

Medsphere

**A Thesis Submitted
In Partial Fulfillment of the Requirements for the Degree of**

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING
by

Rishav Raj (1901270100042)
Shivam Yadav (190127010051)
Sachin Chaubey (190127010046)

Under the Supervision of

Mr. Mukul Maurya
Asst. Professor, CSE Dept. IIMT ENG. Meerut
(Supervisor)

IIMT ENGINEERING COLLAGE



to the
Faculty of Engineering

**Dr. A.P.J. ABDUL KALAM TECHNICAL
UNIVERSITY, UTTAR PRADESH, LUCKNOW**
(Formerly Uttar Pradesh Technical University, Lucknow) JUNE, 2023

ABSTRACT

The Medsphere web application is a comprehensive solution designed to manage daily medical affairs in a hospital setting. It provides functionalities such as patient management, inventory management, prescription generation, and more. The aim of this project is to develop a lightweight and user-friendly sys with a clean user interface that prioritize essential services. Additionally, an autocompletion framework is integrated to facilitate quick and accurate retrieval of medical information by typing a few keystrokes. The project follows a collaborative development approach utilizing popular tools such as GitHub, Notion, and Figma. The frontend development is implemented using React JS, Context API, CSS, jsPDF, and Loadash for utility functions. The backend development is based on Express JS and utilizes PostgreSQL as the database driver. Custom middleware handlers are implemented to ensure efficient error handling.

The system architecture comprises various modules, including patient management, authentication management, prescription management, appointment management, department management, inventory management, and doctor management. Each module is meticulously designed to address specific requirements and optimize workflow processes within the hospital environment.

State-of-the-art technologies are employed throughout the project, ensuring a modern and scalable solution. Integration with other applications and services is a crucial aspect, with a focus on creating a medical ecosystem. This future scope includes potential integration with AI or machine learning algorithms for advanced analytics, automated decision-making, and predictive capabilities. The project also highlights the use of Jest for testing the React components and includes extensive unit testing for the backend using the Jest framework. Robust testing methodologies ensure the reliability, stability, and performance of the Medsphere application. The achievements of this project lie in the development of a lightweight system with a clean user interface, efficient auto-completion framework, and seamless integration with other applications. These aspects contribute to enhanced user experience, improved efficiency, and cost-effectiveness compared to existing systems in the market.

DECELERATION STATEMENT

I hereby declare that the research work reported in the dissertation entitled **“MEDSPHERE A WEB APP TO MANAGE DAY TO DAY MEDICAL AFFAIRS OF A HOSPITAL”** in partial fulfillment of the requirement for the award of **Degree for Bachelors of Technology in Computer Science and Engineering** at **A.K.T.U, UTTAR PRADESH, LUCKNOW**, is an authentic work carried out under supervision of my research supervisor **Mrs. Mukul Maurya**. I have not submitted this work elsewhere for any degree or diploma. I understand that the work presented here with is in direct compliance with A.K.T.U. University's Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

DATE

Signature of Student

Name: Rishav Raj

Roll no: 1901270100042

Signature of Student

Name: Shivam Yadav

Roll no: 1901270100051

Signature of Student

Name: Sachin Chaubey

Roll no: 1901270100046

SUPERVISOR'S CERTIFICATE

Certified that **Mr. Rishav Raj** (Roll No.1901270100042), **Mr Shivam Yadav** (Roll No. 1901270100046) **Mr. Sachin Chaubey** (Roll No. 190170100046) has carried out the research work presented in this thesis entitled “**MEDSPHERE WEB APP TO MANAGE MEDICAL AFFAIRS**” for the award of **Master of Technology (Computer Science and Engineering)** from Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh, Lucknow under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Mr. Mukul Maurya
(Supervisor)

Asst. Professor, CSE Dept.
IIMT ENG. Meerut (U.P)



**DR. A.P.J ABDUL KALAM TECHNICAL UNIVERSITY, UTTAR
PRADESH, LUCKNOW**

(Formerly Uttar Pradesh Technical University, Lucknow)

CERTIFICATE OF THESIS SUBMISSION FOR EVALUATION

(To be submitted in duplicate)

1. Name :
2. Enrollment No. :
3. Thesis title:
.....
.....
4. Degree for which the thesis is submitted:.....
5. Faculty of the University to which the thesis is submitted
.....
6. Thesis Preparation Guide was referred to for preparing the thesis. ☐ YES ☐ NO
7. Specifications regarding thesis format have been closely followed. ☐ YES ☐ NO
8. The contents of the thesis have been organized based on the guidelines. YES NO
9. The thesis has been prepared without resorting to plagiarism. YES NO
10. All sources used have been cited appropriately. YES NO

11. The thesis has not been submitted elsewhere for a degree.

☐ YES ☐ NO

12. All the correction have been incorporated.

☐ YES ☐ NO

(Signature(s) of the Supervisor(s))

(Signature of Candidate)

1) Name:.....

1) Name:.....

2) Name:.....

2) Name:.....

3) Name:.....

ACKNOWLEDEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. I would like to convey my most heartfelt and sincere gratitude to my mentor Mrs. Mukul Maurya for his valuable guidance, advice, understanding and supervision throughout the development of this dissertation study. His willingness to motivate me contributed tremendously to achieve the goal successfully. I would like to thank to the Project Approval Committee members for their valuable comments and discussions. I owe my thanks to my family and friends for their consistent moral support. They are ones who have encouraged me at every step of my life.

Name: Rishav Raj

Roll No: 1901270100042

Name: Shivam yadav

Roll No: 1901270100051

Name: Sachin Chaubey

Roll No: 1901270100046

Table of Content

Cover Page Title.....	1
ABSTRACT.....	2
Declearation Statement.....	3
SUPERVISOR'S CERTIFICATE.....	4
Certificate of final thesis submission.....	5
Acknowledgement.....	7
 Table of Content.....	 8
Table of Figures.....	10
Chapter 1.....	14
Introduction.....	14
1.1 Motivation.....	14
1.2 Problem we solve.....	14
1.3 Scope of the project.....	15
1.4 Methodology and tools.....	18
Chapter 2.....	21
Literature Review.....	21
2.1 Introduction to Hospital Managment Systems.....	21
2.2 Existing systems in market.....	22
2.3 How medsphere different from existing solutions.....	23
2.4 Following the Unix Philosophy.....	24
2.5 Integration.....	26
Chapter 3.....	29
System Design.....	29
3.1 Artitecture and Components.....	29
3.2 Completion Engine.....	33
3.3 UI/UX Design.....	34
3.3.1 Sign in and Sign up Page.....	34
3.3.2 Manage Page.....	35
3.3.3 Appointments Page.....	35
3.3.4 Prescription Page.....	36
Chatper 4.....	37
Implementation.....	37
4.1 Frontend Development.....	37
4.1.1 Routes Component.....	39
4.1.2 Static Component.....	40
4.1.3 Manage Component.....	41
4.1.4 Doctors Entity.....	42
4.1.5 Patients Entity.....	43

4.1.6 Appointment Component.....	44
4.1.7 Completion Component.....	45
4.1.8 Prescription Componet.....	47
4.2 Backend Development.....	50
4.2.1 Api Routes.....	52
4.2.2 Compnent routes.....	53
4.2.3 Auth Route.....	54
4.2.3 Users Route.....	57
4.3 Database Schemas.....	59
4.3.1 Users Schema.....	59
4.3.2 Patients Schema.....	60
4.3.3 Department Schema.....	60
4.3.4 Medicine Schema.....	60
4.3.5 Appointments Schema.....	61
4.3.6 Prescription Schema.....	61
4.3.7 Prescribed Medicine schema.....	61
4.3.8 Prescribed Advice schema.....	61
4.3.9 Constrains between Schemas.....	62
Chapter 5.....	64
System Testing.....	64
5.1 Testing Methodology.....	64
5.2 Frontend testing.....	66
5.3 Backend testing.....	68
5.4 Api testing.....	72
Chapter 6.....	74
Conclusion.....	74
6.1 Summary of Achievements.....	74
6.2 Limitations.....	77
6.3 Future Scope.....	78
6.4 Final remark.....	82

Table of Figures

Figure 3.1	7
Figure 3.2	7
Figure 3.3	7
Figure 3.4	7
Figure 3.5	7
Figure 3.6	7
Figure 3.7	7
Figure 4.1	7
Figure 4.2	7
Figure 4.3	7
Figure 4.4	7
Figure 4.5	7
Figure 4.6	7
Figure 4.7	7
Figure 4.8	7
Figure 4.9	7
Figure 4.10	7
Figure 4.11	7
Figure 4.12.....	7
Figure 4.13	7
Figure 4.14	7
Figure 4.15	7
Figure 4.16	7
Figure 4.17.....	7
Figure 4.18	7
Figure 4.19	7
Figure 4.20.....	7
Figure 4.21	7
Figure 4.22	7
Figure 4.23.....	7
Figure 4.24.....	7
Figure 4.25.....	7
Figure 4.26.....	7
Figure 4.27.....	7
Figure 4.28.....	7

Figure 4.29.....	7
------------------	---

Chapter 1

Introduction

1.1 Motivation

In recent years digitalization spread like wildfire. Many prominent businesses move online or in digital format. If it is for good or not, is it fruitful or not only the solution of the action says. Does moving any business to digital solve the core problem or not? One day our professor Mr. Rajeev Sharma discussing our project ideas and he suggests a solution to the problem “He says there is so much hassle in generating medical prescriptions for doctors can’t we make this process easy” that’s when we decided to work on this idea and turn up into a software.

1.2 Problem we solve

We first need to draw outlines of our idea: what we touch, what we do not, and what our future scope is.

The objective of the Medsphere web app is to streamline medical operations in a hospital setting, by providing a single platform for managing day-to-day medical affairs. The web app serves as an all-in-one solution for prescription generation, inventory management, patient management, and department and doctor management.

The web app aims to improve the efficiency of hospital staff, reduce errors in prescription generation, and ultimately improve patient care. By providing accurate and up-to-date information about patients, inventory, and staff, the Medsphere web app helps healthcare providers make informed decisions and provide timely care to patients.

Overall, the objective of the Medsphere web app is to simplify medical operations for hospitals and healthcare providers and improve the quality of care provided to patients.

- Streamline medical operations in a hospital setting
- Provide a single platform for managing day-to-day medical affairs

- Serve as an all-in-one solution for prescription generation, inventory management, patient management, and department and doctor management
- Improve the efficiency of hospital staff
- Reduce errors in the prescription generation
- Improve patient care by providing accurate and up-to-date information about patients, inventory, and staff
- Help healthcare providers make informed decisions and provide timely care to patients
- Simplify medical operations for hospitals and healthcare providers
- Improve the quality of care provided to patients

1.3 Scope of the project

1. Introduction

The Medsphere Hospital Management Web App is a comprehensive system designed to streamline and automate various medical affairs within a hospital. It encompasses a wide range of functionalities, including patient management, inventory management, prescription generation, and more. This project scope outlines the key features and modules of the Medsphere web app, providing a detailed overview of its capabilities and objectives.

2. Patient Management Module

The patient management module is at the core of the Medsphere web app. It allows hospitals to efficiently manage patient records, including personal details, medical history, appointments, and billing information. The module facilitates tasks such as patient registration, appointment scheduling, and tracking of treatment progress. It also enables communication between healthcare providers and patients through secure messaging.

3. Inventory Management Module

The inventory management module assists hospitals in effectively managing their medical supplies and equipment. It enables tracking and control of stock levels, automates inventory replenishment processes, and generates reports on stock usage, expiration dates, and costs. The module integrates with the purchasing system, ensuring seamless procurement of supplies and efficient supply chain management.

4. Prescription Generation Module

The prescription generation module automates the process of creating accurate and legible prescriptions for patients. It allows healthcare providers to enter medication

details, dosage instructions, and frequency of administration. The module incorporates drug interaction checking, allergy alerts, and dosage calculations to enhance patient safety. It also maintains a prescription history for reference and audit purposes.

5. Reporting and Analytics Module

The reporting and analytics module offers comprehensive data analysis and reporting capabilities. It provides insights into various aspects of hospital operations, including patient demographics, treatment outcomes, inventory utilization, financial performance, and more. The module generates customizable reports and visualizations to assist in decision-making, resource allocation, and performance evaluation.

6. Integration and Scalability

The Medsphere web app is designed to integrate seamlessly with existing hospital systems and third-party applications. It supports interoperability standards, allowing for data exchange with electronic health record (EHR) systems, laboratory information systems (LIS), and other healthcare platforms. The web app is scalable and can accommodate the growth and evolving needs of the hospital, ensuring long-term viability and flexibility.

7. Security and Compliance

The Medsphere web app prioritizes data security and compliance with relevant regulations, such as HIPAA (Health Insurance Portability and Accountability Act). It employs robust authentication mechanisms, encryption protocols, and access controls to safeguard patient information and maintain confidentiality. Regular security audits and updates are conducted to address potential vulnerabilities and ensure compliance with industry standards.

8. User Support and Training

The project includes comprehensive user support and training to ensure a smooth transition and successful adoption of the Medsphere web app. Training programs are designed for healthcare providers, administrators, and support staff, covering all relevant modules and functionalities. Ongoing technical support is provided to address user queries, troubleshoot issues, and optimize system performance.

9. Future Enhancements

The Medsphere web app is built with a vision for future enhancements and expansions. As technology advances and healthcare requirements evolve, the project scope allows for incorporating additional features and modules. This includes

telemedicine integration, artificial intelligence-based decision support systems, remote patient monitoring, and more, enabling hospitals to stay at the forefront of medical innovation.

10. Auto Completion Generation Module

The Auto Completion Generation Module is a highly efficient feature designed to facilitate the process of data entry and retrieval within the Medsphere Hospital Management Web App. It provides a user-friendly interface that allows healthcare providers to quickly access relevant information by typing a few keystrokes and retrieving perfect matches from the backend database.

- a. **Medicine Auto Completion:** When entering medication details, healthcare providers can simply input a few letters of the medicine name, and the Auto Completion Generation Module will instantly fetch a list of perfect matches from the backend database. This functionality saves time and improves accuracy in medication management.
- b. **Patient Auto Completion:** The module also supports auto completion for patient information. By entering a few letters of a patient's name, identification number, or other relevant details, the system quickly retrieves matching records from the patient database. This enables faster access to patient information during registration, appointment scheduling, and other processes.
- c. **Diagnosis Auto Completion:** Healthcare providers can benefit from auto completion when entering diagnoses for patients. As they begin typing the symptoms or medical condition, the module suggests relevant diagnoses based on the database of known medical conditions. This feature assists in efficient and accurate diagnosis recording.
- d. **Procedure Auto Completion:** When documenting medical procedures or treatments, the Auto Completion Generation Module suggests relevant procedure codes or descriptions based on the entered keywords. This functionality streamlines the process of procedure documentation and ensures consistency in coding.
- e. **Inventory Auto Completion:** In the inventory management module, the module assists hospital staff in quickly finding and selecting the correct inventory items by providing auto completion suggestions. This saves time when

entering inventory data, ensuring accurate tracking and efficient inventory management.

The Auto Completion Generation Module significantly enhances data entry speed and accuracy within the Medsphere Hospital Management Web App. By leveraging the backend database, it enables healthcare providers to find and retrieve the required information with just a few keystrokes, thereby improving workflow efficiency and user experience.

1.4 Methodology and tools

The methodology for developing the Medsphere Hospital Management Web App follow a structured and iterative approach

1. **Requirement Analysis:**
Conduct a thorough analysis of the hospital's medical affairs management requirements. This involves gathering input from key stakeholders, such as healthcare providers, administrators, and IT staff, to identify their specific needs and challenges. Document and prioritize the requirements to serve as a foundation for the development process.
2. **System Design:**
Based on the gathered requirements, create a detailed system design for the web app. Define the architecture, modules, and functionalities of the application. Design the database schema, user interfaces, and integration points with external systems. Ensure scalability, security, and usability considerations are incorporated into the design.
3. **Agile Development:**
Adopt an agile development approach, such as Scrum or Kanban, to facilitate iterative development and continuous improvement. Break down the project into manageable sprints or iterations, with each iteration focusing on developing specific modules or functionalities. Develop the web app in an incremental manner, regularly reviewing and incorporating feedback from stakeholders.
4. **Frontend and Backend Development:**
Simultaneously develop the frontend and backend components of the web app. The frontend development involves creating intuitive and user-friendly interfaces using modern web technologies, ensuring responsive design and cross-browser compatibility. The backend development includes implementing the server-side logic, database integration, and APIs to enable data retrieval and manipulation.

a. Frontend Development:

The frontend development team is responsible for designing and implementing the user interface (UI) components of the web app. They collaborate closely with UX/UI designers to ensure a visually appealing and intuitive interface that aligns with industry best practices. The team utilizes responsive design principles to ensure optimal user experience across various devices and screen sizes. Additionally, they ensure cross-browser compatibility to provide a consistent experience for users accessing the web app from different browsers. The frontend developers leverage modern frontend frameworks and libraries, such as React.js to facilitate efficient development, component reusability, and code maintainability. They work closely with the UX/UI designers to translate design mockups into functional UI components, incorporating interactive elements, data visualization, and smooth navigation. Frontend development also involves implementing client-side validation and handling user interactions to enhance usability and responsiveness.

b. Backend Development:

The backend development team focuses on implementing the server-side logic, database integration, and APIs that power the functionality of the Medsphere Hospital Management Web App. They design and develop robust and scalable backend systems using Node.js. The team ensures the proper handling and storage of data by integrating with PostgreSQL. Backend developers create APIs and endpoints that enable data retrieval, manipulation, and communication between the frontend and backend components. They implement business logic, authentication mechanisms, and security measures to safeguard sensitive data and ensure compliance with relevant regulations. Effective collaboration between the frontend and backend development teams is crucial to ensure seamless integration and synchronization of the web app's components. Regular communication and coordination between the teams help resolve any dependencies, address technical challenges, and maintain consistency in the development process. By concurrently developing the frontend and backend components, the Medsphere Hospital Management Web App can be delivered in a timely manner, ensuring a well-coordinated and feature-rich application that meets the needs of healthcare providers and enhances the overall management of daily medical affairs within the hospital.

Using GitHub, Notion, and Figma is an excellent choice for collaboration and project management during the development of the Medsphere Hospital Management Web App.

Throughout the development process, ensure effective communication and collaboration between the development team

1. GitHub for Version Control and Collaboration:

Set up a GitHub repository to manage version control for the project. Create branches for different features or modules to enable parallel development. Collaborate with your team members by pushing code changes, reviewing each other's code through pull requests, and resolving any conflicts that may arise. Utilize GitHub's issue tracking system to manage tasks, assign responsibilities, and track progress. Regularly communicate and provide updates within GitHub's collaboration features, such as comments, discussions, and notifications.

2. Notion for Project Documentation and Task Management:

Utilize Notion as a centralized platform for project documentation and task management. Create a dedicated workspace for the Medsphere Hospital Management Web App project, organizing pages or sections for different aspects, such as requirements, design, development, testing, and deployment. Document important information, decisions, and meeting notes within Notion to ensure easy access and knowledge sharing among team members. Use Notion's task management features to assign and track tasks, set deadlines, and monitor progress.

3. Figma for Design and Prototyping:

Utilize Figma as the primary tool for designing and prototyping the web app's user interfaces. Collaborate with your team members by sharing design files and enabling real-time collaboration. Use Figma's design components and libraries to maintain design consistency and facilitate efficient updates across multiple screens or modules. Leverage Figma's interactive prototyping features to create interactive mockups or wireframes for user testing and validation.

4. Integrating the Tools:

Ensure seamless integration between GitHub, Notion, and Figma by establishing a clear workflow. For example:

- a. Use GitHub links within Notion pages to reference code repositories or specific pull requests related to a task or requirement.
- b. Embed Figma designs or prototypes within Notion pages for easy access and reference.
- c. Update Notion pages with GitHub commit messages or pull request summaries to provide visibility into development progress.

Chapter 2

Literature Review

2.1 Introduction to Hospital Management Systems

Hospital Management Systems (HMS) are comprehensive software solutions designed to streamline and automate various administrative, operational, and clinical processes within healthcare organizations. These systems aim to improve efficiency, enhance patient care, and facilitate effective management of medical affairs in hospitals and other healthcare facilities.

Hospital Management Systems encompass a wide range of functionalities and modules that integrate and centralize critical information and operations. They typically include components such as patient management, appointment scheduling, billing and invoicing, inventory management, prescription generation, laboratory and diagnostic test management, electronic health records (EHR) management, and reporting.

The primary objective of a Hospital Management System is to facilitate the seamless flow of information and streamline workflows across different departments and stakeholders within a healthcare organization. By digitizing and automating processes, HMS eliminates manual paperwork, reduces errors, enhances data accuracy, and improves overall operational efficiency.

Key Benefits of Hospital Management Systems:

1. **Improved Patient Care:** HMS enables healthcare providers to access patient information instantly, including medical history, diagnoses, medications, and treatment plans. This quick and easy access to critical patient data enhances decision-making, reduces medical errors, and improves the quality of care provided to patients.
2. **Efficient Administrative Processes:** HMS automates administrative tasks, such as appointment scheduling, billing, and invoicing, leading to time and cost savings. It facilitates streamlined communication between different departments, optimizing resource allocation and ensuring efficient utilization of hospital facilities.

3. **Enhanced Inventory Management:** The inventory management module of an HMS enables healthcare facilities to effectively manage their medical supplies, pharmaceuticals, and equipment. It ensures accurate stock tracking, timely reordering, and minimizes wastage, ultimately reducing costs and ensuring the availability of essential items when needed.
4. **Data Centralization and Accessibility:** HMS serves as a centralized repository of patient data, medical records, and operational information. This centralized access promotes data integrity, facilitates collaboration among healthcare providers, and ensures continuity of care across different departments and care settings.
5. **Data Security and Compliance:** Hospital Management Systems implement robust security measures to protect sensitive patient information, ensuring compliance with healthcare data protection regulations. This includes measures such as user authentication, data encryption, and audit trails to monitor system access and usage.
6. **Streamlined Reporting and Analytics:** HMS generates comprehensive reports and analytics, providing valuable insights into various aspects of hospital operations, such as patient outcomes, resource utilization, financial performance, and compliance. These reports assist administrators and management in making informed decisions and optimizing processes.

Hospital Management Systems play a crucial role in modern healthcare organizations by automating processes, improving patient care, and optimizing operational efficiency. The Medsphere Hospital Management Web App aims to leverage these benefits and provide an intuitive, comprehensive, and efficient solution for managing daily medical affairs in hospitals, benefiting healthcare providers, administrators, and ultimately, patients.

2.2 Existing systems in market

There are few existing systems in markets

1. **Epic Systems Corporation:**
Epic is a widely used Hospital Management System known for its comprehensive features and interoperability. It offers modules for patient management, scheduling, billing, electronic health records (EHR), and clinical decision support. Epic is known for its scalability and is commonly used in large healthcare organizations.
2. **Cerner Corporation:**

Cerner is another prominent vendor that provides Hospital Management Systems. Their solution encompasses a wide range of functionalities, including patient registration, appointment scheduling, EHR management, billing, and reporting. Cerner focuses on interoperability and integration with other healthcare systems.

3. Allscripts Healthcare Solutions, Inc.:

Allscripts offers Hospital Management Systems that cater to both small clinics and large healthcare organizations. Their system includes modules for patient management, EHR, revenue cycle management, and population health management. Allscripts emphasizes user-friendly interfaces and customizable workflows.

4. MEDITECH:

MEDITECH provides a suite of Hospital Management Systems for different healthcare settings, including hospitals, clinics, and long-term care facilities. Their systems cover various areas, such as patient management, scheduling, EHR, billing, and pharmacy management. MEDITECH offers different modules to cater to the specific needs of different healthcare providers.

5. NextGen Healthcare:

NextGen Healthcare offers Hospital Management Systems that focus on enhancing patient engagement and population health management. Their system includes features for patient registration, appointment scheduling, EHR, billing, and reporting. NextGen Healthcare emphasizes interoperability and seamless data exchange.

6. eClinicalWorks:

eClinicalWorks provides a Hospital Management System that integrates electronic health records, practice management, and revenue cycle management. Their system is known for its user-friendly interface and comprehensive functionality, covering various aspects of healthcare operations.

7. Athenahealth:

Athenahealth offers a cloud-based Hospital Management System that includes modules for patient management, appointment scheduling, EHR, billing, and reporting. Their system focuses on ease of use, scalability, and efficient workflows.

2.3 How medsphere different from existing solutions

Medsphere sets itself apart from existing Hospital Management Systems in several distinct ways. Firstly, Medsphere prides itself on being a lightweight solution that prioritizes essential services, ensuring that healthcare organizations can efficiently manage their daily medical affairs without unnecessary complexities. By focusing on core functionalities, Medsphere streamlines workflows and minimizes the learning curve for users, allowing for faster adoption and increased productivity.

One of Medsphere's standout features is its clean and intuitive user interface (UI). The UI is thoughtfully designed to enhance usability and provide a seamless user experience. The emphasis on simplicity and clarity ensures that healthcare providers can navigate the system effortlessly, reducing the risk of errors and improving overall efficiency. The clean UI contributes to a more intuitive workflow, empowering users to perform their tasks efficiently and effectively.

Another key differentiator of Medsphere is its advanced auto-completion framework. The auto-completion feature significantly improves efficiency when generating prescriptions, searching for medications, or inputting patient data. By intelligently predicting and suggesting options based on a few keystrokes, Medsphere saves time and reduces errors, enabling healthcare providers to focus more on patient care rather than struggling with manual data entry.

Moreover, Medsphere offers a cost-effective alternative to existing systems in the market. With its competitive pricing structure, Medsphere aims to make advanced Hospital Management System functionalities accessible to healthcare organizations of all sizes, including small clinics and hospitals with limited budgets. By providing a cost-effective solution without compromising on essential features, Medsphere ensures that healthcare providers can leverage modern technology to enhance patient care and operational efficiency without financial burdens.

In summary, Medsphere stands out among existing Hospital Management Systems due to its lightweight nature, focus on essential services, clean UI, advanced auto-completion framework, and affordability. By providing a streamlined, user-friendly, and cost-effective solution, Medsphere empowers healthcare organizations to efficiently manage their daily medical affairs while optimizing patient care, reducing costs, and improving overall operational efficiency.

2.4 Following the Unix Philosophy

The Unix philosophy is documented by Doug McIlroy in the Bell System Technical Journal from 1978: Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features". Expect the output of every program to become the input to another, as yet unknown, program.

Medsphere's unique approach to being lean and focused on essential services aligns with the Unix philosophy, which emphasizes modularity, simplicity, and the ability to integrate with other applications seamlessly. By adhering to this philosophy, Medsphere lays the foundation for creating a robust and interconnected medical ecosystem in the future.

The Unix philosophy advocates building software that performs a specific task well while promoting interoperability with other applications. Similarly, Medsphere's lightweight design and focus on core functionalities ensure that it excels at managing daily medical affairs efficiently. By avoiding unnecessary complexities and feature bloat, Medsphere provides a streamlined and intuitive user experience, enabling healthcare providers to navigate the system effortlessly and carry out their tasks effectively.

In embracing the Unix philosophy, Medsphere recognizes the importance of integration and collaboration within the healthcare industry. By creating a modular system with well-defined interfaces, Medsphere positions itself to seamlessly integrate with other medical applications and services. This approach allows for the development of a comprehensive medical ecosystem where different applications can communicate, exchange data, and work together to provide holistic healthcare solutions.

The future vision of Medsphere revolves around fostering interoperability and promoting collaboration among healthcare providers, medical devices, and other healthcare applications. By adhering to the Unix philosophy, Medsphere sets the stage for connecting with electronic health record (EHR) systems, telemedicine platforms, medical imaging systems, laboratory information systems, and more. This interoperability will enable seamless data sharing, improve care coordination, and enhance the overall healthcare experience for patients and healthcare providers alike.

In this envisioned medical ecosystem, Medsphere can serve as a central hub, facilitating the flow of information and streamlining workflows across different healthcare applications. Through standardized interfaces and well-defined APIs, Medsphere can integrate with specialized applications that cater to specific medical domains, expanding its capabilities and providing healthcare providers with a comprehensive suite of tools for managing various aspects of patient care.

Moreover, Medsphere's commitment to the Unix philosophy ensures scalability and adaptability to emerging technologies and advancements in the healthcare industry. As new applications and technologies evolve, Medsphere can easily integrate them into its ecosystem, leveraging their capabilities to enhance patient care, optimize processes, and stay at the forefront of innovation.

By embracing the Unix philosophy and its principles of modularity, simplicity, and integration, Medsphere paves the way for building a connected medical ecosystem. This approach empowers healthcare providers to leverage the strengths of different applications and technologies, fostering collaboration, interoperability, and ultimately improving the quality of care provided to patients. Medsphere's dedication to creating a scalable and adaptable platform positions it as a key player in shaping the future of healthcare management systems.

2.5 Integration

Here are a few examples of applications that could potentially be part of the Medsphere ecosystem:

1. **Electronic Health Record (EHR) Systems:** Integration with EHR systems would allow seamless transfer of patient data, medical histories, diagnoses, and treatment plans between Medsphere and other EHR platforms. This integration promotes comprehensive patient care and facilitates efficient information exchange between healthcare providers.
2. **Telemedicine Platforms:** Integration with telemedicine platforms would enable healthcare providers to conduct virtual consultations, share medical records, and collaborate with remote specialists. By seamlessly connecting Medsphere with telemedicine applications, patients can access healthcare services from the comfort of their homes, improving access to care and enhancing patient convenience.
3. **Medical Imaging Systems:** Integration with medical imaging systems, such as Picture Archiving and Communication Systems (PACS), would allow healthcare providers to view and analyze medical images directly within Medsphere. This integration streamlines the diagnostic process, enables quicker access to critical imaging data, and facilitates seamless collaboration between radiologists and other healthcare professionals.

4. **Laboratory Information Systems (LIS):** Integration with LIS platforms would facilitate the exchange of laboratory test results, ensuring that healthcare providers can access and review diagnostic information directly within Medsphere. This integration streamlines the process of ordering and receiving test results, expedites decision-making, and enhances patient care.
5. **Medication Management Systems:** Integration with medication management systems would enable healthcare providers to access real-time medication information, including dosage, interactions, and patient-specific instructions. This integration helps reduce medication errors, enhances prescription accuracy, and promotes safe and effective medication management.
6. **Health Monitoring Devices:** Integration with health monitoring devices, such as wearable fitness trackers and remote patient monitoring systems, allows for real-time collection of patient data, including vital signs, activity levels, and other health-related metrics. This integration enhances remote patient monitoring, facilitates early detection of health issues, and supports proactive care management.
7. **Analytics and Business Intelligence Tools:** Integration with analytics and business intelligence platforms would enable Medsphere to leverage advanced data analysis techniques to derive meaningful insights from healthcare data. Integration with such tools facilitates performance monitoring, resource optimization, and evidence-based decision-making, thereby improving the overall efficiency and effectiveness of healthcare operations.

We can see the dawn of artificial intelligence and we can also see our application using some of these in future.

Integration with AI and machine learning applications brings significant benefits to the Medsphere Hospital Management Web App, enhancing its capabilities and improving overall efficiency. Here are some key areas where AI and machine learning integration can have a transformative impact:

1. **Predictive Analytics:** By integrating AI and machine learning algorithms into Medsphere, predictive analytics capabilities can be leveraged to identify patterns and trends in patient data. This enables healthcare providers to proactively identify potential health risks, forecast disease progression, and personalize treatment plans. Predictive analytics can also optimize inventory management by forecasting demand, minimizing wastage, and ensuring the availability of critical medical supplies.

2. **Clinical Decision Support:** AI-powered clinical decision support systems can provide evidence-based recommendations to healthcare providers when diagnosing and treating patients. By integrating such systems into Medsphere, physicians can access relevant medical literature, clinical guidelines, and treatment protocols to make informed decisions. This integration helps enhance accuracy, reduce medical errors, and improve patient outcomes.
3. **Natural Language Processing (NLP):** NLP integration allows Medsphere to analyze and interpret unstructured patient data, such as clinical notes and medical reports. By extracting meaningful information from free-text documents, NLP enables faster and more accurate data processing. It facilitates automated coding, streamlines documentation, and enhances data accuracy and completeness.
4. **Image Analysis and Diagnostics:** Integration with AI and machine learning algorithms enables Medsphere to analyze medical images, such as X-rays, CT scans, and MRIs. Advanced image analysis techniques can assist in automated detection of abnormalities, aiding radiologists and physicians in making accurate diagnoses. This integration reduces interpretation time, improves diagnostic accuracy, and enhances patient care.
5. **Voice Recognition and Virtual Assistants:** Integration with voice recognition technology and virtual assistants allows healthcare providers to interact with Medsphere through voice commands. This hands-free approach simplifies data entry, appointment scheduling, and other routine tasks, saving time and reducing administrative burdens. Virtual assistants can also provide immediate responses to inquiries, offer medication reminders, and deliver personalized healthcare information.
6. **Remote Monitoring and IoT Integration:** Medsphere can integrate with IoT devices and remote monitoring systems to collect real-time patient data. Vital signs, activity levels, and other health-related information can be seamlessly transmitted to the Medsphere system, enabling remote monitoring and timely interventions. Integration with IoT devices promotes patient engagement, facilitates remote care, and improves chronic disease management.

The integration of AI and machine learning applications with Medsphere enhances its capabilities, allowing for predictive analytics, clinical decision support, natural language processing, image analysis, voice recognition, and remote patient monitoring.

Chapter 3

System Design

3.1 Artitecture and Components

Our architecture is govern by the goal of the app. Our goal is to generate the prescription effortlessly. So we need information about all the affairs which are happening in the hospital Like departments , patients, doctors and inventory.

So for this purpose we have to make many modules to auto complete when doctors write the prescription.

Modules:

1. Patient Management Module:

The Patient Management Module is responsible for handling all aspects related to patient information and their medical history. It provides functionalities for patient registration, demographics, and contact details. It allows healthcare providers to track and manage patient appointments, medical visits, and treatment plans. The module also includes features for storing and accessing medical records, laboratory test results, and imaging reports. It ensures that patient data is securely stored, easily accessible, and up-to-date, facilitating efficient patient care and effective communication between healthcare providers.

2. Department Management Module:

The Department Management Module focuses on the organization and administration of different departments within the hospital. It allows administrators to create and manage department profiles, assign healthcare providers to specific departments, and track departmental activities and performance. The module facilitates effective communication and coordination among healthcare teams, ensuring seamless collaboration and efficient workflow management.

3. Doctors Management Module:

The Doctors Management Module is dedicated to managing information related to healthcare providers, including doctors, specialists, and other medical professionals. It facilitates efficient management of their profiles, schedules, and availability within the hospital system. The module allows administrators to create and maintain comprehensive profiles for each doctor, including their contact details, qualifications, specialties, and areas of expertise. These profiles serve as a centralized repository of doctor information, making it easily accessible to other healthcare providers and patients. The module allows doctors to assign and manage their patient caseloads. They can review patient information, assign follow-up appointments, and track the progress of treatment plans. This functionality facilitates continuity of care, ensuring that doctors have access to relevant patient data and can provide personalized treatment recommendations. The module includes features that facilitate seamless communication and collaboration among healthcare providers. Doctors can communicate with other doctors, nurses, and support staff through secure messaging systems within the web app. This functionality enhances interdepartmental coordination, enables quick consultations, and supports team-based care. The module incorporates role-based access control to define different levels of privileges and permissions for doctors and healthcare providers. It ensures that doctors have appropriate access to patient records, diagnostic reports, and other relevant information while maintaining data security and patient confidentiality.

4. Appointment Management Module:

The Appointment Management Module enables healthcare providers to efficiently schedule and manage patient appointments. It includes features such as a shared calendar view, appointment booking, rescheduling, and cancellation functionalities. The module provides real-time availability updates, ensuring that healthcare providers can manage their schedules effectively. It may also incorporate automated reminders and notifications to reduce no-shows and improve appointment adherence. The module optimizes appointment scheduling, improves patient satisfaction, and enhances overall clinic productivity.

5. Prescription Management Module:

The Prescription Management Module streamlines the process of creating and managing patient prescriptions. It allows healthcare providers to generate electronic prescriptions, including medication details, dosages, instructions, and duration. The module integrates with medication databases to provide accurate and up-to-date information on medications, interactions, and contraindications. It includes features such as medication auto-completion, dosage calculation, and prescription history

tracking. The module ensures that prescriptions are error-free, easily readable, and readily available for patients and pharmacists.

6. Inventory Management Module:

The Inventory Management Module is responsible for overseeing the hospital's inventory of medical supplies, equipment, and pharmaceuticals. It includes features such as stock tracking, automated reordering, and supplier management. The module maintains an updated inventory database, tracks stock levels, and generates alerts for

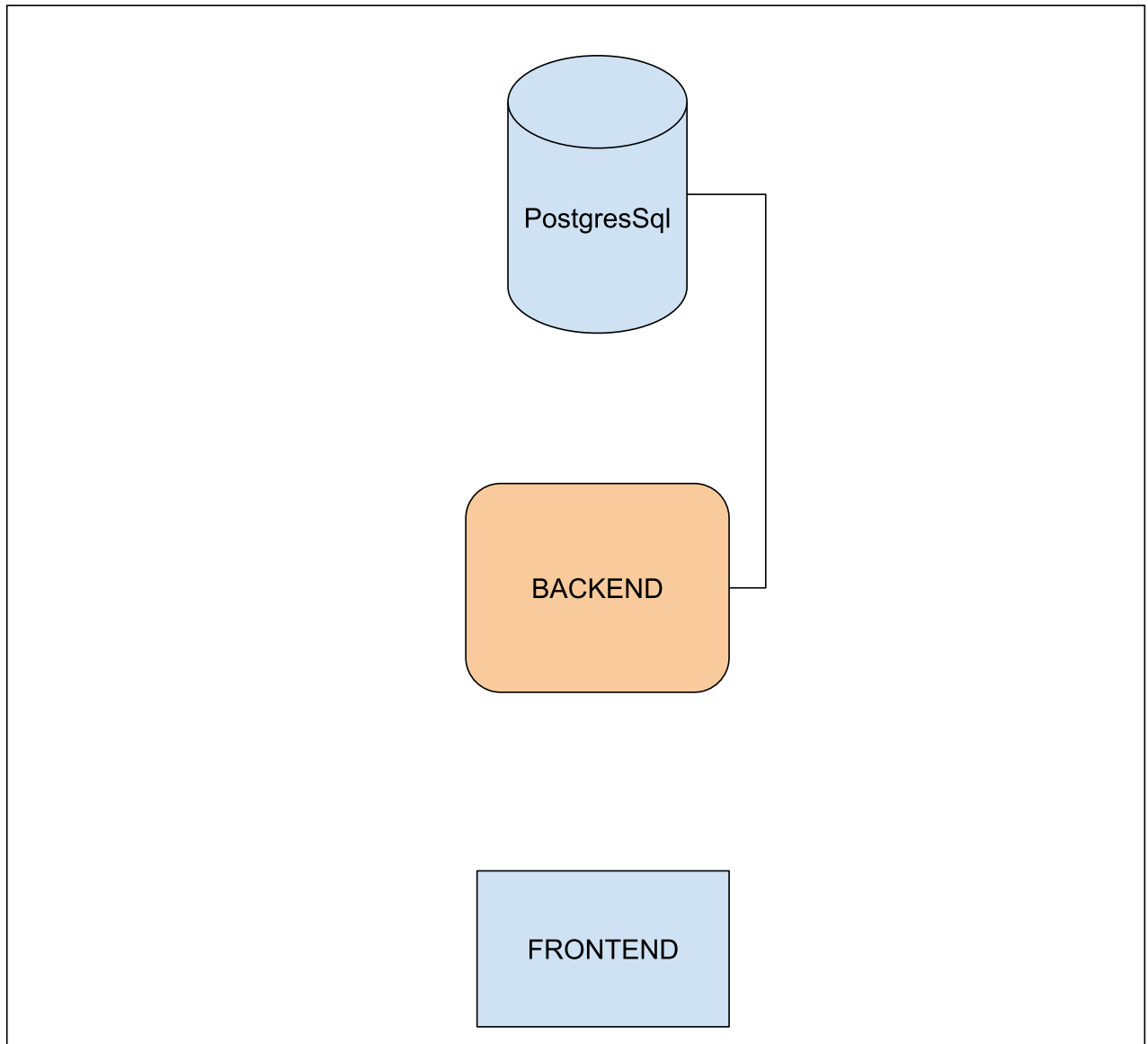


Figure 3.1 High level view of the application

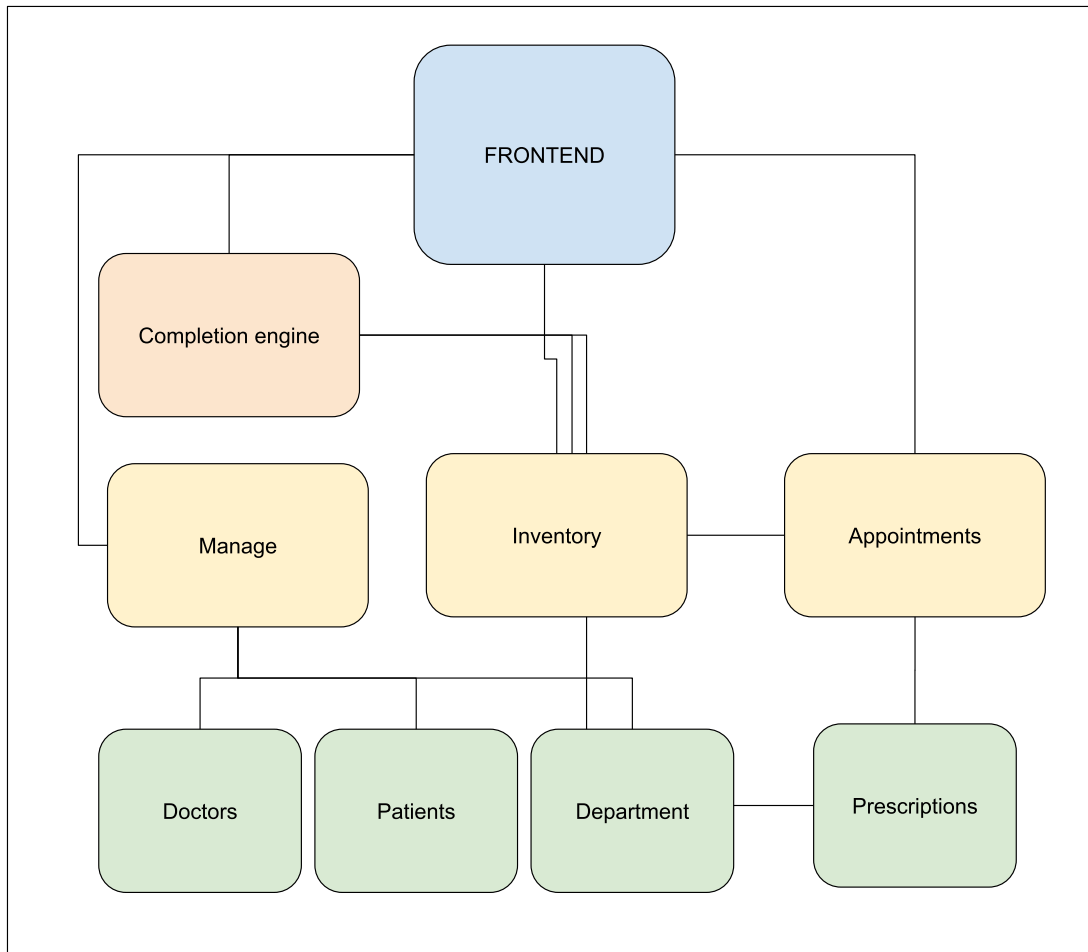


Figure 3.2 Architecture of frontend

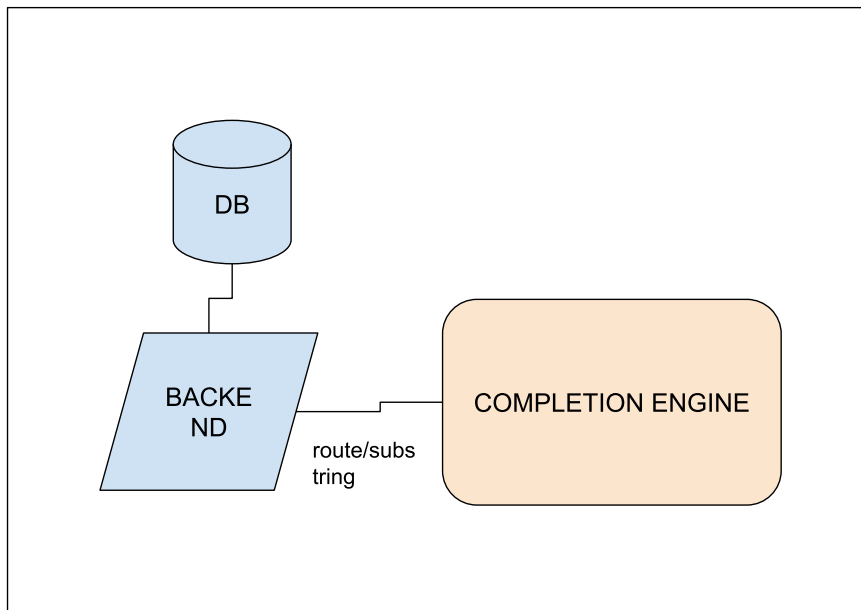


Figure 3.3 Architecture of Completion Engine

3.2 Completion Engine

To implement the auto-completion engine in Medsphere Hospital Management Web App, ensuring efficiency and reducing network overhead:

1. Debounce Technique:

The debounce technique is employed in the auto-completion engine to optimize the search process and reduce unnecessary network requests. When a user begins typing in the search box for medications, the debounce technique introduces a small delay before sending the search query to the backend server. During this delay, if the user continues typing, the debounce function resets the timer, preventing multiple requests from being sent for each keystroke.

By implementing debounce, Medsphere avoids sending numerous requests to the server as the user types rapidly. Instead, only one request is sent when the user pauses or completes typing, reducing network traffic and minimizing the load on the backend server. This improves the overall responsiveness of the auto-completion feature and enhances the user experience by providing quick and accurate suggestions without unnecessary latency.

2. Postgres Text Search:

To implement the auto-completion feature efficiently, Medsphere utilizes the powerful text search capabilities provided by the Postgres database. Postgres offers a specialized text search feature that allows for efficient searching and matching of text-based data. It supports various search techniques, including full-text search, fuzzy matching, and ranking of search results based on relevance.

By leveraging the Postgres text search functionality, Medsphere can quickly retrieve relevant medication names based on the user's input. The database is optimized for handling text-based queries and can efficiently search through a large volume of medication names in real-time. This ensures that the auto-completion suggestions are generated promptly and accurately as the user types.

Additionally, Postgres provides advanced indexing mechanisms, such as trigram indexes, which can further enhance the speed and accuracy of the auto-completion engine. These indexes allow for efficient substring matching and partial word matching, enabling the system to suggest medications based on even a few typed

letters. The Postgres text search capabilities, combined with appropriate indexing techniques, enable Medsphere to deliver precise and responsive auto-completion suggestions to the users.

By combining the debounce technique with the powerful text search capabilities of Postgres, Medsphere optimizes the performance of the auto-completion engine. This approach minimizes unnecessary network requests, reduces latency, and ensures that the suggested medication names are quickly and accurately provided to the users. The implementation of debounce and the utilization of Postgres text search contribute to an efficient and seamless user experience within the Medsphere Hospital Management Web App.

3.3 UI/UX Design

Our primary goal of the app is clean UI. So we go with the minimal UI. UX must be pleasant. Because the number of services can make any one overwhelm so we try to make a effortless UX like manage route. We encapsulate all the utilities to manage the entities in the organisation. We separate inventory and appointment because they deserve their own section. For future design scopes.

3.3.1 Sign in and Sign up Page

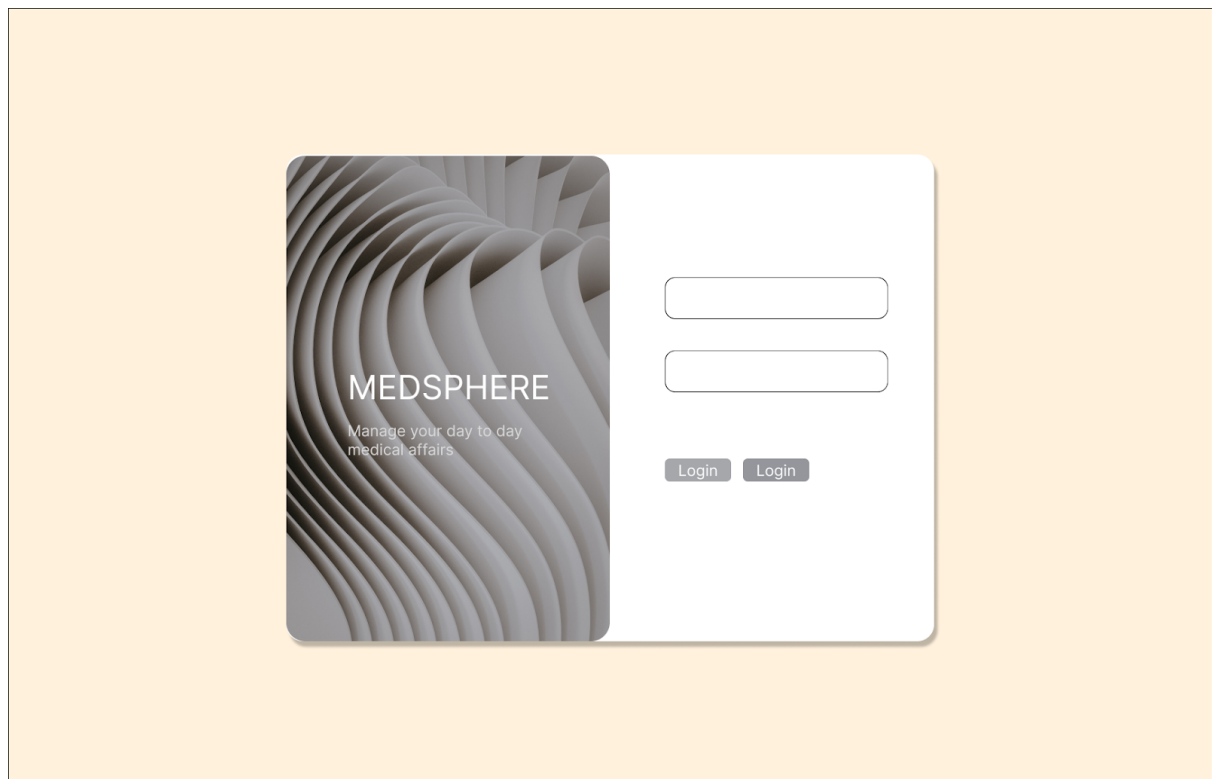


Figure 3.4 Sign In and Sign Up Form

3.3.2 Manage Page

MEDSPHERE

Dashboard

Appointments

Manage

History

Doctors

Name

Age

Department

Rishav Raj

23

Dermatologist

Add a Doctor

Add a Doctor

Name

Email

Password

Confirm Password

Department of Dermatologist

Add

hererere

Figures 3.5 Manage Page

3.3.3 Appointments Page

MEDSPHERE

Dashboard

Appointments

Patients

History

All doctors

10/02/2023

Appointment no.	Patient name	Age	Doctor name

Figures 3.6 Appointments Page

3.3.4 Prescription Page


Dr. Sachin Chaubey M.B.B.S M.D M.S Department of Cardiology REG NO. 1232423		Nalanda Hospital O Pocket, Ganga Nagar, Meerut, Uttar Pradesh 250001 Ph: +9112323233 18003456
ID 2666 - Shivam Yadav (M) Age: 23 Add: modinagar		Date: 27-03-2022
Medicine Name	Dosage	Duration
1. TAB DOLO	1 Morning, 1 Night (BEFORE FOOD)	10 days
2. CAP Sample	1 Morning, 1 Night (AFTER FOOD)	10 days
Advice Given: <ul style="list-style-type: none"> • Avoid spicy food • Exercise daily 		
		<i>Sachin Chaubey</i> Dr. Sachin Chaubey M.B.B.S M.D M.S

Figure 3.7 Prescription Sample

Chapter 4

Implementation

4.1 Frontend Development

The thing which we sure from starting is to make our app look like app not website. So we opt for Single Page Application. So what great than React.js for this. We want an optimised app. We don't worry about SEO. for that we can make a product page.

What we choose and why

1. React JS:

React JS is a popular JavaScript library used for building user interfaces. It provides a component-based approach to frontend development, making it easier to create reusable UI components. Medsphere utilizes React JS as the foundation of its frontend development, allowing for the creation of a modular and scalable user interface.

React JS enables the development team to build dynamic and interactive UI components that respond to user actions efficiently. It also facilitates the management of application state and data flow, ensuring a seamless user experience. React's virtual DOM (Document Object Model) efficiently updates only the necessary parts of the user interface, minimizing performance bottlenecks and enhancing overall application speed.

2. Context API:

Context API is a state management system provided by React JS. It allows for the sharing and management of application-level state across different components without the need for prop drilling. Medsphere utilizes the Context API to manage global states and share data between various components efficiently. For example, it can be used to manage user authentication state, global configurations, or language preferences throughout the application

By leveraging the Context API, Medsphere ensures a consistent and synchronized state across the application, making it easier to access and update shared data without explicitly passing props down the component tree. This enhances code

maintainability, reduces code duplication, and improves overall development productivity.

3. Lodash

Lodash is a JavaScript utility library that provides a wide range of utility functions for simplifying common programming tasks. Medsphere utilizes Lodash to enhance development efficiency and productivity by leveraging its collection of helper functions. Lodash offers utility functions for tasks such as array manipulation, object manipulation, data validation, and functional programming. I use debounce method.

By utilizing Lodash, the development team can optimize code readability, reduce development time, and ensure efficient data manipulation and validation within the Medsphere Hospital Management Web App.

4. jsPDF:

jsPDF is a JavaScript library that allows for the generation of PDF documents on the client-side. Medsphere incorporates jsPDF to enable features such as generating printable reports, prescriptions, or medical records within the web app. With jsPDF, the application can dynamically create PDF documents based on data retrieved from the backend, providing a convenient way to export and share information in a standardized format.

By integrating jsPDF, Medsphere offers healthcare providers and administrators the ability to generate and download PDF documents directly from the web app, reducing the reliance on external tools or manual documentation processes.

5. CSS:

Cascading Style Sheets (CSS) is a styling language used to define the visual appearance and layout of web pages. Medsphere utilizes CSS to create a visually appealing and user-friendly interface for the Hospital Management Web App. CSS allows for the customization of fonts, colors, layouts, and other visual aspects, ensuring a consistent and intuitive design throughout the application.

The development team can leverage CSS frameworks like Bootstrap or Material-UI to expedite the styling process and ensure responsiveness across different devices and screen sizes. CSS provides the flexibility to design a clean and modern user interface that aligns with Medsphere's goal of providing a visually pleasing and user-friendly experience.

4.1.1 Routes Component

Routes component setup my routes. I used react router dom v6 for setting up routes. It is very useful in single page application to change routes without page reloading.

Protected Routes are not accessed by unauthorized person. The AdminRoutes component is conditionally rendered if the user object exists and has the `is_organisation` property set to true. This ensures that only users with administrative privileges or organizational roles can access the routes within this nested block. The routes within the AdminRoutes component include the Manage, InventoryPage, and Settings components.

```
<Routes>
  <Route
    path="/"
    Component={() => (
      <ProtectedRoute auth={!user}>
        <StaticComp />
      </ProtectedRoute>
    )}
  >
    <Route path="/" element={user && <Dashboard />} />
    <Route path="/appointments" element={<Appointments />} />
    <Route path="/patients" element={<Patients />} />
    <Route path="/history" element={<History />} />
    <Route
      path="/prescription/:prescriptionId"
      element={
        <FullScreen>
          <Prescription />
        </FullScreen>
      }
    />
    <Route element={<AdminRoutes admin={user &&
user.is_organisation} />>
      <Route path="/manage" element={<Manage />} />
      <Route path="/inventory" element={<InventoryPage />} />
      <Route path="/settings" element={<Settings />} />
    </Route>
  </Route>
  <Route path="*" element={<NotFound />} />
</Routes>
```

Figure 4.1 Routes component

4.1.2 Static Component

The StaticComp component represents a static layout structure that is shared across multiple pages or components within the Medsphere Hospital Management Web App. It provides a consistent structure and design for the main layout of the application.

```
const StaticComp = () => {  
  return (  
    <div className='StaticComp-container'>  
      <Navbar />  
      <div className='main-layout'>  
        <div className='app-container'>  
          <Sidebar />  
          <Outlet />  
        </div>  
      </div>  
    </div>  
  )  
}
```

Figure 4.2 Static component

1. Navbar:

The Navbar component is rendered at the top of the StaticComp component. It typically contains navigation links, branding elements, user profile information, and other relevant controls. The navbar allows users to access different sections of the application and perform various actions.

2. main-layout and app-container:

The main-layout and app-container div elements define the main content area of the application. They serve as containers for the Sidebar component and the Outlet component.

3. Sidebar:

The Sidebar component is rendered within the app-container div. It typically contains a menu or a navigation panel that provides links or buttons to navigate between different sections or functionalities of the application. The sidebar allows users to easily access different features without having to rely solely on the navbar.

4. Outlet:

The Outlet component represents a placeholder where the content of the current route is rendered. It serves as a dynamic rendering point for the specific component

associated with the current route. When users navigate to different sections of the application, the corresponding component is rendered within the Outlet component, allowing for seamless transitions between different pages or functionalities.

By structuring the layout in this way, the StaticComp component provides a consistent and organized layout for the Medsphere Hospital Management Web App. It includes a navigation bar for easy access to different sections, a sidebar for additional navigation options, and a dynamic content area for rendering the specific components associated with each route. This layout structure enhances the user experience, promotes ease of navigation, and maintains a cohesive design across the application

4.1.3 Manage Component

The giveEntity helper function dynamically render Entities to manage according to selection.

```
function giveEntity(id) {
  switch (id) {
    case 0:
      return <DoctorsEntity />
    case 1:
      return <PatientsEntity />
    case 2:
      return <DepartmentEntity />
  }
}
```

Figure 4.3 giveEntity helper function

```
<div className='manage-container'>
  <Dropdown
    values={entities}
    currentItem={currentEntity}
    setCurrentItem={setCurrentEntity}
    maxWidth={'70px'}
    minWidth={'70px'}
  />
  {giveEntity(currentEntity.id)}
</div>
```

Figure 4.4 Manage Route

Manage Component render the entity and dropdown . Manage Component call the giveEntity function with currentEntity.id to render specific entity from Doctors entity, patients entity and department entity.

4.1.4 Doctors Entity

Doctors Entity encapsulate all the services related to manage doctors in the organisation. This is a admin route which also accessible to admin account.

```

<div className='entityContainer'>
  <DoctorsList />
  <button className='addManager-btn' onClick={() =>
handleDoctorSlider()}>
    <span className='addManager-btn-icon'>
      <FaUserPlus />
    </span>
    <span className='addManger-btn-label'>Add Doctors</span>
  </button>
  <LeftSlideBar open={doctorsSlider}
innerRef={doctorSliderRef}>
    <div className='ManageSlidebar-container'>
      <InputField
        label='Disaplay Name'
        name='displayName'
        onChange={handleChange}
      />

      <InputField label='Age' name='age'
onChange={handleChange} />
      <InputField label='Email' name='email'
onChange={handleChange} />
      <InputField
        label='Password'
        name='password'
        onChange={handleChange}
      />
      <InputField
        label='Confirm password'
        name='confirmPassword'
        onChange={handleChange}

```

```

    />

    <Dropdown
      values={department}
      currentItem={currentDepartment}
      setCurrentItem={setCurrentDepartment}
    />

    <div className='buttons-container'>
      <ButtonPrime text='add' onClick={handleSubmit} />
    </div>
  </div>
</LeftSlideBar>
</div>

```

Figure 4.5 Doctors Entity Component

4.1.5 Patients Entity

Patients Entity encapsulates all the helper function and utils to manage Patients of the organisation.

```

<div className='entityContainer'>
  <PatientsList />

  <button className='addManager-btn' onClick={() =>
handlePatientsSlide()}>
    <span className='addManager-btn-icon'>
      <FaUserPlus />
    </span>
    <span className='addManger-btn-label'
onClick={handlePatientsSlide}>
      Add Patients
    </span>
  </button>

  <LeftSlideBar open={patientsSlider}
innerRef={patientsSliderRef}>
    <div className='ManageSlidebar-container'

```

```

patientsSliderContainer'>
    <InputField
        label='Disaplay Name'
        name='name'
        onChange={handleChange}
    />

    <InputField label='Age' name='age'
onChange={handleChange} />
    <Dropdown
        values={genders}
        currentItem={currentGender}
        setCurrentItem={setCurrentGender}
        maxWidth={'0px'}
        minWidth={'0px'}
    />
    <div className='buttons-container'>
        <ButtonPrime text='add' onClick={handleSubmit} />
    </div>
</div>
</LeftSlideBar>
</div>

```

Figure 4.6 Patients Component

4.1.6 Appointment Component

Appointment component is very complex it levergest many components for managing components and it also give ui to auto completion

```

<div className='AppointmentContainer' id='print'>
    <AppointmentFilter
        date={date}
        setDate={setDate}
        currentDropdown={currentDoctorDropdown}
        handleDoctorsDropdown={setCurrentDoctorDropdown}
    />
    <AppointmentList
        date={date}
        currentDropdownItem={currentDoctorDropdown}
    />
</div>

```

```

        handleDone={({item}) => {
            handleDone(item)
        }}
        Edit={({item}) => {
            handleUpdate(item)
        }}
    />

    const handlePatientsFetch = useCallback(
    debounce(async (value) => {
        const token = getTokenFromLocalStorage()
        if (!token) return
        const { data } = await axios.get(
            `http://localhost:3000/patients/has/${value}`,
            {
                headers: {
                    Authorization: `bearer ${token}`,
                },
            }
        )
        setPatientsArr(data)
    }, 200),
    []
    )

```

Figure 4.7 Appointment Component

4.1.7 Completion Component

This is custom Completion Component which encapsulate a input and a matching text component to render the completion. And many hooks to call when something get clicked.

```

<Completion
    onChange={handleChangeDoctors}
    label='Doctors'
    data={doctorsArr}
    setCurrentItems={setAppointmentForm}
    extractData={{ key1: 'doctors_id', key2: 'uid' }}
    columns={['displayname', 'departmentname']}
    currentItems={appointmentForm}
    value={doctorsInputValue}

```

```

        setInputValue={setDoctorsInputValue}
        currentColumn='displayname'
    />

    <br />

    <Completion
      onChange={handleChangePatients}
      label='Patients'
      data={patientsArr}
      setCurrentItems={setAppointmentForm}
      extractData={{ key1: 'patients_id', key2: 'id' }}
      columns={['name', 'age']}
      currentItems={appointmentForm}
      value={patientsInputValue}
      setInputValue={setPatientsInputValue}
      currentColumn='name'
    />

```

Figure 4.8 Completion Component

This is the internals of Completion Component which uses dbounce from loadash to send request when the user is ideal for 300ms

```

    <div className='suggestionContainer'>
    {data.map((item) => {
      console.log(extractData.key1, item[extractData.key2])
      return (
        <div
          key={item[extractData.key2]}
          className='suggestionList'
          onClick={() => {
            setInputValue(item[currentColumn])
            setCurrentItems({
              ...currentItems,
              [extractData.key1]: item[extractData.key2],
            })
            setSuggestionContainer(false)
          }}
        >

```

Figure 4.9 Suggestion Container

4.1.8 Prescription Component

Very important component of all which provide UI to write prescription and also a page which is the output of the application. It have three groups which is filled by doctor and auto generation engine.

```
<>
  <button onClick={handleExport}>print</button>
  <div id='prescription' className='prescription' ref={input}>
    <div className='groups'>
      <Group1 data={currentPrescription} />
      <Group2 data={currentPrescription} />
      <Group3 data={currentPrescription}
prescriptionId={prescriptionId} />
    </div>
  </div>
</>
```

Figure 4.10 Prescription Component

Hook to fetch medicine from backend by substring, which is types by the doctors while auto completing the medicine name and type.

```
useEffect(() => {
  const getData = setTimeout(async () => {
    const meds = await
getPrescribedMedicineBySubstring(medicine)
    // console.log(meds)
    setMedicineList(meds)
  }, 500)

  return () => clearTimeout(getData)
}, [medicine])
const addType = (type) => {
  switch (type) {
    case 'tablet':
      return (
        <div className='tablet_dosage_selection'>
          <select onChange={dosageSelection}>
            <option>1 morning, 1 night</option>
            <option>3 times a day</option>
            <option>4 times a day</option>
            <option>1 morning</option>
```

```

        <option>1 night</option>
        <option>1 evening</option>
      </select>
    </div>
  )
  case 'syrup':
    return <p>150 ml</p>
  default:
    return (
      <div className='manual-dosage'>
        <input
          type='text'
          value={manualDosage}
          onChange={(e) => {
            setManualDosage(e.target.value)
            setCurrentMedicineToAdd({
              ...currentMedicineToAdd,
              ['dosage']: e.target.value,
            })
          }}
          placeholder='manual dosages'
        />
      </div>
    )
  }
}
const handleClick = (med) => {
  setMedicine(med.brand_name)
  setInputActive(false)
  const { id, dosageform } = med
  const lowerDosageForm = dosageform.toLowerCase()
  setCurrentMedicineToAdd({
    ...currentMedicineToAdd,
    ['id']: id,
    ['dosageform']: lowerDosageForm,
    ['dosage']: '1 morning, 1 night',
  })
}

```

Figure 4.11 Prescription Component

1. Prescription Context:

The component utilizes the PrescriptionContext and its associated methods, such as currentPrescription and getPrescriptionById. These values and functions are obtained through the useContext hook, allowing the component to access and manage prescription-related data. This is defined in prescription.context.js which contains all the helper functions which is used by Prescription component to talk to databases. These helper function is available to every component avoiding prop drilling.

2. useParams:

The useParams hook is used to extract the prescriptionId from the URL parameters. It allows the component to determine which prescription to display based on the prescriptionId parameter.

3. useEffect and getPrescriptionById:

The useEffect hook is used to trigger the getPrescriptionById function when the component mounts. This function is responsible for fetching the specific prescription data based on the provided prescriptionId. By calling this function, the component ensures that the relevant prescription information is retrieved and available for display.

4. handleExport:

The handleExport function is invoked when the "print" button is clicked. It triggers the printDocument function (presumably defined elsewhere) to initiate the printing process. The input ref, assigned to the prescription div element, is passed as a parameter to the printDocument function, allowing the content of the prescription to be printed.

5. Rendering Prescription Details:

The component renders the prescription details within the prescription div. It consists of three groups (Group1, Group2, and Group3), each receiving the currentPrescription data as props. These groups are responsible for rendering different sections of the prescription, such as patient information, medication details, and additional notes.

Overall, the Prescription component provides functionality to retrieve and display a specific prescription based on the provided prescriptionId. It allows users to export and print the prescription using the printDocument function. The component leverages context, URL parameters, and ref to manage the prescription data and enable a seamless viewing experience for healthcare providers and other authorized users.

4.2 Backend Development

In backend i uses Express.js , postgresql driver and few custom middlewares to handle error.

1. Express.js:

Express.js is a popular web application framework for Node.js. Medsphere utilizes Express.js as the backend framework for handling HTTP requests and building the server-side logic of the web app. Express.js provides a simple and flexible way to define routes, handle requests, and manage middleware.

With Express.js, the development team can create route handlers, middleware functions, and implement various API endpoints to handle data retrieval, manipulation, and authentication. It allows for the seamless integration of different modules and components, making it easier to develop and maintain a scalable backend architecture.

2. PostgreSQL Driver:

The PostgreSQL driver is used to establish a connection between the Medsphere Hospital Management Web App and the PostgreSQL database. The driver provides the necessary tools and functionalities to execute database queries, retrieve data, and perform CRUD (Create, Read, Update, Delete) operations.

By utilizing the PostgreSQL driver, Medsphere can securely store and manage medical data, patient records, prescriptions, and other relevant information in a reliable and efficient database system

3. Custom Middleware Handlers:

Middleware functions play a crucial role in the Express.js framework. Medsphere incorporates custom middleware handlers to perform various tasks, including error handling, request validation, authentication, and logging.

- a. `programmerErrorHandler`:

The `programmerErrorHandler` middleware function is responsible for handling errors that occur during the execution of the application code. Here's an overview of its functionality:

- b. `operationalErrorHandler`:

The `operationalErrorHandler` middleware function is specifically designed to handle operational errors. Here's an explanation of its functionality:

```

function programmerErrorHandler(error, req, res, next) {
  console.error(error)

  if (error.isOperational) {
    return next(error)
  }

  return res.status(error.statusCode).send('Internal server
error')
}

function operationalErrorHandler(error, req, res, next) {
  const { isOperational, ...restError } = error
  return res.status(error.statusCode).send(restError)
}

module.exports = { programmerErrorHandler, operationalErrorHandler
}

```

Figure 4.12 Programmer ErrorHandler Middleware

These middleware functions help to handle and manage errors in the Medsphere Hospital Management Web App. The programmerErrorHandler handles general application errors, logging them for debugging purposes and providing an appropriate response to the client. Custom Error object which extends Error to log on to the sever and send to clients.

```

class AppError extends Error {
  constructor(name, statusCode, description, isOperational) {
    super(description)
    Error.call(this)
    Error.captureStackTrace(this)
    this.name = name
    this.statusCode = statusCode
    this.description = description
    this.isOperational = isOperational
  }
}

module.exports = AppError

```

Figure 4.13 AppError Error Object

4.2.1 Api Routes

The provided code showcases the routing configuration for the Medsphere Hospital Management Web App using Express.js. Let's go through each section and explain its functionality:

```
app.get('/', (req, res) => {
  res.send('welcome to medsphere api')
})

app.use('/static', express.static('Images'))
app.use('/auth', authRoute)
app.use('/users', usersRoute)
app.use('/department', departmentRoute)
app.use('/patients', patientsRoute)
app.use('/appointments', appointmentsRoute)
app.use('/inventory', inventoryRoute)
app.use('/prescription', prescriptionRoute)
app.use('/upload', uploadRoute)
app.use(programmerErrorHandler)
app.use(operationalErrorHandler)

module.exports = app
```

Figure 4.14 Api Routes

1. Root Route:

The `app.get('/', ...)` defines the route for the root URL ("/"). When a GET request is made to the root URL, the callback function sends the response with the message "welcome to medsphere api". This serves as a welcome message or a basic landing page for the API.

2. Static Files Route:

The `app.use('/static', express.static('Images'))` configures a route for serving static files. It uses the `express.static` middleware to specify that any requests starting with "/static" should be served from the "Images" directory. This allows the web app to serve static files like images, stylesheets, or client-side JavaScript files.

3. Subroutes:

The subsequent `app.use(...)` statements define the subroutes for various features and resources of the Medsphere web app. Each subroute is mounted with its corresponding router module, such as `authRoute`, `usersRoute`, `departmentRoute`,

patientsRoute, appointmentsRoute, inventoryRoute, prescriptionRoute, and uploadRoute. These subroutes define the endpoints and route handlers for managing authentication, user data, departments, patients, appointments, inventory, prescriptions, and file uploads, respectively.

4. Error Handling Middleware:

The last two `app.use(...)` statements are middleware functions for handling errors. The `programmerErrorHandler` middleware is called when an error occurs during the execution of the application code. It logs the error and handles it accordingly. The `operationalErrorHandler` middleware is responsible for handling operational errors and sending appropriate error responses to the client.

4.2.2 Component routes

1. `app.use('/auth', authRoute):`

This line mounts the `authRoute` module at the `"/auth"` base URL. It means that any requests starting with `"/auth"` will be forwarded to the `authRoute` module for handling. This subroute is typically responsible for managing authentication-related endpoints, such as user registration, login, logout, and password reset.

2. `app.use('/users', usersRoute):`

This line mounts the `usersRoute` module at the `"/users"` base URL. Requests starting with `"/users"` will be directed to the `usersRoute` module for processing. This subroute is responsible for handling user-related operations, such as retrieving user information, updating user profiles, and managing user roles or permissions.

3. `app.use('/department', departmentRoute):`

This line mounts the `departmentRoute` module at the `"/department"` base URL. Requests starting with `"/department"` will be passed to the `departmentRoute` module. This subroute handles operations related to managing departments within the hospital system, such as creating new departments, updating department information, and retrieving department details.

4. `app.use('/patients', patientsRoute):`

Here, the `patientsRoute` module is mounted at the `"/patients"` base URL. Any requests starting with `"/patients"` will be delegated to the `patientsRoute` module. This subroute handles patient-related operations, including patient registration, retrieving patient records, updating patient information, and managing medical histories.

5. `app.use('/appointments', appointmentsRoute):`

This line mounts the `appointmentsRoute` module at the `"/appointments"` base URL. Requests starting with `"/appointments"` will be routed to the `appointmentsRoute` module. This subroute is responsible for managing appointments, such as scheduling appointments, retrieving appointment details, and handling appointment-related actions like cancellation or rescheduling.

6. `app.use('/inventory', inventoryRoute):`

Here, the `inventoryRoute` module is mounted at the `"/inventory"` base URL. Requests starting with `"/inventory"` will be forwarded to the `inventoryRoute` module for processing. This subroute handles inventory management operations, including adding new items to the inventory, updating item quantities, and retrieving inventory details.

7. `app.use('/prescription', prescriptionRoute):`

This line mounts the `prescriptionRoute` module at the `"/prescription"` base URL. Requests starting with `"/prescription"` will be directed to the `prescriptionRoute` module. This subroute handles prescription-related functionalities, such as generating prescriptions, retrieving prescription details, and managing prescription history.

4.2.3 Auth Route

It uses `jwt` middlewares for authentications. Every requisition comes to these two route will go through `jwt` middlewares. Implementation of the `authRoute` module, which handles authentication-related endpoints in the Medsphere Hospital Management Web App. By defining these routes and associating them with the respective controller functions, the `authRoute` module handles user authentication operations such as login and registration

```
const express = require('express')
const router = express.Router()
const auth = require('./auth.controller')
const { validate } = require('../middlewares/schemaValidator')
const { loginSchema, registerSchema } = require('./auth.schema')

router.post('/login', validate(loginSchema), auth.login)
router.post('/register', validate(registerSchema), auth.register)
```

Figure 4.15 Auth Route

Login Controller

```
auth.login = async (req, res, next) => {
  const { email, password } = req.body

  try {
    const { data: user } = await users.getByEmail(email)
    if (!user)
      return next(new AppError('Client Error', 401, 'Wrong
credentials', true))

    if (user.password !== password)
      return next(
        new AppError('Unauthorized', 401, "Password didn't match",
true)
      )

    const token = createAccessToken(user.uid)
    if (token) return res.send(token)
  } catch (error) {
    next(new AppError('Internal sever error', 500, error.message,
false))
  }
}
```

Figure 4.16 Login Controller

Registration Controller

```
auth.register = async (req, res, next) => {
  console.log(req.body)
  const {
    displayName,
    email,
    age,
    password,
    organisation_id,
    is_organisation,
```

```

    department_id,
  } = req.body

  try {
    const userExists = await users.exists(email)

    if (userExists)
      return next(
        new AppError('Client Error', 409, 'email already in use',
true)
      )

    const { data: user } = await users.add(
      displayName,
      age,
      email,
      password,
      organisation_id,
      is_organisation,
      department_id
    )

    const token = createAccessToken(user.uid)

    res.send(token)
  } catch (error) {
    next(new AppError('Internal server error', 500, error.message,
false))
  }
}

```

Figure 4.17 Registratio Controller

```

const jwt = require('jsonwebtoken')
const config = require('../config/config.js')
const AppError = require('../utils/AppError.js')
const jwtUtils = {}
jwtUtils.createAccessToken = (uid) => {
  return jwt.sign({ id: uid }, config.JWT_SECRET)
}

```

Figure 4.18 Jwt utils


```

jwtUtils.verify = (req, res, next) => {
  const header = req.header('Authorization')
  const token = header && header.split(' ')[1]

  if (!token)
    return next(new AppError('Unauthorized', 402, 'token not
provided', true))

  try {
    const resp = jwt.verify(token, config.JWT_SECRET)
    req.id = resp.id
  } catch (error) {
    return next(new AppError('Unauthorized', 402, error.message,
true))
  }
  next()
}

```

Figure 4.19 Verify middleware for auth

4.2.3 Users Route

Users Controller contains three method `users.get()`, `users.getAllDoctors()`, `users.getBySubstring()`.

1. `users.get`:

This function is responsible for handling the GET request to retrieve a specific user's information. It first extracts the user ID from the request (`req.id`). It then calls the `getById` function from the `usersDb` module to fetch the user data based on the extracted ID. If the data is found, it is sent as the response. If the data is not found, it forwards the request to the error handling middleware with an appropriate error message indicating that the user doesn't exist.

```

users.get = async (req, res, next) => {
  const uid = req.id

  try {
    const { data } = await usersDb.getById(uid)

```

```

    if (!data)
      return next(
        new AppError('Client Error', 409, "User doesn't exists",
true)
      )
    res.send(data)
  } catch (error) {
    next(new AppError('Internal server error', 502, error.message,
false))
  }
}

```

Figure 4.20 Users Route

2. users.getAllDoctors:

This function handles the GET request to fetch all doctors' information. It retrieves the user ID from the request (req.id) and calls the getAllDoctors function from the usersDb module to retrieve the data of all doctors. The fetched data is sent as the response.

```

users.getAllDoctors = async (req, res, next) => {
  const uid = req.id

  try {
    const { data } = await usersDb.getAllDoctors(uid)

    res.send(data)
  } catch (error) {
    next(new AppError('Internal server error', 502, error.message,
false))
  }
}

users.getBySubstring = async (req, res, next) => {
  const uid = req.id
  const substring = req.params.substring

  try {
    const { data } = await usersDb.getBySubstring(substring, uid)

    res.send(data)
  } catch (error) {

```

```

        next(new AppError('Internal server error', 502, error.message,
false))
    }
}

```

Figure 4.21 Get all doctors

3. users.getBySubstring:

This function handles the GET request to search for users based on a provided substring. It extracts the user ID from the request (req.id) and the substring from the request parameters (req.params.substring). It then calls the getBySubstring function from the usersDb module, passing the substring and user ID as parameters. The function retrieves the data of users whose names match the provided substring and sends it as the response.

4.3 Database Schemas

4.3.1 Users Schema

```

CREATE TABLE public.users (
    uid serial4 NOT NULL,
    displayname varchar(100) NOT NULL,
    email varchar(100) NOT NULL,
    "password" varchar(300) NOT NULL,
    organisation_id int4 NULL,
    is_organisation bool NULL DEFAULT false,
    department_id int4 NULL,
    age int4 NULL,
    address text NULL,
    phone_number text NULL,
    qualifications text NULL,
    CONSTRAINT users_email_key UNIQUE (email),
    CONSTRAINT users_pkey PRIMARY KEY (uid)
);

```

Figure 4.22 Users Schema

4.3.2 Patients Schema

```
CREATE TABLE public.patients (  
    id serial4 NOT NULL,  
    "name" varchar(130) NOT NULL,  
    age int4 NOT NULL,  
    gender varchar(50) NULL,  
    org_id int4 NULL,  
    address text NULL,  
    blood_group text NULL,  
    CONSTRAINT patients_pkey PRIMARY KEY (id)  
);
```

Figure 4.23 Patients Schema

4.3.3 Department Schema

```
CREATE TABLE public.department (  
    id serial4 NOT NULL,  
    "name" varchar(200) NOT NULL,  
    org_id int4 NOT NULL,  
    CONSTRAINT department_pkey PRIMARY KEY (id)  
);
```

Figure 4.24 Department Schema

4.3.4 Medicine Schema

```
CREATE TABLE public.medicine (  
    id serial4 NOT NULL,  
    brand_name varchar(50) NULL,  
    "type" varchar(50) NULL,  
    generic varchar(70) NULL,  
    dosageform text NULL,  
    manufacturer varchar(100) NULL,  
    org_id int4 NULL,  
    quantity int4 NULL,  
    CONSTRAINT medicine_pkey PRIMARY KEY (id)  
);
```

Figure 4.25 Medicine Schema

4.3.5 Appointments Schema

```
CREATE TABLE public.appointments (  
    id serial4 NOT NULL,  
    patients_id int4 NOT NULL,  
    doctors_id int4 NOT NULL,  
    "timestamp" timestamp NOT NULL,  
    org_id int4 NOT NULL,  
    status varchar(100) NULL,  
    CONSTRAINT appointments_pkey PRIMARY KEY (id)  
);
```

Figure 4.26 Appointment Schema

4.3.6 Prescription Schema

```
CREATE TABLE public.prescription (  
    id serial4 NOT NULL,  
    appointment_id int4 NOT NULL,  
    CONSTRAINT prescription_pkey PRIMARY KEY (id)  
);
```

Figure 4.27 Prescription Schema

4.3.7 Prescribed Medicine schema

```
CREATE TABLE public.prescribed_medicine (  
    id serial4 NOT NULL,  
    prescription_id int4 NOT NULL,  
    medicine_id int4 NOT NULL,  
    dosage text NULL,  
    duration text NULL,  
    CONSTRAINT prescribed_medicine_pkey PRIMARY KEY (id)  
);
```

Figure 4.28 Prescribed Medicine schema

4.3.8 Prescribed Advice schema

```
CREATE TABLE public.prescribed_advice (  
    id serial4 NOT NULL,
```

```

    prescription_id int4 NOT NULL,
    advice text NOT NULL,
    CONSTRAINT prescribed_advice_pkey PRIMARY KEY (id)
);

```

Figure 4.29 Prescribed Advice Schema

4.3.9 Constrains between Schemas

```

ALTER TABLE public.users ADD CONSTRAINT
user_department_fk_department_id FOREIGN KEY (department_id)
REFERENCES public.department(id);
ALTER TABLE public.users ADD CONSTRAINT users_organisation_id_fkey
FOREIGN KEY (organisation_id) REFERENCES public.users(uid);

ALTER TABLE public.patients ADD CONSTRAINT fk_organisation FOREIGN
KEY (org_id) REFERENCES public.users(uid) on delete cascade;

ALTER TABLE public.medicine ADD CONSTRAINT
medicine_org_id_constraint FOREIGN KEY (org_id) REFERENCES
public.users(uid) ON DELETE CASCADE;

ALTER TABLE public.department ADD CONSTRAINT
department_users_org_id FOREIGN KEY (org_id) REFERENCES
public.users(uid) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE public.appointments ADD CONSTRAINT
fk_doctors_doctors_id FOREIGN KEY (doctors_id) REFERENCES
public.users(uid) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE public.appointments ADD CONSTRAINT
fk_patients_patients_id FOREIGN KEY (patients_id) REFERENCES
public.patients(id) ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE public.appointments ADD CONSTRAINT fk_users_org_id
FOREIGN KEY (org_id) REFERENCES public.users(uid) ON DELETE
CASCADE ON UPDATE CASCADE;

ALTER TABLE public.prescription ADD CONSTRAINT
prescription_appointment_id_constraint FOREIGN KEY

```

```

(appointment_id) REFERENCES public.appointments(id) ON DELETE
CASCADE;

ALTER TABLE public.prescribed_medicine ADD CONSTRAINT
prescribed_medicine_medicine_id_constraint FOREIGN KEY
(medicine_id) REFERENCES public.medicine(id) ON DELETE CASCADE;
ALTER TABLE public.prescribed_medicine ADD CONSTRAINT
prescribed_medicine_prescription_id_constraint FOREIGN KEY
(prescription_id) REFERENCES public.prescription(id) ON DELETE
CASCADE;

ALTER TABLE public.prescribed_advice ADD CONSTRAINT
prescribed_advice_prescription_id_constraint FOREIGN KEY
(prescription_id) REFERENCES public.prescription(id) ON DELETE
CASCADE;

```

Figure 4.30 Contrains between table

The decision to use PostgreSQL as the database management system for the Medsphere Hospital Management Web App is based on several factors, including the need for efficient handling of complex relationships between entities and the requirement for fast text search capabilities to optimize the auto-completion feature. Let's delve deeper into these aspects:

1. Handling Complex Relationships:

In a hospital management system, various entities such as patients, doctors, appointments, prescriptions, and inventory items have intricate relationships with each other. PostgreSQL, being a robust relational database, provides excellent support for managing complex relationships through its powerful features like tables, foreign key constraints, and query optimization. By utilizing PostgreSQL, Medsphere can efficiently store and retrieve data while maintaining the integrity of relationships between entities

2. Efficient Text Search:

The auto-completion feature in Medsphere requires fast and accurate text search capabilities. PostgreSQL offers advanced text search functionality through its full-text search feature. This feature enables the system to perform efficient and optimized searches across large volumes of text data, such as medicine names, patient names, or doctor names. By leveraging PostgreSQL's text search capabilities, Medsphere can provide users with quick and accurate auto-completion suggestions, enhancing the user experience and productivity.

Chapter 5

System Testing

5.1 Testing Methodology

Testing is a critical aspect of software development, ensuring that the Medsphere Hospital Management Web App functions reliably, performs as expected, and meets the specified requirements. The testing methodology for Medsphere involves several stages and techniques to validate the functionality, performance, security, and usability of the application. Let's explore each phase in detail:

1. Requirement Analysis:

The testing process begins with a thorough analysis of the system requirements, including functional, non-functional, and user experience aspects. This analysis helps define the testing scope, objectives, and test scenarios for Medsphere.

2. Test Planning:

In this phase, a comprehensive test plan is developed, outlining the testing approach, strategies, resources, and timelines. The plan includes identification of test types (e.g., unit testing, integration testing, system testing) and defines the test environment, test data, and tools required for testing.

3. Test Case Design:

Test cases are designed to cover various aspects of the application's functionality. Each test case specifies a set of inputs, expected outcomes, and test procedures. Test cases are categorized based on different modules and functionalities, such as patient management, appointment scheduling, prescription generation, and inventory management.

4. Unit Testing:

Unit testing focuses on validating individual components or modules of the application. Developers write unit tests to ensure that each unit performs as expected and adheres to the defined specifications. Unit testing frameworks and tools, such as Jest or Mocha, are used to automate the execution of unit tests.

5. Integration Testing:

Integration testing verifies the interaction and collaboration between different components/modules of Medsphere. It ensures that the integrated system functions correctly and handles data flow, communication, and dependencies effectively. Integration tests are performed using techniques like top-down or bottom-up integration.

6. System Testing:

System testing evaluates the application as a whole to validate its compliance with the specified requirements. It encompasses end-to-end testing scenarios, including user flows, user interfaces, data integrity, error handling, security, and performance. Testers execute various test cases to assess the system's behavior and identify any functional or non-functional issues.

7. Performance Testing:

Performance testing measures the application's responsiveness, scalability, and stability under expected and peak load conditions. It involves stress testing, load testing, and performance profiling to identify performance bottlenecks, optimize resource utilization, and ensure that the application performs well under heavy usage.

8. Security Testing:

Security testing focuses on identifying vulnerabilities, weaknesses, and risks associated with the application's security. Testers perform penetration testing, vulnerability scanning, and authentication/authorization testing to ensure that user data is protected, sensitive information is encrypted, and access controls are in place.

9. Usability Testing:

Usability testing evaluates the user-friendliness, intuitiveness, and overall user experience of Medsphere. Testers observe users interacting with the application, gather feedback, and assess the ease of navigation, clarity of information, and efficiency of common tasks. This feedback helps refine the user interface and enhance the overall usability of the application.

10. Test Execution and Defect Tracking:

During test execution, testers execute the prepared test cases, record test results, and identify any deviations from expected behavior. Defects and issues are logged into a defect tracking system, such as Jira or Bugzilla, with detailed information about the

problem, steps to reproduce, and severity level. Defects are then assigned to the development team for resolution.

11. Regression Testing:

Regression testing ensures that the changes or fixes made in response to identified defects do not introduce new issues or impact existing functionalities. It involves re-executing selected test cases to verify the stability and integrity of the system after modifications.

12. Test Reporting:

Test reporting involves documenting the test results, including the status of test cases, identified defects, and overall

5.2 Frontend testing

Jest is a popular JavaScript testing framework widely used for testing React applications, including Medsphere Hospital Management Web App. It provides a comprehensive and efficient testing environment with a focus on simplicity and ease of use. Let's explore some key aspects and features of Jest:

1. Test Suites and Test Cases:

Jest organizes tests into logical units called test suites and individual test cases. Test suites group related test cases, while test cases represent specific scenarios to be tested. For Medsphere, test suites can be created for different components, modules, or functionalities, such as patient management, appointment scheduling, or prescription generation.

2. Test Assertions:

Jest offers a wide range of built-in assertions that allow developers to define expected behaviors and compare actual results with expected outcomes. These assertions verify that the application functions as intended and meets the defined requirements. Examples of assertions include checking if a component renders correctly, if a function returns the expected value, or if a specific UI element is present.

3. Asynchronous Testing:

React applications often involve asynchronous operations like fetching data from an API or handling user interactions. Jest provides robust support for testing asynchronous code by offering features like asynchronous test cases, promises, and

async/await syntax. This allows developers to write test cases that cover scenarios involving asynchronous behavior, ensuring the accuracy and reliability of Medsphere.

4. Mocking and Spies:

Jest includes powerful mocking and spying capabilities to isolate components or functions during testing. This is particularly useful when testing components that rely on external dependencies or APIs. By creating mock versions of these dependencies, developers can control their behavior and simulate different scenarios, ensuring thorough testing of Medsphere's functionalities.

5. Snapshot Testing:

Snapshot testing is a valuable feature of Jest that enables developers to capture and compare snapshots of rendered components or data structures. It allows for easy detection of unintended changes or regressions in UI components by comparing the current output with a previously saved snapshot. Snapshot testing provides a quick and effective way to ensure consistent rendering and prevent UI-related issues in Medsphere.

6. Code Coverage Reporting:

Jest provides code coverage reporting, which measures the percentage of code that is covered by tests. This feature helps identify areas of the codebase that lack test coverage, allowing developers to focus on writing additional tests for those areas. With code coverage reporting, Medsphere can ensure comprehensive test coverage, leading to higher code quality and reduced risk of bugs or issues.

7. Test Watch Mode:

Jest's watch mode is a convenient feature that automatically detects changes in files and re-runs relevant tests. It speeds up the testing process during development by eliminating the need to manually trigger test runs after code changes. This iterative approach enables developers to quickly identify issues and validate the changes they make in real-time.

8. Integration with React Ecosystem:

Jest seamlessly integrates with other tools and libraries commonly used in the React ecosystem. It works well with React's testing utilities (such as React Testing Library or Enzyme) and can be easily configured to test React components with different rendering engines or state management libraries. This flexibility ensures compatibility with Medsphere's React-based architecture and facilitates efficient testing.

Jest offers a robust and feature-rich testing framework for Medsphere Hospital Management Web App. With its intuitive syntax, built-in assertions, asynchronous testing support, mocking capabilities, snapshot testing, code coverage reporting, and seamless integration with the React ecosystem, Jest empowers developers to write comprehensive tests, ensure application quality, and maintain a reliable and robust codebase.

```
import { render, fireEvent, screen } from
"@testing-library/react";
import Counter from "../components/Counter";

//test block
test("increments counter", () => {
  // render the component on virtual dom
  render(<Counter />);

  //select the elements you want to interact with
  const counter = screen.getByTestId("counter");
  const incrementBtn = screen.getByTestId("increment");

  //interact with those elements
  fireEvent.click(incrementBtn);

  //assert the expected result
  expect(counter).toHaveTextContent("1");
});
```

Figure 5.1 Testing frontend with jest

5.3 Backend testing

Testing the backend of Medsphere Hospital Management Web App with Jest involves writing test cases to verify the functionality and behavior of server-side code, API endpoints, database interactions, and other backend components. Here's an overview of how Jest can be used for backend testing:

1. Unit Testing:

Unit tests focus on testing individual units of code in isolation, such as functions, modules, or middleware. With Jest, you can write unit tests for backend components like authentication handlers, data validation functions, utility functions, and database queries. These tests ensure that each unit behaves as expected and handles different scenarios correctly.

2. Integration Testing:

Integration tests verify the interaction between various components of the backend system. They test the integration of different modules, database connections, external services, and API endpoints. Jest provides a testing environment where you can simulate requests and responses, mock external dependencies, and test the overall system behavior. Integration tests help identify issues that arise when multiple components interact, ensuring the seamless functioning of Medsphere.

3. Mocking Dependencies and APIs:

Jest allows you to easily create mock versions of external dependencies, APIs, or services used in the backend. By mocking these dependencies, you can simulate their behavior, define specific responses, and ensure predictable test outcomes. This helps in isolating the code being tested and ensures that the tests focus solely on the functionality of the backend components.

4. Asynchronous Testing:

Many backend operations involve asynchronous operations like database queries or network requests. Jest provides support for testing asynchronous code using promises, `async/await` syntax, or the use of `done` callbacks. This allows you to write test cases that handle asynchronous operations correctly, ensuring that promises are resolved, errors are handled, and the expected results are returned.

5. Database Testing:

When Medsphere interacts with a database, Jest can be used to test the database operations and ensure data integrity. You can create test cases to insert, retrieve, update, and delete data, and verify that the database queries and transactions are executed correctly. Additionally, Jest provides the ability to set up a test database and seed it with predefined data for consistent testing.

6. Error Handling:

Jest enables you to write test cases to validate error handling and ensure that appropriate error responses are returned. You can simulate different error scenarios, throw exceptions, and assert that the correct error messages or status codes are received. Testing error handling ensures that Medsphere gracefully handles errors and provides informative responses to clients.

7. Code Coverage Reporting:

Jest provides code coverage reporting, allowing you to measure the percentage of code covered by tests. This feature helps identify areas of the backend that lack test coverage and guides you in writing additional tests to improve overall coverage. With code coverage reporting, you can ensure that critical components of Medsphere's backend are thoroughly tested.

8. Continuous Integration:

Jest integrates well with popular continuous integration (CI) platforms like Jenkins, Travis CI, or CircleCI. You can configure your CI pipeline to execute Jest tests automatically on every code commit or pull request. This ensures that any changes made to the backend code are validated through automated testing, providing continuous feedback on the code quality.

By leveraging Jest for backend testing, you can ensure the reliability, stability, and functionality of Medsphere Hospital Management Web App. Unit tests, integration tests, mocking dependencies, asynchronous testing, database testing, error handling, code coverage reporting, and continuous integration support provided by Jest contribute to a robust testing strategy for the backend components.

```
const departmentDb = require('./departmentDb'); // Assuming
departmentDb module is imported

describe('department.add', () => {
  it('should add a department and return the data', async () => {
    // Mock the request and response objects
    const req = {
      id: 'user-id',
      body: {
        name: 'Department Name',
      },
    };
    const res = {
      send: jest.fn(),
    };
    const next = jest.fn();

    // Mock the departmentDb.add function
    departmentDb.add = jest.fn().mockResolvedValue({ data:
```

```
'Department Added' }));

    // Call the department.add function
    await department.add(req, res, next);

    // Verify that departmentDb.add is called with the correct
arguments
    expect(departmentDb.add).toHaveBeenCalledWith('Department
Name', 'user-id');

    // Verify that res.send is called with the correct data
    expect(res.send).toHaveBeenCalledWith('Department Added');

    // Verify that next is not called
    expect(next).not.toHaveBeenCalled();
});

it('should handle errors and call next with an error', async ()
=> {
    // Mock the request and response objects
    const req = {
        id: 'user-id',
        body: {
            name: 'Department Name',
        },
    };
    const res = {
        send: jest.fn(),
    };
    const next = jest.fn();

    // Mock the departmentDb.add function to throw an error
    departmentDb.add = jest.fn().mockRejectedValue(new
Error('Database Error'));

    // Call the department.add function
    await department.add(req, res, next);

    // Verify that departmentDb.add is called with the correct
arguments
    expect(departmentDb.add).toHaveBeenCalledWith('Department
Name', 'user-id');
```

```

    // Verify that res.send is not called
    expect(res.send).not.toHaveBeenCalled();

    // Verify that next is called with the correct error
    expect(next).toHaveBeenCalledWith(new AppError('Server Error',
400, 'Database Error', false));
  });
});

```

Figure 5.2 Jest snippet to test express js component

In this example, we are using Jest to mock the request and response objects (req and res), as well as the departmentDb.add function. We have two test cases: one to test the successful addition of a department and another to test error handling. We assert that the function behaves as expected by verifying that the appropriate function calls are made and the expected data is returned or error is thrown.

5.4 Api testing

I am using a fast Neovim http client written in Lua. rest.nvim makes use of a curl wrapper made in pure Lua by tami5 and implemented in plenary.nvim so, in other words, rest.nvim is a curl wrapper so you don't have to leave Neovim!

```

GET http://localhost:3000/users/get
Authorization: bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiaWF0IjoxNjc5MzA5M
Tk4fQ.wDkKdq39e7v6QzGo2TWBspjWA73iuXef3pI13mBW4do

###
POST http://localhost:3000/auth/login
Content-Type:application/json

{
  "email": "rishavinmngo@gmail.com",
  "password": "123456789"
}

```

Figure 5.3 Http file for testing users login and get api point


```

POST http://localhost:3000/auth/register
Content-Type:application/json

{
  "displayName": "vikram raj",
  "email": "vikram@gmail.com",
  "password": "123456789",
  "organisation_id": 3,
  "is_organisation": false
}

###

GET http://localhost:3000/users/getAllDoctors
Authorization: bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiaWF0IjoxNjc5MzA5MTk4fQ.wDkKdq39e7v6QzGo2TWBspjWA73iuXef3pI13mBW4do

```

Figure 5.4 Http file for testing getAllDoctors and register user

```

GET http://localhost:3000/department/getAll
Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

POST http://localhost:3000/department/add
Authorization: bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywiaWF0IjoxNjc5MzA5MTk4fQ.wDkKdq39e7v6QzGo2TWBspjWA73iuXef3pI13mBW4do
Content-Type:application/json
{
  "name": "department of cardiologist"
}

POST http://localhost:3000/department/add
Authorization: bearer token
Content-Type:application/json
{
  "name": "department of cardiologist"
}

```

Figure 5.5 Http file for testing department add and getAll endpoint

Chapter 6

Conclusion

6.1 Summary of Achievements

Throughout the development of the Medsphere web app, several notable achievements have been accomplished:

1. **Comprehensive Hospital Management Solution:** Medsphere provides a comprehensive solution for managing various aspects of hospital operations, including patient management, inventory management, prescription generation, appointment management, and department management. It covers essential functionalities required for efficient hospital management.
2. **Lightweight and Clean UI:** One of the key achievements of Medsphere is its lightweight nature and clean user interface. The application has been designed with a focus on simplicity and usability, ensuring that users can easily navigate and perform their tasks without any unnecessary complexity or clutter.
3. **Auto Completion Framework:** Medsphere incorporates an efficient auto completion framework, powered by debounce and PostgreSQL text search. This framework enables users to quickly search for medicines by typing a few letters, reducing the effort required to find the exact medicine name. This feature enhances the user experience and improves efficiency in medication management.
4. **Cost-Effective Solution:** Medsphere stands out in the market as a cost-effective solution compared to existing hospital management systems. By following the Unix philosophy and prioritizing essential services, Medsphere offers a streamlined and budget-friendly alternative without compromising on critical functionalities.
5. **Integration Potential:** The development approach adopted for Medsphere is aligned with creating a medical ecosystem in the future. By adhering to the Unix philosophy, Medsphere can easily integrate with other applications, allowing for the creation of a robust and interconnected medical ecosystem that enhances collaboration and expands the scope of functionality.

6. **Technology Stack:** Medsphere utilizes a state-of-the-art technology stack, incorporating modern web technologies such as ReactJS for frontend development. The use of ReactJS, along with Context API and CSS, ensures a responsive and intuitive user interface. Additional libraries like jsPDF and lodash are employed for utility functions, further enhancing the application's capabilities.
7. **Effective Routing and Component Structure:** Medsphere employs a well-structured routing system, utilizing the React Router library for managing different routes and rendering appropriate components based on the user's actions. The Routes component efficiently handles navigation and segregation of different sections of the application, ensuring a smooth user experience.
8. **Testing and Error Handling:** Medsphere incorporates a robust testing methodology, utilizing the Jest framework for testing React components and ensuring their functionality and stability. Furthermore, the backend implementation includes custom middleware handlers for error handling, such as the `programmerErrorHandler` and `operationalErrorHandler`, which provide appropriate responses and improve overall system resilience.
9. **Database Integration:** Medsphere leverages the power of PostgreSQL as the database management system. PostgreSQL's support for complex relations and efficient text search capabilities enhances the performance of Medsphere's auto completion feature. This integration enables fast and accurate search results, optimizing the user experience.

Overall, Medsphere has successfully achieved its objectives of providing a lightweight, cost-effective, and user-friendly hospital management system. Its focus on essential services, clean UI, auto completion framework, and potential for integration with other applications positions it as a competitive solution in the market. The utilization of cutting-edge technologies and effective testing methodologies ensures the reliability and scalability of the system, making Medsphere a valuable asset for hospitals and healthcare institutions.

Let's delve into the specific achievements of Medsphere in prescription generation, lightweight and clean UI, and the auto completion framework:

1. **Prescription Generation:**

Medsphere has successfully implemented a prescription generation feature, allowing healthcare professionals to create accurate and detailed prescriptions for their patients within the web app.

The prescription generation module ensures that all necessary information, such as patient details, prescribed medications, dosage instructions, and any additional notes, can be easily inputted and managed.

By providing a structured and intuitive interface for prescription generation, Medsphere simplifies the process for healthcare providers, reducing the potential for errors and ensuring the efficient management of patient medications.

2. Lightweight and Clean UI:

Medsphere has prioritized the development of a lightweight user interface (UI), focusing on simplicity and ease of use.

The UI design emphasizes clean and intuitive layouts, allowing users to navigate through different sections of the web app seamlessly.

By minimizing unnecessary visual elements and reducing clutter, Medsphere creates a visually appealing and uncluttered interface that enhances user experience and reduces cognitive load.

The lightweight nature of the UI ensures fast loading times and optimal performance, even on low-bandwidth or resource-constrained environments.

3. Auto Completion Framework:

Medsphere's auto completion framework is a significant achievement that greatly enhances the efficiency and accuracy of medication searches within the web app.

By utilizing debounce and PostgreSQL text search, the auto completion framework allows users to find the desired medicine by typing just a few letters.

This feature significantly reduces the effort required to search for medicines, eliminating the need for manual and time-consuming searches through long lists or databases.

The integration of debounce ensures that the search process is optimized by minimizing unnecessary API calls, resulting in improved performance and reduced network overhead.

Medsphere's choice of PostgreSQL as the database management system further boosts the speed and accuracy of the auto completion feature, thanks to its efficient text search capabilities.

These achievements in prescription generation, lightweight and clean UI, and the auto completion framework collectively contribute to Medsphere's effectiveness in streamlining medication management processes. The system simplifies prescription creation, provides a

visually pleasing and user-friendly interface, and enables swift and accurate medication searches, ultimately enhancing productivity and user satisfaction within healthcare settings.

6.2 Limitations

While Medsphere offers numerous advantages and innovative features, it's important to consider some of the limitations that exist within the system. These limitations include:

1. **Limited Advanced Functionality:** Medsphere, as a lightweight and cost-effective solution, may lack some of the advanced functionalities found in more comprehensive and higher-priced hospital management systems. While it covers essential services, it may not provide extensive features for specialized medical departments or complex workflows. Organizations with specific requirements or specialized needs may find certain functionalities missing or limited in Medsphere.
2. **Integration Challenges:** Although Medsphere is designed with the intention of integrating with other applications in the future, the process of integrating and expanding the medical ecosystem may pose challenges. Depending on the compatibility and availability of APIs or integration frameworks of external applications, it may require additional development efforts and resources to seamlessly connect Medsphere with other systems. Compatibility issues or limited support from third-party applications can impact the smooth integration of the medical ecosystem.
3. **Scalability Concerns:** While Medsphere is suitable for small to medium-sized hospitals or healthcare facilities, its scalability may be a limitation for larger institutions or organizations experiencing significant growth. As the user base and data volume increase, Medsphere may require additional optimization and infrastructure scaling to ensure optimal performance and responsiveness. Without proper scalability measures in place, the system may encounter performance bottlenecks or decreased efficiency when handling a large number of concurrent users or high data loads.
4. **Learning Curve for New Users:** Despite its clean and user-friendly UI, Medsphere may still have a learning curve for new users who are unfamiliar with the system. The implementation of any new software solution requires training and adjustment for staff members who may be accustomed to different interfaces or workflows. Adequate training and documentation should be provided to ensure a smooth transition and effective utilization of Medsphere's features.

5. **Limited Vendor Support:** As an independent web app, Medsphere may have limited vendor support compared to larger, enterprise-grade hospital management systems. While the development team behind Medsphere can provide support and maintenance, the availability of dedicated customer support channels or 24/7 assistance may be limited. This could potentially impact the resolution of critical issues or the prompt handling of user inquiries.
6. **Dependence on Internet Connectivity:** Medsphere heavily relies on internet connectivity to function effectively. In cases of poor or unreliable internet connections, the performance and accessibility of the system may be compromised. Offline functionality or offline data synchronization may be limited, which can impact the ability to access patient records or perform critical tasks during internet outages.

It is essential for organizations considering Medsphere to carefully evaluate these limitations and assess whether they align with their specific needs and requirements. Understanding the limitations can help organizations make informed decisions and explore potential workarounds or alternative solutions if needed.

6.3 Future Scope

The future scope of the Medsphere project includes the implementation of specific user permissions, the addition of multiple roles, an invoice feature, and a notification section. Here's an elaboration on these features:

1. **Specific User Permissions**

Medsphere can enhance its user management system by implementing specific user permissions. This allows administrators to define granular access controls and restrictions based on user roles and responsibilities.

By assigning specific permissions to different user roles, such as doctors, nurses, administrators, and support staff, Medsphere can ensure that each user has access to the appropriate functionalities and data based on their role within the healthcare organization.

User permissions can be defined at various levels, such as module-level permissions (e.g., patients management, prescription generation), record-level permissions (e.g., patient records access), or action-level permissions (e.g., create, read, update, delete actions). This ensures a fine-grained control over user actions and data access within the system.

2. Multiple Roles:

To mimic a real hospital environment, Medsphere can introduce multiple roles that align with different positions and responsibilities in a healthcare organization.

Typical roles may include doctors, nurses, administrators, pharmacists, laboratory technicians, and more. Each role can have distinct privileges, permissions, and access levels within the system.

Multiple roles enable effective role-based access control, ensuring that users have appropriate access to features, data, and functionalities required for their respective roles.

3. Invoice Feature:

The addition of an invoice feature in Medsphere allows for seamless billing and invoicing processes within the hospital management system.

This feature can integrate with the patient management module to automatically generate invoices based on services rendered, medication provided, laboratory tests conducted, or other billable items.

The invoice feature should support the inclusion of relevant details such as patient information, billing items, prices, discounts, and payment methods.

Additionally, the system should allow for tracking and managing outstanding invoices, generating reports, and facilitating payment processing.

4. Notification Section:

Medsphere can incorporate a notification section within the system to enable efficient communication between administrators and users.

This section allows administrators to send targeted notifications, alerts, or announcements to specific users or user groups within the system.

Notifications can include reminders for upcoming appointments, important announcements, policy updates, or any other relevant information.

Users can receive notifications through various channels, such as in-app notifications, email notifications, or SMS notifications, based on their preferred communication settings.

5. Authentication Using Google and Twitter:

Medsphere can expand its authentication capabilities by integrating popular social media platforms like Google and Twitter for user authentication.

This feature allows users to sign in or register using their existing Google or Twitter accounts, eliminating the need to create separate credentials for Medsphere.

By leveraging OAuth or OpenID Connect protocols, Medsphere can securely authenticate users using their Google or Twitter credentials, ensuring a streamlined and convenient login experience.

The integration with Google and Twitter authentication provides an additional layer of trust and convenience for users, as they can utilize their trusted social media accounts to access Medsphere.

6. Integration of Machine Learning:

Medsphere can explore the integration of machine learning techniques to enhance various aspects of the system.

Machine learning algorithms can be utilized to analyze patient data and generate insights for improved diagnosis, treatment recommendations, or personalized healthcare plans.

By leveraging machine learning models, Medsphere can automate certain tasks, such as identifying patterns in patient records, predicting disease progression, or suggesting optimal treatment options based on historical data.

Additionally, machine learning algorithms can be employed for data analytics and predictive modeling to optimize inventory management, anticipate patient flow, or forecast resource allocation.

Integration of machine learning can lead to more efficient and data-driven decision-making processes within Medsphere, improving patient care, resource utilization, and overall operational efficiency.

7. Plugin System:

As part of the future development of Medsphere, implementing a plugin system can greatly enhance the application's flexibility and extensibility.

A plugin system allows third-party developers or healthcare organizations to develop custom functionalities and integrate them seamlessly into Medsphere.

By defining a well-defined plugin architecture and API, Medsphere can provide developers with the necessary tools and documentation to create plugins that cater to specific needs or requirements.

Plugins can range from specialized modules for specific medical specialties to additional features such as telemedicine integration, electronic health record (EHR) interoperability, or advanced analytics.

The plugin system encourages collaboration and innovation within the healthcare community, as developers can contribute their expertise and create value-added solutions that enhance Medsphere's capabilities.

Healthcare organizations can leverage the plugin system to tailor Medsphere to their unique workflows, requirements, and preferences, fostering a customized and efficient system that aligns with their specific needs.

Furthermore, the plugin system opens up opportunities for Medsphere to establish partnerships and a thriving ecosystem of plugins and extensions, creating a vibrant marketplace for healthcare solutions.

The plugin system should include mechanisms for plugin discovery, installation, management, and versioning, ensuring seamless integration and compatibility with the core Medsphere application.

By adopting a plugin system, Medsphere becomes a platform that continuously evolves and expands its functionality, keeping up with emerging trends, technologies, and evolving healthcare requirements.

The addition of a plugin system to Medsphere empowers users, developers, and healthcare organizations to extend the application's capabilities and customize it to their unique needs. It fosters collaboration, innovation, and the sharing of expertise within the healthcare.

community, while also positioning Medsphere as a versatile and future-proof solution in the ever-evolving healthcare landscape. Furthermore, integrating machine learning capabilities enables the system to leverage advanced analytics and automation, leading to improved UX.

With the implemented plugin system in Medsphere, healthcare organizations have the ability to develop and integrate their own custom plugins. This empowers organizations to enhance the functionality of Medsphere according to their specific requirements and workflows. Here's how the plugin system enables organizations to add their own plugins:

6.4 Final remark

In conclusion, the Medsphere project is a comprehensive hospital management web application designed to streamline and enhance various aspects of medical affairs. Throughout the development process, the project has achieved significant milestones and implemented several noteworthy features.

The project's main objectives were to provide efficient patient management, inventory management, prescription generation, and appointment management. Medsphere has successfully accomplished these goals by developing intuitive and user-friendly interfaces using modern web technologies. The clean and lightweight UI ensures a seamless user experience, allowing healthcare professionals to focus on essential tasks without unnecessary complexity.

One of the standout features of Medsphere is its auto-completion framework, which allows users to quickly search for medicines by typing just a few letters. This feature greatly enhances the efficiency of prescription generation and minimizes the time spent on searching for specific medicines. By implementing a debounce mechanism and leveraging the PostgreSQL text search capabilities, Medsphere optimizes performance and reduces network overhead, resulting in a fast and responsive auto-completion functionality.

Another significant aspect of the project is its integration with other applications and the vision of creating a medical ecosystem. Medsphere follows the Unix philosophy, aiming to establish seamless integration with other applications and foster a collaborative environment within the healthcare industry. This integration potential opens up future opportunities to connect Medsphere with AI or machine learning technologies, allowing for advanced data analytics, predictive modeling, and personalized healthcare approaches.

Furthermore, the project has utilized state-of-the-art technologies to ensure robustness and scalability. Frontend development using React JS, Context API, CSS, and libraries like jsPDF and lodash has enabled the creation of dynamic and interactive interfaces. On the backend, Medsphere leverages the power of Express.js, PostgreSQL, and custom middleware handlers for error handling, ensuring a reliable and secure server-side infrastructure.

The project has also demonstrated a commitment to testing, with frontend testing conducted using Jest and backend testing implemented with the help of the Jest plugin for Neovim. This rigorous testing methodology ensures the stability and quality of the application, reducing the likelihood of bugs and errors.

While Medsphere has achieved significant accomplishments, it is important to acknowledge certain limitations. The project's focus on a lightweight approach and adherence to the Unix philosophy may result in a reduced feature set compared to some existing systems in the market. However, this design choice provides flexibility for future growth and integration possibilities, allowing Medsphere to adapt and expand its functionality as needed.

Looking ahead, Medsphere has an exciting future scope. The project aims to implement specific user permissions and multiple roles to mimic real hospital environments. It also envisions the inclusion of an invoice feature to facilitate seamless billing and a notification section for effective communication between administrators and users. These additions will further enhance Medsphere's functionality and make it a comprehensive and indispensable tool for healthcare organizations.

In summary, the Medsphere project has successfully developed a web application that addresses the daily medical affairs of hospitals. With its lightweight approach, clean UI, and auto-completion framework, Medsphere simplifies essential tasks and improves efficiency. The integration potential, utilization of cutting-edge technologies, and commitment to testing ensure a solid foundation for future growth and innovation. Medsphere has the potential to revolutionize hospital management and contribute to better patient care, resource optimization, and streamlined workflows in the healthcare industry.