A picture containing text

Description automatically generated

|  |  |
| --- | --- |
| Module code and name: | Web Application Development |
| Assessment: | CW 1 |
| Student ID: | 00012938 |
| Submission deadline: | 20.03.2024 |
| Wordcount |  |
| Results date and type of feedback |  |
| Result (Pass or Fail) |  |
| **Feedback** | |
|  | |

**Table of Content**

[Introduction 2](#_Toc161868816)

[Link to the video regarding the project functionality 3](#_Toc161868817)

[Development Plan 3](#_Toc161868818)

[MoSCoW Task List: 3](#_Toc161868819)

[Gannt Chart 3](#_Toc161868820)

[Database Schema 5](#_Toc161868821)

[ERM Diagram: 5](#_Toc161868822)

[Relational Diagram: 5](#_Toc161868823)

[Github 7](#_Toc161868824)

[Link 7](#_Toc161868825)

[Repository Initialization 7](#_Toc161868826)

[Main commit details 7](#_Toc161868827)

[Last Commit 8](#_Toc161868828)

[System Architecture 8](#_Toc161868829)

[Repository Pattern 8](#_Toc161868830)

[Strategy Pattern 12](#_Toc161868831)

[API Explained 14](#_Toc161868832)

[Teacher Controller operations 14](#_Toc161868833)

[Student Controller 16](#_Toc161868834)

[Module 18](#_Toc161868835)

[Assignment 20](#_Toc161868836)

[ModuleStudent 21](#_Toc161868837)

[Grade 22](#_Toc161868838)

[SPA Explained 23](#_Toc161868839)

[Reference 26](#_Toc161868840)

# Introduction

This project is about student grading application. Front-end is done via Angular, Back-end is done via ASP.NET Core Web API.

## Link to the video regarding the project functionality

<https://youtu.be/ZEGBX_gdCUs>

# Development Plan

## MoSCoW Task List:

Must-Have:

1. User Authentication
2. User Registration
3. Minimal Profile Information Input (Name, Picture, Surname, etc.)
4. Roles restriction (Teacher/Student accounts with specific limitations on account type)
5. Module Creation
6. Enrollment of students for a module
7. Assignment Creation
8. Grading Specific Students for specific assignments
9. Grades View Functionality

Should-Have:

1. Dashboard for Students
2. Average Grade Calculator
3. Number of Enrolled Students View Functionality

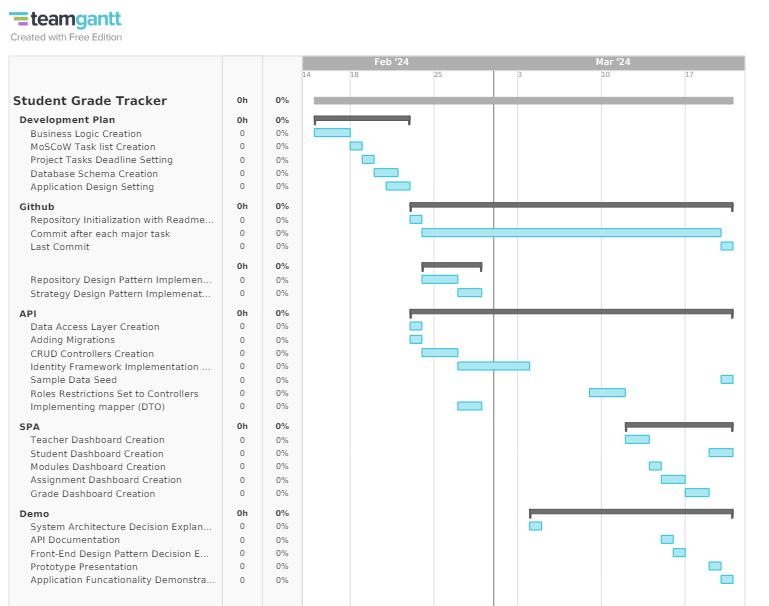
Could-Have:

1. Module Ownership by multiple Teachers
2. File Upload Functionality

Won’t-Have:

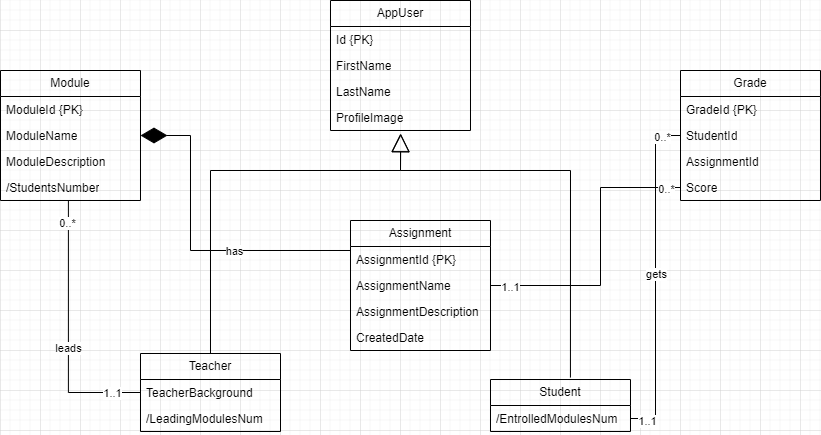
1. Report Generation
2. Average module grade calculator
3. Visual charts for grades
4. Image upload (only link insertion)

## Gannt Chart



## Database Schema

### ERM Diagram:



### Relational Diagram:

Before Applying IdentityFramework for Roles Creation:



ModuleStudent Table was added by Framework a we have many-to-many relation.

After implementing IdentityFramework:

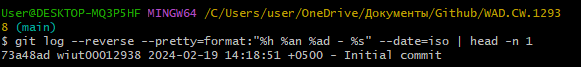


# Github

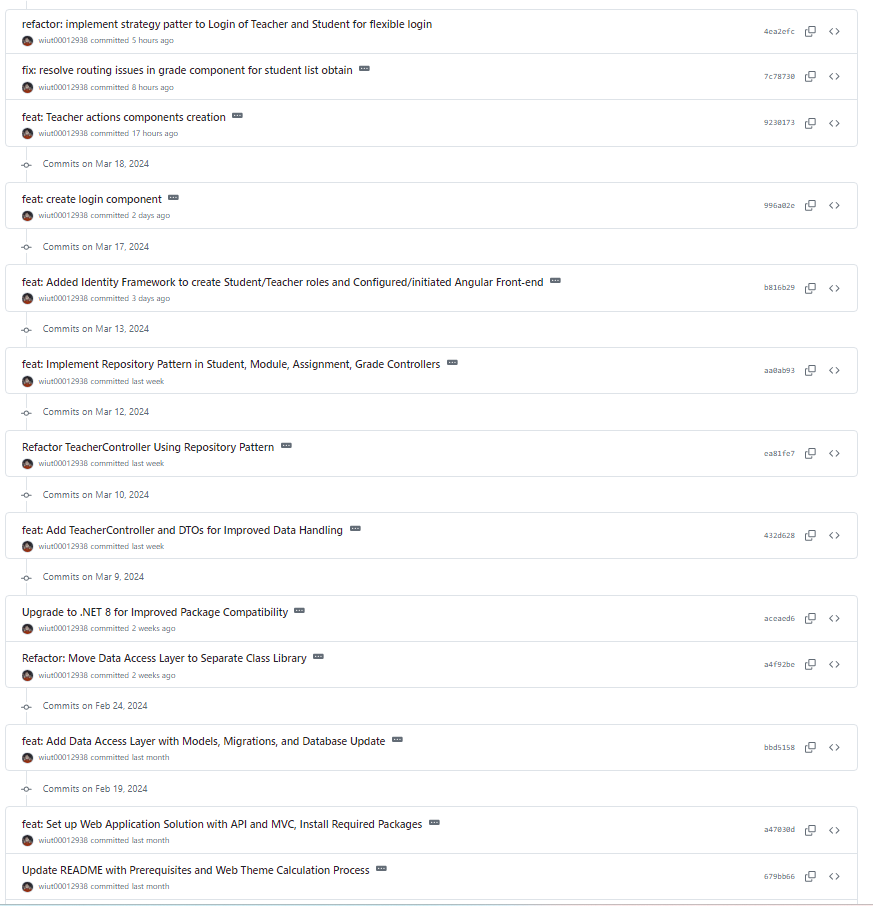
## Link

<https://github.com/wiut00012938/WAD.CW.12938>

## Repository Initialization



## Main commit details

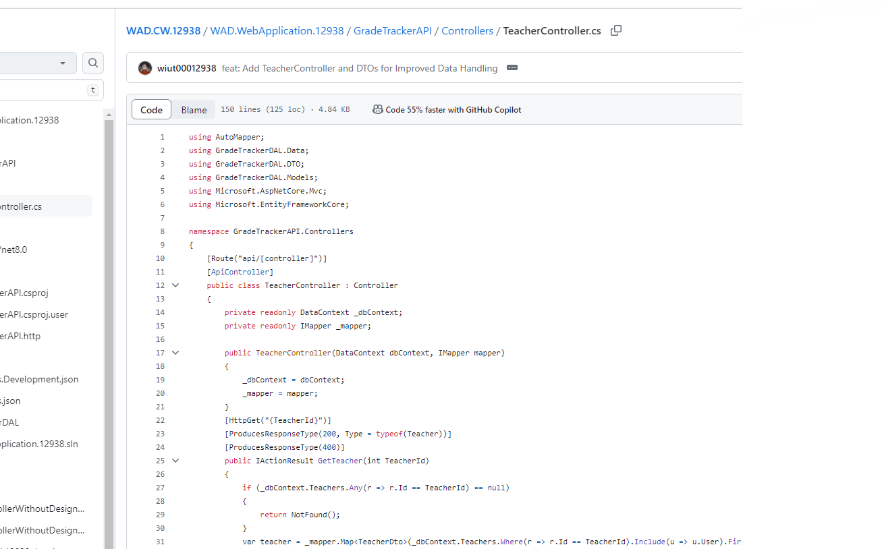


## Last Commit

# System Architecture

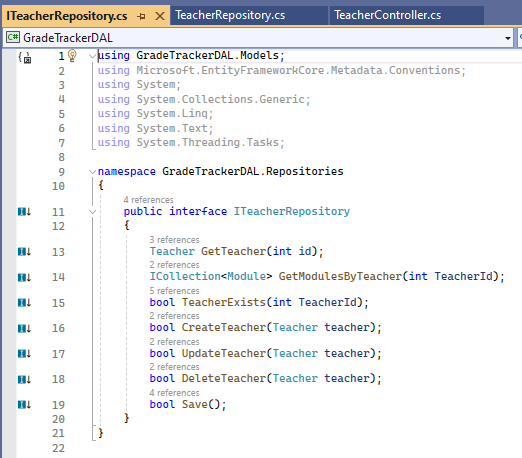
For the back-end part of the application, two different design patterns where used in order to enhance the clearness of the code and follow SOLID principles: the Repository patter and the Strategy pattern. Let’s clarify:

## Repository Pattern[[1]](#footnote-1)





This is an example of Teacher Controller that does basic CRUD operations. But it is not safe to store all the logic(responsibilities) in one file as it doesn’t follow SOLID O and SOLID S principles. The repository pattern can encapsulate a data access logic that will give us an abstraction. Also, by this movement concern can be separated.

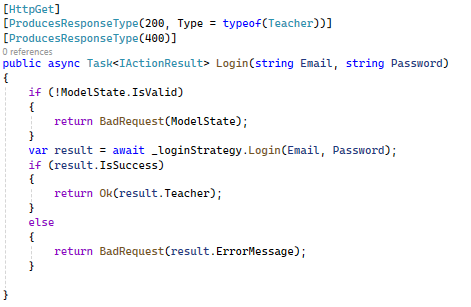
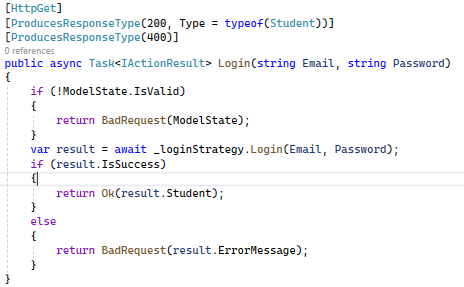


This is how the code looks like after implementing the Repository pattern. Beside the clean architecture and maintainability, integrity was reached. How it works: firstly, the interface that defines a “contract” (must implement things) is created and then an actual repository that will handle our CRUD operations. In controller only a reference will be used.

## Strategy Pattern



These are Login operations in Teacher and Student controllers. Basically, it works fine in this way. When connecting with front-end we can create two buttons and two forms for Teacher/Student Login. However, there is a way to handle this logic by implementing a Strategy Pattern that dynamically handles different logic operations. This approach is good for the case of login/register and if in the future we want to add a login for a new role there is no need to modify the existing code. That means SOLID O principle is followed.

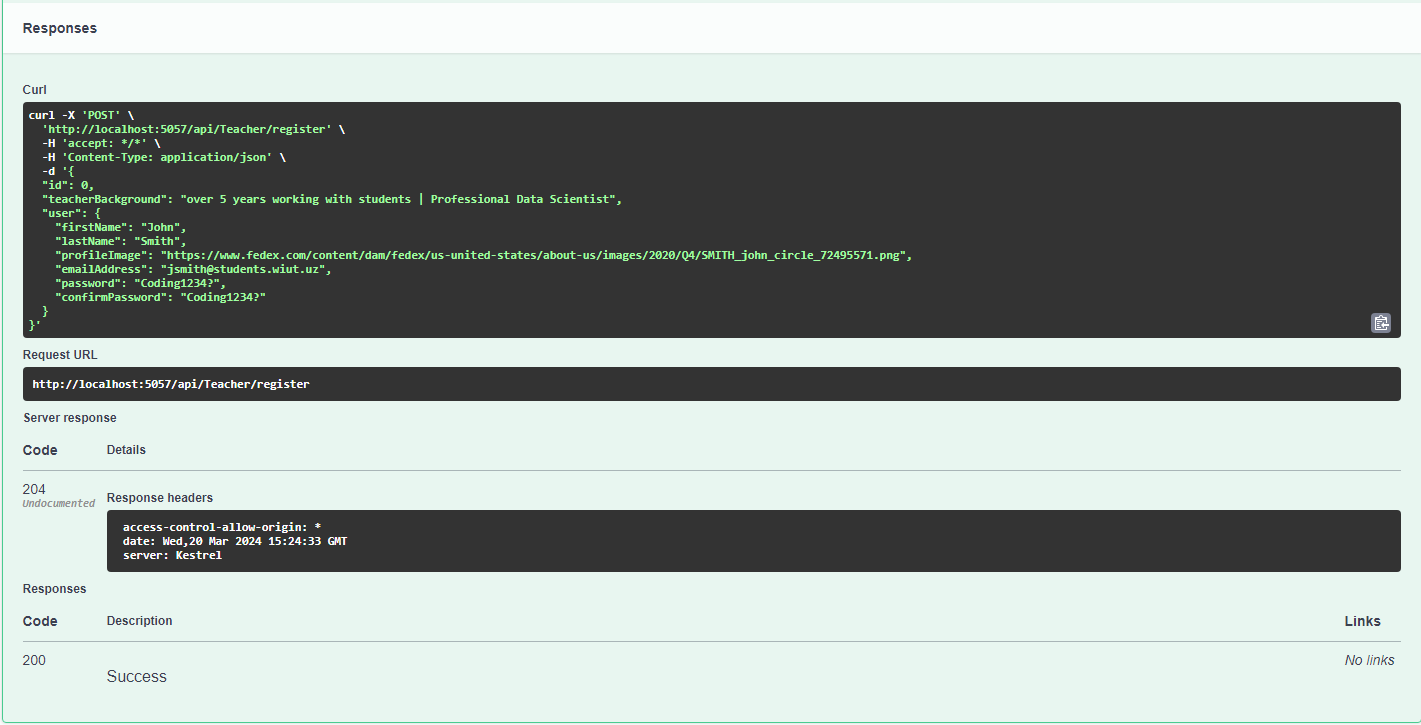


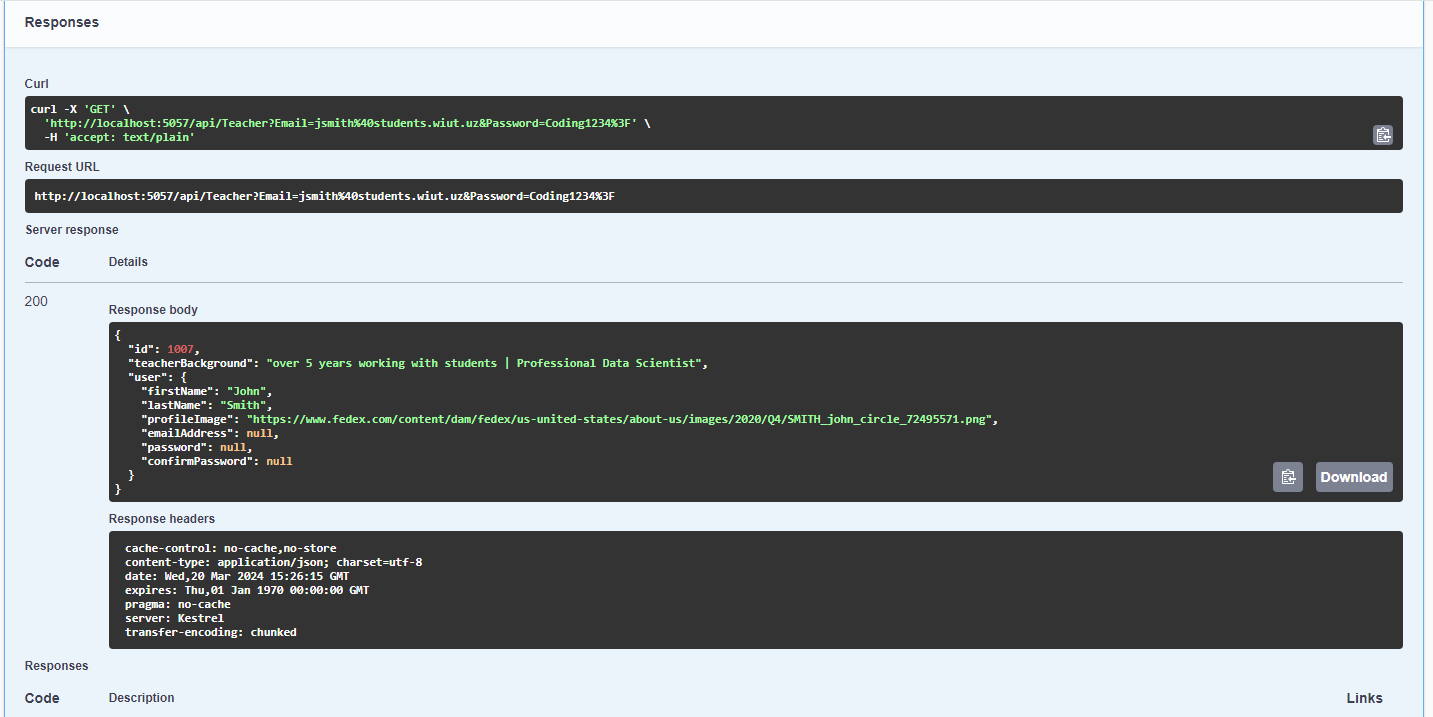


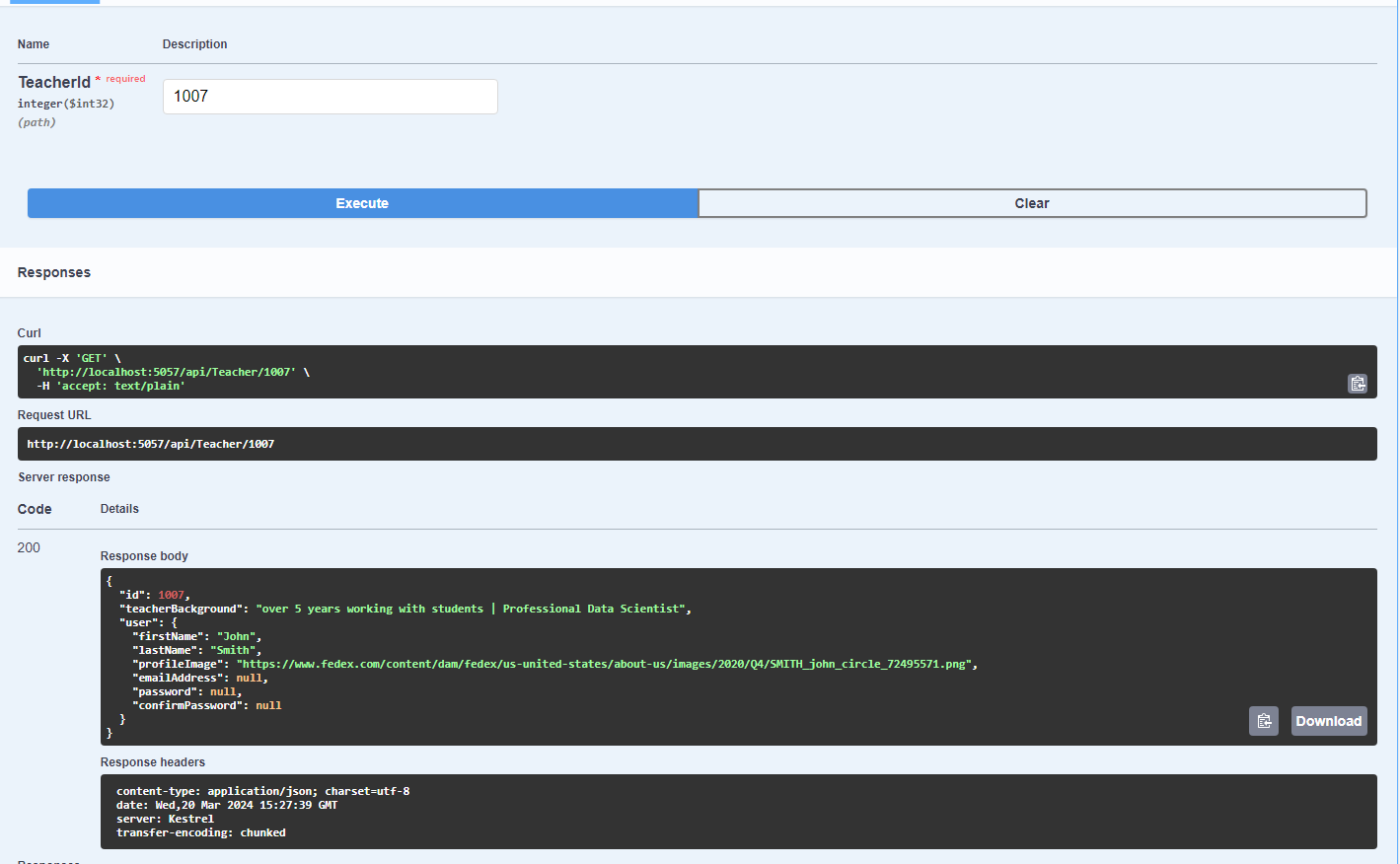
As seen here we are calling ILogicStrategy and we will know what exact strategy will be applied only after execution. With that we are also following Single Responsibility Principle

# API Explained

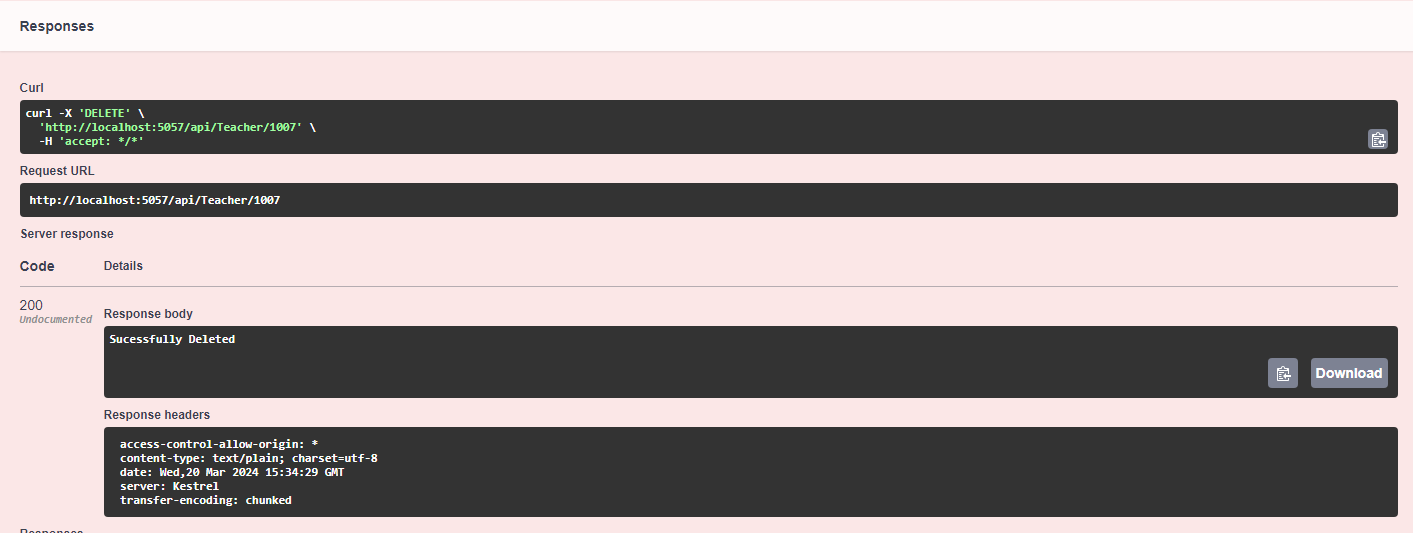
## Teacher Controller operations





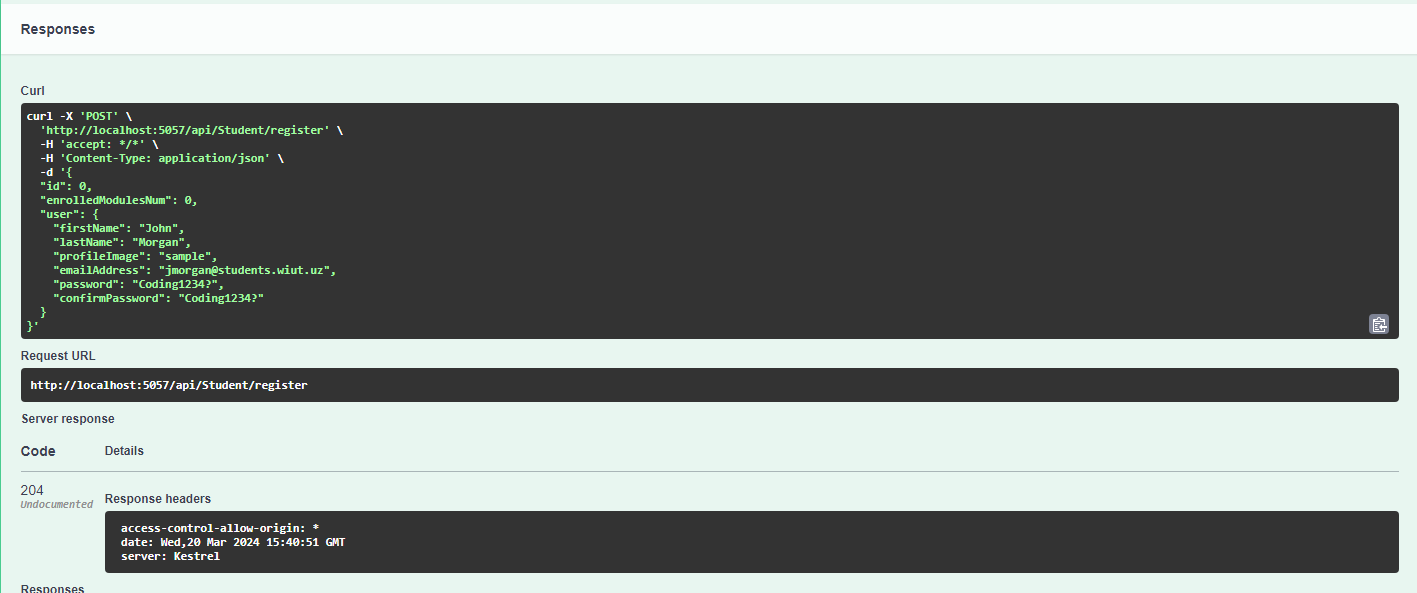


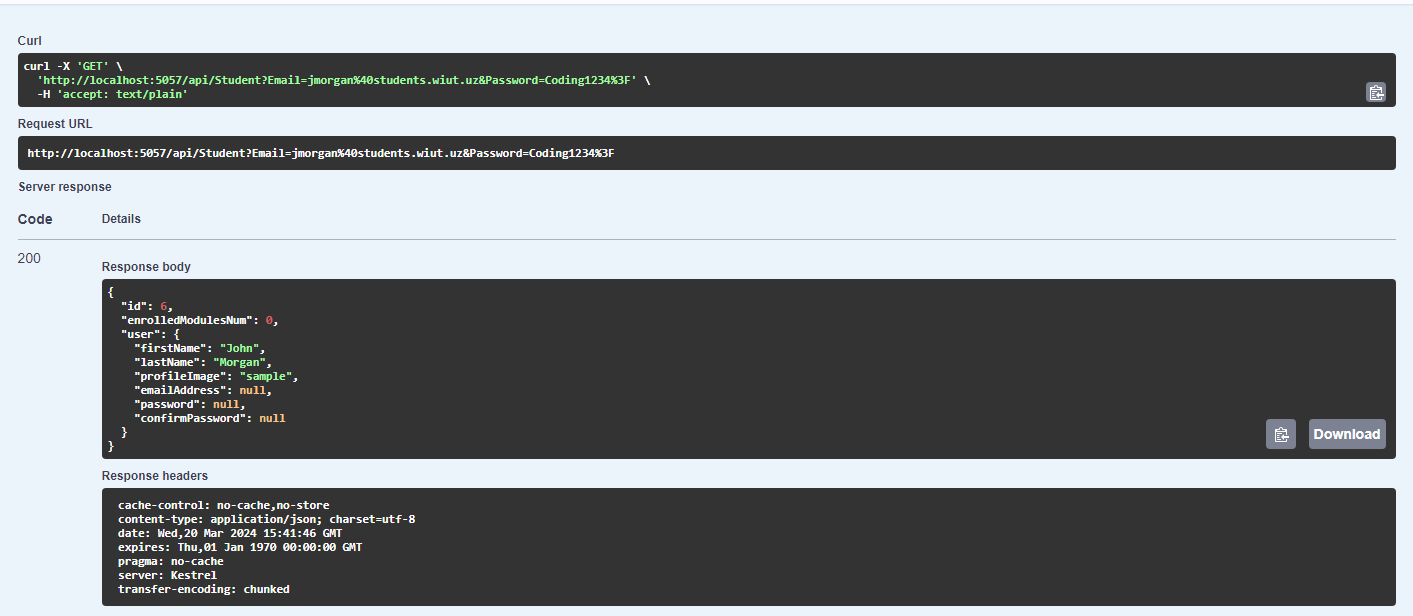


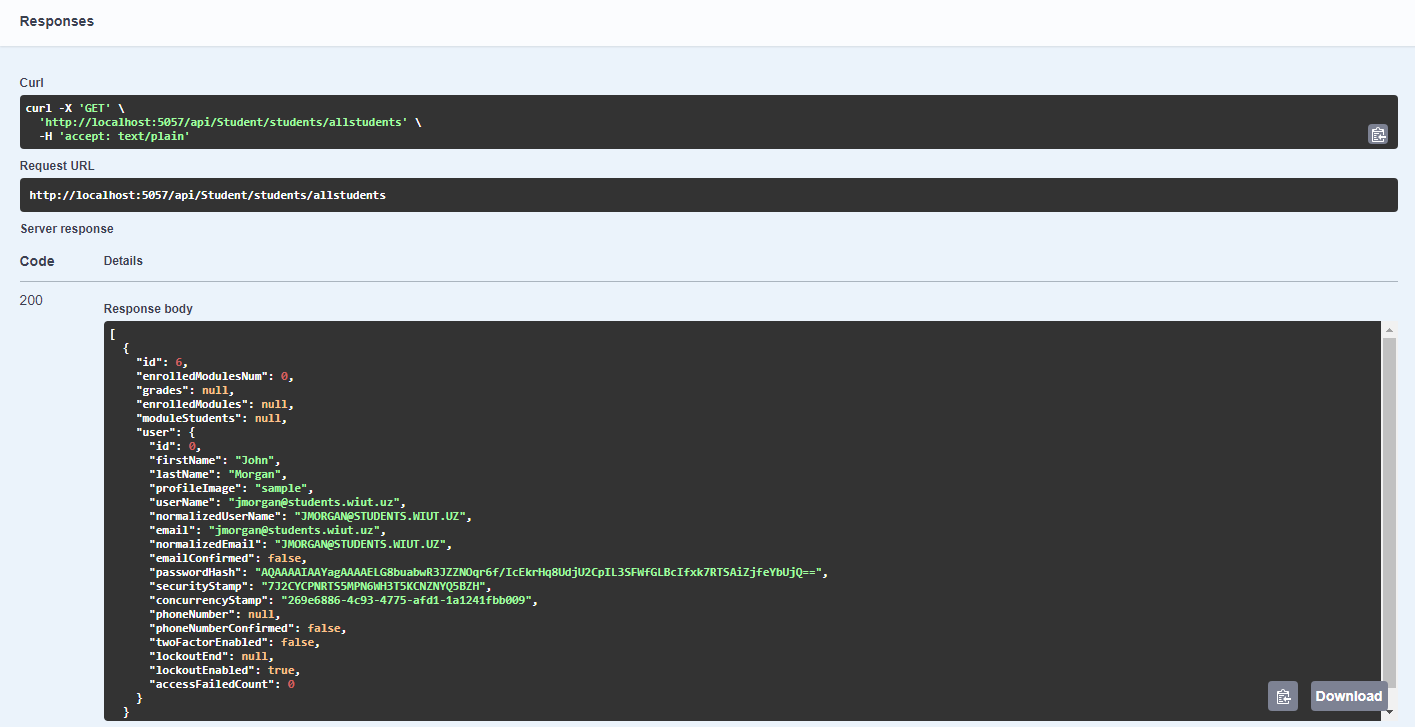


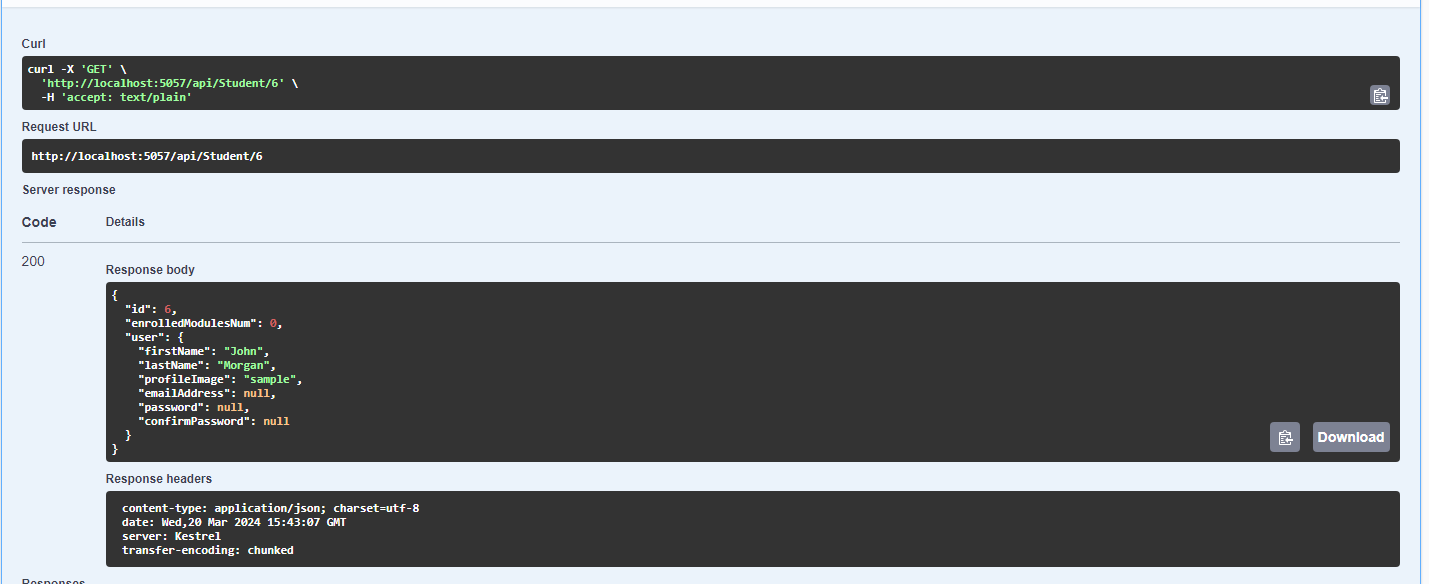


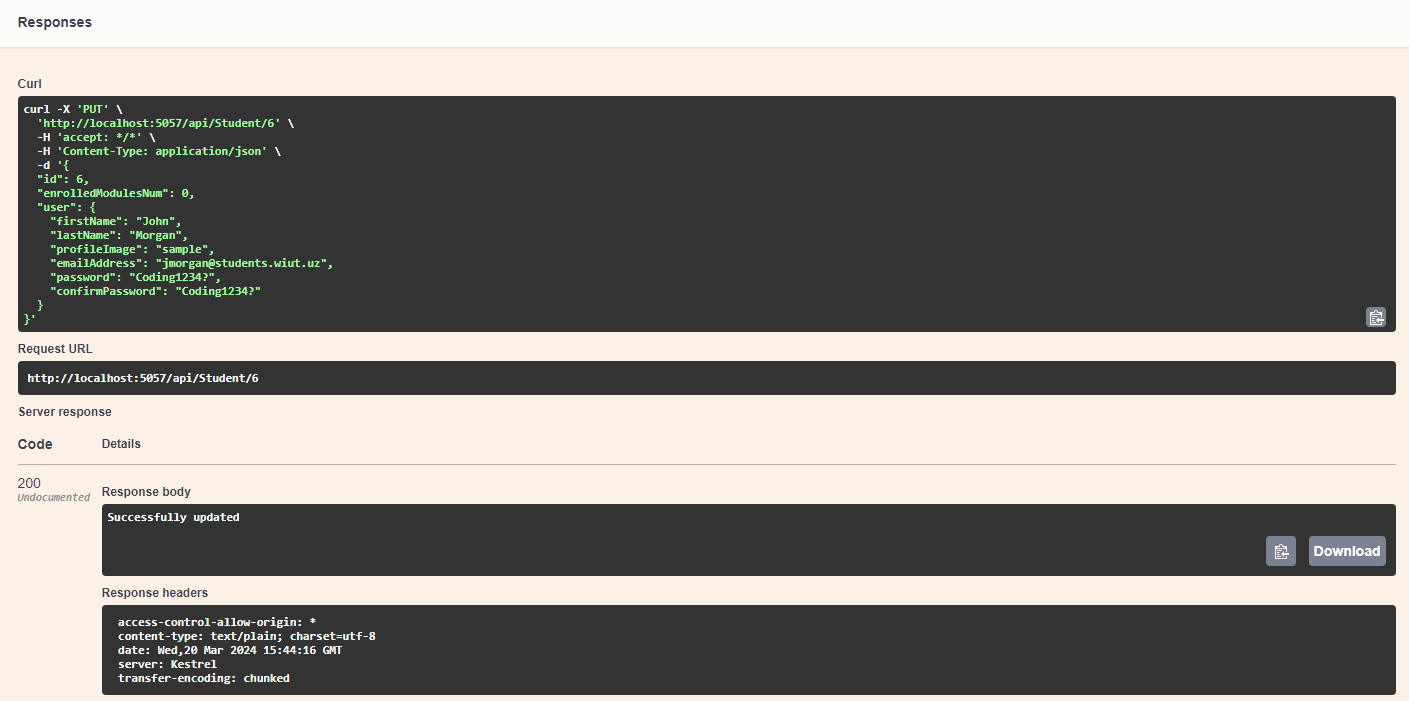
## Student Controller

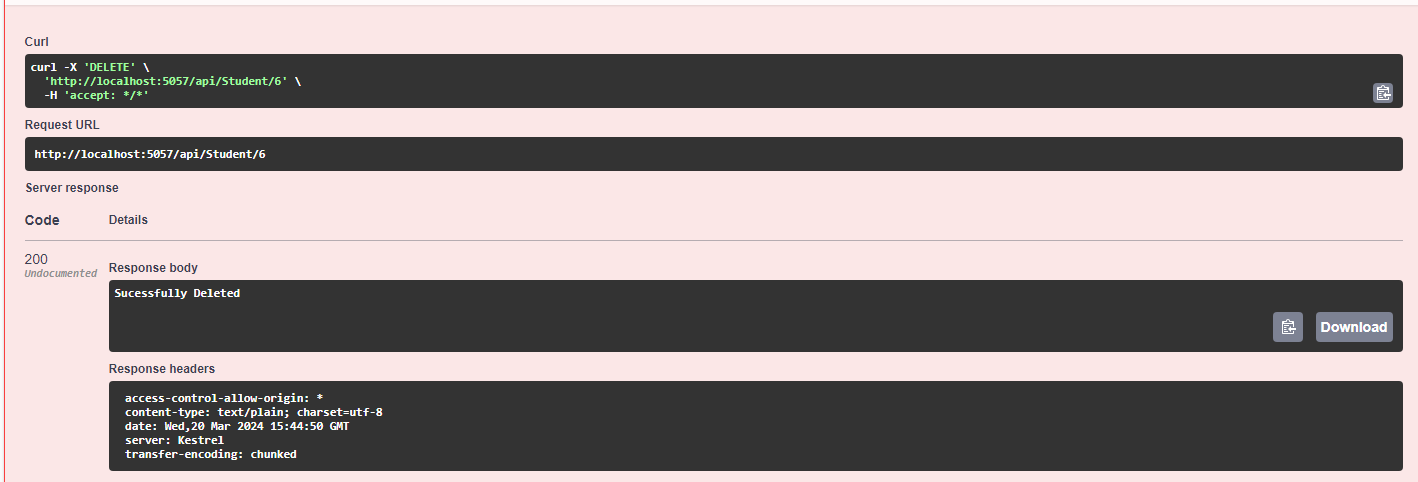




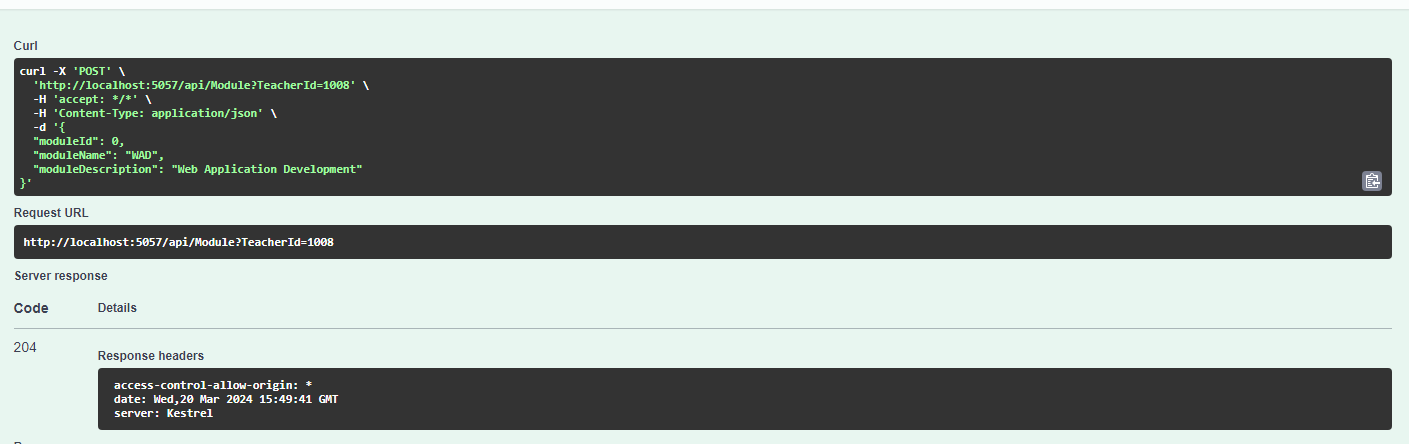


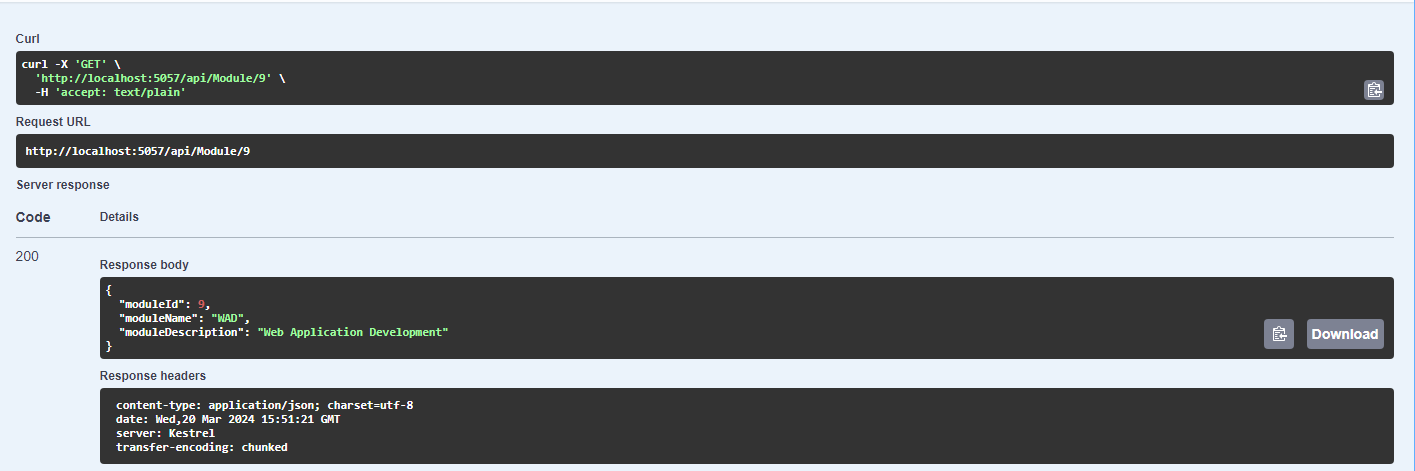


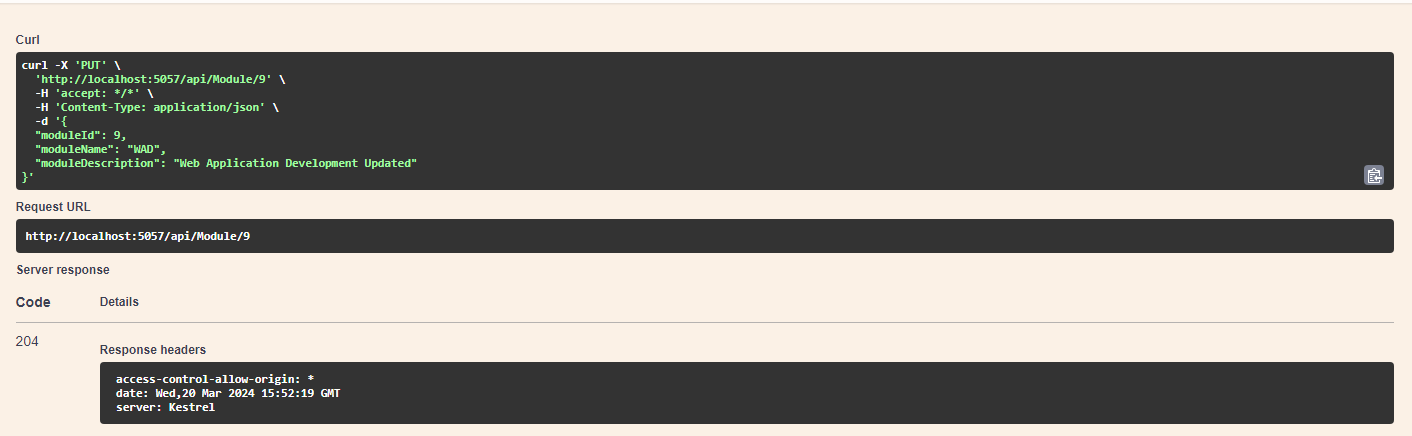


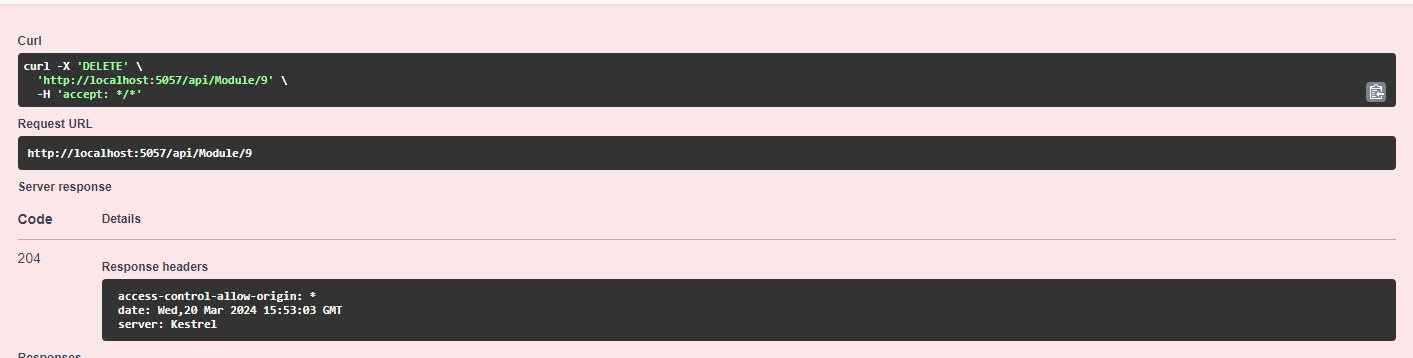


## Module

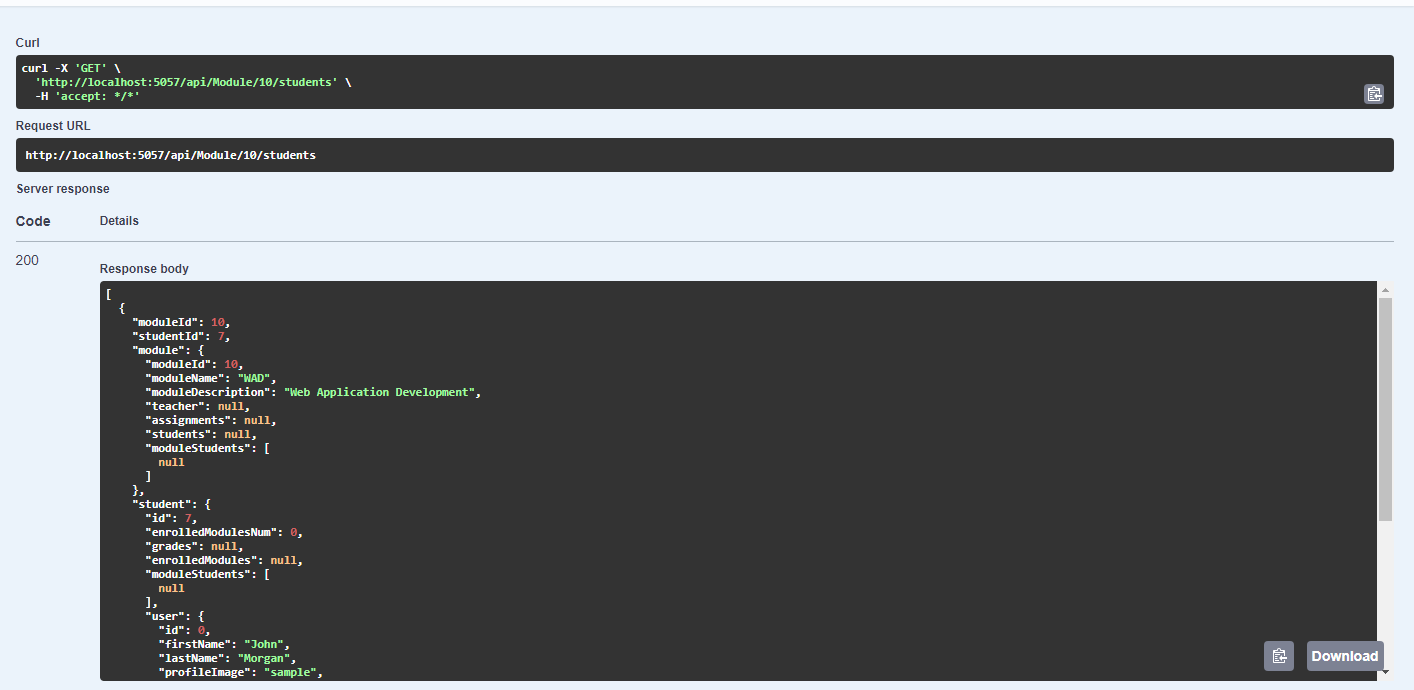




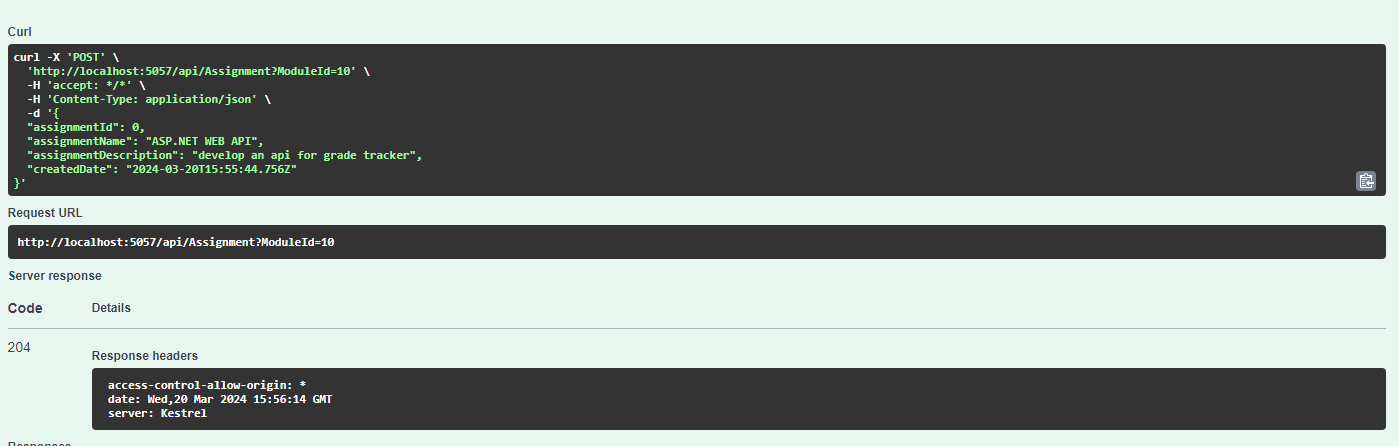




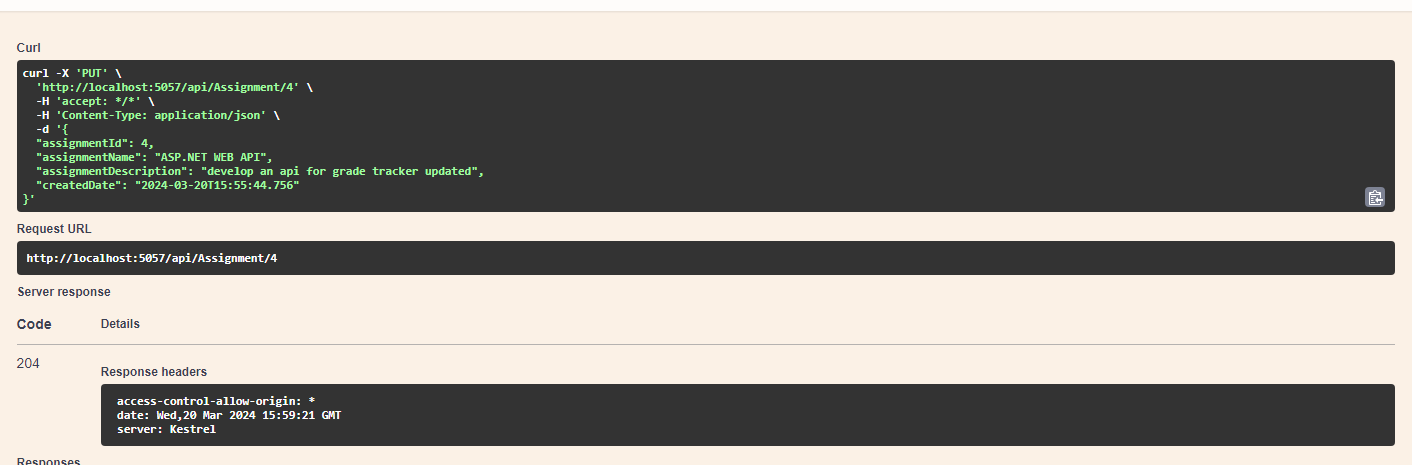




## Assignment







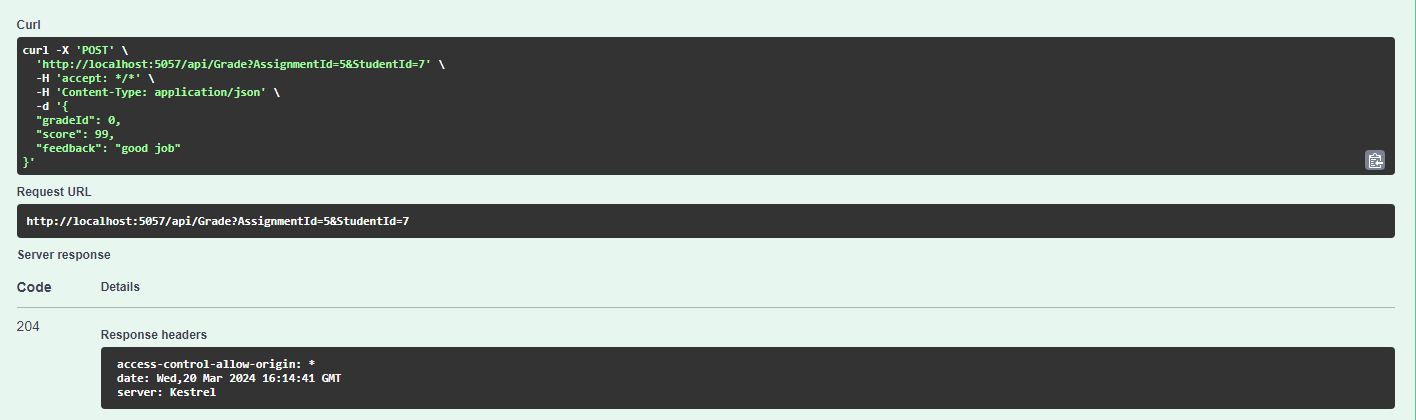


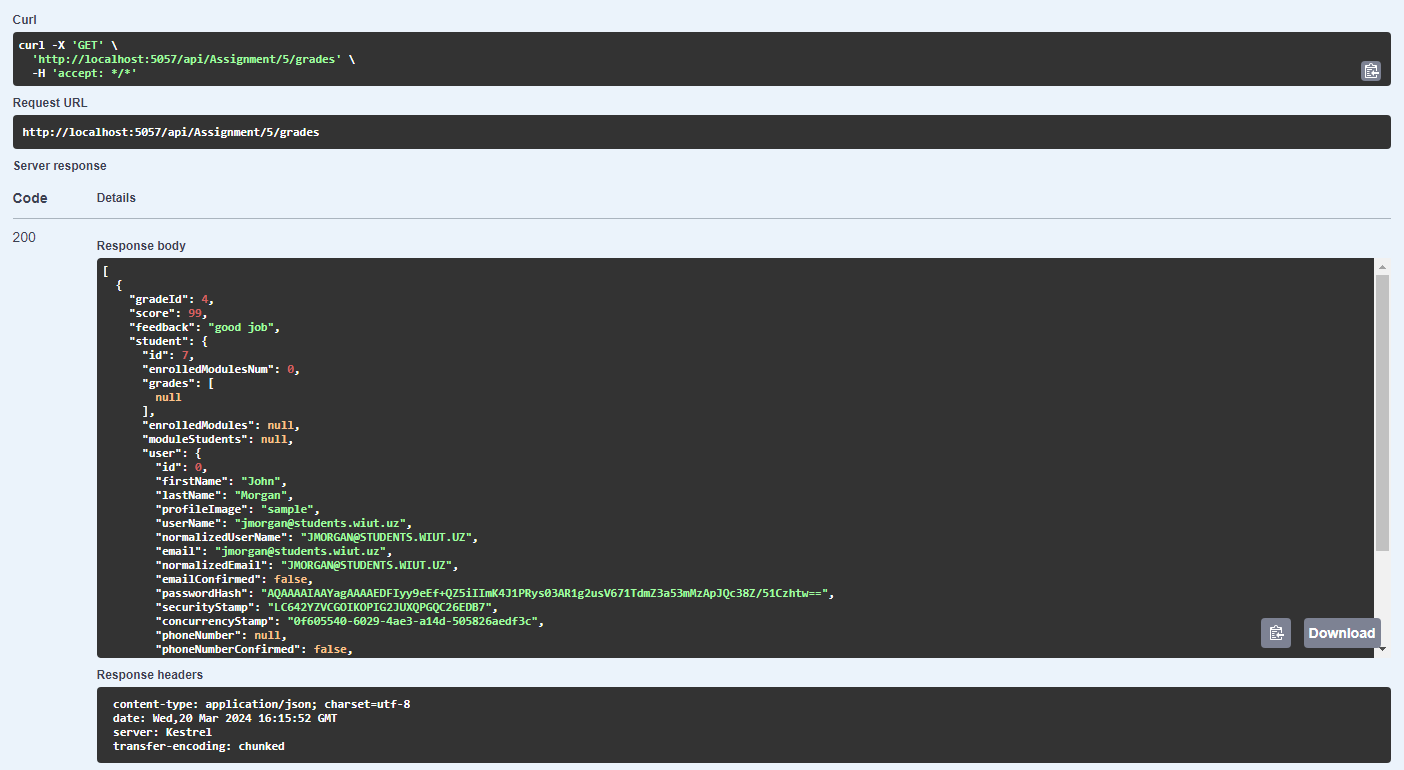
## ModuleStudent

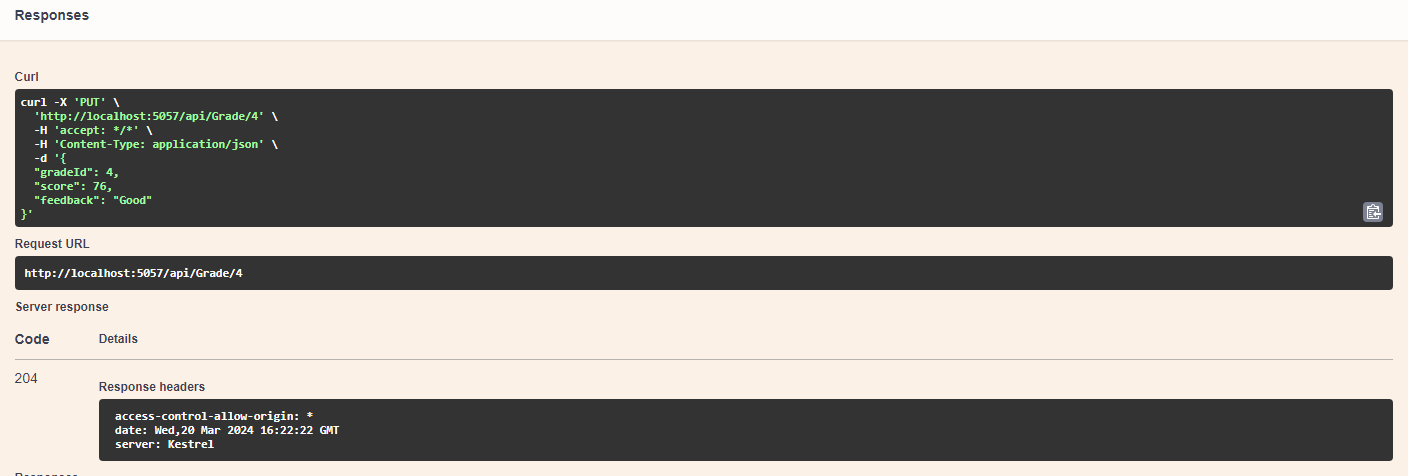




## Grade

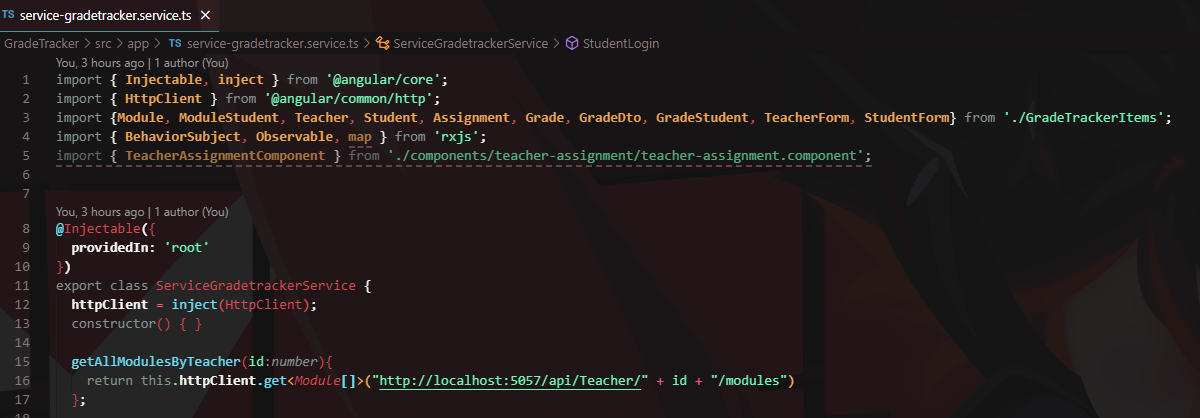








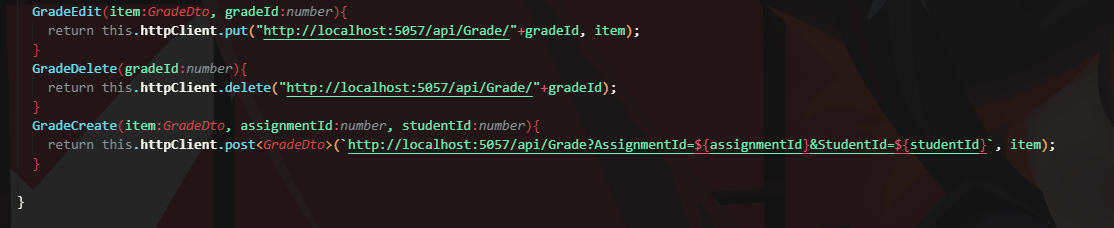
# SPA Explained



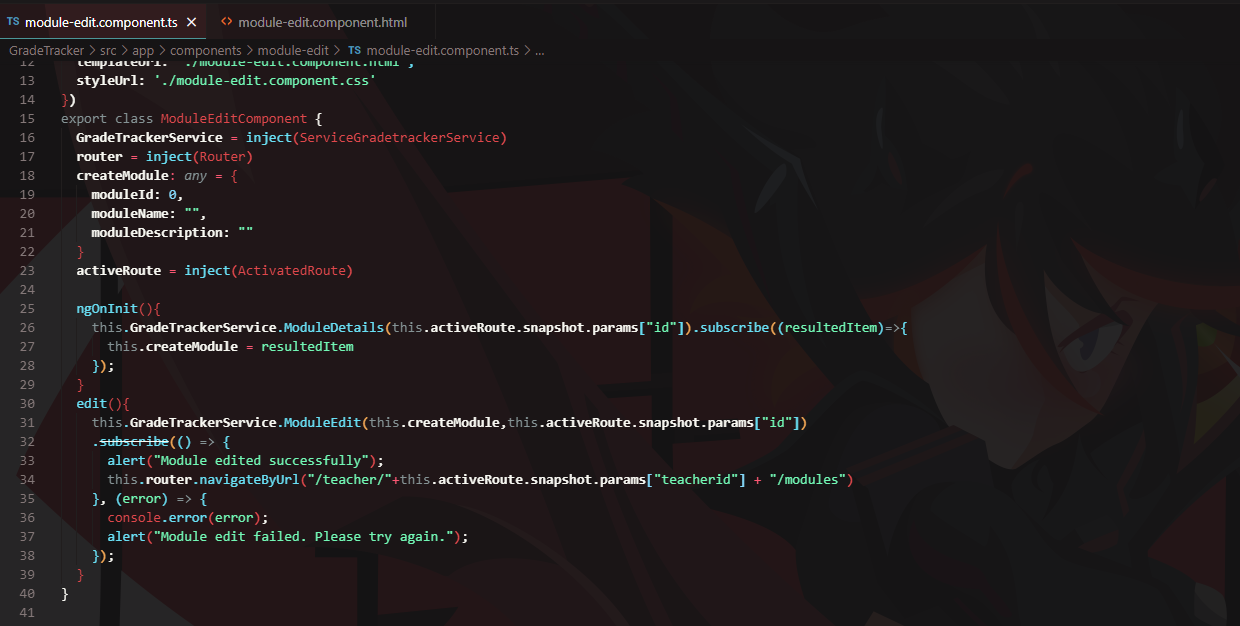


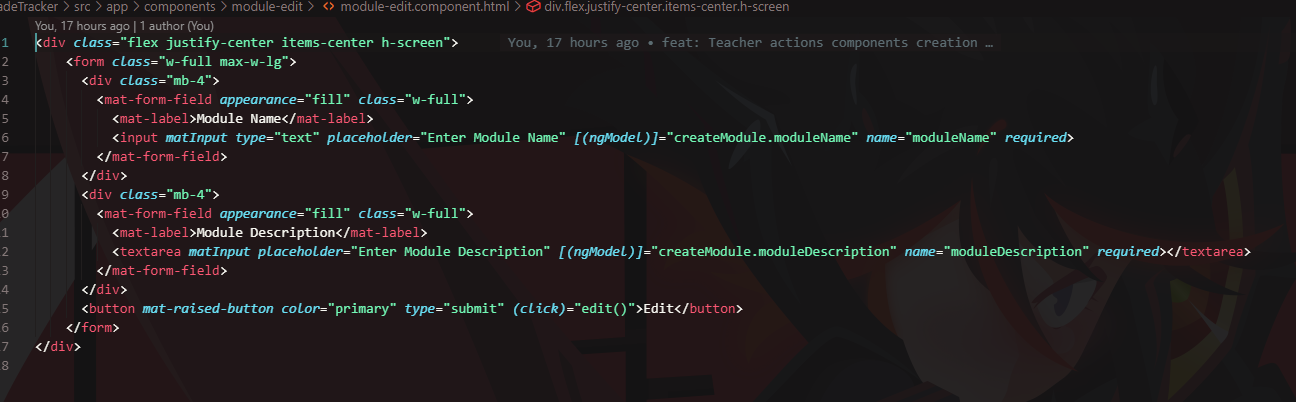






These are all the operations used for fulfilling front-end. ASP.NET WEB API project was configured in order to enable fetch() and connection with external localhost(from 5057 to 4200). How the data is retrieved and exchanged between backend and front-end? By routing. From the screenshots of swagger it is seen the urls used. They enable us to manipulated and access the data.

One example of how from edit module:  




When we get the path "module/:teacherid/edit/:id", we start filling the form with the acutal original data, so the user could see what he/she has now. [(ngModel)] and other staff that is related to angular material helps easialy do it. When edit() button is pressed we pass the new data to a parameter and make a .ModuleEdit().

Regarding the design architecture, most of the things are good and maintained enough because app is based on independent components. However as in the case of back-end, login/register are better to unify by applying know to use Strategy pattern. Currently, teacher/student login and register are separate and from the responsiveness point of view it is not good. Having one login and one register where you can dynamically select a student or teacher.

# Reference[[2]](#footnote-2)

Reference listSmith, T. (2022). ASP.NET Core MVC 2022 .NET 6 - MVC Explained. 20-video course. www.youtube.com. Available from <https://www.youtube.com/watch?v=q2AcJmB03Io&list=PL82C6-O4XrHde_urqhKJHH-HTUfTK6siO>.

Smith, T. (2022). ASP.NET Core Web API .NET 6 2022 - 1. Create Project & Quick Tips. www.youtube.com. Available from <https://www.youtube.com/watch?v=_8nLSsK5NDo&list=PL82C6-O4XrHdiS10BLh23x71ve9mQCln0>

1. All screenshots shown in this report is written and belongs to student the student (the first screenshot for instance shows that the code shown later was written and committed to GitHub [↑](#footnote-ref-1)
2. Some code snippets especially for html table part was take from WAD seminar 9, seminar 10 materials [↑](#footnote-ref-2)