Take-Home Assignment: The Smart Scheduler AI Agent

Welcome to the NextDimension take-home assignment! This project is designed to be a practical and creative challenge that reflects the work we do. Your mission is to build an interactive chatbot that helps a user find and schedule a meeting.

The goal is to test your ability to integrate Large Language Models (LLMs) with external tools and manage a stateful, multi-turn voice based conversation—key skills for building the next generation of AI agents.

Core Task: The Smart Scheduler AI Agent

Build a chatbot that helps a user find a meeting time through a back-and-forth conversation. The agent must be able to understand the user's needs, ask clarifying questions when information is missing, and interact with a Google Calendar to find available slots.

Example Conversation Flow:

Your agent should be able to handle a conversation similar (not exactly) to this:

User: "I need to schedule a meeting."
Bot: "Okay! How long should the meeting be?"
User: "1 hour."
Bot: "Got it. I'm checking for 1-hour slots. Do you have a preferred day or time?"
User: "Sometime on Tuesday afternoon."
Bot: "Great. I have 2:00 PM or 4:30 PM available on Tuesday. Which one works for you?"

Technical Requirements

To build your agent, you must use the following stack. The goal is to test your ability to integrate a "brain" (the conversational engine) with external "tools" (the calendar).

1. Conversational Engine (LLM & Voice) 🗣️

Your agent must be voice-enabled, allowing for a natural, low-latency spoken conversation. You have two main paths to achieve this:

- **Build the Stack Manually:** Use separate services for Text-to-Speech (TTS) and Speech-to-Text (STT), and connect them to a foundational LLM provider.
    - **LLM Providers:** Google AI Studio (Gemini), OpenAI, OpenRouter, Anthropic, etc.
    - **Voice/TTS Providers:** ElevenLabs, Google's TTS, OpenAudio, etc.
- **Use an Integrated Conversational Platform:** Leverage a platform that bundles the LLM, voice, and conversational management together. This is often the fastest way to get a high-quality voice agent running.
    - **Platforms: Bland.ai**, **Retell AI**, **Vapi**, etc.

2. Orchestration & Tool Integration ⚙️

This is the logic that connects your agent's brain to its tools. **This part of the project must be built using one of the following:**

- **Python:** orchestrate by yourself.
- **No-code/Low-code: n8n** or **Make.com** are preferred for faster iteration.

How You'll Be Evaluated

We'll be looking at a few key areas:

- **Agentic Logic:** How effectively does your agent manage the conversation? Does it remember context (like the meeting duration) across turns? Does it correctly identify when to ask questions versus when to call the Calendar API?
- **Prompt Engineering:** The quality of the prompts you design to guide the LLM's conversational abilities and extract necessary information.
- **Coding & API Integration:** The quality of your code and your ability to correctly authenticate and use the Google Calendar API.
- **Voice-Enabled Agent:** Implement a voice interface. The user should be able to speak their requests, and the agent should respond with synthesized speech. The conversation should sound natural with human-expected latency.
- **Advanced Conflict Resolution:** Don't let your agent fail when there are no slots. A superior agent should handle conflicts gracefully. For example, if Tuesday afternoon is fully booked, it could suggest alternative times like "Tuesday is fully booked. Would Wednesday morning work instead?"
- **Smarter Time Parsing:** Enhance your agent's natural language understanding to handle more complex or ambiguous time-based requests, such as:

    - *"sometime late next week"*
    - *"find a time on the morning of June 20th"*
    - *"an hour before my 5 PM meeting on Friday"*
- **Problem-Solving:** How you handle the overall task and any challenges that arise.

🌟 Bonus Points: Go Above and Beyond

Show us your innovative thinking! High scores will be awarded for innovative suggestions and features that significantly enhance the project. Feel free to contribute your own unique ideas.

Submission Guidelines

To complete your submission, please provide the following:

1. **A link to your code repository** (e.g., GitHub), ensuring it is publicly accessible.
2. **A `README.md` file** in your repository that includes:
    - Clear, step-by-step instructions on how to set up and run your project.
    - A brief explanation of your design choices and how your agent works.
3. **A short video demonstration** (screen recording of 2-3 minutes) showing your chatbot in action. Please ensure the audio is clear.

Deadline

Please submit your assignment within 5 days of receiving it.

Good luck—we are excited to see what you build!

Testing Cases:

More Complex Scenarios to Test Your Agent

To further test the robustness of your agent, consider how it would handle more ambiguous and challenging scenarios like these. A truly superior agent will navigate these gracefully.

Smarter Time Parsing Challenges

- **Relative & Contextual Time:**

    - `"I need to meet for 45 minutes sometime before my flight that leaves on Friday at 6 PM."` (Requires the agent to understand a deadline and work backward).
    - `"Let's find a time for a quick 15-minute chat a day or two after the 'Project Alpha Kick-off' event on my calendar."` (Requires the agent to first query the calendar for an event and then use that event's date as a reference).
    - `"Can we schedule a 1-hour meeting for the last weekday of this month?"` (Requires date-based logic, not just simple day/time parsing).

- **Ambiguous & Vague Requests:**
    - `"I'm free sometime next week, but not too early in the morning and not on Wednesday."` (Involves multiple negative and vague constraints that need clarification).
    - `"Let's schedule our usual sync-up."` (Requires the agent to have memory or context of a "usual" meeting duration, e.g., 30 minutes).
    - `"Find a time in the evening, maybe after 7, but I need at least an hour to decompress after my last meeting of the day."` (Combines a time preference with a dynamic buffer based on another calendar event).

Advanced Conflict Resolution Challenges ⚔️

- **Changing Requirements Mid-Conversation:**
    - **User:** `"Find me a 30-minute slot for tomorrow morning."`
    - **Agent:** `"I have 9:30 AM or 11:00 AM available."`
    - **User:** `"Actually, my colleague needs to join, so we'll need a full hour. Are any of those times still available for an hour?"` (Requires the agent to re-run its search with a new duration while retaining the original context of the day and time preference).