<Laboratory User>

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 in combination with JAVA Swing/.NET WinForms to design and implement an application for tracking the laboratory activity for the Software Design laboratory. The requirements of the application are provided in Assignment 2.

# Functional Requirements

* Create a desktop application to consume the APIs. Have screens for login, teacher side: register students, laboratory, assignments, attendance and grading; student side: first time register, login, laboratory, assignments, post attendance and grading. Nice to have: view attendance for students. Any additional feature that you consider helpful is highly appreciated.
* 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. Do not do this logic in every controller method, this logic is common for all controllers, so it has to be at a higher level. Depending on the roles, some APIs might be accessed only by the teacher (for example: add student, add laboratory), only by the student (submit assignment) or by both (view laboratories). - Encrypt the password from the UI with a one direction algorithm and then work with the password encrypted.
* Nice to have: - 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

# Non-functional Requirements

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

• MVC architectural pattern to organize your application.

• API design is RESTful.

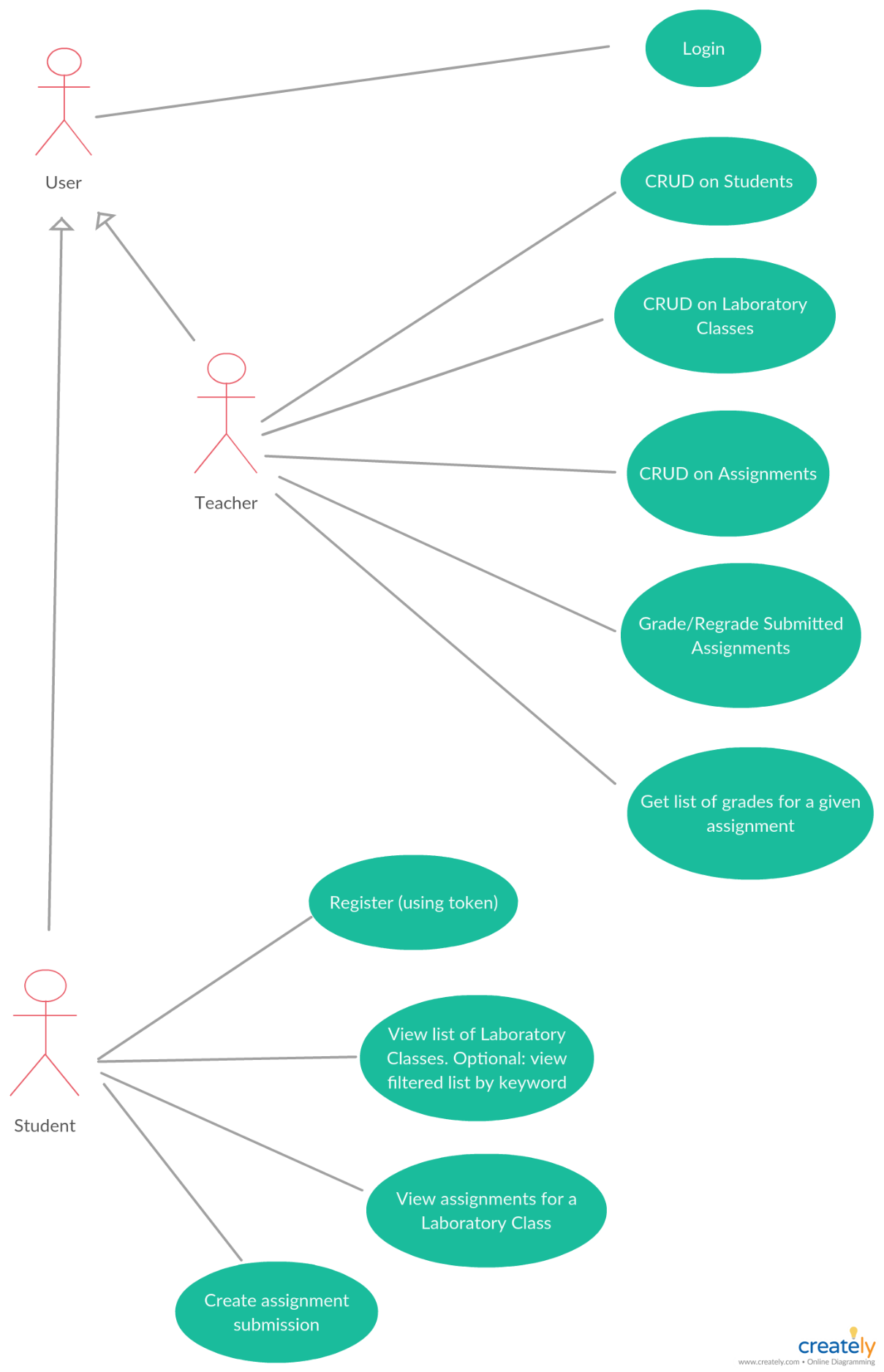
• ORM (Hibernate / Entity framework) to access the database

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

• Connection string should be stored in a separate config file

• Client to consume the APIs

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**

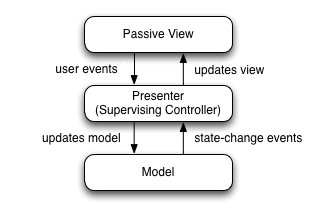
**Model–view–presenter** (**MVP**) is a derivation of the [model–view–controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) (MVC) [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern), and is used mostly for building [user interfaces](https://en.wikipedia.org/wiki/User_interface).

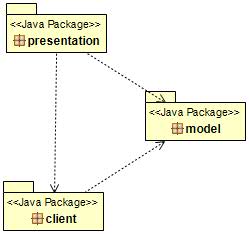
In MVP, the *presenter* assumes the functionality of the "middle-man". In MVP, all presentation logic is pushed to the presenter.

**Components**

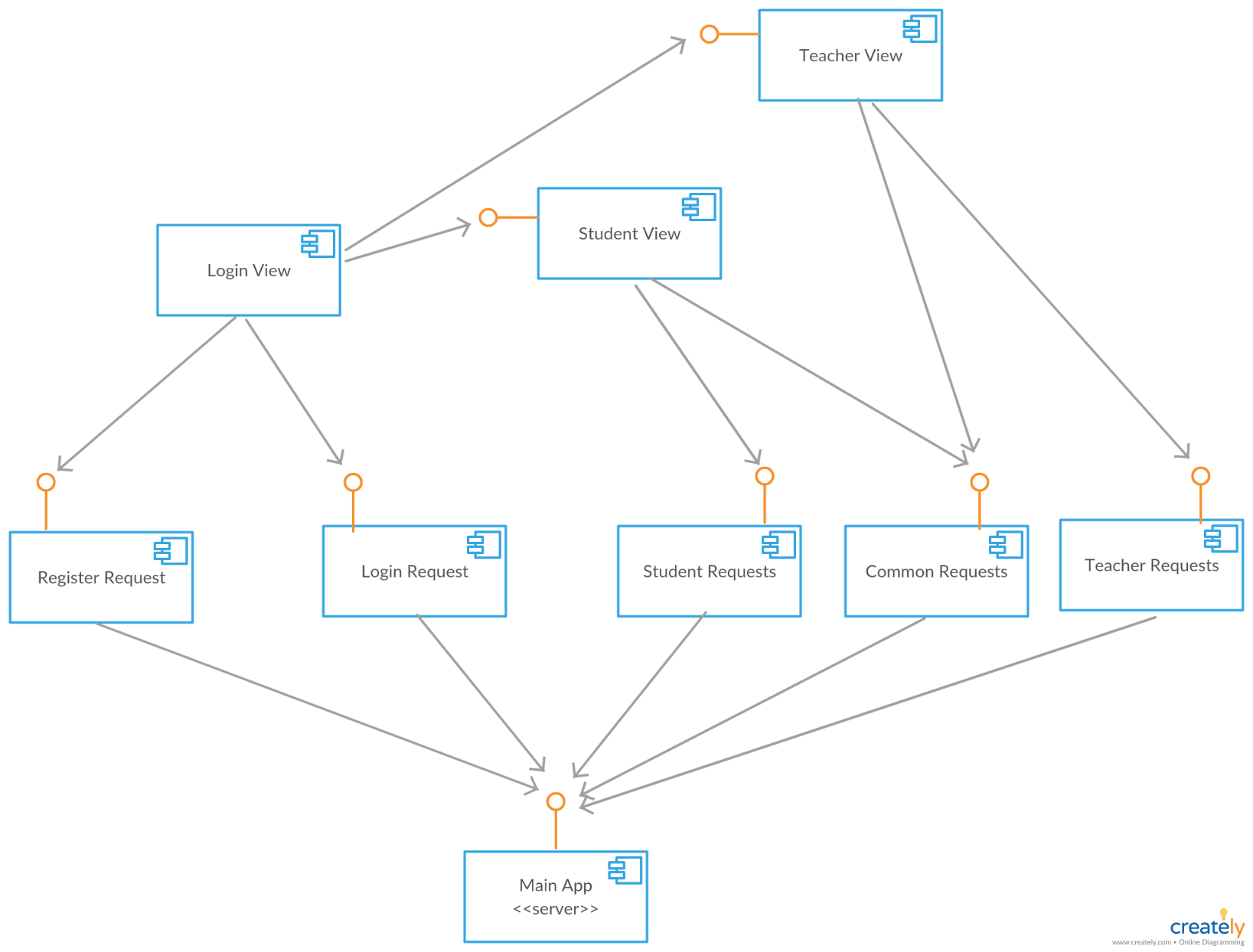
* The *model* is an interface defining the data to be displayed or otherwise acted upon in the user interface.
* The *view* is a passive interface that displays data (the model) and routes user commands ([events](https://en.wikipedia.org/wiki/Event_(computing))) to the presenter to act upon that data.
* The *presenter* acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

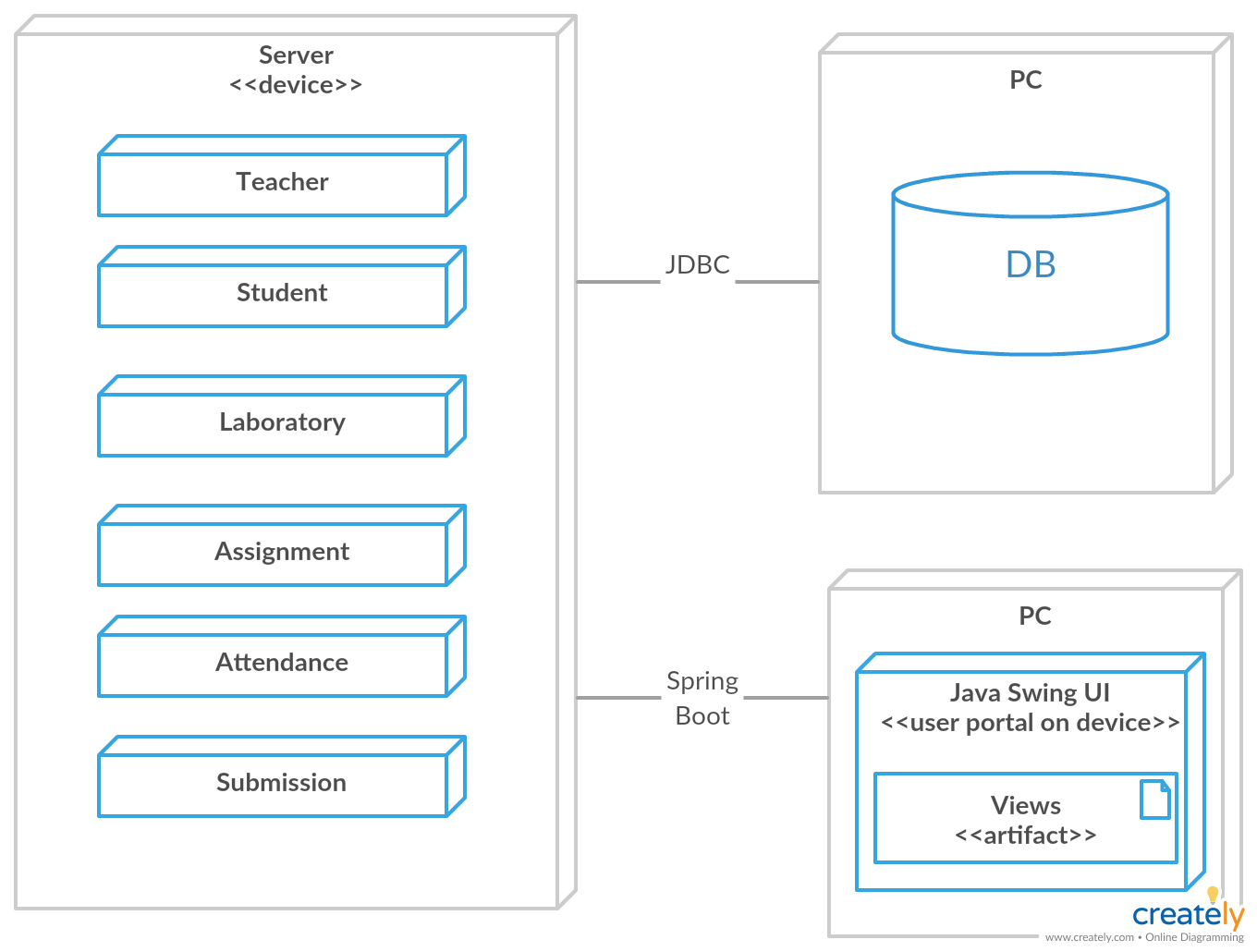
**3.2 Diagrams**



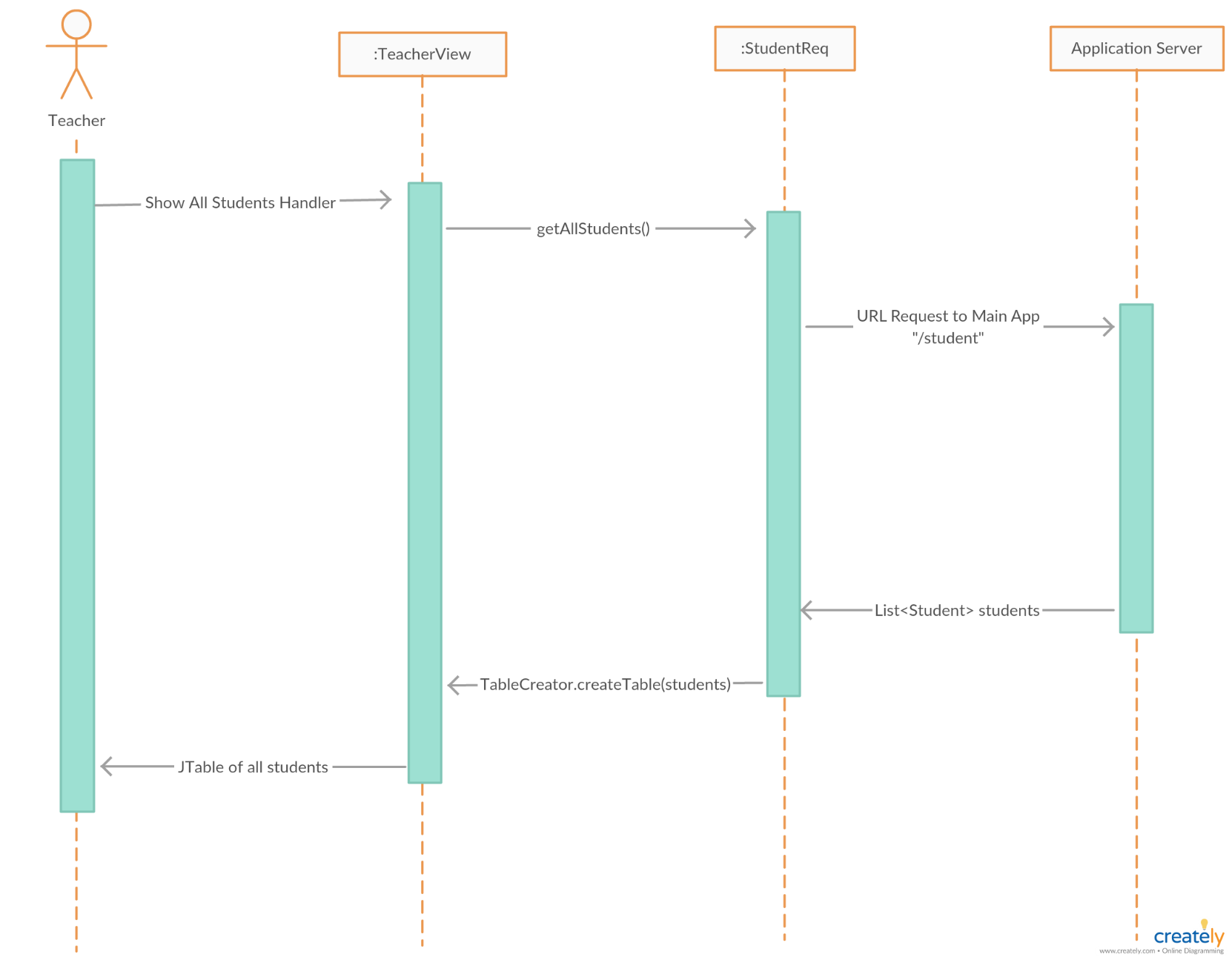


**Exemplification of MVP:** GUI Package Diagram





4. UML Sequence Diagrams

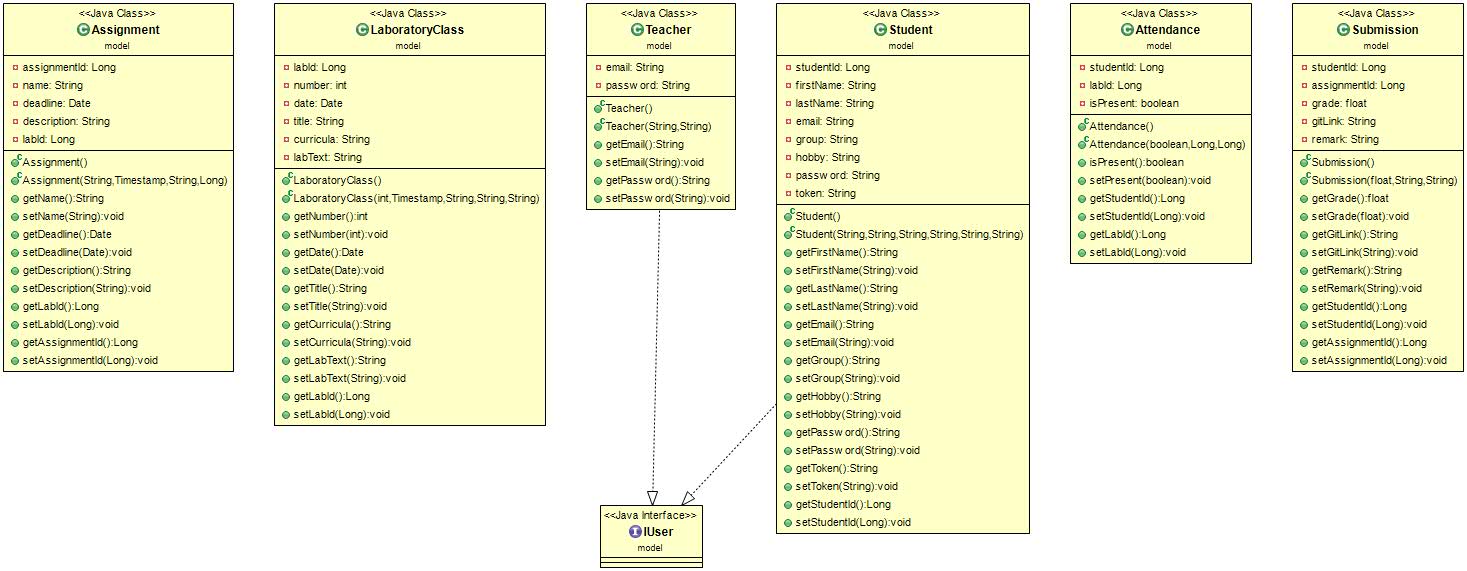


5. Class Design

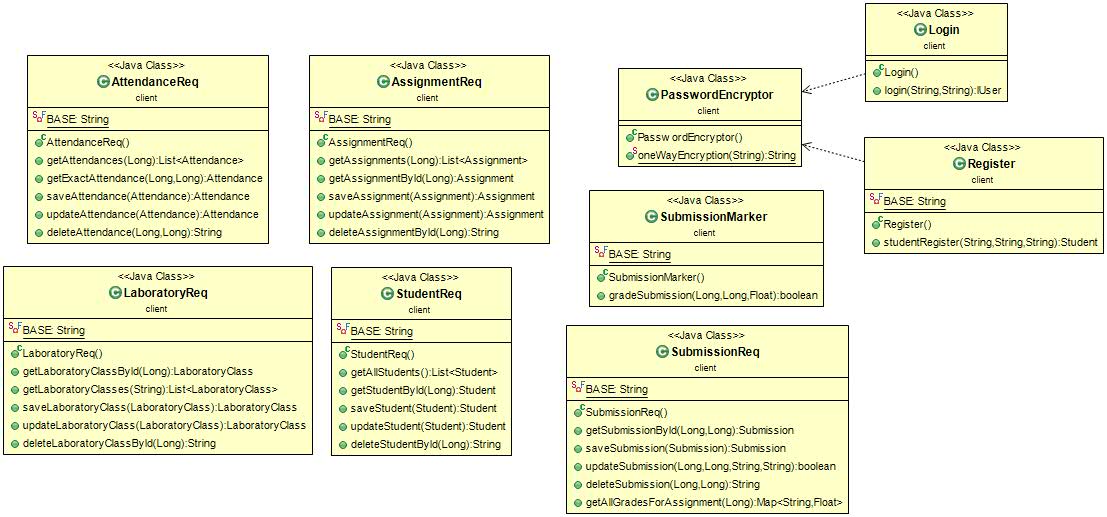
**5.1 Design Patterns Description**

The **MVP** pattern is a UI presentation pattern based on the concepts of the MVC pattern. The pattern separates responsibilities across four components: the view is responsible for rending UI elements, the view interface is used to loosely couple the presenter from its view, the presenter is responsible for interacting between the view/model, and the model is responsible for business behaviors and state management. In some implementations, the presenter interacts with a service (controller) layer to retrieve/persist the model.

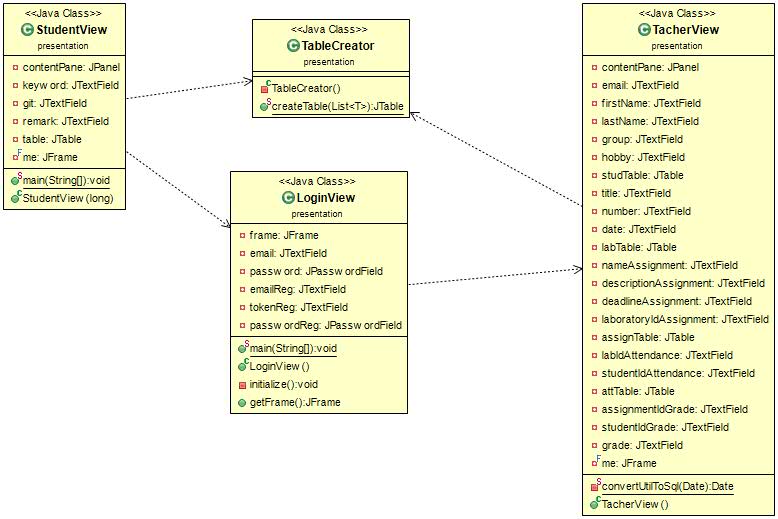
**5.2 UML Class Diagram**

**model**

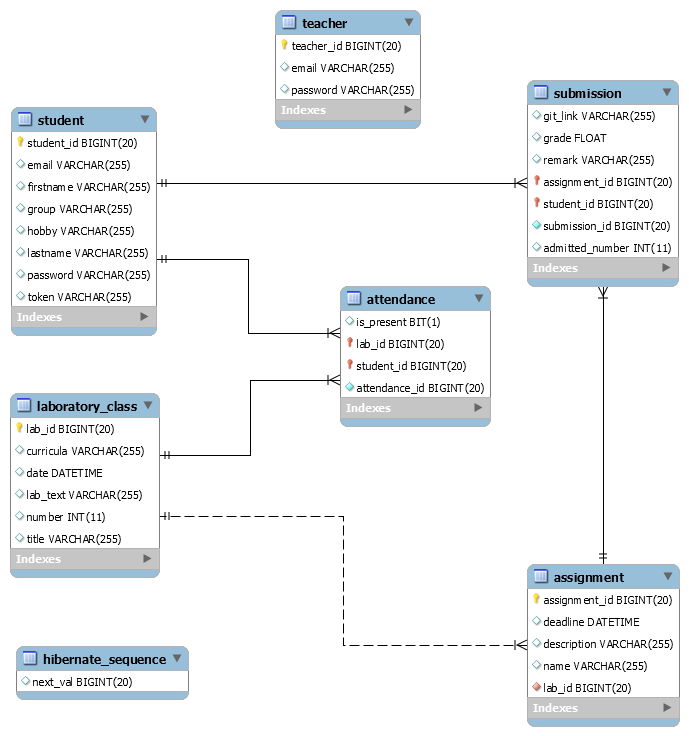
**client**

****

**presentation**

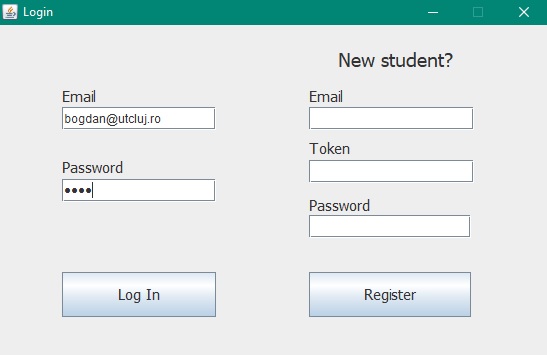
****

6. Data Model

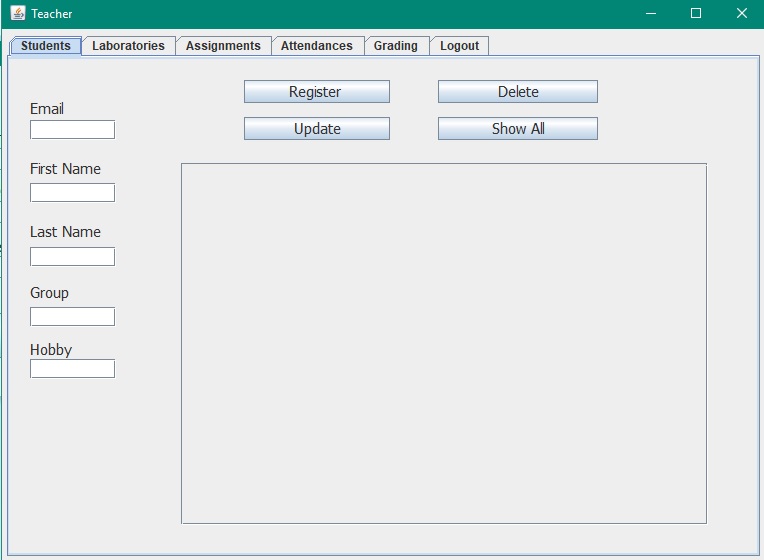
**

7. System Testing

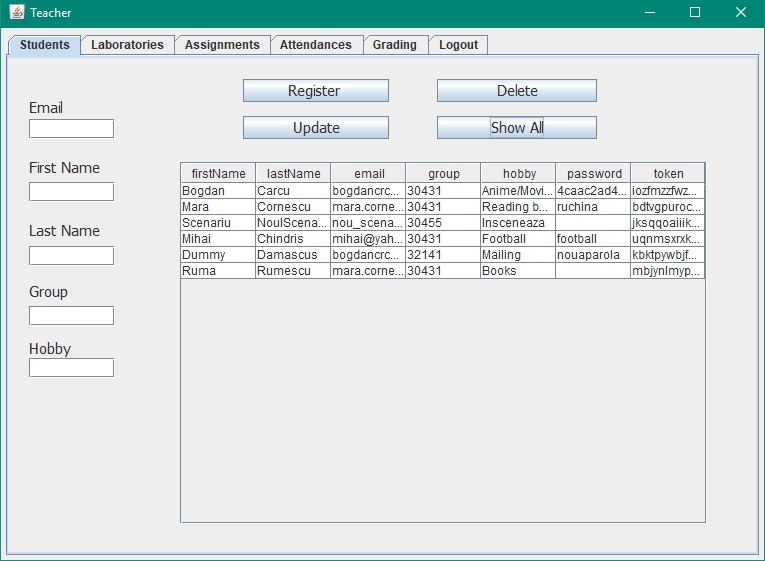
Login as a teacher



Teacher view



Teacher selects the “Show All” button. A GET Request is sent to the main App.



8. Bibliography

* RESTful Java Client with Apache Http Client

<http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-apache-httpclient/>

* MVP

<https://en.wikipedia.org/wiki/Model–view–presenter>

* Java WindowBuilder for designing Swing Applications

https://www.youtube.com/watch?v=oeswfZz4IW0

* Diagram Maker (Creately)

<https://creately.com>