Real-Time Task Collaboration Platform

Frontend Architecture Explanation

• Built using React SPA architecture.
• Component-based modular design (Board, List, Task).
• State handled via React Hooks.
• REST API communication via fetch.
• Real-time updates using Socket.IO client.
• Scalable folder structure.

Backend Architecture Explanation

• Node.js + Express REST API.
• Socket.IO for real-time communication.
• Modular routing structure.
• Middleware ready for authentication & validation.
• Easily extendable to MongoDB/PostgreSQL.

Database Schema Design

Entities:
User(id, name, email, password_hash)
Board(id, name, owner_id)
List(id, name, board_id)
Task(id, title, description, list_id, assigned_user_id)
Activity(id, user_id, action, timestamp)

Relationships:
User → Boards → Lists → Tasks.

API Contract Design

Auth:
POST /signup
POST /login

Boards:
GET /boards
POST /boards

Lists:

POST /boards/:boardId/lists

Tasks:
POST /lists/:listId/tasks
PUT /tasks/:taskId
DELETE /tasks/:taskId

Real-Time Sync Strategy

• WebSocket using Socket.IO.
• On any change → emit 'boardUpdated'.
• All connected clients auto-refresh.
• Ensures multi-user collaboration.

Scalability Considerations

• Stateless backend using JWT.
• Horizontal scaling with load balancer.
• Redis adapter for scaling Socket.IO.
• Indexed database fields.
• Pagination & optimized queries.

API Documentation (Example)

POST /boards

Request:
{
"name": "Project Board"
}

Response:
{
"id": 12345,
"name": "Project Board",
"lists": []
}

Assumptions and Trade-offs

Assumptions:
• Single workspace.
• Basic authentication.

Trade-offs:
• In-memory storage (for demo).
• No advanced RBAC.

Demo Credentials

Email: demo@example.com
Password: password123