Project Title

HealthAI:IntelligentHealthcare Assistant

Project Documentation

1. Introduction

- **Project Title**: Health AI: Intelligent Healthcare Assistant
- **Team Members**:
  - Monisha A
  - Muthulakshmi S
  - Nandhinisri M
  - Nanmani S

2. Project Overview

Purpose:

The purpose of Health AI is to provide intelligent healthcare support to patients and doctors by leveraging AI-powered conversational assistance, health data analysis, and personalized medical insights. The system aims to improve accessibility, reduce workload for healthcare professionals, and empower patients to better manage their health.

Features:

- **Conversational Interface**

  - Natural language interaction with patients and doctors for medical queries.
- **Symptom Checker**

  - Provides possible conditions based on symptoms and suggests next steps.
- **Medical Report Summarization**

  - Converts lengthy medical reports into concise, patient-friendly summaries.
- **Medication & Appointment Reminders**

  - Notifies patients of prescribed medicine schedules and upcoming appointments.

- **Health Risk Prediction**

  - Uses AI models to forecast potential risks like diabetes, heart disease, etc.
- **Doctor Recommendation**

  - Suggests specialists based on patient symptoms and location.
- **Feedback Loop**

  - Allows patients to give feedback to improve the system.
- **Multimodal Input Support**

  - Accepts text, images (like prescriptions, lab reports), and PDFs for analysis.
- **User-Friendly Dashboard**

  - Provides intuitive access to health summaries, reminders, and recommendations.

3. Architecture

- **Frontend (Streamlit/Gradio):**
  Interactive UI for patients and doctors, including chat interface, report upload, and reminders.

- **Backend (FastAPI):**
  Handles medical data processing, chat interactions, and report summarization.

- **LLMIntegration(OpenAI / IBMWatsonx):**
  Used for natural language understanding, report summarization, and chatbot responses.

- **Database(MongoDB / PostgreSQL):**
  Stores patient data, medical history, and reminders securely.

- **ML Modules:**

  - Symptom-to-condition prediction models
  - Risk forecasting (e.g., diabetes, heart disease)
  - Anomaly detection in medical reports

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- API keys for LLM and database access
- Internet access

Installation Process:

1. Clone the repository
2. Install dependencies (`requirements.txt`)
3. Configure .env with credentials
4. Run backend server with FastAPI
5. Launch frontend (Streamlit/Gradio)
6. Upload reports or chat with the assistant

5. Folder Structure

```
app/             # FastAPI backend
app/api/          # API routes for chat, reports, reminders
ui/              # Streamlit/Gradio frontend pages
health_dashboard.py # Entry script for UI
symptom_checker.py # AI-based symptom analysis
report_summarizer.py # Summarizes medical reports
risk_predictor.py   # Predicts chronic disease risks
reminder_system.py # Medicine & appointment reminders
```

6. Running the Application

- Start FastAPI backend server
- Launch Streamlit/Gradio dashboard
- Use sidebar to navigate (chat, reports, reminders, risk predictions)
- Upload medical reports, ask queries, and receive AI-powered insights

7. API Documentation

- **POST /chat/ask** → Patient queries answered

- **POST /upload-report** → Upload and analyze medical reports
- **GET /symptom-checker** → Provides possible conditions
- **GET /risk-predict** → Predicts potential health risks
- **POST /set-reminder** → Schedule medication or appointment reminders
- **POST /feedback** → Collects patient feedback

8. Authentication

- Token-based authentication (JWT)
- Role-based access (Patient, Doctor, Admin)
- Optional OAuth2 for secure login

9. User Interface

- Sidebar navigation
- Chat with AI assistant
- Upload & summarize reports
- Health dashboard with KPIs (risks, reminders, appointments)
- Downloadable summaries/reports

10. Testing

- Unit Testing (AI models, symptom checker)
- API Testing (Postman/Swagger)
- Manual Testing (chat, reports, reminders)
- Edge Case Handling (invalid symptoms, large reports)
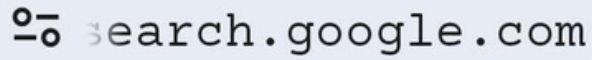
11. Screenshots

- *To be added once UI is implemented*

12. Known Issues

- Limited accuracy in rare medical conditions
- Dependency on internet for cloud AI services

13. Future Enhancements

- Integration with wearable devices (smartwatch, fitness trackers)
- Multilingual support for regional languages
- Voice-based interaction
- Emergency alert system (e.g., fall detection, abnormal vitals)

⌂ ⚬⊸ search.google.com + 12 ⋮

CO △ Healthai.ipynb ☆ ☁   💬 ⚙ 👥 Share 👤
File Edit View Insert Runtime Tools Help

🔍 Commands + Code + Text ▷ Run all ▾   Connect T4 ▾ ⌃

```python
1  import gradio as gr
2  import torch
3  from transformers import AutoTokenizer, AutoModelForCausalLM
4
5  # Load model and tokenizer
6  model_name = "ibm-granite/granite-3.2-2b-instruct"
7  tokenizer = AutoTokenizer.from_pretrained(model_name)
8  model = AutoModelForCausalLM.from_pretrained(
9      model_name,
10     torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
11     device_map="auto" if torch.cuda.is_available() else None
12 )
13
14 if tokenizer.pad_token is None:
15     tokenizer.pad_token = tokenizer.eos_token
16
17 def generate_response(prompt, max_length=1024):
18     inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
19
20     if torch.cuda.is_available():
21         inputs = {k: v.to(model.device) for k, v in inputs.items()}
22
23     with torch.no_grad():
24         outputs = model.generate(
25             **inputs,
26             max_length=max_length,
27             temperature=0.7,
28             do_sample=True,
29             pad_token_id=tokenizer.eos_token_id
30         )
31
32     response = tokenizer.decode(outputs[0], skip_special_tokens=True)
33     response = response.replace(prompt, "").strip()
34     return response
35
36 def disease_prediction(symptoms):
37     prompt = f"Based on the following symptoms, provide possible medical conditions and gener
38     return generate_response(prompt, max_length=1200)
39
40 def treatment_plan(condition, age, gender, medical_history):
41     prompt = f"Generate personalized treatment suggestions for the following patient informat
42     return generate_response(prompt, max_length=1200)
43
44 # Create Gradio interface
45 with gr.Blocks() as app:
46     gr.Markdown("# Medical AI Assistant")
47     gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult health
48
49     with gr.Tabs():
50         with gr.TabItem("Disease Prediction"):
51             with gr.Row():
52                 with gr.Column():
53                     symptoms_input = gr.Textbox(
54                         label="Enter Symptoms",
55                         placeholder="e.g., fever, headache, cough, fatigue...",
56                         lines=4
57                     )
58                     predict_btn = gr.Button("Analyze Symptoms")
59
60                 with gr.Column():
61                     prediction_output = gr.Textbox(label="Possible Conditions & Recommendatio
```

Medical AI Assistant

Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.

Disease Prediction   Treatment Plans

Enter Symptoms

e.g., fever, headache, cough, fatigue...

Analyze Symptoms

Possible Conditions & Recommendations