Laboratory tracker
Analysis and Design Document

Student: Horvath Andrea - Anett
Group: 30434

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The purpose of this application is to obtain a laboratory tracking system for the Software Design laboratory. The application should have two types of users (students and teachers) which must provide a username and a password to use the application. The application will be implemented in Java using Spring Framework.

1.2 Functional Requirements

The teacher 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 (not part of the scope of the application). For each student we should track: email address, full name, group (ex. 30434) and hobby – free field.
- 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.
- 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 individually.

The student can perform the following operations:

- Register using the token generated by the teacher, at this step they must provide the password.
- Login with the username and password.
- View a list of laboratory classes.
- 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 comment (optional) for the teacher.

1.3 Non-functional Requirements

Performance

- Response time (Certain commands must run in < 500ms)
- Can work fine under high load (100 requests/second)

Security

- Authentication and Authorization

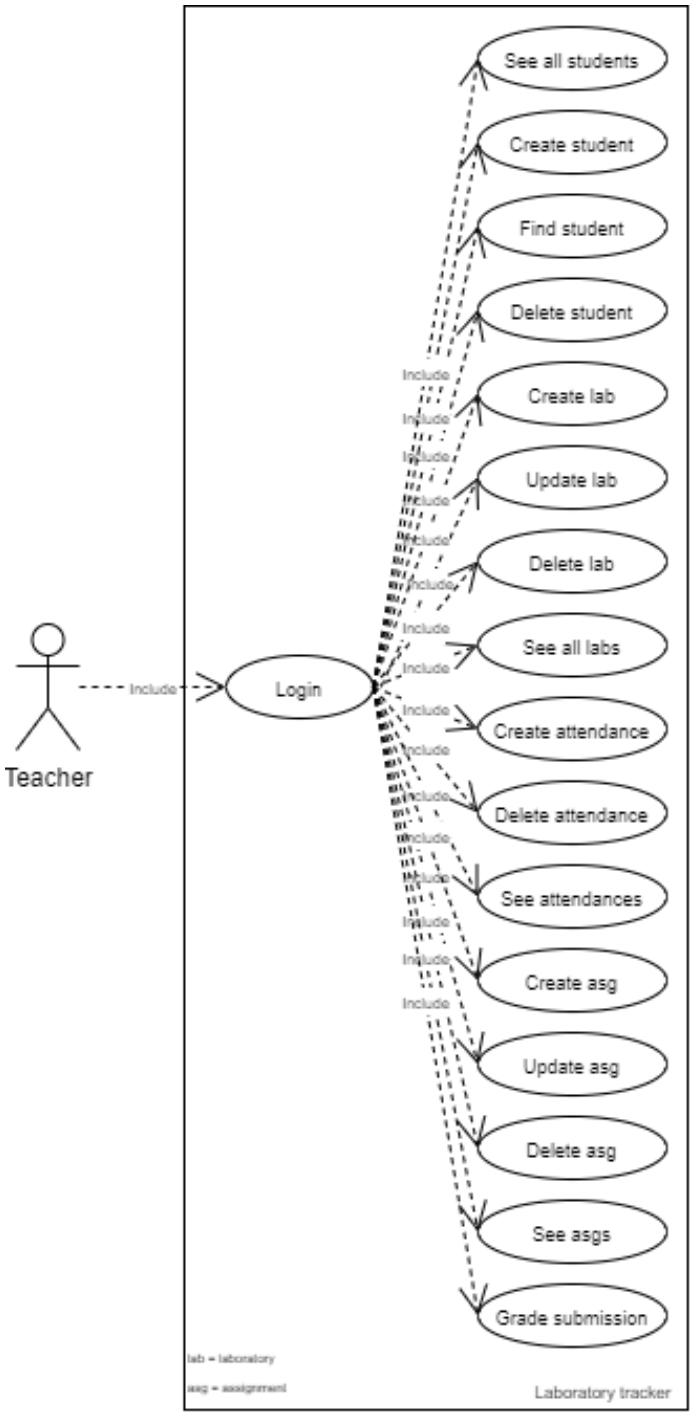
Testability

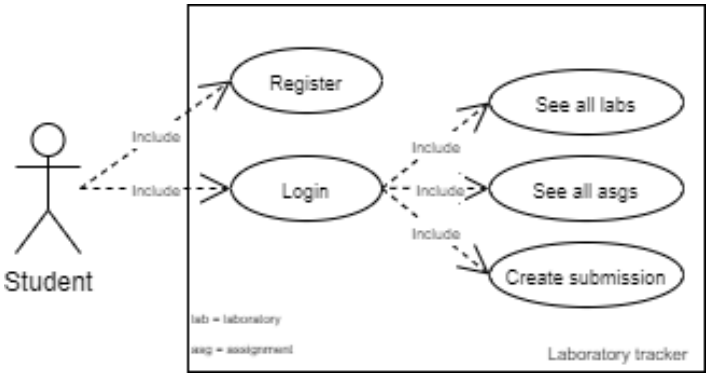
- Unit tests using Postman
- Logging using Postman

Usability

- Writing tests in Postman

2. Use-Case Model





Use case: Grade submission
Level: user-goal level
Primary actor: Teacher
Main success scenario (supposing the Teacher has already logged in successfully):

1. Select the "*Grade submission"* request from "*Submission tests"* collection in Postman.
2. As parameters enter with key *"submissionId"* the value of the id of the submission and with the key *"grade"* the value of the grade.
3. Click *"Send"* and see the result of the action in the results window.

Extensions:

1. Select the "*Grade submission"* request from "*Submission tests"* collection in Postman
2. As parameters enter with key *"submissionId"* the value of the id of an inexistent submission and with the key *"grade"* a value of an invalid grade.
3. Click *"Send"* and a http error response *400 BAD_REQUEST*.

3. System Architectural Design

3.1 Architectural Pattern Description

Model–view–controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. Traditionally used for desktop graphical user interfaces (GUIs), this pattern has become popular for designing web applications.

Model

- The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

View

- Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

Controller

- Accepts input and converts it to commands for the model or view.

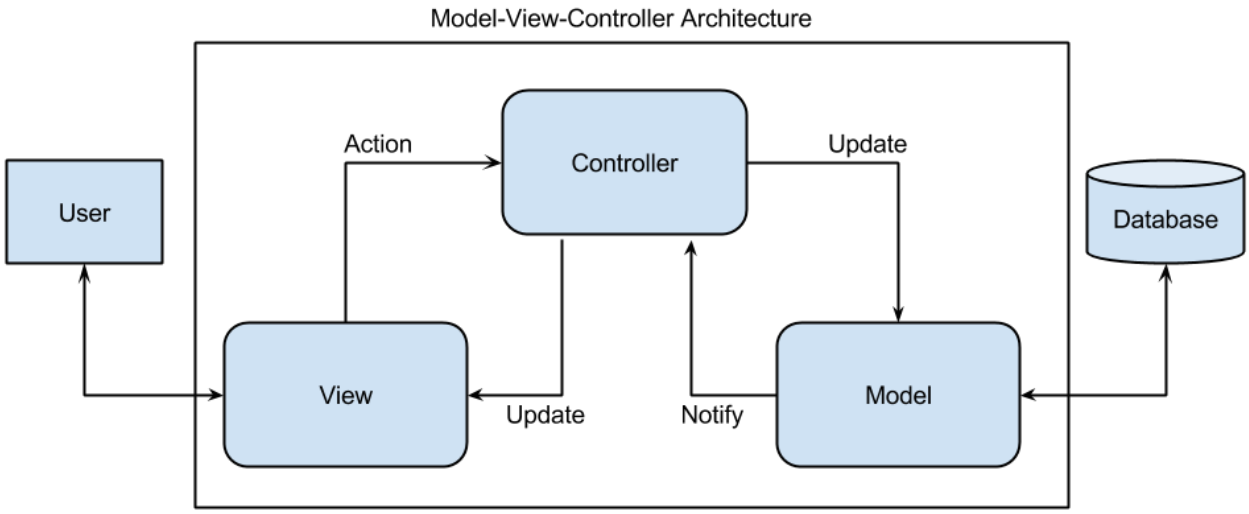
In addition to dividing the application into these components, the model–view–controller design defines the interactions between them.

- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view renders presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The

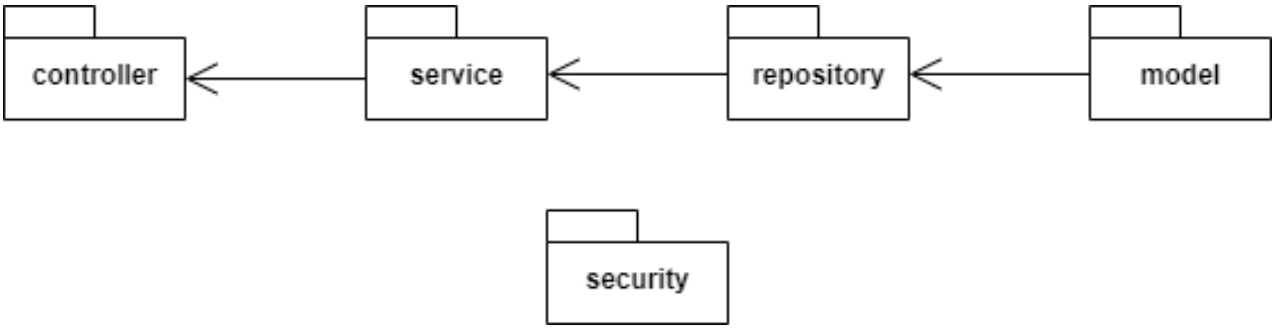
controller receives the input, optionally validates it and then passes the input to the model.

3.2 Diagrams

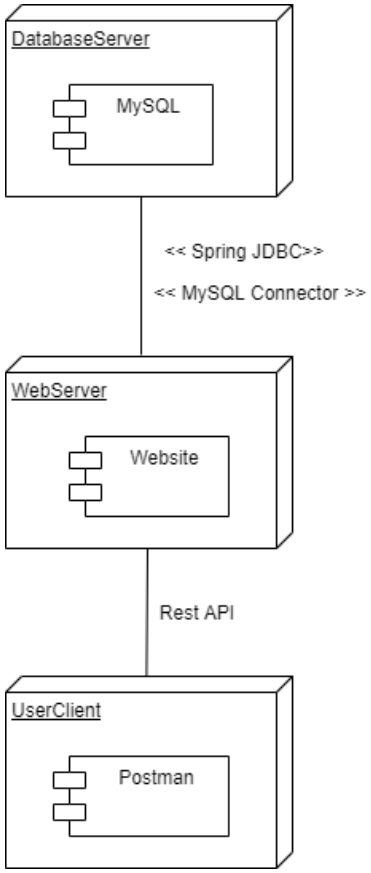
Conceptual diagram:



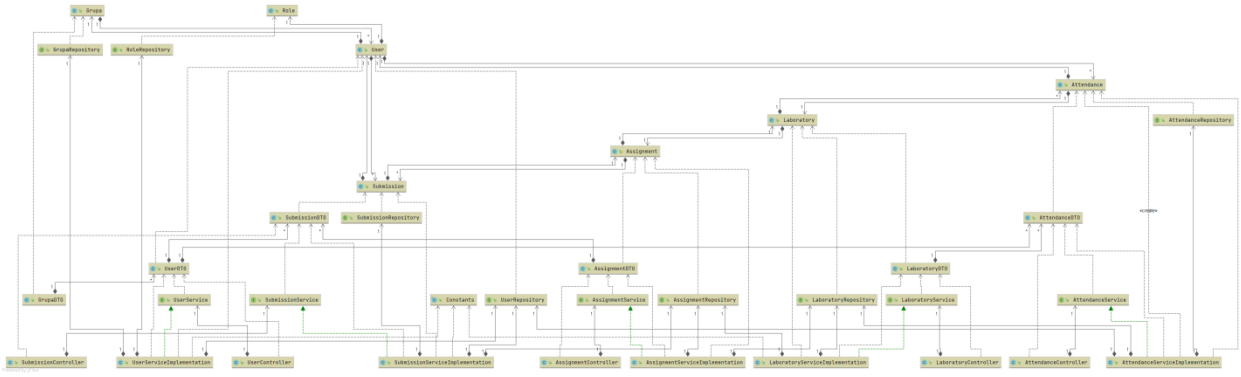
Package diagram:



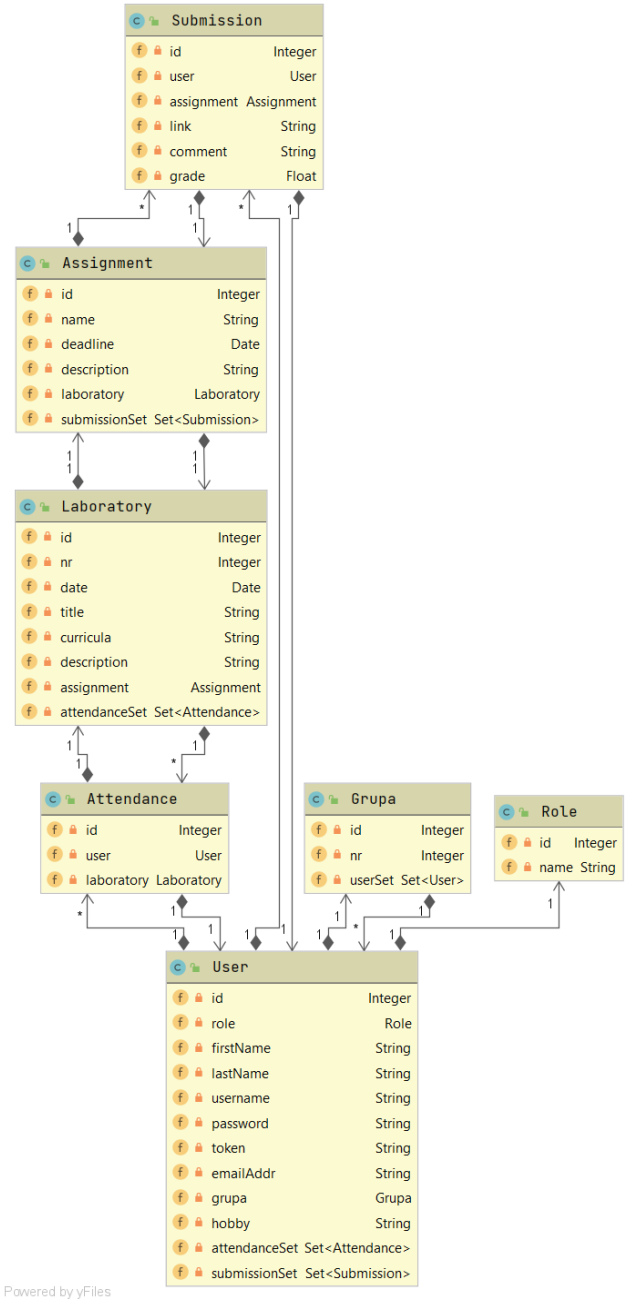
Deployment diagram:



4. Class Design



5. Data Model



6. System Testing

The testing was done using Postman.

7. Bibliography

https://spring.io/

https://www.baeldung.com/

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller