Introduction

Welcome to the presentation of the Personal Task Manager API project.

This project is a Node.js and Express.js-based API that allows users to manage their tasks and categories.

Project Overview

The Personal Task Manager API provides endpoints for user registration, login, and CRUD operations on tasks and categories.

It utilizes MongoDB as the database and Mongoose for object modeling and database interaction.

The API implements JWT-based authentication to secure the endpoints and ensure data privacy.

Project Structure

The project follows a modular structure, separating concerns into different directories:

config: Contains the database configuration file.

controllers: Contains the controller functions for handling API logic.

middlewares: Contains the authentication middleware.

models: Contains the Mongoose models defining the database schemas.

routes: Contains the route files defining the API endpoints.

The app.js file serves as the entry point of the application, setting up the Express server and connecting to the database.

API Endpoints - User Authentication

User Registration:

Endpoint: POST /api/auth/signup

Request Body: { "username": "johndoe", "password": "password123", "email": "johndoe@example.com" }

Response: { "message": "User registered successfully" }

User Login:

Endpoint: POST /api/auth/login

Request Body: { "username": "johndoe", "password": "password123" }

Response: { "token": "<JWT_TOKEN>" }

API Endpoints - Tasks

Create a Task:

Endpoint: POST /api/tasks

Headers: Authorization: Bearer <JWT_TOKEN>

Request Body: { "title": "Sample Task", "description": "This is a sample task", "dueDate": "2023-06-30" }

Response: { "_id": "<TASK_ID>", "title": "Sample Task", ... }

Get All Tasks:

Endpoint: GET /api/tasks

Headers: Authorization: Bearer <JWT_TOKEN>

Response: [{ "_id": "<TASK_ID>", "title": "Sample Task", ... }, ...]

Update a Task:

Endpoint: PATCH /api/tasks/<TASK_ID>

Headers: Authorization: Bearer <JWT_TOKEN>

Request Body: { "title": "Updated Task", "status": "IN_PROGRESS" }

Response: { "_id": "<TASK_ID>", "title": "Updated Task", ... }

Delete a Task:

Endpoint: DELETE /api/tasks/<TASK_ID>

Headers: Authorization: Bearer <JWT_TOKEN>

Response: { "message": "Task deleted successfully" }

API Endpoints - Categories

Create a Category:

Endpoint: POST /api/categories

Headers: Authorization: Bearer <JWT_TOKEN>

Request Body: { "name": "Sample Category", "description": "This is a sample category" }

Response: { "_id": "<CATEGORY_ID>", "name": "Sample Category", ... }

Get All Categories:

Endpoint: GET /api/categories

Headers: Authorization: Bearer <JWT_TOKEN>

Response: [{ "_id": "<CATEGORY_ID>", "name": "Sample Category", ... }, ...]

Update a Category:

Endpoint: PATCH /api/categories/<CATEGORY_ID>

Headers: Authorization: Bearer <JWT_TOKEN>

Request Body: { "name": "Updated Category", "description": "This is an updated category" }

Response: { "_id": "<CATEGORY_ID>", "name": "Updated Category", ... }

Delete a Category:

Endpoint: DELETE /api/categories/<CATEGORY_ID>

Headers: Authorization: Bearer <JWT_TOKEN>

Response: { "message": "Category deleted successfully" }

Code Structure Explanation

The models directory contains the Mongoose models for User, Task, and Category, defining the database schemas and relationships.

The controllers directory contains the controller functions for handling the API endpoints' logic, such as creating, updating, and deleting tasks and categories.

The routes directory contains the route files that define the API endpoints and map them to the corresponding controller functions.

The middlewares directory contains the authentication middleware that verifies the JWT token and attaches the authenticated user's information to the request object.

The app.js file sets up the Express server, connects to the MongoDB database, and defines the API routes.

Error Handling

The API implements error handling middleware to catch and handle any errors that occur during the request processing.

Error responses are sent to the client with appropriate status codes and error messages.

Examples of error handling:

Sending a request with missing required fields will result in a validation error response.

Attempting to access a protected route without a valid JWT token will result in an authentication error response.

Database Interaction

The API interacts with the MongoDB database using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js.

The Mongoose models define the structure and schema of the data stored in the database.

The controller functions use the Mongoose models to perform database operations, such as creating, reading, updating, and deleting documents.

The data is persisted in the MongoDB database, and the API ensures data integrity and consistency.

Demonstration

Live demonstration of the API endpoints using Postman:

User registration and login to obtain a JWT token.

Creating, retrieving, updating, and deleting tasks.

Creating, retrieving, updating, and deleting categories.

Showcasing error handling by sending requests with invalid data or missing authentication.

Explanation of the request headers, request bodies, and response formats for each endpoint.

Conclusion

The Personal Task Manager API provides a robust and secure solution for managing tasks and categories.

It leverages Node.js, Express.js, MongoDB, and Mongoose to create a scalable and efficient API.

The API implements JWT-based authentication to ensure data privacy and security.

The modular project structure and clear separation of concerns make the codebase maintainable and extensible.

The project serves as a solid foundation for building task management applications or integrating task management functionality into existing systems.

Future Enhancements

Potential future enhancements for the project:

Implementing pagination and sorting for retrieving tasks and categories.

Adding user roles and permissions for fine-grained access control.

Integrating with a frontend framework to create a complete task management application.

Implementing real-time updates using technologies like WebSockets or server-sent events.

Thank You

Thank you for attending the presentation of the Personal Task Manager API project.

Feel free to ask any questions or provide feedback.

The project code and documentation are available on GitHub for further reference and exploration.