Project documentation

Table of contents

Introduction

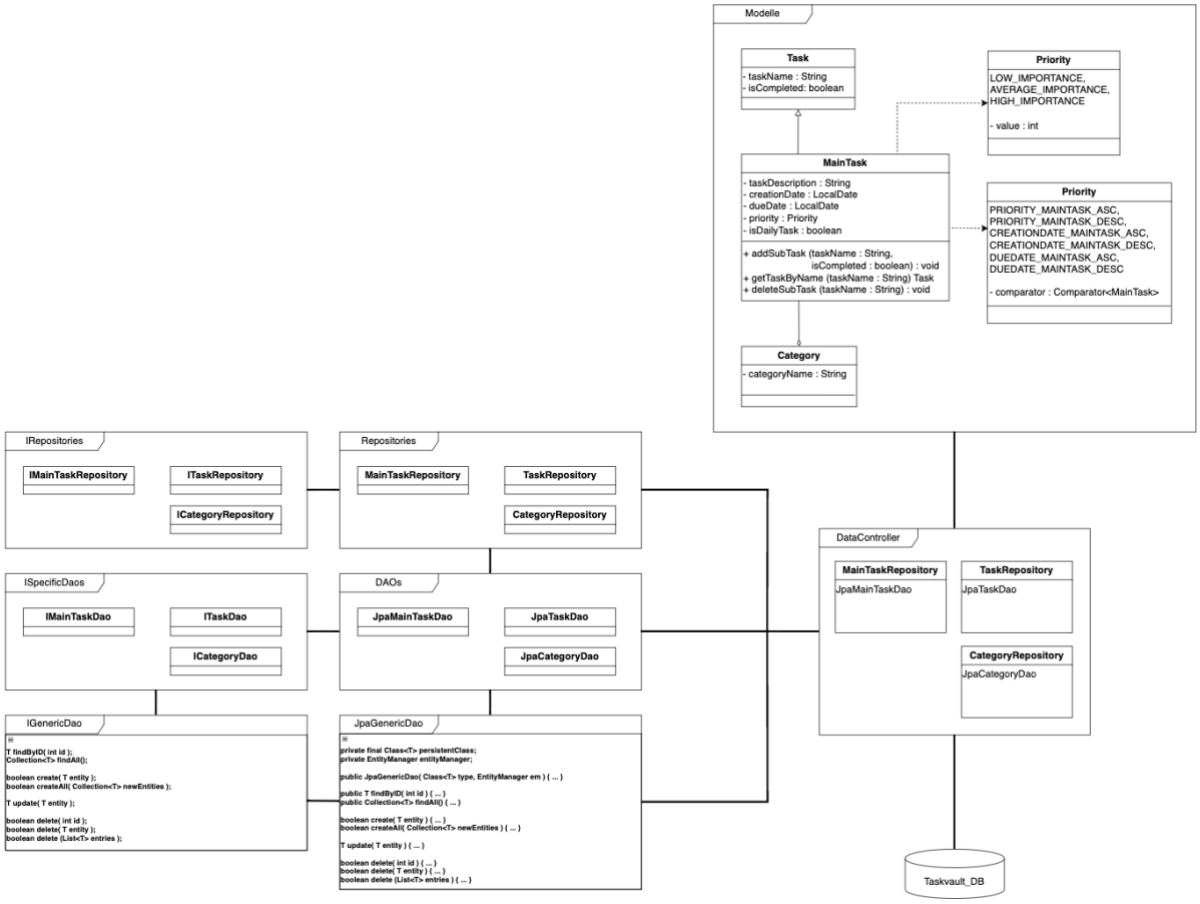
In today's fast-paced world, staying organized and keeping track of tasks is essential to productivity and success. JavaTaskVault is a task management tool that helps its users achieve this. It allows users to efficiently manage tasks and keep track of their to-do list. Larger tasks can be easily divided by subtasks, allowing for a clearer processing of tasks. Different topics? No problem, categories can be used to create different tasks so that you can devote yourself entirely to one task and its subtasks. This allows several tasks to be entered in one category, and the tasks themselves can then be processed one by one using subtasks. Recurring tasks can be processed with the DailyTasks; these are repeated daily and make it easier to build up habits.

Group Responsibilities

Storage - Silas Falke
API - Luise Brumme
Tests - Henri Luli

UML-Diagram

The attached UML diagram shows the current class structure of the application and the relationships between the various components. It illustrates the most important classes and their interactions within the application. This provides a visual representation of the architecture and helps to better understand the structure and data flow within the application.



Implementation

Changes and design decisions

As part of the current development phase, significant changes have been made to the structure and implementation of the Java Task Vault project. A central change concerns the Vault class, which has been completely removed. Originally, this class was used to manage multiple users and securely store their tasks. However, since it was decided not to implement user management in this version, the need for a Vault class became obsolete.

Another significant difference is the removal of the Categories class. Previously, this class was responsible for storing and managing all instances of categories in a separate list. However, with the introduction of a database, the need to manually store these categories in a list has been removed. Instead, the categories are now stored and managed directly in the database.

These changes reflect the transition to a database-driven architecture, which offers improved data consistency and scalability. The decision not to implement user management has also reduced the complexity of the project, making it easier to develop and maintain.

Architecture and structure

The project is divided into three main modules:

api

This module contains the API resources and services that form the heart of the application. It contains the endpoints that can be used to perform various CRUD operations on the tasks and categories. The API is responsible for accepting HTTP requests and returning corresponding responses based on the results of the database operations.

storage

In this module, the focus is on the storage and management of data. The various database daos (Data Access Objects) and repositories that ensure communication with the database are defined here. This layer is crucial for persisting and managing the application data.

test

The test modules contain unit and integration tests that ensure that the various components of the application function correctly. The tests cover both the business logic and the data access layer and thus provide a check of the application.

The combination of these three modules makes it possible to achieve a clean separation of responsibilities and increase the maintainability of the code.

Known Bugs

500 status code in the updateMainTask API function: A problem has been identified where the updateMainTask API function returns an HTTP 500 error even though the corresponding changes have been made in the database.

How to Run

To run the project, please follow the steps below:

1. Make sure that Docker is installed and configured on your system.
2. Open a terminal and navigate tot he directory where the project is located.
3. Execute the following command to create and start the necessary Docker containers:

   Command: „docker-compose up -d“

   This command builds the Docker images and starts the containers in the background.
4. Once the containers are running, start the web application by executing the WebApplication

   „src/main/java/de/fherfurt/taskvault/api/WebApplication.java“ – File.
5. To stop the container and remove all associated resources:

   Command: „docker-compose down“

   This command terminates all running containers and removes the created networks and volumes.