Cairo University

Faculty of Computers and Artificial intelligence

Department of Software Engineering

Medica

ML-powered doctor assistant for customized Medicine Recommendations, Medicine suitability Predictions, and Drug Interaction Checks

Supervised by:

Dr. Cherry Ahmed

Prepared by:

Malak Wael Okasha

Salma Hazem Salah

Nourhan Mohamed Zaki

Jana Adel Sobeih

Nada Yasser Abdelsattar

Graduation Project

Academic Year 2024-2025

Cairo University

Faculty of Computers and Artificial intelligence

Department of Software Engineering

Medica

Supervised by:

Dr. Cherry Ahmed

Implemented by:

Malak Wael Mohamed Okasha	20216101
Nourhan Mohamed Zaki El Gameel	20216113
Salma Hazem Salah	20216047
Jana Mohamed Adel Sobeih	20216031
Nada Yasser Abdelsattar	20216110

Graduation Project

Academic Year 2024-2025

Final Documentation

Table of Contents

List of Figures:

List of Tables:

List of abbreviations

CSV - Comma-Separated Values
EHR - Electronic Health Record
ERD - Entity Relationship Diagram
GERD - Gastroesophageal Reflux Disease
GUI - Graphical User Interface
HbA1c - Hemoglobin A1c (used in diabetes monitoring)
HIPAA - Health Insurance Portability and Accountability Act
HW - Hardware
ML - Machine Learning
NSAIDs - Nonsteroidal Anti-Inflammatory Drugs
RBAC - Role-Based Access Control
SSL - Secure Sockets Layer
SW - Software
TSH - Thyroid Stimulating Hormone
UI/UX - User Interface / User Experience
URL - Uniform Resource Locator

Chapter 1 : Introduction to the main area of the project

1.1 Motivation

The healthcare industry faces numerous challenges when it comes to the accurate and efficient prescribing of medications. One of the most pressing issues is the difficulty experienced by healthcare professionals, particularly fresh medical graduates, in making informed treatment decisions. New medical graduates often lack the experience and in-depth knowledge of the vast array of medications available, as well as the complex interactions between them. This can result in prescribing errors, adverse drug reactions, and suboptimal treatment outcomes for patients. In an increasingly complex medical landscape, the need for real-time, personalized decision support has never been more critical.

Furthermore, the rapid pace of pharmaceutical innovation means that doctors must continuously stay informed about newly introduced drugs and emerging therapies. However, many healthcare professionals struggle to keep up with this flood of new information, especially when pharmaceutical companies do not have a unified platform for disseminating updates to medical practitioners. This creates a significant knowledge gap, which can lead to missed opportunities for improved treatment options and ultimately, poorer patient care.

Additionally, the management of patient medical records is still often carried out manually or through fragmented systems, which can be time-consuming and prone to errors. Healthcare professionals may have difficulty accessing accurate, up-to-date patient information, which can hinder their ability to monitor treatment effectiveness, adjust prescriptions as needed, and provide high-quality care. These inefficiencies put unnecessary strain on healthcare workers and increase the likelihood of errors, leading to compromised patient safety.

Medica was conceived to address these challenges by providing an intelligent, data-driven platform that enables doctors to make better-informed decisions regarding medication prescriptions. By leveraging machine learning algorithms, Medica offers personalized medication recommendations tailored to each patient's unique diagnosis, medical history, and health characteristics. This ensures that doctors have access to accurate, timely, and relevant information, improving the overall quality of care provided.

Medica’s real-time updates from pharmaceutical companies also help bridge the knowledge gap, ensuring that healthcare professionals are always aware of new medications and treatment options. Additionally, the platform’s seamless integration with patient record management allows for accurate tracking of treatment progress and outcomes, which supports data-driven decision-making and enhances the efficiency of the healthcare process.

Ultimately, Medica aims to reduce the pressure on healthcare professionals, particularly new doctors, by providing them with the tools they need to prescribe medications confidently and safely. The platform’s focus on improving decision-making, reducing medication errors, and ensuring up-to-date information will lead to better patient outcomes, a more efficient healthcare system, and a safer environment for both doctors and patients.

1.2 Problem Definition

Prior to the development and implementation of the Medica platform, healthcare professionals, particularly newly graduated doctors, faced substantial challenges in prescribing the correct medications. The existing systems were fragmented, relying heavily on manual searches for drug alternatives, prescription verification, and checking for potential drug interactions, all of which were based on static, often outdated resources. This inefficiency significantly increased the risk of prescription errors, adverse drug reactions, and compromised patient safety.

One of the most significant challenges in the prescribing process was the difficulty doctors faced in keeping up with new medications and treatments. Pharmaceutical companies lacked a centralized platform to efficiently update healthcare professionals on newly introduced drugs, which made it difficult for doctors to stay informed about the latest treatment options. Without timely access to comprehensive and accurate pharmaceutical data, doctors often relied on outdated information or limited resources, which led to gaps in treatment knowledge.

Another key issue was the management of patient records, which were often maintained manually. This made it difficult to effectively track patient medical histories, monitor treatment progress, and make timely adjustments to prescriptions. As a result, the process of prescribing medications was often complex, time-consuming, and prone to error. In addition, the absence of personalized and data-driven decision support tools left doctors with insufficient guidance on how to select the most appropriate medication for each patient, increasing the likelihood of treatment failure.

Medica addresses these critical challenges by providing an intelligent, centralized platform that integrates decision tree-based machine learning models for personalized medication recommendations. The system processes patient diagnosis, medical history, and other key health factors to suggest the most suitable treatment options. Real-time updates from pharmaceutical companies ensure that doctors always have access to the latest available drugs and therapies, bridging the knowledge gap.

Furthermore, Medica streamlines patient record management, allowing doctors to track treatment outcomes and update patient records seamlessly, ensuring accurate and up-to-date medical histories. The integration of role-based access control ensures that only authorized individuals can access sensitive patient data, safeguarding privacy and maintaining compliance with regulations such as HIPAA.

By automating and optimizing the medication prescribing process, Medica reduces prescription errors, improves treatment accuracy, and enhances overall healthcare efficiency, ultimately leading to better patient outcomes.

1.3 Project Objective (Suggested Solution)

The objective of this project is to develop an intelligent healthcare platform designed to assist doctors in making informed, data-driven medication prescriptions. By leveraging decision tree based machine learning models, the system will recommend the most suitable treatments based on patient diagnosis, medical history, and other critical health features. The platform is tailored to support healthcare professionals, especially new doctors, by providing personalized medication suggestions, assessing the likelihood of treatment success, and guiding decision-making to improve patient outcomes. The core functionalities of Medica are driven by its advanced machine learning capabilities, enabling personalized medication recommendations and prediction of medication results. In addition to these capabilities, Medica allows for seamless medication searches, checks for potential drug interactions, and ensures the secure management of up-to-date patient medical records. The platform also features a comprehensive and dynamic database that enables pharmaceutical companies to register, update, and manage medicine datasets, ensuring that doctors have access to the most current and relevant treatments. This functionality ensures continued relevance and supports the timely integration of new drugs and therapies into clinical practice. Data integrity and privacy are paramount; To ensure data security and privacy, Medica employs SSL encryption, role-based access control, stringent user input restrictions and more security practices as it strives to comply with HIPAA to safeguard patient data and maintain confidentiality. Developed as a web-based platform, Medica offers a user-friendly interface that enhances accessibility and usability for doctors in various healthcare settings. By integrating this solution, the system aims to bridge the knowledge gap for doctors, particularly those with less experience, reduce the impact of medication shortages, and optimize treatment decisions. The long-term vision is to enhance healthcare efficiency and safety, ensuring that doctors can make better-informed decisions, thereby improving patient care and treatment outcomes. Medica aligns with the broader goal of advancing technology's role in healthcare. By seamlessly integrating innovative technology into medical decision-making, the platform aims to enhance the precision, efficiency, and personalization of care. This integration will support the transition of healthcare systems toward more technology-driven practices, ensuring that medical decisions are not only informed by clinical experience but also empowered by data and advanced algorithms. The platform will continuously evolve through ongoing updates, ensuring that both clinical practices and pharmaceutical knowledge remain at the forefront of medical advancements. By automating and optimizing the medication prescribing process, this system will support the healthcare industry's transition toward more precise, personalized, and efficient care.

1.4 Gantt chart of project time plan

Graduation Project Phase 1:

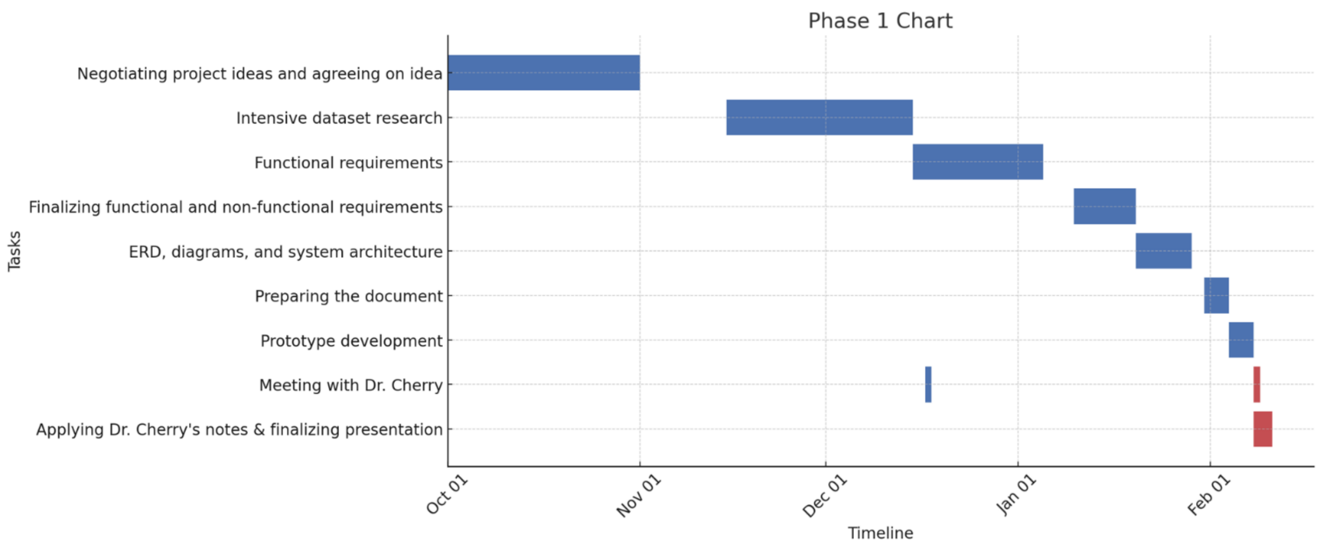


Figure 1: Gantt Chart for graduation project phase 1

Graduation Project phase 2:

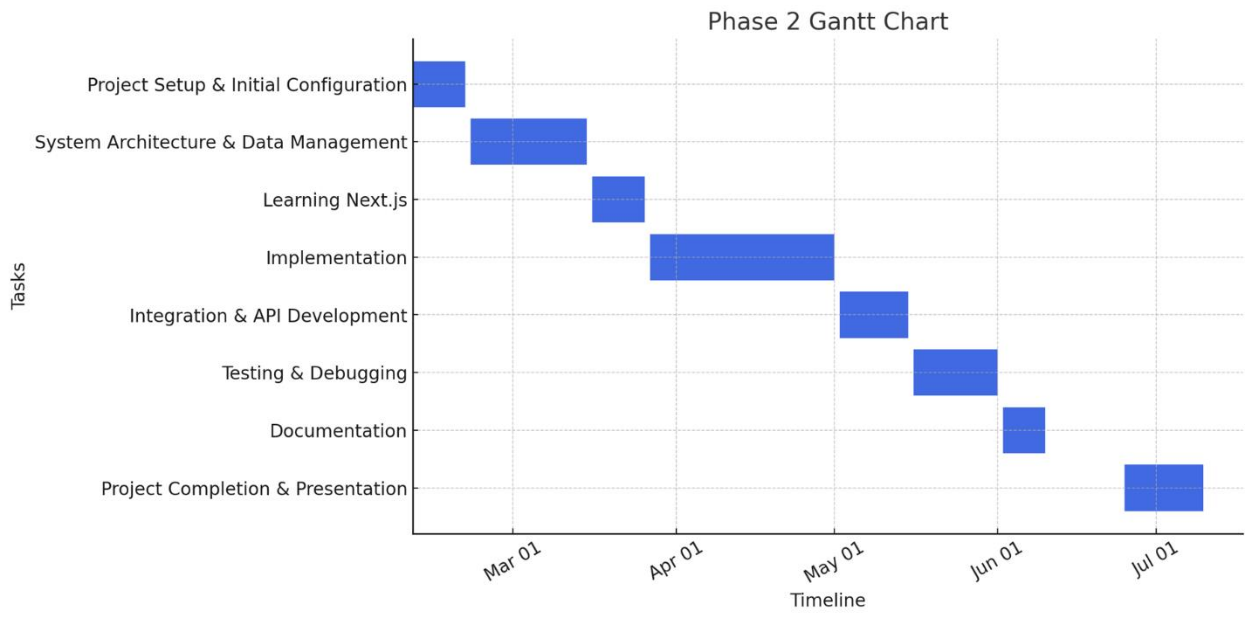


Figure 2: Gantt Chart for graduation project phase 2

1.5 Project development methodology

The development of the **Medica** system followed a hybrid project development methodology, combining aspects of both Waterfall and Agile approaches to accommodate the project's initial planning needs and iterative development nature.

1.5.1 Methodology Overview

The project commenced with a comprehensive planning phase following the **Waterfall model**. During this stage, the team conducted extensive requirement analysis, designed the system architecture, drafted specifications, and finalized the initial documentation. This phase ensured a clear understanding of system functionalities, technology stack, and academic stakeholders expectations.

Subsequently, the development transitioned to an **Agile-based** iterative process, in which implementation and testing were carried out in cycles. Although the documentation was completed early, it was continuously revised in response to new requirements, system limitations, or design optimizations identified during development. This flexibility is a key attribute of the hybrid approach.

1.5.2 Workflow and Phases

The project was structured into the following sequential and iterative phases:

1. **Requirement Analysis and Documentation (Waterfall Phase)**

Comprehensive functional and non-functional requirements were collected and documented. These formed the foundation of system.

Key activities included:

- **Functional Requirements Collection**
  - Identified and documented the core functionalities needed by each user type (doctors, admins, pharmaceutical companies).
  - Included features such as user authentication, patient management, medicine recommendation, drug interaction checks, and pharmaceutical data upload.
- **Non-Functional Requirements Definition**
  - Addressed security needs such as role-based access control and data protection.
  - Defined usability, reliability, and maintainability expectations.
- **Stakeholder Consultation**
  - Engaged with internal stakeholders (development team) and reviewed use cases relevant to doctors and medical professionals to align system goals with real-world needs.

2. **System Design and Architecture**

   A high-level architecture was designed, incorporating a modular structure to support distinct components:

   - **Next.js** was selected for building a responsive frontend interface.
   - **Spring Boot** was used to implement the core backend services related to authentication, administration, and data management and persistence.
   - **FastAPI** was utilized to serve machine learning functionalities due to its lightweight performance and seamless integration with Python-based ML models.
   - **PostgreSQL** served as the centralized relational database for structured data storage.

   System Communication

   - The **frontend (Next.js)** communicates with the **Spring Boot backend** and **FastAPI services** through RESTful APIs.
   - **Spring Boot** interact with **PostgreSQL** via service-layer abstraction, depending on the function.

   Architectural Benefits

   - **Separation of concerns**: Clearly divided components improve code maintainability and testing.
   - **Scalability**: Each service can be scaled independently based on load and demand.
   - **Security**: The architecture supports secure access control mechanisms and data isolation.
   - **Extensibility**: New modules or services (e.g., mobile clients, analytics dashboards) can be added with minimal disruption.

3. **Iterative Development and Testing (Agile Phase)**

   Development was executed in multiple iterations. Each iteration focused on implementing specific modules such as user authentication, medicine recommendation, drug interaction analysis, and pharmaceutical data management. At the end of each cycle:

   - Features were integrated and tested using unit and integration tests.
   - Team feedback was collected and used to refine subsequent iterations.
   - Updates were made to the system documentation to reflect changes or improvements.

4. **Evaluation and Enhancement**

Following each iteration, the system was evaluated in a staging environment. Feedback from supervisors and simulated user scenarios informed additional enhancements.

Based on the observations and feedback collected, the team made targeted enhancements. These included:

- **UI/UX improvements** to optimize user flow, interface responsiveness, and accessibility.
- **Refinements to machine learning models**, such as retraining with updated data, adjusting hyperparameters, or improving model integration for better recommendation accuracy.

5. **Toolchain and Support**

Development and collaboration were supported by the following tools:

- Git and GitHub for version control and codebase management.
- Google Docs and Draw.io for collaborative documentation and design diagrams.

This hybrid methodology provided the necessary structure during planning and flexibility during development, ensuring that **Medica** was delivered with both rigor and adaptability, aligning with academic standards and real-world software engineering practices.

1.6 The used tools in the project (SW and HW)

1.6.1 Machine Learning and Data Science Layer

Python (Programming Language)

Purpose: Development of personalized medication recommendation and predicting

medication suitability ML models.

Selection Rationale:

• Extensive ecosystem of machine learning libraries and frameworks

• Superior data manipulation capabilities through pandas and NumPy

• Seamless integration with Jupyter notebooks for model experimentation

Jupyter Notebook (Interactive Development Environment)

Purpose: Training and evaluation of ML models

Selection Rationale:

• Excellent visualization capabilities for model performance analysis

• Documentation capabilities that combine code, results, and explanations

Scikit-learn (ML Library)

Purpose: Medical Data Encoding and Model Training and Evaluation.

Selection Rationale:

• Comprehensive suite of ML algorithms suitable for healthcare predictions

• Robust preprocessing and feature engineering tools

• Production-ready implementations of classification and regression algorithms

NumPy (Numerical Computing Library)

Purpose: Confusion Matrix data preparation and Model Performance Analysis.

Selection Rationale:

• Efficient memory usage for large datasets.

• High-performance numerical computing capabilities

Pandas (Data Manipulation Library)

Purpose: Medical Dataset Processing and Feature Matrix Construction.

Selection Rationale:

• Efficient handling of structured healthcare data

• Powerful data cleaning and preprocessing capabilities

Matplotlib (Data Visualization Library)

Purpose: Confusion Matrix Visualization and Interactive Plot.

Selection Rationale:

• Comprehensive plotting capabilities for model evaluation

1.6.2 API Layer

FastAPI (Python Web Framework for APIs)

Purpose: Serving real-time ML prediction services endpoints.

Selection Rationale:

• Automatic API documentation generation (OpenAPI/Swagger)

• Easy integration with machine learning models

• Excellent performance for real-time prediction services

1.6.3 Backend Services Layer

Java Spring Boot (Enterprise Backend Framework)

Purpose: Implementing All the Non-Machine learning backend features.

Selection Rationale:

• handle large-scale enterprise applications with ease.

• Excellent performance and scalability characteristics

• Strong support for database transactions and data integrity

1.6.4 Database Layer

PostgreSQL (Relational Database Management System)

Purpose: Storing patient medical records, maintaining comprehensive medication databases, drug interaction databases, and managing user profiles.

Selection Rationale:

• ACID compliance ensuring data integrity for critical healthcare information

• Excellent performance for complex queries

• Strong backup and recovery capabilities essential for healthcare data

• Scalability to handle growing datasets from pharmaceutical companies

1.6.5 Frontend Layer

Next.js (React-based Web Framework)

Purpose: Implementation of multi-role user interfaces encompassing all dashboards, administrative portals, and machine learning features interfaces.

Selection Rationale:

• Server-side rendering for improved performance and SEO

• Built-in optimization features for fast loading times

• React-based component architecture for maintainable UI development

• API routes for seamless backend integration

1.6.6 Security Layer tools

SSL (Secure Sockets Layer Encryption protocol)

Purpose: Encrypting patient medical data transmission.

Selection Rationale:

• Protection of sensitive patient data during transmission

• Compliance with healthcare privacy regulations (HIPAA)

• Prevention of man-in-the-middle attacks

1.6.7 Development and Collaboration Tools

GitHub

Purpose: Version control for medication recommendation algorithms, collaborative development of drug interaction models, and maintaining pharmaceutical dataset integration code across development teams.

Selection Rationale:

• Distributed version control for multi-developer collaboration

• Code review processes ensuring code quality

• Backup and disaster recovery for source code

1.6.8 Used IDEs

PyCharm IDE

Purpose: Python development environment

Selection Rationale:

• Advanced debugging capabilities for API development and ML model troubleshooting

• Integrated testing tools for endpoint validation

IntelliJ IDEA

Purpose: Java development environment

Selection Rationale:

• Advanced Java development tools and code analysis

• Integrated debugging and profiling capabilities

• Seamless integration with Spring Boot framework

• Database tools for PostgreSQL development

Visual Studio Code

Purpose: Frontend development environment for Next.js application

Selection Rationale:

• Lightweight and fast performance for React/Next.js development

• Real-time debugging and hot reload capabilities for frontend development

1.7 Report Organization (summary of the rest of the report)

This report is structured into eight main chapters. Each chapter addresses a specific aspect of our project **Medica**, ranging from conceptual foundations to system implementation and machine learning integration. A brief overview of each chapter is provided below:

- **Chapter 1: Introduction**

  This chapter introduces the overall context and motivation for the **Medica** project. It defines the problem domain, outlines the project objectives, and presents the proposed solution. The chapter also includes the project timeline (Gantt chart), the development methodology followed, and a description of the software and hardware tools used.

- **Chapter 2: Related Work**

  This chapter reviews existing systems and research efforts relevant to **Medica**. It compares their functionalities to our proposed system, identifying similarities and key differences. References to prior work and publicly available platforms are used to support the comparative analysis.

- **Chapter 3: System Analysis**

  This chapter presents the detailed system requirements. It covers both functional and non-functional specifications, followed by use case diagrams that capture the interactions between users and the system components.

- **Chapter 4: System Design**

  This chapter outlines the technical design of the system. It includes the system architecture, component diagram, class diagrams, and sequence diagrams. The Entity Relationship Diagram (ERD) is provided to describe the database schema, along with a graphical design of the system's user interface.

- **Chapter 5: Implementation and Testing**

  This chapter describes the implementation of the system's frontend, backend, and machine learning components. It also provides examples of the system in operation and details selected test cases that verify functionality and ensure system correctness.

- **Chapter 6: Machine Learning Report**

  This chapter focuses on the machine learning models used in *Medica*. It explains the data sources, preprocessing steps, model selection criteria, training procedures, performance evaluation, and the integration of the models into the recommendation system.

- **References**
  This section includes all the sources used in the report, such as datasets, and similar systems. These references support the information provided in our document.

Chapter 2: Related work

The following table presents the related Work, highlighting existing healthcare systems, their core functionalities, and the key differences compared to the Medica platform:

Healthcare System	Description	Existing Similar Implementations	Main Differences with our Project	Official Website
Cerner EHR	A cloud-based EHR system that helps hospitals manage patient health records, treatment plans, and doctor-patient interactions.	Electronic Health Record (EHR) System	Lacks AI-powered medicine recommendations; does not allow pharmaceutical companies to directly interact within the platform.	https://www.cerner.com/
Drugs.com	An online drug database providing information about medicines, their uses, side effects, and drug interactions.	Online Drug Database	Only provides drug information; does not integrate patient records, doctor access, or AI-driven prescription recommendations.	https://www.drugs.com/
Ada Health	A symptom-checking and AI-powered diagnosis application that helps users and doctors determine potential diseases and treatments.	AI-Powered Diagnosis System	Focuses on symptom checking rather than full patient records or doctor prescriptions; no pharmaceutical company involvement.	https://www.ada.com/
MediLedger	A blockchain-based pharmaceutical management system that ensures drug authenticity and secure medicine supply chain tracking.	Pharmaceutical Management System	Blockchain-based pharmaceutical tracking but lacks direct doctor-patient interaction or AI-driven medicine recommendations.	https://www.mediledger.com/

Table 1: Related Work

Chapter 3: System Analysis

3.1 Project specification

3.1.1 Functional requirements:

1. Admin Features

User Authentication & Management

- **Admin Login**

  The system shall allow an admin to securely log in using their registered email and password.

- **Admin Logout**

  The system shall allow a signed-in admin to securely log out of their account at any time.

Admin Account Management

- **Create Admin Account**

  Admins shall be able to register new admin accounts by providing full name, email, password, and contact information.

- **View Admins**

  Admins shall be able to view a list of all registered admin accounts.

- **Edit Admin Information**

  Admins shall be able to update profile information for any admin account.

- **Delete Admin Account**

  Admins shall be able to remove an admin account from the system when required.

Doctor Management

- **Create Doctor Account**

  Admins shall be able to register new doctors by entering name, specialization, contact information, and login credentials.

- **View Doctors**

  Admins shall be able to view a list of all registered doctors and their details.

- **Edit Doctor Information**

  Admins shall be able to update personal and professional details of any doctor.

- **Delete Doctor Account**

  Admins shall be able to remove a doctor’s account from the system when required.

Pharmaceutical Company Management

- **Create Company Account**

  Admins shall be able to register new pharmaceutical companies by providing name, contact information, location, and login credentials.

- **View Companies**

  Admins shall be able to view a list of all registered pharmaceutical companies and their details.

- **Edit Company Information**

  Admins shall be able to update profile information for any company.

- **Delete Company Account**

  Admins shall be able to remove a company account from the system when required.

2. Pharmaceutical Company Features

- **Company Login**

  Companies shall be able to securely log in using their registered email and password.

- **Add Medicine Dataset**

  Companies shall be able to upload bulk medicine datasets, including names, indications, and side effects.

- **Add Medicine**

  Companies shall be able to add individual medicines, specifying name, uses, and potential side effects.

- **Remove Medicine**

  Companies shall be able to delete previously added medicines when necessary.

- **Update Medicine Information**

  Companies shall be able to modify details of existing medicines, including indications and side effects.

3. Doctor Features

- **Doctor Login**
  Doctors shall be able to securely log in using their registered email and password.

- **Search for Medicines**
  Doctors shall be able to search for medicines and view uses, substitutes, and side effects.

- **Add Patient**
  Doctors shall be able to register new patients to their assigned list.

- **Add Patient Visit**
  Doctors shall be able to record patient visits, including date, diagnosis, prescribed medication, and relevant details.

- **View Patient Information**
  Doctors shall be able to retrieve a patient's profile and medical history.

- **View Patient Visits**
  Doctors shall be able to view a list of visits for a specific patient.

- **View Visit Details**
  Doctors shall be able to access detailed information for any specific patient visit.

- **View Assigned Patients**
  Doctors shall be able to view only those patients assigned to them.

- **Customized Medicine Recommendation (ML-Based)**
  Doctors shall enter patient data (diagnosis, history) and receive medicine recommendations from the ML model.

- **Check Drug Interactions**
  Doctors shall input different medications and inquire about potential drug–drug interactions.

- **Assess Medicine Suitability (ML-Based)**
  Doctors shall input a medicine and patient condition to receive an ML-driven suitability assessment.

- **View Registered Companies & Medicines**
  Doctors shall be able to view all registered pharmaceutical companies and their available products.

- **Add Company to Favourites**
  Doctors shall be able to mark companies as favourites.

3.1.2 Non-functional requirements:

Performance

- Response Time
    - Standard operations shall complete within two seconds.
    - Complex queries (e.g., ML inferences, interaction checks) shall complete within five seconds.

Availability

- The system shall maintain 99.9 % uptime.
- Scheduled maintenance shall occur during off-peak hours.

Reliability

- Data consistency shall be maintained across all services, especially for patient records and prescriptions.

Security

- Role-based access control shall enforce feature-level permissions for administrators, doctors, and companies.
- All actions (e.g., record updates) shall be logged for auditing purposes.

Usability

- The user interface shall be intuitive and minimize the learning curve for all user roles.

Maintainability

- The architecture shall be modular, allowing independent updates or replacement of components.
- All code shall include clear documentation and inline comments to facilitate future development.

Security Measures

The Medica platform incorporates a robust suite of security measures designed to safeguard sensitive patient and user data, ensuring confidentiality, integrity, availability, authentication, and non-repudiation throughout the system. The security framework is built to prevent unauthorized access, mitigate common vulnerabilities, and ensure compliance with privacy regulations. By adopting industry-standard practices, hashing protocols, and strict user access controls, the platform guarantees secure handling of patient data while providing healthcare professionals with the tools necessary to make informed, data-driven decisions. The platform's architecture has been designed with a focus on achieving comprehensive security goals, including secure authentication, role-based authorization, data integrity, non-repudiation, and accountability, all of which work together to maintain a secure and efficient healthcare management environment.

Below are the key security implementations within the platform:

1. Password Security

- **Strong Password Policies**: A stringent password policy is enforced across all user roles, including admins, doctors, and pharmaceutical companies. The policy requires:
    - At least 8 characters in length
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one numeric digit
    - At least one special character
- **Password Hashing**: All passwords are hashed before being stored in the database. This ensures that even in the event of a breach, passwords cannot be easily compromised.

2. System Design and Documentation

- **Comprehensive Design Documentation**: The entire system's architecture, including security measures, has been thoroughly documented. This includes detailed specifications for encryption, session management, data handling, and access controls.
- **User and Role-Based Privileges**: User roles (admin, doctor, pharmaceutical company) are clearly defined, with access to system resources restricted based on the user's role. Privileges for each role have been documented to ensure that only authorized users can access or modify sensitive data.

3. Access Control

- **Authorization Based on Roles**: Authorization is implemented using role-based access control (RBAC). Each user role has a predefined set of permissions, ensuring users only access what is necessary for their function.
- **Session Management**: The system employs secure session management practices, ensuring that user sessions are securely maintained. Pages are restricted to authenticated and authorized users only. Users cannot directly access these pages without logging in.

4. Protection Against Common Vulnerabilities

- **Prevention of SQL Injection**: To mitigate the risk of SQL injection attacks, all database queries are parameterized, and proper validation/sanitization is performed on user inputs.
- **Cryptography**: Sensitive data, such as passwords and other critical information, is protected through encryption techniques, primarily focusing on the use of cryptographic hashes for passwords.
- **Secure Failure Mechanisms**: In the event of exceptions or system failures, the system is designed to fail securely, without revealing sensitive system information through error messages. Detailed stack traces and internal details are not exposed to end users, ensuring sensitive data is not inadvertently disclosed.

5. Logging and Auditing

- **User Activity Logging**: All critical actions performed within the system are logged for auditing purposes. These logs include:
    - ❖ Creation of pharmaceutical companies, doctor accounts, and admin accounts.
    - ❖ Deletion of pharmaceutical companies, doctor accounts, and admin accounts.
    - ❖ Modifications made to doctor information.
    - ❖ User login activities.
- These logs are securely stored and are available for administrative review. They are used to track any suspicious or unauthorized activity, ensuring accountability and maintaining the security and integrity of the system.

6. File Upload Security

- **File Upload Restrictions**: When pharmaceutical companies upload files, the system enforces strict file type restrictions to prevent the upload of potentially malicious files. Only approved file formats are accepted, and files are scanned for any signs of malicious code before they are processed.

7. SSL (Secure Sockets Layer) Protocol

SSL encryption was implemented for communication between the Spring Boot backend application and the PostgreSQL database.

SSL is a cryptographic protocol designed to provide secure communication over a computer network, specifically by encrypting the data transmitted between the client and server.

Applying the SSL protocol involved the following:

1- **Generating SSL Certificates:** SSL certificates were created to encrypt communication between the Spring Boot backend and the PostgreSQL database.
2- **Configuring PostgreSQL for SSL:** The PostgreSQL server was set up to accept SSL connections by modifying the postgresql.conf (setting ssl = on) and pg_hba.conf files (allowing SSL connections with hostssl).
3-**Configuring Spring Boot:** SSL parameters were added to the application.properties file in Spring Boot to specify the SSL settings and certificate locations.
4-**Testing SSL Connection:** After configuration, the connection was tested to ensure that data transmission between the backend and the database is securely encrypted.

8. Failure to Authenticate / Unauthorized Access

- **Authorization Based on Defined Roles**: The platform uses clearly defined user roles (admin, doctor, pharmaceutical company) to control access to the system's features. Only users with the appropriate roles are granted access to sensitive patient data or the ability to perform administrative tasks.

- **Deny Access by Default**: The system implements the principle of least privilege, denying access to sensitive pages by default. Explicit permissions are required to access private pages, ensuring that only authorized users can view or modify sensitive data.

9. Data Integrity and Privacy

- **Patient Data Protection**: Given the sensitive nature of patient records, SSL encryption is applied to ensure secure transmission of data between the frontend and backend. The platform follows best practices in securing sensitive healthcare data, ensuring that it remains confidential and intact.

10. Hashing User IDs in URLs

- **Hashing User IDs**: To enhance security and protect user privacy, user IDs are hashed in the URLs of the Medica platform.

    - ❖ This practice prevents the exposure of sensitive user information directly in the URL.
    - ❖ Even if URLs are intercepted or exposed, hashed user IDs ensure that actual user information remains protected and unidentifiable without proper authorization.

3.2 Use case Diagram:

3.2.1 Doctor Use case Diagram:

The following use case diagram illustrates the various functionalities available to the doctors within the Medica system. It highlights key operations that the doctors can perform, including patient management, medicine research,checking drug interactions and recommending treatment for patients.

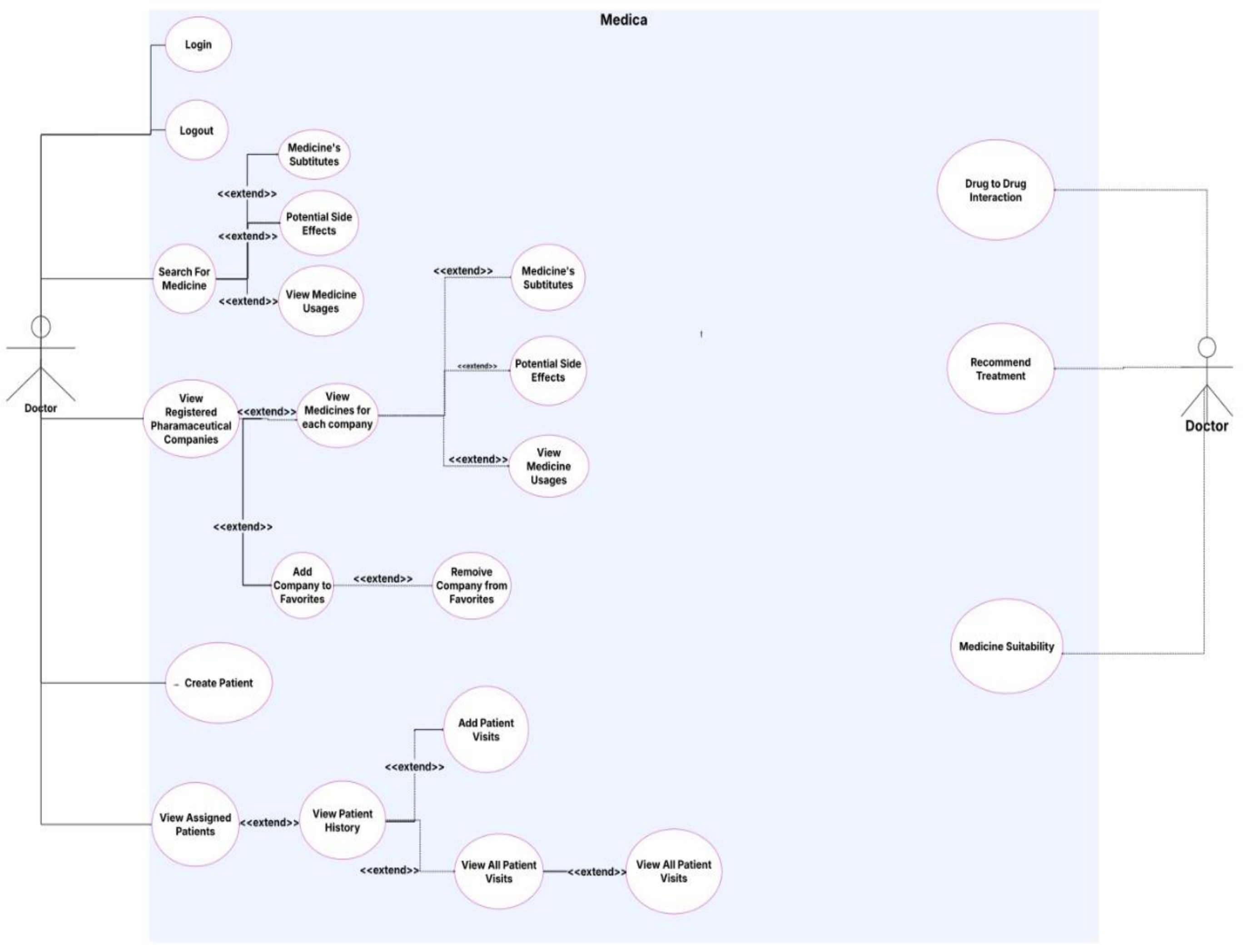


Figure 3 – Doctor Use case diagram

3.2.2 Admin and Pharma Company Use case Diagram:

The following use case diagram illustrates the various functionalities available to different user roles within the system. It primarily focuses on the actions that an Admin and Pharmaceutical Company can execute.

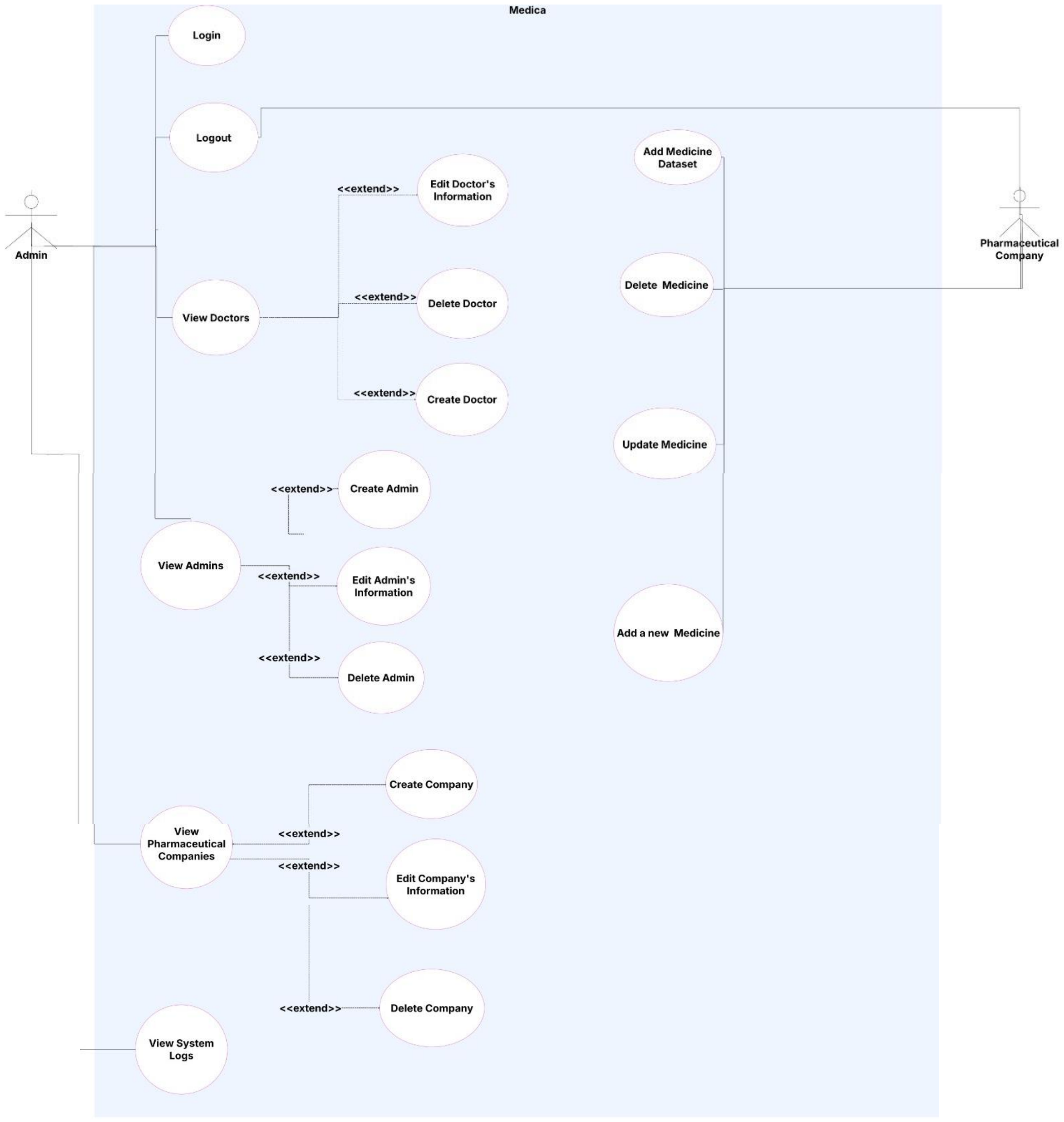


Figure 4 – Admin and Pharma Company Use case Diagram

Chapter 4: System Design

4.1 System Architecture

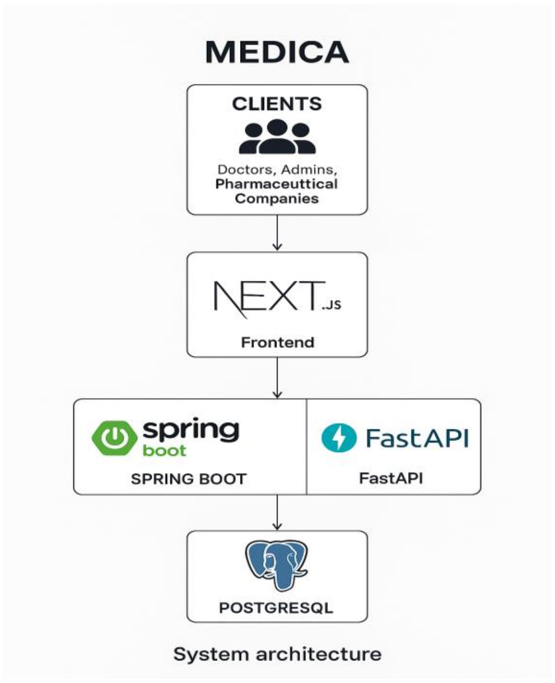


Figure 5 – System Architecture Diagram

The Medica system follows a modular, service-oriented architecture composed of four main layers: client, frontend, backend, and data storage. Users—including doctors, administrators, and pharmaceutical companies—interact with the system through a web interface built using Next.js.

The frontend communicates with two backend services using REST APIs:

- Spring Boot, which handles the core application logic, user management, and data persistence. It is the sole service interacting directly with the PostgreSQL database for storing and retrieving structured data such as user profiles, patient records, and pharmaceutical information.
- FastAPI, which hosts machine learning features which are medicine recommendation, and suitability assessment. It operates independently and does not access the database or communicate with Spring Boot.

Our architecture promotes separation of concerns, scalability, and maintainability by isolating the machine learning logic from core business operations and database management.

The machine learning functionality within the Medica system is encapsulated within the FastAPI service, which is responsible for delivering predictions for medicine recommendation and suitability assessment. These models are trained using synthetically generated, medically relevant datasets and are served as independent, stateless endpoints. This separation ensures flexibility in model updates and allows the system to scale machine learning capabilities independently of the core application logic. A detailed explanation of the machine learning models, data preprocessing, features selection, training procedures, and evaluation metrics is provided in **Chapter 6**.

Datasets used for drug to drug interaction feature and viewing medicine details

 250k Medicines Usage, Side Effects and Substitutes [1]:

This dataset contains comprehensive information on over 248,000 medical drugs from all manufacturers worldwide. It plays a crucial role in enhancing Medica's intelligent recommendation system by providing doctors with accurate and up-to-date information on medications. The dataset contains the following information for each drug:

1. Drug name
2. Adverse reactions and side effects
3. Drug interactions
4. Drug class
5. Substitute drugs

 Drug-Drug Interactions[2]:

This dataset, sourced from the DrugBank database, provides a comprehensive collection of drug-drug interactions. It plays a vital role in enhancing Medica's accuracy by enabling doctors to predict and understand complex drug interactions. By integrating this dataset, Medica improves prescription safety, minimizes adverse drug reactions, and supports data driven decision-making in pharmacology.

The dataset includes:

1. Drug 1
2. Drug 2
3. Interaction Description

4.2 System Component Diagram

The following figure illustrates the structure and interaction of the key components within the Medica platform, including the **Next.js** frontend, **PostgreSQL** database, and the backend layers of **model**, **service**, and **controller**. It also highlights the role of **Machine Learning (ML)** models in supporting decision-making.

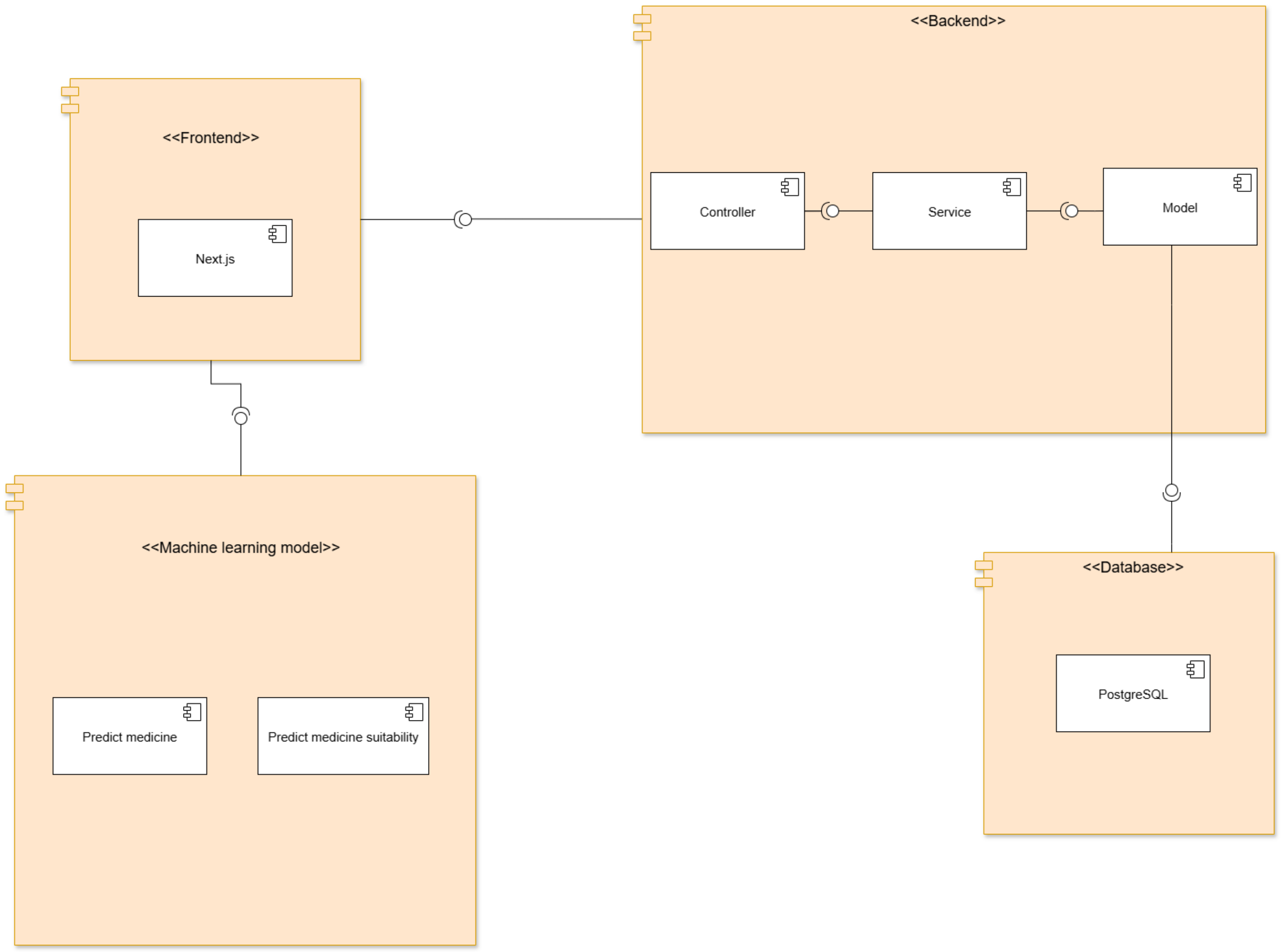
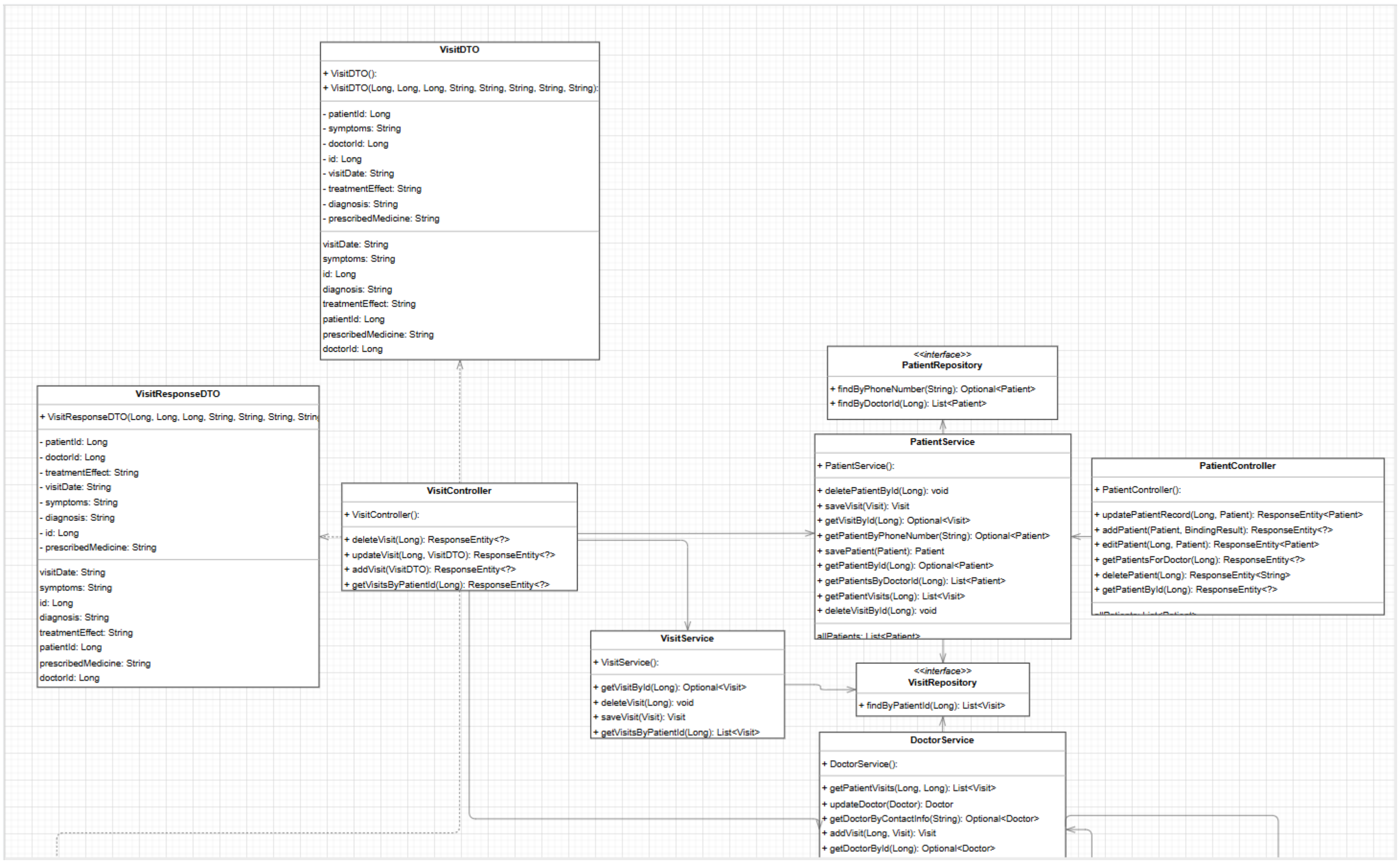
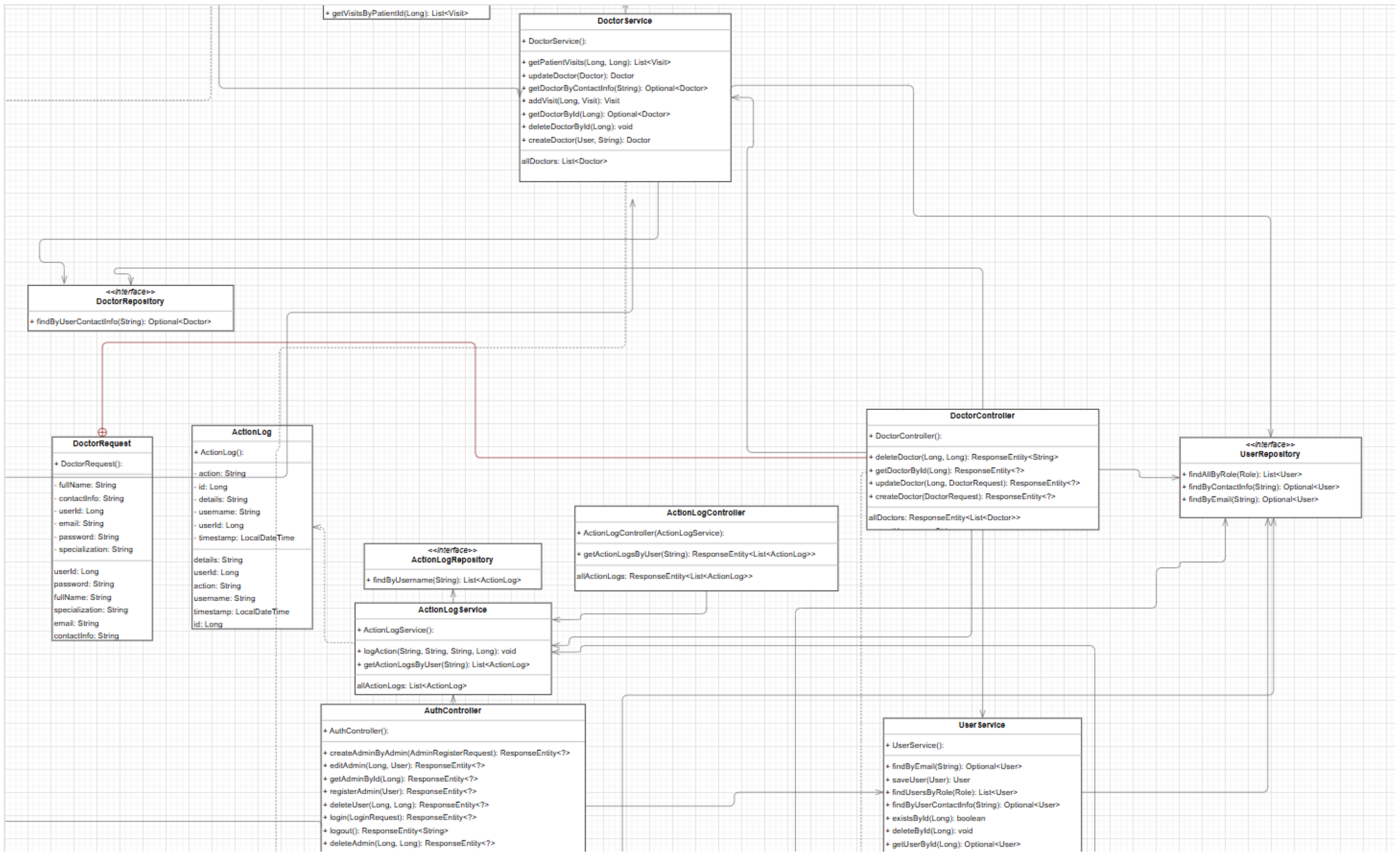


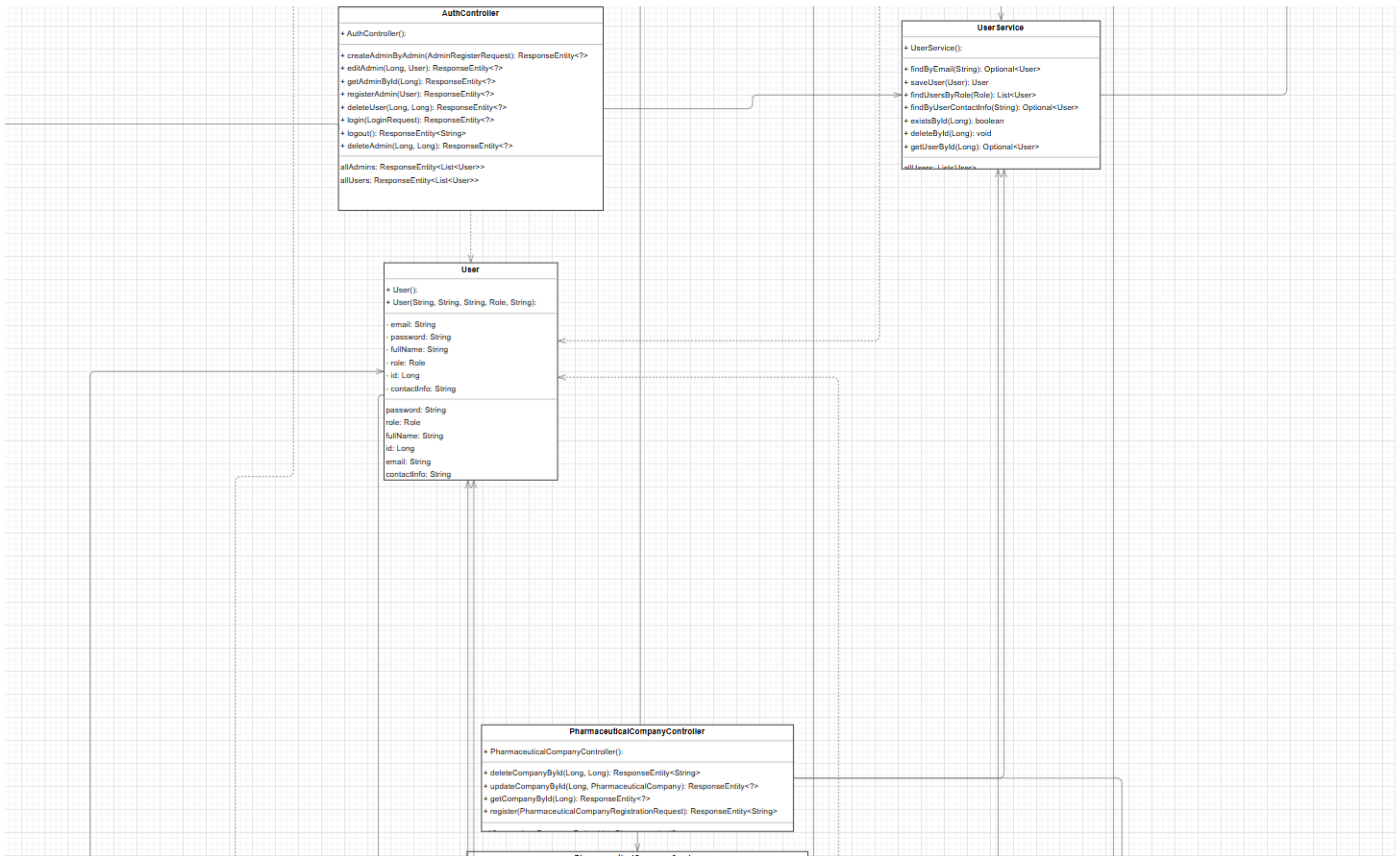
Figure 6 – Component Diagram

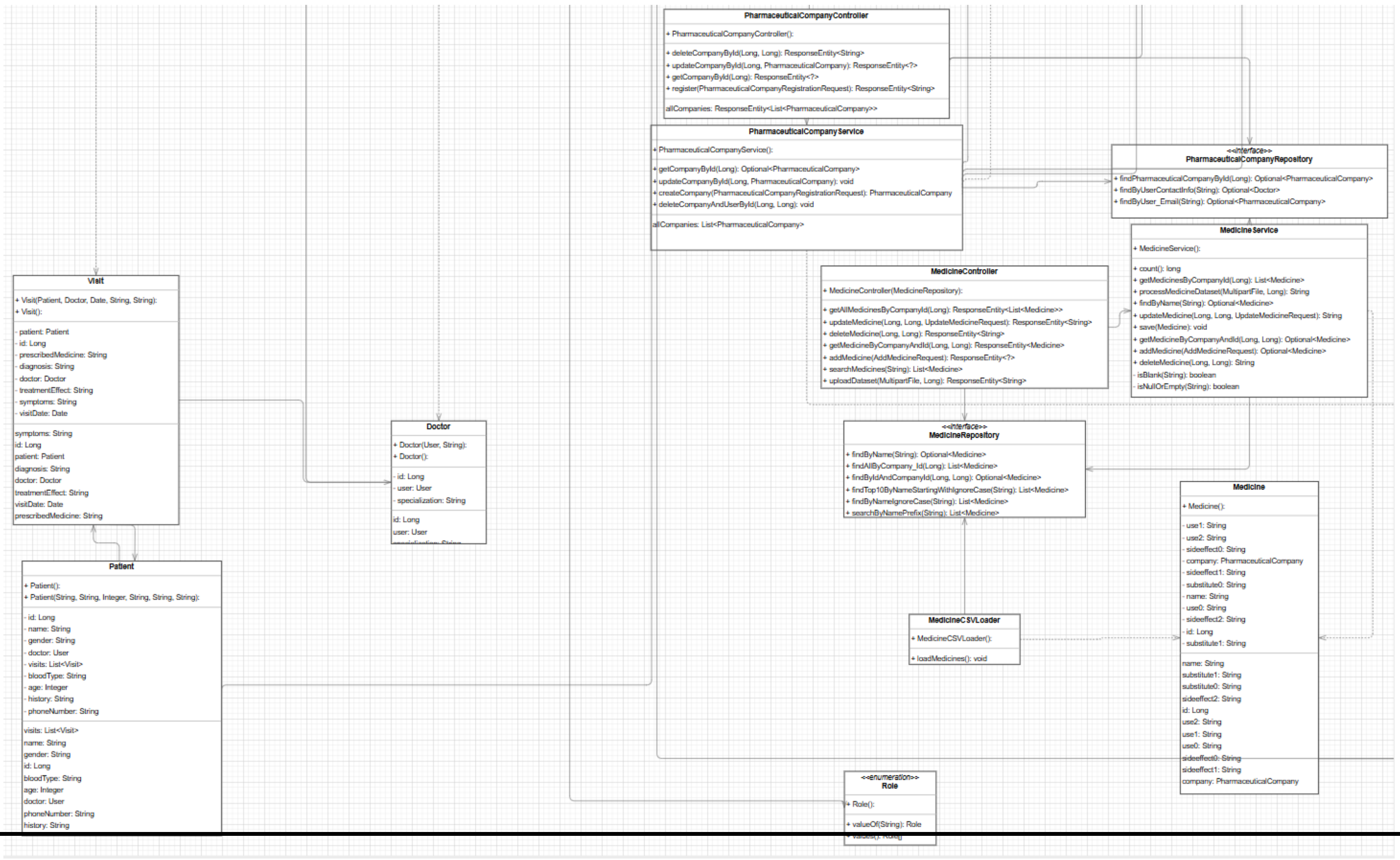
4.3 Class Diagram

The following figure illustrates the class diagram of the Medica platform, showcasing the relationships between classes such as **User**, **Admin**, **Doctor**, **PharmaceuticalCompany**, and **Patient**.


VisitDTO
+ VisitDTO():
+ VisitDTO(Long, Long, Long, String, String, String, String, String):
- patientId: Long
- symptoms: String
- doctorId: Long
- id: Long
- visitDate: String
- treatmentEffect: String
- diagnosis: String
- prescribedMedicine: String
visitDate: String
symptoms: String
id: Long
diagnosis: String
treatmentEffect: String
patientId: Long
prescribedMedicine: String
doctorId: Long
VisitResponseDTO
+ VisitResponseDTO(Long, Long, Long, String, String, String, Strin
- patientId: Long
- doctorId: Long
- treatmentEffect: String
- visitDate: String
- symptoms: String
- diagnosis: String
- id: Long
- prescribedMedicine: String
visitDate: String
symptoms: String
id: Long
diagnosis: String
treatmentEffect: String
patientId: Long
prescribedMedicine: String
doctorId: Long
VisitController
+ VisitController():
+ deleteVisit(Long): ResponseEntity<?>
+ updateVisit(Long, VisitDTO): ResponseEntity<?>
+ addVisit(VisitDTO): ResponseEntity<?>
+ getVisitsByPatientId(Long): ResponseEntity<?>
VisitService
+ VisitService():
+ getVisitById(Long): Optional<Visit>
+ deleteVisit(Long): void
+ saveVisit(Visit): Visit
+ getVisitsByPatientId(Long): List<Visit>
<<interface>>
PatientRepository
+ findByPhoneNumber(String): Optional<Patient>
+ findByDoctorId(Long): List<Patient>
PatientService
+ PatientService():
+ deletePatientById(Long): void
+ saveVisit(Visit): Visit
+ getVisitById(Long): Optional<Visit>
+ getPatientByPhoneNumber(String): Optional<Patient>
+ savePatient(Patient): Patient
+ getPatientById(Long): Optional<Patient>
+ getPatientsByDoctorId(Long): List<Patient>
+ getPatientVisits(Long): List<Visit>
+ deleteVisitById(Long): void
allPatients: List<Patient>
PatientController
+ PatientController():
+ updatePatientRecord(Long, Patient): ResponseEntity<Patient>
+ addPatient(Patient, BindingResult): ResponseEntity<?>
+ editPatient(Long, Patient): ResponseEntity<Patient>
+ getPatientsForDoctor(Long): ResponseEntity<?>
+ deletePatient(Long): ResponseEntity<String>
+ getPatientById(Long): ResponseEntity<?>
<<interface>>
VisitRepository
+ findByPatientId(Long): List<Visit>
DoctorService
+ DoctorService():
+ getPatientVisits(Long, Long): List<Visit>
+ updateDoctor(Doctor): Doctor
+ getDoctorByContactInfo(String): Optional<Doctor>
+ addVisit(Long, Visit): Visit
+ getDoctorById(Long): Optional<Doctor>


+ getVisitsByPatientId(Long): List<Visit>
DoctorService
+ DoctorService():
+ getPatientVisits(Long, Long): List<Visit>
+ updateDoctor(Doctor): Doctor
+ getDoctorByContactInfo(String): Optional<Doctor>
+ addVisit(Long, Visit): Visit
+ getDoctorById(Long): Optional<Doctor>
+ deleteDoctorById(Long): void
+ createDoctor(User, String): Doctor
allDoctors: List<Doctor>
<<interface>>
DoctorRepository
+ findByUserContactInfo(String): Optional<Doctor>
DoctorRequest
+ DoctorRequest():
- fullName: String
- contactInfo: String
- userId: Long
- email: String
- password: String
- specialization: String
userId: Long
password: String
fullName: String
specialization: String
email: String
contactInfo: String
ActionLog
+ ActionLog():
- action: String
- id: Long
- details: String
- username: String
- userId: Long
- timestamp: LocalDateTime
details: String
userId: Long
action: String
username: String
timestamp: LocalDateTime
id: Long
<<interface>>
ActionLogRepository
+ findByUsername(String): List<ActionLog>
ActionLogService
+ ActionLogService():
+ logAction(String, String, String, Long): void
+ getActionLogsByUser(String): List<ActionLog>
allActionLogs: List<ActionLog>
ActionLogController
+ ActionLogController(ActionLogService):
+ getActionLogsByUser(String): ResponseEntity<List<ActionLog>>
allActionLogs: ResponseEntity<List<ActionLog>>
DoctorController
+ DoctorController():
+ deleteDoctor(Long, Long): ResponseEntity<String>
+ getDoctorById(Long): ResponseEntity<?>
+ updateDoctor(Long, DoctorRequest): ResponseEntity<?>
+ createDoctor(DoctorRequest): ResponseEntity<?>
allDoctors: ResponseEntity<List<Doctor>>
<<interface>>
UserRepository
+ findAllByRole(Role): List<User>
+ findByContactInfo(String): Optional<User>
+ findByEmail(String): Optional<User>
AuthController
+ AuthController():
+ createAdminByAdmin(AdminRegisterRequest): ResponseEntity<?>
+ editAdmin(Long, User): ResponseEntity<?>
+ getAdminById(Long): ResponseEntity<?>
+ registerAdmin(User): ResponseEntity<?>
+ deleteUser(Long, Long): ResponseEntity<?>
+ login(LoginRequest): ResponseEntity<?>
+ logout(): ResponseEntity<String>
+ deleteAdmin(Long, Long): ResponseEntity<?>
UserService
+ UserService():
+ findByEmail(String): Optional<User>
+ saveUser(User): User
+ findUsersByRole(Role): List<User>
+ findByUserContactInfo(String): Optional<User>
+ existsById(Long): boolean
+ deleteById(Long): void
+ getUserById(Long): Optional<User>


AuthController
+ AuthController():
+ createAdminByAdmin(AdminRegisterRequest): ResponseEntity<?>
+ editAdmin(Long, User): ResponseEntity<?>
+ getAdminById(Long): ResponseEntity<?>
+ registerAdmin(User): ResponseEntity<?>
+ deleteUser(Long, Long): ResponseEntity<?>
+ login(LoginRequest): ResponseEntity<?>
+ logout(): ResponseEntity<String>
+ deleteAdmin(Long, Long): ResponseEntity<?>
allAdmins: ResponseEntity<List<User>>
allUsers: ResponseEntity<List<User>>
User Service
+ UserService():
+ findByEmail(String): Optional<User>
+ saveUser(User): User
+ findUsersByRole(Role): List<User>
+ findByUserContactInfo(String): Optional<User>
+ existsById(Long): boolean
+ deleteById(Long): void
+ getUserById(Long): Optional<User>
User
+ User():
+ User(String, String, String, Role, String):
- email: String
- password: String
- fullName: String
- role: Role
- id: Long
- contactInfo: String
password: String
role: Role
fullName: String
id: Long
email: String
contactInfo: String
PharmaceuticalCompanyController
+ PharmaceuticalCompanyController():
+ deleteCompanyById(Long, Long): ResponseEntity<String>
+ updateCompanyById(Long, PharmaceuticalCompany): ResponseEntity<?>
+ getCompanyById(Long): ResponseEntity<?>
+ register(PharmaceuticalCompanyRegistrationRequest): ResponseEntity<String>


PharmaceuticalCompanyController
+ PharmaceuticalCompanyController():
+ deleteCompanyById(Long, Long): ResponseEntity<String>
+ updateCompanyById(Long, PharmaceuticalCompany): ResponseEntity<?>
+ getCompanyById(Long): ResponseEntity<?>
+ register(PharmaceuticalCompanyRegistrationRequest): ResponseEntity<String>
allCompanies: ResponseEntity<List<PharmaceuticalCompany>>
PharmaceuticalCompany Service
+ PharmaceuticalCompanyService():
+ getCompanyById(Long): Optional<PharmaceuticalCompany>
+ updateCompanyById(Long, PharmaceuticalCompany): void
+ createCompany(PharmaceuticalCompanyRegistrationRequest): PharmaceuticalCompany
+ deleteCompanyAndUserById(Long, Long): void
allCompanies: List<PharmaceuticalCompany>
<<interface>>
PharmaceuticalCompanyRepository
+ findPharmaceuticalCompanyById(Long): Optional<PharmaceuticalCompany>
+ findByUserContactInfo(String): Optional<Doctor>
+ findByUser_Email(String): Optional<PharmaceuticalCompany>
Medicine Service
+ MedicineService():
+ count(): long
+ getMedicinesByCompanyId(Long): List<Medicine>
+ processMedicineDataset(MultipartFile, Long): String
+ findByName(String): Optional<Medicine>
+ updateMedicine(Long, Long, UpdateMedicineRequest): String
+ save(Medicine): void
+ getMedicineByCompanyAndId(Long, Long): Optional<Medicine>
+ addMedicine(AddMedicineRequest): Optional<Medicine>
+ deleteMedicine(Long, Long): String
- isBlank(String): boolean
- isNullOrEmpty(String): boolean
MedicineController
+ MedicineController(MedicineRepository):
+ getAllMedicinesByCompanyId(Long): ResponseEntity<List<Medicine>>
+ updateMedicine(Long, Long, UpdateMedicineRequest): ResponseEntity<String>
+ deleteMedicine(Long, Long): ResponseEntity<String>
+ getMedicineByCompanyAndId(Long, Long): ResponseEntity<Medicine>
+ addMedicine(AddMedicineRequest): ResponseEntity<?>
+ searchMedicines(String): List<Medicine>
+ uploadDataset(MultipartFile, Long): ResponseEntity<String>
Visit
+ Visit(Patient, Doctor, Date, String, String):
+ Visit():
- patient: Patient
- id: Long
- prescribedMedicine: String
- diagnosis: String
- doctor: Doctor
- treatmentEffect: String
- symptoms: String
- visitDate: Date
symptoms: String
id: Long
patient: Patient
diagnosis: String
doctor: Doctor
treatmentEffect: String
visitDate: Date
prescribedMedicine: String
Doctor
+ Doctor(User, String):
+ Doctor():
- id: Long
- user: User
- specialization: String
id: Long
user: User
<<interface>>
MedicineRepository
+ findByName(String): Optional<Medicine>
+ findAllByCompany_Id(Long): List<Medicine>
+ findByIdAndCompanyId(Long, Long): Optional<Medicine>
+ findTop10ByNameStartingWithIgnoreCase(String): List<Medicine>
+ findByNameIgnoreCase(String): List<Medicine>
+ searchByNamePrefix(String): List<Medicine>
Medicine
+ Medicine():
- use1: String
- use2: String
- sideeffect0: String
- company: PharmaceuticalCompany
- sideeffect1: String
- substitute0: String
- name: String
- use0: String
- sideeffect2: String
- id: Long
- substitute1: String
name: String
substitute1: String
substitute0: String
sideeffect2: String
id: Long
use2: String
use1: String
use0: String
sideeffect0: String
sideeffect1: String
company: PharmaceuticalCompany
MedicineC SVLoader
+ MedicineCSVLoader():
+ loadMedicines(): void
Patient
+ Patient():
+ Patient(String, String, Integer, String, String, String):
- id: Long
- name: String
- gender: String
- doctor: User
- visits: List<Visit>
- bloodType: String
- age: Integer
- history: String
- phoneNumber: String
visits: List<Visit>
name: String
gender: String
id: Long
bloodType: String
age: Integer
doctor: User
phoneNumber: String
history: String
<<enumeration>>
Role
+ Role():
+ valueOf(String): Role

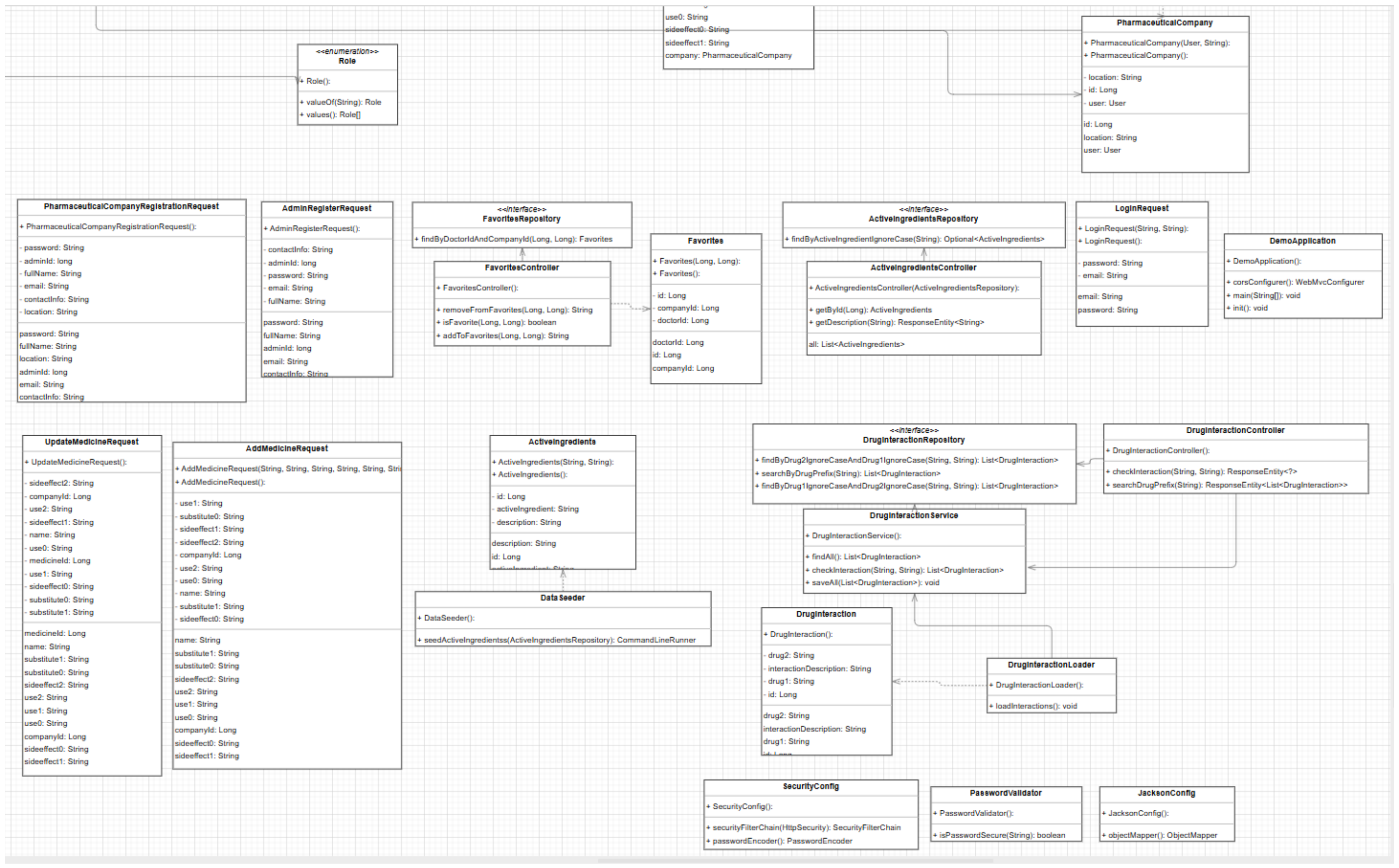


Figure7 – Class Diagram

4.4 Sequence Diagram

1- Admin Login, Create Doctor Account, View Doctors, and Delete Doctor Scenarios:

The admin logs in and creates a doctor account by entering details like name, specialization and contact. The system validates data and confirms creation.
The admin can view a list of registered doctors retrieved from the database.
The admin can delete a doctor's account, and the system removes the record, confirming deletion.

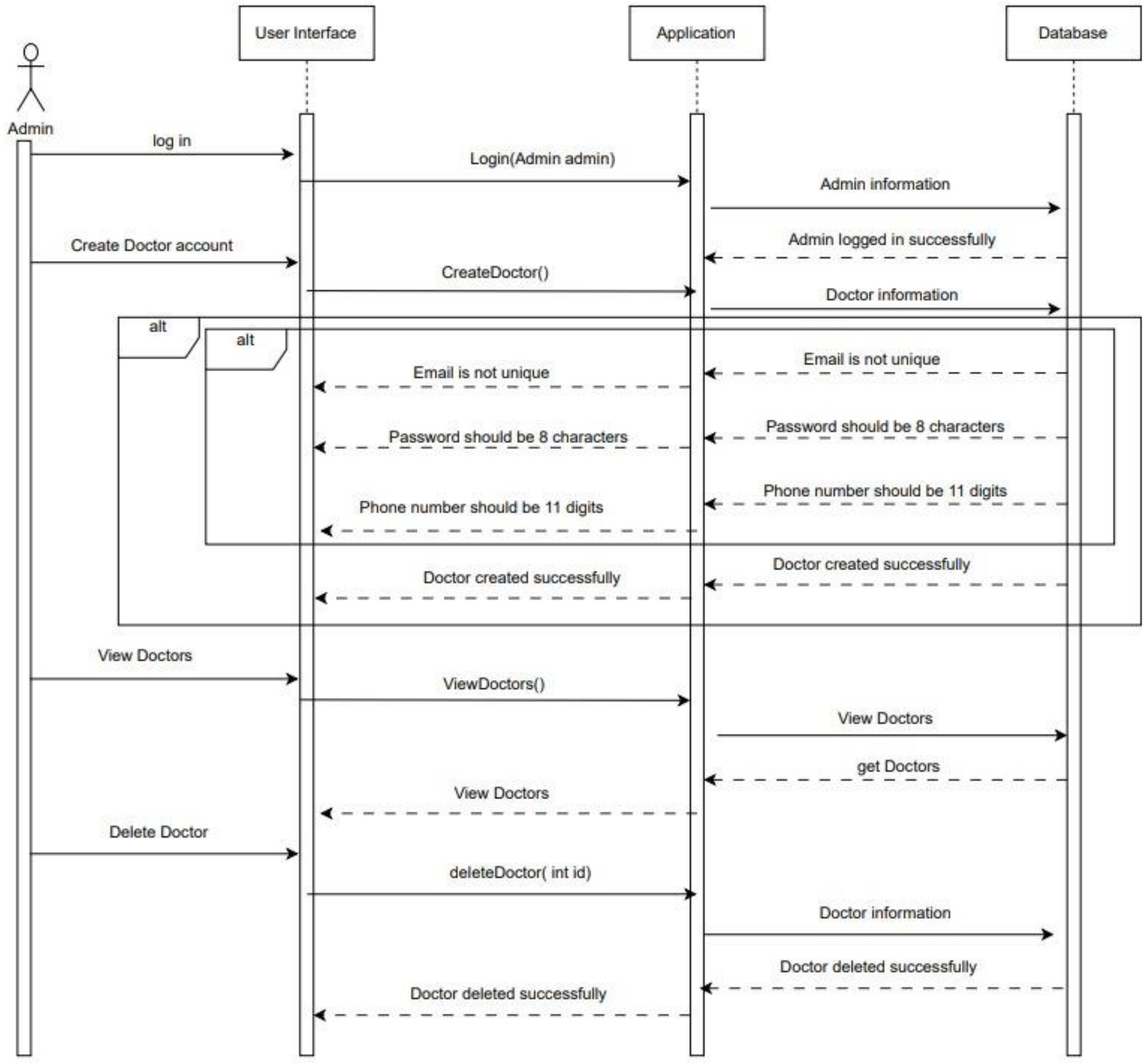


Figure 8 – Sequence Diagram 1

2- Doctor Login, Search Medicines, and Check Drug Interactions Scenarios:

The doctor logs into the system using their registered email and password. Upon successful authentication, access is granted.
The doctor searches for a specific medicine. The system retrieves and displays relevant details, including uses, substitutes, and potential side effects.
The doctor can input multiple medications to check for drug interactions. The system analyzes the combination and provides interaction details.

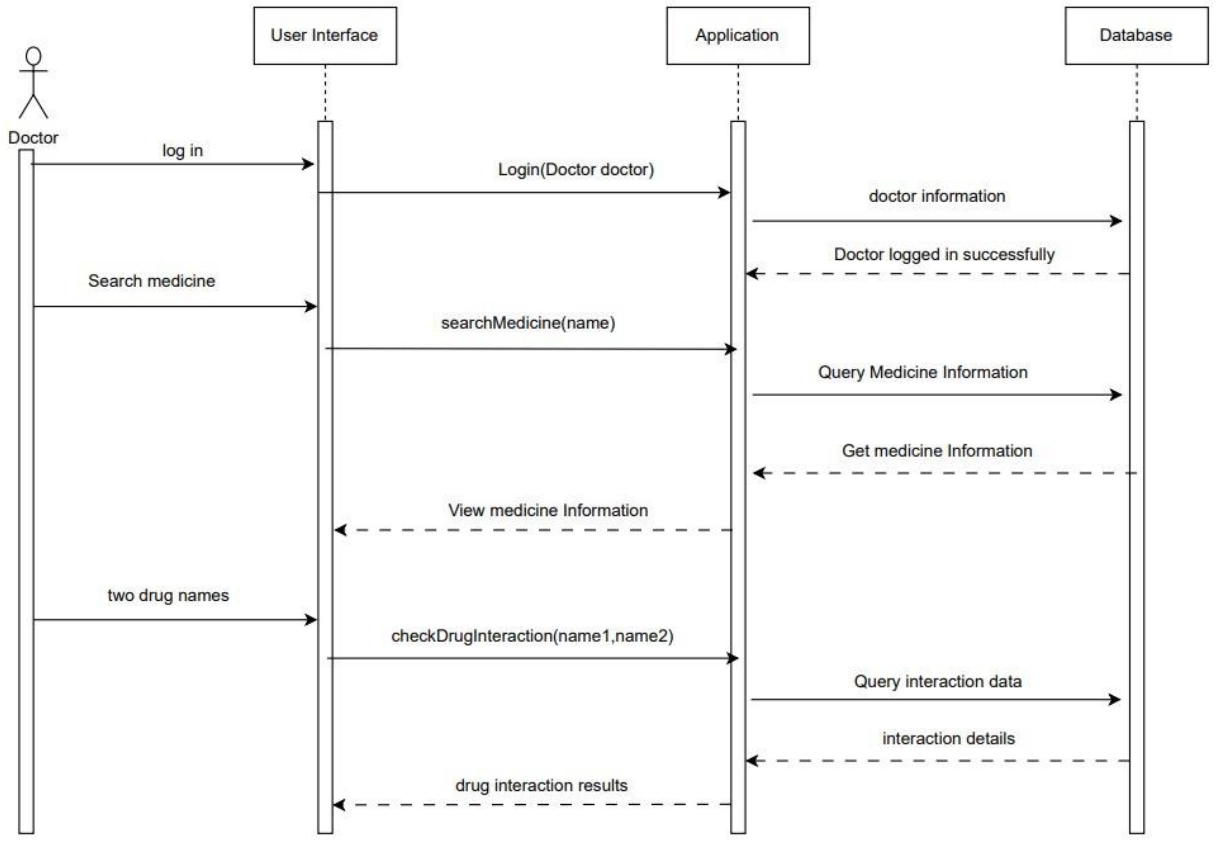


Figure 9 – Sequence Diagram 2

3- Pharmaceutical Company Registration, Login, Add Medicine Dataset, Add Medicine, and Remove Medicine Scenarios:

The company logs in using its registered email and password.
The company adds a medicine dataset, which is stored in the system.
The company adds a new medicine with details like name, uses, and side effects.
The company deletes a previously added medicine, and the system confirms removal.

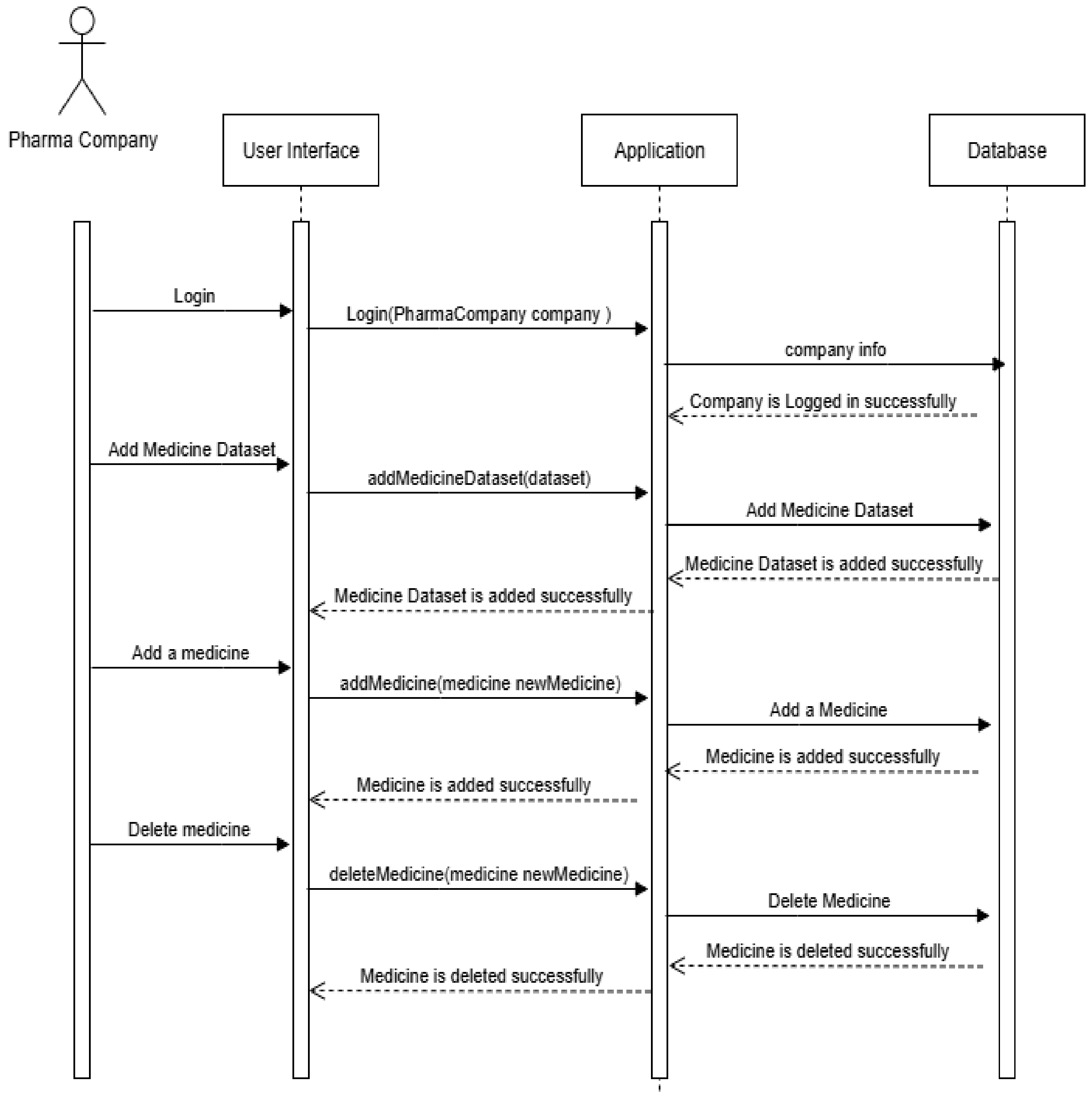

Pharma Company
User Interface
Application
Database
Login
Login(PharmaCompany company)
company info
Company is Logged in successfully
Add Medicine Dataset
addMedicineDataset(dataset)
Add Medicine Dataset
Medicine Dataset is added successfully
Medicine Dataset is added successfully
Add a medicine
addMedicine(medicine newMedicine)
Add a Medicine
Medicine is added successfully
Medicine is added successfully
Delete medicine
deleteMedicine(medicine newMedicine)
Delete Medicine
Medicine is deleted successfully
Medicine is deleted successfully

Figure 10 – Sequence Diagram 3

4- Doctor Login, View Patient Information, Enter Patient Diagnosis & Get Medicine Recommendation, and Save & Update Patient Records Scenarios:

The doctor logs in using their registered email and password.
The doctor retrieves a patient's details, including medical history and previous diagnoses.
The doctor enters a patient's details and diagnosis, and the system suggests appropriate medicines.

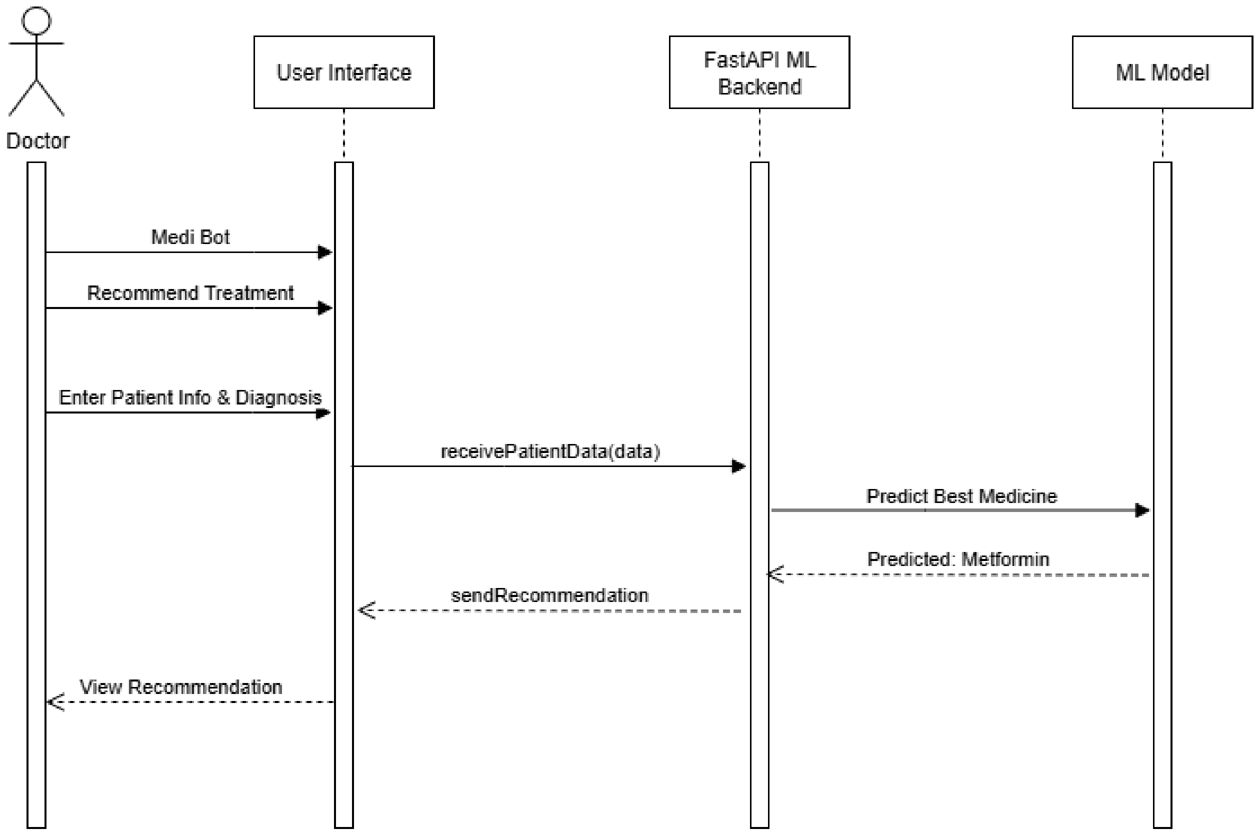


Figure 11 – Sequence Diagram 4

4.5 Project ERD

The ERD for the **Medica** system illustrates the relational structure among various entities involved in managing users, patients, visits, medicines, and pharmaceutical operations. The design supports modular data access, scalability, and integrity across the healthcare platform.

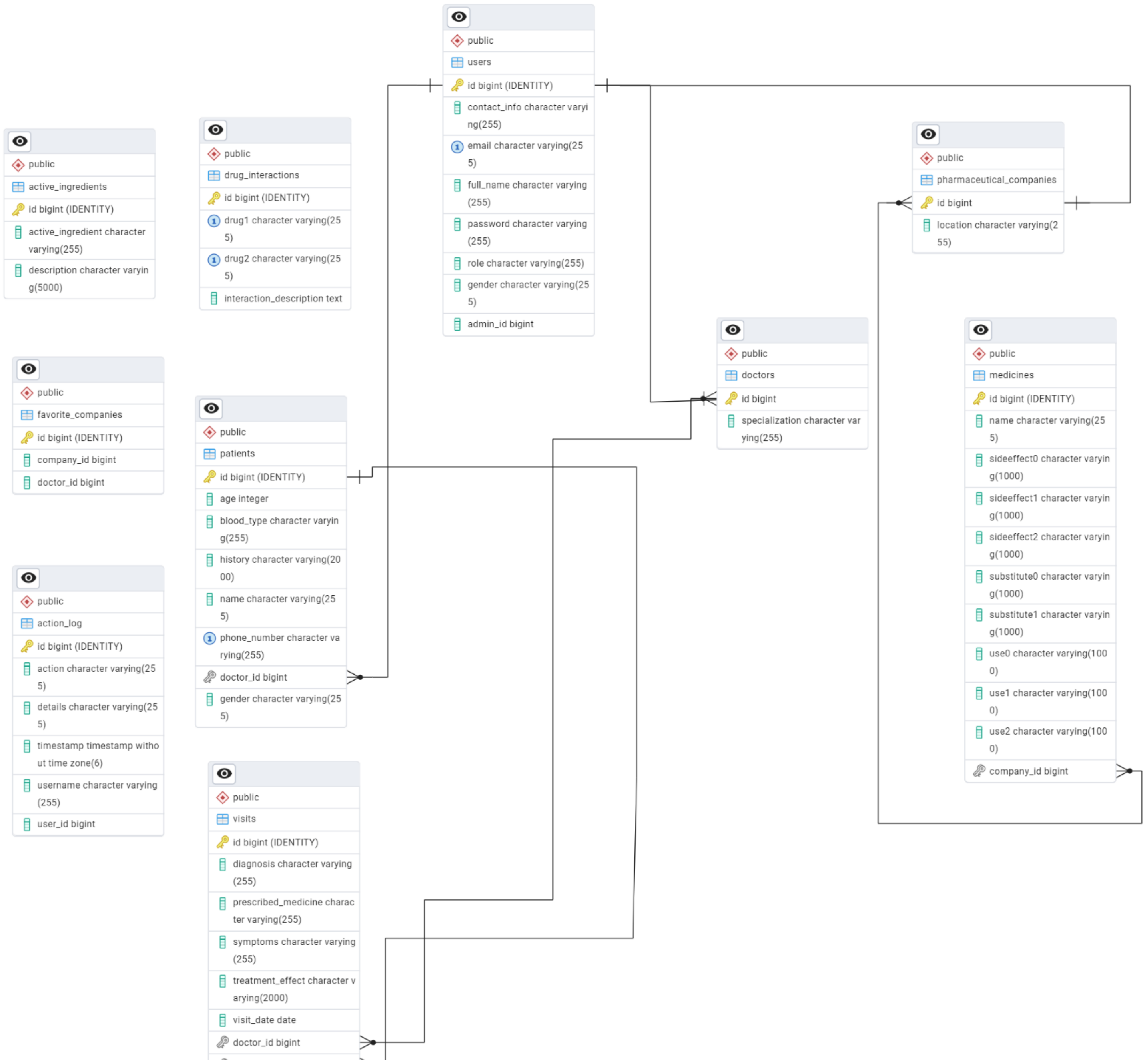

public
users
id bigint (IDENTITY)
contact_info character varying(255)
email character varying(255)
full_name character varying (255)
password character varying (255)
role character varying(255)
gender character varying(255)
admin_id bigint
public
active_ingredients
id bigint (IDENTITY)
active_ingredient character varying(255)
description character varying(5000)
public
drug_interactions
id bigint (IDENTITY)
drug1 character varying(255)
drug2 character varying(255)
interaction_description text
public
pharmaceutical_companies
id bigint
location character varying(255)
public
favorite_companies
id bigint (IDENTITY)
company_id bigint
doctor_id bigint
public
patients
id bigint (IDENTITY)
age integer
blood_type character varying(255)
history character varying(2000)
name character varying(255)
phone_number character varying(255)
doctor_id bigint
gender character varying(255)
public
doctors
id bigint
specialization character varying(255)
public
medicines
id bigint (IDENTITY)
name character varying(255)
sideeffect0 character varying(1000)
sideeffect1 character varying(1000)
sideeffect2 character varying(1000)
substitute0 character varying(1000)
substitute1 character varying(1000)
use0 character varying(1000)
use1 character varying(1000)
use2 character varying(1000)
company_id bigint
public
action_log
id bigint (IDENTITY)
action character varying(255)
details character varying(255)
timestamp timestamp without time zone(6)
username character varying (255)
user_id bigint
public
visits
id bigint (IDENTITY)
diagnosis character varying (255)
prescribed_medicine character varying(255)
symptoms character varying (255)
treatment_effect character varying(2000)
visit_date date
doctor_id bigint

Figure 12 – ERD

4.6 System GUI Design

The system's graphical user interface (GUI) is designed to be intuitive and user-friendly. The main goal was to ensure users can navigate between features with minimal learning curve.

We used Next.js and Tailwind CSS to implement the interface. The design follows a consistent blue color theme with clear labeling and responsive layouts. Below are snapshots of the screens:

Figure 4.6.0 – Landing Page

The landing page, covers the **Login** and **Register** functionalities, outlining the user authentication process for accessing the Medica platform.

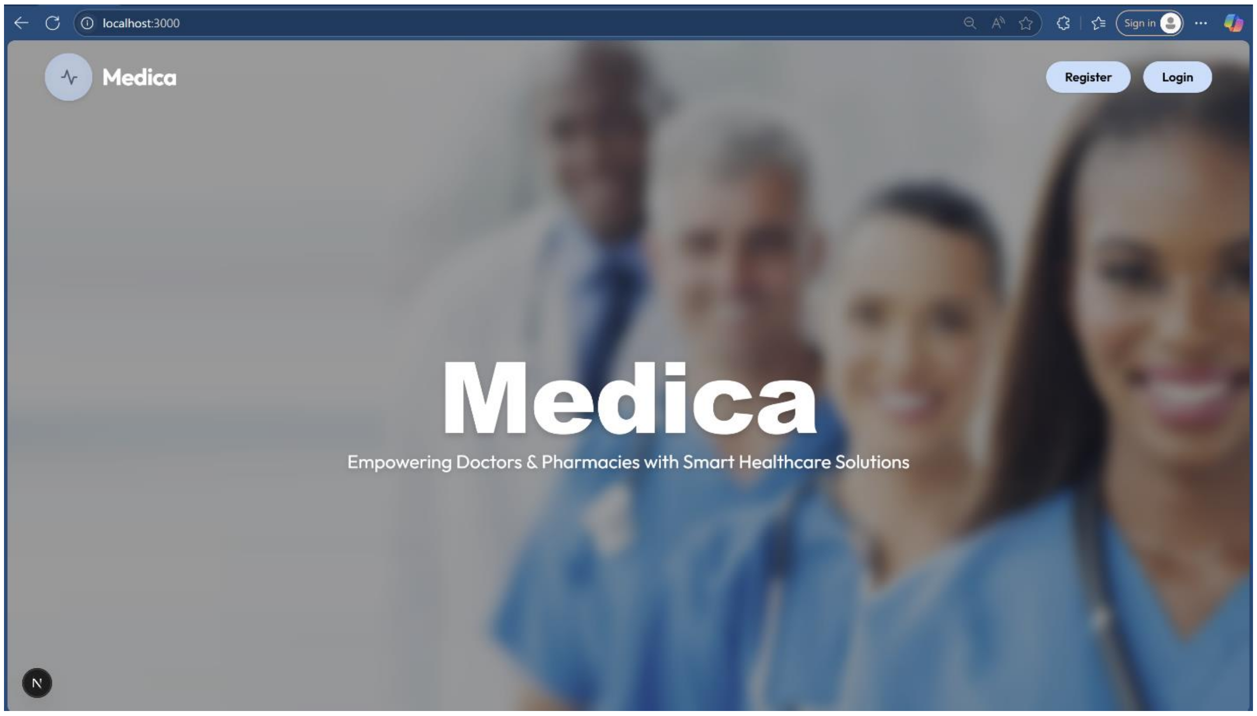

localhost:3000
Sign in
Medica
Register
Login
Medica
Empowering Doctors & Pharmacies with Smart Healthcare Solutions
N

Figure 13 – GUI design of the Landing page

Figure 4.6.1 – Login Page

The login page, allows users to log in using their credentials. It includes validation and error messages.

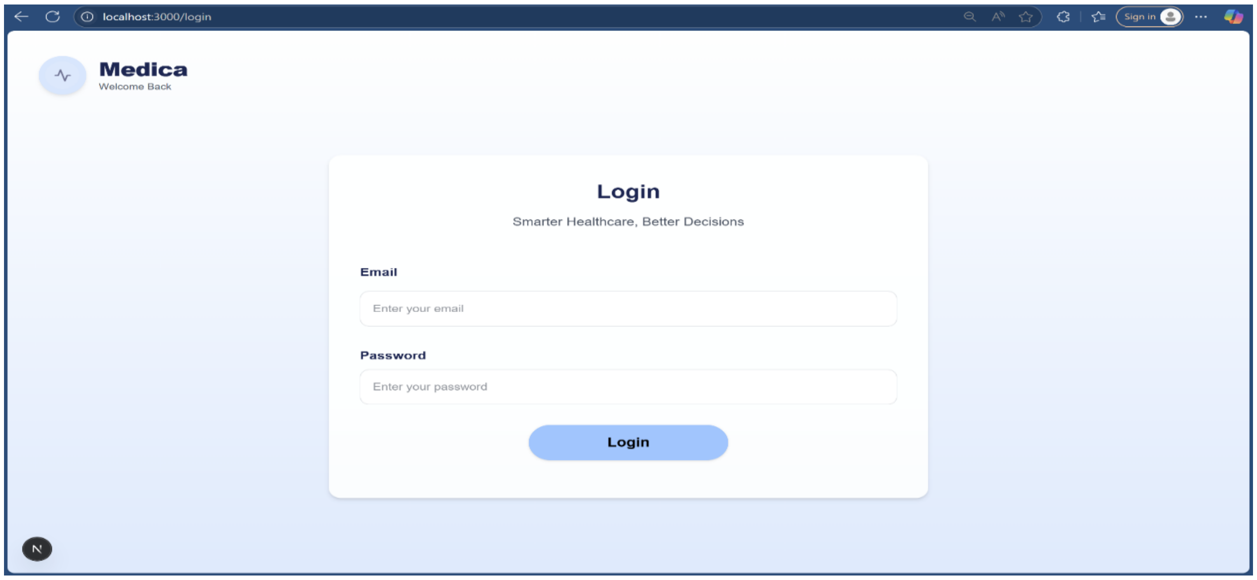


Figure 14 – GUI design of the Login page

Figure 4.6.2 – Admin Dashboard

The Admin Dashboard page ,it is the main interface presented to the admin upon successful login. The Main Action Cards Include: The **Doctors** card which allows the admin to browse, add, modify or remove doctor profiles; The **Admins** card which facilitates creation of new admin accounts, viewing, updating and removing accounts; and the **pharmaceutical** card that directs to the module where pharmaceutical companies can be viewed, updated or deleted.

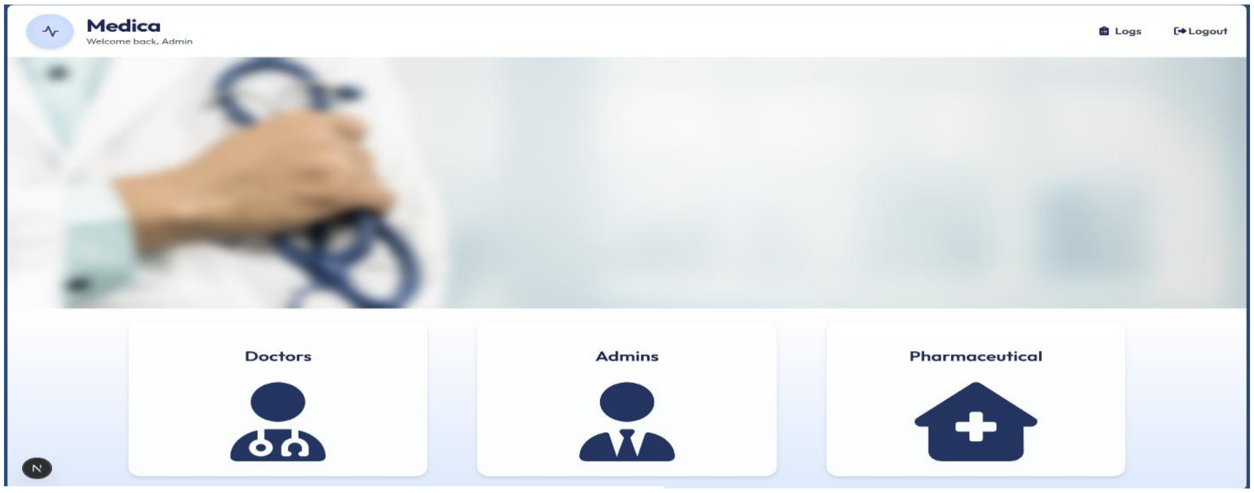
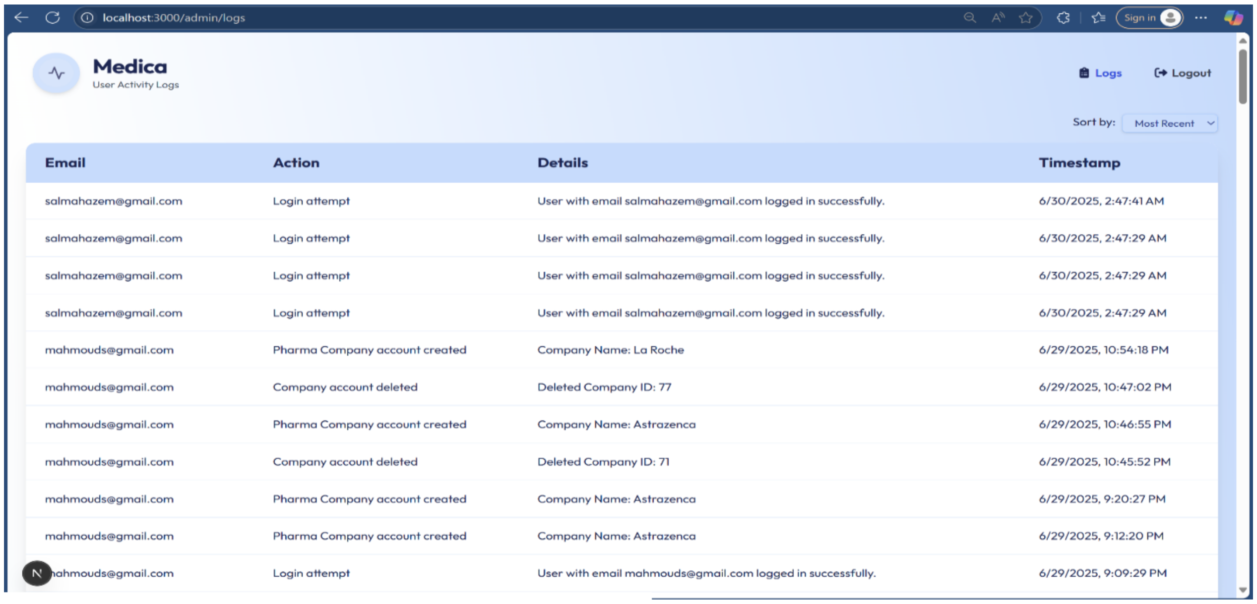


Figure 15 – GUI design of Admin dashboard page

Figure 4.6.3 – Logs Page

The Logs Page , it displays a history of system actions, including successful and failed login attempts, along with admin, doctor, and pharmaceutical company creation and deletion. Each log shows the user's email, action, and timestamp. Logs can be sorted by **most recent** or **oldest**.

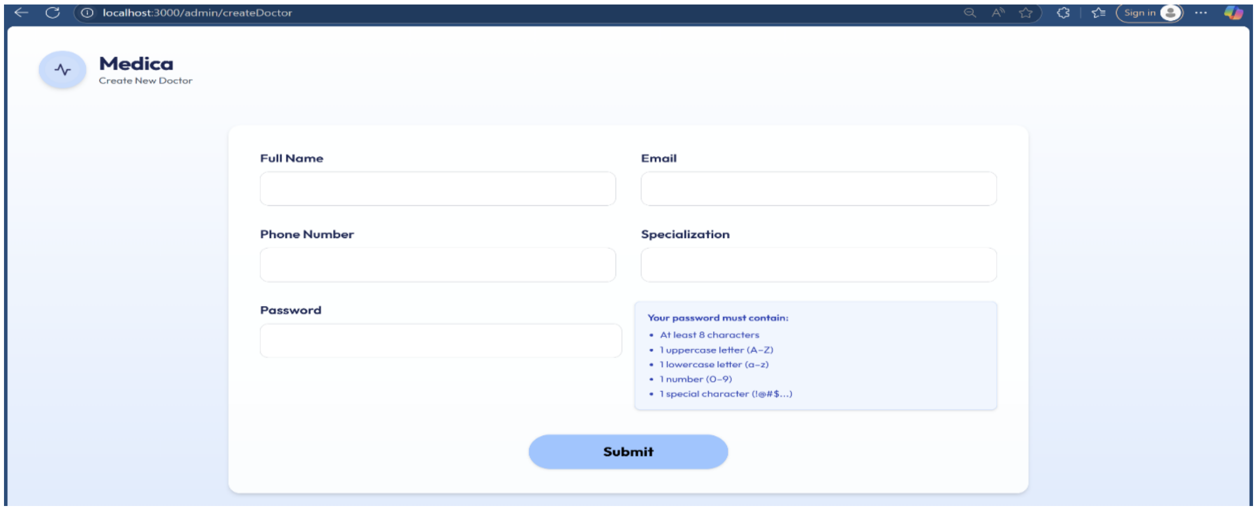


Email	Action	Details	Timestamp
salmahazem@gmail.com	Login attempt	User with email salmahazem@gmail.com logged in successfully.	6/30/2025, 2:47:41 AM
salmahazem@gmail.com	Login attempt	User with email salmahazem@gmail.com logged in successfully.	6/30/2025, 2:47:29 AM
salmahazem@gmail.com	Login attempt	User with email salmahazem@gmail.com logged in successfully.	6/30/2025, 2:47:29 AM
salmahazem@gmail.com	Login attempt	User with email salmahazem@gmail.com logged in successfully.	6/30/2025, 2:47:29 AM
mahmouds@gmail.com	Pharma Company account created	Company Name: La Roche	6/29/2025, 10:54:18 PM
mahmouds@gmail.com	Company account deleted	Deleted Company ID: 77	6/29/2025, 10:47:02 PM
mahmouds@gmail.com	Pharma Company account created	Company Name: Astrazenca	6/29/2025, 10:46:55 PM
mahmouds@gmail.com	Company account deleted	Deleted Company ID: 71	6/29/2025, 10:45:52 PM
mahmouds@gmail.com	Pharma Company account created	Company Name: Astrazenca	6/29/2025, 9:20:27 PM
mahmouds@gmail.com	Pharma Company account created	Company Name: Astrazenca	6/29/2025, 9:12:20 PM
ahmouds@gmail.com	Login attempt	User with email mahmouds@gmail.com logged in successfully.	6/29/2025, 9:09:29 PM

Figure 16 – GUI design of of the Logs page

Figure 4.6.4 – Create Doctor page

The Create Doctor page , it enables authorized users to add new doctors by filling in required details. Password restrictions are clearly mentioned to ensure strong credentials.



Your password must contain:
- At least 8 characters
- 1 uppercase letter (A–Z)
- 1 lowercase letter (a–z)
- 1 number (0–9)
- 1 special character (!@#$...)

Figure 17 – GUI design of the Create doctor page

Figure 4.6.5 – Doctors Homepage

The Doctor Dashboard page, it provides doctors with quick access to key features, including, **Search Medicine** Checking **Drug Interaction,** Assigned **patients, Pharma Companies** and **Medi Bot.**

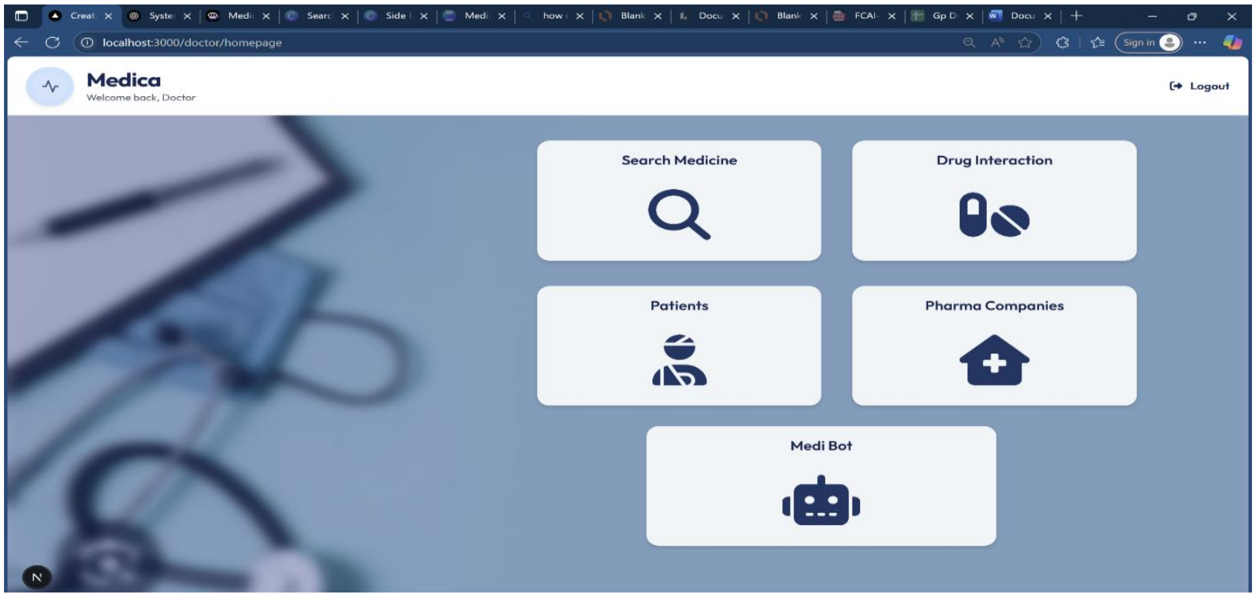
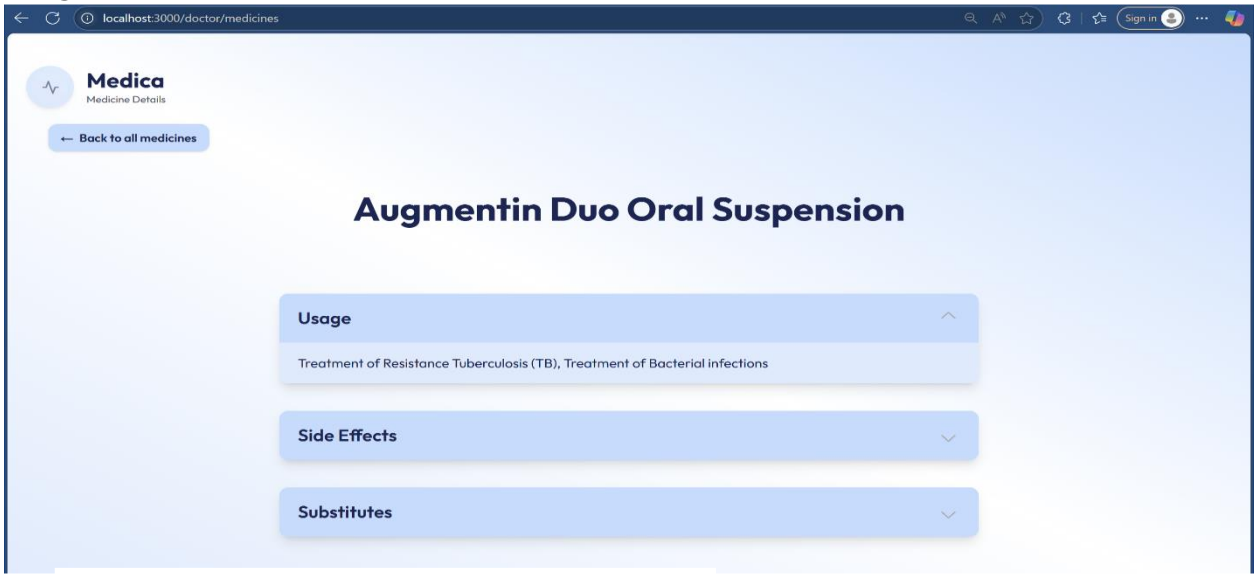


Figure 18 – GUI design of the Doctor Homepage

Figure 4.6.6 - Medicine Details Page

After selecting a medicine from the search suggestions, the system displays a detailed page showing: **Usages**, **Side Effects**, and **Substitutes.**



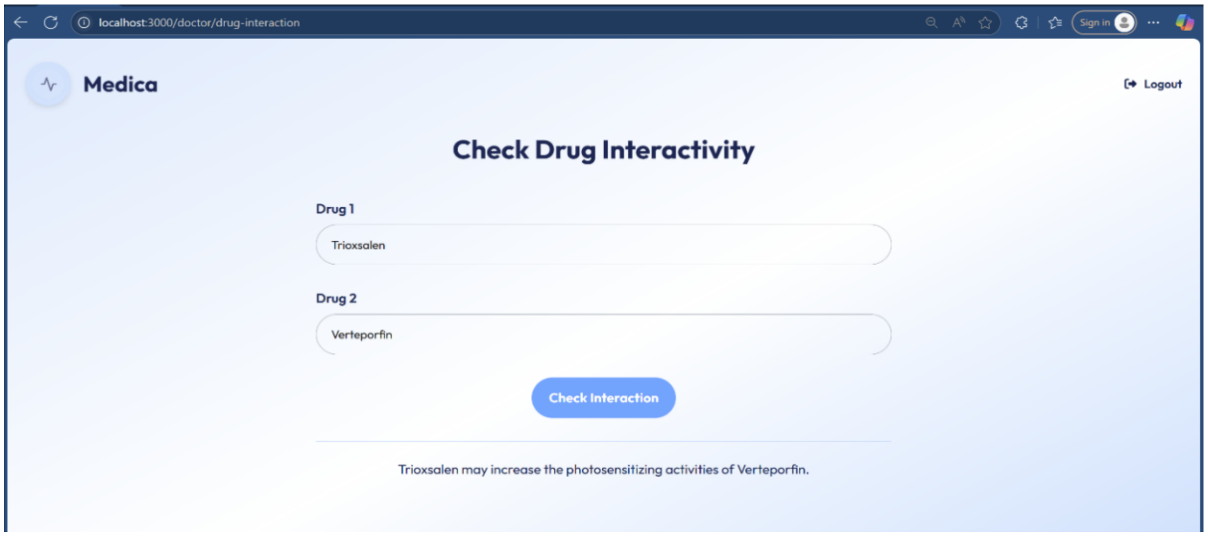
Augmentin Duo Oral Suspension

Usage: Treatment of Resistance Tuberculosis (TB), Treatment of Bacterial infections

Figure 19 – GUI design of the Medicine details page

Figure 4.6.7 - Drug Interactions Page

This page allows doctors to enter the names of two drugs to check for potential interactions. The system displays any known interactions between them.



Check Drug Interactivity

Drug 1: Trioxsalen

Drug 2: Verteporfin

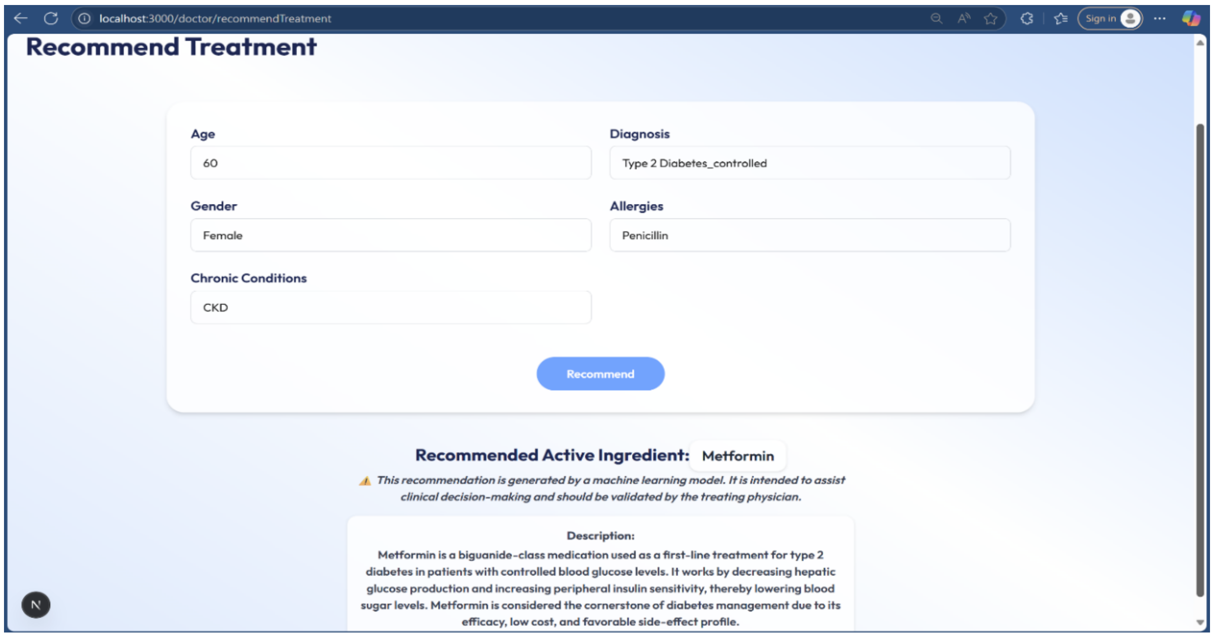
Trioxsalen may increase the photosensitizing activities of Verteporfin.

Figure 20 – GUI design of the Check drug interactivity page

Figure 4.6.8 - Recommend Treatment Page

Doctors enter basic patient data (e.g., age, chronic condition, disease), and the system recommends the **most suitable active ingredient** based on trained ML model.

\



Recommend Treatment

Age: 60	Diagnosis: Type 2 Diabetes_controlled
Gender: Female	Allergies: Penicillin
Chronic Conditions: CKD	

Recommended Active Ingredient: Metformin

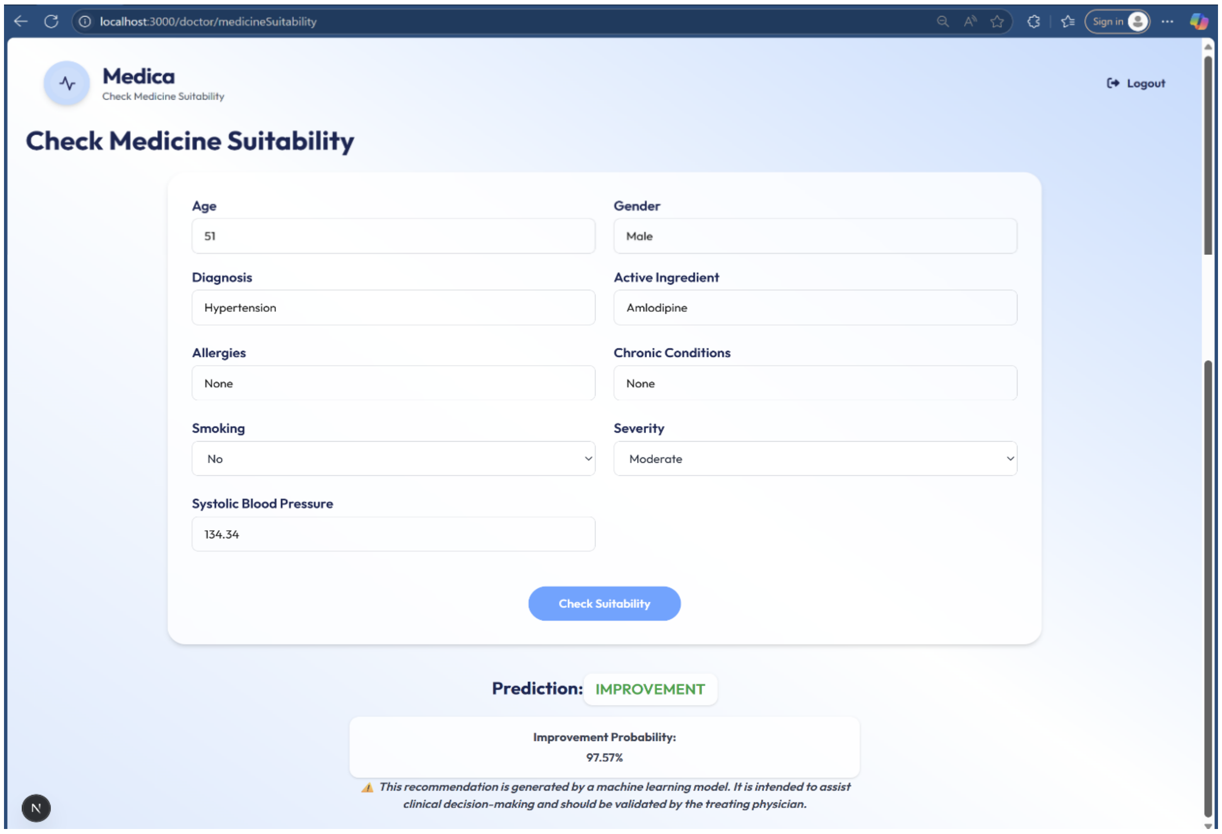
⚠ *This recommendation is generated by a machine learning model. It is intended to assist clinical decision-making and should be validated by the treating physician.*

Description:
Metformin is a biguanide-class medication used as a first-line treatment for type 2 diabetes in patients with controlled blood glucose levels. It works by decreasing hepatic glucose production and increasing peripheral insulin sensitivity, thereby lowering blood sugar levels. Metformin is considered the cornerstone of diabetes management due to its efficacy, low cost, and favorable side-effect profile.

Figure 21 – GUI design of the Recommend treatment page

Figure 4.6.9 - Check Medicine Suitability Page

Like the recommendation page, doctors enter patient data along with a selected **active ingredient**. Inputs like **diagnosis** and based on the chosen diagnosis it asks for specific input of test results, also takes inputs gender, allergies, smoking, severity of the disease, and chronic conditions are dropdowns with autocomplete for increasing the usability.


localhost:3000/doctor/medicineSuitability
Medica
Check Medicine Suitability
Logout
Check Suitability

Check Medicine Suitability

Age: 51
Gender: Male
Diagnosis: Hypertension
Active Ingredient: Amlodipine
Allergies: None
Chronic Conditions: None
Smoking: No
Severity: Moderate
Systolic Blood Pressure: 134.34

Prediction: IMPROVEMENT

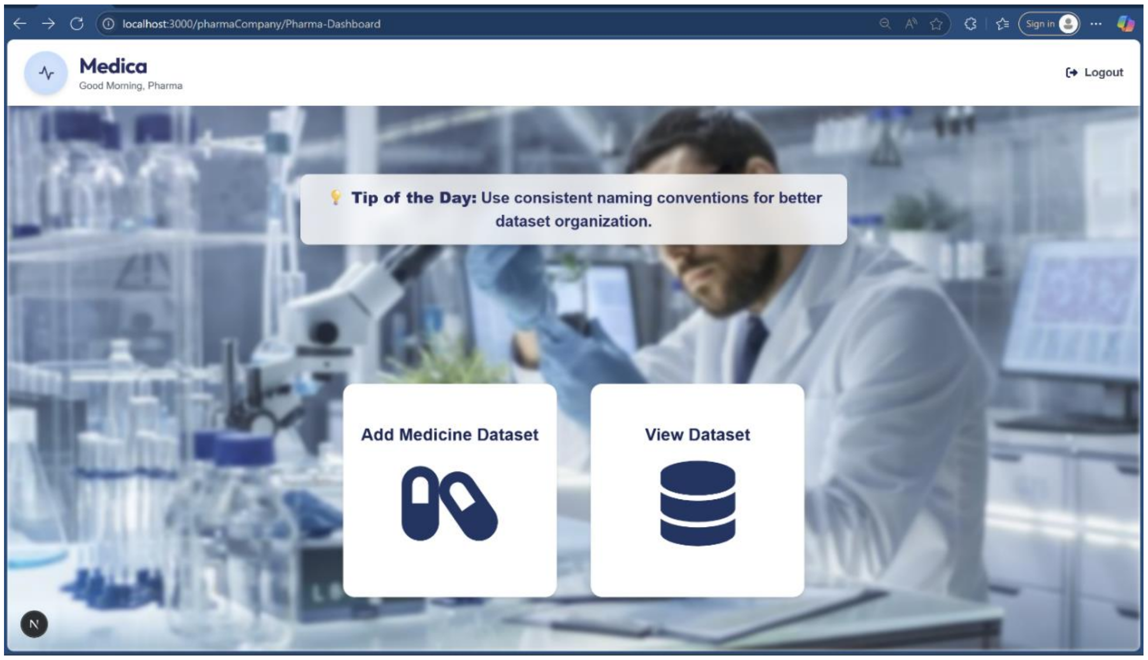
Improvement Probability:
97.57%

⚠ *This recommendation is generated by a machine learning model. It is intended to assist clinical decision-making and should be validated by the treating physician.*

Figure 22 – GUI design of the Check medicine suitability page

Figure 4.6.10 - Pharmaceutical Company Homepage

This is the main dashboard for logged-in pharmaceutical companies. It provides access to key features. The dashboard also displays: **Tip of the Day** which Shows a useful medical or system-related tip to enhance awareness



Tip of the Day: Use consistent naming conventions for better dataset organization.

Figure 23 – GUI design of the Pharmaceutical company Homepage

In Conclusion, The system aims to enhance usability, by including features such as:

- **Autocomplete search bars** for quick access to data
- **Dropdown lists** for structured inputs (e.g., gender, blood type, diagnosis)
- **Minimalist layouts** to reduce clutter and guide users smoothly through their tasks

Overall, the interface was designed with the end user in mind—whether a doctor, admin, or pharmaceutical representative—to ensure a seamless and efficient experience.

Chapter 5 : Implementation and Testing System running and samples of the applied test cases

Below are the **Implementation** and **Testing** details, including the system running status and samples of the applied test cases, presented in tables.

Features :

1-Login

Test Case ID	Description	Input	Output
TC-Login-01	**Tests form validation when both fields are left empty**	**Email: (empty) Password: (empty)**	**Warning: "Email is required" Warning: "Password is required"**
TC-Login-02	**Tests form validation when password field is empty**	**Email: salmahazem@gmail.com** **Password: (empty)**	**Warning: "Password is required"**
TC-Login-03	**Tests form validation when email field is empty**	**Email: (empty) Password:secret123**	**Warning: "Email is required"**
TC-Login-04	**Tests system response to incorrect credentials**	**Email: salmahazem@gmail.com** **Password:wrongpass**	**Warning: "Incorrect email or password"**
TC-Login-05	**Tests successful login with valid credentials**	**Email: salmahazem@gmail.com** **Password: correctpass**	**Redirect to admin homepage**

Table 2: Login feature test cases

Admin Feature:

2-Create Doctor

Test Case ID	Description	Input	Output
TC-DOC-01	**Tests successful creation with valid inputs**	**Valid inputs for all fields**	**Doctor created successfully Redirects to View all doctors**
TC-DOC-02	**Tests email domain validation**	**Email: salma@medica.org**	**Warning: "Only Gmail, Yahoo, Hotmail, Outlook emails are allowed"**
TC- DOC-04	**Tests phone number length validation**	**Phone: 0105**	**Error: " Phone number must be exactly 11 digits. "**
TC-DOC-05	**Tests password complexity and length**	**Password: salma1@**	**Error: " Password is not secure enough. Please follow the declared password criteria. "**
TC-DOC-06	**Tests form submission with incomplete data**	**One or more fields left blank / All Empty Fields**	**Error message: "Please fill in all fields"**

Table 3: Create Doctor feature test cases

The same test cases are applicable to both the 'Create Pharmaceutical Company' and 'Create Admin' functionalities, with the only differences being the specific input fields required for each role.

Doctor Features:

3-Recommend Medicine

Test Case ID	Description	Input	Output
TC_RECOMMEND_01	Valid input submission	Valid inputs for all fields	Success – Displays recommended medicine and its description
TC_RECOMMEND_02	Submission with missing fields	Leave 1+ fields empty	Failure – Displays: " Please fill in all the fields before submitting."
TC_RECOMMEND_03	Age below allowed limit	Age: -5	Warning– "Value must be greater than or equal to 1"
TC_RECOMMEND_04	Case-insensitive dropdown matching	Type "fEm" in Gender field	Success – Filters and matches "Female"

Table 4: Recommend Medicine test cases

4-Check Medicine Suitability

Test Case ID	Description	Input	Output
TC-MS-01	Submitting form with invalid age (less than 1 or more than 120)	Age = -5 or 130	Error message shown: " Please enter a valid age between 1 and 120. "
TC-MS-02	Submitting form without selecting gender	Gender = ""	Error message shown: “Please fill in all the fields before submitting."
TC-MS-03	Displaying correct test field for Type 2 Diabetes diagnosis	Diagnosis = "Type 2 Diabetes"	Shows input field: "HbA1c"
TC-MS-04	Entering non-numeric value in numeric test field	HbA1c = "abc"	Error message shown:”Please enter a valid number for HbA1c”
TC-MS-05	Submitting valid form and getting prediction and probability	Valid inputs for all fields	Displays prediction (e.g., "Improvement") and probability (e.g., "92.14%")

Table 5: Check Medicine Suitability feature test cases

5- Drug Interactions

Test Case ID	Scenario	Input (Drug 1 / Drug 2)	Output
TC_INTER_01	Valid interaction	Trioxsalen / Verteporfin	Shows known interaction details between the two drugs
TC_INTER_02	No known interaction	Venetoclax / Trioxsalen	Displays: “No known interaction between Venetoclax and Trioxsalen”
TC_INTER_03	Same drug entered twice	Verteporfin / Verteporfin	Displays: “Please enter two different drugs to check interaction.”
TC_INTER_04	One or both fields empty	— / —	Displays: “Both fields are required.”

Table 6: Drug Interactions feature test cases

6-Search Medicine

Test Case ID	Description	Input	Output
TC_MED_SEARCH_01	Valid search with existing medicine name	"Augmentin Duo Oral Suspension"	List displays matching medicine(s)
TC_MED_SEARCH_02	Partial name search	"au"	Medicines like "augmentin duo oral suspension “, “augmentin 625 duo tablet" are displayed
TC_MED_SEARCH_03	Case-insensitive search	"augmentin", "AUGMENTIN", "aUGMEnTiN"	Results should match regardless of case
TC_MED_SEARCH_04	Search with no match	"Xyzmed999"	Message: " Sorry, no medicine found. Please try a different name. "
TC_MED_SEARCH_05	Incomplete Search	"A"	Message: " Sorry, no detailed info found for "A "

Table 7: Search Medicine feature test cases

Pharmaceutical Company Feature:

7-Upload Pharmaceutical Company Medicine Dataset

Test Case ID	Description	Input	Output
TC-ADD-MEDICINE-01	**Upload valid CSV with new medicines**	**Valid .csv file**	**Success – "5 medicines added successfully"**
TC-ADD-MEDICINE-02	**Upload fails when all medicines already exist**	**.csv file where all medicines already exist in DB**	**Failure – "0 medicines added successfully already added by another company " - providing medicine names**
TC-ADD-MEDICINE-03	**Upload partially succeeds**	**.csv file with mix of new and existing medicines**	**Success – "3 medicines added successfully, 2 already exist" -providing names of the medicines already existed**
TC-ADD-MEDICINE-04	**Drag-and-drop upload works**	**Drag .csv file into upload area**	**Success – Upload is triggered and processed**

Table 8: Upload Dataset test cases

Chapter 6: Machine Learning Report

6.Machine learning Report

Introduction to the Machine Learning Functionalities:

Our two machine learning functionalities aim to enhance clinical decision-making using machine learning. The first, ″**Enter Patient Diagnosis & Get Medicine Recommendation**″, suggests appropriate medications based on a patient's diagnosis and medical history. The second, ″**Assess Medicine Suitability Using Machine Learning**″, evaluates the suitability of a prescribed medication for a given condition. Both features support healthcare professionals in prescribing effective treatments by leveraging clinical data to improve patient outcomes.

Dataset Generation:

- **Initial attempts to use existing medical datasets were unsuccessful due to:**
  - Missing required features for both medication recommendation and suitability assessment.
  - Limited coverage of conditions, medications, or clinical measurements.
- **To meet specific model requirements, a custom synthetic dataset was created.**
- **Challenges during development:**
  - Early versions had high accuracy but included unrealistic or non-medical features.
  - More realistic datasets introduced complexity, missing data, and reduced accuracy.
- **Final outcome:**
  - Two refined synthetic datasets were produced.
  - These datasets balance medical relevance and data quality, forming the foundation for both machine learning features.

6.1 Machine learning model for predicting medicine:

6.1.1- Initial Dataset Generation:

To develop the medicine prediction model, a synthetic dataset—**predict_medicine_Dataset[3]**—was created, as existing public datasets lacked key clinical features such as allergies, chronic conditions, and diagnosis-medication mappings.

Objective:
Enable the model to recommend appropriate medication based on patient-specific data including age, gender, diagnosis, allergies, and chronic conditions.

Key Elements of Dataset Design:

- **Demographics & Allergies:**
  - Age and gender affect drug choice and dosage.
  - Allergy profiles (e.g., Penicillin, NSAIDs) constrain medication options.
- **Diagnosis-Medication Mapping:**
  - Diagnoses include clinical severity or subtypes (e.g., *Hypertension_<=50*, *Type 2 Diabetes_uncontrolled*).
  - Each diagnosis is linked to a suitable medication based on clinical rules.
- **Chronic Conditions:**
  - Comorbidities (e.g., COPD, CKD) influence medication safety and efficacy.
- **Target Variable:**
  - The model predicts a medication (multi-class classification), selected via rule-based logic to reflect real-world prescribing practices.
- **Dataset Generation:**
  - Built using a custom script (*predict_medicine_Generator.py*), generating 15,000 patient records.
  - Includes 5% label noise to simulate real-world inconsistencies.

This dataset supports clinically relevant, individualized medicine predictions by encoding real-world constraints into synthetic patient records.

6.1.2-Data Preprocessing:

To prepare the dataset for supervised learning, several preprocessing steps were applied to ensure data quality, consistency, and compatibility with classification algorithms.

1- Data Cleaning and Filtering:

- Original dataset: 15,000 records with six features: age, gender, diagnosis, allergies, chronic conditions, and medicine.
- Rows with Azithromycin were removed due to class imbalance, improving model generalization and stability.

2- Handling Missing Values:

- Categorical features (allergies, chronic conditions): Missing values filled with "None".
- Numeric feature (age): No missing values; no imputation required.

3- Categorical Encoding:

- **Label Encoding** applied to:
  - gender → gender_enc
  - diagnosis → diag_enc
  - allergies → allergy_enc
  - chronic conditions → chronic_enc
- **Target variable (medicine)** encoded as multi-class label.
- Encoding preserved categorical semantics, suitable for tree-based models.

4- Feature Set Construction:

- **Feature set X:** age, gender_enc, diag_enc, allergy_enc, chronic_enc
- **Target y:** Encoded medicine

5- Train/Test Split:

- **Stratified 80/20 split** to maintain class distribution across training and testing sets, mitigating bias from class imbalance.

6.1.3 - Feature Selection:

The selection of features for this model was guided by clinical relevance and practical considerations in medication recommendation. Each feature included in the final dataset plays a meaningful role in influencing the appropriateness of a prescribed medication for a given patient profile.

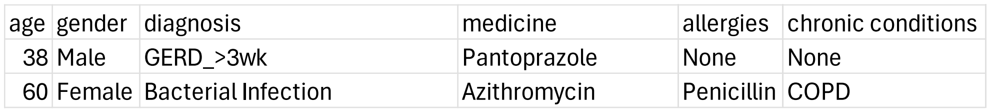
The final set of features used to train the model includes:

- **Age**: A continuous variable representing the patient's age. Age is a critical determinant of medication suitability, as pharmacokinetics "What the body does to the drug" and pharmacodynamics " What the drug does to the body" often vary significantly across age groups.
- **Gender**: Gender can influence how patients metabolize certain drugs and the likelihood of experiencing adverse effects. Including this feature helps account for known gender-based variations in treatment efficacy.
- **Diagnosis**: The primary condition being treated (e.g., Heart Failure, GERD, Type 2 Diabetes). This is the central variable in determining medication appropriateness, as each diagnosis has an associated set of clinically recommended treatments.
- **Allergies**: Known allergies (e.g., Penicillin, NSAIDs) are essential for filtering out medications that may cause harmful reactions. This feature helps the model learn to avoid recommending contraindicated drugs.

- **Chronic Conditions**: Comorbidities (e.g., CKD, COPD) often complicate treatment decisions. Some medications may be contraindicated or require dosage adjustments in patients with chronic conditions, making this feature crucial for individualized recommendations.

These features were not selected from a larger set but rather represent the complete and clinically justified set of attributes necessary for the prediction task. Their inclusion ensures that the model reflects real-world decision-making processes, where age, gender, diagnosis, allergies, and chronic conditions jointly influence the choice of medication. This comprehensive feature set supports the model's ability to provide safe and effective medication recommendations tailored to each patient profile.

Sample of the dataset:



age	gender	diagnosis	medicine	allergies	chronic conditions
38	Male	GERD_>3wk	Pantoprazole	None	None
60	Female	Bacterial Infection	Azithromycin	Penicillin	COPD

Figure 24 – Sample of the dataset of the recommend medicine feature

6.1.4- Model Evaluation:

The models evaluated for this feature include **XGBoost**, **LightGBM**, and **CatBoost**. These are advanced gradient boosting algorithms known for their high performance in structured data classification tasks. Each model was trained on the preprocessed dataset and assessed based on its ability to accurately predict the appropriate medication given a patient’s diagnosis and medical history.

The following table compares the performance of different models evaluated on the test set:

Model Name	**Accuracy**
XGBoost	96.30%
CatBoost	89.80%
LightGBM	96.30%

Table 9: Predict Medicine Evaluation Table

The **XGBoost** model emerged as the best-performing model, achieving an accuracy of 96.30%, which matches that of **LightGBM** but higher than the other models, such as CatBoost (89.80%).

After training and evaluating the models, **XGBoost** was selected as the best model based on its testing accuracy.

Confusion Matrix:

The confusion matrix for the best model (**XGBoost**) was generated to evaluate the model's classification performance. The matrix shows the true positive, false positive, true negative, and false negative counts for the predicted medications:

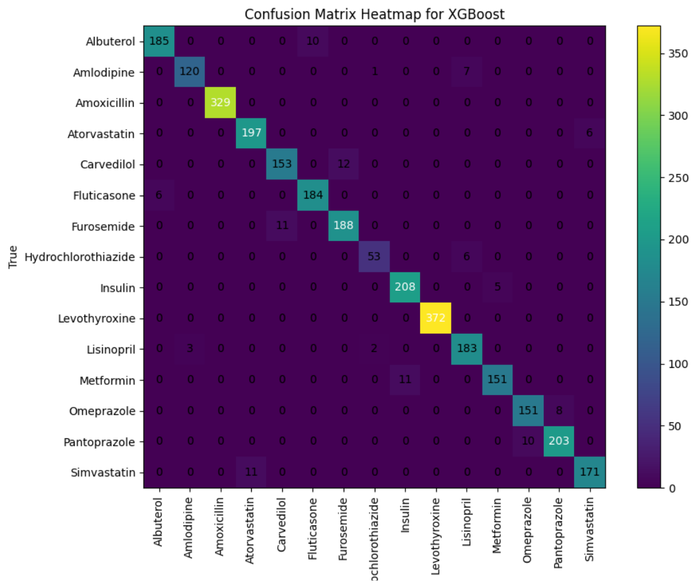
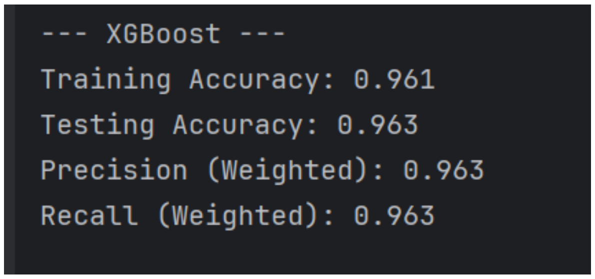


Figure 25 – Recommend medication model confusion matrix

The following image shows the accuracy, precision, and recall metrics of the XGBoot which is the best performing model:



```
--- XGBoost ---
Training Accuracy: 0.961
Testing Accuracy: 0.963
Precision (Weighted): 0.963
Recall (Weighted): 0.963
```

Figure 26 – Recommend medication model metrics

6.2- Machine learning model for predicting patient improvement :

6.2.1- Initial Dataset Generation:

This functionality uses machine learning to assess whether a prescribed medication is likely to result in patient improvement. Due to the lack of a comprehensive real-world dataset with the necessary features, a synthetic dataset—**PredictImprovementDataset[4]**—was created after evaluating over 10 dataset variants.

Development Process:

- **Existing Data Limitations:**
  Public datasets lacked essential combinations of features such as medical test results, diagnoses, and treatment outcomes. Some synthetic datasets achieved high accuracy using unrealistic features (e.g., *severity_score*), while others were medically realistic but posed challenges such as data sparsity, complexity, and low model accuracy.
- **Final Dataset Selection:**
  The chosen dataset balanced medical realism and model performance, supporting accurate predictions of treatment suitability grounded in clinically relevant data.

Key Components of the Dataset:

1. **Patient Demographics:**
   - *Age, gender, smoking status*—influence treatment efficacy and health outcomes.
   - Example: Smoking affects drug metabolism, especially in cardiovascular and respiratory treatments**.**
2. **Medical Conditions:**

   Includes common and clinically significant diagnoses:
   - Hypertension
   - Type 2 Diabetes
   - Hyperlipidemia
   - Heart Failure
   - GERD (Gastroesophageal Reflux Disease)
   - Hypothyroidism

   These were selected for their prevalence and distinct medication requirements.

3. **Medical Tests (Condition-Specific):**
   - Systolic BP (Hypertension)
   - HbA1c (Type 2 Diabetes)
   - LDL Cholesterol (Hyperlipidemia)
   - BNP (Heart Failure)
   - Endoscopy Results (GERD)
   - TSH (Hypothyroidism)

These clinical indicators reflect disease severity and guide treatment suitability assessment.

4. **Target Variable – Improvement:**
   - Binary outcome (*Improved* or *Not Improved*) based on:
     - Patient characteristics
     - Diagnosis-specific test results
     - Prescribed medications
   - An *improvement score* was calculated, with the 55th percentile used as a threshold for classification.

This dataset forms the foundation for training a model capable of evaluating whether a treatment is likely to be effective, using medically grounded, structured inputs.

6.2.2- Data Preprocessing:

To prepare the dataset for model training, several preprocessing steps were applied to handle missing values, encode categorical variables, bin and scale continuous features, and ensure balanced data partitioning.

1. Handling Missing Data:

- ***Categorical features*** **(e.g., allergies, chronic_conditions, medicine):**
  Missing values were filled with "None" to preserve categorical integrity during label encoding.
- ***Numeric features*** **(e.g., Blood Pressure, HbA1c, LDL):**
  Missing and invalid entries were coerced to NaN and imputed using the column mean to maintain distribution and avoid training errors.

2. Categorical Variables Encoding:

- **Label Encoding** was applied to all categorical variables (gender, diagnosis, allergies, chronic_conditions, medicine), converting each category into an integer for compatibility with ML models.For example, the feature gender would be encoded as 0 for "Male" and 1 for "Female", while diagnoses would also be transformed into integers based on their categorical values.

3. Binning Continuous Variables:

- **Age Grouping**: The age feature was converted into an ordinal variable by binning it into discrete age groups: [0-30], [30-50], [50-70], and [70+]. This transformation allows the model to treat age as a categorical variable rather than a continuous one.

4. Scaling Continuous Variables:

- **Standardizing Age**: StandardScaler was used to normalize the age feature (mean = 0, std = 1) to prevent scale-related bias during training, scaling the age feature helps prevent it from dominating the other variables during model training.
- **Use of Scaler Across Training and Testing**: The same scaler was consistently applied to both training and test sets for future compatibility.

5. Final Dataset Preparation:

- **Feature set X:** All predictors after preprocessing
- **Target variable y:** Binary Improved outcome indicating whether a patient's condition improved after treatment
- **Train-test split:** 80/20 using **stratified sampling** to preserve class distribution and mitigate imbalance-related bias.

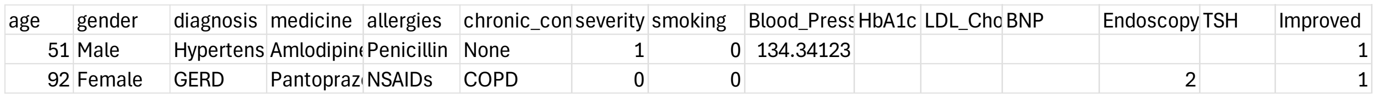
6.2.3 - Feature Selection:

The set of features that were retained in the model included:

- **Age**: A critical factor in determining medication suitability, influencing how a patient responds to treatments.
- **Gender**: Gender differences may affect the effectiveness of certain medications.
- **Diagnosis**: The medical condition (e.g., hypertension, diabetes) directly influences the choice of medication and its potential efficacy.
- **Medicine**: The prescribed medication is one of the primary inputs for the model, with its suitability being the focus of evaluation.
- **Allergies and Chronic Conditions**: These factors are crucial for understanding how a patient's medical history might influence the choice and effectiveness of medication.
- **Severity**: The severity of the condition affects treatment decisions and response.
- **Smoking**: Smoking is known to influence the effectiveness of certain medications and overall health improvement.
- **Medical Tests**: Disease-specific tests are critical for understanding the patient's condition more deeply, aiding in assessing how well the prescribed medication would work.

These features were chosen due to their direct relevance to predicting the improvement of a patient's condition based on medication and medical history. Each feature was considered carefully, and extensive testing was done to assess the impact of various combinations on model performance.

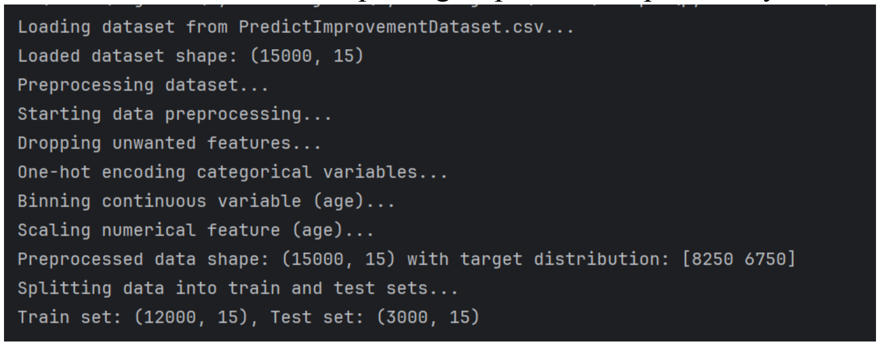
Sample of the dataset:



age	gender	diagnosis	medicine	allergies	chronic_con	severity	smoking	Blood_Press	HbA1c	LDL_Cho	BNP	Endoscopy	TSH	Improved
51	Male	Hypertens	Amlodipine	Penicillin	None	1	0	134.34123						1
92	Female	GERD	Pantopraz	NSAIDs	COPD	0	0					2		1

Figure 27 – Sample of the dataset of the predict medicine

The following screenshot illustrates the steps performed, which align with the preprocessing, feature selection, and data splitting steps outlined previously:



```
Loading dataset from PredictImprovementDataset.csv...
Loaded dataset shape: (15000, 15)
Preprocessing dataset...
Starting data preprocessing...
Dropping unwanted features...
One-hot encoding categorical variables...
Binning continuous variable (age)...
Scaling numerical feature (age)...
Preprocessed data shape: (15000, 15) with target distribution: [8250 6750]
Splitting data into train and test sets...
Train set: (12000, 15), Test set: (3000, 15)
```

Figure 28 – Predict medicine suitability model suitability

6.2.4- Model Evaluation:

After selecting the feature set, various machine learning models, including **XGBoost**, **CatBoost**, **LightGBM**, and **Random Forest**, were evaluated.

The model performance was assessed using accuracy, with the best-performing model being selected for further analysis. The accuracy and feature importance were visualized to aid in the understanding of which features contributed most to the final prediction.

The following table compares the performance of different models evaluated on the test set:

Model Name	Accuracy
XGBoost	96.07%
CatBoost	96.50%
LightGBM	96.50%
RandomForest	95.80%

Table 10: Predict Medicine Evaluation Table

The **LightGBM** model emerged as the best-performing model, achieving an accuracy of **96.50%**, which matches that of **CatBoost** but higher than the other models, such as **XGBoost** (96.07%) and **RandomForest** (95.80%).

Confusion Matrix:

This following heatmap highlights the model's accuracy in predicting both classes (**Improved** and **Not Improved**) and provides insights into the model's performance, with relatively low false positives and false negatives:

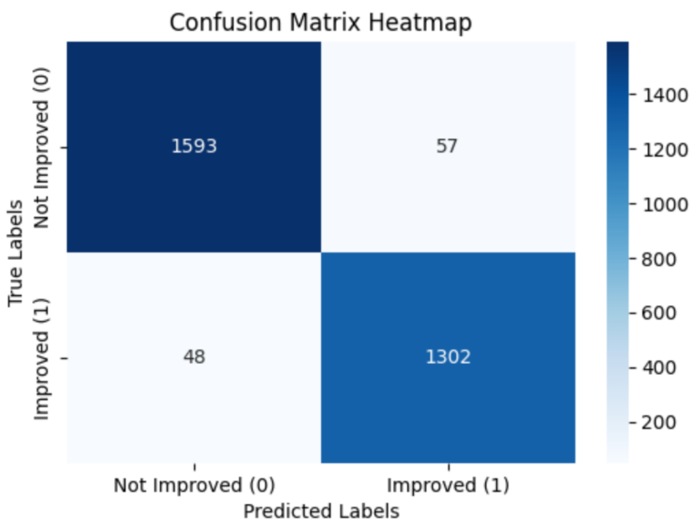


Figure 29 – Predict medicine suitability model confusion

Feature importance diagram:

This following image illustrates the **feature importance** for the trained model. Key features such as **chronic_conditions**, **severity**, and **smoking** are shown to be the most important for predicting the target variable. Other significant features include **blood pressure (Systolic_BP)**, **age_group**, and **TSH**. Features like **gender** and **age** were less influential, which aligns with the model's focus on more clinically relevant factors.

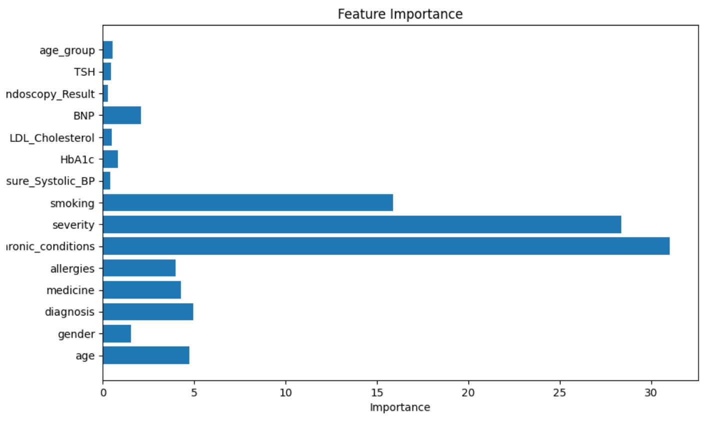


Figure 30 – Predict medicine suitability model feature

References

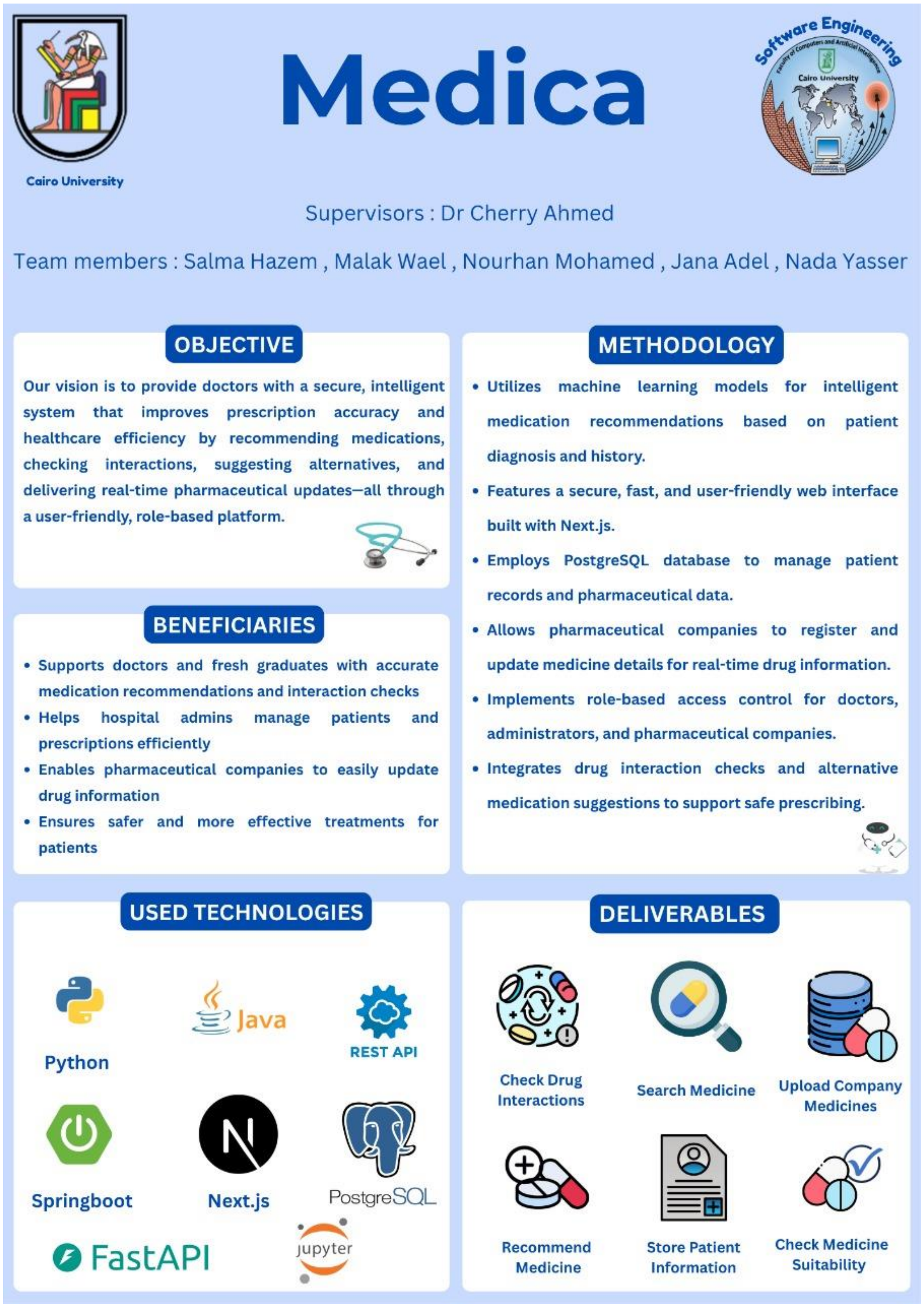
[1] https://www.kaggle.com/datasets/shudhanshusingh/250k-medicines-usage-side-effects-and-substitutes

[2] https://www.kaggle.com/datasets/mghobashy/drug-drug-interactions

[3] https://drive.google.com/file/d/1QAK5QrVUipbDUbpeIVS_gzuf6gz2iicG/view?usp=sharing

[4]https://drive.google.com/file/d/15OPC2TilMEYo5W8n9BCqZCPblWuvHVuR/view?usp=sharing

Poster



Cairo University

Medica

Software Engineering

Supervisors : Dr Cherry Ahmed

Team members : Salma Hazem , Malak Wael , Nourhan Mohamed , Jana Adel , Nada Yasser

OBJECTIVE

Our vision is to provide doctors with a secure, intelligent system that improves prescription accuracy and healthcare efficiency by recommending medications, checking interactions, suggesting alternatives, and delivering real-time pharmaceutical updates—all through a user-friendly, role-based platform.

BENEFICIARIES

- Supports doctors and fresh graduates with accurate medication recommendations and interaction checks
- Helps hospital admins manage patients and prescriptions efficiently
- Enables pharmaceutical companies to easily update drug information
- Ensures safer and more effective treatments for patients

METHODOLOGY

- Utilizes machine learning models for intelligent medication recommendations based on patient diagnosis and history.
- Features a secure, fast, and user-friendly web interface built with Next.js.
- Employs PostgreSQL database to manage patient records and pharmaceutical data.
- Allows pharmaceutical companies to register and update medicine details for real-time drug information.
- Implements role-based access control for doctors, administrators, and pharmaceutical companies.
- Integrates drug interaction checks and alternative medication suggestions to support safe prescribing.

USED TECHNOLOGIES

Python

Java

REST API

Springboot

Next.js

PostgreSQL

FastAPI

jupyter

DELIVERABLES

Check Drug Interactions

Search Medicine

Upload Company Medicines

Recommend Medicine

Store Patient Information

Check Medicine Suitability