esprit
Se former autrement

# 2024 - 2025
# GRADUATION PROJECT

## NATIONAL ENGINEERING DEGREE

## SPECIALTY : INFORMATION TECHNOLOGY

# Smart e-health platform for patient guidance and support


**By:** *Yosr Ben Nagra*

**Academic supervisor:** *Ms. Hiba Lahmer*

**Corporate Internship Supervisor:** *Ms. Imen Chikha*


itSERV

uni Tunis

cdio

European Accreditation of Engineering Programmes EUR-ACE®

Délivrée par: Cti

CONFÉRENCE DES GRANDES ÉCOLES
EXCELLENCE FOR A COMPLEX WORLD

I validate the submission of the student's report:

- Name & Surname : Yosr Ben nagra

Business site Supervisor

- Name & Surname : Ms. Imen Chikha

**Stamp & Signature**

STE IT SERV
5 Impasse Chahrazed 1104 Manzah 4
M.F: 1049342/P/A/M/000
Tél : 71 230 206 -71 233 226
Fax : 71 230 304

esprit
Se former autrement

J'autorise l'étudiant à déposer son rapport de stage pour l'obtention du diplôme

Hiba LAHMAR

# Dedication

This report is dedicated to all those who have supported, guided, and inspired me throughout my academic and personal journey.

**To my family:**
Your unwavering love, encouragement, and belief in me have been the foundation of my strength. The sacrifices you've made for my education and well-being are deeply appreciated and never forgotten. Every achievement in this report is a reflection of your enduring support.

**To my friends:**
Your presence, encouragement, and laughter have made this journey brighter. Thank you for standing by me in both moments of challenge and celebration.

**To my professors and mentors:**
Your guidance, dedication, and insight have played a crucial role in shaping my academic and professional path. I am sincerely grateful for the knowledge and inspiration you've provided.

**To everyone who contributed to this journey:**
Through words of encouragement, acts of kindness, or shared experiences, your support has left a meaningful mark. This accomplishment is the result of a shared path.

**To my dear friends Mehdi, Jallouli, Kamel, Bayoudh, and Nejd:**
Thank you for your genuine friendship and unwavering support. Your advice and presence have been a constant source of motivation, making this journey truly special.

**To my partner:**
Thank you for your encouragement, patience, and emotional support. Your quiet strength brought comfort during stressful times and joy in uncertain moments. I'm deeply grateful to have shared this chapter of my life with you.

# Acknowledgements

I would like to express my sincere gratitude to all those who made this final-year project possible.

First, my deepest thanks go to my supervisor, **Ms. Hiba Lahmar**, for her guidance, valuable insights, and ongoing support throughout this journey. Her expertise played a key role in shaping this work.

I also extend my appreciation to **ESPRIT** for providing the academic environment and resources needed to complete this project.

Special thanks to my internship host, **ITserv**, for the opportunity to grow in a professional setting, and to **Ms. Imen Chikha** for her mentorship and trust during the internship.

On a personal note, I am truly grateful to my family. To my parents, **Ammar and Saïda**, your constant support and sacrifices have been my greatest strength. To my sister, **Naoures**, and my brother, **Oussama**, thank you for always believing in me.

To everyone who supported me thank you. This achievement is ours.

# Contents

# List of Figures

# List of Tables

# General Introduction

Artificial Intelligence is transforming healthcare by enabling accessible, personalized medical guidance tools. This final year project addresses the growing need for intelligent digital health solutions by developing a smart web application that allows users to describe symptoms and receive preliminary health guidance.

The proposed system combines symptom analysis with community features, including a forum, medical articles from certified professionals, and an administrative dashboard for content management. While not replacing medical professionals, it serves as a first-step tool for understanding health conditions and determining appropriate actions.

This project tackles the lack of accessible, personalized digital tools that interpret symptoms in real-time without overwhelming users with technical jargon. The solution employs Retrieval-Augmented Generation (RAG), combining language models with real-time information retrieval through PubMed API integration to provide scientifically-backed health insights.

This report is organized as follows:

- **Chapter 1** presents the host organization, the adopted work methodology (Kanban, Git), and the overall context of the project.
- **Chapter 2** introduces the system's architecture, functional and non-functional requirements, and the tools and technologies employed.
- **Chapter 3** describes the AI component, including model fine-tuning, the implementation of the RAG approach, and the integration of the PubMed API.
- **Chapter 4** details the development of the application and its user interface components.
- **Chapter 5** discusses the DevOps practices implemented, such as CI/CD pipelines, monitoring, and security measures.

CHAPTER 1

# General Context

## Contents



This chapter establishes the foundational context of the AI-Based Medical Assistance Platform project. It begins by introducing ITSERV, the hosting company that provides the technical expertise and infrastructure necessary for the project's development. The chapter then examines the broader project context, highlighting the growing market need for advanced healthcare solutions and the limitations of existing symptom checkers that this initiative seeks to overcome. Finally, it presents the development methodology adopted to ensure efficient project management, collaborative workflows, and rigorous quality assurance throughout the implementation phases.

The healthcare industry currently stands at a pivotal intersection between traditional clinical practice and emerging digital technologies. As healthcare systems worldwide accelerate their digital transformation, there is an increasing demand for intelligent platforms capable of enhancing early symptom assessment, improving health literacy, and guiding patients toward appropriate care pathways. This chapter therefore lays the groundwork for understanding the industrial, technological, and methodological environment in which the project is situated.

## 1.1 Hosting Company

The hosting company for this project is ITSERV, a leading Digital Services Company (ESN) established in 2008 with operations in Tunis, Sfax, and Paris. ITSERV specializes in high-value digital transformation projects, primarily serving the telecommunications, public services, and healthcare sectors. The company is also engaged in the development of collaborative electronic platforms delivered in SaaS mode. Backed by specialized competence centers, ITSERV leverages the latest technologies and industry best practices to deliver innovative, scalable, and efficient digital solutions tailored to the evolving needs of its clients.

### 1.1.1 General Overview

ITSERV is a well-established digital services company with a strong presence in Tunis, Sfax, and Paris. Founded in 2008, the company has earned a solid reputation as a trusted partner in delivering high-impact digital transformation solutions. Guided by a commitment to quality and innovation, ITSERV continuously integrates emerging technologies to address the evolving needs of its diverse clientele.

Operating across several high-value sectors, such as telecommunications, public services, and healthcare, ITSERV positions itself as a strategic partner for organizations seeking to modernize and optimize their operations. In addition, the company has developed expertise in the creation of collaborative electronic platforms in SaaS mode, providing scalable, cloud-based solutions that comply with the latest industry standards and best practices.

ITSERV's multidisciplinary teams bring together technical excellence, business insight, and agile methodologies to design and implement tailor-made digital solutions. This client-centric approach, coupled with the company's emphasis on innovation and efficiency, has enabled ITSERV to stand out in an increasingly competitive market. As a result, ITSERV has become a key player in the digital services ecosystem, not only in Tunisia but also at the international level.

Figure 1.1: ItServ Logo

### 1.1.2 Areas of Expertise

ITSERV Tunisia's core strength lies in delivering high-value digital transformation projects across a range of strategic sectors. The company combines technical excellence with deep domain knowledge to develop scalable, tailored solutions. Key areas of expertise include:

- **Web and Mobile Application Development:** ITSERV develops robust and responsive web and mobile solutions that align with modern usability standards and technological trends. Their solutions are engineered for performance, security, and scalability, ensuring a seamless experience for end-users.
- **Digital Transformation for Key Sectors:** With proven experience in the telecommunications, public services, and healthcare industries, ITSERV delivers customized platforms and IT systems that support mission-critical operations and digital modernization initiatives.
- **Collaborative SaaS Platforms:** ITSERV designs and implements collaborative electronic platforms using a Software-as-a-Service (SaaS) model. These solutions enable efficient resource sharing, communication, and workflow management across distributed teams and organizations.
- **Cloud Integration and Infrastructure:** The company integrates cloud-based technologies to enhance system availability, scalability, and security. Their infrastructure services support businesses in migrating and managing their operations in cloud environments.
- **Agile Project Delivery and Industry Best Practices:** ITSERV follows agile methodologies and adheres to recognized software engineering standards. Their teams are organized into specialized competence centers to ensure expert-level delivery across all phases of development.

### 1.1.3 Current Challenges

Despite ITSERV strong technical foundation and diversified portfolio, the company faces several strategic and operational challenges in maintaining its competitive edge and sustaining long-term growth:

- **Rapid Technological Evolution:** The pace at which new technologies and frameworks emerge presents a continuous challenge. Staying up-to-date with modern stacks, while ensuring compatibility and stability for existing projects requires significant investment in training and adaptation.
- **Healthcare Sector Complexity:** Delivering digital solutions in the healthcare domain involves navigating complex regulatory frameworks, data privacy requirements, and the integration of AI in sensitive diagnostic contexts. These challenges demand high compliance standards and robust security protocols.
- **Talent Acquisition and Retention:** Like many technology-driven organizations, ITSERV faces difficulties in attracting and retaining highly skilled professionals, especially in emerging areas such as AI, cloud computing, and DevOps. Maintaining knowledge continuity across projects is essential to ensure delivery quality.
- **Cross-Sector Customization:** As ITSERV serves multiple industries, tailoring solutions to the unique demands of telecom, public administration, and healthcare clients requires deep domain knowledge and adaptive design strategies.
- **Scalability of Internal Processes:** With the company's growth and increased project load, scaling internal operations, including DevOps pipelines, project management workflows, and quality assurance practices, is crucial to maintain performance and efficiency.

## 1.2 Project Context

This section outlines the foundational aspects of the project. It begins by defining the project's scope and identifying the core problem it aims to address. A review of current approaches in symptom checking is then provided, followed by a critique of existing tools in the domain. Finally, the proposed solution is introduced, highlighting its innovative elements and the expected benefits it brings to the healthcare ecosystem.

### 1.2.1 Project Scope

This project was carried out as part of my final-year engineering internship, a mandatory requirement for obtaining the engineering degree from the Private Higher School of Engineering and Technology (ESPRIT). Spanning six months, the internship was hosted by ITSERV, a digital services company.

During this period, I independently developed a proof-of-concept (POC). The primary objective was to build a web-based solution that enables users to check their symptoms and receive preliminary diagnostic suggestions powered by artificial intelligence.

My primary responsibilities included designing and implementing RESTful API endpoints using Flask, managing a hybrid database system (PostgreSQL for structured data and MongoDB for flexible medical records), and deploying the AI-based symptom analysis engine. Specifically, I worked on the fine-tuning of a pre-trained **T5-base** model using curated medical datasets, enabling the system to process symptom descriptions and generate relevant condition predictions.

Additional contributions included integrating JWT-based authentication, implementing CI/CD automation using Jenkins, setting up code quality analysis with SonarQube, and configuring monitoring dashboards with Prometheus and Grafana. The project adopted a Kanban-based development approach, managed through Jira, to organize tasks and ensure continuous, agile delivery cycles.

The final product is a secure, scalable, and intelligent healthcare assistant designed for future expansion, with the potential to support telemedicine features, mobile applications, and broader clinical integration.

### 1.2.2 Problem Statement

In an era where digital tools are becoming integral to healthcare delivery, patients frequently encounter difficulties in understanding their symptoms and determining the appropriate next steps. While several online platforms offer symptom-checking capabilities, many fall short in delivering personalized, context-aware, and reliable health insights, particularly those powered by AI technologies.

As part of my final-year engineering internship at ITSERV, I was tasked with the development of a **Proof of Concept (POC)** for a smart e-health platform. The goal of this POC was to explore the feasibility and potential of combining artificial intelligence with interactive web technologies to support users in the early stages of medical decision-making.

#### Key Limitations in Current Solutions

The project was designed to address several critical gaps identified in existing healthcare platforms:

- **Symptom Interpretation Gap:** Existing tools often provide generic or static feedback, lacking the nuance and context required for reliable symptom analysis tailored to individual users.
- **Limited Use of AI:** Many platforms do not integrate fine-tuned language models trained on medical data, which limits their diagnostic intelligence and relevance.

- **Absence of Community and Educational Features:** Users rarely have access to peer forums, validated health articles, or expert-reviewed content, reducing trust and engagement.
- **Lack of Administrative Tools:** Managing medical content, moderating discussions, and maintaining data quality is difficult without an administrative dashboard.

**Central Challenge:** The design and implementation of a secure, intelligent, and administrable e-health platform that enables users to better understand their symptoms, receive AI-powered diagnostic suggestions, and access medically validated content, all within the scope of a scalable Proof of Concept.

### 1.2.3 Analysis of Current Approaches in Symptom Checking

#### Rule-Based Systems

In traditional rule-based systems, domain experts encode medical knowledge as if-then rules. While these systems are interpretable and deterministic, they struggle with ambiguity, synonymy, and the combinatorial explosion of rules needed to cover real-world variability. Maintenance becomes costly as medical knowledge evolves, and coverage gaps can lead to brittle behavior compared to data-driven approaches.

#### AI-Powered Symptom Checkers

Artificial Intelligence (AI) is reshaping digital health, offering dynamic ways to interpret symptoms. However, most mainstream platforms remain conservative in adopting full AI-based diagnostic features, particularly those intended for public-facing applications.

**WebMD:** WebMD [1] recently updated its symptom checker disclaimer to note that the platform *"may leverage certain generative artificial intelligence tools to generate results"*. However, the tool continues to emphasize that it is for informational purposes only and not intended for diagnosis. The system still relies on a structured symptom input flow via an interactive body map and produces unranked lists of possible conditions.

While WebMD's mention of generative AI suggests limited integration, possibly in how results are phrased, it does not appear to use AI for condition inference, triage scoring, or adaptive questioning. There is no evidence of Natural Language Processing (NLP) handling free-text input or Machine Learning (ML) powering diagnosis generation.

**Mayo Clinic:** The Mayo Clinic [2] Symptom Checker adopts an even more conservative design. It does not incorporate AI technologies and operates through fixed symptom selection from predefined lists. Its logic appears rule-based and static, offering general health information linked to medically reviewed content. There is no support for NLP, no personalization based on user history, and no signs of data-driven refinement.

### 1.2.4 Critique of Existing Tools

#### WebMD Symptom Checker

The WebMD Symptom Checker [1] is a widely used consumer-facing tool that forms part of the larger WebMD digital health ecosystem. It combines health information with interactive elements aimed at guiding users toward possible conditions based on symptom self-reporting.

**Overview:** Geared toward a broad and non-specialist audience, the tool is visually engaging and emphasizes ease of use. However, its clinical transparency and diagnostic rigor remain limited.

**Input Process:** Users interact with a digital human figure (avatar) to identify the location of symptoms. The system then prompts symptom selection within the chosen body region and allows qualifiers such as intensity or duration. Although visually intuitive, this interface lacks anatomical depth and does not support cross-regional or systemic symptom interactions. Free-text input is not supported, restricting flexibility in symptom expression.

**Output Format:** The system generates a list of potential conditions, each linked to a WebMD article. There is no clear explanation of the underlying reasoning, no presentation of diagnostic likelihood, and no urgency assessment. Additionally, the presence of promotional content raises concerns regarding content neutrality.

**Strengths:**

- Visually intuitive interface enhances user engagement.
- Quick and accessible for lay users with no clinical background.
- Integrated with a vast library of health content and articles.

**Limitations:**

- No transparency in how suggestions are derived or prioritized.
- Lacks personalization or AI-based refinement of inputs.
- Cannot capture interactions between multiple symptoms across systems.
- Advertising elements may compromise the user's perception of objectivity.

The WebMD Symptom Checker provides a highly accessible user experience but sacrifices clinical robustness in the process. Its interface is appealing and functional at the surface level but lacks depth, precision, and explainability, key qualities in credible digital health tools.

### Mayo Clinic Symptom Checker

The Mayo Clinic Symptom Checker [2] is part of the broader digital health platform offered by Mayo Clinic, a globally recognized institution known for its medically reviewed and evidence-based content. The tool is designed to offer users preliminary guidance based on self-reported symptoms, primarily as an informational resource.

**Overview:** The platform is web-based and targets the general public. Its interface prioritizes clarity and simplicity, directing users toward reliable medical topics rather than attempting diagnostic precision.

**Input Process:** Users begin by selecting demographic parameters such as age and gender, then choose from a categorized list of symptom groups. The absence of support for free-text input or interactive body mapping limits flexibility. The process is deterministic and menu-driven, which can streamline usage but also frustrate users with less common or multi-system symptoms.

**Output Format:** After symptom selection, the tool returns a list of related medical conditions or topics, each linked to detailed Mayo Clinic articles. However, the tool does not indicate likelihood rankings, confidence levels, or urgency suggestions. The lack of triage guidance diminishes its usefulness in decision-making under uncertainty.

**Strengths:**

- Developed by a reputable institution with strong medical credibility.

- Intuitive, minimal interface suitable for users with limited technical skills.
- All content is clinically vetted and frequently updated.

**Limitations:**

- Does not employ adaptive questioning or AI for personalized suggestions.
- No support for interactive symptom localization or natural language input.
- Offers no prioritization, confidence scoring, or urgency indicators.
- Lacks integration with downstream healthcare services (e.g., doctor matching, chatbots).

While the Mayo Clinic Symptom Checker delivers trustworthy information in a clear format, it lacks interactivity, personalization, and intelligent response mechanisms. It performs well as a static reference tool but is limited in its capacity to support real-time, user-specific health navigation.

### 1.2.5 Proposed Solution

#### System Overview

The proposed solution is an innovative healthcare platform designed to empower users with informed insights about their health conditions. Positioned as a preliminary step in the healthcare journey, this platform serves as a bridge between self-assessment and professional medical consultation. It addresses the critical need for accessible, accurate, and user-friendly health information in an era where digital solutions are increasingly integral to healthcare delivery.

The platform has been architected with user experience at its core, employing intuitive interfaces that accommodate diverse user capabilities and preferences. Its design philosophy emphasizes accessibility, ensuring that users can easily navigate the system regardless of their technical proficiency or health literacy. This approach democratizes access to preliminary health assessments, potentially reducing barriers to healthcare engagement, particularly for underserved populations.

Beyond its primary function as a symptom checker, the platform serves as a comprehensive health resource hub, integrating community support, educational content, and direct pathways to professional healthcare when needed. This holistic approach recognizes that effective healthcare extends beyond mere diagnosis to encompass education, support, and appropriate intervention planning.

### Key Functionalities

The platform introduces a comprehensive suite of features designed to enhance user engagement while delivering accurate and personalized health information:

Table 1.1: Core Platform Functionalities

| **Functionality** | **Description** |
|---|---|
| Interactive Body Map | Anatomically accurate, interactive body map allowing users to precisely locate symptoms with zoom functionality and multiple viewing angles (anterior/posterior) |
| Dual Input Methods | Flexible symptom reporting through both visual body map selection and traditional text-based entry |
| Hybrid Intelligence System | Combined curated medical database with Retrieval-Augmented Generation (RAG) technology for enhanced diagnostic accuracy |
| PubMed Integration | Automatic interfacing with PubMed API for accessing peer-reviewed medical literature when facing complex or novel symptom patterns |
| Confidence Indicators | Transparent presentation of diagnostic possibilities with calculated accuracy metrics for informed decision-making |
| Community Forum | Structured discussion space with nested reply functionality for user-to-user and user-to-professional interactions |
| Medical Insights | Dedicated section for verified healthcare professionals to publish educational health articles |
| Administrative System | Comprehensive backend for curating medical information, managing body part-symptom relationships, and updating treatment recommendations |

### User-Centric Interface Features

The platform's interactive body map represents a significant advancement in symptom reporting accuracy. Unlike conventional text-based systems, it enables users to:

- Precisely indicate symptom locations
- Zoom into specific anatomical regions
- Toggle between anterior and posterior views
- Visualize the relationship between symptoms and body regions
- Overcome medical terminology barriers
- Improve symptom communication accuracy
- Engage with the platform intuitively
- Experience reduced diagnostic frustration

### Advanced Diagnostic Capabilities

The platform's diagnostic engine represents a hybrid approach that balances the strengths of structured medical databases with the adaptability of AI technologies:

- **Primary Diagnostic System:** Utilizes a curated database of diseases, symptoms, and their relationships for high-confidence matching of common presentations.
- **RAG Enhancement:** Implements Retrieval-Augmented Generation to improve contextual understanding of symptom descriptions and provide more nuanced diagnostic possibilities.
- **PubMed Fallback Mechanism:** Automatically queries the PubMed API when encountering symptom patterns that extend beyond the platform's internal knowledge base.
- **Confidence Transparency:** Presents each potential diagnosis with a calculated accuracy indicator, empowering users with an understanding of diagnostic certainty.

### Community and Educational Components

Beyond symptom checking, the platform serves as a comprehensive health resource through:

- **Community Health Forum:** Structured discussion space with nested reply functionality, enabling multi-level conversations between users and healthcare professionals.
- **Medical Insights Section:** Curated content from verified healthcare providers, enhancing health literacy on relevant medical topics.

### AI and Hybrid Database Architecture

At the heart of the platform's capability is an intelligent system architecture designed to balance precision with adaptability in healthcare information delivery:

**Advanced Intelligence Framework** The platform employs a sophisticated dual-approach intelligence system that combines established medical knowledge with adaptive learning capabilities:

- **Core Knowledge Base:** A foundation of verified medical information that ensures reliable handling of well-documented conditions

- **RAG Technology Integration:** Enhancement of the system's ability to provide contextually relevant health information by dynamically retrieving and synthesizing medical knowledge

**Complementary Database Architecture** The platform employs a purposefully designed hybrid database architecture that separates medical knowledge management from user-driven interactions and content. This division improves both scalability and data access patterns.

Table 1.2: Hybrid Database Structure

| Characteristic | **Medical Knowledge Base (MongoDB)** | **User & Content Database (PostgreSQL)** |
|---|---|---|
| Primary Function | Stores structured medical information used for symptom checking | Manages user activity, community interactions, and platform content |
| Data Types | Diseases, symptoms, body regions, treatments, specialist mappings | Users, forum discussions, articles, user symptom check history |
| Strengths | Schema flexibility for evolving medical data, optimized for document queries | Relational consistency, transaction-safe interactions, advanced query support |
| Use Cases | Symptom-to-condition mapping, treatment logic, AI-enhanced diagnostics | User authentication, personalized history, community features, doctor content |

**Knowledge Extension Mechanisms** The platform extends beyond its internal knowledge through two key pathways:

- **PubMed Integration:** Strategic connection to peer-reviewed medical literature, enabling access to current research when internal knowledge is insufficient
- **Administrative Curation:** Comprehensive backend tools allowing authorized personnel to update the medical database with new conditions, symptoms, and treatment approaches

This architecture enables a self-improving system that leverages the strengths of both structured medical knowledge and adaptive AI technologies. By combining curated medical data with flexible intelligence, the platform delivers increasingly accurate and context-aware diagnostic insights, while preserving the critical human oversight essential in healthcare applications.

The hybrid database model ensures efficient data storage, retrieval, and scalability by accommodating both structured and unstructured data formats. Integrating AI with this dual-database architecture allows the system to generate personalized, evidence-based health information , ultimately enhancing user trust, engagement, and diagnostic reliability.

**Comparative Analysis of Existing Symptom Checkers**

To contextualize the proposed platform, this section compares it against two widely used symptom checkers: Mayo Clinic and WebMD. The comparison focuses on input mechanisms, diagnostic logic, personalization, and system transparency. The aim is to identify current limitations and highlight the unique advantages introduced by the proposed solution.

### 1.2.6 Expected Benefits

- **Enhanced Diagnostic Accuracy:** Leveraging AI algorithms trained on extensive medical datasets can improve the precision of symptom analysis and disease prediction, leading to more accurate diagnoses.
- **Improved Data Management:** A hybrid database architecture combining relational and non-relational databases allows for efficient storage and retrieval of diverse healthcare data types, enhancing data accessibility and integrity .
- **Scalability and Flexibility:** The hybrid approach supports scalable data solutions, accommodating growing data volumes and varying data structures, which is essential for expanding healthcare services.
- **Resource Optimization:** Automating routine tasks and data analysis through AI reduces the burden on healthcare professionals, allowing them to focus on patient care and complex decision-making.
- **Enhanced Patient Engagement:** User-friendly interfaces and personalized health insights encourage active patient participation in their healthcare journey, promoting better health outcomes.

The implementation of an AI-driven healthcare platform with a hybrid database architecture is anticipated to yield several significant benefits:

## 1.3 Development Methodology

### 1.3.1 Agile Workflow Overview

The development process adhered to Agile principles, emphasizing iterative progress, collaboration, and adaptability. Agile methodologies facilitate the delivery of functional software in incremental cycles, allowing for continuous feedback and refinement. This approach is particularly beneficial in dynamic environments where requirements may evolve over time.

Key aspects of the Agile workflow include:

Table 1.3: Comparative Analysis of Symptom Checkers

| Feature | **Mayo Clinic [2]** | **WebMD [1]** | **Proposed Solution** |
|---|---|---|---|
| Symptom Input | Fixed list selection | Body map with predefined options | Body map plus free-text input |
| Anatomical Accuracy | Basic categorization | Moderate detail via body map | Precision with zoom and multi-view |
| AI Integration | None | Limited use of generative AI (not diagnostic) | Yes (NLP, ML, RAG) |
| Medical Sources | Internal articles | Internal content with advertisements | Internal database plus PubMed API |
| Output Clarity | Unranked list of conditions | Basic list without prioritization | Ranked results with confidence scores |
| Triage Guidance | Basic content-based recommendations | Basic content-based recommendations | Includes urgency advice, suggested treatments and doctors |
| Transparency | High content accuracy, low logic clarity | Low transparency | High transparency with source and score explanations |
| Community Features | None | None | Forums with professional participation |
| Administrative Tools | Not available | Not available | Comprehensive backend for content management |
| Educational Content | Static articles | General health articles | Curated articles by verified professionals |
| Role in Patient Journey | Informational reference | Symptom self-check tool | First diagnostic step before professional care |

- **Iterative Development:** Breaking down the project into manageable iterations to deliver functional components regularly.
- **Continuous Feedback:** Engaging stakeholders throughout the development process to gather feedback and make necessary adjustments.
- **Self-Discipline and Autonomy:** Maintaining focus, setting milestones, and managing priorities independently to ensure consistent progress.
- **Adaptability:** Allowing for changes in project scope and requirements, ensuring the final product aligns with user needs.

### 1.3.2 Kanban-Based Project Management

To manage the workflow effectively as a solo developer, the Kanban methodology was adopted. Kanban offers a visual and flexible framework for organizing tasks and optimizing personal productivity. Its core principles were adapted to suit individual development, ensuring clear visibility of progress and maintaining steady momentum.

- **Visualizing Work:** Using a Kanban board to track tasks and their statuses, providing a clear overview of ongoing and upcoming work.
- **Limiting Work in Progress (WIP):** Setting personal limits on simultaneous tasks to maintain focus and avoid context switching.
- **Managing Flow:** Monitoring the movement of tasks through defined stages (e.g., To Do, In Progress, Done) to identify blockers and maintain a smooth workflow.
- **Continuous Improvement:** Regularly reflecting on workflow patterns to refine task organization and improve efficiency.

This approach provided a structured way to manage priorities and adjust quickly to new insights or shifting objectives during the development process.

### 1.3.3 Task Organization and Tracking with Jira

Jira was utilized as the primary tool for task management and tracking [3]. Its features effectively supported the Agile and Kanban methodologies by enabling structured task planning and progress monitoring.

- **Customizable Workflows:** Allowing the creation of workflows tailored to the specific phases of the individual development process.

- **Issue Tracking:** Managing tasks, bugs, and feature enhancements efficiently, with clear categorization and status tracking.
- **Personal Task Organization:** Structuring work items using boards and swimlanes to maintain clarity and prioritize effectively.
- **Reporting and Analytics:** Generating visual summaries to monitor progress over time and support reflective evaluation.

The integration of Jira into the development process enhanced organization, visibility, and focus, contributing significantly to the successful execution of the project.

### 1.3.4 AI Model Development Using CRISP-DM

The development of the AI model adhered to the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, a structured approach widely recognized in data science projects. This framework comprises six iterative phases:

- **Business Understanding:** Defined the objectives of the AI model within the healthcare context, focusing on accurate symptom analysis and disease prediction.
- **Data Understanding:** Collected and explored relevant medical datasets to gain insights into data quality, completeness, and relevance.
- **Data Preparation:** Performed data cleaning, transformation, and integration to ensure the dataset was suitable for modeling. This included handling missing values, encoding categorical variables, and normalizing data.
- **Modeling:** Selected appropriate machine learning algorithms and fine-tuned a pre-existing model using the prepared dataset to enhance predictive accuracy.
- **Evaluation:** Assessed the model's performance using relevant metrics to ensure it met the defined business objectives and provided reliable predictions.
- **Deployment:** Integrated the validated model into the application's backend, enabling real-time symptom analysis for end-users.

This systematic approach facilitated a thorough and efficient development process, ensuring the AI model's effectiveness and reliability within the healthcare platform.

To conclude, this chapter has established the strategic and technical foundations of the project. It introduced the industrial context of ITSERV, outlined the motivations behind developing a next-generation AI-based medical platform, and critically evaluated the limitations

of existing symptom checkers. The proposed solution, supported by a hybrid system architecture and modern AI techniques, aims to bridge current gaps in personalization, community interaction, and medical data management.

The following chapter marks the transition from contextual analysis to technical specification. It defines the system's functional and non-functional requirements, identifies key actors, and presents both the global use case and class diagrams. Additionally, it describes the architecture, tools, technologies, and project management methodologies used to bring the solution to life. These specifications form the blueprint for the implementation phases that follow.

CHAPTER 2

# Project Foundation

## Contents



This chapter lays the groundwork for the technical realization of the project by presenting its foundational elements. It begins with a comprehensive specification of the system's requirements, including the identification of key actors, functional and non-functional requirements, a global class diagram, and visual modeling tools such as the global use case diagram and the product backlog. These components provide a structured understanding of the system's intended behavior and serve as a reference throughout the development process.

Following the requirements specification, the chapter explores the system's architecture from both logical and physical perspectives. This section illustrates how various components interact and integrate to form a cohesive and scalable system aligned with the project's goals.

The chapter concludes with a detailed overview of the tools and technologies used throughout the project. These include project management and communication platforms, development environments and programming technologies, version control systems, and essential APIs and modeling tools. The selection and integration of these tools were critical in ensuring an efficient, collaborative process.

## 2.1 Requirements Specification

### 2.1.1 Actors Identification

In the context of the AI-driven e-health platform developed during the internship at ITSERV, several key actors interact with the system. Identifying these actors is crucial for understanding system requirements, designing user interfaces, and ensuring seamless integration of functionalities. The platform also allows unauthenticated users to access the symptom checker, enabling quick health assessments without requiring registration or login.

- **Unauthenticated Users** *Role:* Visitors who access the platform without creating an account. *Responsibilities:*
  - Using the symptom checker to receive preliminary health assessments.
  - Browsing medical insights (articles), and community hub (forum) discussions.
- **Patients** *Role:* Primary users who create accounts to save their symptom analyses, engage with the platform's community features.
  *Responsibilities:*
  - Registering to create personal health profiles.
  - Using the symptom checker to receive preliminary health assessments and storing their analysis history.
  - Rating and providing feedback on medical articles within the Medical Insights section.
  - Participating in community hub discussions by asking questions, sharing experiences, and contributing to ongoing conversations.
- **Healthcare Professionals** *Role:* Medical experts who contribute clinical expertise, oversee content accuracy, and engage with users through educational and interactive features.
  *Responsibilities:*

  - Creating and validating medical articles within the Medical Insights section.
  - Using the symptom checker to support case analysis and storing relevant assessments.
  - Participating in community hub discussions by responding to patient queries and sharing medical guidance.

- **Administrators** *Role:* Platform managers responsible for overseeing system operations, user access, and the integrity of medical content and data workflows.
  *Responsibilities:*

  - Managing user accounts and permissions, including blocking, editing, or deleting users as necessary.
  - Monitoring platform performance, usage analytics, and ensuring operational stability.
  - Managing medical content, including the creation, review, and organization of articles and forum discussions.
  - Overseeing and curating the symptom checker database, including symptoms associated with different body regions on the anatomical map.
  - Adding and updating diseases within the AI model through the RAG (Retrieval-Augmented Generation) and embedding new medical data.

### 2.1.2 Functional Requirements

1. **Symptom Checker and AI-Powered Analysis**

   - Allow users to input symptoms via an intuitive interface or by interacting with an anatomical body map.
   - Support multi-step symptom selection to ensure detailed and accurate input.
   - Utilize an AI-powered model to analyze symptoms and generate potential health conditions.
   - Recommend relevant specialists based on analysis outcomes.
   - Display results with clear visualizations and confidence scores.
   - Enable users to store and track their symptom analysis history.

2. **Personalized Health Recommendations**

   - Provide tailored advice based on AI analysis of symptoms.
   - Recommend appropriate actions (self-care, medical consultation, emergency care).
   - It provides information on the appropriate medical specialist to consult.

3. **Authentication and Role-Based Access**

- Implement secure JWT-based authentication for session management.
- Support Google OAuth for user login.
- Enable token refresh mechanisms to maintain session continuity.
- Restrict access to routes and features based on user roles (Admin, Doctor, Patient).
- Validate roles at the API level using middleware for secure role-based access control.

4. **Medical Insights (Articles)**

   - Allow healthcare professionals to create, update, and delete validated medical articles.
   - Enable users to search and filter articles by keyword, category, or condition.
   - Allow patients to rate on articles to enhance feedback and engagement.

5. **Community Hub (Forum) Interactions**

   - Enable users to post questions and responses in a structured Q&A format.
   - Implement search and filtering capabilities for forum content.
   - Allow healthcare professionals to participate in discussions, offering expert insights.
   - Include moderation tools to ensure high-quality, respectful interactions.

6. **Administrative Controls**

   - Manage user accounts, permissions, and roles across the platform.
   - Add, update, or remove symptoms and diseases within the system.
   - Embed new medical knowledge into the AI model using the RAG (Retrieval-Augmented Generation).
   - Control the symptom checker database and the mapping of symptoms to specific body regions.
   - Oversee content moderation, including medical articles and community discussions.
   - Monitor platform usage statistics and generate analytical reports.

7. **Dashboard and UI Features**

   - Provide role-specific dashboards for Admins, Doctors, and Patients with relevant data and actions.
   - Support dynamic form layout with input validation.
   - Include table components with pagination, filtering, sorting.

### 2.1.3 Non-Functional Requirements

1. **Security and Privacy**
    - JWT and Google OAuth-based authentication are required; two-factor authentication (2FA) is recommended for administrative and professional accounts.
2. **Performance and Scalability**
    - General page loads and authentication should occur within 1 second.
    - The system must be designed with scalability in mind, using Docker-based containerization to support future horizontal scaling.
3. **AI Model Integration**
    - The primary AI model is hosted locally with a fallback to Hugging Face-hosted model for reliability.
    - Response time for AI analysis may reach up to 20 seconds in fallback scenarios, which is acceptable for PoC evaluation.
    - AI-generated health assessments must present confidence scores.
4. **Usability and Accessibility**
    - Interactive components such as the body map and symptom checker must be optimized for both mouse and touch inputs.
5. **Maintainability and Modularity**
    - The system shall follow a modular monolith structure to ensure maintainability while enabling rapid development during the PoC phase.
    - Clear separation of frontend and backend responsibilities must be maintained.
6. **Availability and Reliability**
    - Appropriate fallback mechanisms must be implemented for AI model failures and database downtime.
    - Robust logging and error handling must be in place to support system monitoring and recovery.

### 2.1.4 Global Class Diagram

#### System Architecture Overview

Our application employs a hybrid database architecture with PostgreSQL and MongoDB to optimize data management:

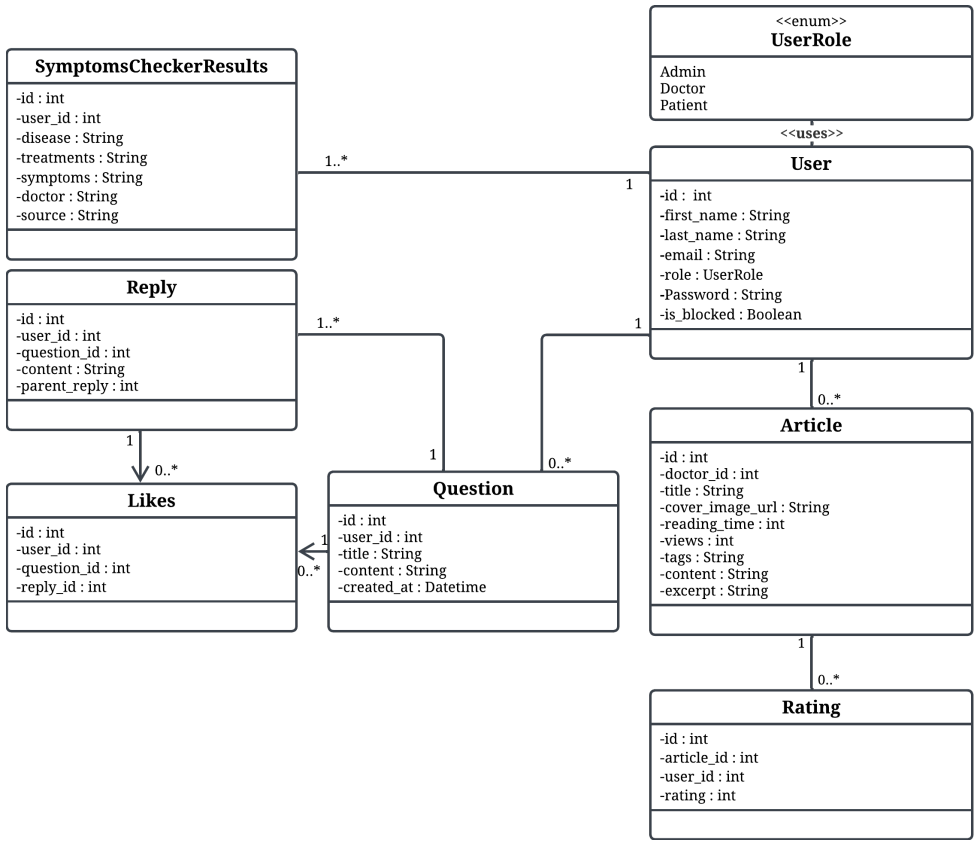


Figure 2.1: User Management Model (PostgreSQL)

**Relational Database (PostgreSQL):**

- **User:** Role-based account management (Admin, Doctor, Patient)
- **Community:** Forum functionality with Question and Reply entities
- **Medical Content:** Article and Rating management
- **SymptomsCheckerResults:** Diagnostic output storage

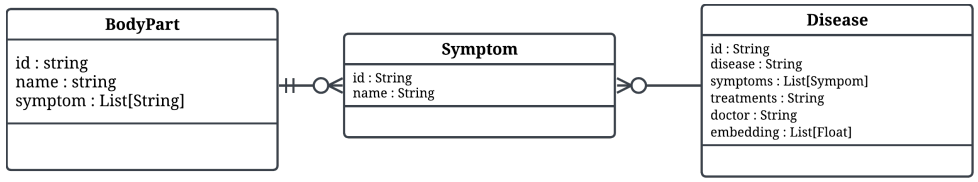


Figure 2.2: Medical Knowledge Model (MongoDB)

**Document Database (MongoDB):**

- **BodyPart:** Anatomical regions with linked symptoms

- **Symptom:** Clinical signs
- **Disease:** Medical conditions with related symptoms, treatments, and specialist doctor.

**Key Integration Benefits:**

- Structured user data in PostgreSQL
- Flexible medical knowledge in MongoDB
- Seamless cross-database integration
- Optimized query performance

### 2.1.5 Global Use Case Diagram

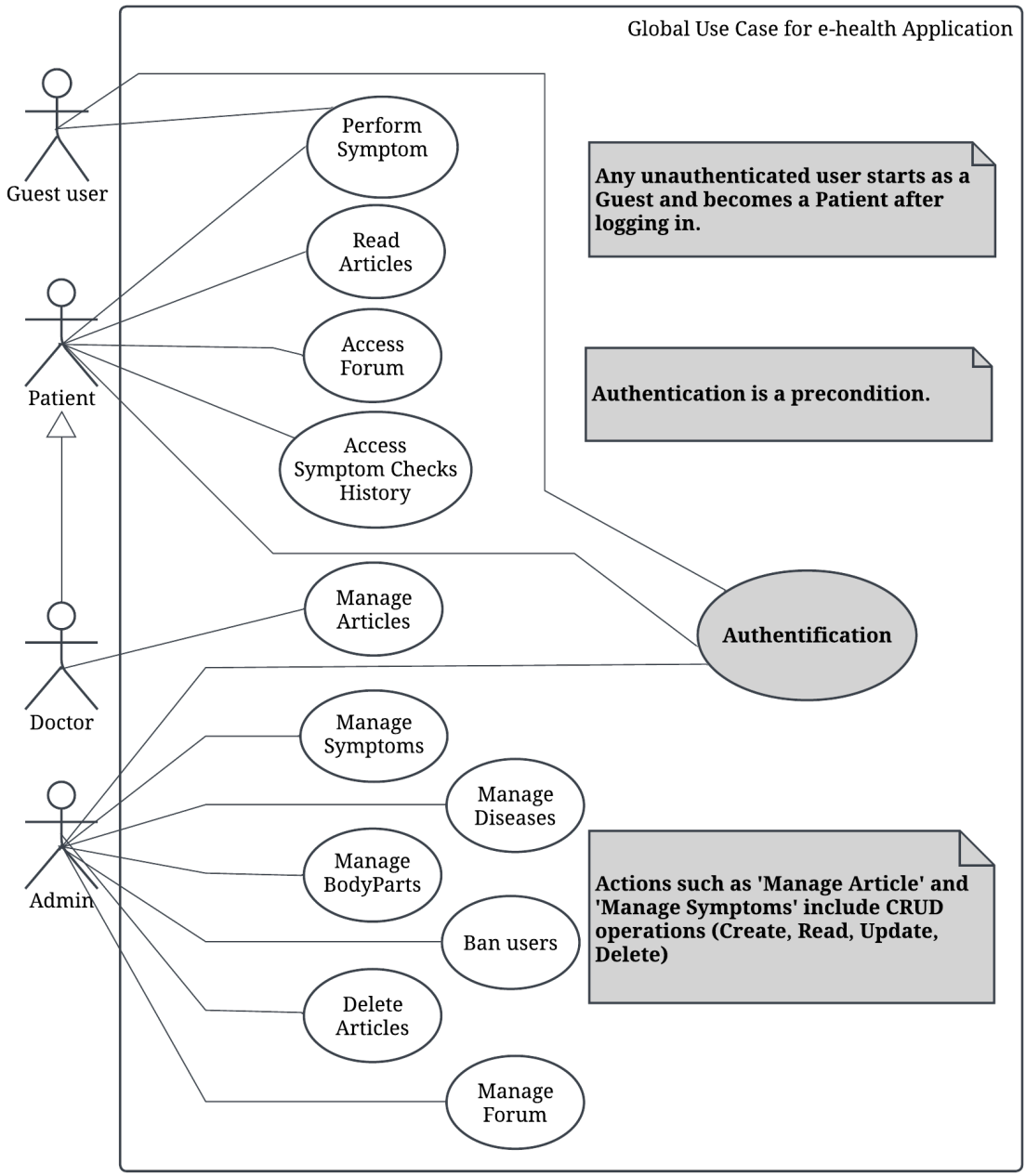


Figure 2.3: Global Use Case Diagram

**Key Actors:**

- **Guest User:** Unauthenticated access with limited functionality
- **Patient:** Authenticated user with full application access

- **Doctor:** Manage Articles
- **Administrator:** System management and moderation capabilities

**Patient Functionalities:**

- Symptom checking
- Medical article access
- Community forum participation
- Personal medical history management

**Administrative Capabilities:**

- Content management (articles, symptoms, diseases, body parts, doctors requests)
- User moderation and access control
- Forum oversight and administration

All patient, admin, doctors functionalities require authentication, while actions such ('Manage Articles', etc..) involve standard CRUD(CREATE, READ, UPDATE, DELETE) operations for system resources.

### 2.1.6 Product Backlog

The product backlog presents user stories organized by role and priority using the following columns:

- **ID:** Unique identifier for each user story for tracking and reference purposes
- **User Story:** Description in the format "As a [role], I want [feature] [reason]" where the reason part (e.g., "so that…", "for that…", "in order to…", etc.) is optional
- **Priority:** Implementation urgency ranked as Highest, High, Medium, or Low

| ID | User Story | Priority |
|---|---|---|
| **Patient Role Features** | | |
| 1 | As a user, I want to select symptoms using an interactive body map | Highest |
| 3 | As a user, I want to receive AI-powered analysis of my symptoms | Highest |
| 2 | As a user, I want tools to enhance the use of the interactive body map | High |
| 8 | As a user, I want to manually enter symptoms | High |
| 9 | As a user, I want to receive treatment recommendations | High |
| 10 | As a user, I want to be referred to appropriate medical specialists | High |
| 13 | As a user, I want to browse health-related questions | High |
| 14 | As a user, I want to ask health questions | High |
| 4 | As a user, I want to receive a tutorial guide for the symptom checker | Medium |
| 5 | As a user, I want to browse medical articles | Medium |
| 6 | As a user, I want to register for an account | Medium |
| 7 | As a user, I want to log in to my account | Medium |
| 11 | As a user, I want to view my symptom check history | Medium |
| 12 | As a user, I want to compare results from different sources | Medium |
| 15 | As a user, I want to search for specific health topics | Medium |
| 16 | As a user, I want to like/unlike questions | Medium |
| 17 | As a user, I want to reply to questions | Medium |
| 20 | As a user, I want to rate articles | Medium |
| 18 | As a user, I want to view my own questions | Low |
| **Doctor Role Features** | | |
| 21 | As a doctor, I want to create medical articles | Medium |
| 22 | As a doctor, I want to upload images for my articles | Medium |
| 23 | As a doctor, I want to edit my published articles | Medium |
| **Administrator Role Features** | | |
| 25 | As an admin, I want to manage the diseases database | High |
| 26 | As an admin, I want to block/unblock users | High |
| 27 | As an admin, I want to review and approve/reject doctor account requests | High |
| 28 | As an admin, I want to manage Articles / Questions | High |
| 24 | As an admin, I want to manage user accounts | Medium |

Table 2.1: Product Backlog: User Stories Organized by Role and Priority

**Note:** Within each role grouping, features are sorted by priority (Highest → High → Medium → Low) to facilitate implementation planning.

## 2.2 Architecture

A well-defined architecture is fundamental to the success of any project. It ensures seamless integration of system components while maintaining scalability, reliability, and efficiency.

This section presents the system architecture in two key aspects: **physical** and **logical**. The physical architecture describes how the system is hosted and deployed across infrastructure components, whereas the logical architecture focuses on the system's functional structure and the interactions between its components.

### 2.2.1 Logical Architecture

The logical architecture defines the interaction between system components to achieve the required functionality. It emphasizes a clear separation of concerns to support maintainability and scalability.

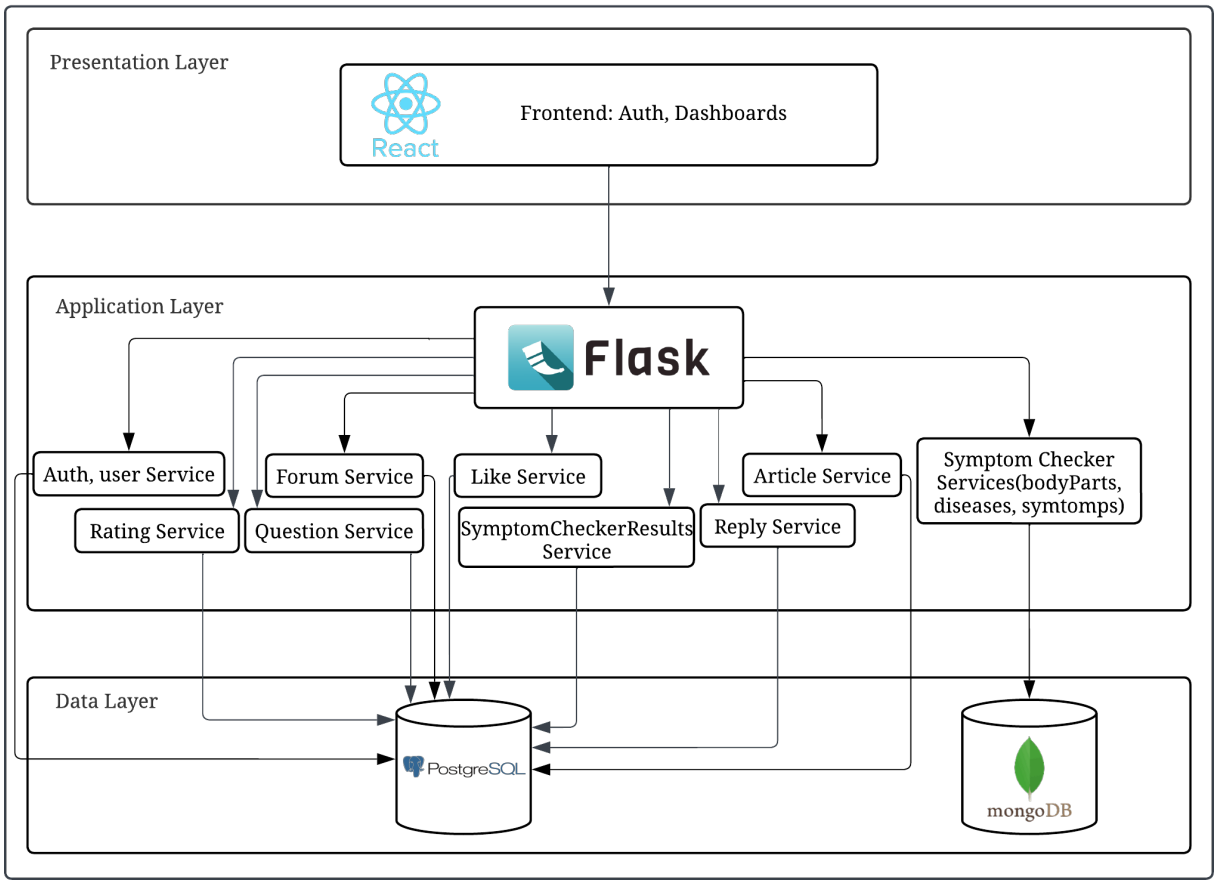


Figure 2.4: Logical Architecture

**Data Layer:** Handles data storage and access.

**Presentation Layer:** The user interface where both end-users and content managers interact.

**Application Layer:** Contains the core logic of the system, including content workflows, user authentication, and data validation, ensuring proper handling of user interactions and system rules.

### 2.2.2 Physical Architecture

The physical architecture outlines the system's deployment across infrastructure components. This project adopts a distributed architecture to enhance availability, performance, and security.

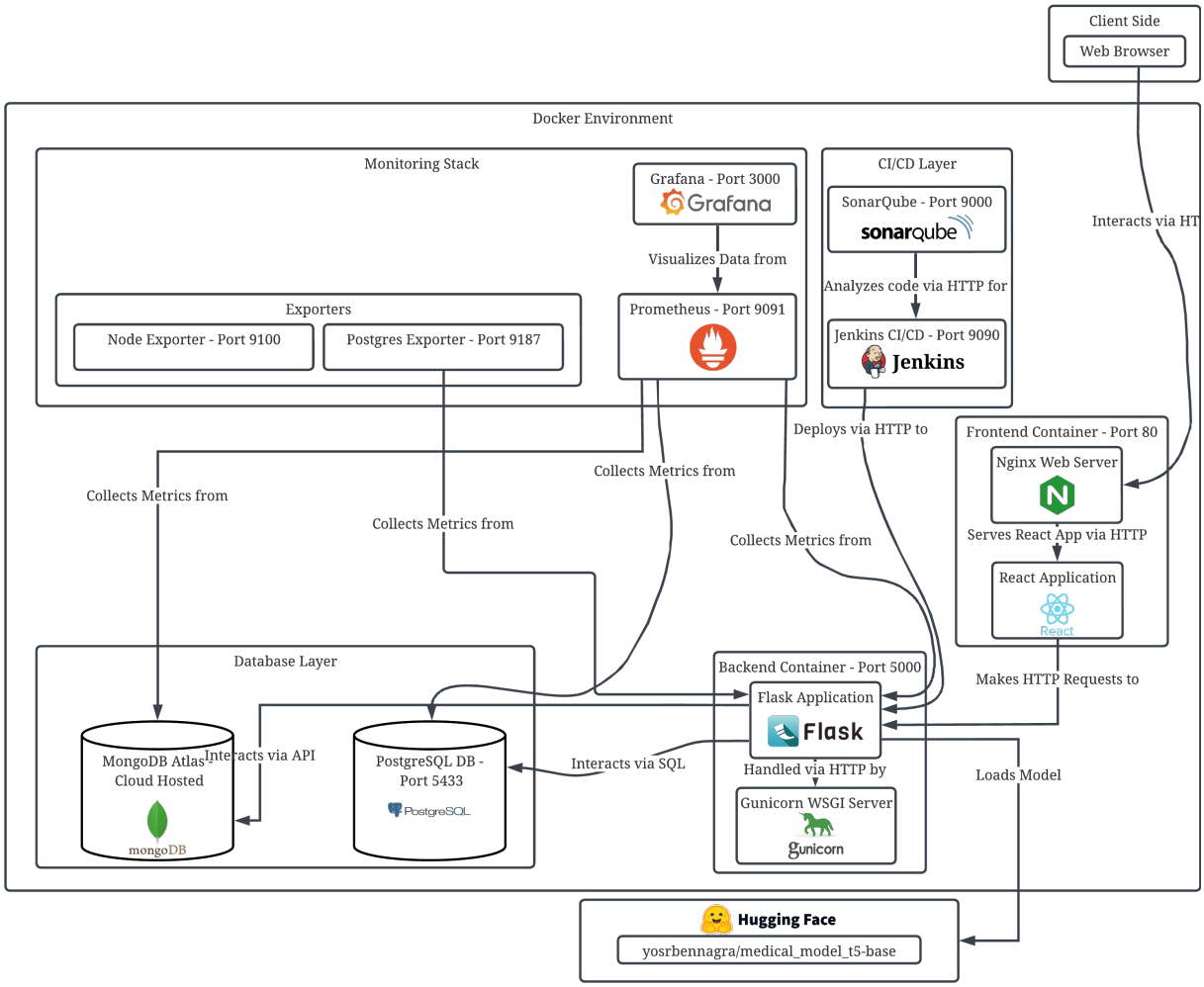


Figure 2.5: Physical Architecture

## 2.3 Tools and Technologies

This section presents the main tools and technologies used throughout the project. It explains their roles and how they contribute to achieving the project's objectives.

### 2.3.1 Project Management

- **Agile Methodology**
  - Employed Agile principles to facilitate iterative development, providing flexibility and encouraging continuous improvement.
  - Although sprints were not used, the iterative approach allowed for regular reviews and adjustments to the project direction.
  - Emphasized collaboration, adaptability, and frequent feedback to accommodate evolving project requirements.
- **Kanban Method**
  - Utilized the Kanban method to visualize workflow, manage task progression, and identify bottlenecks.
  - Tasks were categorized into columns such as *To Do*, *In Progress*, and *Done* to streamline task tracking and improve visibility.
  - Kanban enabled effective task prioritization and maintained a steady flow of work, without rigid time-boxed sprints.

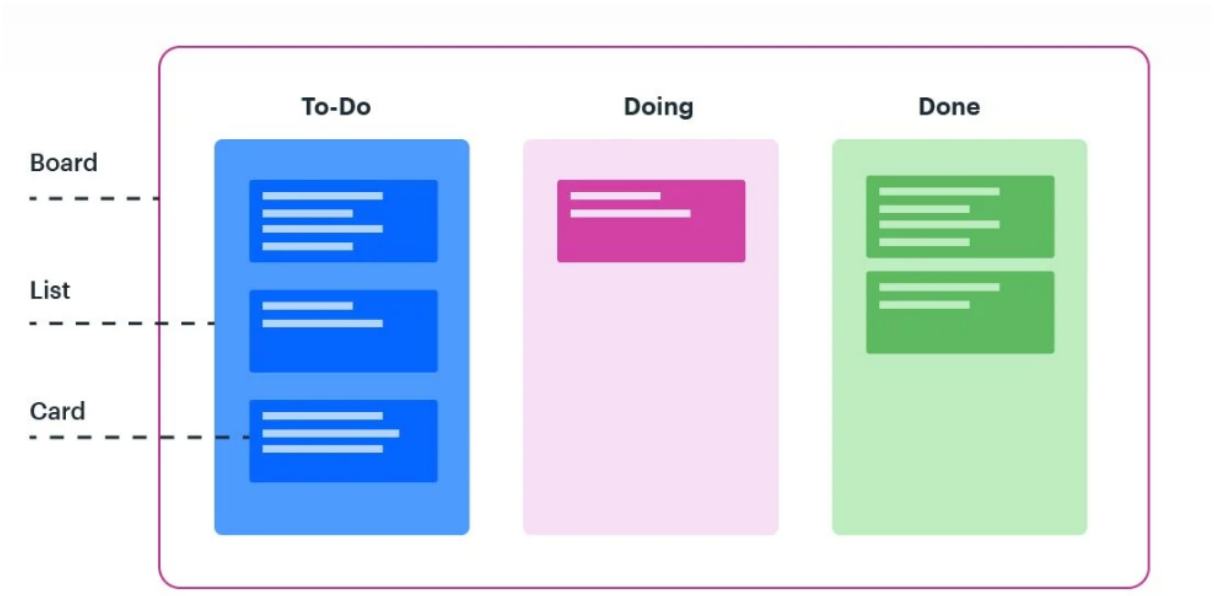


Figure 2.6: Kanban Board

While Scrum is a widely adopted Agile framework, it was not chosen for this project due to its reliance on time-boxed sprints and structured team ceremonies such as sprint planning and daily stand-ups. As this was an individual project, the Scrum framework would have added unnecessary complexity. Kanban was therefore preferred for its lightweight structure, continuous flow, and greater flexibility, making it a more suitable method for solo work and evolving task priorities.

**Value-focused comparison (Kanban vs. Scrum)**

**Overhead** Kanban minimizes ceremonies and documentation; Scrum requires sprint planning, reviews, and stand-ups. For a solo project, Kanban cuts process time and keeps focus on delivery.

**Flow & WIP** Kanban's continuous flow with WIP limits reduces context switching and improves cycle time; Scrum batches work into sprints which can introduce idle time or rollover.

**Adaptability** Kanban allows on-demand reprioritization as scope evolves; Scrum changes mid-sprint are discouraged and typically deferred to the next sprint.

**Visibility** A simple Kanban board provides clear, immediate status (To Do/In Progress/-Done) without extra sprint artifacts.

**Decision** Kanban maximizes throughput and momentum for a single contributor with research-heavy, evolving tasks—delivering more value sooner in this context.

- **Work-In-Progress (WIP) Limits**
  - *What*: An explicit cap on how many items can be in an active column (for example, In Progress) at the same time.
  - *Why*: Reduces context switching, exposes bottlenecks, and shortens cycle time (Little's Law: WIP $\approx$ throughput $\times$ cycle time).
  - *Policy*: When a column reaches its WIP limit, finish or unblock existing work before starting something new.
  - *Applied here*: Solo defaults used were In Progress = 1–2 and Review or Testing = 1 to keep flow steady and fast.
- **Jira for Task Management**
  - Implemented Jira as the primary tool for issue tracking and project management.
  - Created and managed user stories, tasks, and bugs within Jira to maintain a comprehensive and organized project backlog.
  - Leveraged Jira's reporting features to monitor progress, and facilitate informed decision-making.


Figure 2.7: Jira Logo

- **CRISP-DM Framework for AI Model Development**
  - Adopted the Cross-Industry Standard Process for Data Mining (CRISP-DM [4]) framework to guide the AI model development process.
  - Followed the six phases of CRISP-DM: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.
  - Ensured a structured approach to data analysis and model building, which enhanced the reliability and effectiveness of the AI component.

### 2.3.2 Communication

- **Microsoft Outlook**
  - Utilized for formal correspondence, including emails, meeting invitations, and calendar management.
  - Integrated seamlessly with other Microsoft Office applications, facilitating document sharing and collaboration.

Figure 2.8: Outlook Logo

- **Google Meet**
  - Employed for virtual meetings, stand-ups, and video conferencing.
  - Integrated with Google Calendar, allowing for easy scheduling and meeting reminders.
  - Provided features like screen sharing, real-time captions, and chat functionalities, enhancing meeting effectiveness.


Figure 2.9: Google Meet Logo

- **WhatsApp**
  - Used for quick, informal communication among team members.
  - Facilitated instant messaging, voice calls, and group chats, ensuring prompt information exchange.
  - Offered end-to-end encryption, maintaining the confidentiality of communications.

Figure 2.10: Whatsapp Logo

### 2.3.3 Development Environment

#### Code Editor and Version Control

- **Visual Studio Code (VSCode)**: Main code editor used for both frontend and backend development.


Figure 2.11: VSCode Logo

- **Version Control**: Git and GitHub were used for source control, collaboration, and code reviews. See Section 2.3.5 for workflow and tooling details.

#### Containerization and Deployment

- **Docker**: Used to containerize services, ensuring consistency across environments.

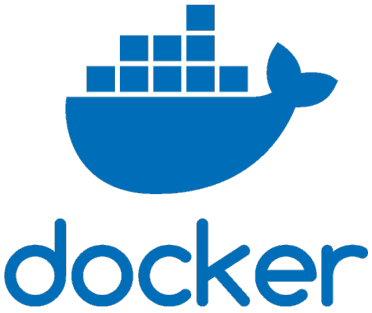


Figure 2.12: Docker Logo

- **Nginx**: Configured as a reverse proxy and static content server [5].

Figure 2.13: Nginx Logo

#### Static Analysis and Formatting

- **isort**: For automatically sorting Python imports.
- **flake8**: For Python code linting and enforcing PEP8 style compliance.

#### CI/CD and Code Quality

- **Jenkins**: Implemented for continuous integration and automated testing workflows [6].


Figure 2.14: Jenkins Logo

- **SonarQube**: For static code analysis and technical debt detection [7].


Figure 2.15: SonarQube Logo

### Monitoring and Observability

- **Grafana**: A visualization tool used to create interactive and real-time dashboards for monitoring system performance and displaying metrics [8].


Figure 2.16: Grafana Logo

- **Prometheus**: An open-source monitoring and alerting toolkit that collects and stores time-series data, used for tracking system performance and generating metrics for analysis [9].

Figure 2.17: Prometheus Logo

### Testing Tools

- **Vitest and Jest**: For unit testing frontend React components [10].
- **Pytest**: For backend testing in Python (Flask), with coverage reporting tools [11].

### Database Management

- **DBeaver**: A free, open-source universal database tool used for database management, SQL development, and data analysis. It supports all popular databases including PostgreSQL, MongoDB, and many others [12].


Figure 2.18: DBeaver Logo

### 2.3.4 Development Technologies

- **Frontend Technologies**
  - **React.js**: A JavaScript library for building user interfaces, particularly suited for single-page applications with reusable components [13].

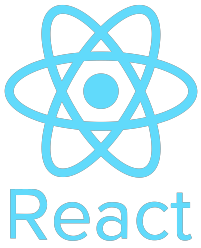


Figure 2.19: React Logo

  - **Vite**: A fast build tool and development server for modern web applications, providing hot module replacement (HMR) for rapid frontend development [14].

Figure 2.20: Vite Logo

  - **Redux Toolkit**: State management library to handle global state in React.


Figure 2.21: Redux Logo

  - **react-auth-kit**: Used to handle client-side authentication logic.

- **Backend Technologies**

  - **Python**: A high-level programming language used for backend development, known for its simplicity and versatility in handling various tasks, including data processing and logic implementation.


Figure 2.22: Python Logo

  - **Flask**: A lightweight Python web framework used to build RESTful APIs, enabling the development of scalable and modular backend services [15].


Figure 2.23: Flask Logo

  - **Node.js**: A JavaScript runtime environment used to execute JavaScript code server-side, enabling backend development with JavaScript.

Figure 2.24: Node.js Logo

  - **npm**: A package manager for Node.js, used to manage frontend and backend dependencies, as well as run scripts for build and development processes.

Figure 2.25: NPM logo

- **Databases and External Services**
  - PostgreSQL: Primary relational database.

Figure 2.26: Postgres Logo

  - MongoDB Atlas: Cloud-hosted NoSQL database for certain dynamic data structures.


Figure 2.27: MongoDB Logo

- Node Exporter and Postgres Exporter: Used to expose system and database metrics to Prometheus.

### 2.3.5 Version Control

**Git** served as the primary distributed version control system for tracking changes in the source code. It enabled multiple developers to work concurrently on different features without interfering with each other's work. Git facilitated branching and merging, allowing for isolated development and integration of new features, which streamlined the development process.


Figure 2.28: Git Logo

**GitHub** hosted the remote repositories, providing a centralized platform for code storage and collaboration. It supported pull requests, which enabled code reviews and discussions before merging changes into the main branch. Additionally, GitHub integrated seamlessly with other tools and services, enhancing the overall development workflow and facilitating efficient collaboration.


Figure 2.29: Github Logo

**Fork** was utilized as a graphical Git client to streamline repository management. It provided an intuitive interface for handling commits, branches, and merges, improving productivity and minimizing errors. Fork also assisted in visualizing the repository's history and structure, which made it easier to understand the project's evolution and to manage its development lifecycle effectively.

Figure 2.30: Fork Logo

### 2.3.6 Supporting Tools and APIs

**Hugging Face :** Hugging Face was used as the hosting platform for the fine-tuned language model developed during this project. This solution provides easy access to pre-trained models as well as public datasets, thus simplifying the fine-tuning process. The trained model was published on their infrastructure, allowing remote access via a REST API. This approach not only streamlined deployment but also ensured the scalability and reproducibility of the results. Using Hugging Face made it possible to centralize all AI-related resources of the project within a secure and collaborative environment.


Figure 2.31: Hugging Face Logo

**PubMed :** The PubMed API was integrated into this project to enable efficient searching of scientific medical articles related to the symptoms being analyzed [16]. Two main endpoints were used: `esearch.fcgi`, which allows keyword-based searches in the database, and `esummary.fcgi`, used to retrieve article summaries based on the obtained identifiers. This API delivers accurate results from peer-reviewed publications, thereby enhancing the reliability of the generated recommendations. *For example*, a query with the keyword "fever" returns a list of recent articles with their titles, identifiers, and summaries. This integration enriches the application with validated medical content while ensuring traceability of the provided information. The results obtained from this API are systematically used to supplement

the answers automatically generated by the model, providing an additional layer of relevance and medical justification.


Figure 2.32: PubMed

**Google Cloud Console :** The Google Cloud Console was used to configure and manage the authentication system based on Google accounts [17]. By enabling the OAuth 2.0 API and setting up the necessary credentials, the application was able to offer users a seamless and secure "Login with Google" experience. This integration not only simplified the sign-in process but also ensured compliance with modern security standards, such as token-based authentication and secure redirection. Using Google Cloud Console facilitated centralized management of OAuth scopes, redirect URIs making it an essential component of the authentication workflow.


Figure 2.33: Google Cloud Logo

### 2.3.7 Modeling and Design Tools

**Lucidchart** Lucidchart is a web-based tool with an intuitive drag-and-drop interface, ideal for creating UML and architecture diagrams. In this project, it enabled efficient, independent system design, helping ensure clarity and alignment with technical requirements.


Figure 2.34: Lucidchart Logo

**Adobe Illustrator** Adobe Illustrator was employed to create a detailed and interactive body map, allowing users to select specific body regions to indicate symptoms. This vector-based design facilitated precise delineation of anatomical areas and ensured scalability across different device resolutions. The body map serves as a central component of the symptom input interface, enhancing user engagement and accuracy in symptom reporting.


Figure 2.35: Abode Illustrator Logo

**Material-UI (MUI)** Material-UI (MUI) is a popular React component library implementing Google's Material Design principles. It was used to design and develop a modern, consistent, and responsive user interface across the e-health platform. Thanks to its pre-built, customizable components and powerful theming system, MUI significantly accelerated frontend development while ensuring a high standard of user experience.


Figure 2.36: Material UI Logo

This chapter has established a solid foundation for the project's technical development. By defining system requirements, architectural design, and tool choices, it provides a structured and informed path toward implementation. Visual modeling and a well-selected toolchain offer clear guidance for the development process and ensure alignment with project goals.

With the architecture and environment in place, the next step focuses on the project's core intelligence: designing, training, and deploying the AI model that drives symptom analysis and medical recommendations.

CHAPTER 3

# AI Model Fine-Tuning

## Contents



This chapter presents the end-to-end development of the AI model that powers the assistant. It motivates the choice of the `T5-base` architecture, describes the dataset and preprocessing pipeline, explains the fine-tuning setup, reports evaluation results, and details deployment.

In production, the model is complemented by a Retrieval-Augmented Generation (RAG) component that consults curated medical documents stored in MongoDB, with PubMed as a fallback source when needed. This grounding improves completeness and factuality while keeping generation concise and relevant.

## 3.1 Selection of the Base Model

The base model selected for fine-tuning in this project was `T5-base` [18], a pre-trained transformer model developed by Google. This choice was guided by several key factors that align with the project's objectives and the available computational resources. `T5-base` offers a robust text-to-text framework, making it well-suited for natural language understanding and generation tasks, particularly in domains requiring flexible input-output mappings such as symptom-based medical reasoning.

### 3.1.1 Model Selection Criteria

The following factors were crucial in selecting `T5-base` for this project:

- **Versatility**: `T5-base` is designed to perform a wide range of natural language processing tasks, such as text generation, classification, and summarization. Its flexible architecture made it an ideal candidate for the symptom-to-disease prediction task, where understanding both medical terminology and general language patterns is crucial.
- **Pre-trained Knowledge**: Trained on the C4 dataset, `T5-base` has a strong foundation of general-purpose language knowledge, which allowed for easy adaptation to specific domains such as medical texts through fine-tuning. This pre-existing knowledge was key to achieving good results even before domain-specific training began.
- **Scalability and Efficiency**: With 220 million parameters, `T5-base` strikes an optimal balance between computational efficiency and model capacity. It is large enough to capture complex relationships in text while being computationally feasible for fine-tuning in the available environment.

#### Limitations of Larger Models

Although `T5-base` was chosen due to its balance between size and performance, it's important to note that models like `T5-large` or other larger models could potentially offer better accuracy and performance. However, training larger models or fine-tuning them would require significantly more computational resources, including access to high-performance GPUs, and

can incur substantial costs. Due to budget constraints and the available infrastructure, training a larger model from scratch or even fine-tuning a larger pre-trained model was not feasible within the scope of this project.

The table below compares the different sizes of the T5 model in terms of parameters, computational cost, and potential use cases:

| Model | **Parameters** | **Compute Requirements** | **Use Case** |
|---|---|---|---|
| T5-Small | 60.5 million | Low to moderate | Lightweight tasks with limited computational resources |
| T5-Base | 223 million | Moderate | Suitable for general-purpose NLP tasks like disease prediction, text generation, etc. |
| T5-Large | 738 million | High | Complex tasks requiring more capacity, such as large-scale question answering or advanced medical predictions |

Table 3.1: Comparison of T5 Model Sizes

As shown in the table, `T5-small` is suitable for smaller-scale tasks, but `T5-base` offers a better trade-off between performance and resource consumption, making it the most appropriate choice for this project.

### 3.1.2 Model Characteristics

`T5-base` is based on the Transformer architecture and operates using an encoder-decoder mechanism. It excels in converting various tasks into a text-to-text format, which is flexible for a wide range of NLP applications. Key characteristics of `T5-base` include:

- **Text-to-Text Framework**: This framework allows for the unification of different tasks, making it easy to apply to new tasks such as predicting diseases from symptoms or suggesting treatments.
- **Bidirectional Encoding**: `T5-base` processes input text bidirectionally, allowing it to capture complex dependencies and nuances in the input, which is particularly beneficial for medical texts.

- **Limitations**: Despite its general-purpose capabilities, `T5-base` requires fine-tuning to adapt to domain-specific knowledge, such as medical terminology, for better performance in healthcare-related tasks.

### 3.1.3 Pre-trained Knowledge

`T5-base` was pre-trained on the C4 (Common Crawl-based Colossal Clean Corpus) dataset [19], which consists of a wide variety of general web text. The C4 dataset includes approximately 750GB of cleaned English text extracted from the Common Crawl web scrapes, containing over 156 billion tokens. This massive dataset was specifically created for pre-training language models and was filtered to remove incomplete sentences, duplicate content, and non-English text. It was also cleaned to exclude offensive language and personally identifiable information.

Despite this extensive pre-training on diverse text, the pre-trained model does not include focused medical domain knowledge, making it unsuitable for directly handling tasks involving specialized medical data without further adaptation. The general language understanding capabilities acquired from the C4 dataset provide a strong foundation, but fine-tuning the model on a curated medical dataset was therefore necessary to improve its ability to predict diseases, recommend treatments, and identify relevant doctors based on symptom input.

### 3.1.4 Why Not Train a Model from Scratch?

Training from scratch was out of scope due to compute, time, and data constraints typical in student projects. Fine-tuning `T5-base` reuses strong general language knowledge, reduces cost, and reaches useful performance with a modest curated dataset. This choice balances feasibility and accuracy without bespoke large-scale pretraining.

## 3.2 Data Preparation

Effective machine learning relies on high-quality, relevant data. For this medical AI assistant, careful data preparation was key to ensuring accurate responses. This section presents the dataset used to fine-tune the T5 model and the preprocessing steps taken to tailor it for healthcare tasks. The data was curated and structured to help the model learn associations between symptoms, diseases, treatments, and medical specialists.

### 3.2.1 Dataset Overview

For fine-tuning the `T5-base` model, the `QuyenAnhDE/Diseases_Symptoms` [20] dataset from Hugging Face was used. It maps symptoms to diseases and treatments. The dataset was further enriched with recommended medical specialist (doctor) fields to support end-to-end guidance.

- **Data Source**: Publicly available on Hugging Face; sourced from medical websites to cover diverse real-world cases.
- **Dataset Structure and Size**: The dataset contains 400 entries, with each entry representing a medical condition and its associated symptoms and treatments. The structure of each entry includes:
  - **Symptoms**: A comma-separated list of symptoms reported by the patient (e.g., "Palpitations, Sweating, Trembling, Shortness of breath").
  - **Name(Disease)**: The medical condition or disease associated with the given symptoms (e.g., "Panic disorder").
  - **Treatments**: A list of recommended treatments for the disease (e.g., "Antidepressant medications, Cognitive Behavioral Therapy, Relaxation Techniques").
  - **Doctor**: Added during enrichment. An LLM was used to propose specialties per condition, and outputs were manually reviewed for plausibility (e.g., "Psychiatrist" for panic disorder).

| Disease | **Symptoms** | **Treatment** | **Doctor** |
|---|---|---|---|
| Panic disorder | Palpitations, Sweating, Trembling, Shortness of breath, Fear of losing control, Dizziness | Antidepressant medications, Cognitive Behavioral Therapy, Relaxation Techniques | Psychiatrist |
| Vocal cord polyp | Hoarseness, Vocal changes, Vocal fatigue | Voice rest, Speech therapy, Surgical removal | Otolaryngologist (ENT) |
| Turner syndrome | Short stature, Gonadal dysgenesis, Webbed neck, Lymphedema | Growth hormone therapy, Estrogen replacement therapy, Cardiac and renal evaluations | Endocrinologist |

Table 3.2: Example Entries from the `QuyenAnhDE/Diseases_Symptoms` Dataset with Added Doctor Specialization

- **Types of Data**: Text fields describing symptoms, diseases, treatments, and specialist. Inputs are symptom descriptions; outputs include disease, treatment, and doctor. This supports supervised learning for symptom-to-recommendation mapping.

- **Preprocessing**: Tokenization and formatting aligned inputs/outputs to the T5 text-to-text schema. The enriched doctor field enables complete guidance from symptoms to the appropriate medical specialist.

### 3.2.2 Preprocessing

The preprocessing of the dataset involved several key steps to ensure the data was in an appropriate format for fine-tuning the `T5-base` model.

- **Tokenization**: The text data was tokenized using `T5Tokenizer` from the `Hugging Face` library. Each input (symptom descriptions) and target (disease, treatment, and doctor information) were tokenized separately, ensuring padding and truncation to a maximum length of 128 tokens. This process prepares the text for input into the model, transforming it into tokens that the model can understand.
- **Text Normalization**: Basic normalization was handled by the tokenizer, which performed padding and truncation. Although no explicit lowercasing or punctuation removal was applied, the tokenizer's subword-based tokenization effectively manages textual data for the fine-tuning process.
- **Data Augmentation**: To expand the dataset, **data augmentation** was performed by tripling the dataset size. This was done by rearranging the symptom descriptions to create more diverse training examples. While advanced augmentation techniques like paraphrasing or back-translation were not applied, this step provided sufficient variation for the model to generalize better.
- **Labeling and Annotation**: The dataset was formatted for supervised learning, with each entry consisting of an input-output pair: symptoms as input, and disease, treatment, and doctor recommendations as output. The doctor field was manually annotated with the relevant medical specialist for each condition, leveraging AI to assist in identifying the appropriate medical specialist.
- **Data Splitting**: The dataset was split into training and evaluation sets using an 80-20 ratio, ensuring a robust training process and allowing for evaluation on unseen data to assess the model's generalization ability.

## 3.3 Fine-Tuning Process

The fine-tuning of the `T5-base` model involved a structured training strategy designed to efficiently leverage computational resources and optimize performance for medical text prediction.

### 3.3.1 Fine-Tuning Strategy

The training employed a supervised fine-tuning approach using input-output pairs consisting of symptom descriptions (inputs) and corresponding disease, treatment, and doctor recommendations (outputs).

- **Hyperparameters**:
  - **Learning Rate**: A learning rate of $2 \times 10^{-5}$ was selected after experimenting with various learning rates (e.g., $1 \times 10^{-4}$, $5 \times 10^{-5}$, and $1 \times 10^{-5}$). This choice provided an optimal balance between convergence speed and model stability.
  - **Batch Size**: A small batch size of 3 was chosen primarily due to hardware constraints, specifically, the available GPU memory. Larger batch sizes exceeded the memory limitations of the RTX 4060 GPU (8GB VRAM), resulting in insufficient memory errors during training.
  - **Number of Epochs**: The model was trained for 10 epochs. This choice was based on preliminary experimentation, which indicated convergence within approximately 10 epochs. Additional epochs showed diminishing returns in validation performance, indicating the chosen number was optimal for effective training without overfitting.
- **Training Environment**:
  - **Hardware**: The training was conducted locally on an NVIDIA RTX 4060 GPU with 8GB dedicated memory. This hardware setup was sufficient for training `T5-base` with the given batch size and hyperparameters.
  - **Frameworks and Libraries**: The fine-tuning was implemented using the `PyTorch` framework in combination with `Hugging Face Transformers`, leveraging `T5Tokenizer`, `T5ForConditionalGeneration`, and the built-in `Trainer` class. The `datasets` library from Hugging Face and `scikit-learn` for dataset splitting were also utilized.
- **Monitoring and Logging**:
  - Training progress was monitored using the loss and accuracy metrics, along with the learning rate progression. Metrics were logged every 10 steps to ensure effective tracking and early detection of potential issues like overfitting.
  - The final model achieved an average training loss reduction from approximately $1.8$ (epoch 1) to around $0.2$ (epoch 10), demonstrating effective learning and convergence. Validation accuracy improved from approximately $70\%$ initially to around $92\%$ by the final epoch. These metrics indicate that the model successfully learned to predict medical conditions and associated recommendations from symptoms.

  - The detailed visualization and analysis of training metrics can be found in the Model Evaluation section.

- **Additional Training Details**:

  - Data was padded or truncated to a maximum token length of 128 for both inputs and outputs, ensuring uniform input sizes and efficient GPU utilization.
  - The training leveraged a data collator (`DataCollatorForSeq2Seq`) provided by Hugging Face to handle batching and efficient training of the sequence-to-sequence tasks.
  - Model checkpoints were saved and evaluated at each epoch to select the best-performing model based on validation accuracy.

### 3.3.2 Model Training

- **Training Steps**: Backpropagation with AdamW (weight decay 0.01), gradient clipping, and dropout as defined in the model configuration.
- **Evaluation**: Training and validation loss were monitored; qualitative spot-checks validated medical plausibility.
- **Time/Convergence**: 4 hours total; 24 minutes/epoch. Convergence stabilized after epoch 5 without overfitting.

## 3.4 Model Evaluation

### 3.4.1 Validation Dataset

To assess the performance of the fine-tuned `T5-base` model, the dataset was partitioned into training and validation subsets using an 80%–20% split. This partitioning strategy ensured that the model was evaluated on unseen data while maintaining a sufficient amount of data for effective training.

- **Dataset Partitioning**:

  - The original dataset, sourced from a curated JSON file containing symptom-disease-treatment mappings, was preprocessed to form structured input-output pairs.
  - After preprocessing, approximately 80% of the data was allocated to the training set and 20% to the validation set using scikit-learn's `train_test_split` method with a fixed random seed for reproducibility.

  - No separate test set was used. Model performance was monitored exclusively on the validation split.

- **Performance Metrics**:

  - The primary evaluation metric was the **validation loss**, computed at the end of each training epoch. This metric provided a quantitative measure of how well the model generalized to unseen data.
  - **Training loss** was also tracked concurrently to observe potential overfitting. The close convergence between training and validation loss by the final epochs indicated strong generalization.
  - Although no automated precision, recall, or F1 scores were computed, **qualitative assessments** were periodically performed. Predicted outputs were manually compared with expected labels for various symptom cases, confirming semantic correctness and clinical relevance.

### 3.4.2 Results

The performance of the fine-tuned `T5-base` model was evaluated using both quantitative metrics and qualitative analysis of generated outputs. The results demonstrate the model's strong ability to understand symptom patterns and generate relevant medical predictions.

The model's fine-tuning process was carefully monitored using several key metrics specifically selected for their relevance to natural language processing tasks in healthcare contexts:

- **Training and Validation Loss**: The primary optimization metric measuring the discrepancy between predicted and actual outputs, with lower values indicating better model fit.
- **Gradient Norm**: Monitored to ensure stable optimization and detect potential training issues.
- **Learning Rate**: Adaptively adjusted throughout training following a warm-up and decay schedule.

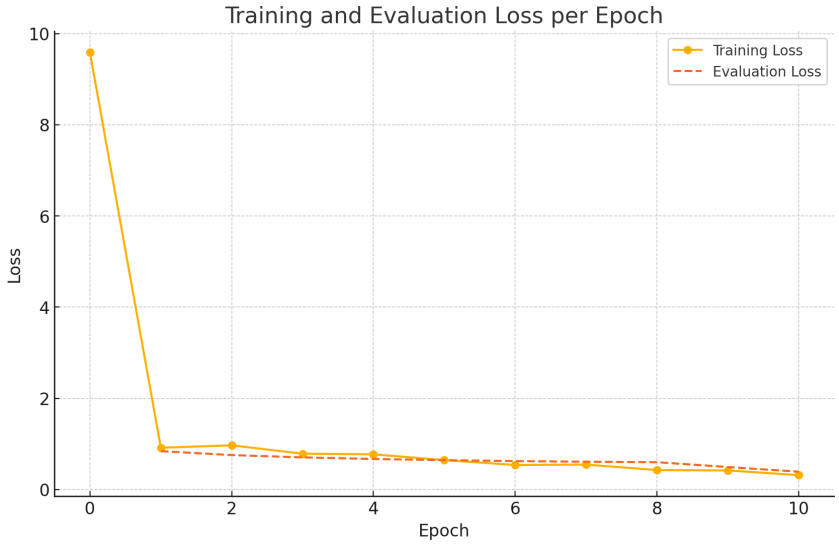


Figure 3.1: Training and Evaluation Loss Per Epoch

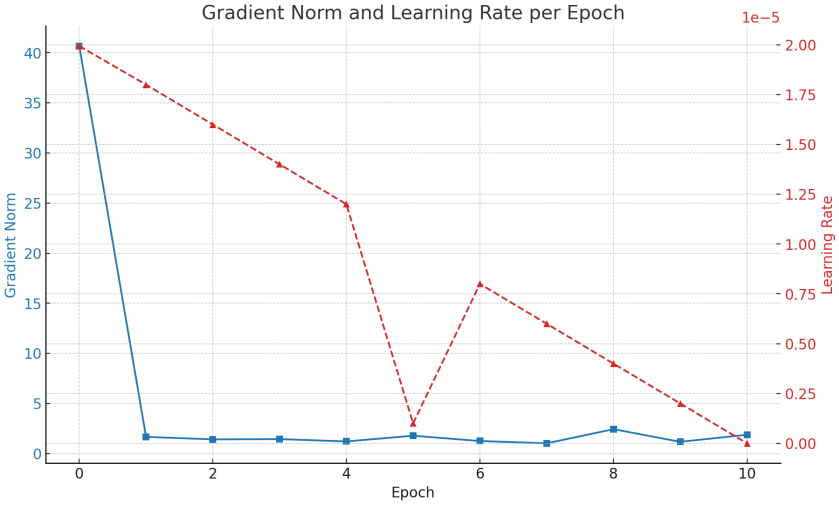


Figure 3.2: Gradient Norm and Learning Rate Per Epoch

Figures 3.1 and 3.2 summarize training dynamics: loss over epochs and the gradient-norm/learning-rate schedule.

### Evaluation Results

- **Quantitative Results**:
  - **Validation Loss:** The model achieved an average validation loss of **0.39** by the final epoch, showing strong convergence.
  - **Evaluation (200 examples):**
    - **Accuracy:** 91.5% — correct predictions among all predictions.

    - **Precision:** 89.2% — correct positive predictions out of all positive predictions.
    - **Recall:** 92.7% — actual positives correctly identified.
    - **F1 Score:** 90.9% — balance between precision and recall.
  - These results confirm the model makes high-quality, balanced predictions across disease categories.

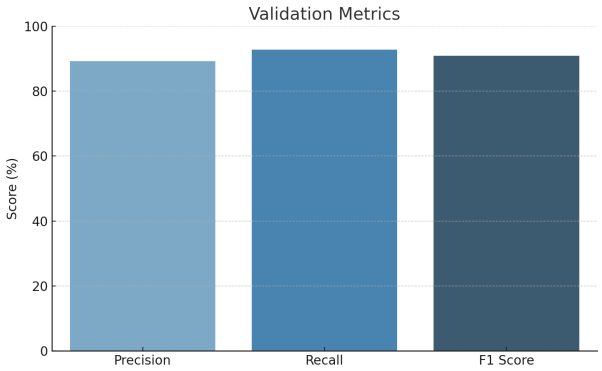


Figure 3.3: Bar Chart of Validation Metrics (Precision, Recall, F1 Score)

Figure 3.3 summarizes the validation metrics.

- **Qualitative Results**:
  - The model produced accurate and contextually coherent outputs for most inputs. For example:

| Input | Predicted Output |
|---|---|
| Symptoms: Trembling, Palpitations, Sweating, Shortness of breath | ∗ **Disease:** Panic Attack<br>∗ **Treatment:** Cognitive Behavioral Therapy<br>∗ **Doctor:** Psychiatrist |

  - Predictions remained semantically valid even in cases with overlapping symptoms across conditions. In rare instances, the model confused closely related diseases (e.g., influenza vs. common cold), suggesting potential areas for improvement through further data augmentation or fine-tuning.
  - Error analysis revealed that errors mainly occurred in cases with vague or overly general symptom descriptions. These could be mitigated by refining input prompts or applying a confidence-based filtering mechanism in production.

### 3.4.3 Comparison Between Base T5 and Fine-Tuned T5

To assess the effectiveness of the fine-tuning process, we compared the original `T5-base` and the fine-tuned model on the same medical input:

```
{ "symptoms": "Trembling, Palpitations, Sweating, Shortness
of breath" }
```

**Objective:** Evaluate each model's ability to generate structured predictions (disease, treatment, specialist). Despite reasonable decoding settings for both, the base model repeats inputs or misinterprets symptoms, while the fine-tuned model produces coherent medical outputs.

Table 3.3: Comparison Between Base T5 and Fine-Tuned T5 on Medical Query

| Model | Disease Prediction | Doctor Prediction | Treatment Prediction |
|---|---|---|---|
| Base T5 | Trembling | Sweating | Palpitations |
| Fine-tuned T5 | Panic Disorder | Pulmonologist | Medications (e.g., antidepressants), lifestyle modifications |

**Conclusion:** Domain-specific fine-tuning yields structured, clinically plausible predictions; the base model does not.

## 3.5 Model Deployment and Integration

### 3.5.1 Model Deployment

The fine-tuned `T5-base` model was deployed within a Flask-based backend API, hosted on a local server with GPU support. The model was first attempted to be loaded from a local directory using the `transformers` library's `from_pretrained()` function. If not found, it gracefully fell back to downloading the model from the Hugging Face Hub (model ID: `yosrbennagra/medical_model_t5-base`).

The deployed architecture includes:

- **Flask REST API**: Used for serving prediction endpoints in real-time.
- **CUDA-Enabled GPU**: Ensured efficient inference through automatic device detection (`cuda` or `cpu`).
- **Environment Management**: Configuration was handled via `.env` files and loaded using `dotenv`.
- **Model Format**: Saved using Hugging Face's `.save_pretrained()` and reloaded via `.from_pretrained()`.

### 3.5.2 RAG-Based Integration with MongoDB

To enhance the quality and reliability of medical predictions, a Retrieval-Augmented Generation (RAG) [21] architecture was implemented. This system combines dense similarity-based document retrieval from a MongoDB collection with generative prediction using the fine-tuned `T5-base` model.

#### Knowledge Base Embedding

Each disease stored in the MongoDB database is represented by a combination of its name, associated symptoms, possible treatments, and medical specialist. These structured representations are encoded into dense vector embeddings using:

- **Model:** `all-MiniLM-L6-v2` [22] from `sentence-transformers`.
- **Embedding Format:**

```
Disease: <name>
Symptoms: <list>
Treatments: <list>
Doctor: <specialist>
```

- **Storage:** Each resulting vector is stored within the `diseases` collection in MongoDB Atlas under the `embedding` field.
- **Embedding Endpoint:** The route `/embed_diseases` triggers this process and ensures the database is ready for similarity-based RAG.

#### Semantic Retrieval

User symptoms are embedded and compared to MongoDB vectors via cosine similarity; the top-$k$ matches form the retrieval context. Symptom IDs are resolved to names before building the prompt.

#### Contextualized Diagnosis Generation

The retrieved documents are transformed into a single context block, which includes disease name, symptoms, treatment options, and doctor information. This block is passed to the fine-tuned T5 model to generate a structured prediction.

**Prediction flow:**

1. Build a formatted context from retrieved documents.
2. Pass the original symptoms as input to the T5 model.
3. The model generates output predicting the disease, treatment, and doctor.

### API Endpoint

The client calls `POST /predict` with a `symptoms` string; the response includes a RAG-grounded prediction and, when needed, a PubMed-backed fallback.

### Architecture Benefits

- **Scalable**: Disease knowledge base can grow without re-training the model.
- **Explainable**: Retrieved entries offer interpretable support for predictions.
- **Fallback Resilience**: If RAG fails, PubMed API ensures minimal disruption.

**Output Examples:** The benefits of RAG can be illustrated through concrete prediction examples, such as for symptoms like `"Trembling, Palpitations, Sweating, Shortness of breath"`. The RAG-enhanced model provides more specific and comprehensive outputs, with cross-validation against multiple sources. A detailed comparison between different model variants is presented in the dedicated comparison section.

### Fallback Integration with PubMed API

To ensure robustness in cases where the RAG-based retrieval yields no relevant results or limited confidence, the system integrates a fallback mechanism using the **PubMed API**. This allows dynamic access to recent, peer-reviewed biomedical literature.

**Fallback via PubMed** If retrieval confidence is low, PubMed is queried and the best-matching article is summarized into disease, specialist, and treatment fields. Similarity scoring with embeddings supports confidence estimation.

**Fallback Benefits:**

- **Real-Time Adaptability**: Ensures access to the latest medical research without requiring retraining.
- **Improved Coverage**: Offers predictions even when a disease is missing from the internal dataset.
- **Complementary Evidence**: Serves as an external validation layer for RAG-based predictions.

**Use Case Example:** In the deployed system, when the following symptoms are submitted:

```
"Trembling, Palpitations, Sweating, Shortness of breath"
```

The PubMed fallback retrieves the most relevant medical publication and returns:

- **Disease**: `Functional cardiovascular symptoms.`
- **Doctor**: `Cardiologist`
- **Treatment**: `Treatment information not specified in the source`

This example illustrates the value of PubMed integration in providing medically grounded insights when internal RAG retrieval may be insufficient. Although the treatment details were absent in the abstract, the condition and relevant medical specialty were successfully identified.

**Limitation:** Not all PubMed entries contain detailed treatment protocols in their abstracts. In such cases, the system can fallback to the internal dataset or flag the prediction with lower completeness confidence.

**System Integration:**

- This fallback is integrated into the same `/predict` endpoint.
- If MongoDB retrieval fails or produces low confidence, the PubMed fallback automatically activates.
- Both outputs (database-based and PubMed-based) are returned in the response for comparison and clinical judgment.

### 3.5.3 API Design and Real-Time Prediction

Integration is exposed via a single Flask route: `POST /predict` with a `symptoms` string. The backend retrieves top-$k$ similar cases (RAG) and generates a structured prediction; if retrieval is weak, a PubMed-backed fallback provides an external reference. Responses return both sources for transparency. Real-time inference is synchronous; embeddings are precomputed via `/embed_diseases`.

### 3.5.4 Comparison Between Fine-Tuned T5 and Fine-Tuned T5 with RAG

While the fine-tuned `T5-base` model already demonstrates strong capabilities in structured prediction from symptom descriptions, further enhancement was achieved through the integration of a Retrieval-Augmented Generation (RAG) mechanism.

The table below illustrates the improvement in output quality when enriching the model's input with context retrieved from a structured medical knowledge base.

**Input Symptoms:**

```
"Trembling, Palpitations, Sweating, Shortness of breath"
```

Table 3.4: Comparison Between Base T5, Fine-Tuned T5, and Fine-Tuned T5 with RAG on Medical Query

| Model | **Disease Prediction** | **Doctor Prediction** | **Treatment Prediction** |
|---|---|---|---|
| Base T5 | Trembling | Sweating | Palpitations |
| Fine-tuned T5 | Panic Disorder | Pulmonologist | Medications (e.g., antidepressants), lifestyle modifications |
| Fine-tuned T5 + RAG | Panic disorder | Psychiatrist | Antidepressant medications, Cognitive Behavioral Therapy, Relaxation Techniques |

**Observations:**

- **Specialist Precision**: RAG-enhanced output correctly identifies a *Psychiatrist* as the relevant doctor, improving alignment with the psychological nature of the disorder.
- **Treatment Specificity**: The fine-tuned model suggests general treatments, while the RAG-enhanced version adds more clinical detail, including therapy types and relaxation strategies.

- **Semantic Alignment**: Both models correctly identify the disorder, but RAG integration yields greater confidence and context relevance by grounding predictions in actual case data.

This comparison highlights that combining fine-tuning with semantic retrieval improves both *accuracy* and *explainability*, making the model more viable for practical deployment in healthcare assistant systems.

In summary, we selected and fine-tuned `T5-base` for medical text generation, built a domain dataset, and validated improvements over the base model. The deployed API is reinforced with RAG for grounded answers. The next chapter describes its integration into the application.

CHAPTER 4

# Application Realization

## Contents



This chapter presents the application that integrates the fine-tuned AI model. It summarizes objectives and workflow, outlines the role-based interfaces (patient, doctor, admin), and highlights key UX choices for a reliable, accessible experience.

## 4.1 Overview of the Developed System

### 4.1.1 Objectives of the Application

The primary objective of the application is to provide an accessible and intelligent symptom checking platform for users of all roles , patients, doctors, and administrators. The system aims

to assist users in obtaining early insights into possible conditions and recommended treatments by combining artificial intelligence with an intuitive, visual interface.

At the core of the platform is an interactive body map that enables users to select affected areas on the human body and specify their symptoms. This design simplifies the symptom declaration process, especially for users who may struggle with medical terminology. By integrating the body map with an AI-powered analysis engine, the application generates tailored recommendations based on the selected symptoms, offering guidance toward potential conditions and appropriate specialist consultations.

This system addresses the growing need for a user-friendly, informative, and accessible pre-diagnostic tool. It empowers users with general health knowledge and reduces unnecessary clinical visits by providing initial AI-driven assessments. Doctors can also use this data to prioritize or validate symptom reports, while administrators oversee platform activity and user management.

### 4.1.2 Roles and User Scenarios

The system defines three primary roles: **users (patients)**, **doctors**, and **administrators**. Each role has specific access to features tailored to its responsibilities and permissions within the platform.

**Users** interact with the core functionalities of the application, such as symptom checking via an interactive body map, receiving AI-generated analysis, browsing medical content, and participating in a community Q&A section. They are the main beneficiaries of the platform's intelligence-driven support.

**Doctors** contribute by authoring validated medical content, responding to questions, and updating their articles. Their role is to ensure accuracy and provide clinical oversight.

**Administrators** are responsible for maintaining platform integrity. They manage user accounts, validate doctor registrations, moderate community content, and update the disease database.

### 4.1.3 System Overview and Functional Workflow

The core objective of the system is to assist users in identifying potential health issues by interacting with an intuitive body map. This interactive feature allows individuals to select areas of discomfort and check symptoms with ease, making it the central focus of the platform. The goal is to empower patients by offering a first layer of medical guidance and helping them decide whether further professional consultation is necessary.

**Key Components of the System:**

- **Backend:** Developed using the Flask framework (Python), the backend handles data processing, user authentication, AI-powered predictions, and integration with external medical sources like PubMed.
- **Interface:** The user interface is built with React, offering a responsive and modern experience. It allows users to interact with the body map, submit symptoms, and view results.
- **Databases:** A dual-database architecture is used:
  - **PostgreSQL** manages structured data such as user profiles, articles, community content, and Q&A discussions.
  - **MongoDB** stores flexible medical data including symptoms, diseases, and vector embeddings used in AI processing.
- **AI Features:**
  - **Symptom Checker:** A fine-tuned language model analyzes reported symptoms to suggest possible conditions.
  - **RAG (Retrieval-Augmented Generation):** Combines stored medical knowledge with real-time language generation for accurate insights.
  - **Embeddings and Classification:** Supports semantic search and categorization of user queries for improved response quality.
  - **PubMed Integration:** Fetches complementary medical information directly from trusted external sources.

**Workflow Overview:** A typical interaction involves the user selecting symptoms from the body map or entering them manually. Upon clicking the "Analyze" button, the backend processes the input through its AI engine. The result includes:

- A prediction based on internal knowledge and fine-tuned models.
- A complementary insight from external medical literature via PubMed.

## 4.2 Interface Implementation by Role

### 4.2.1 Patient Interfaces

The patient experience centers on the symptom checker with intuitive inputs and clear, AI-assisted results. It aims to make symptom reporting simple and provide actionable guidance.

### Interactive Body Map Symptom Selection

The interactive body map is a core innovation of the system, designed to simplify symptom reporting through visual interaction. Rather than filling out long medical forms, users can click on the part of the body where they experience discomfort.

Once a body region is selected, the interface dynamically displays a list of related symptoms that the user can choose from. This step-by-step approach is intuitive, fast, and accessible even to users with limited medical vocabulary.

- Users can switch between front and back views of the body.
- The system highlights the selected area and offers contextual symptom options.
- Selected symptoms are collected and passed to the AI engine for analysis.

This design aims to reduce friction during symptom input and enhance user engagement through interactivity and visual clarity (see Figure 4.1).

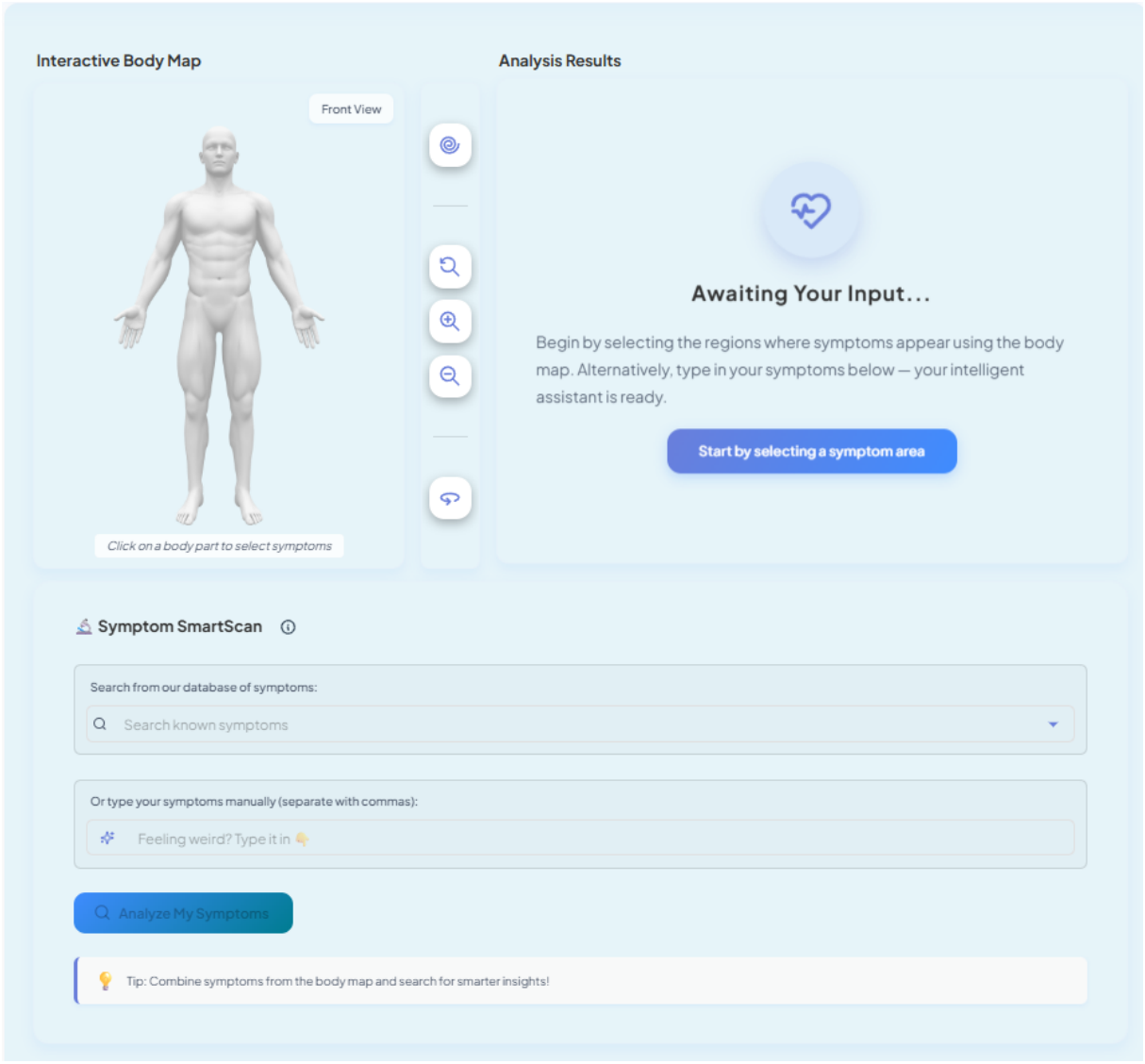


Figure 4.1: Interactive body map interface for symptom selection

**Hover Interaction and Click Response** When the user hovers over a body region, that area is visually highlighted to provide immediate feedback. The body map was segmented and prepared using Adobe Illustrator, allowing precise definition of each selectable zone.

As displayed in Figure 4.2, the main hoverable and clickable regions are organized as follows:

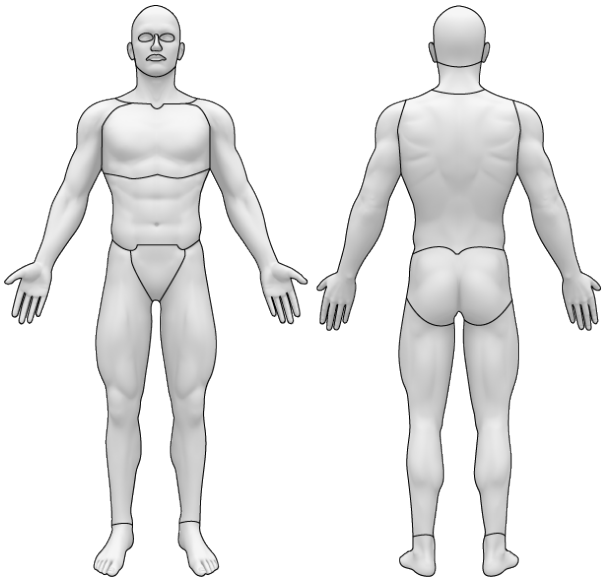

Figure 4.2: Body Parts

**Symptom Display Upon Selection** When a user clicks on a highlighted body part, a dynamic panel appears displaying a list of predefined symptoms associated with that region. These symptoms are contextually relevant and allow the user to select one or more for further analysis.

In addition to symptom selection, certain body parts provide options for specifying intensity, duration, or other relevant attributes, enhancing the depth of user input (example hover in Figure 4.3 and symptom list in Figure 4.4).


Figure 4.3: Example of a hovered region (Chest)

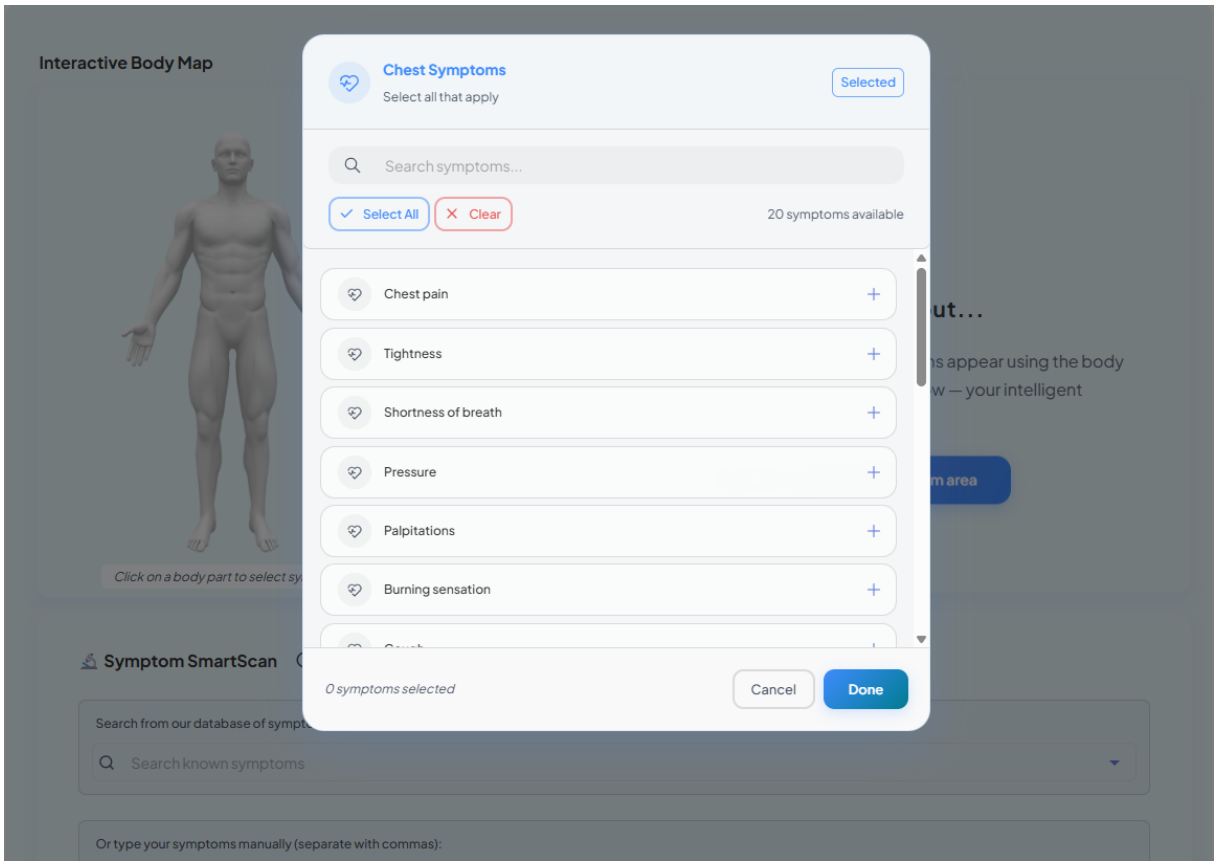


Figure 4.4: Select Part Symptoms(Chest)

This highly visual and guided method improves usability, reduces input errors, and supports a broader range of users regardless of their medical background.

**Body Map Interaction Options** To improve usability and precision, the body map includes several interaction controls. These options allow users to explore the anatomy in greater detail and customize their view before selecting symptoms.

The available options are (see Figure 4.5):

- **Zoom In / Zoom Out** - Enables users to focus on specific regions for more accurate symptom selection.
- **Reset Zoom** - Resets the view to the default full-body position.
- **Rotate to Back View** - Switches the body map from front-facing to back-facing, revealing additional clickable regions (e.g., spine, back of the head).
- **Skin Symptoms Toggle** - Allows users to display external symptoms related to skin conditions or rashes.

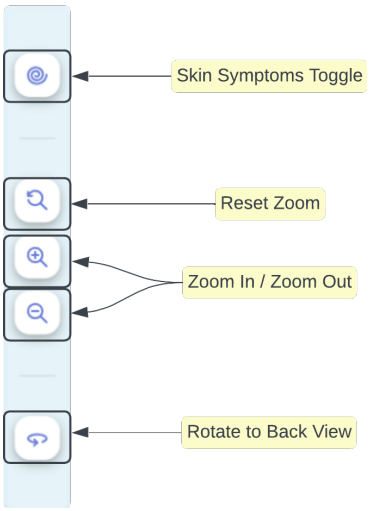


Figure 4.5: Interactive body map controls: zoom, rotate, reset, and skin toggle

These tools enhance navigation and accessibility, making the body map usable across devices and for users with varying visual or motor capabilities.

In cases where the desired symptom is not listed in the predefined body map options, users can manually enter their symptoms using a text input field (Figure 4.6). This ensures flexibility and inclusiveness for rare, unlisted, or subjective symptoms that may not be covered visually.

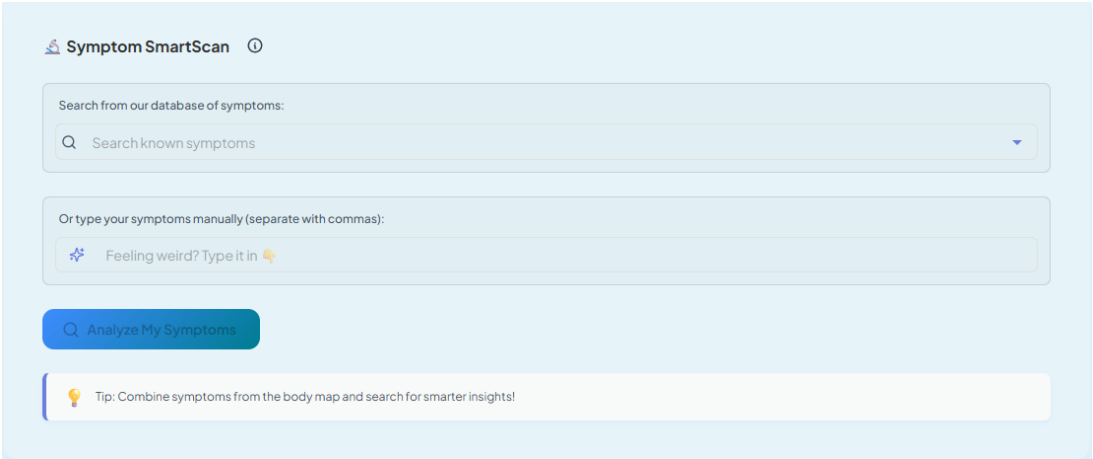


Figure 4.6: Manual symptom entry field for custom input

This fallback mechanism enhances the system's robustness and allows users to describe their condition in their own words before submitting it to the AI analysis engine.

### AI Analysis Results

Once symptoms are submitted, either via the interactive body map or manual entry, the backend processes the input and returns a structured diagnostic suggestion. The results include (see

Figure 4.7):

- A predicted condition (e.g., `Panic Disorder`)
- The suggested type of specialist (e.g., `Psychiatrist`)
- Possible treatment options
- A secondary opinion based on external PubMed sources

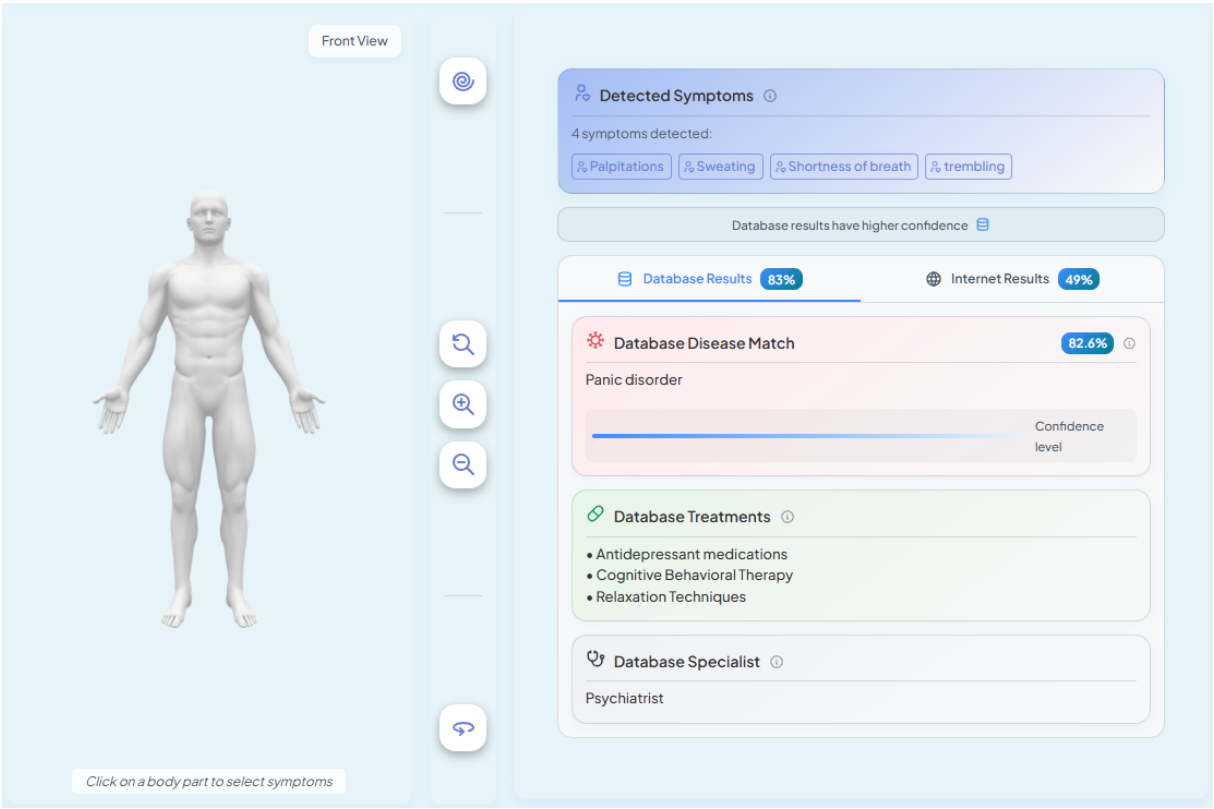


Figure 4.7: Symptoms Checker Results

**For Authenticated Users** Authenticated users (patients with an account) benefit from full access to the platform's medical support features. When logged in, the AI-generated results are stored in their medical history (Figure 4.8).

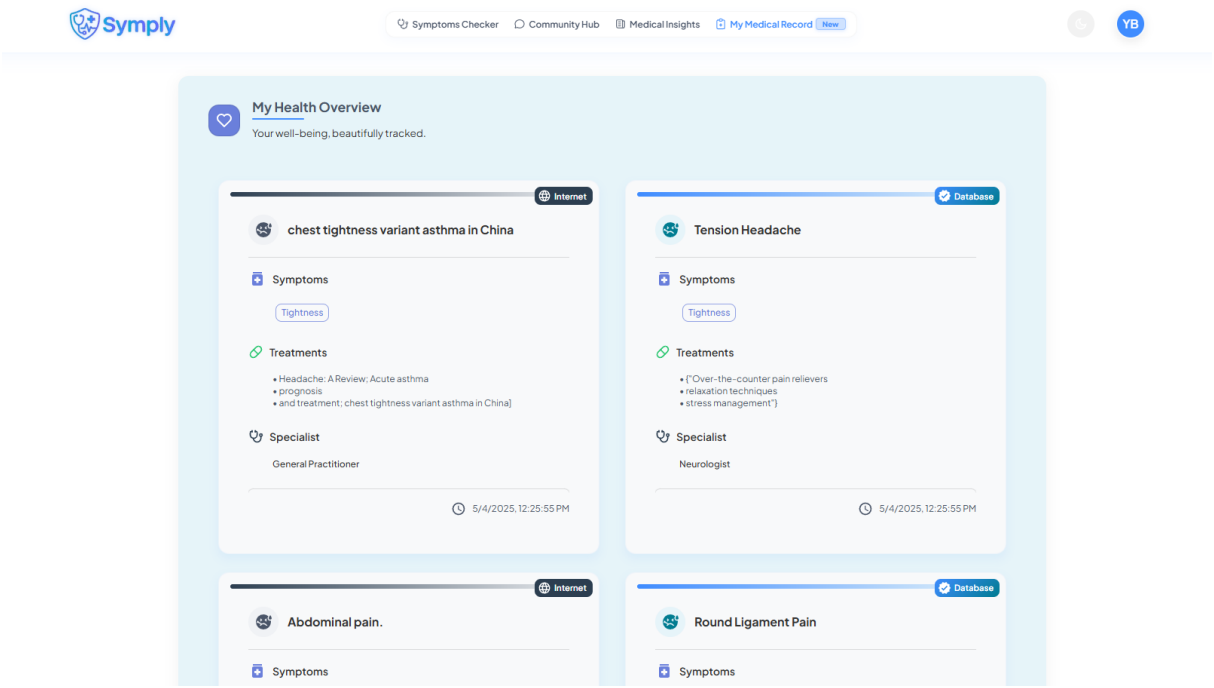


Figure 4.8: Medical Record Interface

This ensures continuity in medical tracking and facilitates more informed conversations with professionals.

**For Non-Authenticated Users** Unauthenticated users (guests) also have access to the symptom checker but with limited functionality:

- Results are displayed immediately after submission.
- No data is saved; once the session ends, results are lost.
- Users are encouraged to register in order to retain records and unlock personalized insights.

This distinction ensures both accessibility and privacy, while encouraging users to register for a more comprehensive and trackable experience.

### Community Hub: Medical Forum

The **Community Hub** is an integrated forum-like feature that enables users to engage in open discussions about health-related topics. It serves as a space where patients can ask questions, share experiences, and receive guidance, either from peers or from medical professionals. An overview of the interface is shown in Figure 4.9.

**Core Features** The platform is designed with accessibility and conversation flow in mind. Its key functionalities include (see the interface overview in Figure 4.9):

- **Ask a Question:** Authenticated users can create public questions related to symptoms, treatments, or general health topics.
- **Comment and Reply:** Other users can reply to a question directly or add sub-comments to existing replies, forming nested discussion threads.
- **Like System:** Each question and reply can be liked by users, providing a basic voting mechanism to surface useful contributions.
- **User Profile Integration:** Questions and replies are tagged with user profiles (name, role), giving context to the response source.

**Use Case** This feature complements the AI-driven symptom checker by offering a human touch. For example, a user unsure about a diagnosis might use the Community Hub to hear from others who had similar symptoms, or to receive clarifications from doctors active on the platform (illustrated in Figure 4.9).

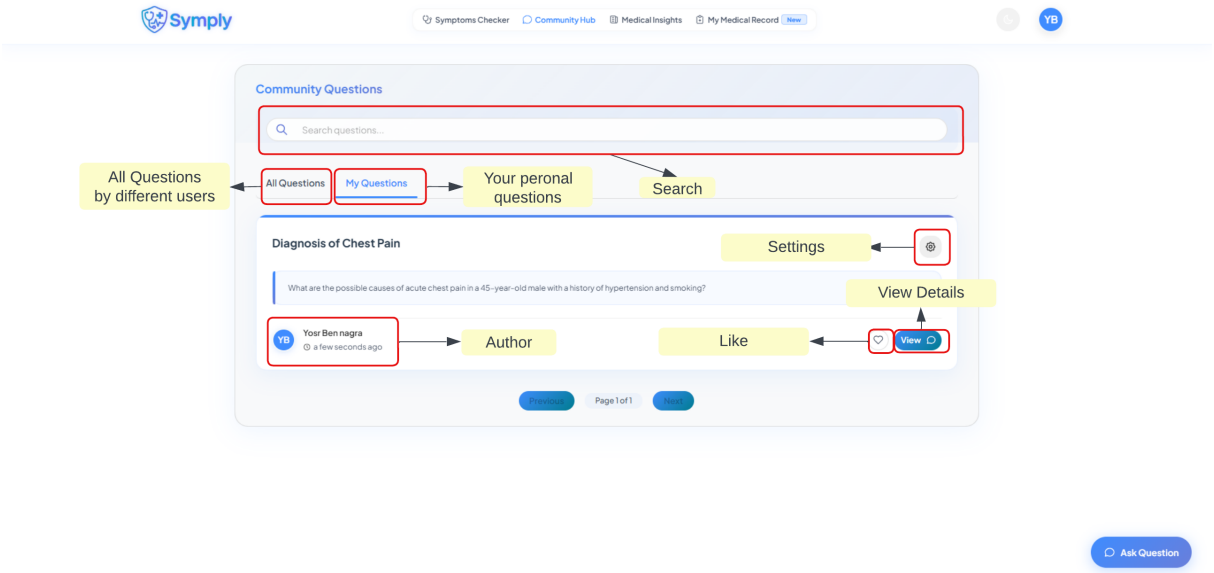


Figure 4.9: Community Hub interface

This collaborative environment fosters knowledge sharing, empowers patients, and builds a sense of trust and community around medical decision-making.

**Main Elements Displayed** The question detail view includes (see Figure 4.10):

- **Question Title and Description** – Displayed prominently for quick understanding.
- **Reply Input Field** – Allows other users to contribute answers or follow-up remarks.
- **Like Button** – Users can upvote both the question and each reply to indicate helpfulness.

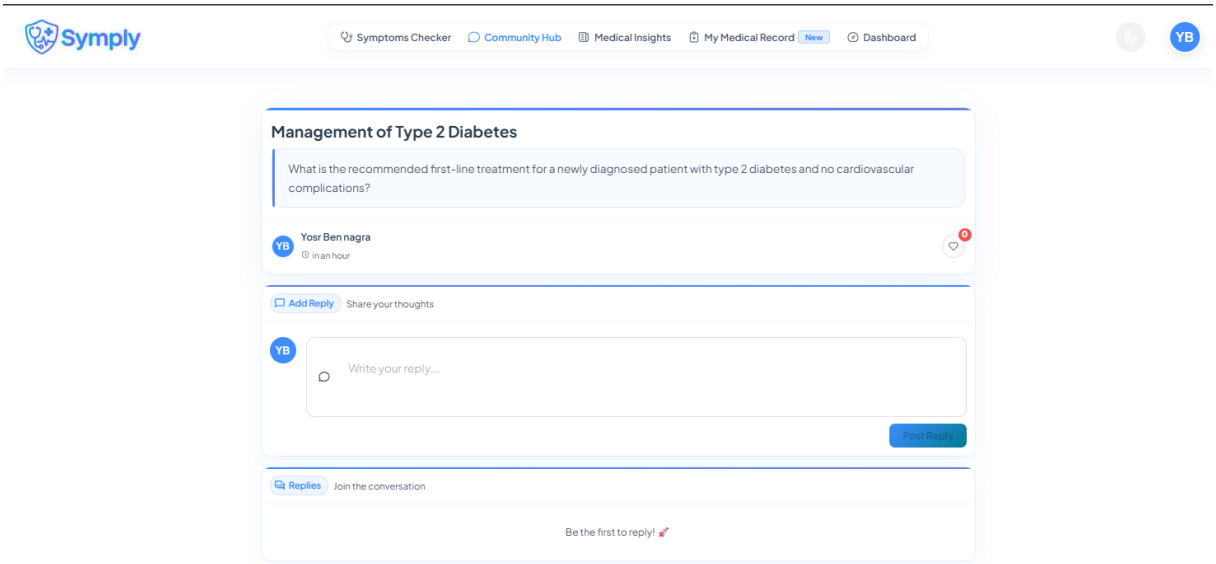


Figure 4.10: Question Details

**Nested Replies** Each reply can also receive comments (sub-replies), which are displayed directly beneath the original reply using an indented format. This structure creates clear conversation threads while preserving readability (example in Figure 4.11).

- Replies are sorted chronologically.
- Each nested reply includes the author's name and submission time.
- Users can reply to any reply, enabling multi-level threaded discussions for deeper conversations.

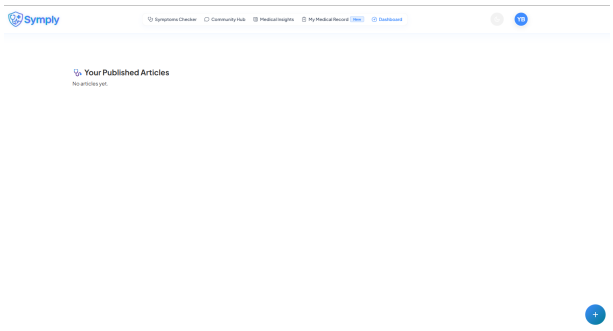


Figure 4.12: Doctor dashboard overview

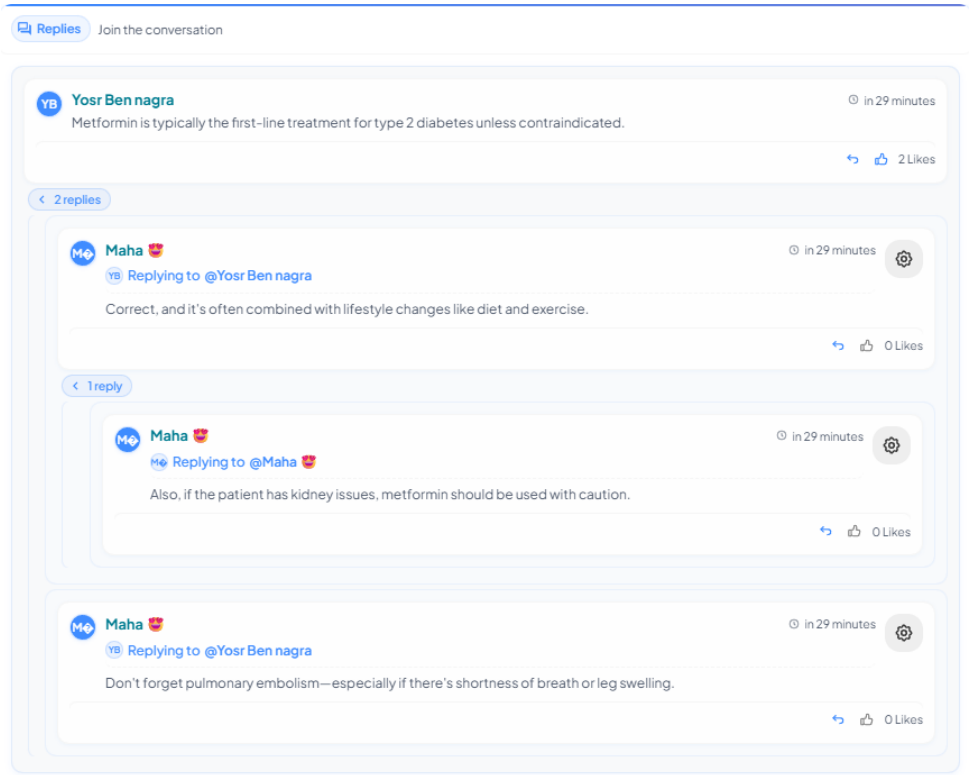


Figure 4.11: Nested Replies

### 4.2.2 Doctor Interfaces

Doctors can use the symptom checker and access dedicated dashboards and content tools.

#### Dashboard Overview

Upon logging in, doctors are directed to a personalized dashboard that summarizes their activity and grants access to doctor-specific modules such as article management and insights.

**Key Elements:**

- Overview of published articles (Figure 4.12).
- Quick access to article creation.

### Doctor Account Request

During the registration process, users who wish to access the Doctor Interface can request a Doctor account by enabling a dedicated option in the sign-up form.

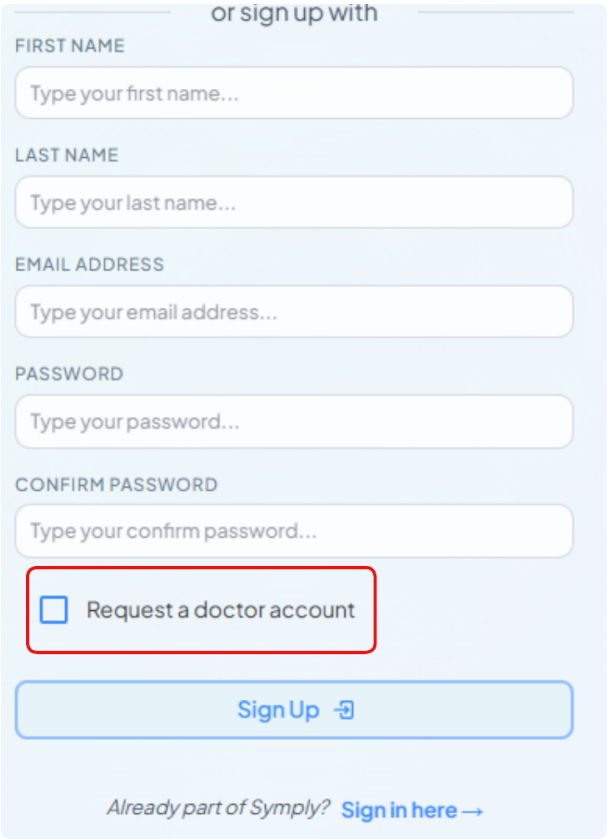


Figure 4.13: Doctor account request option during registration

### Request Flow

- When a user selects “Request a Doctor Account” during registration, their profile is flagged for admin review (see Figure 4.13).
- Until approved, the user is registered as a regular patient with limited access.
- Admins can view, accept, or reject pending doctor requests via the admin dashboard.

- Once approved, the user's role is updated to Doctor, granting access to exclusive features such as article creation.

This approval mechanism ensures that only verified medical professionals can contribute expert content, maintaining the reliability and credibility of the platform.

### 4.2.3 Admin Interfaces

Admins manage users, content, and medical data via a centralized dashboard.

#### Community Hub: Question Moderation

Administrators are granted moderation capabilities within the Community Hub, allowing them to directly manage user questions. These tools are seamlessly integrated into the same interface visible to standard users but include additional admin-only controls.

**Delete Question** Admins have the ability to remove any question that violates platform guidelines or community standards. This option appears as a dedicated icon next to each question, visible only to users with the `Admin` role (see Figure 4.14).

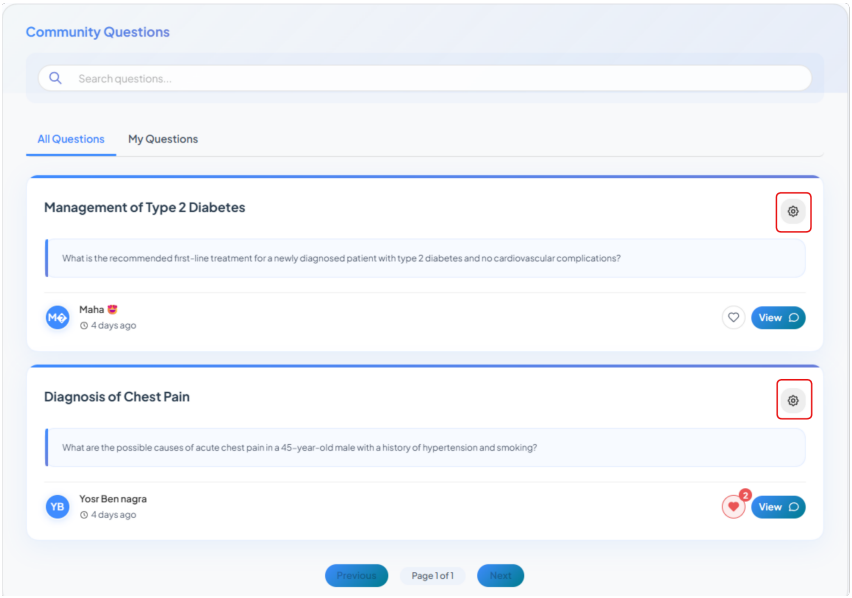


Figure 4.14: Admin moderation options for community questions

The visual design of the moderation icons and their functionality will be detailed separately in another section.

### Admin Dashboard: Medical Data Management

Beyond moderation, the admin dashboard provides complete control over the medical knowledge base used by the platform. This includes managing symptoms, body parts, and diseases, ensuring that the system remains adaptable, medically accurate, and up to date.

**Symptom and Body Part Management** Admins can add new symptoms and assign them to specific body parts. This mapping is essential for powering the interactive body map interface. When a user clicks on a region of the body, the associated symptoms are dynamically retrieved and displayed (see Figures 4.15 and 4.16 for examples).

- Add, delete or edit symptoms.
- Link each symptom to one or more body parts.
- Ensure anatomical accuracy and completeness of selectable symptoms.

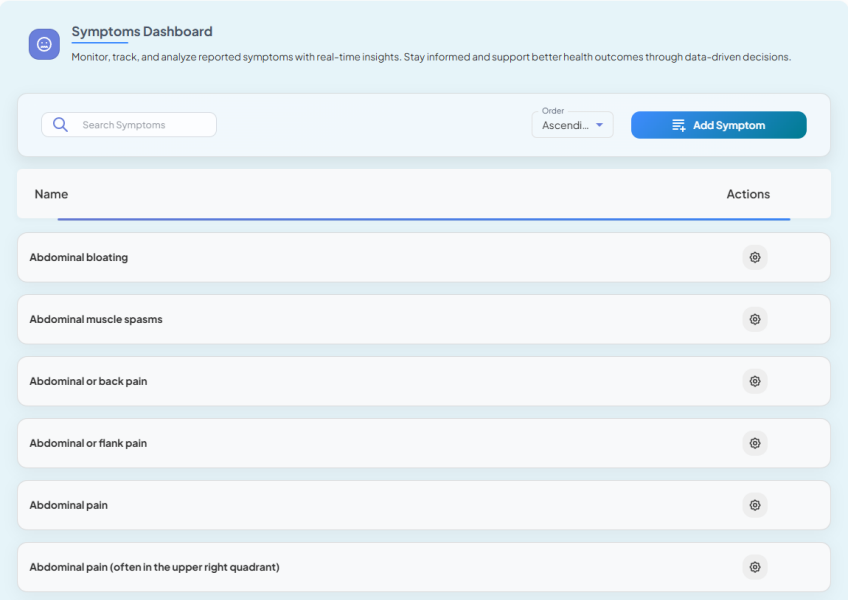


Figure 4.15: Symptom Dashboard

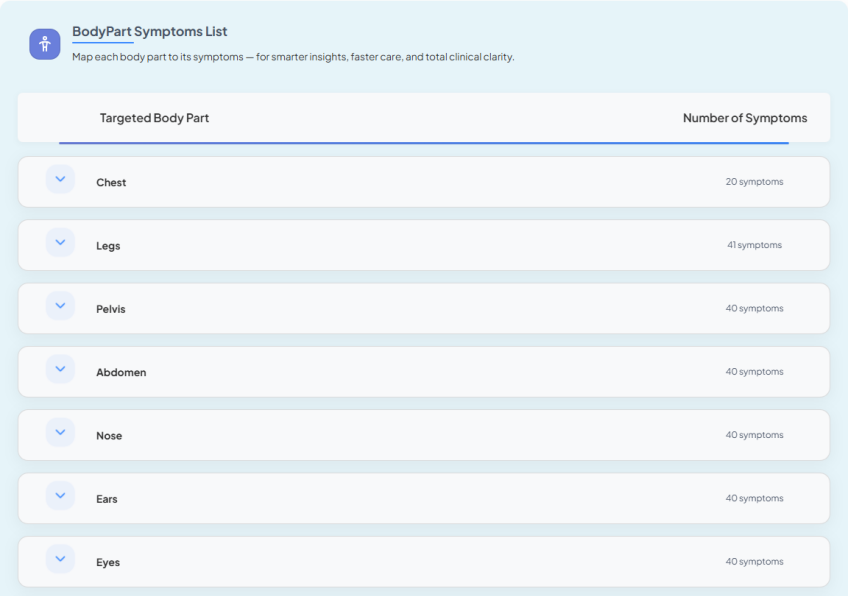


Figure 4.16: Admin interface for mapping symptoms to body parts

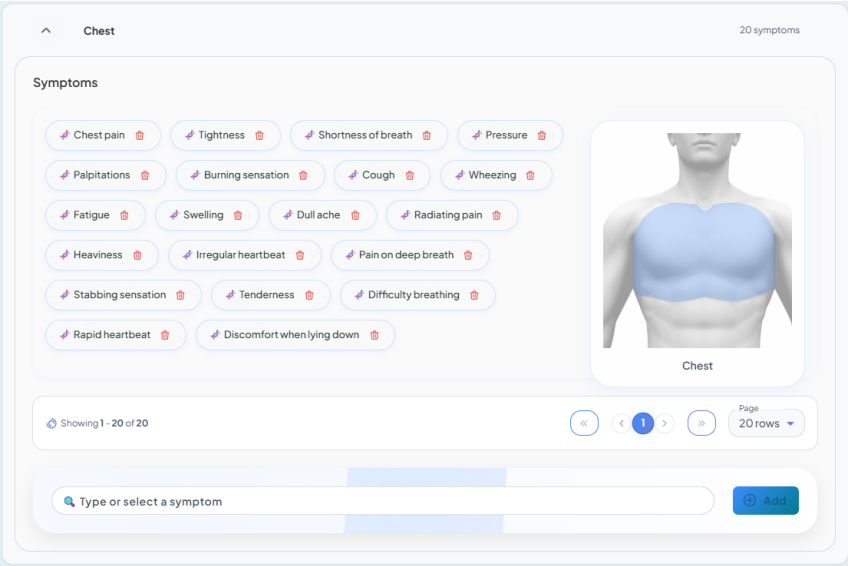


Figure 4.17: Admin interface for mapping symptoms to body parts Detail

**Disease Management and AI Enhancement** Admins also have the ability to create and manage diseases by associating them with relevant symptoms. This curated dataset directly improves the AI model's ability to generate accurate and context-aware predictions via the RAG (Retrieval-Augmented Generation) pipeline (see Figure 4.18).

- Define disease name, type, and associated symptoms.
- Enrich the backend knowledge base used for AI-assisted diagnosis.
- Expand the system's capability to handle rare or new conditions.

**Embedding New Diseases into the AI Model** After a new disease is created and linked with its relevant symptoms, the administrator can trigger an **embedding process** by clicking on the `Embed` button. This operation integrates the newly added disease into the vector store used by the Retrieval-Augmented Generation (RAG) engine.

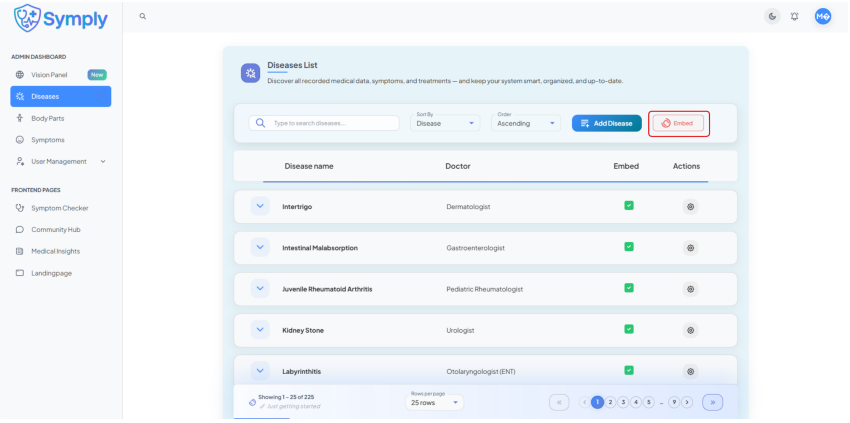


Figure 4.18: Diseases Dashboard

As a result, the AI model can retrieve and consider this new disease during symptom analysis, enabling dynamic updates without retraining the entire model. This feature significantly enhances the platform's flexibility, allowing medical knowledge to evolve in real time.

This powerful interface bridges the gap between user-facing tools and backend intelligence, giving administrators the ability to continuously refine the system's diagnostic accuracy and usability.

### User Management and Doctor Account Requests

The Admin Interface includes a dedicated section for managing users across the platform. This functionality allows administrators to maintain a healthy and secure user ecosystem by reviewing requests and moderating accounts when necessary.

**Doctor Account Requests** When a new user requests a `Doctor` role during registration, the admin is notified through two access points:

- A **notification badge** located in the top navigation bar, which highlights new pending doctor requests.
- A dedicated section in the admin dashboard labeled `Doctor Requests`, where the list of pending, approved, and rejected requests is displayed.

From there, the admin can review applicant details and choose to approve or reject each request (Figures 4.19 and 4.20). Once approved, the user's role is immediately updated to `Doctor`, granting them access to expert-level features like article creation and medical review.

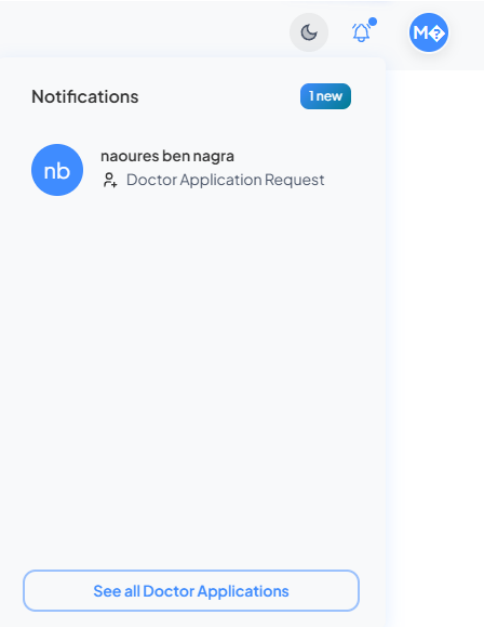


Figure 4.19: Doctor request notification

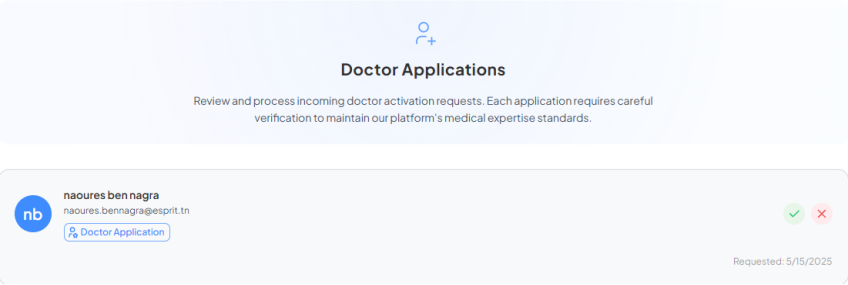


Figure 4.20: Doctor request dashboard

**User Account Management** Admins can also manage all user accounts from a centralized interface (Figure 4.21). For each user, the admin can:

- **Delete Account:** Permanently remove a user and all associated data.
- **Block/Unblock User:** Restrict access to the platform without deleting the account.

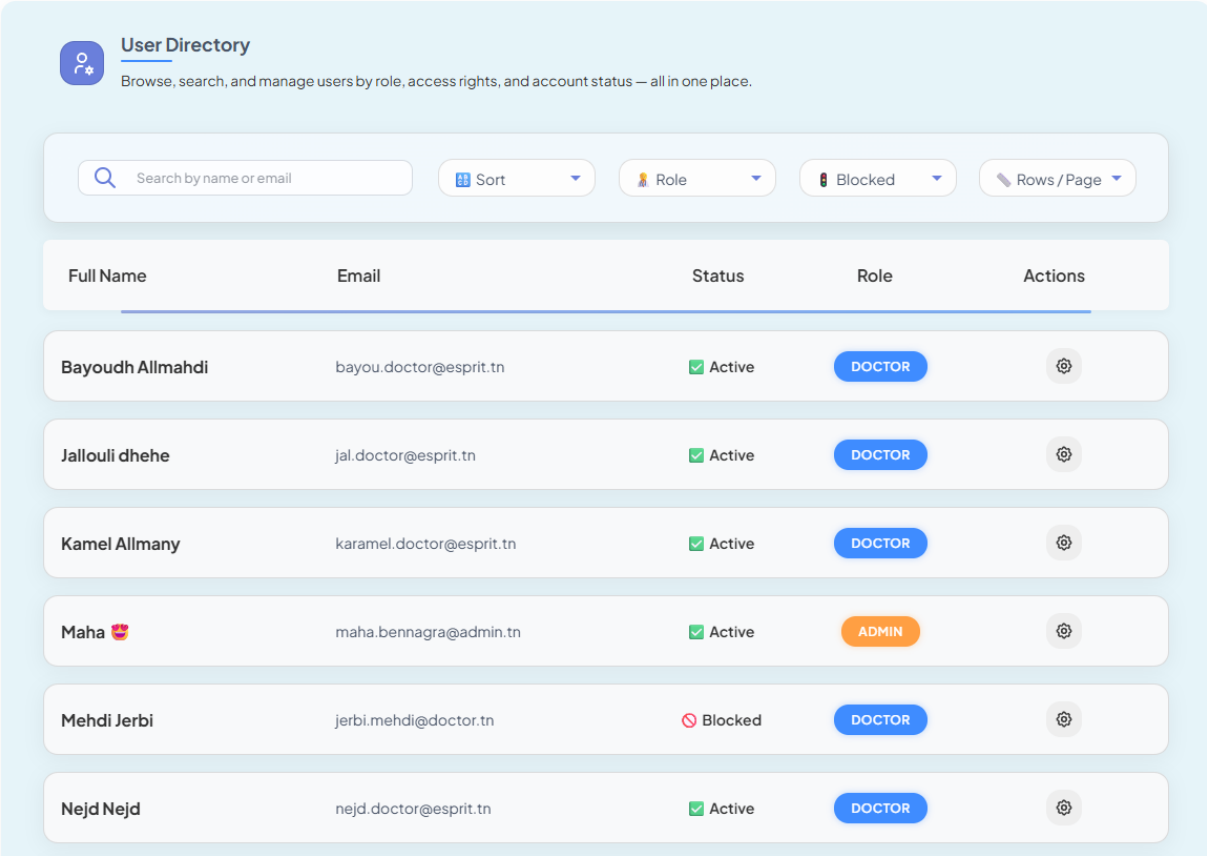


Figure 4.21: User Management

This administrative functionality ensures that the platform remains safe, organized, and efficiently governed, with real-time tools to handle all user-related operations.

## 4.3 UI/UX and Visual Identity

This section summarizes the main UX choices: consistent components, accessible contrast and semantics, and responsive layouts across devices.

**Design Stack** Frontend: `React` with `Material UI`, complemented by custom CSS, light/-dark themes, and SVG/icon sets.

### 4.3.1 Layout and Navigation

The application's layout is designed to offer a clean, organized, and responsive experience that adapts to different user roles and screen sizes. It uses a modular component structure to maintain consistency across pages while allowing for role-specific customization.

**Page Structure** Each page follows a consistent layout that includes:

- **Top Navigation Bar:** Contains the logo, main navigation links, and user-specific actions such as login/logout, notifications, and profile access (see Figure 4.22 for patient view and Figure 4.23 for admin view).

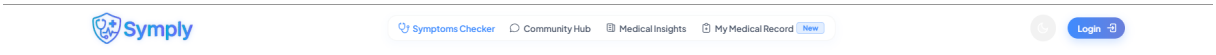


Figure 4.22: Top Navigation Bar


Figure 4.23: Top Navigation Bar for admins

- **Side Navigation (Dashboard):** For admins, a sidebar is available with categorized links to their respective modules (Figure 4.24).

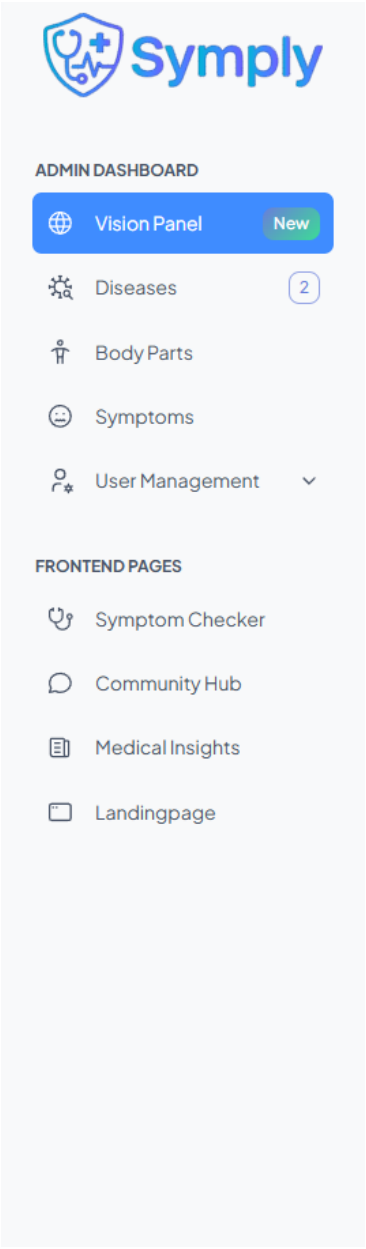


Figure 4.24: Side Navigation

- **Main Content Area:** Displays the core content dynamically based on the selected route, such as the symptom checker, medical articles, or admin panels.

**Navigation System** The application uses a route-based navigation system built with `react-router-dom`, allowing seamless transitions between views without full page reloads. Key features include (breadcrumbs illustrated in Figure 4.25):

- **Dynamic Routing:** Routes are structured and protected based on the user's role (e.g., patient, doctor, admin).

- **Breadcrumbs and Headings:** Each page includes a clear title and breadcrumb path for contextual awareness.

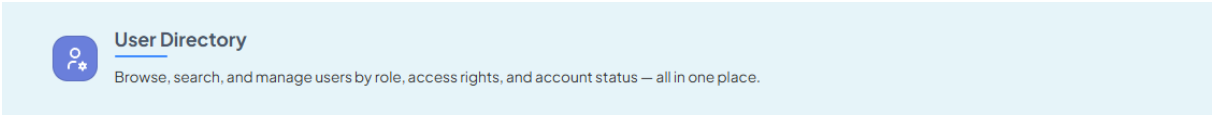


Figure 4.25: Breadcrumb Example

- **Protected Routes:** Sensitive routes are guarded by authentication checks and role-based access control.

### 4.3.2 Dark and Light Mode

To accommodate diverse user preferences and enhance accessibility, the application offers both **Light Mode** and **Dark Mode** interfaces. This feature ensures visual comfort across different lighting conditions and devices, while preserving the application's aesthetic identity.

#### Purpose and Benefits

- **Accessibility:** Improves readability for users with visual impairments or light sensitivity.
- **Energy Efficiency:** Reduces battery consumption on OLED and AMOLED devices in dark mode.
- **User Control:** Allows users to select the mode that best suits their environment or preference.

#### Implementation Details

- Built using `Material UI`'s theme system and React context.
- Colors, shadows, and backgrounds dynamically adapt based on the current theme.
- A toggle switch is placed in the user interface, enabling real-time switching between modes.
- All custom components respond to the selected theme without manual overrides.

**Design Considerations** Both modes were carefully crafted to maintain the platform's visual integrity:

- **Dark Mode:** Uses deep greys, muted blues, and desaturated accents to reduce eye strain in low-light environments.
- **Light Mode:** Relies on clean whites, soft backgrounds, and vibrant highlight colors to convey clarity and simplicity.

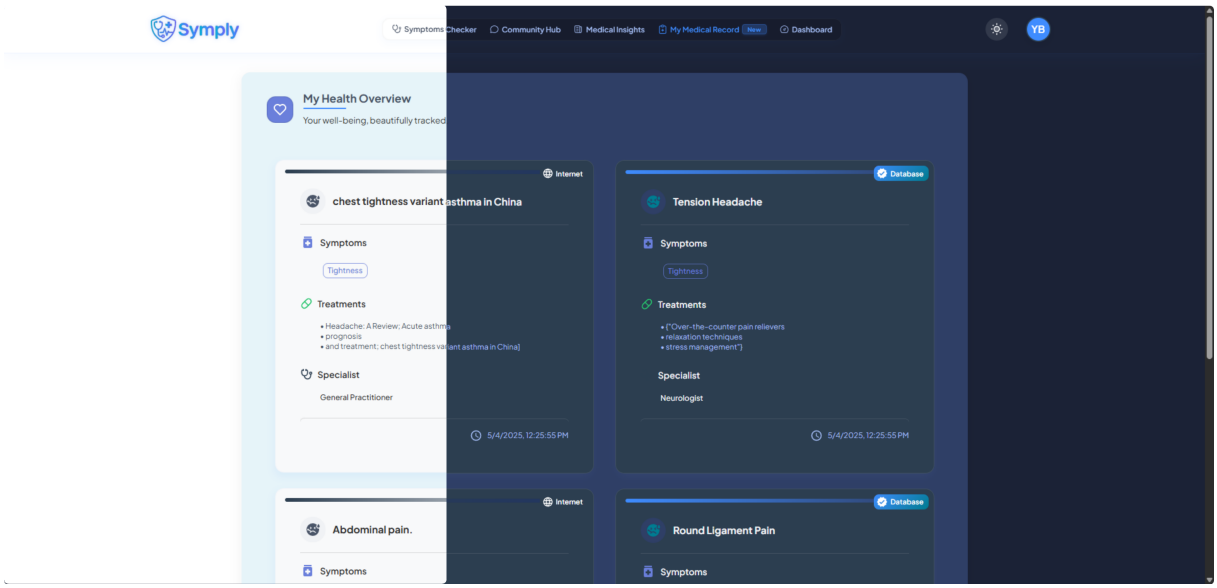


Figure 4.26: Light To Dark Mode Transition

By providing this theme flexibility, the application enhances usability, personal comfort, and modern design expectations across all user roles and usage contexts.

### 4.3.3 Input Design and Validation Feedback

Form interactions play a critical role in the user experience of the platform, especially given the variety of actions users can perform, from symptom submission to content creation and user registration. The application ensures input clarity and error handling through intuitive visual cues and consistent interaction patterns.

#### Form Input Design

- Input fields are styled using `Material UI`, ensuring consistency in spacing, label placement, and focus states.

- Placeholder text and floating labels help guide users while preserving space.
- Fields are responsive and adapt across screen sizes.

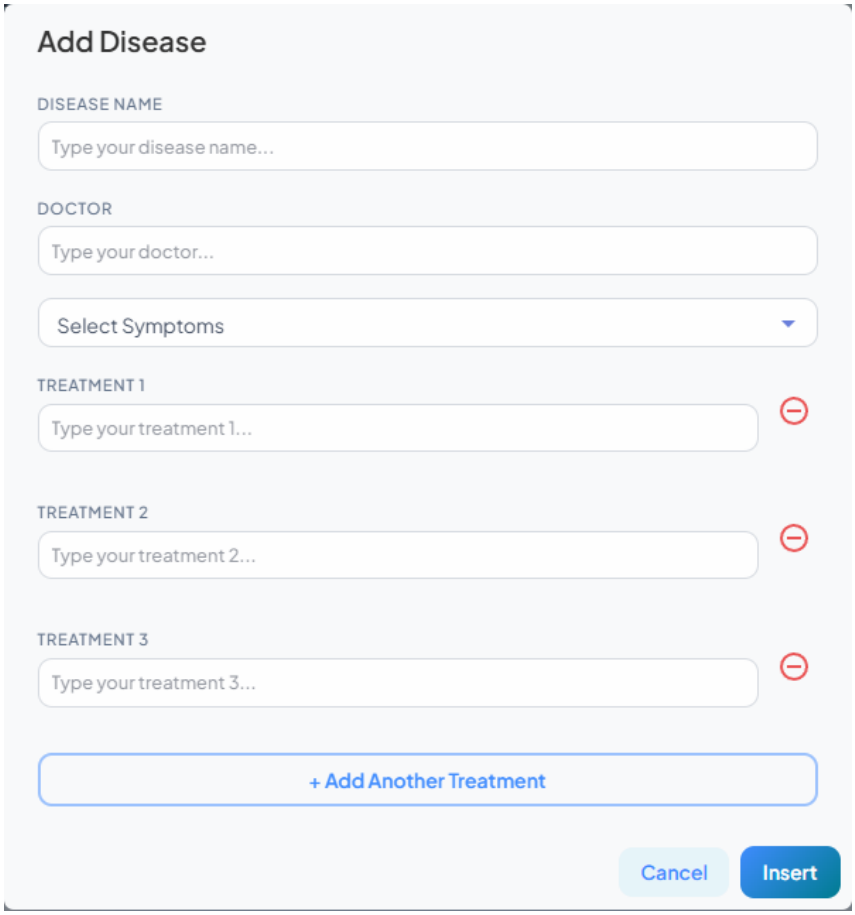


Figure 4.27: Inputs Example

### Validation Feedback

- **Inline Error Messages:** Displayed immediately below fields when validation fails (e.g., empty required field, invalid email format).

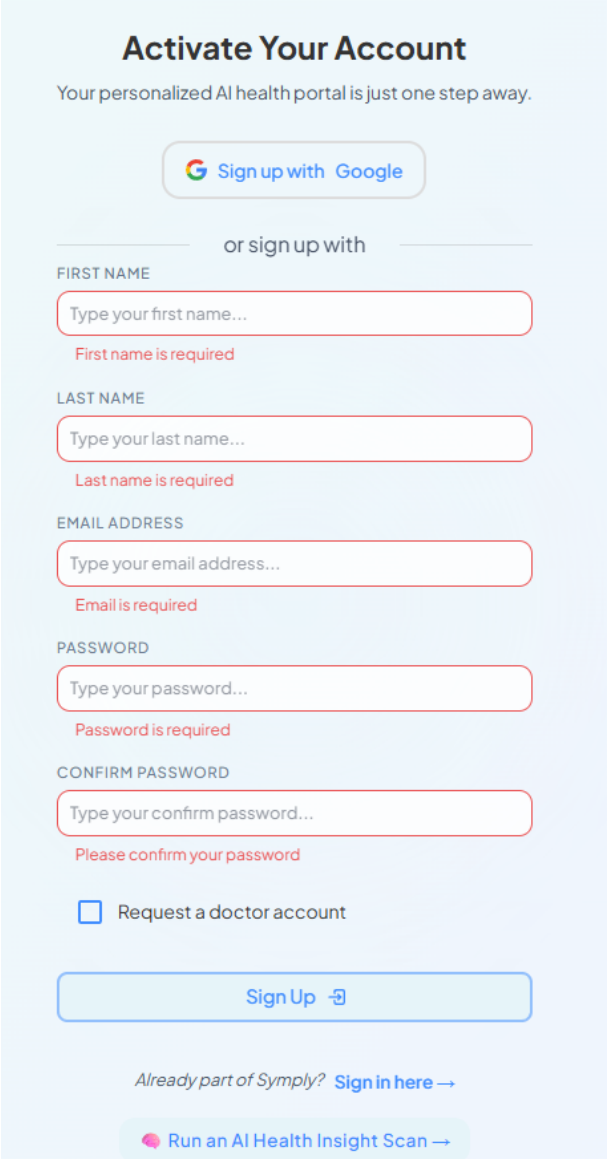


Figure 4.28: Example of validation and inline error messaging

**Settings Icon and Contextual Actions** In many admin and doctor interfaces, interactive tables or lists include a compact `settings icon` (usually represented by a gear or three dots). When clicked, this icon reveals a contextual menu offering relevant actions:

- **Edit:** Opens a modal or drawer to update the entry (e.g., edit a symptom, update user details).
- **Delete:** Prompts a confirmation before removing the item.
- **View Details:** Navigates to a detailed view when applicable (e.g., full article or user profile).

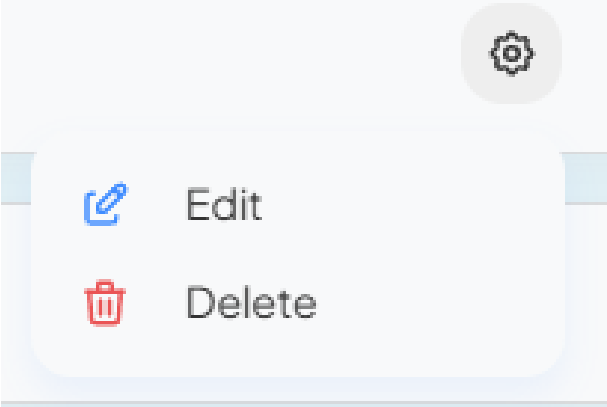


Figure 4.29: Settings Icon

This approach keeps the interface clean and task-focused while giving users direct control over each item.

### 4.3.4 Interactive Tutorials and User Guidance

To ensure that users, especially first-time visitors, can easily understand how to use the symptom checker and its advanced features, the platform includes a built-in interactive tutorial system.

#### Purpose of the Tutorial

- **Onboarding Support:** Helps new users discover features like symptom selection, zoom controls, skin symptom toggles, and more.
- **Contextual Awareness:** The tutorial appears only when necessary (e.g., first visit or upon user request), avoiding unnecessary distraction.
- **Improved Engagement:** Users are more likely to explore and benefit from the platform when features are clearly explained.

**Accessing the Tutorial** Users can start the tutorial at any time by clicking the **Help** icon, represented by a subtle question mark button located in the lower section of the control sidebar (Figure 4.30).

Figure 4.30: Tutorial activation icon

**Step-by-Step Highlights** Once triggered, the tutorial overlays the interface with guided tooltips that explain each control's function. For instance, when introducing the skin symptoms panel, a step appears as shown below (Figure 4.31).

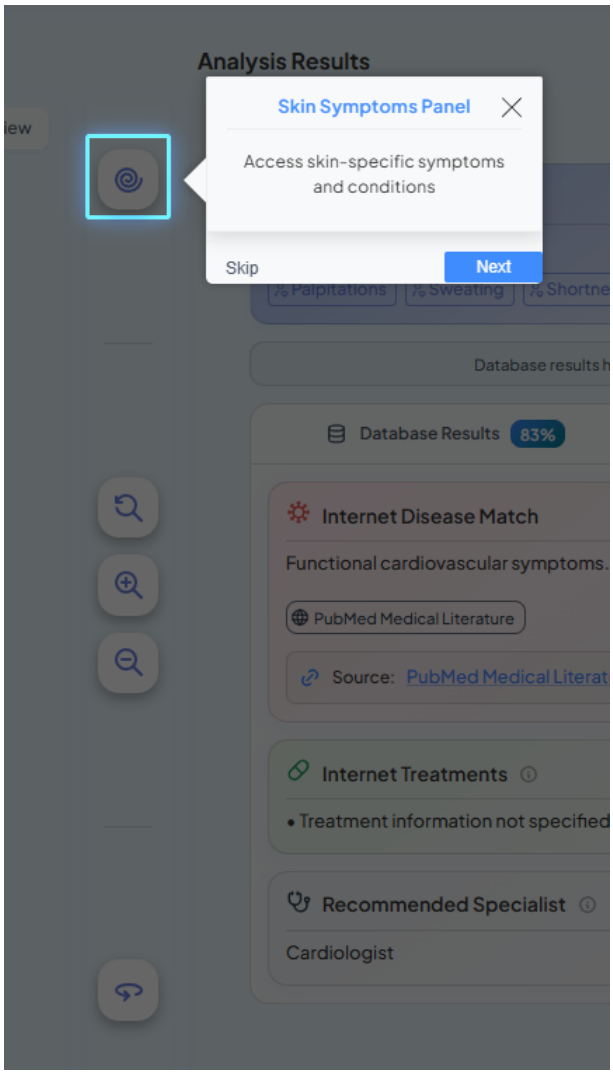


Figure 4.31: Tutorial

In summary, the AI model is embedded in a role-based web app with consistent UX and clear workflows. The next chapter details DevOps practices that support deployment, monitoring, and scalability.

CHAPTER 5

# DevOps, CI/CD, and Infrastructure

## Contents



This chapter presents the DevOps practices and infrastructure setup that support the deployment and maintenance of the application. It covers the integration of automation tools, including Docker for containerization, Jenkins and GitHub for CI/CD pipelines, and SonarQube for code quality checks.

Monitoring and logging are also addressed through the use of Prometheus and Grafana, ensuring observability and system reliability. The chapter concludes with an overview of the deployment strategy, including staging and production workflows.

## 5.1 Introduction to DevOps in the Project

### 5.1.1 Objectives of DevOps Integration

The integration of DevOps in this project was motivated by the need to streamline the development lifecycle and improve the reliability of deployments. The main objectives are summarized as follows:

- **Improve the Development Workflow**: Initially, the development process was fragmented, requiring manual intervention during testing and deployment. The introduction of GitHub for version control and Jenkins for automation significantly enhanced collaboration and productivity. Developers now push code to GitHub, which triggers automated pipelines, reducing manual steps and errors.
- **Ensure Reliability and Repeatability**: A consistent and automated CI/CD pipeline was implemented using Jenkins. The pipeline is triggered via a GitHub webhook, tunneled through Ngrok to a Jenkins container. Upon each push, the pipeline executes a structured sequence: *checkout* $\rightarrow$ *build Docker images* $\rightarrow$ *run tests* $\rightarrow$ *perform SonarQube analysis* $\rightarrow$ *deploy with Docker Compose* $\rightarrow$ *verify containers*.
- **Accelerate Delivery and Feedback**: By automating build and test stages, feedback on code quality and deployment readiness is delivered in minutes. SonarQube integrates into the pipeline to analyze code quality and coverage, while successful builds are immediately deployed using Docker. This allows faster iteration and continuous delivery of updates.
- **Optimize Resource Usage**: Backend services leverage lightweight Docker images, and external resources like MongoDB Atlas and Hugging Face are used to offload infrastructure burden. This minimizes image size and ensures rapid container startup, contributing to an efficient deployment process.
- **Verification and Notification**: After deployment, the pipeline verifies that all containers (backend, frontend, database, exporters) are operational. Logs are inspected to confirm successful starts. Test results and coverage reports are archived, and notification emails are sent automatically.

### 5.1.2 Tools and Platforms Used

The project integrates several tools and platforms to streamline development, deployment, monitoring, and delivery. The following table summarizes each one and its role:

| Tool / Platform | **Description** |
|---|---|
| Docker | Containerization platform used to package the backend, frontend, and services into portable containers. |
| Docker Hub | Remote registry used to store and share Docker images. |
| Flask (Backend API) | Python microframework used to implement the REST API that serves AI inference and business logic. Runs behind Gunicorn in production. |
| React (Frontend) | JavaScript library used to build the single-page application (SPA) for patient, doctor, and admin interfaces. Built with Vite and served via Nginx. |
| PostgreSQL (Container) | Relational database deployed as a Docker container. |
| Jenkins | Automation server managing the CI/CD pipeline with build, test, and deployment stages. |
| GitHub | Version control and collaboration platform used to trigger Jenkins pipelines via webhooks. |
| SonarQube | Static code analysis tool integrated with CI to ensure code quality and security. |
| Prometheus | Time-series monitoring system used to collect metrics from services and containers. |
| Grafana | Dashboard tool used to visualize performance metrics collected by Prometheus. |
| Node Exporter | Prometheus exporter that gathers machine-level metrics (CPU, memory, etc.). |
| Postgres Exporter | Prometheus exporter that collects database metrics from PostgreSQL. |
| Ngrok | Secure tunneling service exposing local Jenkins to the internet for GitHub webhook integration. |
| Nginx | Web server and reverse proxy used to route frontend/backend traffic and serve static files. |
| Gunicorn | WSGI HTTP server used to run the Python backend app in production. |

Table 5.1: DevOps Tools and Platforms Used in the Project

### 5.1.3 Pipeline Workflow and Webhook Integration

To ensure continuous integration and delivery, a webhook-based pipeline was implemented. The process starts from a code push to GitHub and ends with a full deployment of the application

using Docker Compose. The Jenkins server is not exposed publicly, so an **Ngrok tunnel** is used to forward GitHub webhook traffic to Jenkins securely.

The pipeline is composed of the following stages:

1. **GitHub Push**: Developers push code changes to a remote repository.
2. **Webhook Trigger**: A GitHub webhook is configured to send a payload on every push.
3. **Ngrok Tunnel**: The webhook payload hits a public Ngrok URL, which securely tunnels the request to the local Jenkins server.
4. **Jenkins Execution**:
    - Checkout code from GitHub
    - Start supporting services (e.g., PostgreSQL, SonarQube)
    - Run backend unit tests using Pytest in a Docker container
    - Perform code analysis via SonarQube
    - Clean previous containers and prepare the workspace
    - Deploy backend and frontend via Docker Compose
    - Verify containers, logs, and services
5. **Post Actions**:
    - Generate test and coverage reports
    - Send email notifications

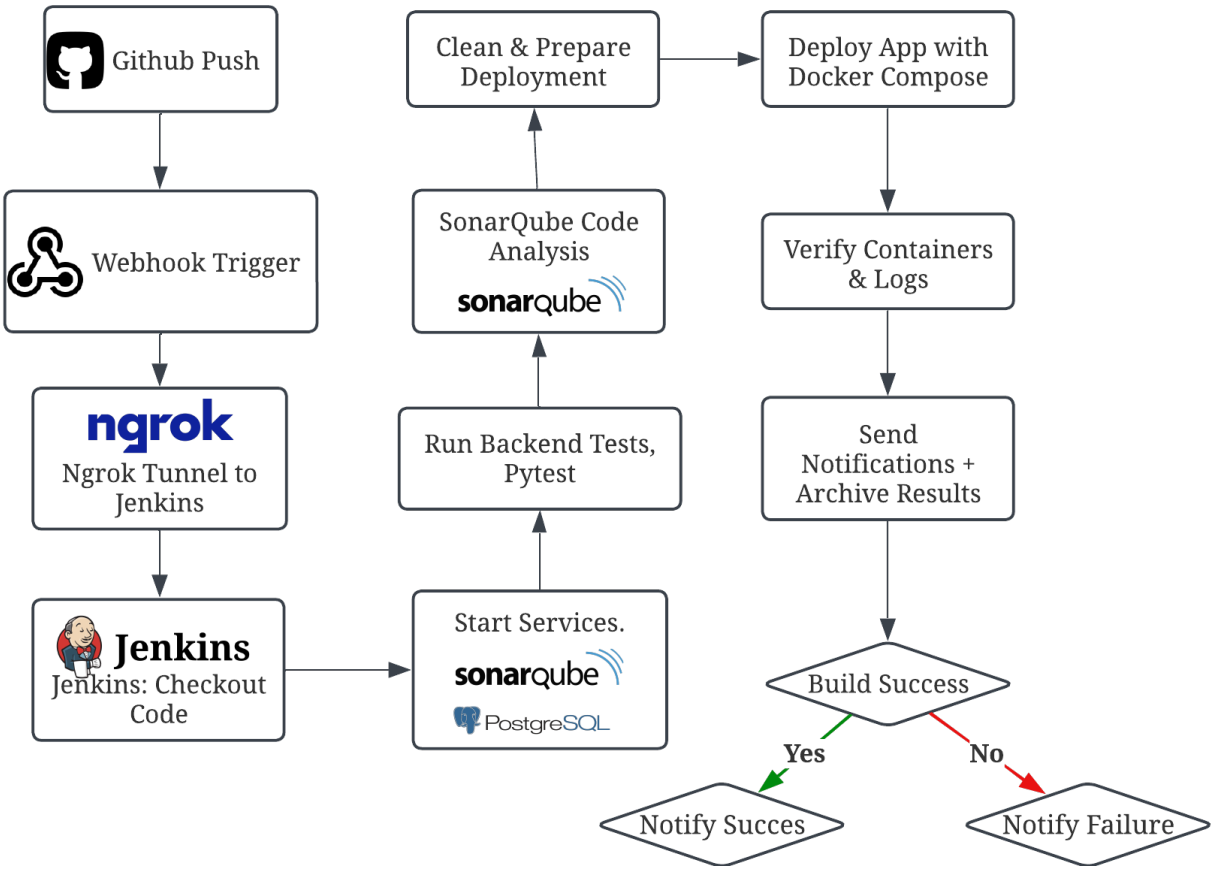


Figure 5.1: DevOps Workflow

## 5.2 Containerization and Environment Setup

### 5.2.1 Docker Usage

Docker helped make both development and deployment more consistent and easier to manage. All main parts of the project (backend, frontend, database, and monitoring tools) were put into containers to avoid setup problems and ensure the same environment everywhere.

- **Development:** Docker made it easy to copy the environment quickly and handle dependencies without issues.
- **Production:** Docker Compose was used to run all services together in a reliable and repeatable way.
- **Benefits:** This setup improved CI/CD integration, made monitoring and debugging easier with container logs, and sped up delivery by using ready-made images from Docker Hub.

### 5.2.2 Dockerfile and Docker Compose

To manage the different services of the application, Docker Compose was used to define and run multiple containers together. Each service has its own role and runs in a separate container. Dockerfiles were used to control how each image is built, including dependencies and startup commands.

**Overview of Service Containers:**

- **frontend** – A React application built with Node.js and served using an Nginx container.
- **backend** – A Flask API that runs with Gunicorn in production; connected to PostgreSQL, MongoDB Atlas, and includes AI model integration.
- **postgres** – A PostgreSQL 16 database for storing backend data.
- **jenkins** – CI/CD server used to automate building, testing, and deploying.
- **sonarqube** – Analyzes source code and provides quality and security reports.
- **prometheus** – Collects metrics from the containers and exporters.
- **grafana** – Displays metrics from Prometheus in customizable dashboards.
- **node-exporter** – Provides system metrics (CPU, memory) to Prometheus.
- **postgres-exporter** – Provides PostgreSQL performance metrics.

**Dockerfile Highlights:**

- The `backend Dockerfile` installs required libraries (including PyTorch) and runs the app with Gunicorn in a clean environment.
- The `frontend Dockerfile` builds the React project and serves it through Nginx in a lightweight container.

This setup makes the system modular, portable, and easier to develop, test, and monitor.

## 5.3 Continuous Integration and Continuous Deployment (CI/CD)

### 5.3.1 CI/CD Goals

The CI/CD pipeline was implemented to automate the entire application delivery lifecycle, from code integration to deployment. This ensures that every change is automatically built, tested, and deployed in a consistent and reliable manner.

By integrating automated tests, static code analysis, and containerized deployment, the pipeline provides immediate feedback and helps maintain code quality.

This approach reduces manual errors, shortens release cycles, and promotes resilience by catching issues early and enforcing best practices throughout the development process.

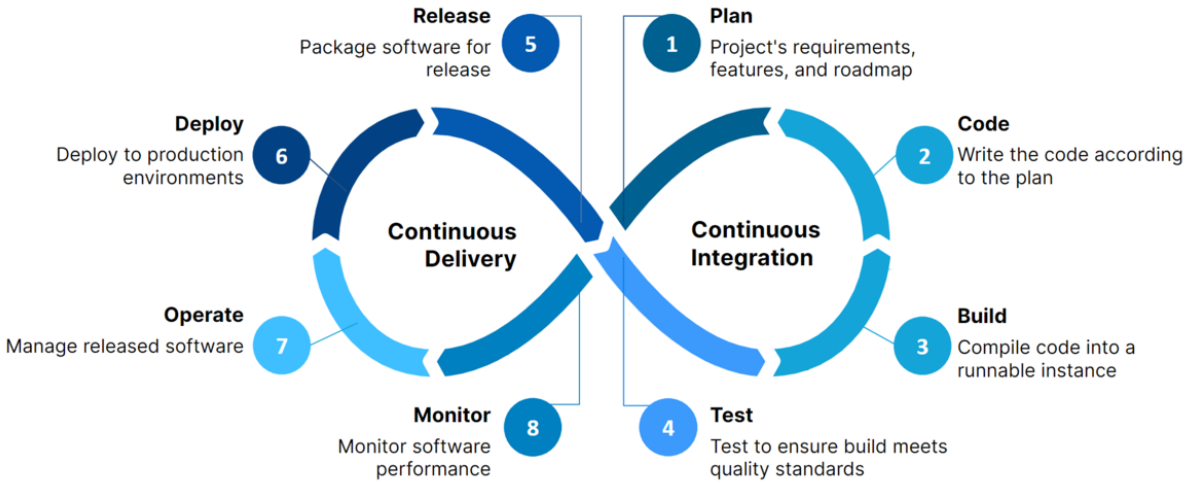


Figure 5.2: CI/CD Logo

### 5.3.2 Jenkins Pipeline

Jenkins was used as the backbone of the CI/CD process in this project, automating all critical steps from code checkout to deployment. The pipeline is declaratively defined in a `Jenkinsfile` and is triggered automatically through a GitHub webhook integrated via Ngrok, enabling Jenkins to react to every push event.

The pipeline runs inside a containerized Jenkins environment and is composed of multiple structured stages, each with a specific responsibility:

- **Checkout:** Retrieves the latest code from the GitHub repository and confirms the project structure.

- **Start Supporting Services:** Initializes dependencies such as PostgreSQL, SonarQube, and monitoring exporters using Docker Compose. The pipeline includes a health check loop to ensure SonarQube is fully up before proceeding.
- **Run Backend Tests:** Executes unit tests for the backend using Pytest inside a dedicated Python Docker container. Code coverage is also collected at this stage.
- **SonarQube Analysis:** Integrates with SonarQube to perform static code analysis and generate quality and security reports based on the backend source code and test coverage.
- **Clean Up:** Stops and removes any containers from previous runs and resets the deployment directory, ensuring a clean and predictable build environment.
- **Prepare:** Copies all application files to a dedicated deployment directory and verifies the presence and permissions of required folders (e.g., logs, uploads).
- **Deploy:** Builds fresh Docker images for the backend and frontend and starts all application containers, including supporting services, using Docker Compose. Logs and statuses are collected to verify successful deployment.
- **Verify:** Performs runtime checks to ensure that backend, frontend, and exporters are running correctly. It also captures logs and container states for further inspection.

In the `post` section of the pipeline, Jenkins automatically stores test results, reports code coverage, and sends email notifications about the build status to the development team. This fully automated pipeline ensures that every change is tested, analyzed, and deployed in a reproducible and reliable manner.

### 5.3.3 GitHub Workflow Integration

The project uses GitHub as the central platform for source code management and collaboration. To maintain code quality and stability, a structured workflow was adopted based on branches, pull requests, and automatic pipeline triggering.

**Branch Strategy:** The main development process follows a feature-branching model. Developers create dedicated branches for each feature or fix, derived from a stable base branch such as `main` or `dev`. This promotes isolation of changes and simplifies code review.

**Pull Requests (PRs):** When a feature is completed, a pull request is opened to merge changes back into the base branch. The pull request serves as a checkpoint for code review, discussion, and automated validation.

**Pipeline Triggering:** Each push to a branch and every pull request triggers a CI/CD pipeline through a GitHub webhook. This webhook notifies Jenkins via a secure Ngrok tunnel, allowing

it to automatically start the pipeline. As a result, builds, tests, and quality checks are performed as part of the development workflow, ensuring that no code is merged without validation.

This GitHub-based workflow reinforces collaborative development, encourages best practices, and integrates tightly with the CI/CD process to maintain a stable and efficient delivery pipeline.

### 5.3.4 SonarQube Integration

To keep the code clean and easy to maintain, SonarQube was added to the CI/CD pipeline as a static code analysis tool. It automatically checks the backend code every time the pipeline runs.

**Code Quality Reports:** SonarQube reviews the code using predefined rules and highlights issues such as duplicated code, code smells, and poor practices. Each pipeline run generates a report that can be viewed on the SonarQube dashboard.

**Security Scanning:** SonarQube also checks for security risks, such as unsafe API usage, SQL injection, or hardcoded secrets. This helps detect problems early before deployment.

**Test Coverage:** Unit tests are run using Pytest with the `pytest-cov` plugin, and the coverage results are sent to SonarQube. This shows which parts of the code are well tested and which need more attention.

The integration is managed in the Jenkins pipeline using the `sonar-scanner` tool. Code analysis runs only after the tests pass, ensuring quality-first development.

Overall, SonarQube helps maintain high coding standards, improves long-term maintainability, and supports writing more secure and reliable software.

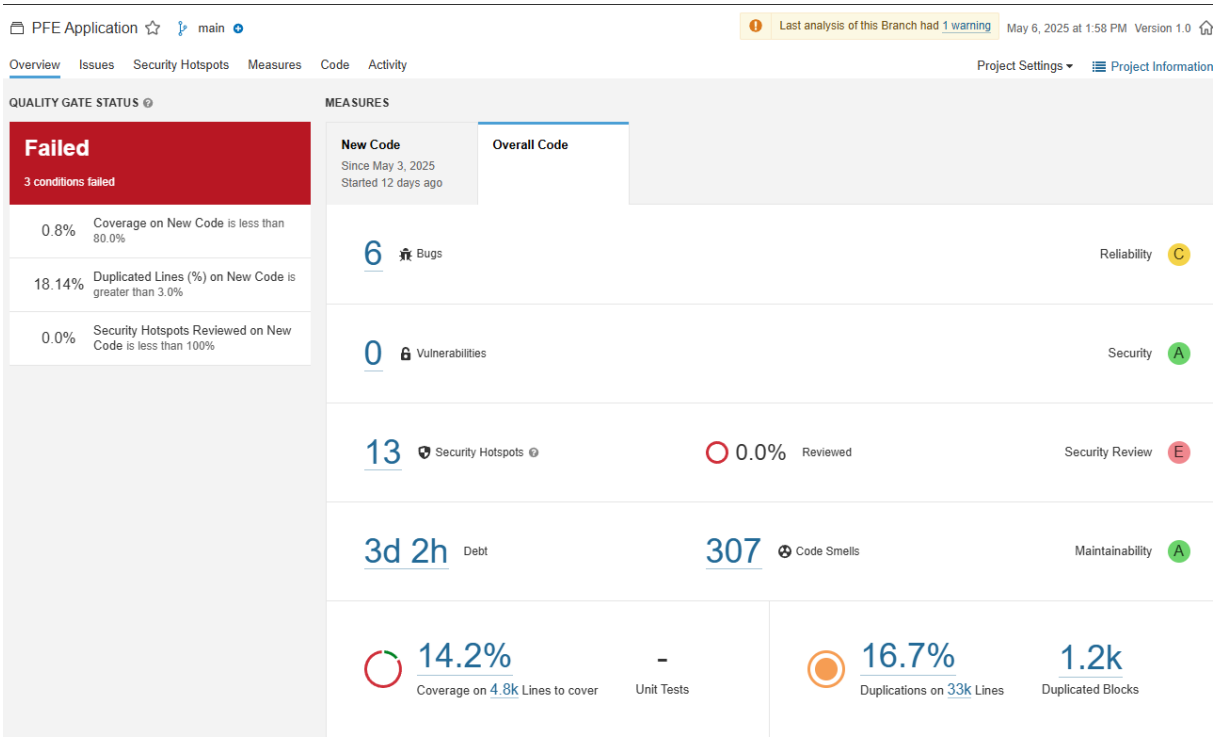


Figure 5.3: SonarQube Dashboard

The figure above shows the SonarQube dashboard with key metrics such as bugs, vulnerabilities, code smells, and test coverage. This visual report makes it easy to track issues and monitor improvements over time, making quality assurance a continuous process.

## 5.4 Infrastructure Monitoring and Logs

### 5.4.1 Why Monitoring Matters

Monitoring plays a crucial role in maintaining the reliability and performance of modern applications. In this project, observability was integrated into the infrastructure to detect issues early, optimize resource usage, and ensure continuous uptime.

By actively tracking system metrics, service health, and application behavior, the development team can respond quickly to unexpected behaviors, performance degradation, or resource exhaustion before they impact end-users. Monitoring also facilitates post-incident analysis, helping identify root causes and implement preventive measures.

In a containerized environment, where multiple services interact dynamically, real-time insights are essential to maintain operational stability. Effective monitoring ensures that deployments are verifiable, resource utilization is transparent, and performance is continuously measurable, ultimately supporting a robust and user-centric application.

### 5.4.2 Prometheus and Grafana

To monitor the application's infrastructure and performance metrics, Prometheus and Grafana were integrated into the Dockerized stack.

**Prometheus** acts as the metrics collection engine. It scrapes data from various exporters such as Node Exporter (for system-level metrics) and Postgres Exporter (for database-specific statistics). These metrics include CPU usage, memory consumption, disk I/O, database connections, and query performance, among others. Prometheus stores this time-series data and provides a powerful query language (PromQL) for extracting meaningful insights.

**Grafana** serves as the visualization layer on top of Prometheus. It was configured with custom dashboards to monitor the health and activity of key services. The dashboards offer real-time views of server load, container uptime, PostgreSQL performance, and other operational indicators.

This monitoring stack enables proactive system administration. Developers and operators can detect anomalies, investigate performance bottlenecks, and maintain consistent system availability.

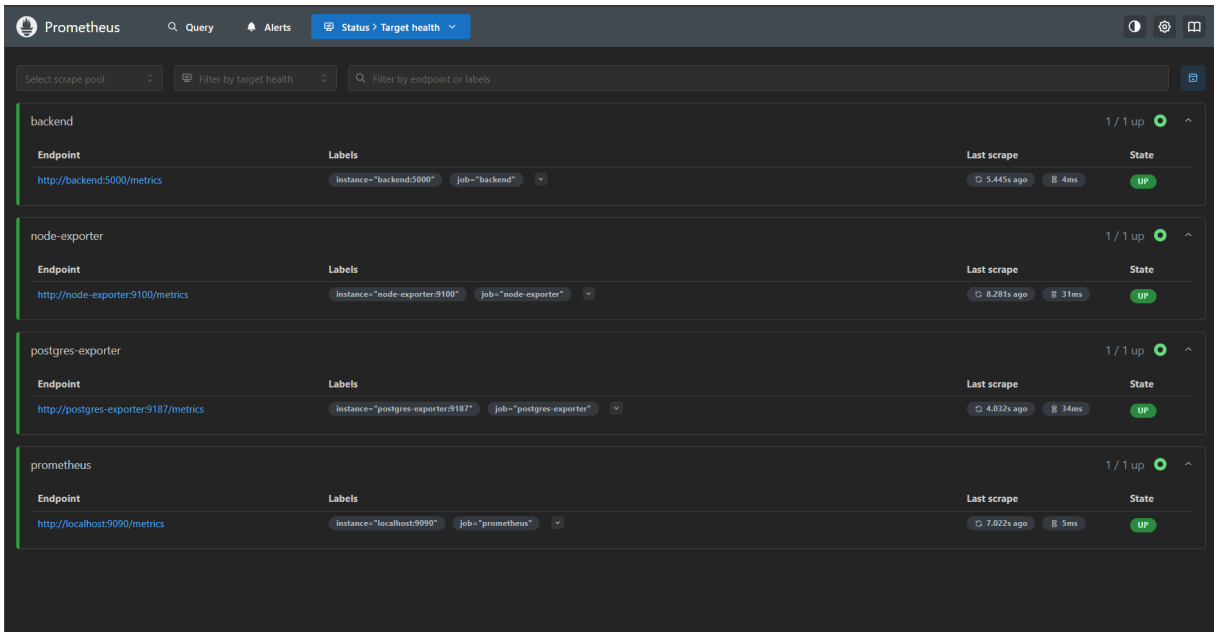


Figure 5.4: Prometheus Interface Showing Active Monitoring Targets

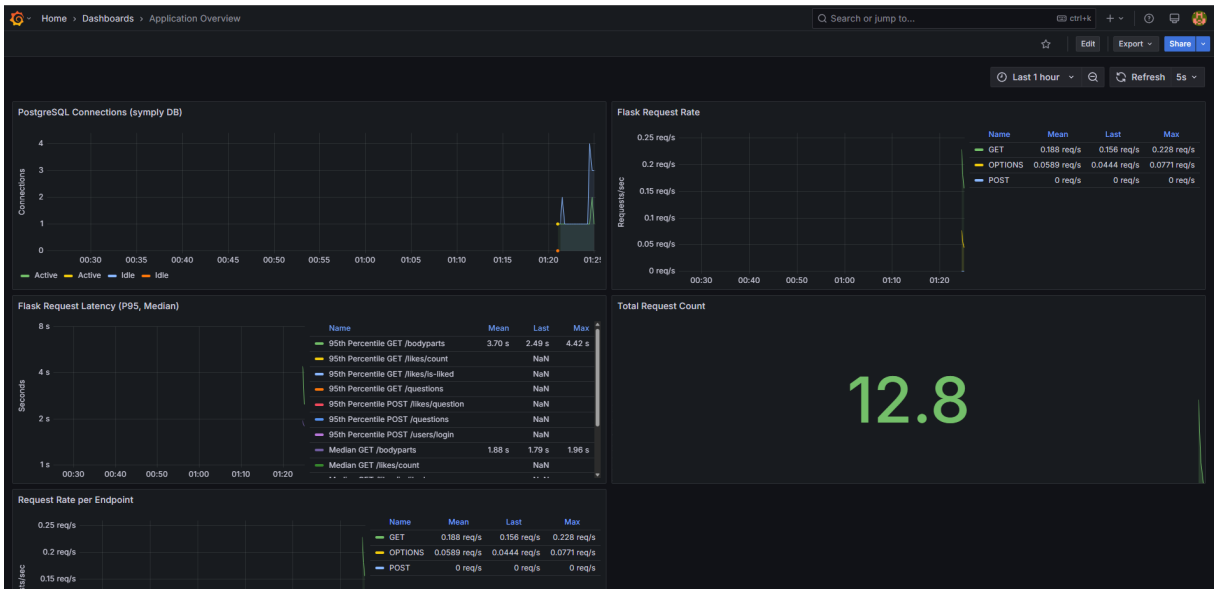


Figure 5.5: Grafana Dashboard Visualizing Prometheus Metrics

### 5.4.3 Logging Strategy

A structured logging system was implemented in the backend to support observability, debugging, and operational auditing. All application logs are stored in a dedicated `logs/` directory within the backend container and persisted across runs.

Two separate log files are maintained:

- **app.log** – General information and runtime events (e.g., startup messages, API actions)
- **error.log** – Critical errors and exception traces captured at the `ERROR` level or higher

The logging system is built using Python's `logging` module with `RotatingFileHandler` to manage log file size and backups. This ensures long-running services do not accumulate unbounded logs, while still retaining historical data for analysis.

Each log entry includes a timestamp, severity level, source module, and descriptive message. The error log additionally records the source file and line number for precise debugging.

This logging strategy provides essential insight into system behavior and supports efficient error tracking and post-mortem analysis during development and production.

## 5.5 Deployment Strategy

### 5.5.1 Staging vs. Production

The project adopts a clear separation between staging and production environments to ensure stability, security, and safe testing before live deployment.

**Staging Environment:** The staging environment is used for development and internal testing. It mirrors the production architecture as closely as possible, including containerized services such as the backend (Flask), PostgreSQL, and frontend (React) served through Nginx. However, it operates using test data, verbose logging, and development-oriented configurations. This setup allows developers to validate new features and fixes without affecting real users.

**Production Environment:** The production environment serves end users and prioritizes performance, security, and reliability. It runs the same Docker containers built during the CI/CD process, but with stricter configurations: debugging is disabled, logging is minimized, and only essential services are exposed. Both the frontend and backend are served through Nginx, with the frontend deployed via Netlify with HTTPS by default.

**Isolation and Configuration:** Environment-specific variables are managed via separate `.env` files or injected through the CI/CD pipeline. These variables define database URIs, secret keys, hostnames, and service modes (e.g., `DEBUG=True` in staging, `False` in production). This separation ensures that changes in one environment do not impact the other and allows for secure testing workflows.

**Secrets Management:** Sensitive data such as JWT secret keys, database credentials, and API tokens are never hardcoded. Instead, they are stored securely as environment variables and passed to containers at runtime. This practice mitigates the risk of credential leakage and aligns with DevSecOps best practices.

By maintaining this strict separation and managing configuration through environment variables, the project ensures safe deployment practices and production-grade operational hygiene.

### 5.5.2 Deployment Flow

The deployment process is orchestrated through Jenkins and defined in a declarative `Jenkinsfile`. Upon pushing code to GitHub, a webhook triggers the pipeline, which automates the steps from build to deployment.

**1. Pipeline Trigger:** Each code push to the GitHub repository activates a webhook that tunnels into the Jenkins container through Ngrok, initiating the CI/CD pipeline.

**2. Image Build and Test:** The pipeline builds the backend and frontend Docker images, runs backend unit tests, performs SonarQube analysis, and verifies the application structure and logging directories.

**3. Service Initialization:** Before deployment, the pipeline ensures that any existing PostgreSQL or backend containers are removed to avoid conflicts. It then starts a fresh PostgreSQL container and waits for it to become ready.

**4. Docker Compose Deployment:** The backend and frontend services are rebuilt using Docker Compose and launched in detached mode. Logs for both services are printed to ensure visibility during the deployment process.

**5. Local Container Deployment:** At this stage, the application is fully deployed and running locally within the Jenkins host. Although images are built and services are live, the deployment is not yet pushed to a remote production environment. This approach enables testing and validation in a controlled environment and serves as a foundation for future production rollout.

This local-first deployment model allows full CI/CD verification while preparing the infrastructure for future remote deployment using tools such as SSH, cloud APIs, or container orchestration platforms.

This chapter highlighted the essential DevOps components and infrastructure choices that enable smooth development, testing, and deployment processes. By integrating containerization, automated CI/CD pipelines, code quality analysis, and real-time monitoring, the project ensures a stable, scalable, and maintainable environment for the application.

These practices not only streamline the delivery workflow but also enhance system reliability and developer efficiency. With a solid infrastructure in place, the application is well-prepared for continuous improvement and real-world deployment.

# Conclusion and Perspectives

This report presented the complete development lifecycle of an intelligent medical support system, from contextual analysis to AI model fine-tuning, system implementation, and DevOps infrastructure. Leveraging natural language processing and structured development methodology, the project delivered a platform offering personalized medical insights based on reported symptoms through a user-friendly, role-based application with robust CI/CD pipelines and real-time monitoring.

The system addresses existing symptom checker limitations by combining a fine-tuned language model with retrieval-augmented architecture using a curated medical knowledge base. Containerized deployment ensures scalability and maintainability while prioritizing user experience design. This demonstrates successful integration of AI and modern software practices to create practical digital health solutions.

Future enhancements should prioritize clinical validation of AI recommendations through medical oversight and user feedback mechanisms for continuous model improvement. Feature expansions include detailed body mapping, comprehensive symptom descriptions, and teleconsultation capabilities for seamless patient-doctor interaction. Security compliance with GDPR and HIPAA regulations is essential for handling sensitive health data, alongside multilingual interfaces and accessibility standards for broader inclusivity.

Infrastructure improvements through Kubernetes migration would enhance availability and adoption potential. These enhancements position the system as a comprehensive digital health companion combining intelligent assistance, real-time support, and clinical validation for improved healthcare outcomes.

# Bibliography


[1] WebMD. *Symptom Checker: Check Your Medical Symptoms*. WebMD LLC. 2025. URL: https://symptoms.webmd.com/ (visited on 05/26/2025).

[2] Mayo Clinic. *Symptom Checker - Mayo Clinic*. Mayo Foundation for Medical Education and Research. 2025. URL: https://www.mayoclinic.org/symptom-checker/select-symptom/itt-20009075 (visited on 05/26/2025).

[3] Atlassian. *Jira Software*. URL: https://www.atlassian.com/software/jira (visited on 05/22/2025).

[4] Data Science Process Management. *What is CRISP-DM?* Last updated: December 9, 2024. 2024. URL: https://www.datascience-pm.com/crisp-dm-2/ (visited on 05/24/2025).

[5] *NGINX*. F5, Inc. 2025. URL: https://nginx.org/ (visited on 05/24/2025).

[6] *Jenkins Documentation*. Jenkins Project. 2025. URL: https://www.jenkins.io/doc/ (visited on 05/24/2025).

[7] *SonarSource: Continuous Code Quality*. SonarSource S.A. 2025. URL: https://www.sonarsource.com/ (visited on 05/24/2025).

[8] *Grafana: The open observability platform*. Grafana Labs. 2025. URL: https://grafana.com/ (visited on 05/24/2025).

[9] *Prometheus: Monitoring and alerting toolkit*. Cloud Native Computing Foundation. 2025. URL: https://prometheus.io/support-training/ (visited on 05/24/2025).

[10] *Vitest: A blazing fast unit test framework*. Vitest Contributors. 2025. URL: https://vitest.dev/ (visited on 05/24/2025).

[11] *pytest: helps you write better programs*. pytest-dev team. 2025. URL: https://docs.pytest.org/en/stable/ (visited on 05/24/2025).

[12] *DBeaver Documentation*. DBeaver Corp. 2025. URL: https://dbeaver.com/docs/dbeaver/ (visited on 05/24/2025).

[13] *React Reference API*. Meta Open Source. 2025. URL: https://react.dev/reference/react (visited on 05/24/2025).

[14] *Vite Guide*. Evan You and Vite Contributors. 2025. URL: https://vite.dev/guide/ (visited on 05/24/2025).

[15] *Flask Documentation*. Pallets Projects. 2025. URL: https://flask.palletsprojects.com/en/stable/ (visited on 05/24/2025).

[16] *NCBI E-utilities API*. National Center for Biotechnology Information. 2025. URL: https://www.ncbi.nlm.nih.gov/home/develop/api/ (visited on 05/24/2025).

[17] *Google Cloud Console*. Google LLC. 2025. URL: https://console.cloud.google.com/ (visited on 05/24/2025).

[18] Google. *T5-base: Text-To-Text Transfer Transformer*. Hugging Face. 2020. URL: https://huggingface.co/google-t5/t5-base (visited on 05/24/2025).

[19] Colin Raffel et al. *C4: Common Crawl-based Colossal Clean Corpus*. The dataset used to pre-train models in the T5 paper. Papers with Code. 2020. URL: https://paperswithcode.com/dataset/c4 (visited on 05/24/2025).

[20] QuyenAnhDE. *Diseases_Symptoms Dataset*. A dataset mapping symptoms to diseases, treatments. Hugging Face. 2023. URL: https://huggingface.co/datasets/QuyenAnhDE/Diseases_Symptoms (visited on 05/24/2025).

[21] *Retrieval-augmented generation*. Wikipedia. 2025. URL: https://en.wikipedia.org/wiki/Retrieval-augmented_generation (visited on 05/24/2025).

[22] sentence-transformers. *all-MiniLM-L6-v2*. A sentence embedding model. Hugging Face. 2022. URL: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (visited on 05/24/2025).


# ESPRIT SCHOOL OF ENGINEERING

**www.esprit.tn - E-mail : contact@esprit.tn**

**Siége Social** : 18 rue de l'Usine - Charguia II - 2035 - Tél. : +216 71 941 541 - Fax. : +216 71 941 889

**Annexe** : Z.I. Chotrana II - B.P. 160 - 2083 - Pôle Technologique - El Ghazala - Tél. : +216 70 685 685 - Fax. : +216 70 685 454