Project Context

You're building a core subsystem of **adam.ai** — a lightweight task management platform designed for fast-paced teams that have user experience similar to Trello or Jira.

This challenge is based on a **real feature within adam.ai and** is meant to assess how you take a real-world product requirement, design a scalable backend system around it, and implement a working proof-of-concept that reflects solid engineering, architecture practices.

Scope of Work

Design and implement an MVP of the **Task Management System**, with the following key capabilities:

1. Task Management Service

- Users can create, edit, assign, delete, and view tasks.
    - Feel free to create a user service or make this service modular monolithic
- Each task includes:
    - title
    - description
    - status (To Do, In Progress, Late, Done)
    - due date
    - assignee
    - creator
- Tasks can be assigned to oneself or to other users.

2. Task Board Interface

- A simple, authenticated web interface that displays tasks in a **Kanban-style** board (segmented by status).
- The users should be able to see tickets status changing **in real-time**
- Status updates should persist in the backend.
- Every task has an order and can be ordered in the board
- (Bonus): User can reorder with drag and drop

- (Bonus): Users must be able to move tasks across statuses via drag-and-drop or equivalent interactions.

3. Reminder Service

- The service should receive messages through message broker from Task Management Service
- This functionality must remain reliable regardless of task creation time (i.e., it should handle long-term and short-term deadlines).
- Tasks approaching their due date (e.g., 24 hours prior) should trigger an email and it sends it through SMTP to the assignee
- (Note: You don't need to provide working credentials in SMTP, we'll use our own so you need to make your service easily configurable)

Bonus Requirement: Propose and Implement an Additional Feature

As part of this technical challenge, candidates are required to **propose and implement one additional feature** of their choice to extend the functionality of the Task Management System.

- Candidates must **research popular task management platforms** (e.g., Jira, Trello, ClickUp, Linear) and select a meaningful feature that aligns with the goals of a collaborative task board.
- The selected feature should add tangible value to the end user and demonstrate thoughtful product thinking. Examples of acceptable features include, but are not limited to:
    - Task comments or threaded discussions
    - Task activity history or change logs
    - Sub-tasks or checklists
    - Watchers/followers and user notifications

The candidate is expected to **exercise architectural judgment** in determining whether the proposed feature should:

- Be **integrated within the existing Task Management Service**, or

- Be implemented as a **standalone service**, communicating with the rest of the system via appropriate interfaces (e.g., REST API, message broker)

Design decisions must be explicitly documented in the submission. Candidates should consider aspects such as scalability, cohesion, separation of concerns, and potential for future extensibility when justifying their approach.

- Candidates will receive **additional credit** for implementing a feature that is non-trivial in complexity or presents interesting design challenges.
- Greater emphasis will be placed on the **quality of the analysis**, **reasoning behind the architectural choice**, and the **clarity of documentation**, rather than the feature's scope alone.

Technical Guidelines

We recommend the following stack, but you may choose alternatives you are confident with:

- **Backend**: Node.js or .NET
- **Frontend**: React
- **Database**: PostgreSQL or MongoDB
- **Message Broker**: RabbitMQ or Kafka
- You may assume a simplified authentication setup; full auth implementation is not required.

Delivery Requirements

Your submission should include the following:

1. **Working Application**
    a. A complete MVP that runs locally via Docker Compose.
    b. All services properly containerized and orchestrated.
    c. A functional frontend interface that connects to your backend.

2. **Documentation**
    a. A README.md file that includes:
        i. A brief description of the system architecture.
        ii. Service breakdown and responsibilities.
        iii. Why you chose SQL/NoSQL
        iv. Setup and run instructions (including Docker commands).
        v. Example user credentials or usage flows.
        vi. Justification for design decisions (e.g., service decomposition, messaging patterns).
        vii. Sequence Diagram
3. **Code Quality**
    a. Use consistent naming, structure, and error handling practices.
    b. Implement appropriate validation and separation of concerns.

Notes

- The goal of this challenge is not to build a production-grade system, but to demonstrate architectural thinking, technical quality, and ownership.
- You will be asked to present and walk through your work as part of the interview process.

We look forward to reviewing your solution and learning more about your approach.