

---

## Generative AI Project using IBM Cloud – HEALTHAI






### Project Documentation Format

---

#### 1. Introduction

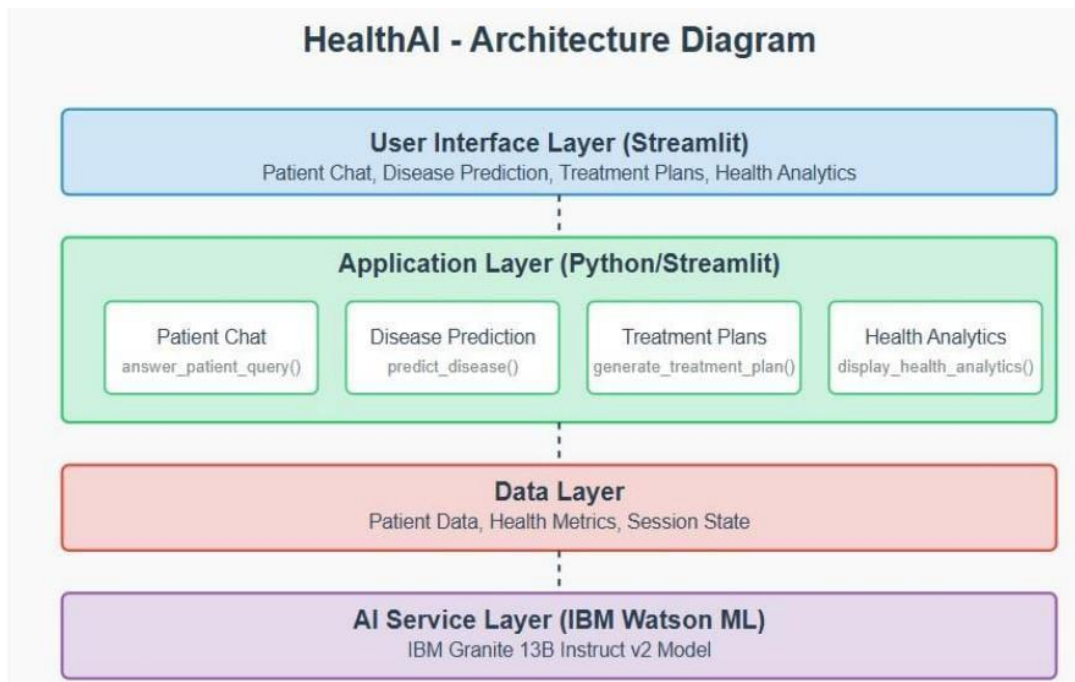
- **Project Title: HEALTHAI: Intelligent Healthcare Assistant using IBM Granite (Generative AI with IBM Cloud)**
  - **Team Members:**
    - **Likitha Putta Reddy (Team Leader – Development & Integration):**  
Led the complete development of the HEALTHAI application, including IBM Granite integration, Streamlit-based UI design, module creation, and model API handling.
    - **Velakaturi Lekhya Sreeya (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.
    - **Sama Pavithra (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-3.3-2b-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

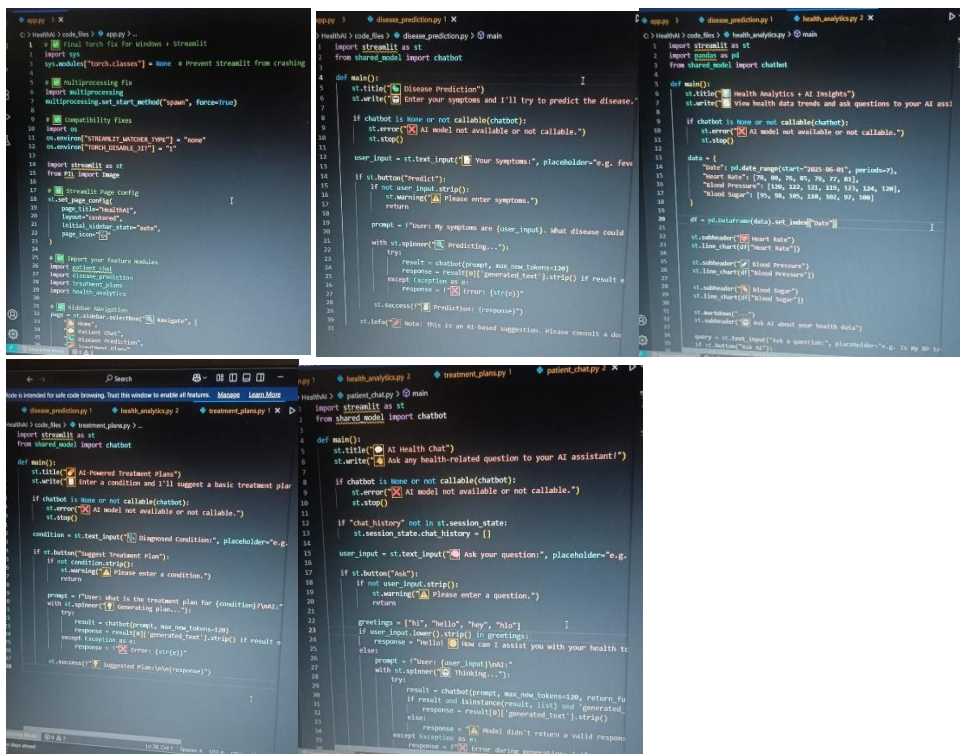
- Centralized theme and branding

## 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 ) :

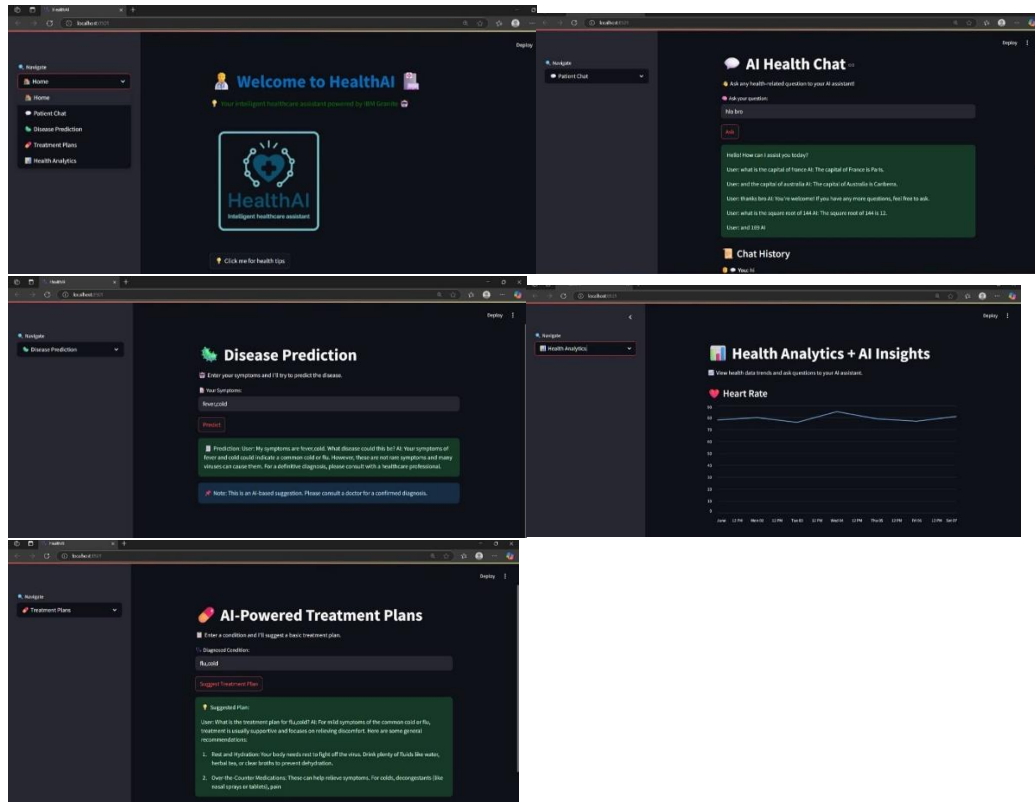


```

# health.py code file
1 # Final torch fix for windows + streamlit
2 import sys
3 sys.modules['torch.classes'] = None # prevent streamlit from crashing
4
5 # multiprocessing fix
6 import multiprocessing
7 multiprocessing.set_start_method('spawn', force=True)
8
9 # compatibility fix
10 import os
11 os.environ['PYTHONUNBUFFERED'] = "true"
12 os.environ['PYTHONIOENCODING'] = "utf-8"
13
14 import streamlit as st
15 from st import import_image
16
17 # Streamlit page config
18 st.set_page_config(
19     page_title="HealthBot",
20     layout="centered",
21     initial_sidebar_state="auto",
22     page_icon="🏥"
23 )
24
25 # Import your feature modules
26 import chatbot
27 import disease_prediction
28 import treatment_plans
29 import health_analytics
30
31 # Sidebar navigation
32 page = st.sidebar.selectbox("🏠 Navigation", [
33     "Home",
34     "Disease Prediction",
35     "Treatment Plans",
36     "Health Analytics"
37 ])
38
39 # Main content area
40 if page == "Home":
41     st.write("Welcome to the HealthBot interface. Select a module from the sidebar to get started.")
42 elif page == "Disease Prediction":
43     # disease_prediction.py code
44     def main():
45         st.title("Disease Prediction")
46         st.write("Enter your symptoms and I'll try to predict the disease.")
47
48         if chatbot is None or not callable(chatbot):
49             st.error("AI model not available or not callable.")
50             st.stop()
51
52         user_input = st.text_input("Your symptoms, placeholder='e.g. fever'")
53
54         if st.button("Predict"):
55             if not user_input.strip():
56                 st.warning("Please enter symptoms.")
57                 return
58
59             prompt = f"User: My symptoms are {user_input}. What disease could it be?"
60             with st.spinner("Predicting..."):
61                 try:
62                     result = chatbot(prompt, max_new_tokens=128)
63                     response = result[0][generated_text.strip()] if result else ""
64                     except Exception as e:
65                         response = f"Error: {str(e)}"
66
67             st.success(f"Prediction: {response}")
68
69         st.info("Note: This is an AI-based suggestion. Please consult a doctor for a proper diagnosis.")
70
71     if __name__ == "__main__":
72         main()
73 elif page == "Treatment Plans":
74     # treatment_plans.py code
75     def main():
76         st.title("AI-Powered Treatment Plans")
77         st.write("Enter a condition and I'll suggest a basic treatment plan.")
78
79         if chatbot is None or not callable(chatbot):
80             st.error("AI model not available or not callable.")
81             st.stop()
82
83         condition = st.text_input("Diagnosed Condition, placeholder='e.g. fever'")
84
85         if st.button("Suggest Treatment Plan"):
86             if not condition.strip():
87                 st.warning("Please enter a condition.")
88                 return
89
90             prompt = f"User: What is the treatment plan for {condition}?"
91             with st.spinner("Generating plan..."):
92                 try:
93                     result = chatbot(prompt, max_new_tokens=128)
94                     response = result[0][generated_text.strip()] if result else ""
95                     except Exception as e:
96                         response = f"Error: {str(e)}"
97
98             st.success(f"Suggested plan: {response}")
99
100     if __name__ == "__main__":
101         main()
102 elif page == "Health Analytics":
103     # health_analytics.py code
104     def main():
105         st.title("Health Analytics + AI Insights")
106         st.write("View health data trends and ask questions to your AI assistant.")
107
108         if chatbot is None or not callable(chatbot):
109             st.error("AI model not available or not callable.")
110             st.stop()
111
112         data = {
113             "User": st.date_input("Select Date Range", start="2023-01-01", end="2023-12-31", period="7D"),
114             "Heart Rate": [70, 80, 90, 85, 75, 72, 88],
115             "Blood Pressure": [120, 125, 130, 128, 135, 140, 138],
116             "Blood Sugar": [90, 95, 100, 105, 110, 115, 112]
117         }
118
119         df = pd.DataFrame(data).set_index("User")
120
121         st.dataframe(df)
122
123         st.subheader("Input Rate")
124         st.line_chart(df["Heart Rate"])
125
126         st.subheader("Blood Pressure")
127         st.line_chart(df["Blood Pressure"])
128
129         st.subheader("Blood Sugar")
130         st.line_chart(df["Blood Sugar"])
131
132         st.markdown("...")
133
134         st.subheader("Ask AI about your health data")
135         query = st.text_input("Ask a question, placeholder='e.g. Is my BP too high?'")
136         if st.button("Ask AI"):
137             # health_analytics.py code
138             def main():
139                 st.title("AI Health Chat")
140                 st.write("Ask any health-related question to your AI assistant!")
141
142                 if chatbot is None or not callable(chatbot):
143                     st.error("AI model not available or not callable.")
144                     st.stop()
145
146                 if "chat_history" not in st.session_state:
147                     st.session_state.chat_history = []
148
149                 user_input = st.text_input("Ask your question, placeholder='e.g. How can I assist you with your health?'")
150
151                 if st.button("Ask"):
152                     if not user_input.strip():
153                         st.warning("Please enter a question.")
154                         return
155
156                     greetings = ["hi", "hello", "hey", "hlo"]
157                     if user_input.lower().strip() in greetings:
158                         response = "Hello! How can I assist you with your health?"
159                     else:
160                         prompt = f"User: {user_input}\nAI: Thinking..."
161                         with st.spinner("Thinking..."):
162                             try:
163                                 result = chatbot(prompt, max_new_tokens=128, return_full_response=True)
164                                 response = result[0][generated_text.strip()] if result else ""
165                             except Exception as e:
166                                 response = f"Error during generation: {str(e)}"
167
168                     st.session_state.chat_history.append((user_input, response))
169                     st.success(response)
170
171             if __name__ == "__main__":
172                 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