Project Plan: MediGo

Agentic AI Medical Appointment Booking System.

1. Vision & Mission

Vision: To create a seamless and intelligent interface that connects patients with the right healthcare professionals, eliminating long wait times and the uncertainty of choosing the correct doctor.

Mission: To develop an AI-powered conversational agent, "MediGo," that understands a patient's health concerns through natural language (text and voice), intelligently recommends a suitable medical specialist, and books an available appointment in real-time.

2. Core Goals & Objectives

- **Reduce Patient Wait Times:** Eliminate the need for physical queues for initial consultation and booking.
- **Improve Accuracy of Care:** Ensure patients are directed to the correct specialist based on their symptoms, improving first-visit outcomes.
- **Enhance Patient Experience:** Provide a user-friendly, 24/7 accessible, and interactive platform for managing appointments.
- **Optimize Hospital Operations:** Automate the initial patient intake and scheduling process, freeing up administrative staff for more critical tasks.

3. Key Features & Functional Requirements

1. **Multi-Modal Input:**
   - The user can initiate a conversation via text input.
   - The user can use voice input, which will be transcribed into text for processing (Speech-to-Text).
2. **Intelligent Symptom Analysis:**
   - The agent will parse the user's description of their illness or symptoms.
   - If the input is vague (e.g., "I feel sick"), the agent will ask clarifying questions (e.g., "Could you tell me more about your symptoms? Are you experiencing a fever, cough, or something else?").

    - The agent will identify key symptoms and infer potential medical conditions or areas of concern.
3. **Doctor Recommendation Engine:**
    - Based on the analyzed symptoms, the agent will query a database of doctors and their specializations.
    - It will map the symptoms to the most relevant medical specialty (e.g., chest pain -> Cardiologist; skin rash -> Dermatologist).
    - It will present the recommended specialty and/or a list of suitable doctors to the user.
4. **Real-Time Slot Availability:**
    - The agent will have access to the hospital's scheduling database.
    - The user can ask for appointments on a specific date/time or ask for the next available slot.
    - The agent will check the schedule of the recommended doctor(s) and present available time slots to the user.
5. **Conversational Booking & Confirmation:**
    - The entire process will be a guided conversation.
    - The agent will confirm each critical step with the user:
        - "Based on your symptoms, I recommend seeing a *Cardiologist*. Is that okay?"
        - "Dr. Smith is available at 3:00 PM and 4:30 PM tomorrow. Which time works for you?"
    - Once a slot is chosen, the agent will ask for final confirmation before booking.
    - Upon successful booking, the agent will provide a confirmation summary (Doctor, Date, Time, Location).
6. **User Profile Management (Optional - Phase 2):**
    - Users can create a simple profile to save their details.
    - The agent can view a user's appointment history.

4. Proposed System Architecture & Technology Stack

This project is a perfect fit for a MERN stack combined with a sophisticated agentic backend.

Frontend:

- **Technology: React.js** (for a web app)
- **Responsibilities:**
    - Provide a clean, intuitive chat interface.
    - Handle user text input and microphone access for voice input.

    - Use a Speech-to-Text library (e.g., react-native-voice) to convert audio to text.
    - Communicate with the backend via REST APIs.

Backend:

- **Technology: Node.js** with **Express.js**.
- **Responsibilities:**
    - Expose API endpoints for the frontend (e.g., /chat, /appointments).
    - Manage user authentication and sessions.
    - Serve as the bridge between the frontend and the AI Agentic Core.
    - Directly handle database queries for non-AI tasks.

Database:

- **Technology: MongoDB**.
- **Responsibilities:**
    - Store data in collections:
        - doctors: (name, specialization, qualifications, schedule, etc.).
        - patients: (name, contact info, history).
        - appointments: (patientId, doctorId, dateTime, status, symptoms_summary).
        - specializations: (name, description, related_symptoms).

AI Agentic Core:

- **This is the "brain" of your application.** It will be a service called by the Node.js backend.
- **Frameworks:** A combination of **LangChain**, **LangGraph**, and **CrewAI** is an excellent choice.
    - **LangChain:** The foundation for creating "tools" that the agent can use (e.g., a check_availability tool, a book_slot tool) and for interfacing with the LLM.
    - **CrewAI:** Perfect for creating a team of specialized agents that collaborate. This makes the logic clean and modular. You can define agents with specific roles:
        1. SymptomAnalyzerAgent: Its only job is to talk to the user and figure out their symptoms.
        2. DoctorMatcherAgent: Takes the symptoms and finds the right specialty/doctor from the database.
        3. BookingCoordinatorAgent: Takes the recommended doctor and user's time preference to handle the scheduling logistics.
    - **LangGraph:** Essential for managing the multi-step, stateful conversation. The booking process is a state machine (e.g., AWAITING_SYMPTOMS ->

AWAITING_DOCTOR_CONFIRMATION -> AWAITING_SLOT_CONFIRMATION -> BOOKED). LangGraph ensures the conversation follows this logical flow without getting stuck in loops or forgetting context.

5. High-Level User & Data Flow

1. **User Interaction:** Patient opens the app and types or says, "I have a bad headache and feel dizzy."
2. **Frontend:** The React app captures the input and sends it to the backend /chat endpoint.
3. **Backend (Node.js):** The Express server receives the request and passes the user's message to the Agentic Core.
4. **Agentic Core (CrewAI + LangGraph):**
    - The SymptomAnalyzerAgent receives the message. It might ask, "How long have you had this headache?"
    - Once symptoms are clear, the task is passed to the DoctorMatcherAgent.
    - The DoctorMatcherAgent uses a LangChain tool to query the MongoDB specializations and doctors collections. It determines a "Neurologist" is appropriate.
    - The task moves to the BookingCoordinatorAgent. It informs the user about the recommendation and uses another tool to query the database for the Neurologist's schedule.
    - The agent presents the available slots. The user chooses one.
    - The agent uses a final book_appointment tool to write the confirmed appointment to the MongoDB appointments collection.
5. **Response:** The final confirmation message is passed back through the backend to the frontend and displayed to the user.