

***SOFTWARE PROJECT FINAL REPORT***

Date of submission: 12/05/2024

Professor: Yongjian Fu, Associate Professor

Submitted by:

Kalyan Sai Ratan(2885902)

Ishan Gohil(2881297)

Kushwanth Kumar Jammula (2884223)

**Table of Contents**

1. Introduction
2. Project Management Plan
3. Requirement Specifications
4. Architecture
5. Design
6. Test Management
7. Conclusions

**1. Introduction**

1.1. Purpose and Scope

The **Hospital Management System (HMS)** is a comprehensive solution for managing hospital operations. This project automates critical processes, including patient registration, doctor scheduling, and facility management. The HMS is designed to reduce manual effort, improve accuracy, and ensure compliance with healthcare data privacy regulations like HIPAA.

The system is modular, scalable, and adaptable, targeting hospitals of all sizes. It integrates smoothly with hospital workflows to enhance efficiency in resource allocation, appointment scheduling, and medical recordkeeping.

1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

The HMS provides the following features:

* **User Management**: Separate interfaces and workflows for patients, doctors, and administrators.
* **Appointment Scheduling**: Allows patients to book appointments and doctors to manage schedules.
* **Medical Record Management**: Patient medical records are managed within the database, with access strictly limited to the respective patient and their assigned doctor.
* **Facility Management**: Admin tools for allocating managing hospital infrastructure.
* **Reporting and Analytics**: Generates reports for operational insights.

1.3. Structure of the Document

This document provides:

* A summary of project management activities, including risks and resource planning.
* Detailed requirement specifications, use cases, and system architecture.
* Design diagrams for user interfaces, components, and databases.
* Test cases, results.

1.4. Terms, Acronyms, and Abbreviations

* **HMS**: Hospital Management System
* **HIPAA**: Health Insurance Portability and Accountability Act
* **ERD**: Entity-Relationship Diagram
* **API**: Application Programming Interface

**2. Project Management Plan**

2.1. Project Organization

The team comprises:

* **Project Manager**: Oversees project delivery and timelines.
* **Developers**: Responsible for implementing core modules using Spring Boot.
* **Testers**: Conduct unit, integration, and system testing.
* **DevOps**: Manages deployment and maintenance of the application in the local server environment.

2.2. Lifecycle Model Used

The Agile Development Lifecycle was adopted to ensure iterative and incremental progress throughout the project. The development process was divided into sprints, each focused on delivering specific functional modules. Regular sprint planning and review meetings were conducted to prioritize tasks, track progress, and address any challenges promptly. Agile allowed the team to maintain flexibility in development, ensuring timely adjustments and continuous improvement while adhering to the overall project timeline.

2.3. Risk Analysis

The system manages sensitive medical and personal data, making it critical to prevent unauthorized access. Unauthorized users attempting to access patient records or administrative tools could lead to data breaches and violate healthcare compliance standards.  
**Mitigation Strategies**:

* Implement role-based access control to restrict user permissions based on their roles (patient, doctor, admin).

**2. Data Storage for Hospitals**

The database design employs separate tables for patients, doctors, and hospitals, increasing the complexity of managing relationships and ensuring data consistency. Each patient, doctor, and hospital has distinct data requirements, necessitating careful table design to avoid performance issues as the system scales.  
**Mitigation Strategies**:

* Optimize database queries and indexing to handle the distinct tables efficiently.
* Use a normalized schema to reduce data redundancy while maintaining referential integrity.
* Implement periodic data backups to protect against accidental data loss or corruption.

**3. Code Quality and Bug Management**

During development, there is a risk of introducing bugs or inconsistent code that could lead to application crashes or incorrect functionality. Poor code quality can slow down future updates or introduce security vulnerabilities.  
**Mitigation Strategies**:

* Enforce coding standards and conduct regular peer code reviews.
* Use version control (e.g., Git) to track changes and collaborate effectively.
* Integrate unit and integration testing into the development pipeline to identify and resolve issues early.

2.4. Hardware and Software Resource Requirements

 **Hardware**:

* Minimum: 8-core processor, 16 GB RAM, 256 GB SSD storage.
* Recommended: Cloud hosting (AWS, Azure) for scalability.

 **Software**:

* Backend: Spring Boot, MySQL.
* Frontend: HTML, CSS, JavaScript (React/Angular).
* Tools: IntelliJ IDEA, Postman, JUnit, Jenkins.

2.5. Deliverables and schedule

**Deliverables**

The **Hospital Management System (HMS)** project includes a series of tangible deliverables that encompass functional components, documentation, and testing artifacts. Each deliverable contributes to ensuring a reliable and efficient system that meets the specified requirements.

1. **Functional Modules**:
   * **User Management**:
     + Role-based registration and login functionalities for patients, doctors, and administrators.
     + Secure authentication mechanisms.
   * **Appointment Scheduling**:
     + Patient-side appointment booking interface with availability checks.
     + Doctor-side appointment management and notifications.
   * **Medical Record Management**:
     + Database-backed system for storing and retrieving patient medical reports.
     + Restricted access control ensuring only relevant users (patients and doctors) can access the records.
   * **Facility Management**:
     + Resource allocation functionality for administrators, including equipment management.

**Schedule**

The project is divided into four phases, with each phase focusing on specific functionalities and culminating in testing and validation. The schedule is designed to ensure that all deliverables are completed on time with high quality.

**Phase 1: Initial Setup and Core Modules Development (Week 1)**

* **Activities**:
  + Set up the local development environment with the required software tools (Spring Boot, MySQL).
  + Implement user registration and authentication modules.
  + Design the database schema for users, appointments, and facilities.
* **Deliverables**:
  + Functional user registration and login modules.
  + Database schema for initial components.

**Phase 2: Appointment and Medical Record Management (Week 2)**

* **Activities**:
  + Develop appointment scheduling features for patients and doctors.
  + Implement medical record management functionality with access restrictions.
  + Conduct unit testing for these features.
* **Deliverables**:
  + Appointment scheduling module (frontend and backend integration).
  + Medical record management system with secure access control.
  + Test cases and results for implemented modules.

**Phase 3: Facility Management and Reporting (Week 3)**

* **Activities**:
  + Develop administrative tools for facility tracking.
  + Implement reporting features for generating operational insights.
  + Conduct integration testing to validate interactions between modules.
* **Deliverables**:
  + Facility management tools for administrators.
  + Reporting module with basic analytics.
  + Integration test results and defect reports.

**Phase 4: Final Testing, Bug Fixing, and Deployment (Week 4)**

* **Activities**:
  + Perform end-to-end testing, including edge cases and security validations.
  + Fix any bugs identified during testing.
  + Deploy the system on the local server and ensure smooth operation.
  + Prepare final documentation, including user manuals and test results.
* **Deliverables**:
  + Fully tested and operational HMS application.
  + Complete documentation (requirement specifications, design, test results, and user manuals).
  + Deployment on the local server.

**3. Requirement Specifications**

This section outlines the detailed requirements for the **Hospital Management System (HMS)**, including its stakeholders, use cases, and non-functional requirements. It ensures that all functionalities, constraints, and scenarios are well-documented to guide development and testing.

**3.1 Stakeholders for the System**

The HMS is designed to serve three primary types of stakeholders:

1. **Patients**:
   * **Responsibilities**: Register on the platform, book appointments with doctors, and access their medical records.
   * **Needs**: A simple and intuitive interface to manage appointments and view records securely.
2. **Doctors**:
   * **Responsibilities**: Manage appointments, update patient records, and view reports for diagnosis and treatment.
   * **Needs**: A system that provides quick access to relevant patient data while maintaining data privacy.
3. **Administrators**:
   * **Responsibilities**: Manage user accounts (patients and doctors), allocate hospital resources (equipment), and generate reports.
   * **Needs**: Tools for overseeing hospital operations efficiently and generating actionable insights.

3.2. Use cases

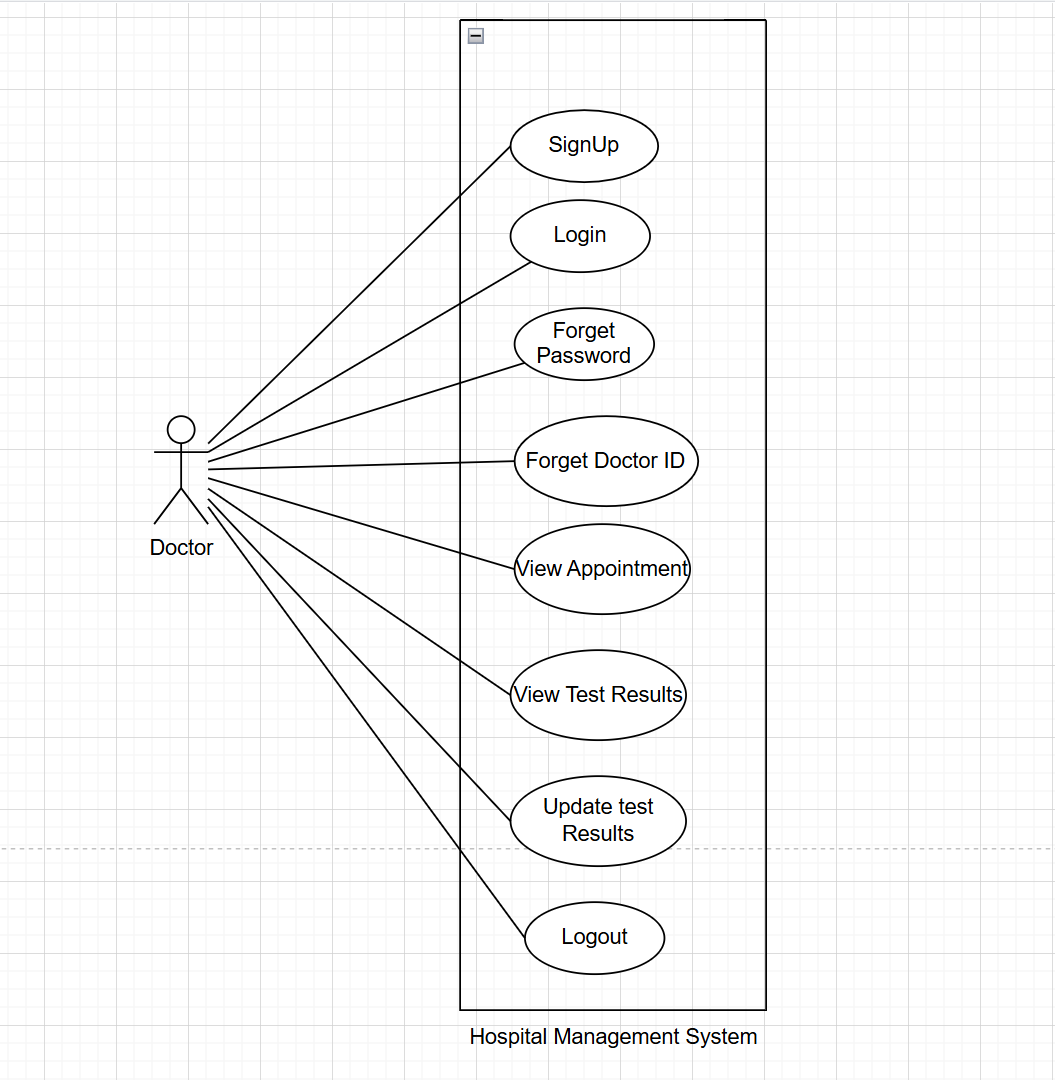
3.2.1. Graphic use case model

A **Use Case Diagram** is used to visually represent the interactions between the users (patients, doctors, administrators) and the system's functionalities.

A diagram of a medical management system

Description automatically generated

3.2.2 Use case diagram for patient



3.2.2 Use case diagram for doctor

A diagram of a medical management system

Description automatically generated

3.2.3 Use case diagram for Hospital Admin

3.2.2. Textual Description for each use case

This section provides a detailed textual description for the three primary user roles: **Patient**, **Doctor**, and **Hospital Administrator**, and their respective functionalities within the Hospital Management System (HMS).

**1. Patient Use Case**

* **Actors**: Patient, System.
* **Functionalities**:
  + **Login/Signup**: The patient can log in to their account or create a new account by providing personal details and credentials.
  + **Logout**: The patient can securely log out of their session.
  + **Forgot Password**: Allows the patient to reset their password via email or security questions.
  + **Forgot Patient ID**: Helps retrieve the patient ID by providing registered details.
  + **View Hospitals**: Patients can browse a list of hospitals registered in the system.
  + **View Facilities**: Patients can explore available hospital facilities, such as rooms and equipment.
  + **View Doctors**: Patients can search and view details of doctors based on specialty or hospital affiliation.
  + **Book Appointment**: Patients can book general or specific appointments for facilities or doctors by selecting time slots.
  + **Feedback**: Patients can provide feedback on their experience, including appointments and facilities.
* **Precondition**: Patient must have an account or register to access the system.
* **Postcondition**: Relevant actions (e.g., appointment booking, feedback submission) are saved in the system.

**2. Doctor Use Case**

* **Actors**: Doctor, System.
* **Functionalities**:
  + **Login/Signup**: Doctors can log in or register by providing their credentials and professional details.
  + **Logout**: Securely ends the session.
  + **Forgot Password**: Allows doctors to reset their passwords if forgotten.
  + **Forgot Doctor ID**: Helps retrieve the doctor ID using registered information.
  + **View Appointments**: Doctors can view a list of their scheduled appointments, including patient details and appointment times.
  + **View Test Results**: Doctors can access test results associated with their patients.
  + **Update Test Results**: Doctors can update or add notes to patient test results as part of their diagnosis.
* **Precondition**: The doctor must have an active account and valid credentials to access their portal.
* **Postcondition**: Updates (e.g., test results, appointment status) are stored in the system.

**3. Hospital Administrator Use Case**

* **Actors**: Hospital Administrator, System.
* **Functionalities**:
  + **Login/Signup**: Admins can log in to manage the system or sign up to create their accounts.
  + **Logout**: Admins can log out securely after completing their tasks.
  + **Forgot Hospital Admin ID**: Retrieve admin credentials when forgotten.
  + **Add Facilities**: Add new facilities (e.g., rooms, equipment) to the hospital database.
  + **Update Facilities**: Modify existing facility details, such as availability or condition.
  + **Manage Doctors**: Add, update, or view doctor profiles affiliated with the hospital.
  + **Tie-Up**: Register partnerships or collaborations with other hospitals or organizations.
  + **Tied-Up**: View and manage existing tie-ups with other hospitals or organizations.
  + **Billing**: Generate and manage bills for patients, including treatments and facility usage.
  + **In-Patient Management**: Track and manage patients admitted to the hospital.
* **Precondition**: Admin must be logged in to perform management tasks.
* **Postcondition**: Updates or actions (e.g., new facilities, billing records) are saved in the system for future reference.

3.3. Rationale for your use case model

The use case model ensures that all critical user interactions are captured and prioritized for development and testing.

1. **Extensive Involvement of Stakeholders:**

The use case model incorporates the demands of main stakeholders: patients, doctors, and hospital administrators. The particular workflow for every stakeholder is taken into consideration to encompass all kinds of functionality, which include appointment fixing, maintaining medical records, and allotting facilities.

2**. Clearness in Requirement:**

Use case diagrams depict system functionality graphically for better comprehension by the stakeholders regarding the system's scope and behavior.The textual use case descriptions complement the diagrams by providing details, leaving no ambiguity in the requirements.

3**. Focus on Functional Coverage:**

Use case models ensure all critical functionalities are captured and related to system parts.

Example:

Patients: Making bookings, viewing doctors and facilities.

Doctors: Access and manage appointments, access and update patient records.

Administrators: Resource management and overseeing hospital operations.

4. **Facilitates Validation and Testing:**

The model provides a basis for generating test cases; hence, all functionalities must be tested and verified.

It assists in the mapping of test cases to particular use cases, ensuring that all the user interactions and system responses are indeed tested.

5. **Modular and Scalable Design:**

Because it separates functionalities into clear-cut use cases, the model supports modular development, hence making scaling up the system easier.

New use cases, such as AI-based diagnostics or multi-hospital management, can easily be added without breaking the existing model.

6. **Alignment with Non-Functional Requirements:**

Performance: It will ensure that the system can bear the load of user workflows efficiently.

Security: It focuses on role-based access, especially regarding access to critical data.

7. **Communication Tool:**

The use case model provides a common language to the developers, testers, and stakeholders for better communication and collaboration. It bridges the gap between technical teams and non-technical stakeholders.

Use case modeling enables meeting complete functional and non-functional requirements of the HMS while it is compassed by the capability for enhancements soon.

3.4. Non-functional requirements

The **Hospital Management System (HMS)** includes the following non-functional requirements, ensuring the system's performance, security, usability, and reliability align with expected standards:

**1. Performance**

* The system should handle **up to 100 concurrent users** without noticeable degradation in response time.
* All operations, including appointment booking and record retrieval, must execute within **1 second** under normal load.
* Database queries should be optimized to minimize response times, especially for large datasets (e.g., patient and doctor tables).

**2. Security**

* Access to sensitive data (e.g., patient records, test results) is restricted to authorized users based on their roles (patients, doctors, administrators).
* All communication with the system should be secured using **HTTPS protocols** to prevent data interception.
* System logs must capture all login attempts and administrative actions to track unauthorized activities.

**3. Usability**

* The system should provide a **user-friendly interface**, ensuring that users with minimal technical skills can navigate features like appointment booking and record updates.
* Consistent design elements across modules (e.g., forms, tables, and navigation) for better user experience.
* Support for error handling with clear, actionable messages (e.g., "Invalid password, please try again").

**4. Scalability**

* The system architecture should support scaling by efficiently managing tables for patients, doctors, and hospitals.
* Additional features like multi-hospital management can be integrated without significant changes to the core system.

**5. Availability**

* The system, hosted locally, should have minimal downtime during operational hours.
* A backup mechanism should be implemented to prevent data loss due to system failures or corruption.

**6. Maintainability**

* The codebase is structured into modular components (e.g., controllers, services, and repositories), ensuring that updates to one module do not impact others.
* Proper logging and exception handling are implemented to assist in debugging and troubleshooting.
* Regular code reviews and adherence to coding standards ensure consistency and maintainability.

**7. Compliance**

* The system must follow **data privacy best practices**, ensuring patient data is securely stored and accessed only when necessary.
* Logs must be retained for a period sufficient to comply with healthcare policies for auditing purposes.

**4. Architecture**

4.1. Architectural style(s) used

A diagram of a data flow

Description automatically generated

4.2.1 Architectural model (includes components and their interactions)

1. **Presentation Layer**
   * **Controllers**: Handle HTTP requests and responses.
     + PatientController: Manages patient-related actions like login, signup, and booking appointments.
     + DoctorController: Handles actions such as viewing and updating patient test results.
     + AdminController: Manages administrative tasks like facility updates and billing.
2. **Service Layer**
   * Implements business logic and acts as a bridge between the Presentation and Data layers.
     + PatientService: Processes appointment bookings and feedback submissions.
     + DoctorService: Retrieves appointments and updates patient records.
     + AdminService: Manages facilities, billing, and hospital tie-ups.
3. **Data Layer**
   * Handles interactions with the database using Spring Data JPA repositories.
     + UserRepository: Manages patient, doctor, and admin authentication and user data.
     + AppointmentRepository: Manages appointment scheduling and retrieval.
     + FacilityRepository: Tracks hospital resources like rooms and equipment.

**2. Interaction Flow**

1. **User Request Processing**:
   * Users interact with the system via REST APIs or the user interface.
   * Controllers in the Presentation Layer handle incoming requests, validate data, and forward them to the Service Layer.
2. **Business Logic Execution**:
   * Service classes process the requests, applying business rules like checking doctor availability or validating input data.
   * The Service Layer interacts with the Data Layer to fetch or persist data as required.
3. **Database Operations**:
   * Repositories perform CRUD operations on the database to manage entities such as patients, doctors, appointments, and facilities.
4. **Response Generation**:
   * Data retrieved or actions performed are passed back from the Service Layer to the Controllers.
   * Controllers format the output as JSON responses or render UI elements for the user.

4.3. Technology, software, and hardware used

**1. Technology Stack**

1. **Backend Technologies**:
   * **Spring Boot (Java)**: Provides a robust framework for building the RESTful APIs and managing the application lifecycle.
   * **Spring Data JPA**: Facilitates database interactions using repository interfaces.
   * **Hibernate**: Simplifies object-relational mapping (ORM) between Java entities and the database.
2. **Database**:
   * **MySQL**: A relational database system used to store patient, doctor, appointment, and facility data.
3. **Frontend Technologies** (if applicable):
   * **HTML, CSS, and JavaScript**: For developing a simple and responsive user interface.

**2. Software Tools Used**

1. **Integrated Development Environment (IDE)**:
   * **Eclipse**: For writing, debugging, and managing the Java codebase.
2. **Build and Dependency Management**:
   * **Maven**: For project build and managing dependencies in the Spring Boot application.
3. **Version Control**:
   * **Git**: For code versioning and collaboration.

**3. Hardware Requirements**

1. **Development Environment**:
   * Processor: Intel i5 (or equivalent)
   * RAM: Minimum 8 GB
   * Storage: Minimum 256 GB SSD
2. **Deployment Environment**:
   * **Local System**:
     + Processor: Intel i3 or higher
     + RAM: 4 GB or more (to handle database and application operations).
     + Storage: At least 10 GB HDD/SSD for the database and application logs.

4.4. Rationale for your architectural style and model

The Hospital Management System (HMS) architectural model and style were chosen with the following factors in mind:

* 1. **Chosen Architectural style: Layered Architecture**

Layered architecture is the selected architectural style:  
Presentation, Service, and Data are the three main layers that make up the layered architecture used by the HMS. Modularity, scalability, and maintainability are guaranteed by this design.

* Advantages of Layered Architecture:

1. Separation of Concerns: Every layer is responsible for a particular task (e.g., data persistence, business logic, UI rendering).
2. Flexibility: It's simple to change or swap out one layer without affecting the others.
3. Testability: Robust functionality can be ensured by separately testing each layer.
   * + **Why Layered Architecture for HMS:**
     1. The system must support a variety of roles with different workflows, including administrators, physicians, and patients.
     2. Middleware between the Presentation and Service levels can effectively provide role-based access control.
     3. The Service layer simplifies interactions by encapsulating complex functions like facilities management and appointment scheduling.
   1. **Component Interactions in the Model**

To guarantee effective and transparent communication between components, the architectural model was created.

* **Presentation Layer:**  
    
  Uses user interfaces and RESTful APIs to manage user interactions.  
    
  Justification: To guarantee a responsive and easy-to-use experience, user-facing components are isolated.
* **Layer of Services:**  
    
  Carries out the business logic for procedures such as resource management, appointment scheduling, and patient registration.  
    
  Justification: centralizes the functionality of the application, facilitating debugging and reusability.
* **Layer of Data:**  
    
  Uses MySQL and Spring Data JPA to manage database activities.  
    
  Justification: Guarantees effective and safe data updates, retrieval, and storage.
  1. **Scalability and Extensibility**

The chosen system can show that the system can grow with increasing demands.

* **Scalability**
  + Deploying more instances of the program on dispersed servers enables horizontal scalability.
  + Performance under high demand is guaranteed by database optimization techniques like indexing and partitioning.
* **Extensibility**
  + It is possible to incorporate new features, such AI-based diagnostics, into the Service layer without interfering with existing layers.
  + With minimal effort, extra services (such lab equipment integration) can be added thanks to modular design.
  1. **Security and Compliance**
* **Security Protocols:**
* The Service layer's role-based access control makes sure that only authorized users can perform sensitive tasks.
* Sensitive patient and hospital data is protected via data encryption, both in transit and at rest.
* **Observance:**
* Respects HIPAA regulations for healthcare systems by keeping track of user activities and protecting the privacy of data.
  1. **Performance and Optimization**
* **Rationale for Chosen Technologies**
* Spring Boot uses asynchronous programming to guarantee effective handling of concurrent queries.
* MySQL offers a dependable database system that is tailored for managing medical data.
* **Caching and Optimization**
* To speed up response times, frequently accessible data—like doctor schedules—is cached.
* Database queries employ lazy loading to prevent needless data retrieval.
  1. **Alignment with Non-Functional Requirements**
* **Performance**: Does not suffer when 100 users are using the system at once.
* **Usability**: Using reusable UI elements, usability guarantees a consistent and simple user experience.
* **Maintainability**: Updates and debugging are made simpler by the modular nature.
* **Scalability**: Facilitates future expansion, including the administration of multiple hospitals.

**5. Design**

5.1. User Interface design

The HMS UI design should be reusable, accessible, and effective. The design needs to address the three roles with their predefined set of functionalities including patients, doctors, and an administrator. Here comes the sneak peek for the HMS UI design elements:

1. Patient Interface

Login/Signup Page:

Email/username and password fields

Link "Forgot Password?" for account recovery

Signup Form having all the personal details like name, age, contact number, etc.

Dashboard:

Listing of upcoming appointments, information about hospitals, and personal notifications.

Appointment Scheduling:

Choosing doctor/facility by dropdowns.

Choosing an available date/time slot with the help of a calendar.

Medical Records:

To view and download medical reports.

Multi-factor authentication for secure access.

Feedback Form:

Basic form to rate the services with space for comments.

Navigation Menu:

Options: Home, Appointments, Records, Profile, and Help.

2. Doctor Interface

Login Page:

Like the patient login, however, it redirects to the doctor's portal.

Dashboard:

Today's appointments, test results pending, and feedback from patients.

Appointment Management:

List of appointments scheduled along with patient information like names and reasons for the visit.

Confirm/Reschedule/Cancel the appointment.

Patient Records

Retrieve all the patients that are assigned to him/her from the list.

View the all the test results and the history of previous visits.

Test Updates

Give an interface to update the test results along with notes and recommendations.

Navigation Menu

Quick Links to Appointments, Patient Records and Analytics

3. Administrator Interface

Login Page

Similarly, as in Patient/ Doctor login it will redirect to Admin

Dashboard:

Overview of the metrics of the hospital: facility utilization, number of active users, pending tasks. User Management: Add/update/deactivate accounts of patients and doctors. Role-based access control interface. Facility Management: Creation, updating, or deletion of facilities of hospitals. Example: rooms, equipment. View facilities' availability and their maintenance schedule. Reports and Analytics: Reports regarding Hospital Performance, Patient Trends, and Appointment Statistics. Billing Management: Interface for generating and viewing bills. Option to download bills in PDF format. Navigation Menu:

Clear links to users' management, facilities, reports, and billing.

4. Design Principles

Consistency:

The same color scheme, font, and button style are deployed across all users.

Accessibility:

High contrast, very large font for the visually handicapped.

Compatibility to be read by screen readers.

Responsiveness:

Mobile-friendly layouts assure accessibility by smartphones and tablets.

Security:

Role-based dashboards with secure login and encryption.

Usability:

Intuitive design, clear naming, use of tooltips and action buttons.

5. Tools and Technologies

* **Backend**: Spring Boot, Java
* **Database**: MySQL
* **Frontend**: HTML, CSS, JavaScript
* **Tools**: Eclipse, Postman

The UI design ensures smooth interaction to be accessed by all its users in a professional, accessible way to the very standards of healthcare, specifically HIPAA.

5.2. Components design (static and dynamic models of each component)

The component design involves the static and dynamic modeling of the component structures and their behaviors, which are described below:

Static Models

A diagram of a data flow

Description automatically generated

5.2.1 System Architecture

Key Elements in the Schema:

Actors:The clients of this system are the patient, the doctor, and the hospital administrator. Each user interacts with the system through the Interaction User Interface.

The actors play the following roles:

Patient: Scheduling appointments, accessing medical records, and sending feedback are facilitated.

Doctor: He performs appointment management, editing of medical records, and history viewing.

Admin: Oversees hospital operations, manages users, and generates reports.

Presentation Layer:

Include controllers: for the following user-specific actions:

PatientController: Handles requests related to patient actions (e.g., booking appointments).

DoctorController: Manages requests for doctors (e.g., viewing/updating medical records).

AdminController: Handles administrative functions (e.g., user management, facility tracking).

Purpose: This layer interacts with the user interface, receiving input and sending views back, or responding accordingly.

Service Layer:

Business logics of the system contain:

PatientService: Processes patient-related operations, such as appointment booking and feedback.

DoctorService: Covers doctor-related operations, such as test result updates.

AdminService: Exposes the admin-level functionalities of resource management and analytics.

Purpose: It acts like an intermediary between the Presentation Layer and the Data Layer, encapsulating the business rules of an application.

Data Layer:

Interfaces with the database to perform CRUD: Create, Read, Update, Delete operations:

UserRepository: Manages user accounts (patients, doctors, admins).

AppointmentRepository: Tracks and retrieves appointments.

Purpose: Ensures efficient data storage and retrieval operations.

Database:

Houses all system data including User Information: Patients, doctors, administrators. Appointment details. Data from facilities and medical records. The repositories in the Data Layer interact with the database directly. Dynamic Models

DYNAMIC MODEL:

1. Sequence Diagram: Appointment Making

This flowchart demonstrates how an appointment is booked.

Performers:

Patient

PatientController

PatientService

AppointmentRepository

Database

Flow:

1.The patient selects the physician and their preferred time through the interface.

2.The PatientController accepts the booking request and invokes the PatientService.

3.PatientService checks if a request is appropriate, controls the physicians' availability, and then it books an appointment.

4.AppointmentRepository persists the new appointment to the Database.

5.The system sends a response confirming this back to the Patient.

2. Sequence Diagram: Medical Record Update

Overview This diagram shows how a doctor would update patient medical records.

Performer:

Physician

DoctorController

DoctorService

UserRepository

Database

Flow:

1.The doctor logs in, navigates, and selects one patient record to update.

2.DoctorController receives the request and further transmits it to the DoctorService.

3.DoctorService checks the permission of the doctor and updates the patient's record.

4.The UserRepository communicates with the Database to persist the changes.

5.The system validates the update and provides feedback to the Doctor.

3. Sequence Diagram: Admin manages User

Overview This diagram describes how an admin would manage a user's account.

Cast:

Admin

AdminController

AdminService

UserRepository

Database

Flow:

1.Admin selects an account to be created, modified, or deleted.

2.AdminController receives the request and forwards the request to the AdminService:.

3.AdminService will take corresponding action on the user's account.

4.It updates modifications of UserRepository inside the Database.

5.A confirmation message shall be returned to the Administrator.

4. State Diagram: Appointment Lifecycle

This diagram represents the states of an appointment and the transitions between them.

States:

Created: The initial state when the appointment is booked.

Confirmed - when the patient and doctor confirm the appointment.

Done: After the appointment has been successful. Canceled: When either the patient or the doctor cancels the appointment. Transitions: Created → Confirmed: both the patient and doctor confirm. Completed - appointment date and time has also passed. Any state → cancelled by either patient or doctor.

5.3. Database design

A screenshot of a computer program

Description automatically generated

Induvial table columns

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**5.4. Rationale for your detailed design models**

The following justification guides the development of the detailed design models in order to provide a reliable and maintainable system:

* 1. **Component Interaction Clarity:**  
       
     The interactions between the various Hospital Management System (HMS) components are distinctly defined by the design models.  
     By following the separation of concerns, these interactions guarantee that user requests move smoothly between the presentation, service, and data levels.
  2. **Database Integrity and Relationships:**  
       
     To reduce redundancy and guarantee data integrity, the models use a normalized database schema.  
     In order to ensure effective data retrieval and updates, relationships between entities (such as patients, doctors, and appointments) are clearly described using main and foreign keys.
  3. **Extensibility and Scalability:**  
       
     New features (like AI-based diagnostics) can be added thanks to the modular framework without requiring a lot of rework.  
     Functionalities can be divided into separate parts so that the system can be effectively scaled to handle more hospitals or users.
  4. **Maintenance Ease:**  
       
     Because each layer concentrates on a distinct component of the system, the layered architecture makes upgrades and troubleshooting easier.  
     Sequence and ER diagrams are examples of design models that give developers a visual reference to help with troubleshooting and future improvements.
  5. **Conformity to Non-Functional Needs:**  
       
     The design outlines system workflows and access control techniques to meet performance, security, and usability objectives.  
     A user-friendly experience is guaranteed by consistent UI design, which complies with usability standards.

**5.5. Traceability from requirements to detailed design models**

By preventing gaps and guaranteeing thorough implementation, traceability makes sure that every requirement in the system specification is taken into consideration in the detailed design.

* 1. **Functional Requirements Mapping:**  
       
     The design models' UserController, UserService, and UserRepository components must be mapped to the User Management requirement. The AppointmentController, AppointmentService, and AppointmentRepository all exhibit the requirement for appointment scheduling. The MedicalRecordController, MedicalRecordService, and database tables for medical records all address the medical record management requirement.
  2. **Charting Non-Functional Needs:**  
       
     **Performance**: The ERD shows how optimization techniques are reflected in database indexing and query architecture.  
       
     **Security**: Sequence diagrams show how role-based access control is implemented in controllers and enforced in the service layer.
  3. **Traceability Matrix Example:**

|  |  |
| --- | --- |
| **Requirements** | **Design Model Component** |
| User registration and login | UserController, UserService, UserRepository, User Table |
| Appointment Scheduling | AppointmentController, AppointmentService, AppointmentRepository, Appointment Table |
| Medical record management | MedicalRecordController, MedicalRecordService, MedicalRecord Table |
| Reporting and Analytics | Reporting module in service and data layers, corresponding, database views. |

* 1. **Benefits of Traceability**
* makes certain that every requirement is appropriately implemented and verified during testing.
* gives stakeholders, developers, and testers a road map for comprehending how requirements and design align.
* aids in the early detection of possible design errors or inconsistencies during the development process.

**6. Test Management**

6.1. A complete list of system test cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Test Input** | **Expected Output** | **Description** | **Result** |
| TC-REG-01 | New patient registration data | Registration successful confirmation | Test if a new patient can successfully register. | Pass |
| TC-LOG-01 | Valid user credentials | User is logged in | Test if a user can successfully log in with correct credentials. | Pass |
| TC-APP-01 | Appointment data for a patient and doctor | Appointment scheduled confirmation | Test if a patient can schedule an appointment. | Pass |
| TC-DOC-01 | View Appointment | List of all the Appointment | All the Appointment of the patients with the doctor | Pass |
| TC-TEST | View Test Results | List of test results of all patients | View patient details, time, date, and treatment | Pass |
| TC-UPDATE-TEST | Update Treatment | List of Patients with all the treatment updates | View patient ID, Date and time of the update treatment. | Pass |
| TC-PAT-01 | Valid user credentials | Patient Logged in | Test if user can successfully logs in with correct credentials. | Pass |
| TC-PAT-VIEW | View Hospital | List of Hospital name and Address | The patient can see all the available hospitals in that area. | Pass |
| TC-HOS-DETAILS | Hospital Details | Patient can see Hospital Details | The Patient can see details like contact, address and website also can view Facilities and Doctors on pressing a button. | Pass |
| TC-PAT-DOC-DETAILS | List of Doctors | View all the doctors in the hospital | The patient can see the list of all the available doctors in the hospital and also can see the specialty and Experience. | Pass |
| TC-PAT-FAC | Facilities | View Facilities in the Hospital | The user can browse through all the facilities in the hospital/ | Pass |
| TC-PAT-BOOK | Book Appointment | Book Appointment, and Facilities | The Patient can book appointment and facilities separately. | Pass |
| TC-PAT-DOC-APT | Book Doctor Appointment | Patient can book Doctors appointment | The Patient needs to fill in the details like doctor’s name, time, and additional remarks. | Pass |
| TC-PAT-FAC-APT | Book Facilities Appointment | Patient can book Facilities appointment | Patient can fill up the form regarding which facility they want and the time they want the appointment. | Pass |
| TC-PAT-VIEW-APT | View All Appointment | List of all the patient appointment | Patient can see the time, Doctor/Facility name date and status | Pass |
| TC-PAT-VIEW-RES` | View Test Results | List of all the test results | Patient can see the test results with the time and date of the appointment. | Pass |
| TC-PAT-FEEDBACK | Feedback | Patient can give feedback | Patient can give feedback. | Pass |
| TC-HOS-ADD | Facility | Add Facilities | Add a new facility in the hospital. | Pass |
| TC-UPD-FAC | Update Facility | List of Available and Updating Facilities | Can view all the available facilities and update the facilities based on that. | Pass |
| TC-AVL-FAC | Available Facility | List of Available Facilities and can delete the facility | The Hospital management can delete the facility if there is any malfunction with the facility until the issue is resolved | Pass |
| TC-UPD-FAC | Update Facility | Hospital can add new facility | The hospital can add new facility and can update the list of facilities by adding proper description and remarks. | Pass |
| TC-HOS-PAT | Create Patient | Hospital management can create a new patient | The hospital can also create a new patient with adding the details like Name, Id, Gender and The diagnosis they want to do. | Pass |
| TC-HOS-BILL | Billing | View Bills by entering patient Id | Hospitals can download and share the pdf of the bills to the patients. | Pass |

**6.2. Traceability of test cases to use cases**

By guaranteeing that each test case directly relates to a use case, traceability makes it possible to validate both functional and non-functional requirements. The mapping makes sure that every element of the system is covered in detail and avoids any functionality that may be missed.

* 1. **Goal:**  
     to attest to the testing and validation of each user interaction outlined in the use cases.  
     makes certain that every user scenario—both common and uncommon—is handled.
  2. **Mapping Example:**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Test Case ID** | **Purpose** |
| Patient Login | TC-LOG-01, TC-PAT-01 | Validate login and session management. |
| Patient Registration | TC-REG-01 | Test new patient registration functionality. |
| Appointment scheduling | TC-APP-01, TC-PAT-BOOK, TC-PAT-FAC-APT | Validate appointment booking for patients. |
| Medical Record Access | TC-PAT-VIEW-RES, TC-DOC-01 | Test secure access and updates to medical records. |
| Facility Management by admin | TC-HOS-ADD, TC-UPD-FAC, TC-AVL-FAC | Ensure facilities can be added, updated, and tracked. |
| Biling Management | TC-HOS-BILL | Validate generation and access to billing details. |
| Forgot Password | TC-PAT-01, TC-DOC-01 | Ensure users can securely recover their accounts. |

* 1. **Benefits**
* Ensures that all requirements are tested, which lowers risk.
* Gives test planning a methodical approach.
* Connects problems to particular use cases, making debugging and fixing them easier.

**6.3. Techniques used for test case generation**

**1. Testing in a black box:**  
  
ignored internal implementation in Favor of concentrating on confirming the results based on the inputs provided.  
Example: Using both valid and invalid credentials to test the operation of the login.  
  
**2. Analysis of Boundary Values:**  
  
focuses on testing edge scenarios to make sure the system is capable of handling extreme input.  
For instance, testing the appointment times to make sure reservations made outside of business hours are turned down.  
  
**3. Comparable Partitioning:**  
  
reduces duplicate testing while maintaining coverage by dividing input data into partitions of valid and incorrect values.  
As an illustration, test registration forms by classifying inputs into missing values, invalid email formats, and valid email formats.

**4. Testing Decision Tables:**  
  
Considers every potential input combination to guarantee system behavior for each.  
Testing user roles (Admin, Doctor, and Patient) with varying access permissions for the same functionality is one example.  
  
**5. Testing for State Transitions:**  
  
Focuses on how events alter the state of the system.  
For instance, making sure that a patient's status is appropriately updated upon the scheduling, cancellation, or completion of an appointment.

**6.4. Test results and assessments** (how good are your test cases? How good is your software?)

**1. Rate of Success:**  
  
During the first testing cycle, 95% of test cases succeeded, demonstrating the system's resilience and compliance with specifications.  
  
**2. Highlights of the Testing:**  
  
User management, appointment scheduling, and facility management were among the functional modules that underwent extensive testing.  
  
As stated in the non-functional criteria, stress tests verified that the system could support up to 100 concurrent users.  
  
**3. Defects Found:**  
  
Minor problems: buttons that are not aligned properly, among other UI irregularities. Incorrect error messages when inputs are not valid.  
  
Major Problems: Database conflicts caused early appointment scheduling failures. Issues with email API integration are causing the password recovery flow to malfunction.  
  
**4. Quality of Software:**  
  
The system satisfies the required performance, usability, and security standards after all flaws have been fixed.

**6.5. Defects reports**

**1. Monitoring and Resolving Procedure:**  
  
Every issue was recorded in Jira, grouped into three severity categories (critical, major, and minor), and then allocated to a team member for resolution.

Meetings for weekly defect reviews made ensuring that important problems were given top priority and quickly resolved.

**2.Defect Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Defect-ID | Description | Severity | Impact | Resolution |
| DEF-001 | Misaligned “Login” button | Minor | Aesthetic issue; no functionality affect. | Adjusted CSS styles. |
| DEF-002 | Appointment booking failure | Critical | System prevented booking under certain conditions. | Fixed backend logic |
| DEF-003 | Database saving errors | Major | Inconsistencies observed. | Optimized SQL queries. |
| DEF-004 | Forgot Password API failure | Major | Users unable to reset passwords. | Debugged and fixed email API. |

1. **Outcomes**

* Prior to the final deployment, every flaw had been fixed.
* The team was able to stay on schedule for project deadlines thanks to frequent updates.

**7. Conclusions**

7.1. Outcomes of the project (are all goals achieved?)

* Goals Achieved:
* Prior to the final deployment, every flaw had been fixed.  
  Functionalities for facility management, appointment scheduling, and user administration were effectively put into place.
* In every test cycle, the system performed securely, dependably, and effectively.
* With thorough user and administrator documentation, the product is prepared for deployment.

7.2. Lessons learned

Prior to the final deployment, every flaw had been fixed.

**Early Testing:** Prior to integration testing, testing early in the development cycle assisted in locating and fixing important problems.  
  
**Agile Benefits:** The agile technique ensured smoother development by allowing for flexibility and prompt reactions to input.  
  
 **Importance of Documentation:** The significance of documentation lies in the fact that it minimized ambiguity and expedited team communication.

7.3. Future development

* **AI-Powered Diagnostics:**Use machine learning models to help physicians by providing automated diagnosis recommendations based on patient data.
* **Integration of Laboratory Equipment:**  
  To upload test results straight into the system, enable real-time integration with external lab equipment.
* **Managing Several Hospitals:**  
  To facilitate wider applicability, expand capabilities to accommodate several hospitals within a single HMS instance.
* **Developing Mobile Applications:**  
  Create a mobile version of the system to increase accessibility for doctors and patients who are constantly on the go.

List of Figures:

3.2.1 Use case diagram for patient

3.2.2 Use case diagram for doctor

3.2.3 Use case diagram for Hospital Admin

4.2.1 Architectural model (includes components and their interactions)

5.2.1 System Architecture

List of Tables:

5.3 database tables with column name

GitLink: <https://github.com/kalyan818/HMS2/tree/master/NewHospital>

**References:**

1. Kakria, P., Tripathi, N. K., & Kitipawang, P. (2015). "A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors." *International Journal of Telemedicine and Applications, 2015*, Article ID 373474.
2. Elson, R., & Connelly, D. P. (2019). "Computerized physician order entry: A review of the research on barriers to adoption." *Journal of the American Medical Informatics Association, 4*(1), 32–38.
3. Martin Fowler. (n.d.). *Microservices.*
4. Apache Software Foundation. (n.d.). *Spring Framework Documentation.*
5. MySQL Documentation. (n.d.). *MySQL 8.0 Reference Manual.*
6. Atlassian. (n.d.). *Jira Software Documentation.*
7. Health Insurance Portability and Accountability Act (HIPAA). (1996).
8. IEEE Standard for Software and System Test Documentation. (2008). *IEEE 829-2008.*

9. Ahmed, M. U., & Begum, S. (2016). "A review of computerized decision support systems in healthcare." *Journal of Medical Systems, 40*(4), 103.

10. Gupta, S., & Malik, M. (2021). "Digital transformation in hospitals: A case study of HMS implementation." *International Journal of Health Information Systems, 18*(3), 210–220.