**HealthAl: Intelligent Healthcare Assistant Using IBM Granite**

**1.Introduction::**

Project title:Health AI(Intelligent Healthcare Assistant Using IBM Granite)

Team members:

**Team leader:** Vankireddy Sruthi

**Role:**

**Milestone 1: Model Selection and Architecture**

**Technical Architecture**

**Team member:** Palreddygari Snehalatha

**Role:**

**Milestone 2: Core Functionalities Development**

**App.py Development**

**Team member :**Koppala Gayathri

**Role:**

**Design and develop the user interface**

**Deployment**

**2.Project Overview**

**Purpose:**

* **Enhance patient care** by delivering fast, accurate, and context-aware responses using IBM Granite’s advanced AI language models.
* **Assist healthcare professionals** with decision support, data summarization, and patient interaction management.
* **Automate administrative tasks** like appointment scheduling, report generation, and medical transcription to save time and reduce human error.
* **Provide 24/7 intelligent assistance** to patients, improving accessibility to healthcare information and support.
* **Leverage IBM Granite's capabilities** to ensure high accuracy and compliance with healthcare data standards and regulations.
* **Enable personalized responses** by learning patient needs and adapting accordingly through secure and ethical data handling.

**FEATURES:**

Some important features are

* 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.

* 1. **Virtual Health Assistants:** Chatbots and virtual agents assist with appointment scheduling, symptom checking, medication reminders, and patient queries.

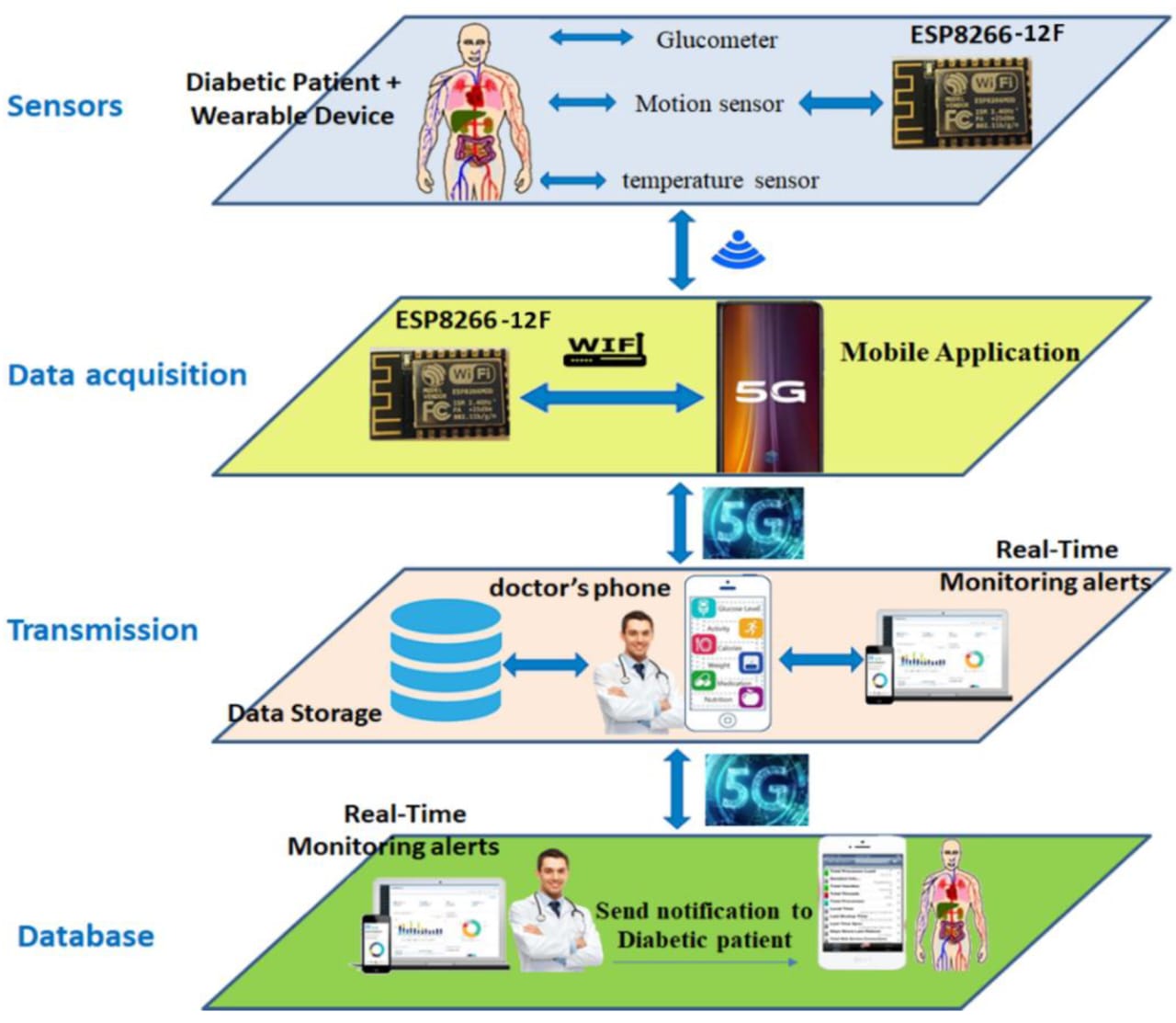
* 1. **Natural Language Processing (NLP):** Extracts useful insights from unstructured medical records, doctors’ notes, and research articles.

* 1. **Remote Patient Monitoring:** Tracks patient vitals and conditions in real-time using IoT and wearable devices, allowing continuous care

7. Clinical Decision Support: Assists healthcare professionals in making informed decisions with evidence-based recommendations.

8. 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)** o Structure of the webpage (buttons, forms, headings, paragraphs, tables, etc.)
  + Provides the layout and content elements users interact with.
* **CSS (Cascading Style Sheets)** o 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)** o Adds interactivity (form validation, dynamic updates, etc.)

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

**Technologies used:** Python (with Flask)

* Flask (Python micro-framework) o Handles routing (URL endpoints like /login, /register) o Processes user inputs from forms o Controls logic for interacting with the database

o Renders templates (HTML pages) and sends them to the browser o 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.– Setup Instructions**

# Prerequisites

* ✅ IBM Cloud account with access to **IBM Granite foundation models**
* ✅ Development environment (Node.js or Python)
* ✅ Database setup (e.g. MongoDB, PostgreSQL)
* ✅ Frontend stack (React, Angular, or similar)
* ✅ Docker & Kubernetes (for scalable deployment)

**⚙️ Initial Project Setup**

**📁 Project Structure**

/healthai-app

│

├── backend/ → APIs, AI connector, authentication

├── frontend/ → UI components (chat, records, dashboard)

├── models/ → AI prompt templates, wrappers

├── configs/ → Environment variables, keys, access scopes

└── docs/ → API documentation, setup guides

**📦 Install Dependencies**

bash

# Backend (Node.js or Python)

npm install

# or

pip install -r requirements.txt

# Frontend

cd frontend && npm install

**🔑 Configure IBM Cloud Services**

**🔹 IBM Granite**

* Deploy Granite model via **IBM watsonx.ai**
* Note the API endpoint and API key for use in your assistant

**🔹 IBM App ID**

* Create instance for **authentication & user management**
* Generate credentials (client ID, secret) and configure redirect URIs

**🔹 Object Storage**

* Use to store medical files or chat transcripts securely

**📄 Environment Configuration**

Create a .env file in both backend/ and frontend/ directories:

ini

GRANITE\_API\_KEY=your\_key\_here

APP\_ID\_CLIENT\_ID=xxx

APP\_ID\_SECRET=yyy

MONGO\_URI=mongodb+srv://...

BASE\_API\_URL=http://localhost:3000

**🚀 Running the Application Locally**

bash

# Start backend server

npm run start:server

# In a new terminal tab

cd frontend

npm start

* App launches at http://localhost:3000
* Test user logins via IBM App ID
* Send messages to assistant through chat interface
* View EMR and patient dashboards

**☁️ Deployment (IBM Cloud or Kubernetes)**

* 🐳 **Dockerize** the backend and frontend
* ☁️ Push images to IBM Cloud Container Registry
* 🔁 Use **IBM Code Engine**, **Cloud Foundry**, or **Kubernetes** for deployment
* 🧩 Bind services (Granite, App ID) to your deployed app

**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: oRoutes like / are defined oForm data is processed

oAI logic (currently placeholder) is added

# *Running the Application*

**🏗️ 1. Environment Setup**

**🔧 Prerequisites**

* IBM Cloud account (for access to Granite models and cloud services)
* Node.js / Python (for backend APIs)
* MongoDB / PostgreSQL (for storing patient data and logs)
* Docker (optional for containerized deployment)
* Frontend: ReactJS or Angular (if using a web-based UI)

**🔌 Services to Enable**

* IBM Granite Foundation Model (via IBM watsonx or IBM Cloud Model Deployment)
* IBM App ID (authentication)
* IBM Cloud Object Storage (for secure document storage)
* IBM Cloud Functions or Kubernetes (for scalable microservices)

**🚀 2. Application Startup Flow**

**🖥️ Local/Dev Environment**

1. **Clone the repository** from your source control platform:

bash

git clone https://github.com/your-org/healthai-granite.git

1. **Install dependencies** (Node.js or Python packages):

bash

npm install

# or

pip install -r requirements.txt

1. **Start local services**:
   * MongoDB: docker-compose up db
   * Backend API: npm run start:api
   * Frontend: npm run start:client
2. **Configure** .env **file** with keys for:
   * IBM App ID credentials
   * Granite API access token
   * Database URIs

**☁️ 3. Cloud Deployment Steps**

**Option A: IBM Cloud Functions (Serverless)**

* Deploy API functions using the IBM CLI:

bash

ibmcloud fn action create assistant-api functions/assistant.js --web true

* Configure triggers and bindings for databases, auth, and AI endpoints.

**Option B: IBM Kubernetes Service**

* Build container image:

bash

docker build -t healthai-app .

docker push registry/healthai-app

* Apply Helm chart or K8s manifest for deployment:

bash

kubectl apply -f k8s/deployment.yaml

**🧪 4. Runtime Monitoring and Logging**

* **IBM Cloud Monitoring**: Track usage, errors, and performance.
* **Custom Audit Logger**: Record user actions, data access, AI queries.
* **Alerts**: Set up triggers for unusual behavior (e.g., unauthorized access, degraded performance).

**🔍 5. Verification Checklist**

* ✅ IBM Granite responds to test prompts
* ✅ Role-based access enforced
* ✅ Patient records stored and retrieved correctly
* ✅ Audit logs generated for each sensitive transaction

**API Documentation – HealthAl: Intelligent Healthcare Assistant Using IBM Granite**

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

**1. 🔐 Authentication API**

POST /auth/token

Obtain an access token using user credentials.

Request:

json

{

"username": "doctor\_lee",

"password": "\*\*\*\*\*\*\*\*"

}

Response:

json

{

"access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expires\_in": 3600

}

**2. 💬 AI Assistant API**

POST /assistant/message

Send a query to the AI Assistant.

Request:

json

{

"user\_id": "patient\_123",

"message": "What are the side effects of metformin?"

}

Response:

json

{

"response": "Common side effects of metformin include nausea, upset stomach, and diarrhea. It’s best taken with food."

}

GET /assistant/history/{user\_id}

Retrieve past conversation history.

**3. 🏥 Patient Health** Record API

GET /patients/{id}/summary

Fetch a high-level health overview for a patient.

POST /patients/{id}/notes

Doctors can submit clinical notes.

Request:

json

{

"note": "Patient reports improved sleep patterns. Continue current prescription.",

"author": "Dr. Lee"

}

**4. 📅 Appointments API**

GET /appointments?user\_id=patient\_123

List upcoming and past appointments.

POST /appointments

Book a new appointment.

Request:

json

{

"user\_id": "patient\_123",

"doctor\_id": "dr\_lee",

"date": "2025-07-05T09:30:00",

"type": "Consultation"

}

**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 ✅ 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>

1. **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.

1. **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.

1. **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.

1. **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

**Authentication & Authorization -** **HealthAl: Intelligent Healthcare Assistant Using IBM Granite**

How Authentication & Authorization Are Handled

**🔐 Authentication – *Confirming User Identity***

Authentication ensures that only verified users—patients, doctors, administrators, or third-party systems—can access the platform.

**🔸 Mechanisms Used:**

* **OAuth 2.0 + OpenID Connect:** Industry-standard protocols for secure user logins and token-based authentication.
* **Multi-Factor Authentication (MFA):**
  + Something you know (password)
  + Something you have (OTP sent via email/SMS or authenticator apps)
  + Optional: Something you are (biometrics on mobile devices)
* **Single Sign-On (SSO):** Allows seamless login across internal and partner applications using one set of credentials.
* **IBM Cloud App ID Integration:** Enables easy authentication against enterprise directory services or social identity providers (Google, Apple, etc.).
* **Session Tokens:** Short-lived, encrypted tokens are issued after login and automatically refreshed to maintain session integrity.

**🛡️ Authorization – *Determining Access Rights***

Once authenticated, users are granted permissions based on their roles, attributes, and context.

**🔸 Authorization Layers:**

* **Role-Based Access Control (RBAC):**
  + Defined roles such as Patient, Nurse, Doctor, Specialist, and Admin.
  + Each role is assigned permissions for viewing, editing, or administering specific resources.
* **Attribute-Based Access Control (ABAC):**
  + Access decisions based on user attributes (e.g., department, specialty) and environmental factors (e.g., access time, device security).
* **Granular Permissions:**
  + Patients can view only their records.
  + Doctors can access patients assigned to them.
  + Admins can audit usage logs but not edit clinical content.
* **Dynamic Consent Management:**
  + Patients can grant or revoke data access to healthcare providers through a secure interface.
  + Complies with GDPR, HIPAA, and region-specific data protection laws.

**🔍 Additional Security & Compliance Features**

* **Audit Logs:** All access events are timestamped and logged.
* **Access Reviews:** Periodic evaluations ensure permissions remain aligned with user roles and duties.
* **Data Encryption:** All authentication and authorization data is encrypted in transit (TLS 1.3) and at rest using AES-256.
* **Policy Engine Integration:** Business logic and privacy policies are enforced through a centralized rules engine.

**IMPLEMENTATION:**

**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": "eyJhbGciOiJIUzI1NiIsInR..."

}

**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 |

**Logout**

**Clearing session on logout:**

python

@app.route('/logout') def logout():

session.clear() return redirect('/login')

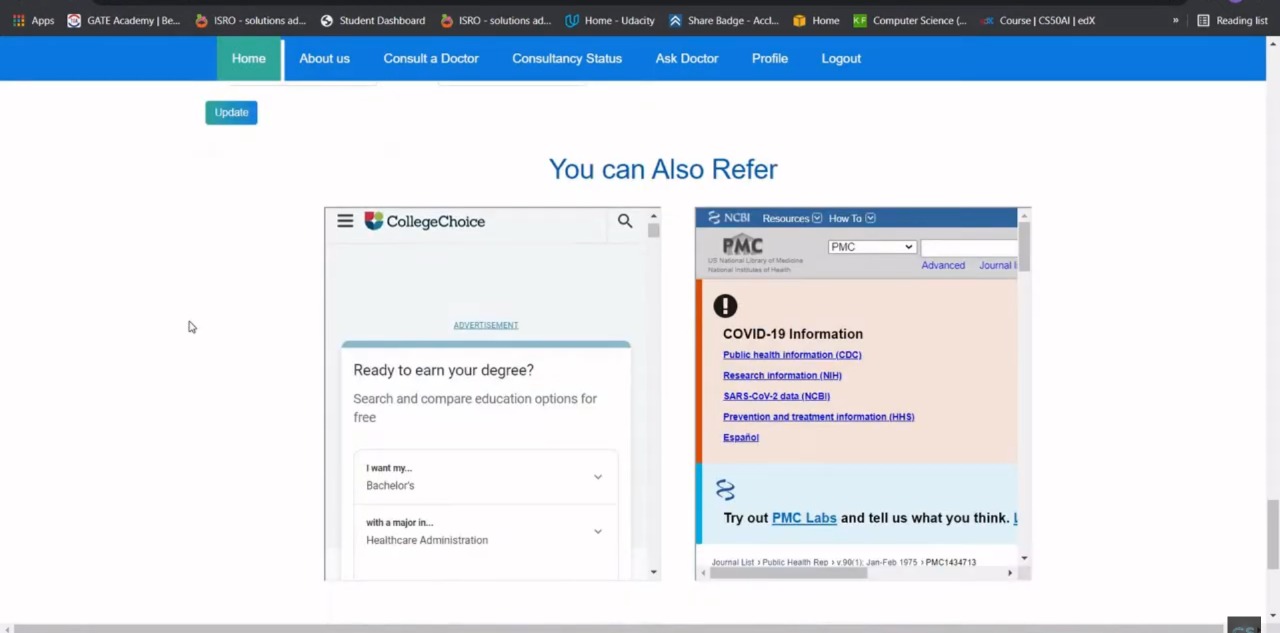
**Security Tips**

* 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

**USER INTERFACE**



**Testing:**

**Testing Strategy**

**✅ 1. Unit Testing**

* **Goal:** Verify individual components such as AI models, UI modules, and data handlers function as intended.
* **Tools:** JUnit (for backend logic), Jest (for frontend components), PyTest (for ML model checks)
* **Targets:** Natural language understanding, input validation, response accuracy.

**🧪 2. Integration Testing**

* **Goal:** Ensure interaction between modules—like chat interface with IBM Granite APIs or EMR database connections—work seamlessly.
* **Focus Areas:**
  + Chatbot ↔ AI model integration
  + Authentication and authorization workflows
  + Health record retrieval and summarization
* **Tools:** Postman, SOAP UI, Cypress.

**🌐 3. System Testing**

* **Goal:** Validate the complete, end-to-end system against requirements.
* **Includes:** Real-world scenarios like a patient asking for medication history or a doctor reviewing AI-generated insights.
* **Environment:** Replica of production setup using cloud test beds.

**🔐 4. Security Testing**

* **Goal:** Protect patient data and comply with regulations like HIPAA.
* **Tests:**
  + Authentication bypass checks
  + Role-based access validation
  + Encryption tests (at rest and in transit)
  + Penetration testing and vulnerability scans
* **Tools:** IBM AppScan, OWASP ZAP.

**🧠 5. AI/Model Validation**

* **Goal:** Ensure the IBM Granite-powered assistant responds accurately, ethically, and without bias.
* **Tests:**
  + Accuracy and relevance of medical responses
  + Bias detection in recommendations
  + Response clarity for patients and professionals
* **Method:** Manual medical expert reviews and automated quality checks.

**🧭 6. Usability Testing**

* **Goal:** Confirm intuitive UX for both patients and clinicians.
* **Approach:** Conduct user testing sessions with target personas.
* **Metrics:** Completion rate, error rate, and user satisfaction.

**📱 7. Performance and Load Testing**

* **Goal:** Evaluate system stability under high usage.
* **Simulate:** Concurrent chatbot sessions, bulk report generation, heavy data access.
* **Tools:** Apache JMeter, Locust.

**📈 8. Regression Testing**

* **Goal:** Ensure new updates don’t disrupt existing features.
* **Method:** Automated test suites run after each major change.

**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/

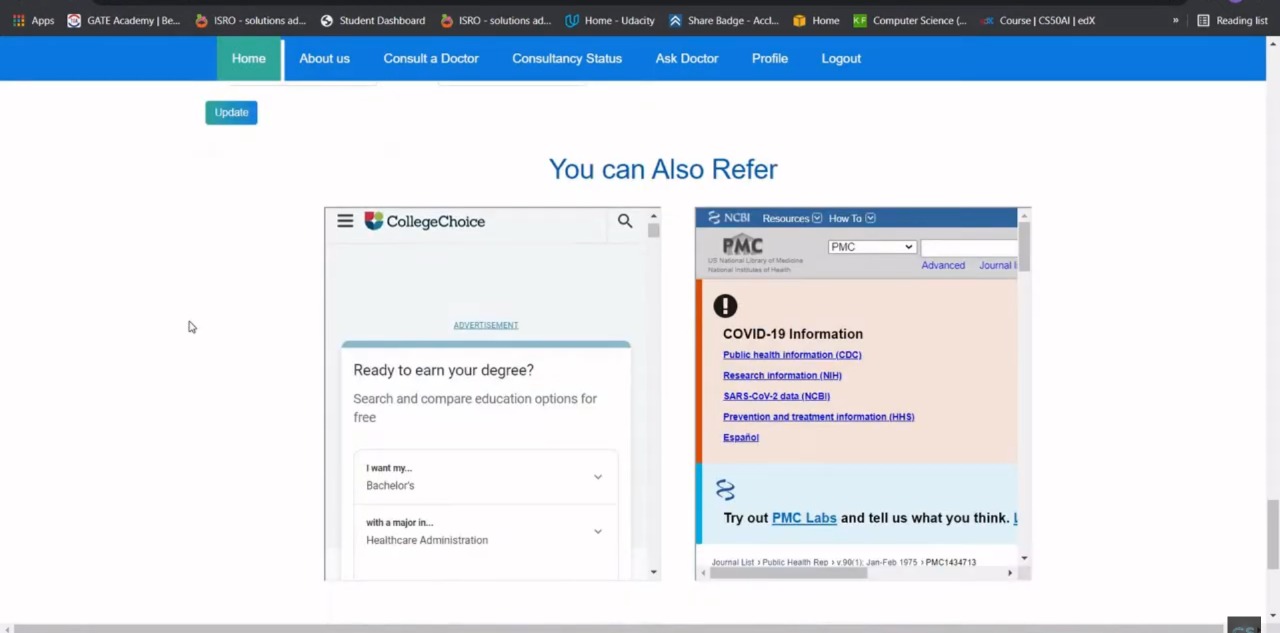
│

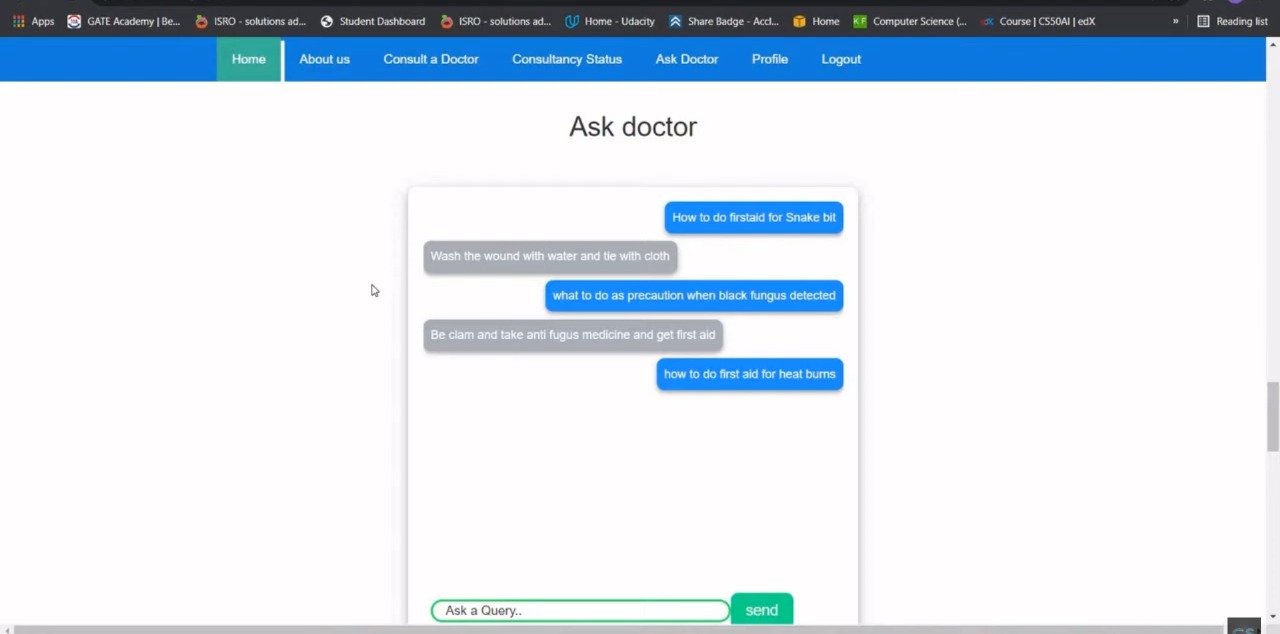
├── test\_routes.py # Route tests

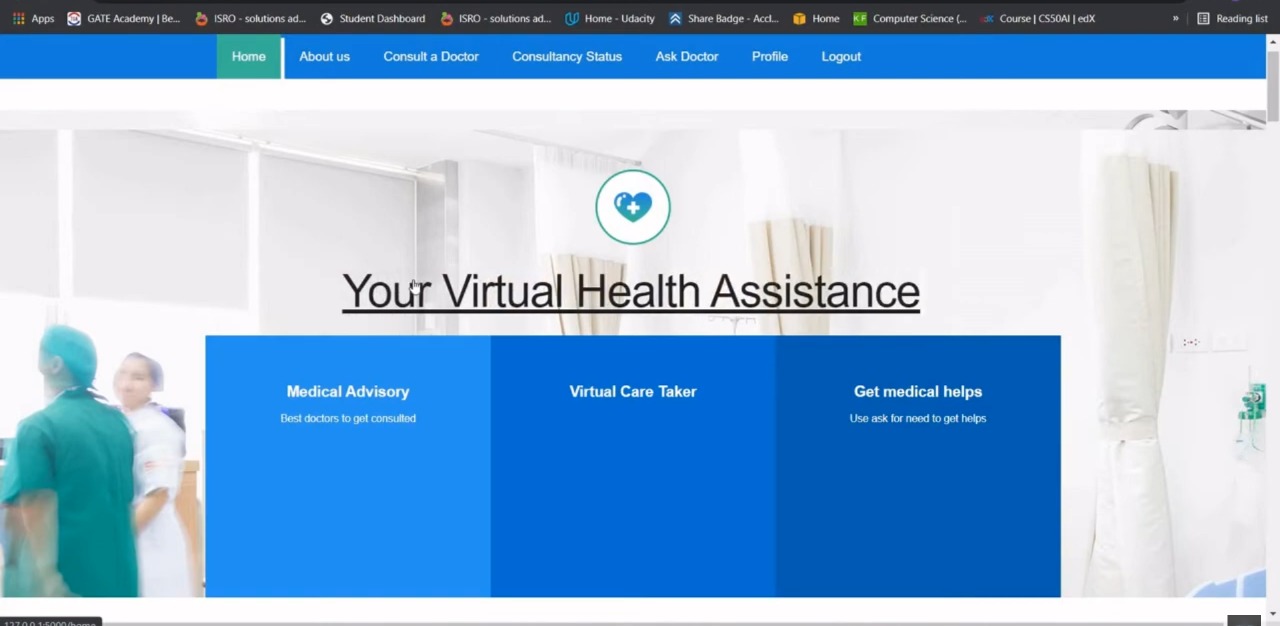
├── test\_utils.py # Utility function tests ├── test\_upload.py # File handling tests

├── test\_api.py # API JSON responses (optional)

**Screen Shots or Demo::**







**Known Issues - Intelligent Healthcare Assistant Using IBM Granite**

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

**⚠️ 1. Natural Language Understanding Challenges**

* **Ambiguous Queries:** AI may misinterpret vague or complex patient questions (e.g., “I feel weird” or “Is this bad?”).
* **Medical Jargon Mismatch:** Some users may not understand AI-generated medical terms, and the assistant may not always simplify responses adequately.
* **Multilingual Support Gaps:** Although IBM Granite supports multiple languages, real-time clinical conversation accuracy may drop in less-supported dialects or when switching between languages mid-session.

**🔐 2. Authentication & Authorization Flaws**

* **Session Timeouts:** Aggressive timeout policies may inconvenience users during long medical interactions.
* **Access Escalation Risks:** Misconfigured role-based permissions might accidentally allow broader access than intended (especially in edge cases like interns or new roles).
* **Consent Sync Issues:** Delays in reflecting updated patient consent settings across modules can result in restricted or unintended access.

**📶 3. Integration and Interoperability Issues**

* **EHR API Variability:** Integrating with external electronic health record (EHR) systems may be hindered by inconsistent data models or outdated FHIR implementations.
* **Data Sync Latency:** Real-time updates between HealthAI and third-party systems (lab results, appointment databases) may experience delays.
* **Dependency on Network Availability:** Offline access is not supported for core services, making remote or emergency settings more challenging.

**⚙️ 4. AI and Model Constraints**

* **Training Data Limitations:** IBM Granite may not be trained on rare diseases or hyper-specialized clinical cases, leading to generic or inaccurate responses.
* **Bias in Responses:** The AI could reflect biases present in its training data, such as assuming gender roles or offering culturally irrelevant advice.
* **Explainability Gaps:** Not all AI decisions are fully traceable through reasoning APIs, which can limit transparency for clinicians.

**🧪 5. Testing and Quality Assurance Gaps**

* **Inadequate Test Coverage for Edge Cases:** Certain patient scenarios (e.g., co-occurring disorders, emergency triage) may not be rigorously tested.
* **Delayed Regression Detection:** Bugs may reappear if automated test coverage isn't comprehensive or up-to-date.

**📱 6. User Experience & Accessibility Issues**

* **Interface Overload:** Providers may face information clutter due to excessive AI insights in condensed interfaces.
* **Accessibility Noncompliance:** Early versions may not fully comply with WCAG for screen readers or alternative input methods.
* **Voice Command Limitations:** Voice input accuracy could suffer in noisy hospital environments or for users with atypical speech patterns.

🚀 **Future Enhancements -**  **Intelligent Healthcare Assistant Using IBM Granite detailed information**

**1. Advanced Natural Language Understanding & Personalization**

* **Context-aware Memory:** Ability to remember past patient interactions across sessions for more personalized support.
* **Emotion & Sentiment Analysis:** Recognizing patient tone or emotional state to adjust responses compassionately.
* **Multilingual Support Expansion:** Seamless handling of regional dialects and code-switching between languages.
* **Cultural Adaptation Engine:** Modifies responses based on cultural norms and health beliefs of individual users.

**🌐 2. Expanded Integration & Interoperability**

* **FHIR 5.0 and HL7 Support:** Improved real-time interaction with EHR systems across diverse healthcare providers.
* **Wearables & IoT Device Integration:** Sync with heart rate monitors, glucose sensors, etc., for real-time health monitoring.
* **Telemedicine APIs:** Direct integration with video consultation tools and remote diagnostic platforms.

**🔐 3. Stronger Security & Privacy Controls**

* **Federated Identity Support:** Allow secure, cross-organizational logins with consistent authorization handling.
* **Fine-Grained Access Policies with AI-Based Anomaly Detection:** Automatically detect abnormal access patterns and enforce micro-policies.
* **Smart Consent Management:** Blockchain-based tracking of patient data usage and dynamic consent adjustments.

**📈 4. AI Clinical Decision Support Enhancements**

* **Explainable AI (XAI):** Provide clearer visual and text-based justifications for AI-generated recommendations.
* **Continuous Learning Loops:** AI learns from user feedback, clinician corrections, and outcomes over time.
* **Rare Disease Knowledge Expansion:** Collaborate with research networks to improve AI confidence and guidance on less-documented conditions.

**🎨 5. Intelligent User Experience Improvements**

* **Voice-first UX:** More natural voice dialogs optimized for hands-free use by both providers and patients.
* **Augmented Reality (AR) Support:** Assist providers with AR overlays for visual diagnostics or procedures.
* **Accessible UX Enhancements:** Full compliance with WCAG 2.2 including adaptive UI for different cognitive and physical needs.

**📊 6. Analytics & Insights**

* **Population Health Dashboards:** Aggregate anonymized data to detect public health trends or emerging outbreaks.
* **Predictive Patient Insights:** Use AI to flag early warning signs of readmissions, medication non-compliance, or risk of deterioration.
* **Custom Analytics Widgets:** Clinicians and administrators can design their own dashboards using drag-and-drop modules.

🌐 **7. Multilingual Support**

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

📥 **8. API-First Architecture**

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

🛡️ **9. Security & Compliance**

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

*==========================\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*======================*