WebSocket-Powered Kanban Board

Submission by –

Raj Patkar
Email : raj5patkar@gmail.com
phone : 8237398789
project repository : https://github.com/Raj-Patkar/websocket-kanban-vitest-playwright-2026

A real-time Kanban board built with **React** and **WebSockets** as part of a take-home assignment.
The implementation focuses on **real-time collaboration**, **predictable UI behavior**, and **practical engineering trade-offs**.

(All requirements / instructions completed successfully ✓)

Live link

Backend (WebSocket Server)

🔗 https://kanban-board-assignment-55ov.onrender.com/

⚠️ **Important Note:**
Visiting the backend URL directly in a browser may show **"Cannot GET /"**.
This is **expected behavior** as there is no html route.

Frontend (UI)

turn on backend instances before frontend

🔗 https://kanban-board-app-0n7x.onrender.com

- This is the main user-facing application
- Open this link to use the Kanban board
- Supports real-time collaboration across multiple browsers

Overview

This project allows multiple users to collaborate on a Kanban board in real time.
All task updates are synchronized instantly across connected browser clients using WebSockets.

The solution prioritizes:

- Correct real-time behavior
- Clean and maintainable UI logic

- Explicit, well-justified technical decisions

Core Features

- Create, update, delete, and move tasks across columns
- Real-time synchronization across multiple browser clients
- Task priority and category selection
- File attachments with live preview
- Live progress visualization
- Unit, integration, and end-to-end testing

Design Decisions

Database Choice (MongoDB Not Used)

MongoDB was intentionally **not used**.

Reasoning:

- The primary evaluation focus is **WebSocket-based real-time synchronization**
- Adding MongoDB would increase complexity without improving real-time behavior
- In-memory state simplifies reasoning about broadcasts and client consistency

The backend maintains task state in memory and broadcasts updates via WebSockets.

Attachment Handling (Base64 Instead of URLs)

Attachments are stored and transmitted as **Base64 data**, not external URLs.

Why this approach was chosen:

- Ensures instant synchronization across all connected browsers
- Avoids dependency on external file servers or cloud storage
- Prevents broken or inaccessible URLs across different clients

Trade-off acknowledged:

- Base64 increases payload size
- This is acceptable due to controlled file size limits and demo scope

This choice prioritizes **correctness and consistency across browsers** over storage optimization.

Testing Strategy

- Unit tests validate UI logic and component behavior
- Integration tests verify WebSocket-driven real-time synchronization
- End-to-end tests cover only critical user journeys

This keeps tests fast, reliable, and maintainable while providing high confidence.

Tests Implemented

Unit Tests (Vitest + React Testing Library)

- Component rendering
- Form validation
- Dropdown selection
- File upload validation
- Task creation and deletion logic

Integration Tests (Vitest)

- WebSocket event handling
- Real-time state synchronization across multiple clients

End-to-End Tests (Playwright)

- User can create a task
- User can delete a task
- Verified on Chromium, Firefox, and WebKit