Assignment 2

Analysis and Design Document

Student: Cosma Felicia-Iulia

**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 goal of this project is to design and implement an application for tracking laboratory activity in the Software Design laboratory. The application will be developed using either Java Spring or C# Web API. It will provide functionalities for two types of users: students and teachers. Both types of users will need to provide their email and password to access the application.

# Functional Requirements

Teacher Functionalities:

* Teachers should be able to log in to the application using their email and password.
* Teachers can create, read, update, and delete student records. Each student record includes email address, full name, group, and hobby.
* Teachers can create, read, update, and delete laboratory classes. Each class includes the laboratory number, date, title, curricula, and a long description.
* Teachers can create, read, update, and delete attendance records for each laboratory class.
* Teachers can create, read, update, and delete assignments. Each assignment includes the name, deadline, and a long description.
* Teachers can grade the submitted assignments individually.

Student Functionalities:

* Students can register using the token generated by the teacher and provide a password.
* Students can log in to the application using their username and password.
* Students can view a list of laboratory classes.
* Students can view the assignments for a laboratory class.
* Students can create an assignment submission by providing a link to a git repository and an optional short comment for the teacher.

# Non-functional Requirements

Data Storage: The application should use a relational database to store data. The database model should adhere to the 1st, 2nd, and 3rd normal forms, with proper relations between tables.

Architecture: The application should follow the MVC (Model-View-Controller) architectural pattern. The backend part should include the Model, Controller, Services (Business layer), and Repositories.

API Design: The API design should be RESTful, not SOAP, to ensure interoperability and ease of integration.

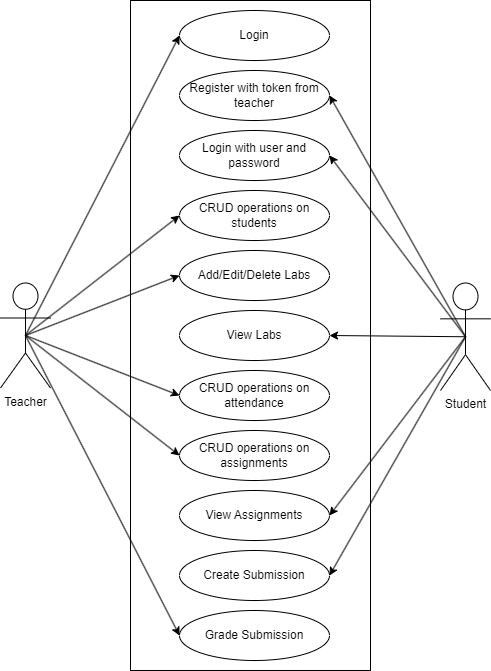
Object-Relational Mapping: An ORM (Hibernate or Entity Framework) should be used to access the database and manage the object-relational mapping.

Dependency Injection: Dependency injection should be employed to inject Services in Controllers and Repositories in Services, promoting modularity and testability.

Documentation: Install and use Swagger to provide an interactive documentation for the APIs. Alternatively, a Postman collection can be provided for API testing and documentation.

Configuration Management: Connection strings and other configuration values should be stored in a separate configuration file to facilitate easy management and deployment.

2. Use-Case Model



Use Case: Create Student

Level: User-goal level

Primary Actor: Teacher

Main Success Scenario:

* The use case starts when the teacher wants to create a new student record.
* The teacher provides the necessary information for the student, including the email address, full name, group, and hobby.
* The system validates the input data.
* The system creates a new student record with the provided information.
* The system generates a 128-character token for the student.
* The system associates the generated token with the student record.
* The system notifies the teacher that the student has been created successfully.

3. System Architectural Design

**3.1 Architectural Pattern Description**

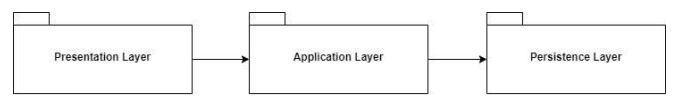
The chosen architectural pattern for the Laboratory Activity Tracking Application is the Model-View-Controller (MVC) pattern.

MVC separates the application into three interconnected components:

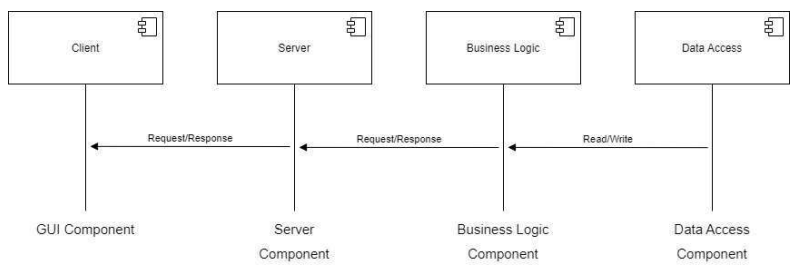
* Model: Represents the data and business logic of the application. In this case, the model consists of entities such as students, laboratory classes, assignments, and attendance records. The model is responsible for managing the data and implementing the business rules.
* View: Handles the presentation and user interface of the application. It provides a way for users (teachers and students) to interact with the system. In this application, the views would include login screens, student registration forms, lists of laboratory classes, and assignment submission forms.
* Controller: Acts as the intermediary between the model and the view. It receives input from the users via the view, interacts with the model to retrieve or update data, and updates the view accordingly. The controller in this application would handle operations such as login, CRUD operations on students, laboratory classes, attendance, assignments, and grading.
* The MVC pattern helps in achieving separation of concerns, making the application modular and easier to maintain. It allows for independent development of the model, view, and controller components, facilitating code reusability and testability.

**3.2 Diagrams**

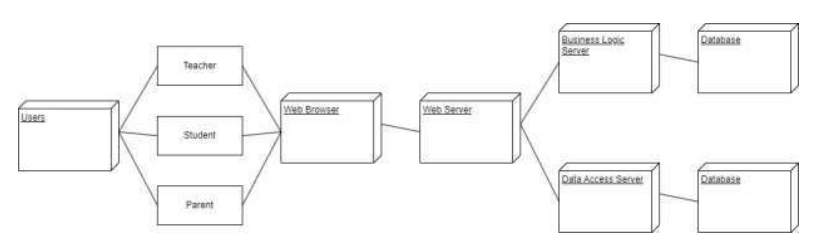
* package diagram



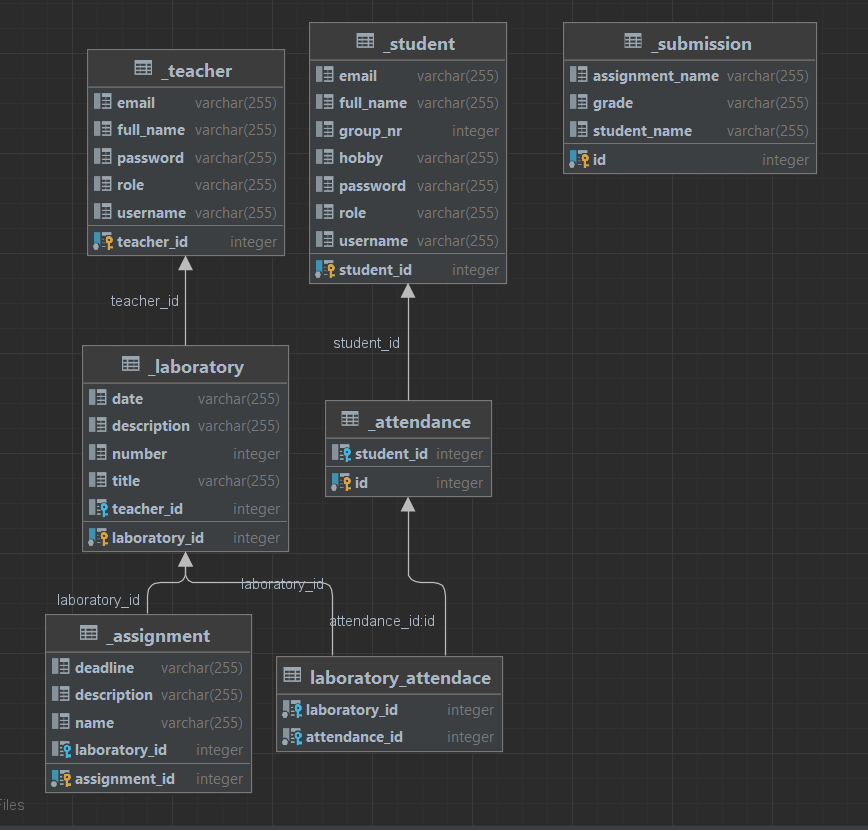
* component diagram



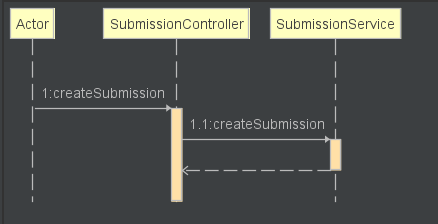
* deployment diagram



* database



4. UML Sequence Diagrams

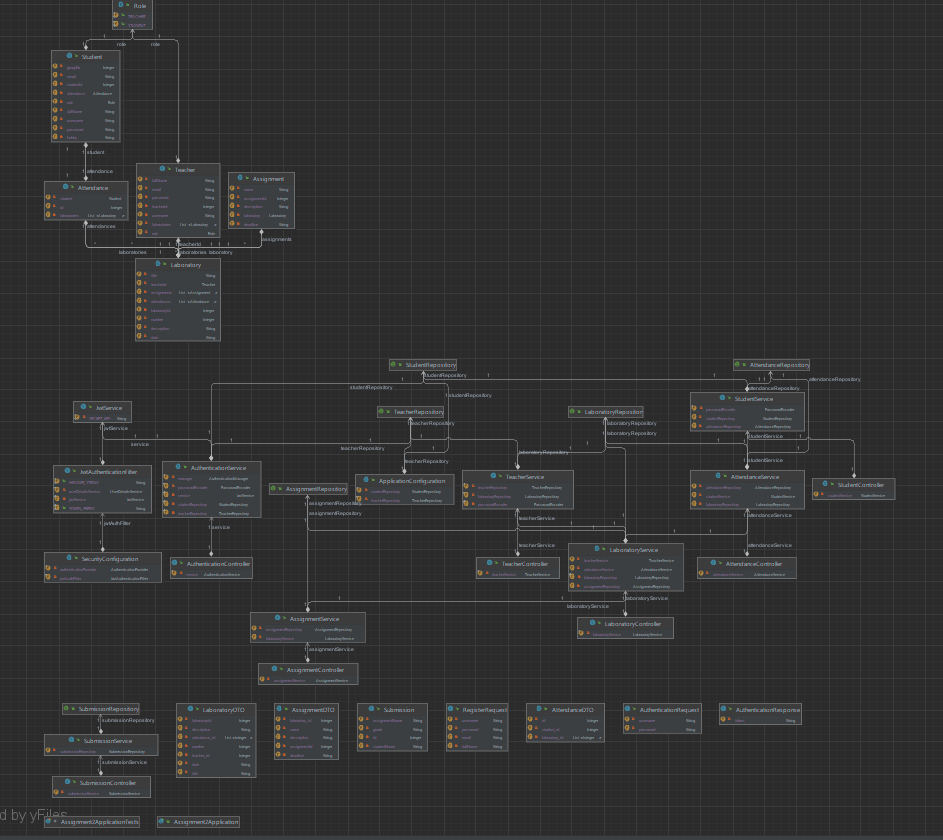
** Create Submission

5. Class Design

**5.1 Design Patterns Description**

The MVC pattern separates the application into three interconnected components: Model, View, and Controller. The Model represents the data and business logic, the View handles the presentation and user interface, and the Controller acts as the intermediary between the Model and View, handling user input and updating the Model and View accordingly. This pattern promotes separation of concerns and modularity.

**5.2 UML Class Diagram**



6. Data Model

* Student:

Attributes: email address, full name, group, hobby

Relationships: None

* Teacher:

Attributes: email address, password

Relationships: None

* LaboratoryClass:

Attributes: laboratory number, date, title, curricula, long description

Relationships: None

* Attendance:

Attributes: student, laboratory class, attendance status

Relationships: Many-to-One relationship with Student and LaboratoryClass

* Assignment:

Attributes: name, deadline, long description

Relationships: Many-to-One relationship with LaboratoryClass

* AssignmentSubmission:

Attributes: student, assignment, git repository link, comment

Relationships: Many-to-One relationship with Student and Assignment

* GradedAssignment:

Attributes: assignment submission, grade

Relationships: One-to-One relationship with AssignmentSubmission

7. Bibliography