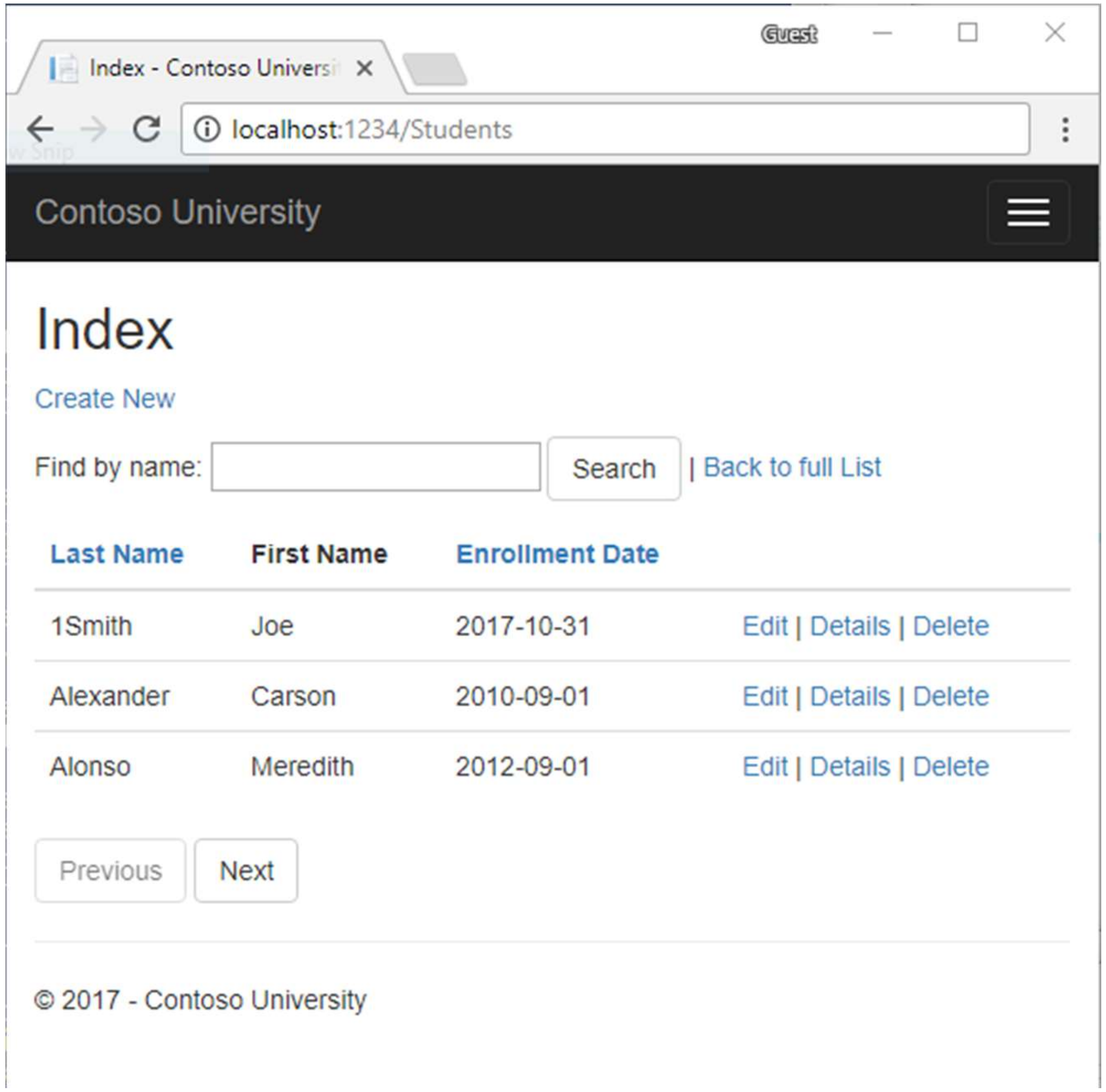
CO550 – Web Applications

UNIT 5 – Creating A Basic Web App, Razor Pages Tutorial and Theory

Building the Web App

What we're aiming for…

Very quickly, we can have a fully working web application up and running with very little coding



Index

Create New

Find by name: Search | Back to full List

Last Name	First Name	Enrollment Date	
1Smith	Joe	2017-10-31	Edit | Details | Delete
Alexander	Carson	2010-09-01	Edit | Details | Delete
Alonso	Meredith	2012-09-01	Edit | Details | Delete

Previous Next

© 2017 - Contoso University

Building the Web App

Let’s recap what the first step of the Razor Pages tutorial covers…

https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-2.1&tabs=visual-studio

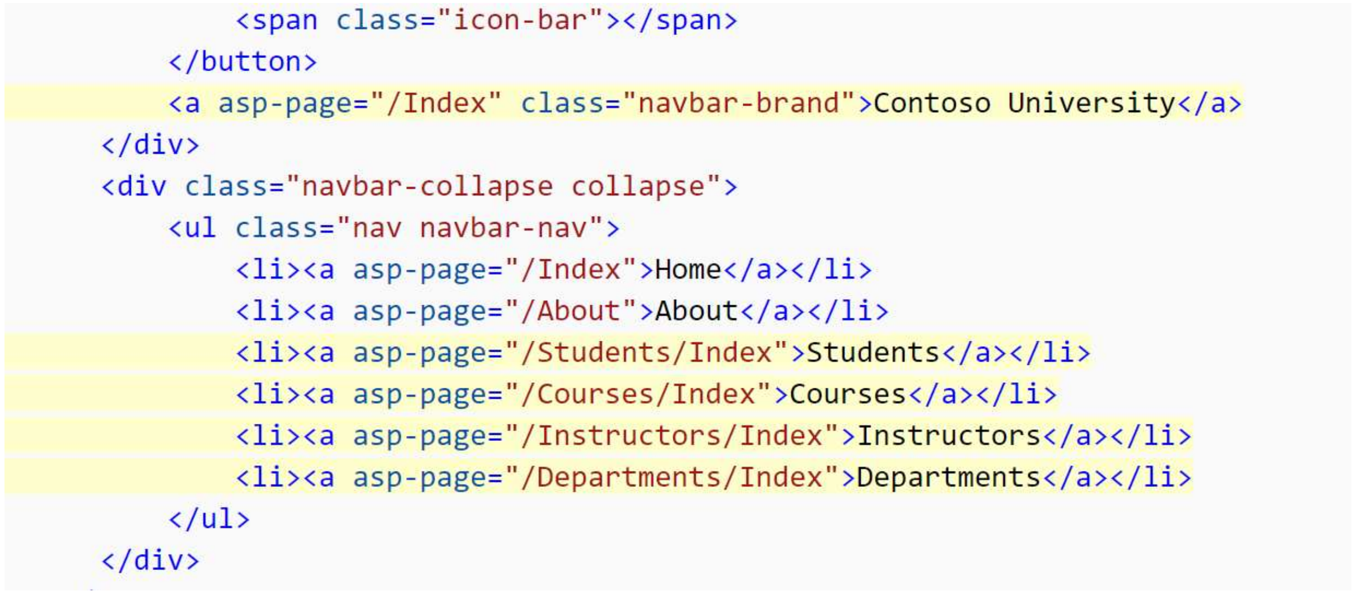
Building the Web App

If we are on Windows…

- Visual Studio **File** menu, select **New** > **Project**.
- Create a new ASP.NET Core Web Application.
- Name the project appropriately (this impacts on namespaces)
- Select **ASP.NET Core 2.1** in the dropdown, then select **Web Application**.

Building the Web App

We then setup the “style” or HTML which gives us the navigation menu for the various pages we will be scaffolding next…

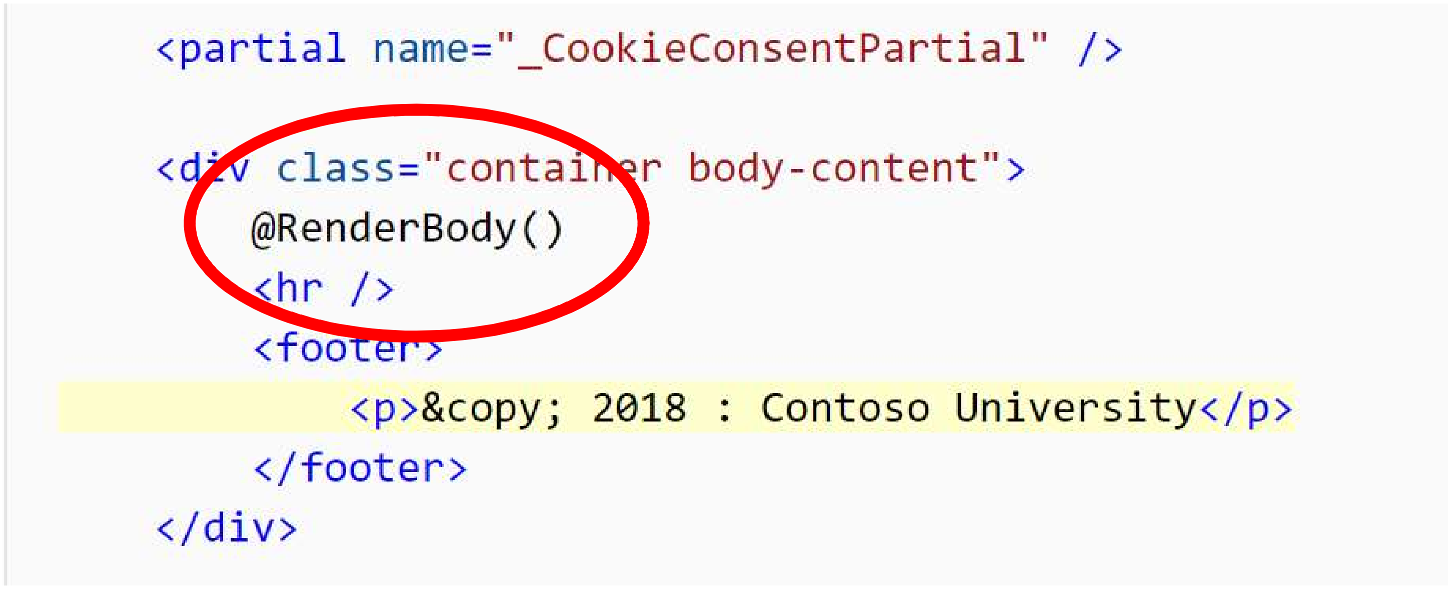


```
                <span class="icon-bar"></span>
            </button>
            <a asp-page="/Index" class="navbar-brand">Contoso University</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li><a asp-page="/Index">Home</a></li>
                <li><a asp-page="/About">About</a></li>
                <li><a asp-page="/Students/Index">Students</a></li>
                <li><a asp-page="/Courses/Index">Courses</a></li>
                <li><a asp-page="/Instructors/Index">Instructors</a></li>
                <li><a asp-page="/Departments/Index">Departments</a></li>
            </ul>
        </div>
```

File edited: ***Pages/Shared/_Layout.cshtml***

Building the Web App

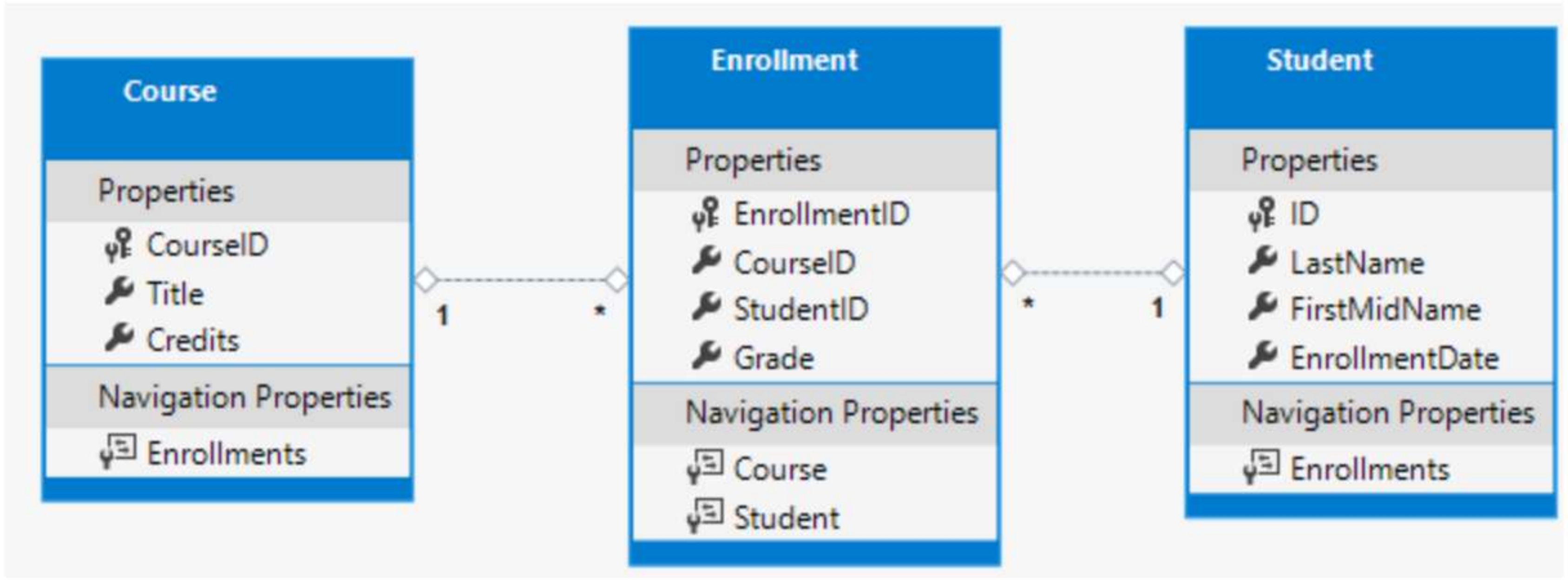
A key part of ***Pages/Shared/_Layout.cshtml***



```
<partial name="_CookieConsentPartial" />

<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; 2018 : Contoso University</p>
    </footer>
</div>
```

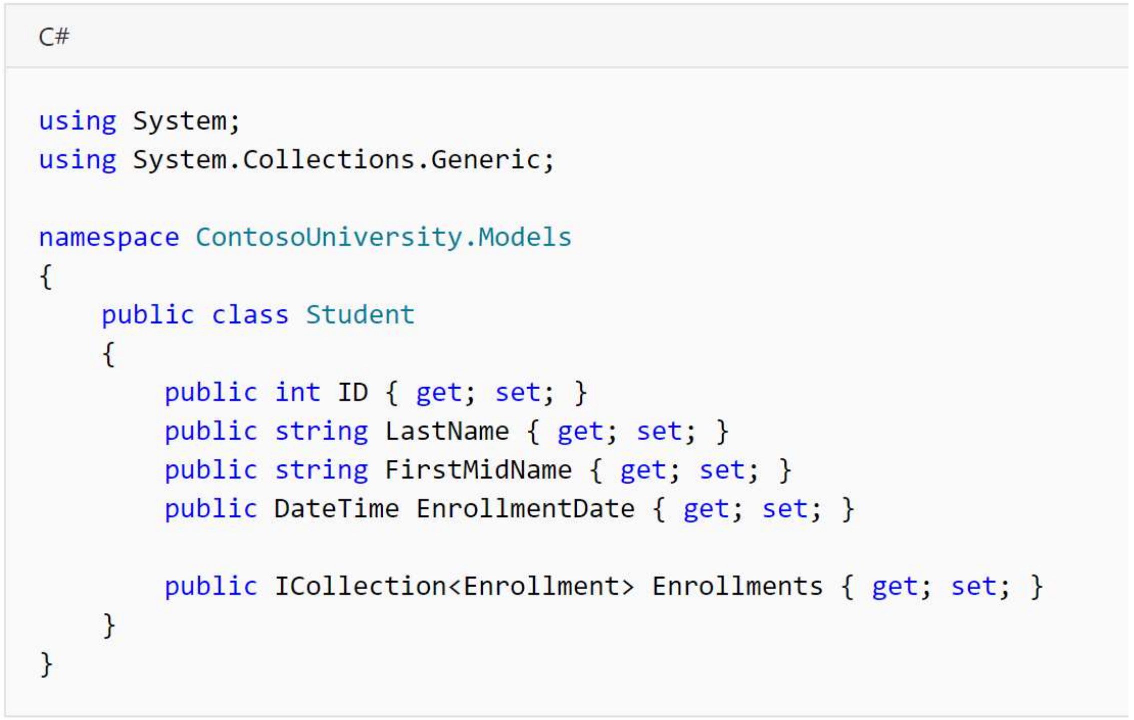
Setting up the Models

Note the relationships….



Setting up the Models

We create a “Models” folder to hold our models (the same as we would with MVC)



C#

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

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

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

Scaffolding

- ASP.NET Scaffolding lets us create the pages for Create, Read, Update, and Delete (CRUD) operations for a particular model (e.g. the student model)

- In **Solution Explorer**, right click on the *Pages/Students* folder > **Add** > **New Scaffolded Item**.
- In the **Add Scaffold** dialog, select **Razor Pages using Entity Framework (CRUD)** > **ADD**.
- etc…

Scaffolding

Files Added

- *Pages/Students* Create, Delete, Details, Edit, Index.
- *Data/SchoolContext.cs*

File Updates

- *Startup.cs*
- *appsettings.json* : The connection string used to connect to a local database is added.

Startup.cs

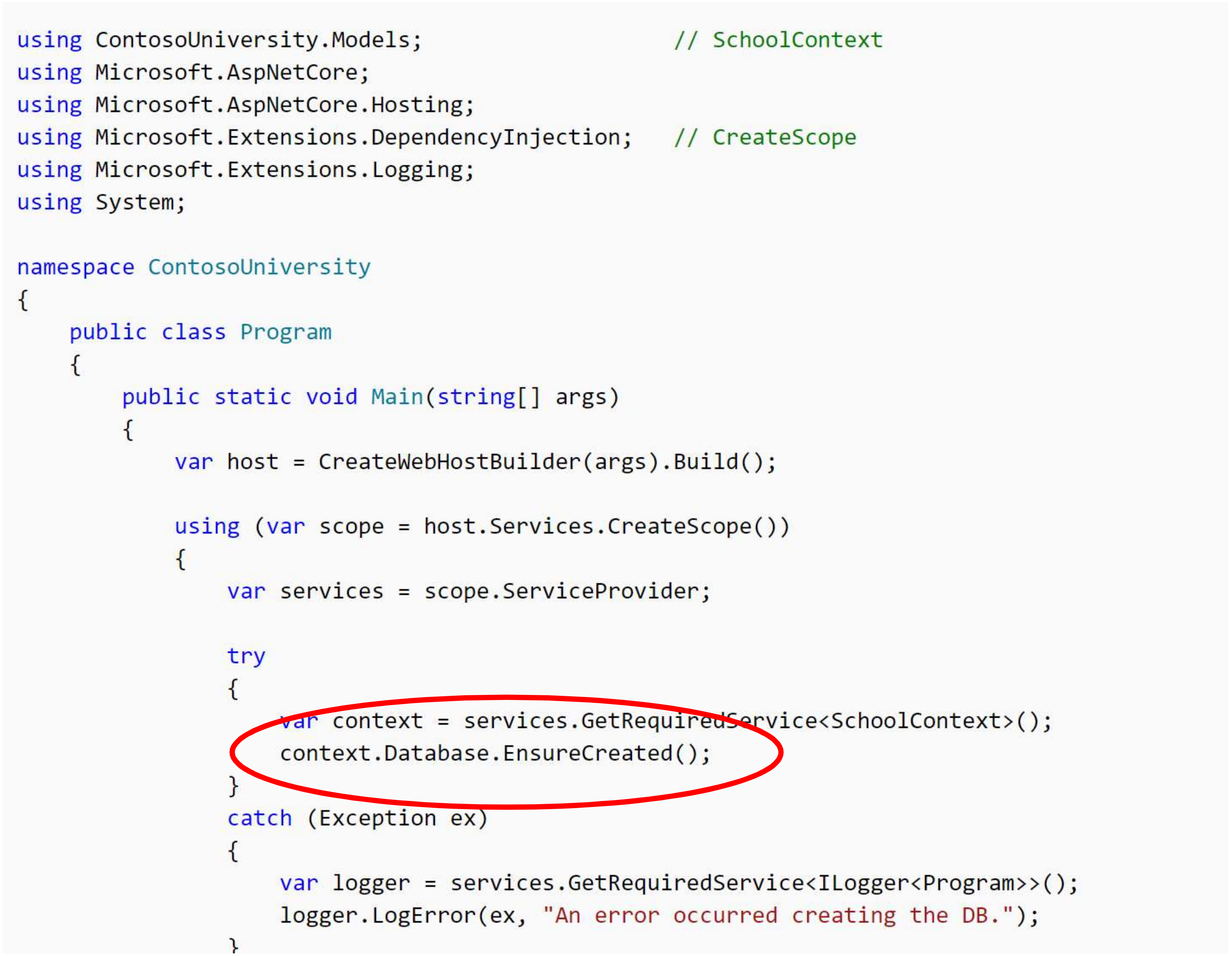
The scaffolding tool automatically created a DB Context and registered it with the dependency injection container.

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for
        //non -essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

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

Making Sure the Database Gets Created

In *Program.cs*, we modify the Main method to do the following:

- Get a DB context instance from the dependency injection container.
- Call EnsureCreated.
- Dispose the context when the EnsureCreated method completes.



```
using ContosoUniversity.Models;                         // SchoolContext
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;    // CreateScope
using Microsoft.Extensions.Logging;
using System;

namespace ContosoUniversity
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateWebHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    var context = services.GetRequiredService<SchoolContext>();
                    context.Database.EnsureCreated();
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred creating the DB.");
                }
```

EnsureCreated

EnsureCreated ensures that the database for the context exists. If it exists, no action is taken. If it does not exist, then the database and all its schema are created.

EnsureCreated does not use migrations to create the database. A database that is created with EnsureCreated cannot be later updated using migrations.

EnsureCreated is called on app start, which allows the following workflow:

- Delete the DB.
- Change the DB schema (for example, add an EmailAddress field).
- Run the app.
- EnsureCreated creates a DB with the EmailAddress column.

EnsureCreated is convenient early in development when the schema is rapidly evolving. Later in the tutorial the DB is deleted and migrations are used.

Seeding the Database

In the *Data* folder, we create a new class file named *DbInitializer.cs…*

```
namespace ContosoUniversity.Models
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            // context.Database.EnsureCreated();

            // Look for any students.
            if (context.Student.Any())
            {
                return;   // DB has been seeded
            }

            var students = new Student[]
            {
            new Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Parse("2005
            new Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse("2002-
            new Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("2003-09-
            new Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse("2002-
            new Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2002-09-01")},
            new Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Parse("2001-09
            new Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse("2003-09-
            new Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Parse("2005-09
            };
            foreach (Student s in students)
            {
                context.Student.Add(s);
            }
            context.SaveChanges();
```

Checking the Database

- Open **SQL Server Object Explorer** (SSOX) from the **View** menu in Visual Studio. In SSOX, click **(localdb)\MSSQLLocalDB > Databases > ContosoUniversity1** (for example)

- Expand the **Tables** node.

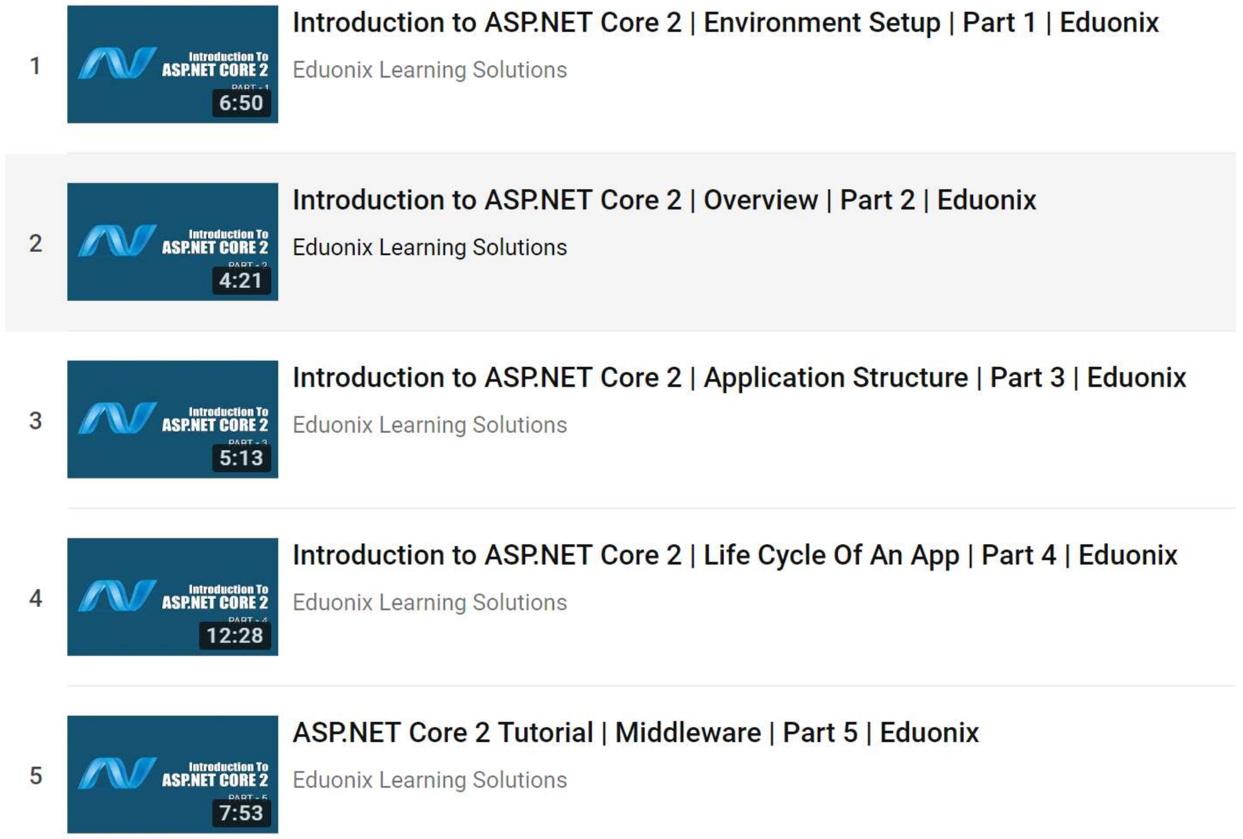
- Right-click a table and click **View Data** to see the columns created and the rows inserted into the table.

Digging Deeper into the theory

VIDEOS For Discussion

https://www.youtube.com/playlist?list=PLDmvslp_VR0x2CmC6c4AZhZfYX7G2nBIo

First 5 videos covering: Environment setup, Application structure, Lifecycle of an app, Middleware



1 Introduction to ASP.NET Core 2 | Environment Setup | Part 1 | Eduonix
Eduonix Learning Solutions
6:50

2 Introduction to ASP.NET Core 2 | Overview | Part 2 | Eduonix
Eduonix Learning Solutions
4:21

3 Introduction to ASP.NET Core 2 | Application Structure | Part 3 | Eduonix
Eduonix Learning Solutions
5:13

4 Introduction to ASP.NET Core 2 | Life Cycle Of An App | Part 4 | Eduonix
Eduonix Learning Solutions
12:28

5 ASP.NET Core 2 Tutorial | Middleware | Part 5 | Eduonix
Eduonix Learning Solutions
7:53

Further Reading

Application Startup

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-2.1

Middleware

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-2.1

NOW: TUTORIAL Workshop

Step 2 of ASP.NET Core Razor Pages tutorial

https://docs.microsoft.com/en-us/aspnet/core/data/ef-rp/crud?view=aspnetcore-2.1

1. Modifying the Student details page
2. Showing related data on the details page

Logbook 3