TITLE: CycleCare-A period tracker

Phase 1: Problem Understanding & Industry Analysis

1. Requirement Gathering

Purpose: Collect detailed requirements from potential users.

Steps:

1. Identify target users:
    - Patients (tracking cycle & appointments)
    - Gynecologists/doctors (appointment management, patient history)
    - Admins (monitoring overall app usage)
2. Gather functional needs:
    - Track cycle dates (start, end, irregularities)
    - Predict ovulation & fertile windows
    - Book doctor appointments & send reminders
    - Locate gynecologists nearby (integration with maps/doctor APIs)
    - Notifications for irregular cycles → suggest medical visits
3. Gather non-functional needs:
    - Mobile-first design
    - Secure handling of sensitive medical data
    - Simple and intuitive UI
    - Scalable for large user base

2. Stakeholder Analysis

Purpose: Understand **who benefits from the app** and what they need.

Stakeholder	Needs	Pain Points
Patients	Cycle tracking, reminders, ovulation prediction	Forgetting cycles, lack of medical awareness
Doctors	Appointment scheduling, patient history access	Time management, lack of patient info
Admins	Manage app usage, reports, compliance	Ensuring data security & accurate insights

3. Business Process Mapping

Purpose: Visualize workflows inside the app.

Example Processes for Cycle Care:

- **Cycle Tracking Flow:**
  User logs cycle → System predicts ovulation → Sends reminders → Flags irregular cycles
- **Appointment Booking Flow:**
  User searches doctor → Finds nearby gynecologist → Books slot → Doctor confirms → Reminder sent
- **Doctor Interaction Flow:**
  Doctor logs in → Views appointments → Accesses patient cycle data → Suggests treatment

(Best represented in a flowchart or BPMN diagram – I can create one if you'd like 👀).

4. Industry -Specific Use Case Analysis

Purpose: See what's already in the **FemTech (female health tech)** industry and tailor accordingly.

Common Features in Similar Apps:

- **Period Tracking Apps (Clue, Flo, Ovia):** AI-based predictions, symptom logging
- **Doctor Appointment Apps (Practo, Zocdoc):** Location-based search, teleconsultations
- **Wellness Apps (Fitbit, Apple Health):** Integration with wearables, lifestyle tracking

Insights for Cycle Care:

- Start simple (cycle tracking + doctor appointments).
- Later enhancements → wearable integration, AI predictions, telemedicine.

5. AppExchange Exploration

Purpose: Research what Salesforce already offers.

Steps:

1. Go to **AppExchange** → Search terms: *"Healthcare", "Patient Management", "Appointments", "FemTech"*
2. Analyze existing apps:
   - Health Cloud (by Salesforce) → for patient records & care plans
   - Doctor Appointment Scheduler apps
   - Maps & Location services
3. Learn from them:
   - What features do they already provide?
   - What gaps can Cycle Care fill? (e.g., personalized female health focus)

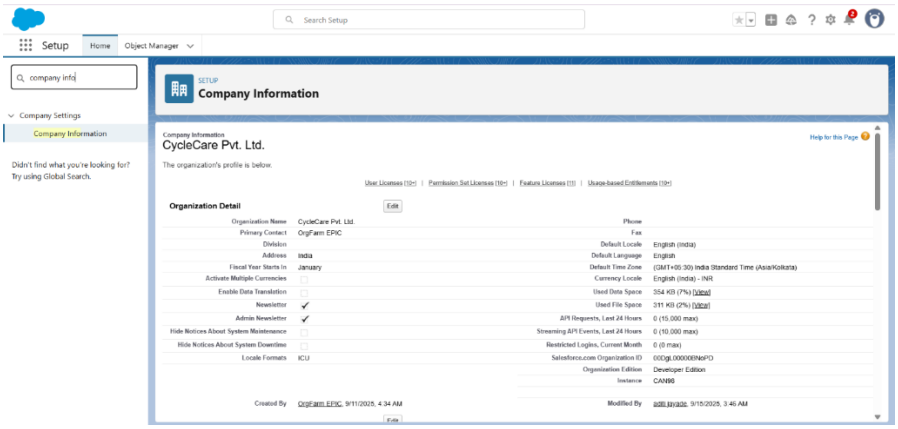
Phase 2: Org Setup & Configuration

1. Salesforce Edition

- Used **Developer Edition**: free, full features, can test apps before bigger editions.

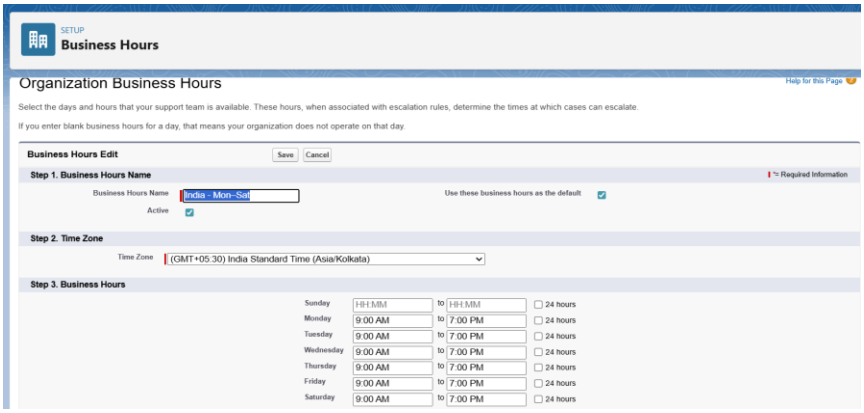
2. Company Profile

- **Settings:** India, currency INR, English.
- **Why:** Correct money calculations, accurate dates, compliance.

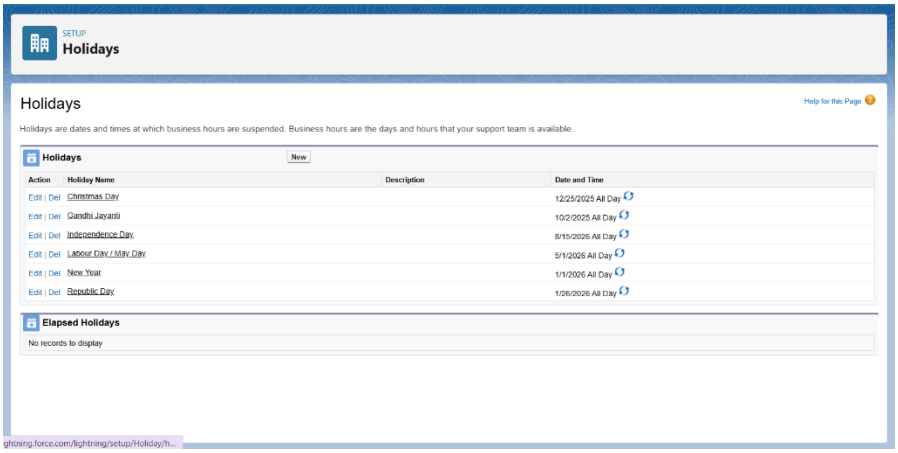


3. Business Hours & Holidays

- **Work hours:** Mon–Sat, 9 AM – 7 PM



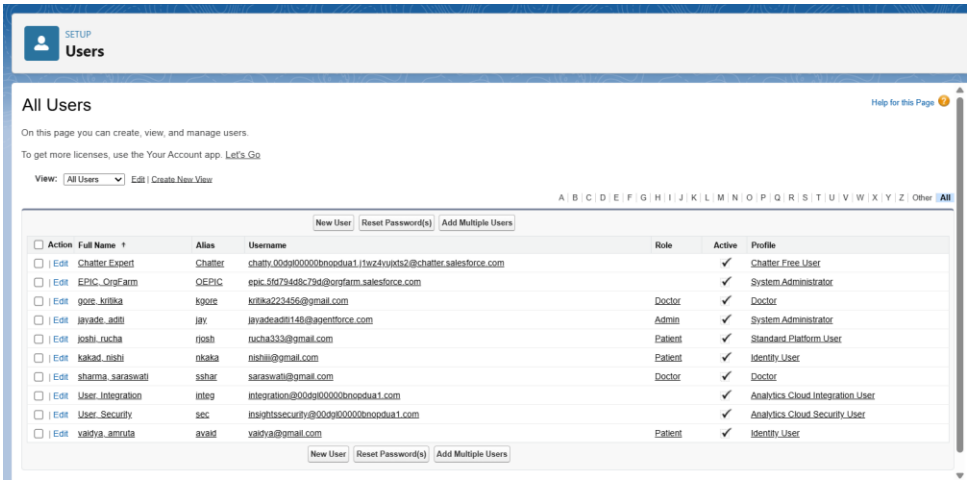
- **Holidays:** National holidays.



- **Why:** Service booking, case escalation, and SLAs depend on working hours.

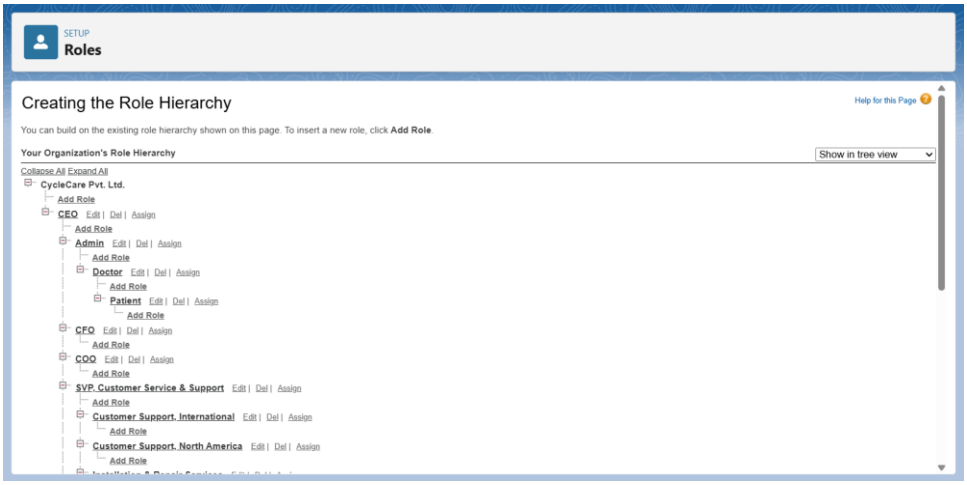
4. User Setup

- Users: Admin,Doctor,Patient.
- Assign proper licenses for access.



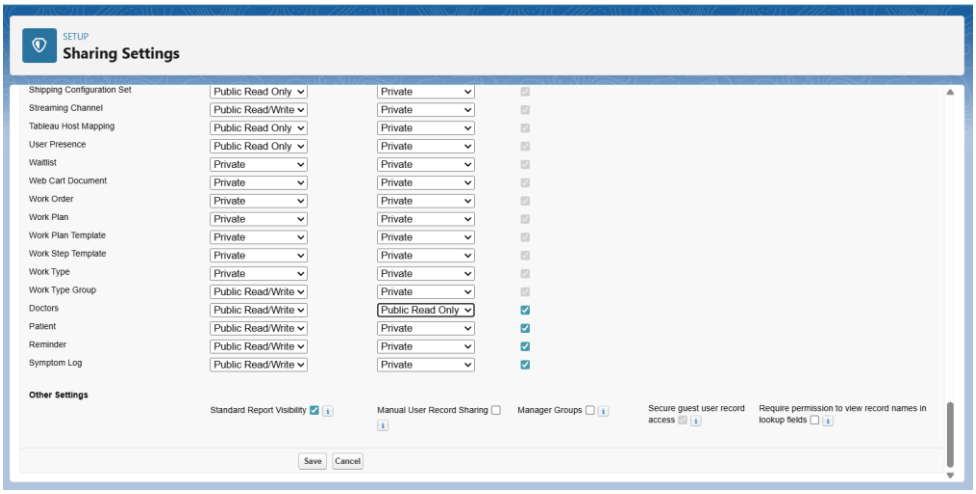
5. Profiles & Roles

- **Profiles:** Control what each user can do.
  - I have basically made three profiles and they are: Admin ,Doctor and Patient
- **Roles:** Show hierarchy (Admin → Doctor →Patient)



6. OWD & Sharing Rules

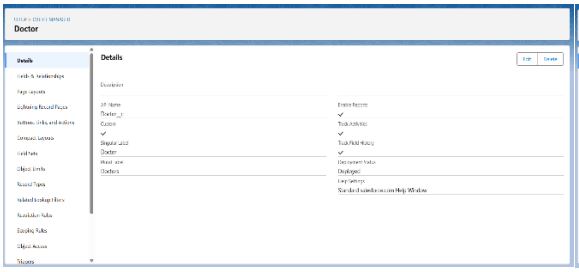
- **OWD:** Private for service records.
- **Sharing:** Doctor sees all, customer sees own records.

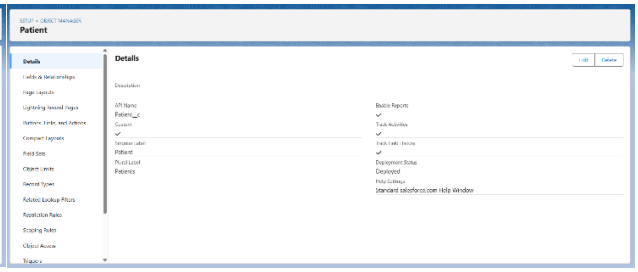


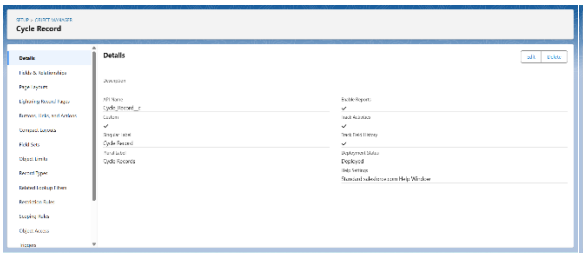
Phase 3: Data Modeling & Relationships

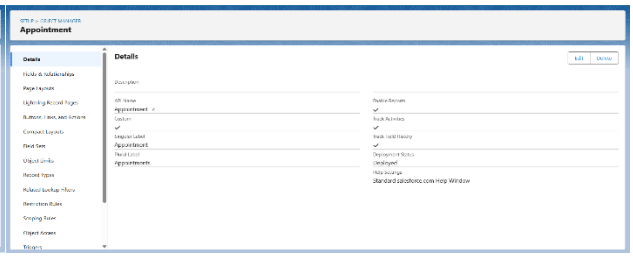
1. Custom Objects

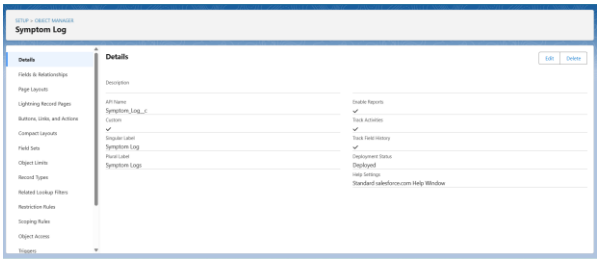
- Doctor, Patient, Appointment, Cycle Record, Symptom logs





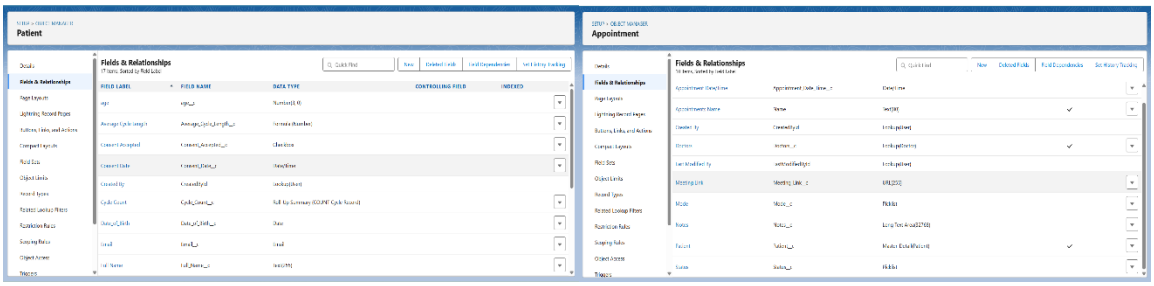


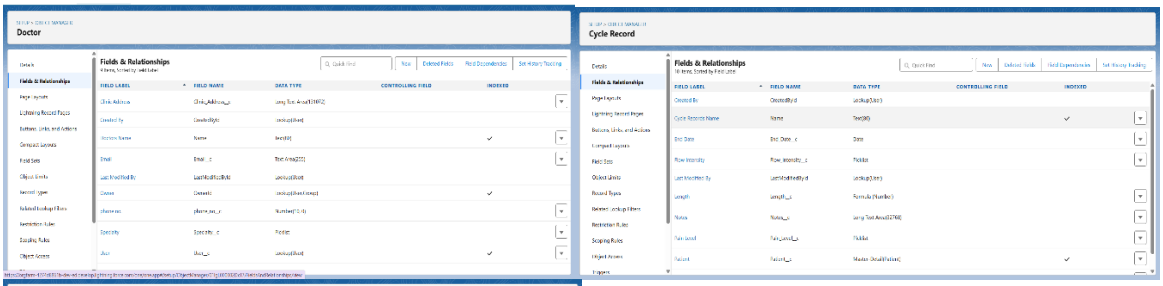


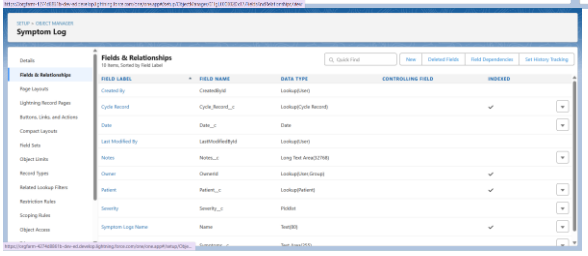


2. Fields

- **Doctor:** Clinic Address, Doctors Name, Email, Number of Patients, phone no, Specialty, User.
- **Patient:** age,avg cycle length,consent accepted, Concent date, Cycle count, DOB, Email, Full Name, Gender, Medical Notes, Paitents ID, Phone, Primary Doctor, Total Cycle Days.
- **Appointment:** Appointment Date/Time, Appointments Name, Doctors, Meeting Link, Mode, Notes, Patient, Status.
- **Cycle_record:** Cycle Records Name, End Date, Flow Intensity, Length, Notes, Pain Level, Paitent, Start Date.
- **Symptom logs:** Cycle Record, Date, Notes, Paitent, Severity, Symptom Logs Name, Symptoms.

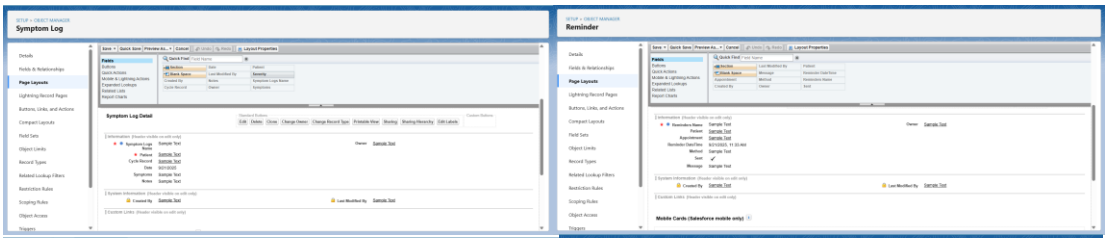


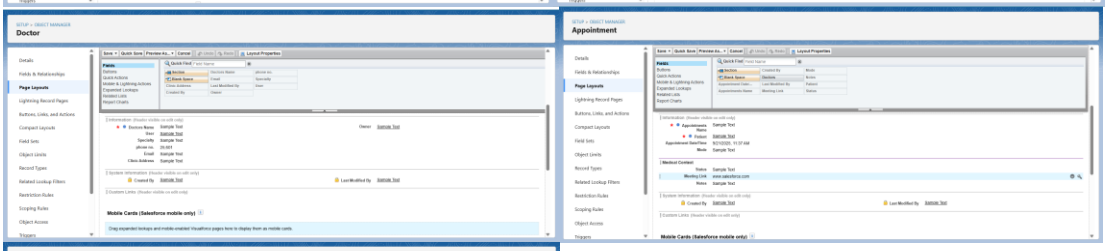


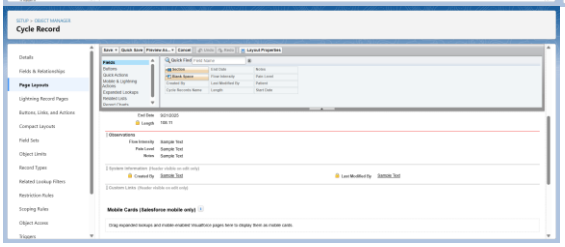


3. Page Layouts & Compact Layouts

- Arrangement of fields, sections, related lists.

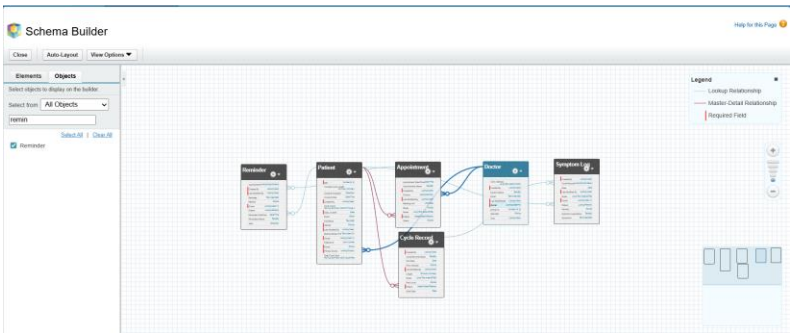






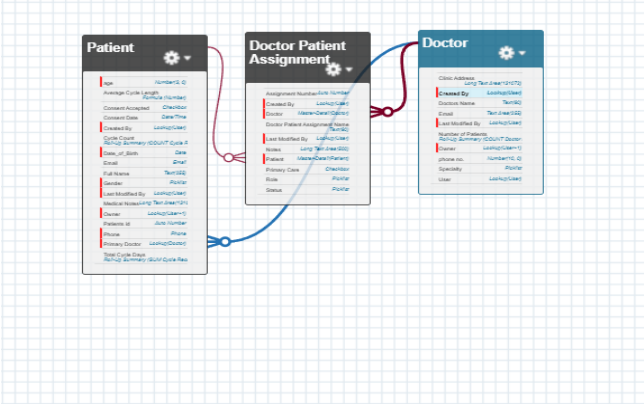
4. Schema Builder (ERD)

- Visual representation of objects and their relationships.
    - Appointment → Lookup → Doctor
    - Appointment → MDR→ Patient
    - Patient → Lookup → Doctor
    - Reminder→ Lookup → Patient
    - Reminder→ Lookup → Appointment
    - Cycle_Record → MDR→ Patient
    - Symptom Log→ Lookup → Patient
    - Symptom Log→ Lookup → Cycle_Record
- **Why:** Proper relationships help reporting and automation.



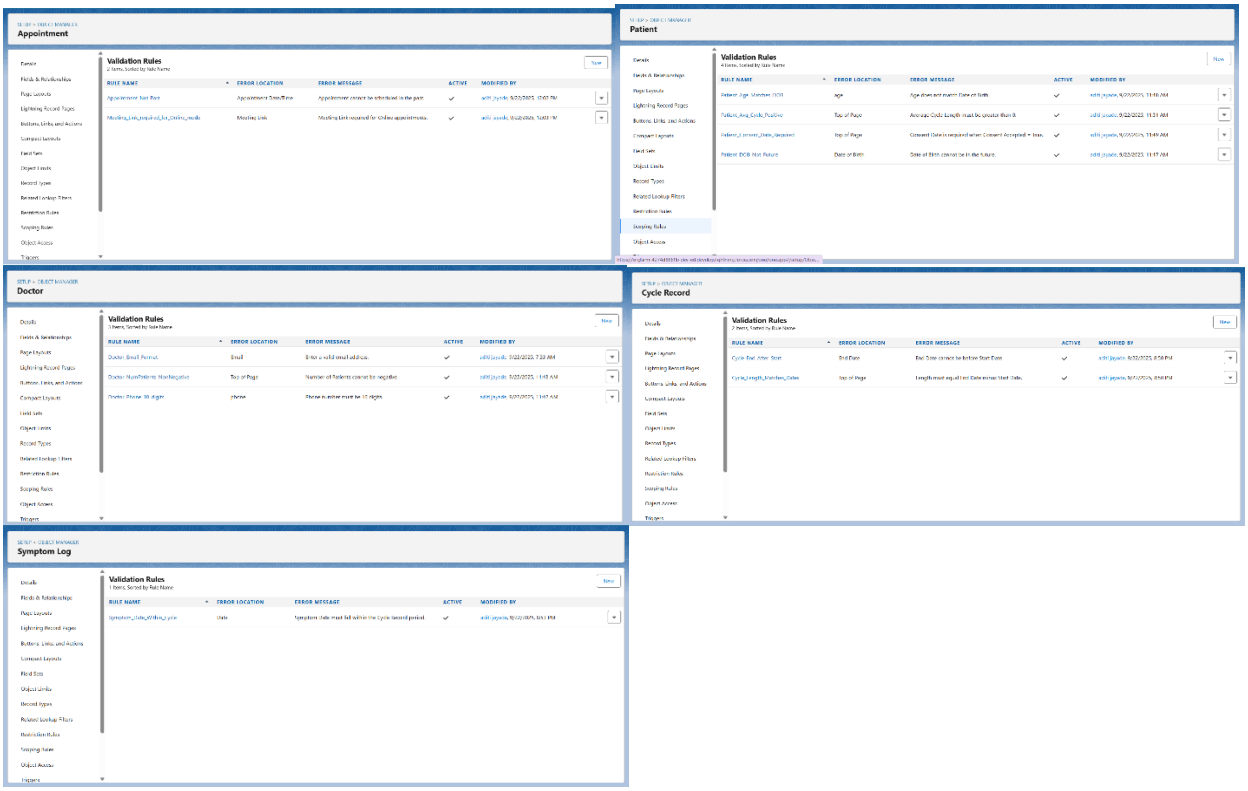
5.Junction Objects

- Handle **many-to-many relationships**.
    - Doctor Patient Assignment→ MDR→ Doctor
    - Doctor Patient Assignment→ MDR→ Patient



Phase 4: Process Automation (Admin)

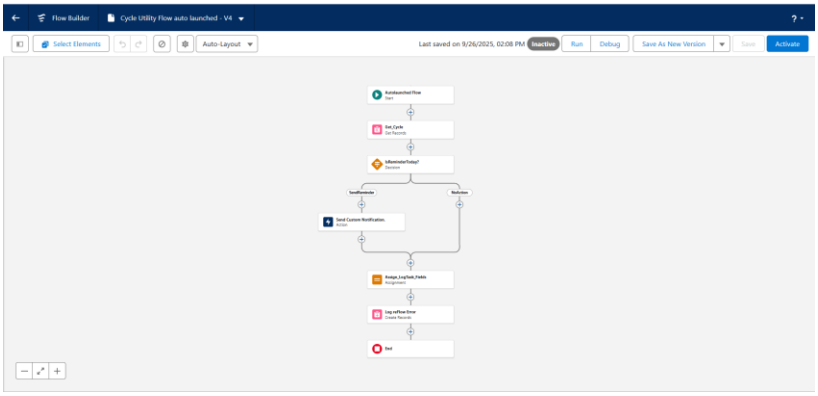
1. Validation Rules



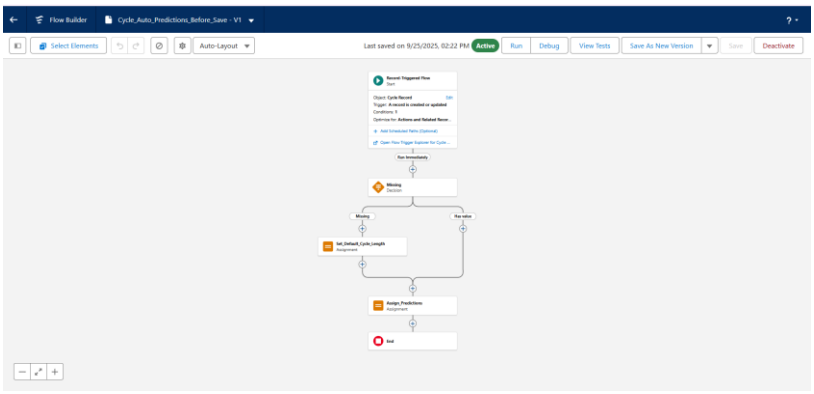
2. Workflow Rules

- AutolaunchedSubflow — Cycle Notifications

Help for this Page



- Cycle_Auto_Predictions_Before_Save



- Automate simple tasks like sending reminders.

custom notification

Phase 5: Apex Programming

1. Apex Triggers (after insert / after update)

```
trigger MenstrualCycleTrigger on Menstrual_Cycle__c (after insert, after update) {
    if (Trigger.isAfter && (Trigger.isInsert || Trigger.isUpdate)) {
        MenstrualCycleTriggerHandler.afterUpsert(Trigger.new);
    }
}
```

2. Trigger Design Pattern — handler class

```
public class MenstrualCycleTriggerHandler {
    public static void afterUpsert(List<Menstrual_Cycle__c> newList) {
        Set<Id> contactIds = new Set<Id>();
        for (Menstrual_Cycle__c m : newList) {
            if (m.Contact__c != null) contactIds.add(m.Contact__c);
        }
        if (contactIds.isEmpty()) return;

        // enqueue queueable job for async prediction (bulk safe)
        System.enqueueJob(new CyclePredictionQueueable(contactIds));
    }
}
```

3. Queueable Apex

```
public class CyclePredictionQueueable implements Queueable {
    private Set<Id> contactIds;
```

```
    public CyclePredictionQueueable(Set<Id> contactIds) {
        this.contactIds = contactIds;
    }
    public void execute(QueueableContext ctx) {
        CyclePredictor.predictForUsers(contactIds);
    }
}
```

4.Future Methods

```
public class NotificationService {
    @future
    public static void sendReminder(List<Id> contactIds) {
        // placeholder for notification logic (callout not allowed unless future(callout=true))
    }
}
```

5.Exception Handling

```
try {
    update updates;
} catch (Exception e) {
    System.debug('CyclePredictor.update error: ' + e.getMessage());
}
```

6.Test Classes

```
@isTest
private class CycleCareTests {
    @isTest static void testTriggerAndPredictor() {
```

```
// prepare parent data
Account acc = new Account(Name='Test Acc');
insert acc;
Contact c = new Contact(FirstName='T', LastName='User', AccountId = acc.Id);
insert c;

// create 4 cycles spaced 28 days apart (most recent equals today)
Date baseDate = Date.today().addDays(-84); // 12 weeks ago
List<Menstrual_Cycle__c> cycles = new List<Menstrual_Cycle__c>();
for (Integer i = 0; i < 4; i++) {
    cycles.add(new Menstrual_Cycle__c(
        Contact__c = c.Id,
        Start_Date__c = baseDate.addDays(28 * i)
    ));
}

Test.startTest();
    insert cycles; // trigger will enqueue Queueable; stopTest runs async jobs
Test.stopTest();

// query most recent cycle and assert predictions
Menstrual_Cycle__c recent = [SELECT Id, Start_Date__c, Predicted_Next_Period__c,
                                    Predicted_Ovulation__c, Fertile_Window_Start__c
                             FROM Menstrual_Cycle__c
```

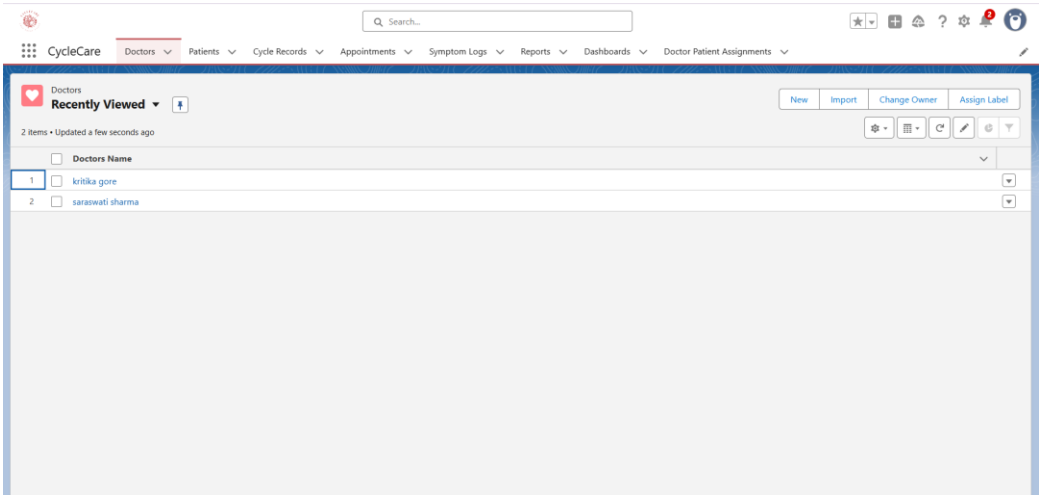
```
                    WHERE Contact__c = :c.Id
                    ORDER BY Start_Date__c DESC
                    LIMIT 1];

        Date expectedNext = recent.Start_Date__c.addDays(28); // expected avg cycle 28
        System.assertEquals(expectedNext, recent.Predicted_Next_Period__c);
        System.assertEquals(expectedNext.addDays(-14), recent.Predicted_Ovulation__c);
        System.assertEquals(expectedNext.addDays(-14).addDays(-5), recent.Fertile_Window_Start__c);
    }
}
```

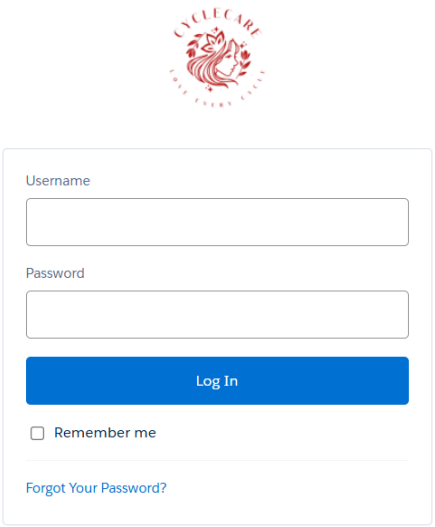
Phase 6: User Interface Development

1. Lightning App Builder

- Assembles app pages with standard and custom components.
- Provide structured navigation.



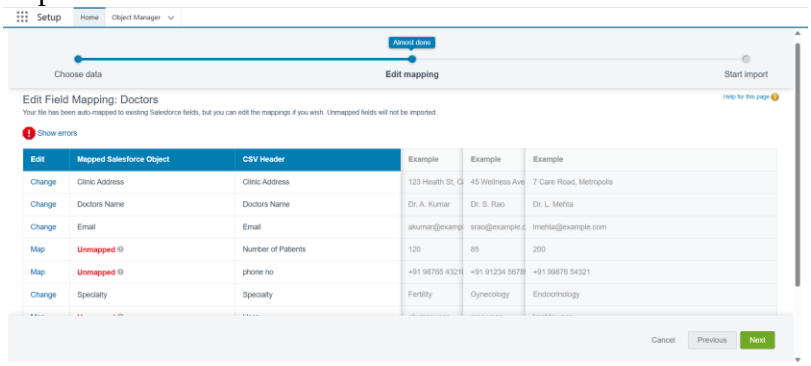
- App login page:

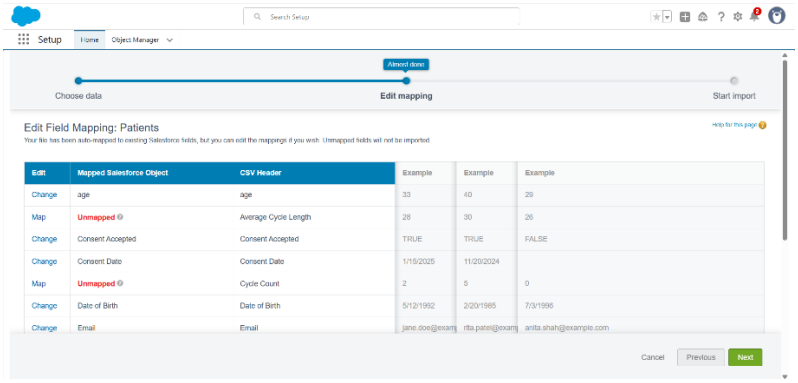


Phase 7: Data Management & Deployment

1. Data Import Wizard

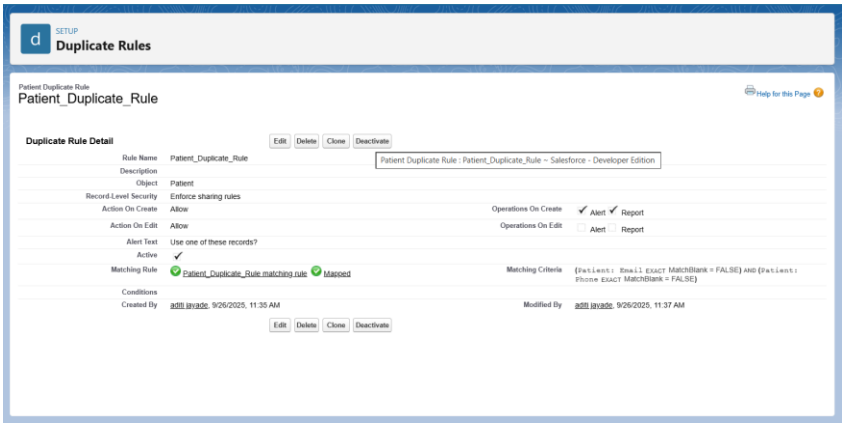
- Import data of Doctor and Patient

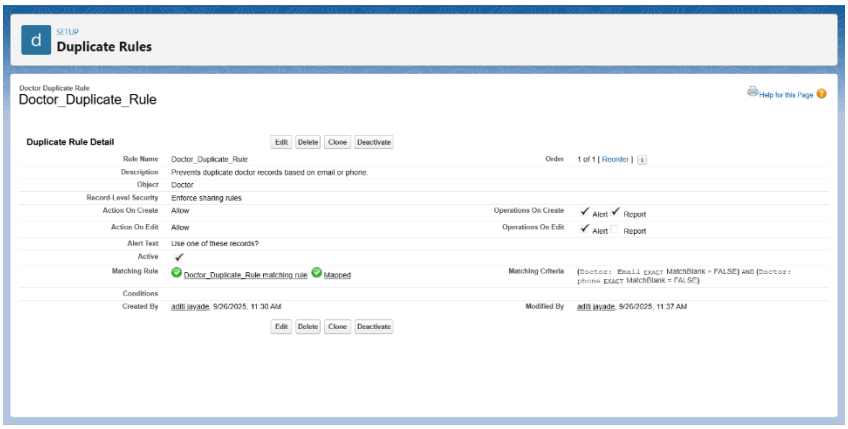




2. Duplicate Rules

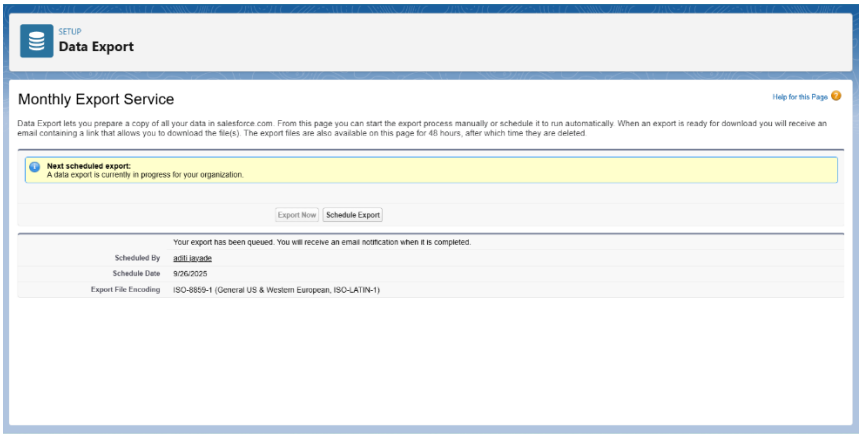
- Prevents duplicate patient or doctor records.
- Doctor and patient duplicate rules





3. Data Export & Backup

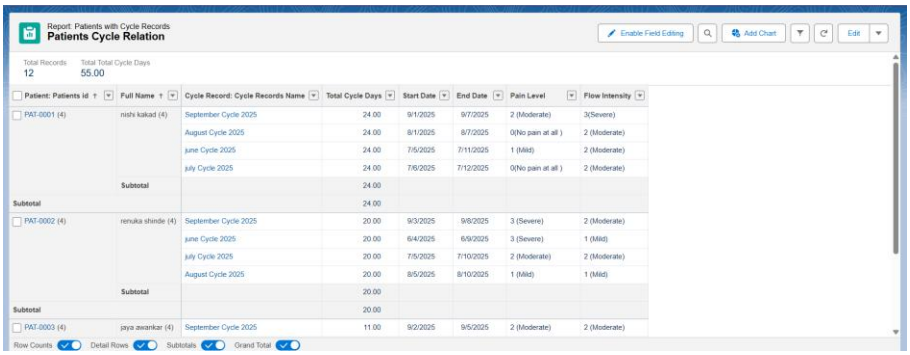
- Safeguards against data loss.

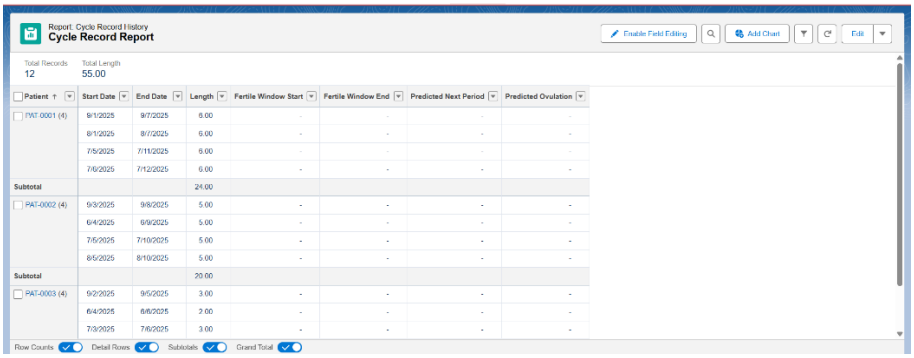


Phase 8: Reporting & Dashboards

1. Reports

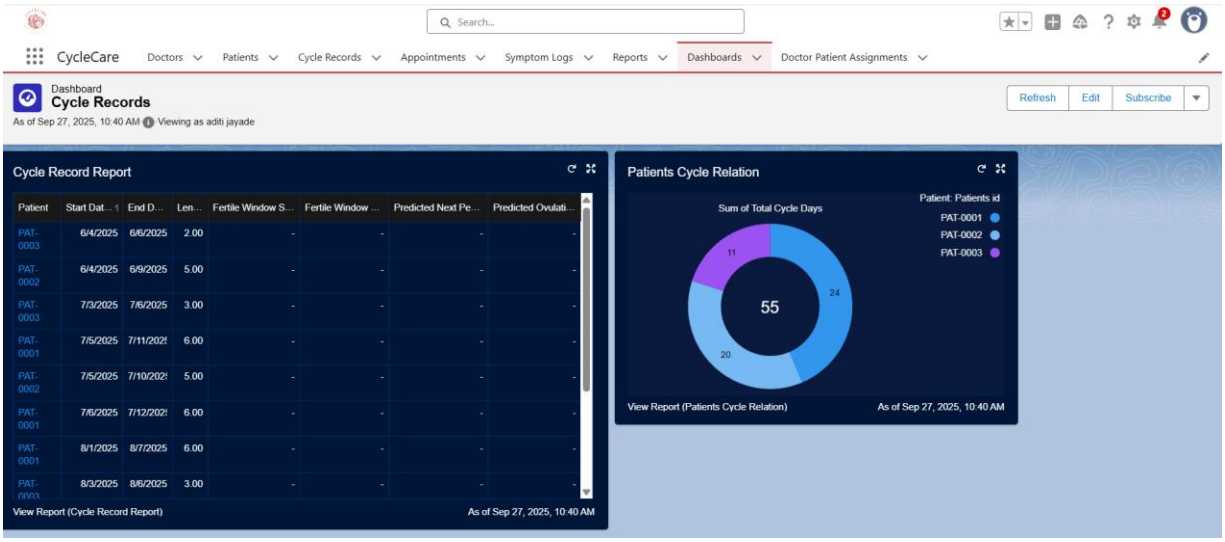
- Generate tabular, summary, matrix, and joined reports for tracking cycles, appointments, and irregularities.





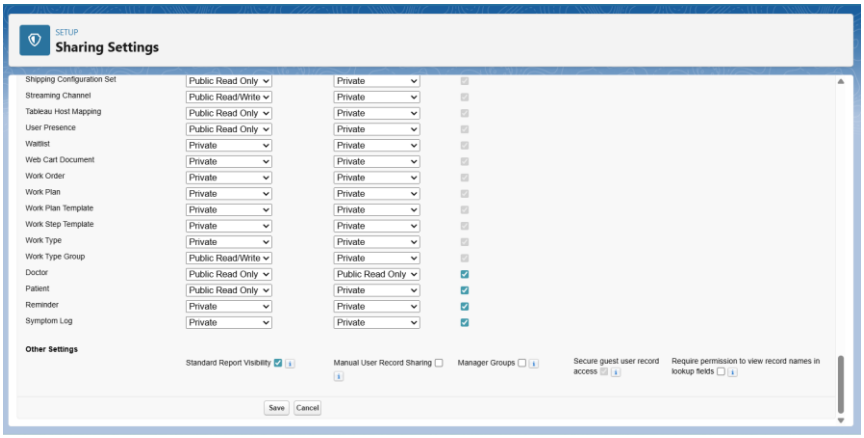
2. Dashboards

- Visualize key health metrics personalized for users and admins.



3. Sharing Settings

- . Define record-level access.



4. Field Level Security

- Protect sensitive medical details.

5. Session Settings & Login IP Ranges

- Strengthen login security.
