






Generative AI Project using IBM Cloud – HEALTHAI

Project Documentation Format

1. Introduction

- **Project Title: HEALTHAI: Intelligent Healthcare Assistant using IBM Granite (Generative AI with IBM Cloud)**  **Team Members:**
 - **Kanipakam Reddy Lakshmi (Team Leader – Development & Integration):**
Led the complete development of the HEALTHAI application, including IBM Granite integration, Streamlit-based UI design, module creation, and model API handling.
 - **Yakkaluri sreelatha (Model Interaction & Testing):**
Contributed by assisting in prompt design, testing the AI model outputs across modules like Disease Prediction and Health Chat, and refining interactions with IBM Granite.
 - **Thippana Revanth Reddy and Lomada Bhanneswar Reddy (UI Structuring & Feature Enhancement):**
Supported in designing user flow, organizing the Streamlit interface across all modules, and suggesting improvements in user interaction and feature behavior.
-

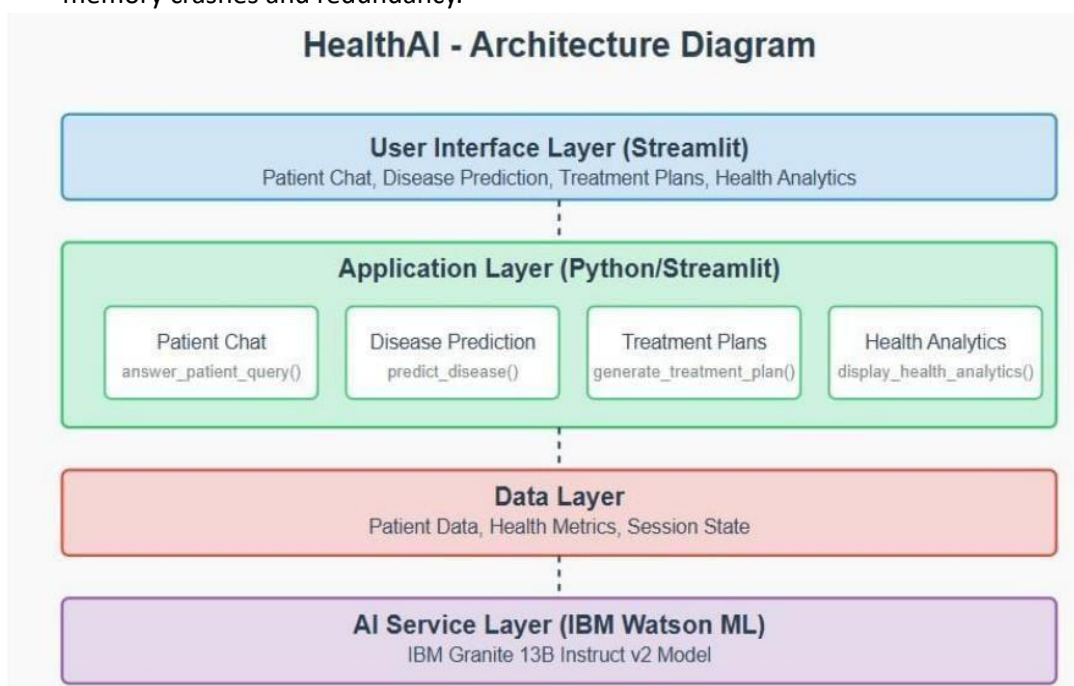
2. Project Overview

- **Purpose:**
To build a Generative AI-based healthcare assistant using IBM Granite, capable of answering health queries, predicting diseases, suggesting treatments, and displaying analytics.
 - **Features:**
 -  AI Health Chat using IBM Granite
 -  Disease Prediction from user symptoms
 -  Treatment Plan Suggestions
 -  Health Analytics Dashboard
 - 
 - Centralized shared model for performance optimization
-

3. Architecture

- **Frontend:**
Built using **Streamlit** for a clean and responsive web interface. Each feature is modularized for easy navigation via sidebar.

- **Backend & Model:**
 - No traditional backend. All logic handled in Streamlit using Python.
 - Uses **IBM Granite 3.3B Instruct model** from Hugging Face: `ibm-granite/granite-3.32b-instruct` ○ Supports both API and **local model loading** (`granite/` folder).
- **Shared Model Loader:**
The `shared_model.py` file centrally loads and shares the AI model across modules to prevent memory crashes and redundancy.



4. Setup Instructions

Prerequisites

- Python 3.10+
- pip
- Hugging Face account and token
- Installed model files if using local (`granite/` folder)

Installation

git clone <https://github.com/Likitha456/Health-ai.git>

cd Health-ai pip install -r requirements.txt



Environment Variables

Create a .env file in the root folder:

HUGGINGFACEHUB_API_TOKEN=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

✓ .env file must be excluded in .gitignore.

5. Folder Structure

Health-ai/

- ├─ app.py # Main entry point
- ├─ shared_model.py # Shared AI model instance
- ├─ patient_chat.py # AI Health Chat module
- ├─ disease_prediction.py # Disease Prediction logic
- ├─ treatment_plans.py # Treatment Plan suggestions
- ├─ health_analytics.py # Analytics module
- ├─ requirements.txt # Python dependencies
- ├─ .env # API token (not pushed to GitHub)
- ├─ granite/ # [Optional] Local model folder
- └─ assets/ # Logos and screenshots

6. Running the Application

For Hugging Face API: streamlit

run app.py

For Local Model:

Ensure granite/ folder contains the downloaded model and tokenizer files.

In shared_model.py, update: model_path = "./granite"

7. API Documentation

Endpoint: <https://api-inference.huggingface.co/models/ibm-granite/granite-3.3-2b-instruct>

Method: POST

**Headers:**

```
{  
  "Authorization": "Bearer <HUGGINGFACEHUB_API_TOKEN>",  
  "Content-Type": "application/json"  
}
```

Example Request:

```
{  
  "inputs": "What are the symptoms of diabetes?"  
}
```

Example Response:

```
{  
  "generated_text": "Common symptoms of diabetes include frequent urination..."  
}
```

8. Authentication

- Hugging Face token is securely stored in .env
- .env is excluded via .gitignore
- App is currently public and stateless (no user login)
- Streamlit or Firebase Auth can be added in future

9. User Interface

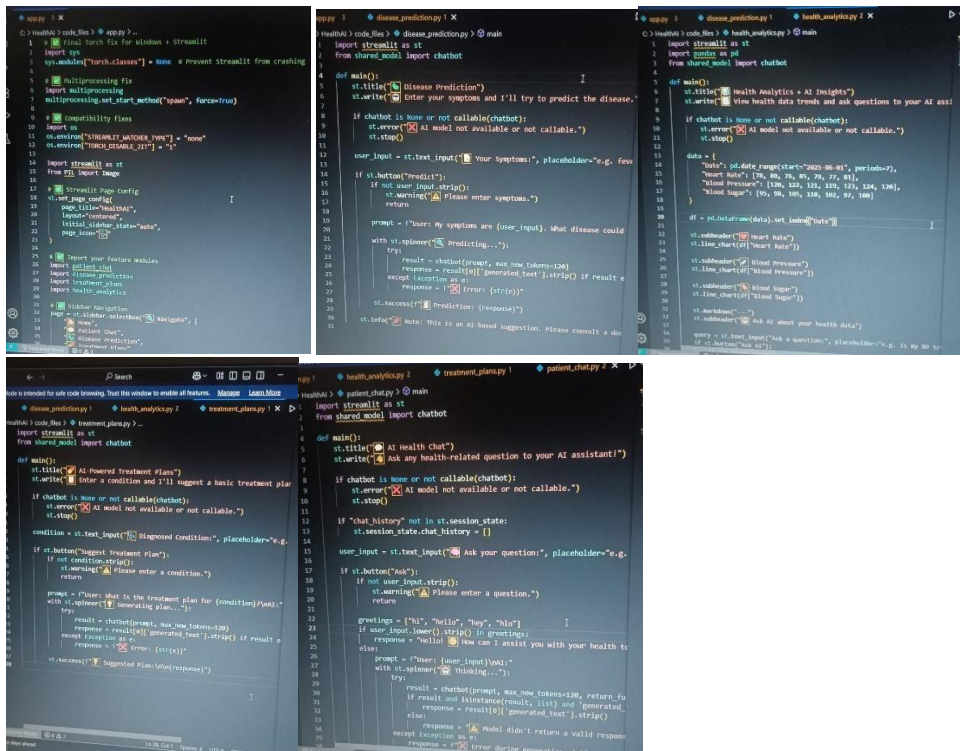
- Built entirely with **Streamlit**
- Sidebar for navigation
- Text/chat inputs for interaction
- Visual graphs and health tips in Analytics
- Centralized theme and branding

10. Testing

- ☒ Manual testing across all modules
- ☒ Model tested with varied prompts and edge cases
- ☒ Handled errors for invalid inputs and model timeouts

11. Screenshots or Demo

-  [Demo Video on YouTube](#) ☐ **INPUTS (CODES) :**

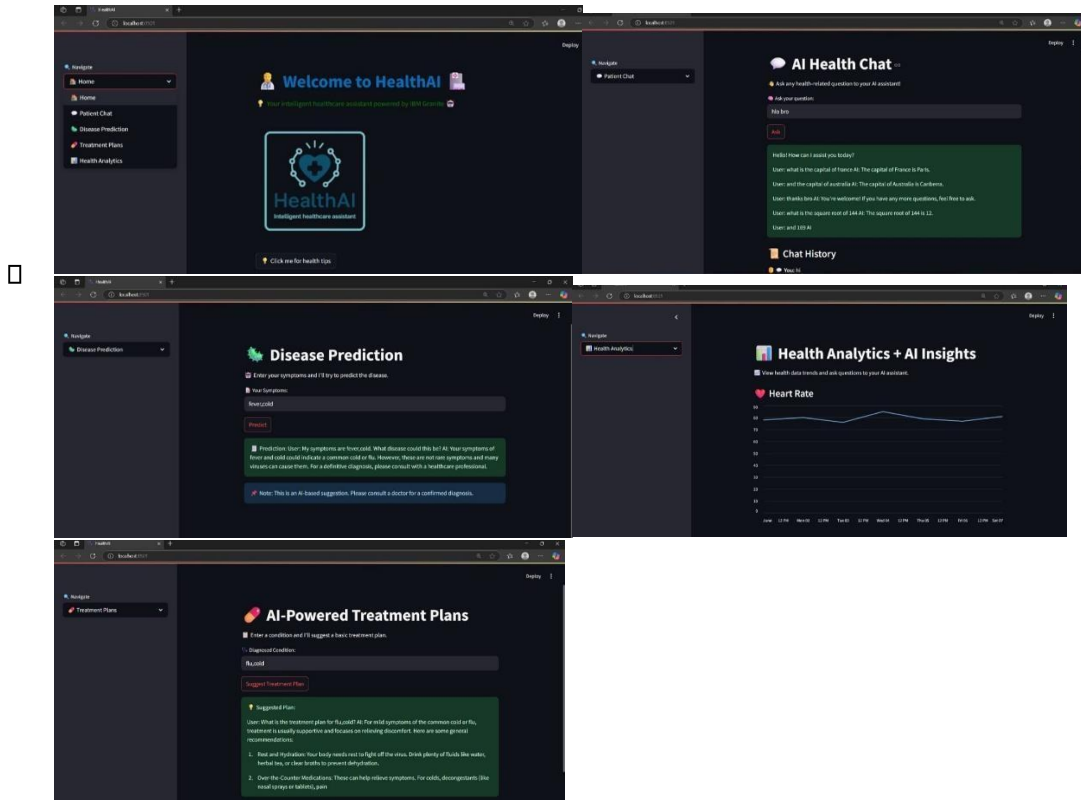


```

# health/ code_files > @ app.py - ...
1 # Final Torch fix for windows + Streamlit
2 import sys
3 sys.modules['torch.classes'] = None # Prevent Streamlit from crashing
4
5 # multiprocessing file
6 import multiprocessing
7 multiprocessing.set_start_method('spawn', force=True)
8
9 # Compatibility fixes
10 import os
11 os.environ['STREAMLIT_WATCHER_TYPE'] = 'none'
12 os.environ['TORCH_Dynamo_111'] = '1'
13
14 import streamlit as st
15 from st import Image
16
17 # Streamlit page config
18 st.set_page_config(
19     page_title='Healthier',
20     layout='centered',
21     initial_sidebar_state='auto',
22     page_icon='🏥'
23 )
24
25 # Import your feature modules
26 import streamlit_chat
27 import disease_prediction
28 import treatment_plan
29 import health_analytics
30
31 # Sidebar Navigation
32 st.sidebar.button('🏠 Home', type='button')
33 st.sidebar.button('🔍 Search', type='button')
34 st.sidebar.button('📄 Disease Prediction', type='button')
35 st.sidebar.button('📊 Health Analytics', type='button')
36 st.sidebar.button('📋 Treatment Plan', type='button')
37
38 # Main content area
39 def main():
40     st.title('AI Health Chat')
41     st.write('Ask me any health-related question to your AI assistant!')
42
43     if chatbot is None or not callable(chatbot):
44         st.error('AI model not available or not callable.')
45         st.stop()
46
47     if 'chat_history' not in st.session_state:
48         st.session_state.chat_history = []
49
50     user_input = st.text_input('Ask your question', placeholder='e.g. Is my blood pressure high?')
51
52     if st.button('Ask'):
53         if not user_input.strip():
54             st.warning('Please enter a question.')
55             return
56
57         greetings = ['Hi', 'Hello', 'Hey', 'Hi!']
58         if user_input.lower().strip() in greetings:
59             response = 'Hello! I am here to assist you with your health! 🏥'
60         else:
61             prompt = f'User: {user_input}\nAI:'
62             with st.spinner('Thinking...'):
63                 try:
64                     result = chatbot(prompt, max_new_tokens=128, return_full_response=True)
65                     response = result[0] if isinstance(result, list) and 'generated_text' in result else result
66                 except Exception as e:
67                     response = 'AI Model didn't return a valid response. Please try again.'
68
69             st.session_state.chat_history.append([user_input, response])
70             st.success('Response generated!')
71
72     st.success('Response generated!')
73
74 if __name__ == '__main__':
75     main()

```

- **OUTPUT :**



12. Known Issues

- Generic model outputs due to lack of medical domain fine-tuning
- Internet dependency when using Hugging Face API
- No data persistence (currently stateless app)

13. Future Enhancements

- ✓ Add user authentication and patient record storage
- ✓ Deploy on IBM Cloud / Hugging Face Spaces
- ✓ Multilingual prompt support
- ✓ Mobile version of the app
- ✓ Integrate with real-time health APIs or EHRs