LOCAL HEALTH CENTRE MANAGEMENT SYSTEM

PHASE 2

PROJECT TEAM:

GLORIA ABINEZA – BSCNRB387723

MUSA KIPLANGAT BRIAN – BSCNRB592823

IAN MWANIKI KANYI – BSCNRB344123

CHRISTINE OBURE NYARESO – BSCNRB399923

SUBMITTED TO:

LECTURER CECILIA NANFUKA

1. **PROJECT DEFINITION**

This Phase 2 report presents the development and implementation of the Health Centre Management System (HCMS), created using Python. The project aimed to address the operational challenges faced by a local health centre. The primary goals included enhancing the efficiency of patient registration, appointment scheduling, and prescription management while improving the overall patient experience.

In this report, we will discuss the design considerations, including user interface design and the architectural patterns used in the system. Following that, we will present the implementation details, including pseudocode that outlines the system’s core functionalities. Finally, we will outline our comprehensive testing plan, detailing the various testing levels conducted to ensure the system’s reliability and usability.

LINK TO PROJECT REPOSITORY

<https://github.com/BSCNRB344123/SOFTWARE-ENGINEERING-GROUP-E/tree/main>

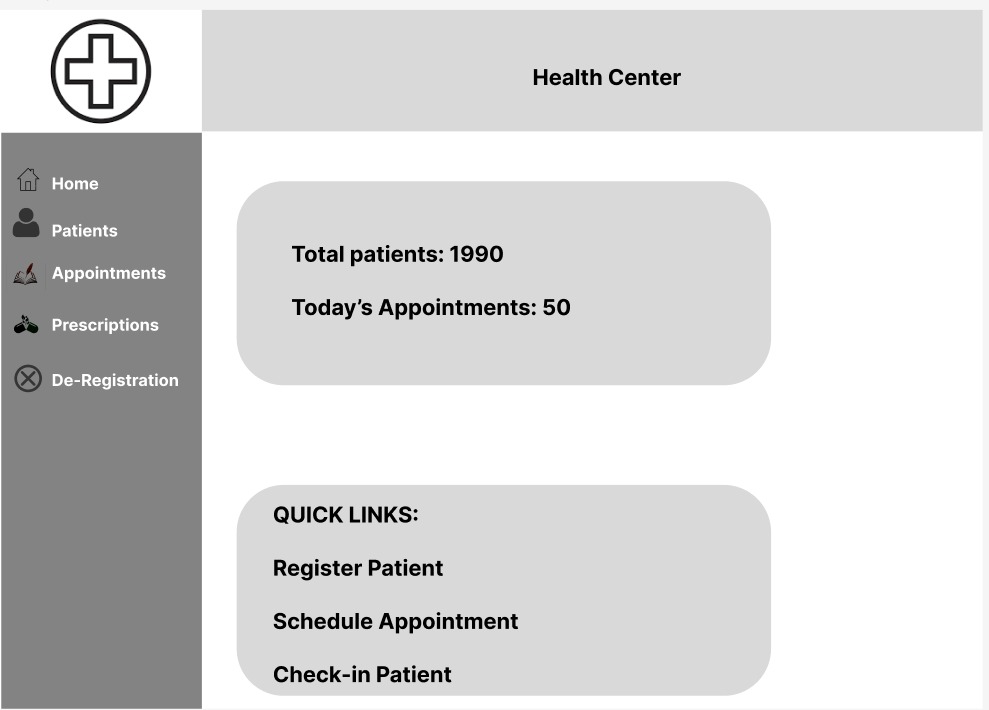
1. **SOFTWARE DESIGN CONSIDERATIONS**
   1. **UI DESIGN**

The user interface was crafted to ensure an intuitive and user-friendly experience for both patients and staff.

Key wireframes designed during the UI development phase:

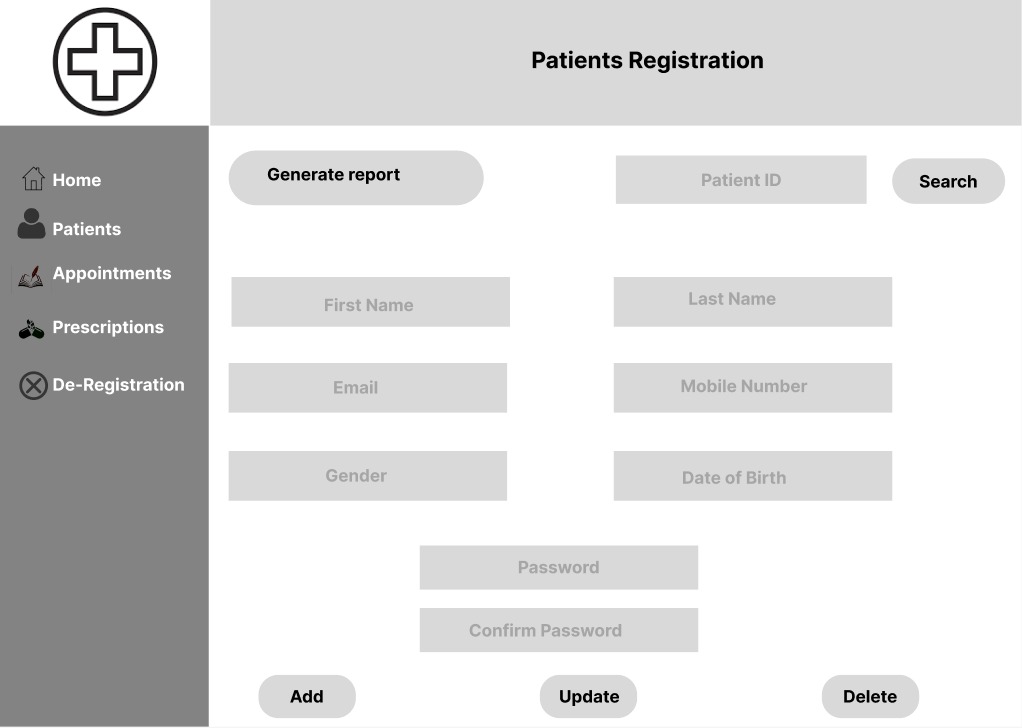
1. **Home Page:**

Features quick login links for patients patient registration, scheduling an appointment, alongside a navigation menu for various services.



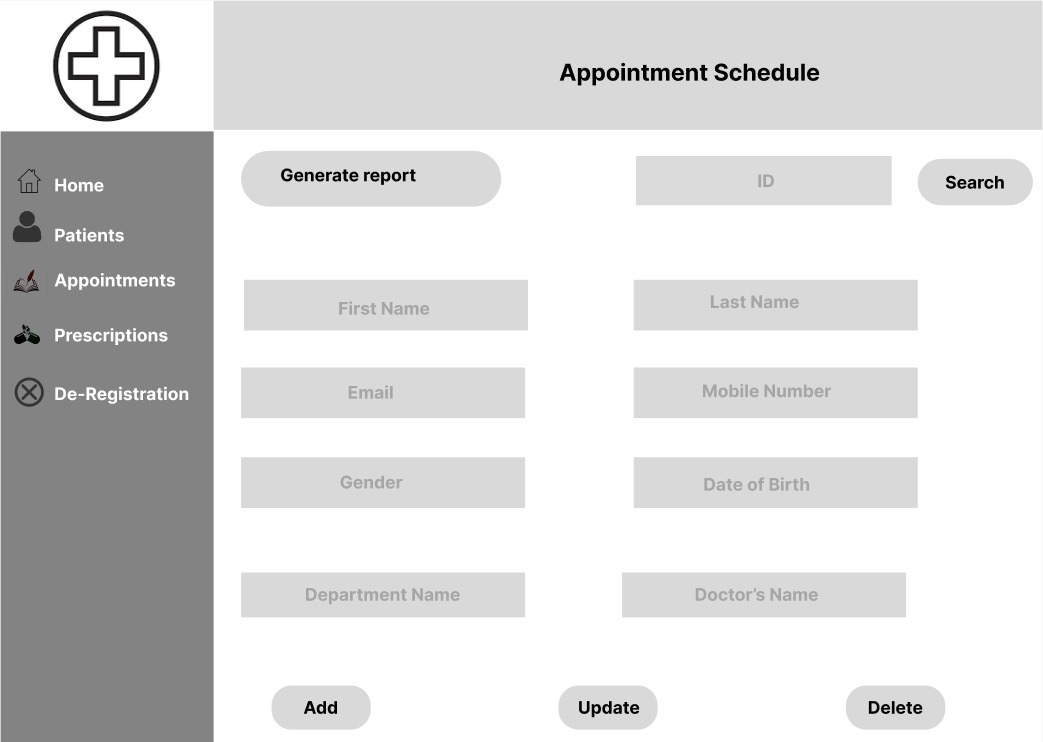
1. **Registration Page:**

Includes input fields for patient details (Name, Date of Birth). Validation checks ensure that all necessary fields are completed before registration.



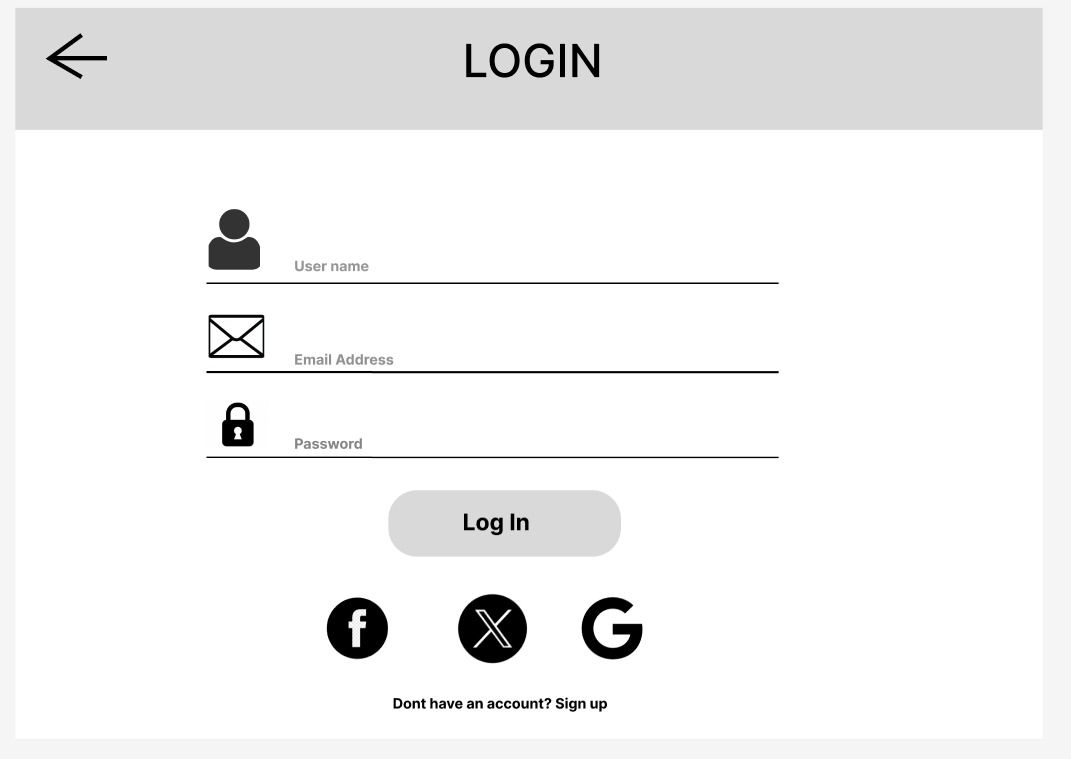
1. **Appointment Booking Page**

Allows patients to enter their unique patient ID number, date of birth, providing immediate confirmation upon successful check-in.



1. **Login Page:**

This interface allows users to log in to the system securely. It includes fields for username and password. It also has an easier option to login using an already existing google, x or Facebook account



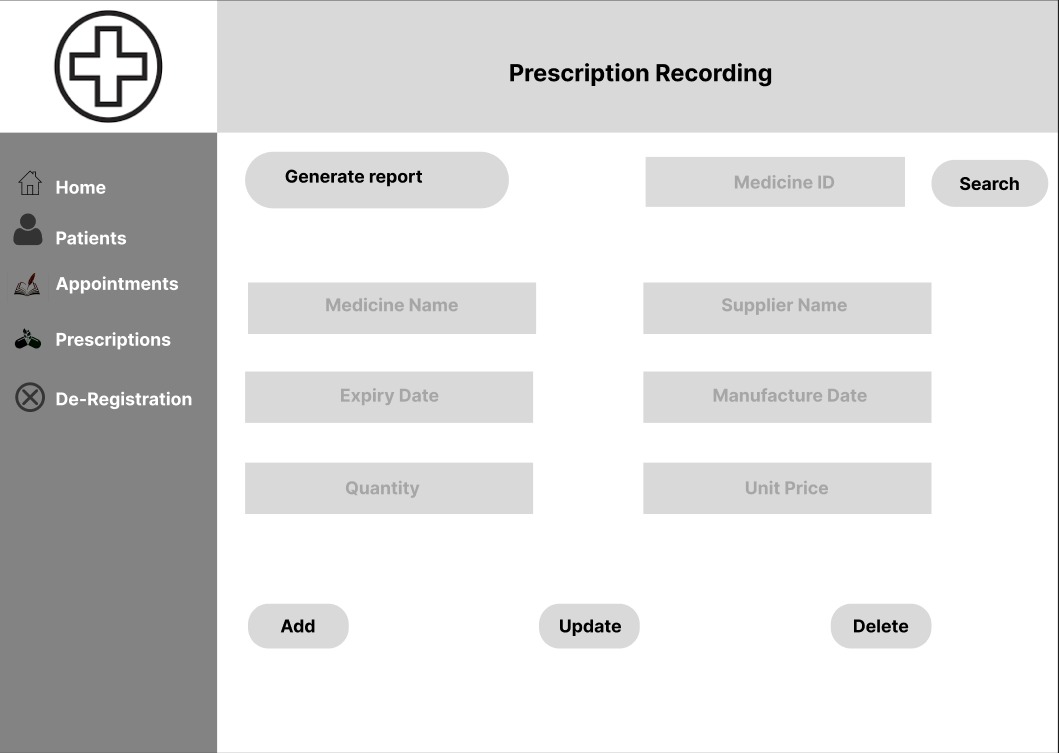
1. **Deregistration Page:**

Allows receptionists to de-register patients who have moved away or are no longer receiving care. This interface requires confirmation to prevent accidental deregistration and includes a search function to find patients by ID.



1. **Prescription Recording**

Enables doctors to input prescription details during or after an appointment. This page includes fields for medication name, dosage, frequency, and any additional instructions.



* 1. **DESING PATTERN: MVC**

The MVC (Model-View-Controller) design pattern was implemented to ensure a clear separation of concerns within the application

1. Model: This layer manages patient data, appointment details, and prescription records. It encapsulates the business logic required to handle operations such as patient registration.
2. View: The view represents the user interface of the system. It includes all user-facing elements, such as the registration form, appointment booking interface, and prescription recording page, ensuring a seamless and intuitive experience for both patients and healthcare staff.
3. Controller: The controller acts as an intermediary between the model and the view. It handles user inputs from the UI, processes the data through the model, and updates the view accordingly. For instance, when a patient books an appointment, the controller retrieves available time slots from the model and updates the view to reflect the booking confirmation.

**Benefits of using MVC design pattern**

1. Maintainability: The separation of concerns allows changes in one layer—such as updating the user interface or modifying business logic—without affecting other areas of the application. This flexibility is crucial in a healthcare environment where regulations and user needs may evolve.
2. Reusability: Components developed in the MVC structure can be reused across different parts of the application. For example, the patient registration logic can be reused in various contexts, such as for new patients or for updating existing patient information. This modularity enhances development efficiency and reduces redundancy.
3. **SOFTWARE IMPLEMENTATION**

The Health Centre Management System was developed using Python enabling the creation of a scalable and maintainable application.

* 1. **DEVELOPMENT ENVIRONMENT**

1. Programming Language: Python was selected for its simplicity and readability, which facilitated rapid development and maintenance. Its strong community support further enhanced the development experience.
2. Frameworks: Flask was utilized as the web framework, allowing for efficient routing. This streamlined the integration of front-end technologies and contributed to a structured development process.
3. User Interface Design: Wireframes served as a blueprint for the UI, ensuring that the application effectively meets user needs. The focus on user-friendly design enhances the overall experience for both patients and healthcare staff.
   1. **PSEUDOCODE**
4. Main Application Configuration

1. Initialize Flask Application

Create Flask app instance.

Set app configuration (database URI, secret key).

Blueprints for register patients, appointments, docs, auth.

2. Set up SQLite Database

Define models for Patient, Doctor, Appointment, Prescription.

Define associations between models (e.g., Appointment belongs to Patient and Doctor).

Apply migrations using Flask-Migrate.

3. Define Routes

GET / - Home page.

POST /patients/register – Register new patients.

POST /appointments/book - Book new appointments.

GET /doctors/<doctor\_id>/appointments - List doctor's appointments.

POST /auth/login - User authentication.

4. Render Templates

Extend base.html as a common layout.

Single pages for patient registration, appointment booking, and doctors' dashboards.

1. **Patient Management Module**

1. Register Patient

Collect patient information from form (name, DOB, address).

Generate unique patient ID.

Save patient information to SQLite.

2. De-register Patient

Validate patient exists.

Remove patient records from the database.

1. **Appointment Management Module**

1. Book Appointment

Collect patient ID, date, and time from form.

Verify slot availability.

Assign physician (by request or first available).

Save appointment to SQLite.

2. Cancel Appointment

Assert appointment exists.

Mark appointment as canceled.

1. **Doctors' Dashboard Module**

1. View Appointments

Query database for appointments related to doctor.

Show appointment list.

2. Record Results

Collect appointment ID and outcome details.

Store results and prescriptions in SQLite.

1. **Security and Error Handling**

1. Authentication

Use Flask-WTF for login forms.

Hash and validate passwords using Werkzeug.

2. Error Handling

Validate user input.

Return friendly error messages for errors (e.g., invalid patient ID).

1. **Deployment**

1. Set Up Local Server

Setup deploy.sh to install dependencies and initialize the database.

Run the Flask app in production mode.

1. **SOFTWARE TEST PLAN**

The Software Testing Plan for the Health Centre Management System encompasses various levels of testing, including unit testing, integration testing, system testing, and user acceptance testing (UAT). The objective is to verify that all components function as intended and meet user requirements.

1. **Unit Testing:**
2. Objective: To validate the functionality of individual components of the system.
3. Results: Successfully completed unit testing for key functionalities, including:

* Patient Registration: Valid inputs registered correctly; error messages were displayed for invalid inputs.
* Appointment Scheduling: Valid appointments were booked without issues; conflicts were properly identified and managed.
* Prescription Recording: Prescriptions were saved accurately to patient records, maintaining data integrity.
* Check-In Process: Patients checked in successfully with valid identifiers; invalid entries were appropriately rejected.
* Appointment Cancellation: Appointments were canceled correctly, with the system handling invalid cancellation requests gracefully.

1. Tool Used: pytest
2. **Integration testing**
3. Objective: To ensure that different components of the system work together seamlessly.
4. Results: Integration testing confirmed that modules interacted effectively:

* Patient Registration and Appointment Booking: New patient data was immediately accessible for booking appointments, with no delays.
* Check-In Process: Patients who registered and scheduled appointments could check in without issues, validating data synchronization across modules.

1. Tools Used: Flask-Testing
2. **System Testing**
3. Objective: To assess the overall functionality and performance of the Health Centre Management System.
4. Results: Comprehensive system testing yielded positive outcomes:

* Full Patient Lifecycle Simulation: The entire process from registration to appointment booking and check-in was executed smoothly, demonstrating robust integration.
* Stress Testing: The system effectively handled multiple concurrent users, maintaining performance and responsiveness under load.

1. Tools Used: Selenium for UI testing.
2. **User Acceptance Testing (UAT)**
3. Objective: To gather feedback from end users to ensure the system meets their needs and expectations.
4. Results: While the system was not used by actual end users in a live environment, we conducted simulated UAT sessions with a small group of team members acting as receptionists and doctors. Their feedback provided valuable insights into the system's design and usability.

* Feedback Collection: Participants performed common tasks within the application and offered their observations. They noted areas of strength as well as aspects that could benefit from improvement.
* Iterative Improvements: Based on user feedback, several adjustments were made to enhance the user interface and workflow, leading to a more intuitive experience.

**CONCLUSION**

The Health Centre Management System developed in in Python and Flask framework effectively achieved its objectives of enhancing patient registration, appointment scheduling, and prescription management. Key wireframes served as a blueprint for the UI design, ensuring it met user needs effectively. The implementation of the MVC design pattern facilitated a maintainable and scalable architecture, while thorough testing—including unit, integration, and simulated user acceptance testing—ensured the system's reliability and functionality. Overall, the system is positioned to significantly improve operational efficiency and patient experiences, with opportunities for future enhancements to adapt to evolving healthcare needs.

**REFERENCES**

[1] Sandesh Bhujbal. (2023). MVC Design Pattern. *Exploring the MVC Design Pattern in Software Development.* Retrieved Nov 10th 2024 from <https://medium.com/@sandesh__30_/mvc-design-pattern-ff40d66990e3>

[2] Thomas Hamilton. (2024). *What is Software Testing?* Retrieved Nov 10th 2024 from <https://www.guru99.com/software-testing-introduction-importance.html>