

ANNA MA CHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES, TIRUPATI (AUTONOMOUS)

SMAERTBRIDGE __ PROJECT (DATA SCIENCE)

GENERATIVE AI WITH IBM CLOUD

HEALTH AI (INTELLIGENT HEALTHCARE ASSISTANT)

1. Introduction::

→ **PROJECT TITLE:** Health AI (Intelligent Healthcare Assistant)

→ TEAM MEMBERS::

Team leader: Lingala Harshitha(22AK1A3211)

Role:

Milestone 1: Model Selection and Architecture

Milestone 2: Core Functionalities Development

Team member: Garalaan VishnuVardhan(23AK5A3208)

Role:

1. Technical Architecture:

2. Pre-requisites:

Team member :Kuruba Meghana(22AK1A3217)

ROLE:

1. App.py Development

2. Frontend Development

Team member: Majjiga swathi(22AK1A3239)

ROLE:

1. Deployment

2. Project Overview

Purpose:

- The purpose of the *Intelligent Healthcare AI* project is to revolutionize medical services by integrating artificial intelligence into healthcare systems.
- The primary goal is to enhance diagnostic accuracy, improve patient outcomes, and optimize hospital workflows through intelligent data analysis.
- By leveraging AI technologies like machine learning and natural language processing, the system can detect diseases early, suggest personalized treatment plans, and predict patient risks.
- It supports healthcare professionals in making faster, evidence-based decisions while reducing human errors.
- The project aims to improve accessibility to quality healthcare, especially in underserved areas.
- It also facilitates remote patient monitoring and automates routine administrative tasks.
- Ultimately, the goal is to create a smart, data-driven ecosystem that promotes proactive healthcare.
- The system continuously learns from vast medical datasets, adapting to new findings and trends.
- This leads to a more efficient, affordable, and patient-centric healthcare infrastructure.

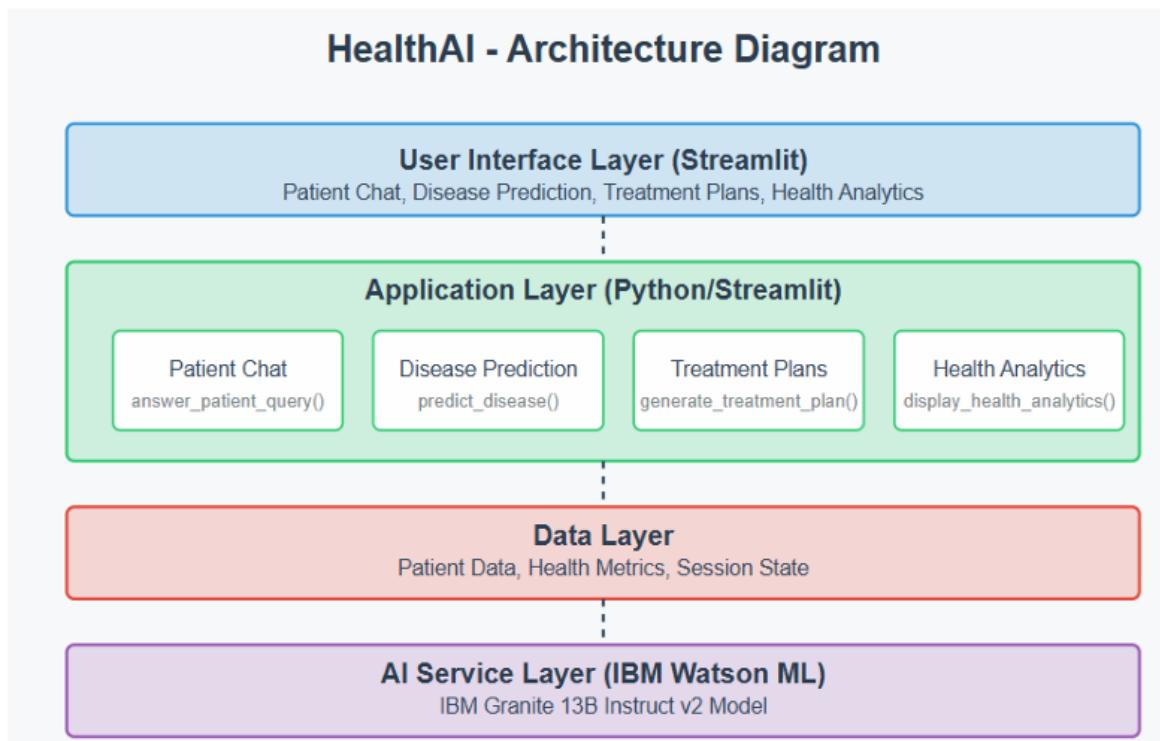
Features:

Here are the key features and functionalities of *Intelligent Healthcare AI*:

1. **AI-Powered Diagnostics:** *Uses machine learning to analyze medical images, lab results, and patient data for accurate and early disease detection.*

2. **Predictive Analytics:** Forecasts health risks and disease progression by analyzing historical and real-time data, enabling preventive care.
3. **Personalized Treatment Plans:** Recommends tailored treatment options based on patient history, genetics, and current health status.
4. **Virtual Health Assistants:** Chatbots and virtual agents assist with appointment scheduling, symptom checking, medication reminders, and patient queries.
5. **Natural Language Processing (NLP):** Extracts useful insights from unstructured medical records, doctors' notes, and research articles.
6. **Remote Patient Monitoring:** Tracks patient vitals and conditions in real-time using IoT and wearable devices, allowing continuous care.
7. **Electronic Health Record (EHR) Integration:** Seamlessly connects with existing EHR systems to provide a unified view of patient history and care.
8. **Automated Administrative Tasks:** Reduces workload by handling billing, documentation, and report generation with AI automation.
9. **Clinical Decision Support:** Assists healthcare professionals in making informed decisions with evidence-based recommendations.
10. **Data Privacy and Security:** Incorporates advanced encryption and compliance protocols (like HIPAA) to ensure patient data confidentiality and safety.

3. Architecture:



1. Frontend (User Interface)

Technologies used: HTML, CSS

- **HTML (HyperText Markup Language)**
 - Structure of the webpage (buttons, forms, headings, paragraphs, tables, etc.)
 - Provides the layout and content elements users interact with.
- **CSS (Cascading Style Sheets)**
 - Handles styling and design (colors, fonts, layouts, responsiveness).
 - Can use frameworks like Bootstrap or Tailwind for faster styling.
- **Optional JavaScript (not mentioned, but usually part of frontend)**
 - Adds interactivity (form validation, dynamic updates, etc.)

2. Backend (Application Logic & Server-side Processing)

Technologies used: Python (with Flask)

- Flask (Python micro-framework)
 - Handles routing (URL endpoints like /login, /register)
 - Processes user inputs from forms
 - Controls logic for interacting with the database
 - Renders templates (HTML pages) and sends them to the browser
 - Manages sessions, authentication, API integrations, etc.

Core Backend Components:

- app.py or main.py: Main entry point of the Flask app
 - Routes (defined using Flask decorators like @app.route)
 - Controllers to handle logic and communication between frontend and backend
 - Templates rendered using render_template() function
-

4. Intelligent Healthcare AI – Setup Instructions

Prerequisites

Before setting up the project, ensure the following tools and software are installed:

1. Python 3.8+
2. pip (Python package manager)
3. Text editor or IDE (e.g., VS Code)
4. Web browser (e.g., Chrome)
5. Optional: Virtual environment (recommended)

Installation Steps

1. Clone or Download the Project
2. Create and Activate a Virtual Environment (optional but recommended)
3. Install Required Python Packages:
`pip install flask`
4. Project Structure

5. intelligent-healthcare-ai/

```
|  
|   ├── app.py  
|   ├── ht.py  
|   ├── utils.py  
|   ├── requirements.txt  
|   ├── venv/  
|   |   └── __pycache__/  
|   ├── model/  
|   ├── granite-3.3-2b-instruct/  
|   ├── txt/  
|   |  
|   |   └── static/  
|   |       └── styles.css  
|   |  
|   |   └── templates/  
|   |       ├── index.html  
|   |       ├── disease.html  
|   |       ├── treatment.html  
|   |       ├── result.html  
|   |       ├── report_result.html  
|   |       ├── analytics.html  
|   |       └── upload.html
```

6 Code Implementation

7.Run the Application

5. Project Folder Structure

```
>pycache  
> granite-3.3-2b-instruct  
model  
>requirements.txt  
>txt  
>>static  
# styles.css  
>>>templates  
<> analytics.html  
<> disease.html  
<> index.html  
<> report_result.html  
<> result.html  
<> treatment.html  
<> upload.html  
>> venv  
app.py  
>ht.py  
>> utils.py
```

CLIENT (Frontend)

Purpose: Handles the user interface displayed in the browser.

Contents:

- HTML (templates/): Template files rendered by Flask (e.g., index.html)
- CSS (static/css/): Styling for layout, typography, buttons, etc.
- Images (static/images/): Icons, illustrations, logos, etc.

SERVER (Backend)

Purpose: Handles business logic, routing, and communication between the UI and future AI/database components.

Contents:

- app.py: Flask app where:
 - Routes like / are defined
 - Form data is processed
 - AI logic (currently placeholder) is added

6. Running the Application

1. Activate Your Virtual Environment

- **On Windows:**
 - bash
 - CopyEdit
 - venv\Scripts\activate
- **On macOS/Linux:**
 - bash
 - CopyEdit
 - source venv/bin/activate

2. Install Dependencies (if not already installed)

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

3. Start the Flask Server

- **If your main Flask app is in `app.py`, run:**
 - bash
 - CopyEdit
 - python app.py

4. Access the Application

- After running the above command, Flask will start the development server on:
 - cpp
 - CopyEdit
 - `http://127.0.0.1:5000/`
 - Open this URL in your browser to see the app.

Optional: Set Flask Environment Variables (for development)

- **On Windows (CMD):**

- cmd
- CopyEdit
- set FLASK_APP=app.py
- set FLASK_ENV=development
- flask run

On macOS/Linux (bash/zsh) :

```
bash  
CopyEdit  
export FLASK_APP=app.py  
export FLASK_ENV=development  
flask run
```

This enables **auto-reload** when you make code changes.

Let me know if you want:

- a startup script,
- debug mode settings,
- or to integrate AI models from the model/ or granite-3.3-2b-instruct/ folders.

7. API Documentation – Intelligent Healthcare AI

This documentation describes all endpoints exposed by the backend. It includes request methods, required parameters, and example responses.

Base URL (localhost)

Cpp::

http://127.0.0.1:5000/

1. GET /

Description:

Returns the homepage with a form to enter symptoms for diagnosis.

- Method: GET
- Parameters: None

Response:

html

CopyEdit

<!DOCTYPE html>

<html>

...

</html>

2. POST /diagnose

Description:

Accepts user-submitted symptoms and returns a suggested diagnosis.

- **Method:** POST
- **Content-Type:** application/x-www-form-urlencoded

◊ Parameters:

Name	Type	Required	Description
------	------	----------	-------------

symptoms	string	<input checked="" type="checkbox"/>	List of user symptoms
----------	--------	-------------------------------------	-----------------------

Example Request (Form):

bash

```
POST /diagnose
```

```
symptoms = fever, cough, fatigue
```

Example Response (HTML Render):

html

```
<p>AI Diagnosis Suggestion for symptoms: fever, cough, fatigue</p>
```

3. GET /disease/<disease_name>

Description:

Fetches information about a specific disease.

- **Method:** GET
- **Path Parameter:** disease_name (string)



Example Request:

bash

CopyEdit

```
GET /disease/diabetes
```



Example Response (JSON):

json

CopyEdit

```
{  
  "disease": "Diabetes",  
  "description": "A chronic condition that affects glucose metabolism.",  
  "symptoms": ["Fatigue", "Weight loss", "Blurred vision"],  
  "severity": "Moderate"  
}
```

4. GET /treatment/<disease_name>

Description:

Returns treatment options for a given disease.

- **Method:** GET
- **Path Parameter:** disease_name (string)

Example Request:

bash

CopyEdit

GET /treatment/hypertension

Example Response (JSON):

json

CopyEdit

```
{  
  "disease": "Hypertension",  
  "treatments": [  
    "Lifestyle changes",  
    "ACE inhibitors",  
    "Calcium channel blockers"  
  ]  
}
```

5. POST /upload

Description:

Uploads a report file for processing (PDFs, images, etc.).

- **Method:** POST
- **Content-Type:** multipart/form-data

Parameters:

Name	Type	Required	Description
------	------	----------	-------------

report file	<input checked="" type="checkbox"/>		File to be uploaded
-------------	-------------------------------------	--	---------------------

Example Request (curl):

bash

CopyEdit

```
curl -X POST http://127.0.0.1:5000/upload \
```

```
-F "report=@report.pdf"
```

Example Response (HTML Render):

html

CopyEdit

```
<p>Report uploaded and processed successfully.</p>
```

6. GET /analytics

Description:

Displays analytics visualizations for disease reports or diagnosis data.

- **Method:** GET
- **Parameters:** None

Example Response:

Rendered HTML page showing graphs/statistics from system data.

7. GET /report/result

Description:

Returns the result page showing detailed AI analysis after report upload or diagnosis.

- **Method:** GET
- **Parameters:** None

Example Response:

HTML-rendered report page with structured medical insights.

8. GET /result

Description:

Displays the diagnosis/treatment result after form submission.

- **Method:** GET
- **Parameters:** None

Example Response:

Page with diagnosis summary, disease prediction, or suggested treatments.

9. GET /upload

Description:

Returns the file upload form page.

- **Method:** GET
- **Parameters:** None

Response:

html

CopyEdit

```
<form action="/upload" method="POST" enctype="multipart/form-data">  
...  
</form>
```

Error Responses (Generic)

Code	Meaning	Description
400	Bad Request	Missing or invalid parameters
404	Not Found	Endpoint or resource not available
500	Internal Server Error	Logic or backend error

8.Authentication & Authorization – Intelligent Healthcare AI

How Authentication & Authorization Are Handled

In a basic Flask setup, this can be implemented using:

- ◆ Sessions (for Login State Management)

Flask's built-in session object stores user data across requests after login.

- ◆ Password Hashing

User passwords should be securely stored using hashing (e.g., with Werkzeug or bcrypt).

- ◆ Login-Required Decorators

Protected routes can be wrapped using custom or library-based @login_required decorators.

Implementation Details

A. Session-Based Authentication (Most Common in Flask)

Login Flow:

1. User submits login credentials via form.
2. Server checks username and hashed password in the database.
3. If valid, the server sets:

python

CopyEdit

```
session['user_id'] = user.id  
session['username'] = user.username
```

B. Token-Based Authentication (For APIs / Mobile / SPA)

If you're exposing a REST API, you may use JWT (JSON Web Tokens):

- Tokens are issued after login.
- Sent in the Authorization header in future requests.
- Verified on each request without server-side sessions.

Example Token Response:

```
json
{
  "token": "eyJhbGciOiJIUzI1NilsInR..."
}
```

Flask Extensions:

- flask-jwt-extended for full token auth
- itsdangerous for generating signed tokens



Sessions vs Tokens: Comparison

Feature	Sessions	Tokens (JWT)
Stateful	Yes (server stores session)	No (stateless)
Suitable for Web apps		APIs, mobile apps
Storage	Cookie-based	Header-based (Authorization)
Revocable	Yes	Requires token blacklist

Authorization Logic (Role-Based Access)

If the system has roles like Doctor, Admin, Patient:

- Store the role in session or token claims
- Example:

```
python
```

```
if session.get('role') != 'Doctor':  
    return "Access Denied", 403
```

5 Logout

Clearing session on logout:

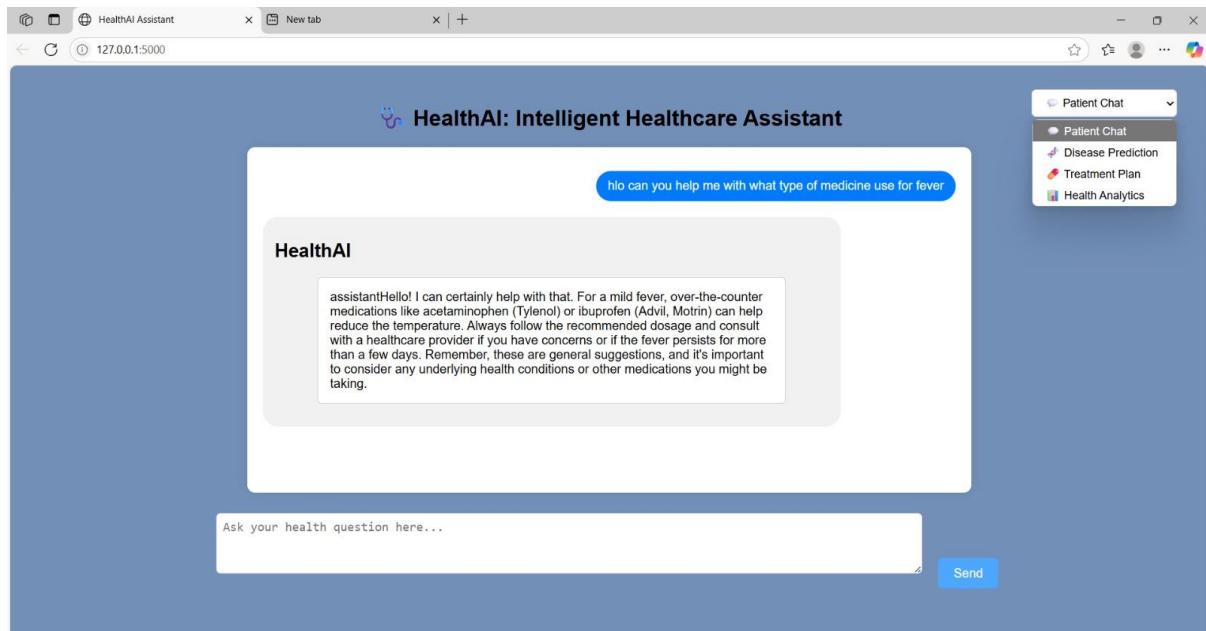
```
python
```

```
@app.route('/logout')  
  
def logout():  
    session.clear()  
  
    return redirect('/login')
```

6 Security Tips

- Always hash passwords
(werkzeug.security.generate_password_hash)
- Use @login_required on protected routes
- Use https in production
- Use secure cookies (session.permanent = True,
app.config['SESSION_COOKIE_SECURE'] = True)
- Implement CSRF protection if using forms

9.USER INTERFACE



10.Testing:

Testing Strategy

A. Unit Testing

Tests individual components (functions, routes, etc.).

- **What to test:**

- Flask routes (/diagnose, /upload)
- Utility functions (e.g., symptom parsing, diagnosis logic)
- Data formatting functions (e.g., for charts)

- **Tool:**

unittest (Python standard library)

- Example:

```
python
import unittest
from app import app
class FlaskTestCase(unittest.TestCase):
    def test_home_page_loads(self):
        tester = app.test_client(self)
        response = tester.get('/')
        self.assertEqual(response.status_code, 200)
```



B. Integration Testing

Ensures multiple components work together correctly.

- **What to test:**

- End-to-end flow: submit symptoms → get diagnosis
- File upload process and output
- Rendering of result and analytics pages

- **Tool:**

pytest + Flask test client + sample data



C. UI Testing

Tests how the user interface behaves in the browser.

- **What to test:**

- Form submissions
- Button clicks
- Input validations

- **Tools:**

- Selenium (for browser automation)
 - Flask's debug mode (for manual testing)
-



D. Functional Testing

Verifies that features work as intended.

- **Examples:**

- Submitting symptoms shows correct diagnosis message
 - Uploading a report gives expected result
 - Navigation bar links route to correct pages
-



E. Security Testing

Optional but recommended for healthcare-related apps.

- **Tests:**

- Form input sanitization (avoid injection)
- File upload type restrictions (e.g., only .pdf)
- Session hijacking prevention

- **Tools:**

- Manual input testing
 - bandit (Python security linter)
-



F. Performance/Load Testing

If the app will handle many users or reports.

- **Tools:**

- locust.io or Apache JMeter (optional)
 - Test endpoint response times
-



Tools Used Summary

Tool	Purpose
unittest	Unit testing in Python
pytest	Advanced test framework
Flask test_client	Simulate requests to Flask routes
Selenium	Browser-based UI testing
bandit	Security linting for Python code
Postman	API testing (for endpoints like /diagnose)
locust	Load testing (optional)



📁 Suggested Test Folder Structure

bash

```
/tests/
|
├── test_routes.py      # Route tests
├── test_utils.py       # Utility function tests
├── test_upload.py      # File handling tests
└── test_api.py         # API JSON responses (optional)
```

11.Screen Shots or Demo::

The image displays two identical web browser windows side-by-side. Both windows have a title bar 'Health Analytics' and a URL '127.0.0.1:5000/analytic'. The main content area is a form titled 'Enter Patient Vitals for Health Analytics'. It contains four input fields with the following values:

- Field 1: 120
- Field 2: 87
- Field 3: 60
- Field 4: 75

Below the input fields is a blue 'Analyze' button.

The second window shows the same form with different input values:

- Field 1: 80
- Field 2: 110
- Field 3: 75
- Field 4: 75

Again, there is a blue 'Analyze' button at the bottom.

The image shows a single web browser window with a title bar 'Disease Prediction' and a URL '127.0.0.1:5000/disease'. The main content is a form titled 'Disease Prediction' with the following fields:

- Text input: cough and cold
- Text input: 17
- Text input: Male (with a dropdown arrow)
- Text input: no
- Text input: 120
- Text input: BP Systolic
- Text input: BP Diastolic
- Text input: Glucose
- Text input: sneezing

At the bottom of the form is a blue 'Predict' button.

Treatment Plan

127.0.0.1:5000/treatment

Treatment Plan

hairfall
20
Female
thyroid

Generate Plan

HealthAI Assistant

127.0.0.1:5000

HealthAI: Intelligent Healthcare Assistant

hi

assistantHello! I'm here to help with your health concerns. As an AI, I can provide general information, but I can't diagnose. Please share your symptoms or questions, and I'll do my best to offer reliable, easy-to-understand facts and advice.

what is the use of dolo tablet

Ask your health question here...

Send

Upload Report

HealthAI Assistant

127.0.0.1:5000

HealthAI: Intelligent Healthcare Assistant

assistantHello! I'm here to help with your health concerns. As an AI, I can provide general information, but I can't diagnose. Please share your symptoms or questions, and I'll do my best to offer reliable, easy-to-understand facts and advice.

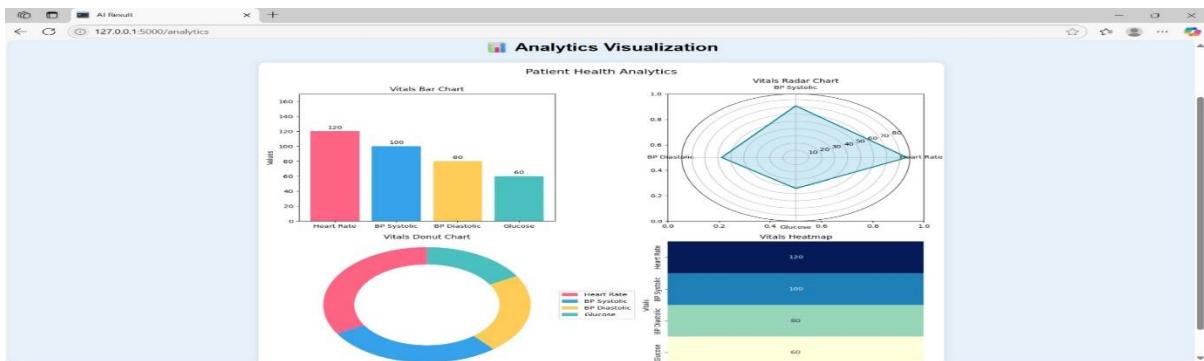
what is the use of dolo tablet

assistantDolo tablets are commonly used for pain relief, primarily due to their active ingredient, diclofenac. They work by reducing inflammation and suppressing the production of certain chemicals in the body that cause pain and swelling. Always follow your healthcare provider's instructions for dosage and duration, as they can vary based on individual needs and medical conditions.

Ask your health question here...

Send

Upload Report



HealthAI

assistantAs a medical AI, based on the provided information, the top 3 likely conditions for a 17-year-old male with a cough, cold, sneezing, and a heart rate of 120 bpm are: 1. Common Cold: This is the most probable condition, given the symptoms of cough and sneezing. The likelihood is high, as these are typical signs of a viral upper respiratory tract infection. Next steps would involve recommending rest, hydration, and over-the-counter remedies for symptom relief. 2. Acute Bronchitis: This condition could also be possible, especially if the cough persists for more than a week. It's a bacterial infection of the bronchial tubes, often following a cold. The likelihood is moderate without a recent bacterial culture, so a doctor's evaluation might be needed for confirmation and potential antibiotics. 3. Allergic Rhinitis: Given the presence of sneezing, this condition, often caused by allergies, could be a consideration. The likelihood is lower without a known allergen history, but further investigation, such as allergy testing, might be recommended for confirmation.

HealthAI

assistantFor a 20-year-old female with hairfall, especially considering a thyroid condition, a comprehensive treatment plan would include: 1. **Medications**: A thyroid hormone replacement therapy if necessary to stabilize thyroid function, which can significantly impact hair health. 2. **Lifestyle**: Managing stress through relaxation techniques like yoga or meditation, as stress can exacerbate hair loss. 3. **Diet**: Incorporating a balanced diet rich in vitamins A, C, and E, iron, and zinc, which are essential for hair growth. Foods like leafy greens, nuts, seeds, and fatty fish can be beneficial. 4. **Physical Activity**: Regular, moderate exercise to improve overall health and circulation, which supports hair follicles. 5. **Follow-up Care**: Regular check-ups with an endocrinologist to monitor thyroid levels and adjust medications if needed. 6. **Mental Health Tips**: Practicing mindfulness or seeking professional help if stress or anxiety about hair loss becomes overwhelming, as these can contribute to hair fall.

12.Known Issues – Intelligent Healthcare AI

This section documents any current limitations, bugs, or areas requiring caution for users or developers working with the project.

⚠ 1. Limited AI Logic Integration

- **Issue:** *Diagnosis logic is currently placeholder-based, not powered by a real ML model.*
 - **Impact:** *May return generic or inaccurate health suggestions.*
 - **Recommendation:** *Integrate a trained model using scikit-learn, TensorFlow, or HuggingFace for better accuracy.*
-

⚠ 2. No Database Integration

- **Issue:** *User data (e.g., login info, uploaded report history) is not saved.*
 - **Impact:** *App is currently stateless, and users lose session/history after refresh.*
 - **Recommendation:** *Add SQLite or PostgreSQL integration for persistent storage.*
-

⚠️ 3. No Authentication in Production Mode

- **Issue:** Authentication uses basic session logic and is not production-secure.
 - **Impact:** Sensitive features (like analytics or upload) can be accessed by anyone.
 - **Recommendation:** Add encrypted passwords, login throttling, and user roles with Flask-Login or JWT.
-

⚠️ 4. File Upload Vulnerability (Validation Missing)

- **Issue:** Uploaded files are not strictly checked for type/content.
 - **Impact:** Potential risk if non-medical files (e.g., scripts or executables) are uploaded.
 - **Recommendation:** Restrict file types using secure MIME-type checking and a whitelist (e.g., .pdf, .jpg only).
-

⚠️ 5. No Mobile Responsiveness

- **Issue:** The UI is optimized for desktop but not fully responsive on small screens.
 - **Impact:** Users on tablets or phones may have layout issues.
 - **Recommendation:** Apply responsive design using Bootstrap or Tailwind CSS.
-

6. No Input Sanitization

- **Issue:** Form inputs (e.g., symptoms) are directly used in logic/output.
 - **Impact:** May result in formatting bugs or XSS vulnerabilities.
 - **Recommendation:** Sanitize inputs on the backend using Python string filtering or Flask-WTF forms.
-

7. Static Analytics Data

- **Issue:** The analytics charts display hardcoded demo values.
 - **Impact:** Real-time user health data is not yet reflected in graphs.
 - **Recommendation:** Link chart data to backend calculations or database queries.
-

8. Error Handling Is Minimal

- **Issue:** Some routes lack proper try-except or 404 error pages.
 - **Impact:** Unhandled errors may crash the app or expose stack traces.
 - **Recommendation:** Implement error handlers using @app.errorhandler() decorators.
-

13. Future Enhancements – Intelligent Healthcare AI

1. AI-Powered Diagnosis Engine

- Integrate a trained machine learning or deep learning model to:
 - Predict possible diseases based on symptoms
 - Provide confidence scores or risk levels
 - Use models like logistic regression, random forest, or transformers (e.g., BioBERT).
-

2. Database Integration

- Add a relational database (e.g., SQLite, PostgreSQL) to:
 - Store patient records and reports
 - Maintain user history, uploaded files, and diagnosis logs
 - Enable user profile management
-

3. Secure User Authentication & Role Management

- Implement secure login/registration using Flask-Login or JWT
 - Add user roles (e.g., Patient, Doctor, Admin) with access control
 - Enable password hashing, session timeouts, and two-factor authentication (2FA)
-

4. Live & Dynamic Analytics Dashboard

- Connect visualizations to real-time patient or system data
 - Add charts for:
 - Disease distribution
 - Diagnosis accuracy over time
 - User demographics
-

5. Responsive Mobile UI

- Improve UI to support all devices (mobile/tablet/desktop)
 - Use CSS frameworks like Bootstrap or Tailwind CSS for better responsiveness
-

6. Advanced Report Upload & OCR

- Allow PDF/image uploads for scanned reports
 - Use OCR (Optical Character Recognition) to extract and analyze text (via Tesseract or Google Vision API)
-

7. Printable Medical Report Generation

- Automatically generate downloadable or printable diagnosis reports in PDF format using libraries like reportlab or xhtml2pdf
-

8. Doctor Consultation Portal

- Enable doctor–patient interaction via:
 - Chat system
 - Appointment booking
 - Video/audio consultation API integration
-

9. Multilingual Support

- Add translations for UI and medical terms
 - Support for local languages (e.g., Hindi, Telugu, Tamil)
-

10. API-First Architecture

- Refactor backend into a RESTful API
 - Expose endpoints for:
 - Symptoms submission
 - Diagnosis result retrieval
 - Report upload/download
 - Allow integration with mobile apps or third-party health platforms
-

11. Security & Compliance

- Implement HIPAA/GDPR-compliant data handling
 - Enable encrypted storage and secure file uploads
 - Add audit logs for diagnosis and report access
- =====*****=====