University of Bahrain

College of Information Technology

Department of Computer Science

ITSE302
Software Architecture and Design
Project: Clinic Management System

Prepared by

Student name	Student ID
Mohamed Mahdi Radhi	202209265
Mariam Sharif	202207933
Lora Adel	202110274

Table of Contents

Requirement Description

1. Introduction

The **Clinic Management System (CMS)** is designed to streamline and automate the operations of a healthcare clinic. The system aims to **enhance patient care, reduce administrative workload, and improve operational efficiency** by digitizing essential functions such as patient registration, appointment scheduling, medical records management, billing, and prescription management. The CMS provides a **centralized platform** for patients, doctors, receptionists, and administrators to interact efficiently while ensuring data security and compliance with healthcare standards.

2. Objectives

The primary objectives of the **Clinic Management System** are:

- Provide a **user-friendly** and **efficient** interface for managing clinic operations.
- Automate **appointment scheduling** and **reduce waiting times**.
- Maintain **digital medical records**, ensuring easy access for doctors and patients.
- Facilitate **billing and payment processing**, improving financial tracking.
- Manage **prescriptions electronically**, reducing paperwork and errors.
- Generate **performance reports and analytics** for clinic administration.

3. Scope

The **Clinic Management System** covers the following essential features:

3.1 Functional Scope

- **User Management**: Patients, doctors, receptionists, and administrators can register and authenticate their accounts.
- **Appointment Scheduling**: Patients can book, reschedule, and cancel appointments based on the doctor's availability.
- **Electronic Medical Records (EMR)**: Doctors can update and access patients' medical history and diagnoses.
- **Billing & Payment Processing**: The system generates invoices and allows payments via multiple methods (cash, insurance, online).

- **Prescription Management**: Doctors can prescribe medications digitally, and pharmacists can access prescription details.
- **Reports & Analytics**: The system generates reports on appointments, financial transactions, and clinical efficiency.
- **Role-Based Access Control**: Different users have specific permissions (e.g., only doctors can edit patient records).

3.2 Exclusions

- The CMS **does not** provide telemedicine consultations in this version.
- The system is designed **for a single clinic**, with future scalability options for multiple branches.

4. Stakeholders

Stakeholder	Role & Responsibilities
Patients	Book appointments, view medical records, make payments, and receive prescriptions.
Doctors	Manage appointments, update patient medical records, issue prescriptions, and access reports.
Administrators	Oversee clinic operations, manage users, configure settings, and generate reports.
Pharmacists	Access prescription data and manage medication dispensing.

5. Constraints & Assumptions

5.1 Constraints

- Patient **data privacy regulations (e.g., HIPAA, GDPR)** must be adhered to.
- The system should be **scalable** for future expansion.
- The system will **support English as the primary language**.
- The system requires a **stable internet connection** for cloud-based operations.

5.2 Assumptions

- Users **have basic computer literacy** to operate the system.
- The clinic has **a dedicated IT support team** for maintenance and troubleshooting.

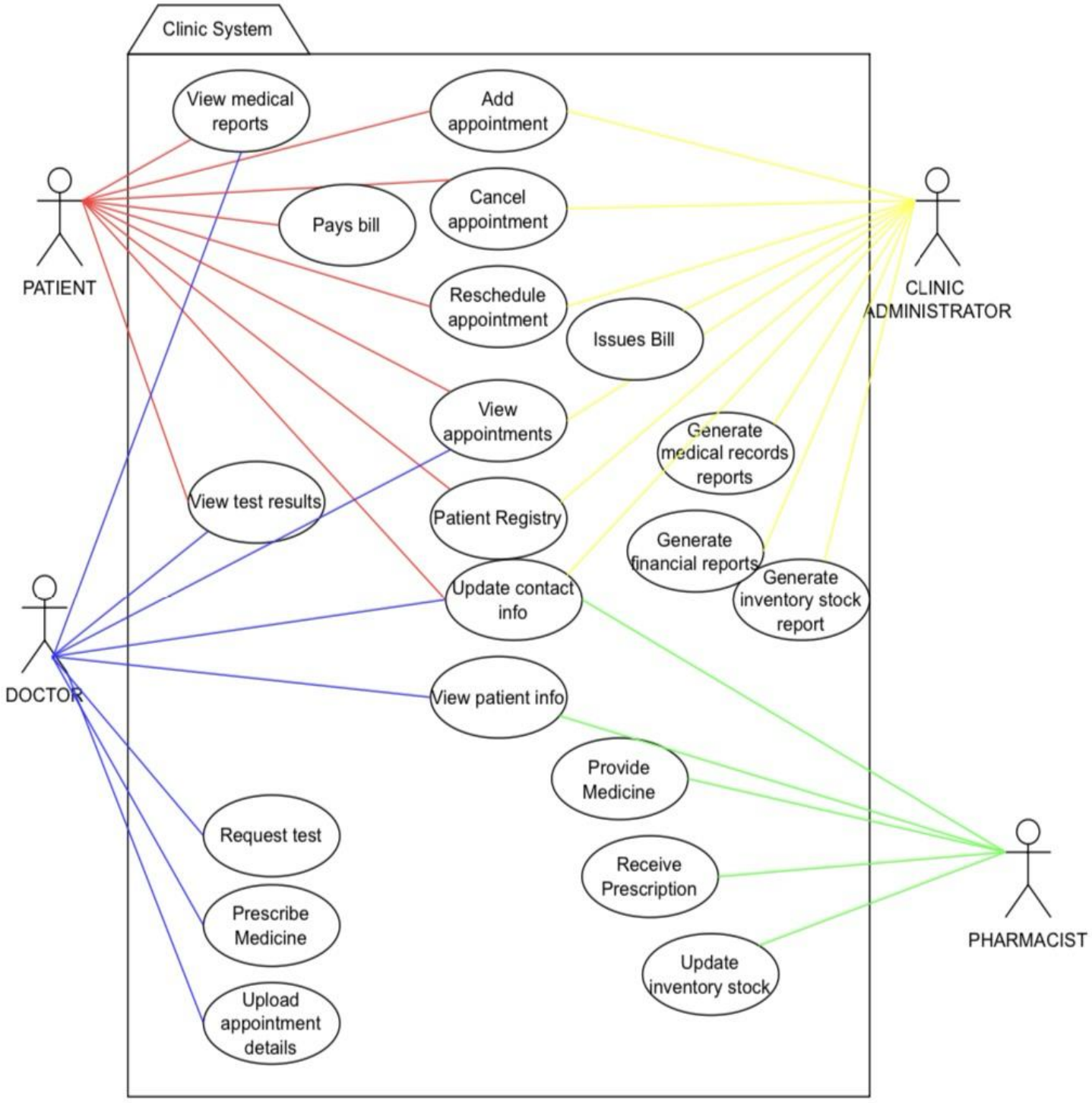
6. Functional Requirements

Requirement ID	Requirement Title	Description	Priority
CMS-001	User Authentication	Users must log in with secure credentials based on role-based access.	High
CMS-002	Patient Registration	Patients can create an account and update personal information.	High
CMS-003	Appointment Scheduling	Patients can book, modify, or cancel appointments. Doctors and receptionists can manage schedules.	High
CMS-004	Medical Records Management	Doctors can store and retrieve patient medical history securely.	High
CMS-005	Prescription Management	Doctors can issue electronic prescriptions, and pharmacists can access them.	High
CMS-006	Billing & Payments	The system must generate invoices and track payments for services.	High
CMS-007	Reports & Analytics	Administrators can generate reports on patient visits, revenue, and system performance.	Medium
CMS-008	User Role Management	The system must support distinct roles with distinct permissions.	High

7. Non-Functional Requirements

Requirement ID	Requirement Title	Description	Priority
CMS-NF-001	Security	Patient data must be encrypted to prevent unauthorized access.	High
CMS-NF-002	Performance	The system should manage at least one hundred concurrent users without lag.	High
CMS-NF-003	Availability	The system should be available 99.9% of the time, except for maintenance.	High
CMS-NF-004	Usability	The system must have a user-friendly interface for non-technical users.	High
CMS-NF-005	Scalability	The system must support future expansion to multiple clinics.	Medium
CMS-NF-006	Response Time	The system should respond to user actions within 3 seconds.	High

Use Case Model



Use Case Descriptions for Clinic Management System:

Actors:

1. Patient
2. Doctor
3. Pharmacist
4. Clinic Administrator

Use Cases:

1. **Add Appointment**
    a. Actor: Patient
    b. Description: The patient schedules a new appointment with a doctor.
2. **Update Contact Info**
    a. Actor: Patient
    b. Description: The patient updates their personal contact information.
3. **Update Inventory Stock**
    a. Actor: Pharmacist
    b. Description: The pharmacist updates the inventory stock of medicines and supplies.
4. **View Medical Reports**
    a. Actor: Doctor
    b. Description: The doctor views the medical reports of patients.
5. **View Test Results**
    a. Actor: Doctor
    b. Description: The doctor views the test results of patients.
6. **View Patient Info**
    a. Actor: Doctor
    b. Description: The doctor views the personal and medical information of patients.
7. **Provide Medicine**
    a. Actor: Pharmacist
    b. Description: The pharmacist provides prescribed medicine to patients.
8. **Prescribe Medicine**
    a. Actor: Doctor

b. Description: The doctor prescribes medicine to patients.

9. **Upload Appointment Details**
    a. Actor: Clinic Administrator
    b. Description: The clinic administrator uploads details of patient appointments.
10. **Generate Medical Records Reports**
    a. Actor: Clinic Administrator
    b. Description: The clinic administrator generates reports of medical records.
11. **Generating Stock Report**
    a. Actor: Clinic Administrator
    b. Description: The clinic administrator generates reports of inventory stock.
12. **Request Test**
    a. Actor: Doctor
    b. Description: The doctor requests medical tests for patients.
13. **Patient Registry**
    a. Actor: Clinic Administrator
    b. Description: The clinic administrator manages the patient registry.
14. **Receive Prescription**
    a. Actor: Pharmacist
    b. Description: The pharmacist receives prescriptions from doctors.
15. **Cancel Appointment**
    a. Actor: Patient
    b. Description: The patient cancels a scheduled appointment.
16. **Reschedule Appointment**
    a. Actor: Patient
    b. Description: The patient reschedules a previously scheduled appointment.
17. **View Appointments**
    a. Actor: Patient
    b. Description: The patient views their upcoming and past appointments.

18. Pay bill

a. Actor: Patient

b. Description: the patient selects appointments to pay for and selects payment method and pays.

19. Issue bill

a. Actor: Clinic Administrator

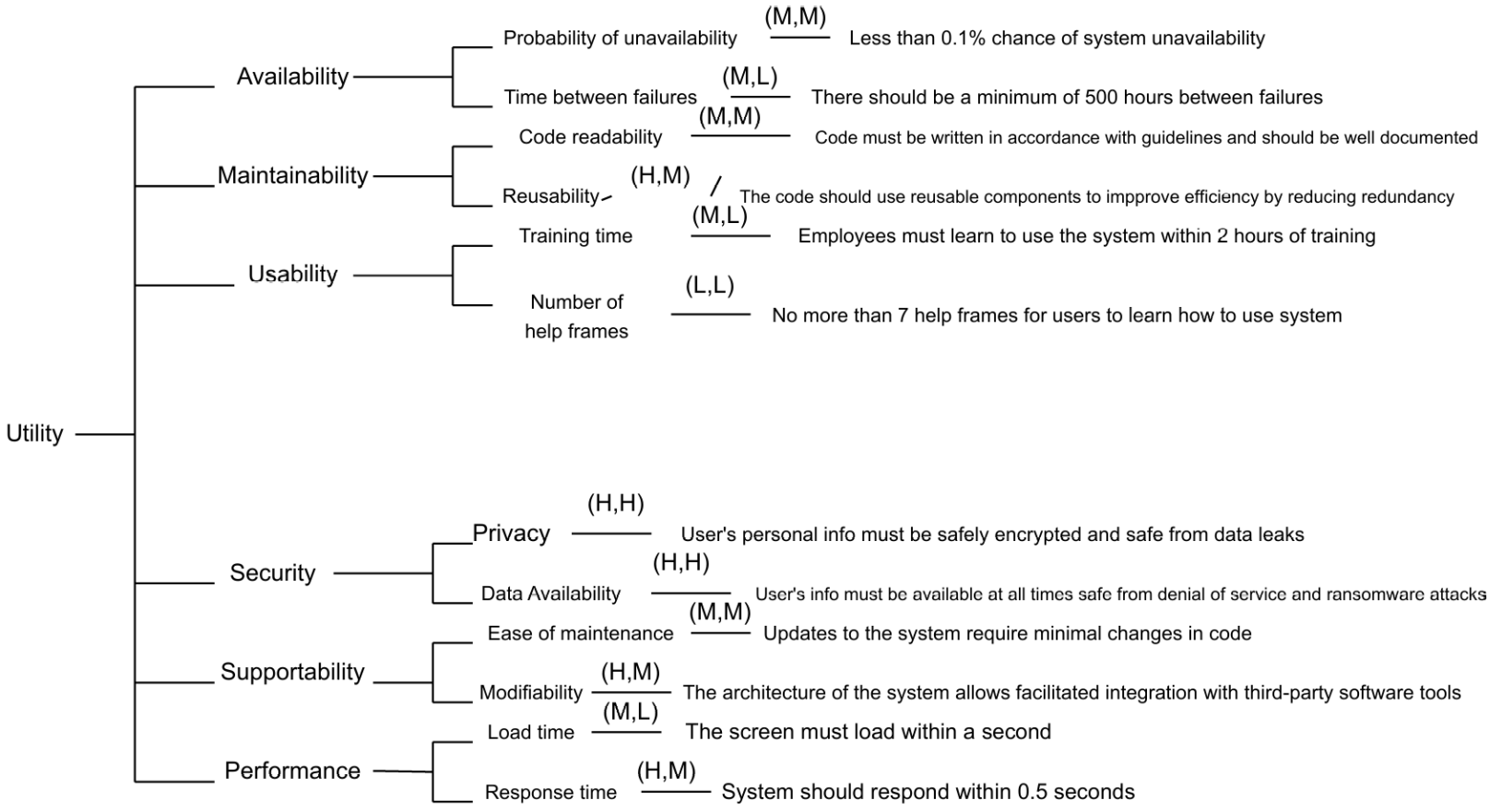
b. Description: clinic admin issues appointments bills.

20. Generate financial reports

a. Actor: Clinic Administrator

b. Description: clinic administrator may generate financial reports from accumulated financial data.

Utility Tree



Quality Attributes

ID	Quality Attributes	Scenario
QA-1	**Usability**	A receptionist with minimal computer knowledge logs in and schedules an appointment within 25 seconds without external help.
QA-2	**Availability**	During peak hours, the system remains operational 99.8% of the time, allowing doctors to access patient records without disruptions.
QA-3	**Security**	A nurse attempts to access a patient's financial records but is denied access due to role-based security policies.
QA-4	**Reliability**	A power outage occurs, and the system automatically switches to a backup server without losing patient data.
QA-5	**Performance**	The system should load a patient's full medical history within 3 seconds even when 200 concurrent users are logged in.
QA-6	**Maintainability**	A software update is deployed for insurance claim processing, and the update is completed within 8 minutes without disrupting clinic operations.

Priority Table:

Priority (Business Importance, Technical Risk)

	Low	Medium	High
Low		4	
Medium	5		1,3
High		2	6

Constraints

ID	Constraint
CON-1	The system must be accessible on all modern web browsers and compatible with common devices (desktop, tablet, mobile).
CON-2	A stable internet connection is required to access the cloud-hosted system.
CON-3	All medical and transactional records must be stored and accessible for a minimum of the last 5 years.
CON-4	The system must comply with applicable healthcare regulations (e.g., HIPAA, GDPR) regarding data privacy, access control, and audit logging.
CON-5	The system must support at least 100 concurrent users without degradation in performance.
CON-6	To prevent data loss, the system should auto-save and back up sensitive data in real time or at a minimum interval of every 5 minutes.

Concerns

ID & Title	Concern
CRN-1: Medical Data Storage	The database must be flexible and scalable to accommodate large volumes of structured and unstructured medical data, such as prescriptions, lab results, and patient history.
CRN-2: Secure Authentication	The login system should support multi-factor authentication (MFA) and role-based access to ensure secure and personalized access for each type of user (doctor, admin, patient, pharmacist).
CRN-3: Appointment Booking Reliability	The appointment module must ensure no double-booking and real-time updates of doctor availability to maintain clinic workflow and avoid conflicts.
CRN-4: Data Sensitivity and Audit Logging	All user actions, especially those involving patient data and medical records, must be logged securely for accountability and traceability in case of audits.

ADD Steps for Iteration 1: Establishing an Overall System Structure

Step 1: Review Inputs

Category	Details
Design Purpose	Define the foundational architecture of the CMS, enabling scalability, modularity, and performance optimization.
Primary Functional Requirements	- UC-1: User Login (secure access) - UC-5: Manage Appointments (core function) - UC-11: Manage Users (admin control)
Quality Attributes	- QA-1: Usability - QA-2: Availability - QA-3: Security - QA-5: Performance
Constraints	- CON-1: Must run on Mac, Windows, Linux - CON-5: Support 100+ concurrent users - CON-6: Real-time data update
Architectural Concerns	- CRN-1: Flexible database for patients, doctors, appointments - CRN-2: Secure login - CRN-3: Real-time performance for bookings

Step 2: Establish Iteration Goal

In this first iteration, the primary goal is to **establish the high-level architecture of the Clinic Management System (CMS)**. This includes setting up the structure that will support secure user access, reliable appointment handling, and efficient performance under concurrent usage.

This step focuses on selecting drivers (functional and non-functional) that will shape how the system's key architectural components are organized. The selected drivers for this iteration are:

Iteration Goals:

- **Define the overall system structure** using layered architecture (presentation, logic, and data).
- **Ensure the system supports performance at scale** (e.g., loading patient records under 3 seconds even with 200 users).
- **Lay the foundation for secure access** with login and RBAC mechanisms.
- **Enable real-time responsiveness** and high availability through cloud-hosting and data caching strategies.
- **Address architectural concerns such as:**
  - CRN-1: Flexible and scalable database design
  - CRN-2: Strong authentication mechanisms
  - CRN-3: Real-time performance for critical operations like booking

Step 3: Choose Elements to Refine

As a greenfield system, we refine the **entire CMS system**:

- Interface Layer
- Application Logic Layer
- Data Management Layer
- Authentication & Access Module
- Appointment Scheduling Module

Step 4: Choose Design Concept That satisfy the Selected Drivers

In this step, the overall system structure is refined by selecting design concepts that best satisfy the drivers identified in Step 2. These decisions are intended to align the architecture with performance, usability, availability, and maintainability goals, while also addressing system constraints and architectural concerns.

Design Decision	Rationale
Use web-based reference architecture	Enables platform independence via web browsers (CON-1), improves accessibility and user responsiveness (QA-1, QA-2).
Use 3-tier deployment pattern	Separates UI, logic, and data layers for better scalability (QA-5), maintainability, and load balancing (QA-2).
Use cloud-hosted services	Provides scalability and fault tolerance (QA-2) and ensures consistent real-time performance (CON-6).
Use externally developed UI tech (HTML/CSS/JS)	Familiar technologies for faster development (CRN-3), ensuring cross-platform UX compatibility (CON-1).
Enable real-time updates and caching	Supports quick data access and synchronized updates for users (QA-2, QA-5).

These design concepts will guide the system architecture going forward. The decisions establish the foundational structure and technologies that will be further refined in the next iteration.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The following table summarizes the architectural components and their corresponding responsibilities, designed to satisfy the performance requirements (QA-5), as well as support other system concerns such as availability (QA-2), real-time updates, and external integrations.

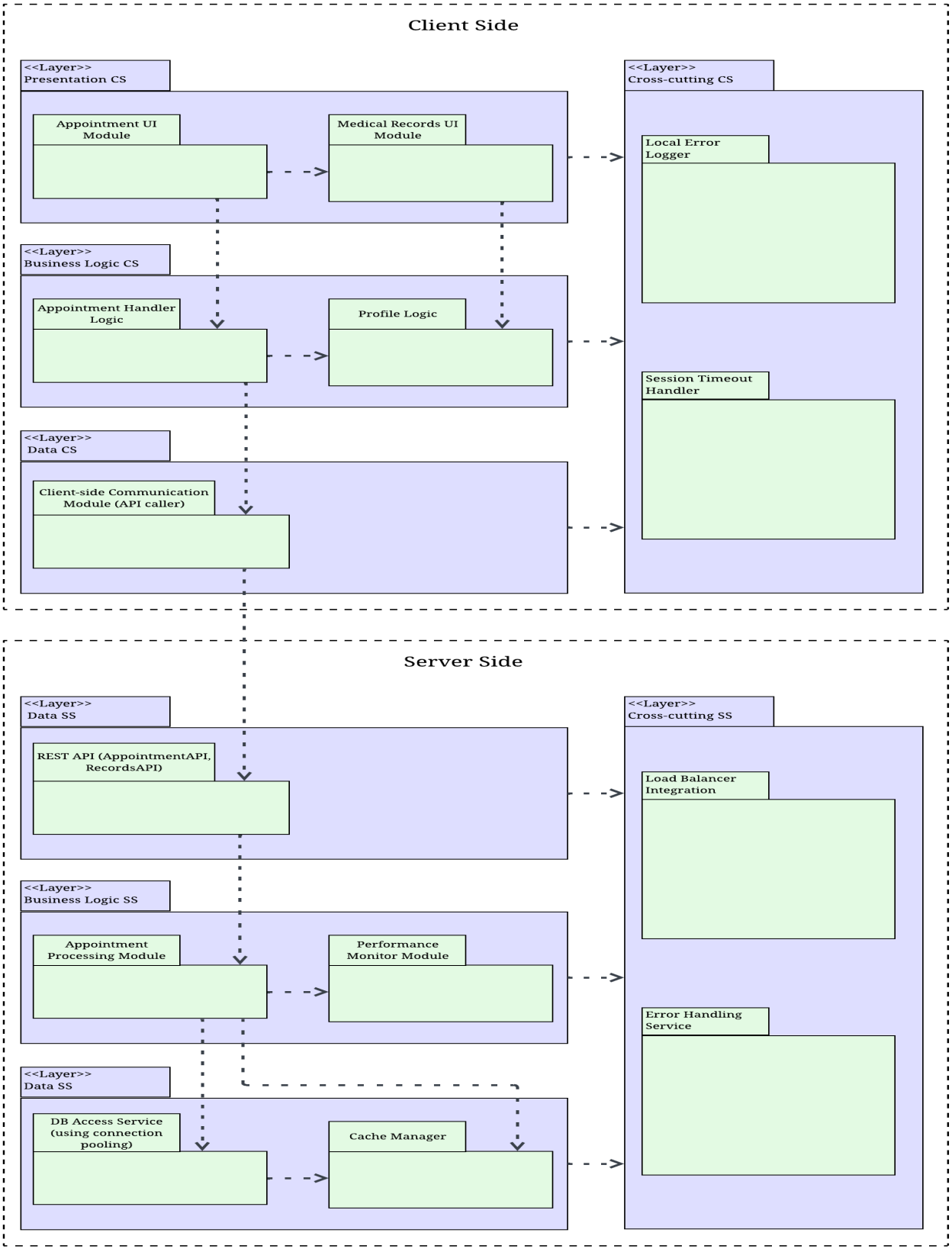
Design Decision and Location	Rationale
Implement a load balancer at the infrastructure layer	Distributes incoming appointment and patient data requests across multiple servers, ensuring fast response times and preventing bottlenecks (QA-5).
Introduce caching mechanisms for patient history in the service layer	Temporarily stores frequently accessed medical data to reduce database queries, enabling fast retrieval under high concurrency (QA-5).
Use a cloud-hosted relational database with autoscaling	Supports high-speed read/write access and automatically scales with the load, reducing latency during peak hours (QA-2, QA-5).
Separate service modules for appointment handling and record retrieval	Decomposes the logic layer to isolate high-traffic modules and avoid processing delays; enhances modularity and load handling (QA-5, CRN-3).
Remove local data storage in the client interface	All operations are handled by server-side through secure API calls; ensures centralized control and consistent performance regardless of client hardware (CON-2).
Use asynchronous messaging for queued operations	Non-blocking task queues for updates and notifications allow faster front-end interaction, ensuring the UI remains responsive (QA-5, QA-1).

This step establishes the key technical structures that underpin the system's ability to meet performance requirements. These components will be refined further in subsequent iterations when exact interface definitions are developed.

Step 6: Sketch Views and Record Design Decisions

The architecture of the Clinic Management System is structured based on a modular, layered approach to support scalability, maintainability, and performance. The following diagram and its accompanying table illustrate the key components of both the client and server sides, organized by architectural layers and refined according to the decisions made in previous steps.

Architecture View: Layered Module Diagram



Summary of Key Components and Responsibilities

Component	Layer	Responsibility
Appointment UI Module	Client - Presentation	Allows receptionist to search and book appointments.
Medical Records UI Module	Client - Presentation	Enables viewing of patient history.
Business Modules (CS/SS)	Business Logic	Handles rules for appointment eligibility, updates, and record fetch.
Caching Service (New)	Cross-cutting (SS)	Caches patient history and common queries to improve speed (QA-5).
Load Balancer Integration	Cross-cutting (Infra)	Distributes requests among servers for performance under high load (QA-5).
Error Handling Module	Cross-cutting (SS)	Ensures system resilience under failed requests.
Database Connector	Data Layer (SS)	Handles secure, real-time queries to the medical records and appointment tables.
Service Interfaces (REST APIs)	Server - Presentation	Offers secure, fast, and stateless interaction between client and server.

Deployment Diagram and Architectural Responsibilities

To complement the module view, the following deployment diagram illustrates the physical distribution of components in the Clinic Management System (CMS). This provides insight into where system parts reside and how they interact during operation.

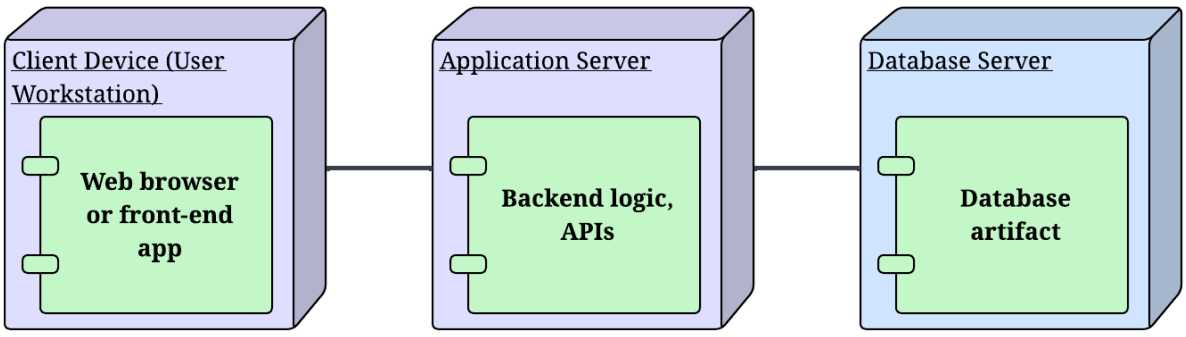
Deployment Diagram Summary

Element	Responsibility
User Workstation	Represents the user's device (PC, tablet, smartphone). Hosts the client-side logic for scheduling, viewing medical records, and interacting with the UI.
Application Server	Hosts the server-side logic, including business logic, RESTful APIs, and middleware modules. It acts as the core processing unit of the system.
Database Server	Maintains a relational database storing all system data, including appointment schedules, patient records, billing, and logs. Ensures data persistence and recovery.

Relationships Between Components

Relationship	Description
Between Application Server and Database Server	Communicates via JDBC protocol to retrieve, update, and delete data securely and efficiently from the relational database.
Between Client and Application Server	Uses HTTPS (REST API) for secure communication between the frontend modules and backend services.

Deployment Diagram



Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

The following table summarizes the evaluation of design progress using a Kanban-style approach:

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		CON-1	Ensured compatibility across Mac, Windows, and Linux using a standard web-based reference architecture (HTML, CSS, JS).
		CON-6	Enabled real-time data updates via cloud and three-tier architecture to boost performance and responsiveness.
		CRN-3	Developed a responsive booking module with real-time checks to prevent overlaps and enhance user experience.
	QA-1		Applied caching and layered structure to optimize responsiveness, though additional load testing is required.
	QA-3		Introduced cloud hosting to improve uptime, but redundancy and automatic failovers need further implementation.
QA-4			No accessibility features like tooltips, screen reader support, or UI feedback were addressed this iteration.
QA-5			No plans for plugin support or update management mechanisms were included in this phase.
CON-5			Performance analytics and error trend logging were not included in this stage of the architecture.

Summary:

This iteration laid a strong architectural foundation for performance-driven requirements, particularly in responsiveness, concurrency, and real-time data access. However, enhancements in usability, maintainability, and operational resilience are deferred to future iterations.

ADD Steps for Iteration 2: Identifying Structures to Support Primary Functionality

Step 1: Review Inputs

This iteration refines the architecture by detailing modules that support core clinic functionalities. It considers quality attributes such as availability, security, scalability, and testability. Drivers include the following use cases:

- UC-1: Add appointment
- UC-13: Registry
- UC-18: Pay bills

The iteration also accounts for:

- QA-2 (Availability), QA-3 (Security), QA-5 (Performance), QA-6 (Maintainability)
- Constraints: CON-1, CON-2, CON-4, CON-5
- Architectural Concerns: CRN-1, CRN-2, CRN-3

Step 2: Establish Iteration Goal by Selecting Drivers

This is the second iteration in the design of a greenfield system. The goal is to define detailed modules for the system's primary functions, enable distributed development, align with sprints, and address scalability, security, and maintainability.

Primary Use Cases:

- **UC-1: Add appointment**
- **UC-13: Registry**
- **UC-18: Pay bills**

Step 3: Choose One or More Elements of the System to Refine

This step focuses on refining the **Clinic Management System's** structure within each layer. Components such as Patient, Doctor, Appointment, and MedicalRecord are decomposed across layers to support implementation and parallel development.

Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

This step identifies architectural patterns, tactics, and design concepts that address the selected drivers. The goal is to define the structures needed to support implementation while satisfying quality attributes. The following table summarizes the selection of design decisions:

Design Decisions and Location	Rationale and Assumptions
Create a Domain Model for the application	Establishes structure for core clinic entities such as **User, Appointment, MedicalRecord,** etc. before diving into implementation. Clarifies relationships and avoids design chaos.
Identify Domain Objects that map to functional requirements	Each system function (e.g., login, appointment booking, medical access) is represented by a dedicated object. Prevents missed interactions like validation, conflict resolution, or role-based access.
Decompose Domain Objects into general and specialized Components	Breaks each domain object into smaller modules across layers (UI, Logic, Data). Enhances modularity, simplifies testing, and supports distributed team development.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

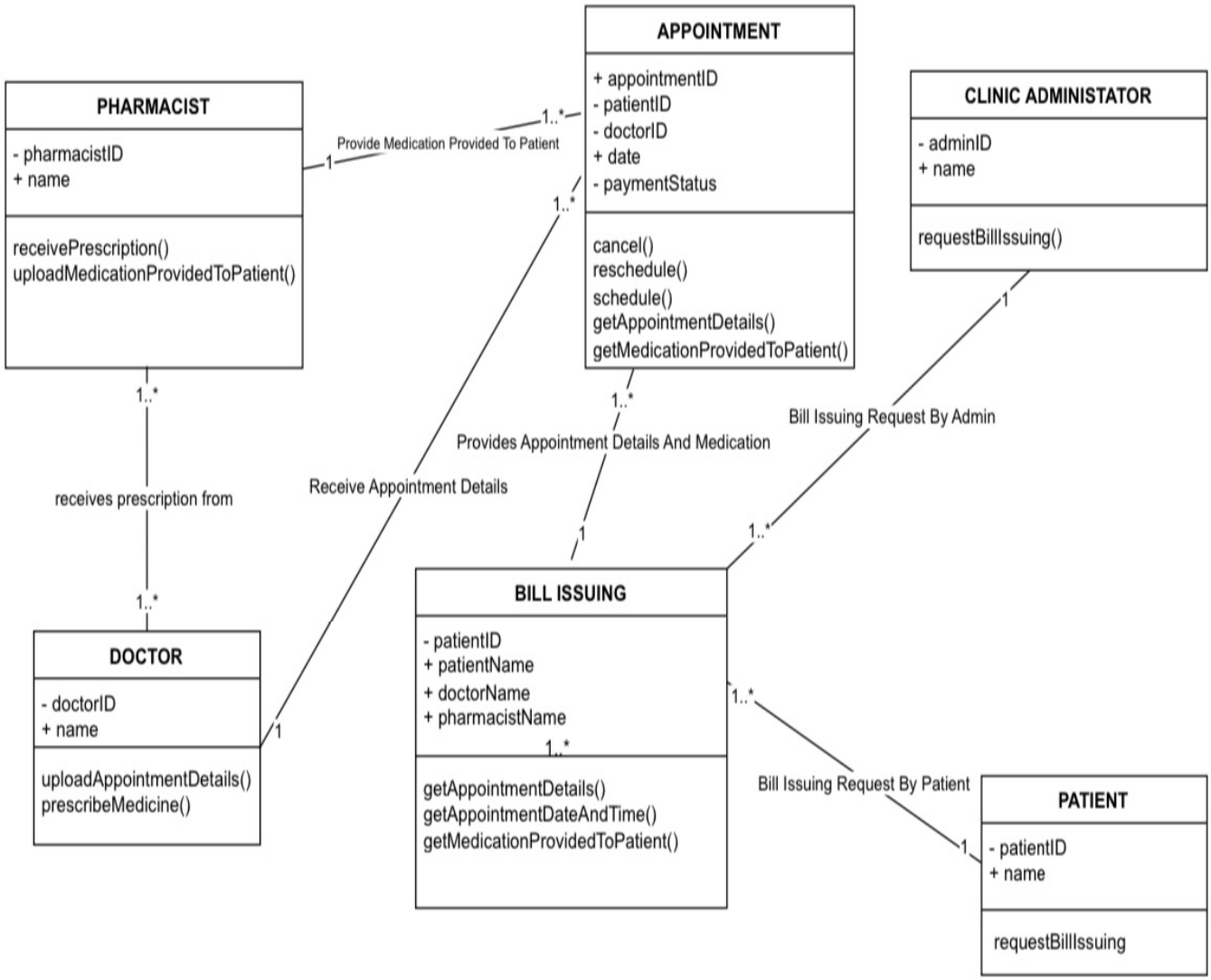
The following table provides an overview of the basic design choices taken during this iteration:

Design Decisions and Location	Rationale and Assumptions
Map use cases to high-level functional modules	Use cases (UC-1, UC-3, UC-6) guide the creation of functional modules like AuthenticationManager, SearchEngine, and ReservationManager. This ensures architectural alignment with requirements and supports modular development.
Design modular responsibilities across layers	Modules are distributed across UI, Business Logic, and Data layers to promote separation of concerns. For example, SearchPage connects to AvailabilityChecker and AppointmentDatabase, enabling parallel work across teams.
Define preliminary interface contracts between modules	Interface boundaries are defined early between modules (e.g., LoginPage → AuthenticationManager) to reduce coupling, clarify integration, and allow for easier implementation handoffs.
Address unit testability (CRN-4)	Modules in the logic layer are designed with testability in mind, using mockable dependencies (e.g., UserRepository) to enable isolated testing, in support of CRN-4.

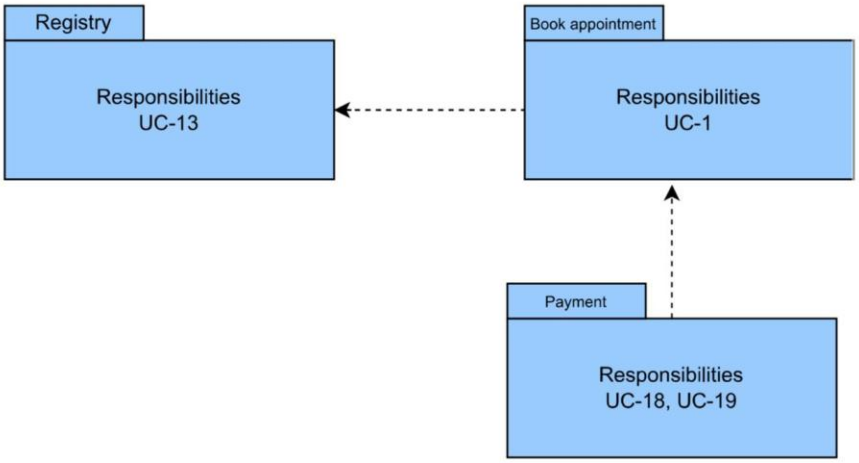
Step 6: Sketch Views and Record Design Decisions

Domain Model

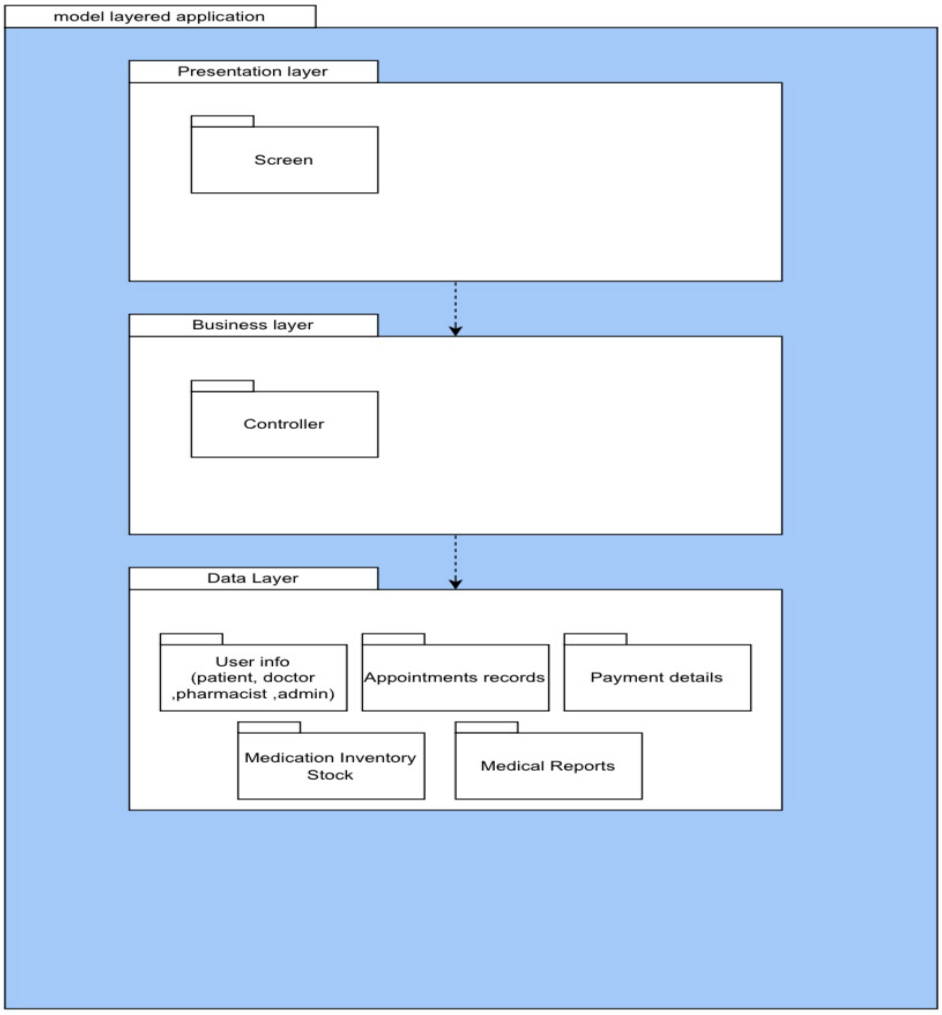
The figure below displays a domain model for a clinic management system based on the primary use cases. Adjustments have been made to the domain model to improve modularity and scalability. Entities of this domain model are doctor, patient, pharmacist, clinic administrator, appointment, bill issuing.


APPOINTMENT
+ appointmentID
- patientID
- doctorID
+ date
- paymentStatus
cancel()
reschedule()
schedule()
getAppointmentDetails()
getMedicationProvidedToPatient()
PHARMACIST
- pharmacistID
+ name
receivePrescription()
uploadMedicationProvidedToPatient()
CLINIC ADMINISTATOR
- adminID
+ name
requestBillIssuing()
Provide Medication Provided To Patient
1..*
1
1..*
1..*
receives prescription from
1..*
Provides Appointment Details And Medication
Receive Appointment Details
Bill Issuing Request By Admin
1
1..*
DOCTOR
- doctorID
+ name
uploadAppointmentDetails()
prescribeMedicine()
1
BILL ISSUING
- patientID
+ patientName
+ doctorName
+ pharmacistName
1..*
getAppointmentDetails()
getAppointmentDateAndTime()
getMedicationProvidedToPatient()
1..*
Bill Issuing Request By Patient
1
PATIENT
- patientID
+ name
requestBillIssuing

Instantiated Domain Objects for Use Case model

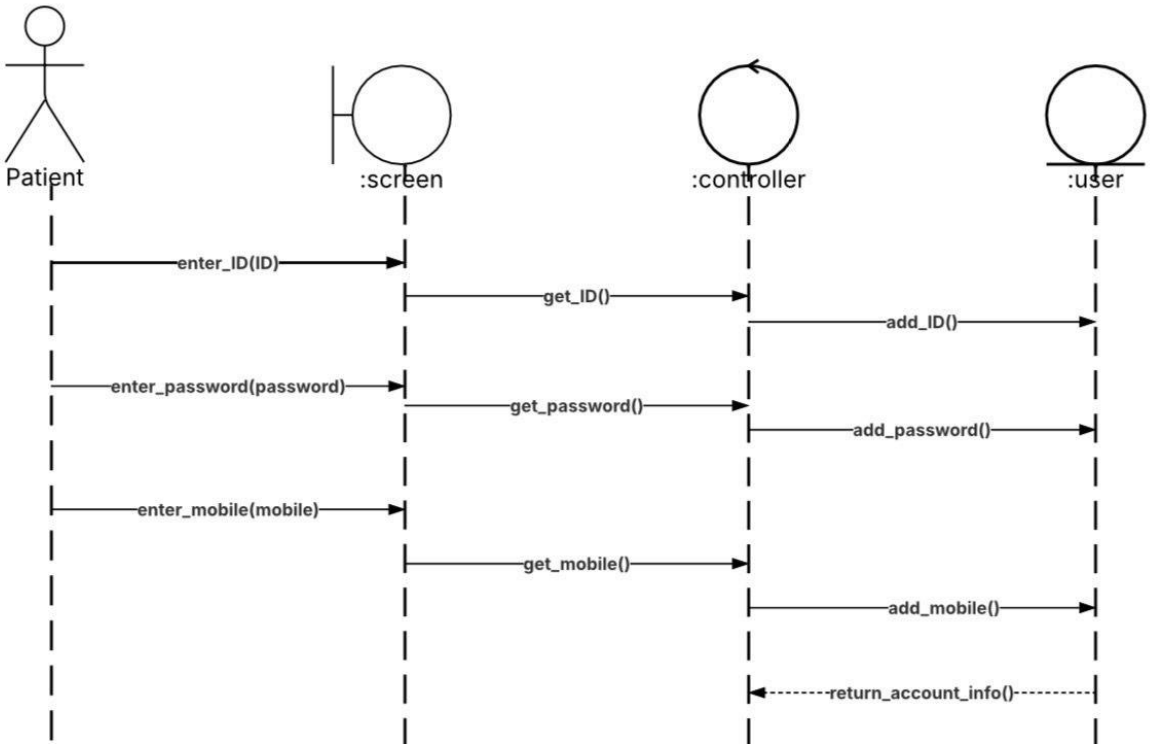


Sketch of module view



Responsibilities of identified elements:

Element	Responsibility
Screen	Displays an accurate representation of user activity, and is regularly updated upon each event
Controller	Renders necessary details to presentation layer for displaying network representation
User info	Details regarding users are stored such as phone number and email address
Payment details	Holds each payment transaction's details and status
Appointment records	Has information regarding every appointment booked (patient, doctor, diagnosis, prescription...,)
Medication inventory stock	Tracked medication inventory information are saved here
Medical reports	All medical reports of all patients are sorted by date and saved for reference



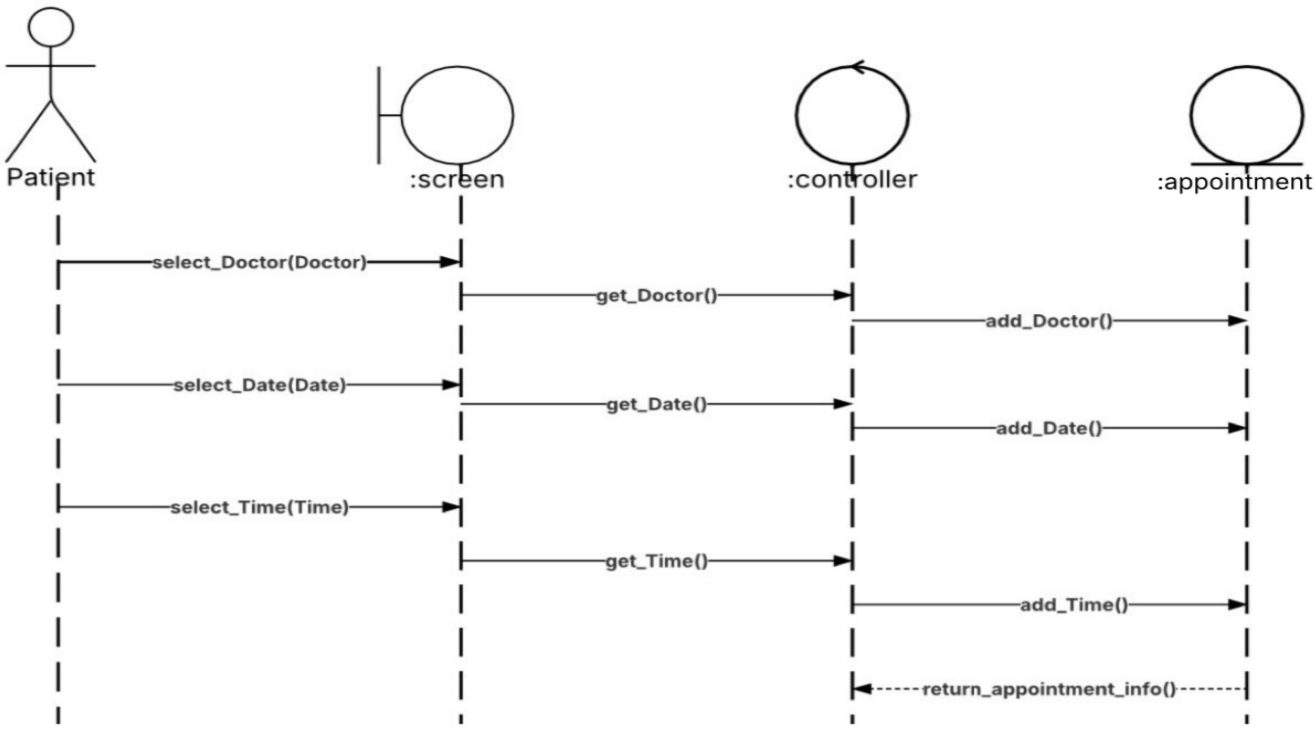
Sequence diagram for UC-13: Registry

- Sequence diagram for UC-13 (Key UML):
- Add appointment scenario:
  1. Sara enters her ID, password, and mobile
  2. The system retrieves the ID, password, and mobile to proceed.
  3. The system adds the ID, password, and mobile to user info.
  4. Once the details are successfully added, the system will return account details.

Method Name	Description
Screen	
enter_ID(ID)	Allows user to enter his/her ID
enter_password(password)	Allows user to enter his/her password
Enter_mobile(monile)	Allows user to enter his/her mobile
Controller	
get_ID ()	System retrieves ID entered
get_password ()	System retrieves password entered
get_mobile ()	System retrieves mobile entered

return_account_info ()	System displays to user's screen all information retrieved from him/her
User	
add_ID ()	System adds ID to user database
add_password ()	System adds password to user database
add_mobile ()	System adds mobile to user database

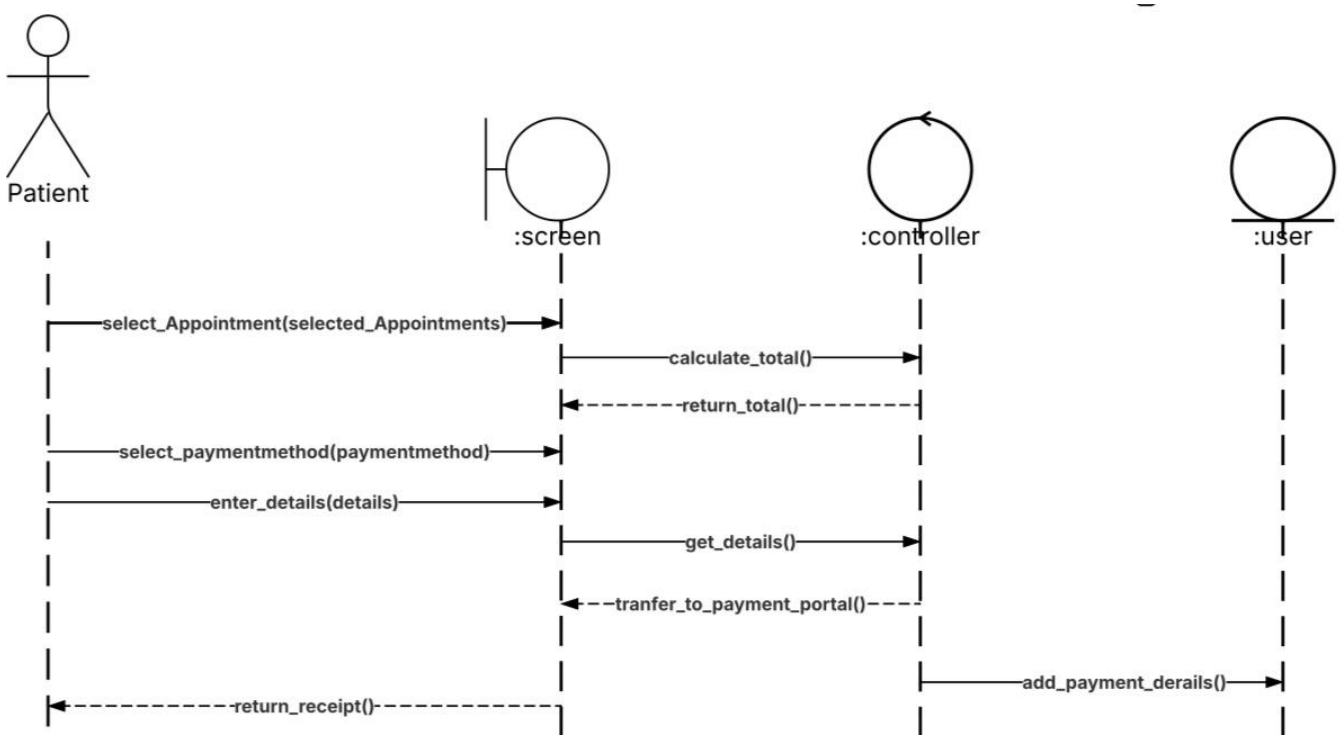
Sequence diagram for UC-1: Add appointment



- Sequence diagram for UC-13 (Key UML):
- Add appointment scenario:
    1. Sara selects her preferred doctor, date and time for an appointment.
    2. The system retrieves appointment details.
    3. The system adds an appointment at the specified date and time with the specified doctor.
    4. Once the appointment is successfully added, the system will return appointment details.

Method Name	Description
Screen	
select_Doctor (Doctor)	Allows user to select preferred doctor
select_Date (Date)	Allows user to select preferred date
select_Time (Time)	Allows user to select preferred time
Controller	
get_Doctor ()	System retrieves selected Doctor
get_Date ()	System retrieves selected Date
get_Time ()	System retrieves selected Time
return_appointment_info ()	System displays to user's screen all appointment details
Appointment	
add_Doctor ()	System appointment doctor to appointment database
add_Date ()	System adds appointment date to appointment database
add_Time ()	System adds appointment time to appointment database

Sequence diagram for UC-18: Payment



- Sequence diagram for UC-13 (Key UML):
- Payment scenario:
    1. Sara selects the appointments (one or more) she wants to pay for.
    2. The system calculates the total amount and returns it to Sara's screen in the local currency.
    3. Sara selects a payment method from these options: Apple Pay, Benefit, Credit/Debit card.
    4. Sara enters the required information, then the system transfers her to the payment portal to finish the transaction.
    5. The system adds this transaction to its payment database.

Method Name	Description
Screen	
select_Appointment(selected_Appointments)	Allows user to select appointment(s) to pay for
select_paymentmethod(paymentmethod)	Allows user to select preferred payment method
enter_details(details)	Allows user to enter additional details regarding payment
return_total ()	System displays total amount on user's screen
return_receipt ()	System display receipt on user's screen
transfer_to_payment_portal ()	User is transferred to payment portal page to complete payment process
Controller	
calculate_total ()	System adds up selected appointments prices to get a total
get_details ()	System retrieves extra details added by user
Payment	
add_payment_details ()	System adds this payment transaction details to payment database

Step 7: Perform Analysis of Current Design and Review Iteration Goal

Unaddressed	Partially addressed	Completely Addressed	Design decision made during the iteration
		UC-13 (Registry)	Modules across all layers were identified
		UC-1 (add appointment)	Modules across all layers were identified
		UC-18 (pay bill)	Modules across all layers were identified
	CON-3		All data is stored by the system in different databases (user, appointment, payment)
QA-3 (security)			No major decisions were made to improve security
QA-5 (performance)			No major decisions were made to improve performance
QA-2 (availability)			No major decisions were made to improve availability