AI Development Log

AI Whiteboard — Collaborative Drawing Application

1. Tools & Workflow

- **Claude Code (CLI)** — Anthropic's agentic coding tool, powered by Claude Opus 4.6. Used as the primary development agent for all code generation, debugging, refactoring, and deployment orchestration.
- **Vercel Platform** — Hosting and serverless functions. Deployments triggered directly from the CLI via the Vercel MCP, enabling a tight code!'build!'deploy!'test loop within a single conversation.
- **Supabase** — PostgreSQL database, Row-Level Security, Auth, and Realtime (Broadcast, Presence, Postgres Changes). Managed entirely through the Supabase MCP without leaving the agent session.
- **React + Konva + Zustand** — Frontend stack chosen for canvas-based rendering, with state management via Zustand. All component code was generated and iterated on by the AI agent.

The workflow followed rapid iteration cycles: the user described a feature or bug, Claude Code explored the codebase, wrote the code, built, deployed to Vercel production, committed, and pushed—all within the same conversation turn.

2. MCP Usage

- **Vercel MCP** — Enabled one-command production deployments (deploy_to_vercel), inspection of build and runtime logs, fetching deployment URLs to verify fixes, and listing project/team metadata. This eliminated context-switching to the Vercel dashboard.
- **Supabase MCP** — Used to query the database schema (list_tables, execute_sql), check RLS policies, verify Realtime publication configuration (supabase_realtime), inspect replica identity settings, search Supabase documentation, and retrieve project URLs and API keys. Critical for diagnosing the real-time sync issue.

Both MCPs allowed the AI agent to perform full-stack diagnostics without the user needing to manually check dashboards or run separate CLI tools.

3. Effective Prompts

1. Plan-based feature request: "Implement the following plan: [detailed plan for Transformer-based resize handles + canvas minimap]." Providing a structured, step-by-step plan with specific file paths, component names, and prop signatures produced the most accurate first-pass implementation.

2. Miro-style behavioral reference: "Arrows should behave like connectors, as in Miro." Referencing a well-known product gave the agent a clear mental model for snap-to-object behavior, edge-point calculation, and live re-routing when connected objects move.

3. Multi-feature batch request: "The cursor should display a label with the name of the user. Objects should also rotate. Add frames to group and organize content areas. Allow single and multi-select. Also allow duplicate objects and copy/paste." Batching related features in one prompt kept them architecturally consistent.

4. Bug report with screenshot: "Cursor label is not displaying. Minimap is not showing objects. Multi-select should move all objects at once. Please fix this and the warning reported in the attached file." Pairing visual evidence (screenshot of console errors) with clear descriptions of expected behavior was the most efficient way to communicate bugs.

5. Cross-browser + real-time issue: "Cursor label is displayed in Chrome but not in Edge. Objects created by one user should appear for all users in real time." Specifying the browser discrepancy directed the agent to investigate rendering differences and choose a more portable approach (SVG Path vs closed Line).

4. Code Analysis

Estimated breakdown of the final codebase:

- **~95% AI-generated** — All frontend components (WhiteboardCanvas, Minimap, Toolbar, AIPanel, AuthModal), the Zustand store, the Vercel serverless API function, Supabase schema, and RLS policies were written entirely by Claude Code.
- **~5% human-authored** — Initial project scaffolding, environment variable configuration, Supabase project/dashboard setup, and high-level architectural plans provided as prompts.

Across ~10 iterative sessions, the agent produced roughly 2,500 lines of application code spanning 8 source files, plus database migrations and deployment configuration.

5. Strengths & Limitations

Where AI excelled:

- **Rapid multi-file iteration** — Coordinated changes across store, components, and config in a single pass.
- **Debugging from visual evidence** — Diagnosed 500 errors, missing dependencies, and RLS misconfigurations from screenshots and error text.
- **Integrated deployment** — Build!'deploy!'verify cycles completed in under 60 seconds via MCP tools.
- **API and SDK knowledge** — Correctly used Konva Transformer API, Supabase Realtime Broadcast/Presence, and Anthropic tool-use patterns.

Where AI struggled:

- **Ref management in React-Konva** — Initial inline ref callbacks caused objects to disappear on re-render; required a second iteration to switch to layer.findOne().
- **Cross-browser rendering** — Assumed canvas Line rendering was consistent across browsers; needed to switch to SVG Path for Edge compatibility.
- **Negative-dimension edge cases** — Right-to-left drawn shapes produced negative widths that broke the minimap; took two passes to fix both bounds calculation and drawing code.
- **Real-time sync architecture** — Initially relied solely on Postgres Changes, which depends on RLS evaluation; had to supplement with explicit Broadcast for reliability.

6. Key Learnings

Structured plans outperform vague requests. Providing a step-by-step plan with file paths and component signatures produced correct code on the first attempt, whereas open-ended prompts required more iterations.

Screenshots are the best bug reports. Attaching a browser screenshot with console errors let the agent pinpoint root causes (missing package.json, invalid JSON parsing) in a single turn.

MCP integrations collapse the feedback loop. Being able to deploy, check logs, query the database, and verify fixes without leaving the conversation reduced each iteration from minutes to seconds.

Small, incremental iterations beat large batches. Each feature or fix cycle (code !' build !' deploy !' test) kept changes reviewable and bugs isolated. Rolling back was never necessary because changes were small and focused.

AI agents need guardrails for edge cases. The agent consistently handled the happy path well but missed edge cases (negative dimensions, cross-browser quirks, race conditions). Human testing after each deploy was essential for catching these.