<Laboratory Manager>

Cornescu Andreea Mara

**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 in combination with JAVA Swing/.NET WinForms to design and implement an application for tracking the laboratory activity for the Software Design laboratory.

# 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.

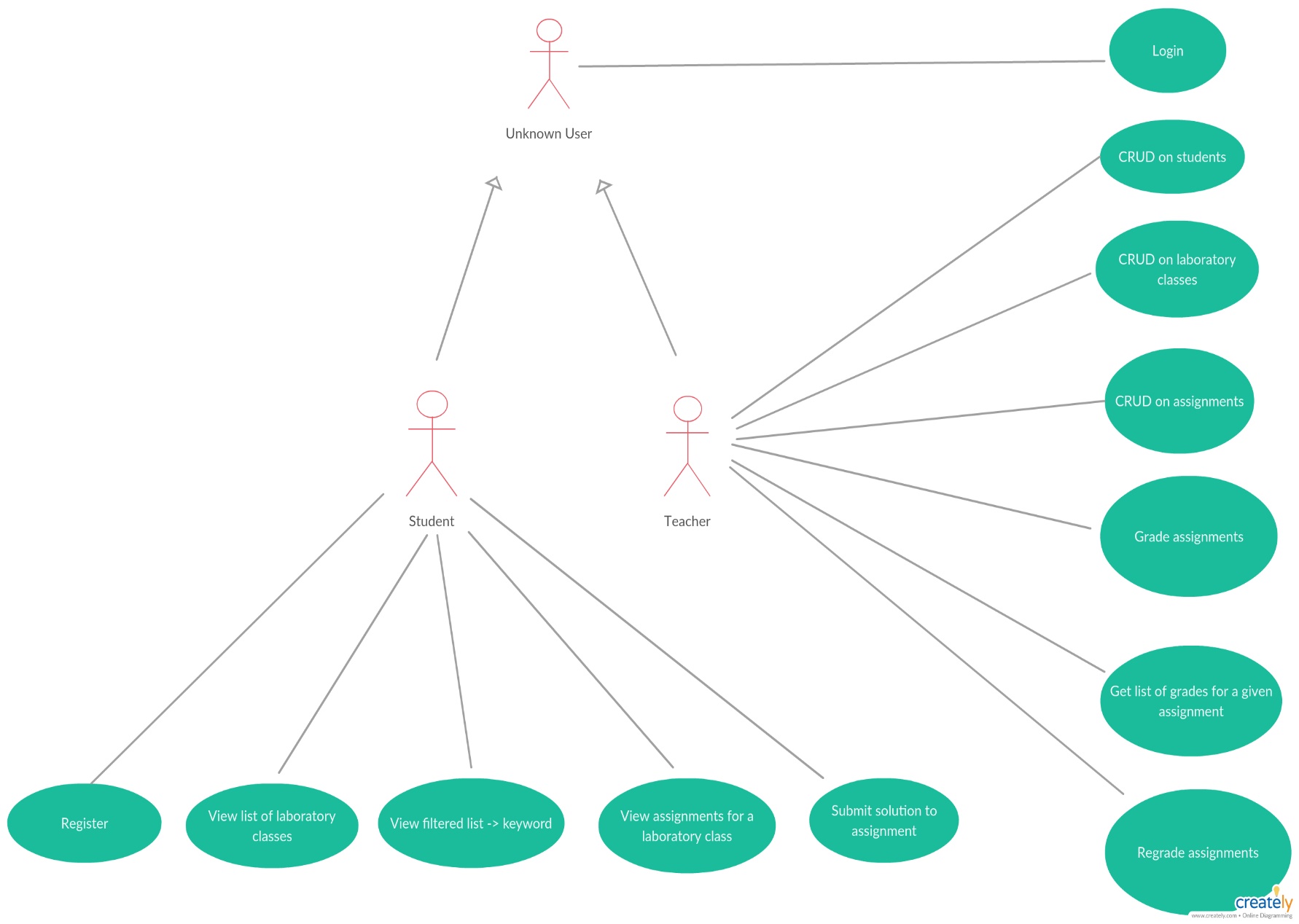
3) View a list of laboratory classes. Also view a filtered list: student inserts a keyword and that keyword is searched in the curricula and long description.

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

* Encrypt the password from the UI with a one direction algorithm and then work with the password encrypted;
* Create a system that notifies by email all students when a new assignment is posted by the teacher;
* When a new user is added by the teacher, the token is sent by email
* Secure all APIs with an authorization header that contains an Email and a Password. First, you will have to check that the password is correct and then find out the role (Teacher/Student) for that given user.

2. Use-Case Model

3. System Architectural Design

**3.1 Architectural Pattern Description**

Model-View-Presener(MVP) is a variation of the Model-View-Controller (MVC) architectural pattern for building user interfaces. The main goal of MVP and its variants is separation of concerns between the user interface (UI), the model (application data), and the presenter (controller that handles business/presentation logic). This version of MVP is called passive because the View does not actively update itself, instead choosing to allow the presenter to handle that task.

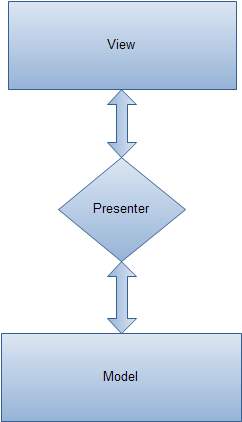
The *model* is an interface defining the data to be displayed or otherwise acted upon in the user interface.

The *view* is a passive interface that displays data (the model) and routes user commands ([events](https://en.wikipedia.org/wiki/Event_(computing))) to the presenter to act upon that data.

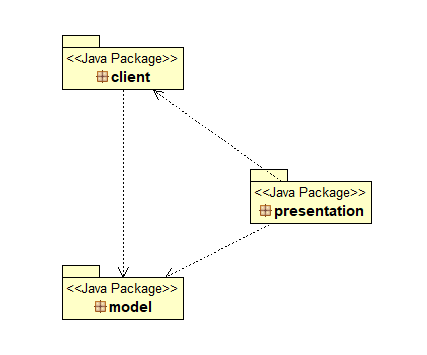
The *presenter* acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

**3.2 Diagrams**

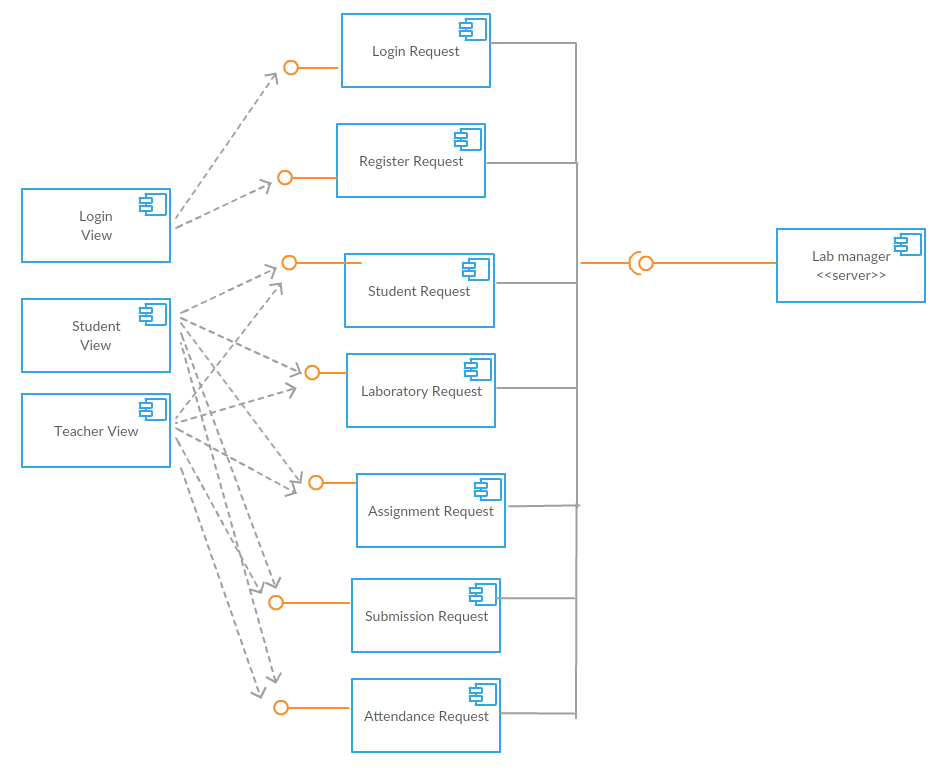
*System conceptual architecture*

**

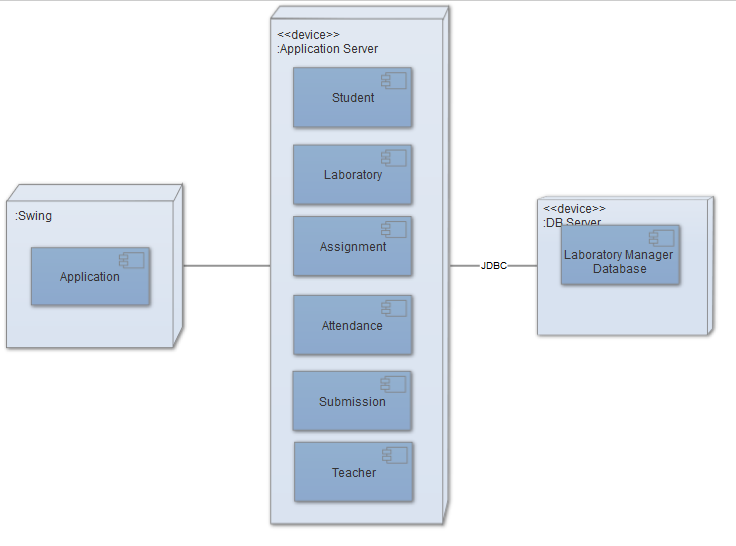
*Package diagram*

**

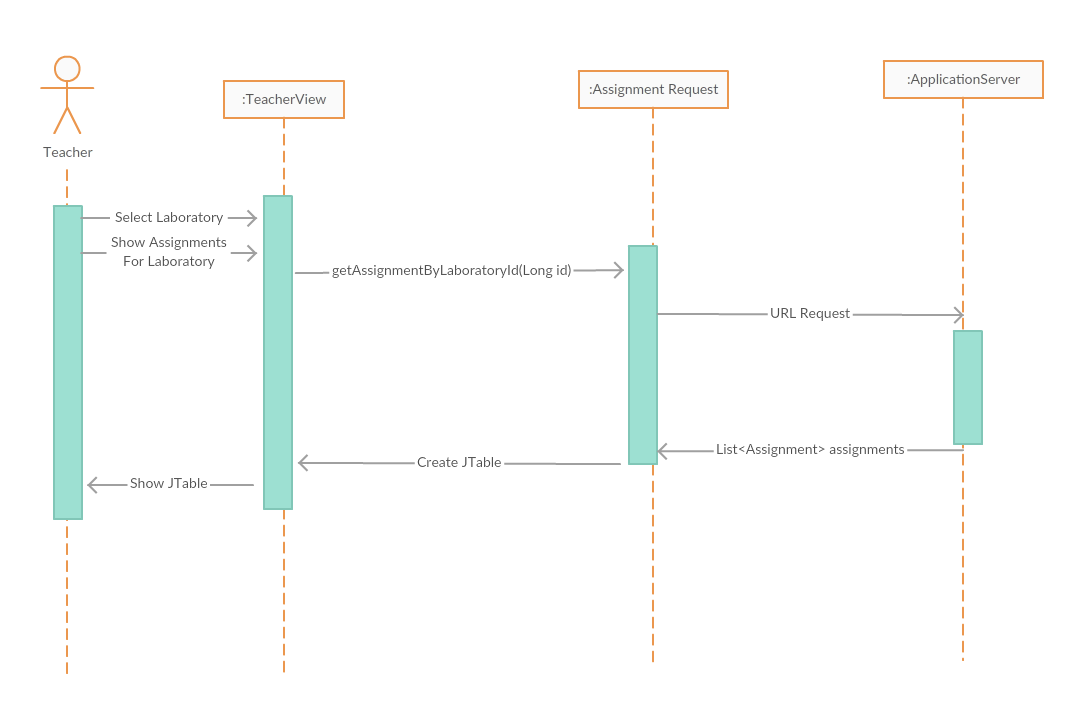
*Component diagram*

**

*Deployment diagram*

**

4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

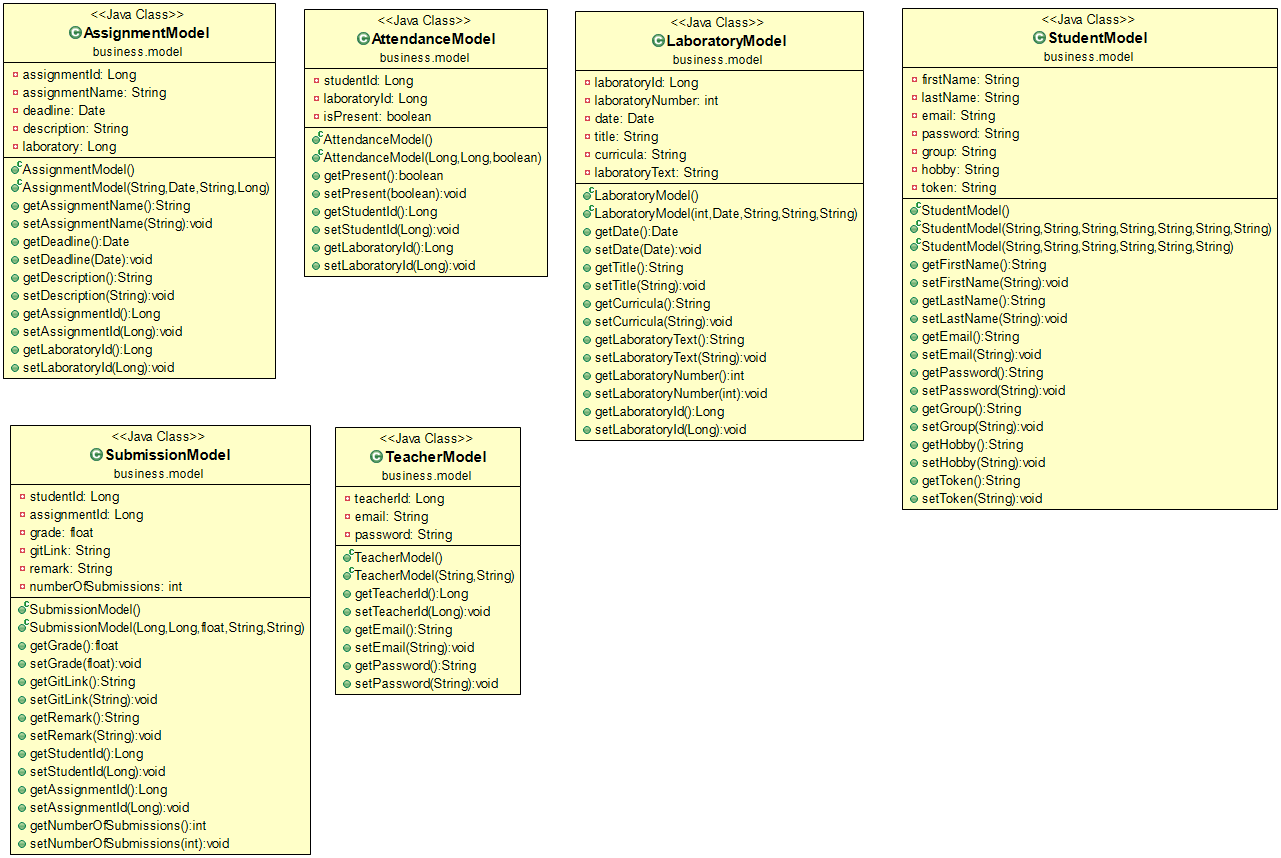
The *model* is an interface defining the data to be displayed or otherwise acted upon in the user interface.

The *view* is a passive interface that displays data (the model) and routes user commands ([events](https://en.wikipedia.org/wiki/Event_(computing))) to the presenter to act upon that data.

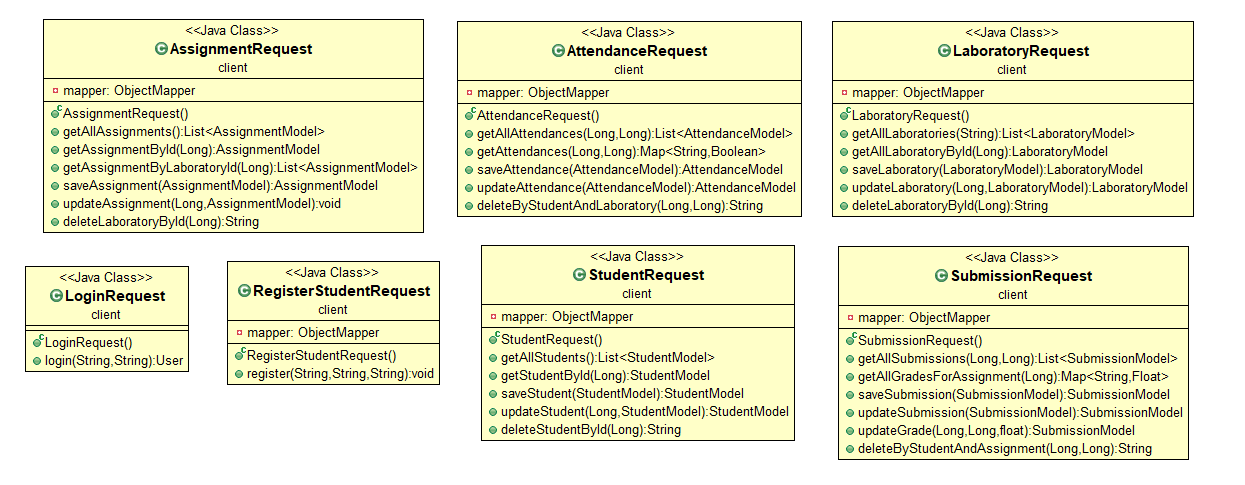
The *presenter* acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

**5.2 UML Class Diagram**

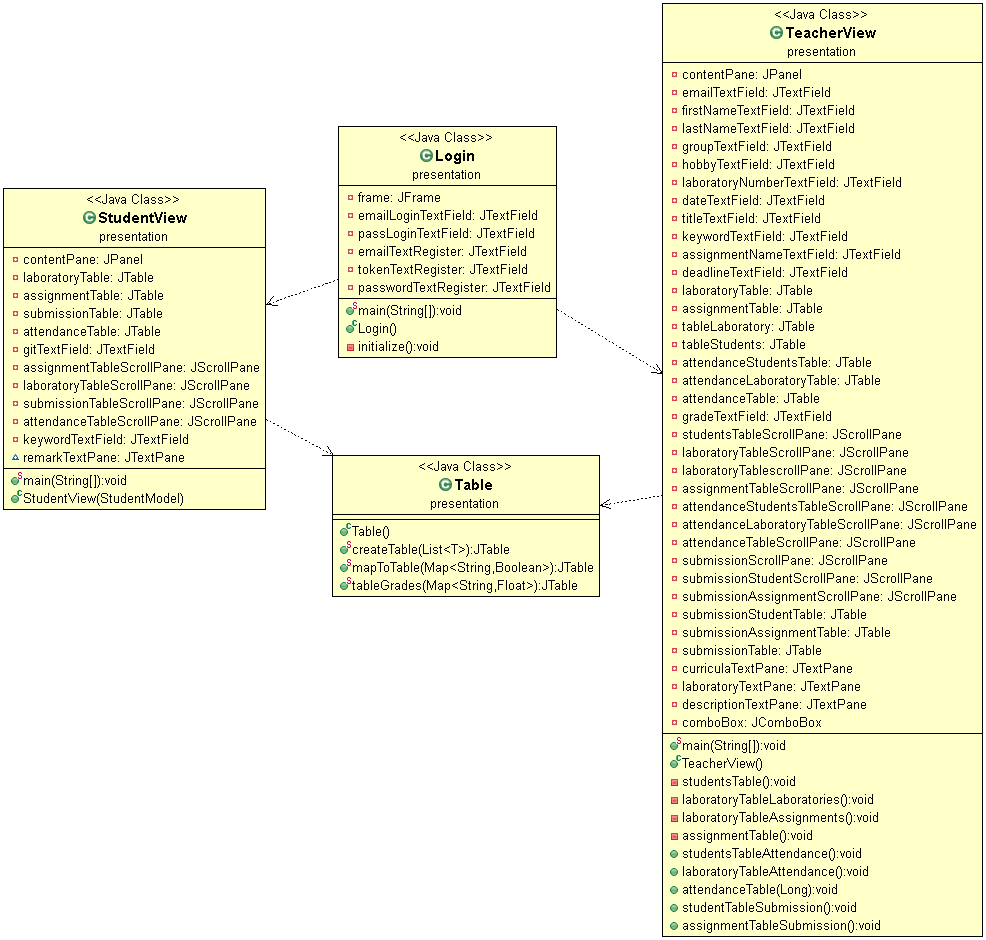
*Model package*

**

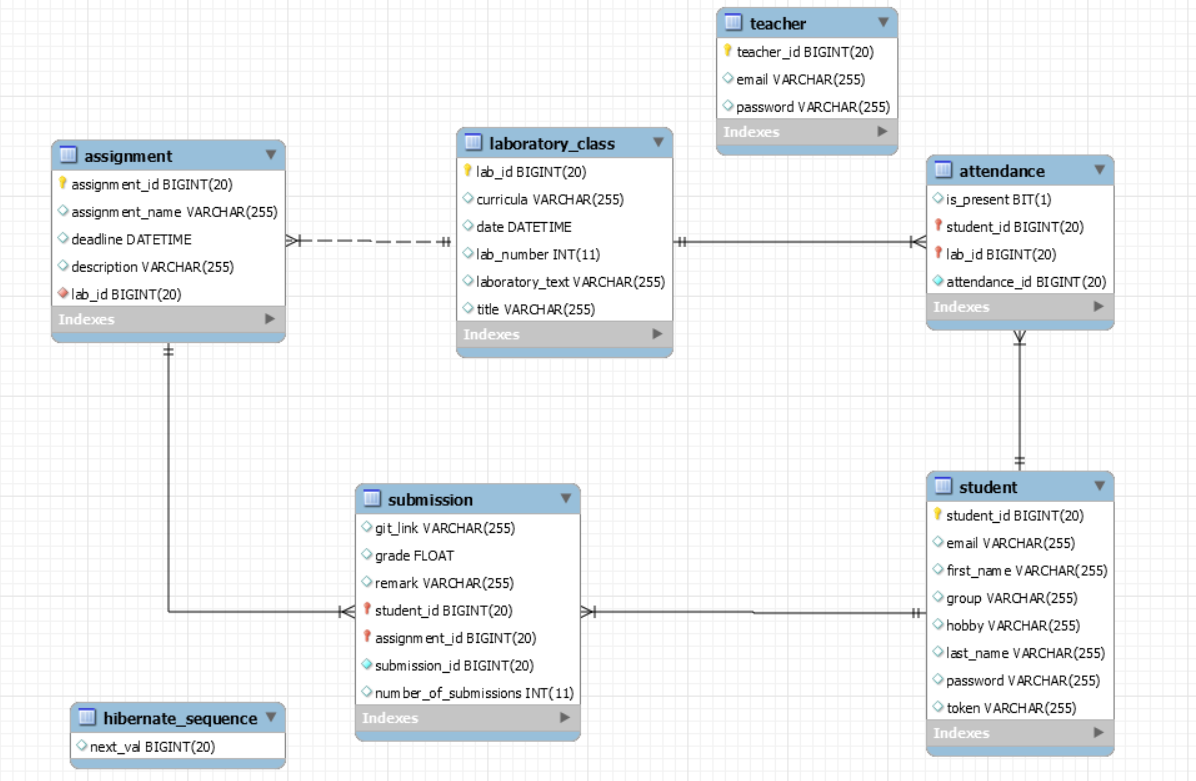
*Client package*

**

Presentation Package

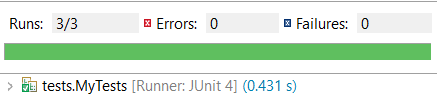


6. Data Model

**

7. System Testing

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.

**

**

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>