Task Management System

GitHub Repository

https://github.com/Venkat546/Task-Management-System-Application.git

Introduction

This project is a **Task Management API** designed to handle basic task operations such as creating, updating, retrieving, and deleting tasks. It adheres to the RESTful API design principles, ensuring scalability, maintainability, and ease of use.

Key Features

- Task CRUD (Create, Read, Update, Delete) operations.
- Marking tasks as completed using a PATCH endpoint.
- Comprehensive error handling with user-friendly responses.
- Data validation at the DTO level to ensure only valid inputs are processed.
- In-memory H2 database for ease of setup and testing.
- Automated unit tests to ensure functionality and reliability.

The project showcases modern Java development practices, leveraging the **Spring Boot** framework to build a robust backend application.

Technologies

The project is built using the following technologies:

- **Java 23**: The latest version of Java, providing modern language features and enhancements.

• **Spring Boot 3.3.5**: A powerful framework for building microservices and REST APIs.

• **Spring Data JPA**: Used for data access and persistence with the H2 in-memory database.
• **H2 Database**: A lightweight, in-memory database used for rapid testing and development.• **Lombok**: A Java library that simplifies code with annotations, used for reducing boilerplate in entity and model classes.

• **Maven**: Dependency management and build tool used for packaging the application.

Prerequisites

Before running the application, ensure you have the following tools installed:

- **Java**: Version 23
- **Maven**: Version 3.6
- **Postman**

Dependencies

The following dependencies are included in the project via the pom.xml file:

- **Spring Boot Starter Web**: For building web applications, including REST APIs.
- **Spring Boot Starter Data JPA**: For database integration and data persistence using JPA.
- **H2 Database**: Lightweight, in-memory database for quick development and testing.
- **Spring Boot Starter Validation**: Provides bean validation support to enforce constraints on data.
- **Lombok**: Used for reducing boilerplate code such as getters, setters, and constructors.

- **JUnit 5**: Framework for writing and running unit tests.
- **Mockito**: A mocking framework used for creating mock objects in unit tests.

Steps to Run the Application

1. **Clone the Repository:**

   git clone https://github.com/Venkat546/Task-Management-System-Application.git

2. **Install Required Software:** Ensure Java 23 and Maven are installed. Verify with:
   java -version
   mvn -version

3. **Build the Project:** In the project directory, run:

   mvn clean install

4. **Run the Application:** Run the app using:

   mvn spring-boot:run

5. **Access the Application:** Open a browser and go to:

   http://localhost:8080

6. **Test the API Endpoints:** Use Postman to test the following endpoints:

Method	Endpoint	Description	Sample Payload
GET	/tasks	Fetch all tasks	-
GET	/tasks/{id}	Fetch a task by ID	-
POST	/tasks	Create a new task	{ "title": "Task 1", "description": "Test task", "dueDate": "2024-12-31", "status": "PENDING" }

PUT	/tasks/{id}	Update a task by ID	{ "title": "Updated Task", "description": "Updated Desc", "dueDate": "2024-12-31", "status": "IN_PROGRESS" }
PATCH	/tasks/{id}/complete	Mark task as completed	-
DELETE	/tasks/{id}	Delete a task by ID	-

7. **Access the H2 Database Console:** Go to:

   http://localhost:8080/h2-console

   Use the credentials:
   - JDBC URL: jdbc:h2:mem:testdb
   - Username: sa
   - Password: password

8. **Run Unit Tests (Optional):** Run tests with:

   mvn test

9. **View Logs and Errors:** Check the console for logs and errors.

Automated Testing

1. **Unit Tests** Execute the following command to run all unit tests:

   mvn test

2. **Verify Test Results**

   - Check the output to ensure all tests pass successfully.
   - Tests include:

     - **testCreateTask()**: Validates task creation.
     - **testUpdateTask()**: Ensures task updates work as expected.
     - **testValidation()**: Ensures invalid data is rejected.

Notes

 Error Handling: If you provide invalid data, the API will return a 400 Bad Request response with error details.

 Default Database: The application uses an H2 in-memory database for simplicity. Data will reset on each restart.

Design Decisions

1. **Framework Selection**:
    - **Spring Boot**: Chosen for its ease of use, scalability, and ability to quickly build RESTful APIs. It simplifies dependency management and reduces boilerplate code.
2. **Database Choice**:
    - **H2 In-Memory Database**: Selected for its lightweight nature and ease of setup, making it ideal for testing and development purposes.
    - Schema management via **JPA annotations**, ensuring automatic table creation and updates.
3. **Layered Architecture**:
    - **Controller**: Handles API requests and sends appropriate responses.
    - **Service**: Contains business logic to process task-related operations.
    - **Repository**: Interfaces with the database using Spring Data JPA.
    - **Model**: Represents the Task entity.
    - **DTO**: Separates input validation and data transfer from business logic.
4. **Validation**:
    - Implemented **DTO-based validation** using jakarta.validation annotations to ensure valid user inputs.

- Clear error messages returned for invalid requests to improve user experience.

5. **Error Handling**:
    - **Custom Exception Handling** with TaskNotFoundException for specific errors.
    - A centralized GlobalExceptionHandler provides structured and user-friendly error responses.
6. **Testing**:
    - Automated unit tests using **JUnit 5** and **Mockito** to validate core functionalities like task creation, updates, and error scenarios.
    - Manual testing facilitated with tools like **Postman** and the **H2 console**.
7. **Scalability**:
    - Adhering to RESTful API principles ensures the API can be easily scaled and integrated with other systems.
    - Modular design makes it simple to extend functionalities, such as adding authentication.

Assumptions

1. **User Context**:
    - The application assumes a single-user context, with no multi-user or authentication requirements implemented.
2. **Task Management Scope**:
    - Tasks are self-contained entities without any dependencies on external systems.
    - No categorization, prioritization, or user assignment for tasks in the current version.

3. **Error Scenarios**:

    - The API expects valid input and provides structured responses for invalid data. For example, the status must be one of PENDING, IN_PROGRESS, or COMPLETED.

4. **Database Behavior**:

    - The H2 database is reset with every application restart, so tasks do not persist across sessions.

5. **Environment**:

    - The application is designed to run on a local development environment.
    - Deployment configurations (e.g., Docker, cloud hosting) are not included in the current scope.

6. **Time Constraints**:

    - No detailed logging or audit trails implemented due to the project’s simplicity.
    - Dates and times are assumed to be in the local server's timezone.