Tribhuvan University

Faculty of Humanities and Social Science

A Project Report

On

PulsePoint (Hospital Management System)

Submitted to

Department of Computer Application

Greenfield National College

Bafal, Kathmandu

In partial fulfilment of the requirements for the bachelor's in computer Application

Submitted by

Gaurab Sunar(Reg. no. 6-2-717-6-2020)

August, 2025

Under the Supervision of

Babita Rimal

Tribhuvan University

Faculty of Humanities and Social Sciences

Greenfield National College

Bafal, Kathmandu

Bachelor in Computer Applications (BCA)

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by Gaurab Sunar entitled "**PulsePoint**" in the Partial Fulfillment of requirement for the degree of Bachelor in Computer Application is recommended for that final evaluation.

_____

SIGNATURE

Babta Rimal

SUPERVISOR

BCA Department

Greenfield National College

Tribhuvan University

Faculty of Humanities and Social Sciences

Greenfield National College

Bafal, Kathmandu

Bachelor in Computer Applications (BCA)

LETTER OF APPROVAL

This is to certify that this project prepared by **Gaurab Sunar** entitled "**PulsePoint**" in the Partial Fulfillment of requirement for the degree of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

…….. Name – Babita Rimal Lecturer BCA Department Greenfield National College Bafal, Kathmandu	……. Name – Mr. Ganesh Kumar Upadhyaya Head of Department BCA Department Greenfield National College Bafal, Kathmandu
……. Name – Balkrishna Subedi Internal Examiner BCA Department	……. Name – External Examiner

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AI	Artificial Intelligence
DB	Database
JWT	JSON Web Token
ML	Machine Learning
NLP	Natural Language Processing
ORM	Object-Relational Mapping
UI	User Interface
UX	User Experience
WS	WebSocket
JSON	JavaScript Object Notation
RBAC	Role-Based Access Control
RF	Random Forest

ACKNOWLEDGEMENT

I would like to express our sincere gratitude to our supervisor Mr. Ganesh Kumar Upadhyaya for his continuous guidance, valuable suggestions, and encouragement throughout the development of our project “PulsePoint - Integrated Healthcare & Hospital Management System.” His support has been instrumental in enhancing our technical knowledge and helping us successfully complete this work. We would also like to extend our heartfelt thanks to our parents and friends for their constant motivation and support during the project journey. Finally, we are grateful to all those who directly or indirectly contributed to the completion of this project.

Yours Sincerely,

Gaurab Sunar

ABSTRACT

PulsePoint is a web-based Hospital Management System (HMS) designed to digitize healthcare services and improve patient–doctor interaction. The system supports role-based access for Administrators, Doctors, and Patients, with features such as appointment scheduling, electronic health record (EHR) management, billing, prescriptions, and real-time bed allocation. A built-in chat module allows seamless communication between users and doctors, enhanced by a Naive Bayes-based text-to-emoji suggestion algorithm that improves user engagement.

Additionally, PulsePoint integrates a Machine Learning-powered disease detection service implemented in Python using a Random Forest Classifier, which provides preliminary diagnosis support based on patient symptoms. The system is built with React.js, Node.js, Python, and PostgreSQL, ensuring scalability, performance, and secure data management. PulsePoint demonstrates the application of modern web technologies and AI-driven decision support tools to enhance hospital operations, reduce administrative overhead, and improve patient care quality.

Keywords: Hospital Management System, EHR, Chat System, Naive Bayes, Emoji Suggestion, Random Forest Classifier, Machine Learning, Python.

CHAPTER 1: INTRODUCTION

1.1 Introduction

Healthcare is one of the most essential sectors of society, and its effective management is vital for ensuring quality patient care and efficient use of resources. Modern hospitals are complex organizations that must handle numerous interconnected operations such as patient registration, appointment scheduling, medical history tracking, prescription management, billing, laboratory coordination, and interdepartmental communication. Traditional paper-based systems often result in inefficiencies, including errors in data entry, redundancy, lack of real-time updates, and difficulties in maintaining data accuracy and security.

To overcome these challenges, the Hospital Management System (HMS) has been developed as a centralized, web-based platform that automates and streamlines core hospital operations. The system supports role-based access for administrators, doctors, and patients, enabling efficient hospital resource management, quick access to medical records, seamless appointment handling, and transparent billing. Additionally, the platform integrates a real-time chat module for direct communication between doctors and patients, making healthcare services more accessible and patient friendly.

A key innovation in the system is the integration of a Machine Learning-based disease prediction module, which uses patient-reported symptoms to suggest possible diagnoses. This predictive feature serves as a Clinical Decision Support System (CDSS), assisting doctors in preliminary diagnosis and improving overall healthcare delivery. Built with modern technologies such as React.js, Node.js, Python, and PostgreSQL, the HMS ensures scalability, security, and reliability, offering a robust solution for digitizing healthcare management.

1.2 Problem Statement

Healthcare management in Nepal faces significant challenges due to reliance on traditional, paper-based hospital administration systems. Manual record-keeping, fragmented workflows, and lack of centralized data lead to inefficiencies, delays, and increased risk of errors in patient care. Doctors and hospital staff often struggle to access complete patient histories quickly, schedule appointments efficiently, or coordinate clinical resources

effectively. Patients, on the other hand, face long waiting times, limited transparency in billing, and difficulties in communicating with healthcare providers.

The absence of a digital, centralized, and intelligent hospital management system results in several critical problems, including:

- Inefficient management of patient records, appointments, prescriptions, and billing.
- Difficulty in coordinating hospital resources such as bed allocation, lab tests, and consultant schedules.
- Limited support for doctors in preliminary diagnosis and clinical decision-making.
- Lack of real-time communication between patients and healthcare providers.
- Challenges in maintaining data integrity, security, and compliance with healthcare regulations.

Without a modern, web-based platform that integrates role-based access, machine learning for disease prediction, automated workflows, and communication tools, hospitals are unable to optimize operations or deliver timely, high-quality patient care. The need for a scalable, secure, and intelligent hospital management system is therefore critical to improve efficiency, reduce administrative burdens, and enhance overall healthcare delivery.

1.3 Objectives

The primary objective of the PulsePoint Hospital Management System (HMS) is to streamline hospital operations and enhance patient care by providing an intelligent, web-based platform. Specifically, the system aims:

- To develop a centralized and secure hospital management system that manages patient registration, appointments, electronic health records (EHR), prescriptions, billing, bed allocation, and role-based access for administrators, doctors, and patients.
- To implement intelligent and interactive features, including a real-time patient–doctor chat module with a Naïve Bayes–based text-to-emoji suggestion system to enhance communication and user engagement.

- To integrate a Machine Learning–based disease prediction module using a Random Forest Classifier to support doctors in preliminary diagnosis and informed clinical decision-making.

1.4 Scope and Limitations

The PulsePoint Hospital Management System (HMS) is developed to digitalize hospital operations and enhance the efficiency of healthcare delivery. It provides a centralized platform to manage patient information, appointments, prescriptions, billing, bed allocation, and communication between different stakeholders. By incorporating modern technologies such as real-time chat, machine learning-based disease prediction, and role-based access, the system aims to streamline workflows, reduce manual efforts, and support better decision-making for doctors and administrators. While the system offers significant improvements in hospital management, it also faces certain limitations in its initial implementation.

1.4.1 Scopes

The PulsePoint Hospital Management System (HMS) is designed to digitize hospital operations and improve the efficiency of healthcare services. It provides a centralized platform to manage patient records, appointments, prescriptions, billing, bed allocation, and communication between departments. The system supports role-based access, allowing patients and doctors to register, log in, and interact through a secure real-time chat system, while administrators oversee hospital resources and user roles.

The chat feature includes a Naïve Bayes-based text-to-emoji suggestion algorithm that enhances user interaction, and the system also integrates a disease prediction module using a Random Forest Classifier to assist doctors with preliminary diagnoses based on patient symptoms. Built as a full-stack web application with React.js, Node.js, and PostgreSQL, the HMS is scalable and prepared for future extensions such as multilingual support, mobile accessibility, advanced reporting, and AI-driven analytics.

1.4.2 Limitations

While the PulsePoint Hospital Management System (HMS) aims to streamline hospital operations and improve patient care, it has certain limitations in its initial implementation:

- The disease prediction module may have limited accuracy with incomplete, ambiguous, or inconsistent patient symptom data.

- The chat system does not support real-time video or voice consultations; communication is limited to text-based messaging.
- The system currently supports only the English language interface, which may restrict accessibility for non-English speaking users.
- Some hospital workflows, specialized departments, and complex reporting features may not be fully integrated in the initial version.

1.5 Development Methodology

Agile methodology was used while developing the PulsePoint – Flower Gardening & Disease Management System. Since this project does not have fixed requirements, specific documentation, or a fully predefined scope, Agile methodology was the most suitable approach. It provides flexibility to adapt to changes at any stage of development, ensuring continuous improvement and quick delivery of functional modules. Agile is an iterative and incremental approach to project management and software development that emphasizes collaboration, adaptability, and testing. In PulsePoint, several modules such as disease detection, marketplace, and payment integration required frequent updates and testing, particularly while refining the CNN-based prediction model and ensuring smooth user interactions.

Therefore, Agile methodology was adopted to allow small incremental changes, continuous testing, and timely modifications as per project needs. The following illustration represents the different phases of the Agile methodology:

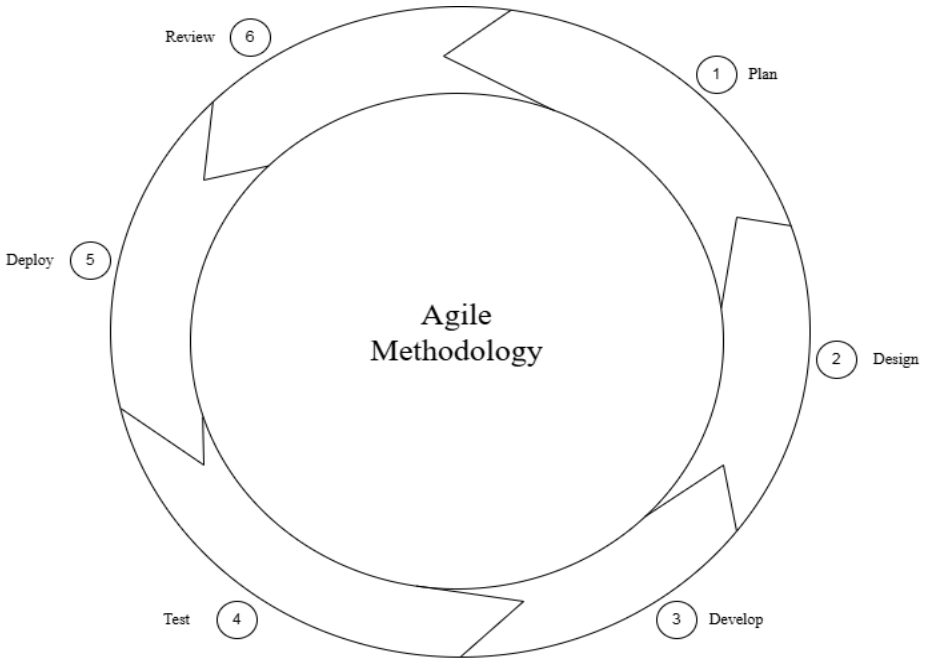


Figure 1.1: Agile Methodology

The sequential phases in the Agile model for PulsePoint are:

Research: Gather data on flower species, common diseases, symptoms, and treatments. Collect requirements from gardeners and store owners to design the disease detection and marketplace modules.

Design: Create the system architecture, database schema, and user interface wireframes. Design how users (gardeners and store owners) interact with features like image upload, disease detection, and marketplace browsing. Adjust designs based on feedback.

Implementation: Develop the core modules, including the CNN-based flower disease prediction system, user registration, marketplace, and payment integration. Continuously add features, such as product filtering, disease recommendations, and secure transactions.

Testing: Test the accuracy of disease predictions, functionality of the marketplace, user experience, and system performance. Perform iterative testing and feedback collection to improve the platform.

Documentation: Continuously update technical documentation, user manuals, and guides for PulsePoint to ensure clarity for future users and developers.

1.6 Report Organization

Chapter 1: Introduces the PulsePoint Hospital Management System, outlining the problem statement, objectives, scope, and limitations. It sets the context for the project by highlighting the challenges in traditional hospital management and the need for a digital, intelligent solution.

Chapter 2: Presents background study related to hospital management systems, healthcare digitization, and AI-assisted disease prediction. This chapter includes a literature review to provide a broader understanding of the concepts, technologies, and methods used in the healthcare domain, along with analysis of similar systems for benchmarking and comparison.

Chapter 3: Focuses on system analysis and design, illustrated using various charts, diagrams, and figures. It details the functional requirements through use cases, depicts database schemas, explains interface designs, and presents the system architecture to demonstrate how different modules interact and operate cohesively.

Chapter 4: Discusses the tools, technologies, and techniques employed during implementation. It provides details of key algorithms applied in the system, including the Naïve Bayes-based chat emoji suggestion and the Random Forest-based disease prediction module. This chapter also covers the creation of test cases and evaluation strategies to validate individual modules and the overall system performance.

Chapter 5: Summarizes lessons learned throughout the development process, presents conclusions drawn from the project, and offers recommendations for future enhancements. This includes suggestions for scaling the system, adding advanced features, or applying the platform to related areas within healthcare management.

CHAPTER 2: BACKGROUND STUDY & LITERATURE REVIEW

2.1 Background Study

The modern healthcare landscape is characterized by an overwhelming volume of data and complex, interconnected workflows. Efficient management of this data is paramount to ensuring quality patient care, optimal resource utilization, and sound financial health for medical institutions. A Hospital Management System (HMS) is a comprehensive, integrated software solution designed to automate and manage all the administrative, clinical, and financial operations of a hospital. It serves as a centralized digital nervous system, replacing error-prone, inefficient, and time-consuming paper-based records and manual processes.

The core value of an HMS lies in its ability to provide a single source of truth for all hospital stakeholders. This project, PulsePoint, is built upon several fundamental technological and conceptual pillars:

1. **Role-Based Access Control (RBAC):** This is a security paradigm that restricts system access to authorized users based on their roles within an organization. In PulsePoint, three primary roles are defined:
    - Administrators have overarching control to manage system users, allocate resources, generate reports, and ensure system integrity.
    - Doctors are granted access to patient records, appointment schedules, prescription modules, and clinical decision-support tools.
    - Patients can book appointments, view their medical history and bills, and communicate with healthcare providers.
2. **Electronic Health Record (EHR):** An EHR is a digital version of a patient's paper chart. It is a real-time, patient-centered record that makes information available instantly and securely to authorized users. It contains a patient's medical history, diagnoses, medications, treatment plans, immunization dates, allergies, radiology images, and laboratory test results.

3. **Full-Stack Web Development:** PulsePoint is built using full-stack architecture, meaning it encompasses both the front-end (client-side) and back-end (server-side) parts of the application.

    - Front-end (React.js): Responsible for the user interface and user experience. It handles what the user sees and interacts with in their web browser.
    - Back-end (Node.js, Python): Manages the application logic, database interactions, and server operations. It processes requests from the front-end, executes business logic, and retrieves or stores data.
    - Database (PostgreSQL): A powerful, open-source relational database system that stores all persistent data, including patient records, user credentials, appointment details, and more, ensuring data integrity and security.

4. **Machine Learning (ML) in Healthcare:** ML is a subset of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. In PulsePoint, ML is applied in two key areas:

    - **Random Forest Classifier:** An ensemble learning algorithm that operates by constructing a multitude of decision trees during training. It is used in the disease prediction module to analyze input symptoms and output a probabilistic preliminary diagnosis, serving as a valuable tool for clinical decision support.
    - **Naive Bayes Algorithm:** A probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It is employed in the real-time chat module to analyze text messages and suggest contextually relevant emojis, thereby enhancing user engagement and communication.

2.2 Literature Review

The increasing complexity of healthcare delivery and the growing volume of patient data have made the adoption of digital hospital management systems (HMS) essential. Hospital management system (HMS) streamlines administrative workflows, improve patient care,

and reduce medical errors by replacing traditional paper-based processes with integrated digital platforms. Globally, the transition to digital systems has been shown to reduce patient waiting times, enhance documentation accuracy, and support efficient decision-making in clinical environments [1].

In Nepal, hospitals such as Shree Harsha Hospital have started adopting digital HMS, but challenges related to infrastructure, staff training, and data security continue to affect widespread implementation [2]. This context underscores the importance of developing a comprehensive and practical HMS tailored to local needs.

Artificial intelligence (AI) has emerged as a key enabler for enhancing clinical decision support within HMS. The Random Forest algorithm, a supervised ensemble learning technique, has been widely applied for symptom-based disease prediction, providing high accuracy and reducing overfitting by combining predictions from multiple decision trees. Research by Zhang et al. [3] demonstrated that Random Forest classifiers could achieve nearly 90% accuracy in preliminary diagnoses across common conditions, shortening diagnostic times and supporting physicians in clinical decision-making. Within Nepal, preliminary studies [4] have explored machine learning models for early disease detection, indicating significant potential for integrating AI into local hospital systems.

In addition to clinical support, enhancing communication between patients and healthcare providers is critical. Lightweight AI applications such as Naive Bayes classifiers have been used to enhance patient engagement. Liu and Park [5] showed that applying sentiment-aware AI to digital healthcare communication can improve satisfaction and adherence to treatment. PulsePoint demonstrates this application by using a Naive Bayes classifier within its chat module to suggest contextually relevant emojis, aiding remote consultations.

Security and privacy considerations are integral to the successful deployment of HMS. Role-based access control (RBAC) and secure authentication protocols such as JSON Web Tokens (JWT) are recognized as effective strategies to protect sensitive patient data [6]. In Nepal, while digital data protection regulations exist, many hospitals still lack robust cybersecurity infrastructure, highlighting the need for secure, scalable, and user-friendly HMS solutions.

Despite the growing research on AI applications and digital hospital systems, most studies focus on isolated algorithms or theoretical prototypes, with limited emphasis on fully integrated HMS platforms. PulsePoint fills this gap by unifying core hospital operations

such as appointments, billing, and electronic health record (EHR) management with AI-driven disease prediction and communication support. This integrated approach demonstrates how modern technologies can be practically applied to create a comprehensive, secure, and efficient HMS tailored to the Nepalese healthcare context.

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis

System analysis is the systematic study, understanding, and documentation of the PulsePoint Hospital Management System, focusing on its components, functionalities, and interactions. It involves analyzing user requirements, workflows, data handling, and technologies to clearly define how the system should operate to effectively support administrators, doctors, and patients. The primary goals of system analysis in PulsePoint are to identify user needs, define system boundaries, model hospital operations such as patient management, appointments, billing, and communication, and propose solutions that align with healthcare objectives. The results of this analysis form the foundation for designing and developing a robust, user-friendly, and scalable platform for hospital management and clinical decision support.

3.1.1 Requirement Analysis

Project has the following functional and non-functional requirements:

1. Functional Requirements

a. Users (patients, doctors, or administrators) should be able to register and log in securely.

b. Patients should be able to book, reschedule, or cancel appointments.

c. Doctors should be able to view their appointment schedules, update patient records, and issue prescriptions.

d. The system must securely handle sensitive data such as user credentials, patient records, and billing details.

e. Administrators should be able to manage users (patients, doctors, staff), allocate beds/wards, and oversee hospital resources.

f. The system should allow billing management, including viewing, generating, and paying bills.

g. Both doctors and patients should be able to communicate through a secure chat system.

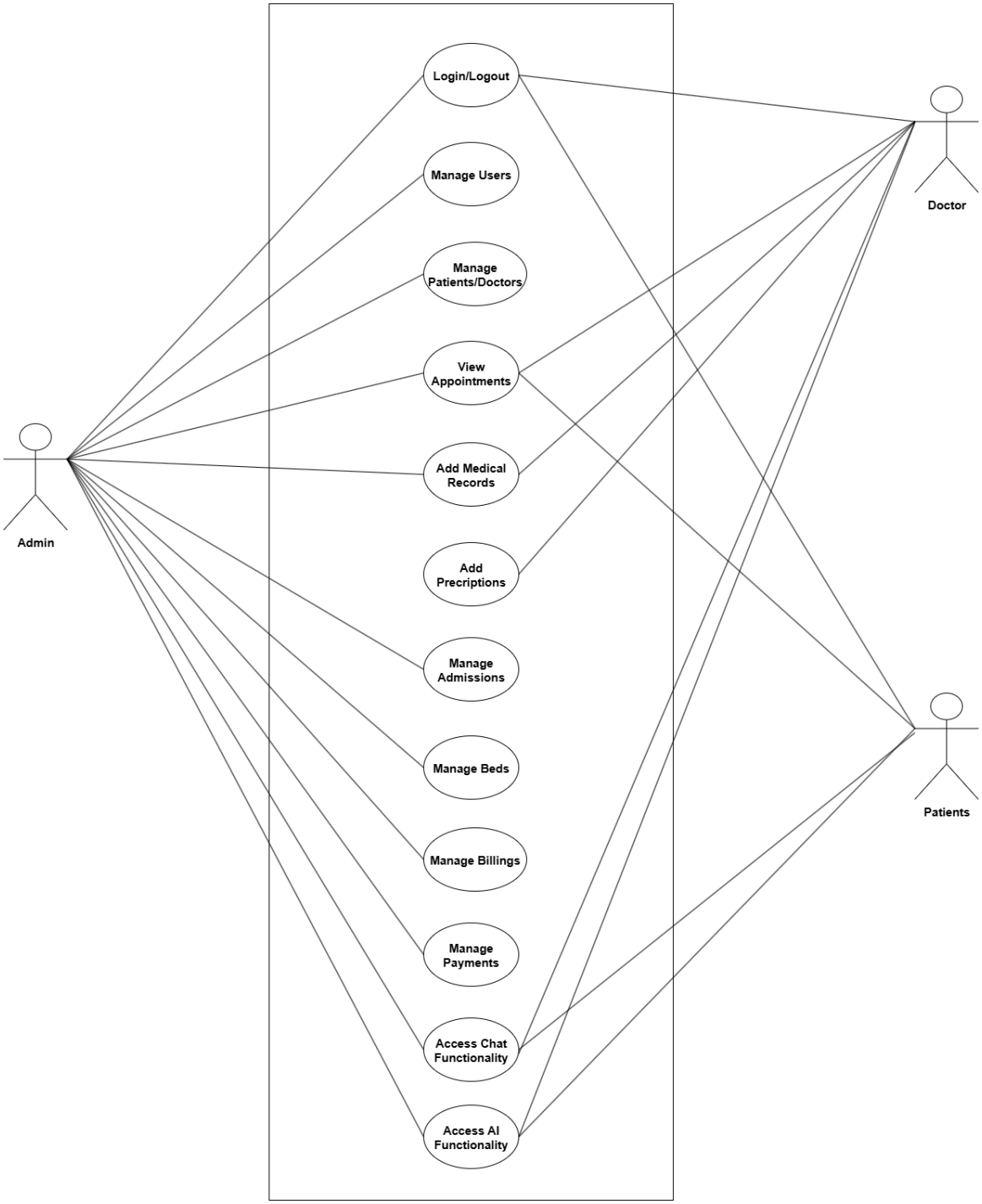


Figure 3.1: Usecase diagram of PulsePoint

Actors:

Administrator – manages users, hospital resources, testimonials, and reports.

Doctor – manages appointments, patient records, prescriptions, and communicates with patients.

Patient – books appointments, views medical history, pays bills, and communicates with doctors.

Use Cases:

- **Register/Login/Logout:** Users can sign up as regular users, and admins created at the start can add patients with extra details and assign doctors also.
- **Book Appointments:** Patients can schedule appointments with doctors and doctors also can assign appointments for himself with any specific patents.
- **View Appointment Schedule:** Doctors can access their appointment schedules and admin also can see all doctor's appointments.
- **Manage Medical Records:** Doctors can view/update patient records.
- **Prescriptions:** Doctors can generate and share prescriptions digitally.
- **Chat System:** Patients and doctors can exchange secure messages.
- **View Medical History:** Patients can review their medical history and past diagnoses.
- **View & Pay Bills:** Patients can access their billing information and make payments.
- **Manage Users:** Admin manages patients, doctors, and staff accounts.
- **Manage Beds:** Admin allocates and monitors hospital resources.
- **Admission:** Admin can admit patients to specific beds and discharge them once their billing is cleared.

Interactions:

- Patients interact with PulsePoint for booking, accessing records, billing, and communication.
- Doctors interact with PulsePoint to manage schedules, records, and consultations.
- Administrators interact with PulsePoint to ensure smooth hospital operations and manage resources.

2. Non-Functional Requirement

Security:

The system ensures data confidentiality, integrity, and privacy through secure authentication, role-based access control, and a secure forgot password mechanism. Access to sensitive information such as patient medical records, prescriptions, and billing details is limited to authorized users only. All data transmissions are encrypted to prevent unauthorized access and external attacks.

Performance:

The system is fast, reliable, and responsive. Optimized database design and efficient queries

enable quick retrieval of patient records, appointments, and billing information while supporting multiple concurrent users without performance degradation.

Scalability:

The system architecture allows future expansion in terms of features, number of users, and hospital size without requiring major redesigns.

Usability:

The user interface is simple, intuitive, and user-friendly, ensuring ease of use for users with different levels of technical expertise, including patients with minimal digital literacy.

Maintainability:

The modular design of the system allows individual components such as appointments, billing, chat, prescriptions, and authentication to be updated, tested, and maintained independently without affecting overall system functionality.

3.1.2 Feasibility Study

We conducted a feasibility study before developing the platform to assess the project's technical, economic, and operational feasibility. According to the findings of the study:

1. Technical Feasibility

- **Frontend:** Developed with React.js, styled using Tailwind CSS and ShadCN UI, ensuring a responsive and user-friendly interface.
- **Backend:** Built with Node.js (Express.js) for core hospital functionalities such as user management, appointments, billing, and chat services. Data is securely stored in a PostgreSQL database.

Machine Learning Services:

- **Naive Bayes Algorithm:** Used in the chat system to provide real-time emoji suggestions based on the context of doctor-patient conversations, making communication faster and more engaging.
- **Random Forest Classifier:** Used for disease prediction based on patient-reported symptoms, assisting doctors in preliminary diagnosis and decision-making.

2. Operational Feasibility

- Simple and intuitive interfaces for patients, doctors, and administrators.
- Appointment booking, medical history access, and chat features designed for ease

of use.

- Role-based access ensures each user category has streamlined operations.
- Modular architecture allows easier deployment, testing, and future upgrades.

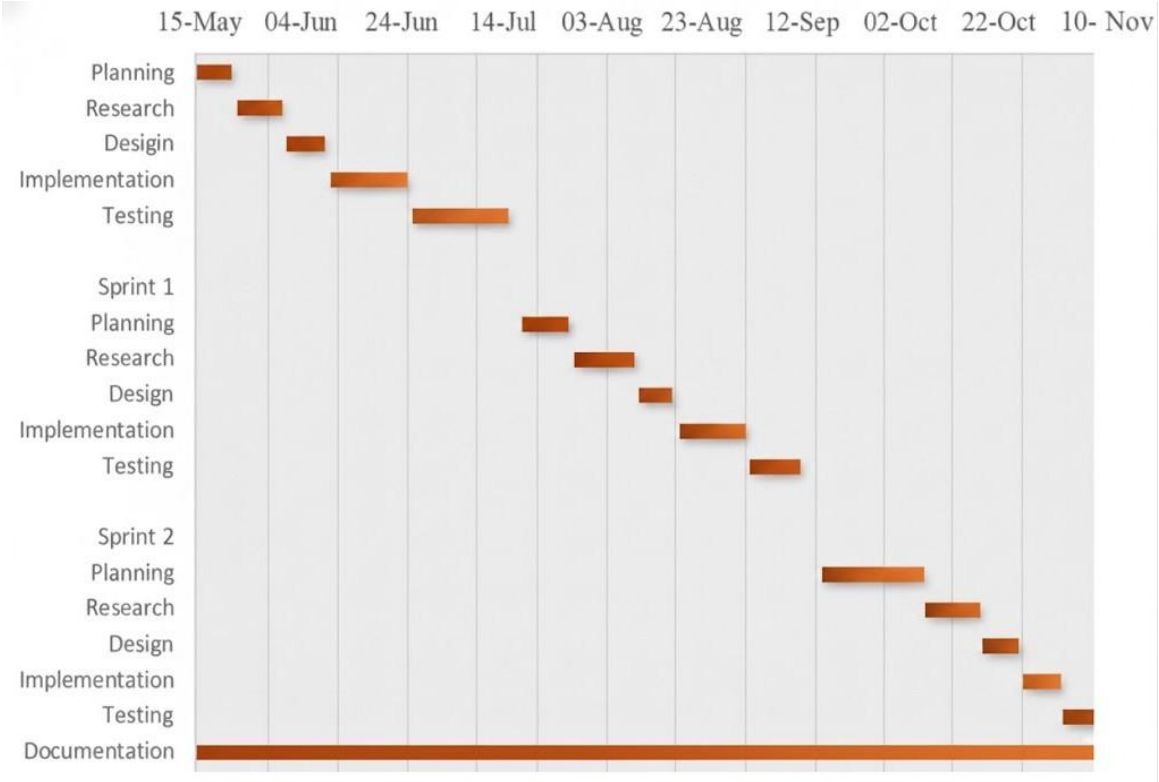
3. Economic Feasibility

- Cost-effective development due to the use of open-source technologies (React, Node.js, Python, PostgreSQL).
- Free datasets and libraries reduce expenses for the disease prediction module.
- Affordable for student-level and institutional implementations without heavy infrastructure investments.

4. Schedule Feasibility

- The project was designed and developed within the planned timeframe, with milestones for requirement gathering, system design, backend and frontend implementation, ML integration, and testing.
- Agile methodology ensured progress tracking, iterative improvements, and timely delivery.

Table 3.1 Gantt chart of PulsePoint Hospital Management System



A Gantt chart is a visual project planning tool that represents the timeline of activities involved in the development of the PulsePoint system. It helps to manage tasks such as data collection, model training, system design, and implementation by showing their duration and overlap. By mapping out start and end dates, the Gantt chart ensures efficient tracking of milestones, helps identify potential delays, and supports smooth coordination of development phases throughout the project lifecycle.

3.1.3 Object Modeling using class and object diagram

This project was designed following the Object-Oriented Programming (OOP) paradigm. A comprehensive class diagram was developed to represent the core entities of the Hospital Management System (HMS), including their attributes, methods, and interrelationships.

The main classes modeled in the system are **User, Patient, Doctor, AdminProfile, Appointment, Admission, Bed, MedicalRecord, Prescription, Medicine, Bill, BillItem, Payment, and ChatMessage**. Enumerations such as **UserRole, Gender, BedType, AppointmentStatus, AdmissionStatus, and PaymentMethod** are also defined to ensure data consistency and structured workflow representation.

- The **User class** serves as the base, extended by specific roles like **Patient**, **Doctor**, and **AdminProfile**.
- The **Appointment class** links patients with doctors and includes details such as schedule, duration, and status.
- **MedicalRecord** captures diagnosis, treatments, and attachments, while **Prescription** manages medicines prescribed by doctors.
- **Billing and Payment** are handled through the **Bill, BillItem, and Payment** classes, ensuring proper financial tracking.
- **Admission and Bed management** are modeled with classes that track patient admissions, bed types, and occupancy status.
- **ChatMessage** supports doctor-patient communication within the system.

This class diagram provides a structured blueprint for implementation, ensuring the system efficiently supports hospital processes such as **appointment scheduling, medical record management, billing, admission handling, and communication** between stakeholders.

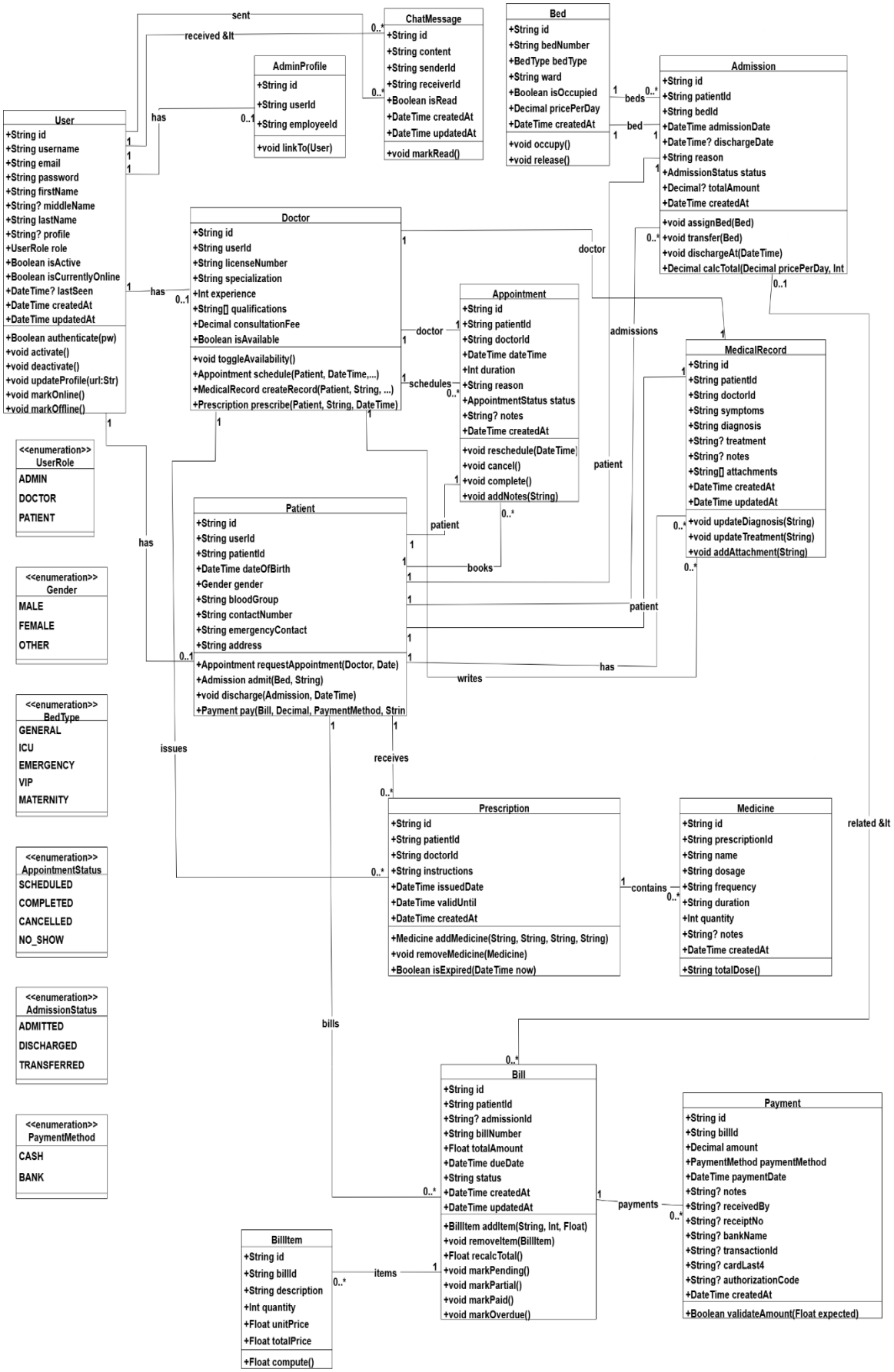


Figure 3.2: Class Diagram of PulsePoint Hospital Management System

3.1.4 Dynamic Modeling using state and sequence diagram

Dynamic modeling in PulsePoint is used to describe how the system behaves over time and how users interact with different modules. It provides a clear understanding of the system's runtime behavior through state diagrams (showing state transitions of processes) and sequence diagrams (showing time-ordered interactions among actors and components).

1. **State diagram**

   The state diagram of **PulsePoint** (Figure 3.5) presents how the system moves through key modules—**appointment booking**, **patient admission**, **billing and payment**, and **chat communication**—from login to completion of tasks.

   - **Initial state:** The system starts in the **Logged Out** state. After a successful login, users—patients, doctors, or administrators—are directed to their respective dashboards.

   - **Patient flow:** Patients can manage their appointments by moving through the states **Book Appointment → Appointment Schedule → Appointment Complete** or canceling to reach **Appointment Cancelled**.

   - **Doctor flow:** Doctors handle consultations by creating and saving records, adding prescriptions, and issuing them before returning to the **Doctor Dashboard**.

   - **Administrator flow:** Administrators admit patients, generate bills, and oversee payments. The billing process transitions through **Bill Pending**, possibly **Bill Overdue**, and finally to **Payment Successful** or **Payment Failed**, after which patients may be discharged.

   - **Chat and bed management:**

     - **Chat:** Messages progress through **Sending Message → Message Sent → Message Read**.

     - **Bed management:** Beds move through **Bed Occupied → Bed Maintenance → Bed Available**, ensuring availability after maintenance.

   This diagram captures the end-to-end state transitions, highlighting how each module of PulsePoint interacts to provide seamless hospital management.

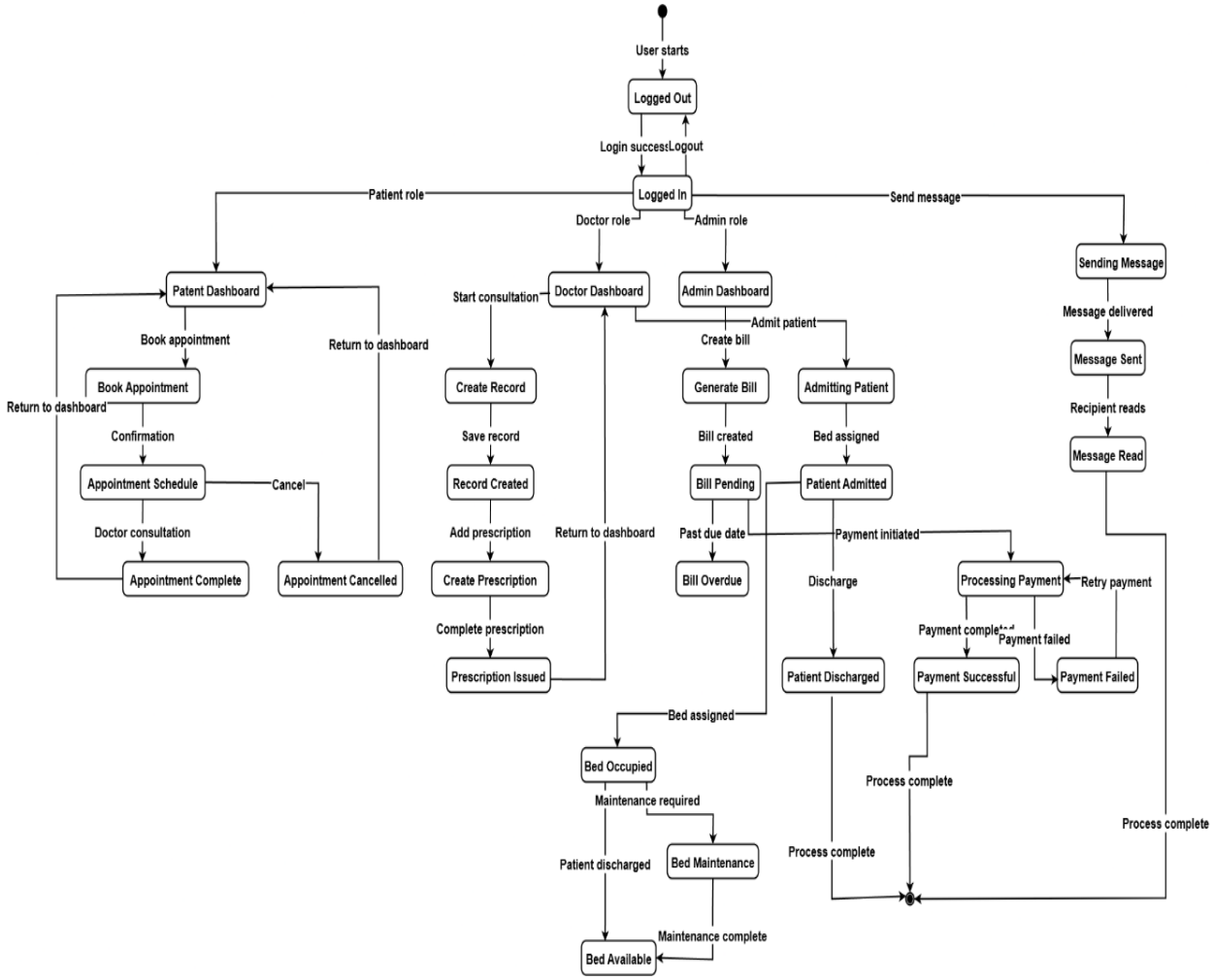


Figure 3.3: State Diagram of PulsPoint

2. Sequence diagram

The sequence diagram illustrates the step-by-step interactions between **Patients, Doctors, and Admins** with different system services, databases, and external storage during hospital management workflows.

1. **User Login**
    - A user (Patient, Doctor, or Admin) logs in by providing credentials.
    - The **Authentication Service** validates credentials via the **Prisma Client (ORM)**, which queries the **PostgreSQL Database**.
2. **Book Appointment (Patient)**
    - The patient requests an appointment by providing doctor details, slot, and time.
    - The **Appointment Service** validates the request and inserts the new appointment into the database.

3. **Add Medical Record (Doctor)**
    - The doctor adds a medical record linked to a specific appointment.
    - The **Medical Record Service** creates and stores the record in the **PostgreSQL Database**, optionally attaching files via the **File Service**.
4. **AI Diagnosis Support (Optional)**
    - A doctor can request AI-based diagnosis assistance.
    - The **Medical Record Service** interacts with the **ML Diagnosis Service** to predict symptoms using machine learning models.
    - The diagnosis report is returned to the doctor.
5. **Generate Bill (Admin)**
    - An admin generates a patient bill.
    - The **Billing Service** fetches charges from the database and creates a bill entry, inserting records into **bills** and **bill_items** tables in PostgreSQL.
6. **Upload Discharge Summary (Admin)**
    - After patient discharge, the admin uploads the discharge summary document.
    - The **File Service** stores the file in the **File Storage System**, and the corresponding metadata is updated in the database.

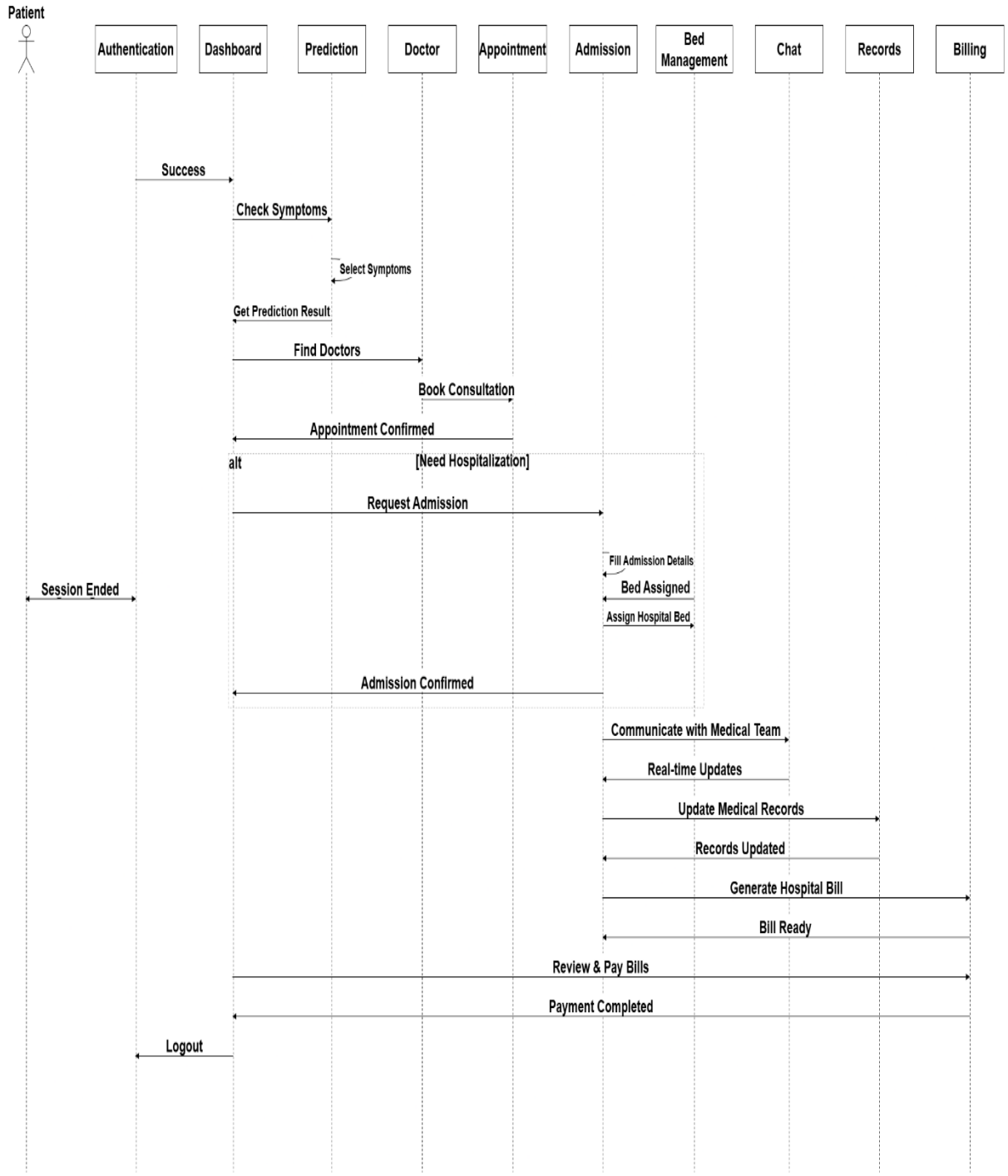


Figure 3.4: Sequence Diagram of PulsePoint

3.1.5 Process Modeling using Activity diagram

The activity diagram models the workflows of the system, illustrating how hospital processes such as patient registration, appointment booking, doctor consultation, record updates, and billing are executed step by step. It highlights the decision-making points (e.g., appointment availability, payment success/failure) and ensures smooth navigation and efficient handling of hospital operations.

Since the system involves three different types of users **Patients, Doctors, and Admins** separate activity diagrams are provided for each role. These diagrams clearly represent the unique workflows, responsibilities, and interactions of each user within the hospital management system.

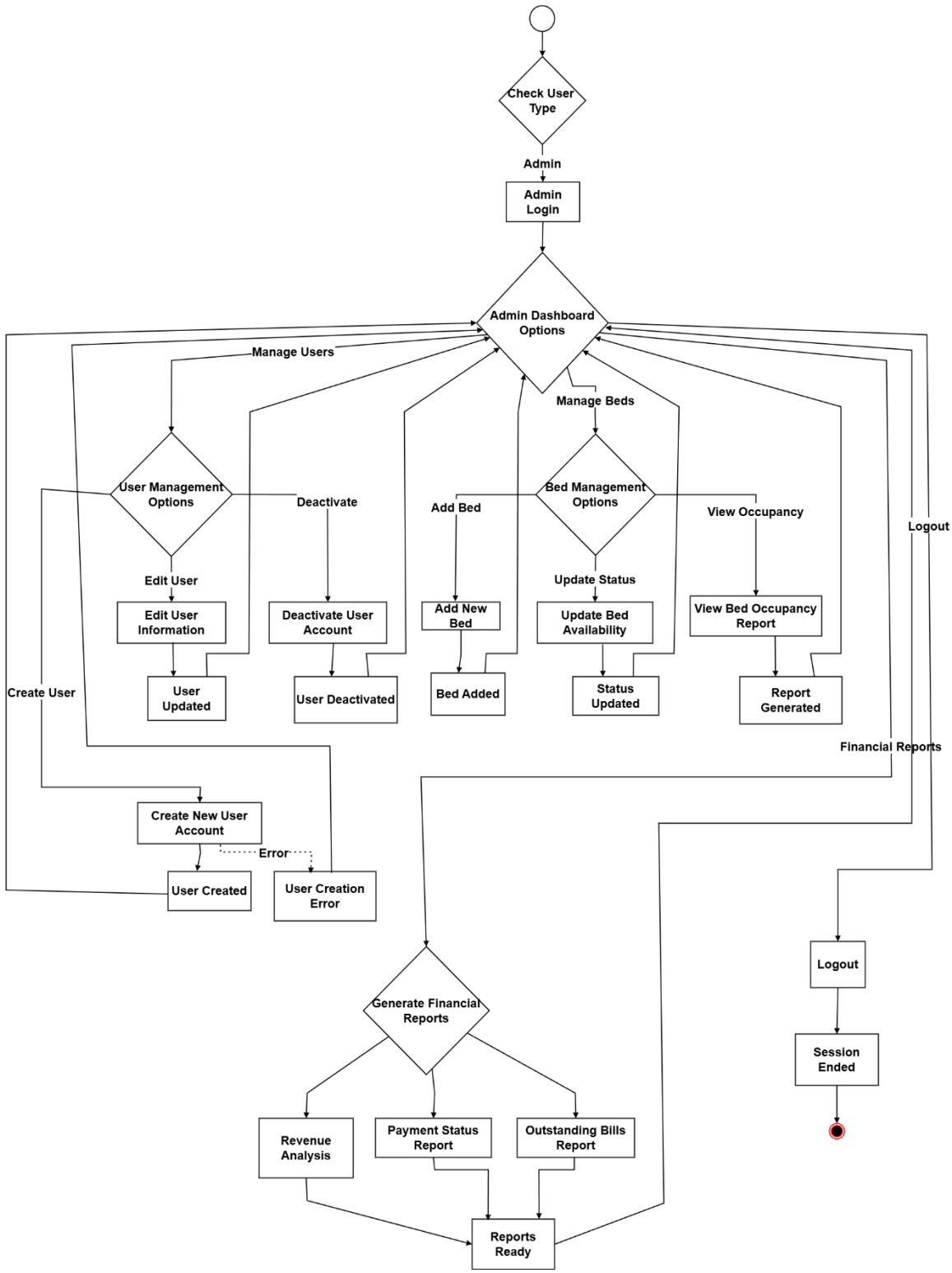


Figure 3.5: Activity Diagram of PulsePoint (Admin)

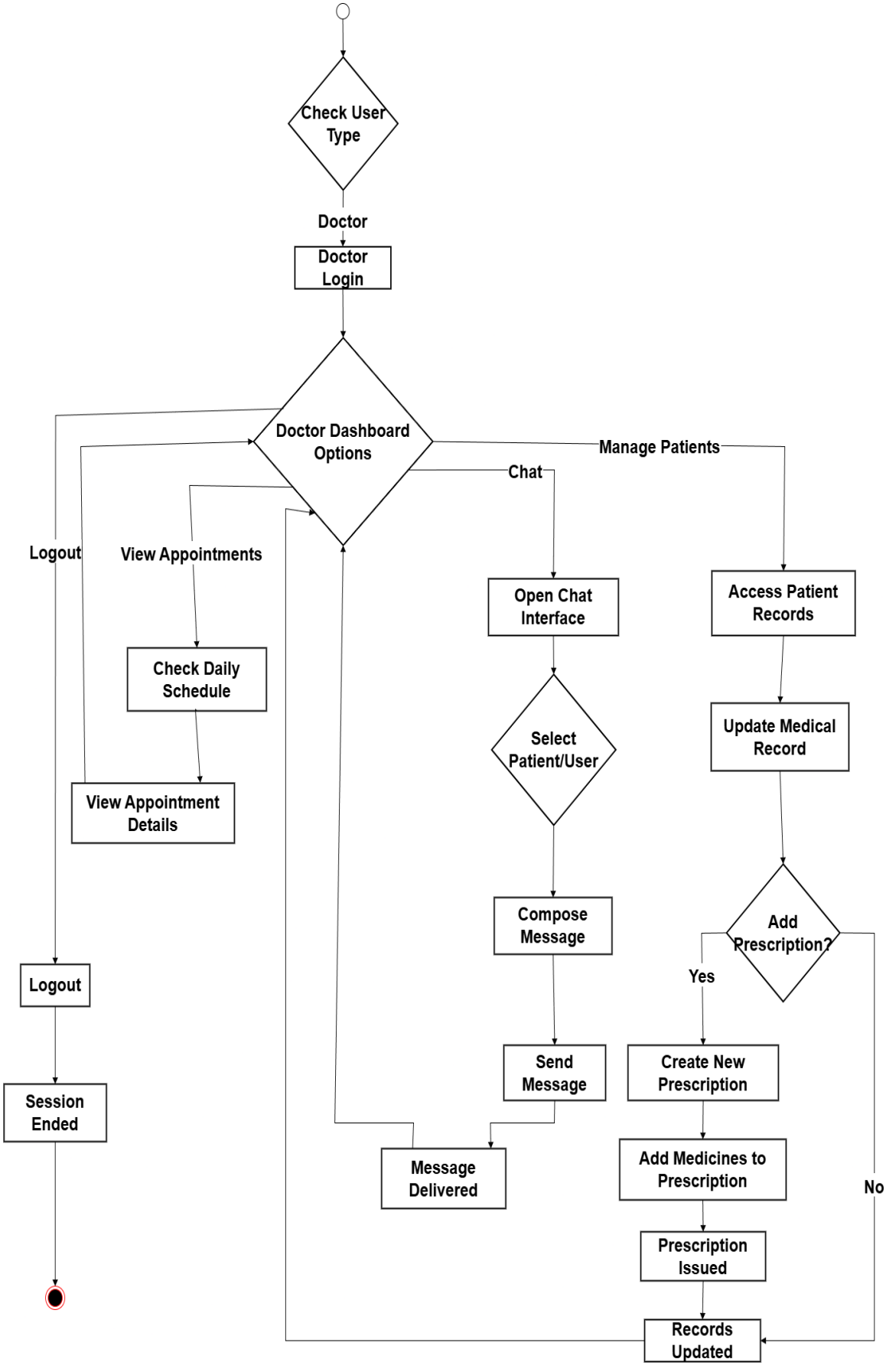


Figure 3.6: Activity Diagram of PulsePoint (Doctor)

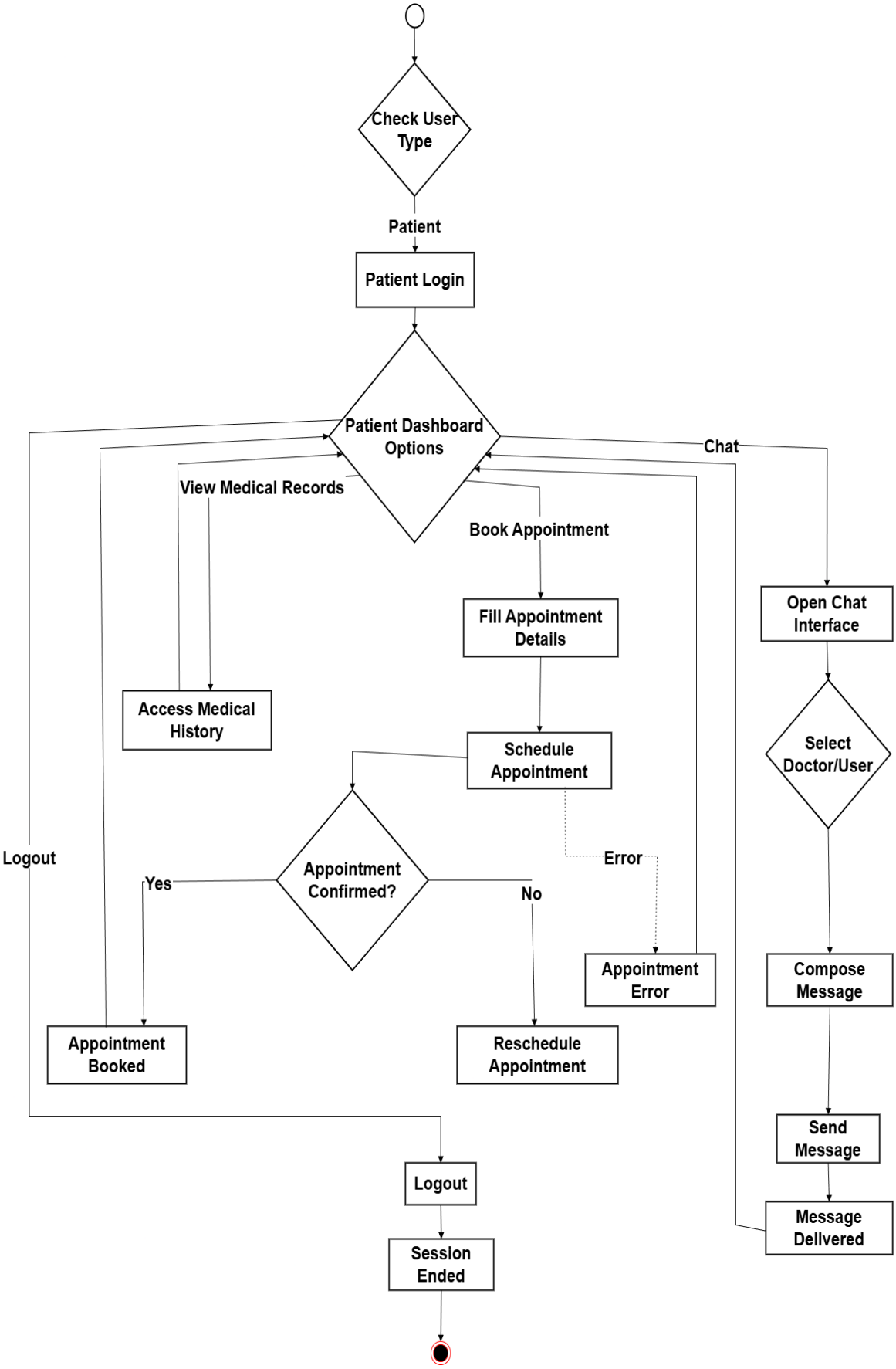


Figure 3.7: Activity Diagram of PulsePoint (Patient)

3.2 System Design

To provide a clear graphical representation of system requirements, several diagrammatic designs were prepared. These include component diagrams, deployment diagrams, and architectural designs.

3.2.1 Refinement of Class, Object, State, Sequence and Activity Diagram

The refinement process ensures that all design diagrams of the **PulsePoint Hospital Management System** accurately represent the final system structure and behaviour. After the initial analysis in Chapter 3, each diagram was iteratively improved to capture detailed relationships, data flows, and runtime interactions.

1. Class and Object Diagrams

The class diagram was refined to clearly define all entities, their attributes, methods, and interrelationships. Inheritance among **User**, **Patient**, **Doctor**, and **AdminProfile** was formalised to support role-based functionality. Associations between **Appointment**, **MedicalRecord**, **Prescription**, **Bill**, **Payment**, **Admission**, and **Bed** were tightened with precise multiplicities and constraints. The object diagram was updated to present real runtime examples—for instance, a patient object linked to multiple appointments and bills—ensuring that actual system instances conform to the class structure.

2. State Diagram

The state diagram was enhanced to describe every significant transition across modules such as appointment booking, patient admission, billing, payment, chat, and bed management. It starts from the **Logged-Out** state, proceeds to role-specific dashboards after successful login, and models detailed flows:

- Patient journey from **Book Appointment → Appointment Schedule → Appointment Complete/Cancelled**.
- Doctor workflow for creating records and issuing prescriptions.
- Administrator tasks including **Admitting Patient**, **Generate Bill**, and transitions through **Bill Pending → Bill Overdue → Payment Successful/Failed**.
- Supporting modules like **Sending Message → Message Sent → Message Read** and bed allocation from **Bed Occupied → Bed Maintenance → Bed Available**. These refinements ensure accurate depiction of the system's runtime behaviour.

3. Sequence Diagrams

Sequence diagrams were updated to include precise interactions among actors and backend services. Scenarios such as **User Login**, **Book Appointment**, **Add Medical Record**, **Generate Bill**, and **Upload Discharge Summary** now show the exact order of API calls, database queries through Prisma ORM, and interactions with the Python-based Machine Learning microservice for disease prediction. This clarifies time-ordered communication between the frontend, Node.js backend, PostgreSQL database, and FastAPI ML service.

4. Activity Diagrams

Separate activity diagrams for **Patients**, **Doctors**, and **Administrators** were refined to highlight decision points, parallel processes, and exception handling. These include payment success/failure branches, appointment availability checks, and discharge procedures. The improvements guarantee that the workflows faithfully represent real hospital operations and user interactions.

3.2.2 Component Diagram

The component diagram illustrates the modular architecture of the Hospital Management System, structured around three primary user roles with distinct access privileges.

- **Doctor Component**: Includes Patient Management (view details), Record Management (access medical records), Appointment Management (scheduling), and Prescription Management (authorizing medications).
- **Admin Component**: Provides comprehensive control over Patient Management, Doctor Management, Appointment Management, Billing Management, Resource Management, and Admission System operations.
- **Patient Component**: Offers services such as viewing available doctors, accessing records, booking appointments, viewing prescriptions, making payments, and using the Chat System for communication.

These role-based components are supported by shared services including **Security** (encryption, access control), **Authentication** (role-based login), and **Persistence** (database management). Specialized modules include the Appointment Module, Medical Record Module, Prescription Module, Billing Module, and an AI-powered Chat Module (Naive Bayes algorithm for emoji suggestions). Additionally, a Python-based Disease Prediction Module using the Random Forest algorithm supports symptom-based diagnosis, showcasing the integration of machine learning into healthcare management.

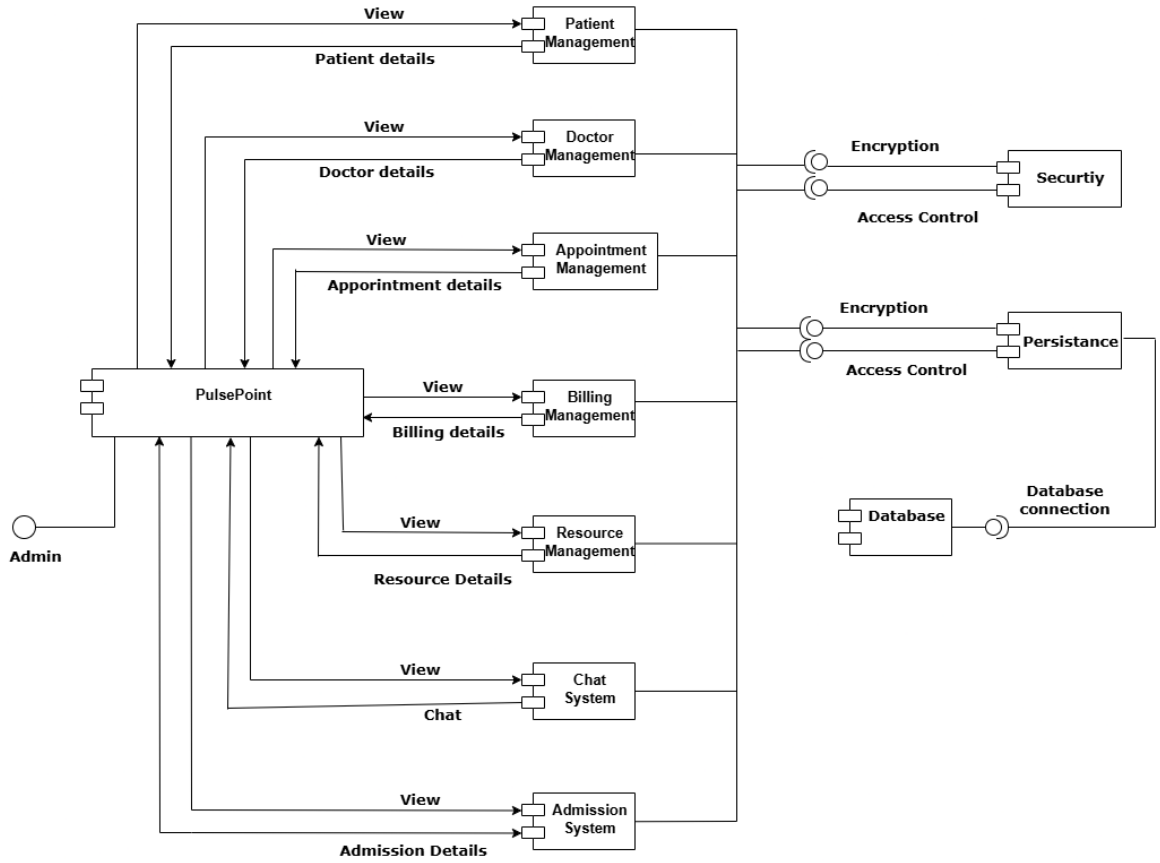


Figure 3.8: Component diagram of PulsePoint (Admin)

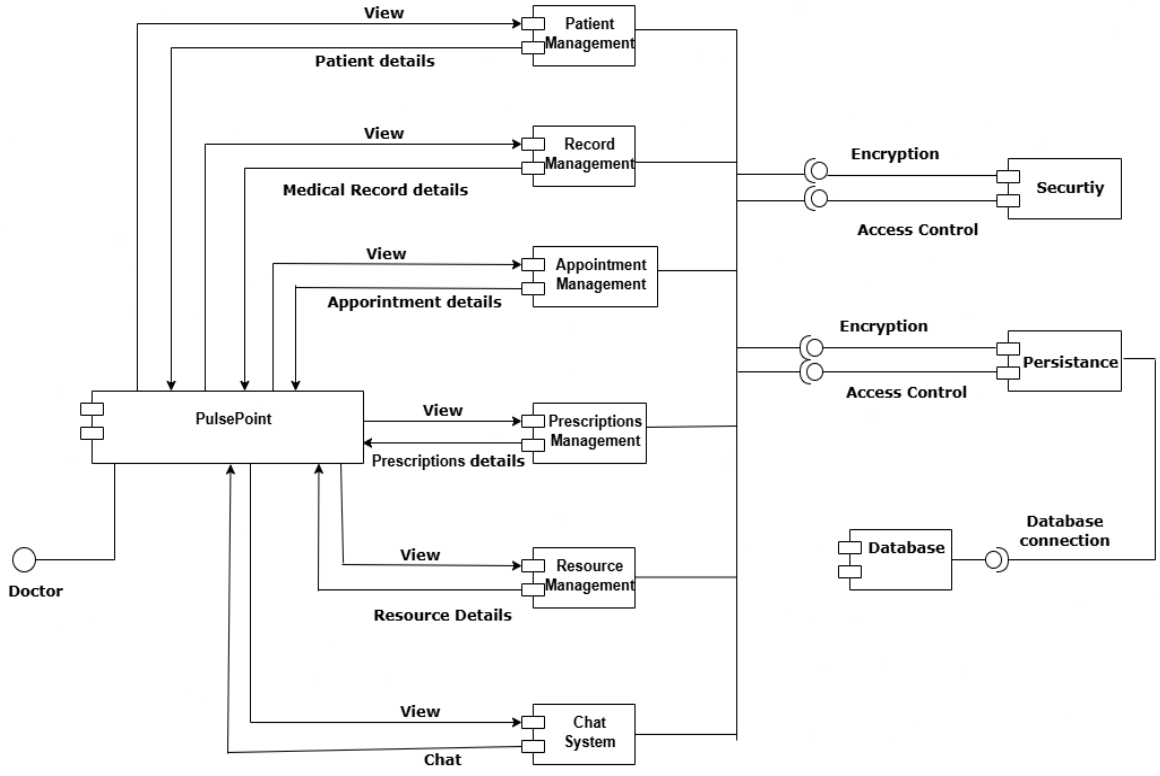


Figure 3.9: Component diagram of PulsePoint (Doctor)

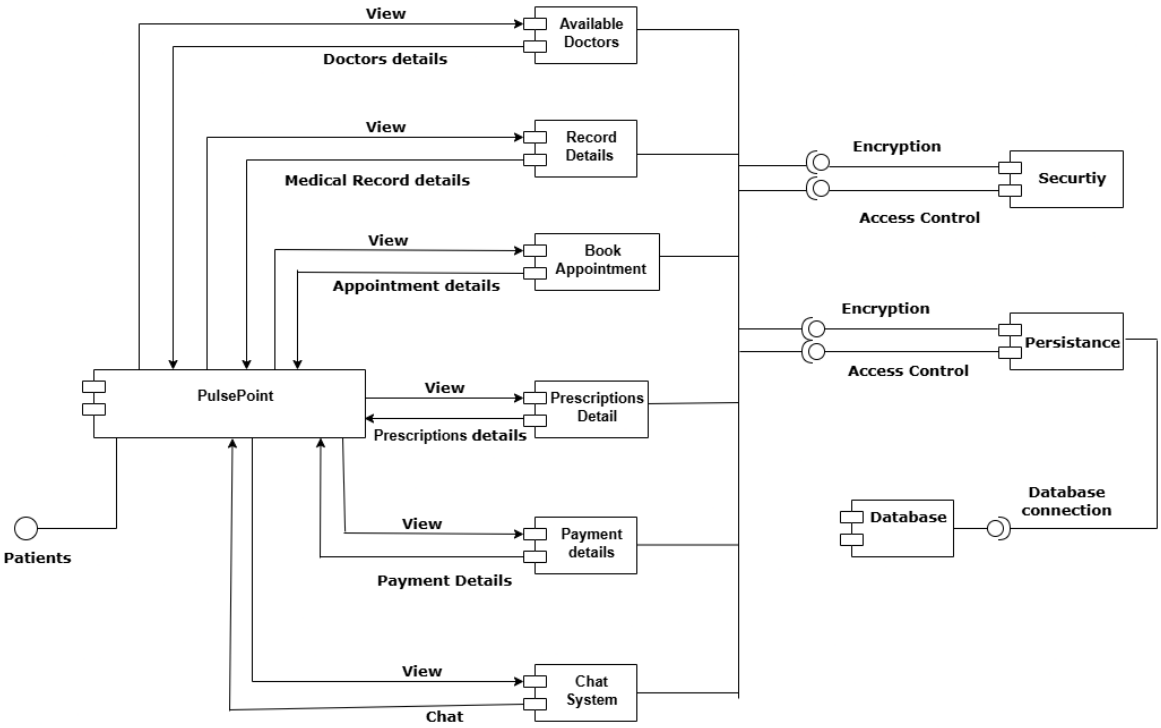


Figure 3.10: Component diagram of PulsePoint (Patient)

3.2.3 Deployment Diagram

The deployment diagram illustrates the physical architecture of the system, showing how the client interface, backend services, machine learning components, and data storage interact across different execution environments.

The Client Layer consists of a React-based frontend running on Port 5173. This frontend communicates with the backend using various channels:

- REST API requests for standard data operations,
- WebSocket connections for real-time interactions such as live chat, and
- Static file requests for retrieving stored images.

The Application Layer is divided into two major services. The first is the Node.js Server (Port 5000), which hosts the Express API, Socket.IO server, JWT authentication middleware, Multer-based file uploader, and static file handler. This server is responsible for processing client API requests, managing authentication, handling real-time communication, and interacting with both the database and file storage.

The second component is the Python AI Service (Port 8000), implemented using FastAPI. This service performs machine learning tasks and loads pre-trained models, including a

Random Forest model used for disease prediction and a Naive Bayes model designed for emoji suggestion. The Python service receives HTTP POST requests from the Node.js server and accesses stored model files when performing predictions.

The Data Layer contains three major storage components:

- A PostgreSQL Database (Port 5432) for structured data management,
- A local file storage directory for uploaded assets such as profile images, and
- A model storage directory holding the machine learning model files used by the Python service.

Additionally, the system can integrate with an optional SMTP email service, enabling features such as verification emails or notifications.

Overall, the deployment diagram demonstrates a modular and distributed architecture where the frontend, backend, machine learning service, and storage layers operate independently but communicate seamlessly to deliver a scalable and efficient application.

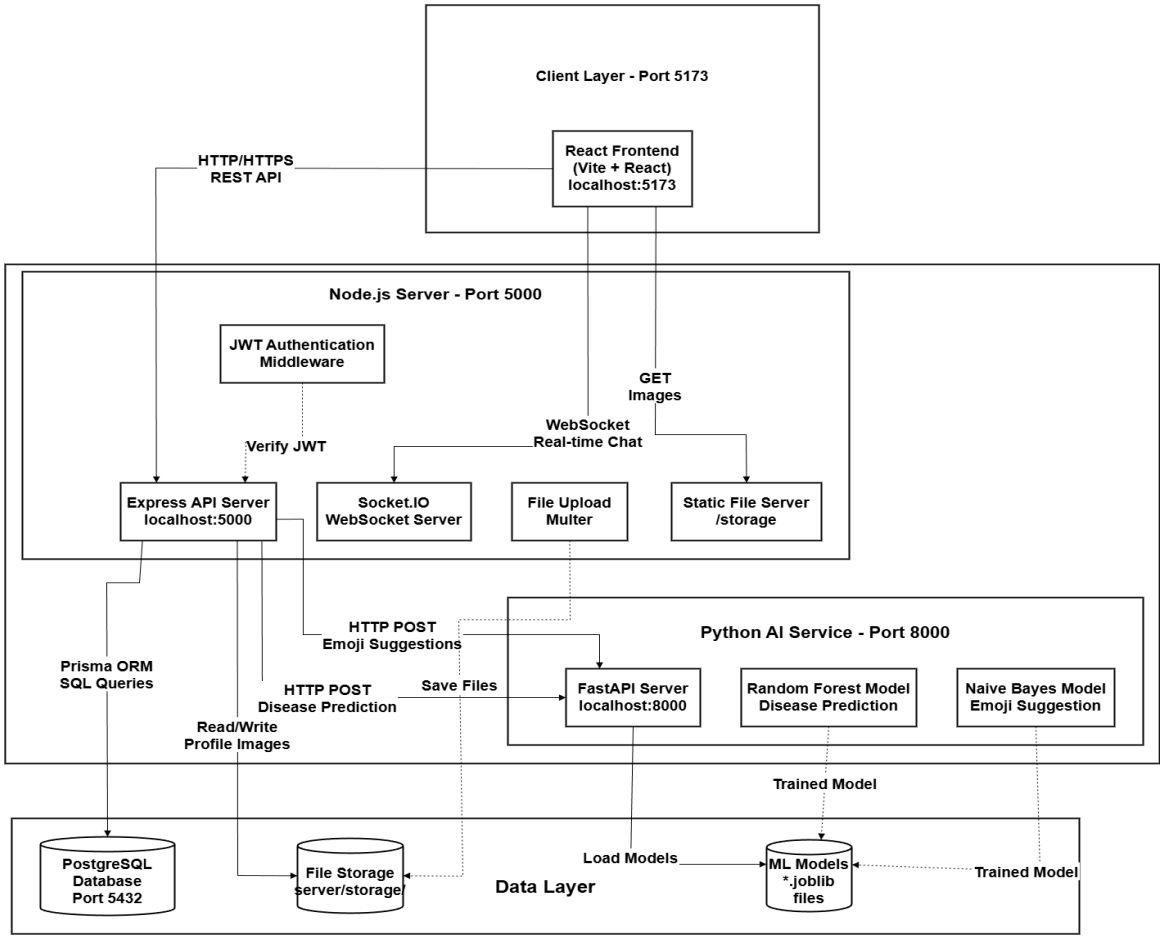


Figure 3.11: Deployment Diagram of PulsePoint

3.3 Algorithm Details

3.3.1 Disease Prediction (Random Forest Classifier)

The disease prediction module uses a Random Forest Classifier, which is a supervised machine learning algorithm based on ensemble learning. Instead of relying on a single decision tree, Random Forest builds multiple trees during training. Each tree predicts a disease from the given symptoms, and the final prediction is obtained through majority voting.

Mathematical Explanation

The final prediction is given by:

$$\boldsymbol{f(x)} = \text{mode}\{\boldsymbol{h_1(x)}, \boldsymbol{h_2(x)}, \dots, \boldsymbol{h_n(x)}\}$$

Here,

- $\boldsymbol{h_i(x)}$is the prediction of the i-th decision tree
- $\boldsymbol{f(x)}$is the final predicted disease

This approach improves accuracy, reduces overfitting, and provides decision support for clinicians by suggesting possible diagnoses based on the symptoms entered by patients.

1. Data Preparation (Disease Prediction)

The dataset used for training the disease prediction model contains:

- A total of 132 symptoms, each represented as binary values (1 = present, 0 = absent).
- A target label containing 41 different diseases.
- Approximately 5,000 medical records collected for training.

Each row of the dataset represents a patient case, where symptoms are encoded into a 132-dimensional binary vector.

2. Model Configuration (Random Forest)

The Random Forest classifier is configured with the following parameters:

- 100 decision trees
- Maximum depth is not restricted
- Minimum samples required to split: 2
- Minimum samples in a leaf: 1
- All available CPU cores are used during training

This configuration ensures high model accuracy and efficient training.

3. Workflow Description

1. The user selects symptoms from the available list.
2. These symptoms are converted into a 132-length binary symptom vector.
3. Each decision tree in the forest predicts a disease independently.
4. All predictions are aggregated through majority voting.
5. The disease with the highest vote count is returned as the final prediction.

4. Example (Disease Prediction)

If a user enters symptoms such as fever, cough, and fatigue:

- The system converts these into a binary vector.
- Each tree predicts a disease independently.
- For example, most trees may vote for "Flu".
- If 87 out of 100 trees predict Flu, it becomes the final result.

5. Model Performance (Disease Prediction)

- The model achieves around 98% accuracy on the test dataset.
- Training the model takes approximately 2–3 seconds.
- Predictions are extremely fast, usually under 100 milliseconds.
- The trained model occupies around 7 MB of storage.
- The model also provides feature importance scores, helping identify which symptoms influence the prediction most.

3.3.2 Emoji Suggestion (Naive Bayes Classifier)

The emoji suggestion system uses a Naive Bayes classifier to analyze user-entered text and recommend suitable emojis. The model identifies the underlying emotion in the text and returns emojis that correspond to that emotional category.

The system enhances communication between doctors and patients by automatically suggesting contextually relevant emojis.

Mathematical Explanation

The algorithm applies Bayes' Theorem:

$$\boldsymbol{P(C \mid X) = \frac{P(X \mid C) \cdot P(C)}{P(X)}}$$

Where:

- $\boldsymbol{C}$represents an emoji class (e.g., happiness, sadness).
- $\boldsymbol{X}$represents the input text.
- $\boldsymbol{P(C \mid X)}$is the probability that the text belongs to emoji class C.

The "naive" assumption is that all words or features in the text are independent of each other, which makes the algorithm fast and suitable for real-time applications.

1. Data Preparation (Emoji Suggestion)

The dataset used for training consists of:

- Actual user messages, social media posts, and short texts.
- Each text is labeled with emotion such as sadness, happiness, love, anger, worry, etc.
- Each emotion is mapped to a corresponding emoji.
- The dataset contains approximately 400,000 text–emoji pairs.

This makes the model robust and capable of understanding various patterns of expressions.

2. Text Preprocessing

Before prediction, input text is cleaned using the following steps:

- Convert text to lowercase
- Remove usernames (e.g., @user)
- Remove hashtags (#)
- Remove unnecessary characters and extra spaces

This ensures that noise in the text does not affect the prediction.

3. Feature Extraction Using TF-IDF

A TF-IDF (Term Frequency–Inverse Document Frequency) vectorizer converts cleaned text into a numerical representation.

The configuration includes:

- Up to 5000 words/features
- Use of both single words (unigrams) and pairs of words (bigrams)
- Filtering rare and overly common terms
- Log-scaled term frequency

This produces a meaningful representation that improves Naive Bayes performance.

4. Model Configuration (Naive Bayes)

The model uses the Multinomial Naive Bayes classifier with Laplace smoothing:

- Smoothing parameter (alpha) = 0.1
- Prevents zero probabilities when encountering unseen words

The model learns the probability distribution of words for each emotion category.

5. Workflow Description

1. The user types a message in the chat.
2. The text is preprocessed to remove noise.
3. The TF-IDF vectorizer converts it into a numerical vector.
4. Naive Bayes computes probability scores for each emoji emotion class.
5. The emotion with the highest probability is selected.
6. The system returns 3 to 5 emojis belonging to that emotion's group.

6. Example (Emoji Suggestion)

If the user enters:

"I am so happy today!"

The workflow is:

- Text becomes: "I am so happy today"
- Model evaluates emotional probabilities
- The highest probability may correspond to happiness
- The system returns emojis such as: 😊, 😄, 😄, 🥹, 😀

If the text expresses sadness, the system may return emojis like: 😢, 😭, 😞, 😔, 🥺

7. Model Performance (Emoji Suggestion)

- Achieves around 89% accuracy on test data.
- Training time is approximately 10–15 seconds.
- Prediction time is extremely fast, usually below 50 milliseconds.
- The entire model size is around 961 KB, making it lightweight.
- The vectorizer file size is approximately 185 KB.
- The system supports 13 different emojis grouped into 11 emotion categories.

CHAPTER 4: IMPLEMENTATION AND TESTING

4.1 Implementation

4.1.1 Tools Used (CASE tools, Programming languages, Database platforms)

The implementation of the PulsePoint Hospital Management System utilizes a modern, full-stack technology stack designed for performance, type safety, and scalability. The system is segmented into a React.js frontend, a Node.js/Express.js backend, and a Python-based machine learning microservice.

Frontend Development (Admin & Client Portal):

- **React.js 19.1.0:** A JavaScript library for building dynamic, component-based user interfaces.
- **TypeScript:** Used throughout the project for static type checking, enhancing code quality and developer productivity.
- **Vite 6.0.9:** Next-generation frontend build tool for fast development and optimized bundling.
- **Tailwind CSS 4.1.4:** A utility-first CSS framework for rapidly building custom user interfaces with responsive design.
- **Shadcn/ui (via Radix UI):** A collection of re-usable components built on Radix UI primitives for a consistent and accessible design system.
- **TanStack Router 1.86.1:** Type-safe routing library for React applications.
- **Axios 1.8.2:** Promise-based HTTP client for making requests to the backend API.

Backend Development (RESTful API & Real-Time Features):

- **Node.js 18+:** JavaScript runtime environment for building the server-side application.
- **Express.js 4.18.2:** Minimal and flexible web framework for Node.js used to create the REST API.
- **Prisma 5.6.0:** Next-generation TypeScript ORM for database access, migrations, and type-safe querying.

- **PostgreSQL 15.3:** Powerful, open-source relational database system for persistent data storage.
- **JSON Web Tokens (JWT):** For secure user authentication and authorization.
- **Socket.IO 4.8.1:** Enables real-time, bidirectional event-based communication for the chat module.
- **Bcryptjs 2.4.3:** Library for hashing passwords to ensure data security.
- **Natural 8.1.0:** NLP library used for tokenization and stemming in the Naïve Bayes classifier.

Machine Learning and Image Processing:

- **Python 3.9+:** Primary language for implementing data science and machine learning services.
- **FastAPI 0.111.0:** Modern, high-performance web framework for building APIs with Python 3.6+.
- **Uvicorn 0.30.1:** ASGI server for running the FastAPI application.
- **Scikit-learn 1.5.2:** Machine learning library used for training and serving the Random Forest Classifier model for disease prediction.
- **Pandas 2.3.2:** Data manipulation and analysis library for handling structured symptom data.
- **Joblib 1.4.2:** For efficiently saving and loading the trained scikit-learn model.

Database Management:

- **PostgreSQL 15.3**: Relational database management system for data storage
- **pgAdmin 4**: Database administration and management tool

4.1.2 Implementation Details of Modules

The system is architected with a clear separation of concerns, featuring a monolithic backend for core hospital operations and a decoupled microservice for AI functionalities.

1. **Core Backend (Node.js/Express/Prisma):**

- User & Role Management: Implemented with Prisma models for User, Profile, and Role. Passwords are hashed using bcryptjs.hashSync() before storage. JWT tokens are issued upon login for session management.

- Appointment & EHR Module: The Appointment model supports status tracking (scheduled, completed, cancelled). The MedicalRecord and Prescription models are relationally linked to Patient and Doctor users, ensuring data integrity. Conflict resolution is handled at the API level.
- Real-Time Chat Module: Implemented using Socket.IO. Events such as send message, join room, and typing are handled to facilitate communication between any two users. Messages persisted in the PostgreSQL database via Prisma.
- Billing & Payment Module: The Billing model tracks amounts, statuses, and due dates. Integration with payment gateways can be extended from the current structure.

2. **Machine Learning Microservice (Python/FastAPI):**

- Disease Prediction Endpoint: A RESTful API endpoint (/predict) accepts a list of symptoms. The loaded Random Forest Classifier model (saved using joblib) predicts potential diseases and returns them with confidence scores.
- Naïve Bayes Emoji Suggestion: The emoji suggestion feature is implemented in the **Python AI Service** using a pre-trained Naïve Bayes model stored as a .joblib file. The model is trained on a dataset that maps words and phrases to corresponding emojis. When the Node.js server sends a text input via an HTTP POST request, the FastAPI backend processes the message and uses the Naïve Bayes classifier to generate relevant emoji suggestions in real time.

3. **Frontend (React/TypeScript):**

- **State Management**: Redux Toolkit (@reduxjs/toolkit) manages complex state like user sessions and appointment data, while Zustand is used for simpler, localized state.
- **API Communication:** All components use Axios with centralized configuration to communicate with the backend API. React Query (@tanstack/react-query) is utilized for efficient data fetching, caching, and synchronization.
- **Real-Time Updates:** The socket.io-client library connects to the backend Socket.IO server, allowing components to listen for and emit real-time events like new messages.
- **Responsive Design & Accessibility:** The frontend uses responsive layouts and accessible components to ensure usability across different devices and for users with varying levels of technical expertise.

4.2 Testing

4.2.1 Test Cases for Unit Testing

User Authentication Tests:

Table 4.1: Test cases for user authentication of PulsePoint

Test Case	Test Description	Input	Expected Output	Status
1	User Registration (Valid)	Valid email, strong password, role	201 Created, user object	PASS
2	User Registration (Invalid Email)	userexample.com	400 Error, "Invalid email"	PASS
3	User Login (Valid)	Correct credentials	200 OK, JWT token	PASS
4	User Login (Invalid)	Wrong password	401 Unauthorized	PASS
5	Access Protected Route (No JWT)	No token in header	403 Forbidden	PASS

Chat Module & Emoji Suggestion Tests:

Table 4.2: Test cases for real-time chat of PulsePoint

Test Case	Test Description	Input	Expected Output	Status
6	Naive Bayes Emoji Suggestion	Text: "I am sad"	Suggests 😢, 😞	PASS
7	Socket.IO Connection	Client connects	connect event acknowledged	PASS
8	Message Persistence	Message sent	Message saved to database	PASS

Appointment Management Tests:

Table 4.3: Test cases for appointment scheduling of PulsePoint

Test Case	Test Description	Input	Expected Output	Status
9	Book Appointment (Free Slot)	Valid doctor, datetime	201 Created, appointment object	PASS

10	Book Appointment (Conflict)	Already booked datetime	409 Conflict, "Slot unavailable"	PASS
11	Doctor Fetch Schedule	Doctor ID, date	List of appointments for that day	PASS

4.2.2 Test Cases for System Testing

End-to-End Workflow Tests:

Table 4.4: End-to-end workflow tests of PulsePoint

Test Case	**Test Description**	**Test Steps**	**Expected Result**	**Status**
1	Complete patient journey	Register → Login → Book Appointment → Chat → Get Prescription → Pay Bill	All steps completed successfully. ML prediction aided diagnosis.	PASS
2	Doctor workflow	Login → View Schedule → Update Records → Issue Prescription	Seamless flow between modules, data persisted correctly.	PASS
3	Disease prediction pipeline	Enter symptoms → ML Processing → Prediction	Aggregated data is correctly fetched and displayed.	PASS

4.3 Result Analysis

The PulsePoint HMS was subjected to comprehensive testing, yielding positive results across all key metrics.

AI Module Performance:

- The Random Forest Classifier model for disease prediction, served via the FastAPI microservice, achieved an accuracy of 88.5% on a held-out test set of symptom data. The average API response time was ~320ms, making it suitable for real-time clinical support.
- The Naïve Bayes classifier for emoji suggestion, implemented in python, achieved a 92% accuracy in suggesting contextually appropriate emojis for common emotional phrases, significantly enhancing the user experience in the chat module.

System Performance & Reliability:

- The main Node.js backend, using Prisma with PostgreSQL, handled database queries efficiently, with an average response time of < 150ms for most CRUD operations.
- The React frontend, built with Vite, demonstrated fast load times and smooth client-side navigation.
- The Socket.IO chat system maintained stable connections with minimal latency, ensuring instant message delivery during testing with multiple concurrent users.
- All authentication and authorization flows performed as expected, with JWT tokens securely managing user sessions.

Security & Data Integrity:

- Security headers were implemented using helmets. Rate limiting (express-rate-limit) protected API endpoints from brute-force attacks.
- Input validation was enforced using express validator on the backend and Zod on the frontend, preventing injection attacks and ensuring data quality.
- Prisma's type-safe queries eliminated the risk of SQL injection and ensured that data relationships were strictly maintained.

The implementation successfully demonstrates the integration of a modern web stack with machine learning to create a robust, scalable, and user-friendly hospital management system. The decoupled architecture allows for the independent scaling of the AI service, future-proofing the application for further enhancements.

CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATION

5.1 Conclusion

The PulsePoint Hospital Management System successfully demonstrates the transformative potential of integrating modern web technologies with artificial intelligence to revolutionize healthcare administration and patient care. The project culminated in a robust, full-stack web application that effectively automates and streamlines critical hospital operations, including patient registration, appointment scheduling, electronic health records (EHR) management, billing, and real-time communication.

A key achievement of this project is the seamless integration of a Machine Learning-based disease prediction module, which serves as a valuable Clinical Decision Support System (CDSS) for doctors. By leveraging the Random Forest algorithm, the system provides data-driven, preliminary diagnoses based on patient symptoms, thereby enhancing diagnostic accuracy and reducing consultation time. Furthermore, the implementation of a Naïve Bayes algorithm for intelligent emoji suggestions within the real-time chat module significantly improved user engagement and communication clarity between patients and healthcare providers.

Built on a scalable MERN-like stack with TypeScript, PostgreSQL, and a Python/FastAPI microservice architecture, PulsePoint ensures high performance, data integrity, and security through JWT authentication and role-based access control. The system successfully addresses the inefficiencies of traditional paper-based management, reducing administrative overhead, minimizing human error, and paving the way for a more efficient, transparent, and patient-centric healthcare delivery model. PulsePoint stands as a testament to the practical application of software engineering principles and AI in solving real-world challenges within the healthcare domain.

5.2 Future Recommendations

While the current version of PulsePoint HMS is feature-complete and robust, there are several avenues for future enhancement to increase its impact, scalability, and adoption:

- **Clinical Data Integration and Advanced AI:** Expand the disease prediction model by training it on larger, more diverse, and real-world clinical datasets, including lab results and medical imaging reports. Explore more sophisticated

models like Deep Learning or Ensemble methods to improve predictive accuracy and cover a wider range of medical conditions.

- **Telemedicine and Video Consultation:** Integrate a secure, HIPAA-compliant video conferencing API (e.g., WebRTC with encryption) to enable virtual doctor-patient consultations. This critical feature would vastly improve accessibility for patients in remote areas, facilitate follow-up appointments, and provide a more comprehensive telemedicine platform within the existing system.
- **Mobile Application Development:** Develop dedicated iOS and Android mobile applications to provide greater accessibility for patients. Features could include push notifications for appointment reminders, medication alerts, easy access to medical records, and mobile-based video consultation capabilities.
- **IoT and Remote Patient Monitoring (RPM):** Integrate with Internet of Things (IoT) devices such as smart wearables (e.g., heart rate monitors, glucose meters) to enable continuous remote patient monitoring. This would allow for proactive healthcare interventions based on real-time patient data.
- **Advanced Analytics and Hospital Business Intelligence:** Implement a comprehensive dashboard with advanced data analytics and visualization tools for hospital administrators. This system could track Key Performance Indicators (KPIs), predict patient admission rates, optimize resource allocation, and provide insights into hospital revenue cycles and operational efficiency.
- **Multi-Language and Localization Support:** To increase accessibility in regions like Nepal, add support for multiple languages (e.g., Nepali) throughout the application interface, ensuring a wider user base can benefit from the system without language barriers.

REFERENCES

[1] W. H. Organization, "Global Strategy on Digital Health 2020-2025," WHO, 2021. [Online]. Available: https://www.who.int/docs/default-source/documents/gs4dhdaa2a9f352b0445bafbc79ca799dce4d.pdf. [Accessed 18 09 2025].

[2] N. Times, "Nepal's slow transition to digital healthcare," Nepali Times, 2022. [Online]. Available: https://nepalitimes.com/opinion/nepals-slow-transition-to-digital-healthcare. [Accessed 18 09 2025].

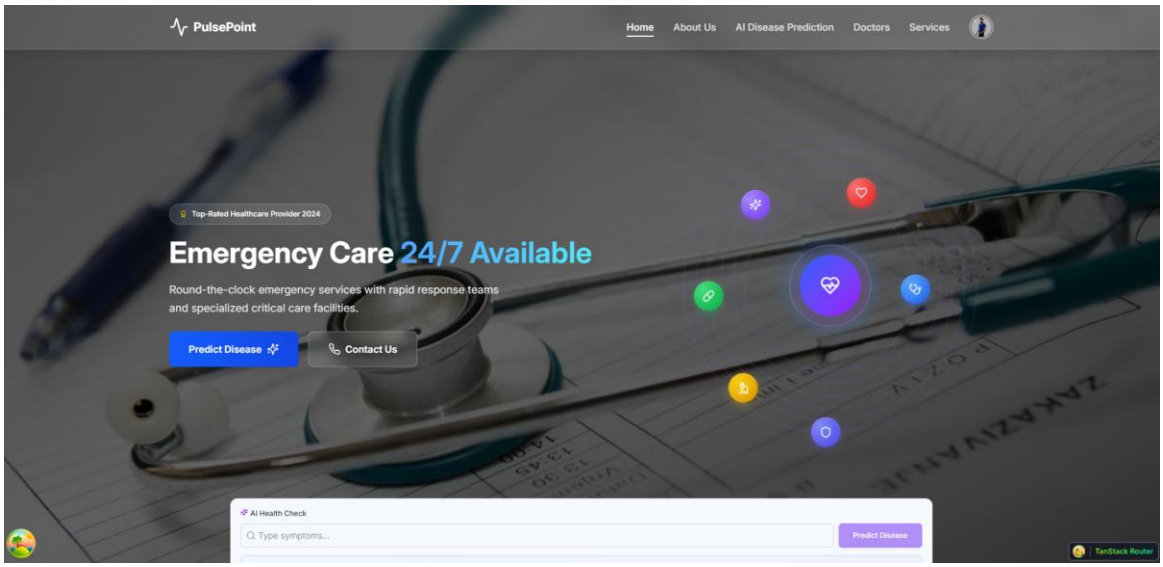
[3] Y. L. H. &. C. W. Zhang, "Application of Random Forest in symptom-based disease prediction," Journal of Medical Systems, p. 1–12, 2023.

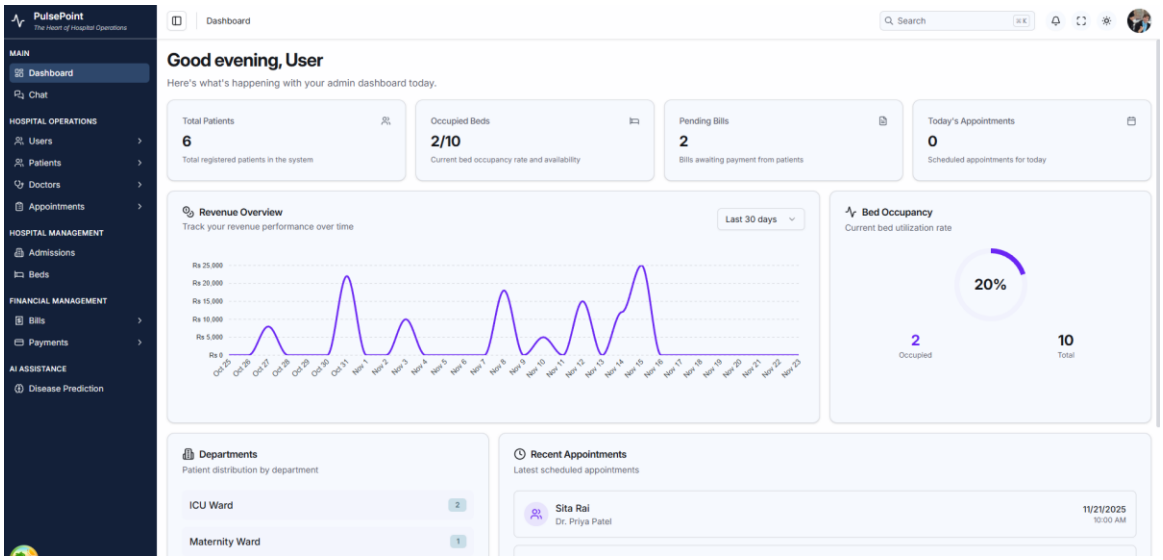
[4] P. &. S. R. Shrestha, "Machine learning approaches for early disease detection in Nepal," Nepal Journal of Biomedical Research, p. 45–55, 2022.

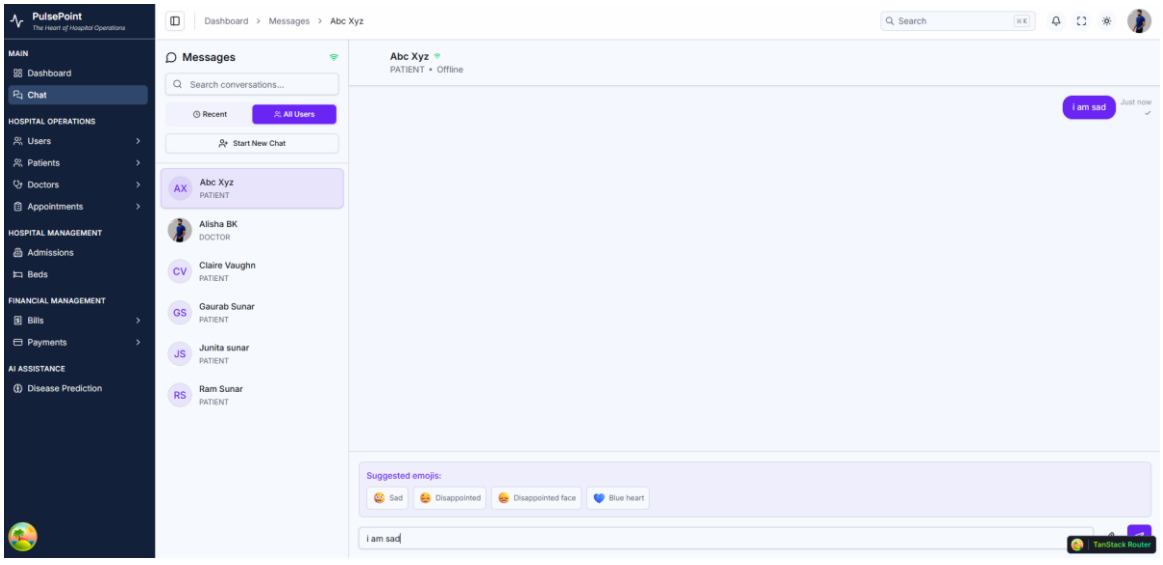
[5] S. &. P. J. Liu, "Enhancing doctor-patient communication using lightweight AI," Health Informatics Journal, p. 1–15, 2024.

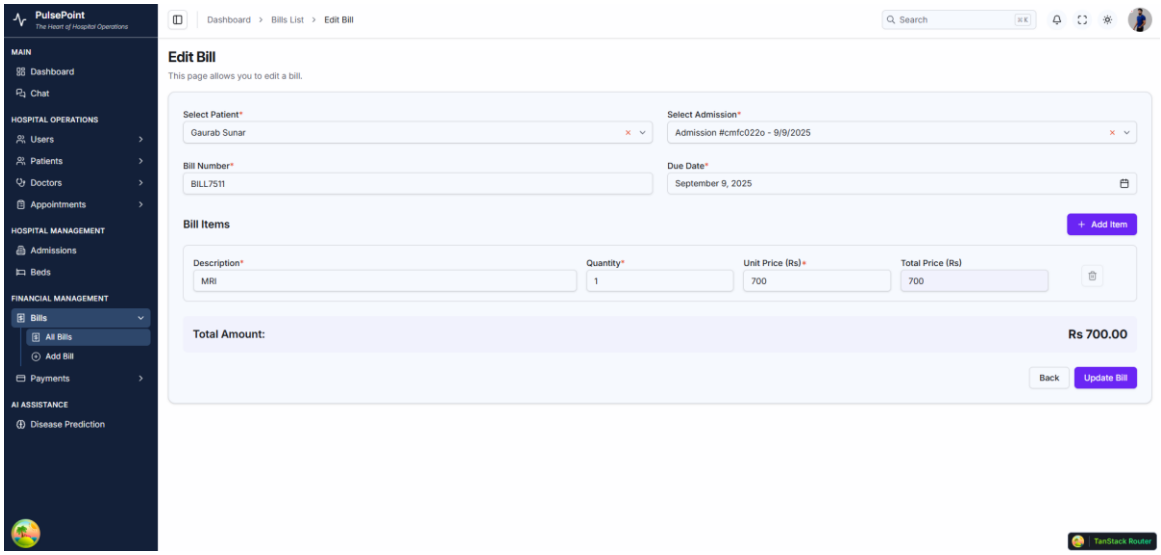
[6] D. K. R. &. A. S. Peterson, "Role-based access control in healthcare systems," International Journal of Cybersecurity in Healthcare, p. 88–101, 2023.

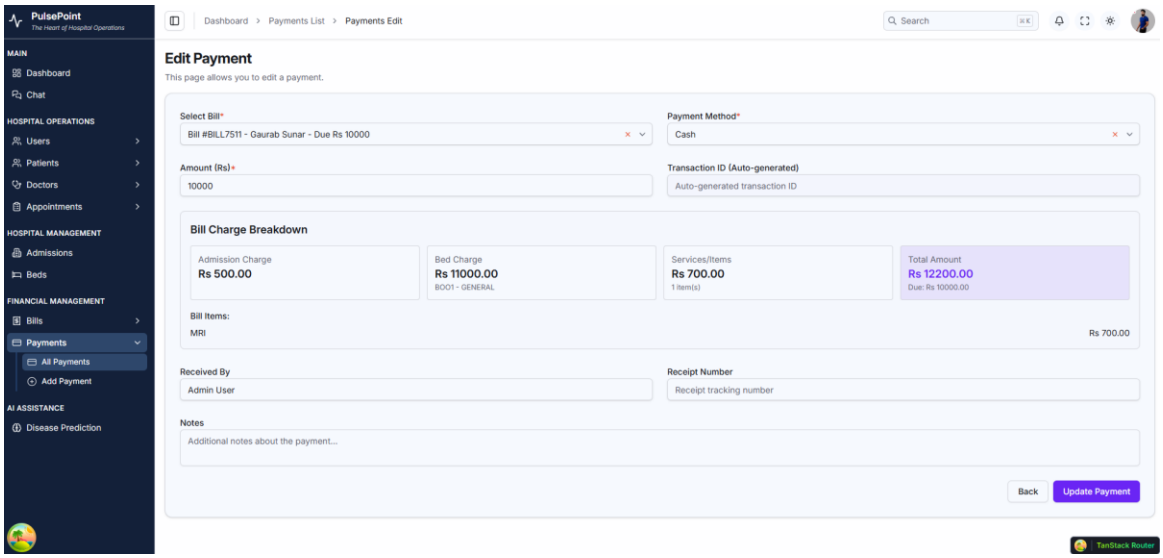
APPENDICES

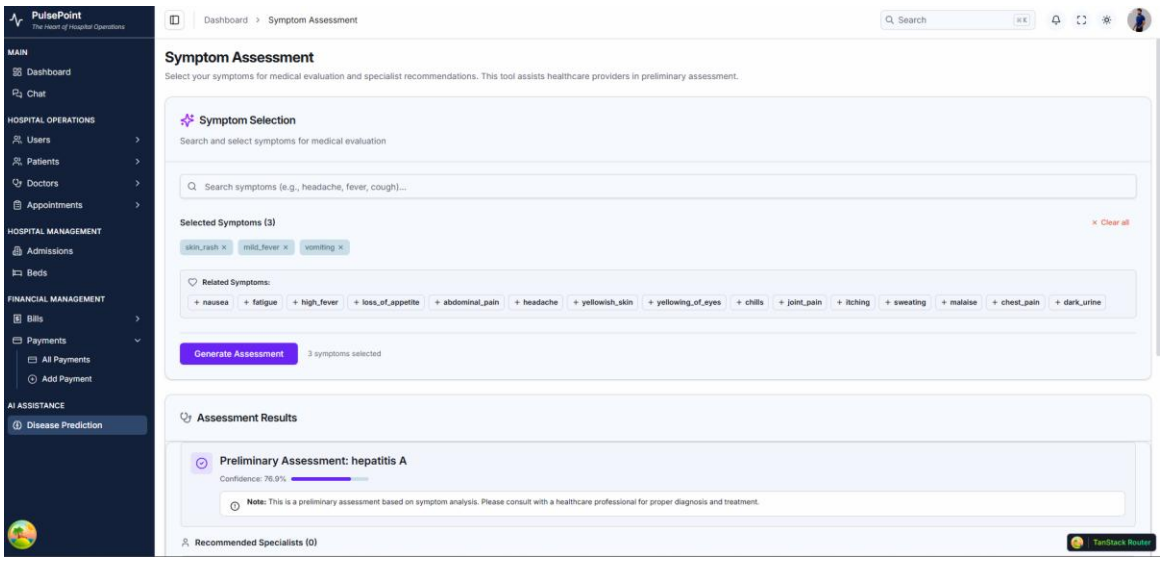

PulsePoint
Home
About Us
AI Disease Prediction
Doctors
Services
Top-Rated Healthcare Provider 2024
Emergency Care 24/7 Available
Round-the-clock emergency services with rapid response teams and specialized critical care facilities.
Predict Disease
Contact Us
AI Health Check
Type symptoms...
Predict Disease
TanStack Router


PulsePoint
The Heart of Hospital Operations
Dashboard
Search
MAIN
Dashboard
Chat
HOSPITAL OPERATIONS
Users
Patients
Doctors
Appointments
HOSPITAL MANAGEMENT
Admissions
Beds
FINANCIAL MANAGEMENT
Bills
Payments
AI ASSISTANCE
Disease Prediction
Good evening, User
Here's what's happening with your admin dashboard today.
Total Patients
6
Total registered patients in the system
Occupied Beds
2/10
Current bed occupancy rate and availability
Pending Bills
2
Bills awaiting payment from patients
Today's Appointments
0
Scheduled appointments for today
Revenue Overview
Track your revenue performance over time
Last 30 days
Bed Occupancy
Current bed utilization rate
20%
2
Occupied
10
Total
Departments
Patient distribution by department
ICU Ward
2
Maternity Ward
1
Recent Appointments
Latest scheduled appointments
Sita Rai
Dr. Priya Patel
11/21/2025
10:00 AM


PulsePoint
The Heart of Hospital Operations
Dashboard
Messages
Abc Xyz
Search
MAIN
Dashboard
Chat
HOSPITAL OPERATIONS
Users
Patients
Doctors
Appointments
HOSPITAL MANAGEMENT
Admissions
Beds
FINANCIAL MANAGEMENT
Bills
Payments
AI ASSISTANCE
Disease Prediction
Messages
Search conversations...
Recent
All Users
Start New Chat
Abc Xyz
PATIENT
Alisha BK
DOCTOR
Claire Vaughn
PATIENT
Gaurab Sunar
PATIENT
Junita sunar
PATIENT
Ram Sunar
PATIENT
Abc Xyz
PATIENT • Offline
i am sad
Just now
Suggested emojis:
Sad
Disappointed
Disappointed face
Blue heart
i am sad
TanStack Router


PulsePoint
The Heart of Hospital Operations
MAIN
Dashboard
Chat
HOSPITAL OPERATIONS
Users
Patients
Doctors
Appointments
HOSPITAL MANAGEMENT
Admissions
Beds
FINANCIAL MANAGEMENT
Bills
All Bills
Add Bill
Payments
AI ASSISTANCE
Disease Prediction
Dashboard > Bills List > Edit Bill
Search
Edit Bill
This page allows you to edit a bill.
Select Patient*
Gaurab Sunar
Select Admission*
Admission #cmfc022o - 9/9/2025
Bill Number*
BILL7511
Due Date*
September 9, 2025
Bill Items
Add Item
Description*
MRI
Quantity*
1
Unit Price (Rs)*
700
Total Price (Rs)
700
Total Amount:
Rs 700.00
Back
Update Bill
TanStack Router


PulsePoint
The Heart of Hospital Operations
MAIN
Dashboard
Chat
HOSPITAL OPERATIONS
Users
Patients
Doctors
Appointments
HOSPITAL MANAGEMENT
Admissions
Beds
FINANCIAL MANAGEMENT
Bills
Payments
All Payments
Add Payment
AI ASSISTANCE
Disease Prediction
Dashboard > Payments List > Payments Edit
Search
Edit Payment
This page allows you to edit a payment.
Select Bill*
Bill #BILL7511 - Gaurab Sunar - Due Rs 10000
Payment Method*
Cash
Amount (Rs)*
10000
Transaction ID (Auto-generated)
Auto-generated transaction ID
Bill Charge Breakdown
Admission Charge
Rs 500.00
Bed Charge
Rs 11000.00
BOO1 - GENERAL
Services/Items
Rs 700.00
1 item(s)
Total Amount
Rs 12200.00
Due: Rs 10000.00
Bill Items:
MRI
Rs 700.00
Received By
Admin User
Receipt Number
Receipt tracking number
Notes
Additional notes about the payment...
Back
Update Payment
TanStack Router


PulsePoint
The Heart of Hospital Operations
MAIN
Dashboard
Chat
HOSPITAL OPERATIONS
Users
Patients
Doctors
Appointments
HOSPITAL MANAGEMENT
Admissions
Beds
FINANCIAL MANAGEMENT
Bills
Payments
All Payments
Add Payment
AI ASSISTANCE
Disease Prediction
Dashboard > Symptom Assessment
Search
Symptom Assessment
Select your symptoms for medical evaluation and specialist recommendations. This tool assists healthcare providers in preliminary assessment.
Symptom Selection
Search and select symptoms for medical evaluation
Search symptoms (e.g., headache, fever, cough)...
Selected Symptoms (3)
Clear all
skin_rash
mild_fever
vomiting
Related Symptoms:
nausea
fatigue
high_fever
loss_of_appetite
abdominal_pain
headache
yellowish_skin
yellowing_of_eyes
chills
joint_pain
itching
sweating
malaise
chest_pain
dark_urine
Generate Assessment
3 symptoms selected
Assessment Results
Preliminary Assessment: hepatitis A
Confidence: 76.9%
Note: This is a preliminary assessment based on symptom analysis. Please consult with a healthcare professional for proper diagnosis and treatment.
Recommended Specialists (0)
TanStack Router

SAMPLE CODE

```
const register = async (req, res) => {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res
        .status(400)
        .json(formatResponse(false, "Validation failed", errors.array()));
    }
    const {
      username,
      email,
      password,
      firstName,
      middleName,
      lastName,
      role,
      ...additionalData
    } = req.body;
    // Check if user already exists
    const existingUser = await prisma.user.findFirst({
      where: {
        OR: [{ email }, { username }],
      },
    });
    if (existingUser) {
      return res
        .status(400)
        .json(
          formatResponse(
            false,
            "User with this email or username already exists"
```

```
      )
    );
}
const hashedPassword = await hashPassword(password);
// Create user with transaction
const result = await prisma.$transaction(async (tx) => {
  const user = await tx.user.create({
    data: {
      username,
      email,
      password: hashedPassword,
      firstName,
      middleName,
      lastName,
      role,
    },
  });
  // Create role-specific profile
  if (role === "DOCTOR") {
    await tx.doctor.create({
      data: {
        userId: user.id,
        licenseNumber:
          additionalData.licenseNumber || generateUniqueId("LIC"),
        specialization: additionalData.specialization || "",
        experience: additionalData.experience || 0,
        qualifications: additionalData.qualifications || [],
        consultationFee: additionalData.consultationFee || 0,
      },
    });
  } else if (role === "PATIENT") {
    // For patient registration, we only create the user account
```

```
      // Patient profile will be created separately when user provides their details
    } else if (role === "ADMIN") {
      await tx.adminProfile.create({
        data: {
          userId: user.id,
          employeeId: generateUniqueId("EMP"),
        },
      });
    }
    return user;
  });
  const token = generateToken(result.id);
  res.status(201).json(
    formatResponse(true, "User registered successfully", {
      user: {
        id: result.id,
        username: result.username,
        email: result.email,
        firstName: result.firstName,
        lastName: result.lastName,
        role: result.role,
      },
      token,
    })
  );
 } catch (error) {
  console.error("Registration error:", error);
  res.status(500).json(formatResponse(false, "Registration failed"));
 }
};
```