CollabBoard

Pre-Search Document

Building Real-Time Collaborative Whiteboard Tools with AI-First Development

Stack: Next.js + tldraw + Liveblocks + NextAuth.js + GPT-4.1-mini + Vercel

February 2026

Solo Developer | T3 Stack | 1-Week Sprint

Table of Contents

Phase 1: Define Your Constraints

Phase 2: Architecture Discovery

Phase 3: Post-Stack Refinement

Final Stack Summary & Build Plan

Phase 1: Define Your Constraints

1. Scale & Load Profile

Users at launch: 5-20 concurrent users per board (demo and evaluation context for the Gauntlet sprint).

Users in 6 months: 50-100 concurrent users if the project grows beyond the sprint into a portfolio piece or production app.

Traffic pattern: Spiky. Collaborative whiteboards see intense bursts during workshops, brainstorms, and retros, then long idle periods. The system must handle sudden room joins (5+ users joining within seconds) without degradation.

Real-time requirements: Critical and non-negotiable. WebSocket connections are mandatory for cursor sync (<50ms latency), object sync (<100ms latency), and presence awareness. HTTP polling is not acceptable.

Cold start tolerance: Very low. Users joining a board expect instant connectivity. Cold starts exceeding 2-3 seconds would visibly break the real-time experience.

2. Budget & Cost Ceiling

Monthly spend limit: Flexible. Prioritize free tiers where possible for MVP. Willing to invest up to $50/month for production deployment.

Pay-per-use vs fixed costs: Pay-per-use preferred. Spiky traffic pattern means fixed infrastructure would be wasteful during idle periods. Serverless and usage-based pricing models align well with whiteboard usage patterns.

Cost optimization strategy: Trade developer time for simpler infrastructure. Managed services (Liveblocks, Vercel) save implementation time. For AI, GPT-4.1-mini was chosen specifically to minimize per-command costs while maintaining 100% tool call reliability.

3. Time to Ship

MVP deadline: Tuesday (24 hours). Must deliver: infinite board with pan/zoom, sticky notes, at least one shape type, real-time sync between 2+ users, multiplayer cursors, presence awareness, user authentication, and public deployment.

Early submission: Friday (4 days). Full feature set including all board features, AI agent with 6+ command types, connectors, frames, transforms, and polished UX.

Final deadline: Sunday 10:59 PM CT. Polish, documentation, demo video, deployment verification.

Priority stance: Speed-to-market dominates. This is a 1-week sprint, not a 6-month product build. Choose tools that minimize implementation time, even at the cost of long-term flexibility.

4. Compliance & Regulatory Needs

Not applicable. No health data (HIPAA), no EU-specific user base (GDPR), no enterprise compliance (SOC 2), and no data residency constraints. This is a technical evaluation project for the Gauntlet program.

5. Team & Skill Constraints

Team: Solo developer building the entire stack.

Strong skills: Next.js, TypeScript, tRPC, Prisma, React ecosystem (T3 stack).

Learning appetite: Moderate. Willing to learn tldraw SDK and Liveblocks hooks, but not a new language/paradigm within a 1-week sprint.

Implication: Strongly favor React-native tools with first-class TypeScript support. Avoid platforms requiring deep infrastructure knowledge. Managed services with good documentation are essential.

Phase 2: Architecture Discovery

6. Hosting & Deployment

DECISION: Vercel (free tier)

Since Liveblocks handles the entire WebSocket layer externally, the Next.js app does not need persistent WebSocket connections. This makes Vercel the ideal deployment target: zero-config Next.js deployment, preview deployments, edge functions, and a generous free tier.

Options Evaluated

Platform	WebSocket Support	Cold Starts	Free Tier	Verdict
Vercel (chosen)	N/A (Liveblocks handles)	N/A	Generous	Best fit for Liveblocks + Next.js
Railway	Native	None	$5/mo min	Excellent if self-hosting WebSockets
Cloudflare Workers	Durable Objects	None (edge)	Free	Best free alternative for Yjs
Render	Native	30-60s (free tier)	Limited	Cold starts disqualify for real-time
Fly.io	Native	None	No free tier	Good but CLI-only, steeper curve
AWS	API Gateway	Possible	$200 credits	Overkill for solo sprint

Fallback: If Vercel limitations arise (e.g., API route timeouts for AI calls), Railway ($5/mo) is a drop-in alternative.

7. Authentication & Authorization

DECISION: NextAuth.js (Auth.js) with Prisma adapter

NextAuth.js is the natural choice for a T3 stack project. It integrates natively with Next.js App Router, supports Prisma as a database adapter, and provides a wide range of OAuth providers. It is open source, self-hosted, and avoids vendor lock-in.

Options Evaluated: NextAuth.js vs Clerk

Factor	NextAuth.js (chosen)	Clerk
T3 stack integration	Native. First-class support.	Requires replacing T3 auth layer
Cost	Free (open source)	Free to 10K MAU, then paid

Factor	NextAuth.js (chosen)	Clerk
Vendor lock-in	None. Self-hosted.	High. Proprietary auth layer.
Setup time	1-2 hours with Prisma adapter	30 minutes (faster)
Provider support	Google, GitHub, Discord, 50+	Google, GitHub, email, SSO
Customization	Full control over UI and flow	Pre-built components (less flexible)

Auth providers: Google and GitHub OAuth for MVP. **Session strategy:** JWT-based for simplicity.

8. Database & Data Layer

DECISION: Supabase PostgreSQL (free tier) via Prisma ORM

With Liveblocks handling all real-time board state (objects, positions, colors, text), the database has a narrow scope: user profiles, board metadata (name, owner, created date), and NextAuth.js session/account records.

- **Database type:** Relational (PostgreSQL). Fits Prisma and T3 stack.
- **Real-time sync:** Handled entirely by Liveblocks. Database is not involved.
- **Persistence model:** Liveblocks Storage persists board state. DB persists user/board metadata only.
- **Read/write ratio:** Read-heavy for board listing. All write-heavy ops flow through Liveblocks.
- **Alternative:** Neon (serverless Postgres) or PlanetScale (serverless MySQL) are equivalent.

9. Backend/API Architecture

DECISION: Next.js API Routes + tRPC (monolith)

Architecture: Monolith. All API logic in Next.js API routes and tRPC routers. No microservices needed.

API style: tRPC for type-safe client-server communication. REST endpoint for AI agent commands.

Key routes: tRPC board CRUD, /api/liveblocks-auth (Liveblocks auth endpoint), /api/ai/command (GPT-4.1-mini processing).

Background jobs: None. AI commands are synchronous request-response.

10. Frontend Framework & Canvas Rendering

DECISION: Next.js 14+ (App Router) + tldraw SDK

The frontend splits into two modes: standard Next.js pages for landing/auth/dashboard (SSR/SSG), and a fully client-side tldraw canvas for the board itself.

Options Evaluated: tldraw vs react-konva

Factor	tldraw (chosen)	react-konva
Time to working canvas	Minutes (drop-in component)	8-12 hours (build from scratch)
Infinite pan/zoom	Built-in with animations	Manual implementation needed
Text editing	Native in-canvas editing	HTML overlay pattern (fiddly)
Selection system	Multi-select, shift-click, lasso	Basic Transformer only
Undo/redo	Built-in	Must implement from scratch
Keyboard shortcuts	Full set included	Must implement from scratch
AI integration	Editor API: editor.createShapes()	Direct React state manipulation
Performance (500+ obj)	Excellent (viewport culling)	Excellent (layer-based)
License	Custom (OK for non-commercial)	MIT (fully open)
Customization	Override components/tools	Unlimited (raw canvas)

Other Canvas Libraries Considered

Library	500+ Obj 60FPS	Pan/Zoom	Text Edit	React	Why Not Chosen
Fabric.js	Good	Poor pan perf	Excellent	Adequate	Pan/zoom performance issues
PixiJS	Excellent	Manual	Poor	Good	No text editing, game-focused
Excalidraw	Good	Good	Good	Good	More opinionated, less customizable

Key advantage: tldraw saves an estimated 15-20 hours of interaction plumbing (selection, transforms, text editing, keyboard shortcuts, undo/redo). For a solo dev in a 1-week sprint, this is the difference between shipping a polished product and a bare-bones prototype.

Liveblocks integration: Official tldraw + Liveblocks integration available. Board state stored as Liveblocks Storage, synced via CRDT. Cursors and presence handled by Liveblocks presence API.

11. Real-Time Collaboration Layer

DECISION: Liveblocks (free tier, 100 MAU)

This is the most critical architectural decision. The spec emphasizes that **bulletproof multiplayer sync matters more than feature richness**. Liveblocks was chosen because it eliminates the hardest engineering problem and provides presence, cursors, and CRDT storage out of the box.

What Liveblocks Provides vs Building It Yourself

Feature	With Liveblocks	Without (Custom Socket.io)
Multiplayer cursors	Built-in useOthers() hook	Build from scratch (4-6 hrs)
Presence awareness	Built-in useSelf() + useOthers()	Build from scratch (2-3 hrs)
Object sync	LiveObject/LiveList CRDT	Build conflict resolution (8-16 hrs)
Conflict resolution	Automatic (CRDT)	Manual last-write-wins or OT (8+ hrs)
Reconnection handling	Automatic with state recovery	Build retry + resync (3-4 hrs)
State persistence	Built-in (survives disconnects)	DB writes + recovery (4-6 hrs)
Total implementation	2-3 hours	29-47 hours

All Options Evaluated

Criteria	Liveblocks	Yjs + PartyKit	Supabase RT	Socket.io
Time to implement	1-2 hours	4-8 hours	3-6 hours	8-16 hours
Cursor sync <50ms	Yes (configurable)	Yes	Yes (Broadcast)	Yes
Object sync <100ms	Yes	Yes	Rate limit risk	Yes
Conflict resolution	Built-in CRDT	Built-in CRDT	Manual	Manual
5+ concurrent users	Yes	Yes	Conditional	Yes
Free tier	100 MAU	Free (Cloudflare)	$0 (rate-limited)	Open source
Vendor lock-in	High	Low	Medium	None
Solo dev feasibility	Excellent	Good	Good	Risky

Why Not the Alternatives

- **Yjs + PartyKit:** Excellent open-source CRDT but requires building cursor/presence UI from scratch. Adds 4-8 hours. Best fallback if vendor lock-in becomes a concern post-sprint.
- **Supabase Realtime:** Rate limits (20-50 events/sec) make it unsuitable for high-frequency cursor updates across multiple users. Not designed for this use case.

- **Custom Socket.io:** Maximum control but maximum risk. Building conflict resolution, reconnection, and presence from scratch is too risky for a 1-week solo sprint.

12. AI Board Agent

DECISION: OpenAI GPT-4.1-mini with structured outputs (strict mode)

The AI agent processes natural language commands and translates them into tldraw Editor API calls. The commands are structured and well-defined (create sticky note, move object, change color), making this an ideal use case for a smaller, cheaper model with reliable tool call execution.

Options Evaluated: Model Comparison

Factor	GPT-4.1-mini (chosen)	Claude Sonnet 4.5	GPT-4.1
Tool call reliability	100% (strict mode)	88%+ (strict mode)	100% (strict mode)
Latency (TTFT)	< 1 second	< 1 second	1-2 seconds
Cost per command*	$0.00021	$0.0045	$0.0028
100 users/month	$0.42	$9.00	$5.60
1,000 users/month	$4.20	$90.00	$56.00
10,000 users/month	$42	$900	$560
Multi-step planning	Sequential calls	Programmatic Tool Calling	Sequential calls
Spatial reasoning	Adequate	Good	Good

** Assumes 500 input + 200 output tokens per command, 10 commands/user/session, 2 sessions/user/month.*

Why GPT-4.1-mini wins for this project: The board commands are intent parsing + structured output problems, not complex reasoning tasks. GPT-4.1-mini achieves 100% JSON schema compliance in strict mode (zero parsing errors) at 20x lower cost than Claude Sonnet. The cost savings are substantial at every scale.

Complex commands strategy: For multi-step commands like "create a SWOT analysis template," the LLM determines intent and application code computes exact coordinates programmatically. This keeps LLM calls simple and reliable.

Tool Schema (11 Tools, Covering All Required Categories)

- **createStickyNote**(text, x, y, color) — Creation
- **createShape**(type, x, y, width, height, color) — Creation
- **createFrame**(title, x, y, width, height) — Creation
- **createConnector**(fromId, toId, style) — Creation
- **moveObject**(objectId, x, y) — Manipulation
- **resizeObject**(objectId, width, height) — Manipulation
- **updateText**(objectId, newText) — Manipulation
- **changeColor**(objectId, color) — Manipulation

- **arrangeInGrid**(objectIds, columns, spacing) — Layout
- **createTemplate**(type) — Complex (SWOT, retro, journey map)
- **getBoardState**() — Context (returns current objects for AI awareness)

13. Third-Party Integrations

Service	Provider	Purpose	Free Tier	Lock-in Risk
Real-time sync	Liveblocks	Multiplayer CRDT + presence	100 MAU	High (proprietary)
Canvas SDK	tldraw	Whiteboard rendering	Free (custom license)	Medium
Authentication	NextAuth.js	OAuth + sessions	Free (open source)	None
Database	Supabase	PostgreSQL for metadata	500MB, unlimited reads	Low
AI agent	OpenAI	GPT-4.1-mini tools	Pay-per-use	Low (swappable)
Hosting	Vercel	Next.js deployment	Generous free tier	Low

Overall vendor lock-in: Medium. Highest risk is Liveblocks (proprietary storage API), but migration to Yjs is well-documented. All other services are open source, standard protocols, or easily replaceable.

Phase 3: Post-Stack Refinement

14. Security Vulnerabilities

Known risks and mitigations for the chosen stack:

- **Liveblocks room access:** Use server-side auth tokens. Never expose room secrets on client. Validate board ownership in the Liveblocks auth endpoint.
- **NextAuth.js sessions:** Use CSRF protection (built-in). Secure cookie flags in production. Validate tokens server-side.
- **OpenAI API key:** Store in env variables. Route all AI calls through server-side API routes. Never import OpenAI SDK client-side.
- **XSS via sticky notes:** Sanitize user-generated text. tldraw handles basic sanitization; add DOMPurify for custom rendering.
- **Rate limiting:** Add rate limiting to /api/ai/command (20 commands/min/user) via upstash/ratelimit.
- **Dependencies:** Lock versions in package-lock.json. Run npm audit before deploy.

15. File Structure & Project Organization

Monorepo with standard Next.js App Router structure, organized by feature:

```
src/app/ — Pages and layouts (App Router)
  /board/[id]/ — Board page with tldraw + Liveblocks provider
  /dashboard/ — Board listing and management
  /api/liveblocks-auth/ — Liveblocks authentication endpoint
  /api/ai/command/ — AI agent command processing

src/components/ — React components
  /board/ — Custom tldraw tools, overlays, AI command input
  /ui/ — Shared UI (toolbar, sidebar, modals)

src/lib/ — Shared utilities
  /liveblocks.ts — Liveblocks config + types
  /ai/ — Tool definitions, prompts, OpenAI client
  /auth.ts — NextAuth.js configuration

src/server/ — tRPC routers
src/types/ — TypeScript definitions
prisma/schema.prisma — DB schema (User, Board, Account, Session)
```

16. Naming Conventions & Code Style

- **TypeScript:** Strict mode. No explicit 'any'. Zod for runtime validation of AI outputs.
- **Components:** PascalCase filenames (BoardCanvas.tsx, AICommandBar.tsx).

- **Hooks:** camelCase with 'use' prefix (useBoardState.ts, useAIAgent.ts).
- **Types:** PascalCase (BoardObject, StickyNoteData, AICommand).
- **API routes:** kebab-case directories (api/ai/command, api/liveblocks-auth).
- **Linting:** ESLint (Next.js config) + Prettier + prettier-plugin-tailwindcss.

17. Testing Strategy

Primary method (MVP): Manual testing with two browser windows side-by-side. Most effective for catching real-time sync issues during rapid development.

Unit tests: Vitest for AI tool parsing and board state utilities.

E2E tests: Playwright for critical paths.

E2E scenarios from spec:

- 2 users editing simultaneously in different browsers
- One user refreshing mid-edit (state persistence check)
- Rapid creation and movement of sticky notes (sync performance)
- Network throttling and disconnection recovery
- 5+ concurrent users without degradation

Coverage target: No strict percentage. Prioritize sync reliability and AI command accuracy.

18. Recommended Tooling & DX

- **AI dev tools:** Claude Code (scaffolding/architecture) + Cursor (iterative editing). Satisfies the 2-tool requirement.
- **VS Code:** ESLint, Prettier, Tailwind IntelliSense, Prisma, Error Lens.
- **CLI:** Vercel CLI (vercel dev), npx prisma studio.
- **Debugging:** React DevTools, Liveblocks DevTools extension, Chrome Network tab (WebSocket frames), Performance tab (FPS).
- **Env management:** dotenv locally, Vercel env vars in production. Never commit .env files.

Final Stack Summary & Build Plan

Locked-In Stack

Layer	Choice	Rationale
Frontend Framework	Next.js 14+ (App Router)	Native T3 stack. Best Vercel integration.
Canvas SDK	tldraw	Saves 15-20 hrs. Full whiteboard UX out of the box.
Real-Time Sync	Liveblocks	Eliminates hardest problem. CRDT, presence, cursors.
Authentication	NextAuth.js	Native T3. Open source. Google + GitHub OAuth.
Database	Supabase PostgreSQL	Free Postgres for metadata. Dashboard for debugging.
ORM	Prisma	Native T3. Type-safe queries. Auto-migration.
AI Agent	OpenAI GPT-4.1-mini	100% tool call reliability. 20x cheaper than alternatives.
Styling	Tailwind CSS	T3 stack standard. Rapid prototyping.
Type Safety	TypeScript + tRPC	End-to-end type safety from DB to frontend.
Deployment	Vercel	Zero-config Next.js. Free tier. No WebSocket server needed.
AI Dev Tools	Claude Code + Cursor	Satisfies 2-tool AI-first development requirement.

Build Priority Order

Following the spec's recommended priority, with time estimates:

#	Task	Time	Gate	Key Tech
1	Project setup (Next.js + tldraw + Liveblocks + NextAuth)	2-3h	MVP	T3 scaffold + configs
2	Cursor sync (Liveblocks presence + tldraw)	1-2h	MVP	useOthers(), presence API
3	Object sync (Liveblocks storage + tldraw store)	2-3h	MVP	LiveObject, store sync
4	Conflict handling (automatic via CRDT)	0h	MVP	Built into Liveblocks
5	State persistence (automatic via storage)	0h	MVP	Built into Liveblocks
6	Auth (NextAuth + Google/GitHub OAuth)	1-2h	MVP	Prisma adapter, JWT
7	Deploy to Vercel	30m	MVP	vercel deploy
8	Board features (shapes, frames, connectors)	4-6h	Early	tldraw custom shapes

#	Task	Time	Gate	Key Tech
9	AI commands: basic (create, move, color)	3-4h	Early	GPT-4.1-mini tools
10	AI commands: complex (templates, layouts)	4-6h	Final	App-side coord math
11	Polish, docs, demo video	4-6h	Final	README, screen record

Total estimated time: 22-34 hours across the week. The tldraw + Liveblocks combo saves 20-30 hours vs react-konva + custom Socket.io. This is the critical margin for a solo developer.

Risk Mitigation

Risk	Impact	Likelihood	Mitigation
Liveblocks free tier hit	Medium	Low	Monitor MAU. 100 is far above sprint needs.
tldraw breaking update	High	Low	Pin version. Don't upgrade during sprint.
GPT-4.1-mini fails on layouts	Medium	Medium	Compute coords in app code. Keep LLM simple.
Vercel API route timeout	Medium	Low	Mini responds in <1s. Use streaming if needed.
tldraw license (commercial)	Low	N/A	OK for evaluation. Review if productionizing.
500+ object perf drop	Medium	Low	tldraw viewport culling. Test early with data.
NextAuth + Liveblocks auth	Medium	Low	Both use JWT. Pass session to LB endpoint.

AI Cost Analysis (Production Projections)

Assumptions: 500 input + 200 output tokens/command. 10 commands/user/session. 2 sessions/user/month. GPT-4.1-mini: $0.40/M input, $1.60/M output.

Scale	Monthly Commands	Input Tokens	Output Tokens	AI Cost/Month	Est. Total Infra
100 users	2,000	1M	400K	$1.04	$1-5/month
1,000 users	20,000	10M	4M	$10.40	$35-60/month
10,000 users	200,000	100M	40M	$104	$200-350/month
100,000 users	2,000,000	1B	400M	$1,040	$2,000-4,000/month

Key takeaway: GPT-4.1-mini makes AI costs nearly negligible below 100K users. Even at 100K users ($1,040/month), it is a fraction of what larger models would cost ($9,000/month for Claude Sonnet). The real cost driver at scale is Liveblocks pricing, not AI.