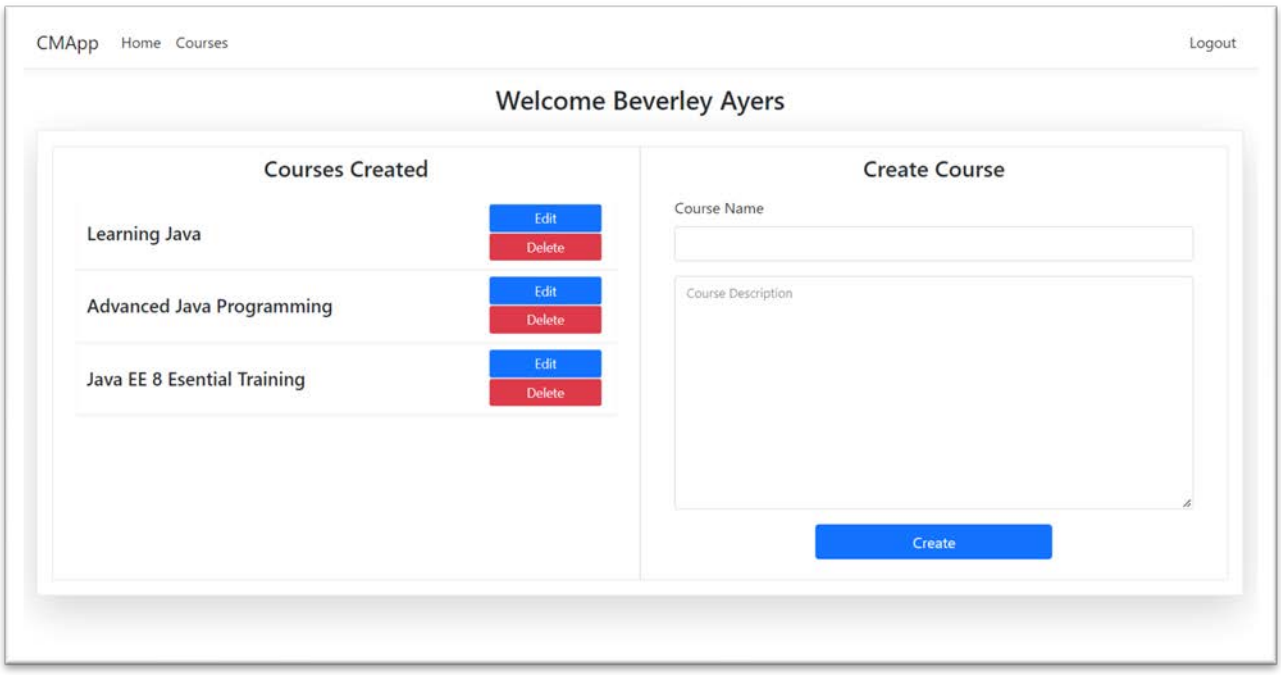
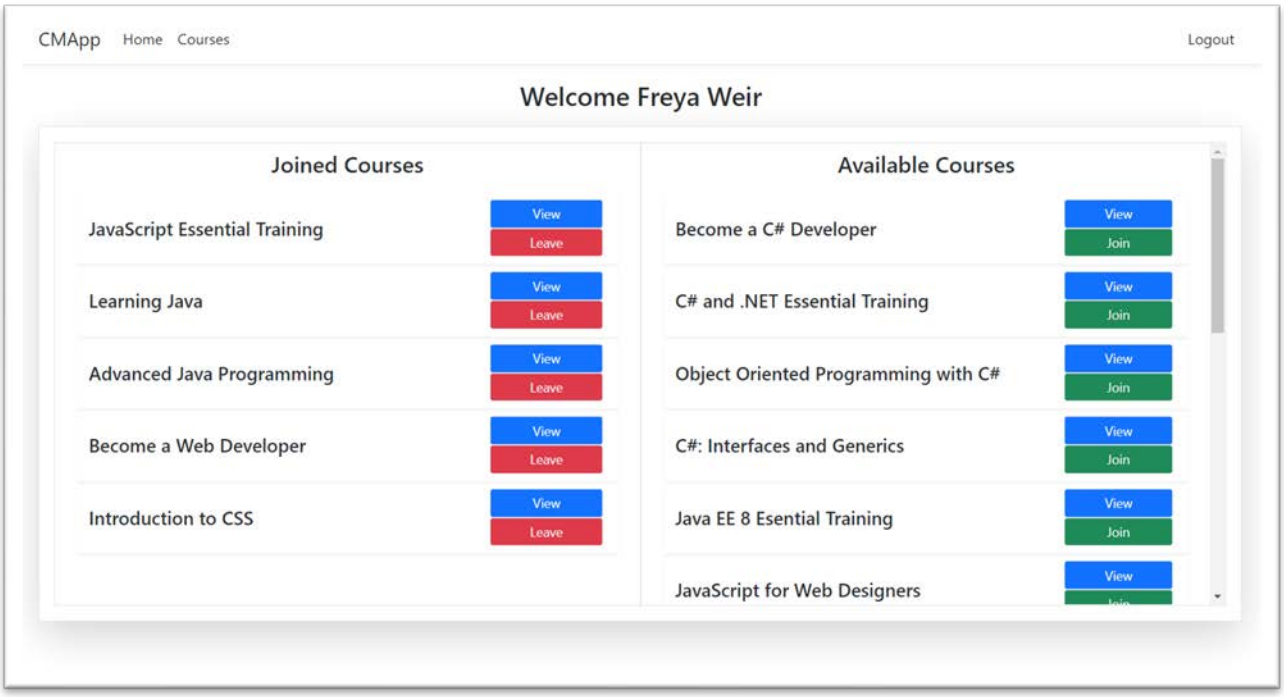
ΕΙΣΑΓΩΓΗ

Τελικό project για ΣΕΒ ΟΠΑ – Τεχνικός Εφαρμογών Πληροφορικής

Το project εστιάζει στην δημιουργία μιας διαχειριστικής εφαρμογής student/teacher οι οποίοι σχετίζονται με courses. Η συσχέτιση αυτή γίνεται μέσω ενός Account για τον καθένα όπου δίνει πρόσβαση σε διάφορα “features” αυτής της εφαρμογής.





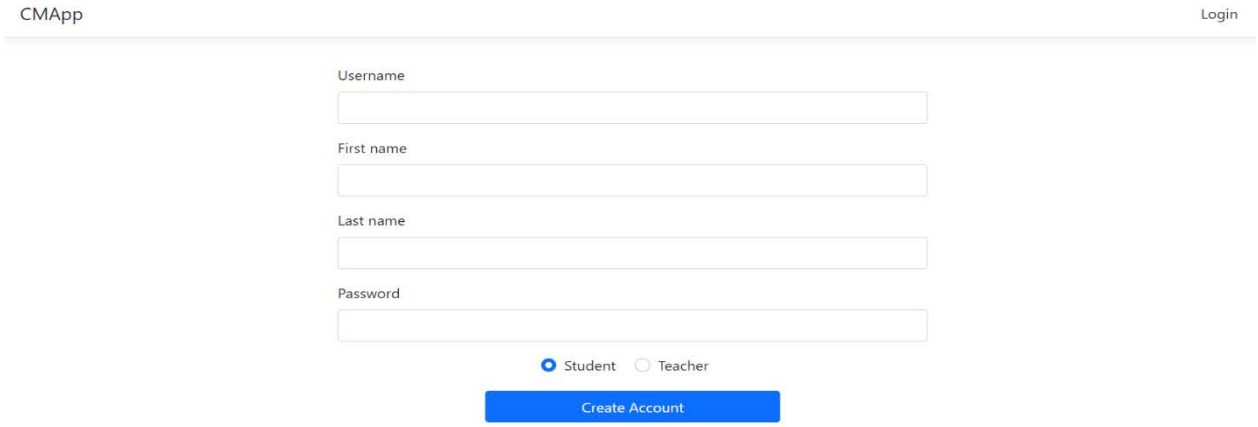
ΑΝΑΛΥΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

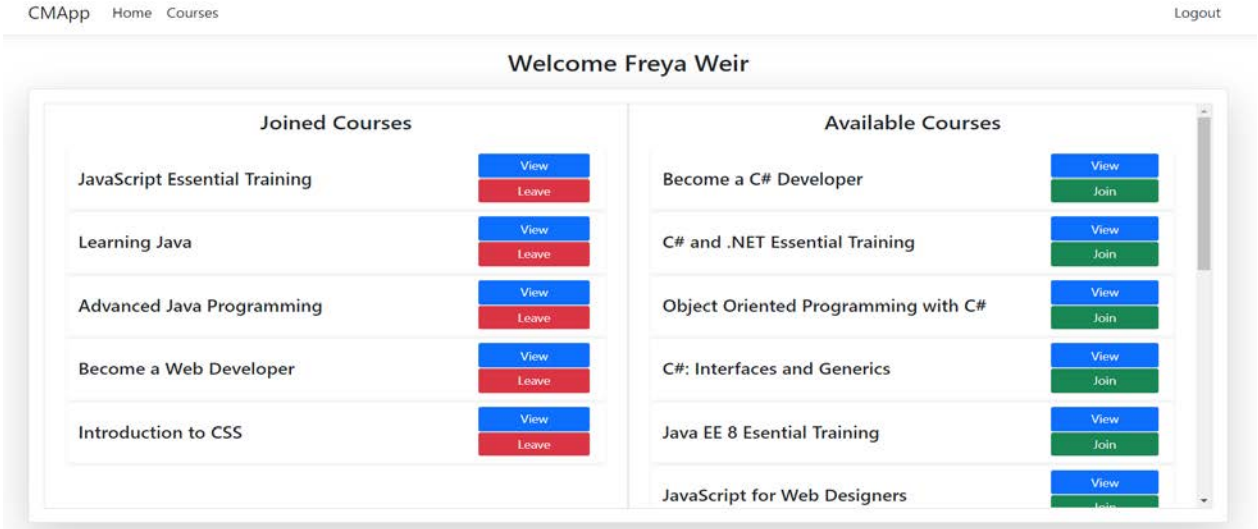
Για την δημιουργία αυτού του project θα χρησιμοποιήσουμε C# - Razor Pages μαζί με Entity Framework για το back-end. Το front-end θα υλοποιηθεί με την βοήθεια των Razor Pages όπου περιέχουν HTML, CSS, JavaScript και το Bootstrap library. Για την αποθήκευση των δεδομένων μας θα χρησιμοποιήσουμε Microsoft SQL Database.

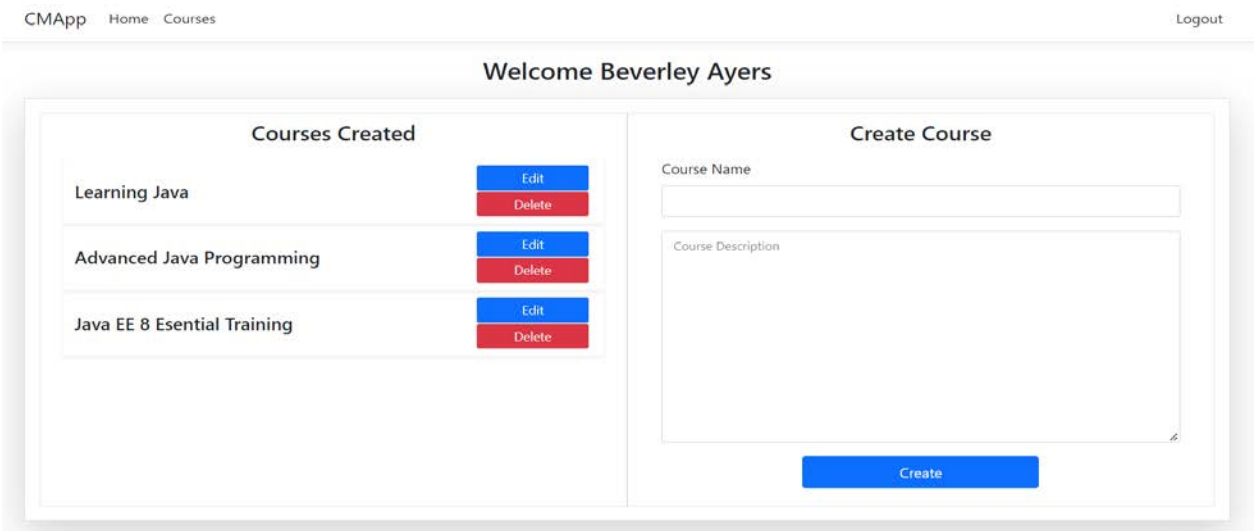
Με τον συνδυασμό αυτών τον τεχνολογιών θα καταφέρουμε να έχουμε ένα Web Application το οποίο θα μας δίνει την δυνατότητα δημιουργίας λογαριασμού Teacher ή Student όπου στην συνέχεια αναλόγως με το είδος του λογαριασμού θα μπορούμε να κάνουμε Create, Read, Update και Delete Courses.

ΣΧΕΔΙΑΣΜΟΣ

Εφαρμογή

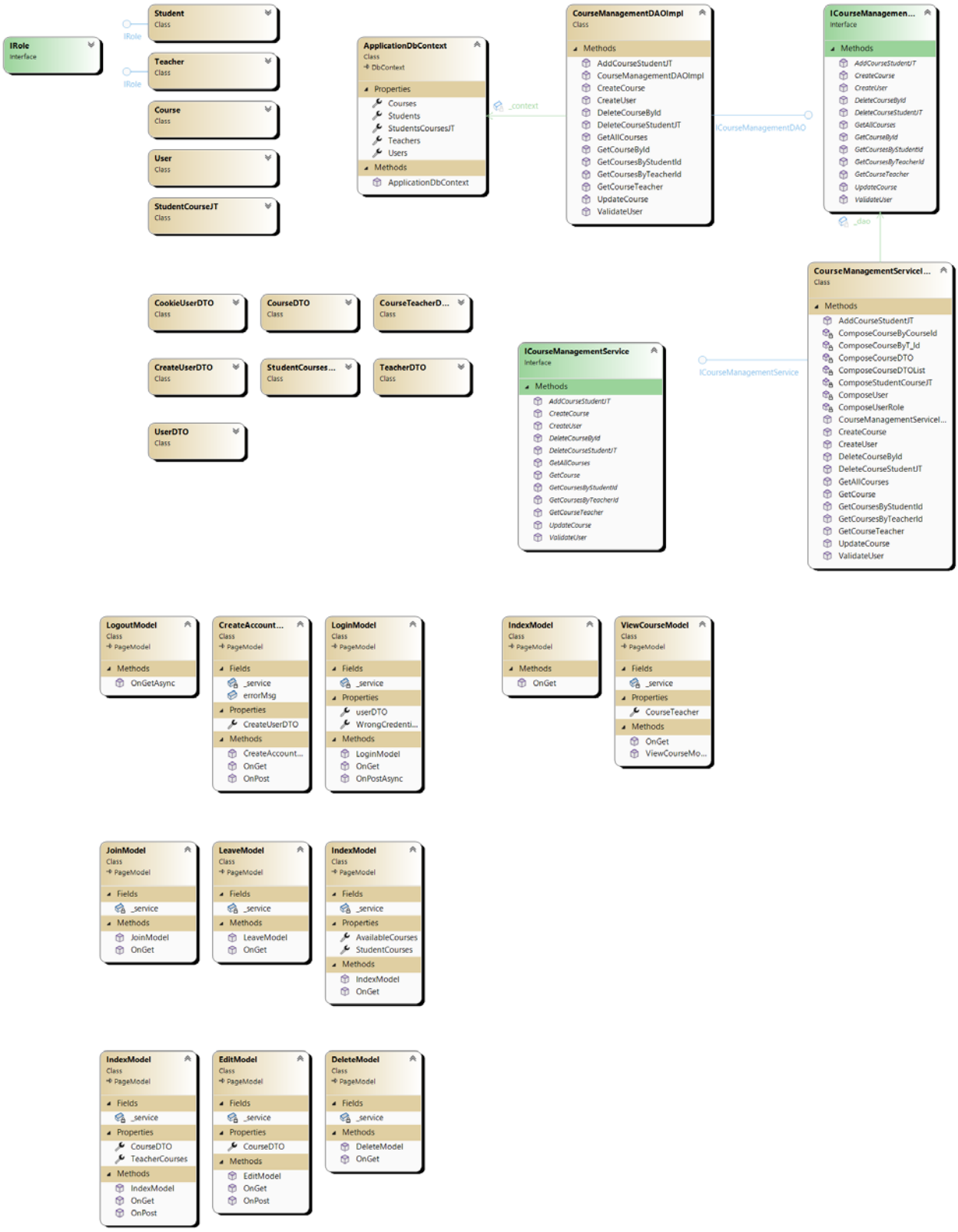






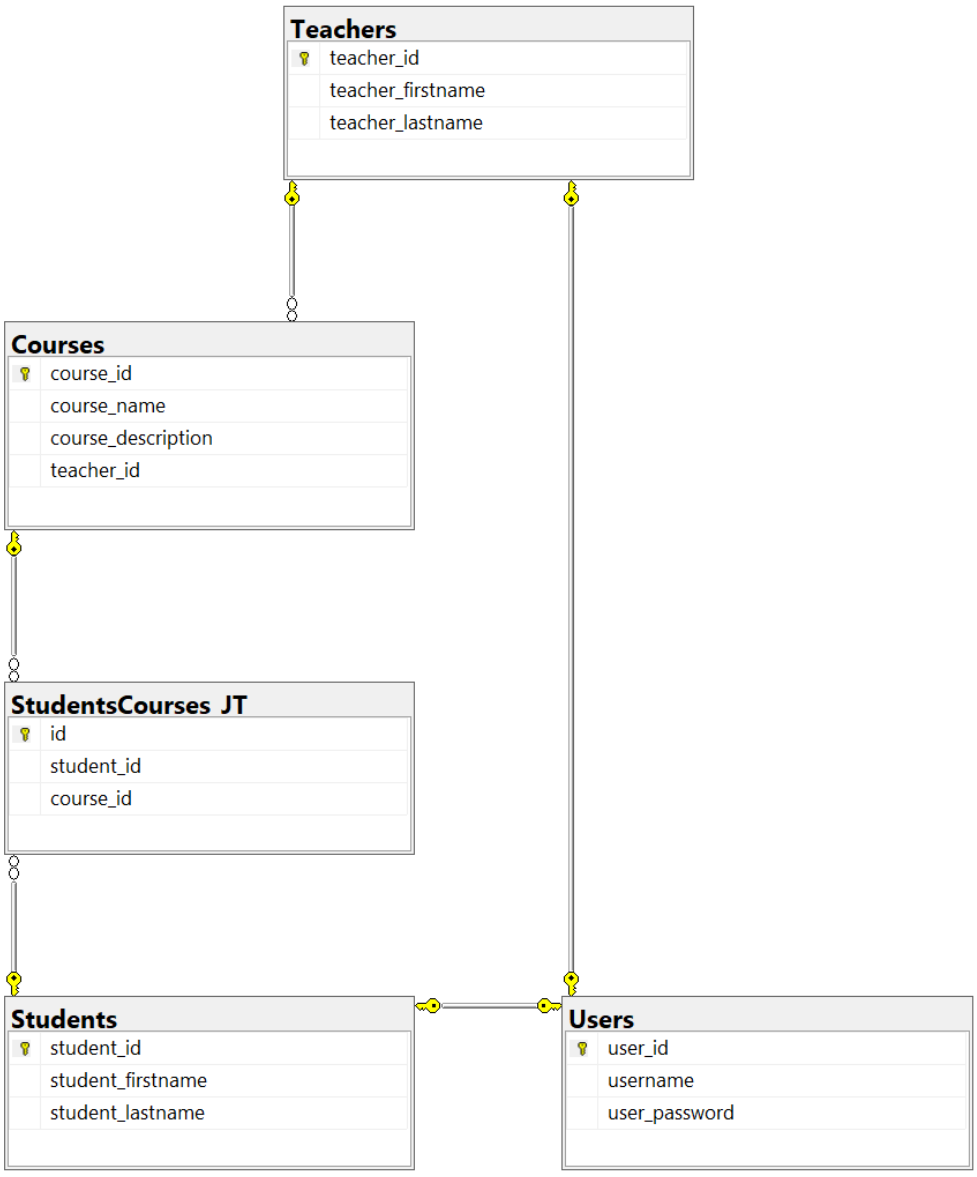
ΣΧΕΔΙΑΣΜΟΣ

Κώδικας



ΣΧΕΔΙΑΣΜΟΣ

Βάση Δεδομένων



ΣΧΕΔΙΑΣΜΟΣ

Η υλοποίηση γίνεται με την monolithic αρχιτεκτονική MVC.

- Για την μεταφορά δεδομένων από τον χρήστη στην εφαρμογή μας χρησιμοποιούμε DTO αντικείμενα όπου με την βοήθεια του ASP.NET έχουμε και validation των δεδομένων που περιέχουν. Ένα DTO αντικείμενο μοιάζει κάπως έτσι:

```
public class CreateUserDTO
{
    [BindProperty]
    [Required]
    [MinLength(4)]
    [MaxLength(16)]
    public string? Username { get; set; }

    [BindProperty]
    [Required]
    [MinLength(3)]
    [MaxLength(55)]
    public string? Firstname { get; set; }

    [BindProperty]
    [Required]
    [MinLength(3)]
    [MaxLength(55)]
    public string? Lastname { get; set; }

    [BindProperty]
    [Required]
    [MinLength(6)]
    [MaxLength(24)]
    public string? Password { get; set; }

    [BindProperty]
    [Required]
    public string? Role { get; set; }

    public CreateUserDTO()
    {
        Username = "";
        Firstname = "";
        Lastname = "";
        Password = "";
        Role = "";
    }
}
```

Annotation από το ASP.NET, αυτοματοποιεί το data binding χωρίς να χρειαστεί να κάνουμε HttpGet.

Annotation από το ASP.NET, βοηθάει στο validation δεδομένων κάνοντας το συγκεκριμένο field αναγκαίο.

Annotation από το ASP.NET, βοηθάει στο validation δεδομένων κάνοντας το συγκεκριμένο field να έχει τουλάχιστον τρεις χαρακτήρες, έτσι ώστε όταν καλεστεί το ModelState.IsValid() μέσα στο “Model” των Razor Pages να πάρουμε error.

Annotation από το ASP.NET, βοηθάει στο validation δεδομένων κάνοντας το συγκεκριμένο field να έχει το πολύ είκοσι τέσσερις χαρακτήρες, έτσι ώστε όταν καλεστεί το ModelState.IsValid() μέσα στο “Model” των Razor Pages να πάρουμε error.

Απλό field με Get/Set για να παίρνουμε ή να βάζουμε δεδομένα.

Ένας απλός constructor όπου μας διευκολύνει στην δημιουργία αντικειμένων τύπου CreateUserDTO.

ΣΧΕΔΙΑΣΜΟΣ

- Για την συνδεσιμότητα στην βάση δεδομένων μας χρησιμοποιούμε τις δυνατότητες του Entity Framework, όπου μετατρέπει τα data models μας σε αντικείμενα όπου αντιστοιχούν στα tables μέσα στην βάση δεδομένων. Στην συνέχεια όπου θα δούμε μπορούμε να κάνουμε και LINQ queries πάνω στα αντικείμενα αυτά. Την σύνδεση στην βάση δεδομένων μας την καταφέρνουμε με αυτό εδώ το class:

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
   options) : base(options) { }

    public DbSet<User> Users { get; set; }
    public DbSet<Student> Students { get; set; }
    public DbSet<Teacher> Teachers { get; set; }
    public DbSet<Course> Courses { get; set; }
    public DbSet<StudentCourseJT> StudentsCoursesJT { get; set; }
}
```

Κληρονομούμε από το DbContext όπου μας δίνει το Entity Framework για να μπορούμε να κάνουμε Queries στην βάση δεδομένων.

Dependency Injection

Με βάση τον τύπο των δεδομένων όπου δέχονται τα DbSets γίνεται και η σωστή αντιστοίχιση τους στην βάση δεδομένων μας.

```
[Table("StudentsCourses_JT")]
public class StudentCourseJT
{
    [Key]
    [Column("id")]
    public int Id { get; set; }

    [ForeignKey("Student")]
    [Column("student_id")]
    public int StudentId { get; set; }

    [ForeignKey("Course")]
    [Column("course_id")]
    public int CourseId { get; set; }
}
```

Annotation όπου χρησιμοποιείται για την αντιστοίχιση του αντικειμένου σε κάποιο table μέσα στην βάση δεδομένων μας.

Model class για την δημιουργία αντικειμένων όπου δέχεται το DbSet.

Annotation όπου χρησιμοποιείται για την δήλωση του field Id ως primary key.

Annotation όπου χρησιμοποιείται για την δήλωση του field StudentId ως foreign key.

Annotation όπου χρησιμοποιείται για την αντιστοίχιση του field CourseId με το column course_id μέσα στην βάση δεδομένων μας.

ΣΧΕΔΙΑΣΜΟΣ

- Για την οργάνωση του κώδικα μας χρησιμοποιούμε ένα DAO class όπου στην συνέχεια γίνετε injected μέσα στο Service μας, το interface του DAO είναι κάπως έτσι:

```
public interface ICourseManagementDAO
{
    bool ValidateUser(User user, out CookieUserDTO? cud);
    bool CreateUser(User user, IRole role);
    bool CreateCourse(Course course);
    bool UpdateCourse(Course course);
    List<Course> GetAllCourses();
    List<CourseDTO> GetCoursesByStudentId(int id);
    List<CourseDTO> GetCoursesByTeacherId(int id);
    Course GetCourseById(int id);
    CourseTeacherDTO GetCourseTeacher(int id);
    bool AddCourseStudentJT(StudentCourseJT data);
    bool DeleteCourseStudentJT(int id);
    bool DeleteCourseById(int id);
}
```

Αυτή η method υλοποιημένη, ελέγχει username και password όπου έχει δοθεί από τον χρήστη με την βάση δεδομένων για το authentication του, και στην συνέχεια δημιουργεί ένα Cookie για το authorization level όπου θα έχει ο χρήστης στο app μας.

The interface speaks for itself. 😊

ΣΧΕΔΙΑΣΜΟΣ

```
public class CourseManagementDAOImpl : ICourseManagementDAO
{
  ApplicationDbContext _context;
  public CourseManagementDAOImpl(ApplicationDbContext context)
   {
      _context = context;
   }

  public List<CourseDTO> GetCoursesByTeacherId(int id)
   {
      try
      {
          var query =
              (from course in _context.Courses
               where course.T_Id == id
               select course).ToList();

          List<CourseDTO> result = new();

          foreach (var course in query)
          {
              result.Add(new()
              {
                  Id = course.Id,
                  Name = course.Name,
              });
          }

          return result;
      }
      catch (Exception)
      {
          throw;
      }
   }
```

Υλοποίηση του ICourseManagementDAO interface

Dependency Injection του DbContext όπου είδαμε πριν, έτσι ώστε να μπορούμε να κάνουμε queries στην βάση δεδομένων.

Με LINQ και Entity Framework μπορούμε να κάνουμε με ευκολία queries στην βάση δεδομένων μας.

Με βάση το προηγούμενο query δημιουργούμε ένα αντικείμενο CourseDTO για την επιστροφή των δεδομένων στο service.

ΣΧΕΔΙΑΣΜΟΣ

```
public bool ValidateUser(User user, out CookieUserDTO? cud)
{
    cud = null;
    try
    {
        var result = _context.Users.FirstOrDefault(x => x.Username ==
            user.Username && x.Password == user.Password);

        if (result is not null)
        {
            cud = new();
            int? id = result?.Id;
            cud.Username = result?.Username!;
            cud.Id = result?.Id.ToString()!;

            if (_context.Teachers.Any(x => x.Id == result!.Id))
            {
                var teacher = _context.Teachers.FirstOrDefault(
                x => x.Id == id);
                cud.Firstname = teacher?.Firstname!;
                cud.Lastname = teacher?.Lastname!;
                cud.Role = "Teacher";
            }
            else
            {
                var student = _context.Students.FirstOrDefault(
                x => x.Id == id);
                cud.Firstname = student?.Firstname!;
                cud.Lastname = student?.Lastname!;
                cud.Role = "Student";
            }
            return true;
        }
        return false;
    }
    catch (Exception)
    {
        throw;
    }
}
.....
}
```

Υλοποίηση της method ValidateUser όπου επιστρέφει ένα αντικείμενο όπου χρησιμοποιείται ως Cookie και true/false αναλόγως εάν το validation είναι επιτυχές.

Εδώ κάνουμε ένα query στην βάση δεδομένων μας με βάση το username και password όπου έχει δοθεί από το service.

Εάν το query δεν μας επίστρεψε κάτι σημαίνει ότι το service έδωσε στοιχεία για χρήστη όπου δεν υπάρχουν.

Προετοιμασία του Cookie αντικειμένου με βάση το username id όπου έγινε validated .

Ο Κώδικας είναι ελλιπής λόγω χώρου, παρόμοια υλοποίηση υπάρχει και στις άλλες μεθόδους. Μπορείτε να προβείτε στα αρχεία κώδικα για περισσότερες πληροφορίες.

ΣΧΕΔΙΑΣΜΟΣ

- Μετά από το DAO πηγαίνουμε στο Service το οποίο γίνεται Injected μέσα στους Controllers μας όπου βρίσκονται μέσα στα Razor Pages σαν “Models”. Το interface του Service είναι κάπως έτσι:

```
public interface ICourseManagementService
{
    bool ValidateUser(UserDTO userDTO, out CookieUserDTO? cud);

    bool CreateUser(CreateUserDTO createUserDTO);

    bool CreateCourse(CourseDTO courseDTO);

    bool UpdateCourse(CourseDTO courseDTO);

    List<CourseDTO>? GetAllCourses();

    List<CourseDTO>? GetCoursesByStudentId(int id);

    List<CourseDTO>? GetCoursesByTeacherId(int id);

    CourseDTO? GetCourse(int id);

    CourseTeacherDTO? GetCourseTeacher(int id);

    bool AddCourseStudentJT(StudentCoursesJTDTO studentCoursesJTDTO);

    bool DeleteCourseStudentJT(int id);

    bool DeleteCourseById(int id);
}
```

Το interface για το Service.

```
public class CourseManagementServiceImpl : ICourseManagementService
{
        private ICourseManagementDAO _dao;
        public CourseManagementServiceImpl(ICourseManagementDAO dao)
        {
            _dao = dao;
        }

        public bool CreateUser(CreateUserDTO createUserDTO)
        {
            try
            {
                ComposeUserRole(createUserDTO, out User? user, out IRole?
                role);
                return _dao.CreateUser(user!, role!);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }
        }
    ……
}
```

Υλοποίηση του Service με το interface.

Dependency Injection του DAO

Helper function που δίνει τιμές στο user και role όπου στην συνέχεια στέλνονται στο DAO.

Παρατηρούμε ότι δεν χρειάζεται να κάνουμε validate τα δεδομένα όπου παίρνουμε από τον χρήστη διότι είναι ήδη validated από το κάθε controller όπου έρχονται με την βοήθεια του ASP.NET Razor Pages / MVC.

Ο Κώδικας είναι ελλιπής λόγω χώρου, παρόμοια υλοποίηση υπάρχει και στις άλλες μεθόδους. Μπορείτε να προβείτε στα αρχεία κώδικα για περισσότερες πληροφορίες.

ΣΧΕΔΙΑΣΜΟΣ

- Μετά από το Service πηγαίνουμε στο View και Controller όπου είναι τα Razor Pages. Ένα σημαντικό Razor Page στην εφαρμογή μας είναι το Login όπου βρίσκεται μέσα στο Pages/Account/ directory. Το Login.cshtml είναι το view και το Login.cshtml.cs είναι ο Controller. Είναι σημαντικό το Login διότι εδώ γίνεται η δημιουργία του Cookie όπου χρησιμοποιείται για το Authorization level του χρήστη όπου έχει κάνει login. Το View και ο Controller του Login είναι κάπως έτσι:

Το View υπάρχει περίπτωση να επιστρέψει στον Controller ένα username εάν ο χρήστης έχει δώσει λάθος στοιχεία.

```
@page "{username?}"
@model CourseManagementApp.Pages.Account.LoginModel
@{
    ViewData["Title"] = "Login";
}
```

Με την βοήθεια του _Layout.cshtml όπου βρίσκεται μέσα στο Pages/Shared directory δίνουμε έναν τίτλο για την τελική υλοποίηση του View.

Δημιουργία μιας απλής form για να στείλουμε δεδομένα στον Controller.

```
<div class="container w-50 py-3">
    <form method="POST">
        <div class="mb-3">
            <label for="username" class="form-label">Username</label>
            <input name="userDTO.Username" type="text" class="form-control"
              id="username" value="@Model.userDTO.Username">
            <span asp-validation-for="@Model.userDTO.Username" class="text-
             danger"></span>
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input name="userDTO.UserPassword" type="password" class="form-
              control" id="password">
            <span asp-validation-for="@Model.userDTO.UserPassword"
             class="text-danger"></span>
        </div>
        @if (Model.WrongCredentials)
        {
            <span class="text-danger">Wrong credentials inserted!</span>
        }
        <div class="text-center my-2">
            <span class="text-center">Don't have an account? <a
             href="/Account</a></span>
        </div>
        <div class="text-center
            <button type="submit" class="btn btn-primary w-50">Login</button>
        </div>
    </form>
</div>
```

Με αυτό το name ο Controller καταλαβαίνει για ποιο αντικείμενο μιλάμε.

Error μήνυμα που προέρχεται από το ASP.NET Razor Pages / MCV εάν το password δεν τηρεί κάποιους κανόνες οι οποίοι βρίσκονται μέσα στο userDTO με την μορφή annotations.

Μήνυμα που εμφανίζεται στον χρήστη εάν έχει δώσει λανθασμένα στοιχεία για το login του.

ΣΧΕΔΙΑΣΜΟΣ

Annotation όπου αφήνει Unauthorized χρήστες να εισέλθουν στο Login Page.

Υλοποίηση του “Controller” για το Login page.

Annotations για το Validation και Data Binding των δεδομένων μας που παίρνουμε από το View.

Dependency Injection του Service στον Controller μας.

Όταν το Login Page καλείται το OnGet method τρέχει και κάνει έλεγχο για το εάν ο χρήστης είναι είδη logged in, έτσι ώστε να τον κάνει redirect σε κάποια άλλη σελίδα.

Όταν ο χρήστης κάνει submit δεδομένα με την βοήθεια του form, καλείται το OnPostAsync, το οποίο πρέπει να είναι asynchronous διότι δημιουργεί το Cookie μας, το οποίο χρησιμοποιείται από το app για όλα τα ζητήματα όπου αφορούν το Authorization.

Το ModelState δεν θα είναι valid εάν τα δεδομένα μέσα στα DTO μας δεν τηρούν τους κανόνες όπου τους έχουν δοθεί.

Call στην βάση δεδομένων μας με την χρήση του service για το Authentication των στοιχείων όπου έχει δώσει ο χρήστης.

Δημιουργία λίστας τύπου Claim που χρησιμο-ποιείται από το Cookie μας για το Authorization level του χρήστη.

Εάν ο χρήστης δεν είναι βούρλο και μας έχει δώσει σωστά Credentials θα γίνει redirected από το Login Page, αλλιώς θα μήνη στο ίδιο για κιάλι προσπάθεια στο να κάνη login.

```
[AllowAnonymous]
public class LoginModel : PageModel
{
    [BindProperty, Required]
    public UserDTO userDTO { get; set; }

    [BindProperty]
    public bool WrongCredentials { get; set; }

    private readonly ICourseManagementService _service;
    public LoginModel(ICourseManagementService service)
    {
        _service = service;
    }

    public IActionResult OnGet(string? username)
    {
        if (User.Identity!.IsAuthenticated)
        {
            return Redirect("/");
        }

        userDTO = new();
        if (username is not null)
            userDTO.Username = username;

        return Page();
    }

    public async Task<IActionResult> OnPostAsync()
    {
        if (ModelState.IsValid)
        {
            if (_service.ValidateUser(userDTO, out CookieUserDTO? cud))
            {
                var claims = new List<Claim> {
                    new Claim(ClaimTypes.Role, cud?.Role!),
                    new Claim(ClaimTypes.Sid, cud?.Id!),
                    new Claim(ClaimTypes.Name, cud?.Firstname! + " " +
                        cud?.Lastname)
                };

                var identity = new ClaimsIdentity(claims, "CookieAuth");
                ClaimsPrincipal claimsPrincipal = new
                        ClaimsPrincipal(identity);

                await HttpContext.SignInAsync("CookieAuth",
                        claimsPrincipal);

                return Redirect("/");
            }else
            {
                WrongCredentials = true;
            }
        }

        return Page();
    }
}
```

ΣΧΕΔΙΑΣΜΟΣ

- Στην συνέχεια είναι το Program.cs στο οποίο ετοιμάζουμε των Kestrel Server, τα Middleware όπου θα χρησιμοποιήσουμε, και κάνουμε configure μερικά Dependency Injections που θα χρειαστούν:

```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddRazorPages();

        builder.Services.AddAuthentication("CookieAuth")
          .AddCookie("CookieAuth", options =>
        {
            options.Cookie.Name = "CookieAuth";
            options.LoginPath = "/Account/Login";
            options.LogoutPath = "/Account/Logout";
            options.AccessDeniedPath = "/";
        });

        builder.Services.AddScoped<ICourseManagementService,
                CourseManagementServiceImpl>();
        builder.Services.AddScoped<ICourseManagementDAO,
                CourseManagementDAOImpl>();

        builder.Services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(builder.Configuration
                .GetConnectionString("DefaultConnection"))
        );

        var app = builder.Build();

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthentication();
        app.UseAuthorization();

        app.MapRazorPages();

        app.Run();
    }
}
```

Αντικείμενο που μας βοηθάει στην δημιουργία των Configurations του Kestrel Server.

Configuration όπου κάνει των Kestrel Server να αναγνωρίζει ότι χρησιμοποιούμε Razor Pages.

Configuration δημιουργίας Authentication τύπου Cookie στο οποίο περνάμε options σχετικά με το Cookie.

Configuration έτσι ώστε να μπορούμε να κάνουμε Dependency Injection με το DbContext, DAO, και Service μας.

Middleware που προσθέτει redirection των http requests σε https.

Method το οποίο μας κάνει Build τον Kestrel Server με τα configurations του.

Το UseRouting middleware προσθέτει routing για την εύρεση των controllers.

Middleware που μας δίνει την δυνατότητα να έχουμε πρόσβαση σε αρχεία μέσα στον φάκελο wwwroot από οποιαδήποτε καλεσμένη σελίδα.

Middleware για Authentication και Authorization.

Αυτό το Middleware προσθέτει endpoints στα Razor Pages μας, έτσι ώστε το UseRouting να μπορεί να συνδεθεί μαζί τους.

Terminal middleware το οποίο τρέχει των Kestrel Server με όλα τα configurations και middleware που του έχουμε δώσει.

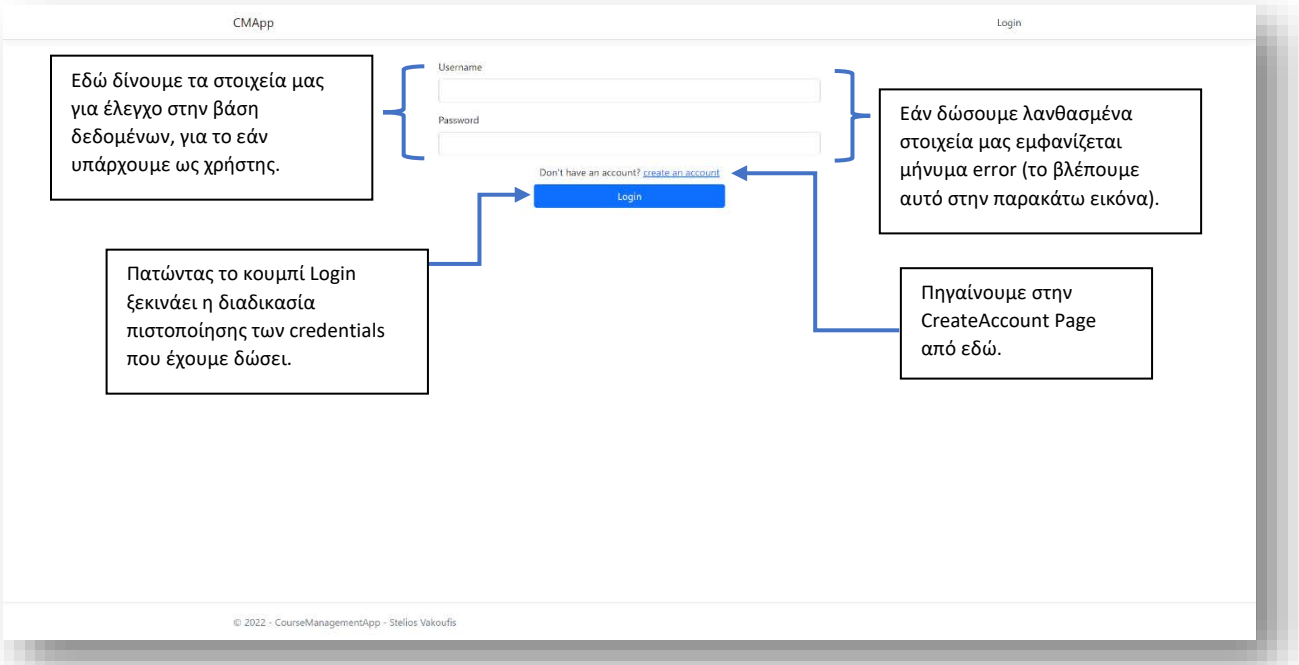
ΣΧΕΔΙΑΣΜΟΣ

- Στο τέλος είναι η βάση δεδομένων η οποία υλοποιείται με το παρακάτω script:

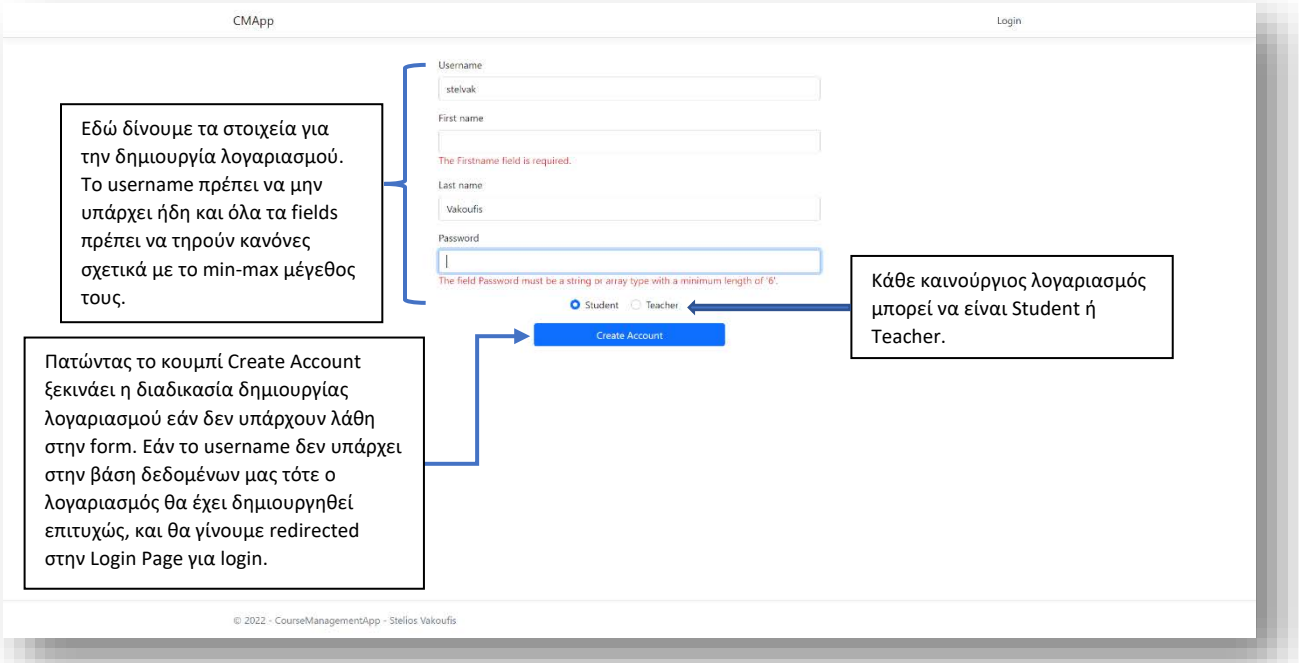
```
-- DATABASE CREATION
CREATE DATABASE FinalExerciseKEDIVIM

USE FinalExerciseKEDIVIM

-- TABLES CREATION
CREATE TABLE Users(
	[user_id] int UNIQUE NOT NULL IDENTITY(1, 1),
	[username] VARCHAR(16) UNIQUE NOT NULL,
	[user_password] VARCHAR(24) NOT NULL,
	PRIMARY KEY ([user_id])
);

CREATE TABLE Teachers (
	teacher_id int NOT NULL,
	teacher_firstname VARCHAR(55) NOT NULL,
	teacher_lastname VARCHAR(55) NOT NULL,
	PRIMARY KEY(teacher_id),
	FOREIGN KEY(teacher_id) REFERENCES Users([user_id])
);

CREATE TABLE Students(
	student_id int NOT NULL,
	student_firstname VARCHAR(55) NOT NULL,
	student_lastname VARCHAR(55) NOT NULL,
	PRIMARY KEY(student_id),
	FOREIGN KEY(student_id) REFERENCES Users([user_id])
);

CREATE TABLE Courses (
	course_id int NOT NULL IDENTITY(1, 1),
	course_name VARCHAR(55) NOT NULL,
	course_description VARCHAR(5000) NOT NULL,
	teacher_id int NOT NULL,
	PRIMARY KEY(course_id),
	FOREIGN KEY(teacher_id) REFERENCES Teachers(teacher_id)
);

CREATE TABLE StudentsCourses_JT(
	id int NOT NULL IDENTITY(1, 1),
	student_id int NOT NULL,
	course_id int NOT NULL,
	PRIMARY KEY(id),
	FOREIGN KEY(student_id) REFERENCES Students(student_id),
	FOREIGN KEY(course_id) REFERENCES Courses(course_id) ON DELETE CASCADE
);
```

ΕΠΙΔΕΙΞΗ ΚΑΙ ΕΛΕΓΧΟΣ

- Το πρώτο πράγμα που βλέπουμε όταν ανοίγουμε την Web εφαρμογή είναι το Login Page. Στο Login Page μπορούμε να κάνουμε login εάν έχουμε ήδη λογαριασμό ή να πατήσουμε create an account για την δημιουργία ενός.

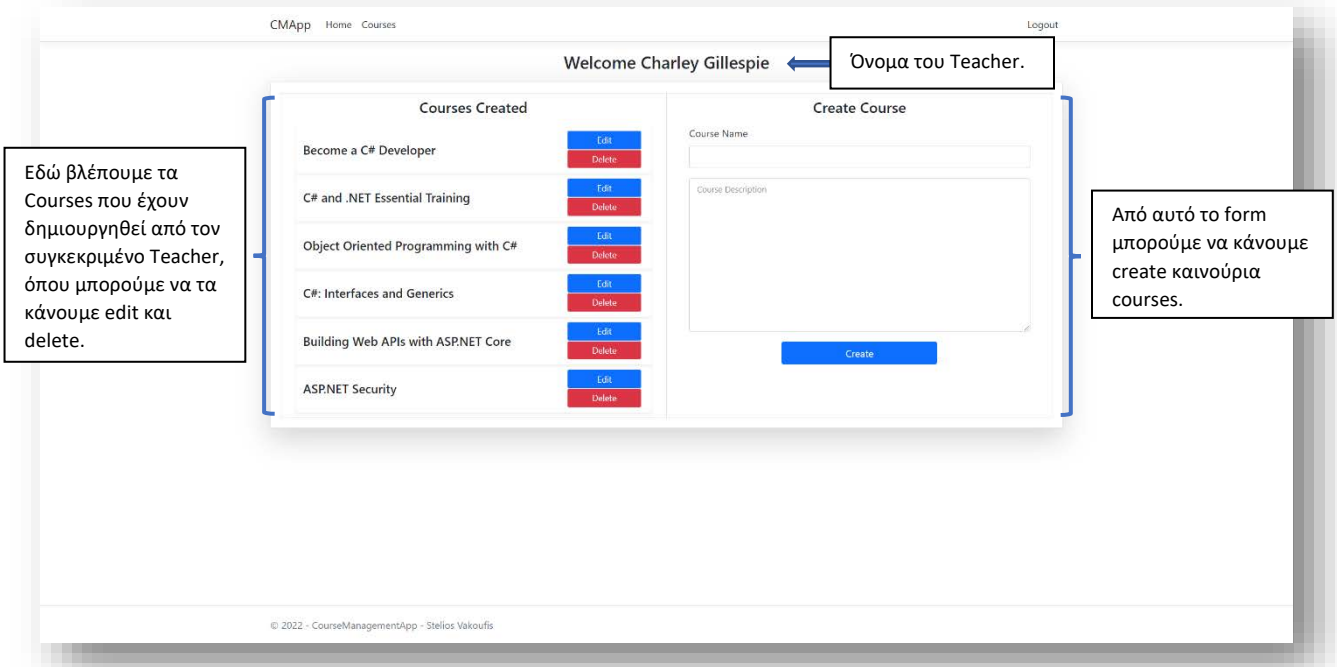


- Το create an account page είναι κάπως έτσι:

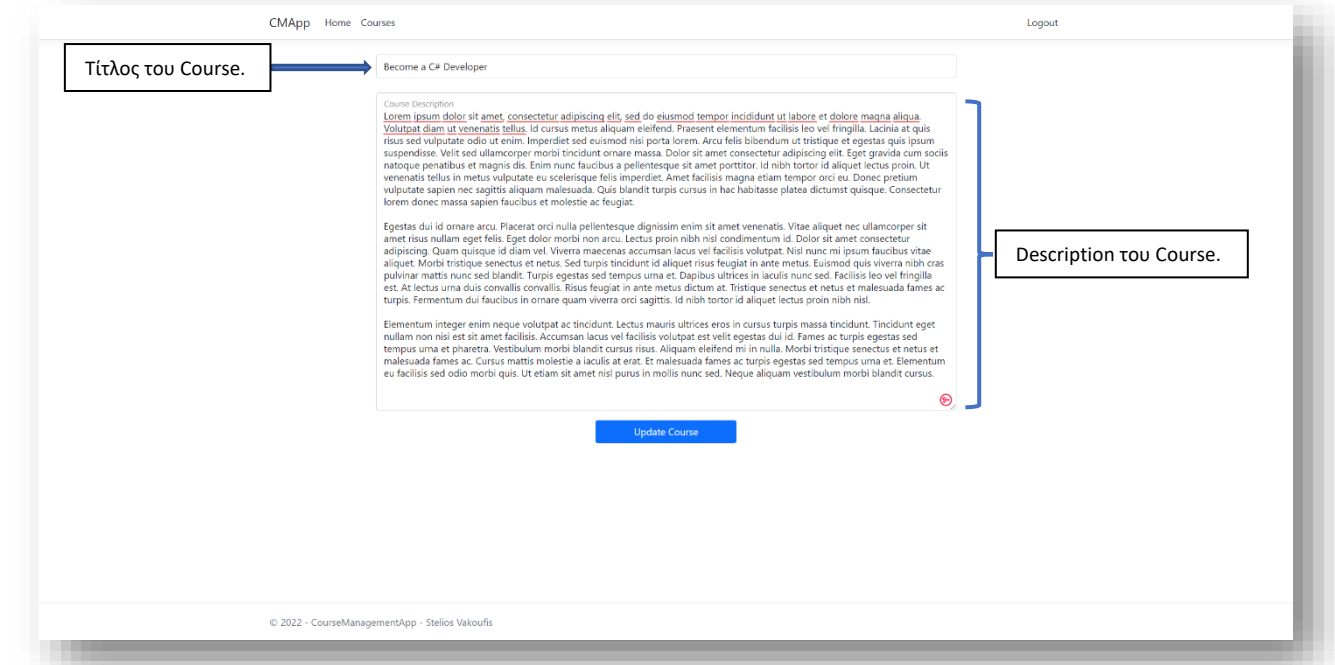


ΕΠΙΔΕΙΞΗ ΚΑΙ ΕΛΕΓΧΟΣ

- Μετά από επιτυχές Login, αναλόγως με το εάν είμαστε Student ή Teacher έχουμε πρόσβαση και σε διαφορετική Course Page. Το Course Page του Teacher είναι κάπως έτσι:

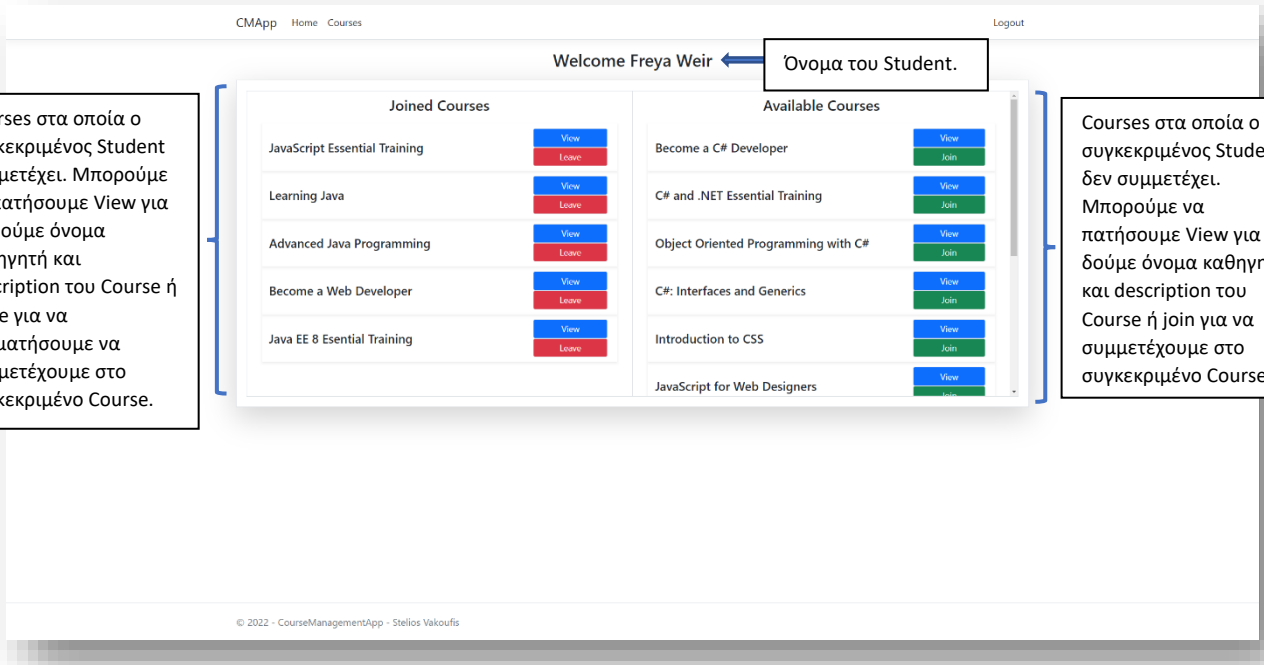


- Το edit που μπορεί να κάνει ο Teacher φαίνεται κάπως έτσι:

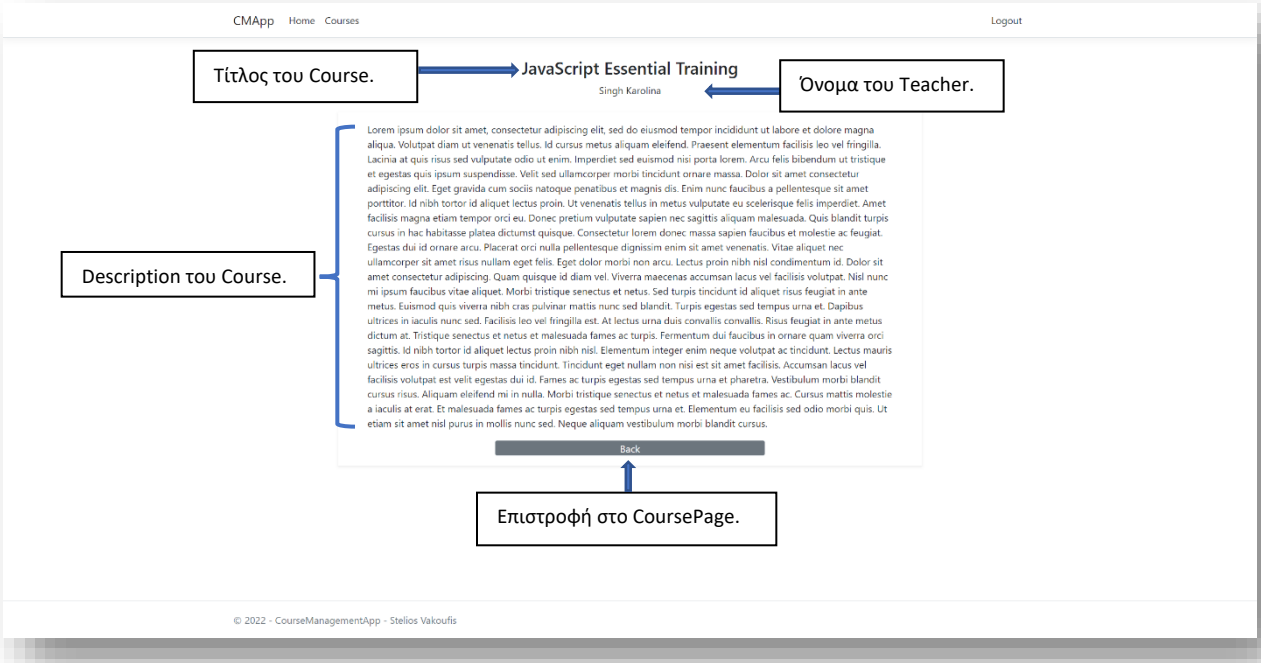


ΕΠΙΔΕΙΞΗ ΚΑΙ ΕΛΕΓΧΟΣ

- Τέλος το Course Page του Student είναι κάπως έτσι:



- Το View Course είναι κάπως έτσι:



ΒΙΒΛΙΟΓΡΑΦΙΑ

- Ο Κώδικας της εφαρμογής βρίσκεται ανεβασμένος στο GitHub https://github.com/Zeenus/CMApp
- Πολλές απορίες λυθήκαν με την βοήθεια του Stack Overflow και Google
- Καινούριες δεξιότητες αναπτύχθηκαν από το LinkedIn Learning και YouTube.