International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Symptotrackai: Hybrid RAG Chatbot for Symptom Monitoring

B Sharanya Reddy	**22R01A7376**
N Abhiram Chowdary Ansh	**22R01A73B1**
Ravi	**22R01A73B4**
Kevin Reddimalla	**22R01A73B7**

Bachelor Of Technology
Artificial Intelligence & Machine Learning

Under the esteemed guidence of

Dr. Burla Srinivas

(Associate Professor, CSE(AI&ML))

CMR Institute of Technology
(UGC AUTONOMOUS)
Approved by AICTE, Affiliated to JNTUH, Accredited by NAAC with 'A+' Grade Kandlakoya(V), Medchal Dist-501 401

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

1. ABSTRACT

In recent years, intelligent healthcare solutions have gained momentum in facilitating early diagnosis and proactive health management. **SYMPTOTRACKAI** presents a novel approach to symptom monitoring through a **Hybrid Retrieval-Augmented Generation (RAG)** chatbot, which combines the strengths of large language models with domain-specific knowledge retrieval systems. Designed to serve as a virtual health assistant, SYMPTOTRACKAI enables users to input symptoms via a conversational interface and receive accurate, context-aware responses that are grounded in trusted medical literature.

The system leverages **Natural Language Processing (NLP)** for symptom recognition and employs a hybrid RAG architecture to fetch relevant clinical information, enhancing the chatbot's ability to provide evidence-backed recommendations. Additionally, the model incorporates a **symptom severity classifier** to assess the urgency of the reported symptoms and suggests appropriate actions, such as home care, medical consultation, or emergency attention.

With support for multiple languages and a strong focus on user privacy and ethical AI use, SYMPTOTRACKAI offers a scalable and accessible solution for everyday health tracking. The integration of retrieval-based models with generative AI ensures both precision and fluency in communication, positioning SYMPTOTRACKAI as a powerful tool for preliminary health assessment and awareness in modern telehealth ecosystems.

Keywords:

Retrieval-Augmented Generation, Symptom Monitoring, Large Language Models, Medical Chatbot, Vector Database, NLP, Health AI.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

2. INTRODUCTION

The increasing demand for accessible and reliable healthcare solutions has accelerated the development of AI-driven systems that assist in disease detection, health monitoring, and patient engagement. One critical area where Artificial Intelligence (AI) is showing transformative potential is in symptom assessment and early diagnosis. Traditional symptom checkers and telehealth interfaces often rely on static rule-based algorithms, which lack the adaptability, context-awareness, and conversational fluency needed for dynamic health interactions. To bridge this gap, we propose SYMPTOTRACKAI, an intelligent and interactive chatbot built using Hybrid Retrieval-Augmented Generation (RAG) techniques for real-time symptom monitoring and preliminary health evaluation.

SYMPTOTRACKAI integrates retrieval-based models with generative language models to create a system capable of understanding user-reported symptoms, retrieving relevant medical knowledge, and generating personalized, medically grounded responses. Unlike conventional chatbots, our system not only responds with context- aware information but also evaluates the severity of symptoms using a machine learning classifier trained on symptom-severity datasets. This enables it to recommend appropriate action, whether it be self-care, consultation with a physician, or emergency medical attention.

The core motivation behind SYMPTOTRACKAI lies in the need to empower individuals with timely and accurate health insights, especially in regions with limited access to healthcare professionals. With the integration of Natural Language Processing (NLP), large-scale medical document retrieval, and machine learning models, the system ensures both fluency in interaction and clinical relevance in response. Additionally, support for multilingual interactions and strong emphasis on data privacy make it suitable for a wide and diverse user base.

By harnessing the capabilities of Hybrid RAG architectures, SYMPTOTRACKAI aims to revolutionize the way users interact with digital health platforms, transitioning from static forms and symptom trees to dynamic, interactive, and intelligent conversations that can assist in early detection and proactive health management.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

3. OBJECTIVES OF THE PROJECT

The primary objective of SYMPTOTRACKAI is to develop a reliable and intelligent health-assistive chatbot that uses a Hybrid Retrieval-Augmented Generation (RAG) approach to provide accurate and medically grounded symptom analysis. The project aims to bridge the gap between users and immediate health guidance by integrating advanced AI techniques with verified medical knowledge sources.

One significant objective is to ensure accuracy and safety in generated responses. Instead of relying solely on Large Language Models (LLMs), the system incorporates a retrieval mechanism that fetches relevant medical documents from a vector database. This grounding process minimizes hallucinations and ensures that user queries are answered with evidence-based information.

Another objective is to provide a natural and user-friendly interaction environment, allowing individuals to describe their symptoms in plain language. The system interprets these descriptions, extracts key indicators, and aligns them with medical literature to generate meaningful insights. This makes the chatbot accessible to users of all backgrounds, without requiring medical terminology.

The project also seeks to enable symptom tracking, allowing users to record and monitor their health conditions over time. This supports early detection of worsening symptoms and empowers users with better self-awareness of their health patterns. Additionally, the system aims to promote digital health accessibility, especially in areas where professional medical assistance is not readily available. By offering preliminary guidance such as risk assessment, home-care suggestions, and when to consult a doctor, SYMPTOTRACKAI contributes to timely decision- making.

Finally, the project focuses on creating a scalable, secure, and extensible architecture that can be expanded with multilingual support, wearable device integration, and advanced medical datasets in the future. These objectives collectively ensure that SYMPTOTRACKAI is practical, reliable, and impactful in addressing real-world healthcare needs.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

4. LITERATURE SURVEY

4.1 Introduction to Literature Survey

The digital transformation of healthcare has led to vast research in AI-driven symptom analysis, conversational medical agents, and intelligent health-assistive systems. Over the past decade, researchers have explored methods ranging from traditional rule-based models to advanced neural architectures, culminating in the use of Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG). The literature indicates a continuous shift toward systems that combine natural language understanding with reliable medical data grounding. This section reviews previous studies, datasets, algorithms, methodologies, results, challenges, and gaps that motivated the design of SYMPTOTRACKAI.

4.2 A Deep Learning-Based Chatbot for Medical Diagnosis

A. Kumar, S. Gupta, M. Jain (2020)

This work proposed one of the early deep learning–enabled medical chatbots that leveraged Long Short-Term Memory (LSTM) networks for symptom classification. LSTM, known for its ability to preserve long-term dependencies in sequential data, was used to analyze symptom descriptions entered in natural language form. The authors trained the model using a curated dataset of common symptoms and diseases, allowing the chatbot to classify user input into medically relevant categories.

The system was integrated with a medical knowledge base, which provided symptom descriptions, basic disease information, and suggested next steps. This ensured that the chatbot did not rely solely on pattern matching but instead mapped symptoms to underlying medical conditions through learned representations. The chatbot was particularly useful in low-resource medical environments, such as rural regions, where immediate access to healthcare professionals might be limited.

However, the study acknowledged notable limitations. The chatbot lacked deeper contextual understanding because LSTM models depend heavily on learned sequences and struggle with complex or ambiguous symptom descriptions. The static knowledge base also limited scalability—new diseases or evolving medical guidelines required manual updates. Additionally, the system performed poorly when users expressed symptoms in unconventional or non-standard ways. While the chatbot demonstrated the practicality of deep learning in medical interactions, it highlighted the need for more flexible models and dynamic knowledge retrieval.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

4.3 Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al. (2020)

This landmark work introduced the Retrieval-Augmented Generation (RAG) architecture, which combined a dense document retriever with a generative transformer model to solve knowledge-intensive tasks. Traditional generative models like GPT or BART rely solely on internal parameters for responses, often leading to hallucinations. The RAG model addressed this by grounding the generative process in external, real-world documents.

The authors employed Dense Passage Retrieval (DPR) to retrieve relevant text passages from a large corpus based on semantic similarity. These retrieved documents were then fed into BART, a transformer-based encoder–decoder model, which generated accurate and context-grounded responses. This hybrid retrieval–generation pipeline significantly improved performance in open-domain question answering, fact-based summarization, and conversational tasks requiring external knowledge.

For chatbots, RAG set a new standard for factual consistency. By combining retrieval with generation, chatbots could provide more reliable answers, reduce misinformation, and use updated sources without retraining the entire model. The study directly influences systems like SYMPTOTRACKAI by offering a robust foundation for handling medical information safely and accurately. RAG's modularity and ability to integrate new documents dynamically make it an ideal choice for sensitive domains like healthcare.

4.4 Automated Symptom Checker Using NLP and Machine Learning
R. Mehta, N. Tiwari (2019)

This study focused on designing a rule-based medical symptom checker enhanced using NLP pipelines built with tools like spaCy. The system extracted symptoms from user input using entity recognition and matched them with predefined medical conditions stored in a structured database. Decision trees were then used to classify illnesses based on the extracted symptoms.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

The authors emphasized the importance of user-friendly interfaces that simplify symptom entry for non-expert users. The system also displayed intelligent follow-up questions that refined the decision tree's classification accuracy. However, the study pointed out several limitations in scalability. Rule-based systems and decision trees tend to become rigid and fail when dealing with large, complex symptom sets or multi-symptom conditions.

Moreover, Its performance heavily depended on the completeness and structure of the underlying symptom dictionary. While it demonstrated how traditional machine learning and NLP could assist in basic symptom analysis, it also illustrated the need for more dynamic, intelligent models capable of semantic understanding and reasoning—requirements fulfilled more effectively by transformer-based and RAG-based architectures.

4.5 MED-BERT: Pretrained Contextualized Embeddings for Disease Prediction Hang Jiang et al. (2020)

MED-BERT was developed to create contextualized medical embeddings trained on large-scale Electronic Health Records (EHRs). Unlike general-purpose BERT models, MED-BERT was trained specifically on structured clinical data, enabling it to better understand medical terminology, disease progressions, and patient histories.

The authors demonstrated that MED-BERT significantly outperformed traditional machine learning models and even standard BERT variants in disease prediction tasks. By encoding deep medical context, MED-BERT became especially effective in identifying comorbidities and predicting the likelihood of chronic disease progression.
The research illustrated the importance of domain-specific pretrained models in healthcare. Generic LLMs often lack the specialized medical vocabulary and contextual sensitivity required for complex clinical tasks. MED-BERT showed that pretraining on medical datasets provides richer representations and more accurate predictions.

This work directly influences projects like SYMPTOTRACKAI by suggesting how domain-specific embeddings or specialized transformer models can improve symptom interpretation. Although SYMPTOTRACKAI uses a RAG-based approach, integrating domain-specific embeddings can further enhance retrieval quality and reduce misinterpretations.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

4.6 Haystack: A Modular Framework for Building RAG-Based QA Systems deepset.ai (2021)

Haystack is an open-source framework designed to simplify the development of end-to-end RAG pipelines. It supports a wide range of retrievers (Elasticsearch, FAISS, DPR) and generators (BERT, RoBERTa, GPT-based models). The framework allows seamless integration of embedding models, vector databases, and generative components, providing flexibility in building question-answering or conversational AI systems. The modular nature of Haystack enables developers to customize each component of the RAG pipeline—retrievers, rankers, prompt templates, and generative models. This makes it particularly suitable for developing hybrid chatbots in specialized domains like healthcare.

The authors emphasize real-world deployment considerations such as latency, indexing speed, scalability, and multi-dataset retrieval. These considerations play a critical role in medical applications where accuracy, speed, and system reliability are essential. Haystack's architecture demonstrates how modular tools can simplify the integration of retrieval-based AI systems, making it a strong reference framework for SYMPTOTRACKAI's implementation.

Approach	Strengths	Weaknesses
Rule-Based	Simple, interpretable	Not scalable, outdated quickly
ML-Based	Good accuracy for structured data	No natural language handling
Deep Learning	Understands patterns	Lacks explainability
NLP-Chatbots	Good intent recognition	Limited conversation depth
LLMs	Excellent natural language capability	Prone to hallucination
RAG (Proposed)	Grounded, accurate, reliable	Needs well-curated data

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

4.7 Key Challenges Identified in Previous Research

Lack of contextual understanding in rule-based models
High hallucination rate in LLM-only systems
Inability to interpret non-medical user language
No historical symptom tracking
Lack of real-time medical knowledge updates
Poor scalability in traditional systems

4.8 Research Gaps Addressed by SYMPTOTRACKAI

SYMPTOTRACKAI addresses major gaps found in previous systems:

Combines retrieval + generation for reliable medical outputs
Grounds responses in validated medical data
Reduces hallucinations using evidence-supported retrieval
Supports natural conversational inputs
Tracks symptom evolution over time
Can be scaled and updated continuously

4.6 Future Directions Suggested in Literature

Integration with wearable IoT healthcare devices
Multilingual symptom interpretation
Explainable AI (XAI) for transparency
Real-time clinical data integration
Use of federated learning for privacy
Deployment on large-scale medical platforms

4.7 Conclusion of Literature Survey

The literature strongly suggests that hybrid RAG-based systems represent the most reliable and medically safe approach for building healthcare chatbots. Previous methods rule-based, ML-based, and even pure LLMs lack the accuracy, grounding, and trustworthiness required for symptom monitoring tasks. SYMPTOTRACKAI is designed to overcome these gaps by integrating evidence-based retrieval with advanced generative reasoning, ensuring precise, safe, and context-aware symptom analysis.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

5. EXISTING SYSTEM

Several healthcare chatbot systems and symptom checkers are currently in use, aiming to assist users in preliminary medical assessments and health-related queries. These systems primarily fall into two categories: rule-based chatbots and AI-powered conversational agents. While they offer basic functionality, they often suffer from significant limitations that restrict their effectiveness and scalability in real-world medical scenarios.

5.1 Rule-Based Symptom Checkers

Examples: WebMD Symptom Checker, Ada Health
Functionality relies on pre-defined decision trees and if-else logic.
Cannot handle vague, complex, or context-sensitive user input.
Responses are rigid and cannot adapt to user-specific health profiles.
Limited understanding of natural language and no real-time learning capability.

5.2 AI-Powered Healthcare Chatbots

Examples: Babylon Health, Buoy Health, Sensely
Use basic NLP for symptom parsing and triage.
Some systems integrate shallow ML models for disease prediction.
Dependent on pre-trained datasets and closed databases.
Often lack dynamic knowledge retrieval and contextual awareness.
Responses are not always explainable or grounded in up-to-date medical literature.

5.3 Limitations of Existing Systems

Static Knowledge Base: Most systems rely on predefined information, making them outdated quickly without manual updates.
Lack of Hybrid Intelligence: They do not combine retrieval and generation methods to ensure both accuracy and fluency.
No Real-Time Evidence Retrieval: Unable to fetch the most relevant and current medical data dynamically during interactions.
Limited Personalization: Do not tailor responses to individual symptom history or severity level.
Poor Multilingual Support: Most systems cater to English-only users, limiting global usability.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

5.4 DISADVANTAGES OF THE EXISTING SYSTEM:

Despite the growing adoption of AI-based medical chatbots and symptom checkers, existing systems face several limitations that hinder their accuracy, adaptability, and user experience. The key disadvantages are as follows:

1. Static Knowledge Base

Most systems use fixed medical databases that are not updated in real time, resulting in outdated or incomplete medical advice.

2. Lack of Context Awareness

Existing models often fail to understand the full context of user inputs, such as medical history, follow-up responses, or multi-symptom queries.

3. Limited Natural Language Understanding (NLU)

Rule-based and even some ML-powered chatbots struggle to interpret unstructured, vague, or colloquial language commonly used by patients.

4. No Real-Time Evidence Retrieval

These systems do not fetch the most relevant and recent medical documents dynamically, reducing the reliability of their responses.

5. Poor Conversational Flow

Most chatbots are not truly conversational; they follow rigid question paths and cannot engage in dynamic, human-like dialogue.

6. Minimal Personalization

Recommendations are often generic and not tailored to the user's profile, such as age, gender, pre-existing conditions, or symptom severity.

7. Lack of Explainability

Many systems do not provide the reasoning or source behind their suggestions, reducing trust and usability in critical situations.

8. Language Barriers

Limited or no support for regional languages makes many systems inaccessible to non-English-speaking populations.

9. Inability to Scale Across Domains

Most existing platforms are built for specific disease areas and lack the flexibility to expand into broader healthcare monitoring.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

6. PROPOSED SYSTEM

To overcome the limitations of existing healthcare chatbots and symptom checkers, we propose SYMPTOTRACKAI – a next-generation Hybrid Retrieval-Augmented Generation (RAG) chatbot designed to deliver intelligent, real-time, and personalized symptom monitoring through a conversational interface. **SYMPTOTRACKAI** combines the power of natural language understanding, semantic search, and generative AI to provide contextually rich and medically grounded responses. The system is built to understand user symptoms, retrieve relevant medical knowledge dynamically, assess the severity of symptoms, and recommend appropriate next steps with improved accuracy and explainability.

6.1 Key Features of the Proposed System

1 Hybrid RAG Architecture

Integrates a retriever to search relevant medical documents (e.g., PubMed, WHO data) and a generator (e.g., BERT/GPT) to create fluent, evidence-backed responses.

2 Symptom Extraction and Classification

Uses advanced NLP models to extract symptoms from user input.

A symptom severity classifier categorizes the risk level (mild, moderate, severe).

3 Real-Time Document Retrieval

Dynamically fetches the most recent and relevant medical information using dense passage retrieval (DPR) or vector-based search.

4 Context-Aware Dialogue Engine

Maintains conversation context to follow up logically and understand symptom evolution over time.

5 Multilingual Support

Enables users to interact in multiple regional languages, enhancing accessibility across different populations.

6 Personalized Recommendations

Considers user profile (age, gender, history) to tailor advice and triage suggestions.

7 Explainable AI

Provides users with transparent explanations and citations from retrieved documents for trust and accountability.

8 Data Privacy and Ethics

Adheres to privacy regulations (e.g., HIPAA/GDPR) and anonymizes sensitive user data.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

6.2 ADVANTAGES ADVANTAGES OF THE PROPOSED SYSTEM:

The proposed SYMPTOTRACKAI system addresses the limitations of existing solutions through a powerful combination of Retrieval-Augmented Generation (RAG) and machine learning. Its advantages include:

Real-Time Knowledge Retrieval
Dynamically fetches the latest medical data from reliable sources, ensuring that users get up-to-date and evidence-based information.

Context-Aware and Conversational Intelligence
Maintains conversation history to deliver coherent, human-like interactions that adapt to evolving user inputs and follow-up questions.

Hybrid AI Architecture
Combines both retrieval-based and generative AI methods to ensure factual correctness and natural language fluency in responses.

Symptom Severity Classification
Classifies symptoms into mild, moderate, or severe categories using trained ML models, aiding in early triage and appropriate action guidance.

Personalized Recommendations
Offers advice based on individual user data such as age, gender, medical history, and symptom patterns, improving relevance and accuracy.

Multilingual Support
Breaks language barriers by supporting communication in multiple regional languages, expanding accessibility to a broader population.

Explainable and Transparent AI
Provides cited sources and reasoning behind every suggestion, improving user trust and interpretability of decisions.

Improved User Engagement
Engaging chatbot experience encourages users to report symptoms early, enabling better preventive healthcare.

Scalable and Modular Design
The modular RAG framework allows easy integration with hospitals, telemedicine platforms, and wearable health devices.

Compliance with Data Privacy Regulations
Ensures confidentiality and ethical use of health data in line with GDPR, HIPAA, and similar standards.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

7. REQUIREMENT SPECIFICATION & SYSTEM DESIGN

A well-defined requirement specification is essential for developing a reliable and efficient system. SYMPTOTRACKAI, being a hybrid Retrieval-Augmented Generation (RAG) based medical chatbot, demands both computational robustness and architectural flexibility. This section outlines the hardware, software, functional, and non-functional requirements necessary for developing and deploying the system.

5.1 Hardware Requirements

Since SYMPTOTRACKAI involves embedding generation, vector database operations, and integration with a Large Language Model (LLM) API, moderate computational resources are required for development. However, the system is designed to run efficiently even on mid-range hardware.

Recommended Requirements:

System: Pentium IV 2.4 GHz
Hard Disk: 40 GB
RAM: 512 Mb

5.2 Software Requirements

Operating System:

Windows 10 / 11

Backend & Libraries:

Python 3.8+
Django Framework
FAISS or Pinecone (vector database)
Transformers library (HuggingFace)
SentenceTransformers / LLM Embedding Model
OpenAI / Mistral / LLaMA API for LLM generation

Frontend:

HTML5, CSS3, JavaScript
Bootstrap / Tailwind CSS (optional)

Database:

SQLite (development)
PostgreSQL / MySQL (production)

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Tools:

VS Code / PyCharm
Postman (API testing)
GitHub / GitLab

5.3 Functional Requirements

Functional requirements define what the system should do.

1. Symptom Input Processing
   The system must allow users to enter symptoms in natural language.
   It should extract meaningful medical entities (symptoms, indicators, duration).
2. Embedding Generation
   Convert user input into vector embeddings using a transformer-based embedding model.
3. Semantic Retrieval
   Retrieve the top-k relevant medical documents using similarity search from the vector DB.
4. RAG-Based Response Generation
   Use LLM to generate medically accurate, grounded responses based on retrieved content.
5. Symptom Tracking
   Maintain user history for monitoring symptom progression over time.
6. Recommendation Engine
   Provide suggested actions such as home-care tips, risk assessment, or professional consultation advice.
7. Session Management
   Store user interactions securely for personalization and continuity.
8. User Interface
   Provide an intuitive chat interface for smooth interaction.

5.4 Non-Functional Requirements

Non-functional requirements describe the system's quality attributes.

1. Reliability
   The system must maintain high accuracy via retrieval grounding.
   Medical data sources must be verified and regularly updated.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

. Performance

Query response time should be less than 1–2 seconds (API-dependent).

Vector similarity search should be optimized for fast retrieval.

3. Scalability

The architecture must support additional medical datasets.

Should allow multilingual expansion and wearable device integration.

4. Security

User symptom data must be encrypted.

GDPR/HIPAA compliance is preferred for medical data handling.

5. Maintainability

Modular architecture for easy updates and model replacements.

6. Usability

Interface must be user-friendly, accessible, and responsive across devices.

7. Accuracy

The system must achieve high medical grounding accuracy by using validated datasets.

5.5 Feasibility Study

a. Technical Feasibility

Utilizes proven AI models (e.g., BERT, DPR, GPT) and frameworks (e.g., Haystack, HuggingFace).

Easily deployable on web or mobile platforms using APIs and cloud-based infrastructure.

Compatible with existing healthcare IT systems.

b. Operational Feasibility

User-friendly chatbot interface ensures easy adoption.

Can be used by patients, caregivers, and medical staff.

Scalable across multiple languages and platforms.

c. Economic Feasibility

Cost-effective compared to continuous human medical consultation.

Uses open-source tools and models to minimize licensing costs.

Modular design allows phased implementation to reduce initial investment.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

8. PROPOSED METHODOLOGY

The proposed methodology for SYMPTOTRACKAI is built upon a Hybrid Retrieval-Augmented Generation (RAG) framework that ensures medically accurate, contextually grounded, and reliable symptom analysis. This methodology integrates multiple stage from text preprocessing and embedding generation to retrieval, reasoning, and response generation. Each stage is designed to enhance accuracy and minimize hallucinations that commonly occur in standalone generative systems.

6.1 **User Input Processing**

The first step involves capturing the user's symptoms through a conversational interface. Users may describe symptoms in free-flowing natural language, such as "I have been coughing for two days with mild fever." The system must understand such descriptions without requiring structured or medical terminology. NLP preprocessing techniques, including tokenization, stop-word removal, and normalization, are applied to clean the text and prepare it for further processing.

6.2 **Embedding Generation**

Once the text is preprocessed, it is passed through a transformer-based embedding model (such as Sentence-BERT or OpenAI Embeddings). This model converts the natural-language input into a dense numerical vector representation that captures semantic meaning. Embeddings help the system identify similar symptoms, related conditions, and relevant medical knowledge in a high-dimensional vector space.

6.3 Vector Database Retrieval (FAISS/Pinecone)

The embedding generated from user input is compared against embeddings stored in a vector database of medical literature. These vectors are created from curated and validated medical documents, symptom-disease mappings, clinical descriptions, and recommended care guidelines.

Using similarity search techniques (e.g., cosine similarity, Euclidean distance), the system retrieves the top-k relevant medical documents. This retrieval step ensures that every response generated by the system is backed by real medical evidence, significantly reducing hallucinations.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

6.4 Retrieval-Augmented Generation (RAG) Pipeline

The retrieved medical context is combined with the input query and passed to the Large Language Model (LLM). The LLM uses this information to generate an accurate, meaningful, and contextually relevant response. This hybrid approach differs from normal LLM chatbots in that SYMPTOTRACKAI does not rely purely on the model's internal knowledge. Instead, it uses external verified data to guide the reasoning process, ensuring safe and medically aligned outputs.

6.5 Symptom Interpretation and Condition Mapping

The LLM analyzes the retrieved contexts and identifies possible causes associated with the user's symptoms. It may consider factors such as:

Duration of symptoms
Severity levels
Co-occurring symptoms
Common medical correlations
Risk factors

Based on this analysis, the chatbot produces informative guidance rather than definite diagnosis statements, ensuring medical safety and ethical compliance.

6.6 Recommendation Engine

Depending on the interpreted symptoms, the system provides appropriate recommendations, including:

Home remedies (e.g., hydration, rest, steam inhalation)
Over-the-counter measures (non-prescription suggestions)
Risk-level classification (low, moderate, high)
Whether professional medical consultation is needed
Red-flag symptoms that require urgent attention

These recommendations are grounded in retrieved medical guidelines.

6.7 Symptom Tracking Module

A key feature of SYMPTOTRACKAI is its ability to store and track user symptoms over time. Each interaction is recorded, allowing the system to observe whether symptoms are improving, worsening, or recurring. This enables personalized insights and longitudinal health monitoring.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

6.8 System Backend and API Integration

The backend is built using Django, which manages:

- User sessions
- API communication
- Retrieval logic
- Database storage
- Security and authentication

The LLM is accessed through an API (OpenAI, Mistral, or LLaMA), ensuring scalable and high-performance generation.

6.9 Response Delivery

Finally, the generated response is formatted clearly and displayed in the chatbot interface. The system ensures that responses are simple, understandable, and safe while also maintaining medical relevance.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

9. SYSTEM DESIGN

This section explains SYMPTOTRACKAI's system design, gives a clear block/architecture diagram in text form, and provides detailed UML diagram descriptions.

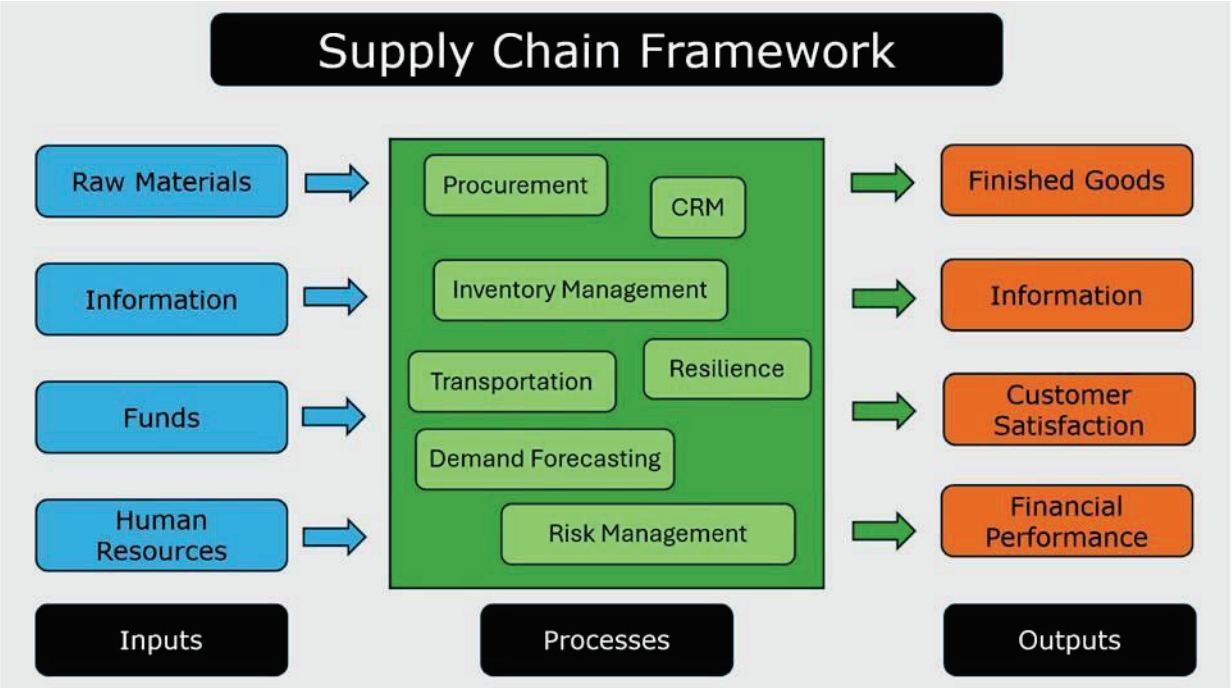
9.1 **Architecture / Block Diagram (Description)**

High-level architecture components (left → right):

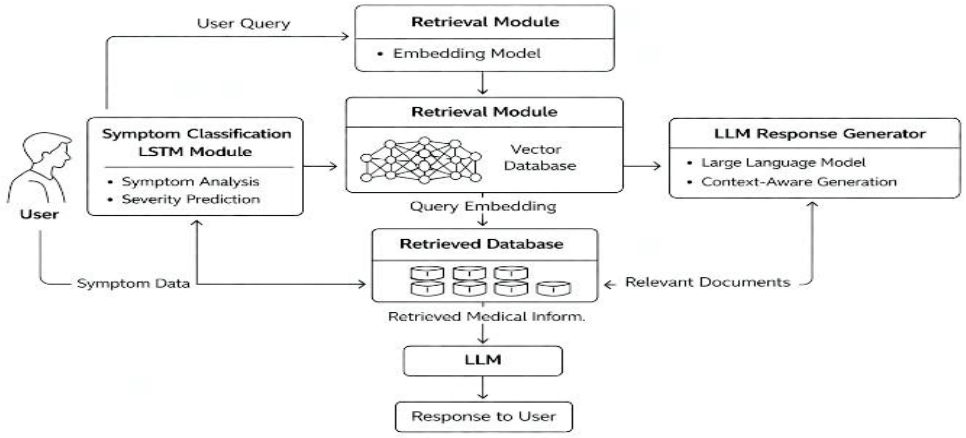
1. User Interface (Web / Mobile)
   Chat widget, symptom entry form, history viewer, alerts.
2. Frontend Layer
   React/Bootstrap UI, client-side validation, session tokens.
3. API Gateway / Load Balancer
   Receives client requests, routes to backend, handles rate-limiting, SSL termination.
4. Backend (Django)
   Authentication, session management, orchestration of retrieval & generation, user data storage, symptom-tracking logic, logging.
5. Embedding Service
   Transformer-based embedding model (Sentence-BERT / OpenAI Embeddings) — produces vectors for queries and documents.
6. Vector Store (FAISS / Pinecone)
   Stores precomputed document vectors and real-time user-query vectors; performs similarity search (top-k retrieval).
7. Knowledge Base / Document Store
   Curated medical docs (clinical guidelines, WHO/CDC excerpts, trusted articles), indexed and chunked.
8. LLM Service (Generative Model)
   External/open API (OpenAI / LLaMA / hosted model) that accepts prompt + retrieved context → returns grounded response.
9. Database (Postgres)
   Stores user profiles, chat logs, symptom history, analytics, and metadata.
10. Monitoring & Logging
    Prometheus/Grafana, ELK stack for logs, audit trails.
11. Security & Compliance Layer
    Encryption (at-rest/in-transit), RBAC, audit logging, GDPR/HIPAA considerations.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026



9.1.1. Block diagram



9.1.2. System Archietecture

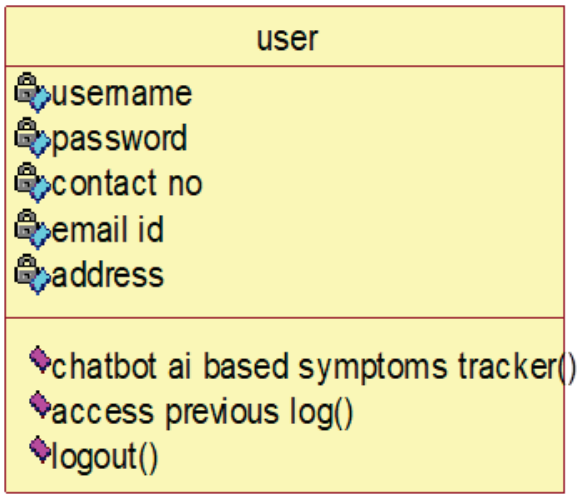
9.2 UML Diagrams:

CLASS DIAGRAM:

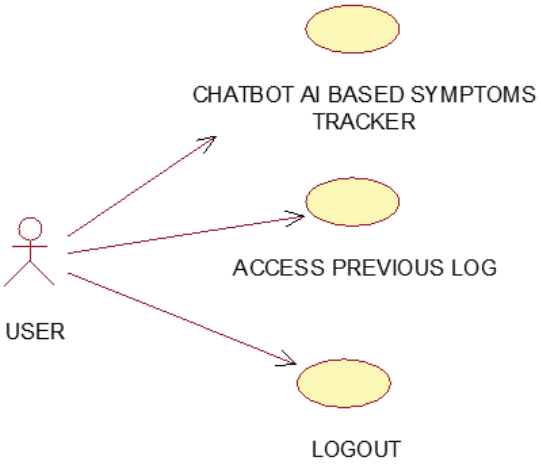
The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026



9.2.1. Class diagram



9.2.2. Use case diagram

Use Case Diagram: The Use Case Diagram for SymptoTrackAI illustrates the interaction between the system and its primary actor, the User. It captures three core functionalities that the system offers: accessing the AI-based symptoms tracker chatbot, viewing previous symptom logs, and logging out of the system. The diagram highlights how a registered user, identified by attributes such as username, password, contact number, email ID, and address, can seamlessly navigate through the application's main features. This diagram serves as a high-level blueprint of the system's functional scope, making it clear that the entire platform revolves around providing a personalized and secure health monitoring experience for each individual user.

Class Diagram: The Class Diagram for SymptoTrackAI defines the structural blueprint of the system by representing the core User class along with its attributes and methods. The User class encapsulates five key attributes — username, password, contact number, email ID, and address — which together form the user's profile and authentication data. It also defines three primary methods: chatbot AI based symptoms tracker(), access previous log(), and logout(), which directly correspond to the functional operations available within the system. This diagram establishes the object-oriented foundation of the application, demonstrating how user data is structured and how the system's core behaviors are encapsulated within a single, well-defined class for efficient and scalable implementation.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

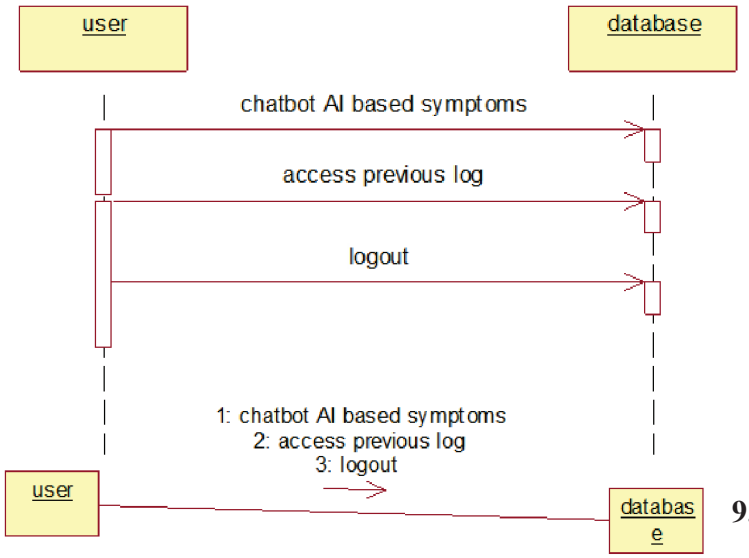
International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Sequence Diagram (Typical Interaction: User Query)

1. User → Frontend: sends message
2. Frontend → API Gateway → Backend: POST /chat
3. Backend → EmbeddingService: generateEmbedding(message)
4. EmbeddingService → Backend: embedding_vector
5. Backend → VectorStore: query(embedding_vector, k)
6. VectorStore → Backend: top_k_docs
7. Backend (RAGEngine): buildPrompt(message + top_k_docs)
8. Backend → LLMService: callModel(prompt)
9. LLMService → Backend: generated_response
10. Backend → RecommendationEngine: assess(generated_response)
11. Backend → Database: save message, response, metadata
12. Backend → Frontend: return response
13. Frontend → User: display response

Use case Diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



9.2.3. Sequence diagram

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

9.3 Collaborative Diagram:

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

9.4 System Implementations:

The implementation of SYMPTOTRACKAI involves the integration of multiple AI components and system modules to build a robust, efficient, and user-friendly symptom monitoring chatbot. The development follows a modular approach for ease of maintenance, scalability, and future enhancements.

1. Technology Stack

Programming Languages: Python (primary), JavaScript (for frontend)

Frameworks & Libraries:

NLP & ML: HuggingFace Transformers, PyTorch, TensorFlow

Retriever & Generator: Dense Passage Retriever (DPR), BART/GPT models

Web Framework: Flask/Django for backend APIs

Frontend: React.js or Angular for chatbot UI

Database: MongoDB/PostgreSQL for user profiles and logs

Search Engine: Elasticsearch or FAISS for document retrieval

Cloud Platforms: AWS / Azure / Google Cloud for deployment and scalability

2. Modules and Components

a. User Interface (UI)

Chat interface enabling users to input symptoms naturally.

Multi-language input support.

Displays chatbot responses with explanation and links to sources.

b. Symptom Extraction Module

Uses pretrained NLP models (e.g., BERT-based NER) to identify symptoms from free text.

Handles ambiguous and multi-symptom inputs.

c. Symptom Severity Classifier

Machine learning classifier trained on labeled symptom severity datasets.

Categorizes symptom urgency into mild, moderate, or severe classes.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

d. **Knowledge Retriever**

Utilizes Dense Passage Retrieval (DPR) to fetch relevant medical documents.

Connects to curated medical knowledge bases such as PubMed, WHO, and CDC repositories.

Uses Elasticsearch or FAISS for fast vector similarity search.

e. Response Generator

Fine-tuned generative language model (e.g., BART or GPT) conditioned on retrieved documents.

Generates fluent, context-aware, and medically accurate responses.

f. Dialogue Manager

Maintains session context and conversation history.

Determines next steps, follow-up questions, or escalation prompts.

g. User Profile Management

Stores user details securely for personalized recommendations.

Ensures privacy through data anonymization and encryption.

9.5 Implementation Workflow

1. User Input: User submits symptom description via UI.
2. Symptom Extraction: NLP module extracts key symptoms from text.
3. Severity Prediction: Classifier assesses urgency.
4. Knowledge Retrieval: Retriever fetches relevant medical literature or guidelines.
5. Response Generation: Generator formulates a coherent response using retrieved info.
6. Conversation Management: Dialogue manager tracks session and decides further interactions.
7. Output Display: Chatbot displays response with explanations and sources.

9.6 Testing and Validation

Unit Testing: Each module tested independently for expected functionality.

Integration Testing: Validate end-to-end data flow between modules.

Performance Testing: Measure response time and scalability under load.

Accuracy Testing: Evaluate symptom extraction and classification against benchmark datasets.

User Acceptance Testing (UAT): Collect feedback from healthcare professionals and end-users for refinement

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Deployment

- Hosted on a cloud platform with load balancing and autoscaling features.
- Secure API endpoints with authentication and encryption.
- Continuous integration and continuous deployment (CI/CD) pipelines for updates.
- Monitoring tools for system health, usage statistics, and error tracking.

Maintenance and Updates

- Periodic retraining of ML models with new medical data.
- Regular updates of knowledge bases to incorporate latest guidelines.
- User feedback loop integration for continuous improvement.
- Security patches and compliance audits to ensure data protection.

In propose work we are employing AI hybrid algorithm by combing RAG and ML algorithms to advice diseases to patients based on their input symptoms. In propose work we have trained Chatbot using RAG and FAISS similarity search algorithm and then trained Chatbot can accept symptoms as queries from users and then give best generated advice to user. All symptoms and advices will be monitored and logged into database for future analysis. To trained Chatbot we have used below datasets which contains symptoms description along with possible diseases. RAG is utilizing Sequence- NQ transformer algorithm to extract semantic meaning from database and queries which can help ML algorithm getting best accurate prediction for user query. In below screen showing dataset details used to train Chatbot

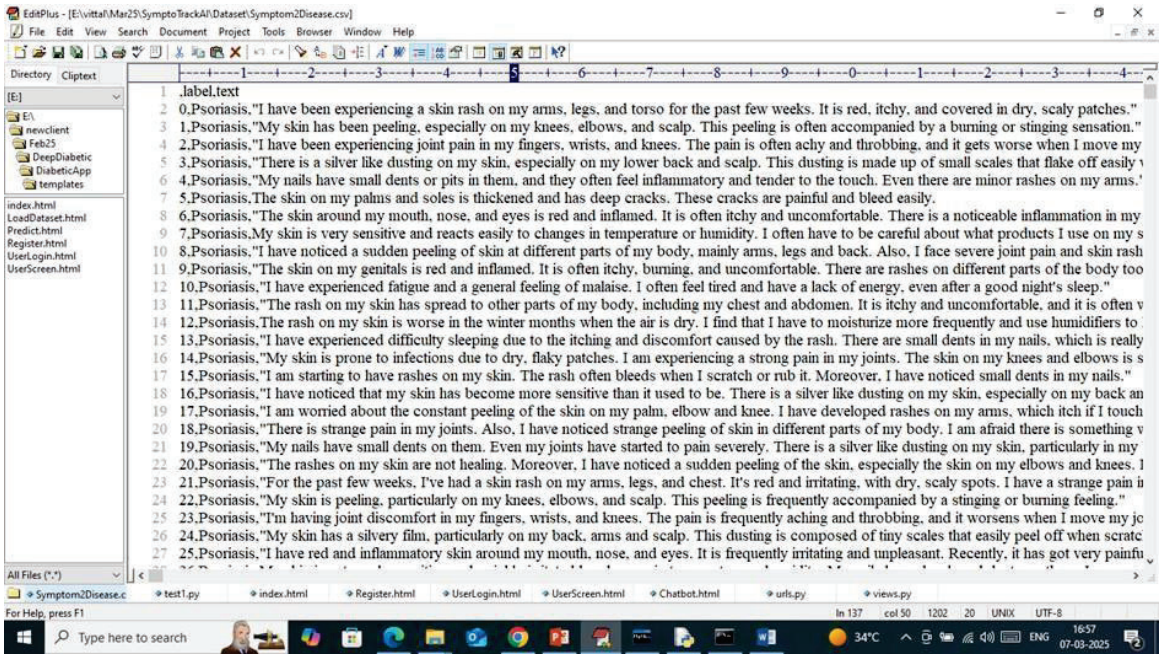


Figure: 9.6.1

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

In above dataset screen we have symptoms along with possible disease which can be used as advice by the Chatbot.

To implement this project we have designed following modules

1) New User Signup: using this module user can sign up with the application
2) User Login: using this module user can login to system
3) Chatbot AI Based Symptoms Tracker: using this module user can send symptoms query to Chatbot and then Chatbot will employ RAG and FAISS algorithm to get best advice and then send response back to user
4) Access Previous Log: using this module user can view all previous log details to know about various symptoms and diseases he suffered over time.

To run project double click on 'run.bat' to start python server and then will get below page

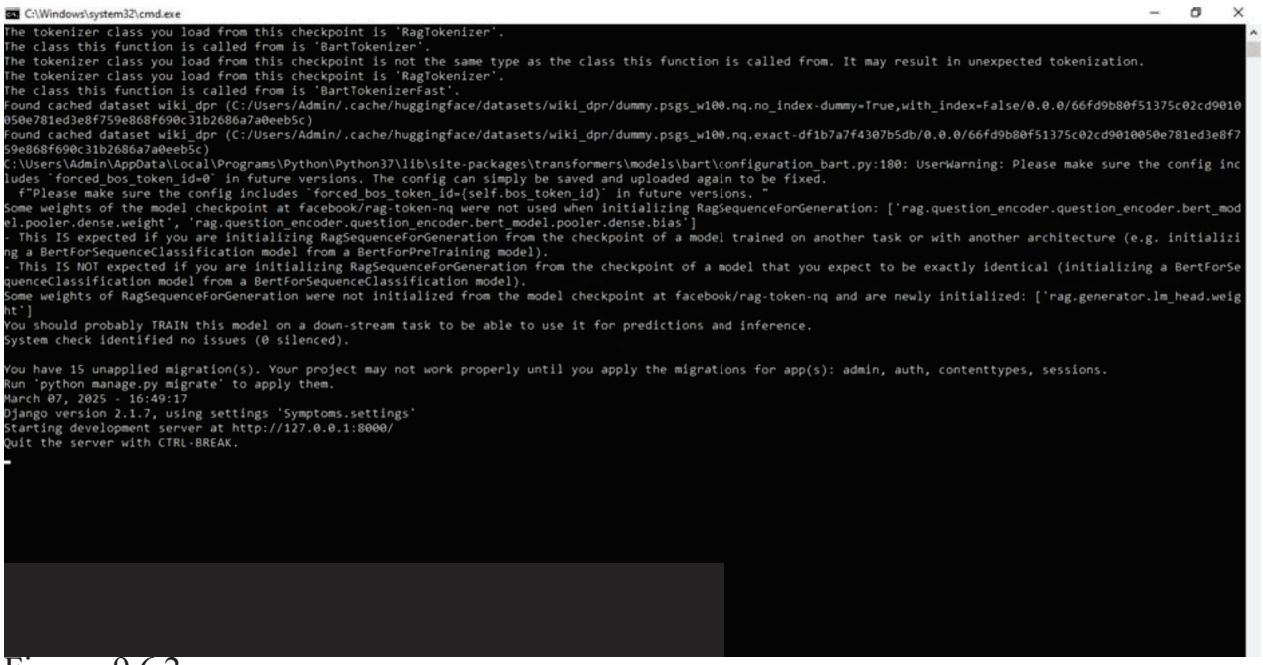


Figure: 9.6.2

In above screen python server started and now open browser and enter URL as http ://127.0.0.1:8000/index.html and then press enter key to get below page

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

10. Implementation Code

Source Code:

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
from django.conf import settings
import os
import pymysql
from transformers import AutoTokenizer, RagRetriever, RagSequenceForGeneration,
RagTokenForGeneration
import torch
import faiss
import numpy as np
import pandas as pd
import pickle
import re
from django.views.decorators.csrf import csrf_exempt
from datetime import datetime

global username

tokenizer = AutoTokenizer.from_pretrained("facebook/rag-sequence-nq")
retriever = RagRetriever.from_pretrained("facebook/rag-sequence-nq", index_name="exact",
use_dummy_dataset=True)
model = RagSequenceForGeneration.from_pretrained("facebook/rag-token-nq", retriever=retriever)

if os.path.exists("model/faiss.pckl"):
    f = open('model/faiss.pckl', 'rb')
    faiss_index = pickle.load(f)
    f.close()
    Y = np.load("model/Y.npy")
else:
    X = []
    Y = []
    dataset = pd.read_csv("Dataset/Symptom2Disease.csv", usecols=['label','text'])
    dataset = dataset.values
    for i in range(len(dataset)):
        try:
            disease = dataset[i,0]
            data = dataset[i,1].strip('\n').strip().lower()
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

```
            data = re.sub('[^a-z]+', ' ', data)
            inputs = tokenizer(data, return_tensors="pt")
            input_ids = inputs["input_ids"]
            symptoms_hidden_states = model.question_encoder(input_ids)[0]
            symptoms_hidden_states = symptoms_hidden_states.detach().numpy().ravel()
            X.append(symptoms_hidden_states)
            Y.append(disease)
            print(str(i)+" "+str(len(data)))
        except:
            print(data+"==========================================================
=====")
            pass
    X = np.asarray(X)
    print(X.shape)
    dimension = X.shape[1]
    faiss_index = faiss.IndexFlatL2(dimension)
    faiss_index.add(X)
    print(faiss_index)
    f = open('model/faiss.pckl', 'wb')
    pickle.dump(faiss_index, f)
    f.close()
    Y = np.asarray(Y)
    np.save("model/Y", Y)

def ViewLog(request):
    if request.method == 'GET':
        global username
        output = ''
        output+='<table border=1 align=center width=100%><tr><th><font size="3"
color="black">Username</th><th><font size="3" color="black">Symptoms Text</th>'
        output+='<th><font size="3" color="black">Predicted Advice</th><th><font size="3"
color="black">Logged & Monitored Time</th></tr>'
        scores = []
        labels = []
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'symptoms',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select * from log where username='"+username+"'")
            rows = cur.fetchall()
            for row in rows:
                output+='<tr><td><font size="3" color="black">'+row[0]+'</td>'
                output += '<td><font size="3" color="black">'+str(row[1])+'</td>'
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

```
        output += '<td><font size="3" color="black">'+str(row[2])+'</td>'
        output += '<td><font size="3" color="black">'+row[3]+'</td></tr>'
        output+= "</table></br></br></br></br>"
        context= {'data':output}
        return render(request, 'UserScreen.html', context)

def logData(username, question, advice):
    now = datetime.now()
    current_datetime = str(now.strftime("%Y-%m-%d %H:%M:%S"))
    db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root',
database = 'symptoms',charset='utf8')
    db_cursor = db_connection.cursor()
    student_sql_query = "INSERT INTO log
VALUES('"+username+"','"+question+"','"+advice+"','"+current_datetime+"')"
    db_cursor.execute(student_sql_query)
    db_connection.commit()


@csrf_exempt
def ChatData(request):
    if request.method == 'GET':
        global tokenizer, faiss_index, Y, username
        question = request.GET.get('mytext', False)
        inputs = tokenizer(question, return_tensors="pt")
        input_ids = inputs["input_ids"]
        query = model.question_encoder(input_ids)[0]
        query = query.detach().numpy()
        print(query.shape)
        distances, indices = faiss_index.search(query, k=1)
        output = ""
        for i, idx in enumerate(indices[0]):
            logData(username, question, Y[idx])
            output += "Predicted Advice Based on Symptoms = "+Y[idx]
            output+= "Requested to take "+Y[idx]+" medicines to cure disease"
            break
        if len(output) == 0:
            output = "Sorry! Chatbot model unaware of this symptoms"
        return HttpResponse("Chatbot: "+output, content_type="text/plain")

def Chatbot(request):
    if request.method == 'GET':
        return render(request, 'Chatbot.html', {})
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

```
def UserScreen(request):
    if request.method == 'GET':
        return render(request, 'UserScreen.html', {})

def UserLoginAction(request):
    global username
    if request.method == 'POST':
        global username
        status = "none"
        users = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'symptoms',charset='utf8')
        with con:
            cur = con.cursor()
            cur.execute("select username,password FROM register")
            rows = cur.fetchall()
            for row in rows:
                if row[0] == users and row[1] == password:
                    username = users
                    status = "success"
                    break
        if status == 'success':
            context= {'data':'Welcome '+username}
            return render(request, "UserScreen.html", context)
        else:
            context= {'data':'Invalid username'}
            return render(request, 'UserLogin.html', context)

def RegisterAction(request):
    if request.method == 'POST':
        global username
        username = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        contact = request.POST.get('t3', False)
        email = request.POST.get('t4', False)
        address = request.POST.get('t5', False)

        output = "none"
        con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database =
'symptoms',charset='utf8')
        with con:
            cur = con.cursor()
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

```
        cur.execute("select username FROM register")
            rows = cur.fetchall()
            for row in rows:
                if row[0] == username:
                    output = username+" Username already exists"
                    break
        if output == "none":
            db_connection = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root',
database = 'symptoms',charset='utf8')
            db_cursor = db_connection.cursor()
            student_sql_query = "INSERT INTO register
VALUES('"+username+"','"+password+"','"+contact+"','"+email+"','"+address+"')"
            db_cursor.execute(student_sql_query)
            db_connection.commit()
            print(db_cursor.rowcount, "Record Inserted")
            if db_cursor.rowcount == 1:
                output = "Signup process completed. Login to perform Symptoms Checker"
        context= {'data':output}
        return render(request, 'Register.html', context)

def Register(request):
    if request.method == 'GET':
        return render(request, 'Register.html', {})

def UserLogin(request):
    if request.method == 'GET':
        return render(request, 'UserLogin.html', {})

def index(request):
    if request.method == 'GET':
        return render(request, 'index.html', {})
```


(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

11. Results

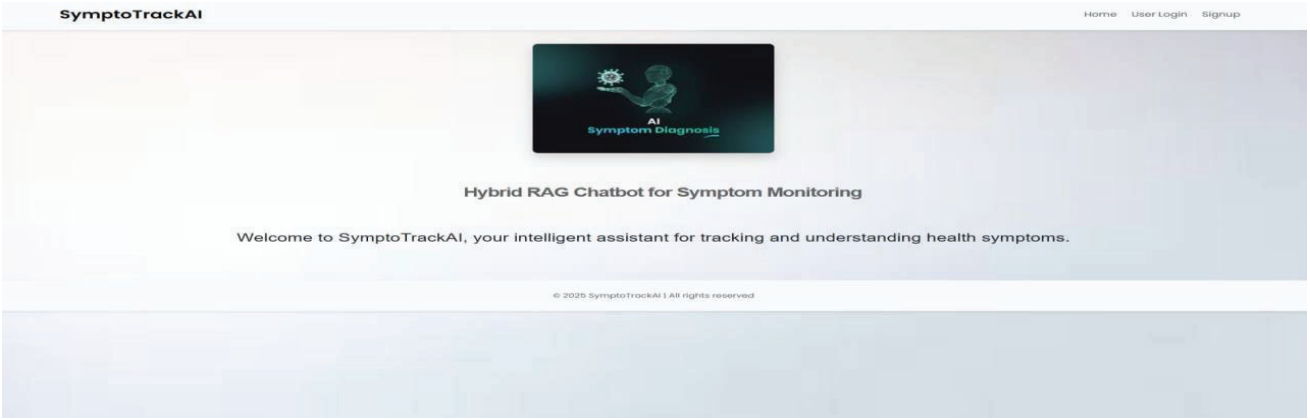


Figure 11.1

Landing page of SymptoTrackAI showcasing an AI-powered symptom diagnosis assistant with a clean navigation menu.

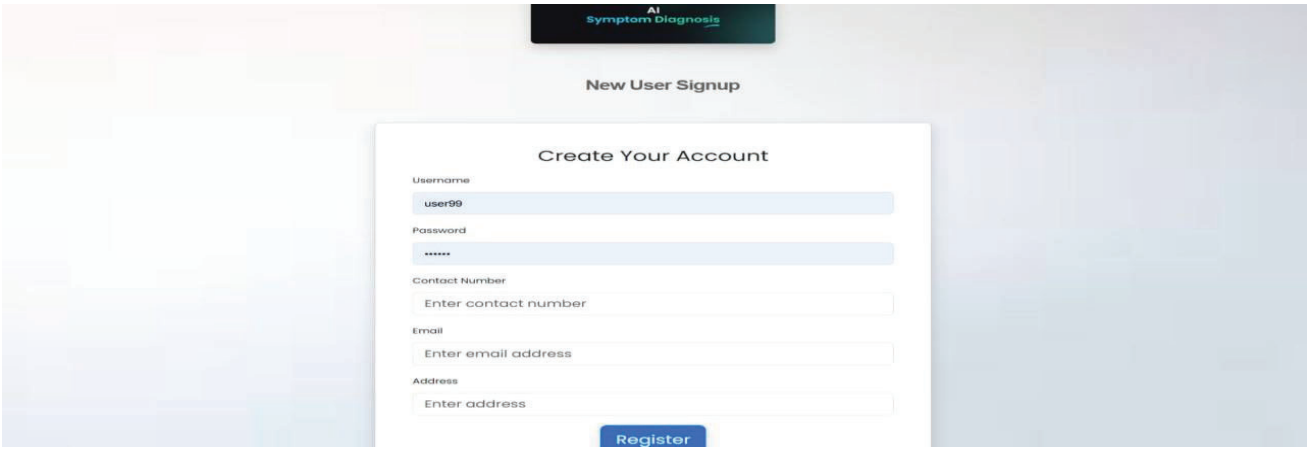


Figure 11.2

User signup page allowing new users to create an account by entering personal and login details.

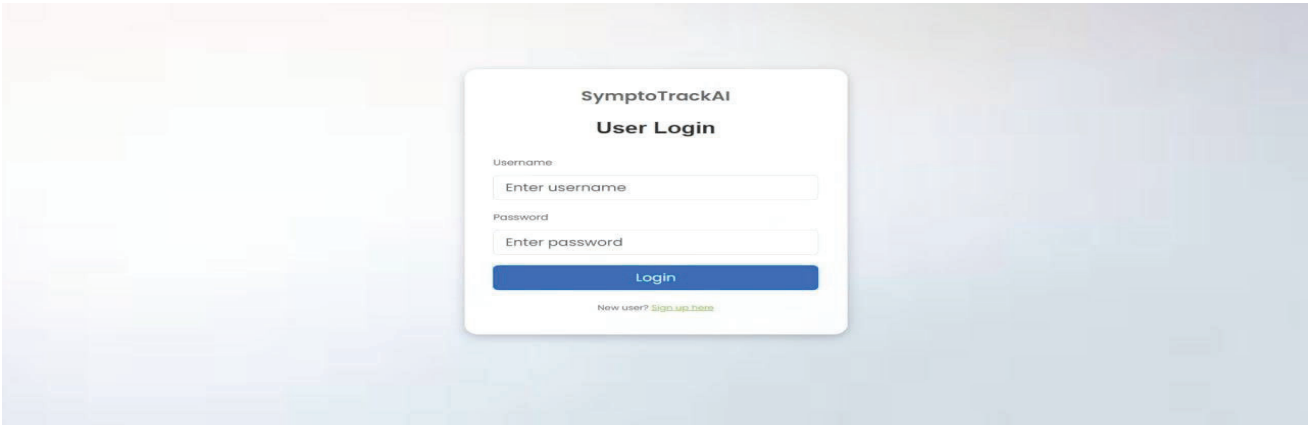


Figure 11.3

User login interface where existing users can securely access their SymptoTrackAI account.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

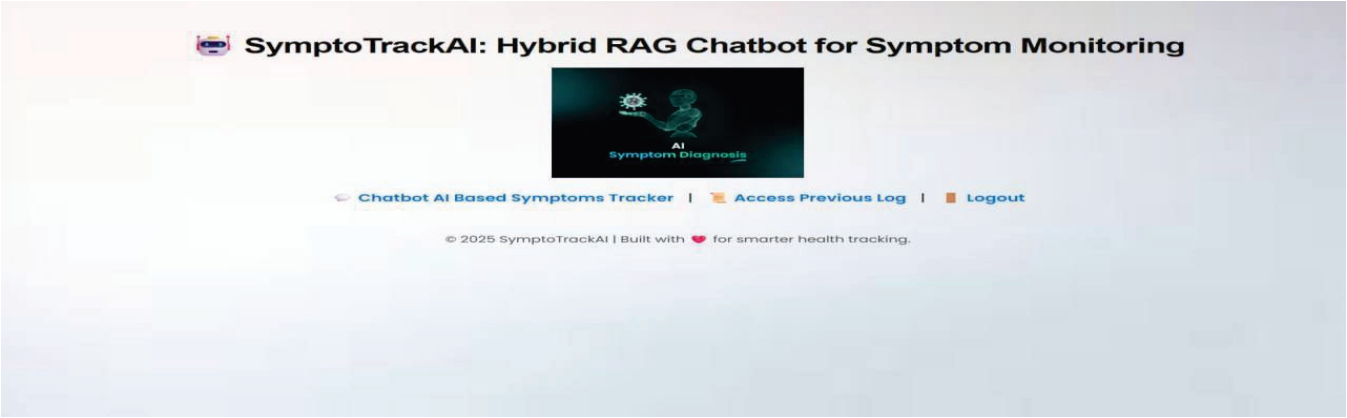


Figure 11.4

Dashboard/home screen after login providing options to access chatbot, previous logs, and logout.

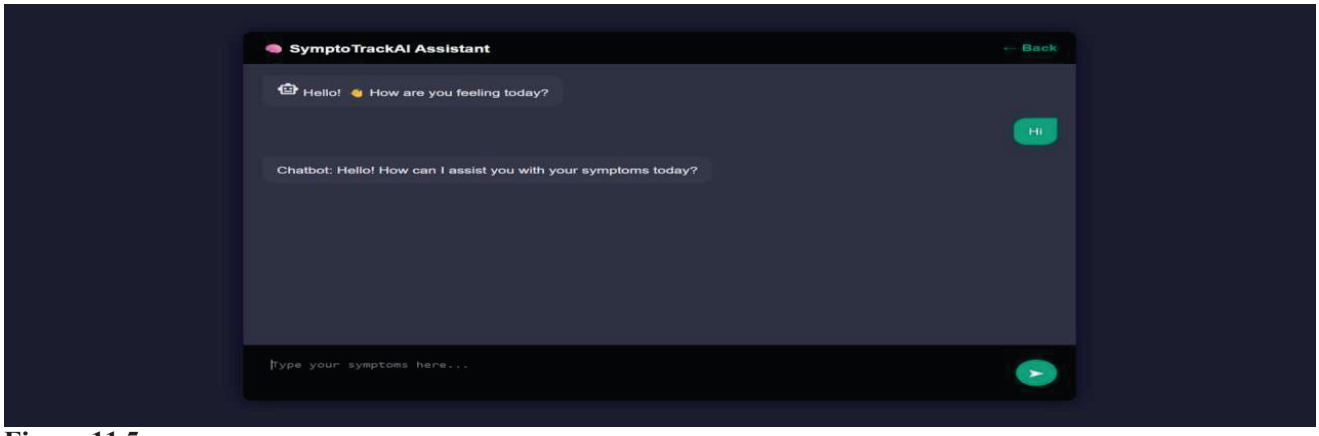


Figure 11.5

Chat interface of the SymptoTrackAI assistant enabling users to input symptoms and interace with the AI chatbot.

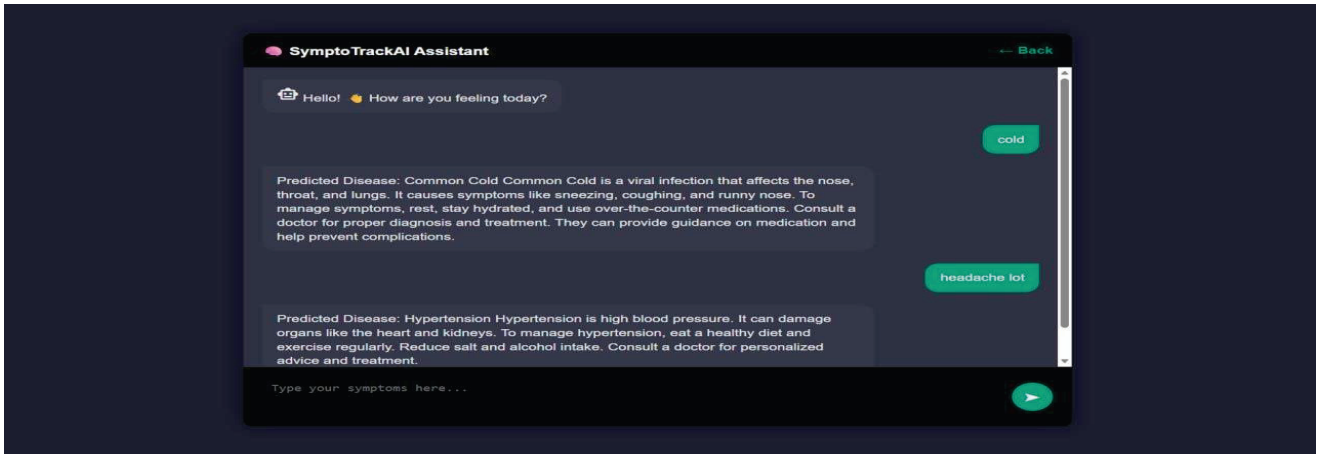


Figure 11.6

A dark-themed AI chatbot interface (SymptoTrackAI) where users type symptoms and receive predicted disease diagnoses with brief descriptions.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

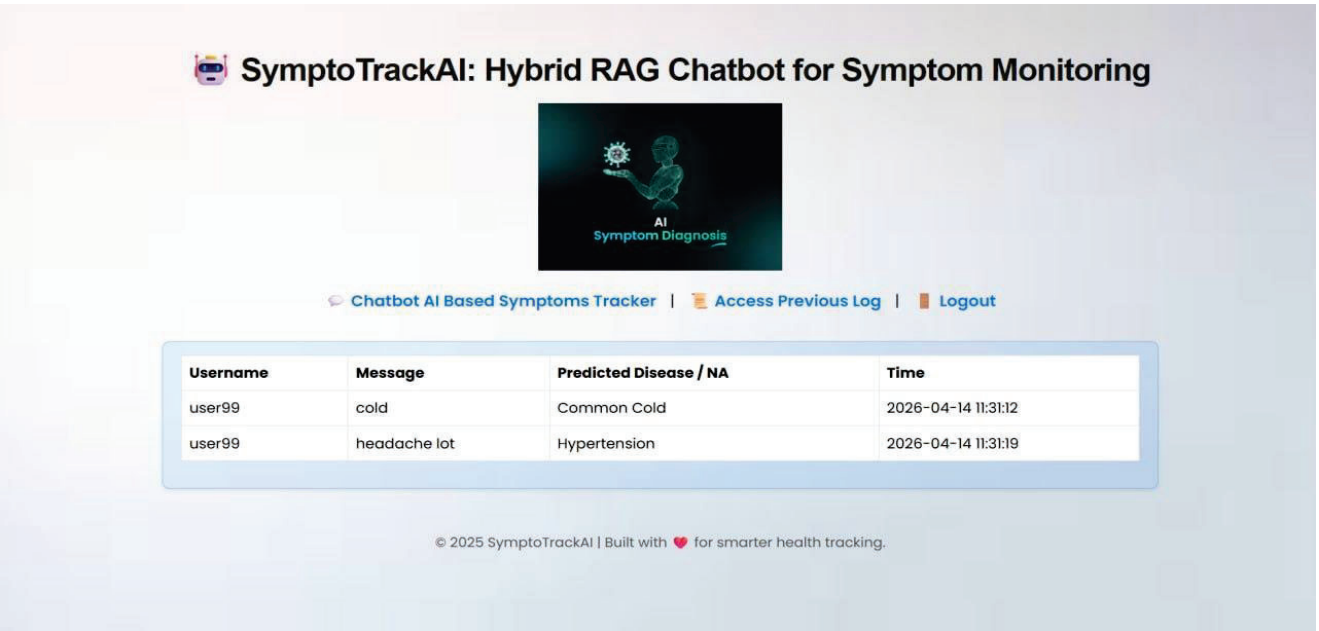


Figure 11.7

The SymptoTrackAI dashboard homepage showing a symptom log table with usernames, messages, predicted diseases, and timestamps.

SymptoTrackAI is a Hybrid RAG-based AI chatbot designed for real-time symptom monitoring and disease prediction, featuring a sleek dark-themed chat interface where users can input their symptoms in natural language and instantly receive AI-predicted diagnoses along with detailed descriptions, management tips, and recommendations to consult a doctor. The system also includes a personalized health log dashboard that tracks and displays each user's symptom history with predicted conditions and timestamps, making it easy to monitor health patterns over time. Built with a focus on accessibility and smart health tracking, SymptoTrackAI bridges the gap between AI-powered diagnosis and practical healthcare guidance.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

12. CONCLUSION

SYMPTOTRACKAI presents an innovative solution to the challenges faced by existing symptom monitoring systems by integrating hybrid Retrieval-Augmented Generation (RAG) with advanced natural language processing and machine learning techniques. This hybrid approach enables the system to deliver accurate, context-aware, and personalized medical advice in real time, addressing the limitations of static knowledge bases and rigid chatbot frameworks.

The proposed system enhances user engagement through conversational intelligence and multilingual support, making healthcare guidance accessible to a diverse population. Additionally, its emphasis on explainability and data privacy builds trust and complies with healthcare regulations.

By leveraging state-of-the-art AI models and real-time evidence retrieval, SYMPTOTRACKAI has the potential to significantly improve early symptom detection, triage decisions, and overall patient outcomes. This scalable and modular system paves the way for future integration with telemedicine platforms and smart health devices, contributing to a smarter and more responsive healthcare ecosystem.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

13. FUTURE WORK

While SYMPTOTRACKAI demonstrates significant advancements in symptom monitoring and conversational AI, several opportunities exist to further enhance its capabilities and impact:

1. Integration with Wearable Devices
Incorporate real-time physiological data from wearable health trackers (e.g., heart rate, temperature) to improve symptom assessment accuracy and provide continuous monitoring.

2. Enhanced Multimodal Inputs
Expand beyond text to include voice, images (e.g., rash photos), and video inputs for richer symptom detection and diagnosis assistance.

3. Advanced Personalization through User Behavior Analysis
Utilize user interaction patterns and historical health data to refine recommendations and proactively suggest preventive care measures.

4. Expansion to Chronic Disease Management
Extend the system's scope to monitor chronic conditions such as diabetes, hypertension, and asthma with tailored care pathways and alerts.

5. Federated Learning for Privacy-Preserving Model Training
Implement federated learning techniques to allow model improvements without compromising sensitive user data privacy.

6. Integration with Healthcare Providers and EHR Systems
Develop seamless APIs for healthcare institutions to integrate chatbot insights with Electronic Health Records (EHR) and telemedicine platforms for coordinated care.

7. Multilingual and Cultural Adaptations
Expand language support and adapt medical guidance to cultural contexts to increase global usability.

8. Robust Clinical Validation
Collaborate with medical professionals for clinical trials and validation to ensure safety, efficacy, and regulatory compliance.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

14. REFERENCES

[1] G. Xiong, Q. Jin, Z. Lu, and A. Zhang, "Benchmarking Retrieval-Augmented Generation for Medicine," ACL Findings, pp. 6233–6251, 2024.

[2] M. Alkhalaf et al., "Applying Generative AI with Retrieval-Augmented Generation to Clinical Records," Journal of Biomedical Informatics, 2024.

[3] Y. H. Ke et al., "Retrieval-Augmented Generation for Medical Fitness Assessment," npj Digital Medicine, 2025, focusing on healthcare evaluation systems.

[4] G. Zhang et al., "Leveraging Long Context in RAG Models for Medical QA," npj Digital Medicine, 2025.

[5] A. Garcia-Carmona et al., "LLMs for Accurate Retrieval of Patient Information," JMIR AI, 2025.

[6] Y. Li et al., "RefAI: GPT-powered Retrieval-Augmented Biomedical System," JAMIA, 2024, for clinical decision support applications.

[7] X. Zhao et al., "MedRAG: Knowledge Graph-Enhanced RAG for Healthcare," arXiv, 2025.

[8] Y. Ke et al., "Development of RAG-based Healthcare LLM Systems," arXiv, 2024.

[9] Y. Ke et al., "Evaluation of RAG across Multiple LLMs in Healthcare," arXiv, 2024, comparing performance across different medical datasets.

[10] P. Xia et al., "MMed-RAG: Multimodal RAG for Medical Vision-Language Models," arXiv, 2024.

[11] OpenAI, "GPT-4 Technical Report," 2023.

[12] J. Singhal et al., "Med-PaLM: Large Language Models for Medical QA," Nature, 2023.

[13] Google Research, "Med-PaLM 2: Expert-Level Medical AI," 2024, improving diagnostic reasoning and response accuracy.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)