### 1. \*\*System Design\*\*

- \*\*System Architecture Diagram:\*\*

- I have created a diagram illustrating how different components of your system will interact.

- Include:

- \*\*Client-side (Frontend):\*\* Single Page Application (SPA) built with React/Angular/Vue.js.

- \*\*Backend:\*\* RESTful API built with Nodejs.

- \*\*Database:\*\* Choose a suitable database like MongoDB

- \*\*Real-time Communication:\*\* WebSocket server for real-time updates (e.g., Socket.IO).

- \*\*Authentication/Authorization:\*\* JWT-based authentication.

- \*\*Load Balancer and Caching:\*\* Use NGINX as a load balancer, and Redis for caching.

- \*\*Database Schema model design:\*\*

- Design a schema that can handle users, tasks, projects, and real-time updates.

- Key tables:

- \*\*Users:\*\* `id` , `name` , `email` , `password\_hash` , etc.

- \*\*Projects:\*\* `id` , `name` , `description` , `created\_by` , `created\_at` , etc.

- \*\*Tasks:\*\* `id` , `title` , `description` , `status` , `assigned\_to` , `project\_id` , `created\_at` , `updated\_at` , etc.

- \*\*Comments:\*\* `id` , `task\_id` , `user\_id` , `content` , `created\_at` .

- \*\*Activity Logs:\*\* `id` , `task\_id` , `action\_type` , `user\_id` , `timestamp` .

- \*\*API Structure:\*\*

- Outline RESTful API endpoints:

- \*\*/api/users:\*\* CRUD operations for users.

- \*\*/api/projects:\*\* CRUD operations for projects.

- \*\*/api/tasks:\*\* CRUD operations for tasks, including assigning and updating tasks.

- \*\*/api/comments:\*\* Add comments to tasks.

- \*\*/api/notifications:\*\* Real-time notifications via WebSockets.

- Describe how endpoints will handle real-time updates, authentication, and error handling.

- \*\*Real-time Communication:\*\*

- Use WebSockets (e.g., with Socket.IO) for real-time task updates and notifications.

- Explain the WebSocket connection lifecycle, including how clients subscribe to task updates and how the server broadcasts changes.

- \*\*Scalability Considerations:\*\*

- Discuss horizontal scaling using multiple server instances behind a load balancer.

- Implement caching with Redis to reduce database load.

- Use a distributed database (like PostgreSQL with read replicas) to handle large-scale data.

### 2. \*\*Backend Implementation\*\*

- \*\*Core Backend Services:\*\*

- Explain your choice of Python framework (Flask/Django/FastAPI).

- Describe how you'll structure the application with models, views, controllers, and services.

- \*\*RESTful APIs:\*\*

- Plan the implementation of CRUD operations for users, projects, and tasks.

- Include placeholder files with pseudocode for context.

- \*\*Real-time Updates:\*\*

- Use WebSockets for broadcasting task updates.

- Describe the setup of WebSocket endpoints and how you'll ensure that updates are efficiently handled.

- \*\*Authentication/Authorization:\*\*

- Implement JWT authentication for securing API endpoints.

- Describe how you'll manage user roles and permissions.

### 3. \*\*Frontend Implementation\*\*

- \*\*Responsive SPA Development:\*\*

- framework (React)

- Describe how you'll structure components, state management (using Redux), and routing.

- \*\*Real-time Updates on Frontend:\*\*

- Describe how the frontend will establish and manage WebSocket connections to receive real-time task updates.

- \*\*UI for Task Management:\*\*

- Plan for drag-and-drop functionality using libraries like React DnD or Vue Draggable.

- Include wireframes or mockups to demonstrate your UI design.