UNIMANAGE

ASP.NET FULL STACK WEB APPLICATION

DEVELOPED BY : BENCHAFSI GIASIN

INTRODUCTION

Athens University of Economics and Business presented the need for a user management application that will perform the " CRUD " operations on a database. This application will be used by their database administrator.They previously had an offline management application.

PROBLEM ANALYSIS

- First of all we create a database to store our students,teachers etc..
- Then we connect our web app so operrations can be done.
- We need a friendly and error-proof user interface so the user can seamlessly navigate.
- The problem with offline database management applications is that you had to install the exe and also having the installer file available if you had to change a device.With the web app you dont need instalations, from any device you can hop on and start the operations you need so we cocluded that a web app is the best solution.

DESIGN-APPLICATION

- The application will be built using the .Net Core that includes the ASP.NET with Razor pages and the C# programming language.The user will interact in a menu of Razor pages that will also return feedback.For each operation i will create diferent pages and the cotrollers will handle the routing between them.

DESIGN-CODE

The pattern our code will follow in this project is called :

- SOA(service oriented architecture), it defines a way to make software components reusable and interoperable via service interfaces thus making our code maintainable,reusable,testable and scalable.

  Also known as N-tier where the tiers are the following:

- DTO is an object that transfers data between processes like collecting browser/user input data or returning data to a page.
- DAO layer is responsible for running queries in the database.
- MODEL layer is a representation of our database data.
- SERVICE layer is our business logic, PUBLIC API with all the methods and documentation the user will be calling.
- CONTROLLER layer is the layer that handles the requests from the user and communicates with the service for calling it's methods and also for responding back with the appropriate data.
- VIEW layer is the cshtml pages the user will see.

DESIGN-DATABASE

- The database will consist of four tables (as per client request).
- Teachers table(professors)
- Students table(undergraduates)
- Courses table(subjects)
- StudentsCoursess table meaning(enrollments)

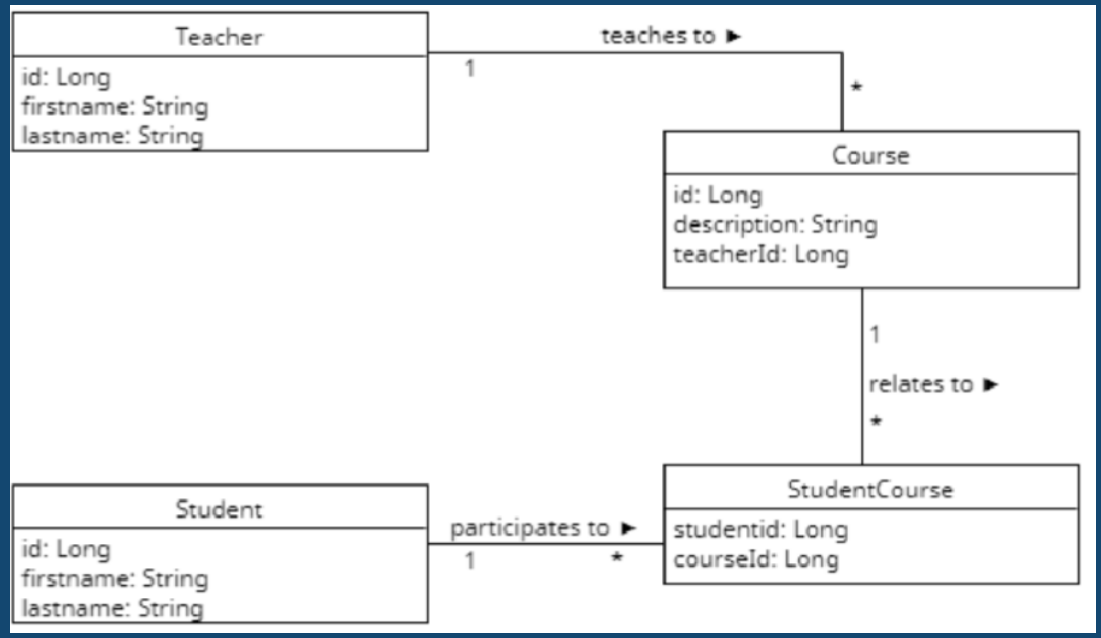
IMPLEMENTATION

LETS START BY CREATING OUR DATABASE.

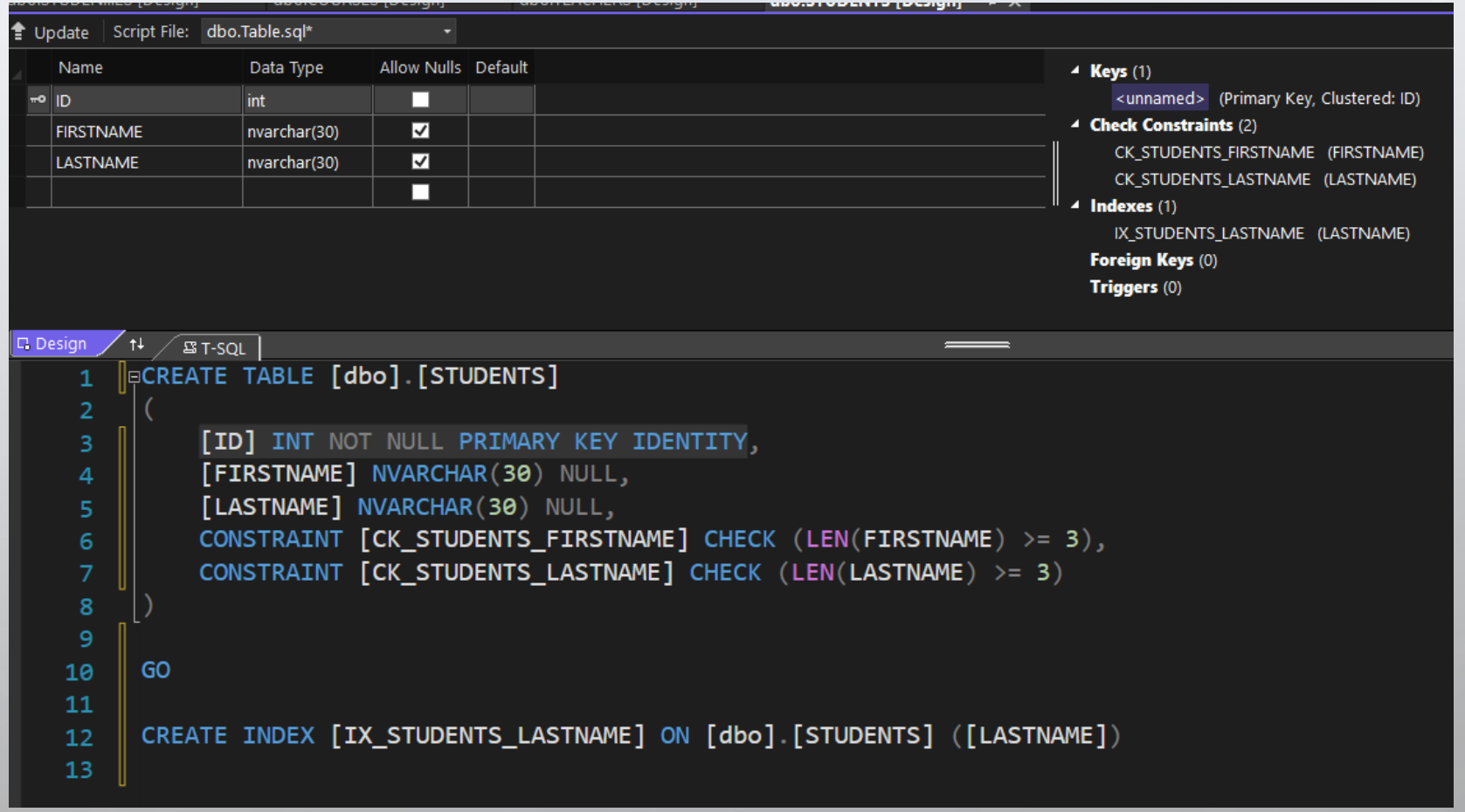
WE USE SQL EXPRESS 2019

AND FROM INSIDE THE VISUAL STUDIO ENVIRONMENT WE CREATE THE TABLES

THE DOMAIN MODEL REQUESTED IS BELOW



STUDENTS TABLE



TEACHERS TABLE

Name	Data Type	Allow Nulls	Default
ID	int	☐	
FIRSTNAME	nvarchar(30)	☑	
LASTNAME	nvarchar(30)	☑	
		☐	

- **Keys** (1)
  - <unnamed> (Primary Key, Clustered: ID)
- **Check Constraints** (2)
  - CK_TEACHERS_FIRSTNAME (FIRSTNAME)
  - CK_TEACHERS_LASTNAME (LASTNAME)
- **Indexes** (1)
  - IX_TEACHERS_LASTNAME (LASTNAME)
- **Foreign Keys** (0)
- **Triggers** (0)

Design | T-SQL

```
1  CREATE TABLE [dbo].[TEACHERS]
2  (
3      [ID] INT NOT NULL PRIMARY KEY IDENTITY,
4      [FIRSTNAME] NVARCHAR(30) NULL,
5      [LASTNAME] NVARCHAR(30) NULL,
6      CONSTRAINT [CK_TEACHERS_FIRSTNAME] CHECK (LEN(FIRSTNAME) >= 3),
7      CONSTRAINT [CK_TEACHERS_LASTNAME] CHECK (LEN(LASTNAME) >= 3)
8  )
9
10 GO
11
12 CREATE INDEX [IX_TEACHERS_LASTNAME] ON [dbo].[TEACHERS] ([LASTNAME])
13
```

COURSES TABLE

```sql
CREATE TABLE [dbo].[COURSES] (
    [ID]          INT           IDENTITY (1, 1) NOT NULL,
    [DESCRIPTION] NVARCHAR (50) NULL,
    [TEACHER_ID]  INT           NULL,
    PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_COURSES_ToTEACHERS] FOREIGN KEY ([TEACHER_ID]) REFERENCES [dbo].[TEACHERS] ([ID]) ON DELETE CASCA
    CONSTRAINT [CK_COURSES_DESCRIPTION] CHECK (len([DESCRIPTION]) >= (3))
);


 GO
CREATE NONCLUSTERED INDEX [IX_Table_DESCRIPTION]
    ON [dbo].[COURSES]([DESCRIPTION] ASC);
```

STUDENTSCOURSES TABLE

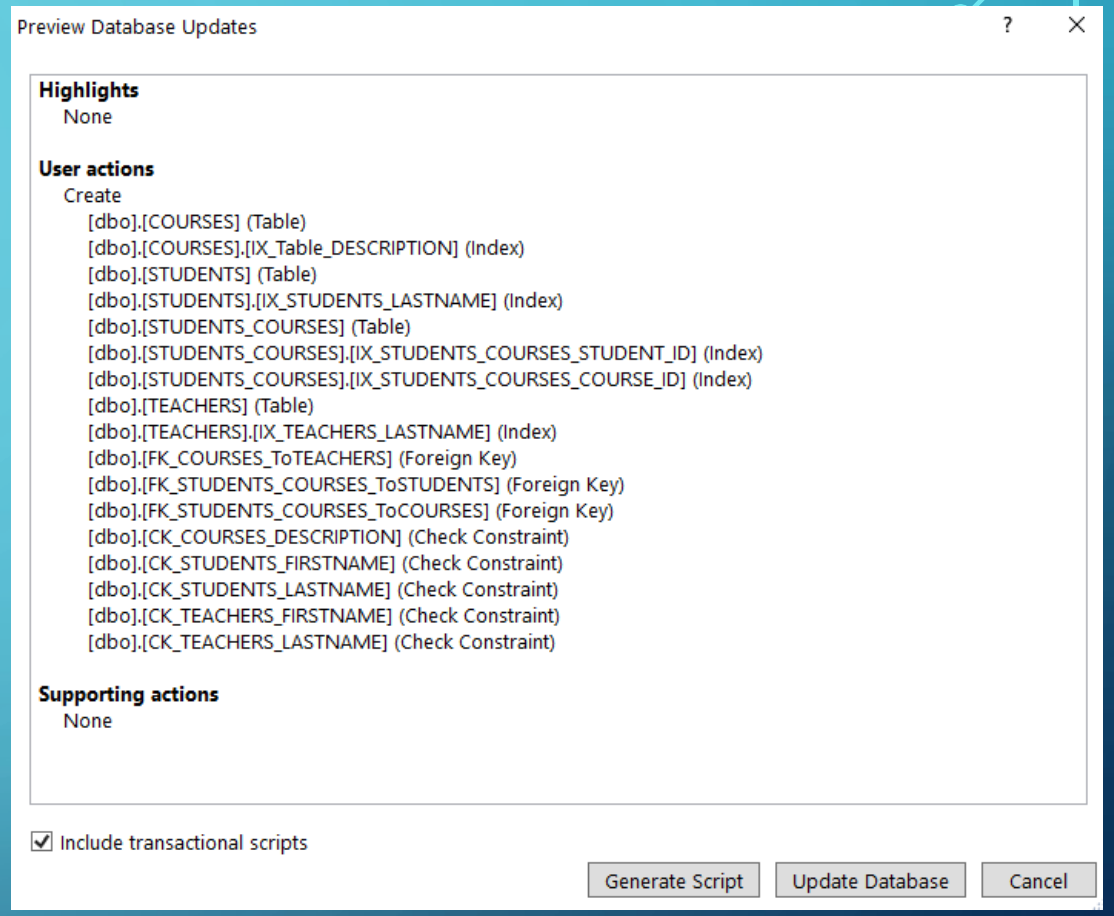
```
CREATE TABLE [dbo].[STUDENTS_COURSES] (
    [STUDENT_ID] INT NOT NULL,
    [COURSE_ID]  INT NOT NULL,
    CONSTRAINT [PK_STUDENTS_COURSES] PRIMARY KEY CLUSTERED ([COURSE_ID] ASC, [STUDENT_ID] ASC),
    CONSTRAINT [FK_STUDENTS_COURSES_ToSTUDENTS] FOREIGN KEY ([STUDENT_ID]) REFERENCES [dbo].[STUDENTS] ([ID]) ON DELETE CASCADE,
    CONSTRAINT [FK_STUDENTS_COURSES_ToCOURSES] FOREIGN KEY ([COURSE_ID]) REFERENCES [dbo].[COURSES] ([ID]) ON DELETE CASCADE
);


GO
CREATE NONCLUSTERED INDEX [IX_STUDENTS_COURSES_STUDENT_ID]
    ON [dbo].[STUDENTS_COURSES]([STUDENT_ID] ASC);


GO
CREATE NONCLUSTERED INDEX [IX_STUDENTS_COURSES_COURSE_ID]
    ON [dbo].[STUDENTS_COURSES]([COURSE_ID] ASC);

```

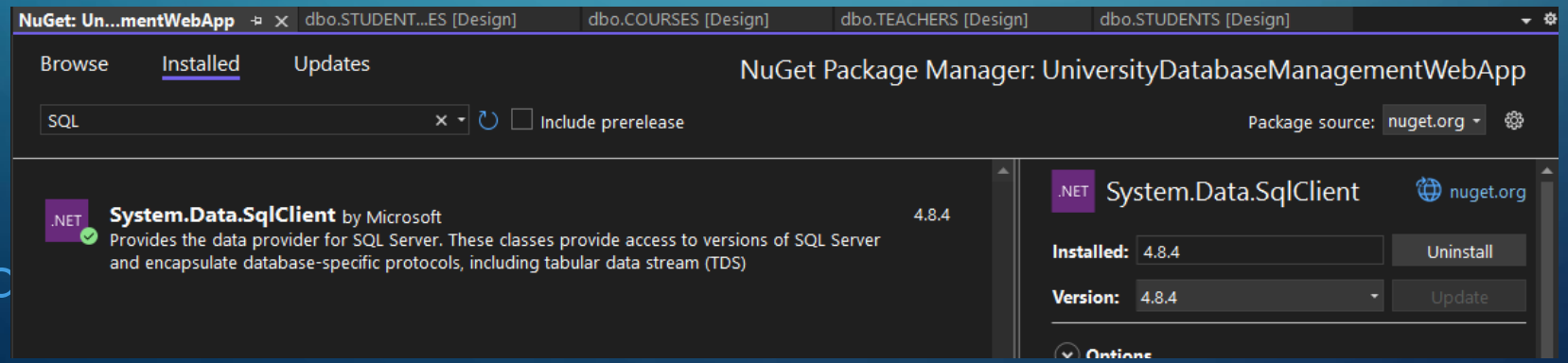
CREATING THE DATABASE

We are ready to create the database

When we hit update our university database is created.

Also nugget package sqlclient is needed



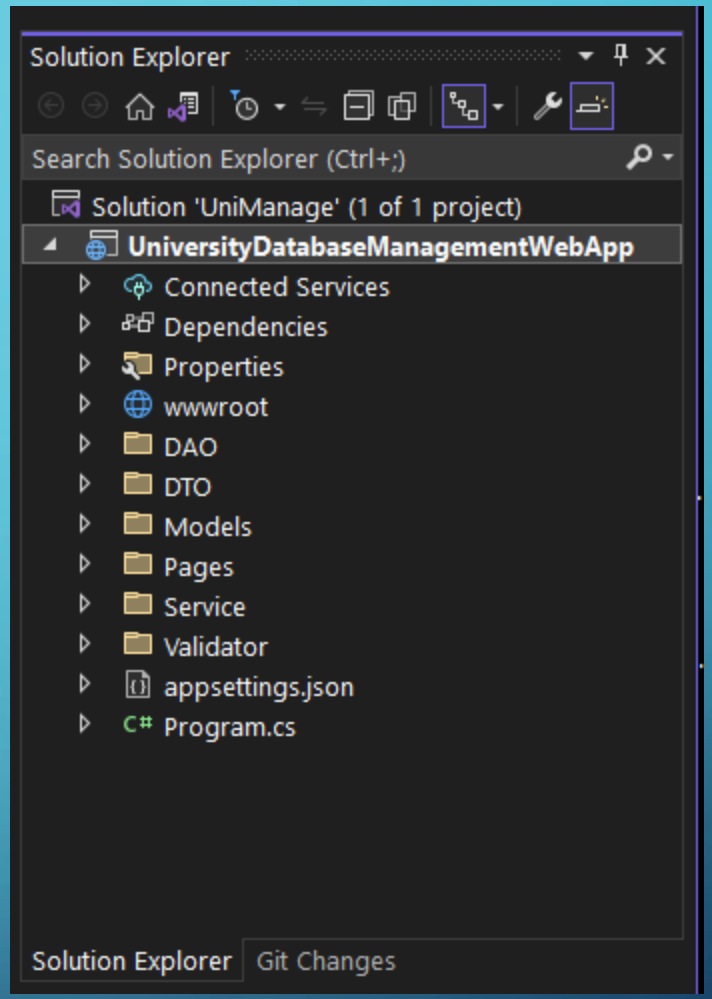


DATABASE INFO

- Students and teachers are simple database tables with index in lastname

  (for searches) and character checking constraints (>=3)
- Courses are depended on the teacher that teaches them so we have created a teacher id foreign key,if i delete a teacher i want to also delete the course for integrity.This can also be done by the database with triggers but for this project we are going to do it via cascade.
- StudentsCourses table represents the many to many relation that students and courses have so the student id and the course id merge to become its primary key.If a student or a course is deleted the StudentCourse containing one of the two foreign keys should be deleted

PROJECT STRUCTURE



- DBUtil is a utility class we use to connect our app and database using a connection string that is stored in appsettings.jason.We can also store that in a file a read from that file for security purposes.

```
public class DBHelper
{
    private static SqlConnection? conn;

    //no instances of this class should be available
    0 references
    private DBHelper() { }

    22 references
    public static SqlConnection? GetConnection()
    {
        try
        {
            ConfigurationManager configurationManager = new();
            configurationManager.AddJsonFile("appsettings.json");
            string url = configurationManager.GetConnectionString("DefaultConnection")
            conn = new SqlConnection(url);
            return conn;
        }
        catch (Exception e)
        {
            Console.WriteLine(e.StackTrace);
            return null;
        }
    }

    0 references
    public static void CloseConnection()
    {
        if (conn is not null)
        {
            conn.Close();
        }
```

MODEL

First comes the model layer thats going to represent our database

```
namespace UniversityDatabaseManagementWebApp.Models
{
    45 references
    public class Student
    {
        13 references
        public int Id { get; set; }
        10 references
        public string? Firstname { get; set; }
        10 references
        public string? Lastname { get; set; }
    }
}
```

```
namespace UniversityDatabaseManagementWebApp.Models
{
    48 references
    public class Teacher
    {
        14 references
        public int Id { get; set; }
        11 references
        public string? Firstname { get; set; }
        11 references
        public string? Lastname { get; set; }
    }
}
```

```
namespace UniversityDatabaseManagementWebApp.Models
{
    22 references
    public class StudentCourse
    {
        7 references
        public int StudentId { get; set; }
        7 references
        public int CourseId { get; set; }
    }
}
```

```
namespace UniversityDatabaseManagementWebApp.Models
{
    45 references
    public class Course
    {
        13 references
        public int Id { get; set; }
        10 references
        public string? Description { get; set; }
        8 references
        public int TeacherId { get; set; }
    }
}
```

DAO

- On this layer for each model(or table) we have an inteface and a class that implements this interface running queries on the database

```
using UniversityDatabaseManagementWebApp.Models;

namespace UniversityDatabaseManagementWebApp.DAO
{
    7 references
    public interface ITeacherDAO
    {
        2 references
        void Insert(Teacher? teacher);
        2 references
        void Update(Teacher? teacher);
        2 references
        Teacher? Delete(Teacher? teacher);
        2 references
        Teacher? GetTeacher(int id);
        2 references
        List<Teacher> GetAll();
    }
}
```

```
using System.Data.SqlClient;
using UniversityDatabaseManagementWebApp.DAO.DBUtil;
using UniversityDatabaseManagementWebApp.Models;

namespace UniversityDatabaseManagementWebApp.DAO
{
    4 references
    public class TeacherDAOImpl : ITeacherDAO
    {
        2 references
        public void Insert(Teacher? teacher)...

        2 references
        public void Update(Teacher? teacher)...

        2 references
        public Teacher? Delete(Teacher? teacher)...

        2 references
        public Teacher? GetTeacher(int id)...

        2 references
        public List<Teacher> GetAll()...

        1 reference
        private void DeleteCourseAndStudentCourse(int id)...
    }
}
```

- Insert method

```
public void Insert(Teacher? teacher)
{
    if (teacher is null) return;

    try
    {
        using SqlConnection? conn = DBHelper.GetConnection();

        if (conn is not null)
        {
            conn.Open();

        }
        else { return; }

        string sql = "INSERT INTO TEACHERS " +
            "(FIRSTNAME, LASTNAME) VALUES " +
            "(@firstname, @lastname)";

        using SqlCommand command = new SqlCommand(sql, conn);

        command.Parameters.AddWithValue("@firstname", teacher.Firstname);
        command.Parameters.AddWithValue("@lastname", teacher.Lastname);

        command.ExecuteNonQuery();

    }
    catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
        throw;
    }
}
```

- Update method

```
public void Update(Teacher? teacher)
{
    if (teacher is null) return;

    try
    {
        using SqlConnection? conn = DBHelper.GetConnection();

        if (conn is not null)
        {
            conn.Open();

        }
        else { return; }

        string sql = "UPDATE TEACHERS SET FIRSTNAME = @firstname, " +
                     "LASTNAME = @lastname WHERE ID = @id";

        using SqlCommand command = new SqlCommand(sql, conn);

        command.Parameters.AddWithValue("@firstname", teacher.Firstname);
        command.Parameters.AddWithValue("@lastname", teacher.Lastname);
        command.Parameters.AddWithValue("@id", teacher.Id);

        command.ExecuteNonQuery();

    }
    catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
        throw;
```

Delete method in teachers is tricky because we also delete the course that this teacher was teaching and any studentscourses with the course id of the course deleted.This happens in the database.

```
2 references
public Teacher? Delete(Teacher? teacher)
{
    if (teacher is null) return null;

    try
    {
        using SqlConnection? conn = DBHelper.GetConnection();

        if (conn is not null)
        {
            conn.Open();

        }
        else { return null; }

        string sql = "DELETE FROM TEACHERS WHERE ID = @id";

        using SqlCommand command = new SqlCommand(sql, conn);

        command.Parameters.AddWithValue("@id", teacher.Id);

        int rowsAffected = command.ExecuteNonQuery();

        return (rowsAffected > 0) ? teacher : null;

    }
    catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
        throw;
    }
}
```

- GetTeacher and GetAll

```
public Teacher? GetTeacher(int id)
{
    Teacher? teacher = null;

    try
    {
        using SqlConnection? conn = DBHelper.GetConnection();

        if (conn is not null)
        {
            conn.Open();
        }

        string sql = "SELECT * FROM TEACHERS WHERE ID = @id";

        using SqlCommand command = new SqlCommand(sql, conn);

        command.Parameters.AddWithValue("@id", id);

        using SqlDataReader reader = command.ExecuteReader();

        if (reader.Read())
        {
            teacher = new Teacher()
            {
                Id = reader.GetInt32(0),
                Firstname = reader.GetString(1),
                Lastname = reader.GetString(2)
            };
        }

        return teacher;
    }
```

```
public List<Teacher> GetAll()
{
    List<Teacher> teachers = new();
    try
    {
        using SqlConnection? conn = DBHelper.GetConnection();

        if (conn is not null)
        {
            conn.Open();
        }

        string sql = "SELECT * FROM TEACHERS";

        using SqlCommand command = new SqlCommand(sql, conn);
        using SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Teacher teacher = new Teacher()
            {
                Id = reader.GetInt32(0),
                Firstname = reader.GetString(1),
                Lastname = reader.GetString(2)
            };

            teachers.Add(teacher);
        }

        return teachers;
```

SERVICE LAYER

- This layer is our public api so we must provide documentation. There is an interface and an implementation for each of our models,essentialy the service layer is going to implement its methods by calling the dao via

Dependecy injection

```
public class TeacherServiceImpl : ITeacherService
{
    private readonly ITeacherDAO dao;

    4 references
    public TeacherServiceImpl(ITeacherDAO dao)
    {
        this.dao = dao;
    }
```

- ITeacherService interface with documentation

```
namespace UniversityDatabaseManagementWebApp.Service
{
    5 references
    public interface ITeacherService
    {

        /// <summary>
        /// Inserts a teacher record into the database
        /// with parameters taken from the user
        /// </summary>
        /// <param name="dto">The user input that is converted
        /// from TeacherDTO to type Teacher.Then it is inserted into
        /// the TEACHERS table in our database </param>
        2 references
        void InsertTeacher(TeacherDTO? dto);

        /// <summary>
        /// Updates a teacher record in the database
        /// with parameters taken from the user
        /// </summary>
        /// <param name="dto">The user input that is converted
        /// from TeacherDTO to type Teacher.Then the teacher's
        /// first name and last name get replaced
        /// by the new values in TEACHERS table in our database</param>
        2 references
        void UpdateTeacher(TeacherDTO? dto);

        /// <summary>
        /// Deletes a teacher record if it matches the user input or returns
        /// null if the record wasn't found
        /// </summary>
        /// <param name="dto">The user input that is converted
        /// from TeacherDTO to type Teacher.</param>
        /// <returns>Returns the teacher that was deleted or null
        /// if the specified teacher was not found</returns>
        2 references
        Teacher? DeleteTeacher(TeacherDTO? dto);

        /// <summary>
        /// Returns a teacher based on id
        /// </summary>
        /// <param name="id">The teachers's id</param>
        /// <returns>A Teacher object</returns>
        2 references
        Teacher? GetTeacher(int id);

        /// <summary>
        /// Returns a list of all teachers currently
        /// in our database.
        /// </summary>
        /// <returns>A list of Teacher objects</returns>
        2 references
        List<Teacher> GetAllTeachers();
    }
```

- All the methods of the teacher service implementation

```
public class TeacherServiceImpl : ITeacherService
{
    private readonly ITeacherDAO dao;

    4 references
    public TeacherServiceImpl(ITeacherDAO dao)
    {
        this.dao = dao;
    }
```

```
    public void InsertTeacher(TeacherDTO? dto)
    {
        if (dto is null) return;

        try
        {
            Teacher? teacher = Convert(dto);
            dao.Insert(teacher);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            throw;
        }
    }

    2 references
    public void UpdateTeacher(TeacherDTO? dto)
    {
        if (dto is null) return;

        try
        {
            Teacher? teacher = Convert(dto);
            dao.Update(teacher);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            throw;
        }
    }
```

```
    2 references
    public Teacher? DeleteTeacher(TeacherDTO? dto)
    {
        if (dto is null) return null;

        try
        {
            Teacher? teacher = Convert(dto);
            return dao.Delete(teacher);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            throw;
        }
    }

    2 references
    public Teacher? GetTeacher(int id)
    {
        try
        {
            return dao.GetTeacher(id);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            throw;
        }
    }
```

```
    2 references
    public List<Teacher> GetAllTeachers()
    {
        try
        {
            return dao.GetAll();

        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            return new List<Teacher>();
        }
    }

    3 references
    private Teacher? Convert(TeacherDTO dto)
    {
        if (dto is null) return null;

        return new Teacher()
        {
            Id = dto.Id,
            Firstname = dto.Firstname,
            Lastname = dto.Lastname
        };
    }
```

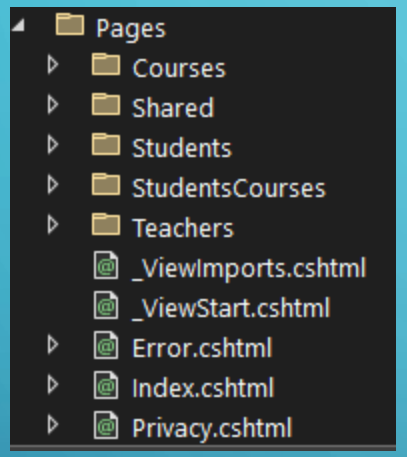
VALIDATOR

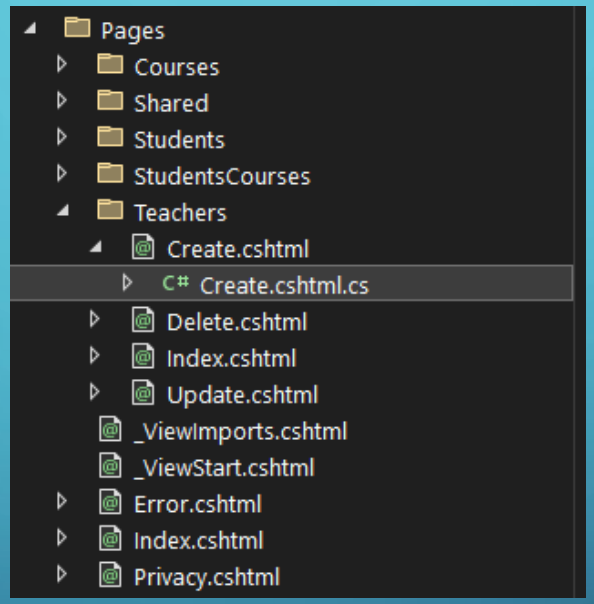
```
namespace UniversityDatabaseManagementWebApp.Validator
{
    3 references
    public class TeacherValidator
    {
        /// <summary>
        /// No instances of this class should be available
        /// </summary>
        0 references
        private TeacherValidator() { }

        2 references
        public static string Validate(TeacherDTO? dto)
        {
            if (dto is not null && dto.Firstname is not null && dto.Lastname is not null)
            {
                if ((dto.Firstname.Length < 3) || (dto.Lastname.Length < 3))
                {
                    return "Firstname or Lastname should not be less than three characters";
                }
                else
                {
                    return "";
                }


            }
            else
            {
                return "";
            }
        }
    }
}
```

CONTROLLER AND VIEW LAYERS

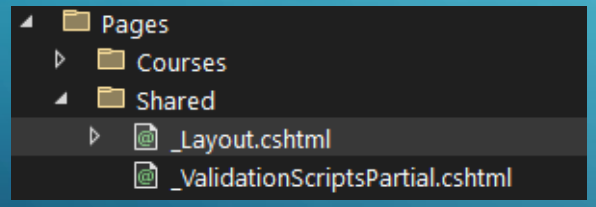
On our pages folder we have the views(.cshtml) and their cotrollers(.cs)

1-1 relationship





- We are going to have a layout page.For each operation we are just swapping the body of our layout.
- @RenderBody()




```
<!DOCTYPE html>
<html lang="en">
<head>...
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand text-decoration-underline" asp-area="" asp-page="/Index"><strong>UniManage</strong></a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls=
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Students/Index">Undergraduates</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Teachers/Index">Professors</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Courses/Index">Subjects</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/StudentsCourses/Index">Enrollments</a>
                        </li>

                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container-fluid bg-dark">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">...
```

Teacher index controller

We use Dependency injection so we can call servive layer methods.

```
namespace UniversityDatabaseManagementWebApp.Pages.Teachers
{
    6 references
    public class IndexModel : PageModel
    {
        private readonly ITeacherDAO teacherDAO = new TeacherDAOImpl();
        private readonly ITeacherService? service;

        internal List<Teacher> teachers = new();
        0 references
        public IndexModel()
        {
            service = new TeacherServiceImpl(teacherDAO);
        }
        0 references
        public void OnGet()
        {
            try
            {
                teachers = service!.GetAllTeachers();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                throw;
            }
        }
    }
}
```

Teacher index view

We take the data from the controller and displays it to the user.

```
@page
@model UniversityDatabaseManagementWebApp.Pages.Teachers.IndexModel
@{
}

<h2 class="text-center text-decoration-underline text-light">PROFESSORS</h2>

<a class="btn btn-outline-warning fw-bold btn-sm"
href="/Teachers/Create">Add Professor</a>


<table class="table table-dark table-bordered table-hover">
    <thead>
        <tr>
            <th>ID</th>
            <th>Firstname</th>
            <th>Lastname</th>
        </tr>
    </thead>
    <tbdody>
        @if (Model.teachers is not null)
        {
            @foreach (var teacher in Model.teachers)
            {
                <tr>
                    <td>@teacher.Id</td>
                    <td>@teacher.Firstname</td>
                    <td>@teacher.Lastname</td>
                    <td>
                        <a class="btn btn-outline-info fw-bold btn-sn" href="/Teachers/Update?id=@teacher.Id">Update</a>

                        <a class="btn btn-outline-danger fw-bold btn-sn" onclick="return confirm('
                        This action will delete the specific and all related entries. Are you sure you want to proceed?')
                        " href="/Teachers/Delete?id=@teacher.Id">Delete</a>
                    </td>
                </tr>


            }
        }
    </tbdody>
</table>
```

Create Teacher controller

We take the data from the form(user input).

We validate them and we pass it to the servise method hat calls the dao and a query with a new teacher is given to our database .

We also redirect the user to the teacher index.

```
6 references
public class CreateModel : PageModel
{
    private readonly ITeacherDAO teacherDAO = new TeacherDAOImpl();
    private readonly ITeacherService? service;

    internal List<Teacher> teachers = new();
    0 references
    public CreateModel()
    {
        service = new TeacherServiceImpl(teacherDAO);
    }

    internal TeacherDTO teacherDTO = new();
    public string errorMessage = "";

    0 references
    public void OnGet()
    {

    }

    0 references
    public void OnPost()
    {
        errorMessage = "";

        teacherDTO.Firstname = Request.Form["firstname"];
        teacherDTO.Lastname = Request.Form["lastname"];

        errorMessage = TeacherValidator.Validate(teacherDTO);

        if (!errorMessage.Equals("")) return;

        try
        {
            service!.InsertTeacher(teacherDTO);
            Response.Redirect("/Teachers/Index");

        }
        catch (Exception e)
        {
            errorMessage = e.Message;
            return;
        }

    }
}
```

- Create teacher view

```
@page
@model UniversityDatabaseManagementWebApp.Pages.Teachers.CreateModel
@{
}

<h2 class="text-light text-center mb-3">Add Professor</h2>

@if (!Model.errorMessage.Equals(""))
{
    <h6><strong>@Model.errorMessage</strong></h6>
}

<form class="container" method="POST">
    <div class="row mb-3">
        <label for="firstname" class="col-md-1 col-form-label">Firstname</label>
        <div>
            <input type="text" class="form-control" name="firstname" id="firstname"
            placeholder="Enter Firstname" value="@Model.teacherDTO.Firstname" />
        </div>
    </div>
    <div class="row mb-3">
        <label for="lastname" class="col-md-1 col-form-label">Lastname</label>
        <div>
            <input type="text" class="form-control" name="lastname" id="lastname"
            placeholder="Enter Lastname" value="@Model.teacherDTO.Lastname" />
        </div>
    </div>
    <div class="row mb-3">
        <div class="offset-md-1 col-md-3 d-grid">
            <button type="submit" class="btn btn-success">Submit</button>
        </div>

        <div class="offset-md-1 col-md-3 d-grid">
            <a href="/Teachers/Index" role="button" class="btn btn-primary">Cancel</a>
        </div>
    </div>

</form>
```

- Teacher delete cotroller

```
6 references
public class DeleteModel : PageModel
{
    private readonly ITeacherDAO teacherDAO = new TeacherDAOImpl();
    private readonly ITeacherService? service;

    internal List<Teacher> teachers = new();
    0 references
    public DeleteModel()
    {
        service = new TeacherServiceImpl(teacherDAO);
    }

    internal TeacherDTO teacherDTO = new();
    public string errorMessage = "";

    0 references
    public void OnGet()
    {

        try
        {
            Teacher? teacher;

            int id = int.Parse(Request.Query["id"]);
            teacherDTO.Id = id;

            teacher = service!.DeleteTeacher(teacherDTO);
            Response.Redirect("/Teachers/Index");
        }
        catch (Exception e)
        {
            errorMessage = e.Message;
            return;
        }
    }
}
```

• Teacher update controller

```
namespace UniversityDatabaseManagementWebApp.Pages.Teachers
{
    6 references
    public class UpdateModel : PageModel
    {
        private readonly ITeacherDAO teacherDAO = new TeacherDAOImpl();
        private readonly ITeacherService? service;

        internal List<Teacher> teachers = new();
        0 references
        public UpdateModel()
        {
            service = new TeacherServiceImpl(teacherDAO);
        }

        internal TeacherDTO teacherDTO = new();
        public string errorMessage = "";

        0 references
        public void OnGet()
        {
            errorMessage = "";

            try
            {
                Teacher? teacher;

                int id = int.Parse(Request.Query["id"]);
                teacher = service!.GetTeacher(id);
                if (teacher != null)
                {
                    teacherDTO = ConvertToDTO(teacher);
                }
            }
            catch (Exception e)
            {
                errorMessage = e.Message;
                return;
            }
        }
```

```
    0 references
    public void OnPost()
    {
        errorMessage = "";

        teacherDTO.Id = int.Parse(Request.Form["id"]);
        teacherDTO.Firstname = Request.Form["firstname"];
        teacherDTO.Lastname = Request.Form["lastname"];

        errorMessage = TeacherValidator.Validate(teacherDTO);

        if (!errorMessage.Equals("")) return;

        try
        {
            service!.UpdateTeacher(teacherDTO);
            Response.Redirect("/Teachers/Index");
        }
        catch (Exception e)
        {
            errorMessage = e.Message;
            return;
        }
    }

    1 reference
    private TeacherDTO ConvertToDTO(Teacher dto)
    {
        return new TeacherDTO()
        {
            Id = dto.Id,
            Firstname = dto.Firstname!.Trim(),
            Lastname = dto.Lastname!.Trim()
        };
    }
```

• Update Teacher View

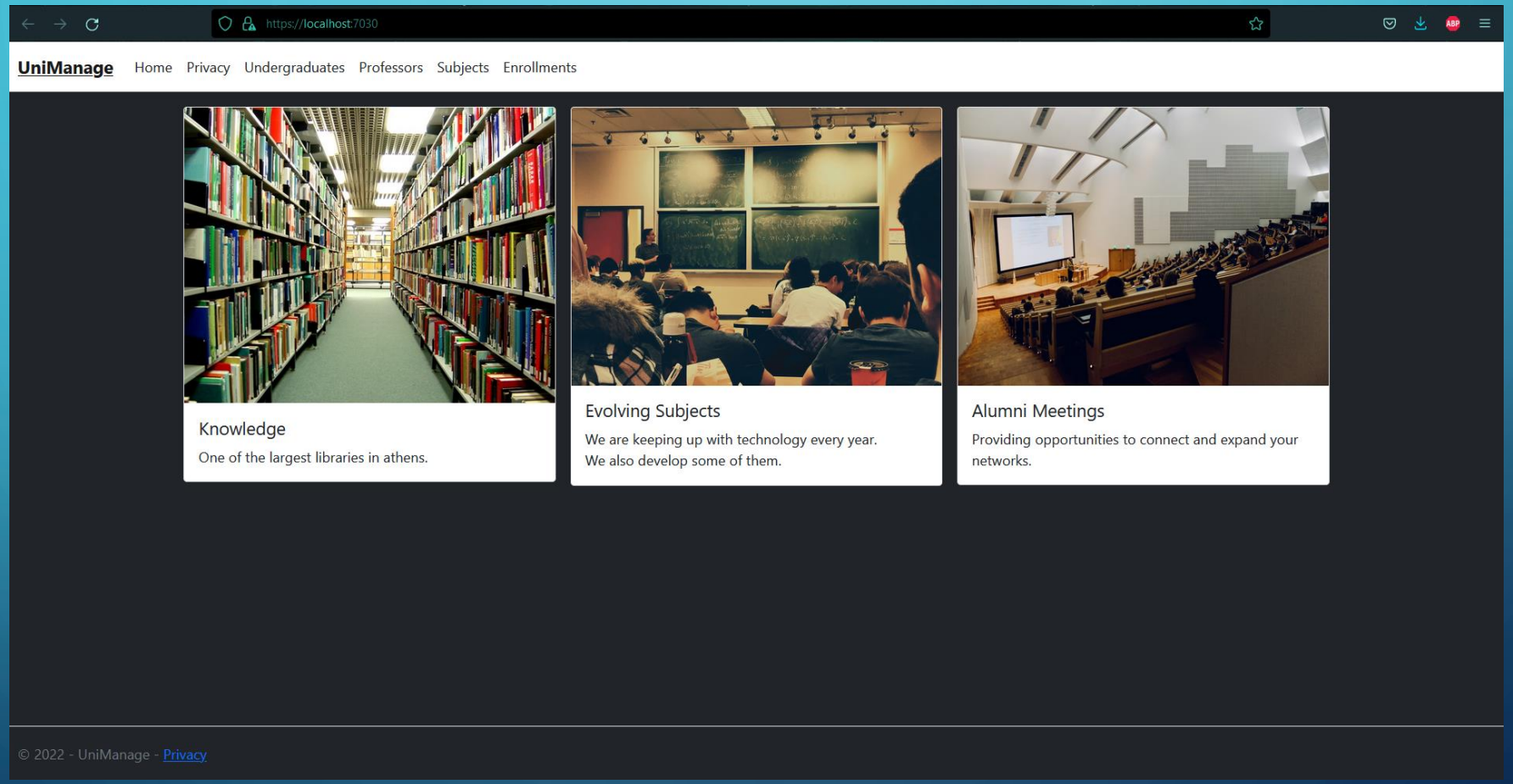
```
<h2 class="mb-3 text-center text-light">Update Professor</h2>
@if (!Model.errorMessage.Equals(""))
{
    <h6><strong>@Model.errorMessage</strong></h6>
}

<form class="container" method="POST">
    <input type="hidden" name="id" value="@Model.teacherDTO.Id" />
    <div class="row mb-3">
        <label for="firstname" class="col-md-1 col-form-label">Firstname</label>
        <div>
            <input type="text" class="form-control" name="firstname" id="firstname" placeholder="Enter Firstname" value="@Model.teacher
        </div>
    </div>
    <div class="row mb-3">
        <label for="lastname" class="col-md-1 col-form-label">Lastname</label>
        <div>
            <input type="text" class="form-control" name="lastname" id="lastname" placeholder="Enter Lastname" value="@Model.teacherDTO
        </div>
    </div>
    <div class="row mb-3">
        <div class="offset-md-1 col-md-3 d-grid">
            <button type="submit" class="btn btn-success">Submit</button>
        </div>
        <div class="offset-md-1 col-md-3 d-grid">
            <a href="/Teachers/Index" role="button" class="btn btn-primary">Cancel</a>
        </div>
    </div>

</form>
```

We have teachers to be able to teach a course,their id is taken and put together with a course that this teacher will be responsible for teaching.

But our administrator might forget the id so when an id would need to be placed in order to create a new course we create HTML drop down list with all the available teachers for convinience in the course's view.

```
<form class="container" method="POST">
    <div class="row mb-3">
        <label for="description" class="col-md-1 col-form-label">Description</label>
        <div>
            <input type="text" class="form-control" name="description" id="description" placeholder
        </div>
    </div>
    <div class="row mb-3">
        <label for="teacherid" class="col-md-1 col-form-label">Professor:</label>
        <div>

            @{
                List<SelectListItem> listItems = new List<SelectListItem>();
                @foreach (var teacher in Model.teachers)
                {
                    listItems.Add(new SelectListItem
                {
                    Text = @teacher.Firstname + " " + teacher.Lastname,
                    Value = @teacher.Id.ToString()
                });

                }
            }

            @Html.DropDownList("teacherid", listItems, new { @class = "form-control" })

        </div>
    </div>
    <div class="row mb-3">
        <div class="offset-md-1 col-md-3 d-grid">
            <button type="submit" class="btn btn-success">Submit</button>
        </div>

        <div class="offset-md-1 col-md-3 d-grid">
            <a href="/Courses/Index" role="button" class="btn btn-primary">Cancel</a>
        </div>
    </div>
```

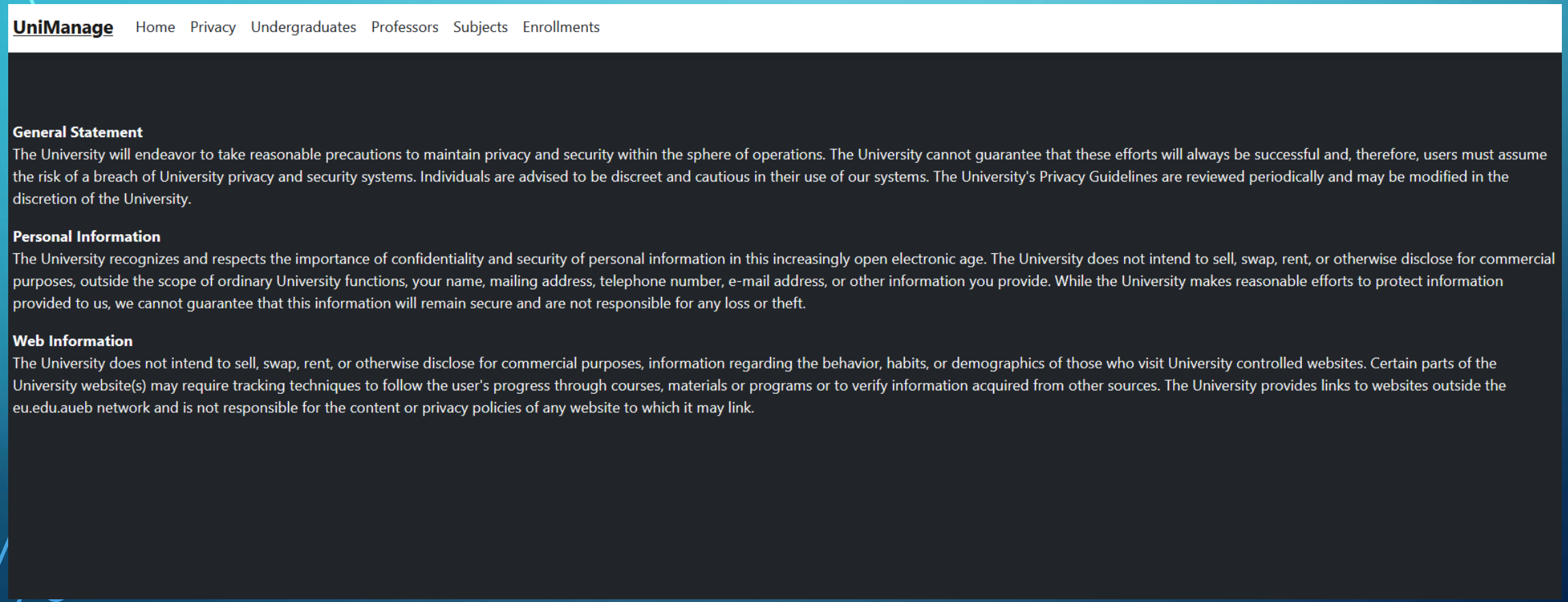
SHOWCASE-HOMEPAGE



HOMEPAGE-CODE

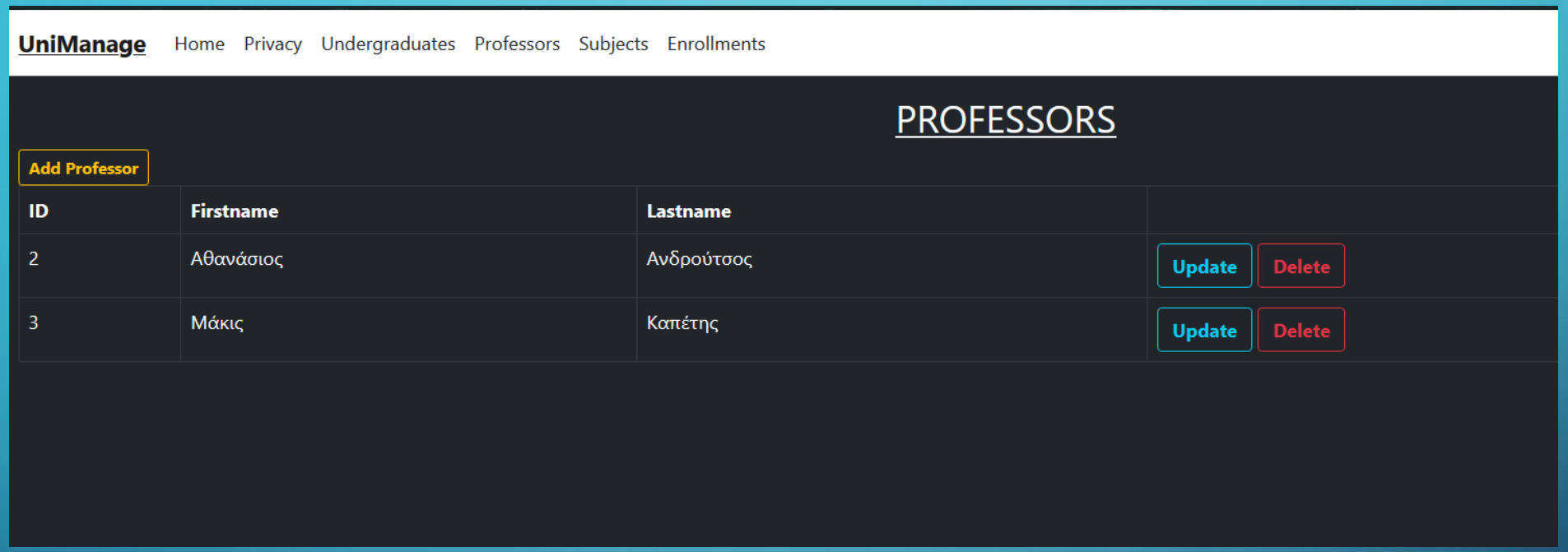
```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}


<div class="container">
    <div class="row g-3">
        <div class="col-12 col-md-6 col-lg-4">
            <div class="card">
                <img src="~/indexhomeimgs/uni1.jpg" alt="University Library" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Knowledge</h5>
                    <p class="card-text">
                        One of the largest libraries in athens.
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-6 col-lg-4">
            <div class="card">
                <img src="~/indexhomeimgs/uni2.jpg" alt="University Subject" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Evolving Subjects</h5>
                    <p class="card-text">
                        We are keeping up with technology every year.<br />
                        We also develop some of them.

                    </p>
                </div>
            </div>
        </div>
        <div class="col-12 col-md-6 col-lg-4">
            <div class="card">
                <img src="~/indexhomeimgs/uni3.jpg" alt="University Meeting" class="card-img-top">
                <div class="card-body">
                    <h5 class="card-title">Alumni Meetings</h5>
                    <p class="card-text">
                        Providing opportunities to connect and expand your networks.
                    </p>
                </div>
            </div>
```

PRIVACY POLICY



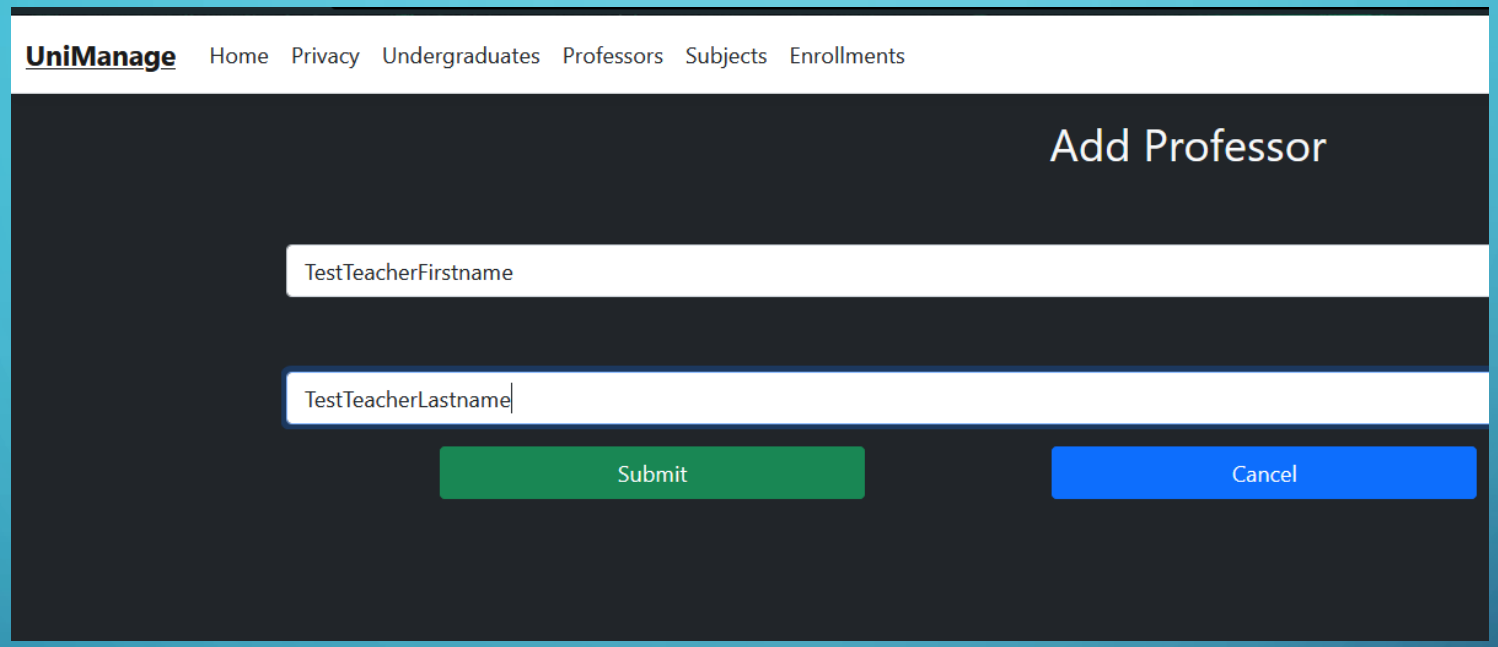
- Adding a teacher showcase

We press the Add Professor button

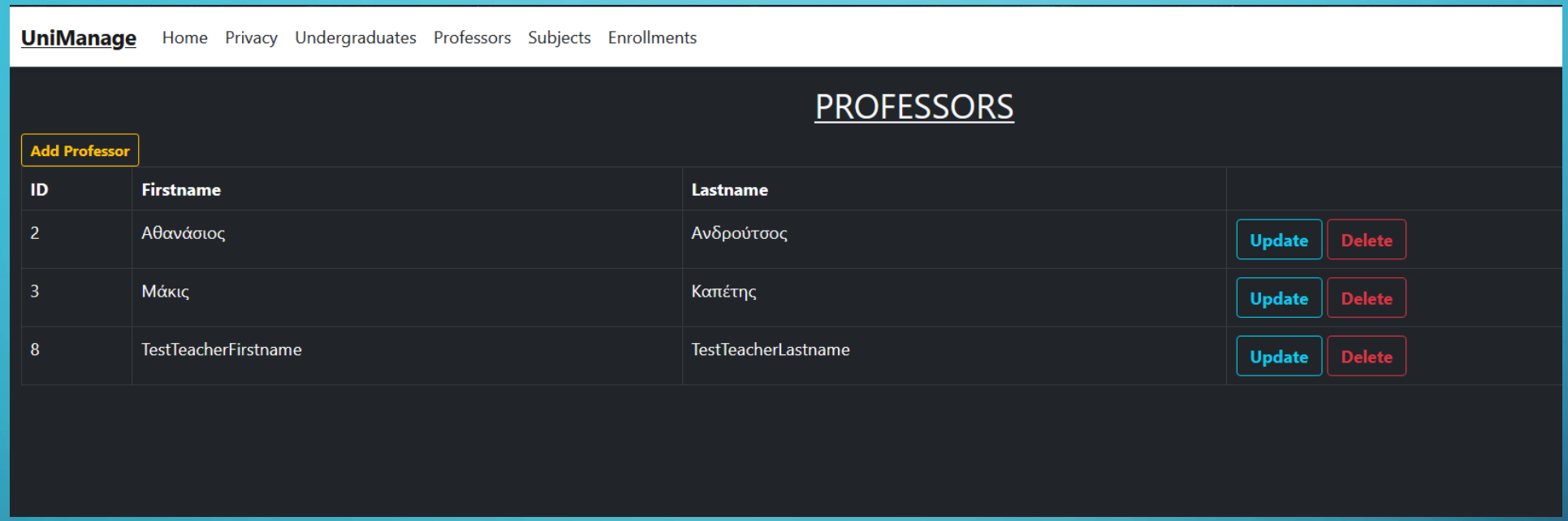


- Adding a teacher showcase

We provide our desired data and hit the submit button



- The teacher is created!



Teacher delete is trivial

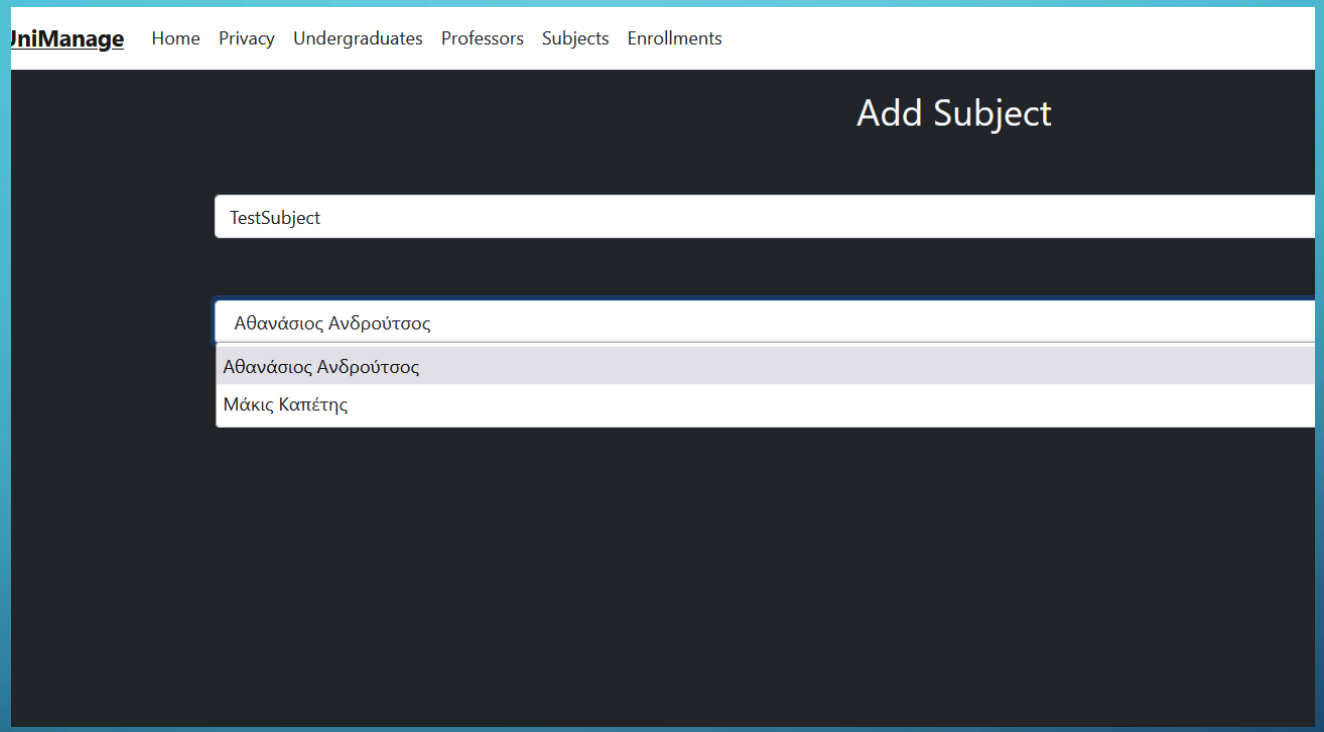
We just hit the delete button and the teacher is gone.

Updating teacher is similar to creating one.

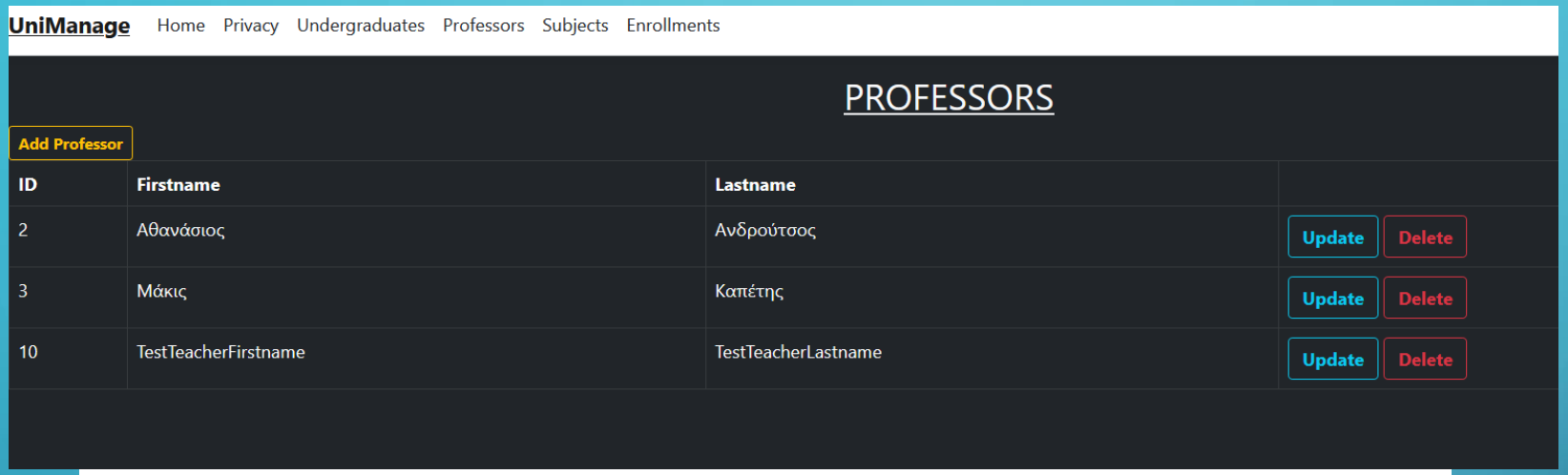
Lets create a new course

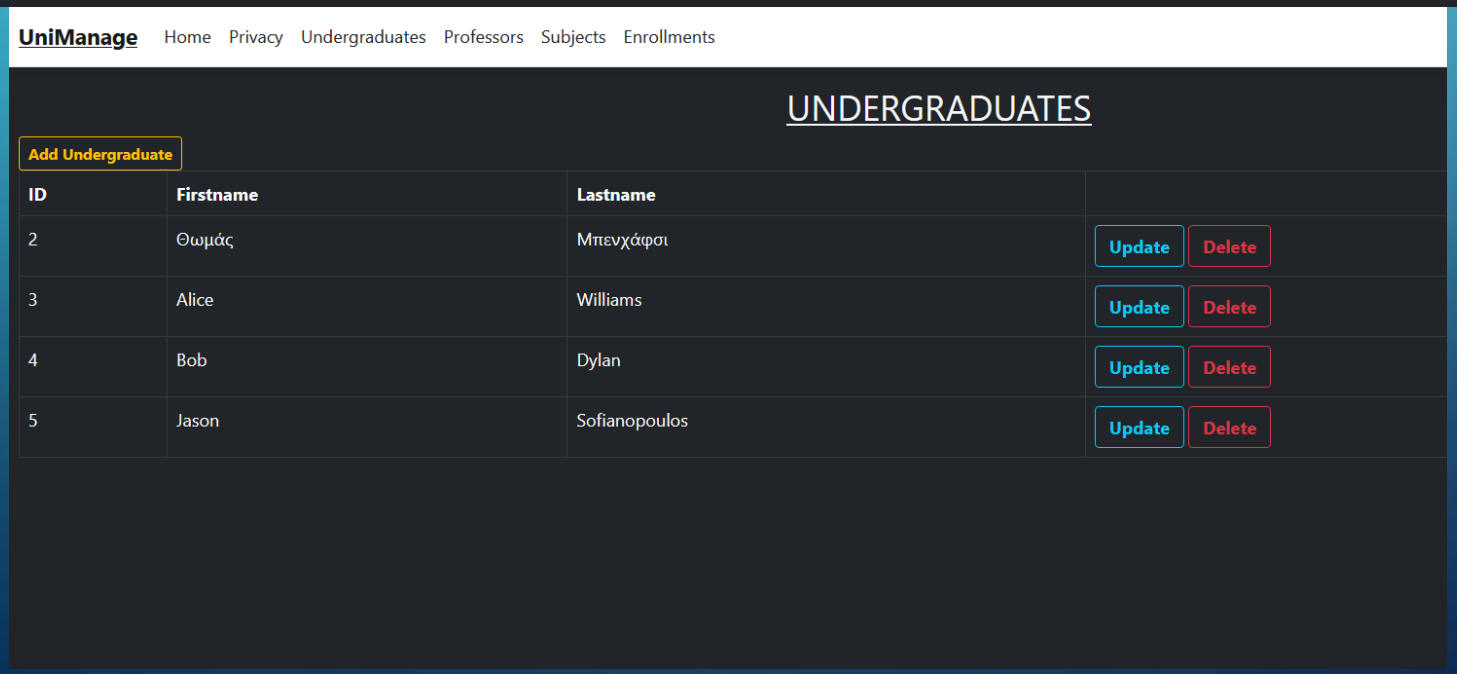
A drop down list shows us all available teachers to choose from

we make our choice and hit submit

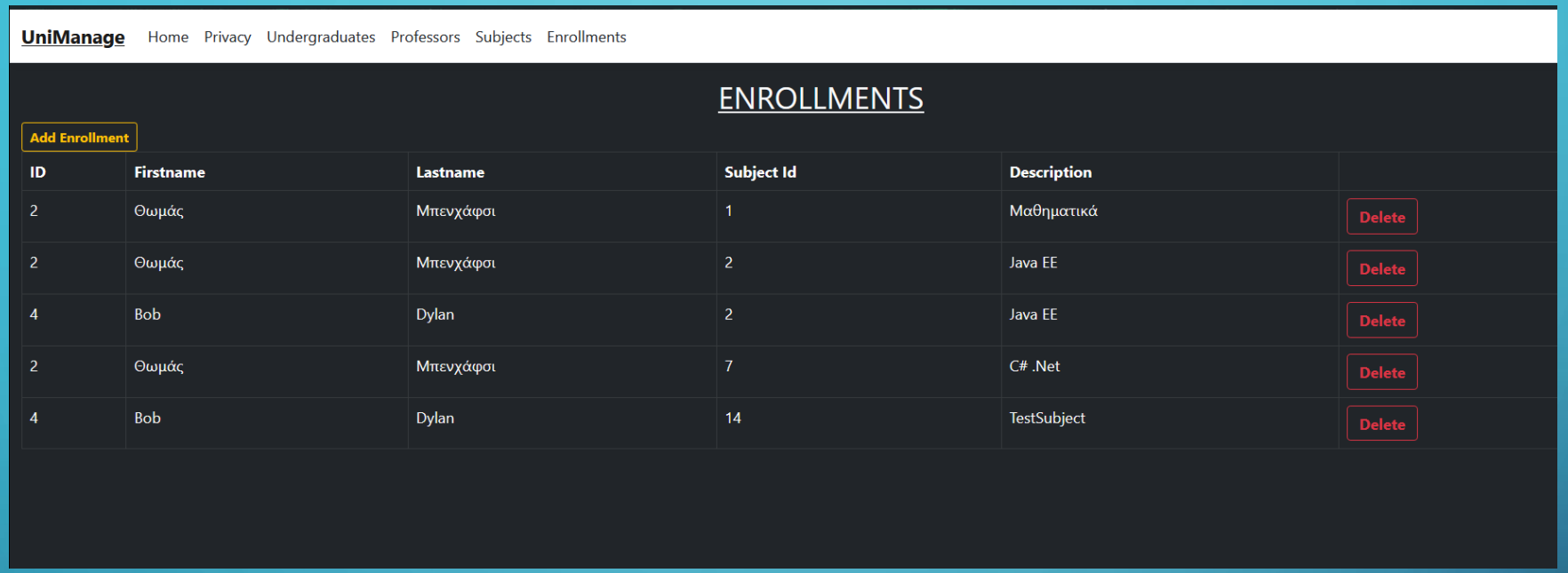


Lets create a test teacher give him a test course and a test student course and delete him.Bob dylan will be the student

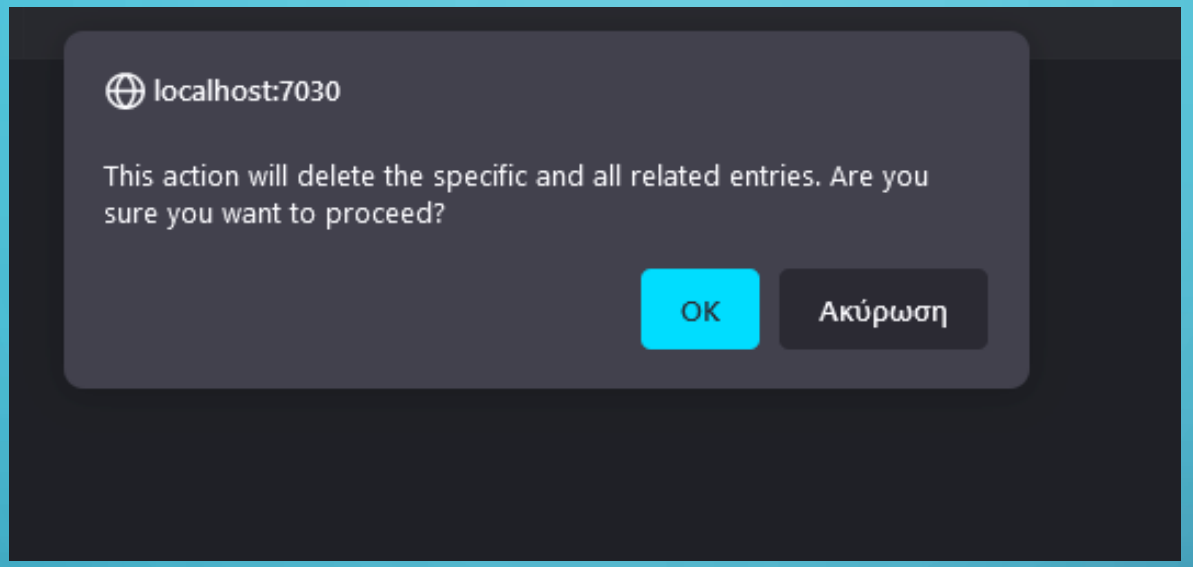




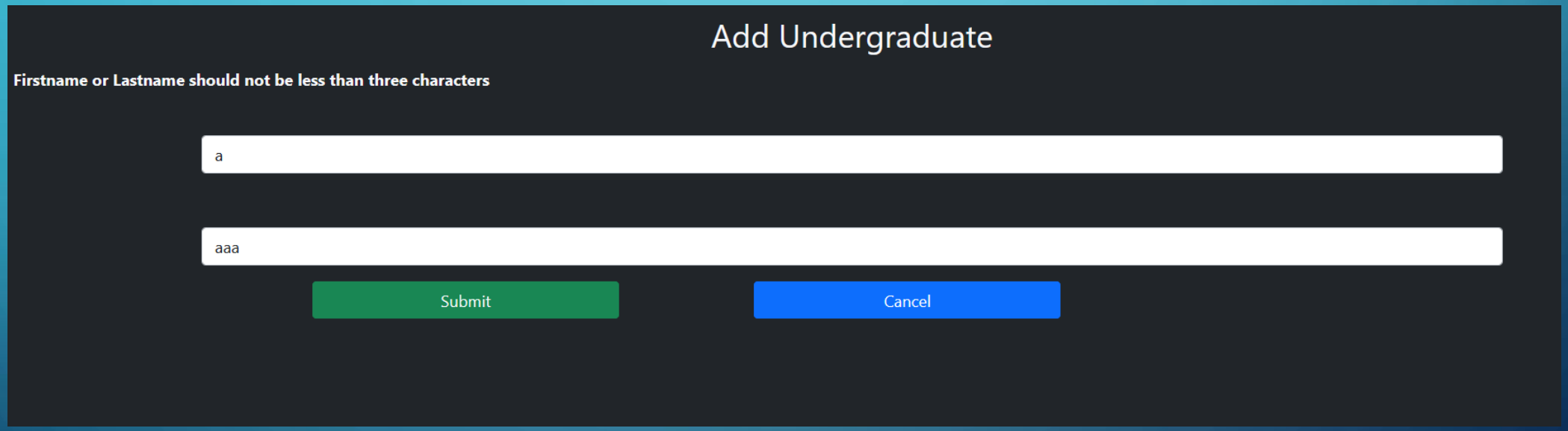
- Bob dylan enrolls test subject



- If we now hit the delete button the test teacher the course he teaches and all the studentscourses are gone so we warn our user



ERROR MESSAGE



THANKS FOR READING

- Bibliography

Athanasios Androutsos OPA