**CLASS: BCS 3106 SOFWARE ENGINEERING**

**TITTLE: HEALTH CENTER MANAGEMENT SYSTEM**

**PHASE 2: HEALTH CENTER SOLUTION REPORT**

**GROUP F**

Jamal Yakub –BSCNRB186024

Joy Winslet – BSCNRB 101924

Lidian Kitiyo – BSCNRB191424

Lebu Ayiga – BSCNRB734421

**SUBMITTED TO: CECILIA ANGELA NAFUKA**

**DATE: 15th NOVEMBER 2024**

**Project Definition**

The Health Center Management System (HCMS) project intends to improve healthcare services at a local health center with around 3000 registered patients. This system will help with important activities such patient registration, appointment scheduling and administration, emergency response, and specialized care in High Dependency Units (HDUs) and maternity units. The HCMS will be modular thanks to a microservices design, which allows each function to run as an independent service, improving scalability, reliability, and maintainability.

This solution will be built with Python for backend services and React for frontend interfaces. The system is intended to improve workflow efficiency, increase patient happiness, and minimize administrative duties for healthcare workers, particularly receptionists and physicians.

**Core Functionalities**

* Patient Registration: Enables receptionists to register new patients and keep current records of all registered patients.
* Appointment Scheduling and Management: Patients may make appointments, and receptionists can assign time slots depending on doctor availability. Supports regular and emergency appointments.
* Emergency Handling: Offers a fast-track mechanism for emergency cases, bypassing regular processes and routing patients to the closest available clinician.
* Medical Records and Prescriptions: Doctors may record visit results and issue prescriptions, which are securely maintained and easily available to both staff and patients.
* specialist Care Modules (HDU and Maternity): Oversees the assignment of patients who require specialist treatments, ensuring that they are sent to the proper care facilities and professionals.

**Objectives**

* Improve efficiency by simplifying and automating common operations such as patient registration and appointment scheduling, hence eliminating manual paperwork.
* Improve Patient Care: Ensure that patients, particularly those in critical condition, receive prompt care by prioritizing emergency and HDU management.
* Ensure Data Security: Encrypt sensitive patient data and use access control to limit who may read and manage medical information.
* Scalability: Create a system that can grow to support an expanding patient population and more services in the future.
* User-Friendly Interface: Create a simple and intuitive interface that is easily accessible to all users, including patients, physicians, and receptionists.

**Technology Stack**

* Backend: Python (Flask/Django) for developing REST APIs, data processing, and microservice architecture.
* Frontend: Use React (JavaScript/TypeScript) to create responsive and interactive user interfaces.
* MySQL is a database designed for safe data storage and healthcare record administration.
* Security features include JWT for secure user authentication and data encryption for sensitive data storage.

**GitHub Repository**

All development effort, including source code, project updates, and version history, will be stored on a GitHub repository. This repository monitors project progress, documenting each stage of development from basic setup to final implementation.

Repository Link: <https://github.com/Ayigah/Hospital-Healthcare-System>

**Software Design Considerations**

The Health Center Management System (HCMS) was designed with receptionists, physicians, and patients in mind, with a clean and straightforward interface. Using consistent design patterns makes the system modular, easy to maintain, and expandable for future needs.

**A. UI Design**

-The system interface is intended to provide simple, role-specific dashboards for receptionists, physicians, and patients. Each user type will have access to the required functionality, resulting in a smooth user experience. Key interface concerns include simplicity, ease of navigation, and security for sensitive healthcare data.

*UI Components by User Type:*

Receptionist Dashboard.

* Patient registration includes input areas for patient information (name, ID, DOB, and contact information), as well as buttons for storing or changing data.
* Appointment scheduling: Dropdown menus for choosing available physicians, date and time selectors, and the ability to label appointments as routine or emergency.
* Emergency Fast-Track: A button that activates emergency mode, prioritizing the patient for quick attention and notifying the available doctor.

Doctor's Dashboard:

* Appointment Viewer: List or calendar view of booked appointments, including patient information.
* Outcome Reporting: A form for recording visit results such as diagnosis, recommended drugs, and referrals.
* Patient Record Access: Provides secure access to patient histories, with a searchable interface for rapid retrieval.

Patient interface:

* Appointment Booking: A simple interface for checking doctor availability, booking appointments, and viewing or canceling future appointments.
* Medical information Access: Patients may securely access their medical information and medications, with the option to download or print as required.

*Wireframes*

Login page:

* Input fields include the username and password.
* Button: Login with a "Forgot Password" link for recovery.

Receptionist Dashboard:

* Patient Registration: The left pane has input areas for name, DOB, and other information.
* Appointment Scheduling: The right pane displays a calendar view, a time slot choice, and an emergency bypass button.
* Search Bar: Quickly search for registered patients.

Doctor's Dashboard:

* Appointment Calendar: The main section displays today's appointments, along with patient information.
* Patient Records Access: Appointment entries provide rapid access to patient histories.
* Outcome Reporting: Text space and choices for saving notes from each patient visit.

Patient Dashboard:

* Appointments Overview: Shown as a list, with the ability to book, view, or cancel appointments.
* Medical Records Access: You may access medical records, download medications, and contact support.

**B. Design Patterns**

The project's fundamental design patterns are the Model-View-Controller (MVC) pattern and the Microservices Architecture. These patterns were designed to isolate functionality while improving modularity, simplicity of maintenance, and system scalability.

*MVC (Model-View-Controller) Pattern.*

* Model: Handles data logic and represents the system's data structures. In HCMS, the Model layer is in charge of patient data, appointment scheduling, and medical records.
* View: this is the UI layer through which users interact with the system. Views are built using React, which provides responsive and interactive interfaces for a variety of user types.
* Controller: Handles communication between the Model and View layers, processing user requests and finding the best answer. In this system, the Controller organizes and processes user inputs such as appointment bookings and record access.

*Microservice Architecture*

* Service Separation: Each feature (e.g., patient registration, appointment scheduling, medical records) runs as a separate microservice. This separation enables separate services to be created, tested, and scaled independently.
* Service Communication: Microservices communicate via APIs, which allow each service to interact while staying decoupled.
* Benefits of Microservices:

Scalability: Each module may be scaled according to demand. For example, the appointment booking service may demand additional resources at busy hours, whilst other services stay consistent.

Fault Isolation: If one microservice fails (for example, appointment scheduling), it has no impact on other services such as patient registration or medical records.

Technology Flexibility: Different services can utilize the most appropriate technology stack. MongoDB, for example, might be used to store patient records, whereas MySQL is utilized for appointment data.

*Benefits of Reusing Design Patterns:*

* Maintainability: MVC and microservices simplify code administration, allowing certain components to be changed or corrected without affecting the overall system.
* Speed of Development: By separating concerns (data, logic, and interface), development teams may work on separate modules concurrently, increasing development speed.
* Consistency and Reusability: Using MVC and microservices principles across modules guarantees a consistent approach, which simplifies future additions and modifications.

*Security and Privacy Considerations*

* Data Encryption: All sensitive data, particularly medical information, is encrypted during transit and at rest.
* Role-Based Access Control (RBAC): Ensures that only authorized individuals (e.g., physicians, receptionists) may access certain modules.
* Authentication: Use secure login with JWT for session management to ensure user sessions are secure and verifiable.

**Software Implementation**

The HCMS implementation takes a modular approach, with each essential function implemented as its own module to ensure maintainability and scalability. The system's software is designed to segregate responsibilities using the Model-View-Controller (MVC) pattern and Microservices Architecture, allowing each module to function independently.

**A. Pseudocode for the entire system**

The pseudocode below depicts the high-level structure of the key modules, including how each function runs and interacts with other components.

**// System Initialization**

*Function InitializeSystem()*

*Load User Roles and Permissions*

*Setup Database Connections*

*Configure API Endpoints for Microservices*

*Initialize User Authentication (e.g., JWT tokens)*

*Return System Ready*

*End Function*

**// User Authentication**

*Function AuthenticateUser(username, password)*

*If username and password are valid Then*

*Assign User Role*

*Generate JWT Token*

*Redirect to Dashboard based on User Role*

*Else*

*Display Authentication Error*

*End Function*

**// Receptionist Functions**

*Function RegisterPatient(patientDetails)*

*If ValidateDetails(patientDetails) Then*

*Store Patient Information in Database*

*Return Confirmation Message*

*Else*

*Return Error Message*

*End Function*

*Function ScheduleAppointment(patientID, doctorID, dateTime)*

*If CheckDoctorAvailability(doctorID, dateTime) Then*

*Create Appointment Record in Database*

*Send Notification to Patient and Doctor*

*Return Appointment Confirmation*

*Else*

*Return Doctor Unavailable Message*

*End Function*

*Function ManageEmergency(patientID)*

*Mark Patient Record as Emergency*

*Assign Nearest Available Doctor*

*Notify Doctor with Urgent Status*

*Return Emergency Confirmation*

*End Function*

**// Doctor Functions**

*Function ViewAppointments(doctorID)*

*Retrieve Appointments from Database where DoctorID = doctorID and Date = Today*

*Display Appointment List*

*End Function*

*Function RecordOutcome(appointmentID, diagnosis, prescription, treatmentNotes)*

*Update Patient Record with Outcome Details*

*If Prescription is Provided Then*

*Save Prescription to Patient Record*

*End If*

*Return Outcome Saved Confirmation*

*End Function*

**// Patient Functions**

*Function BookAppointment(patientID, doctorID, dateTime)*

*If CheckDoctorAvailability(doctorID, dateTime) Then*

*ScheduleAppointment(patientID, doctorID, dateTime)*

*Return Booking Confirmation*

*Else*

*Return Doctor Unavailable Message*

*End Function*

*Function ViewMedicalRecords(patientID)*

*If User is Authorized Then*

*Retrieve Medical Records from Database*

*Display Records to Patient*

*Else*

*Display Access Denied Message*

*End Function*

**// Specialized Services for HDU and Maternity**

*Function AssignToHDU(patientID)*

*Update Patient Status to HDU*

*Assign HDU Care Team*

*Notify Assigned Medical Staff*

*Return HDU Assignment Confirmation*

*End Function*

*Function AssignToMaternity(patientID)*

*Update Patient Status to Maternity*

*Assign Maternity Care Team*

*Notify Assigned Medical Staff*

*Return Maternity Assignment Confirmation*

*End Function*

**// Notification System**

*Function SendNotification(recipientID, messageType)*

*Generate Message based on MessageType (e.g., Appointment Confirmation, Emergency Alert)*

*Send Message via SMS or Email to Recipient*

*Return Notification Sent Confirmation*

*End Function*

**// System Initialization Call**

*InitializeSystem()*

**// Main Program Flow**

*While System is Running*

*User logs in with AuthenticateUser()*

*If User is Receptionist Then*

**// Receptionist Actions**

*Call RegisterPatient() for patient registration*

*Call ScheduleAppointment() to book patient appointments*

*Call ManageEmergency() to fast-track emergency cases*

*Else If User is Doctor Then*

**// Doctor Actions**

*Call ViewAppointments() to display daily schedule*

*Call RecordOutcome() to save appointment outcomes*

*Else If User is Patient Then*

**// Patient Actions**

*Call BookAppointment() to schedule appointments*

*Call ViewMedicalRecords() to access medical history*

*Else If User Role involves Specialized Services Then*

**// Specialized Care Actions**

*Call AssignToHDU() or AssignToMaternity() for specialized cases*

*End If*

**// Notifications**

*Call SendNotification() as needed for appointments and emergencies*

*End While*

**Software Test Plan**

The test plan seeks to confirm that the HCMS functions properly, delivers a pleasant user experience, and complies to data security regulations. This requires several layers of testing, including unit, integration, performance, security, and user acceptability testing (UAT).

1. **Unit testing**

* Objectives: ensures that individual functions and methods perform as intended.
* Scope: Test each core module (e.g., patient registration, appointment scheduling, emergency handling) separately to ensure it meets functional requirements.
* Examples:

1. Patient Registration Module: Ensure that the system successfully takes and retains patient information.
2. Appointment Scheduling Module: Ensure that the system only makes appointments when a doctor is available and appropriately provides errors for time slots that are unavailable.
3. Pytest, a Python-based tool for automated unit testing.

* Test cases:

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Description** | **Expected Outcome** |
| 01 | Register a new patient | Patient details saved and confirmation returned |
| 02 | Schedule an appointment with availability | Appointment is scheduled and confirmed |
| 03 | Schedule an appointment with no availability | Error message returned |
| 04 | Record an appointment outcome | Outcome saved to patient record |

1. **Integration Testing**

* Objective: Ensure that diverse modules connect and integrate seamlessly, with data consistency across services.
* Scope: Test interactions between receptionist, doctor, and patient modules to ensure data flow and consistency in the database.
* Examples:

1. Appointment Scheduling and Physician Dashboard: Confirm that planned appointments display on both the doctor's dashboard and the patient's appointment list.
2. Emergency Handling and Notification: When an emergency appointment is scheduled, send an instant notification to the allocated doctor.

* Tools Used: Postman for API testing, ensuring that requests and replies between modules execute properly.

|  |  |  |
| --- | --- | --- |
| Test Case ID | Description | Expected Outcome |
| 01 | Appointment created in Reception module | Appointment appears in Doctor Dashboard |
| 02 | Emergency case marked by Reception | Notification sent to nearest available doctor |
| 03 | Patient views updated medical records | Updated record accessible from Patient Dashboard |

1. **Performance Testing**

* Objective: Evaluate the system's responsiveness and stability under regular and peak loads, guaranteeing that it can support numerous concurrent users.
* Scope: Evaluate important operations such as appointment scheduling and patient record retrieval under various load circumstances.
* Examples:

1. Appointment Scheduling: Run a simulation with 100+ people scheduling appointments at the same time to ensure that response time is acceptable.
2. Medical Record Access: Ensure that several users attempting to access records at the same time do not cause the system to delay or crash.

* Tools used include JMeter, which simulates concurrent users and measures response times.

|  |  |  |
| --- | --- | --- |
| Test Case ID | Description | Expected Outcome |
| 01 | Simulate 50 concurrent users booking appointments | Response time < 2 seconds per request |
| 02 | Access 100 patient records concurrently | No system slowdown or crashes |
| 03 | Peak load test on login system | Authentication remains stable and responsive |

1. **Security Testing**

* Objective: Ensure that patient data is safeguarded from unwanted access and security vulnerabilities in accordance with healthcare data standards.
* Scope: Evaluate user authentication, data encryption, and access control for sensitive information.
* Examples:

1. User authentication: Ensure that only authorized users have access to the system and their allocated modules.
2. Data Encryption: Ensure that patient data is encrypted while in transit and at rest.

* Tools used include OWASP ZAP for security vulnerability evaluation and penetration testing.

|  |  |  |
| --- | --- | --- |
| Test Case ID | Description | Expected Outcome |
| 01 | Attempt unauthorized access to patient data | Access denied and logged |
| 02 | Check for SQL Injection vulnerabilities | No vulnerability detected |
| 03 | Verify data encryption in transit | Data is encrypted during transfer |

1. **User Acceptance Testing (UAT)**

* Objective: Ensure that the system meets user expectations for usability, functionality, and efficiency.
* Scope: Engage hospital staff (doctors, nurses, receptionists) to test the system in a controlled environment, gathering feedback on usability and identifying any final adjustments needed before deployment.
* Examples:

1. Receptionist Workflow: Ensure that receptionists find it easy to register patients, book appointments, and manage emergency cases.
2. Doctor Workflow: Confirm that doctors can view their schedules, access patient records, and record outcomes efficiently.

* Outcome: Collect feedback and make minor adjustments based on user experience, especially for UI navigation and notification clarity.

|  |  |  |
| --- | --- | --- |
| Test Case ID | Description | Expected Outcome |
| 01 | Receptionist books and cancels an appointment | Actions complete successfully and intuitively |
| 02 | Doctor records an appointment outcome | Outcome saved, and doctor navigates easily |
| 03 | Patient accesses medical records | Patient views records smoothly with no errors |

**Conclusion**

The Health Center Management System (HCMS) has been meticulously planned, installed, and tested to fulfill the different requirements of a healthcare center with a big patient population. Each module, from patient registration to specialized treatment in HDU and maternity, was designed with scalability, security, and usability in mind. Comprehensive testing, including unit, integration, performance, security, and user acceptability testing, has confirmed the system's functionality, integration, efficiency, and adherence to healthcare data standards.

This rigorous approach assures that HCMS will optimize workflows, improve patient care, and protect data privacy, making it a valued asset for both healthcare practitioners and patients. The system is now ready for deployment, with the ability to scale and react to changing requirements.