Task Manager Application

Introduction

A task management application designed to streamline project and task handling for teams. Built using a microservices architecture, the system provides secure authentication, project and task management features, and a modular frontend served by NGINX.

Backend

The backend of the Task Manager Application is built with NestJS, a framework designed for scalable server-side applications. It handles all API requests, business logic, and communication with the PostgreSQL database through TypeORM.

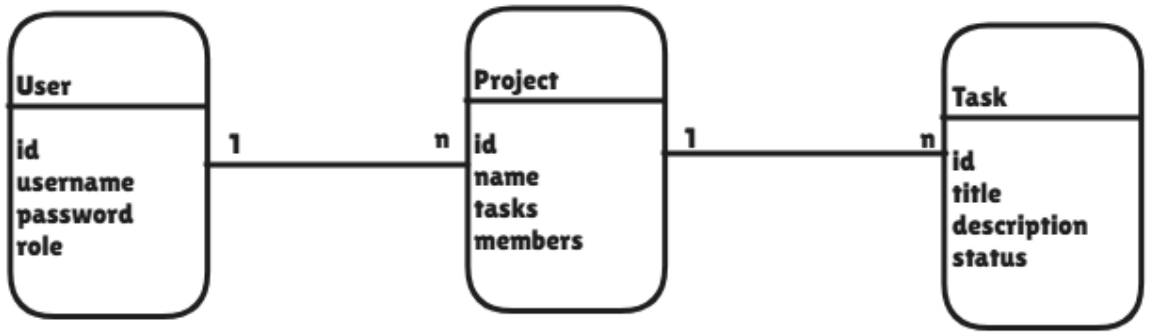

Fig1. Database representation

Security and Authentication

Security is a core aspect of the backend, achieved through JWT (JSON Web Tokens) and Passport, a widely-used authentication middleware.

1. **User Authentication**:
    - When a user logs in, the backend validates their credentials using the Local Strategy.
    - If valid, a JWT is generated and sent to the client. This token contains essential user information (e.g., username, role) and is signed with a secret key to prevent tampering.
2. **Token-Based Authorization**:
    - For each protected route, the backend uses the JWT Strategy to validate incoming requests.
    - The JWT is extracted from the Authorization header in the form Bearer <token>.

    - If the token is valid and not expired, the request is authorized, and the user's identity is attached to the request object (req.user).
    - If the token is invalid or missing, the request is rejected with an UnauthorizedException.
3. **Role-Based Access Control** :
    - Users are assigned roles such as user or team_leader.
    - Specific endpoints, such as project member management, may require elevated permissions based on the user's role.

Backend Services and Modules

The backend is modular, with each feature (e.g., authentication, projects, tasks) implemented as a separate module. These modules are structured into:

- Controllers: Handle incoming HTTP requests and responses.
- Services: Contain business logic, including database interactions and task execution.
- Entities: Define the database schema using TypeORM.

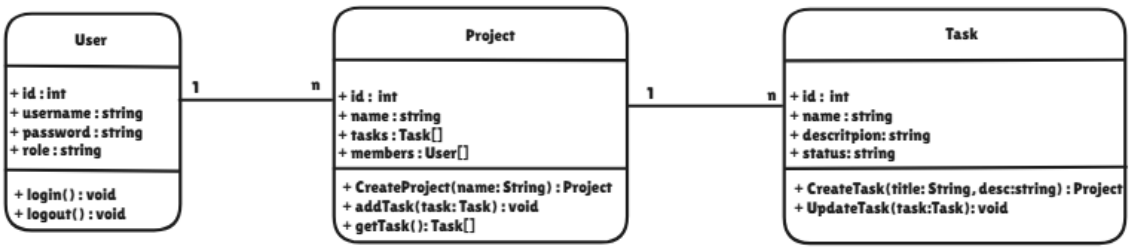

Fig2. UML class diagram

Nginx

The web load balancing setup in this project is handled by NGINX to distribute traffic across multiple backend instances efficiently

Frontend

The frontend of the Task Manager Application is built using a micro-frontend architecture. Each micro-frontend serves a specific feature of the application, allowing independent development and deployment.

Technologies Used:

- **HTML**: Structure of the frontend components.

- **CSS** (with Bootstrap): Styling and responsive design for a consistent and mobile-friendly user interface.
- **JavaScript** (Vanilla): Handles client-side logic and API communication without relying on heavy frameworks.
- **NGINX**: Serves static frontend files and manages routing between micro-frontends.

Micro-Frontend Modules:

The application is divided into multiple frontend modules, each focusing on a core feature:

1. **Auth Module**:
    - Handles user authentication (login, registration).
    - Communicates with the backend /auth API for token-based login.
2. **Dashboard Module**:
    - Displays an overview of projects and tasks assigned to the user.
    - Fetches data from the backend /projects and /tasks APIs.
3. **Projects Module**:
    - Manages project details, including tasks and team members.
    - Provides features to create, update, and delete projects.

Docker

The Task Manager Application uses Docker to containerize and manage the deployment of its various services. Docker ensures that the application can run consistently across different environments, making setup and scaling easier.

Dockerized Components:

1. **Backend (NestJS)**:
    - Multiple backend instances are containerized and load-balanced by NGINX.
    - Containers run the API and handle business logic.
2. **Frontend (Micro-Frontends)**:
    - Static files for each micro-frontend (auth, dashboard, projects) are served by NGINX inside a container.
3. **PostgreSQL Database**:
    - The database is isolated in its own container, with persistent storage using Docker volumes.
4. **NGINX**:
    - Acts as a reverse proxy and static file server, distributing traffic to backend instances and serving frontend content.