

---

☐ **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:**
  - **Upputholla Swathi (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.
  - **Repana Purnachandu , Reddy Bhuvana Lakshmi(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.
  - **Sanka Abhinaya Sri (Documentation, Deployment & API Integration):**  
Contributed significantly to the project's deployment strategy and documentation. Managed environment setup, assisted in IBM Cloud deployment and configuration, and ensured proper functioning of external APIs (e.g., health data visualization, patient record management). Developed supporting documentation for end users and stakeholders, and helped bridge technical efforts with presentation materials to showcase the application effectively.

---

## 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:

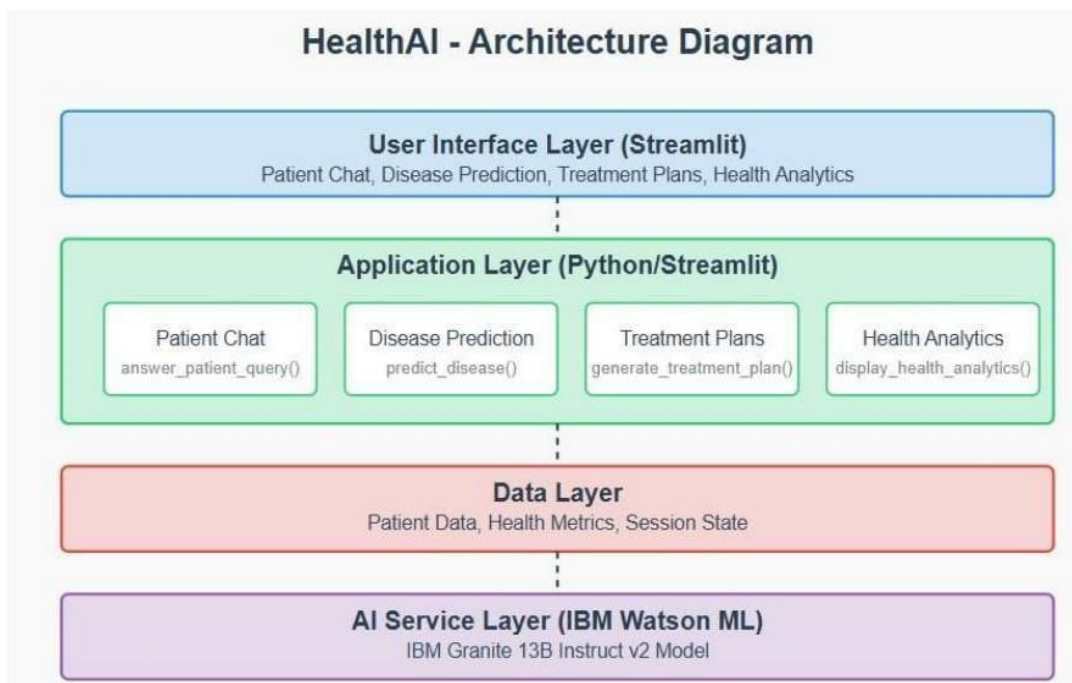
Built using **Streamlit** for a clean and responsive web interface. Each feature is modularized for easy navigation via sidebar.

- **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 : [abhinayasri36/HealthAI-Intelligent-Healthcare-](https://github.com/abhinayasri36/HealthAI-Intelligent-Healthcare-)

[Assistant-Using-IBM-Granite](#)

cd Health-ai

pip install -r requirements.txt

### Environment Variables

Create a .env file in the root folder:

HUGGINGFACEHUB\_API\_TOKEN=hf\_EPKOkQWaTrYYRwbVgrfzpiTWNrSADVjnd

✓ .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

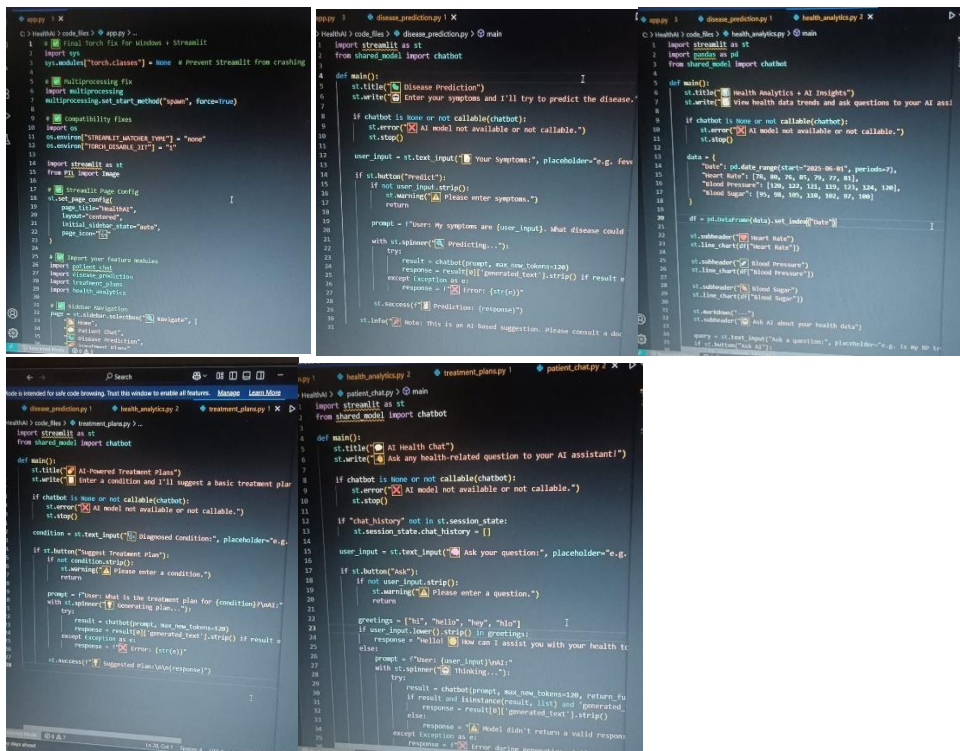
- Centralized theme and branding

## 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

- INPUTS ( CODES ) :



```

# health.py
import sys
sys.modules['torch.classes'] = None # Prevent Streamlit from crashing

# multiprocessing fix
import os
os.environ['STREAMLIT_WATCHER_TYPE'] = 'none'
os.environ['TORCH_DISABLE_TVT'] = '1'

# Import Streamlit as st
from st import Image

# Streamlit Page Config
st.set_page_config(
    page_title="Healthier",
    layout="centered",
)
st.sidebar.state="none",
page_name="1"

# Import your feature modules
import st.session_state
import disease_prediction
import treatment_plan
import health_analytics

# Sidebar Navigation
st.sidebar.subheader("Navigate", [
    "Home",
    "Disease Prediction",
    "Treatment Plan",
    "Health Analytics",
])

# disease_prediction.py
def main():
    st.title("Disease Prediction")
    st.write("Enter your symptoms and I'll try to predict the disease.")
    if chatbot is None or not callable(chatbot):
        st.error("AI model not available or not callable.")
        st.stop()
    user_input = st.text_input("Your symptoms", placeholder="e.g. fever")
    if st.button("Predict"):
        if not user_input.strip():
            st.warning("Please enter symptoms.")
            return
        prompt = f"User: My symptoms are {user_input}. What disease could it be?"
        with st.spinner("Predicting..."):
            try:
                result = chatbot(prompt, max_new_tokens=128)
                response = result[0][generated_text.strip()] if result else ""
            except Exception as e:
                response = f"Error: {str(e)}"
        st.success(f"Prediction: {response}")
    st.info("Note: This is an AI-based suggestion. Please consult a doctor.")

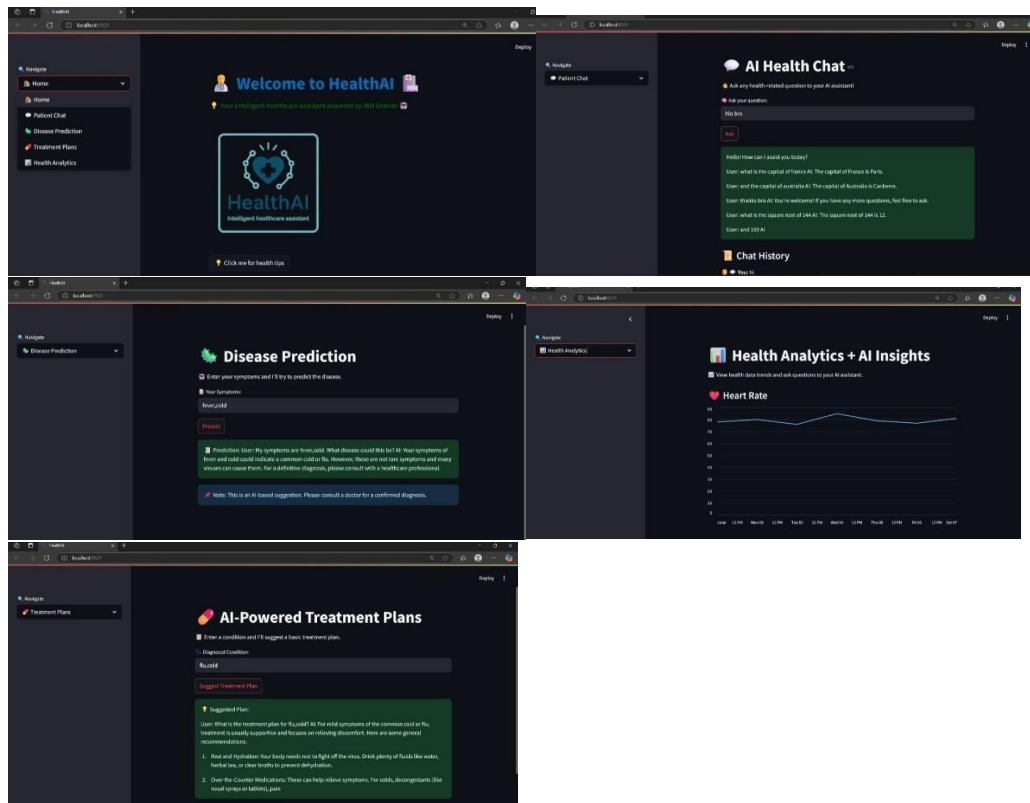
# health_analytics.py
def main():
    st.title("Health Analytics & AI Insights")
    st.write("View health data trends and ask questions to your AI assistant.")
    if chatbot is None or not callable(chatbot):
        st.error("AI model not available or not callable.")
        st.stop()
    data = {
        "Date": st.date_input("Start Date", value=st.date(2023-06-01), period="7d"),
        "Report Date": [70, 80, 90, 85, 75, 77, 81],
        "Blood Pressure": [120, 125, 130, 128, 124, 126],
        "Blood Sugar": [95, 98, 100, 102, 101, 103]
    }
    df = pd.DataFrame(data).set_index("Date")
    st.subheader("Heart Rate")
    st.line_chart(df["Heart Rate"])
    st.subheader("Blood Pressure")
    st.line_chart(df["Blood Pressure"])
    st.subheader("Blood Sugar")
    st.line_chart(df["Blood Sugar"])
    st.subheader("...")
    st.subheader("Ask AI about your health data")
    query = st.text_input("Ask a question", placeholder="e.g. Is my BP in the normal range?")
    if st.button("Ask AI"):
        pass

# treatment_plan.py
def main():
    st.title("AI-Powered Treatment Plans")
    st.write("Enter a condition and I'll suggest a basic treatment plan.")
    if chatbot is None or not callable(chatbot):
        st.error("AI model not available or not callable.")
        st.stop()
    condition = st.text_input("Diagnosed Condition", placeholder="e.g. cold")
    if st.button("Suggest Treatment Plan"):
        if not condition.strip():
            st.warning("Please enter a condition.")
            return
        prompt = f"User: What is the treatment plan for {condition}?"
        with st.spinner("Generating Plan..."):
            try:
                result = chatbot(prompt, max_new_tokens=128)
                response = result[0][generated_text.strip()] if result else ""
            except Exception as e:
                response = f"Error: {str(e)}"
        st.success(f"Suggested plan: {response}")

# patient_chat.py
def main():
    st.title("AI Health Chat")
    st.write("Ask any health-related question to your AI assistant!")
    if chatbot is None or not callable(chatbot):
        st.error("AI model not available or not callable.")
        st.stop()
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = []
    user_input = st.text_input("Ask your question", placeholder="e.g. How can I assist you with your health?")
    if st.button("Ask"):
        if not user_input.strip():
            st.warning("Please enter a question.")
            return
        greetings = ["Hi", "Hello", "Hey", "Hi!"]
        if user_input.lower().strip() in greetings:
            response = "Hello! How can I assist you with your health?"
        else:
            prompt = f"User: {user_input} \n AI: "
            with st.spinner("Thinking..."):
                result = chatbot(prompt, max_new_tokens=128, return_full_response=True)
                response = result[0][generated_text.strip()] if result else ""
            except Exception as e:
                response = f"Error: {str(e)}"
        st.session_state.chat_history.append([user_input, response])
        st.success(response)

```

- 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