HEALTH AI - INTELLIGENT HEALTHCARE ASSISTANT

# Project Documentation

# 1. Introduction

Project title: Health AI – Intelligent Healthcare Assistant

Team members:

1. Jegavarshini S (Team lead)
2. Janani B
3. Harini T
4. Jeevalakshmi K

# 2. Project Overview

## Purpose:

The purpose of The Health AI Assistant is designed to provide informational support to users regarding potential medical conditions and general treatment suggestions based on their symptoms. By leveraging a powerful language model, the application offers a preliminary analysis and personalized treatment suggestions. It is critical to note that this is a non-diagnostic tool and is intended for informational purposes only. The platform explicitly and repeatedly advises users to consult a qualified healthcare professional for a proper diagnosis and treatment plan.

## Features:

* **Disease Prediction**: Users can input a list of symptoms and receive a list of possible medical conditions and general medication suggestions.
* **Personalized Treatment Plans**: Based on a user's stated condition, age, gender, and medical history, the system provides generalized treatment suggestions, including home remedies.

# 3. Architecture

The application is built using a layered architecture to handle user interaction and AI processing. The core of the application is a Python script that uses the Gradio library to create a user-friendly web interface.

* **User Interface Layer**:
  + **Technology**: Gradio
  + **Functionality**: This layer provides the front-end interface where users can interact with the application through text boxes and buttons. It manages the input of symptoms and patient information and displays the generated AI responses.
* **AI & Processing Layer**:
  + **Technology**: PyTorch, Hugging Face Transformers, IBM Granite model (granite-3.2-2b-instruct)
  + **Functionality**: This layer is responsible for the core logic. It loads the pre-trained IBM Granite model and tokenizer. Based on the user's input, it crafts specific prompts for either disease prediction or treatment plan generation and sends them to the model for a response.
* **Data Management Layer**:
  + **Technology**: None (Local Processing)
  + **Functionality**: All data is processed in real-time and not stored persistently. This ensures user privacy, as no medical information is saved.

# 4. Setup Instructions

## Prerequisites:

* Python 3.7+ – Required to run the project.
* Libraries: gradio, torch, transformers
* Hardware: The ibm-granite/granite-3.2-2b-instruct model requires significant resources. A GPU with at least 8GB of VRAM is highly recommended for faster inference. CPU-only execution is possible but will be very slow.
* Internet: Required to download the model files from the Hugging Face Hub.

## Installation:

• Python 3.7+ – Programming environment.

• PyTorch – Deep learning framework (with optional CUDA support for GPU).

• Hugging Face Libraries – transformers , gradio

# 5. Folder Structure

project-root/  
│── health\_ai.py # Main application script  
│── Requierments.txt # Dependencies  
│── README.md # Documentation

# 6. Running the Application

* Save the provided code as a Python file (e.g., health\_ai.py).
* Open a terminal or command prompt.
* Run the application with the command: python health\_ai.py
* The application will start, and a local URL

# 7. API Documentation

Currently only available through the Gradio UI. Future versions may expose FastAPI endpoints.

# 8. Authentication

Open app runs without authentication for demo purposes.

Future enhancements may include login, role-based access, and teacher/student accounts.

# 9. User Interface

**Technology**: Gradio

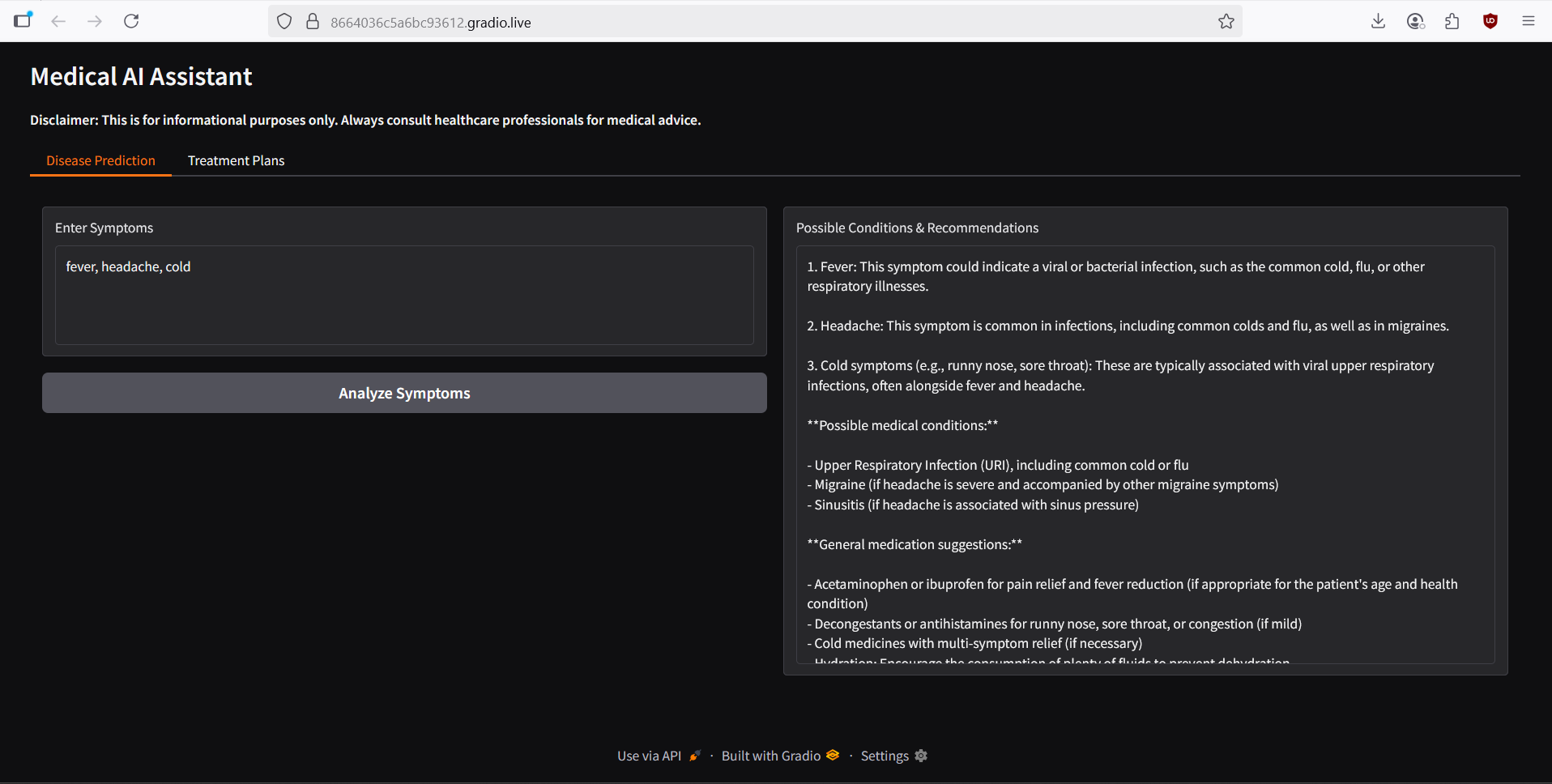
**Functionality**:

This layer provides the front-end interface where users can interact with the application through text boxes and buttons. It manages the input of symptoms and patient information and displays the generated AI responses.

# 10. Testing

* Unit Testing: Check individual functions like AI response generation and data handling.
* Integration Testing: Verify that frontend (HTML/CSS) and backend (Flask + AI) work together.
* Performance Testing: Ensure AI responses are fast and system handles multiple users.
* Debugging: Identify and fix errors in backend, frontend, or AI modules.
* Final Validation: Confirm the app works smoothly before deployment.

# 11. Screenshots



# 12. Known Issues

* High resource usage – Large AI models require lots of RAM/VRAM.
* Slow CPU performance – Responses are slower without GPU.
* Lack of Persistence - The application does not store any user history or data. Each interaction is a new, isolated request.
* AI Model Limitations - The AI may occasionally provide inaccurate, irrelevant, or non-specific information. It is crucial to remember this is a general-purpose model and not a specialized medical tool.

# 13. Future Enhancements

* **Specialized Fine-Tuning**: Fine-tune the language model on a curated medical dataset to improve accuracy and specificity.
* **Persistent User Sessions**: Implement a database (like Firebase Firestore) to store user history (with proper security and anonymization) to provide more contextually aware responses.
* **Mobile and Web Access**: Develop a standalone mobile app or a more robust web interface using frameworks like React or Angular for better user experience.
* **Integration with Medical Databases**: Connect the assistant to external, trusted medical databases for more accurate information.