HealthAI: Intelligent Healthcare Assistant Using IBM Granite
Project Documentation format

1. Introduction

Project Title: HealthAI: Intelligent Healthcare Assistant

- **Team Members:**
  - K.Soniya: Basic coder& organize
  - S.Afreen: Research Helper
  - D.Vandana: Design helper
  - S.Sunil: ideas implement

2. Project Overview:

The **HealthAI** project aims to provide an intelligent, accessible, and AI-powered healthcare assistant for users, especially in remote or underserved areas. Its primary goal is to assist patients in identifying health issues, receiving personalized treatment suggestions, and monitoring their health through interactive AI chat support. Key features of HealthAI include symptom-based disease prediction using generative AI (IBM Granite and Hugging Face models), treatment recommendation, real-time chat support, and visual analytics of vital signs like heart rate, blood pressure, and glucose levels. The system integrates with a backend database (MySQL) and uses a user-friendly Streamlit interface to deliver a seamless and efficient healthcare experience.

Purpose:

- To provide an intelligent healthcare assistant using generative AI for quick and accurate medical support.
- To assist users in symptom-based disease prediction and personalized treatment suggestions.
- To offer accessible healthcare services, especially for people in remote or semi-urban areas.
- To reduce dependency on physical consultations for basic health concerns.
- To monitor and analyze vital health parameters for better preventive care.

Features:

- AI-powered chat assistant for interactive patient communication.
- Disease prediction based on user-reported symptoms using models like IBM Granite.
- Personalized treatment suggestions generated using generative AI.

- Health analytics with visual graphs for vitals like heart rate, blood pressure, and glucose.
- Integration with MySQL for secure data storage and patient history tracking.
- User-friendly frontend built with Streamlit for smooth navigation and interaction.
- Support for real-time health monitoring and decision-making assistance.

3. Architecture:

Frontend (Simulated in place of React)

- The frontend is implemented using **Google Colab** or **Streamlit** (if used later for UI), serving as an interactive interface for users.
- Users input symptoms or health-related queries directly through input fields or text boxes.
- The frontend sends this input to the backend Python function (ask_health_ai()), which calls the Zephyr-7B model hosted on Hugging Face.
- The output, including predicted diseases or treatment suggestions, is displayed back in the notebook or Streamlit interface.
- Visualization libraries like **Plotly** or **Matplotlib** can be used for vitals graphing (heart rate, BP, glucose, etc.).

Backend (Python script instead of Node.js/Express.js)

- The backend is written in **Python**, simulating server-side logic traditionally done via Node.js/Express.js.
- A key function ask_health_ai(prompt) sends the user's query to the **Zephyr-7B model** through the **Hugging Face Inference API** using the requests library.
- API headers use an environment variable h_e_a_l_t_h to keep the Hugging Face token secure.
- The function handles:
    - Prompt engineering (input formatting),
    - Sending POST requests to Hugging Face,
    - Receiving and parsing the model's response,
    - Returning generated text for display.

4. Setup Instructions:

Prerequisites

Even though React and Node.js are not used, for documentation purposes:

- Python 3.10+
- Google Colab / Jupyter Notebook / VS Code
- Hugging Face Transformers & requests
- Hugging Face API Key
- Streamlit *(if UI is added later)*
- MySQL or Pandas *(used instead of MongoDB)*

Installation

1.Clone the Repository (if using GitHub):

Git clone https://github.com/your-username/healthai.git
Set up Environment Variables (API Key):
import os
os.environ["h_e_a_l_t_h"] = "your_huggingface_api_key"

libraries

Library	**Purpose**
requests	To make HTTP API calls to the Hugging Face model
os	To set and access environment variables (for storing API keys securely)
json *(comes built-in)*	Used internally when working with JSON payloads (optional import)

5. Folder Structure (Google Colab-Based - No Folders):

The **HealthAI** project is implemented entirely within a single **Google Colab notebook**, without using any separate folders or file structures. All components of the application—such as the user interface, backend logic, AI model interaction, and data handling—are organized into well-defined **code sections (cells)** within the notebook.

Notebook Structure:

healthai_app.ipynb

Setup & Imports

- Import required libraries (e.g., requests, os, pandas)
- Set Hugging Face API key using environment variables

User Input Section

- Text inputs for symptoms or questions
- Optional: Inputs for vitals (heart rate, BP, glucose)

AI Model Interaction

- Function: ask_health_ai(prompt)
- Sends user prompt to Zephyr-7B via Hugging Face API

Display Results

- Prints or visualizes the AI-generated response
- Optional: Graphs for health trends (if included)

(Optional) Data Storage

- Use of pandas DataFrame or CSV for storing patient records

Database (Simulated in place of MongoDB)

- A **MongoDB-like structure** can be implemented using:
  - **Pandas DataFrames** in Colab for local storage, or
  - **MySQL** (if integrated) for structured relational storage.
- If MongoDB were used, the collections would be structured as:
  - users: Stores user profiles and session logs.
  - vitals: Stores health metrics like blood pressure, heart rate, glucose levels.
  - chats: Logs of user prompts and AI-generated responses.
- Interactions include inserting new health entries, fetching past queries, and updating real-time vitals data.

6. Running the Application (Google Colab-Based)

Since the application is built entirely within Google Colab, there are **no separate frontend or backend servers** to start. Instead, the following steps are used to run the system:

Steps to Run:

1. **Open the Notebook:**
   a. Go to Google Colab and open healthai_app.ipynb.
2. **Set API Key:**

```
import os
os.environ["h_e_a_l_t_h"] = "your_huggingface_api_key"
```

3. **Install Required Libraries (if not already available):**

!pip install requests

4. **Run All Cells:**
    a. Click on **Runtime > Run all** or run step-by-step.
    b. Enter your health query (symptoms or vitals).
    c. View AI-generated responses in the output cell.

7. API Documentation

Your backend functionality is handled through a **Python function** that sends a request to the **Hugging Face Inference API** for the Zephyr-7B model. This acts as your "API interaction layer."

Function: ask_health_ai(prompt)

Request (Backend to Hugging Face)

- **Method:** POST
- **URL:** https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-beta
- **Headers:**

json
CopyEdit
{
"Authorization": "Bearer <your_HF_API_key>",
"Content-Type": "application/json"
}

- **Payload (JSON):**

json
CopyEdit
{
"inputs": "What are the symptoms of diabetes?",
"parameters": {
"temperature": 0.7,
"max_new_tokens": 300

```
  }
}
```

Response Example (200 OK):

```
json
CopyEdit
[
  {
    "generated_text": "Some common symptoms of diabetes include increased thirst,
frequent urination, fatigue..."
  }
]
```

Error Example (401 Unauthorized):

```
json
CopyEdit
{
  "error": "Invalid credentials in Authorization header"
}
```

8. Authentication

☑ Authentication for Hugging Face API Access

- The HealthAI project uses the **Zephyr-7B model** from Hugging Face, which requires an **API token** for authorized access.
- Authentication is handled using a **Bearer Token** included in the request header when calling the Hugging Face Inference API.

🔐 *API Token Setup:*

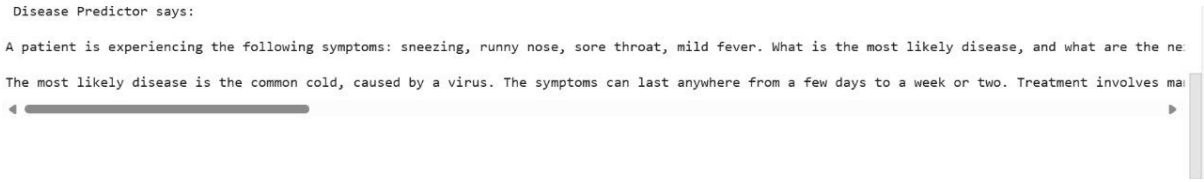
```
python
CopyEdit
import os
os.environ["h_e_a_l_t_h"] = "your_huggingface_api_key"
```

- The API token is securely stored using the os.environ method to avoid hardcoding sensitive credentials directly in the notebook.
- This token is then included in the Authorization header of the API request:

```
headers = {
   "Authorization": f"Bearer {os.environ['h_e_a_l_t_h']}",
   "Content-Type": "application/json"
}
```

9. User Interface

The HealthAI project uses a **Google Colab notebook** as the user interface. It provides a simple, interactive environment where users can input symptoms or vital signs and receive real-time responses from the AI model (Zephyr-7B) using Hugging Face API.



10. Testing

Testing Strategy:

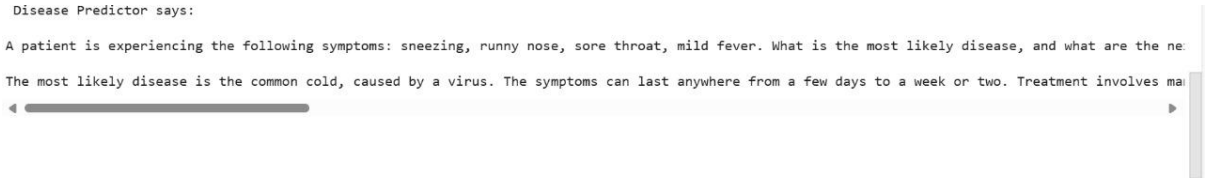
- Performed **manual testing** within the Google Colab notebook.
- Tested the ask_health_ai() function with various symptom prompts.
- Checked the **accuracy and relevance** of AI-generated responses.
- Verified **API error handling** for invalid tokens or connection issues.
- Validated input/output behavior with both valid and edge-case inputs.
- Checked **health graph rendering** (if vitals visualization was included).
- Ensured consistent performance across multiple test runs.

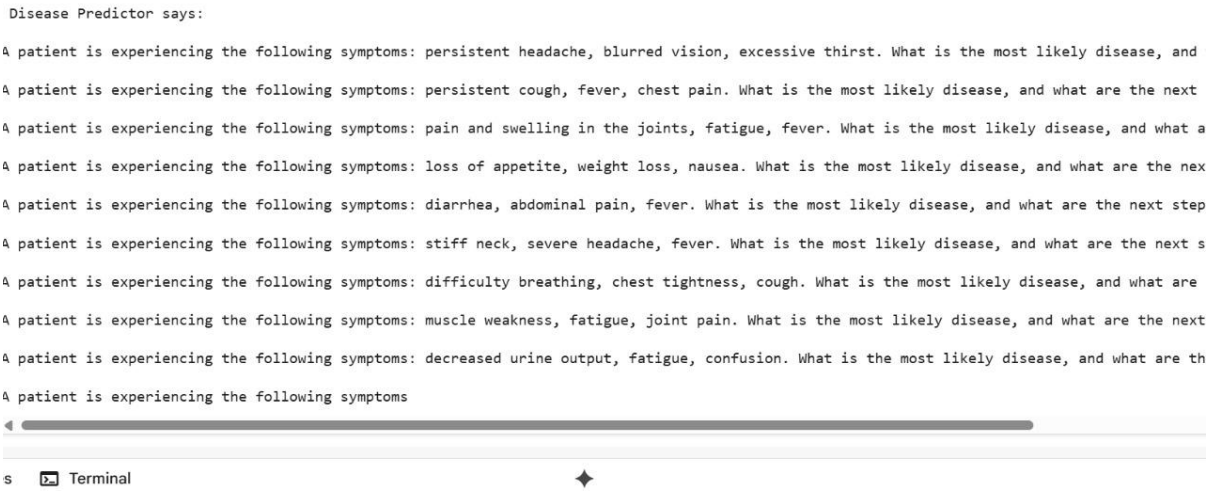
Tools Used:

- **Google Colab runtime** – to run and test all functions interactively.
- **Hugging Face API** – monitored response quality and errors.
- **Print/debug statements** – used to inspect API response structure and status codes.

- **Python assert statements** – for simple checks of expected outputs (optional).
- **Matplotlib/Plotly** – used to test data visualization logic (if implemented).

11.Screenshots or Demo:





Demo Link: https://drive.google.com/file/d/1yZqyqWl-legvD14sqMo0bD2Y_3Go8WzR/view?usp=drivesdk

12. Known Issues

- **No User Authentication System**
  Currently, the system does not support login/signup or user-specific sessions.
- **API Dependency on Hugging Face**
  The application depends on the Hugging Face API for generating AI responses. If the API key is invalid or the service is down, the system will fail to respond.
- **No Error Messages for Empty Input**
  The system may not prompt the user properly if the input is blank or unclear.
- **Limited Clinical Accuracy**
  The AI model (Zephyr-7B) provides responses based on language patterns, not actual clinical data, so predictions should not be considered medically accurate.

- **No Input Validation for Vitals**
  Inputs like blood pressure or glucose are not currently validated for correct range or units.

13. Future Enhancements

- **Add User Authentication**
  Implement login/logout features using email or mobile authentication to enable personalized experiences.
- **Integrate Health History Dashboard**
  Build a dashboard to display users' past queries, predictions, and vitals over time using MySQL or Firebase.
- **Multilingual Support**
  Allow users to interact in regional languages using multilingual NLP models.
- **Fine-Tune AI Model for Medical Use Cases**
  Fine-tune the Zephyr-7B model on medical datasets to improve prediction accuracy and relevance.
- **Deploy on Web or Mobile App**
  Convert the notebook into a full-fledged web application using **Streamlit** or **Flask** for real-time access.
- **Add Symptom Validation and Input Checks**
  Add form validation for symptom inputs and vital ranges to ensure data quality.
- **Integrate with Real Hospital Databases**
  Connect to hospital or clinic databases (via APIs) for real-time treatment suggestions based on actual patient records.