HEALTH AI

PROJECT DOCUMENTATION

1.INTRODUCTION

- Project title: Health AI-Intelligent Healthcare Assistant Using IBM Granite
- Team member: Ashraf Nisha. A
- Team member: Jayashree. S
- Team member: Jerusha Majella. M
- Team member: Rakshana. C

2.PROJECT OVERVIEW

• **Purpose:** Health AI harnesses IBM Watson Machine Learning and Generative AI to provide intelligent healthcare assistance, offering users accurate medical insights. The platform includes a Patient Chat for answering health-related questions, Disease Prediction that evaluates user-reported symptoms to deliver potential condition details, Treatment Plans that provide personalized medical recommendations, and Health Analytics to visualize and monitor patient health metrics.

Scenario 1: A user inputs their symptoms into the Disease Prediction system, describing issues like persistent headache, fatigue, and mild fever. The system analyses the symptoms along with the patient's profile and health data to provide potential condition predictions, including likelihood assessments and recommended next steps.

Scenario 2: A user needs personalized treatment recommendations for a diagnosed condition. By entering their condition in the Treatment Plans generator, the AI processes the information along with patient data to create a comprehensive, evidence-based treatment plan that includes medications, lifestyle modifications, and follow-up testing.

Scenario 3: A user wants insights about their health trends. Using the Health Analytics dashboard, they can visualize their vital signs over time (heart rate, blood pressure, blood glucose, etc.) and receive AI-generated insights about potential health concerns and improvement recommendations.

FEATURES:

Conversational Interface

- Key Point: Natural language interaction
- Functionality: Enables patients and healthcare providers to ask medical queries, receive health updates, and get guidance in simple, easy-to-understand language.

Policy Summarization

- Key Point: Simplified medical guideline understanding
- Functionality: Transforms lengthy healthcare guidelines, medical reports, and research documents into concise, actionable summaries for both patients and healthcare professionals.

Resource Forecasting

- Key Point: Predictive analytics
- Functionality: Estimates future healthcare resource needs such as doctor availability, medicine stock, and patient appointments by analysing historical and real-time medical data.

Health Tip Generator

- Key Point: Personalized healthcare advice
- Functionality**:** Provides daily health tips, wellness suggestions, and lifestyle recommendations based on user health data and behaviour.

Patient Feedback Loop

- Key Point: Patient engagement
- Functionality: Collects and analyses patient feedback to improve healthcare services, enhance diagnosis accuracy, and refine treatment recommendations.

KPI Forecasting

- Key Point: Strategic healthcare planning support
- Functionality: Projects key health performance indicators such as patient recovery rate, hospital resource usage, and treatment efficiency to help healthcare providers track progress and plan ahead.

Anomaly Detection

- Key Point: Early health warning system
- Functionality: Detects unusual patterns in patient symptoms, reports, or health data to identify potential medical risks at an early stage.

Multimodal Input Support

- Key Point: Flexible healthcare data handling
- Functionality: Accepts text, PDFs, and CSVs containing medical reports, prescriptions, and patient records for analysis and forecasting.

Streamlit or Gradio UI

- Key Point: User-friendly interface
- Functionality: Provides an intuitive dashboard for both patients and healthcare professionals to interact with the intelligent assistant easily.

3.ARCHITECTURE

Frontend (Streamlit):
The frontend is developed using Streamlit, providing an interactive web-based UI for healthcare users. It includes multiple pages such as patient dashboards, medical report uploads, AI-driven chat interface, feedback forms, and health report viewers. Navigation is managed through a sidebar with the Streamlit-option-menu library, and each page is modularized to ensure scalability and smooth user interaction.

Backend (FastAPI):
FastAPI acts as the backend REST framework, powering API endpoints for medical document processing, chat-based healthcare interactions, health tip generation, patient report creation, and vector embedding. It is optimized for asynchronous performance and integrates easily with Swagger for testing and documentation.

LLM Integration (IBM Granite):
Granite LLM models from IBM Watsonx are integrated for natural language understanding and response generation. Carefully designed prompts enable the model to summarize medical records, provide healthcare advice, and generate patient-friendly reports.

Vector Search (Pinecone):
Uploaded medical documents and reports are embedded using Sentence Transformers and stored in Pinecone. Semantic search with cosine similarity allows patients and healthcare professionals to retrieve relevant information using natural language queries.

ML Modules (Forecasting and Anomaly Detection):
Lightweight ML models built with Scikit-learn are applied for forecasting patient flow, medicine requirements, and healthcare trends, as well as detecting anomalies in patient data. Time-series data is processed, modeled, and visualized using Pandas and Matplotlib.

4. SETUP INSTRUCTIONS

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- API keys for IBM Granite (Watsonx) and Pinecone
- Internet access to connect with cloud-based healthcare services

Installation Process:

- Clone the project repository
- Install all dependencies listed in requirements.txt
- Create a .env file and configure API credentials (IBM Granite, Pinecone, etc.)
- Start the backend server using FastAPI
- Launch the frontend using Streamlit
- Upload healthcare data and interact with the AI-powered modules

5. FOLDER STRUCTURE

- app/ – Contains all FastAPI backend logic including routers, healthcare models, and integration modules.

- app/api/ – Subdirectory for modular API routes such as medical chat, patient feedback, health report generation, and medical document vectorization.
- ui/ – Contains frontend components for Streamlit pages, health dashboards, card layouts, and patient form UIs.
- smart_dashboard.py – Entry script for launching the main Streamlit healthcare dashboard.
- granite_llm.py – Manages communication with IBM Granite model for medical summarization, patient Q&A, and chat.
- document_embedder.py – Converts medical documents and patient reports into embeddings and stores them in Pinecone.
- kpi_file_forecaster.py – Forecasts future healthcare trends such as patient inflow, medicine demand, and doctor availability using regression.
- anomaly_file_checker.py – Detects unusual patterns or anomalies in patient health data.
- report_generator.py – Generates AI-based healthcare reports and summaries for patients and healthcare providers.

6. RUNNING THE APPLICATION

To start the project:

- launch the FastAPI server to expose backend healthcare endpoints.
- Run the Streamlit dashboard to access the interactive web interface.
- Navigate through different pages using the sidebar menu.
- Upload medical documents or patient CSV files, interact with the AI healthcare chat assistant, and view outputs such as health reports, summaries, and predictions.
- All interactions run in real-time, with backend APIs dynamically updating the frontend for a smooth healthcare experience.

Frontend (Streamlit):

The frontend is developed using Streamlit, providing an interactive web UI for healthcare applications. It includes multiple pages such as patient dashboards, medical file uploads, AI-powered chat interface, feedback forms, and health report viewers. Navigation is managed through a sidebar using the Streamlit-option-menu library, and each page is modularized to ensure scalability.

Backend (FastAPI):
FastAPI serves as the backend REST framework, handling API endpoints for medical document processing, healthcare chat interactions, health tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and integrates seamlessly with Swagger for testing and documentation.

7. API DOCUMENTATION

Backend APIs available include:

POST /chat/ask – Accepts a patient query and responds with an AI-generated healthcare message.
POST /upload-doc – Uploads and embeds medical documents or patient records in Pinecone.
GET /search-docs – Returns semantically similar medical documents or reports based on the input query.
GET /get-health-tips – Provides personalized healthcare and wellness tips for users.
POST /submit-feedback – Stores patient feedback for further review and healthcare service improvement.
Each endpoint is tested and documented in **Swagger UI** for quick inspection and trial during development.

8. AUTHENTICATION

In this project version, the application runs in an open environment for demonstration purposes. However, for secure deployments, authentication methods can be integrated such as:

- Token-based authentication (JWT or API keys)
- OAuth2 with IBM Cloud credentials
- Role-based access (admin, doctor, patient)

Planned enhancements include secure user sessions and health history tracking to provide a personalized healthcare experience.

9. USER INTERFACE

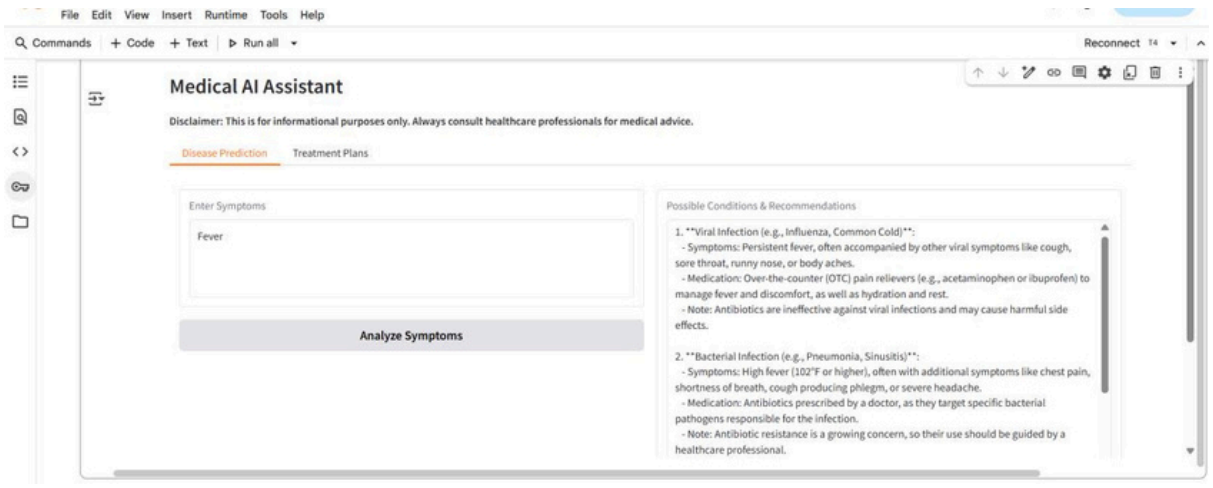
The interface is minimalist and functional, designed to be accessible for non-technical healthcare users. It includes a sidebar for navigation, KPI visualizations with summary cards, tabbed layouts for chat, health tips, and patient data forecasting, real-time form handling, and the ability to download PDF health reports. The design emphasizes clarity, speed, and user guidance with help texts and intuitive workflows.

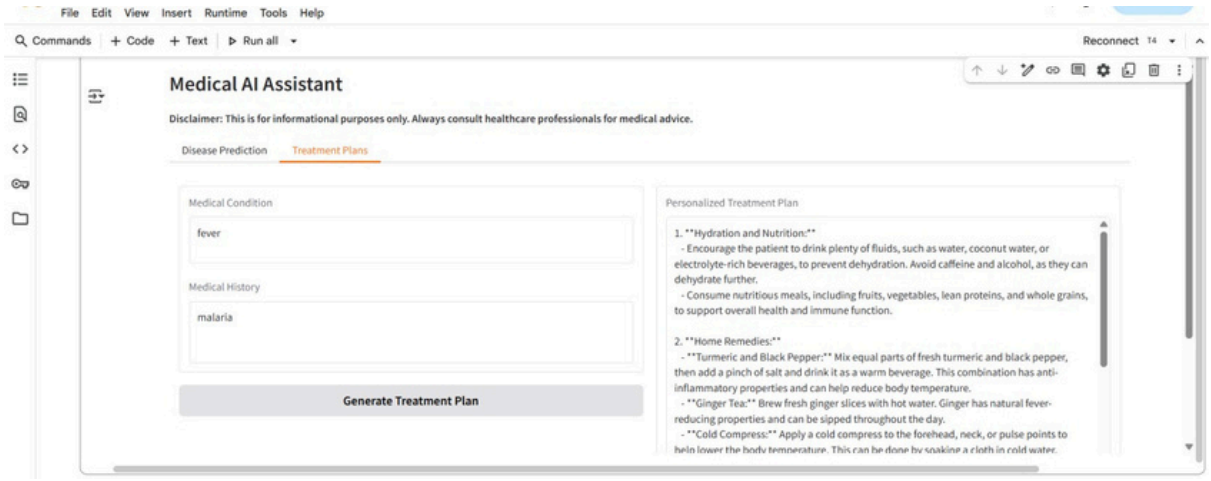
10. TESTING

Testing was conducted in multiple phases for the healthcare assistant project:

Unit Testing: Tested prompt engineering functions, document processing, and utility scripts.
API Testing: Performed via Swagger UI, Postman, and automated test scripts to ensure all endpoints respond correctly.
Manual Testing: Verified file uploads, AI chat responses, report generation, and output consistency.
Edge Case Handling: Managed malformed inputs, large medical files, and invalid API keys.
All functions were validated to ensure reliability in both offline and API-connected modes.

11. SCREENSHOT





12. Known Issues

- Large medical documents may take longer to process and embed.
- Some rare or complex medical queries may produce less accurate AI responses.
- Real-time dashboard updates can experience minor delays under heavy load.
- Limited error handling for corrupted or unsupported file formats.
- Authentication and role-based access are not fully implemented in the demo version.
- Integration with external healthcare databases is not yet live.

13. Future Enhancements

- Implement full **role-based authentication** and secure user sessions for patients, doctors, and admins.
- Integrate with **external healthcare databases** and electronic health records (EHR) for real-time data access.
- Improve AI model accuracy for **complex medical queries** and personalized health recommendations.
- Enable **multi-language support** for broader accessibility.

- Add **mobile-friendly interface** for on-the-go patient and doctor interaction.
- Introduce **analytics dashboards** for hospital management to monitor trends and resources.
- Incorporate **automated notifications and alerts** for appointments, medicine schedules, and critical health anomalies.