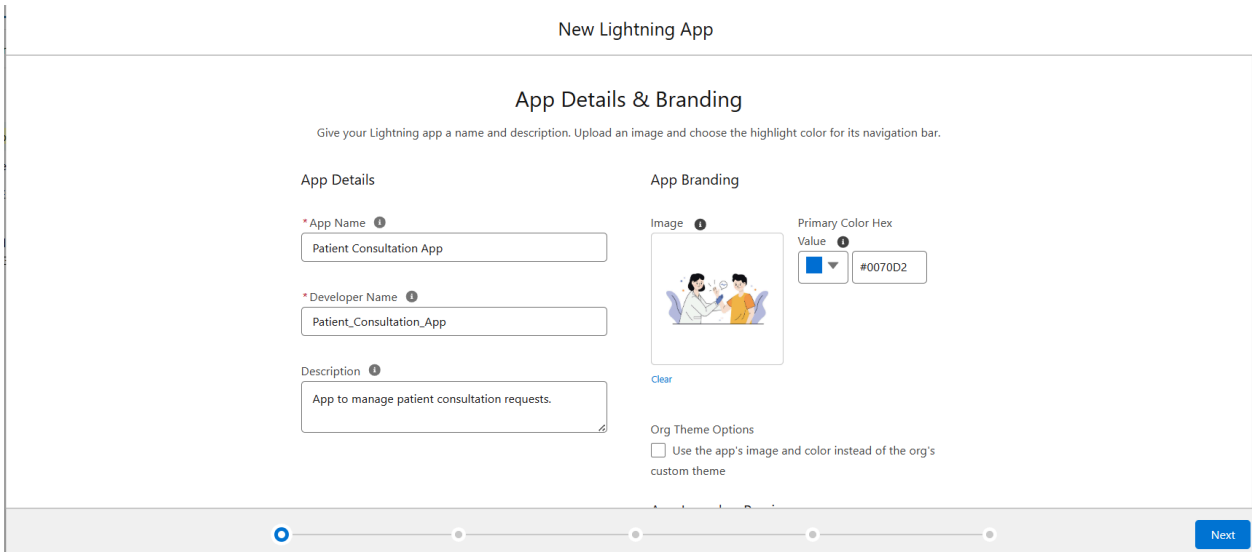
2) Basic Company Setup:

Gear icaon->set up->Company information



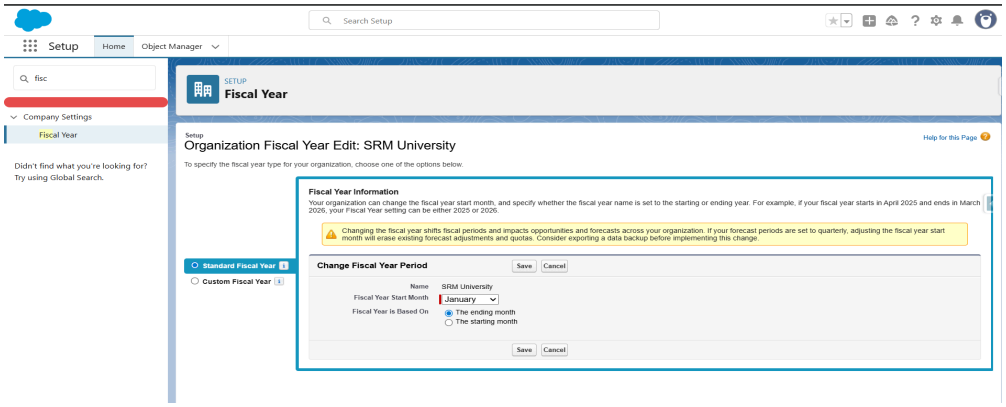
3)Creation of Lightning App:

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



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.**



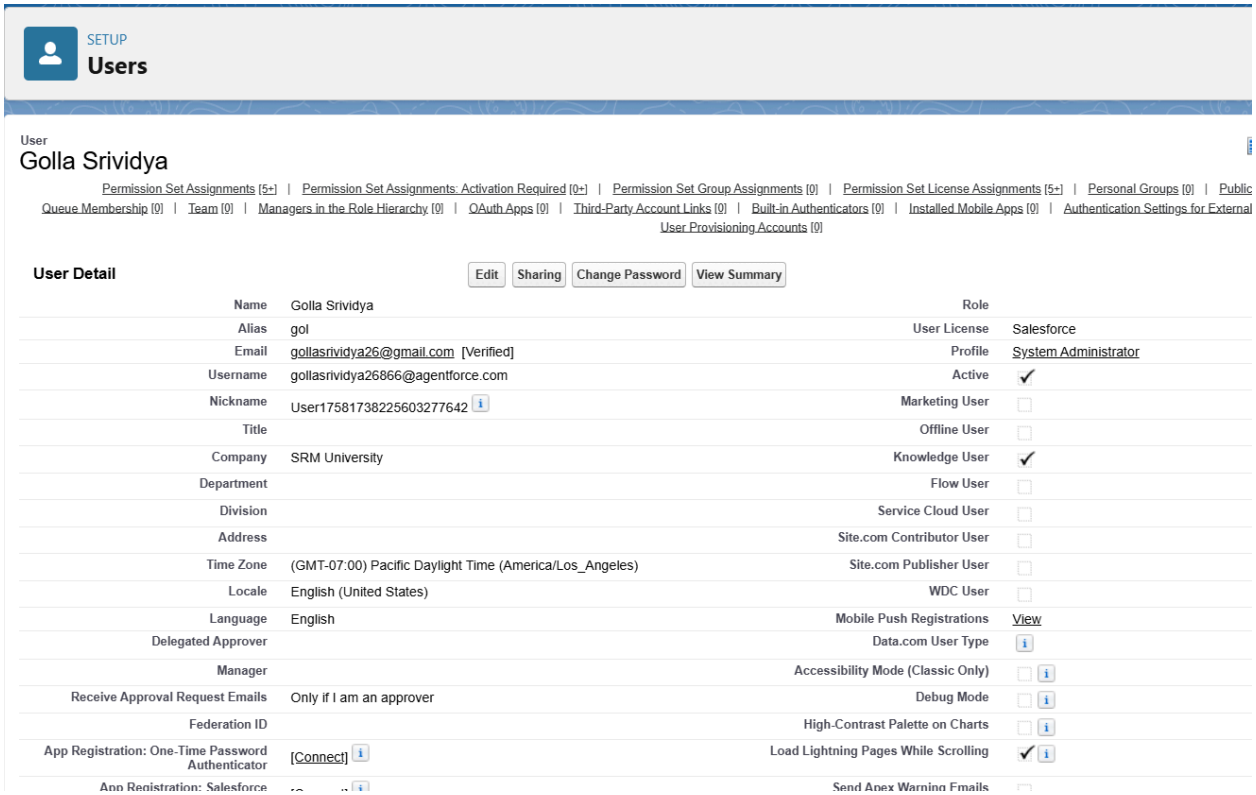
2. User Setup & Licenses

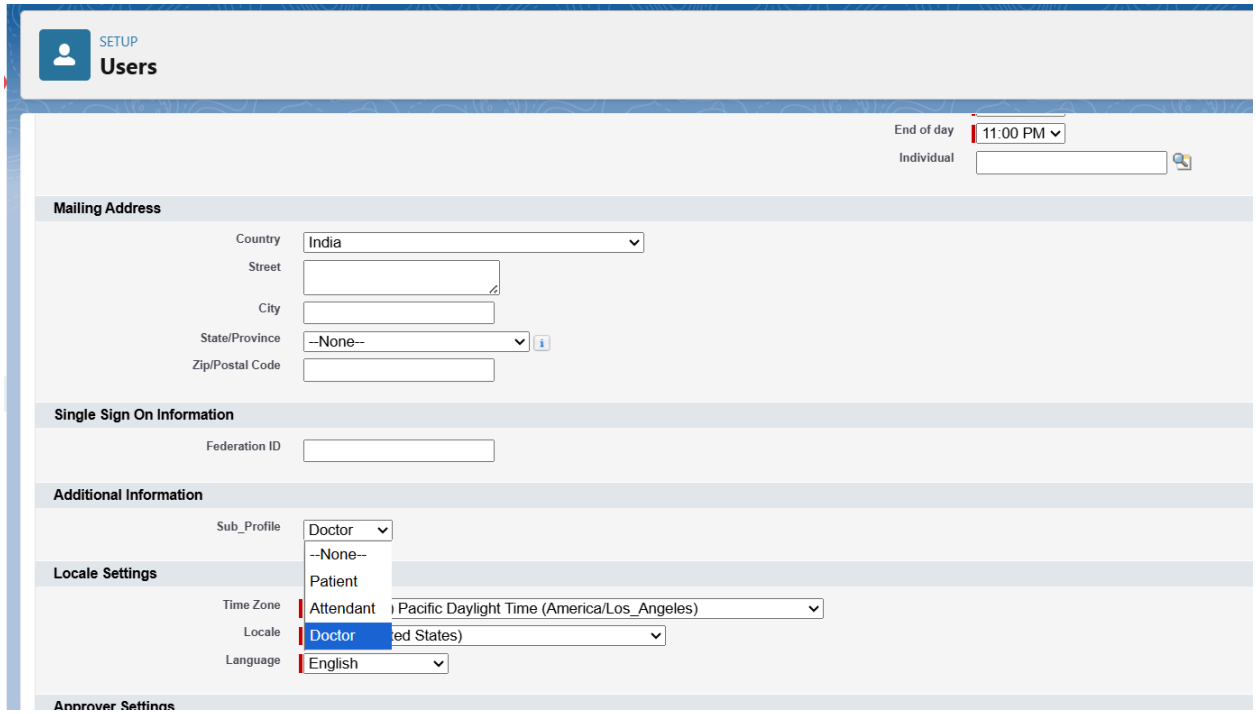
- **Users: Each doctor, attendant, or admin requires a Salesforce user.**

- **Licenses:**

    - **Salesforce License: access based on sub profile(for doctors/admins).**

- **Screenshot suggestion: Show list of users with their license type.**

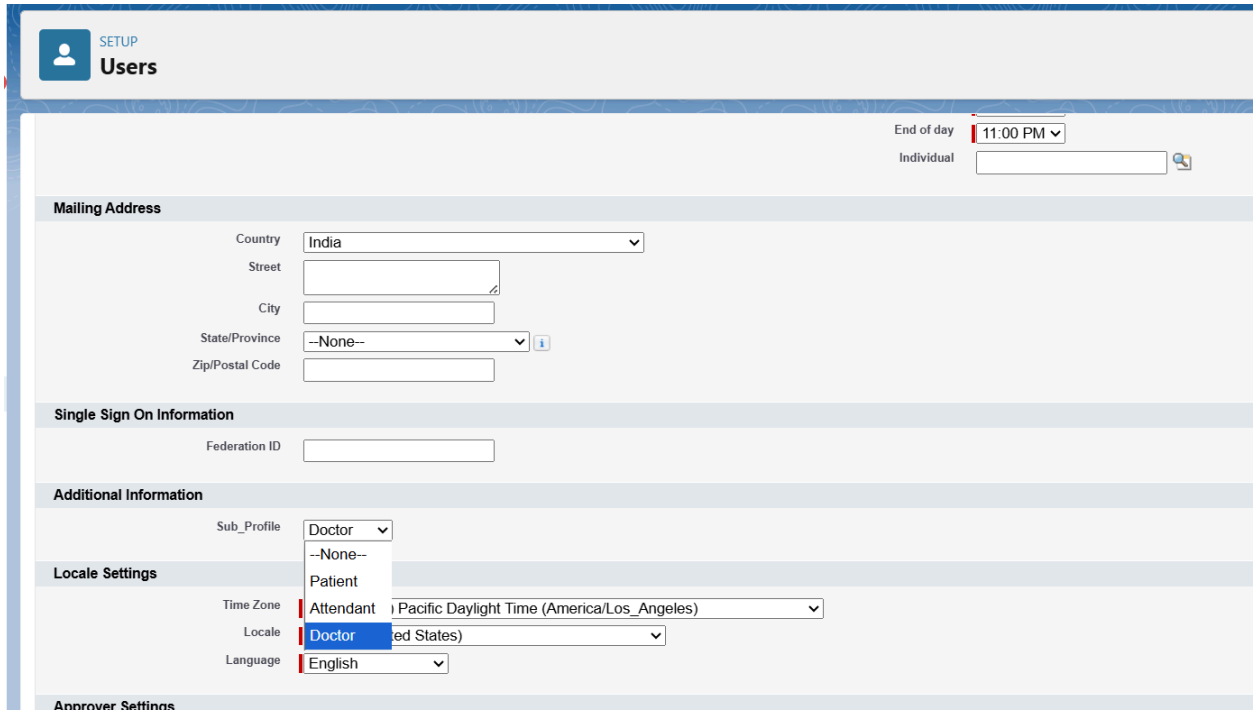




3. Profiles

- **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



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:

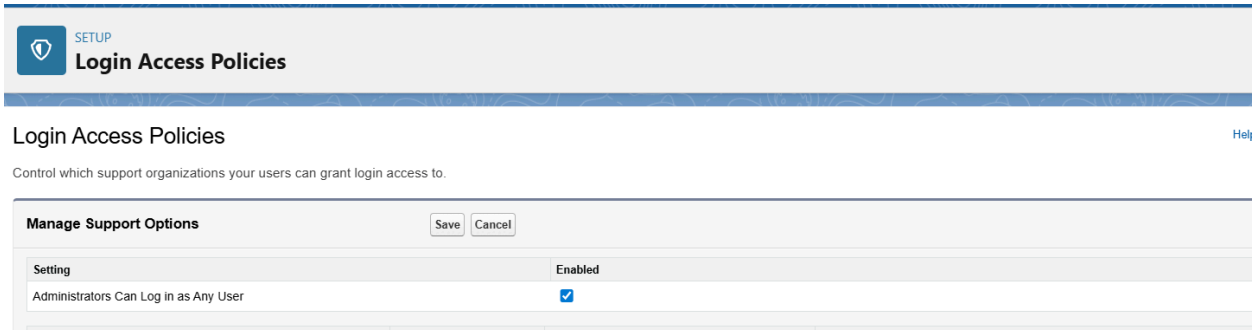
- **Patient__c: Default OWD is used. Patients can see only their own records, while Doctors and Attendants can see patient records based on the Sub Profile access logic implemented in the project.**

- **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.**

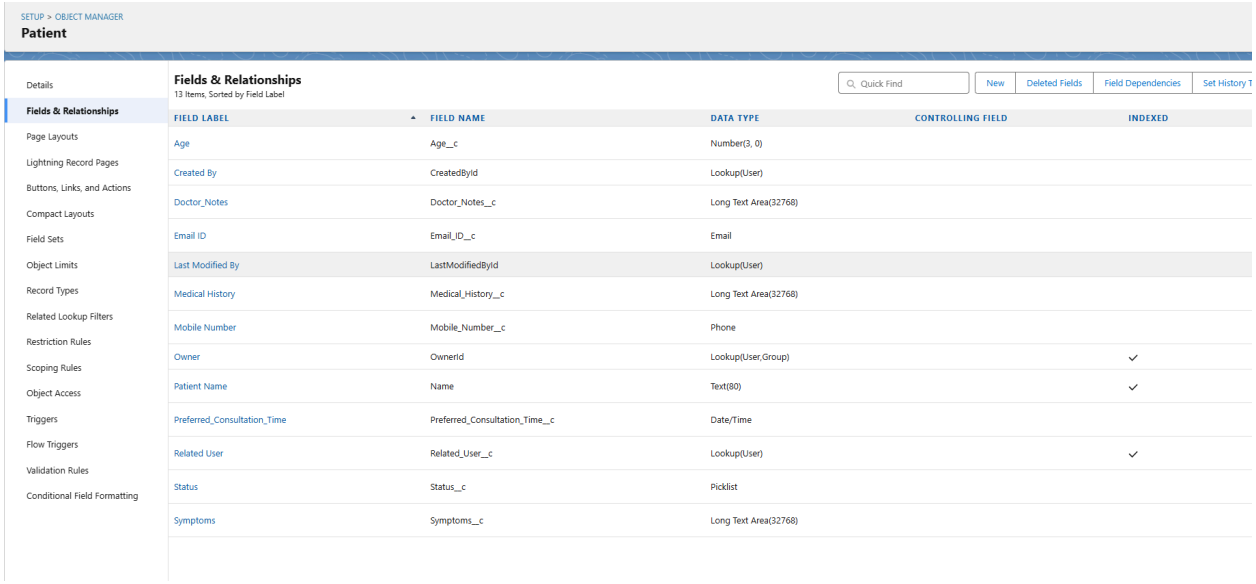
- **Project relevance: Admin login access for troubleshooting.**



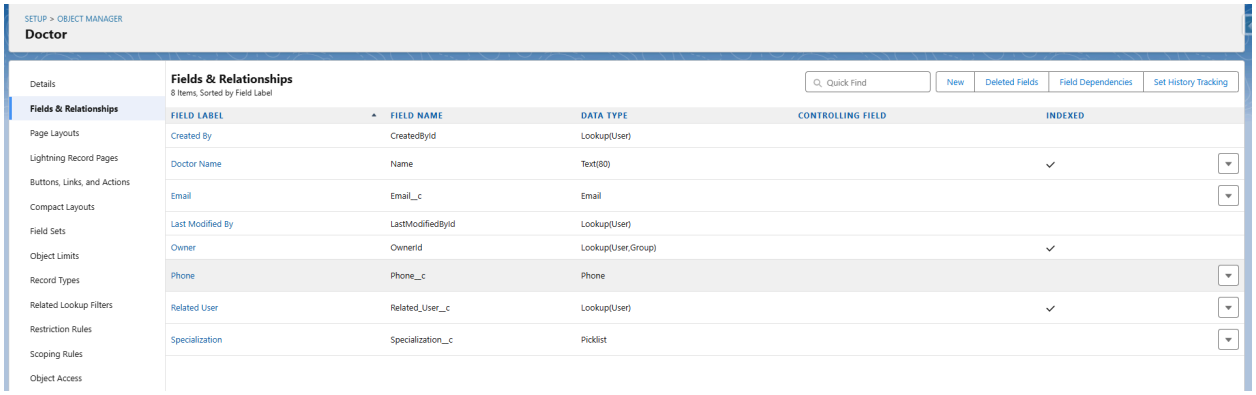
8. Dev Org Setup

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

Patient:

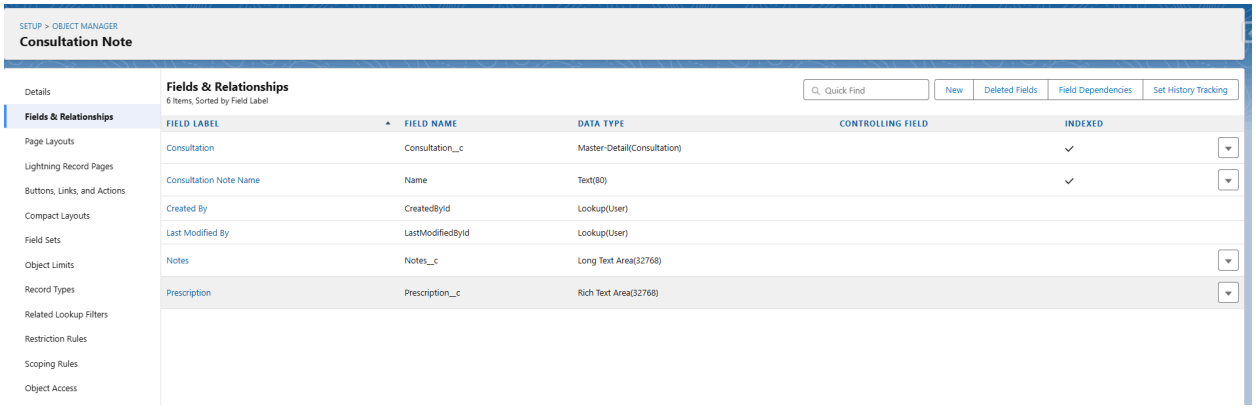


Doctor:



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Doctor Name	Name	Text(80)		✓
Email	Email__c	Email		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓
Phone	Phone__c	Phone		
Related User	Related_User__c	Lookup(User)		✓
Specialization	Specialization__c	Picklist		

Consultation notes:



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Consultation	Consultation__c	Master-Detail(Consultation)		✓
Consultation Note Name	Name	Text(80)		✓
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Notes	Notes__c	Long Text Area(32768)		
Prescription	Prescription__c	Rich Text Area(32768)		

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

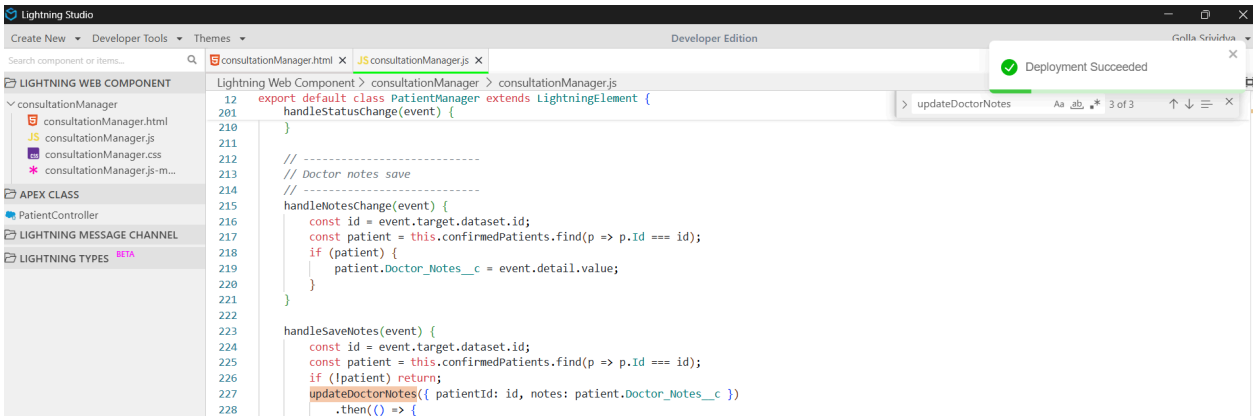
Project Relevance:

- **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

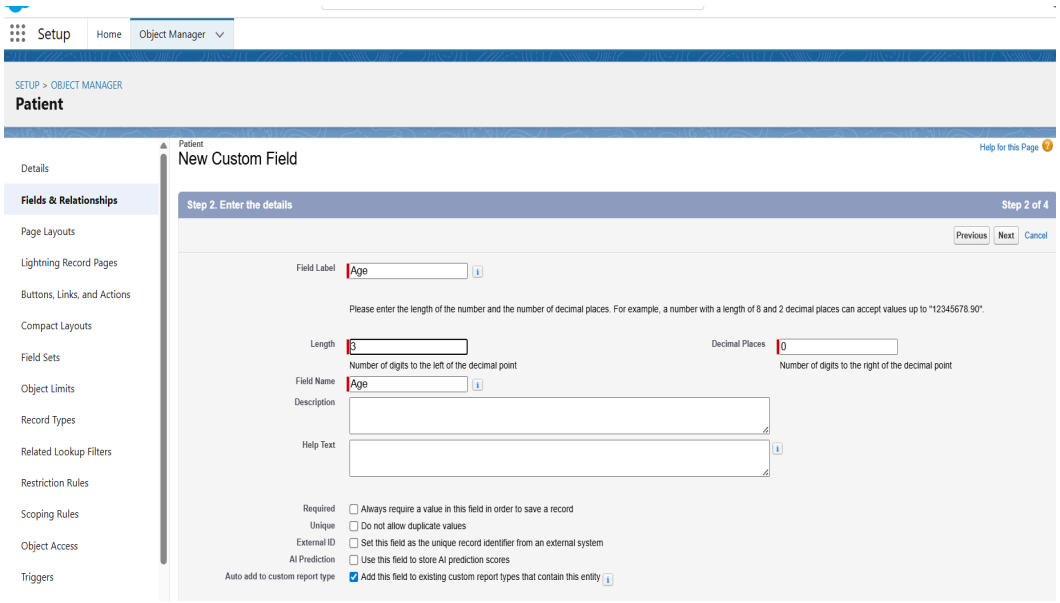
1. Go to **Setup (⚙️)** → **Object Manager** → **Create** → **Custom Object**.
2. Fill details:
    - Label: **Patient**
    - Plural Label: **Patients**
    - Object Name: **Patient**
    - Record Name: **Patient Name** → Data Type: **Text**
    - Check: **Allow Reports, Allow Activities, Track Field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

Step 2: Create Fields

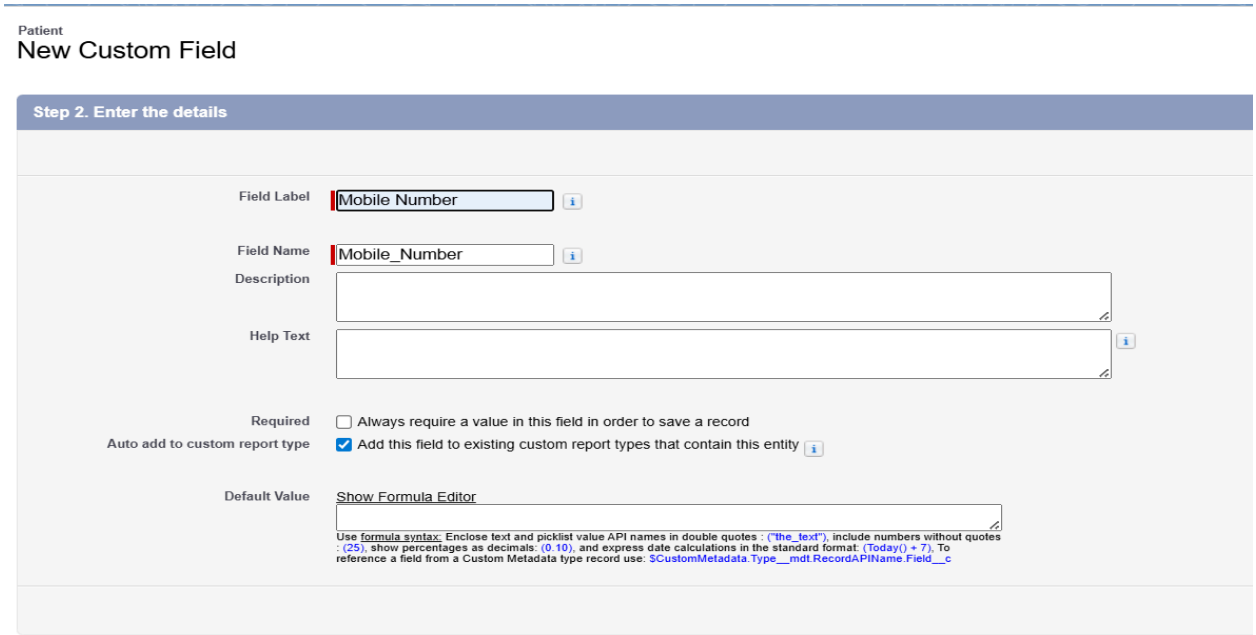
- Go to **Patient__c** → **Fields & Relationships** → **New**.

- Add the following one by one:

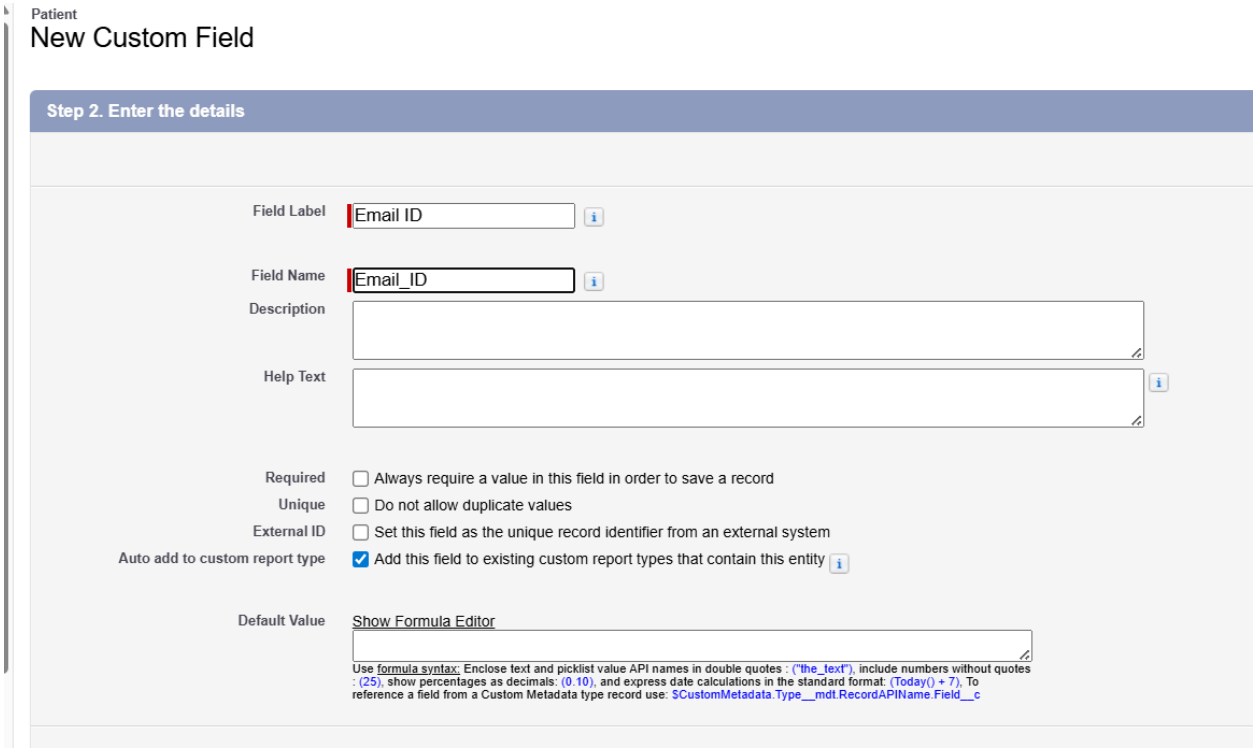
1. **Age__c**
    - Data Type: **Number**
    - Length: 3, Decimal Places: 0
    - Label: Age
    - Next → Save.



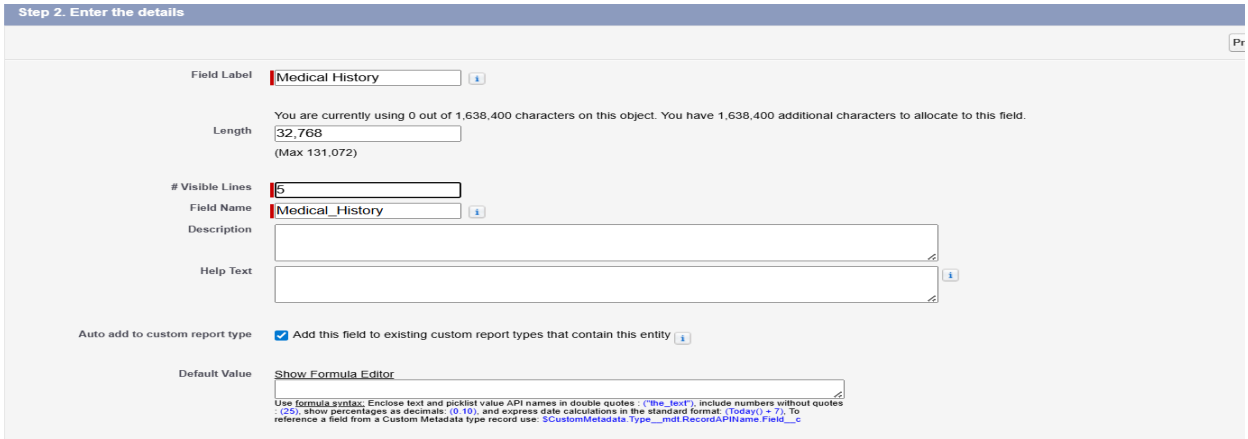
2. **Mobile__c**
    - Data Type: **Phone**
    - Label: Mobile Number
    - Next → Save.



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

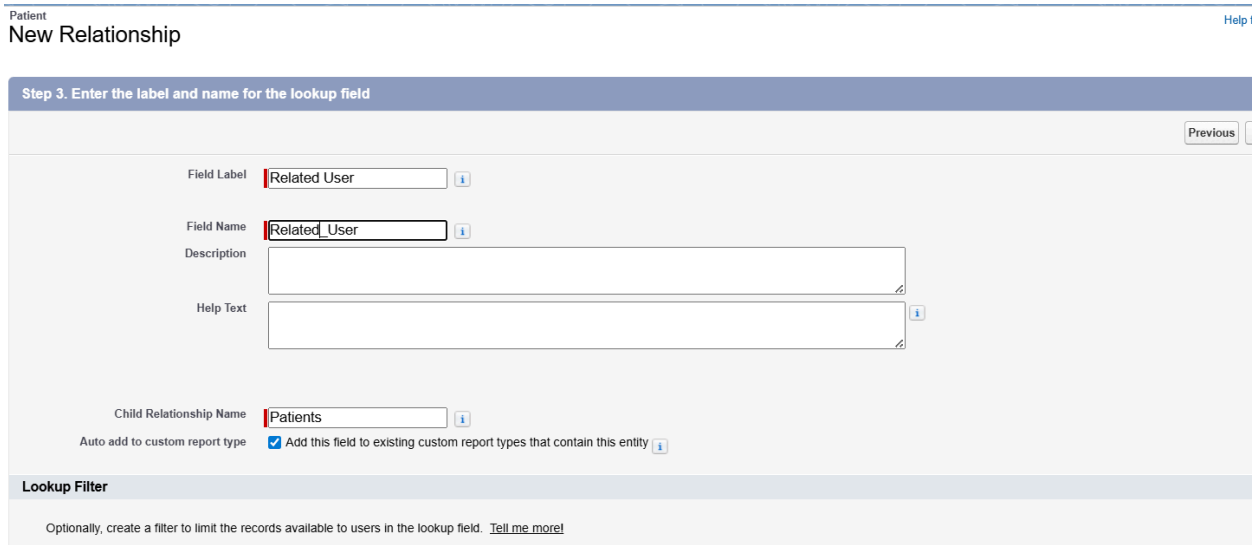


4. **History__c**
    - Data Type: **Long Text Area**
    - Label: Medical History
    - Length: 32000, Visible Lines: 5
    - Next → Save.



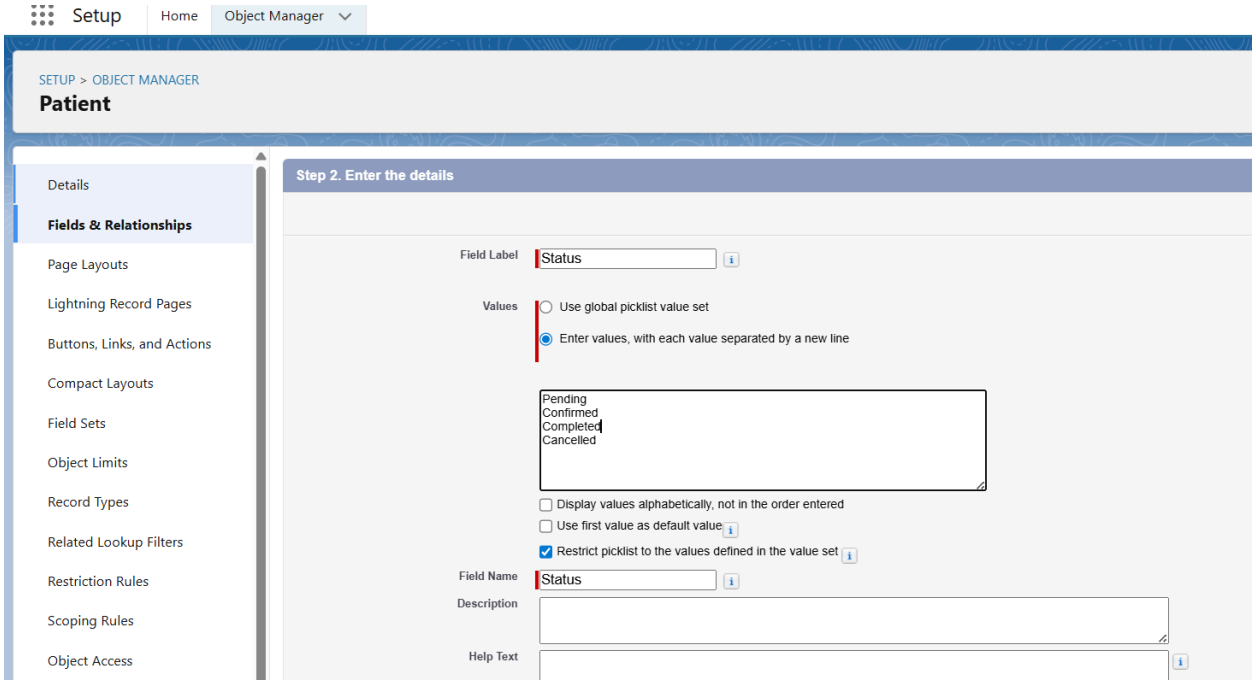
5. **User__c**

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



Status

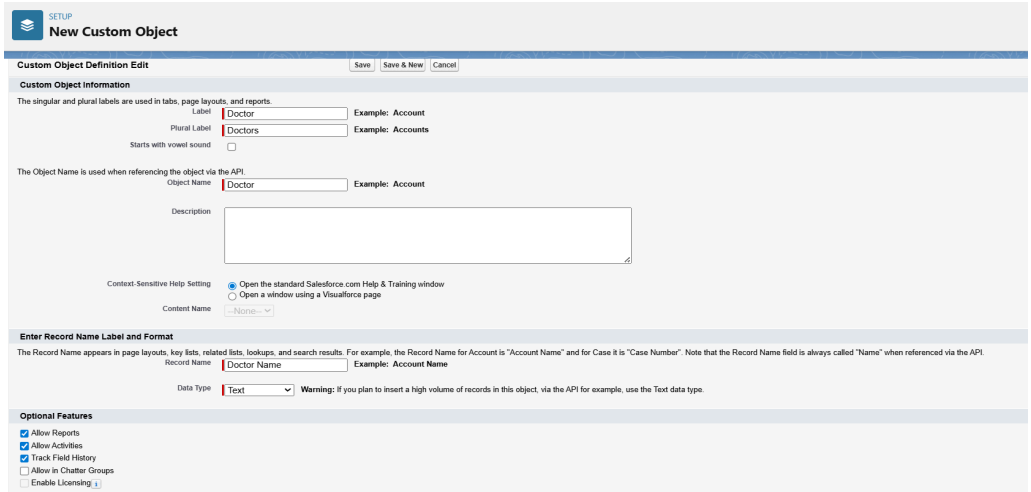
Picklist : **Pending, Confirmed, Completed, Cancelled**



2. Create `Doctor__c` Object

Step 1: Create Object

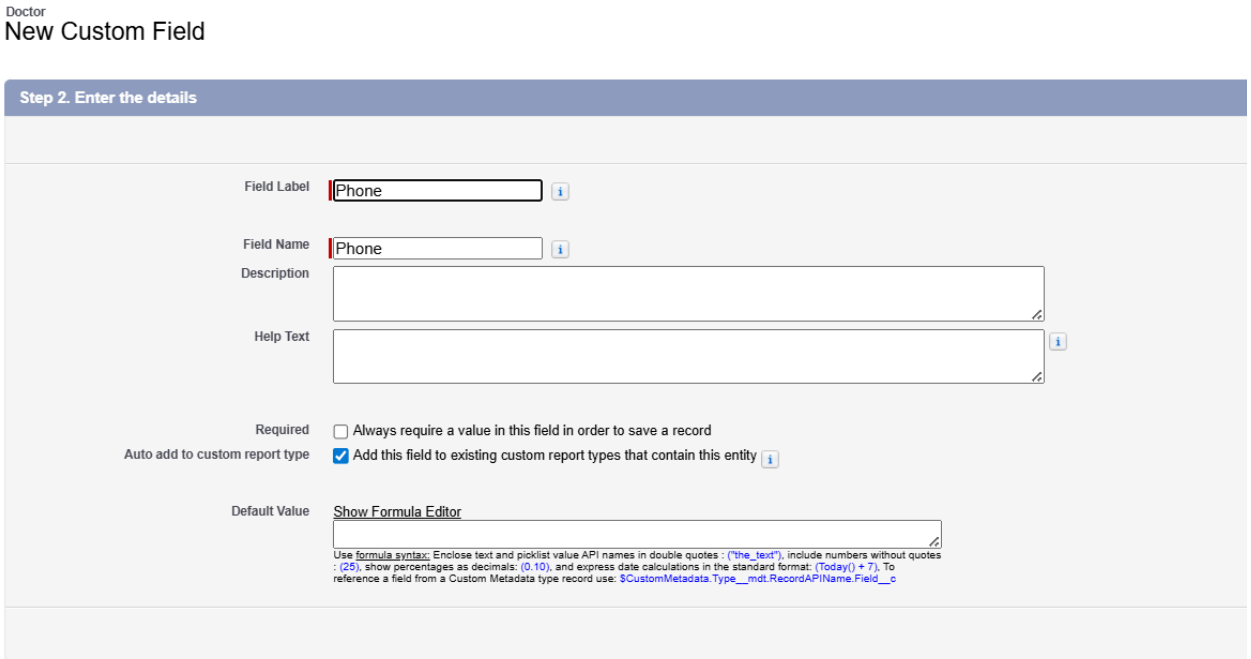
- **Same path: Setup → Object Manager → Create → Custom Object.**
- **Fill:**
  - **Label: Doctor**
  - **Plural: Doctors**
  - **Record Name: Doctor Name (Text)**
  - **Enable Reports, Activities, Track History**
  - **Save.**



Step 2: Create Fields

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

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



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

Doctor
New Custom Field

Step 2. Enter the details

Field Label: Email

Field Name: Email

Description:

Help Text:

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

Unique: ☐ Do not allow duplicate values

External ID: ☐ Set this field as the unique record identifier from an external system

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

4. User__c

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

Doctor
New Relationship

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

Field Label: Related User

Field Name: Related_User

Description:

Help Text:

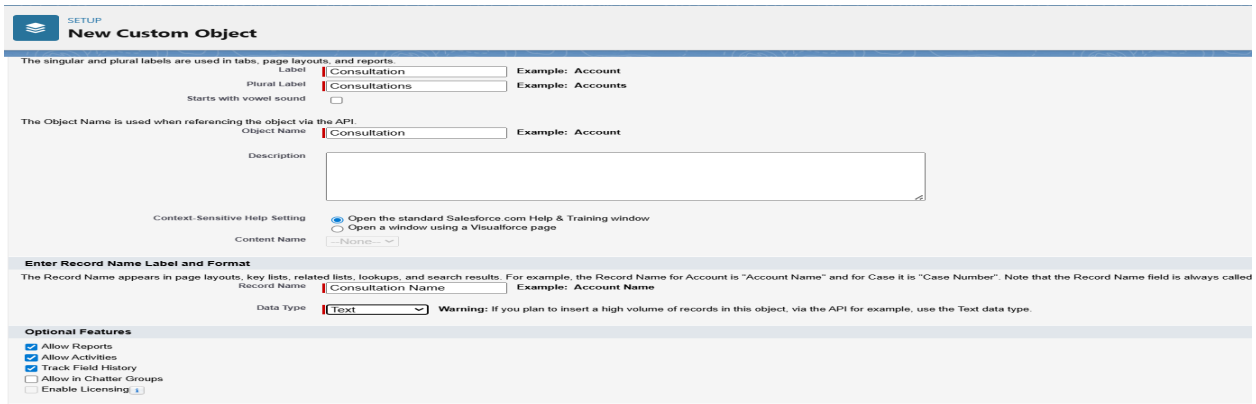
Child Relationship Name: Doctors

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

3. Create `Consultation__c` Object

Step 1: Create Object

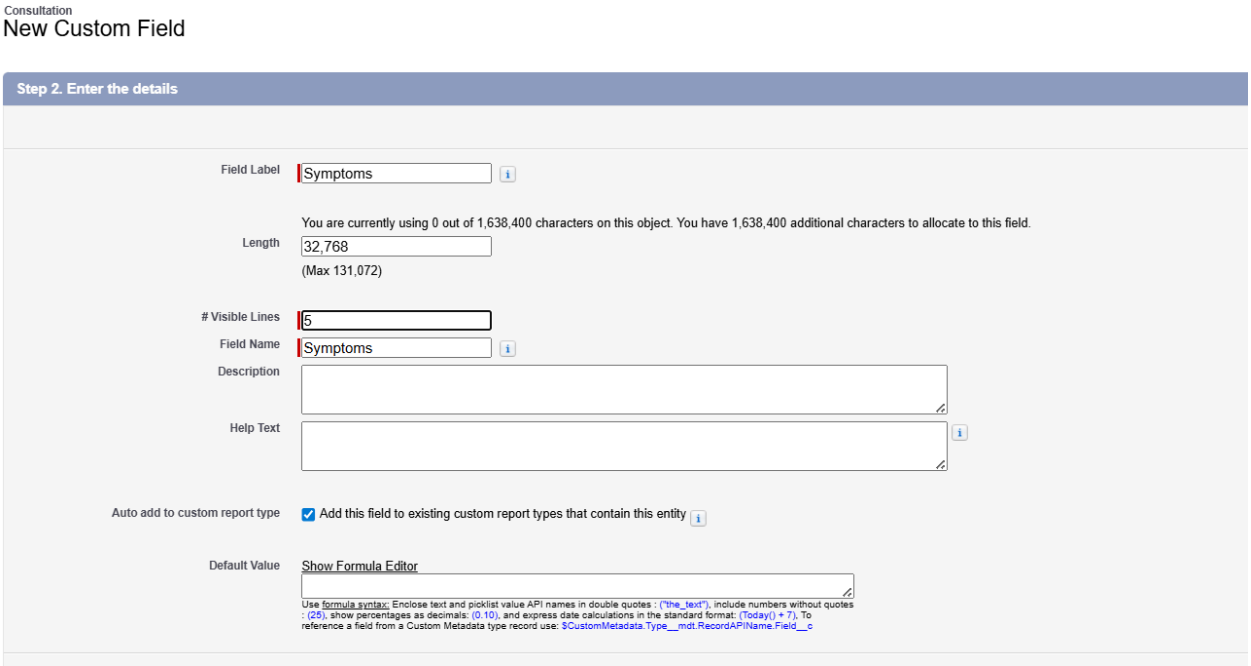
- **Label: Consultation**
- **Plural: Consultations**
- **Record Name: Consultation Name**
- **Starting Number: 1**
- **Enable Reports, Activities, Track History**
- **Save.**



Step 2: Create Fields

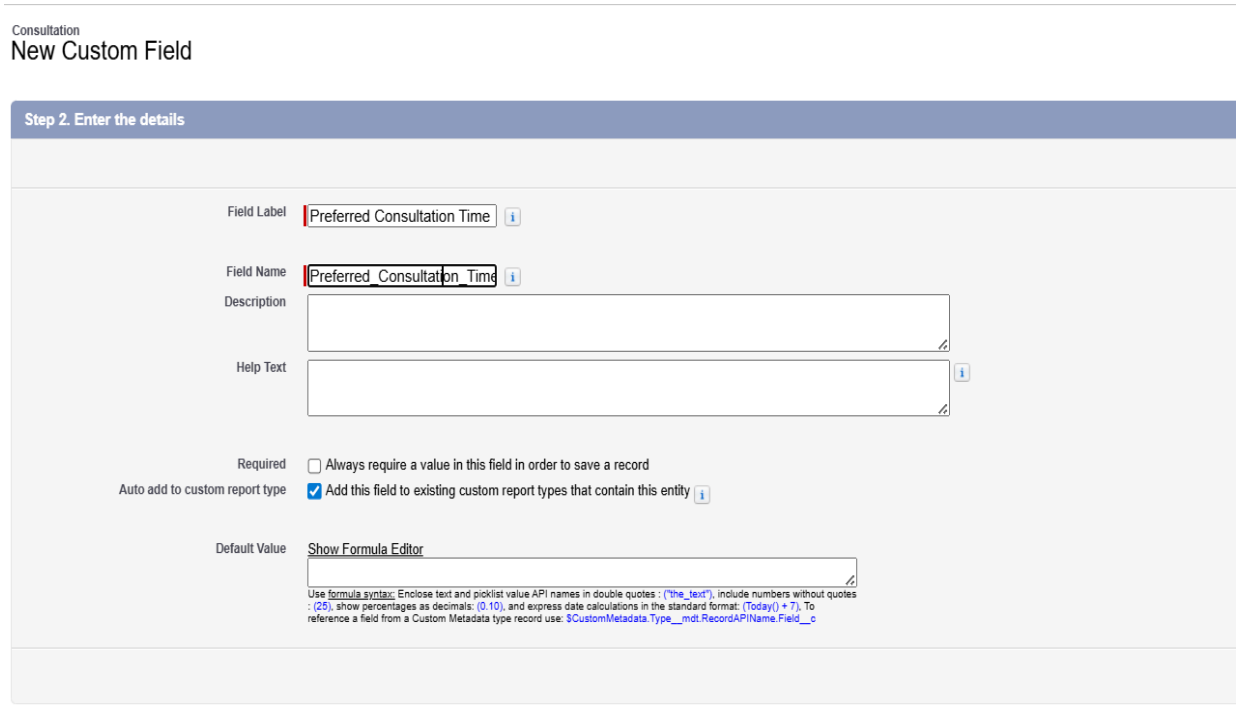
1. **Symptoms__c**
    - **Data Type: Long Text Area**

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



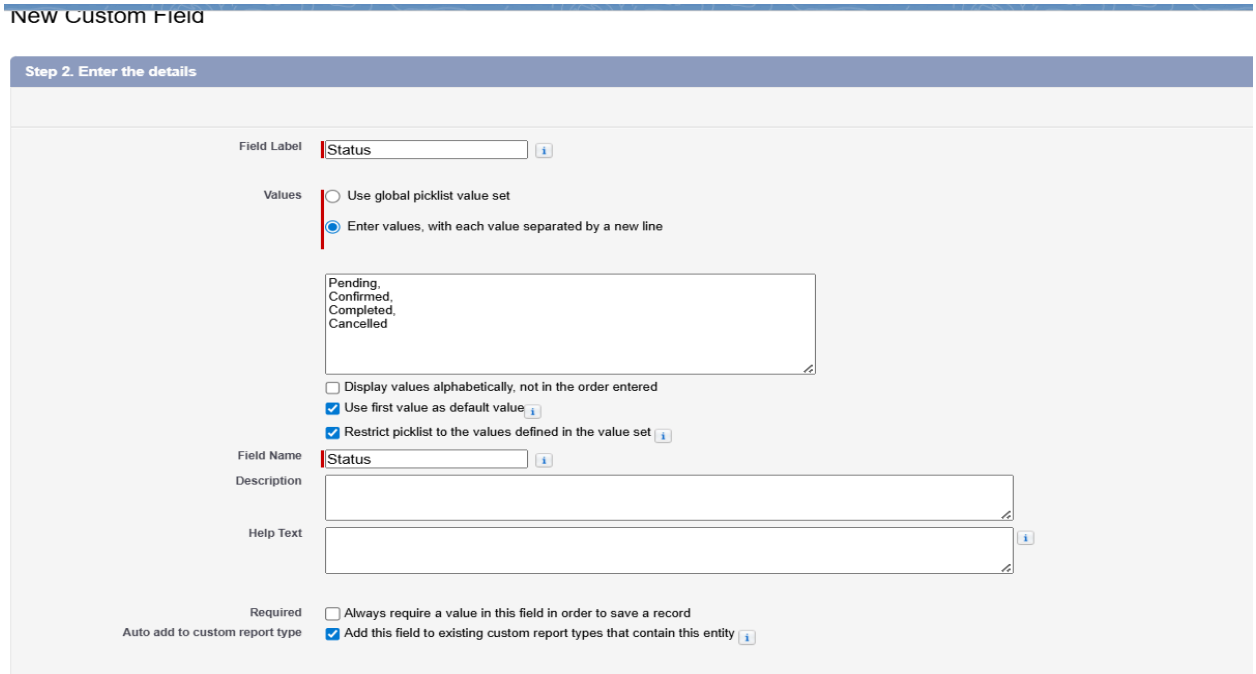
2. Preferred_Consultation_Time__c

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

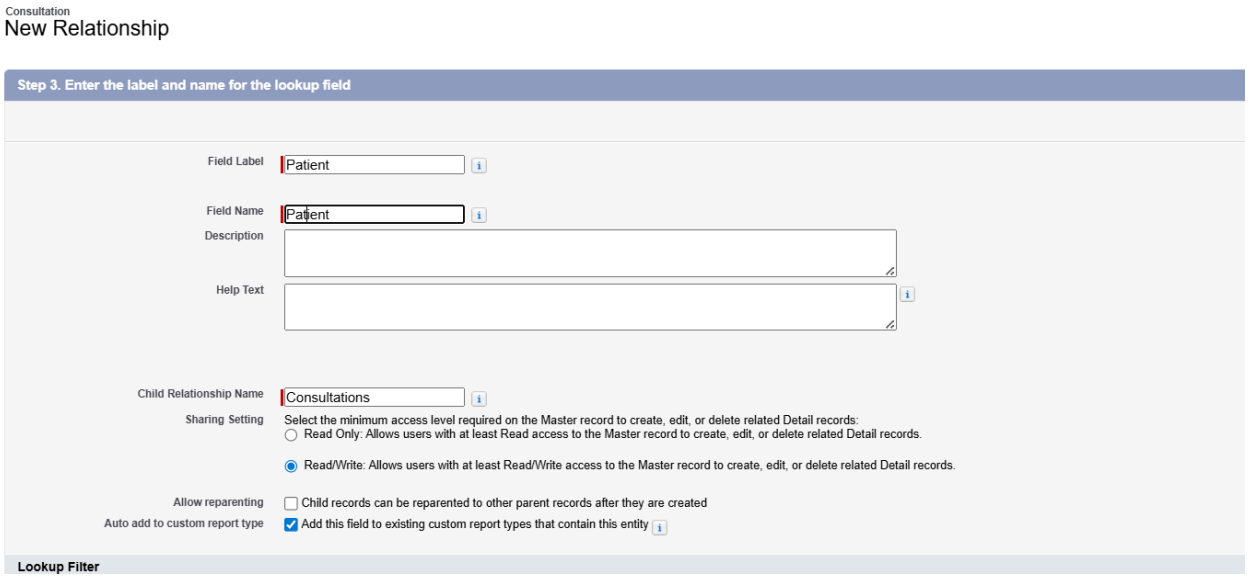


3. Status__c

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

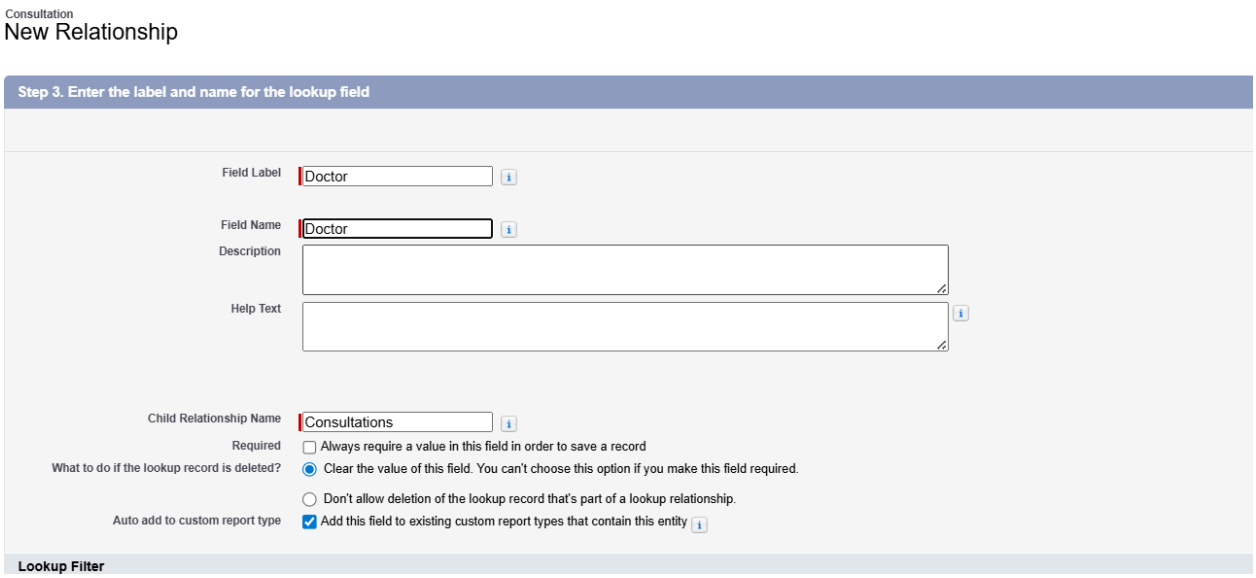


4. **Patient__c**
    - **Data Type: Master-Detail Relationship**
    - **Related To: Patient__c**
    - **Label: Patient**
    - **Save.**



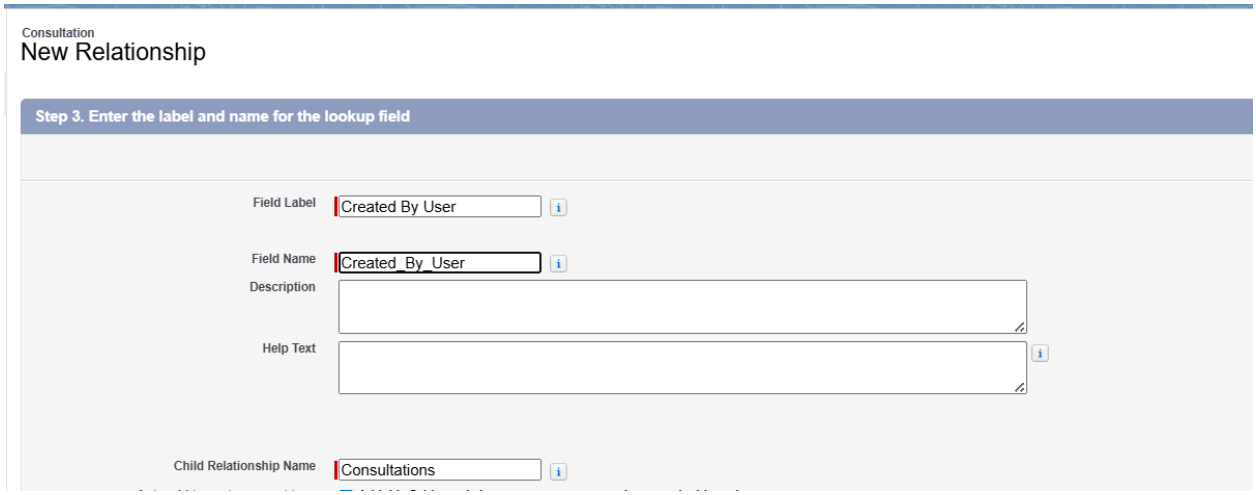
5. Doctor__c

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



6. Created_By_User__c

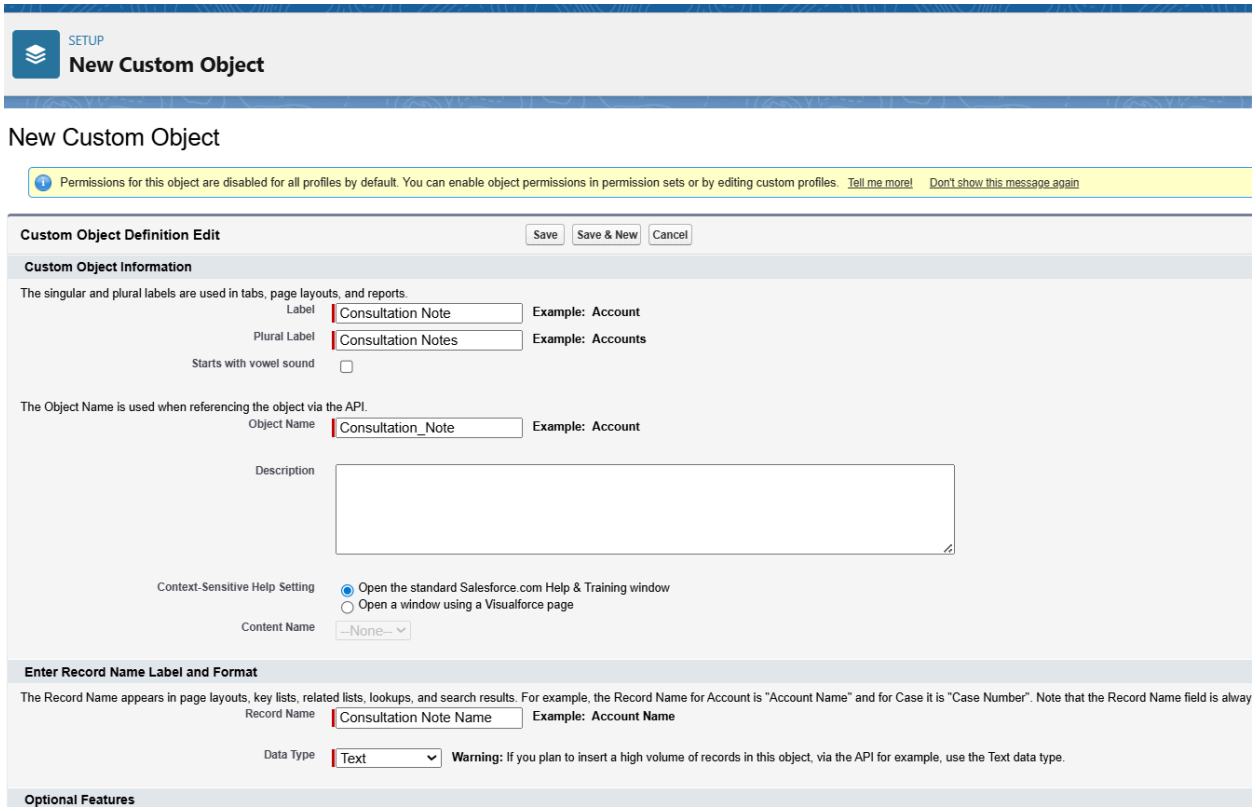
- **Data Type: Lookup Relationship**
- **Related To: User**
- **Label: Created By User**
- **Save.**



4. Create `ConsultationNote__c` Object

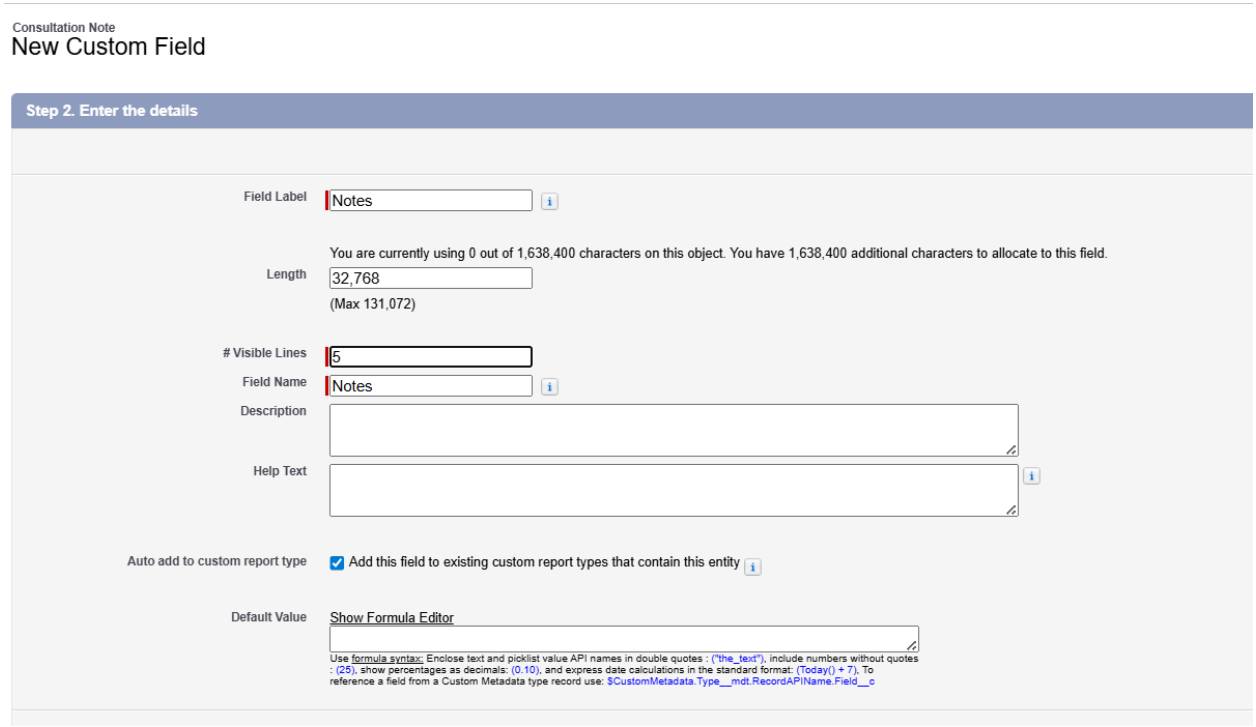
Step 1: Create Object

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



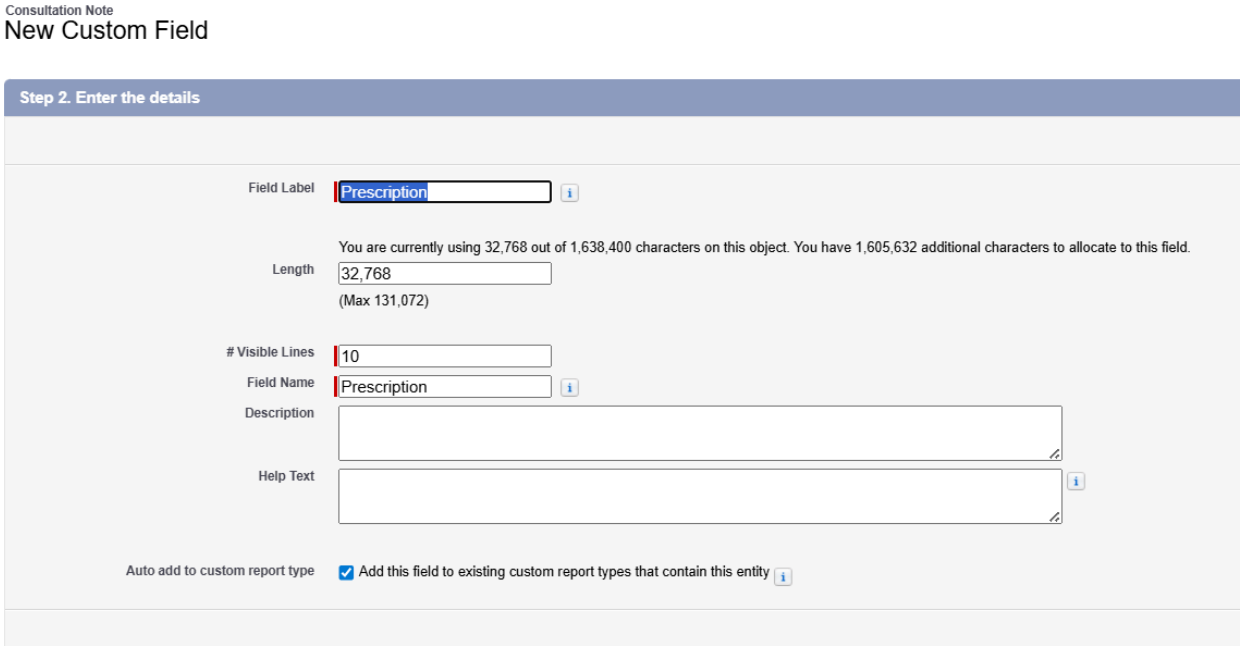
Step 2: Create Fields

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



2.Prescription__c

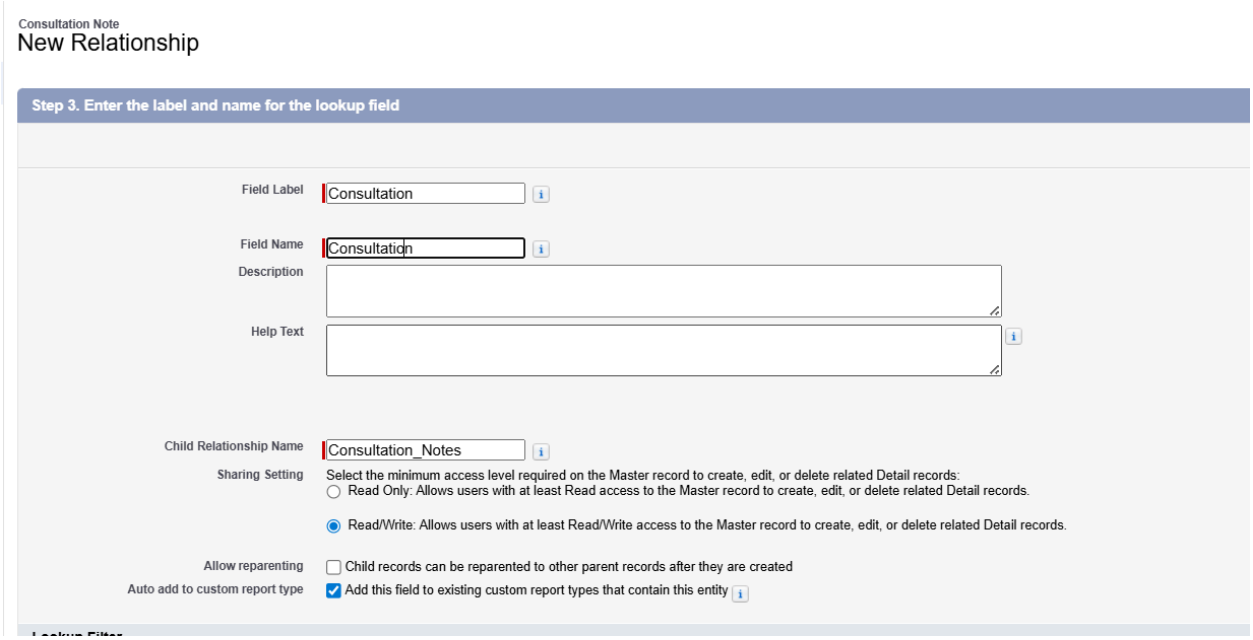
- **Data Type: Rich Text Area**
- **Label: Prescription**
- **Length: 32768, Visible Lines: 5**
- **Save.**



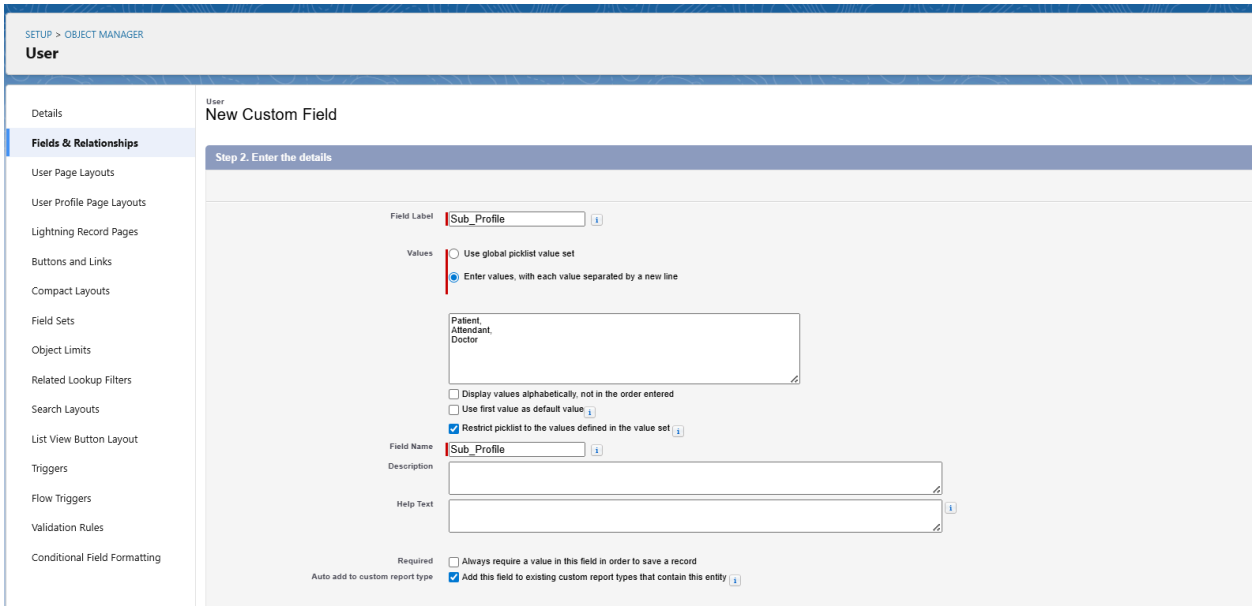
3.Consultation__c

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

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



5)Create field in User Object



Step 2: Relationships Summary

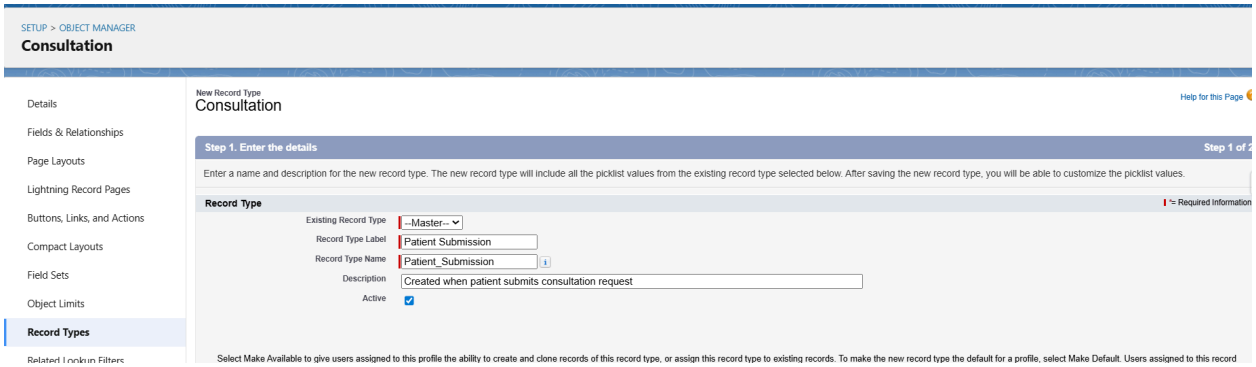
- **Patient__c → Consultation__c → Master-Detail**
- **Doctor__c → Consultation__c → Lookup**
- **Consultation__c → ConsultationNote__c → Master-Detail**

Step 3: Record Types

Record Types (Consultation__c)

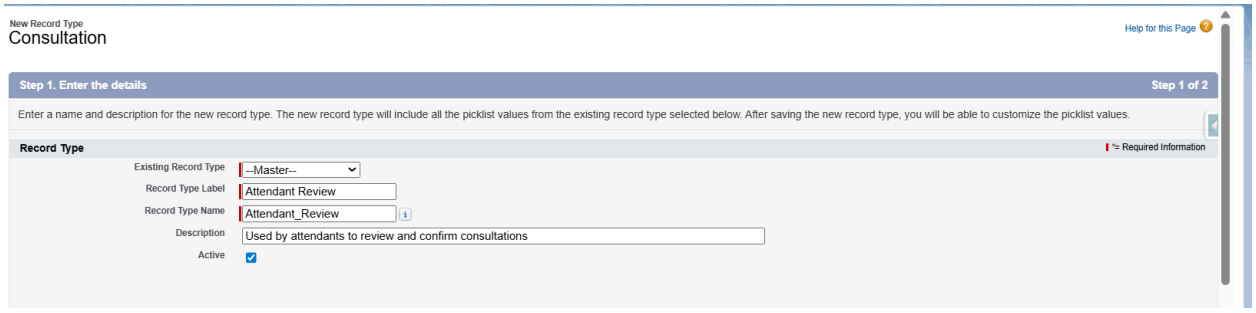
1. **Patient Submission**
    - **Record Type Label: Patient Submission**

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

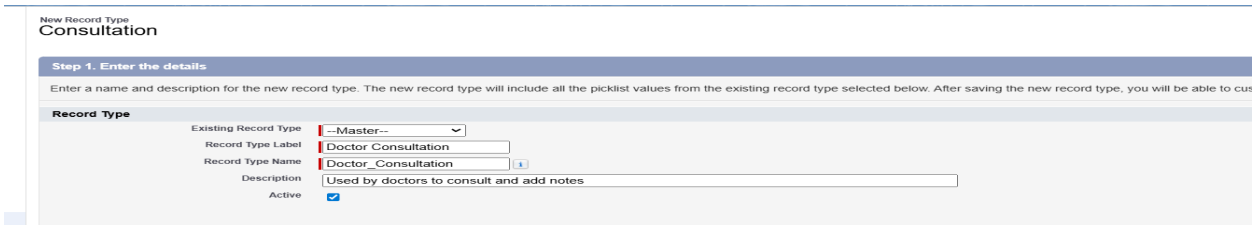


2. Attendant Review

- **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)**



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)**



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

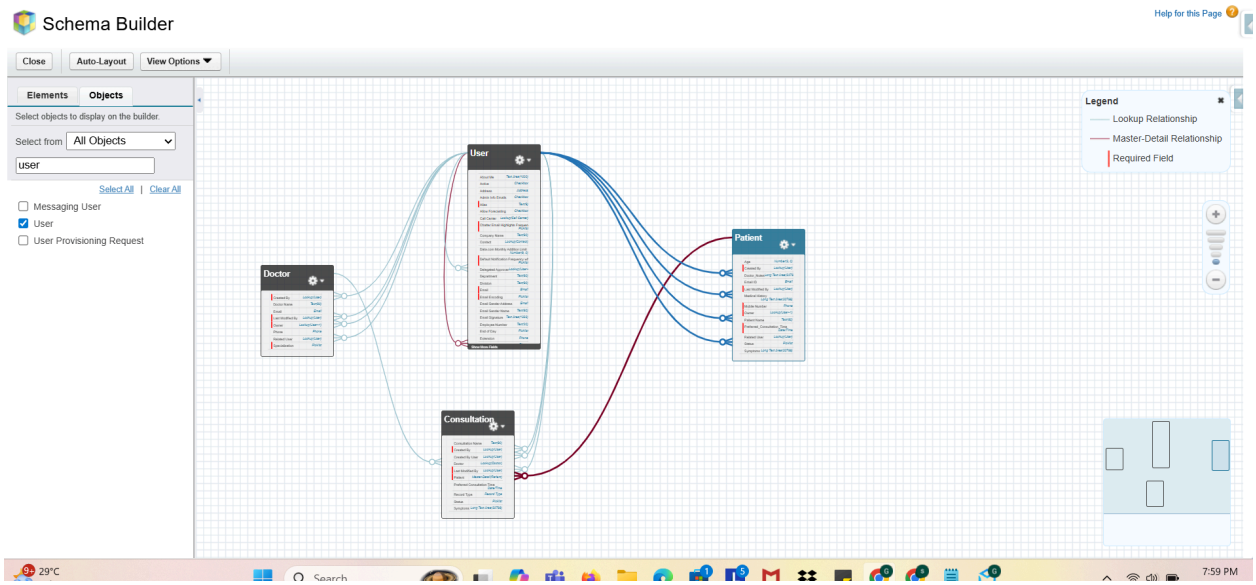
- **Sections:**
  - **Note Details: Consultation__c, Notes__c, Prescription__c**

🔹 **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 :



Lookup vs Master-Detail vs Hierarchical Relationships

1. Patient__c → User (Related_User__c)

- **Type: Lookup Relationship**
- **Purpose: Associates a patient record with the Salesforce user who created it.**
- **Behavior:**
  - **Each patient is linked to one user.**
  - **The patient record can exist independently; deleting the user does not delete the patient record.**

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

```
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);
    }
}

```

File Edit Debug Test Workspace Help < >

PatientController.apxc | PatientTrigger.apxt | PatientTriggerHandler.apxc | PatientTriggerHandlerTest.apxc

Code Coverage: All Tests 100% API Version: 64

```
public with sharing class PatientTriggerHandler {
    public static void sendStatusEmails(List<Patient__c> patients) {
        List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();

        for (Patient__c p : patients) {
            if (String.isBlank(p.Email_ID__c)) continue; // skip if no email

            String subject = '';
            String body = '';

            if (p.Status__c == 'Confirmed') {
                subject = 'Consultation Confirmed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'Your consultation scheduled at ' +
                       String.valueOf(p.Preferred_Consultation_Time__c) +
                       ' is confirmed. Please be ready on time.\n\n' +
                       'Wishing you the best for your consultation!\n\nThank you!';
            }
            else if (p.Status__c == 'Completed') {
                subject = 'Consultation Completed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'You have successfully completed your consultation with the doctor. ' +
                       'We hope the session was helpful for your recovery.\n\n' +
                       'Wishing you good health and a speedy recovery!\n\nThank you for trusting our service.';
            }
```

```
            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);
        }
    }
}
```

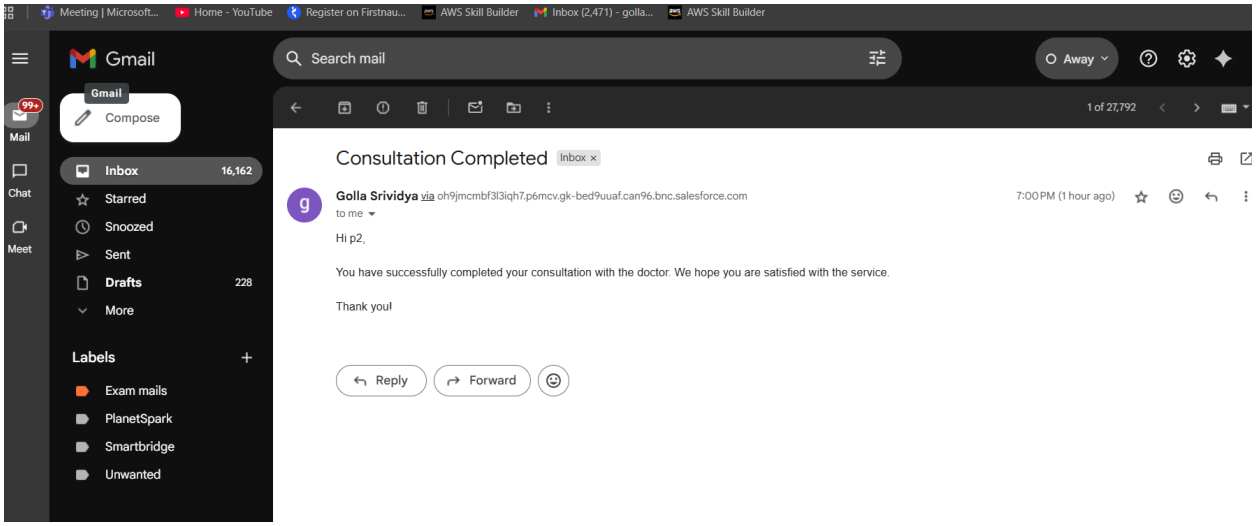
- **Field Updates / Workflow:** Managed via **user actions** in the UI (patients, doctors, attendants) instead of workflow rules or Process Builder.

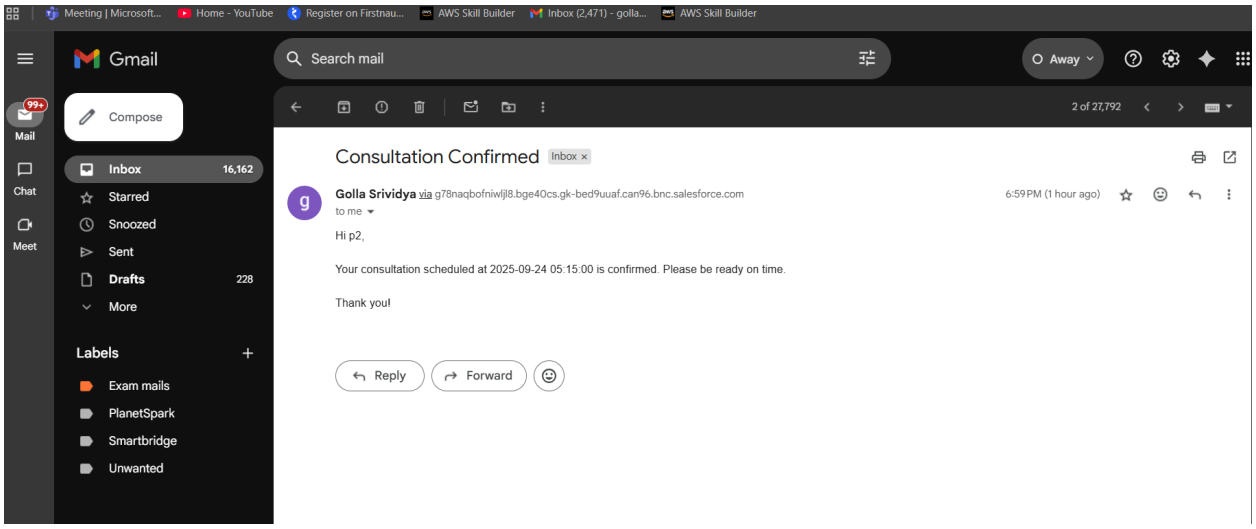
Not Used / Optional:

- **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

```
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);
    }
}
```

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

```java
public with sharing class PatientTriggerHandler {
    public static void sendStatusEmails(List<Patient__c> patients) {
        List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();

        for (Patient__c p : patients) {
            if (String.isBlank(p.Email_ID__c)) continue; // skip if no email

            String subject = '';
            String body = '';

            if (p.Status__c == 'Confirmed') {
                subject = 'Consultation Confirmed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'Your consultation scheduled at ' +
                       String.valueOf(p.Preferred_Consultation_Time__c) +
                       ' is confirmed. Please be ready on time.\n\n' +
                       'Wishing you the best for your consultation!\n\nThank you!';
            }
            else if (p.Status__c == 'Completed') {
                subject = 'Consultation Completed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'You have successfully completed your consultation with the doctor. ' +
                       'We hope the session was helpful for your recovery.\n\n' +
                       'Wishing you good health and a speedy recovery!\n\nThank you for trusting our service.';
            }
```

```
            }
            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);
        }
    }
}
```

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

```java
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,   // changed to Datetime (matches datetime-local input)
        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;
    }
}
```

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.

```
@isTest
public class PatientTriggerHandlerTest {
    @testSetup
    static void setupData() {
        // Create a test patient with email
        Patient__c p = new Patient__c(
            Name = 'Test Patient',
            Age__c = 30,
            Mobile_Number__c = '9876543210',
            Email_ID__c = 'test@example.com',
            Medical_History__c = 'No prior issues',
            Preferred_Consultation_Time__c = Date.today()
        );
        insert p;
    }

    @isTest
    static void testConfirmedEmail() {
        Test.startTest();
        Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
        patient.Status__c = 'Confirmed';
        update patient;
        Test.stopTest();

        // If no exception is thrown, the email logic ran successfully
        System.assert(true, 'Confirmed email sent successfully');
    }

    @isTest
    static void testCompletedEmail() {
        Test.startTest();
        Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
        patient.Status__c = 'Completed';
        update patient;
        Test.stopTest();

        System.assert(true, 'Completed email sent successfully');
    }

```

```
    }

    @isTest
    static void testCancelledEmail() {
        Test.startTest();
        Patient__c patient = [SELECT Id, Status__c FROM Patient__c LIMIT 1];
        patient.Status__c = 'Cancelled';
        update patient;
        Test.stopTest();

        System.assert(true, 'Cancelled email sent successfully');
    }
}
```

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.

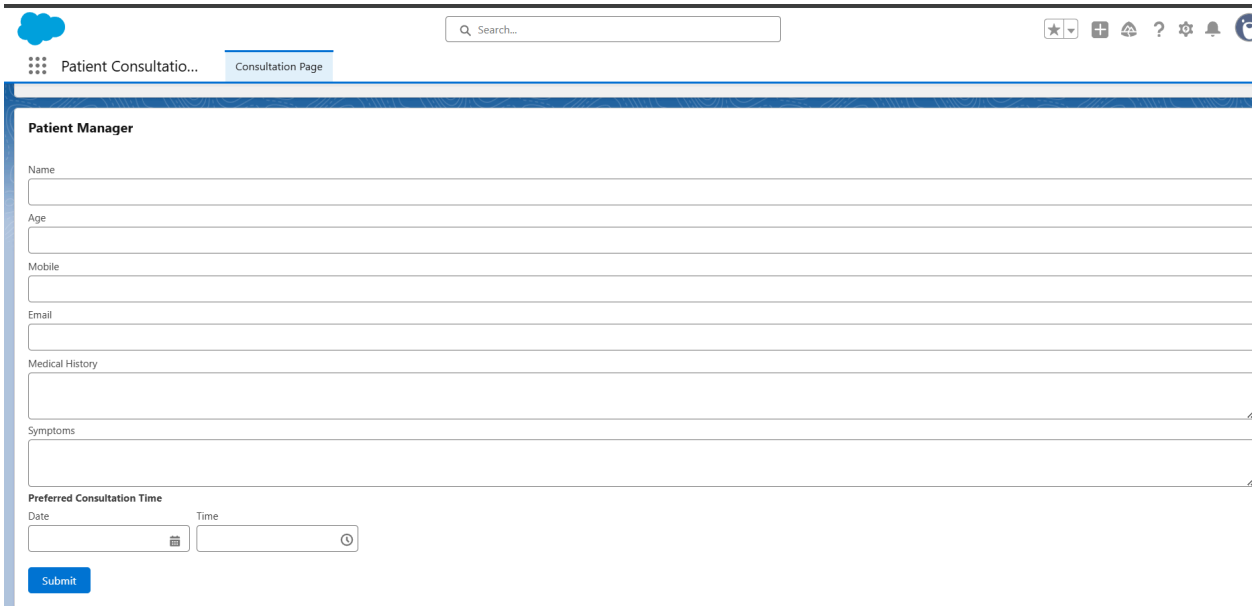
- **Wire Adapters:** Reactive fetching of records and user profiles.

- **Imperative Apex Calls:** For actions like creating patients, updating status, deleting records, and saving doctor notes.

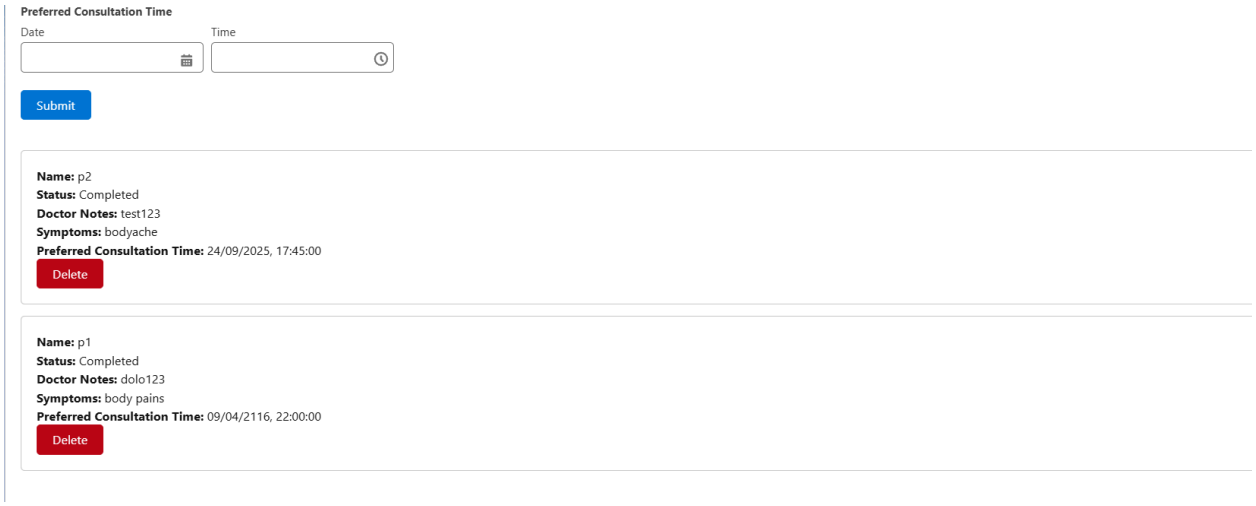
2. Project Relevance / Implementation

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.

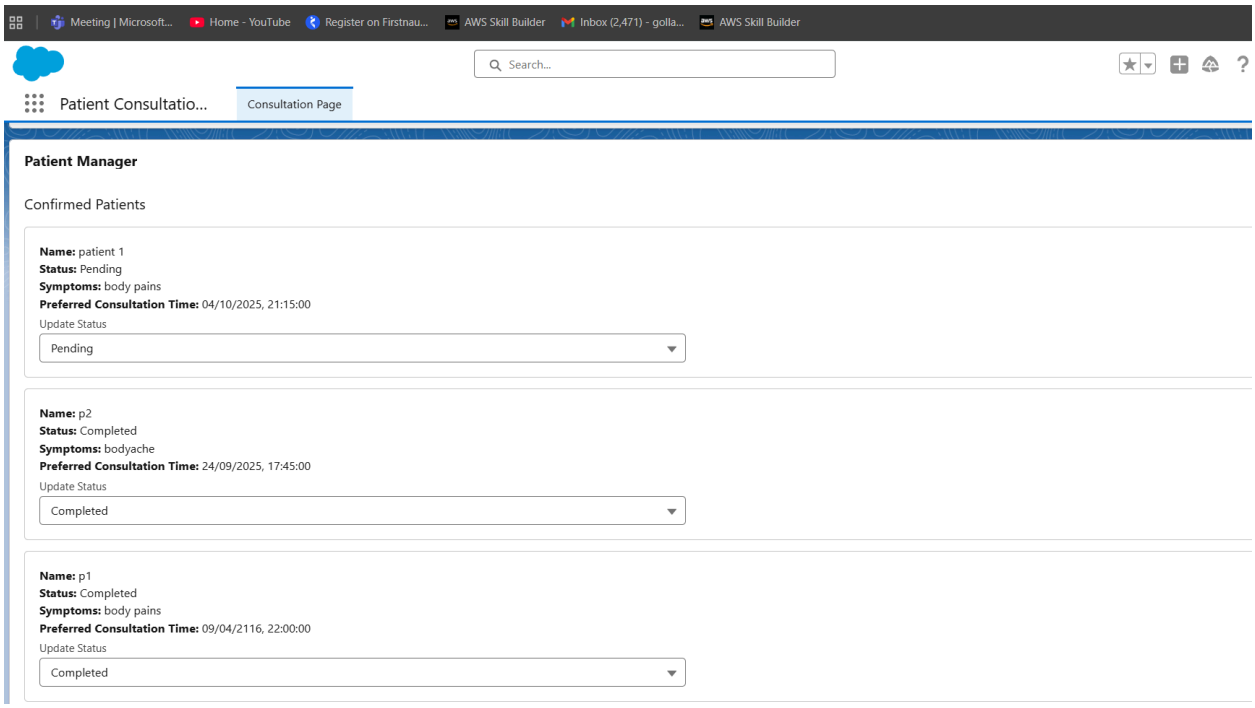


Patient history records:

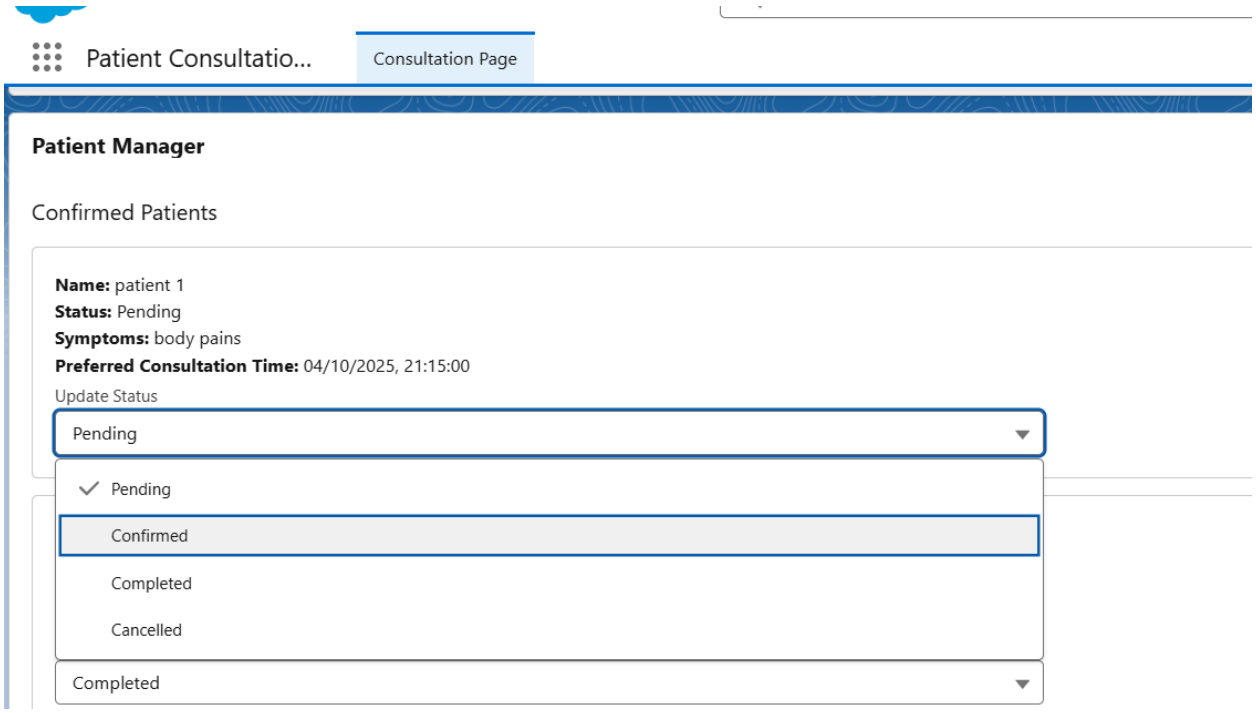


Doctor / Attendant UI

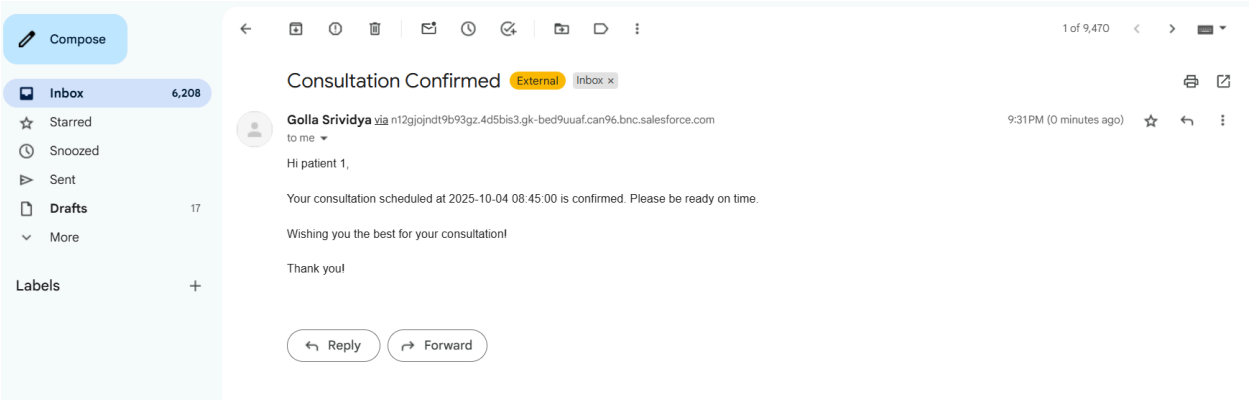
- Displays patient records to doctors and attendants.
- **Doctors:** Can add notes and update status.
- **Attendants:** Can update patient status.



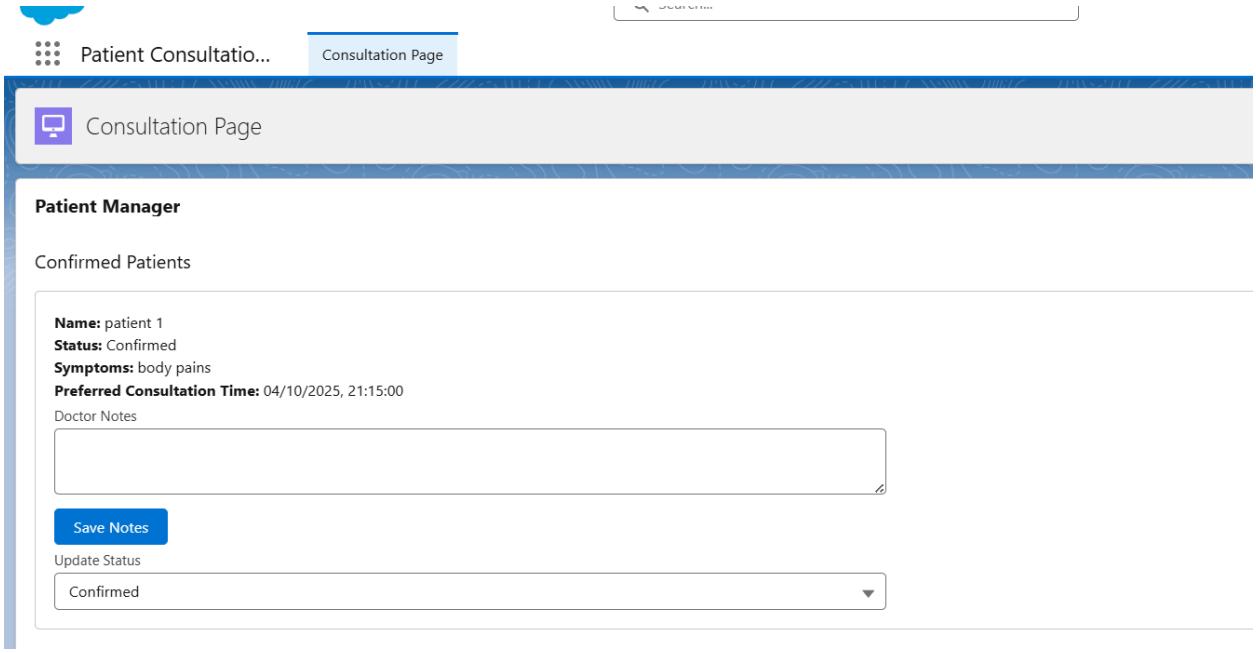
Attendant UI



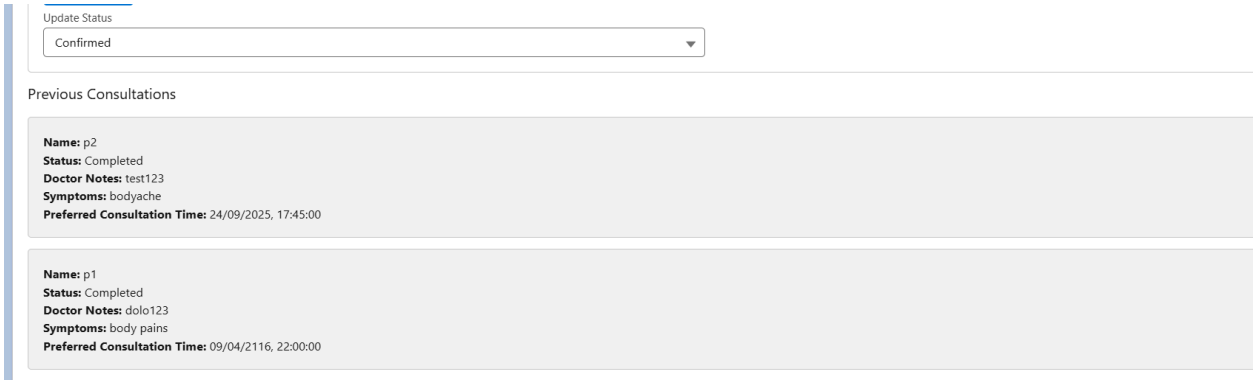
Email Triggered after the status is updated to confirmed



Doctor UI



Previous history:



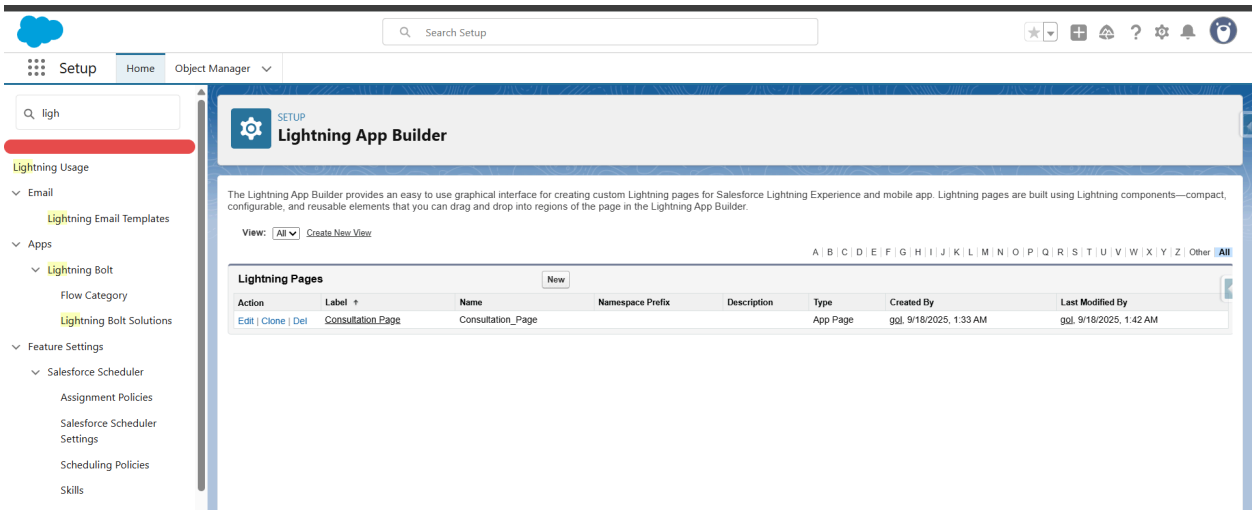
- Uses **wire adapters** for reactive record fetching and **imperative Apex** for record updates.

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>
```

```
                <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>
```

```
                <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>

                                <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>
                <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','pat
ientHistory','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 {
```

```
    public static void sendStatusEmails(List<Patient__c>
patients) {
        List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();

        for (Patient__c p : patients) {
            if (String.isBlank(p.Email_ID__c)) continue; // skip
if no email

            String subject = '';
            String body = '';

            if (p.Status__c == 'Confirmed') {
                subject = 'Consultation Confirmed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'Your consultation scheduled at ' +

String.valueOf(p.Preferred_Consultation_Time__c) +
                       ' is confirmed. Please be ready on
time.\n\n' +
                       'Wishing you the best for your
consultation!\n\nThank you!';
            }
            else if (p.Status__c == 'Completed') {
                subject = 'Consultation Completed';
                body = 'Hi ' + p.Name + ',\n\n' +
                       'You have successfully completed your
consultation with the doctor. ' +
                       'We hope the session was helpful for your
recovery.\n\n' +
                       'Wishing you good health and a speedy
recovery!\n\nThank you for trusting our service.';
            }
            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);
        }
    }
}
```