AI Development Log — CollabBoard

Tools & Workflow

CollabBoard's AI agent was built using **LangChain JS** (orchestration), **OpenAI GPT-4o-mini** (LLM), **Zod** (schema validation), and **Langfuse** (observability/cost tracking). The workflow follows a structured-output pattern: user prompts and serialized board state are sent to GPT-4o-mini in JSON mode, which returns an array of typed board actions. These actions are validated against a Zod discriminated union of 13 action schemas, then executed client-side against the tldraw editor API. Langfuse traces every LLM call with session/user metadata for latency monitoring and cost attribution.

Multiple AI coding tools were used throughout development: **Claude Code** for scaffolding, refactoring, and test generation; **GitHub Copilot** for inline code completion; and **ChatGPT** for code reviews, requirements analysis, brainstorming architectural approaches, and comparing implementation strategies (e.g., Yjs vs Liveblocks, PartyKit vs Cloudflare Durable Objects).

MCP Usage

No external MCP servers were used at runtime. During development, Claude Code with built-in tools (file read/write, bash, grep, glob) was used for agentic coding tasks. ChatGPT was used conversationally for requirements review, brainstorming feature approaches, and comparing technology trade-offs. Langfuse acts as the observability layer that an MCP-based tracing server would typically provide.

Effective Prompts

1. **Tool schema enforcement:** *"Every action object MUST have a 'type' field set to one of these EXACT strings: create_sticky, create_multiple_stickies, create_text, create_shape..."* — Eliminated type invention by GPT-4o-mini.
2. **Board-state context injection:** *"CURRENT BOARD STATE (N items): [{id, type, x, y, text}...]"* — Appended to every user message so the LLM reasons about existing content and avoids overlaps.
3. **Layout guidelines:** *"Sticky notes are ~200x200px. Space them at least 230px apart. For brainstorming, use a grid layout starting from (-400, -300)."* — Produced clean layouts without post-processing.
4. **Template examples:** *"SWOT Analysis: Create 4 frames in a 2x2 grid, each ~500x400. Add starter stickies inside each."* — One-shot examples dramatically improved output quality.
5. **Type normalization:** A 25+ entry alias map (createSticky->create_sticky, note->create_sticky) catches LLM type-name drift, improving reliability from ~85% to ~99% parse success.

Code Analysis

Approximately **75% AI-generated, 25% hand-written** (across Claude Code, Copilot, and ChatGPT combined). Claude Code generated the scaffolding (agent.ts, tools.ts, executeActions.ts), Zod schemas, and test suites. Copilot assisted with inline completions. ChatGPT was used for reviewing requirements, brainstorming system prompt strategies, and comparing architectural options. Hand-written work focused on the type normalization map, Langfuse tuning, Yjs-tldraw sync, PartyKit server, and production hardening.

Strengths & Limitations

Excelled: Generating Zod schemas for 13 action types, writing comprehensive test suites (1,600+ lines), scaffolding LangChain boilerplate, iterating on system prompt wording, and producing structured repetitive code.

Struggled: tldraw-specific APIs (shape creation props, editor method signatures) required manual correction. Yjs-tldraw sync had subtle race conditions AI couldn't debug alone. Firebase credential configuration and platform-specific native binary issues required human domain knowledge.

Key Learnings

Structured output (JSON mode + Zod) is the most impactful pattern for AI agents — it makes LLM output deterministic enough for programmatic consumption. A type-alias normalization layer is essential since LLMs occasionally invent type names. One-shot examples in system prompts are worth more than paragraphs of instructions. Langfuse tracing is invaluable for debugging latency spikes and cost attribution.

AI Cost Analysis

Development & Testing Costs

Metric	Value
LLM API costs (OpenAI)	~$0.80
Input tokens consumed	~320,000

Output tokens consumed	~85,000
Total API calls	~150
Langfuse hosting	Free tier (50K obs/mo)
Other AI costs	$0 (no embeddings/fine-tuning)

Production Cost Projections

Assumptions: 5 AI commands/user/session, 8 sessions/user/month, ~1,200 input tokens/command (system prompt + board state), ~500 output tokens/command. GPT-4o-mini: $0.15/1M input, $0.60/1M output.

Scale	Commands/mo	Input tokens	Output tokens	Monthly cost
100 users	4,000	4.8M	2.0M	**~$1.92**
1,000 users	40,000	48M	20M	**~$19.20**
10,000 users	400,000	480M	200M	**~$192**
100,000 users	4,000,000	4.8B	2.0B	**~$1,920**

At scale, costs are dominated by output tokens. OpenAI prompt caching could reduce input costs ~50%. Batching commands and response streaming can further optimize perceived latency.