04

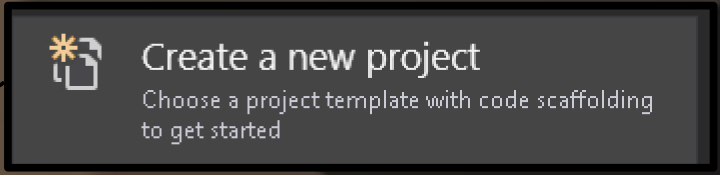
MVC Project

A Brief overview of the Routing in ASP.NET MVC core 3

We will take a look on how the routing is different in MVC
Now when we have completed the Razor application, it is time to explore another type of application which is an MVC Project.

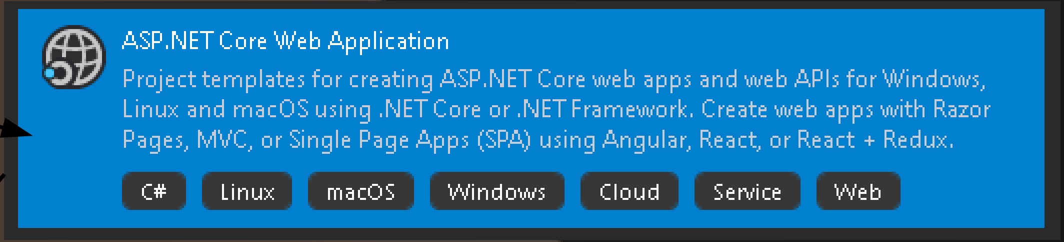
Let's Create a new MVC Project for CRUD operations

1. Open Visual Studio.
2.Select Create a New Project.
3 Select Asp.Net Core Web Application
4. Name your Project BokListMVC
5.Select preferred location
6. click Create button.



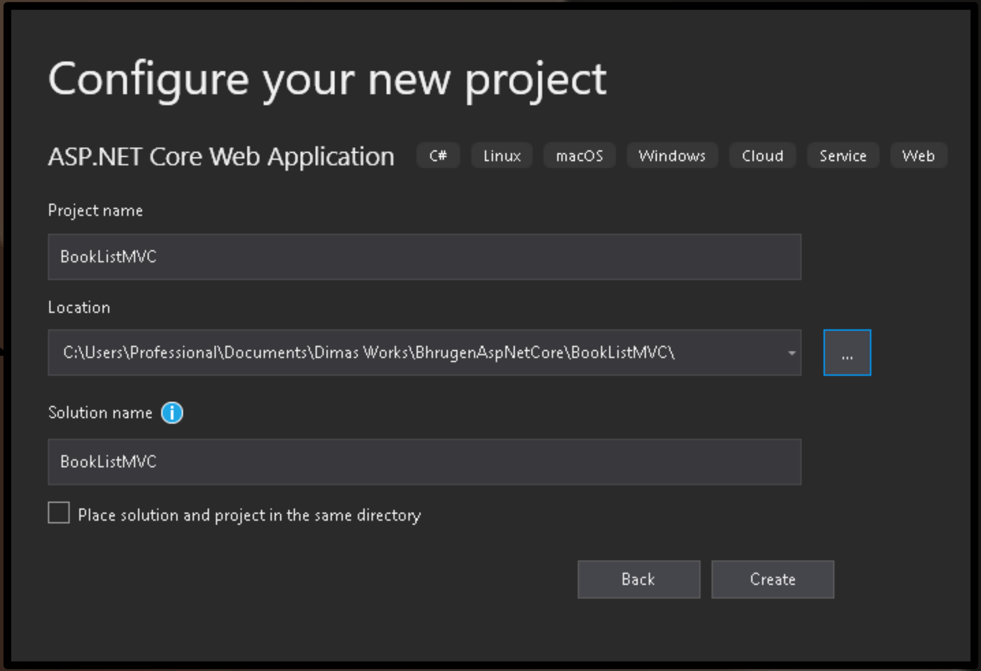
Create a new project

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.



Configure your new project

ASP.NET Core Web Application

Project name: BookListMVC

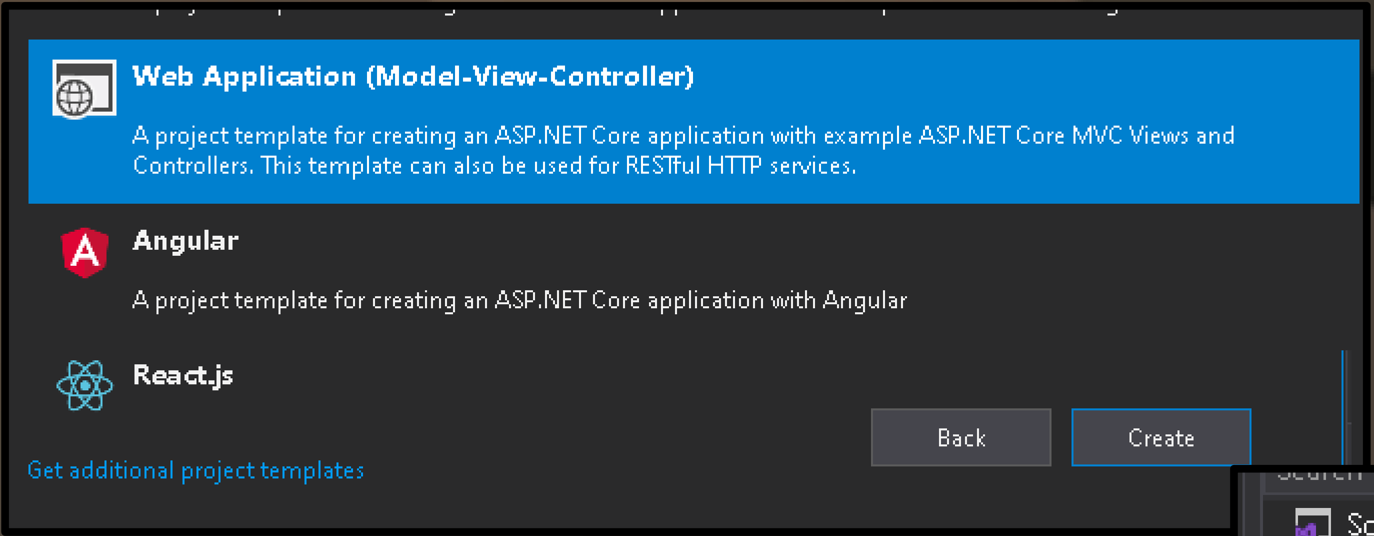
Location: C:\Users\Professional\Documents\Dimas Works\BhrugenAspNetCore\BookListMVC\

Solution name: BookListMVC

☐ Place solution and project in the same directory

Select Option for our project

1.Select Asp.Net Core 3.1 from a Dropdown list.
2 Select Model-View-Controller (MVC) for our project
3. Click Create button.



Web Application (Model-View-Controller)

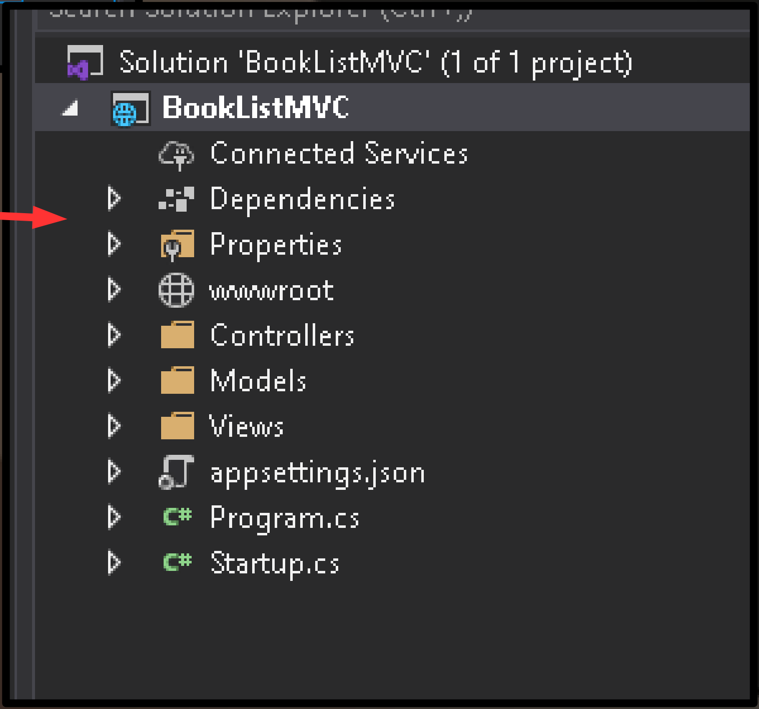
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular

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

React.js

4. The MVC project being created.



This is slightly different than what we have seen. Here we will have

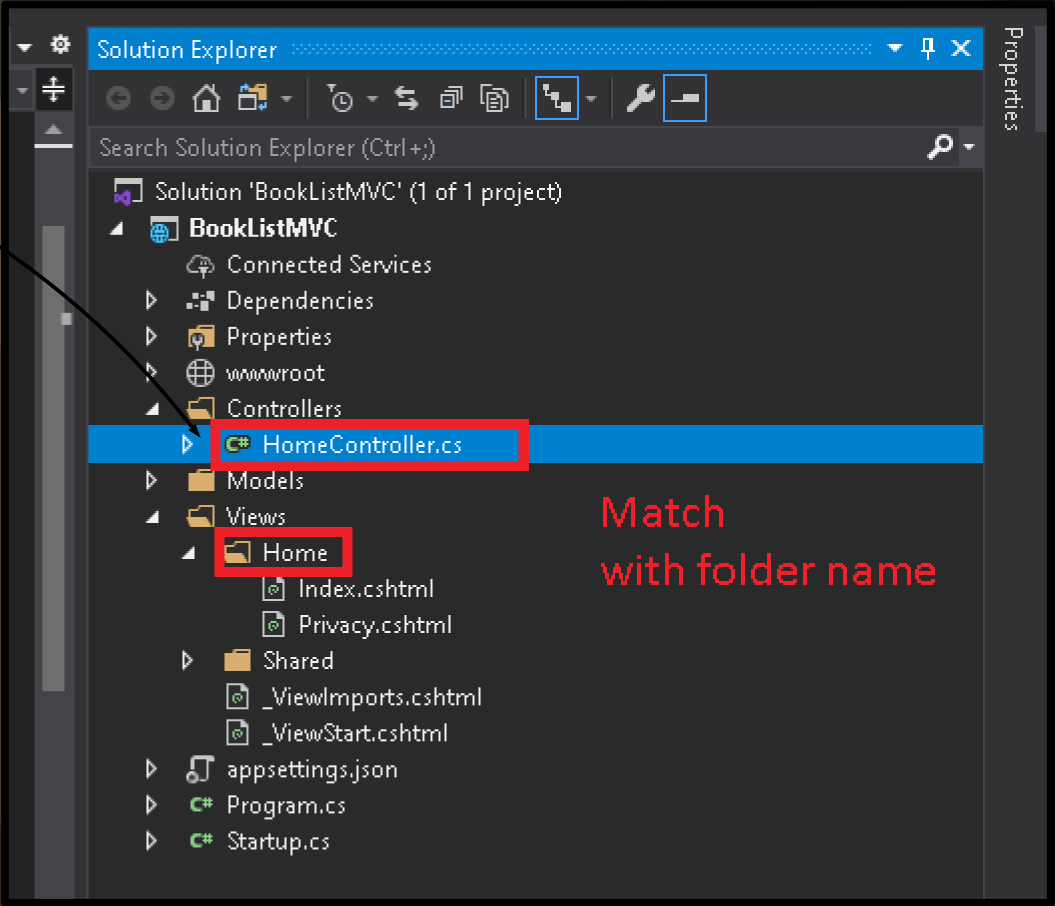
- Controllers which will have the logic.
- Models will have the data, and
- Views will be the UIComponent.
-

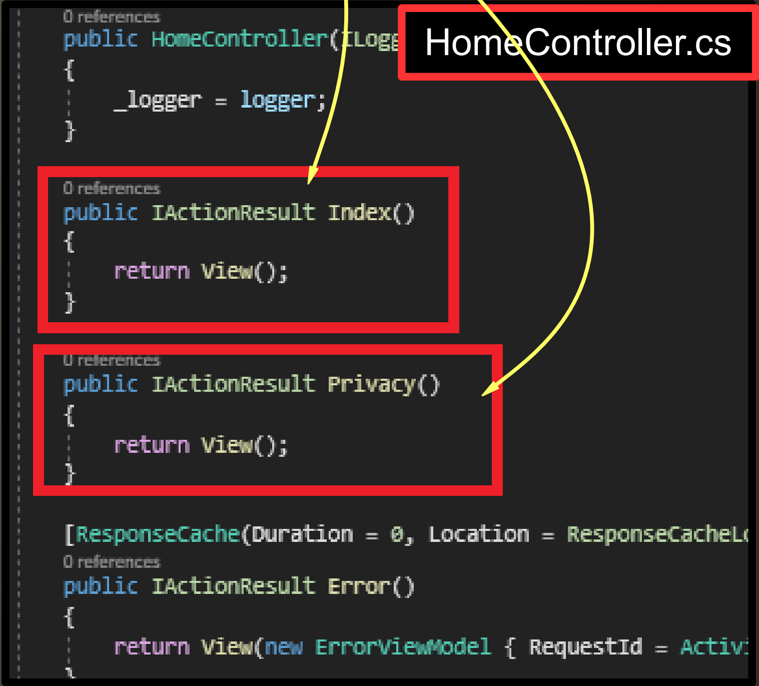
The main Difference is inside Startup.cs file, inside ConfigureServices() **method**
We do not have Add.RazorPages. Instead we have:
services.AddControllersWithViews(); method

We have 3 main folders: Controllers, Models, and Views.
First folder is the Controllers folder: Inside of which we have the main logic of our application.
The HomeController starts from the Word Home which matches the Home Folder inside Views folder.

So if we have a view for home page, the Controller name will be Home followed by The Controller Keyword. Like HomeController

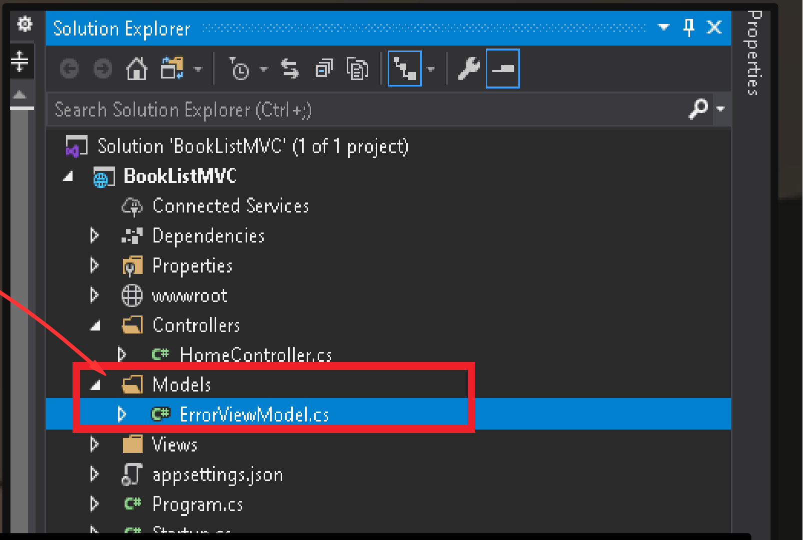
Inside a HomeController we have Methods. These methods called as Actions.




HomeController.cs

```
0 references
public HomeController(ILogg
{
    _logger = logger;
}

0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult Privacy()
{
    return View();
}

[ResponseCache(Duration = 0, Location = ResponseCacheLo
0 references
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activi
```

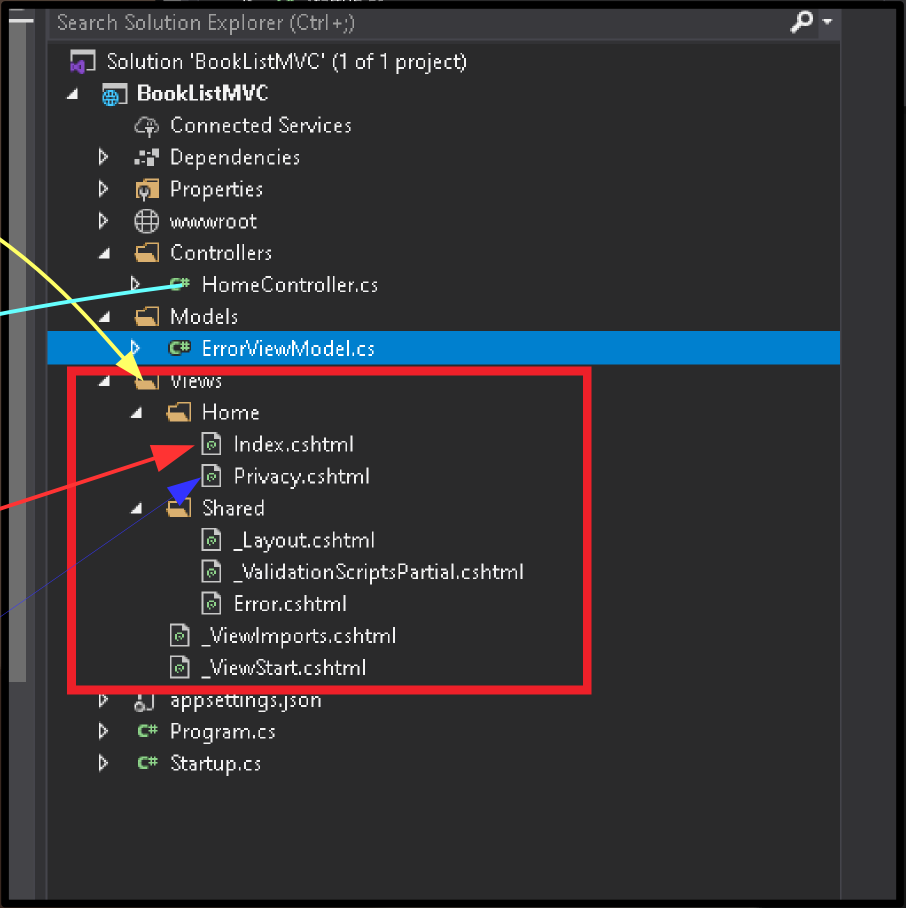
Next folder that we have is the Models folder.
As we proceed we will be adding a new models in this folder.
Any data table we have inside a database, we will have to create a corresponding C# class-model to this folder.
We will also have a ViewModels, which is a combination of multiple models and we will discuss them in detail later on.



```
Solution 'BookListMVC' (1 of 1 project)
  BookListMVC
      Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      HomeController.cs
    Models
      ErrorViewModel.cs
    Views
    appsettings.json
    Program.cs
    Startup.cs
```

Finally we have a Views folder.
If we expand the Shared folder, we have Layout.cshtml, and _ValidationScripts files that we've discussed in the Razor project.
We also have a new Folder Named Home.
In which we have Index.cshtm, and Prtivacy.cshtml.

Notice that inside the HomeController we have Actions named Index(), and Privacy()



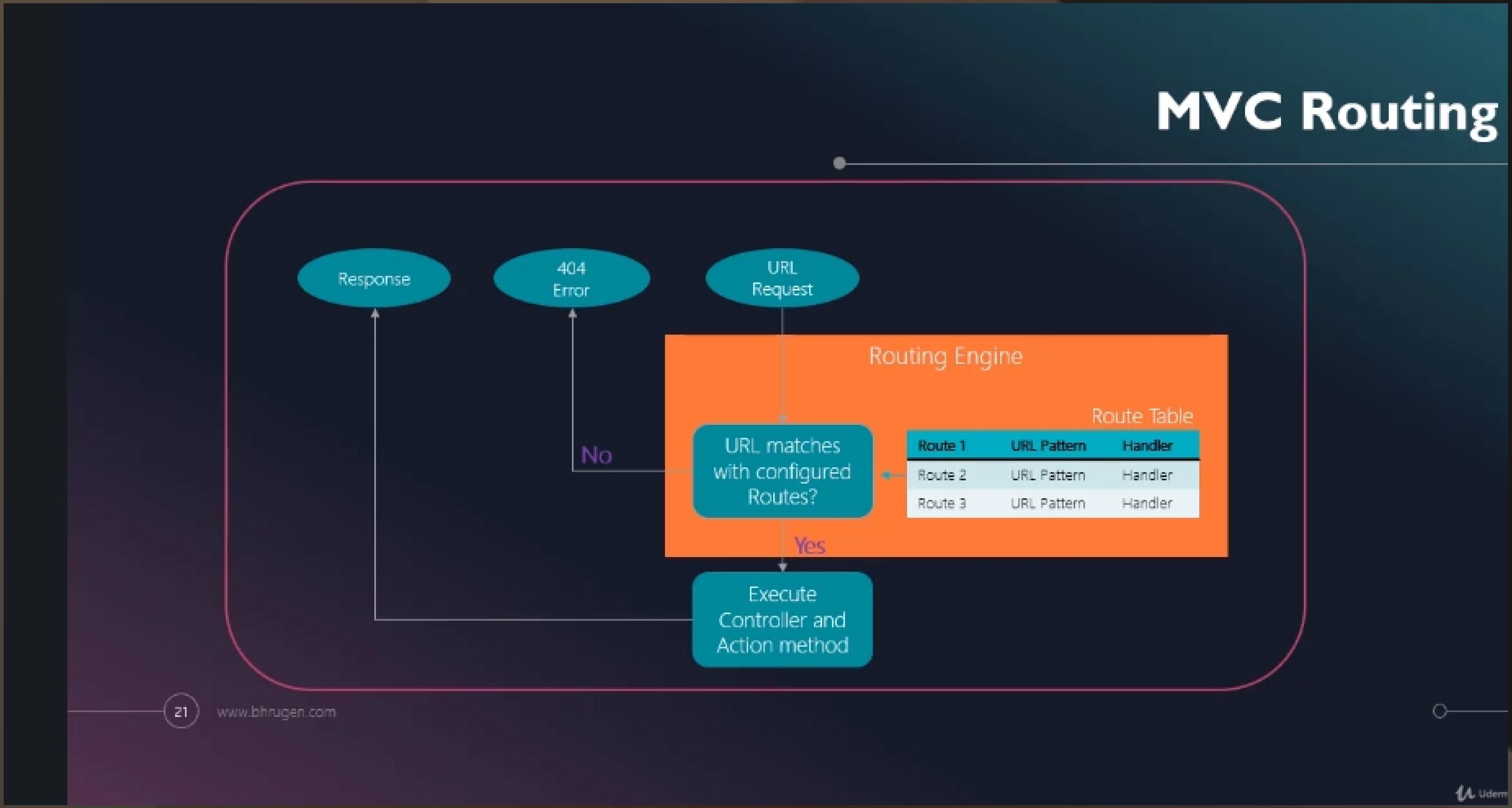
```
Solution 'BookListMVC' (1 of 1 project)
  BookListMVC
      Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      HomeController.cs
    Models
      ErrorViewModel.cs
    Views
      Home
        Index.cshtml
        Privacy.cshtml
      Shared
        _Layout.cshtml
        _ValidationScriptsPartial.cshtml
        Error.cshtml
      _ViewImports.cshtml
      _ViewStart.cshtml
    appsettings.json
    Program.cs
    Startup.cs
```

HomeController.cs

```
public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }
```

If we add a new controller, let's say BhrugenController then inside the Views folder
we will have to add a new folder named **Bhrugen,** and create a **Bhrugen View** inside of it.
You will understand this concept more, when we start coding out project.
This was a brief overview of the Models, Controller, and Views folders.

How routing fork in typical MVC?



Route 1	URL Pattern	Handler
Route 2	URL Pattern	Handler
Route 3	URL Pattern	Handler

How routing work in typical MVC

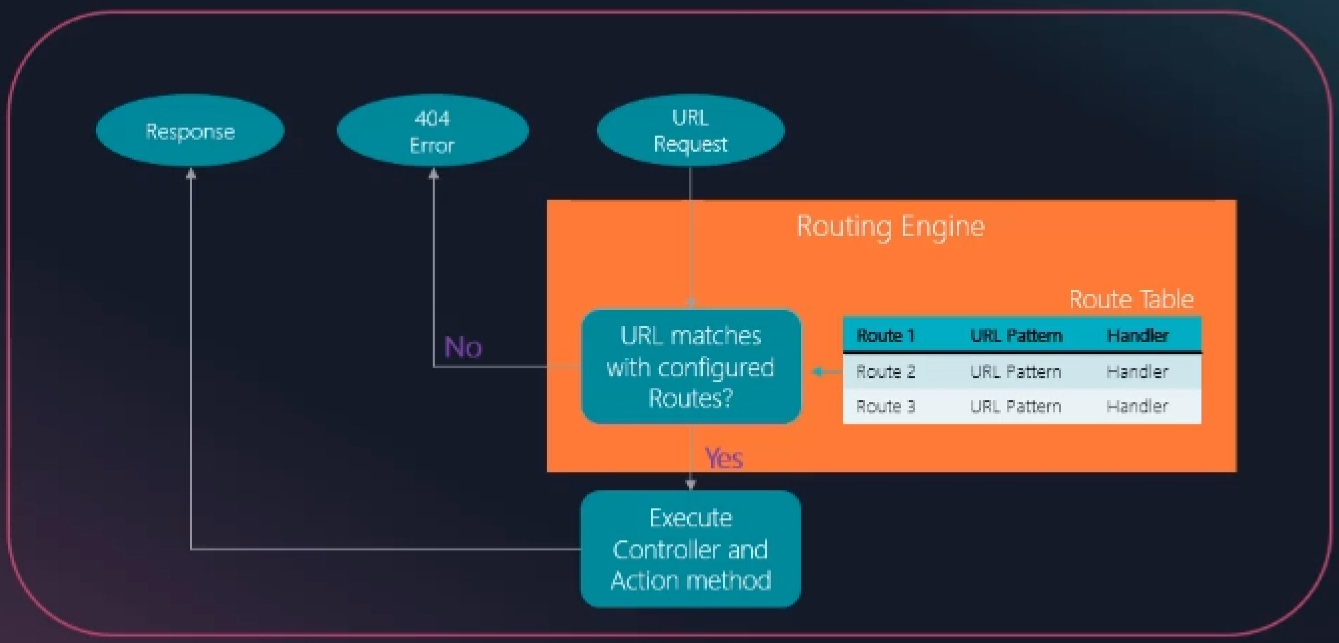
Asp.net MVC is a pattern merging system which enables you to match incoming request to a particular MVC action defined in the controller. When asp.net routing engine receives a Request at Runtime, it finds a match against the URL pattern defined in the route table. If any match is found, then it forwards this request to the controller. Otherwise it will return 404 not found message. When we have created our project, Routes where added automatically To our MVC project. If you go to your application,and open **Startup.cs** file you will see the following: We add **services.AddControllersWithViews()**. And in the middleware we see: **app.UseRouting()** And **app.UseEndPoints()**;

Startup.cs

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

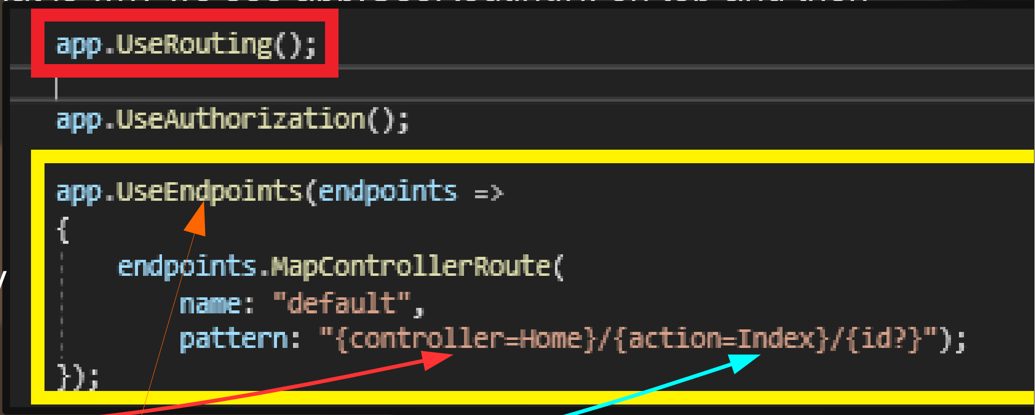
Startup.cs

```
app.UseRouting();

            app.UseAuthorization();

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



Response
404 Error
URL Request
Routing Engine
Route Table
URL matches with configured Routes?
No
Yes
Execute Controller and Action method
Route 1 | URL Pattern | Handler
Route 2 | URL Pattern | Handler
Route 3 | URL Pattern | Handler

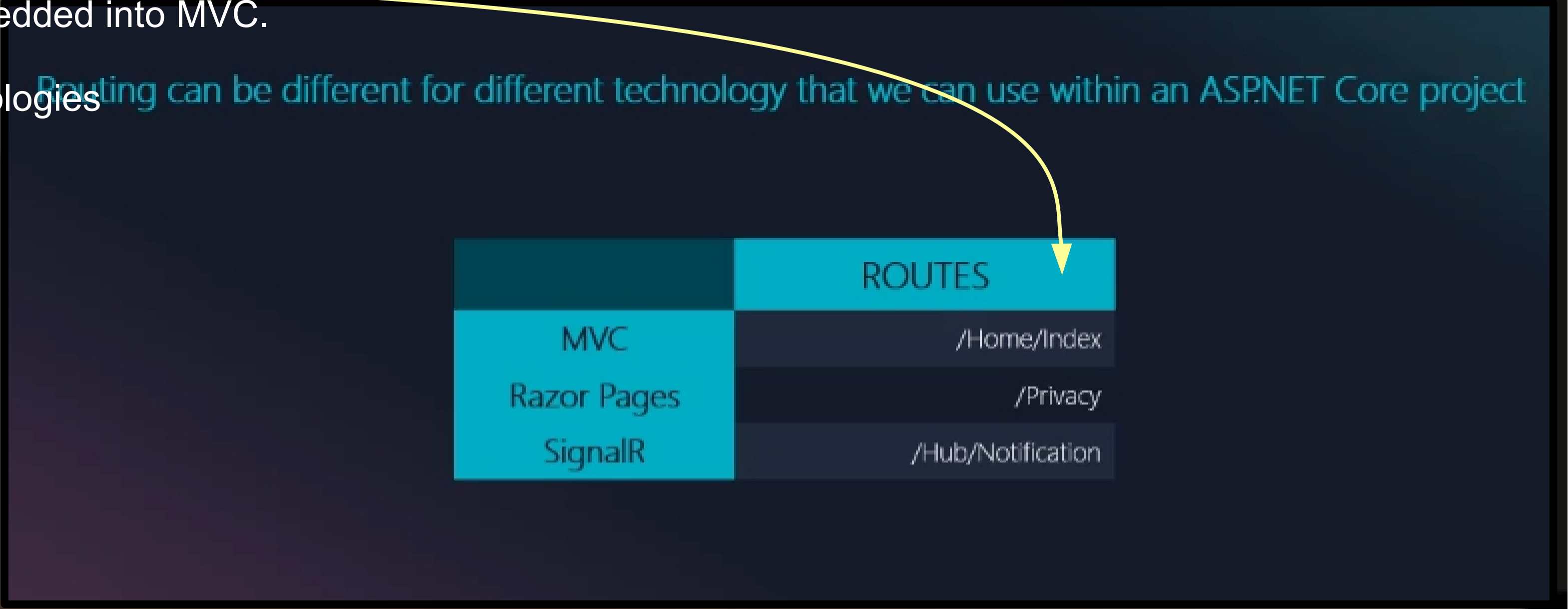
You can keep endpoints, or you can also use the routing without the endpoints.
In version 1, an 2 of Asp.net Core Enpoints was a part of MVC. But now it is a separate piece of middleware to make Routing available to all of the middleware, not just to MVC. That is why we see app.UseRouting(); on top and then app.UseEndpoints(); at the bottom.



```
app.UseRouting();

app.UseAuthorization();

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

Endpoint is a **URL** where incoming request will end up processing by the middleware.
If you have used MVC before, you know that you have to specify the routes for this, and right here we specify a default route in which if nothing is defined, it will look for the **HomeController**, and it will cal the Index() action inside that.

Asp.Net Core it is not just MVC. It supports different technologies which uses **Routes.**
Like Razor pages, or SignalR, and MVC. And the routing is different for each one of them.
All of these technologies use middlewares that registers the **endpoints.** Ass you see we registered only one endpoint for now which is MVC (pattern)   because that is what we going to use in this application. In previous version of - Asp.Net Core, routing was embedded into MVC.
But that cannot work anymore.
Think of using a different technologies
Which use different routes.

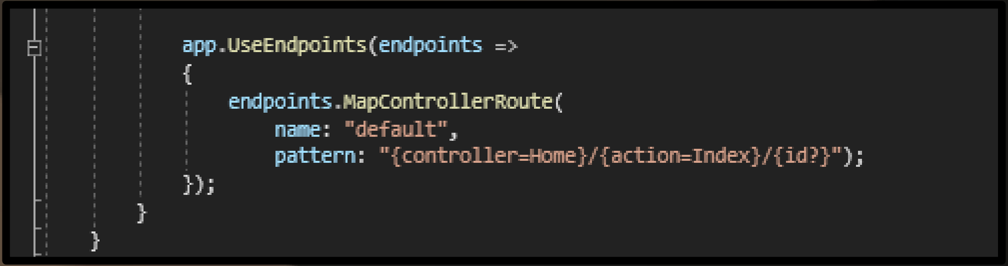


Routing can be different for different technology that we can use within an ASP.NET Core project

	ROUTES
MVC	/Home/Index
Razor Pages	/Privacy
SignalR	/Hub/Notification

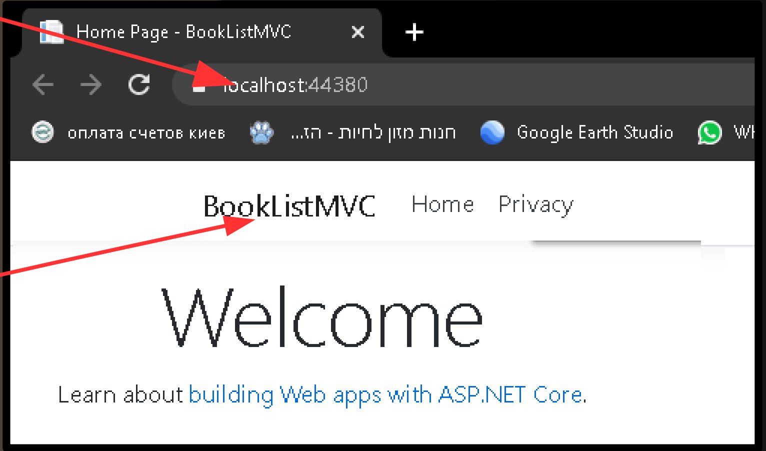
First we will add a user routing which will make selected endpoint choices **available** to all the middlewares that follows after That, and when we will do use endpoints, at that point we will be able to register and execute the endpoints.

In **Lambda expression** we have several **extension methods** to register endpoints.
One of them is the **MapControllerRoute()**
Let's try run the application, and see what happens.



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

In the URL we don't have anything defined. It is just a localhost and the port number.



Home Page - BookListMVC

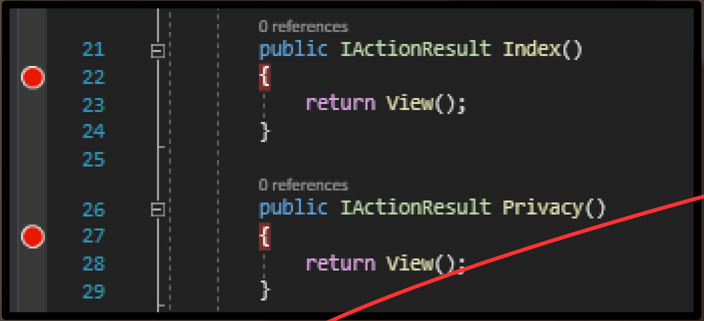
localhost:44380

BookListMVC Home Privacy

Welcome

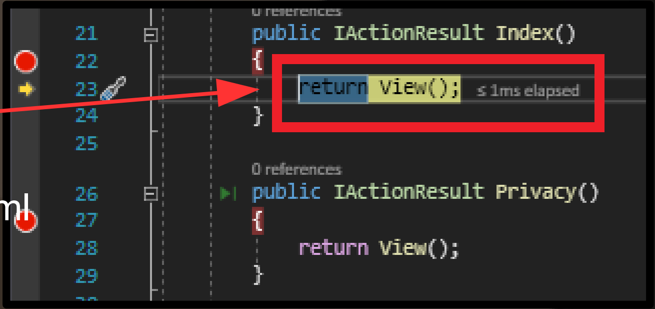
Learn about building Web apps with ASP.NET Core.

1.Open HomeController.cs file
2. Add a breakpoint inside Index(), and Privacy() Action-methods.

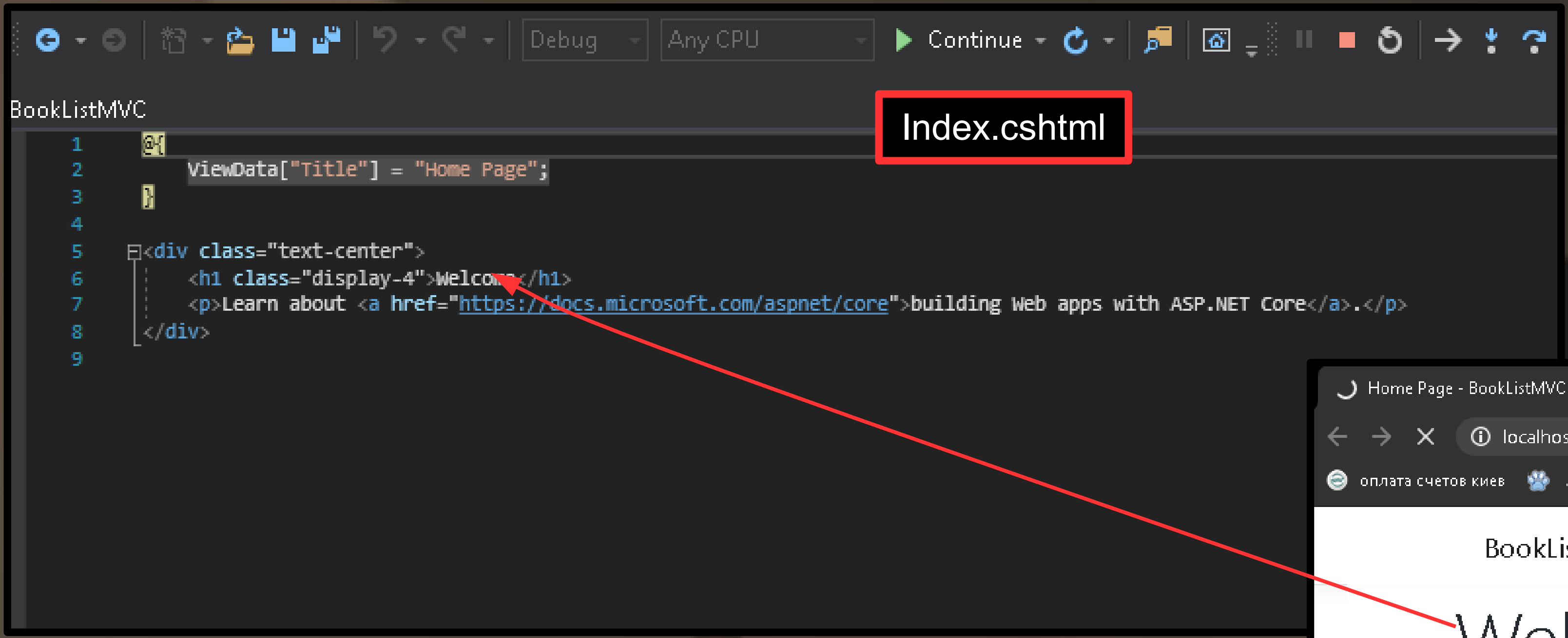


```
        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Privacy()
        {
            return View();
        }
```

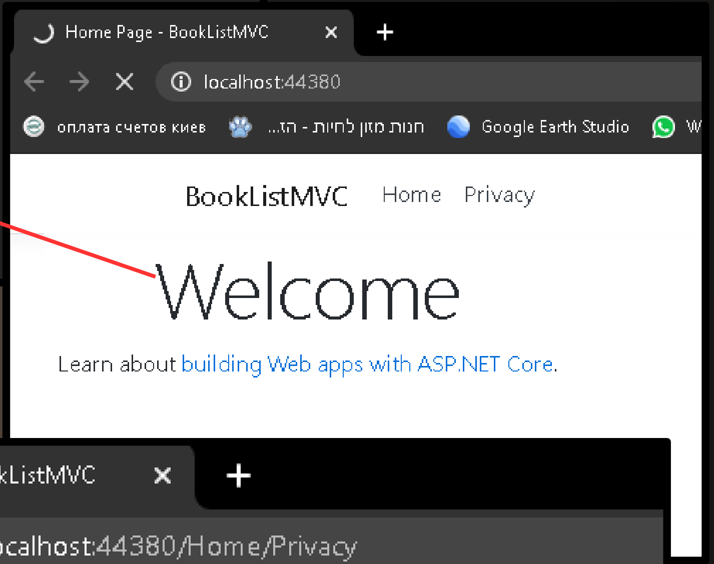
3. Click the BookListMVC link. See what happens.
4. The program execution should stop at IActionResult Index() method
   As it appears in this image.
5. When you click Continue, this will fire Index() method which will open index.cshtml



```
        0 references
        public IActionResult Index()
        {
            return View();  ≤ 1ms elapsed
        }

        0 references
        public IActionResult Privacy()
        {
            return View();
        }
```



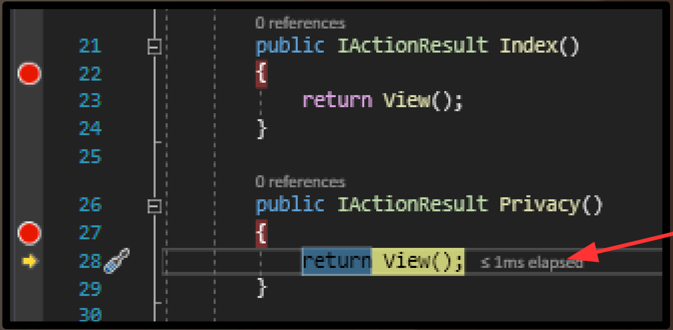
```
@{
    ViewData["Title"] = "Home Page";
}

<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>
```



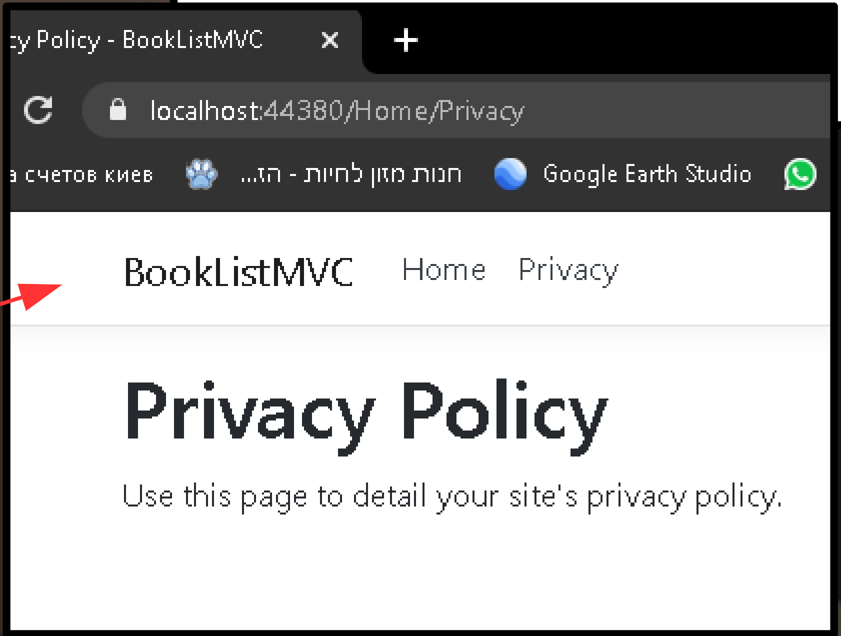
Welcome

Learn about building Web apps with ASP.NET Core.

6. When we click on Privacy. We should stop at privacy break point.



```
0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult Privacy()
{
    return View(); ≤ 1ms elapsed
}
```



Privacy Policy

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

7. Click Continue and you will get the privacy.cshtml.

localhost:44380/Home/Privacy

Controller name. Action method

If you have not define anything it will go to Index.cshtml by default.
So if you define localhost:5000/Home, it loads the HomeController and loads the Index() method.
If you not define localhost:5000/Home, it will again go to HomeController and load |Index.cshtml.
That is because you have define it in the Endpoints to use as a **default** controller.

Startup.cs

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

This was a brief overview of routing in MVC

Proceed to the next page.

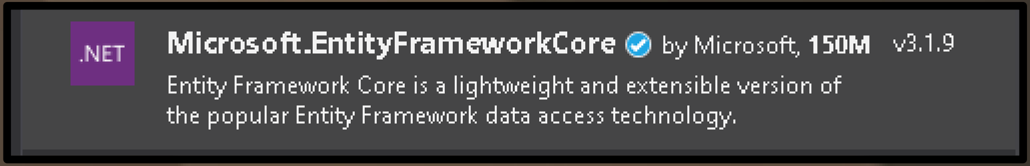
05

Book List MVC

Create Book Model and push to database

Before you proceed add these NuGet Packages to your project.

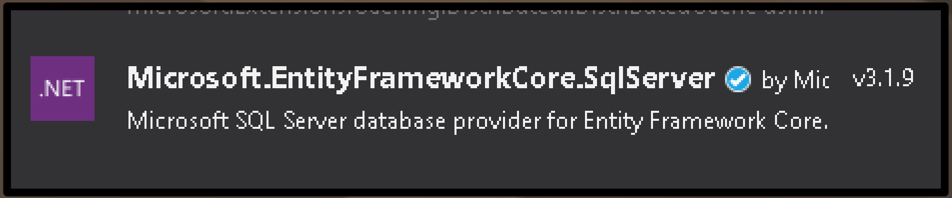
1



.NET **Microsoft.EntityFrameworkCore** by Microsoft, **150M** v3.1.9

Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

2

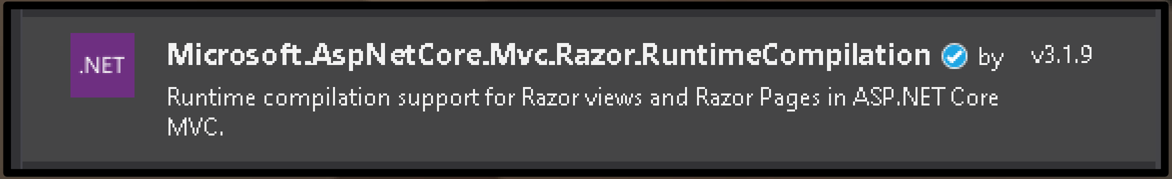


.NET **Microsoft.EntityFrameworkCore.SqlServer** by Mic v3.1.9

Microsoft SQL Server database provider for Entity Framework Core.

SQL server connection

3

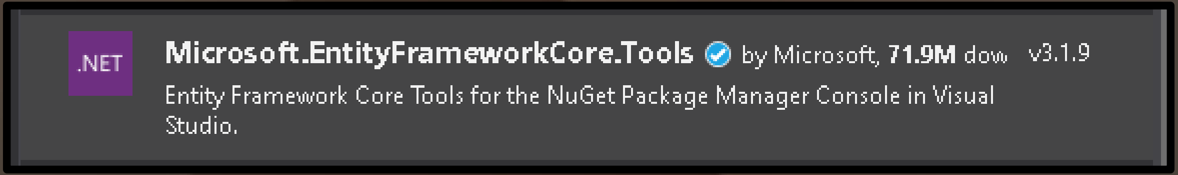


.NET **Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation** by v3.1.9

Runtime compilation support for Razor views and Razor Pages in ASP.NET Core MVC.

Enables page refresh while
Running the app

4



.NET **Microsoft.EntityFrameworkCore.Tools** by Microsoft, **71.9M** dow v3.1.9

Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Enable console commands
Such add-migration, or
update-database

1.Open Solution Explorer.
2. In Models' folder create a new class. Name it as Book.
3. Your code should look as follows:

Book.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace BookListMVC.Models
{
    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; }

    }
}
```

4. Once the model in place, we need to add it to database.
There are multiple things that we have to do to add it to database.
First we will create a dbContext class inside our models folder.

5. Create a new class inside Models folder, and name it as **ApplicationDbContext**

6. Paste the following code:

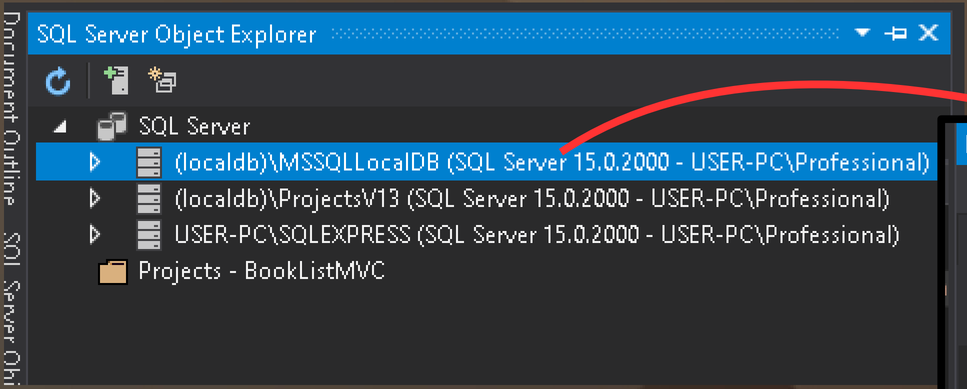
ApplicationDbContext

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

namespace BookListMVC.Models
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {

        }

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

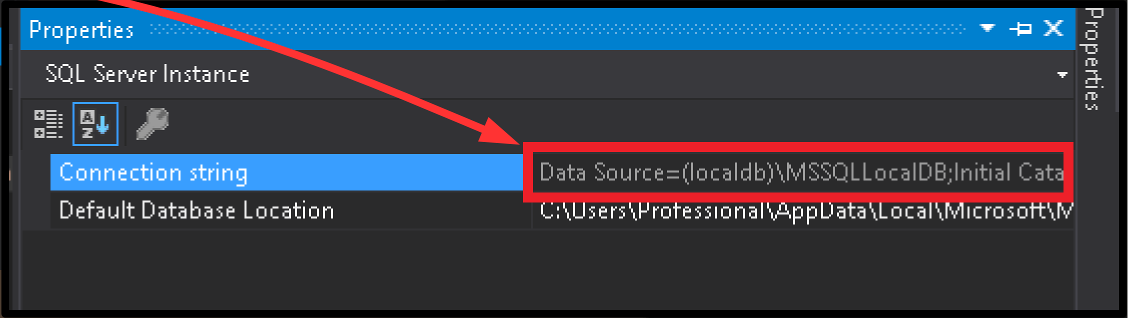
7. Next go to **appsettings.Json** and paste the following connection string property.

```
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=(localdb)\\MSSQLLocalDB;Initial Database=BookListMVC;Integrated Security=True;
                          TrustedConnection=true;MultipleActiveResultSets=true"
  },
```

8.You can use SqlServerManagment studio, or Visual studio internal Server Object Explorer to check your connection string.


SQL Server Object Explorer

```
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 - BookListMVC
```


Properties

SQL Server Instance

Connection string	Data Source=(localdb)\MSSQLLocalDB;Initial Cata
Default Database Location	C:\Users\Professional\AppData\Local\Microsoft\M

1.Right click the local db and choose properties
2.Copy the connection string, and paste it within connectionStrings section Inside appsettings.json.
3. If needed set the TrustedConnection= true; and MultipleActiveResultSets= true;
4. Last thing that we need to do is to configure **Startup.cs** file to use that connection string.
5.Open startup.cs file.
6. Locate ConfigureSerices() method
7.Add the following code to this method:

```
.AddRazorRuntimeCompilation();
```

This method allows us to refresh the page while running the project. Rather then stopping the whole application and start again.

Startup.cs

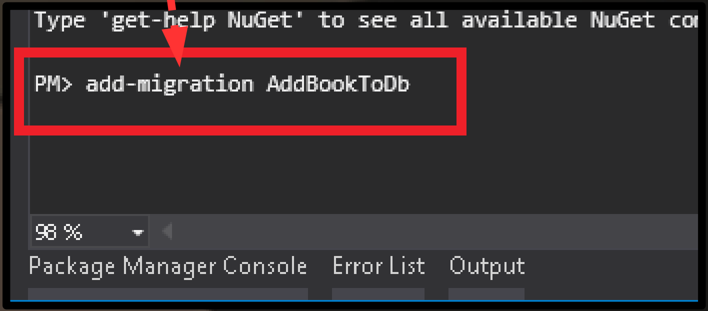
```
public void ConfigureServices(IServiceCollection services)
        {
            //Don't forget to install EntityFrameworkCore.SqlServer
            //Passing the connection string from configuration
            services.AddDbContext<ApplicationDbContext>(options
                => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
            services.AddControllersWithViews().AddRazorRuntimeCompilation();
        }
```

Next step Adding Migration.

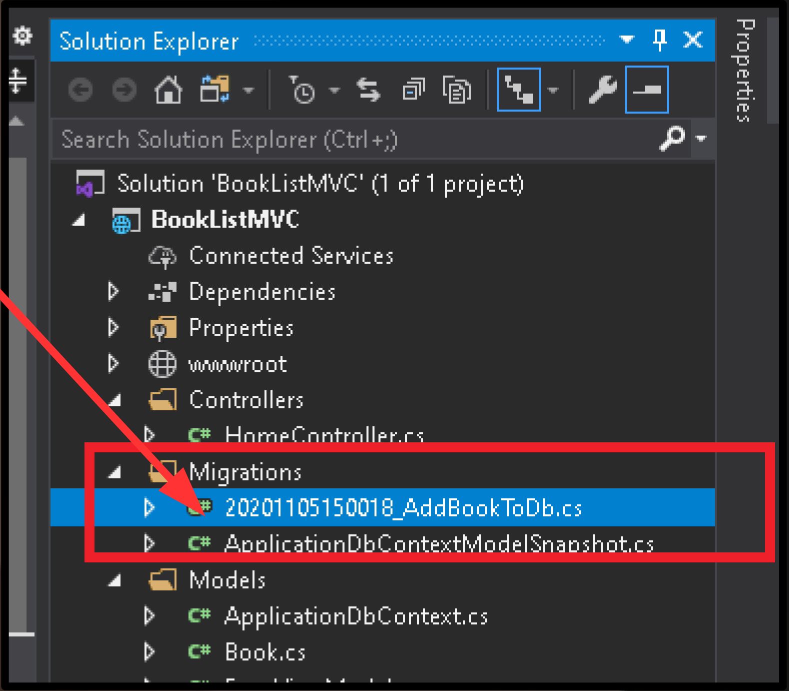
1. Open Tools/NugetPackageManaqger/PackageManagerConsole.
2. Execute the following command: **add-migration AddBookToDb**

20201105150018_AddBookToDb
This file was created automatically By running add-migration command.
Think of this file as of SQL SCRIPT file
You Can run this "script" by typing update-database command in NugetManager Console.
add-migration [yourfilename] creates the script.
Update-database   executes this script.

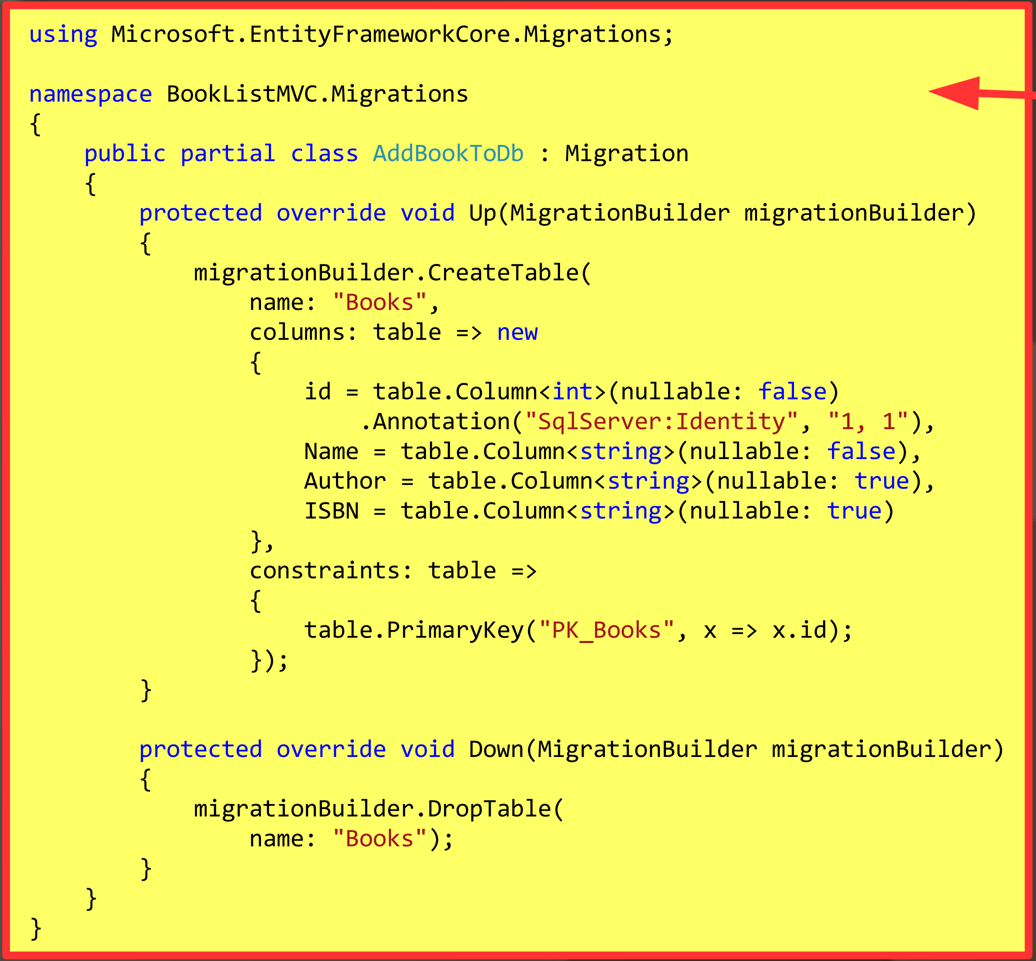
```
using Microsoft.EntityFrameworkCore.Migrations;

namespace BookListMVC.Migrations
{
    public partial class AddBookToDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Books",
                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),
                    ISBN = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Books", x => x.id);
                });
        }

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



```
Type 'get-help NuGet' to see all available NuGet co
PM> add-migration AddBookToDb
```



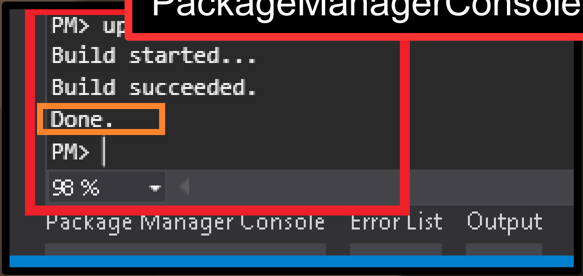
```
Solution 'BookListMVC' (1 of 1 project)
  BookListMVC
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      HomeController.cs
    Migrations
      20201105150018_AddBookToDb.cs
      ApplicationDbContextModelSnapshot.cs
    Models
      ApplicationDbContext.cs
      Book.cs
```



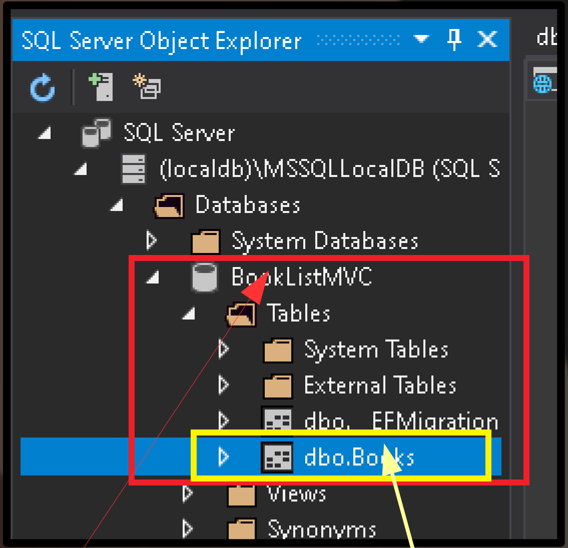
```
using Microsoft.EntityFrameworkCore.Migrations;

namespace BookListMVC.Migrations
{
    public partial class AddBookToDb : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Books",
                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),
                    ISBN = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Books", x => x.id);
                });
        }

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

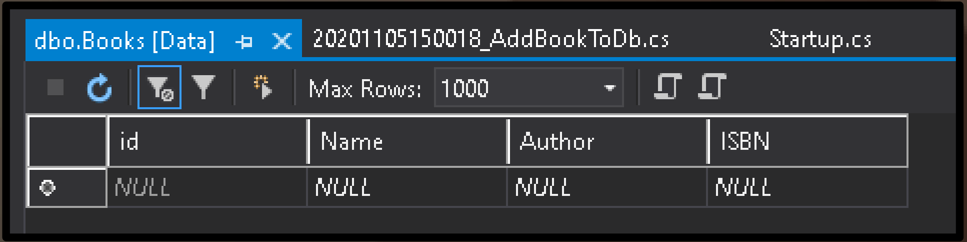
This is a script file
Next step is to execute this script
By typing the following command:
update-database

PackageManagerConsole



```
PM> up
Build started...
Build succeeded.
Done.
PM> |
```





id	Name	Author	ISBN
NULL	NULL	NULL	NULL

As you see update-database command have created our **BookListMVC** database.
Then it created a table named **Books**

Next step
Create BooksController (But first... Open _layout.cshtml)

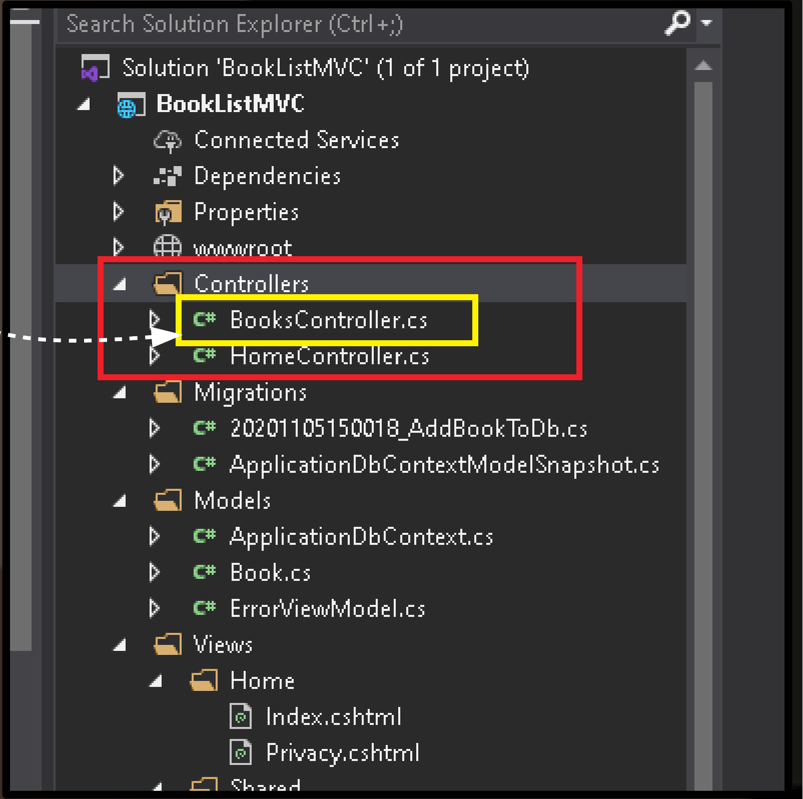
1. Open a master page called **_Layout.cshtml** .You will find it within **Views/Shared/** folder
   All we want is to add a Link to our BooksController.
2. Add another <li> object after a Privacy Link as follows:

_Layout.cshtml

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Books" asp-action="Index">Book List MVC</a>
</li>
```

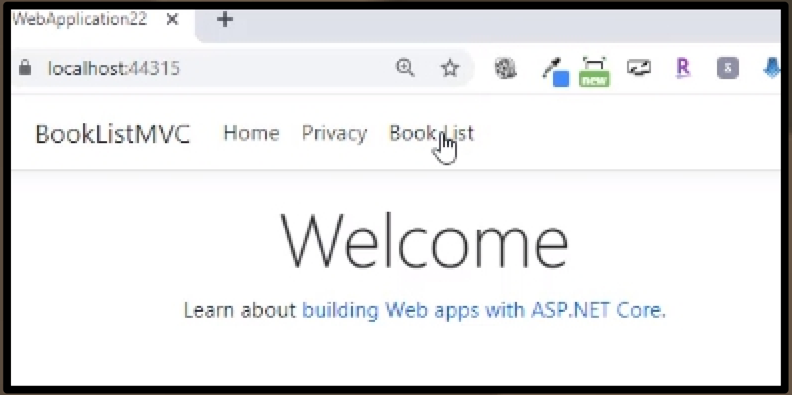
Finally add a new Controller inside Controllers folder

1. Add a new Controller inside the Controllers folder.
2. Choose empty controller, and name it as: **BooksController**
3. Run the application.

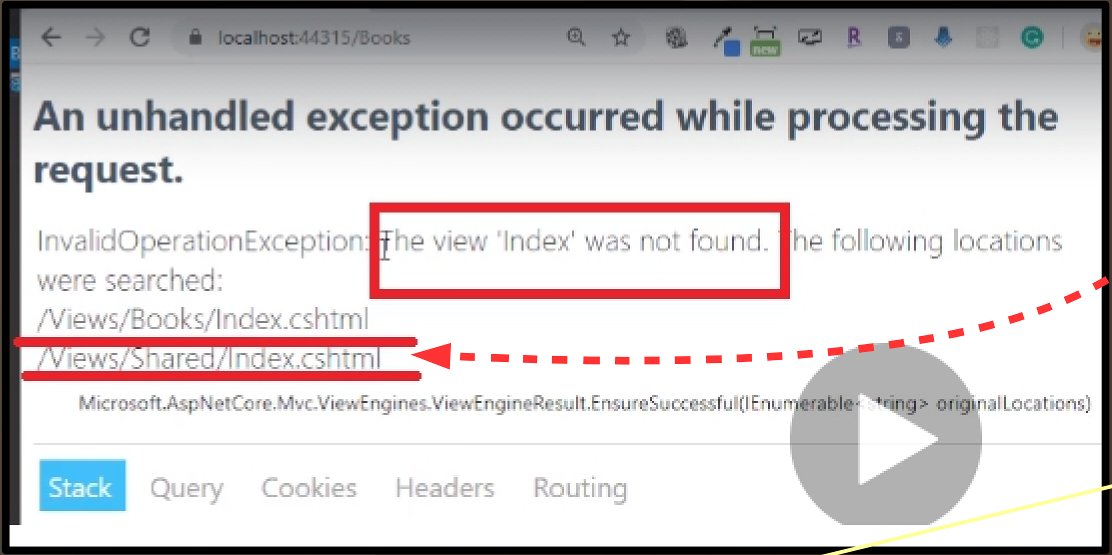


```
Search Solution Explorer (Ctrl+;)
Solution 'BookListMVC' (1 of 1 project)
◢ BookListMVC
    Connected Services
  ▷ Dependencies
  ▷ Properties
  ▷ wwwroot
  ◢ Controllers
    ▷ BooksController.cs
    ▷ HomeController.cs
  ◢ Migrations
    ▷ 20201105150018_AddBookToDb.cs
    ▷ ApplicationDbContextModelSnapshot.cs
  ◢ Models
    ▷ ApplicationDbContext.cs
    ▷ Book.cs
    ▷ ErrorViewModel.cs
  ◢ Views
    ◢ Home
        Index.cshtml
        Privacy.cshtml
    ◢ Shared
```

Try clicking on Book List link



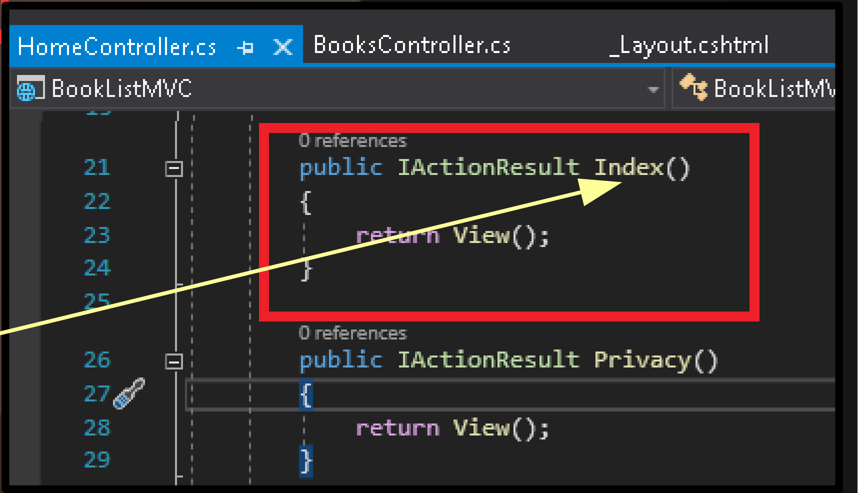
You will end up with the exception that says it cannot find Index.cshtml, that corresponds to BooksController **/Views/Books/Index.cshtml**
It is also looked inside Shared folder trying to find **Index.cshtml**



An unhandled exception occurred while processing the request.

InvalidOperationException: The view 'Index' was not found. The following locations were searched:
/Views/Books/Index.cshtml
/Views/Shared/Index.cshtml

Microsoft.AspNetCore.Mvc.ViewEngines.ViewEngineResult.EnsureSuccessful(IEnumerable<string> originalLocations)



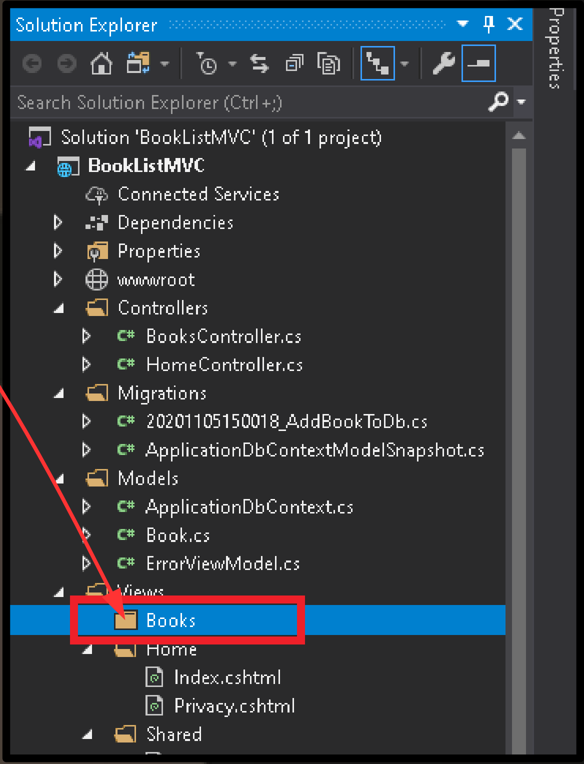
```
public IActionResult Index()
{
    return View();
}

public IActionResult Privacy()
{
    return View();
}
```

Even though we have an Index Action method, we still don't have a view

Adding a View

The Folder containing the view must be named as [yourname]controller. Example: we have a **BooksController.** We will need a corresponding **Books folder** inside **Views Folder.** Only then we will create a new view named Index.cshtml.

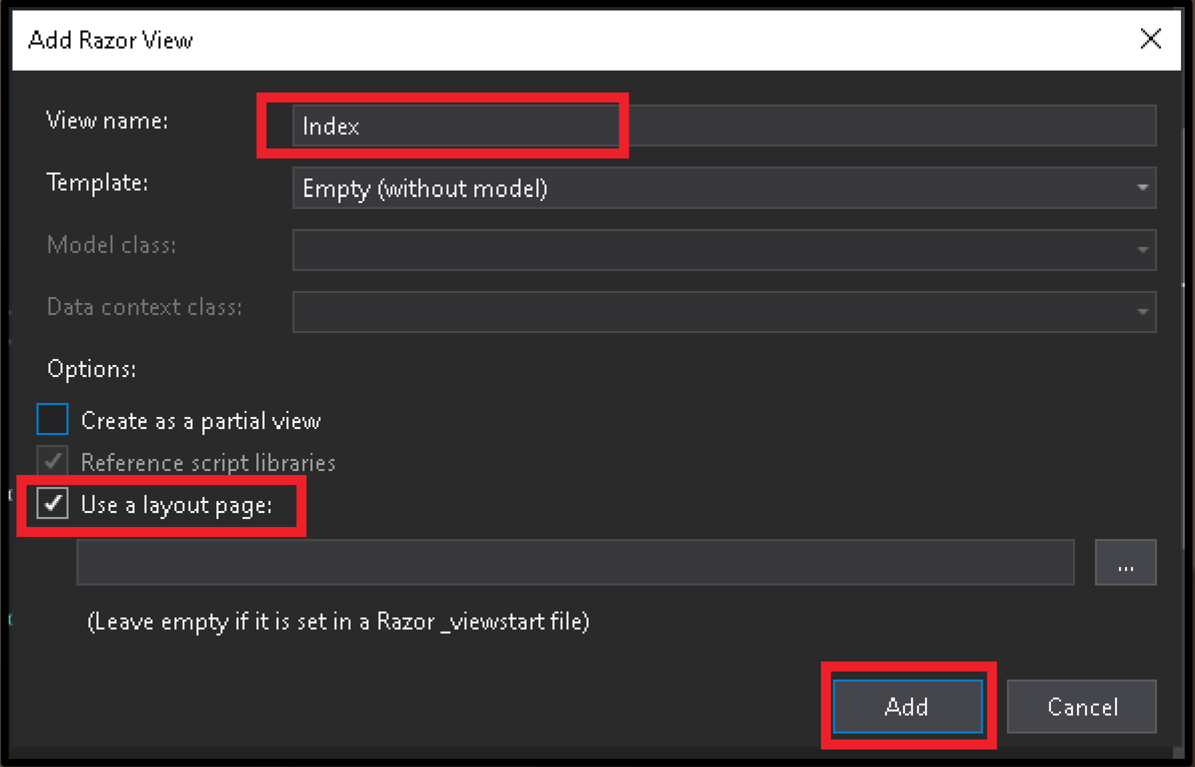
1. Open Views folder.
2. Create a new folder inside the Views folder, and name it **Books**
3. Create a new View by right clicking the Books folder then choose Add/View
4. Choose a Razor View second option(Not Empty) and click Add.
5. Name your Page As Index. Template (Empty)
6. Choose use layout Page, and click Add.



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'BookListMVC' (1 of 1 project)
  BookListMVC
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      BooksController.cs
      HomeController.cs
    Migrations
      20201105150018_AddBookToDb.cs
      ApplicationDbContextModelSnapshot.cs
    Models
      ApplicationDbContext.cs
      Book.cs
      ErrorViewModel.cs
    Views
      Books
      Home
        Index.cshtml
        Privacy.cshtml
      Shared
```



Add Razor View

View name: Index

Template: Empty (without model)

Model class:

Data context class:

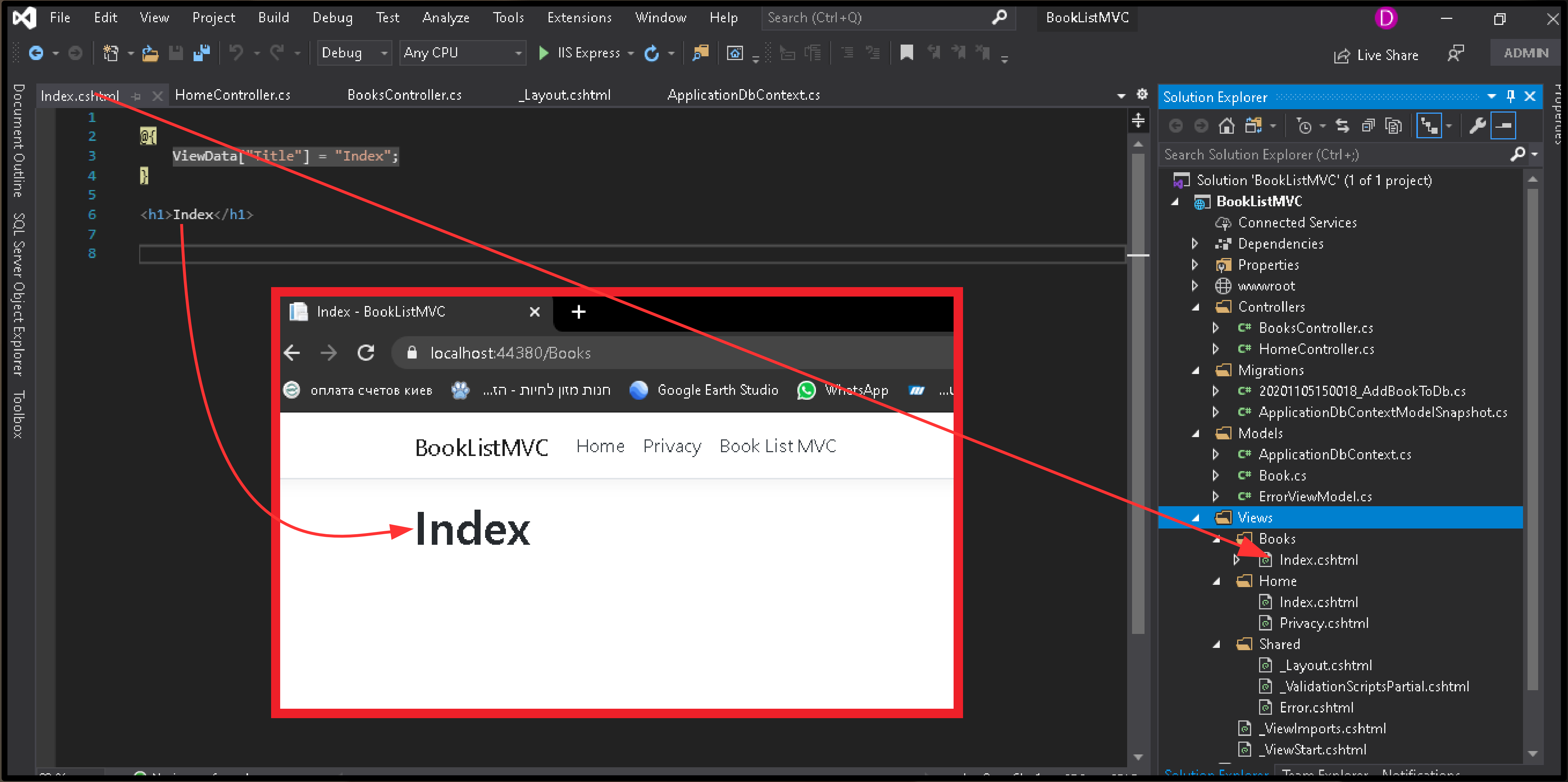
Options:

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

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

7. Run the application.

8. As you can see now, by running the application we will receive word: index, as it appears inside Views/Books/Index.cshtml.
You can see how the Index() Action fired Index view.



```
@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>
```

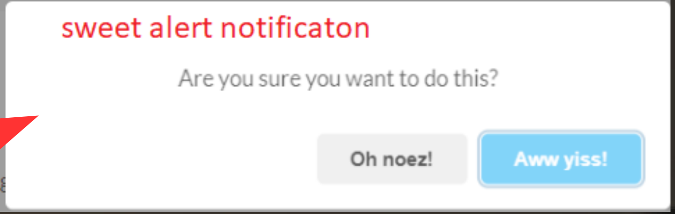
localhost:44380/Books

BookListMVC Home Privacy Book List MVC

Index

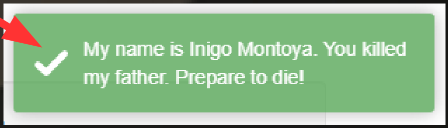
Third-Party APIs

I'd like to install a few of the third party tools
1. Sweet Alert. https://sweetalert.js.org/guides/ Will give us Nice Alert
2. Toastr.js https://codeseven.github.io/toastr/ Will give us Nice Toaster Notifications.
3. Datatables https://datatables.net/ Will give us a nice UI table with advanced features



sweet alert notificaton

Are you sure you want to do this?



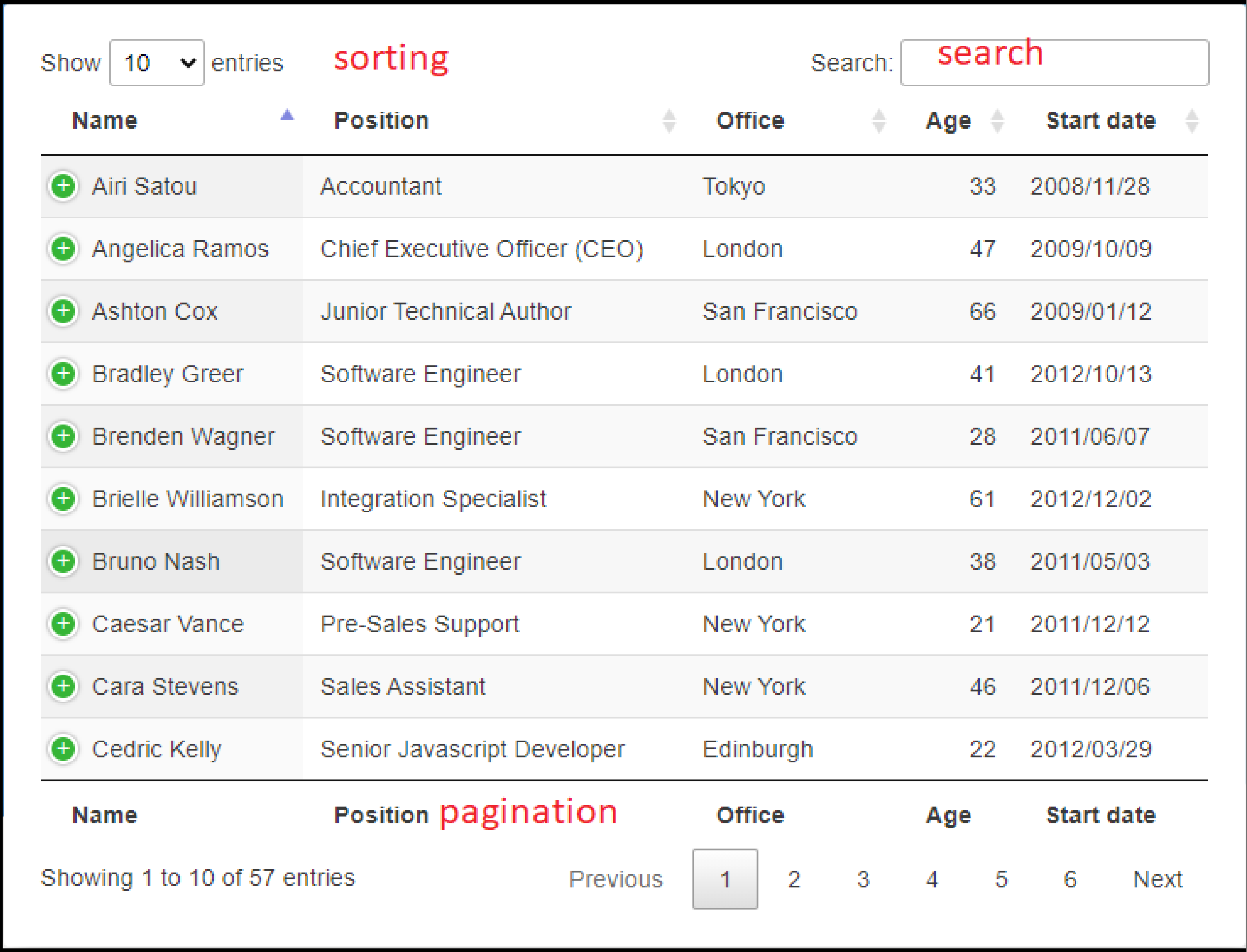
My name is Inigo Montoya. You killed my father. Prepare to die!

All we trying to achieve here is to retrieve a book list from a database,and display these books in a "beautiful shape" (Advanced Grid). We will retrieve this list of books in a Json format, (we will do this in BooksController) Then we will be passing Json file to our Data-Table API. Our java-Script file will contain methods from the API which can retrieve the Json file from BooksController and render needed html tables. All wee need is to setup a general table inside our view, and the API will render the rest of the<td>,<tr> elements for us depending on our data.
For mor information please visit DataTables.net
DataTables it is a JavaScript API. But How it works?
1.First you include a reference to two CDN files in your _layout page.
One for CSS in <header>
One for JS in <scripts>
2. Create a Javascript file, and call This function :

```
$(document).ready( function () {
    $('#myTable').DataTable();
} );
```

3. Create a table with the id of myTable and at least one must have <thead> html tag.The Api will render the missing html tags for you.

4. You can decide how to render html using Json format. See booklist.js file in the upcoming pages



Show 10 entries sorting Search: search

Name	Position	Office	Age	Start date
Airi Satou	Accountant	Tokyo	33	2008/11/28
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12
Bradley Greer	Software Engineer	London	41	2012/10/13
Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07
Brielle Williamson	Integration Specialist	New York	61	2012/12/02
Bruno Nash	Software Engineer	London	38	2011/05/03
Caesar Vance	Pre-Sales Support	New York	21	2011/12/12
Cara Stevens	Sales Assistant	New York	46	2011/12/06
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29
Name	Position	Office	Age	Start date

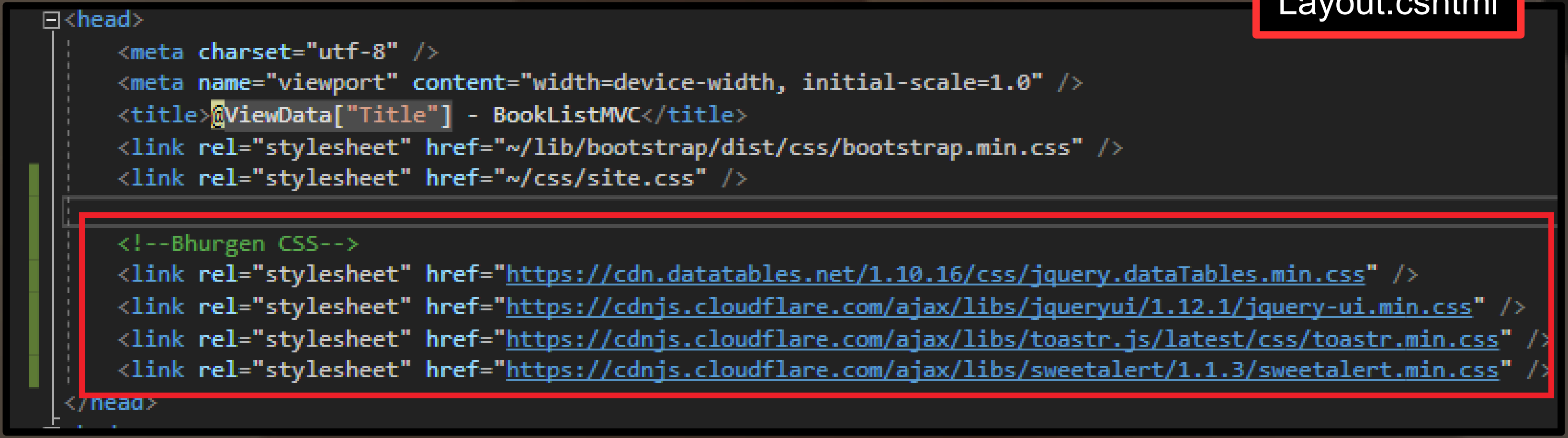
pagination

Showing 1 to 10 of 57 entries

Int this project We will be using 3 different APIs
Sweet Alerts, Toatsr.js, and DataTables

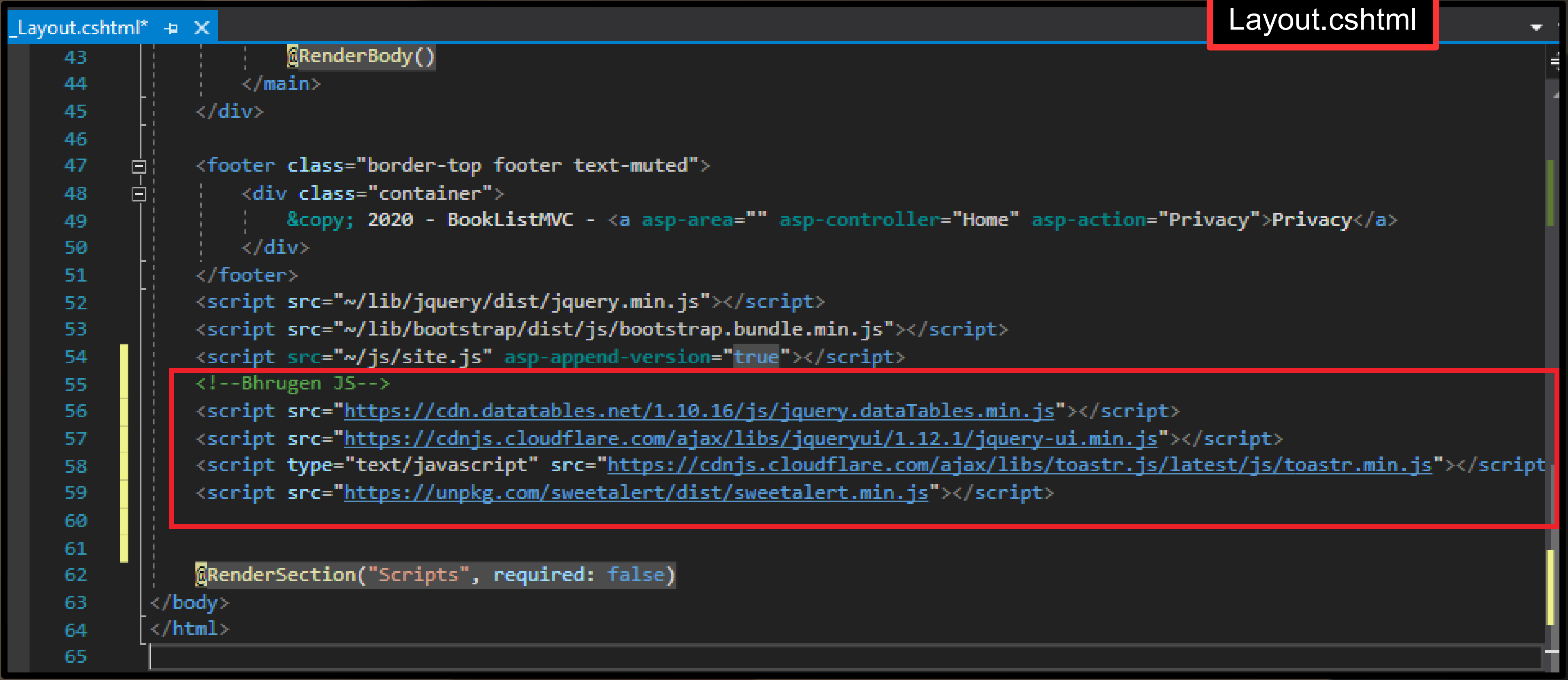
```
CSS:
<link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.css" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css"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

JS:
<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>
```

1. The CSS section will be added in Master Page named _Layout.cshtml (inside Views/Shared folder)
2. Add the css section within <head></head> section of the _Layout page.


Layout.cshtml

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - BookListMVC</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />

    <!--Bhurgen 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" />
</head>
```

3. Add The JS section at the same page but at the very bottom of the html just before </body> and</html> Closing tags.



```
                @RenderBody()
            </main>
        </div>

        <footer class="border-top footer text-muted">
            <div class="container">
                &copy; 2020 - BookListMVC - <a asp-area="" asp-controller="Home" asp-action="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>
</html>
```

Importan note from author of this guide.
Dear student from this point it is very easy to make a typing mistake, and spend hours trying to find a problem. I advise you to download a working solution of this project from Bhrugen Patel Official repository,https://github.com/bhrugen/BookListMVC and have a reference to the working code. The APIs is very sensitive, and every little typo can cause very annoying problems.
I appreciate your time, and don't want you to spend 2 days as I did trying to find a small typo in html, o Json file. The author adds a code little by little to the same BooksController.cs file. That is why it is very easy to make a mistake. So remember to download a working solution just in case.

BookList JS and API Calls

BooksController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListMVC.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BookListMVC.Controllers
{
    public class BooksController : Controller
    {
        //Adding ApplicationDB Context
        private readonly ApplicationDbContext _db;

        //Adding constructor
        //Getting a DbContext using Dependency injection.
        public BooksController(ApplicationDbContext db)
        {
            _db = db;

        }

        public IActionResult Index()
        {
            return View();
        }

        //API Calls
        #region API Calls
        [HttpGet]
        //Very important to make it async. Otherwise the data will not be displayed!!!!
        public async Task<IActionResult> GetAll()
        {
            return Json(new { data = await _db.Books.ToListAsync() });
        }

        [HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
            var bookFromDb = await _db.Books.FirstOrDefaultAsync(u => u.id == id);
            if (bookFromDb == null)
            {
                return Json(new { success = false, message = "Error while Deleting" });
            }
            _db.Books.Remove(bookFromDb);
            await _db.SaveChangesAsync();
            return Json(new { success = true, message = "Delete successful" });
        }
        #endregion
    }
}
```

1.Open BooksController, and paste the below code

First we created a readonly ApplicationDbContext _db variable. Then we added a constructor Where we connecting this _db context to the base class.
ApplicationDbContext: inherits from DbContext which is a class inside `Assembly Microsoft.EntityFrameworkCore`
So a DbCo0ntext class is our base class.

Every time when a BooksController object is created, it will pass the ApplicationDBContext variable as a parameter to its constructor.

As you can see the API Calls located inside #region element
GetAll(), and Delete() methods are still the same as in BookListRazor project we worked before.

We will be using the Index View method as a container for the API Calls. The IActionResult() method will receive information only from the API Calls.
We will be loading this view using theDataTables API's DataTable() function (inside booklist.js)

2. Proceed to the next page.

BookList.js

3. Open wwwroot/js/ folder
4. Add a new booklist.js to it. Right click,selectAdd/NewItem/JavaScript file
5. The Javascript Will be the same as in BookListRazor project, but with slightly different modifications.
6.Paste the following code inside your newly created bookList.js
7.All we have to do now is Add a new Table inside/Books/Index.cshtml and call this JavaScript for the index.cshtml We will do this in the next page.
Do not make a typo here. This is very important!

```
var dataTable;

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

function loadDataTable() {
    dataTable = $('#DT_load').DataTable({
        "ajax": {
            "url": "/books/getall/",
            "type": "GET",
            "datatype": "json"
        },
        "columns": [
            { "data": "name", "width": "20%" },
            { "data": "author", "width": "20%" },
            { "data": "isbn", "width": "20%" },
            {
                "data": "id",
                "render": function (data) {
                    return `<div class="text-center">
                        <a href="/BookList/Edit?id=${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);
                    }
                }
            });
        }
    });
}
```

Remember
data is= book.id

As you see when page is ready we calinng a loadDataTable() function. This function then
Calls the API's internal function called DataTable(). This function recieves Ajax parameters, that calls /books/getall/ which is Controller/method call.
Later on we will create a method named GetAll()
Inside BooksController.cs file.
After that, we create columns in JSON format for the name, author, and ISBN. This is a blueprint for our table. We will be passing this blueprint to render function. The render function will render the Html Based on this blueprint.
"Flip the screen" and think of this as a real column to get the idea the easy way!!!!
Every column has a width%, name, and data. Where data Is the id of the book. data variable will pass the id of each book, and the corresponding name, author, and ISBN will be rendered based on this id.
Take a look at this table. Where data is the id of the book.

"data"	"data"	"data"
"name"	"author"	"isbn"
Edit/Delete	Edit/Delete	Edit/ Delete

Is the same as.

id :	id :	id :
BooksID.name	BooksId.author	Books.id.isbn

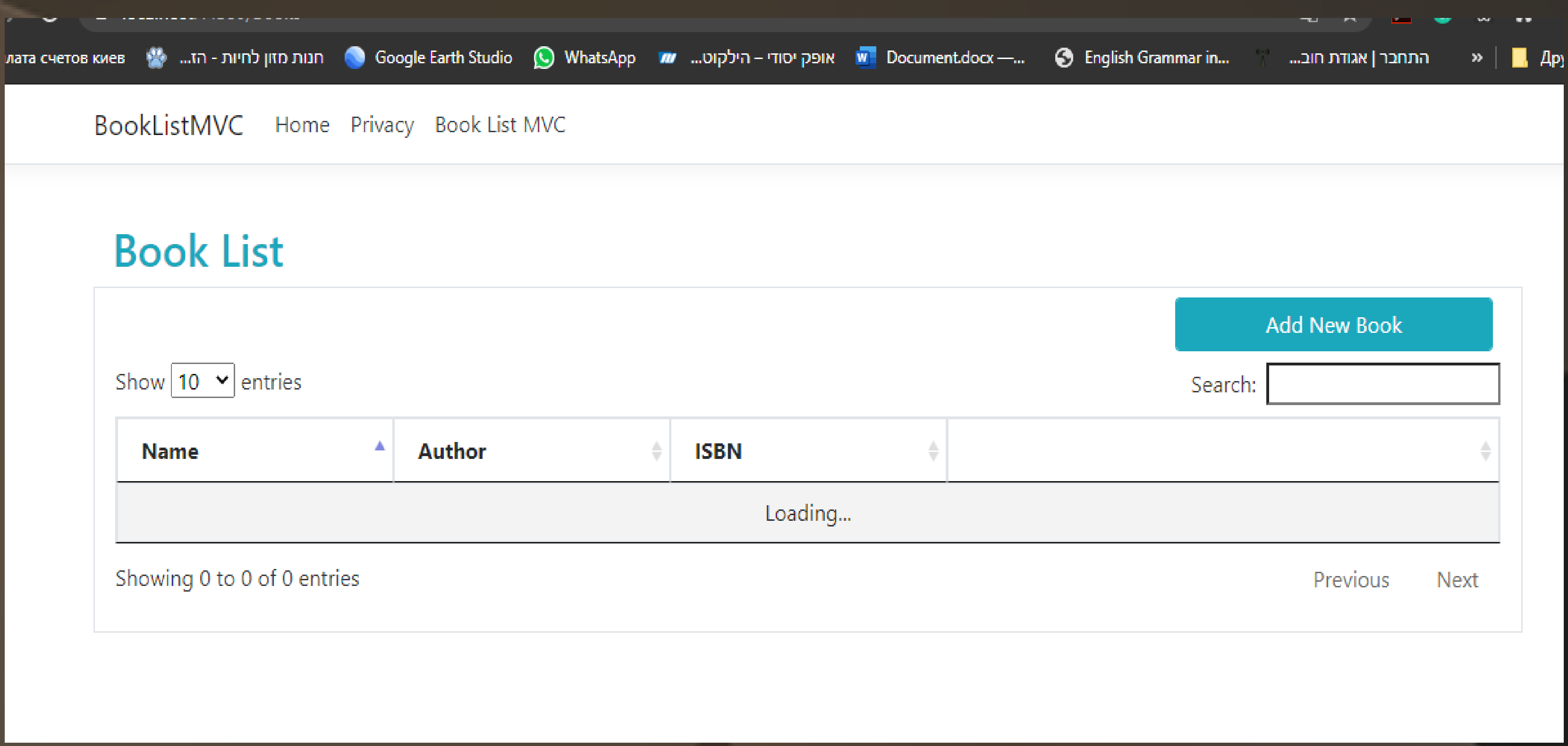
Now it make sence right?
Then we setup API's **render** function passing the Json **data** parameter
Everything from this line of code will be rendered by DataTables API
We wrap all of this into a <div> tag. And creating two buttons: One for Create, and one for Delete. We will pass the data id to each of this

1.Open Books/Index.cshtml file, and add the following code to the view:
We don't need the models inside this page, because we will be using to load everything using DataTables.

Index.cshtml

```
<br />
<div class="container row p-0 m-0">
    <div class="col-6">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-3 offset-3">
        <a asp-action="Upsert" asp-controller="Books" class="btn btn-info form-control text-white">
            Add New Book
        </a>
    </div>
    <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>
}
```

2. Run the application and check how things look.


BookListMVC Home Privacy Book List MVC

Book List

Add New Book

Show 10 entries Search:

Name	Author	ISBN	
Loading...			

Showing 0 to 0 of 0 entries Previous Next

By clicking the Book List MVC we get this nice Table. But there is nothing in there.
This is because we have not created any book Yet.
If you try to click the Create new book button nothing will happen, because we have not added Action Method, or view For Adding a new book. Let's add this functionality in the next page.

Creating
Upsert Get Action Method + View

Upsert view will be used for two places:
1. To Add a Book.
2. To Create a Book.

Based on that it will sometime retrieve an id if it is for edit, but if it is for create, there won't be any id passing.

3.Open BooksController.cs file and add the following code:

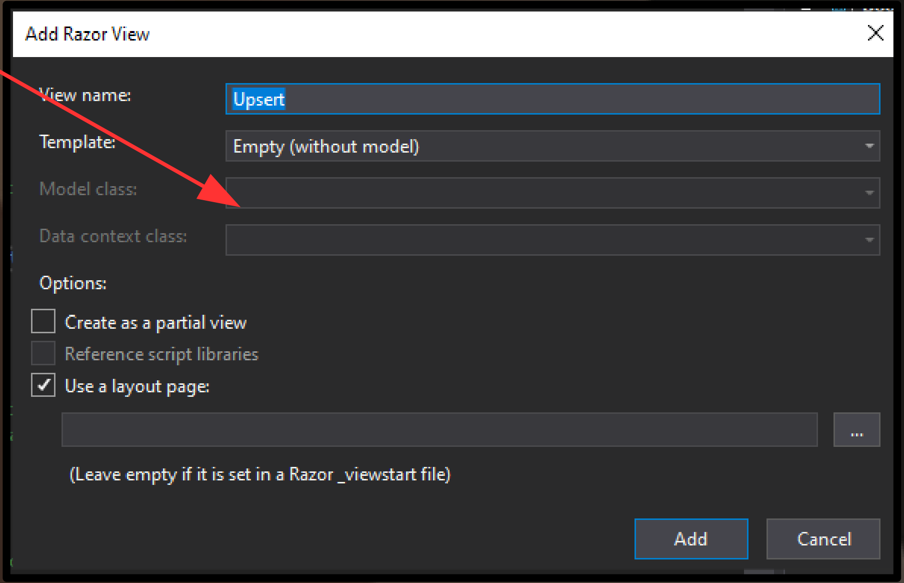
We will be adding more code to this file later on.
If you have any trouble refer to the GitHub's source code
To download a complete code

BookListController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListMVC.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BookListMVC.Controllers
{
    public class BooksController : Controller
    {
        private readonly ApplicationDbContext _db;
        [BindProperty]
        public Book Book { get; set; }
        public BooksController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Upsert(int? id)
        {
            Book = new Book();
            if (id == null)
            {
                //create
                return View(Book);
            }
            //update
            Book = _db.Books.FirstOrDefault(u => u.Id == id);
            if (Book == null)
            {
                return NotFound();
            }
            return View(Book);
        }

 #region API Calls
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            return Json(new { data = await _db.Books.ToListAsync() });
        }

        [HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
            var bookFromDb = await _db.Books.FirstOrDefaultAsync(u => u.Id == id);
            if (bookFromDb == null)
            {
                return Json(new { success = false, message = "Error while Deleting" });
            }
            _db.Books.Remove(bookFromDb);
            await _db.SaveChangesAsync();
            return Json(new { success = true, message = "Delete successful" });
        }
        #endregion
    }
}
```

Next – Create the Upsert View

1. Right Click the Upsert() method and select add View
2. Select Razor View, then Name it as Upsert and click Add.
3 Proceed to the next page.


Add Razor View

View name: Upsert

Template: Empty (without model)

Model class:

Data context class:

Options:

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

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

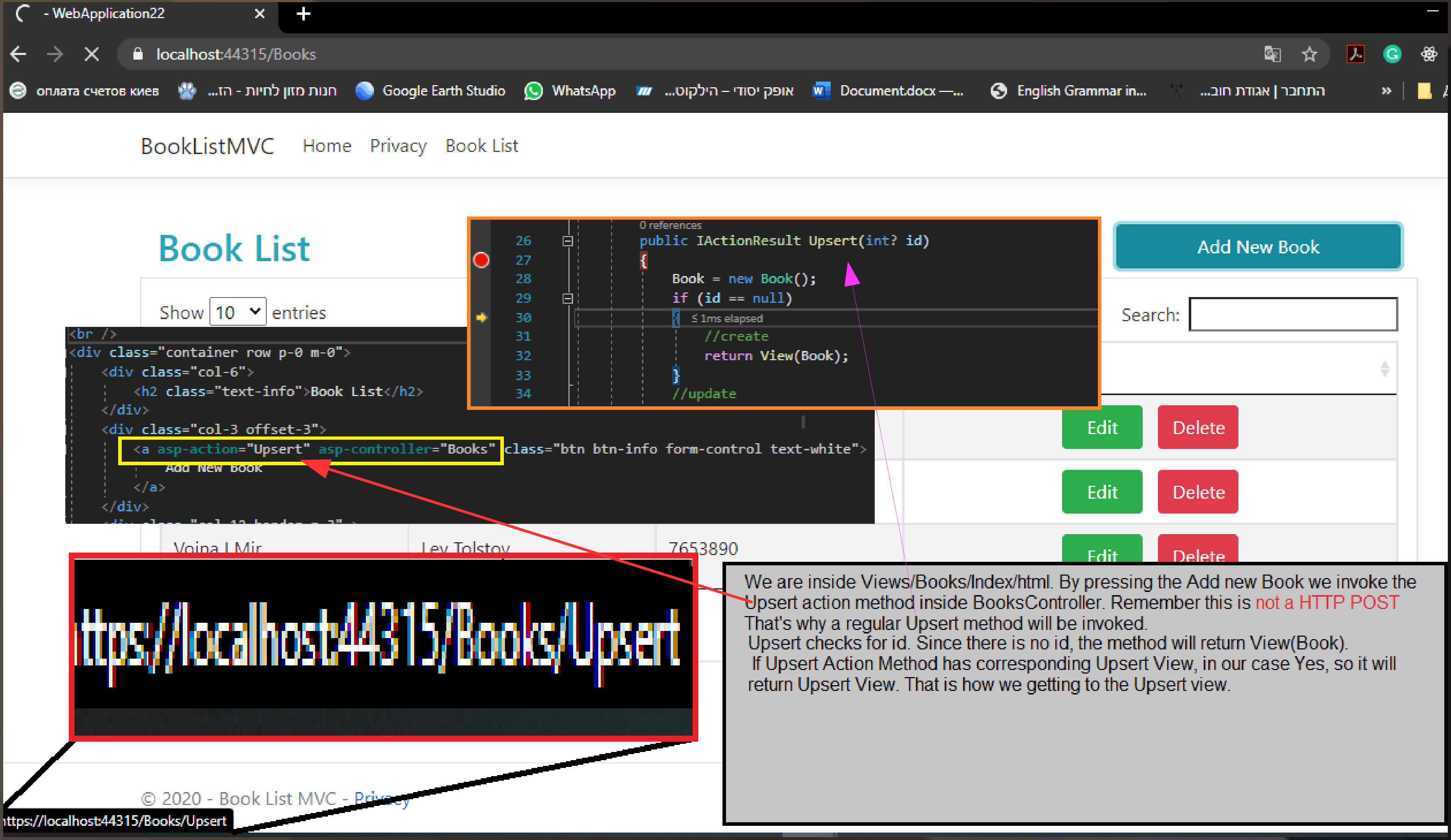
Upsert.cshtml

4. First, define a @model of the Book in the Upsert View
5. Copy the following code:
6. Run the appliaction.

```
@model BookListMVC.Models.Book

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

<div class="border container" style="padding:30px;">
    <form method="post">
        @if (Model.Id != 0)
        {
            <input type="hidden" asp-for="Id" />
        }
        <div class="text-danger" asp-validation-summary="ModelOnly"></div>
        <div class="form-group row">
            <div class="col-3">
                <label asp-for="Name"></label>
            </div>
            <div class="col-6">
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group row">
            <div class="col-3">
                <label asp-for="Author"></label>
            </div>
            <div class="col-6">
                <input asp-for="Author" class="form-control" />
                <span asp-validation-for="Author" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group row">
            <div class="col-3">
                <label asp-for="ISBN"></label>
            </div>
            <div class="col-6">
                <input asp-for="ISBN" class="form-control" />
                <span asp-validation-for="ISBN" class="text-danger"></span>
            </div>
        </div>
        <div class="form-group row">
            <div class="col-3 offset-3">
                <button type="submit" class="btn btn-primary form-control">
                    @(Model.Id != 0 ? "Update" : "Create")
                </button>
            </div>
            <div class="col-3">
                <a asp-action="Index" class="btn btn-success form-control">Back to List</a>
            </div>
        </div>
    </form>
</div>

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

Here we define a simple
Html page. Using Ternary operator
Read more about ternary operator_
Here
The title will be based on model.id
If Model has id, the title will be Create,
Otherwise the title will be Edit.

Then we create a form which will have
a hidden id property.
If a model does not equal to zero,
place the id inside the hidden
property, so the `public IActionResult Upsert(int? id)` method could take
this id as a parameter. You can see
that this is a nullable parameter.
If Model has an id, it will be passed to
this method.



```
0 references
public IActionResult Upsert(int? id)
{
    Book = new Book();
    if (id == null)
    {  ≤ 1ms elapsed
        //create
        return View(Book);
    }
    //update
```

```
<br />
<div class="container row p-0 m-0">
    <div class="col-6">
        <h2 class="text-info">Book List</h2>
    </div>
    <div class="col-3 offset-3">
        <a asp-action="Upsert" asp-controller="Books" class="btn btn-info form-control text-white">
            Add New Book
        </a>
    </div>
```

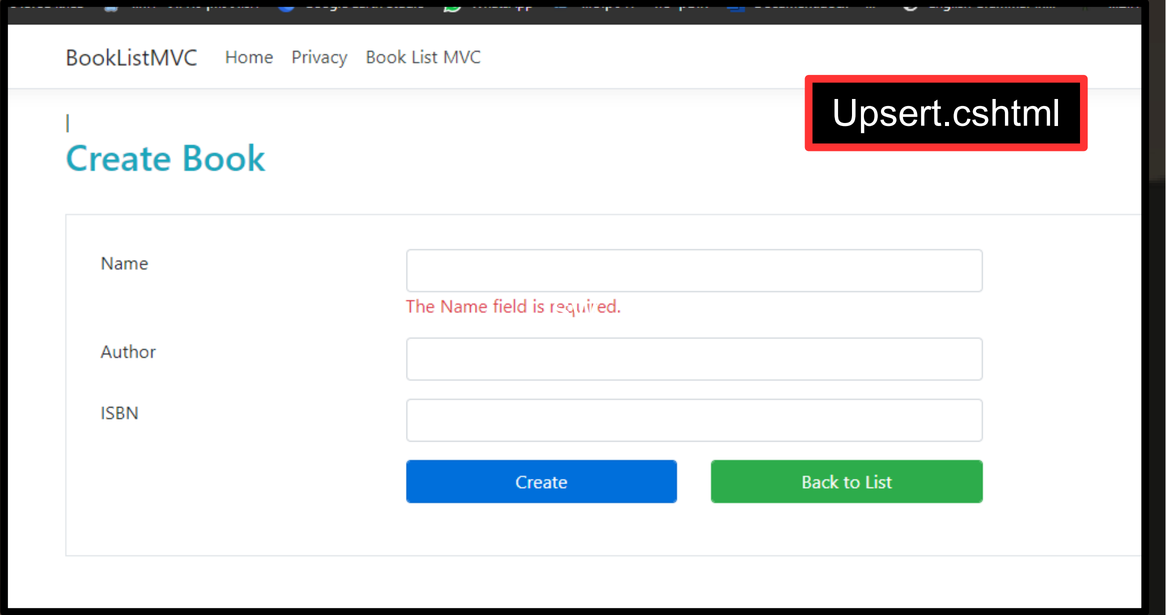
We are inside Views/Books/Index/html. By pressing the Add new Book we invoke the Upsert action method inside BooksController. Remember this is not a HTTP POST That's why a regular Upsert method will be invoked.
Upsert checks for id. Since there is no id, the method will return View(Book).
If Upsert Action Method has corresponding Upsert View, in our case Yes, so it will return Upsert View. That is how we getting to the Upsert view.

1. Go Back to Solution explorer,and open Upsert.cshtml
2.Add the following code to the very end of the page to Enable the validations.

Upsert.cshtml

```
@section Scripts{
    <script name="_ValidationScriptsPartial"></script>
}
```

3.Once you click Create button the validation should fire.
But if you try to create the book nothing will happens.
This is because we have not configured the post action method for Upstert.
Let's create one.


Upsert.cshtml

BookListMVC Home Privacy Book List MVC

Create Book

Name

The Name field is required.

Author

ISBN

Create Back to List

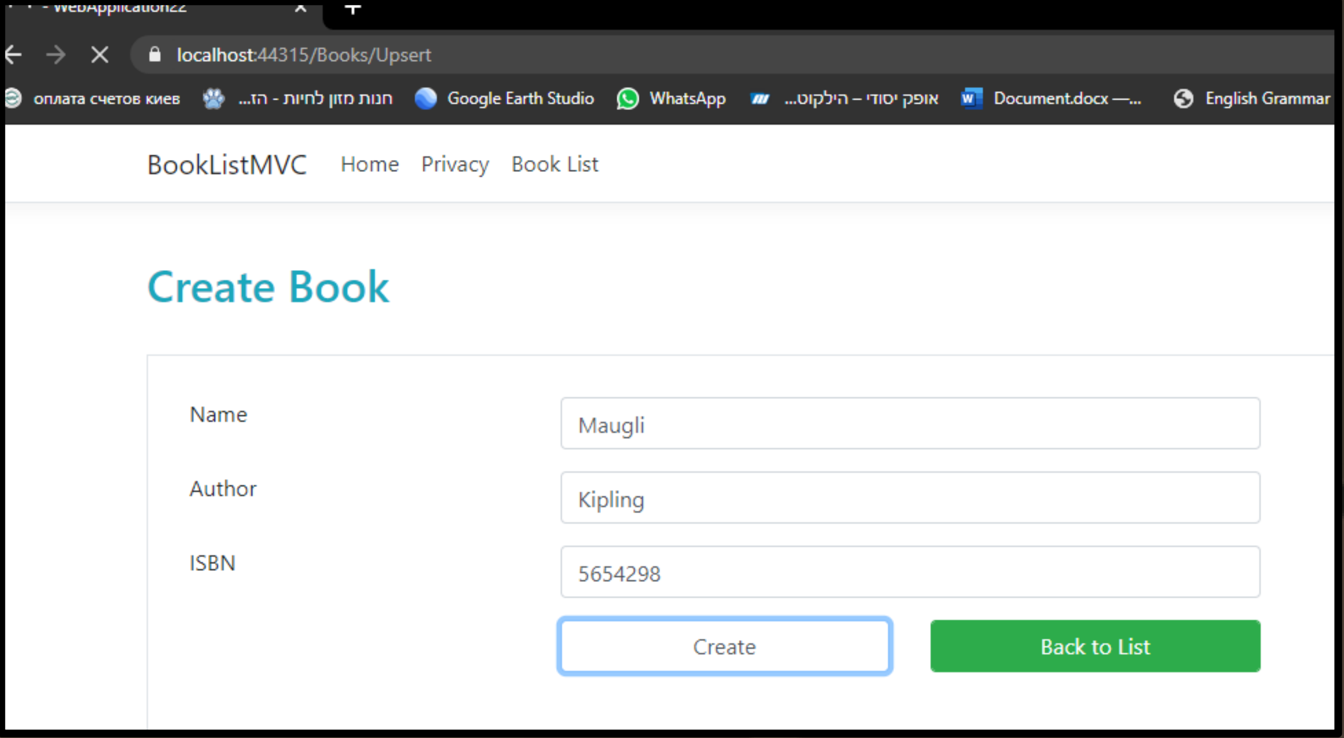
Upsert Post and Delete

1.Open BooksController, and add Upsert() Post action method to it right after th first Upsert() action method.
Don't forget the [httpPost] attribute

BooksController

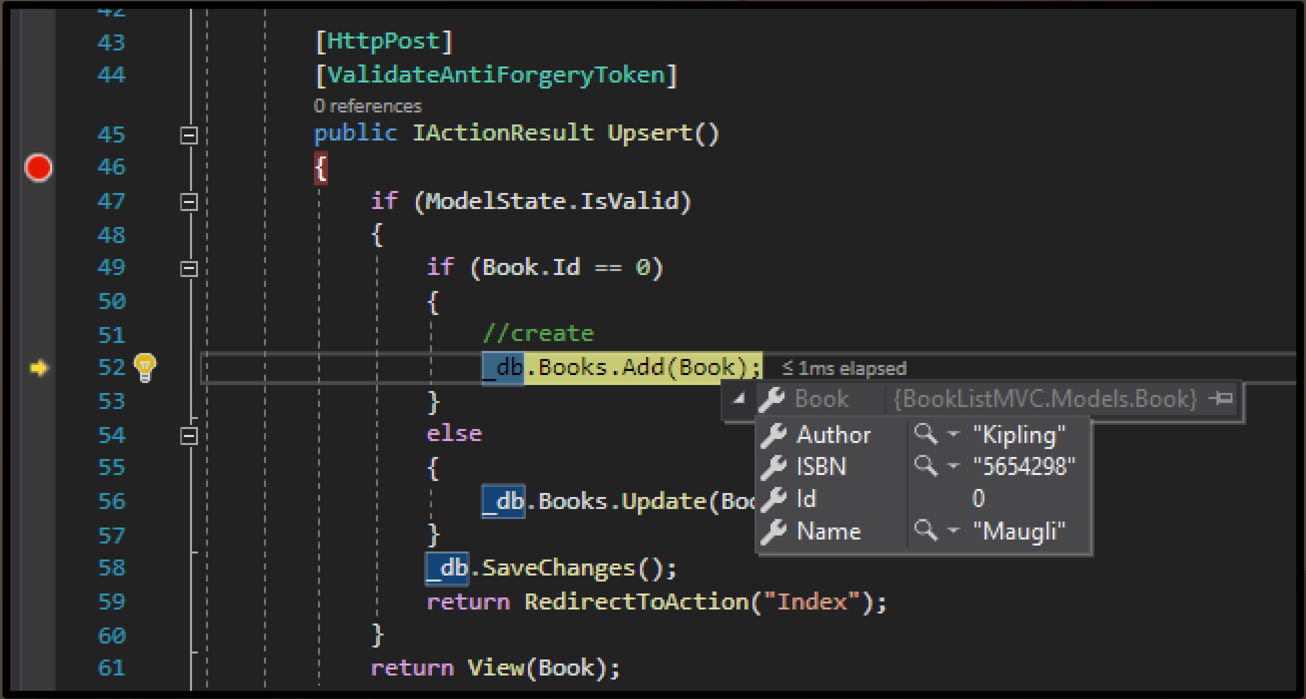
```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Upsert()
        {
            if (ModelState.IsValid)
            {
                if (Book.Id == 0)
                {
                    //create
                    _db.Books.Add(Book);
                }
                else
                {
                    _db.Books.Update(Book);
                }
                _db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(Book);
        }
```

2. Set a break point on this method And run the application. When you Create a new book debugger should End the execution in this HttpPost action method. See the next page.



Create Book

Name	Maugli
Author	Kipling
ISBN	5654298



```
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public IActionResult Upsert()
        {
            if (ModelState.IsValid)
            {
                if (Book.Id == 0)
                {
                    //create
                    _db.Books.Add(Book);  ≤1ms elapsed
                }
                else
                {
                    _db.Books.Update(Boo
                }
                _db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(Book);
```

```
Book    {BookListMVC.Models.Book}
  Author   "Kipling"
  ISBN     "5654298"
  Id       0
  Name     "Maugli"
```

As you can see by pressing the Create button we Invoke httpPost upsert() Action method. Then we check if the book's id is equal to zero. In Our case it is true because we came to Upsert View by pressing Create new book button. We are adding a new book to _db context object. Then we update the book inside the database, and finally saving changes. At the end we redirecting to index html.

But If we were pressing Edit button istead, then we would be redirected to Upsert view with the pressed Edit-book's id. Then this id would be passed to Upsert.cshtml hidden property remember?

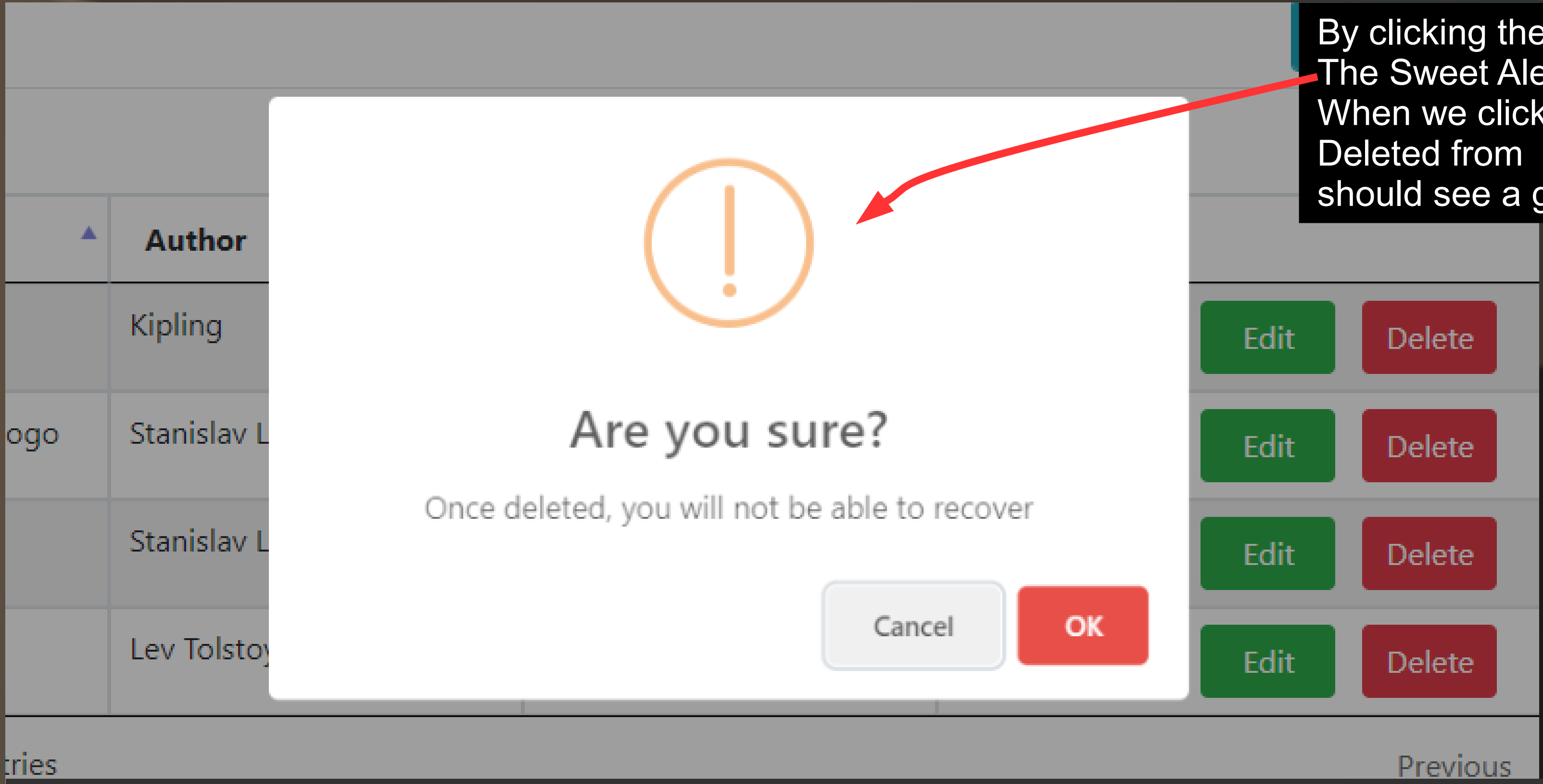
Use break points to understand the code flow. It really helps.

BooksController

Here is the full code of BooksController.cs

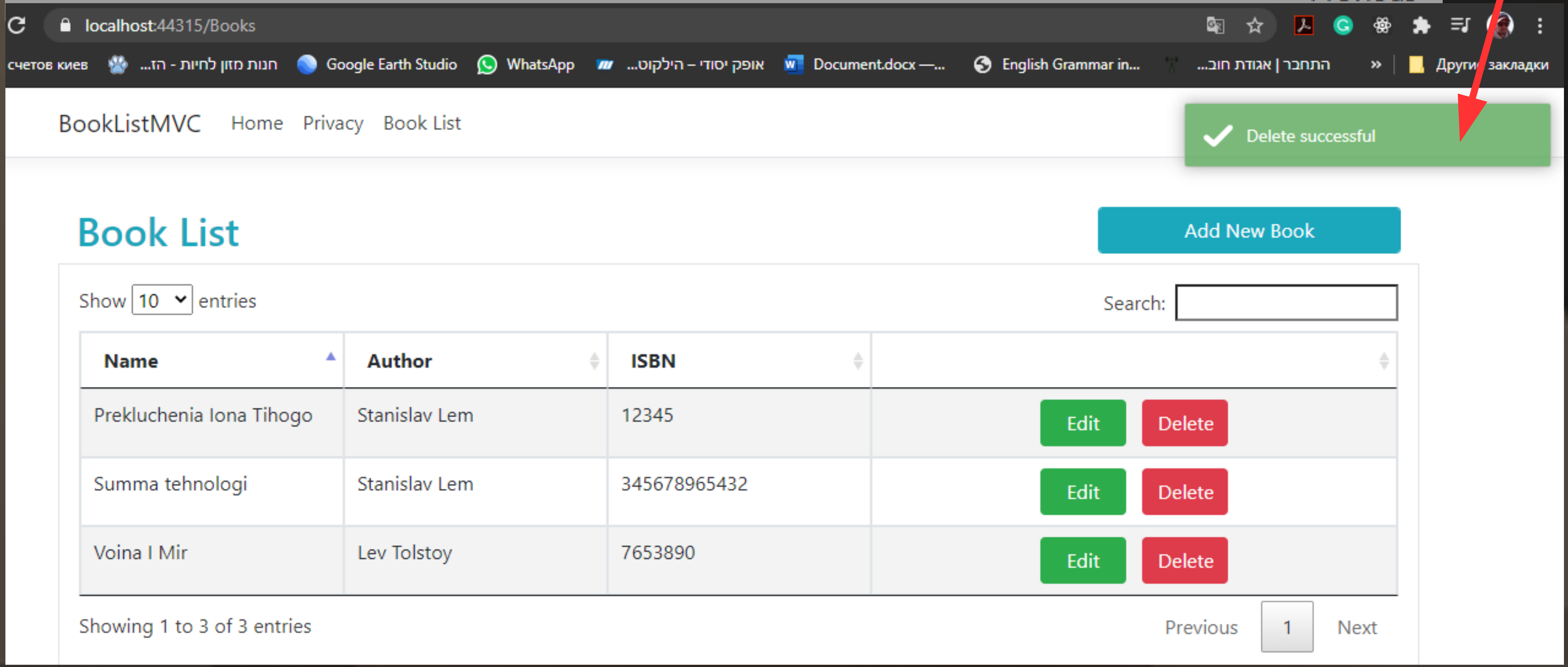
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BookListMVC.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace BookListMVC.Controllers
{
    public class BooksController : Controller
    {
        private readonly ApplicationDbContext _db;
        [BindProperty]
        public Book Book { get; set; }
        public BooksController(ApplicationDbContext db)
        {
            _db = db;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Upsert(int? id)
        {
            Book = new Book();
            if (id == null)
            {
                //create
                return View(Book);
            }
            //update
            Book = _db.Books.FirstOrDefault(u => u.Id == id);
            if (Book == null)
            {
                return NotFound();
            }
            return View(Book);
        }
        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Upsert()
        {
            if (ModelState.IsValid)
            {
                if (Book.Id == 0)
                {
                    //create
                    _db.Books.Add(Book);
                }
                else
                {
                    _db.Books.Update(Book);
                }
                _db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(Book);
        }
        #region API Calls
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            return Json(new { data = await _db.Books.ToListAsync() });
        }

        [HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
            var bookFromDb = await _db.Books.FirstOrDefaultAsync(u => u.Id == id);
            if (bookFromDb == null)
            {
                return Json(new { success = false, message = "Error while Deleting" });
            }
            _db.Books.Remove(bookFromDb);
            await _db.SaveChangesAsync();
            return Json(new { success = true, message = "Delete successful" });
        }
        #endregion
    }
}
```

By clicking the delete button we will get
The Sweet Alert notification.
When we click Ok button the book will be
Deleted from a database, and you
should see a green notification



Are you sure?

Once deleted, you will not be able to recover



Delete successful

Book List

Name	Author	ISBN	
Prekluchenia Iona Tihogo	Stanislav Lem	12345	Edit Delete
Summa tehnologi	Stanislav Lem	345678965432	Edit Delete
Voina I Mir	Lev Tolstoy	7653890	Edit Delete

Showing 1 to 3 of 3 entries

You can do a lot withASP.NET CORE
But this was only the tip of the iceberg
Please visit Bhrugen Patel at
http://bhrugen.com
or
https://www.dotnetmastery.com/
For more outstanding courses.