🧑‍💻 Collaborative Task Manager: Full-Stack Engineering Assessment

🎯 Objective

Design and build a complete, production-ready, full-stack Task Management application. This assessment evaluates your ability to implement core application logic, ensure data integrity, and handle real-time communication using modern JavaScript/TypeScript best practices across the frontend and backend.

⚙️ Technical Specifications (The Stack)

To ensure a consistent and modern development environment, please adhere to the following technologies:

Layer	Technology	Requirement
Frontend	**React** (via Vite or Next.js Pages Router)	Must use **TypeScript** and **Tailwind CSS**.
Data Fetching	**SWR** or **React Query**	Mandatory for managing server state and caching.
Backend	**Node.js + Express** (with TypeScript)	Must implement a clear service/repository pattern.
Database	**PostgreSQL** or **MongoDB**	Choose one and justify the selection in your README.
ORM/ODM	**Prisma** (Highly Recommended) or Mongoose	Use an industry-standard library for database interaction.
Real-Time	**Socket.io**	Mandatory for collaboration features.
Deployment	Vercel/Netlify (FE) + Render/Railway (BE/DB)	Mandatory live deployment.

🧹 Core Requirements (Must Haves)

1. User Authentication & Authorization

- **Secure Auth:** Implement user registration and login. Passwords must be hashed (e.g., using bcrypt).

- **Session Management:** Implement session handling, preferably using **JWT (JSON Web Tokens)** stored securely (e.g., HttpOnly cookies).
- **User Profiles:** Allow users to view and update their name/profile information.

2. Task Management (CRUD)

Implement full CRUD operations for tasks. A task must include the following attributes:

- title (string, max 100 chars)
- description (string, multi-line)
- dueDate (date/time)
- priority (Enum: Low, Medium, High, Urgent)
- status (Enum: To Do, In Progress, Review, Completed)
- creatorId (The ID of the user who created the task)
- assignedToId (The ID of the registered user currently responsible for the task)

3. Real-Time Collaboration (Mandatory)

Implement real-time updates using **Socket.io**:

- **Live Updates:** When a task's status, priority, or assignee is updated by *any* user, all other users currently viewing the task list or dashboard must see the change instantly.
- **Assignment Notification:** Send an instant, persistent in-app notification to a user when a task is assigned to them.

4. User Dashboard & Data Exploration

- **Personal Views:** A dedicated dashboard page showing:
  - Tasks assigned to the current user.
  - Tasks created by the current user.
  - Tasks that are **Overdue** (based on the dueDate).
- **Filtering & Sorting:** Implement filtering on the task list based on **Status** and **Priority**. Implement sorting by **Due Date**.

🏗️ Engineering & Architecture Quality

The evaluation will heavily weigh how you approach the engineering challenge:

1. Backend Reliability

- **Clear Architecture:** Use a proper architecture (e.g., MVC-like separation, with dedicated Controllers, Services, and Repositories/Data Access Layers).
- **Data Transfer Objects (DTOs):** Implement DTOs (using Zod or Class Validator) for input validation on all API endpoints (e.g., CreateTaskDto, UpdateTaskDto).
- **Error Handling:** Implement consistent and meaningful HTTP status codes and error responses for failed requests (e.g., 401 for unauthorized, 404 for not found, 400 for validation errors).

2. Frontend UX & Data Management

- **Responsive UI:** The application must be fully responsive and optimized for both desktop and mobile views (using Tailwind CSS).
- **Loading States:** Implement visible **Skeleton Loading States** (or similar smooth transitions) when fetching data using SWR/React Query.

- **Form Management:** Use a dedicated library (e.g., React Hook Form, Formik) with validation (e.g., Zod, Yup) for task creation/editing forms.

3. Code Quality & Testing

- **TypeScript Usage:** Utilize strong typing throughout the application (both FE and BE) to enforce type safety.
- **Documentation:** Add JSDoc/TSDoc comments for complex functions, data models, and API endpoints.
- **Mandatory Testing:** Implement at least **3 unit tests** for a critical piece of backend business logic (e.g., the service function that handles task creation validation, or the WebSocket logic for assignment). Use Jest or Mocha.

✉️ Submission & Deliverables

Please submit the following through your designated submission portal:

Google Form Link to submit the assignment : https://forms.gle/fv1uZuya2jQR4bK76

1. **Public Git Repository Link:** (e.g., GitHub, GitLab)
   - The repository must have a clear commit history demonstrating a logical development process.
   - Code must be organized into clear frontend/ and backend/ directories.
2. **Live Deployed Project Link(s):** (Frontend URL and Backend API URL, if separate)
3. **Comprehensive README.md** file containing:
   - Setup instructions (how to run the FE/BE locally).
   - API Contract documentation (list of key endpoints: e.g., POST /api/v1/tasks).
   - Architecture Overview & Design Decisions (e.g., Why you chose your DB, how you handled JWT, how you implemented the service layer).
   - A brief explanation of how Socket.io was integrated for real-time functionality.
   - Any trade-offs or assumptions made.

🧪 Evaluation Rubric

Category	**Weight**	**Focus Areas**
Correctness & Functionality	35%	Completeness of CRUD, proper filtering/sorting, secure Auth implementation, and successful real-time updates.
Architecture & Engineering	25%	Clear separation of concerns (Services/Repositories/Controllers), correct use of TypeScript types, and implementation of DTO

		validation.
Data Management & Real-Time	15%	Effective use of SWR/React Query for caching, proper state management, and robust Socket.io integration.
UX & Aesthetics	10%	Responsiveness (mobile-first approach), clean Tailwind CSS design, and clear loading/error states.
Testing & Reliability	10%	Coverage of critical backend logic with unit tests and appropriate error handling/logging.
Documentation & Deployment	5%	Quality of the README, clarity of commit history, and successful live deployment.

✨ Bonus Challenges (Optional, Highly Valued)

If you have time and want to showcase advanced skills:

1. **Optimistic UI:** Implement optimistic updates for task status changes using SWR/React Query features.
2. **Audit Logging:** Implement a simple logging mechanism (on the backend) to record who updated a task's status and when.
3. **Dockerization:** Provide a Dockerfile and a docker-compose.yml file to spin up the entire application stack (FE, BE, DB) locally with a single command.

Important :

- **The process of evaluation of assignment can take 4-8 weeks.**
- **Without a live deployment link the assignment would be rejected.**