HealthAI Documentation

Table of Contents

1. Introduction

Purpose

HealthAI is an intelligent healthcare assistant built to provide preliminary health guidance, predictive insights, and interactive support through AI. It is not a replacement for clinical diagnosis but a tool to assist users, especially in settings with limited access to medical professionals.

Scope

- Disease prediction (from symptoms)
- Treatment plan suggestions
- Conversational chat / Q&A for health queries
- Health analytics (vital signs, trends, etc.)

Audience

- Users seeking health guidance
- Developers maintaining or extending the system
- Researchers verifying validity of outputs

Audience

- Users seeking health guidance
- Developers maintaining or extending the system
- Researchers verifying validity of outputs

---

2. System Overview

High-level Components

- **Frontend / UI**: A web interface (e.g., built with Streamlit) for inputting symptoms, chatting, viewing analytics dashboards.
- **Backend / Model Layer**: Generative AI model (e.g. IBM Granite / Granite-13b-instruct-v2) to perform inference. Logic for disease prediction, treatment recommendation, etc.
- **Data Layer**: Modules for processing user inputs, symptom data, vital signs, etc. Possibly some stored sample datasets for analytics charts.

User Flow

1. User launches app → sees homepage / menu.
2. Selects disease prediction → inputs symptoms →

3. Selects treatment generator → gets suggestions.
4. Uses chat module for specific questions.
5. Enters vital signs or health data into analytics module → sees charts, insights.

---

3. Features / Modules

Module	Input
Disease Prediction	User enters symptoms (fre text or form)
Treatment Generator	Diagnosed (or predicted) condition + possibly user profile
Patient Chat Module	Free text queries (health questions)
Health Analytics Dashboard	Vital signs/history (BP, sug pulse, etc.)

---

4. Architecture & Tech Stack

- **Programming Language**: Python
- **Frontend/UI Framework**: Streamlit for quick web UI

4. Architecture & Tech Stack

- **Programming Language**: Python
- **Frontend/UI Framework**: Streamlit for quick web UI prototyping and dashboards Scribd +1
- **AI / Model**: IBM Granite (Granite-13b-instruct-v2) for generative inference. Scribd +1
- **Data Handling**: Pandas, NumPy for data manipulation; Matplotlib / Seaborn for visualization. Scribd +2
- **Version Control & Hosting**: GitHub; possible use of hosting platform (e.g. cloud services) for deployment. Scribd +1
- **Folder / Code Structure** (based on what's reported):

Copy code

```
HealthAI/
  ├── main.py
  ├── disease_predictor.py
  ├── treatment_generator.py
  ├── chat_interface.py
  ├── analytics_dashboard.py
  ├── requirements.txt
  └── README.md
```

5. API / Model Interaction

- **Model Endpoint**: Either local model loading or via Hugging Face API. E.g., IBM Granite model endpoint. Studocu
- **Authentication**: If using external API (like Hugging Face), use API tokens stored securely (e.g. via `.env`) and not committed in source. Studocu
- **Input Format**:
  - For disease prediction: symptoms in structured or free text form
  - For treatment plan: the condition name + possibly user profile or preferences
  - For chat: natural language questions
  - For analytics: numerical or historical data (vitals, time series)
- **Output Format**: Usually plain text / JSON (if API) + charts / visualization (for analytics)
- **Error Handling**: Deal with invalid inputs (missing fields, malformed symptoms), timeouts in model response, etc.

6. Installation & Deployment

Requirements

- Python 3.x
- Libraries: (as per `requirements.txt`) — streamlit, pandas, numpy, matplotlib, model-inference related libs, etc. Scribd +1
- Possible external dependencies: Hugging Face account / token if using their model; IBM Granite access.

Setup Steps

1. Clone repository from GitHub
2. Create virtual environment, install dependencies:

   Bash Copy code

   ```bash
   pip install -r requirements.txt
   ```

3. Set up API token / model files (`.env`)
4. Run locally:

   Bash Copy code

   ```bash
   streamlit run main.py
   ```

7. Testing & Validation

- **Unit Testing**: For modules (disease prediction logic, treatment generator) to check expected outputs for given inputs.
- **Prompt Testing**: Use varied prompts/questions to ensure model responses are coherent, medically reasonable.
- **Edge Cases**: No symptoms, contradictory inputs, rare disease symptoms — ensure system handles these gracefully.
- **User Acceptance**: (If possible) feedback from medical professionals / domain experts.

---

8. Limitations, Risks & Ethical Considerations

- **Non-clinical tool**: HealthAI is for guidance, not diagnosis. Must include disclaimers.
- **Bias & Data Quality**: Model may reflect biases from training data; predictions may be inaccurate.
- **Privacy & Security**: If taking personal health data, ensure secure transmission, storage, user consent.

9. Future Work & Enhancements

- Real-time health monitoring (e.g. wearable integrations)
- Enhanced chatbot with better context memory + multilingual support
- More modules: image analysis, lab reports, etc.
- Backend improvements: more robust REST/API interface, user authentication, logging / audit trails
- Regulatory compliance, clinical validation studies

---

10. References & Glossary

References

- Original project documentation / academic write-ups (if any) Studocu +1
- IBM Granite model documentation
- Literature on AI in health applications

- Original project documentation / academic write-ups (if any) Studocu +1
- IBM Granite model documentation
- Literature on AI in health applications

Glossary

Term	Meaning
Generative AI	AI models that generate text output given inputs / prompts
Prompt	Input given to a generative model to steer its response
FHIR / ICD	Standard medical coding systems (if used)
Latency / Inference	Time taken by model to respond