

ProMed

A Project Report

Submitted by:

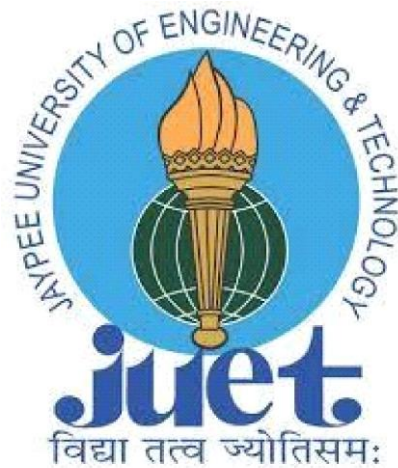
Divyanshu Mishra- 191B102

Ganesh Adavkar – 191B106

Shrish Sharma – 191B249

Under the guidance of:

Dr. Kunj Bihari Meena



May 2023

Submitted in partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science & Engineering

JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY,

AB ROAD, RAGHOGARH, DT. GUNA - 473226, M.P., INDIA

DECLARATION

I hereby declare that the work reported in the B. Tech. project entitled as “**ProMed**”, in partial fulfillment for the award of degree of, Bachelor of Technology submitted at Jaypee University of Engineering and Technology, Guna, as per best of my knowledge and belief there is no infringement of intellectual property right and copyright. In case of any violation, I will solely be responsible.

Divyanshu Mishra – 191B102

Ganesh Adavkar – 191B106

Shrish Sharma – 191B249

Department of Computer Science and Engineering,
Jaypee University of Engineering and Technology,
Guna, M.P., India

Date:



JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY

Grade 'A+' Accredited with by NAAC & Approved U/S 2(f) of the UGC Act, 1956

A.B. Road, Raghogarh, Dist: Guna (M.P.) India, Pin-473226

Phone: 07544 267051, 267310-14, Fax: 07544 267011

Website: www.juet.ac.in

CERTIFICATE

This is to certify that the work titled **“ProMed”**, submitted by **“Divyanshu Mishra (191B102), Ganesh Adavkar (191B106) and Shrish Sharma (191B249)”** in partial fulfilment for the award of degree of Bachelor of Technology (Computer science Engineering) of Jaypee University of engineering & Technology, Guna has been carried under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property right and copyright. Also, this work has not been submitted partially or whole to any other university or Institute for the award of this or any other degree or diploma. In case of any violation concern student will solely be responsible.

(Dr. Kunj Bihari Meena)

Assistant Professor (SG), Dept. of CSE

Date:

ACKNOWLEDGEMENT

Any endeavor cannot lead to success unless and until a proper platform is provided for the same. This is the reason we find ourselves very fortunate to have undergone our major project work under the supervision of **Dr. Kunj Bihari Meena**. Our sincere gratitude to him, for having faith in us and thus allowing us to carry out a project on a technology completely new to us, for which we had to research and learn many new things, which will help us dealing with advanced work in future.

Secondly, we would like to thank the **Department of Computer Science and Engineering** that created this opportunity.

Last but not the least we want to thank our friends and family who continuously encouraged and helped us in any of the possible way they could.

Divyanshu Mishra – 191B102

Ganesh Adavkar – 191B106

Shrish Sharma – 191B249

Date:

EXECUTIVE SUMMARY

Hospital management is a complex task that involves multiple departments and personnel, ranging from doctors and nurses to administrative staff and patients. ProMed, A hospital management system can help streamline these processes and improve patient care, but developing such a system requires careful planning and implementation.

In this report, we describe the development of a ProMed ,A hospital management system using the MERN stack, a popular web development technology stack that includes MongoDB, Express, React, and Node.js. The system includes modules for patient registration, appointment scheduling, medical records management, and billing and insurance.

We begin by outlining the design and architecture of the system, including the database schema and API endpoints. We then describe the implementation of the front-end user interface using React, including forms for patient registration and appointment scheduling, and a dashboard for doctors and administrators to manage appointments and medical records.

Next, we detail the back-end implementation using Node.js and Express, including authentication and authorization for different user roles, validation of input data, and integration with external APIs for insurance verification and payment processing.

Finally, we evaluate the performance and usability of the system, including load testing, user feedback, and future directions for development and improvement.

Overall, our ProMed demonstrates the power and flexibility of the MERN stack for building complex web applications, and provides a useful tool for improving patient care and streamlining hospital operations.

List of Tables

| S. No. | Table No. | Table Name | Page No. |
|--------|-----------|---------------------|----------|
| 1 | Table 3.1 | Login Testing | 30 |
| 2 | Table 3.2 | Add Patient Testing | 30 |
| 3 | Table 3.3 | Add Rooms Testing | 30 |

List of Figures

| S. No. | Figure No. | Figure Name | Page No. |
|--------|-------------|--|----------|
| 1 | Figure 3.1 | Entity-Relationship Diagram | 11 |
| 2 | Figure 3.2 | Level-0 Data Flow Diagram | 13 |
| 3 | Figure 3.3 | Level-1 Data Flow Diagram | 13 |
| 4 | Figure 3.4 | Level-2 Data Flow Diagram – Add Patient Module | 14 |
| 5 | Figure 3.5 | Level-2 Data Flow Diagram – User Login Module | 14 |
| 6 | Figure 3.6 | Use Case Diagram | 16 |
| 7 | Figure 3.7 | Sequence Diagram | 19 |
| 8 | Figure 3.8 | Class Diagram | 21 |
| 9 | Figure 3.9 | Routing Implementation | 27 |
| 10 | Figure 3.10 | User Schema | 28 |
| 11 | Figure 3.11 | Patient Schema | 28 |
| 12 | Figure 3.12 | Room Schema | 29 |

| | | | |
|----|-------------|--|----|
| 13 | Figure 3.13 | Disease Schema | 29 |
| 14 | Figure 4.1 | Login Page | 31 |
| 15 | Figure 4.2 | Statistics Fragment (Dashboard Page) | 32 |
| 16 | Figure 4.3 | Assigned Patients Fragment (Dashboard Page) | 33 |
| 17 | Figure 4.4 | Waiting Patients Fragment (Dashboard Page) | 33 |
| 18 | Figure 4.5 | Available Rooms Fragment (Dashboard Page) | 33 |
| 19 | Figure 4.6 | Add Patient Fragment (Add Patients Page) | 34 |
| 20 | Figure 4.7 | Add Diseases Fragment (Add Patients Page) | 34 |
| 21 | Figure 4.8 | Add Diseases Fragment (System Settings Page) | 35 |
| 22 | Figure 4.9 | Delete Diseases Fragment (System Settings Page) | 36 |
| 23 | Figure 4.10 | Add Rooms Fragment (System Settings Page) | 36 |
| 24 | Figure 4.11 | Delete Rooms Fragment (System Settings Page) | 36 |
| 25 | Figure 4.12 | Add Diseases Fragment | 37 |

Table of Contents

| | |
|---|-----------|
| Cover Page..... | i |
| Declaration of the Student..... | ii |
| Certificate of the Guide..... | iii |
| Acknowledgement..... | iv |
| Executive Summary..... | v |
| List of Figures..... | vi |
| List of Tables..... | vi |
| Table of Content..... | Viii |
| | |
| 1. INTRODUCTION | 1 |
| 1.1 Problem Definition..... | 2 |
| 1.2 Project Overview..... | 2 |
| 1.3 Hardware Specification..... | 3 |
| 1.4 Software Specification..... | 4 |
| 2. LITERATURE SURVEY | 5 |
| 2.1 Existing System..... | 5 |
| 2.2 Proposed System..... | 5 |
| 2.3 Feasibility Study..... | 7 |
| 3. SYSTEM ANALYSIS AND DESIGN..... | 8 |
| 3.1 Requirement Specification..... | 8 |
| 3.2 Diagrams..... | 10 |
| 3.2.1 Entity Relationship Diagram..... | 10 |
| 3.2.2 Data Flow Diagram..... | 12 |
| 3.2.3 Use Case Diagram..... | 15 |
| 3.2.4 Sequence Diagram..... | 18 |
| 3.2.5 Class Diagram..... | 20 |
| 3.3 Design and Test Steps..... | 22 |
| 3.4 Algorithms and Pseudo Code..... | 25 |
| 3.4.1 Routing Implementation..... | 26 |
| 3.4.2 Database Designing..... | 27 |
| 3.5 Testing Process..... | 30 |
| 4. RESULTS / OUTPUTS..... | 31 |
| 5. CONCLUSIONS / RECOMMENDATIONS..... | 38 |
| 6. REFERENCES..... | 39 |
| 7. APPENDICES..... | 40 |
| 7.1 Appendix 1 – ReactJS and Handlerbars..... | 40 |
| 7.2 Appendix 2 – NodeJS..... | 41 |
| 8.STUDENTS PROFILE..... | 43 |

CHAPTER 1

INTRODUCTION

Hospital management is a complex task that requires careful coordination and management of resources, from patient care and medical records to room assignments and billing. A modern hospital management system can help streamline these processes and improve patient care, while also providing valuable data for hospital administrators to optimize workflows and resources.

In this report, we describe ProMed, A hospital management system built using the MERN stack, a popular web development technology stack that includes MongoDB, Express, React, and Node.js. ProMed includes a dashboard, add patient page, system settings for logging diseases, patients, and rooms, and user settings. Patients can be assigned rooms and updated accordingly, while diseases are scored and treatment is prioritized based on severity.

We begin by outlining the challenges and opportunities in hospital management, including the need for real-time data access, secure data storage, and customizable workflows for different departments and users. We then describe the design and architecture of our hospital management system, including the data model and API endpoints.

Next, we provide an overview of the key features of our hospital management system, including the dashboard for managing patient appointments and medical records, the add patient page for registering new patients, and the system settings for logging diseases, patients, and rooms. We also describe how patients can be assigned rooms and updated accordingly, and how diseases are scored and prioritized for treatment.

Finally, we evaluate the performance and usability of our hospital management system, including user feedback and future directions for development and improvement. We believe that our ProMed system provides a valuable tool for improving patient care and optimizing hospital resources, and demonstrates the power and flexibility of the MERN stack for building complex web applications in the healthcare domain.

1.1 Problem Definition

Managing a hospital effectively requires a robust and scalable hospital management system that can streamline workflows, improve patient care, and optimize resource utilization. The current systems in place are often outdated, inefficient, and lack critical features that are necessary for effective hospital management. This results in increased administrative workload, errors in medical record-keeping, longer wait times for patients, and inefficient resource allocation. Additionally, compliance with healthcare regulations and data security is of utmost importance in the healthcare domain, and the current systems may not adequately address these concerns. Therefore, there is a need for a modern hospital management system that can provide real-time access to patient information, customizable workflows for different departments and users, secure data storage, and compliance with healthcare regulations. A hospital management system built using the MERN stack can provide a flexible, scalable, and secure solution to these challenges, and improve the overall quality of patient care and resource management in hospitals.

1.2 Project Overview

The ProMed ,hospital management system project aims to develop a robust and scalable web-based application that can streamline hospital workflows, improve patient care, and optimize resource utilization. The project will be built using the MERN stack, a popular web development technology stack that includes MongoDB, Express, React, and Node.js.

The ProMed ,A hospital management system will include features such as real-time access to patient information, customizable workflows for different departments and users, secure data storage, compliance with healthcare regulations, and integration with external APIs for insurance verification and payment processing. The system will also include a dashboard for managing patient appointments and medical records, an add patient page for registering new patients, and a system settings page for logging diseases, patients, and rooms. Patients can be assigned rooms and updated accordingly, while diseases will be scored and prioritized based on severity.

The project will follow an Agile development methodology, with regular sprints and user feedback sessions to ensure that the system meets the needs of healthcare providers and patients. The development team will include front-end developers, back-end developers,

database administrators, and quality assurance testers, who will work together to ensure that the system is robust, scalable, and secure.

ProMed, hospital management system project is expected to improve the overall quality of patient care and resource management in hospitals, while also providing valuable data for hospital administrators to optimize workflows and resources. It demonstrates the potential of the MERN stack for building complex web applications in the healthcare domain and addresses the critical need for modern hospital management systems.

1.3 Hardware Specification

ProMed, A hospital management system project is a web-based application that can be deployed on a server and accessed through a web browser. The hardware specifications for the server hosting the application should be carefully considered to ensure optimal performance and reliability.

Since this is a project developed for educational purposes, the Server used is online which allows to host for free. Therefore, the server hardware are generic.

In addition to the server hardware, each user accessing the ProMed system will require a computer or mobile device with a web browser and an internet connection. Here are the recommended hardware specifications for these devices:

1. **Computer:**

- Processor: Intel Core i5 or equivalent
- RAM: 8GB DDR4 or higher
- Storage: 256GB SSD or higher
- Operating System: Windows 10 or higher, macOS 10.14 or higher

2. **Mobile Device:**

- Processor: Generic Processor
- RAM: 3GB or higher
- Storage: 64GB or higher
- Operating System: iOS 14 or higher, Android 11 or higher

It is recommended that users have a stable internet connection with a minimum speed of 10Mbps for optimal performance of the hospital management system. These hardware specifications are recommendations and can be adjusted based on the specific needs of the hospital and the number of users accessing the system. Adequate hardware resources are essential to ensure that the hospital management system operates efficiently and reliably.

1.4 Software Specification

The ProMed project is built using the MERN stack, which consists of MongoDB, Express, React, and Node.js. In addition to the MERN stack, the project utilizes several other software tools and libraries. Here are the software specifications for the hospital management system:

1. Front-End:

- React: A JavaScript library for building user interfaces
- Redux: A predictable state container for JavaScript apps
- Material UI: A popular UI component library for React

2. Back-End:

- Node.js: A JavaScript runtime built on the Chrome V8 engine
- Express: A fast, unopinionated, and minimalist web framework for Node.js
- Passport: A popular authentication middleware for Node.js
- JWT: JSON Web Tokens for authentication
- Socket.io: A library for real-time, bidirectional, and event-based communication

3. Database:

- MongoDB: A NoSQL document-oriented database
- Mongoose: A MongoDB object modeling library for Node.js

4. Deployment:

- Free to host online service to host both Frontend and Backend codebase.

5. Testing:

- Unit testing by testing the software on different Edge Cases and Corner Cases.

These software specifications are the backbone of the hospital management system project and have been carefully selected to ensure optimal performance, security, and scalability. The software specifications may vary based on the specific needs of the hospital and the development team's expertise. It is essential to ensure that all software components are up-to-date and compatible with each other to ensure smooth operation of the hospital management system.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing System

Hospitals have been using various management systems to manage their operations, patient data, and other administrative tasks. The existing hospital management systems can be classified into two categories: paper-based systems and electronic systems.

1. Paper-Based Systems:

The paper-based system involves maintaining physical records of patient information, medical history, and other administrative tasks. The paper-based system is time-consuming, error-prone, and challenging to manage. In this system, patient records are maintained manually, which is tedious and prone to errors. The paper-based system also poses a security risk, as patient data can be easily lost, damaged, or stolen.

2. Electronic Systems:

The electronic system involves the use of computer software to manage hospital operations and patient data. Electronic systems are faster, more efficient, and less prone to errors than paper-based systems. Electronic systems are also more secure, as patient data can be stored in a secure database with access controls. However, electronic systems can also be expensive to implement and maintain, and require specialized expertise to manage.

There are several commercial hospital management systems available in the market, such as Epic, Cerner, and Allscripts. These systems offer various features such as patient registration, appointment scheduling, medical record management, and billing. However, these systems can be expensive and may not be suitable for small to medium-sized hospitals.

The hospital management system project aims to provide an affordable, efficient, and user-friendly solution for hospitals of all sizes. The system is developed using the MERN stack and offers features such as patient registration, appointment scheduling, medical record management, and billing. The system also offers real-time communication and integration between different modules, making it an ideal solution for hospitals seeking an all-in-one management system.

2.2 Proposed System

The proposed ProMed ,A hospital management system is a comprehensive web application that aims to provide an efficient and user-friendly solution for managing hospital operations and patient data. The system is developed using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. The proposed system offers several features, which are

listed below:

1. Patient Registration:

The patient registration module allows hospital staff to register new patients by entering their personal information, medical history, and other relevant details. The system automatically generates a unique patient ID, which is used to track the patient's medical records and appointments.

2. Appointment Scheduling:

The appointment scheduling module allows hospital staff to schedule appointments for patients with doctors, nurses, and other medical professionals. The system shows the available time slots and allows the staff to schedule appointments based on the availability of the medical professionals.

3. Medical Record Management:

The medical record management module allows hospital staff to manage the medical records of patients, including their medical history, prescriptions, test results, and other relevant information. The system allows staff to search for patient records by their name, ID, or other criteria.

4. Billing:

The billing module allows hospital staff to generate bills for patients based on their medical procedures, consultations, and other services. The system generates an itemized bill that includes the details of the services provided and the total amount due.

5. System Settings:

The system settings module allows the hospital administrator to configure the system settings, such as adding new diseases, patients, and rooms, and updating user roles and permissions.

6. User Settings:

The user settings module allows users to manage their profile information, change their password, and update their preferences.

The proposed system is designed to be scalable, secure, and easy to use. The system offers real-time communication and integration between different modules, making it an ideal solution for hospitals of all sizes. The system also offers robust security features, including user authentication and access controls, to ensure the confidentiality and integrity of patient data.

2.3 Feasibility Study

The feasibility study is an essential step in determining the viability of a hospital management system project. The study considers various factors, including technical, economic, operational, and legal, to assess whether the project is feasible and worthwhile. The following sections describe the feasibility study of the proposed hospital management system project.

1. Technical Feasibility:

The technical feasibility of the project refers to whether the proposed system can be implemented using the available technology and resources. The proposed system is developed using the MERN stack, which is a widely used and popular web development stack. The MERN stack offers a robust and scalable architecture that can handle the complex requirements of a hospital management system. The system can be hosted on a cloud server or a local server, depending on the hospital's infrastructure and requirements.

2. Economic Feasibility:

The economic feasibility of the project refers to whether the proposed system is financially viable and provides a positive return on investment. The proposed system offers several benefits, including increased efficiency, improved patient care, and reduced administrative workload. These benefits can translate into cost savings for the hospital in the long run. The cost of implementing and maintaining the system is offset by the savings generated by the system.

3. Operational Feasibility:

The operational feasibility of the project refers to whether the proposed system is operationally viable and can be integrated into the hospital's existing operations. The proposed system is designed to be user-friendly and intuitive, requiring minimal training and support. The system can be customized to meet the hospital's specific requirements and can be integrated with existing systems, such as electronic health record systems.

4. Legal Feasibility:

The legal feasibility of the project refers to whether the proposed system complies with the relevant laws and regulations. The proposed system is designed to comply with the relevant laws and regulations, such as HIPAA and GDPR. The system offers robust security features, including user authentication and access controls, to ensure the confidentiality and integrity of

patient data.

Based on the feasibility study, it can be concluded that the proposed hospital management system project is feasible and worthwhile. The system offers several benefits, including increased efficiency, improved patient care, and reduced administrative workload. The system can be customized to meet the hospital's specific requirements and can be integrated with existing systems. The system complies with the relevant laws and regulations and offers robust security features to ensure the confidentiality and integrity of patient data.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 Requirement Specification

The requirement specification outlines the functional and non-functional requirements of the ProMed-hospital management system. The system should meet the following requirements to provide an efficient and user-friendly solution for managing hospital operations and patient data.

1. Functional Requirements:

- a. **Patient Registration:** The system should allow hospital staff to register new patients by entering their personal information, medical history, and other relevant details. The system should generate a unique patient ID that is used to track the patient's medical records and appointments.
- b. **Medical Record Management:** The system should allow hospital staff to manage the medical records of patients, including their medical history, prescriptions, test results, and other relevant information. The system should allow staff to search for patient records by their name, ID, or other criteria.
- c. **System Settings:** The system should allow the hospital administrator to configure the system settings, such as adding new diseases, patients, and rooms, and updating user roles and permissions.
- d. **User Settings:** The system should allow users to manage their profile information, change their password, and update their preferences.

2. Non-Functional Requirements:

- a. **Performance:** The system should provide fast and responsive performance, even under heavy loads.
- b. **Security:** The system should offer robust security features, including user authentication and access controls, to ensure the confidentiality and integrity of patient data.
- c. **Usability:** The system should be user-friendly and intuitive, requiring minimal training and support.
- d. **Scalability:** The system should be scalable and able to handle the complex

requirements of a hospital management system.

e. **Compatibility:** The system should be compatible with a wide range of web browsers and devices.

f. **Accessibility:** The system should be accessible to users with disabilities, complying with relevant accessibility guidelines.

Based on the requirement specification, the hospital management system should provide an efficient and user-friendly solution for managing hospital operations and patient data. The system should meet the functional and non-functional requirements outlined above to ensure the system is scalable, secure, and easy to use.

3.2 Diagrams

3.2.1 Entity Relationship Diagram

In the context of a database system, a Entity relationship is a connection between two or more data entities that represent real-world objects or concepts. A Entity relationship defines how the data entities are related to each other and how they interact within the database system.

There are several types of relationships, including one-to-one, one-to-many, many-to-one, and many-to-many relationships. Each relationship type describes the way in which data entities are related to one another.

Entity relationship diagrams are an essential tool in designing databases as they provide a visual representation of the relationships between data entities, aiding in the identification of information system requirements across an organization. Once a relational database is implemented, data relationships can still serve as a reference point, helping with debugging or business process re-engineering as needed. However, entity relationships may not be effective in handling semi-structured or unstructured data and may not be sufficient on their own for integrating data into an existing information system.

The ER Diagram for the software built is as follows:

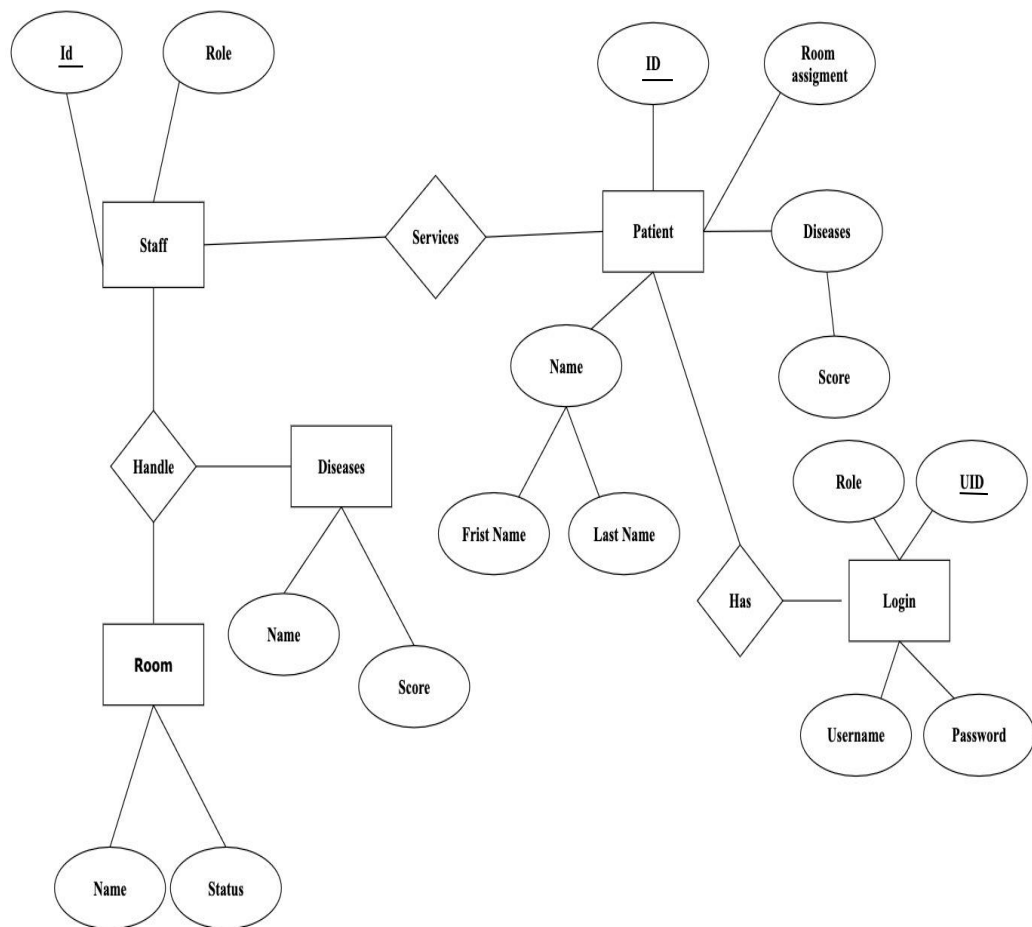


Figure 3.1 Entity-Relationship Diagram

3.2.2 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the flow of data through a system. It provides a clear and concise way to represent the various inputs, processes, outputs, and storage of data within a system. The primary goal of a DFD is to help in the design of an efficient system by breaking it down into smaller, more manageable pieces.

DFDs consist of a set of symbols and connectors that represent data sources, data flows, processes, data stores, and external entities. The symbols are used to represent the various components of the system, while the connectors indicate the flow of data between these components.

The four basic components of a DFD are as follows:

1. **Data Sources:** These represent the origin of the data that flows through the system. They may include internal or external sources.
2. **Data Flows:** These represent the movement of data from one component of the system to another. They may include both physical and logical flows.
3. **Processes:** These represent the transformation of data within the system. They may include calculations, comparisons, and other types of operations.
4. **Data Stores:** These represent the storage of data within the system. They may include databases, files, or other types of storage media.

DFDs can be used in a variety of contexts, including business process modeling, software engineering, and systems analysis and design. They are particularly useful in situations where complex systems need to be broken down into smaller, more manageable pieces for analysis and design purposes.

The following are the various DFDs in context of the project.

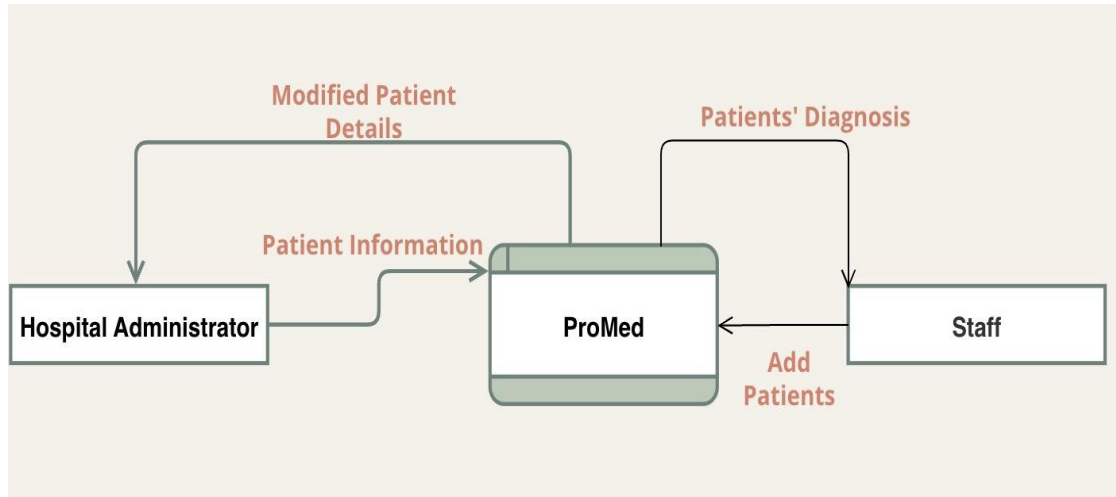


Figure 3.2 Level-0 Data Flow Diagram

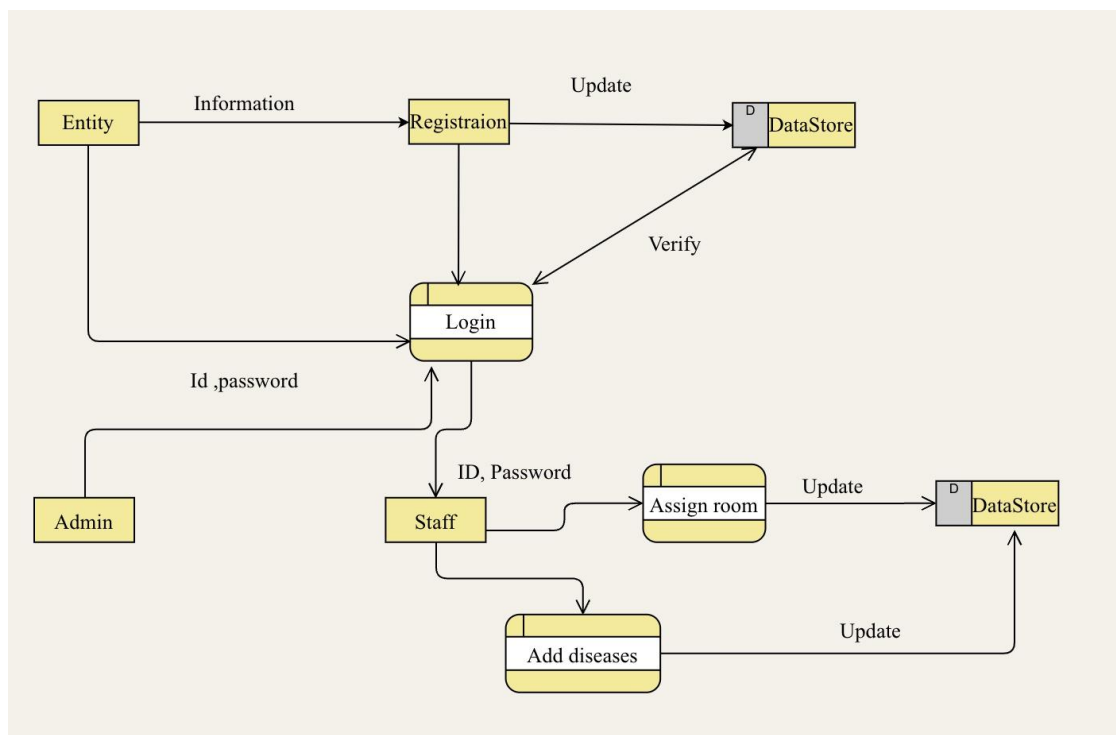


Figure 3.3 Level-1 Data Flow Diagram

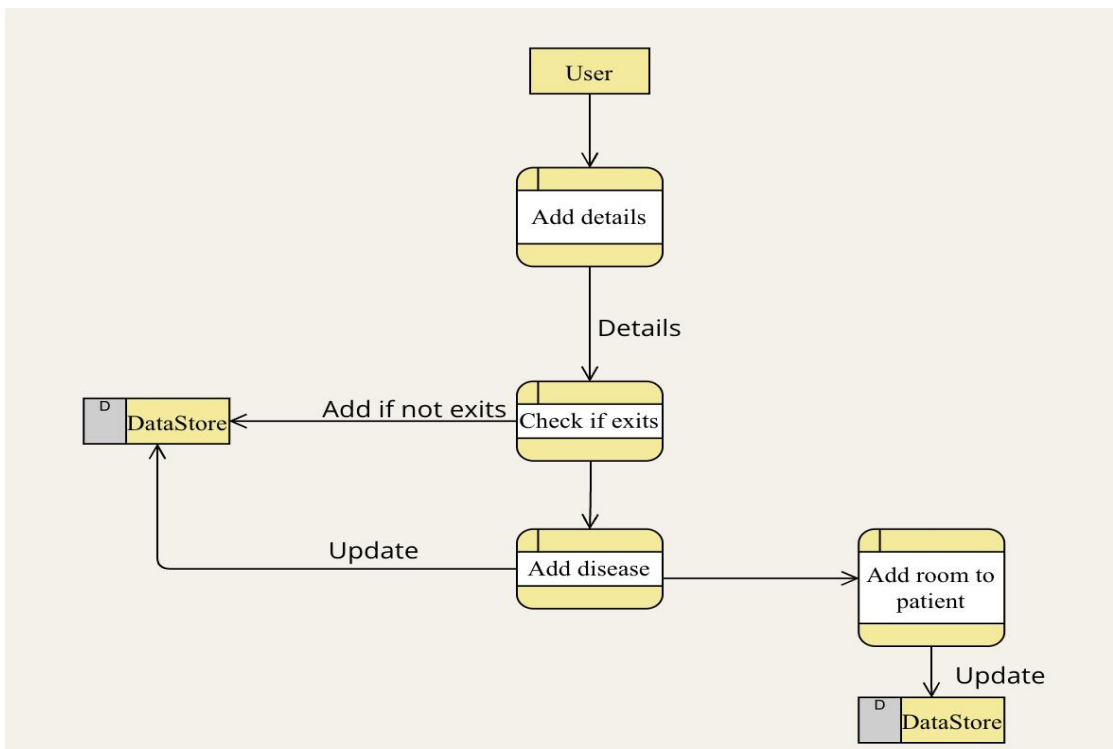


Figure 3.4 Level-2 Data Flow Diagram – Add Patient Module

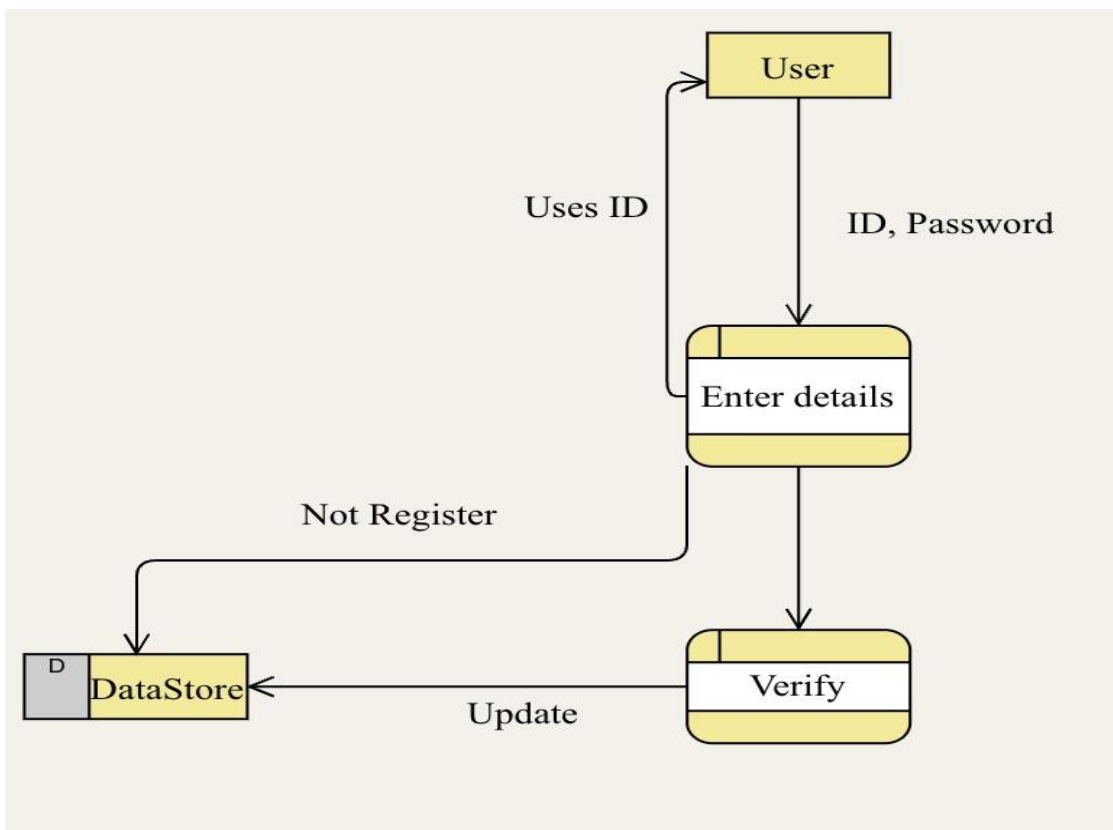


Figure 3.5 Level-2 Data Flow Diagram – User Login Module

3.2.3 Use Case Diagram

A use case diagram is a type of behavioural diagram in Unified Modeling Language (UML) that describes the behaviour of a system from an external point of view. It provides a visual representation of the interactions between actors (users) and a system, highlighting the functionality that the system provides to its users.

The key components of a use case diagram are actors, use cases, and relationships. Actors represent the users of the system, while use cases represent the actions or tasks that the system can perform. Relationships connect actors to use cases, indicating the interactions between them.

Use case diagrams can be used to model various systems, including software applications, business processes, and organizational structures. They are especially useful in identifying the functional requirements of a system, as well as the potential interactions between different actors and use cases.

Some common types of relationships that are typically represented in a use case diagram include:

1. Association: Indicates that an actor interacts with a use case.
2. Extend: Indicates that a use case can be extended by another use case under certain conditions.
3. Include: Indicates that a use case includes the functionality of another use case.

Use case diagrams are an important tool for system analysts, software engineers, and business analysts. They provide a clear and concise way to represent the functionality of a system, aiding in the design, development, and testing of software systems.

The following is the Use Case Diagram for the proposed system design: (next page)

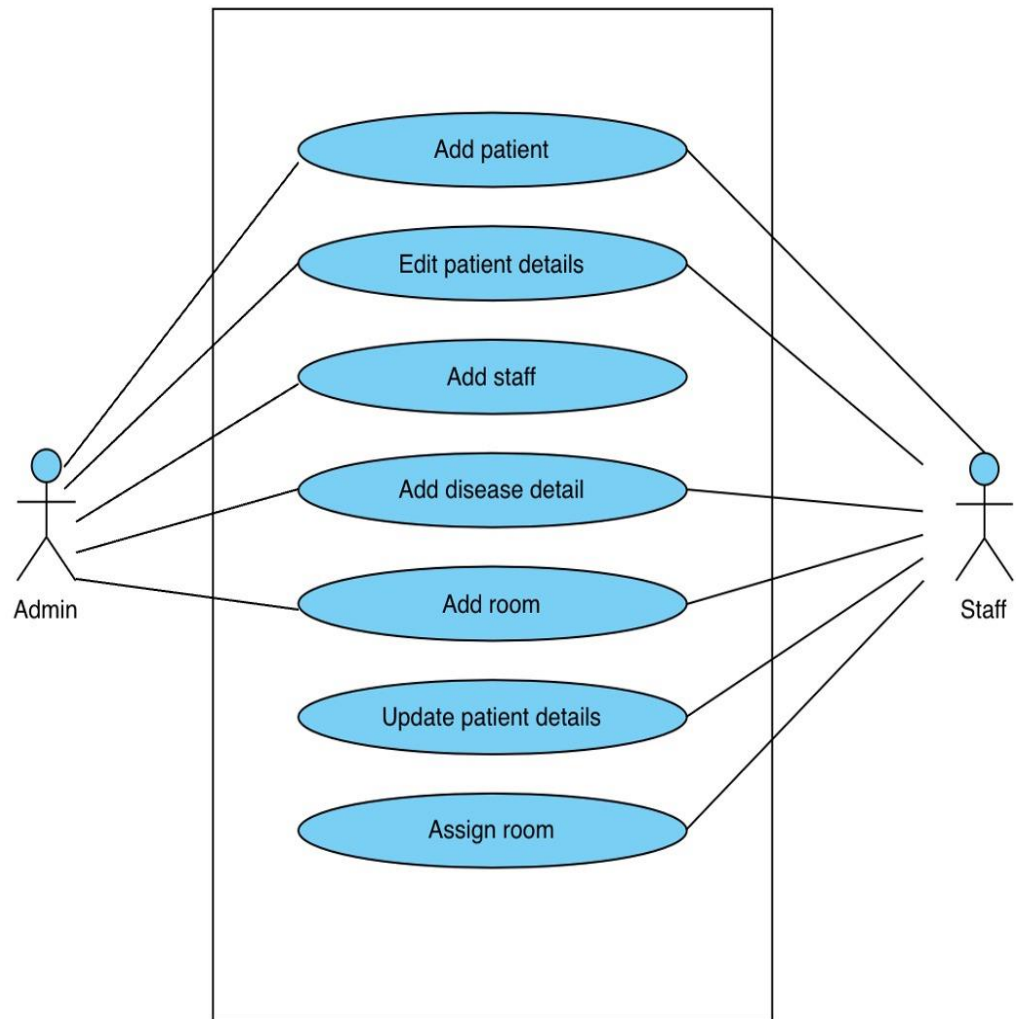


Figure 3.6 Use Case Diagram

Use Case Description

1. Admin: The Admin can manage system settings, such as adding, updating, or deleting diseases, patients, and rooms. The Admin can also manage user accounts, including adding or removing users, resetting passwords, and assigning roles.

Flow of Events:

- a. The Admin navigates to the System Settings page.
- b. The Admin selects the type of setting they want to manage (e.g., diseases, patients, rooms, users).
- c. If adding a new setting:
 - The Admin enters the necessary information.

- The Admin clicks the "Add" button.
- d. If updating an existing setting:
 - The Admin selects the setting to update.
 - The Admin edits the necessary information.
 - The Admin clicks the "Update" button.
- e. If deleting an existing setting:
 - The Admin selects the setting to delete.
 - The Admin clicks the "Delete" button.
- f. If managing user accounts:
 - The Admin selects the User Settings page.
 - The Admin selects the user to manage.
 - The Admin performs the necessary action (e.g., reset password, assign role).

Preconditions:

- The Admin must be logged in to the hospital management system.
- The Admin must have appropriate permissions to manage system settings.

Postconditions:

- The changes made by the Admin are saved in the system.

2. Staff: The Staff can manage patient information, such as adding, updating, or deleting patient records. The Staff can also assign patients to rooms and update their treatment status.

Flow of Events:

- a. The Staff navigates to the Manage Patients page.
- b. The Staff selects the patient they want to manage.
- c. If adding a new patient:
 - The Staff enters the necessary information.
 - The Staff clicks the "Add" button.
- d. If updating an existing patient:
 - The Staff edits the necessary information.
 - The Staff clicks the "Update" button.
- e. If deleting an existing patient:
 - The Staff selects the patient to delete.
 - The Staff clicks the "Delete" button.

- f. If assigning a patient to a room:
 - The Staff selects the patient to assign.
 - The Staff selects an available room.
 - The Staff clicks the "Assign" button.
- g. If updating a patient's treatment status:
 - The Staff selects the patient to update.
 - The Staff updates the treatment status (e.g., medication, surgery, discharge).
 - The Staff clicks the "Update" button.

Preconditions:

- The Staff must be logged in to the hospital management system.
- The Staff must have appropriate permissions to manage patient information.

Postconditions:

- The changes made by the Staff are saved in the system.

3.2.4 Sequence Diagram

A sequence diagram is a type of UML diagram used to visualize the interactions between objects or components in a system over time. It shows the sequence of messages passed between the different objects or components involved in a particular use case or scenario.

A sequence diagram consists of vertical lines representing the lifelines of the objects or components involved in the interaction, with messages passing horizontally between them. Each message indicates a specific action that is performed by an object or component in response to a previous message.

Sequence diagrams can be used to model a wide variety of interactions within a system, such as the flow of control in a use case scenario, the order in which processes or services are executed, or the interaction between a user interface and a backend system.

Sequence diagrams are useful for both high-level design and detailed implementation, as they provide a clear and concise visualization of how different parts of a system interact with each other. They are especially helpful for identifying potential issues or bottlenecks in a system's performance, and can aid in the testing and debugging of complex systems.

The Sequence Diagram for the proposed system is as follows:

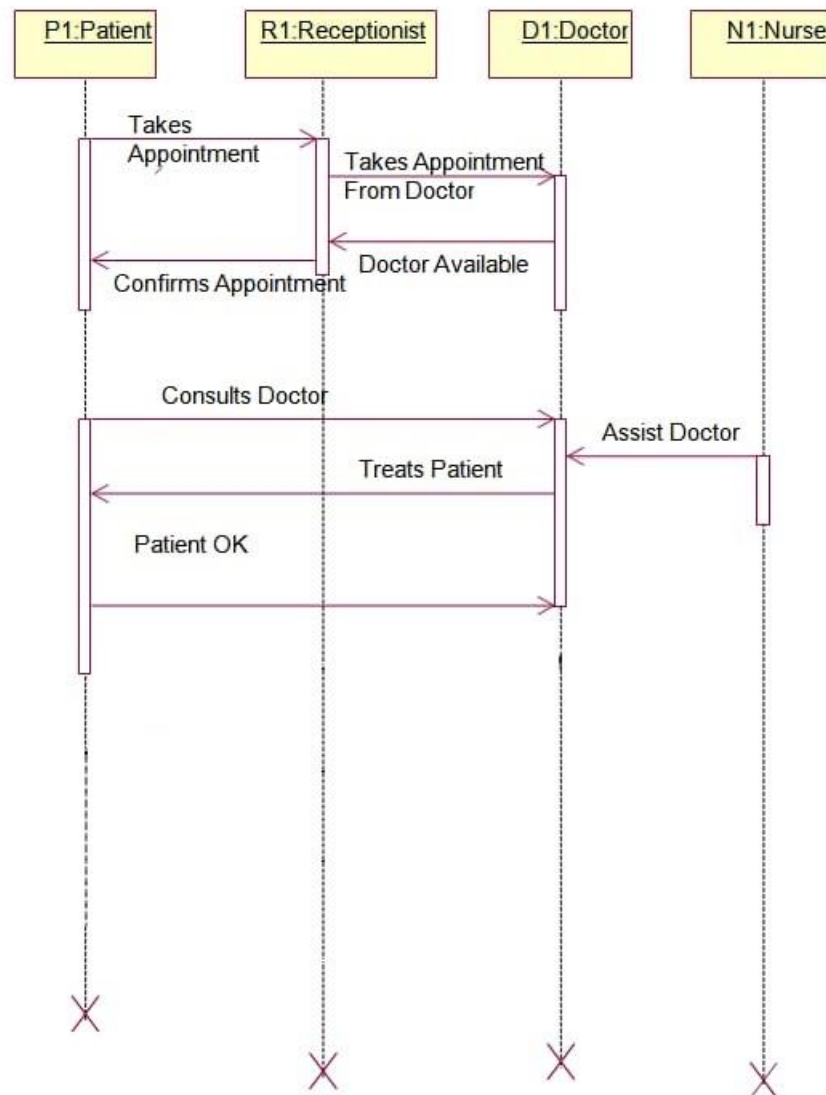


Figure 3.7 Sequence Diagram

3.2.5 Class Diagram

A class diagram is a type of UML diagram that depicts the structure of a system by showing the classes, their attributes, methods, and the relationships between them. It is a fundamental tool for software developers, as it helps to visualize the overall architecture of the system being designed.

In a class diagram, classes are represented as boxes with three sections. The top section contains the class name, the middle section lists the class attributes, and the bottom section lists the class methods. Class attributes are the properties or characteristics of a class, while class methods are the functions that can be performed on the class.

The relationships between classes are also shown in a class diagram. The most common types of relationships are inheritance, association, and aggregation. Inheritance is a relationship where one class inherits the properties and methods of another class. Association is a relationship between two classes, indicating that they are connected in some way. Aggregation is a special type of association that represents a "has-a" relationship, where one class contains or is composed of another class.

Class diagrams are useful for designing and communicating the architecture of a software system, and can be used in both the analysis and design phases of software development. They can help to identify potential issues with the system's design, such as redundant or unnecessary classes, and can aid in the creation of efficient and maintainable software systems.

The Class Diagram for ProMed is as follows:(next page)

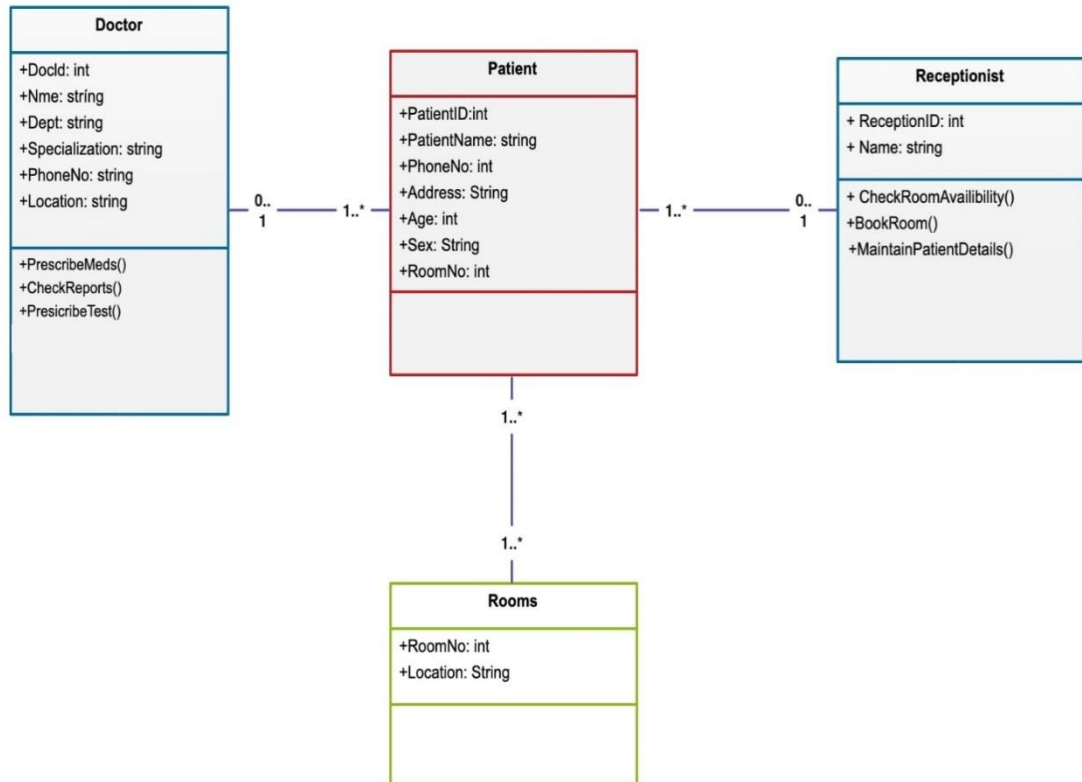


Figure 3.8 Class Diagram

3.3 Design and Test Steps

The design and test steps are crucial components of the software development process. They ensure that the software being developed meets the requirements and functions as intended. Here are the general steps involved in the design and testing phases:

1. Design Phase

During the design phase, the following steps are typically followed:

- Requirements gathering: This is the process of gathering and documenting the requirements of the system. This can be done through interviews, questionnaires, and other techniques.
- System design: This involves creating a design specification for the system. This includes creating a high-level architecture of the system, defining the software components, and determining how they will interact with each other.
- Detailed design: This involves creating a detailed design of each component in the system. This includes specifying the interfaces, data structures, algorithms, and any other relevant details.

2. Test Phase

During the test phase, the following steps are typically followed:

- Test planning: This involves creating a test plan that outlines the testing strategy, test objectives, and test cases to be executed.
- Test case development: This involves creating test cases based on the test plan. The test cases should be designed to ensure that all aspects of the software are thoroughly tested.
- Test execution: This involves executing the test cases and recording the results. Any defects or issues found during testing are documented and tracked.
- Defect management: This involves managing and tracking the defects found during testing. The defects are prioritized, assigned to developers, and tracked until they are resolved.

Overall, the design and testing phases are critical to the success of a software development project. By following these steps, software developers can ensure that the software being developed meets the requirements and functions as intended.

The project design was carried out in three phases which are as follows –

- Designing the Frontend.
- Creating the Backend.

- Integrating both Frontend and Backend.

Developing a robust frontend user interface is very important as it is the medium with which the user will interact with its data.

When done integrating the system, the testing procedure starts. System testing is the process of checking whether the developed system is working according to the objective and requirement.

Types of Testing:

Testing is an integral part of software development. It helps ensure that the software meets the desired quality and functional requirements. There are various types of testing that can be performed on software, including:

1. Unit Testing:

Unit testing is a type of testing that focuses on testing individual units or components of the software. It involves writing automated tests for each unit and checking if they work as expected.

2. Integration Testing:

Integration testing involves testing the interaction between different modules or components of the software. It is performed after unit testing to ensure that the different components of the software work together as expected.

3. System Testing:

System testing involves testing the entire software system as a whole. It is performed after integration testing to ensure that the entire system meets the functional and non-functional requirements.

4. Acceptance Testing:

Acceptance testing is performed to ensure that the software meets the requirements of the end-user. It is typically performed by the end-user or a representative of the end-user.

5. Regression Testing:

Regression testing is performed after making changes to the software to ensure that the existing functionality of the software is not impacted by the changes.

6. Performance Testing:

Performance testing is performed to ensure that the software performs well under different load conditions. It helps identify the maximum number of users or requests

that the software can handle without performance degradation.

7. Security Testing:

Security testing involves testing the software for vulnerabilities and weaknesses that can be exploited by malicious actors. It helps ensure that the software is secure and protects the user's data and privacy.

Moreover, in order to ensure the quality and functionality of software applications, the following Testing Procedures are also followed:

Black-Box Testing:

Black-box testing is a testing method where the tester has no knowledge of the internal workings of the software being tested. The tester only focuses on the inputs and outputs of the system, testing its functionality without any knowledge of the code behind the software. The purpose of black-box testing is to verify that the software works as expected from the user's perspective.

Advantages of Black-Box Testing:

1. The tester doesn't require any knowledge of the software's internal workings.
2. Tests are performed from the user's perspective, ensuring that the software meets the user's requirements.
3. Black-box testing can help find issues with the software that are not apparent from the code.

Disadvantages of Black-Box Testing:

1. The tester is limited to testing only what the software should do and not what it actually does.
2. Black-box testing can be time-consuming and requires a thorough understanding of the software's requirements.
3. It may be difficult to isolate the root cause of an issue with the software.

White-Box Testing:

White-box testing is a testing method where the tester has knowledge of the internal workings of the software being tested. The tester has access to the code and can test the software at the code level. The purpose of white-box testing is to verify that the software is performing as expected from the developer's perspective.

Advantages of White-Box Testing:

1. The tester has access to the code and can test the software at a granular level.

2. White-box testing can help identify coding errors and issues that could lead to vulnerabilities.
3. The test coverage can be measured more accurately.

Disadvantages of White-Box Testing:

1. The tester needs to have knowledge of the software's internal workings.
2. White-box testing can be more time-consuming than black-box testing.
3. White-box testing may not be effective in testing the software from the user's perspective.

In conclusion, both black-box and white-box testing methods are valuable and have their advantages and disadvantages. Depending on the software being developed, one or both of these testing methods can be utilized to ensure the quality and functionality of the software.

3.4 Algorithms and Pseudo Code

An algorithm is a set of instructions designed to solve a specific problem or perform a specific task. It is a step-by-step procedure that describes how to accomplish a task in a finite amount of time. Algorithms can be implemented in a variety of ways, including computer programs, flowcharts, or natural language descriptions.

In computer science, algorithms are used to perform tasks such as searching and sorting data, calculating mathematical functions, and solving complex problems. They are an essential part of computer programming, as they provide a blueprint for developers to follow when creating software.

A well-designed algorithm is efficient, reliable, and easy to understand. It should be able to handle a wide range of input data, and produce accurate and consistent results every time it is executed.

Furthermore, Pseudocode is a method of writing out a computer program or algorithm in plain language, with some syntax that resembles the structure of a programming language. Pseudocode is not a programming language itself, but rather an informal way of describing the logic of a program before it is written in a specific programming language.

The purpose of pseudocode is to help programmers plan and organize their code before they begin writing it. By breaking down a program into smaller, more manageable steps, pseudocode can help identify potential issues or logic errors in a program before it is implemented in code. Additionally, pseudocode can serve as a form of documentation for a program, making it easier for others to understand and modify the code in the future.

Pseudocode typically uses standard programming concepts such as conditional statements

(if/else), loops (for/while), and functions. However, the syntax used in pseudocode is not as strict as that of a programming language, and there are no formal rules for how it should be written. As such, pseudocode can be written in a variety of ways, depending on the preferences of the programmer.

3.4.1 Routing Implementation

The Express framework is used to handle routing for your application. Express is a powerful tool that allows you to easily handle HTTP requests and responses. It provides a simple and flexible way to define routes for your application.

You have implemented routing in your project by creating a separate file for each major functionality of your application, such as patient management or system settings. Each file defines the routes and the corresponding handlers for each HTTP request method (e.g., GET, POST, PUT, DELETE). For example, for patient management, you have a file named **patientRoutes.js** that defines the following routes:

- **/patients** (GET): retrieves a list of all patients
- **/patients/:id** (GET): retrieves a specific patient by ID
- **/patients** (POST): adds a new patient to the system
- **/patients/:id** (PUT): updates an existing patient
- **/patients/:id** (DELETE): deletes a patient from the system

Each route is associated with a handler function that defines the behaviour of the application when the route is accessed. For example, the **/patients** (GET) route handler might retrieve a list of patients from the database and render them to the client-side as JSON. The **/patients** (POST) route handler might receive a new patient object from the client-side, add it to the database, and then redirect the client to the newly added patient's page.

By using Express routing, you have organized your code in a modular way that makes it easy to maintain and update. You can add new functionality to your application by simply adding a new file with its own set of routes and handlers. Additionally, you have used middleware functions to handle authentication and other common tasks that can be reused across multiple routes, reducing code duplication and making your application more efficient.

```
var login = require('./routes/login');
var users = require('./routes/users');
var appRoute = require('./routes/app');
var patients = require('./routes/patients');
var settings = require('./routes/settings');
var diseases = require('./routes/diseases');
var rooms = require('./routes/rooms');

app.use('/', login);
app.use('/', appRoute);
app.use('/', users);
app.use('/', patients);
app.use('/', settings);
app.use('/', diseases);
app.use('/', rooms);
```

Figure 3.9 – Routing Implementation

3.4.2 Database Designing:

The database schema consists of tables such as **Patients**, **Diseases**, **Rooms**, and **Users**. The **Patients** table stores information such as patient ID, name, contact information, assigned room, and disease ID. The **Diseases** table stores disease information such as disease ID, disease name, and its severity level. The **Rooms** table stores room information such as room ID, room type, and the current patient assigned to the room. The **Users** table stores user information such as username, password, and user type (admin or staff).

The database design also includes relationships between the tables. For example, the **Patients** table has a foreign key that references the **Diseases** table, indicating which disease the patient is diagnosed with. Similarly, the **Patients** table also has a foreign key that references the **Rooms** table, indicating which room the patient is assigned to. The relationships between the tables ensure that the patient information is organized and can be easily accessed when required.

To implement the database design, you have used a NoSQL database such as MongoDB. MongoDB provides a flexible data model that allows for easy storage and retrieval of data. You have also used an Object Data Modeling (ODM) library such as Mongoose to define the schema and perform operations such as CRUD (Create, Read, Update, Delete) on the database.

Overall, the database design implementation in your hospital management project ensures efficient management and organization of patient information, enabling smooth and effective healthcare services.

```
var UserSchema = mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
});
```

Figure 3.10 – User Schema

```
var PatientSchema = mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  dateOfBirth: {
    type: String,
    required: true,
  },
  sex: {
    type: Boolean,
    required: true,
    default: true
  },
  hospitalNumber: {
    type: String,
    required: true,
    unique: true
  },
  diseases: {
    type: Array,
    default: []
  },
  score: {
    type: Number,
    required: true,
    default: 0
  },
  room: {
    type: String,
    required: true,
    default: 'noroom'
  },
  lastUpdate: {
    type: Number,
    required: true
  }
});
```

Figure 3.11 – Patient Schema

```
var RoomSchema = mongoose.Schema({
  name: {
    type: String,
    unique: true,
    required: true,
  },
  availability: {
    type: Boolean,
    required: true,
    default: false
  }
});
```

Figure 3.12 – Room Schema

```
var DiseaseSchema = mongoose.Schema({
  name: {
    type: String,
    unique: true,
    required: true
  },
  score: {
    type: Number,
    required: true,
    default: 0
  }
});
```

Figure 3.13 – Disease Schema

3.5 Testing Process

Software testing can be stated as the process of verifying and validating that software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Table 3.1 – Login Testing

| S. No. | Input Values | Test Case | Conditions | Result |
|--------|--------------|---------------|------------------------|---------|
| 1 | Email | Empty | Valid Email | Success |
| 2 | Email | Already Exist | Unique Email | Success |
| 3 | Password | Empty | Valid Password | Success |
| 4 | Password | Wrong | Wrong Password | Success |
| 5 | Password | Length (<2) | More than 2 characters | Success |

Table 3.2 – Add Patient Testing

| S. No. | Input Values | Test Case | Conditions | Result |
|--------|--------------|----------------------|--------------|---------|
| 1 | First Name | Empty | Valid Name | Success |
| 2 | Last Name | Empty | Valid Name | Success |
| 3 | Hospital No. | Empty | Valid ID | Success |
| 4 | DOB | Empty / (DD/MM/YYYY) | Valid Format | Success |
| 5 | Gender | - | - | - |
| 6 | Disease | Select min. 1 | Not Empty | Success |

Table 3.3 –Add Rooms Testing

| S. No. | Input Values | Test Case | Conditions | Result |
|--------|--------------|---------------|-------------|---------|
| 1 | Room Name | Empty | Valid Name | Success |
| 2 | Room Name | Already Exist | Unique Name | Success |

CHAPTER 4

RESULTS / OUTPUTS

The following are the individual outputs and functions that are performed by the hospital management system:

1. Login Page:

The login page is the initial point of entry for both administrators and staff members. The purpose of the login page is to ensure the security and privacy of patient data by allowing only authorized users to access the system.

The login page requires users to enter their username and password, which are then verified against the database of authorized users. If the credentials are correct, the user is granted access to the system and directed to their respective dashboard, either the admin dashboard or the staff dashboard.

If the user enters incorrect login credentials, an error message is displayed, indicating that the username or password is incorrect. The user will then be prompted to re-enter their credentials.

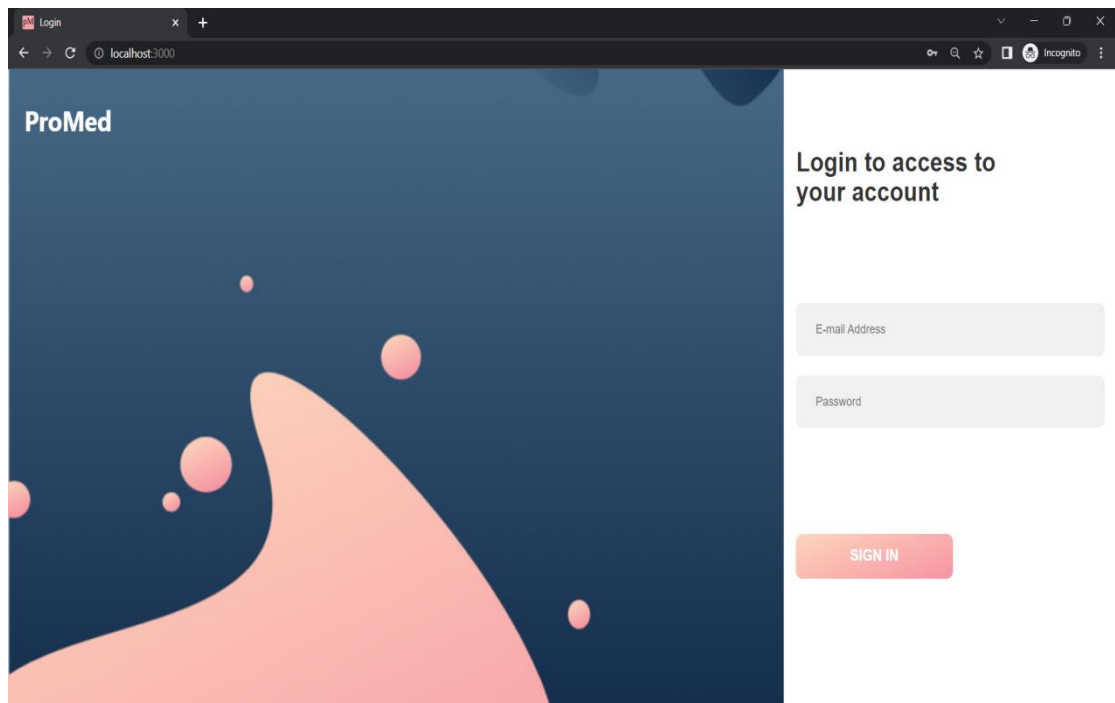


Figure 4.1 –Login Page

2. Dashboard Page:

The dashboard page serves as the main interface for both administrators and staff members to manage and view important information related to patient care and hospital operations.

The dashboard provides an overview of the system's key features and functions, such as patient management, room management, and disease management. It allows administrators and staff members to easily navigate to different areas of the system and perform various tasks, such as adding or editing patient information, updating room availability, and logging diseases and treatments.

The dashboard also provides important metrics and statistics related to hospital operations, such as the number of patients currently admitted, the number of available rooms, and the prevalence of different diseases. This information can be used by hospital administrators to make informed decisions about resource allocation and treatment prioritization.

Overall, the dashboard page is an essential component of the hospital management system, providing users with a comprehensive view of the system's features and allowing them to efficiently manage patient care and hospital operations.

a. Statistics Display:

The total stats regarding the total number of Allotted Patients with Rooms, Waiting Patients and Available Rooms is displayed in this fragment.

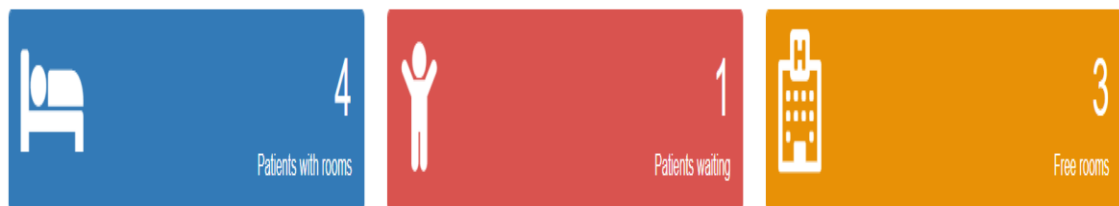


Figure 4.2 –Statistics Fragment (Dashboard Page)

b. Assigned Patients:

The details of patients that have assigned with Rooms and are attended to frequently. The red marker shows if the patient hasn't been attended to for a while.

The screenshot shows a web interface titled "Assigned Patients". It features a search bar labeled "Search patient in room...". Below the search bar is a table with columns: "no.", "Patients with rooms", "Room", and "Score". The table contains two rows: one for patient 103 (Aditya Brar) in room H01-01 with a score of 100, and another for patient 105 (Khushi Nair) in room H01-02 with a score of 100. The "Score" column has a red background. At the bottom left of the table is an information icon (i).

| no. | Patients with rooms | Room | Score |
|-----|---------------------|--------|-------|
| 103 | Aditya Brar | H01-01 | 100 |
| 105 | Khushi Nair | H01-02 | 100 |

Figure 4.3 –Assigned Patients Fragment (Dashboard Page)

c. Waiting Patients:

The patients details which have been onboarded but no rooms have been assigned yet are displayed here.

The screenshot shows a web interface titled "Waiting Patients". It features a search bar labeled "Search patient waiting...". Below the search bar is a table with columns: "no.", "Patients waiting", and "Score". The table contains three rows: one for patient 104 (Abhiram Bhattacharyya) with a score of 65, one for patient 101 (Aadesh mane) with a score of 55, and one for patient 102 (Aruna Mittal) with a score of 15. The "Score" column has a red background for the first two rows and a yellow background for the third. At the bottom left of the table is an information icon (i).

| no. | Patients waiting | Score |
|-----|-----------------------|-------|
| 104 | Abhiram Bhattacharyya | 65 |
| 101 | Aadesh mane | 55 |
| 102 | Aruna Mittal | 15 |

Figure 4.4 –Waiting Patients Fragment (Dashboard Page)

d. Available Rooms:

The details of Free Rooms available to assign are displayed here.

The screenshot shows a web interface titled "Available Rooms". It features a search bar labeled "Search room...". Below the search bar is a table with a single column labeled "Free rooms". The table contains five rows: H01-03, H01-04, H01-05, H02-01, and H02-02. At the bottom left of the table is an information icon (i).

| Free rooms |
|------------|
| H01-03 |
| H01-04 |
| H01-05 |
| H02-01 |
| H02-02 |

Figure 4.5 –Available Rooms Fragment (Dashboard Page)

3. Add Patient Page:

This page allows administrators and staff members to add new patient information into the system.

The page includes a form that collects key patient information such as their name, date of birth, gender, contact information, and medical history. Administrators and staff members can also assign the patient to a specific room and assign a doctor to oversee their care.

In addition to collecting patient information, the "Add Patient" page also provides error handling to ensure that all required fields are filled out and that the entered information is valid. Once all required fields are filled out, the information is saved to the database and the patient record is added to the system.

The "Add Patient" page is a critical component of the hospital management system, allowing administrators and staff members to efficiently add new patients to the system and manage their care.

a. Adding Patient Details:

Add patient

First Name

Enter first name...

Last Name

Enter last name...

Hospital no.

Enter Hospital no...

Date of birth

DD/MM/YYYY

Sex

☐ Male ☐ Female

Figure 4.6 –Add Patient Fragment (Add Patients Page)

b. Updating the Disease Details:

Search disease

| Disease | Score | Diagnosis |
|------------------|-------|--------------------------|
| Conjunctivitis | 15 | <input type="checkbox"/> |
| Diabetes | 55 | <input type="checkbox"/> |
| Hepatitis a & e | 25 | <input type="checkbox"/> |
| Hepatitis b & c | 10 | <input type="checkbox"/> |
| Immunodeficiency | 25 | <input type="checkbox"/> |

Add patient

Figure 4.7 –Add Diseases Fragment (Add Patients Page)

4. System Settings:

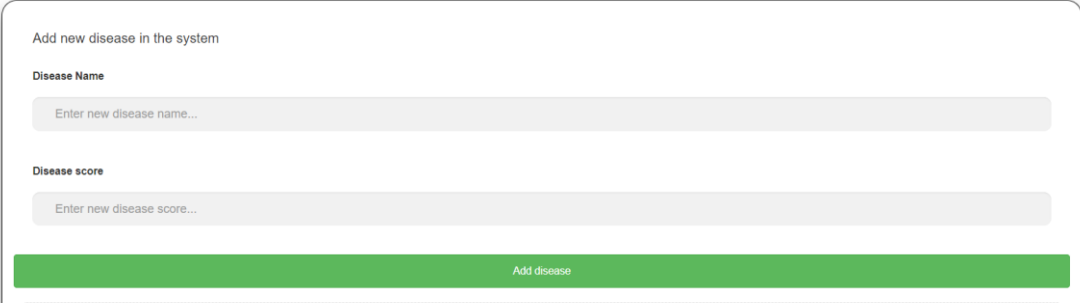
This is a centralized location for administrators to manage the system settings that control how the application behaves and processes data.

This page provides administrators with the ability to add, edit, and delete the different categories and options that make up the hospital management system. Some of the settings that can be managed from this page include:

- Diseases: A list of all the diseases that are stored in the system, which can be used for diagnosis and treatment purposes.
- Patients: A list of all the patients currently registered in the system.
- Rooms: A list of all the rooms within the hospital that can be assigned to patients.
- Users: A list of all the users who have access to the system, including administrators, staff members, and doctors.

From the "System Settings" page, administrators can easily add new diseases, patients, rooms, and users as well as edit and delete existing ones. This centralized approach to managing system settings allows for better organization and control over the data used by the hospital management system.

a. Add Diseases to Database:



The screenshot shows a web form titled "Add new disease in the system". It contains two input fields: "Disease Name" with a placeholder "Enter new disease name..." and "Disease score" with a placeholder "Enter new disease score...". Below these fields is a green button labeled "Add disease".

Figure 4.8 –Add Diseases Fragment (System Settings Page)

b. Delete Diseases from Database:

Delete diseases from the system

Search disease...

| Disease | Score | Select |
|------------------|-------|--------------------------|
| Conjunctivitis | 15 | <input type="checkbox"/> |
| Diabetes | 55 | <input type="checkbox"/> |
| Hepatitis a & e | 25 | <input type="checkbox"/> |
| Hepatitis b & c | 10 | <input type="checkbox"/> |
| Immunodeficiency | 25 | <input type="checkbox"/> |

Delete diseases

Figure 4.9 –Delete Diseases Fragment (System Settings Page)

c. Add Rooms to the Database:

Add new room in the system

Room name

Enter new room name...

Add room

Figure 4.10 –Add Rooms Fragment (System Settings Page)

d. Delete Rooms from Database:

Delete rooms from the system

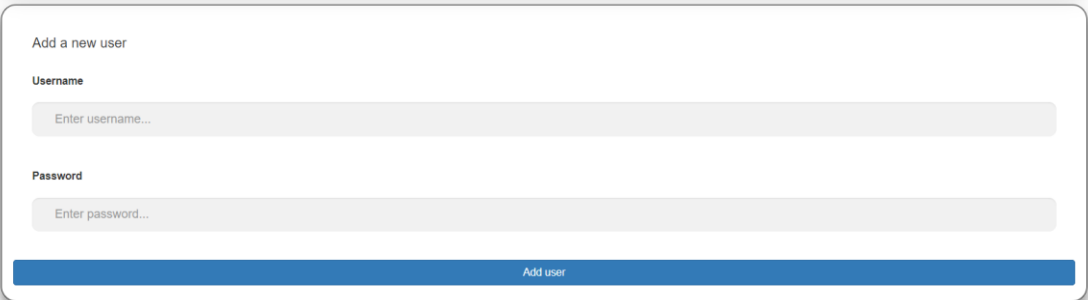
Search room...

| Room | Select |
|--------|--------------------------|
| H01-01 | <input type="checkbox"/> |
| H01-02 | <input type="checkbox"/> |
| H01-03 | <input type="checkbox"/> |
| H01-04 | <input type="checkbox"/> |
| H01-05 | <input type="checkbox"/> |
| H02-01 | <input type="checkbox"/> |
| H02-02 | <input type="checkbox"/> |

Delete rooms

Figure 4.11 –Delete Rooms Fragment (System Settings Page)

e. Adding Users:



A form titled "Add a new user" with two input fields: "Username" and "Password". The "Username" field has a placeholder "Enter username..." and the "Password" field has a placeholder "Enter password...". A blue button labeled "Add user" is at the bottom right.

Add a new user

Username

Enter username...

Password

Enter password...

Add user

Figure 4.12 –Add Diseases Fragment (System Settings Page)

CHAPTER 5

CONCLUSION

In conclusion, ProMed developed using MERN stack is a comprehensive solution for managing hospital operations. The system has been designed to provide a seamless experience for both patients and hospital staff, streamlining the process of managing patient data, treatment plans, and administrative tasks.

The development of the system involved a detailed analysis of the hospital's requirements, which was used to identify the necessary features and functionalities. The team followed a systematic approach, including feasibility studies, requirement analysis, and software design and testing, to ensure that the final product meets the needs of the users.

The system features a user-friendly interface and easy navigation that allows for quick access to critical information, helping to optimize workflows and reduce the time and effort needed to manage hospital operations. With features such as the "Add Patient" page, "System Settings," and the Dashboard, the system offers a comprehensive solution for managing patient data and administrative tasks.

Overall, the Hospital Management System developed using MERN stack provides a robust, efficient, and reliable solution for managing hospital operations. The project demonstrates the importance of effective software development practices, including requirement analysis, testing, and user feedback, in delivering successful solutions that meet the needs of the end-users.

CHAPTER 6

REFERENCES

- [1] **W3schools.com**. (2023). JavaScript Tutorial. [online] Available at: <https://www.w3schools.com/js> [Accessed 1 February 2022].
- [2] **Apna College**(2020). [online] Available at: <https://www.youtube.com/watch?v=1lEssrLxt7E&feature=youtu.be> [Accessed 1 February 2023].
- [3] **React.dev**. (2022). *React* – A JavaScript library for building user interfaces. [online] Available at: <https://react.dev/> [Accessed 1 February 2023].
- [4] **Node.js**. (2023). Documentation | Node.js. [online] Available at: <https://nodejs.org/en/docs/> [Accessed 1 February 2023].

CHAPTER 7

APPENDICES

7.1 Appendix 1 – ReactJS and Handlebars

Introduction

ReactJS is a popular JavaScript library used for building user interfaces. It was developed by Facebook and released in 2013 as an open-source project. ReactJS follows a component-based approach to building UIs, where each component represents a small part of the user interface. Components can be reused and combined to build complex UIs.

Features

Some key features of ReactJS include:

- Virtual DOM: ReactJS uses a virtual DOM to update the actual DOM efficiently.
- JSX: ReactJS allows developers to write HTML-like syntax in JavaScript code using JSX.
- One-way data binding: ReactJS follows a unidirectional data flow, where data flows from parent to child components.
- Server-side rendering: ReactJS can be used for server-side rendering to improve page load times and SEO.

Advantages

Some advantages of using ReactJS include:

- Fast rendering: ReactJS uses a virtual DOM and efficient diffing algorithms to update the DOM quickly.
- Reusability: ReactJS components can be reused in multiple parts of an application, making development faster and more efficient.
- Community: ReactJS has a large and active community, with many libraries and tools available to make development easier.

Disadvantages

Some disadvantages of using ReactJS include:

- Steep learning curve: ReactJS can be difficult to learn for developers who are new to JavaScript or component-based UI development.
- Complexity: ReactJS can become complex as applications grow, making it harder to maintain and debug.

- Limited functionality: ReactJS only provides the view layer of an application, so developers need to use other libraries or frameworks to handle state management, routing, and other functionality.

Handlebars:

Handlebars is a popular templating engine for creating dynamic HTML templates. It allows developers to create templates with placeholders for dynamic data, and then use JavaScript code to fill in those placeholders with actual data.

To use Handlebars in a web application, you typically need to include the Handlebars library in your project, along with any other dependencies. Once you've done that, you can create Handlebars templates by writing HTML code with special placeholders, called "handlebars expressions," which look like this: `{{variable}}`. The **variable** inside the double braces represents the dynamic data that will be filled in at runtime.

Handlebars also supports more advanced features, such as conditionals, partials, and helpers, which can be used to create more complex templates.

7.2 Appendix 2 – NodeJS

Node.js is an open-source, cross-platform runtime environment that enables developers to run JavaScript code outside of a web browser. It's built on the V8 JavaScript engine, the same engine that powers Google Chrome, and provides an event-driven, non-blocking I/O model that makes it ideal for building scalable, high-performance applications.

Node.js has a vast ecosystem of modules and packages available through the npm (Node Package Manager) registry, which makes it easy to extend its functionality and reuse existing code. It's commonly used for building server-side applications, such as web servers, APIs, and microservices, as well as command-line tools and desktop applications.

Some of the key features of Node.js include:

- Asynchronous I/O: Node.js provides an event-driven, non-blocking I/O model that allows multiple I/O operations to be performed concurrently without blocking the main thread, which makes it ideal for building scalable, high-performance applications.
- CommonJS modules: Node.js uses the CommonJS module system, which provides a simple and standardized way of organizing and sharing code between different files and modules.
- Built-in modules: Node.js provides a set of built-in modules, such as **http**, **fs**, and **path**, that provide core functionality for building web servers, file systems, and other

common tasks.

- **npm:** Node.js comes with npm (Node Package Manager) pre-installed, which provides access to a vast ecosystem of modules and packages that can be easily installed and managed.
- **Cross-platform:** Node.js is available on a wide range of platforms, including Windows, macOS, Linux, and many others.

Overall, Node.js is a powerful and flexible runtime environment that's well-suited for building a wide range of applications, from simple command-line tools to complex web servers and APIs. Its event-driven, non-blocking I/O model and extensive ecosystem of modules and packages make it a popular choice among developers for building scalable, high-performance applications.

STUDENT PROFILE



Name: Divyanshu Mishra

Enrolment No.: 191B102

Email: divyanshu8770@gmail.com

Address: A-one colony, Guna, M.P (473001)

University: Jaypee University of engineering and Technology, Guna

Contact: 8770716784



Name: Ganesh Adavkar

Enrolment No.: 191B106

Email: ganeshadavkar01@gmail.com

Address: Muhal colony, Bamori, Guna, M.P.(473105)

University: Jaypee University of Engineering and Technology, Guna

Contact: 8959494418



Name: Shrish Sharma

Enrolment No.: 191B249

Email: shrish0608@gmail.com

Address: Naveen School Road, Vindhyachal Colony, Guna, 473001

University: Jaypee University of Engineering and Technology, Guna

Contact: 9669977046