<Laboratory Management>

Analysis and Design Document

Student: Cârcu Bogdan

**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

Use JAVA Spring/C# Web API to design and implement an application for the tracking the laboratory activity for the Software Design laboratory. The application should have two types of users (student and teacher) which must provide an email and a password to use the application.

# Functional Requirements

The **teache**r can perform the following operations:

- Login

- CRUD on students. When you create a student, a 128 characters token is created. Using that token student should be able to register. Teacher will send the token by email manually. For each student we should track: email address, full name, group (ex. 30431) and top 1 hobby.

- Can add/edit/delete Laboratory classes. For each class we should track: laboratory number (1-14), date, title, curricula for what are the topics presented in that lab and a long description with the laboratory text (should accept html).

- CRUD on attendance for each lab.

- CRUD on assignments. Some of the laboratory will have assignments: for each assignment we must track the name, deadline and a long description with the assignment text.

- Grade the submitted assignments. It should also be possible to regrade the assignment.

- Get the list of grades for all students for a given assignment.

The **student** can perform the following operations:

- Register using the token generated by the teacher.

- Login with the username and password.

- View a list of laboratory classes. Also view a filtered list: student inserts a keyword and that keyword is searched in the curricula and long description.

- View the assignments for a laboratory class.

- Create an assignment submission. Here, students should be able to insert a link to a git repository and a short remark for the teacher.

# Non-functional Requirements

The data is stored in a **database**.

• MVC architectural pattern to organize your application.

• API design is RESTful.

• ORM (Hibernate / Entity framework) to access the database

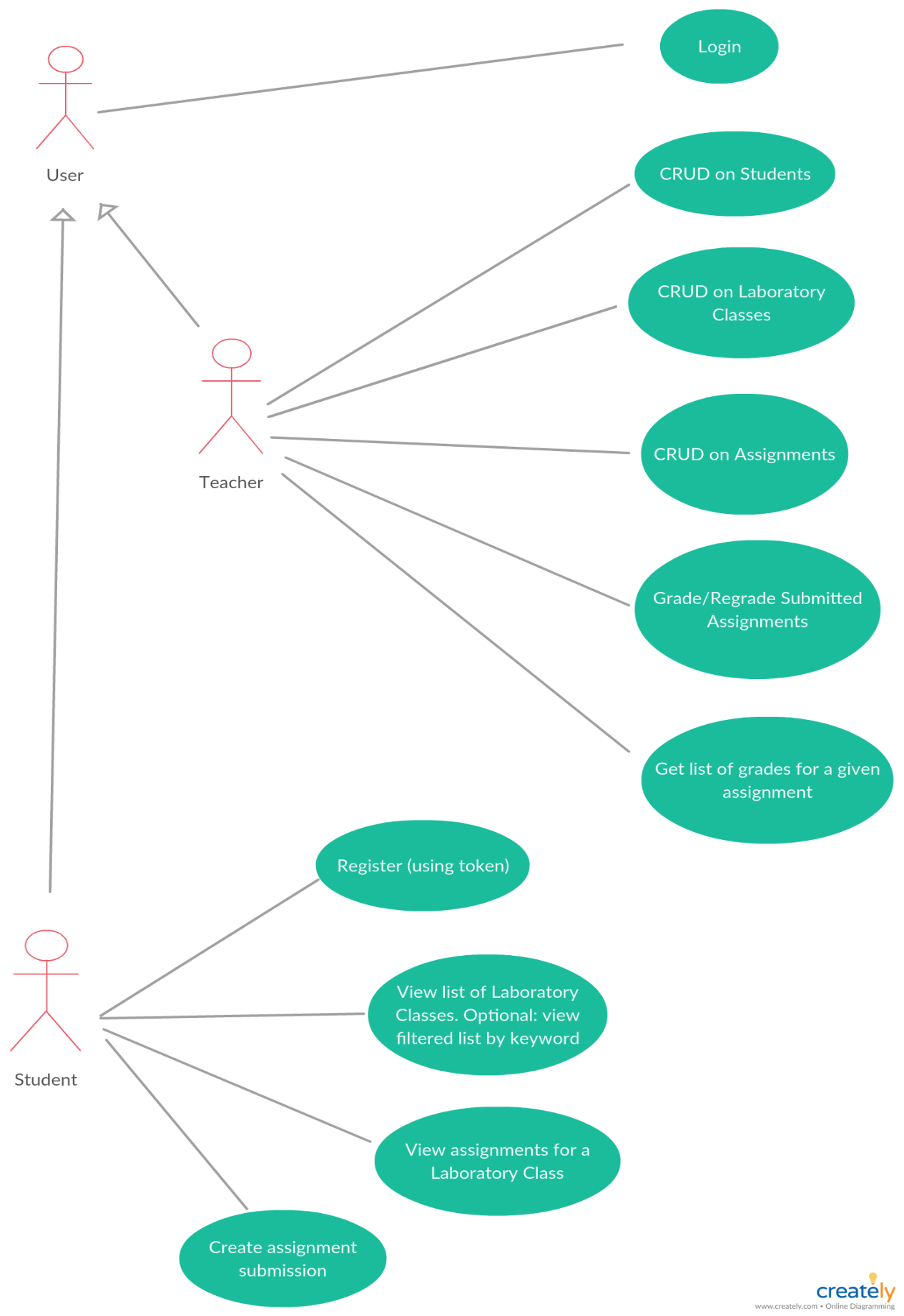
• Dependency injection to inject Services in Controllers and Repositories in Services

• Swagger to call your APIs.

• Connection string should be stored in a separate config file

• Create one set of API tests for one of the controllers

2. Use-Case Model



Use case: System usage

Level: user-goal level

Primary actors: User, Teacher, Student

Main success scenario: all actions are performed successfully

Extensions: no action performed in case of failure

3. System Architectural Design

**3.1 Architectural Pattern Description**

**MVC**

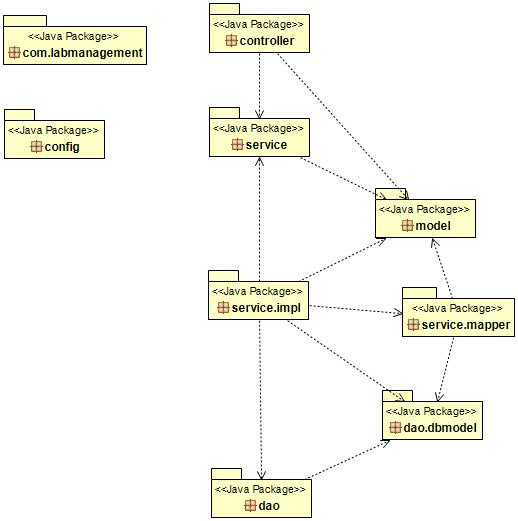
**Model–view–controller** (**MVC**) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

**Components**

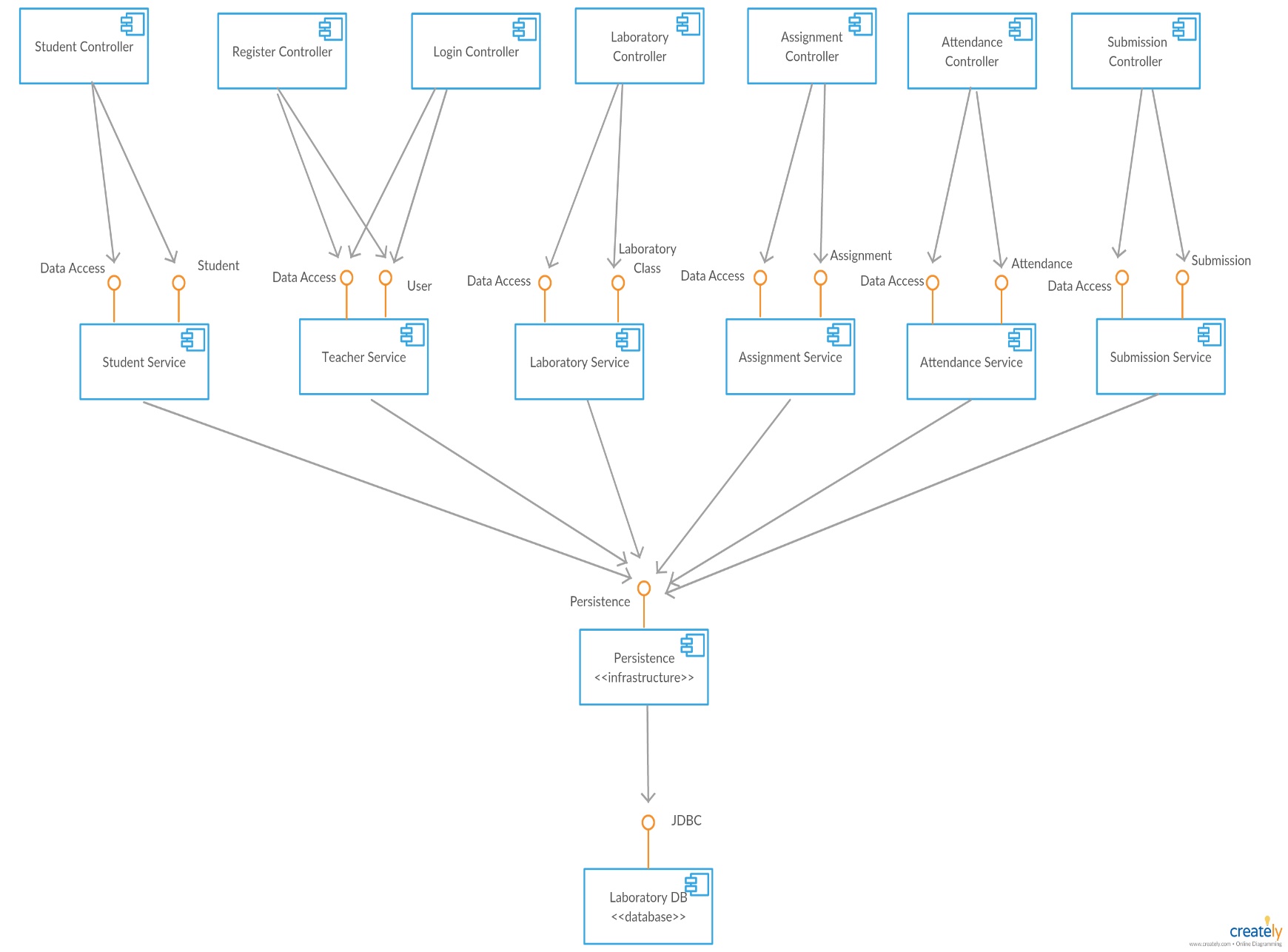
* The *model* is the central component of the pattern. It expresses the application's behavior in terms of the [problem domain](https://en.wikipedia.org/wiki/Problem_domain), independent of the user interface. It directly manages the data, logic and rules of the application.
* A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible.
* The third part or section, the *controller*, accepts input and converts it to commands for the model or view.

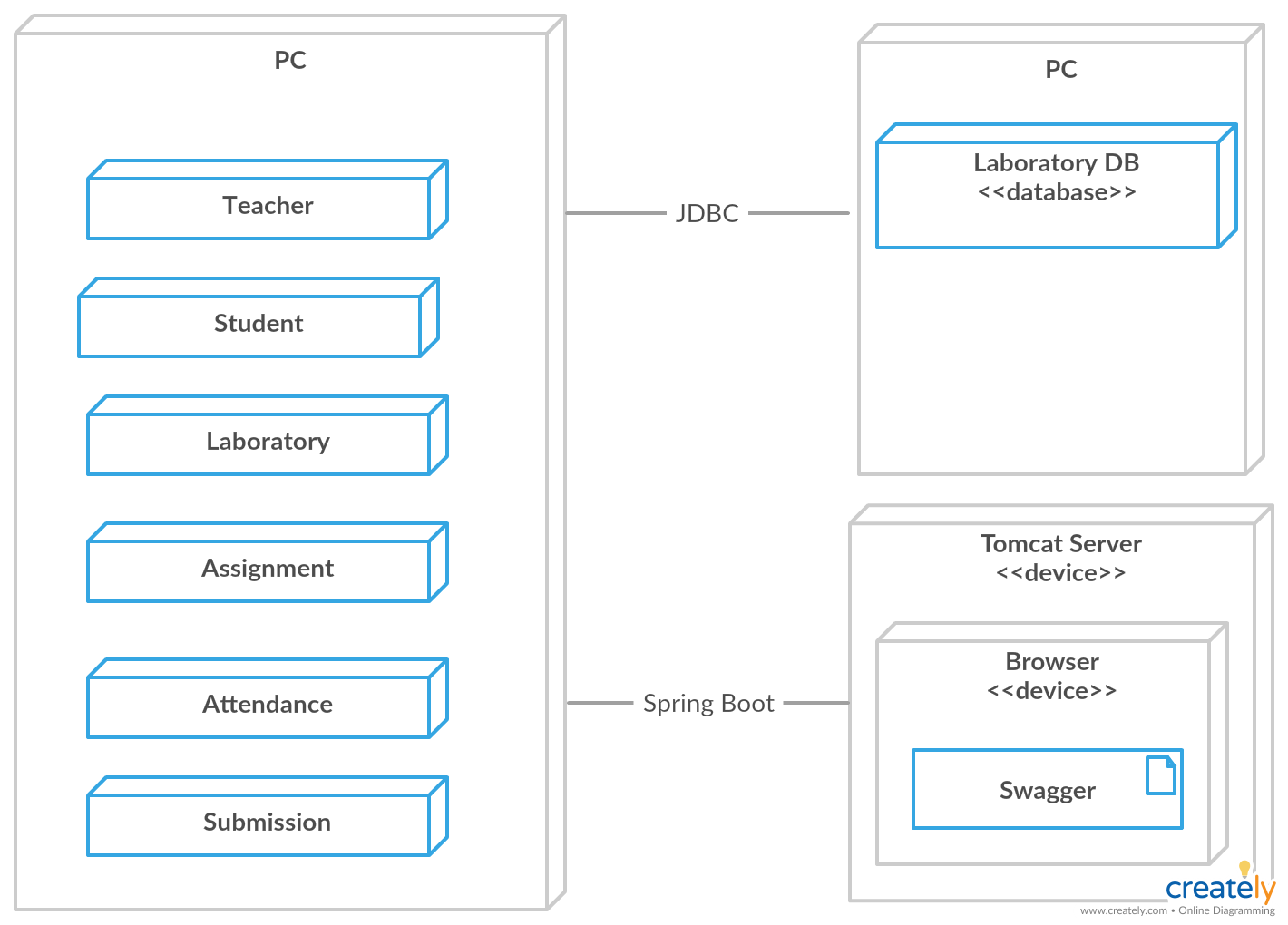
**3.2 Diagrams**





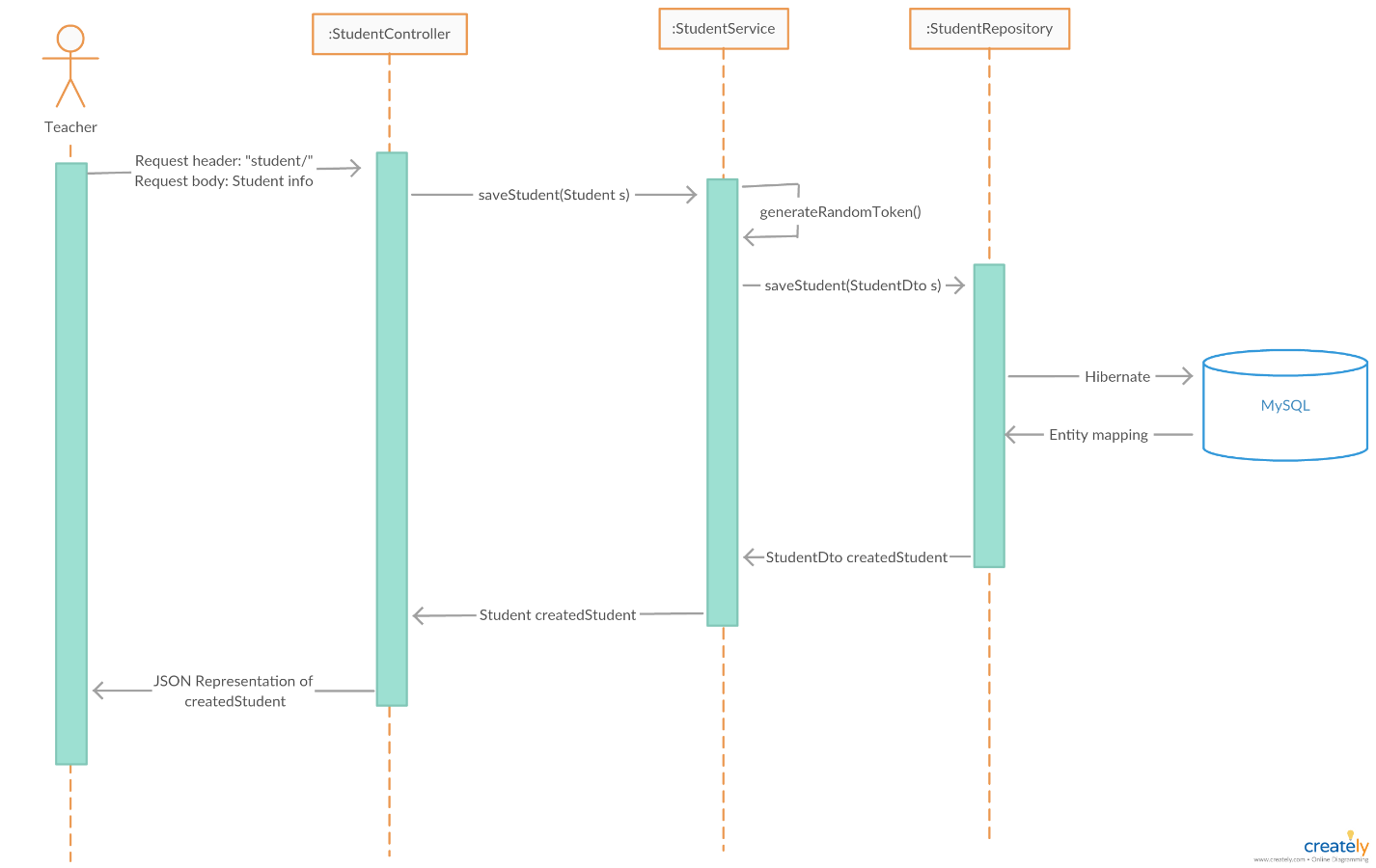
Note: only backend part is created. **Exemplification of MVC**.





4. UML Sequence Diagrams

* Teacher creates a new student account. A token is generated.



5. Class Design

**5.1 Design Patterns Description**

**MVC** is a software architecture - the structure of the system - that separates domain/application/business logic from the rest of the user interface. It does this by separating the application into three parts: the model, the view, and the controller.

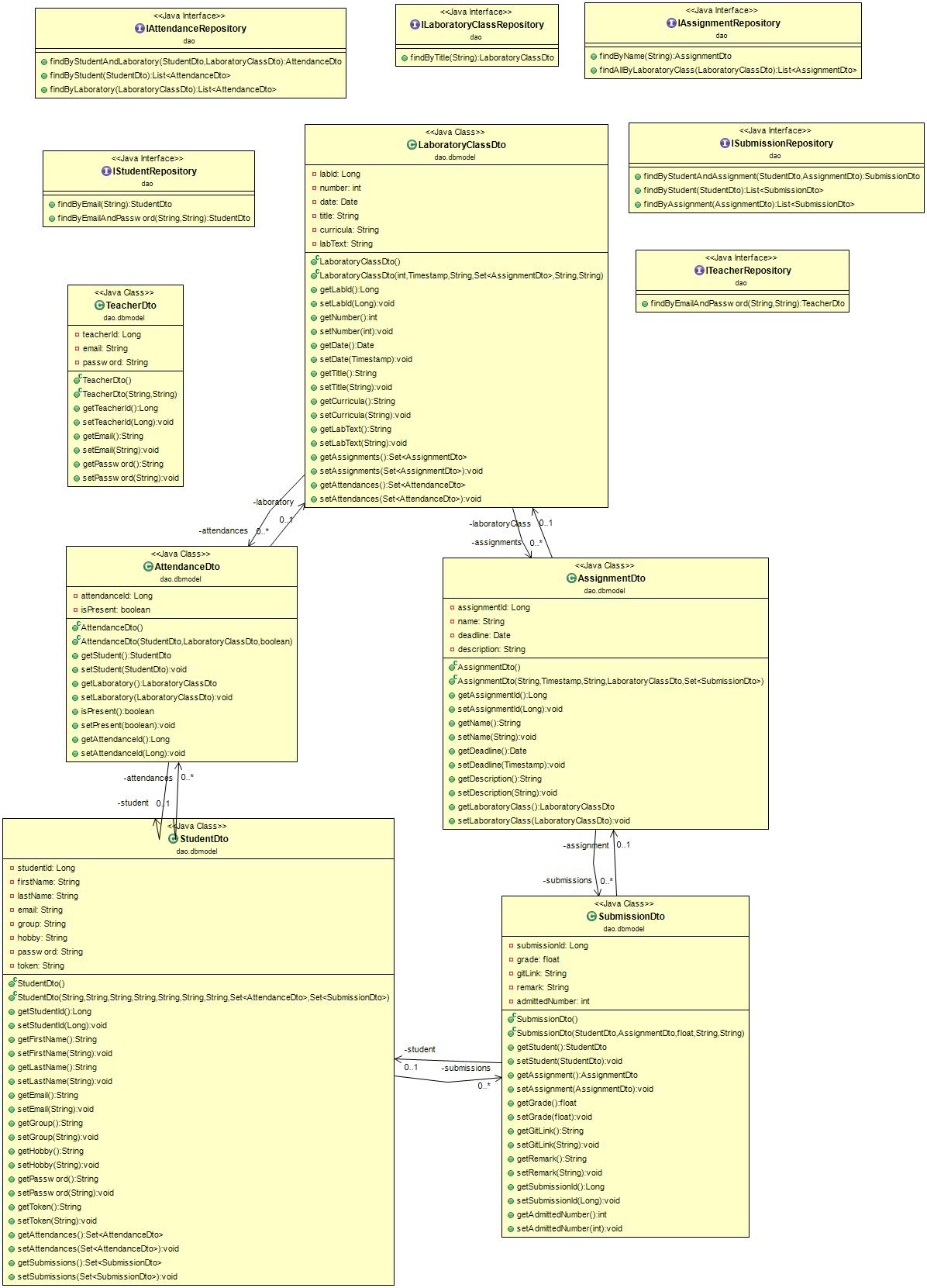
The model manages fundamental behaviors and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, and even to notify observers in event-driven systems when information changes. This could be a database.

The view effectively provides the user interface element of the application. It'll render data from the model into a form that is suitable for the user interface.

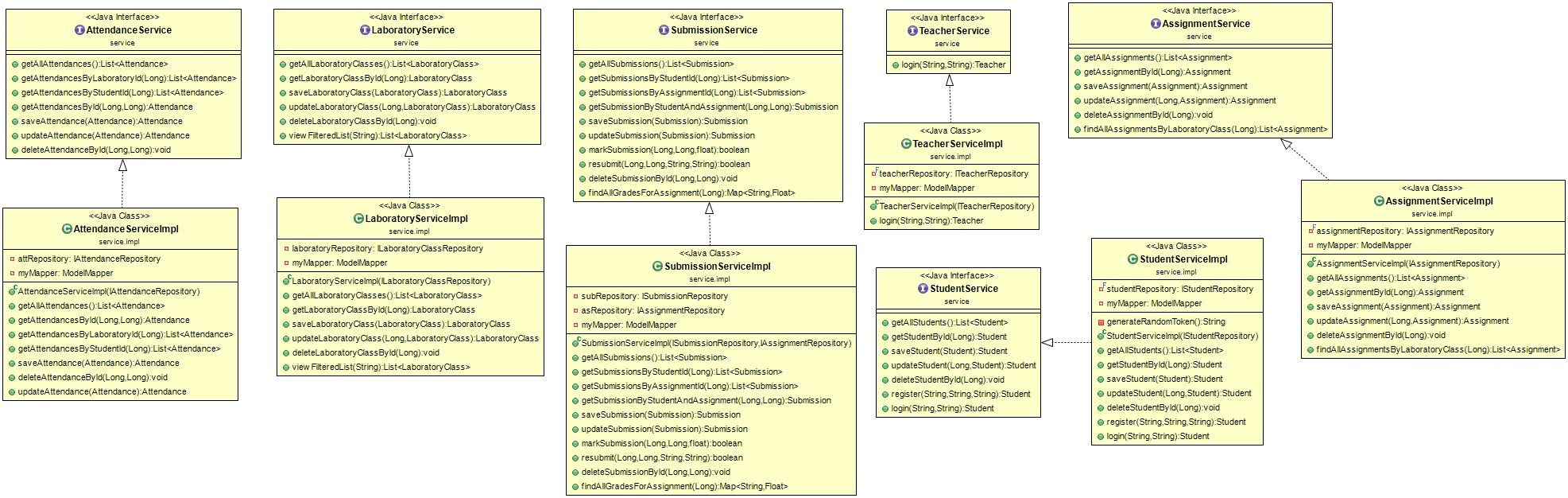
The controller receives user input and makes calls to model objects and the view to perform appropriate actions.

**5.2 UML Class Diagram**

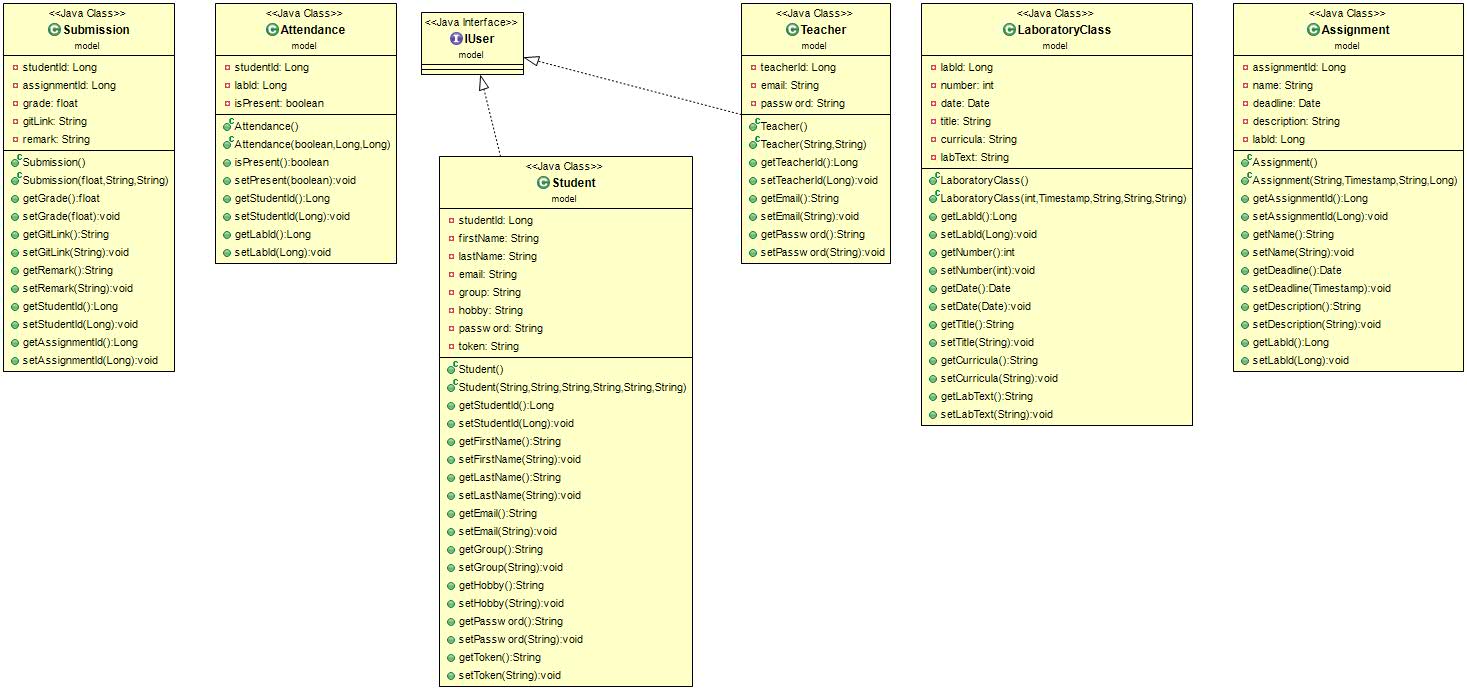
DAO



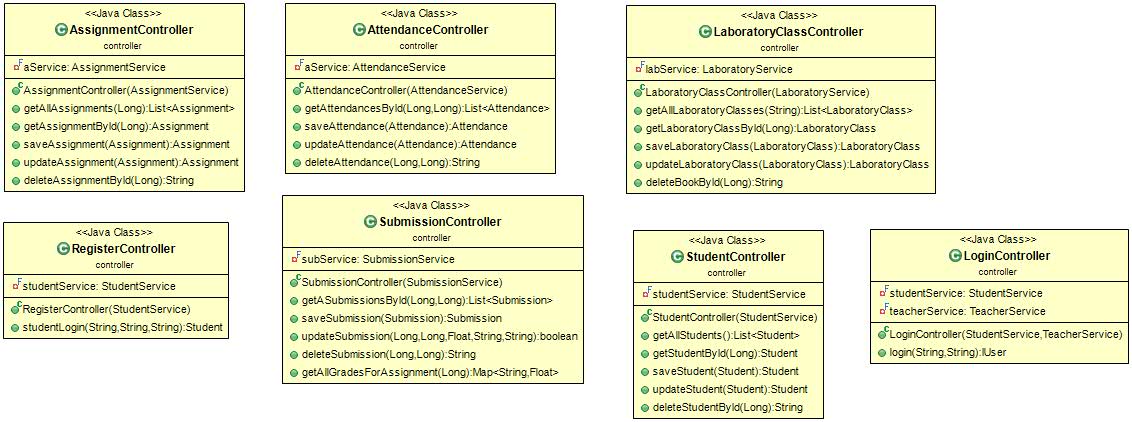
SERVICE



MODEL

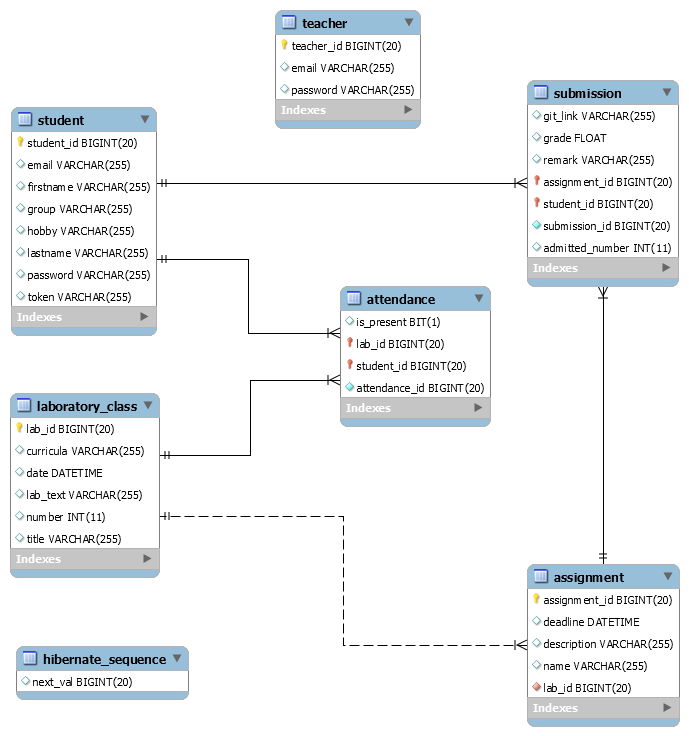


CONTROLLER



Note: Other utility classes (mappers and config classes) are used but excluded from the diagrams above.

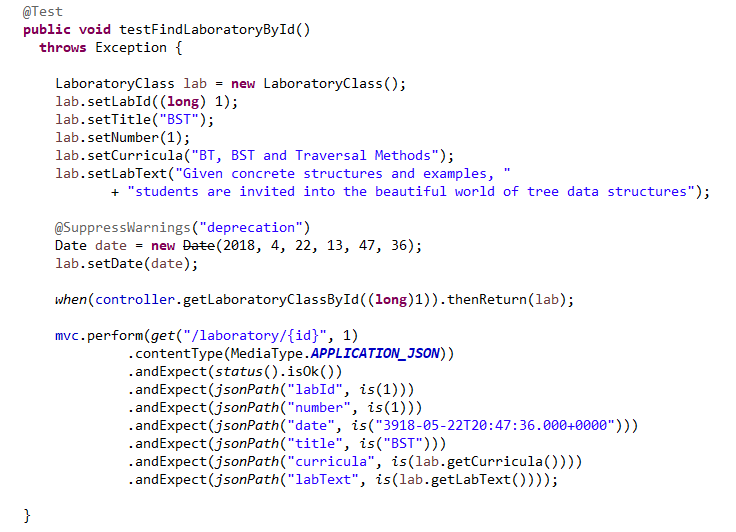
6. Data Model

**

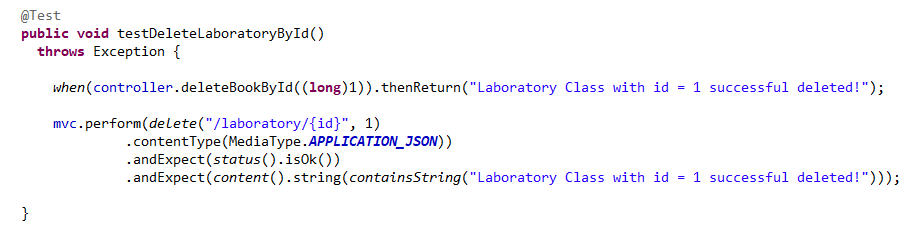
7. System Testing

Strategies: Junit, mocking (Mockito).

* Find a laboratory by a given id

**

* Delete a laboratory by a given id

**

8. Bibliography

* MVC

<https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>

* Testing in Spring Boot

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

* Spring Boot Many-to-Many associations

<https://vladmihalcea.com/the-best-way-to-map-a-many-to-many-association-with-extra-columns-when-using-jpa-and-hibernate/>

* Setting up Swagger2 with a Spring REST API

<http://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>

* Building a RESTful Web Service

<https://spring.io/guides/gs/rest-service/>