Project Documentation: Healthcare AI Assistant

1. Introduction

Project Title: Healthcare AI Assistant using IBM Granite
Team Size : 1
Team ID : LTVIP2025TMID32662
Team Members : Bharath Byreddi - Generative -AI- Developer

2. Project Overview

Purpose: The primary goal of this project is to create an intelligent, user-friendly web application that leverages a powerful Large Language Model (LLM) from IBM's watsonx.ai platform. The assistant provides preliminary health-related information to users in a safe and structured manner, acting as an educational tool rather than a diagnostic one.
Features: The application offers three core functionalities:
Symptom Checker: Users can describe their symptoms in natural language, and the AI provides a list of potential associated conditions for consideration.
Prescription Analysis: The AI analyzes prescription text to extract key information like drug name, dosage, and frequency into a simple summary.
Diet Recommendation: Based on a user's stated health goals or conditions, the AI provides general dietary suggestions and a sample meal plan.

3. Architecture

Frontend: The user interface is built with standard HTML5 and styled with a custom CSS stylesheet. It operates as a classic server-side rendered application, where the Jinja2 templating engine dynamically generates the HTML on the backend. No client-side JavaScript is required for the core functionality.
Backend: The backend is a high-performance API server built with FastAPI, a modern Python web framework, running on a Uvicorn ASGI server. It uses a modular structure, with logic for each core functionality separated into its own Python file for better organization and scalability.
AI Service (Instead of a Database): This project does not use a traditional database. Its intelligence layer is powered by IBM watsonx.ai. The backend communicates with this

platform via secure REST API calls, facilitated by the ibm-watson-machine-learning Python SDK, to get responses from the ibm/granite-3-2b-instruct model.

---

4. Setup Instructions

Prerequisites:
Python 3.10 or higher
An active IBM Cloud Account

Installation:
Clone the Repository:
Generated bash
```
git clone [your-repository-url]

cd healthcare-assistant
```
content_copy
download

Create and Activate a Virtual Environment:
Generated bash
```
# Create the environment

python -m venv .venv

# Activate on Windows (PowerShell)

.\.venv\Scripts\activate
```

Install Dependencies: Install all required Python packages from the requirements.txt file.
Generated bash
```
pip install -r requirements.txt
```

Set Up Environment Variables:
Create a file named .env in the root project directory.
Add your IBM Cloud credentials to this file:
Generated code
```
IBM_CLOUD_API_KEY="your_api_key_here"
```

WATSONX_PROJECT_ID="your_project_id_here"

WATSONX_URL="https://us-south.ml.cloud.ibm.com"

---

5. Folder Structure

The project is organized to separate concerns, making it easy to navigate and maintain.

Frontend-related files (
templates/: Contains the index.html Jinja2 template.
static/: Contains the style.css stylesheet for the UI.

Server (
main.py: The main FastAPI application file that defines routes and handles requests.
functionalities/: Each core feature (symptom checker, etc.) has its own file here, containing the business logic.
watsonx_client.py: A dedicated module for handling all communication with the IBM watsonx.ai API.
prompts.py: Stores and formats the prompts sent to the AI model.

---

6. Running the Application

Provide commands to start the backend server locally. Since this is not a separate client/server architecture like MERN, there is only one command.
Backend & Frontend Server:
Generated bash

```
# Make sure your virtual environment is active

# From the root directory, run:

python -m uvicorn app.main:app --reload
```

Open a web browser and navigate to http://127.0.0.1:8000.

---

7. API Documentation

The backend exposes two routes on a single endpoint.

Endpoint: /
Method: GET
Description: Renders and returns the main index.html page.
Response: 200 OK with HTML content.

Method: POST
Description: Processes user input from the three-card form. It determines which function was requested and calls the appropriate AI handler.
Request Body (Form Data):
functionality (string, required): The name of the function to execute (e.g., symptoms_checker).
user_input_symptoms (string, optional): The text from the symptom checker textarea.
user_input_prescription (string, optional): The text from the prescription analysis textarea.
user_input_diet (string, optional): The text from the diet recommendation textarea.

Response: 200 OK with the re-rendered index.html page, now including the AI's response.

---

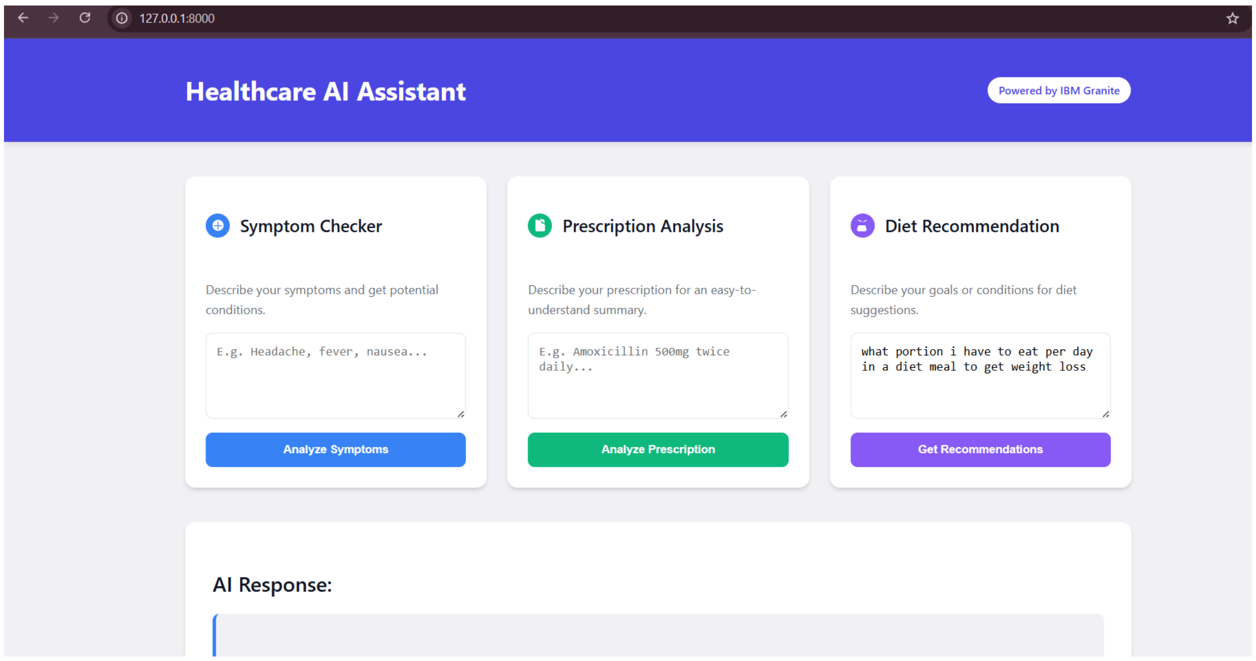
8. Authentication

Application Authentication: The web application does not have user authentication (login/password systems). It is open for public use.
API Service Authentication: Authentication to the backend IBM watsonx.ai service is handled securely. The application loads a secret IBM Cloud API Key from a .env file, which is kept out of version control. This key is sent with every API request to IBM to authorize the use of the Granite foundation model.

---

9. User Interface

The UI is designed to be clean, intuitive, and professional. It features a responsive three-card layout where each card corresponds to a core function, providing an intuitive experience for the user on both desktop and mobile devices.


127.0.0.1:8000
Healthcare AI Assistant
Powered by IBM Granite
Symptom Checker
Describe your symptoms and get potential conditions.
E.g. Headache, fever, nausea...
Analyze Symptoms
Prescription Analysis
Describe your prescription for an easy-to-understand summary.
E.g. Amoxicillin 500mg twice daily...
Analyze Prescription
Diet Recommendation
Describe your goals or conditions for diet suggestions.
what portion i have to eat per day in a diet meal to get weight loss
Get Recommendations
AI Response:

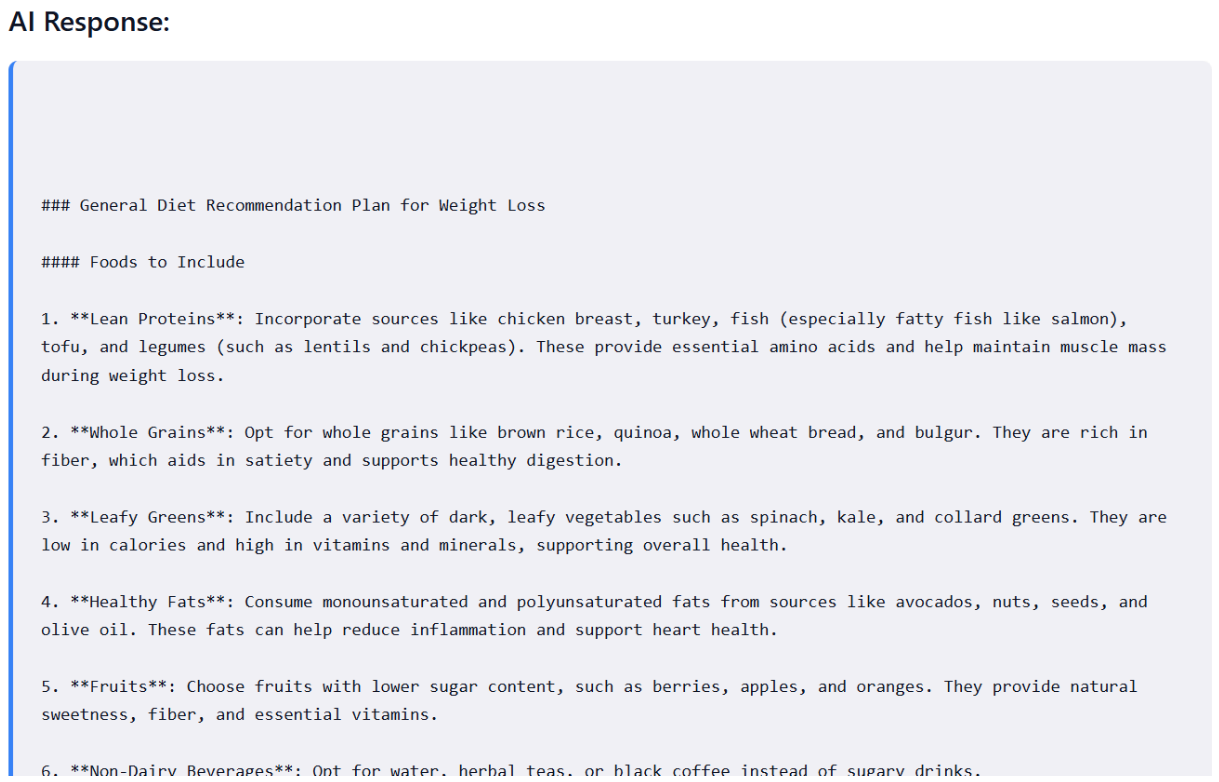
---

10. Testing

The primary method of testing was manual, end-to-end testing. This involved:
Starting the server and loading the web page in multiple browsers.
Testing each of the three functionalities with a variety of valid, invalid, and empty inputs to ensure proper error handling and response generation.
Verifying that the AI responses were relevant, correctly formatted, and included the safety disclaimer.
Ensuring the UI updated correctly after form submission and remained visually appealing.

---

11. Screenshots or Demo


AI Response:

```
### General Diet Recommendation Plan for Weight Loss

#### Foods to Include

1. **Lean Proteins**: Incorporate sources like chicken breast, turkey, fish (especially fatty fish like salmon),
tofu, and legumes (such as lentils and chickpeas). These provide essential amino acids and help maintain muscle mass
during weight loss.

2. **Whole Grains**: Opt for whole grains like brown rice, quinoa, whole wheat bread, and bulgur. They are rich in
fiber, which aids in satiety and supports healthy digestion.

3. **Leafy Greens**: Include a variety of dark, leafy vegetables such as spinach, kale, and collard greens. They are
low in calories and high in vitamins and minerals, supporting overall health.

4. **Healthy Fats**: Consume monounsaturated and polyunsaturated fats from sources like avocados, nuts, seeds, and
olive oil. These fats can help reduce inflammation and support heart health.

5. **Fruits**: Choose fruits with lower sugar content, such as berries, apples, and oranges. They provide natural
sweetness, fiber, and essential vitamins.

6. **Non-Dairy Beverages**: Opt for water, herbal teas, or black coffee instead of sugary drinks.
```

---

12. Known Issues

The SyntaxWarning messages from the ibm-watson-machine-learning library appear in the terminal during startup. These are harmless, originate from the library itself, and do not affect functionality.
As an AI-based service, the quality of the output is highly dependent on the quality and clarity of the user's input. Vague inputs may result in generic or less helpful responses.

---

13. Future Enhancements

Asynchronous JavaScript (Fetch): Re-architect the frontend to use JavaScript fetch calls. This would allow each card to be submitted independently without reloading the entire page, creating a smoother User Experience.
User Accounts: Implement a user login system to save a history of a user's queries and responses.
Streaming Responses: Implement response streaming so the user can see the AI's answer appear word-by-word instead of waiting for the full response to be generated.