CollabBoard

Pre-Search Document

Completed before writing any code. Documents architecture decisions, tradeoffs, and rationale.

Stack	React 18 + Vite + TypeScript + react-konva + Tailwind CSS
Backend	Firebase (Firestore + RTDB + Auth + Hosting + Cloud Functions v2)
AI	Anthropic Claude Sonnet 4.5 with function calling
Deployment	Firebase Hosting (firebase deploy)
Timeline	MVP in 24hrs | Full in 4 days | Polish in 7 days

Phase 1: Define Your Constraints

1. Scale & Load Profile

Question	Decision
Users at launch?	5-10 (demo/evaluation)
Users in 6 months?	100-500 (if extended beyond sprint)
Traffic pattern	Spiky - concurrent users during collaboration sessions
Real-time requirements	Critical - WebSocket-based live sync for cursors (<50ms) and objects (<100ms)
Cold start tolerance	Low - users expect instant board load. No serverless cold starts on sync layer.

Rationale: This is a real-time collaboration tool. Latency is the primary constraint. The sync layer must be always-on with sub-100ms delivery. Firebase Realtime Database provides this out of the box with no cold start penalty. The AI layer uses Cloud Functions v2 with minInstances=1 to avoid cold starts on the AI endpoint.

2. Budget & Cost Ceiling

Question	Decision
Monthly spend limit	$0-25/month during development; free tier for demo
Pay-per-use acceptable?	Yes - Firebase Blaze pay-as-you-go is ideal for low-volume demo
Trade money for time?	Yes - use managed services (Firebase) over self-hosted to hit 24hr MVP

Rationale: 7-day sprint. Developer time is the scarcest resource. Firebase free tier covers all MVP needs (50K reads/day, 20K writes/day, 10K auth/month). AI API cost ~$5 during dev using Claude Sonnet 4.5 ($3/$15 per 1M tokens). Cloud Functions v2 with minInstances=1 adds ~$3-5/month.

3. Time to Ship

Question	Decision
MVP timeline	24 hours (hard gate)
Priority	Speed-to-market first, maintainability second
Iteration cadence	Daily - MVP (Tue) → Full (Fri) → Polish (Sun)

Rationale: The 24-hour MVP gate forces the fastest path to working multiplayer. Firebase eliminates backend boilerplate (auth, database, hosting, real-time sync) so focus stays on canvas interactions and multiplayer UX.

4. Compliance & Regulatory Needs

Question	Decision
HIPAA	No
GDPR	No - US-only demo

Question	Decision
SOC 2	No
Data residency	No requirements

Rationale: Sprint project for evaluation, not a production SaaS product. No compliance constraints apply.

5. Team & Skill Constraints

Question	Decision
Solo or team?	Solo developer with AI-first workflow
Known frameworks	React, TypeScript, Firebase, Node.js
Learning appetite	Low - shipping speed over learning new paradigms
AI tools	Claude Code + Cursor (satisfies 'at least 2' requirement)

Rationale: Solo developer means every technology choice must minimize debugging surface area. React + TypeScript + Firebase is well-documented and AI coding tools are highly effective with it.

Phase 2: Architecture Discovery

6. Hosting & Deployment

Option	Pros	Cons	Verdict
Firebase Hosting	Integrated with Firebase, free SSL, CDN, easy deploy	Limited to static/SPA	**SELECTED**
Vercel	Great DX, 100GB bandwidth free, edge functions	Separate from backend, adds complexity	Backup option
Render	Can host backend services, 750 free hours	15-min spin-down kills WebSocket connections	Rejected

Decision: Firebase Hosting for the SPA frontend. Single ecosystem simplifies deployment. CI/CD via firebase deploy from CLI. Firebase auto-scales with no configuration.

7. Authentication & Authorization

Option	Pros	Cons	Verdict
Firebase Auth	Built-in, multiple providers, free 50K MAU	Vendor lock-in	**SELECTED**
Supabase Auth	Good, open source	Requires Supabase ecosystem	Rejected
Auth0/Clerk	Feature-rich	Overkill, adds dependency	Rejected
Custom JWT	Full control	Too slow to build for MVP	Rejected

Decision: Firebase Auth with Google sign-in + anonymous auth.

- **Google sign-in:** One-click auth, provides display name for cursor labels
- **Anonymous auth:** Evaluators test without accounts. Auto-generates fun display names (e.g. "Blue Fox", "Swift Otter") for cursor labels and presence.
- **RBAC:** Not needed. All authenticated users have equal board access.
- **Multi-tenancy:** Each board is a separate Firestore collection, keyed by board ID.

8. Database & Data Layer

Option	Pros	Cons	Verdict
Firebase RTDB	Sub-50ms sync, presence/cursors optimized	Less structured than Firestore	**SELECTED for cursors/presence**
Firestore	Rich queries, offline support, scalable	Slightly higher latency than RTDB	**SELECTED for board objects**
Supabase (Postgres)	SQL, real-time subscriptions	Real-time less mature	Rejected
Yjs + custom WebSocket	True CRDT, conflict-free	Significant setup time	Rejected for MVP

Decision: Dual Firebase approach (both from day one)

- **Firestore** for board objects (sticky notes, shapes, frames, connectors) — 100-200ms listener propagation, acceptable for user-initiated actions.
- **Firebase RTDB** for ephemeral data (cursor positions, presence) — 20-50ms latency. Built-in onDisconnect() hook for automatic presence cleanup.

Why both from day one: PRD build priority starts with cursor sync. Building cursors on Firestore first and migrating later risks a mid-sprint rewrite. The two databases handle completely separate concerns (RTDB: useCursors.ts, usePresence.ts; Firestore: useBoard.ts) with only boardId as a shared key. Setup overhead is ~30 minutes for correct architecture from the start.

- **Conflict Resolution:** Last-write-wins via Firestore. Each object is a separate document. Concurrent edits use server timestamp ordering.
- **Read/Write Ratio:** ~60/40 for board objects; ~20/80 for cursors.

9. Backend/API Architecture

Question	Decision
Architecture	Serverless monolith — Firebase Cloud Functions v2 for AI agent, Firestore/RTDB for everything else
API style	No traditional API — Firestore SDK handles CRUD from client. Cloud Function v2 for AI endpoint only.
Background jobs	None needed for MVP. AI commands are synchronous request-response.
Cold start mitigation	Cloud Functions v2 with minInstances: 1 (~$3-5/mo). Eliminates 5-10s cold start.

Cloud Function v2 AI endpoint flow:

1. Receives natural language command from client
2. Calls Claude Sonnet 4.5 API with function-calling tools
3. Writes resulting objects to Firestore via Admin SDK
4. Returns confirmation to client

Why v2 over v1: Runs on Cloud Run infrastructure. Supports minInstances (eliminates cold starts), concurrency up to 1000 requests/instance, and shorter cold starts when they occur (2-4s vs 5-10s).

10. Frontend Framework & Rendering

Framework

Option	Pros	Cons	Verdict
React + Next.js	Huge ecosystem, AI tools know it well	Heavier than needed for SPA	Considered
React (Vite)	Fastest build, lightweight, SPA-focused	No SSR (not needed)	**SELECTED**
Vue	Good DX	Smaller canvas library ecosystem	Rejected
Svelte	Great performance	Fewer canvas integrations	Rejected

Canvas Library

Option	Perf (Chrome)	React Integration	API Level	Verdict
Konva.js	23 FPS (8K boxes)	react-konva (declarative)	High-level	**SELECTED**
PixiJS	60 FPS (8K boxes)	Manual bindings	Low-level	Backup
Fabric.js	9 FPS (8K boxes)	No React bindings	Mid-level	Rejected

Decision: React 18 + Vite + TypeScript + react-konva + Tailwind CSS

> *Konva's React integration and high-level shape API saves significant development time. 23 FPS at 8K boxes means ~500 objects (target) will run at 60 FPS easily. PixiJS is backup if performance issues arise.*

11. Third-Party Integrations

Service	Purpose	Pricing	Verdict
Firebase (full suite)	Auth, Firestore, RTDB, Hosting, Functions	Free tier covers demo; Blaze pay-as-you-go	**SELECTED**
Claude API (Sonnet 4.5)	AI agent function calling	$3/$15 per 1M tokens; ~$5 dev cost	**SELECTED**
Claude Haiku 4.5	Faster/cheaper alternative	$1/$5 per 1M tokens	Backup at scale
OpenAI GPT-4o-mini	AI agent alternative	$0.15/$0.60 per 1M tokens	Backup option

Vendor lock-in risk: Moderate with Firebase. Mitigation: board object schema is simple JSON, migratable to any document DB. Rate limits: Firebase free tier 50K reads/day, 20K writes/day. Neither hit during evaluation.

Phase 3: Post-Stack Refinement

12. Security Vulnerabilities

Risk	Mitigation
Firestore security rules misconfiguration	Write strict rules: users must be authenticated; can only write to boards they access
XSS via sticky note text	Sanitize text input; react-konva renders to canvas (not DOM), so XSS risk is minimal
AI prompt injection	Validate AI tool call outputs before writing to Firestore; constrain tool schemas
Firebase API key exposure	Firebase client keys are safe to expose (security via rules); do NOT expose Claude API key
Cloud Function secrets	Store Claude API key in Firebase Functions config, never in client code

13. File Structure & Project Organization

```
collabboard/
■■■ public/
■■■ src/
■   ■■■ components/
■   ■   ■■■ Board/          # Main canvas + Konva stage
■   ■   ■■■ Toolbar/        # Drawing tools, shape selector
■   ■   ■■■ Cursors/        # Multiplayer cursor overlay
■   ■   ■■■ Presence/       # Online users panel
■   ■   ■■■ AIChat/         # AI command input
■   ■   ■■■ Auth/           # Login/signup
■   ■■■ hooks/
■   ■   ■■■ useBoard.ts     # Board state + Firestore sync
■   ■   ■■■ useCursors.ts   # Cursor sync via RTDB
■   ■   ■■■ usePresence.ts  # Online presence via RTDB
■   ■   ■■■ useAI.ts        # AI command handler
■   ■■■ services/
■   ■   ■■■ firebase.ts     # Firebase config + init
■   ■   ■■■ boardService.ts # Firestore CRUD for board objects
■   ■   ■■■ aiService.ts    # Cloud Function client
■   ■■■ types/
■   ■   ■■■ board.ts        # TypeScript interfaces
■   ■■■ utils/
■   ■   ■■■ colors.ts       # Color palette constants
■   ■■■ App.tsx
■   ■■■ main.tsx
■■■ functions/
■   ■■■ src/
■       ■■■ index.ts        # Cloud Function entry points
■       ■■■ aiAgent.ts      # Claude API + tools
■■■ firestore.rules
■■■ database.rules.json
■■■ firebase.json
■■■ package.json
■■■ tsconfig.json
■■■ vite.config.ts
```

Monorepo: Single repo. functions/ directory for Cloud Functions, root for frontend. Firebase CLI handles both.

14. Naming Conventions & Code Style

Convention	Standard
Language	TypeScript (strict mode)
Components	PascalCase (StickyNote.tsx)
Hooks	camelCase with use prefix (useBoard.ts)
Services	camelCase (boardService.ts)
Types/Interfaces	PascalCase (BoardObject, StickyNote)
CSS	Tailwind CSS (utility-first)
Linter	ESLint with React + TypeScript recommended rules
Formatter	Prettier (default config)

15. Testing Strategy

Level	Tool	Coverage Target	Priority
Unit	Vitest	Board object CRUD logic	Low (MVP focus)
Integration	Vitest + Firebase emulator	Firestore sync, auth flows	Medium
E2E	Playwright	Multi-browser collaboration	High
Manual	Two browser windows	Real-time sync, cursors, presence	**Critical for MVP**

MVP testing: Manual testing with 2+ browser windows is the primary validation. Automated E2E tests are a stretch goal. Mocking via Firebase emulator suite.

16. Recommended Tooling & DX

Tool	Purpose
Claude Code	AI-first development, code generation, debugging
Cursor	IDE with AI integration (satisfies 2-tool requirement)
Firebase Emulator Suite	Local development without hitting production
Firebase CLI	Deploy, manage rules, functions
Chrome DevTools	Performance profiling (FPS, network)
Multiple browser profiles	Test multiplayer locally
Network throttling	Test sync resilience

Final Stack Decision

FRONTEND	React 18 + Vite + TypeScript + react-konva Tailwind CSS | Firebase Hosting (CDN + SSL)
REAL-TIME LAYER	Firestore → Board objects (shapes, notes, frames) Realtime DB → Cursors, presence (ephemeral data)
AUTH & SECURITY	Firebase Auth (Google sign-in + anonymous) Auto-generated display names | Security Rules
AI AGENT	Firebase Cloud Functions v2 (minInstances=1) Anthropic Claude Sonnet 4.5 | Function calling
DEPLOYMENT	firebase deploy (single command)

Why This Stack

1. **Speed to MVP:** Firebase gives auth + database + real-time + hosting in one npm install. No backend to build.
2. **Real-time performance:** Firebase RTDB delivers <50ms cursor sync with onDisconnect() for presence cleanup. Firestore delivers <100ms object sync.
3. **AI integration:** Single Cloud Function v2 endpoint (no cold starts). Claude Sonnet 4.5 provides reliable multi-step function calling. ~$5 total dev cost.
4. **Developer experience:** React + TypeScript + Vite + Tailwind is the most AI-tool-friendly stack. Claude Code and Cursor both excel with this combination.
5. **Zero DevOps:** firebase deploy handles everything. No Docker, Kubernetes, or CI/CD pipeline needed.

Tradeoffs Accepted

Tradeoff	Why It's Acceptable
No CRDT (last-write-wins)	Sufficient for 5-10 users. Object-level granularity means conflicts are rare.
Firebase vendor lock-in	7-day sprint. Migration is not a concern.
Konva.js not fastest canvas lib	500 objects at 60 FPS is achievable. PixiJS adds complexity for minimal gain.
No offline support	Real-time collaboration requires connectivity. Offline is contradictory.
Client-side Firestore writes	Security rules enforce auth. Simpler than building a REST API.
Sonnet over Haiku (higher cost)	Reliability on complex multi-step commands worth 3x cost at dev scale (~$5 vs ~$1.50).
CF v2 min-instance cost	~$5/mo for guaranteed <2s AI response. Eliminates cold start risk during evaluation.
Dual database (Firestore + RTDB)	Clean separation. RTDB gives onDisconnect() for presence and <50ms cursor sync.

Production Cost Projections

Assumptions

- Average AI commands per user per session: 5
- Average sessions per user per month: 8
- Average tokens per AI command: ~1,500 input + ~500 output
- Board operations per session: ~200 reads + ~50 writes (Firestore)
- Cursor updates per session: ~5,000 writes (RTDB, throttled to 10/sec over ~8 min avg session)

AI Costs (Claude Sonnet 4.5: $3/$15 per 1M tokens)

Scale	AI Commands/Month	Input Tokens	Output Tokens	AI Cost
100 users	4,000	6M	2M	~$48/mo
1,000 users	40,000	60M	20M	~$480/mo
10,000 users	400,000	600M	200M	~$4,800/mo
100,000 users	4,000,000	6B	2B	~$48,000/mo

Firebase Costs (Blaze pay-as-you-go, includes CF v2 min-instance)

Scale	Firestore Reads	Firestore Writes	RTDB BW	CF v2	Firebase Cost
100 users	160K/mo	40K/mo	~2 GB	$5/mo	~$5/mo
1,000 users	1.6M/mo	400K/mo	~20 GB	$5/mo	~$10/mo
10,000 users	16M/mo	4M/mo	~200 GB	$5/mo	~$55/mo
100,000 users	160M/mo	40M/mo	~2 TB	$5/mo	~$505/mo

Total Estimated Monthly Cost

100 Users	1,000 Users	10,000 Users	100,000 Users
~$53/month	**~$490/month**	**~$4,855/month**	**~$48,505/month**

Cost optimization at scale: *AI API costs dominate at all scales. To reduce costs: (1) cache common templates (SWOT, retro boards) as static JSON — eliminates AI calls for popular commands; (2) tiered model routing — use Haiku ($1/$5 per 1M tokens) for simple single-step commands, Sonnet only for complex multi-step operations; (3) prompt optimization — minimize input tokens by sending only relevant board state via getBoardState() filtering.*