

Assessment Title: Real-Time Kanban Board (2-User Sync)

1. Problem Statement

Build a real-time Kanban board where two users working on the same board can see updates instantly without refreshing the page.

Users should be able to:

- Create cards
- Edit card details
- Drag cards within a column and between columns
- See all changes reflected in real time for other users

This assessment focuses on backend architecture, real-time synchronization, and frontend state consistency, not UI polish.

2. Functional Requirements

Kanban Board Structure

The board must contain exactly 3 columns:

- Todo
- Doing
- Done

Card Data Model

Each card must have the following fields:

Field	Type	Notes
id	string / uuid	Unique identifier
title	string	Required
description	string	Optional
status	enum	Todo / Doing / Done
order	number	Used for sorting within a column
updatedAt	timestamp	Used for conflict handling

3. Core Features

Card Operations

- Create a new card in any column
- Edit card title and description
- Move cards:
 - Within the same column (reorder)
 - Across different columns
- Maintain correct order after drag & drop

Drag & Drop

- Must support drag-and-drop using any library (e.g. `@dnd-kit`, `react-beautiful-dnd`)
- Order must be persisted in the database
- Reordering must sync in real time across users

4. Real-Time Synchronization (Critical)

- Two users (User A and User B) should see updates **instantly**
- No page refresh allowed
- Must use WebSockets
 - Socket.IO or native WebSocket (`ws`) is acceptable
- Events to sync:
 - Card created
 - Card updated
 - Card moved
 - Card reordered

REST APIs should be used for initial data fetch, WebSockets for live updates.

5. Conflict Handling (Required)

Concurrent edits must be handled explicitly.

You must choose one of the following strategies and explain it clearly in the README:

Option A – Last Write Wins

- Use `updatedAt` or a `version` field

- Latest update overwrites previous ones

Option B – Optimistic Locking

- Maintain a **version** number per card
- Reject conflicting updates with **409 Conflict**

Option C – Soft Locks

- Short-lived locks when a user edits or drags a card
- Other users see the card as “locked”

*Whatever you choose:

- Implement it properly
- Explain trade-offs in the README

6. Authentication (Lightweight)

- Full authentication is **NOT required**
- Mock users are sufficient(provide credentials in README):
 - User A
 - User B
- You may hardcode users or simulate via headers / query params / socket connection

7. Required Tech Stack

Frontend

- **Next.js 14 or more**
 - **TypeScript**
 - App Router preferred
- Clean state management (React state, Zustand, etc.)

Backend

- **Node.js**
- Either:
 - Next.js Route Handlers
 - OR a custom Node server

Database

- PostgreSQL

ORM

Choose one:

- Prisma
- Drizzle
- Raw SQL (must be well-structured)

Real-Time

- WebSockets
- Socket.IO or `ws`

8. API Requirements

Implement REST APIs for:

- Fetch board data
- Create card
- Update card
- Update card position / order

WebSockets should broadcast changes to all connected users.

9. Deliverables

You must submit:

- Git repository (GitHub / GitLab)
- `README.md` containing:
 - Setup instructions
 - Architecture overview
 - Database schema explanation
 - Real-time communication design
 - Conflict-handling strategy & trade-offs
 - Stretch question answer (see below)
- `.env.example` file (no secrets)

10. Evaluation Criteria

You will be evaluated on:

- Backend architecture & data modeling
- Real-time correctness (no desync issues)
- Frontend state management quality
- TypeScript usage & type safety
- Code readability & structure
- Clear communication in README
- Thoughtful trade-off explanations

UI design is **not** a primary focus.

11. Stretch Question (Answer in README)

If this system needed to support 100,000 concurrent users, what would you change first and why?