<Laboratory activity system>

Student: Tusa Laurentiu

**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 4

3. System Architectural Design 5

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 10

7. Bibliography 10

1. Requirements Analysis

# Assignment Specification

The laboratory activity application should allow students to upload their work for a given lab assignment, view their submissions, and receive grades for their work. Teachers should be able to track the submissions and attendances of students, grade the assignments, and communicate feedback to students. The application should have different access levels for students and teachers, with appropriate user authentication and authorization mechanisms. Additionally, the application should have a user-friendly interface for easy navigation and use.

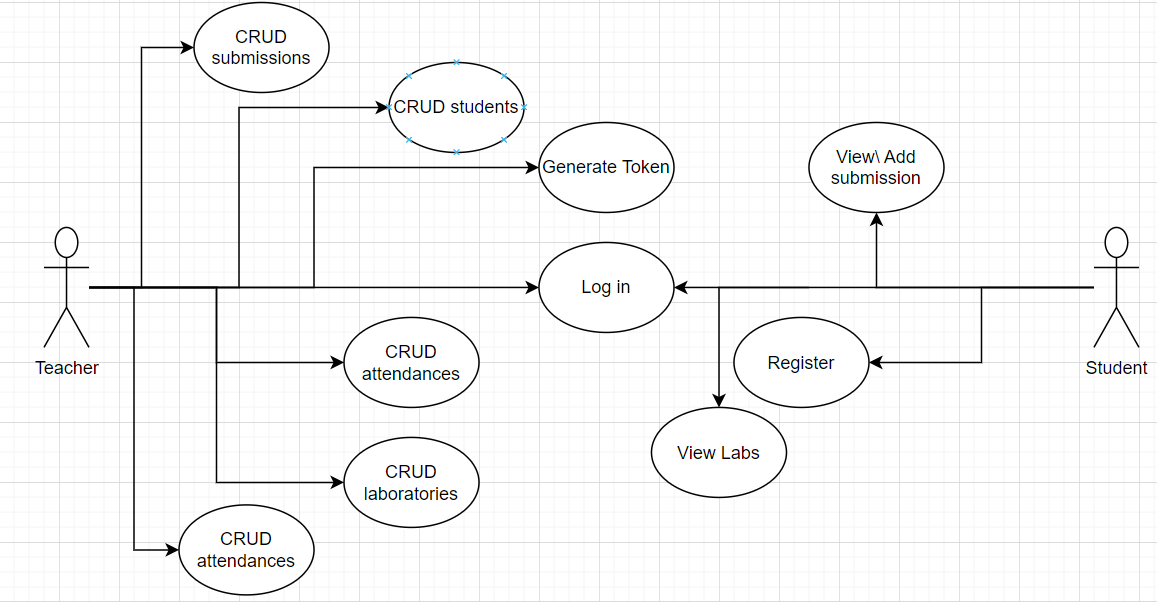
# Functional Requirements

* User Authentication: The app should allow students and teachers to create their accounts and log in securely with encrypted passwords.
* Student Submissions: Students should be able to upload their work for a particular laboratory activity by specifying the name of the laboratory activity, the submission file(s), and any additional information about their submission.
* Submission Tracking: The app should allow teachers to see a list of all submissions for a particular laboratory activity, with the ability to filter by submission date and student name.
* Attendance Tracking: The app should allow teachers to mark attendance for each student for a particular laboratory activity.
* Grading: The app should allow teachers to assign grades to each submission, based on a predefined grading rubric. Teachers should be able to see a list of all submissions for a particular laboratory activity with grades assigned, and the grades should be visible to students as well.
* Notifications: The app should notify students when their submissions have been graded, and allow them to view their grades and feedback from the teacher.
* Security: The app should ensure the security of the stored data, by implementing secure password encryption and taking measures to prevent unauthorized access to the system.
* CRUD on students, teachers, attendances, laboratories, submissions, grades.

# Non-functional Requirements

* Security: The application should have secure password storage and transmission of data to ensure confidentiality of user information.
* Performance: The application should have fast response times and minimal downtime to ensure users can access it whenever they need to.
* Scalability: The application should be able to handle a large number of users and submissions without impacting performance or availability.
* Usability: The application should be intuitive and easy to use for both students and teachers, with clear navigation and minimal user errors.
* Reliability: The application should have high availability and be able to recover quickly in case of any failures or errors.
* User Experience: The application should provide an enjoyable and seamless user experience, with clear and concise instructions, user-friendly interfaces, and minimal load times.

2. Use-Case Model



Use case: Student submits a lab assignment Level: User-goal level Primary actor: Student

Main success scenario:

1. Student logs into the system with valid credentials.
2. The system displays a list of lab assignments available for submission.
3. Student selects the lab assignment they want to submit.
4. The system displays the lab assignment details and a submission form.
5. Student uploads their lab assignment file to the system.
6. Student clicks on the "Submit" button.
7. The system saves the submission to the database.
8. The system returns to the laboratories page.

Extensions:

* 2a. If the student has not yet registered or their credentials are wrong, the system displays an error message. Go back to step 1.
* 5a. If the student uploads a file that is not in the required format, the system displays an error message. Go back to step 5.

3. System Architectural Design

**3.1 Architectural Pattern Description**

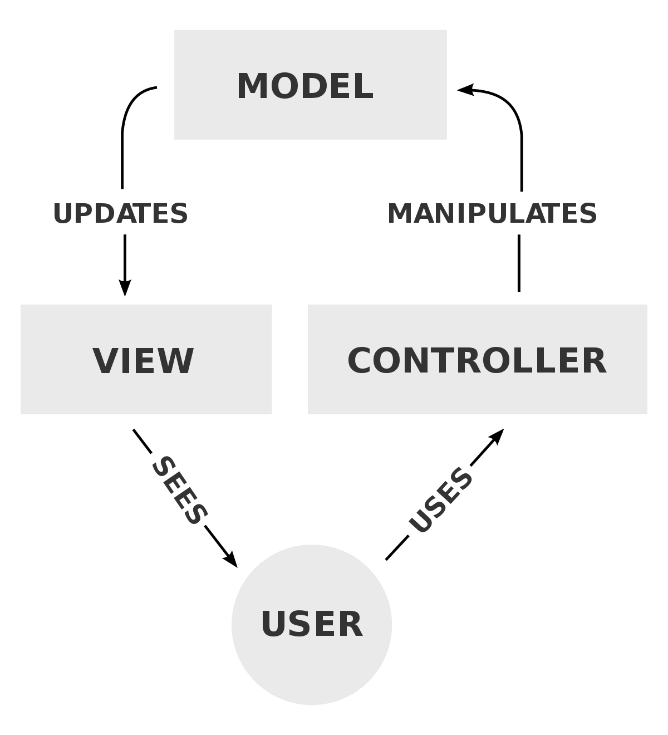
The architectural pattern used in this application is Model-View-Controller (MVC). This pattern separates the concerns of the application into three distinct components:

1. Model: The model represents the application's data and business logic. In this case, Microsoft.EntityFrameworkCore is used to interact with the database and define the data models.
2. View: The view is responsible for rendering the user interface of the application. In this case, Razor Views are used to generate HTML markup that is returned to the user's browser. Views for List display, update, create were used.
3. Controller: The controller is responsible for handling incoming requests, processing them, and returning a response. In this case, ASP.NET Core 6 controllers are used to handle requests from the user's browser, interact with the model to retrieve and modify data, and return the appropriate view to the user.

Overall, the MVC pattern provides a structured approach to building web applications that allows for separation of concerns and flexibility in development.

**3.2 Diagrams**

MVC Design pattern

******

In this MVC ASP.NET Core 6 application, the Asg2.DAL folder would typically contain the data access layer, which includes classes that handle data storage and retrieval from a database.

The Asg2.BLL folder would contain the business logic layer, which includes classes that perform operations on the data retrieved from the database.

The Asg2 folder would contain the controllers and views, which implement the presentation layer of the application.

Here's a breakdown of how the MVC pattern is applied in an ASP.NET Core 6 application with the given folder structure:

* Asg2.DAL: This folder would typically contain classes that implement the repository pattern or some other data access pattern. These classes would handle data storage and retrieval from a database. They would be responsible for interacting with the data context and mapping data to model objects.
* Asg2.BLL: This folder would typically contain service classes that perform business logic operations on the data retrieved from the database. These classes would be responsible for implementing the logic that processes the data retrieved from the database.
* Asg2.Controllers: This folder would contain classes that implement the controllers for the application. Each controller would be responsible for handling HTTP requests, processing input from the user, and returning an appropriate response. They would interact with the service classes in the BLL layer to perform business logic operations on the data.
* Asg2.Views: This folder would contain the views for the application. Each view would be responsible for displaying data to the user. They would interact with the controllers to retrieve data and receive updates from the user.

Overall, the MVC pattern helps to separate the concerns of the application into distinct components, making the application more modular and easier to maintain.

Package Diagram

O imagine care conține diagramă

Descriere generată automat

Deployment diagram:

O imagine care conține diagramă

Descriere generată automat

4. UML Sequence Diagrams

*O imagine care conține diagramă

Descriere generată automat*

5. Class Design

**5.1 Design Patterns Description**

The MVC architectural pattern stands for Model-View-Controller and is widely used for building web applications. In this pattern, the application is divided into three interconnected components: the Model, the View, and the Controller.

The Model represents the application data and business logic, the View displays the data to the user, and the Controller handles user input and updates the Model and View accordingly.

For a ASP.NET Core 6 application with the given folder structure, we can apply the MVC pattern in the following way:

Asg2.DAL: This folder contains the data access layer of the application. It uses the Microsoft.EntityFrameworkCore.SqlServer framework to interact with a SQL Server database. The models and database context are defined here. The Model component of the MVC pattern is implemented in this layer.

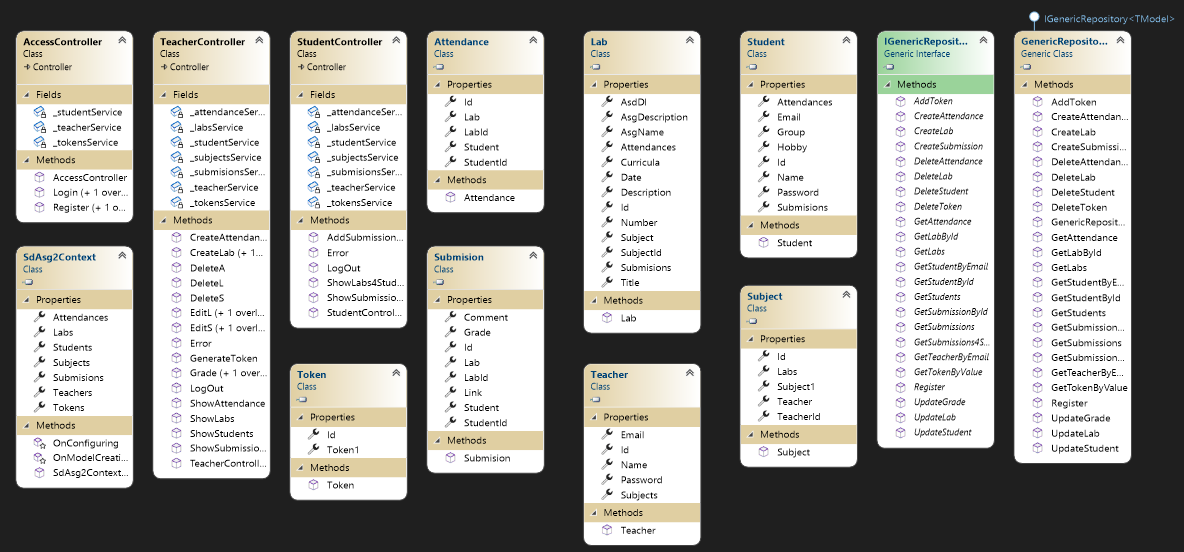
Asg2.BLL: This folder contains the business logic of the application. It includes Services that provide operations on the data models. The Services receive data from the DAL layer, process it as required, and return it to the Controllers. The Model component is further extended in this layer.

Asg2: This folder contains the Controllers and Views of the application. The Controllers receive user input and use the Services from the BLL layer to interact with the Model and update the View accordingly. The View is responsible for presenting the data to the user in a user-friendly way. The Controller and View components are implemented in this layer.

Overall, the MVC pattern helps to separate concerns in the application and promotes modularity, maintainability, and testability.

In summary, the Asg2.DAL layer handles the data, the Asg2.BLL layer handles the business logic, and the Asg2 layer handles the presentation layer. Together, they form a complete ASP.NET Core 6 MVC application.

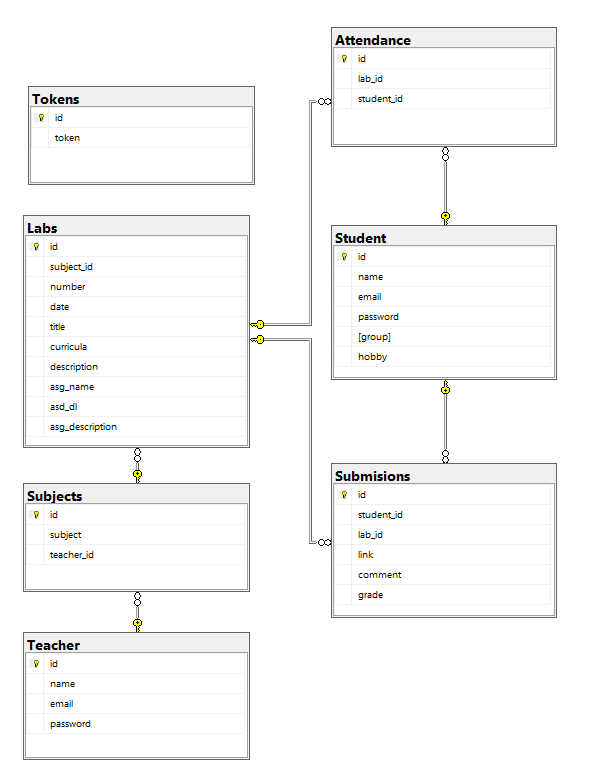
**5.2 UML Class Diagram**

**

O imagine care conține text

Descriere generată automat

6. Data Model



7. Bibliography

https://www.youtube.com/watch?v=j2AYkOSzTUw

<https://app.diagrams.net/>

<https://www.youtube.com/watch?v=xxSDmYumUwU>

https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-6.0?view=aspnetcore-7.0