Lab Managing System

Analysis and Design Document

Student: Budea Patrick

**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

The application represents a lab managing system which could be used for tracking the status of a laboratory during a semester. The application is more of an API, since it lacks a dedicated user interface, something which can be further implemented.

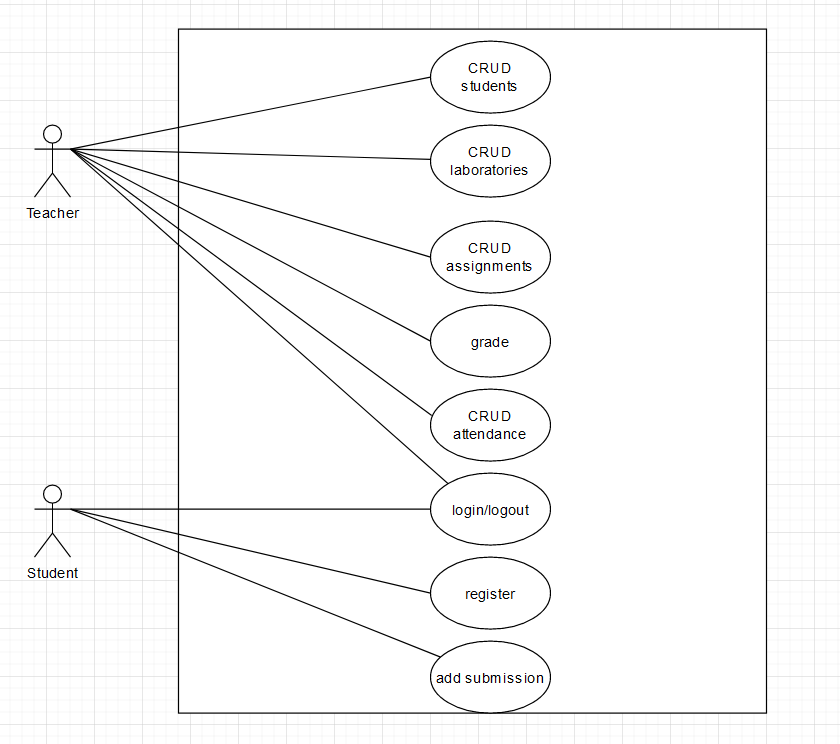
# Functional Requirements

As functional requirements, the system has to allow 2 types of users (teacher and students), which can apply CRUD specific operations on a series of entities. The system should also allow the grading of submissions by the teacher. The user credentials should also be encrypted. Data persistence is made using an SQL based system.

# Non-functional Requirements

As non-functional requirements, the system should handle multiple instances for multiple users, as long as the instances are on different ports. Since the application is using a minimal set of dependencies, the memory footprint is relatively small.

2. Use-Case Model

*]* 

Use case: add a submission

Level: ?

Primary actor: student

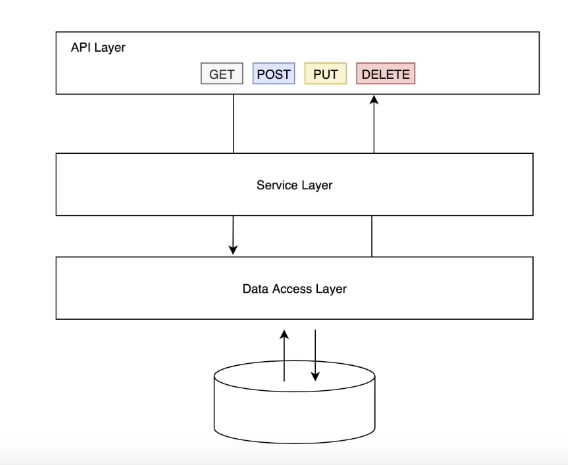
Main success scenario: the student introduces their credentials. If a successful login is achieved, the student should be able to add a submission to an existing laboratory assignment, later to be graded by the teacher.

Extensions: the system will prompt the user that their credentials are wrong, or if the selected assignment does not exist.

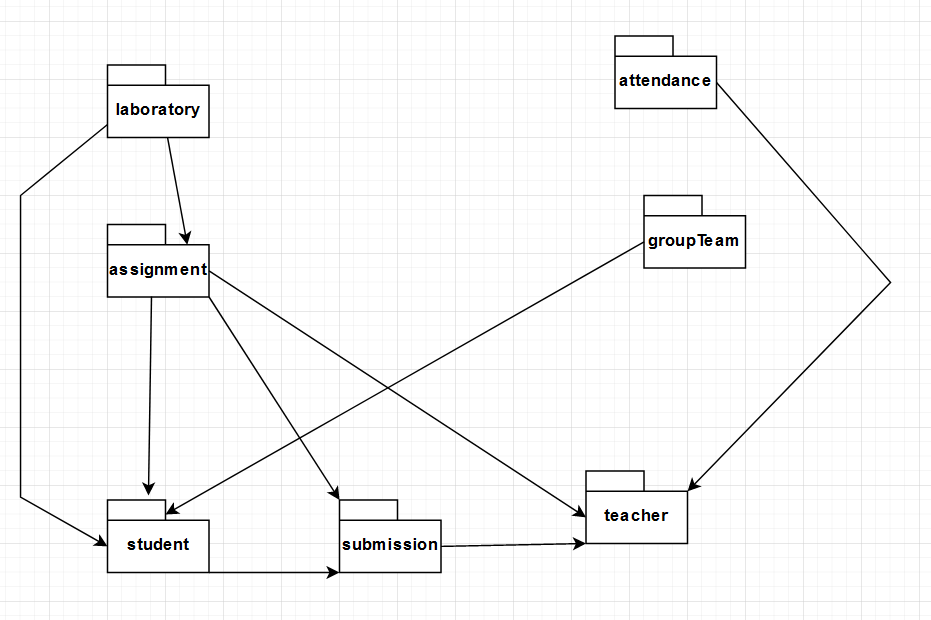
3. System Architectural Design

**3.1 Architectural Pattern Description**

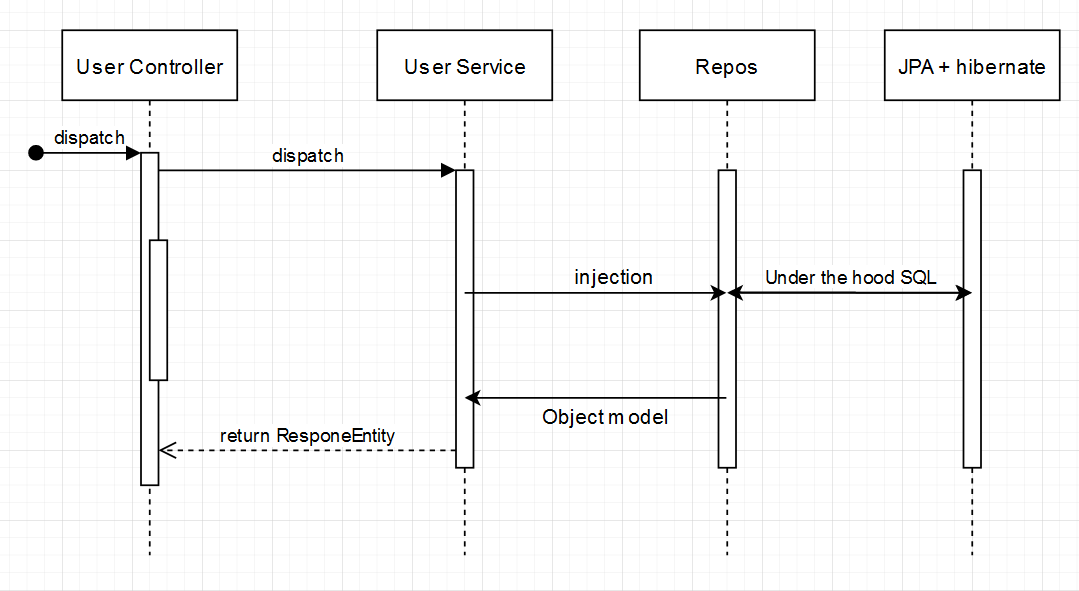
The architectural pattern employed is the MVC architecture. The system lacks a user interface, but the object modelling and the request controller is present, allowing for API calls on endpoints, specific to each operation.



**3.2 Diagrams**



4. UML Sequence Diagrams

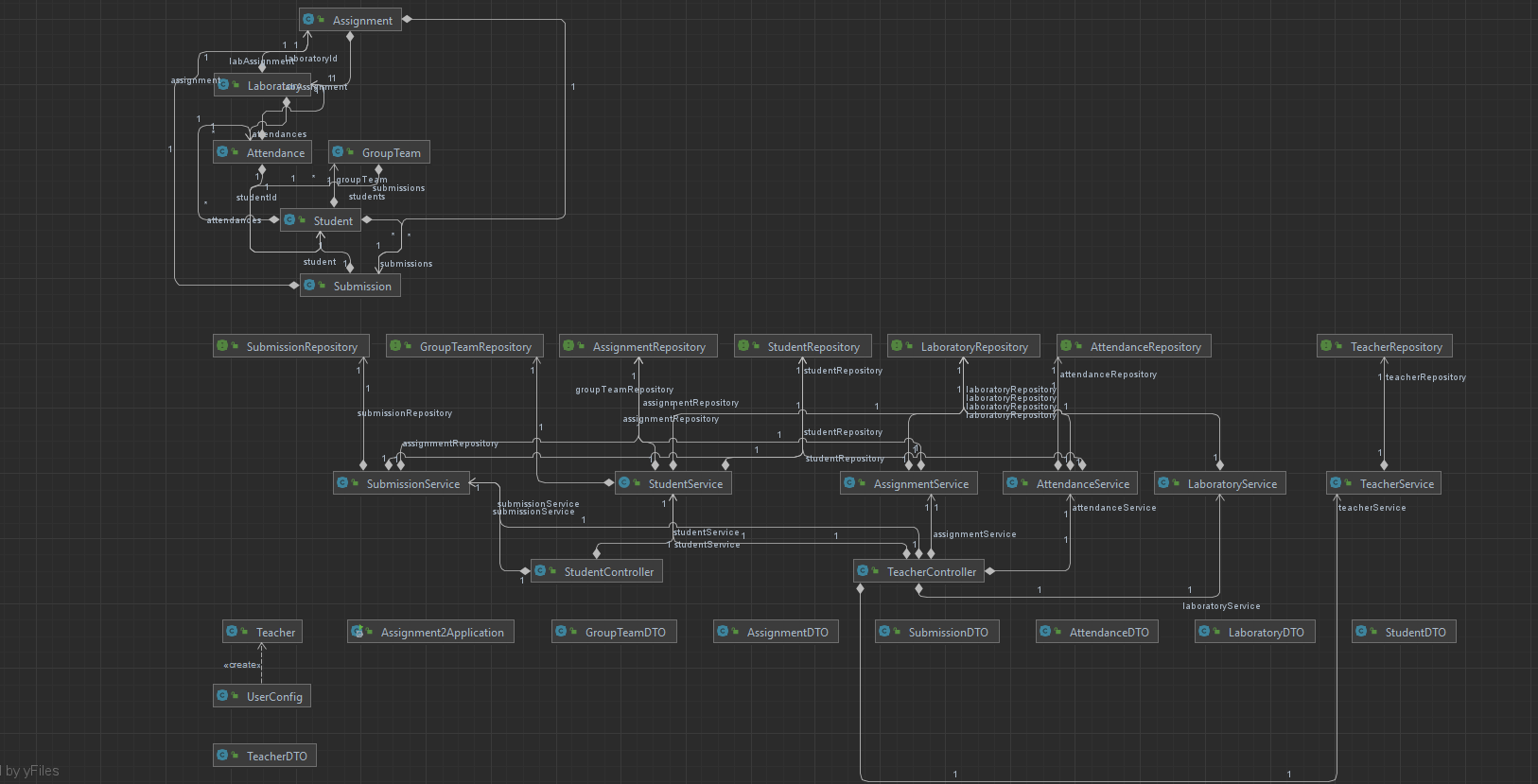


5. Class Design

**5.1 Design Patterns Description**

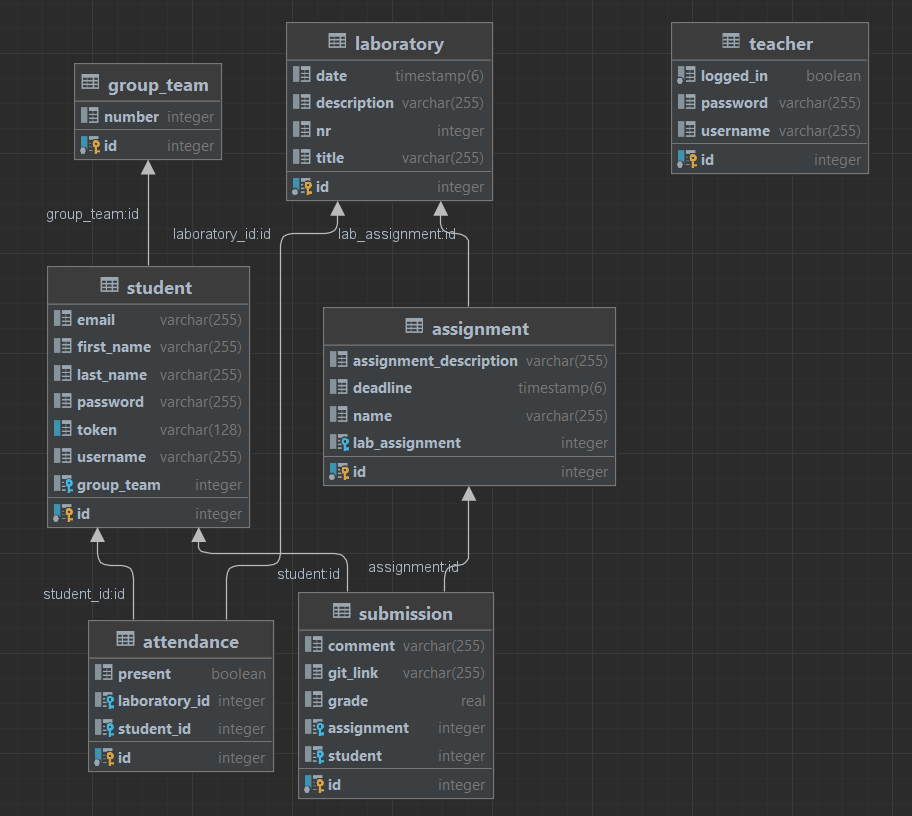
The pattern that has been used is dependency injection. This is a form of inversion of control and aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.

**5.2 UML Class Diagram**



The pattern has been used to inject the service into the REST controller and also to inject into service the specific repositories for each entity for CRUD operations and specific methods.

6. Data Model



7. System Testing

The system has been thoroughly tested by sending requests specific to the use cases. The collection is present in Insomnia.

8. Bibliography