1. Team Roles & Detailed Responsibilities

Member 1: The Game Engine Architect

- **Target:** A robust, state-machine-driven Poker Engine.
- **Tasks:**
  - **Phase 1:** Implement the core deck, shuffling, and hand-evaluation logic using the `Treys` or `Pheval` libraries for speed.
  - **Phase 2:** Build the **Betting State Machine** (Pre-flop, Flop, Turn, River). It must handle specific "actions" like Check, Call, Fold, and Raises.
  - **Phase 3:** Create a **JSON Serializer** for the game state. The React frontend needs to receive the game state in a clean format: `{ "pot": 100, "community_cards": ["As", "Kd"], "player_hand": ["Jh", "Jc"] }`.

Member 2: The Cartographer (Abstraction Lead)

- **Target:** Compressing the 10^{71} state space into a "Solveable" model.
- **Tasks:**
  - **Phase 1:** Write the **Expected Hand Strength (EHS)** calculator. Use Monte Carlo simulations to rank hands on every street.
  - **Phase 2:** Implement **Information Abstraction (Bucketing)**. Group similar hands into "strength buckets" (e.g., "Weak Flush Draw"). This prevents the AI from needing a unique strategy for every card combo.
  - **Phase 3:** Create the **Mapping Table** that links the "Live Game" cards to the "AI Strategy" buckets.

Member 3: The CFR Scientist (AI Lead)

- **Target:** The "Brain" that learns unexploitable play.
- **Tasks:**
  - **Phase 1:** Implement **External Sampling CFR (MCCFR)**. This allows the AI to train by playing only selected paths, saving massive computation time.
  - **Phase 2:** Run the **Training Loop**. The AI plays millions of hands against itself, updating "Regret" values for every action.
  - **Phase 3:** Export the final **Strategy Profile** (a lookup table or neural net) that the Backend will use to make real-time moves against the human player.

Member 4: The Infrastructure & Bridge Lead

- **Target:** Real-time communication between Python (AI) and React (User).
- **Tasks:**
  - **Phase 1:** Set up a **FastAPI** server with **WebSockets**. This is crucial for "pushing" the AI's moves to the React UI without refreshing.
  - **Phase 2:** Manage the **Experience Replay Buffer** during training (where Member 3 stores the AI's "memories").
  - **Phase 3:** Handle **Session Management**. If the user refreshes their browser, the WebSocket must reconnect them to the same game state.

Member 5: The UX & Integration Developer

- **Target:** A high-fidelity, interactive Poker Table.
- **Tasks:**
  - **Phase 1:** Create the **React Component Tree**: `<Table />`, `<Card />`, `<Controls />`, and `<Pot />`. Use Tailwind CSS for the layout.

- **Phase 2:** Implement **WebSocket Listeners**. When the server sends a "DEAL" message, React must trigger an animation to move cards to the player.
- **Phase 3:** Build the **Player Action Handlers**. Clicking "Raise" in React must send a JSON packet back to Member 4's server: `{ "action": "raise", "amount": 200 }`.

---

2. The 1-Month Implementation Roadmap

Week	Milestone	Deliverable
Week 1	**The Skeleton**	React app talks to FastAPI; Game engine can deal cards and rank hands.
Week 2	**The Brain**	EHS Bucketing is complete; CFR AI begins training on a local machine.
Week 3	**The Integration**	WebSockets are live. React UI updates when cards are dealt in the Python engine.
Week 4	**The Arena**	AI training is "frozen." Final testing: Humans play against the AI in the browser.

---

3. Software Architecture Overview

The system operates on a **Message-Response** loop. Because Poker is a sequential game, the data flow looks like this:

1. **User Action:** Human clicks "Call" in React.
2. **Transmission:** React sends `{ "type": "MOVE", "val": "call" }` via WebSocket.
3. **Processing:** Member 1's Engine updates the pot; Member 4's Bridge alerts the AI.
4. **AI Decision:** Member 3's AI looks up the best move in the Strategy Table.
5. **State Update:** Member 1 serializes the new state (e.g., it's now the "Flop").
6. **Visual Update:** WebSocket broadcasts the new state; Member 5's React UI animates the community cards.

---

4. Key Technical Necessities

- **Backend:** Python 3.10+, FastAPI, `websockets`, `Treys` (Hand evaluation).
- **Frontend:** React (Vite), Tailwind CSS, `Lucide-React` (for icons).
- **Communication:** JSON over WebSockets (standard for real-time games).
- **Training:** Minimum 16GB RAM and a multi-core CPU (for Member 3's CFR simulations).