Java3-team1-service3
Faculties

Intro

Hello and welcome to our documentation of the service Faculty. This service supplies faculty-related data to students and lecturers and functionality to facilitate information finding. For this, we have working server functionalities that reflect the standard database queries and functions for more specified use-cases.

The project is developed by

- Christopher Orth

- Falko Kühn

- Marvin Pohl

- Niklas Herzog

- Sarah Sharafat

It is created as an exam for the module Programming Java 2 of the course applied computer science. This project uses:

- Docker

- Java 11

- Maven

- Junit 5

- AssertJ

- H2 Database

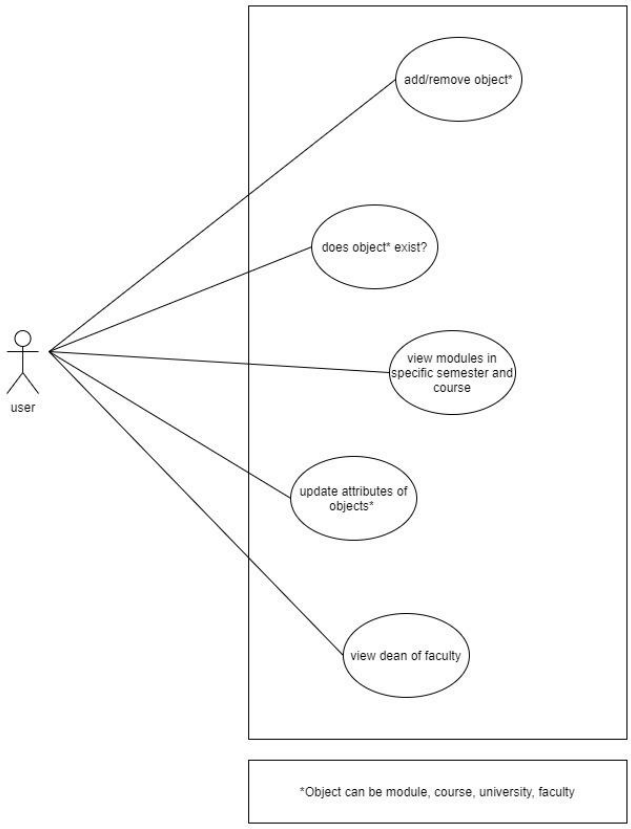
- Hibernate

- Jersey RESTful

- Jetty

- Jackson for JSON parsing

In the following you can see the use-case-diagram which shows a selection of functionalities, which will be provided by the final micro-service:



Architecture

Packages

The Faculties-Project's package name is de.fherfurt.faculty and all of the functionality is stored inside the „data" Package which is a remnant from out thought process while working on the project for the 1st Java-Module we had. The actual logic of the project is divided into the following packages:

- classes
- repository
- server

Each one has unique functionality. In general the „server" package might be closest to being the presentation layer if we think of the project as a layered application. If we continue that train of

thought the „repository“ package is the data access layer and also includes a bit of business logic, whilst „classes“ mainly supplies the data model for the application.

Classes

Here the data model is described by the four classes „University“, „Faculty“, „Course“ and „Module“ which is expanded by enums, which are stored in the corresponding sub-package. The mentioned classes are also displayed with their relationships at the bottom of this page.

Repository

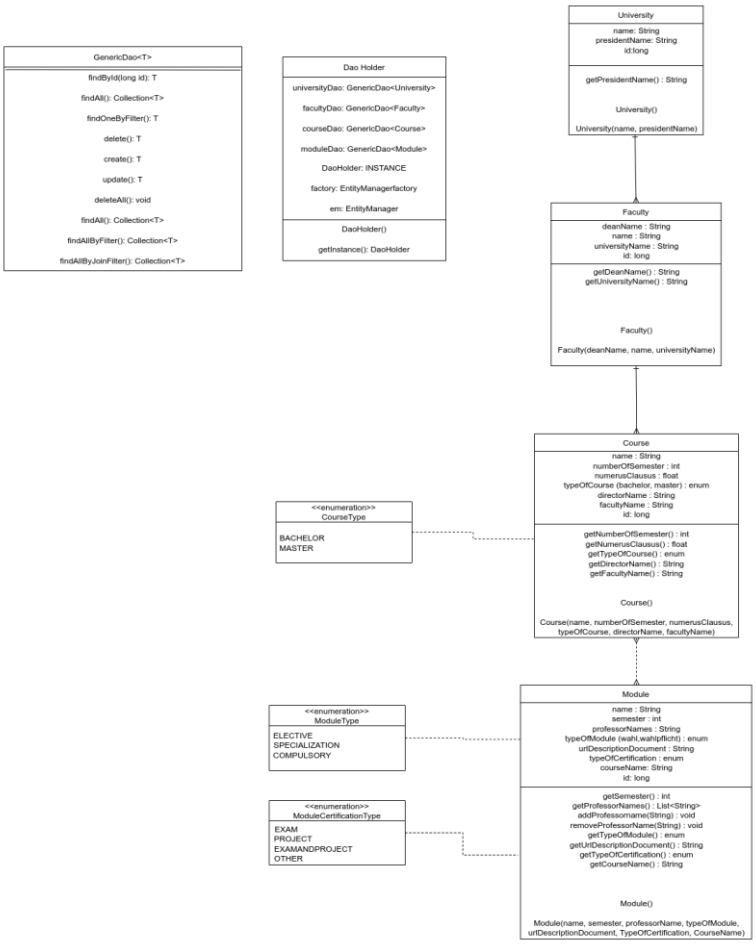
In the „repository“ Package there is logic to access data in a meriad of ways. It contains the sub-package „core“ which included our Generic Dao which included the most important CRUD operations like „findByID“ or „Delete“ and also some more exotic functionalities like „findAllByFilter“ which wraps a Query for usage as a normal function, that makes it possible to just call the function with the name of a piece of data and a value and it gives back all the values that match the criteria. There is also „findAllByJoinFilter“, which adds a Join to this, so a set of data can be searched across classes.

Outside the „core“ package there is the Class DaoHolder. It instantiates the Generic DAO for the 4 data classes and also includes EntityManager. It is realised in a Singleton Pattern, so there can only be one entrypoint to access the data. Also there are 4 other classes for providing additional functionality and access to specific pieces of data or the ability to change specific pieces of data for the 4 types of data in the database. Most of get passed a ID of the specific set of data they are supposed to be targeting as either a string or a long. This is because the general query wrappers „findAllByFilter“ and „findAllByJoinFilter“ take Strings as attributes and the other specific one take „long“s because that's the data type the IDs have natively.

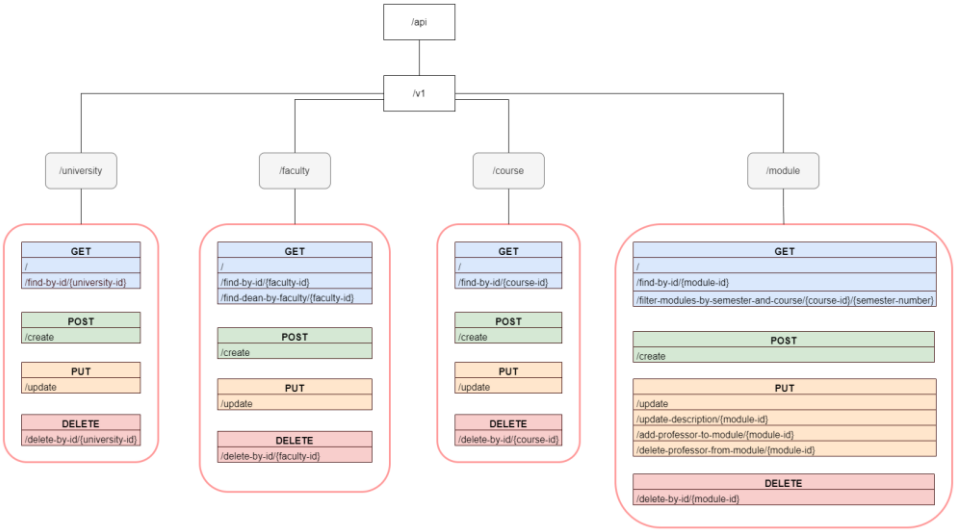
Server

In the main „server“ package there is the class „Backend“, which includes code to start a Jetty Server and load the REST-Configuration stored in the accordingly named class. The „REST-Configuration“ however accesses the 4 resource-classes located in the Package „resources“. They define the specific endpoints for the API of the Application. They are created to make functions from both the Generic DAO and also the function-classes in the Package „repository“ accessible by clients. Since the aim of the project is to create a backend for a mobile application managing the data for the „FH-Erfurt“ the functions are oriented on editing and accessing what such a application might need. Also worth noting is the Class „TestData“ in „resources“. This class includes test data that we also used in the unit-tests for the project and we included it here to be able to test the API endpoints with data. To do this we called getTestData in the TestData class, which created the data in the database so it would be accessible when calling the API. The change this behaviour one can simply change the value of the variable „useTestData“ to false.

Class diagram for classes and repository



Web Server REST Endpoints



The POST Endpoints require complete Objects (with all attributes) of their respective Class. The same is the case for the "/update" paths.