CollabBoard Pre-Search

Completed: Feb 16, 2026
Tom Fuertes / @tomfuertes / tomfuertes@gmail.com

Phase 1: Define Your Constraints

1. Scale & Load Profile

- **Users at launch?** 5-10 (cohort demo and judging). Architecture supports horizontal scaling - each board is an independent Durable Object, so 1000 concurrent boards = 1000 independent DO instances running on Cloudflare's edge network.
- **In 6 months?** If productionized: architecture handles viral growth naturally. DOs scale per-board. Cost analysis projects 100/1K/10K/100K users.
- **Traffic pattern?** Spiky for the exercise (demo sessions). Architecture handles steady traffic identically - DOs wake on demand, hibernate when idle. No scaling configuration needed.
- **Real-time requirements?** Yes, mandatory. WebSockets for cursor sync (<50ms target) and object sync (<100ms target). Durable Objects provide single-hop WebSocket connections at the edge.
- **Cold start tolerance?** DO hibernation wake is ~50ms. Acceptable. First user to open a dormant board sees a brief pause, then full-speed operation.

2. Budget & Cost Ceiling

- **Monthly spend limit?** Development week: ~$30-60 total. CF Workers paid plan: $5/mo. LLM API (development + testing): ~$20-50. D1 within free tier.
- **Pay-per-use acceptable?** Yes, ideal. CF Workers, D1, DO requests, and LLM APIs are all pay-per-use. Zero cost when idle. No fixed infrastructure beyond the $5/mo Workers plan.
- **Where trade money for time?** (1) LLM API for AI agent quality - pay for good responses vs building custom NLP. (2) Better Auth library vs rolling auth from scratch. (3) react-konva vs raw canvas - trade library dependency for saved development hours on transforms, hit detection, layering.
- **Unit economics framing:** For SaaS productionization - main cost driver is LLM API calls per user session. Caching identical prompts via CF AI Gateway reduces redundant calls. Break-even analysis will be part of the cost analysis deliverable.

3. Time to Ship

- **MVP timeline?** 24 hours (hard gate). Gates are reference points for self-managed pacing.
- **Speed vs maintainability?** Full quality from day 1. TypeScript strict, ESLint, Prettier, clean separation of concerns. But pragmatic - no edge case rabbit holes. Build it right the first time to avoid rewrites during the AI agent phase.
- **Iteration cadence?** Continuous deployment via Cloudflare git integration (auto-deploy on push to main). GitHub Actions for lint and test checks.

4. Compliance & Regulatory Needs

- **HIPAA?** No health data. Not applicable.
- **GDPR?** Not required for exercise. For production: board content could contain PII from EU users. Roadmap includes privacy policy page and data deletion endpoint. CF edge deployment means data is distributed - can pin DOs to specific regions if data residency is required.
- **SOC 2?** Not required. Roadmap item for enterprise sales.
- **Data residency?** CF DOs colocate with users automatically. Can be pinned to specific jurisdictions if needed. Documented in roadmap as a fast-follow for enterprise requirements.
- **Decision:** Implement basic privacy policy page and data deletion endpoint as part of polish phase. Document full compliance path in `docs/roadmap.md`.

5. Team & Skill Constraints

- **Solo or team?** Solo developer with AI pair programming (Claude Code + Cursor). AI-first development methodology documented in AI Development Log.
- **Languages/frameworks known well?** Cloudflare ecosystem (Workers, DOs, D1, WebSockets), Better Auth, React, TypeScript. Durable Object WebSocket patterns are familiar.
- **Learning appetite vs shipping speed?** Learn while building. Invest in understanding react-konva and CF Workers AI patterns properly rather than copy-pasting. The Pre-Search document itself is a learning artifact. Prioritize understanding the "why" behind architectural decisions.

Phase 2: Architecture Discovery

6. Hosting & Deployment

- **Serverless vs containers vs edge vs VPS?** Edge. Cloudflare Workers (edge compute) + Workers Static Assets (CDN for frontend). No containers, no VPS. Durable Objects provide stateful edge compute. Zero ops burden.

- **CI/CD requirements?** Cloudflare git integration for auto-deploy on push to main (configured in CF dashboard). GitHub Actions for lint + test checks only. No wrangler CLI in CI.
- **Scaling characteristics?** DOs scale horizontally per board (each board is independent). CDN-cached static assets for frontend - effectively infinite scale. Per-board DO handles thousands of WebSocket messages/sec. Well beyond the 5+ user requirement. At scale: DO request pricing is $0.15/million. D1 free tier: 5M reads/day.

7. Authentication & Authorization

- **Auth approach?** Email/password via Better Auth with Cloudflare D1 for sessions. No email verification for MVP (reduces complexity, no email service dependency). Social OAuth (GitHub/Google) on roadmap as fast-follow.
- **RBAC needed?** No. Anyone with the board URL can join and edit. Security by obscurity via GUID-based board URLs. Board access model is intentionally simple for the exercise. Roadmap documents path to owner/editor/viewer roles.
- **Multi-tenancy?** Board-level tenancy. Each board is a separate Durable Object with isolated storage. Users can create multiple boards. Board list in D1. No org/workspace abstraction for MVP.

8. Database & Data Layer

- **Database type?** Hybrid. (1) DO Storage (KV) for board objects - colocated with real-time sync logic, sub-millisecond reads. (2) Cloudflare D1 (SQLite) for users, sessions, board metadata - relational queries.
- **Real-time sync?** Yes, via DO WebSocket broadcast. No full-text search needed. No vector storage (AI agent uses function calling, not RAG). No explicit cache layer - DO storage IS the cache (colocated).
- **Read/write ratio?** ~100:1 during active collaboration. Cursor broadcasts (reads to all connected clients) dominate. Object mutations (writes) are less frequent. Most "reads" are WebSocket message sends, not storage reads.

9. Backend/API Architecture

- **Monolith or microservices?** Monolith Worker with Hono routing framework. Simple and sufficient for the scope.
- **REST vs GraphQL vs tRPC?** REST for auth endpoints (Better Auth) and board CRUD. WebSocket for all real-time communication. No GraphQL/tRPC complexity needed.
- **Background jobs?** None needed. All operations are synchronous within the request/WebSocket lifecycle.

10. Frontend Framework & Rendering

- **SEO requirements?** None. Whiteboard app behind auth has no SEO needs. React SPA via Vite.

- **Offline support/PWA?** Not required. Real-time collaboration requires connectivity by definition.
- **SPA vs SSR?** SPA. Deployed as Cloudflare Workers static assets (formerly Pages).

11. Third-Party Integrations

- **External services?** (1) Workers AI binding (free) for AI agent - `env.AI.run()` with `runWithTools()` for function calling. (2) CF AI Gateway as upgrade path to Claude/GPT-4 if free models lack quality. (3) Better Auth (library, not service). No payments, email, or analytics services for MVP.
- **Pricing cliffs and rate limits?** D1 free tier: 5M reads/day, 100K writes/day. Workers AI free tier included in paid plan. AI Gateway: transparent proxy, no additional cost for caching/logging. Main cliff at scale: LLM API costs if upgrading to Claude/GPT-4 (token-based pricing).
- **Vendor lock-in risk?** Low. CF Workers uses standard Web APIs (fetch, WebSocket, crypto). DOs are CF-specific but the pattern (WebSocket server with state) ports to any platform. Better Auth is framework-agnostic. AI agent uses standard function calling patterns portable across LLM providers. Only lock-in is DO storage format - trivially exportable JSON KV.

Phase 3: Post-Stack Refinement

12. Security Vulnerabilities

- **Known pitfalls?** XSS via user-input text rendered on canvas - sanitize all text content. WebSocket connection hijacking - validate Better Auth session cookie before upgrading. CSRF on auth endpoints - Better Auth includes CSRF protection.
- **Common misconfigurations?** Exposing AI API keys to client bundle (mitigated: all AI calls are server-side in Worker). Permissive CORS on WebSocket endpoint (mitigated: validate origin). D1 SQL injection (mitigated: parameterized queries via Better Auth/ORM).
- **Dependency risks?** react-konva is actively maintained. Better Auth is actively maintained with CF support. Workers AI is a first-party CF service. Low supply chain risk.

13. File Structure & Project Organization

- **Monorepo vs polyrepo?** Monorepo. Single repo, single git history. Client and server code in `src/client/` and `src/server/` with shared types in `src/shared/`.
- **Feature/module organization?** Feature-based on client side (e.g., `src/client/features/board/`, `src/client/features/auth/`). Keeps related components, hooks, and utils colocated.

14. Naming Conventions & Code Style

- **Naming patterns?** camelCase for variables/functions, PascalCase for React components and TypeScript types/interfaces. File names: kebab-case for utilities, PascalCase for React components.
- **Linter and formatter?** ESLint with TypeScript strict config. Prettier for formatting. Enforced via GH Actions.

15. Testing Strategy

- **Unit, integration, e2e tools?** Manual 2-browser testing as primary validation throughout development. Playwright for e2e tests if time permits in polish phase.
- **Coverage target for MVP?** No coverage target. Speed priority with pragmatic quality. Full quality code (TS strict, clean patterns) but no automated test suite for MVP.
- **Mocking patterns?** N/A for MVP. If adding tests later: mock WebSocket connections and DO storage for unit tests.

16. Recommended Tooling & DX

- **AI development tools (required by spec):** Claude Code (primary, CLI-based pair programming) + Cursor (secondary, IDE-based). Documents AI-first development workflow.
- **CLI tools:** wrangler for local CF development. Chrome DevTools for network throttling and performance profiling.
- **Debugging setup:** Two browser windows side-by-side as primary dev workflow. Console logging for WebSocket messages. React DevTools for state inspection.

Architecture Decision: CRDTs vs Durable Objects

Evaluated CRDTs (specifically LWW Registers and LWW Maps from state-based CRDT literature). These solve distributed peer-to-peer consistency without a central coordinator.

Decision: Durable Objects instead of CRDTs. DOs provide a central coordinator per board. All mutations serialize through one instance, giving LWW semantics by construction. CRDTs would be needed for offline editing (make changes while disconnected, merge on reconnect) - a feature not required by the spec. The simpler architecture wins.

Trade-off acknowledged: No offline editing capability. If users disconnect, they must reconnect to continue. This is acceptable for a real-time collaboration tool where connectivity is assumed.