**Medical AI Assistant: Disease Prediction & Treatment Planning App**

**Project Documentation**

**1. Introduction**

**Project Title:**  
Medical AI Assistant: Disease Prediction & Treatment Planning App

**Team Members:**

* + Team Member 1: Dharunya Mahalakshmi B
  + Team Member 2: Shree Varshitha R
  + Team Member 3: Maanasha A
  + Team Member 4: Pavithra M

**2. Project Overview**

**Purpose:**  
The Medical AI Assistant is designed to provide users with intelligent support for symptom analysis, possible disease prediction, and personalized treatment planning. Leveraging state-of-the-art natural language models, the app not only facilitates rapid and accessible health guidance—but also educates users about the importance of professional diagnosis. The assistant uses advanced AI models to analyze user-reported symptoms and health history, returning relevant medical information, potential conditions, and safe home and medication suggestions. Each result is accompanied by necessary disclaimers, upholding medical ethics and user safety throughout interaction.

**Features:**

* **Conversational Interface**
  + **Key Point:** Natural, clear language interaction
  + **Functionality:** Users interact with the assistant through chat and intuitive form fields, enabling detailed symptom description and health scenarios effortlessly.
* **Disease Prediction**
  + **Key Point:** Intelligent condition assessment
  + **Functionality:** Accepts a wide range of symptom inputs and employs LLM-powered analysis to suggest possible medical conditions with responsible recommendations.
* **Treatment Plan Generation**
  + **Key Point:** Personalized health guidance
  + **Functionality:** Creates tailored treatment suggestions based on user condition, age, gender, and medical history. Outputs emphasize consultation with healthcare experts and include home remedies, lifestyle changes, and general medication pointers.
* **Customizable Modern Interface**
  + **Key Point:** Professional, user-centric experience
  + **Functionality:** The Gradio frontend uses visually appealing gradients, interactive tabs, animated headings, and accessible documentation to ensure clarity and engagement for all users.
* **Data Privacy and Disclaimer Enforcement**
  + **Key Point:** Responsible AI for healthcare
  + **Functionality:** Strict messaging that outputs are informational and should never replace professional medical advice. No personal data is stored or shared.
* **Expandable Framework**
  + **Key Point:** Scalable and flexible
  + **Functionality:** Designed for easy extension with additional modules, languages, and diagnostic capabilities as needed.

**3. Architecture**

**Frontend (Gradio Blocks UI)**

The application interface is constructed using Gradio’s Blocks API, which allows for modular page layouts, interactive components, and direct linking of user inputs to backend logic. The UI offers:

* Multiple tabs for symptom analysis and treatment planning
* Custom CSS for styling, gradients, shadows, and animations
* Markdown text for helpful guidance and disclaimers
* Responsive design for desktop and mobile users

**Backend (Transformers + PyTorch)**

* The backend is powered by the Transformers library and PyTorch, providing robust integration with IBM Granite LLM models for natural language processing.
* All backend operations, including prompt construction, model inference, and output formatting, are handled within secure, isolated contexts.

**Modules and Logic**

* **Disease Prediction Module:**
  + Accepts free-text description of symptoms
  + Constructs context-aware prompts for the LLM
  + Returns structured disease suggestions and medication tips
* **Treatment Plan Generator:**
  + Uses provided medical condition, demographic data, and history
  + Outputs actionable home remedies and general medication guidelines
  + Ensures all results are accompanied by critical disclaimers

**Security and Privacy**

* No persistent data storage
* No external data transmission except model inference
* Disclaimer and privacy messaging integral to every result

**Deployment**

* Local or cloud GPU support for optimal performance
* Browser-based app, easy sharing via public Gradio links
* Isolated session for each user

**4. Setup Instructions**

**Prerequisites**

* Python 3.9 or newer
* pip and virtualenv
* Internet connection for model downloads
* Sufficient RAM/VRAM to run AI models (ideally >=8GB)
* IBM Granite model access or Hugging Face credentials if needed

**Installation Process**

1. Clone the repository from GitHub:

bash

**git clone https://github.com/your-repo/med-ai-app.git**

**cd med-ai-app**

1. Install dependencies:

bash

**pip install -r requirements.txt**

* + Typical requirements: transformers, torch, gradio

1. (Optional) Setup IBM Watsonx or Hugging Face API credentials in .env
2. Launch the application:

bash

**python app.py**

1. Open the local or public Gradio URL in your browser.

**Usage Instructions**

* Select the appropriate tab (Disease Prediction / Treatment Plan).
* Enter symptoms or medical information as prompted.
* Read and understand the output, noting disclaimers.
* For advanced use, share the link for remote access.

**5. Folder Structure**

| **Folder/File** | **Description** |
| --- | --- |
| app.py | Main script to launch Gradio dashboard and wire backend logic |
| README.md | Setup, usage, and architecture summary |
| requirements.txt | Dependency list |
| custom\_css.py | Contains custom CSS configuration for advanced UI designs |
| disease\_predictor.py | Symptom analysis and disease prediction module |
| treatment\_planner.py | Personalized health/treatment generation module |
| /assets/ | UI icons, screenshots, documentation images |
| /tests/ | Unit tests for input/output consistency |

**6. Running the Application**

1. Start the FastAPI server if an API layer is used (not required for basic demo).
2. Launch the Gradio app using your preferred method.
3. Open the link provided after launch (e.g., http://localhost:7860).
4. Use the sidebar and tabs to navigate between disease prediction and treatment planning modules.
5. Input required data and receive AI-generated responses in real time.
6. Outputs include markdown-formatted content and clearly stated disclaimers.

**7. API Documentation**

If you extend the app with a FastAPI backend or similar, standard REST endpoints can be used, such as:

* POST /predict\_disease  
  Accepts: Symptom text  
  Returns: Disease suggestion and recommendations
* POST /generate\_plan  
  Accepts: JSON with condition, age, gender, history  
  Returns: Personalized plan as Markdown text
* GET /status  
  Returns: Model health/status

API requests authenticated using JWT or OAuth for production.

**8. Authentication**

* For demonstration, the app runs in open mode without authentication.
* Secure deployments can include:
  + JWT token-based authentication
  + OAuth2 integration (IBM Cloud, Google, etc.)
  + Role-based access (admin, user, researcher)
* Future plans include session management, user history, and secure record keeping.

**9. User Interface**

The UI is minimal, modern, and optimized for accessibility:

* **Sidebar:** Navigation between major modules
* **Tabs:** Disease Prediction, Treatment Plan
* **Visualizations:** Summary cards, markdown outputs, animated headings
* **Form Layouts:** Large input fields for symptoms and medical data, tooltips for guidance
* **Disclaimer Display:** Prominent at top and bottom of results
* **PDF/Markdown Download:** Optionally allows users to export results for reference

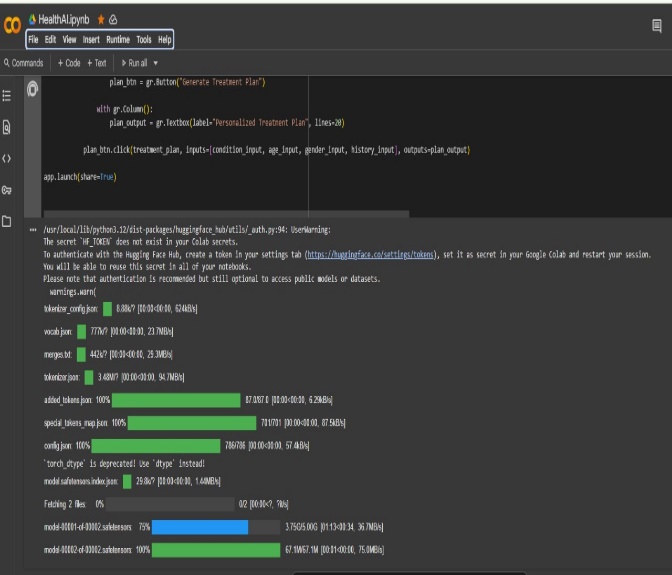
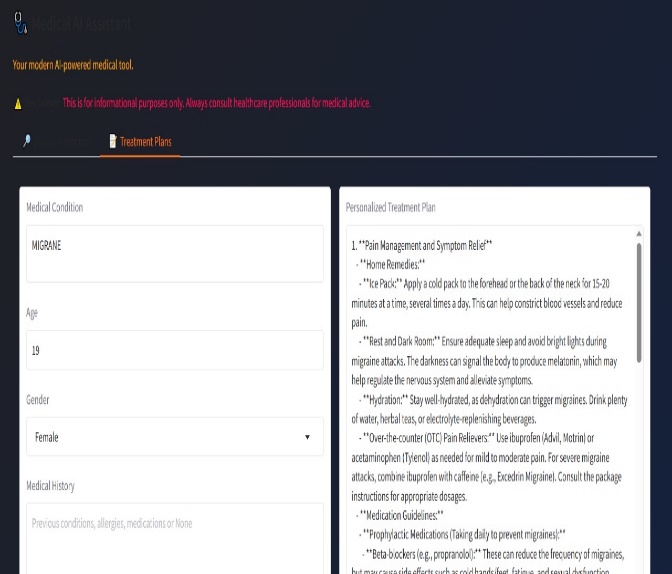
Custom CSS ensures vibrant look and feel with gradient backgrounds and soft shadows.

**10. Testing**

Comprehensive multi-phase testing includes:

* **Unit Testing:** For input validation, prompt construction, output parsing
* **API Testing:** Using Swagger UI, Postman, test scripts (if API endpoints are added)
* **Manual Testing:** For UI flows, input/output handling, incorrect/edge case inputs
* **Performance Testing:** On local and cloud hardware for model latency, scalability
* **Edge Case Handling:**
  + Malformed or empty inputs
  + Large text entries and potential abuses
  + Invalid model outputs and fallback messaging

**11. Screenshots**

**12. Known Issues**

* Outputs can sometimes be generic due to LLM limitations.
* The model does not perform real medical diagnoses or suggest prescription medicines.
* No persistent user account/history management.
* Only English supported; extension to other languages is future work.
* GPU support required for optimal inference speed.

**13. Future Enhancements**

* Integrate EHR/medical record uploads for richer context and history.
* Add support for multi-language and region-specific guidance.
* Build physician dashboard and admin analytics for broader healthcare applications.
* Expand to include more medical scenarios (e.g., emergency triage, imaging analysis).
* Enhance security with session management, encrypted data flow, and CORS support.