Hackathon Problem Statement: "Doctor's AI Assistant for Cardiology Consults"

Context

In the current healthcare environment, cardiologists are often overwhelmed with follow-up appointments and patient queries, particularly those describing symptoms over text or calls. This project aims to create a smart digital assistant that serves as a bridge between the patient and the cardiologist by:

- Understanding patient symptoms.
- Asking relevant follow-up questions based on a structured database.
- Automatically scheduling a consultation.

The goal is to build an AI-powered assistant that communicates with patients, collects detailed symptom information according to a rule-based system, and then notifies the cardiologist via Telegram and Google Calendar.

The Problem

Your task is to build a **"Doctor Assistant Agent"** with the following capabilities:

Patient Interaction

The agent will initiate a chat with the patient via a web app to collect the following information:

- Name
- Email ID
- Symptoms (e.g., chest pain, shortness of breath, fatigue)

For each symptom, the agent must query a **Symptom Rule DB** to find relevant follow-up questions, ask these questions via the chat, and collect all the patient's answers.

LLM intelligence

- Use LLM to power up the agent and act like an assistant
- The agent **must not** use its own medical knowledge.
- The agent should ask symptom-related questions based on the rule provided in the db

Doctor Notification
Once all patient data is collected, the agent must:

- **Send a whatsapp/Telegram message to the cardiologist** containing the patient's name, symptom, and all follow-up responses.
- **Schedule a Google Calendar meeting** with the patient's email, a pre-configured doctor's email, a pre-filled event description with patient details, and a meeting time (e.g., the next available 15-minute slot after one hour).

Technical Requirements

- **LLM:** Any LLM like ChatGPT or Claude
- **Frontend (ReactJS):** A chat interface for patient communication that uses WebSockets or Socket.io for real-time communication with the backend. ● **Backend (Node.js):** Receives and stores patient chat. It must interface with: ○ Generative AI API (like OpenAI or Claude) for agent implementation ○ Symptom Rule DB (SQLite/Mongo/PostgreSQL).
  - ○ Data Store DB (SQLite/Mongo/PostgreSQL).
  - ○ Telegram Bot API for doctor notifications.
  - ○ Google Calendar API for appointment creation.
- **Symptom Rule DB:** A database that stores symptoms and their associated follow-up questions. It should be structured like this example:
- **Data Store DB**: A database to store the collected information of the patient

```
{
 "chest pain": {
   "follow_up_questions": [
     "When did the pain start?",
     "Is it constant or does it come and go?",
     "Does it get worse with activity?"
   ]
 },
 "shortness of breath": {
   "follow_up_questions": [
     "How long have you been experiencing this?",
     "Does it occur during rest or activity?",
     "Do you have any other symptoms, like a cough or wheezing?"
```

```
    ]
  }
}
```

Example Workflow

1. A patient opens the chat and starts the conversation. The patient starts to talk to the LLM agent
2. LLM agent will start acting like an assistant, ask the basic questions like name, gender and email id and save it to a database
3. After that agent should ask for the symptoms
4. Patient replies with symptoms
5. The agent should query the database and find the follow-up questions and ask the patient
6. All the data collected should be saved in the database
7. The Node.js backend executes the query, retrieves the questions, and sends them to the frontend.
8. The patient answers the questions.
9. The backend sends a Telegram message to the doctor with all the patient's information and creates a Google Calendar invite.

Constraints

- **No LLM-based symptom inference:** The symptom collection questions must adhere strictly to the rules in the database.
- **Functional integrations:** The Telegram and Google Calendar integrations must be functional (test tokens or a sandbox environment are acceptable). ● **Smooth UI:** The chat interface should be responsive and provide a real-time feel without page reloads.
- **Defined logic:** The logic for determining the meeting time must be clearly defined.