Project Documentation Format

1.INTRODUCTION:

* **Project Title**: Health AI-Intelligent Healthcare Assistant Using Granite
* **Team ID:** LTVIP2025TMID33406
* **Team size**:5
* **Team members**:

1. kiran kumar Bhogam (Team Leader)
2. Uppara Anusha
3. T. Deekshitha
4. R. Aswitha
5. H. Anjali

2. PROJECT OVERVIEW:

Purpose

A Conceptual or actual project using artificial intelligence to improve healthcare. If you have a specific company, product, or context in mind (e.g., diagnostics, wearable integration, hospital operations), I can tailor this further.

**Objective:**

To harness the power of artificial intelligence to improve patient outcomes, streamline healthcare operations, and enable personalized, predictive, and preventive healthcare.

**Key Goals:**

* Early Diagnosis & Detection
* Personalized Treatment Plans
* Remote Monitoring & Virtual Care
* Operational Efficiency
* Clinical Decision Support

**Core Technologies Used:**

* Machine Learning (ML) and Deep Learning
* Natural Language Processing (NLP) for analyzing clinical notes
* Computer Vision for interpreting medical imaging
* Wearable and IoT Integration
* Chatbots and Virtual Assistants

**Expected Outcomes:**

* Improved diagnostic accuracy
* Reduced hospital readmissions
* Enhanced patient engagement
* Cost savings through automation and early intervention
* More equitable care through data-driven insights

**Features:**

**Key features**

* Natural Language summarization
* AI Assistant for Q & A
* Sentiment Analysis
* Automated status report
* Task and Issue analysis

**Functionalities**

1.Symptom Identifier:

User enters symptoms → AI suggests possible diseases (non-diagnostic).

2.Home Remedies Generator:

User enters disease → AI gives natural/home-based remedies.

3.Health Chat Assistant:

Users can ask general health questions → AI provides informative, non-diagnostic answers.

4.Disease Prediction:

Based on symptoms input, AI suggests potential conditions (e.g., cold, flu).

5.Personalized Treatment Plan:

Suggests age/gender/history-based tips, non-prescriptive guidance, lifestyle improvements.

**3.ARCHITECTURE:**

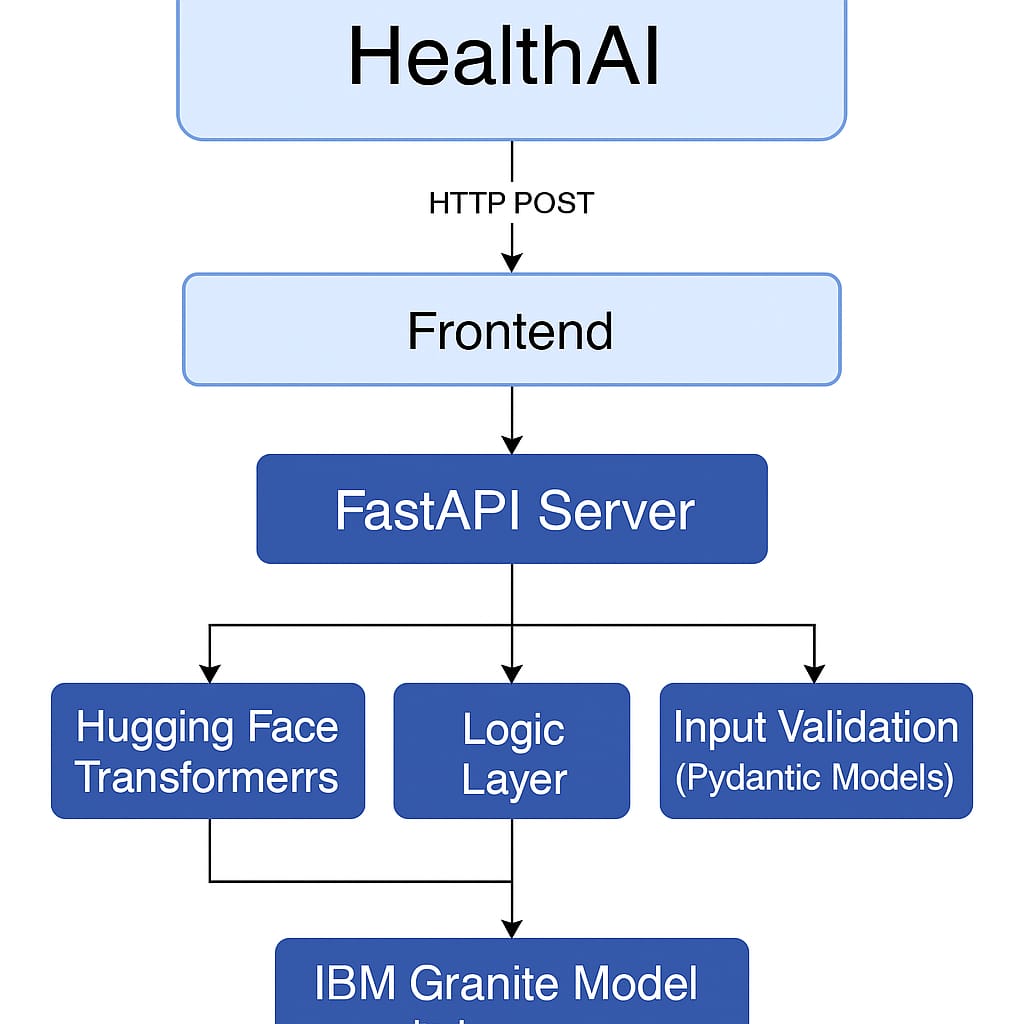
Frontend

**Tech Stack**:

HTML, CSS, JavaScript (or optionally GRADIO in COLAB), running in browser.

**Components:**

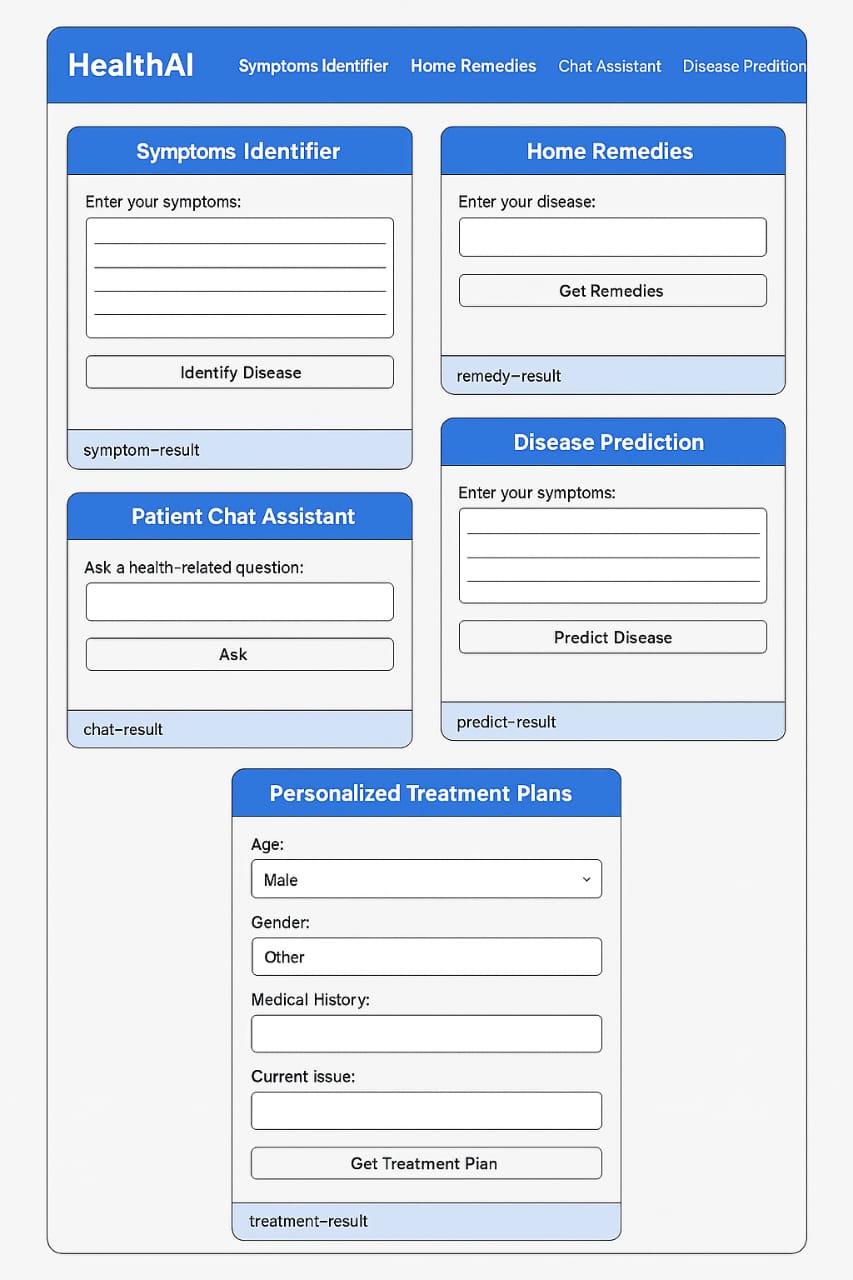
|  |  |
| --- | --- |
| **section** | **Functionality** |
| Symptom Identifier | User enters symptoms → AI suggests possible diseases |
| Home Remedies | User enters disease → AI gives natural /home-based remedies. |
| Patient Chat | Users can ask general health questions → AI provides informative, non-diagnostic answers. |
| Disease Prediction | Based on symptoms input, AI suggests potential conditions (e.g., cold, flu). |
| Treatment Plans | Suggests age/gender/history-based tips, non-prescriptive guidance, lifestyle improvements. |



**Backend**

**Tech Stack:**

* Framework: Fast API
* Model Integration: Hugging Face Transformers (IBM-granite/granite-3b-instruct)
* Google Collab to run the code



**Data base**

**Collections & Schemas**

Interactions with IBM Granite:

All model interactions (symptom ID, chat, remedies, treatment plans) follow this pattern:

Frontend form submission → Fast API

IBM\_granite(prompt) function

Prepares the prompt

Sends to IBM Granite model (via Hugging Face)

Returns inference result

**Advantages of MongoDB**

|  |  |
| --- | --- |
| Feature | MongoDB Benefit |
| User flexibility | Schema-less — add/remove fields as needed |
| AI logs | Store varied input/output pairs easily |
| Medical records | Embed arrays or nested documents |
| Fast API integration | Simple with motor (async MongoDB driver) |

4. SETUP INSTRUCTIONS:

Prerequisites:

**1. Programming Knowledge**

* Python (basic to intermediate)
* HTML & CSS (for frontend)
* Basic JavaScript (if needed)
* Understanding of how AI models work (basic)

**2. Tools & Libraries**

* Fast API – for backend API
* UVI corn – to run the Fast API server
* Transformers (Hugging Face) – to load the IBM Granite AI model
* MongoDB – to store user data, symptoms, and results
* GRADIO (optional) – for simple UI in GOOGLE COLAB

**3. Accounts & Access**

* Hugging Face account (to access IBM Granite model)
* MongoDB Atlas account or local MongoDB installation

**4. Environment Setup**

* Python installed (version 3.9 or above)
* Code editor (e.g., VS Code)
* Internet connection (to use AI models and APIs)

Installation:

Step-by-Step Installation

**Step 1:** Clone the Project Repository

**Step 2:** Set Up a Python Virtual Environment

**Step 3:** Install Required Dependencies

* Fast API
* UVI-corn
* transformers
* torch
* motor

**Step 4:** Set Up Environment Variables

**Step 5**: Update Your Code to Use Environment Variables

**Step 6:** Run the Fast API Server

**Step 7 (Optional):** Test the API

**Step 8:** (Optional for Colab)

You’re Done!

You now have:

Fast API running locally

IBM Granite model loaded from Hugging Face

**5.FOLDER STRUCTURE:**

* Client

|  |  |
| --- | --- |
| **Feature** | **Frontend Component** |
| Symptom Identifier | SymptomForm.jsx + SymptomChecker.jsx |
| Home Remedies | RemedyForm.jsx + Remedies.jsx |
| Patient Chat Assistant | ChatAssistant.jsx + Chat.jsx |
| Disease Prediction | DiseasePredictor.jsx + Prediction.jsx |
| Personalized Treatment Plan | TreatmentPlan.jsx + Treatment.jsx |

* Server

|  |  |
| --- | --- |
| **File** | **Description** |
| symptomController.js | Handles symptom input, predicts disease |
| remedyController.js | Returns natural remedies based on disease |
| chatController.js | Handles chatbot questions and responses |
| predictionController.js | Processes symptoms to predict possible conditions |
| treatmentController.js | Suggests treatment plans based on user profile |

6.RUNNING THE APPLICATION:

**Frontend setup for Health AI**

Objective:

Run a static HTML/CSS frontend using npm start in the client directory.

* Create client/index.html
* Create client/style.css
* Create client/package.json
* Install Dependencies
* Start the Frontend Server

**Backend setup for Health AI**

the backend should be run from the server/ directory — not the client/ directory — because it uses Python (Fast API), not Node.js. However, if you want to use npm start as a command to run the Fast API backend (for consistency), we can set that up.

Backend Setup for Health AI Using IBM Granite + Fast API

1. Install Python Dependencies

Create server/requirements.txt:

* Fast API
* UVI-corn
* transformers
* torch

> Run this in the server/ directory to install

7.API DOCUMENTATION:

HealthAI API Documentation

Overview:

The Health AI backend uses the IBM Granite language model to provide intelligent, non-diagnostic health guidance. The API is built using Fast API and serves as the engine behind the frontend UI.

Base URL (local development):

http://localhost:8000/

All endpoints accept JSON input and return JSON output.

1. POST /symptom\_identifier

Description:

Predict the disease based on symptoms provided by the user.

Example Request:

{

"input": "fever, body ache, sore throat"

}

Example Response:

"response": "Based on your symptoms, you might be experiencing the flu

**8.AUTHENTICATION:**

**Goal:**

To secure access to the Health AI backend endpoints (especially in production), by ensuring that only authorized users or clients can access them.

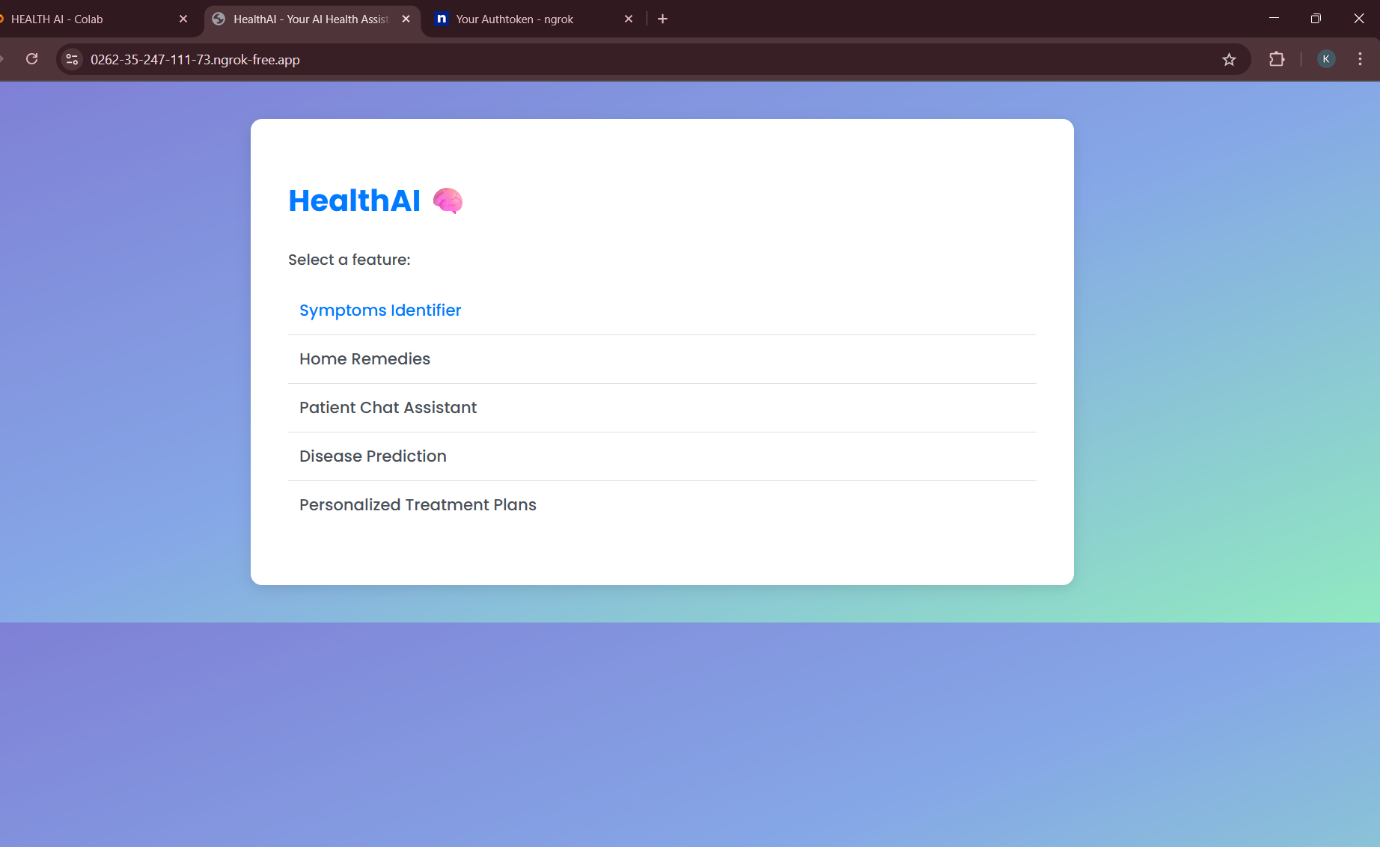
Recommended Authentication Strategy: Token-Based Authentication

Because Health AI is a stateless API (Fast API backend with HTML/JS frontend or possibly a GRADIO app), the best approach is token-based authentication using OAuth2 with JWT (JSON Web Tokens).

**Implementation Outline in FastAPI**

* Install Security Libraries
* User & Token Models
* Auth Logic (Simplified)
* Secure Endpoints Example

**9.USER INTERFACE:**



**Health AI Features to Showcase Visually:**

1. Symptoms Identifier

➤ User inputs symptoms → AI predicts disease.

2. Home Remedies

➤ User enters disease → AI returns natural remedies.

3. Patient Chat Assistant

➤ Chat-like Q&A with AI (non-diagnostic).

4. Disease Prediction

➤ Multi-symptom input → multiple condition suggestions.

5. Personalized Treatment Plans

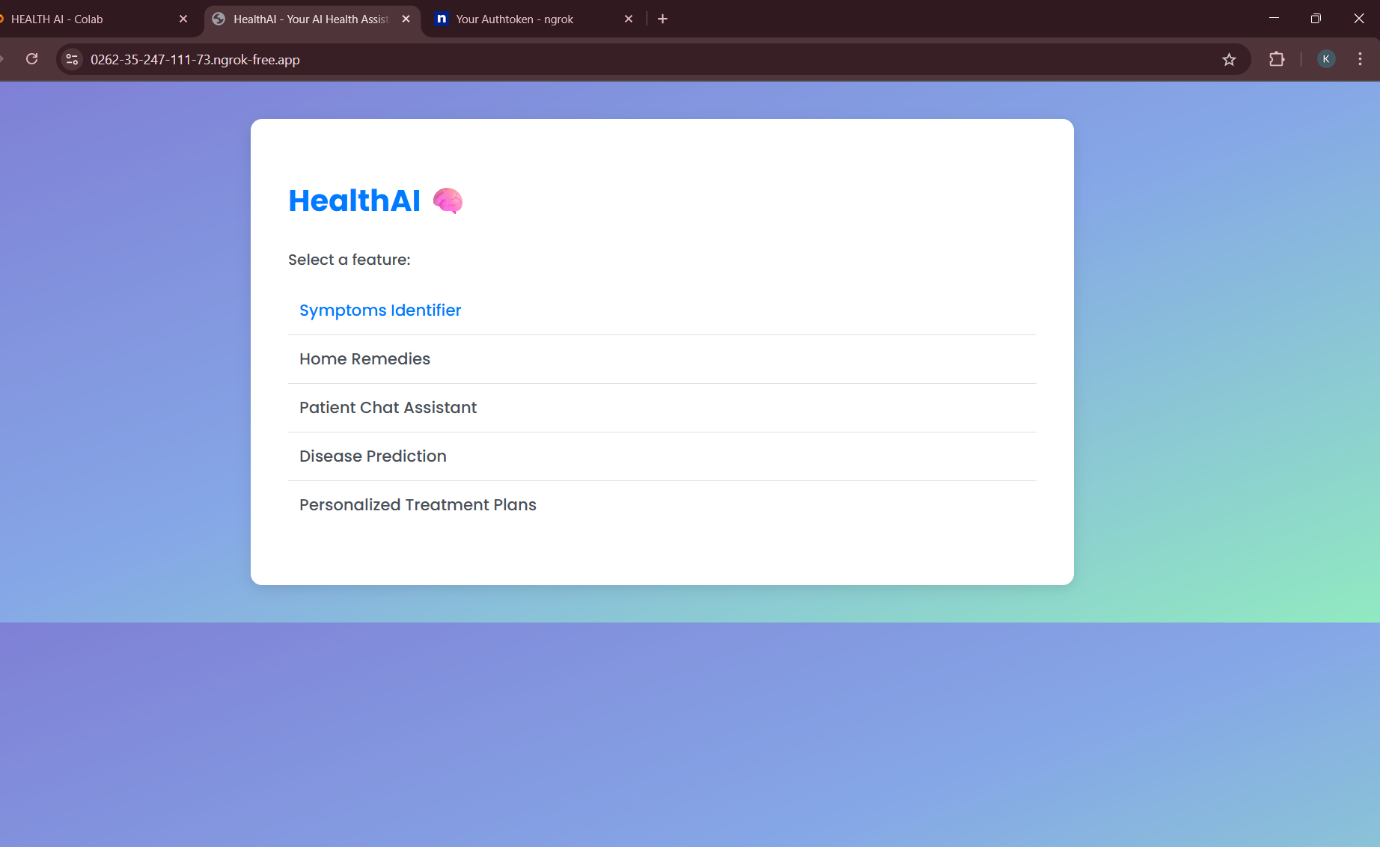
➤ Inputs include age/gender/history → AI responds with tailored plan.

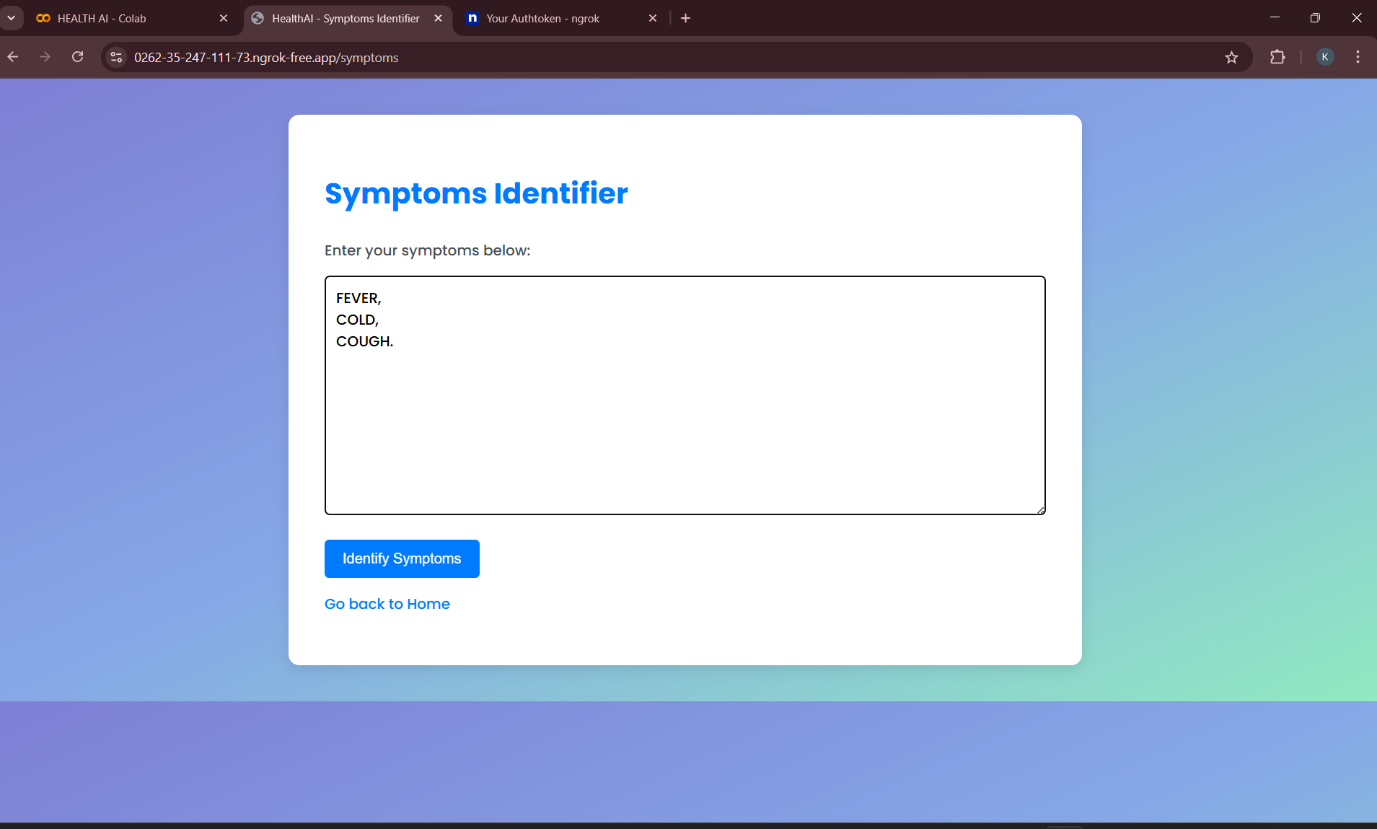
**10.TESTING:**

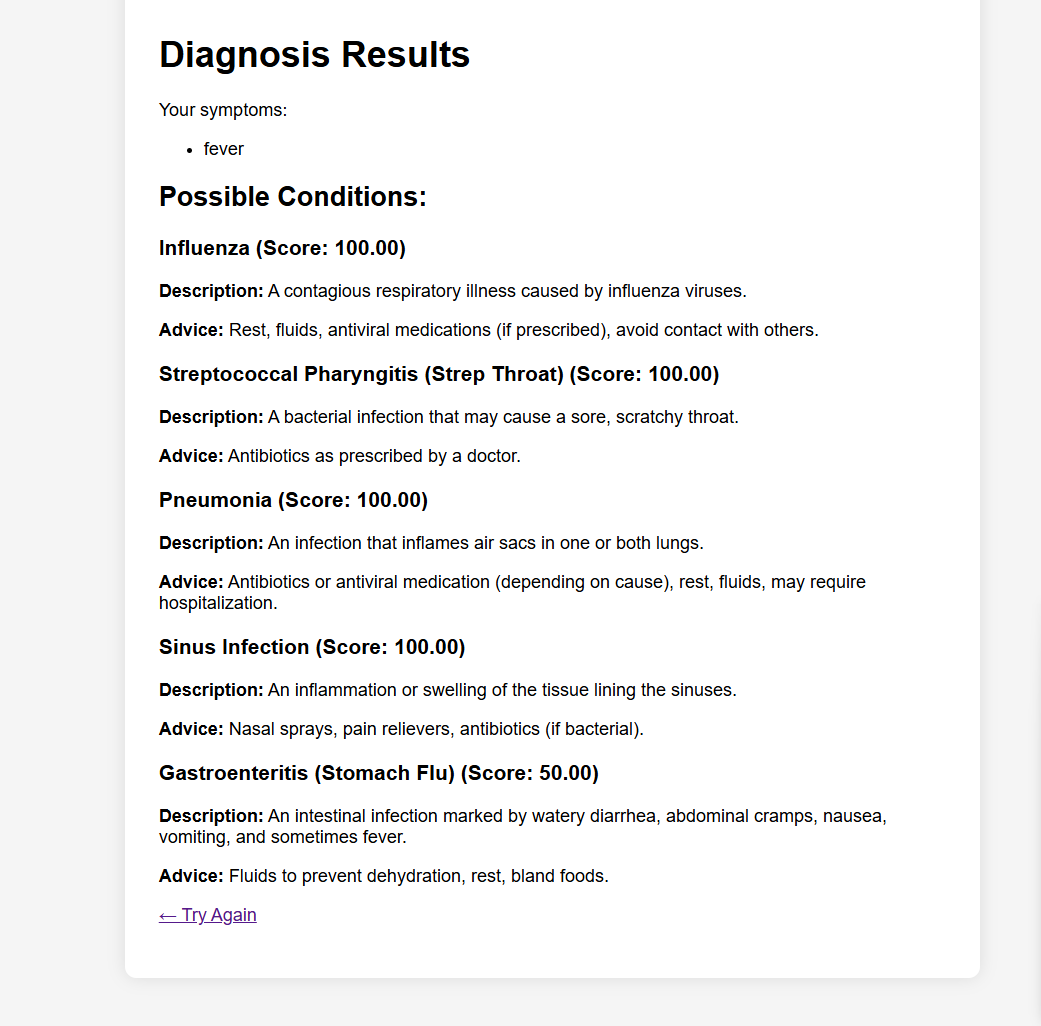
Performance Testing

* Test input validation
* Number input validation
* Content Generation
* API connection check
* Response Time test
* API speed test
* File upload load test

**11.SCREENSHOTS OR DEMO:**







**DEMO LINK:**

**12. KNOWN ISSUES:**

* Model Latency / Slow Response Times
* Non-Deterministic Model Output
* No True Medical Diagnosis Capability
* Granite Model Token Limit
* Input Sanitization Required

**13.FUTURE ENHANCEMENTS:**

* Multimodal Diagnostic Assistance
* Real-time Triage Assistant
* Automated Coding & Billing
* Personalized Healthcare & Treatment Planning
* Clinical Documentation Automation
* Medical Research & Drug Discovery
* Multilingual & Global Health Applications
* Predictive & Preventive Healthcare