<Laboratory Manager>

Student: Cornescu Andreea Mara

**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 teacher can perform the following operations:

1) Login

2) 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.

3) 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).

5) CRUD on attendance for each lab.

6) 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.

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

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

* The student can perform the following operations:

1) Register using the token generated by the teacher.

2) Login with the username and password.

3) 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.

4) View the assignments for a laboratory class.

5) 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

1) The data will be stored in a database.

2) Use the MVC architectural pattern to organize your application. For this assignment we will create only the backend part (Model, Controller, Services and Repositories).

3) API design should be RESTful.

4) Use and ORM (Hibernate / Entity framework) to access the database

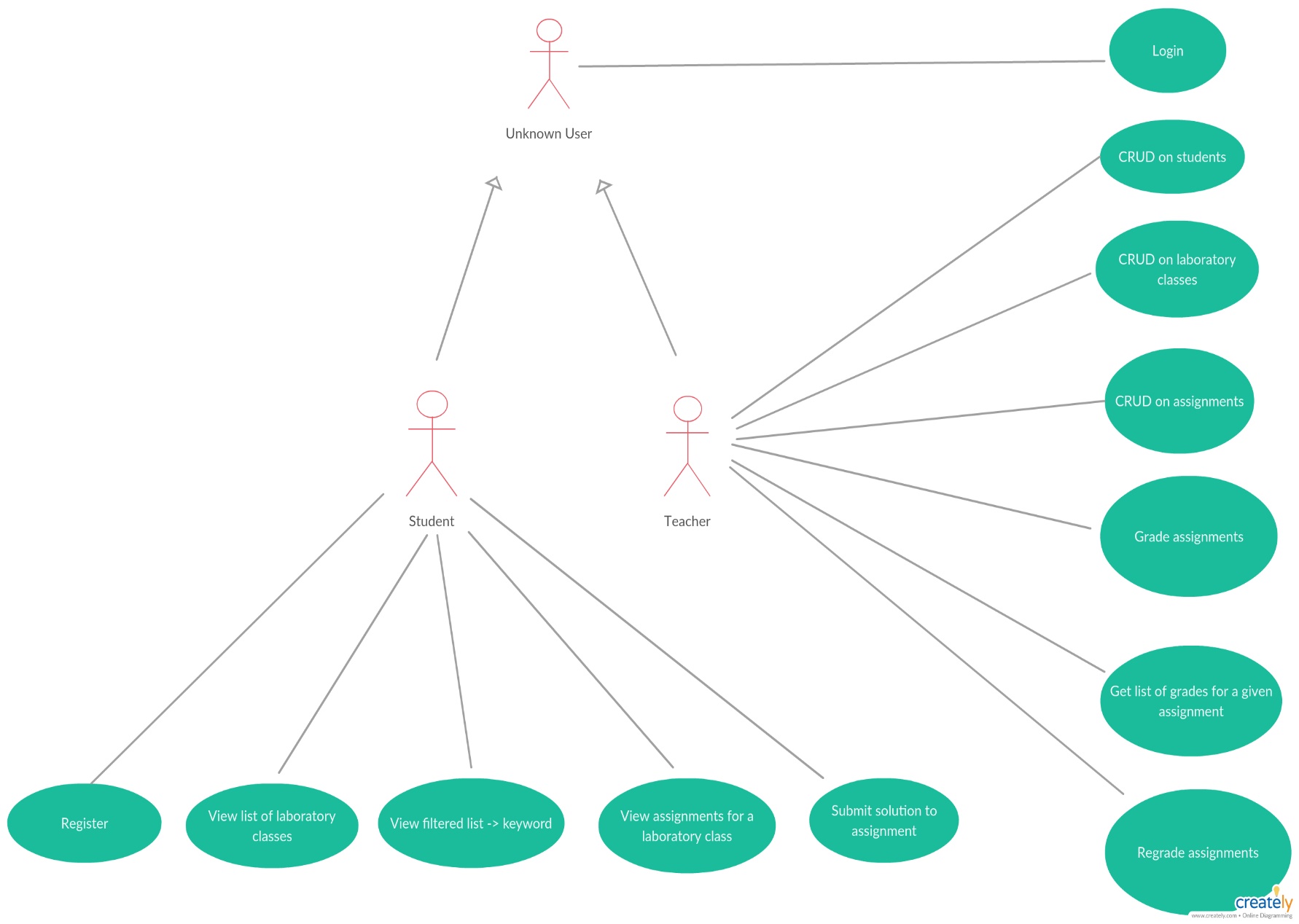
5) Use dependency injection to inject Services in Controllers and Repositories in Services

6) Install and use Swagger to call your APIs.

7) Connection string should be stored in a separate config file

8) Create one set of API tests for one of the controllers (for example: LaboratoryClassController)

2. Use-Case Model



3. System Architectural Design

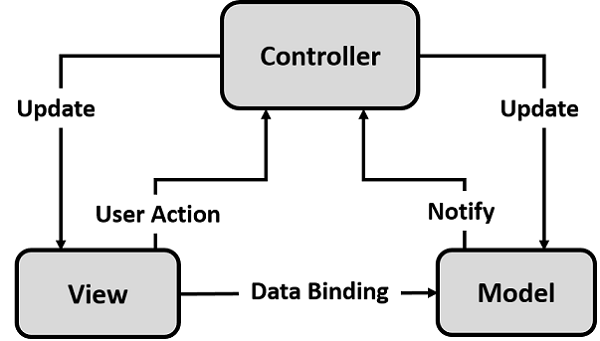
**3.1 Architectural Pattern Description**

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

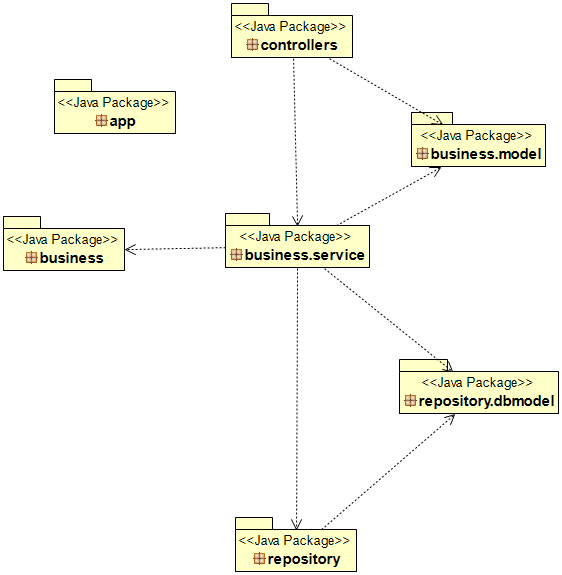
* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

**3.2 Diagrams**

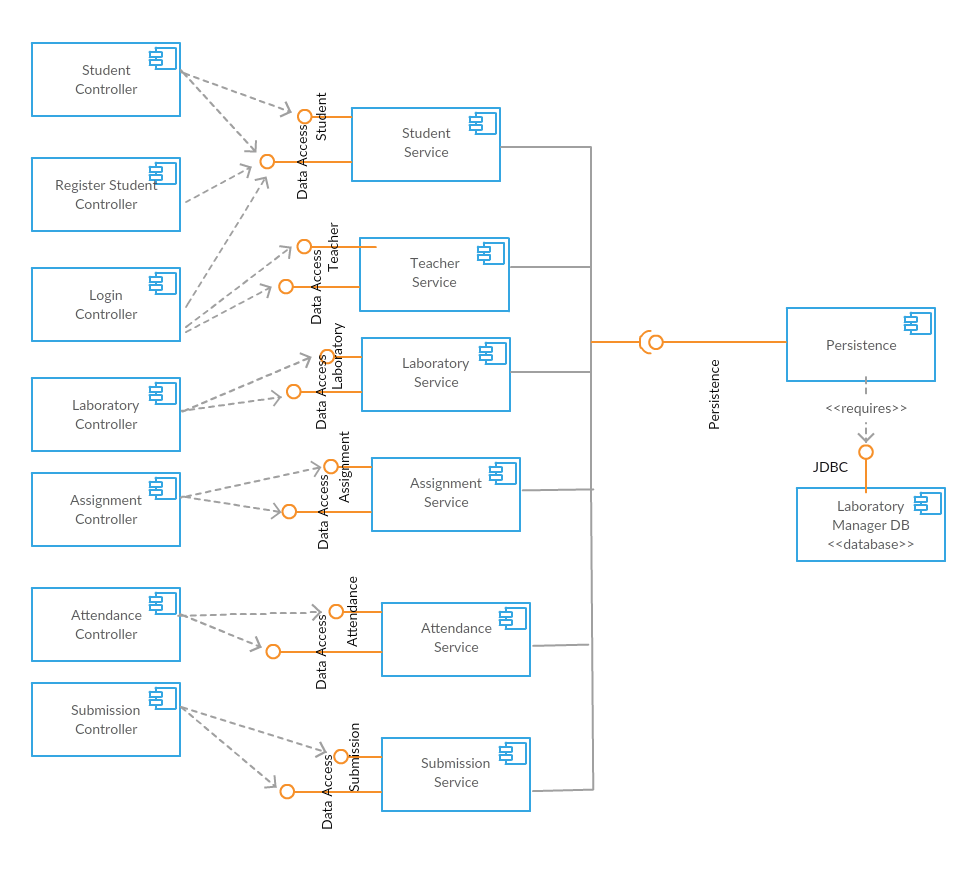
**Conceptual Architecture**



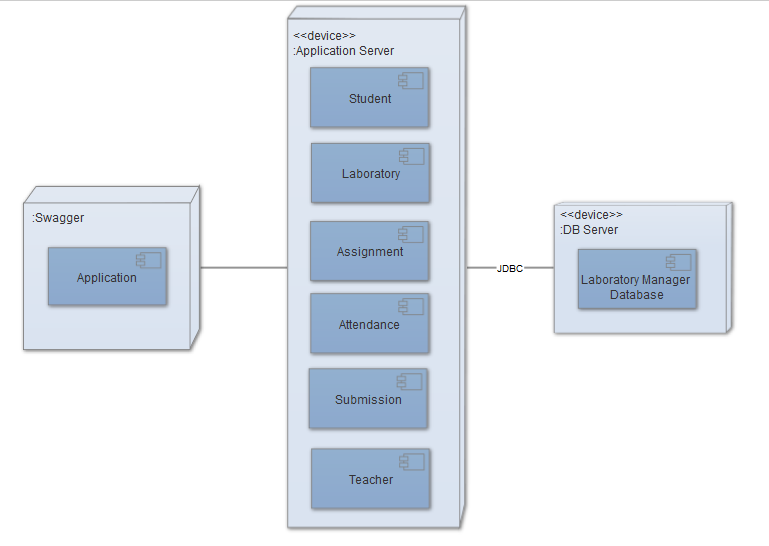
**Package Diagram**



**Component Diagram**

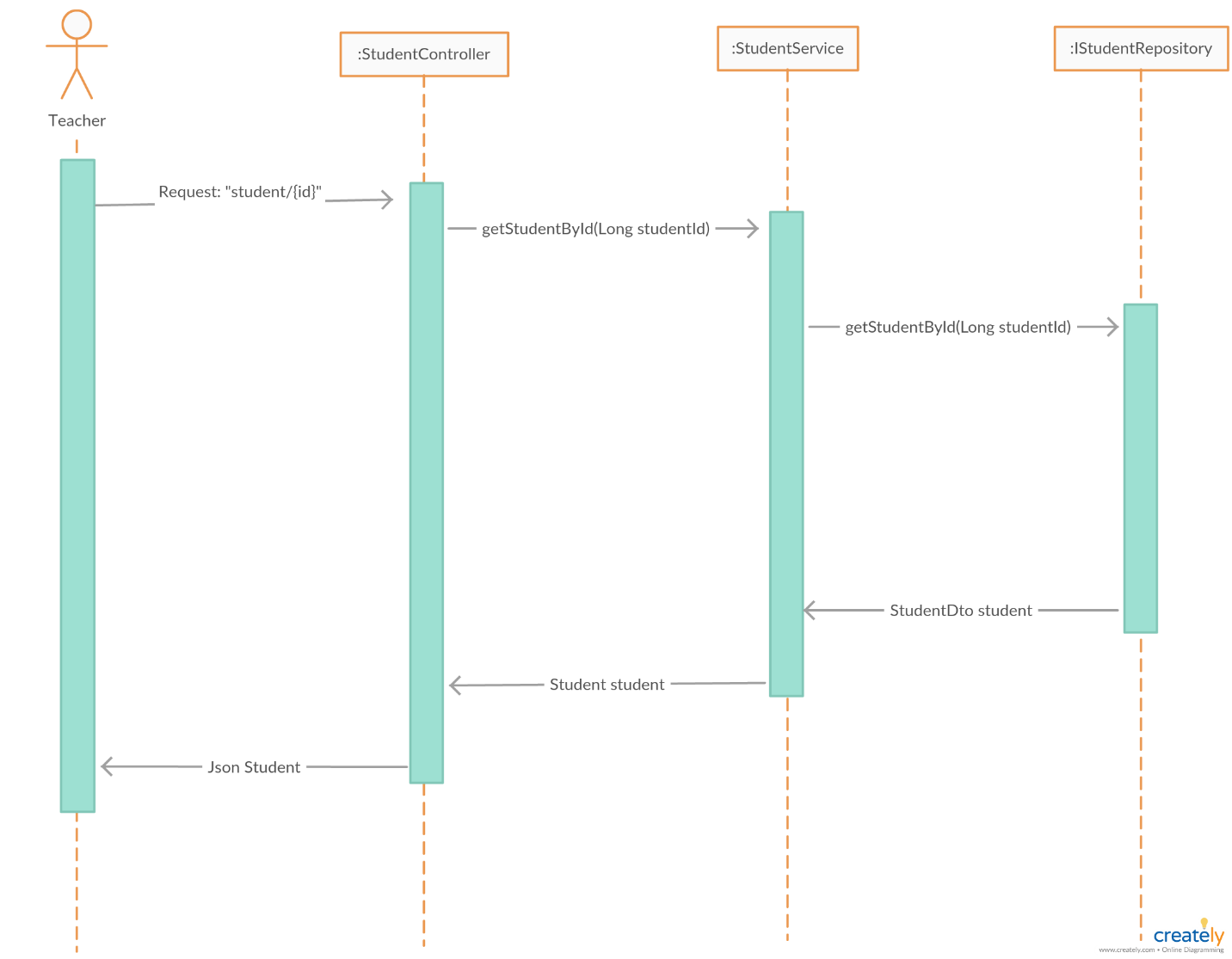
****

**Deployment Diagram**

****

4. UML Sequence Diagrams

Sequence diagram for finding a student by id:



5. Class Design

**5.1 Design Patterns Description**

**Models**

Models represent knowledge. A model could be a single object (rather uninteresting), or it could be some structure of objects.

There should be a one-to-one correspondence between the model and its parts on the one hand, and the represented world as perceived by the owner of the model on the other hand.

**Views**

A view is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a *presentation filter*.

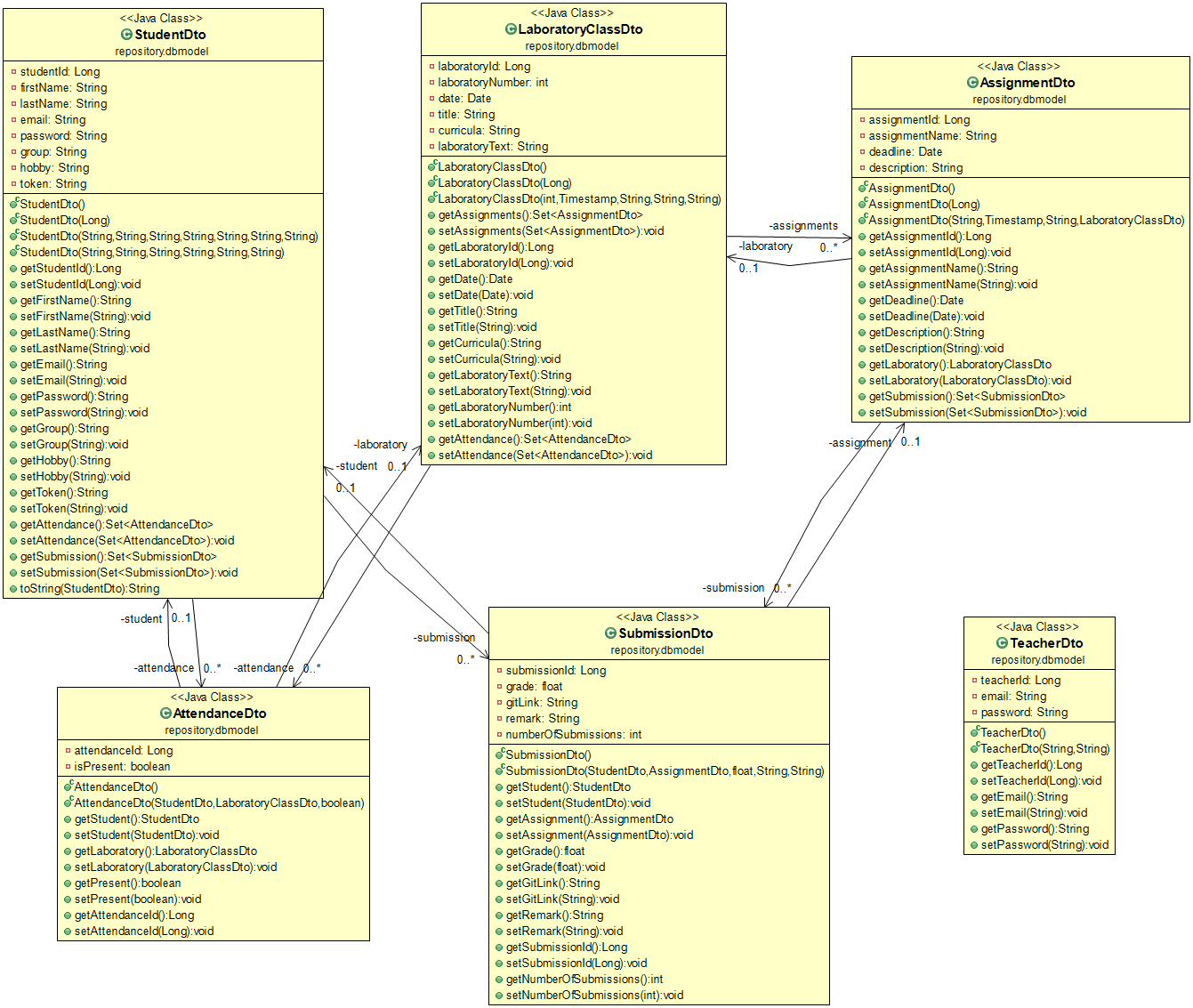
A view is attached to its model (or model part) and gets the data necessary for the presentation from the model by asking questions. It may also update the model by sending appropriate messages. All these questions and messages have to be in the terminology of the model, the view will therefore have to know the semantics of the attributes of the model it represents.

**Controllers**

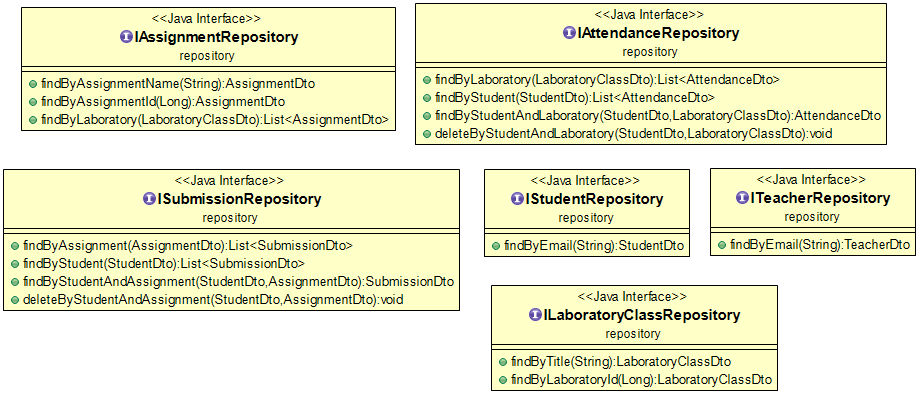
A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and passes these messages on to one or more of the views.

**5.2 UML Class Diagram**

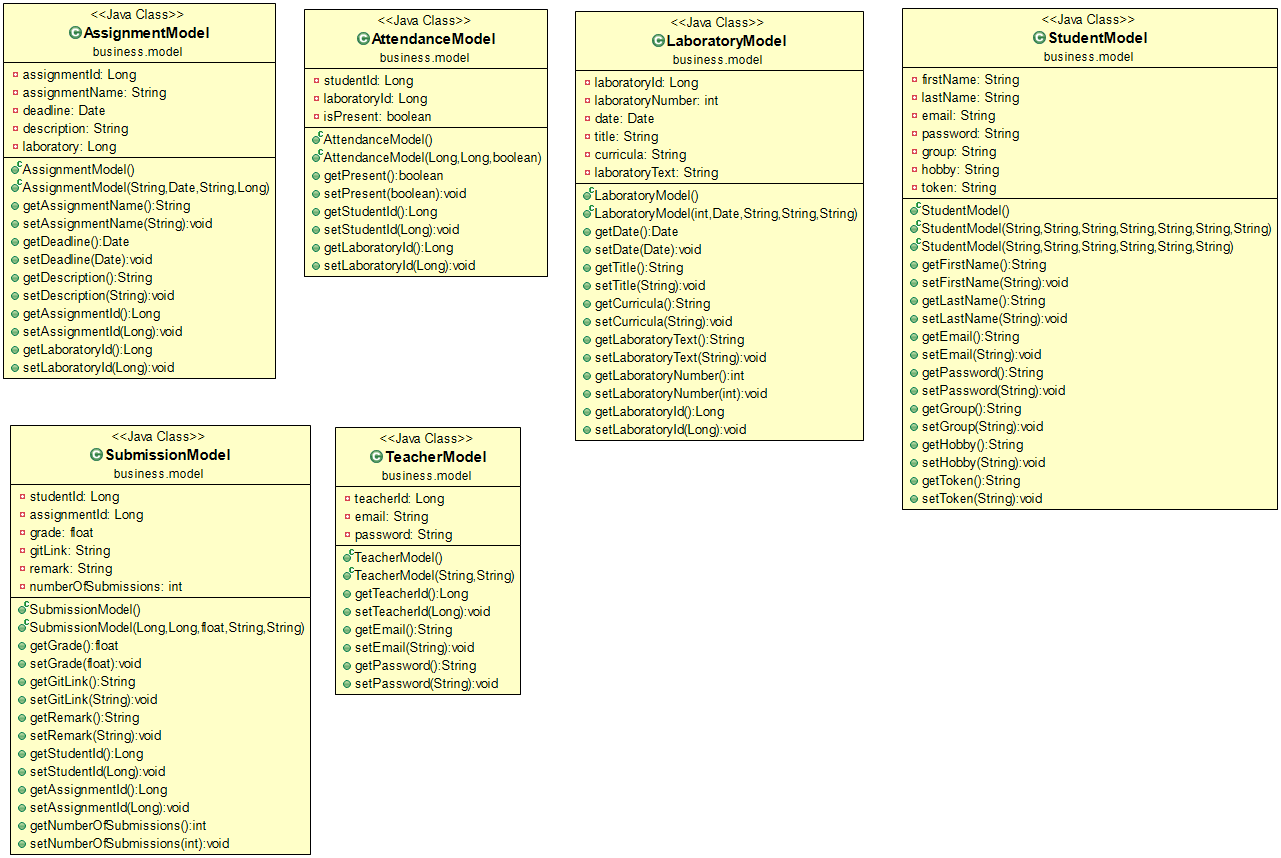
* dbmodel package



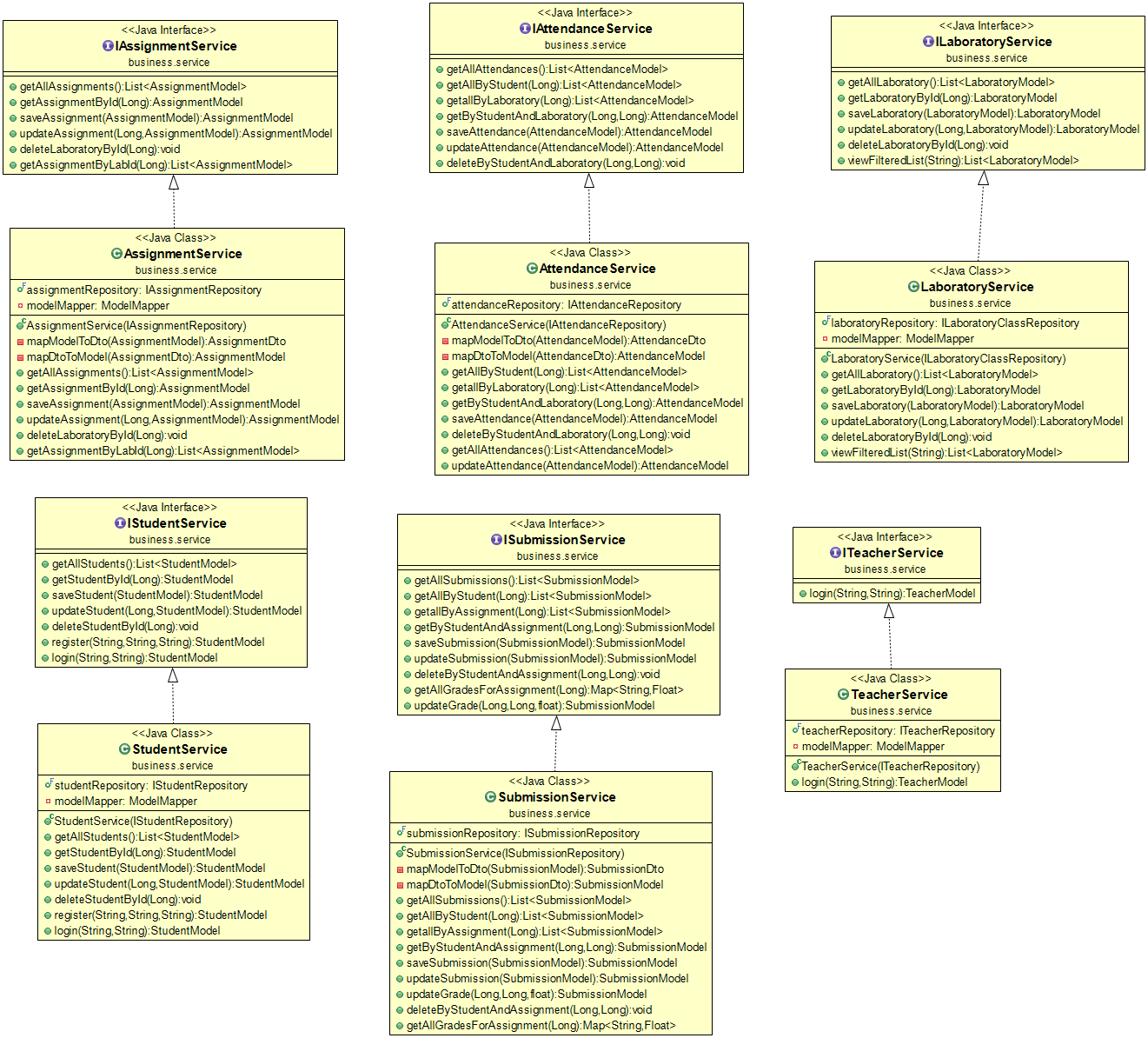
* repository package



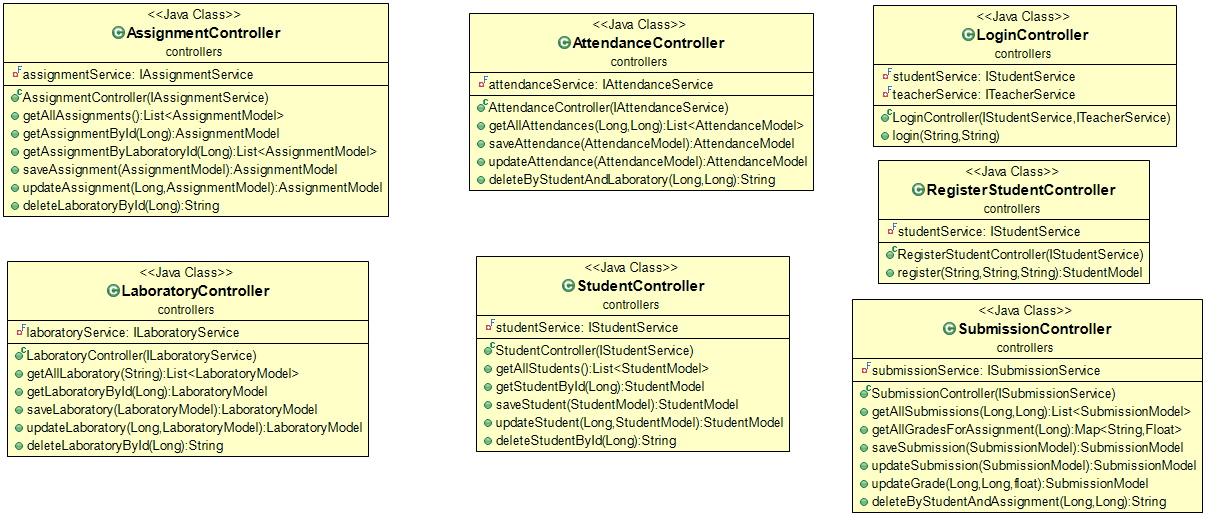
* business.model package



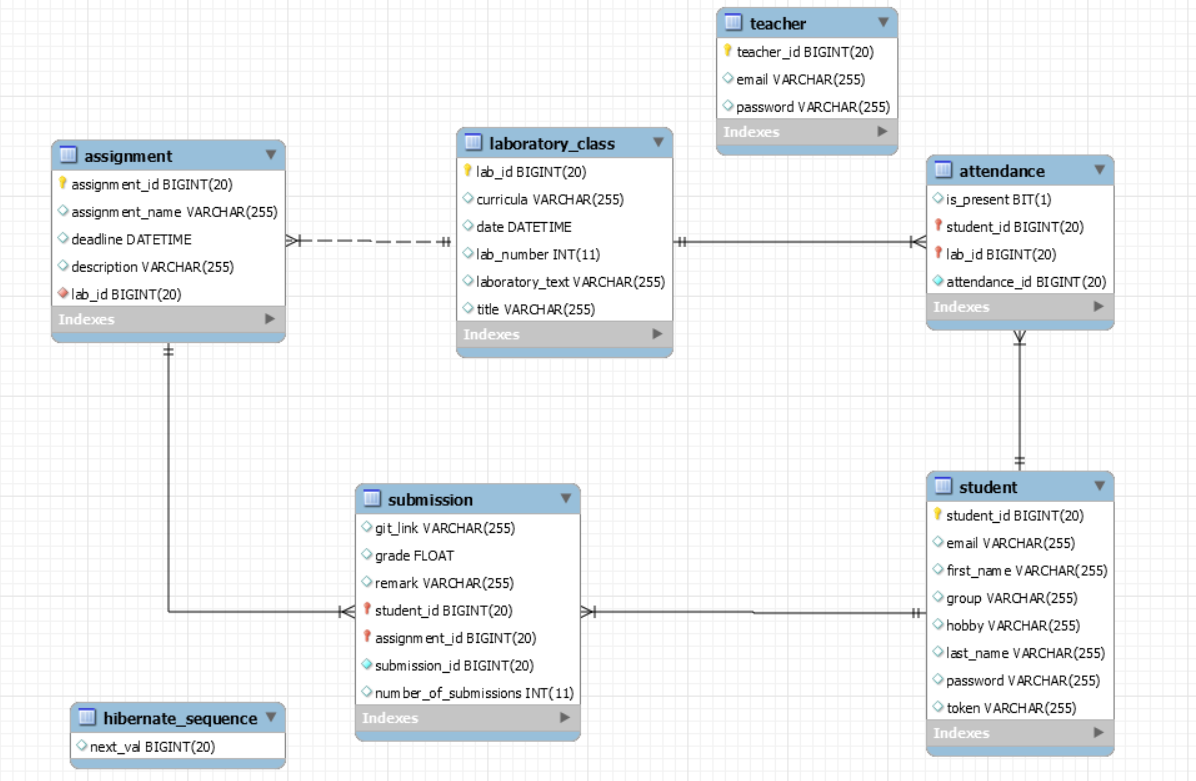
* business.service package



* controllers package



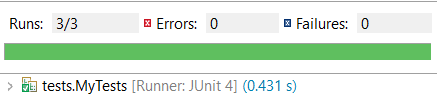
6. Data Model

**

7. System Testing

Unsing: Junit, mocking (Mockito).

Unit testes were designed to verify if the methods for saving a laboratory, find laboratory by id, and find all laboratories execute correctly. To find all show a mocking method for the database was used.



Find all laboratories test:



8. Bibliography

* MVC: <http://www.tutorialsteacher.com/mvc/mvc-architecture>
* Diagrams: <https://creately.com>
* Spring Boot: <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html>
* Testing: <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html>