Unified Medical System (UMS) for India with Early Disease Outbreak Detection

Mini Project Report

Submitted by

Devadethan R

Reg. No.: AJC20MCA-I029

In Partial fulfillment for the Award of the Degree Of

MASTER OF COMPUTER APPLICATIONS

(INTEGRATED MCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with 'A' grade. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**Unified Medical System (UMS)"** is the bona fide work of **DEVADETHAN R (Regno: AJC20MCA-I029)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2024-25.

Ms. Jetty Benjamin	**Mr. Binumon Joseph**
Internal Guide	**Coordinator**

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report **"Unified Medical System (UMS)"** is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the **Integrated Master of Computer Applications (MCA)** from **APJ Abdul Kalam Technological University**, during the academic year **2024-2025**.

Date:	**DEVADETHAN R**
KANJIRAPPALLY	**Reg: AJC20MCA-I029**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr. Binumon Joseph (Assistant Professor)** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Jetty Benjamin (Assistant Professor)** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

DEVADETHAN R

ABSTRACT

The Unified Medical System (UMS) is a comprehensive digital healthcare platform designed to revolutionize India's healthcare landscape. By integrating patient portals, provider portals, a centralized data platform, and an AI-powered chatbot, UMS aims to improve patient access, streamline provider interactions, and enhance public health monitoring. A key feature of UMS is its ability to detect potential disease outbreaks early through machine learning analysis of patient symptoms, enabling timely public health interventions. The platform's robust tech stack, including Jinja, DaisyUI, Bootstrap Flask, and MongoDB, ensures scalability, security, and user-friendliness. By prioritizing patient empowerment and data privacy, UMS empowers patients to manage their health information securely while facilitating seamless data exchange between healthcare providers. With its focus on accessible, informed, and proactive healthcare, UMS is poised to strengthen India's healthcare infrastructure and drive positive health outcomes for its diverse population.

CONTENT

List of Abbreviation

- **UMS** - Unified Medical System
- **ABDM** - Ayushman Bharat Digital Mission
- **AI** - Artificial Intelligence
- **API** - Application Programming Interface
- **EMR** - Electronic Medical Records
- **EHR** - Electronic Health Records
- **HL7** - Health Level 7 (Interoperability Standard for Health Information)
- **FHIR** - Fast Healthcare Interoperability Resources
- **GDPR** - General Data Protection Regulation
- **PHI** - Protected Health Information
- **ROI** - Return on Investment

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Unified Medical System (UMS) for India is a digital healthcare platform that seeks to streamline access to medical services, empower patients, and enhance public health surveillance. Designed to support efficient healthcare delivery across India, UMS integrates essential functions like appointment booking, medical record management, and a patient support chatbot, creating a cohesive system for patients, healthcare providers, and public health authorities. The platform uses machine learning to detect potential disease outbreaks by analyzing real-time symptom data from patients across regions, allowing early intervention and control measures. This unified approach aims to strengthen healthcare accessibility, patient education, and data-driven disease monitoring.

1.2 PROJECT SPECIFICATION

The UMS platform is built on a robust tech stack that includes bootstrap for web applications, Flutter for mobile apps, Flask for backend services, and MongoDB for secure data storage. Key modules include:

- Patient Portal: Patients can register, book appointments, access medical records, log symptoms for disease surveillance, and receive guidance from an AI-powered chatbot.
- Healthcare Provider Portal: Enables providers to securely access patient records (with consent), manage appointments, and share data within the system for streamlined patient care.
- Unified Health Data Platform: Standardizes and securely stores medical data, supporting efficient data sharing between users with strict privacy protocols.
- Disease Outbreak Detection Module: Utilizes machine learning for real-time analysis of patient symptoms, allowing the system to detect and alert health authorities to potential disease outbreaks early.
- Patient Support Chatbot: Assists patients by answering frequently asked medical questions, guiding them through the system, and providing self-care information.

Unique Features: The UMS platform stands out with its early disease outbreak detection capability, seamless health data exchange among healthcare providers, and patient-centered design that empowers individuals to manage their health information. The AI-powered chatbot further enhances patient accessibility by providing guidance and support within the platform, contributing to a comprehensive healthcare experience in India.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The healthcare landscape in India faces numerous challenges, including fragmented service delivery, inadequate access to medical information, and the pressing need for timely disease outbreak detection. In response to these challenges, the Unified Medical System (UMS) emerges as an innovative solution that aims to revolutionize healthcare accessibility and enhance public health outcomes across the country. This digital healthcare platform integrates essential services such as appointment scheduling, secure medical record management, and a patient support chatbot, creating a cohesive ecosystem for patients, healthcare providers, and public health authorities.

UMS leverages cutting-edge technology, including machine learning algorithms, to analyze patient-reported symptoms in real time, enabling early identification of potential disease outbreaks. This proactive approach not only facilitates timely interventions but also strengthens the overall public health infrastructure. By centralizing health data and promoting seamless communication among users, UMS empowers patients to take control of their health information while ensuring healthcare providers can deliver coordinated care.

Built on a scalable tech stack, UMS is designed to be user-friendly and compliant with data privacy regulations, ensuring that sensitive health information is securely managed. By fostering a patient-centric environment and enhancing disease surveillance capabilities, the Unified Medical System aims to transform healthcare delivery in India, ultimately contributing to improved health outcomes and a more resilient healthcare system.

2.2 LITERATURE REVIEW

The RoBERTa Model for Question-Answering in Outbreak Detection

The RoBERTa model, developed by Facebook AI, is a robust transformer-based model optimized for natural language understanding tasks. Unlike its predecessor BERT, RoBERTa is designed to improve language comprehension by refining the pretraining process, which makes it well-suited for complex contextual understanding.

Why RoBERTa for Outbreak Detection?

The RoBERTa (Robustly optimized BERT approach) model, developed by Facebook AI, is an NLP transformer-based model that refines language understanding by improving BERT’s pretraining methodology [7]. RoBERTa’s enhanced ability to handle complex questions and interpret nuanced context makes it particularly effective for question-answering (QA)

Technical Overview of RoBERTa in Outbreak Detection

RoBERTa operates on the Hugging Face library, with configurations for outbreak-related QA. By referencing a database of patient records and symptom logs, RoBERTa can quickly answer outbreak-specific questions by parsing through large amounts of text. This enables healthcare officials to get real-time responses, assisting in rapid outbreak monitoring. This model enables quick response generation to specific queries, allowing public health officials to monitor and respond to suspected outbreaks efficiently

Geospatial Visualization in Outbreak Detection Systems

Geospatial visualization allows health authorities to visually interpret outbreak data, identifying patterns and geographic concentrations. Integrating geospatial tools like Folium enhances the detection system by mapping outbreak locations, which can be critical for understanding the spread and directing resources efficiently.

Implementation of Folium for Visualization

Using Folium, outbreak detection systems can generate visual maps of reported cases, highlighting disease hotspots. This is achieved by extracting location data from text analysis and representing it visually to help health officials identify and address areas with high disease prevalence effectively. This map generation provides an immediate visualization of disease clusters, which can be critical in fast-tracking containment efforts.

2.3 PROPOSED SYSTEM

The proposed Unified Medical System (UMS) is designed to address the fragmentation and inefficiencies in India’s current healthcare system by providing a centralized, patient-centric digital platform. UMS allows patients to manage their health profiles, book appointments, and securely access their medical records, empowering them to make informed health decisions. Healthcare providers can securely access and share patient information (with consent), ensuring continuity of care and fostering collaborative treatment. A key innovation is the integration of machine learning for real-time analysis of patient symptoms across regions, enabling early detection of disease outbreaks and timely alerts to public health authorities. Additionally, an AI-powered chatbot offers patients basic health guidance, directs them to relevant services, and assists with appointment scheduling, enhancing user engagement and accessibility. Built with robust security and data privacy protocols, UMS ensures a seamless, coordinated healthcare experience that improves patient outcomes and strengthens public health resilience in India.

2.3 ADVANTAGES OF PROPOSED SYSTEM

- Integrated and streamlined care.
- Enhanced patient empowerment.
- Early disease outbreak detection.
- Improved accessibility and support.
- Secure data sharing and privacy compliance.
- Efficient resource utilization.
- Scalable and flexible architecture.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The feasibility study for the Unified Medical System (UMS) assesses the technical, operational, and economic viability of developing a centralized healthcare platform for India. It examines India’s current healthcare infrastructure, data security regulations, and challenges in existing systems, alongside the potential acceptance by stakeholders. Through interviews, data collection, and research analysis, the study identifies the strengths and challenges associated with implementing the UMS. Technical feasibility is supported by growing internet infrastructure and cloud technology; operational feasibility relies on user training and system integration; and economic feasibility indicates long-term benefits, including reduced healthcare costs and improved outcomes. This assessment highlights the UMS’s potential, with careful planning and phased implementation recommended.

3.1.1 Economical Feasibility

The economic feasibility of the Unified Medical System (UMS) focuses on its potential to deliver long-term financial benefits that outweigh the costs of development, implementation, and maintenance. By streamlining healthcare processes, reducing administrative burdens, and enabling early disease detection, the UMS promises cost savings for healthcare providers and improved outcomes for patients. A sustainable funding model could be established through a mix of public and private investments, with possible user fees for advanced services. While initial costs may be significant, the anticipated return on investment (ROI) from enhanced healthcare efficiency and better resource allocation makes the UMS economically promising.

3.1.2 Technical Feasibility

The technical feasibility of the Unified Medical System (UMS) is supported by India’s growing internet infrastructure, especially in urban areas, and scalable cloud computing options for data storage and access. Implementing strong data security protocols, such as encryption, and aligning with data privacy standards ensures patient data protection. Integrating UMS with existing healthcare systems using standardized formats (e.g., HL7 FHIR) promotes interoperability.
Key components include:

- Infrastructure Availability: Urban internet connectivity supports UMS; phased rollout can address rural access challenges.
- Cloud and Data Storage: Scalable cloud solutions enable secure and accessible data storage.

- Data Security: Encryption and compliance with privacy regulations ensure patient data protection.
- System Interoperability: Standardized formats (e.g., HL7 FHIR) allow seamless data exchange with existing systems.
- Skilled Development Team: Expertise in healthcare IT, cybersecurity, and cloud infrastructure is essential.
- Scalability: Modular architecture (Vue.js, Flask, MongoDB) supports system growth and high user traffic.

3.1.3 Behavioral Feasibility

The behavioral feasibility of the Unified Medical System (UMS) examines the willingness and ability of users—including patients, healthcare providers, and administrative staff—to adopt and effectively utilize the new platform. Key considerations include:

- User Acceptance: The success of UMS largely depends on its acceptance by stakeholders. Patients and healthcare providers must recognize the system's benefits, such as improved access to medical records, streamlined appointment scheduling, and enhanced communication.
- Training and Support: Comprehensive training programs will be crucial to ensure users are comfortable with the new system. Ongoing support and resources can help users navigate the platform effectively, reducing resistance to change.
- Usability and Interface Design: A user-friendly interface tailored to diverse user groups (patients, doctors, hospital staff) will facilitate easy navigation and engagement. Positive user experiences are critical for long-term adoption.
- Cultural Factors: Understanding cultural attitudes towards technology and healthcare is vital. Tailoring communication and training to address these factors can improve acceptance and engagement.

Overall, addressing these behavioral aspects is essential for the successful implementation and sustained use of the Unified Medical System.

3.1.4 Questionnaire

Current Practices and Challenges:

1. Do you currently utilize a digital system for patient appointment booking and management? (Yes/No)

Yes, we use a basic online system, and integrate with all departments in a single hospital.

2. How often do you share patient medical records with other healthcare providers (e.g., specialists, hospitals)? (Always/Sometimes/Rarely/Never)

We share records occasionally with specialists, typically by physical records and emails, which are insecure.

3. In your experience, what are the biggest challenges related to managing patient medical records in the current system? (e.g., data fragmentation, accessibility, security)

Fragmentation is a big issue. Labs and specialists often have separate systems, making it time-consuming to get a complete picture of a patient's history.

4. How much time on average do you spend per day searching for or retrieving patient medical records? (Minutes/Hours)

On average, 5-10 minutes per patient, searching through different systems and paper charts.

Perceptions on a Unified Medical System:

5. How beneficial do you believe a national Unified Medical System (UMS) would be for improving patient care in India? (Very beneficial/Somewhat beneficial/Neutral/Not beneficial)

I believe a UMS could be very beneficial for patient care. Streamlined records and easier data sharing would improve efficiency and continuity of care.

6. What functionalities within a UMS would be most valuable to you in your daily practice? (e.g., secure data sharing, appointment scheduling, patient portal access)

Secure data sharing, online appointment scheduling, and a patient portal for accessing medical history would be most valuable.

7. Would you be comfortable using a UMS for accessing patient information from other hospitals or clinics? (Yes/No)

Yes, I would be comfortable using a UMS to access patient information from other hospitals, as long as it's secure and reliable.

Security and Privacy:

8. What security measures are most important to you regarding patient data stored within a UMS? (e.g., encryption, access control, audit logs)

Encryption, access control based on user roles, and a strong audit log to track data access are crucial.

9. How can a UMS ensure patient privacy while still facilitating data exchange for improved healthcare delivery?

A UMS should ensure privacy through strong authentication, clear patient consent for data sharing, and anonymized data for research and outbreak detection.

Looking Ahead:

10. Do you have any suggestions or specific requirements for a UMS that would be helpful in your practice?

A user-friendly interface for doctors and patients would be essential. Additionally, the system should be designed with offline functionality in case of internet disruptions.

3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor	- Intel Core i3
RAM	- 8 GB
Hard disk	- 256 GB SSD or higher

3.2.2 Software Specification

Front End	-	BOOTSTRAP v 5.2, Daisy UI
Backend	-	FLASK, Cloud Mongo
Client on PC	-	Windows 7 and above.
Technologies used	-	JS, HTML5, AJAX, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 Eg. FLASK

Flask is a lightweight and flexible web framework written in Python, designed for creating web applications quickly and with minimal overhead. It follows a modular approach, offering only essential tools and allowing developers to add extensions as needed for database connections, form handling, authentication, and more. Flask operates on the WSGI (Web Server Gateway Interface) standard and uses Jinja2 for templating, providing robust tools for rendering dynamic HTML with minimal effort.

Being “micro,” Flask is unopinionated, meaning it does not enforce any project structure or dependency requirements, giving developers control over the app’s architecture and behavior. Its simplicity and adaptability make it ideal for small to medium-sized applications, while its scalability supports more complex setups when combined with additional components. As a "micro-framework," Flask includes only the essential components for handling requests, routing, and templating, making it exceptionally lightweight and flexible. It leverages WSGI (Web Server Gateway Interface) for handling web requests and Jinja2, a fast and expressive templating engine, for creating dynamic HTML responses.

One of Flask’s standout features is its extensibility—developers can choose specific libraries or extensions for functionalities like database integration, form validation, and authentication without carrying any unnecessary dependencies. This "plug-and-play" approach allows developers to build both simple applications and scalable, production-level systems. Flask is also highly compatible with other Python libraries, making it popular for projects that include machine learning, data visualization, or complex business logic. Its straightforward syntax and clear documentation make Flask a top choice for beginners, while its flexibility and powerful extensions appeal to experienced developers who need a tailored, high-performing application stack.

3.3.2Eg. MongoDB

MongoDB is a NoSQL database known for its scalability, flexibility, and document-based structure. Unlike traditional relational databases that store data in tables with rows and columns, MongoDB stores data in BSON (Binary JSON) documents, allowing it to handle unstructured or semi-structured data efficiently. Each document is a self-contained data unit, making MongoDB well-suited for applications that require rapid data integration, real-time analytics, and frequent updates to data structure without disrupting the entire system.

MongoDB’s schema-less nature allows developers to add or modify fields easily without predefined data schemas. This flexibility makes it ideal for projects that evolve quickly or handle large volumes of diverse data types. It also supports a rich query language, indexing, aggregation, and powerful features like horizontal scaling (sharding) and replication for high availability, which enhances its performance and fault tolerance. MongoDB integrates well with modern tech stacks, making it a popular choice for full-stack developers and organizations adopting cloud-native, distributed application architectures.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to effective system. The term “design” is defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. It may be defined as a process of applying various techniques and principles for the purpose of defining a device, a process, or a system in sufficient detail to permit its physical realization. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance, and accuracy levels. The design phase is a transition from a user-oriented document to a document to the programmers or database personnel. System design goes through two phases of development: Logical and Physical Design.

4.2 UML DIAGRAM

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. UML stands for Unified Modeling Language. UML is different from the other common programming languages such as C++, Java, COBOL, etc. UML is a pictorial language used to make software blueprints. UML can be described as a general-purpose visual modeling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. After some standardization, UML has become an OMG standard. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system. The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete. UML includes the following nine diagrams.

- Class diagram

- Object diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A Use Case Diagram is a visual representation of a system's functionality, highlighting the interactions between users (actors) and the system itself. It serves as a high-level overview, showing how users achieve their goals by utilizing the system's various functions, represented as "use cases." These diagrams are commonly used in the early stages of software design to communicate system requirements, focusing on what the system does rather than how it does it.

In a Use Case Diagram:

- Actors represent the users or external systems that interact with the system, such as admins, customers, or third-party services.
- Use Cases represent specific functionalities or actions the system performs, such as "Login," "Register," "Purchase Product," or "View Orders."
- Associations are the lines connecting actors and use cases, indicating interactions or relationships.
- System Boundary is a rectangle that encapsulates all the use cases, defining the scope of the system.

Use Case Diagrams are useful for understanding user requirements, identifying main functionalities, and defining roles within a system. They also serve as a basis for further detailed analysis and design, helping ensure the system aligns with user needs.

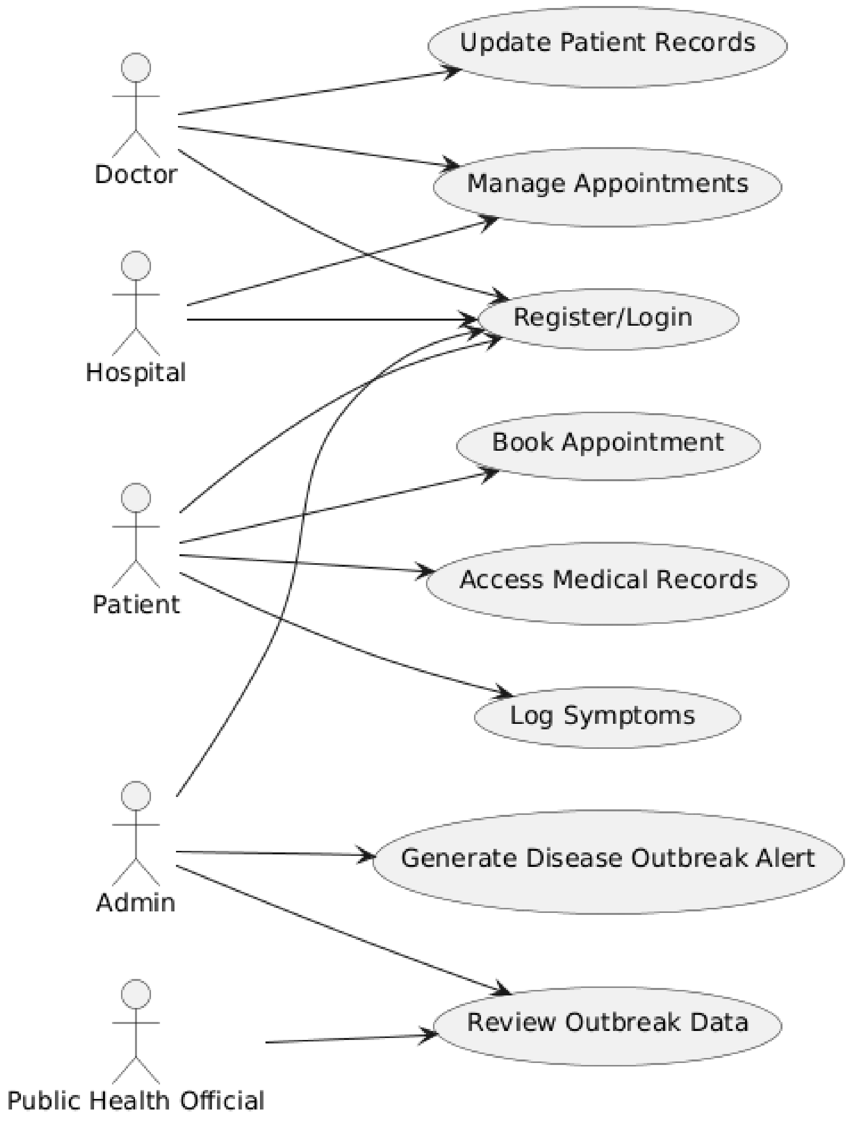


Fig 1: Use case diagram

4.2.1 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram in Unified Modeling Language (UML) that shows how objects interact in a particular sequence within a system. It visually represents the flow of messages or events between objects over time, illustrating how and in what order operations are carried out.

Key components of a sequence diagram include:

Actors: Represent users or external systems interacting with the system.

Objects or Classes: Represent entities within the system that perform actions.

Lifelines: Vertical lines showing the life span of an object during the sequence.

Activation Bars: Rectangles on the lifelines indicating the period when an object is active.

Messages: Arrows between objects indicating communication; they can be synchronous or asynchronous.

Return Messages: Dashed lines that show the return or response after a message is processed.

Use and Advantages:

Sequence diagrams are integral to modelling the dynamic aspects of systems and are especially useful during the design phase for:

- Clear Communication: They provide a visual, step-by-step breakdown of processes, improving communication among team members.
- Requirement Validation: Diagrams help validate requirements by demonstrating how processes should flow and helping stakeholders confirm intended functionality.
- System Design and Debugging: Developers use sequence diagrams to refine system architecture, identify bottlenecks, and troubleshoot issues by understanding exact message flow.

Sequence diagrams support collaboration by showing the workflow in a way that is easy to understand and highly detailed. They help developers, stakeholders, and testers confirm requirements and understand how different components should interact, assisting in debugging, performance optimization, and overall design clarity.

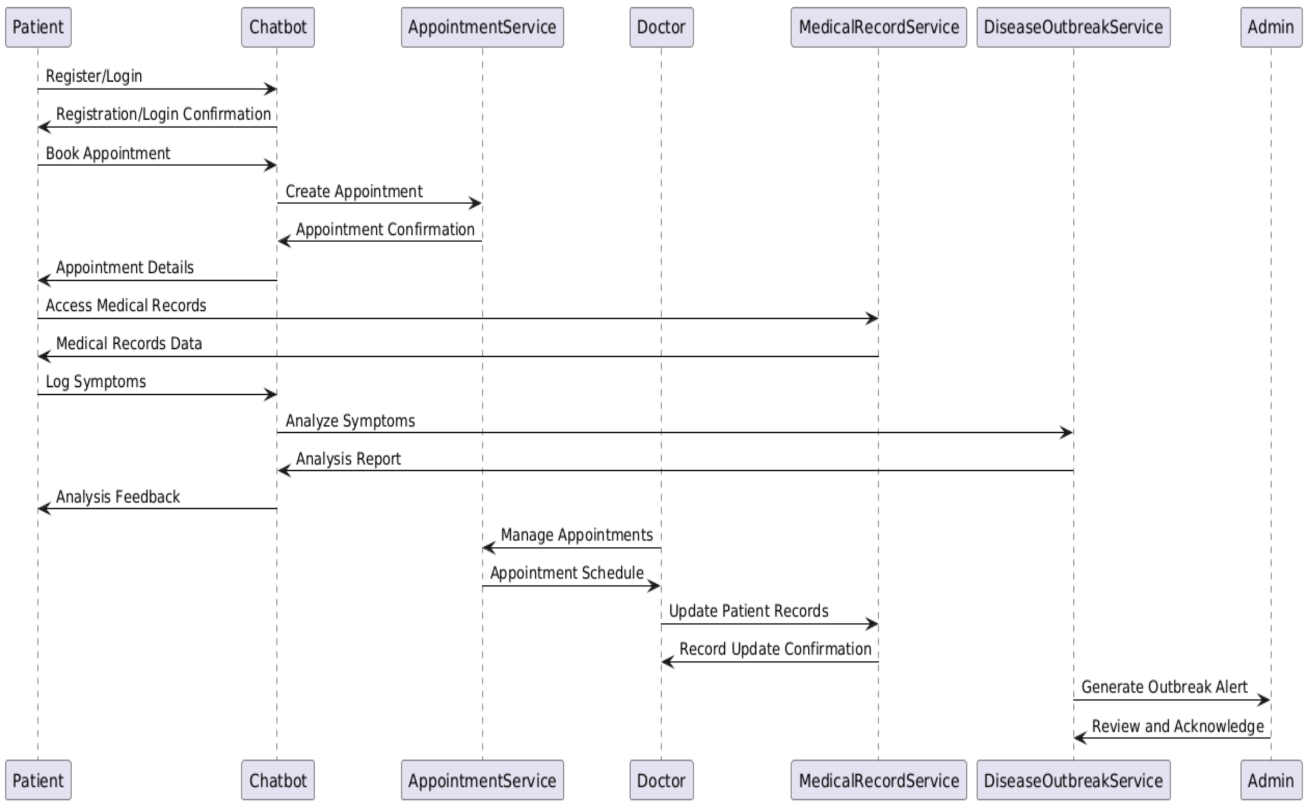


Fig 2: Sequence Diagram

4.2.2 State Chart Diagram

A state chart diagram, or state diagram, illustrates the various states an object or system component can undergo throughout its lifecycle, along with the transitions between those states triggered by events. Each state is represented as a rounded rectangle, while transitions are shown as arrows connecting the states, labeled with the events that cause the transitions.

Key Elements:

States: Conditions an object can be in (e.g., Idle, Processing).

Transitions: Arrows indicating changes between states, triggered by events.

Events and Actions: Triggers for state changes and activities that occur during transitions.

Entry and Exit Actions: Actions taken when entering or leaving a state.

Composite States: Hierarchical states that contain sub-states for complex behaviors.

Use and Benefits:

State chart diagrams model dynamic behavior, simplify complex systems, and validate that all expected states and transitions are covered. They are commonly used in applications with state-dependent behavior, such as user interfaces and workflow management systems, providing a clear view of how an object responds to various events.

4.2.2 Activity Diagram

An activity diagram is a behavioral diagram in Unified Modeling Language (UML) that illustrates the flow of activities or actions within a system, highlighting the dynamic aspects of processes. It features key elements such as activities, represented by rounded rectangles, which indicate tasks performed within the system. The diagram begins with a start node, depicted as a filled circle, and concludes with an end node, a circle encased within another circle, marking the process's completion. Transitions, represented by arrows, connect activities to demonstrate the flow from one task to another. Decision nodes, illustrated as diamonds, indicate points where the flow can diverge based on specific conditions, while forks and joins (depicted as bars) represent parallel activities. Activity diagrams are particularly valuable for modeling complex workflows, use case scenarios, and business processes, as they clarify the sequence of operations, identify potential bottlenecks, and facilitate communication among stakeholders by visualizing activities and their interdependencies.

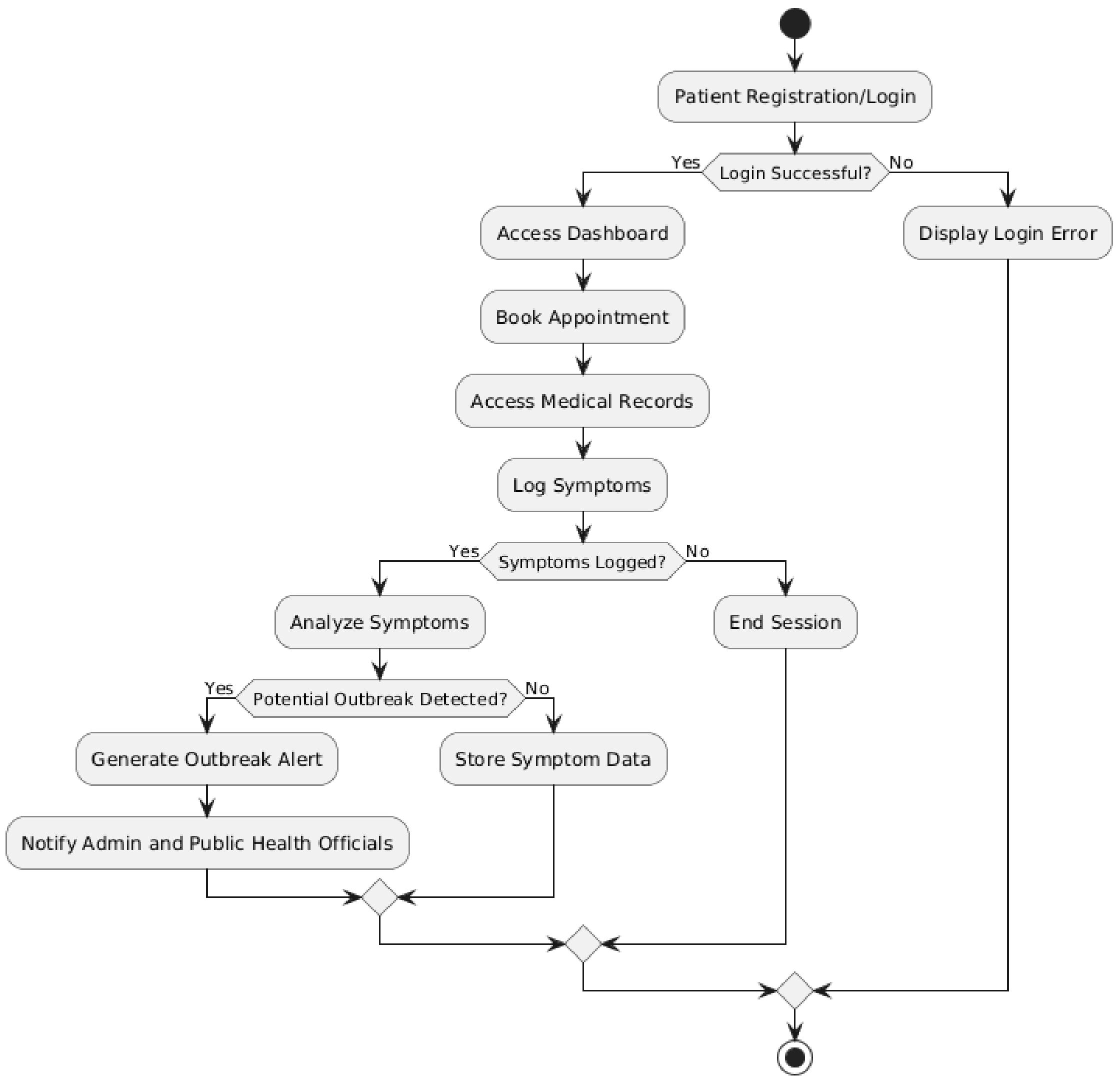


Fig 3: Activity Diagram

4.2.3 Class Diagram

A class diagram is a static structure diagram in Unified Modeling Language (UML) that represents the system's classes, their attributes, methods, and the relationships among them. It provides a visual blueprint of the system's architecture, highlighting how different classes interact and how data is organized. Each class is depicted as a rectangle divided into three sections: the top section contains the class name, the middle section lists its attributes, and the bottom section shows its methods (or operations). Relationships between classes are illustrated through lines, with various notations to indicate the type of relationship, such as inheritance (generalization), associations, aggregations, and compositions. Class diagrams are essential for object-oriented design, as they help in understanding the system's structure, guiding the implementation of classes, and ensuring that all components interact correctly. They also serve as a communication tool among developers, designers, and stakeholders by providing a clear representation of the system's data model and its relationships.

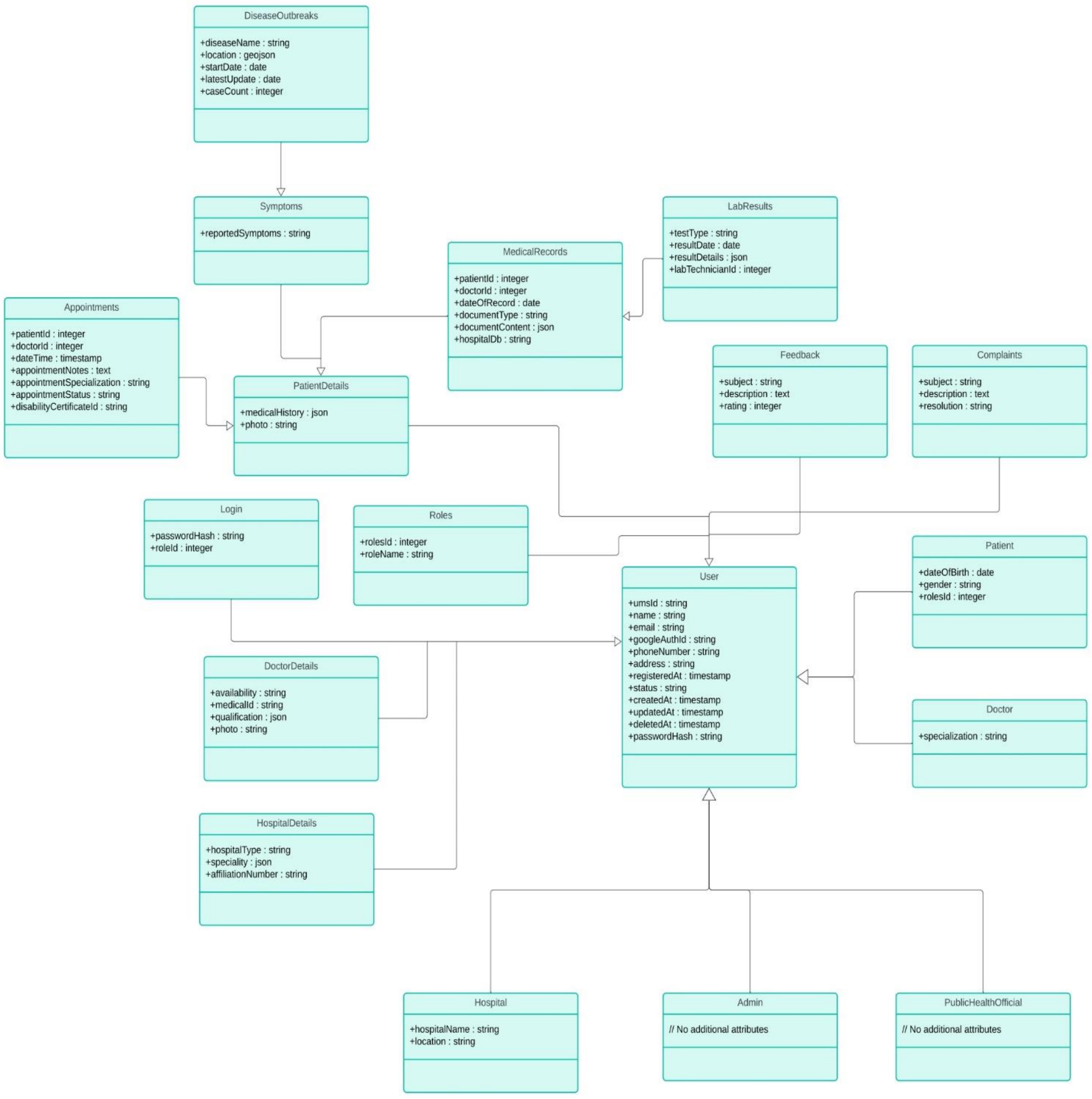


Fig 4: Class Diagram

4.2.4 Object Diagram

An object diagram is a type of static structure diagram in Unified Modeling Language (UML) that provides a snapshot of the instances of classes (objects) and their relationships at a specific moment in time. Unlike class diagrams, which focus on the blueprint of classes and their relationships, object diagrams emphasize the concrete examples of those classes in a particular state.

Each object is represented by a rectangle, similar to a class in a class diagram, but it includes the object's name and its current state or attribute values. Lines connecting the objects illustrate their relationships, such as associations, aggregations, or compositions. Object diagrams are particularly useful for visualizing complex relationships and interactions among objects in a system, demonstrating how they collaborate to perform tasks or represent data.

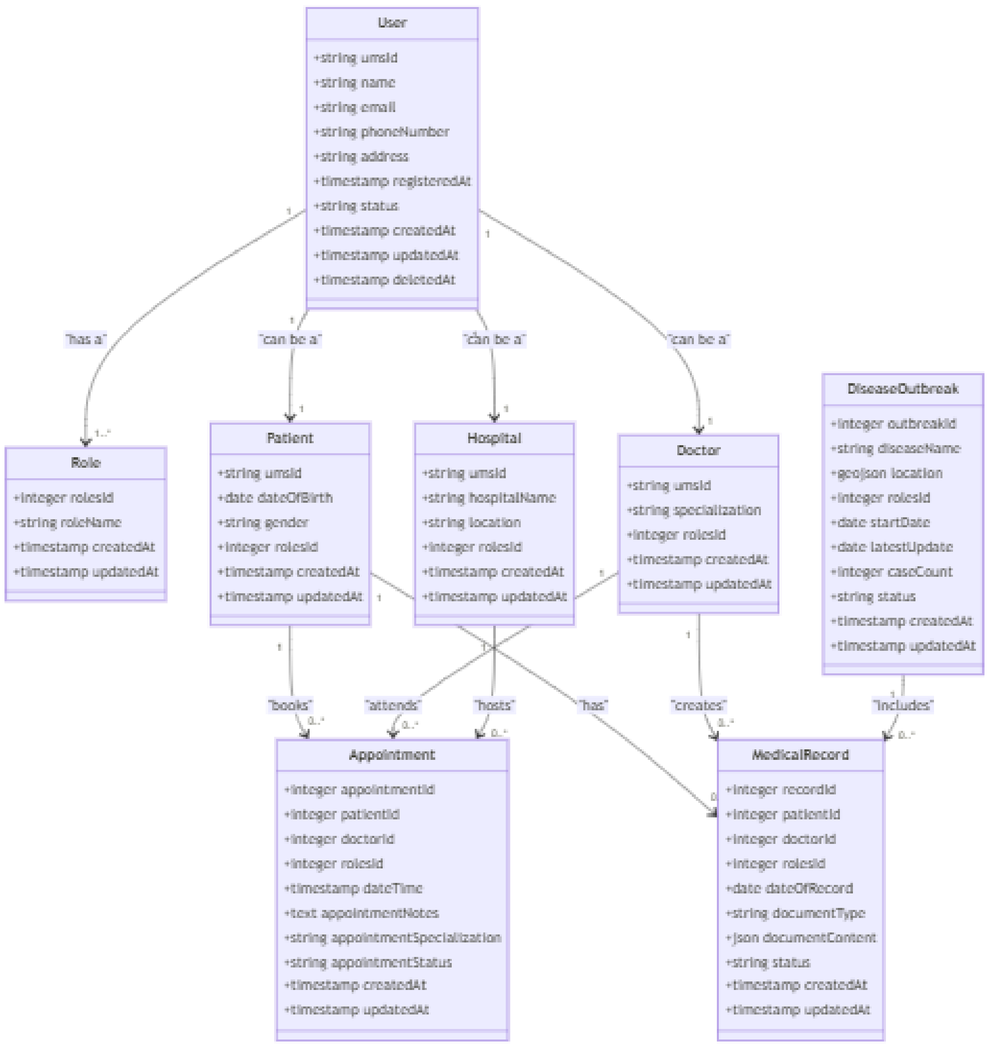


Fig 5: Object Diagram

4.2.5 Component Diagram

A component diagram is a type of structural diagram in Unified Modeling Language (UML) that represents the physical components in a system, such as software applications, libraries, and packages. It illustrates how these components interact with one another and their dependencies, providing a high-level view of the system's architecture.

Components are depicted as rectangles with two smaller rectangles on the left side, indicating that they can be independently developed and deployed. They may also include interfaces that define the operations available for other components to use. Component diagrams are particularly useful for modeling complex systems, as they help to visualize the relationships between various software components, their functionalities, and how they fit into the overall system architecture. This aids in understanding the system's modular structure, facilitates better planning for system integration, and enhances communication between developers.

4.2.8 Deployment Diagram

A deployment diagram is another type of UML diagram that shows the physical deployment of artifacts on nodes. It illustrates how software is distributed across hardware and how different components communicate within the system infrastructure. Deployment diagrams are particularly useful in visualizing the physical arrangement of software components in a distributed system, such as client-server architectures or cloud-based applications.

In a deployment diagram, nodes represent physical devices (like servers, computers, or mobile devices), and artifacts (such as executables, libraries, or database schemas) represent the software deployed on those nodes. Connections between nodes depict communication paths, showing how information flows within the system.

4.2.9 Collaboration Diagram

A collaboration diagram, or communication diagram, is a type of UML diagram that illustrates the interactions between objects in a system. It emphasizes the relationships and associations among objects while depicting the messages exchanged between them. Each object is represented as a rectangle, and lines connect these objects to indicate their relationships.

Messages are numbered to show the sequence of interactions, providing a clear view of how different components work together to achieve a specific task. Collaboration diagrams help visualize dynamic behavior, clarify roles, and enhance communication among team members during development, making them a valuable tool in system design.

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Login

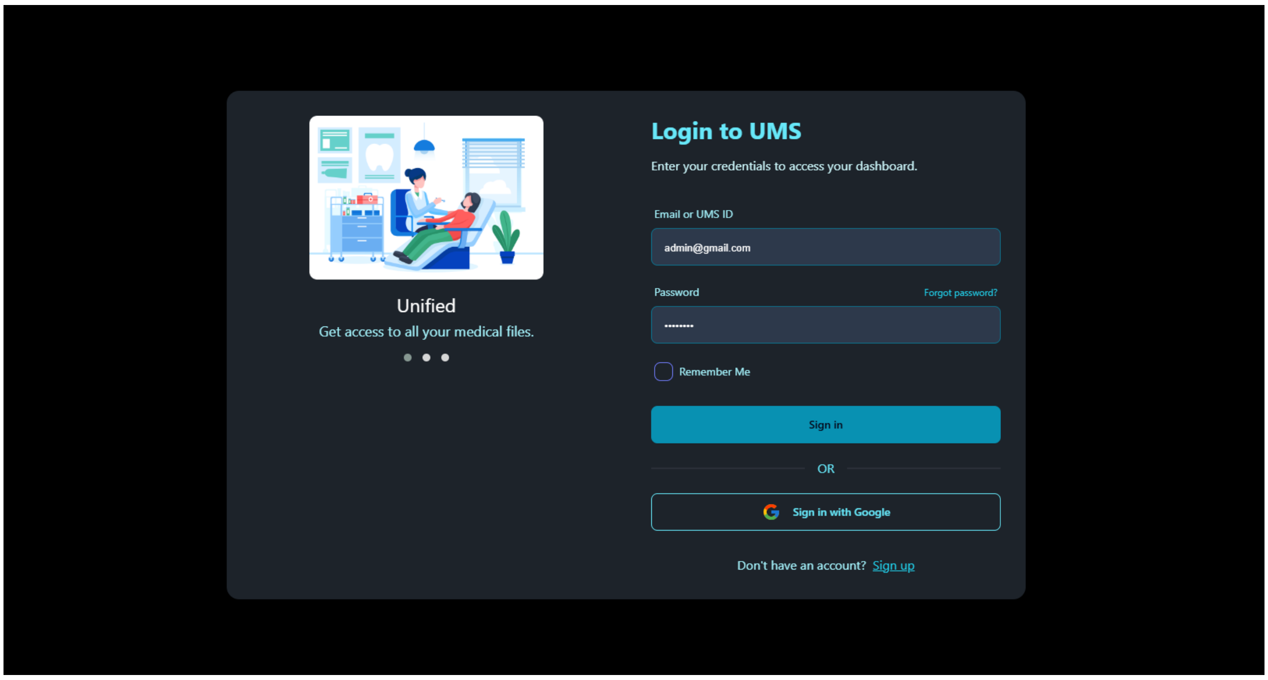

Unified
Get access to all your medical files.
Login to UMS
Enter your credentials to access your dashboard.
Email or UMS ID
admin@gmail.com
Password
Forgot password?
Remember Me
Sign in
OR
Sign in with Google
Don't have an account? Sign up

Fig 6: Figma Login Design

Form Name: Register

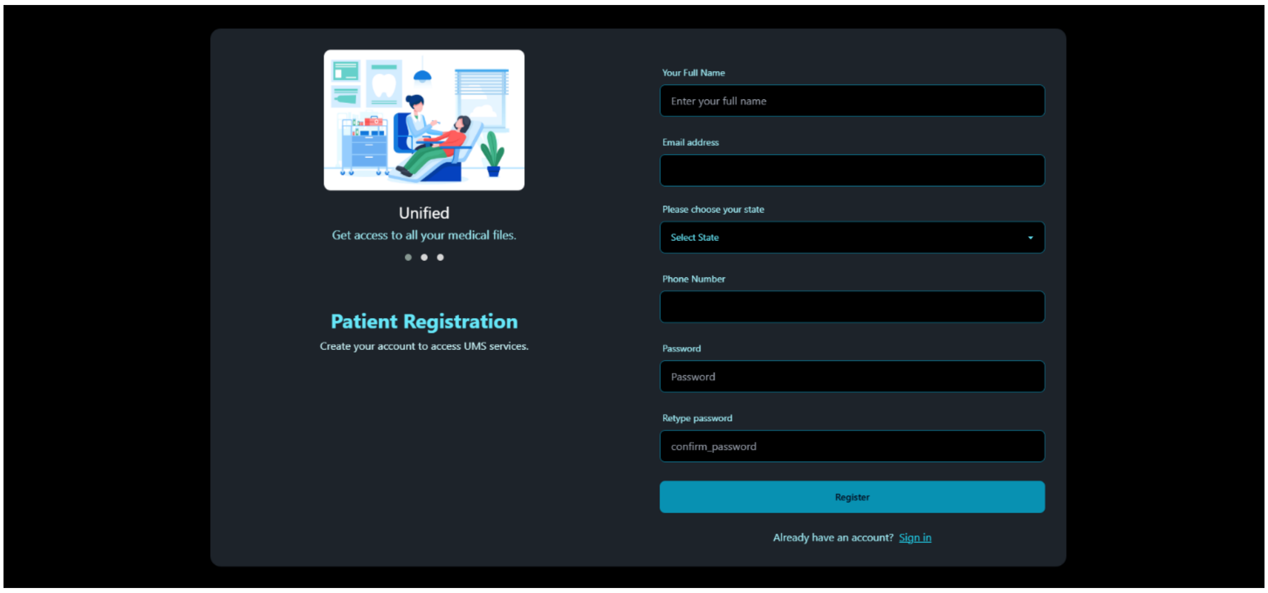


Fig 7: Figma Register Design

Form Name: Admin Home Page

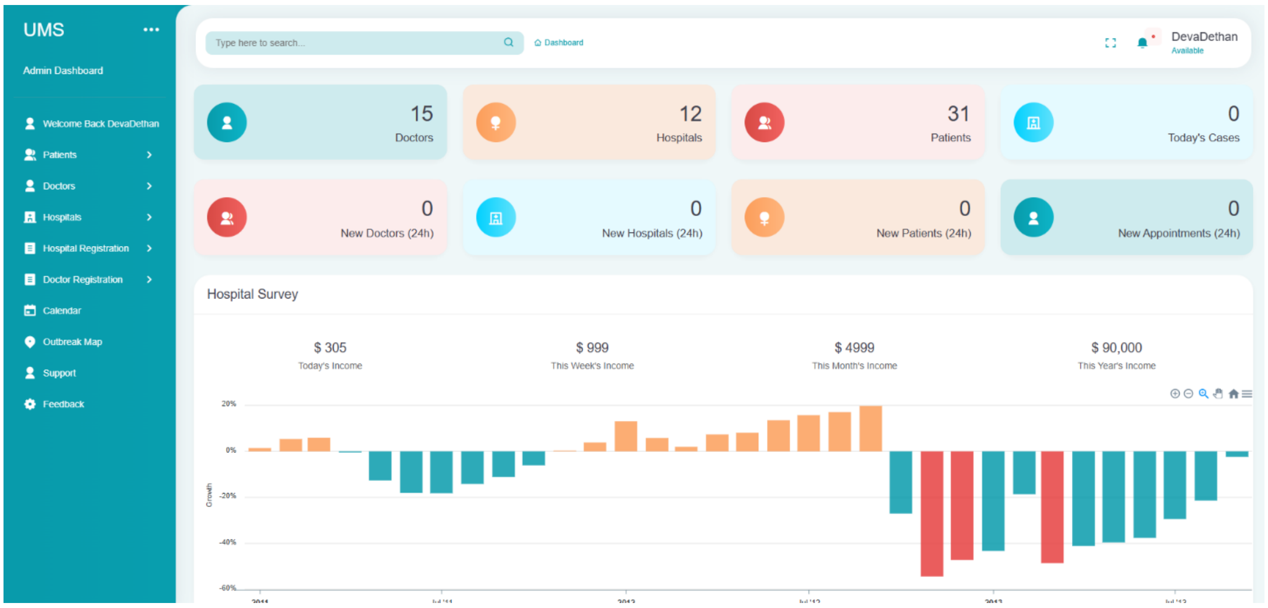


Fig 8: Figma Dashboard Design

4.4 DATABASE DESIGN

4.4.1 Non - Relational Database Management System (RDBMS)

In a non-relational database (NoSQL), data is organized in collections and documents rather than tables and rows, allowing for flexible schemas where each document in a collection can have a unique structure. Documents, similar to objects, contain fields that can store diverse data types, including arrays or nested documents, enabling rich and hierarchical data structures. Relationships are managed by embedding related data directly within a document or referencing other documents by ID, rather than enforcing strict referential integrity.

Relationships

- Table relationships are established using Key. The two main keys of prime importance are Primary Key & Foreign Key. Entity Integrity and Referential Integrity Relationships can be established with these keys.
- Entity Integrity enforces that no Primary Key can have null values.
- Referential Integrity enforces that no Primary Key can have null values.
- Referential Integrity for each distinct Foreign Key value, there must exist a matching Primary Key value in the same domain. Other key is Super Key and Candidate Keys

4.4.2 Normalization

Data are grouped together in the simplest way so that later changes can be made with minimum impact on data structures. Normalization is formal process of data structures in manners that eliminates redundancy and promotes integrity. Normalization is a technique of separating redundant fields and breaking up a large table into a smaller one. It is also used to avoid insertion, deletion, and updating anomalies. Normal form in data modelling use two concepts, keys, and relationships. A key uniquely identifies a row in a table. There are two types of keys, primary key and foreign key. A primary key is an element or a combination of elements in a table whose purpose is to identify records from the same table. A foreign key is a column in a table that uniquely identifies record from a different table. All the tables have been normalized up to the third normal form. As the name implies, it denotes putting things in the normal form. The application developer via normalization tries to achieve a sensible organization of data into proper tables and columns and where names can be easily correlated to the data by the user. Normalization eliminates repeating groups at data and thereby avoids data redundancy which proves to be a great burden on the computer resources. These include:

✓ Normalize the data

- ✓ Choose proper names for the tables and columns.
- ✓ Choose the proper name for the data.

First Normal Form

The First Normal Form states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. In other words, 1NF disallows “relations within relations” or “relations as attribute values within tuples”. The only attribute values permitted by 1NF are single atomic or indivisible values. The first step is to put the data into First Normal Form. This can be donor by moving data into separate tables where the data is of similar type in each table. Each table is given a Primary Key or Foreign Key as per the requirement of the project. In this we form new relations for each non-atomic attribute or nested relation. This eliminated repeating groups of data.

Second Normal Form

According to Second Normal Form, for relations where primary key contains multiple attributes, no non-key attribute should be functionally dependent on a part of the primary key. In this we decompose and set up a new relation for each partial key with its dependent attributes. Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it. This step helps in taking out data that is only dependent on a part of the key. A relation is said to be in second normal form if and only if it satisfies all the first normal form conditions for the primary key and every non-primary key attribute of the relation is fully dependent on its primary key alone.

Third Normal Form

According to Third Normal Form, Relation should not have a non-key attribute functionally determined by another non-key attribute or by a set of non-key attributes. That is, there should be no transitive dependency on the primary key. In this we decompose and set up relations that includes the non-key attributes that functionally determine other non-key attributes. This step is taken to get rid of anything that does not depend entirely on the Primary Key. A relation is said to be in third normal form if only if it is in second normal form and moreover the non key attributes of the relation should not be dependent on another non-key attribute.

Fourth Normal Form

The fourth normal form (4NF) is a database normalization rule that further refines data modeling by addressing multi-valued dependencies. When a table contains multiple independent sets of repeating data, we can break it down into smaller tables, with each table containing only one set of related data.

This reduces data redundancy and improves data consistency by ensuring that each table represents a single, well-defined concept or entity. To achieve 4NF, we need to ensure that all multi-valued dependencies are removed from the table, and that each table contains only attributes that are functionally dependent on the primary key.

Fifth Normal Form

5NF is indeed the highest level of normalization in relational database design, and it deals with complex data models that involve multiple overlapping multi-valued dependencies. In 5NF, tables are decomposed into smaller tables in order to eliminate any possible redundancy caused by overlapping dependencies, while ensuring that there is no loss of data.

The goal of 5NF is to ensure that each table represents a single entity or relationship, and that the data is organized in a way that minimizes redundancy, eliminates anomalies, and improves data integrity. By breaking down tables into smaller, more specialized tables, 5NF helps to eliminate potential issues with update anomalies, insertion anomalies, and deletion anomalies, which can occur when data is not properly normalized.

4.4.3 Sanitization

An automated procedure called "sanitization" is used to get a value ready for use in a SQL query. This process typically involves checking the value for characters that have a special significance for the target database. To prevent a SQL injection attack, you must sanitize(filter) the input string while processing a SQL query based on user input. For instance, the user and password input is a typical scenario. In that scenario, the server response would provide access to the 'target user' account without requiring a password check.

4.4.4 Indexing

By reducing the number of disk accesses needed when a query is completed, indexing helps a database perform better. It is a data structure method used to locate and access data in a database rapidly. Several database columns are used to generate indexes. The primary key or candidate key of the table is duplicated in the first column, which is the Search key. To make it easier to find the related data, these values are kept in sorted order. Recall that the information may or may not be kept in sorted order.

4.5 TABLE DESIGN

1.Tbl_user

Eg.Primary key: umsId

Eg.Foreign key: umsId references table **Tbl_login**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	umsId	string	primary key	Alpha-numeric UUIDv4, unique identifier for the user
2.	name	string		Name of the user
3.	email	string	unique	Email address of the user
4.	googleAuthId	string		Google authentication ID
5.	phoneNumber	string		Phone number of the user
6.	address	string		Physical address of the user
7.	status	string		Status of the user account
8.	passwordHash	string		Hashed and salted passwor

2.Tbl_roles

Eg.Primary key: **loginid**

Eg.Foreign key: **loginid** references table **Tbl_users_login**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	rolesId	integer	primary key	Unique identifier for the role
2.	roleName	string		Name of the role

3.Tbl_login

Eg.Primary key: umsId

Eg.Foreign key: umsId references table **Tbl_patients**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	umsId	string	primary key, foreign key (users.umsId)	Alpha-numeric identifier linking to users table

2.	passwordHash	string		Hashed and salted password
3.	email	string	unique	User's email address
4.	roleId	integer	foreign key (roles.rolesId)	Role identifier
5.	status	string		Login status

4.Tbl_Patients

Eg.Primary key: umsId

Eg.Foreign key: umsId references table **Tbl_patientsdetails**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	umsId	string	primary key, foreign key (users.umsId)	UMSG-UUIDv4 format patient identifier
2.	dateOfBirth	date		Patient's date of birth
3.	gender	string		Patient's gender
4.	rolesId	integer	foreign key (roles.rolesId)	Role identifier

5.Tbl_Patient detail

Eg.Primary key: **patientId**

Eg.Foreign key: **patientId** references table **Tbl_appointments**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	patientId	integer	primary key, foreign key (patients.umsId)	Patient identifier
2.	medicalHistory	json		Patient's medical history in JSON format
3.	photo	string		Path to patient's photo
4.	status	string		Current status
5.	patientId	integer	primary key, foreign key (patients.umsId)	Patient identifier

6.Tbl_Doctors

Eg.Primary key: **umsid**

Eg.Foreign key: **umsid** references table **Tbl_patients**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	umsId	string	primary key, foreign key (users.umsId)	UMSD-UUIDv4 format doctor identifier
2.	specialization	string		Doctor's medical specialization
3.	rolesId	integer	foreign key (roles.rolesId)	Role identifier

7.Tbl_doctor details

Eg.Primary key: **doctorid**

Eg.Foreign key: **doctorid** references table **Tbl_appointments**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	doctorId	integer	primary key, foreign key (doctors.umsId)	Doctor identifier
2.	availability	string		Doctor's availability schedule
3.	medicalId	string		Alpha-numeric medical license ID
4.	qualification	json		Doctor's qualifications in JSON format
5.	photo	string		Path to doctor's photo
6.	status	string		Current status
7.	doctorId	integer	primary key, foreign key (doctors.umsId)	Doctor identifier

8.Tbl_Hospitals

Eg.Primary key: **umsid**

Eg.Foreign key: **umsid** references table **Tbl_hospital_Details**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	umsId	string	primary key, foreign key (users.umsId)	UMSH-UUIDv4 format hospital identifier

2.	hospitalName	string		Name of the hospital
3.	location	string		Hospital location
4.	rolesId	integer	foreign key (roles.rolesId)	Role identifier

9.Tbl_hospital details

Eg.Primary key: **hospitalid**

Eg.Foreign key: **hospitalid** references table **Tbl_user**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	hospitalId	integer	primary key, foreign key (hospitals.umsId)	Hospital identifier
2.	hospitalType	string		Type of hospital
3.	speciality	json		Hospital specialities in JSON format
4.	affiliationNumber	string		Alpha-numeric affiliation number
5.	status	string		Current status

10.Tbl_appointments

Eg.Primary key: **appointmentid**

Eg.Foreign key: **loginid** references table **Tbl_users_login**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	appointmentId	integer	primary key	Unique identifier for appointment
2.	patientId	integer	foreign key (patients.umsId)	Patient identifier
3.	doctorId	integer	foreign key (doctors.umsId)	Doctor identifier
4.	rolesId	integer	foreign key (roles.rolesId)	Role identifier
5.	dateTime	timestamp		Appointment date and time
6.	appointmentNotes	text		Notes about the appointment
7.	appointmentSpecialization	string		Specialization required for appointment

8.	appointmentStatus	string		Status of the appointment
9.	disabilityCertificateId	string		Alpha-numeric disability certificate ID
10.	status	string		Current status

11.Tbl_medical records

Eg.Primary key: **recordid**

Eg.Foreign key: **recordid** references table **Tbl_appointment**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	recordId	integer	primary key	Unique identifier for medical record
2.	patientId	integer	foreign key (patients.umsId)	Patient identifier
3.	doctorId	integer	foreign key (doctors.umsId)	Doctor identifier
4.	rolesId	integer	foreign key (roles.rolesId)	Role identifier
5.	dateOfRecord	date		Date of the medical record
6.	documentType	string		Type of medical document
7.	documentContent	json		Medical record content in JSON format
8.	hospitalDb	string		Hospital database reference
9.	status	string		Current status

12.Tbl_symptoms

Eg.Primary key: **symptomid**

Eg.Foreign key: **symptomid** references table **Tbl_appointment**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	symptomId	integer	primary key	Unique identifier for symptom record
2.	patientId	integer	foreign key (patients.umsId)	Patient identifier
3.	rolesId	integer	foreign key (roles.rolesId)	Role identifier
4.	date	date		Date of symptom report

5.	reportedSymptoms	string	increment	List of reported symptoms
6.	status	string		Current status

13.Tbl_disease outbreaks

Eg.Primary key: **outbreakId**

Eg.Foreign key: **outbreakId** references table **Tbl_medical records**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1.	outbreakId	integer	primary key	Unique identifier for disease outbreak
2.	diseaseName	string		Name of the disease
3.	location	geojson		Geographic location data
4.	rolesId	integer	foreign key (roles.rolesId)	Role identifier
5.	startDate	date		Outbreak start date
6.	latestUpdate	date		Date of latest update
7.	caseCount	integer		Number of cases
8.	status	string		Current status

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

Software Testing is the process of executing software in a controlled manner, in order to answer the question - Does the software behave as specified? Software testing is often used in association with the term's verification and validation. Validation is the checking or testing of items, includes software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques such as reviews, analysis, inspections, and walkthroughs. Validation is the process of checking that what has been specified is what the user wanted.

Other activities which are often associated with software testing are static analysis and dynamic analysis. Static analysis investigates the source code of software, looking for problems and gathering metrics without executing the code. Dynamic analysis looks at the behavior of software while it is executing, to provide information such as execution traces, timing profiles, and test coverage information.

Testing is a set of activity that can be planned in advanced and conducted systematically. Testing begins at the module level and work towards the integration of entire computers-based system. Nothing is complete without testing, as it vital success of the system testing objectives, there are several rules that can serve as testing objectives. They are:

Testing is a process of executing a program with the intent of finding an error.

- A good test case is one that has high possibility of finding an undiscovered error.
- A successful test is one that uncovers an undiscovered error

If a testing is conducted successfully according to the objectives as stated above, it would uncover errors in the software. Also testing demonstrate that the software function appear to be working according to the specification, that performance requirement appear to have been met. There are three ways to test program.

- For correctness
- For implementation efficiency
- For computational complexity

Test for correctness are supposed to verify that a program does exactly what it was designed to do. This is much more difficult than it may at first appear, especially for large programs.

5.2 TEST PLAN

A test plan implies a series of desired course of action to be followed in accomplishing various testing methods. The Test Plan acts as a blue print for the action that is to be followed. The software engineers create a computer program, its documentation, and related data structures. The software developers is always responsible for testing the individual units of the programs, ensuring that each performs the function for which it was designed. There is an independent test group (ITG) which is to remove the inherent problems associated with letting the builder to test the thing that has been built. The specific objectives of testing should be stated in measurable terms. So that the mean time to failure, the cost to find and fix the defects, remaining defect density or frequency of occurrence and test work-hours per regression test all should be stated within the test plan.

The levels of testing include:

- ❖ Unit testing
- ❖ Integration Testing
- ❖ Data validation Testing
- ❖ Output Testing

5.2.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design – the software component or module. Using the component level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered scope established for unit testing. The unit testing is Whitebox oriented, and step can be conducted in parallel for multiple components. The modular interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that all statements in a module have been executed at least once. Finally, all error handling paths are tested.

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit properly, all other tests are moot. Selective testing of execution paths is an essential task during the unit test. Good design dictates that error conditions be anticipated and error handling paths set up to reroute or cleanly terminate processing when an error does occur. Boundary testing is the last task of unit testing step. Software often fails at its boundaries.

5.2.2 Integration Testing

Integration testing is systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. The entire program is tested as whole. Correction is difficult because isolation of causes is complicated by vast expanse of entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

5.2.3 Validation Testing or System Testing

This is the final step in testing. In this the entire system was tested with all forms, code, modules, and class modules. This form of testing is popularly known as Black Box testing or System tests.

Black Box testing method focuses on the functional requirements of the software. That is, Black Box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.

5.2.4 Output Testing or User Acceptance Testing

The system considered is tested for user acceptance; here it should satisfy the firm’s need. The software should keep in touch with perspective system; user at the time of developing and making changes whenever required. This done with respect to the following points:

- Input Screen Designs.
- Output Screen Designs

The above testing is done taking various kinds of test data. Preparation of test data plays a vital role in the system testing. After preparing the test data, the system under study is tested using that test data. While testing the system by which test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

5.2.5 Automation Testing

A test case suite is executed using specialized automated testing software tools as part of the software testing technique known as automation testing. The test stages are meticulously carried out by a human performing manual testing while seated in front of a computer. Additionally, the automation testing software may generate thorough test reports, compare expected and actual findings, and enter test data into the System Under Test. Software test automation necessitates significant financial and material inputs.

5.2.6 Selenium Testing

Selenium is a free and open-source tool for testing web applications across multiple browsers and operating systems. Selenium Test Scripts can be written in different programming languages, including Java, C#, JavaScript, Python, etc. Automation performed using the Selenium framework is referred to as Selenium Automation testing.

Example:

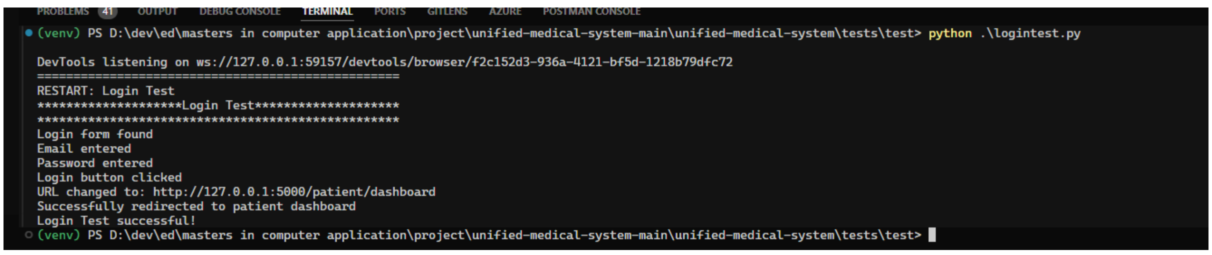
Test Case 1

Code

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException,
ElementClickInterceptedException, NoSuchElementException
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import requests
from selenium.common.exceptions import WebDriverException
import time

def setup_driver():
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service)
    return driver

def wait_for_overlay_to_disappear(driver, timeout=10):
    try:
        WebDriverWait(driver, timeout).until_not(
            EC.presence_of_element_located((By.CSS_SELECTOR, ".overlay,
.modal, .loading"))
        )
    except TimeoutException:
        print("No overlay found or it didn't disappear")

def click_element_safely(driver, element):
    try:
        element.click()
    except ElementClickInterceptedException:
        wait_for_overlay_to_disappear(driver)
        try:
            element.click()
        except ElementClickInterceptedException:
            driver.execute_script("arguments[0].click();", element)

def test_successful_login(driver):
    driver.get("http://127.0.0.1:5000/auth/login")
    # Wait for the login form to be loaded
    try:
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, "loginForm"))
        )
        print("Login form found")
    except TimeoutException:
        print("Login form not found within timeout")
        raise

    # Fill in credentials
    try:
        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "identifier"))
        )
        email_field.send_keys("janepat@gmail.com")
        print("Email entered")

        password_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "password"))
        )
        password_field.send_keys("PPpp!@12")
        print("Password entered")

        login_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH,
"//button[@type='submit']"))
        )
        click_element_safely(driver, login_button)
        print("Login button clicked")

        # Check for error messages
        try:
            error_message = driver.find_element(By.CLASS_NAME, "alert")
            print(f"Login error: {error_message.text}")
            return
        except NoSuchElementException:
            pass

        # Wait for URL change with longer timeout
        WebDriverWait(driver, 20).until(
            lambda driver: driver.current_url ==
"http://127.0.0.1:5000/patient/dashboard"
        )
        print(f"URL changed to: {driver.current_url}")

        # Verify exact URL match
        assert driver.current_url ==
"http://127.0.0.1:5000/patient/dashboard", \
            f"Unexpected URL after login: {driver.current_url}"
        print("Successfully redirected to patient dashboard")

    except Exception as e:
        print(f"Test failed: {str(e)}")
        print(f"Current URL: {driver.current_url}")
        # Take screenshot on failure
        driver.save_screenshot("login_error.png")
        raise
```

```python
def check_server_running():
    try:
        response = requests.get("http://127.0.0.1:8000/")
        return response.status_code == 200
    except requests.ConnectionError:
        return False

def main():
    driver = setup_driver()
    try:
        test_successful_login(driver)
    finally:
        driver.quit()

if __name__ == "__main__":
    main()
```

Eg. Screenshot



```
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test> python .\logintest.py

DevTools listening on ws://127.0.0.1:59157/devtools/browser/f2c152d3-936a-4121-bf5d-1218b79dfc72
==================================================
RESTART: Login Test
********************Login Test********************
**************************************************
Login form found
Email entered
Password entered
Login button clicked
URL changed to: http://127.0.0.1:5000/patient/dashboard
Successfully redirected to patient dashboard
Login Test successful!
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test>
```

Eg.Test Report

Test Case 1					
Project Name: Unified Medical System					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Devadethan R**		
Test Priority(Low/Medium/High):			**Test Designed Date: 24/10/2024**		
Module Name: Login Page			**Test Executed By: Jetty Benjamin**		
Test Title : Login page of user account			**Test Execution Date: 26/10/2024**		
Description: Login page to user					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Navigate to the login page	http://127.0.0.1:5000/auth/login	Login page should be displayed	Login page was loaded	Pass

2	Enter email	“janepat@gmail.com”	The email field should be filled with “janepat@gmail.com”	email entered	Pass
3	Enter Password	“PPpp!@12”	The password field should be filled with “PPpp!@12”	Password entered	Pass
4	Click on the submit button	N/A	If login is successful, user should be redirected to user login page	User was redirected to user login page	Pass
Post-Condition: User should be logged in and have access to patient dashboard					

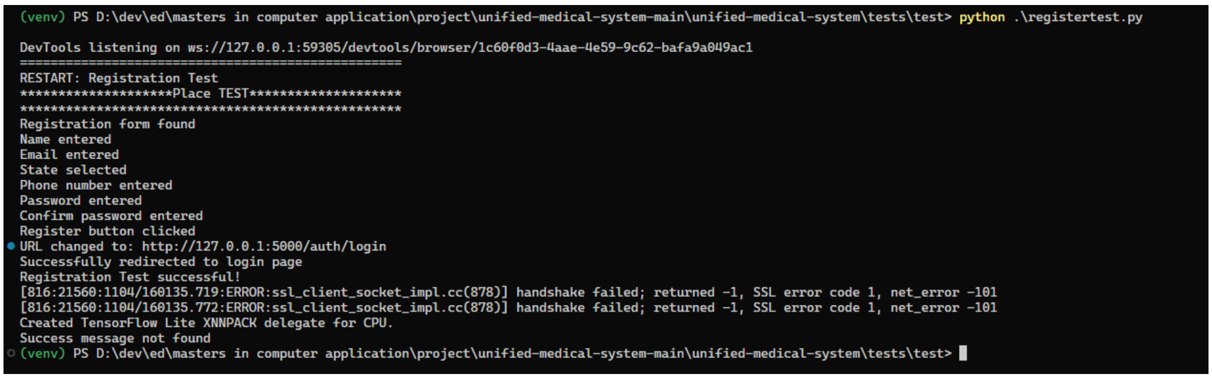
Test Case 2:

Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import TimeoutException,
ElementClickInterceptedException, NoSuchElementException
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import requests
import time
import random
import string

def setup_driver():
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service)
    return driver

def wait_for_overlay_to_disappear(driver, timeout=10):
    try:
        WebDriverWait(driver, timeout).until_not(
            EC.presence_of_element_located((By.CSS_SELECTOR, ".overlay,
.modal, .loading"))
        )
    except TimeoutException:
        print("No overlay found or it didn't disappear")

def click_element_safely(driver, element):
    try:
        element.click()
    except ElementClickInterceptedException:
        wait_for_overlay_to_disappear(driver)
        try:
            element.click()
        except ElementClickInterceptedException:
            driver.execute_script("arguments[0].click();", element)

def generate_random_email():
    # Generate a random string for the email
    random_string = ''.join(random.choices(string.ascii_lowercase, k=8))
    return f"test_{random_string}@example.com"

def print_test_header():
    print("=" * 50)
    print("RESTART: Registration Test")
    print("*" * 20 + "Place TEST" + "*" * 20)
    print("*" * 50)

def print_test_success():
    print("Registration Test successful!")

def test_successful_registration(driver):
    print_test_header()
    driver.get("http://127.0.0.1:5000/patient/register")

    # Wait for the registration form to be loaded
    try:
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, "registerForm"))
        )
        print("Registration form found")
    except TimeoutException:
        print("Registration form not found within timeout")
        raise

    try:
        # Fill in registration details
        name_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "name"))
        )
        name_field.send_keys("Test User")
        print("Name entered")

        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "email"))
        )
        email_field.send_keys(generate_random_email())
        print("Email entered")

        # Select state
        state_select = Select(WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "state"))
        ))
        state_select.select_by_value("KA")  # Selecting Karnataka as an
example
        print("State selected")

        phone_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "phonenumber"))
        )
        phone_field.send_keys("9876543210")
        print("Phone number entered")

        password_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "password"))
        )
        password_field.send_keys("Test@123")
        print("Password entered")

        confirm_password_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "confirm_password"))
        )
        confirm_password_field.send_keys("Test@123")
        print("Confirm password entered")

        # Submit the form
        register_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH,
"//button[@type='submit']"))
        )
        click_element_safely(driver, register_button)
        print("Register button clicked")

        # Check for error messages
        try:
            error_message = driver.find_element(By.CLASS_NAME, "text-
error")
            print(f"Registration error: {error_message.text}")
            return
        except NoSuchElementException:
            pass

        # Wait for redirect to login page
        WebDriverWait(driver, 20).until(
            lambda driver: driver.current_url ==
"http://127.0.0.1:5000/auth/login"
        )
        print(f"URL changed to: {driver.current_url}")

        # Verify exact URL match
        assert driver.current_url == "http://127.0.0.1:5000/auth/login", \
            f"Unexpected URL after registration: {driver.current_url}"
        print("Successfully redirected to login page")

        # After successful assertion and before the success message check
        print_test_success()

        # Check for success message
        try:
            success_message = WebDriverWait(driver, 10).until(
                EC.presence_of_element_located((By.CLASS_NAME, "success"))
            )
            print(f"Success message displayed: {success_message.text}")
        except TimeoutException:
            print("Success message not found")
```

```python
    except Exception as e:
        print(f"Test failed: {str(e)}")
        print(f"Current URL: {driver.current_url}")
        # Take screenshot on failure
        driver.save_screenshot("registration_error.png")
        raise

def check_server_running():
    try:
        response = requests.get("http://127.0.0.1:8000/")
        return response.status_code == 200
    except requests.ConnectionError:
        return False

def main():
    driver = setup_driver()
    try:
        test_successful_registration(driver)
    finally:
        driver.quit()

if __name__ == "__main__":
    main()
```

Screenshot



```
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test> python .\registertest.py

DevTools listening on ws://127.0.0.1:59305/devtools/browser/1c60f0d3-4aae-4e59-9c62-bafa9a049ac1
==================================================
RESTART: Registration Test
********************Place TEST********************
**************************************************
Registration form found
Name entered
Email entered
State selected
Phone number entered
Password entered
Confirm password entered
Register button clicked
URL changed to: http://127.0.0.1:5000/auth/login
Successfully redirected to login page
Registration Test successful!
[816:21560:1104/160135.719:ERROR:ssl_client_socket_impl.cc(878)] handshake failed; returned -1, SSL error code 1, net_error -101
[816:21560:1104/160135.772:ERROR:ssl_client_socket_impl.cc(878)] handshake failed; returned -1, SSL error code 1, net_error -101
Created TensorFlow Lite XNNPACK delegate for CPU.
Success message not found
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test>
```

Test Report:

Test Case 2	
Project Name: Unified Medical System	
Register Test Case	
Test Case ID: Test_2	**Test Designed By: Devadethan R**
Test Priority(Low/Medium/High):	**Test Designed Date: 24/10/2024**
Module Name: Patient Registration	**Test Executed By : Jetty Benjamin**
Test Title : Register page of user account	**Test Execution Date: 26/10/2024**

Description: Register page to user					
Pre-Condition: Registration system is accessible					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to registration page	http://127.0.0.1:5000/patient/registe	Registration page should be displayed	Registration page loaded	Pass
2	Enter full name	"Test User"	Name field should accept input	Name entered successfully	Pass
3	Enter email	[Random generated email]	Email field should accept input	Email entered successfully	Pass
4	Select state	"KA" (Karnataka)	State should be selectable	State selected successfully	Pass
5	Enter phone number	"9876543210"	Phone field should accept input	Phone number entered	Pass
6	Enter password	"Test@123"	Password field should accept input	Password entered	Pass
7	Confirm password	"Test@123"	Confirm password should match	Password confirmed	Pass
8	Submit registration	N/A	Should redirect to login page	Redirected to login page	Pass
Post-Condition: New user account should be created and ready for login					

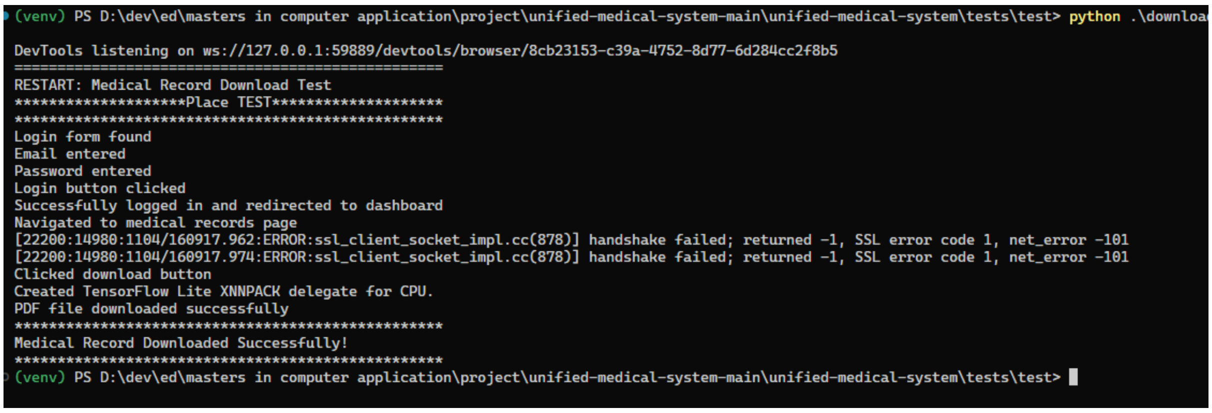
Test Case 3:

Code

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException,
ElementClickInterceptedException, NoSuchElementException
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import requests
import time
import os
```

```
def setup_driver():
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service)
    return driver

def wait_for_overlay_to_disappear(driver, timeout=10):
    try:
        WebDriverWait(driver, timeout).until_not(
            EC.presence_of_element_located((By.CSS_SELECTOR, ".overlay,
.modal, .loading"))
        )
    except TimeoutException:
        print("No overlay found or it didn't disappear")

def click_element_safely(driver, element):
    try:
        element.click()
    except ElementClickInterceptedException:
        wait_for_overlay_to_disappear(driver)
        try:
            element.click()
        except ElementClickInterceptedException:
            driver.execute_script("arguments[0].click();", element)

def print_test_header():
    print("=" * 50)
    print("RESTART: Medical Record Download Test")
    print("*" * 20 + "Place TEST" + "*" * 20)
    print("*" * 50)

def print_test_success():
    print("*" * 50)
    print("Medical Record Downloaded Successfully!")
    print("*" * 50)

def login(driver):
    driver.get("http://127.0.0.1:5000/auth/login")

    try:
        # Wait for login form
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, "loginForm"))
        )
        print("Login form found")

        # Fill in credentials
        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "identifier"))
        )
        email_field.send_keys("janepat@gmail.com")
        print("Email entered")

        password_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.NAME, "password"))
        )
        password_field.send_keys("password")
        print("Password entered")
        # Click login button
        login_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.XPATH,
"//button[@type='submit']"))
        )
        click_element_safely(driver, login_button)
        print("Login button clicked")

        # Check for error messages
        try:
            error_message = driver.find_element(By.CLASS_NAME, "alert")
            print(f"Login error: {error_message.text}")
            return False
        except NoSuchElementException:
            pass

        # Wait for redirect to dashboard
        WebDriverWait(driver, 20).until(
            lambda driver: driver.current_url ==
"http://127.0.0.1:5000/patient/dashboard"
        )
        print("Successfully logged in and redirected to dashboard")
        return True

    except Exception as e:
        print(f"Login failed: {str(e)}")
        driver.save_screenshot("login_error.png")
        return False

def test_medical_record_download(driver):
    print_test_header()

    # First login
    if not login(driver):
        raise Exception("Login failed")

    try:
        # Navigate to medical records page
        driver.get("http://127.0.0.1:5000/patient/medical_records")
        print("Navigated to medical records page")

        # Wait for medical records to load
        time.sleep(5)  # Allow time for records to load

        # Find and click the first download button
        download_button = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, ".btn-
download"))
        )
        click_element_safely(driver, download_button)
        print("Clicked download button")

        # Wait for download to complete
        time.sleep(5)

        # Check if file was downloaded
        downloads_path = os.path.expanduser("~/Downloads")
        files = os.listdir(downloads_path)
        pdf_files = [f for f in files if f.startswith("medical_record_")
and f.endswith(".pdf")]

        if pdf_files:
            print("PDF file downloaded successfully")
            print_test_success()
        else:
            raise Exception("PDF file not found in downloads folder")

    except Exception as e:
        print(f"Test failed: {str(e)}")
        print(f"Current URL: {driver.current_url}")
        driver.save_screenshot("download_error.png")
        raise

def check_server_running():
    try:
        response = requests.get("http://127.0.0.1:5000/")
        return response.status_code == 200
    except requests.ConnectionError:
        return False

def main():
    driver = setup_driver()
    try:
        test_medical_record_download(driver)
    finally:
        driver.quit()

if __name__ == "__main__":
    main()
```

Screenshot



```
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test> python .\downloa

DevTools listening on ws://127.0.0.1:59889/devtools/browser/8cb23153-c39a-4752-8d77-6d284cc2f8b5
==================================================
RESTART: Medical Record Download Test
********************Place TEST********************
**************************************************
Login form found
Email entered
Password entered
Login button clicked
Successfully logged in and redirected to dashboard
Navigated to medical records page
[22200:14980:1104/160917.962:ERROR:ssl_client_socket_impl.cc(878)] handshake failed; returned -1, SSL error code 1, net_error -101
[22200:14980:1104/160917.974:ERROR:ssl_client_socket_impl.cc(878)] handshake failed; returned -1, SSL error code 1, net_error -101
Clicked download button
Created TensorFlow Lite XNNPACK delegate for CPU.
PDF file downloaded successfully
**************************************************
Medical Record Downloaded Successfully!
**************************************************
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test>
```

Test report

Test Case 3	
Project Name: Unified Medical System	
Medical Record Download Test Case	
Test Case ID: Test_3	**Test Designed By: Devadethan R**

Test Priority(Low/Medium/High):			**Test Designed Date: 24/10/2024**		
Module Name: Medical Record Download			**Test Executed By : Jetty Benjamin**		
Test Title: Medical Record Download			**Test Execution Date: 26/10/2024**		
Description: Medical Record Download					
Pre-Condition: User is registered and has medical records					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Login to system	“janepat@gmail.com/PPpp!@12”	Login should be successful	Login successful	Pass
2	Navigate to medical records	http://127.0.0.1:5000/patient/medical_records	Medical records page should load	Page loaded successfully	Pass
3	Locate download button	N/A	Download button should be visible	Button found	Pass
4	Click download button	N/A	Download should initiate	Download started	Pass
5	Verify download	N/A	PDF file should be in downloads folder	File downloaded successfully	Pass
Post-Condition: Medical record PDF should be available in user's downloads folder					

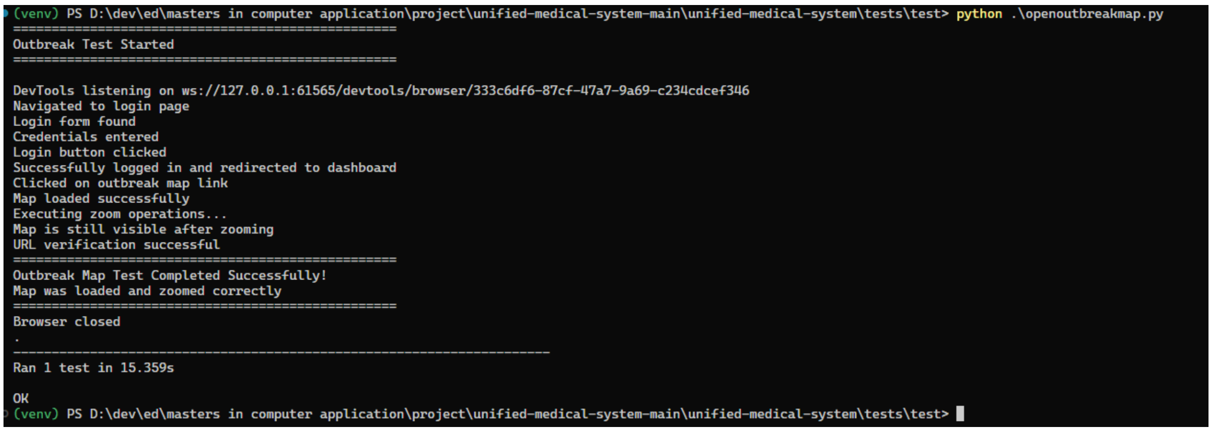
Test Case 4:

Code

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
import unittest
import time

def print_test_header():
    print("=" * 50)
    print("Outbreak Test Started")
    print("=" * 50)

def print_test_success():
    print("=" * 50)
    print("Outbreak Map Test Completed Successfully!")
    print("Map was loaded and zoomed correctly")
    print("=" * 50)

class TestOutbreakMap(unittest.TestCase):
    def setUp(self):
        print_test_header()
        self.driver = webdriver.Chrome()
        self.driver.maximize_window()
        # Navigate to login page
        self.driver.get("http://127.0.0.1:5000/auth/login")
        print("Navigated to login page")

        try:
            # Wait for login form
            WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located((By.ID, "loginForm"))
            )
            print("Login form found")

            # Login as admin
            email_field = self.driver.find_element(By.NAME, "identifier")
            password_field = self.driver.find_element(By.NAME, "password")

            email_field.send_keys("d.dethanr@gmail.com")
            password_field.send_keys("AAaa!@12")
            print("Credentials entered")

            login_button = self.driver.find_element(By.XPATH,
"//button[@type='submit']")
            login_button.click()
            print("Login button clicked")

            # Wait for dashboard to load
            WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located((By.CLASS_NAME, "content-
page"))
            )
            print("Successfully logged in and redirected to dashboard")

        except Exception as e:
            print(f"Setup failed: {str(e)}")
            self.driver.save_screenshot("login_error.png")
            raise

    def test_open_outbreak_map(self):
        try:
            # Navigate to outbreak map
            outbreak_map_link = WebDriverWait(self.driver, 10).until(
                EC.element_to_be_clickable((By.XPATH,
"//a[@href='/admin/outbreakmap']"))
            )
            outbreak_map_link.click()
            print("Clicked on outbreak map link")

            # Wait for map to load
            WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located((By.ID,
"map_ae515ffac4699a595b9141b0f72c6e09"))
            )
            print("Map loaded successfully")

            # Get the map element
            map_element = self.driver.find_element(By.ID,
"map_ae515ffac4699a595b9141b0f72c6e09")

            # Execute JavaScript to zoom in multiple times
            self.driver.execute_script("""
                var map =
document.querySelector('#map_ae515ffac4699a595b9141b0f72c6e09');
                var leafletMap = map._leaflet_map;
                if (leafletMap) {
                    // Store initial zoom level
                    var initialZoom = leafletMap.getZoom();
                    console.log('Initial zoom level:', initialZoom);

                    // Zoom in 3 times with delay
                    setTimeout(function() {
                        leafletMap.setZoom(initialZoom + 2);
                        console.log('First zoom completed');

                        setTimeout(function() {
                            leafletMap.setZoom(initialZoom + 4);
                            console.log('Second zoom completed');

                            setTimeout(function() {
                                leafletMap.setZoom(initialZoom + 6);
                                console.log('Final zoom completed');
                            }, 1000);
                        }, 1000);
                    }, 1000);
                }
            """)
            print("Executing zoom operations...")

            # Wait for zooming to complete
            time.sleep(4)

            # Verify map is still visible after zoom
            self.assertTrue(map_element.is_displayed())
            print("Map is still visible after zooming")

            # Verify current URL
            expected_url = "http://127.0.0.1:5000/admin/outbreakmap"
            self.assertEqual(self.driver.current_url, expected_url)
            print("URL verification successful")

            print_test_success()

        except TimeoutException:
            print("Test failed: Timeout waiting for map elements")
            self.driver.save_screenshot("map_error.png")
            self.fail("Timeout waiting for map elements")
        except Exception as e:
            print(f"Test failed: {str(e)}")
```

```
                self.driver.save_screenshot("test_error.png")
                raise

    def tearDown(self):
        if self.driver:
            self.driver.quit()
            print("Browser closed")

if __name__ == "__main__":
    unittest.main()
```

Screenshot



```
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test> python .\openoutbreakmap.py
==================================================
Outbreak Test Started
==================================================

DevTools listening on ws://127.0.0.1:61565/devtools/browser/333c6df6-87cf-47a7-9a69-c234cdcef346
Navigated to login page
Login form found
Credentials entered
Login button clicked
Successfully logged in and redirected to dashboard
Clicked on outbreak map link
Map loaded successfully
Executing zoom operations...
Map is still visible after zooming
URL verification successful
==================================================
Outbreak Map Test Completed Successfully!
Map was loaded and zoomed correctly
==================================================
Browser closed
.
----------------------------------------------------------------------
Ran 1 test in 15.359s

OK
(venv) PS D:\dev\ed\masters in computer application\project\unified-medical-system-main\unified-medical-system\tests\test>
```

Test Report:

Test Case 4

Project Name: Unified Medical System					
Outbreak Map Test Case					
Test Case ID: Test_4			**Test Designed By: Devadethan R**		
Test Priority(Low/Medium/High):			**Test Designed Date: 24/10/2024**		
Module Name: Admin Dashboard			**Test Executed By : Jetty Benjamin**		
Test Title : Outbreak Map Functionality			**Test Execution Date: 26/10/2024**		
Description: Verify outbreak map loading and interaction					
Pre-Condition: Admin account exists and has necessary permissions					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Login as admin	"d.dethanr@gmail.com/AAaa!@12"	Admin login should succeed	Login successful	Pass

2	Navigate to outbreak map	Click outbreak map link	Map page should load	Page loaded successfully	Pass
3	Verify map loading	N/A	Map should be visible	Map displayed correctly	Pass
4	Test zoom functionality	Multiple zoom levels	Map should zoom properly	Zoom operations successful	Pass
5	Verify map visibility	N/A	Map should remain visible after zoom	Map remained visible	Pass
Post-Condition: Admin should be able to view and interact with outbreak map					

CHAPTER 6

IMPLEMENTATION

6.1INTRODUCTION

Implementation is the stage of the project where the theoretical design is turned into a working system. It can be the most crucial stage in achieving a successful new system gaining the users confidence that the new system will work and will be effective and accurate. It is primarily concerned with user training and documentation. Conversion usually takes place about the same time the user is being trained or later. Implementation simply means convening a new system design into operation, which is the process of converting a new revised system design into an operational one.

At this stage the main work load, the greatest upheaval and the major impact on the existing system shifts to the user department. If the implementation is not carefully planned or controlled, it can create chaos and confusion.

Implementation includes all those activities that take place to convert from the existing system to the new system. The new system may be a totally new, replacing an existing manual or automated system or it may be a modification to an existing system. Proper implementation is essential to provide a reliable system to meet organization requirements. The process of putting the developed system in actual use is called system implementation. This includes all those activities that take place to convert from the old system to the new system. The system can be implemented only after through testing is done and if it is found to be working according to the specifications. The system personnel check the feasibility of the system. The more complex the system being implemented, the more involved will be the system analysis and design effort required to implement the three main aspects: education and training, system testing and changeover. The implementation state involves the following tasks:

- Careful planning.
- Investigation of system and constraints
- Design of methods to achieve the changeover.

6.2 IMPLEMENTATION PROCEDURES

Implementation of software refers to the final installation of the package in its real environment, to the satisfaction of the intended uses and the operation of the system. In many organizations someone who will not be operating it, will commission the software development project. In the initial stage people doubt about the software but we must ensure that the resistance does not build up, as one must make sure that:

- The active user must be aware of the benefits of using the new system. Their confidence in the software is built up.
- Proper guidance is imparted to the user so that he is comfortable in using the application.

Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not up running on the server, the actual process will not take place

6.2.1 User Training

User training is designed to prepare the user for testing and converting the system. To achieve the objective and benefits expected from computer-based system, it is essential for the people who will be involved to be confident of their role in the new system. As system becomes more complex, the need for training is more important. By user training the user comes to know how to enter data, respond to error messages, interrogate the database, and call up routine that will produce reports and perform other necessary functions.

6.2.2 Training on the Application Software

After providing the necessary basic training on computer awareness the user will have to be trained on the new application software. This will give the underlying philosophy of the use of the new system such as the screen flow, screen design type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the ways to correct the date entered. It should then cover information needed by the specific user/ group to use the system or part of the system while imparting the training of the program on the application. This training may be different across different user groups and across different levels of hierarchy.

6.2.3 System Maintenance

System maintenance is an effective component such that System maintenance refers to the ongoing activities required to ensure that a system or application operates effectively and efficiently after it has been implemented. It involves regular updates, bug fixes, and performance optimizations to keep the system running smoothly and securely.

System maintenance is essential to ensure that a system remains operational and effective after implementation. By establishing maintenance procedures and following them consistently, project teams can ensure that the system operates smoothly, remains secure, and continues to meet the needs of the end-users.

6.2.4 Hosting

Hosting allows a website to be accessible online by storing its files on a server. Different hosting types cater to various needs: shared hosting is budget-friendly, VPS offers more control, dedicated hosting provides full server access, and cloud hosting improves scalability. Managed hosting handles technical tasks for you, while self-hosting allows complete control for tech-savvy users. Good hosting ensures website reliability, speed, and security, impacting user experience and SEO.

Render Hosting

Render hosting often includes features like server-side rendering (SSR), static site generation (SSG), and content delivery networks (CDNs) to optimize load times and improve scalability. By pre-rendering pages or using SSR, the server sends fully constructed HTML to users, boosting speed and SEO performance.

Procedure for hosting a website on 000Webhost:

Step 1: Set up a Render Account

• Visit Render's website and sign up for a new account or log in if you already have one.

Step 2: Link GitHub Repository

• Once logged in to Render, navigate to the Dashboard and click on the New button to create a new service.

• Select Web Service as the type of service.

• When prompted, connect your GitHub account to Render if you haven't already done so.

• After linking your GitHub account, choose the UMS repository from the list of available repositories.

Step 3: Configure Deployment Settings

• Select the appropriate branch for deployment (usually main or master).

• Under Environment, choose the correct runtime for your project (for example, Python 3.x for Flask applications).

• Set the Build Command to install dependencies: pip install -r requirements.txt.

• Set the Start Command to run the flask application: gunicorn UMS.wsgi.

Step 4: Set up the Database (MongoDB)

• Since the project uses mongodb as the database, no complex database configuration is required on Render. However, make sure the database file is included in the repository

and is properly configured within your project.

• If you plan to migrate your database, ensure that the correct MongoDB commands are run

during deployment to apply migrations (e.g., python manage.py migrate).

Step 5: Deploy the Project

• Once everything is set up, click Create Web Service to begin the deployment process.

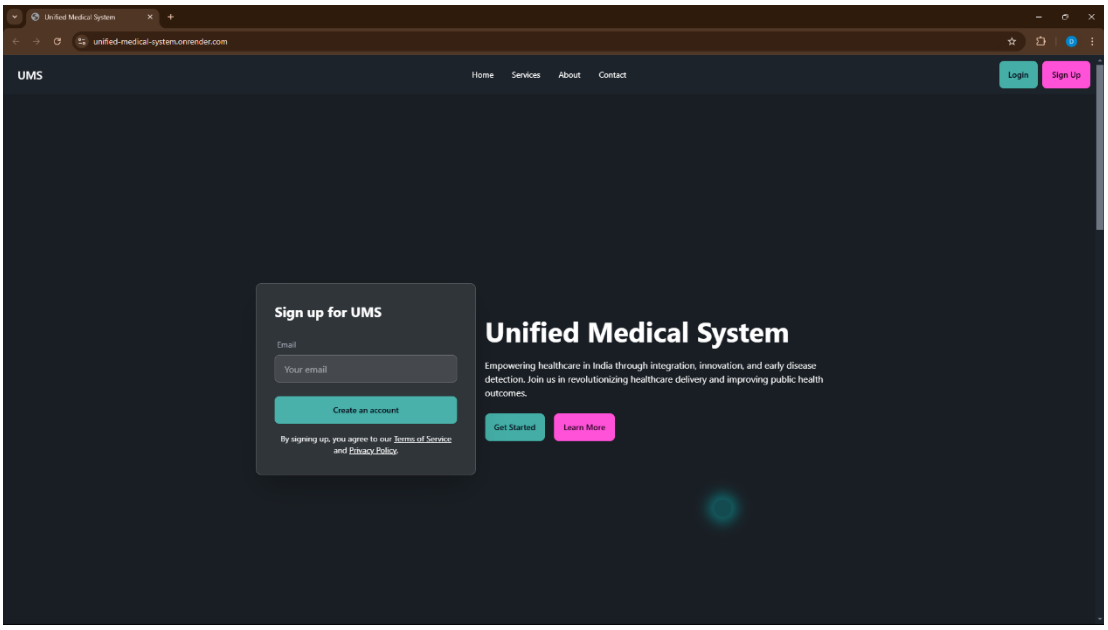
• Render will automatically build the project, install dependencies, and start the web service.

• After the deployment is complete, your project will be live at the provided URL.

Hosted Link: https://unified-medical-system.onrender.com/

Hosted Link QR Code

Screenshot



Sign up for UMS

Email

Your email

Create an account

By signing up, you agree to our Terms of Service and Privacy Policy.

Unified Medical System

Empowering healthcare in India through integration, innovation, and early disease detection. Join us in revolutionizing healthcare delivery and improving public health outcomes.

Get Started Learn More

Fig 9: Hosted Project

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In conclusion, the Unified Medical System (UMS) presents a transformative solution for healthcare delivery in India, centralizing services such as appointment booking, medical record access, data sharing, and early disease outbreak detection within a secure, user-friendly platform. By leveraging modern technologies, the UMS not only enhances healthcare accessibility and efficiency but also empowers patients to manage their health information actively and promotes proactive public health measures. Though challenges exist in infrastructure, data privacy, and user adoption, a phased implementation, combined with robust data security protocols and comprehensive user training, ensures the system's technical, economic, and behavioral feasibility. The UMS holds significant potential to improve healthcare outcomes, optimize resource allocation, and support India's vision for a more integrated, responsive, and resilient healthcare system.

7.2 FUTURE SCOPE

The future scope of the Unified Medical System (UMS) holds immense potential for transforming healthcare in India. As infrastructure develops, the UMS can be expanded into rural areas, ensuring equitable access to healthcare resources nationwide. Telemedicine integration offers patients the convenience of remote consultations, making healthcare accessible from any location, especially beneficial for those in remote areas. With advanced data analytics, the system can leverage machine learning to predict healthcare needs, allowing for better resource allocation and proactive care. The UMS could also integrate data from wearable devices to enable real-time monitoring, early intervention for chronic diseases, and preventive healthcare measures. Additionally, aggregated health data from the UMS can aid public health officials in tracking trends, forecasting disease outbreaks, and creating evidence-based policies, improving public health outcomes. AI-driven diagnostics and personalized care solutions based on patient history will also empower healthcare providers to offer targeted, efficient care. With these advancements, the UMS has the potential to become a model for healthcare systems globally, facilitating international collaboration and contributing to a more resilient, interconnected healthcare framework.

.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- Chatterjee, A., Kumar, V., & Mahapatra, S. (2020). Machine learning models for disease surveillance: A review. *Journal of Public Health and Epidemiology*, 12(4), 98–104.
- Kumar, R., & Singh, P. (2022). Digital health infrastructure in India: A review of Ayushman Bharat Digital Mission. *Journal of Health Policy and Systems Research*, 10(2), 67–73.
- Nair, S., Gupta, A., & Bhatia, M. (2021). The impact of AI-driven chatbots on healthcare: Improving patient engagement and self-management. *Healthcare Informatics Research*, 27(1)

WEBSITES:

- https://doi.org/10.5897/JPHE2020
- https://doi.org/10.1016/j.jhpsr.2022.07.004
- https://doi.org/10.1146/hsj2021.60103

CHAPTER 9

APPENDIX

9.1 Sample Code

Login

re

```
import from flask import Blueprint, render_template, redirect, url_for, request, flash, session
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required
from app import mongo, login_manager, oauth, mail
from app.models import User
from bson.objectid import ObjectId
from flask_mail import Message
from app.utils import generate_patient_id
from datetime import datetime
import random

auth_bp = Blueprint('auth', __name__)

@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)

@auth_bp.route('/register', methods=['GET'])
def register():
    return render_template('common/register.html')

@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        identifier = request.form['identifier']
        password = request.form['password']
        user = User.find_by_identifier(identifier)
        if user:
            if user.passwordHash is None:
                flash('User password is missing')
                return redirect(url_for('auth.login'))

            #role selection
            if check_password_hash(user.passwordHash, password):
                session['umsId'] = user.umsId
                if user.rolesId == 4:
                    login_user(user)
                    return redirect(url_for('patient.index'), code=302)
                elif user.rolesId == 3:
                    login_user(user)
                    if user.status == 'awaiting_approval':
                        flash('Your account is waiting for admin approval')
                        return render_template('common/awaiting_response.html')
                    elif user.status == 'active':
                        return redirect(url_for('doctor.index'), code=302)
                elif user.rolesId == 2:
                    login_user(user)
                    return redirect(url_for('hospital.index'), code=302)
                elif user.rolesId == 1:
```

```
          login_user(user)
          return redirect(url_for('admin.index'), code=302)
      else:
        flash('Invalid identifier or password')
    else:
      flash('User not found, please create an account')
  return render_template('auth/login.html')
```

Register

```
def generate_patient_id():
  return 'UMSP' + re.sub('-', '', str(uuid.uuid4()))[:8].upper()

@patient_bp.route('/register', methods=['GET', 'POST'])
def register():
  if request.method == 'POST':
    email = request.form['email']
    password = request.form['password']
    if password != request.form['confirm_password']:
      flash('Passwords do not match. Please try again.', 'error')
      return redirect(url_for('patient.register'))

    existing_user = mongo.db.users.find_one({'email': email})
    if existing_user:
      flash('Email already exists! Please login.', 'error')
      return redirect(url_for('patient.register'))
    name = request.form['name']
    phone_number = request.form['phonenumber']
    state = request.form['state']
    umsId = generate_patient_id()
    created_at = updated_at = datetime.now()
    new_user = {
      'umsId': umsId,
      'name': name,
      'email': email,
      'phoneNumber': [phone_number],
      'state': state,
      'status': 'active',
      'createdAt': created_at,
      'updatedAt': updated_at,
      'passwordHash':[generate_password_hash(password)],
      'rolesId': 4
    }
    mongo.db.users.insert_one(new_user)
    mongo.db.patients.insert_one({
      'umsId': umsId,
      'dateOfBirth': None,
      'gender': None,
      'createdAt': created_at,
      'updatedAt': updated_at
    })
    mongo.db.login.insert_one({
      'umsId': umsId,
      'email': email,
      'passwordHash': new_user['passwordHash'],
      'rolesId': 4,
      'status': 'active',
    })
    flash('User registered successfully!', 'success')
    return redirect(url_for('auth.login'))
  return render_template('patient/register.html')
```

Book Appointment

```
@patient_bp.route('/book_appointment', methods=['POST'])
@login_required
def book_appointment():
    data = request.json
    patient_id = current_user.umsId
    hospital_id = data.get('hospitalId')
    doctor_id = data.get('doctorId')
    category = data.get('category')
    appointment_date_str = data.get('appointmentDate')
    is_disabled = data.get('isDisabled', False)
    disability_id = data.get('disabilityId')
    reason = data.get('reason')
    if not all([hospital_id, doctor_id, category, appointment_date_str]):
        return jsonify({'success': False, 'message': 'Missing required fields'}), 400
    try:
        appointment_date = parser.isoparse(appointment_date_str)
        appointment_date_utc = appointment_date.astimezone(timezone.utc)
        new_appointment = {
            'patientId': patient_id,
            'hospitalId': hospital_id,
            'doctorId': doctor_id,
            'category': category,
            'appointmentDate': appointment_date_utc,
            'status': 'pending',
            'isDisabled': is_disabled,
            'disabilityId': disability_id if is_disabled else None,
            'reason': reason,
            'createdAt': datetime.now(timezone.utc),
            'updatedAt': datetime.now(timezone.utc)
        }
        result = mongo.db.appointments.insert_one(new_appointment)
        if result.inserted_id:
            return jsonify({'success': True, 'message': 'Appointment booked successfully'})
        else:
            return jsonify({'success': False, 'message': 'Failed to book appointment'}), 500
    except Exception as e:
        return jsonify({'success': False, 'message': 'An error occurred while booking the appointment'}), 500
```

Download medical certificate

```
@patient_bp.route('/api/medical_records/<block_id>/pdf', methods=['GET'])
@login_required
```

```
def download_medical_record_pdf(block_id):
    # Find the specific block
    block = mongo.db.medicalRecords.find_one({'_id': ObjectId(block_id)})
    if not block:
        return jsonify({'error': 'Record not found'}), 404
    # Find the transaction for the current user
    transaction = next((t for t in block['transactions'] if t['patientId'] == current_user.umsId), None)
    if not transaction:
        return jsonify({'error': 'Record not found for this patient'}), 404
    # Create a PDF
    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=letter,
                            rightMargin=72, leftMargin=72,
                            topMargin=72, bottomMargin=18)
    # Container for the 'Flowable' objects
    elements = []
    # Styles
    styles = getSampleStyleSheet()
    styles.add(ParagraphStyle(name='Justify', alignment=1))
    styles.add(ParagraphStyle(name='Center', alignment=1))
    # Add UMS logo
    logo_path = os.path.join(current_app.root_path, 'static', 'images', 'ums_logo.png')
    if os.path.exists(logo_path):
        logo = Image(logo_path, width=1.5*inch, height=1.5*inch)
        elements.append(logo)
    # Add title
    title = Paragraph("Unified Medical System", styles['Heading1'])
    elements.append(title)
    elements.append(Spacer(1, 12))
    # Add subtitle
    subtitle = Paragraph("Medical Certificate", styles['Heading2'])
    elements.append(subtitle)
    elements.append(Spacer(1, 24))
    # Add content
    content = f"""
    This is to certify that the patient with UMS ID: {current_user.umsId} has been examined and treated at
our facility.
    The following medical record details the diagnosis, treatment, and recommendations for the patient.
    """
    elements.append(Paragraph(content, styles['Justify']))
    elements.append(Spacer(1, 12))
    # Create a table for the medical record details
    data = [
        ['Patient ID:', current_user.umsId],
        ['Date of Record:', transaction['createdAt'].strftime('%Y-%m-%d %H:%M:%S')],
        ['Doctor ID:', transaction['doctorId']],
        ['Hospital ID:', transaction['hospitalId']],
        ['Symptoms:', transaction['Symptoms']],
        ['Diagnosis:', transaction['Diagnosis']],
        ['Treatment Plan:', transaction['TreatmentPlan']],
        ['Prescription:', transaction['Prescription']],
        ['Additional Notes:', transaction['AdditionalNotes']],
        ['Follow-up Date:', transaction['FollowUpDate']],
    ]
    table = Table(data, colWidths=[2*inch, 4*inch])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (0, -1), colors.lightblue),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.darkblue),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 10),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 12),
        ('BACKGROUND', (1, 1), (-1, -1), colors.lightgreen),
        ('BOX', (0, 0), (-1, -1), 1, colors.black),
        ('GRID', (0, 0), (-1, -1), 0.5, colors.black),
    ]))
    elements.append(table)
    elements.append(Spacer(1, 24))
    # Add footer
    footer_text = f"This medical certificate is electronically generated and is valid without a
signature.\nBlock Hash: {block['hash']}"
    footer = Paragraph(footer_text, styles['Center'])
    elements.append(footer)
    # Build the PDF
    doc.build(elements)
    buffer.seek(0)
    return send_file(buffer, as_attachment=True, download_name=f'medical_certificate_{block_id}.pdf',
mimetype='application/pdf')
```

9.1 Screen Shots

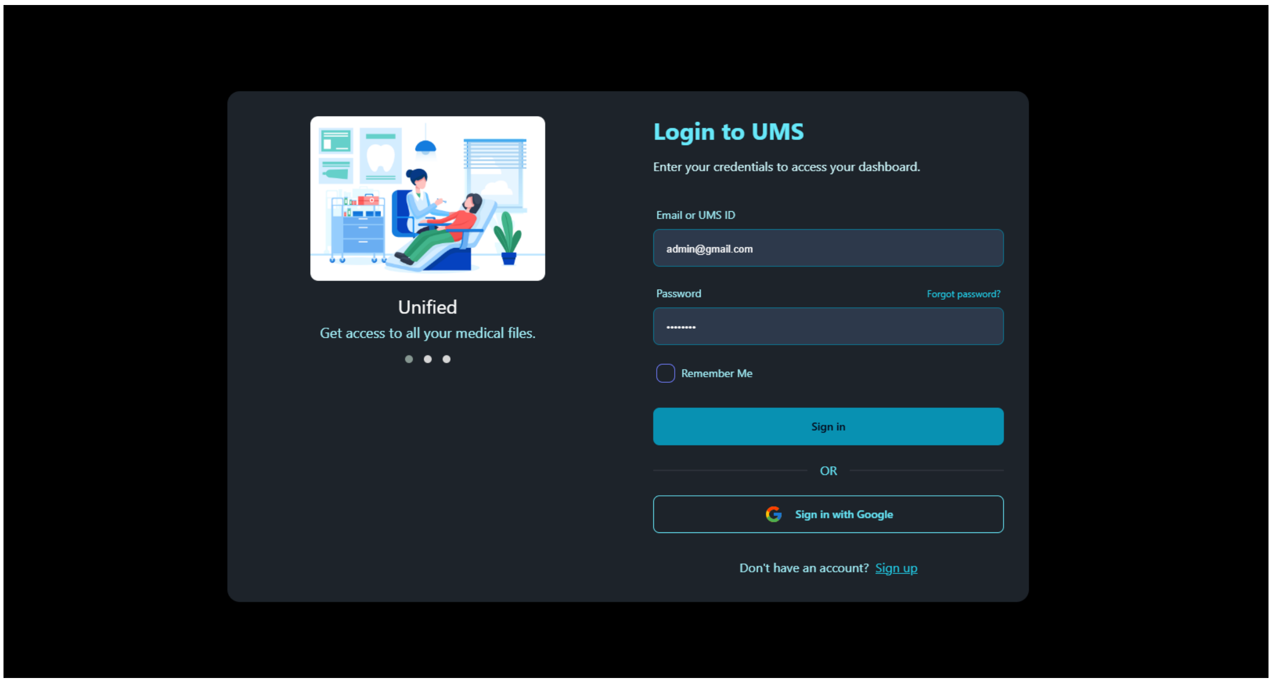


Fig 10: Login

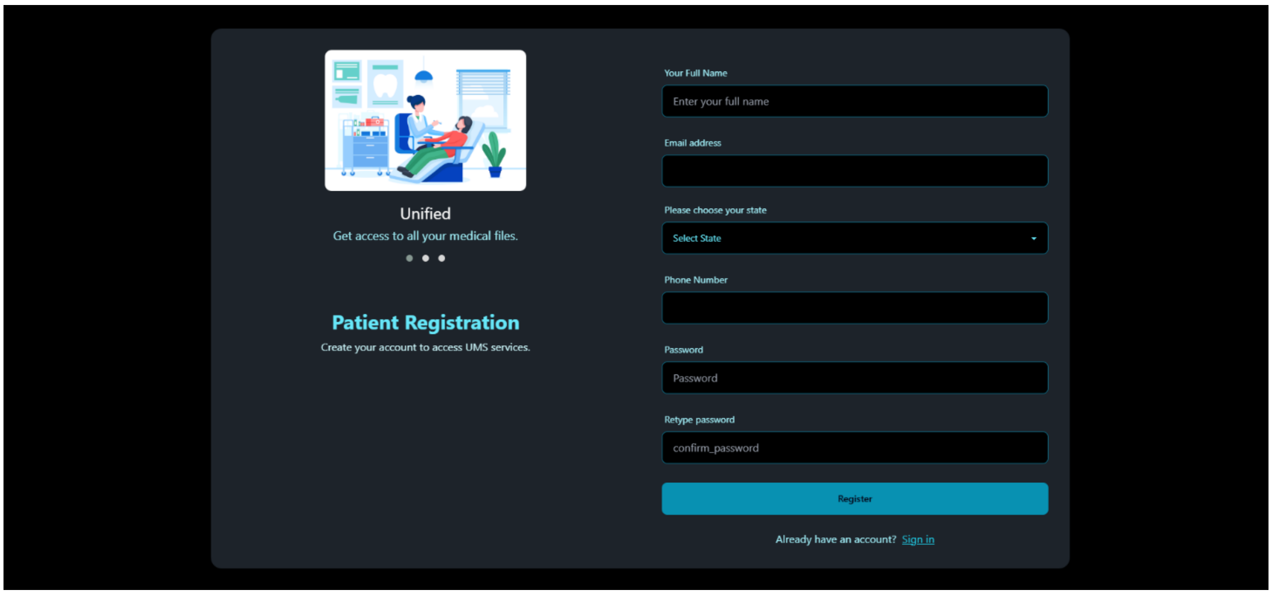


Fig 11: Register

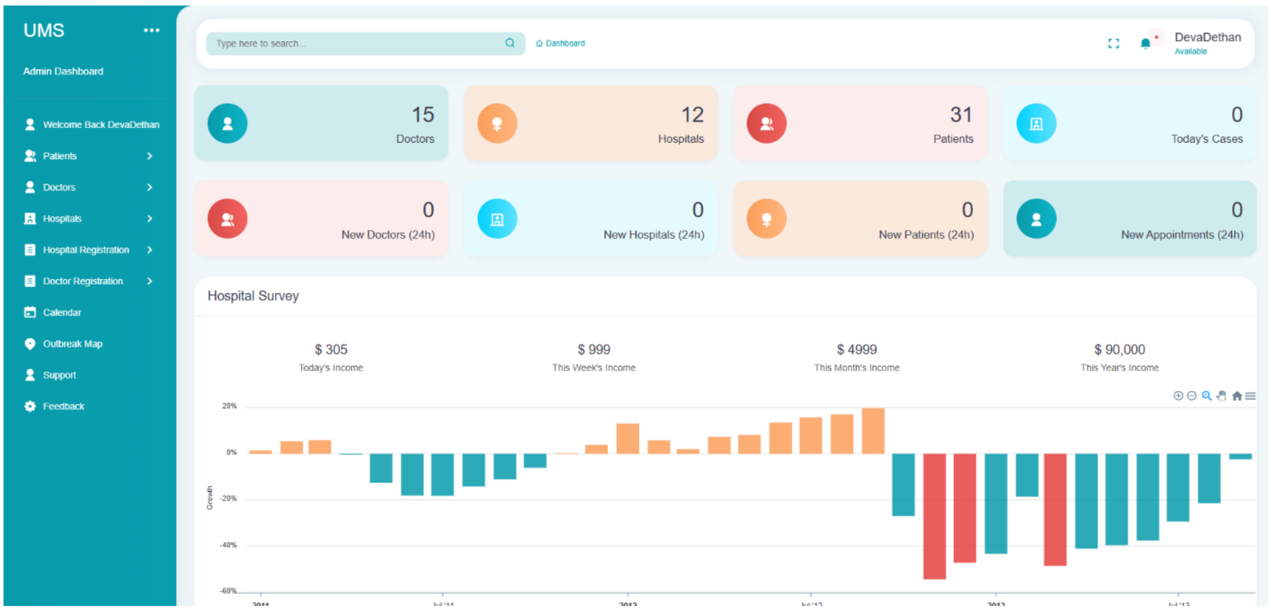
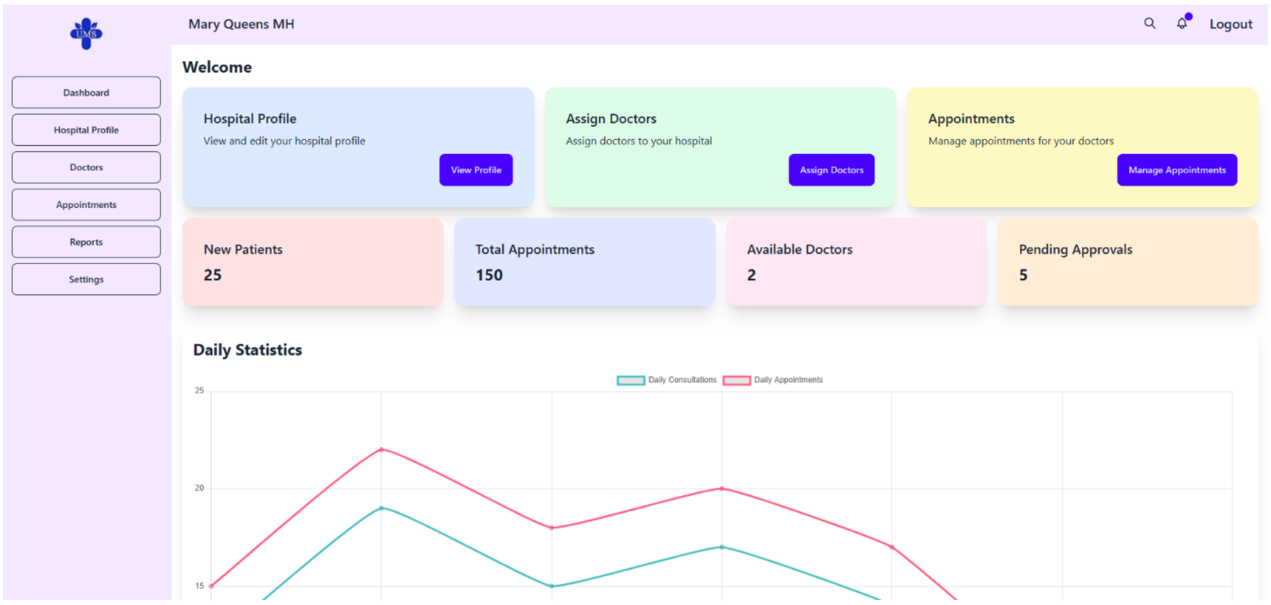


Fig 12: Admin Dashboard


Mary Queens MH

Logout

Dashboard

Hospital Profile

Doctors

Appointments

Reports

Settings

Welcome

Hospital Profile
View and edit your hospital profile
View Profile

Assign Doctors
Assign doctors to your hospital
Assign Doctors

Appointments
Manage appointments for your doctors
Manage Appointments

New Patients
25

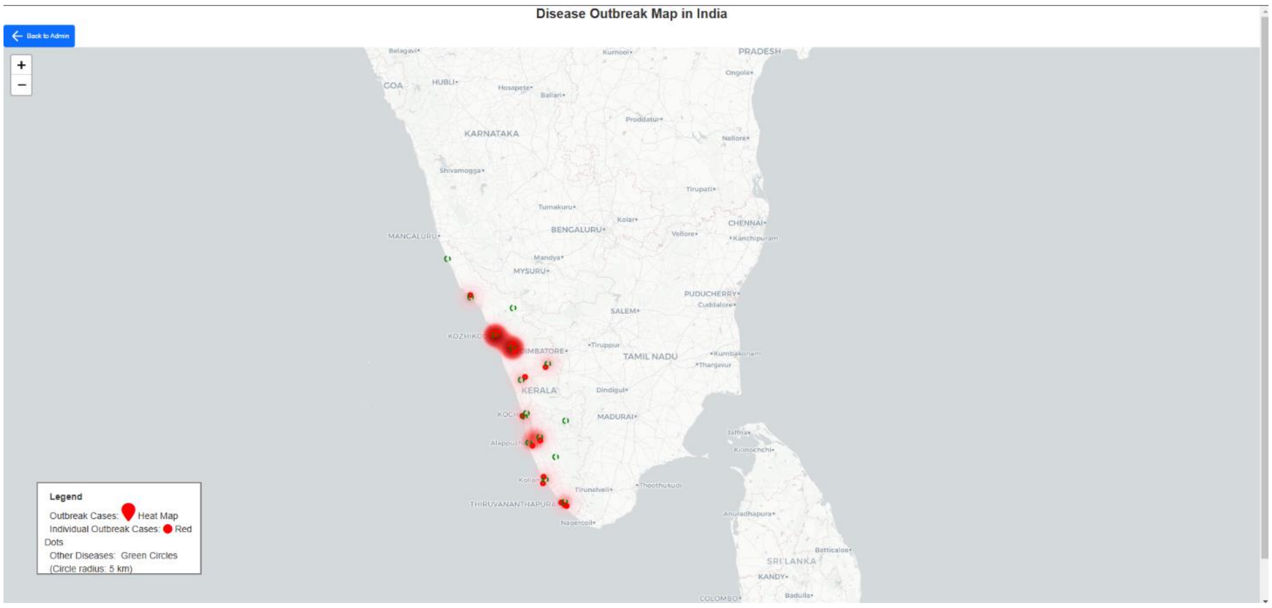
Total Appointments
150

Available Doctors
2

Pending Approvals
5

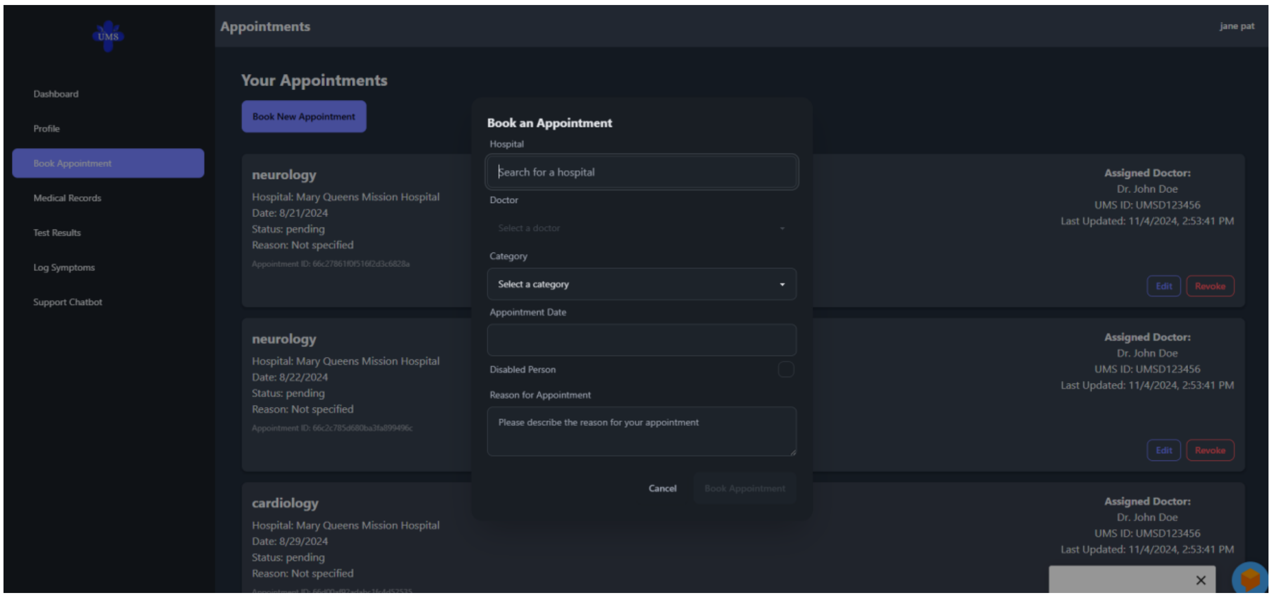
Daily Statistics

Fig 13: Hospital Dashboard


Disease Outbreak Map in India

Legend
Outbreak Cases: Heat Map
Individual Outbreak Cases: ● Red Dots
Other Diseases: Green Circles
(Circle radius: 5 km)

Fig 14: Outbreak Detection map



Your Appointments

neurology
Hospital: Mary Queens Mission Hospital
Date: 8/21/2024
Status: pending
Reason: Not specified
Appointment ID: 66c27861f0f516f2d3c6828a

Assigned Doctor:
Dr. John Doe
UMS ID: UMSD123456
Last Updated: 11/4/2024, 2:53:41 PM

neurology
Hospital: Mary Queens Mission Hospital
Date: 8/22/2024
Status: pending
Reason: Not specified
Appointment ID: 66c2c785d680ba3fa899496c

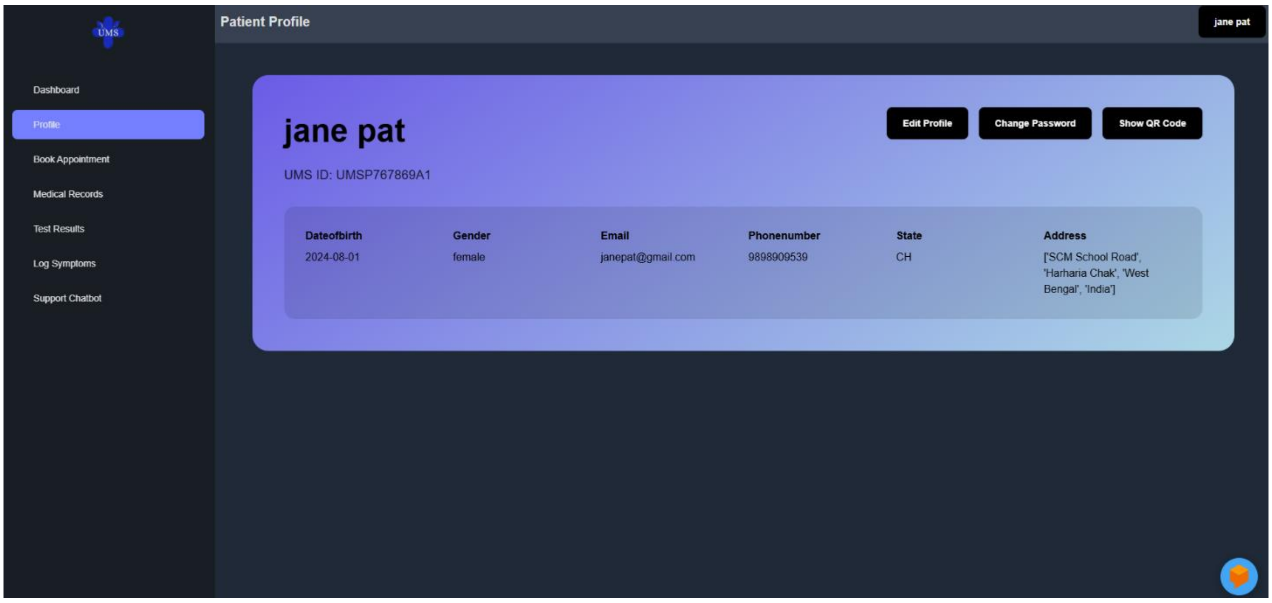
Assigned Doctor:
Dr. John Doe
UMS ID: UMSD123456
Last Updated: 11/4/2024, 2:53:41 PM

cardiology
Hospital: Mary Queens Mission Hospital
Date: 8/29/2024
Status: pending
Reason: Not specified

Assigned Doctor:
Dr. John Doe
UMS ID: UMSD123456
Last Updated: 11/4/2024, 2:53:41 PM

Book an Appointment
Hospital: Search for a hospital
Doctor: Select a doctor
Category: Select a category
Appointment Date:
Disabled Person
Reason for Appointment: Please describe the reason for your appointment

Fig 15: Patient Booking



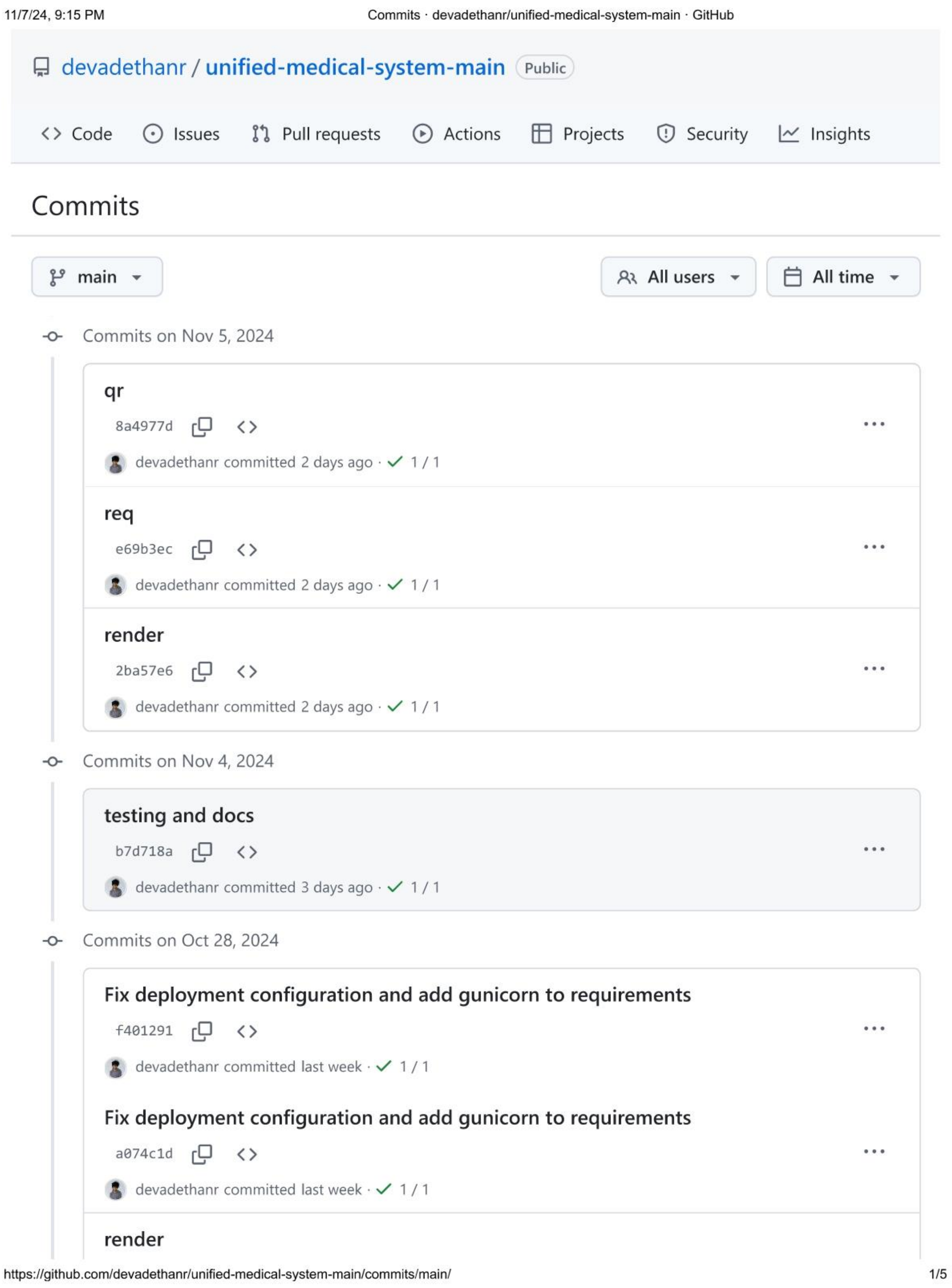
jane pat

UMS ID: UMSP767869A1

Dateofbirth	Gender	Email	Phonenumber	State	Address
2024-08-01	female	janepat@gmail.com	9898909539	CH	['SCM School Road', 'Harharia Chak', 'West Bengal', 'India']

Fig 16: Patient user profile

9.3 Git Log


11/7/24, 9:15 PM Commits · devadethanr/unified-medical-system-main · GitHub

devadethanr / unified-medical-system-main Public

Commits

Commits on Nov 5, 2024

qr
8a4977d
devadethanr committed 2 days ago · ✓ 1 / 1

req
e69b3ec
devadethanr committed 2 days ago · ✓ 1 / 1

render
2ba57e6
devadethanr committed 2 days ago · ✓ 1 / 1

Commits on Nov 4, 2024

testing and docs
b7d718a
devadethanr committed 3 days ago · ✓ 1 / 1

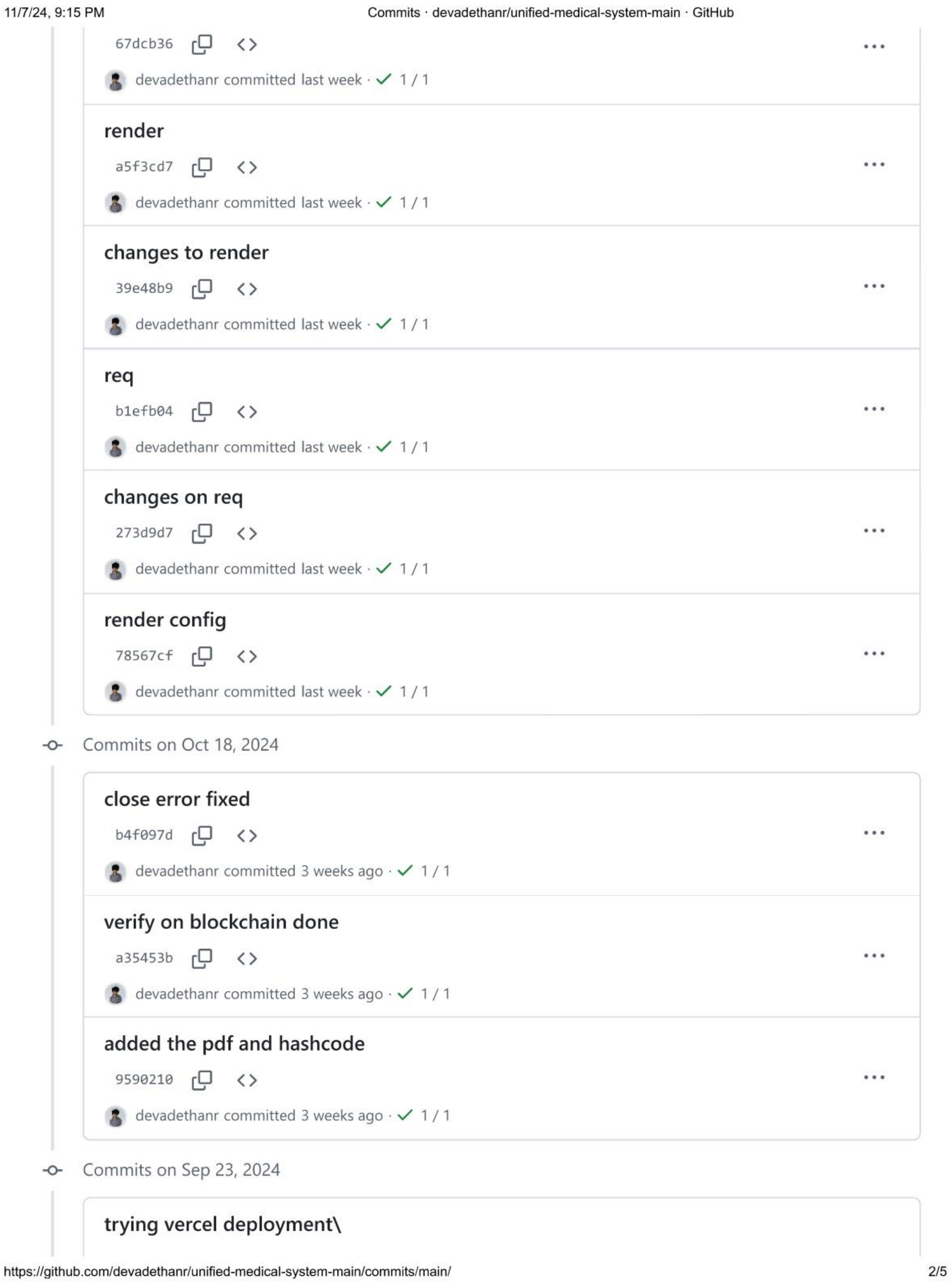
Commits on Oct 28, 2024

Fix deployment configuration and add gunicorn to requirements
f401291
devadethanr committed last week · ✓ 1 / 1

Fix deployment configuration and add gunicorn to requirements
a074c1d
devadethanr committed last week · ✓ 1 / 1

render

https://github.com/devadethanr/unified-medical-system-main/commits/main/ 1/5



11/7/24, 9:15 PM Commits · devadethanr/unified-medical-system-main · GitHub

67dcb36

devadethanr committed last week · ✓ 1 / 1

render

a5f3cd7

devadethanr committed last week · ✓ 1 / 1

changes to render

39e48b9

devadethanr committed last week · ✓ 1 / 1

req

b1efb04

devadethanr committed last week · ✓ 1 / 1

changes on req

273d9d7

devadethanr committed last week · ✓ 1 / 1

render config

78567cf

devadethanr committed last week · ✓ 1 / 1

Commits on Oct 18, 2024

close error fixed

b4f097d

devadethanr committed 3 weeks ago · ✓ 1 / 1

verify on blockchain done

a35453b

devadethanr committed 3 weeks ago · ✓ 1 / 1

added the pdf and hashcode

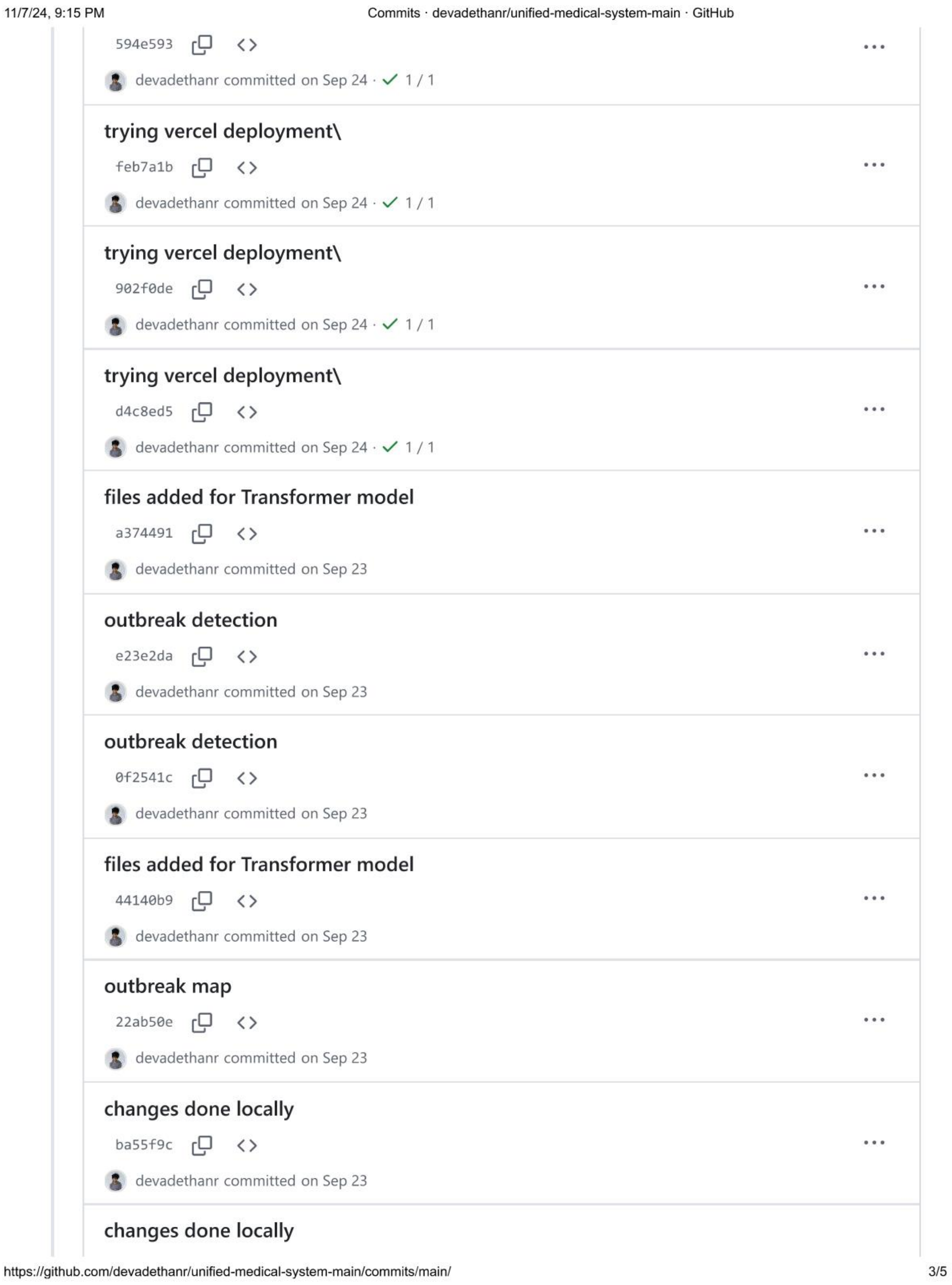
9590210

devadethanr committed 3 weeks ago · ✓ 1 / 1

Commits on Sep 23, 2024

trying vercel deployment\

https://github.com/devadethanr/unified-medical-system-main/commits/main/ 2/5


11/7/24, 9:15 PM Commits · devadethanr/unified-medical-system-main · GitHub

594e593
devadethanr committed on Sep 24 · ✓ 1 / 1

trying vercel deployment\
feb7a1b
devadethanr committed on Sep 24 · ✓ 1 / 1

trying vercel deployment\
902f0de
devadethanr committed on Sep 24 · ✓ 1 / 1

trying vercel deployment\
d4c8ed5
devadethanr committed on Sep 24 · ✓ 1 / 1

files added for Transformer model
a374491
devadethanr committed on Sep 23

outbreak detection
e23e2da
devadethanr committed on Sep 23

outbreak detection
0f2541c
devadethanr committed on Sep 23

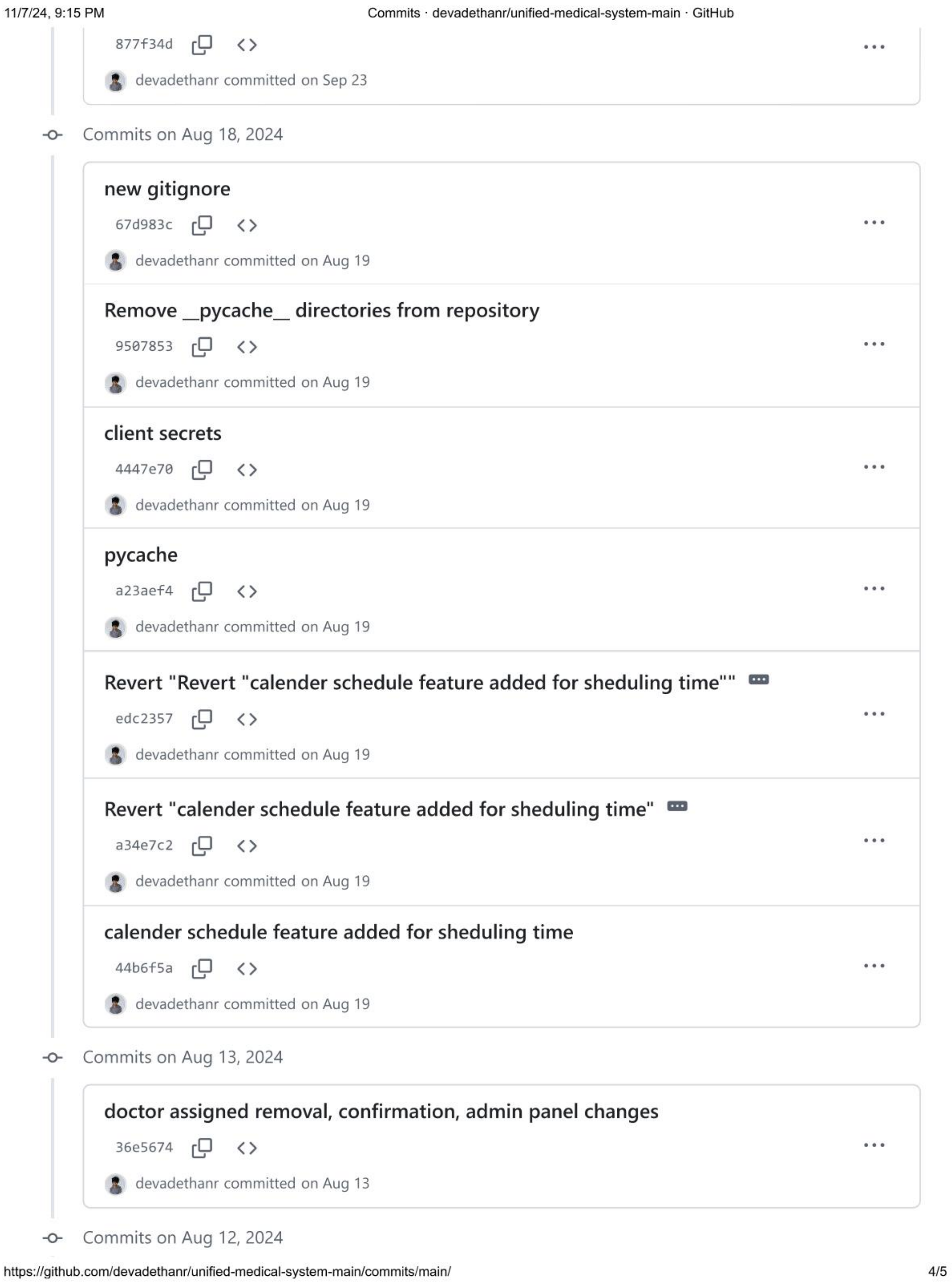
files added for Transformer model
44140b9
devadethanr committed on Sep 23

outbreak map
22ab50e
devadethanr committed on Sep 23

changes done locally
ba55f9c
devadethanr committed on Sep 23

changes done locally

https://github.com/devadethanr/unified-medical-system-main/commits/main/ 3/5


11/7/24, 9:15 PM Commits · devadethanr/unified-medical-system-main · GitHub

877f34d

devadethanr committed on Sep 23

Commits on Aug 18, 2024

new gitignore
67d983c
devadethanr committed on Aug 19

Remove __pycache__ directories from repository
9507853
devadethanr committed on Aug 19

client secrets
4447e70
devadethanr committed on Aug 19

pycache
a23aef4
devadethanr committed on Aug 19

Revert "Revert "calender schedule feature added for sheduling time""
edc2357
devadethanr committed on Aug 19

Revert "calender schedule feature added for sheduling time"
a34e7c2
devadethanr committed on Aug 19

calender schedule feature added for sheduling time
44b6f5a
devadethanr committed on Aug 19

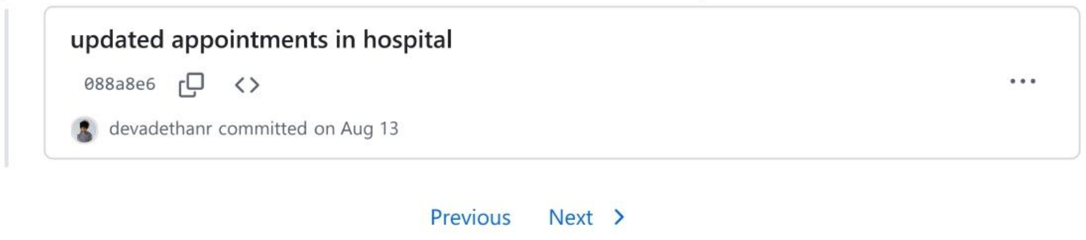
Commits on Aug 13, 2024

doctor assigned removal, confirmation, admin panel changes
36e5674
devadethanr committed on Aug 13

Commits on Aug 12, 2024

https://github.com/devadethanr/unified-medical-system-main/commits/main/ 4/5

11/7/24, 9:15 PM Commits · devadethanr/unified-medical-system-main · GitHub



updated appointments in hospital

088a8e6

devadethanr committed on Aug 13

https://github.com/devadethanr/unified-medical-system-main/commits/main/ 5/5