DEVELOPMENT OF A MEDICAL INFORMATION SYSTEM FOR THE UNIVERSITY HEALTH CENTER

A PROJECT REPORT SUBMITTED BY

D.K.Y.Y. DASSANAYAKE
(S/20/335)

To the Board of Study in
DEPARTMENT OF STATISTICS AND COMPUTER SCIENCE

in partial fulfillment of the requirement for the award of the degree of

B.Sc. (Honours) in Computer Science

of the

UNIVERSITY OF PERADENIYA
SRI LANKA
2024

I do hereby declare that the work reported in this project report was exclusively carried out by me under the supervision of **Dr. Hakim Usoof**. It describes the results of my own independent work except where due reference has been made in the text. No part of this project report has been submitted earlier or concurrently for the same or any other degree.

Date: 23/ 11/ 2025 ……

Signature of the Candidate

Certified by:

1. Instructor: **Dr. Erunika O. Dayaratne**

   Date: ……. Signature:…….

2. Supervisor: **Dr. Hakim Usoof**

   Date: ……. Signature:…….

Department Stamp:

ABSTRACT

HEALTH CENTER MEDICAL INFORMATION SYSTEM UNIVERSITY OF PERADENIYA

D.K.Y.Y. Dassnayake

Department of Statistics and Computer Science, University of Peradeniya, Peradeniya, Sri Lanka

This report details the development of a Medical Information System (MIS), a comprehensive digital platform designed to modernize and streamline the healthcare services provided to a community, such as a university. At its heart, this project is about replacing fragmented, paper-based processes with a single, secure, and intuitive digital ecosystem, ensuring that patients receive timely and coordinated care. The system caters to the entire spectrum of clinical operations: for patients, it simplifies essential tasks like scheduling appointments, submitting required medical forms, and securely accessing their personal health records, including lab results and prescription history. For doctors and medical staff, the platform acts as an integrated workspace, enabling them to efficiently manage patient demographics, record vital signs, document diagnoses, issue electronic prescriptions, and seamlessly order lab investigations. Beyond the clinical workflow, the MIS incorporates dedicated modules for pharmacy inventory management, ensuring medicine availability and accurate dispensing, and a laboratory module for handling test requests and results reporting. Furthermore, it features robust administrative tools for user lifecycle management, audit logging for accountability, and a centralized announcement system. Built on a modern, robust technical foundation, security was paramount, incorporating data encryption and stringent access control to protect sensitive health information, ensuring user trust and regulatory compliance. Ultimately, this Medical Information System serves as a critical infrastructure upgrade, moving beyond simple digitization to fundamentally improve the speed, quality, and accessibility of healthcare for every member of the community

ACKNOWLEDGEMENTS

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CHAPTER 01
INTRODUCTION

1.1. BACKGROUND

The Health Centre in the University of Peradeniya has immense importance in ensuring the well-being of a diverse and large community, encompassing thousands of students, academic staff, and administrative personnel. This center operates much like a dedicated primary care facility, handling everything from routine check-ups and preventative health services to complex medical record maintenance, laboratory tests, and prescription dispensing. Traditionally, the core operational processes-which include patient registration, detailed medical history, diagnosis, and follow-up on prescription flow-depend heavily on conventional paper-based methods and various communication channels.

While the commitment of the healthcare staff is unquestionable, manual systems are inherently burdened with considerable inefficiencies in a dynamic, high-volume environment such as a modern university. Managing health information for a large, transient population of students, along with the need for timely and secure access to medical records in case of emergencies, highlighted the pressing need for cohesive digital transformation. The MIS project was, therefore, started to directly address these systemic inefficiencies by introducing one modern digital platform for centralizing and securing all health-related data and simplifying care delivery throughout the institution.

1.2. PROBLEM STATEMENT

Relying on manual workflows in the Health Centre creates several critical bottlenecks and risks to the efficiency, quality, and security of patient care.

The main issues identified in the existing non-digitized environment:

- Scattered data and inefficient access: Medical records remain scattered in paper files, with retrievals slow, prone to human error, and inaccessible virtually in real time. This fragmentation negatively impacts doctors' capability for quick, informed decisions in emergency situations.
- Bottlenecks in Workflow: Lab test requests, writing, and dispensing of

prescriptions are very time-consuming when done manually. This includes the long waiting period for patients and excessive administrative burdens that reduce staff productivity.

- Security and Compliance Risk: Large-scale physical record keeping inherently involves security risks regarding data privacy and long-term compliance. Without standardized, role-based digital access, the confidentiality of sensitive patient health information is at stake.
- Lack of Integrated Communication: Absence of any dedicated system for integrated communication; doctors, pharmacists, and lab technicians work in silos, leading to prescription fulfillment errors and lab result correlation.

This project directly addresses the need for overcoming those limitations by implementing a secure, integrated, and fully automated digital Medical Information System.

1.3. OBJECTIVES

The main objective of the project is to successfully design, develop, and deploy a robust MIS for the University Health Centre. Key objectives to guide the development process are:

- Centralize and Secure Patient Data: Establish one source database of all patient medical records that is encrypted, highly secure, confidential, and integral, with guarantees of immediate, authorized access by care providers.
- Improve Clinical Workflows: Automate and computerize key clinical and administrative workflows such as patient registration, appointment scheduling, processing of medical forms, and end-to-end lab request and prescription dispensing management.
- Ensure Role-Based Access Control: To implement strict authentication and authorization layers, including Google OAuth 2.0 integration and domain-based identification, to ensure that users-students, staff, doctors, or pharmacists-access data relevant to their specific role.
- Improve cross-functional integration by providing a seamless digital interface that integrates the doctors, the pharmacy, and the laboratory to enable automated communication, ordering, and sharing of results without common procedural

errors.

- Improve Transparency and Accountability: To implement comprehensive audit logging for all major system actions and transactions, establishing a transparent record of activity for compliance and administrative oversight.

1.4. SCOPE

The MIS project will be an integrated digital solution comprising diverse functional modules to cater to the multi-faceted needs of the University Health Centre. The system is architected to handle four primary user roles, namely, Patient, Doctor/Medical Officer, Lab Technician, and Pharmacist, including an Administrative role.

The key features that i was responsible for in the scope of this system include the following:

- Patient Management Portal: The system allows patients-students and staff-to register using their University credentials, manage personal health profiles, review medical records-diagnosis, medication, and lab results-online, and schedule appointments online.
- Administrative Oversight and Security: Includes essential admin functionalities, including the management and approval of user accounts, role access controls, community-wide Announcement management, and system-wide Audit Log review for security and operational purposes.
- Support and Communication: The system provides a Support Ticket system where users and staff alike can report any issues; a Notifications system keeps all users informed of appointment changes, prescription readiness, and lab results.

CHAPTER 02
METHODOLOGY

This chapter describes the methodology used for the Medical Information System (MIS) project, with an emphasis on the steps I took to finish the tasks I was given, which included the Administrator Portal, the Patient Portal, and core security mechanisms.

2.1. REQUIREMENT GATHERING & ANALYSIS

The successful design of the MIS was based on a very rigorous and collaborative Requirement Gathering and Analysis phase. This aimed at smoothly transitioning critical healthcare services into an integrated digital platform from its current legacy manual structure. To this end, my contribution in defining the technical architecture and security framework during this phase was key.

Stakeholder Engagement and Documentation

The process started with a series of structured engagements, comprising formal meetings and interviews of the principal stakeholders at the University Health Centre. Key participants included the Chief Medical Officer (CMO), Medical Officers (Doctors), Pharmacists, and other support staff. These sessions were meant to map the "as-is" paper-based processes, identify major bottlenecks, such as prescription handoffs and record retrieval delays, and define the "to-be" digitized workflow.

My main responsibilities regarding documentation included:

- Database Schema Design: I led the design of the relational database schema, focusing on storing sensitive data securely and retrieving it efficiently for Patient demographics, Vitals, Medical Records, Prescriptions, and Audit Logs while ensuring the integrity and normalization of the data.
- SRS: I led the writing of the final SRS, converting the qualitative input provided during staff meetings into a quantitative, verifiable set of requirements using the professional standards for such documents.
- RTM: I created the RTM to ensure that every agreed feature could be traced back to a defined business/user requirement. This is important for project scope management and facilitates testing.

- Definition of RBAC System: One of the key deliverables involved the detailed specification of the RBAC system that defined security policies and permissions regarding the different user roles: Admin, Doctor, Pharmacist, Lab Technician, and Patient/Student/Staff.

Detailed Requirement View

The outcome of the analysis phase resulted in detailed functional requirements across major system components that fell under my area of responsibility, including the RBAC System, the Patient Portal, and the Admin Portal.

I. Role-Based Access Control (RBAC) System

This forms the basis of this system for secure patient data and compliance, enforcing strict privilege separation:

- Authentication & Authorization: Implement secure user login verified against University credentials, ensuring each user is assigned exactly one primary role upon activation.
- Profile management: Allow all users-Patients, Doctors, and Staff-easily and securely to update their personal profile information.
- Scoping Access: Ensure data visibility is strictly enforced by role. For example, Patients can only view their own medical records and prescriptions, while Doctors can only view records for assigned or currently visiting patients.
- Audit Logging: Include audit logging for all critical activities such that no critical operations, like login attempts, record creation/viewing, prescription dispensing, changes in user roles, go unnoticed.

II. Patient Portal Capabilities

The Patient Portal is the main interface for all members of the University to interact with the Health Centre:

- Self-Registration and Profile: Provide for easy registration and profile management using authenticated University credentials.
- Appointment Scheduling: Provide the facility to book appointments and automatically send confirmations and reminders.

- Medical Records Access: Provide secure, read-only access to their complete medical history, including Vitals recorded by staff, Diagnoses, approved Medical Certificates, and all historical Prescriptions.
- Lab Result Status: Allow patients to see the status of their requested lab test.
- Support Ticket Submission: Provide a means through which to submit, track, and manage support and technical assistance requests via the portal itself.
- Notification System: This will notify the patients about their appointments or the completion of lab tests.

III. Administrator Portal Functionalities

The administrative interface centralizes management functions that are essential to system oversight and community communication:

- User Life Cycle Management: The different functionalities in this are approval of a new user account by the administrator, modification in the details of an existing user, and change or assignment of user roles such as Doctor, Pharmacist, Lab Technician, and Staff.
- Announcement Management: Provide tools for the creation and publishing of system-wide or targeted announcements for the community.
- System Monitoring: Provide access to the Audit Log, which monitors user activities and system health, important in the review of security and debugging.
- Support Ticket Oversight: This will allow the administration team to see all the tickets coming in, reassign, and manage the support tickets from the users.

2.2. KEY PROJECT DOCUMENTATION

Effective management of large-scale system development required the creation and diligent maintenance of several formal project documents. These documents were used as a single source of truth for planning, execution, stakeholder communication, and quality assurance.

Project Charter

The Project Charter also gave a high-level overview of the purpose, the intended final product (the MIS), and defined the boundaries of the project. The document was important in attaining early consensus between the University administration and the Health Centre. It documented the high-level objectives of the project in writing, identified key stakeholders and the initial project manager, established authority to commit resources, and set the initial scope and constraints.

Stakeholder Analysis Document

The Stakeholder Analysis Document identified all individuals, groups, or organizations involved in or affected by the MIS project. Examples include the CMO, Doctors, Patients, IT Center, and Pharmacists. It listed the levels of power and interest of each, priorities, and strategies required for engagement. This focused analysis clearly mapped the needs and potential levels of influence of stakeholders so that requirements-gathering sessions, as described in Section 2.1, focused on the input of high-power, high-interest groups, such as the CMO and core clinical staff. In this way, requirement drift was avoided, and the delivered system aligned with the critical operational needs.

Software Requirements Specification (SRS)

The SRS thoroughly identified all the functional and non-functional requirements, including performance, security (specifically, RBAC as explained above), quality attributes, and external interfaces. The SRS was the binding agreement between the development team and the client. It was the main reference for the entire design and implementation phase because it contained concrete specifications required to develop secure, high-quality code. Moreover, it identified the acceptance criteria for final system validation.

Requirement Traceability Matrix (RTM)

The RTM guarantees that each and every requirement in the SRS is implemented, tested, and verifiable. It became priceless in assuring quality and verification. It assured that no critical requirements related to the highest-priority features, such as secure patient data access and prescription processing, were missed during implementation or skipped during vigorous testing; thus, it ensured the delivered MIS was complete and met all the initial expectations of the client.

2.3. SYSTEM DESIGN

The system has been engineered with a scalable Three-Tier Architecture to ensure clear separation of concerns, flexibility, and maximum data security.

Core Architectural Roles

My individual approach was first to provide a solid foundation, on which all the other features should stand. My design contributions focused on the Application and Data Tiers:

- Security Architecture Lead: Designed the integration points for JSON Web Tokens and Spring Security to enforce the defined RBAC rules across the whole backend.
- API Design-modular: I designed the modular structure of the Application Tier, specifically defining the RESTful API endpoints of PatientController and AdminController, making sure that logical grouping was followed with security standards.

Backend System Design Breakdown

Backend is created with consideration to handle core business logic, enforcing security, and persisting data for the MIS. This promotes clean code architecture, maintainability, and efficient scaling.

- Technology & Framework: This layer is developed on Spring Boot (Java); therefore, it gives a solid, much-modular foundation for enterprise applications.
- Controller Layer (API Gateway): This layer is the external interface of the system. Its only responsibility is to receive the requests, perform a basic format validation, apply security by means of Spring Security filters, and forward the request to the appropriate service.
- Service Layer (Business Logic): This layer contains all the business rules and computational logic, encapsulated in classes such as AuditService.java and UserService.java. It also maintains a strict decoupling from Controller and Data Access for the ability to make independent changes to the presentation or persistence without touching core functionality. Some of the major logics developed here include transaction management, appointment validation, and calls for encryption/decryption of PHI.

- Data Access/Repository Layer: This is where database interaction is abstracted using Spring Data JPA, providing a convenient way to define CRUD related methods instead of writing boilerplate SQL.
- Data Tier: MySQL provides the persistent store for all relational data. The schema of the database is designed around the entities - Models that the Repository layer manages - with ACID compliance, ensuring a secure and centralized storage of data.

2.4. DEVELOPMENT

Development of the MIS was organized in successive, prioritized increments, within an overall Iterative and Incremental Development methodology. My responsibilities covered the security foundation, all Patient-Side features, and all Administrator functions, and I was responsible for the complete end-to-end implementation, Frontend and Backend, of the Patient Portal and Administrator Portal functionalities.

Security and Infrastructure Development

This foundational phase established the security and authentication framework, which was non-negotiable and implemented on both tiers:

RBAC :

- Backend: I developed the core logic of Role-Based Access Control with Spring Security using JwtTokenFilter.java.
- Frontend: Ensured dynamic rendering of UIs based on the authenticated user's role received via the JWT by securely controlling feature visibility.

User Authentication & Profile: I implemented the full-stack solution for secure user registration and profile management. This would include the backend CRUD logic to manage user data and corresponding React components for updating a user's profile and securely logging in or out of the application.

Patient Portal Feature Implementation

I developed the full functionality of end-to-end features available to students and staff

members:

First-Time Setup :

- Backend: Developed services and APIs to handle the submission and validation of the initial patient Medical Form data, along with other required personal information.
- Frontend: Designed the MedicalForm.jsx component and a reminder to show in the patient's dashboard, for users to finish the profile setup.

Appointment Scheduling :

- Backend: Developed full service and controller logic in order to handle slot validation, submission, and persistence while maintaining proper database management of appointment status.
- Frontend: The AppointmentsTab.jsx component was created to provide the user interface to book appointments, validate them, and show user appointments.

Medical Records and Reporting :

- Backend: In the file PatientController.java, implemented the protected endpoint to retrieve strictly limited information: diagnosis notes, medical recommendations, and lab report completion status. The controller enforces the rule against disclosing the full medical record.
- Frontend: Developed the ReportsTab.jsx component in React that consumes the limited API data and displays the summaries of authorized visits, recommendations, and lab status data, without the inclusion of unauthorized content or download links for comprehensive records.

System Notifications :

- Backend: Implemented complete service and controller logic for sending automated system notifications, such as appointment confirmations or result availability.
- Frontend: Created the required components for fetching and displaying these notifications to the patients.

Support Ticket Submission :

- Backend: Created the endpoints in SupportController.java and designed the service logic to receive and persist new support tickets submitted by the patient.
- Frontend: A SupportPage.jsx component was built, allowing patients to submit forms and view the status of their requests.

Administrator Portal Feature Implementation

I have completed the full end-to-end implementation of all key administrative and monitoring functionalities.

Admin/User Lifecycle Management :

- Back-end: Wrote robust controller and service logic in AdminController.java to give admins full CRUD capability over user accounts, including crucial role assignment.
- Frontend: Created the comprehensive UserManagement.jsx component in React to drive these administrative operations via a user-friendly interface.

Audit Logging and Monitoring:

- Backend: I created the AuditLog entity, implemented the AuditService.java to log vital actions across the system, and developed the secure retrieval endpoint in AdminController.java.
- Frontend: Designed and developed the AuditLogs.jsx component that showcases the fine-grained audit trail coming from the backend, thus ensuring accountability within all system activities.

Announcement Management:

- Backend: Developed AnnouncementController.java and related services, responsible for creating, editing, and publishing announcements across the site.
- Frontend: Designed the CreateAnnouncement.jsx component, providing a dedicated interface for administrators to manage announcements.

Support Ticket Management :

- Backend: Created the endpoints to be used in AdminController.java and services for fetching, assigning, and updating status regarding submitted support tickets.

- Frontend: Added the component SupportTicketsPage.jsx to allow administrators to efficiently manage the ticketing queue.

Final Integration and Quality Assurance

In the final stage, I concentrated on the unit and integration testing of all the developed modules against the RTM for functional completeness and strict enforcement of RBAC rules. The comprehensive quality assurance step ensures that the Patient and Admin Portals are fully cohesive, secure, and ready for deployment.

CHAPTER 03
IMPLEMENTATION & RESULTS

The implementation phase was a transition from the structured design artifacts (SRS, RTM, and Database Schema) into a fully functional software system. I was personally responsible for the end-to-end development of the core security framework, all Patient Portal features, and the complete Administrator Dashboard functionality. This chapter details the specific technological choices and the layered development process I followed.

3.1. TECHNOLOGY STACK & TOOLS

The technology stack was chosen based on requirements for high security, maintainability, and rapid feature development.

Component	Technology	Rationale and Use
Backend Framework	Spring Boot (Java 17)	Chosen for its robust, enterprise-level features, built-in security modules (Spring Security), and native support for transactional integrity and RESTful API development.
Build Tool	Maven	Used for dependency management and standardized build cycles, ensuring project consistency.

Database	MySQL 8	Selected for its reliability, relational integrity, and mature security features for persistent storage of PHI. Configured locally using secure JDBC parameters in application.properties.
Persistence	Spring Data JPA / Hibernate	Facilitated object-relational mapping (ORM), reducing boilerplate code for database operations and managing transactional boundaries.
Authentication	JSON Web Tokens (JWT) / Spring Security	Provides a stateless, secure method for authorizing API requests after initial authentication, crucial for enforcing the RBAC model.
Frontend Framework	ReactJS	Used a functional, component-based approach for highly modular and reusable UI elements, allowing rapid construction of complex, role-specific dashboards.

Table 3.1: Utilized Technology Stack

3.2. BACKEND IMPLEMENTATION

I started the project structure from Spring Initializer, including the necessary starters such as Web, JPA, Security, and MySQL, handling all the dependencies with Maven. Before working on any feature implementation, I took time to properly design the core domain model and security framework: I created an immutable enum Role.java that defines fixed roles like ADMIN, DOCTOR, and PATIENT, hence enforcing strong separation of roles throughout the system. The central User.java entity mapped to the database was set up with

essential security features, including a unique constraint on the email address; dedicated fields for password hashes; and Spring's @Enumerated(EnumType.STRING) for the role and account status to avoid unauthorized role or account status assigning and ensure data consistency.

I implemented strict controlled registration logic in the AuthService that makes initial user access according to the requirements of the stakeholders. Students will be strictly allowed Google OAuth flow, utilizing their university email domain for instant activation. Staff members, considering possible external affiliations, can utilize either Google OAuth or a manual login method. Most importantly, I designed the system so that any manual registration by staff, along with the registration of all non-patient provider roles Doctor, Pharmacist, Lab Technician automatically sets their AccountStatus to PENDING. The administrator account which plays an integral part of the system security gets created at the system launch after checking for an already existing administrator account.

The AuthController handles the complex login authentication, it manages password verification for the manual accounts, while integrating the dedicated GoogleTokenVerifierService for the validation of the external tokens before the user provisioning and issuance of the highly restricted JSON Web Tokens with a user's ID and role which is the cornerstone of our stateless security mechanism.

Among the key system requirements was to digitize the students' past medical data in such a way that it would be accessible with ease by the workers of medical care. I designed a medical form that can only be submitted once by students using the MedicalFormService and its corresponding endpoints to manage the initial collection of essential data.

I implemented the Patient Dashboard so that patients could get a quick understanding of their recent activities with the health center. I designed it to display the main patient details and recent activities, including appointments, prescriptions, and laboratory tests. The Patient Dashboard uses the same API endpoints used in the reports tab created in the PatientController to fetch this information.

The end-to-end Appointment Scheduling system involved the creation of transactional logic in the AppointmentService. I designed the service to manage complex slot validation, ensuring that overlap or double-bookings are rejected before persistence. The PatientController exposes a secure endpoint for submitting the appointment request.

I enforced strict API-level data segregation for access to medical records. The challenge was to ensure that the solution conformed with privacy standards without eliminating the possibility of useful patient visibility. My design decision was to implement a filter layer within the PatientService that explicitly screens the data before returning it via the PatientController. This filter ensures the patient only ever gets diagnosis notes, medical recommendations, and the lab report completion status and prevents the accidental exposure or download of full, raw medical records, which is critical for compliance and patient confidentiality.

I completed the System Notifications and Support Ticket Submission to round out the Patient Portal. NotificationService, through events inside the system, such as a lab result being ready, initiates an entry into the database. For the Support Ticket Submission, the SupportController handles the receipt and logging into the SupportTicket entity of the detailed support request.

The User Lifecycle Management feature in the Admin Portal was designed to meet a vital need for administration oversight for new staff and provider accounts. I wrote logic within AdminService, which manages the transition of the AccountStatus of pending users (manually registered staff and all non-patient roles). This is secured through endpoints inside AdminController, called by the frontend via specialized buttons on the UserManagement.jsx component to carry out approval and the assignment of roles. This workflow ensures that all elevated and manually created accounts undergo necessary human verification prior to accessing the live system.

I designed and implemented Audit Logging and Monitoring in the system. Throughout the backend, the AuditService is integrated to log each and every critical action like login success, account approval, and account disabling into the dedicated AuditLog entity stored in MySQL. The AdminController exposes a very secure endpoint for accessing this whole audit trail. The corresponding AuditLogs.jsx component in React includes filtering and searching, enabling the administrator to effectively inspect user activities for accountability and security reviews.

The Announcement Management and Support Ticket Management features implemented centralized, administrative communication and workflow. For announcements, I developed the full CRUD cycle via AnnouncementController that allows administrators to draft, edit,

and publish messages across the system. For support tickets, AdminController exposes endpoints for fetching the list of all active tickets, while SupportTicketService holds the logic to perform status updates, such as assigning a ticket or marking it resolved. The SupportTicketsPage.jsx component provides the actual visual interface used by the administrative team to effectively monitor and manage the entire ticket queue.

3.3. FRONTEND IMPLEMENTATION

I selected ReactJS for the Presentation Tier and have bootstrapped the project using the default Create React App scaffold, using the native Fetch API for all network communication in order to minimize dependencies on external libraries and to maintain low-level, manual control over request construction and security headers. For global state management, I make use of the native React Context. Upon a user's successful authentication, their role is extracted from the returned JWT and store it securely within a non-persistent React Context, all subsequent components can then rely on that in-memory context to conditionally render navigation, features, and content, hence rigorously enforcing the visual aspect of the Role-Based Access Control model. In terms of making all requests secure, I created a wrapper function in src/api/api.js that wraps the native Fetch call, manually injecting the JWT into the Authorization: Bearer header for every protected endpoint.

The most critical component was the LoginPage.jsx, where I used the native Fetch API for handling credential submission and exchanging them for Google OAuth tokens. In the design for the login screen, I led students explicitly to the Google OAuth option while presenting options for staff members on both Google OAuth and manual login. For the latter, specifically the Manual Login, I configured a raw Fetch POST request, manually building up a JSON body and setting the header Content-Type: application/json.

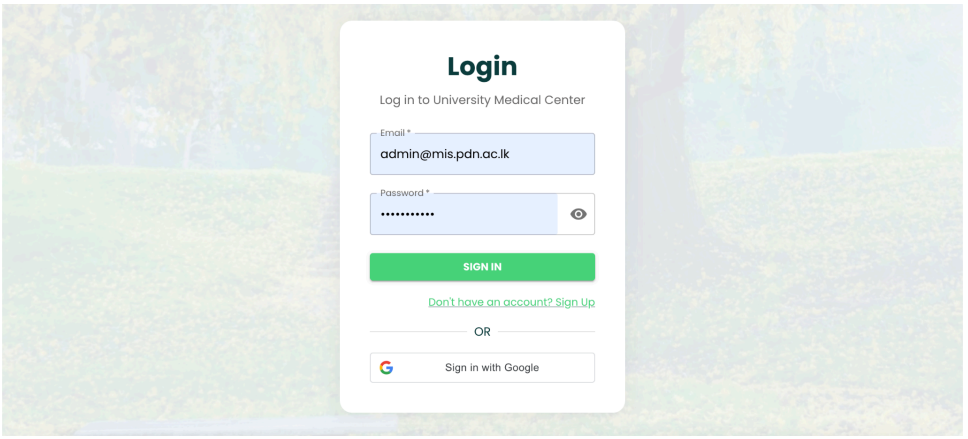


Figure 3.1: Login Page

The Patient Dashboard was architected for fast performance using the native Fetch API in order to make concurrent requests. First, I wrapped a Promise.all around three separate calls-for appointments, prescriptions, and lab status so the component fetches all the summary data it needs at once. I have designed the component to handle the visual loading state nicely while waiting for the native promise fulfillment for minimal perceived latency and to provide the patient with an immediate, current, actionable snapshot of their medical status from the moment they land on it.

I created the Medical Form component, MedicalForm.jsx, to collect critical medical information from newly registered patients. The component makes use of Fetch's POST method to handle the request payload that contains sensitive patient history, which I send to the appropriate backend service. This feature is essential for doctors to have access to patients' medical history before their engagement with the health center.

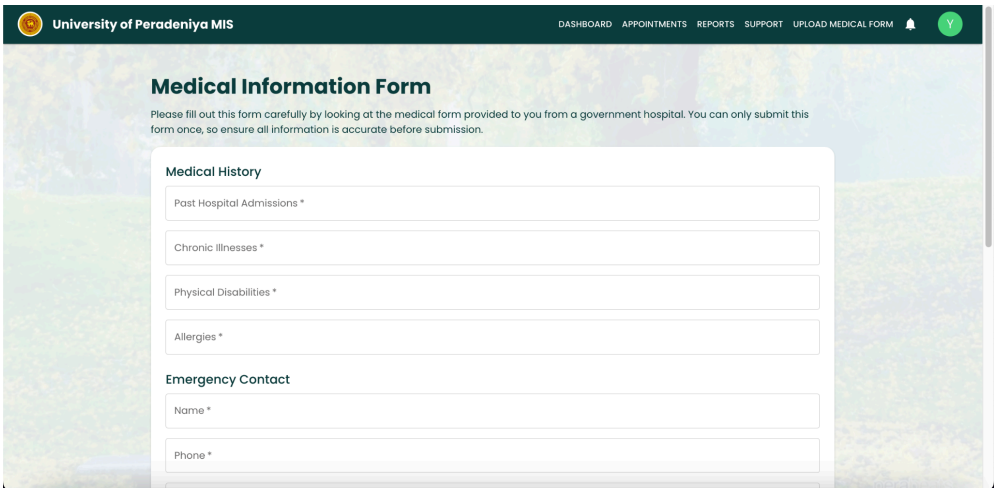


Figure 3.2: Medical Form for Students

In the ReportsTab.jsx for the Restricted Reports View, I integrated the native Fetch API to pull the patient's list of records from the backend endpoint, which was highly constrained. After parsing the response data inside the promise chain, the client-side implementation enables filtering and searching, enabling the user to query the results against the available list by date or diagnosis keyword. This component is purposely engineered to display only the authorized, restricted data set, including doctor diagnosis notes and recommendations, while strictly adhering to the requirements of privacy by excluding all raw PII and clinical data.

To complete the patient's transactional toolset, I implemented the necessary components for System Notifications and Support Ticket Submission. The notification system uses polling to fetch the latest information for each user. Repeated native Fetch calls are used in the NotificationBell.jsx to query the notification API to render unread alerts in a responsive dropdown. The SupportPage.jsx provides the interface for submitting a new ticket via a Fetch POST request, and then fetching the status of previously submitted tickets for tracking.

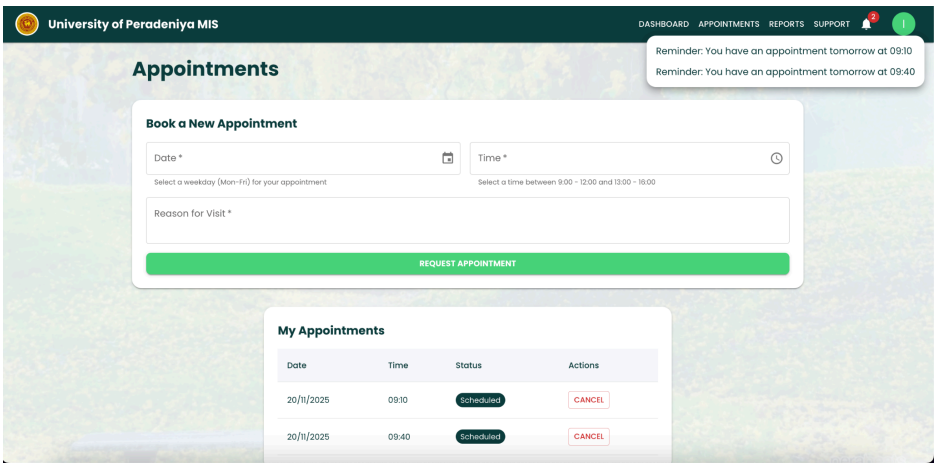


Figure 3.3: Notification System

In the UserManagement.jsx component, I developed the functionality of User Lifecycle Management for the administrative workflow. It fetches the list of users and integrates functionality buttons that trigger specific native Fetch PUT requests to the AdminController endpoints to manage account approval and role assignments. I also created client-side logic that allowed filtering by user, action type, and date range without the use of an external complex data library to allow the administrator to efficiently search and view the history.

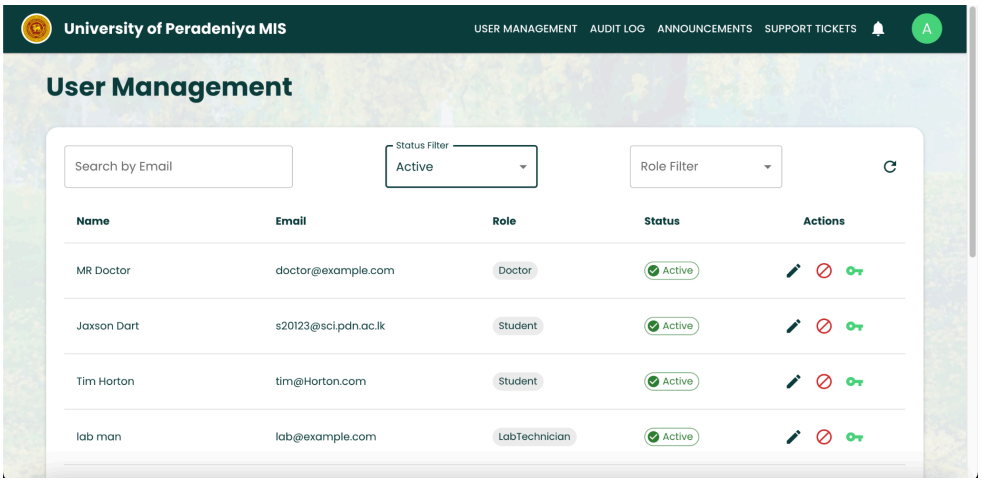


Figure 3.4: User Management Page

I implemented the Audit Logging and Monitoring feature in the AuditLogs.jsx component to provide administrators with visibility into system accountability. This component fetches the entire security audit trail from the dedicated administrative backend endpoint. To make the large volume of data manageable, I designed advanced client-side filtering capabilities, enabling administrators to efficiently search and analyze log events by user ID, specific action type, or custom date ranges.

CreateAnnouncement.jsx also depends on the native Fetch API for Announcement Management, as does Support Ticket Management in SupportTicketsPage.jsx. In the support component, I have set up Fetch requests to pull the global ticket queue and fired off PUT requests to change a ticket's assigned staff member or its resolution status, ensuring that all administrative actions against the backend are explicitly and securely managed.

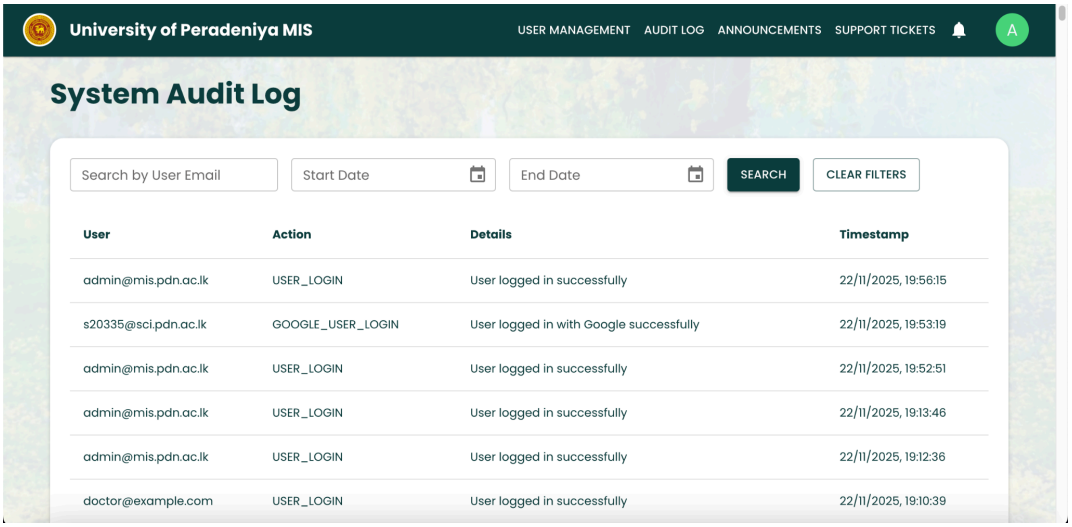


Figure 3.5: Audit Logs Page

3.4. INTEGRATION OF BACKEND AND FRONTEND

This phase in integration ensured smooth, secure, and reliable communication between the Presentation Tier (ReactJS) and the Application Tier (Spring Boot).

CORS Configuration: I then allowed Cross-Origin Resource Sharing (CORS) in the Spring Security filter chain to permit requests from the origin URL of the React frontend.

Secure Data Transmission: I wrapped the native Fetch API call with a custom function that

automatically attached the in-memory JWT to the header Authorization: Bearer <token> for every API request. This strict adherence to header injection ensures all protected endpoints are gated by the necessary token validation handled by Spring Security's filter chain.

End-to-End Flow Validation

Authentication and Approval Workflow: I have thoroughly tested the complicated registration flow: a student upon login with Google is immediately set to ACTIVE, a manually registered staff member or a medical care worker is set to PENDING.

Support Ticket Resolution Flow: Verified the following continuous flow: A patient creates a ticket - SupportPage.jsx using Fetch POST, Admin navigates to ticket in SupportTicketsPage.jsx using Fetch GET, then updates status, e.g., to RESOLVED using Fetch PUT, which reflects back on Patient's SupportPage.jsx.

CHAPTER 04
CONCLUSIONS

4.1. CONCLUSION

The Medical Information System project represents a crucial modernization effort, successfully turning what was a fragmented, paper-based workflow at the University Health Centre into a cohesive and secure digital platform. Initial challenges in slow patient access, high risk of data errors, and the complete absence of real-time administrative oversight were systematically overcome through the application of modern software engineering principles with secure, layered architecture.

The outcome was a dynamic, role-based user experience for all constituents. Now patients have easier navigation through the entire digital journey, from appointment management and filling in the necessary medical history forms to securely accessing limited health summaries through the portal. At the same time, administrative staff and security personnel are given unmatched real-time tools to manage user lifecycles, post community announcements, and monitor accountability continuously with the Audit Log.

The main contribution I made was in constructing the essential security and patient interaction base. I was responsible, individually, for designing and implementing the complete Role-Based Access Control system and building the full end-to-end functionality for both the Patient Portal and the Administrator Portal. This involved rigorous API development using Spring Boot and frontend implementation using ReactJS with

management of all data flow through the native Fetch API. Amongst the really tough but rewarding technical aspects I had to face was precisely the setting up of seamless, secure communication and enforcing strict data filters for the patient reports.

In this process, the iterative development methodology, along with constant checking against the Stakeholder Analysis, was very fundamental in ensuring that the system met both functional needs and critical security mandates. The project successfully delivers a robust, near-commercial grade application. This meets not only the immediate needs of the Health Centre but brings a fundamental enhancement in operational efficiency by drastically reducing the occurrence of manual errors, improving the integrity of data, and significantly securing sensitive patient information.

4.2. LIMITATIONS

During the making of the MIS, a variety of inherent limitations driven by technology stack choices and project scope were realized, which may impact long-term scalability and specialized performance.

One critical constraint is the vertical scaling limitations of MySQL as the main database system. While MySQL is highly dependable for transactional operations, it does not match distributed database solutions in terms of scaling horizontally for potentially massive data sets, which can grow over many decades as the population of universities does. This will eventually require more complex and costly clustering configurations in the future. Second, the required PHI Encryption Service brings slight, non-avoidable extra performance overhead: any read and write access to sensitive data has to execute the AES encryption or the decrypt routine, respectively, hence affecting database throughput at extremely high concurrency. This was a necessary trade-off to achieve the required level of security of the data.

Another critical constraint is the nature of the technology stack chosen, the steep learning curve associated with some of the more advanced components. The complex logic in Spring Boot (Java) and asynchronous state management in ReactJS require professional assistance. While this complexity is crucial for a robust solution, it negatively impacts the ease with which new teams can be on-boarded for maintenance and may also raise ongoing support costs.

Other aspects that appeared deal with scalability and integration complexity. Maintaining a system built around a dedicated Java Spring Boot security layer, custom JWT filters, and JPA converters requires very specific knowledge; this increases the complexity and cost of their future maintenance compared to simpler architectures. Also, as much as the system would be very secure, the fact that an explicit decision to leverage solely the native Fetch API is made prevents taking advantage of off-the-shelf features like automatic request cancellation or built-in progress tracking provided by client libraries and would require more complex custom coding for advanced performance optimizations on the frontend.

Finally, because of project scoping, the system utilizes manual verification of provider accounts, creating a human bottleneck. While necessary for security, this manual transition of status from PENDING to ACTIVE creates a bottle-neck.

4.3. FUTURE WORKS & RECOMMENDATIONS

I strongly recommend the next set of strategic enhancements aimed at future-proofing the MIS through the integration of advanced technologies that further automate the system.

- AI diagnostics and support would significantly enhance this system in terms of its clinical value. That will be made possible by the development of aggregate analysis tools-anonymized patient data for the earlier detection of outbreaks of diseases-or a decision-support module flagging possible drug interactions or other dosing issues from patient records, thus enhancing the quality of care.

- Blockchain Integration for Record Security: This should involve the use of Blockchain technology to secure the Audit Log and patient consent records, ensuring immutable auditability and allowing safe data sharing with third-party providers. In this case, Blockchain is a kind of distributed ledger technology that would supply an unparalleled level of transparency and tamper-proof verification of critical transactions to meet future regulatory demands.

- Integration with the Existing University System: In development, strong API links that will be integrated with the University's central Human Resources and Student Information Systems are extremely important. The integration of initial user

provisioning would automate the process, staff would no longer have to manually review user accounts.

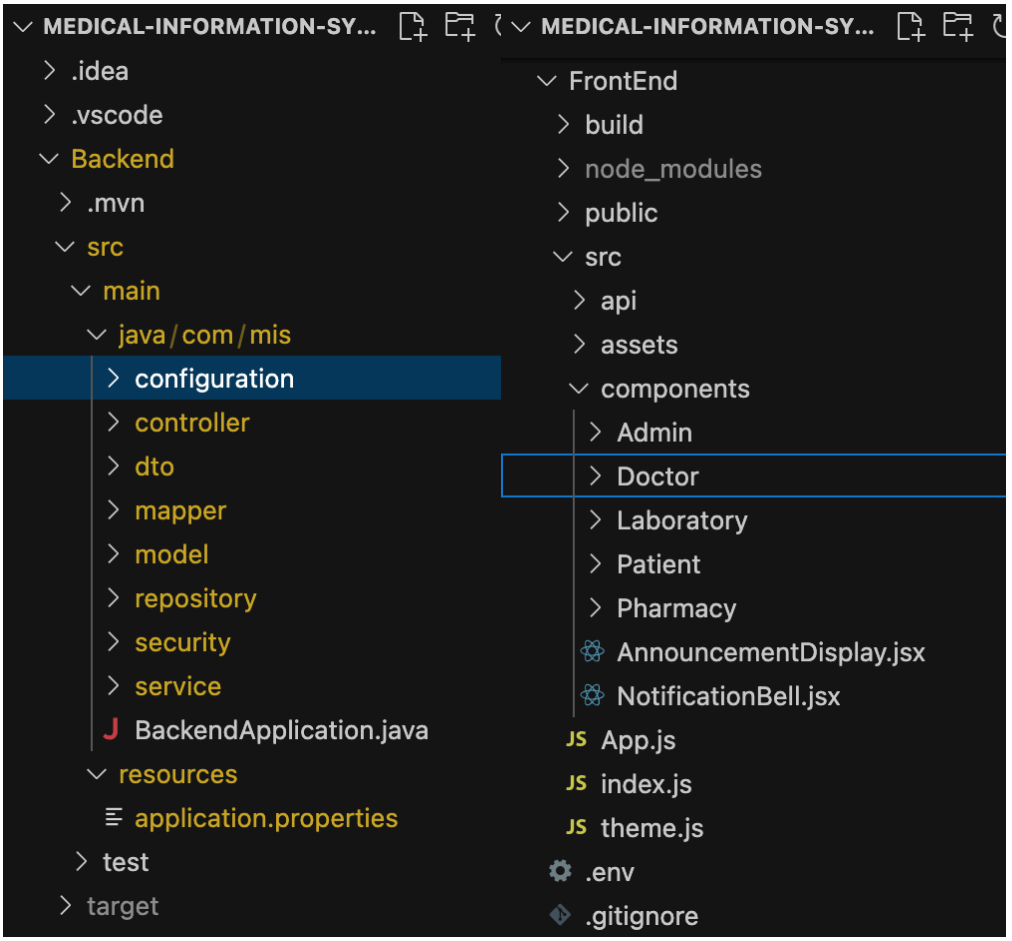
- Advanced Data Analytics Platform: Building upon the currently implemented Audit Log, further work should create a dedicated analytics platform that analyzes usage patterns, patient flow, and resource utilization. This will enable Health Centre management to understand where their bottlenecks are, predict resource requirements such as pharmacy stock based on seasonal diagnoses-and optimize staff deployment in data-informed ways.

- The development of a mobile application for core patient features and the integration of a secure, video-consultation module to enable telehealth services constitute the key usability enhancement. This will significantly extend access and convenience for the Health Centre's services through routine follow-ups to the wider university community.

APPENDICES

Appendix A : Project Structure

Frontend, Backend File Structure



Backend Frontend