PRESEARCH.md

Date: 2026-02-16
Source: mvp-1-collab-board/G4 Week 1 - CollabBoard-requirements.pdf

1) Problem and Constraints

We need to ship a real-time collaborative whiteboard with an AI board agent in one sprint.

Hard deadlines:

- Pre-Search checkpoint: Monday, 2026-02-16 (first hour)
- MVP checkpoint: Tuesday, 2026-02-17 (24 hours)
- Early submission target: Friday, 2026-02-20
- Final deadline: Sunday, 2026-02-22, 10:59 PM CT

MVP hard gate (must all pass):

- Infinite board with pan/zoom
- Sticky notes with editable text
- At least one shape type
- Create/move/edit objects
- Real-time sync (2+ users)
- Multiplayer cursors with labels
- Presence awareness
- User authentication
- Public deployment

2) Phase 1: Define Constraints

2.1 Scale and Load Profile (assumptions for sprint)

- Launch: 5-20 concurrent users per board.
- 6 months: 100-500 weekly active users if project extends.
- Traffic pattern: spiky (class demos and review sessions).
- Real-time requirements: object sync target <100ms, cursor sync target <50ms.
- Cold start tolerance: low for collaboration path, medium for AI command path.

2.2 Budget and Cost Ceiling

- Sprint dev/testing budget target: <= $150.
- Early production target (first 1,000 users): <= $700 per month.
- Tradeoff: pay managed infra costs to reduce implementation risk.

2.3 AI Cost Modeling (required)

Assumptions:

- Commands per session: 6
- Sessions per user per month: 8
- Command mix: 80% simple, 20% complex

- Token model:
    - simple command: 900 input + 300 output
    - complex command: 1600 input + 1200 output
- Weighted average tokens per command: 1520
- Tokens per user per month: 6 * 8 * 1520 = 72,960
- Blended LLM cost assumption: $3.20 per 1M tokens (planning value, provider-adjustable)

Projected monthly costs:

Scale	LLM Tokens / Month	LLM Cost	Infra Cost (hosting/db/functions)	Total
100 users	7.296M	$23	$90	$113
1,000 users	72.96M	$233	$220	$453
10,000 users	729.6M	$2,335	$1,250	$3,585
100,000 users	7.296B	$23,347	$7,500	$30,847

2.4 Time to Ship

- Primary priority this week: speed-to-market with stable multiplayer.
- Maintainability guardrails: typed contracts, decision log, automated tests.
- Iteration cadence: daily cut and checkpoint review.

2.5 Compliance and Regulatory

- No healthcare scope in MVP.
- Baseline privacy: least-privilege rules and no secrets in client runtime.
- If later targeting enterprise/government contexts: SOC 2 controls and audit logging roadmap.

2.6 Team and Skills

- Execution baseline: TypeScript-first.
- Best velocity stack this week: React + Firebase + server-side AI command orchestration.

3) Phase 2: Architecture Discovery

Option Comparison

Option	Stack	Pros	Cons	Fit for 1-week sprint
A	React + Konva + Firebase (Auth + Firestore + RTDB presence) + Cloud Functions	Fastest setup, managed auth, realtime primitives	Vendor lock-in, Firestore modeling discipline needed	Best
B	React + Konva + Supabase (Auth + Postgres + Realtime) + Edge Functions	SQL flexibility, good DX	More realtime conflict plumbing for board semantics	Good
C	React + custom WebSocket + Redis + Postgres	Maximum control and tuning	Highest implementation and ops risk	Poor

Selected option: A.

3.1 Hosting and Deployment

- Frontend: Firebase Hosting.
- API and AI actions: Firebase Cloud Functions.
- CI: GitHub Actions for lint, tests, and deploy checks.

3.2 Authentication and Authorization

- Primary MVP auth: Firebase Auth with Google OAuth.
- Fallback if blocked: Firebase email-link.
- Board access control enforced in Firestore rules.

3.3 Data Model Contracts (explicit)

Canonical object schema:

```
type BoardObject = {
  id: string
  boardId: string
  type: 'stickyNote' | 'shape' | 'text' | 'frame' | 'connector'
  position: { x: number; y: number }
  size?: { width: number; height: number }
  rotation?: number
  zIndex: number
  color?: string
  text?: string
  shapeType?: 'rectangle' | 'circle' | 'line'
  fromId?: string
  toId?: string
  createdBy: string
  createdAt: number
  updatedBy: string
  updatedAt: number
  version: number
  deleted?: boolean
}
```

Presence and cursor schema:

```
type CursorPresence = {
  boardId: string
  userId: string
  displayName: string
  color: string
  x: number
  y: number
  lastSeen: number
  connectionId: string
}
```

Storage layout:

- Firestore:

    - boards/{boardId} metadata
    - boards/{boardId}/objects/{objectId} canonical objects
    - boards/{boardId}/aiCommands/{commandId} idempotency and status records

- Realtime Database:
    - presence/{boardId}/{userId} ephemeral online status and cursor coordinates

3.4 Conflict Resolution Strategy (explicit)

- Conflict model: Last-Write-Wins (LWW) using authoritative server updatedAt.
- Every object write increments version and sets updatedAt.
- Client UX uses optimistic updates, then reconciles to server state if overwritten.
- Text edits commit on blur/submit (not every keystroke) to reduce collision rate.
- Multi-object AI writes use batched writes; dependent edits use transaction checks where needed.

3.5 Sync Performance Model (explicit)

- Cursor sync: throttle publishes to <=20 updates/sec/user (~50ms interval target).
- Object drag sync: throttle to <=10 updates/sec/object while dragging, plus final commit on pointer up.
- Presence heartbeat: update lastSeen every 10-15s.
- Listener topology per board:
    - one object listener for boards/{boardId}/objects
    - one presence listener for presence/{boardId}
- Index strategy:
    - rely on single-field indexes for MVP subcollection access
    - add composite index only if sort/filter combinations require it
- Batched writes for template generation keep write bursts bounded and predictable.

3.6 AI Agent Execution Model (explicit)

Request path:

1. Client sends command with clientCommandId.
2. Cloud Function validates command and checks idempotency store.
3. Server fetches board context via getBoardState() (MVP cap: latest 500 objects summary).
4. LLM planner returns tool-call plan.
5. Server executes tool calls sequentially and writes updates.
6. Command result persisted in aiCommands and shared to all users through normal board sync.

Concurrency and safety:

- Dispatcher exposes rubric tool functions: createStickyNote, createShape, createFrame, createConnector, moveObject, resizeObject, updateText, changeColor, getBoardState.
- Tool calls execute server-side only (no direct client key usage).
- Idempotency key: clientCommandId per board.
- Simultaneous AI commands: FIFO per-board queue semantics.
- If two users submit commands concurrently, ordering is deterministic by command creation timestamp.

3.7 Offline and Reconnect Strategy

- Enable Firestore web offline persistence.
- Use RTDB onDisconnect() to remove stale presence entries.
- On reconnect:
    - display Reconnecting and Syncing UI states
    - replay local pending writes via Firestore client
    - refresh board state from canonical object listener

3.8 Frontend Rendering Risk and Mitigation

Risk:

- Naive React + Konva re-renders can miss 60 FPS at 500+ objects.

Mitigation:

- Keep hot interaction state in an imperative stage manager (refs/store), not full React tree updates.
- Use React state for controls and metadata, not per-frame pointer movement.

3.9 Testing Tooling

- Unit: Vitest for reducers/transforms/command parsing.
- Integration: Firebase Emulator Suite for auth/rules/sync flows.
- E2E: Playwright with two browser contexts and scripted multi-user scenarios.

4) Phase 3: Post-Stack Refinement

Security Risks and Mitigations

- Risk: over-permissive database rules.
    - Mitigation: deny-by-default rules and explicit board membership checks.
- Risk: prompt injection via AI command text.
    - Mitigation: strict tool schema validation and no arbitrary code execution.
- Risk: leaked API keys.
    - Mitigation: server-only keys in environment secrets.

Project Structure

- apps/web/ React whiteboard app
- apps/functions/ AI and server workflows
- packages/shared/ shared types and schemas

Naming and Style

- TypeScript strict mode.
- ESLint + Prettier.
- Conventions: camelCase vars/functions, PascalCase components/types.

5) Final Stack Decision

- Frontend: React + TypeScript + Konva
- Realtime and auth: Firebase (Firestore + RTDB + Auth)
- AI: function-calling through server-side dispatcher
- Hosting: Firebase Hosting + Cloud Functions

Why this stack now:

- Minimum integration overhead for 24-hour MVP gate.
- Fastest path to stable collaboration and auth.
- Lock-in risk accepted for short-term delivery confidence.