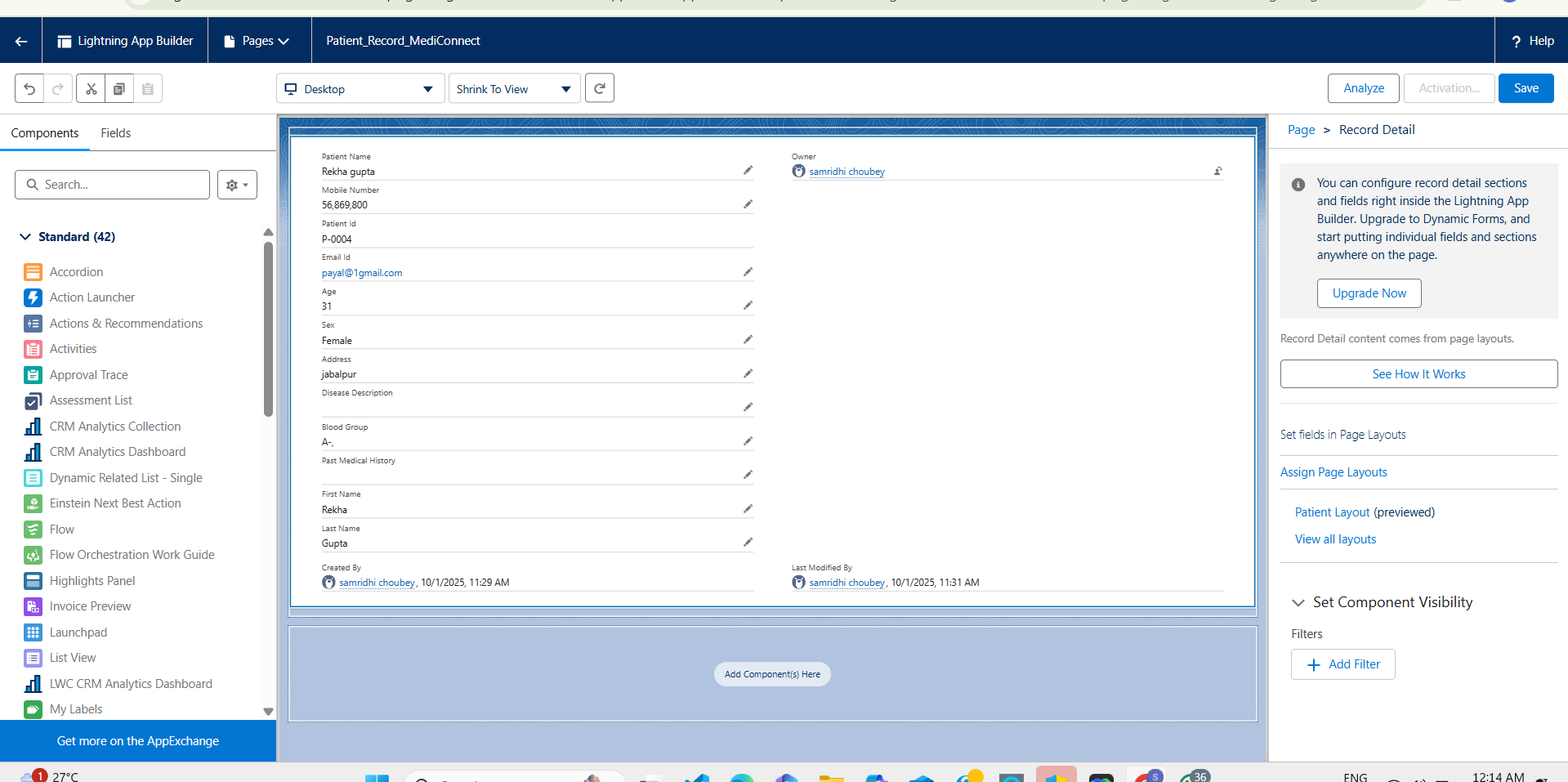
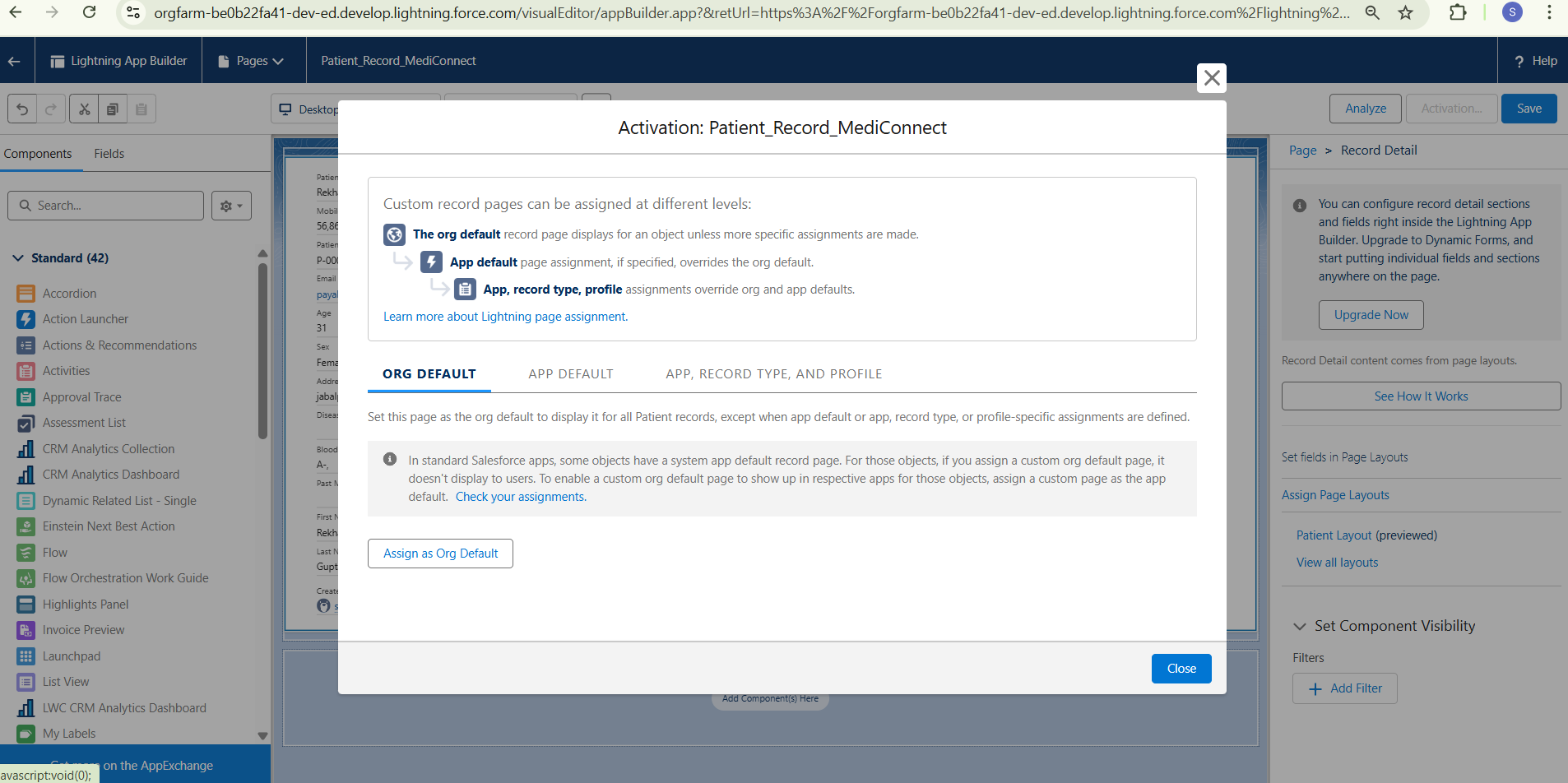
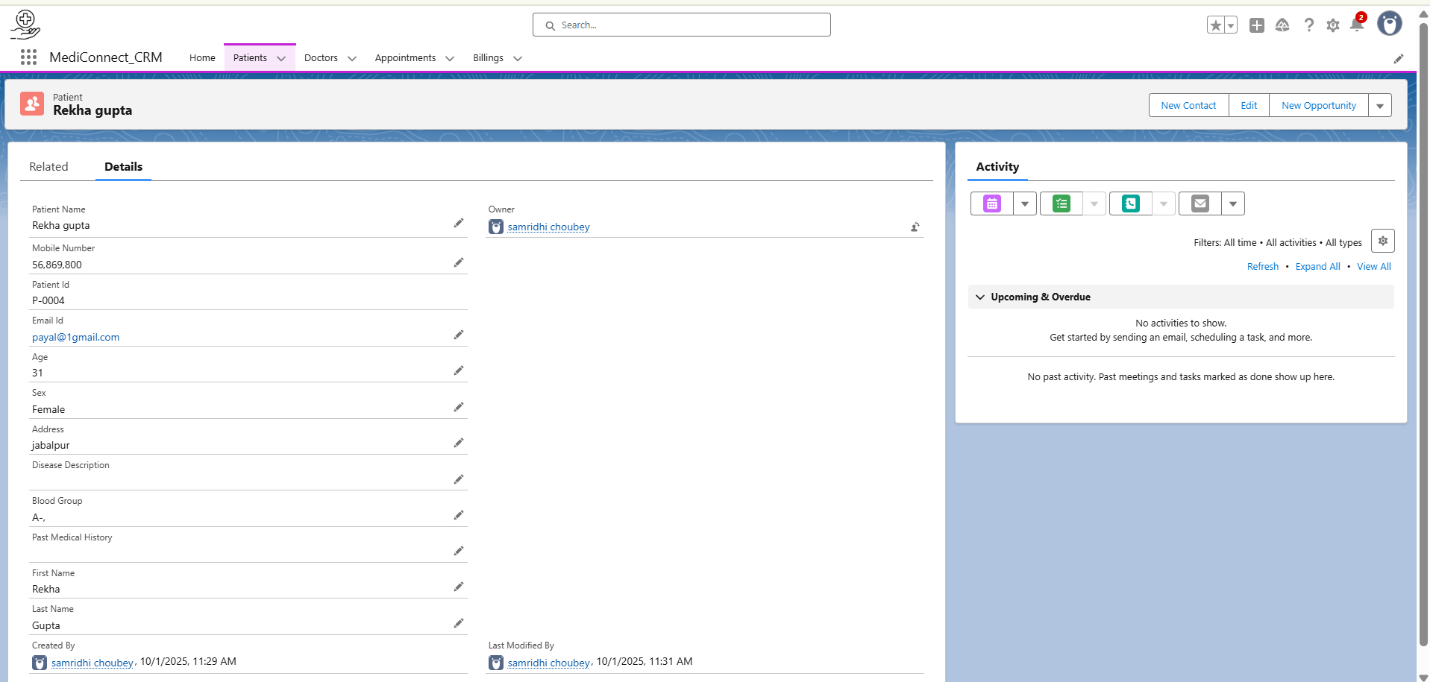
**SPhase 6: User Interface Development**

**1.Lightning App Builder  and Record Pages:**

* We used **Lightning App Builder** to design a customized user interface for the MediConnect application.
* A **Record Page** was created for the Patient\_\_c object using the **Header + One Region** template.
* In the **Header**, we added a **Highlights Panel** to display key patient details like Patient Name, Age, Mobile Number, and Blood Group.
* In the **Main Region**, we placed the **Record Details** component to show complete patient information, controlled through the Page Layout of Patient\_\_c.
* We also included **Related Lists** (like Appointments and Prescriptions) to provide quick access to associated records.
* The customized page was then **activated** for the **MediConnect App**, so that users see this layout only inside the MediConnect application.
* This ensures that healthcare staff can quickly view and manage patient data with a clear and organized interface.







**2.Tabs:**

Tabs allow you to organize different objects or pages inside the MediConnect App, so users can quickly navigate between Patients, Appointments, Doctors, Reports, etc.

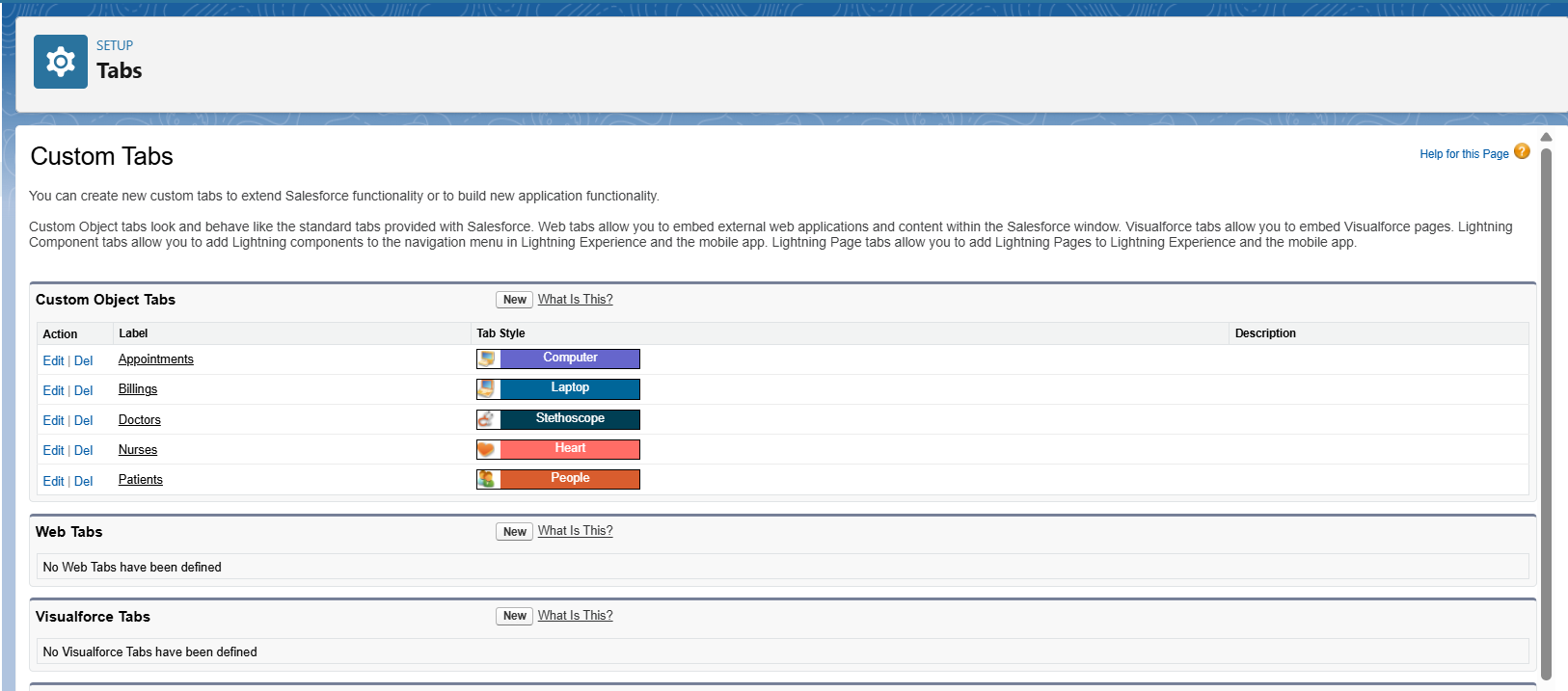
**Tabs in MediConnect:**

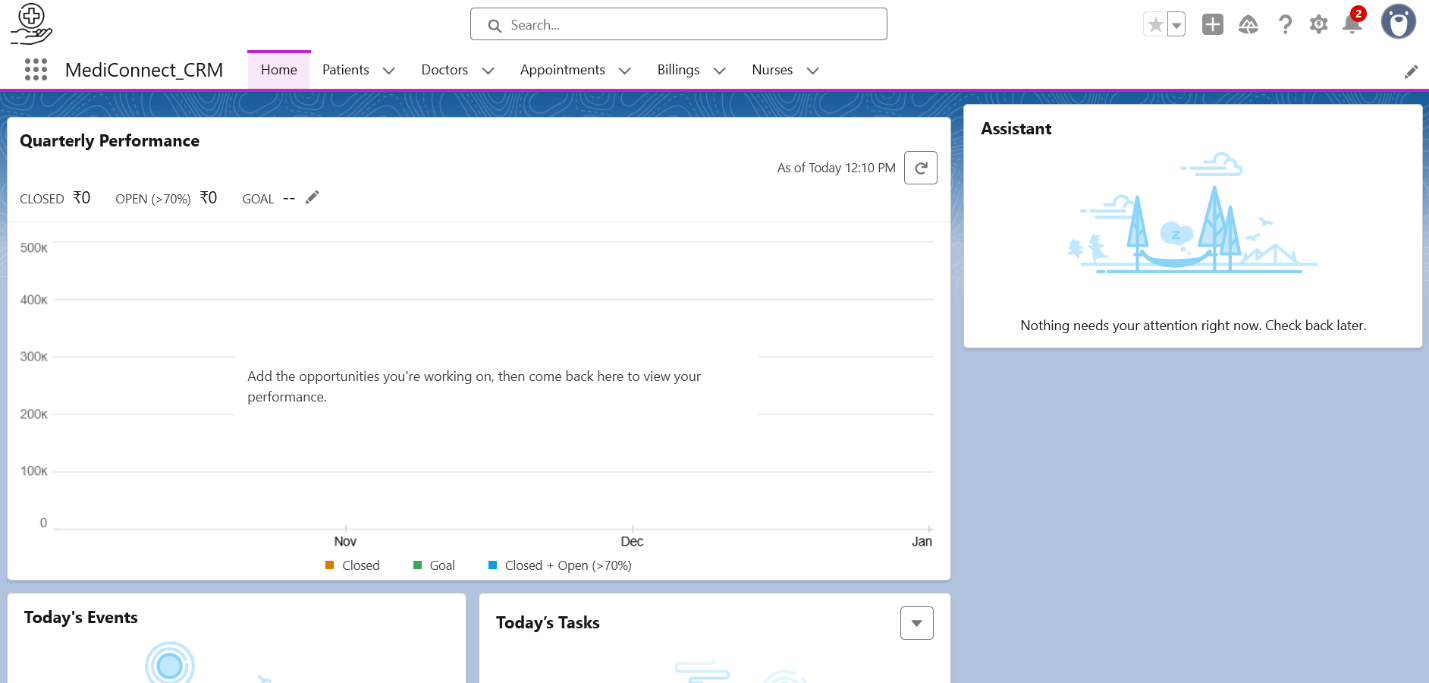
1. **Standard Tab :**

* Home

1. **Custom Tabs :**

* Doctors
* Patients
* Appointments
* Billings
* Nurse



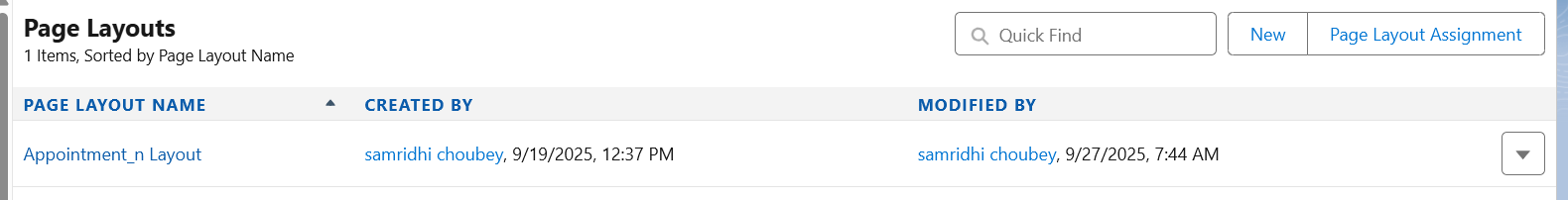


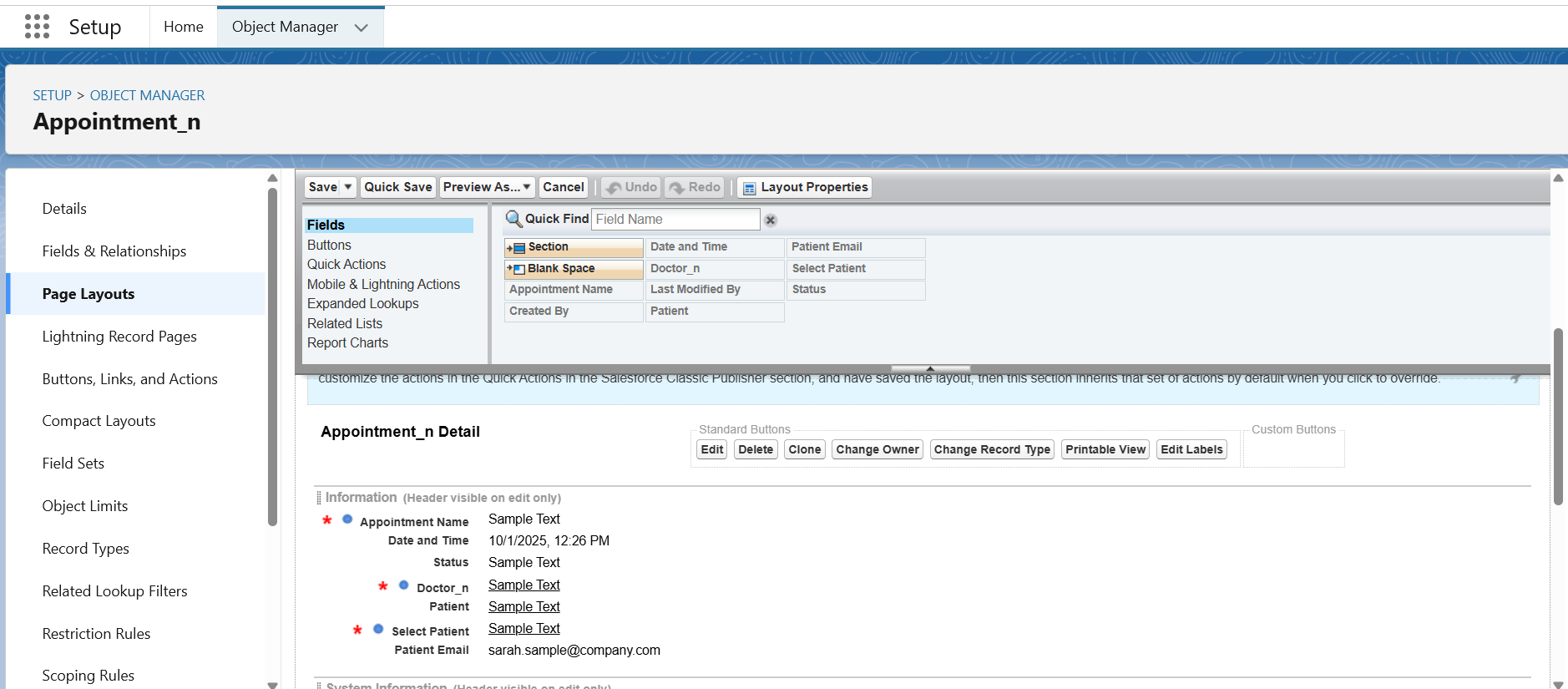
**3. Page Layouts:**

Home Page Layouts let you create a **custom dashboard** for the MediConnect App so that users can quickly see important information like today’s appointments, new patients, reports, and announcements as soon as they open the app.

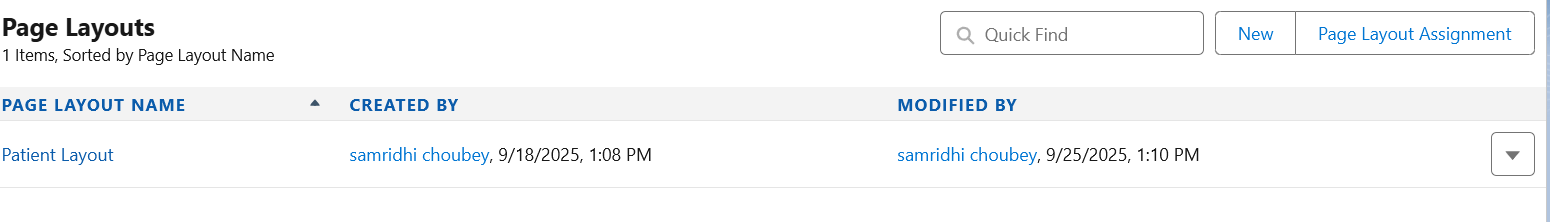
**Page Layouts in MediConnect:**

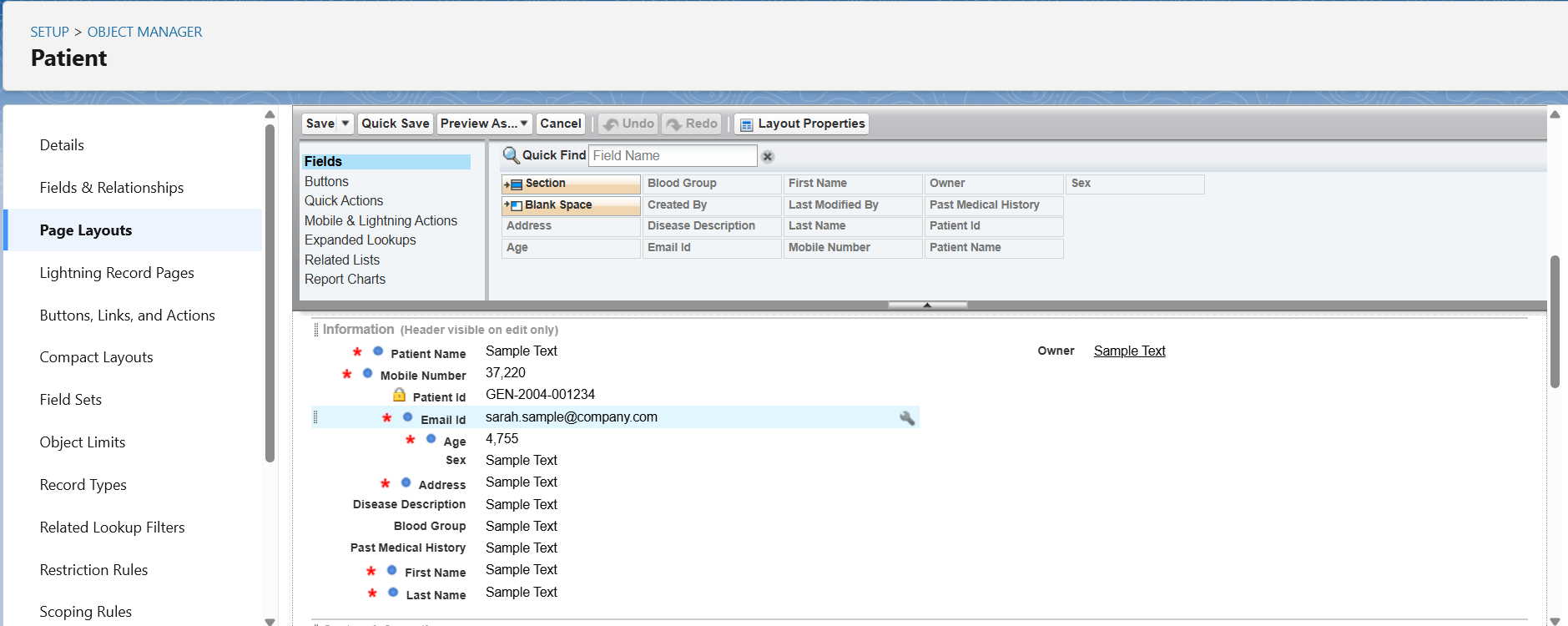
1. **Appointment\_n Layout:**



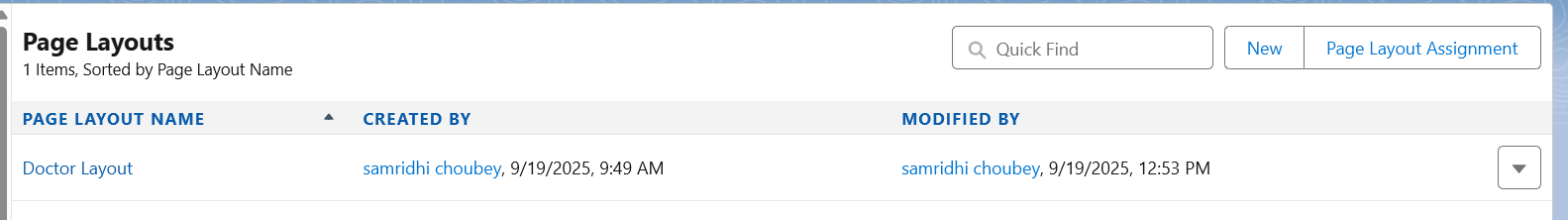


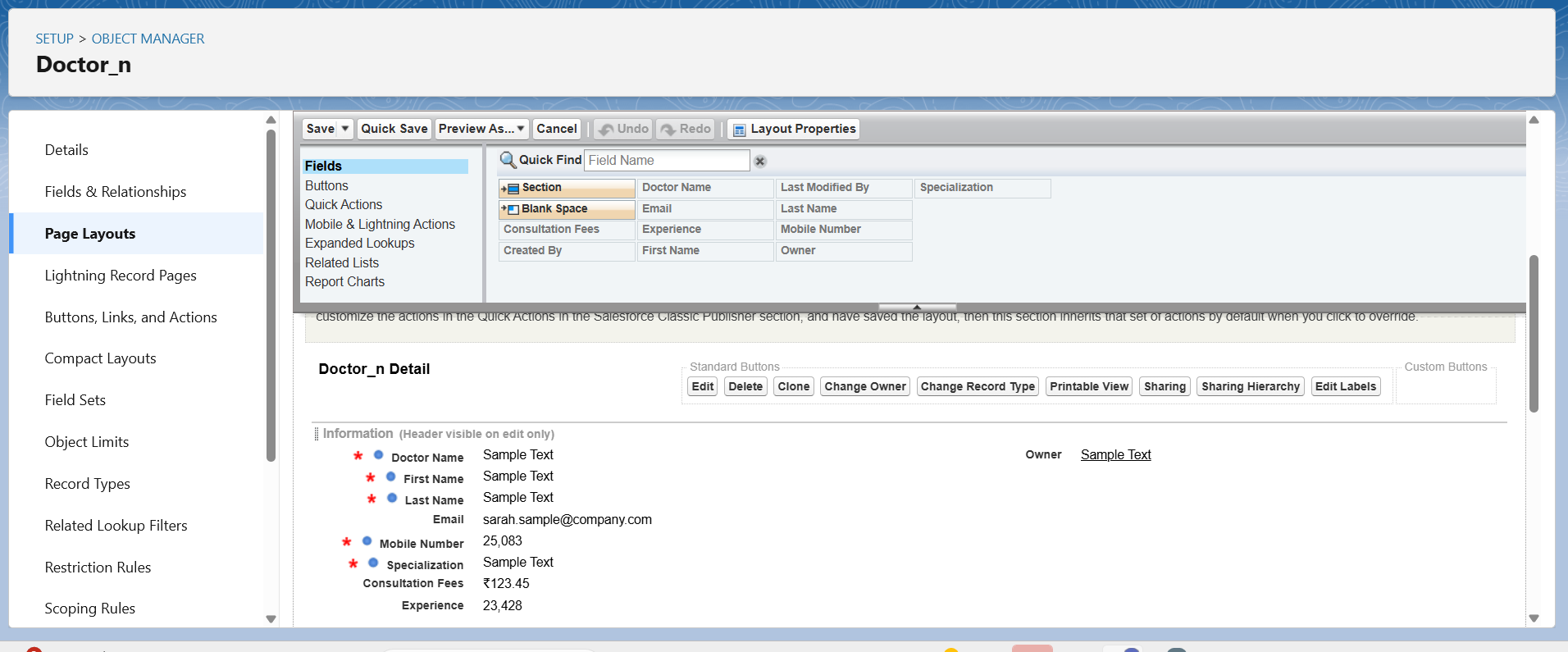
1. **Patient Layout:**



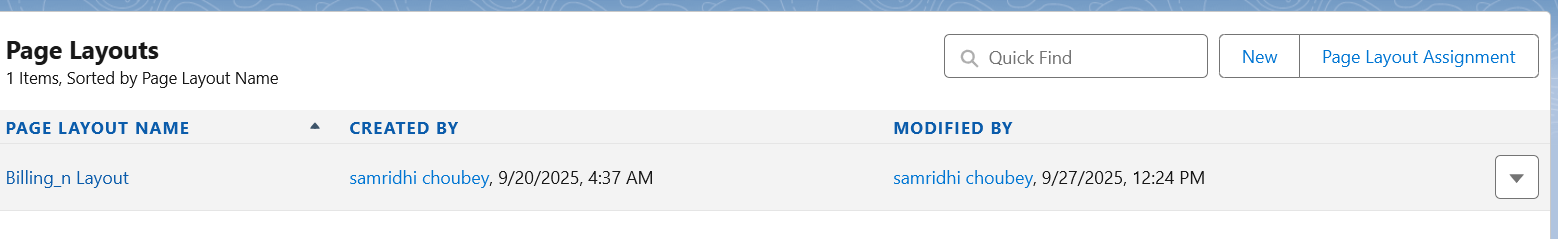


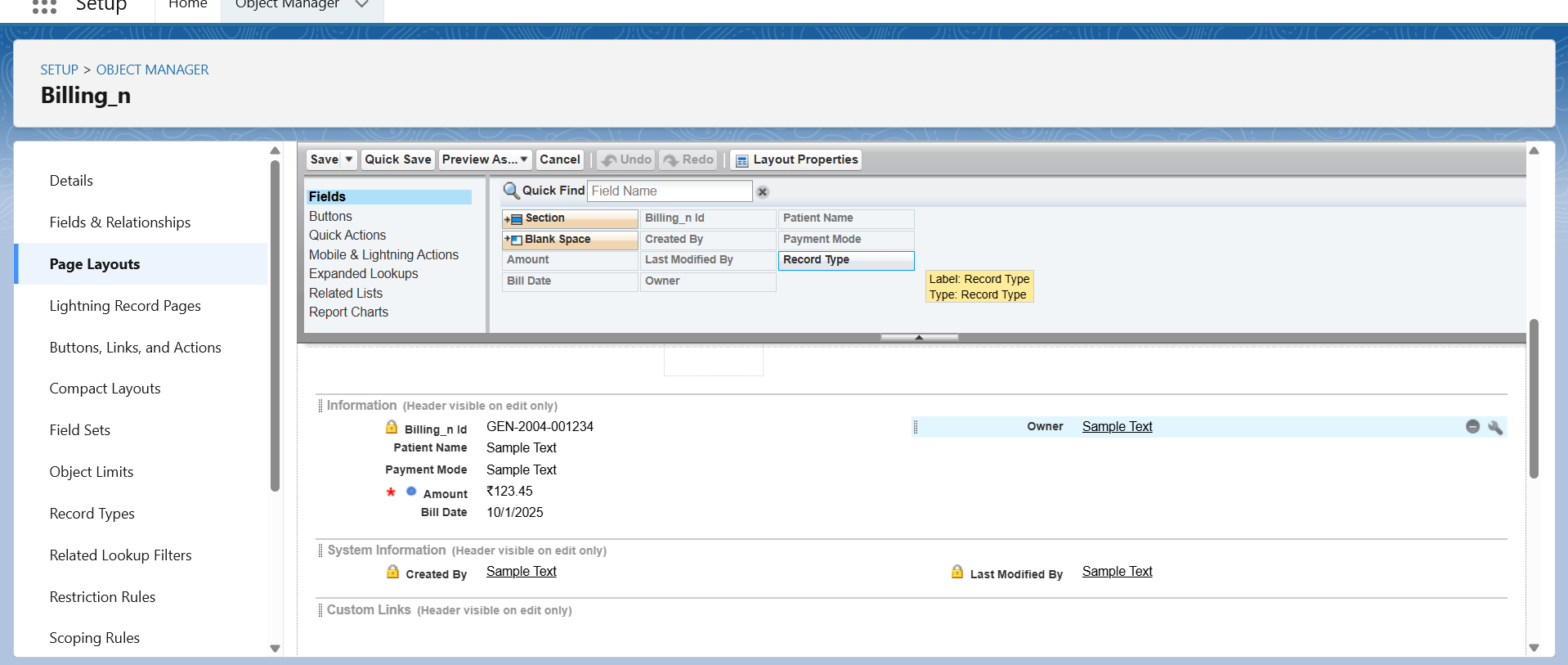
1. **Doctor Layout:**





1. **Billing\_n Layout:**

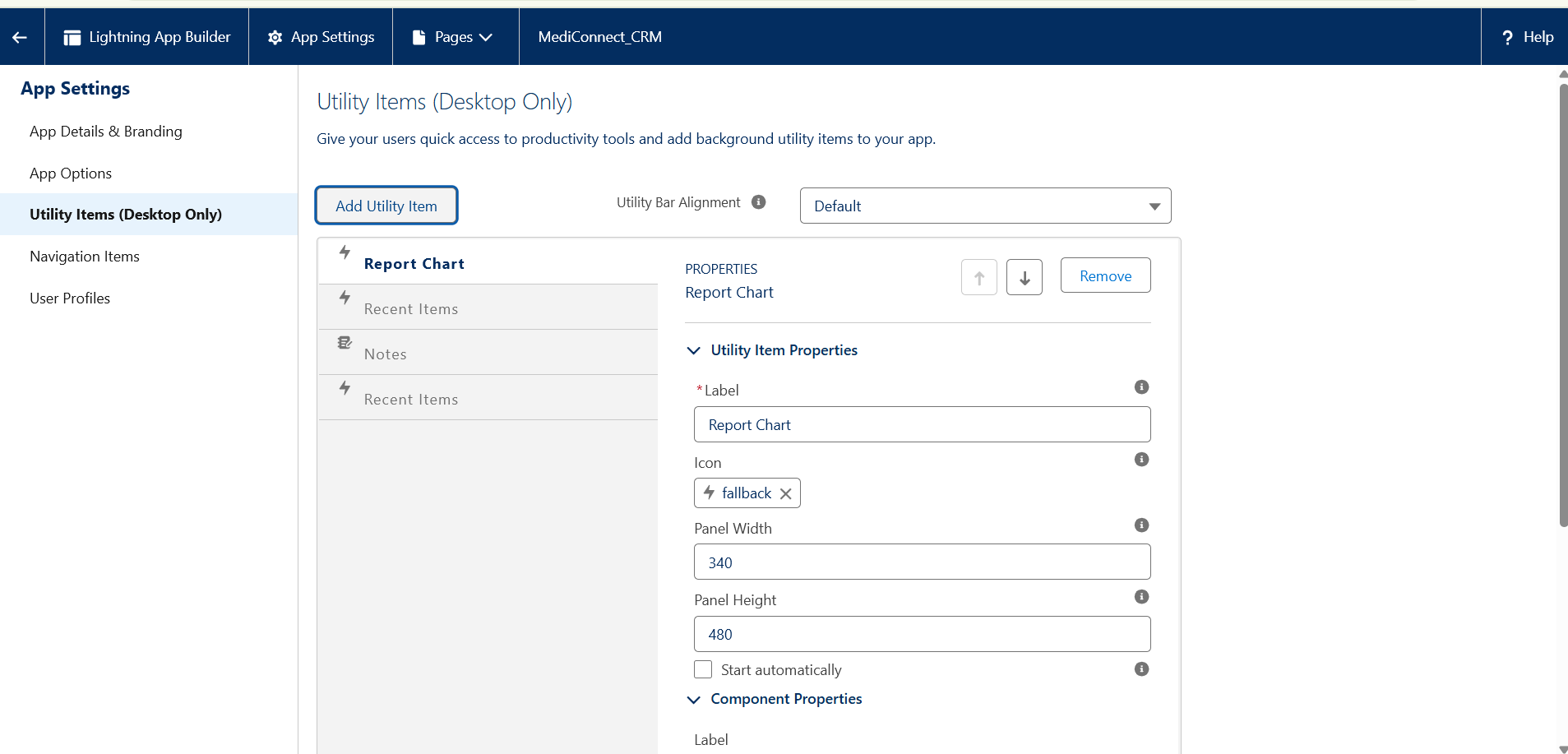




**4.Utility Bar:**

The Utility Bar provides quick access to important tools and actions at the bottom of the MediConnect app, so users can perform frequent tasks without leaving the page.

* Items included:
  + **Recent Items** → access recently viewed records.
  + **Notes** → add/view patient notes quickly.
  + **Report Charts** → track key statistics like daily appointments.
  + **Custom LWCs** → quick patient registration or alerts.
* Utility Bar ensures **frequent tasks can be performed without navigating away** from the current page.



**5.Lightning Web Components (LWC Basics)**

**Objective:** Create a basic Lightning Web Component and deploy it to Salesforce.

**Tools Used**

* VS Code with Salesforce Extension Pack
* Salesforce Developer Org
* Salesforce CLI

**Steps Performed**

1. **Authorize Salesforce Org**
   * Open VS Code → Press Ctrl+Shift+P → Select SFDX: Authorize an Org
   * Choose Sandbox/Developer Org
   * Login to Salesforce when prompted
2. **Create Lightning Web Component**
   * Press Ctrl+Shift+P → Select SFDX: Create Lightning Web Component
   * Enter component name: myFirstLWC
   * Choose default directory: force-app/main/default/lwc
   * VS Code creates the following folder structure.

**6.Apex with LWC**

**Objective:** Connect LWC with Salesforce backend using Apex.

**Theory:**

* Apex is a server-side language on Salesforce.
* LWC can call Apex methods to fetch or update data.
* There are **two ways** to call Apex:
  1. **@wire (Reactive method)** – automatically fetches data when component loads.
  2. **Imperative method** – fetches data on-demand, e.g., button click.
* Apex methods must be annotated with @AuraEnabled.

**Key Points:**

* @AuraEnabled(cacheable=true) → For wire adapters; improves performance.
* @AuraEnabled → For imperative calls; cacheable optional.

**Steps:**

1. Create Apex class with method annotated with @AuraEnabled.
2. Import Apex method into LWC using import methodName from '@salesforce/apex/ClassName.methodName'.
3. Call Apex method in JS using either @wire or imperative call.
4. Display data in component HTML using <template for:each>.

**Result:** LWC can dynamically fetch Salesforce data from backend

**7. Events in LWC**

**Objective:** Enable communication between components (Parent-Child).

**Theory:**

* LWC uses **custom events** to communicate between child and parent components.
* **Child to Parent communication**:
  1. Child component dispatches a **CustomEvent**.
  2. Parent component listens using on<eventname> attribute.
* **Parent to Child communication**:
  1. Pass data via **@api properties** in child component.

**Steps:**

1. In child component, create and dispatch CustomEvent with detail property.
2. In parent component, handle the event and access event.detail.
3. Update parent UI based on event data.

**Result:** Interactive components can communicate without tightly coupling logic.