**Building Websites Using ASP.NET Core Razor Pages**

# Introduction

Imagine you're an employee of IT department in a University. Your manager has asked you to develop a web application for student, course and enrollment management

Course(CourseID, Title, Credits, CourseID)

Enrollment(EnrollmentID, CourseID, StudentID, Grade)

Student(ID IDENTITY(1,1), LastName, FirstMidName, EnrollmentDate)

The application has to support adding, viewing, modifying, and removing students—a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating an application using Razor Pages, ASP.NET Core, and C#. An **SQL Server** **Database** will be created to persist the product data that will be used for reading and managing product data by **Entity Framework Core.**

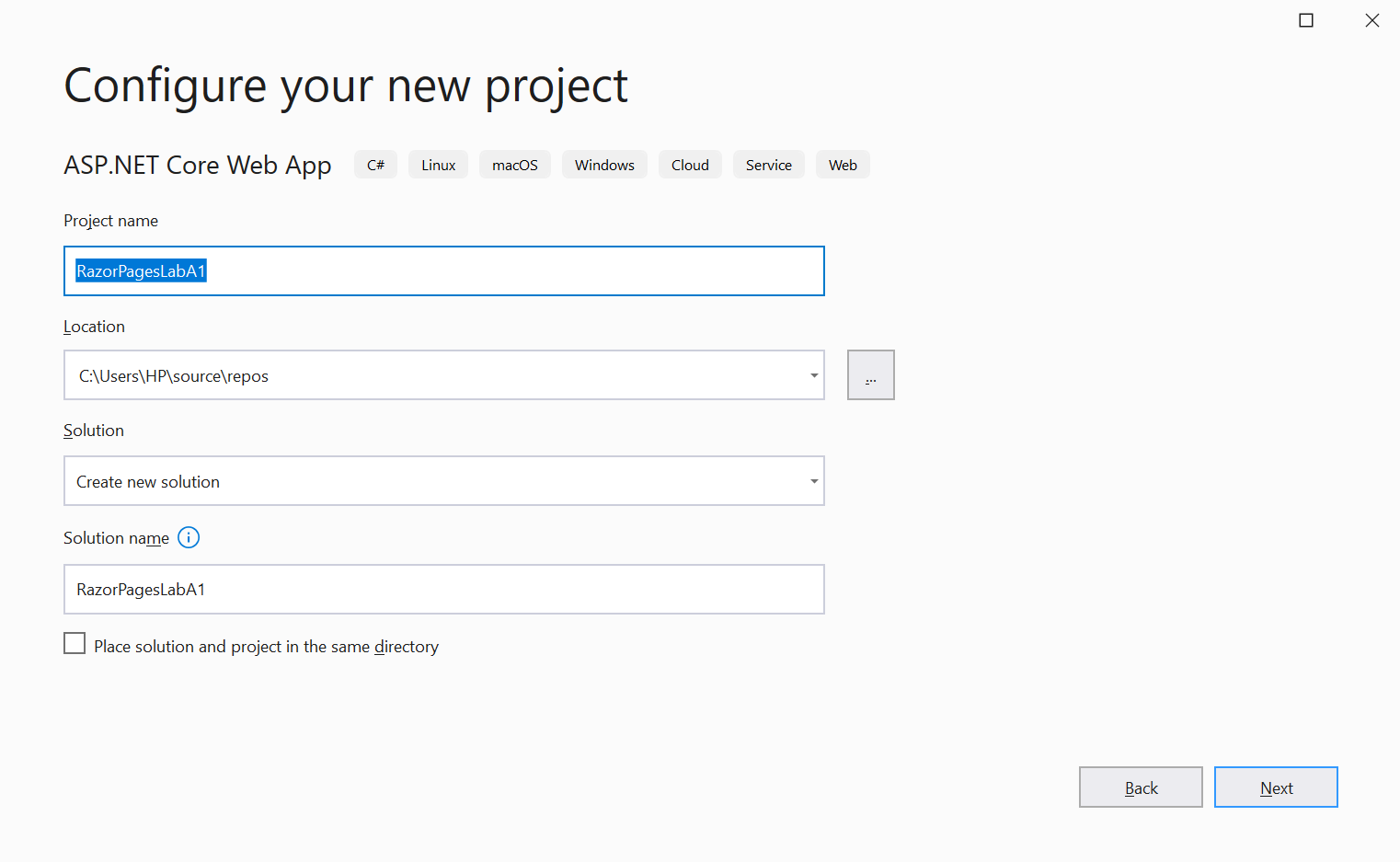
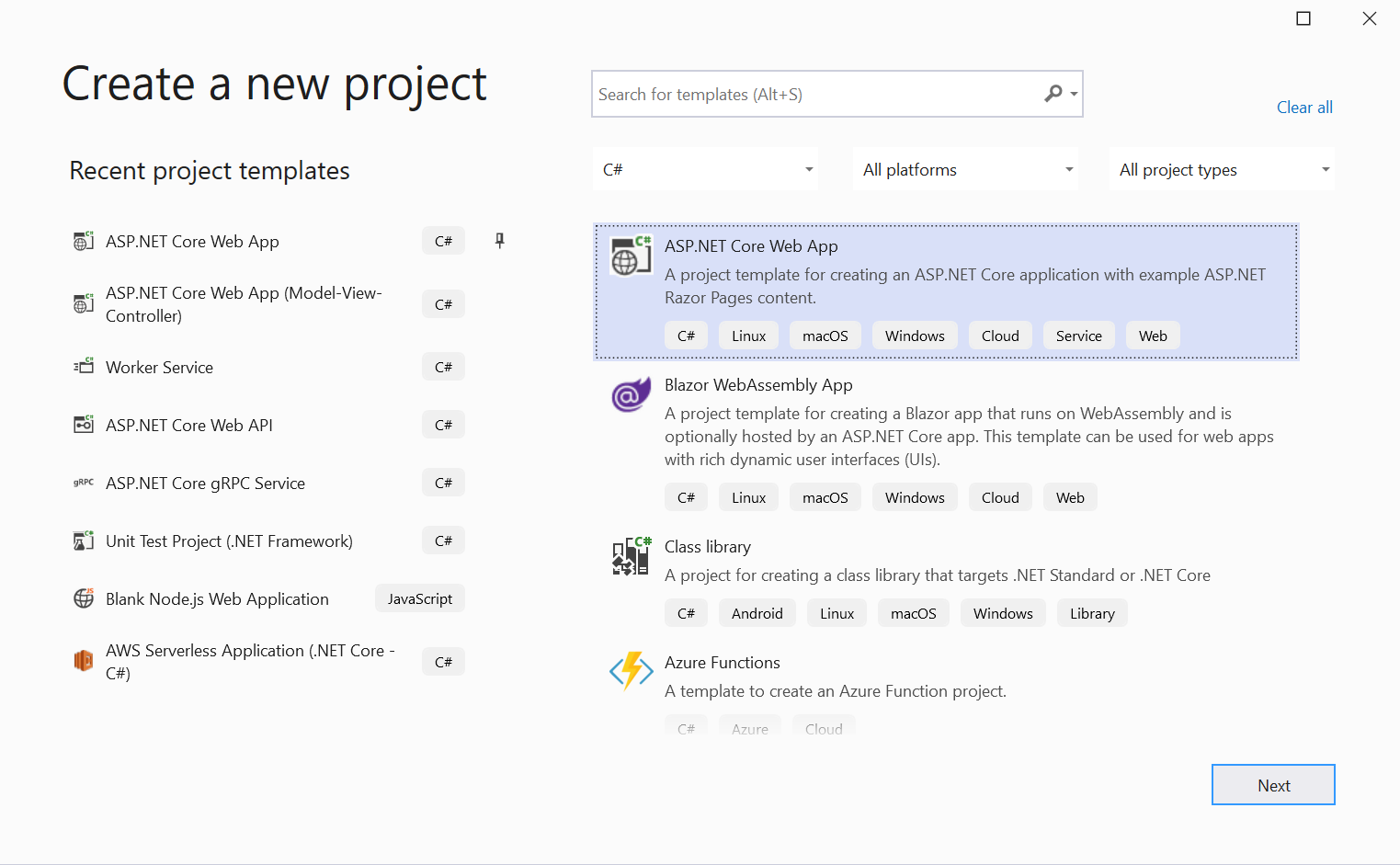
# Lab Objectives

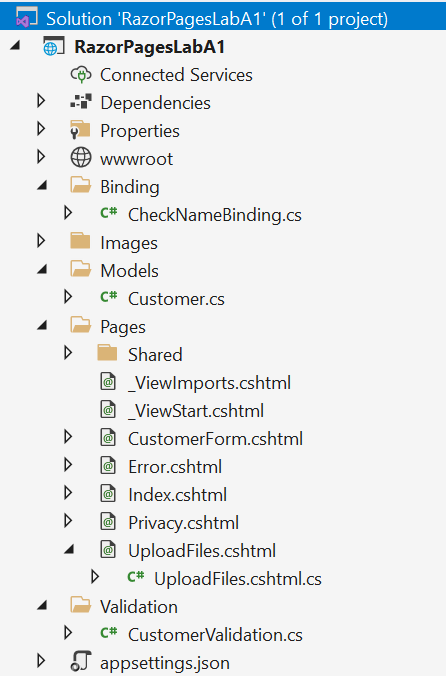
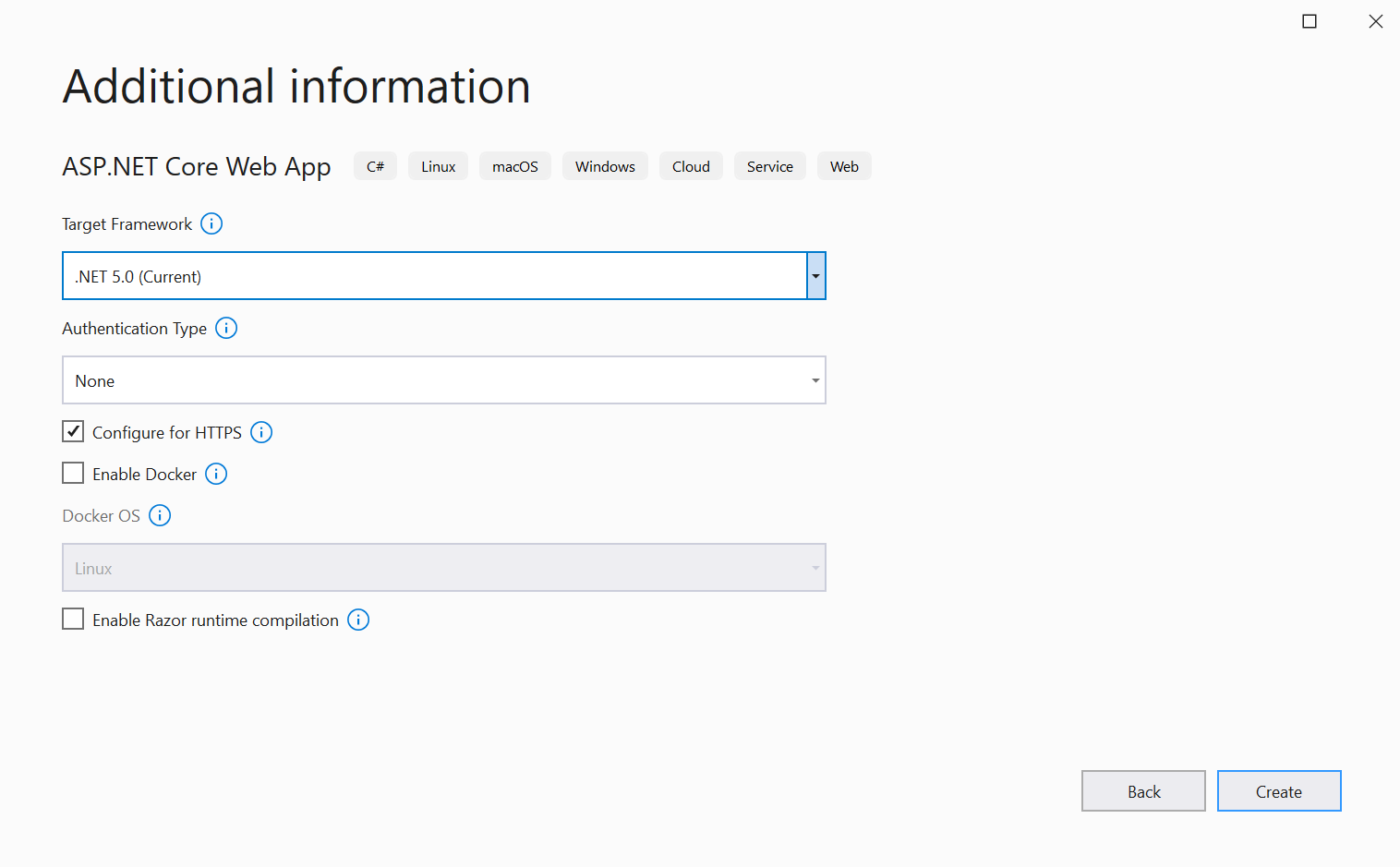
In this lab, you will:

* Use the Visual Studio.NET to create ASP.NET Core Web Application Project.
* Develop application using Razor Pages.
* Apply annotations and custom validation for validating input data.
* Use Entity Framework to Create a SQL Server database named SchoolContextDB that has three tables: Student, Course, Enrollment.
* Develop Entity classes a and DBContext class to perform CRUD actions using Razor Pages.
* Create Search and Paging functions for web application.
* Run the project and test the application actions.

# Activity 01: Simple Razor Pages with Validation and File(s) uploading

**Step 01**. Create ASP.NET Core Web App (A project template for creating an ASP.NET application with example ASP.NET Razor Pages content.)





4

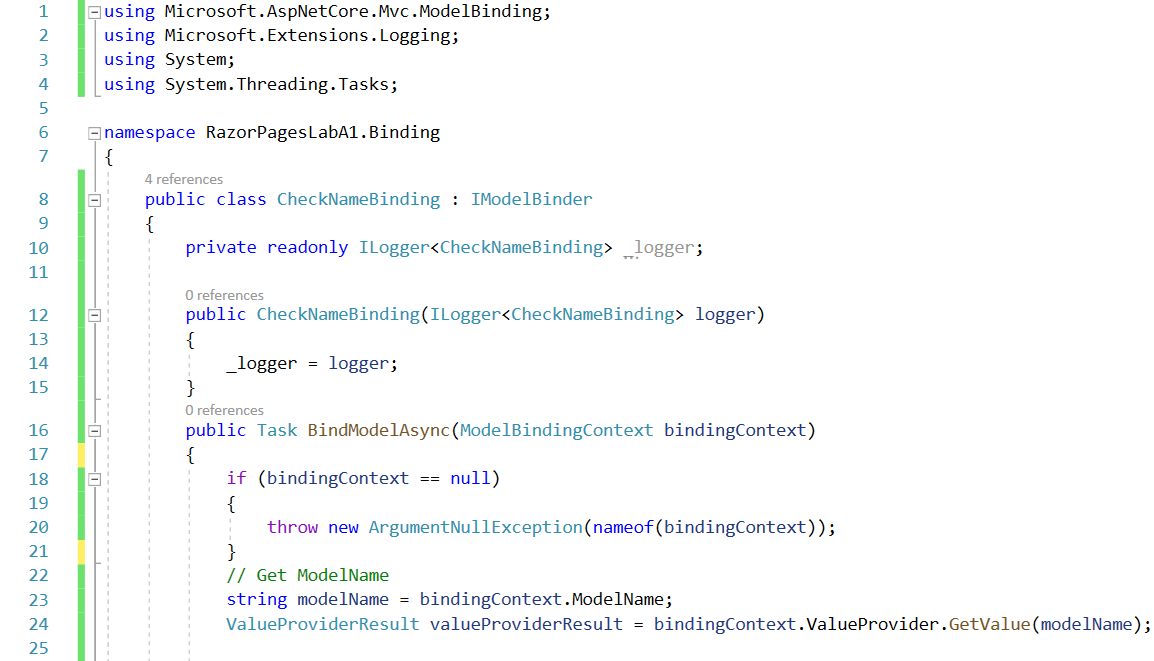
3

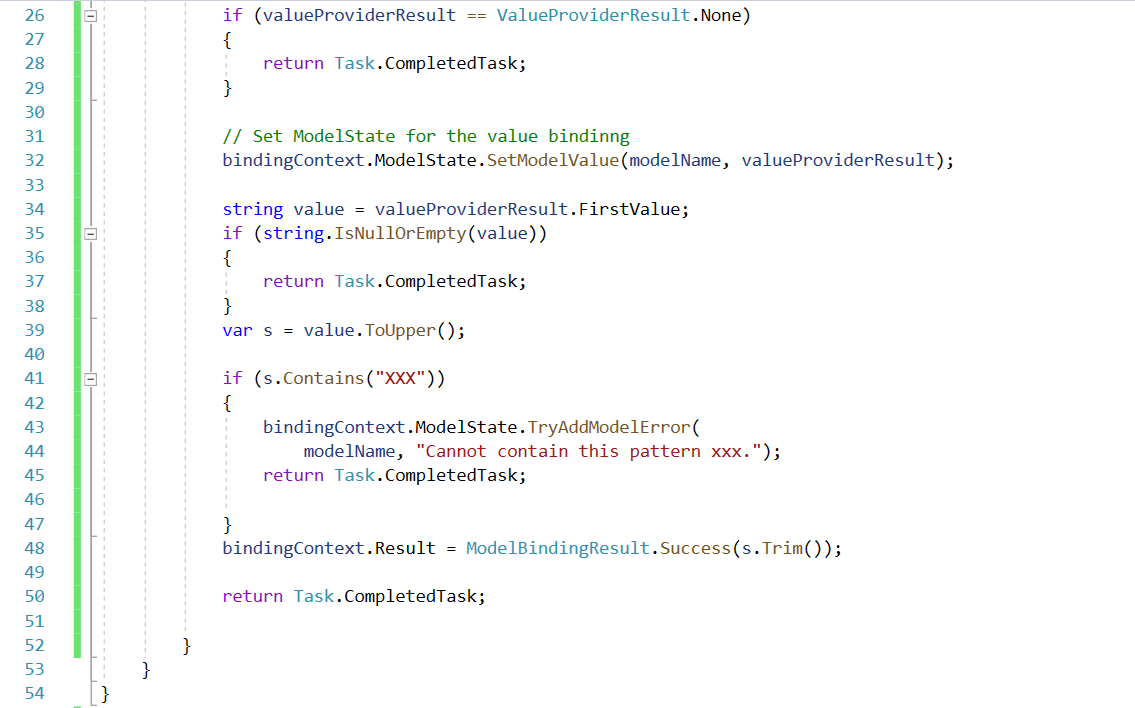
3

1

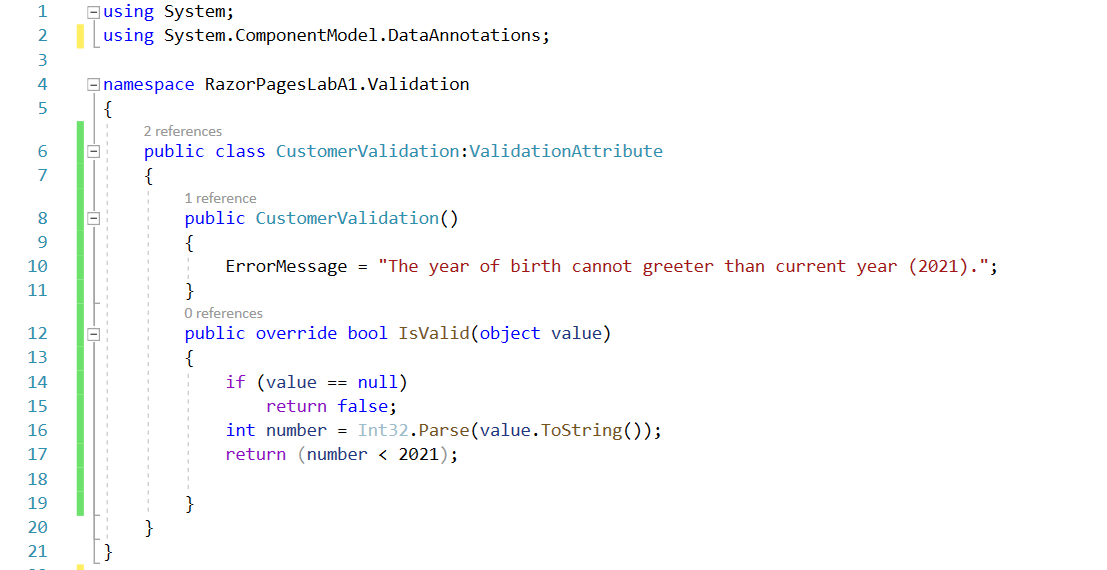
**Step 02.** Using Model Binding in Razor Pages to takes values from HTTP requests and maps them to handler method parameters or PageModel properties.

Create class “CheckNameBinding” implements IModelBinder interface.





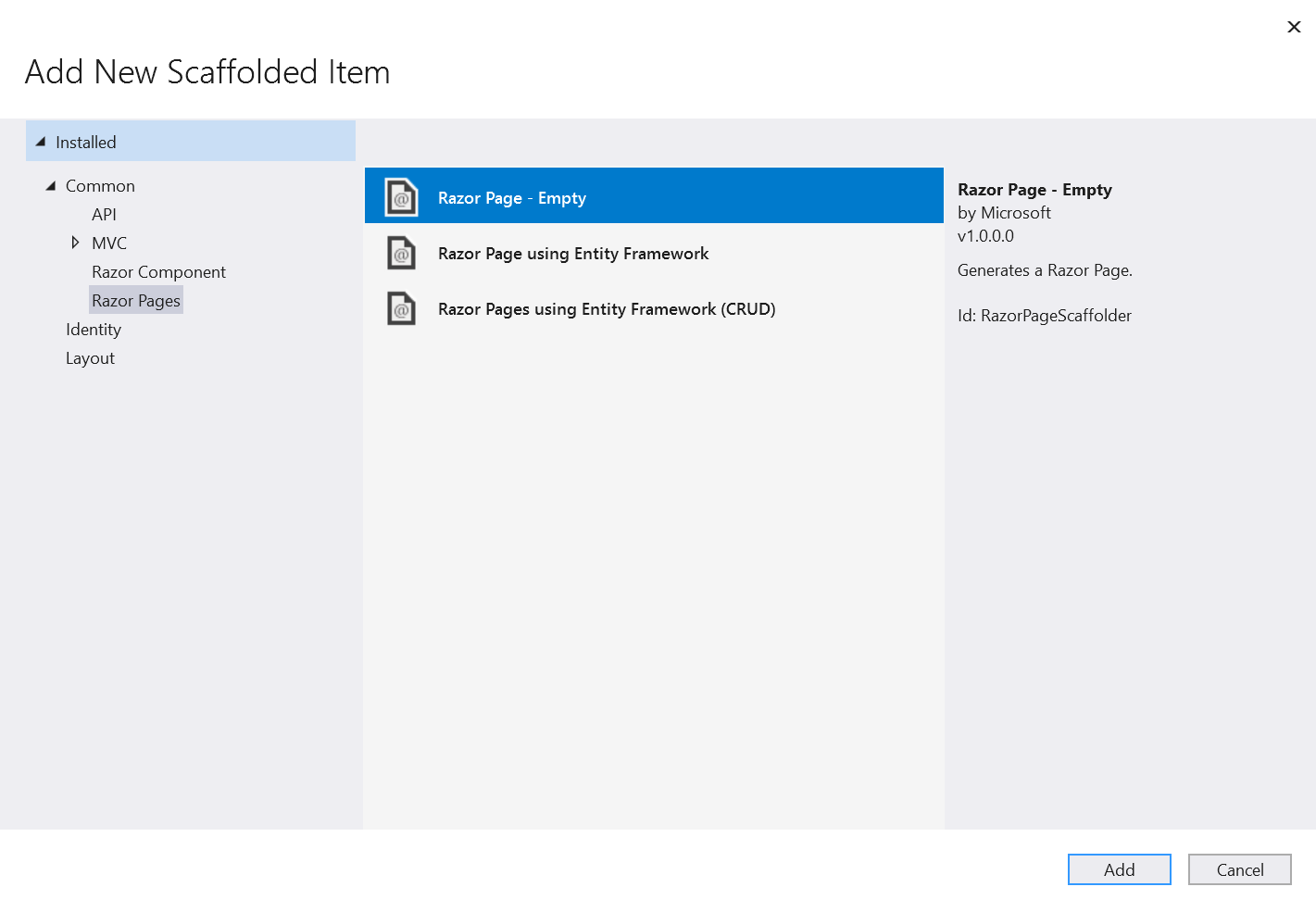
**Step 03**. Create custom validation class

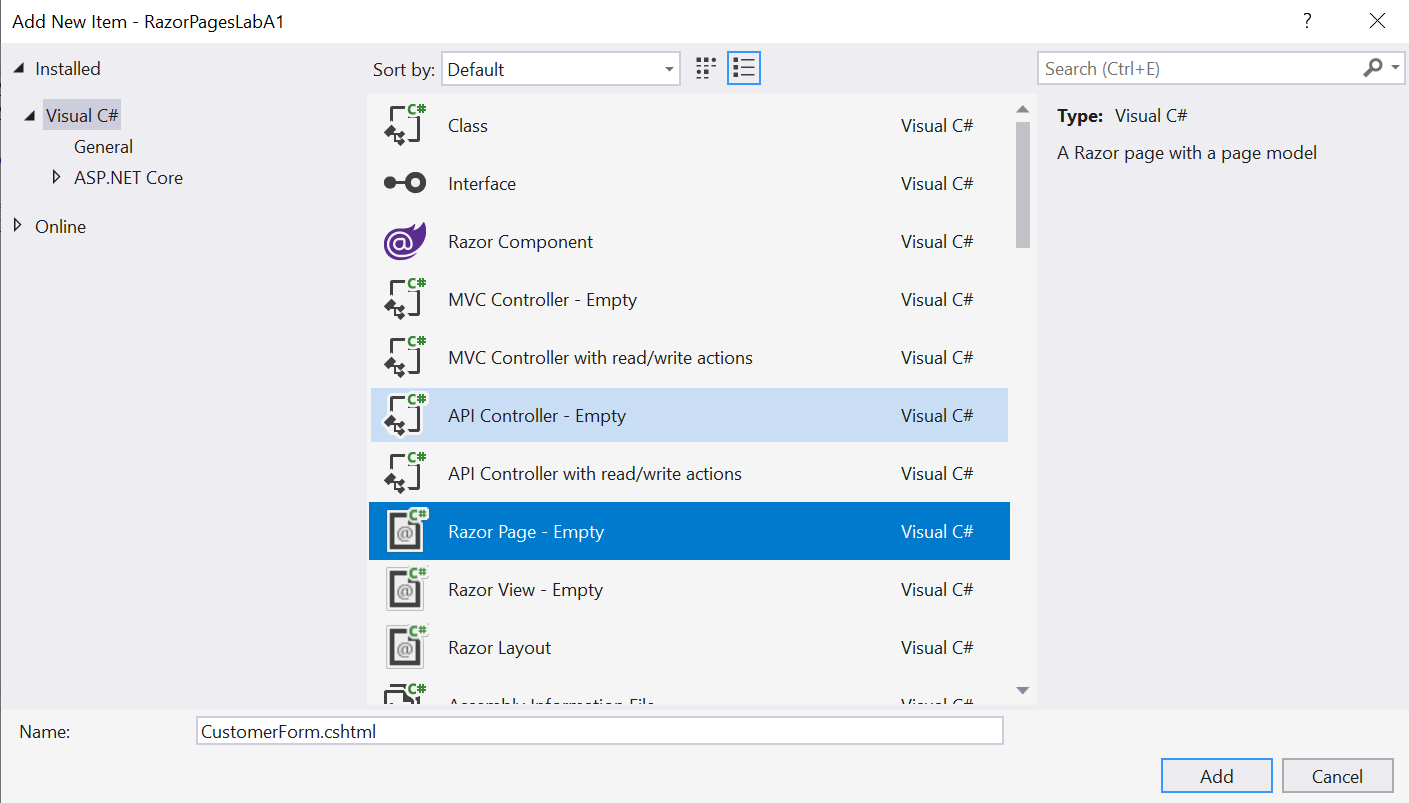


**Step 04**. Create a Model using DataAnnotations and custom validation.



**Step 05**. Create Razor Page (Empty) for form validation

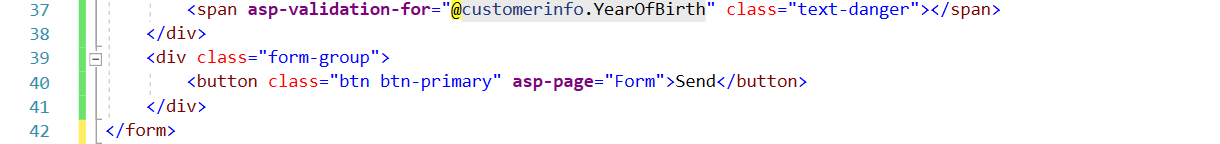


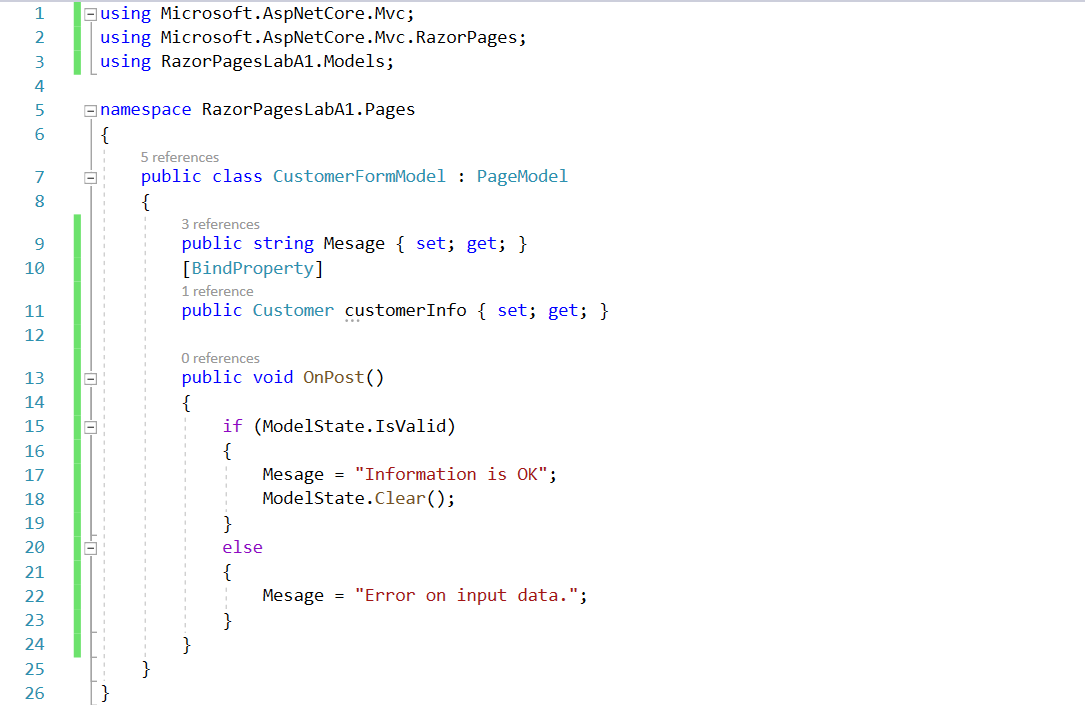


Change information for viewing form (CustomerForm.cshtml)

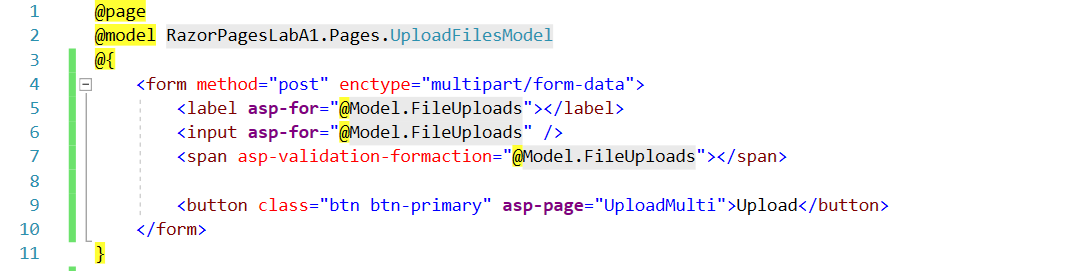


Message form Model

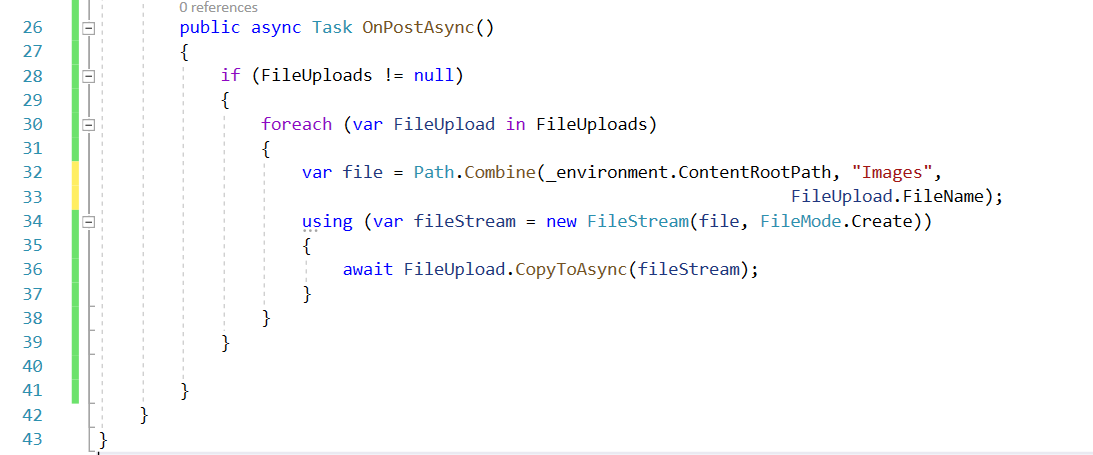


Change the code in CustomerForm.cshtml.cs

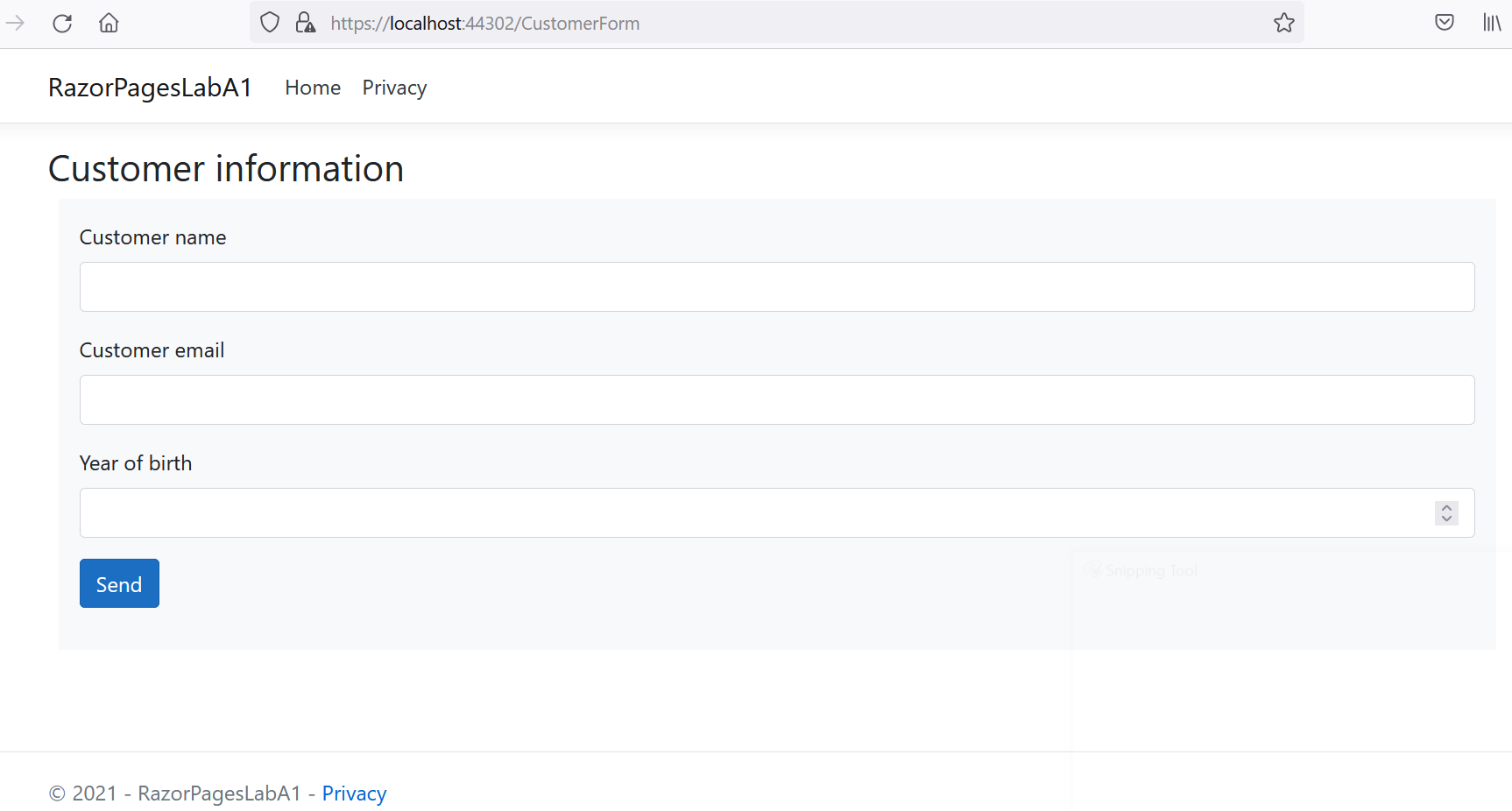
**Step 06**. Create Razor Page (Empty) for uploading files

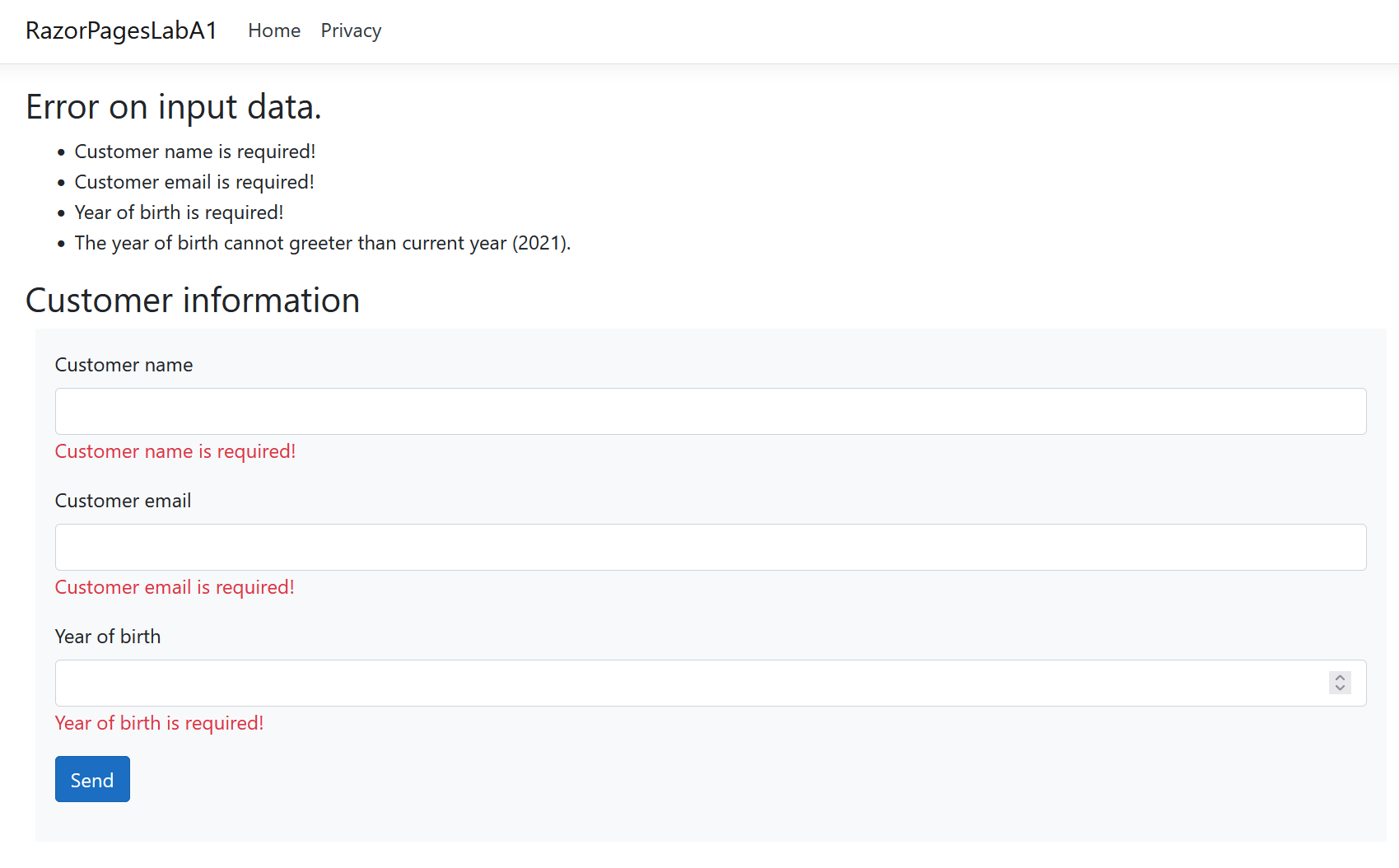






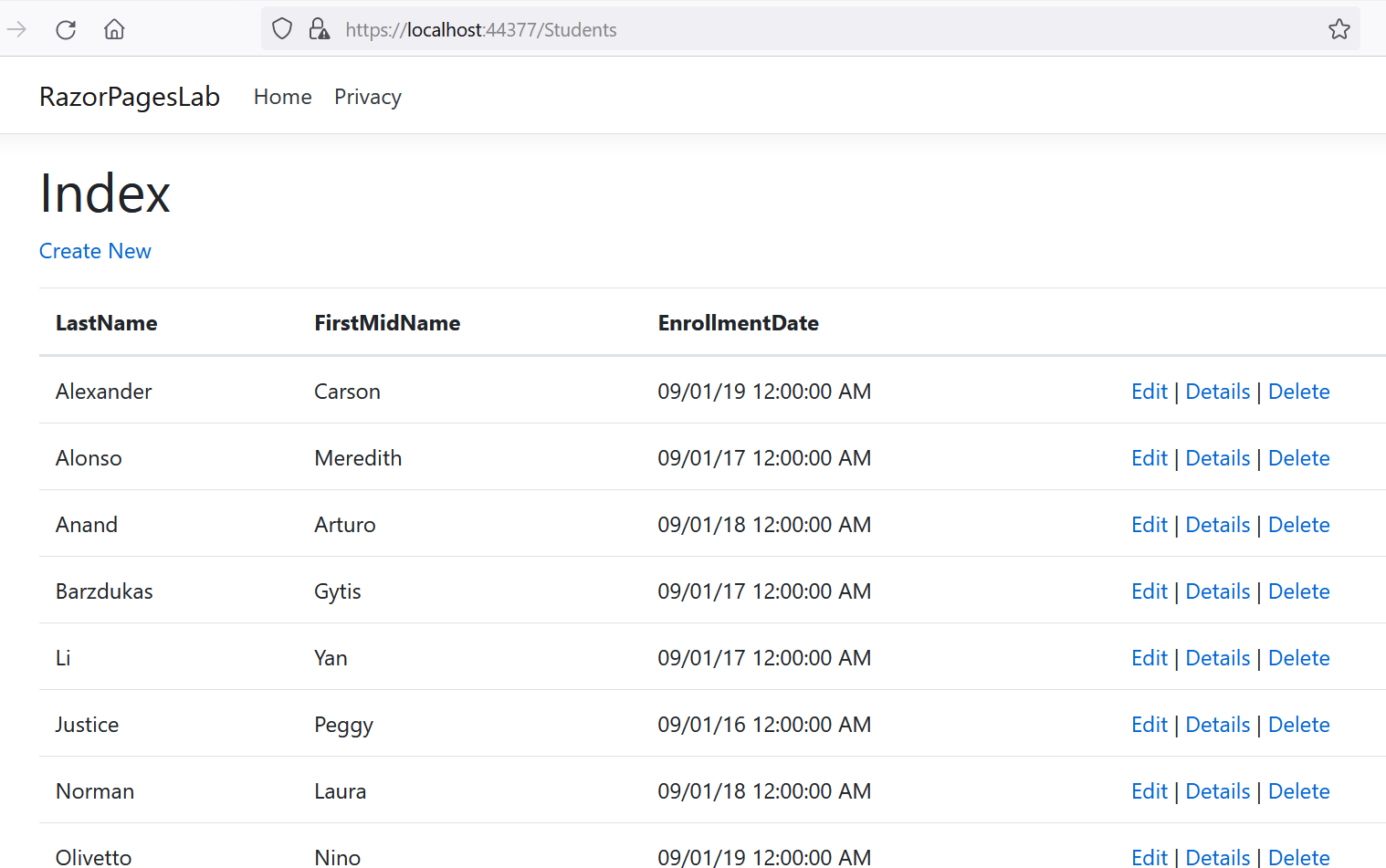
**Step 07**. Build and run Project.



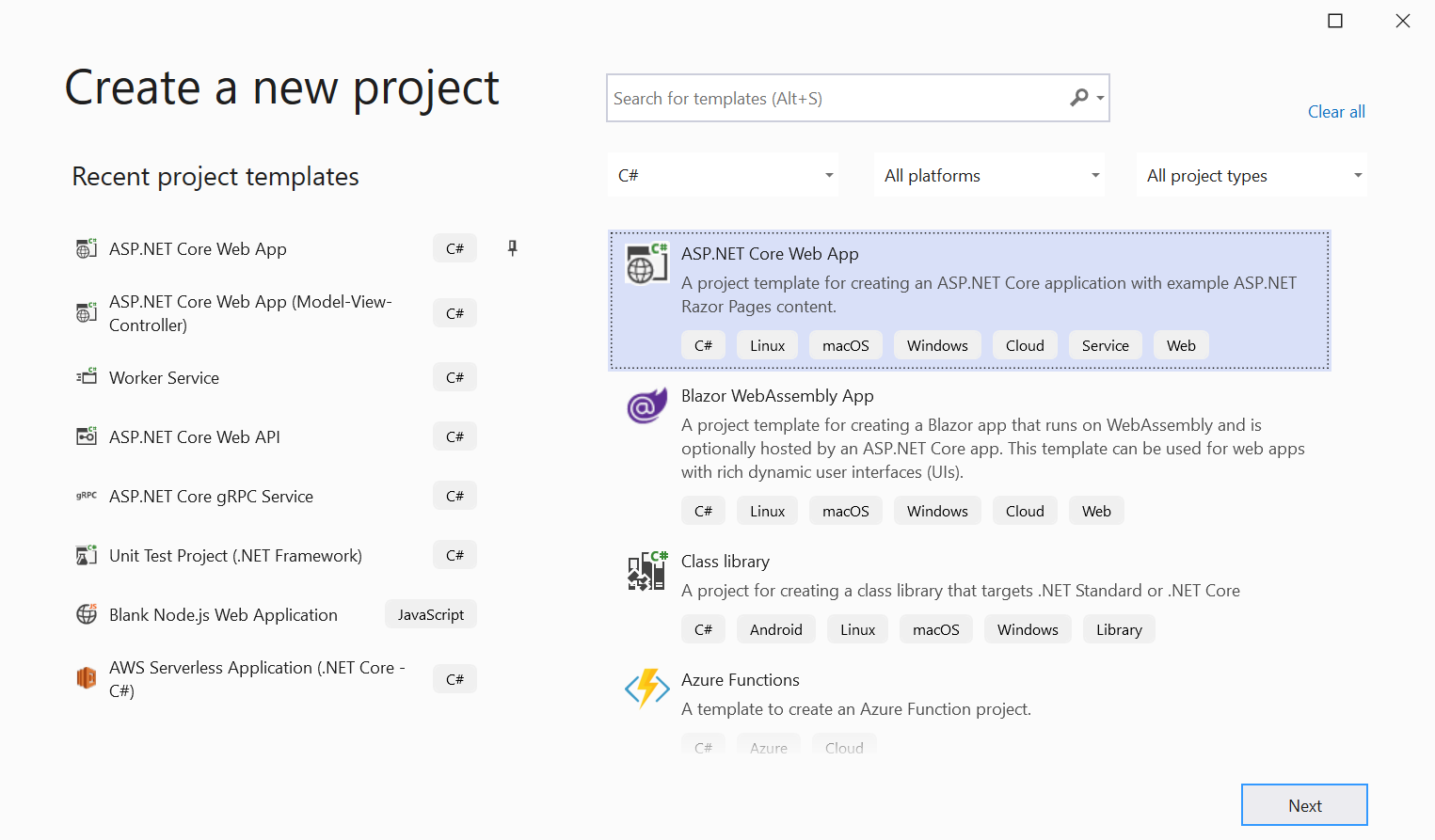


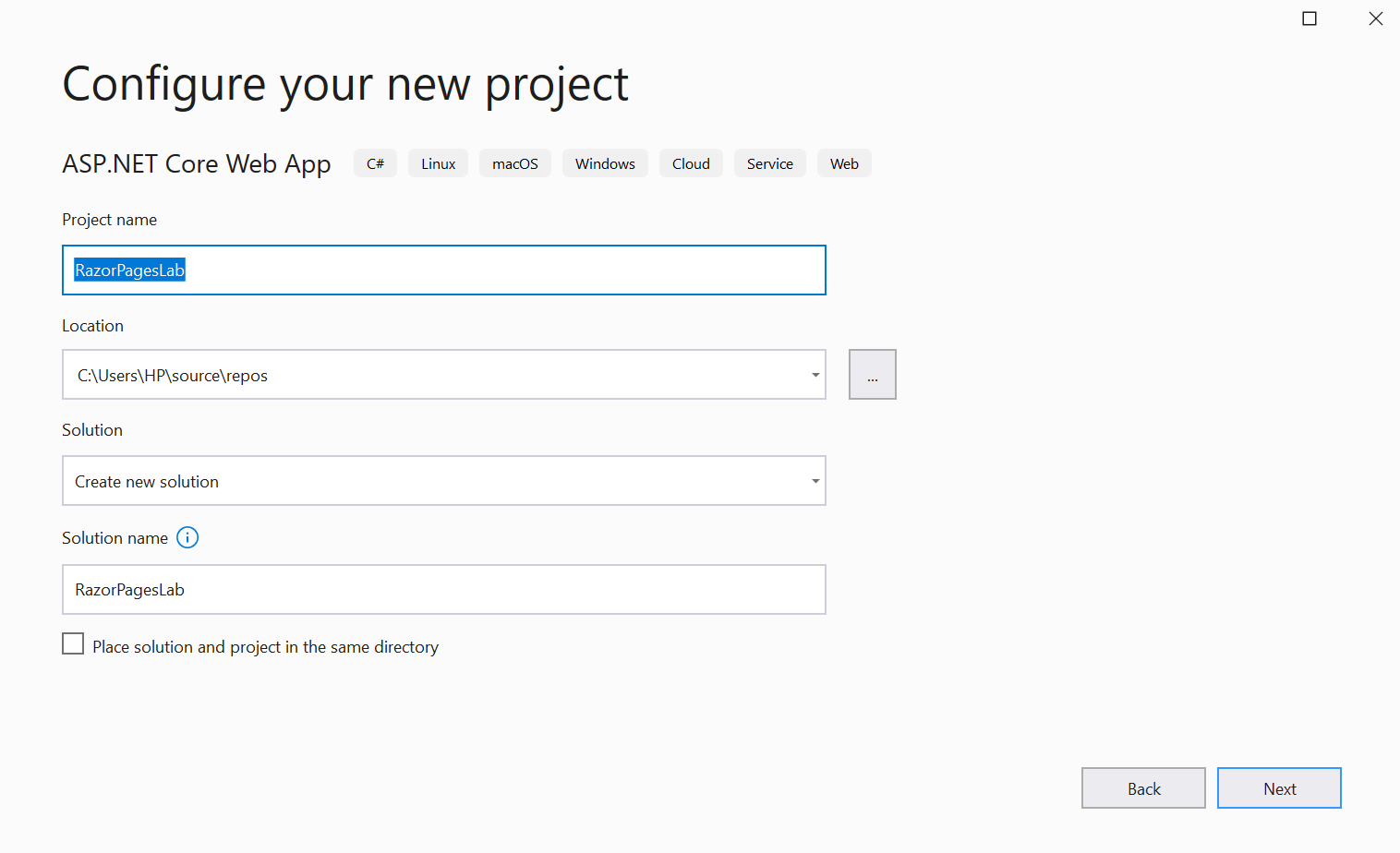


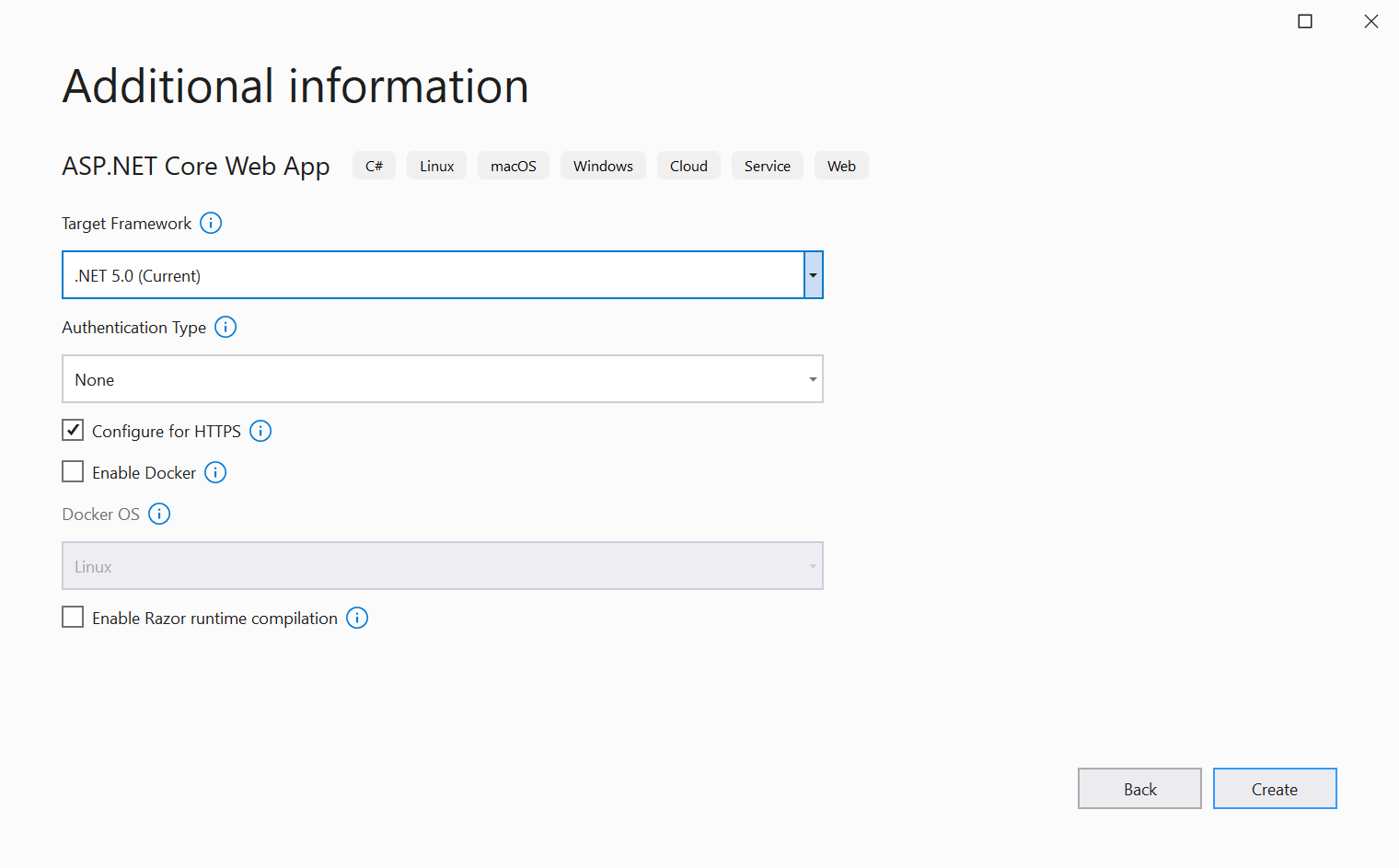
# Activity 02: Razor Pages with Entity Framework CRUD

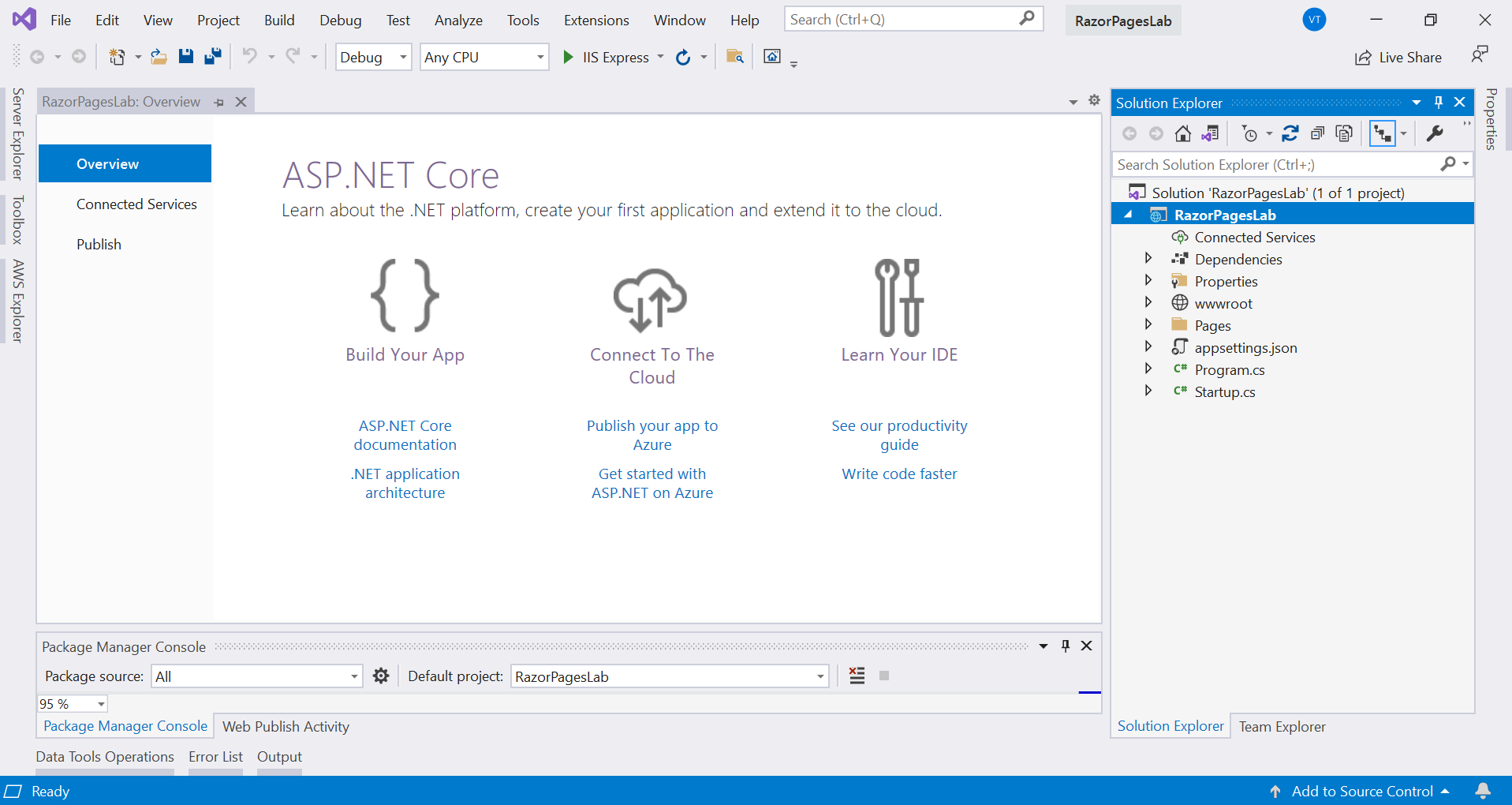


**Step 01.** Create ASP.NET Core Web Application



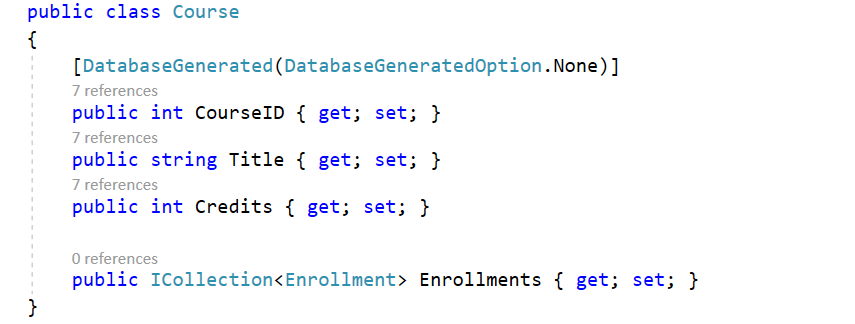


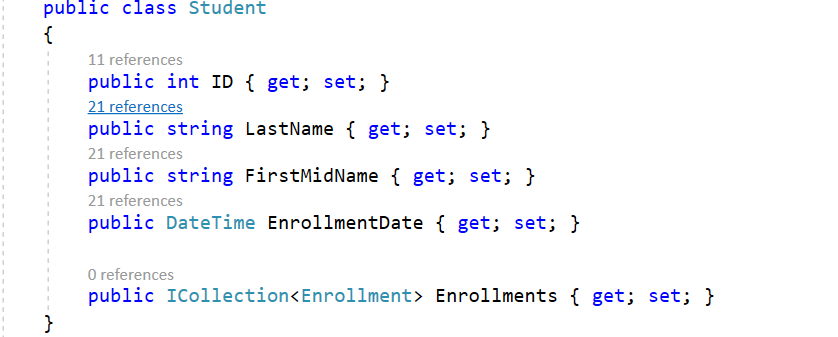


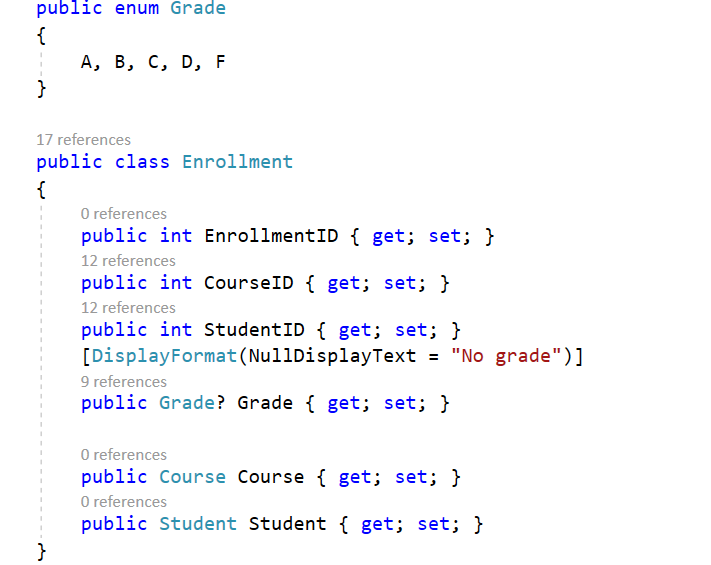


**Step 02**. Add model – Student, Course, Enrolment.

A student can enroll in any number of courses, and a course can have any number of students enrolled in it.







The database context class SchoolContext.cs.



DbInitializer.cs

using System;

using System.Linq;

using RazorPagesLab.Models;

namespace RazorPagesLab.Data

{

public class DbInitializer

{

public static void Initialize(SchoolContext context)

{

// Look for any students.

if (context.Students.Any())

{

return; // DB has been seeded

}

var students = new Student[]

{

new Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Parse("2019-09-01")},

new Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse("2017-09-01")},

new Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("2018-09-01")},

new Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse("2017-09-01")},

new Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2017-09-01")},

new Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Parse("2016-09-01")},

new Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse("2018-09-01")},

new Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Parse("2019-09-01")}

};

context.Students.AddRange(students);

context.SaveChanges();

var courses = new Course[]

{

new Course{CourseID=1050,Title="Chemistry",Credits=3},

new Course{CourseID=4022,Title="Microeconomics",Credits=3},

new Course{CourseID=4041,Title="Macroeconomics",Credits=3},

new Course{CourseID=1045,Title="Calculus",Credits=4},

new Course{CourseID=3141,Title="Trigonometry",Credits=4},

new Course{CourseID=2021,Title="Composition",Credits=3},

new Course{CourseID=2042,Title="Literature",Credits=4}

};

context.Courses.AddRange(courses);

context.SaveChanges();

var enrollments = new Enrollment[]

{

new Enrollment{StudentID=1,CourseID=1050,Grade=Grade.A},

new Enrollment{StudentID=1,CourseID=4022,Grade=Grade.C},

new Enrollment{StudentID=1,CourseID=4041,Grade=Grade.B},

new Enrollment{StudentID=2,CourseID=1045,Grade=Grade.B},

new Enrollment{StudentID=2,CourseID=3141,Grade=Grade.F},

new Enrollment{StudentID=2,CourseID=2021,Grade=Grade.F},

new Enrollment{StudentID=3,CourseID=1050},

new Enrollment{StudentID=4,CourseID=1050},

new Enrollment{StudentID=4,CourseID=4022,Grade=Grade.F},

new Enrollment{StudentID=5,CourseID=4041,Grade=Grade.C},

new Enrollment{StudentID=6,CourseID=1045},

new Enrollment{StudentID=7,CourseID=3141,Grade=Grade.A},

};

context.Enrollments.AddRange(enrollments);

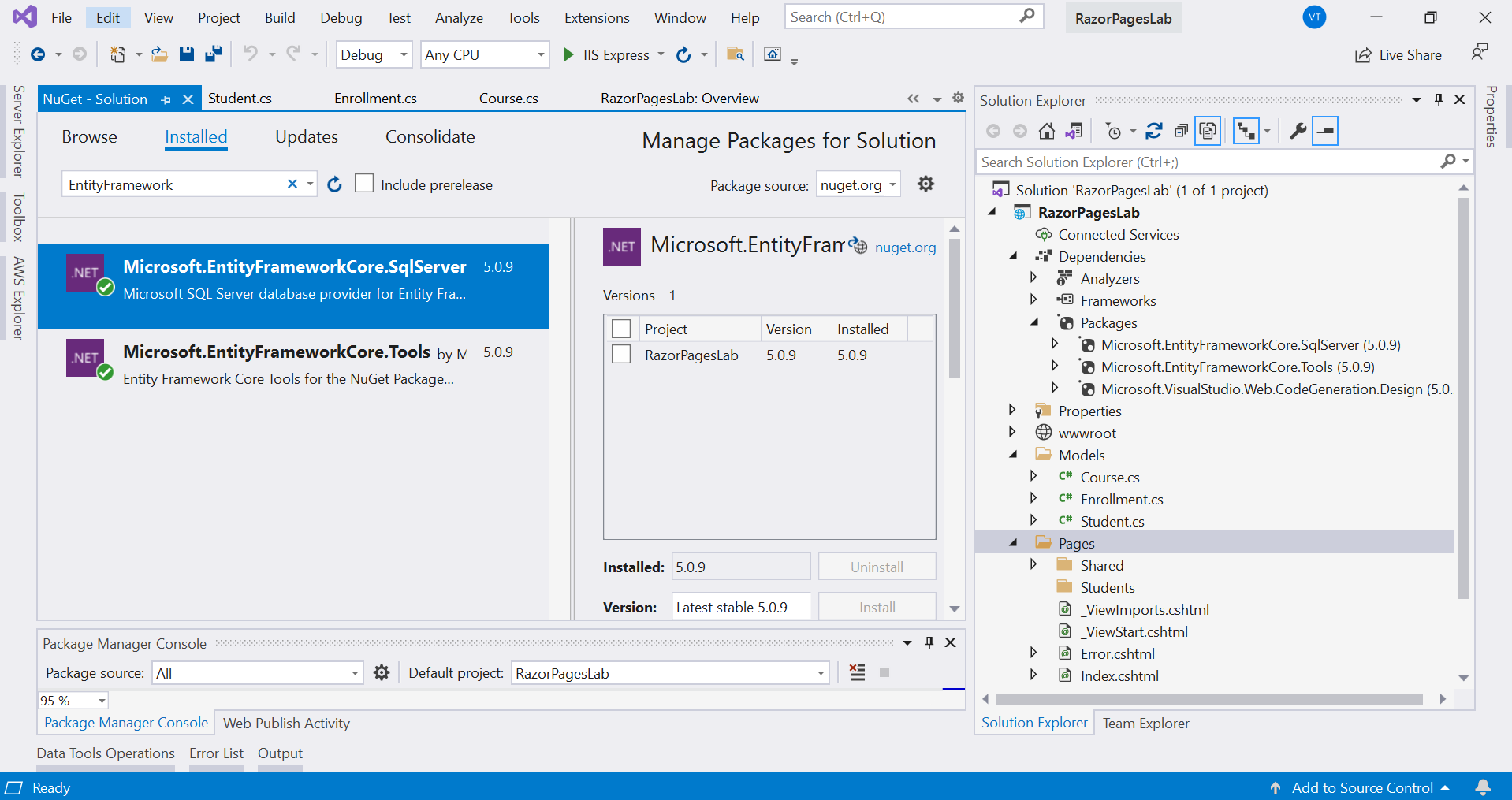
context.SaveChanges();

}

}

}

**Step 03**. Manage NuGet packages for Solution/Project



**Step 04**. Add Connection string (appsettings.json file)

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

**"ConnectionStrings": {**

**"DefaultConnection": "Persist Security Info=False;User ID=sa;Password=1234567890;Initial Catalog=SchoolContextDB;Data Source=.;Connection Timeout=100000"**

**}**

}

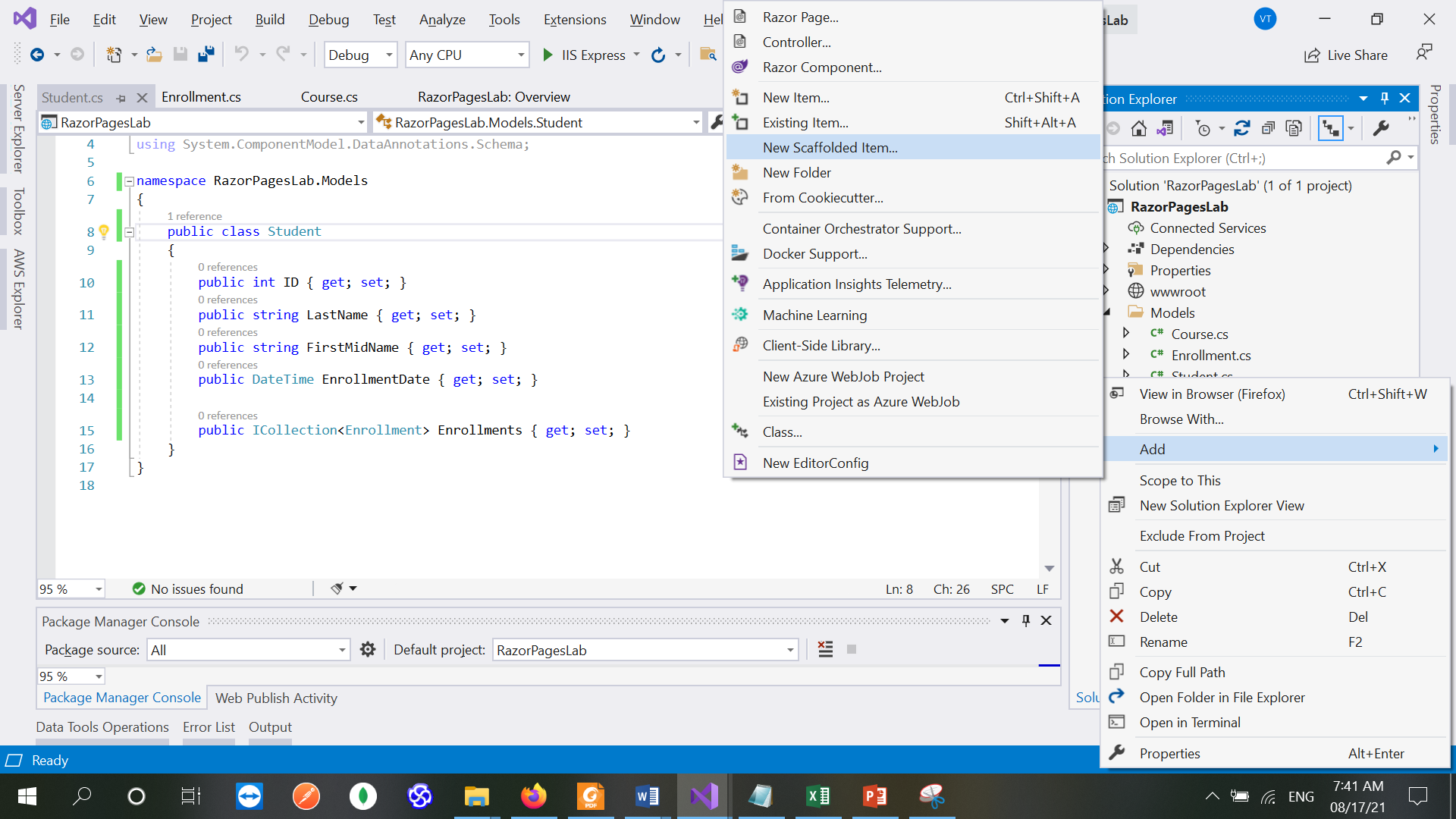
**Step 05**. Scaffold Student pages

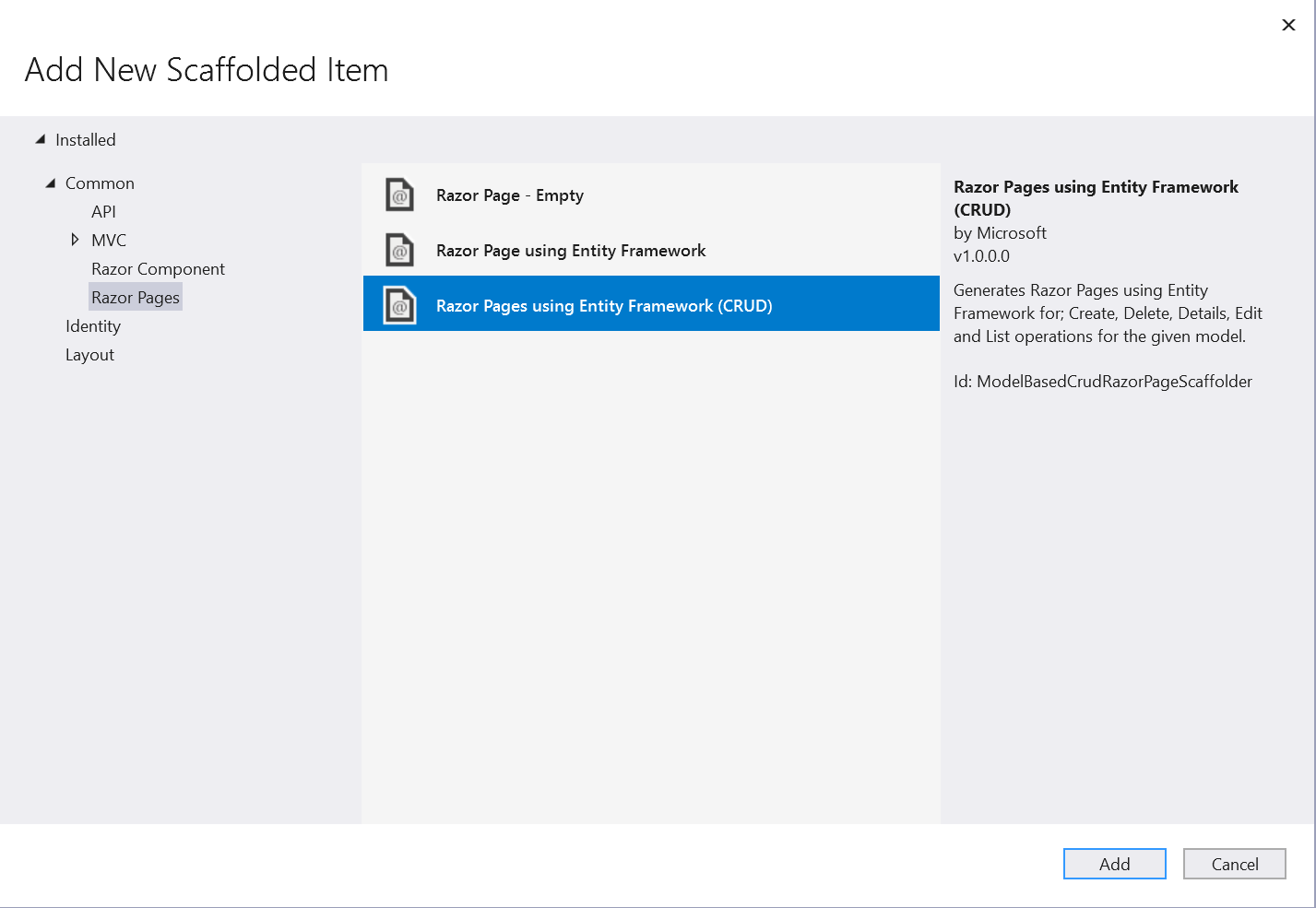
The following packages are automatically installed:

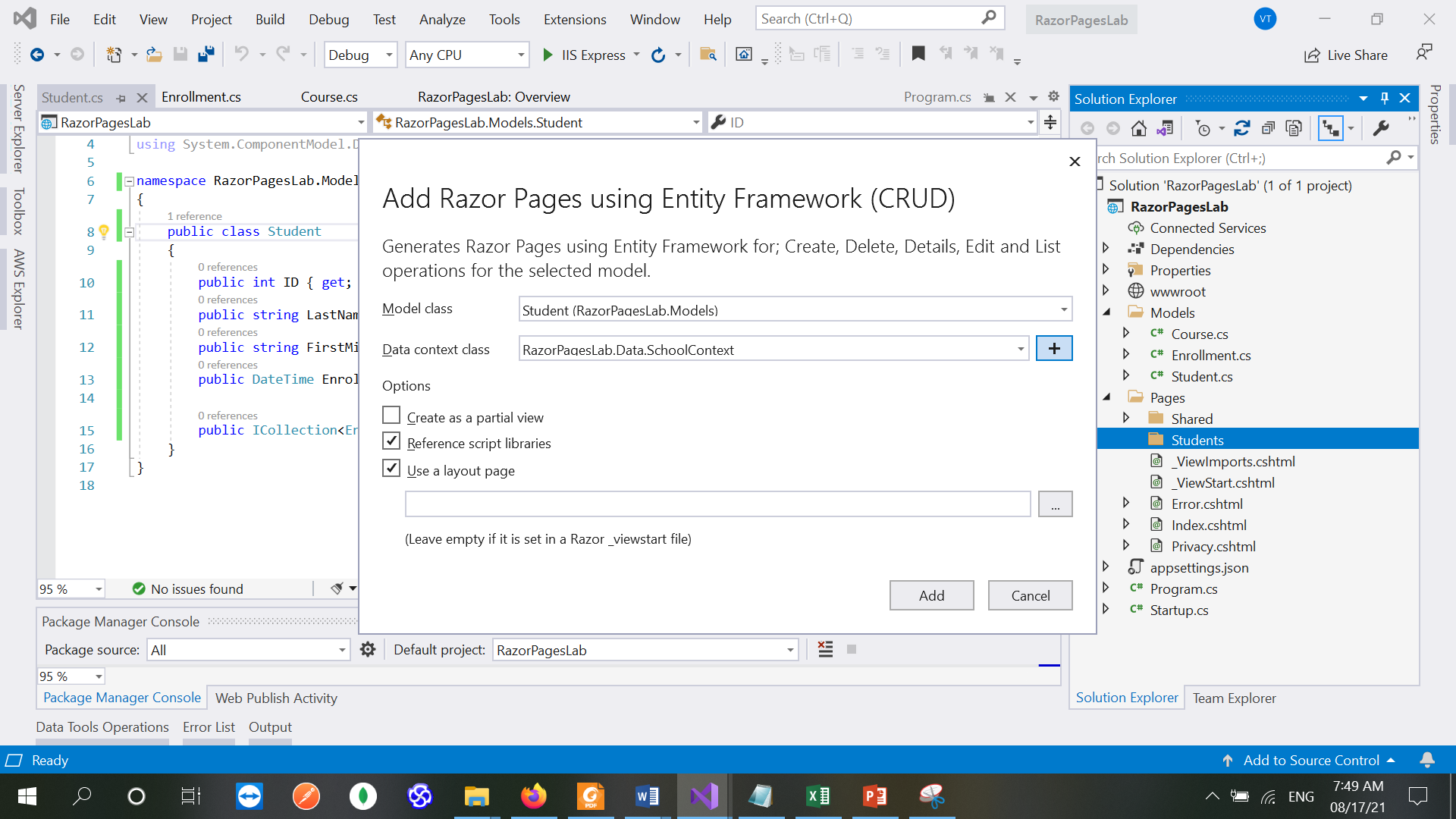
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools
* **Microsoft.VisualStudio.Web.CodeGeneration.Design**

The scaffolding process will provide these files (creates Razor pages in the *Pages/Students* folder)

* Create.cshtml and Create.cshtml.cs
* Delete.cshtml and Delete.cshtml.cs
* Details.cshtml and Details.cshtml.cs
* Edit.cshtml and Edit.cshtml.cs
* Index.cshtml and Index.cshtml.cs







**Step 06**. Change the code on Startup.cs and Program.cs

Add the context to dependency injection in *Startup.cs.*

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

## Register the SchoolContext

public void ConfigureServices(IServiceCollection services)

{

services.AddRazorPages();

services.AddDbContext<SchoolContext>(options =>

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

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseDeveloperExceptionPage();

app.UseMigrationsEndPoint();

}

else

{

app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapRazorPages();

});

}

}

Check if the database does not exist in the *Program.cs*

public class Program

{

public static void Main(string[] args)

{

var host = CreateHostBuilder(args).Build();

**CreateDbIfNotExists(host);**

host.Run();

}

**private static void CreateDbIfNotExists(IHost host)**

**{**

**using (var scope = host.Services.CreateScope())**

**{**

**var services = scope.ServiceProvider;**

**try**

**{**

**var context = services.GetRequiredService<SchoolContext>();**

**context.Database.EnsureCreated();**

**// DbInitializer.Initialize(context);**

**}**

**catch (Exception ex)**

**{**

**var logger = services.GetRequiredService<ILogger<Program>>();**

**logger.LogError(ex, "An error occurred creating the DB.");**

**}**

**}**

**}**

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

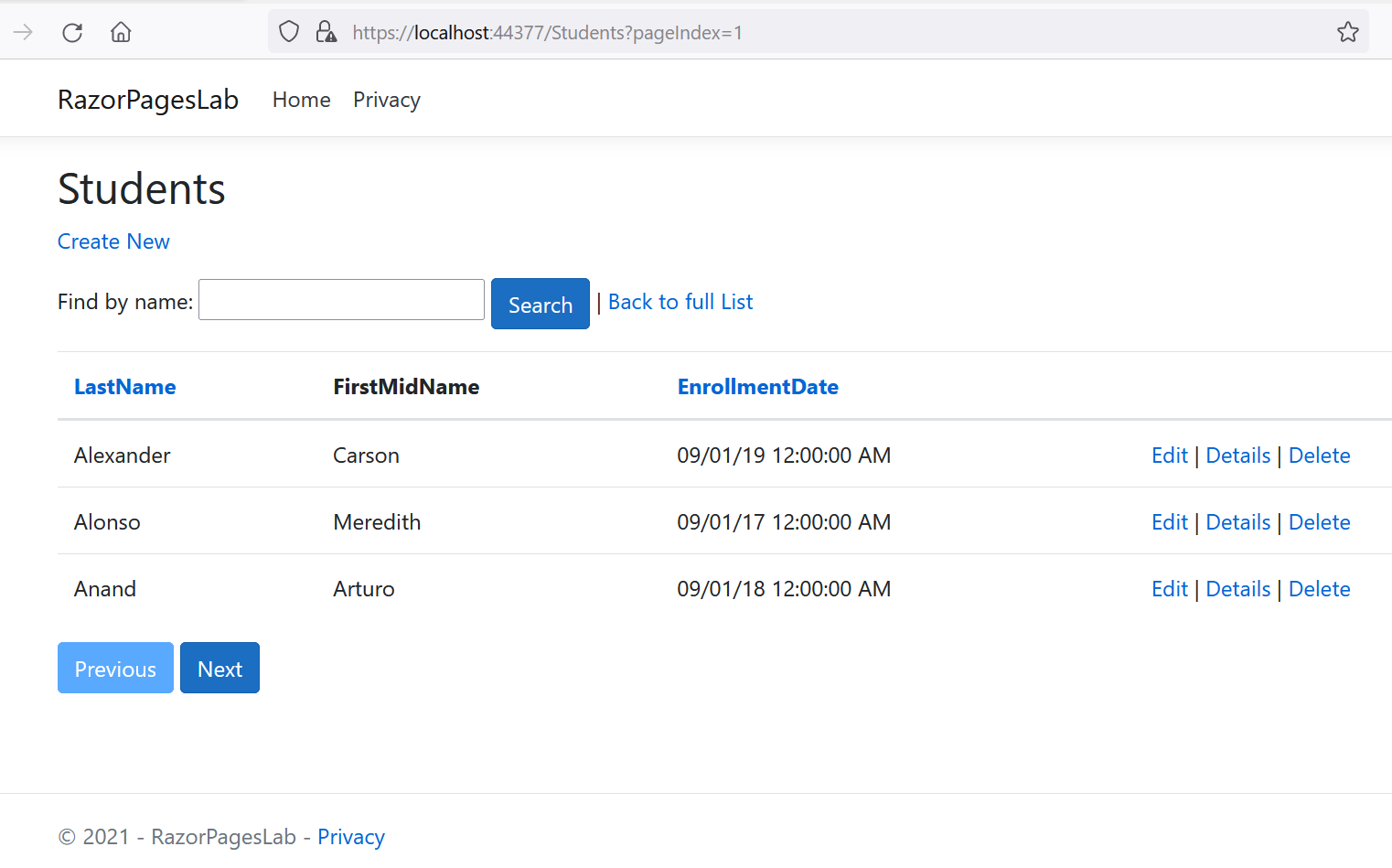
webBuilder.UseStartup<Startup>();

});

}

**Step 07**. Build and run Program.

# Activity 03: Razor Pages with Entity Framework - Sort, Filter, Paging



**Step 00**. Using Project in Activity 2.

**Step 01**. Working with Index.cshtml.cs

The method OnGetAsync using the parameters

* the *sortOrder* as parameter from the query string in the URL
* the *currentFilter* as parameter for filtering and saves the parameter value in the CurrentFilter property.
* the *searchString* as parameter for search string
* the *pageIndex* as parameter for page index

**Step 02**. Add pagging. Create PaginatedList class to support paging. The PaginatedList class uses Skip and Take statements to filter data on the server instead of retrieving all rows of the table.

public class PaginatedList<T> : List<T>

{

public int PageIndex { get; private set; }

public int TotalPages { get; private set; }

public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)

{

PageIndex = pageIndex;

TotalPages = (int)Math.Ceiling(count / (double)pageSize);

this.AddRange(items);

}

public bool HasPreviousPage

{

get

{

return (PageIndex > 1);

}

}

public bool HasNextPage

{

get

{

return (PageIndex < TotalPages);

}

}

public static async Task<PaginatedList<T>> CreateAsync(

IQueryable<T> source, int pageIndex, int pageSize)

{

var count = await source.CountAsync();

var items = await source.Skip(

(pageIndex - 1) \* pageSize)

.Take(pageSize).ToListAsync();

return new PaginatedList<T>(items, count, pageIndex, pageSize);

}

}

### Add page size to configuration

{

**"PageSize": 3,**

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

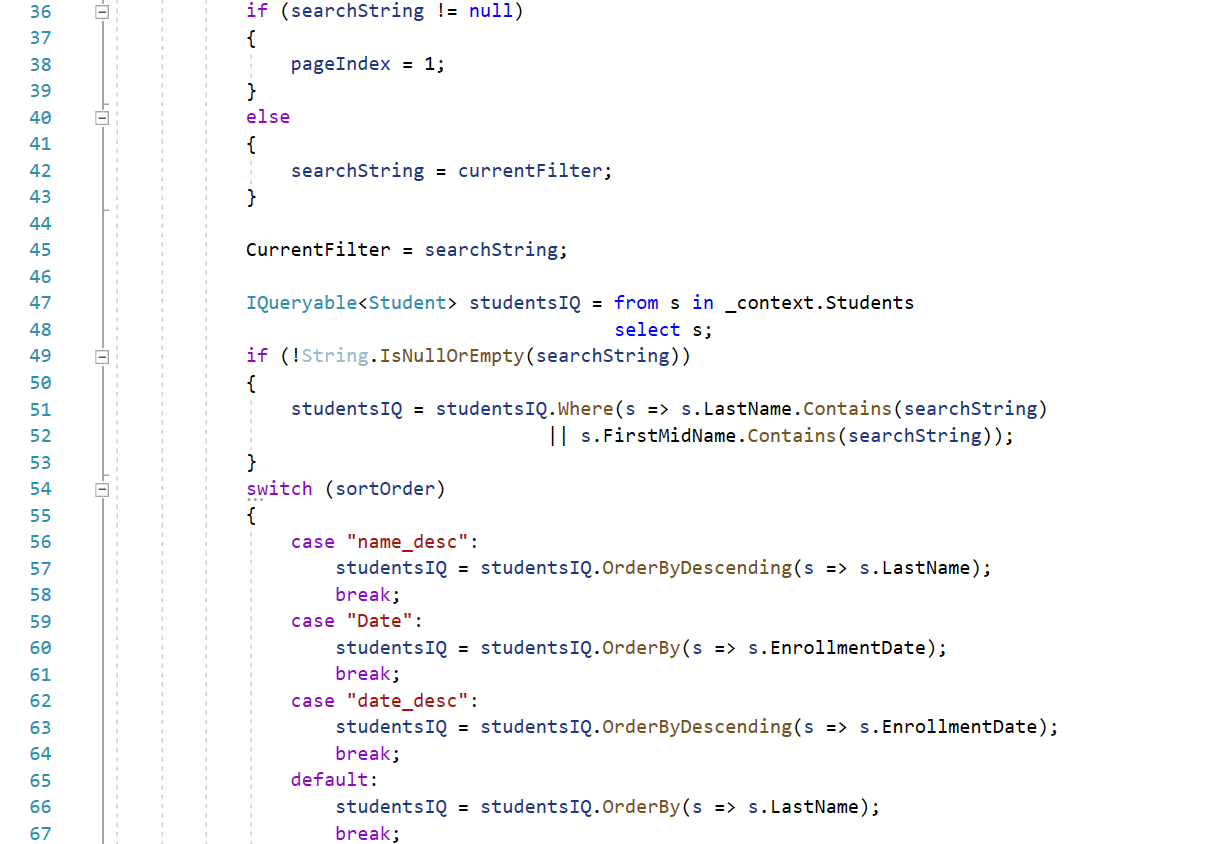
"ConnectionStrings": {

"SchoolContext": "Server=.;Database=SchoolContextLab;Trusted\_Connection=True;MultipleActiveResultSets=true"

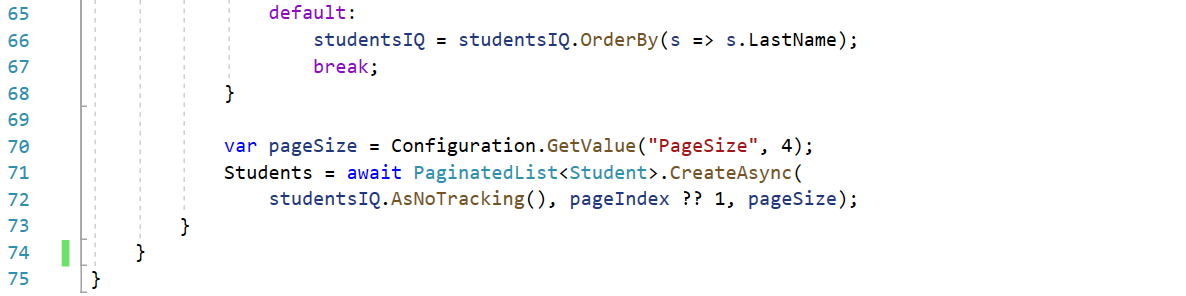
}

}



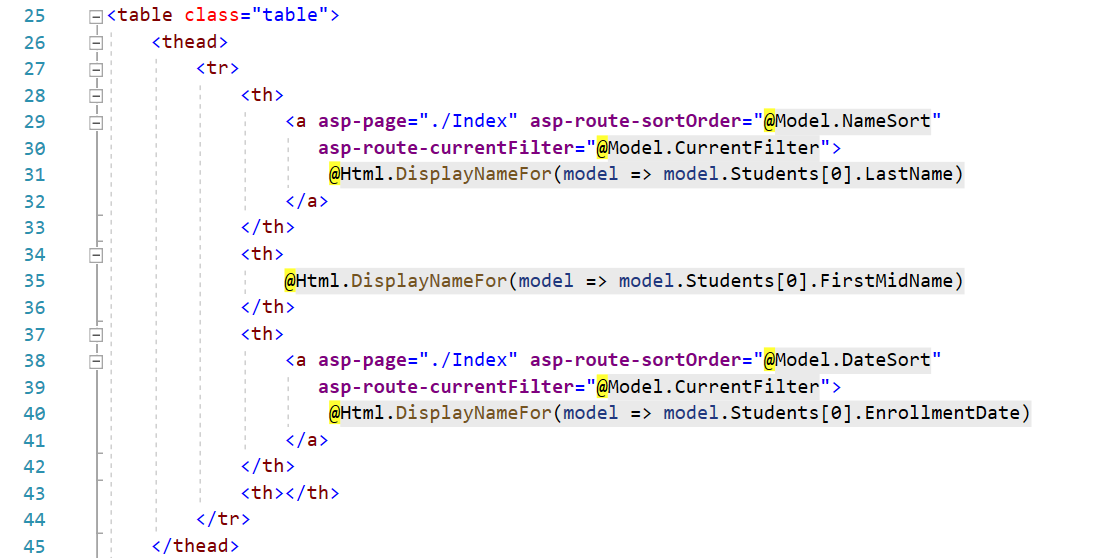


The method uses LINQ to Entities to specify the column to sort by.

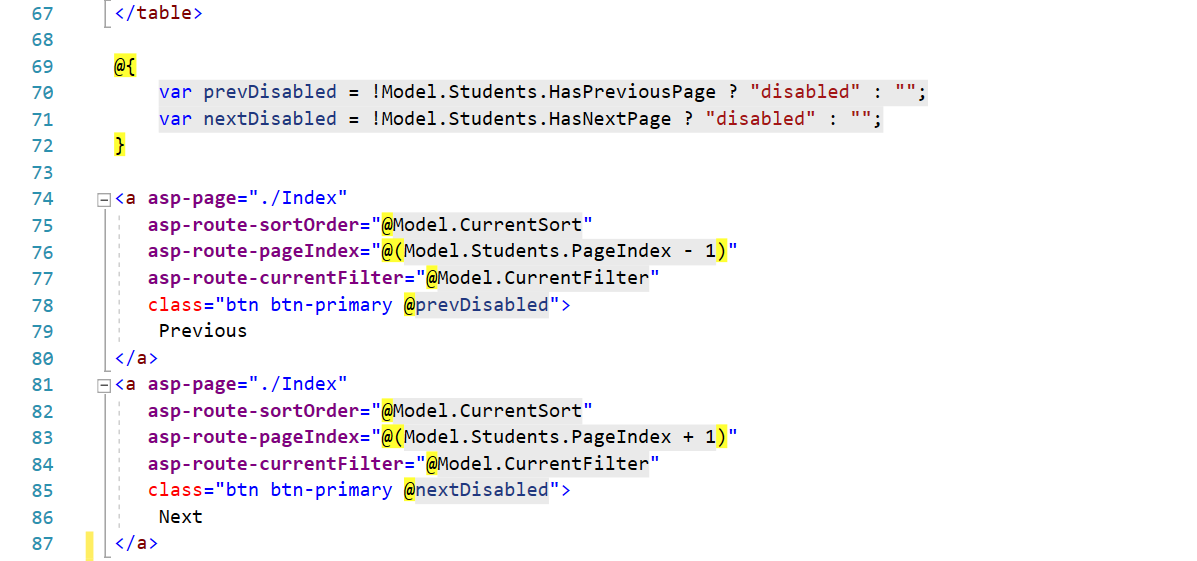


**Step 03**. Work with Student Index page Index.cshtml









**Step 04**. Run and test all functions of Project.

