TaskFlow Backend API

This is the backend API for TaskFlow, a project management and collaboration tool. It is built using Node.js, Express.js, MySQL, and Socket.IO for real-time features.

Features

- User Authentication (Registration, Login)
- Project Management (CRUD operations for projects)
- Task Management (CRUD operations for tasks within projects)
- Comment Management (CRUD operations for comments on tasks)
- Notification Management (create, retrieve, mark as read, delete notifications)
- Real-time Notifications (using Socket.IO)
- Input Validation
- Error Handling
- CORS Enabled

Technologies Used

- Node.js
- Express.js
- MySQL
- `mysql2/promise` for asynchronous MySQL connectivity
- `bcryptjs` for password hashing
- `jsonwebtoken` for JWT authentication
- `dotenv` for environment variable management
- `express-validator` for input validation
- `cors` for Cross-Origin Resource Sharing

- `socket.io` for real-time communication

Setup Instructions

Prerequisites

- Node.js (v14 or higher)
- npm (Node Package Manager)
- MySQL Server (v8 or higher)

1. Clone the repository

```
git clone <repository_url>
cd taskflow-backend
```

2. Install Dependencies

```
npm install
```

3. Database Setup

1. **Create MySQL Database and User:**

   Open your MySQL client (e.g., MySQL Shell, MySQL Workbench, or command line) and run the following commands to create the database and a dedicated user:

   `sql CREATE DATABASE taskflow_db; CREATE USER \'taskflow_user\'@\'localhost\' IDENTIFIED BY \'taskflow_password\'; GRANT ALL PRIVILEGES ON taskflow_db.* TO \'taskflow_user\'@\'localhost\'; FLUSH PRIVILEGES;`

   Note: You can change the database name, username, and password as per your preference. If you change them, make sure to update the `.env` file accordingly.

2. **Run Migration Script:**

Execute the `taskflow_setup.sql` script to create the necessary tables. You can do this from your terminal:

`bash sudo mysql -u taskflow_user -ptaskflow_password < taskflow_setup.sql` (Enter `taskflow_password` when prompted for the password)

Alternatively, you can copy the content of `taskflow_setup.sql` and run it directly in your MySQL client.

4. Environment Variables

Create a `.env` file in the root directory of the project and add the following environment variables:

```
 DB_HOST=localhost
DB_USER=taskflow_user
DB_PASSWORD=taskflow_password
DB_NAME=taskflow_db
PORT=3001
JWT_SECRET=your_taskflow_jwt_secret_key
```

- `DB_HOST`: Your MySQL host (usually `localhost`).
- `DB_USER`: The MySQL username you created (e.g., `taskflow_user`).
- `DB_PASSWORD`: The password for your MySQL user (e.g., `taskflow_password`).
- `DB_NAME`: The name of your MySQL database (e.g., `taskflow_db`).
- `PORT`: The port on which the server will run (e.g., `3001`).
- `JWT_SECRET`: A strong, random string for signing JWT tokens. Generate a long, complex string for production.

5. Start the Server

```
node server.js
```

The server will start on the port specified in your `.env` file (default: `3001`). You should see a message like `TaskFlow Server running on port 3001` in your console.

API Endpoints

The API endpoints are designed to be RESTful. Below is a summary of the available endpoints and their functionalities.

Authentication

- `POST /api/auth/register`: Register a new user.
- `POST /api/auth/login`: Log in a user and get an authentication token.

Project Management

- `GET /api/projects`: Get all projects.
- `GET /api/projects/:id`: Get a project by ID.
- `POST /api/projects`: Create a new project.
- `PUT /api/projects/:id`: Update a project by ID.
- `DELETE /api/projects/:id`: Delete a project by ID.

Task Management

- `GET /api/tasks`: Get all tasks.
- `GET /api/tasks/:id`: Get a task by ID.
- `POST /api/tasks`: Create a new task.
- `PUT /api/tasks/:id`: Update a task by ID.
- `DELETE /api/tasks/:id`: Delete a task by ID.

Comment Management

- `GET /api/comments/task/:taskId`: Get all comments for a specific task.
- `POST /api/comments`: Create a new comment.
- `PUT /api/comments/:id`: Update a comment by ID.
- `DELETE /api/comments/:id`: Delete a comment by ID.

Notification Management

- `GET /api/notifications/user/:userId`: Get all notifications for a specific user.
- `PUT /api/notifications/mark-read/:id`: Mark a notification as read.
- `DELETE /api/notifications/:id`: Delete a notification by ID.

Real-time Features (Socket.IO)

The TaskFlow backend uses Socket.IO for real-time communication, primarily for notifications and task updates.

Events:

- `connection`: A new client connects to the WebSocket server.
- `disconnect`: A client disconnects from the WebSocket server.
- `taskUpdate`: (Client to Server) When a task is updated, the client can emit this event with the updated task data.
- `taskUpdated`: (Server to Clients) The server broadcasts this event to all connected clients when a `taskUpdate` event is received, allowing all clients to synchronize their task data in real-time.

Example Client-Side Usage (JavaScript):

```javascript
const socket = io("http://localhost:3001"); // Replace with your backend URL

socket.on("connect", () => {
    console.log("Connected to WebSocket server");
});

socket.on("disconnect", () => {
    console.log("Disconnected from WebSocket server");
});

socket.on("taskUpdated", (data) => {
    console.log("Real-time task update received:", data);
    // Update your UI with the new task data
});

// To send a task update from the client:
socket.emit("taskUpdate", { taskId: 1, status: "Completed", title: "Updated
Task" });
```

Next Steps

- Implement more robust logging.
- Add comprehensive unit and integration tests.
- Implement pagination, filtering, and sorting for API endpoints.
- Consider using an ORM (Object-Relational Mapper) like Sequelize for easier database interactions.
- Explore Docker for containerization.