

CollabBoard: Pre-Search Document

Architecture Discovery & Stack Decision Record

Developer	Alan (solo)	Date	February 16, 2026
Stack	Next.js + Supabase + Konva.js	AI Tools	Claude Code + MCP integrations
Deploy	Vercel	AI Agent	Anthropic Claude (function calling)

Phase 1: Define Your Constraints

1. Scale & Load Profile

Users at launch: 5-10 concurrent (Gauntlet evaluators). In 6 months: portfolio project, so 50-100 concurrent max if it gains traction.

Traffic pattern: Spiky. Evaluation periods will have concentrated usage; otherwise near-zero. This rules out always-on infrastructure costs and favors serverless/managed services.

Real-time requirements: Yes, critical. WebSocket connections for live cursor streaming (<50ms), object sync (<100ms), and presence awareness. Supabase Realtime (built on Phoenix Channels/WebSockets) handles this natively with Postgres changes broadcast.

Cold start tolerance: Moderate. The board itself must feel instant once loaded, but an initial page load of 1-2 seconds is acceptable. Vercel's edge network helps here. The AI agent can tolerate 1-2s response times per the spec.

Decision: Supabase Realtime for sync, targeting 5-10 concurrent users with headroom to 50+.

Rationale: Supabase Realtime is a managed WebSocket layer over Postgres. It eliminates the need to build custom WS infrastructure for an MVP while still giving us full Postgres for persistence and querying.

AI-first note: Supabase has official MCP integrations, meaning Claude Code can directly query, modify, and inspect the database schema during development. This dramatically accelerates iteration speed by letting the AI agent understand and work with the live data layer without manual context-passing.

2. Budget & Cost Ceiling

Monthly spend limit: \$450 available on a Ramp card, but cost-consciousness is a priority. Infrastructure should stay on free tiers. AI API costs (Anthropic, OpenAI) are the primary variable expense, budgeted at \$10-30 for development and testing. We have room to spend tokens when needed, but should avoid waste.

Pay-per-use vs fixed: Pay-per-use is ideal given spiky traffic. Supabase Free tier gives 500MB DB, 50K monthly active users, and Realtime connections. Vercel Free tier gives 100GB bandwidth. Both sufficient.

Trade money for time: Everywhere possible. Using managed services (Supabase Auth, Supabase Realtime, Vercel) instead of self-hosting means less DevOps and more time on core features. AI coding tools (Claude Code) and LLM API calls are worthwhile investments in velocity — we have budget for them.

Decision: Free-tier infra. ~\$10-30 for AI API usage during dev. \$450 available but spend conservatively.

3. Time to Ship

MVP timeline: 24 hours (hard gate). This is the single most important constraint. Every stack decision must be evaluated through the lens of 'can I ship multiplayer in 24 hours?'

Speed vs maintainability: Speed wins decisively for MVP. Maintainability matters for the full submission (7 days) but only in the sense of 'can I extend this without rewriting.' Choosing well-structured tools (Next.js, Supabase) gives both.

Iteration cadence: Continuous over the week. MVP Tuesday, full features Friday, polish Sunday.

Decision: Optimize every choice for shipping speed. No custom infrastructure that could be managed.

4. Compliance & Regulatory Needs

No formal compliance frameworks (HIPAA, SOC 2) are required for this portfolio project. However, several considerations are worth documenting to show awareness of what a production deployment would require:

User data and authentication: The app collects email addresses and OAuth tokens via Supabase Auth. Even for a portfolio project, these are real user credentials. Supabase handles token storage and session management server-side, and we should not store sensitive auth data in client-side storage beyond session tokens.

GDPR considerations: If the deployed app is publicly accessible (which it will be for evaluation), EU users could theoretically access it. At minimum this means: a clear indication of what data is collected, the ability for users to delete their data (board deletion), and no unnecessary data retention. For this project, we implement board deletion and keep no analytics or tracking beyond Supabase's defaults.

Data retention: Board data persists indefinitely in the current design. A production system would need retention policies and auto-deletion of inactive boards. For the sprint, manual cleanup is acceptable.

Logging and analytics: No third-party analytics (no Google Analytics, no Mixpanel). Server-side logs are limited to Vercel's built-in function logs and Supabase's query logs, both of which are provider-managed and not exported.

Decision: No formal compliance needed. Standard security practices, minimal data collection, no third-party tracking.

Awareness: In a production context, GDPR compliance (consent, data deletion, privacy policy) and SOC 2 (for enterprise clients) would be the first two frameworks to address.

5. Team & Skill Constraints

Solo developer. Strong proficiency in TypeScript, React, Next.js, and PostgreSQL. Comfortable with Go and AWS but choosing the JS/TS ecosystem for speed and consistency across the full stack. Experience with Supabase is moderate but Postgres knowledge is deep.

Canvas/graphics: No prior experience with HTML5 Canvas libraries (Konva.js, Fabric.js, PixiJS). This is the biggest learning curve risk. Mitigated by choosing react-konva which integrates with React's component model (familiar territory) and has strong AI training data for code generation.

Learning vs shipping: Shipping. Will lean on Claude Code heavily for canvas rendering code and multi-file refactors, with MCP integrations (e.g., Supabase MCP, filesystem MCP) for direct tool access. ChatGPT used as a secondary discussion partner for architecture questions. Personal effort focused on the real-time

sync architecture (higher complexity, harder to generate).

Decision: TypeScript/React throughout. Claude Code + MCPs for AI-assisted dev. Focus manual effort on sync.

Phase 2: Architecture Discovery

6. Hosting & Deployment

Approach: Serverless (Vercel) + Managed Backend (Supabase)

Option	Pros	Cons	Fit
Vercel + Supabase	Zero-config Next.js deploys, edge CDN, automatic previews, native Supabase integration	Serverless function limits (10s default timeout), vendor coupling	Best
Firebase Hosting	Tight integration with Firestore Realtime, generous free tier	NoSQL only (no relational queries), proprietary query language, weaker for complex data	Good
Render / VPS	Full control, persistent WebSocket servers, no cold starts	Must manage infra, slower deploys, costs at any scale	Overkill

CI/CD: Vercel's GitHub integration provides automatic deployments on push. Preview deployments on PRs. No additional CI/CD setup needed.

Scaling: Vercel scales automatically (serverless). Supabase Realtime handles connection pooling. For the 5+ concurrent user target, both free tiers are more than sufficient.

Decision: [Vercel for frontend/API routes + Supabase for DB/Auth/Realtime. Zero DevOps overhead.](#)

AI-first note: Both Vercel and Next.js have extensive representation in LLM training data, meaning Claude Code and other AI agents generate highly accurate code for this stack. The well-documented, convention-heavy nature of Next.js App Router reduces the need for manual correction of AI-generated code.

7. Authentication & Authorization

Approach: Supabase Auth with social login (Google/GitHub) + email/password fallback.

Supabase Auth is built-in, requires no additional service, and integrates directly with Row Level Security (RLS) policies on the database. This means auth and authorization are handled at the data layer, not the application layer.

RBAC: Minimal. For MVP, all authenticated users on a board have equal permissions. Post-MVP could add board owner/editor/viewer roles using Supabase RLS policies.

Multi-tenancy: Each board is a separate namespace. Users can create/join boards via shareable links. Board isolation is enforced by foreign key relationships and RLS.

Decision: [Supabase Auth with Google + GitHub OAuth. RLS for board-level isolation.](#)

8. Database & Data Layer

Approach: PostgreSQL (Supabase) with Realtime subscriptions.

The data model is relational: boards have objects, objects have properties, users have presence state. Postgres handles this naturally with proper indexing. Supabase Realtime broadcasts row-level changes to subscribed clients via WebSockets.

Core schema concept:

boards (id, name, created_by, created_at) → board_objects (id, board_id, type, x, y, width, height, properties JSONB, z_index, updated_at, updated_by) → presence (user_id, board_id, cursor_x, cursor_y, last_seen)

Why JSONB for properties: Different object types (sticky notes, shapes, connectors, frames) have different properties. A JSONB column avoids an explosion of nullable columns or complex inheritance tables. Postgres JSONB is indexable and queryable.

Real-time sync: Supabase Realtime listens to Postgres changes (INSERT/UPDATE/DELETE) and broadcasts them. Cursor positions use Supabase's Broadcast channel (ephemeral, not persisted) for <50ms latency.

Read/write ratio: Write-heavy during active collaboration (every drag, every keystroke). Debouncing/throttling on the client (e.g., batch position updates every 50ms) is essential to avoid overwhelming the database.

Decision: Postgres with JSONB properties column. Supabase Realtime for object sync, Broadcast for cursors.

Rationale: Separating persistent object sync (DB-backed Realtime) from ephemeral cursor sync (Broadcast) avoids writing thousands of cursor positions per second to the database.

9. Backend / API Architecture

Approach: Next.js API Routes (monolith) + Supabase client SDK.

No separate backend server. Next.js API routes handle any server-side logic (AI agent orchestration, board management). The Supabase JS client handles DB queries and Realtime subscriptions directly from the browser for read/write operations, with RLS providing security.

AI agent endpoint: A single Next.js API route (/api/ai-command) receives natural language commands, calls Anthropic's API with tool definitions, and executes the resulting tool calls against Supabase. Results propagate to all clients via Realtime subscriptions automatically.

Why not microservices: Solo developer, one-week timeline. A monolith in Next.js is the fastest path. The AI agent is just another API route, not a separate service.

Decision: Next.js API routes monolith. Direct Supabase client for real-time operations.

AI-first note: A monolith means the AI coding agent has full context of the entire codebase in one project. No cross-service boundaries, no separate deployment configs. Claude Code can reason about the full request path from frontend component to API route to database query in a single context window.

10. Frontend Framework & Rendering

Approach: Next.js (React) + react-konva for canvas rendering.

Library	Pros	Cons	Fit
react-konva	React component model, declarative API, good docs, built-in drag/transform, solid community	Performance ceiling lower than raw PixiJS for 1000+ objects	Best

Fabric.js	Rich built-in shapes, serialization, mature	Imperative API, no React integration, heavier bundle	OK
PixiJS	WebGL-accelerated, highest performance ceiling	Game-engine complexity, overkill for whiteboard, steep learning curve	Overkill
Raw Canvas	No dependencies, full control	Must build everything from scratch (hit detection, transforms, etc.)	Too slow

SSR/SEO: Not needed. The whiteboard is a fully client-side interactive app. Next.js is used for routing, API routes, and deployment convenience, not for SSR. The canvas page will be a client component.

Offline/PWA: Not in scope. Real-time collaboration is the core feature; offline mode is contradictory.

Decision: react-konva. React-native API, built-in transforms, sufficient perf for 500+ objects.

AI-first note: react-konva's declarative, component-based API maps well to how AI coding agents reason about UI code. Generating a Konva <Rect> or <Text> component is structurally identical to generating a React <div>, which means AI tools produce higher-quality canvas code with fewer corrections needed.

11. Third-Party Integrations

Service	Provider	Purpose	Cost
AI Agent LLM	Anthropic Claude	Natural language board commands with function calling	~\$3/1M input tokens
Database + Realtime	Supabase	Postgres, Auth, Realtime subscriptions, Broadcast	Free tier
Hosting	Vercel	Next.js hosting, edge CDN, serverless functions	Free tier
Canvas	react-konva	2D canvas rendering with React integration	OSS / free

Vendor lock-in risk: Moderate with Supabase (Postgres is portable, but Realtime/Auth are proprietary). Low with Vercel (Next.js is framework-agnostic). Acceptable for a one-week sprint.

Rate limits: Anthropic API has rate limits based on tier. For dev/testing with a handful of users, well within limits. Supabase free tier allows 200 concurrent Realtime connections.

Decision: Anthropic Claude for AI, Supabase for backend, Vercel for hosting. All free-tier compatible.

Phase 3: Post-Stack Refinement

12. Security Vulnerabilities

Known pitfalls for this stack:

Supabase RLS misconfiguration is the #1 risk. If RLS policies are not set correctly, any authenticated user could read/modify any board. Every table must have RLS enabled with explicit policies. The anon key is exposed client-side by design, so RLS is the security boundary, not the key.

AI prompt injection: Users could craft board commands attempting to manipulate the AI agent beyond its intended tool schema. Mitigation: the AI agent only has access to predefined tools (createStickyNote, moveObject, etc.) with validated parameters. No raw SQL or arbitrary code execution.

XSS via sticky note text: User-entered text rendered on the canvas must be sanitized. Konva's Text nodes render to canvas (not DOM), which provides natural XSS protection. However, any HTML rendering of board data (e.g., in sidebars) must be sanitized.

Dependency risks: react-konva and konva are well-maintained. Pin versions in package.json and run npm audit before submission.

Realtime abuse / event flooding: This is the highest-impact runtime risk. The app is write-heavy and WebSocket-heavy by design — every cursor movement and object drag generates events. A malicious or buggy client could flood the Realtime channel and degrade the experience for all users on a board. Mitigations: client-side debouncing/throttling of position updates (50ms minimum interval), server-side rate limiting on the AI command API route (e.g., max 10 commands per minute per user), and Supabase's built-in connection limits. For the MVP, client-side throttling is the primary defense; server-side rate limiting is added during the polish phase.

Decision: RLS on every table, tool-schema-only AI execution, input sanitization, pinned deps.

13. File Structure & Project Organization

Monorepo: Single Next.js project.

```
collabboard/  
  app/ — Next.js App Router pages and API routes  
    (auth)/ — login, callback routes  
    board/[id]/ — main board page  
    api/ai-command/ — AI agent endpoint  
  components/ — React components  
    canvas/ — Konva canvas, objects, toolbar  
    ui/ — shared UI (buttons, modals, etc.)  
    collaboration/ — cursors, presence, AI chat  
  lib/ — utilities and clients  
    supabase/ — client, types, queries  
    ai/ — tool definitions, prompt templates  
    hooks/ — custom React hooks (useBoard, usePresence, etc.)  
  types/ — TypeScript types and interfaces  
  supabase/ — migrations and seed data
```

Decision: Feature-grouped folders under a single Next.js App Router project.

14. Naming Conventions & Code Style

TypeScript/React conventions: PascalCase for components and types, camelCase for functions and variables, SCREAMING_SNAKE_CASE for constants. Files: kebab-case for utilities, PascalCase for components.

Linter/formatter: ESLint with Next.js config + Prettier. Configured in the project root. Strict TypeScript (strict: true in tsconfig). No 'any' types without explicit justification.

Database: snake_case for all table and column names (Postgres convention). Supabase generates TypeScript types from the schema, ensuring type safety end-to-end.

Decision: Standard TS/React conventions, ESLint + Prettier, strict TypeScript, snake_case DB.

15. Testing Strategy

Principle: Test every graded requirement before submission. The spec lists explicit testing scenarios and performance targets. Our testing strategy maps directly to these to ensure a confident submission. That said, this is a one-week sprint — testing effort must be time-boxed and prioritized. The priority order below reflects what matters most for passing evaluation.

Priority 1 — MVP Checklist (gate, test first):

Each MVP requirement gets a pass/fail check in multiple browsers: infinite board pan/zoom, sticky note creation and editing, shape creation, object CRUD, real-time sync between 2+ users, multiplayer cursors with name labels, presence awareness, authentication flow, and deployed accessibility. Maintain a checklist doc and check off each item with screenshots.

Priority 2 — Spec Testing Scenarios (evaluators will run these):

The spec defines 5 specific test scenarios that evaluators will run. We test all of them before submission: (1) 2 users editing simultaneously in different browsers, (2) one user refreshing mid-edit to verify state persistence, (3) rapid sticky note and shape creation/movement for sync performance, (4) network throttling and disconnection recovery via Chrome DevTools, (5) 5+ concurrent users without degradation (open 5+ browser tabs/windows).

Priority 3 — Performance Validation (time permitting, measure what we can):

Verify against the spec's performance targets: 60 FPS during pan/zoom (Chrome DevTools Performance tab), <100ms object sync latency (timestamp logging), <50ms cursor sync latency, 500+ objects without performance drops (scripted bulk creation), 5+ concurrent users. Document results with measurements.

Priority 4 — AI Agent Testing:

Test all required command categories: creation ('Add a yellow sticky note'), manipulation ('Move all pink sticky notes'), layout ('Arrange in a grid'), and complex ('Create a SWOT analysis'). Verify 6+ distinct command types work, <2s response latency, and that all users see AI-generated results in real-time. Test simultaneous AI commands from multiple users.

Priority 5 — Unit Tests (Vitest, targeted only):

Targeted unit tests for high-risk logic: AI tool parameter validation and schema enforcement, object serialization/deserialization, conflict resolution logic, and board state queries. These catch regressions during the rapid iteration phase.

Priority 6 — E2E Tests (Playwright, only if time remains after polish):

If time permits after feature completion and polish, automate the 5 spec testing scenarios in Playwright for repeatable regression testing. Realistically, manual testing will carry most of the weight in a one-week sprint.

Decision: Prioritized manual testing mapped to spec requirements. Automated tests where time allows. Ship confidence over coverage.

16. Recommended Tooling & Developer Experience

AI coding tools (required): Claude Code as the primary AI agent for complex architectural decisions, multi-file refactors, and code generation. MCP integrations within Claude Code (Supabase MCP for direct DB interaction, filesystem MCP for project navigation) satisfy the second required tool. ChatGPT used as a secondary discussion partner for brainstorming and architecture review.

VS Code extensions: ESLint, Prettier, Tailwind CSS IntelliSense, Supabase (for schema browsing), Error Lens (inline error display).

CLI tools: supabase CLI for local development and migrations, vercel CLI for deployment management, npx supabase gen types for generating TypeScript types from the database schema.

Debugging: React DevTools for component state inspection, Chrome DevTools Network tab for WebSocket frame inspection (critical for debugging Realtime issues), Supabase Dashboard for real-time database monitoring.

Decision: Claude Code + MCP integrations for AI-assisted dev. Supabase CLI for local dev. Standard VS Code setup.

Architecture Summary

Layer	Choice	Key Reason
Frontend	Next.js 14 + React + TypeScript	Strongest skill, Vercel-native, App Router for API routes
Canvas	react-konva (Konva.js)	Declarative React API, built-in drag/resize/transform
Database	PostgreSQL (Supabase)	Deep PG expertise, relational model fits, free tier
Real-time	Supabase Realtime + Broadcast	Managed WS, DB-change streaming, ephemeral broadcast for cursors
Auth	Supabase Auth (OAuth)	Built-in, RLS integration, zero additional setup
AI Agent	Anthropic Claude (function calling)	Strong function calling, familiar API, good for structured tool use
Deployment	Vercel	Zero-config Next.js deploys, edge CDN, free tier
Dev Tools	Claude Code + MCP integrations	Primary agent for code gen + MCPs for Supabase/filesystem access

How This Document Was Generated

This Pre-Search document was produced using an AI-first methodology, consistent with the assignment's emphasis on AI-assisted development workflows.

Process: The CollabBoard assignment PDF was uploaded to Claude (Anthropic), which read the full spec and extracted the 16-item Pre-Search checklist. Claude then asked a series of targeted questions to establish the key decision points where personal preference and context mattered (backend choice, deployment platform, AI coding tools, budget). Based on those answers and knowledge of the developer's existing skill set (TypeScript, React, Next.js, PostgreSQL), Claude generated a comprehensive first draft covering all 16 sections with decisions, tradeoff analysis, and comparison tables.

Review cycle: The draft was then shared with ChatGPT (OpenAI) for an independent review pass. ChatGPT identified five areas for improvement: (1) adding AI-first justifications to architecture decisions, (2) expanding the compliance section to show awareness beyond 'not applicable,' (3) adding realtime abuse/event flooding as a security consideration, (4) making the testing strategy more pragmatic and prioritized, and (5) adding this section documenting the generation process. These revisions were implemented back in Claude and the PDF was regenerated.

Tools used: Claude (claude.ai, Opus) for primary document generation and PDF creation. ChatGPT (OpenAI) for independent review and critique. The full AI conversation is available as a reference document.