**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**



**A Project Report**

**On**

**“KU Health Care”**

**[Code No: COMP 207]**

**(For partial fulfillment of II Year/ II Semester in Computer Engineering)**

**Submitted by**

**Saimon Neupane (27)**

**Risham Raj Byahut (43)**

**Parikchit Sen (47)**

**Keshav Raj Sharma (48)**

**Prabesh Sharma (49)**

**Submitted to**

**The Project Coordinator**

**Department of Computer Science and Engineering**

**Submission Date: 2025/08/04**

**Bona fide Certificate**

**This project work on**

**“KU Health-Care”**

**is the bona fide work of**

**“Saimon Neupane” (27)**

**“Risham Raj Byahut” (43)**

**“Parikchit Sen” (47)**

**“Keshav Raj Sharma” (48)**

**“Prabesh Sharma” (49)**

**who carried out the project work under my supervision.**

**Project Supervisor**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Dr. Rabindra Bista**

**Associate Professor,**

**Department of Computer Science and Engineering**

**Date: 2025/08/04**

# Acknowledgement

We would like to express our heartfelt gratitude to our Project Supervisor, Dr. Rabindra Bista, for his constant support, thoughtful guidance, and valuable insights throughout the development of the KU Health Care management system. His mentorship played a vital role in shaping our approach to solving complex real-world problems and in ensuring the successful execution of our project.

We are also sincerely thankful to the Department of Computer Science and Engineering (DoCSE) for providing us with the opportunity and platform to work on this impactful project. The development of KU Health Care has been a transformative experience—allowing us to explore concepts in healthcare technology, database design, and user-centric software development.

This project has significantly enhanced our technical and analytical skills and strengthened our ability to work as a cohesive team. From designing structured information flows to integrating automated communication systems, every phase of this journey has deepened our understanding of how technology can improve healthcare service delivery.

We deeply appreciate the support, encouragement, and resources that made this project possible and meaningful.

# Abstract

# KU Health Care is a comprehensive healthcare management system developed to enhance efficiency and coordination in clinical environments. In response to the growing need for streamlined patient care and improved administrative workflows, this platform addresses critical challenges such as appointment scheduling, patient tracking, and interdepartmental communication. It achieves this through a centralized database architecture that connects patients, doctors, receptionists, and laboratory staff, enabling structured information flow and automated notifications. The system includes features like patient registration, appointment allocation, lab test requisition, and prescription management, effectively reducing bottlenecks in care delivery. By supporting both clinical decisions and administrative processes, KU Health Care fosters secure and collaborative interactions among stakeholders, improving patient experience and operational performance in medical institutions.

# **Keywords**: *Healthcare Management, Patient Registration, Appointment Scheduling, Laboratory Integration*

Table of Contents

[Acknowledgement ii](#_Toc205286100)

[Abstract iii](#_Toc205286101)

[List of Figures vi](#_Toc205286102)

[Acronyms/ Abbreviation viii](#_Toc205286103)

[Chapter 1: Introduction 1](#_Toc205286104)

[1.1 Background 1](#_Toc205286105)

[1.2 Objectives 2](#_Toc205286106)

[1.3 Motivation and Significance 3](#_Toc205286107)

[Chapter 2: Related Works 4](#_Toc205286108)

[2.1 Epic Systems 4](#_Toc205286109)

[2.2 OpenMRS 5](#_Toc205286110)

[2.3 Bahmni 6](#_Toc205286111)

[Chapter 3: Design and Implementation 7](#_Toc205286112)

[3.1 System Requirement Specifications 13](#_Toc205286113)

[3.1.1 Functional Requirements 13](#_Toc205286114)

[3.1.2 Non-Functional Requirements 14](#_Toc205286115)

[Chapter 4: Working Mechanism 15](#_Toc205286116)

[4.1 Frontend Interface 15](#_Toc205286117)

[4.2 Backend Architecture 15](#_Toc205286118)

[4.2.1 Database Models (via Prisma) 15](#_Toc205286119)

[4.2.2 API Routes (via Express.js) 17](#_Toc205286120)

[4.3 Role-Based Interaction Flow 17](#_Toc205286121)

[4.3.1 User Registration and Authentication 17](#_Toc205286122)

[4.3.2 Appointment Management 17](#_Toc205286123)

[4.3.3 Lab Test Workflow 18](#_Toc205286124)

[4.3.4 Notification System 18](#_Toc205286125)

[4.4 Security Measures 18](#_Toc205286126)

[4.5 Scalability & Performance 18](#_Toc205286127)

[Chapter 5: Discussion on Achievement 19](#_Toc205286128)

[5.1 Features 19](#_Toc205286129)

[5.1.1 Doctor-Facing Features 19](#_Toc205286130)

[5.1.2 Receptionist-Facing Features 20](#_Toc205286131)

[5.1.3 Admin-Facing Features 20](#_Toc205286132)

[5.1.4 Lab Assistant-Facing Features 21](#_Toc205286133)

[Chapter 6: Conclusion & Recommendations 22](#_Toc205286134)

[6.1 Limitations 22](#_Toc205286135)

[6.1.1 Lack of Real-Time Communication 22](#_Toc205286136)

[6.1.2 No Integration with External Services 22](#_Toc205286137)

[6.1.3 Manual Lab Test Entry 22](#_Toc205286138)

[6.1.4 Limited Mobile Experience 23](#_Toc205286139)

[6.2 Future Enhancement 23](#_Toc205286140)

[6.2.1 Telemedicine Support 23](#_Toc205286141)

[6.2.2 Mobile Application Development 23](#_Toc205286142)

[6.2.3 Online Payment Integration 23](#_Toc205286143)

[6.2.4 Automated Reporting and Analytics 23](#_Toc205286144)

[References 24](#_Toc205286145)

[Appendix 25](#_Toc205286146)

# List of Figures

[Figure 2.1. 1: Epic System Software 4](#_Toc205282954)

[Figure 2.2. 1: OpenMRS Online Demo 5](#_Toc205159739)

[Figure 2.3. 1 Bahmni Online Demo 6](#_Toc205159743)

[Figure 3. 1: Use Case: Receptionist 8](#_Toc205283136)

[Figure 3. 2: Use Case: Admin 9](#_Toc205283137)

[Figure 3. 3: Use Case: Doctor 9](#_Toc205283138)

[Figure 3. 4: Use Case: Lab Assistant 10](#_Toc205283139)

[Figure 3. 5: Context Diagram (DFD- Level 0) 10](#_Toc205283140)

[Figure 3. 6: Dataflow Diagram (Level 1) 11](#_Toc205283141)

[Figure 3. 7: Dataflow Diagram (Level 2) 11](#_Toc205283142)

[Figure 3. 8: Entity Relationship Diagram (ERD) 12](#_Toc205283143)

[Figure i: Gantt Chart 25](#_Toc205283313)

[Figure ii: Work Division 26](#_Toc205283314)

[Figure iii: KU Health Care Login page 26](#_Toc205283315)

[Figure iv: Admin Dashboard 27](#_Toc205283316)

[Figure v: List of Doctors Page 27](#_Toc205283317)

[Figure vi: New Doctor Add Page 28](#_Toc205283318)

[Figure vii: Lab Assistant Page 28](#_Toc205283319)

[Figure viii: Add New Lab Assistant Page 29](#_Toc205283320)

[Figure ix: List of Receptionist Page 29](#_Toc205283321)

[Figure x: Patient List Page 30](#_Toc205283322)

[Figure xi: Receptionist Dashboard 30](#_Toc205283323)

[Figure xii: List of Doctors and Patients Appointed 31](#_Toc205283324)

[Figure xiii: Lab Assistant Page 31](#_Toc205283325)

[Figure xiv: Appointment List Page 32](#_Toc205283326)

# Acronyms/ Abbreviation

**API**: Application Programming Interface

**SQL**: Standard Query Language

**REST**: Representational State Transfer

**UI**: User Interface

**HTTP**: Hypertext Transfer Protocol

**JWT**: JavaScript Web Token

**SSD**: Solid State Drive

**SQL**: Standard Query Language

**ORM**: Relational Mapping

**OS**: Operating System

**DFD**: Data Flow Diagram

**ERD**: Entity Relationship Diagram

# Chapter 1: Introduction

KU Health Care is a comprehensive healthcare management system designed to streamline patient care and administrative workflows within clinical environments. As healthcare institutions face increasing demands for efficiency and coordination across departments, this integrated platform addresses the critical challenges of patient management and appointment scheduling. This application addresses these needs by offering a user-friendly platform that connects patients, receptionists, doctors, and laboratory staff through automated notification systems and structured information pathways, enabling healthcare providers to deliver better care while reducing the administrative burden typically involved in healthcare processes.

## Background

In recent years, the rapid advancement of technology has significantly transformed various sectors, with healthcare being no exception. Healthcare institutions are increasingly adopting digital solutions to enhance operational efficiency and improve patient care. Alongside these advancements, the need for seamless coordination among different departments has become more critical than ever. Managing patient information, scheduling appointments, and ensuring smooth interdepartmental communication present persistent challenges in many healthcare settings.

While many institutions have recognized these challenges and taken steps toward digital transformation, most have implemented partial solutions. Separate systems are often deployed for tasks such as patient registration, appointment booking, and laboratory management. However, these systems typically operate in isolation, leading to fragmented workflows, inefficiencies, and communication gaps between departments.

Despite notable progress in healthcare digitization, current solutions in the market tend to focus on specific functionalities rather than offering an integrated approach. Many platforms overlook essential features such as centralized patient data, automated notifications, or streamlined laboratory test management. As a result, healthcare professionals often struggle to maintain efficient workflows and deliver high-quality care.

KU Health Care aims to resolve these inefficiencies by offering an all-in-one platform where stakeholders can communicate and operate using the same structured and secure information ecosystem. This will not only enhance the patient experience but also empower healthcare professionals with the tools needed to provide accurate and timely care.

## Objectives

The primary goal of this project is to design and implement a digital healthcare management system that supports both clinical and administrative workflows through an integrated web-based application. The specific objectives of the system are outlined as follows:

* To create a reliable healthcare management system dedicated to improving

patient care through efficient administrative processes

* To facilitate seamless communication between patients, receptionists, doctors, and laboratory staff via an automated notification system
* To provide reliability and accuracy by maintaining complete patient information, appointment details, and laboratory test data
* To develop a user-friendly healthcare platform by creating an intuitive, easy-to-use web application that allows efficient patient registration, appointment scheduling, and doctor assignment.

## Motivation and Significance

The motivation behind KU Health Care arises from the inefficiencies in traditional hospital workflows, where different departments often work in isolation, leading to delays, communication gaps, and poor patient experience. Existing systems are either limited in scope or lack integration, making healthcare management cumbersome.

Our project aims to address these challenges by developing a centralized platform that connects patients, receptionists, doctors, and laboratory staff. Through automation and streamlined communication, the system enhances coordination, reduces administrative burden, and improves the overall quality of patient care.

This solution is significant as it promotes a more organized, efficient, and tech-driven approach to healthcare, especially in environments still reliant on partial or manual systems.

# Chapter 2: Related Works

## 2.1 Epic Systems

Epic provides a comprehensive healthcare management platform with integrated workflows across departments, similar to your centralized approach. Its strength lies in connecting patient records seamlessly between reception, clinical staff, and laboratories through a unified database architecture.

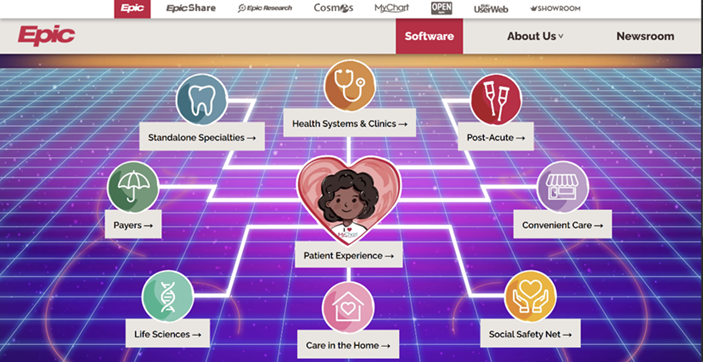


Figure 2.1. : Epic System Software

## 2.2 OpenMRS

OpenMRS is an open-source electronic medical record system designed primarily for use in resource-constrained environments. It offers modular architecture that enables customization according to institutional needs, making it suitable for small clinics, research environments, and university health centers. OpenMRS supports patient registration, clinical observations, encounter tracking, and data reporting, which aligns well with KU Health Care’s goals of building a flexible and scalable health management platform tailored to a specific community.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2.2. : OpenMRS Online Demo

## 2.3 Bahmni

Bahmni is an open-source hospital information system built on top of OpenMRS, designed specifically for low-resource healthcare environments. It integrates electronic medical records, laboratory management, pharmacy, billing, and radiology into a unified platform. Bahmni provides an online demo environment that allows users to explore its full suite of clinical workflows in a real-time setting. Its modular and user-friendly design makes it a valuable reference for developing scalable and comprehensive health care solutions such as KU Health Care.

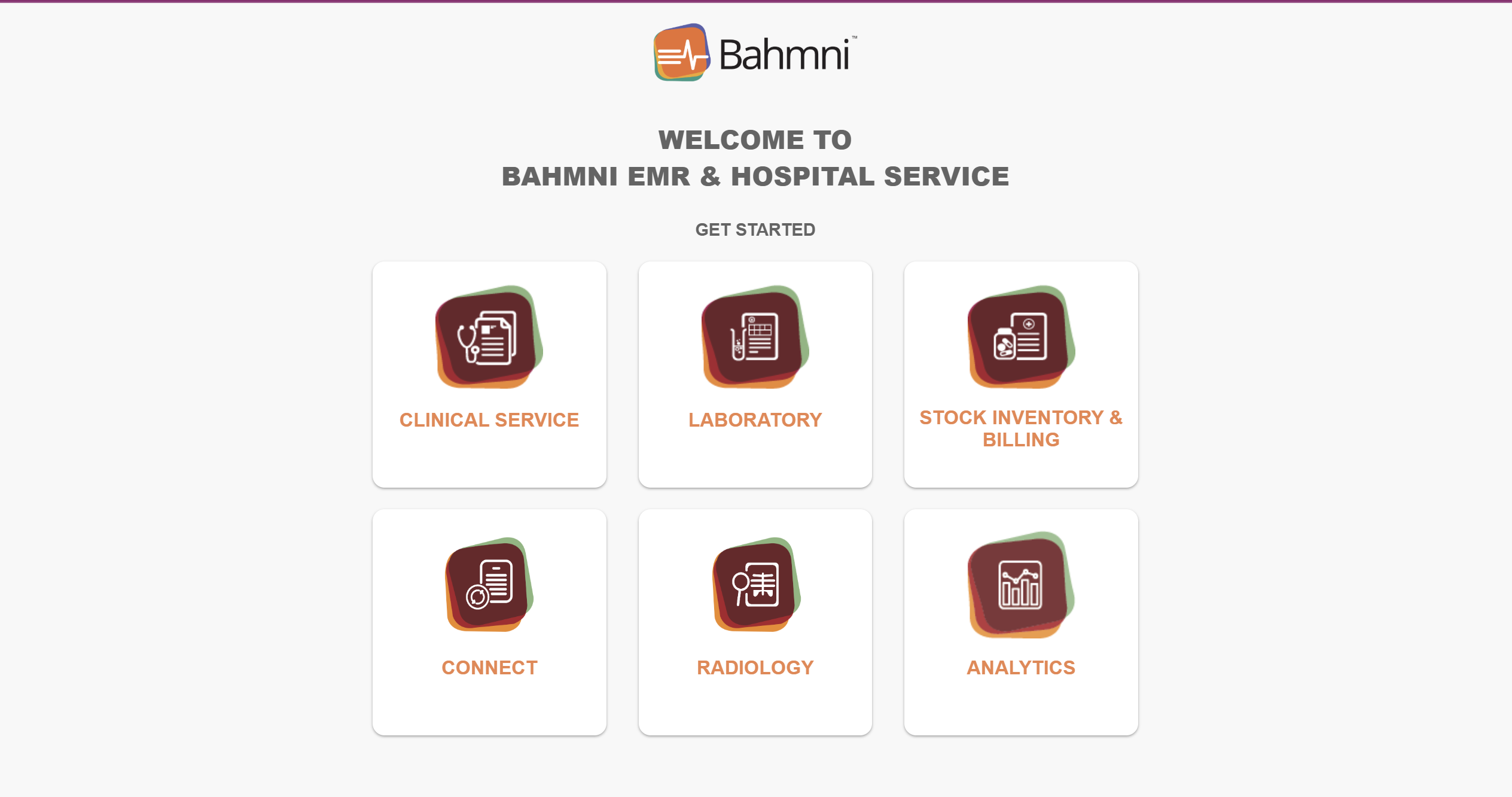


Figure 2.3. Bahmni Online Demo

# Chapter 3: Design and Implementation

This chapter outlines the step-by-step process and methodologies followed throughout the development of the KU Health Care project.

The following are the steps we implemented to complete this project:

**Step 1. Project Planning:**

* The project was planned by dividing tasks among team members according to their expertise in frontend and backend development.
* Clear milestones and deadlines were set to ensure timely progress and continuous collaboration.

**Step 2. System Design and Architecture:**

* The system architecture was designed after gathering and finalizing the project requirements. It includes defining the database schema, user interface components, and the overall modular structure of the application.
* Data flow and interactions between components, such as patient management, appointment scheduling, and medical records handling, were mapped out through flowcharts and system diagrams.

**Step 3. Development and Coding:**

* The frontend was developed using React, leveraging TypeScriptto create a responsive and user-friendly interface.
* For the backend and database, PostgreSQL was used to implement a robust relational database system to manage patient data securely and efficiently.
* API endpoints and business logic were built to support core functionalities such as user authentication, appointment management, and data retrieval.

**Step 4. Testing and Debugging:**

* Comprehensive testing was performed, including unit testing of individual components, and integration testing to verify interactions between modules.
* Bugs and issues were tracked continuously, with fixes applied iteratively to maintain system stability and performance.

**Step 5. Documentation:**

* Detailed documentation was prepared, including user manuals to guide end-users and technical documentation to support future development and maintenance.

Throughout the project, continuous communication and collaboration among the development team were maintained to align with project goals and deliver a reliable healthcare platform tailored to the Kathmandu University community.

**A diagram of a person's work flow

AI-generated content may be incorrect.**

Figure 3. : Use Case: Receptionist

A diagram of a medical procedure

AI-generated content may be incorrect.

Figure 3. 2: Use Case: Admin

A diagram of a process

AI-generated content may be incorrect.

Figure 3. 3: Use Case: Doctor

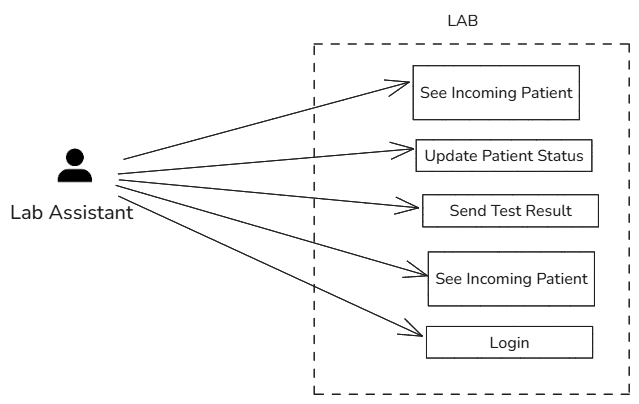


Figure 3. 4: Use Case: Lab Assistant

A diagram of a health care system

AI-generated content may be incorrect.

Figure 3. 5: Context Diagram (DFD- Level 0)

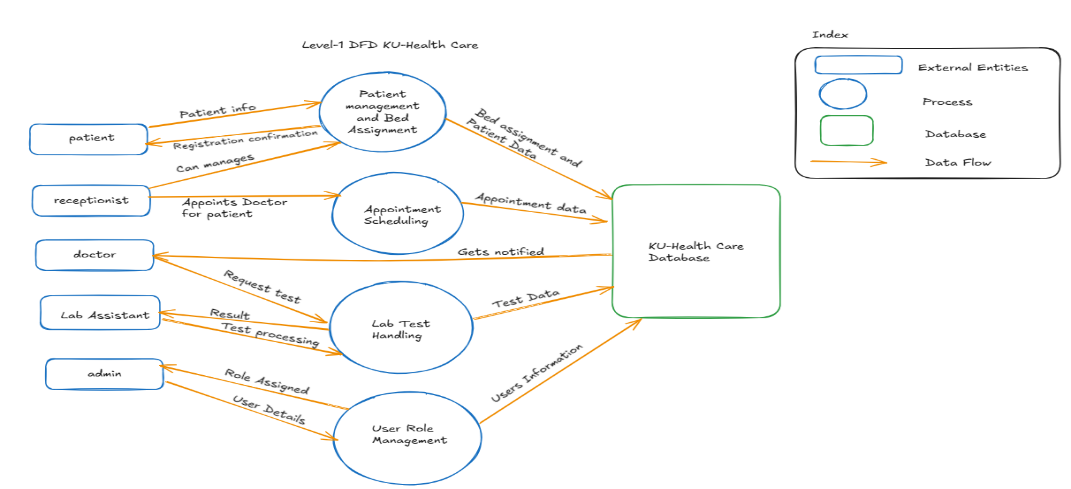
**

Figure 3. : Dataflow Diagram (Level 1)

**A diagram of a data flow

AI-generated content may be incorrect.**

Figure 3. : Dataflow Diagram (Level 2)

The KU-Health Care System is modeled using Context, Level-1, and Level-2 Data Flow Diagrams (DFDs). The Context DFD presents a high level view of interactions between the system and external entities. The Level-1 DFD in *Figure 3.6* breaks the system into core processes such as patient management, appointment scheduling, lab tests, and user role management. The Level-2 DFD in *Figure 3.7* further decomposes these processes into more detailed sub-processes to clarify internal data flow and enhance system understanding.

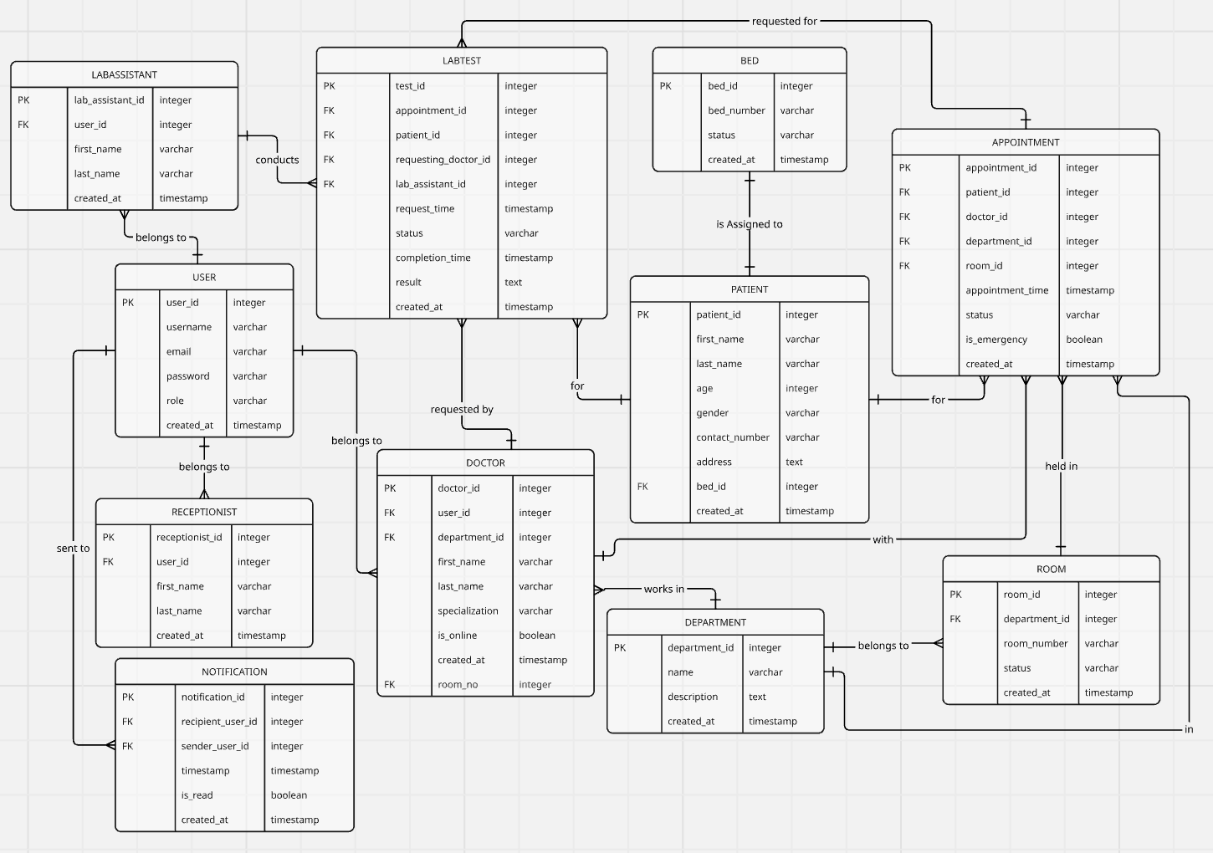


Figure 3. : Entity Relationship Diagram (ERD)

This Entity Relationship Diagram represents a comprehensive hospital management system database that manages patient appointments, laboratory operations, and hospital resources. The system enables patients to schedule appointments with doctors in specific rooms within departments, while supporting a complete lab testing workflow where doctors can request tests that are conducted by lab assistants. The database includes role-based user management for patients, doctors, lab assistants, and receptionists, tracks bed assignments for patient admissions, and incorporates a notification system to facilitate communication between different users. Key relationships connect patients to doctors through appointments, link doctors' test requests to lab assistants and patients, and organize hospital infrastructure through departments, rooms, and bed management.

## 3.1 System Requirement Specifications

### 3.1.1 Functional Requirements

1. User Interface (UI) Interaction

* The system should provide a responsive and user-friendly interface for patients, doctors, receptionists, and laboratory staff.
* The UI should be built using **React.js** to enable component-based dynamic rendering and real-time interaction.

2. API and Server-side Logic

* The system shall expose RESTful APIs developed with Express.js to handle business logic, user authentication, appointment handling, and lab report processing.
* The server ensures secure and efficient communication between the frontend and the database.

3. Database Management

* The system store and retrieve structured healthcare data using PostgreSQL.
* It shall manage patient records, appointments, prescriptions, and laboratory tests with relational integrity and support for complex queries.

4. Authentication and Authorization

* The system shall support role-based access control, restricting access to sensitive features and data based on user roles (e.g., patient, doctor, receptionist).

5. Automated Notifications

* The system shall send real-time notifications (e.g., appointment confirmation, prescription availability, test result updates) to relevant users.

### 3.1.2 Non-Functional Requirements

1. Performance Requirements

* The system shall respond to user interactions within 2 seconds under standard network conditions.
* The backend APIs shall handle a minimum of 100 concurrent requests efficiently.

2. Scalability

* The system architecture shall support horizontal scaling to accommodate an increasing number of users and records over time

3. Portability and Accessibility

* The system should be accessible from various platforms, including desktop computers, laptops, and smartphones.
* The UI shall be compatible with modern browsers and screen sizes using responsive design principles.

4. Security

* The system shall ensure data protection through secure authentication protocols, encrypted data storage (where necessary), and prevention of common vulnerabilities (e.g., SQL Injection, XSS).

5. Maintainability

* The system shall follow modular coding practices and maintain proper documentation to facilitate easy updates and debugging.

6. Minimum Hardware Requirements

* Processor: Intel Core i5 or equivalent
* RAM: 8GB
* Storage: 256 GB SSD
* Graphics Card: Integrated Intel UHD Graphics
* Operating System: Windows 8, macOS or Linux Based OS

# Chapter 4: Working Mechanism

The working mechanism of the KU Health Care system is structured into distinct yet interconnected layers involving frontend interaction, backend processing, and secure data management. The system facilitates seamless communication among users—patients, doctors, receptionists, and lab assistants—via automated workflows and role-based access.

## 4.1 Frontend Interface

* User Interface: The frontend of KU Health Care is developed using React**.**js, which enables a component-based, interactive experience. Different user types—such as patients, doctors, receptionists, and lab assistants—interact through intuitive modules such as appointment booking, patient registration, and lab test access. Conditional rendering based on roles ensures that each user sees functionalities relevant to them.
* Responsive Design: The application layout adapts to multiple screen sizes, offering full accessibility across desktop, tablet, and mobile devices. This ensures convenience for healthcare staff on-site and patients accessing the platform remotely.

## 4.2 Backend Architecture

The backend of KU Health Care is powered by Express.js, interfacing with a PostgreSQL database via Prisma ORM. The architecture is built with modular API routes and a normalized data model to ensure clarity and maintainability.

### 4.2.1 Database Models (via Prisma)

The system includes the following key entities and relationships:

* User Model: Centralized table for all authenticated users. Includes *user\_id,* username, email, password, and role. Each user may be associated with a Doctor, Receptionist, or *Lab Assistant* role.
* Doctor**,** Receptionist, Lab Assistant Models: Role-specific user extensions storing personal information and linked to a common User.

* Patient Model: Stores patient data including demographic info, contact, and address. Each patient can be linked to multiple Appointments and Lab Tests and optionally assigned a Bed.
* Appointment Mode**l**: Manages appointment bookings, capturing relations between Patient, Doctor, Room, and Optional Department. Fields include *appointment\_time*, status, and *is\_emergency*.
* Department and Room Models: Define physical and organizational structure. Doctors and appointments are associated with specific departments and rooms.
* LabTest Model: Manages lab test workflows. Linked to Appointment, Patient, Doctor, and optionally a LabAssistant. Includes status, result, and completion time tracking.
* Notification Model: Supports in-app messaging with fields like *sender\_user\_id,* *recipient\_user\_id,* *is\_read*, and *created\_at*. Help inform users of appointments, test results, or administrative actions.
* Bed Model: Tracks available and assigned hospital beds. Each bed can be assigned to multiple patients (over time).

### 4.2.2 API Routes (via Express.js)

The Express backend defines RESTful endpoints to manage data flow and business logic:

* Authentication Routes: Handle user signup, login, and secure session management using JWT.
* Appointment Routes: Enable patients to book appointments, and doctors to accept or review them.
* Patient and Record Routes: Allow doctors to update medical records; patients can view their history.
* Lab Test Routes: Manage the lab test lifecycle from request to result entry by lab assistants.
* Notification Routes: Send and retrieve notifications between users.

## 4.3 Role-Based Interaction Flow

### 4.3.1 User Registration and Authentication

* Users sign up with credentials and a selected role.
* JWT-based authentication secures user sessions and ensures proper authorization.

### 4.3.2 Appointment Management

* Patients browse available doctors and book appointments.
* Doctors view appointment requests in their dashboard and mark them as complete.
* Receptionists manage department and room allocations.

### 4.3.3 Lab Test Workflow

* Doctors request lab tests during appointments.
* Lab assistants see test assignments and enter results upon completion.
* Patients can access their test results securely.

### 4.3.4 Notification System

* Users receive real-time alerts for appointments, lab test updates, and administrative messages

## 4.4 Security Measures

* JWT Authentication: Ensures secure and stateless login sessions.
* Password Hashing: All passwords are encrypted before storage.
* Input Validation: All endpoints include validation to prevent

SQL injection and other attacks.

* Role-Based Access Control: System functionality is strictly segmented.

## 4.5 Scalability & Performance

* Efficient Querying: Prisma-generated SQL queries are optimized, and PostgreSQL indexing supports fast lookups.
* Modular Architecture: The separation of models, routes, and services enables easier scaling and feature integration.
* RESTful Design: Stateless APIs enable horizontal scalability in deployment scenarios.

Chapter 5: Discussion on Achievement

The KU HealthCare project was a valuable learning experience for our team. Working with technologies like React, Express, TypeScript, PostgreSQL, and Prisma, we faced several challenges especially while managing API integrations, handling relational data, and implementing role-based access.

We tackled issues through consistent research, team coordination, and trial-and-error. Designing a smooth appointment system and secure user management required careful planning and logic implementation.

Analyzing existing health platforms helped us create a simple, user-friendly interface. Overall, this project enhanced our practical skills and improved our ability to work collaboratively under real-world constraints.

## 5.1 Features

After successfully developing the KU Health Care system, the following role-based features were implemented to ensure seamless, efficient, and secure experience for all stakeholders:

### 5.1.1 Doctor-Facing Features

1. Appointment Overview

* Doctors can view a comprehensive list of upcoming and past appointments.
* Each entry displays essential patient details, appointment time, and status (e.g., confirmed, completed, or pending).

1. Lab Test Requests

* Doctors can request laboratory tests directly from the appointment interface.
* The system allows selection of test types and links them to a specific patient and appointment.

1. Lab Report Notifications

* Once lab tests are completed by lab assistants, doctors receive automated notifications.

### 5.1.2 Receptionist-Facing Features

* 1. Appointment Management
* Receptionists can schedule, update, or cancel appointments based on doctor availability.
* Emergency appointments can be prioritized and flagged accordingly.
  1. Patient Record Entry
* New patient records can be created, including demographics, contact information, and bed assignments.
* Existing records can be updated as needed during registration or discharge.
  1. Receptionist Dashboard
* A centralized interface provides quick access to appointments, patient data, room availability, and department assignments.
* Dashboard includes search and filter options for efficient data handling.

### 5.1.3 Admin-Facing Features

1. User Management

* Admins can create, update, or delete user accounts for doctors, lab assistants, receptionists, and patients.
* Role-based access control is enforced during account setup.

1. System Monitoring

* Admins can monitor system usage, view logs, and ensure all modules are functioning correctly.
* The system provides oversight into activities such as logins, lab test traffic, and appointment flow.

1. Admin Dashboard

* A dedicated dashboard offers insights into key system metrics (e.g., total users, daily appointments, pending lab tests).
* Admins can perform maintenance tasks and manage department or room data.

### 5.1.4 Lab Assistant-Facing Features

* 1. Lab Test Queue Management
* Lab assistants can view a real-time queue of assigned lab tests.
  1. Test Result Entry
* Completed reports are time-stamped and automatically linked to the relevant patient and appointment.
  1. Result Submission Notifications
* Once a test result is submitted, relevant users (e.g., requesting doctor, patient) are notified through the system.

# Chapter 6: Conclusion & Recommendations

To conclude, “KU Health-Care” represents a successful implementation of a healthcare management system developed using React for the frontend, Express and TypeScript for the backend, PostgreSQL for database management, and Prisma as the ORM. The platform effectively supports role-based access for admins, doctors, receptionists, and patients, enabling smooth coordination of appointments, medical records, and lab test workflows.

Throughout the development process, we encountered several challenges especially in backend logic, database structuring, and managing role-specific features. However, these obstacles provided valuable learning experiences that strengthened our understanding of full-stack development and system design.

By the end, we were able to deliver a clean, functional, and user-friendly platform that simplifies healthcare workflows and improves communication between departments. The responsive UI ensures accessibility across devices, making the system practical for real-world healthcare environments.

## 6.1 Limitations

While "KU Health-Care" offers a comprehensive healthcare solution, it also has several limitations that are important to consider:

### 6.1.1 Lack of Real-Time Communication

* The system does not support live chat or video consultations between patients and doctors, limiting remote care capabilities.

### 6.1.2 No Integration with External Services

* Features such as online payments, insurance verification, or integration with pharmacy inventory systems are currently not implemented.

### 6.1.3 Manual Lab Test Entry

* Lab assistants must enter test results manually, which may lead to errors and inefficiencies during high patient volumes.

### 6.1.4 Limited Mobile Experience

* The system is responsive but lacks a native mobile application, which may affect accessibility and user experience on smartphones.

## 6.2 Future Enhancement

### 6.2.1 Telemedicine Support

* Add real-time chat and video consultation features to facilitate remote medical services.

### 6.2.2 Mobile Application Development

* Create native Android and iOS apps to enhance accessibility and user convenience.

### 6.2.3 Online Payment Integration

* Integrate payment gateways for handling appointment fees, lab charges, and billing.

### 6.2.4 Automated Reporting and Analytics

* Develop advanced dashboards for system analytics, doctor performance, and patient trends.

# References

React. (n.d.). React – A JavaScript library for building UI. Retrieved from <https://reactjs.org/>

Express. (n.d.). Express - Node.js web application framework. Retrieved from <https://expressjs.com/>

TypeScript. (n.d.). TypeScript: JavaScript with syntax for types. Retrieved from <https://www.typescriptlang.org/>

PostgreSQL Global Development Group. (n.d.). PostgreSQL Documentation. Retrieved from <https://www.postgresql.org/docs/>

Prisma. (n.d.). Prisma Documentation. Retrieved from <https://www.prisma.io/docs/>

Epic Systems Corporation. (n.d.). Epic Electronic Health Record (EHR) Software. Retrieved from <https://www.epic.com/software>

OpenMRS Community. (n.d.). OpenMRS: Open-source medical record system platform. Retrieved from <https://openmrs.org/>

Bahmni. (n.d.). Bahmni: An easy-to-use EMR and Hospital System. Retrieved from <https://www.bahmni.org/>

# Appendix

A Gantt chart is a vital tool to visualize and manage the timeline of a project. It presents activities and milestones in a clear, graphical format. Essentially, a Gantt chart is a type of bar chart that displays the start and end dates of different tasks and phases within a project. These tasks and phases represent the breakdown of the project’s overall work structure.

Our project progressed alongside the growth of our skills and understanding. We successfully completed the project over a span of 10 weeks, with the timeline and key activities outlined as follows:

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure : Gantt Chart

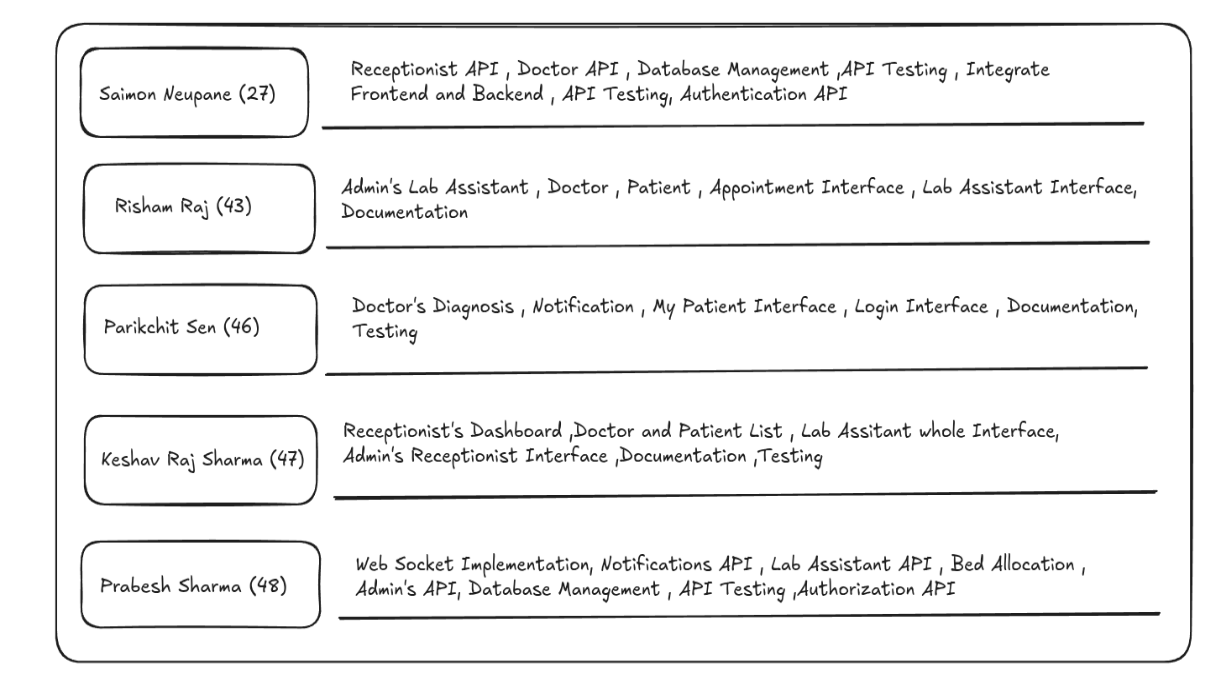




Figure : Work Division

A screenshot of a login form

AI-generated content may be incorrect.

Figure : KU Health Care Login page

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Admin Dashboard

A list of doctors with a list of medical records

AI-generated content may be incorrect.

Figure : List of Doctors Page

A screenshot of a medical form

AI-generated content may be incorrect.

Figure : New Doctor Add Page

A computer screen shot of a list of lab assistants

AI-generated content may be incorrect.

Figure : Lab Assistant Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Add New Lab Assistant Page

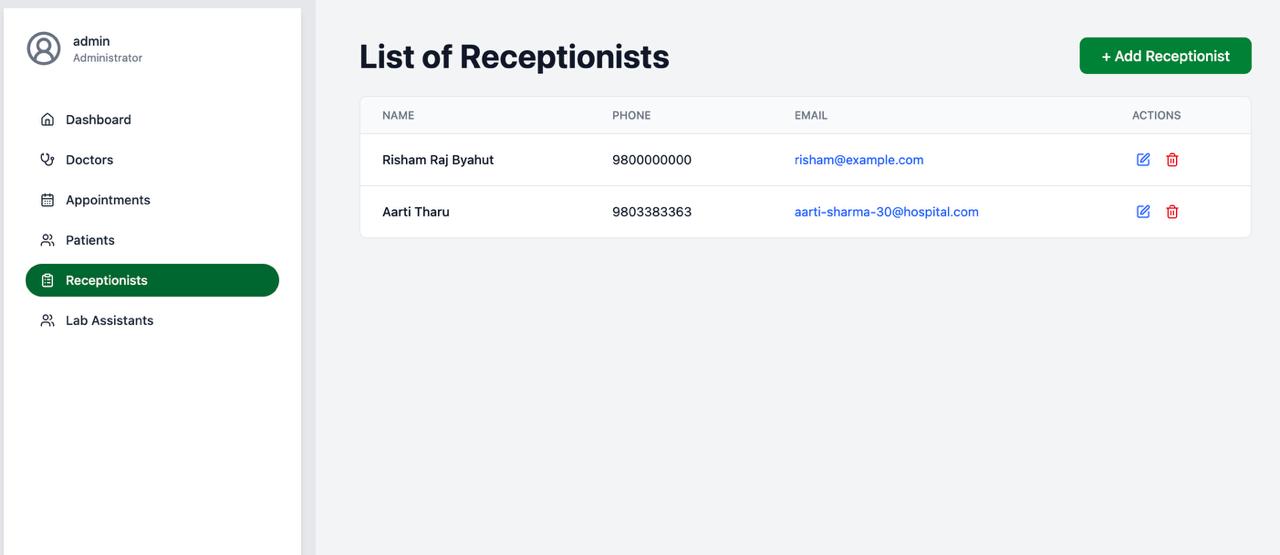


Figure : List of Receptionist Page

A screenshot of a medical form

AI-generated content may be incorrect.

Figure : Patient List Page

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Receptionist Dashboard

A screenshot of a computer

AI-generated content may be incorrect.

Figure : List of Doctors and Patients Appointed

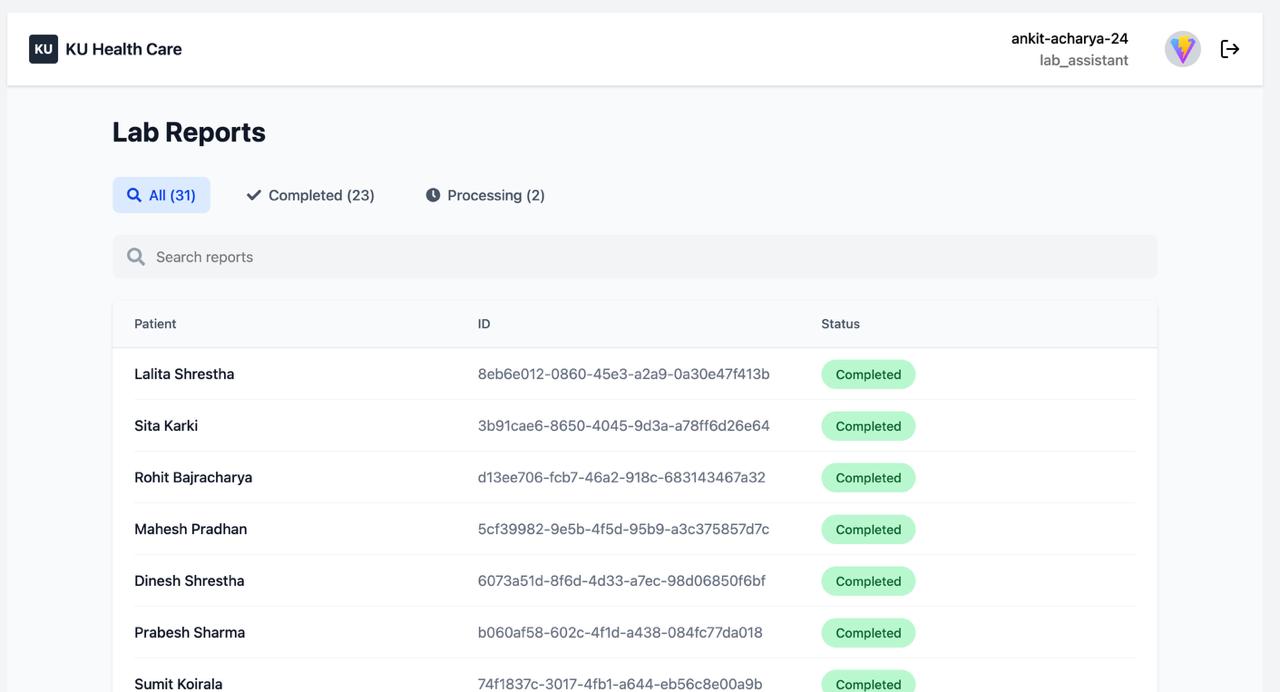


Figure : Lab Assistant Page

A screenshot of a list of appointment

AI-generated content may be incorrect.

Figure : Appointment List Page