PRESEARCH.md

Project Goal: Deliver a production-scale collaborative whiteboard with bulletproof multiplayer synchronization and a natural language AI agent. Project completion is a hard gate for Austin admission. A simple, solid board built "correctly" beats a feature-rich broken one.

Phase 1: Define Your Constraints

- **1. Scale & Load Profile:** * *Users & Traffic:* MVP target is 5+ concurrent users with spiky traffic during collaborative sessions.
  - *Performance:* High real-time requirements. Must maintain 60 FPS while supporting 500+ objects. Cursor sync latency must be <50ms and object sync latency <100ms.
  - *Scale Strategy:* By adopting a Conflict-free Replicated Data Type (CRDT) architecture, the system is fundamentally designed to gracefully scale to thousands of concurrent users without the merge conflicts or "ghost locks" experienced in traditional setups.
- **2. Budget & Cost Ceiling:** * *Budget:* $450 per month (program-sponsored).
  - *Trade-offs:* With a generous monthly budget, I will aggressively trade money for time. The budget will primarily fund premium AI APIs (Claude via Max subscription) and managed infrastructure to buy back development hours, ensuring I stay close to a 40-hour workweek. Pay-per-use scaling is highly preferred.
- **3. Time to Ship:** * *Timeline:* 1-week sprint. MVP due in 24 hours; early submission in 4 days; final in 7 days.
  - *Priority:* Speed-to-market is the primary objective. However, the architectural mandate is to solve the multiplayer problem *correctly*. We will prioritize long-term maintainability by adopting industry-standard CRDTs (Yjs) over managed WebSockets rather than hacking together a brittle, custom syncing engine.
- **4. Compliance & Regulatory Needs:** * *Requirements:* None. This is an admission/portfolio project. No HIPAA, GDPR, or SOC 2 requirements apply.
- **5. Team & Skill Constraints:** * *Team:* Solo developer.
  - *Skills:* Strongest in React + TypeScript.
  - *Strategy:* I am taking on the learning curve of implementing Yjs to solve the problem "correctly" for maximum scale. To mitigate the risk to my shipping speed, I will use Claude Code (Anthropic CLI) as my senior pairing partner to execute the architectural transplant and generate boilerplate.

Phase 2: Architecture Discovery

- **6. Hosting & Deployment:** * *Decision:* **Vercel** (Frontend SPA & Serverless Functions).
  - *Why:* Zero-configuration deployment for React/Vite with automatic CI/CD. Vercel Serverless Functions will securely handle our AI API proxy calls.
- **7. Authentication & Authorization:** * *Decision:* **Firebase Auth** (Email/Password).
  - *Why:* Salvaging this from my V1 build strictly for speed. It provides a fast, drop-in UI and JWT generation to meet the MVP user authentication requirement without wasting time rebuilding auth.
- **8. Database & Data Layer (Real-time Engine):** * *Decision:* **Yjs (CRDT) + PartyKit** (Managed WebSocket Server).
  - *Why:* This is the "build it right" solution. Yjs mathematically guarantees that all users end up with the same canvas state, eliminating the race conditions seen in standard Firestore/RTDB polling. PartyKit (running on Cloudflare Edge) provides a zero-config, infinitely scalable WebSocket backend built specifically to host Yjs documents.
- **9. Backend/API Architecture:** * *Decision:* Serverless / Edge WebSockets.
  - *Why:* The React client connects directly to PartyKit rooms for real-time CRDT sync. Vercel Serverless Functions are used strictly as a secure backend to call the AI Agent APIs without exposing keys.
- **10. Frontend Framework & Rendering:** * *Decision:* **React + TypeScript (Vite) + `react-konva`**.
  - *Why:* Single Page Application (SPA). SEO and SSR are unnecessary. Konva allows declarative 2D canvas manipulation that pairs perfectly with React state, enabling the infinite board, sticky notes, and shapes required for the MVP.
- **11. Third-Party Integrations:** * *AI APIs:* **Anthropic Claude API** (Primary via Max subscription) with **xAI (Grok)** as a cheap, personal fallback.
  - *Tooling:* **LangChain**.
  - *Why:* LangChain abstracts the LLM provider. This allows me to write complex schema parsing for the AI Board Agent once, and hot-swap between Claude and xAI instantly via environment variables.

Phase 3: Post-Stack Refinement

- **12. Security Vulnerabilities:** * *Mitigation:* AI API keys will be securely stored in Vercel environment variables and executed via serverless functions. They will never be exposed to the browser.
- **13. File Structure & Project Organization:** * *Refactor Strategy:* "Hybrid Transplant." I will instantiate a clean Vite project (V2) to host the new Yjs/PartyKit state layer, and strategically port over the battle-tested pure UI components (`ShapeRenderer`, `Canvas`, `DetailPane`) from my V1 repository.
  - *State Management:* Stripping out the V1 global Zustand shape store and replacing it with a Yjs-backed React hook (`useYjsCanvas`), keeping the Konva UI components agnostic to the network layer.
- **14. Naming Conventions & Code Style:** * *Standards:* Strict TypeScript typing, ESLint, and Prettier. Strict typing is especially critical when defining the JSON schemas that LangChain will force the LLMs to output.
- **15. Testing Strategy:** * *Unit:* Vitest for testing pure functions (LangChain Parsers, canvas math operations).
  - *Integration/E2E:* Due to the 1-week timeline, multiplayer testing will primarily be manual E2E using multiple incognito browser windows to continuously test concurrent users, simultaneous edits, and network throttling (verifying CRDT merge behavior and reconnection).
- **16. Recommended Tooling & DX:** * *Workflow:* AI-first development methodology utilizing **Claude Code** (Anthropic CLI) directly in the terminal to read the old V1 file system and safely transplant the logic into the new V2 CRDT architecture.