Project documentation java1-team1-service3

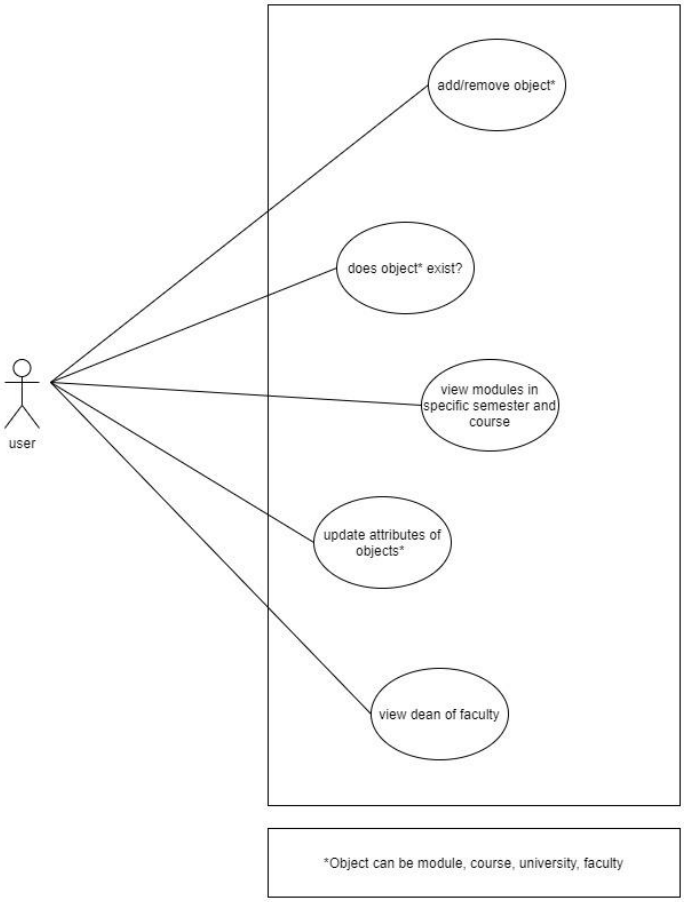
Hello and welcome to our documentation of the service **Faculty**. This service supplies faculty-related data to students and lecturers and functionality to facilitate information finding. The project is developed by

- Christopher Orth (Dev Lead)
- Falko Kühn
- Marvin Pohl
- Niklas Herzog
- Sarah Sharafat

It is created as an exam for the module **Programming Java 1** of the course **applied computer science**. This project uses

- Java 17
- Maven
- Junit 5
- AssertJ

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:



Modules

- The Project contains two Modules. The first and main module is the `faculty-service` module. It contains the logic of our Service. The `faculty-client` module contains interfaces for extern services needing our functionalities.

Classes

- To represent the data, we created classes for Courses, Modules, Faculties, and Universities. Each class contains a unique `name`-property which is ensured by them extending the `Basic`-class. The remaining properties are adapted to the special requirements of the respective class. In addition, this is where the `set`- and `get`- functions for each property can be found.

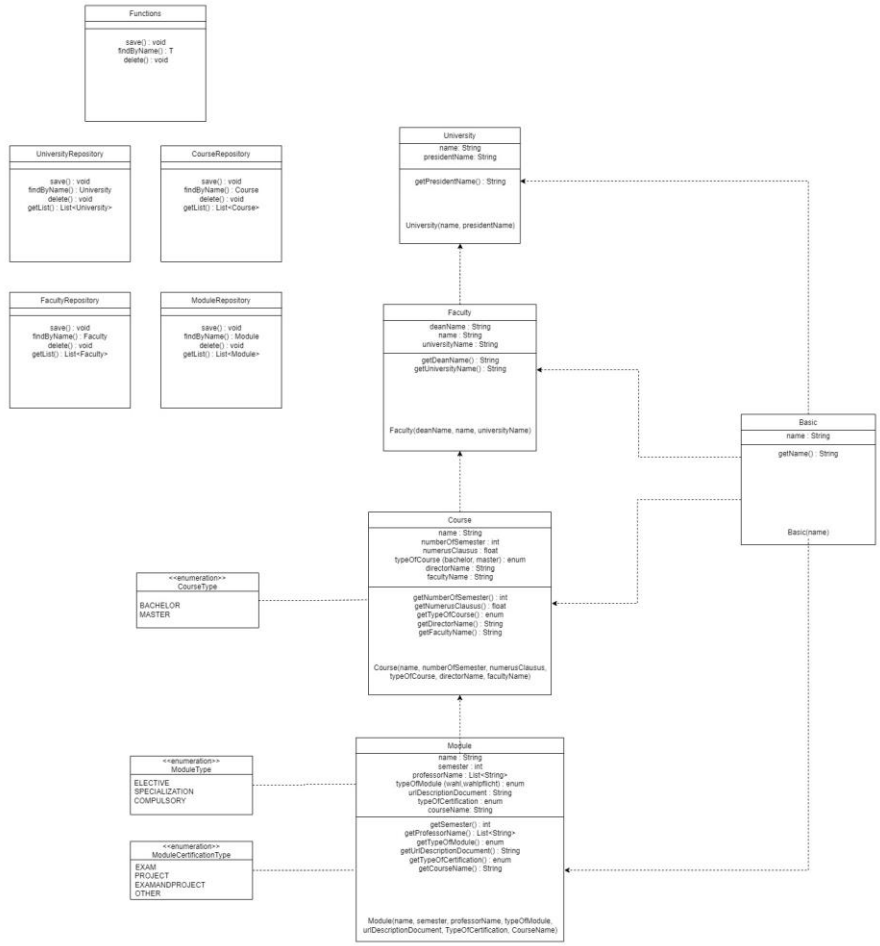
Enums

- In some cases, the properties of a class needed options as values. This led to the creation of enums. With `CourseType` objects of the class course can be separated into groups, based on the kind of degree that may be obtained upon graduation. At this point in time, this enum only comprises `BACHELOR` and `MASTER` as valid options, but can be easily extended in the future to i.e. include more specialized types of degrees such as `BACHELOR OF SCIENCE` and the like. The enum `ModuleCetificationType` contains the different types of examinations necessary to pass a certain module. The current options provided by this enum are `EXAM`, `PROJECT`, `EXAMPROJECT` and `OTHER`. Another enum required by the module class is `ModuleType`. Other than the previous one, this enum can be used to specify, whether a certain module is `ELECTIVE`, `SPECILIZATION` OR `COMPULSORY`.

Core and other Class-Files

- Since each object needs a unique identifier, we decided to give each class a `name`-property, which contains a unique name. To make sure, every class has its `name`-property, we created a `Basic`-class which is extended by all the other classes. Since every `Module` belongs to a `Course`, the `Module`-class has a `courseName`-property that can be used to find its `Course`-object. This also applies to the relations of `Course` to `Faculty` and `Faculty` to `University`.

The following diagram shows the relations between the classes we have designed.



The Repository-Infrastructure

The Repository-Infrastructure is at the heart of this microservice. It is the place where the imitation of the database functionality is happening, beginning with the package de.fherfurt.faculty.data.repository.core.

de.fherfurt.faculty.data.repository.core

- This package delivers the core operations necessary for how this service manages data. It is oriented on database operations. The 3 functionalities given here are SAVE, DELETE and FINDBYNAME. SAVE is used to either update a set of data or to insert it in a List (which we will get to right after i named the other functionalities), DELETE to remove an existing set of Data from a List and FINDBYNAME to identify and return a set of data from a List. So SAVE is a placeholder for UPDATE and INSERT, DELETE for DELETE and FINDBYNAME for SELECT.
- As the defining Attribute of a set of data we decided on the Name, since this is unique, and introducing an ID would have only complicated the functions unnecessarily. Although this is the case it should be kept in mind, that this reliance on the name can be changed rather flexibly with little changes in the code, should it be desired by later maintainers of this service. This might be necessary for the future when a database or other external data management would be introduced.
- We are using Array Lists to manage data because the Java API provides useful functionalities for it that otherwise might have been needed to be written manually. We also see it as a safe means to keep the data.
- These functions are written to be as general as possible to avoid redundancies. They all receive various parameters. Amongst these, there are 2 parameters that are universal to the

functionalities. A Data Type T and an Array List with Elements of the type T. There is also always one additional parameter. Since the functions all resemble a certain operation, the list is the target of these operations and the other parameter is the piece of data used to invoke the operation. For example, on `findByName` we get a List from which the name given in the other parameter is to be found and the associated set of data is to be returned. The Type T exists so these functionalities can be used for every data type and thus for every one of the lists. This is possible cause the functionalities have been kept general enough to not depend upon the other various attributes of the complex types defined by our data model.

de.fherfurt.faculty.data.repository

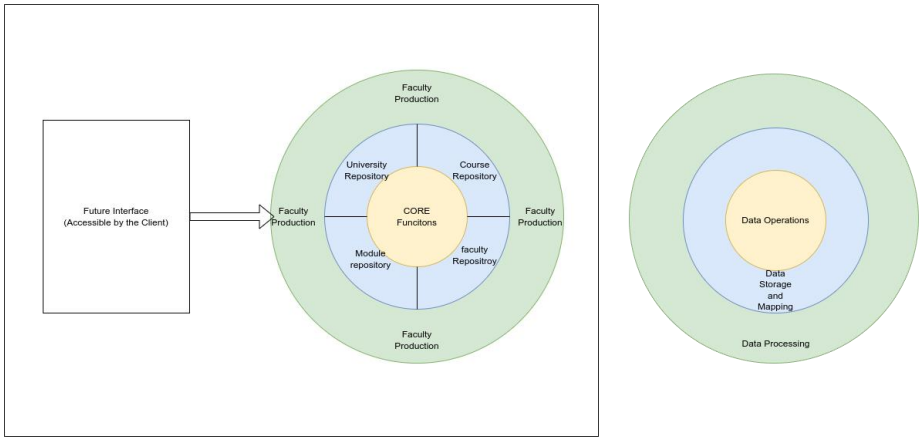
- In the respective classes found in this layer, there is a List used for storage of the data this service intends to manage and also functions to map the functions found in the before discussed core to the data type this class manages. There is one class for every data type to be managed and every one of those will have an associated database table or other pool of storage in the case this service gets extended for the usage of such.

de.fherfurt.faculty.data.FacultyProduction.java

- These repository classes, that instantiate the functions class get instantiated themselves in the `FacultyProduction` class in the `data` package.
- These instances or objects of the repository classes are included as attributes and the class also provides functions useful for managing and reading the data from the point of view of a client. In the future there will be interfaces making these functions accessible from actual clients.
- In these here defined functions the underlying functions of SAVE, `DELETE` and `FINDBYNAME` can be called by usage of the Repository objects included. The `FINDBYNAME` functionality also returns a full object, making it possible access getters and setters of the data model in this layer, making it possible to edit and read singular values of data sets as well. Since Java handles Lists and Complex Data types by references only it is possible to edit all the data sets without it being necessary to explicitly overwrite a List after editing it in one of the functions (e.g. SAVE).

Conclusion

- In the `de.fherfurt.faculty.data.repository.core` the functions providing the core functionalities of data management on a database-level are written.
- In the `de.fherfurt.faculty.data.repository` the Lists holding the information are attributed to classes specialized on a data type and the core functionalities above are mapped to these specific data types.
- In the `de.fherfurt.faculty.data` the class `FacultyProduction` holds instances of the aforementioned Repository Classes and provides functionality to access and manage the data while using the underlying functions provided by the repositories or data types themselves.
- All of this is designed to provide all the necessary functionality for later frontends or communication with other services, while being as easy to maintain and efficient as possible.



Test

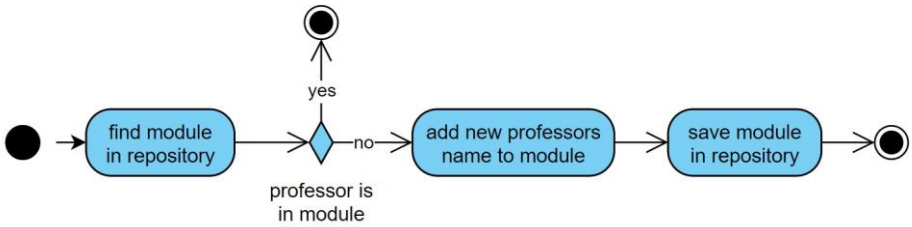
Test Data

- TestData serves as a substitute for a database. It contains two example-objects for each class, which can be used to run tests.

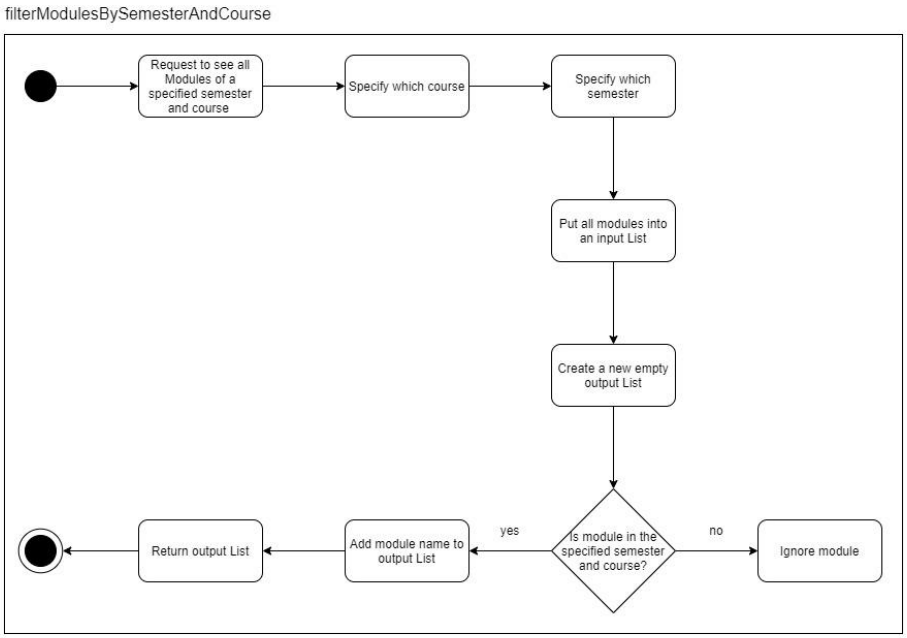
FacultyProductionTest

- This is the designated class for Junit-testing of the functions defined in `FacultyProduction`.

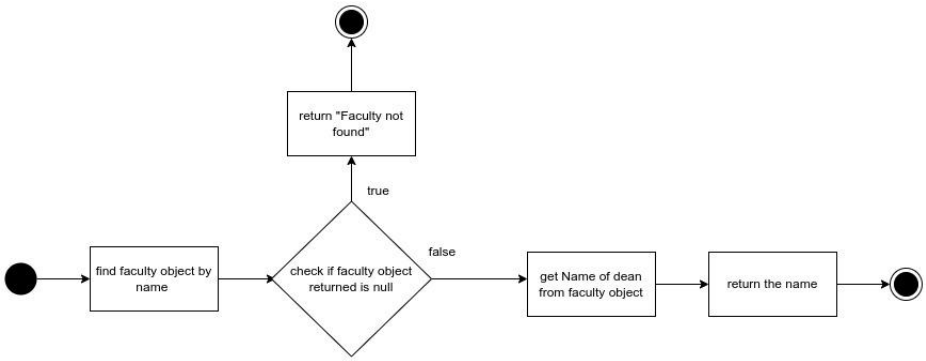
addProfessorToModule



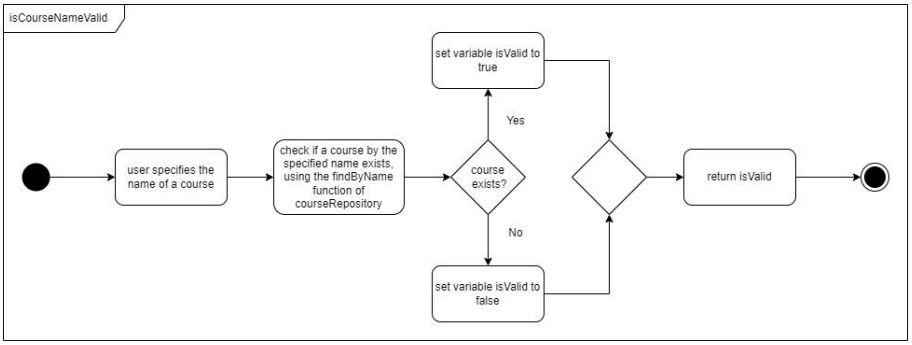
filterModulesBySemesterAndCourse



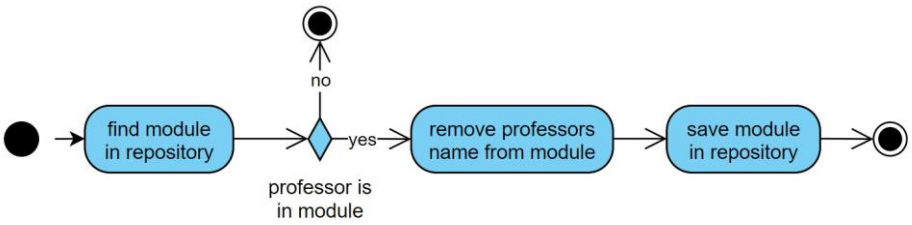
getDeanByFaculty



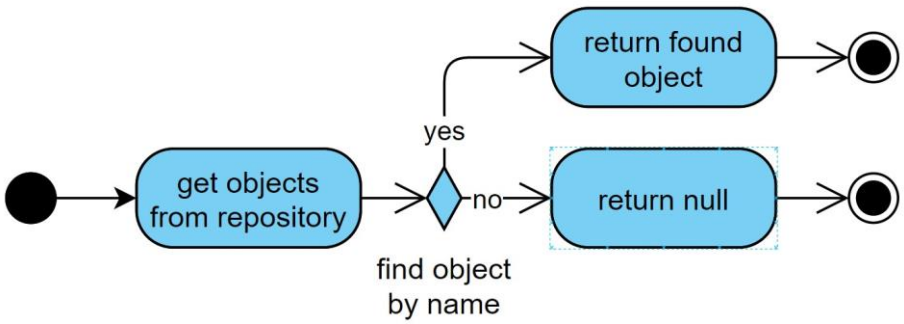
isNameValid



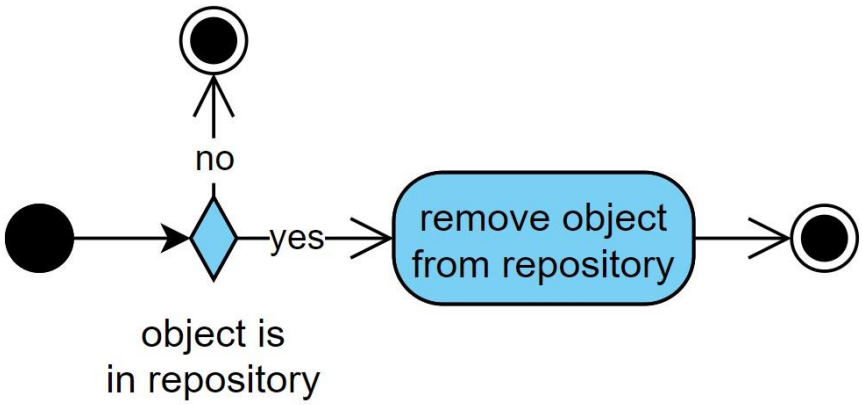
removeProfessorFromModule



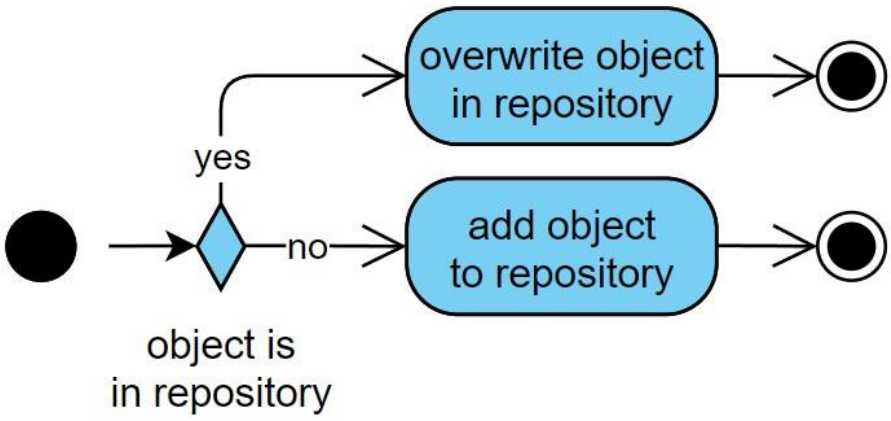
repositoryFindByName



repositoryRemove



repositorySave



updateDescriptionDocument