D4: Detailed Design

Project: Healthcare Appointment Booking System

Project members:

Vishali Chhabra, Angie Wang, Troy Johnson , Cordelle WaldenCameron

Project repository: https://github.com/angie-W-auemail/groupproj_3506.git

1. Introduction

- **Objective:**

This project is intended to build an online interactive healthcare appointment booking system supporting the organization's daily business activities and generate records and data storage.The system has 3 types of intended end users (admin/doctors/patients) across all platforms with different id access for their granted permissions. The admin staff will have highest permission to manage users, grant permissions, daily schedules, pricing, track records, and generate reports. The doctors will have permissions to manage personal profile and schedules, track and generate records, provide feedback/recommendations to patients. Patients will be able to view schedule availability and book appointments , view visit history and treatment progress,make payments and receive medical reports, and update personal details and manage account settings.

- **Scope:**

The scope of this system covers functionalities such as user authentication, role based access control, user management, appointment schedules, display user information, generate reports, external payment gateway integration, it will not allow cancellation and refund online, and patients will have to contact the organization if desired.

- **Assumptions:**

-The system assumes the availability of a stable internet connection for real-time updates.

-The system assumes that error message popup to attempting of double booking

-The system assumes on good practice from patients where they show up timely

-The system assumes best practice by doctors input new entry of comments and medical record at every new visit from a patient

-The system assumes payment up-front, patients will have to pay the amount due before successfully booking an appointment, the system does not allow cancellation and refund online, and patients will have to contact the organization for such matters.

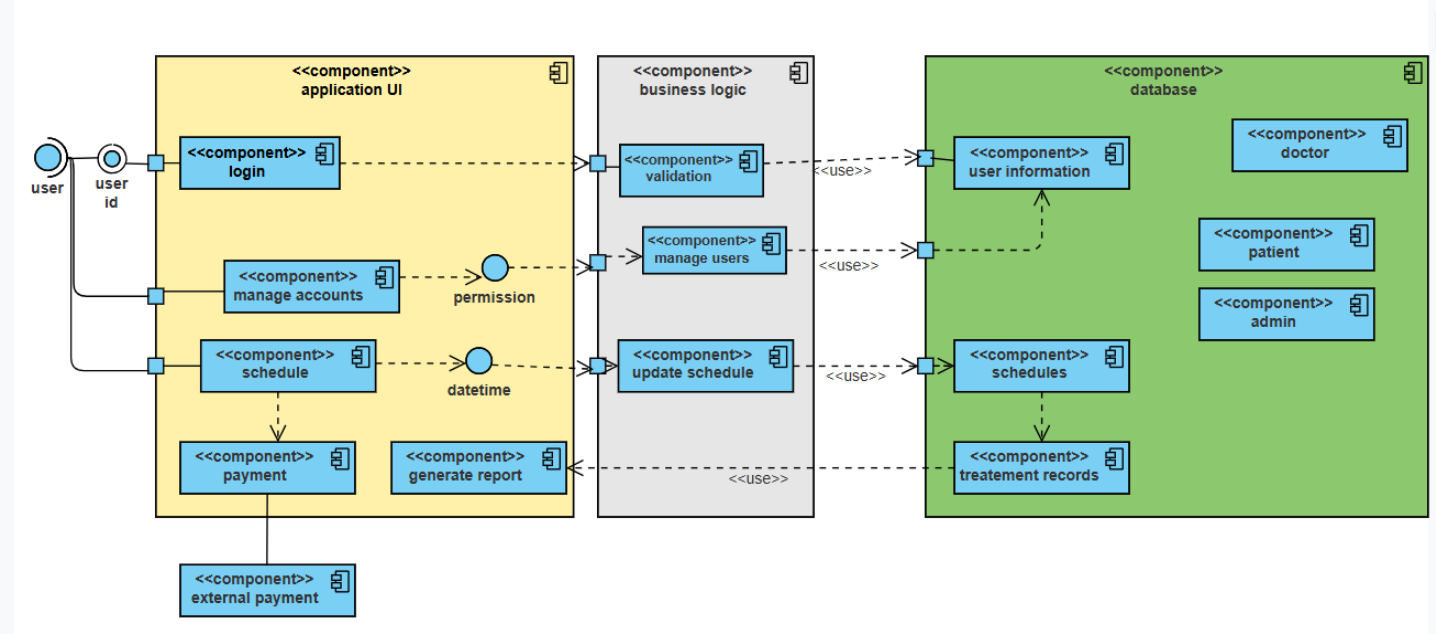
-The user id and password is assigned at user creation when HR set up account, not allowed for change

2. System Architecture Design

- **Architectural Overview:**

The system follows a three tier architecture with the front end layer (user interface), business logic layer (application services), and data layer (database)

UML Diagram - Component Diagram:



Components: application UI, login, schedule appointments, payment, generate report, services, account validations, manage users, update schedule, database, user information, patient information, doctor information, admin information, schedule records, treatment records, external payment

Connectors: user_id, user_password, user_permission, appointment datetime

External Interfaces: users (patients/admin/doctors)

3. Detailed Module Descriptions

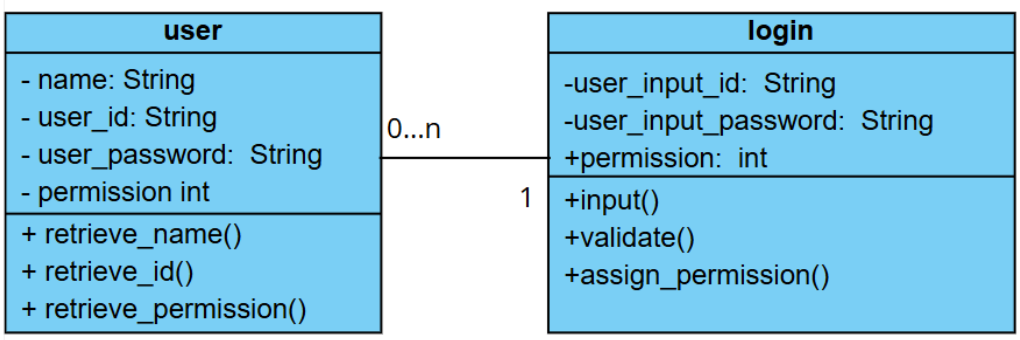
User authentication

Responsibilities: this module is responsible for user authentication while user input user id and password when logging in and retrieve assigned user permission for loading home page

Dependencies: retrieve from database table user recorded id and password for validation

Input: user_id (string), password (string) Output: permission (1/2/3 for patient/doctor/hr staff -1 for mismatched input)

UML Diagram–Class Diagrams:



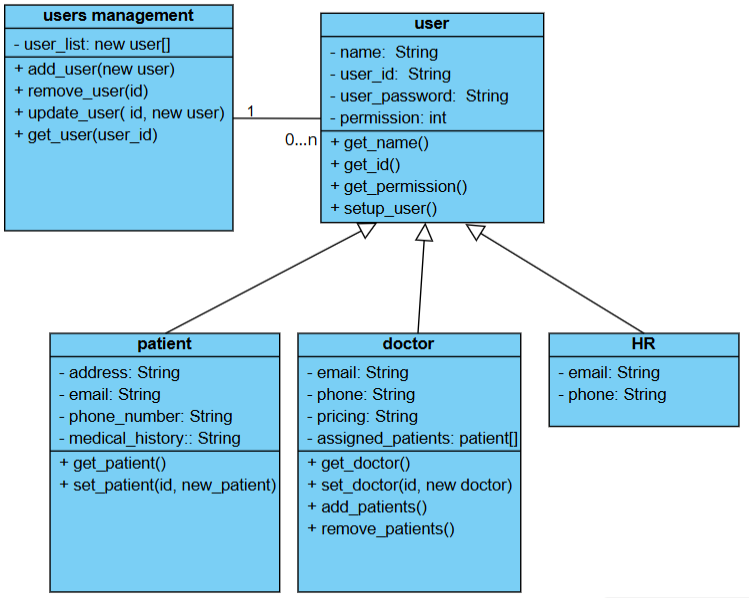
User Management

Responsibilities: this module is responsible for CRUD of user account information

Dependencies: this module is dependent on the SQL database for CRUD operations

Input: new user, user_id Output: database with new/updated account row or deleted account

UML Diagram–Class Diagrams:



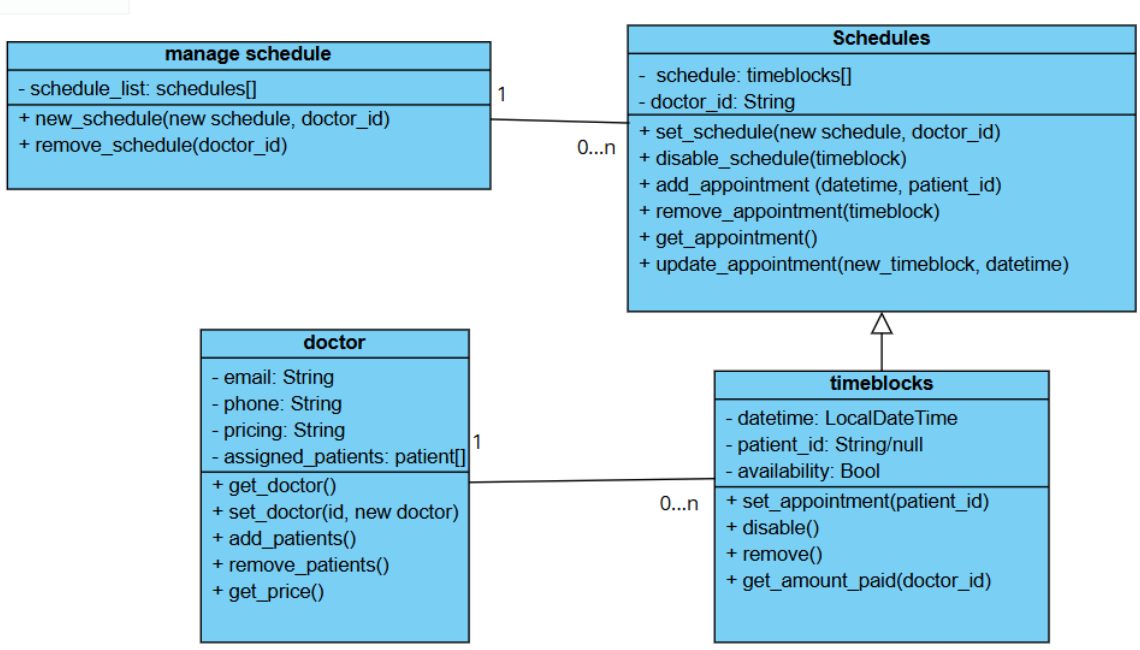
Appointment Schedules

Responsibilities: this module is responsible for appointment management, where patients book appointments and doctors/admin manage the schedule and record all into treatment records table

Dependencies: this module is dependent on granted permission

Input: datetime-hour block Output: updated schedule time series data

UML Diagram–Class Diagrams:



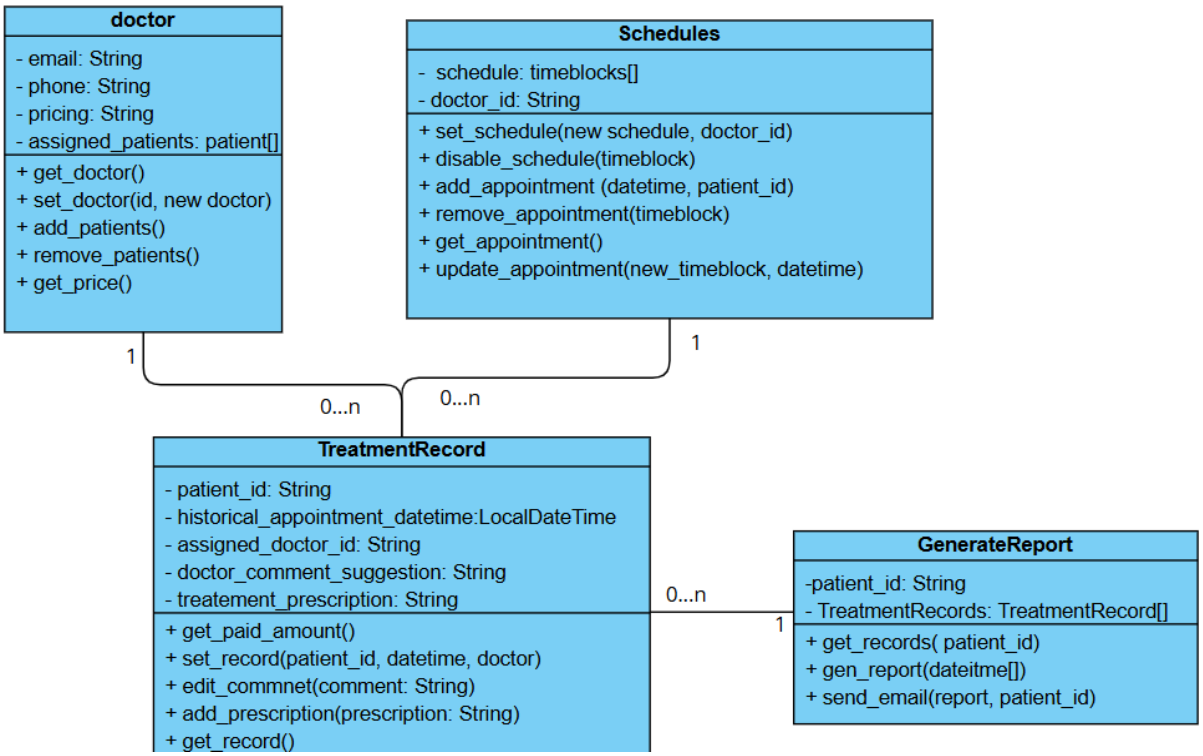
Generate reports

Responsibilities: this module is responsible for retrieving patient historical visits from treatment records, update/insert doctor comments and send email to requested patient if triggered

Dependencies: this module is dependent on scheduled appointments, records will receive new entry after patient visit

Input: patient_id Output: updated treatment records/treatment records table

UML Diagram–Class Diagrams:



External payment gateway integration

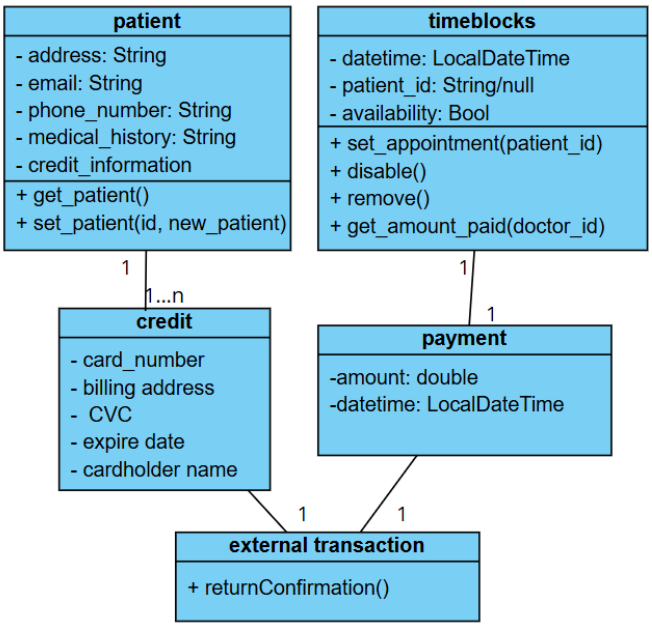
Responsibilities: this module is responsible for sending payment request to external payment gateway and receive confirmation to proceed on appointment booking

Dependencies: this module is dependent on scheduled appointments, amount payable is determined if patient attempt to book appointment

Input: patient_id, amount_payable, datetime(now), credit_information

Output: payment validation, appointment datetime

UML Diagram–Class Diagrams:



4. Database Design

User (User_ID, registered password, permission(Patient/Doctor/HR))

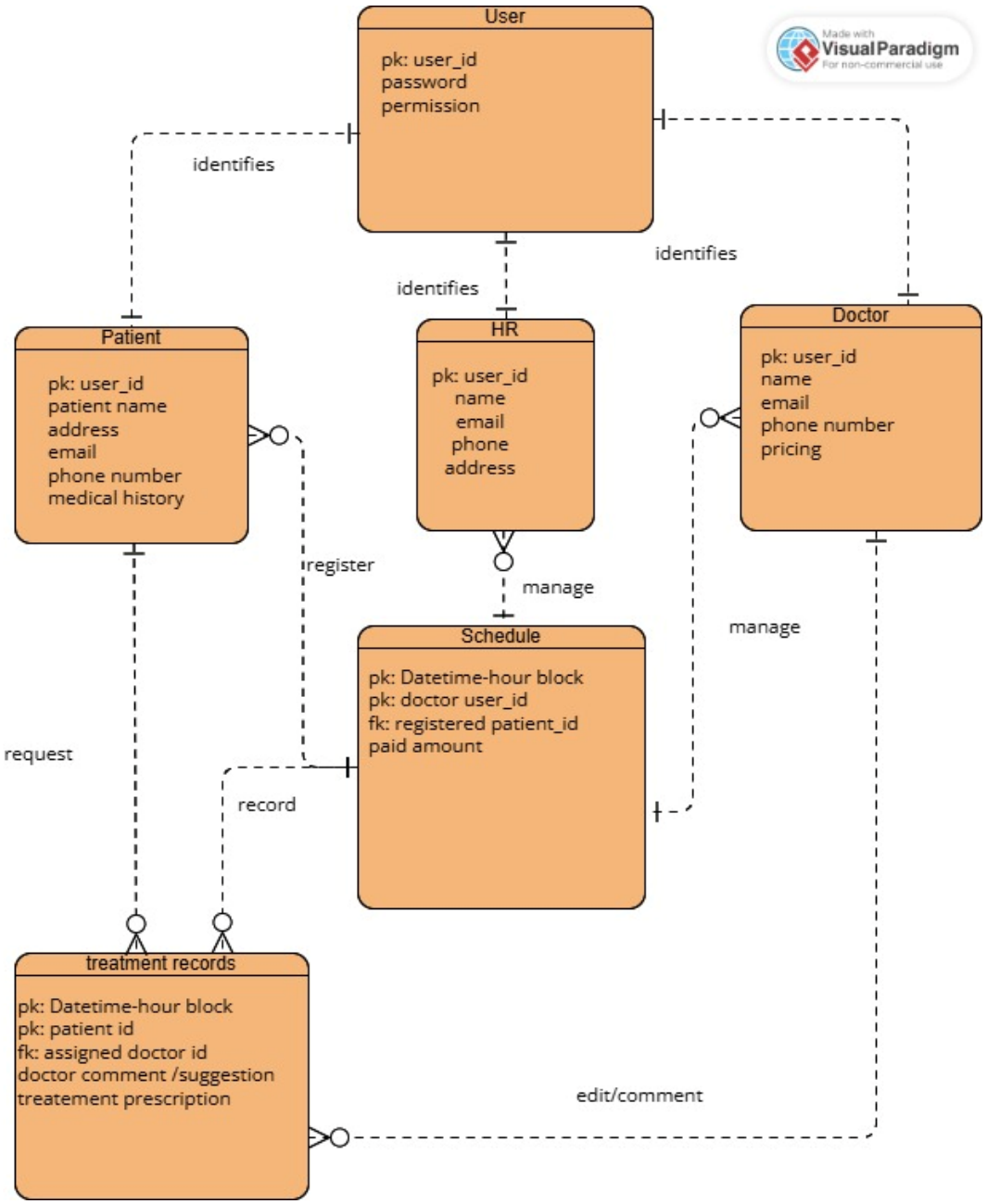
Patient (User_ID, patient name, address, email, phone number, medical history)

Doctor (User_ID, doctor name, email, pricing, business phone number)

HR ((User_ID, name, email, phone number, address)

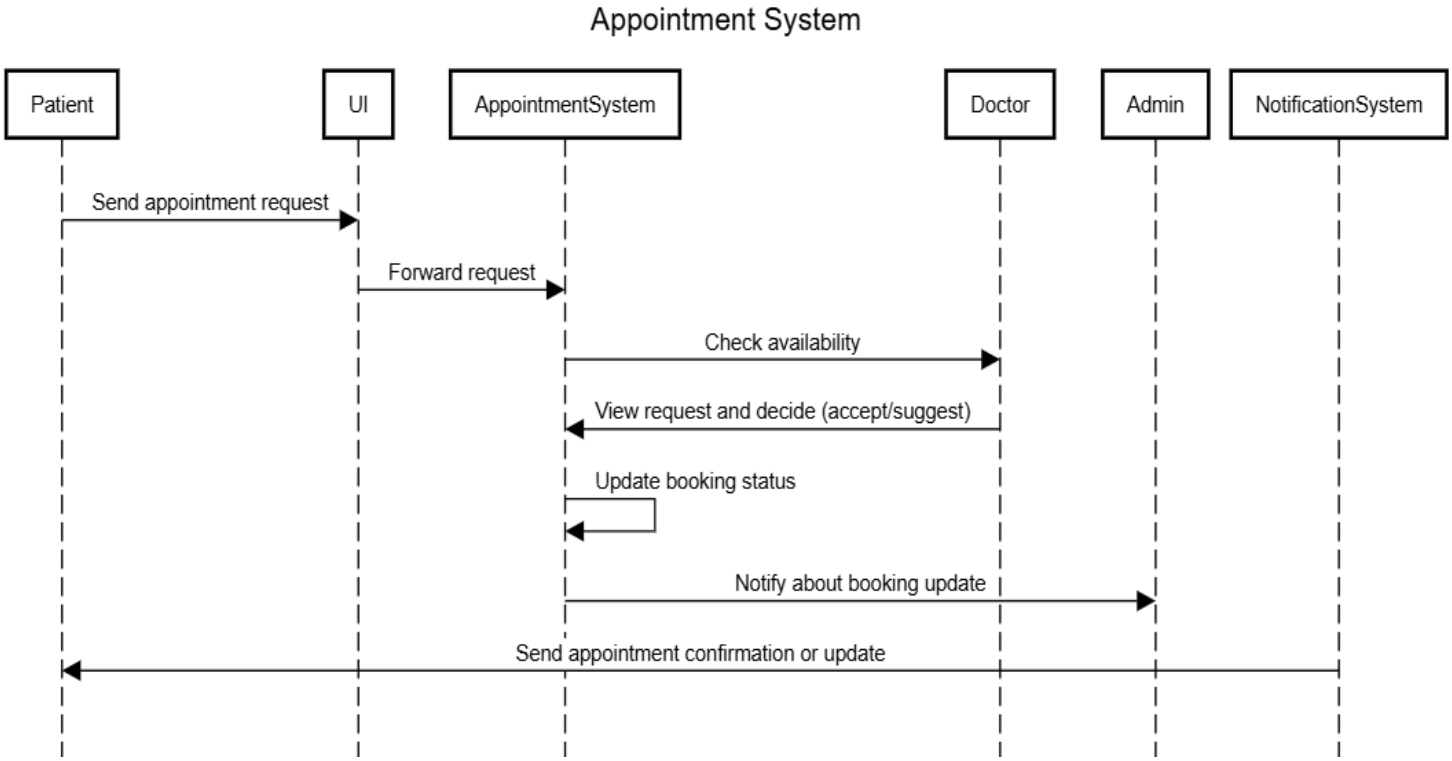
Schedule (DateTime-hour block, doctor_id, registered patient id, paid amount)

Treatments (DateTime-hour block, patient id, doctor id, doctor's comment/suggestion, treatment prescription)



5. System Flow Design

- **Sequence Diagrams:**



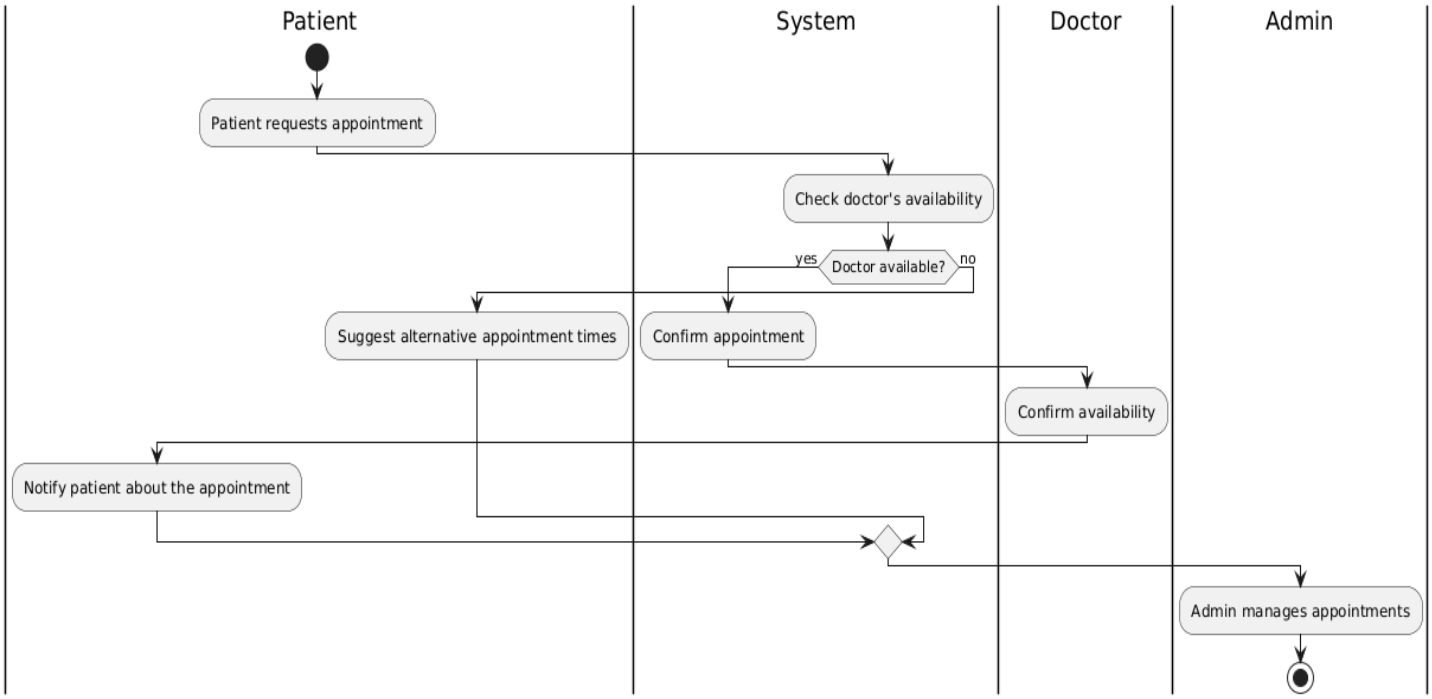
- **Actors -** Patients, Admins, doctors, appointment system and the notification system,
- **Messages**

1. Patients send appointment requests to the UI.
2. The UI then sends the request to the appointment system.
3. The appointment system then checks the availability with the doctor.
4. The doctor then views the request and then accepts or makes another suggestive date.
5. The appointment system then updates the booking and then notify the admin.
6. The final step is where the notification system then sends a message to the patient with the booking information.

Diagram representation of the system

1. The patient interacts with the UI to fill out the appointment request.
2. The UI then sends the request to the backend appointment system.
3. The appointment system then query with the doctor to check for availability.
4. The doctor then responds with the availability status and if available is there he will accept if not he will suggest otherwise.
5. The notification system then notifies the admin about the new appointment.
6. Then the notification system sends the confirmation message to the client or patient.

- **Activity Diagrams:**



- **Activities**

Patient request appointment

The system checks for the doctor's availability.

Confirming appointment

Notifying the patient about the appointment.

Admin manages appointments.

- **Decisions**

    If doctors available it confirms request

    If not, doctors suggest alternatives.

- **Synchronization Bars**

Initial request

This is the process when the patient requests an appointment.

Checking availability

A synchronized bar occurs before checking doctor availability. The system is processing requests and then determining availability.

Parallel activities

Once availability is confirmed two activities is then performed:

The doctor will then confirm availability

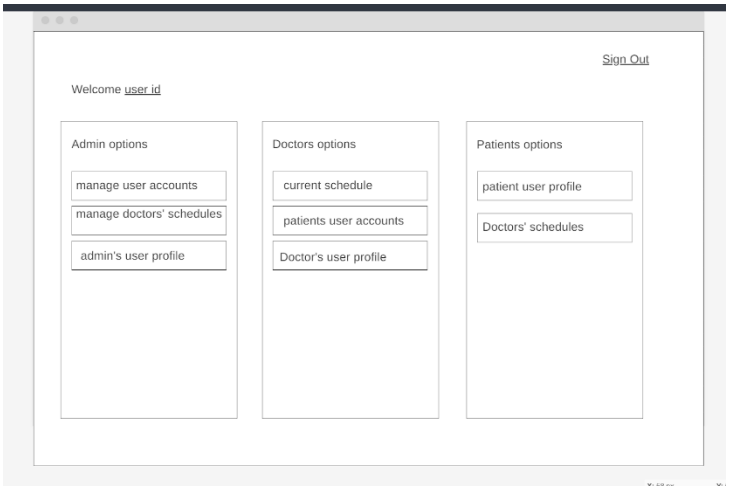
The patient will receive a notification about the appointment.

6. User Interface Design (Optional in Detailed Design)

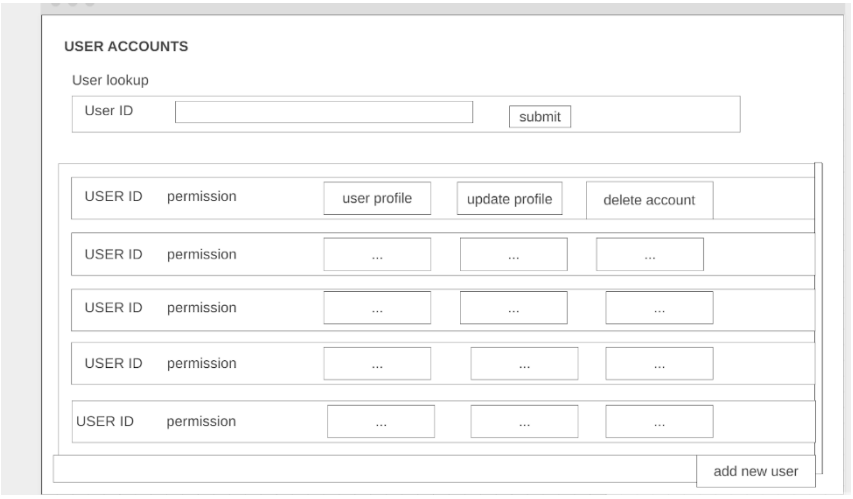
Wireframes:

Sign in Wireframe

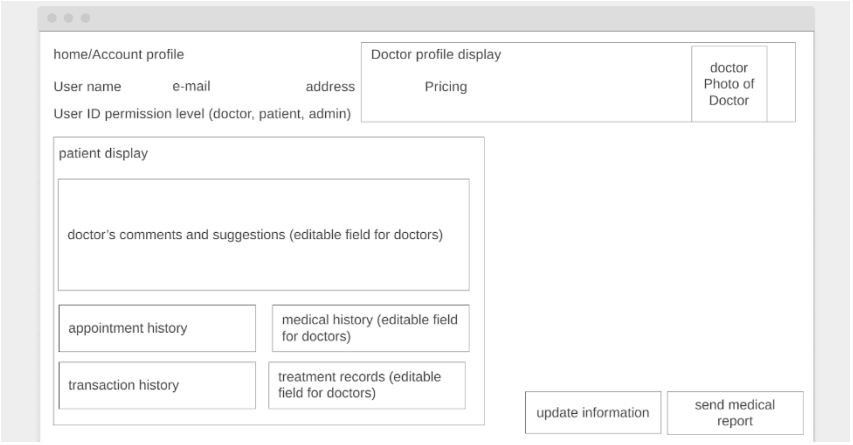
Homepage Wireframe



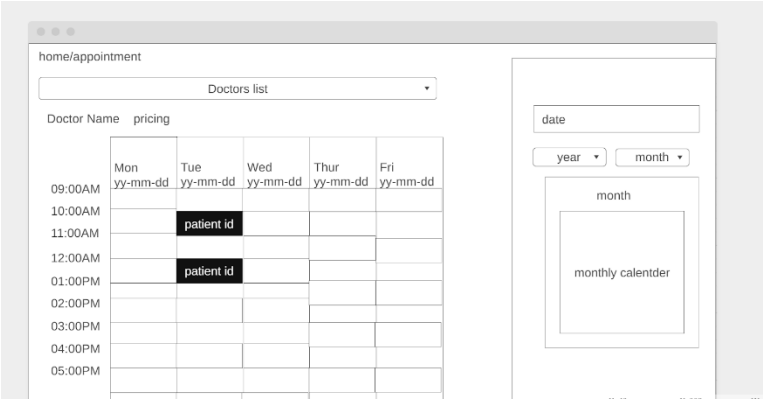
Users account page



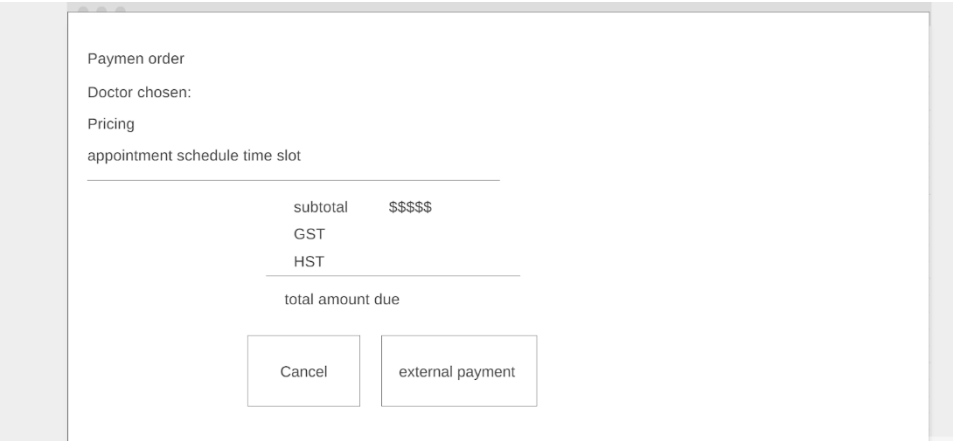
User profile page



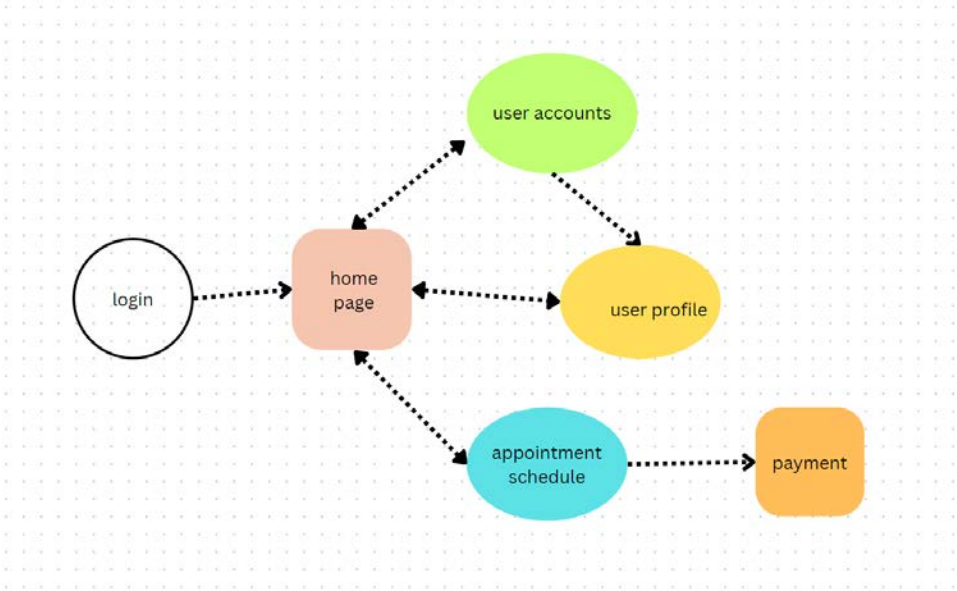
Appointment



Payment page

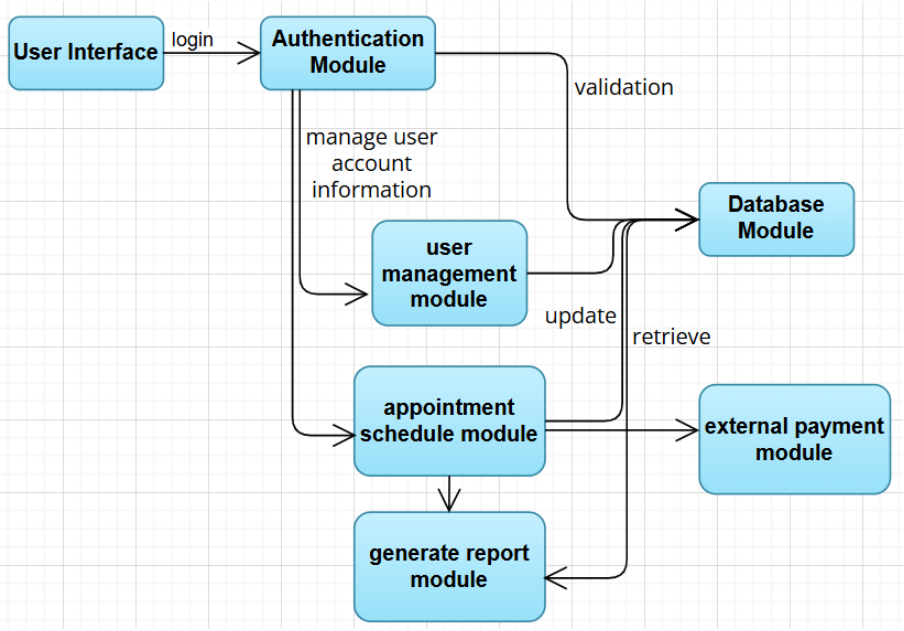


Navigation Flow:



7. Interaction Between Components

- Collaboration Diagrams (Communication Diagrams):



8. Non-Functional Requirements

- **Performance Considerations:**

The system should provide response in approximately two seconds ro be considered an efficient system.

The system should be able to handle 100 concurrent users requests without a degradation in the system.

The system should be able to handle high load and still be able to perform efficiently and also be able to still book, search and notification should be processed efficiently.

- **Security Measures:**

1. Information such as patient information, and appointment details should be all encrypted using formats such as TLS and so on.
2. Multi factor authentication is a good method to ensure that patient and healthcare providers have secure access to this system.
3. The use of access control to ensure that patients, admins and healthcare providers all have different access and different privileges.
4. The implementation of an audit logging will allow admin to monitor users for suspicious activities and also allow and also ensure that users are complying with the healthcare regulations.
5. Implementing a vulnerability management system will help to constantly check for vulnerability and loopholes in the system to prevent breach from outsiders.

- **Scalability:**

1. This system should support horizontal scalability, having the ability to add more servers, to handle increased load in the future.
2. Having a system that distributes load across multiple instances of the application to make sure that it has reliable performance.
3. Using an adequate database system such as a distributed or cloud based system that can grow as the number of users for the system grows.
4. Implementing a caching system for frequently accessing data to reduce the load on the system will also lead to a much faster response time.

- **Reliability:**

1. The system should have an uptime of about 99.9%, therefore will ensure availability to the user all the time.
2. Data redundancy- This will prevent loss of data during a downtime as it will have backup or replication of data such as patient information and so on.
3. Failover mechanism- This will ensure during a failure it will switch over to the backup system during a

failure to ensure continuity in the system.

4. Monitor alert system- This will ensure that it will scan the system for health, performance metrics and error rate, to make sure that the system is up to the optimal performance for the user.

9. System Interfaces

External Interfaces:

User Interface -

1. **Admin portal -** this is where they can manage users, appointment statistics and also configure system
2. **Patient portal -** This is the mobile or web interface that allows the patient to book an appointment.
3. **Healthcare portal -** This is the system that allows the doctor to manage their appointments, such as confirming or making suggestions for the patient.

API interface :

1. **Restful API** - This is where third party apps can interact with the system to check for availability for the patient and so on.
2. **Payment Gateway** - Allow a user or patient or the healthcare provider to accept payment with companies such as paypal, stripe or credit/debit card.
3. **Email/message notification system** - This allows patients to receive notification by email or message about their appointment with confirmation details.
4. **Health care provider system** - This is a system that allows the system to access patient information or records and also ensures doctors have relevant information about the booking.

Internal interfaces.

Module interface

1. This will have an appointment management module that will encompass the user interface, database module and notification service to manage all the functionality of the system.
2. The user module will manage all the user data, authentication and roles which will be the module interacting with the appointment module system.

Database interface

1. This will query requests received throughout the system for booking and also using CRUD admin or even the healthcare to approve, deny or other operation for the functionality of the system.
2. A caching system will also be implemented to save frequently accessed data for a faster and more efficient system.

Service interface

1. Notification service - this will do multiple stuffs such as providing reminders and sending appointment details for patients.
2. Another service could be a system that will analyze and check for performance between the user management and appointment module.

- **Internal Interfaces:**

10. Design Patterns (If Applicable)

- **Patterns Used:**

Model-View-Controller (MVC)

- **Justification:**

MVC is beneficial for separating concerns between the UI and business logic, making the system more modular and maintainable.

11.Detailed Algorithm Descriptions

Login:

If username and password are not empty:

Look up user in database

If user exists and password matches:

Approve login and open homepage

Else:

Notify of invalid username or password

Else:

Notify of empty username or password

Payment:

If payment method is valid:

Check if sufficient funds are available

If funds are sufficient:

Deduct payment amount

Confirm payment to customer

Else:

Notify customer of insufficient funds

Else:

Notify customer of invalid payment method

Appointment Booking:

If desired appointment slot is available:

Check patient eligibility

If customer is eligible:

Confirm appointment and notify customer

Else:

Notify patient of eligibility issues

Else:

Notify customer of unavailable appointment slot

Update Profile:

If user is logged in:

Validate new profile information

If information is valid:

Update profile with new details

Confirm update to the user

Else:

Inform the user of validation errors

Else:

Prompt the user to log in

12. Hardware and Software Requirements

- **Hardware Requirements:**

Desktop:

Screen Resolution: 1024x768 and up.

Browser Compatibility: Modern browsers like Chrome, Firefox, Safari, and Edge.

RAM: At least 4 GB.

Processor: Dual core 2.35ghz and up.

Storage: Minimum 256 GB.

Network: Reliable connection with at least 10 Mbps upload/download.

Mobile:

Screen Resolution: 360x640 and up.

Browser Compatibility: Mobile versions of Chrome, Safari, Firefox.

RAM: At least 2 GB7.

Processor: 32-bit or higher

- **Software Requirements:**

OS: Windows 7 and up, Mac os 11 and up, Android 8 and up, IOS 12 and up.

Server:

Operating System: Linux (Ubuntu 18.04+), Windows Server 2016 or later.

Web Server: Apache or Nginx.

Database: MySQL, PostgreSQL, or MongoDB.

Runtime Environment: Java, Node.js, or Python

13. Appendices

Glossary:

CRUD:create, read, update, and delete (CRUD), four basic operations of persistent storage

References:

Jama Software. (n.d.). *) PX TPO OVODUPOBMFRVJSFN FOUJN QBDUQSPEVDUEFWFMPQN FOU* **Jama Software.** https://www.jamasoftware.com/requirements-management-guide/writing-requirements/how-non-functional-requirements-impact-product-development