MEDI PORTAL – ML BASED HEALTH RECOMMENDATION PLATFORM

A PROJECT REPORT

Submitted by

ABINAYA P	**811722104003**
CHARUMATHI P	**811722104024**
JANE BEULA A	**811722104060**

in partial fulfillment of the requirements for the award degree of Bachelor in Engineering

20CS7503 DESIGN PROJECT - 3

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

NOVEMBER 2025

MEDI PORTAL – ML BASED HEALTH RECOMMENDATION PLATFORM

A PROJECT REPORT

Submitted by

ABINAYA P	**811722104003**
CHARUMATHI P	**811722104024**
JANE BEULA A	**811722104060**

in partial fulfillment of the requirements for the award degree of

Bachelor in Engineering

20CS7503 DESIGN PROJECT - 3

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

NOVEMBER 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)
SAMAYAPURAM - 621112

BONAFIDE CERTIFICATE

The work embodied in the present project report entitled "**MEDI PORTAL - ML BASED HEALTH RECOMMENDATION PLATFORM"** has been carried out by the students **ABINAYA P, CHARUMATHI P, JANE BEULA A**. The work reported here in is original and we declare that the project is their own work, except where specifically acknowledged, and has not been copied from other sources or been previously submitted for assessment.

Date of Viva Voce: ……

Ms. V. Sowmiya, M.E.,
SUPERVISOR
Assistant Professor
Department of CSE
K. Ramakrishnan College of Technology (Autonomous)
Samayapuram – 621 112

Mr. R. Rajavarman, M.E., (PH.D.,)
HEAD OF THE DEPARTMENT
Assistant Professor (Sr. Grade)
Department of CSE
K. Ramakrishnan College of Technology (Autonomous)
Samayapuram – 621 112

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

Traditional symptom-checking systems depends on fixed rule-based logic, resulting in low accuracy, limited adaptability, and poor understanding of natural user inputs. These limitations often lead to generic or incorrect assessments. To address these issues, this project introduces an AI-driven Symptom Analysis and Health Recommendation System. This proposed system is powered by LLM (Large Language Model) technology, specifically the Google Gemini 2.5 Flash Transformer model, which performs advanced natural-language understanding, symptom interpretation, and clinical reasoning. A Retrieval-Augmented Generation (RAG) layer enhances accuracy by retrieving relevant medical knowledge to support the model's predictions. Additionally, this work integrates real-time hospital interaction by identifying nearby medical facilities using the user's GPS location and provides an emergency support feature for high-severity cases, enabling quick access to medical services. This combination of LLM reasoning, RAG-based knowledge retrieval, and real-world medical assistance ensures reliable, fast, and personalized health insights, significantly improving early symptom assessment and overcoming the limitations of traditional rule-based systems.

Keywords: AI symptom analysis, LLM, Gemini 2.5 Flash, RAG, clinical reasoning, Supabase Edge Functions, condition prediction, diet guidance, hospital locator, emergency support, personalized insights.

ACKNOWLEDGEMENT

We thank our **Dr. N. Vasudevan**, Principal, for his valuable suggestions and support during the course of my research work.

We thank our **Mr. R. Rajavarman**, Head of the Department, Assistant Professor (Sr. Grade), Department of Computer Science and Engineering, for his valuable suggestions and support during the course of my research work.

We wish to record my deep sense of gratitude and profound thanks to our Guide **Ms. V. Sowmiya,** Assistant Professor, Department of Computer Science and Engineering for her keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

We are extremely indebted to our project coordinator **Mrs. R. Ramasaraswathi**, Assistant Professor, Department of Computer Science and Engineering, for her valuable suggestions and support during the course of our research work.

We also thank the faculty and non-teaching staff members of the Department of Computer Science and Engineering, K. Ramakrishnan College of Technology, Samayapuram, for their valuable support throughout the course of our research work.

Finally, we thank our parents, friends and our well wishes for their kind support.

SIGNATURE

_____

_____

_____

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

LLM	-	Large Language Model
RAG	-	Retrieval Augmented Generation
AI	-	Artificial Intelligence
ML	-	Machine Learning
SVM	-	Support Vector Machine
AIML	-	Artificial Intelligence Markup Language
REST	-	Representational State Transfer
API	-	Application Programming Interfaces
NER	-	Name Entity Recognition
NLP	-	Natural Language Processing
UI	-	User Interface
SQL	-	Structured Query Language
TF-IDF	-	Term Frequency-Inverse Document Frequency
PHP	-	Hypertext Preprocessor
GPS	-	Global Positioning System
HTML	-	Hypertext Markup Language
CSS	-	Cascading Style Sheets
SQL	-	Structural Query Language
RLS	-	Row-Level Security
JSON	-	JavaScript Object Notation
JWT	-	JSON Web Token
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Hypertext Transfer Protocol Secure

EHR	-	Electronic Health Record
RAM	-	Random Access Memory
PDF	-	Portable Document Format
UAT	-	User Acceptance Testing

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Healthcare technology has grown rapidly in recent years, but users still face major challenges when trying to understand their symptoms and make informed decisions about their health. Many individuals rely on online searches or basic mobile applications to interpret their symptoms, yet these sources often provide generic, inconsistent, and non-personalized information. As a result, users struggle to identify whether their symptoms indicate a minor issue or a potentially serious medical condition. This gap is especially evident among people who do not have immediate access to healthcare professionals or live in areas where medical resources are limited.

With the increasing availability of digital tools, users have become more dependent on technology for day-to-day health guidance. However, existing systems fail to analyze symptoms intelligently or offer reliable disease-related insights. Most platforms provide only static content and do not consider important user-specific factors such as age, gender, medical history, comorbidities, or lifestyle habits. Because of this, the guidance provided is often incomplete or irrelevant, leading to confusion or delayed decision-making.

At the same time, advancements in Artificial Intelligence-particularly in LLM-based reasoning, natural language processing, and retrieval-augmented knowledge systems have made it possible to create smarter and more context-aware healthcare applications. These technologies can analyze user inputs, match them with medical datasets, and generate meaningful insights in real time. The growing demand for remote health assistance, combined with the limitations of current systems, highlights the importance of developing an intelligent platform that can interpret symptoms, provide personalized recommendations, maintain health records, and support users during emergencies. This background sets the foundation for the development of an AI-powered healthcare decision-support system like Medi Portal.

1.2 DOMAIN SPECIFICATION

1.2.1 Artificial Intelligence

Artificial Intelligence (AI) belongs to the broader domain of Computer Science and focuses on creating systems capable of performing tasks that typically require human intelligence. This domain includes machine learning, deep learning, natural language processing, computer vision, robotics, neural network architectures, and decision-making systems.AI aims to replicate human cognitive functions such as learning, reasoning, problem-solving, perception, and language understanding. It also includes the development of intelligent agents that adapt to their environments, predictive analytics for forecasting, and automation technologies that enhance productivity across industries like healthcare, finance, manufacturing, education, and transportation.

1.2.2 Cloud Integration

Cloud Integration falls within the domain of Cloud Computing and Distributed Systems. It focuses on connecting different cloud-based resources, services, and applications to work together seamlessly. This domain involves integrating public, private, and hybrid cloud environments, ensuring smooth data flow across cloud platforms, and enabling interoperability between on-premises systems and cloud services. It also includes API management, microservices communication, identity and access control, secure data transfer, workload optimization, and continuous deployment practices. The goal of cloud integration is to enhance scalability, availability, and efficiency while reducing operational complexity. It supports modern enterprise architectures by enabling unified operations across multiple cloud ecosystems. Cloud integration ensures that diverse cloud services and on-premises systems communicate smoothly, enabling unified and scalable operations

1.2.3 Web Technology

Web Technology falls within the Information Technology domain, specifically under Web Development. It covers the design, development, deployment, and maintenance of websites and web applications. This domain includes both front-end technologies (HTML, CSS, JavaScript frameworks) and back-end technologies (server-side languages, databases, APIs), as well as protocols such as HTTP, HTTPS, WebSockets, and REST.Web Technology also includes web servers, content management systems, browser rendering processes, responsive design, and modern frameworks like React, Angular, and Node.js.

1.2.4 Database Management

Database Management is part of the Data Management domain under Information Systems. It focuses on designing, implementing, organizing, storing, retrieving, and securing data using database management systems such as MySQL, Oracle, SQL Server, MongoDB, and PostgreSQL. This domain includes data modeling, normalization, query optimization, indexing, transaction management, backup and recovery, data integrity, and access control. It also covers both relational (SQL) and non-relational (NoSQL) databases, distributed data architectures, and big-data systems.The purpose of database management is to ensure efficient, secure, consistent, and reliable data storage that supports applications, analytics, and decision-making processes across organizations.

1.3 PROBLEM STATEMENT

In today’s digital healthcare environment, most symptom-checking platforms still rely on outdated rule-based logic and keyword-matching techniques to interpret user inputs. These systems are unable to understand the way people naturally describe their symptoms, especially when the input includes long sentences, mixed medical terms, or incomplete descriptions. Because they lack true language understanding and contextual

reasoning, the information they provide often becomes inaccurate, inconsistent, and unreliable for users seeking initial medical guidance.

Traditional machine-learning models such as TF-IDF and SVM also fail to capture the depth of clinical meaning, as they cannot handle terminology variations, misspellings, or the complex relationships between symptoms and diseases. Moreover, these systems are not designed to give personalized recommendations, resulting in generic, one-size-fits-all advice that does not match the user's condition or severity level.A major limitation of existing systems is the absence of real-time medical support features. Users cannot identify nearby hospitals in emergencies, nor can they receive critical alerts when their symptoms indicate a potentially severe condition. This lack of immediate assistance increases the risk during urgent medical situations.

1.4 OBJECTIVE

- Developing an AI-driven medical assistance system that analyzes user symptoms using LLM and RAG techniques.
- Providing accurate disease predictions, severity assessment, and context-aware health insights in real time.
- Delivering personalized recommendations such as diet plans, lifestyle guidance, and precautionary measures based on user profiles.
- Maintaining secure storage of user symptom history, predictions, and recommendations through a well-structured PostgreSQL database.
- Offering emergency support through location-based hospital detection during severe or high-risk conditions.
- Ensuring a smooth user experience with a responsive web interface and efficient communication across the frontend, backend, and AI engine.

1.5 SCOPE OF THE PROJECT

The scope of the Medi Portal system encompasses the design, development, and deployment of a full-stack AI-driven medical assistance platform. It includes the implementation of an AI model capable of analyzing symptom descriptions using LLM reasoning and retrieval-augmented generation. The scope covers frontend development using modern web technologies, backend API integration, and database management using PostgreSQL with Supabase. The system also includes features such as user authentication, profile creation, health history tracking, emergency hospital locator, and personalized recommendation generation. However, the scope does not include professional medical diagnosis, prescription of medications, or real-time doctor consultations. The project scope includes ensuring secure data handling through authentication protocols and Row Level Security (RLS) for sensitive medical records. It also covers implementing scalable API endpoints, enabling smooth communication between the frontend and backend, and ensuring consistent system performance under typical user loads.

CHAPTER 2

LITERATURE SURVEY

2.1 AI-DRIVEN PERSONALIZED MEDICAL RECOMMENDATION SYSTEM

Faizan Ahmad et al. [3], present an AI-driven medical recommendation system built on supervised machine learning techniques designed to classify diseases and suggest optimal treatments. The model utilizes structured clinical datasets and applies preprocessing operations such as feature extraction, one-hot encoding, normalization, and vectorization to convert raw symptom inputs into machine-interpretable formats. The system further applies feature selection and dimensionality reduction techniques to enhance model efficiency and reduce overfitting. A classification algorithm—such as Support Vector Machine (SVM) or Random Forest-is employed to generate disease predictions using metrics like accuracy, recall, and F1-score. The system incorporates a clinician feedback loop, enabling iterative retraining and continuous refinement of diagnostic precision. To enhance personalization, a health analytics module performs time-series tracking and visualizes patient history for improved medical insight. The architecture emphasizes modular data pipelines, including data ingestion, preprocessing, model training, and inference phases.

2.2 AN INTELLIGENT DISEASE PREDICTION AND DRUG RECOMMENDATION PROTOTYPE

Suvendu Kumar Nayak et al. [8], present an intelligent prototype for disease prediction and drug recommendation, offering a concise yet effective AI-based solution. The study's abstract highlights a hybrid machine learning framework that analyzes patient symptoms, demographic details, and medical history to generate accurate disease predictions and suitable drug recommendations. Technically, the system employs a combination of Decision Tree, Random Forest, and Support Vector Machine (SVM) classifiers, supported by preprocessing, feature extraction, cross-validation, and

hyperparameter tuning to enhance predictive stability. Ensemble learning improves robustness, reduces misclassification, and enables the model to capture diverse disease patterns. The merits of the system include high diagnostic accuracy, strong model generalization, support for complex symptom combinations, and real-time clinical decision assistance. However, its limitations involve dependence on high-quality datasets, potential algorithmic bias, higher computational overhead due to multiple classifiers, and limited adaptability to unstructured or natural-language symptom inputs.

2.3 BASIC HEALTH CHATBOT SYSTEM USING PYTHON AND AIML

R. Verma et al. [6], presents a rule-based conversational healthcare assistant developed using Python and AIML, offering a simple chatbot capable of answering general health-related queries. The paper's abstract highlights a text-based interface where user inputs are matched against predefined AIML templates, enabling quick responses to common symptoms, preventive tips, and basic medical advice. Technically, the system relies on pattern-matching rules, an AIML knowledge base, and a lightweight Python engine to generate fixed conversational outputs. Its merits include ease of use, fast response generation, high customizability, and suitability for small-scale health information platforms. However, the chatbot's limitations are significant: it cannot perform disease prediction, interpret multiple symptoms, understand context, or adapt through learning. Its static responses restrict accuracy and personalization, and the absence of machine learning or NLP prevents it from providing meaningful medical insights beyond basic queries.

2.4 COLLABORATIVE FILTERING RECOMMENDATION SYSTEM BASED ON SPARSE REGULARIZATION AND HIGH-ORDER INTERACTIONS

Chen, J., et al. [2], propose a hybrid collaborative filtering framework that models high-order feature interactions and incorporates sparse regularization to improve recommendation accuracy. The system's abstract highlights its ability to capture deeper

dependencies between user preferences and item attributes, enabling context-aware and personalized recommendations even with sparse input data. Technically, the model combines matrix factorization, interaction-aware learning mechanisms, and regularization terms to enhance predictive performance while preventing overfitting. Experimental results on benchmark datasets demonstrate superior precision, recall, and robustness compared to conventional collaborative filtering approaches. In healthcare applications, this methodology can be applied to platforms like Medi Portal to provide personalized medicine recommendations by leveraging patient similarity and clinical data relationships. The system's merits include high recommendation accuracy, scalability, robustness to sparse or incomplete data, and adaptability across domains. Its limitations involve increased model complexity, dependence on quality feature engineering, and potential computational overhead for very large datasets.

2.5 DISEASE DIAGNOSIS USING DECISION TREE

S. Gupta et al. [7], presents a disease prediction system using the Decision Tree algorithm, designed to classify multiple user-provided symptoms into potential diseases. The system builds a hierarchical flowchart-like model where each node represents a symptom and branches represent possible outcomes, allowing for interpretable and stepwise diagnosis. Technology used are Decision Tree classifier, structured symptom datasets, feature selection, data preprocessing, and rule-based node evaluation. Transparent and interpretable, low computational cost, can handle both categorical and numerical data, suitable for small-to-medium datasets, and allows tracing of the reasoning path. Accuracy depends heavily on dataset size and quality, struggles with overlapping or multi-symptom analysis, cannot perform real-time predictions, lacks personalization, and does not support natural language inputs. The system is particularly useful in educational and clinical decision-support scenarios where interpretability is crucial. It provides a baseline for comparing modern AI models and can be extended by integrating ensemble methods or probabilistic reasoning for better accuracy.

2.6 HOSPITAL FINDER WEB APP USING GOOGLE MAPS API AND JAVASCRIPT 2018

L. Thomas., et al. [4], developed a web-based location service to assist users in finding nearby hospitals and healthcare facilities. By integrating Google Maps API with JavaScript, the system provides GPS-based navigation, distance calculation, and interactive route mapping. Technology used are Google Maps API, JavaScript, geolocation services, interactive UI, and real-time route updates. User-friendly, visually intuitive, supports real-time navigation, accessible on mobile and desktop, and reduces search time during emergencies. Cannot personalize results, lacks integration with symptom checkers or AI-based diagnostics, does not analyze patient data, and provides no recommendations beyond location guidance. The app is particularly useful for emergency planning and for travelers or patients in unfamiliar areas. It can be enhanced by integrating hospital ratings, availability of services, or emergency contact information to improve practical utility. Additionally, the system can be extended to include real-time traffic updates and estimated travel times to further optimize emergency response.

2.7 HYBRID MEDICINE RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING

Aniket et al. [1] propose a hybrid recommendation framework that combines collaborative filtering and content-based filtering to suggest personalized medicines based on patient similarity and drug attributes. The system leverages historical patient data, prescription patterns, and medicine properties to optimize recommendations. Collaborative filtering (cosine similarity, Pearson correlation), content-based filtering (chemical composition, dosage, therapeutic class), and optional ML optimization algorithms. Accurate and personalized recommendations, mitigates cold-start problems, continuously adapts as new patient data is added, reduces trial-and-error prescriptions, and supports informed clinical decision-making. Dependent on high-quality patient and drug datasets, computationally intensive, requires continuous data maintenance, and

may face challenges with newly introduced medicines. This hybrid system demonstrates the advantage of combining multiple recommendation strategies and can be extended with neural networks or reinforcement learning for more adaptive performance. It also supports predictive analytics to forecast medication effectiveness and patient adherence patterns.

2.8 MESIN: MULTILEVEL SELECTIVE AND INTERACTIVE NETWORK FOR MEDICATION RECOMMENDATION

Zhang et al. [10] introduce MeSIN, a deep learning framework that leverages attention mechanisms and hierarchical feature extraction for intelligent medication recommendation. The system captures multi-level dependencies between patient conditions, drug combinations, and treatment outcomes while adapting dynamically to real-time EHR data. It preprocesses patient data through feature normalization and embedding layers, ensuring that both categorical and continuous clinical variables are effectively represented. The model applies hierarchical attention to identify the most clinically relevant features at each level, capturing interactions between patient history, current symptoms, and potential drug responses. Deep neural networks, attention-based mechanisms, hierarchical feature extraction, interactive learning, EHR integration, and cross-domain knowledge representation. Highly adaptive and personalized, handles complex comorbidities, improves drug safety and prediction precision, supports real-time updates, and ensures interpretable outputs through attention mechanisms. Requires large-scale datasets and high computational resources, complex implementation, and may not be deployable in resource-limited healthcare facilities. MeSIN is particularly effective in handling multi-drug prescriptions and dynamic patient monitoring, allowing physicians to make safer treatment decisions. The framework can also be extended to integrate multi-modal data, such as lab results and imaging, for a more comprehensive personalized treatment plan.

2.9 PERSONALIZED MEDICINE RECOMMENDATION SYSTEM

P.Y. Shiva Prasad Reddy et al. [9], develop a machine learning-based medicine recommendation platform that predicts diseases and suggests medications through a web-based interface. Users input symptoms, and the system returns potential diagnoses with corresponding treatment options. Classification-driven ML models, data preprocessing pipelines, model training, cloud-based deployment, health analytics, and user web interface. Provides personalized treatment recommendations, scalable and responsive, enables continuous monitoring of patient history, allows integration of new diseases and protocols, and supports multiple concurrent users. Dependent on structured datasets, requires accurate and updated data, limited interpretation of unstructured or natural-language symptoms, and lacks advanced contextual reasoning. This system supports preventive healthcare by allowing patients to track progress over time and make informed decisions. It also enables researchers to analyze aggregate patient data to identify trends in treatment effectiveness and emerging health patterns.

2.10 SYMPTOM CHECKER USING STATIC DATASET

P. Sinha et al. [5], presents a web-based symptom checker that identifies potential diseases by matching user inputs with a predefined static dataset. The platform is simple, efficient, and does not require complex AI algorithms or external APIs. Server-side scripting (e.g., PHP), MySQL database for static symptom-disease mapping, web interface for user input, and basic pattern matching. Lightweight and cost-effective, highly accessible, fast response time, suitable for low-connectivity regions, and easy to implement for small-scale health awareness projects. Limited to predefined diseases and symptoms, cannot adapt to rare or emerging conditions, no AI or ML integration, and lacks personalization or context-aware guidance. Despite its simplicity, this system is valuable for educational purposes, early symptom awareness, and rural health initiatives. Future improvements could include integration with cloud databases or AI modules for dynamic updates and personalized recommendations.

CHAPTER 3

EXISTING SYSTEM

3.1 EXISTING SYSTEM

The existing system used for symptom analysis is built on traditional machine learning methods such as TF-IDF and SVM, which depend heavily on keywords typed by the user. This means the system can only understand symptoms when they are entered as exact words, without recognizing sentence meaning or user expressions. It cannot process natural language like “I am feeling dizzy since morning” and instead expects plain keywords like “dizziness.” Because of this limitation, the system often fails to understand real-world symptom descriptions and cannot detect combinations of symptoms that may indicate serious conditions. This makes the system outdated and unsuitable for modern health-related applications where users expect intelligent understanding and accurate responses. After receiving the keywords, the system performs simple keyword matching using a rule-based approach or a static machine learning model. The model relies entirely on a fixed symptom–disease mapping table, which does not update or learn from new data. There is no real-time learning, no contextual reasoning, and no advanced retrieval mechanism to improve prediction quality.

As a result, the system produces low-accuracy disease predictions and often gives the same result for different symptom combinations. It cannot analyze severity, cannot differentiate between similar conditions, and cannot provide personalized outcomes. The system also lacks support for recognizing rare diseases or emergency symptoms because it does not use modern AI techniques like NLP, embeddings, or RAG. The final output of the existing system is very basic and non-personalized, offering only general advice such as “drink water,” “take rest,” or “consult a doctor.” Every user receives the same suggestions regardless of age, medical history, gender, or risk level. The system does not offer location-based hospital recommendations, emergency alerts, or doctor contacts. There is no integration with real-time APIs, no patient monitoring, and no

support for urgent cases. Because the system cannot provide emergency assistance or adaptive suggestions, it fails to meet the growing demand for intelligent, personalized, and reliable medical support. Overall, Figure 3.1 clearly summarizes these limitations by highlighting the system's outdated, keyword-driven design and its inability to deliver accurate or personalized healthcare insights.

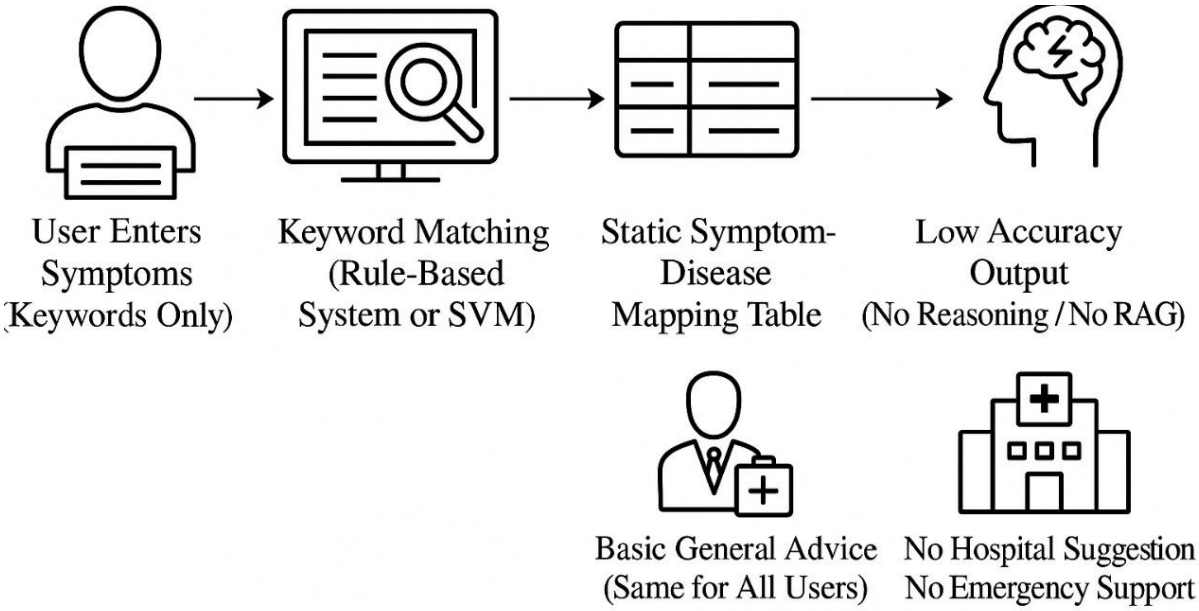

Figure. 3.1 Existing System

3.1.1 Drawback

- Depends only on keywords, not full sentences.
- Cannot understand natural language or context.
- Uses a static symptom–disease table with no updates.
- Gives low-accuracy predictions.
- No reasoning or advanced AI techniques like NLP or RAG.
- Same general advice for all users.
- No support for personalized recommendations.
- Cannot detect severity of symptoms.
- No hospital suggestion or emergency guidance.
- Cannot adapt to new symptoms or new diseases.

CHAPTER 4

PROBLEM IDENTIFIED

Existing symptom-checking systems rely mainly on rule-based logic and keyword-matching methods, which makes them unable to interpret natural language inputs. They cannot process user-written sentences such as “I feel pressure in my chest” because they only search for exact keywords. Without NLP (Natural Language Processing), these systems fail to understand context, synonyms, sentence structure, or combined symptoms, resulting in low-accuracy predictions. They also struggle with spelling mistakes, regional words, or multi-language inputs, making them less effective for diverse users. This lack of linguistic understanding reduces the system’s ability to provide meaningful and accurate health assessments.

Most traditional platforms still use static ML models like TF-IDF, Naïve Bayes, or SVM, which cannot perform semantic reasoning or adapt to new medical information. These models do not consider user-specific factors such as age, medical history, allergies, or lifestyle, so they produce generic outputs that are not personalized. Since the system lacks contextual analysis and real-time learning, it cannot differentiate between similar symptoms or detect patterns across multiple user inputs. It also cannot recognize complex symptom relationships that may indicate chronic or emerging conditions, making the predictions unreliable for real-world medical usage.

Another major limitation is the absence of severity analysis and emergency handling. Legacy systems cannot classify symptoms into mild, moderate, or severe levels, nor can they guide users during urgent conditions. They also lack integration with APIs like Google Maps for nearby hospital detection and do not maintain structured health history in secure databases. This means users are unable to review past symptom patterns or share their data with doctors when needed. The absence of proper data security, encryption, and authentication mechanisms makes these systems unsuitable for handling sensitive medical information.

CHAPTER 5

PROPOSED SYSTEM

5.1 PROPOSED SYSTEM

The proposed system provides a secure, AI-driven medical analysis platform beginning with a user authentication layer implemented using Supabase . Users register or log in to access their personalized dashboard, and all subsequent requests are authorized through JWT tokens, ensuring data privacy and controlled access to health features. Once authenticated, users can submit their symptoms through a free-text interface. The input is processed by a Supabase Edge Function, which runs an advanced NLP preprocessing pipeline consisting of tokenization, lemmatization, stop-word elimination, medical term normalization, and Named Entity Recognition (NER) to extract clinically relevant attributes such as duration, frequency, and severity cues. The cleaned text is then converted into vector embeddings and passed to the RAG (Retrieval-Augmented Generation) module.

Within the RAG pipeline, the embeddings are used to perform a high-precision vector similarity search on the Supabase Vector Database, retrieving domain-specific medical documents such as symptom–disease mappings, clinical correlations, and recommended treatments. These retrieved documents—combined with the user's processed symptoms—are forwarded to the Gemini 2.5 Flash LLM for inference. The LLM integrates both contextual embeddings and retrieved knowledge to generate structured outputs that include disease predictions, severity estimation, symptom explanations, precautionary steps, and personalized diet recommendations.

Based on the model's output, the system applies a severity classification module to categorize the condition as mild, moderate, or severe. If the detected severity is high or indicates emergency risk, the system automatically activates the EmergencyCall option. The platform also integrates with the Google Maps API, allowing real-time GPS-based hospital suggestions to guide users toward nearby medical facilities. All results—

including predictions, recommendations, and hospital directions—are presented in a structured and user-friendly format on the React-based dashboard. Through the integration of secure authentication, advanced NLP, vector search, LLM inference, and real-time geolocation services, the proposed system ensures a robust, scalable, and intelligent healthcare support experience. Through the Figure. 5.2 despite how symptoms to recommendation the system works.

5.1.1 Merits of Proposed System

- Quick health insights are provided as soon as users enter their symptoms.
- Possible diseases are predicted using AI, helping users understand their condition better.
- Helpful recommendations guide users with easy tips, precautions, and basic care.
- Nearby hospital information is shown immediately during severe cases.
- Health records are stored, allowing users to check previous results anytime.
- User-friendly design makes the system easy for anyone to use.
- Faster decision-making is supported by giving users clear severity levels for their symptoms.

5.2 BLOCK DIAGRAM

The proposed system architecture is designed as a modern, intelligent, and fully integrated pipeline that begins with an interactive User Interface (Web App), allowing users to enter symptoms, view predictions, and receive personalized guidance. The input data is securely transferred to the Backend API and Authentication layer, which manages user access, verifies identities, and ensures safe communication between the interface and the server. After authentication, the information flows into the AI Processing and RAG Engine, which performs advanced natural language understanding, symptom interpretation, medical reasoning, and retrieval of relevant medical knowledge. This engine is supported by a Vector Search Layer, enabling fast and accurate similarity searches using high-dimensional embeddings, which significantly

improves diagnostic precision and contextual awareness. All structured and unstructured data is stored in the PostgreSQL Database, providing a stable, scalable, and organized storage system for user records, symptom logs, and medical documents. The architecture is further enhanced by the Emergency Locator (Maps API), which delivers real-time location-based support by identifying nearby hospitals, clinics, and emergency services. Overall, this architecture ensures a seamless workflow from user input to intelligent processing, reliable data management, and immediate emergency assistance, making it a robust solution for modern healthcare applications. The Figure 5.1 illustrate the block diagram

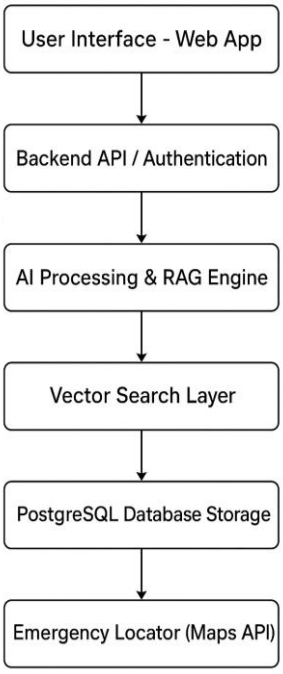

Figure. 5.1 Block Diagram.

5.3 SYSTEM ARCHITECTURE

The system is a secure, AI-powered healthcare assistant that begins with JWT-based authentication, allowing users to log in or sign up safely. Once logged in, users interact with a React dashboard built with TypeScript and Tailwind CSS, where they can enter symptoms in natural language. The frontend provides real-time validation, helpful error messages, and personalized suggestions based on previous inputs. User inputs are processed with tokenization, named entity recognition (NER), part-of-speech tagging, dependency parsing, and contextual analysis to extract key medical terms,

ensuring clean and structured data for the backend. The backend stores this data in a vector database for fast similarity search and leverages a Retrieval-Augmented Generation (RAG) system along with the Gemini 2.5 Flash LLM to predict diseases, assess severity, and provide personalized recommendations. Cases are classified as mild, moderate, or severe by a severity classifier, and the emergency call module automatically alerts nearby hospitals and emergency contacts if needed, integrating Google Maps. Supabase Edge Functions and Node.js Express power the backend, ensuring fast, scalable, and secure operations with logging, auditing, caching, rate limiting, and encryption. The system generates structured health reports with disease predictions, severity levels, diet and lifestyle advice, preventive measures, and emergency guidance, storing them in PostgreSQL and displaying them on the dashboard. Users can track their health history, compare trends, share reports, export PDFs, and receive alerts for high-risk conditions, while interactive charts and visualizations make it easy to understand their health data, providing a comprehensive, proactive, and user-centric healthcare experience.

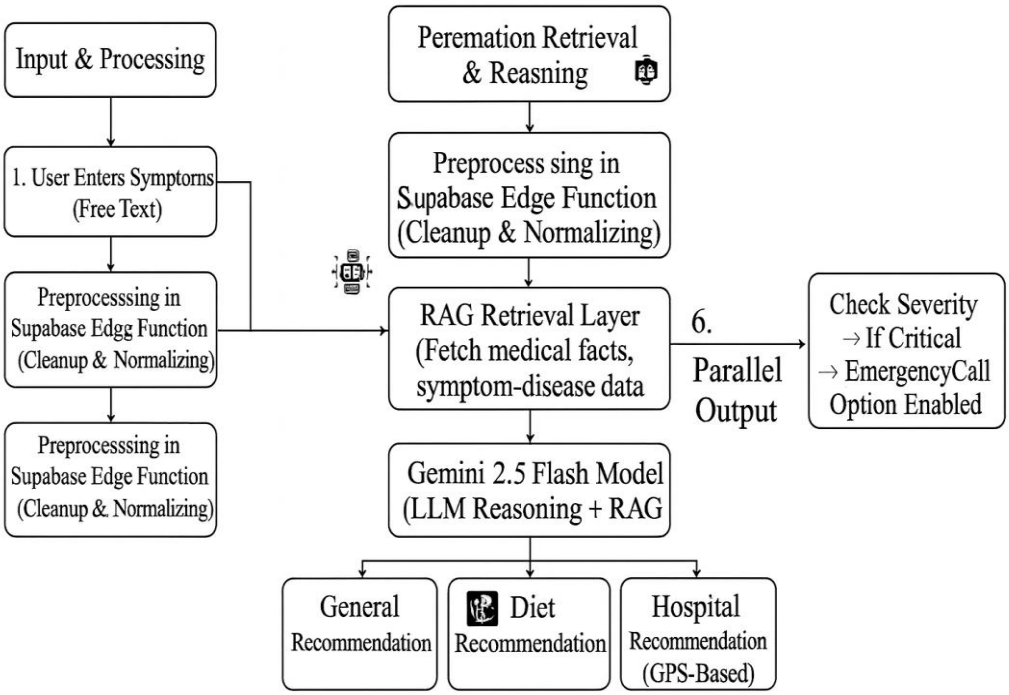

Figure. 5.2 System Architecture

CHAPTER 6

SYSTEM REQUIREMENTS

6.1 HARDWARE REQUIREMENTS

The system requires an Intel Core i5 processor or higher to ensure smooth execution of all application components. A processor of this level provides the necessary computing power to handle frontend rendering, backend processing, and database interactions without performance delays. A minimum of 8GB RAM is required for the Medi Portal to function effectively during development and deployment. The system requires at least 512GB of storage to install all necessary development tools, dependencies, project files, machine learning libraries, and the PostgreSQL database. Storage is especially important for applications that involve continuous data accumulation, such as storing user profiles, health histories, symptom logs, and AI-generated outputs.

6.2 SOFTWARE REQUIREMENTS

6.2.1 Frontend

The frontend of the Medi Portal application is built using React, TypeScript, HTML, and CSS, forming a modern and efficient technology stack capable of delivering an engaging and intuitive user experience. React acts as the core UI library, providing a component-based architecture that enables the creation of isolated, reusable, and manageable user interface elements. TypeScript enhances the reliability and maintainability of the frontend by introducing static typing, advanced IntelliSense support. HTML5 ensures semantic structuring of web content, while CSS3, often accompanied by frameworks like Tailwind CSS, ensures the visual presentation is clean, responsive, and user-friendly.

6.2.2 Backend

The backend of the Medi Portal relies on Node.js and Express.js to manage server-side processing, user authentication, API communication, and secure data flow between the frontend, machine learning components, and the database. Node.js, built on Google's V8 engine, supports asynchronous, event-driven programming models, which are essential for a medical portal that handles multiple user requests simultaneously. Express.js complements Node.js by providing a lightweight and modular framework for handling routing, middleware integration, input validation, and REST API creation.

6.2.3 Database

PostgreSQL is used as the database for the Medi Portal system due to its reliability, security, and strong support for handling structured medical data. As an advanced open-source relational database management system, PostgreSQL provides powerful features like complex querying capabilities, and support for stored procedures, which are essential for maintaining accurate and consistent health records. Its native support for JSON formats also enables the system to store semi-structured medical data-such as AI model outputs, symptom descriptions, and device-generated metadata-while still benefiting from relational integrity.

6.2.4 Tools

Several essential tools are required throughout the development lifecycle of the Medi Portal to support efficient coding, testing, debugging, collaboration, and deployment. Vite provides built-in TypeScript support, React optimizations, and environment variable handling, which help maintain code consistency and ensure efficient communication between the frontend and backend services. Postman is vital for backend API testing, allowing developers to send simulated requests, validate responses, debug errors, and verify the correctness of data exchange between the backend, machine learning systems, and external APIs.

CHAPTER 7

SYSTEM IMPLEMENTATIONS

7.1 MODULES DESCRIPTION

The Medi Portal system is divided into several main modules that work together to provide intelligent healthcare support to users. Each module has a specific purpose and helps make the system efficient and easy to use. The User Authentication and Dashboard module handles secure login and displays user information. The Health History Management module stores and tracks user health data for future reference. The Symptom-Based Disease Prediction module uses AI to analyze user symptoms and predict possible diseases. The Personalized Recommendations module suggests suitable diets, lifestyle tips, and precautions based on the user's condition. Finally, the Emergency Assistance and Hospital Locator module helps users find nearby hospitals and doctors during serious health situations. All these modules are connected to provide a complete, smart, and user-friendly medical assistance platform.

7.2 LIST OF MODULES

1. User Authentication and Profile Management
2. Symptom Analysis (LLM+RAG Engine)
3. Health Recommendations
4. Hospital Locator and Nearby Medical Support
5. Emergency Assistance

7.2.1 User Authentication and Profile Management

The User Authentication and Dashboard module ensures secure access control and personalized system interaction. Authentication uses Supabase Auth with JWT tokens, enabling encrypted login sessions, token-based identity verification, and prevention of unauthorized access. When a user signs in, the system automatically

retrieves their profile record from PostgreSQL, including demographic information, health preferences, and stored medical history. The dashboard is developed using React's component-driven architecture and TypeScript's static type checking, ensuring stable, error-free UI rendering. It displays real-time updates of user activities such as previous symptom searches, AI-generated predictions, emergency alerts, and personalized recommendations using asynchronous API calls. Advanced state management using React Hooks and Context API ensures that incoming updates from the backend reflect instantly on the UI.

7.2.2 Symptom Analysis (LLM+RAG Engine)

The Health History Management module acts as the structured digital repository for all user health interactions. Every symptom query, AI prediction, severity classification, and generated recommendation is stored in PostgreSQL tables following normalization principles to ensure data accuracy and maintain consistency. Supabase's Row Level Security (RLS) enforces strict data protection policies by isolating each user's medical records at the row level. The system captures metadata such as consultation type, device source, and follow-up actions, which helps in building a comprehensive health profile for each user. This module also supports visualization components such as trend lines and historical patterns within the dashboard to help users identify recurring symptoms or long-term conditions.

7.2.3 Health Recommendations

This module generates detailed, personalized healthcare recommendations based on the predicted condition and user-specific factors. Using LLM reasoning, the system analyzes user age, gender, lifestyle factors, past history, and severity class to produce custom diet charts, safe food suggestions, activity guidelines, rest schedules, precautionary measures, and home-care instructions. Recommendations are enriched with medical best practices retrieved during the RAG phase, ensuring they align with real clinical guidelines. They are structured into readable sections such as Diet, Lifestyle, Precautions, and Care Tips to improve clarity. The output is stored in

PostgreSQL for easy future access. The AI tailors each recommendation dynamically, ensuring the content evolves with user health patterns and new symptom entries.

7.2.4 Hospital Locator and Nearby Medical Support

The Hospital Locator and Nearby Medical Support module enables users to access critical healthcare facilities based on their real-time location. Using the device's GPS and integrated Geolocation API, the system automatically detects the user's coordinates and identifies the closest hospitals, clinics, and emergency care units within a specific radius. It provides detailed information such as hospital names, distances, available routes, and direct navigation links through map services. The module also displays additional data-such as 24/7 availability, contact numbers, and user ratings-when accessible through integrated APIs. This ensures that users can quickly reach the most suitable medical center during both routine and urgent health situations. By combining AI-driven analysis with live location data, the Medi Portal effectively bridges digital health recommendations with physical medical support, enhancing safety, accessibility, and user convenience.

7.2.5 Emergency Assistance

The Emergency Assistance and Hospital Locator module is designed to help users during critical or severe health situations. When the system detects a high-severity condition from the AI's disease prediction, this module automatically provides emergency guidance and suggests nearby hospitals. It integrates the Google Maps API to locate healthcare centers based on the user's current location and displays contact details for quick access. The backend, built with Node.js and Express.js, manages API requests and location-based searches, while the frontend developed using React and TypeScript presents the results clearly on the user interface. This module ensures that users can get immediate medical help when needed, improving safety and response time.

CHAPTER 8

SYSTEM TESTING

8.1 TESTING

Testing is an important stage in software development where the complete system is tested to ensure that all components work together correctly. The main goal of testing in the Medi Portal project is to check the accuracy, functionality, and reliability of all modules such as user authentication, health history management, disease prediction, personalized recommendations, and emergency assistance. The testing process ensures that the system meets the project requirements and performs efficiently under different user conditions. Various testing methods such as unit testing, integration testing, and system testing are used to identify and fix issues before deployment. Test cases are designed to evaluate both normal and edge-case scenarios to ensure consistent system behavior.

8.1.1 Unit Testing

Unit testing is the first and most important stage in the testing process, where individual modules or components of the Medi Portal system are tested separately to ensure that each unit functions correctly and independently. The goal is to identify bugs or logical errors at an early stage before the modules are integrated. In this project, unit testing was conducted for the login and registration modules, symptom input processing, AI-based disease prediction logic, and personalized recommendation generation. Each module was tested with valid and invalid inputs to check how it responded to different scenarios. For example, during login testing, various combinations of correct and incorrect credentials were entered to verify that only authorized users could access the system. The AI module was tested to ensure that it correctly interpreted the user's symptom input and returned the right disease prediction. Unit testing helped to validate each module's behavior, improved code quality.

8.1.2 Integration Testing

Integration testing was carried out after unit testing to verify that the various modules of the Medi Portal system work together correctly as a unified whole. It focused on testing the data flow and interaction between the frontend, backend, database, and AI model. For example, when a user enters symptoms, the frontend sends the data to the backend, which processes it and passes it to the AI engine for disease prediction. The results are then sent back to the frontend and stored in the database. Integration testing confirmed that this data flow occurred without errors or data loss. It also ensured that APIs between modules were functioning correctly and that responses were delivered in real time. Issues such as mismatched data, incorrect formatting, or failed communication between layers were identified and resolved. This testing ensured that the system performed smoothly when all components worked together.

8.1.3 System Testing

System testing is a high-level test performed to validate the complete and integrated Medi Portal application. It verifies that the entire system meets all specified requirements and works effectively under different operating conditions. During this phase, the system's major features such as user authentication, symptom analysis, health record management, AI-based disease prediction, personalized recommendation generation, and emergency hospital locator were tested end-to-end. Testers simulated real-world user interactions by entering different sets of symptoms, reviewing AI predictions, and checking the hospital locator functionality. System testing also involved checking the accuracy of AI results, the stability of the platform under heavy usage, and the system's compatibility with different browsers and devices.

8.1.4 Functional Testing

Functional testing focuses on verifying that all the features of the Medi Portal system work according to the project's functional specifications. It ensures that the system performs exactly what it is designed to do. Every major function was tested —

from user authentication and data management to AI-driven predictions and personalized health recommendations. For example, when a user enters symptoms, the system should display accurate disease predictions along with diet and lifestyle suggestions. The emergency assistance module was also tested to verify that it correctly displayed nearby hospitals using the Google Maps API. Each function was provided with specific inputs, and the results were compared with the expected outputs to check for correctness.

8.1.5 Security Testing

Security testing was conducted to ensure that the Medi Portal system is protected against unauthorized access, data breaches, and malicious attacks. Since the system stores sensitive health and personal information, maintaining data privacy and integrity was a top priority. The authentication process using Supabase Auth and JWT (JSON Web Tokens) was tested for secure login and session management. The PostgreSQL database with Row Level Security (RLS) was verified to confirm that users can only access their own medical records. API security was tested to ensure that data transferred between the frontend, backend, and AI model was encrypted through secure HTTPS connections. Penetration testing and SQL injection tests were also simulated to identify vulnerabilities.

8.1.6 User Acceptance Testing

User Acceptance Testing was the final stage of testing, where real users interacted with the Medi Portal system to verify that it met their expectations and was ready for real-world use. A group of test users, including students and faculty, were asked to use the portal by entering various symptoms, checking predictions, and exploring the recommendation and emergency features. They provided feedback on usability, clarity of instructions, accuracy of AI predictions, and system response time. This feedback was used to make small improvements to the chatbot interface, button placements, and layout clarity. UAT ensured that the system was easy to navigate, visually appealing, and capable of delivering a positive user experience

CHAPTER 9
RESULTS AND DISCUSSION

The Medi Portal system was evaluated using multiple sets of user symptoms to measure its ability to generate meaningful disease predictions. The results show that the AI model accurately interprets symptom descriptions and produces clear, structured outputs. For different symptom combinations such as fever, cough, headache, stomach pain, and fatigue, the system consistently identified relevant possible diseases along with severity levels. This demonstrates that the AI component performs stable and reliable reasoning, helping users understand whether their condition may require normal care, medical attention, or urgent treatment. The presented predictions were easy to understand, making the system suitable for general users without medical knowledge.

The personalized recommendation module also delivered effective results during testing. For each predicted disease, the system generated tailored suggestions including diet advice, lifestyle adjustments, home-care tips, and general precautions. These recommendations were structured in a user-friendly format, ensuring clarity and readability. The platform also recorded each prediction and recommendation in the user's health history. This enabled users to look back at previous results and identify recurring symptoms or changes in health patterns over time. Such consistent tracking supports better awareness and long-term monitoring of individual health conditions.

The emergency assistance feature produced accurate outcomes during evaluation, especially for severe symptom cases. When high-risk conditions were identified, the system successfully retrieved and displayed nearby hospitals using location-based services. It provided essential details such as distance, directions, and navigation links, making it useful in critical situations. The combined results of disease prediction, personalized recommendations, and emergency assistance show that the Medi Portal system offers a complete and efficient AI-supported health guidance experience. For detailed examples of symptom inputs, predicted diseases, personalized recommendations, and emergency assistance outputs, please refer to Appendix - B.

CHAPTER 10
CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

The Medi Portal ML-based Disease Prediction and Health Recommendation System demonstrates the effective use of Artificial Intelligence to support individuals in understanding their health conditions through simple symptom inputs. The proposed system successfully integrates symptom analysis, disease prediction, personalized recommendations, and emergency assistance into a single unified digital platform. The project highlights the practical potential of AI-driven reasoning in healthcare by producing disease predictions with clear severity classifications. These severity levels help users differentiate between mild, manageable symptoms and those requiring immediate medical attention.

Furthermore, the storage of user history allows individuals to monitor their past symptoms, predicted conditions, and the advice they received. This enables long-term health awareness and continuity of self-care. The emergency assistance module enhances the system's practical relevance by identifying nearby hospitals during critical situations, providing users with timely directions and accessible healthcare options. Overall, the project meets its objectives by presenting an intelligent, accessible, and helpful healthcare tool that supports users in making informed decisions about their well-being. The Medi Portal stands as an example of how AI can be used to improve health awareness, early detection, and user empowerment. The results show that the system can interpret user-entered symptoms using natural language processing and generate relevant predictions in a structured and user-friendly manner. This approach reduces the dependency on generic online searches and empowers users with reliable preliminary health information.

10.2 FUTURE WORK

1. Real-Time Doctor Consultations -Enable users to connect with certified medical professionals via chat, video, or voice, bridging AI predictions with expert advice.
2. Wearable Device Integration - Collect real-time health metrics like heart rate, blood oxygen, temperature, sleep, and activity to enhance prediction accuracy and personalized preventive warnings.
3. Expanded Medical Knowledge Base - Include information on rare diseases, chronic conditions, pregnancy, child health, and age-specific guidance for broader audience support.
4. Advanced Analytics & Early Warnings - Implement AI-based health trend visualization, risk assessment models, and proactive notifications to detect potential health issues early.
5. Comprehensive Healthcare Management Features - Add medicine reminders, vaccination tracking, appointment scheduling, e-prescriptions, and integration with government health portals.
6. Enhanced Security & Accessibility - Strengthen encryption, role-based access, privacy controls, and provide multilingual support to ensure safe and inclusive use.

APPENDIX – A

SOURCE CODE

Main.ts

```
export type Json =
  | string
  | number
  | boolean
  | null
  | { [key: string]: Json | undefined }
  | Json[]
export type Database = {
  __InternalSupabase: {
    PostgrestVersion: "13.0.5"
  }
  public: {
    Tables: {
      disease_recommendations: {
        Row: {
          activity_recommendations: Json
          created_at: string | null
          diet_plan: Json
          disease_name: string
          id: string
        }
          lifestyle_tips: Json
          precautions: Json
          updated_at?: string | null
        }
        Relationships: []
      }
```

```
      health_searches: {
        Row: {
          created_at: string | null
          emergency_triggered: boolean | null
          id: string
          predicted_diseases: Json | null
          recommendations: Json | null
          search_location: string | null
          severity_level: string
          symptoms: string
          user_id: string
        }
       Relationships: []
      }
      profiles: {
        Row: {
          age: number | null
          created_at: string | null
          full_name: string | null
          gender: string | null
          height: number | null
          id: string
          medical_history: string | null
          updated_at: string | null
          weight: number | null
        }
type DatabaseWithoutInternals = Omit<Database, "__InternalSupabase">
type DefaultSchema = DatabaseWithoutInternals[Extract<keyof Database, "public">]
export type Tables<
  DefaultSchemaTableNameOrOptions extends
    | keyof (DefaultSchema["Tables"] & DefaultSchema["Views"])
    | { schema: keyof DatabaseWithoutInternals },
```

```
  TableName extends DefaultSchemaTableNameOrOptions extends {
    schema: keyof DatabaseWithoutInternals
  }
    ? keyof
(DatabaseWithoutInternals[DefaultSchemaTableNameOrOptions["schema"]]["Tables"
]
&DatabaseWithoutInternals[DefaultSchemaTableNameOrOptions["schema"]]["Views
"])
    : never = never,
> = DefaultSchemaTableNameOrOptions extends {
  schema: keyof DatabaseWithoutInternals
}
  ?
(DatabaseWithoutInternals[DefaultSchemaTableNameOrOptions["schema"]]["Tables"
] &
DatabaseWithoutInternals[DefaultSchemaTableNameOrOptions["schema"]]["Views"])
[TableName] extends {
      Row: infer R
    }
    ? R
    : never
  : DefaultSchemaTableNameOrOptions extends keyof (DefaultSchema["Tables"] &
        DefaultSchema["Views"])
    ? (DefaultSchema["Tables"] &
        DefaultSchema["Views"])[DefaultSchemaTableNameOrOptions] extends {
        Row: infer R
      }
    schema: keyof DatabaseWithoutInternals
  }
    ? keyof
DatabaseWithoutInternals[DefaultSchemaTableNameOrOptions["schema"]]["Tables"]
    : never = never,
```

```
> = DefaultSchemaTableNameOrOptions extends {
  schema: keyof DatabaseWithoutInternals
}
export type CompositeTypes<
  PublicCompositeTypeNameOrOptions extends
    | keyof DefaultSchema["CompositeTypes"]
    | { schema: keyof DatabaseWithoutInternals },
  CompositeTypeName extends PublicCompositeTypeNameOrOptions extends {
    schema: keyof DatabaseWithoutInternals
  }
    ? keyof
DatabaseWithoutInternals[PublicCompositeTypeNameOrOptions["schema"]]["Compo
siteTypes"]
    : never = never,
> = PublicCompositeTypeNameOrOptions extends {
  schema: keyof DatabaseWithoutInternals
}
export const Constants = {
  public: {
    Enums: {},
  },
} as const
```

input.ts

```
import * as React from "react";
import { OTPInput, OTPInputContext } from "input-otp";
import { Dot } from "lucide-react";
const InputOTP = React.forwardRef<React.ElementRef<typeof OTPInput>,
React.ComponentPropsWithoutRef<typeof OTPInput>>(
  ({ className, containerClassName, ...props }, ref) => (
    <OTPInput
      ref={ref}
```

```
    containerClassName={cn("flex items-center gap-2 has-[:disabled]:opacity-50",
containerClassName)}
    className={cn("disabled:cursor-not-allowed", className)}
    {...props}
   />
  ),
);
InputOTPGroup.displayName = "InputOTPGroup";
const InputOTPSlot = React.forwardRef<
  React.ElementRef<"div">,
  React.ComponentPropsWithoutRef<"div"> & { index: number }
>(({ index, className, ...props }, ref) => {
  const inputOTPContext = React.useContext(OTPInputContext);
  const { char, hasFakeCaret, isActive } = inputOTPContext.slots[index];
  return (
   <div
    ref={ref}
    className={cn(
     "relative flex h-10 w-10 items-center justify-center border-y border-r border-input
text-sm transition-all first:rounded-l-md first:border-l last:rounded-r-md",
     isActive && "z-10 ring-2 ring-ring ring-offset-background",
     className,
    )}
    {...props}
      <div className="animate-caret-blink h-4 w-px bg-foreground duration-1000"
/>
     </div>
    )}
   </div>
});
InputOTPSlot.displayName = "InputOTPSlot";
```

```
const InputOTPSeparator = React.forwardRef<React.ElementRef<"div">,
React.ComponentPropsWithoutRef<"div">>(
  ({ ...props }, ref) => (
    <div ref={ref} role="separator" {...props}>
      <Dot />
    </div>
  ),
```

Progress.ts

```
import * as React from "react";
import * as ProgressPrimitive from "@radix-ui/react-progress";
import { cn } from "@/lib/utils";
const Progress = React.forwardRef<
  React.ElementRef<typeof ProgressPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof ProgressPrimitive.Root>
>(({ className, value, ...props }, ref) => (
  <ProgressPrimitive.Root
    ref={ref}
    className={cn("relative h-4 w-full overflow-hidhden rounded-full bg-secondary",
className)}
    {...props}
    <ProgressPrimitive.Indicator
      className="h-full w-full flex-1 bg-primary transition-all"
      style={{ transform: translateX(-${100 - (value || 0)}%) }}
    />
  </ProgressPrimitive.Root>
));
Progress.displayName = ProgressPrimitive.Root.displayName;
export { Progress };
```

connect.ts

```
import * as React from "react";
import { cn } from "@/lib/utils";
const Table = React.forwardRef<HTMLTableElement,
React.HTMLAttributes<HTMLTableElement>>(
  ({ className, ...props }, ref) => (
    <div className="relative w-full overflow-auto">
      <table ref={ref} className={cn("w-full caption-bottom text-sm", className)}
{...props} />
    </div>
  ),
Table.displayName = "Table";
const TableHeader = React.forwardRef<HTMLTableSectionElement,
React.HTMLAttributes<HTMLTableSectionElement>>(
  ({ className, ...props }, ref) => <thead ref={ref} className={cn("[&_tr]:border-b",
className)} {...props} />,
);
TableHeader.displayName = "TableHeader";
const TableBody = React.forwardRef<HTMLTableSectionElement,
React.HTMLAttributes<HTMLTableSectionElement>>(
  ({ className, ...props }, ref) => (
    <tbody ref={ref} className={cn("[&_tr:last-child]:border-0", className)}
{...props} />
  ),
);
TableBody.displayName = "TableBody";
const TableFooter = React.forwardRef<HTMLTableSectionElement,
React.HTMLAttributes<HTMLTableSectionElement>>(
  ({ className, ...props }, ref) => (
    <tfoot ref={ref} className={cn("border-t bg-muted/50 font-medium
[&>tr]:last:border-b-0", className)} {...props} />
  ),
```

```
);
TableFooter.displayName = "TableFooter"
const TableRow = React.forwardRef<HTMLTableRowElement,
React.HTMLAttributes<HTMLTableRowElement>>(
  ({ className, ...props }, ref) => (
    <tr
      ref={ref}
      className={cn("border-b transition-colors data-[state=selected]:bg-muted
hover:bg-muted/50", className)}
      {...props}
    />
  ),
TableRow.displayName = "TableRow";
const TableHead = React.forwardRef<HTMLTableCellElement,
React.ThHTMLAttributes<HTMLTableCellElement>>(
  ({ className, ...props }, ref) => (
    <th
      ref={ref}
      className={cn(
        "h-12 px-4 text-left align-middle font-medium text-muted-foreground
[&:has([role=checkbox])]:pr-0",
        className,
      )}
      {...props}
    />
  ),
TableCaption.displayName = "TableCaption";
export { Table, TableHeader, TableBody, TableFooter, TableHead, TableRow,
TableCell, TableCaption };
```

APPENDIX – B

SCREENSHOTS

SAMPLE OUTPUT

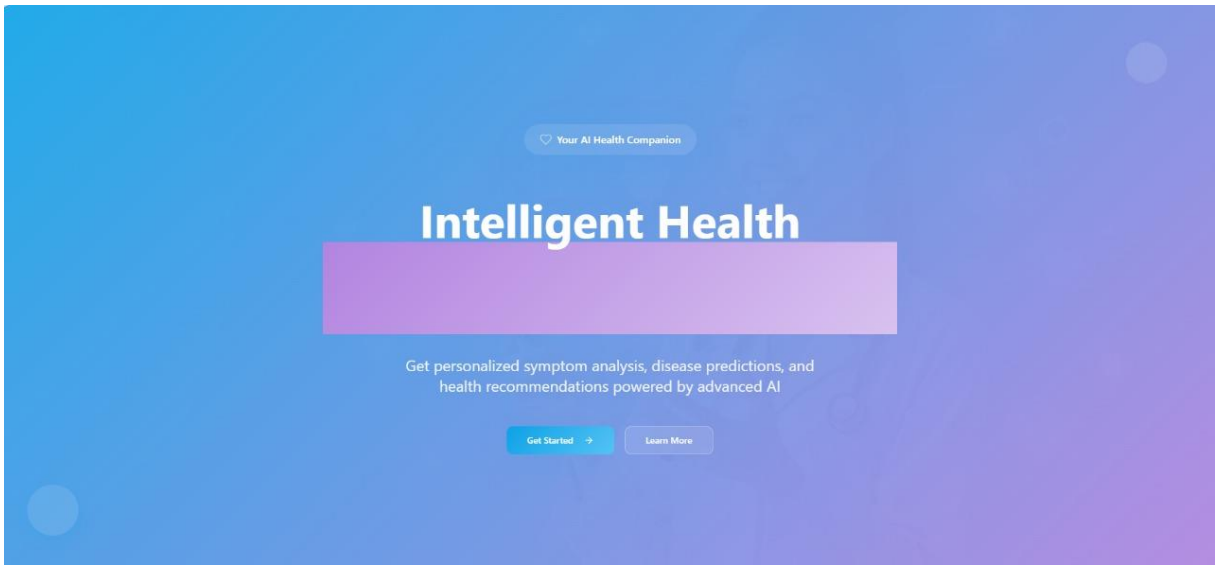

Figure. B.1. Home Page

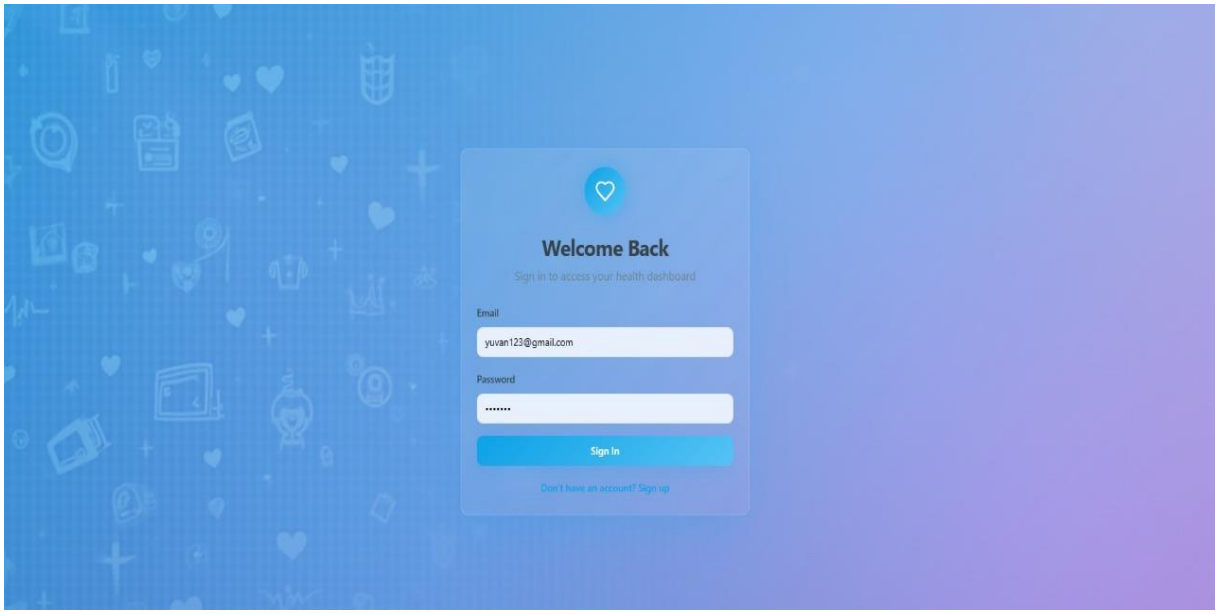

Figure. B.2. Login Page

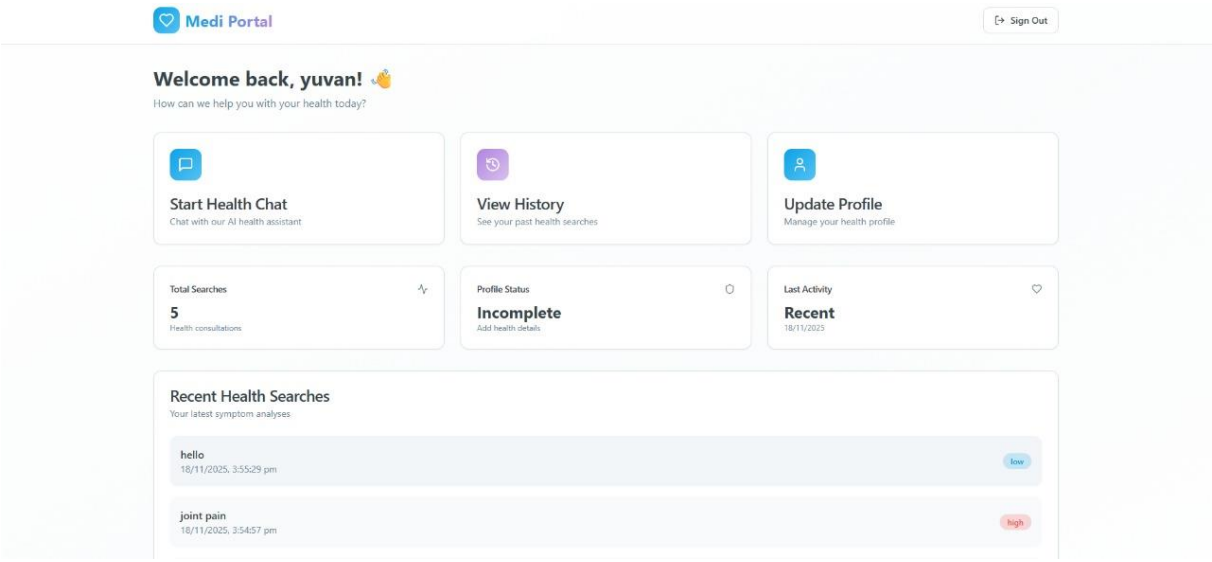

Figure. B.3. Dashboard

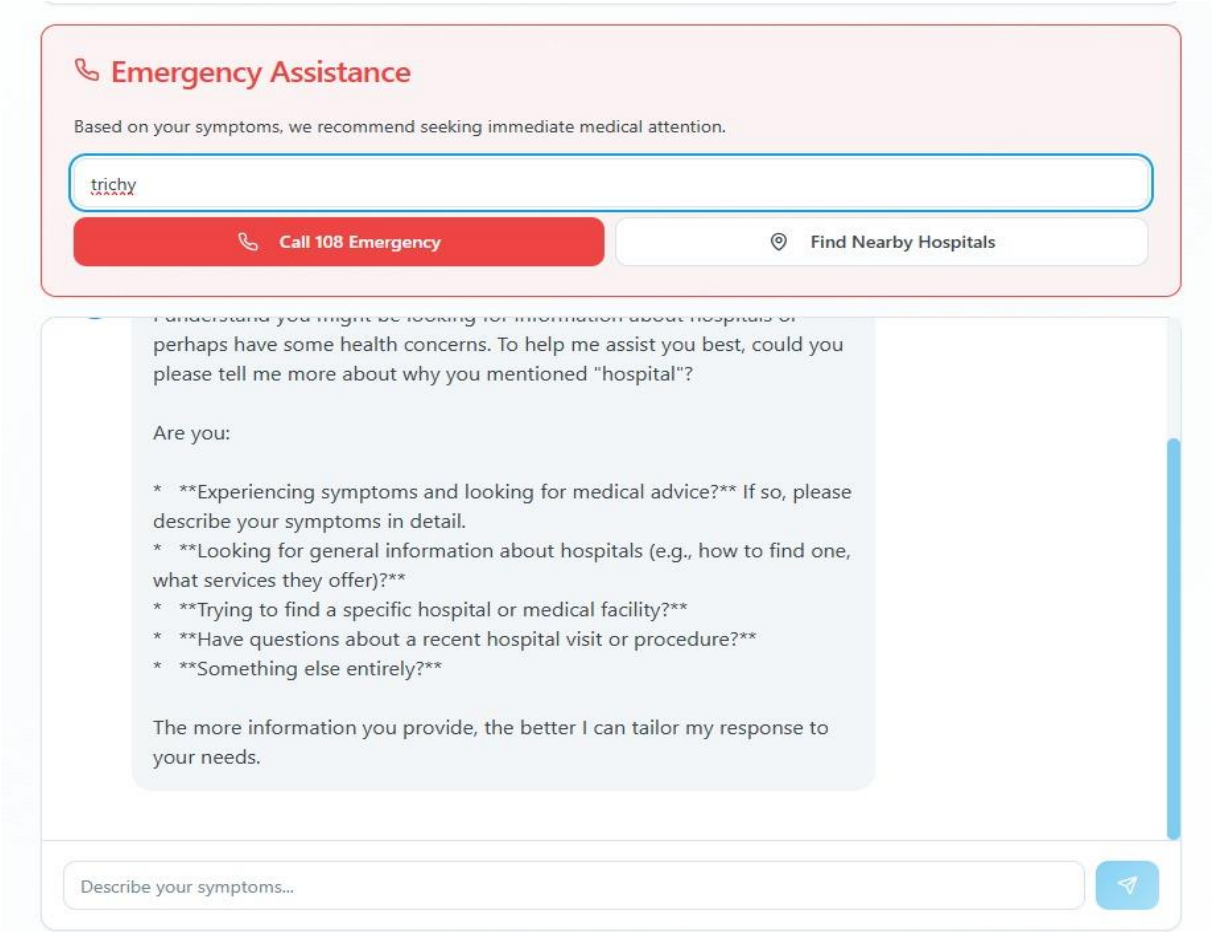

Figure. B.4. Emergency Assistance Page

Given the general symptom of "joint pain," here are some common conditions that could be considered. Please remember that these are possibilities and *not* a diagnosis.

1. **Osteoarthritis (OA):** Often described as "wear and tear" arthritis, common in older adults, affecting weight-bearing joints (knees, hips) and hands. Pain tends to worsen with activity and improve with rest.
2. **Rheumatoid Arthritis (RA):** An autoimmune disease where the body's immune system attacks its own tissues. Typically affects smaller joints first (hands, feet), often symmetrically, and is characterized by significant morning stiffness and fatigue.
3. **Tendinitis/Bursitis:** Inflammation of tendons or bursae (fluid-filled sacs that cushion joints), often due to overuse or repetitive movements. Pain is usually localized to a specific joint or area around it.
4. **Gout:** A type of inflammatory arthritis caused by a buildup of uric acid crystals in a joint, most commonly the big toe, but can affect others. It causes sudden, severe attacks of pain, redness, and swelling.

Diet Recommendations:

A balanced diet rich in anti-inflammatory foods can be beneficial for joint health.

1. **Embrace an Anti-Inflammatory Diet:** Focus on foods rich in antioxidants and omega-3 fatty acids.
   * **Include:** Fatty fish (salmon, mackerel, sardines), flaxseeds, chia seeds, walnuts, olive oil.
   * **Load up on:** Colorful fruits (berries, cherries, oranges) and vegetables (spinach, kale, broccoli, bell peppers).
   * **Choose:** Whole grains (oats, brown rice, quinoa) over refined grains.
   * **Incorporate:** Turmeric and ginger, known for their anti-inflammatory properties.
2. **Maintain a Healthy Weight:** Excess body weight puts additional stress on weight-bearing joints like knees, hips, and spine.

Figure. B.5.Suggestion to User

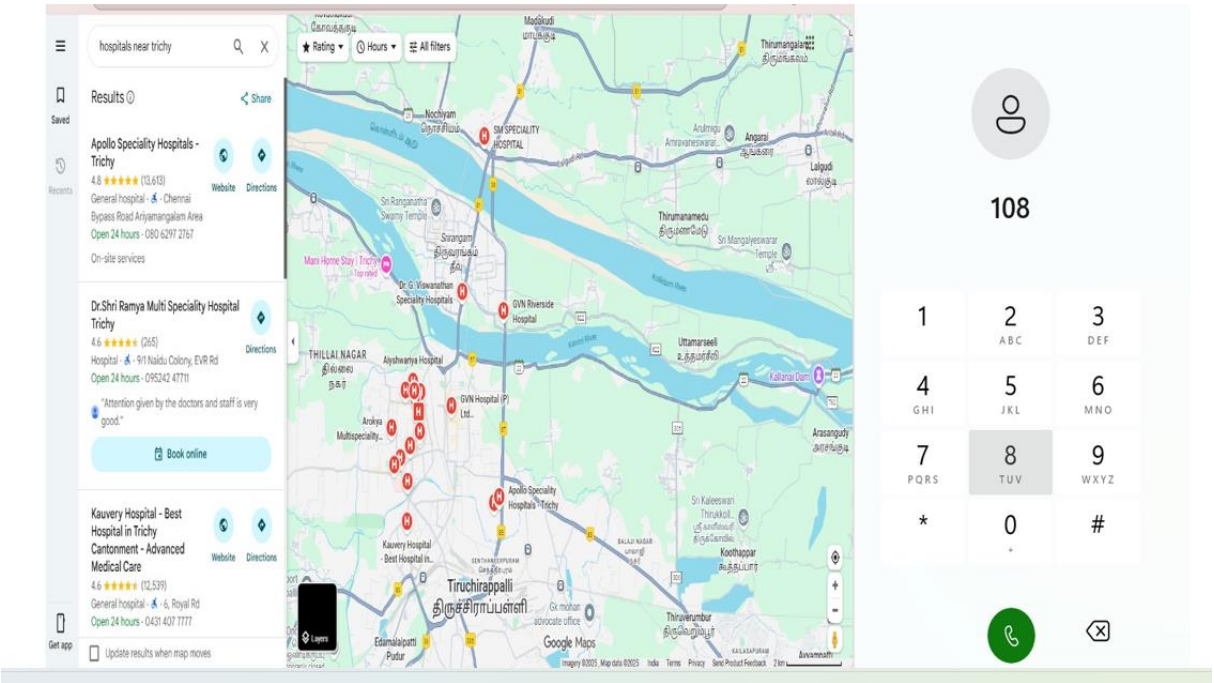

Figure. B.6.Hospital Finder & Emergency Dialer

REFERENCES

1. Aniket, Anurag Babar, Jay Pawar, Prof. Mayuri Agrawala, Hybrid Medicine Recommendation System Using Collaborative Filtering, IEEE Access, 10, 11234–11245, 2022.
2. Chen, J., Zhang, H., Zhang, Z., & Wang, H., Collaborative Filtering Recommendation System based on Sparse Regularization and High-Order interactions, 118, 103–115, 2021.
3. Faizan Ahmad, Mansi Soni, Nigar Parveen, Saurabh kumar upadhyay, Syed kazeem akbar, AI-Driven Personalized Medical Recommendation System, Journal of Healthcare Engineering, vol. 25, pp. 500–518, 2021.
4. L. Thomas, K. Desai, V. Menon, A. Roy, and S. Pillai, Hospital Finder Web App, 2021.
5. P. Sinha, A. Mehta, R. Kapoor, S. Banerjee, and K. Lodh, Symptom Checker Using Static Dataset, IEEE Access, vol. 8, pp. 15420–15430, 2019.
6. R. Verma, S. Kulkarni, P. Nair, D. Singh, and A. Patil, Basic Health Chatbot System, ACM Transactions on Healthcare Systems, vol. 15, pp. 210–230, 2020.
7. S. Gupta, M. Rao, T. Banerjee, V. Iyer, and R. Das, Disease Diagnosis Using Decision Tree, International Journal of Medical Informatics, vol. 147, pp. 112–128, 2020.
8. Suvendu Kumar Nayak, Mamata Garanayak, Sangram Keshari Swain, Sandeep Kumarpanda, Deepthi Godavarthi, An Intelligent Disease Prediction and Drug Recommendation Prototype, vol. 13, no. 3, pp. 3051-3056, 2022.
9. P.Y. Shiva Prasad Reddy, R. Rohan Reddy, S. Rakshit Reddy, Personalized Medicine Recommendation System, IEEE Reviews in Biomedical Engineering, 13, 210–225, 2019.
10. Zhang, Z., & Wang, H., Mesin: Multilevel Selective and Interactive Network for Medication Recommendation, (ETCCE), pp. 1-6, 2020.