Real-Time Collaborative Whiteboard using WebSockets

Project Abstract · K-1814 · Branch: Computer / IT · Type: Software (built-to-order)

1. Introduction / Background

Real-time collaboration is now an everyday expectation: distributed teams brainstorm on shared canvases, tutors teach on digital whiteboards, and study groups sketch ideas together across cities. The web platform supports this through WebSockets — a protocol that upgrades an HTTP connection into a persistent, full-duplex channel over which either side can push data at any moment. Combined with Node.js on the server and the HTML5 Canvas API in the browser, WebSockets make it practical for a student team to build a genuine multi-user product: a whiteboard where every participant draws on one shared canvas and sees everyone else's strokes appear live, with no page refresh and no polling.

2. Problem Statement

Remote sketching tools available today are either view-only (screen-sharing a drawing app) or closed SaaS products whose synchronization internals are invisible to the learner. A student who wants to truly understand real-time systems — persistent connections, event broadcast, room isolation, presence, and late-joiner state convergence — must build such a system. The problem this project addresses is therefore twofold: (a) provide a working multi-user whiteboard that small teams can actually use for brainstorming and design reviews, and (b) make every layer of its real-time synchronization explicit and explainable, so the build doubles as a teaching vehicle for WebSocket-based architectures.

3. Objectives

- Design and implement a JSON event protocol for whiteboard operations (stroke start/point/end, shape add, sticky-note add, text add, undo, clear).
- Build a Node.js + Express + Socket.IO server that hosts isolated drawing rooms, broadcasts events, tracks presence, and replays history to late joiners.
- Build a browser client on the HTML5 Canvas API with pen, eraser, shapes, sticky notes, text labels, synchronized undo/redo, and PNG export.
- Demonstrate convergence: all participants in a room see the identical board state at all times during live editing.
- Document the system (report, presentation, viva Q&A) so the student can defend the protocol, architecture, and trade-offs.

4. Proposed Methodology

The project follows an event-sourced collaboration design. The canvas is treated as a function of an ordered event log rather than as shared mutable state. The methodology proceeds in four stages. First, the protocol is specified: every whiteboard operation is mapped to a typed JSON message with the fields needed to reproduce it on any client. Second, the server is implemented: Socket.IO rooms isolate sessions, incoming events are appended to a per-room in-memory log and broadcast to the room, and join/leave events maintain presence. Third, the client is implemented: a canvas rendering engine applies both local and remote events through one code path, with pointer input decimated before emission. Fourth, the system is validated with a buyer-run testing procedure: multi-browser sessions on a LAN, disconnect/reconnect behaviour, late-joiner replay correctness, and PNG export fidelity.

5. System Architecture / Working

The architecture has three parts. The **client** (vanilla JavaScript, HTML5 Canvas) captures pointer input, serializes operations into protocol events, renders local and remote events identically, and manages undo/redo stacks. The **signaling server** (Node.js + Express + Socket.IO) authenticates room membership by room id, appends events to the room log, broadcasts to room members, emits presence updates, and serves the replay log to newcomers. The **protocol** is the contract between them: *stroke.start* / *stroke.point* / *stroke.end* for freehand drawing, *shape.add* / *note.add* / *text.add* for discrete objects, *board.undo* / *board.clear* for board-level operations, and *room.join* / *client.connect* for presence. A participant's stroke therefore travels client -> WebSocket -> server log -> room broadcast -> every other client's canvas, typically within tens of milliseconds on a LAN (design target; actual latency depends on the network and is measured by the buyer during testing).

6. Hardware / Software Requirements

- **Server:** Node.js 18 or newer, npm dependencies (express, socket.io); runs on a laptop, a LAN machine, or a small VPS — no GPU or special hardware needed.
- **Client:** any modern browser with WebSocket and HTML5 Canvas support (Chrome, Edge, Firefox, Safari); no plugins or installs.
- **Development:** a code editor, Git for version control, and two browser windows/tabs (or devices) to observe live synchronization during testing.
- **Optional deployment:** a VPS or campus server with an open port for public access; a process manager such as PM2 for keeping the server running.

7. Expected Outcomes

- A working multi-user whiteboard: N participants draw, annotate, undo and export on one converged canvas in real time (design target: 10+ concurrent editors per room on a single Node.js instance).
- A documented, extensible JSON event protocol that the student can explain message-by-message in the viva.
- Demonstrated late-joiner convergence: a newcomer reconstructs the exact current board from the server's event log.

- A complete submission kit: source code, working demo, project report, presentation, and viva Q&A.

8. Applications

- Remote brainstorming and sprint planning for student project teams.
- Online tutoring: teacher and students sketch on the same board during a lesson.
- Design reviews: UI wireframes and flowcharts annotated collaboratively.
- Classroom demonstrations of WebSocket and event-driven architectures.
- A base platform for advanced final-year extensions (authentication, persistence, CRDT sync).

9. Future Scope

- User authentication and per-room access control (join-by-link is the current model).
- Persistent storage of boards (Redis or a document database) so history survives server restarts.
- Offline editing with queued-stroke resend and conflict-free replicated data types (CRDTs) for concurrent edits.
- Live cursors and per-user stroke colors for richer presence.
- Mobile application clients consuming the same event protocol.
- Version history / snapshots of the board over time.

10. Conclusion

This project delivers a real, usable piece of real-time infrastructure — not a mockup and not a tutorial copy. By implementing the event protocol, the room server, and the canvas client from first principles with standard web technology, the student gains a concrete, defensible understanding of how modern collaborative applications stay in sync. The built-to-order deliverable includes the complete source, a working demo, and full documentation, ready for submission and viva.