**Technical Note: Curiosity Compass MVP**

**The Project**

The goal of this project was to take the idea of the "Curiosity Compass" and build a small, functional demo. We focused on creating a real-time multiplayer game where players can submit a word or picture, and an AI generates a riddle for others to guess. The game includes a turn-based system, timers, hints, and a scoring system, with a clear endgame.

The entire demo is a single web application, built to prove the core concept and showcase the technology stack.

**How to Play the Game**

The game is designed to be straightforward and fun. Here's a quick guide to how it works:

1. **Start a Game:** A player creates a new room from the home screen. This generates a unique room ID and a shareable link.
2. **Join the Room:** Other players can join the game by entering the room ID or by clicking the shareable link. Once everyone is in the lobby, the person who created the room can start the game.
3. **The Submission Phase (45 seconds):** The game starts with the first player's turn. This player has **45 seconds** to either type a word or upload an image of an object. The AI then processes this input and creates a new riddle.
4. **The Guessing Phase (70 seconds):** Once the riddle is ready, the timer is reset to **90 seconds**. All other players must type their guess into the input box.
   * **Hints:** If no one has guessed the riddle, hints will be revealed automatically at **2 different** times on the clock.
   * **Scoring:** The first player to guess correctly gets the most points. Other players who guess correctly a bit later receive slightly fewer points. If the timer runs out or no one guesses, no one gets points for that round.
5. **Next Turn:** After a round is over, the turn automatically passes to the next player. This continues until all players have had a chance to submit an object.
6. **End of Game:** The game consists of **5 rounds**. Once all rounds are complete, a final scoreboard is displayed showing everyone's final score.

**Technology and Trade-offs**

I chose a lean, modern tech stack to get this project up and running quickly while demonstrating key development principles.

* **Core Technologies:**
  + **Frontend:** The UI is built with **React** and styled with **Tailwind CSS**. This combination allowed for a dynamic, single-page application with a clean and responsive design, all contained in a single HTML file for easy sharing.
  + **Backend:** **Firebase** was the natural choice here. We used its **Firestore** service for real-time multiplayer functionality, which handles all the game state synchronization instantly. **Firebase Auth** provided anonymous user IDs, removing the need for a complex login system in the demo.
  + **AI:** For the AI component, I used a single, powerful model from the **Gemini API (gemini-1.5-flash)**. This was a key decision, as a single, multi-modal model can both analyze an image and generate creative text in a single API call, streamlining the entire game flow.
* **Trade-offs for the MVP:** To deliver a working prototype quickly, a few important trade-offs were made:
  + **Cost vs. Performance:** By using the gemini-1.5-flash model, we prioritized speed and cost-effectiveness. This model is highly efficient for generating short, creative text and understanding images quickly, which is perfect for a fast-paced game. A more powerful model like gemini-1.5-pro could be used for more complex riddles but would increase latency and cost.
  + **Simplicity vs. Scalability:** The entire application is a single index.html file, which makes it extremely easy to run and share. However, for a production-level product, we would split the code into separate files for better organization and maintainability.
  + **Security:** For the prototype, we used a client-side API key and anonymous authentication. This is a deliberate trade-off to enable quick testing. A real product would use a secure server to call the Gemini API, protecting the API key from being exposed.

**The Next Step: From Prototype to Product**

This demo is more than just a game; it's a solid foundation. To expand on this and build a full product, the next phases would involve:

* **Modular Architecture:** The single index.html file would be broken down into a standard multi-file React project with a dedicated backend. The core game logic would be migrated to **Firebase Cloud Functions** to prevent cheating, secure the game state, and allow you to hide the API key.
* **Enhanced User Experience:** We could add persistent user accounts, the ability for players to set their own display names, and a more robust user interface for creating and joining rooms.
* **Expanded Gameplay:** The game could be extended with new features. For example, we could introduce new game modes where the AI generates a short story about the object or suggests "mini-missions" for players to complete. We could also add a global leaderboard and achievements to increase long-term engagement.