Lab 8 – ASP.Net Core with Razor/MVC- Code First - Contents

Code First Approach with 2 ASP.NET framework : ASP.NET Core Web Application using Razor Pages and ASP.NET Core Web Application with MVC

Create an ASP.NET Core Web Application using Razor Pages with the Code First approach. This approach allows you to define your database schema using C# classes, and Entity Framework Core will generate the database based on these classes.

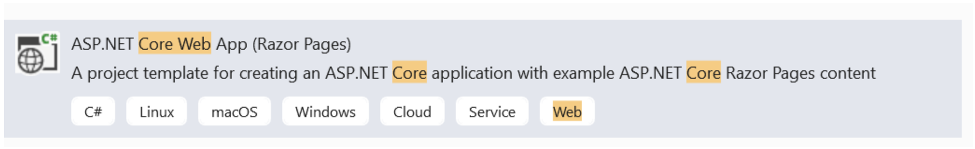
Here are the steps via two exercises for your hand-on practice (Exercise 1 : School App with ASP.Net Core Web App with Razor & Exercise 2 : School App with ASP.Net Core Web App with MVC)

Exercise 1: Setting Up the ASP.NET Core Web Application with Razor

Objective: Create a new ASP.NET Core Web Application with Razor Pages and set up the necessary packages.

Step 1: Setting up an environment

1. **Create a New Project:**
    - Open Visual Studio and create a new ASP.NET Core Web Application project.



ASP.NET Core Web App (Razor Pages)
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content

    - Name your project SchoolAppCoreRazor and choose a location to save it.
    - Select the Web Application template and ensure Razor Pages is selected. Click Create.
2. **Install Entity Framework Core:**
    - Right-click on your project in the Solution Explorer and select Manage NuGet Packages.
    - Search for Microsoft.EntityFrameworkCore.SqlServer and install it.
    - Search for Microsoft.EntityFrameworkCore.Tools and install it.

Step 2: Modify the Menu in _Layout.cshtml

1. **Open the _Layout.cshtml File:**
    - In Visual Studio, **navigate to the Views/Shared folder for an MVC application** or the Pages/Shared folder for a Razor Pages application.
    - Open the _Layout.cshtml file.
2. **Locate the Navigation Section:**
    - Find the section in the _Layout.cshtml file where the navigation menu is defined. This is typically within a <nav> element.
3. **Add Navigation Links:**
    - Add links for Students, Courses, Instructors, Enrollments, and Departments to the navigation menu. Use the asp-area and asp-controller or asp-page attributes to
4. **Example Code for Razor Pages Application:**

```
<nav class="navbar navbar-expand-sm navbar-light bg-light">
    <a class="navbar-brand" asp-page="/Index">SchoolApp</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" asp-page="/Index">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-page="/Students/Index">Students</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-page="/Courses/Index">Courses</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-page="/Instructors/Index">Instructors</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-page="/Enrollments/Index">Enrollments</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-page="/Departments/Index">Departments</a>
            </li>
        </ul>
    </div>
</nav>
```

Step 3: Defining the Data Models

Objective: Create C# classes to define the database schema.

Steps:

1. **Create the Models Folder:**
   - Right-click on your project in the Solution Explorer, select Add > New Folder, and name it Models.
2. **Create the Entity Classes:**
   - Right-click on the Models folder, select Add > Class, and name it Department.cs.
   - Add the following code to the Department.cs file:

```
using System;
using System.Collections.Generic;

namespace SchoolAppCoreRazor.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }
        public required string DepartmentName { get; set; }
        public decimal Budget { get; set; }
        public DateTime StartDate { get; set; }

        public ICollection<Course>? Courses { get; set; }
    }
}
```

- Repeat similar steps for the other entity classes (Instructor, Course, Student, Enrollment).

```
//Course.cs
using System.Collections.Generic;
namespace SchoolAppCoreRazor.Models
{
    public class Course
    {
        public int CourseID { get; set; }
        public required string Title { get; set; }
        public int Credits { get; set; }
        public int DepartmentID { get; set; }
        public Department? Department { get; set; }
        public  ICollection<Enrollment>? Enrollments
{ get; set; }
    }
}
```

```
// Instructor.cs
using System;
using System.Collections.Generic;
namespace SchoolAppCoreRazor.Models
{
    public class Instructor
    {
        public int InstructorID { get; set; }
        public required string LastName { get; set; }
        public required string FirstName { get; set; }
        public DateTime HireDate { get; set; }
    }
}
```

You may add the ErrorViewModel .cs class (located in Models) which is part of an ASP.NET Core Razor Pages or MVC project. Its role is to manage error-related information passed to the view, likely displayed when an error occurs during a user request. The RequestId could assist with debugging, while ShowRequestId determines if the ID should be shown in the error message.

```
namespace SchoolAppCoreRazor.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

```
// Student.cs
using System;
using System.Collections.Generic;
namespace SchoolAppCoreRazor.Models
```

```
//Enrolment.cs
using System;
using System.Collections.Generic;
using System.Linq;
```

```
{
    public class Student
    {
        public int StudentID { get; set; }
        public required string LastName { get; set;
}
        public required string FirstName { get; set;
}
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment>? Enrollments
{ get; set; }
    }
}
```

```
namespace SchoolAppCoreRazor.Models
{
    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public decimal Grade { get; set; }
        public  Course? Course { get; set; }
        public  Student? Student { get; set; }
    }
}
```

Step 4: Creating the DbContext

Objective: Create a DbContext class to manage the database connection and entity sets.

Steps:

1. **Create the SchoolContext Class:**
    - Right-click on the Models folder, select Add > Class, and name it SchoolContext.cs.
    - Add the following code to the SchoolContext.cs file:

```
using Microsoft.EntityFrameworkCore;
namespace SchoolAppCoreRazor.Models
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext>
options) : base(options) { }

        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
    }
}
```

Step 5: Configuring the Database Connection

Objective: Configure the connection to the database using Entity Framework Core.

Steps:

1. **Define the Connection String in appsettings.json:**
   - Open the appsettings.json file in your project.
   - Add the connection string within the ConnectionStrings section:

```
{
  "ConnectionStrings": {
    "SchoolDBConnectionString":
"Server=(local)\\SQLEXPRESS;Database=SchoolDB;Trusted_Connection=Tru
e;TrustServerCertificate=True"
  },
    "Logging": {
      "LogLevel": {
        "Default": "Information",
        "Microsoft.AspNetCore": "Warning"
      }
    },
    "AllowedHosts": "*" }
```

2. **Configure the DbContext in Startup.cs / Program.cs:**
   - Open the Startup.cs / **Program.cs (for .NET 6+)** file

   If Startup.cs then add the following code in the ConfigureServices method:

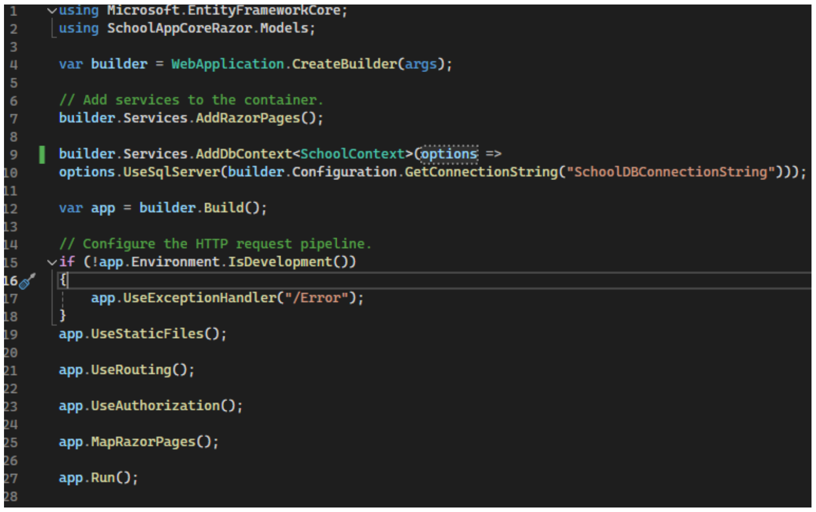
```
public void ConfigureServices(IServiceCollection services){

  services.AddDbContext<SchoolContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("SchoolDBConnectionString")));

  services.AddRazorPages();

}
```

   If no Startup.cs then open **Program.cs (for .NET 6+)** **.** Please add:

```
using Microsoft.EntityFrameworkCore;

using SchoolAppCoreRazor.Models;

...

Builder.Services.AddRazorPages();

builder.Services.AddDbContext<SchoolContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolDBConnection
String")));
```

....

Notes: SchoolDBConnectionString ~ check in **appsettings.json**



```
using Microsoft.EntityFrameworkCore;
using SchoolAppCoreRazor.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

builder.Services.AddDbContext<SchoolContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolDBConnectionString")));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

File Program.cs

Step 6: Creating the Initial Migration and Updating the Database

Objective: Create the initial migration to generate the database schema and update the database.

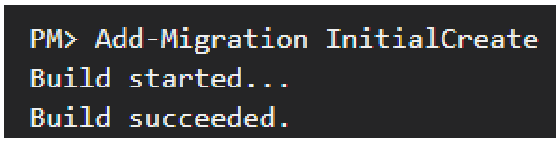
Steps:

1. **Add the Initial Migration:**

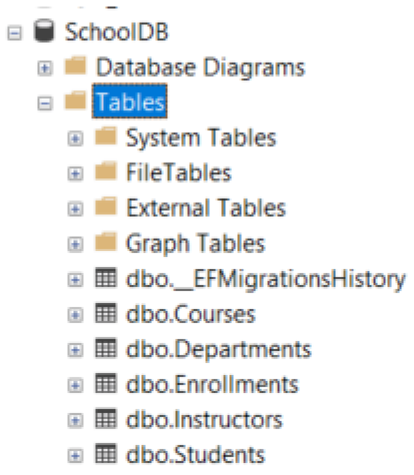
   - Open the Package Manager Console (Tools > NuGet Package Manager > Package Manager Console).

   - Run the following command to add the initial migration:

     Add-Migration InitialCreate



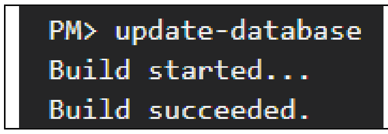
```
PM> Add-Migration InitialCreate
Build started...
Build succeeded.
```



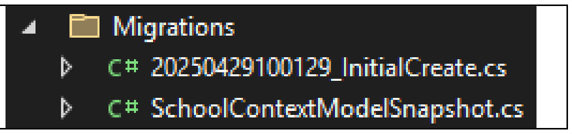
2. **Update the Database:**

   - Run the following command to update the database:

     Update-Database



```
PM> update-database
Build started...
Build succeeded.
```



Migrations
20250429100129_InitialCreate.cs
SchoolContextModelSnapshot.cs

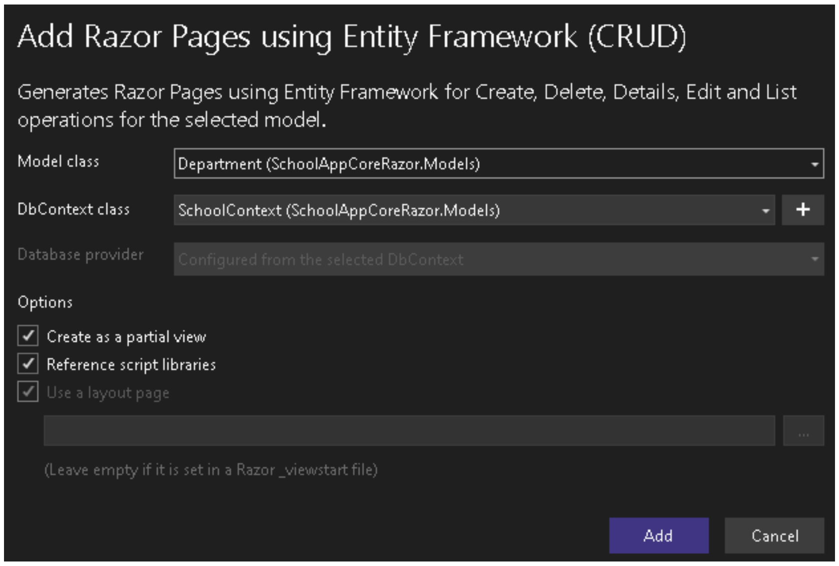
Step 7: Creating Razor Pages

Objective: Create Razor Pages to manage Students, Courses, Instructors, Enrollments, and Departments.

Steps:

1. **Create Razor Pages:**

   - Right-click on the **Pages** folder, select Add > New Folder, and name it Departments.
   - Right-click on the Departments folder, select Add > Razor Page, and choose Razor Pages using Entity Framework (CRUD).
   - Select Department as the model class and SchoolContext as the data context class. Click Add.



Add Razor Pages using Entity Framework (CRUD)

Generates Razor Pages using Entity Framework for Create, Delete, Details, Edit and List operations for the selected model.

Model class: Department (SchoolAppCoreRazor.Models)

DbContext class: SchoolContext (SchoolAppCoreRazor.Models)

Database provider: Configured from the selected DbContext

Options

- [✓] Create as a partial view
- [✓] Reference script libraries
- [✓] Use a layout page

(Leave empty if it is set in a Razor _viewstart file)

   - Repeat the process for Students, Courses, Instructors, and Enrollments.

Step 8: Testing the Application

Objective: Run the application and test the functionality.

Steps:

1. **Run the Application:**

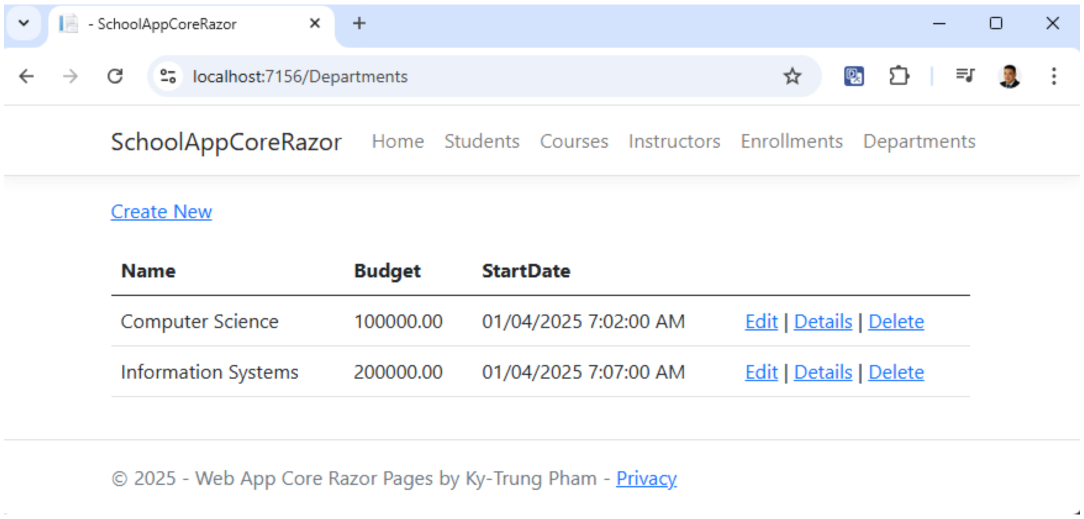
   - Press F5 or click the Start button to run the application.
   - Navigate to the respective URLs (e.g., /Departments, /Students) to access the list of entities.

2. **Verify the Output:**

   - Check the lists displayed on the pages to verify that the data is being retrieved from the database correctly.

3. Amend the web form layout

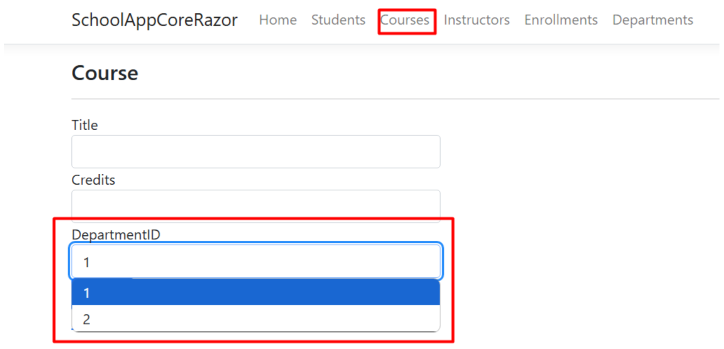
By default, the dropdown in the combo box displays only the **DepartmentID**. This behavior occurs because the current implementation binds the dropdown's display value to the **DepartmentID** field. To provide a more intuitive user experience, the layout of the form can be amended to show both DepartmentID and the corresponding department name in the dropdown.



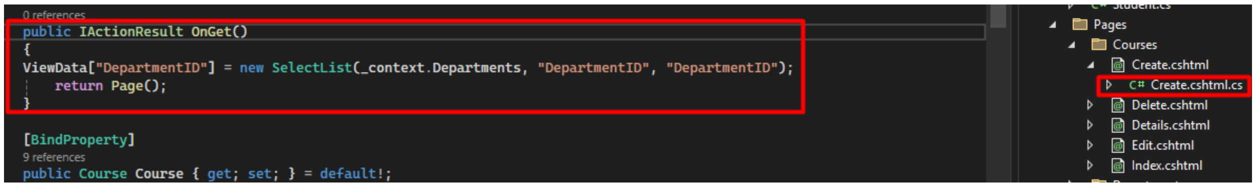
Name	Budget	StartDate	
Computer Science	100000.00	01/04/2025 7:02:00 AM	Edit | Details | Delete
Information Systems	200000.00	01/04/2025 7:07:00 AM	Edit | Details | Delete

© 2025 - Web App Core Razor Pages by Ky-Trung Pham - Privacy

Then click Courses\Create New



By default, the dropdown in the combo box displays only the DepartmentID.



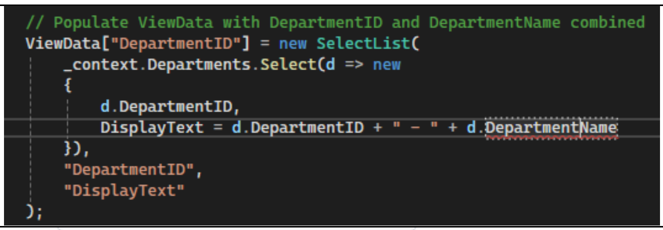
```
public IActionResult OnGet()
{
ViewData["DepartmentID"] = new SelectList(_context.Departments, "DepartmentID", "DepartmentID");
    return Page();
}

[BindProperty]
public Course Course { get; set; } = default!;
```

To show both the DepartmentID and DepartmentName in the dropdown menu, you'll need to make some modifications to your code:

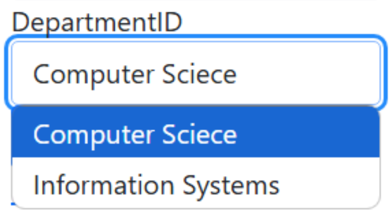
```
public IActionResult OnGet()
 {
 ViewData["DepartmentID"] = new SelectList(_context.Departments, "DepartmentID",
"DepartmentName");
     return Page();
 }
```

Here is some hint to change the combo box dropdown:

```
public IActionResult OnGet()
 {
 ViewData["DepartmentID"] = new
SelectList(_context.Departments,
"DepartmentID", "DepartmentName");
     return Page();
 }
```



```
// Populate ViewData with DepartmentID and DepartmentName combined
ViewData["DepartmentID"] = new SelectList(
    _context.Departments.Select(d => new
    {
        d.DepartmentID,
        DisplayText = d.DepartmentID + " - " + d.DepartmentName
    }),
    "DepartmentID",
    "DisplayText"
);
```

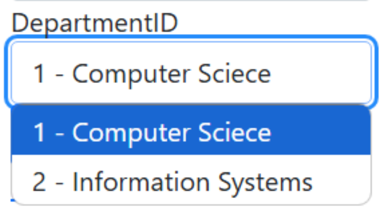


DepartmentID

Computer Sciece

Computer Sciece

Information Systems



DepartmentID

1 - Computer Sciece

1 - Computer Sciece

2 - Information Systems

//Update the code for IActionResult OnGet of Courses' Create.cshtml.cs

```
public IActionResult OnGet()
{
    // Check if the Departments table is null or empty
    if (_context.Departments == null || !_context.Departments.Any())
    {
        // Handle the case where there are no departments
        ViewData["DepartmentID"] = new SelectList(new List<SelectListItem>(), "Value", "Text");
        return Page(); // Optionally, inform the user or display a message
    }
    //ViewData["DepartmentID"] = new SelectList(_context.Departments, "DepartmentID", "Name");
    // Populate ViewData with DepartmentID and DepartmentName combined
    ViewData["DepartmentID"] = new SelectList(
        _context.Departments.Select(d => new
        {
            d.DepartmentID,
            DisplayText = d.DepartmentID + " - " + d.DepartmentName
        }),
        "DepartmentID",
        "DisplayText"
    );
    return Page();
}
```

The updates made to the Department combo box are just an example. You should carefully review the entire form and make any necessary adjustments to enhance usability and create a more user-friendly experience.

Exercise 2: Setting Up the ASP.NET Core Web Application - MVC

By following these exercises, you will learn how to create an ASP.NET Core Web Application with MVC, define your database scheme using the Code First approach, and implement CRUD operations for managing Students, Courses, Instructors, Enrollments, and Departments.

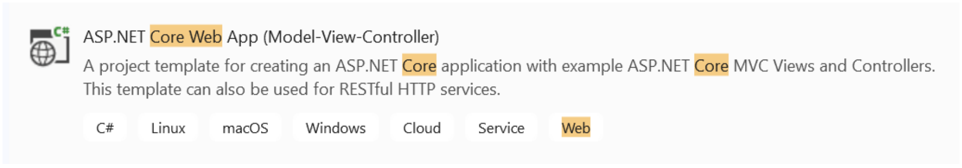
create an ASP.NET Core Web Application using the Model-View-Controller (MVC) pattern with the Code First approach. This approach allows you to define your database schema using C# classes, and Entity Framework Core will generate the database based on these classes. Here are the steps to guide you through the process:

Objective: Create a new ASP.NET Core Web Application with MVC and set up the necessary packages.

Step 1: Setting up an environment

1. **Create a New Project:**

    - Open Visual Studio and create a new ASP.NET Core Web Application project.



ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

    - Name your project SchoolAppCoreMVC and choose a location to save it.
    - Select the Web Application (Model-View-Controller) template and click Create.

2. **Install Entity Framework Core:**

    - Right-click on your project in the Solution Explorer and select Manage NuGet Packages.
    - Search for Microsoft.EntityFrameworkCore.SqlServer and install it.
    - Search for Microsoft.EntityFrameworkCore.Tools and install it.

Step 2: Modify the Menu in _Layout.cshtml

2. **Open the _Layout.cshtml File:**

    - In Visual Studio, navigate to the Views/Shared folder for an MVC application or **the Pages/Shared folder for a Razor Pages application**.
    - Open the _Layout.cshtml file.

3. **Locate the Navigation Section:**

- Find the section in the _Layout.cshtml file where the navigation menu is defined. This is typically within a <nav> element.

4. **Add Navigation Links:**

   - Add links for Students, Courses, Instructors, Enrollments, and Departments to the navigation menu. Use the asp-area and asp-controller or asp-page attributes to generate the correct URLs.

5. **Example Code for MVC Application**

```
<nav class="navbar navbar-expand-sm navbar-light bg-light">
    <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">SchoolApp</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Home" asp-
action="Index">Home</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Students"
asp-action="Index">Students</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Courses" asp-
action="Index">Courses</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Instructors"
asp-action="Index">Instructors</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Enrollments"
asp-action="Index">Enrollments</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" asp-area="" asp-controller="Departments"
asp-action="Index">Departments</a>
            </li>
        </ul>
    </div>
</nav>
```

Step 3: Defining the Data Models

Objective: Create C# classes to define the database schema. You can use the same Models in Exercise 1

Steps:

1. **Create the Models Folder:**
   - Right-click on your project in the Solution Explorer, select Add > New Folder, and name it Models.
2. **Create the Entity Classes:**
   - Right-click on the Models folder, select Add > Class, and name it Department.cs.
   - Add the following code to the Department.cs file:

```
using System;
using System.Collections.Generic;

namespace SchoolAppCoreMVC.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }
        public required string DepartmentName { get; set; }
        public decimal Budget { get; set; }
        public DateTime StartDate { get; set; }

        public ICollection<Course>? Courses { get; set; }
    }
}
```

   - Repeat similar steps for the other entity classes (Instructor, Course, Student, Enrollment).
   - You can reuse the models from Exercise 1, remember to change the namespace.

Step 4: Creating the DbContext

Objective: Create a DbContext class to manage the database connection and entity sets.

Steps:

1. **Create the SchoolContext Class:**
   - Right-click on the Models folder, select Add > Class, and name it SchoolContext.cs.
   - Add the following code to the SchoolContext.cs file:

```
using Microsoft.EntityFrameworkCore;
```

```
namespace SchoolAppCoreMVC.Models
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) :
base(options) { }

        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
    }
}
```

Step 5: Configuring the Database Connection

Objective: Configure the connection to the database using Entity Framework Core.

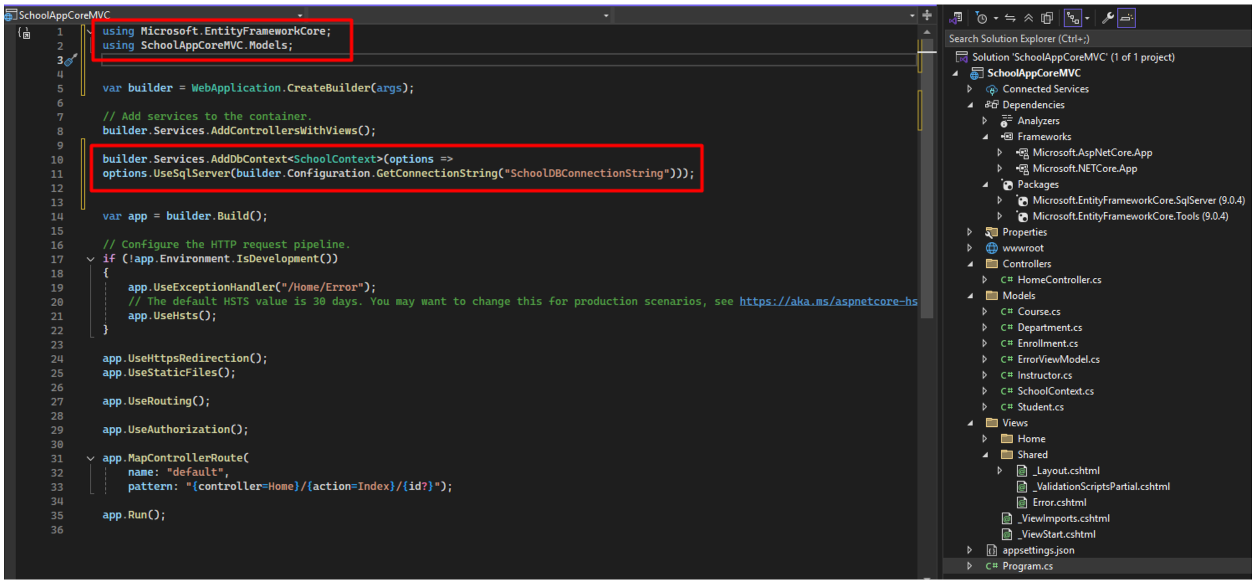
Steps:

1. **Define the Connection String in appsettings.json:**
    - Open the appsettings.json file in your project.
    - Add the connection string within the ConnectionStrings section:

```
{
 "ConnectionStrings": {
   "SchoolDBConnectionString":
"Server=YourServer;Database=SchoolDB;Trusted_Connection=True;
TrustServerCertificate=True "
 },
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft.AspNetCore": "Warning"
  }
 },
 "AllowedHosts": "*"
}
```

2. **Configure the DbContext in Program.cs (<.net 6.0 in Startup.cs):**

```
//Update Program.cs

using Microsoft.EntityFrameworkCore;
using SchoolAppCoreMVC.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<SchoolContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolDBConnecti
onString")));
```



```
SchoolAppCoreMVC
using Microsoft.EntityFrameworkCore;
using SchoolAppCoreMVC.Models;


var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<SchoolContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolDBConnectionString")));


var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hs
    app.UseHsts();
}

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

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

```
Search Solution Explorer (Ctrl+;)
Solution 'SchoolAppCoreMVC' (1 of 1 project)
  SchoolAppCoreMVC
    Connected Services
    Dependencies
      Analyzers
      Frameworks
        Microsoft.AspNetCore.App
        Microsoft.NETCore.App
      Packages
        Microsoft.EntityFrameworkCore.SqlServer (9.0.4)
        Microsoft.EntityFrameworkCore.Tools (9.0.4)
    Properties
    wwwroot
    Controllers
      C# HomeController.cs
    Models
      C# Course.cs
      C# Department.cs
      C# Enrollment.cs
      C# ErrorViewModel.cs
      C# Instructor.cs
      C# SchoolContext.cs
      C# Student.cs
    Views
      Home
      Shared
        _Layout.cshtml
        _ValidationScriptsPartial.cshtml
        Error.cshtml
      _ViewImports.cshtml
      _ViewStart.cshtml
    appsettings.json
    C# Program.cs
```

If .Net Core < 6, then Open the Startup.cs file and add the following code in the ConfigureServices method:

```
public void ConfigureServices(IServiceCollection services)

{

  services.AddDbContext<SchoolContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("SchoolDBConnectionString")));

  services.AddControllersWithViews();

}
```

Step 6: Creating the Initial Migration and Updating the Database

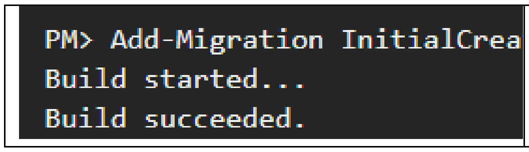
Objective: Create the initial migration to generate the database schema and update the database.

Steps:

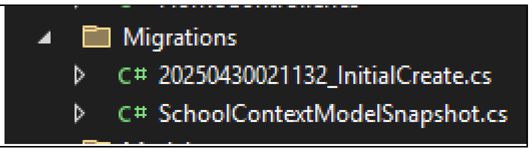
1. **Add the Initial Migration:**

- Open the Package Manager Console (Tools > NuGet Package Manager > Package Manager Console).
- Run the following command to add the initial migration:

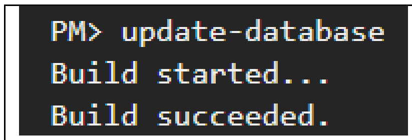
  Add-Migration InitialCreate



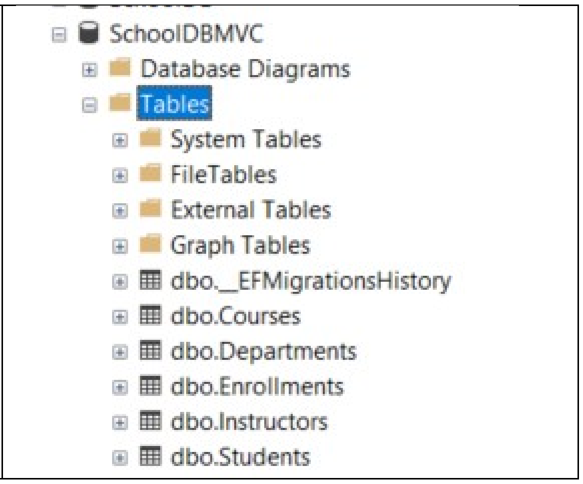
```
PM> Add-Migration InitialCrea
Build started...
Build succeeded.
```


Migrations
C# 20250430021132_InitialCreate.cs
C# SchoolContextModelSnapshot.cs

2. **Update the Database:**
   - Run the following command to update the database:
   - Update-Database



```
PM> update-database
Build started...
Build succeeded.
```

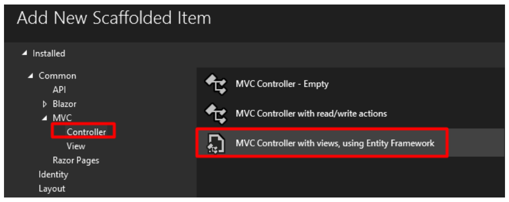


Step 7: Creating Controllers and Views

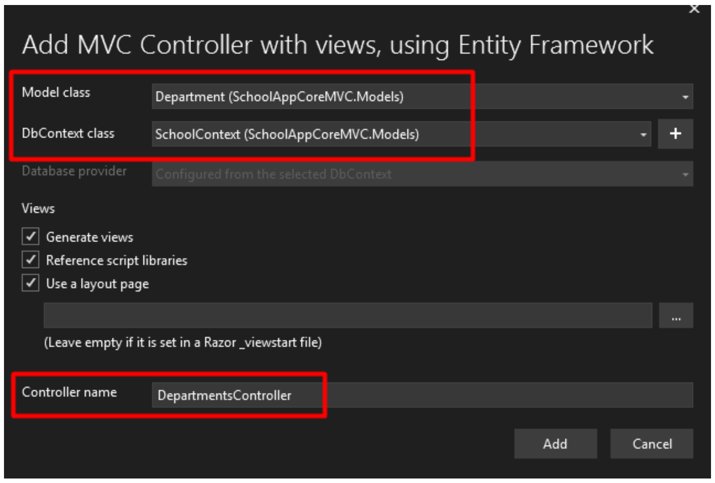
Objective: Create controllers and views to manage Students, Courses, Instructors, Enrollments, and Departments.

Steps:

1. **Create Controllers:**
   - Right-click on the Controllers folder, select Add > Controller, and choose MVC Controller with views, using Entity Framework.



- Then click Add



Add MVC Controller with views, using Entity Framework

Model class: Department (SchoolAppCoreMVC.Models)

DbContext class: SchoolContext (SchoolAppCoreMVC.Models)

Database provider: Configured from the selected DbContext

Views

✓ Generate views

✓ Reference script libraries

✓ Use a layout page

(Leave empty if it is set in a Razor _viewstart file)

Controller name: DepartmentsController

- Select Department as the model class and SchoolContext as the data context class. Check name / Name it DepartmentsController. Click Add.

- Repeat the process for Students, Courses, Instructors, and Enrollments.

2. **Modify the Views:**

   - Open the views for each controller (e.g., Views/Departments/Index.cshtml) and ensure they display the relevant data.

Step 8: Testing the Application

Objective: Run the application and test the functionality.

Steps:

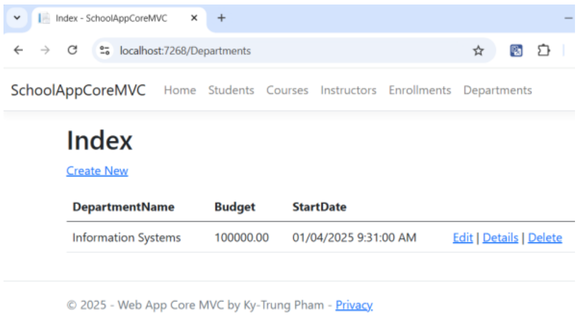
1. **Run the Application:**

   - Press F5 or click the Start button to run the application.

   - Navigate to the respective URLs (e.g., /Departments, /Students) to access the list of entities.

2. **Verify the Output:**

   - Check the lists displayed on the pages to verify that the data is being retrieved from the database correctly.



Index

Create New

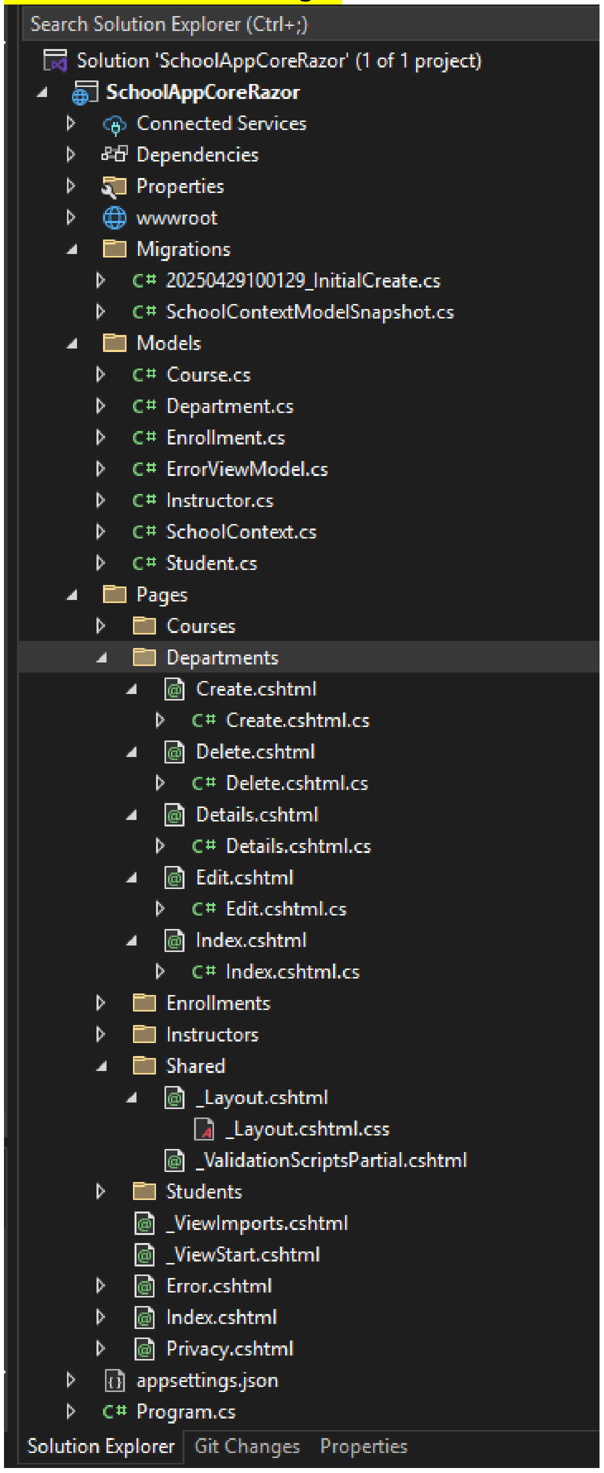
DepartmentName	Budget	StartDate	
Information Systems	100000.00	01/04/2025 9:31:00 AM	Edit | Details | Delete

© 2025 - Web App Core MVC by Ky-Trung Pham - Privacy

ASP.Net Core Razor vs ASP.Net Core MVC

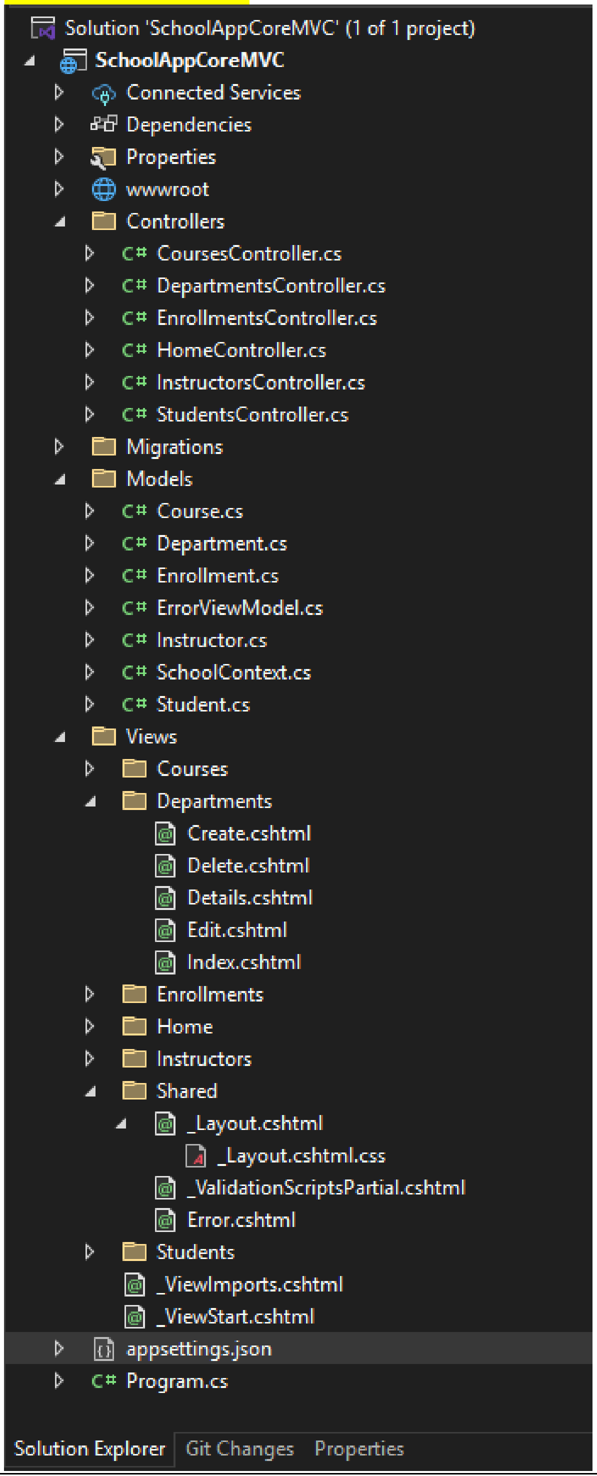
Project Structure

ASP.NET Core Razer Pages



```
Search Solution Explorer (Ctrl+;)
Solution 'SchoolAppCoreRazor' (1 of 1 project)
  SchoolAppCoreRazor
    Connected Services
    Dependencies
    Properties
    wwwroot
    Migrations
      20250429100129_InitialCreate.cs
      SchoolContextModelSnapshot.cs
    Models
      Course.cs
      Department.cs
      Enrollment.cs
      ErrorViewModel.cs
      Instructor.cs
      SchoolContext.cs
      Student.cs
    Pages
      Courses
      Departments
        Create.cshtml
          Create.cshtml.cs
        Delete.cshtml
          Delete.cshtml.cs
        Details.cshtml
          Details.cshtml.cs
        Edit.cshtml
          Edit.cshtml.cs
        Index.cshtml
          Index.cshtml.cs
      Enrollments
      Instructors
      Shared
        _Layout.cshtml
          _Layout.cshtml.css
        _ValidationScriptsPartial.cshtml
      Students
      _ViewImports.cshtml
      _ViewStart.cshtml
      Error.cshtml
      Index.cshtml
      Privacy.cshtml
    appsettings.json
    Program.cs
Solution Explorer   Git Changes   Properties
```

ASP.NET Core MVC



```
Solution 'SchoolAppCoreMVC' (1 of 1 project)
  SchoolAppCoreMVC
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      CoursesController.cs
      DepartmentsController.cs
      EnrollmentsController.cs
      HomeController.cs
      InstructorsController.cs
      StudentsController.cs
    Migrations
    Models
      Course.cs
      Department.cs
      Enrollment.cs
      ErrorViewModel.cs
      Instructor.cs
      SchoolContext.cs
      Student.cs
    Views
      Courses
      Departments
        Create.cshtml
        Delete.cshtml
        Details.cshtml
        Edit.cshtml
        Index.cshtml
      Enrollments
      Home
      Instructors
      Shared
        _Layout.cshtml
          _Layout.cshtml.css
        _ValidationScriptsPartial.cshtml
        Error.cshtml
      Students
      _ViewImports.cshtml
      _ViewStart.cshtml
    appsettings.json
    Program.cs
Solution Explorer   Git Changes   Properties
```

ASP.NET Core Razor Pages

Structure:

- **Page Model (.cshtml.cs):** Contains the page's logic and handles requests.
- **Razor Page (.cshtml):** Contains the HTML and Razor markup.

Use Cases:

- Suitable for applications with simple page-based navigation.
- Ideal for small to medium-sized applications where rapid development is needed.
- Good for scenarios where the page-centric approach is more intuitive.

Benefits:

- **Simplicity:** Easier to set up and manage, with less boilerplate code.
- **Tight Integration:** The page model and view are closely integrated, making it easier to manage.
- **Rapid Development:** Faster to develop for simple CRUD operations.

Drawbacks:

- **Limited Flexibility:** Less control over the URL structure and routing compared to MVC.
- **Scalability:** May not be as suitable for very large applications with complex interactions.

ASP.NET Core MVC

Structure:

- **Controllers:** Handle user input, work with the model, and select a view to render.
- **Views:** Display the user interface.
- **Models:** Represent the data and business logic.

Use Cases:

- Suitable for applications with complex user interactions and multiple views.
- Ideal for large-scale applications where separation of concerns is crucial.
- Good for applications that require extensive use of routing and URL manipulation.

Benefits:

- **Separation of Concerns:** Clear separation between the controller, view, and model.
- **Testability:** Easier to unit test due to the separation of concerns.
- **Flexibility:** More control over the HTML and URL structure.

Drawbacks:

- **Complexity:** Can be more complex to set up and manage, especially for small applications.
- **Overhead:** More boilerplate code compared to Razor Pages.

Comparison Summary

Feature	**ASP.NET Core MVC**	**ASP.NET Core Razor Pages**
Structure	Controllers, Views, Models	Page Model (.cshtml.cs), Razor Page (.cshtml)
Use Cases	Complex interactions, large-scale apps	Simple page-based navigation, small to medium apps
Benefits	Separation of concerns, testability, flexibility	Simplicity, tight integration, rapid development
Drawbacks	Complexity, overhead	Limited flexibility, scalability issues

Example Scenarios

ASP.NET Core Razor Pages

- A personal blog or portfolio website with straightforward navigation.
- An internal tool for managing employee records with basic CRUD operations

ASP.NET Core MVC:

- An e-commerce platform with multiple product categories, user authentication, and complex business logic.
- A content management system (CMS) with various user roles and permissions.

Exercise 3: Upload Projects to GitHub

Demonstrate how to use GitHub with Visual Studio to upload and organize Exercise 1 (Razor Pages) and Exercise 2 (MVC) projects. Steps:

1. **Open the Project**
   - Launch Visual Studio and open your **SchoolAppCoreRazor** or **SchoolAppCoreMVC** project.
2. **Commit Changes**
   - Use the **Git Changes** panel to stage your files for commit.
   - Enter a commit message, such as "Initial commit for Exercise 1/2," and click **Commit All**.
3. **Push to GitHub**
   - Publish the project to your GitHub repository. If you already have repositories set up:
     - Select the repository you wish to push to.
     - Push changes to the main branch.
4. **Organize Projects**
   - In your solution explorer, create folders named **Exercise1_RazorPages** and **Exercise2_MVC**.
   - Move each project into its respective folder.
   - Stage and commit these changes with a message like "Organized projects into folders," and push to GitHub.
5. **Update README**
   - Add a `README.md` file with instructions and descriptions for both exercises. Example content:

```
o # School App Projects
o ## Exercise 1: Razor Pages Application
o - Description: ASP.NET Core Razor Pages project using Code First
  approach.
o - Instructions: Open in Visual Studio, press F5 to run.
o ## Exercise 2: MVC Application
o - Description: ASP.NET Core MVC project using Code First
  approach.
o - Instructions: Open in Visual Studio, press F5 to run.
```

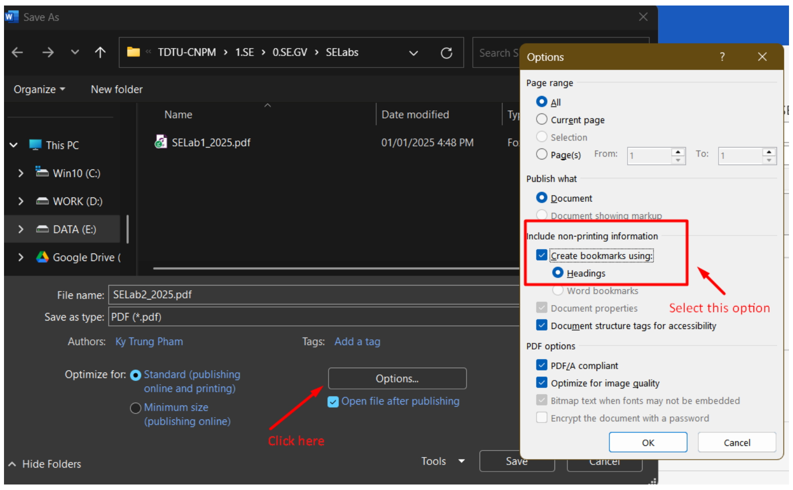
6. **Verify Changes on GitHub**
   - Check your repository to ensure both projects are uploaded correctly and organized in separate folders.

Submit on eLearning: Summary Report

- Introduction: Briefly describe the purpose of the exercises.
- Exercises: List each exercise with a brief description of what you did and learned.
- Screenshots: Include the screenshots of every step for each exercise and steps (both Visual Studio and Github) (Index each step with sequential number) and the output when press F5.
- Conclusion: Summarize your overall learning experience and any challenges you faced.

Submit 2 files on eLearning:

- StudentID_StudentName.zip (contain Windows Web Project & .sql file) for both Class Activity and Homework.
- StudentID_StudentName.pdf (require creating bookmarks) includes GitHub link as Appendix



SELab1_2025.pdf 01/01/2025 4:48 PM

File name: SELab2_2025.pdf
Save as type: PDF (*.pdf)
Authors: Ky Trung Pham Tags: Add a tag
Optimize for: [✓ Standard (publishing online and printing)] Minimum size (publishing online)
Options...
[✓] Open file after publishing

Page range
[✓ All] Current page Selection Page(s) From: 1 To: 1

Publish what
[✓ Document] Document showing markup

Include non-printing information
[✓] Create bookmarks using:
[✓ Headings] Word bookmarks
[✓] Document properties
[✓] Document structure tags for accessibility

PDF options
[✓] PDF/A compliant
[✓] Optimize for image quality
[✓] Bitmap text when fonts may not be embedded
[ ] Encrypt the document with a password

Notes: There may be a typo/mistake during creating this document. Please correct it by yourself.

References:

Microsoft Learn - ASP.Net Web Core MVC can be navigated:

- Tutorial: Get started with Razor Pages in ASP.NET Core | Microsoft Learn
- Tutorial: Get started with EF Core in an ASP.NET MVC web app | Microsoft Learn
- ASP.NET Core Razor Pages Application - Dot Net Tutorials

MVC Build a complete Ecommerce App with Asp.net Core MVC (Full Series of Step-by-Step): https://www.youtube.com/watch?v=QBkA5_DaasQ&list=PL2Q8rFbm-4ruTcZY39MNOsEu4p76HQ5VX .

- A demo can be watched here : https://youtu.be/cnqi91ugnVQ .

@TDTUers: Email: tg_phamthaikytrung@tdtu.edu.vn . @Others : Email / MS Team: trung.phamthaiky@outlook.com