Laboratory Activity
Analysis and Design Document

Student: Bar Luca-Narcis
Group: 30431/1

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The application is a Lab Class Management System that allows teachers to create, manage and grade lab classes. Students can view their class schedules and submit their lab reports to the teacher.
For this specific assignment I've decided to use C# and the .NET framework for the implementation, and PostgreSQL as the database server. I've also used Entity Framework to generate ORMs from my database and BCrypt's hashing and verifying methods for password encryption and security.

1.2 Functional Requirements

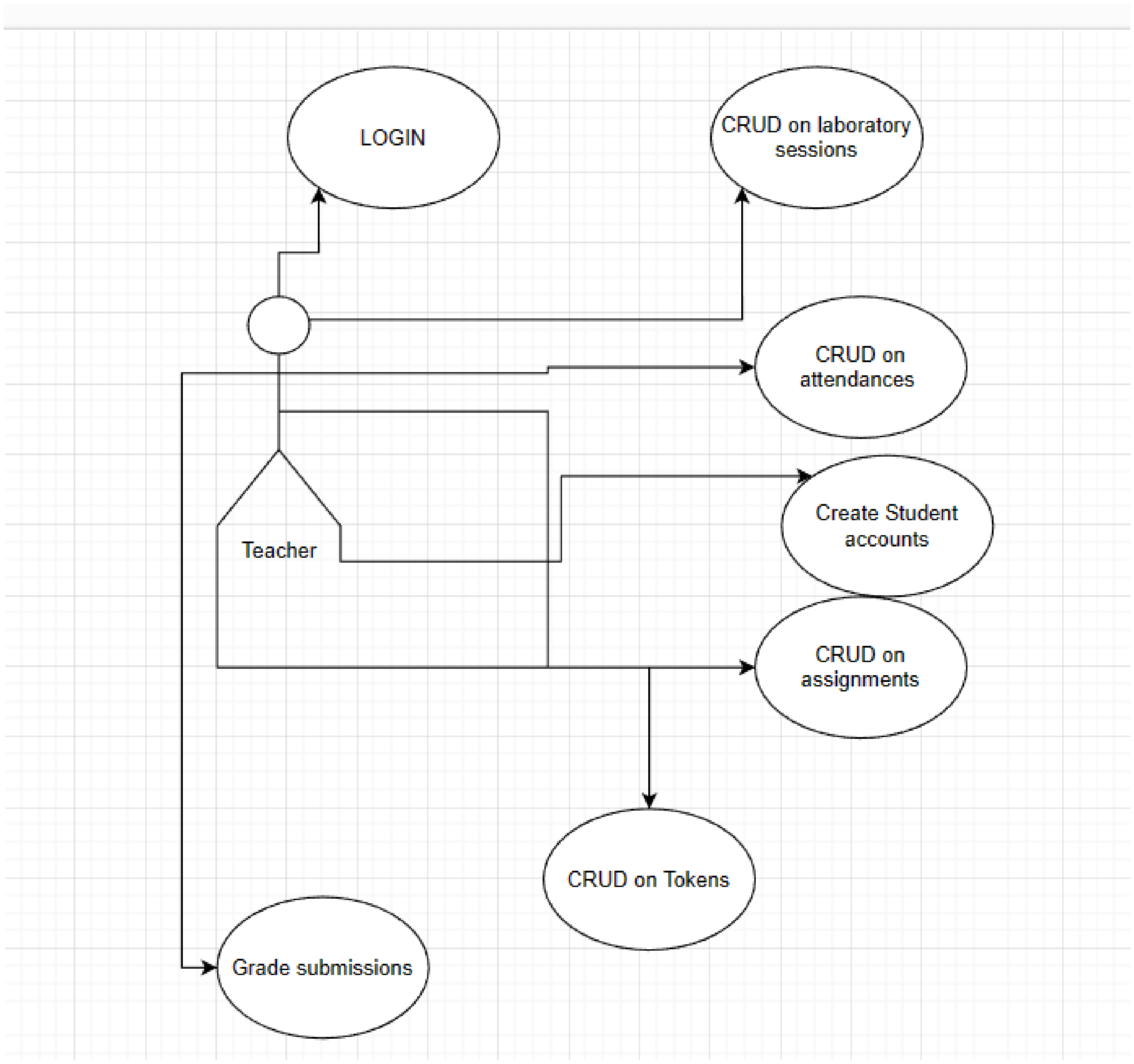
Teachers should be able to login.
Teachers should be able to create, edit and delete a Lab Class with a unique number, date, title, curricula, and lab text.
Teachers should be able to add, delete, and edit students from a Lab Class.
Teachers should be able to view a list of students who attended a Lab Class.
Teachers should be able to grade lab submissions.
Students should be able to view their Lab Class schedule.
Students should be able to submit lab reports to the teacher.
Students should be able to register with a token given by the teacher.

1.3 Non-functional Requirements

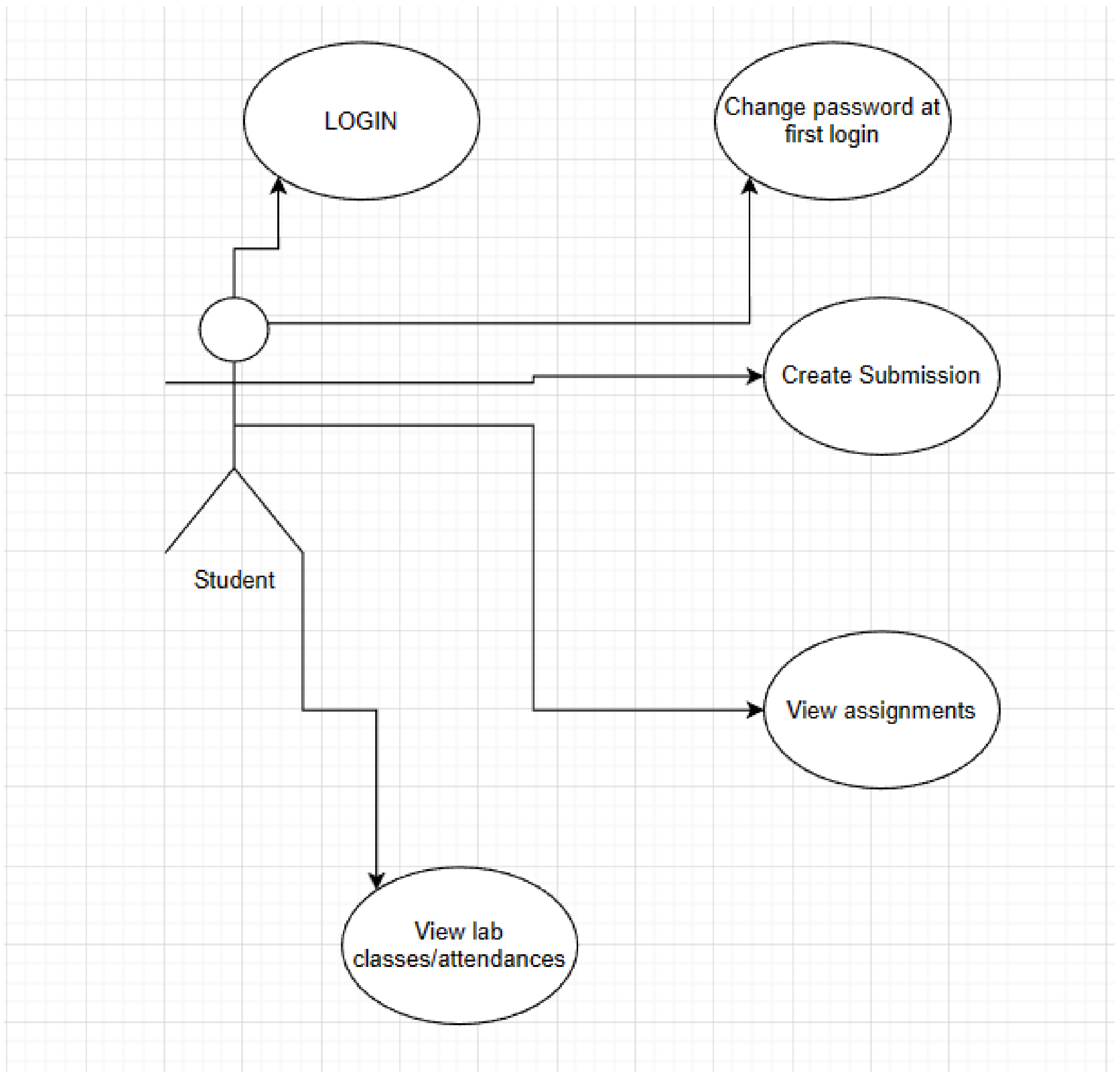
- Data will be stored in a relational database.
- Use the Layers architectural pattern to organize your application.
- Passwords must be encrypted when stored to the database with a one-way encryption algorithm (BCrypt Hashing).
- Postman collection of operations
- ORM Hibernate framework to access the database
- API design should be RESTful, not SOAP

2. Use-Case Model

Use-case diagram for the Teacher :



Use case diagram for the Student:



Use case: Adding a new student
Level: user goal level
Primary actor: teacher
Main success scenario:

1.Teacher logs in.
2.Teacher navigates to the create student page
3.Teacher fills the data fields with the student's name, email and hobby and selects a token to use as first-time password from a dropdown list with the unused tokens.
4.Student is created successfully.

Extensions:

a)Teacher enters wrong credentials, resulting in an unsuccessful login.
b)There may be no unused tokens, which means that they need to create a new one.

c)Teacher might add incorrect data types in the fields, which is not allowed.
d)The database connection might fail, resulting in an unsuccessful attempt at creating the student.
e)The student might already be created, resulting in an error.

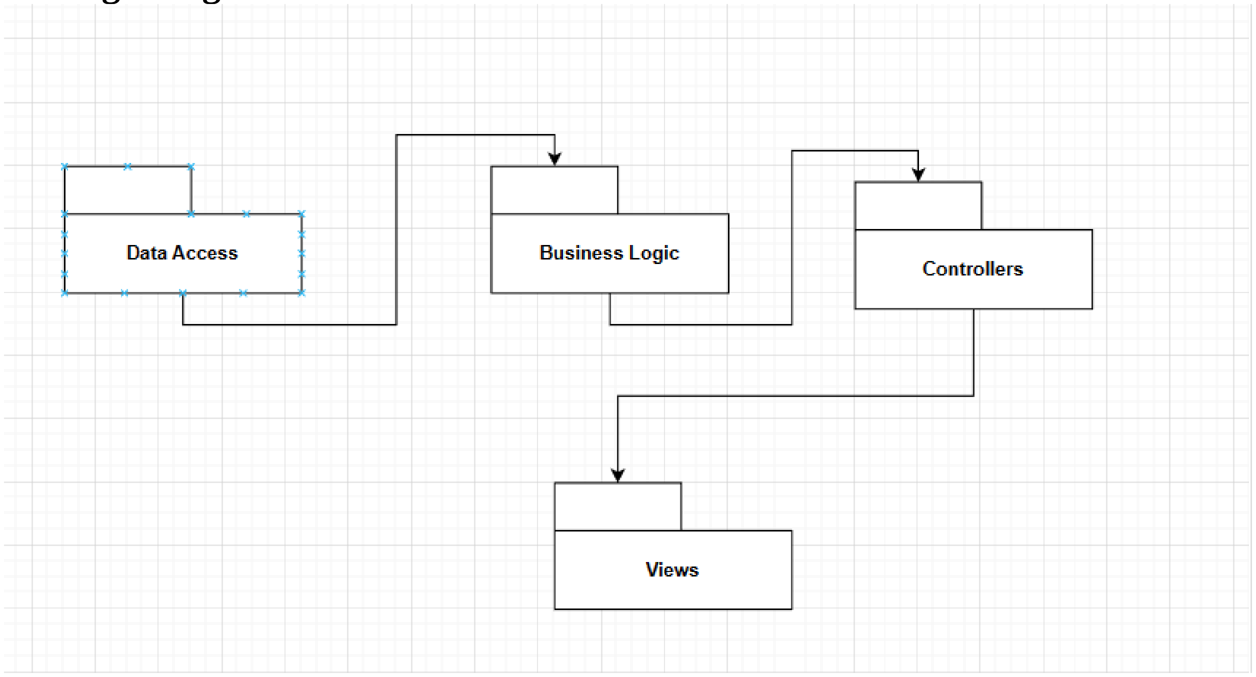
3. System Architectural Design

3.1 Architectural Pattern Description

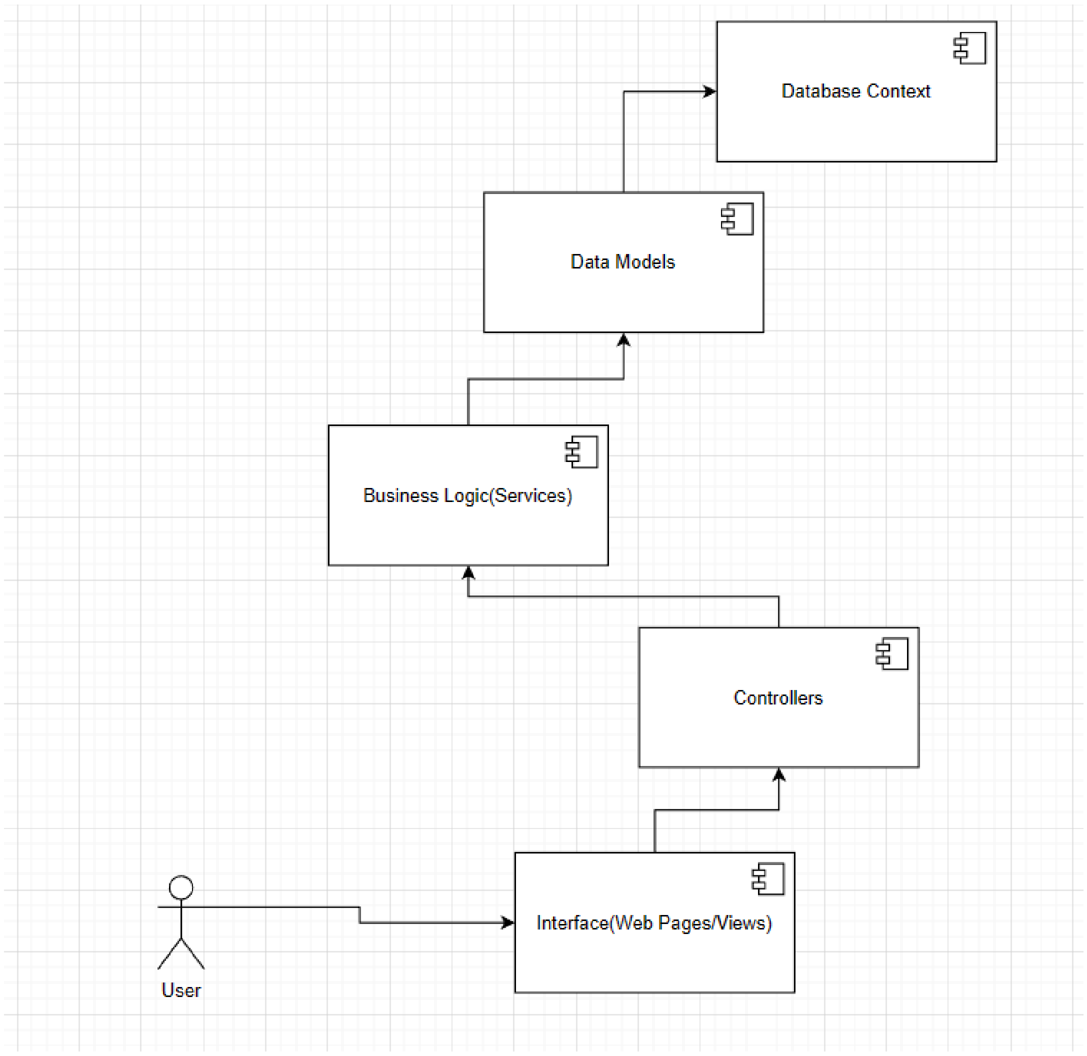
The Model-View-Controller (MVC) is a software architecture pattern commonly used in designing applications with graphical user interfaces (GUIs). It separates an application into three main interconnected components: the Model, the View, and the Controller.
In an MVC layered architecture, these three components are organized in layers, where each layer has a specific responsibility. The Model typically resides in the bottom layer, the View in the top layer, and the Controller in the middle layer. The Model and View are loosely coupled, meaning that changes in one component do not directly affect the other. The Controller acts as an intermediary, facilitating communication between the Model and View while keeping them separate. This separation of concerns promotes maintainability, reusability, and scalability in software development.

3.2 Diagrams

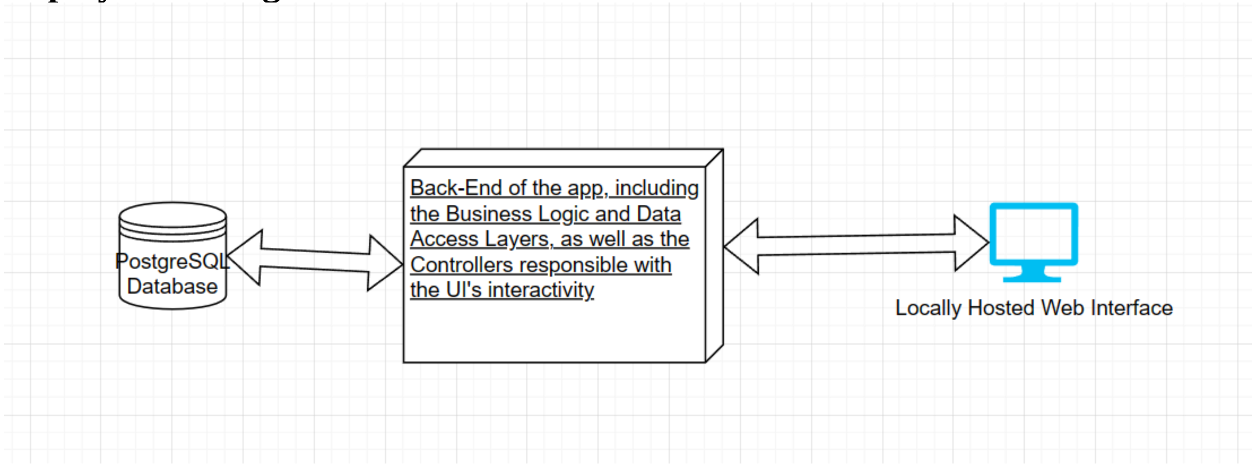
Package Diagram:



Component Diagram:

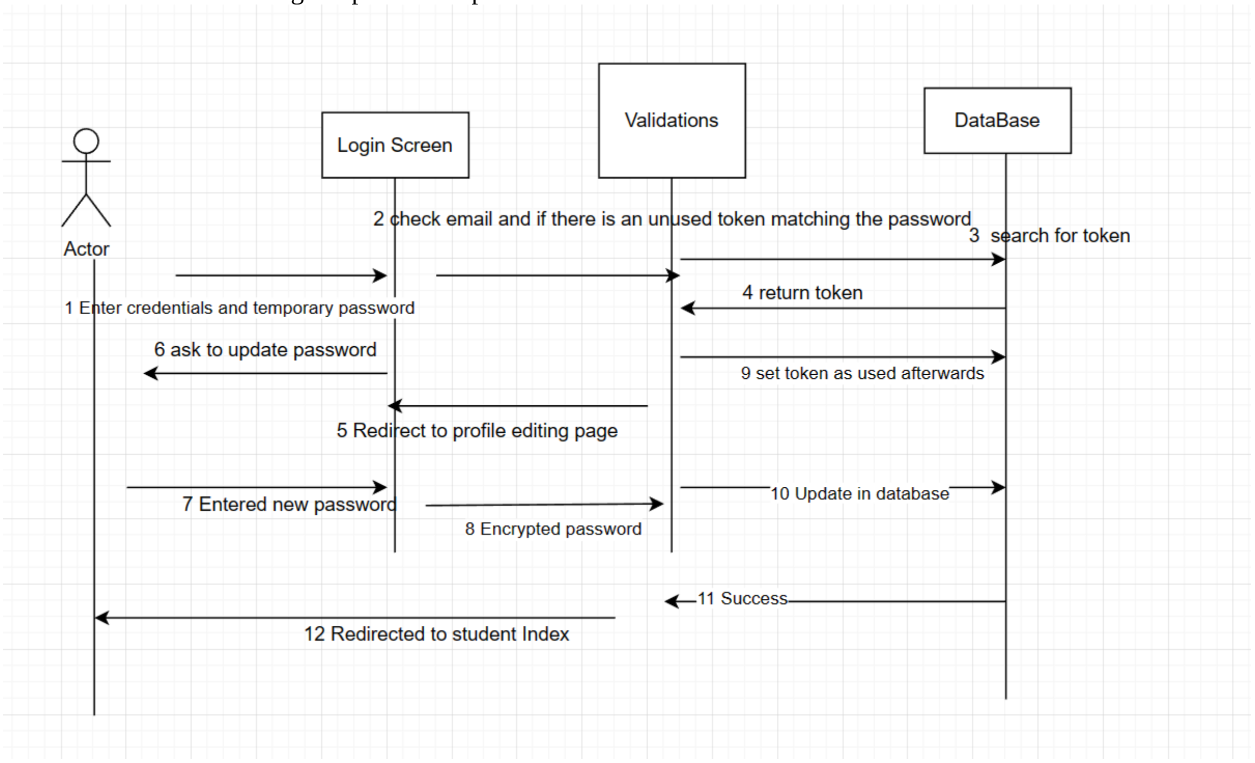


Deployment Diagram:


PostgreSQL Database
Back-End of the app, including the Business Logic and Data Access Layers, as well as the Controllers responsible with the UI's interactivity
Locally Hosted Web Interface

4. UML Sequence Diagrams

Sequence Diagram for a first-time login attempt from a student with the token given from the teacher, which would result in the student needing to update their password:


Actor
Login Screen
Validations
DataBase
1 Enter credentials and temporary password
2 check email and if there is an unused token matching the password
3 search for token
4 return token
5 Redirect to profile editing page
6 ask to update password
7 Entered new password
8 Encrypted password
9 set token as used afterwards
10 Update in database
11 Success
12 Redirected to student Index

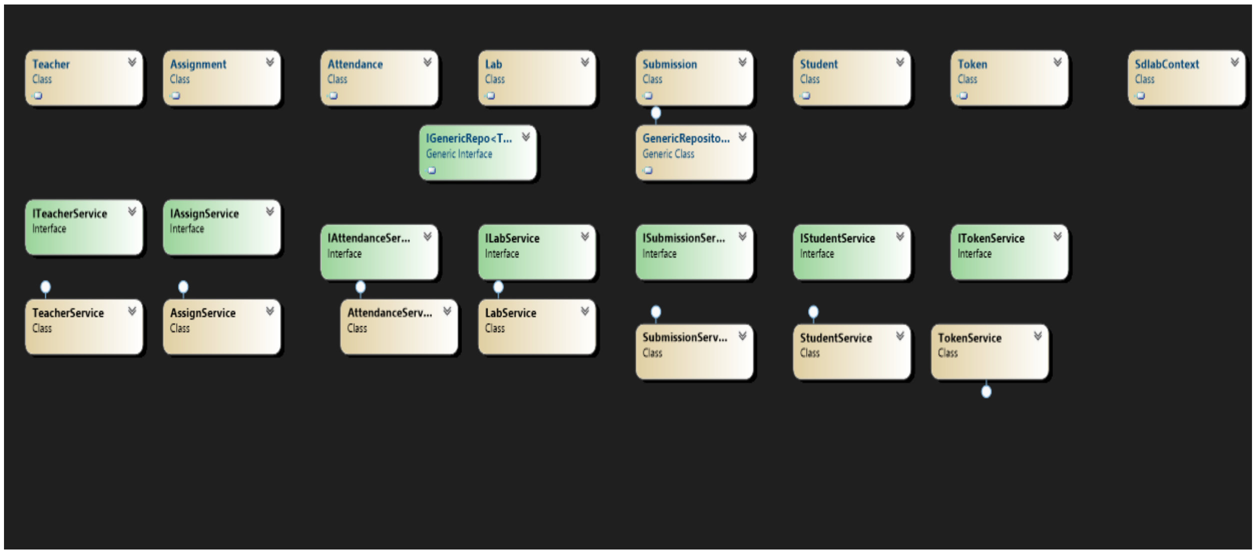
5. Class Design

5.1 Design Patterns Description

the Repository Design Pattern is a software architectural pattern that provides an abstraction layer between the data access logic and the rest of the application. It defines a common interface for performing common CRUD (Create, Read, Update, Delete) operations on data entities, and it is responsible for abstracting the details of data storage and retrieval. The Repository acts as a mediator between the business logic and the data access layer, promoting separation of concerns and maintainability in software development.

In a .NET Core app, the Repository Design Pattern involves using a Repository interface to define common data access operations (such as Add, Get, Update, and Delete) that are implemented by concrete classes in the Data Access Layer. The Repository acts as a mediator between the Business Logic Layer and the Data Access Layer, allowing for loose coupling and separation of concerns. The Business Logic Layer contains the application's business logic and interacts with the Repository through the interface, without being concerned about the details of data access implementation. Dependency Injection (DI) can be used to inject the Repository into the Business Logic Layer, promoting maintainability and testability.

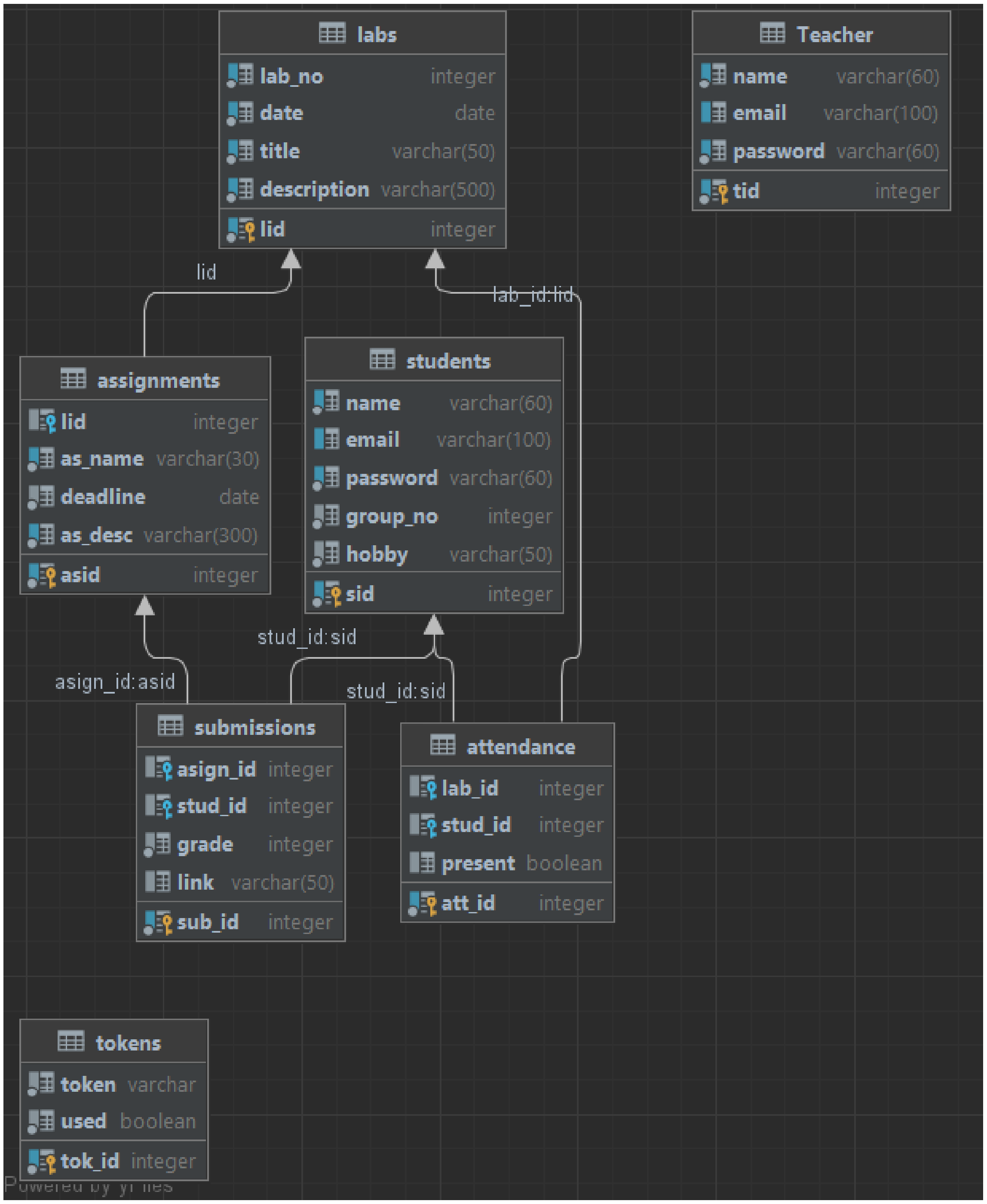
5.2 UML Class Diagram



In the above class diagram, the first 2 rows correspond to the Data access layer, containing the data models and the database context class and also the IGenericRepo interface which is implemented by the GenericRepository class , in which all database actions are performed.
The third and fourth row contain the service interfaces and classes which provide the business logic of the application and act as an intermediary between the Data Access Layer and the actual app. The app itself consists of the Controllers and their respective views, and deals with the user interface represented through web pages and how the different http requests are handled as per the REST Api guidelines.

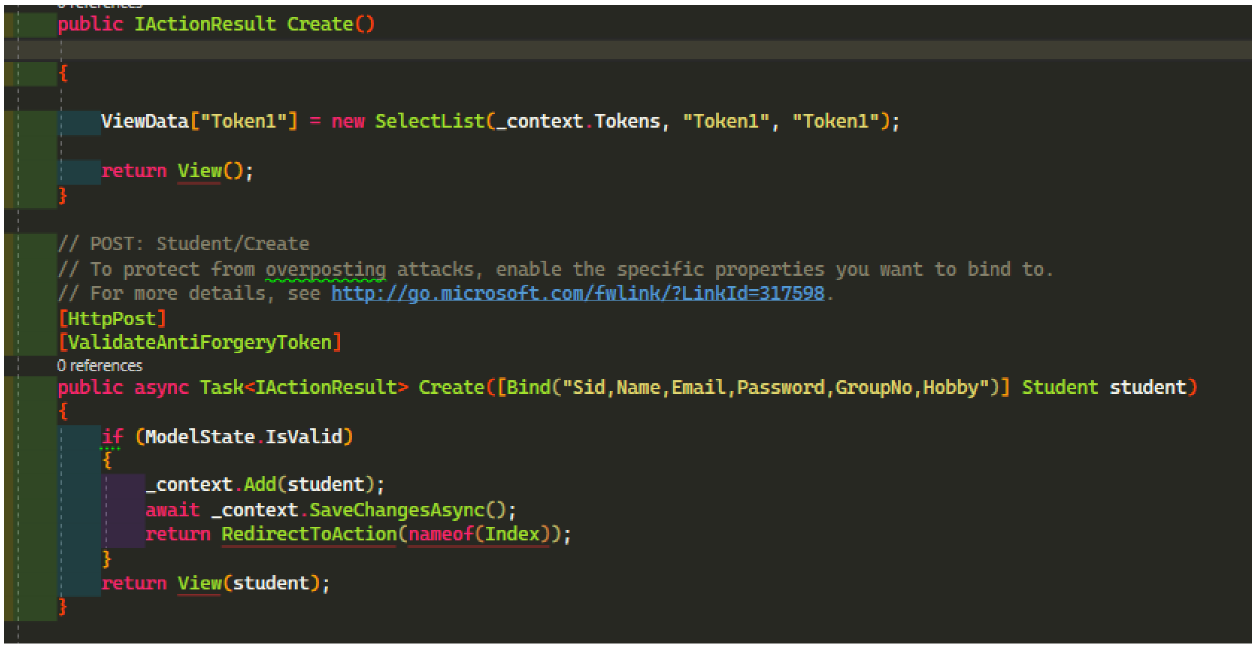
6. Data Model

For this second assignment, I've used PostgreSQL when it comes to the database provider, as it's one of the most stable and easy to integrate ones, being compatible with most programming languages and frameworks. I've used NPGSQL Entity Framework Package for .NET in order to integrate it into my application. The tables are : Teacher, labs, students, assignments, submissions, attendance and tokens with the relationships between them and their specific fields being easily visible in the following diagram:



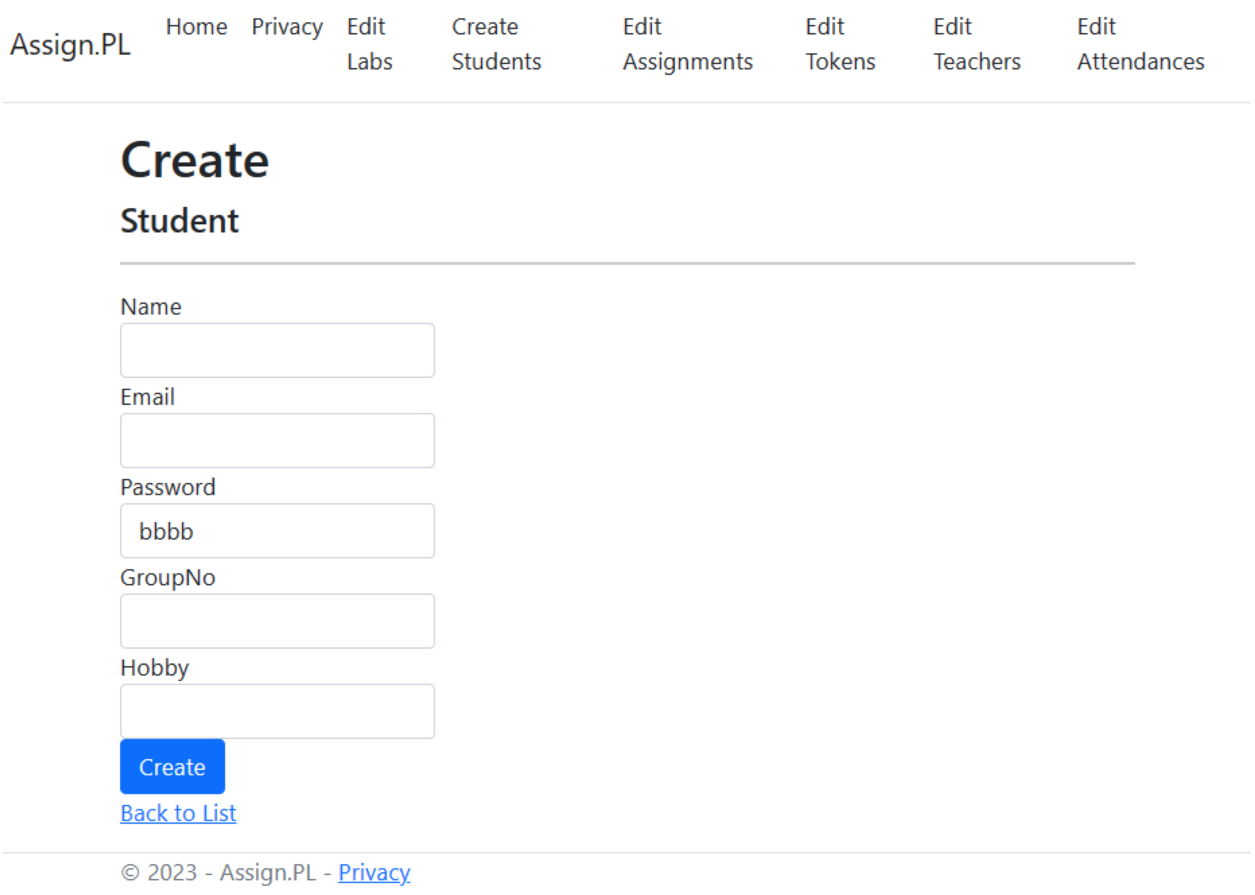
7. System Testing

When it comes to testing, everything was done using the Views from the application. The controllers would handle all the requests coming in from the Views and the functions would then perform the methods from the associated services.



```
public IActionResult Create()

{

    ViewData["Token1"] = new SelectList(_context.Tokens, "Token1", "Token1");

    return View();
}

// POST: Student/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Sid,Name,Email,Password,GroupNo,Hobby")] Student student)
{
    if (ModelState.IsValid)
    {
        _context.Add(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(student);
}
```

In this example the Create view is displayed, where we can enter the details that we want for our student:



Assign.PL Home Privacy Edit Labs Create Students Edit Assignments Edit Tokens Edit Teachers Edit Attendances

Create

Student

Name

Email

Password

bbbb

GroupNo

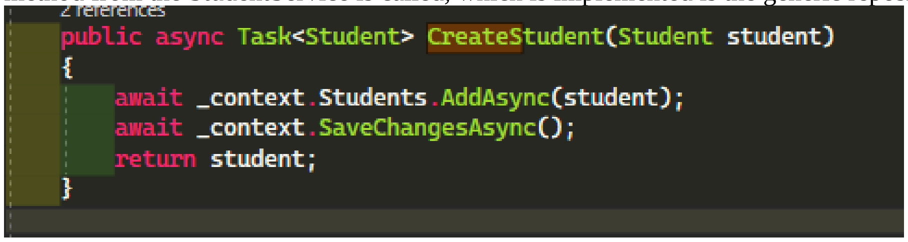
Hobby

Create

Back to List

© 2023 - Assign.PL - Privacy

Afterwards, the POST request is sent once we press the create button in the page and the method validates the data passed in the fields, making sure the data types correspond and the number of fields also correspond. Then the method from the StudentService is called, which is implemented is the generic repository as follows:



```
2 references
public async Task<Student> CreateStudent(Student student)
{
    await _context.Students.AddAsync(student);
    await _context.SaveChangesAsync();
    return student;
}
```

The Method uses a Student object generated using the fields from the page, and is then added to the database using the database context class. And all of the operations are performed in a similar manner.

8. Bibliography

CristinaMadalinaMihai/UTCNSoftwareDesignLaboratory: lab resources (github.com)

An awesome guide on how to build RESTful APIs with ASP.NET Core (freecodecamp.org)
Tutorial: Create a web API with ASP.NET Core | Microsoft Learn
Npgsql Entity Framework Core Provider | Npgsql Documentation