https://www.youtube.com/watch?v=C5cnZ-gZy2I

Learn ASP.NET Core 3.1 - Full Course for Beginners

A free course By Bhrugen Patel

Detailed guide

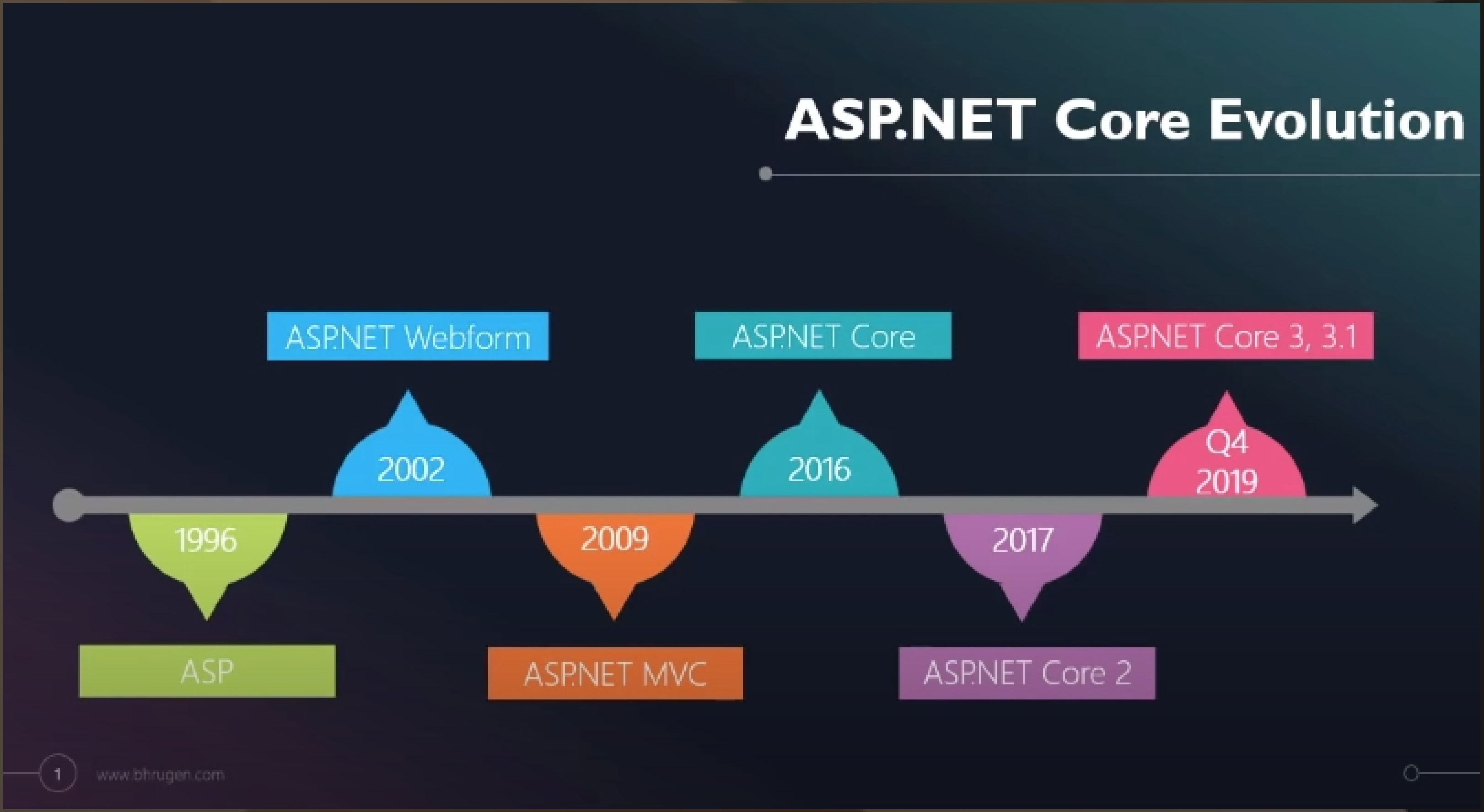
Created by
antony.kidis@gmail.com
https://github.com/antonykidis

There are two ways building MVC applications

1. MVC Application

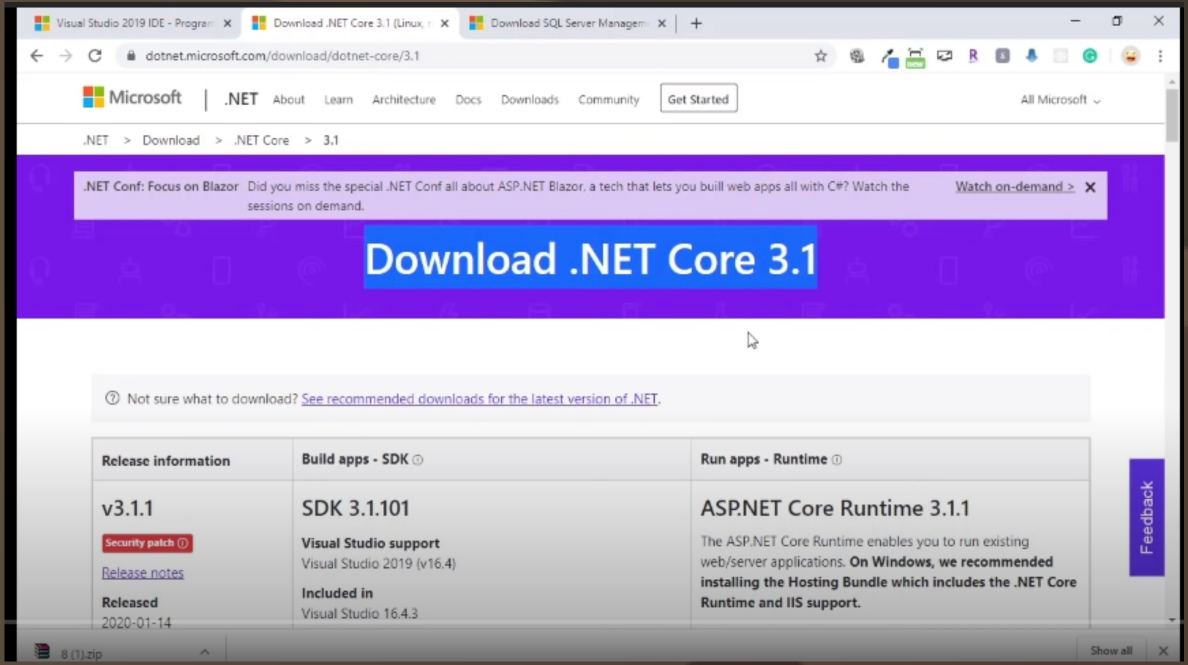
2 Razor Pages Application

We will build the project using both techniques to understand how they work.



ASP.NET Core Evolution

1. download Visual studio 2019 comunitry version
2. download a Dot.Net Core 3.1


Microsoft | .NET About Learn Architecture Docs Downloads Community Get Started
.NET > Download > .NET Core > 3.1

.NET Conf: Focus on Blazor Did you miss the special .NET Conf all about ASP.NET Blazor, a tech that lets you buill web apps all with C#? Watch the sessions on demand. Watch on-demand >

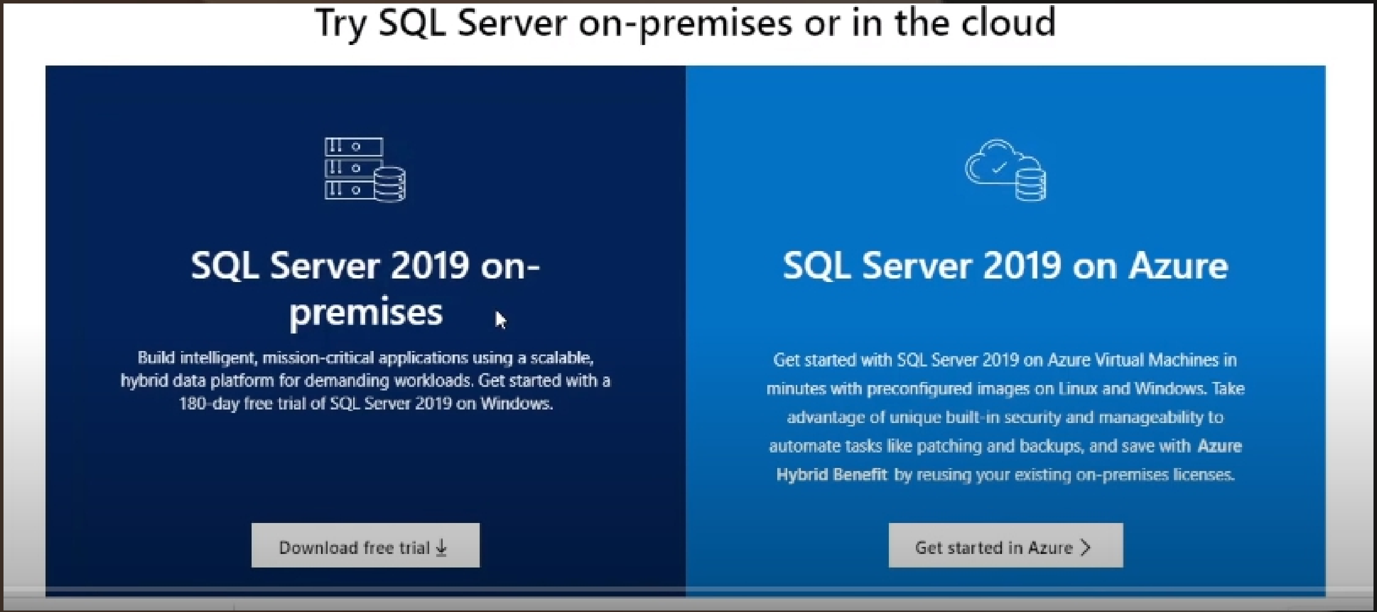
Download .NET Core 3.1

Not sure what to download? See recommended downloads for the latest version of .NET.

Release information	Build apps - SDK	Run apps - Runtime
v3.1.1 Security patch Release notes Released 2020-01-14	SDK 3.1.101 Visual Studio support Visual Studio 2019 (v16.4) Included in Visual Studio 16.4.3	ASP.NET Core Runtime 3.1.1 The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommended installing the Hosting Bundle which includes the .NET Core Runtime and IIS support.**

3.Download SQL SERVER 2019 Download it for free for a developer version

4. download SQLServer Managment Studio



Try SQL Server on-premises or in the cloud

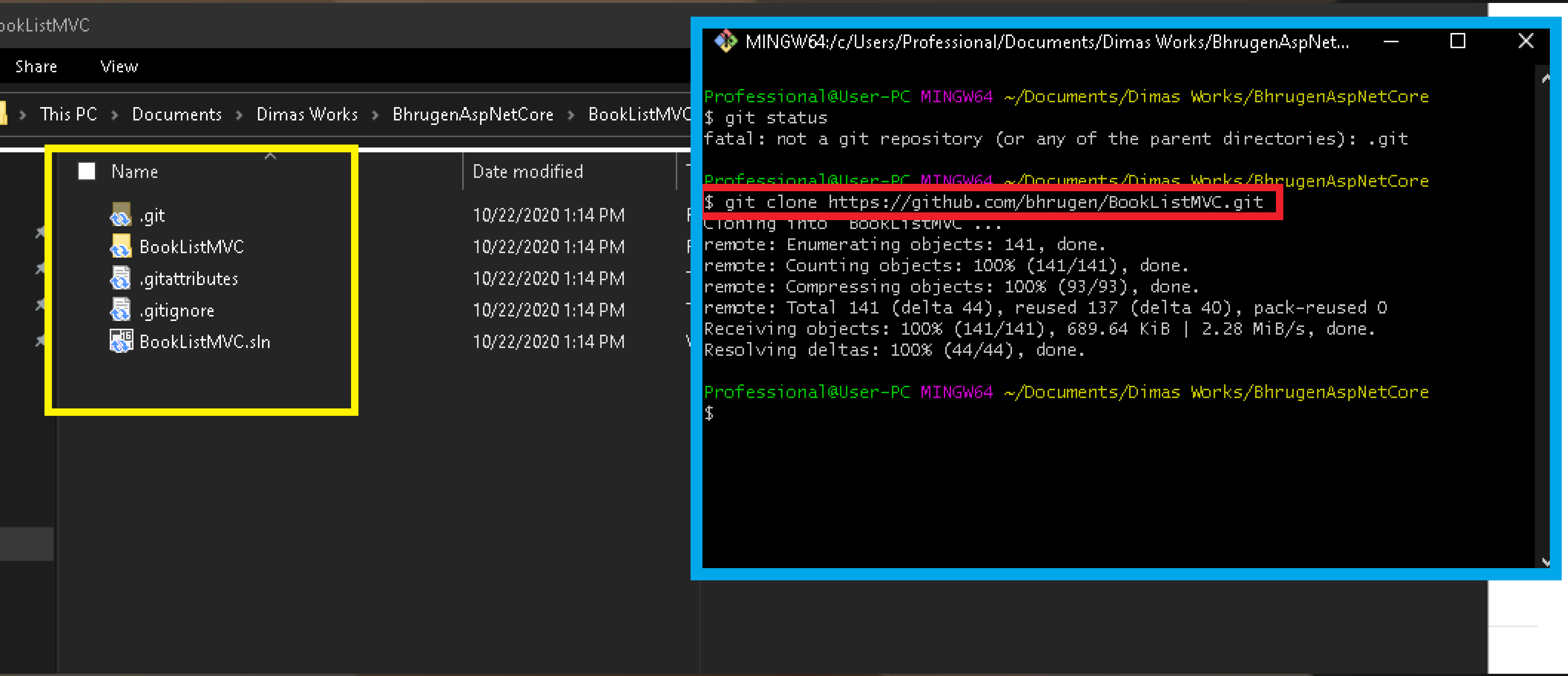
SQL Server 2019 on-premises

Build intelligent, mission-critical applications using a scalable, hybrid data platform for demanding workloads. Get started with a 180-day free trial of SQL Server 2019 on Windows.

SQL Server 2019 on Azure

Get started with SQL Server 2019 on Azure Virtual Machines in minutes with preconfigured images on Linux and Windows. Take advantage of unique built-in security and manageability to automate tasks like patching and backups, and save with **Azure Hybrid Benefit** by reusing your existing on-premises licenses.

We can clone our repo to a local PC using bash.exe
You have to install GIT to be able using git commands inside your terminal(if you on Windows)
Once you install git, you will be able to use git commands in powershell, or other preferred Console
Type git clone git@github.com:bhrugen/BookListRazor.git
Another way is to simply download a Zip file: https://github.com/bhrugen/BookListRazor/archive/master.zip



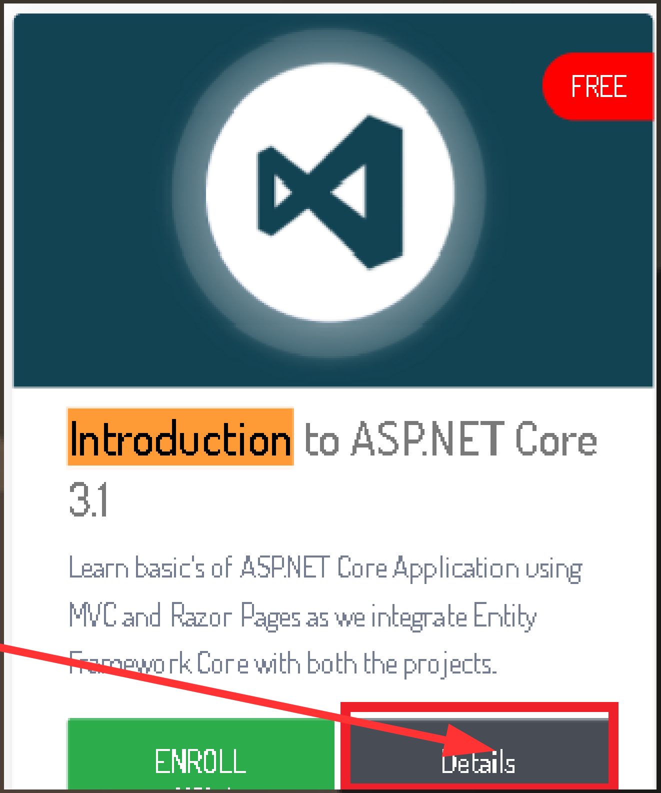
ookListMVC
Share
View
This PC > Documents > Dimas Works > BhrugenAspNetCore > BookListMVC
Name
Date modified

Name	Date modified
.git	10/22/2020 1:14 PM
BookListMVC	10/22/2020 1:14 PM
.gitattributes	10/22/2020 1:14 PM
.gitignore	10/22/2020 1:14 PM
BookListMVC.sln	10/22/2020 1:14 PM

MINGW64:/c/Users/Professional/Documents/Dimas Works/BhrugenAspNet...

```
Professional@User-PC MINGW64 ~/Documents/Dimas Works/BhrugenAspNetCore
$ git status
fatal: not a git repository (or any of the parent directories): .git

Professional@User-PC MINGW64 ~/Documents/Dimas Works/BhrugenAspNetCore
$ git clone https://github.com/bhrugen/BookListMVC.git
Cloning into 'BookListMVC'...
remote: Enumerating objects: 141, done.
remote: Counting objects: 100% (141/141), done.
remote: Compressing objects: 100% (93/93), done.
remote: Total 141 (delta 44), reused 137 (delta 40), pack-reused 0
Receiving objects: 100% (141/141), 689.64 KiB | 2.28 MiB/s, done.
Resolving deltas: 100% (44/44), done.

Professional@User-PC MINGW64 ~/Documents/Dimas Works/BhrugenAspNetCore
$
```

You will find the source code in my website
http://bhrugen.com
https://www.dotnetmastery.com/

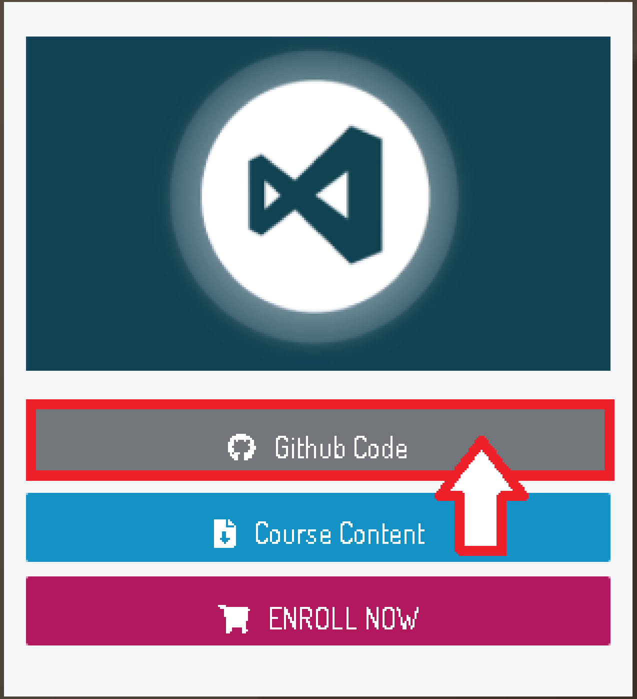


Introduction to ASP.NET Core 3.1

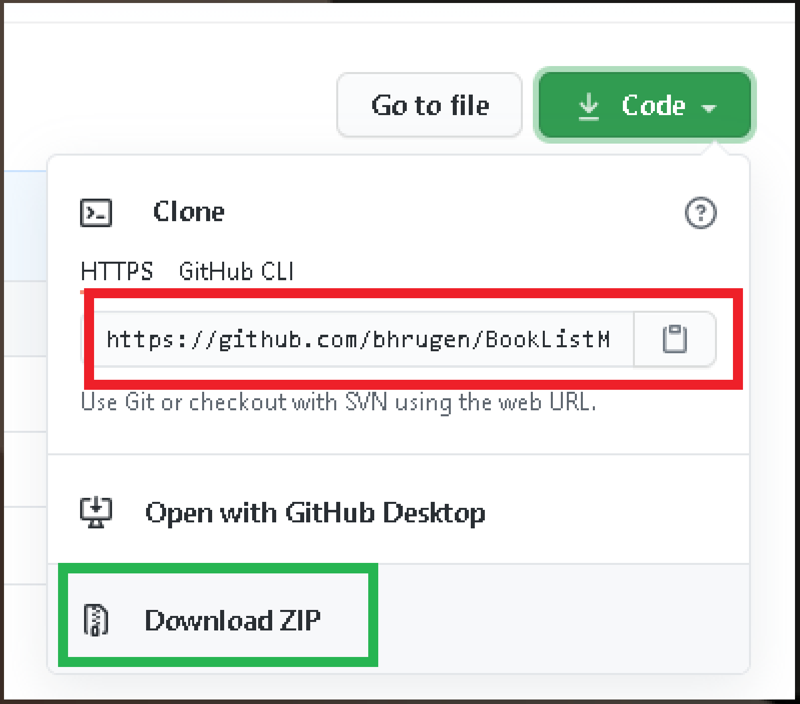
Learn basic's of ASP.NET Core Application using MVC and Razor Pages as we integrate Entity Framework Core with both the projects.

1.Scroll down the page and find a course
Introduction to ASP.NET Core
Click Details button.

2.Click Github Code



4. download zip file
Or clone a repo
Using bash, or
Windows powershell

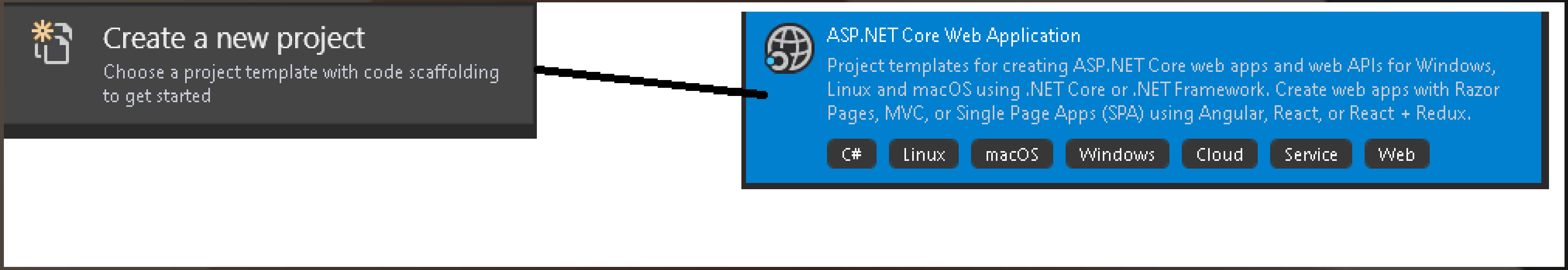


```
https://github.com/bhrugen/BookListM
```

Use Git or checkout with SVN using the web URL.

Lets create a project

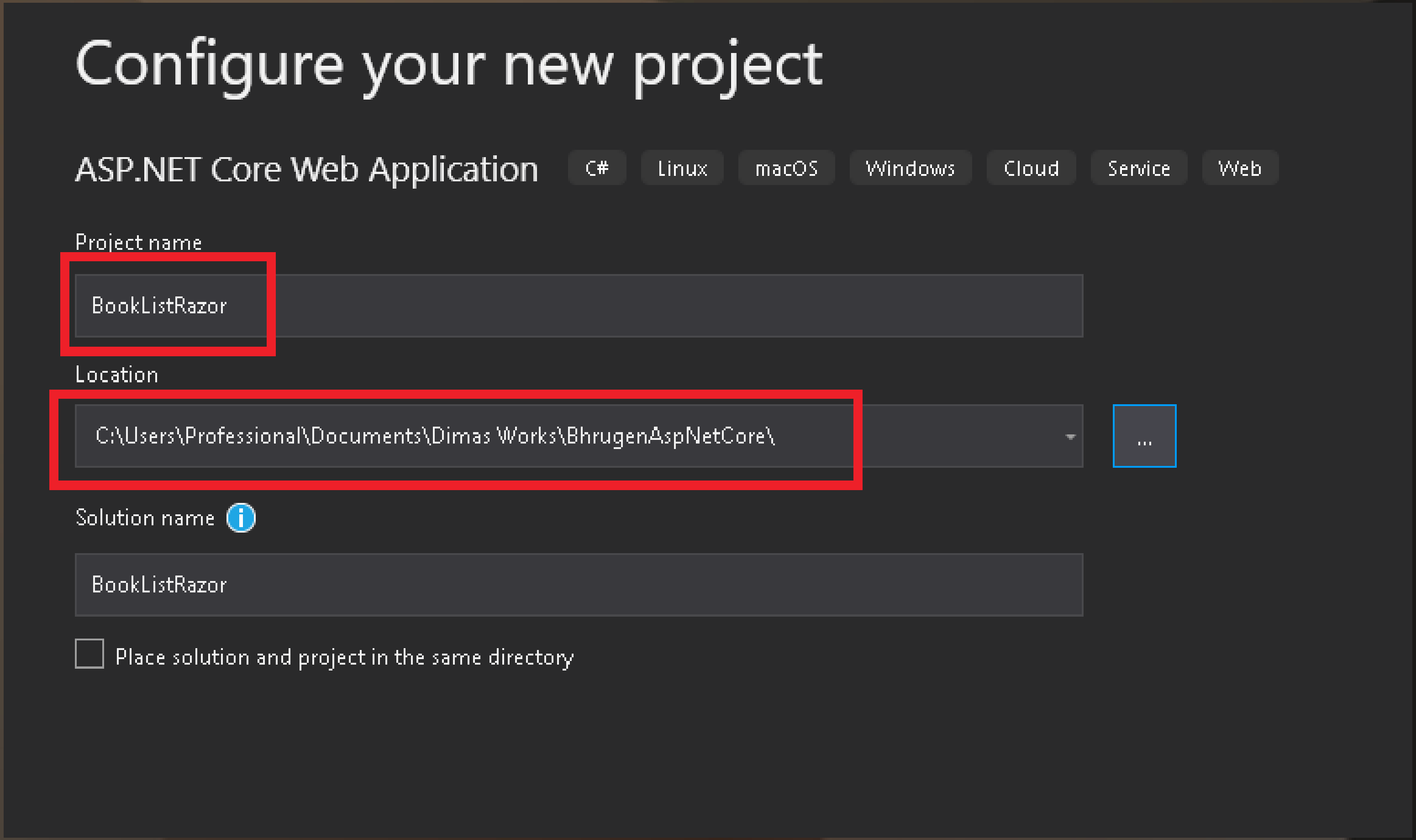
1.Open Visual Studio and create a new project
Select Asp.net Core Web Application



Create a new project
Choose a project template with code scaffolding to get started

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

Name your project, and select appropriate location


Configure your new project

ASP.NET Core Web Application

Project name

BookListRazor

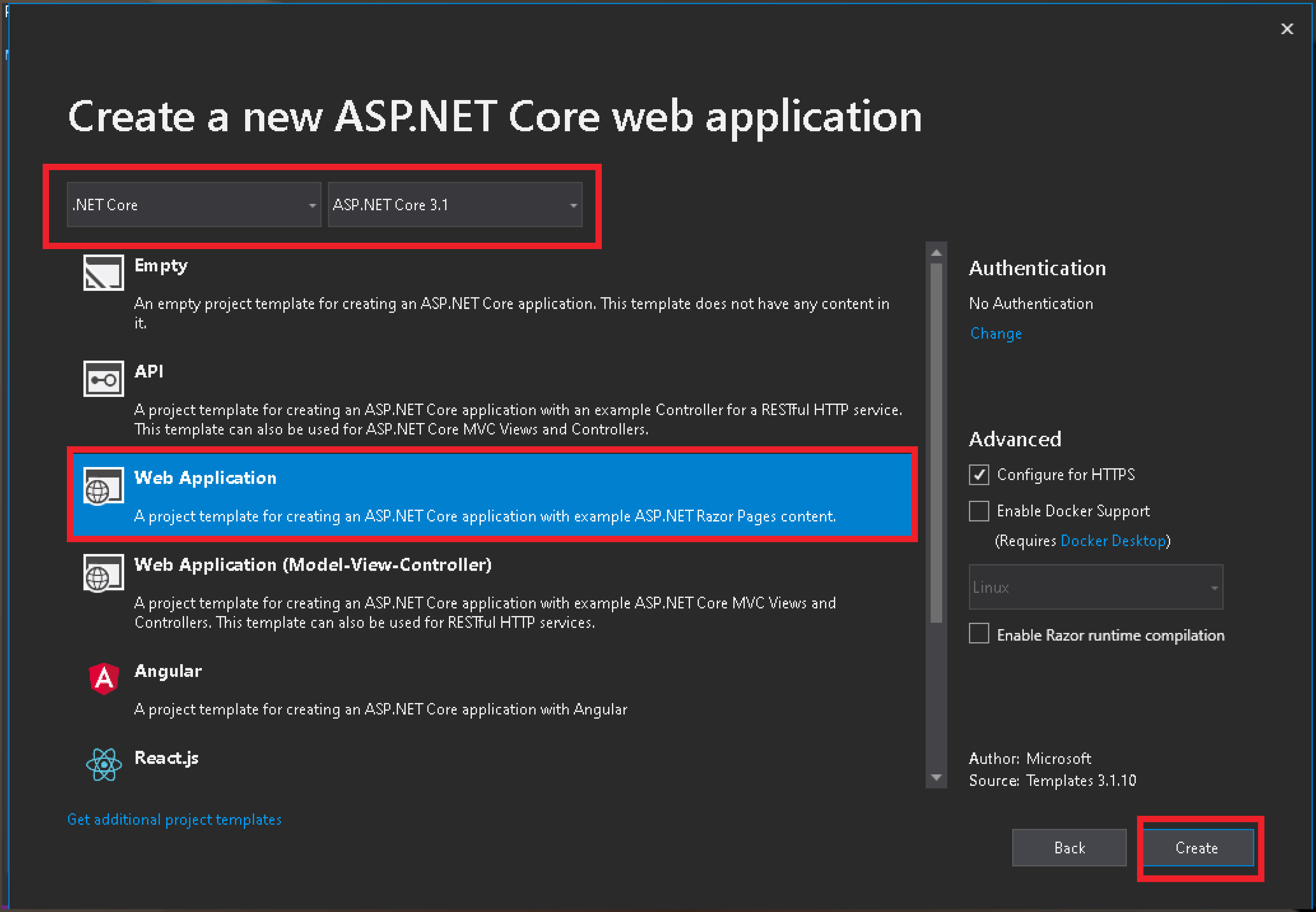
Location

C:\Users\Professional\Documents\Dimas Works\BhrugenAspNetCore\

Solution name

BookListRazor

☐ Place solution and project in the same directory



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

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

Web Application (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.

Angular
A project template for creating an ASP.NET Core application with Angular

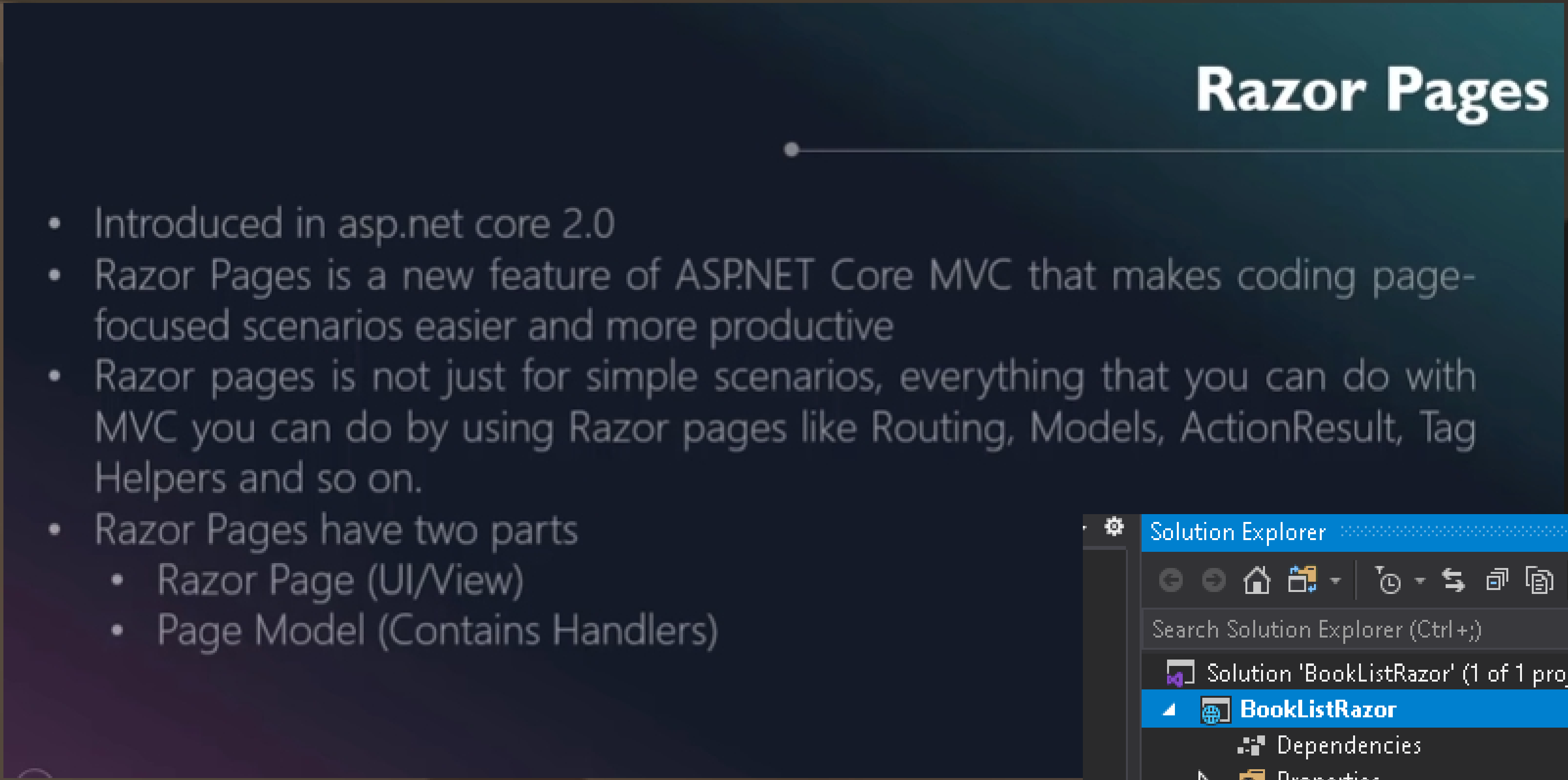
React.js

Get additional project templates

Authentication
No Authentication
Change

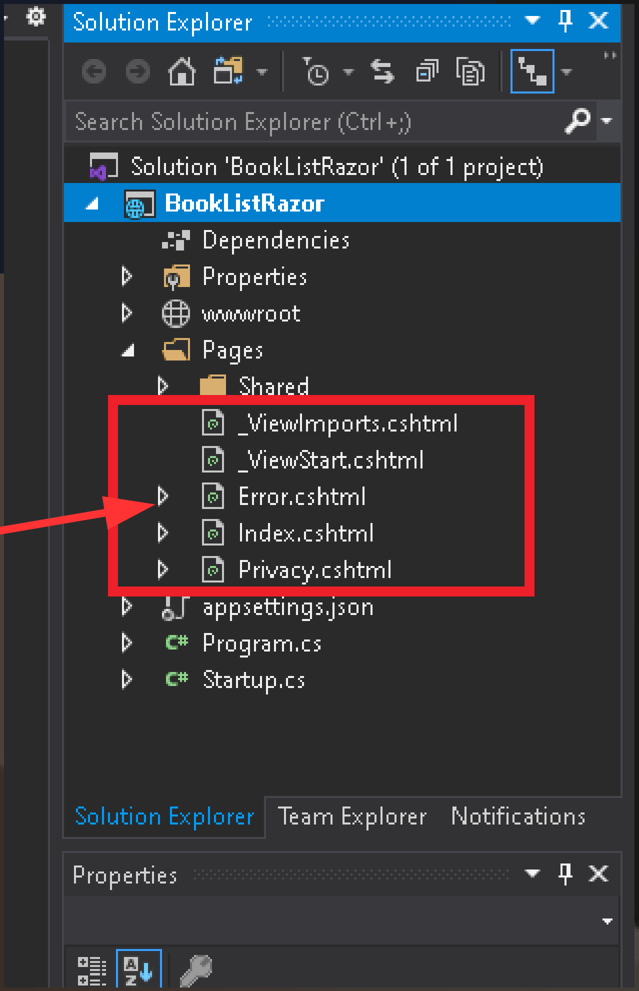
Advanced
☑ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Linux
☐ Enable Razor runtime compilation

Author: Microsoft
Source: Templates 3.1.10



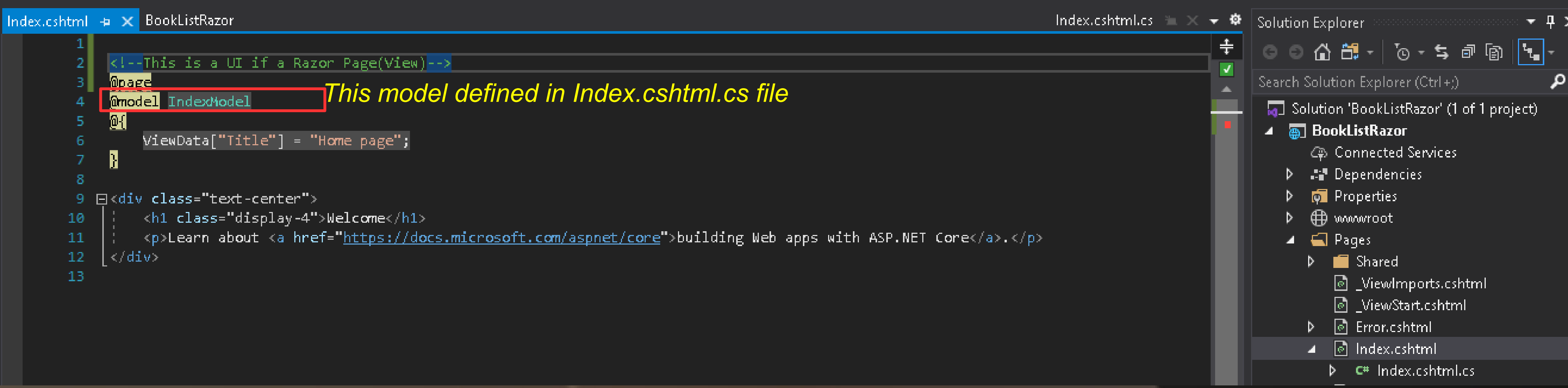
Razor Pages

- Introduced in asp.net core 2.0
- Razor Pages is a new feature of ASP.NET Core MVC that makes coding page-focused scenarios easier and more productive
- Razor pages is not just for simple scenarios, everything that you can do with MVC you can do by using Razor pages like Routing, Models, ActionResult, Tag Helpers and so on.
- Razor Pages have two parts
  - Razor Page (UI/View)
  - Page Model (Contains Handlers)



Razor pages

Each razor page has .cs file
Each razor file represents a a UI, or a view like page.

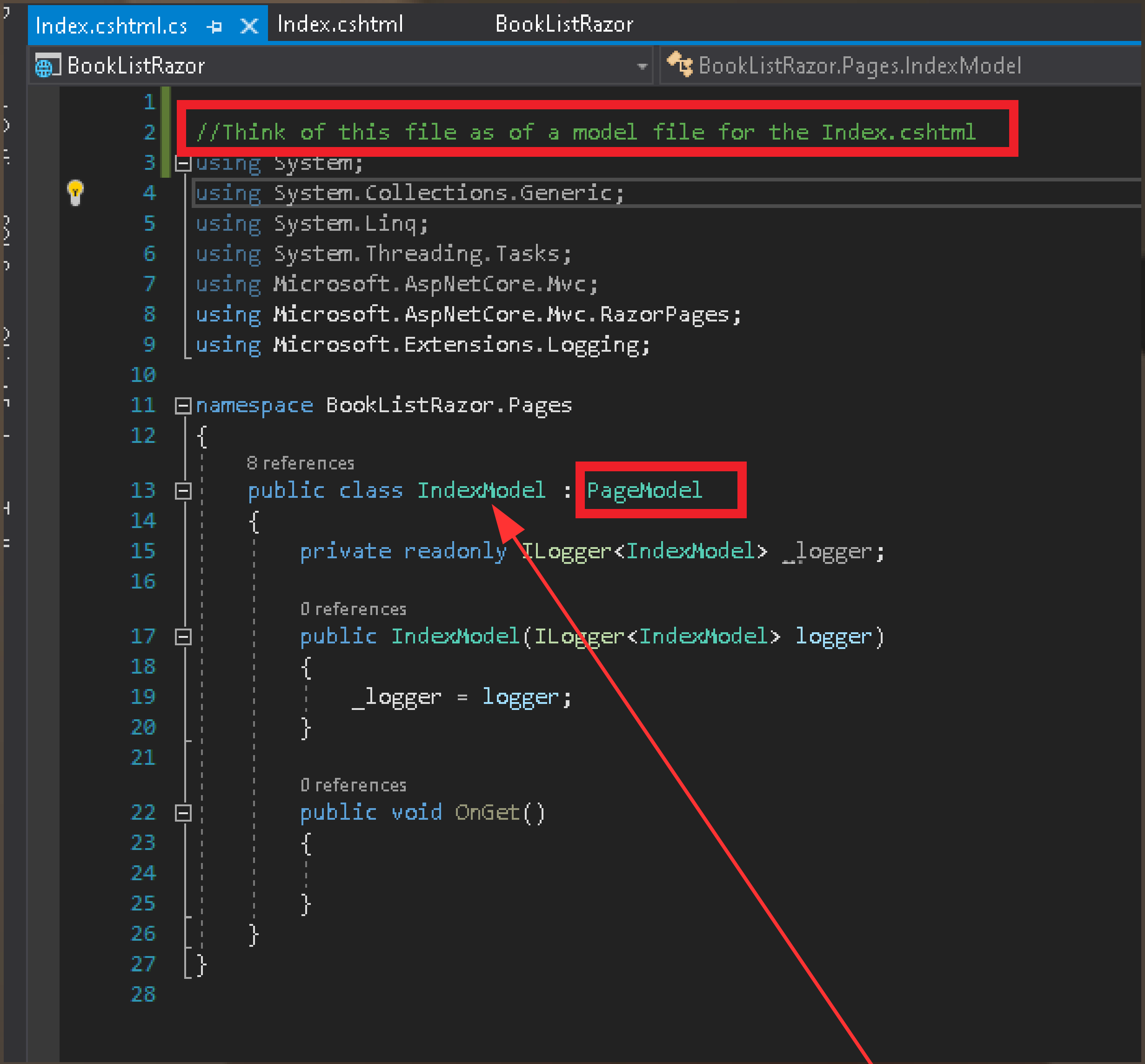


```

<!--This is a UI if a Razor Page(View)-->
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>

```

This model defined in Index.cshtml.cs file

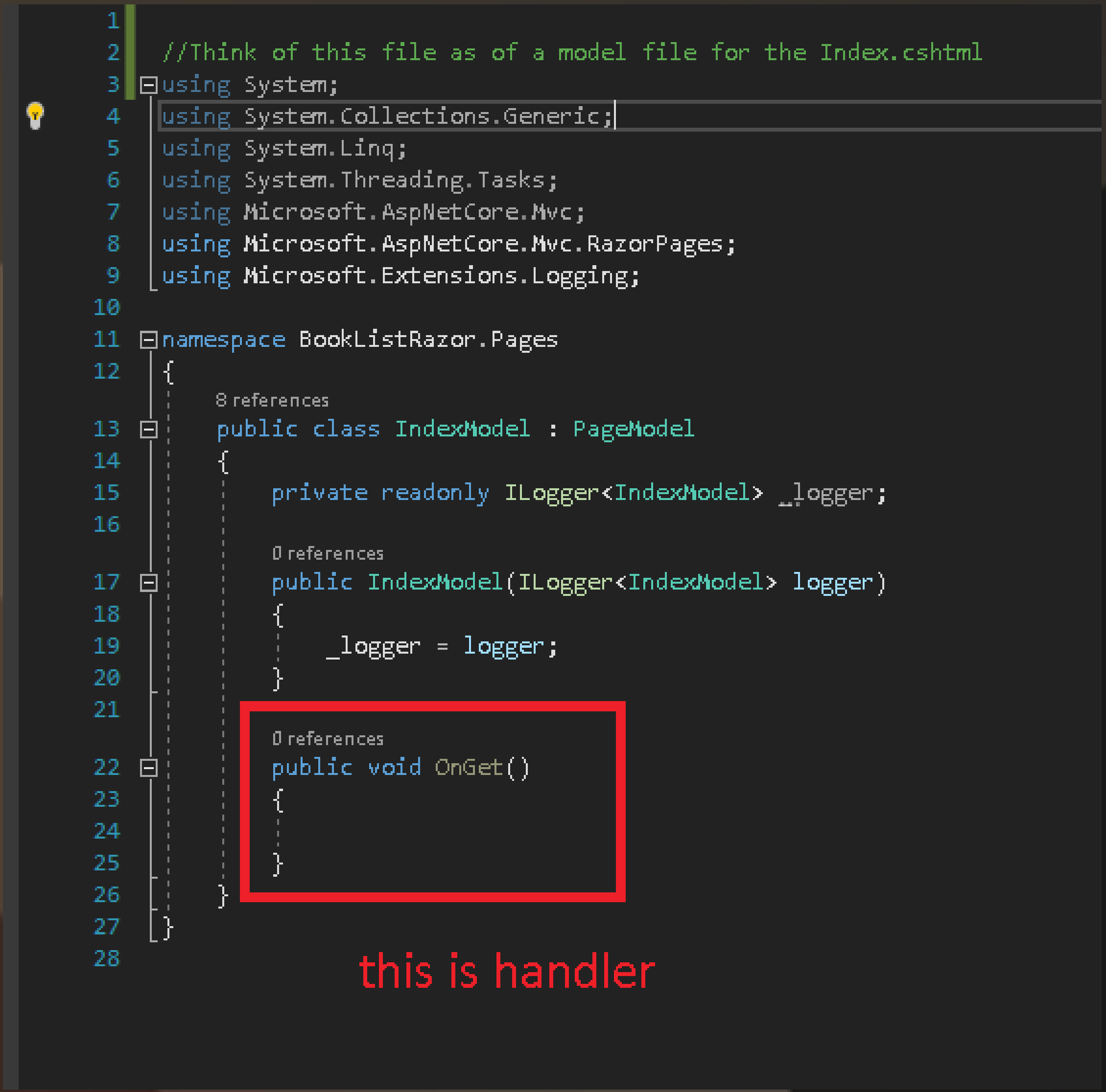
Each razol file has .cs file which is a model of a Razor file
So the index'.cshtml model is index.cshtml.cs file

Index'.cshtml model class



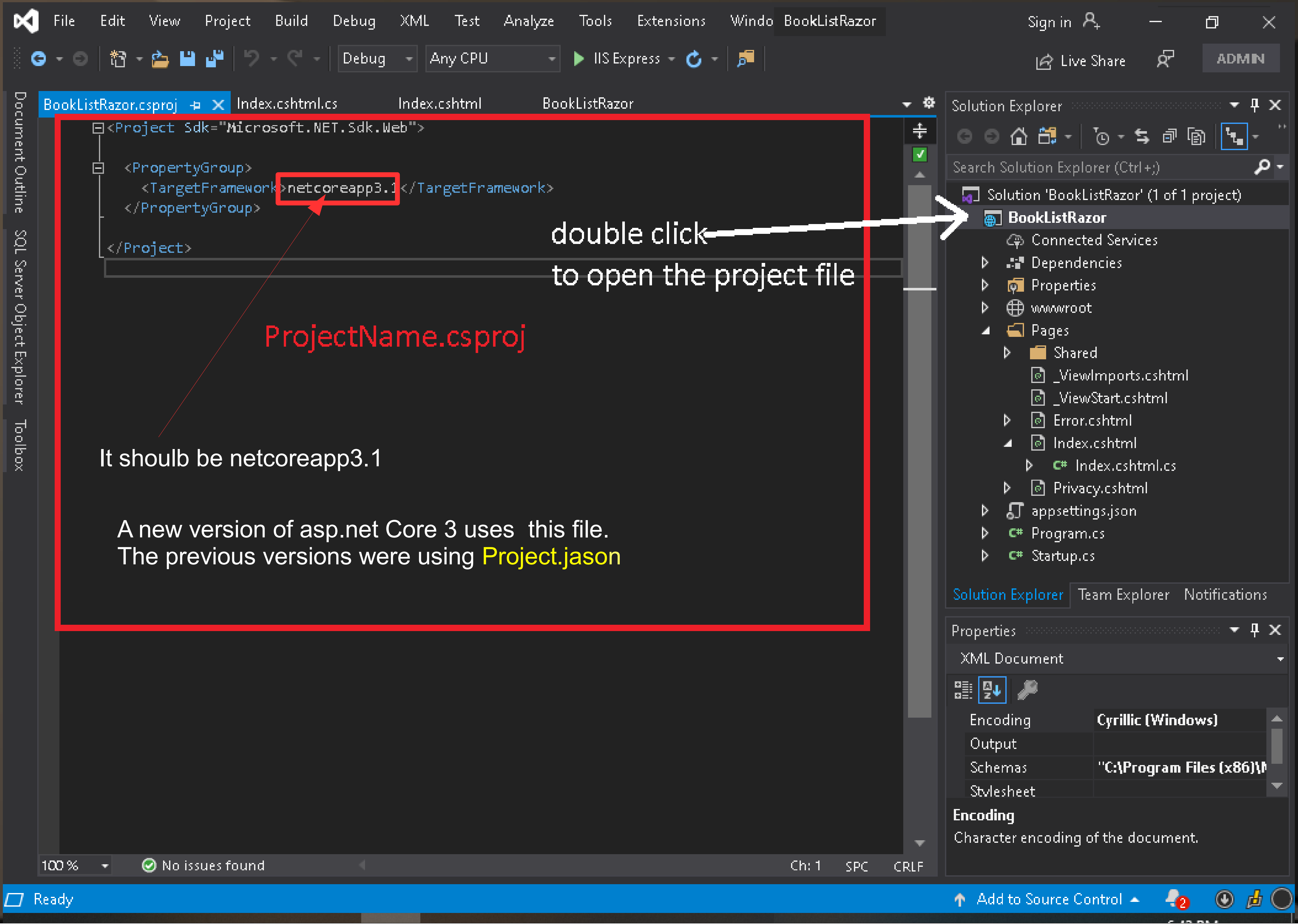
```
//Think of this file as of a model file for the Index.cshtml
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace BookListRazor.Pages
{
    8 references
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        0 references
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        0 references
        public void OnGet()
        {

        }
    }
}
```

Inside this model class we will define a model for view index.cshtml view



```

//Think of this file as of a model file for the Index.cshtml
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace BookListRazor.Pages
{
    8 references
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        0 references
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        0 references
        public void OnGet()
        {

        }
    }
}

```

Let's open project configuration file



```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

</Project>
```

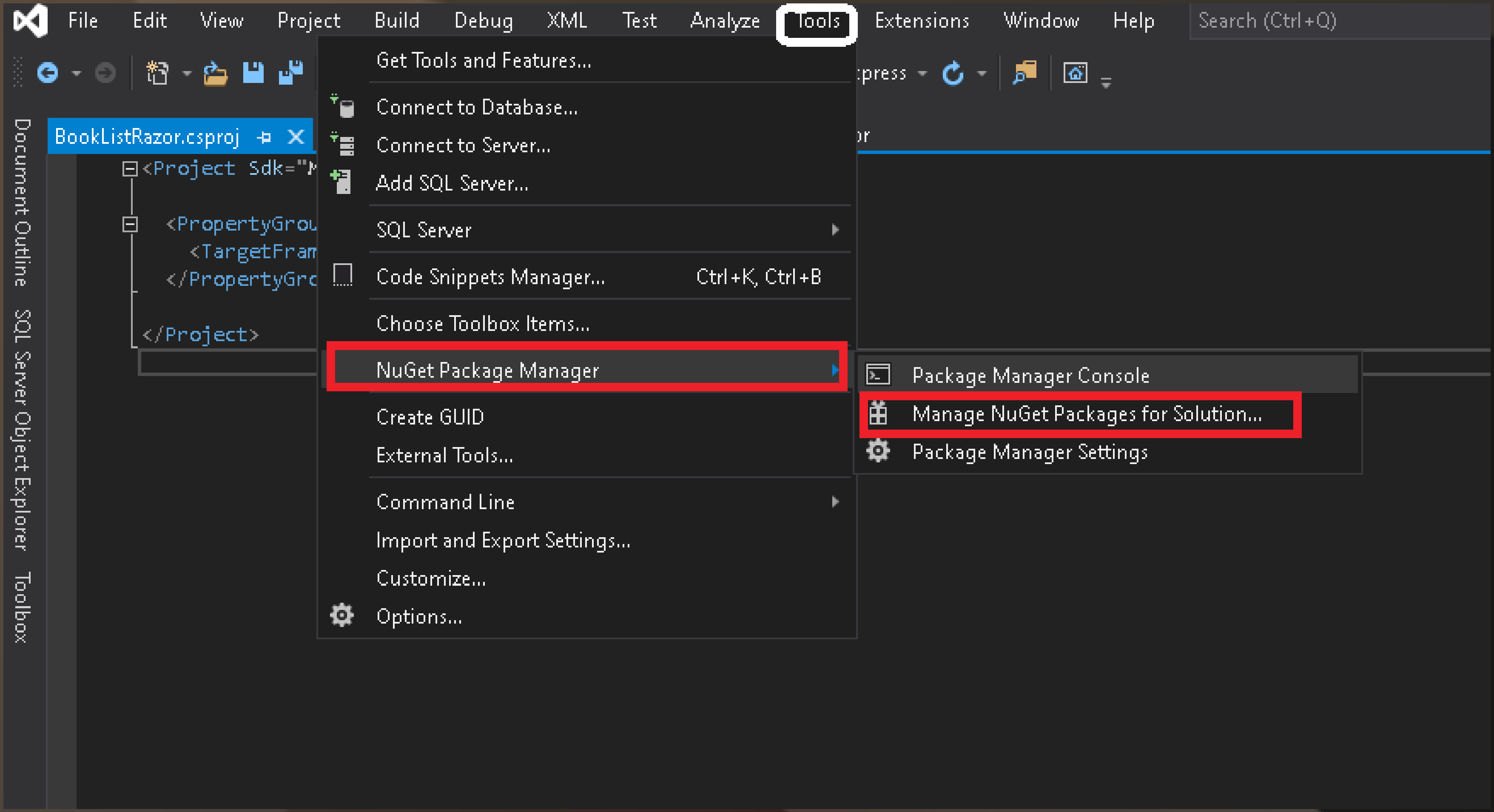
double click
to open the project file

ProjectName.csproj

It shoulb be netcoreapp3.1

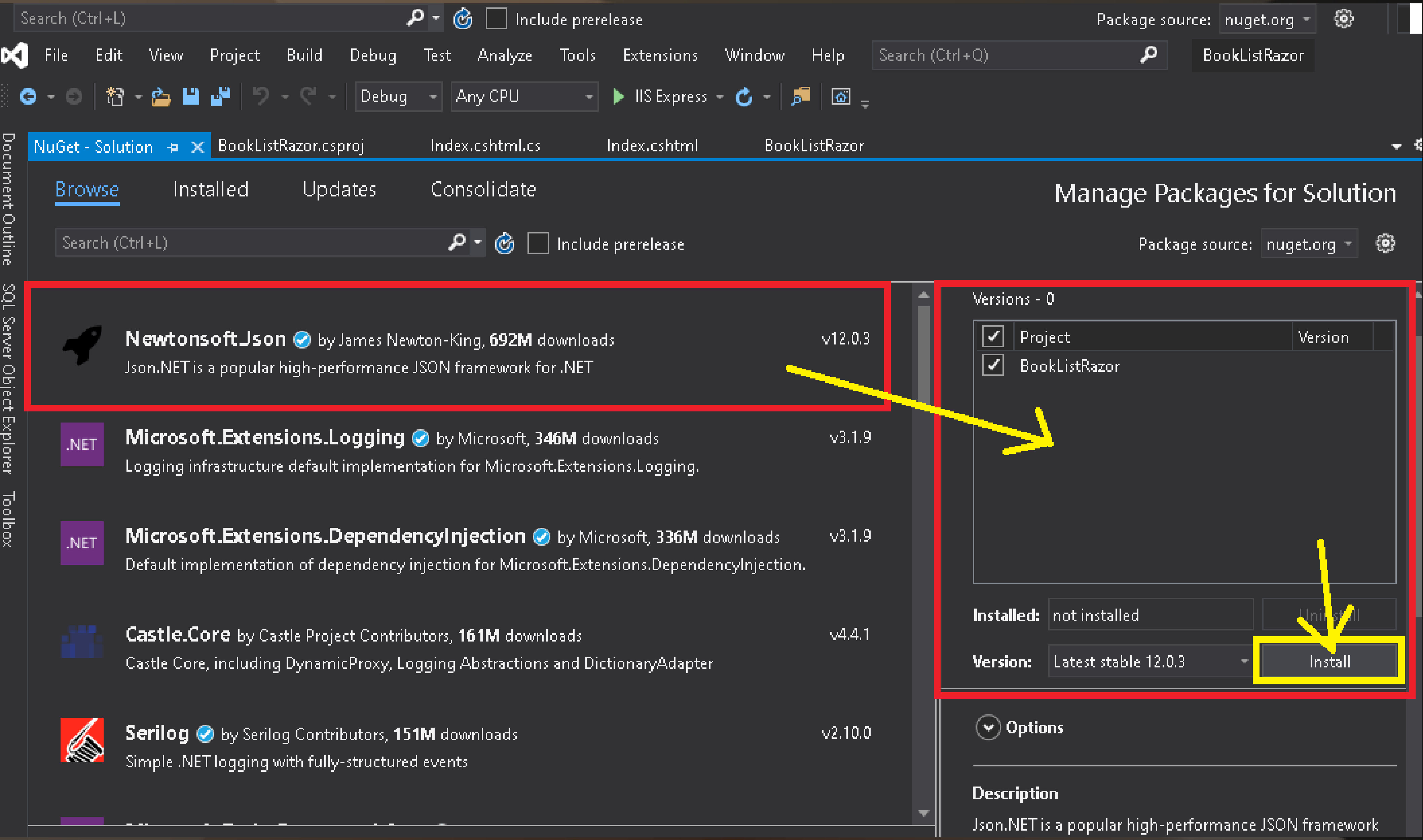
A new version of asp.net Core 3 uses this file.
The previous versions were using Project.jason

Later on we will get additiopnal packages to our solution via Nuget Package Manager



```
<Project Sdk="M
  <PropertyGrou
    <TargetFram
  </PropertyGro
</Project>
```

FOR A DEMO PURPOSE LET'S IMSTALL Newtonsoft.json
We will Uninstall this package later on.



Newtonsoft.Json by James Newton-King, 692M downloads v12.0.3
Json.NET is a popular high-performance JSON framework for .NET

Microsoft.Extensions.Logging by Microsoft, 346M downloads v3.1.9
Logging infrastructure default implementation for Microsoft.Extensions.Logging.

Microsoft.Extensions.DependencyInjection by Microsoft, 336M downloads v3.1.9
Default implementation of dependency injection for Microsoft.Extensions.DependencyInjection.

Castle.Core by Castle Project Contributors, 161M downloads v4.4.1
Castle Core, including DynamicProxy, Logging Abstractions and DictionaryAdapter

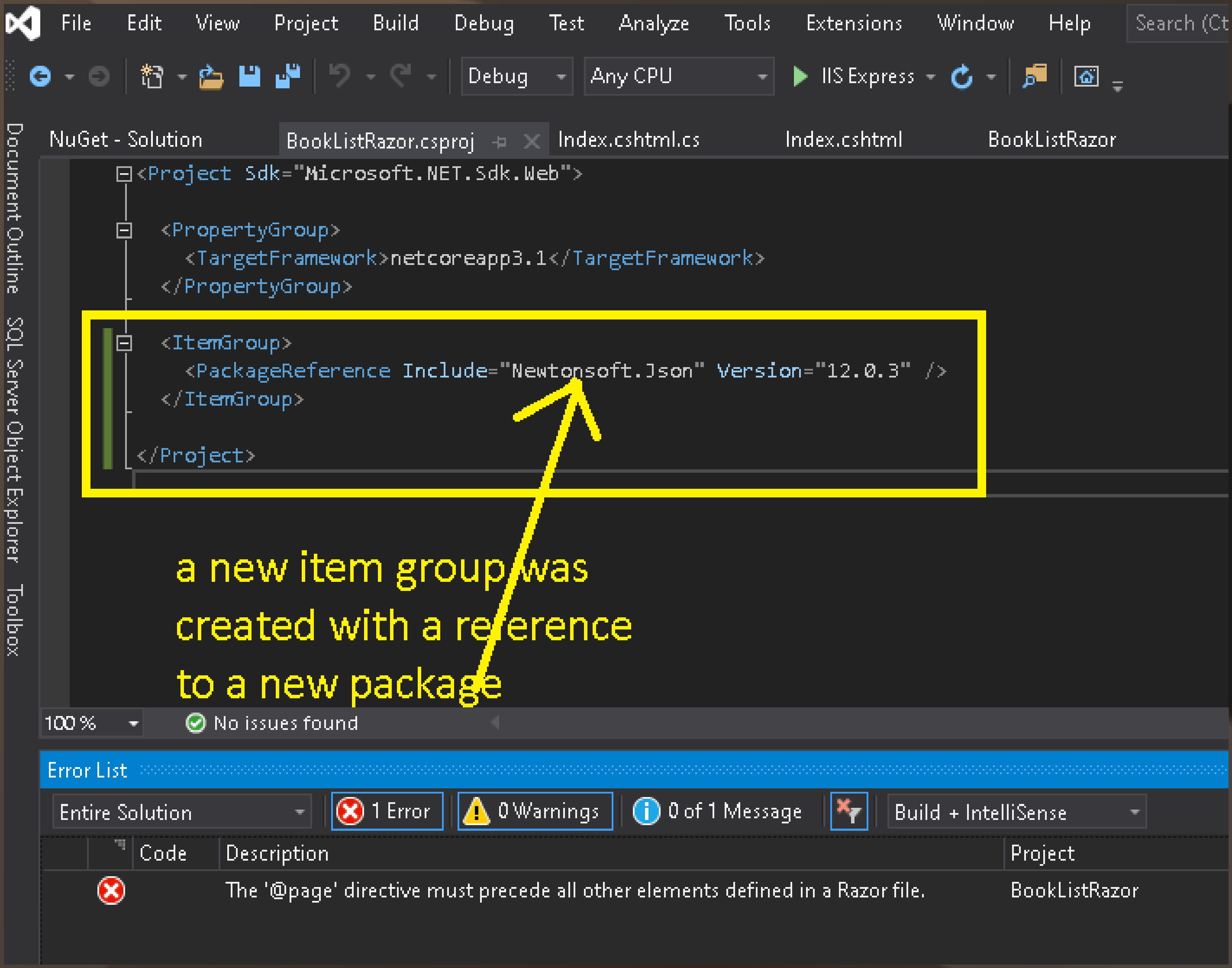
Serilog by Serilog Contributors, 151M downloads v2.10.0
Simple .NET logging with fully-structured events

Versions - 0

	Project	Version
✓	BookListRazor	

Installed: not installed
Version: Latest stable 12.0.3

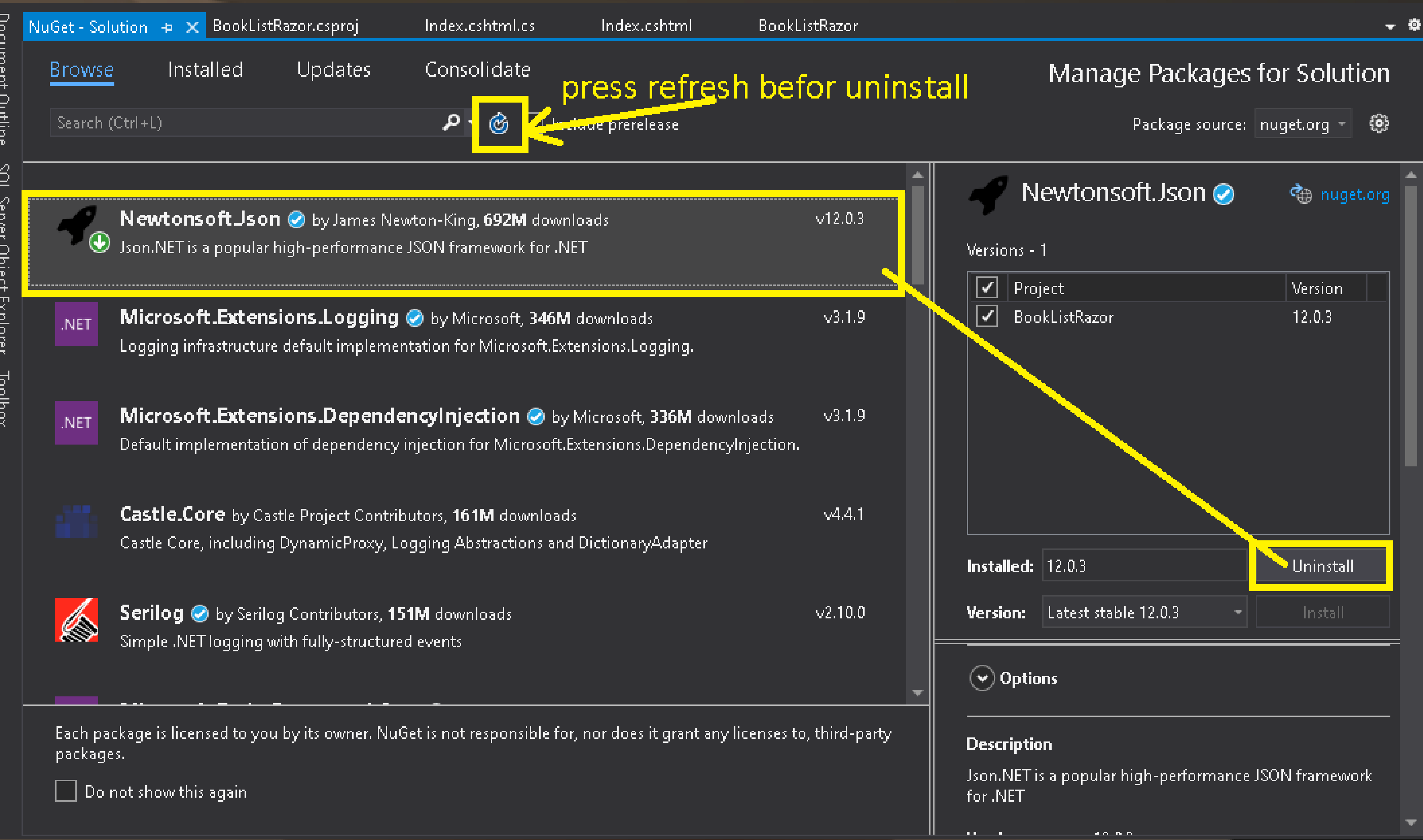
Description
Json.NET is a popular high-performance JSON framework



```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="12.0.3" />
  </ItemGroup>

</Project>
```

Code	Description	Project
	The '@page' directive must precede all other elements defined in a Razor file.	BookListRazor

Everytime you add a package to your project this package reference will be added here


NuGet - Solution | BookListRazor.csproj | Index.cshtml.cs | Index.cshtml | BookListRazor

Browse | Installed | Updates | Consolidate

Manage Packages for Solution

press refresh befor uninstall

Search (Ctrl+L)

Include prerelease

Package source: nuget.org

Newtonsoft.Json by James Newton-King, **692M** downloads v12.0.3
Json.NET is a popular high-performance JSON framework for .NET

Microsoft.Extensions.Logging by Microsoft, **346M** downloads v3.1.9
Logging infrastructure default implementation for Microsoft.Extensions.Logging.

Microsoft.Extensions.DependencyInjection by Microsoft, **336M** downloads v3.1.9
Default implementation of dependency injection for Microsoft.Extensions.DependencyInjection.

Castle.Core by Castle Project Contributors, **161M** downloads v4.4.1
Castle Core, including DynamicProxy, Logging Abstractions and DictionaryAdapter

Serilog by Serilog Contributors, **151M** downloads v2.10.0
Simple .NET logging with fully-structured events

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Newtonsoft.Json nuget.org

Versions - 1

✓	Project	Version
✓	BookListRazor	12.0.3

Installed: 12.0.3 Uninstall

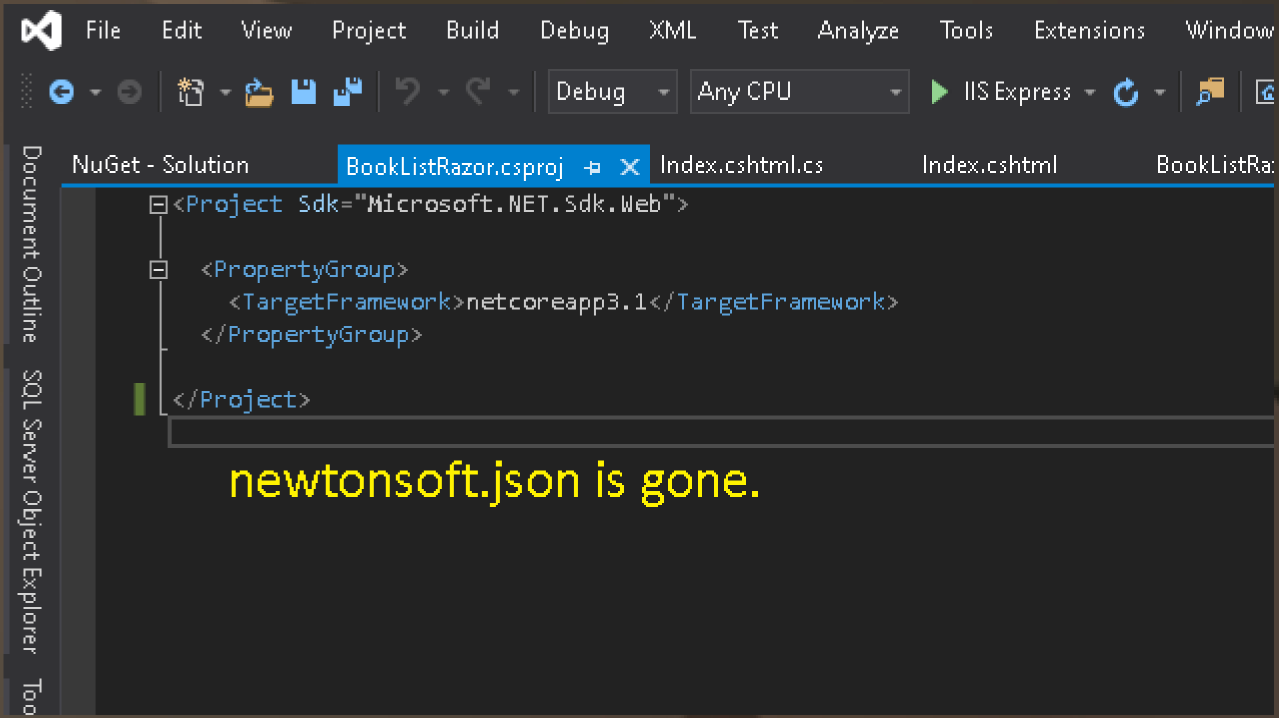
Version: Latest stable 12.0.3 Install

Options

Description

Json.NET is a popular high-performance JSON framework for .NET

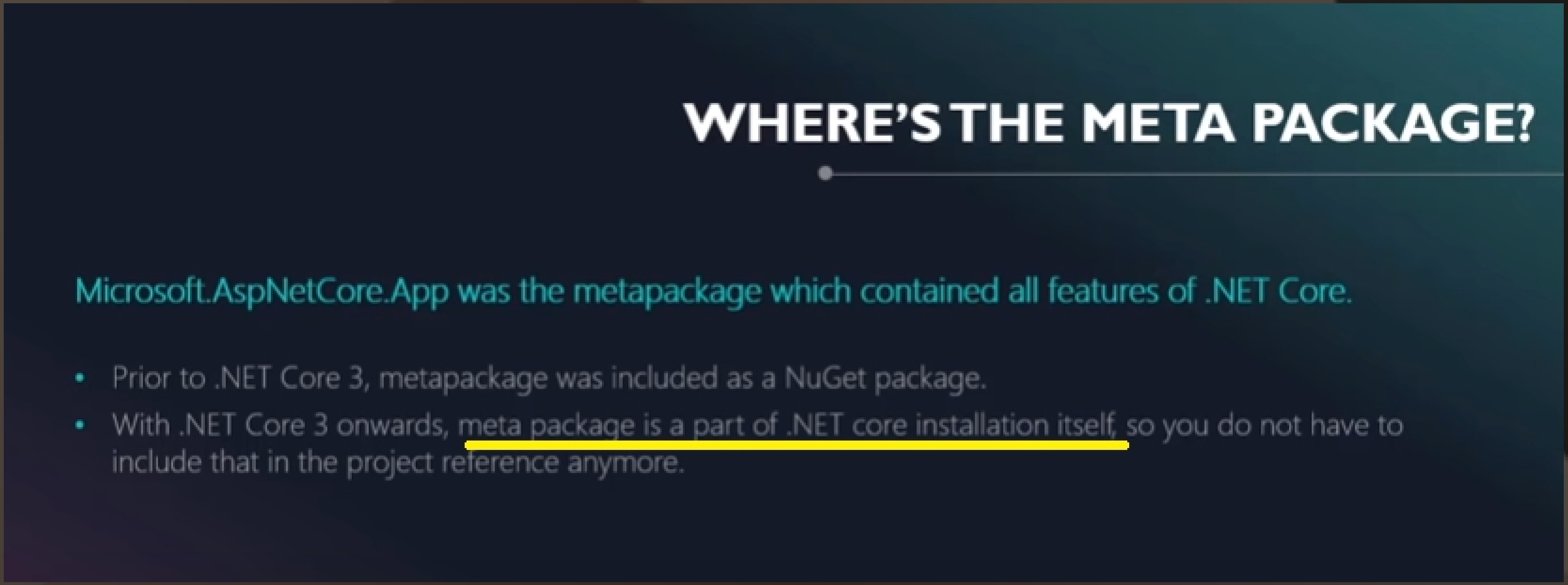
After uninstalling newtonsoft.json



```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

</Project>
```

newtonsoft.json is gone.

The previous Asp.net core project were using a MetaPackage and what is this all about?

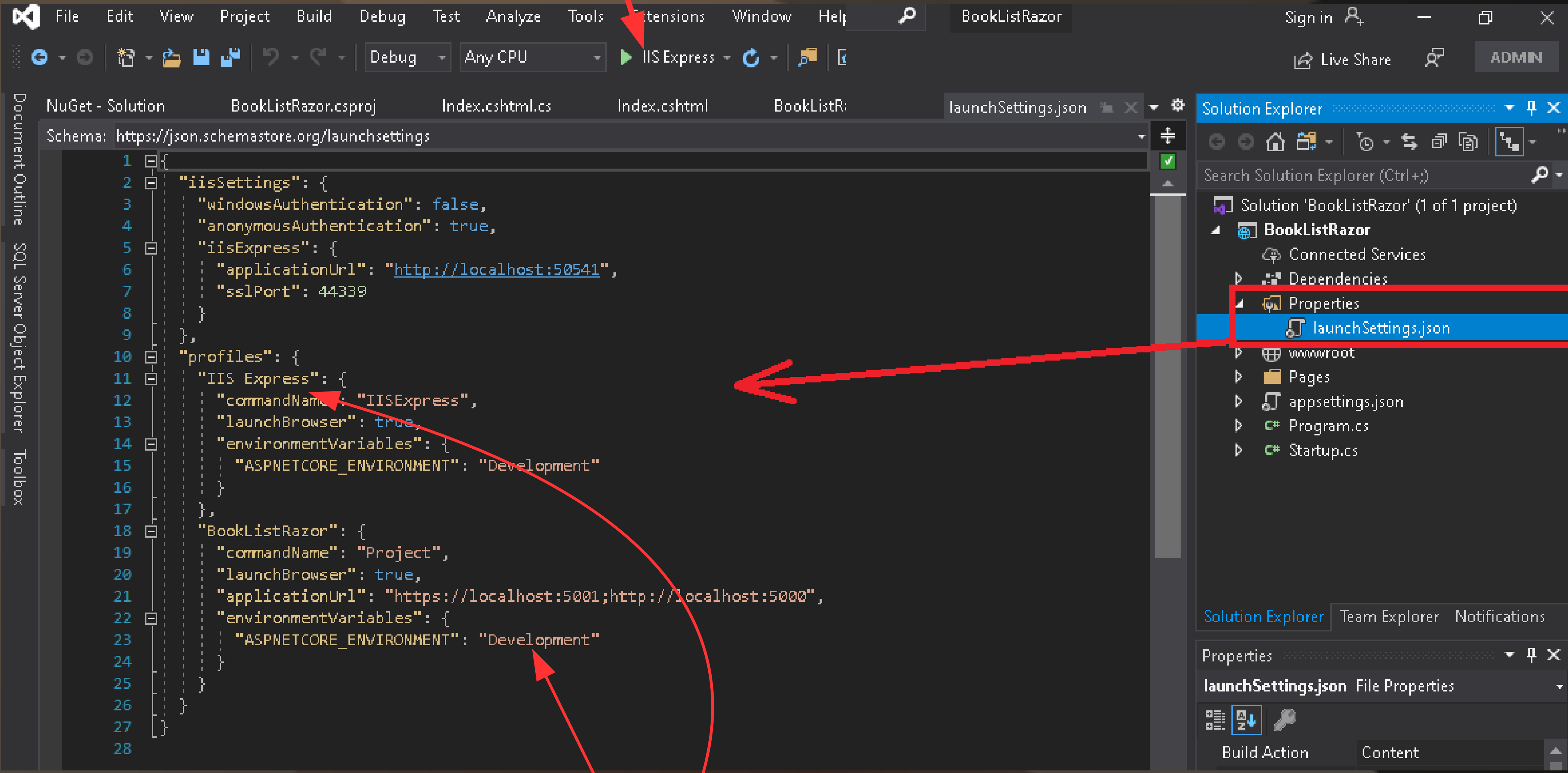


WHERE'S THE META PACKAGE?

Microsoft.AspNetCore.App was the metapackage which contained all features of .NET Core.

- Prior to .NET Core 3, metapackage was included as a NuGet package.
- With .NET Core 3 onwards, meta package is a part of .NET core installation itself, so you do not have to include that in the project reference anymore.

Expand a Properties Folder in out soulution explorer
LaunchSettings.json this file tels a visual studio what to do
When you press a **run button**

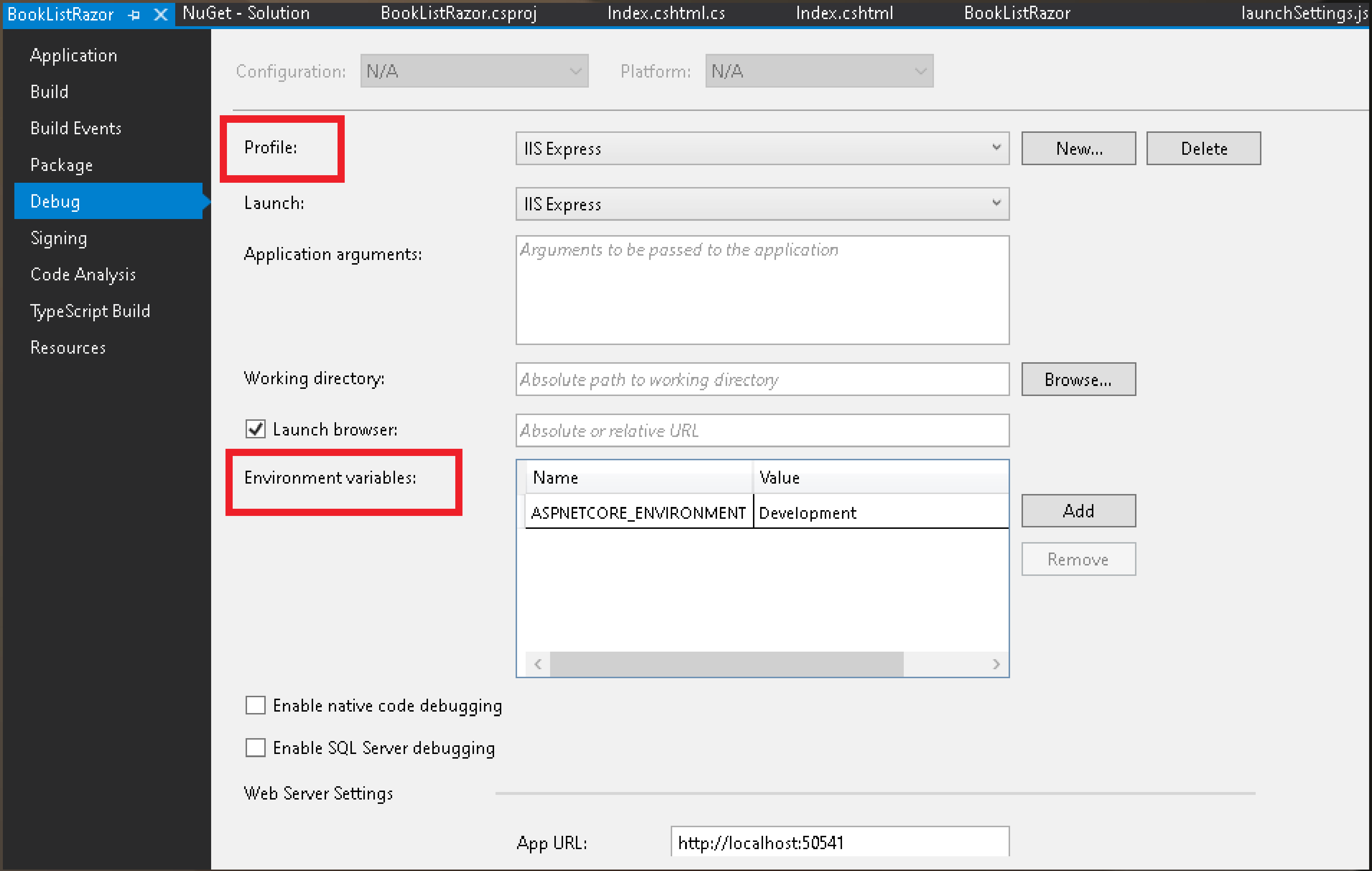


```json
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:50541",
      "sslPort": 44339
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "BookListRazor": {
      "commandName": "Project",
      "launchBrowser": true,
      "applicationUrl": "https://localhost:5001;http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

By default we have a few profiles here: 1st is IIS which will hoste the application and launch a web browser.
It will also set the enviromentvariable to Development
We will load a full CSS file, or a minified version of a CSS depending on environmentVariables!

Another way to change these settings is via UI designer

1.Right click you project and select properties
2. Open Debug Tab
3. You can change Profile, and Environment variables as needed. **We will not be altering any of this now**



Configuration: N/A
Platform: N/A

Profile: IIS Express

Launch: IIS Express

Application arguments: *Arguments to be passed to the application*

Working directory: *Absolute path to working directory*

☑ Launch browser: *Absolute or relative URL*

Environment variables:

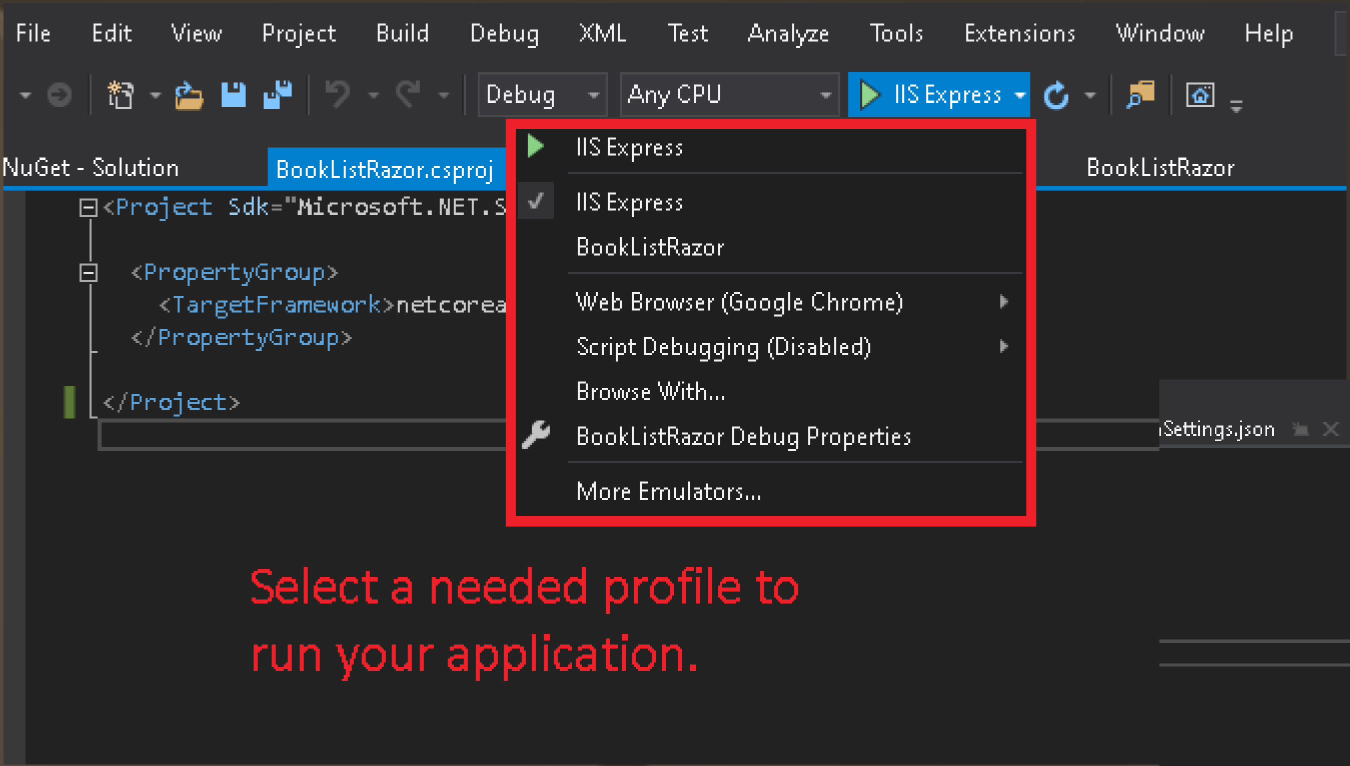
Name	Value
ASPNETCORE_ENVIRONMENT	Development

☐ Enable native code debugging

☐ Enable SQL Server debugging

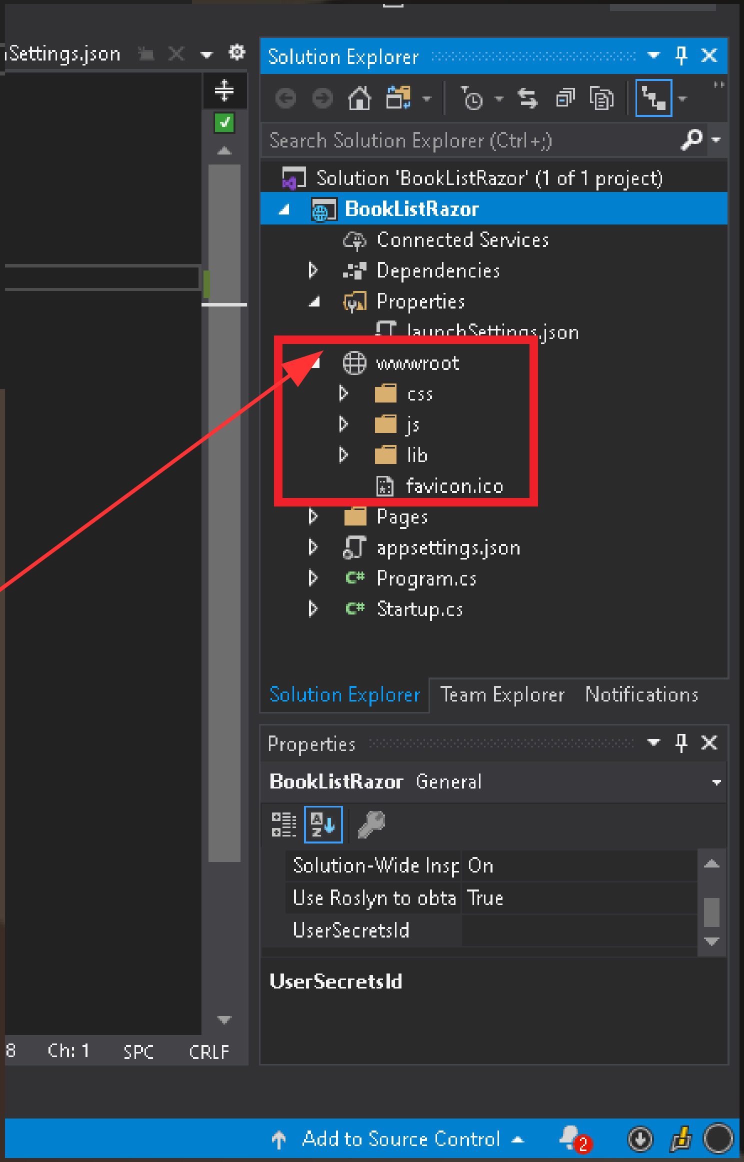
Web Server Settings

App URL: http://localhost:50541



```
<Project Sdk="Microsoft.NET.S

  <PropertyGroup>
    <TargetFramework>netcorea
  </PropertyGroup>

</Project>
```

Select a needed profile to run your application.



Next we'll expand a wwwroot folder

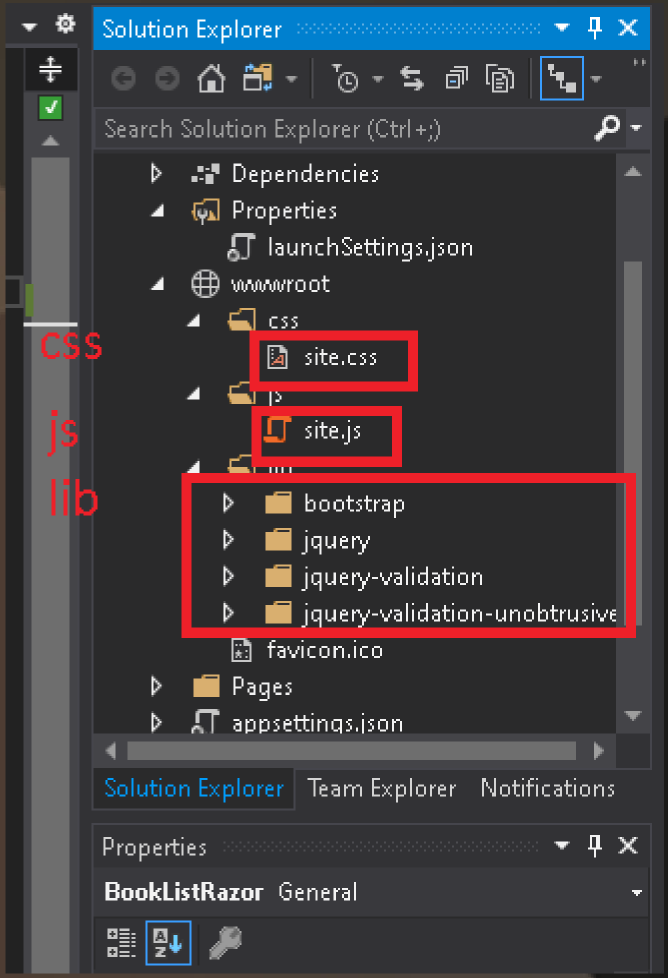
CSS
Js
Lib
These folders are new to asp.net core 3
And they being create automatically
This folder is a root folder for our website
All the static images, and html files will be placed inside
This folder.
You should't be placing any Razor, or Csharp files here!

Visual studio created a few static files for us while creating the project.

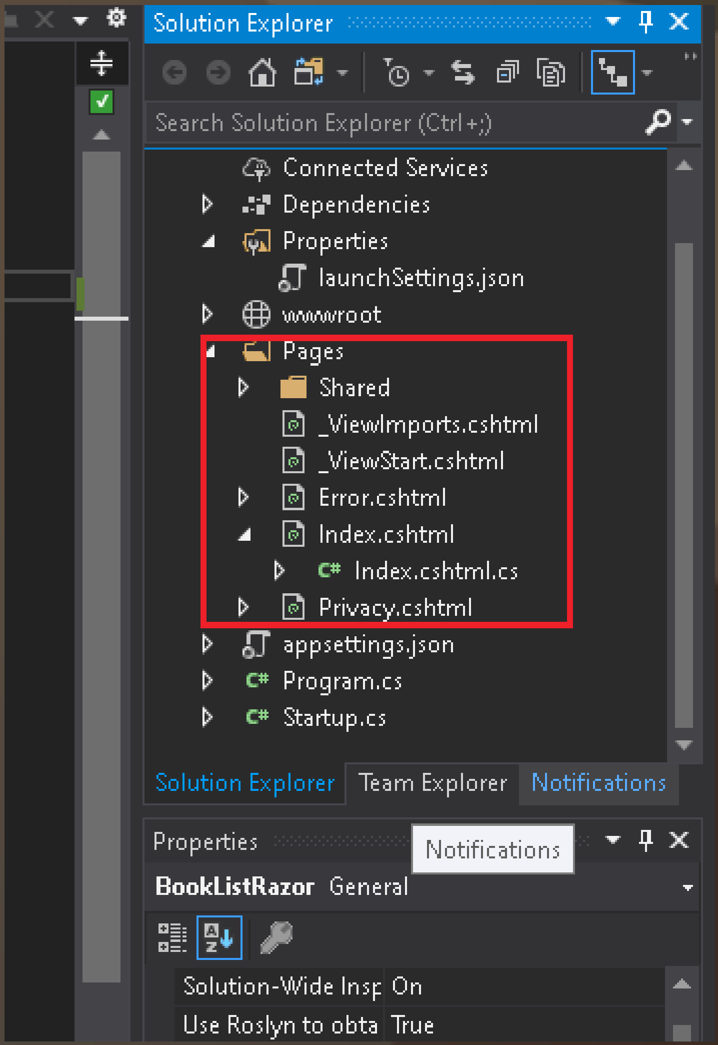
The reason we have these files is because
We've created our application as a **Razor Page**

If we select Empty application we would
Enter these files by ourselves.

When we will be adding more CSS, or JavaScript
We will adding them inside this wwwroot folder



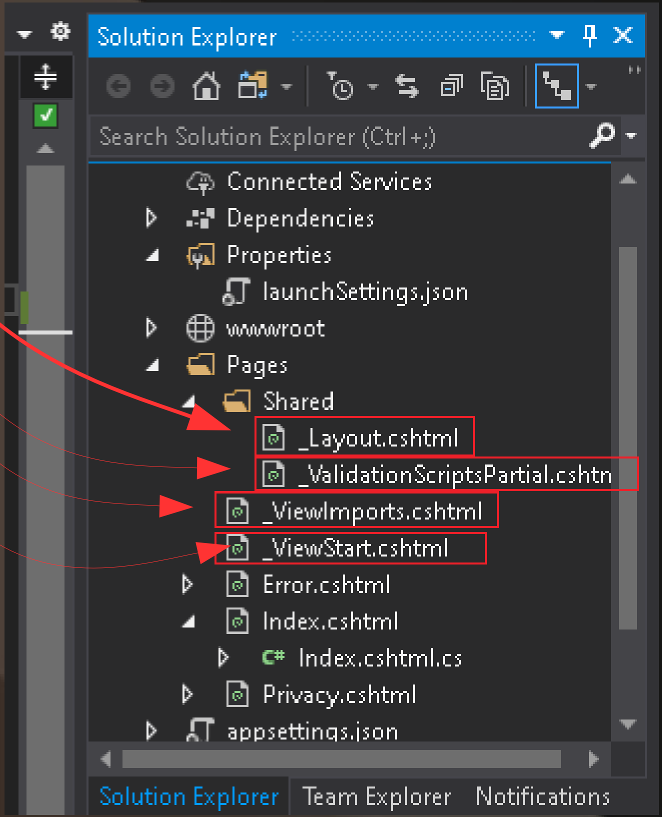
Next Let's see what is inside a **Pages** Folder


Solution Explorer
Search Solution Explorer (Ctrl+;)
Connected Services
Dependencies
Properties
launchSettings.json
wwwroot
Pages
Shared
_ViewImports.cshtml
_ViewStart.cshtml
Error.cshtml
Index.cshtml
Index.cshtml.cs
Privacy.cshtml
appsettings.json
Program.cs
Startup.cs
Solution Explorer Team Explorer Notifications
Properties Notifications
BookListRazor General
Solution-Wide Insp On
Use Roslyn to obta True

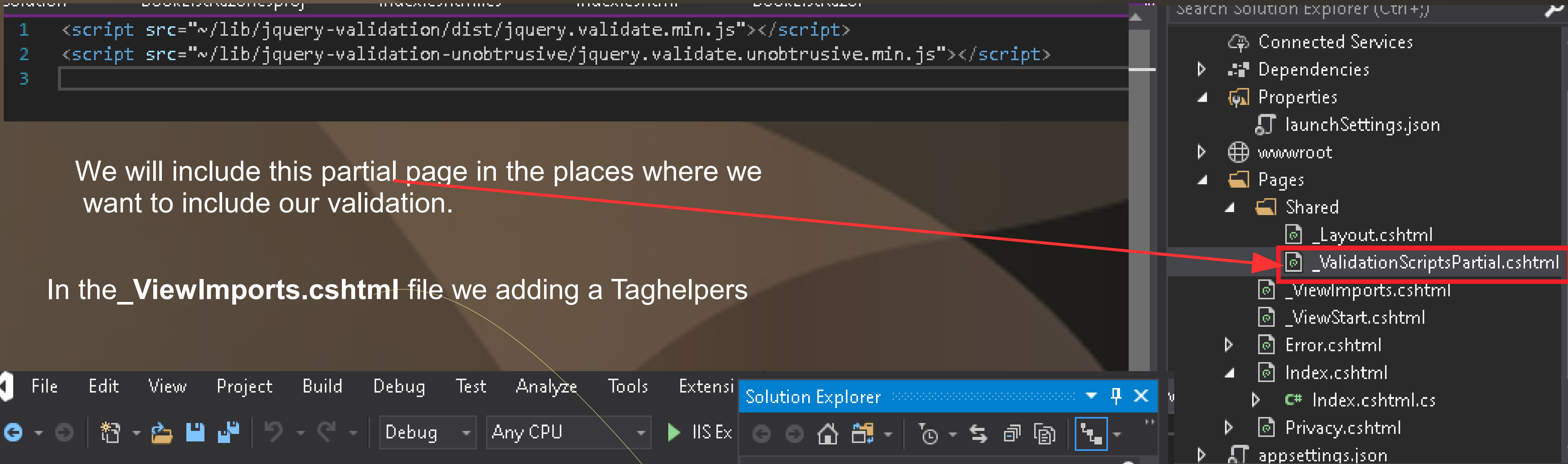
Pges folder is the main folder inside any Razor Projects.
The Pages starts with the underscore means these
Pages are a **Partial Views**
These like user components, and you can reuse them
Multiple times in your application.

_Layout.cshtml

Is a defaoult **MasterPage**
Of your application


Solution Explorer
Search Solution Explorer (Ctrl+;)
Connected Services
Dependencies
Properties
launchSettings.json
wwwroot
Pages
Shared
_Layout.cshtml
_ValidationScriptsPartial.cshtn
_ViewImports.cshtml
_ViewStart.cshtml
Error.cshtml
Index.cshtml
Index.cshtml.cs
Privacy.cshtml
appsettings.json
Solution Explorer Team Explorer Notifications

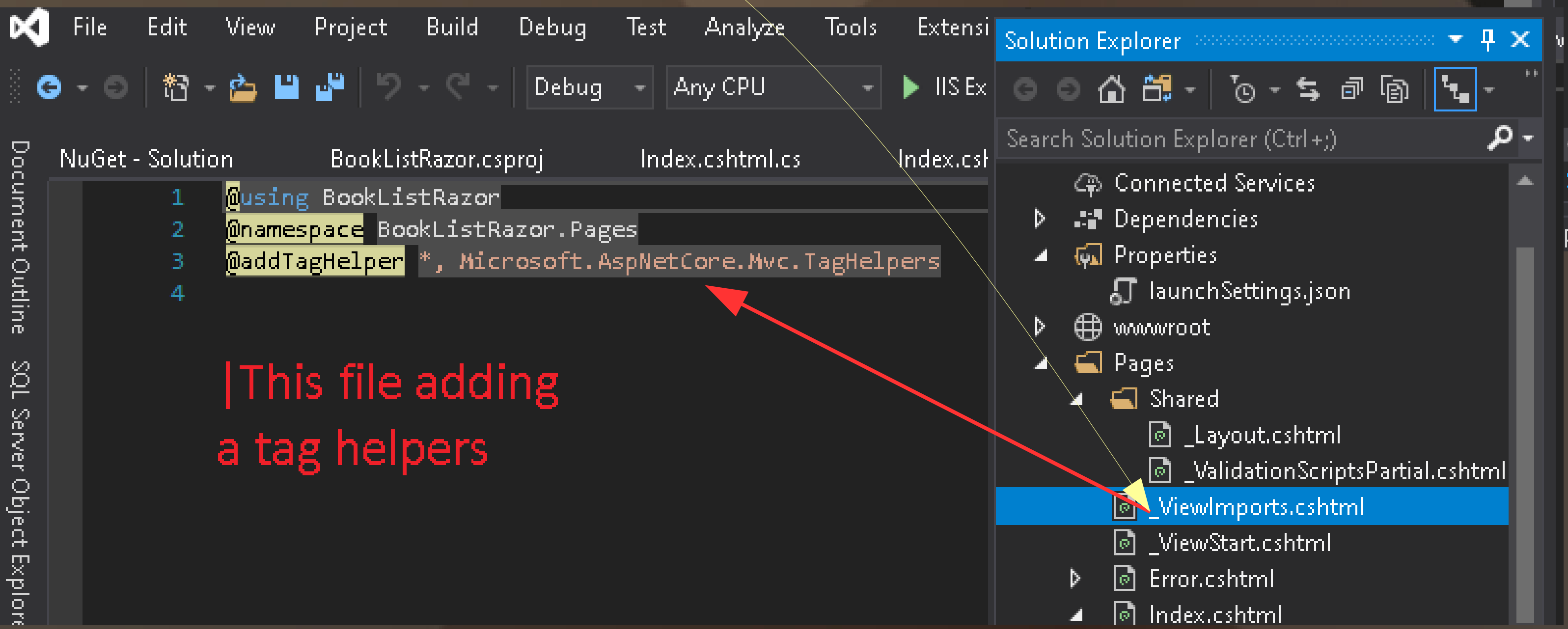
Next file in Shared folder is _ValidationScriptsPartial.cshtml



```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

We will include this partial page in the places where we want to include our validation.

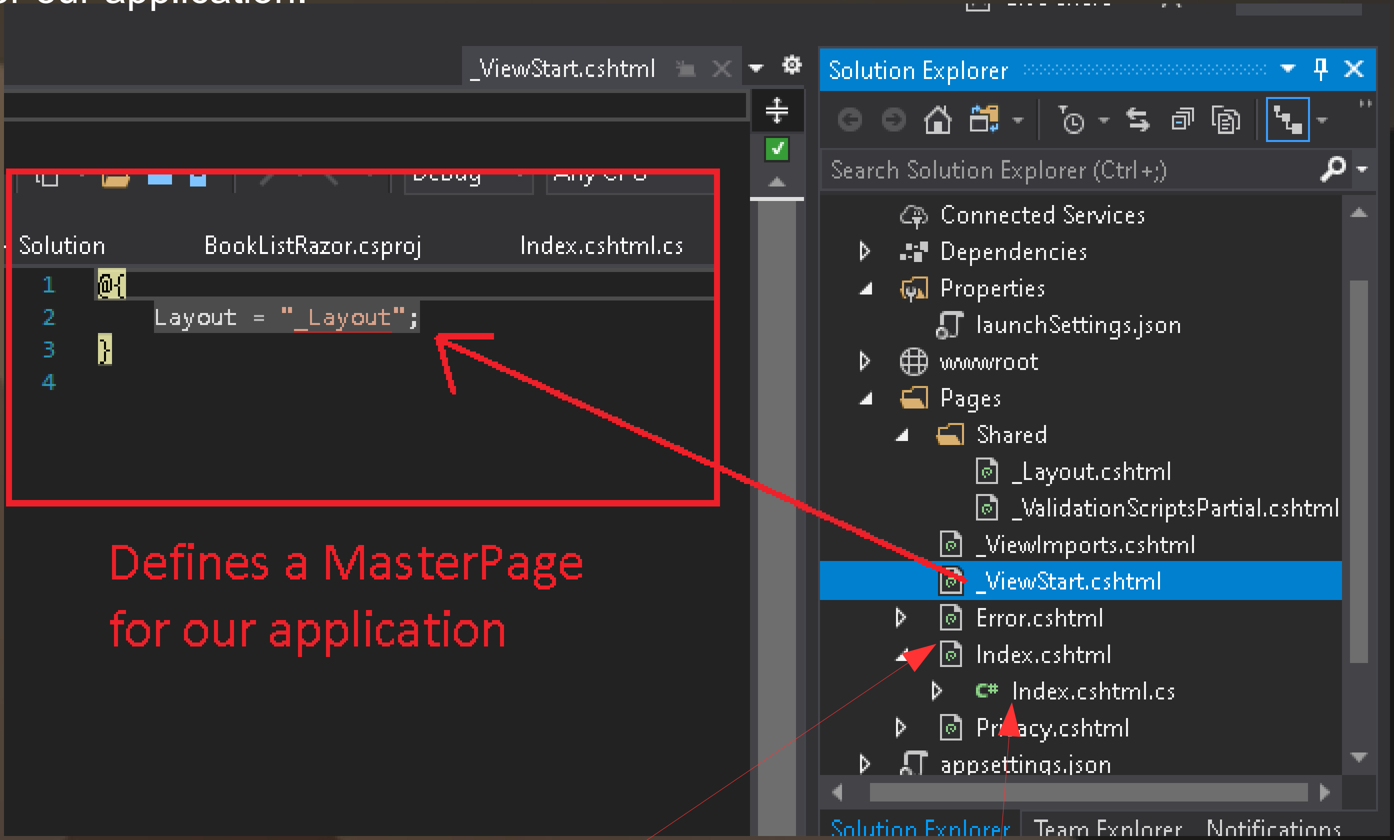
In the **_ViewImports.cshtml** file we adding a Taghelpers



```
@using BookListRazor
@namespace BookListRazor.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

This file adding a tag helpers

The next file _ViewStart.cshtml
Defines a Master Page for our application.



```
@{
    Layout = "_Layout";
}
```

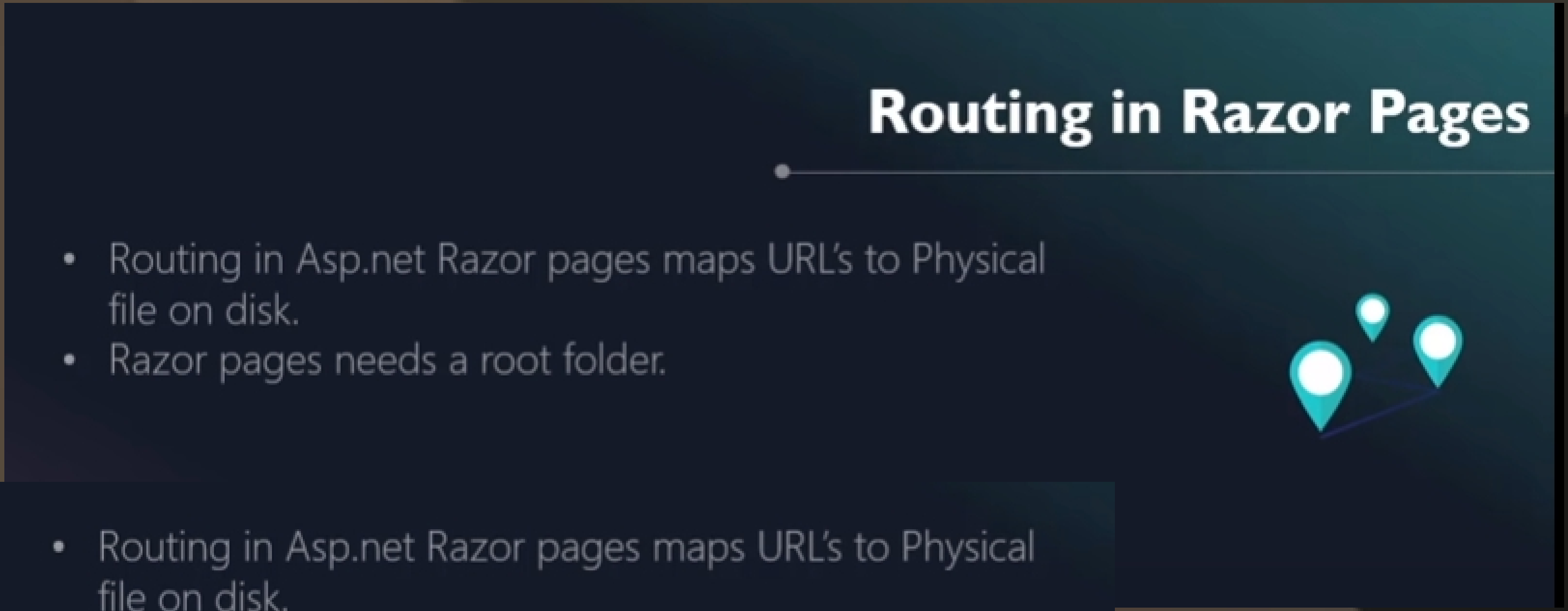
Defines a MasterPage for our application

Reminder:
As you can see in Asp.net Razor we don't have Controllers.

For example if we have **Index.cshtml**, so the code behind for this page
Will be **Index.cshtml**

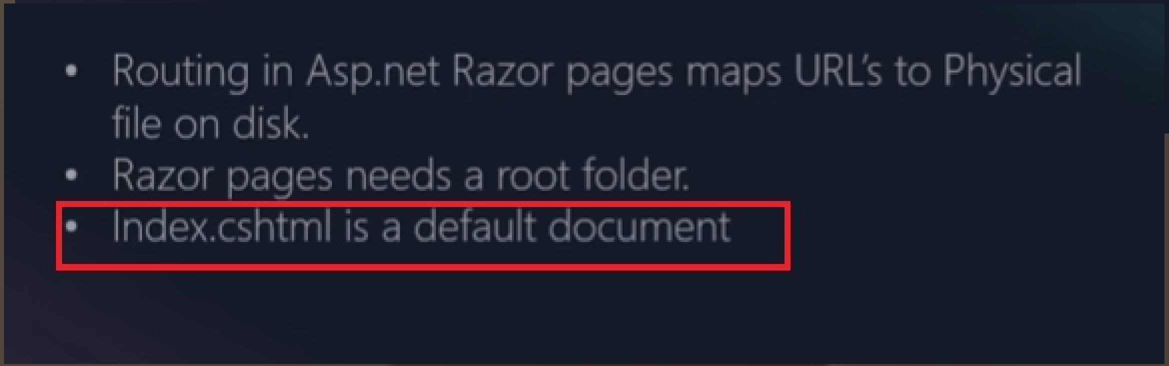
So the Index.csHtml will be View or a **razor page**
Index.cshtml.cs will be a **model**

Routing in Razor pages



Routing in Razor Pages

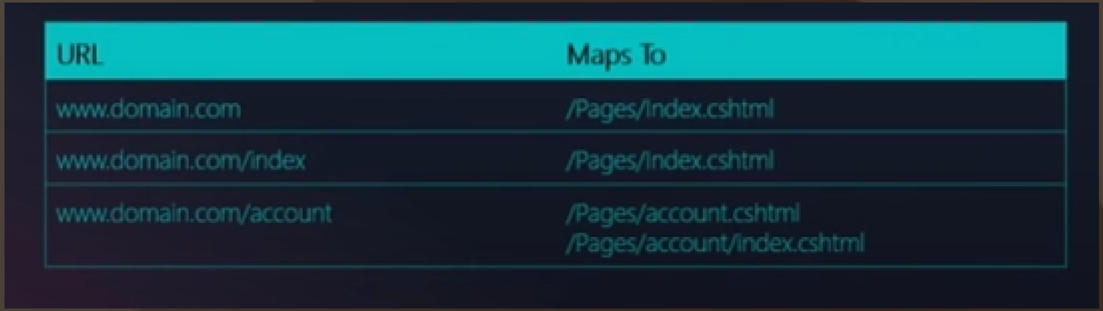
- Routing in Asp.net Razor pages maps URL's to Physical file on disk.
- Razor pages needs a root folder.



- Routing in Asp.net Razor pages maps URL's to Physical file on disk.
- Razor pages needs a root folder.

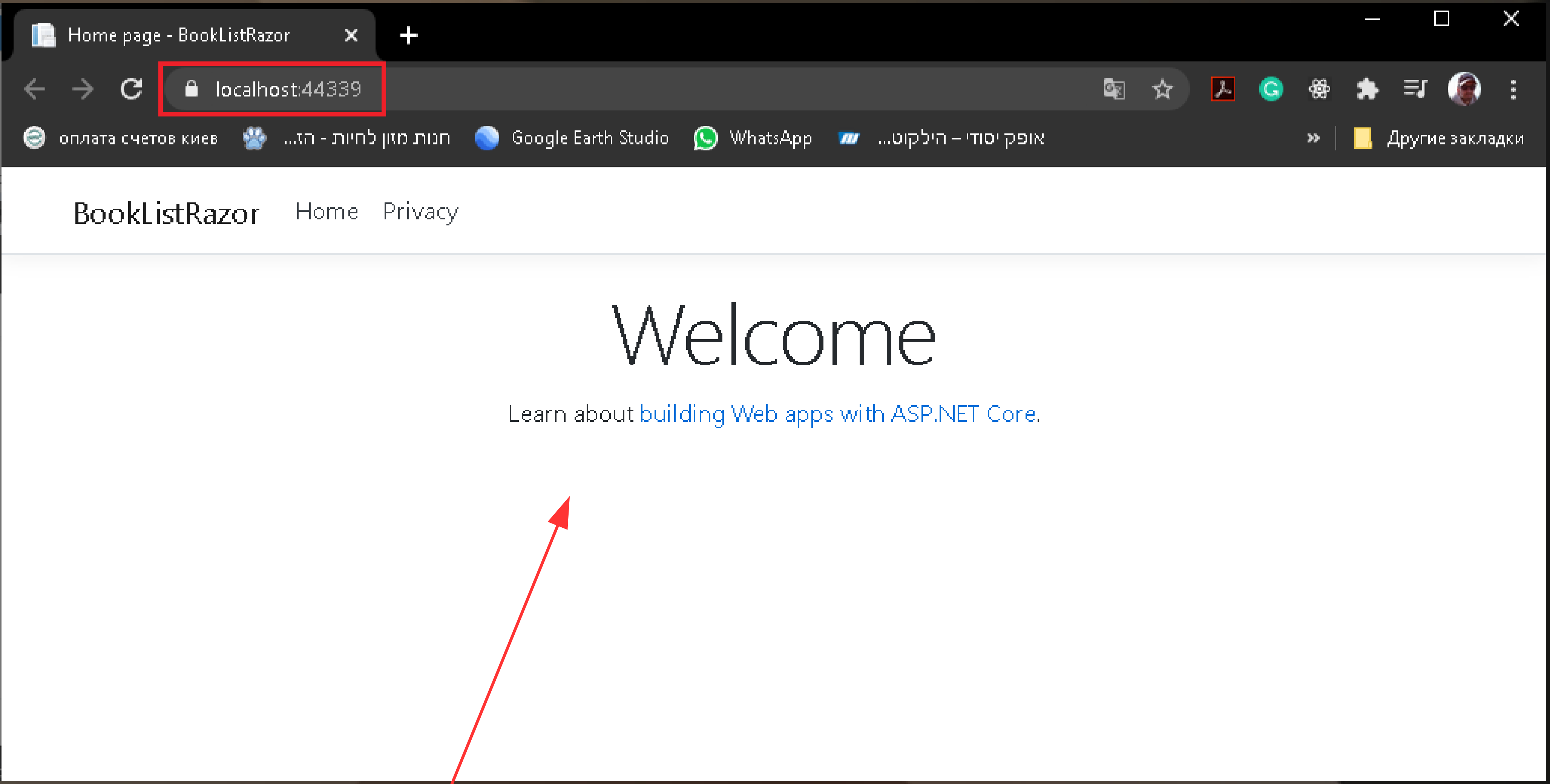
- Index.cshtml is a default document

Example:



URL	Maps To
www.domain.com	/Pages/Index.cshtml
www.domain.com/index	/Pages/Index.cshtml
www.domain.com/account	/Pages/account.cshtml /Pages/account/index.cshtml

It's time to Run Our application.
1. Go back to the project, and press **F5 to** r.un the project



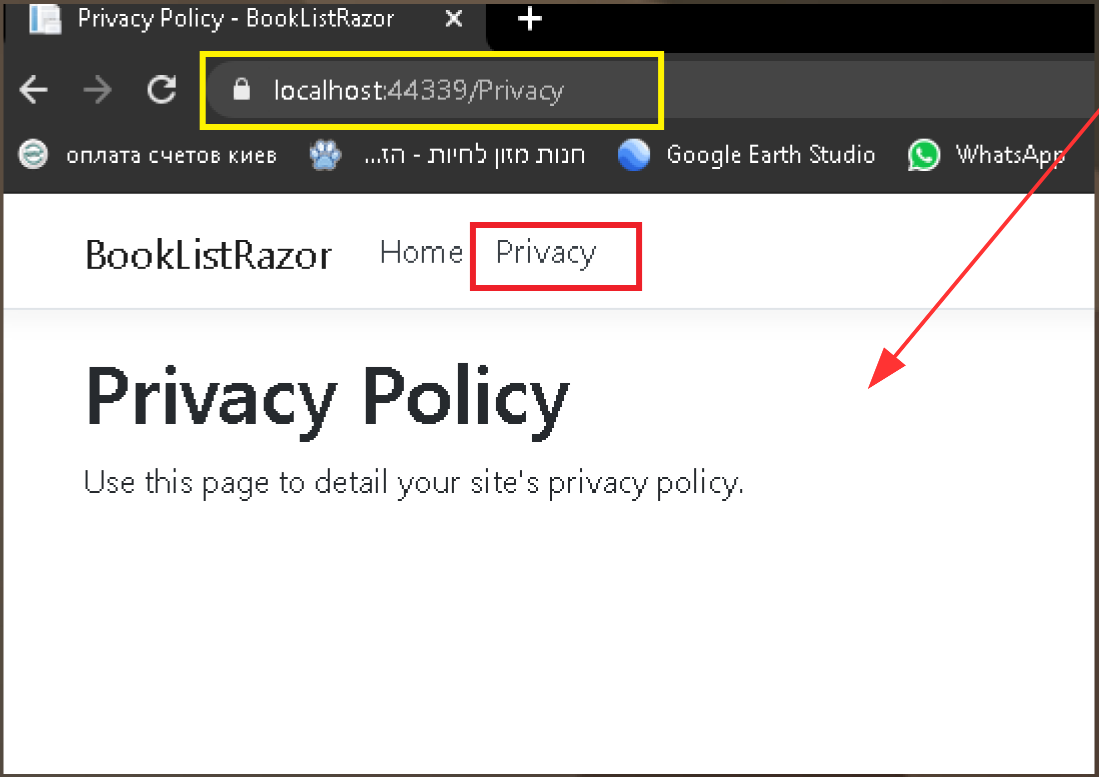
BookListRazor Home Privacy

Welcome

Learn about building Web apps with ASP.NET Core.

By defauly it is loading the **index.cshtml** Page.

By clicking on Privacy link it will open the privacy.cshtml page

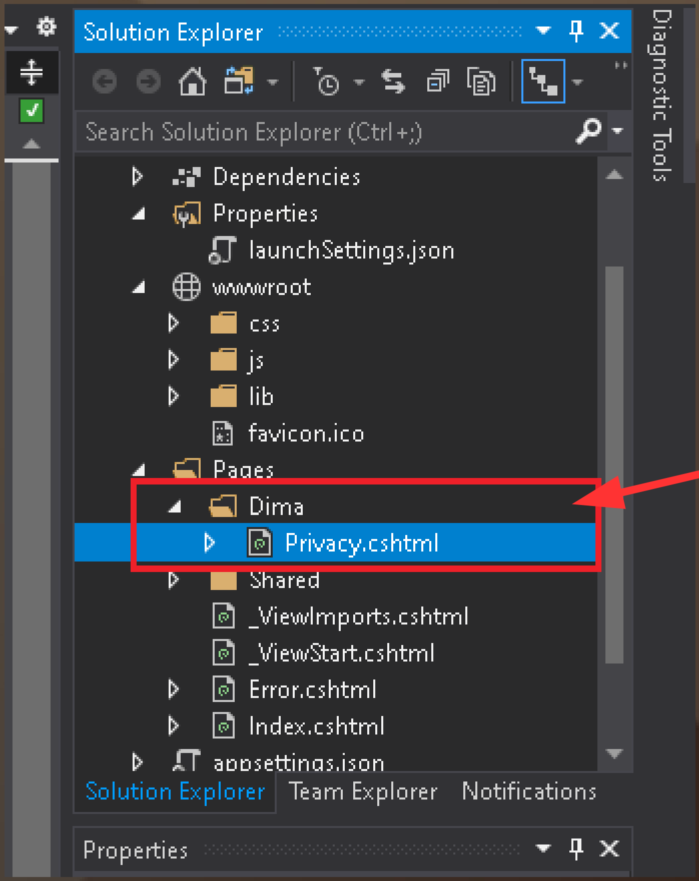


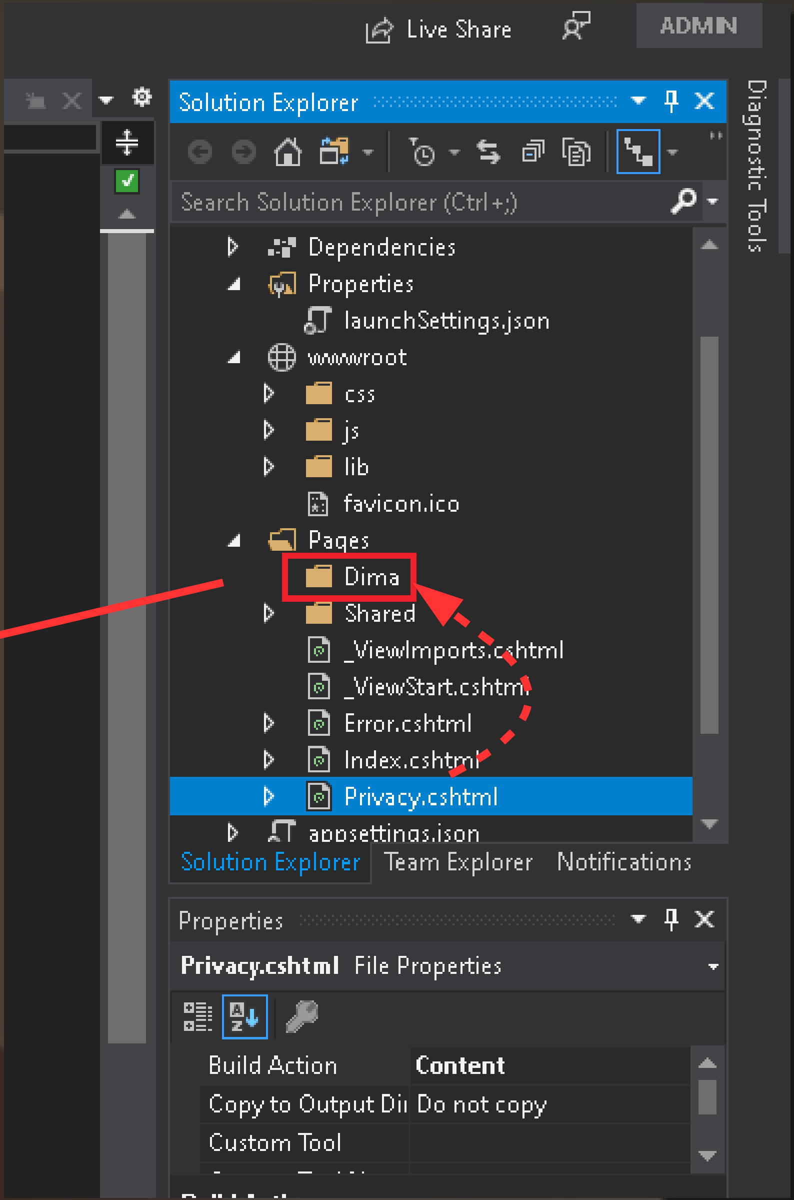
Privacy Policy

Use this page to detail your site's privacy policy.

Stop The application, and add a new folder within **wwwroot/Pages**

1. Add a new folder. Name it your name
2. Move the **privacy.cshtml** file to your new folder.

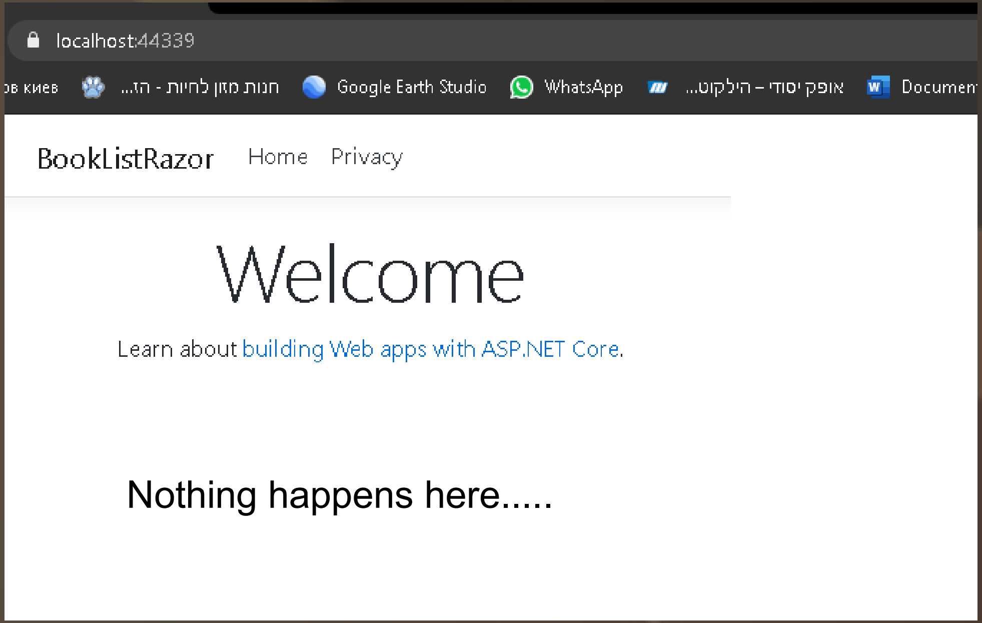




Build Action	Content
Copy to Output Dir	Do not copy
Custom Tool	

3. Run the application (F5)
4. Try to access the privacy.cshtml again

When you click the Privacy link you will be redirected to a homepage
Because Privacy.cshtml now located in different place.

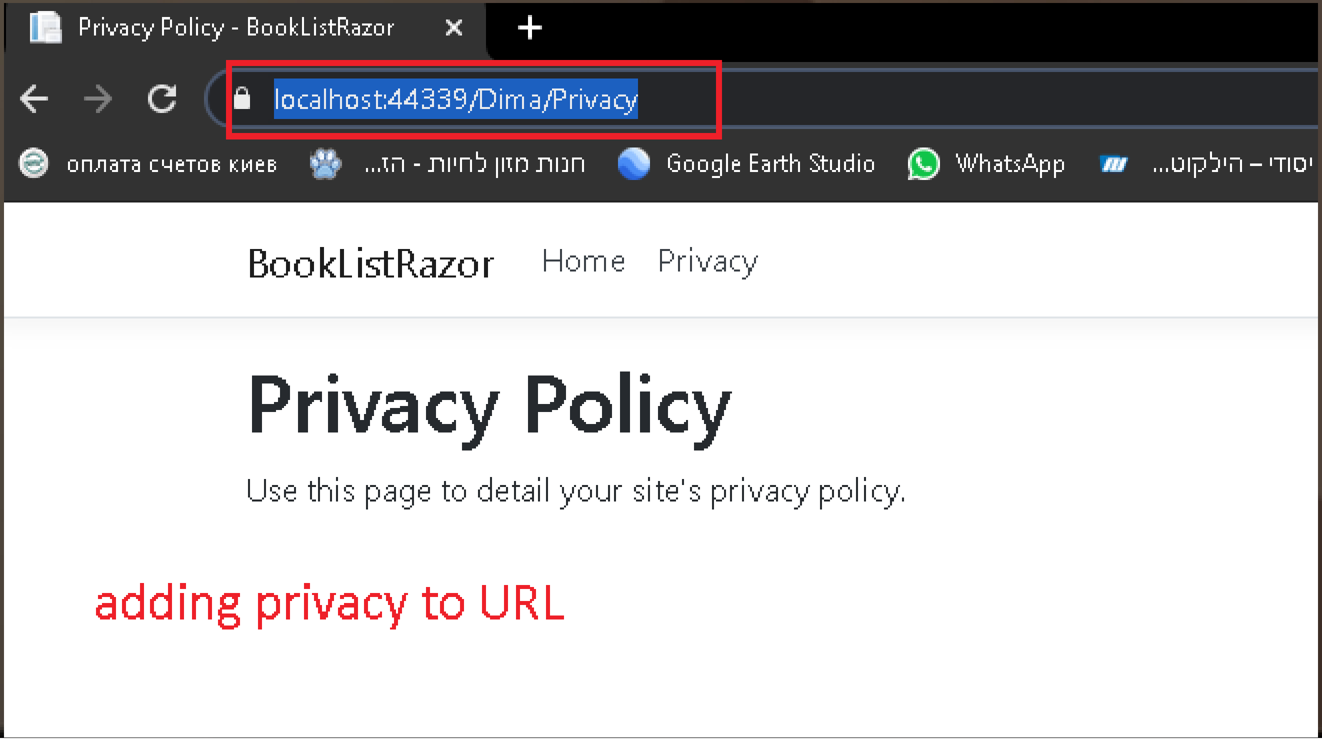


Welcome

Learn about building Web apps with ASP.NET Core.

Nothing happens here.....

In order to access the privacy.cshtml you have to provide a folder name in the URL



Privacy Policy

Use this page to detail your site's privacy policy.

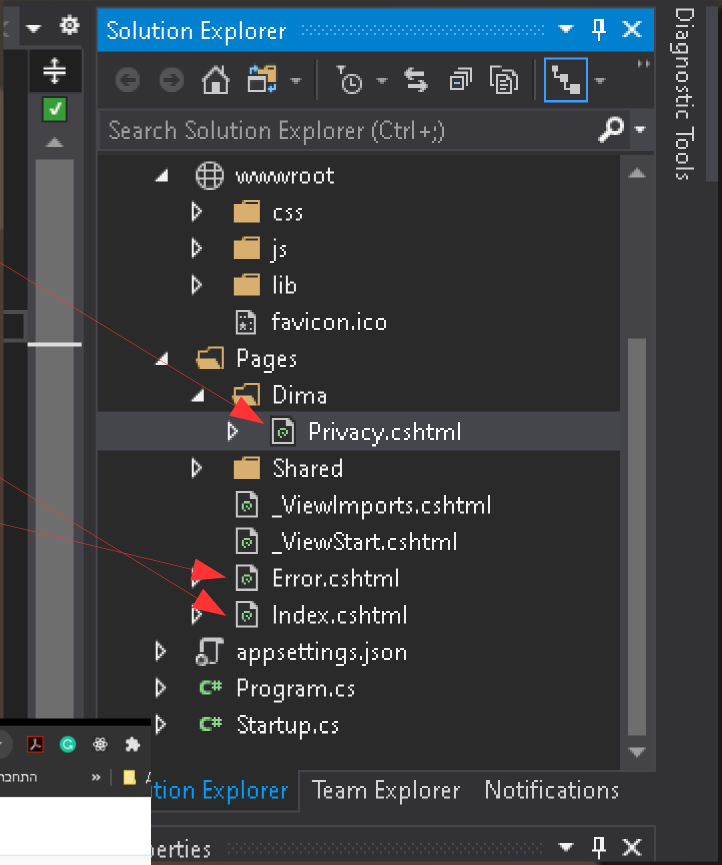
adding privacy to URL

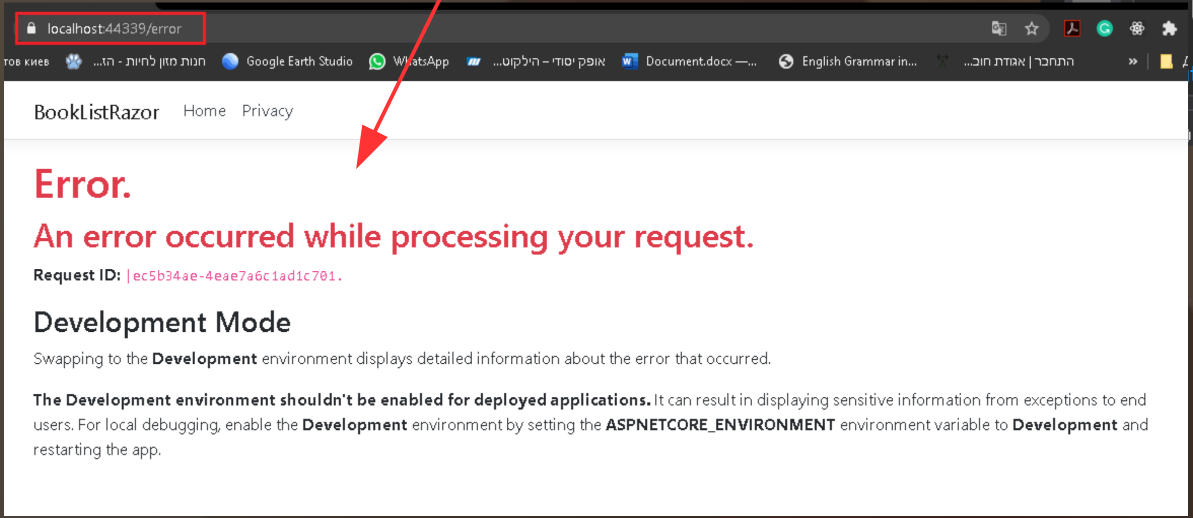
So as you see the linking is exactly as you see them in Pages Folder

Localhost https://localhost:44339/
Localhost Localhost https://localhost:44339/Dima/Privacy

Localhost https://localhost:44339/Error

Error.cshtml page




localhost:44339/error

BookListRazor Home Privacy

Error.

An error occurred while processing your request.

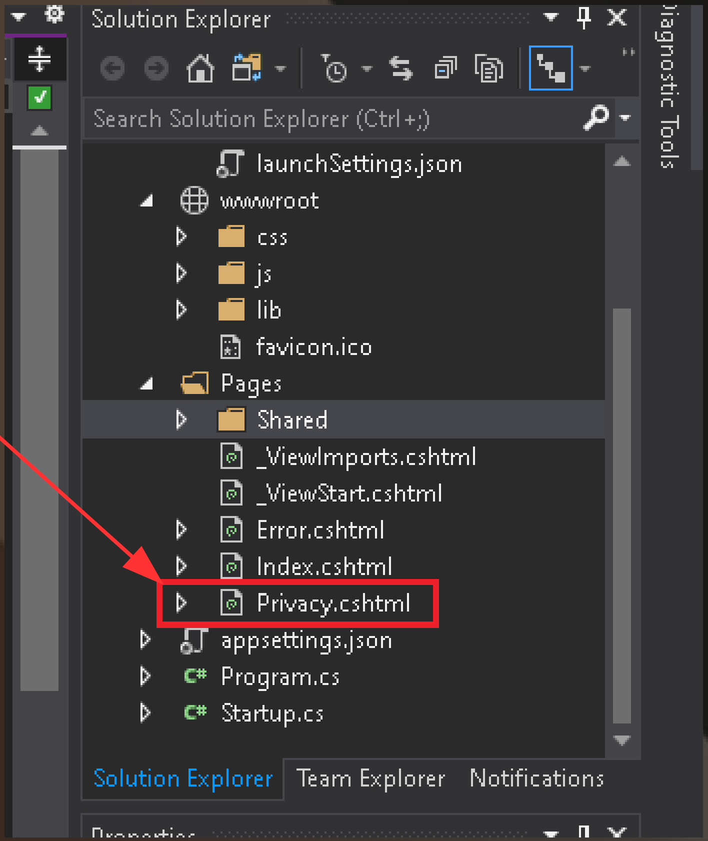
Request ID: |ec5b34ae-4eae7a6c1ad1c701.

Development Mode

Swapping to the **Development** environment displays detailed information about the error that occurred.

The Development environment shouldn't be enabled for deployed applications. It can result in displaying sensitive information from exceptions to end users. For local debugging, enable the **Development** environment by setting the **ASPNETCORE_ENVIRONMENT** environment variable to **Development** and restarting the app.

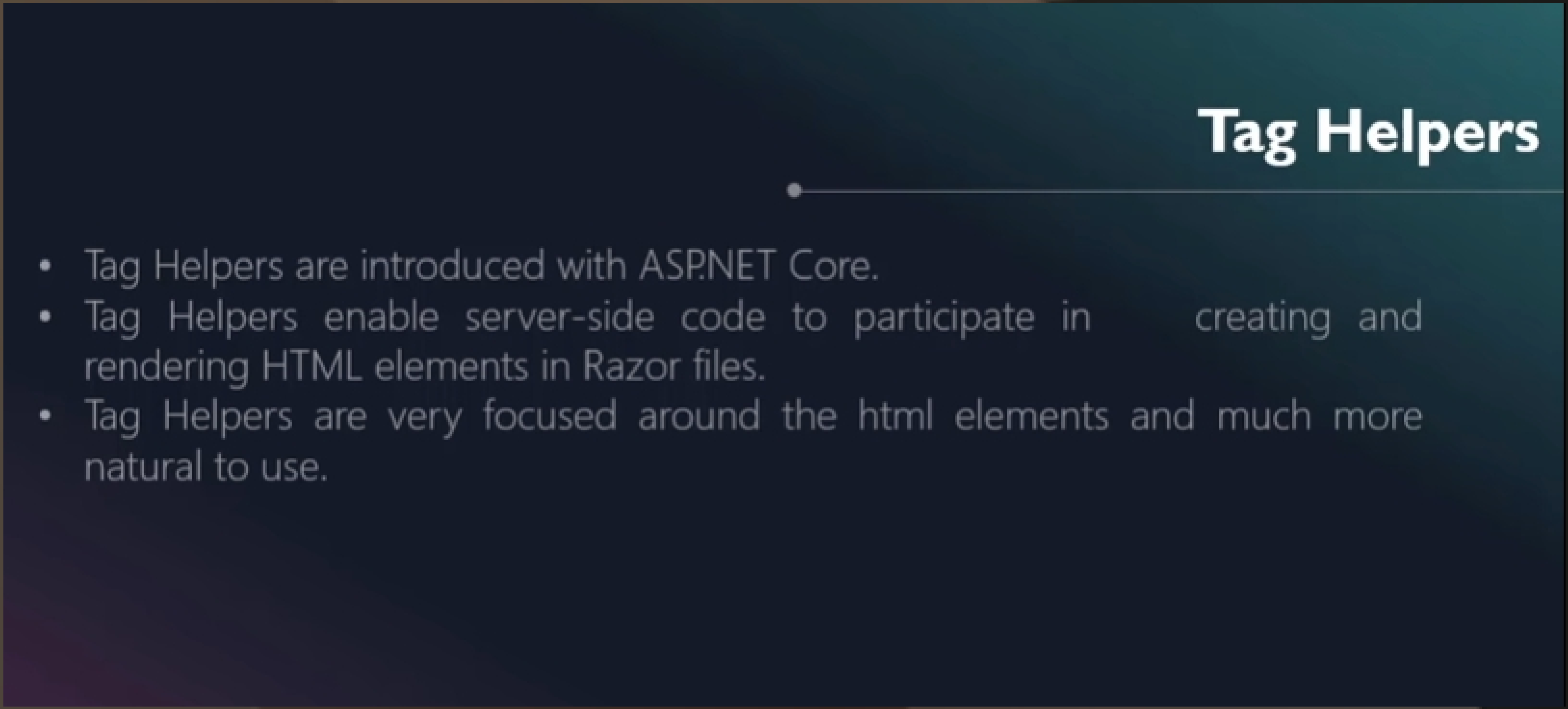
Finally Move the Privacy.cshtml inside Pages folder, and delete your Previously cretaed folder



This was a brief overview of How routing works

Tag Helpers

Tag helper are brand new to ASP. NET Core



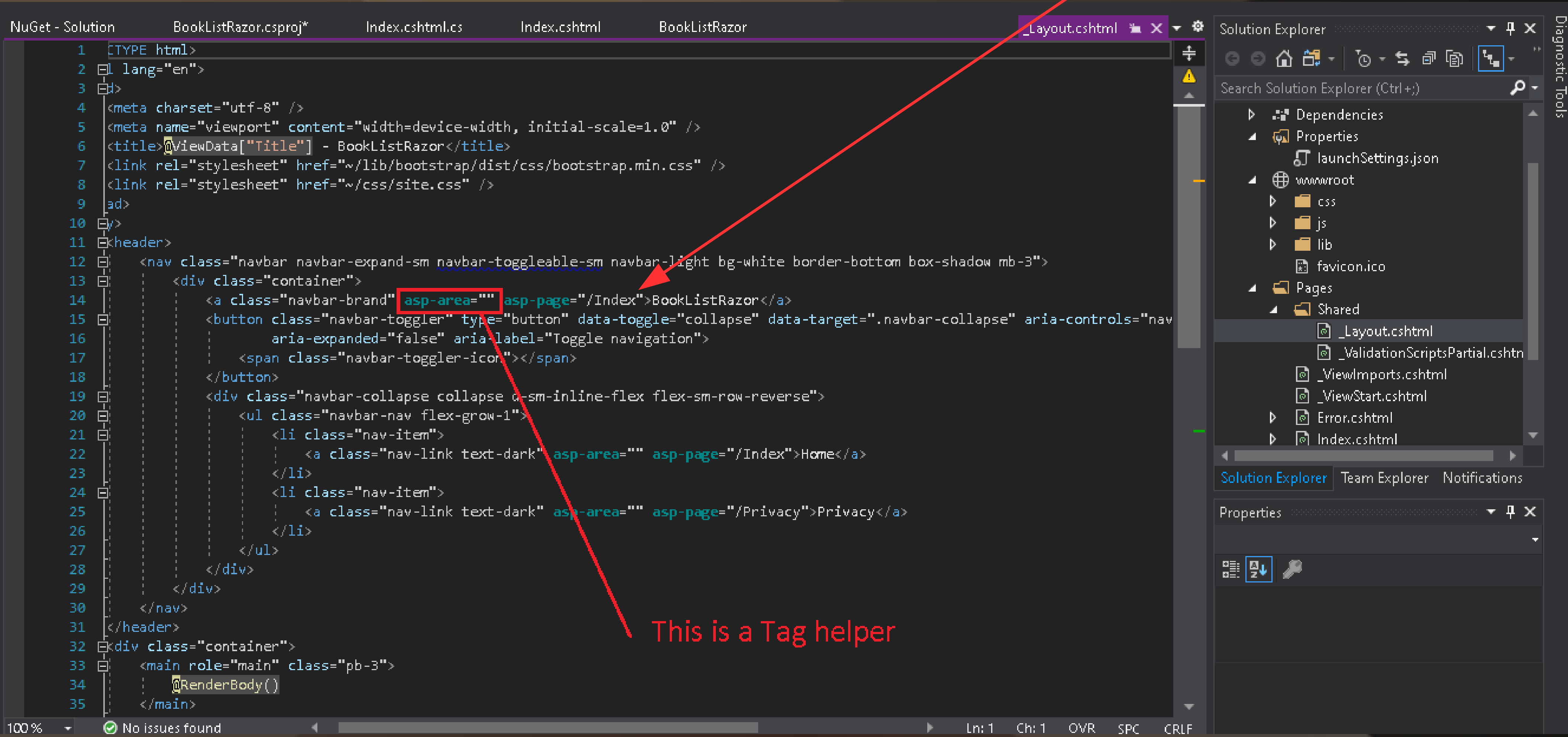
Tag Helpers

- Tag Helpers are introduced with ASP.NET Core.
- Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files.
- Tag Helpers are very focused around the html elements and much more natural to use.

Go back to your project and open index.cshtml file

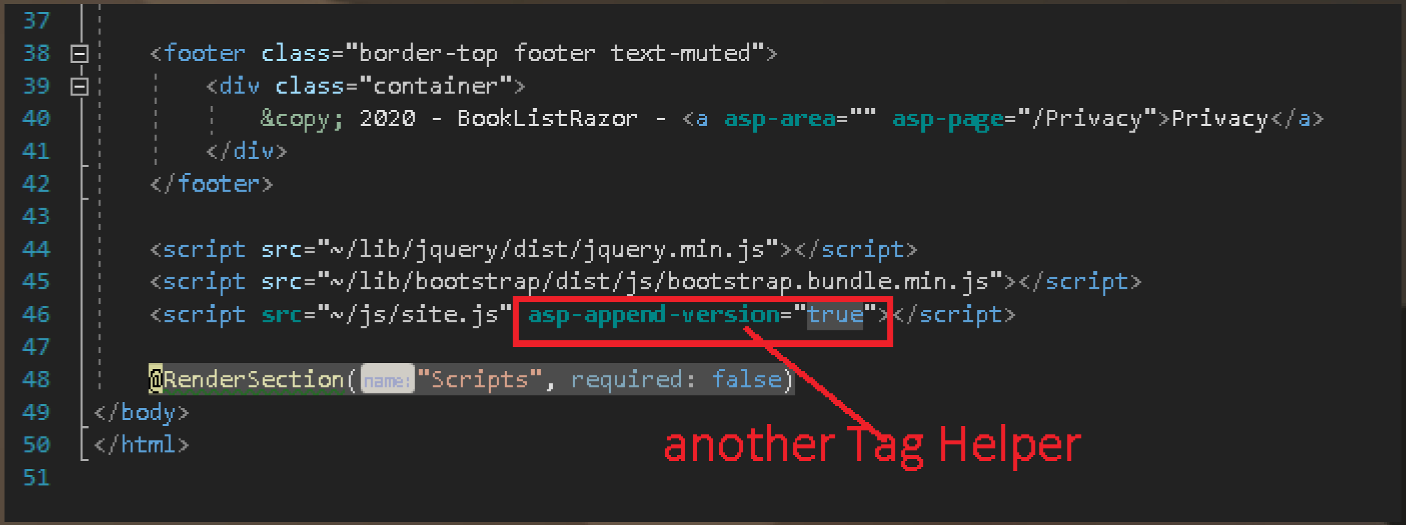
Up until now we do not have a Tag helpers yet associated with Index.cshtml
But if you go to _layout.cshtml there should be a plenty of them.

When we have to **redirect** to any of the **razor pages**, we will helper tag **asp-page**=“/Index“>BookListRazor</a>



```
CTYPE html>
l lang="en">
d>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>@ViewData["Title"] - BookListRazor</title>
<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
<link rel="stylesheet" href="~/css/site.css" />
ad>
y>
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-page="/Index">BookListRazor</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="nav
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
```

Another tag-helper at the bottom of the index.cshtml page



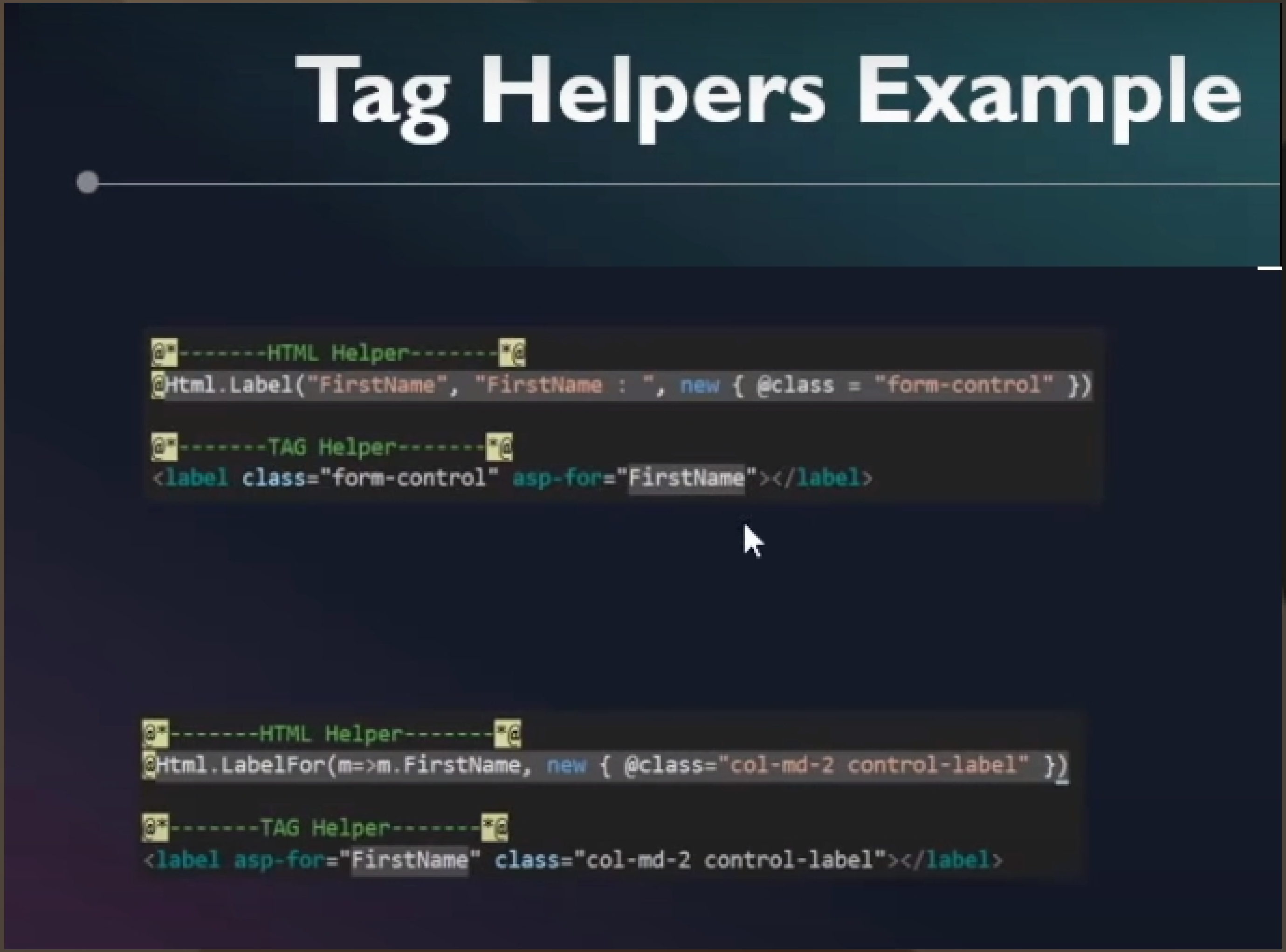
```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2020 - BookListRazor - <a asp-area="" asp-page="/Privacy">Privacy</a>
        </div>
    </footer>

    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>

    @RenderSection(name: "Scripts", required: false)
</body>
</html>
```

Later on we will use Tag helpers in our course.
You can use a regular html, and append a Tag helpers just like you saw and the above code
We will use Tag helpers for different controls later on.

The concept behind a Tag helpers is: You can use a regular tags+ adding ASP Tags.

See how a Tag Helpers works

See the similarity between Html Helpers
And TAG helprs



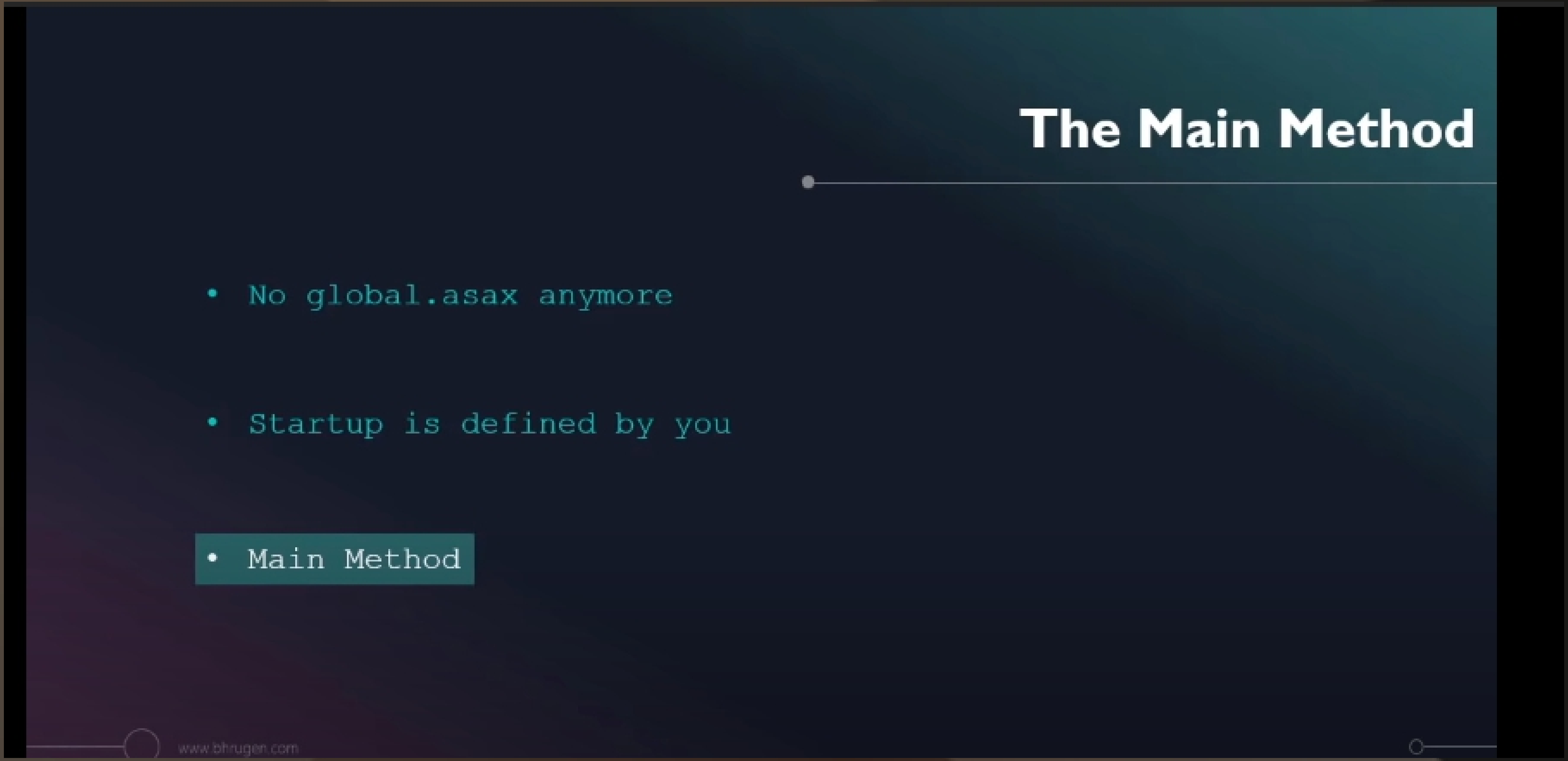
Tag Helpers Example

```
@*-------HTML Helper-------*@
@Html.Label("FirstName", "FirstName : ", new { @class = "form-control" })

@*-------TAG Helper-------*@
<label class="form-control" asp-for="FirstName"></label>
```

```
@*-------HTML Helper-------*@
@Html.LabelFor(m=>m.FirstName, new { @class="col-md-2 control-label" })

@*-------TAG Helper-------*@
<label asp-for="FirstName" class="col-md-2 control-label"></label>
```

Both of these tags performs same functionality, but Html Tags are less readable
All you have to du is to use **asp-for** tag helper

Back in the day a classic ASP.net were using **global.asax** file to contain all the Custom logic.
Novadays the steps needed to start the application are now determined by you.
That starts from a Program class file – **Program**.cs
Program.cs contain a Main() method which is the entry point of the application.
When Run time Excecutes the appl,ication it looks for this main() method.
Most DotNet applications startup using this method(main())



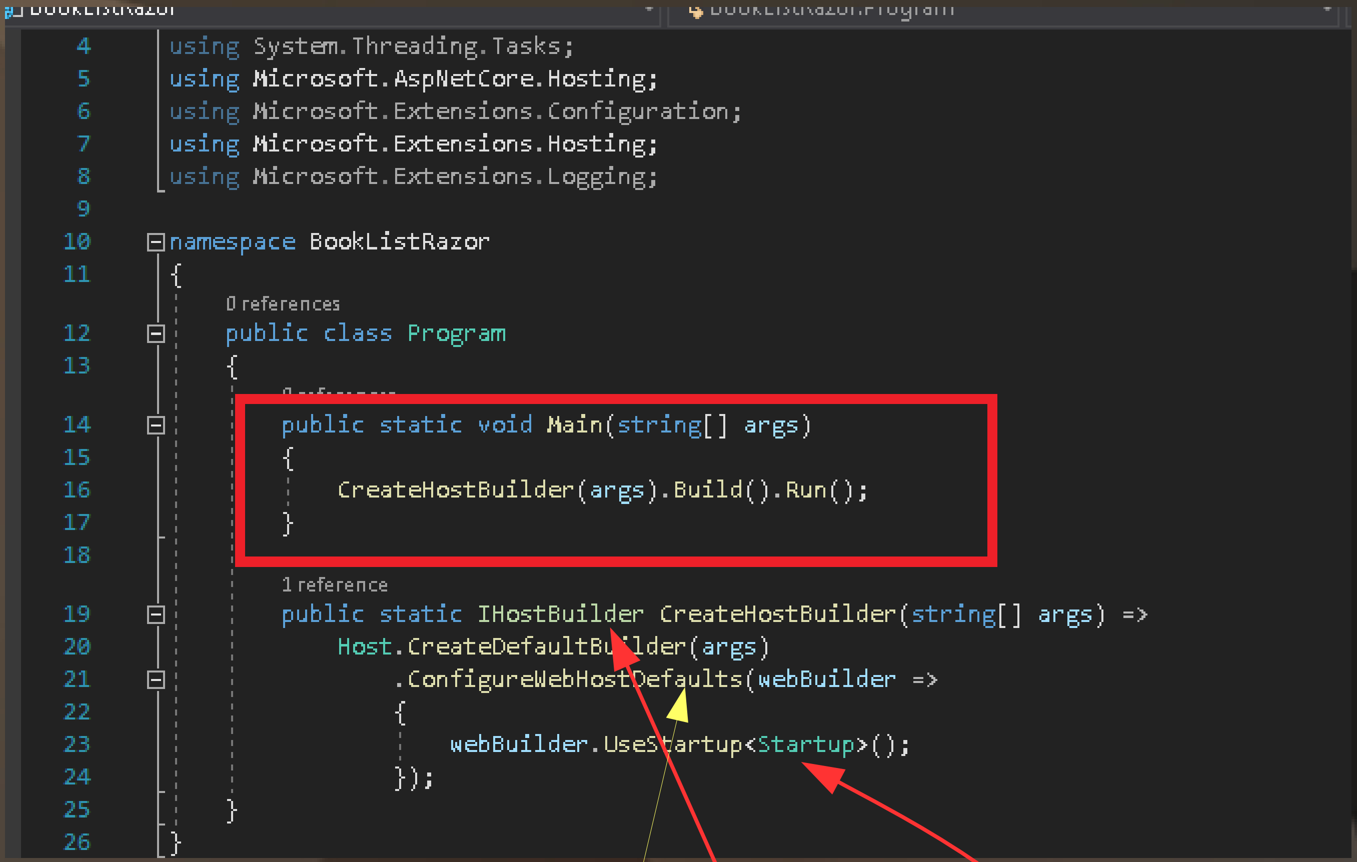
The Main Method

- No global.asax anymore
- Startup is defined by you
- Main Method

www.bhrugen.com

Let's open our application and examine **start.sc** file.

This is the Main() Method.

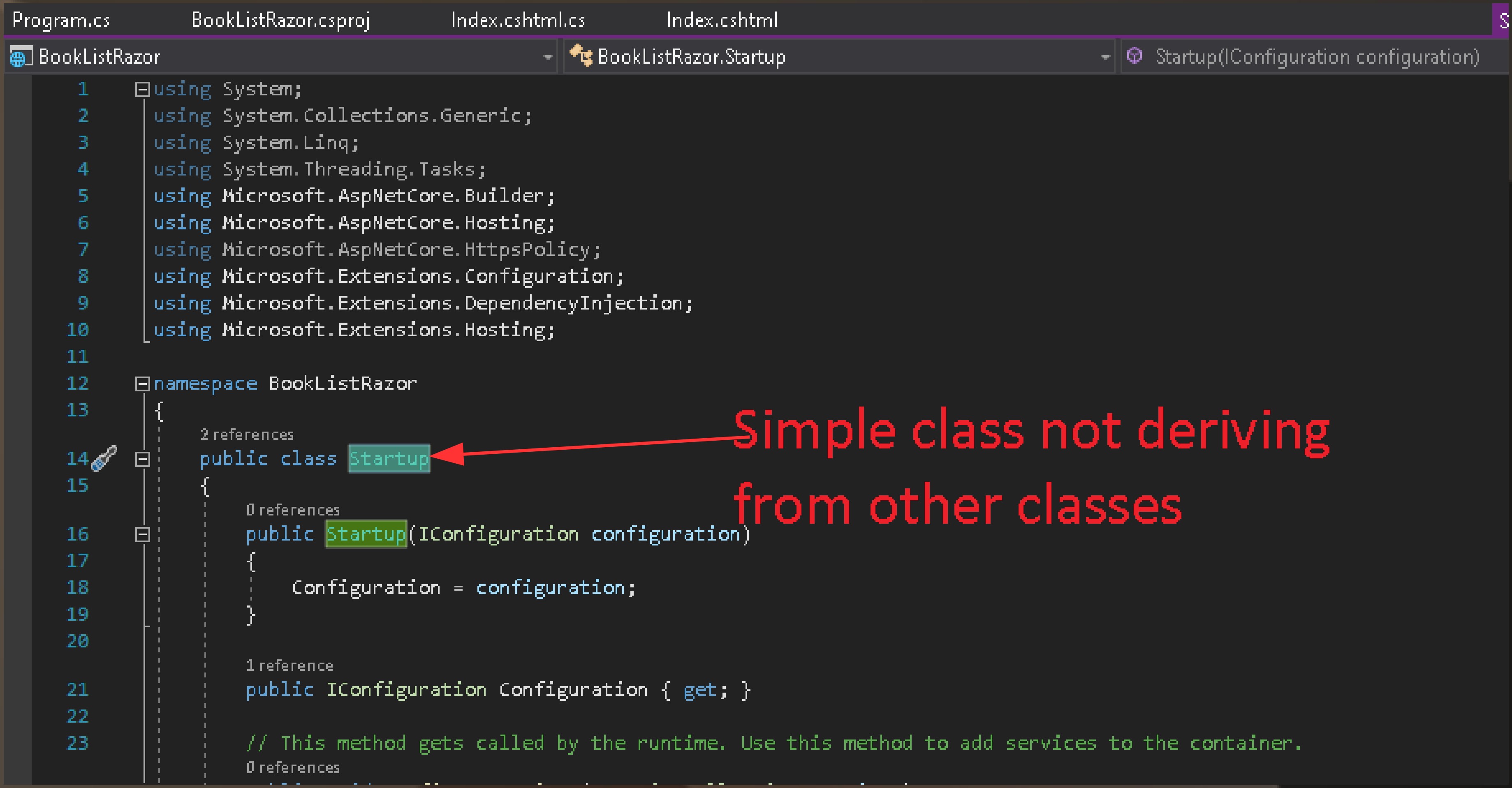


```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace BookListRazor
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Configuration is build by calling **CreateHostBuilder** Method
Which is of type **IHostBuilder, and it returns IhostBuilder**
Then a **Build**. And **Run** Methods are called.

That configures the web Host using **defaults**
It deals with configurations how the Asp.Net Application deals with a web server configuration,
Files, routing, and so on.

A webBuilder is also configured to use a **startup** class file. You can open the startup class file
By pressing F12, or open it from a solution explorer.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace BookListRazor
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        1 reference
        public IConfiguration Configuration { get; }

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

The runtime will cal 2 methods in this Program class

1. ConfigureServices

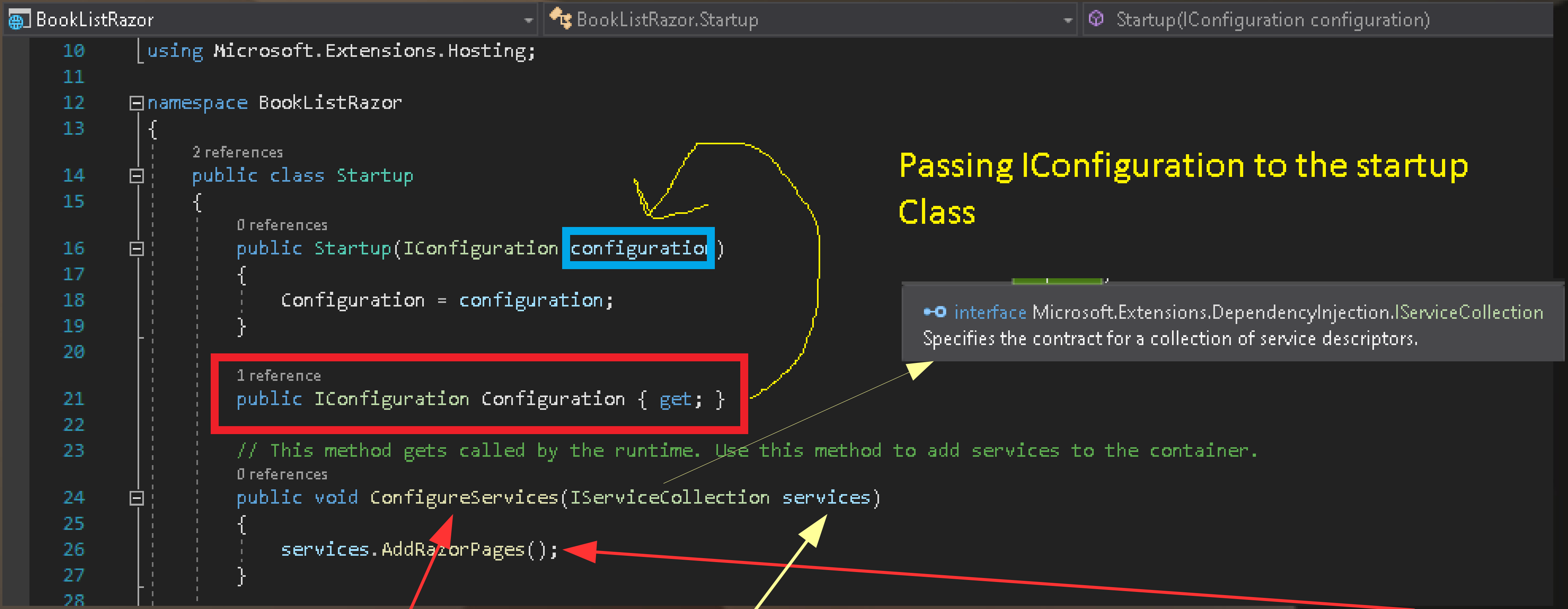
```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
}
```

2. and Configure method

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
...
```

The runtime executes Main() method, which among other thing configures the **startup** class

The runtime will call methods, configure services, and configure the „whole thing“
Here we have **IConfiguration** Object that is being passed as **DependencyInjection** to the Startup class



BookListRazor | BookListRazor.Startup | Startup(IConfiguration configuration)

```
using Microsoft.Extensions.Hosting;

namespace BookListRazor
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        1 reference
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
        }
```

Passing IConfiguration to the startup Class

interface Microsoft.Extensions.DependencyInjection.IServiceCollection
Specifies the contract for a collection of service descriptors.

This method gets called by the runtime.Use this method to add services to the cont*ainer*
Container means our application

The purpose of **ConfigureServices** method is to configure **Dependency Injection.**
Depoendency injection was optional in a classic ASP.NET.
However it forms an **integral** part of ASP.NET CORE itself.
So **ConfigureServices()** method adds services to the application to make them available.
You get the **service collection object** that injected into the method as **paramet**er
Now you can use this to build-on the services that wil available to this application.
Examples of the services would be: **Entity framework core**, **Identity service**, and many more.

By default you will have AddRazorPages() method avaliable

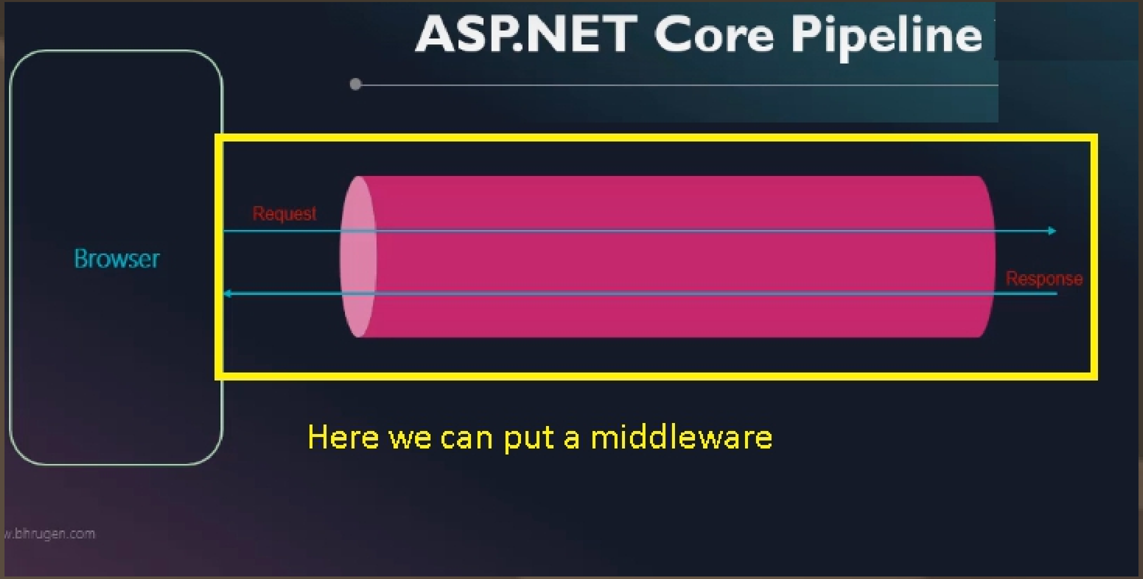
```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
        }
```

Another Deafault method will be:

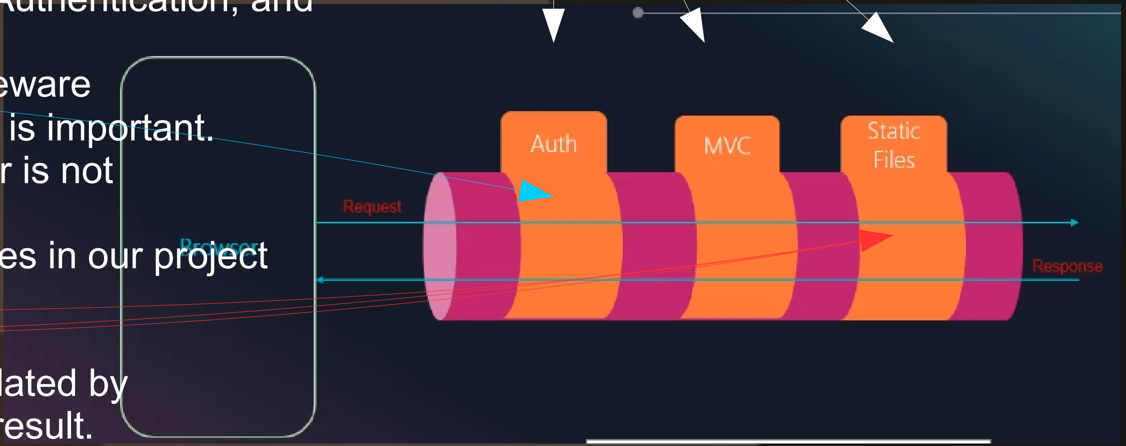
```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            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();
            });
        }
```

This method is used to configure http pipeline.
The pipeline specifies how application showld respond to http request.
Pipline is composed of individual parts which is called **middleware.**

Let's see a presentation to explain this pattern.



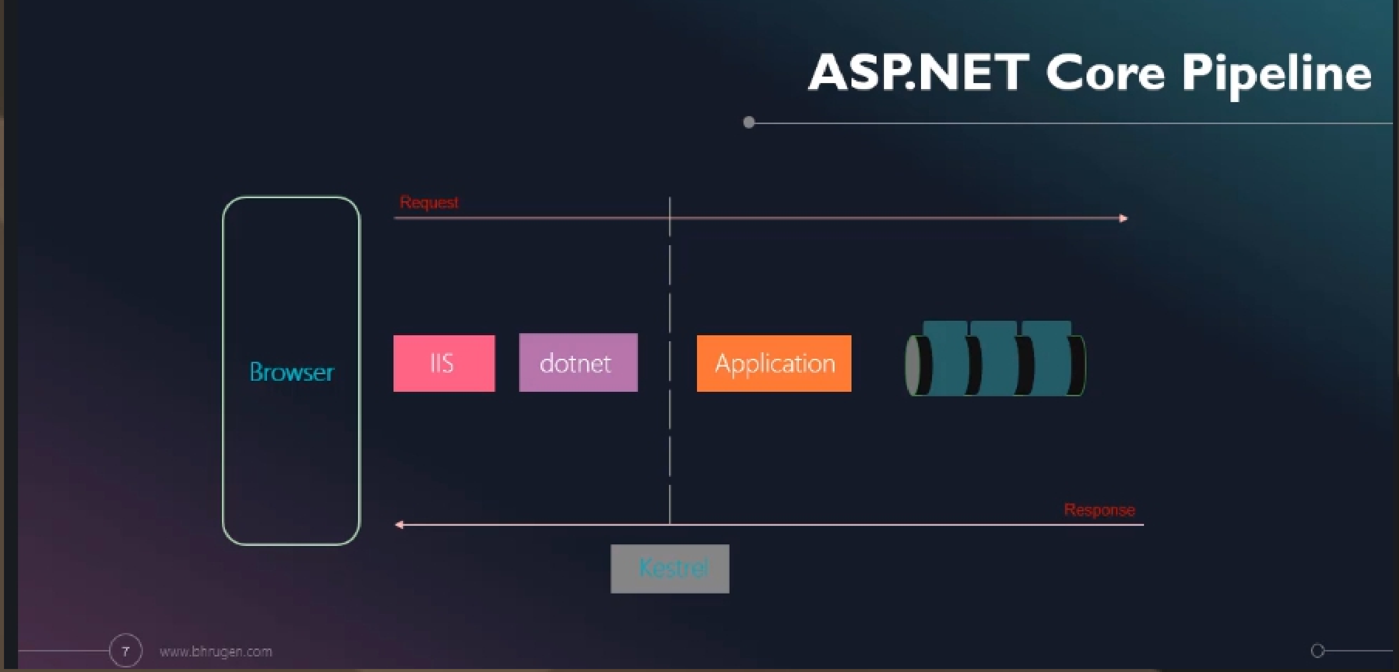
Order is important!!!



The individual parts that makeup a pipeline are called the middlewares.
Let's consider a few of the middlewares that we can add in a pipeline.
One of them can be MVC, and then we can also add Authentication, and
Static files.
You should notice when we add authentication middleware
It showld be done **before** we add MVC, and the order is important.
We do not want to load MVC, and findout that the user is not
Authenticated.
We also have to configure the middleware for static files in our project
Like html files, images, CSS, or java script files.

When data travels through the pipeline it gets manipulated by
Individual middlewares and solves the response or a result.

Proceed to the next slide.



When a request is made by a web browser, it first arrives at web server. Like **IIS**
IIS will invoke the DOT.NET runtime which will load the CLR (Common Language RunTime)
Then look for the entry point in your application. It Will find it in the Main() method in the program class,
And execute it. Which starts internal webserver in your application. We will have **Cashed Route** in our application
The main Method, and the startup class would configure the application, and the request will be routed from IIS to
Cashed route. And then it will be pushed to the application. After that it will be processed by all the middlewares,
And the generated response will be routed back to the cashed route, which will route it back to IIS. That will finally
Produce the responce to the browser. This is more efficient then the old System.Web approach.
The classic system which lies heavily in system.web. Which was tied to IIS. But using a pipeline approach we only
Plug in the middlewares we need. Every middleware we blugged in lies in its own assembly in nuget package.
Since system.Web was tied to IIS, and IIS is Tied to Windows. For that reason you cannot run classic asp.net on other
Webservers then IIS, and windows. Since that no longer the case, Asp.Net Core applications can run on webservers,
and operating systems. One thing you should keep in mind, is that there are two webservers: 1 external server like IIS,
Apache, or linux. And there is also an enternal web server hosted by your application. Request from the external web serve
are passed to the internal web server, and other way around. You can choose different internal web server. But most
common is **Kestrel** since it has first class support in ASP.NET CORE. Kestrel is a lightweight web server which can only
execute a request, **because of which** you need external web server to configure different options like Security, chashing,
and so on. This was a brief overview of how the pipeline comes to the picture.

Let's switch back to our application.

As you see in the Configure method we have a plenty of a middleware objects
They appears as **app.** and then a middleware name

Startup.cs

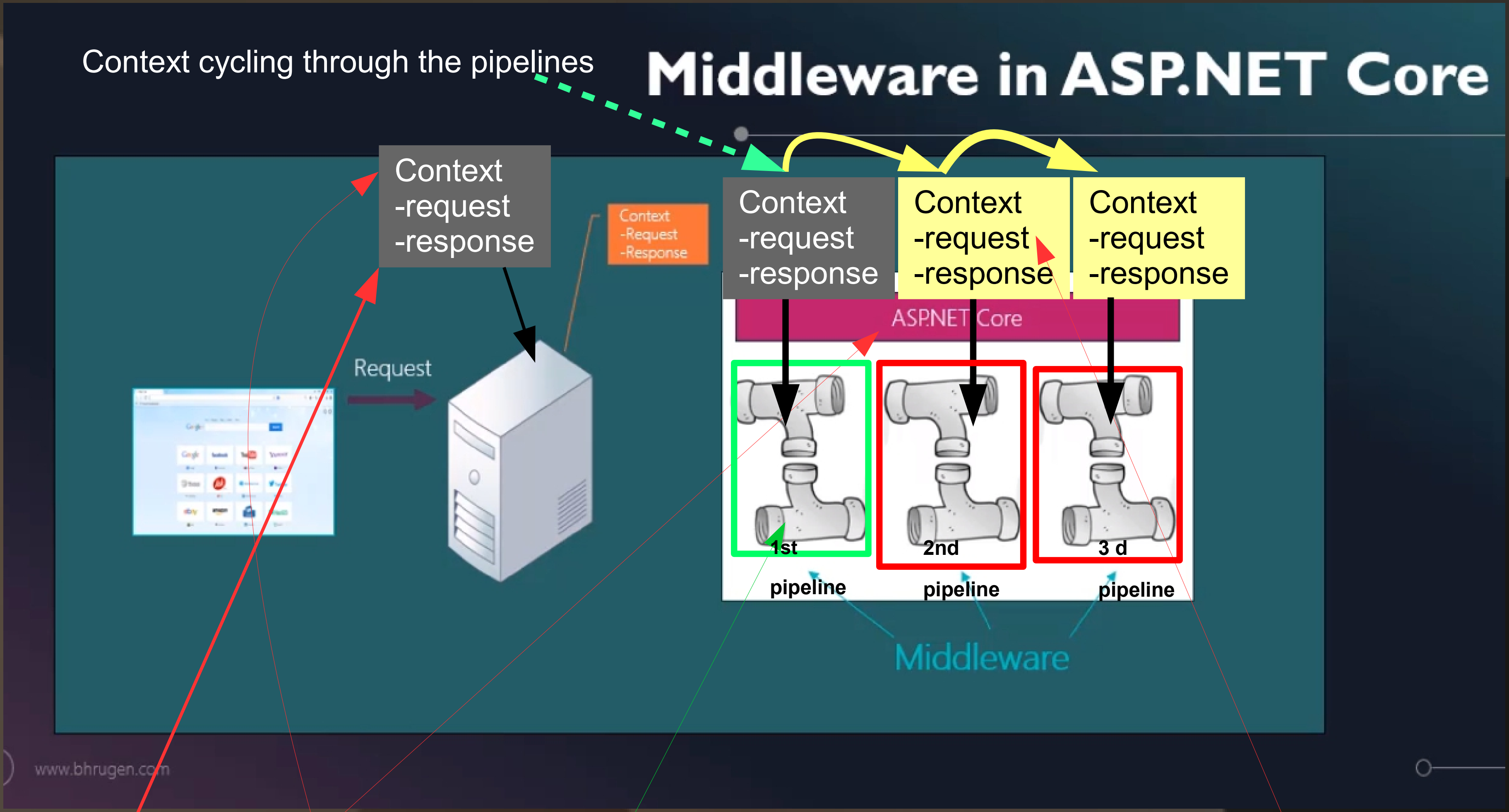
Adding Middlewares to a pipline

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            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();
            });
        }
```

Here we want to use DeveloperException Page.
Else, we want to use a simple **generic Error page .** Then we have httpsRedirection() middleware
Then We have a middleware for **static files** (images, css,javascript) because if this static middleware we able to use
Our static files in the soulution explorer (wwwroot folder).
Then we have **UseRouting**, then **UseAuthorization**.
Finally we have **UseEndPoint.** With a DotNet Core 3 they had intoduced **end point routing.** Here you can configure
 multiple routes. We can add different endpoint here, for different technologies. We will see how to with endpoints
In upcoming examples.
For now it is important to understand how we are plug in different middleware.

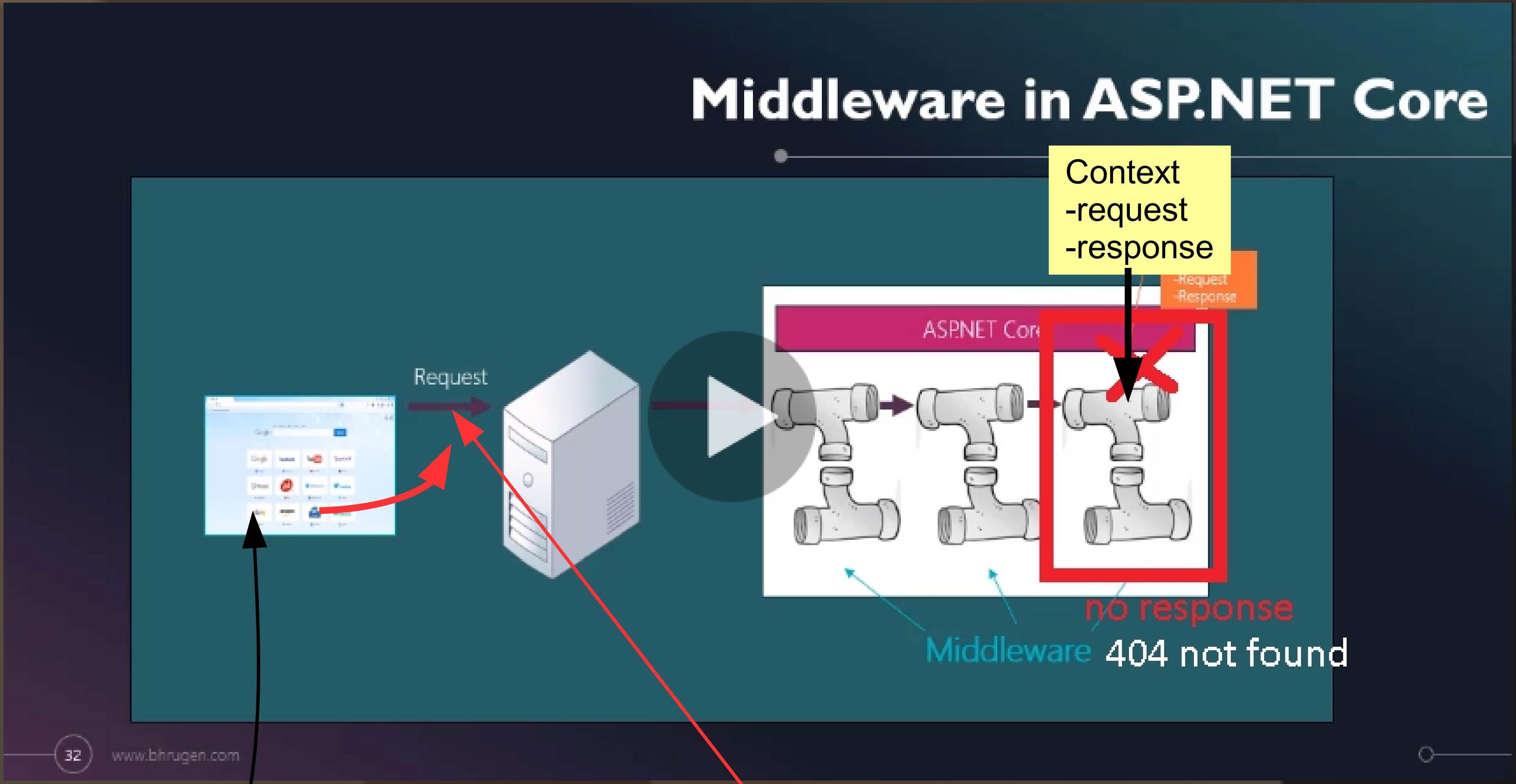
Next step is to understand what is the middle ware are?

Please proceed to the next slide:

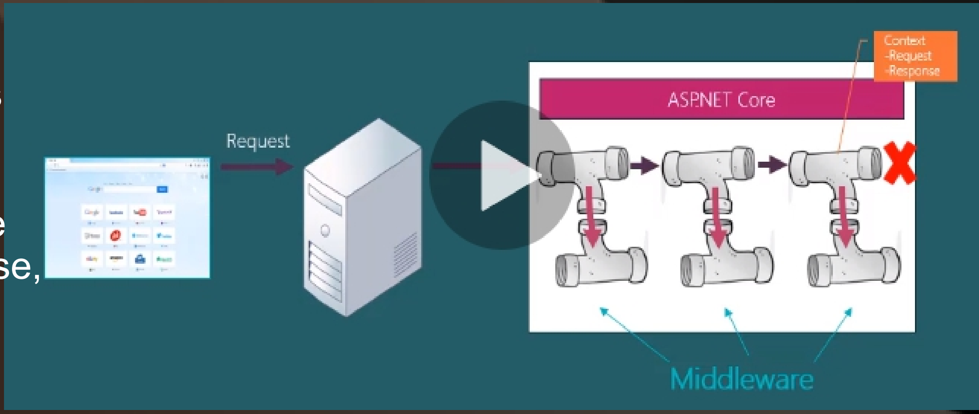

Context cycling through the pipelines
Middleware in ASP.NET Core
Context -request -response
Context -Request -Response
Context -request -response
Context -request -response
Context -request -response
Request
ASP.NET Core
1st pipeline
2nd pipeline
3 d pipeline
Middleware
www.bhrugen.com

Let's undersatnd Pipeline and middlewares in much more detail.
Whenever HTTP request comes in, something must handle this request. So it eventually results in an HTTP response.
Those pieces of code that handles the request and results in a response, Make up the request **Pipeline.** What we can do
Is configure this request' pipeline by adding middlewares, wich are **software components,** that are assebled into an application pipeline
To handel request, and response.
So typically a browser will send a request to your server. This **request** will interpreted by the server, and **handled** by some piece of software
At first the request is **attached** to **context object.** As a part o software that manages that software, in our case it will be ASP.NE CORE
Middlewear. You can essentially think of it as a pipeline, which is a series of pipes that is going to determine what is goint to happen to
the **context.**
First the request is passed along the first pipe. The pipe **interpetes** this request, and determines if a response is needed. If Yes, it attaches
It to a **context.**
If there's no immediate response that should be hadled back to the server, then the context i**s passed along to the next pipe line**
It goes on and on until it reaches the last pipe.
It is also possible that in the end of the pipeline no response has being found. That wil cause a 404 not fopund error.

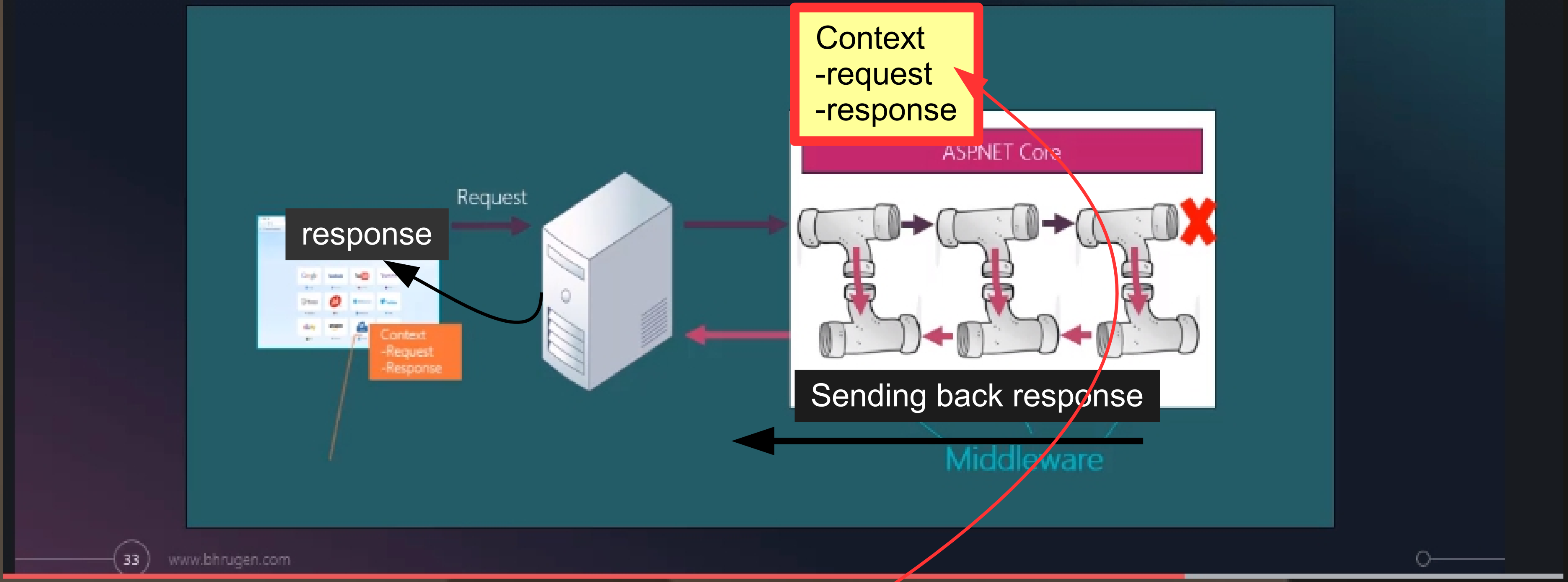
No response has being found at the end of the pipeline



This will **write back** the error message to whoever sent a request
However it is possible that in anyone or more of these **middlewares** there maybe a response that needs to be **passed back**
It could happen in any of the pipes.
So sometimes it could happen that **middleware whould** not pass the context along the next piece, but rather says: OK I have a response that i need to send back!
But typically your **context** will gow all the way through the pipeline
Till the end, where the last piece of a middleware sends a response,
Which gets back through the pipeline, to the server, and server
Then sends the response to the browser.
This is a simplified version of how request works.



Middleware in ASP.NET Core



1.When the request comes to the server, server then acceses Dot.Net Framework.
2.DotNetFramefork Puts your request into **context object**
3. The context passes through all the **Middlewares** in the pipeline.
4. If one of the Middlewares has a response, it will **attach** that response to the **context object**
5. It will **pass back** that **context** object back through the pipeline to the **server**
6. Then a server will send **back** a response to a **web browser.**

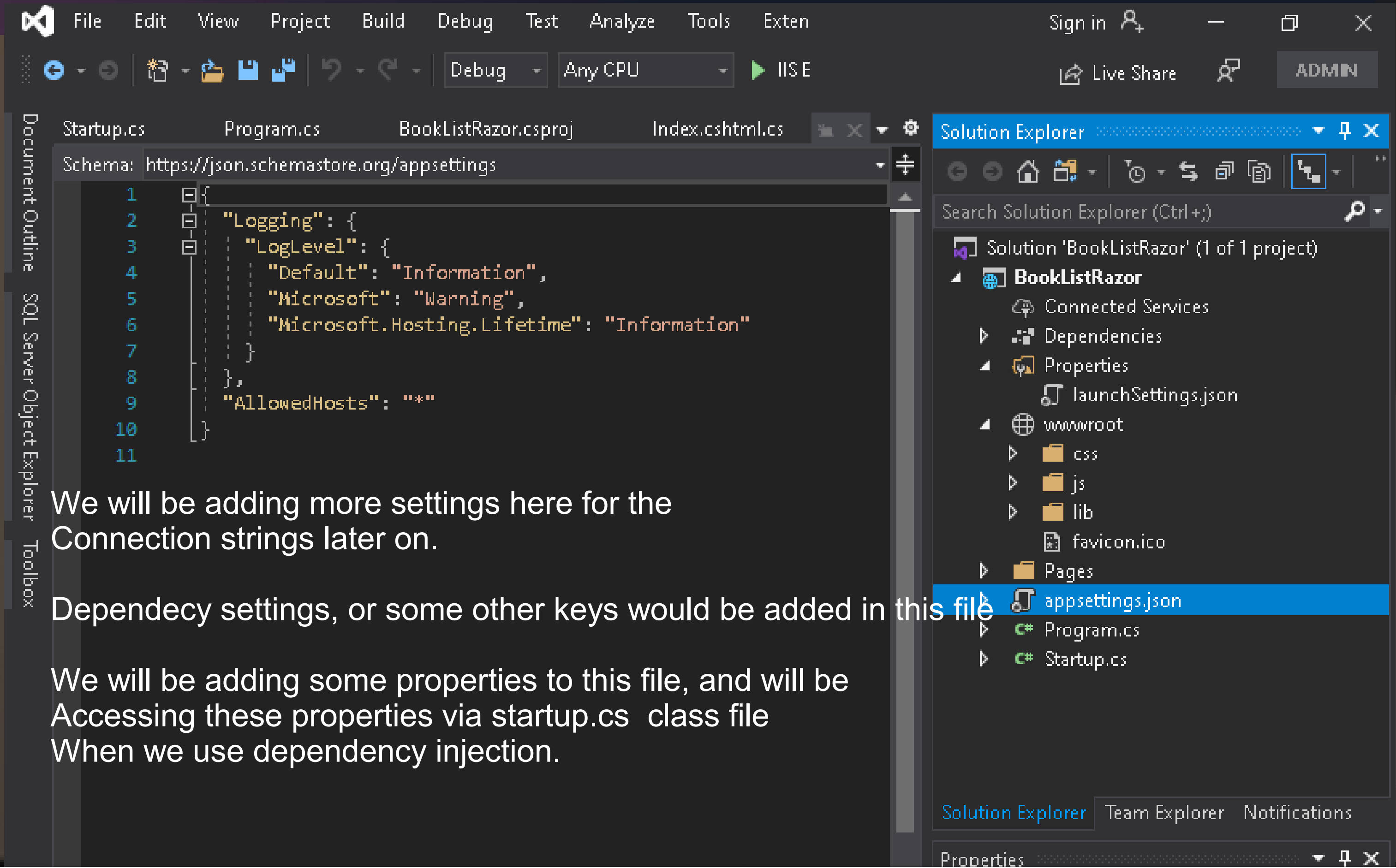
7. Remember: The **order** of the pipeline is **important**!
It alwas get passed from **first-to-last**

An example would be authentication middleware:
If the middleware component finds out that a request is **NOT authorized**, this will immediately send
The non-authorized response back, hence the context object will stop walk through the pipelines. And will stop.
The authentication middleware **is added before** other middlewares in the pipeline.

A Brief view of AppSettings.json file

AppSettings.json

- All of the application's settings are contained in a file named appsettings.json.
- Any changes to the appsettings.json file will require restarting the "Microsoft IIS Administration" service to take effect.



Schema: https://json.schemastore.org/appsettings

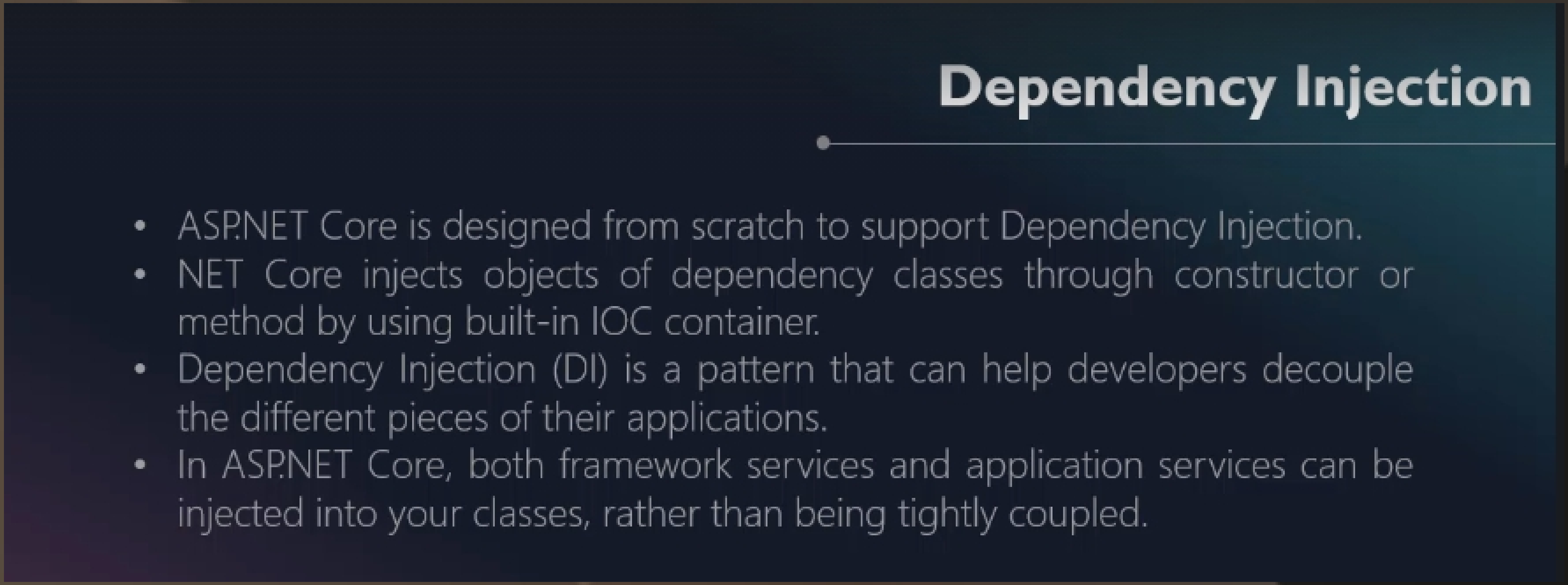
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

We will be adding more settings here for the
Connection strings later on.

Dependecy settings, or some other keys would be added in this file

We will be adding some properties to this file, and will be
Accessing these properties via startup.cs class file
When we use dependency injection.

Dependency injection

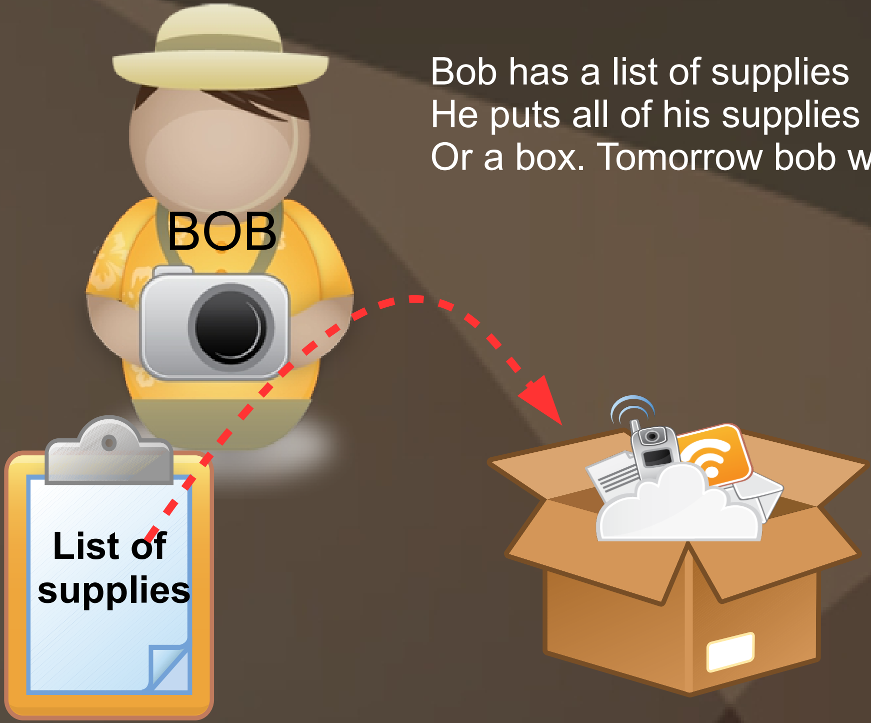


Dependency Injection

- ASP.NET Core is designed from scratch to support Dependency Injection.
- NET Core injects objects of dependency classes through constructor or method by using built-in IOC container.
- Dependency Injection (DI) is a pattern that can help developers decouple the different pieces of their applications.
- In ASP.NET Core, both framework services and application services can be injected into your classes, rather than being tightly coupled.

Dependency injection is a technique for achieving inversion of control between
Classes and their dependencies.
You might be wondering what is IOC, or **Invertion Of Control cointainer.**
IOC Container is a framework for implementing automatic **Dependency injection.**
It manages object creation, its' lifetime, and also injects dependencies to the class.
IOC Container **creates object of a specified class**, and also **injects all of the dependency objects** through
A **constructor** property at runtime, and disposes it at appropriate time. This is done so that we do not have to create
And manage object manually. Support for dependency injection is build into ASP.NET Core.
In ASP.NET CORE both Framework services,and application services can be injected into your classes, rather than being tightly coupled.
Dependency injction is a design pattern in which a class, or object has its' dependent classes **injected** rather than
Creating them **directly**.
Dependency injection can help developers **decouple** different pieces of their application.

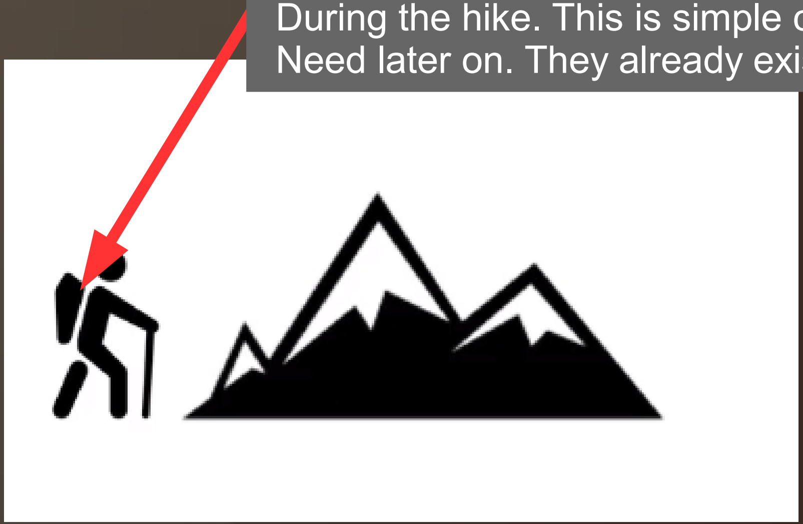
Proceed to the next page for example.



Bob has a list of supplies
He puts all of his supplies in a backpack
Or a box. Tomorrow bob will go hiking in the mountains.

The next day when he goes hiking, he takes the backpack with him

The backpack is a container. So whenever he needs something he takes it oput from the container
During the hike. This is simple concept, when you put some items into the container that you will
Need later on. They already exists inside a container. Just use them whenever you need them.



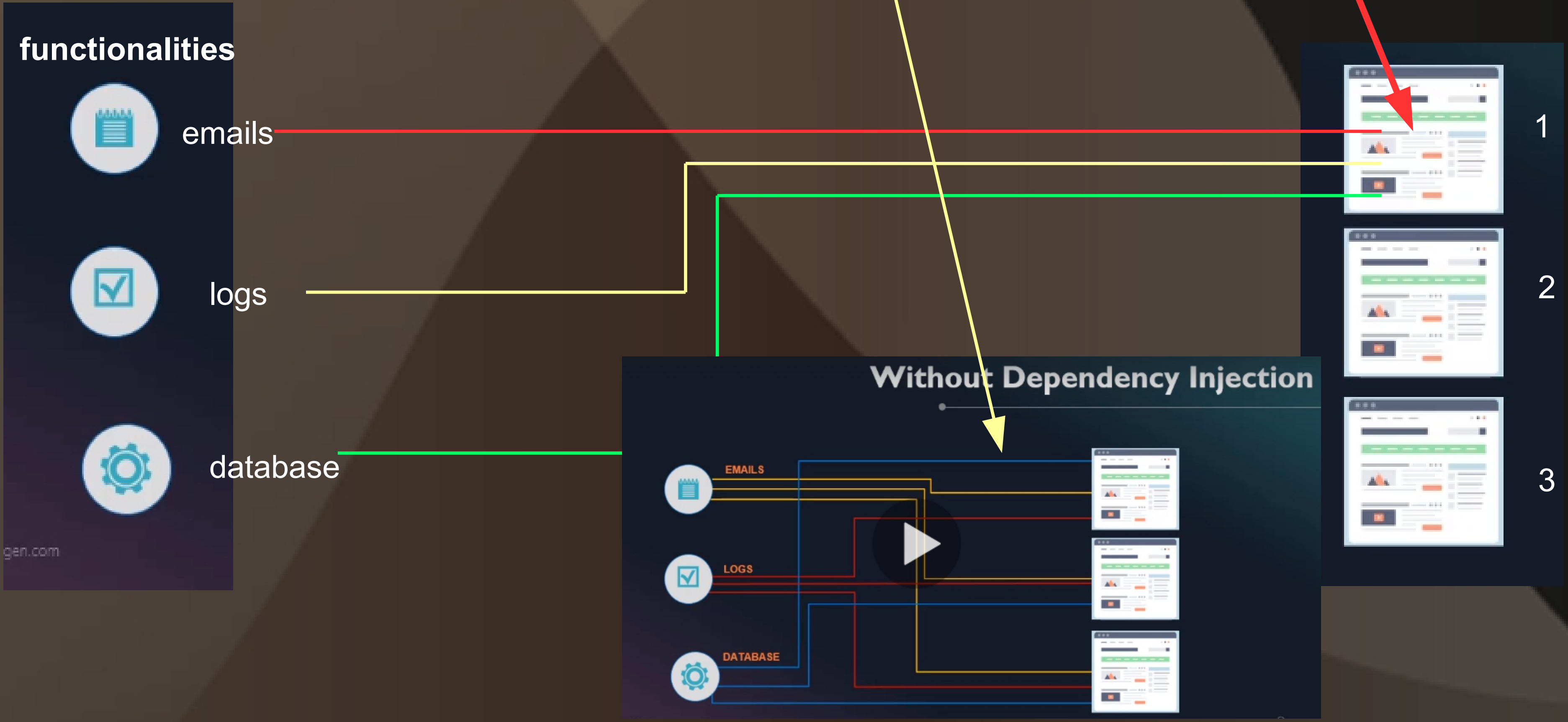
Let's understand this concept in a coding maner

See next slide...

Wityhout dependency injection

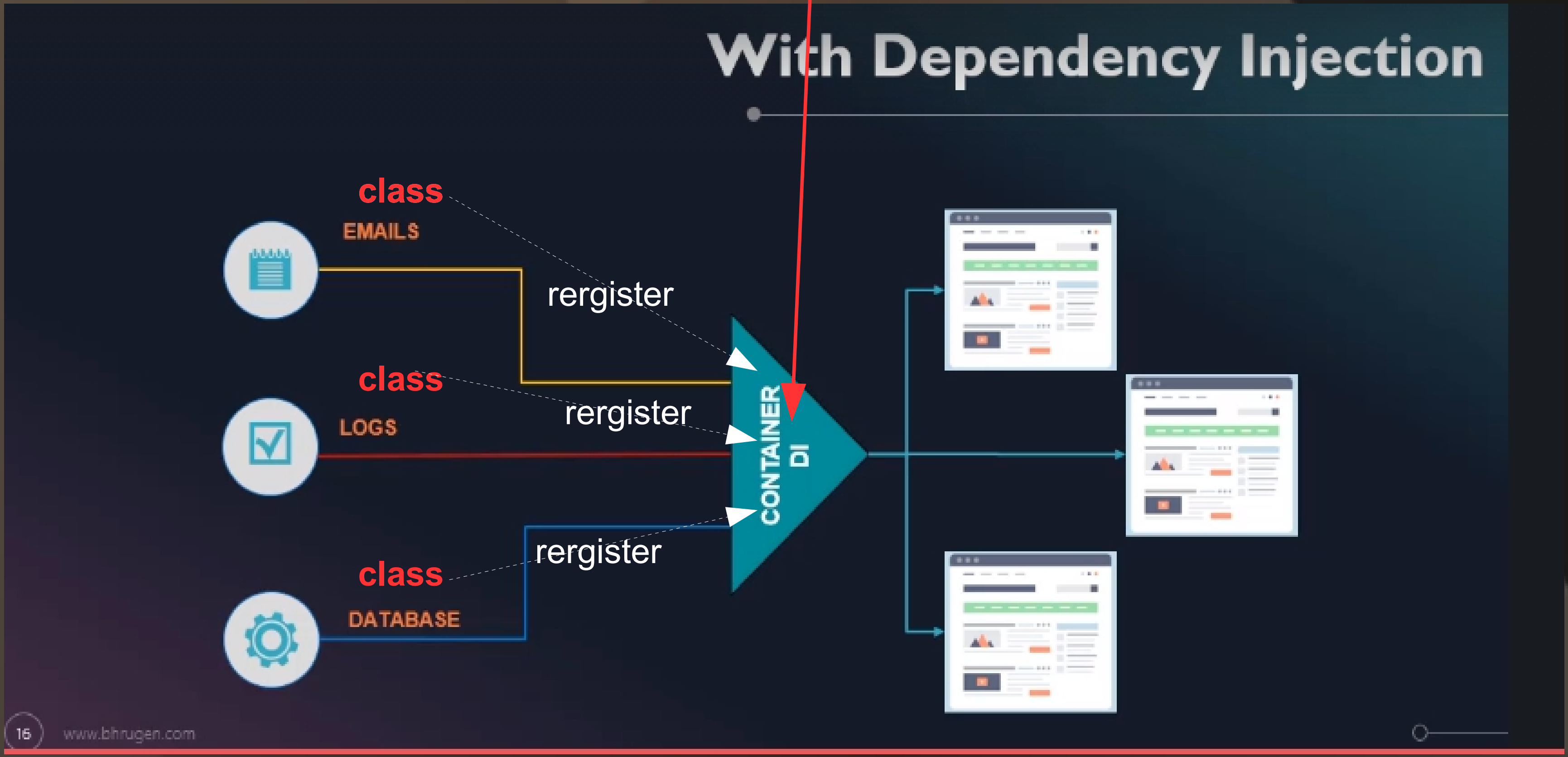
Let's imagine our application have 3 pages. in each page
We will need three functionalities for example:
Emails, logs, ore we need to save something to a database.
We will need to create objects of these functions.
Each object will contain different functionality.

In the past we were creating objects of Email, Logger, and Database in the very first page
Then we will do the same in the second, and a third page.



But this is different with Dependency injection.

We have same three pages with a three different functionalities, or the classes.
But this time we have Dependency Injection Container.
We will register all the 3 classes inside the container. Whenever any of the page will need anything
We will exctract it directly from the DI container. Rather then creating an individual object in
Individual pages.
It is created, and registered. We only have to use it.
This way DI Container deals with **creating** **registering**, **using** and **disposing** rather then creating them
In every page. This how dependecy injection works.



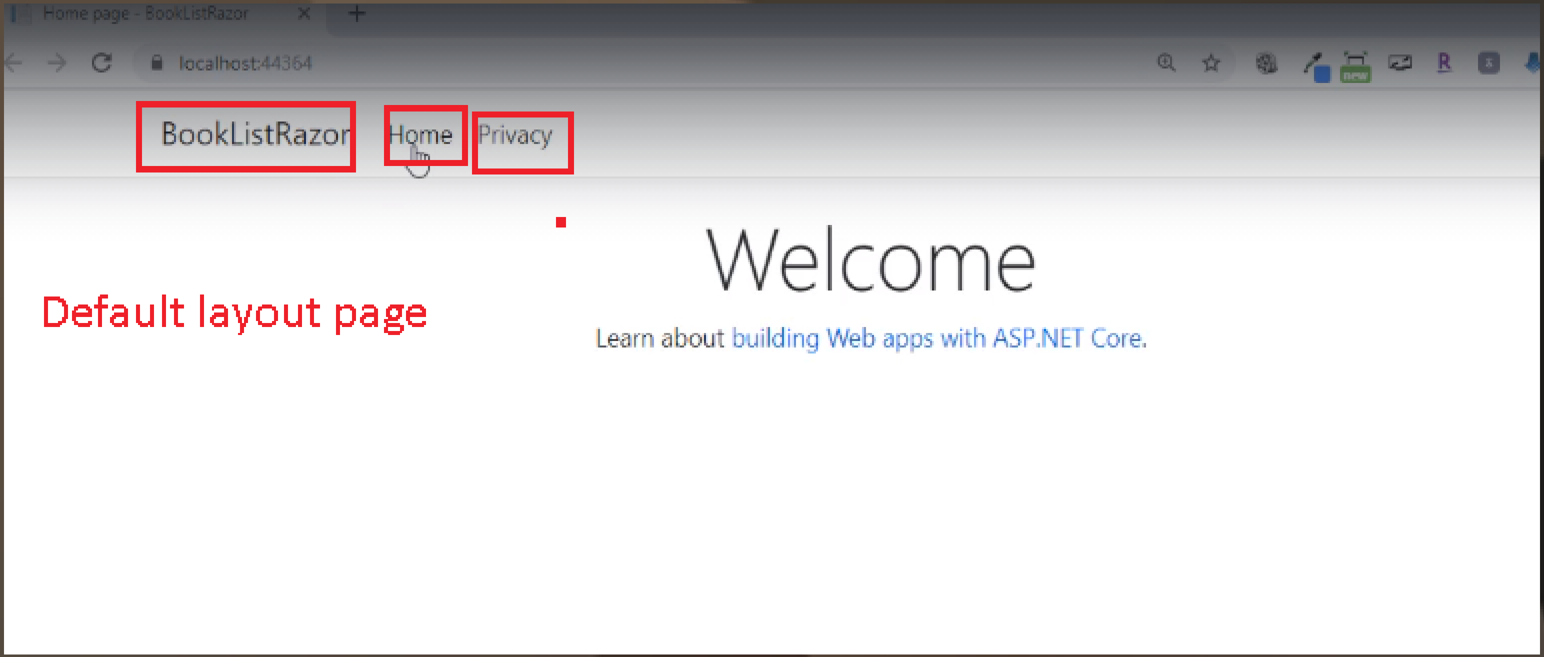
From now we will start creating all the functionality necessary to run this project

1. We will create a Model and push it to a database.
2. Perform CRUD operations on Book List.

Doing the above we will complete our Razor pages project

So let's get started!

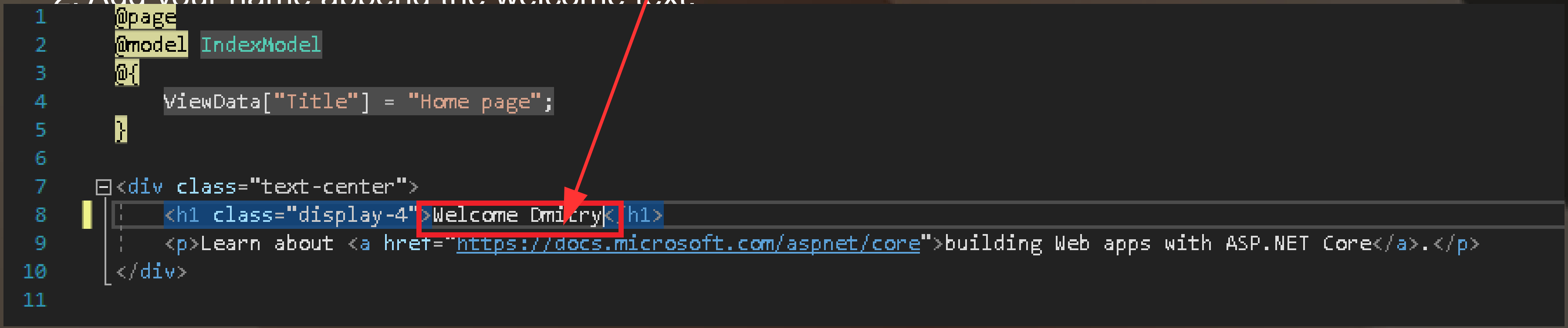
Let's install our first Nuget Package.
1. Run The application F5 and i'll show why we need that package?
By running the project it will open a default page.



Welcome

Learn about building Web apps with ASP.NET Core.

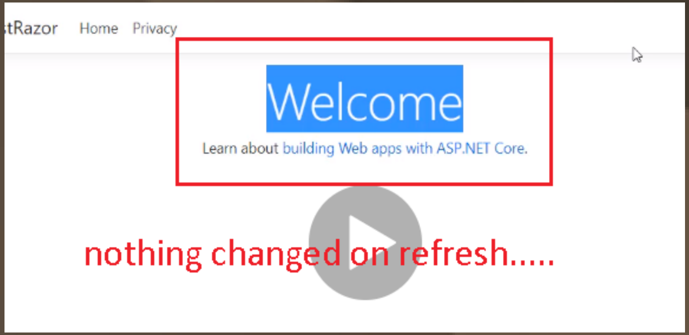
Don't stop running the project.
1. Go Back to Visual Studio, and open Index.schtml inside Pages folder
2. Add your name append the welcome text.



```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome Dmitry</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

3.Save the project CTRL+ S go back to a web page, and refresh the page to see the changes.

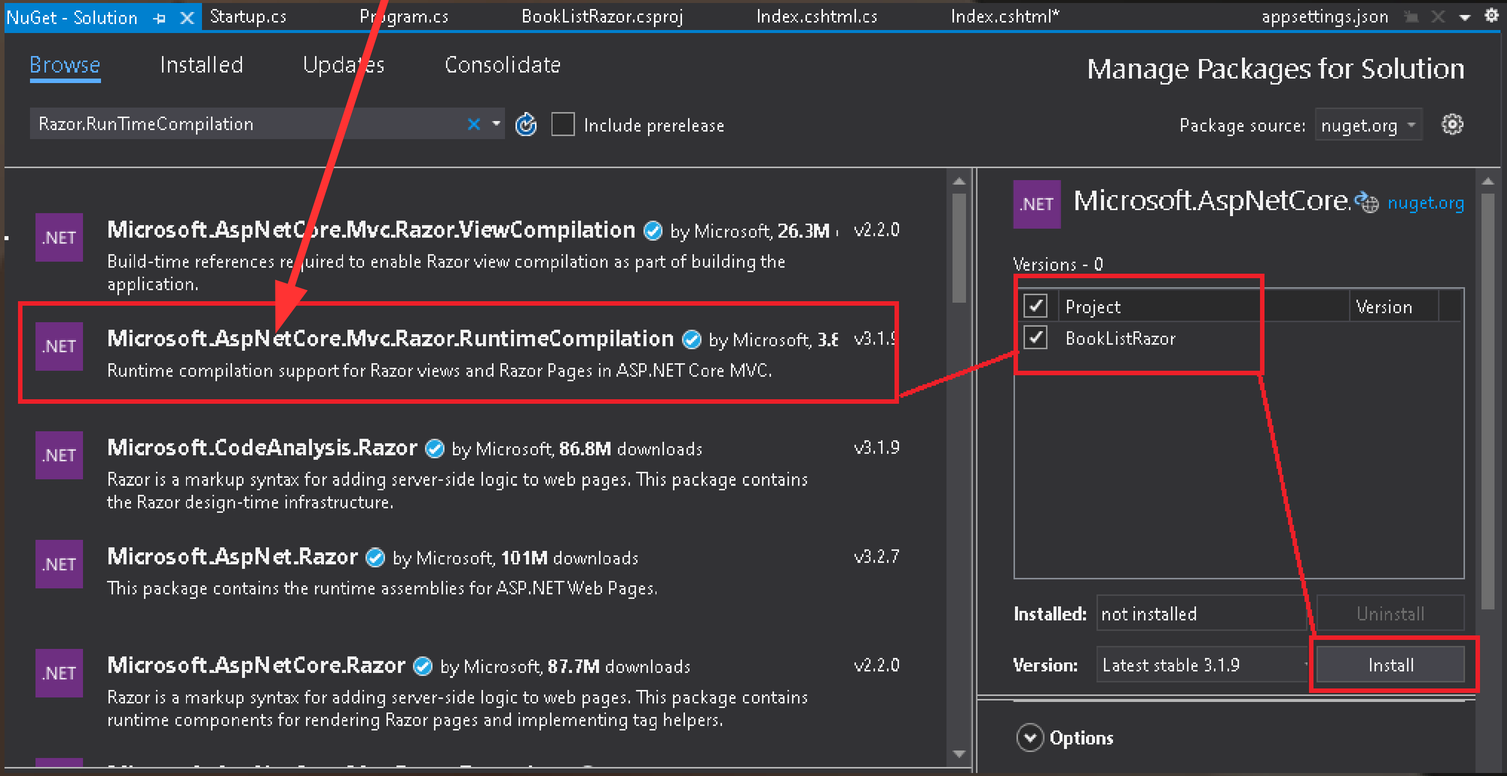
As you see the refresh was successful but the content here did not change!?


tRazor
Home
Privacy
Welcome
Learn about building Web apps with ASP.NET Core.
nothing changed on refresh.....

This was an exiisting feature befor ASP.NET CORE 3
But in this vertion of ASP.NET.CORE Microsoft desided to add a separate Nuget Packe
To enable a refresh feature.
1. Stop the application.
2. Open tools/NuGetPackageManager/ManageNugetPackagesForSolution
3. Inside the browse tab search for: **Razor.TimeCompilation**

4. Install the package.
5. Once installation is complete

We will need to add a few
Lines of code to our startup.cs file.



NuGet - Solution | Startup.cs | Program.cs | BookListRazor.csproj | Index.cshtml.cs | Index.cshtml* | appsettings.json

Browse | Installed | Updates | Consolidate

Manage Packages for Solution

Razor.RunTimeCompilation   Include prerelease

Package source: nuget.org

Microsoft.AspNetCore.Mvc.Razor.ViewCompilation by Microsoft, **26.3M** v2.2.0
Build-time references required to enable Razor view compilation as part of building the application.

Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation by Microsoft, 3.8 v3.1.9
Runtime compilation support for Razor views and Razor Pages in ASP.NET Core MVC.

Microsoft.CodeAnalysis.Razor by Microsoft, **86.8M** downloads v3.1.9
Razor is a markup syntax for adding server-side logic to web pages. This package contains the Razor design-time infrastructure.

Microsoft.AspNet.Razor by Microsoft, **101M** downloads v3.2.7
This package contains the runtime assemblies for ASP.NET Web Pages.

Microsoft.AspNetCore.Razor by Microsoft, **87.7M** downloads v2.2.0
Razor is a markup syntax for adding server-side logic to web pages. This package contains runtime components for rendering Razor pages and implementing tag helpers.

Microsoft.AspNetCore. nuget.org

Versions - 0

	Project	Version
✓	BookListRazor	

Installed: not installed   Uninstall
Version: Latest stable 3.1.9   Install

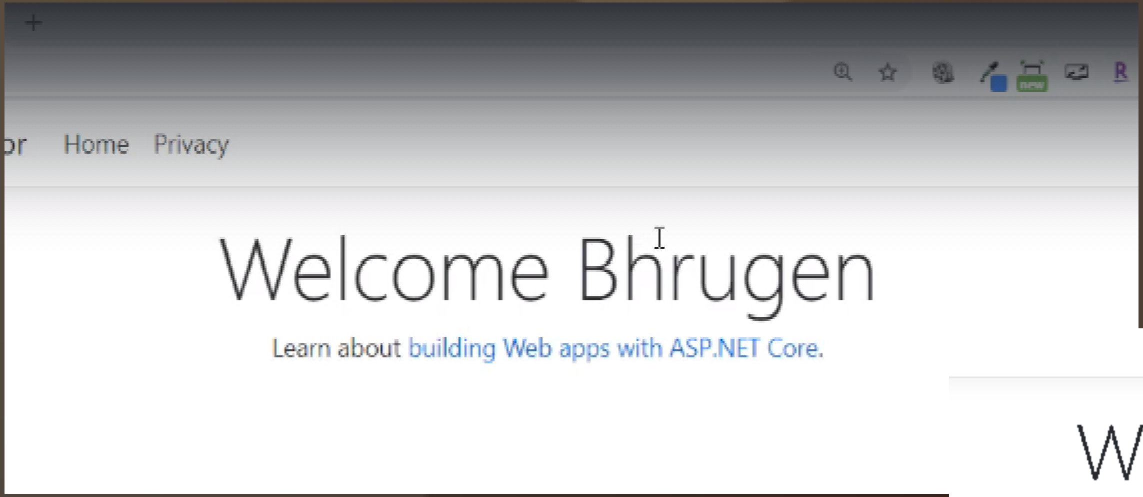
Options

1. Open startup.cs file
2. locate ConfigureServices() method.
3. We will add `AddRazorRuntimeCompilation();`

Code startup.cs:

```
 public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages().AddRazorRuntimeCompilation();
        }
```

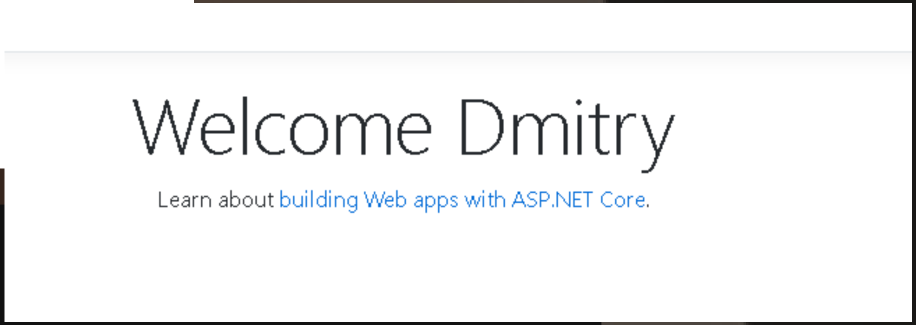
4. Click Save CTRL+S, ad run the project.
5. Check the default page the welcome message should change now to **Welcome Bhrugen**


Home Privacy

Welcome Bhrugen

Learn about building Web apps with ASP.NET Core.

Your name here...

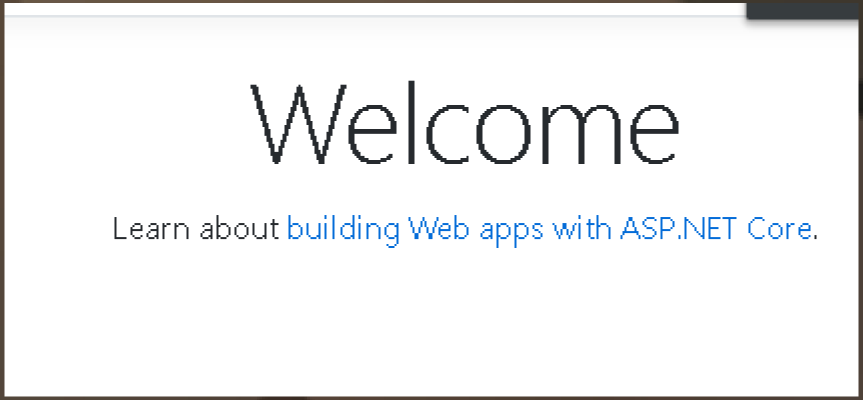

Welcome Dmitry

Learn about building Web apps with ASP.NET Core.

6. It Works now. Perfect!
7. switch back to visual studio while project is running.
8. Open index cshtml again and remove your name,
   And leave the word Welcome.
9.Go back to the browser and refresh the page, to see if your name
  Disappeared from page.

```
<div class="text-center">
    <h1 class="display-4">Welcome remove your name</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

This time it should automatically reload
And it showld display only Welcome.


Welcome

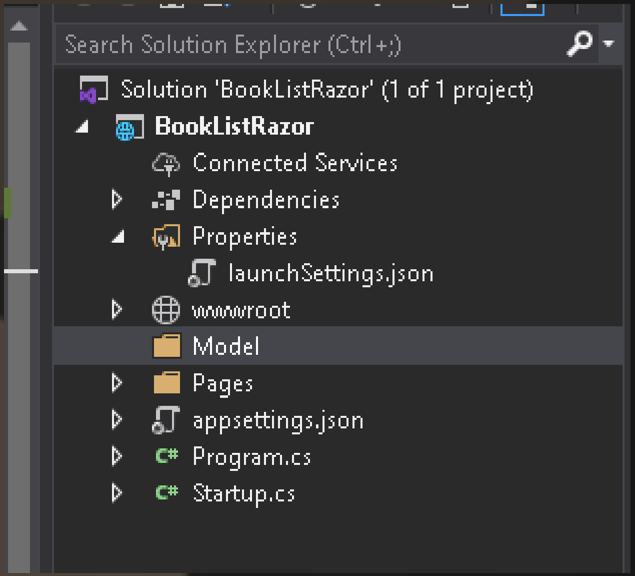
Learn about building Web apps with ASP.NET Core.

This enable you change the razor contents, while application is running.
Then Save the changes, and refresh the page. You should see changes.
It is achieved with a help of **Razor.TimeCompilation** package.
It helps to work on projects without stopping and re-running them again and again.

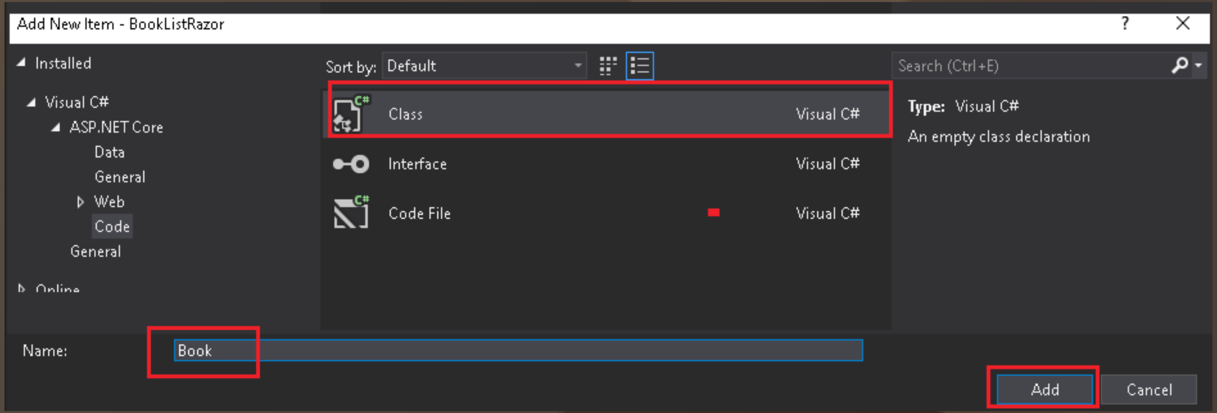
Building a Model

In this project we want to manage a list of books.
For this reazon we will have to create a Model class.
1. Right click you project, and select Add/New Folder
 Name it Model.

2. Rightclick the Model and select Add/New Class
3. Name your class as Book.cs



Search Solution Explorer (Ctrl+;)
Solution 'BookListRazor' (1 of 1 project)
BookListRazor
Connected Services
Dependencies
Properties
launchSettings.json
wwwroot
Model
Pages
appsettings.json
Program.cs
Startup.cs



Add New Item - BookListRazor

Installed
Visual C#
ASP.NET Core
Data
General
Web
Code
General
Online

Sort by: Default

Class	Visual C#
Interface	Visual C#
Code File	Visual C#

Search (Ctrl+E)

Type: Visual C#
An empty class declaration

Name: Book

4. We will add a few properties inside this class.

```
public class Book
    {
        [Key]
        public int Id { get; set; }

       [Required]
       public string Name { get; set; }
       public string Author { get; set;
    }
```

The **[Key]** Will automatically add id as an identity column,
So that way we dont have to pass the value.
It will create an Id Value automatically.

[required]
 means the name cannot be null.(in database)

5. Next step is to Add this model to a database.

See the next Slide.

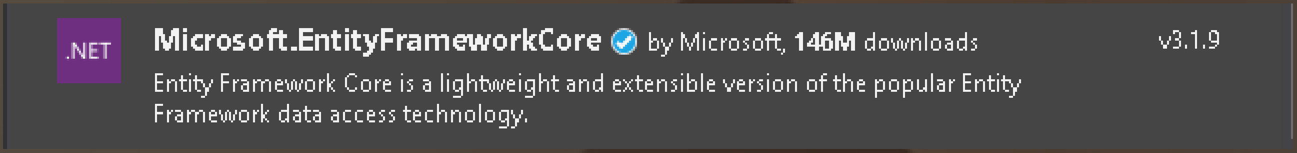
Before Creating a database we have to install the necessary NuGet packages

1. Setup Entity Framework
2. Setup a connection string.

Install packages

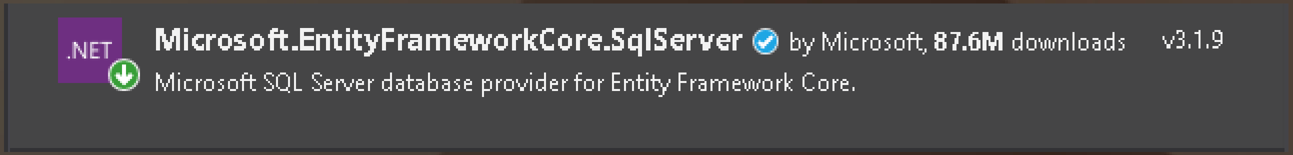
We will be using Entity Framework
To access the database.

1. Go to Tools. NugetPackagemanager/ManagePackagesForSolution
2. Search for **Microsoft.EntityFrameworkCore** install this package.



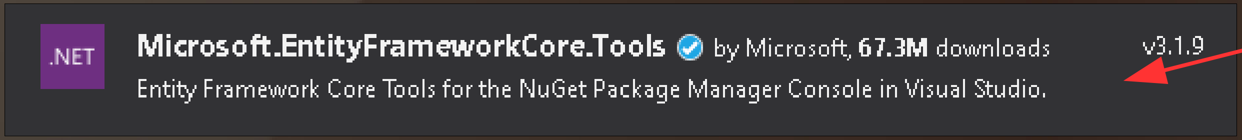
Microsoft.EntityFrameworkCore by Microsoft, **146M** downloads v3.1.9
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

3. search, and install: **Microsoft.EntityFrameworkCore.sqlServer**



Microsoft.EntityFrameworkCore.SqlServer by Microsoft, **87.6M** downloads v3.1.9
Microsoft SQL Server database provider for Entity Framework Core.

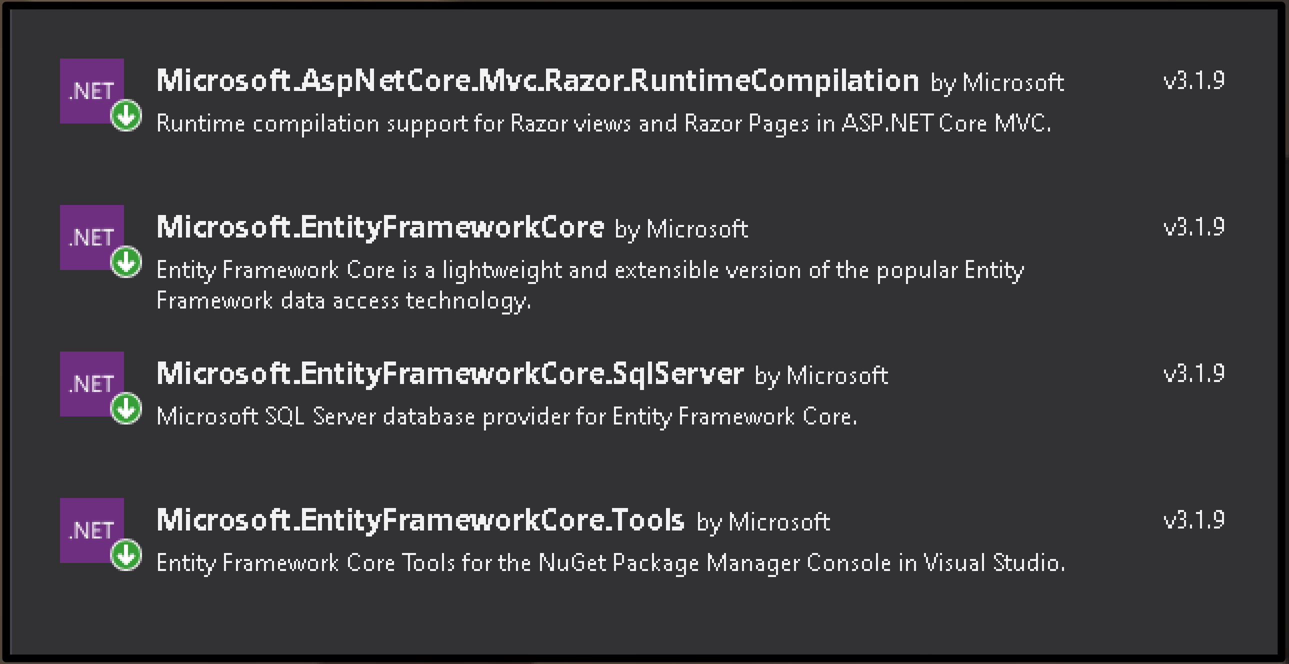
4. search and install: **Microsoft.EntityFrameworkCore.Tools**



Microsoft.EntityFrameworkCore.Tools by Microsoft, **67.3M** downloads v3.1.9
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Tools required because we will be running **Migrations**

So far we've installed these packages


Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation by Microsoft v3.1.9
Runtime compilation support for Razor views and Razor Pages in ASP.NET Core MVC.

Microsoft.EntityFrameworkCore by Microsoft v3.1.9
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

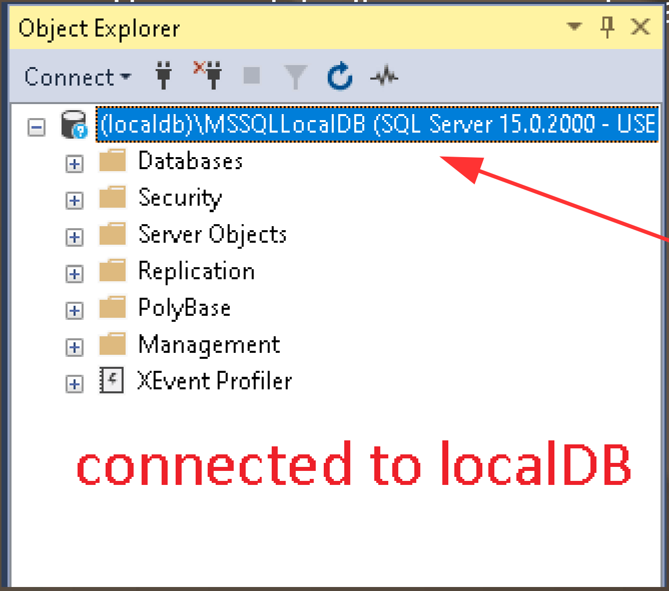
Microsoft.EntityFrameworkCore.SqlServer by Microsoft v3.1.9
Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.Tools by Microsoft v3.1.9
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Next step is: Setup our connection string.

- Make sure SqlServer is installed. (express version is free google it)
- Make sure LocalDb is installed.
- Make sure SqlServerManagment studio is installed (google it its free)

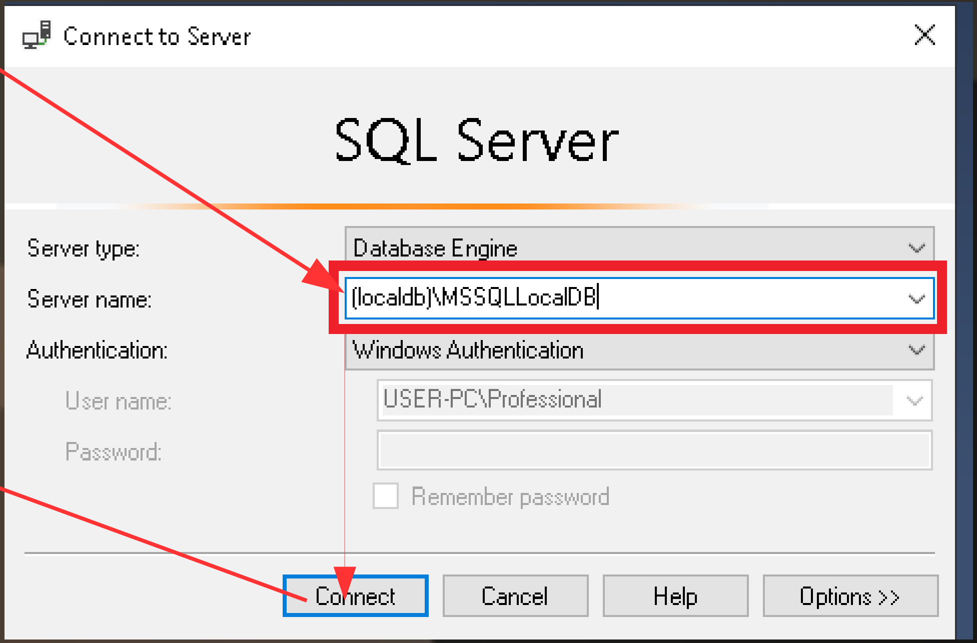
You have to 2 options:
1. Use this default connection
(localdb)\MSSQLLocalDB
a local Db


Object Explorer

Connect

(localdb)\MSSQLLocalDB (SQL Server 15.0.2000 - USE
- Databases
- Security
- Server Objects
- Replication
- PolyBase
- Management
- XEvent Profiler

connected to localDB


Connect to Server

SQL Server

Server type:	Database Engine
Server name:	(localdb)\MSSQLLocalDB
Authentication:	Windows Authentication
User name:	USER-PC\Professional
Password:	

☐ Remember password

We will be using this default server name **(localdb)\MSSQLLocalDB** within a connection string

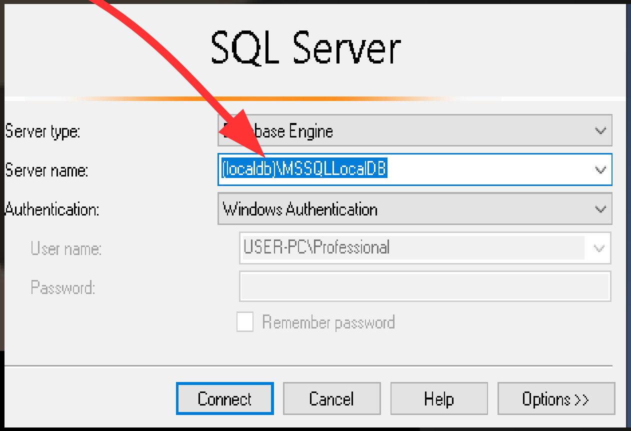
Connect to the automatic instance
This is Microsoft's default local DB connection string
The easiest way to use LocalDB is to connect to the automatic instance owned by the current user by using the connection string **Server=(localdb)\MSSQLLocalDB;Integrated Security=true**.
To connect to a specific database by using the file name, connect using a connection string similar to Server=(LocalDB)\MSSQLLocalDB;Integrated Security=true;AttachDbFileName=D:\Data\MyDB1.mdf.

1. Open Visual Studio, and open **appsettings.json** file
2. Just before logging section paste this code: (You canchange the name default, to any name you want)

```
 "ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB; Database = BookListRazor; Integrated Security=true."
 }
```

```
Basic connection string

 "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB; Database = BookListRazor; Integrated Security=true."
  },
```

* We used exact same server name as in sqlServer managment studio.
* We will create a new database called BookListRazor. Database= BookListRazor
   Make Sure you do not create this database from sql managment studio
* We will create a BookList Razor DataBase inside Visual Studio.
 Then we have trusted_connections set to true, and Multiple active results set to true



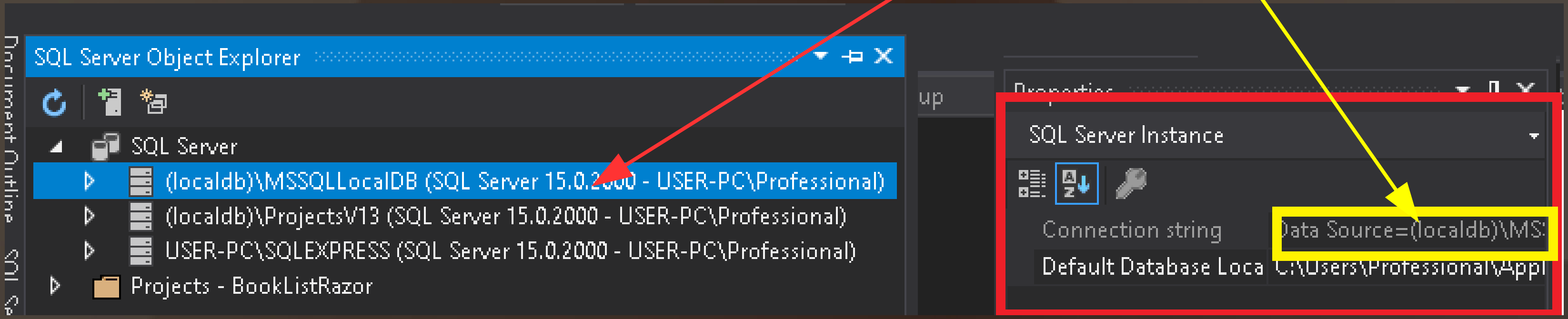
Server type: Database Engine
Server name: (localdb)\MSSQLLocalDB
Authentication: Windows Authentication
User name: USER-PC\Professional
Password:
Remember password

These properties are **Optional** properties: you can still use this connection string.
* *Trusted_Connection = true*
* *MultipleActiveResultSets= true*

```
Final connection string

"ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB; Database = BookListRazor;" +
                             "Integrated Security=true; MultipleActiveResultSets=true; Trusted_Connection=true"
  },
```

Easy Method that works 100%

Second option is to simply open a Visual studios' Server Object explorer
If a local Db is installed you will see it inside the Server explorer window.
Simply rightclick the local db and choose properties.
Go to properties window, and simply copy/paste the connection string from there



SQL Server
(localdb)\MSSQLLocalDB (SQL Server 15.0.2000 - USER-PC\Professional)
(localdb)\ProjectsV13 (SQL Server 15.0.2000 - USER-PC\Professional)
USER-PC\SQLEXPRESS (SQL Server 15.0.2000 - USER-PC\Professional)
Projects - BookListRazor

SQL Server Instance
Connection string Data Source=(localdb)\MS
Default Database Loca C:\Users\Professional\Appl

Next step Configure startup class file

1. Save the connection string CTRL+S and open **startup.cs**

Now that we have a connection string inside **appsettings.json**
It's time to configure our services with **Entity FrameWork.**
In order to configure that we need **ApplicationDbContext, or a DbContext Class**

1.Open **Model** folder right click and select **Add/New class.**
2.Create a new class Inside a Model folder, and give it aname of: **ApplicationDbContext**
3. **ApplicationDbContext** should inherit from **DbContext** Class which is a class inside
`Microsoft.EntityFrameworkCore.  Yes, don't be confused, this is going to be our BASE class`

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookListRazor.Model
{
    public class ApplicationDbContext :DbContext
    {
    }
}
```

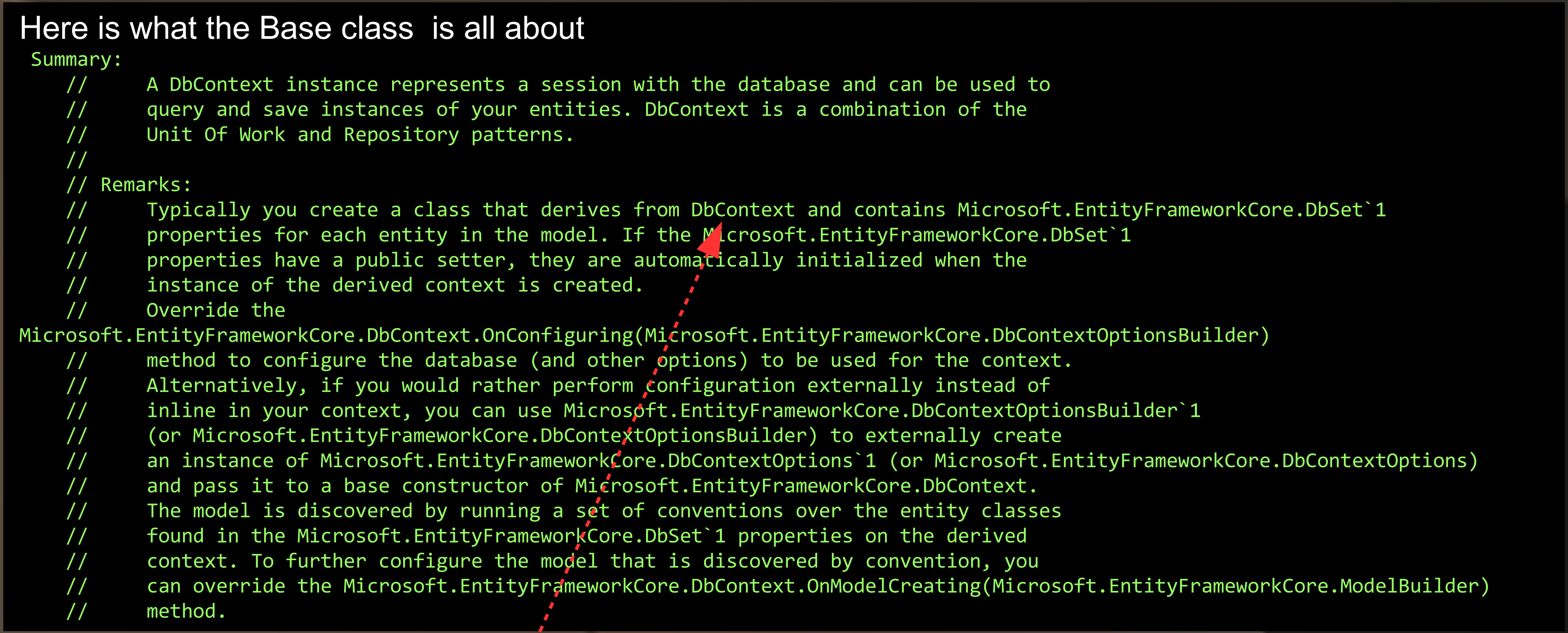
4. Next, we need to implement a **constructor**, and we
Have to pass a **DbContextOptions** as a parameter.

5. Create a constructor.
6. Pass a parameter named **options** of type
DbContextOptions<ApplicationDbContext>
To this constructor.

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

        }
    }
```

Please read this tutorial on Creating a base class constructors
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/base

In general:
We've created a class, and constructor that recieves a prarameter of type `DbContextOptions<ApplicationDbContext>` named **options**
When we create a new object of **this** class we passing a parameter **options** to a **base** class. A base class is always an inherited one.
This means that this inherited **DbContext** class must have a constructor that recieves **options** parameter. See next slide.....

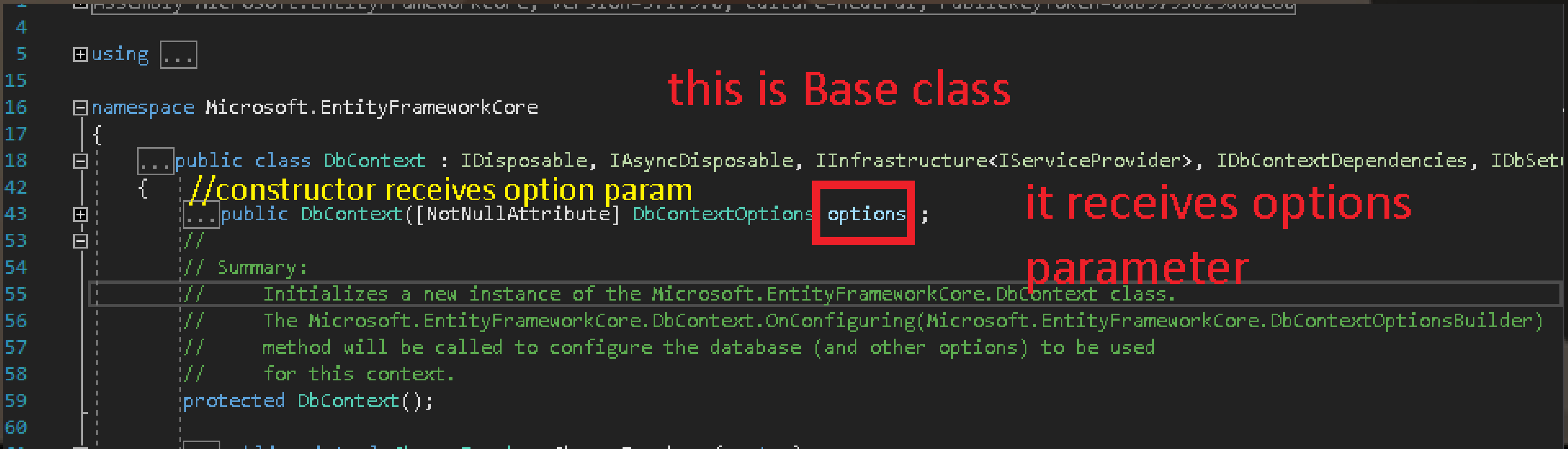


Here is what the Base class is all about

```
 Summary:
    //     A DbContext instance represents a session with the database and can be used to
    //     query and save instances of your entities. DbContext is a combination of the
    //     Unit Of Work and Repository patterns.
    //
    // Remarks:
    //     Typically you create a class that derives from DbContext and contains Microsoft.EntityFrameworkCore.DbSet`1
    //     properties for each entity in the model. If the Microsoft.EntityFrameworkCore.DbSet`1
    //     properties have a public setter, they are automatically initialized when the
    //     instance of the derived context is created.
    //     Override the
Microsoft.EntityFrameworkCore.DbContext.OnConfiguring(Microsoft.EntityFrameworkCore.DbContextOptionsBuilder)
    //     method to configure the database (and other options) to be used for the context.
    //     Alternatively, if you would rather perform configuration externally instead of
    //     inline in your context, you can use Microsoft.EntityFrameworkCore.DbContextOptionsBuilder`1
    //     (or Microsoft.EntityFrameworkCore.DbContextOptionsBuilder) to externally create
    //     an instance of Microsoft.EntityFrameworkCore.DbContextOptions`1 (or Microsoft.EntityFrameworkCore.DbContextOptions)
    //     and pass it to a base constructor of Microsoft.EntityFrameworkCore.DbContext.
    //     The model is discovered by running a set of conventions over the entity classes
    //     found in the Microsoft.EntityFrameworkCore.DbSet`1 properties on the derived
    //     context. To further configure the model that is discovered by convention, you
    //     can override the Microsoft.EntityFrameworkCore.DbContext.OnModelCreating(Microsoft.EntityFrameworkCore.ModelBuilder)
    //     method.
```

So we cretaed a class that derives from DbContext

BY selecting DbContext and pressing F12 we can inspect its contents


this is Base class
//constructor receives option param
it receives options parameter

```
using ...

namespace Microsoft.EntityFrameworkCore
{
    ...public class DbContext : IDisposable, IAsyncDisposable, IInfrastructure<IServiceProvider>, IDbContextDependencies, IDbSet
    {
        ...public DbContext([NotNullAttribute] DbContextOptions options);
        //
        // Summary:
        //     Initializes a new instance of the Microsoft.EntityFrameworkCore.DbContext class.
        //     The Microsoft.EntityFrameworkCore.DbContext.OnConfiguring(Microsoft.EntityFrameworkCore.DbContextOptionsBuilder)
        //     method will be called to configure the database (and other options) to be used
        //     for this context.
        protected DbContext();
```

1.So far we've created ApplicationDbContext class
2. we've created a constructor that passes option parameter to a base class

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookListRazor.Model
{
    public class ApplicationDbContext :DbContext
    {
        //constructor-1
        //here we passing options parameter of type DbContextOptions<ApplicationDbContext>
        //to the base class-DbContext, which is a class from Microsoft.EntityFrameworkCore;
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options): base(options)
        {
           // nothing here

        }
    }
}
```

This is an empty constructor, but the parameter needed for **dependecy injection**

Step 3

3. Let's add a Book Model

In order to add any model to a data base inside a „DBContext" in our case **ApplicatiopnDbContext** class We need to create an **entry point of type DBSet <yourmodelHere>**

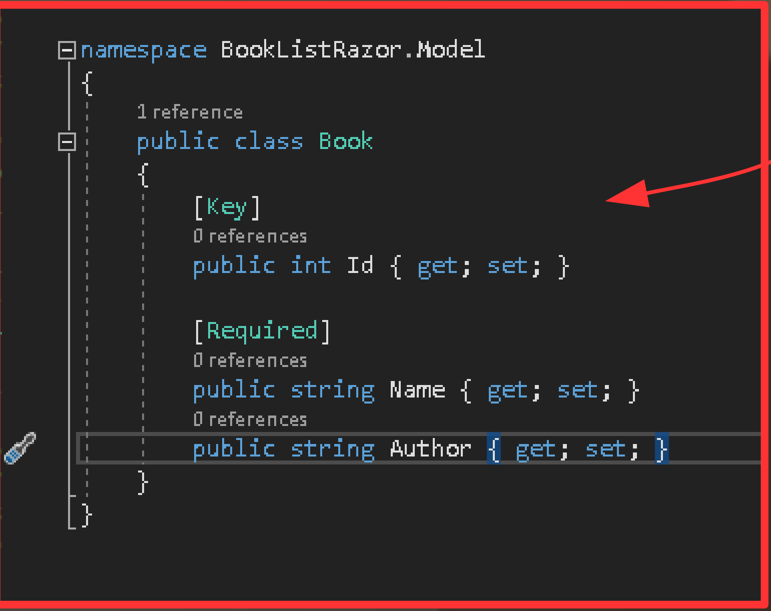
```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookListRazor.Model
{
    public class ApplicationDbContext :DbContext
     {
        //constructor-1
        //here we passing options parameter of type DbContextOptions<ApplicationDbContext>
        //to the base class-DbContext, which is a class from Microsoft.EntityFrameworkCore;

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options):
base(options)
          {
           //Nothing here. This is an empty constructor.
           //The parameter needed for the dependency injection.
          }

        public DbSet<Book> Book { get; set; }
     }
}
```

Once you added the Book inside a DbContext, next step is to add it inside **startup.cs** file

4. Open **startup.cs** file. Let's add a dbContext to our pipeline.
5.Locatre `ConfigureServices`() method and add the folowing code inside this method.

```
public void ConfigureServices(IServiceCollection services)
     {
         services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
         services.AddRazorPages().AddRazorRuntimeCompilation();
     }
```

This is the configuration we had to do to add the entity framework inside our pipeline.
Once its done all you have to do is to **push this** into a database.



```
namespace BookListRazor.Model
{
    1 reference
    public class Book
    {
        [Key]
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public string Name { get; set; }
        0 references
        public string Author { get; set; }
    }
}
```

1. Go to Tools/NugetPackageManager/PackageManagerConsole

2. Enter the following command inside console

Add-migration AddBookToDB

3. This will create a script that will execute iside a database.

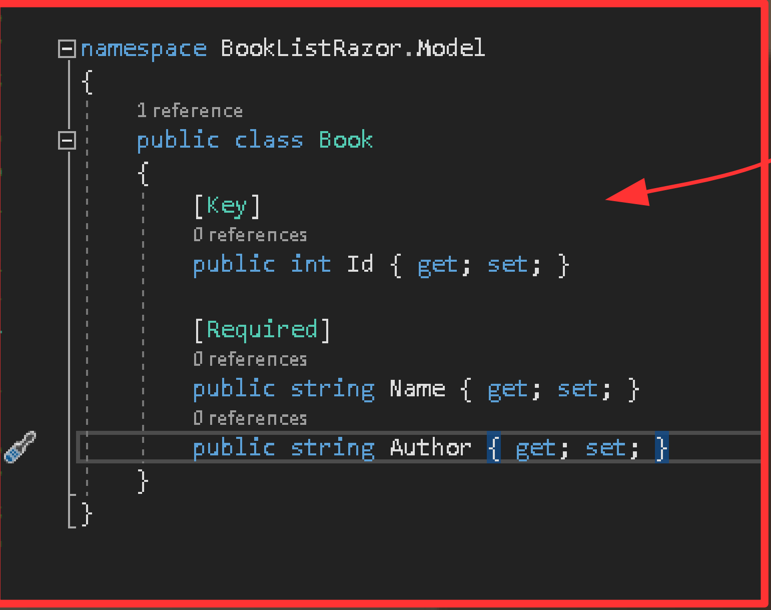
4 .Proceed to the next slide.

Once you added the Book inside a DbContext, next step is to add it inside **startup.cs** file

4. Open **startup.cs** file. Let's add a dbContext to our pipeline.
5.Locatre `ConfigureServices`() method and add the folowing code inside this method.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
            services.AddRazorPages().AddRazorRuntimeCompilation();
        }
```

This is the configuration we had to do to add the entity framework inside our pipeline.
Once its done all you have to do is to **push this** into a database.



```
namespace BookListRazor.Model
{
    1 reference
    public class Book
    {
        [Key]
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public string Name { get; set; }
        0 references
        public string Author { get; set; }
    }
}
```

1. Go to Tools/NugetPackageManager/PackageManagerConsole

2. Enter the following command inside console

Add-migration AddBookToDB

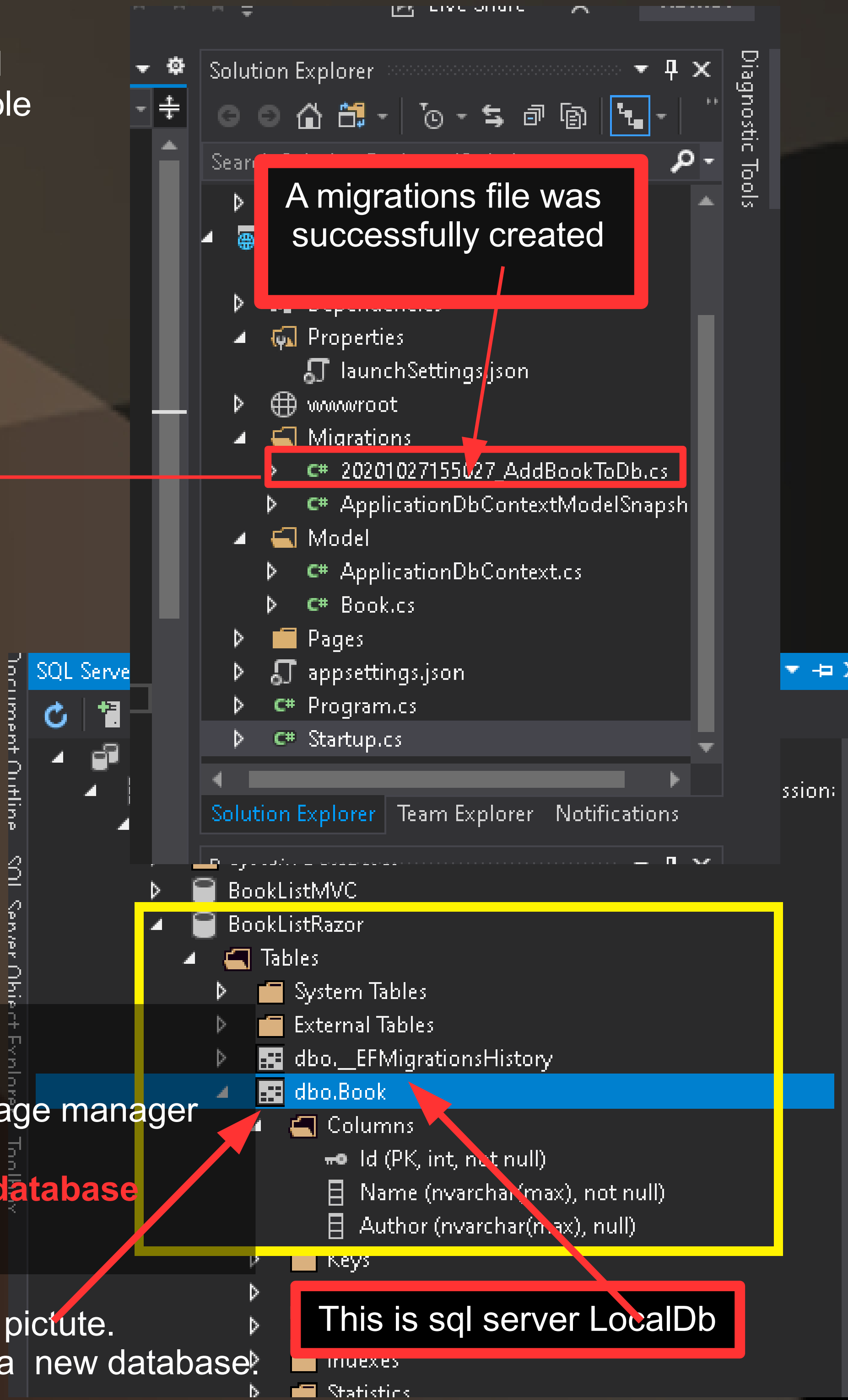
3. This will create a script that will execute iside a database.

4 .Proceed to the next slide.

Congratulations! You've just executed **add-migration** command.
Now using this C# script file you can execute it. This will create
A Physical database. To Execute the migration script we will need
Run **update-database** command inside PackageManager Console

Contents of Migration file

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace BookListRazor.Migrations
{
    public partial class AddBookToDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Book",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(nullable: false),
                    Author = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Book", x => x.Id);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Book");
        }
    }
}
```



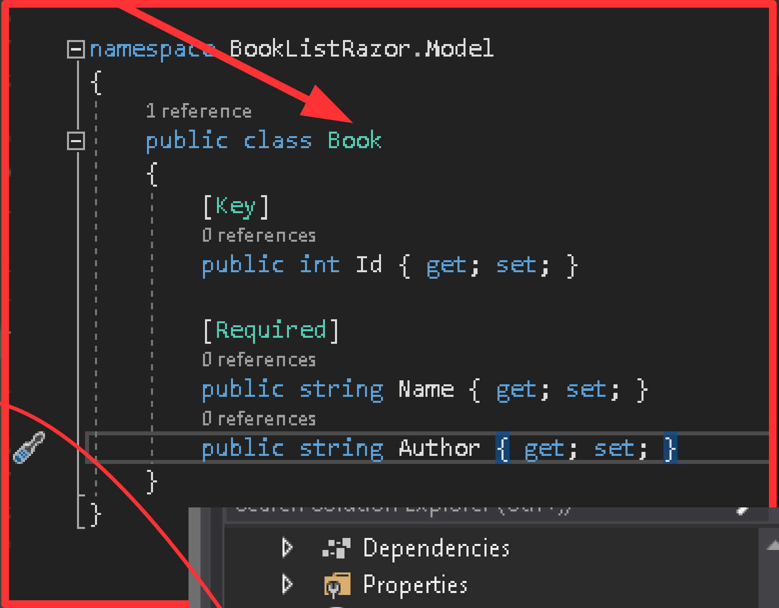
Later on this script will automatically create a table called Book.
It will add Columns Id, Name, and author.
As i mentioned before Next step is to execute this script inside Package manager Console to create a physical database inside sqlLocalDb
1. Open NugetPackageManagerConsole window, and type **update-database** command

After running this scrip, check if a table was created, a seen on this picture.
This will also check if a database already exists, if not it will create a new database.
It will also push any other migration-files (if exists) into a database.

The purpose of this application is to perform **CRUD** operations on our **Book Object**
To implement these features we need to create **special Razor Pages**, where
Each page correspond to each functionality

1. Create a new book. Page
2. Edit a book. Page
3. Delete a book Page
4. View all available books. Page

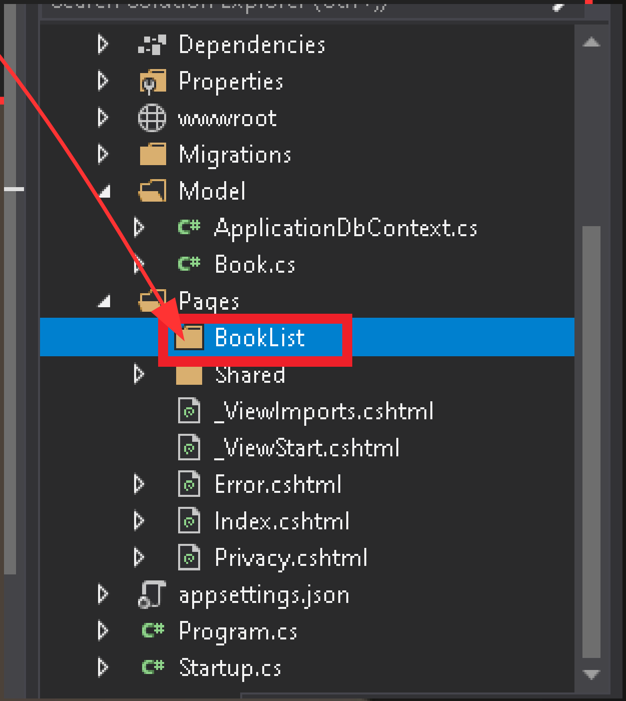
The above pages will be added to a separate folder named **BookList**



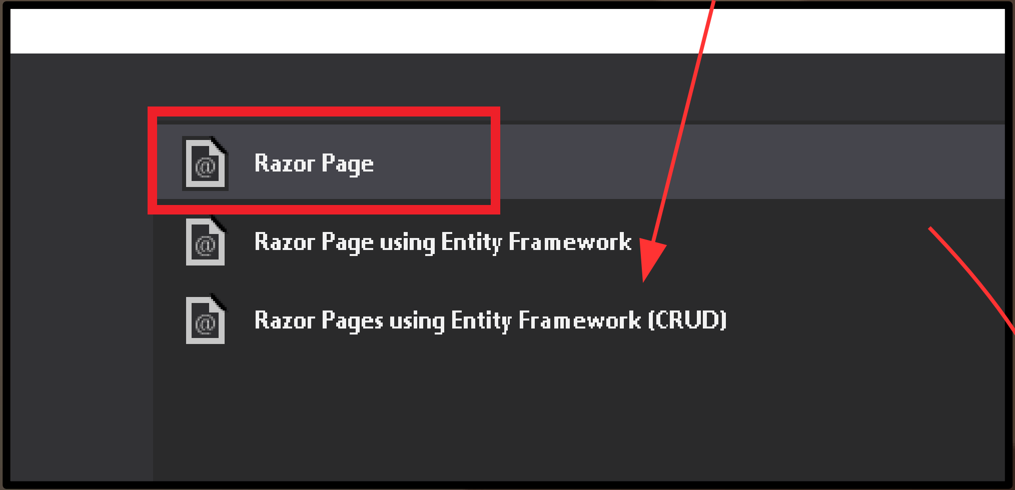
```
namespace BookListRazor.Model
{
    1 reference
    public class Book
    {
        [Key]
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public string Name { get; set; }
        0 references
        public string Author { get; set; }
    }
}
```

Next Step

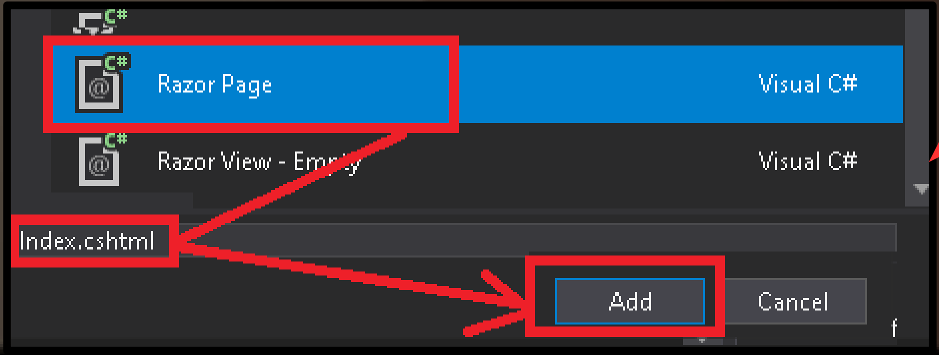
1. Open Sulution Explorer.
2. Create a new folder **BookList** inside **Pages** Folder../Pages/BookList
3. Add a new empty razor page inside BookList folder:



```
Dependencies
Properties
wwwroot
Migrations
Model
    ApplicationDbContext.cs
    Book.cs
Pages
    BookList
    Shared
    _ViewImports.cshtml
    _ViewStart.cshtml
    Error.cshtml
    Index.cshtml
    Privacy.cshtml
appsettings.json
Program.cs
Startup.cs
```


Razor Page
Razor Page using Entity Framework
Razor Pages using Entity Framework (CRUD)

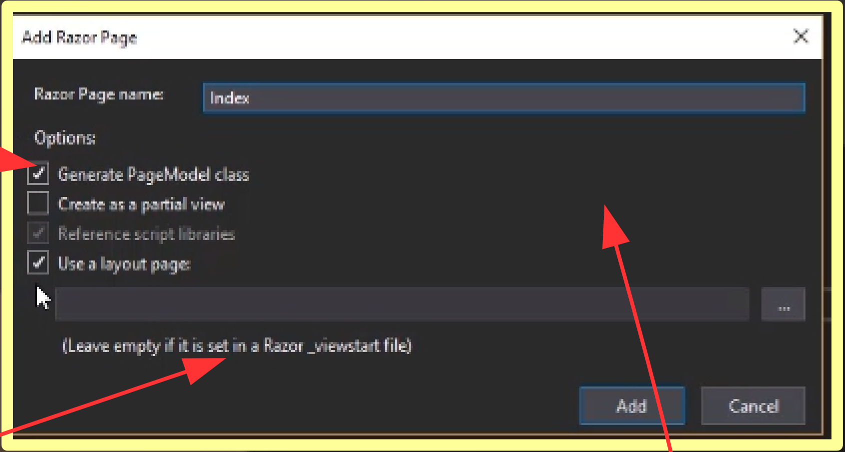
4. Give it a name of **Index.cshtml**



If you usung the older version of Visual studio, you will recieve this window.

4. We have some different options here:
First option is to create a Page model.
We need a page model class because, we need to populate all of the books from the database, and pass this information to our page(**Index.cshtml**) to display them.
This page is **NOT** a Partial View! So don't check this option.

Partial View it is a small subsection, like a group of buttons that you want to reuse in multiple pages.


Add Razor Page

Razor Page name: Index

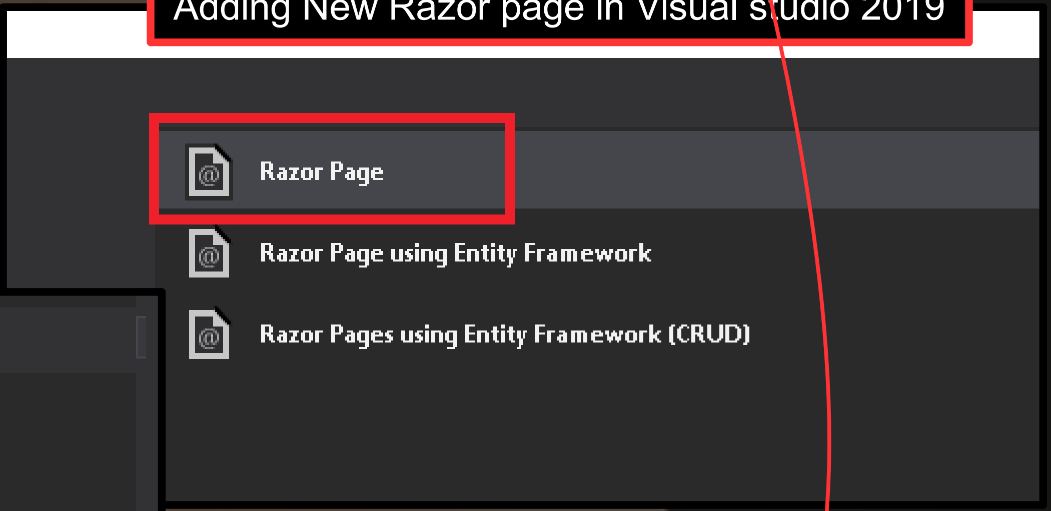
Options:

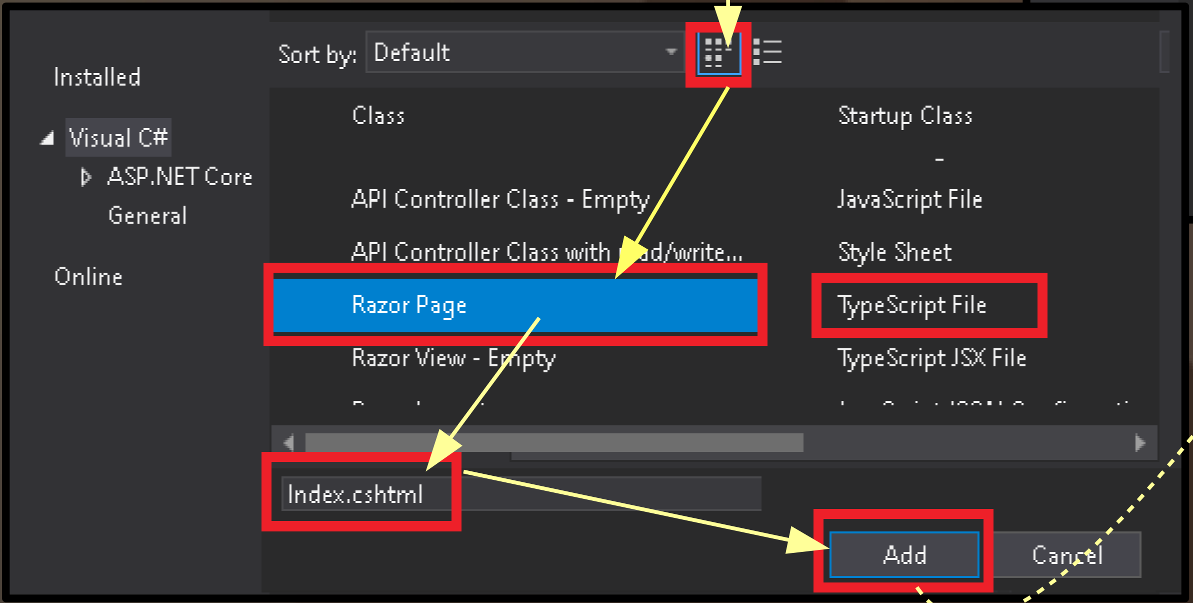
- [✓] Generate PageModel class
- [ ] Create as a partial view
- [✓] Reference script libraries
- [✓] Use a layout page:

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

If you using a latest version of Visual Studio 2019
You will not see this dialog window
Do the following:
1.When adding a new razor page choose **Razor Page**
2.Then Choose Razor page (**typescript File**)

Adding New Razor page in Visual studio 2019


Razor Page
Razor Page using Entity Framework
Razor Pages using Entity Framework (CRUD)



3. click ADD.
This will create a Razor Page with c# model class exactly as in the previous example.

Let's now work on our Pages/BookList/**Index.cshtml** file.

Inside **Index.cshtml.cs (model)** we want to retrieve all of the books from our Database.
For that we need **ApplicationDbContext** to be injected into this page.

Next step

1.Open **Index.cshtml.cs** file
2. Create a private readonly of **ApplicationDbCointext** variable
3. Next we have to initiate our constructor `IndexModel`

```
public class IndexModel : PageModel
   {

        private readonly ApplicationDbContext _Db;
        public void OnGet()
        {
        }
   }
```

```
public class IndexModel : PageModel
   {

       private readonly ApplicationDbContext _Db;

       public IndexModel(ApplicationDbContext db)
       {
           _Db = db;
       }
       public void OnGet()
       {
       }
   }
```

This way we extract the application **DbContext** and **Inject** it into **Index.cshtml** page.

4. Create **IEnumerable<Book> Books** property

```
public class IndexModel : PageModel
    {

        private readonly ApplicationDbContext _Db;

        public IndexModel(ApplicationDbContext db)
        {
            _Db = db;
        }

        public IEnumerable<Book> Books { get; set; }
        public void OnGet()
        {
        }
    }
```

5. Implement **OnGet()** method by removing **void** keyword,
and adding **async** keyword, and a **Task** keyword before **Onget()**
See the below code:

```
public class IndexModel : PageModel
    {

        private readonly ApplicationDbContext _Db;

        public IndexModel(ApplicationDbContext db)
        {
            _Db = db;
        }

        public IEnumerable<Book> Books { get; set; }

        public async Task OnGet()
        {
            Books = await _Db.Book.ToListAsync();
        }
    }
```

Here we are invoke a OnGet() Method .
We are going to a database, and retrieving all the books, and storing them inside **IEnumerable<>** object called **Books**

Next step, is to setup our index.cshtml so we could display the contents of this IEnimerable<book> Books opbject on the screen (in webbrowser)

Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel
@{
    ViewData["Title"] = "Index";
}
```

Later we will implement a logic to display the books
Insede this file

```
<h1>Book List Index</h1>
```

Remarks:
Click on links to learn more about These subjects:

The **await** operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes.

Use the **async** modifier to specify that a method, lambda expression, or anonymous method is asynchronous. If you use this modifier on a method or expression, it's referred to as an async method.

The **Task** Parallel Library (TPL) is based on the concept of a task, which represents an asynchronous operation. In some ways, a task resembles a thread or ThreadPool work item, but at a higher level of abstraction. The term task parallelism refers to one or more independent tasks running concurrently. Tasks provide two primary benefits:

Additional info

Async Enambles you to run multiple tasks at a time until it is awaited

```
Books = await _Db.Book.ToListAsync();
```

Here we need to await until all the books being found.

Index.cshtml.cs

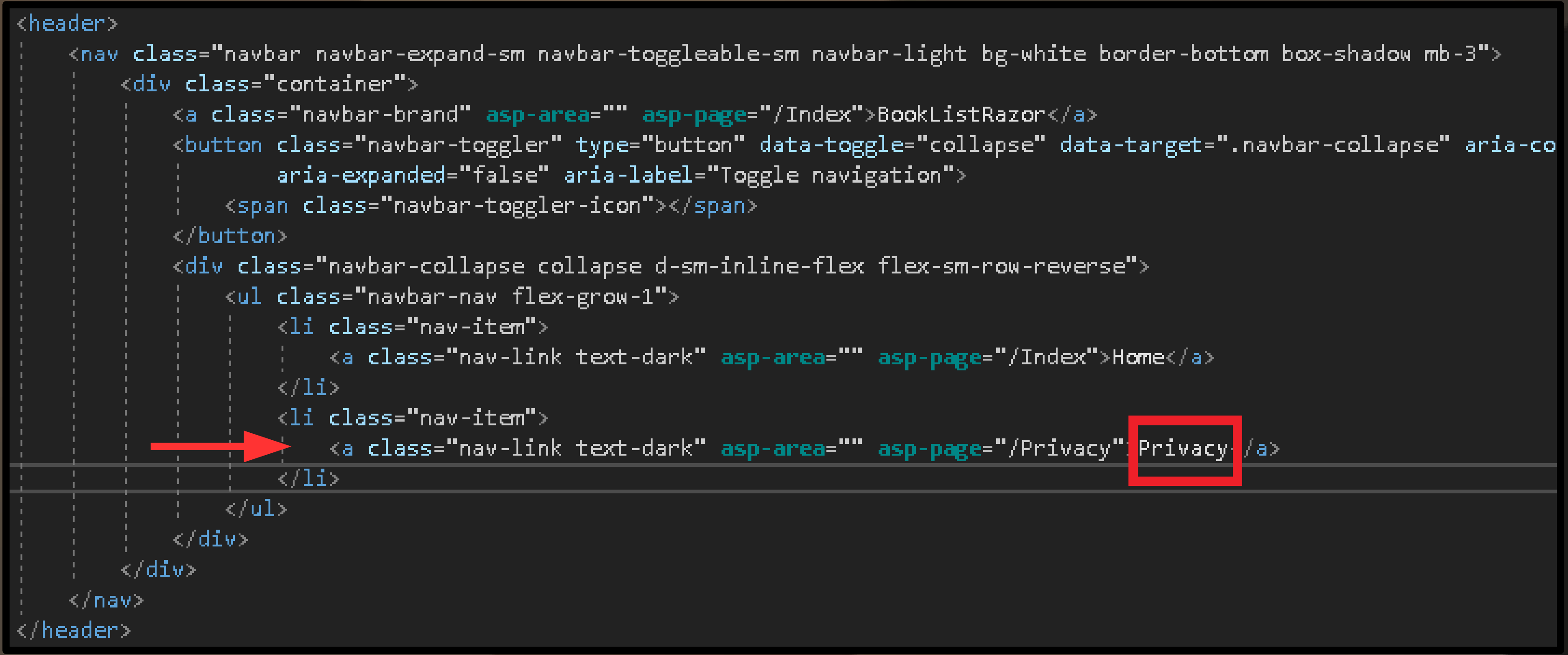
```
public async Task OnGet()
       {
            Books = await _Db.Book.ToListAsync();
       }
```

OnGet Method it is also async Task.
If it was **MVC**, and not **Razor**. We would use **Action** Methods insted.
But with Razor pages, inside the page model, we have **handlers**

Okay

It's time to display the books and add some UI

1.Go to solution Explorer.
2. Open Page/Shared/_Layout.cshtml
3. Search privacy link inside <header> section of the page



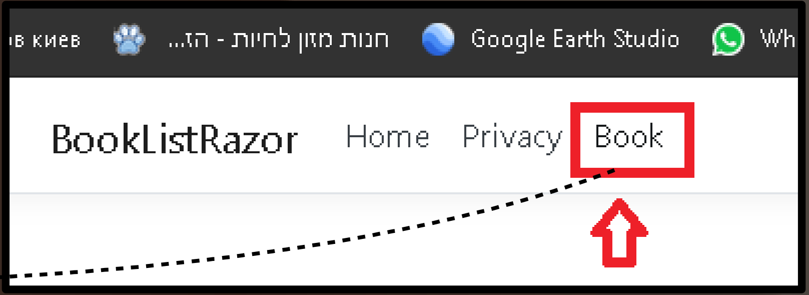
```
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
        <div class="container">
            <a class="navbar-brand" asp-area="" asp-page="/Index">BookListRazor</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-co
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                <ul class="navbar-nav flex-grow-1">
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
```

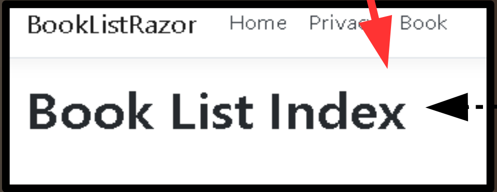
4. change this line of code so it matches this code

```
<a class="nav-link text-dark" asp-area="" asp-page="/Booklist/Index">Book</a>
```

5. Save application and run the project **F5**
6. Click the Book link.

You should see this output:





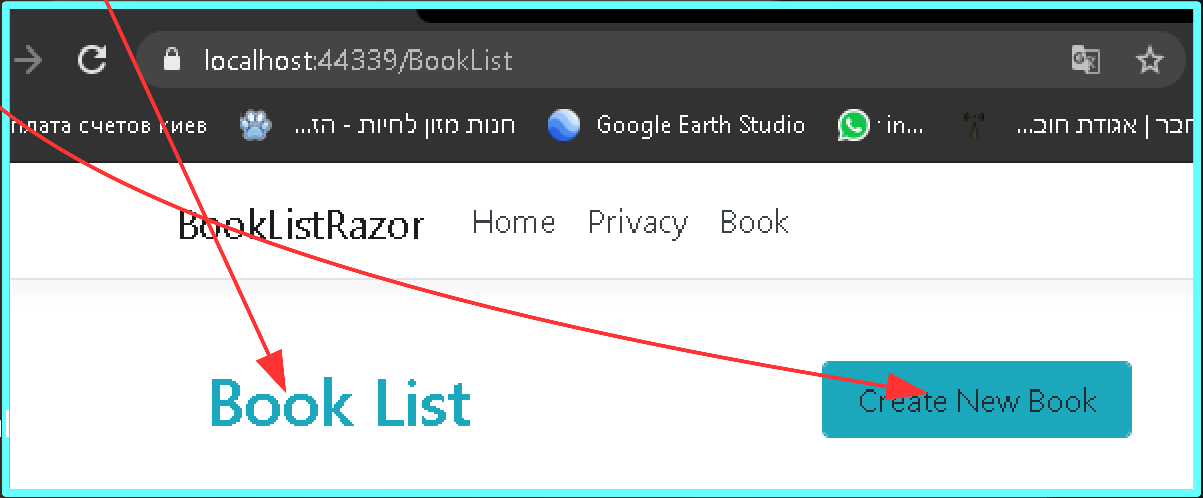
7. While project is running open Pages/BookList/Index.cshtml
8. Add the following code to the page:

Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel

<br />
<div class="container row p-0 m-0" >


    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a class=" btn btn-info form-control text-white">Create New Book</a>

    </div>
</div>
```

Bootstrap divides a page into 12 columns.
That's why first div is the size of a 10 columns, and div 2 is the size of 2 columns. 10+2=12

9. save this page, and refresh the webpage page.
You should get the following output.
Bootstrap classes helped us to style the button, and label.

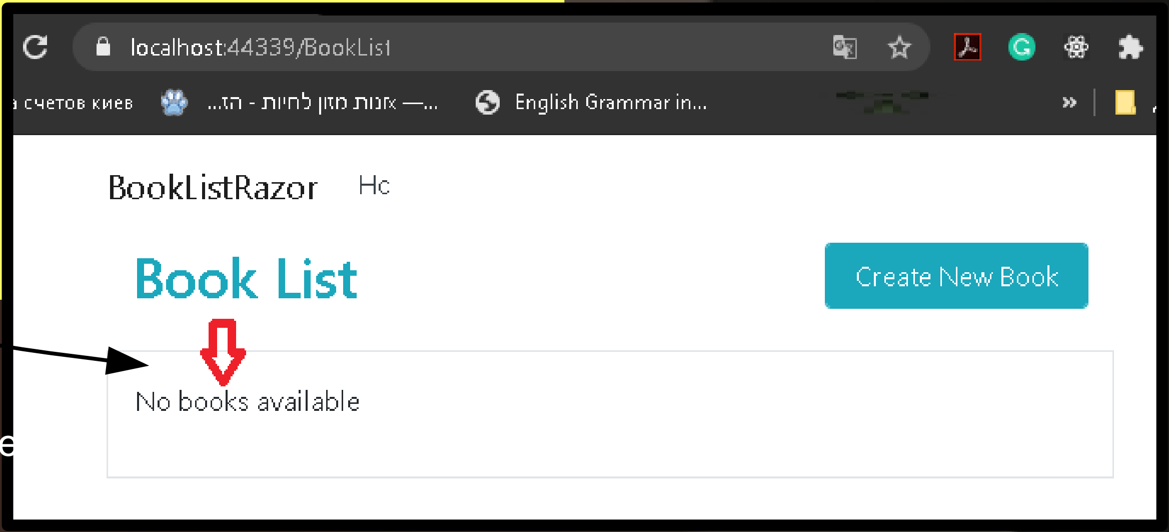


10. Next step is to create a table inside Index,cshtml
This new table will display book information.
Proceed to the next slide.

11. Let's implement a books table feature.
Open index,cshtml and copy the below code to match yours

Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel

<br />
<div class="container row p-0 m-0" >


    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a class=" btn btn-info form-control text-white">Create New Book</a>

    </div>

    <div class="col-12 border p-3 mt-3">
        <form method="post">
            @if (Model.Books.Count() > 0)
            {
                //displaya table
            }
             else{
                <p>No books available</p>
            }
        </form>
    </div>
</div>
```


localhost:44339/BookList

BookListRazor

Book List

Create New Book

No books available

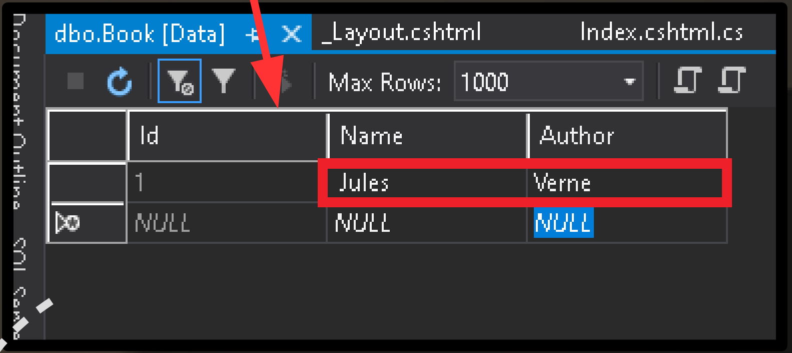
We try to cycle through the Book.Count, and check
Whether it contains a book, or not.
If a book exists do somethig, if not, display a message

As you can see from a screenshot there is no books in our table so far.
We will have to create a fake table in the next example within SqlServer Managment studio. Or VS Server Object Explorer

12. Before you proceed, open SqlServer Managment studio, and add a fake book to a table.

Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel

<br />
<div class="container row p-0 m-0" >
    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a class=" btn btn-info form-control text-white">Create New Book</a>

    </div>

    <div class="col-12 border p-3 mt-3">
        <form method="post">
            @if (Model.Books.Count() > 0)
            {
                //displaya table
                <table class="table table-striped border">
                    <tr class="table-secondary">
                        <th>
                            <label asp-for="Books.FirstOrDefault().Name"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().Author"></label>
                        </th>
                        <th>

                        </th>
                    </tr>

                    @foreach (var item in Model.Books)
                    {
                <tr>
                    <td>
                        @Html.DisplayFor(m => item.Name)
                    </td>
                    <td>
                        @Html.DisplayFor(m => item.Author)
                    </td>
                </tr>
                    }
                </table>
            }
             else{
                <p>No books available</p>
            }
        </form>
    </div>
</div>
```



Max Rows: 1000

	Id	Name	Author
	1	Jules	Verne
	NULL	NULL	NULL

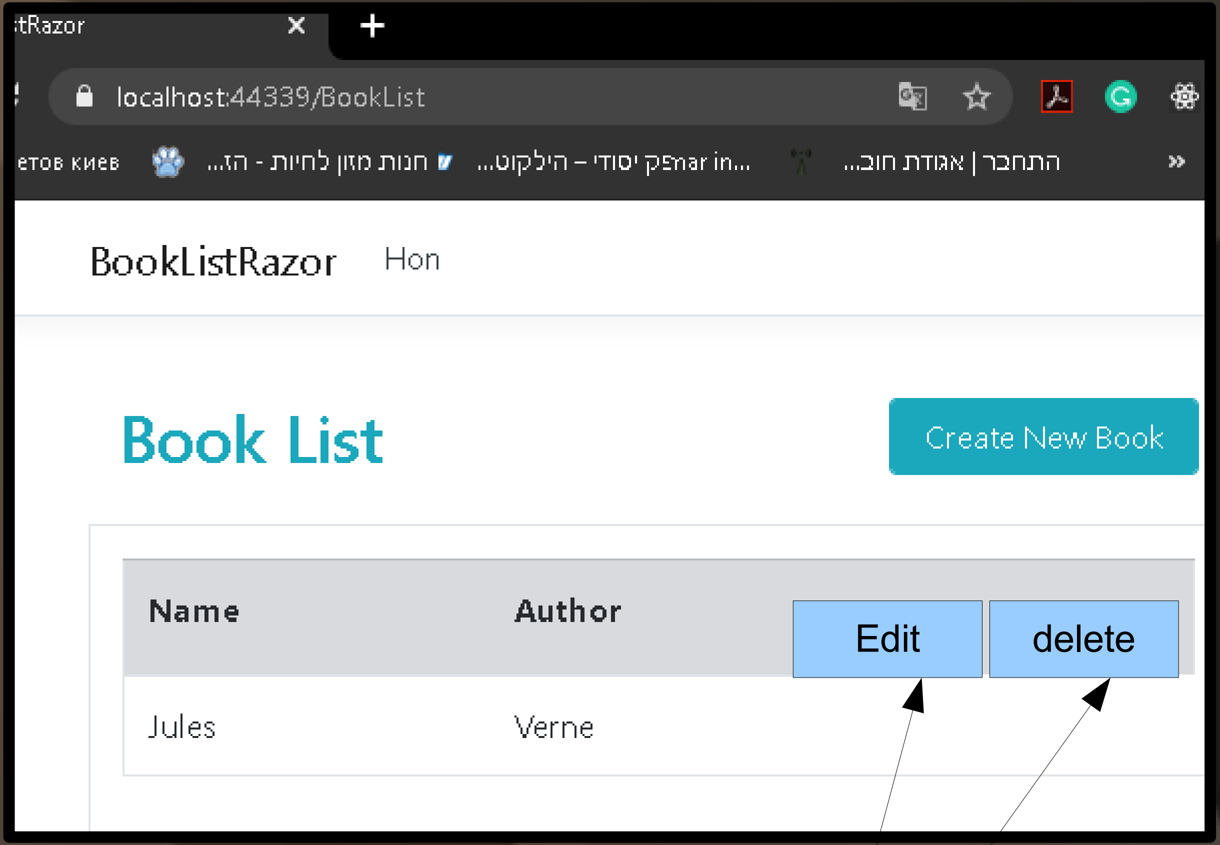
13 .Add another piece of code to **index.cshtml**
So it matches this code.

As you see we've added foreach lloop that runs
Through the books object, and if the book was found
It wraps **item.Name**, and **item.Author** into a <td>
Tags. It will do this for all of the books available in
The database.
This will display a table row for all the items in the
Book.
For all items in the Book object we will display
a table row

14. Save, and run the project F5.
15. If it's alredy running just refresh the webpage.

16. See if a changes took place, and you see a list
Of books.

As you see we've successfully received our fake book!



Name	Author
Jules	Verne

Next step

17. Let's add a Delete, and edit buttons to each book.
Like in this example.

Index.cshtml

```
...

                      @foreach (var item in Model.Books)
                      {
                  <tr>
                      <td>
                          @Html.DisplayFor(m => item.Name)
                      </td>
                      <td>
                          @Html.DisplayFor(m => item.Author)
                      </td>
                          <td>
                             <button class=" btn btn-danger btn-sm">Delete</button>
                             <a class=" btn btn-success btn-sm">Edit</a>
                          </td>
                  </tr>
                      }
                  </table>
              }
               else{
                  <p>No books available</p>
              }
          </form>
      </div>
</div>
```

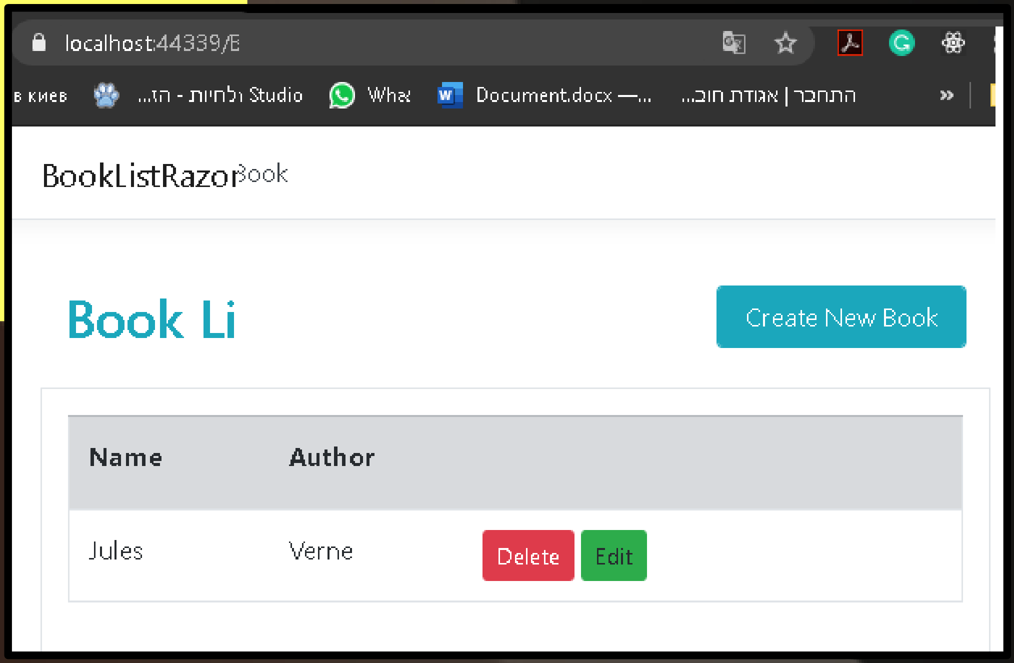
18. Add a Delete button to a book.

19. Add a Edit <a> button to a book

20. Save, And Run the project. F5

21. Inspect the changes.

22. we've successfully created Edit, and Delet buttons.



BookListRazor Book

Book Li

Create New Book

Name	Author	
Jules	Verne	Delete Edit

Index.cshtml

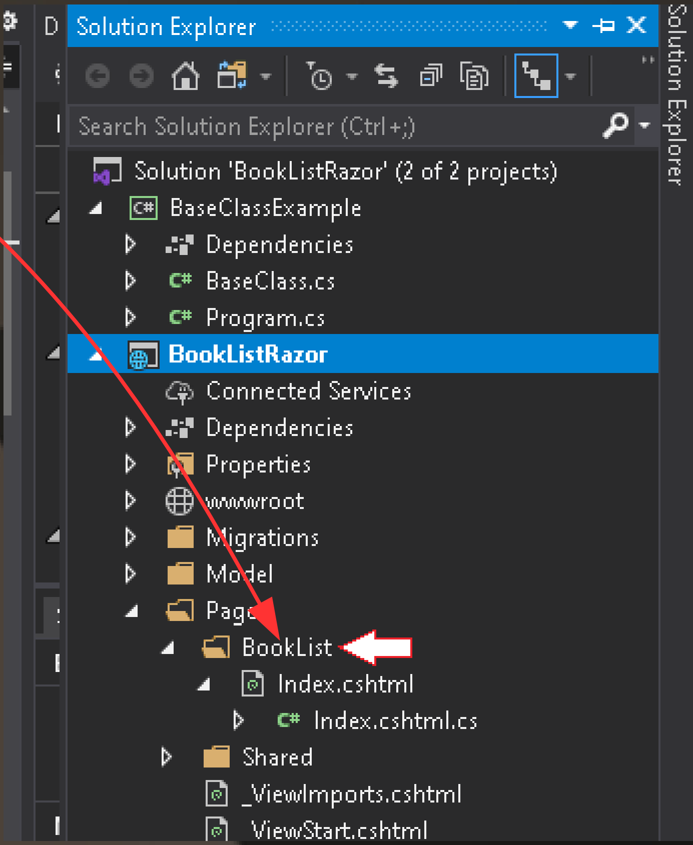
```
@page
@model BookListRazor.Pages.BookList.IndexModel

<br />
<div class="container row p-0 m-0">

    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a asp-page="Create" class=" btn btn-info form-control text-white">Create New Book</a>

    </div>

    <div class="col-12 border p-3 mt-3">
        <form method="post">
            @if (Model.Books.Count() > 0)
            {
                //displaya table
                <table class="table table-striped border">
                    <tr class="table-secondary">
                        <th>
                            <label asp-for="Books.FirstOrDefault().Name"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().Author"></label>
                        </th>
                        <th>

                        </th>
                    </tr>

                    @foreach (var item in Model.Books)
                    {
                        <tr>
                            <td>
                                @Html.DisplayFor(m => item.Name)
                            </td>
                            <td>
                                @Html.DisplayFor(m => item.Author)
                            </td>
                            <td>
                                <button class=" btn btn-danger btn-sm text-white">Delete</button>
                                <a class=" btn btn-success btn-sm text-white">Edit</a>
                            </td>
                        </tr>
                    }
                </table>
            }
            else
            {
                <p>No books available</p>
            }
        </form>
    </div>
</div>
```

23. Create a button link which will create a book. So that once I press it, it redirects me to **Create** razor page.

24. Open Index.cshtml, and scroll to the very top.
25. Add asp-page tag heleper as it appears in the Code

26.Proceed to the next page.

We will need to create a Create razor page to achieve this functionality
This page will live inside **BookList** Folder


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'BookListRazor' (2 of 2 projects)
  BaseClassExample
    Dependencies
    BaseClass.cs
    Program.cs
  BookListRazor
    Connected Services
    Dependencies
    Properties
    wwwroot
    Migrations
    Model
    Pages
      BookList
        Index.cshtml
          Index.cshtml.cs
      Shared
      _ViewImports.cshtml
      _ViewStart.cshtml
```

Let's add Create Razor Page

1. Stop the application if its running
2. In the Solution Explorer Locate **BookList** folder.
3. Right click the folder and select Add/ New Razor Page
4. Select Empty Razor Page.

5. Inside Create.cshtml.cs file add this code:
   This will bint our Create.cshtml to a database

Create.cshtml.cs

```
    public class CreateModel : PageModel
     {

         private readonly ApplicationDbContext _Db;
         public void OnGet()
         {
         }
     }
```

6. Next Implement a construcntor as it appears in the code:

7. Add a model

8 OnGet() method will remain empty.

```
public class CreateModel : PageModel
   {
       //bind the dbcontext
       private readonly ApplicationDbContext _Db;

       public CreateModel(ApplicationDbContext Db)
       {
           _Db = Db;
       }
       //adding a model
       public Book Book { get; set; }
       public void OnGet()
       {
       }
   }
```

9 .We don't need to implement nothing in OnGet() Mehtod. Because this is going to be an empty new book Later inside Create View, you will be able to access this book object, and display labels, and textboxes.

Adding another property to a Book Model

1. Open a Book.cs
2. Add a new property: ISBN

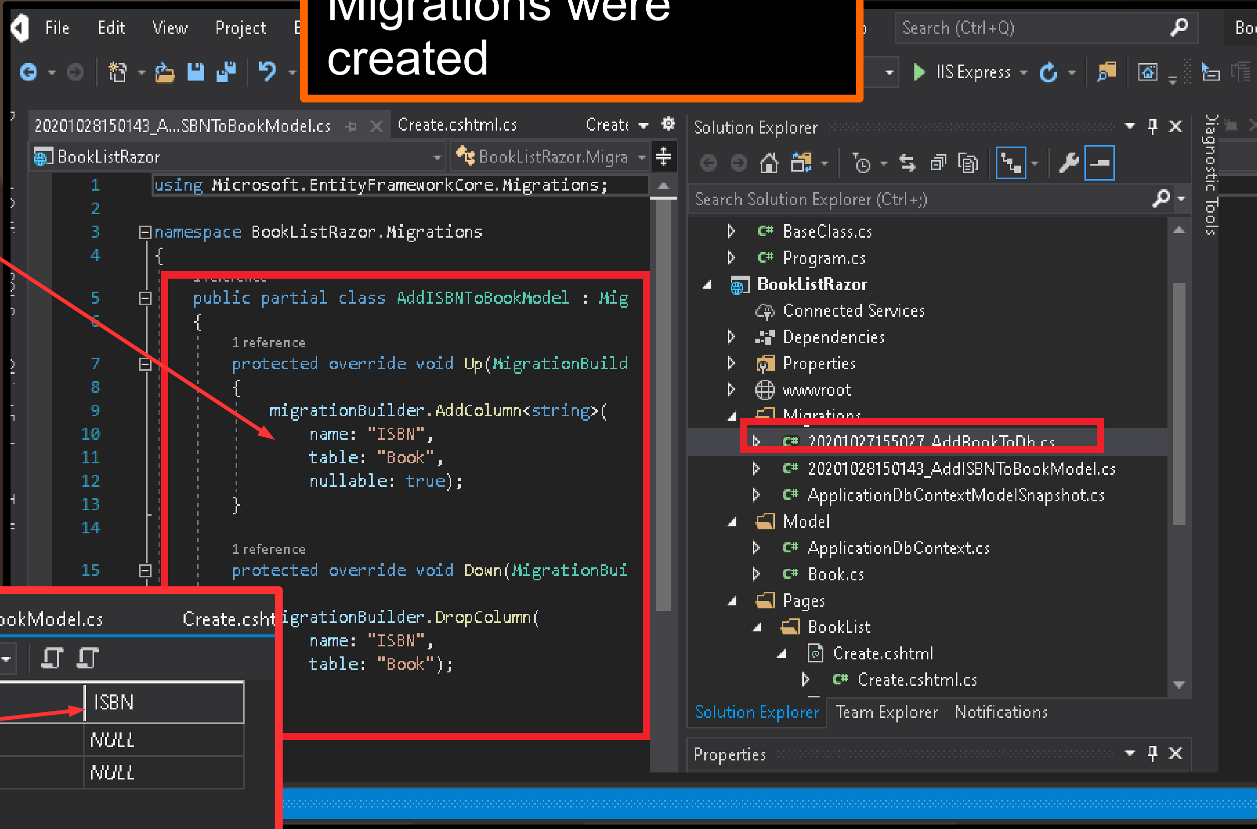
```
public class Book
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public string Name { get; set; }
        public string Author { get; set; }

        public string ISBN { get; set; }
    }
```

Next: update your migration files

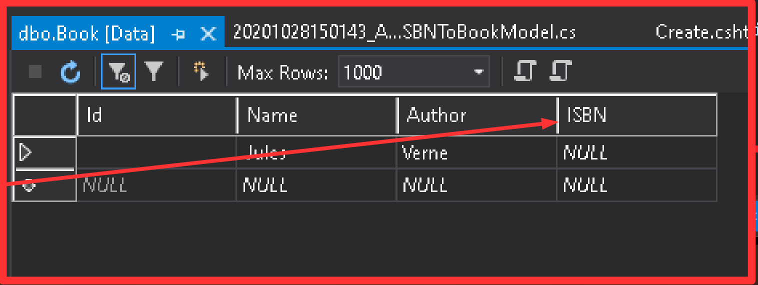
3. Open Package manager Console window, and Run this Command:
 add-migration AddISBNToBookModel

5. Next update the database by entering following command: **update-database**

Migrations were created



```
using Microsoft.EntityFrameworkCore.Migrations;

namespace BookListRazor.Migrations
{
    public partial class AddISBNToBookModel : Mig
    {
        1 reference
        protected override void Up(MigrationBuild
        {
            migrationBuilder.AddColumn<string>(
                name: "ISBN",
                table: "Book",
                nullable: true);
        }

        1 reference
        protected override void Down(MigrationBui
        {
            igrationBuilder.DropColumn(
                name: "ISBN",
                table: "Book");
```



Id	Name	Author	ISBN
	Jules	Verne	NULL
NULL	NULL	NULL	NULL

We've just added ISBN column To our table

```
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">
    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a asp-page="Create" class=" btn btn-info form-control text-white">Create New Book</a>
    </div>
    <div class="col-12 border p-3 mt-3">
        <form method="post">
            @if (Model.Books.Count() > 0)
            {
                //display table
                <table class="table table-striped border">
                    <tr class="table-secondary">
                        <th>
                            <label asp-for="Books.FirstOrDefault().Name"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().Author"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().ISBN"></label>
                        </th>
                        <th>

                        </th>
                    </tr>

                    @foreach (var item in Model.Books)
                    {
                <tr>
                    <td>
                        @Html.DisplayFor(m => item.Name)
                    </td>
                    <td>
                        @Html.DisplayFor(m => item.Author)
                    </td>
                    <td>
                         @Html.DisplayFor(m => item.ISBN)
                    </td>
                    <td>
                        <button class=" btn btn-danger btn-sm text-white">Delete</button>
                        <a class=" btn btn-success btn-sm text-white">Edit</a>
                    </td>
                </tr>
                    }
                </table>
            }
            else
            {
                <p>No books available</p>
            }
        </form>
    </div>
</div>
```

Next step

Adding a missing labels to Index.cshtml

We willbe adding ISBN Label to a column of a table, and to an Item.ISBN object

1.Open Pages/BookList/index.cshtml, and modify your code to match this code:

In the Next step we will work on Create page.

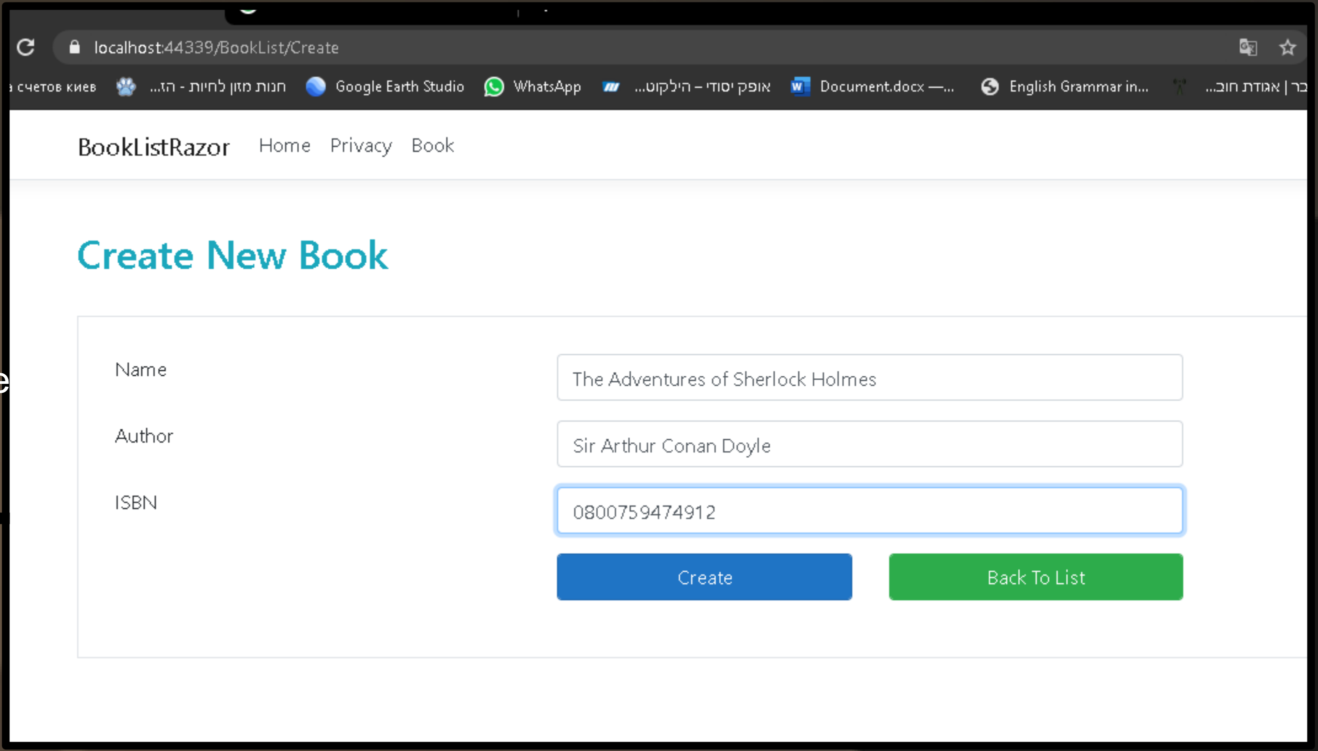
Create Book Page UI

1. Open Pages/BookList/Create.cshtml file
2. Paste the below code so it mathces yours:
3.Run the application and check the Create

```
@page
@model BookListRazor.Pages.BookList.CreateModel
@{
}

<br />
<h2 class="text-info">Create New Book</h2><br />
<div class="border container" style="padding:30px;">
    <!---We will be posting data back to a Page Handler-->
    <form method="post">

        <div class="form-group row">
            <div class=" col-4">
                <label asp-for="Book.Name"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Name" class="form-control" />
            </div>
        </div>
        <div class="form-group row">
            <div class=" col-4">
                <label asp-for="Book.Author"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Author" class="form-control" />
            </div>
        </div>
        <div class="form-group row">
            <div class=" col-4">
                <label asp-for="Book.ISBN"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.ISBN" class="form-control" />
            </div>
        </div>

        <div class="form-group row">
            <div class="col-3 offset-4">
                <input type="submit"  value="Create" class="btn btn-primary form-control" />
            </div>
            <div class="col-3">
                <a asp-page="Index" class="btn btn-success form-control">Back To List</a>
            </div>

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

You should see the following output

Next step Is to add a logic to
Submit (Create) button.
If we press a Create button nothing
Happen. It is because we do not have
A post handler inside our **Create**.cs mode


localhost:44339/BookList/Create

BookListRazor Home Privacy Book

Create New Book

Name: The Adventures of Sherlock Holmes
Author: Sir Arthur Conan Doyle
ISBN: 0800759474912

Let's see how can we add a post handler
How we get the data, and save it to a
Database.

Create Book and Validations

We must be sure when we adding a book information and pressing submit button
We will get back to a book list. But before that we need to implement some basic validation.
Stop the application.

1. Open Pages/BookList/Crate.cshtml.cs
2. Create a Post handler. Paste the following
   Code right after **OnGet()** handler method

```
public async Task<IActionResult> OnPost()
       {
           if (ModelState.IsValid)
           {
               //if modelstate is valid add a book to a queue object
               await _Db.Book.AddAsync(Book);
               //now save changes to a database From the queue object
               await _Db.SaveChangesAsync();
               //Changes are now saved to a database.

               //once a data is pushed to a database
               //redirect to the Pages/BookList/Index.cshtml page to see the list of books
               return RedirectToPage("Index");
           }
           else
           {
               return Page();
           }
       }
```

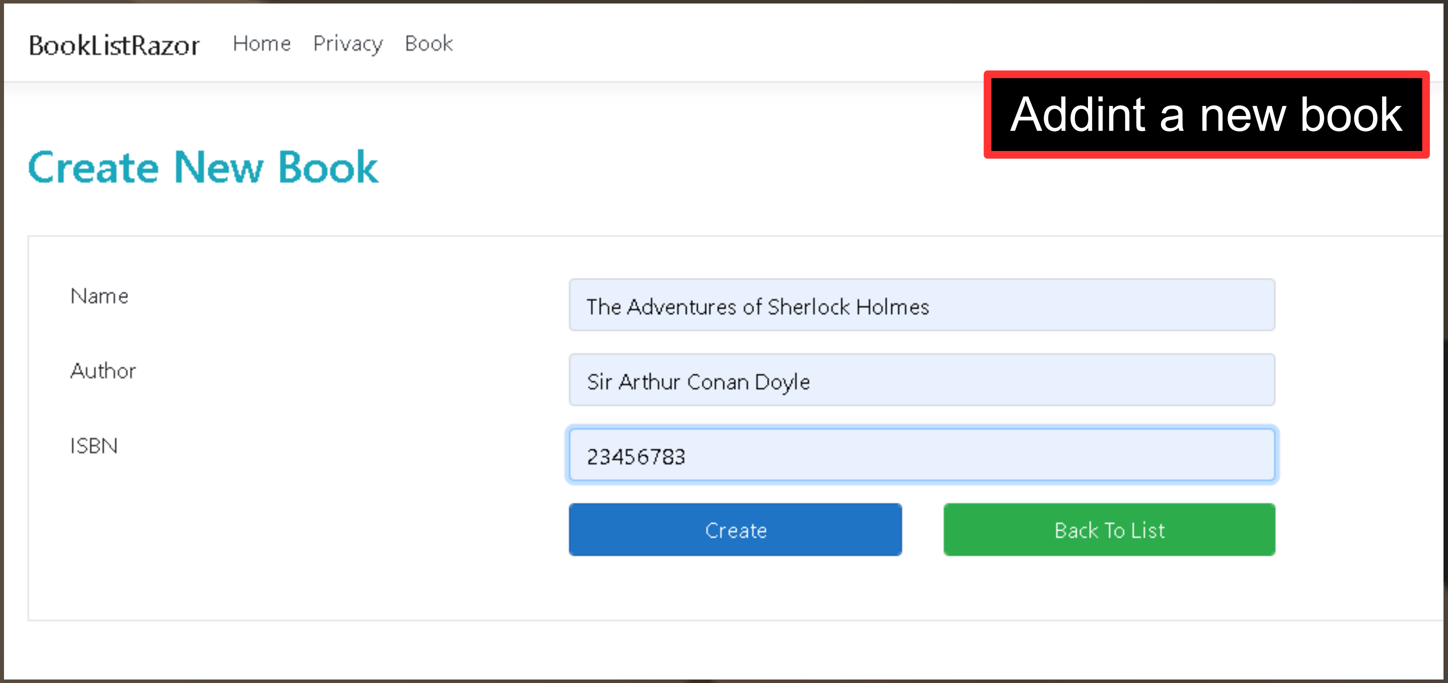
See the full source code of **Create.cshtml**.cs
On the next page.

Create.cshtml.cs Complete page

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BookListRazor.Pages.BookList
{
    public class CreateModel : PageModel
    {
        //bind the dbcontext
        private readonly ApplicationDbContext _Db;

        public CreateModel(ApplicationDbContext Db)
        {
            _Db = Db;
        }

        //adding a model
        [BindProperty]
        public Book Book { get; set; }
        public void OnGet()
        {
        }

        public async Task<IActionResult> OnPost()
        {
            if (ModelState.IsValid)
            {
                //if modelstate is valid add a book to a queue object
                await _Db.Book.AddAsync(Book);
                //now save changes to a database From the queue object
                await _Db.SaveChangesAsync();
                //Changes are now saved to a database.

                //once a data is pushed to a database
                //redirect to the Pages/BookList/Index.cshtml page to see the list of books
                return RedirectToPage("Index");
            }
            else
            {
                return Page();
            }
        }
    }
}
```

3. Try running the application and add A new book.

Here is the following output

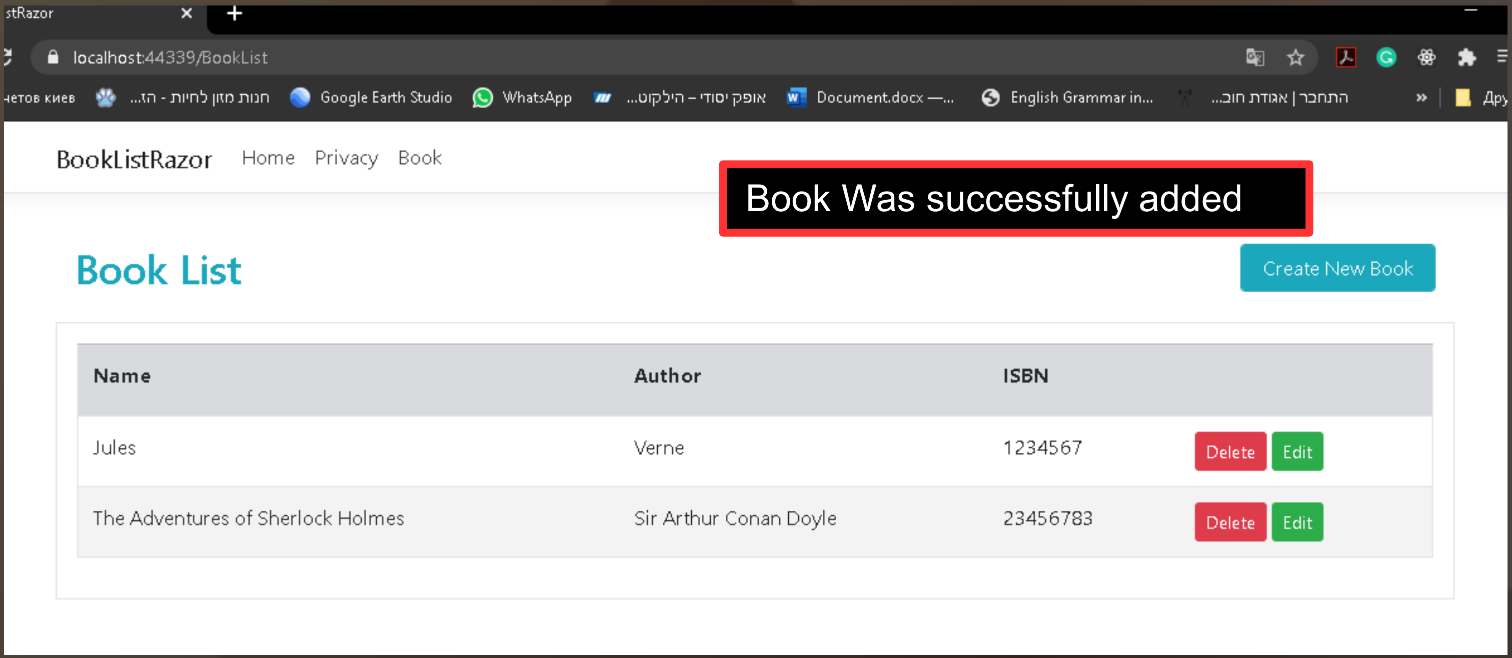


Addint a new book

BookListRazor Home Privacy Book

Create New Book

Name	The Adventures of Sherlock Holmes
Author	Sir Arthur Conan Doyle
ISBN	23456783



Book Was successfully added

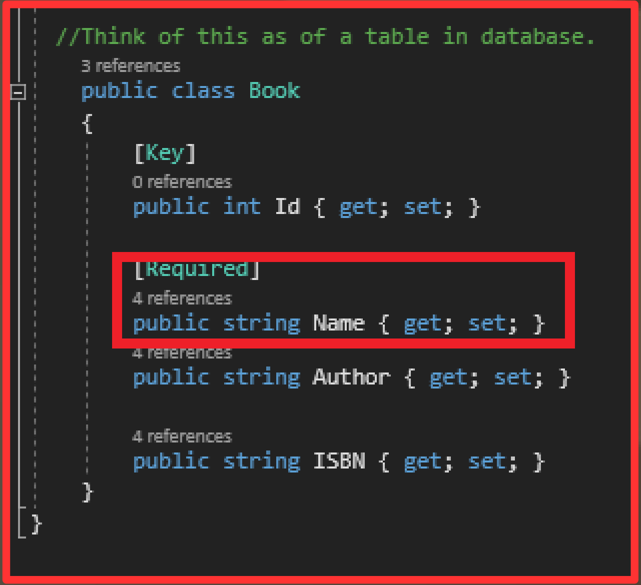
localhost:44339/BookList

BookListRazor Home Privacy Book

Book List

Name	Author	ISBN	
Jules	Verne	1234567	Delete Edit
The Adventures of Sherlock Holmes	Sir Arthur Conan Doyle	23456783	Delete Edit

You can try to create a book with an empty name, but it won't let you do this
Because the name property assigned as a [Required] property inside the Book model.
However, we still need to provide the user with useful information on validation errors



```
//Think of this as of a table in database.
    3 references
    public class Book
    {
        [Key]
        0 references
        public int Id { get; set; }

        [Required]
        4 references
        public string Name { get; set; }
        4 references
        public string Author { get; set; }

        4 references
        public string ISBN { get; set; }
    }
}
```

```
@page
@model BookListRazor.Pages.BookList.CreateModel
@{
}

<br />
<h2 class="text-info">Create New Book</h2>
<br />
<div class="border container" style="padding:30px;">
    <!---We will be posting data back to a Page Handler-->
    <form method="post">
        <!---ValidationInfo div-->
        <div class="text-danger" asp-validation-summary="ModelOnly">

        </div>
        <!--End validation div-->
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Name"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Name" class="form-control" />
            </div>
            <span asp-validation-for="Book.Name" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Author"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Author" class="form-control" />
            </div>
            <span asp-validation-for="Book.Author" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.ISBN"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.ISBN" class="form-control" />
            </div>
            <span asp-validation-for="Book.ISBN" class="text-danger"></span>

        </div>

        <div class="form-group row">
            <div class="col-3 offset-4"><!--By pressing Submit we will invoke Create.cs-->
                <input type="submit" value="Create" class="btn btn-primary form-control" />
            </div>
            <div class="col-3">
                <a asp-page="Index" class="btn btn-success form-control">Back To List</a>
            </div>

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

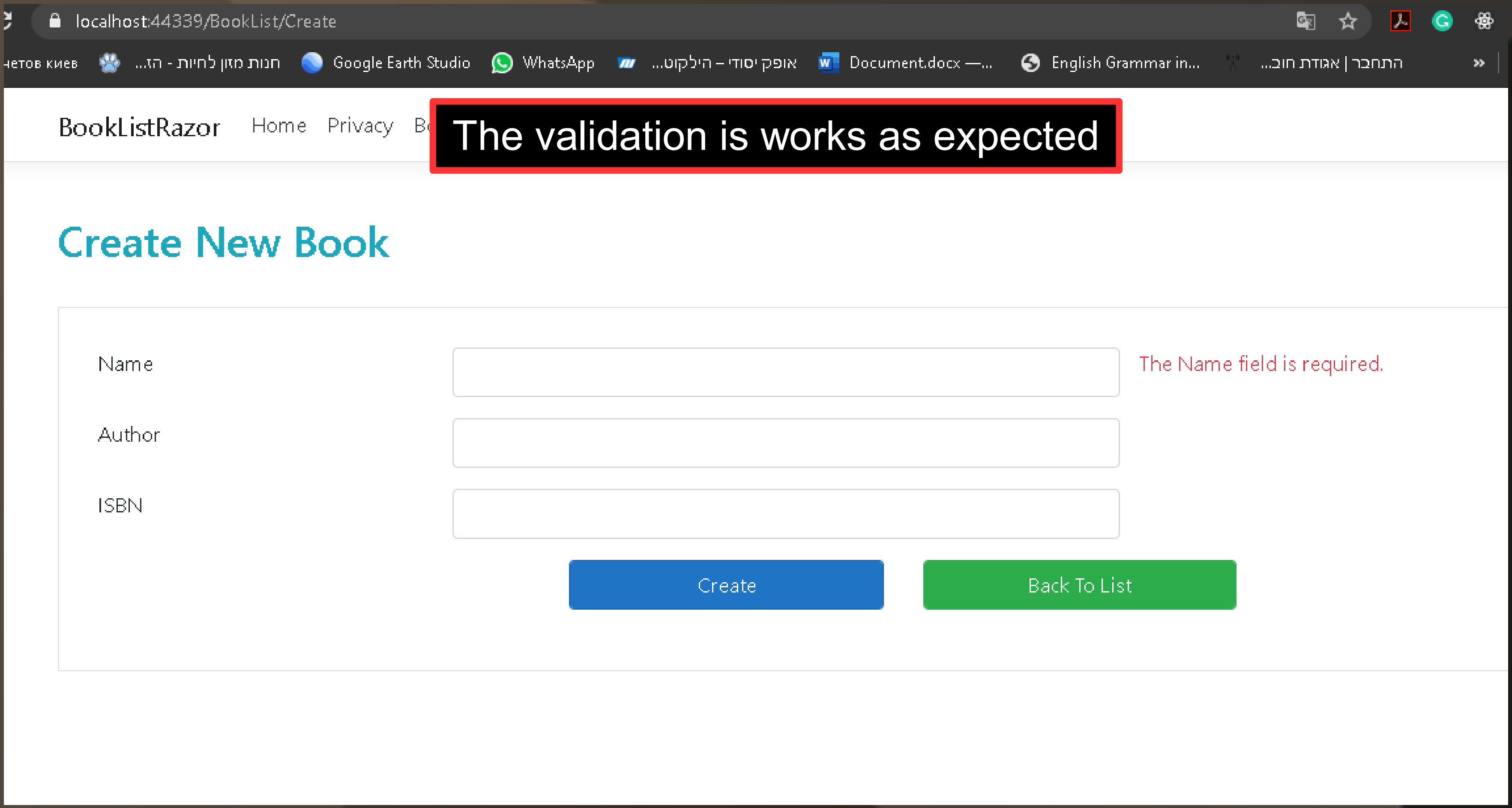
Provide user With validation Info

1. Open Pages/BookList/Create.cshtml.cs file
2. Add the folowing code inside <Form> tag

3.Add a Validation-summarry tag helper
4. Add asp-validation-for span to each section

5. Once fonished, Run application F5 and see the changes.

6. Proceed to the next page



The validation is works as expected

Create New Book

Name		The Name field is required.
Author		
ISBN		

Client side validations

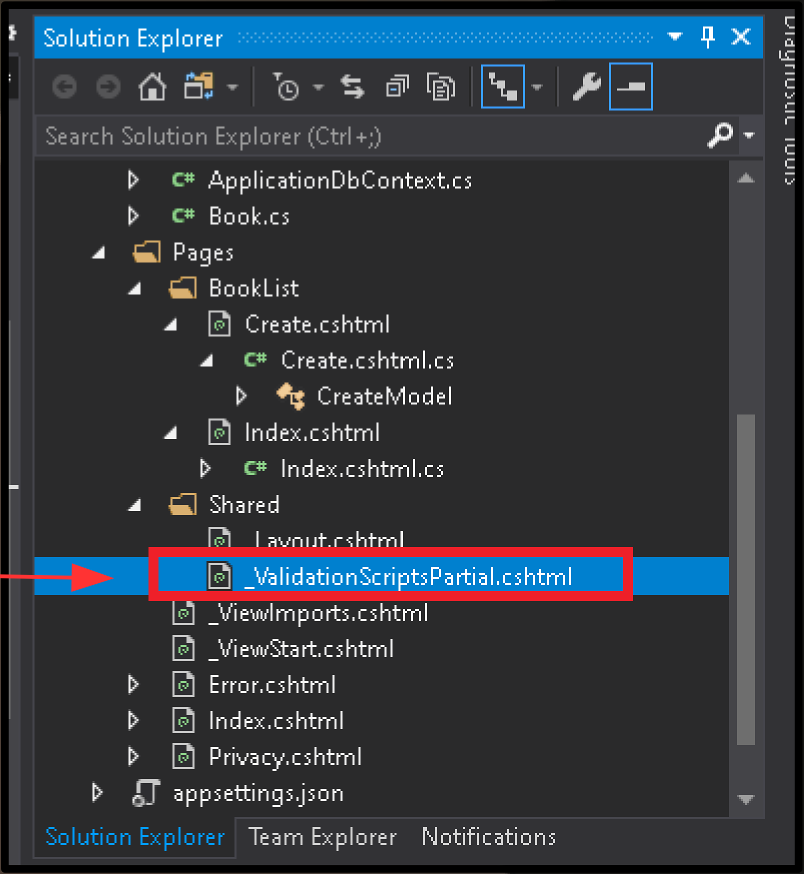
We want to add a validation on the client side and validate textboxes just before form gets posted.
We want to post a form ONLY after validation iss passed. For this wi will use a special_ValidationScriptsPartial file, locatyed in Shared folder.
For that we will create a reference to this file inside Create.cshtml file.

1. Open Pages/BookList/Create.cshtml file.
2. Add the following code to the very bottom of the Create.cshtml.

Create.cshtml

```
@section    scripts{
    <partial name="_ValidationScriptsPartial" />
}
```

Create a Reference to   _ValidationScriptsPartial,cshtml

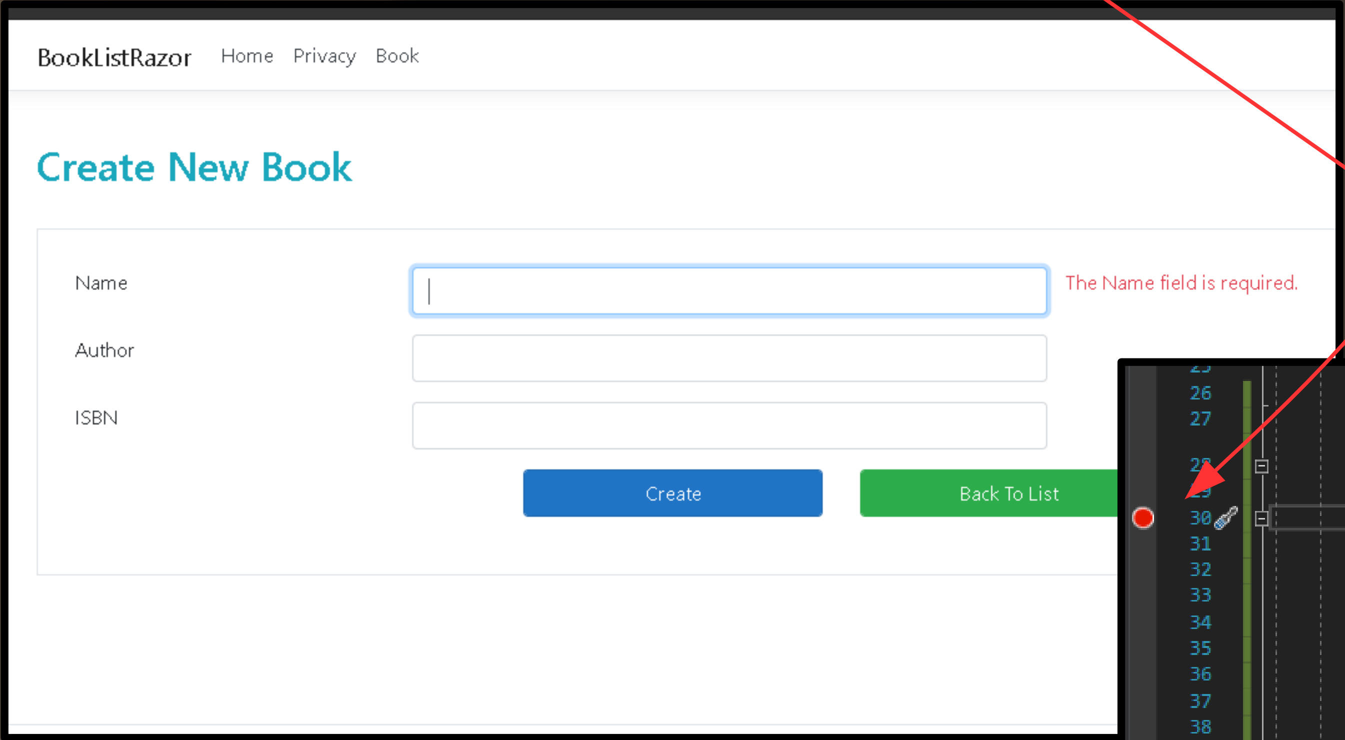


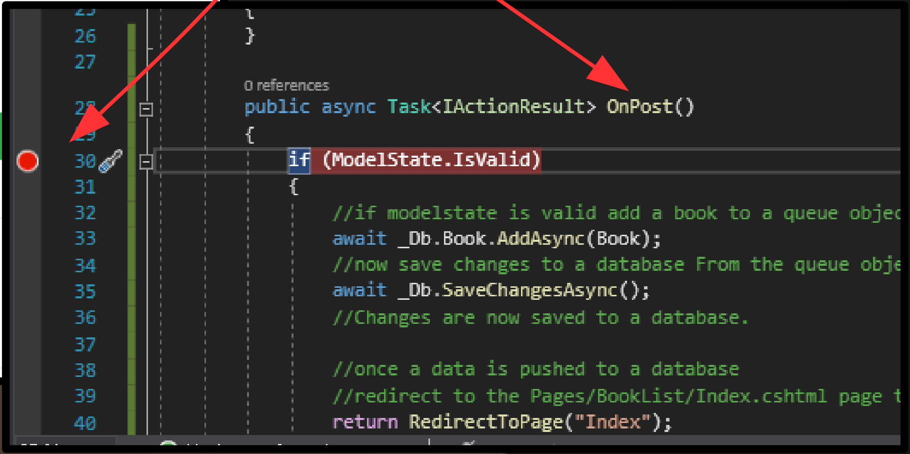
3. Save project, and run the application.
4. The validation should now happen before a postback.
5. Set a breakpoint inside Create.cshtml.cs, on the line that says:

`Breackpoint here` **if (ModelState.IsValid)**

6. Try creating an empty book now.

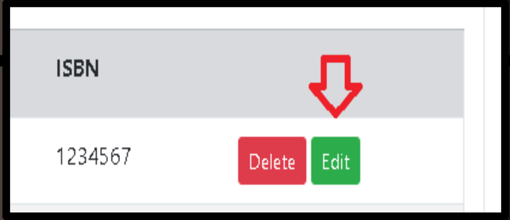
7. As you see debugger is not hitting the break point now.
It is because the validation happened before a postback





```
}

0 references
public async Task<IActionResult> OnPost()
{
    if (ModelState.IsValid)
    {
        //if modelstate is valid add a book to a queue objec
        await _Db.Book.AddAsync(Book);
        //now save changes to a database From the queue obje
        await _Db.SaveChangesAsync();
        //Changes are now saved to a database.

        //once a data is pushed to a database
        //redirect to the Pages/BookList/Index.cshtml page t
        return RedirectToPage("Index");
```

We ended with double validation. Cliend side, and Server side.



Edit book Get Handler

1. We will need to pass the routing wen pressing Edit button.
whenever a user clicks the edit button we want to pass the ID of field that we are editing.
For this we are using **asp-route-id=""** tag helper.
2. Alter the code inside **Index.cshtml** file. Locate the edit anchor tag, and edit it so it matches this code:

Index.cshtml

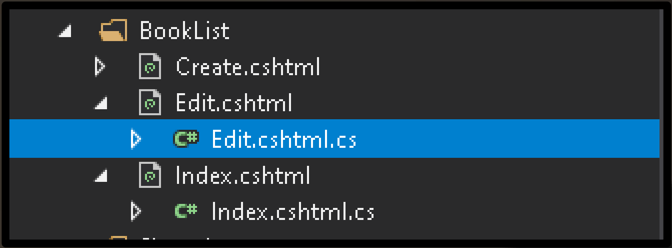
```
<td>
    <button class=" btn btn-danger btn-sm text-white">Delete</button>
    <a asp-page="Edit" asp-route-id="@item.Id" class=" btn btn-success btn-sm text-white">Edit</a>
</td>
```

Your Pages/BookList/ Index.cshtml should look like this

Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">
    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a asp-page="Create" class=" btn btn-info form-control text-white">Create New Book</a>
    </div>
    <div class="col-12 border p-3 mt-3">
        <form method="post">
            @if (Model.Books.Count() > 0)
            {
                //display table
                <table class="table table-striped border">
                    <tr class="table-secondary">
                        <th>
                            <label asp-for="Books.FirstOrDefault().Name"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().Author"></label>
                        </th>
                        <th>
                            <label asp-for="Books.FirstOrDefault().ISBN"></label>
                        </th>
                        <th>
                        </th>
                    </tr>
                    @foreach (var item in Model.Books)
                    {
                <tr>
                    <td>
                        @Html.DisplayFor(m => item.Name)
                    </td>
                    <td>
                        @Html.DisplayFor(m => item.Author)
                    </td>
                    <td>
                        @Html.DisplayFor(m => item.ISBN)
                    </td>
                    <td>
                        <button class=" btn btn-danger btn-sm text-white">Delete</button>
                        <!--Passing the routing-->
                        <!--whenever a user clicks the edit button we want to pass the  ID of field that we are editing-->
                        <!--For this we are using asp-route-id=""-->
              →         <a asp-page="Edit" asp-route-id="@item.Id" class=" btn btn-success btn-sm text-white">Edit</a>
                    </td>
                </tr>
                    }
                </table>
            }
            else
            {
                <p>No books available</p>
            }
        </form>
    </div>
</div>
```

2. Create a new Razor page inside Pages/BookList folder. Name it as Edit.cshtml


BookList
Create.cshtml
Edit.cshtml
Edit.cshtml.cs
Index.cshtml
Index.cshtml.cs

We will pass a parameter id to OnGet() handler.
We will Edit the book based on this integer.
But First We will bind the Edit model to a data base as in previous examples.

3. Open Edit.cshtml.cs file and edit as follows:

Edit.cshtml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BookListRazor.Pages.BookList
{
    public class EditModel : PageModel
    {

        private ApplicationDbContext _DB;
        public EditModel(ApplicationDbContext DB)
        {
            _DB = DB;
        }

        [BindProperty]
        public Book Book { get; set; }

        public async Task OnGet(int id)
        {
            Book = await _DB.FindAsync(id);
        }
    }
}
```

Proceed to the next page

Create Edit Razor page

Edit.cshtml.cs

```
@page
@model BookListRazor.Pages.BookList.EditModel
@{
}
<br />
<h2 class="text-info">Edit Book</h2>
<br />
<div class="border container" style="padding:30px;">
    <!---We will be posting data back to a Page Handler-->
    <form method="post">
        <!---ValidationInfo div-->
        <div class="text-danger" asp-validation-summary="ModelOnly">
        </div>
        <!--End validation div-->
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Name"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Name" class="form-control" />
            </div>
            <span asp-validation-for="Book.Name" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Author"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Author" class="form-control" />
            </div>
            <span asp-validation-for="Book.Author" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.ISBN"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.ISBN" class="form-control" />
            </div>
            <span asp-validation-for="Book.ISBN" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class="col-3 offset-4">
                <!--By pressing Submit we will invoke Update.cshtml.cs-->
                <input type="submit" value="Update" class="btn btn-primary form-control" />
            </div>
            <div class="col-3">
                <a asp-page="Index" class="btn btn-success form-control">Back To List</a>
            </div>
        </div>
    </form>
</div>
@section scripts{
    <partial name="_ValidationScriptsPartial" />
}
```

id=3

id=3

id=3

The Edit page UI will be similar to Create page but only difference will be, is the edit page book's data will be loaded based on previously pressed book=ID. For example: id=3

1. Open Create.cshtml file.
2. Copy all of its contents besides **@page** and **@Model**.

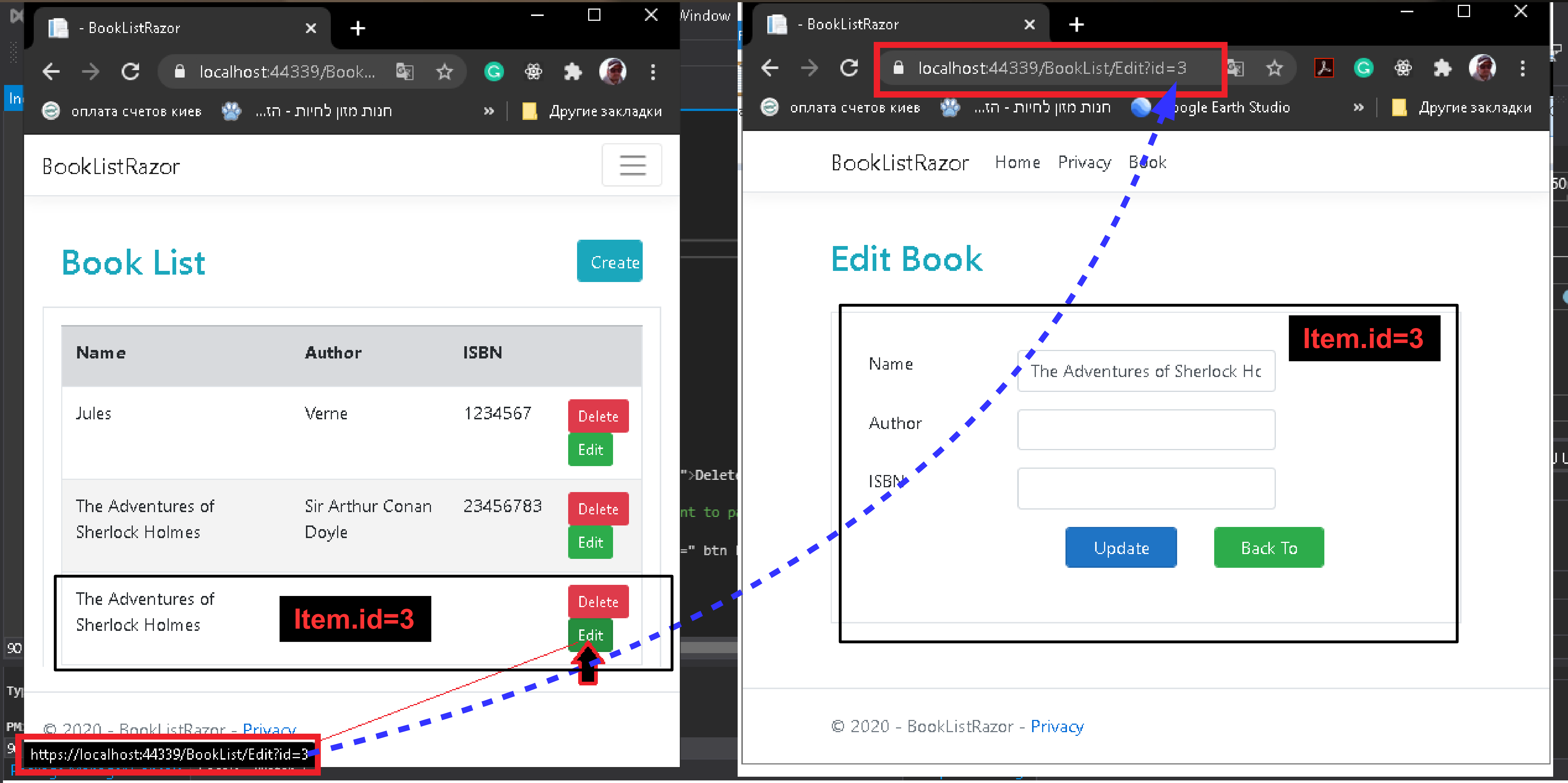
3. Open Edit.cshtml file, and paste the code as it appears in this code.

Arrows represents the main difference between Create, and Edit pages.

When you redirected to the Edit page by pressing edit button in the index page, the following texboxes will be filled with the corresponding information, based on previously clicked book= id
Book.Name
Book.Author
Book.ISBN items
These properties will be populated based on Previous page' item.id
Remember this?
`asp-route-id="@item.Id"`

For example:
While being on the Index page we decided to edit the third book (id=3). After pressing the edit button we should be redirected to the edit page, and all of its text boxes will
populated with the information based on
a previously clicked book's Edit button with the id of =3
We have achieved this with the lep of asp-route-id tag helper

4. Save, and run the application.
5. Try to edit a book. Pay attention to how a clicked edit button passes all the information to the Edit Page.



Book List

Name	Author	ISBN
Jules	Verne	1234567
The Adventures of Sherlock Holmes	Sir Arthur Conan Doyle	23456783
The Adventures of Sherlock Holmes		

Edit Book

Name	The Adventures of Sherlock Hc
Author	
ISBN	

As you see this is a very powerfull helper tag, which helps us to route the Item.id to any page we want

```
<a asp-page="Edit" asp-route-id="@item.Id" class=" btn btn-success btn-sm text-white">Edit</a>
```

If you click the Update Button nothing happens, it is because we have not created a post handler for this yet
Inside **Edit.cshtml.cs** file

1. Open Edit.cshtml.cs file, and create a post handler like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BookListRazor.Pages.BookList
{
    public class EditModel : PageModel
    {

        private ApplicationDbContext _DB;
        public EditModel(ApplicationDbContext DB)
        {
            _DB = DB;
        }

        [BindProperty]
        public Book Book { get; set; }

        public async Task OnGet(int id)
        {
            Book = await _DB.Book.FindAsync(id);
        }


        //We will be redirecting
        public async Task<IActionResult> OnPost()
        {
            if (ModelState.IsValid)
            {
                //Book == current page book
                //BookFromDb == book in the database.
                //if valid Assign a new values to a book from a database.
                //retrieve the book from a database based on the current book.id
                var BookFromDb = await _DB.Book.FindAsync(Book.Id);
                //assign a new values to the book
                BookFromDb.Name = Book.Name;
                BookFromDb.Author = Book.Author;
                BookFromDb.ISBN = Book.ISBN;

                await _DB.SaveChangesAsync();

                //After pushing to a database Redirect to index.cshtml
               return RedirectToPage("Index");

            }
            else
            {
               return  RedirectToPage();
            }
        }
    }
}
```

2. proceed to the next page

Edit.cshtml

```
@page
@model BookListRazor.Pages.BookList.EditModel
@{
}
<br />
<h2 class="text-info">Edit Book</h2>
<br />
<div class="border container" style="padding:30px;">
    <!---We will be posting data back to a Page Handler-->
    <form method="post">
        <input type="hidden" asp-for="Book.Id" />
        <!---ValidationInfo div-->
        <div class="text-danger" asp-validation-summary="ModelOnly">
        </div>
        <!--End validation div-->
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Name"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Name" class="form-control" />
            </div>
            <span asp-validation-for="Book.Name" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.Author"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.Author" class="form-control" />
            </div>
            <span asp-validation-for="Book.Author" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class=" col-3">
                <label asp-for="Book.ISBN"></label>
            </div>
            <div class="col-6">
                <!--TextBox-->
                <input asp-for="Book.ISBN" class="form-control" />
            </div>
            <span asp-validation-for="Book.ISBN" class="text-danger"></span>
        </div>
        <div class="form-group row">
            <div class="col-3 offset-4">
                <!--By pressing Submit we will invoke Update.cshtml.cs-->
                <input type="submit" value="Update" class="btn btn-primary form-control" />
            </div>
            <div class="col-3">
                <a asp-page="Index" class="btn btn-success form-control">Back To List</a>
            </div>
        </div>
    </form>
</div>

@section scripts{
    <partial name="_ValidationScriptsPartial" />
}
```

Name

Author

ISBN

3. Important note.
 We have to add another hidden property to Edit.cshtml file to make it work correctly.

4. Open Edit.cshtml file and add a hidden field right after a<form> tag as it appears in this code:

This will update the book, because it will find an ID.

Important!
Allways make sure that inside **Edit pages** in such Text fields you have the Id, or any other properties you might need for updateing. If you have no Id Text box inside you page, it must be presented in The **hidden** property (again if you don't have this inside A text box)

As you see there is no ID text box linked to ID
Because of the we use a hidden filed to hold this value

Next step Implement the Delet button

With the delete button we can use the same texhnique as in Create, and Edit
We can create a Delete Page, were we can display some details befor deleting the book.
And we can bind a post event to a delete button.
But I decided to do somethong new:
When a user clicks a delete button i'd like to shopw an Alert message with Okay button.
Once ther okay button is clicked The book will be removed, and deleted from the index page.
For that let's first implement the pop-up window first

1. OpenPages/BookList/ Index.cshtml
2. Locate the Delete button and add the following **onclick=““** method

```
      <td>
        <button asp-page-handler="Delete" asp-route-id="@item.Id" onclick="return confirm('Are you Shure you want to delete 
this book?')" class=" btn btn-danger btn-sm text-white">Delete</button>
       <a asp-page="Edit" asp-route-id="@item.Id" class=" btn btn-success btn-sm text-white">Edit</a>
    </td>
```

We've added onclick method. That will return a Confirm box.
If it return **true** we will go to a page handler on the same index page (we will have to create this handler-method).
Asp-page-handler=“Delete“.

When Deleting we again have to pass the ID of the book that we want to delete. For that we use
asp-rote-id=“@item-id:

Next step:
Implement Delete handler method inside index.cshtml page
3. Proceed to the next page.

Index.cshtml

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

namespace BookListRazor.Pages.BookList
{
    public class IndexModel : PageModel
    {

        private readonly ApplicationDbContext _Db;


        //this way we inject ApplicationDbContext inside this page (another words: this is how we connect this page with a database object)
        //Because ApplicationDbContext is inside a Dependency injection container.
        public IndexModel(ApplicationDbContext db)
        {
            _Db = db;
        }

        public IEnumerable<Book> Books { get; set; }

        public async Task OnGet()
        {
            Books = await _Db.Book.ToListAsync();
        }

        //We use IActionResult because we redirecting to the same page
        //We Use the OnPost and then adding the Handler
        //we get OnPostDelete
        public async Task<IActionResult> OnPostDelete(int id)
        {
           var book = await _Db.Book.FindAsync(id);

            if (book== null)
            {
                //if nothing found display message
                return NotFound();
            }
            else
            {
                //if yes, a clicked book(id) will equal to the database book id
                //and the database' book with same id will be removed.
                _Db.Book.Remove(book);
               await  _Db.SaveChangesAsync();

                //once done let's return to the index page

                return RedirectToPage("Index");
            }
        }

    }
}
```

4. Open **Index.cshtml.cs** model
5. Add the following code to your file.
6. Save , and run ther application F5

You should be able to Delete the books now.
Congratulations we've completed our CRUD Operations.

Up until this moment we have being using a basic functionalities of HTML and Asp.Net Core.
We recieved the List of books using A Basic html+asp.net core.
Next step is to add some fancy functionality to our applicfation so we could display a book of list „In a fancy way". We will create **custom local API**. This API will retrieve the list of books from a database In a Json format. Then we will add a few CDN packges which is a set of Javascripts, and CSS
You can reference the package, or install it locally. We will be puting these references inside **_layout.cshtml** file.

From this very moment pay a close attention to your code. Double check every code you write.
The next section is very sensitive to a code errors. You could face unexpected output while running the Application. This is why it is very-very important to focus on every little detail in the next steps.

We will be adding 3 „packages"

1. Sweetalert https://sweetalert2.github.io/ to get nice alerts
2. Toasters https://codeseven.github.io/toastr/demo.html for fancy notifications
3. Datatables https://datatables.net/ for creating fancy tables

---

2. You will need to copy-paste these CDN sylesheets into your **<head>** tag of Pages/Shared/ _Layout.cshtml file

CSS:
```
<link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/sweetalert/1.1.3/sweetalert.min.css" />
```

JS: Paste this code After </footer> section in Pages/Shared/ **_Layout.cshtml file**

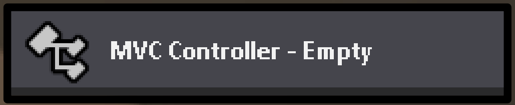
```
<script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js"></script>
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
```

_Layout.cshtml file

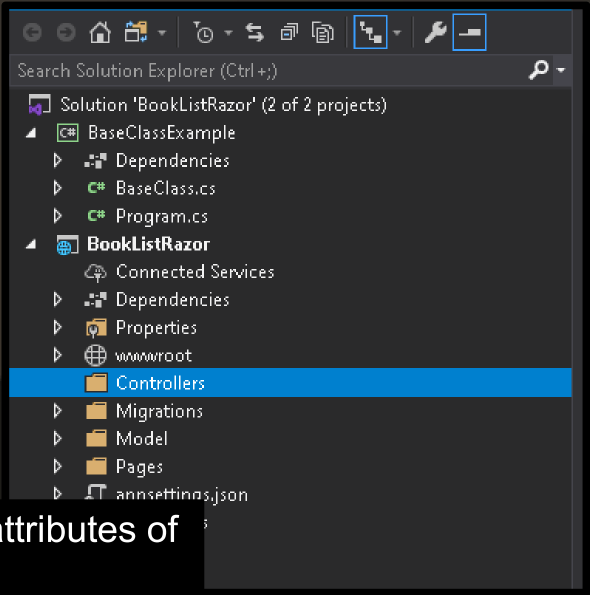
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - BookListRazor</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <!--CSS Brhrugen-->
    <link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/sweetalert/1.1.3/sweetalert.min.css" />

</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-page="/Index">BookListRazor</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/BookList/Index">Book</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2020 - BookListRazor - <a asp-area="" asp-page="/Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    <!--Bhrugen JS-->
    <script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js"></script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>

    @RenderSection("Scripts", required: false)
</body>
```

For using a datatables we will have to make API Calls, to retrieve the books in JSON format.
To anable API Calls we have to add API Controller to our solution.

1.Right click BookListRazor and select: **Add a new folder** .Name it **Controllers**
2.Right click Controllers folder and select **Add New Controller**
3. Select MVC Controller(Empty) from the list.


MVC Controller - Empty

4.Give it a neme of **BookController**


Search Solution Explorer (Ctrl+;)
Solution 'BookListRazor' (2 of 2 projects)
BaseClassExample
Dependencies
BaseClass.cs
Program.cs
BookListRazor
Connected Services
Dependencies
Properties
wwwroot
Controllers
Migrations
Model
Pages
appsettings.json

5. Open newly created BookController, and create a new **_Db** context object as we did Previously in this course.
Here is the code:

BookController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;

namespace BookListRazor.Controllers
{
    public class BookController : Controller
    {
        private readonly ApplicationDbContext _Db;
        public BookController(ApplicationDbContext db)
        {
            _Db = db;
        }
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Step-1

Next we will need to implement attributes of
1. [HttpGet]
2. [HttpDelete]

6. Rename the IActionResult Index() method to :“
public async Task<IActionResult> GetAll()
7. Add **[HttpGet]** attribute above <IActionResult> GetAll() method.
8. return Json object

BookController.cs

Step-2

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;

namespace BookListRazor.Controllers
{
    public class BookController : Controller
    {
        private readonly ApplicationDbContext _Db;
        public BookController(ApplicationDbContext db)
        {
            _Db = db;
        }
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
         return Json(new { data = await _Db.Book.ToListAsync() });
        }
    }
}
```

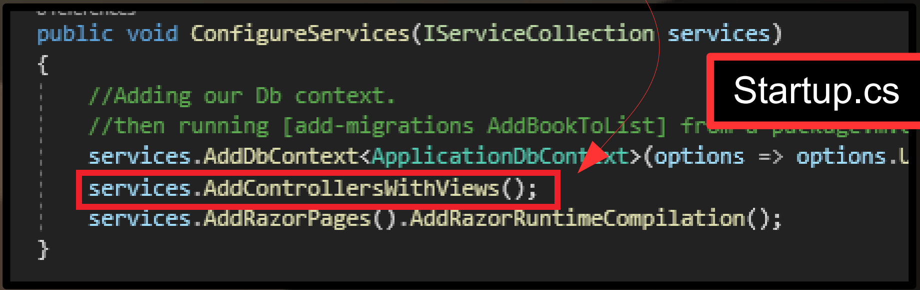
9.Proceed to the next page...

10. Open **Startup.cs** file. We will add a new **service** to support **API calls**
12. Add the following line of code inside **ConfigureServices() method**

```
services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
      services.AddControllersWithViews(); /(adding our API Controller)
      services.AddRazorPages().AddRazorRuntimeCompilation();
```

Startup.cs

14.while in Startup.cs ,scroll down Inside a Configure() method, and search for app.UseEndpoints() method.
15. Add The following line of code inside this method:
This way we've added a Controller to our middleware.



```
public void ConfigureServices(IServiceCollection services)
{
    //Adding our Db context.
    //then running [add-migrations AddBookToList] fr
    services.AddDbContext<ApplicationDbContext>(options => options.L
    services.AddControllersWithViews();
    services.AddRazorPages().AddRazorRuntimeCompilation();
}
```

Startup.cs

```
  app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
                endpoints.MapRazorPages();
            });
```

This way the Controller' API will be called.

16.Open **BookController.cs** file agian.
.17.Add **[Route(„api/Book“)]** and
[ApiController] attributes to the top of the
Namespace as it appears in this image:

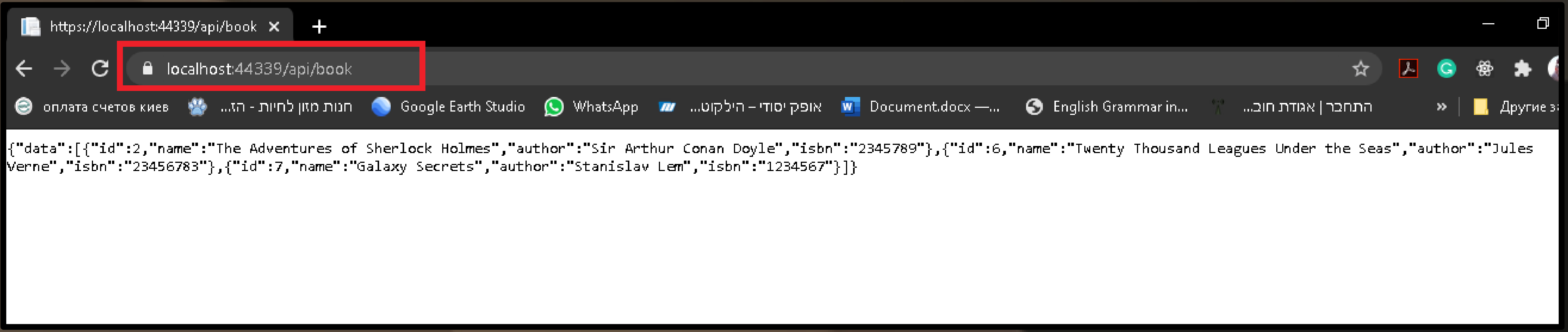
data variable will conatin all of the json information.

This way define the controller is an API controller.
And this is the route that will be used `[Route("api/Book")]`
Since we have added a **Map controller** Inside startup.cs
you can navigate to **this URL :** `"api/Book` and Get request
will return the data from `_Db.Book.ToList()` `as Json file`
Simply run the application and type this URL
https://localhost:44339/api/book
This will list all of the books in json format
On the screen. If you see the books listed in json format
You have successfully called an API.

BookController.cs
Final version

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;

namespace BookListRazor.Controllers
{
    //This way we define a BookController as API Controller
    [Route("api/Book")]
    [ApiController]

    public class BookController : Controller
    {
        private readonly ApplicationDbContext _Db;
        public BookController(ApplicationDbContext db)
        {
            _Db = db;
        }
        [HttpGet]
         //Very important to make it async.
         // Otherwise the data will not be displyed!!!!
         public async Task<IActionResult> GetAll()
        {
            return Json(new { data =await _db.Book.ToListAsync() });
        }
    }
}
```

Calling /api/book/

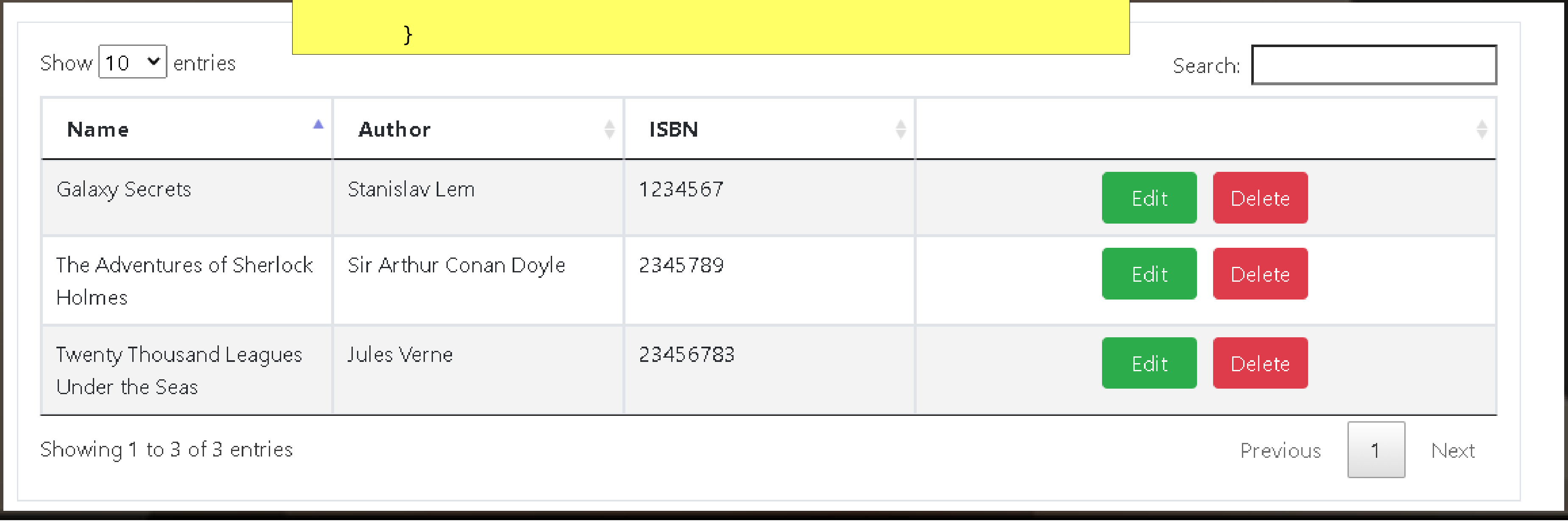


{"data":[{"id":2,"name":"The Adventures of Sherlock Holmes","author":"Sir Arthur Conan Doyle","isbn":"2345789"},{"id":6,"name":"Twenty Thousand Leagues Under the Seas","author":"Jules Verne","isbn":"23456783"},{"id":7,"name":"Galaxy Secrets","author":"Stanislav Lem","isbn":"1234567"}]}

We have successfully recieved a list of books from database in Json format.
That means is our API works as expected.
The next step is to bind this JSON output to a **DataTable API** (the fancy table) Read more here:
Once the Fancy DataTable recieves the Json object it will **render** the books in A fancy format.
Yes, this **DataTable API** Works with a Json format. That is why we needed to transform a **List of books** to a Json format, remember?

BookController.cs

```
[HttpGet]
        public async Task<IActionResult> GetAll()
        {

            return Json(new { data = await _Db.Book.ToListAsync() });


        }
```



Show 10 entries   Search:

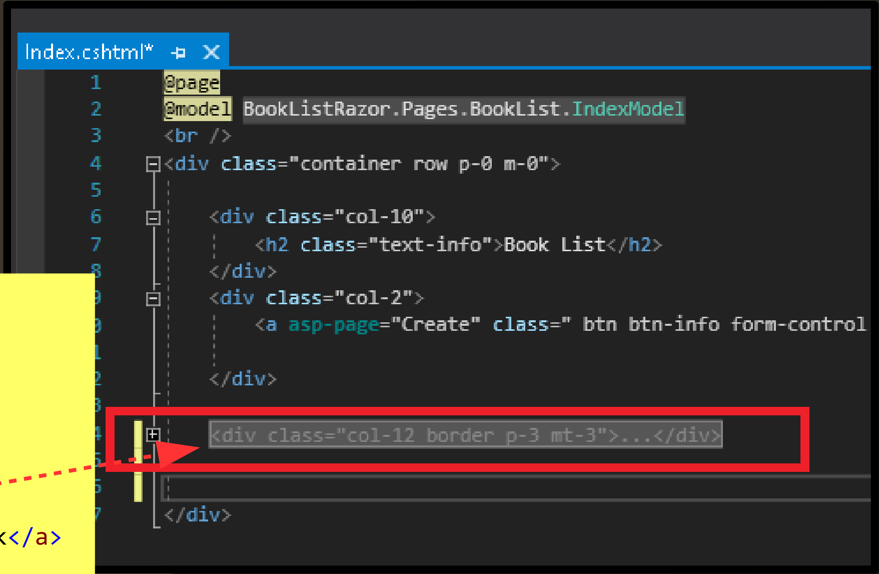
Name	Author	ISBN	
Galaxy Secrets	Stanislav Lem	1234567	Edit Delete
The Adventures of Sherlock Holmes	Sir Arthur Conan Doyle	2345789	Edit Delete
Twenty Thousand Leagues Under the Seas	Jules Verne	23456783	Edit Delete

Showing 1 to 3 of 3 entries   Previous 1 Next

From the athor of this PDF Guide
Stay super sharp
before proceeding to the next steps.
Every single mistake in Json file could
cause render ussues.
If you have problems, download a repository
https://github.com/bhrugen/BookListRazor
And check if it matches you code.
I had problem with this section and stucked on it for 2 days
just because of a small typo in Json file.

Datatables

Next, we will make changes inside Pages/BookList/**Index.cshtml** page
1.Collapse the first <div> as it appears in this image:
2.Create another <div> below the collapsed one as follows:



```
Index.cshtml*
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">

    <div class="col-10">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-2">
        <a asp-page="Create" class=" btn btn-info form-control

    </div>

    <div class="col-12 border p-3 mt-3">...</div>

</div>
```

```
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">
    <div class="col-9">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-3">
        <a asp-page="Create" class="btn btn-info form-control text-white">Create New Book</a>
    </div>

<!--Collapsed Div here-->

 <!--OR dive just a little separator-->
    <div class="col-12" style="text-align:center">
        <br />
        <span class="h3 text-info">OR</span>
        <br /><br />
    </div>

          <!--Json starts here-->
    <div class="col-12 border p-3">

        <table id="DT_load" class="table table-striped table-bordered" style="width:100%">
            <thead>
                <tr>
                    <th>Name</th>
                    <th> Author</th>
                    <th> ISBN</th>
                    <th></th>
                </tr>
            </thead>
        </table>
    </div>
</div>


@section Scripts
    {
      <script src="~/js/bookList.js"></script>

    }
```

3. We crete a table with the id of **DT_load**
4. Then we have to add <thead> tag
To make it work. The DataTables Api has to have atleast one <thead> tag to render other items.

Important to understand:
1.The BookController Api will send the Json **data** object to **BookList.js** file.
Don't worry we will create this file in a minute.

2. The code inside BookList.js file
Will rference to this table's Id= **DT_load**

3 It Will render all the neccesary columns as Needed.

4. First Proceed to the next page and check
If you code matches the code on the next page

Pages/BookList/ Index.cshtml

```
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">
    <div class="col-9">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-3">
        <a asp-page="Create" class="btn btn-info form-control text-white">Create New Book</a>
    </div>
    <!--Collapsed div-->
    <div class="col-12 border p-3 mt-3">
       <form method="post">
           @if (Model.Books.Count() > 0)
           {
               <table class="table table-striped border">
                    <tr class="table-secondary">
                       <th>
                         <label asp-for="Books.FirstOrDefault().Name"></label>
                       </th>
                       <th>
                            @*@Html.DisplayNameFor(m=>m.Books.FirstOrDefault().Author)*@
                           <label asp-for="Books.FirstOrDefault().Author"></label>
                      </th>
                       <th>
                           <label asp-for="Books.FirstOrDefault().ISBN"></label>
                       </th>
                        <th>

                       </th>
                   </tr>
                     @foreach (var item in Model.Books)
                        {
                           <tr>
                               <td>
                                    @Html.DisplayFor(m => item.Name)
                               </td>
                               <td>
                                    @Html.DisplayFor(m => item.Author)
                                </td>
                                <td>
                                    @Html.DisplayFor(m => item.ISBN)
                                </td>
                                <td>
                                   <button asp-page-handler="Delete" asp-route-id="@item.Id" onclick="return confirm('Are you sure you want to delete?')" class="btn btn-danger btn-sm">Delete</button>
                                   <a asp-page="Edit" asp-route-id="@item.Id" class="btn btn-success btn-sm text-white">Edit</a>
                               </td>
                           </tr>
                       }
                 </table>
           } //end if
       else
           {
             <p>No books available.</p>
           }
       </form>
    </div>
    <!--End of Firs Div-->
    <!--OR dive just a little separator-->
    <div class="col-12" style="text-align:center">
        <br />
        <span class="h3 text-info">OR</span>
        <br /><br />
    </div>
    <!--Json starts here-->
    <div class="col-12 border p-3">
        <table id="DT_load" class="table table-striped table-bordered" style="width:100%">
            <thead>
                <tr>
                    <th>Name</th>
                    <th> Author</th>
                    <th> ISBN</th>
                    <th></th>
                </tr>
            </thead>
        </table>
    </div>
</div>
@section Scripts
    {
      <script src="~/js/bookList.js"></script>
    }
```

5. This is a full source code Of index.cshtml file

6. Proceed to the next Page.

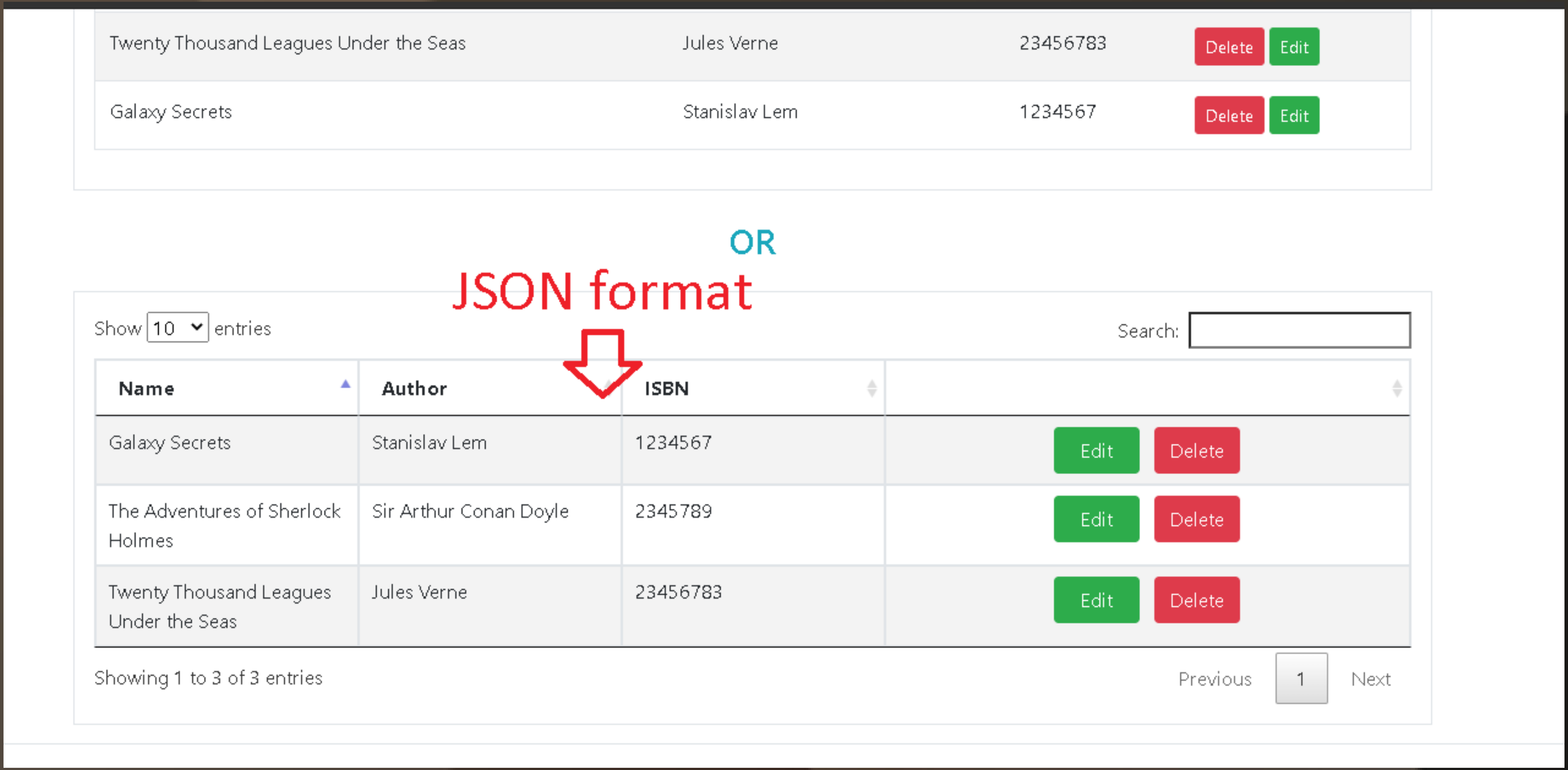
7. Create a New BookList.js file inside wwwroot/Js/ folder.
8. Rightclick wwwrootfolder, and choose Add/NewItem/ choose Javascrip file, and name it as **BookList.js**
9. Add the following code to your BookList.js file.
10.Run the application to see this in action.

```
var dataTable;

$(document).ready(function () {
    loadDataTable();
});

function loadDataTable() {
    dataTable = $('#DT_load').DataTable({
        "ajax": {
            "url": "/api/book",
            "type": "GET",
            "datatype": "json"
        },
        "columns": [
            { "data": "name", "width": "20%" },
            { "data": "author", "width": "20%" },
            { "data": "isbn", "width": "20%" },
            {
                "data": "id",
                "render": function (data) {
                    return `<div class="text-center">
       <a href="/BookList/Edit?id=${data}" class='btn b
            &nbsp;
       <a class='btn btn-danger text-white' style='cursor:pointer; width:70px;' onclick=Delete('/api/book?id='+${data})> Delete</a>
                 </div>`;
                }, "width": "40%"
            }
        ],
        "language": {
            "emptyTable": "no data found"
        },
        "width": "100%"
    });
}
```

Don't make a typos here
This is important!
Double check all the names
If needed.

We will be calling DataTable-API's DataTable. This DataTable Will calll local /api/book. This will get the booklist object
2.It will define collumns headers, name,Author,Isbn, and id.
3.It will then render data based on previously allocated key/value Pairs „data“:“name“ and so on.
4.Finally The dataTableApi will render the <div> element with data of name,author,and isbn to the corresponding collumns based on data: id, + adding <a> tag named Edit + another <a> tag named Delete. Tthe data: id will correspond to BookList/Edit page using ?id={data} parameter.
Please reffer to **this example** for better understanding of how It works.

9. Save and run the application. You should see the following output:



Twenty Thousand Leagues Under the Seas	Jules Verne	23456783	Delete Edit
Galaxy Secrets	Stanislav Lem	1234567	Delete Edit

OR

JSON format

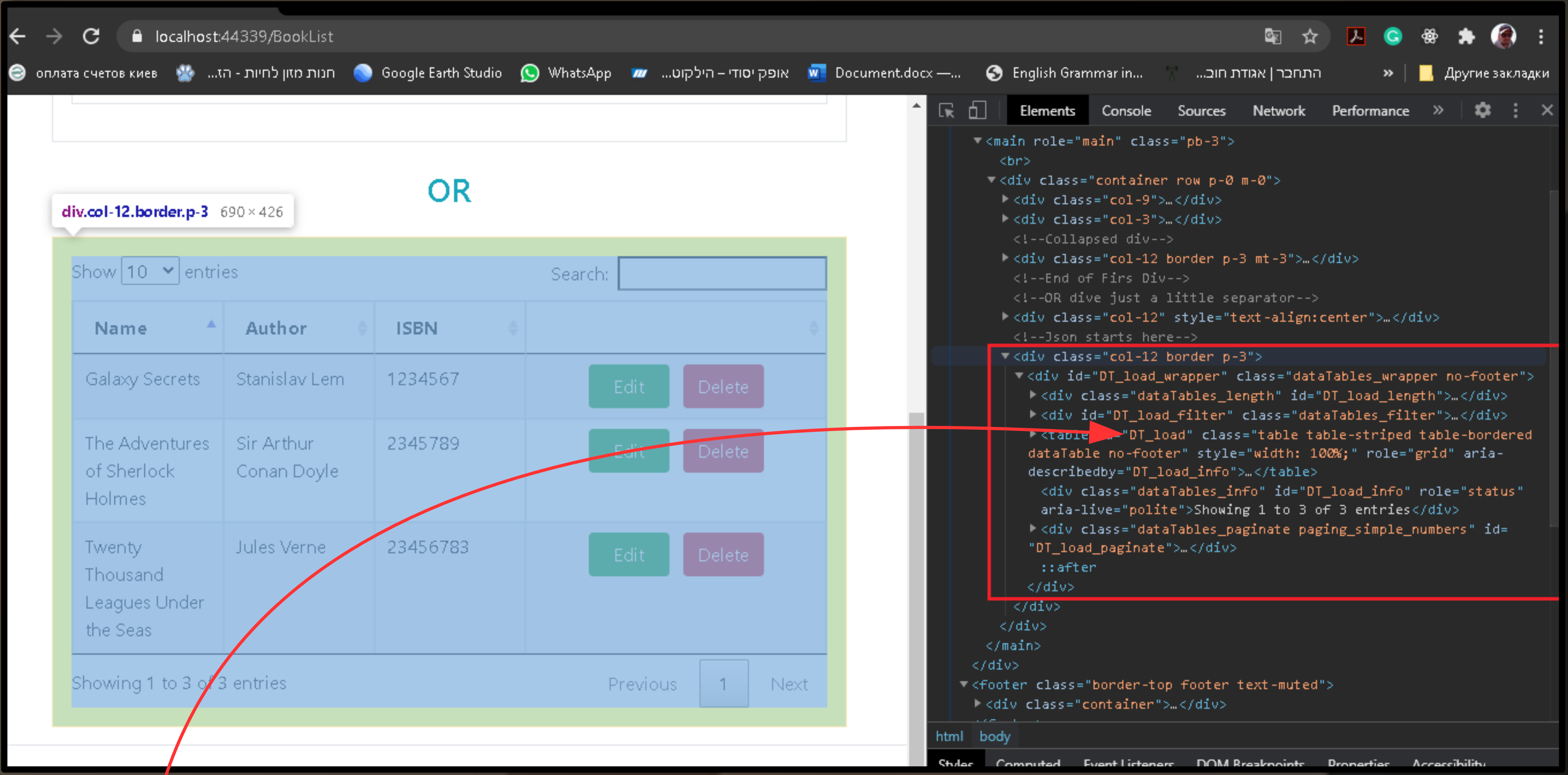
Show 10 entries Search:

Name	Author	ISBN	
Galaxy Secrets	Stanislav Lem	1234567	Edit Delete
The Adventures of Sherlock Holmes	Sir Arthur Conan Doyle	2345789	Edit Delete
Twenty Thousand Leagues Under the Seas	Jules Verne	23456783	Edit Delete

Showing 1 to 3 of 3 entries

Previous 1 Next

10. you can open Dev.Tools F12 and see how the table was rendered to represent the rows, and colllumns.


localhost:44339/BookList
Elements
Console
Sources
Network
Performance
div.col-12.border.p-3 690 × 426
OR
html body
Styles Computed Event Listeners DOM Breakpoints Properties Accessibility

Show 10 entries Search:

Name	Author	ISBN	
Galaxy Secrets	Stanislav Lem	1234567	Edit Delete
The Adventures of Sherlock Holmes	Sir Arthur Conan Doyle	2345789	Edit Delete
Twenty Thousand Leagues Under the Seas	Jules Verne	23456783	Edit Delete

Showing 1 to 3 of 3 entries Previous 1 Next

```
<main role="main" class="pb-3">
  <br>
  <div class="container row p-0 m-0">
    <div class="col-9">…</div>
    <div class="col-3">…</div>
    <!--Collapsed div-->
    <div class="col-12 border p-3 mt-3">…</div>
    <!--End of Firs Div-->
    <!--OR dive just a little separator-->
    <div class="col-12" style="text-align:center">…</div>
    <!--Json starts here-->
    <div class="col-12 border p-3">
      <div id="DT_load_wrapper" class="dataTables_wrapper no-footer">
        <div class="dataTables_length" id="DT_load_length">…</div>
        <div id="DT_load_filter" class="dataTables_filter">…</div>
        <table id="DT_load" class="table table-striped table-bordered
        dataTable no-footer" style="width: 100%;" role="grid" aria-
        describedby="DT_load_info">…</table>
        <div class="dataTables_info" id="DT_load_info" role="status"
        aria-live="polite">Showing 1 to 3 of 3 entries</div>
        <div class="dataTables_paginate paging_simple_numbers" id=
        "DT_load_paginate">…</div>
        ::after
      </div>
    </div>
  </div>
</main>
</div>
<footer class="border-top footer text-muted">
  <div class="container">…</div>
```

```
<table id="DT_load" class="table table-striped table-bordered dataTable no-footer" style="width: 100%;" role="grid" aria-describedby="DT_load_info">
        <thead>
            <tr role="row">
<th class="sorting_asc" tabindex="0" aria-controls="DT_load" rowspan="1" colspan="1" aria-label="Name: activate to sort column descending" aria-sort="ascending" style="width: 48px;">Name</th>
<th class="sorting" tabindex="0" aria-controls="DT_load" rowspan="1" colspan="1" aria-label=" Author: activate to sort column ascending" style="width: 48px;"> Author</th>
<th class="sorting" tabindex="0" aria-controls="DT_load" rowspan="1" colspan="1" aria-label=" ISBN: activate to sort column ascending" style="width: 48px;"> ISBN</th>
<th class="sorting" tabindex="0" aria-controls="DT_load" rowspan="1" colspan="1" aria-label=": activate to sort column ascending" style="width: 133px;"></th></tr>
        </thead>   Rendered area begins here - - - - - - - - - - - - - - - - - - - - - - - - - - - - ►
      <tbody><tr role="row" class="odd"><td class="sorting_1">Galaxy Secrets</td><td>Stanislav Lem</td><td>1234567</td><td><div class="text-center">
                              <a href="/BookList/Edit?id=7" class="btn btn-success text-white" style="cursor:pointer; width:70px;"> Edit </a>
                                &nbsp;
                              <a class="btn btn-danger text-white" style="cursor:pointer; width:70px;" onclick="Delete('/api/book?id='+7)"> Delete </a>
                          </div></td></tr><tr role="row" class="even"><td class="sorting_1">The Adventures of Sherlock Holmes</td><td>Sir Arthur Conan Doyle</td><td>2345789</td><td><div class="text-
center">
                              <a href="/BookList/Edit?id=2" class="btn btn-success text-white" style="cursor:pointer; width:70px;"> Edit </a>
                                &nbsp;
                              <a class="btn btn-danger text-white" style="cursor:pointer; width:70px;" onclick="Delete('/api/book?id='+2)"> Delete </a>
                          </div></td></tr><tr role="row" class="odd"><td class="sorting_1">Twenty Thousand Leagues Under the Seas</td><td>Jules Verne</td><td>23456783</td><td><div class="text-
center">
                              <a href="/BookList/Edit?id=6" class="btn btn-success text-white" style="cursor:pointer; width:70px;"> Edit </a>
                                &nbsp;
                              <a class="btn btn-danger text-white" style="cursor:pointer; width:70px;" onclick="Delete('/api/book?id='+6)"> Delete </a>
                          </div></td></tr></tbody></table>
```

Next step i to implement the Delete Button,

We will have to add some code to BookList.js to deal with this functionality.
By pressing the Delete Button We will Display fancy alert box, and ask if it's okay to delete?
For that we will have to add API call inside our BookController.

1. Open BookController, and add the following code:

```
[HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
            var bookFromDb = await  _Db.Book.FirstOrDefaultAsync(u => u.Id == id);
            if (bookFromDb== null)
            {
                return Json(new { success = false, message = "Error While Deleting" });
            }
            else
            {
                //if bookFromDb[id] == to selected Delete(id)
                //Remove the book a database
                _Db.Book.Remove(bookFromDb);
                await _Db.SaveChangesAsync();
                return Json(new {success= true, message="Delete Successful" });
            }
        }
```

2. Open BookList.Js, and add the following code for the Delete button.

```
<a class='btn btn-danger text-white' style='cursor:pointer; width:70px;'onclick=Delete('/api/book?id='+${data})> Delete </a>
```

3. Next Add a Delete () function to the bottom of your js file
See the code on the next page:

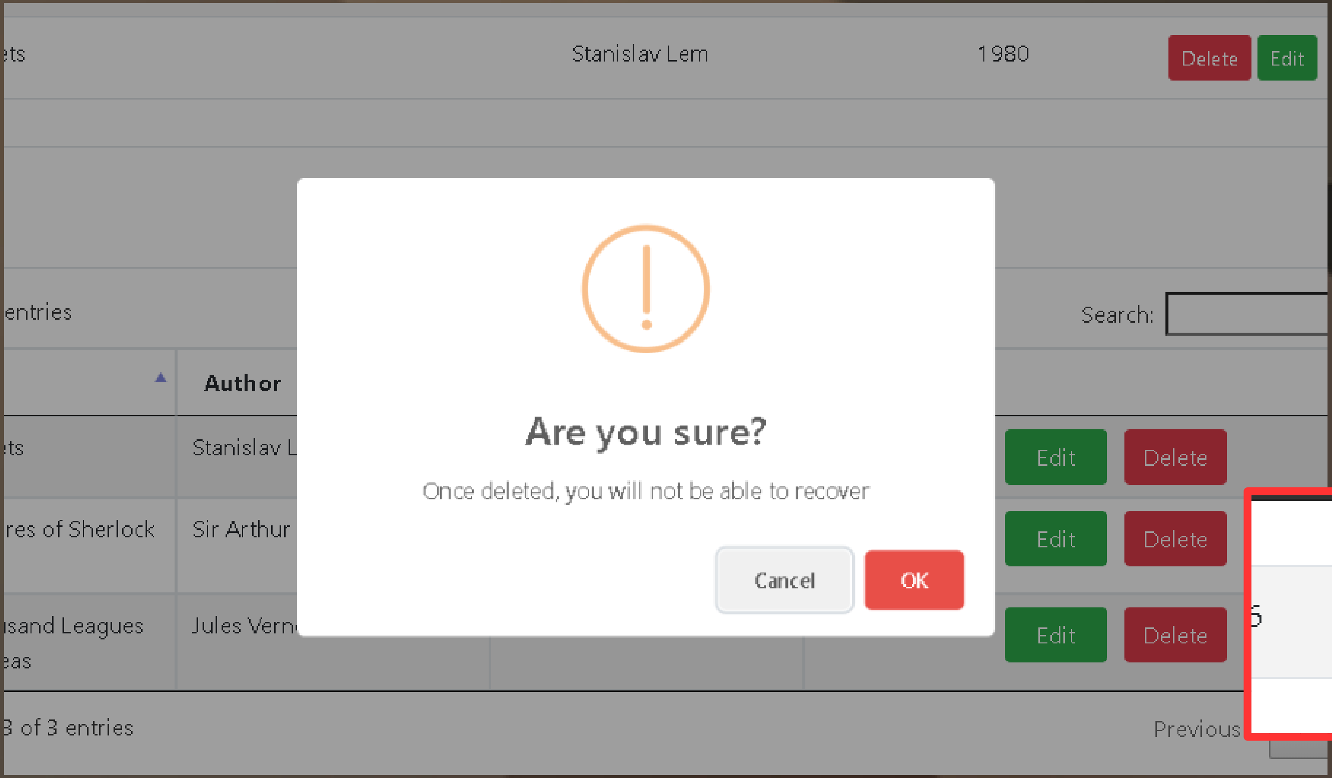
```
var dataTable;

$(document).ready(function () {
    loadDataTable();
});

function loadDataTable() {
    dataTable = $('#DT_load').DataTable({
        "ajax": {
            "url": "/api/book",
            "type": "GET",
            "datatype": "json"
        },
        "columns": [
            { "data": "name", "width": "20%" },
            { "data": "author", "width": "20%" },
            { "data": "isbn", "width": "20%" },
            {
                "data": "id",
                "render": function (data)
                        {
                           return `<div class="text-center">
                                                <a href="/BookList/Edit?id=${data}" class='btn btn-success text-white' style='cursor:pointer; width:70px;'> Edit </a>
                       &nbsp;
                                         <a class='btn btn-danger text-white' style='cursor:pointer; width:70px;'onclick=Delete('/api/book?id='+${data})> Delete </a>
                                   </div>`;
                        }, "width": "40%"
            }
        ],
        "language": {
            "emptyTable": "no data found"
        },
        "width": "100%"
    });
}
function Delete(url) {
    swal({
        title: "Are you sure?",
        text: "Once deleted, you will not be able to recover",
        icon: "warning",
        buttons: true,
        dangerMode: true
    }).then((willDelete) => {
        if (willDelete) {
            $.ajax({
                type: "DELETE",
                url: url,
                success: function (data) {
                    if (data.success) {
                        toastr.success(data.message);
                        dataTable.ajax.reload();
                    }
                    else {
                        toastr.error(data.message);
                    }
                }
            });
        }
    });
}
```

1.The delete recieves the url parameter. Then we display a SweetAlert by using Swal() function.
2.We define title, text, icon properties. (See the list of icons)
The title, text, and icon properties is the build itn SweetAlert Api properties
3.We then set up buttons-true, and dangermode-true.
4.Then we using ajax by typing .then(willDelete)=> this will be the response. Based on this we will hava a function.
5. If user decides to delete we will make an ajax call.
Ajax call will have the type of „DELETE“ a url: and a passed url parameter, and a success parameter will have a function with a retrieved data.
If data success we will be displaying **toastr notification.** Toastr.js It's made for fancy notifications.
6. If data.success, then display a success message, else diplay error message.

7 pay attention of how we were calling the SweetAlert API's functions
And Toastr.js functions

Reeffer to the SweetAlert Website for more info.
Reffer to Toastr.js for more info

Finally Run the application and try deleting one of the books in the lower gridview

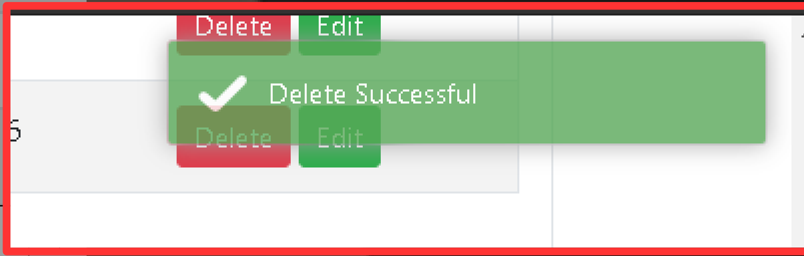


Are you sure?

Once deleted, you will not be able to recover

1.By pressing Delete, we should recieve
A nice popup window.

2. Click Ok button, and watch the Toastr
Notification pops up at the top-right
Corner of the page.



Delete Successful

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListRazor.Model;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

namespace BookListRazor.Pages.BookList
{
    public class UpsertModel : PageModel
    {
        private ApplicationDbContext _DB;
        public UpsertModel(ApplicationDbContext DB)
        {
            _DB = DB;
        }

        [BindProperty]
        public Book Book { get; set; }
        public async Task<IActionResult> OnGet(int? id)
        {
            Book = new Book();
            if (id== null)
            {
                //Create
                return Page();
            }
            else
            {
                //update
                Book = await _DB.Book.FirstOrDefaultAsync(u=>u.Id==id);
                if (Book==null) //if the book is not null retrieve the book from the database
                {
                    return NotFound();
                }
                else
                {
                    return Page();
                }
            }
        }
        //We will be redirecting
        public async Task<IActionResult> OnPost()
        {
            if (ModelState.IsValid)
            {
                if (Book.Id==0)
                {
                    _DB.Book.Add(Book);
                }
                 else
                {
                    _DB.Book.Update(Book);
                }

                await _DB.SaveChangesAsync();

                //After pushing to a database Redirect to index.cshtml
                return RedirectToPage("Index");
            }
            else
            {
                return RedirectToPage();
            }
        }
    }
}
```

Upsert.cshtml.cs

One of the Bhrugen students asked if it's possible to use create, and edit features within a single view, or one razor page?
Yes, absolutely it can be done!

Usually the name for such page is **UpSert.** Because it is a combination of Update, and Insert.

1. Create a new Razor Page in Pages/BookList/ folder and name it **Upsert.** It will be similar to Edit page but with some modifications.
3 Copy the contents of the Edit page iside Upsert.
4. Copy the conntents of the Edtit.cshtml.cs to Upsert.cshtml.cs
5. Or simply copy this code.

6. Save changes.

7.Proceed to the next page.

8.Open Upsert.cshtml. Paste the altered code as follows:

1. Add the following condition statement

Upsert.cshtml

```
<h2 class="text-info">@(Model.Book.Id!=0?"Edit": "Create")</h2>
```

2.Add the same condition property to the hidden property inside <form> tag
If book.id!=0 then display id
Else
  do not show id.

Upsert.cshtml

```
 <form method="post">
        @if (Model.Book.Id!=0)
    {
        //if true we will have the id.
    <input type = "hidden" asp -for= "Book.Id" />
    }
```

3.Lastly we have to do the same for the submit button.

Upsert.cshtml

```
                <button type="submit" class="btn btn-primary form-control">@(Model.Book.Id!=0?"Update": "Create")</button>
```

The label will be changed based on the book.id.
If the book id equals zero, show Create label.
If the book.id not equals to zero, then show update button

So far so good. But how do we test it?
1. Go to BookList.js, and copy this file
2. create a Backup folder inside wwwroot/js/ folder
3.paste the working BikList.js inside backup folder.
4.Go back to a BookList.js file and change thi line of code:

BookList.Js

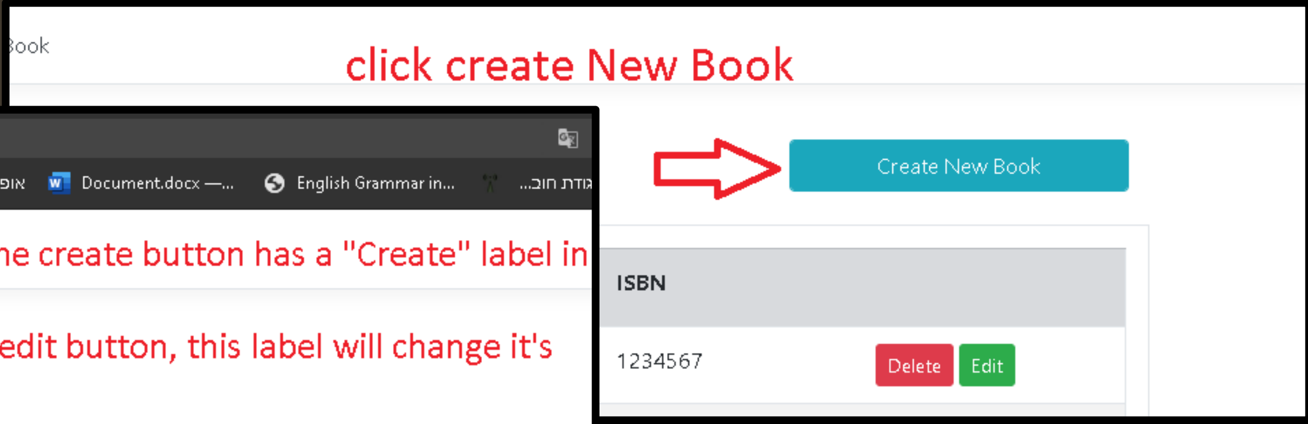
```
 "render": function (data)
            {
           return `<div class="text-center">
                 <a href="/BookList/Upsert?id=${data}" class='btn btn-success text-white' style='cursor:pointer; width:70px;'> Edit </a>
                    &nbsp;
                <a class='btn btn-danger text-white' style='cursor:pointer; width:70px;'onclick=Delete('/api/book?id='+${data})> Delete </a>
                           </div>`;
                }, "width": "40%"
```

5.finally Go back to index.cshtml and change this line of code to **Upsert**

Index.cshtml

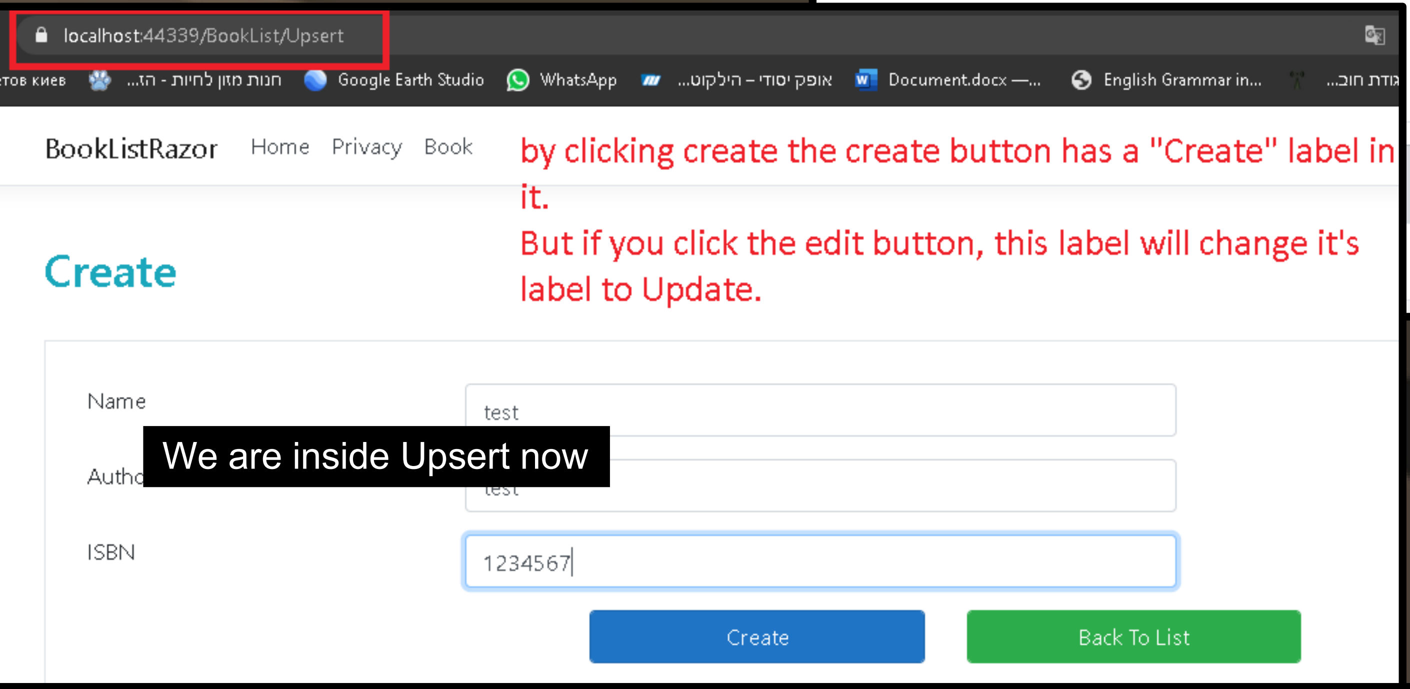
```
<div class="col-3">
      <a asp-page="Upsert" class="btn btn-info form-control text-white">Create New Book</a>
   </div>
```

6. Let's run the application.



click create New Book

ISBN	
1234567	Delete Edit



by clicking create the create button has a "Create" label in it.
But if you click the edit button, this label will change it's label to Update.

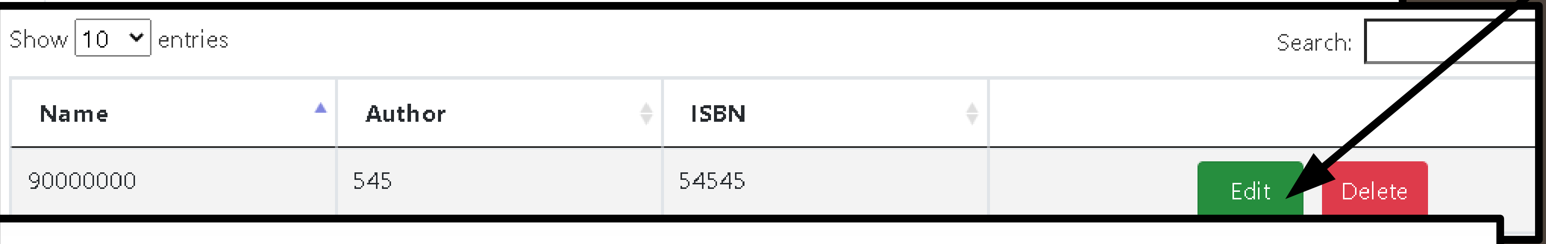
Create

Name	test
Author	test
ISBN	1234567

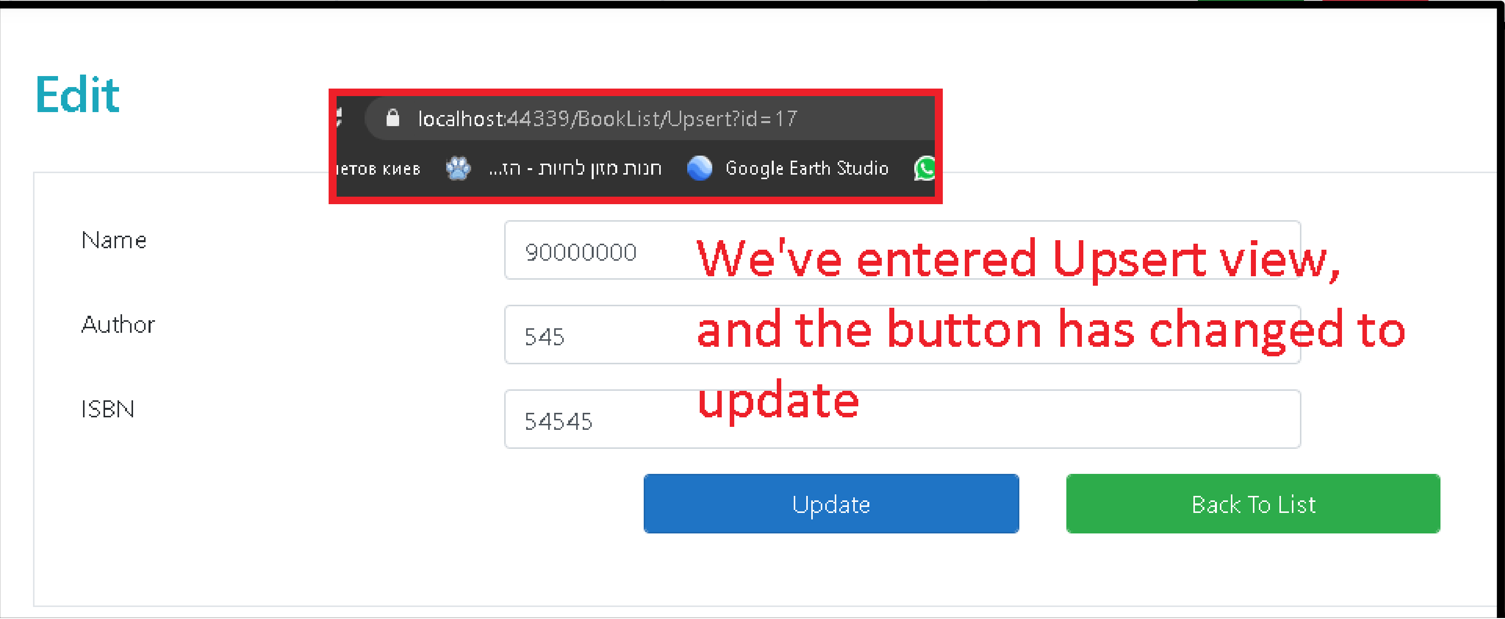
Create   Back To List

We are inside Upsert now

Let's edit Book.



Name	Author	ISBN	
90000000	545	54545	Edit Delete



Edit

Name	90000000
Author	545
ISBN	54545

Update   Back To List

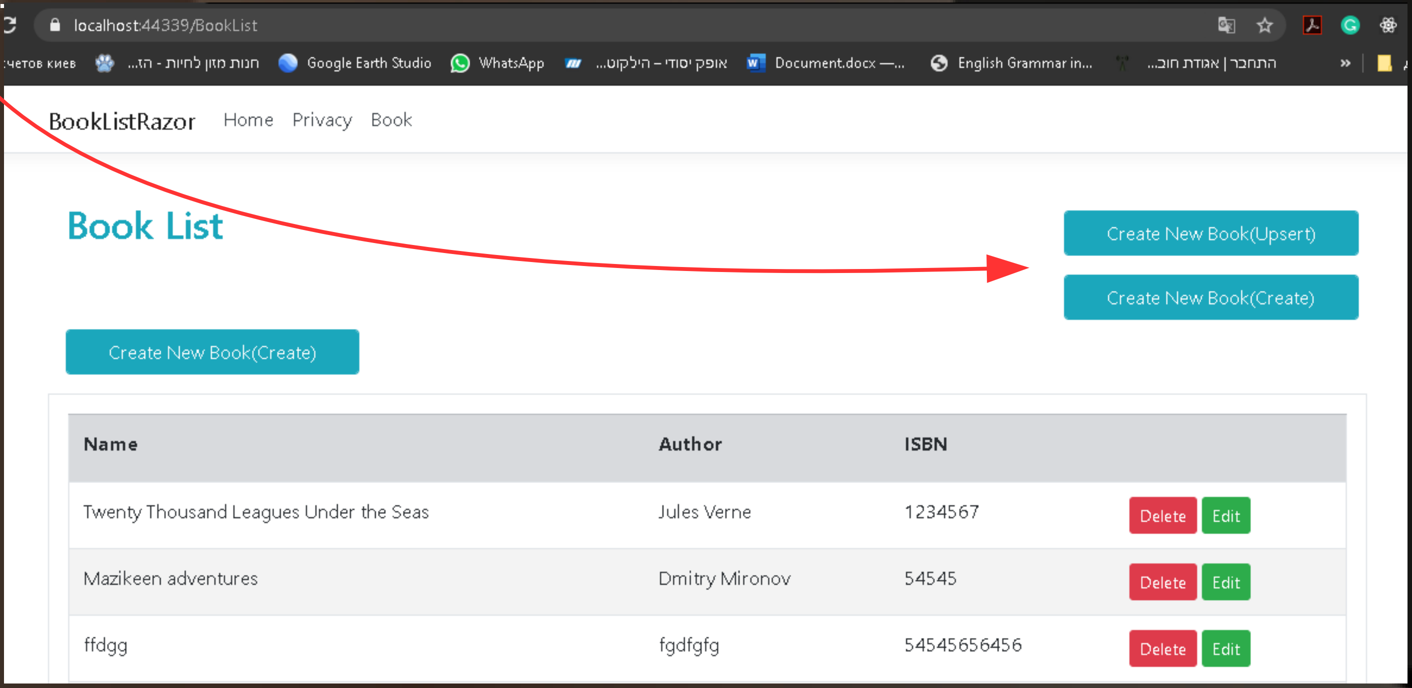
We've entered Upsert view, and the button has changed to update

Next step is to create another Create Button for Create.
So we will have two buttons: 1. for Upsert.cshtml, and 2. for Create.cshtml
So we could choose from one.It is made for educational puprose only.

1. OpenPages/BookList/Index.cshtml
2. Copy/Paste this code to create another button. + Add a simple<br/>
3.Change the tag helper to be asp-page="Create". You will end up with two buttons, one for each RazorView.

```
@page
@model BookListRazor.Pages.BookList.IndexModel
<br />
<div class="container row p-0 m-0">
    <div class="col-9">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-3">
        <a asp-page="Upsert" class="btn btn-info form-control text-white m-2">Create New Book(Upsert)</a>
        <br />
        <a asp-page="Create" class="btn btn-info form-control text-white m-2">Create New Book(Create)</a>
    </div>
```

4.Run the application to see changes.



Name	Author	ISBN
Twenty Thousand Leagues Under the Seas	Jules Verne	1234567
Mazikeen adventures	Dmitry Mironov	54545
ffdgg	fgdfgfg	54545656456

End of Part-1

You successfully completed the Razor pages project

See the MVC PDF guide here