Laboratory activity tracking API

Student:Andrei Baraian

**Group:30431**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The assignment requested to implement an application for the tracking of laboratory activity for the Software Design laboratory, but with the possibility of extending it for more types of laboratories.

# Functional Requirements

The application should have two types of users: student and teacher, which have to authenticate with an email and password.

The teacher can create an account for a student. When created, a 128-bit token is generated and he has to manually send this token to the student’s email, so that he can register. He can also perform CRUD on students, on laboratories, on attendances and on assignments. Some laboratories will have assignments, for which we have to track the name, deadline, and a long description with the assignment text. The teacher can grade the submitted assignments (also regrade them) and get a list of grades for a particular assignment.

For a student, we have to track the email address, full name, group and top 1 hobby. He can perform the following operations: register using the token generated by the teacher, view a list of laboratory classes and also a filtered list of laboratories based on a keyword, which will be searched in the curricula and in the description. He can view the assignments for a laboratory (there can be more assignments for a lab) and he can create a submission for an assignment, where he should insert a git repository link and a short remark. To be noted, he can submit three times for an assignment.

# Non-functional Requirements

*Availability*

The application should be available for most of the time after it is up on the market. The maintenance time should be reduced as much as possible. This can be enforced by having a good design which can easily respond to new requests or solve bugs.

*Security*

The security of the app consists of having different access rights for the administrator and normal user (cashier in our case). This is implemented by offering different functionalities based on the login user. Another security feature is to store the password in an encrypted form, using a one-way encryption algorithm. I have used SHA-256, provided by the MessageDigest class in Java. Message digests are secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value.

*Testability*

I provided unit tests for one of the Controller classes in the project.

*Design Constraints*

The implementation language is Java. The database is MySQL. For the connection to the database, I have used Hibernate ORM. Also, I used dependency injection to inject Services in Controllers and Repositories in Services.

The architecture of the project is Layer Architecture. The software is divided in 4 layers: Presentation, Service, Repository, Unit Tests. The Service and Repository layers have each their own model.

2. Use-Case Model

Use case: Create submission

Level: user-goal level

Primary actor: student

Main success scenario: successfully create a submission for an assignment

Extensions: fail to create a submission for an assignment, either by selecting the wrong assignment id, or by exceeding the maximum number of submission per assignment.

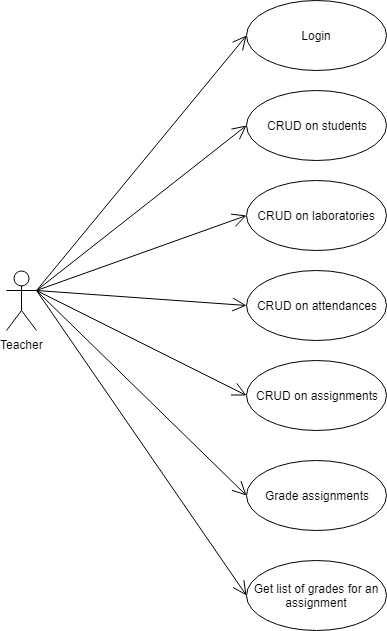
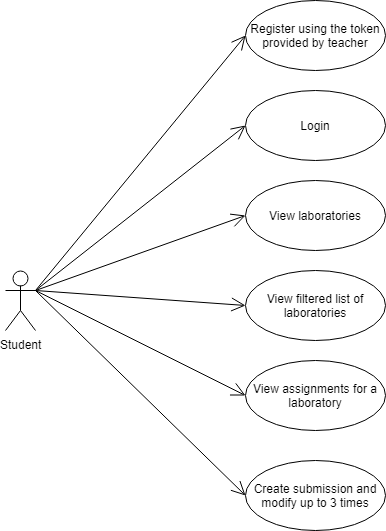
Use case: Create student

Level: user-goal level

Primary actor: teacher

Main success scenario: successfully create an account for a student and generate a unique token

Extensions: fail to persist the database with the student account or generate an already present token.



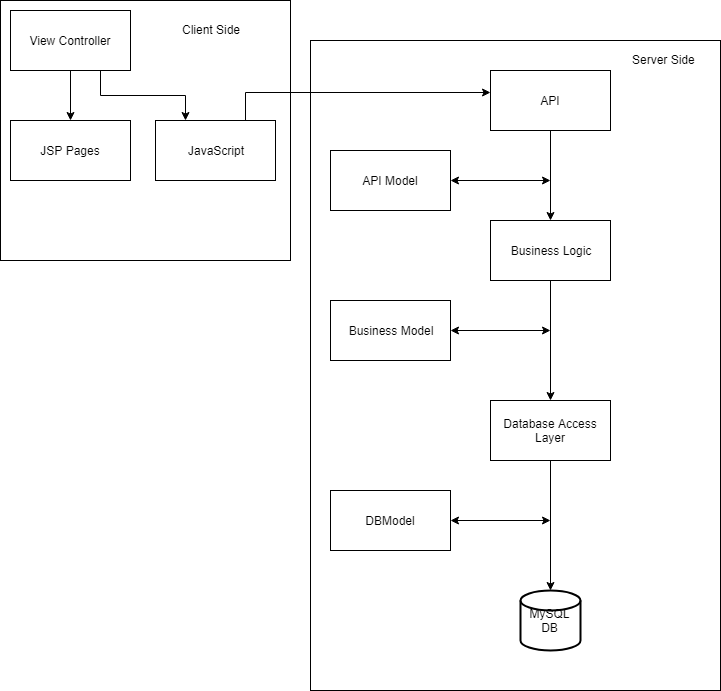
3. System Architectural Design

**3.1 Architectural Pattern Description**

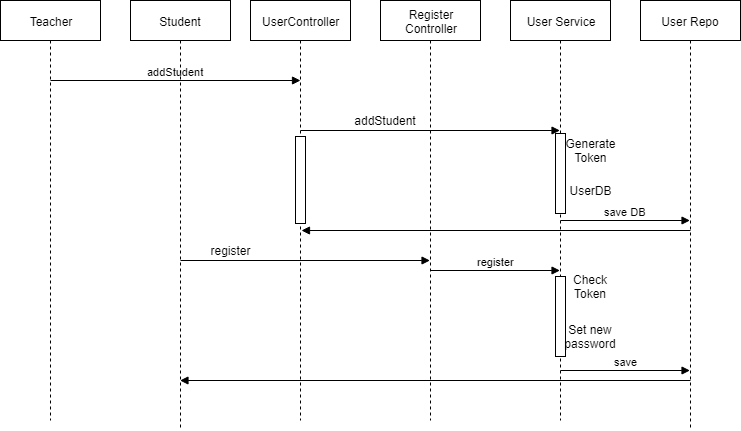
For this project, I have implemented the MVC architectural pattern. It separates the application in three major types of components: Models, Views and Controllers. It facilitates maintenance and portability and same information is capable of presentation in different formats in different windows.

Along MVC, I have used the Layered Architectural Pattern for better separation of responsibilities.

**3.2 Diagrams**



4. UML Sequence Diagrams



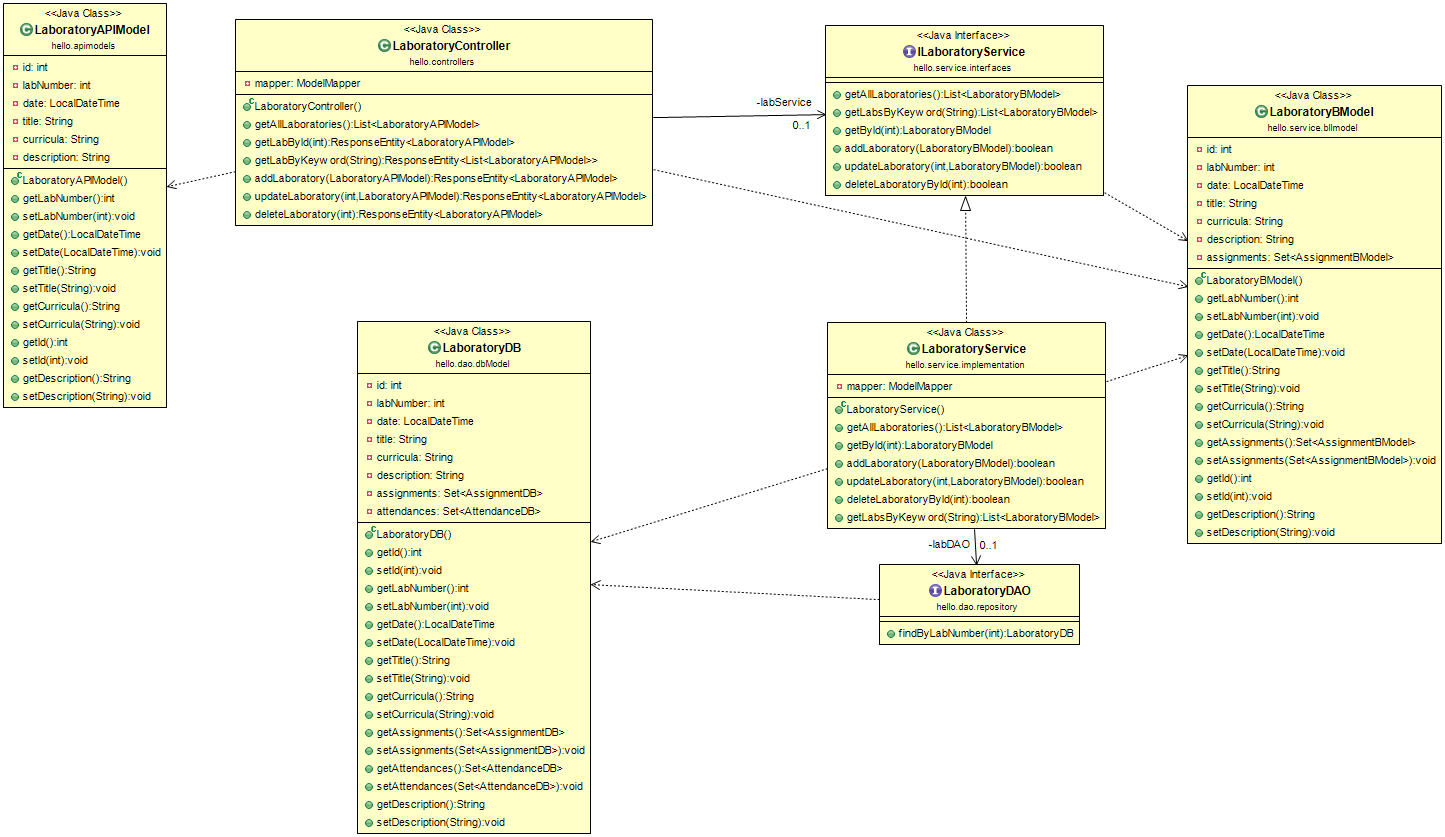
5. Class Design

**5.1 Design Patterns Description**

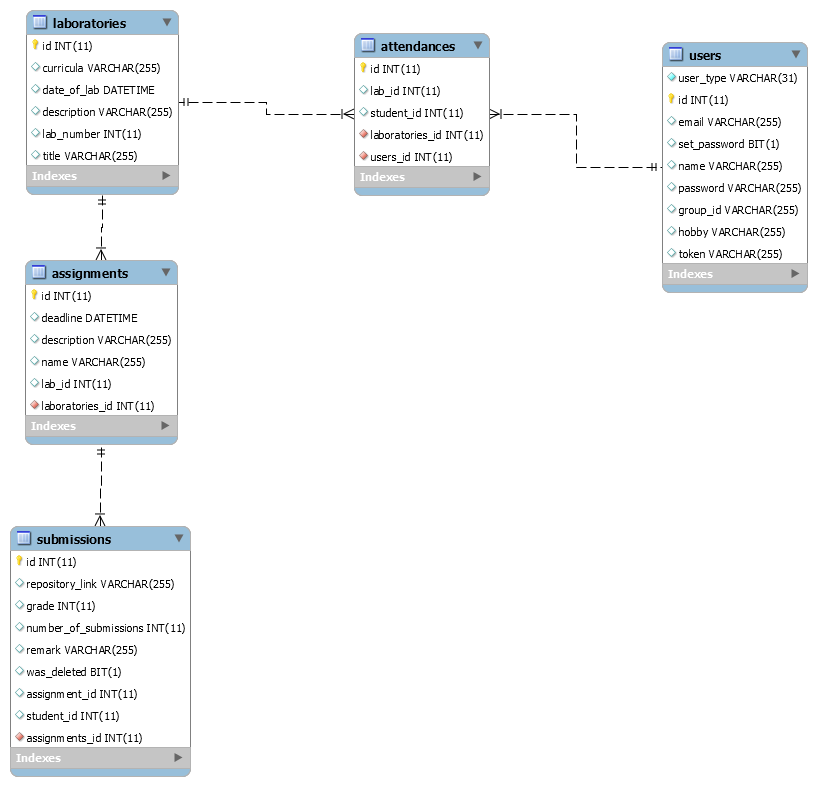
Regarding Design Patterns, I have used Dependency Injection. Java components/classes should be as independent as possible of each other. This increase the possibility to reuse them and to test in isolation. To decouple classes from each other the dependency to a certain class should get injected into them rather than that class itself creates that object.

In the Spring framework, this is done using the @Autowired annotation.

**5.2 UML Class Diagram**



6. Data Model

**

7. System Testing

In order to test the application, I have used Unit testing for the laboratory controller class. For this, I have mocked the service and I verified the response entities that I got back (the http codes and the json’s returned).

8. Bibliography

<http://www.vogella.com/tutorials/SpringDependencyInjection/article.html>

<http://www.baeldung.com/spring-boot-testing>

<https://spring.io/guides/tutorials/bookmarks/>