Create Course Class and using Viewmodel to create View Assign

Objective: Build a robust ASP.NET Core MVC application managing Students, Courses, and Classes using SOLID principles and Service Pattern.

Architecture:

- **Data Layer:** Entity Framework Core (SQL Server).
- **Service Layer:** Interfaces & Implementations (Business Logic).
- **Presentation Layer:** Controllers (Thin) & Views.

STEP 1: Define Domain Models (Entities)

Define the core objects. A Class has many Students, many Courses, and one Lecturer.

1.1. Models/Course.cs

```
using System.ComponentModel.DataAnnotations;

namespace StudentManagementMvc.Models
{
    public class Course
    {
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        [Display(Name = "Course Title")]
        public string Title { get; set; }

        [Range(1, 10)]
        public int Credits { get; set; }

        // Many-to-Many relationship with Class
        public ICollection<Class> Classes { get; set; } = new List<Class>();
    }
}
```

1.2. Models/Class.cs

```
using System.ComponentModel.DataAnnotations;
```

```
using System.ComponentModel.DataAnnotations.Schema;

namespace StudentManagementMvc.Models
{
    public class Class
    {
        public int Id { get; set; }

        [Required]
        [StringLength(20)]
        [Display(Name = "Class Name")]
        public string ClassName { get; set; }

        // Link to Lecturer (ApplicationUser)
        [Display(Name = "Lecturer")]
        public string? LecturerId { get; set; }

        [ForeignKey("LecturerId")]
        public ApplicationUser? Lecturer { get; set; }

        // Many-to-Many: A class teaches multiple courses
        public ICollection<Course> Courses { get; set; } = new List<Course>();

        // One-to-Many: A class has multiple students
        public ICollection<Student> Students { get; set; } = new List<Student>();
    }
}
```

1.3. Models/Student.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace StudentManagementMvc.Models
{
    public class Student
    {
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        public string FullName { get; set; }
```

```
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        // Foreign Key to Class
        public int? ClassId { get; set; }

        [ForeignKey("ClassId")]
        public Class? Class { get; set; }
    }
}
```

STEP 2: Database Context & Migration

Configure relationships in EF Core.

2.1. Data/ApplicationDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using StudentManagementMvc.Models;

namespace StudentManagementMvc.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<Class> Classes { get; set; }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            // Configure Many-to-Many for Class <-> Course
            builder.Entity<Class>()
```

```
            .HasMany(c => c.Courses)
            .WithMany(co => co.Classes)
            .UsingEntity(j => j.ToTable("ClassCourses")); // Join table name
    }
  }
}
```

2.2. Apply Migration

Run these commands in **Package Manager Console**:

```
Add-Migration FinalSchemaSetup
Update-Database
```

STEP 3: ViewModels (DTOs)

Create a folder ViewModels. These classes transport data between Controller and View.

3.1. ViewModels/AssignCourseToClassViewModel.cs

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.ComponentModel.DataAnnotations;

namespace StudentManagementMvc.ViewModels
{
  public class AssignCourseToClassViewModel
  {
    [Required(ErrorMessage = "Please select a class.")]
    public int SelectedClassId { get; set; }

    [Required(ErrorMessage = "Please select a course.")]
    public int SelectedCourseId { get; set; }

    // Dropdown data
    public IEnumerable<SelectListItem>? ClassList { get; set; }
    public IEnumerable<SelectListItem>? CourseList { get; set; }
  }
}
```

3.2. ViewModels/AssignStudentToClassViewModel.cs

```
using Microsoft.AspNetCore.Mvc.Rendering;
```

```
using System.ComponentModel.DataAnnotations;

namespace StudentManagementMvc.ViewModels
{
    public class AssignStudentToClassViewModel
    {
        [Required]
        public int SelectedStudentId { get; set; }

        [Required]
        public int SelectedClassId { get; set; }

        public IEnumerable<SelectListItem>? StudentList { get; set; }
        public IEnumerable<SelectListItem>? ClassList { get; set; }
    }
}
```

3.3. ViewModels/AssignLecturerViewModel.cs

```
using Microsoft.AspNetCore.Mvc.Rendering;
using System.ComponentModel.DataAnnotations;

namespace StudentManagementMvc.ViewModels
{
    public class AssignLecturerViewModel
    {
        [Required]
        public int ClassId { get; set; }

        [Required]
        public string LecturerId { get; set; }

        public IEnumerable<SelectListItem>? ClassList { get; set; }
        public IEnumerable<SelectListItem>? LecturerList { get; set; }
    }
}
```

STEP 4: Service Layer (Business Logic)

Create folders: Services and Services/Interfaces.

4.1. Interfaces (Services/Interfaces/)

ICourseService.cs

```
using StudentManagementMvc.Models;

namespace StudentManagementMvc.Services.Interfaces
{
    public interface ICourseService
    {
        Task<List<Course>> GetAllCoursesAsync();
        Task AddCourseAsync(Course course);
        Task UpdateCourseAsync(Course course);
        Task DeleteCourseAsync(int id);
        Task<Course?> GetCourseByIdAsync(int id);
    }
}
```

IClassService.cs

```
using StudentManagementMvc.Models;

namespace StudentManagementMvc.Services.Interfaces
{
    public interface IClassService
    {
        Task<List<Class>> GetAllClassesAsync();
        // Secure method to get details with Authorization logic
        Task<Class?> GetClassDetailsForUserAsync(int classId, string userId, string userEmail,
bool isAdmin);
        Task AddClassAsync(Class cls);
        Task UpdateClassAsync(Class cls);
        Task DeleteClassAsync(int id);
        Task<Class?> GetClassByIdAsync(int id);
    }
}
```

IAssignmentService.cs

```
namespace StudentManagementMvc.Services.Interfaces
{
    public interface IAssignmentService
```

```
    {
        Task AssignCourseToClassAsync(int classId, int courseId);
        Task AssignStudentToClassAsync(int studentId, int classId);
        Task AssignLecturerToClassAsync(int classId, string lecturerId);
    }
}
```

4.2. Implementations (Services/)

CourseService.cs

```
using Microsoft.EntityFrameworkCore;
using StudentManagementMvc.Data;
using StudentManagementMvc.Models;
using StudentManagementMvc.Services.Interfaces;

namespace StudentManagementMvc.Services
{
    public class CourseService : ICourseService
    {
        private readonly ApplicationDbContext _context;
        public CourseService(ApplicationDbContext context) => _context = context;

        public async Task<List<Course>> GetAllCoursesAsync() => await
_context.Courses.ToListAsync();
        public async Task<Course?> GetCourseByIdAsync(int id) => await
_context.Courses.FindAsync(id);

        public async Task AddCourseAsync(Course course) {
            _context.Courses.Add(course);
            await _context.SaveChangesAsync();
        }
        public async Task UpdateCourseAsync(Course course) {
            _context.Courses.Update(course);
            await _context.SaveChangesAsync();
        }
        public async Task DeleteCourseAsync(int id) {
            var c = await _context.Courses.FindAsync(id);
            if (c != null) { _context.Courses.Remove(c); await _context.SaveChangesAsync(); }
        }
    }
}
```

ClassService.cs (Contains Authorization Logic)

```
using Microsoft.EntityFrameworkCore;
using StudentManagementMvc.Data;
using StudentManagementMvc.Models;
using StudentManagementMvc.Services.Interfaces;

namespace StudentManagementMvc.Services
{
   public class ClassService : IClassService
   {
      private readonly ApplicationDbContext _context;
      public ClassService(ApplicationDbContext context) => _context = context;

      public async Task<List<Class>> GetAllClassesAsync()
         => await _context.Classes.Include(c => c.Lecturer).ToListAsync();

      public async Task<Class?> GetClassByIdAsync(int id) => await
_context.Classes.FindAsync(id);

      public async Task<Class?> GetClassDetailsForUserAsync(int classId, string userId, string
userEmail, bool isAdmin)
      {
         var cls = await _context.Classes
            .Include(c => c.Students)
            .Include(c => c.Courses)
            .Include(c => c.Lecturer)
            .FirstOrDefaultAsync(c => c.Id == classId);

         if (cls == null) return null;

         // --- AUTHORIZATION LOGIC ---
         if (isAdmin) return cls; // Admin sees all
         if (cls.LecturerId == userId) return cls; // Lecturer sees their own class
         if (cls.Students.Any(s => s.Email.ToLower() == userEmail.ToLower())) return cls; //
Student sees their class

         return null; // Access Denied
      }

      public async Task AddClassAsync(Class cls) {
```

```
        _context.Classes.Add(cls);
        await _context.SaveChangesAsync();
    }
    public async Task UpdateClassAsync(Class cls) {
        _context.Classes.Update(cls);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteClassAsync(int id) {
        var c = await _context.Classes.FindAsync(id);
        if (c != null) { _context.Classes.Remove(c); await _context.SaveChangesAsync(); }
    }
  }
}
```

AssignmentService.cs

```
using Microsoft.EntityFrameworkCore;
using StudentManagementMvc.Data;
using StudentManagementMvc.Services.Interfaces;

namespace StudentManagementMvc.Services
{
  public class AssignmentService : IAssignmentService
  {
    private readonly ApplicationDbContext _context;
    public AssignmentService(ApplicationDbContext context) => _context = context;

    public async Task AssignCourseToClassAsync(int classId, int courseId)
    {
        var cls = await _context.Classes.Include(c => c.Courses).FirstOrDefaultAsync(c => c.Id
== classId);
        var course = await _context.Courses.FindAsync(courseId);
        if (cls != null && course != null && !cls.Courses.Contains(course))
        {
            cls.Courses.Add(course);
            await _context.SaveChangesAsync();
        }
    }

    public async Task AssignStudentToClassAsync(int studentId, int classId)
    {
        var student = await _context.Students.FindAsync(studentId);
```

```
        if (student != null) {
            student.ClassId = classId;
            await _context.SaveChangesAsync();
        }
    }

    public async Task AssignLecturerToClassAsync(int classId, string lecturerId)
    {
        var cls = await _context.Classes.FindAsync(classId);
        if (cls != null) {
            cls.LecturerId = lecturerId;
            await _context.SaveChangesAsync();
        }
    }
  }
}
```

STEP 5: Register Services

Open Program.cs. Add this **before** builder.Build():

```
// Register Services for Dependency Injection
builder.Services.AddScoped<ICourseService, CourseService>();
builder.Services.AddScoped<IClassService, ClassService>();
builder.Services.AddScoped<IAssignmentService, AssignmentService>();
```

STEP 6: Controllers

6.1. AdminManageController.cs (Assigning Logic)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using StudentManagementMvc.Models;
using StudentManagementMvc.Services.Interfaces;
using StudentManagementMvc.ViewModels;

namespace StudentManagementMvc.Controllers
{
  [Authorize(Roles = "Admin")]
  public class AdminManageController : Controller
```

```
  {
    private readonly IAssignmentService _assignService;
    private readonly IClassService _classService;
    private readonly ICourseService _courseService;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly StudentManagementMvc.Data.ApplicationDbContext _context; // For
simple Student list

    public AdminManageController(IAssignmentService assignService, IClassService
classService,
        ICourseService courseService, UserManager<ApplicationUser> userManager,
        StudentManagementMvc.Data.ApplicationDbContext context)
    {
        _assignService = assignService;
        _classService = classService;
        _courseService = courseService;
        _userManager = userManager;
        _context = context;
    }

    // 1. Assign Course
    public async Task<IActionResult> AssignCourse()
    {
        var model = new AssignCourseToClassViewModel {
            ClassList = (await _classService.GetAllClassesAsync()).Select(c => new
SelectListItem { Value = c.Id.ToString(), Text = c.ClassName }),
            CourseList = (await _courseService.GetAllCoursesAsync()).Select(c => new
SelectListItem { Value = c.Id.ToString(), Text = c.Title })
        };
        return View(model);
    }

    [HttpPost]
    public async Task<IActionResult> AssignCourse(AssignCourseToClassViewModel model)
    {
        if (ModelState.IsValid) {
            await _assignService.AssignCourseToClassAsync(model.SelectedClassId,
model.SelectedCourseId);
            TempData["Success"] = "Course assigned successfully!";
            return RedirectToAction(nameof(AssignCourse));
        }
        return View(model);
    }
```

```
    // 2. Assign Student
    public async Task<IActionResult> AssignStudent()
    {
        var model = new AssignStudentToClassViewModel {
            ClassList = (await _classService.GetAllClassesAsync()).Select(c => new
SelectListItem { Value = c.Id.ToString(), Text = c.ClassName }),
            StudentList = _context.Students.Select(s => new SelectListItem { Value =
s.Id.ToString(), Text = s.FullName })
        };
        return View(model);
    }

    [HttpPost]
    public async Task<IActionResult> AssignStudent(AssignStudentToClassViewModel model)
    {
        if (ModelState.IsValid) {
            await _assignService.AssignStudentToClassAsync(model.SelectedStudentId,
model.SelectedClassId);
            TempData["Success"] = "Student assigned successfully!";
            return RedirectToAction(nameof(AssignStudent));
        }
        return View(model);
    }

    // 3. Assign Lecturer
    public async Task<IActionResult> AssignLecturer()
    {
        var faculty = await _userManager.GetUsersInRoleAsync("Faculty");
        var model = new AssignLecturerViewModel {
            ClassList = (await _classService.GetAllClassesAsync()).Select(c => new
SelectListItem { Value = c.Id.ToString(), Text = c.ClassName }),
            LecturerList = faculty.Select(u => new SelectListItem { Value = u.Id, Text = u.FullName
?? u.Email })
        };
        return View(model);
    }

    [HttpPost]
    public async Task<IActionResult> AssignLecturer(AssignLecturerViewModel model)
    {
        if (ModelState.IsValid) {
            await _assignService.AssignLecturerToClassAsync(model.ClassId, model.LecturerId);
```

```
            TempData["Success"] = "Lecturer assigned successfully!";
            return RedirectToAction(nameof(AssignLecturer));
        }
        return View(model);
    }
  }
}
```

6.2. ClassesController.cs (CRUD + Auth)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using StudentManagementMvc.Models;
using StudentManagementMvc.Services.Interfaces;

namespace StudentManagementMvc.Controllers
{
  [Authorize]
  public class ClassesController : Controller
  {
    private readonly IClassService _classService;
    private readonly UserManager<ApplicationUser> _userManager;

    public ClassesController(IClassService classService, UserManager<ApplicationUser>
userManager)
    {
      _classService = classService;
      _userManager = userManager;
    }

    public async Task<IActionResult> Index() => View(await
_classService.GetAllClassesAsync());

    // DETAILS: The core of Security Logic
    public async Task<IActionResult> Details(int id)
    {
      var user = await _userManager.GetUserAsync(User);
      if (user == null) return Challenge();

      var isAdmin = await _userManager.IsInRoleAsync(user, "Admin");
```

```
        // Service handles logic: returns null if unauthorized
        var cls = await _classService.GetClassDetailsForUserAsync(id, user.Id, user.Email,
isAdmin);

        if (cls == null) return View("AccessDenied");

        return View(cls);
    }

    [Authorize(Roles = "Admin")]
    public IActionResult Create() => View();

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> Create(Class cls)
    {
        if (ModelState.IsValid) {
            await _classService.AddClassAsync(cls);
            return RedirectToAction(nameof(Index));
        }
        return View(cls);
    }

    // Add Edit/Delete actions here calling _classService...
  }
}
```

STEP 7: Views (The UI)

7.1. Views/Classes/Details.cshtml

Shows everything (Lecturer, Courses, Students).

```
@model StudentManagementMvc.Models.Class

<div class="container mt-4">
  <div class="d-flex justify-content-between align-items-center">
    <h1>Class: @Model.ClassName</h1>
    <a asp-action="Index" class="btn btn-secondary">Back to List</a>
  </div>
  <hr />
```

```
<div class="alert alert-info">
  <strong>👨‍🏫 Lecturer:</strong> @(Model.Lecturer?.FullName ?? "Not Assigned")
</div>

<div class="row">
  <!-- Courses Column -->
  <div class="col-md-6">
    <div class="card shadow mb-3">
      <div class="card-header bg-primary text-white">
        <h5>📚 Courses</h5>
      </div>
      <ul class="list-group list-group-flush">
        @if(Model.Courses.Any()) {
          @foreach(var c in Model.Courses) {
            <li class="list-group-item">@c.Title (@c.Credits credits)</li>
          }
        } else {
          <li class="list-group-item text-muted">No courses assigned.</li>
        }
      </ul>
    </div>
  </div>

  <!-- Students Column -->
  <div class="col-md-6">
    <div class="card shadow mb-3">
      <div class="card-header bg-success text-white">
        <h5>👨‍🎓 Students</h5>
      </div>
      <table class="table table-hover mb-0">
        <thead>
          <tr><th>Name</th><th>Email</th></tr>
        </thead>
        <tbody>
          @if(Model.Students.Any()) {
            @foreach(var s in Model.Students) {
              <tr><td>@s.FullName</td><td>@s.Email</td></tr>
            }
          } else {
            <tr><td colspan="2" class="text-muted">No students enrolled.</td></tr>
          }
        </tbody>
      </table>
```

```
        </div>
      </div>
    </div>
</div>
```

7.2. Views/AdminManage/AssignLecturer.cshtml

(Similar structure for AssignCourse/AssignStudent, just change fields).

```
@model StudentManagementMvc.ViewModels.AssignLecturerViewModel

@{ ViewData["Title"] = "Assign Lecturer"; }

<div class="container mt-5">
  <div class="card shadow col-md-8 mx-auto">
    <div class="card-header bg-warning text-dark">
      <h3>👨‍🏫 Assign Lecturer to Class</h3>
    </div>
    <div class="card-body">
      @if(TempData["Success"] != null) {
        <div class="alert alert-success">@TempData["Success"]</div>
      }

      <form asp-action="AssignLecturer" method="post">
        <div class="mb-3">
          <label class="form-label fw-bold">Select Class</label>
          <select asp-for="ClassId" asp-items="Model.ClassList"
class="form-select"></select>
          <span asp-validation-for="ClassId" class="text-danger"></span>
        </div>

        <div class="mb-3">
          <label class="form-label fw-bold">Select Lecturer</label>
          <select asp-for="LecturerId" asp-items="Model.LecturerList"
class="form-select"></select>
          <span asp-validation-for="LecturerId" class="text-danger"></span>
        </div>

        <button type="submit" class="btn btn-warning w-100">Save Assignment</button>
      </form>
    </div>
  </div>
```

```
</div>
```

7.3. Views/Shared/AccessDenied.cshtml

```
@{ ViewData["Title"] = "Access Denied"; }

<div class="text-center mt-5">
    <h1 class="display-1 text-danger fw-bold">403</h1>
    <h2 class="text-danger">Access Denied</h2>
    <p class="lead">You do not have permission to view this class resource.</p>
    <a href="/" class="btn btn-primary">Go Home</a>
</div>
```