Zewail City of Science, Technology and Innovation

University of Science and Technology

School of Computational Sciences and Artificial Intelligence

CSAI 203 - Fall 2025

Introduction to Software Engineering

ClinicEase

Design Document

Team Number: 22

Team Members:

Name: Soha Essam ID: 202300335

Name: Farah Ahmed ID: 202300267

Name: Malak Mohamed ID: 202301650

Name: Mohamed Khaled ID: 202300282

Name: Nour Nader ID: 202300065

Representative Contact info:

s-nour.ayoub@zewailcity.edu.eg

23/11/2025

1. **Introduction:**
    - **1.1) Purpose of the document:**

This Software Design Document (SDD) aims to offer a thorough architectural and detailed design solution for the ClinicEase system. This paper outlines how the system will be constructed, whereas the earlier SRS document specified what the system must accomplish.The development team (Team 22) uses it as their main guide, converting the functional and non-functional requirements into particular technological structures, data models, and algorithms. In order to make it easier for developers to construct the frontend interfaces (HTML/CSS/JS), database administration (MySQL/SQLite), and backend logic (Flask), this guide attempts to guarantee common technical knowledge.

- **1.2) Scope of the Design Phase:**

The architectural choices and comprehensive component specifications required to construct the ClinicEase web application are covered throughout the design process. In particular, this paper includes:

- **System Architecture:** Using the Model-View-Controller (MVC) architectural pattern, this high-level structural architecture is appropriate for the Flask framework.
- **Data Design:** In-depth database schema design that complies with ACID properties, including Entity-Relationship (ER) diagrams and table requirements for entities like patients, doctors, appointments, prescriptions, and inventory.
- **Component Design:** The interfaces and logic for the main modules, such as:

    - Role-based access control (RBAC) and user authentication
    - Logic for scheduling appointments and prioritizing emergencies
    - Prescription management and patient records.
    - Medical stock notifications and inventory tracking
    - Notification services and chatbot logic
    - The creation of the visual structure and navigation flows for the three distinct user dashboards (Administrator, Doctor, and Patient) is known as user interface design
    - Interface design includes specifications for internal HTTP/HTTPS communication between the web client and

server as well as integration points for external services like payment gateways.

- **1.3) Intended Audience:**

  The following ClinicEase project stakeholders are the target audience for this document:
  - **Software Developers (Team 22):** To oversee front-end and back-end component code and guarantee MVC architectural compliance.
  - **Database designers:** To accurately create the relational database structure (SQLite/MySQL) using the relationships and data dictionary.
  - **Software Testers/QA:** To create integration and unit test cases, they must comprehend the internal logic and data flow of the system.
  - **Project Supervisors/Instructors:** To assess the suggested solution's technical viability, architectural soundness, and design quality.

- **1.4) Overview of the contents:**

  The rest of this document is structured as follows:
  - **Section 2 (System Architecture):** Offers a high-level overview of the system, including the deployment view, the MVC decomposition, and the choice of technology stack (Flask or Python).
  - The database architecture, comprising the conceptual, logical, and physical data models, is described in Section 3 (Data Design).
  - **Section 4 (Component Design):** Describes the internal organization and functionality of particular software modules, such as the Payment Processor, Scheduling Engine, and Chatbot Module.
  - The design of the user experience, including screen layouts, navigation routes, and dashboard mockups for various user roles, is presented in Section 5 (User Interface Design).
  - **Section 6 (Requirements Traceability):** To provide complete coverage, the design elements are mapped back to the precise functional requirements specified in the SRS

2. System Overview:

- **2.1) Brief Description of the System:**

  ClinicEase is a web-based clinic management system designed to digitize and streamline daily clinic operations. It centralizes patient records, appointments, prescriptions, and communication into a single platform that is accessible to patients, doctors, and administrators. By replacing traditional paper-based processes, the system enhances efficiency, accuracy, and data accessibility. The system allows doctors to manage appointments, access medical histories, update patient records, and issue digital prescriptions. Patients can book or reschedule appointments, view medical records, receive notifications, and access prescriptions or lab results. Administrators oversee all operations, including user management, reporting, inventory tracking, and overall clinic performance. Through an intuitive interface supported by Flask, HTML/CSS/JS, and a relational database, ClinicEase improves communication, reduces errors, and supports smooth clinic workflow.

- **2.2) Key Design Goals and Constraints:**

  - **Key Design Goals:**

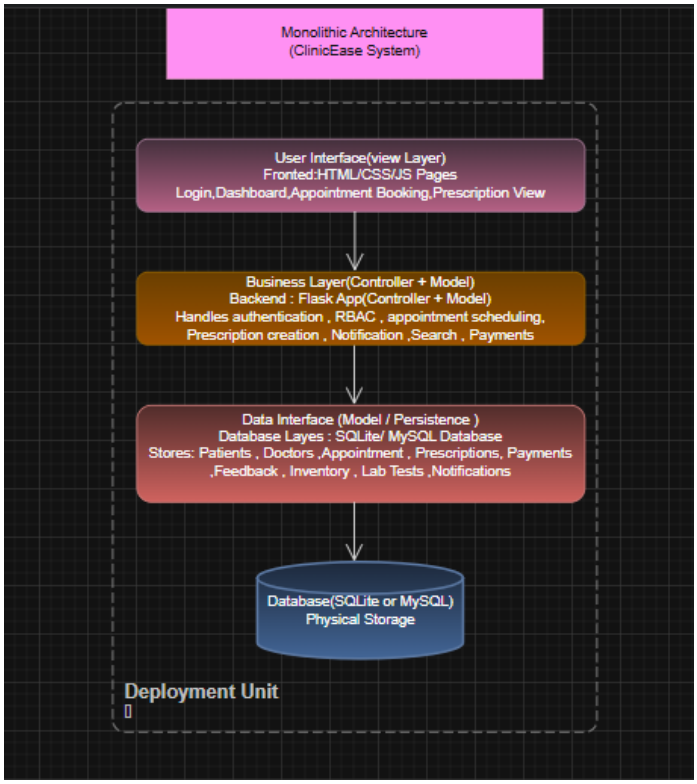
    - **Scalability:** The system is built to handle different types of users (patients, doctors, and admins) and support multiple modules such as appointments, prescriptions, payments, inventory, lab results, and more.
    - **Usability:** The system's goal is to offer a responsive, user-friendly interface with role-based dashboards and simple navigation.
    - **Security:** Includes encrypted data storage, secure login, HTTPS communication, and strict role-based access control to ensure data confidentiality.
    - **Reliability:** Designed to support multiple simultaneous users, with proper error handling and recovery mechanisms.

- **Constraints:**

  - Must use **Flask** for backend and **HTML/CSS/JavaScript** for frontend
  - Must follow the **MVC architecture** for code organization
  - System must operate over **HTTPS** for secure communication
  - The database (MySQL or SQLite) must comply with **ACID** properties
  - No **paid external APIs** may be used for core system functionality
  - Dependent on internet connectivity, compatible web browsers, and backend/database server availability

3. Architectural Design:

- **3.1) System Architecture Diagram:**



- **3.2) Discussion of Architectural Style and Components:**

  As required by the course requirements, the ClinicEase system has a monolithic architectural design. This method packages and deploys all essential features, such as the user interface, business logic, and data

storage, as a single, coherent unit. This design option was chosen for a team-based academic project with a clear scope and timetable because of its simplicity, ease of development, and straightforward deployment. The monolithic architecture is composed of three primary, tightly integrated layers: the User Interface (View Layer), the Business Layer (Controller + Model), and the Data Interface (Model / Persistence). These layers work in concert to deliver the full suite of features outlined in the SRS, from user authentication and appointment scheduling to prescription management and real-time notifications.

- **User Interface (View Layer):**

  The User Interface (UI) layer serves as the front-facing component of the system, responsible for rendering all web pages and handling user input. It is built using HTML5 and CSS3 to ensure compatibility across modern browsers and adherence to the course's front-end constraints. This layer presents distinct, role-specific dashboards for Patients, Doctors, and Administrators, Key UI components include:

  - Login and Signup forms
  - Appointment for booking and rescheduling interfaces
  - Prescription viewing and downloading pages
  - Patient medical record summaries
  - Notification panels and search/filter tools
  - Admin panels for user and inventory management

All interactions initiated by users (e.g., clicking "Book Appointment" or "Submit Feedback") trigger HTTP requests that are routed to the Business Layer for processing.

- **Business Layer (Controller + Model):**
  The Business Layer is the heart of the ClinicEase application, implemented using Flask framework in Python. It encapsulates the system's core logic enforcing business rules, managing state, and coordinating communication between the UI and the database. This layer fulfills the roles of both the Controller and the Model within the mandatory MVC design pattern.

- **Controller Responsibilities:**

The Controller component handles incoming HTTP requests, routes them to appropriate functions, and prepares responses to be sent back to the View layer. Key controller functions include:

- Authenticating users via credentials validation
- Enforce Role-Based Access Control (RBAC) to ensure patients, doctors, and admins can only access their permitted features
- Processing appointment scheduling, cancellation, and approval of workflows
- Managing the creation, editing, and retrieval of prescriptions
- Generating and dispatching automated notifications for appointments, payments, and lab results
- Handling online payment transactions and generating receipts
- Executing search and filter queries across patient records, appointments, and inventory

- **Model Responsibilities:**
  The Model component within this layer implements the business logic and data manipulation rules. It interacts with the Data Interface to perform CRUD operations on entities such as Patient, Doctor, Appointment, Prescription, Payment, and Inventory. The Model ensures data integrity by validating inputs, applying business rules (e.g., preventing double-booking), and maintaining consistency across related records. This layer also manages complex features like:

  - The Emergency Appointment module, which prioritizes urgent requests and notifies available doctors
  - Chatbot provides predefined responses to common user inquiries
  - The Feedback/Rating System, which stores and aggregates user reviews for performance evaluation
  - The Lab Test Results tracking, which links uploaded files to specific patient records

- **Data Interface (Model / Persistence):**

The Data Interface layer is responsible for abstracting the underlying database and providing a consistent API for the Business Layer to store and retrieve data. It is implemented using SQLite or MySQL, as specified in Section 2.4 of the SRS, ensuring ACID compliance for reliable data transactions. This layer maps directly to the entities defined in Appendix A (Data Dictionary) of the SRS, including tables for:

- Patient, Doctor, Admin (user management)
- Appointment, Prescription, Payment (core clinical workflow)
- Feedback, Lab_Test, Notification (auxiliary features)
- Inventory (medicine stock tracking)

The Data Interface handles all database queries, including:

- Creating new records (e.g., a new patient registration).
- Reading and filtering data (e.g., searching for appointments by date or doctor).
- Updating existing records (e.g., changing an appointment status or updating a prescription).
- Deleting records (e.g., canceling an appointment)

It also manages relationships between entities, such as linking a Prescription to a specific Appointment and Patient or associating Feedback with a particular Doctor.

- **Interaction Flow:**
  a) A user (Patient, Doctor, or Admin) interacts with the User Interface (e.g., click “Schedule Appointment”).
  b) The UI sends an HTTP request (GET/POST) to the Business Layer
  c) The Controller receives the request, validates the user’s role via RBAC, and invokes the appropriate Model logic.
  d) The Model performs the required business operation (e.g., checks doctor availability) and communicates with the Data Interface to query or update the database.
  e) The Data Interface executes the SQL command and returns the result to the Model
  f) The Controller processes the result, formats it into HTML (or JSON), and sends it back to the User Interface

g) The User Interface renders the updated page or displays a confirmation message to the user

Monolithic architecture is demonstrated by this smooth integration within a single deployment unit, which permits quick development and testing while guaranteeing that all system components stay coordinated and controllable. The MVC pattern mandated in Section 4.1 of the Design Document Outline is explicitly supported by this topic. The Controller's (managing requests) and Model's (implementing logic and data access) combined duties are clearly represented by the Business Layer, whilst the View is represented by the User Interface.

By selecting this architecture, ClinicEase guarantees a targeted, effective development process that satisfies all functional and non-functional requirements specified in the SRS, including security (through password hashing and RBAC), performance (by reducing inter-service latency), and reliability (through ACID compliant transactions).

- **3.3) Technology Stack and Tools:**
  - **Core Technologies:**
    - **Backend Framework:** Python + Flask Used to implement all business logic, route handling, authentication, RBAC, and database interactions.
    - **Frontend:** HTML5 and CSS3 Provides a responsive, role-specific user interface for Patients, Doctors, and Administrators.
    - **Database:** SQLite (with optional migration path to MySQL) A relational database system that stores all entities (Patients, Doctors, Appointments, Prescriptions, etc.)
    - **Communication Protocol:** HTTPS All client-server communication is secured using HTTPS.
  - **Development and Collaboration Tools:**
    - **Version Control:** Git + GitHub
      The team maintains a structured repository with /src, /docs, /tests, and /deployment folders as required by the course

(Guidelines, Section “GitHub Repository Structure”). All members contribute through meaningful commitments.

- **IDE/Code Editor:** Visual Studio Code (VS Code) or PyCharm Used for writing, testing, and debugging Flask backend code and frontend HTML/CSS/JS files.
- **Diagramming Tools:** Draw.io (diagrams.net) and Microsoft Power Point Used exclusively for creating UML diagrams, architecture diagrams, wireframes, and sequence diagrams.
- **Documentation:** Microsoft Word → exported to PDF All project documents (SRS, Design Document, Reports) are written in Word and converted to PDF with proper formatting.

- **Testing Tools:**
  - **Manual Testing:** Browser-based validation All frontend features (login, dashboard navigation, form submissions) are manually tested across Google Chrome, Mozilla Firefox, and Microsoft Edge
  - **Database Testing:** Direct query validation CRUD operations are verified using SQLite Browser or MySQL Workbench to ensure data integrity and correct schema implementation.

4. Detailed Design:

• 4.1) Model–View–Controller (MVC) Design Pattern:

○ 4.1.1) Description of MVC Pattern:

Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.

○ Why it was chosen:

The MVC architecture was selected for the ClinicEase system because it provides a clear separation of concerns, supports team

development, improves maintainability, enhances testability, and offers scalability for future features. Given that ClinicEase manages multiple user roles, sensitive medical data, and complex workflows such as appointments, prescriptions, and administrative management, MVC ensures a clean, modular, and extensible design that aligns with Flask's development pattern and supports long-term evolution of the platform.

- **4.1.2) Mapping of Project Components to MVC:**
  1. **Model layer:**
     Model Components:

Model Class / Table	Description
User	Stores login credentials, personal info, and roles. Base class for all system users.
Patient	Contains patient-specific data and links to medical history.
Doctor	Holds doctor information, specialization, availability schedule.
Admin	Manages system-level operations and reports.
Appointment	Stores appointment details, timestamps, and statuses.
MedicalRecord	Contains diagnoses, symptoms, visit details.
Prescription	Stores medicines prescribed to patients by doctors.
Medicine	Database of medication names, dosages, and types.
Payment	Tracks appointment payments and billing.
Feedback	Stores patient feedback and doctor ratings.
LabTest	Records laboratory test results and attachments.

Notification	Stores appointment reminders and system alerts.
ChatbotQuery	Logs chatbot interactions and responses.
AuditLog	Tracks all critical actions for security auditing.

2. **View layer (Front – End UI):**
   View Components:

View (Template)	Purpose
login.html	Allows users to log in.
patient_dashboard.html	Displays upcoming appointments, records, notifications.
doctor_dashboard.html	Shows schedule, patients, prescriptions to write.
admin_dashboard.html	Overview of system operations, reports, and management tools.
book_appointment.html	Form for patients to select doctor/time.
appointments_list.html	Shows appointments for patients and doctors.
medical_record_view.html	Displays medical history.
prescription_view.html	Lists prescribed medicines.
payment_page.html	Displays payment status or invoice.
feedback_form.html	Patient submits feedback.
labtest_upload.html	Upload or view lab reports.
notification_center.html	All reminders for the user.
chatbot.html	Chat interface for user–AI interaction.

3. **Controller Layer:**

Controller Modules

Controller Module	Main Functions
auth_controller.py	Login, logout, registration, password hashing.
patient_controller.py	Booking/rescheduling appointments, viewing medical records.
doctor_controller.py	Managing schedule, writing prescriptions, accessing patient history.
admin_controller.py	Managing doctors, patients, medicines, reports.
appointment_controller.py	Handles booking, cancellation, double-booking prevention.
medicalrecord_controller.py	Create/update medical records after visits.
prescription_controller.py	Add medicines, generate prescription.
payment_controller.py	Invoice creation, payment status checking.
feedback_controller.py	Submitting and processing patient feedback.
labtest_controller.py	Uploading lab test results and linking files.
notification_controller.py	Sending reminders and notifications.
Chatbot_controller.py	Handles chatbot queries and responses.

- **4.1.3) Responsibilities of Model, View, and Controller**
  a) Responsibilities of the model layer:
  Model responsibilities in ClinicEase:
  I. Data Storage & Retrieval:
  - Stores information about patients, doctors, admins, appointments, prescriptions, medical records, payments, feedback, lab tests, and notifications

- Performs CRUD operations on all tables

II. Business Logic & Rules:
- Validates appointment scheduling (e.g., prevent double-booking for doctors)
- Enforces constraints such as mandatory prescription details
- Ensures proper linking between medical records, prescriptions, and appointments

III. Data Relationships & Integrity:
- Manages foreign key relationships between entities
- Ensures referential integrity across user → patient → appointment → medical record chains

IV. Security & Validation:
- Hashes user passwords before saving
- Validates data formats (emails, dates, medicine dosage)

V. Domain-Specific Logic:
- Calculates prescription dosages
- Tracks patient history and past visits
- Handles payment totals and statuses

b) Responsibilities of the view layer

View responsibilities in ClinicEase:

I. Present Information:
- Display patient dashboards, doctor schedules, and admin control panels
- Show medical records, prescriptions, feedback, and notifications

II. User Input Collection:
- Appointment booking form
- Login and registration forms
- Prescription entry fields for doctors
- Admin forms for adding doctors or medicines
- Feedback input screens

III. Data Formatting & Presentation:
- Show data in tables, cards, charts, and visual components

- Render appointment calendars in a clean, readable layout

IV. User Experience & Navigation:
- Provide easy navigation between pages
- Ensure accessible and responsive design

V. View Rendering:
- Use Jinja2 templates to transform controller-provided data into HTML

c) Responsibilities of the control layer:
Controller responsibilities in ClinicEase:

I. Request Handling:
- Handle GET/POST requests from the user
- Validate form data submitted by patients, doctors, or admins

II. Business Logic Coordination
- Schedule, reschedule, or cancel appointments
- Generate prescriptions and attach medicines
- Create or update medical records after a visit
- Upload lab test files and attach them to records

III. Authentication & Authorization
- Manage login, logout, and session handling
- Restrict access based on user role (patient, doctor, admin)

IV. Interaction With Models
- Query models to retrieve data for dashboards
- Save new appointments, feedback, or lab tests
- Update prescriptions or doctor availability

V. Selecting and Updating Views:
- Pass model data to templates (Views)
- Decide which page to show after an action is completed

VI. Error Handling:
- Display error messages (e.g., invalid credentials)
- Catch scheduling conflicts and notify user

- **4.1.4) Interaction Between Components:**

The ClinicEase system uses the MVC architectural pattern to maintain a clean separation of responsibilities. During system operation, data flows in a structured pipeline between the View,

Controller, and Model components. This ensures maintainability, scalability, and clarity in how the system processes user actions. The following describes how data flows when a user interacts with the ClinicEase platform.

- Data Flow in MVC:

a) User Interacts With the View:
A user performs an action on a webpage (View), such as:
- Logging in
- Booking an appointment
- Uploading a lab test
- Writing a prescription
- Viewing medical records

These actions trigger a request (usually via a form or a button)

b) The Controller Receives the Request:

The Controller acts as the “middleman” , it:
- Interprets the request
- Validates input (e.g., check date format, ensure fields aren't empty)
- Determines which Model to interact with

Example: If a patient books an appointment, the appointment_controller receives the request.

c) Controller Communicates with the Model:

After validating input, the Controller calls Model functions to perform business logic:
Examples:
- Verify a doctor is available at the chosen time
- Insert new appointment data
- Retrieve medical history
- Create a prescription entry
- Fetch a list of medicines
- Update lab test results

The Model handles:

- Database operations

- ❖ Data validation
- ❖ Enforcing business rules
- ❖ Relationships between entities

d) Model Returns Data to the Controller:
The Model returns:

- ❖ Success or failure status
- ❖ Requested records (e.g., patient history)
- ❖ Updated objects (e.g., new appointment ID)
- ❖ Error messages (e.g., double-booking detected)

The Controller interprets this response.

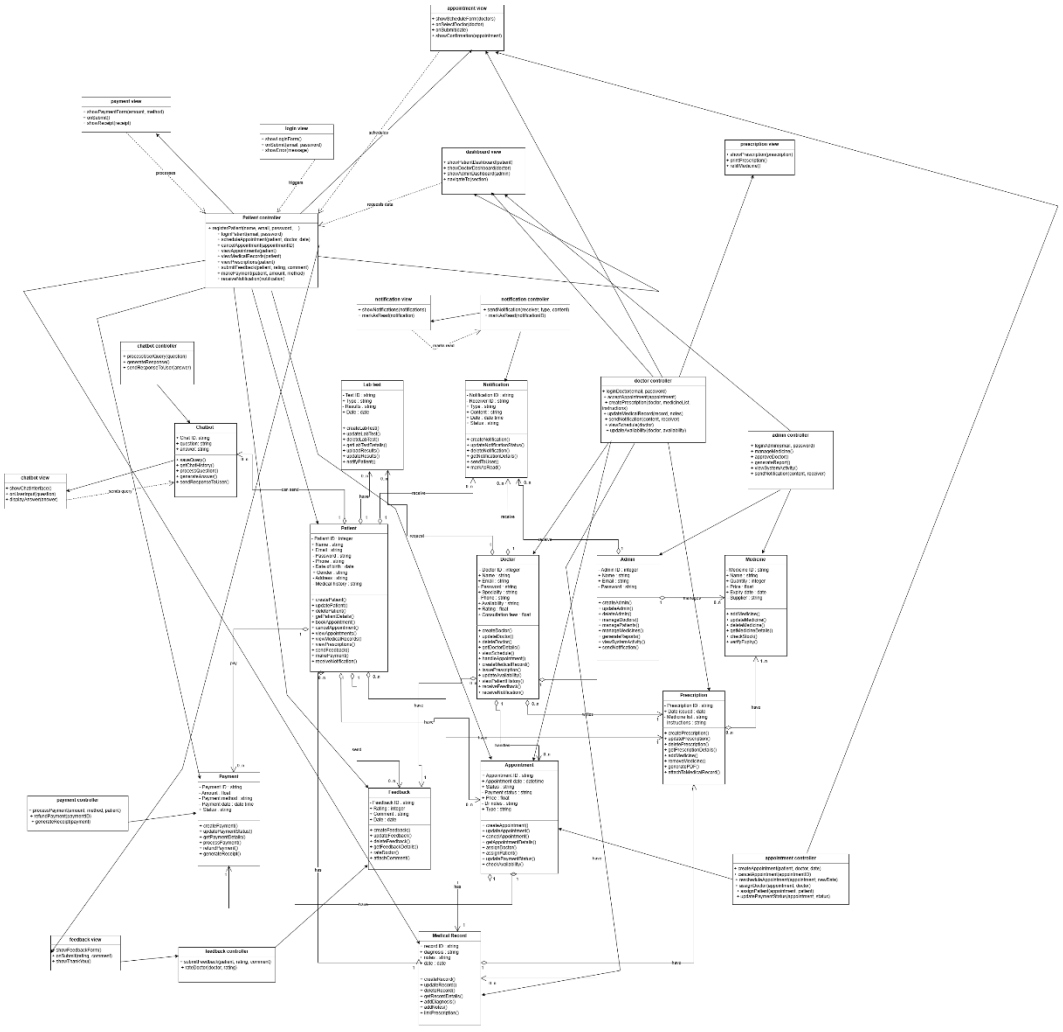
e) Controller Updates the View:
Finally, the Controller:

- ❖ Selects the appropriate template (View)
- ❖ Sends processed data to be displayed to the user

Examples:

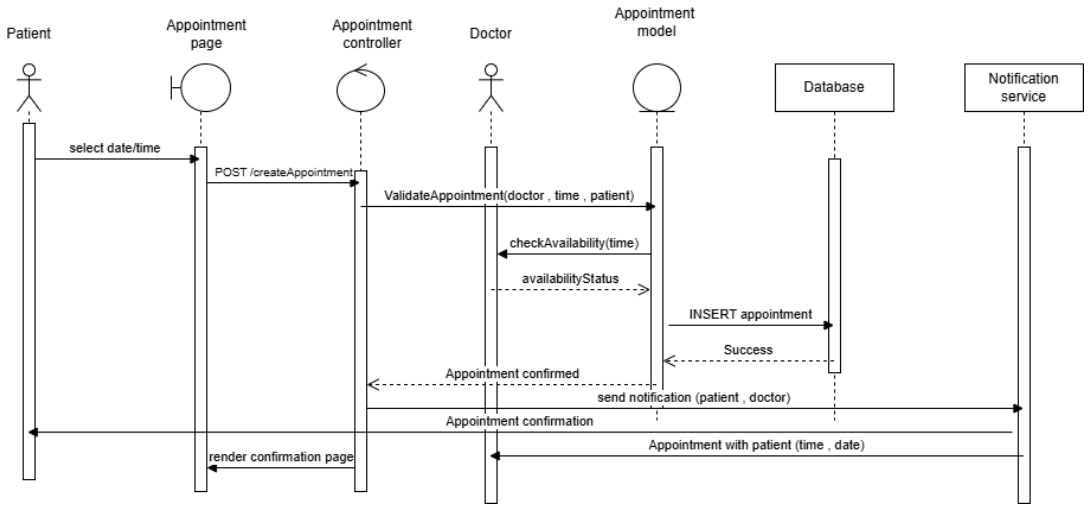
- ❖ After booking an appointment → show confirmation page
- ❖ After login → show dashboard
- ❖ After writing a prescription → show success message
- ❖ When viewing history → display formatted records

- **4.2) UML Diagrams:**
  - 4.2.1) Detailed Class Diagram:

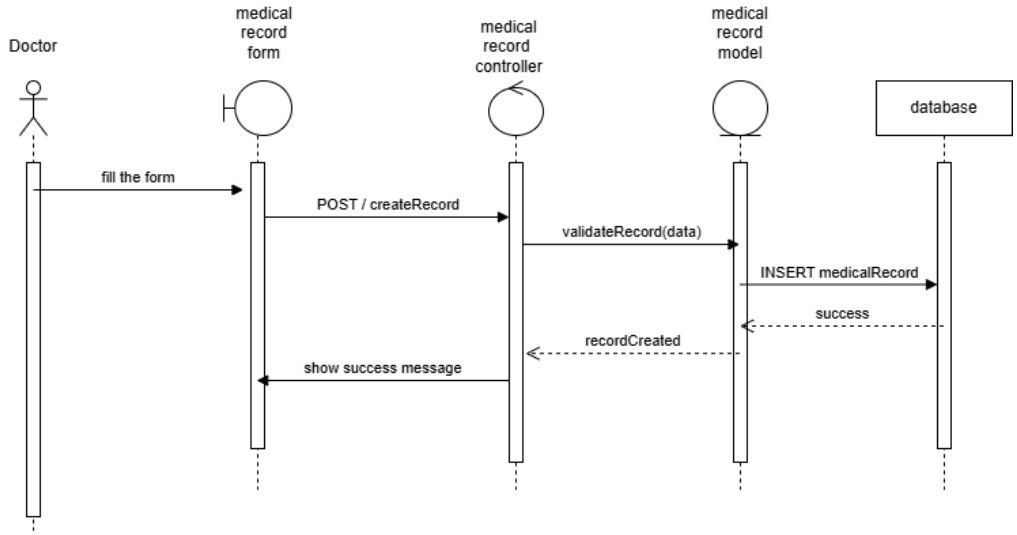


- 4.2.3) Sequence Diagram:

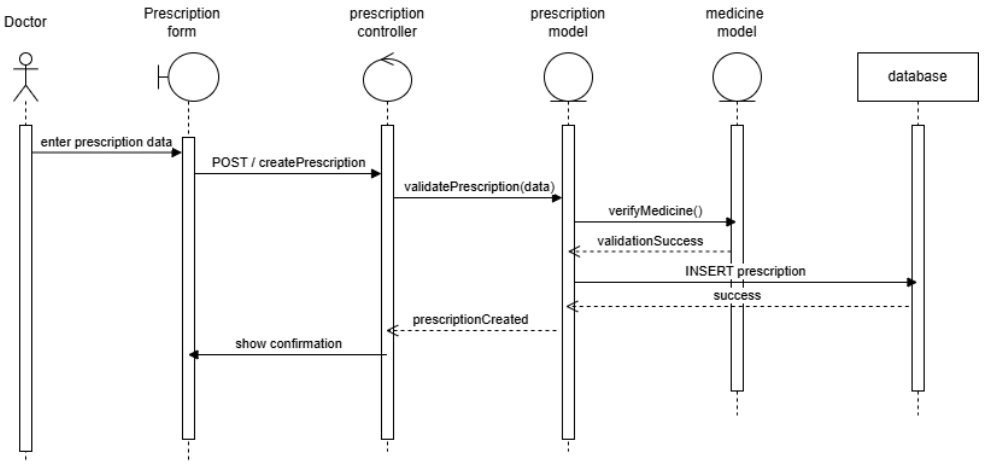
  - Sequence diagram 1: Patient books an appointment



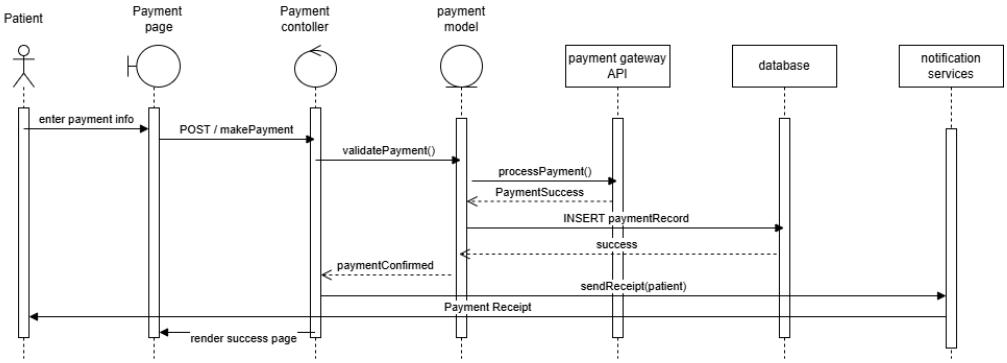
- Sequence diagram 2: Doctor creates a medical record



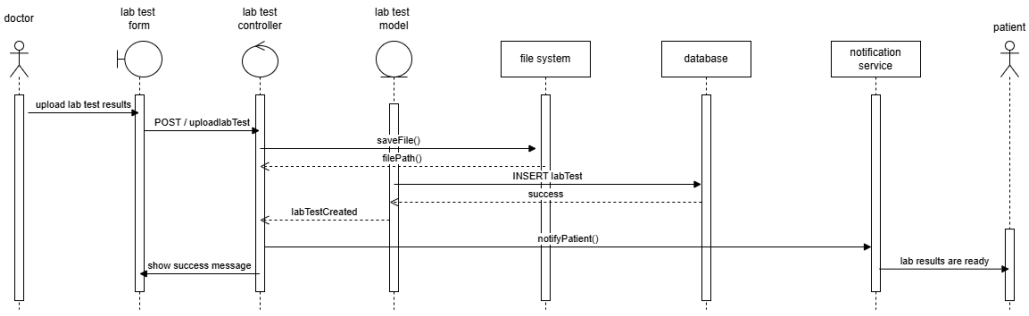
- Sequence diagram 3: Doctor issues a prescription



➢ Sequence diagram 4: Payment process

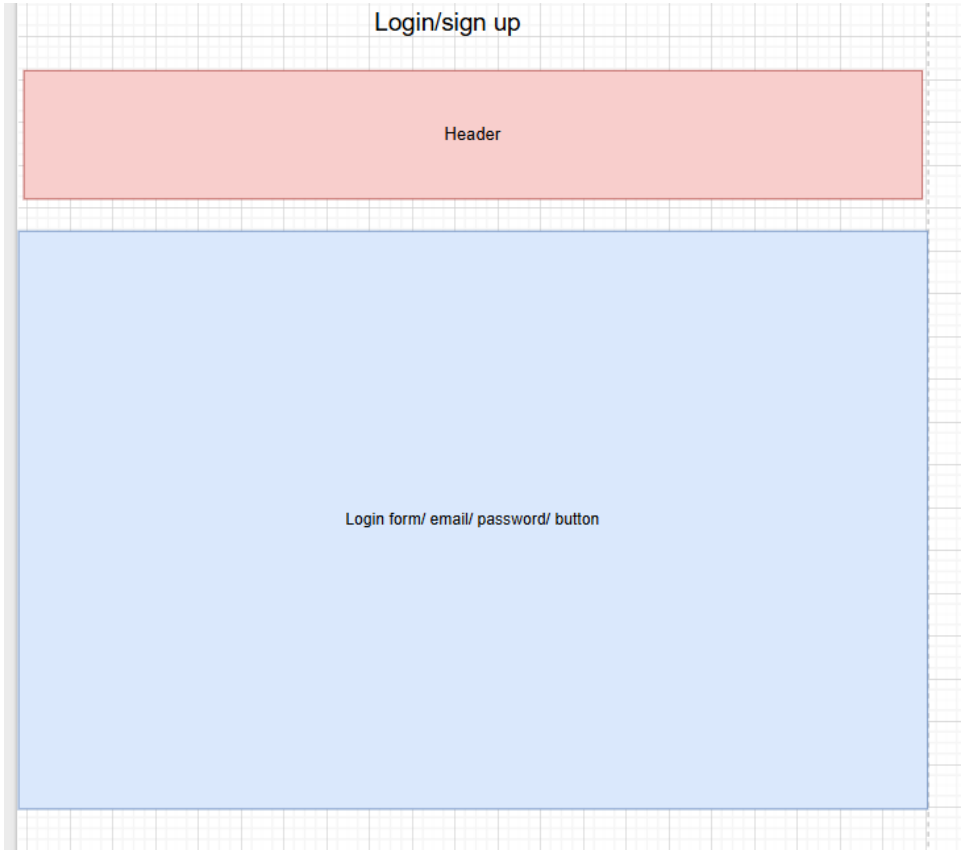


➢ Sequence diagram 5: lab test upload workflow

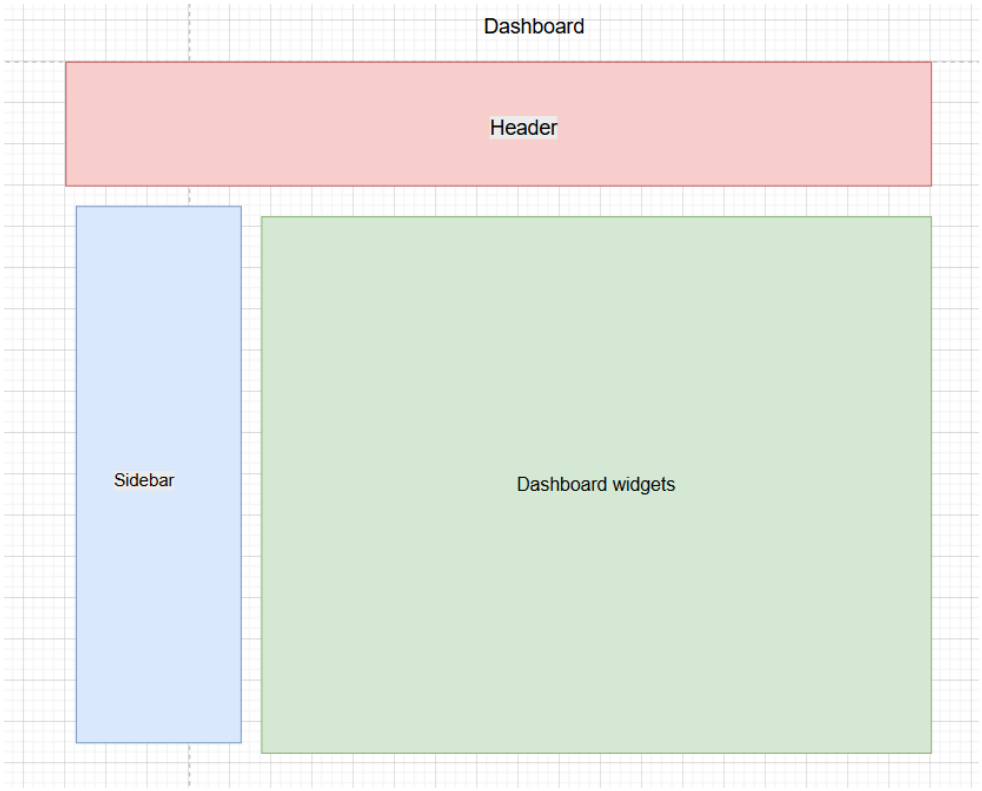


- 4.3) UI/UX workflow:

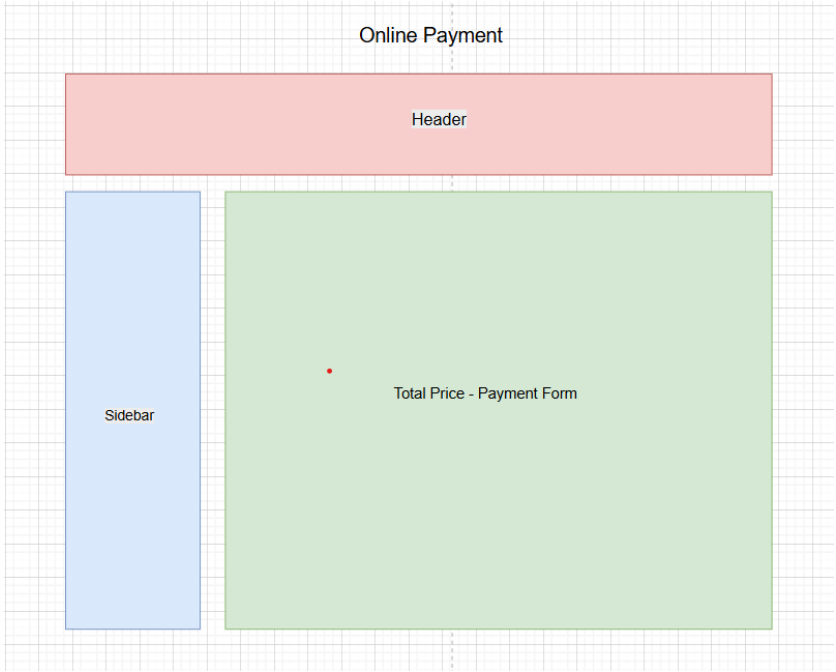
I. This mockup shows the login interface where users enter their email and password. It includes a centered authentication card, input fields, and a login button.



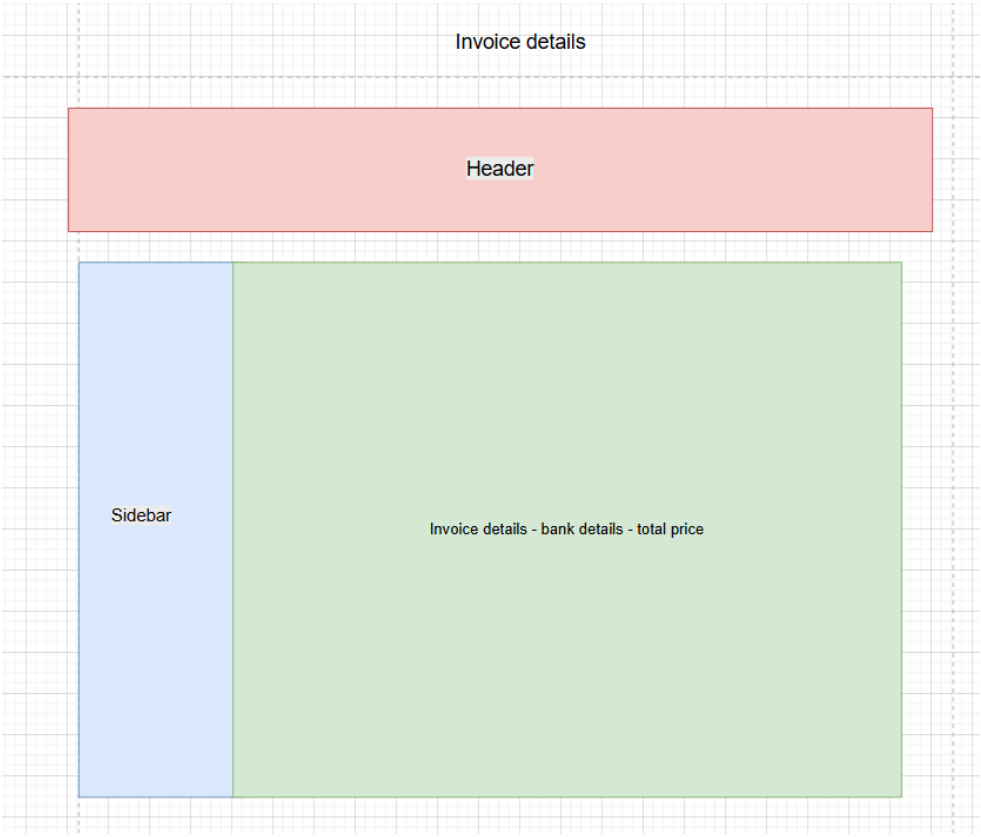
II. This wireframe represents the dashboard showing key widgets such as Today's Schedule, Upcoming Deadlines, Notifications, and Quick Actions.



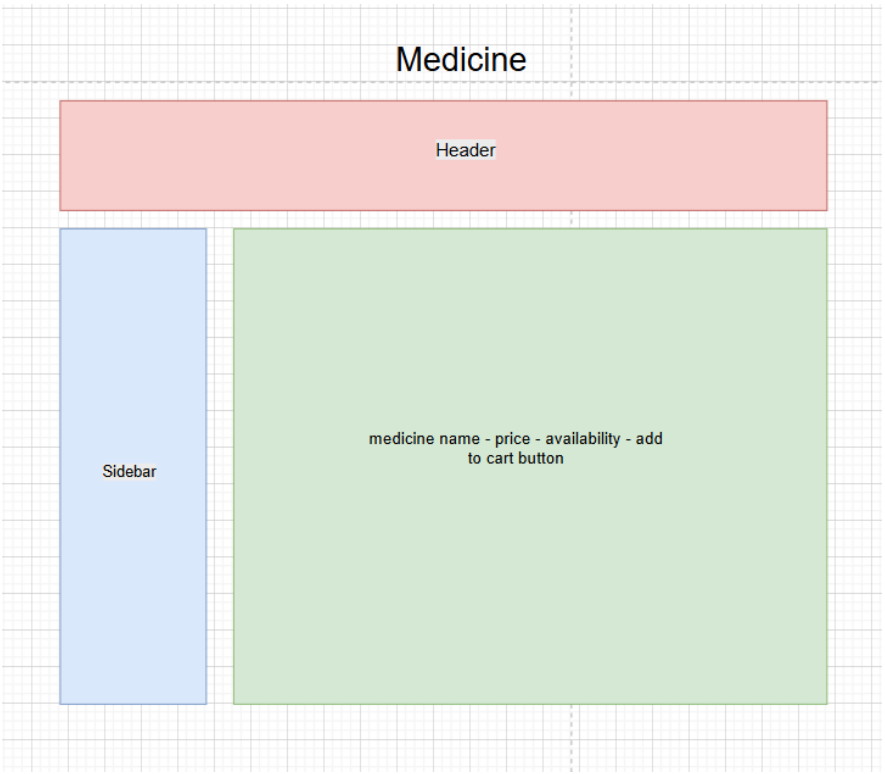
III. This wireframe represents the online payment of the patient.



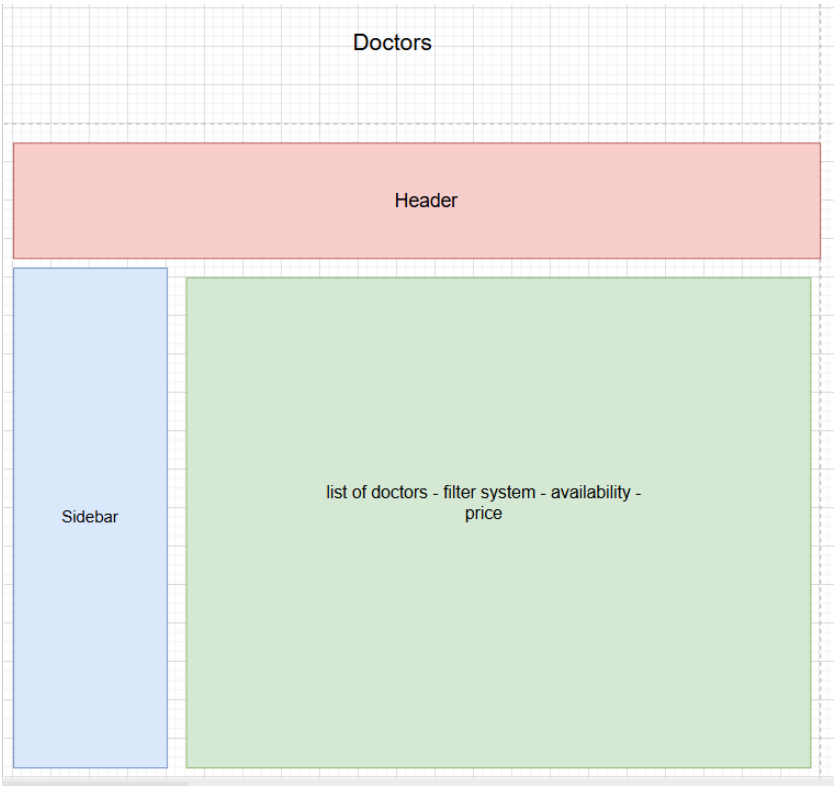
IV. This wireframe represents the invoice details page



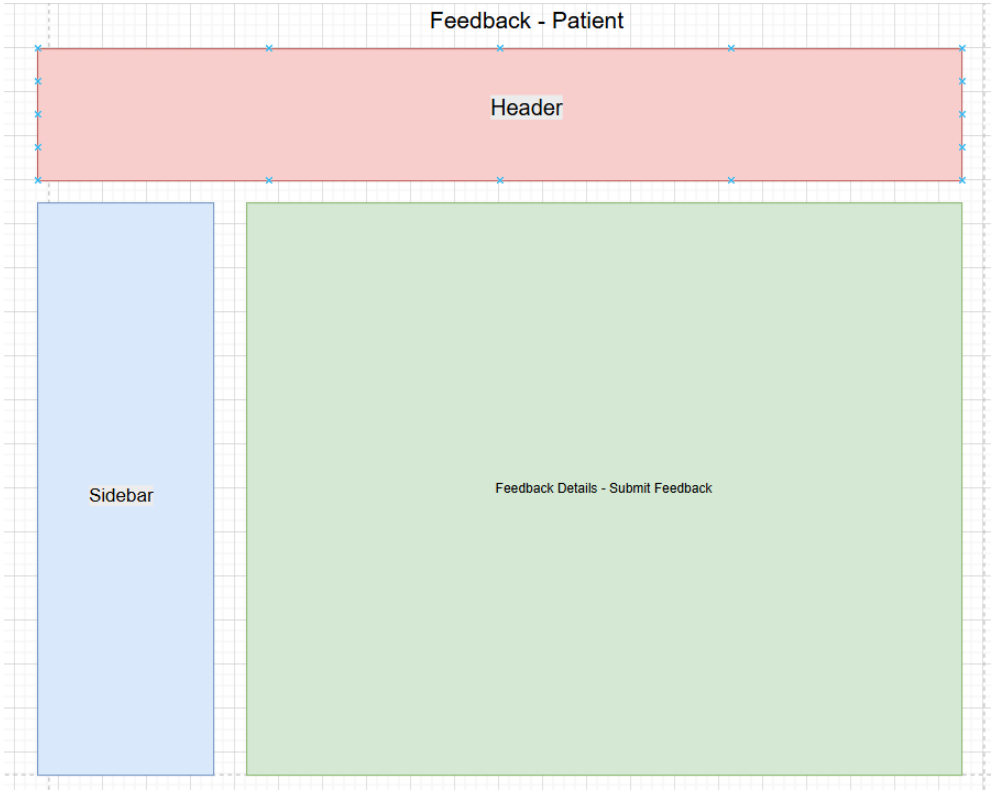
V. This wireframe represents the medicine page where the patient can view all the medicines available.



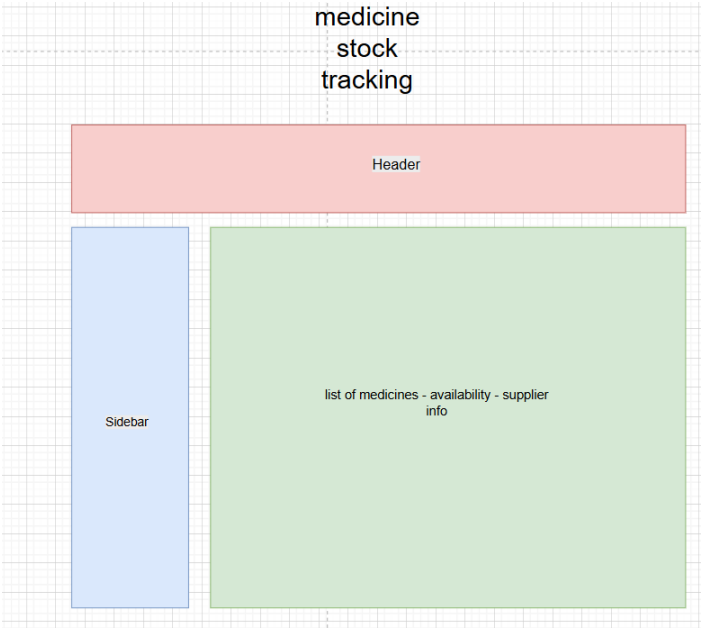
VI. This wireframe represents the list of doctors that work at the clinic



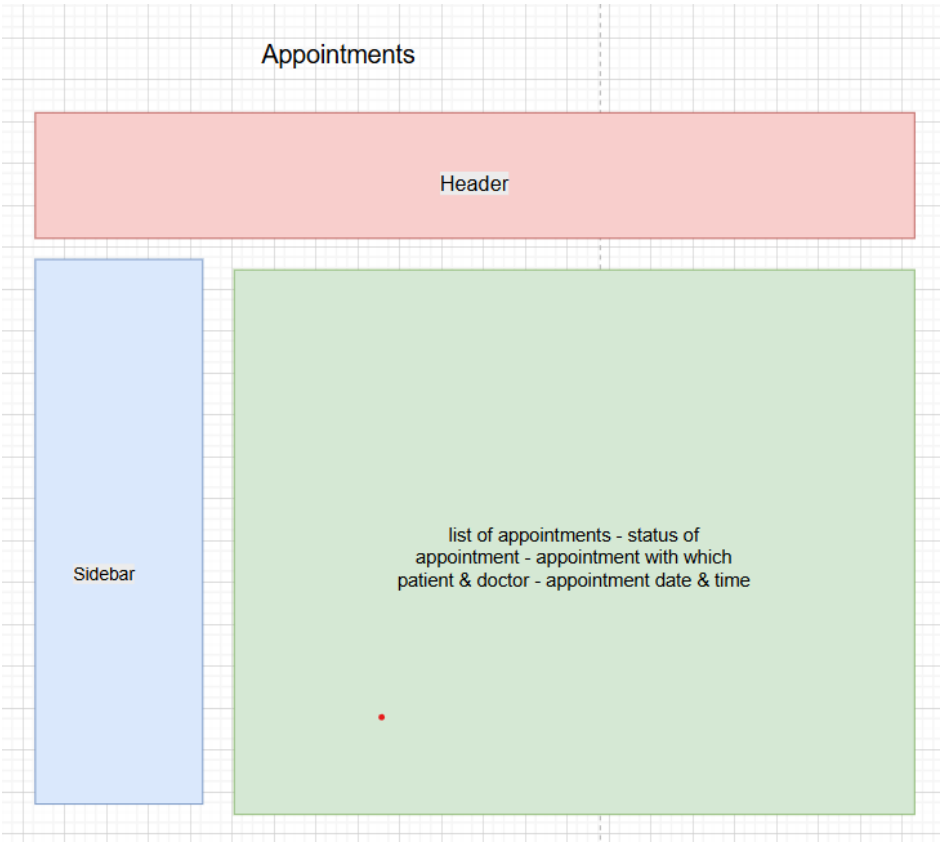
VII. This wireframe represents the feedback page where the patient submits their feedback



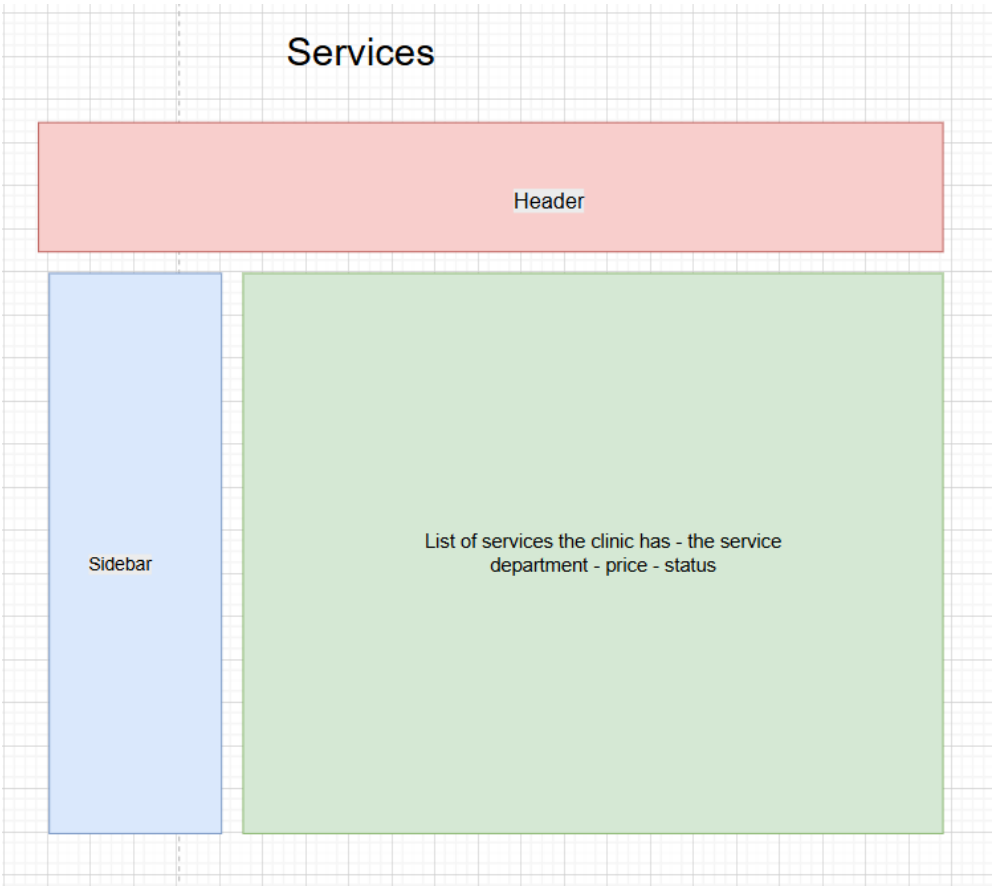
VIII. This wireframe represents the medicine stock tracking page for the admin users



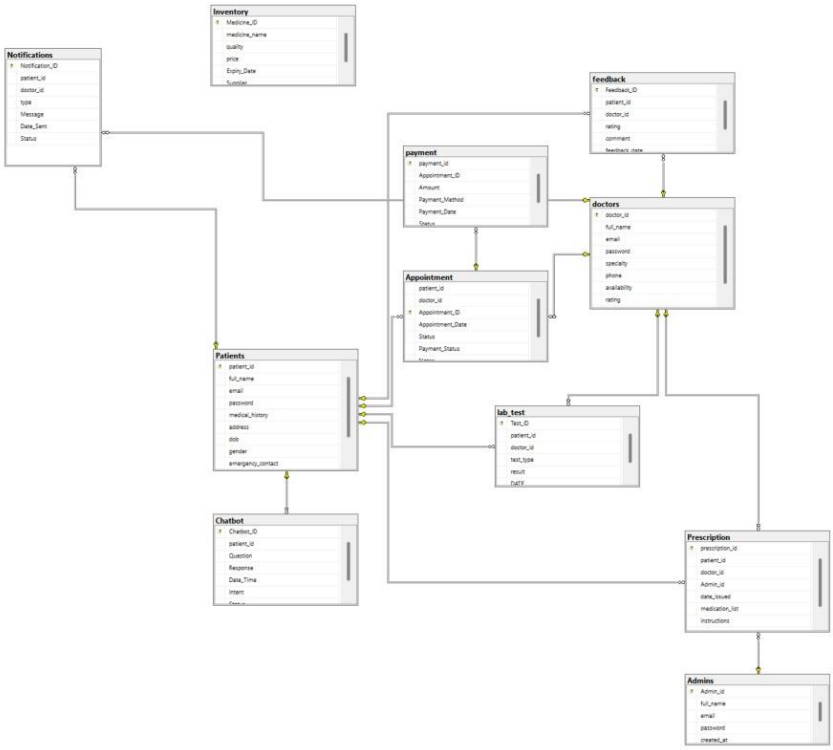
IX. This wireframe represents the appointments page where the user can see the registered appointments



X. This wireframe represents the services page where the patient can see all the available services the clinic does.



- **4.4) Data Design:**
  - **4.4.1) Database Schema:**



- **4.4.3) Data Dictionary:**

1. Patients:

Purpose: Stores all demographic and personal information about registered patients.

Column Name	Data Type	Purpose
Patient_id	INT(Primary Key)	Unique identifier for each patient.
Full_name	VARCHAR(255)	The full legal name of the patient.
Email	VARCHAR(255)	The patient's primary email address for communication.
Password	VARCHAR(255)	Hashed password for patient account authentication.
Medical_history	TEXT	A free-text or structured record of the patient's past medical conditions and treatments.
Address	VARCHAR(255)	The patient's residential address.

Dob	DATE	The patient's date of birth.
Gender	VARCHAR(10)	The patient's gender.
Emergency_contact	VARCHAR(255)	The name and/or phone number of the patient's emergency contact person.

2. Doctors:

Purpose: Stores information about the medical professionals registered with the system.

Column Name	Data Type	Purpose
Doctor_id	INT(Primary Key)	Unique identifier for each doctor.
Full_name	VARCHAR(255)	The full legal name of the doctor.
Email	VARCHAR(255)	The doctor's professional email address.
Password	VARCHAR(255)	Hashed password for doctor account authentication.
Specialty	VARCHAR(255)	The doctor's area of medical specialization
Phone	VARCHAR(255)	The doctor's direct phone number.
Availability	TEXT	A description or schedule indicating when the doctor is available for appointments.
Rating	DECIMAL(2, 1)	An average rating given by patients, typically on a scale of 0.0 to 5.0.

3. **Appointment:**

Purpose: Records all scheduled consultations between patients and doctors.

Column Name	Data Type	Purpose
Patient_id	INT (Foreign Key)	Links to the Patients table to identify the patient.
Doctor_id	INT (Foreign Key)	Links to the Doctors table to identify the doctor.
Appointment_ID	INT(Primary Key)	Unique identifier for each appointment.

Appointment_Date	DATETIME	The exact date and time of the scheduled appointment.
Status	VARCHAR(50)	The current status of the appointment.
Payment_Status	VARCHAR(50)	Indicates whether payment for this appointment has been made.
Notes	TEXT	Any additional clinical notes or remarks added by the doctor or staff regarding the appointment.

4. Payment:

Purpose: Tracks financial transactions related to appointments.

Column Name	Data Type	Purpose
Payment_id	INT(Primary Key)	Unique identifier for each payment record.
Appointment_ID	INT (Foreign Key)	Links to the Appointment table to associate the payment with a specific appointment.
Amount	DECIMAL(10, 2)	The monetary amount paid for the service.
Payment_Method	VARCHAR(50)	The method used for payment.
Payment_Date	DATETIME	The date and time when the payment was processed.
Status	VARCHAR(50)	The status of the payment.

5. Prescription:

Purpose: Stores details of medications prescribed by doctors to patients.

Column Name	Date Type	Purpose
Prescription_ID	INT(Primary Key)	Unique identifier for each prescription.
Patient_ID	INT(Foreign Key)	Links to the Patient table to identify the patient receiving the prescription.

Doctor_ID	INT(Foreign Key)	Links to the Doctors table to identify the prescribing doctor.
Admin_ID	INT(Foreign Key)	Links to the Admins table to identify the administrator who processed or verified the prescription.
Date_issued	DATE	The date on which the prescription was written and issued.
Medication_List	TEXT	A list of one or more medications prescribes, including dosage and frequency.
Instructions	TEXT	Specific instructions for taking the medication.

6. **Lab_Test:**

Purpose: Records laboratory tests ordered for patients and their results.

Column Name	Data Type	Purpose
Test_ID	INT(Primary Key)	Unique identifier for each lab test record.
Patient_ID	INT(Foreign Key)	Links to the Patients table to identify the patient.
Doctor_ID	INT(Foreign Key)	Links to the Doctors table to identify the ordering doctor.
Test_Type	VARCHAR(225)	The type of laboratory test performed.
Result	TEXT	The outcomes or findings of the lab test.
Notes	TEXT	Additional comments or interpretations from the lab technician or doctor.

7. **Feedback:**

Purpose: Captures patient feedback and ratings for doctors after appointments.

Column Name	Data Type	Purpose
Feedback_ID	INT(Primary Key)	Unique identifier for each feedback entry.
Patient_ID	INT(Foreign Key)	Links to the Patients table to identify the patient providing feedback.
Doctor_ID	INT(Foreign Key)	Links to the Doctors table to identify the doctor being rated.
Rating	INT	A numerical rating given by the patient.
Comment	TEXT	Optional text comment from the patient describing their experience.
Feedback_Date	DATE	The date when the feedback was submitted.

8. **Notifications:**

Purpose: Manages automated alerts and messages sent to patients and doctors.

Column Name	Data Type	Purpose
Notification_ID	INT(Primary Key)	Unique identifier for each prescription.
Patient_ID	INT(Foreign Key)	Links to the Patients table to identify the recipient patient.
Doctor_ID	INT(Foreign Key)	Links to the Doctors table to identify the recipient doctor.
Type	VARCHAR(50)	The category of the notification.
Message	TEXT	The body of the notification message.
Date_Sent	DATETIME	The timestamp when the notification was dispatched.
Status	VARDHAR(50)	The delivery status of the notification.

9. Chatbot:

Purpose: Logs interactions between patients and an AI chatbot for FAQs and preliminary triage.

Column Name	Data Type	Purpose
Chatbot_ID	INT(Primary Key)	Unique identifier for each chatbot interaction log.
Patient_ID	INT(Foreign Key)	Links to the Patients table to identify the user interacting with the chatbot.
Question	TEXT	The question or query posed by the patient.
Response	TEXT	The answer or response generated by the chatbot.
Data_Time	DATETIME	The date and time when the interaction occurred.
Intent	VARCHAR(255)	The detected intent or category of the patient's question.
Confidence	DECIMAL(3, 2)	A score (0.00 to 1.00) representing the chatbot's confidence in its response.

10. Admins:

Purpose: Stores information about administrative staff who manage the system.

Column Name	Data Type	Purpose
Admin_ID	INT(Primary Key)	Unique identifier for each administrator.
Full_Name	VARCHAR(255)	The full legal name of the administrator.
Email	VARCHAR(255)	The administrator's email address.
Password	VARCHAR(255)	Hashed password for admin account authentication.
Created_at	DATETIME	The timestamp when the administrator's account was created.

11. Inventory:

Purpose: Tracks the stock levels and details of medicines and supplies in the clinic/pharmacy.

Column Name	Data Type	Purpose
Medicine_ID	INT(Primary Key)	Unique identifier for each medicine item.
Medicine_Name	VARCHAR(255)	The official name of the medicine.
Quantity	INT	The current stock level of the medicine.
Price	DECIMAL(10, 2)	The retail price per unit of the medicine.
Expiry_Date	DATE	The date after which the medicine should not be used.
Supplier	VARCHAR(255)	The name of the supplier or manufacturer of the medicine.

5) Conclusion:

The design phase established a complete, MVC-based blueprint for *ClinicEase*, a Flask-powered web application for clinic management. The system follows a monolithic architecture with clear separation into Model (data & business logic), View (role-specific UIs), and Controller (request handling & coordination).

Key deliverables include:

- A relational database schema (11 normalized tables) ensuring ACID compliance and supporting core entities (Patients, Doctors, Appointments, Prescriptions, etc.).
- UML diagrams (class and sequence) modeling system structure and key workflows (e.g., booking, prescriptions, payments).
- Wireframes for Patient, Doctor, and Admin dashboards, emphasizing usability and role-based access.
- RBAC enforcement, secure authentication, emergency appointment logic, chatbot logging, and inventory tracking, all designed within course constraints (Flask, HTML/CSS/JS, HTTPS, no paid APIs).