An-Najah National University
Computer Engineering Department

Title name:

Task Management System - Project Report

Submatied to : Dr.Amjad Abu Hassan

Student Name:

1-Ahmad Samaaneh

2-Hazim AL Sayel

3-Ahmad Saad

Introduction:

This Task Management System is a web-based application designed to facilitate project and task coordination between administrators and students.

The system provides dashboards, project/task management, and a chat interface. It integrates a secure authentication mechanism and applies a clean, modern UI with theme toggling (light/dark).

Backend Technologies:

- Node.js with Express.js (RESTful API server)
- MySQL (relational database)
- GraphQL

Database Schema and Tables:

1. **users**
   - id: UUID of the user
   - username: Unique username
   - password: Hashed password
   - role: 'admin' or 'student'

2. **projects**
   - id, title, description, category, start_date, end_date, progress

3**. project_members**

- project_id, user_id

4. **tasks**

- id, title, description, status (pending/in_progress/completed), assigned_to, project_id, due_date

5. **conversations**

Id ,created_at

6. **conversation_participants**

conversation_id user_id

7. **messages**

id conversation_id sender_id content is_read created_at

REST API Features:

Authentication:

- Login & JWT-based auth

- Role-based access

Projects:

- GET /projects

- POST /projects

- PATCH /projects/:id

- DELETE /projects/:id

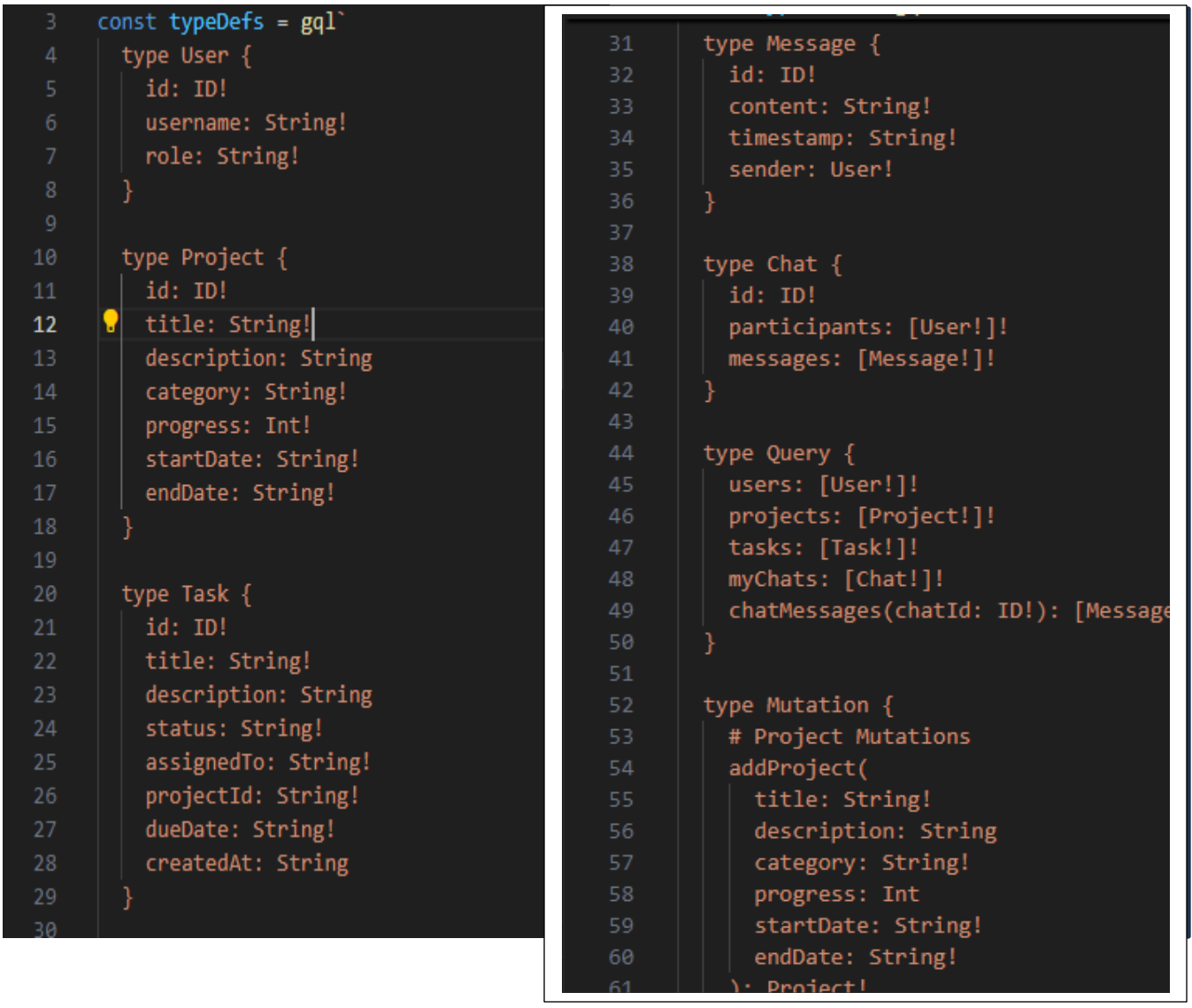
Tasks:

- GET /tasks
- POST /tasks
- PATCH /tasks/:id/status
- PATCH /tasks/:id
- DELETE /tasks/:id

Dashboard:

- GET /dashboard

GraphQL :



```
const typeDefs = gql`
  type User {
    id: ID!
    username: String!
    role: String!
  }

  type Project {
    id: ID!
    title: String!
    description: String
    category: String!
    progress: Int!
    startDate: String!
    endDate: String!
  }

  type Task {
    id: ID!
    title: String!
    description: String
    status: String!
    assignedTo: String!
    projectId: String!
    dueDate: String!
    createdAt: String
  }

  type Message {
    id: ID!
    content: String!
    timestamp: String!
    sender: User!
  }

  type Chat {
    id: ID!
    participants: [User!]!
    messages: [Message!]!
  }

  type Query {
    users: [User!]!
    projects: [Project!]!
    tasks: [Task!]!
    myChats: [Chat!]!
    chatMessages(chatId: ID!): [Message
  }

  type Mutation {
    # Project Mutations
    addProject(
      title: String!
      description: String
      category: String!
      progress: Int
      startDate: String!
      endDate: String!
    ): Project!
```

GraphQL provided flexibility with query, mutation, and resolver patterns.

It was evaluated for reducing round-trips, but REST was chosen for frontend alignment.

Frontend Features:

Built with:

- React (TypeScript)

- Tailwind CSS (dark/light mode support)

- Zustand (state management)

Pages:

- Dashboard, Projects, Tasks, Chat

Components:

- Sidebar, Header (theme toggle), AddTaskModal, ProjectCard, etc.

Chat System :

The Task Management System includes a real-time chat feature that enables communication between admin and students. This chat system has been designed to ensure immediacy and responsiveness in communication without the need for constant page refreshes or polling.

To achieve real-time capabilities, WebSockets have been integrated into the chat architecture. WebSockets provide a persistent connection between the client and server, allowing messages to be sent and received in real-time.

Key components of the implementation include:

-WebSocket Server: A Node.js or Express server (with packages like ws or socket.io) listens for new connections and message events.

Client Integration: The React frontend establishes a WebSocket connection on component mount using `new WebSocket('ws://localhost:PORT')`. Messages are transmitted and received through this connection.

Message Handling: When a message is sent, it is broadcast to relevant connected clients using the WebSocket protocol. This ensures the recipient sees the message instantly.

UI Feedback: The frontend listens for message events and updates the chat interface dynamically using React state.

This feature enhances the interactivity of the system and simulates a real-world task coordination platform. Messages are stored temporarily using in-memory structures (like arrays or matrices in state), suitable for demonstration and local session communication.

In a production setup, these messages would typically be stored in a database (e.g., MySQL or MongoDB) for persistence and historical records. Security concerns such as user authentication over WebSockets and message validation are also considerations for future expansion.

Additional Features:

- Theme toggle across all layouts and components
- Live Clock in Header
- Task Sorting (by status or name)
- Role-Specific Views for Admin/Student

Testing & Validation:

- Manual UI and API testing (Postman)
- Form validation using react-hook-form
- Responsive layout tested across browsers

Conclusion:

The system provides a robust platform for academic or organizational task tracking. The real-time chat enriches communication, and REST APIs offer scalability and performance. Future enhancements may include file sharing in chat, push notifications, and email reminders.