Pre-Search Checklist Answers: CollabBoard

Patrick Lynch

Status: Complete-----**## Phase 1: Define Your Constraints**1. Scale & Load Profile

- **Users at launch / in 6 months:** Planning assumption is ~25 total users at launch and ~250 total users in 6 months. Hard requirement is supporting 5 concurrent users on one canvas, with short-term bursts up to ~10 concurrent users across multiple boards.
- **Traffic pattern (steady, spiky, unpredictable):** Spiky/unpredictable.
- **Real-time requirements (websockets, live updates):** Non-negotiable real-time object synchronization and live cursor visibility for every active user on the canvas. Low-latency requirement (targeting sub-100ms object sync and sub-50ms cursor updates).
- **Cold start tolerance:** Brief wake-up delay is acceptable to reduce cost, but once warm the app must remain snappy and responsive.

2. Budget & Cost Ceiling

- **Monthly spend limit:** Prefer free tier; otherwise keep costs very low.
- **Pay-per-use vs fixed costs:** Prefer small fixed hosting cost over unpredictable variable spend.
- **Where to trade money for time:** Willing to spend where needed to get the product running and shipped on time (especially for managed services that reduce implementation risk).

3. Time to Ship

- **MVP timeline:** Hard target is the 24-hour checkpoint (tomorrow). MVP must include infinite board with pan/zoom, editable sticky notes, at least one shape type, create/move/edit objects, real-time sync for 2+ users, multiplayer cursors with names, presence awareness, user authentication, and public deployment.
- **Speed-to-market vs long-term maintainability:** Explicitly prioritize speed-to-market for this submission. Use only lightweight maintainability safeguards needed to avoid delivery risk.
- **Iteration cadence after launch:** Ship once with minimal follow-up, unless fixes are required for evaluation/completion.

4. Compliance & Regulatory Needs

- **HIPAA:** Not applicable.
- **GDPR:** In scope (EU users should be supported). Keep implementation lightweight for MVP while avoiding obvious GDPR issues.
- **SOC 2 / enterprise needs:** Out of scope for MVP.
- **Data residency requirements:** No strict residency constraint for MVP; choose region/provider based on fastest reliable setup.

5. Team & Skill Constraints

- **Solo or team:** Solo developer orchestrating AI coding agents (LLMs) for implementation.
- **Known languages/frameworks:** Most comfortable with React; leaning toward Next.js and deployment on Vercel for speed and familiarity.
- **Learning appetite vs shipping speed:** Strongly prioritize shipping speed. Avoid learning-heavy architecture, but willing to use Firebase with step-by-step guidance because it is likely faster and safer than building custom WebSocket infrastructure.

-----**## Phase 2: Architecture Discovery**6. Hosting & Deployment

- **Serverless vs containers vs edge vs VPS:** Default to managed/serverless setup with Next.js on Vercel and Firebase services; revisit only if a blocker appears.
- **CI/CD requirements:** Simple push-to-deploy workflow is sufficient for MVP. Add stricter checks (lint/tests) later if needed.
- **Scaling characteristics needed:** Optimize for required MVP load (5+ concurrent users per board) and defer broader scaling concerns until after submission unless performance issues appear.

7. Authentication & Authorization

- **Auth approach:** Firebase Authentication using Google sign-in for fastest setup, with optional email-link fallback if needed.
- **RBAC needed?:** Yes. Board-level permissions are required: owner-managed access with `read/edit` control, default open-edit mode, and per-board ability to disable open-edit and enforce an explicit editor allowlist.
- **Multi-tenancy considerations:** User-scoped board ownership with a cap of 3 boards per user. Each board has its own access policy (open-edit by default or restricted allowlist).

8. Database & Data Layer

- **Database type:** Firestore document database as primary data layer for board state, permissions, and collaboration metadata.
- **Real-time sync / search / vector / cache needs:** Real-time sync is required. Full-text search, vector storage, and dedicated caching layer are not required for MVP.
- **Read/write ratio:** Write-heavy workload expected (frequent object mutation and presence/cursor updates).

9. Backend/API Architecture

- **Monolith vs microservices:** Single Next.js monolith for MVP (frontend + backend routes together).
- **API style (REST/GraphQL/tRPC/gRPC):** REST-style API routes for MVP.
- **Background jobs/queues:** Not required for MVP hard-gate scope. Handle key flows synchronously; revisit queues only if post-MVP AI or async workloads demand it.

10. Frontend Framework & Rendering

- **SEO requirements:** Not important for MVP.
- **Offline support / PWA:** Out of scope for MVP.
- **SPA vs SSR vs static vs hybrid:** Hybrid in Next.js. Use client-rendered SPA behavior for the real-time board/canvas route, while keeping surrounding app pages simple with default Next.js rendering where useful.

11. Third-Party Integrations

- **External services needed:** Firebase (Auth + Firestore), Vercel (deployment/hosting), and OpenAI API for AI board commands. OpenAI API key setup is a planned follow-up step.
- **Pricing cliffs / rate limits:** Accept MVP pricing/rate-limit risk with basic guardrails: cap prompt and completion tokens per request, add simple per-user/per-board request throttling, set conservative daily AI usage limits, and log token usage/cost so limits can be tuned.
- **Vendor lock-in risk posture:** Accept lock-in for MVP speed. Mitigate with thin internal wrappers/adapters around Firebase and OpenAI integrations to preserve swap options later.

-----## Phase 3: Post-Stack Refinement12. Security Vulnerabilities

- **Known stack pitfalls:** Overly permissive Firestore rules, relying on client-side permission checks, exposing OpenAI/Firebase admin secrets in client code, unvalidated AI tool inputs, and high-frequency collaboration writes that can amplify abuse/cost.
- **Common misconfigurations to avoid:** `allow read, write: if true` Firestore rules, missing server-side auth/board-access checks in API routes, no per-user/per-board throttling on AI endpoints, logging sensitive tokens/prompts, and missing board-level validation when open-edit is disabled.
- **Dependency risks strategy:** Keep dependencies minimal, pin versions with lockfile committed, run regular `npm audit`/dependency updates, avoid unmaintained canvas/collab packages, and patch high-severity vulnerabilities immediately during MVP window.

13. File Structure & Project Organization

- **Folder structure standard:** Use standard Next.js App Router layout with `src/app` routes, `src/features/board` for whiteboard/collaboration logic, `src/components` for shared UI, `src/lib/firebase` for Firebase config/wrappers, and `src/server` for API/auth/permission helpers.
- **Product structure requirement:** Include a user boards management page for listing owned/accessible boards and supporting create/delete actions.
- **Monorepo vs polyrepo:** Single repository.
- **Feature/module organization:** Organize primarily by feature modules (for example

`board`, `auth`, `ai`, `boards-list`) to keep implementation clear and iteration fast.

14. Naming Conventions & Code Style

- **Naming patterns:** Follow standard TypeScript/Next.js conventions: `camelCase` for variables/functions, `PascalCase` for React components/types, and `kebab-case` for route segments.
- **Linter/formatter config:** Use default Next.js ESLint configuration plus Prettier with minimal custom rules for MVP speed.

15. Testing Strategy

- **Unit / integration / e2e tools:** Aim to include all three at MVP-appropriate depth if time allows: targeted unit tests, minimal integration tests for critical flows, and 1-2 end-to-end smoke tests (auth + core collaboration path).
- **Coverage target for MVP:** No strict percentage target for MVP. Success criterion is reliable coverage of hard-gate behaviors (pan/zoom, object CRUD, real-time sync, cursors, presence, auth, and deployed accessibility) with focused tests on the riskiest collaboration flows.
- **Mocking patterns:** Mock OpenAI responses for deterministic and low-cost tests, use Firebase Emulator (or thin integration stubs) for auth/data integration tests, and avoid heavy mocking of core board-collaboration behavior in higher-level tests so multiplayer logic is validated realistically.

16. Recommended Tooling & DX

- **VS Code extensions:** Keep a minimal set: ESLint, Prettier, Firebase tooling, and Tailwind CSS IntelliSense if Tailwind is used. Primary development workflow is Codex app / Codex CLI in terminal, with VS Code extensions as secondary support.
- **CLI tools:** Use a minimal CLI stack: `firebase-tools`, `vercel`, `npm` scripts, and `eslint`/`prettier` commands, with Codex CLI as the primary coding interface.
- **Debugging setup:** Use browser DevTools + React DevTools for client debugging, run Firebase Emulator/console checks for auth and Firestore behavior, and rely on Vercel + Next.js server logs for backend issues. For collaboration bugs, test with multiple browser sessions and network throttling. Use simple structured API logging (boardId, userId, action, latency, error) while avoiding sensitive data in logs.

-----## Tradeoff Log

Decision	Alternatives Considered	Why Chosen	Risk Accepted

Next.js monolith	Microservices split	Fastest path for solo MVP delivery with low coordination overhead	Tighter coupling and less long-term service isolation
Vercel + Firebase managed stack	Custom infra, VPS, container platform	Minimal ops burden and fastest production deployment	Vendor lock-in and hosted platform limits
Firestore for collaboration data	Supabase Postgres/realtime, custom WebSocket datastore	Good fit for real-time document sync and rapid setup	Cost/perf sensitivity on high-frequency writes and schema/query constraints
Firebase Auth (Google-first)	Custom auth, other OAuth providers first	Quick secure login implementation and low auth surface area	Dependency on Firebase identity model and provider availability
REST API routes	GraphQL/tRPC/gRPC	Simple and fast for narrow MVP API surface	Less typed end-to-end ergonomics than tRPC and less flexible query modeling than GraphQL
No background queues in MVP	Event bus/job workers from day one	Keeps architecture simple and focused on hard-gate features	Some async tasks may need retrofit later
Hybrid Next.js with client-rendered board	Heavy SSR strategy	Real-time canvas behavior is primarily client-side; reduces complexity	Less SSR benefit on board route
OpenAI integration with basic usage guardrails	No AI, or AI with advanced quota/billing controls	Meets AI-agent requirement quickly while controlling spend enough for MVP	Basic limits may be coarse and need tuning

Lock-in accepted with adapter wrappers	Full abstraction from day one	Maintains speed while preserving future swap path	Wrapper coverage may be incomplete initially