# ACKNOWLEDGEMENT

We express our sincere thanks to the management and **Dr. J. JANET M.E., Ph.D.,** Principal, Sri Krishna College of Engineering and Technology, Coimbatore for providing us the facilities to carry out this project work.

We are thankful to **Dr. K.SASI KALA RANI M.E., Ph.D.,** Professor and Head, Department of Computer Science and Engineering, for her continuous evaluation and comments given during the project work.

We express our deep sense of gratitude to our supervisor **Dr. M.ROHINI M.E, Ph.D.,** Assistant Professor, Department of Computer science and Engineering for her valuable advice, guidance and support during the course of our project work.

We express our heartfelt sense of gratitude and thanks to our beloved parents, family and friends who have helped during the project course.

## ABSTRACT

The Online Hospital Management System (OHMS) is a comprehensive application designed to revolutionize and improve various facets of hospital operations, ultimately enhancing the overall patient experience. At its core, the system provides three indispensable functionalities: patient appointment scheduling, efficient medical record management, and seamless facility booking, all of which collectively contribute to the hospital's operational efficiency and the well-being of its patients. Patient Appointments are a cornerstone feature of the OHMS, enabling patients to effortlessly schedule appointments online. The OHMS further simplifies the often complex task of Medical Record Management. Healthcare providers gain easy access to patient medical records, allowing them to electronically update and maintain accurate patient information. The system seamlessly integrates with the hospital's Electronic Health Record (EHR) system, ensuring the availability of up-to-date patient data. Moreover, it facilitates the recording of patient diagnoses, personalized treatment plans, and prescription details, thus guaranteeing the delivery of tailored and well-documented healthcare services. Facility Booking services are yet another vital component of the OHMS, offering patients the convenience of reserving medical facilities within the hospital premises. Patients can select from a range of specialized treatment areas, procedure rooms, or surgical theaters, all while choosing their preferred dates and times for reservations. In summary, the Online Hospital Management System (OHMS) stands as a pivotal digital platform that significantly enhances the patient experience. It achieves this by facilitating convenient online appointment scheduling, simplifying medical record management, and providing an efficient facility booking service. Ultimately, the OHMS optimizes hospital operations and guarantees the delivery of high-quality healthcare services, setting a new standard for modern hospital management.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| ABBREVIATION | FULL FORM |
|---|---|
| SB | SPRING BOOT |
| API | APPLICATION PROGRAMMING INTERFACE |
| JWT | JAVA WEB TOKENS |
| NPM | NODE PACKAGE MANAGER |
| JSX | JAVASCRIPT XML (**Extensible Markup Language**) |
| MUI | MATERIAL UI |
| JS | JAVA SCRIPT |

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW OF HOSPITAL MANAGEMENT

The Hospital Management System is an advanced healthcare administration solution designed to revolutionize and optimize hospital operations. This innovative platform offers a multifaceted approach to managing hospital resources, patient information, and medical facilities, ensuring a seamless healthcare experience. At its core, it provides three essential functionalities: patient care and appointment management, medical records and information management, and facility booking for medical procedures. Patients can conveniently schedule appointments, healthcare providers can securely access and update patient information, and medical facilities can be efficiently reserved. The system employs a modern user interface, enhancing usability and flexibility, making it suitable for healthcare institutions of varying sizes. Additionally, it supports a wide range of fundamental healthcare operations, aiding in resource allocation and predictive analytics to enhance patient care and streamline hospital operations. In essence, the Hospital Management System is a transformative solution that prioritizes efficiency, data accuracy, and patient-centric care, empowering healthcare institutions to deliver high-quality services in a rapidly evolving healthcare landscape.

## 1.2 KEY FEATURES

1. **Patient Appointment Management:** Efficiently schedule and manage patient appointments, including real-time availability of healthcare providers, appointment booking, and reminders.

2. **Electronic Health Records (EHR):** Securely store and manage patient medical records electronically, ensuring easy access, data accuracy, and comprehensive patient histories.

3. **Facility and Resource Booking:** Allow for the reservation and management of medical facilities, surgical theaters, equipment, and medical staff scheduling for procedures and surgeries.

4. **Patient Registration:** Simplify the patient registration process, capturing essential demographic and insurance information for accurate billing and medical record creation.

5. **Inventory and Pharmacy Management:** Manage hospital inventory efficiently, including tracking medications, medical supplies, and equipment, ensuring availability when needed.

6. **Doctor and Staff Management:** Maintain comprehensive records of healthcare providers and staff, including their qualifications, schedules, and roles within the hospital.

7. **Pharmacy Integration:** Seamlessly integrate with hospital pharmacies for prescription fulfillment and medication dispensing, ensuring accurate and timely medication administration

## 1.3 BENEFITS

**Enhanced Patient Care:** Improved appointment scheduling and medical record management result in better patient outcomes and experiences.

**Efficiency and Productivity:** Automation reduces administrative tasks, boosting productivity and allowing more time for patient care.

**Reduced Errors:** Digital records and automation minimize medical errors and billing inaccuracies, enhancing patient safety.

**Cost Savings:** Streamlined operations and reduced administrative overhead lead to significant cost savings.

**Improved Decision-Making:** Real-time data and analytics empower informed decisions about resource allocation and patient care.

**Patient Convenience:** Online appointment scheduling and medical record access improve patient convenience and reduce wait times.

**Data Security:** Prioritize patient data security, meeting privacy regulations such as HIPAA.

**Scalability**: Systems can scale with the growth of healthcare institutions.

**Resource Optimization:** Efficiently allocate healthcare providers, equipment, and facilities.

**Inventory Control:** Efficient inventory management ensures timely access to medications and medical supplies.

These benefits collectively modernize healthcare administration, improve patient care, and optimize hospital operations.

# CHAPTER – 2

# REQUIREMENTS

## 2.1 FUNCTIONAL REQUIREMENTS

**Patient Appointment Management:** The system should enable the scheduling and management of patient appointments. It must provide real-time visibility into healthcare provider availability, allowing users to book, modify, or cancel appointments conveniently.

**Facility Booking and Resource Management:** The system should allow for the booking and management of medical facilities, surgical theaters, equipment, and medical staff scheduling for procedures and surgeries. It should ensure efficient resource allocation and availability tracking.

**Patient Registration and Demographics:** The system should facilitate patient registration, capturing essential demographic and insurance information. It must ensure accurate patient identification for billing and medical record creation.

**Inventory and Pharmacy Management:** If applicable, the system should manage hospital inventory efficiently, including tracking medications, medical supplies, and equipment. It must ensure the availability of medical resources when needed.

**Doctor and Staff Management:** The system should maintain comprehensive records of healthcare providers and staff, including their qualifications, schedules, and roles within the hospital.

**Data Security and Access Control:** The system should implement stringent data security measures to safeguard patient information. It must ensure role-based access control to protect sensitive patient data.

**Mobile Accessibility:** The system should offer mobile capabilities for healthcare

providers and staff to access patient information, appointments, and medical records on mobile devices.

## 2.2 TECHNICAL REQUIREMENTS

**React Front-End**: The user interface should be developed using React, ensuring a responsive and intuitive design that facilitates user interactions and provides a seamless experience across devices.

**MySQL Database**: A MySQL relational database should be employed to store and manage product details, order data, customer information, and other relevant datasets securely.

**Spring Boot Back-End**: The back-end application should be developed using Spring Boot, providing robust security features, RESTful API endpoints, and efficient data processing capabilities.

**Security:** The system should implement stringent security measures, including data encryption, secure authentication mechanisms, and role-based access control to protect sensitive information and ensure the privacy of user data.

**Microservices:** Microservices architecture is a modern approach to software development where an application is broken down into a collection of small, independently deployable services that work together to deliver the application's functionality.

## 2.3 HARDWARE SPECIFICATIONS

- Processor Type : Intel Core Pentium
- Speed : 2.20 GHz
- RAM : 4 GB
- Hard Disk : 256 GB Hard Drive
- Keyboard : 101/102 Standard Keys

- Mouse : Optical Mouse

## 2.4 SOFTWARE SPECIFICATIONS

- Operating System : OS Independent
- Front End : React JS
- Coding Language : JavaScript
- Database : MySQL

# CHAPTER-3

# MODULE DESCRIPTION

## 3.1 DATA

The Data module of the Hospital Management System encompasses the storage and organization of essential data elements. This includes patient records, medical histories, healthcare provider information, appointment schedules, billing data, and inventory details. The module ensures secure and structured storage, retrieval, and manipulation of data across the entire system, facilitating seamless interactions between various modules and users.

## 3.2 PATIENT MANGEMENT MODULE

The Patient Management module is a central component of the HMS, dedicated to managing patient-related information. It provides functionalities for patient registration, demographics, medical history storage, and real-time access to patient records by authorized healthcare providers. This module plays a crucial role in streamlining patient care, enhancing patient experiences, and ensuring accurate medical records for diagnosis and treatment.

## 3.3 KEY IMPLEMENTATION CONSIDERATIONS

In the implementation phase of the Hospital Management System, several critical considerations are addressed to ensure the success and efficiency of the system:

### 3.3.1 Database Design

Efficient and secure database design is crucial for the storage and retrieval of patient data, medical records, and system information. Proper schema, relationships, and optimization techniques are applied to ensure data accuracy and rapid access.

### 3.3.2 User Interface (UI) Design

A user-friendly and intuitive user interface (UI) design is essential for healthcare providers, staff, and patients to navigate the system easily. Responsive design ensures accessibility across various devices, enhancing usability and convenience.

### 3.3.3 Security Measures

Robust security measures, including authentication, authorization, and data encryption, are implemented to protect sensitive patient information. Ensuring compliance with healthcare data privacy regulations, such as HIPAA, is a top priority.

### 3.3.4 Integration with External Systems

The system integrates seamlessly with external healthcare systems, laboratories, insurance providers, and government health agencies. Effective data exchange ensures comprehensive patient care and compliance with industry standards.

### 3.3.5 Scalability and Performance

The system is designed to accommodate a growing number of patients, healthcare providers, and medical facilities. Performance optimization techniques, such as caching and code optimization, are employed to maintain system responsiveness.

## 3.4 APPOINTMENT MANAGEMENT MODULE

The Appointment Management module streamlines the scheduling and management of patient appointments. It provides patients with the ability to book, modify, or cancel appointments online, while healthcare providers can view their schedules and patient appointments. The module ensures efficient resource allocation and reduces wait times for patients.

.

### 3.5 ADMINISTRATION MANAGEMENT MODULE

The Administration Management module encompasses administrative tasks such as user management, access control, and system configuration. It enables administrators to create and manage user accounts, define roles and permissions, and configure system settings to meet the hospital's specific requirements. This module plays a critical role in maintaining the integrity and security of the HMS while ensuring efficient system administration.

# CHAPTER – 4

# DESIGN OF PROPOSED SYSTEM

## 4.1 SYSTEM DESIGN

### 4.1.1 INPUT DESIGN

Input design is the process of converting the user-originated input to a computer-basedformat. The design decision for handling input specifies how data is accepted for computer processing. Input design is a part of overall system design that needs careful attention.

The collection of input data is the most expensive part of the systemdesign. Since the inputs must be planned in such a way to get the relevant information, extreme care is taken to obtain the pertinent information. If the data going into the system is incorrect then the processing and outputs will magnify these errors. The goal ofdesigning input data is to make data entry as easy, logical and free from errors as possible. The following are the objectives of input design:

- To produce a cost-effective method of input.
- To make the input forms understandable to the end users.
- To ensure the validation of data inputs.

The nature of input data is determined partially during logical system design. However,the nature of inputs is made more explicit during the physical design. The impact of inputs onthe system is also determined. Effort has been made to ensure that input data remains accurate from the stage at which it is recorded and documented to the stage at which it is accepted by the computer. Validation procedures are also present to detect errors in data input, which is beyond control procedures. Validation procedures are designed to check each record, data item or field against certain criteria.

### 4.1.2 OUTPUT DESIGN

The output is designed in such a way that it is attractive, convenient and informative. Forms are designed in python with various features, which make the console output more pleasing. As the outputs are the most important sources of information to the users, better design should improve the system's relationships with us and also will help in decision-making. Form design elaborates the way output is presented and the layout available for capturing information.

In this project, the output is designed in the form reports. The system is designed to generate various user-friendly reports to help the business process. Following are the differentreports supported:

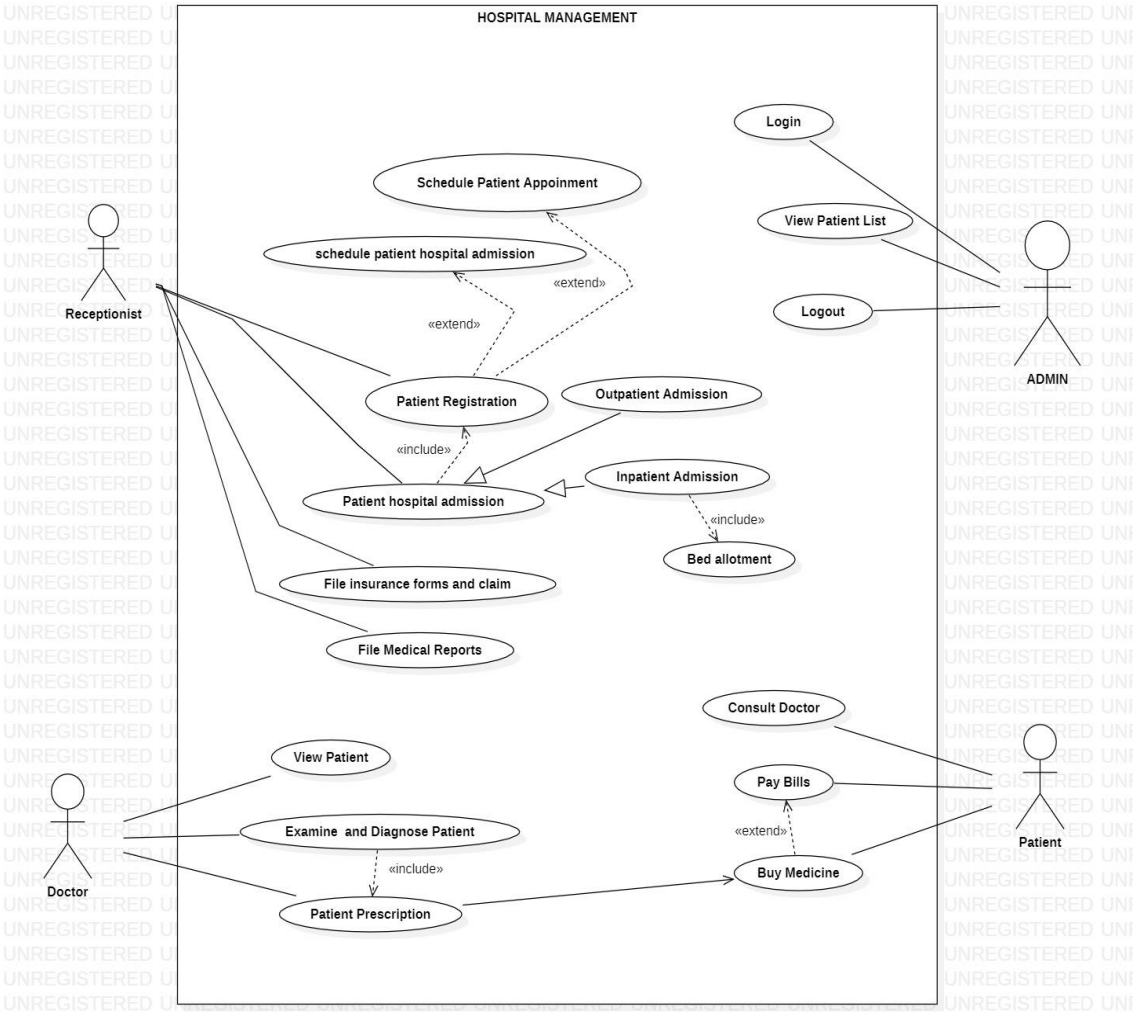



**Fig 4.1:** Use Case

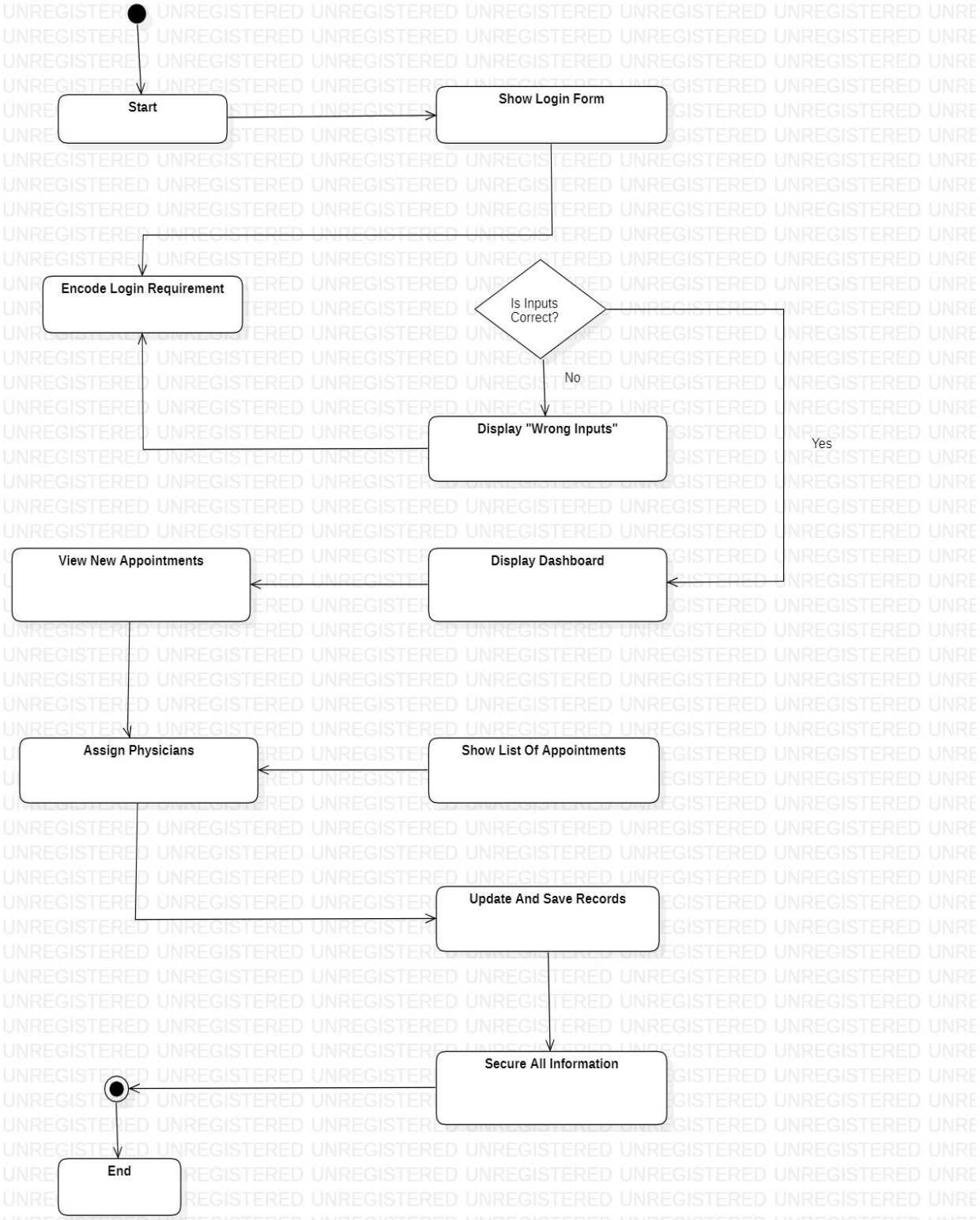



**Fig 4.2:** Activity Diagram

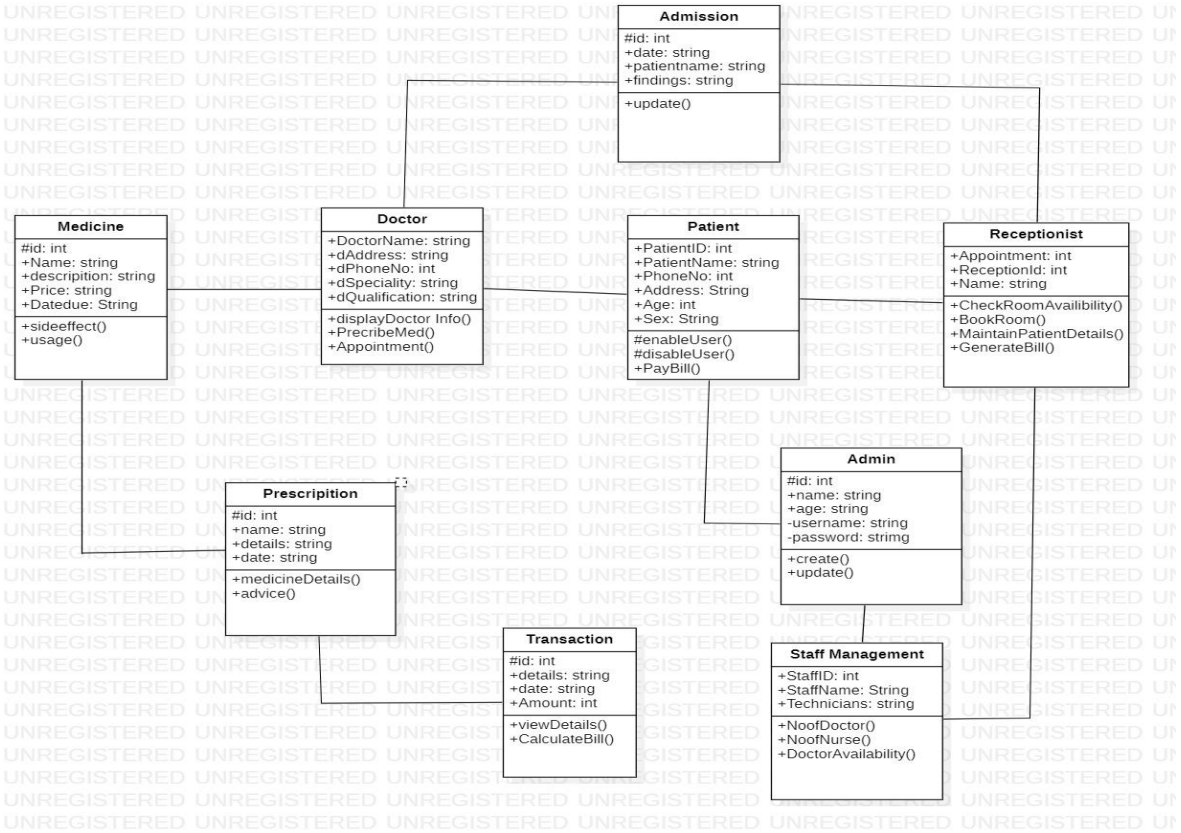



**Fig 4.3:** Class Diagram

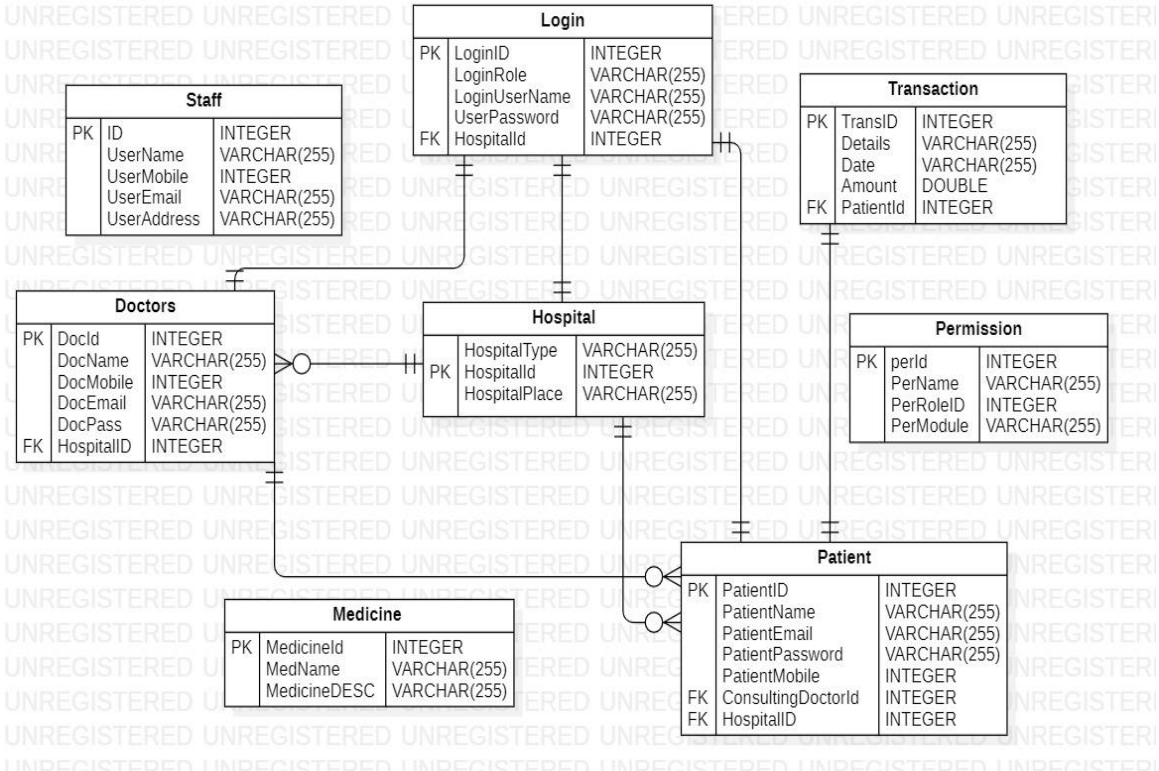



**Fig 4.4:** ER Diagram

## 4.2 DESIGN CONSTRAINTS AND STANDARDS

### DESIGN CONSTRAINTS

**Budget Constraints:**

- The HMS project must adhere to a predefined budget, which may limit the selection of software, hardware, and development resources.
- Budget constraints should be carefully considered when making decisions about technology stack, infrastructure, and customization.

**Resource Limitations:**

- Constraints related to human resources, time, and technology infrastructure can impact the project's scope and timeline.
- The availability of skilled healthcare IT professionals, developers, and support staff should be considered when planning the project.

**Compatibility:**

- The HMS must be compatible with the existing software and hardware infrastructure used by the hospital or healthcare organization.
- Compatibility testing should be conducted to ensure seamless integration with legacy systems and medical devices.

**Scalability:**

- The system should be designed to handle both current and future hospital requirements, accommodating potential growth in the number of patients, medical records, and healthcare services.
- Scalability planning should consider factors like increased patient load and the addition of new hospital facilities.

**Data Privacy and Security:**

- Compliance with data privacy regulations, such as GDPR (General Data Protection Regulation) or HIPAA (Health Insurance Portability and Accountability Act), is critical.
- The system must implement robust security measures to protect sensitive patient health information (PHI) from unauthorized access or breaches.

**User Training and Adoption:**

Users, including healthcare professionals and administrative staff, may require training to effectively use the HMS.

- The system's design should prioritize user-friendliness and provide clear training materials and support to ensure user adoption**.**

**Integration Requirements:**

- The HMS may need to integrate with other enterprise systems commonly used in healthcare settings, such as electronic health record (EHR) systems, billing software, and laboratory information systems.
- Ensuring seamless data exchange and interoperability between systems is crucial for efficient hospital operations.

## STANDARDS AND BEST PRACTICES

**User Interface (UI) Standards:**

- Adherence to healthcare data standards, such as HL7 (Health Level Seven) and DICOM (Digital Imaging and Communications in Medicine), is essential for data interoperability and exchange with external healthcare systems.

**Data Standards:** Use of standardized data formats and structures for consistent and interoperable data exchange.

**Security Standards:**

- The HMS should implement security standards and protocols, including encryption (SSL/TLS), secure authentication, and audit trails, to safeguard patient data.

**Regulations:**

- The system design should align with healthcare regulatory standards specific to the region or country in which it operates, such as FDA regulations in the United States or NHS Digital standards in the India.

**Accessibility Standards:** Ensure that the portal is accessible to individuals with disabilities, following accessibility standards such as WCAG (Web Content Accessibility Guidelines).

**Performance Standards:**

Set performance benchmarks and standards to ensure the system responds quickly.

**Backup and Disaster Recovery:** Implement data backup and disaster recovery standards to prevent data loss and ensure business continuity.

**Audit Trails and Logging:** Maintain comprehensive audit trails and logs to track changes and user activities within the portal, meeting compliance and security requirements.

**Documentation:** Maintain thorough documentation, including user manuals, system architecture diagrams, and code comments, to facilitate maintenance and troubleshooting.

**Testing Standards:** Follow rigorous testing standards, including unit testing, integration testing, and user acceptance testing, to ensure the system's reliability and functionality.

**Regulatory Compliance:** Ensure that the portal complies with industry-specific regulations and standards, such as FDA regulations for pharmaceutical inventory or ISO standards for quality management.

## 4.3 DESIGN ALTERNATIVES

When designing a Hospital Management System (HMS), it's crucial to explore various design alternatives tailored to meet the unique requirements of healthcare institutions. One option is to consider Commercial Off-the-Shelf (COTS) HMS software, which offers pre-built solutions that can be customized to align with specific healthcare needs. Alternatively, Open-Source Software presents a flexible and cost-effective choice, allowing extensive customization to suit diverse healthcare requirements. Cloud-Based Hospital Management Systems offer scalability and accessibility advantages, making them suitable for healthcare facilities of all sizes. Hybrid solutions provide a balanced approach by combining on-premises and cloud elements, offering control and flexibility.

For a modern and mobile-centric approach, adopting a Mobile-First Solution ensures accessibility across a range of devices, accommodating the mobility needs of healthcare

professionals. Integration of Internet of Things (IoT) technology enables real-time tracking and monitoring of medical equipment and patient vital signs, enhancing overall healthcare management. Leveraging the power of Artificial Intelligence (AI) and Machine Learning (ML) can provide predictive analytics for disease diagnosis, patient care optimization, and resource allocation. Thoughtfully evaluating these design alternatives will lead to the selection of an optimal HMS solution that effectively supports hospital operations and enhances patient care.

## 4.4 FLOW DIAGRAM

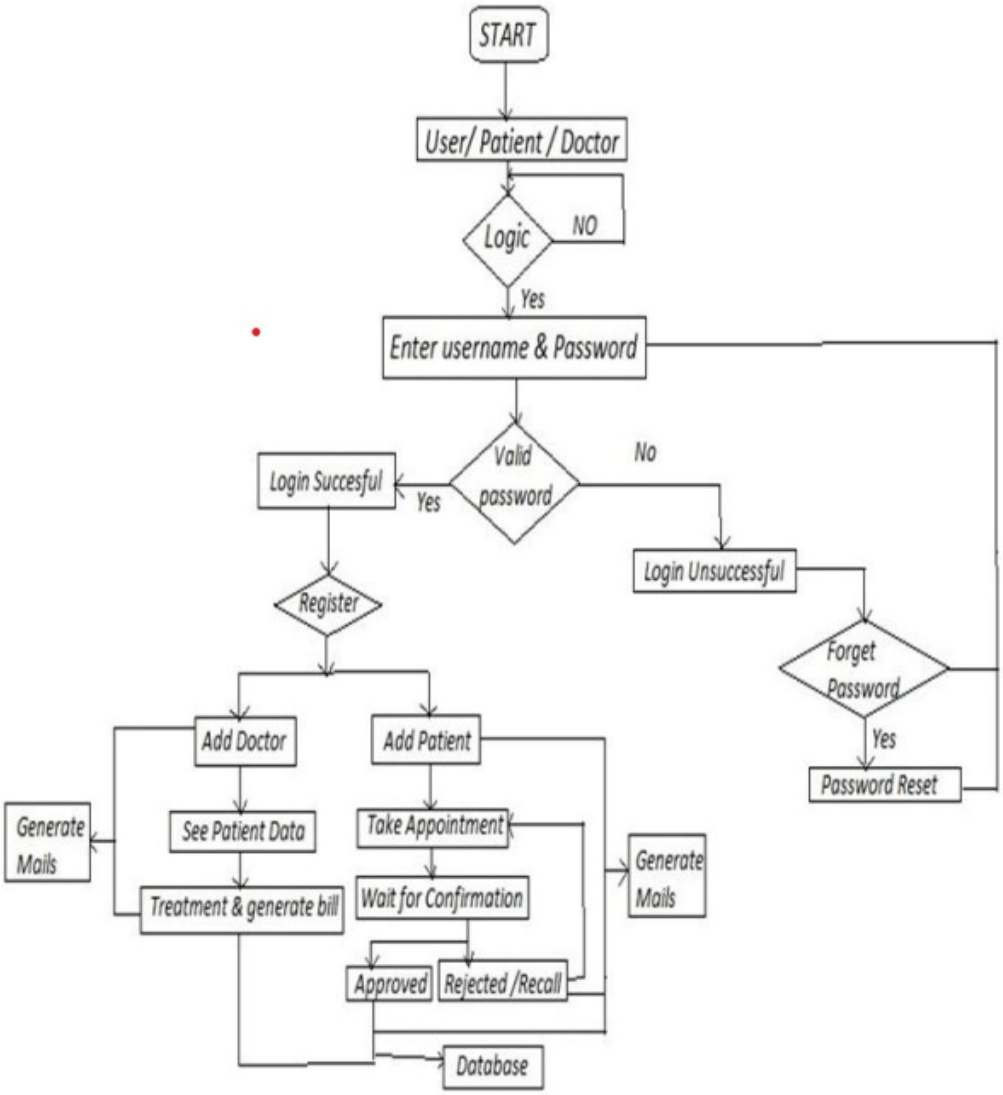



**Fig 4.5:** OVERALL WORKFLOW DIAGRAM

## 4.5 DATABASE DESIGN

### 4.5.1 Database Schema

- Provide an ER (Entity-Relationship) diagram depicting the relationships between tables.
- Define primary and foreign keys for data integrity.
- Discuss how data redundancy and normalization are handled.

### 4.5.2 Data Storage

- Choose an appropriate database system (e.g., MongoDB, PostgreSQL) based on the project's requirements.
- Define storage capacity and data retention policies.
- Implement backup and recovery procedures to prevent data loss.

### 4.5.3 Data Access

- Specify data access methods (e.g., RESTful APIs) for the React app to communicate with the database.
- Ensure proper handling of data retrieval, updates, and deletions.

### 4.5.4 Data Validation and Constraints

- Specify data validation rules and constraints to ensure data integrity.
- Define allowed data formats, ranges, and mandatory fields for each table.
- Implement validation checks at both the application and database levels.

### 4.5.5 Indexing and Query Optimization

- Create appropriate indexes on frequently queried columns to improve database performance.
- Optimize database queries by using query analysis tools and database profiling.
- Implement caching mechanisms for frequently accessed data.

## 4.6 SECURITY

### 4.6.1 Data Encryption

- Use encryption protocols (e.g., SSL/TLS) to secure data in transit.
- Implement encryption mechanisms (e.g., AES) to protect sensitive data at rest.

### 4.6.2 Authentication

- Employ multi-factor authentication (MFA) for enhanced login security.
- Monitor and log authentication attempts for security auditing.

### 4.6.3 Access Controls

- Enforce role-based access control (RBAC) to limit access to sensitive features and data.
- Implement session management and token expiration policies.

### 4.6.4 Security Testing

- Perform regular security assessments, including penetration testing and code reviews.
- Keep libraries and dependencies up to date to patch security vulnerabilities

### 4.6.5 Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) Protection

- Implement input validation and output encoding to prevent XSS attacks.
- Utilize anti-CSRF tokens to protect against CSRF attacks.
- Consider using security libraries or middleware to automate these protections.

### 4.6.6 Session Management

- Implement secure session management mechanisms, such as session timeouts and secure cookie attributes.
- Use random and complex session IDs to prevent session fixation attacks.

## 4.7 CONTENT DELIVERY NETWORK

Implementing a Content Delivery Network (CDN) plays a pivotal role in optimizing the performance, scalability, and security of your inventory management portal React app. To leverage the full potential of a CDN, it's essential to take into account various factors. Firstly, the geographical distribution of CDN edge servers should align with your user base, ensuring minimal latency and faster content delivery globally. Prioritizing security, ensure the CDN supports HTTPS, employs SSL/TLS certificates for data encryption, and offers security features like DDoS protection and web application firewalls.

Cache invalidation and content versioning are critical for serving updated content effectively, and dynamic content caching should be configured thoughtfully. Employing an origin shield within the CDN architecture can ease the load on your origin server and boost cache efficiency. Monitor cache hit ratios closely and adapt caching policies accordingly. Optimize content for mobile devices using the CDN and consider routing and load balancing capabilities for traffic distribution.

Edge scripting and serverless functions can add versatility to content delivery tasks. Comprehensive monitoring and analytics tools provided by the CDN can offer valuable insights, helping you fine-tune performance and cost-efficiency. Lastly, ensure security headers and policies are set at the CDN level, and establish traffic redundancy and failover plans to maintain service continuity. By addressing these aspects, your inventory management portal can harness the full potential of CDNs for a seamless, secure, and high-performing user experience on a global scale.

# CHAPTER – 5

## IMPLEMENTATION OF PROPOSED SYSTEM

### 5.1 INTRODUCTION

The implementation of the proposed Hospital Management Portal represents a pivotal phase in the project's lifecycle, where the conceptual design is translated into a functional and robust system. This process involves a series of systematic steps to bring the portal to life and ensure it meets the specified requirements and objectives.

First and foremost, the technical architecture of the system is established. This includes setting up the necessary hardware infrastructure, selecting appropriate software technologies, and configuring the development environment. Critical decisions are made regarding the choice of programming languages, frameworks, and databases to align with the project's scalability, security, and performance goals.

Database design plays a central role in implementation, with the creation of the database schema, tables, and relationships. This is crucial for efficiently storing and managing product, order, and warehouse data. Furthermore, the implementation phase involves the development of the user interface (UI) to create an intuitive and user-friendly portal. Responsive design principles are applied to ensure accessibility across a range of devices, including mobile platforms.

Security measures are paramount during implementation, encompassing robust authentication and authorization mechanisms to protect sensitive data. Encryption protocols are implemented for data both in transit and at rest to fortify the system's overall security posture.

The integration with external systems is a key implementation consideration. This involves establishing seamless connections with payment gateways, shipping APIs, and potentially other third-party systems. It ensures smooth order processing and data consistency in integration scenarios.

## 5.2 FRAMEWORK USED

### FRONT END: REACT

React, also known as React.js or **ReactJS**, is an open-source JavaScript library developed by Facebook. It is primarily used for building user interfaces (UIs) for web applications. React allows developers to create reusable UI components and manage the dynamic rendering of these components based on changes in data or user interactions. React follows a component-based architecture, which promotes modular and maintainable code.

### FEATURES OF REACT:

### 1. COMPONENT BASED ARCHITECTURE

React is built around the concept of reusable components. Each component represents a part of the user interface and can be composed and nested to create complex UIs. This promotes modularity and maintainability in code.

### 2. VIRTUAL DOM

React uses a virtual representation of the DOM (Virtual DOM) to improve performance. When the state of a component changes, React calculates the difference between the current virtual DOM and the previous one, and then efficiently updates only the necessary parts of the actual DOM. This minimizes the number of direct manipulations to the DOM, resulting in faster updates.

### 3. DECLARATIVE SYNTAX

React 's declarative syntax allows developers to describe the desired UI state based on the current data. Developers specify what the UI should look like, and React handles the underlying updates, which simplifies the coding process and reduces the potential for bugs.

### 4. REACT NATIVE

While React is primarily used for web development, React Native is an extension of React that allows developers to build native mobile applications using the same concepts and

components. This enables code sharing and faster development across multiple platforms.

**5. JSX (JavaScript XML):**

JSX is a syntax extension that allows developers to write HTML-like code within their JavaScript code. JSX makes it easier to visualize and understand the structure of components, as well as how they relate to one another.

**6. UNIDIRECTIONAL DATA FLOW**

React enforces a unidirectional data flow, which means that data flows in a single direction—from parent components to child components. This simplifies the tracking of data changes and prevents unexpected side effects.

**7. STATE MANAGEMENT**

React components can manage their own internal state using the useState hook or class component state. This state represents data that can change over time, and when it changes, React automatically re-renders the component to reflect the updated data.

**8. PROPS**

Props are inputs passed to components from their parent components. They allow data to flow down the component tree, enabling customization and data sharing between components.

**9. LIFECYCLE METHODS**

Class components in React have lifecycle methods that allow developers to execute code at specific points in the component's life cycle. This includes actions like component initialization, updates, and cleanup.

**10. HOOKS**

Introduced in React 16.8, hooks allow developers to add state and other React features to functional components without class components. Hooks like useState , useEffect, and more provide a more streamlined way to manage state and side effects in functional components.

## 10. REACT ROUTER

React Router is a library that enables developers to implement routing and navigation in single-page applications. It helps manage different views or pages within a React application.

## 12. COMMUNITY AND ECOSYSTEM

React has a large and active community, which has led to the development of numerous third-party libraries, tools, and extensions that complement and enhance React's capabilities.

## 13. DEVLOPER TOOLS

React offers browser extensions and integrated developer tools that facilitate debugging, profiling, and inspecting React components and their state.

## JAVASCRIPT

JavaScript (JS) is a lightweight interpreted (or just-in-time compiled) programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.

JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation (via eval), object introspection (via for...in and Object utilities), and source-code recovery (JavaScript functions store their source text and can be retrieved through toString()). **JavaScript is *not* "Interpreted Java"**. Both "Java" and "JavaScript" are trademarks or registered trademarks of Oracle in the U.S. and other countries. However, the two programming languages have very different syntax, semantics, and use.

JavaScript is a high-level, often just-in time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for

working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

## FEATURES OF JAVASCRIPT

JavaScript is a versatile and widely used programming language primarily known for its role in web development. It's a client-side scripting language that allows you to add interactivity and dynamic behavior to websites. Here are some of the key features of JavaScript

### 1. INTERACTIVITY

JavaScript enables interactive web pages by responding to user actions. It can be used to create features like forms, buttons, sliders, and more, allowing users to interact with the content.

### 2. EVENT HANDLING

JavaScript can listen for and respond to various events like clicks, mouse movements, keyboard inputs, and more. This allows developers to create responsive and user-friendly interfaces.

### 3. DOM MANIPULATION

The Document Object Model (DOM) represents the structure of an HTML document. JavaScript can manipulate the DOM to add, modify, or remove elements and content on a webpage dynamically.

### 4. ASYNCHRONOUS PROGRAMMING

JavaScript supports asynchronous programming through techniques like callbacks, promises, and async/await. This allows operations such as fetching data from servers or performing animations without blocking the main thread.

### 5. CROSS-BROWSER COMPACTIBILITY

JavaScript works across different web browsers, making it a widely adopted language for web development.

### 6. FUNCTIONS AS FIRST-CLASS CITIZENS

Functions in JavaScript can be assigned to variables, passed as arguments to other functions, and returned from functions. This allows for powerful functional programming techniques.

### 7. CLOSURES

Closures allow functions to "remember" the variables from the outer scope even after they have finished executing. This enables more advanced programming patterns.

### 8. PROTOTYPAL INHERITANCE

JavaScript is dynamically typed, meaning that variables are not bound to a specific data type. They can hold different types of values during runtime.

### 9. DYNAMIC TYPING

JavaScript is dynamically typed, meaning that variables are not bound to a specific data type. They can hold different types of values during runtime.

### 10. MODULARITY

While JavaScript doesn't have built-in modules (prior to ES6), developers can achieve modularity through techniques like the Revealing Module Pattern or by using module bundlers like Webpack.

### 11. JSON

JSON is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It's often used for transmitting data between a server and a web application.

**12. RICH ECOSYSTEM**

JavaScript has a vast ecosystem of libraries and frameworks like React, Angular, and Vue.js for building complex web applications. It's also used for server-side development (Node.js) and even in non-browser environments (like IoT devices).

**BACK END: JAVA**

Java is a popular, versatile, and widely used programming language that was initially developed by Sun Microsystems (now owned by Oracle Corporation). It was first released in 1995 and has since become one of the most widely used programming languages for various purposes, including web development, mobile app development, enterprise software, and more.

Java is known for its "write once, run anywhere" philosophy, which means that code written in Java can run on different platforms without modification, if the platform has a Java Virtual Machine (JVM) installed. This platform independence is achieved through a combination of compiling Java source code into an intermediate form called bytecode and then executing that bytecode on the JVM. Java is a widely used programming language for coding web applications. It has been a popular choice among developers for over two decades, with millions of Java applications in use today.

Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform. It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies.

# CHAPTER - 6
# TESTING

## 6.1 INTRODUCTION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive. A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned and conducted systematically. The underlyingmotivation of program testing is to affirm software quality with methods that can economicallyand effectively apply to both strategic to both large and small-scale systems.

## 6.2 STRATEGIC APPROACH TO SOFTWARE TESTING

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behaviors, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral along streamlines that decrease the level of abstractionon each turn. A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested.

## 6.3 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing, we have is white box oriented and some modules the steps are conducted in parallel.

### 6.3.1 WHITE BOX TESTING

To follow the concept of white box testing we have tested each form. We have created independently to verify that Data flow is correct, and all conditions are exercised to check their validity.

- All loops are executed on their boundaries. This type of testing ensures that all independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity paths.

### 6.3.2 CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested so that each path that may generate on a particular condition is traced to uncover any possible errors.

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable were declared. The definition-use chain method was used in this type of testing.

### 6.3.3 DATA FLOW TESTING

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was used in this type of testing.

### 6.3.4 LOOP TESTING

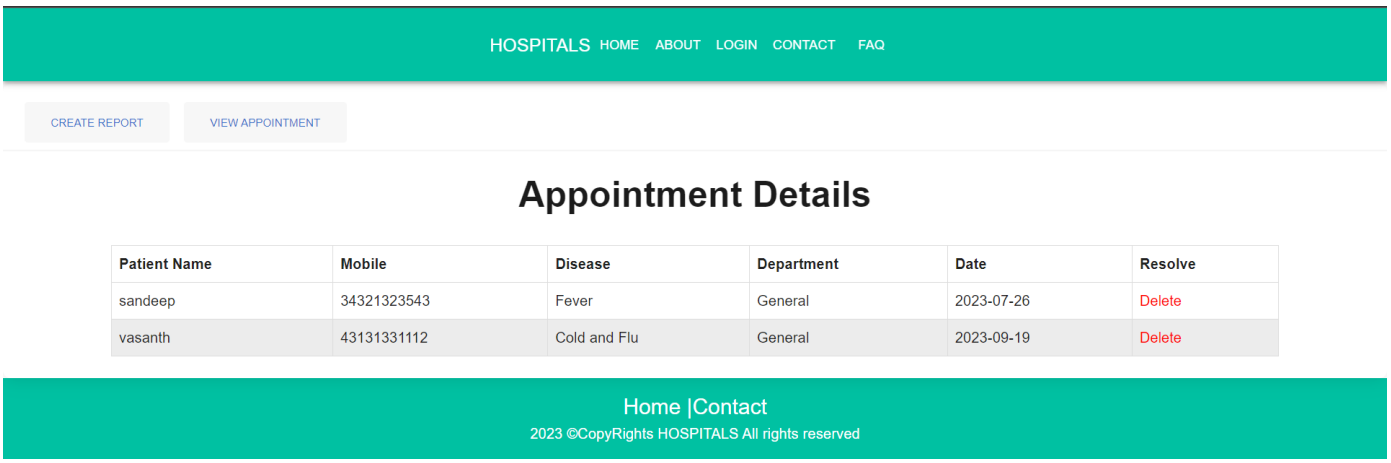
In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them. All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards. For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.
- Each unit has been separately tested by the development team itself and all the input has been validated.

## 6.4 TEST CASE

A test case, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released. Test cases are often referred to as test scripts, particularly when written - when they are usually collected into test suites.

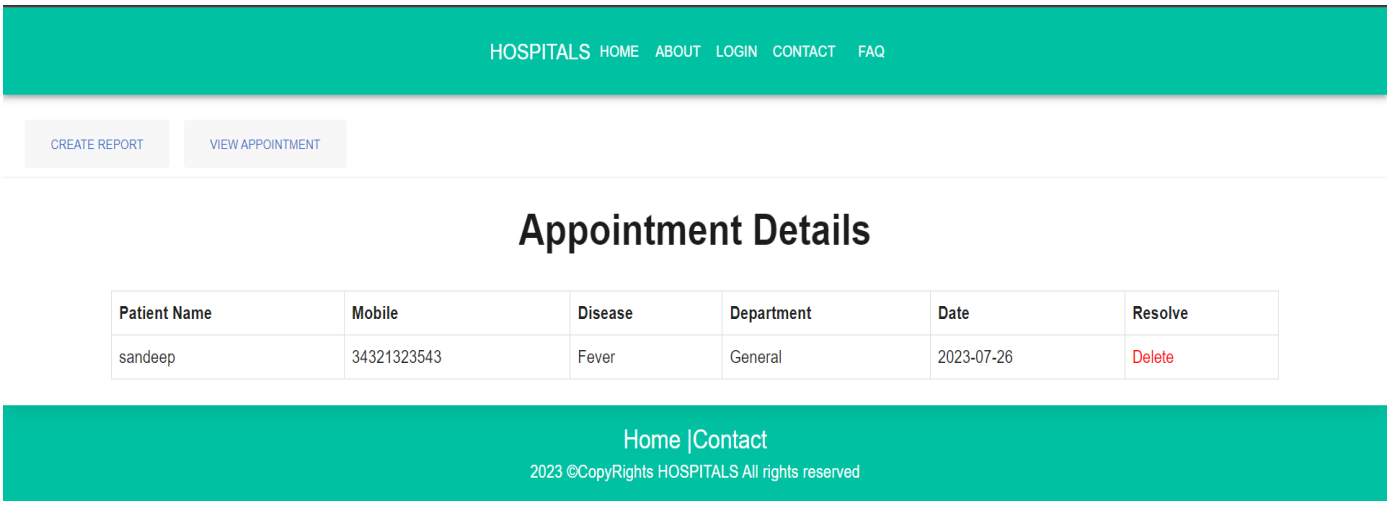
### 6.4.1 TEST CASE I




**EXPECTED:** A NEW APPOINTMENT SHOULD BE BOOKED AND ADDED

**ACTUAL :** A NEW APPOINTMENT IS BOOKED AND ADDED

### 6.4.2 TEST CASE II




**EXPECTED:** APPOINTMENT GETS DELETED IF THE PATIENT HAD VISITED THE DOCTOR

**ACTUAL:** APPOINTMENT IS SUCCESSFULLY DELETED

# CHAPTER 7

# CONCLUSION AND FUTURE WORK

## 7.1 CONCLUSION

In conclusion, the implementation of a Hospital Management System represents a pivotal advancement in healthcare administration, offering a wide array of advantages across medical facilities of varying sizes and specialties. This holistic solution transcends mere patient record management, profoundly impacting the efficiency, quality of care, and overall success of a healthcare institution.

By seamlessly integrating patient information, scheduling and other critical functions, the Hospital Management System not only streamlines administrative processes but also enhances patient care and safety. Real-time access to patient data, automation of routine tasks, and comprehensive analytics empower medical professionals to make informed decisions, reduce errors, and improve treatment outcomes.

In an era of escalating healthcare demands and increasing regulatory complexity, having a robust Hospital Management System is indispensable. It not only elevates the quality of patient care but also contributes to operational efficiency, cost-effectiveness, and revenue generation. It empowers healthcare organizations to focus on their core mission: delivering superior healthcare services.

Incorporating such a system into healthcare operations is more than a technological investment; it's an investment in the well-being of patients, the productivity of medical staff, and the sustainability of healthcare institutions. It positions healthcare providers to excel in a dynamic, data-centric healthcare landscape and continue to deliver excellence in patient care.

## 7.2 FUTURE WORK

We must collect all required data by giving GPS locations of a land and by taking access from Rain forecasting system of by the government, we can predict crops by just giving GPS location. Also, we can develop the model to avoid over and under crisis of the food.

## REFERENCES


[1] Sibbel, R., & Urban, C. (2001). Agent-based modeling and simulation for hospital management. In *Cooperative agents: applications in the social sciences* (pp. 183-202). Dordrecht: Springer Netherlands.

[2] Balaraman, Premkumar, and Kalpana Kosalram. "E-hospital management & hospital information systems-changing trends." *International Journal of Information Engineering and Electronic Business* 5.1 (2013): 50.

[3] Ghasemi, M., Nejad, M. G., & Bagzibagli, K. (2017). Knowledge management orientation: an innovative perspective to hospital management. *Iranian journal of public health*, *46*(12), 1639.

[4] Adebisi, O. A., et al. "Design and implementation of hospital management system." *International Journal of Engineering and Innovative Technology (IJEIT)* 5.1 (2015).

[5] WEB REFERENCES:

https://www.w3schools.com/react/default.asp

https://stackoverflow.com/

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

## APPENDIX A1

### FRONT END CODE

**HomePage.js**

```
import React from "react";
import '../Components/homepage.css'
import animation from '../images/animation.mp4'
import Button from '@mui/material/Button';
import img from '../servicespics/servicesallday.mp4';

function HomePage() {

    return (

        <>
        <div className="splitscreen">
        <div className="toppane">
        <h1 className="h1">"Welcome to Our Hospital
Website"</h1>

<h2 className="h2">At Hospital, we are committed to
providing exceptional healthcare services to our community.
  With a team of highly skilled medical professionals and
state-of-the-art facilities,
  we offer a wide range of medical specialties and
treatments to meet the diverse needs of our patients.</h2>
  <div className="getstarted">
  <Button variant="contained" color="success"
className="homebutton" >
        Get Started
      </Button>

  </div>
        </div>
        <div className="bottompane">
        <video className='videoTag' autoPlay loop muted>
           <source src={animation} type='video/mp4' />
      </video>
        </div>
      </div>
      <div className="flex-box">
        <div className="hr-line"></div></div>
      <div className="split">
      <div className="left">
```

```
        <video className='videoTag' autoPlay loop muted>
           <source src={img} type='video/mp4' />
      </video>
        </div>
        <div className="right">
        <h1 className="h1">"Find the Best Services"</h1>

<h2 className="h2">Our hospital offers a comprehensive range
of healthcare services to meet the needs of our patients.
Our highly skilled medical team provides specialized care in
various departments, including cardiology, neurology,
orthopedics, oncology, and obstetrics.
We offer state-of-the-art diagnostic facilities, advanced
surgical procedures, and personalized treatment plans. </h2>

  </div>
        </div>
        </>
     );
}

export default HomePage;
```

**STAFFLOGIN.js**

```
import { useState } from "react";
import {
  Flex,
  Box,
  FormControl,
  FormLabel,
  Input,
  Checkbox,
  Stack,
  Link,
  Button,
  Heading,
  Text,
  useColorModeValue,
  ChakraProvider,
} from '@chakra-ui/react';
import { FaUserAlt, FaLock } from "react-icons/fa";
// import { useNavigate } from "react-router-dom";
import {useSelector , useDispatch} from 'react-redux';
import { useNavigate } from "react-router-dom";
import axios from "axios";
import { selectUser } from "../../redux/features";

const  StaffLoginForm = () => {
```

```
  const [password, setPassword] = useState("");
  const [email, setEmail] = useState("");
  const [showPassword, setShowPassword] = useState("");

  const handleShowClick = () =>
setShowPassword(!showPassword);
  const navigate=useNavigate();
  const dispatch = useDispatch();
  const username = useSelector((state) => state.username);
  const handleUsernameChange = (e) => {
    dispatch({ type: "SET_USERNAME", payload: e.target.value
});
  };

  const handleEmailChange = (e) => {
    setEmail(e.target.value);
  }
  const handlePasswordChange = (e) => {
    setPassword(e.target.value);
  }
  const user = useSelector(selectUser);
  const submitHandler = (event) => {
    event.preventDefault();
    // const formData = new FormData();
    // formData.append("email",
event.target.elements.email.value);
    // formData.append("password",
event.target.elements.password.value);

    // const data = { email: formData.get("email"),
password: formData.get("password") };
    const data={
      email:email,
      password:password
    }
    console.log(data);
    axios.post("http://localhost:8080/auth/login", data)
      .then((response) => {
        console.log(response);
        const jwtToken = response.data.accessToken;
        localStorage.setItem("jwtToken", jwtToken)
      }).then(() => {if(user.role === "Nurse") {
        navigate("/staffmainhome/appointment");
      }
      else if(user.role === "Admin"){
          navigate("/adminmainhome/doctor");

        }
        else{
```

```
          navigate("/doctormainhome/createReport");
        }})
      .catch((error) => {
        console.log(error);
      });
      console.log(email)
  };


  return (
    <>
    <ChakraProvider>
     <Flex
      minH={'100vh'}
      align={'center'}
      justify={'center'}
      bg={useColorModeValue('gray.50', 'gray.800')}>
      <Stack spacing={8} mx={'auto'} maxW={'lg'} py={12}
px={6}>
        <Stack align={'center'}>
          <Heading fontSize={'4xl'}>Login into Staff
account</Heading>
          <Text fontSize={'lg'} color={'gray.600'}>
            Your details will not be revealed
          </Text>
        </Stack>
        <Box
          rounded={'lg'}
          bg={useColorModeValue('white', 'gray.700')}
          boxShadow={'lg'}
          p={8}>
          <Stack spacing={4}>
            <FormControl id="email">
              <FormLabel>Email address</FormLabel>
              <Input type="email"  name="email"
onChange={handleEmailChange}/>
            </FormControl>
            <FormControl id="password">
              <FormLabel>Password</FormLabel>
              <Input type="password"  name="password"
onChange={handlePasswordChange}/>
            </FormControl>
            <Stack spacing={10}>
              <Stack
                direction={{ base: 'column', sm: 'row' }}
                align={'start'}
                justify={'space-between'}>
                <Checkbox>Remember me</Checkbox>
                <Link color={'blue.400'}
```

```
href="/staffsignup">Sign up</Link>
              </Stack>
              <Button
                bg={'#00C1A2'}
                color={'white'}
                _hover={{
                  bg: '#00C1A2',
                }}
                //
onClick={()=>navigate('/staffmainHome/appointment')}
                onClick={submitHandler}
                >
                Sign in
              </Button>
            </Stack>
          </Stack>
        </Box>
      </Stack>
    </Flex>
    </ChakraProvider>
    </>
  );
};

export default StaffLoginForm;
```

**BACK END CODE**

```
package com.neo.security.controller;

import javax.validation.Valid;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.authentication.AuthenticationMa
nager;
import
org.springframework.security.authentication.BadCredentialsEx
ception;
import
org.springframework.security.authentication.UsernamePassword
AuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
```

```
import
org.springframework.web.bind.annotation.RestController;

    import com.neo.security.entity.StaffEntity;
    import com.neo.security.request.AuthRequest;
    import com.neo.security.response.AuthResponse;
    import com.neo.security.service.StaffServiceImpl;
    import com.neo.security.util.JwtUtil;

    import lombok.RequiredArgsConstructor;

    @RequiredArgsConstructor
    @CrossOrigin(origins = "http://localhost:3000")
    @RestController
    @RequestMapping("/auth")
    public class AuthController {

        private final AuthenticationManager
    authenticationManager;
        private final JwtUtil jwtUtil;

        @Autowired
        private StaffServiceImpl service;

        @PostMapping("/login")
        public ResponseEntity<?> login(@RequestBody @Valid
    AuthRequest request){
```

```
                try {
                Authentication authentication =
authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword()));
                StaffEntity user = (StaffEntity)
authentication.getPrincipal();

//              String accessToken = "JWT token !!!!";
                String accessToken =
jwtUtil.generateToken(user);
                AuthResponse response = new
AuthResponse(user.getEmail(), accessToken);

                return
ResponseEntity.status(HttpStatus.OK).body(response);
          }catch (BadCredentialsException e) {
                return
ResponseEntity.badRequest().body("Login failed");
          }
     }
     @PostMapping("/register")
     public Boolean addUser(@RequestBody StaffEntity user) {
          return service.addUser(user);
     }
}
```

# APPENDIX A2

## 7.1 HOME PAGE

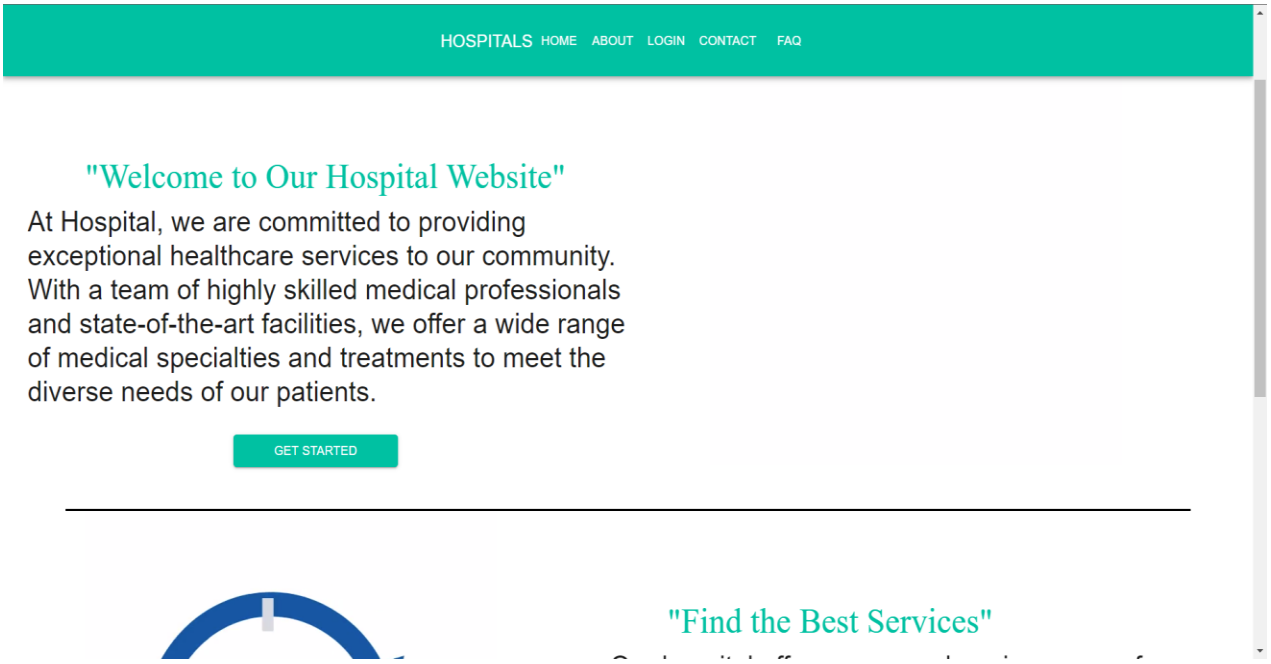



**Fig 7.1 :** HOME PAGE

## 7.2 BOOK APPOINTMENT

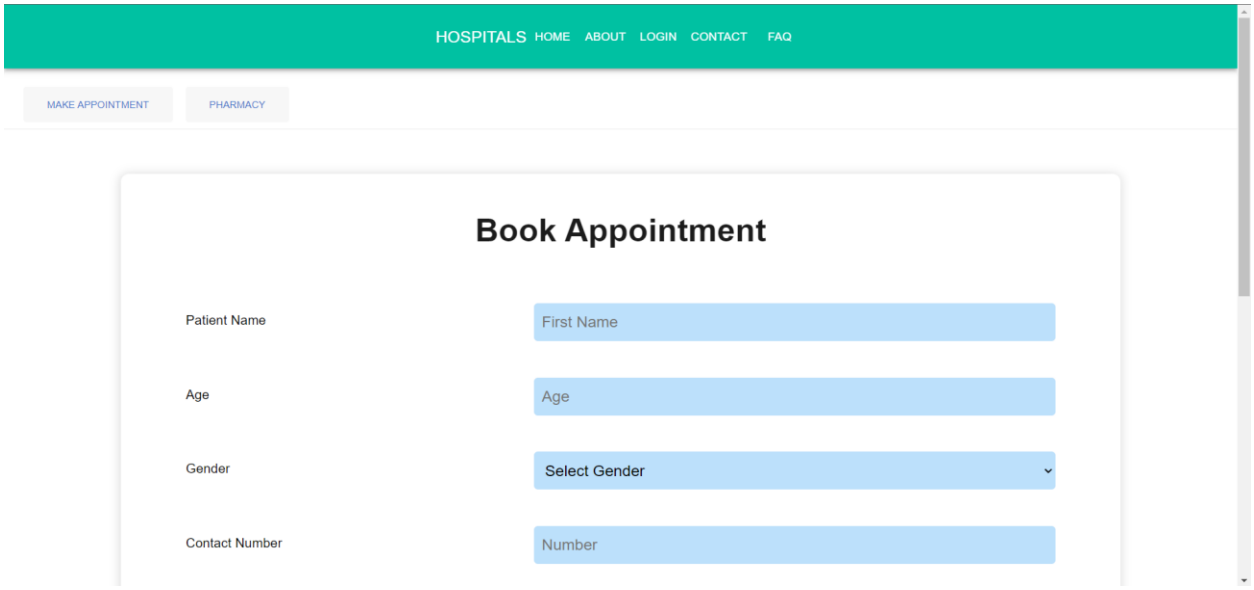



**Fig 7.2(a) : BOOK APPOINTMENT**

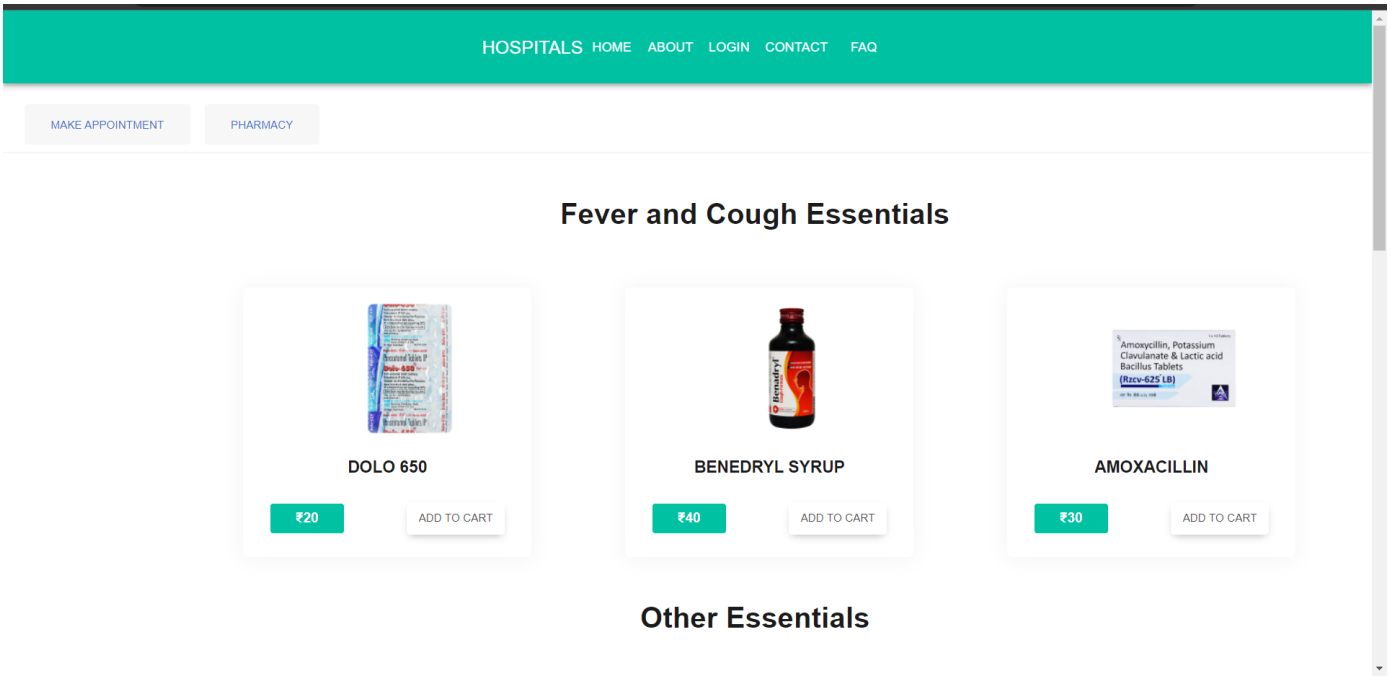



**Fig 7.2(b) :** PHARMACY PAGE

## 7.3 REPORT PAGE

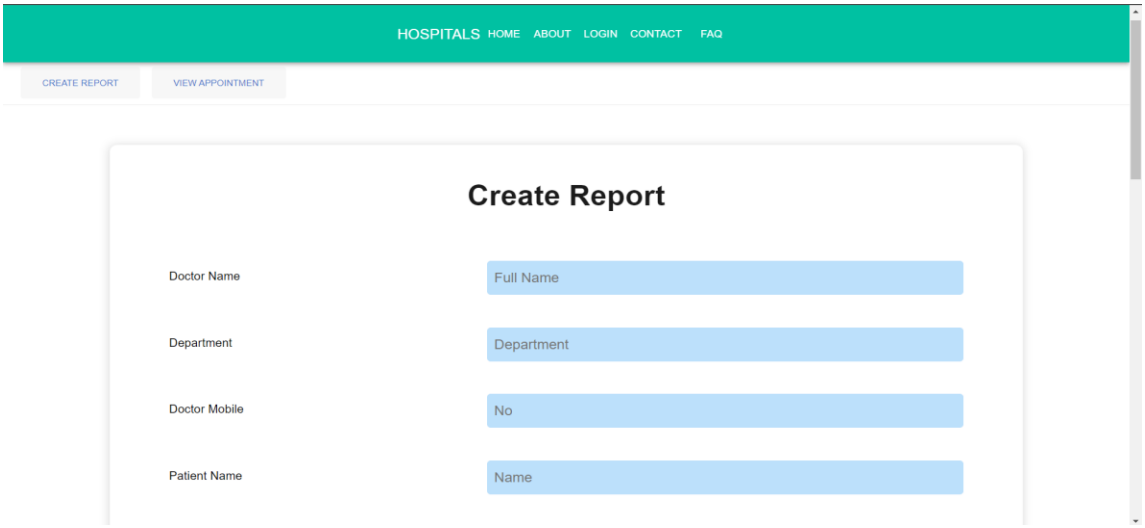



**Fig 7.3 :** REPORT PAGE

## 7.4 ADMIN PAGE

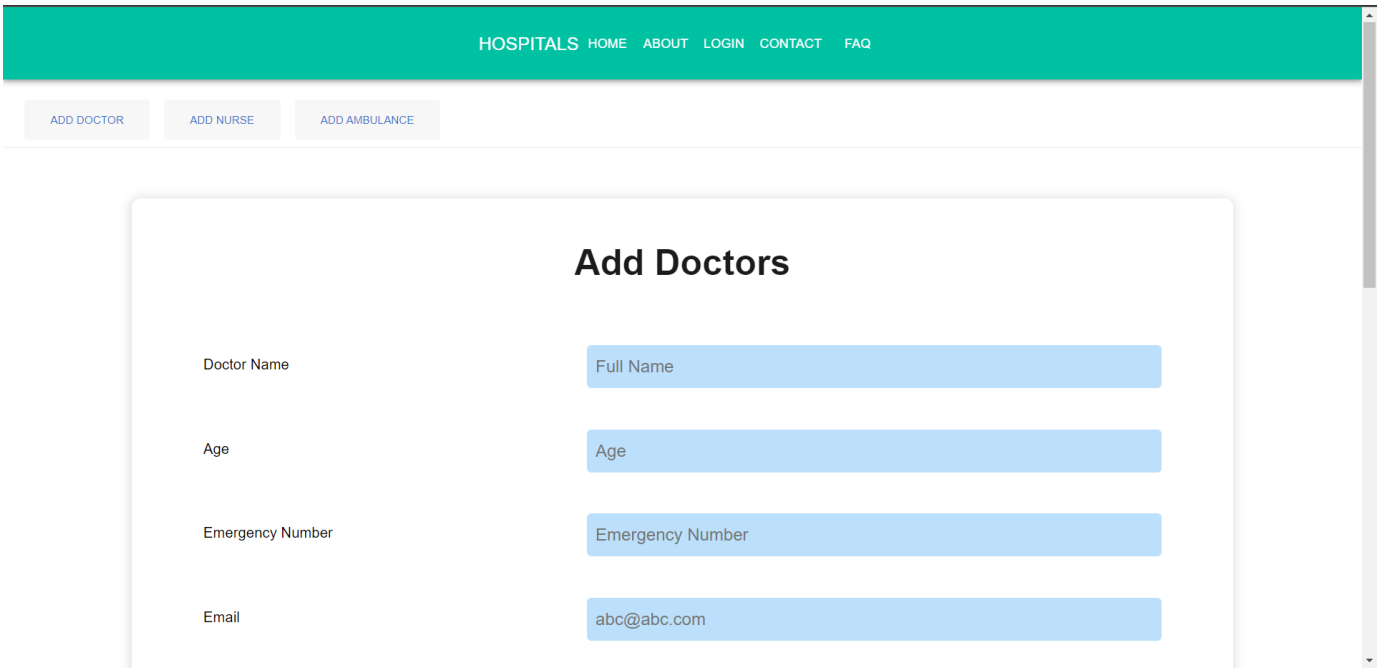



**Fig 7.4 :** ADMIN PAGE