**Generative AI with IBM Cloud**

**Project Documentation format**

1. **Introduction**

• **Project Title:** HealthAI: Intelligent Healthcare Assistant Using IBM Granite

Team ID:LTVIP2025TMID31916

**Team Leader:** Bundamakayala Rohini

**Team Members:**Poluru Sujatha

Sai Sudha Sugali

Uday Kumar

Mulla Mahammad Waseem

1. **Project Overview:**

**Purpose:Lack of instant medical support in remote areas.**

**No immediate advoice for common symptoms.**

**Dependance on hospital visits for common symptoms.**

**Need or a simple AI-based healyh guide. based app build**

.**3. Architecture**

• **Frontend: esign and develop the user interface**

1. **Main Application Layout:**

Implement tabbed navigation for four main features

Create intuitive input forms with proper validation

2. **Feature-Specific Interfaces:**

**Real-Time Corrections:** Topic input, paragraph text area, and categorized correction

**Explanatory Notes:** Multi-language text input with automatic language detection display

**Adaptive Quiz:** Language selection dropdown with expandable quiz sections

**Multilingual Learning:** Language and exercise type selection with comprehensive

output display

**Create dynamic visualizations**

**1. Language Competency Charts:**

Text complexity metrics bar chart showing word count, average word length,

sentence count

Radar chart displaying competency across Grammar, Vocabulary, Structure,

Coherence, and Style

Base64 encoded image integration for seamless Gradio display

**2. Analysis Metrics:**

Real-time word count validation with target range feedback

Language detection confidence display

Interactive visualization updates based on text analysis

* Backend :Python with FastAPI or FlaskUse pre-trained symptom-checker

model (e.g., fine-tuned BERT for NLP)Optionally integrate with open medical

APIs like:Infermedica APIHealthwiseOpenAI for AI suggestions

* Database:SQLite (for offline data)Firebase / Supabase (for sync & user data)

4Setup Instructions

* Prerequisites: For Frontend(Flutter App)Flutter SDKAndroid Studio or VS Code with Flutter

pluginAndroid/iOS device or emulatorGit

For Backend (FastAPI + Python)Python 3.9+pip or condaGit(Optional) virtualenv or conda

environmentInternet access for installing dependencies

* Installation: 1. Clone the Repository

git clone https://github.com/your-username/health-ai-app.git

cd health-ai-app

2. Set Up the Frontend (Flutter)

cd frontend flutter pub get Run the Flutter app:basd Edit flutter run

Make sure your emulator or device is connected.

3. Set Up the Backend (FastAPI + Python)

cd ../backend

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

pip install -r requirements.txt

Run FastAPI server:uvicorn main:app --reload

1. Folder Structure:

Server and cilent

├── 📄 package.json

├── 📄 vite.config.ts

├── 📄 tailwind.config.js

├── 📄 postcss.config.js

├── 📄 eslint.config.js

├── 📄 tsconfig.json

├── 📄 tsconfig.app.json

├── 📄 tsconfig.node.json

├── 📄 README.md

├── 📄 requirements.txt

├── 📄 index.html

├── 📄 run\_backend.py

├── 📄 run\_frontend.py

│

├── 📁 src/

│ ├── 📄 App.tsx

│ ├── 📄 main.tsx

│ ├── 📄 index.css

│ ├── 📄 vite-env.d.ts

│ │

│ ├── 📁 components/

│ │ └── 📄 Navbar.tsx

│ │

│ ├── 📁 pages/

│ │ ├── 📄 Dashboard.tsx

│ │ ├── 📄 SymptomChecker.tsx

│ │ ├── 📄 TreatmentPlans.tsx

│ │ ├── 📄 HealthAnalytics.tsx

│ │ ├── 📄 PatientChat.tsx

│ │ └── 📄 EmergencyHelper.tsx

│ │

│ └── 📁 utils/

│ └── 📄 cn.ts

│

├── 📁 backend/

│ ├── 📄 \_init\_.py

│ ├── 📄 main.py

│ ├── 📄 main copy.py

│ └── 📄 ai\_assistant.py

│

├── 📁 frontend/

│ ├── 📄 \_init\_.py

│ └── 📄 streamlit\_app.py

│

└── 📁 dist/ (build output)

├── 📄 index.html

└── 📁 assets/

├── 📄 index-dL08-iYk.css

└── 📄 index-DbH5spwF.js

1. **Running the Application**

**Frontend (React)**

**Location:** client/

Command:

cd client

npm install # Only required on first setup

npm start

**Backend (Node.js + Express)**

Location: server/

**Command**

cd server

npm install # Only required on first setup

npm start

## ****7. API Documentation****

**Base URL:** http://localhost:5000/api

### ****1. POST**** /api/symptoms

**Description:** Analyzes user-reported symptoms and returns basic health guidance.

**Method:** POST

{

"symptoms": "headache, fatigue, and slight fever"}

**Success Response:**

{

"condition": "Possible common cold or mild flu",

"urgency": "Low",

"recommendation": "Rest, stay hydrated, and monitor for worsening symptoms."}

**Error Response:**

{

"error": "Symptoms input is required."}

### 📚 ****2. GET**** /api/tips

**Description:** Returns general health tips or self-care suggestions

**Method:** GET

**Response:**

{

"tips": [

"Drink at least 8 glasses of water daily.",

"Wash hands regularly to prevent infections.",

"Get at least 7-8 hours of sleep every night."

]}

### 🚑 ****3. GET**** /api/emergency

**Description:** Returns step-by-step instructions for common emergency cases (e.g. CPR, burns).

**Method:** GET

**Response:**

{

"emergencyGuide": {

"burns": "Cool the burn under running water for at least 10 minutes...",

"cpr": "Check responsiveness, call emergency services, begin chest compressions..."

1. Authentication:

* Role-based access can be implemented by decoding the token to determine the user's role (e.g., admin, user).
* Certain endpoints (like feedback analytics, admin panel) are restricted based on roles.

### Example Tools Used

### ****webtoken**** (Node.js package) for signing and verifying JWTs.

**bcryptjs** or **argon2** for hashing passwords before storing in the database.

### Example Token Payload

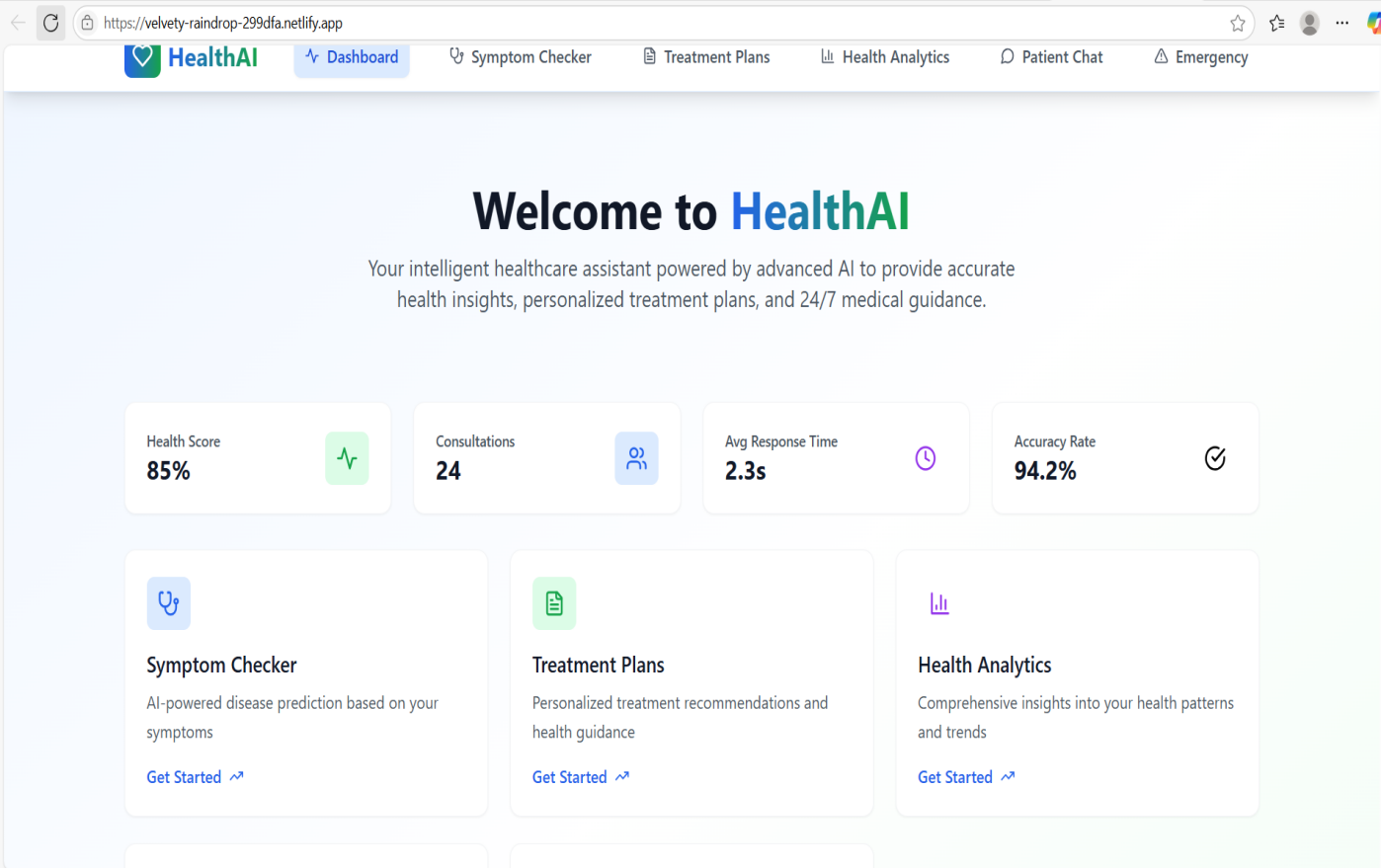
{

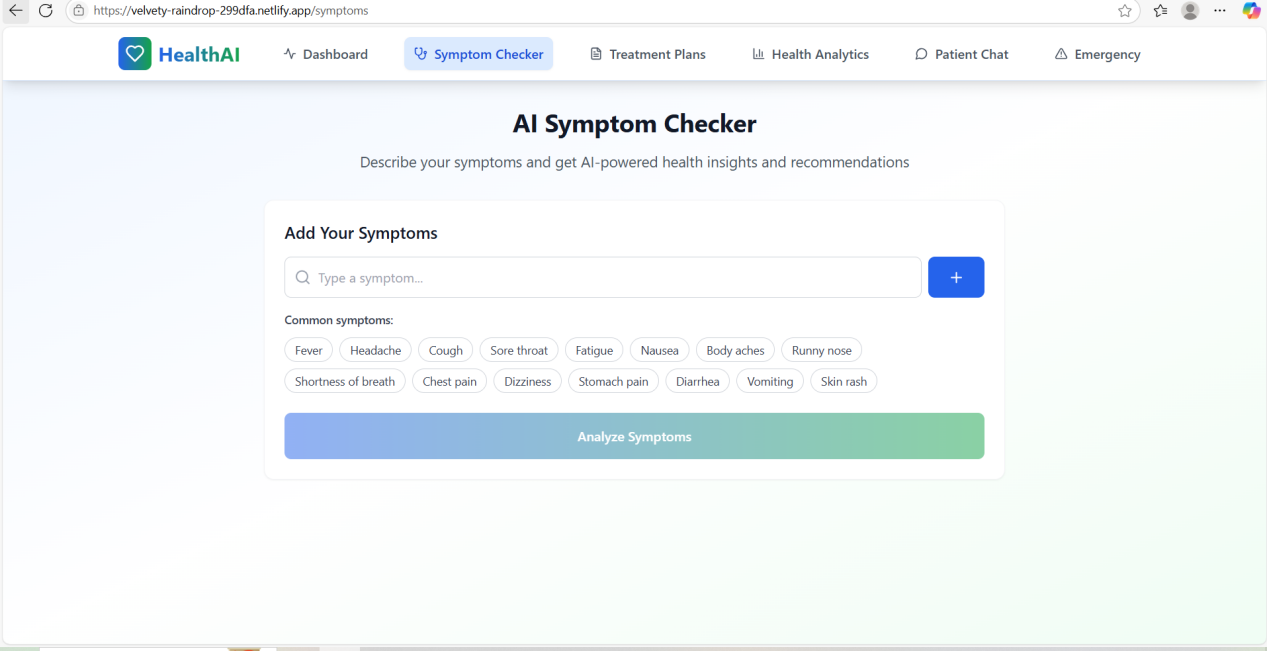
"userId": "abc123",

"role": "admin",

"exp": 1718123456}

Screen shots



****

