

EUROPEAN UNIVERSITY OF LEFKE

Faculty of Engineering
Department of Computer
Engineering



COMP 452 GRADUATION PROJECT II **Moodle for Faculty of Engineering**

Developed by: SHAKHRIYOR TOYLOKOV

Student Number: 184119

Overview:

Introduction:

Working of the application:

Front End Side Implementation(Angular 10):

Back End Side Implementation(C# .NET 5):

Database Side (SQL):

Security (Authentication and Authorization):

Conclusion:

Introduction:

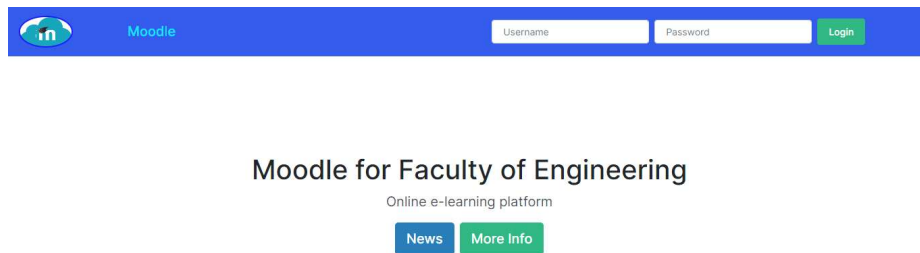
For my graduation project I have developed Moodle e-learning web application for faculty of Engineering. For developing the application I have used C# .NET 5 for Backend and Angular 10 for Front end side, as well as that, SQL was used for database side. According to my implementation, the application has three user types as student, instructor (teacher) and administrator. For each of them the different user roles have been given so that the one is not able to do the role or job of the other one since it has been authorized and implemented carefully of both API and Client sides code implementations. The usernames and passwords are provided by admin or saved to database by admin by the students' std number or instructors' instructor number and for passwords, the users can enter their passport or id number. Also, the security side of the application such as running it in HTTPS protocol or using different algorithms to encode and decode passwords and JWT authentication and authorization tokens and some other features has also been developed to provide secure website for the users' data. Moreover, Error handling methods have been implemented to handle different kinds of errors separately if they occur which will be discussed with details in the Error handling section of the document.

As for the application, the main features of e-learning platform such as Announcement, Lecture Notes, Lecture Videos, Outlines, Quizzes, Homeworks and etcetera have been developed for different kind of users. As well as that, there is registration feature for admins as registering new students or teachers by uploading a file and getting the users registered according to the information in the file or registering manually by filling the forms individually. Also, the admin can register new courses and assign courses to students also teachers. The web application has very user-friendly interface, easy to use with the essential features for different user types, the ability for updating the user data according to different user roles and interactivity between users. The more detailed information about all the features of the application will be discussed in the upcoming sections of the document.

Working of Application:

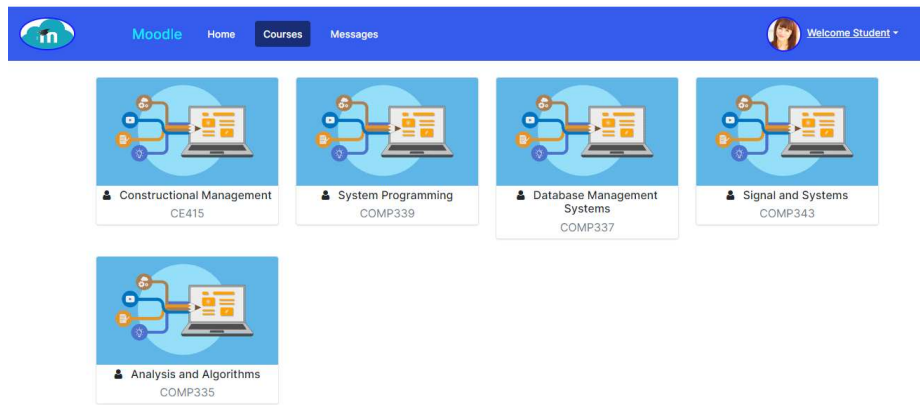
In this section, I will be explaining the working of the application by the help of some screenshots and comments.

Main Page:



In the main page there is a common nav bar with common Log In form which can identify the user type whether he is student, teacher or instructor according to their username and navigates to their pages by providing different user roles. The users are not able to navigate another pages if they are not logged in, since it has been secured for both by API and Client side. As well as that, There are common News button which navigates the user to news page about the faculty and More Info button to navigate the user to introduce with how to use and detailed information about the application. To log in the if the user is a student he needs to enter his student number starting with std and if the teacher is the user he needs to enter his number starting with ins and for admin starting with @admin and number and provided passwords (such as passport numbers) by the admin.

Users' Main Page:



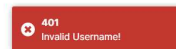
In the main page of users' either if the user is student or teacher, he will be redirected to his courses page if the entered username and password are correct. Otherwise, there will be a toastr prompt show up telling the user that the entered username or password is wrong.



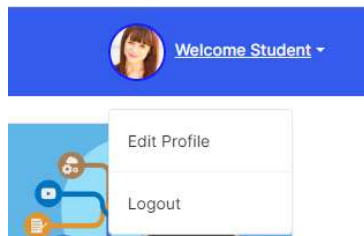
Moodle for Faculty of Engineering

Online e-learning platform

[News](#) [More Info](#)



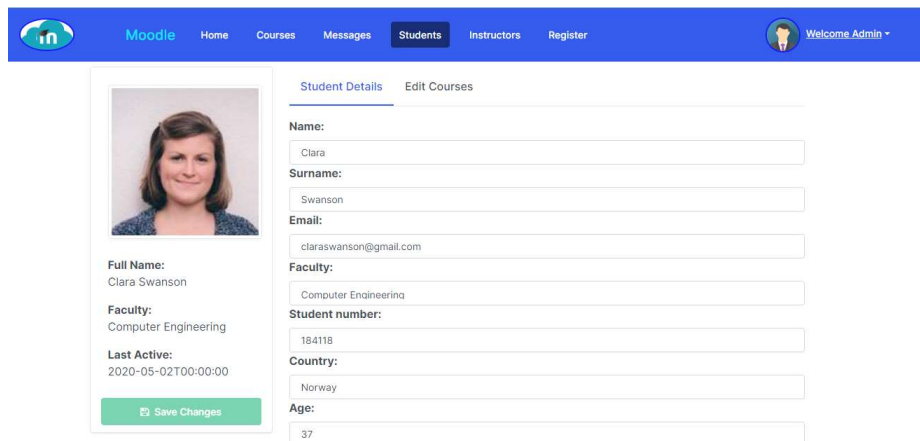
User Dropdown:



On the corner of the navbar there is a dropdown list to give the user ability to edit his profile such as editing his name and surname (not username or password) and email, also adding a profile photo, deleting photos. Also there is a link to log out from the profile.

Editing user information :

For updating the user data, different roles are given to different users. For example, as a student or teacher, they can only update their name, surname, email and photos of their profiles. Whereas, for admins, the main role is given to update the students and teachers all information such as username, password, date of birth, country, email and other personal information and also courses that they are taking or giving as a teacher.



The screenshot shows the Moodle user profile page for Clara Swanson. The page has a blue header with navigation links: Moodle, Home, Courses, Messages, Students (active), Instructors, and Register. A user profile picture is shown on the left. The main content area has two tabs: 'Student Details' (active) and 'Edit Courses'. The 'Student Details' tab contains a form with the following fields:

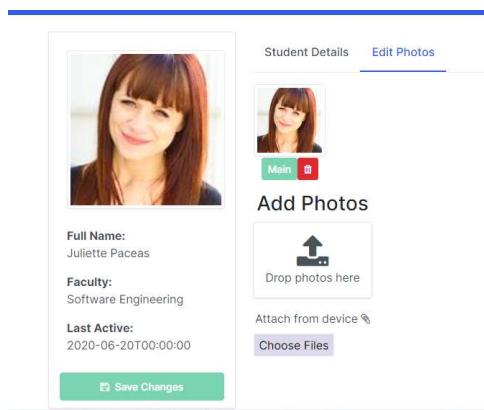
Name:	Clara
Surname:	Swanson
Email:	claraswanson@gmail.com
Faculty:	Computer Engineering
Student number:	184118
Country:	Norway
Age:	37

On the left side of the profile, the following information is displayed:

- Full Name:** Clara Swanson
- Faculty:** Computer Engineering
- Last Active:** 2020-05-02T00:00:00

A green 'Save Changes' button is located at the bottom left of the profile section.

For editing photo, there is a photo uploader and profile main photo setting ability of the users



The screenshot shows the Moodle user profile page for Juliette Paceas. The page has a blue header with navigation links: Moodle, Home, Courses, Messages, Students (active), Instructors, and Register. A user profile picture is shown on the left. The main content area has two tabs: 'Student Details' and 'Edit Photos' (active). The 'Edit Photos' tab contains a form with the following fields:

Main	<input type="checkbox"/>
-------------	--------------------------

Below the 'Main' field, there is a section titled 'Add Photos' with a large upload icon and the text 'Drop photos here'. Below this, there is a section titled 'Attach from device' with a 'Choose Files' button.

On the left side of the profile, the following information is displayed:

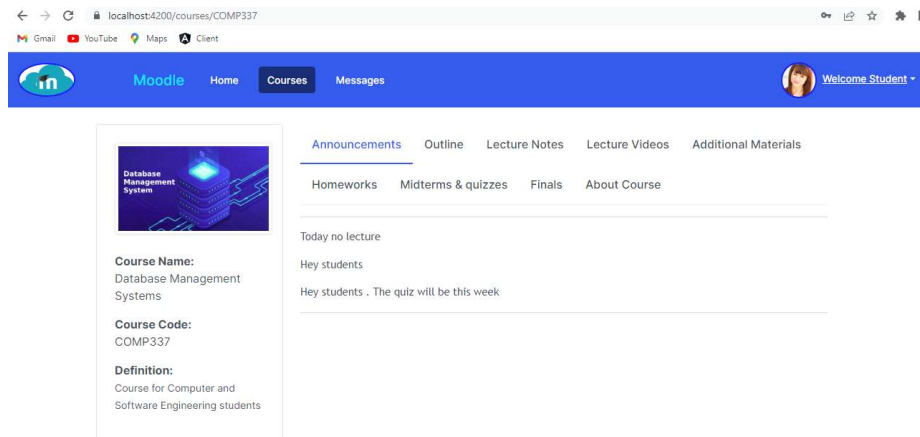
- Full Name:** Juliette Paceas
- Faculty:** Software Engineering
- Last Active:** 2020-06-20T00:00:00

A green 'Save Changes' button is located at the bottom left of the profile section.

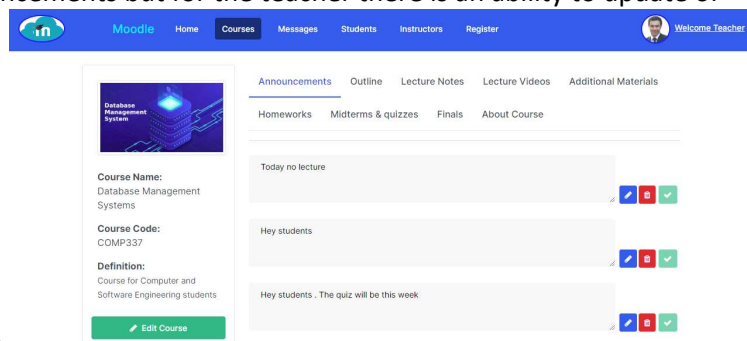
In the course lists, the course cards has icon buttons to navigate the user to different parts of the course like dashboard or news and if the user is teacher or admin there is a button to navigate to course-edit page



Dashboard of the course

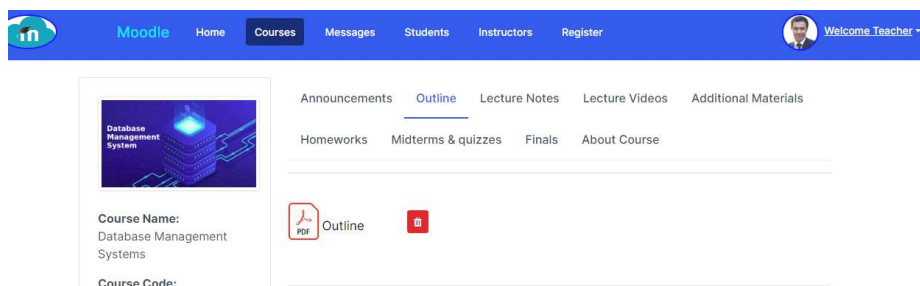


So in the dashboard of the courses, there are many tabs that show up the different sections and information on that section to users. For example in the announcement section the student can see the announcements but for the teacher there is an ability to update or



delete announcement also.

Outline tab--> So for outline tab , the student can see the outline document which the teacher has uploaded and the teacher is able to change or delete the outline as well.



Lecture Notes Tab--> For lecture notes tab , the student can see and download the lecture notes uploaded by teacher and the teacher has ability to delete them if they want in this tab.

The screenshot shows the Moodle interface for a course titled "Database Management Systems" (Course Code: COMP337). The "Lecture Notes" tab is selected, displaying a list of files: "Presentation 3 (7).pdf", "Presentation 3 (7).pdf", "Presentation 3 (7).pdf", "templatenev.pdf", "Presentation 3 (3).pdf", and "Presentation 3 (3).pdf". The left sidebar contains course details: Course Name, Course Code, and Definition.

Lecture Videos Tab--> For lecture videos tab, the student can see the video links with the name of videos uploaded by teachers and the teachers can delete those links as well.

The screenshot shows the Moodle interface for the same course, but with the "Lecture Videos" tab selected. It displays a list of video uploads: "Lecture 5 video", "Lecture 5 video", "New Lecture 9", and "New Lecture 2". Each video entry has a red delete icon. The left sidebar contains course details and an "Edit Course" button.

Homeworks Tab--> For homework section there is a homework upload page for the students and checks if the student has uploaded homework, he is able to edit it and upload new homework answer as well.

The screenshot shows the Moodle interface for the same course, with the "Homeworks" tab selected. It displays a table with submission status, grading status, and time remaining. The "Add Submission" button is visible at the bottom.

Submission Status	Grading Status	Time remaining
No attempt	Not graded	6 days 8 hours

In add submission page there is an uploading session for student to upload their homework's

The screenshot shows the Moodle 'Add Files' interface. On the left, there's a 'Drop photos here' area with an upload icon, and a 'Choose Files' button below it. On the right, the 'Upload queue' section shows a table with columns for NAME, SIZE, and PROGRESS. A single file, 'project-individual.pdf', is listed with a size of 0.269 MB. Below the table, a 'Queue progress' bar is shown, followed by buttons for 'Upload all', 'Cancel all', and 'Remove all'. At the bottom left, there is a 'Back to Course' button.

And if it is successfully uploaded the success toaster will show up and redirect the user to the main page of the course. Also, the status will change to submitted and add submission button will change to edit submission. So in the edit submission page the student can delete his previous homework and upload the new one.

This screenshot displays the Moodle course page for 'Database Management Systems'. The left sidebar contains course details: Course Name, Course Code (COMP337), and Definition. The main content area has a top navigation bar with links like Announcements, Outline, Lecture Notes, etc. Below this, a 'Homeworks' tab is selected, showing a table with submission status. The status is 'Submitted', and the grading status is 'Not graded'. A 'Time remaining' of 6 days 8 hours is shown. An 'Edit Submission' button is located at the bottom of the table.

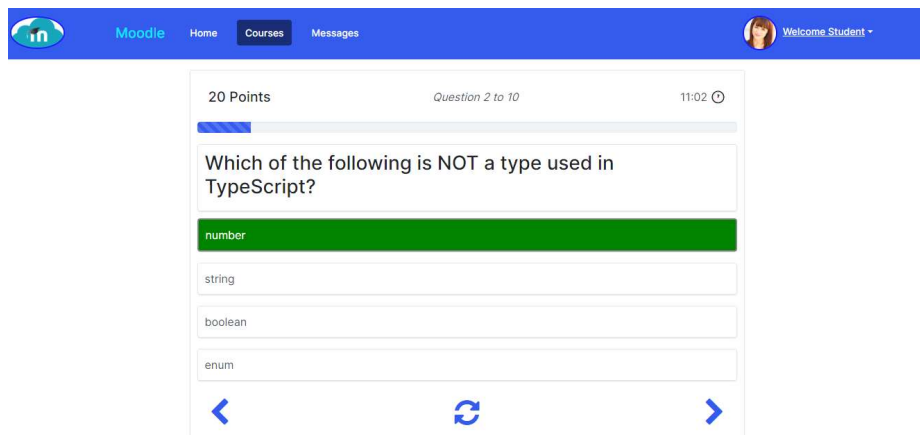
Midterms & Quizzes Tab--> In the midterms and quizzes tab, the student is able to answer the test or quiz formatted questions prepared and uploaded by a teacher. As well as that, after finishing the quiz the student is able to see his results and review each questions with their correct answers and explanations

Quiz main page:

The screenshot shows the 'Welcome to Quiz' page in Moodle. The left sidebar is identical to the previous screenshot. The main content area has a navigation bar with 'Midterms & quizzes' selected. Below the navigation bar, the page title is 'Welcome to Quiz'. It states 'The quiz contains total 10 questions. Each question holds 10 Points' and 'Quiz for Software Engineering Students'. Under 'Rules:', there are three points: 1. Correct questions give you 10 points, 2. You will have 12 minutes to answer all the question, 3. Refreshing the page will reset the quiz. A 'Start the Quiz!!' button is at the bottom.

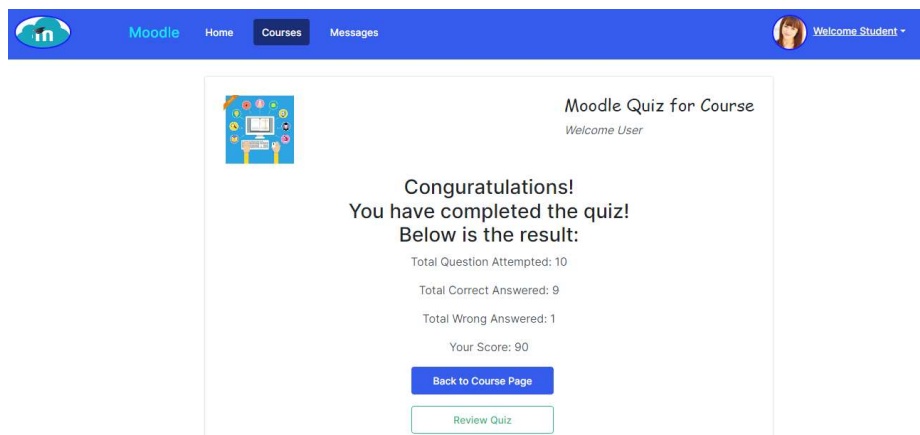
Questions page: The student can see the process bar and time time that started counting.

Also there are buttons to go to next question previous question and refreshing the quiz.



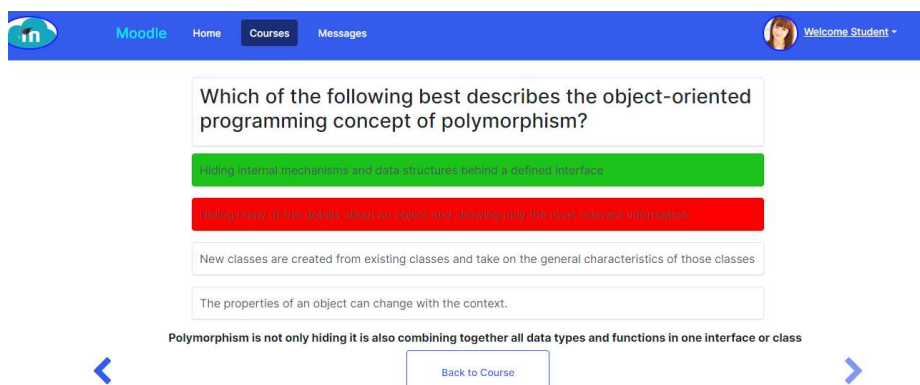
The screenshot shows a Moodle quiz interface. At the top, there's a blue navigation bar with 'Moodle', 'Home', 'Courses', and 'Messages' links, and a 'Welcome Student' message. Below the bar, the quiz details are shown: '20 Points', 'Question 2 to 10', and a timer '11:02'. The question is 'Which of the following is NOT a type used in TypeScript?'. There are four radio button options: 'number' (selected and highlighted in green), 'string', 'boolean', and 'enum'. At the bottom of the question box, there are three navigation buttons: a left arrow, a refresh icon, and a right arrow.

Quiz Result page: In the quiz result page, the student can see his result (points, correct answer, wrong answer) and can go to review quiz page to review the quiz result one by one.



The screenshot shows a Moodle quiz result page. At the top, there's a blue navigation bar with 'Moodle', 'Home', 'Courses', and 'Messages' links, and a 'Welcome Student' message. Below the bar, the quiz details are shown: 'Moodle Quiz for Course' and 'Welcome User'. The main content area has a congratulatory message: 'Congratulations! You have completed the quiz! Below is the result:'. Below this, the results are listed: 'Total Question Attempted: 10', 'Total Correct Answered: 9', 'Total Wrong Answered: 1', and 'Your Score: 90'. At the bottom, there are two buttons: 'Back to Course Page' and 'Review Quiz'.

Quiz Review page: In the quiz review page the student is able to see his answer red color if it is wrong and correct answer with green color and the explanation of the answer which is uploaded in the questions file by teacher. (Uploading questions will be discussed in course edit part!)



The screenshot shows a Moodle quiz review page. At the top, there's a blue navigation bar with 'Moodle', 'Home', 'Courses', and 'Messages' links, and a 'Welcome Student' message. Below the bar, the quiz details are shown: 'Which of the following best describes the object-oriented programming concept of polymorphism?'. There are four radio button options: 'Hiding internal mechanisms and data structures behind a defined interface' (selected and highlighted in green), 'Hiding many of the details about an object, and showing only the most relevant information' (highlighted in red), 'New classes are created from existing classes and take on the general characteristics of those classes', and 'The properties of an object can change with the context.'. Below the options, there's a text box containing the explanation: 'Polymorphism is not only hiding it is also combining together all data types and functions in one interface or class'. At the bottom, there are two navigation buttons: a left arrow and a 'Back to Course' button.

About Course Tab--> In the about course tab, the users are able to see more information about the course

The screenshot shows a Moodle course page. The top navigation bar is blue with 'Moodle' in white, and links for 'Home', 'Courses', and 'Messages'. A user profile icon and 'Welcome Student' are on the right. The course card on the left has a blue header 'Database Management System' and lists: Course Name: Database Management Systems, Course Code: COMP337, and Definition: Course for Computer and Software Engineering students. The main content area has a horizontal menu with 'Announcements', 'Outline', 'Lecture Notes', 'Lecture Videos', and 'Additional Materials'. Below this is another menu with 'Homeworks', 'Midterms & quizzes', 'Finals', and 'About Course' (which is underlined). The course details section below the menu repeats the course name, code, and definition.

Teacher User Role --> Editing Courses:

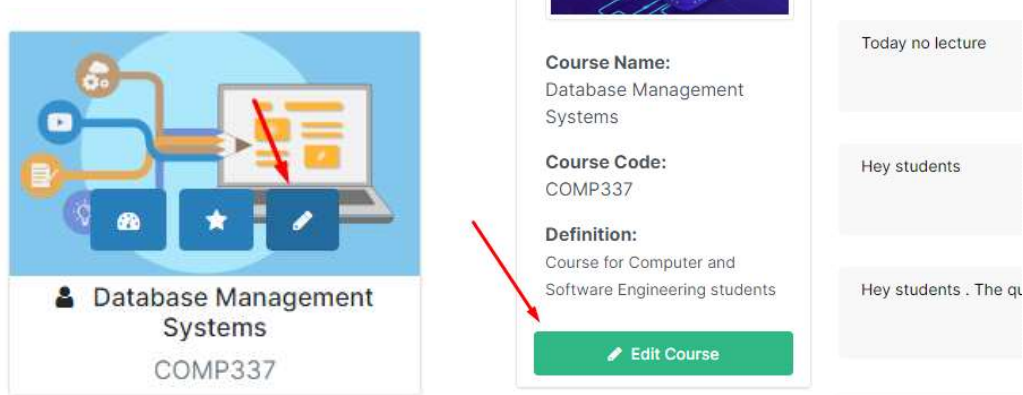
The teacher user type has an authority to edit courses or upload or update course details that students are not able to navigate these pages. For example, if students try to navigate to course edit endpoint which is responsible for editing course details or uploading lecture notes, quizzes, homeworks and etcetera, they get an error toaster saying that you cannot navigate to that page since they don't have an authority to go and use that page.



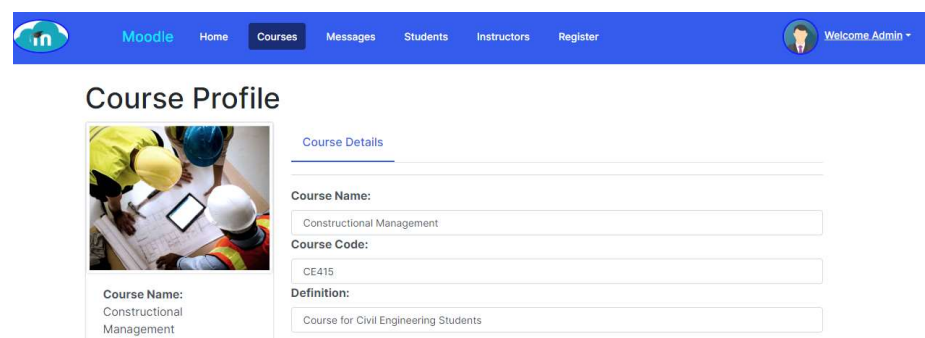
Moodle for Faculty of Engineering

Online e-learning platform

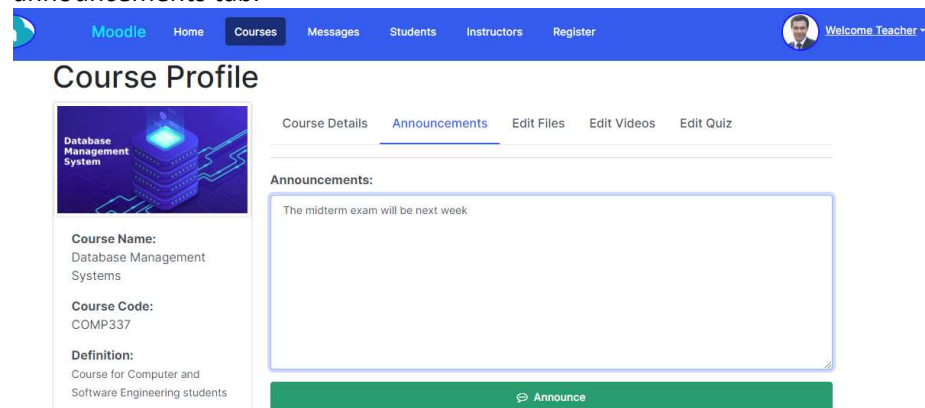
But for teacher, they are able to update course details such as uploading lecture notes, lecture videos, homeworks, quizzes, outlines, announcements. They can make these changes in courses/CourseCode/edit endpoint. For that endpoint, there are direct button links in course card and course –details pages that will be shown only if the teacher has entered as user.



Course Details tab --> For course details section admin has authority to update course code, name of the course and definition of the course whereas teacher only can change the definition of the course.

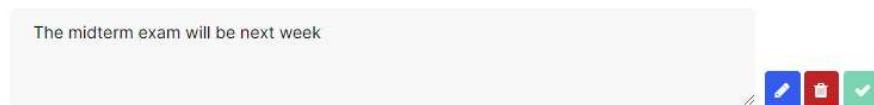


Announcement edit tab --> For adding new announcements, the user type must be teacher otherwise this section will not be shown and reached by other types of users. To announce new announcement the teacher needs to write the announcement text and press announce button to announce, then he gets the success toaster and announced in the course announcements tab.

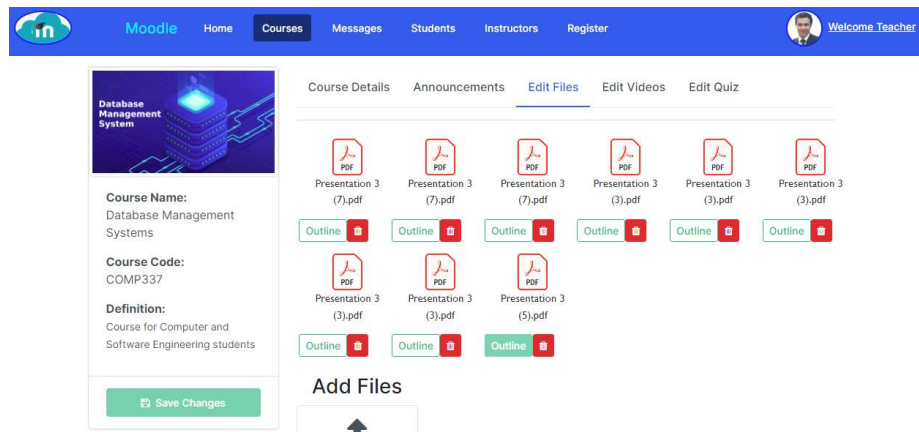


To note that, these announcements are saved in database so the teacher can

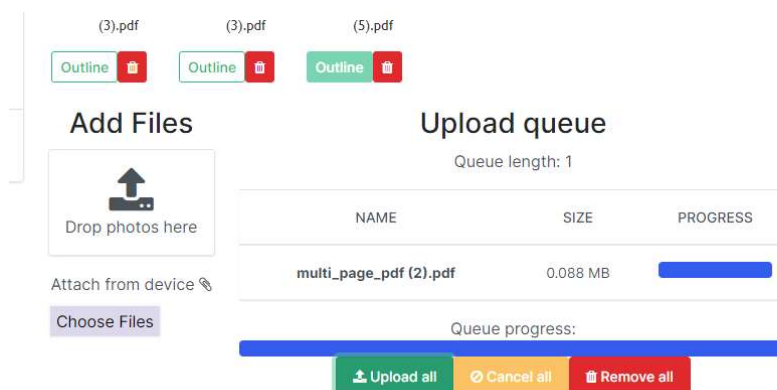
update and delete them as well.



Edit Files tab --> So in the edit files tab the teacher can upload new lecture notes and see all the uploaded files, can delete them and also set one of them as outline by simply pressing outline button and this file will be displayed as an outline to the course.




In the uploader part of the files, the user can upload by choosing a file and also by dragging the file directly as well. All files are saved in the cloudinary and the url for that file link is saved to the database so that the users can download the files by pressing on it.



Edit Videos tab --> In the edit videos tab, the teacher can enter the link for the video and give a name to video and this will be shown in the lecture videos section of the course.

Course Profile



Course Name:
Database Management Systems

Course Code:
COMP337

Definition:



[Course Details](#)
[Announcements](#)
[Edit Files](#)
[Edit Videos](#)
[Edit Quiz](#)


Video Link:

Name of Video:

[Save](#)

The teachers can delete the lecture videos since the video links and names are saved in the database


[Home](#)
[Courses](#)
[Messages](#)
[Students](#)
[Instructors](#)
[Register](#)

[Welcome Teacher](#)



Course Name:
Database Management Systems



Course Code:
COMP337



Definition:
Course for Computer and Software Engineering students



[Edit Course](#)


[Announcements](#)
[Outline](#)
[Lecture Notes](#)
[Lecture Videos](#)
[Additional Materials](#)

[Homeworks](#)
[Midterms & quizzes](#)
[Finals](#)
[About Course](#)

 **Lecture 5 video** 

 **Lecture 5 video** 

 **New Lecture 9** 

 **New**

Edit Quiz tab --> For edit quiz tab, the teacher first needs to set the time (in minutes) for the quiz and definition or explanation of the quiz so that it will be shown in the quiz main page to inform students about quiz details. After that the teacher needs to upload a file that has questions in it as json format so that the program itself (Backend side implementation) will automatically reads the file and sets a new quiz according to that questions in the file.


[Course Details](#)
[Announcements](#)
[Edit Files](#)
[Edit Videos](#)
[Edit Quiz](#)

Time:


Definition:

[Set](#)

Add Files



Drop photos here

Attach from device 

[Choose Files](#) No file chosen

Upload queue

Queue length: 1

NAME	SIZE	PROGRESS
questions.json	0.005 MB	<div></div>

Queue progress:

[Upload all](#)
[Cancel all](#)
[Remove all](#)

So the quiz made up according to the information that the teacher entered.

20 Points

Question 3 to 9

9:45

How can we specify properties and methods for an object in TypeScript?

Use classes.

Use interfaces.

Use enums.

Use async/await.

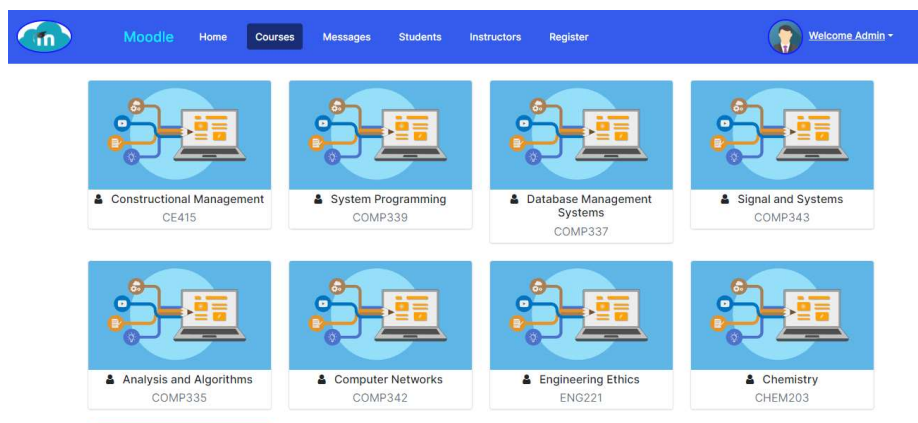
<

↺


>


Admin roles

As for the admin roles, the admin is more responsible for updating users data or course information, registering new students and teachers and also new courses, assigning courses to students or teachers and etcetera. For using the application as an admin, the user has to enter the specific number saved in the database (given by programmer) starting with @admin and specific password. Then, the admin is able to see all the courses as an initial page and he can route to another pages such as all students, teachers, registering users by clicking the route anchor tags in the nav bar.




Courses edit by admin side--> Admin can see all the courses whereas students and teachers are only able to see and use the courses that they are assigned for. Admin is able to change or update the general information of the courses however they are not able to edit the parts that the teacher can such as uploading homeworks, quizzes ...


[Moodle](#)
[Home](#)
[Courses](#)
[Messages](#)
[Students](#)
[Instructors](#)
[Register](#)


Welcome Admin

Course Profile





Course Name:
Constructional Management


Course Code:
CE415

Definition:
Course for Civil Engineering Students


Students by Admin Side--> Admin can also see all of the students profiles and update their general information, and assign course if the student has no any course.


[Moodle](#)
[Home](#)
[Courses](#)
[Messages](#)
[Students](#)
[Instructors](#)
[Register](#)



Welcome Admin




Clara Swanson
Computer Engineering




Juliette Paceas
Software Engineering




Lola Albert
Software Engineering




Sarah Gonzalez
Electrical Engineering




Letitia Quinn
Computer Engineering




Moss Skinner
Civil Engineering




Brock May
Electrical Engineering




Melton Conley
Computer Engineering




John Smith
Computer Engineering



Bruce Erik
Software Engineering





Shakhriyor Toylokov
Computer Engineering




Lola Albert
Electrical Engineering

Student Profile--> One of the good parts of the application is that, it can act as a container of information about students and teachers so that only admin has an authority to change the data of users. In the student and teacher Profiles the admin can see information about a user, courses that they are taking if student or the courses that they are assigned for if he a teacher and photos of the user.


[Moodle](#)
[Home](#)
[Courses](#)
[Messages](#)
[Students](#)
[Instructors](#)
[Register](#)


Welcome Admin



Full Name:
Juliette Paceas

Faculty:
Software Engineering

Last Active:
2020-06-20T00:00:00

About Student | [Courses](#) | [Photos](#) | [Messages](#)

Full Name: Juliette Paceas

Faculty: Software Engineering

Student Number: 155478

Email: juliepace@conveyer.com

Age: 46

Country: Uganda

Gender: female

Student since: 2019-03-03T00:00:00

[Message](#)
[Delete](#)



Full Name:
Juliette Paceas

Faculty:

About Student Courses Photos Messages

Courses

Constructional Management ↔ CE415

System Programming ↔ COMP339

Database Management Systems ↔ COMP337

Signal and Systems ↔ COMP343

Analysis and Algorithms ↔ COMP335


Editing information of users by admin--> Admin is able to update students and teachers information and assign courses. For this, there is a direct link in the student and teacher cards by pencil icon which redirects the admin to the edit page of the users.



Brock May
Electrical
Engineering


In edit page, there are two section to edit, the one is the general information about the student or teacher, the other one is the course edit part.



Moodle Home Courses Messages **Students** Instructors Register

Welcome Admin


Student Details Edit Courses



Full Name:
Brock May

Faculty:
Electrical Engineering

Last Active:
2020-06-27T00:00:00

 Save Changes

Name:
Brock

Surname:
May

Email:
brockmay@comveyer.com


Faculty:
Electrical Engineering

Student number:
131456

Country:
Russian Federation

Age:
65


For course edit, I gave authority to assign new courses only if the student does not have any courses. Otherwise, no need to change his courses.



Full Name:
Janny Alice

Faculty:
Civil Engineering

Last Active:
2020-06-20T00:00:00

 Save Changes



Student Details Edit Courses

The student has no any courses yet!

Press the button below to assign courses

New Courses

New courses button navigates the admin to assign courses page where he can assign new courses for a student. In that page he can select the courses and press assign button which assigns the courses in the database side by making relationship between a student and courses and shows that change in the student profile


[Moodle](#)
[Home](#)
[Courses](#)
[Messages](#)
[Students](#)
[Instructors](#)
[Register](#)
 Welcome Admin

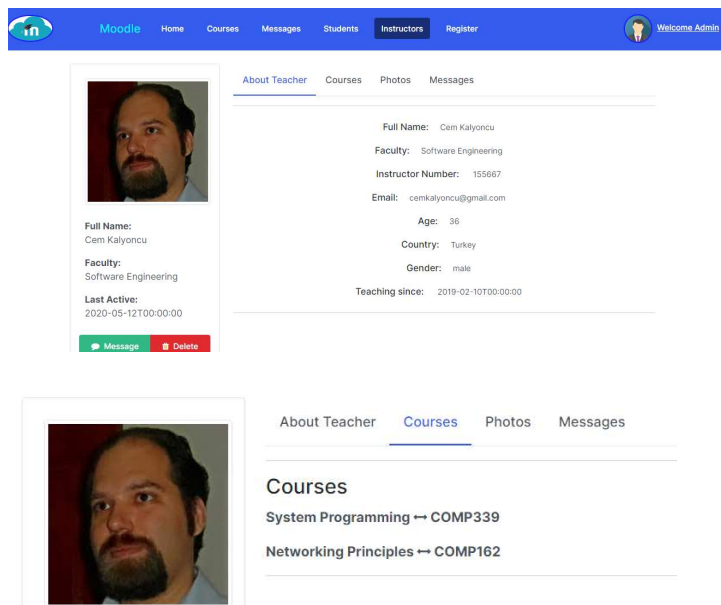
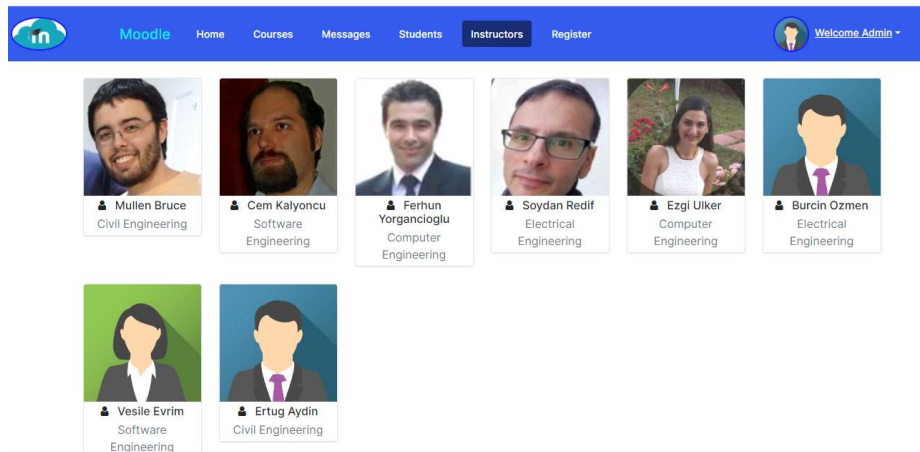
Assign New Course

Database Management Systems x
 System Programming
 Construction Management
 Signal and Systems x

☐ Select All

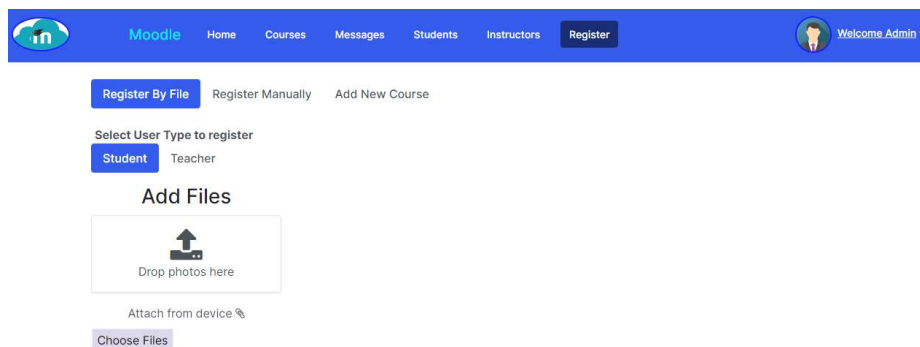
☒ Constructional Management
 ☒ System Programming
 ☒ Database Management Systems
 ☒ Signal and Systems
 ☐ Analysis and Algorithms
 ☐ Computer Networks

Teachers List & Details:--> Teachers list and details and also edit pages are same as the students. The only difference is that I did not add assign new course functionality to teachers, I decided to assign them when they are registered for the first time by admin.



Register Section of Admin

The most important section of the admin is register section where he can register new student, teacher and courses by different kind of ways.



To be more precisely, the admin can register them by manually or uploading a file which has json formatted data and the program itself reads that file and generates new users automatically whether they are students or teachers according to the file


information.

For this the admin needs to select a user type first then upload the file which has all information about new list of users and upload to the uploader.

[Register By File](#) [Register Manually](#) [Add New Course](#)

Select User Type to register
[Student](#) [Teacher](#)

Add Files


Drop photos here


Attach from device %
[Choose Files](#)


Upload queue


Queue length: 1

NAME	SIZE	PROGRESS
StudentSeedData.json	0.001 MB	<div></div>

Queue progress:

 Upload all

 Cancel all


 Remove all

Register Manually section--> For register manually section, the admin needs to fill the form one by one to register new student or teacher. It has validations as well. If any of the fields are not validated the register button is not enabled.

[Register By File](#) [Register Manually](#) [Add New Course](#)

Sign Up


Shakhriyor

Surname 

Please enter a Surname


184119

std184119

as 

Password must be at least 6 characters

std184119

as 

Password must be at least 6 characters

shaxrik99uz@gmail.com

Uzbekistan

11 February 2002

Male

Computer Engineering

Register

Cancel

So after filling all the fields by their validations the admin can register a user .

Add new course section--> There is also a section to add or register a new course and assign

teacher for that course where the admin has to enter values one by one in the form

Register By File Register Manually **Add New Course**

Add new course

Register Cancel

This was the brief explanation of working or how to use of the application. The detailed programming parts will be discussed upcoming paragraphs.

Front end of the Application:

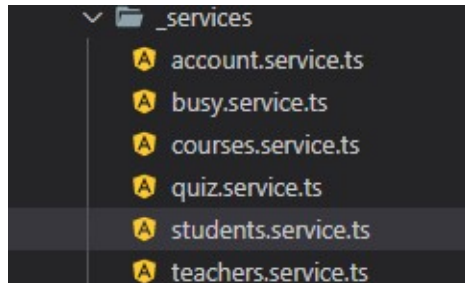
As it has been mentioned in the introduction part of the documentation, Angular 10 has been used to develop front end or client side of the application. To develop front-end side I have used many features of Angular such as services, components, routing, modules, directives, interceptors, guards, reactive forms and additional services for angular such as ngx-spinner, ngx-datepicker, ngx-file uploader, ngx-toastr, @kolkov/ngx-gallery and etcetera. The main important usage or reasons and benefits of using these features will be explained in below.

Services:

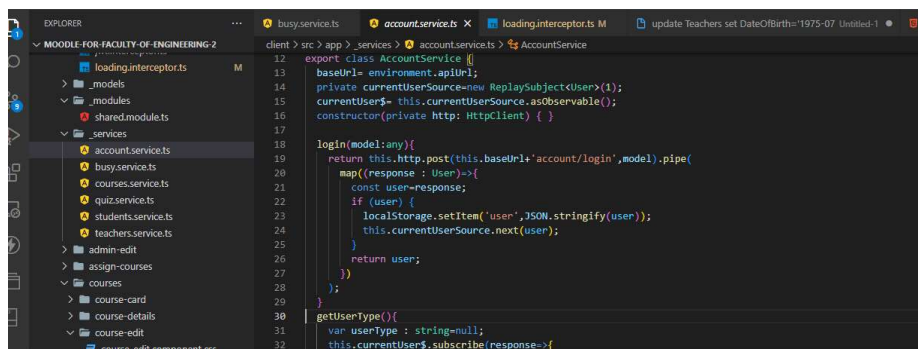
I have used services to do most important jobs in the client side. As it has been known by the technical side of the services in Angular that they are singeltons and they are alive throughout the application working or living. So, we can use of services to do most of the jobs and call them wherever we need such as components in most cases by injecting those services.

In my case, I implemented and used 6 services. Account service are used for making http requests for log in and registering, also finding out the user type and setting the user to the local storage by JWT Token so that it makes easier job to log in since JWT token compares the tokens rather than sending the values to the backend and

calling database. Student, Teacher and course services are used to work with them differently such as getting the students, teachers or courses information or updating or deleting their records by sending different requests. Busy service is used for loading spinner so that when there is a delay it shows the spinner loading instead of empty page. Lastly, quiz service is used for getting the uploaded questions by the teacher for quizzes.

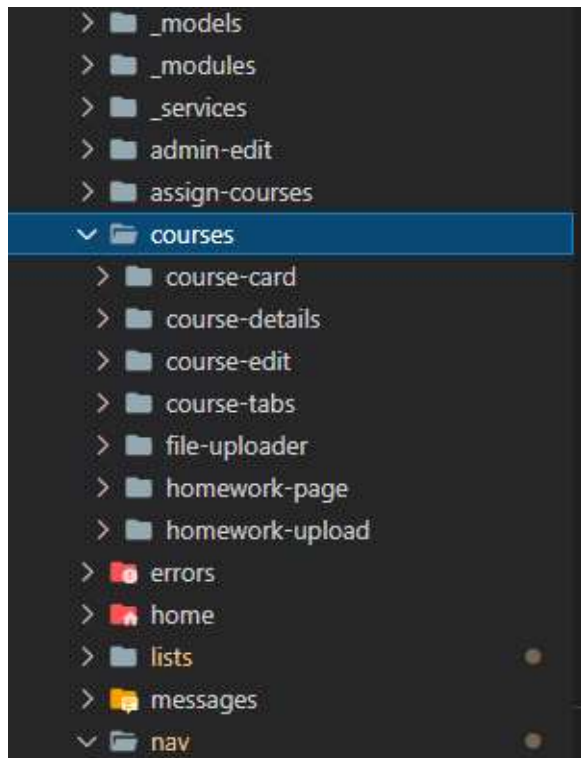


E.g: Account Service



Components:

The components are the main building blocks of the front end side since it has the main .ts file for using different services or angular features or defining the business logic of the front end and also html and css parts so that the developer can display the records as they want. In my application I have used plenty of components by giving them different jobs that helped me to separate the front-end side implementation easily. I stored different components in different folders such as courses folder which includes course-card, course-edit, course-details, course-tabs, file uploader, homework uploader components where each components business logic is different. And other components also saved different directories such as student, teacher, admin,nav...



Routing :

Routing feature of Angular is used to route the user different pages by assigning the components to different routes. In this routing, I have also make use of guards so that some components has some guards to prevent unassigned or unauthorized users for that routes which gives a great advantage to navigate the students to edit pages or register pages as an example. Also the main guard is to guard the not logged in users to navigate to other pages other than the main page of the application

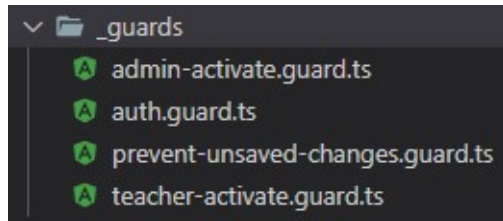
```

1 path: '',
2 runGuardsAndResolvers: 'always',
3 canActivate: [AuthGuard],
4 children: [
5   {path: 'students', component: StudentListsComponent},
6   {path: 'teachers', component: TeachersListsComponent},
7   {path: 'register-manually', component: RegisterManuallyComponent, canActivate: [AdminActivateGuard]},
8   {path: 'register', component: RegisterComponent, canActivate: [AdminActivateGuard]},
9   {path: 'courses', component: CoursesListsComponent},
10  {path: 'messages', component: MessagesComponent},
11  {path: 'students/:username', component: StudentDetailsComponent},
12  {path: 'teachers/:username', component: TeacherDetailsComponent},
13  {path: 'courses/:coursecode', component: CourseDetailsComponent},
14  {path: 'students/:username/edit', component: StudentEditComponent, canDeactivate: [PreventUnsavedChanges]},
15  {path: 'teachers/:username/edit', component: TeacherEditComponent, canActivate: [TeacherActivateGuard]},
16  {path: 'courses/:coursecode/edit', component: CourseEditComponent, canActivate: [TeacherActivateGuard]},
17  {path: 'admin/edit', component: AdminEditComponent},
18  {path: 'student/:username/edit', component: StudentSettingsComponent},
19  {path: 'students/:username/edit/assign-course', component: StudentAssignCourseComponent},
20  {path: 'courses/:coursecode/quiz', component: QuizPageComponent},
21  {path: 'courses/:coursecode/homework-upload', component: HomeworkUploadComponent}
22 ]

```

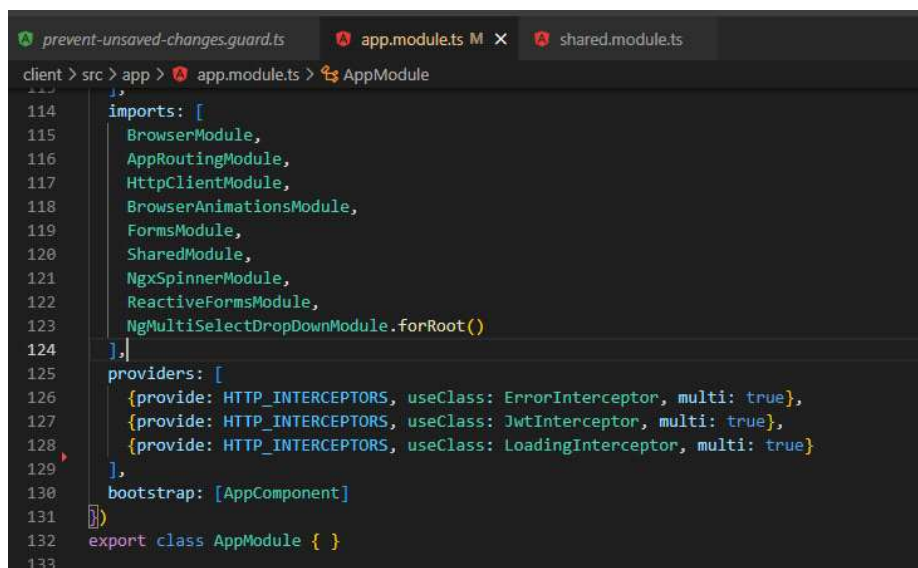
Guards in detail:

As we know technically, guards are used to prevent unauthorized users to navigate to particular pages or components that has guarded. I have used 4 guards in my application those are admin-activate guard which is used for to give authority only admin to navigate that pages, auth-guard which is the main guard to prevent not logged in students to navigate other pages without authentication, teachers guard that is responsible for only teachers to navigate these pages such as adding course homeworks, quizzes, lecture notes, lecture videos and etcetera, and finally prevent unsaved changes guard to warn the users to show the message that the unsaved changes will be lost if the user wants to navigate to other page when updating the data.



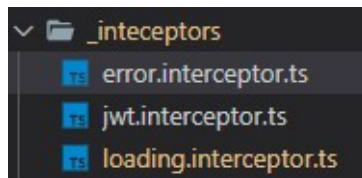
Modules :

Modules are used to import new service modules and to declare new components and directives and provide interceptors in my application. By the help of modules I would be able to introduce new services' modules such as toastr module, file uploader module and others to use in the application. To note that, I have made different module called shared module to introduce those services modules and export and used this shared module inside the main module. Also interceptors also declared and provided in the main app module.



Interceptors :

In the application three interceptors are used those are error interceptor, jwt interceptor, loading interceptor. Generally interceptors are used to deal with all the requests and responses and catch anything between them and modify them.



So, error interceptor is used to catch any error if occurred in the request and responses and behave correctly such as sending to particular error pages or showing an error toastr with proper message. More precisely, it is kind of the implementation of error handling in front end side.

```
intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
  return next.handle(request).pipe(
    catchError(error=>{
      if (error) {
        switch (error.status) {
          case 400:
            if (error.error.errors) {
              const modalStateErrors=[];
              for(const key in error.error.errors){
                if (error.error.errors[key]) {
                  modalStateErrors.push(error.error.errors[key]);
                }
              }
              throw modalStateErrors.flat();
            }
            else{
              this.toastr.error(error.error,error.status);
            }
            break;
          case 401:
            this.toastr.error(error.error === null ? "Unauthorized" : error.error, error.status);
            break;
          case 404:
            this.router.navigateByUrl('/not-found');
            break;
          case 500:
            const navigationExtras: NavigationExtras = {state: {error: error.error}}
            this.router.navigateByUrl('/server-error',navigationExtras)
            break;
          default:
            this.toastr.error("Something unexpected went wrong!");
            console.log(error);
            break;
        }
      }
    })
  );
}
```

For Jwt interceptor , it is used to set JWT authentication bearer when the user has logged in.

```
intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
  let currentUser: User;
  this.accountService.currentUser$.pipe(take(1)).subscribe(user=>currentUser=user);
  if (currentUser) {
    request= request.clone({
      setHeaders: {
        Authorization: `Bearer ${currentUser.token}`
      }
    })
  }
  return next.handle(request);
}
```

And for loading interceptor, it is used to control or start and stop busy service for loading

spinner when there is a delay

```
intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {  
    this.busyService.busy();  
  
    return next.handle(request).pipe(  
        delay(100),  
        finalize(()=>{  
            this.busyService.idle();  
        })  
    );  
}
```

Directives:

Technically, Directives are classes that add additional behavior to elements for the application. So I have used only 2 directives both of them to implement quiz features.

One is change background directive to change the background color to green when click event occurred by the user when answered .

```
@HostListener('click') answer(){  
    if(this.isCorrect ){  
        this.render.setStyle(this.el.nativeElement,'background','green');  
        this.render.setStyle(this.el.nativeElement,'color','#fff');  
        this.render.setStyle(this.el.nativeElement,'border','2px solid grey');  
    }  
}
```

The other one is review quiz directive to set color to green for right answers and red for answer if the user answer is incorrect.

```
check(){  
    if(this.isCorrect ){  
        this.render.setStyle(this.el.nativeElement,'background','green');  
        this.render.setStyle(this.el.nativeElement,'color','#fff');  
        this.render.setStyle(this.el.nativeElement,'border','2px solid grey');  
    }  
    else{  
        this.render.setStyle(this.el.nativeElement,'background','red');  
        this.render.setStyle(this.el.nativeElement,'color','#fff');  
        this.render.setStyle(this.el.nativeElement,'border','2px solid grey');  
    }  
}
```

Reactive Forms:

In the application reactive forms are used to make up forms and apply front end validations and get the entered data to the forms and send them to the proper requests.

So first I have initilized the form by form builder by giving the names for the properties of the form

```

initilizeForm(){
  this.registerForm= this.fb.group({
    username: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(15),
    this.matchValues('Idnum'), this.isContainingNumbers()]],
    name: ['', [Validators.required]],
    surname: ['', [Validators.required]],
    Idnum: ['', [Validators.required, Validators.minLength(3), Validators.maxLength(12)]],
    gender: ['male', [Validators.required]],
    email: ['', [Validators.minLength(6)]],
    country: ['', [Validators.required]],
    dateOfBirth: ['', [Validators.required]],
    faculty: ['', [Validators.required]],
    password: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(15)]]
  })

  this.registerForm.controls.Idnum.valueChanges.subscribe(()=>{
    this.registerForm.controls.username.updateValueAndValidity();
  })
}

```

Apart from the Validations that comes with Reactive forms such as Validators.required and others, we can also make up our own validations and set them in the proper fields of the form. E.g:

```

matchValues(matchTo: string): ValidatorFn{
  return (control: AbstractControl)=>{
    if(matchTo==='Idnum'){
      var usernameString: string=control?.value
      var stringNums= usernameString?.slice(3);
      console.log(stringNums);

      return stringNums===control?.parent?.controls[matchTo].value ? null: {isMatchingIdnum:true}
    }
    return control?.value===control?.parent?.controls[matchTo].value ? null: {isMatching:true}
  }
}

isContainingNumbers(): ValidatorFn{
  return (control: AbstractControl)=>{
    var usernameString = control.value;
    var stringNums= usernameString?.slice(3);
    var matches = stringNums?.match(/\d/g);
    console.log(matches);

    return matches===null ? null: { isContainNums: true}
  }
}

```

In my own validators, for example in match values validator method it compares the student or Instructor number and the username of the user which starts with std or ins and Id number so that if they don't match the validator error message will show up. And for isContainingNumber method validator, I planned to observe the student or instructor number to find out that they only contain numbers also for username field to observe that it contains only numbers after std or ins otherwise the error will show up and does not let the admin to register the user until the validations are valid. To show the validation error messages, I used another component called text-input by sending the validation field and messages so that it makes the code clearer rather than combining all the code in one place.

```

1 <div class="form-group">
2   <input type={{type}}
3     [class.is-invalid]="ngControl.touched && ngControl.invalid"
4     class="form-control"
5     [formControl]="ngControl.control"
6     placeholder={{label}}>
7   <div *ngIf="ngControl.control.errors?.required"
8     class="invalid-feedback">Please enter a {{label}}</div>
9   <div *ngIf="ngControl.control.errors?.minlength"
10    class="invalid-feedback">{{label}} must be at least {{ngControl.control.errors.minlength['requiredLength']}} characters</div>
11   <div *ngIf="ngControl.control.errors?.maxlength"
12    class="invalid-feedback">{{label}} must be at most {{ngControl.control.errors.maxlength['requiredLength']}} characters</div>
13   <div *ngIf="ngControl.control.errors?.isMatching"
14    class="invalid-feedback">{{label}} must match password</div>
15   <div *ngIf="ngControl.control.errors?.isMatchingIdnum"
16    class="invalid-feedback">{{label}} number must match Id Number</div>
17   <div *ngIf="ngControl.control.errors?.isContainNums"
18    class="invalid-feedback">{{label}} should not contain letters after {{userType}}</div>
19
20 </div>
21

```

Control value accessor is used to access the values of the form field (control)

```

17
export class TextInputComponent implements ControlValueAccessor {
  @Input() label: string;
  @Input() type: 'text';
  @Input() userType: string;
  constructor(@Self() public ngControl: NgControl) {
    this.ngControl.valueAccessor = this;
  }
  writeValue(obj: any): void {
  }
  registerOnChange(fn: any): void {
  }
  registerOnTouched(fn: any): void {
  }
}

```

Backend of the Application:

As it has been mentioned C# .NET 5 was used to implement backend side of the application. The reason why I have used C# .NET framework is that making web applications is easy and understandable by its MVC architecture and secure by authentication and authorization and also supports many new features to develop the application.

As for the skeleton of the backend, firstly I made a plan to implement user types and their properties by entities as Model in MVC architecture. After that I connected Database to these entities by creating tables for different user types such as

student, teacher and admin, I made up controller methods to get the individual user an list of users. Meaning that, I have made implementations for the backend step by step which is know as baby-steps between programmers that makes the implementation of the program understandable and prevent unwanted errors.

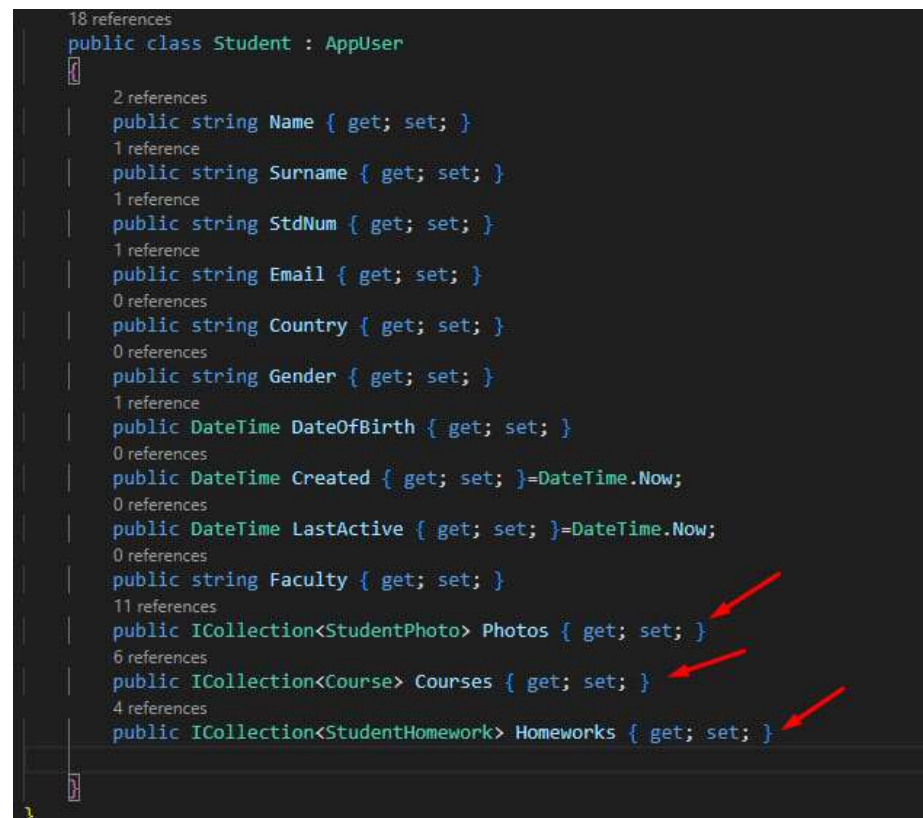
Entities:

As for user type entities, there are three types of users such as student, teacher and admin with their properties and methods. As well as that, there are Courses, Photos, Faculties and some more entities for course details such as homeworks, quizzes and etcetera.

Importantly, the entities are related with each other with the different kinds of relationship methods such as one-to-many, many-to-one, one-to-one, many-to-many. For example, Student and course are related by many-to-many relationship meaning that, a student can have many courses and a course can have many students.

Student Entity Relationships

```
18 references
public class Student : AppUser
{
    2 references
    public string Name { get; set; }
    1 reference
    public string Surname { get; set; }
    1 reference
    public string StdNum { get; set; }
    1 reference
    public string Email { get; set; }
    0 references
    public string Country { get; set; }
    0 references
    public string Gender { get; set; }
    1 reference
    public DateTime DateOfBirth { get; set; }
    0 references
    public DateTime Created { get; set; }=DateTime.Now;
    0 references
    public DateTime LastActive { get; set; }=DateTime.Now;
    0 references
    public string Faculty { get; set; }
    11 references
    public ICollection<StudentPhoto> Photos { get; set; }
    6 references
    public ICollection<Course> Courses { get; set; }
    4 references
    public ICollection<StudentHomework> Homeworks { get; set; }
}
```

A screenshot of a code editor showing the C# code for the Student entity. The code is a public class Student that inherits from AppUser. It contains several properties: Name, Surname, StdNum, Email, Country, Gender, DateOfBirth, Created, LastActive, Faculty, Photos, Courses, and Homeworks. The Photos, Courses, and Homeworks properties are of type ICollection and are annotated with 11, 6, and 4 references respectively. Red arrows point to these three collection properties.

Course Entity Relationships


```

public class Course
{
    2 references
    public int Id { get; set; }
    1 reference
    public string NameOfCourse { get; set; }
    15 references
    public string CourseCode { get; set; }
    0 references
    public string Definition { get; set; }
    0 references
    public string photoUrl { get; set; }
    0 references
    public DateTime LastAccessed { get; set; } = DateTime.Now;
    9 references
    public ICollection<Announcements> Announcements { get; set; }
    0 references
    public ICollection<Faculty> Faculties { get; set; }
    1 reference
    public Teacher Teacher { get; set; }
    1 reference
    public int TeacherId { get; set; }
    0 references
    public ICollection<Student> Students { get; set; }
    12 references
    public ICollection<CourseUploadFile> CourseFiles { get; set; }
    7 references
    public ICollection<LectureVideos> LectureVideos { get; set; }
    2 references
    public ICollection<Homework> Homeworks { get; set; }
    2 references
    public ICollection<QuizFiles> QuizFiles { get; set; }
}

```

They are connected with each other in the database side as well.

Controllers

Controllers are the main block of the API. They are used for different methods of the requests such as get, post, put, and delete method requests. So in my application as well I have implemented all the RESTFULL API methods.

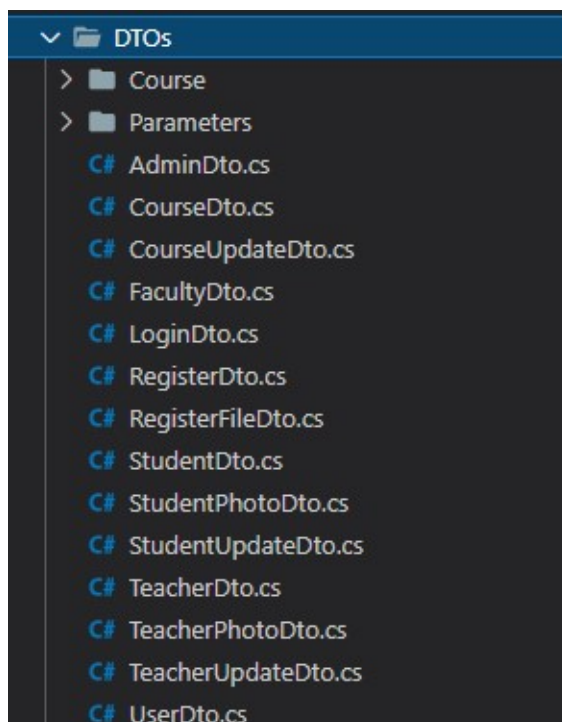
I divided the controllers according to different business logics such as Account controller for only dealing with the works of accounts such as log in, register. Student and Teacher controllers to deal with the methods of the students and teachers respectively for example getting the students or teachers, updating their records, posting new records such as homeworks for student, and deleting the records from the database. Importantly, I used AutoMapper to map the updated or posted records sent by the request body to map with the entities' properties.

```

public async Task<ActionResult> UpdateStudentCourse(StudentCoursesUpdateDto studentUpdateDto)
{
    var username = studentUpdateDto.Username;
    var student = await _userRepository.GetStudentByUsernameAsync(username);
    for (int i = 0; i < student.Courses.Count; i++)
    {
        for (int j = 0; j < studentUpdateDto.Courses.Count; j++)
        {
            if (student.Courses.ElementAt(i).Id == studentUpdateDto.Courses.ElementAt(j).Id)
            {
                var samecourse = student.Courses.ElementAt(i);
                student.Courses.Remove(samecourse);
            }
        }
    }
    var updatedCourses = _mapper.Map(studentUpdateDto, student);
    try
    {
        _userRepository.UpdateStudent(student);
    }
    catch (System.Exception ex)
    {
        Console.WriteLine(ex.Message);
        return BadRequest("Failed to add Course");
    }
    if (await _userRepository.SaveAllChangesAsync()) return NoContent();
    return BadRequest("Failed to update student course!");
}

```

DTO's--> Dtos also used to accept the request body object parameters and for returning the data. By using dtos to return data instead of entity it self, it helps us to hide some information from the user. So that we can return the data as like as we want and prevent sending some data back to the user for hiding purposes. There are plenty of DTOs as entities.



Moreover, there is also a course controller to control the business logic of the courses such as getting the courses, posting new courses, posting new records to existing courses, updating the records and deleting the course records as well.

Importantly, the methods for announcement, file uploading, lecture notes, lecture videos uploading, quiz uploading, assigning new homework and other course edit features are implemented in this controller

E.g: add-quiz endpoint method

```
return BadRequest("Problem occurred while uploading announcement.");
}
[HttpPost("add-quiz")]
0 references
public async Task<ActionResult<LectureVideosDto>> AddQuizFile(IFormFile file, string courseCode, int time)
{
    var course = await _context.Courses.Include(x => x.QuizFiles).AsSplitQuery()
        .SingleOrDefaultAsync(x => x.CourseCode.ToLower() == courseCode.ToLower());
    var result = file.FileName.Contains(".pdf") ? await _fileService.AddPDFFileAsync(file) :
        await _fileService.AddFileAsync(file);

    if (result.Error != null) return BadRequest(result.Error.Message);
    await _seedService.SeedQuiz(_context, file);
    var quizFile = new QuizFiles
    {
        Time = time,
        QuizDefinition = definition,
        Url = result.SecureUrl.AbsoluteUri,
        PublicId = result.PublicId,
        FileName = file.FileName
    };
    course.QuizFiles.Add(quizFile);
    if (await _context.SaveChangesAsync() > 0)
    {
        return Ok("Successfully uploaded quiz");
    }
    return BadRequest("Problem occurred while uploading announcement!");
}
```

There is also Buggy controller to handle the errors or sending back proper responses when the specific error is occurred


```

3 references
private readonly DataContext _context;
0 references
public BuggyController(DataContext context)
{
    _context = context;
}

[Authorize]
[HttpGet("auth")]
0 references
public ActionResult<string> GetAuthError(){
    return "You are authorized to see this!";
}

[HttpGet("not-found")]
0 references
public ActionResult<AppUser> GetNotFoundError(){

    var thing = _context.Students.Find(-1);
    if (thing==null)
    {
        return NotFound("The user is not found");
    }
    return Ok(thing);
}

[HttpGet("server-error")]
0 references
public ActionResult<string> GetServerError(){
    var thing = _context.Students.Find(-1);
    var thingToReturn = thing.ToString();
    return thingToReturn;
}

```

Cloudinary

Cloudinary was used to save the files and photos to the cloud and the links for that files and photos are saved to the database so that the users can download them by the provided link in the client side.

```

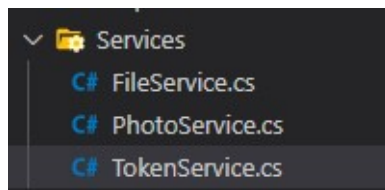
[HttpPost("add-photo")]
0 references
public async Task<ActionResult<StudentPhotoDto>> AddPhoto(IFormFile file)
{
    var username = User.GetUsername();
    var user = await _userRepository.GetStudentByUsernameAsync(username);
    var result = await _photoService.AddPhotoAsync(file);
    if (result.Error != null) return BadRequest(result.Error.Message);
    var photo = new StudentPhoto
    {
        Url = result.SecureUrl.AbsoluteUri,
        PublicId = result.PublicId
    };
    user.Photos.Add(photo);
    if (await _userRepository.SaveAllChangesAsync())
    {
        return CreatedAtRoute("GetStudent", new { username = user.UserName },
            _mapper.Map<StudentPhotoDto>(photo));
    }

    return BadRequest("Problem occurred while adding a photo!");
}

```

Services in Backend:

Different services are used to do different jobs and they make the code much clearer by injecting the service and calling the method by hiding the implementations rather than combining everything inside the controller.



There are three service classes implemented and used for the backend in my application such as FileService to upload different types of files to the cloudinary according to the filenames and delete them from cloudinary also whenever the user wants to delete.

```
1 reference
public class FileService : IFileService
{
    4 references
    private readonly Cloudinary _cloudinary;

    0 references
    public FileService(IOptions<CloudinarySettings> config)
    {
        var acc = new Account(
            config.Value.CloudName,
            config.Value.ApiKey,
            config.Value.ApiSecret
        );
        _cloudinary = new Cloudinary(acc);
    }

    4 references
    public async Task<RawUploadResult> AddFileAsync(IFormFile file)
    {
        var uploadResult = new RawUploadResult();
        if (file.Length > 0) {
            using var stream = file.OpenReadStream();
            var uploadParams = new RawUploadParams() {
                File = new FileDescription(file.FileName, stream),
            };
            uploadResult = await _cloudinary.UploadAsync(uploadParams);
        }
        return uploadResult;
    }
}
4 references
```

Same as photo service, it is used to upload photos and delete photos from the cloudinary.

As for the token service it is considered one of the most important parts of the application to provide the job of creating the tokens to the users

```

public class TokenService : ITokenService
{
    2 references
    private readonly SymmetricSecurityKey _key;
    0 references
    public TokenService(IConfiguration config)
    {
        _key=new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]));
    }

    6 references
    public string CreateToken(AppUser user)
    {
        var claims= new List<Claim>{
            new Claim(JwtRegisteredClaimNames.NameId,user.UserName)
        };

        var creds= new SigningCredentials(_key,SecurityAlgorithms.HmacSha512Signature);

        var tokenDescriptor= new SecurityTokenDescriptor{
            Subject=new ClaimsIdentity(claims),
            Expires=DateTime.Now.AddDays(7),
            SigningCredentials=creds
        };

        var tokenHandler= new JwtSecurityTokenHandler();
        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}

```

Seed method

Seed method is considered as the main method for uploading files and reading the data in these files and implementing the business logic to get the desired output. For example, in this class there are methods for seeding the users when the admin uploads a file with data of users to register these methods come on to the field to implement the logic to read these data and register users.

```

1 reference
public async Task SeedStudents(DataContext context, IFormFile file)
{
    var filename= ReadAsStringAsync(file).Result.ToString();
    var students = JsonSerializer.Deserialize<List<Student>>(filename);
    foreach (var student in students)
    {
        using var hmac= new HMACSHA512();
        student.UserName=student.UserName.ToLower();
        student.PasswordHash= hmac.ComputeHash(Encoding.UTF8.GetBytes("password"));
        student.PasswordSalt= hmac.Key;
        context.Students.Add(student);
    }
    await context.SaveChangesAsync();
}

3 references
public static async Task<string> ReadAsStringAsync(IFormFile file)
{
    var result = new StringBuilder();
    using (var reader = new StreamReader(file.OpenReadStream())){
        while (reader.Peek() >= 0)
            result.AppendLine(await reader.ReadLineAsync());
    }
    return result.ToString();
}

```

As well as that, there is a method to read the quiz uploaded file and construct the quiz according to the questions inside the file. It saves the questions in the local file.

```

1 reference
public async Task SeedQuiz(DataContext context, IFormFile file)
{
    var path= "C:/Users/Dell/Downloads/GR_project/moodle-for-faculty-of-engineering-2/client/s
    var fileString= ReadAsStringAsync(file).Result.ToString();
    File.WriteAllText(path, fileString);
    await context.SaveChangesAsync();
}
1 reference

```

Security of the Application:

In the application, I tried to make it as much as secure by programming side. Also, I made it run in HTTPS protocol rather than HTTP, which provides our application to be effectively secure in the browser as well.

As a first types of security, I added to the controllers attributes such as [Authorize] so that the unauthorized users do not have an authority to use these controllers and reach their methods. Meaning that using the main functionalities of the application users needs to be logged in.

```

namespace API.Controllers
{
    [Authorize]
    0 references
    public class StudentsController : BaseApiController
    {
        22 references
        private readonly IStudentRepository _userRepository;
        5 references
        private readonly IMapper _mapper;
        3 references
        private readonly IPhotoService _photoService;
        2 references
        private readonly DataContext _context;

        3 references
        private readonly IFileService _fileService;
        0 references
        public StudentsController(IStudentRepository userRepository, IMapper mapper, DataContext context
        , IPhotoService photoService, IFileService fileService)
    {

```

Account Security

As well as that, for the account security, the passwords are not saved to the database directly, firstly they are hashed and then salted by HMACSHA512 algorithm then this hashed and salted long strings are saved to the database. So that only this algorithm can decode back the passwords to compare to the original password. This gives us benefit that, even if the hackers can have an access our database they are not able to see the passwords saved in database.

```

[HttpPost("register-student")]
1 reference
public async Task<ActionResult<UserDto>> RegisterStudent(RegisterDto registerDto)
{
    if(await StudentExists(registerDto.Username)) return BadRequest("Username is taken!");
    var student = _mapper.Map<Student>(registerDto);
    student.StdNum=registerDto.IdNum;
    using var hmac = new HMACSHA512();
    student.UserName=registerDto.Username.ToLower();
    student.PasswordHash= hmac.ComputeHash(Encoding.UTF8.GetBytes(registerDto.Password));
    student.PasswordSalt=hmac.Key;

    _context.Students.Add(student);
    await _context.SaveChangesAsync();

    return new UserDto{
        Username=student.UserName,
        Token= _tokenService.CreateToken(student)
    };
}

```

For all kind of users the passwords are hashed and saved the hashed string to the database.

JWT Tokens

For the authentication of the users, JWT is used since it allows decode, verify and generate user tokens and compare the tokens themselves rather than comparing all the data in the database and it gives also performance increase since it does not need to go to the database to compare the values and authenticate.

```

public class TokenService : ITokenService
{
    2 references
    private readonly SymmetricSecurityKey _key;
    0 references
    public TokenService(IConfiguration config)
    {
        _key=new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["TokenKey"]));
    }

    6 references
    public string CreateToken(AppUser user)
    {
        var claims= new List<Claim>{
            new Claim(JwtRegisteredClaimNames.NameId,user.UserName)
        };

        var creds= new SigningCredentials(_key,SecurityAlgorithms.HmacSha512Signature);

        var tokenDescriptor= new SecurityTokenDescriptor{
            Subject=new ClaimsIdentity(claims),
            Expires=DateTime.Now.AddDays(7),
            SigningCredentials=creds
        };

        var tokenHandler= new JwtSecurityTokenHandler();
        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}

```

For generating tokens for every user, there is a class made up to write tokens called TokenService.

So this class CreateToken() method is called when the user is registered for the first time and logged in to the application and the tokens are compared so that if they match the user is authenticated to use the other parts of the applications otherwise they will not be able to continue.

```

[HttpPost("register-teacher")]
1 reference
public async Task<ActionResult<UserDto>> RegisterTeacher(RegisterDto registerDto)
{
    if(await TeacherExists(registerDto.Username)) return BadRequest("Username is taken!");
    var teacher = _mapper.Map<Teacher>(registerDto);
    teacher.InsNum=registerDto.IdNum;
    using var hmac = new HMACSHA512();

    teacher.UserName=registerDto.Username.ToLower();
    teacher.PasswordHash= hmac.ComputeHash(Encoding.UTF8.GetBytes(registerDto.Password));
    teacher.PasswordSalt=hmac.Key;
    _context.Teachers.Add(teacher);
    await _context.SaveChangesAsync();
    return new UserDto{
        Username=teacher.UserName,
        Token= _tokenService.CreateToken(teacher) ←
    };
}

```

The token is saved in the browser, so when the user send a request to log in the browser will send him token response so the token can be decoded by JWT token decoder so that if the user data matches with token decoded data, the user is considered to be authenticated.

Example of token decoding

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMekKF2QT4fwpMeJf36P0k6yJV_adQssw5c

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

{
"alg": "HS256",
"typ": "JWT"
}

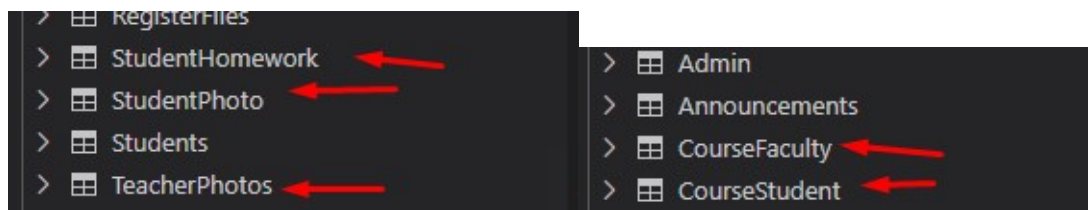
PAYLOAD: DATA

{
"sub": "1234567890",
"name": "John Doe",
"iat": 1516239022
}

VERIFY SIGNATURE

Database of the Application:

As for the database, the relational database is used throughout the application since it is considered to be the best choice when we have relationships between data tables. The application database tables have many of relationship, exempli gratia, relationships between courses and students or teachers, relationship between users and photos, students and homeworks and etcetera. Since those relations are made up in entities using entity framework relationships, for the database side as well the relationships are made up .



And in those related tables, only the id of the tables are stored to make relationship between them.

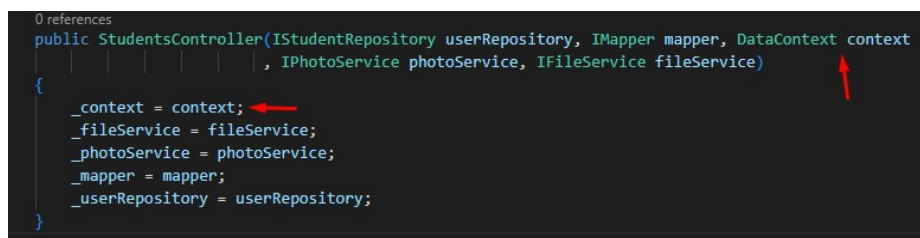
So that it shows and separates clearly the relationships from other data of the database tables



CoursesId	StudentsId
1	2
1	3
2	2
2	6
3	2
3	5
3	7
3	8
4	8
4	2
4	5
5	2
5	5
6	3
6	7
6	8
5	4

Data Context:

To make a bridge between the code and database, Data context class was implemented in the backend which injects DbContext which gives us possibility to get, insert, update, delete the records from the database. To use this data context in the controller classes we need to inject it first and make use of it's DbSet's and query the data of the database.



```
0 references
public StudentsController(IStudentRepository userRepository, IMapper mapper, DataContext context
    , IPhotoService photoService, IFileService fileService)
{
    _context = context;
    _fileService = fileService;
    _photoService = photoService;
    _mapper = mapper;
    _userRepository = userRepository;
}
```



```

[HttpPut]
0 references
public async Task<ActionResult> UpdateStudent(StudentUpdateDto studentUpdateDto)
{
    var username = studentUpdateDto.Username;
    var student = await _context.Students. ←
        SingleOrDefaultAsync(x => x.UserName.ToLower() == username.ToLower());
    var updatedstd = _mapper.Map(studentUpdateDto, student);
    _userRepository.UpdateStudent(updatedstd);
    if (await _userRepository.SaveAllChangesAsync()) return NoContent();
    return BadRequest("Failed to update student!");
}

```

In this class, the DbSet's are made up to have an access to the data of the important tables into the code by the entities.

```

public class DataContext : DbContext
{
    0 references
    public DataContext(DbContextOptions options) : base(options)
    {
    }

    13 references
    public DbSet<Student> Students { get; set; } ←
    11 references
    public DbSet<Teacher> Teachers { get; set; }
    9 references
    public DbSet<Administrator> Admin { get; set; }
    16 references
    public DbSet<Course> Courses { get; set; }
    4 references
    public DbSet<Faculty> Faculties { get; set; }
    1 reference
    public DbSet<Announcements> Announcements { get; set; }
    0 references
    public DbSet<RegisterFile> RegisterFiles { get; set; }
}

```

As well as that, Model builder function is also implemented to assign some relationship with different delete behaviours, to mention the real id of the entities to make relationship and to prevent some warnings during the querying process.

```

1 reference
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    base.OnConfiguring(optionsBuilder); ←
    optionsBuilder.ConfigureWarnings(w=>w.Ignore(CoreEventId.RowLimitingOperationWithoutOrderByWarn
}

1 reference
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Course>()
        .HasKey(x=>x.Id); ←
    modelBuilder.Entity<StudentPhoto>(entity=>{
        entity.HasOne(pt => pt.Student)
        .WithMany(t => t.Photos)
        .HasForeignKey(pt => pt.StudentId)
        .OnDelete(DeleteBehavior.Cascade); ←
    });

    modelBuilder.Entity<Course>()
        .HasOne(x=>x.Teacher)
        .WithMany(y=>y.Courses).HasForeignKey(x=>x.TeacherId);
}

```

Repository Pattern:

To deal with the database works, the repository pattern architecture or logic is used in the backend. To be more precisely, rather than combining all the code together inside the controller it is better to separate them to some other services which gives us more clear code and the program is considered to obey the principles of the clear coding. So, for the database works I have separated them into another class and that class contains methods to get the different data from database update and save all changes according to the student, teacher repositories separately.

```
0 references
public async Task<Student> GetByIdAsync(int id)
{
    return await _context.Students.FindAsync(id);
}

7 references
public async Task<Student> GetByUsernameAsync(string username)
{
    return await _context.Students.Include(x=>x.Courses).Include(x=>x.Photos).Include(x=>x.Homework)
        .SingleOrDefaultAsync(x=>x.UserName.ToLower() == username.ToLower());
}

0 references
public async Task<IEnumerable<Student>> GetAllAsync()
{
    return await _context.Students.Include(x=>x.Courses).Include(x=>x.Photos).AsSplitQuery().ToListAsync();
}

1 reference
public async Task<IEnumerable<StudentDto>> GetMemberStudentsAsync()
{
    return await _context.Students
        .ProjectTo<StudentDto>(_mapper.ConfigurationProvider).AsSingleQuery()
        .ToListAsync();
}
```

For example, above methods of the student repository allows the programmer to get different data by different ways, as an example, the user should be able to get the data by the username, Id and the other GetMember methods are used to map the values in the repository itself rather than in the controller. It would be a plenty of work in the controller and make our code as spaghetti code if I would not use repository pattern. For these repository classes, Interfaces are generated so that it will make the other programmers easy to read and understand the code and makes us of the methods easier in the injected classes of this interface.

```

0 references
public interface IStudentRepository
{
    3 references
    void UpdateStudent(Student student);
    9 references
    Task<bool> SaveAllChangesAsync();
    0 references
    Task<IEnumerable<Student>> GetStudentsAsync();
    0 references
    Task<Student> GetStudentByIdAsync(int id);
    7 references
    Task<Student> GetStudentByUsernameAsync(string username);
    1 reference
    Task<IEnumerable<StudentDto>> GetMemberStudentsAsync();
    1 reference
    Task<StudentDto> GetMemberStudentAsync(string username);
    1 reference
    Task<StudentDto> GetMemberStudentByNameAsync(string name);
}

```

To use these repository methods, in the controller class constructor the interfaces should be injected so that the class can make use of their methods.

```

0 references
public StudentsController(IStudentRepository userRepository, IMapper mapper, DataContext context
    , IPhotoService photoService, IFileService fileService)
{
    _context = context;
    _fileService = fileService;
    _photoService = photoService;
    _mapper = mapper;
    _userRepository = userRepository;
}

```

```

[HttpGet("{username}", Name = "GetStudent")]
0 references
public async Task<ActionResult<StudentDto>> GetStudent(string username)
{
    var studentUsername = await _userRepository.GetMemberStudentAsync(username);
    var studentName = await _userRepository.GetMemberStudentByNameAsync(username);
    var student = (studentUsername != null) ? studentUsername : studentName;
    return Ok(student);
}

```

Conclusion:

In this application, the main features of the e-learning platform are developed such as courses, students, teachers, registering users, user information, assigning courses, announcements, lecture notes, lecture videos, homeworks, quizzes, outlines, file uploaders to upload homeworks and lecture notes, downloading them and etcetera. However, there are still many features to add and downsides to implement. As an example, the discussion between students features can be implemented or messaging features can be implemented between all type of users, meaning that, the admin can message publicly so that all the users will be aware of the message whereas the students and teachers can message each other. By the code implementations, the classes can be sorted out further and shortened to make the same job with less code by providing some more services or inheriting and injecting more interfaces. For the database side implementation increase, we can make all the user types in one table and give them different roles. For example, all users student, teacher and admin are considered as people so we can make one general table for all of them called users, but it requires more knowledge and very good understanding by database side and user roles.

Despite of the further implementations, still the application can be considered usable and beneficial with its light and user friendly interface, as well as that, secure and RESTFULL API backend.