Phase 1: Prove the Brain (CrewAI in Python, completely outside of Xcode)

Why this first? This is your biggest unknown. You need to confirm that the core conversational logic is compelling and technically feasible before you build an entire app around it. By isolating it, you can iterate much faster without getting bogged down by iOS development overhead.

Your Goal: Create a working, text-based prototype of a single AI character interaction.

Actionable Steps:

1. **Setup a Python Environment:** On your Mac, set up a virtual environment (e.g., using `venv`).
2. **Install CrewAI:** `pip install crewai crewai-tools`
3. **Build a Character Crew:** Create the Python script for one of your Echoes. Let's use "Silas, the Paranoid Cryptographer."
    - Define his `Gatekeeper Agent` (checks for a passphrase).
    - Define his `Personality Agent` (ensures he always sounds cryptic).
    - Define his `Information Broker Agent` (provides the data fragment).
    - Create a `Task` for each agent and assemble them into a `Crew` with a sequential process.
4. **Test via Command Line:** Make the script runnable from your terminal. You should be able to have a back-and-forth conversation with "Silas" by typing inputs and seeing his printed outputs. Does the passphrase logic work? Is he staying in character?
5. **Wrap it in an API:** Once the logic is solid, wrap your CrewAI instance in a simple web server using **Flask** or **FastAPI**. This is the critical step. You'll create a single API endpoint (e.g., `/chat_with_silas`) that accepts a user's message and returns the AI's JSON response.

Outcome of Phase 1: You will have a running local server with an API endpoint that is the "brain" of your character. You've de-risked the hardest part and now have a clear interface for your iOS app to talk to.

Phase 2: Build the Skeleton (MapKit in Xcode)

Why this second? The map is the primary navigation layer of your app. It's the skeleton that connects the physical world to your game events. You have experience here, so it's a good way to build momentum within Xcode.

Your Goal: Create a functional map that can trigger an event when you enter a specific zone.

Actionable Steps:

1. **Start Your Xcode Project:** Set up a new project using Swift and SwiftUI.
2. **Integrate MapKit:** Display a map, get the user's location permission, and show their current location.
3. **Mock Mission Data:** Hard-code the GPS coordinates for 2-3 of your "Echo" locations in San Francisco (e.g., one in Dolores Park, one by the Ferry Building). Display them as custom

annotations on the map.

4. **Implement Geofencing:** Use `CoreLocation` to monitor when the user's device enters a radius (e.g., 50 meters) of a mission coordinate.
5. **Trigger a Placeholder Event:** When the geofence is triggered, don't worry about AR or AI yet. Simply show a `UIAlert` or a new screen that says: "You've found the Echo. [Engage] button."

Outcome of Phase 2: A working iOS app that successfully gets the user to a physical location and knows when they've arrived.

Phase 3: Bring it to Life (ARKit + AI Integration)

Why this last? This is the integration phase where you connect your proven components. It's much easier to do this when you know the AI backend works (from Phase 1) and the location triggers work (from Phase 2).

Your Goal: When a user arrives at a location, they can see and talk to the AR character.

Actionable Steps:

1. **Create the AR View:** When the placeholder event from Phase 2 is triggered, present a new view that opens an `ARView` (using RealityKit and ARKit).
2. **Plane Detection & Placement:** Use ARKit to detect a horizontal plane (the ground). Once found, place a placeholder 3D model (even a simple sphere or cube is fine) at that location. This is your "Echo."
3. **Network Layer:** Build the networking code in Swift (using `URLSession` or `Alamofire`) to call the local Flask/FastAPI endpoint you built in Phase 1.
4. **Build the Chat UI:** In your AR view, overlay a simple text input field and a text label for the AI's responses.
5. **Wire Everything Together:**
    - User types a message in the text field.
    - Your app makes a POST request to your local AI server.
    - The server runs the CrewAI logic and returns Silas's response.
    - Your app displays the response in the text label.
6. **Polish:** Replace the placeholder sphere with your animated 3D character model. Implement voice-to-text (for user input) and text-to-speech (for the AI's response) to create a more immersive, hands-free experience.