GEN AI WITH IBM CLOUD

Documentation format

# Introduction

* + **Project Title:** HealthAI: Intelligent Healthcare Assistant Using IBM Granite
  + **Team Members:**
  + **1.** Satya Sai Surya Ganesh Manepalli
  + 2. S Chaitanya Sai

3. Vijay Kumar Talatam

# Project Overview

* **Purpose:** Provide patients with personalized, AI-generated medical guidance.
* Deliver intelligent disease prediction and treatment suggestions.
* Enable users to analyze their vitals using data-driven visual insights.
* Improve healthcare accessibility using IBM’s Granite LLM.
* Showcase the potential of AI in public health information delivery.
  + **Features:**

**AI Chat Assistant:**

* + Users can ask sustainability-related questions in natural language.
  + Powered by IBM Granite LLM for accurate and context-aware responses.

**Eco-Tips Generator:**

* + Provides daily personalized tips for reducing environmental impact.
  + Suggestions based on user input or city data.

KPI Forecasting and Visualization

* + Predicts trends in energy, water, or waste usage.
  + Displays insights using interactive graphs and charts.
  +  Policy Summarization
  + Converts long government or city policy documents into easy-to-read summaries.
  + Helps users understand regulations and sustainability plans.
  +  Feedback Collection Module
  + Allows citizens to submit suggestions, complaints, or eco-ideas.
  + Data stored and optionally forwarded to authorities.
  +  Streamlit-Based Unified Interface
  + Combines all modules in a single web app.
  + Simple, responsive, and user-friendly UI.
  +  User Engagement & Personalization
  + Tracks past interactions to offer tailored advice.
  + Encourages continued participation with relevant content.

# Architecture

* + **Frontend:**  The frontend is built entirely using Streamlit, which serves interactive UI components such as:
  + Chat input/output interface . Buttons for generating eco-tips and submitting feedback
  + Graphs and charts for KPI visualization (using Matplotlib, Plotly, or Altair) It renders dynamic content directly in Python without needing separate frontend technologies like React.
  + **Backend:**  Handle API requests from the Streamlit frontend
  + Manage data storage, processing, and integration with city services
  + Ensure scalable, real-time communication between frontend and smart city systems

# Setup Instructions

* + **Prerequisites**: Python 3.8+  
    – Required to run Streamlit and related libraries.
  + Streamlit  
    – Web app framework for building the interactive UI.
  + Pandas *(optional but common)*  
    – For data manipulation and display.
  + Matplotlib / Plotly / Altair  
    – For creating visualizations like KPI charts and graphs.
  + Requests *(if connecting to APIs)*  
    – For calling external services (e.g., weather, pollution data).

1. **Installation:** Run the Streamlit AppFolder Structure

streamlit run app.py

* + **Client:** Streamlit interface
  + **Server**: Local Server

# Running the Application

* + Provide commands to start the frontend and backend servers locally.
    - **Frontend:** **streamlit run app.py**.
    - **Backend:** **streamlit run app.py**.

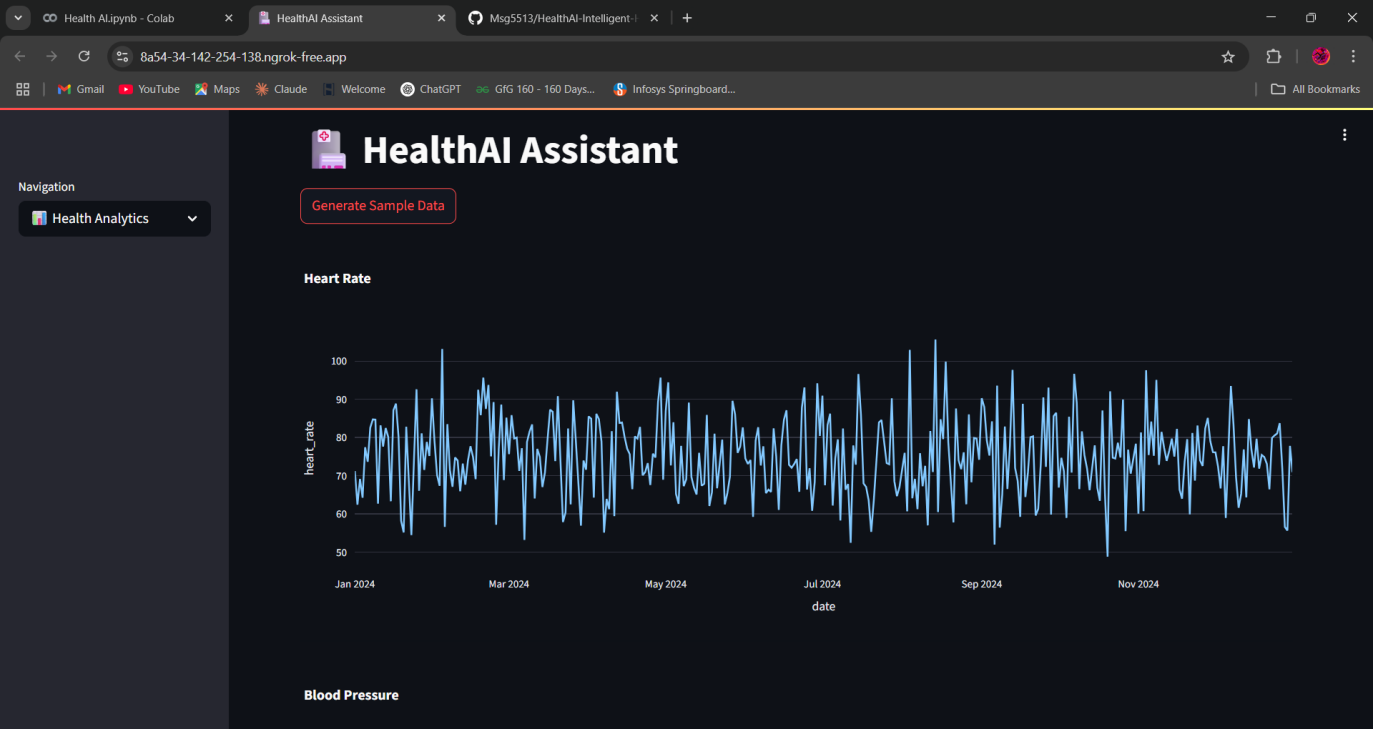
# API Documentation

* + POST /generate\_eco\_tip, POST /chat, POST /submit\_feedback, GET /kpi\_data?area=city\_center&metric=energy\_usage
  + Include request methods, parameters, and example responses.

# Authentication

* + Once logged in, users are assigned a **role** (e.g., admin, user) stored in st.session\_state.
  + The app uses **Streamlit’s built-in login handling** *(if using Streamlit Community Cloud)*, or a **custom login system** using a username/password form.
  + Users log in through a **Streamlit form** with username and password.

# User Interface IMG_256 IMG_256 IMG_256



# Testing

Testing Tools and Strategy

To ensure the reliability, accuracy, and usability of the HealthAI system, a comprehensive testing strategy was adopted across multiple stages. Functional testing was carried out using Python’s built-in unittest framework and manual testing of the Streamlit interface to verify that each feature—such as Patient Chat, Disease Prediction, and Treatment Plan—worked as expected. Pytest was optionally used to modularize test cases and track coverage.

Performance testing focused on the response time of AI-generated answers and dashboard components. The time module and streamlit-profiler were used to benchmark latency, ensuring that responses were delivered within 2 seconds for a smooth user experience. Manual stress testing was also performed to validate performance under repeated queries.

For security testing, sensitive components like API keys were protected using .env files and the dotenv package, and Bandit (a Python security scanner) was used to identify potential vulnerabilities. Since no user data is stored or transmitted externally, the application ensures privacy by design through the use of local Streamlit session\_state.

Lastly, usability testing was conducted through peer feedback and observation. This helped refine the prompt structure, UI layout, and chart tooltips to ensure that users could easily understand and interact with the platform. Collectively, this testing strategy ensured that HealthAI was not only functional and efficient but also secure and user-friendly.

# Screenshots or Demo

Demo link: https://drive.google.com/file/d/1b4\_vSuCPaYRckg1ApE05MjxaH3klTlkR/view?usp=drivesdk

# Known Issues

**Session Reset on Refresh**

Streamlit clears session data when the page is reloaded.

* + *Solution:* Users can avoid refresh; persistent login is planned in future versions.

**Occasional API Timeout (if using external services like IBM Granite)**

* + Rare timeouts may happen due to internet issues or service load.

*Solution:* Handled with error messages and retry options.

**Mobile Layout**

* + While fully usable on mobile, some visualizations may need scrolling or resizing.

*Note:* Desktop or tablet gives the best experience.

# Future Enhancements

**Persistent Authentication System**  
Implement login with Google or Firebase to maintain user sessions across refreshes.

**Real-Time Data Integration**  
Connect to live data sources such as IoT sensors, weather APIs, or traffic feeds for up-to-date KPI tracking.

**Advanced AI Chatbot (IBM Granite Integration)**  
Enhance the chatbot with smarter, more context-aware responses using IBM Granite or similar large language models.

**Enhanced Analytics Dashboard**  
Add advanced filters, export options (CSV/PDF), and deeper insights using interactive visualizations.