ASP.NET TUTORIALS

INTRODUCTION TO ASP.NET CORE MVC

What is .NET:

.NET is a free, cross-platform, open source developer platform for building many different types of applications. With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, IOT, and more.

What is ASP.NET:

ASP.NET is a free web framework for building great websites and web applications using HTML, CSS, and JavaScript. You can also create Web APIs and use real-time technologies like Web Sockets. ASP.NET Core is an alternative to ASP.NET

OR

ASP.NET extends the .NET developer platform with tools and libraries specifically for building web apps.

Introduction to ASP.Net Core:

- Nowadays, everyone is talking about open source, cross platform development. Microsoft has always been known for its Windows based products, but now we are in the new age of development. For this, a new revolutionary product came into the market, which is Microsoft .NET Core.
- NET Core is a free open source and cross platform framework, created for building modern Cloud based Applications and every .NET developer feels proud of it. Now, there are no boundaries for the platform. Now, every .NET developer can say, yes I am platform independent, I am using an open source.
- In the revolution of software development, Microsoft launched its first .NET framework in the year 2000 with its first .NET framework 1.0. This

framework plays a major role in the field of software development. People love Microsoft technology products, because these products are easy to use and easy to learn.

- In the year 2016, Microsoft has come up with a new revolution, Microsoft .NET Core 1.0. People always love ASP.NET, because it's working over WorldWideWeb.
- ASP.NET Core is a new open source, cross platform framework to create modern Web based Cloud based systems, which means, now you are not only working for Windows, you can run in Linux, Mac; i.e., wherever you want.

OPEN SOURCE MEANS:

- In general, open source refers to any program whose source code is made available for use or modification as users or other developers see fit. Open source software is usually developed as a public collaboration and made freely available.
- The software being distributed must be redistributed to anyone else without any restriction.
- The source code must be made available (so that the receiving party will be able to improve or modify it).

ASP.NET Core Features:

- Cross platform, open source now runs your app over Linux, Windows, Mac; i.e., wherever you want.
- Fast Development- fast work over the Browsers.
- Work in your editors - now you can work not only in Visual Studio. You can also choose Visual Studio code.

.NET CORE Features:

- ASP.NET Core MVC
- ASP.Net Core API
- IOT
- Universal Windows App.

Difference:

.NET FRAMEWORK	.NET CORE FRAMEWORK
OLD Framework	New Framework
Not Open Source	Open Source
ONLY FOR WINDOWS PLATFORM	CROSS PLATFORM (WINDOWS, LINUX, MAC)
ASP.NET	**ASP.NET CORE**
Targets .Net framework	Targets .Net core framework
All components are available like asp.Net MVC, web API, asp.Net web Forms etc.	Asp.Net web forms are not there in asp.Net core, we have asp.Net MVC core and web API in .Net core.

History of ASP.NET:

- ASP.NET has been used from many years to develop web applications. Since then, the framework went through a steady evolutionary change and finally led us to its most recent descendant ASP.NET Core 1.0.
- ASP.NET Core 1.0 is not a continuation of ASP.NET 4.6.

- It is a whole new framework, a side-by-side project which happily lives alongside everything else we know.
- It is an actual re-write of the current ASP.NET 4.6 framework, but much smaller and a lot more modular.
- Some people think that many things remain the same, but this is not entirely true. ASP.NET Core 1.0 is a big fundamental change to the ASP.NET landscape.

History of ASP.NET Core:

- ASP.NET Core is an open source and cloud-optimized web framework for developing modern web applications that can be developed and run on Windows, Linux and the Mac. It includes the MVC framework, which now combines the features of MVC and Web API into a single web programming framework.
- ASP.NET Core apps can run on .NET Core or on the full .NET Framework.
- It consists of modular components with minimal overhead, so you retain flexibility while constructing your solutions.

Pre-Requisite:

- HTML
- CSS
- JavaScript, JQuery
- Bootstrap
- C# Programming language.
- MVC
- Visual Studio 2017 or higher version

- We use Visual Studio 2019 in this tutorial series.

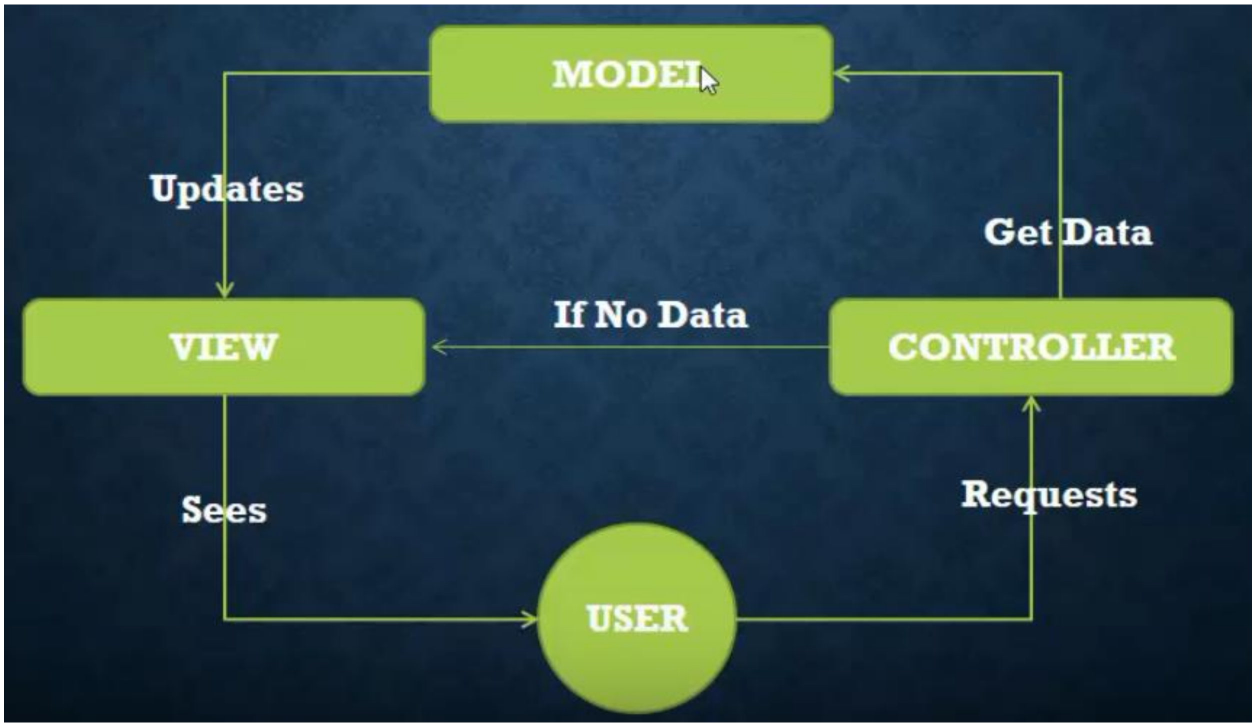
What is MVC Pattern:

- **MVC** Stand for **M**odel **-** **V**iew – **C**ontroller
- **MVC** is an architectural design pattern. It means this design pattern is used at the architecture level of an application.
- Model, View & Controller are the main components of MVC pattern.

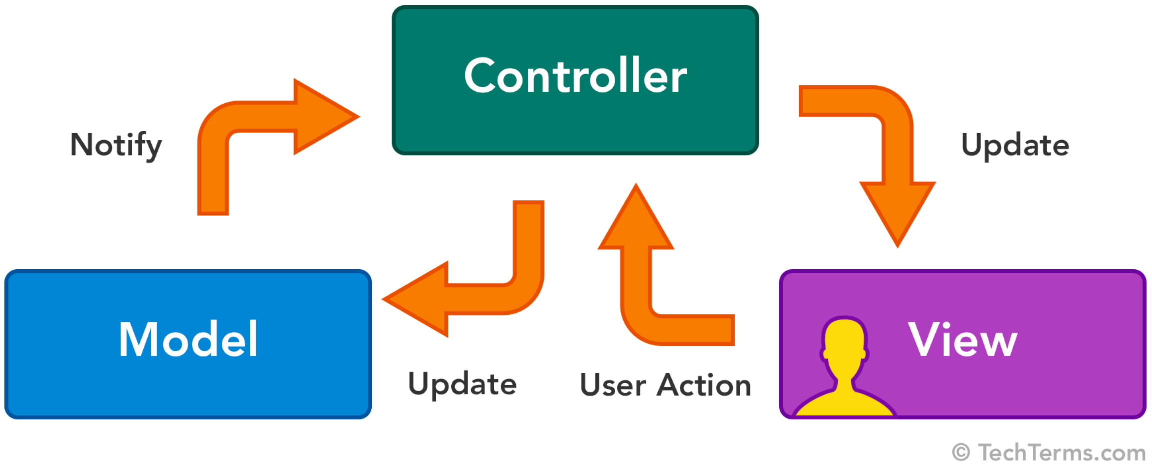
Role of components of MVC:

- Model (Business Entities)
- View (Presentation Logic)
- Controller (Business Logic)

Work Flow of MVC:



MVC WORKFLOW:



<u>**Real Life Example:**</u>

Controller:

- A controller is .cs (for c#) file which have some methods called Action Method.
- When a request comes on controller, it actually hits an action method.
- Represents the logic responsible for coordination between the view and model classes.
- Now everything depends on Action method what to return from it. It may return only view, only data, or both of them.

Model:

- A model in Asp.Net Core MVC is a simple class which has some properties.
- Handles requests to the application by the user.
- Model is used to pass data from controller (action method) to view and vice versa i.e View to Controller (action method).
- Model is also used for server side data validation. And with some technique we can generate client side validation also.
- It is not mandatory that each action method will return some model.

View:

- A View is a combination of Html and C#.
- Hence for C# application the extension of a view is. CSHTML**.**
- Whatever you see on your browser is a view.
- When a view (containing C#) gets rendered on browser then all its C# is converted into HTML. It means on a browser we will only see HTML and data.

Benefits of MVC:

- Separation of concern.
- Each component has a specific job hence it is easy to debug the code.

<u>Advantage of Separation of Concerns(SOS):</u>

- Allows work on individual pieces of the system in isolation
- Facilitates reusability
- Ensures the maintainability of a system
- Ensures extensibility
- Enables users to better understand the system

Environment Set in Machine (window, Linux, MacOS):

Editor or IDE (Integrated Development Environment).

- Visual Studio (Community)
- Visual Studio Code
- Sublime etc.

Dot Net Core SDK:

Download and Install dot Net core SDK. (any version).

Source: https://dotnet.microsoft.com/en-us/download

Check is it Install?

Command: dotnet –version

CREATE Project ASP.NET CORE MVC

System Configuration:

OS: Windows, Linux , MacOS

RAM: At least 8 GB

Hard Drive must be: SSD.

Hard Drive Must have 50GB Space

Environment Set in Machine (window, Linux, MacOS):

Editor or IDE (Integrated Development Environment).

- Visual Studio (Community)
- Visual Studio Code
- Sublime etc.

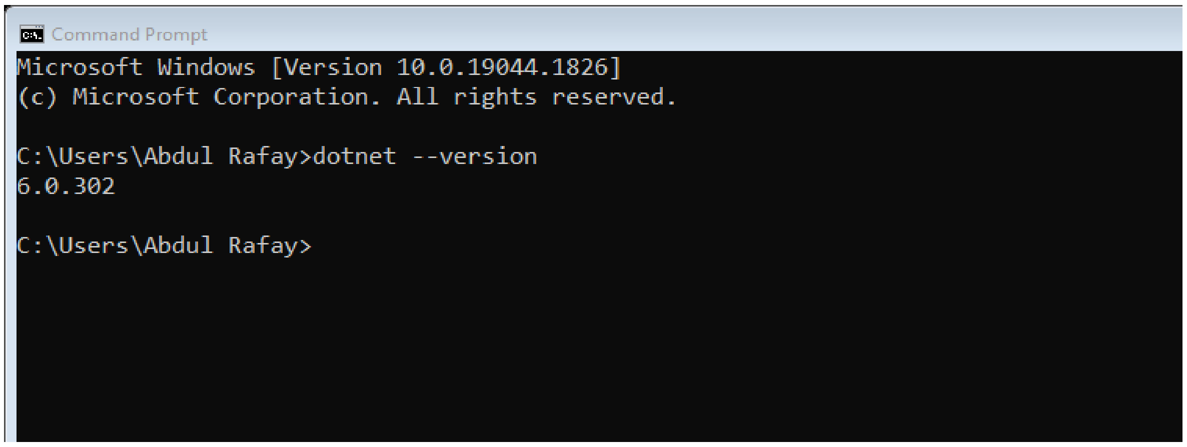
Dot Net Core SDK:

Download and Install dot Net core SDK. (any version).

Source: https://dotnet.microsoft.com/en-us/download

Check is it Install is complete?

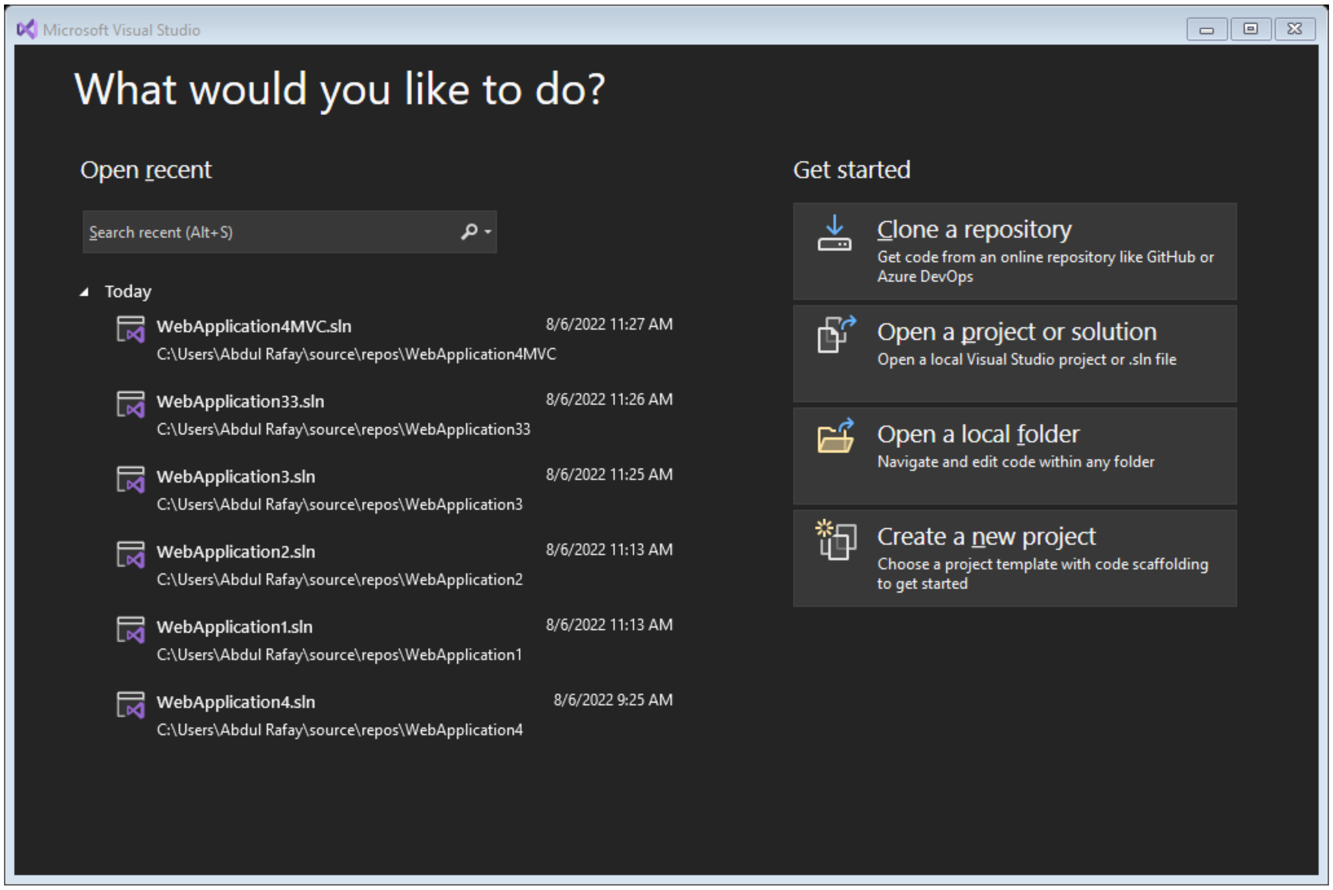
Command: dotnet –version



```
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Abdul Rafay>dotnet --version
6.0.302

C:\Users\Abdul Rafay>
```

Start Visual Studio:

Create a New Project.



What would you like to do?

Open recent

Search recent (Alt+S)

Today

WebApplication4MVC.sln 8/6/2022 11:27 AM
C:\Users\Abdul Rafay\source\repos\WebApplication4MVC

WebApplication33.sln 8/6/2022 11:26 AM
C:\Users\Abdul Rafay\source\repos\WebApplication33

WebApplication3.sln 8/6/2022 11:25 AM
C:\Users\Abdul Rafay\source\repos\WebApplication3

WebApplication2.sln 8/6/2022 11:13 AM
C:\Users\Abdul Rafay\source\repos\WebApplication2

WebApplication1.sln 8/6/2022 11:13 AM
C:\Users\Abdul Rafay\source\repos\WebApplication1

WebApplication4.sln 8/6/2022 9:25 AM
C:\Users\Abdul Rafay\source\repos\WebApplication4

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

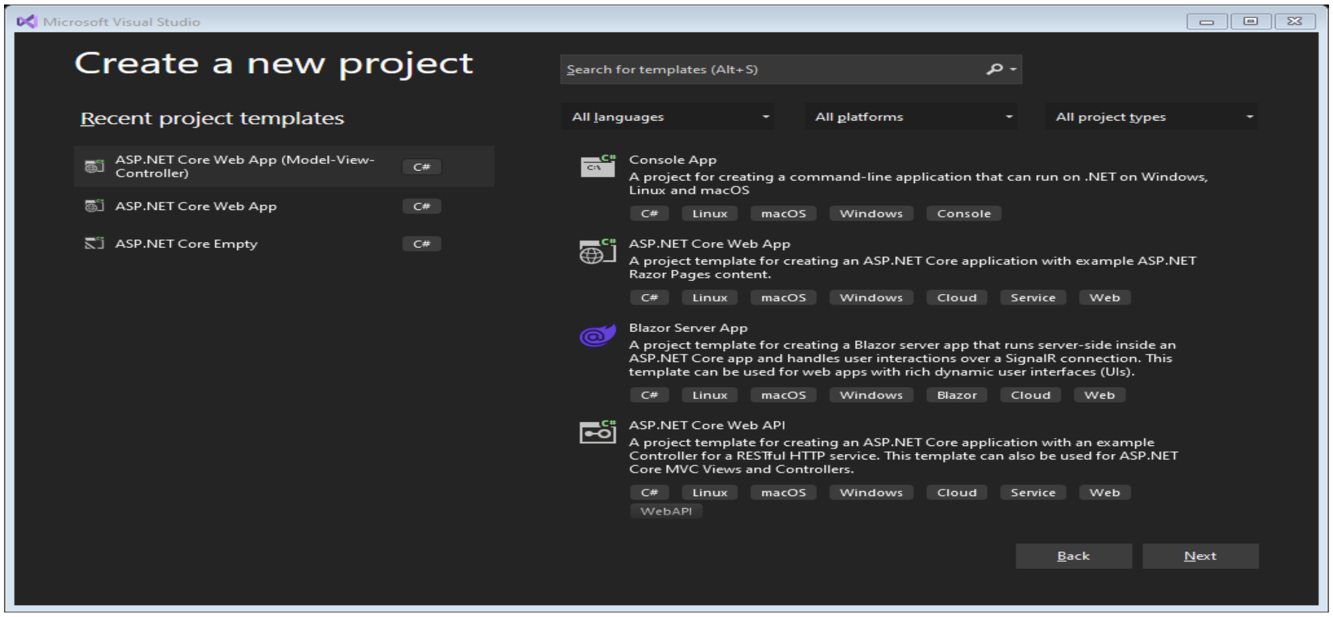
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

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

Select Technology with Template:

We will select ASP.NET Core Web Applicaton(MVC).



Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

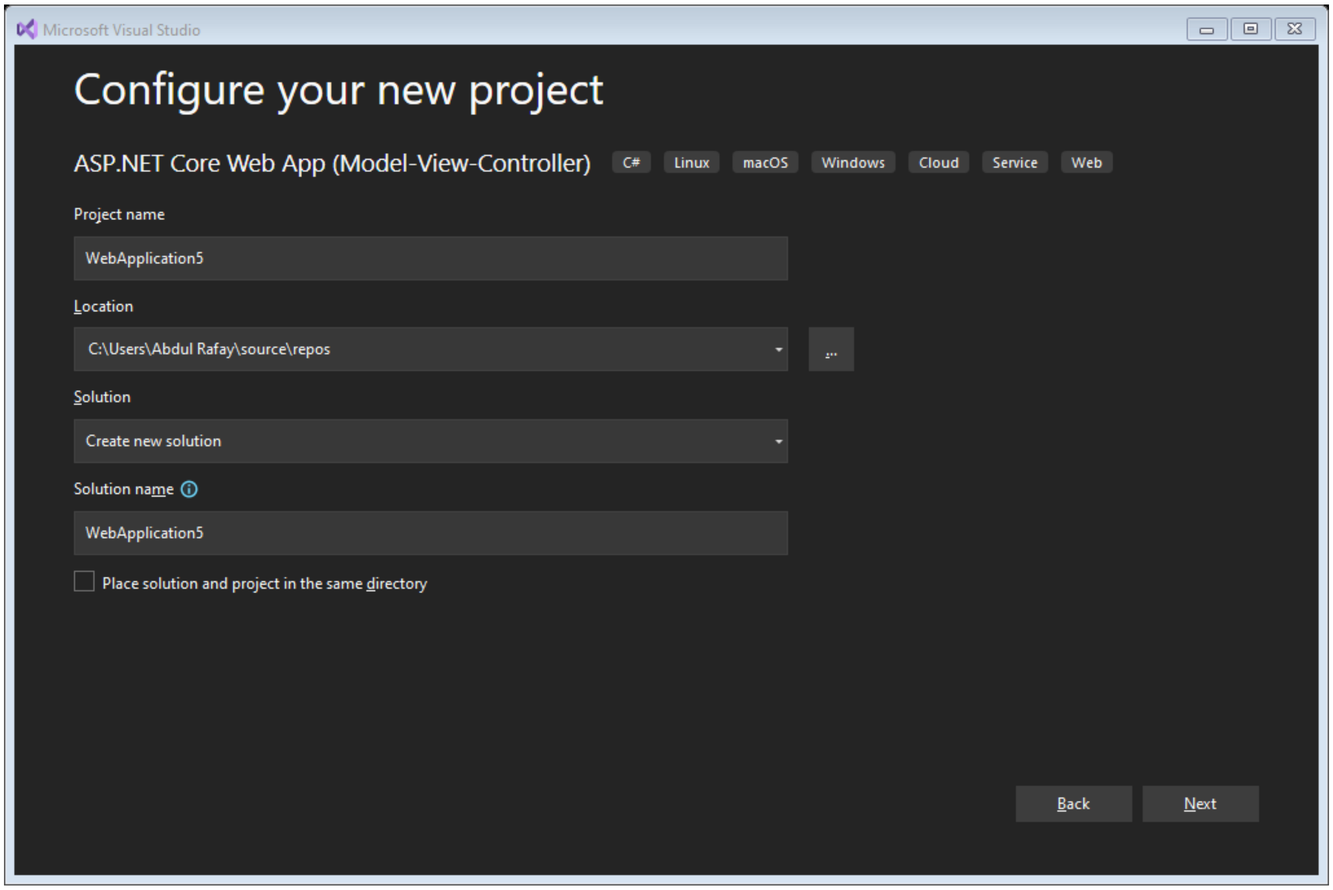
ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

ASP.NET Core Web 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.
C# Linux macOS Windows Cloud Service Web WebAPI

Configure the Project:

- **Set Name**
- **Set Directory**


Microsoft Visual Studio

Configure your new project

ASP.NET Core Web App (Model-View-Controller)

Project name: WebApplication5

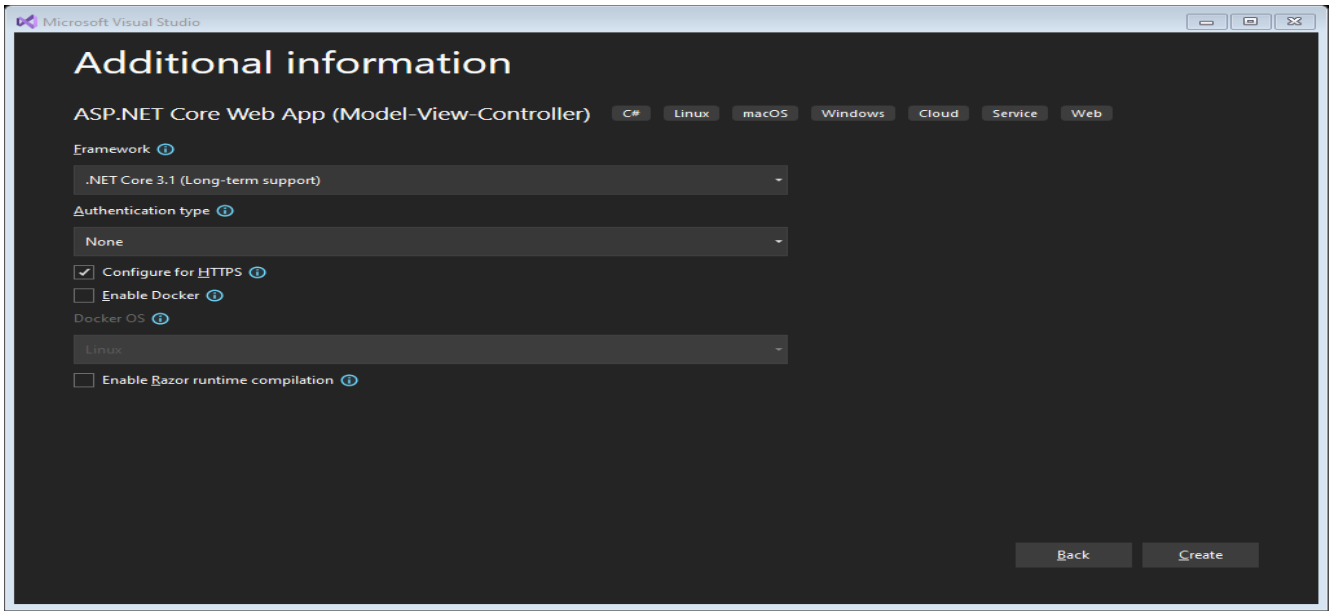
Location: C:\Users\Abdul Rafay\source\repos

Solution: Create new solution

Solution name: WebApplication5

☐ Place solution and project in the same directory

Additional Information(Select .Net Core version):


Microsoft Visual Studio

Additional information

ASP.NET Core Web App (Model-View-Controller)

Framework: .NET Core 3.1 (Long-term support)

Authentication type: None

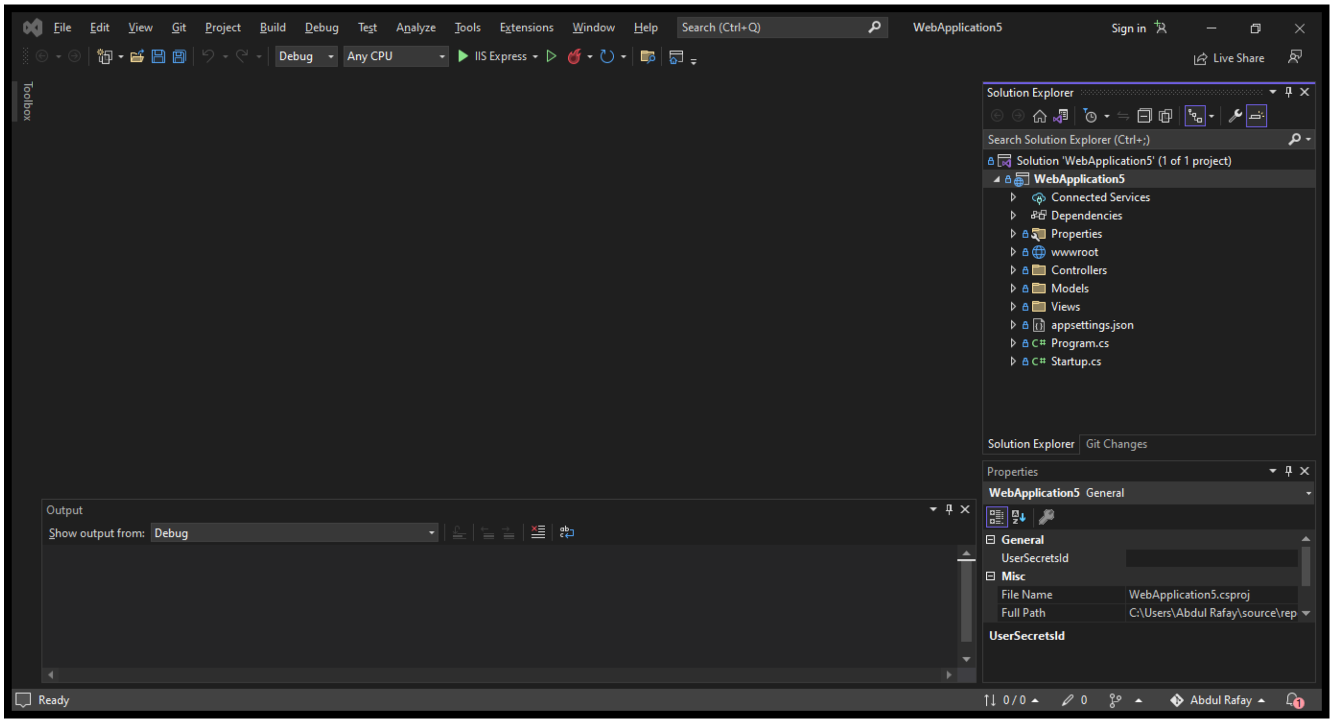
☑ Configure for HTTPS

☐ Enable Docker

Docker OS: Linux

☐ Enable Razor runtime compilation

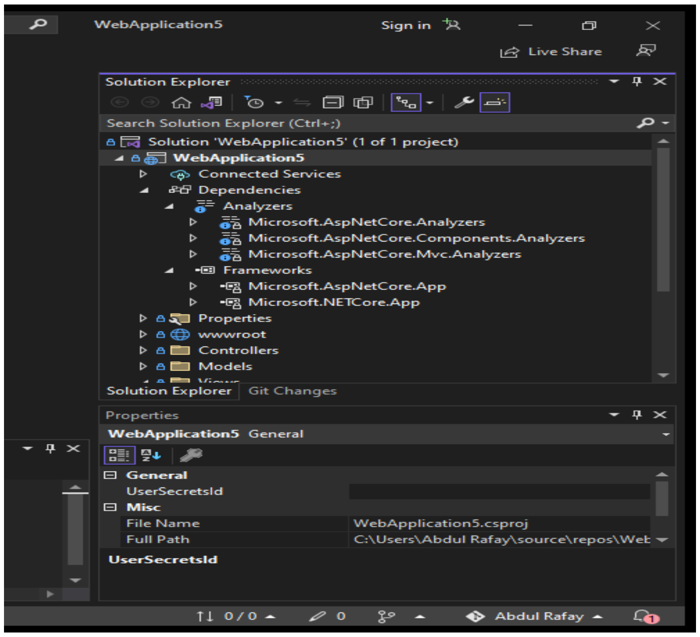
Complete build Project:



ASP.NET Core:

Here, you will learn about the project structure and significance of each file created by ASP.NET Core 2.0 application template using Visual Studio 2017.

The following is a default project structure when you create an empty ASP.NET Core application in Visual Studio.



Dependencies:

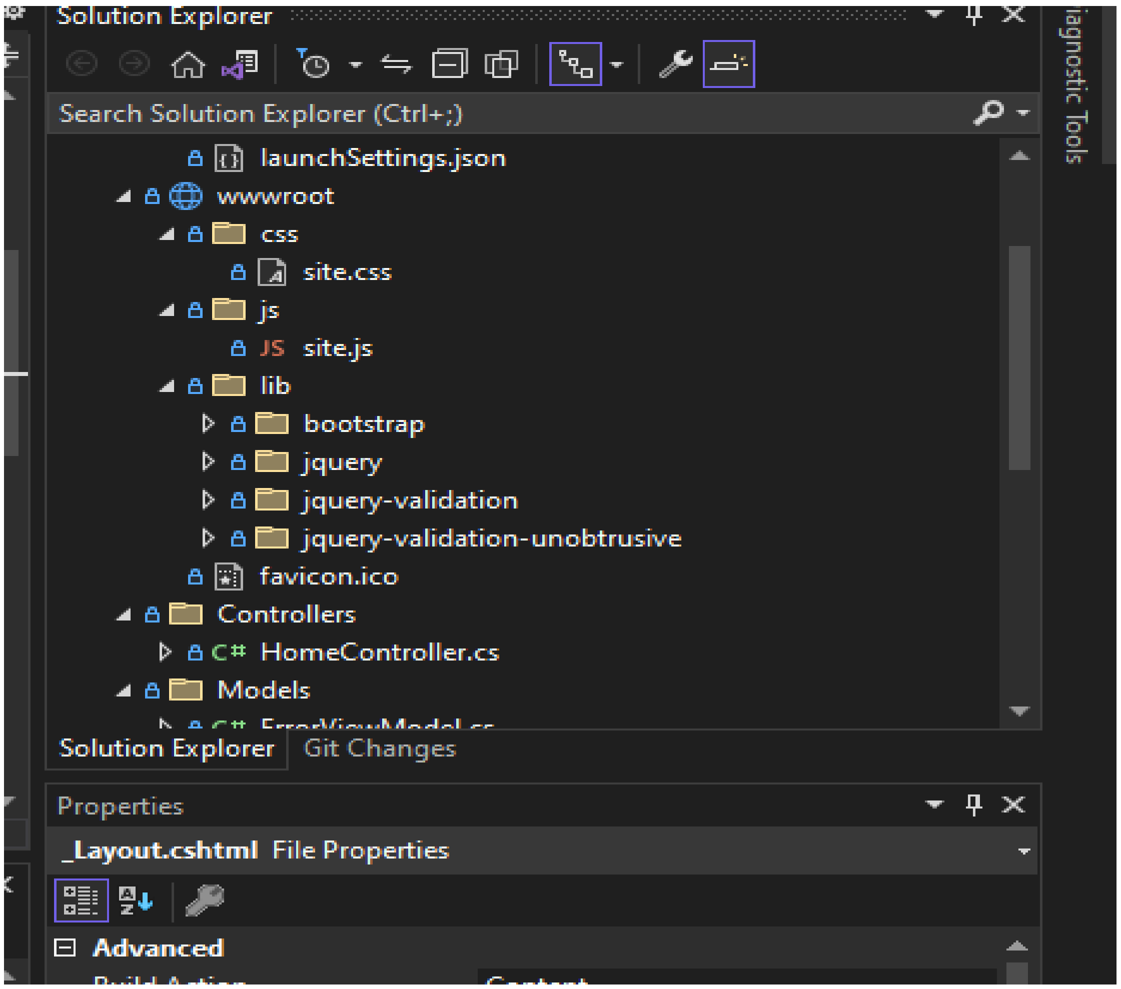
The Dependencies in the ASP.NET Core 3.1 project contain all the installed server-side NuGet packages, as shown below.

ASP.NET Core – WWWRoot Folder:

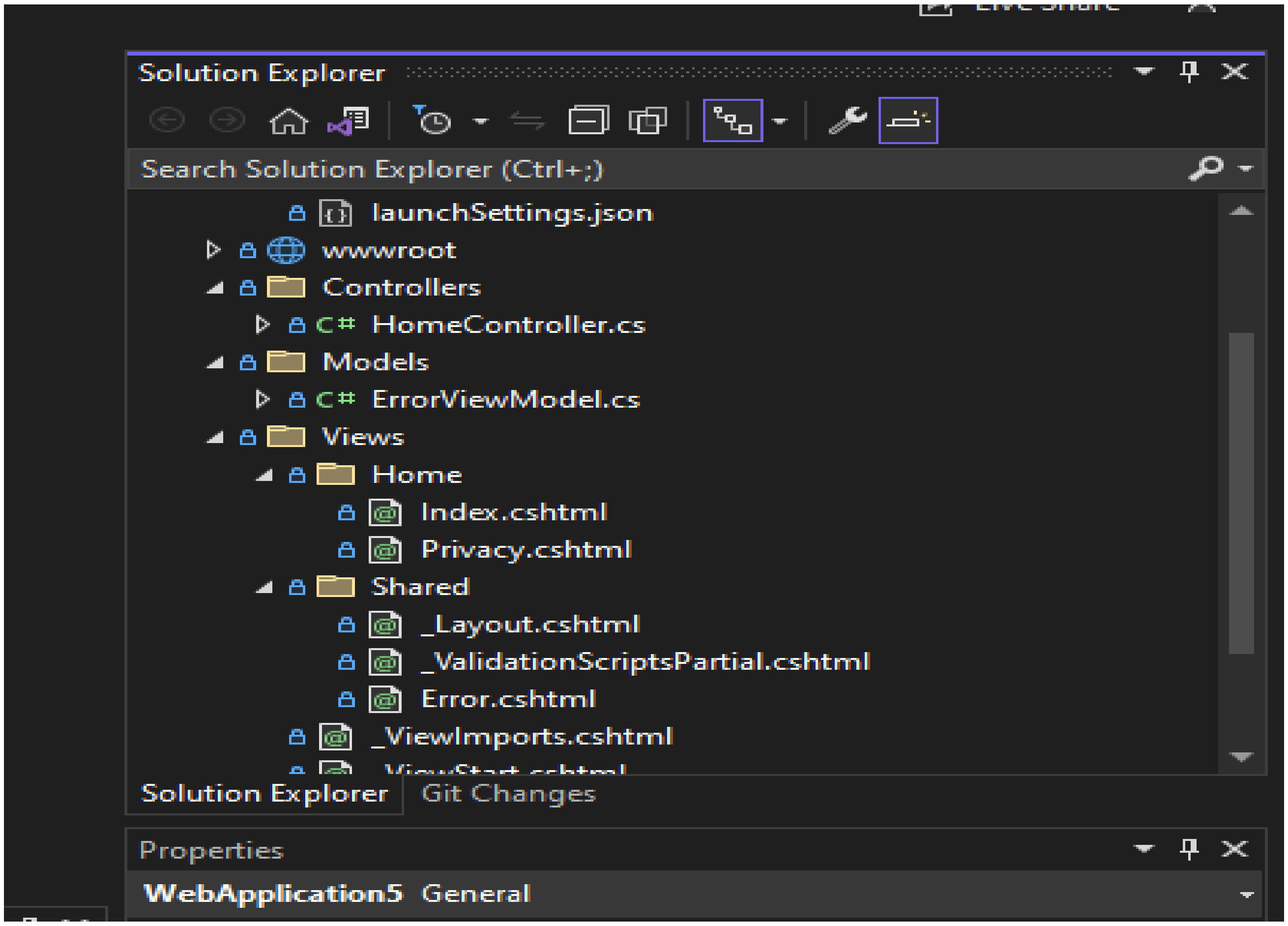
By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

In the standard ASP.NET application, static files can be served from the root folder of an application or any other folder under it. This has been changed in ASP.NET Core. Now, only those files that are in the web root - wwwroot folder can be served over an http request. All other files are blocked and cannot be served by default.

Generally, there should be separate folders for the different types of static files such as JavaScript, CSS, Images, library scripts etc. in the wwwroot folder as shown below.



Project MVC (Model View Controller) Structure:


Solution Explorer
Search Solution Explorer (Ctrl+;)

```
    launchSettings.json
  wwwroot
  Controllers
    HomeController.cs
  Models
    ErrorViewModel.cs
  Views
    Home
      Index.cshtml
      Privacy.cshtml
    Shared
      _Layout.cshtml
      _ValidationScriptsPartial.cshtml
      Error.cshtml
    _ViewImports.cshtml
```

Controller:

The Controllers folder contains class files for the controllers. A Controller handles users' request and returns a response. MVC requires the name of all controller files to end with "Controller". You will learn about the controller in the next section.

Model:

The Models folder contains model class files. Typically model class includes public properties, which will be used by the application to hold and manipulate application data.

View:

The Views folder contains HTML files for the application. Typically view file is a .cshtml file where you write HTML and C# or VB.NET code.

The Views folder includes a separate folder for each controller. For example, all the .cshtml files, which will be rendered by HomeController will be in View > Home folder.

The Shared folder under the View folder contains all the views shared among different controllers e.g., layout files.

MVC IN DETAIL WITH PRACTICAL

What is Controller?

A controller, in an ASP.NET CORE MVC application does the following:

- A Controller is a special class with .cs (C#) extension.
- Manages the flow of the application.
- Is responsible for intercepting incoming requests and executing the appropriate application code.
- Communicates with the models of the application and selects the required view to be rendered for the request.
- In MVC template a Controller class is inherited from Controller class.
- Allows separating the business logic of the application from the presentation logic.

In an ASP.NET MVC application, a controller is responsible to:

- Locate the appropriate method to call for an incoming request.
- Validate the data of the incoming request before invoking the requested method.
- Retrieve the request data and passing it to requested method as arguments.
- Handle any exceptions that the requested method throws.
- Help in rendering the view based on the result of the requested method.
- Caching, Security etc. can also be applied on controllers.

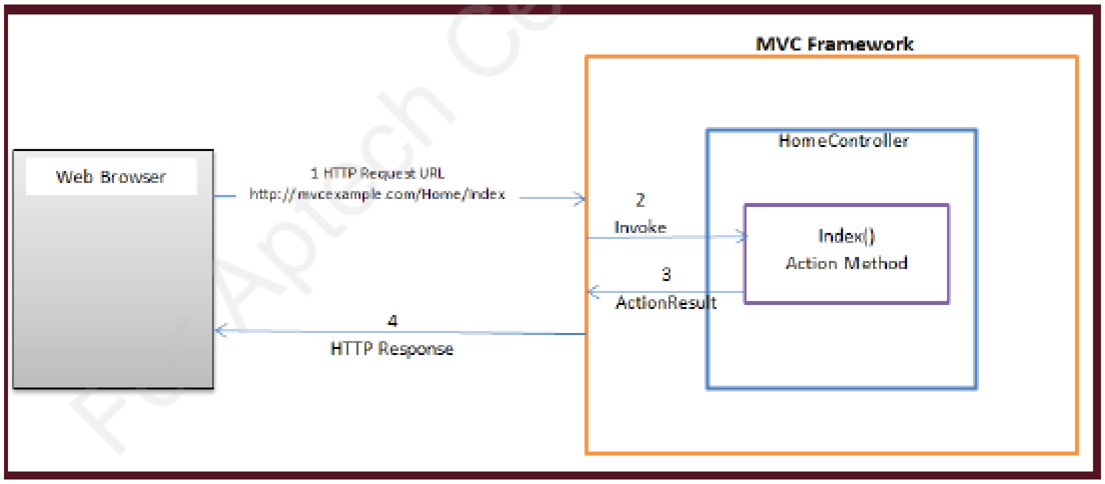
Action Method in Controller

- All the public methods of the Controller class are called Action methods. They are like any other normal methods with the following restrictions
- All Method in Controller called Action Method.

Restriction on Action Method:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

Let’s Practical of Controller.



What is View:

- A view is a file .cshtml (C sharp HTML) extension.
- A view is the combination of programming language & HTML.
- View are generally return form action method.

Role of View:

- A View in asp.net core MVC application is responsible for UI i.e application data presentation.
- We display information About the website on browser using Views.
- A user generally performs all the actions on a view.
- Location of Views in **View folder.**

What is View Engine?

- The View Engine is responsible for producing an HTML response when invoked by the Controller Action method.

- View Engine is a piece of code which is used to render server side code into view.

The Controller Action methods can return various types of responses, which are collectively called as Action Results. The ViewResult is the ActionResult which produces the HTML Response.

What is Razor View Engine?

The Razor View Engine is the default View Engine for the ASP.NET Core apps. It looks for Razor markup in the View File and parses it and produces the HTML response.

Razor Syntax?

Everything starts with @

The following are the two ways; by which you can achieve the transitions.

- Using Razor code expressions
- Using Razor code blocks.

Practical:

Make Basic Application with some views.

- Conditional Statements
- Loops
- Use Static Files (images,css,js)
- Install Bootstrap, font-awesome other library

What is WWWROOT:

By default, the wwwroot folder in the ASP.NET Core project is treated as a **web root folder**. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

Razor File Compilation:

Install : Microsoft.aspnetcore.mvc.razor.runtimecopilation

And add services in configure file.

```
services.AddRazorPages().AddRazorRuntimeCompilation();
```

- A model is the Class with .cs (C#) extension.

Responsible of Model:

- A model is responsive for data.
- We Get/Set data from/to a source in form of Model.
- The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it.
- Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.
- Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

MVC IN DETAIL WITH PRACTICAL

What is Model:

- A model is the Class with .cs (C#) extension.

Responsible of Model:

- A model is responsive for data.
- We Get/Set data from/to a source in form of Model.
- The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it.
- Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.
- Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

Make One Model Class:

Example:

```
namespace complete_web_application.Models
{
   public class student
   {
        public int stid{ get; set; }
        public string stname{ get; set; }

        public int stage{ get; set; }

    }
}
```

In start of Controller:

```
        public List<student> getstudents()
        {
            return new List<student>()
            {
                new student(){ stid = 1, stname="farhan" , stage = 20},
                new student(){ stid = 2, stname="ali" , stage =12},
                new student(){ stid = 3, stname="rehman" , stage = 27}
            };
        }
```

Create Controller:

```
   public List<student> getstudents()
        {
            return new List<student>()
            {
                new student(){ stid = 1, stname="farhan" , stage = 20},
                new student(){ stid = 2, stname="ali" , stage =12},
                new student(){ stid = 3, stname="rehman" , stage = 27}
            };
        }
```

Check URL:

https://localhost:44346/home/getstudents

Show model list data on View:

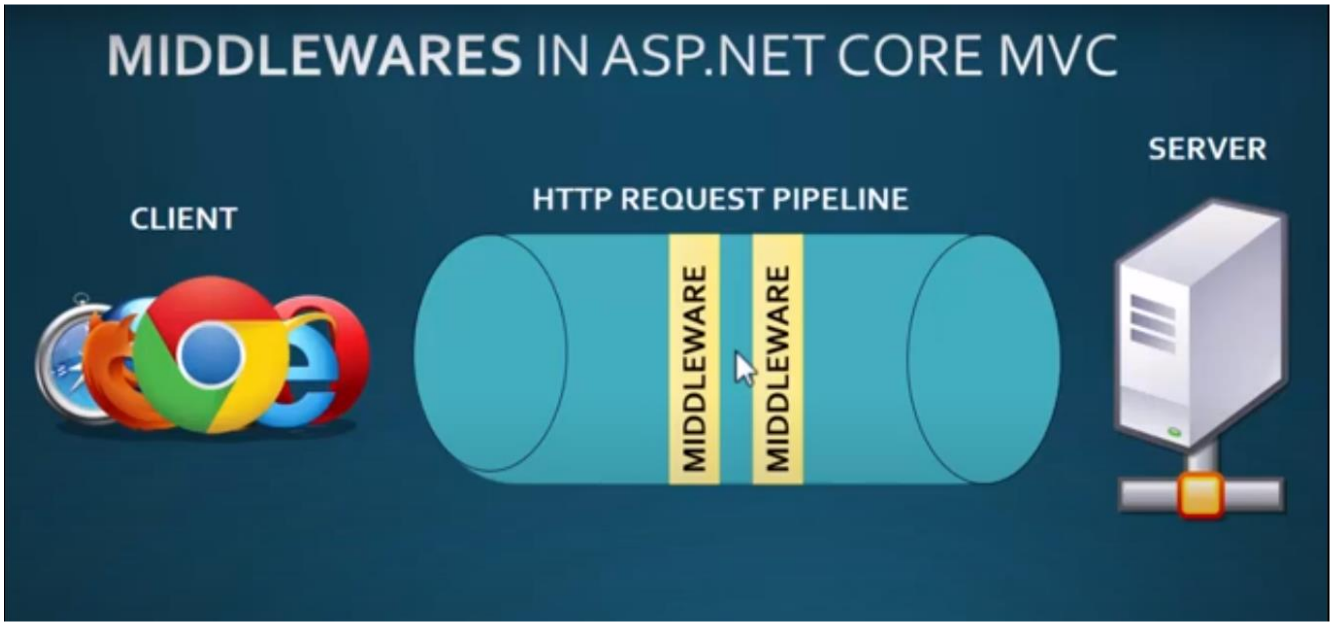
Action Method:

```
     public IActionResult getstudents()
        {
            var data = getstudents();
            return View(data);
        }
```

View:

```
@model IEnumerable<complete_web_application.Models.student>;

<h1>getbooks</h1>

<div class="container">
    <div class="row">


@foreach (var item in Model)
{
    <div class="col-md-4">
        <div class="card border px-5">
             <p>@item.stid</p>
             <p>@item.stname</p>
             <p>@item.stage</p>

            </div>
            </div>
}

    </div>
</div>
```

MIDDLEWARES IN ASP.NET CORE MVC



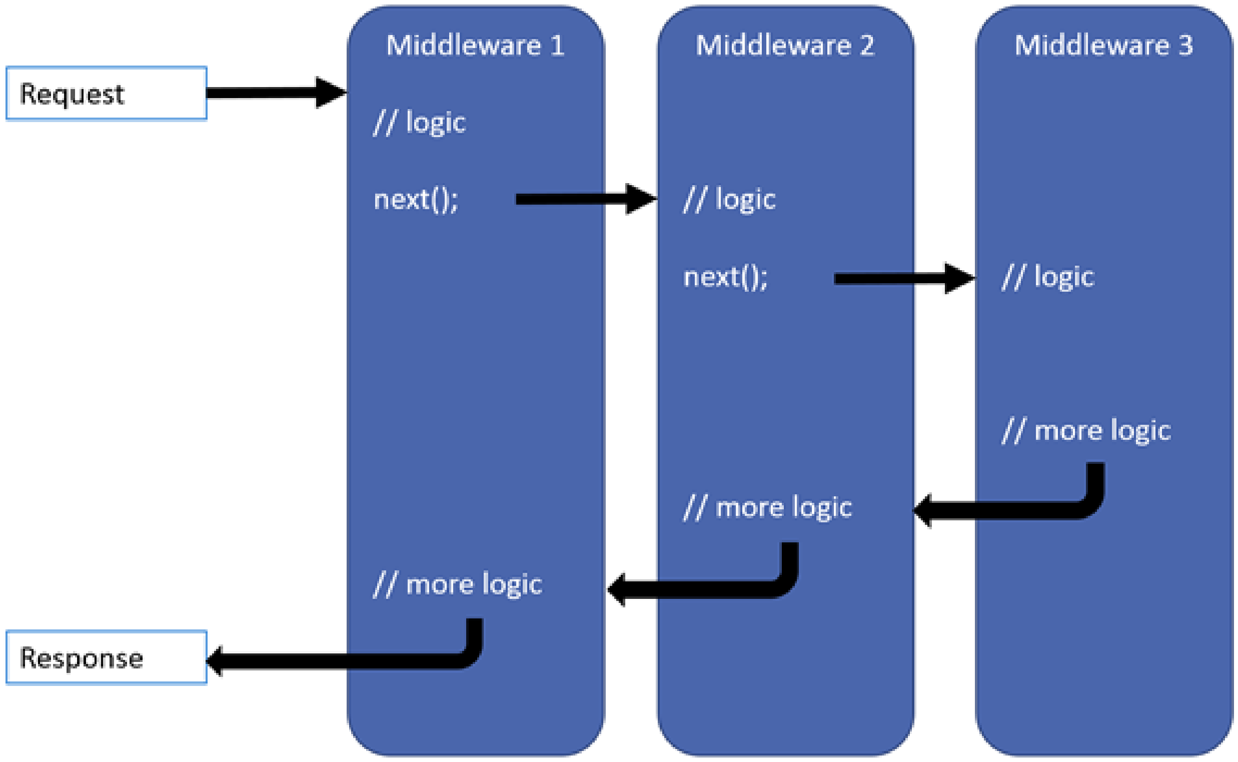
Introduction:

- ASP.NET Core introduced a new concept called Middleware. A middleware is nothing but a component which is executed on every request in ASP.NET Core application.
- Middleware in ASP.NET Core controls how our application responds to HTTP requests.
- Middleware are software components that are assembled into an application pipeline to handle request and responses.
- It can also control how our application looks when there is an error.
- It is a key piece in how we **authenticate** and **authorize** a user to perform specific actions.
- Each piece of middleware in ASP.NET Core is an object, and each pieces has a very specific, focused, and limited role.
- Ultimately, we need many pieces of middleware for and application to behave appropriately.
- Middleware has access to all the request and response.

- Middleware is a series of components present in this processing pipeline.
- Middleware are located in **Configure** method in **Startup.cs** class file.
- We can manage multiple middleware's for some specific task like,
    - Middleware's for Routing – e-g: app.UseMvcWithDefaultRout();
    - Middleware's for Static files – e-g: app.UseStaticFiles();
    - Middleware's for MVC functionality – e-g: app.UseMvc();

Configure Middleware:

- We can configure middleware in the **Configure** method of the Startup class using IApplicationBuilder instance generally called app.



Basic Example:

```
app.Use(async (context, next) =>
{
   await context.Response.WriteAsync("This si farhan webstie");
});
```

Example:

```
app.Use(async (context, next) =>
{
   await context.Response.WriteAsync("hello this is middleware 1");
   await next();

   await context.Response.WriteAsync("hello this is middelware 1 resonse");

});

app.Use(async (context, next) =>
{
   await context.Response.WriteAsync("this is middleware 2");

   await next();

});



app.UseRouting();

app.UseEndpoints(endpoints =>
{
   endpoints.MapGet("/", async context =>
   {
      await context.Response.WriteAsync("Hello World!");
   });
});
```

For Work MVC:

```
app.UseRouting();

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

ROUTING MVC WITH PRACTICAL

There are two types of Routing in ASP.NET CORE MVC.

1. Convention based Routing (Default in MVC)
2. Attribute Routing

Attribute Routing example:

```
 [Route("")]
[Route("Home")]
[Route("Home/Index")]
[Route("Home/Home")]
public IActionResult Index()
{
    return View();
}
[Route("Home/Home/Detail/{id}")]
public int Detail(int id)
{
    return id;
}
```

Razor Engine In ASP.NET CORE MVC:

Basics:

```
<h1>This is website : @DateTime.Now</h1>
<h1>@DateTime.Now.Year</h1>

@for (var i = 0; i < 10; i++)
{
    <p>This is Counter: @i</p>
}
```

Another example:

```
@{
    var a = 10;
    var b = "Farhan";
}
<p>This is @a</p>
<p>This is @b</p>
```

Another:

```
@{
    string[] names = ["farhan", "ali", "afan", "shahid"];
    foreach(var item in names)
    {
        <h2>This is @item</h2>
    }
}
```

MVC IN DETAIL WITH PRACTICAL

What is ViewBag?

The ViewBag in ASP.NET MVC is used to transfer temporary data (which is not included in the model) from the controller to the view.

Internally, it is a dynamic type property of the ControllerBase class which is the base class of the Controller class.

This type of data binding is known as loosely binding.

Example:

```
ViewBag.mytitle = "Home Page by Farhan";

ViewBag.data = new {Id = 1, name = "Farhan"};
```

View:

```
<h1>@ViewBag.mytitle</h1>

<p>@ViewBag.data</p>
```

What is ViewData?

In ASP.NET MVC, ViewData is similar to ViewBag, which transfers data from Controller to View. ViewData is of Dictionary type, whereas ViewBag is of dynamic type. However, both store data in the same dictionary internally.

ViewData is a dictionary, so it contains key-value pairs where each key must be a string.

Example:

```
ViewData["prop1"] = new { id = 1,name = "farhan jani " };
```

View:

```
var info = ViewData["prop1"];
<p>@info</p>
```

What is TempData?

In ASP.NET MVC, TempData is similar to ViewData, which transfers data from Controller to another View. TempData is of Dictionary type, whereas ViewData is of dynamic type. However, both store data in the same dictionary internally.

TempData is a dictionary, so it contains key-value pairs where each key must be a string.

Example:

```
    TempData["prop1"] = new { id = 1,name = "farhan " };
```

View:

```
var info = TempData["prop1"];
<p>@info</p>
```

What is Strongly Types View?

- Strongly typed view or strong types object is used to pass data from controllers to view.
- The View which binds with any model is called as strongly types view.
- You can bind any class as a model to view.

Example:

```
public IActionResult Index()
{
   var student = new List<Student>
   {
      new Student{ rollid = 1,  stname = "Fahran" , standard = "10th"},
      new Student{ rollid = 2,  stname = "ALI" , standard = "11th"},
      new Student{ rollid = 3,  stname = "AHMED" , standard = "12th"},
   };

   return View(student);
}
```

```
@{
@model IEnumerable<tutorial2.Models.Student>
}
<table border="1">
    @{
        foreach (var item in students)
        {
            <tr>
            <td>@item.rollid</td>
            <td>@item.stname</td>
            <td>@item.standard</td>
            </tr>
        }
    }
</table>
```

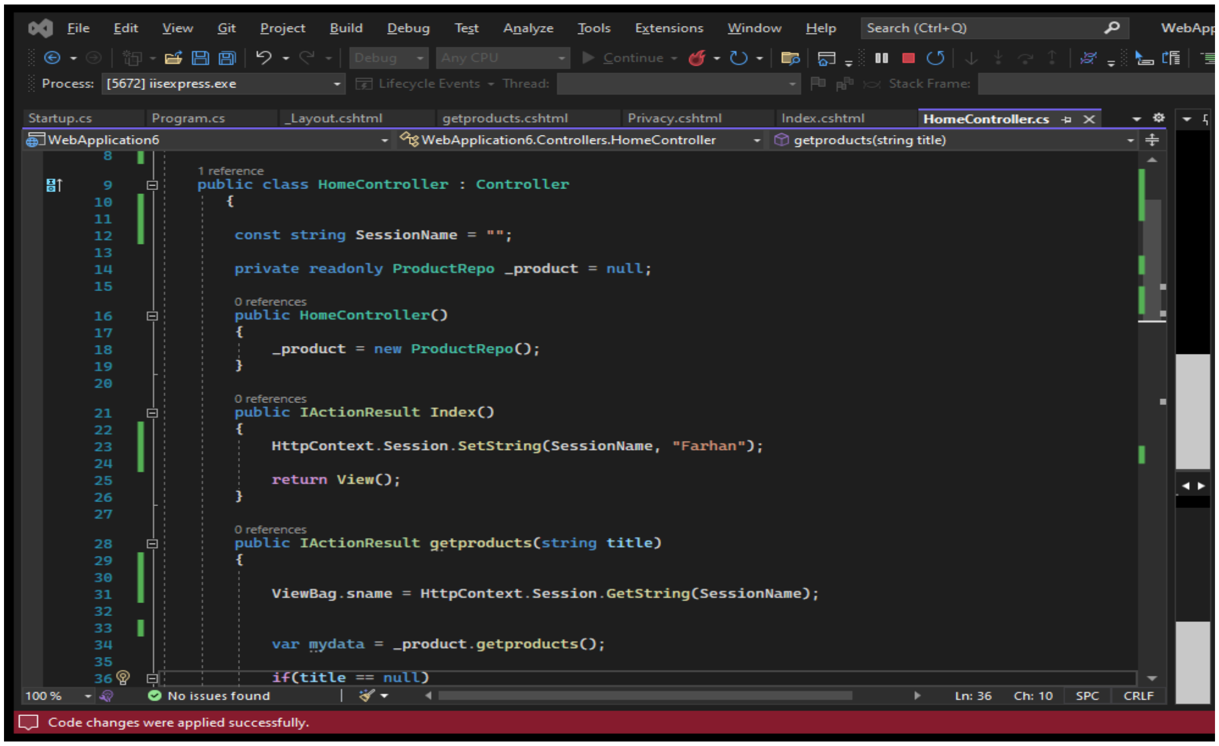
MVC IN DETAIL WITH PRACTICAL

What is Session?

Session state is an ASP.NET Core scenario for storage of user data while the user browses a web app. Session state uses a store maintained by the app to persist data across requests from a client. The session data is backed by a cache and considered ephemeral data.

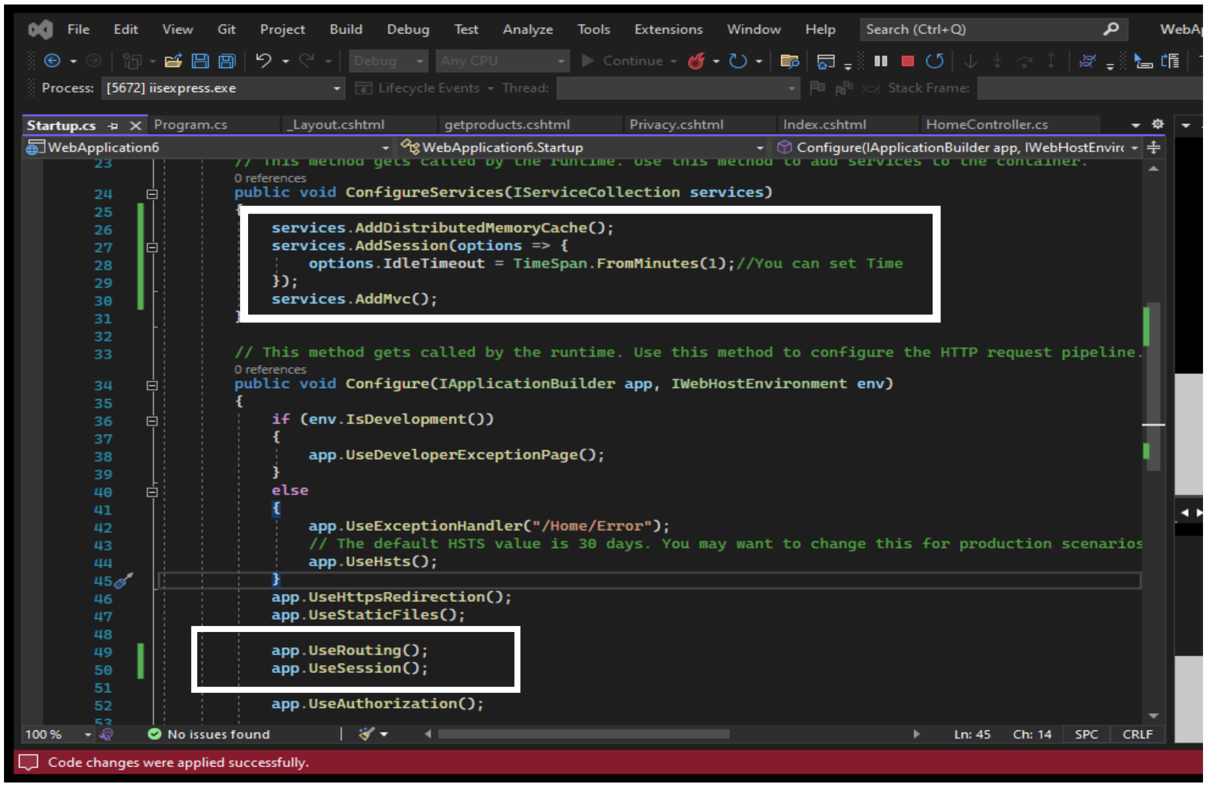
Steps of include Session in Application:

1. Download Session Package from “Manage NuGet Packages.
2. Work on Controller:



```
    1 reference
    public class HomeController : Controller
    {

        const string SessionName = "";

        private readonly ProductRepo _product = null;

        0 references
        public HomeController()
        {
            _product = new ProductRepo();
        }

        0 references
        public IActionResult Index()
        {
            HttpContext.Session.SetString(SessionName, "Farhan");

            return View();
        }

        0 references
        public IActionResult getproducts(string title)
        {

            ViewBag.sname = HttpContext.Session.GetString(SessionName);


            var mydata = _product.getproducts();

            if(title == null)
```

3. Configure the Services and add session services with time method.



```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDistributedMemoryCache();
    services.AddSession(options => {
        options.IdleTimeout = TimeSpan.FromMinutes(1);//You can set Time
    });
    services.AddMvc();
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();
    app.UseSession();

    app.UseAuthorization();
```

100 % No issues found Ln: 45 Ch: 14 SPC CRLF

Code changes were applied successfully.

MVC IN DETAIL WITH PRACTICAL

Html Helper Methods In Asp.Net Mvc 5:

In ASP.NET MVC Framework, helper methods:

- Are extension methods to the Html Helper class, can be called only from views.
- An HTML helper is a method that is used to render html content in a view.
- Simplifies the process of creating a view.
- Allows generating HTML markup that you can reuse across the Web application.

Some of the commonly used helper methods while developing an MVC application are as follows:

- Html.ActionLink()
- Html.BeginForm() and Html.EndForm()
- Html.Label()
- Html.TextBox()
- Html.TextArea()
- Html.Password()
- Html.CheckBox()
- Html.Hidden()

Html.ActionLink():

Html.ActionLink() helper method allows you to generate a hyperlink that points to an action method of a controller class.

The general syntax of the Html.ActionLink() helper method is as follows:

@Html.ActionLink(<link_text>,<action_method>,<optional_controller>)

An HTML helper is a method that is used to render html content in a view

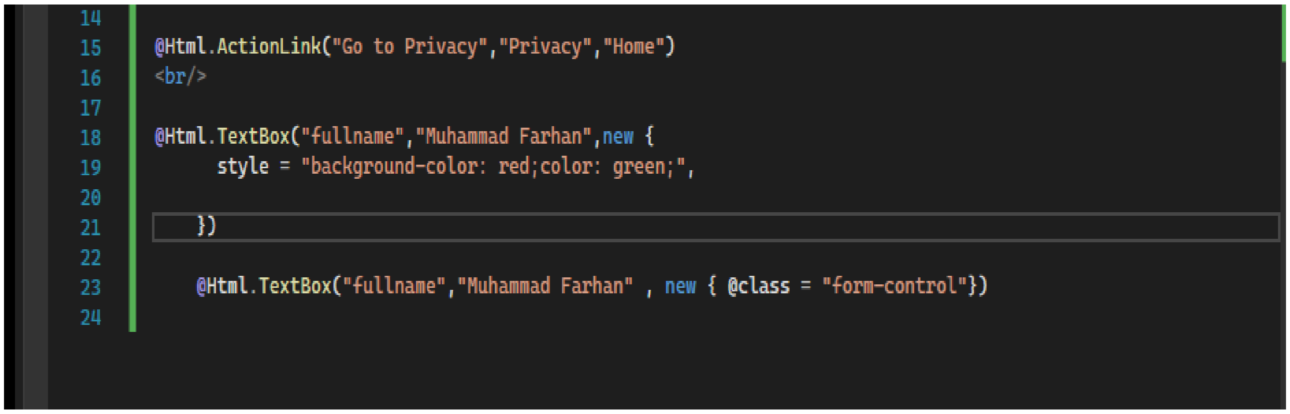
- To create the html for a textbox with id = “fullName” and name = “fullName”
    - **<input type=”text” name = “fullName” id = “fullName”>**

OR

- We can use the “TextBox” Html helper class method.
    - **@Html.TextBox(“fullName”)**

- There are several overloaded versions. To set a value, along with name of a textbox
    - **@Html.TextBox(“fullName”,”Adil”)**

To set HTML attributes, use the following overloaded version. Note that: We are passing HTML attributes (style and title) as an anonymous type.



```
@Html.ActionLink("Go to Privacy","Privacy","Home")
<br/>

@Html.TextBox("fullname","Muhammad Farhan",new {
      style = "background-color: red;color: green;",

   })

   @Html.TextBox("fullname","Muhammad Farhan" , new { @class = "form-control"})
```

- We can use external style sheet to give the style to the textbox.
- Applying bootstrap class to textbox.
- If you want to use attributes that are reserved in Csharp programming language like class, readonly etc. Then you have to use @ symbol before the name of attribute.

Html.BeginForm() helper method:

- Html.BeginForm() allows you to mark the start of a form.
- Html.BeginForm() co-ordinates with the routing engine to generate a URL.
- Html.BeginForm() is responsible for producing the opening tag.

The general syntax of the Html.BeginForm() helper method is as follows:

```
@{Html.BeginForm(<action_method>,<controller_name>);}
```

where,

- **action_method:** Is the name of the action method.
- **controller_name:** Is the name of the controller class.

Once you use the **Html.BeginForm()** helper method to start a form, you need to end a form using the **Html.EndForm()** helper method.

Following code snippet shows using the Html.BeginForm() and Html.EndForm() helper methods:

```
@{Html.BeginForm("Browse","Home");}
<p>Inside Form</p>
@{Html.EndForm();}
```

In the above code the **Html.BeginForm()** method specifies the Browse action of the HomeController as the target action method to which the form data will be submitted.

You can also avoid using the **Html.EndForm()** helper method to explicitly close the form by using the **@using** statement before the **Html.BeginForm()** method.

Html.Label() helper method:

- Allows you to display a label in a form.
- Enables attaching information to other input elements, such as text inputs, and increase the accessibility of your application.

The general syntax of the Html.Label() helper method is as follows:

```
@Html.Label(<label_text>)
```

where,

label_text: Is the name of the label.

Html.TextBox() helper method:

- Allows you to display an input tag.
- Used to accept input from a user.

To create the html for a textbox with id = “fullName” and name = “fullName”

```
<input type=”text” name = “fullName” id = “fullName”>
```

OR

We can use the “TextBox” Html helper class method

```
@Html.TextBox(“fullName”)
```

There are several overloaded versions. To set a value, along with name of a textbox

```
@Html.TextBox(“fullName”,”Adil”)
```

To set HTML attributes, use the following overloaded version. Note that: We are passing HTML attributes (style and title) as an anonymous type.

Following code snippet shows using a Html.TextBox()method:

```
@{Html.BeginForm("Browse","Home");}
@Html.Label("User Name:")</br>
@Html.TextBox("textBox1")</br></br>
<input type="submit" value="Submit">
@{Html.EndForm();}
```

Html.TextArea() helper method:

- Allows you to display a element for multi-line text entry.</p>
- Enables you to specify the number of columns and rows to be displayed in order to control the size of the text area.

Following code snippet shows using a Html.TextArea() method:

```
@{Html.BeginForm("Browse","Home");}
@Html.Label("User Name:")</br>
@Html.TextBox("textBox1")</br></br>
@Html.Label("Address:")</br>
@Html.TextArea("textarea1")</br></br>
<input type="submit" value="Submit">
@{Html.EndForm();}
```

Html.Password() helper method:

You can use the **Html.Password()** helper method to display a **password** field with bulleted text inside in place of original password.

Following code snippet shows using a Html.Password() method:

```
@{Html.BeginForm("Browse","Home");}
@Html.Label("User Name:")</br>
@Html.TextBox("textBox1")</br></br>
@Html.Label("Address:")</br> @Html.TextArea("textarea1")</br></br>
@Html.Label("Password:")</br>@Html.Password("password")</br></br>
<input type="submit" value="Submit">
@{Html.EndForm();}
```

Html.CheckBox() helper method:

You can use the **Html.CheckBox()** helper method to display a check box that enables the user to select a true or false condition.

Following code snippet shows using a Html.CheckBox() method:

```
@{Html.BeginForm("Browse","Home");}
@Html.Label("User Name:")</br>
@Html.TextBox("textBox1")</br></br>
@Html.Label("Address:")</br> @Html.TextArea("textarea1")</br></br>
@Html.Label("Password:")</br>@Html.Password("password")</br></br>
@Html.Label("I need updates on my mail:")
@Html.CheckBox ("checkbox1")</br> </br>
<input type="submit" value="Submit"> @{Html.EndForm();}
```

In this code, the Html.CheckBox() helper method renders a hidden input in addition to the check box input. The hidden input ensures that a value will be submitted, even if the user does not select the check box.

Html.DropDownList() helper method:

- Return a <**select>** element that shows a list of possible options and also the current value for a field.
- Allows selection of a single item.

The general syntax of the Html.DropDownList() helper method is as follows:

where,

- **value1**,**value2** and **value3** are the options available in the drop-down list.
- **Choose:** Is the value at the top of the list.

Following code snippet shows using a Html.DropDownList() method:

In Above code, the **Html.DropDownList()** method creates a drop-down list in a form with myList as its name and contains three values that a user can select from the drop-down list.

Html.RadioButton() helper method:

The **Html.RadioButton()** helper method allows you to provide a range of possible options for a single value.

The general syntax of the Html.RadioButton() helper method is as follows:

```
@Html.RadioButton("name","value",isChecked)
```

where,

- **name:** Is the name of the radio button input element.
- **value:** Is the value associated with a particular radio button option.
- **isChecked:** Is a Boolean value that indicates whether the radio button option is selected or not.

```
@{Html.BeginForm("Browse","Home");}
@Html.Label("User Name:")</br>
@Html.TextBox("textBox1")</br></br>
@Html.Label("Address:")</br> @Html.TextArea("textarea1")</br></br>
@Html.Label("Password:")</br>@Html.Password("password")</br></br>
@Html.Label("I need updates on my mail:")@Html.CheckBox("checkbox1")</br> </br>
@Html.Label("Select your city:") @Html.DropDownList("myList", new SelectList(new
[] {"New York", "Philadelphia", "California"}), "Choose")</> </br></br>
Male @Html.RadioButton("Gender", "Male", true)</br>
Female @Html.RadioButton("Gender", "Female")</br> </br>
<input type="submit" value="Submit">
@{Html.EndForm();}
```

In Above code, the Html.RadioButton() helper methods is used to create two radio buttons to accept the gender of a user.

Url.Action() helper method:

The **Url.Action()** helper method generates a URL that targets a specified action method of a controller.

The general syntax of the Url.Action() helper method is as follows:

```
@Url.Action(<action_name>, <controller_name>)
```

where,

- **action_name:** Is the name of the action method.
- **controller_name:** Is the name of the controller class.

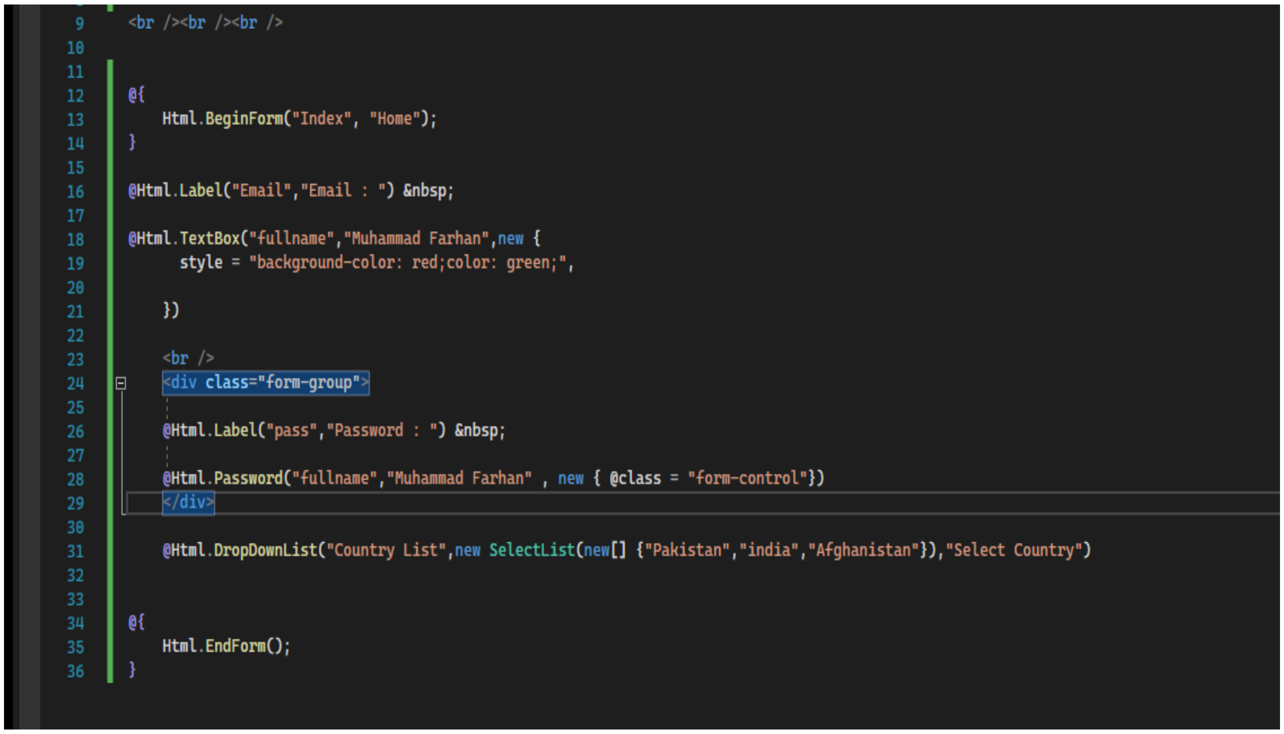
Following code snippet shows the Url.Action() method:

```
<a href='@Url.Action("Browse", "Home")'>Browse</a>
```

Above code creates a hyperlink that targets the URL generated using the **Url.Action()** method. When a user clicks the hyperlink, the Browse action of the Home controller will be invoked.

Is it compulsory to use html helpers in ASP.Net MVC Application?

No, you can type the required html, but using html helpers will greatly reduce the amount of HTML markup that we have to write in a view. Views should be as simple as possible.



```
<br /><br /><br />


@{
    Html.BeginForm("Index", "Home");
}

@Html.Label("Email","Email : ") &nbsp;

@Html.TextBox("fullname","Muhammad Farhan",new {
      style = "background-color: red;color: green;",

    })

    <br />
    <div class="form-group">

    @Html.Label("pass","Password : ") &nbsp;

    @Html.Password("fullname","Muhammad Farhan" , new { @class = "form-control"})
    </div>

    @Html.DropDownList("Country List",new SelectList(new[] {"Pakistan","india","Afghanistan"}),"Select Country")


@{
    Html.EndForm();
}
```

Tag Helpers:

What is Tag Helpers?

- Tag helpers are used to render server side code on a Razor (.cshtml) file to create and render HTML elements.

Example:

- Create Image tag
- Create form tag
- Create Link etc.

Main Point:

- There are many built-in Tag Helpers for common tasks, such as creating forms, links, loading assets etc. Tag Helpers are authored in C#, and they target HTML elements based on the element name, the attribute name, or the parent ta
- If you are familiar with HTML Helpers, Tag Helpers reduce the explicit transitions between HTML and C# in Razor views.

How to use of start tag helpers?

First set the scope

- @addTagHelper directive
- @removeTagHelper directive

Add the Directive in ViewImport File.

@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

TAG HELPERS ADVANTAGES

Now, we need to understand why Tag Helpers is important or what are the advantages of it's over the HTML Helper objects. So that can compare these two different helper objects. So, before going to in deep discussion about Tag Helpers, let’s discuss the main advantages of Tag Helpers over HTML Helpers objects

1. Tag Helpers use server-side binding without any server-side code.
2. This helper object is very much use full when HTML developers do the UI designing who does not have any idea or concept about Razor syntax.
3. It provides us an experience of working on basic HTML environments.the

4. It Supports rich IntelliSense environment support to create a markup between HTML and Razor

Anchor Tag Helper:

- Anchor tag helper is used to create a link <a></a> html element.
- By default all tag helpers attributes are prefixed with asp-

Main Attributes:

- Asp-controller
- Asp-action
- Asp-area
- Asp-route

Example:

```
<a asp-controller="Home" asp-action="Privacy">To go Privacy</a>
```

Example but Params:

```
<a  asp-controller="Home" asp-action="getproductinfo" asp-route-id="@item.proid" >
```

Image Tag Helper:

- Image tag helper is used to give some additional capabilities to HTML img tag.
- Image tab helper is uses to provide cache-busting behavior for static image.

Attribute:

- Asp-append-version

Example:

```
<img src="~/images/img2.jpg" width="50px" asp-append-version="true"/>
```

Environment Tag Helpers:

This Tag Helper renders the enclosed HTML content based on the current hosting environment conditionally. This helper object contains a single attribute named name. This attribute accepts string type value and value can be assigned as a comma separation method.

Attributes:

- **Names**
- **Include**
- **Exclude**

Example:

```
<environment names="Development">

   <h1>Development</h1>

</environment>

<environment names="Production">

   <h1>Production</h1>

</environment>
```

Another Example:

```
<environment include="Staging,Production">
   <strong>IWebHostEnvironment.EnvironmentName is Staging or Production</strong>
</environment>

<environment names="Production">

   <h1>Production</h1>

</environment>

<environment exclude="Development">
   <strong>IWebHostEnvironment.EnvironmentName is not Development</strong>
</environment>
```

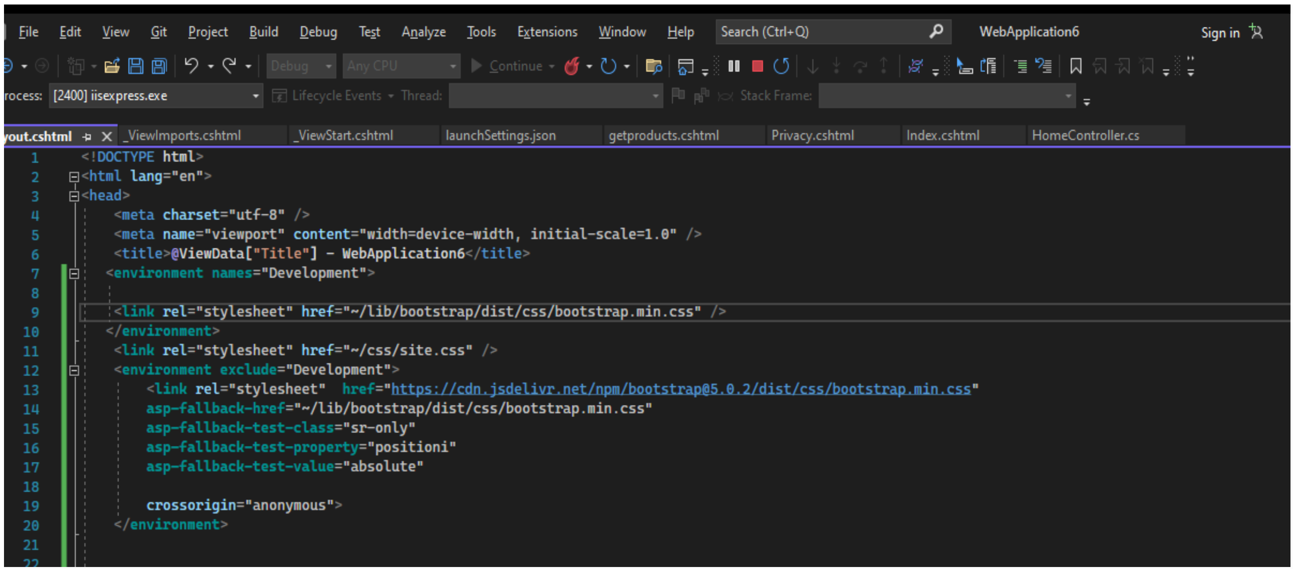
Link Tag Helper:

The Link Tag Helper generates a link to a primary or fall back CSS file. Typically the primary CSS file is on a Content Delivery Network (CDN).

A CDN:

- Provides several performance advantages vs hosting the asset with the web app.
- Should not be relied on as the only source for the asset. CDNs are not always available, therefore a reliable fallback should be used. Typically the fallback is the site hosting the web app.

The Link Tag Helper allows you to specify a CDN for the CSS file and a fallback when the CDN is not available. The Link Tag Helper provides the performance advantage of a CDN with the robustness of local hosting.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplication6</title>
   <environment names="Development">

    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
   </environment>
    <link rel="stylesheet" href="~/css/site.css" />
    <environment exclude="Development">
        <link rel="stylesheet"  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
        asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
        asp-fallback-test-class="sr-only"
        asp-fallback-test-property="positioni"
        asp-fallback-test-value="absolute"

        crossorigin="anonymous">
    </environment>
```

Create Form in Asp.Net Core using tag helpers:

```
<div class="container">
   <div class="row">

      <div class="offset-md-3 col-md-6">

<form method="post" asp-action="AddProduct">

      <div class="form-group">

     <label asp-for="protitle">Title : </label>
    <input asp-for="protitle"  class="form-control"/>
      </div>

      <input type="file" asp-for="proimg" />
      <br />
      <div class="form-group">

    <label asp-for="proshortdesc">Description (Short) : </label>
    <input asp-for="proshortdesc" class="form-control" />
    </div>
    <div class="form-group">
    <label asp-for="prolongdesc">Description (Long) : </label>
      <textarea asp-for="prolongdesc" class="form-control">
      </textarea>
    </div>
      <br />
      <div class="form-group">

    <label asp-for="proprice">Price : </label>
      <input asp-for="proprice" class="form-control" />
   </div>


    <button type="submit" value="addproduct" class="btn btn-primary" > Add product</button>

</form>

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

Controller Work:

```
    public IActionResult AddProduct()
        {
            return View();

        }
        [HttpPost]
        public JsonResult AddProduct(Product product)
        {

            return Json(product);
        }
```

MVC IN DETAIL WITH PRACTICAL

Model:

```
public IFormFile coverimg { get; set; }
```

View:

```
<label asp-for="coverimg"></label>
    <input  asp-for="coverimg"  /><br/>
```

Controller:

```
private readonly IWebHostEnvironment _webHostEnvironment;

public HomeController(IWebHostEnvironment webHostEnvironment)
{
    _webHostEnvironment = webHostEnvironment;
}
```

IN Action Method

```
if (model.img != null)
{
    folder = "images/";
    folder += Guid.NewGuid().ToString() + "_" + model.img.FileName;
    string serverfolder = Path.Combine(environment.WebRootPath, folder);

    model.img.CopyTo(new FileStream(serverfolder, FileMode.Create));

}
```

MVC IN DETAIL WITH PRACTICAL

Data Validation in Dotnet Core:

Validation is a process of checking an activity whether it meets the desired level of compliance.

In Asp.Net Core, validation is the process of checking fields of a form in order to meet the defined criteria.

Namespace: System.ComponentModel.DataAnnotations

How to check the Validation : ModelState.IsValid

When we developed any type of application such as web-based or desktop based, in both the application one is the key part of development is Data Validation or Form Validation. These validations always ensure us that user fills the correct form of data into the application so that we can process that data and save for future use. Actually, normally users always fill the form with necessary information and submit the form. But, sometimes the user make some mistakes. In this case, form validation is required to ensure that the user always provides the required information in the proper format which is required for the successful submission of the form. In this article, we will discuss this different types of form validations.

Server Side Validation:

In case of Server-side validation, information filled by the users are send to the server and then validate that information using the server-side language i.e. in the Asp.Net Core. In this process, if the data not validated, then a response along with proper error message is sent back to the client where these messages are shown to the user so that those wrong data can be rectified. This type of validation is always secure because if the user disabled the scripting language in the browser then also it will work.

Example:

```
public class Employee

{

public int Id { get; set; }



[Required]
[StringLength(100)]
public string EmployeeName { get; set; }

[Required]
[DataType(DataType.Date)]
public DateTime JoinDate { get; set; }

[Required]
[StringLength(100)]
public string Designation { get; set; }

[Range(0, 999.99)]
public decimal Salary { get; set; }

[Required]
public Gender Gender { get; set; }

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

Data Validation	Definition
Required	This validation attributes makes any property as required or mandatory.
StringLength	This validation attributes validate any string property along with itslength
Compare	This validation attribute basically compares the two property values in a model match
EmailAddress	This validation attribute validates the email address format
Phone	This validation attribute validates the phone no format
CreditCard	This attribute validates a credit card format
Range	This validation attributes any property to check it is exist within the given range or not.

Url	This validation attributes validates the property contains an URL format or not
RegularExpression	This validation attributes normally match the data which specified the regular expression format.

Example:

View: (HTML):

```
  <div class="text-danger" asp-validation-summary="All">
    </div>

    <form asp-controller="Home" asp-action="Index" method="post">
      <label asp-for="Name"></label>
      <input asp-for="Name" /><br />
      <span asp-validation-for="Name" class="text-danger"></span>
      <br/>
      <label asp-for="Email"></label>
      <input asp-for="Email" /><br />
      <span asp-validation-for="Email" class="text-danger"></span>
      <br/>
      <label asp-for="password"></label>
      <input asp-for="password" /><br />
      <span asp-validation-for="password" class="text-danger"></span>
      <br/>
       <label asp-for="conpassword"></label>
      <input asp-for="conpassword" /><br />
      <span asp-validation-for="conpassword" class="text-danger"></span>
      <br/>
      <label asp-for="channel"></label>
      <input asp-for="channel" /><br />
      <span asp-validation-for="channel" class="text-danger"></span>
      <br/>
      <label asp-for="Age"></label>
      <input asp-for="Age" /><br/>
      <span asp-validation-for="Age" class="text-danger"  ></span>
      <br/>
      <input type="submit" value="Submit" class="btn btn-primary"/>
    </form>
```

Model:

```
 public class Employee
   {
     [Required(ErrorMessage = "Name is Must Fill")]
     [StringLength(15,MinimumLength = 3,ErrorMessage = "Name length must be 3")]
     public string  Name { get; set; }

     [Required]
     [EmailAddress(ErrorMessage = "Email is Must")]
     public string Email{ get; set; }
```

```
        [Required(ErrorMessage = "Age is Must !")]
        [Range(10,50,ErrorMessage = "Must be between 10 to 50")]
        public int? Age { get; set; }
        [Required]
        [RegularExpression(@"(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!*@#$%^&+=]).*$",ErrorMessage =
"Password must be strong!")]
        public string password { get; set; }
        [Required]
        [Compare("password")]
        [Display(Name = "Confirm Password")]
        public string conpassword { get; set; }
        [Required]
        [Url()]
        public string channel { get; set; }

    }
```

Controller:

```
      public IActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public IActionResult Index(Employee e)
        {

            return View();
        }
```

ENTITY FRAMEWORK:

Install 3 packages:

Microsof.entityframeworkcore.sqlserver

Microsof.entityframeworkcore.Tools

Microsof.entityframeworkcore.design

Step1: Install Entity framework Core from manage nuget package.

Step2: Create Model class database Context Class.

Step3: create databse connection in appsetting.json file

Step4: Register connection in Program.cs

Step4: Create Controller with Views.

Step5: Create Migration by using this command “add-migrationInitialCreate”.

Step6: Then type “update-database” command.

Explain:

Step1: is so simple.

Step2:

```
using Microsoft.EntityFrameworkCore;

namespace modelwork.Models
{
    public class StudentContext : DbContext
    {
        public StudentContext(DbContextOptions option) : base(option)
        {

        }

        public DbSet<Student> Students { get; set; }   } }
```

Step3:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
   }
  },
  "ConnectionStrings": {
   "dbcs": "server=MUHAMMAD-FARHAN; database=schoolsys; trusted_connection=true; TrustServerCertificate=true;";
  },
  "AllowedHosts": "*"
}
```

Step4:

```
var provider = builder.Services.BuildServiceProvider();
var config = provider.GetRequiredService<IConfiguration>();
builder.Services.AddDbContext<StudentDbContext>(item =>
item.SqlServer(config.GetConnectionString("dbcs"));
```

Step5:

Add-Migration init

Command: add-migration createdb

Step6:

Update-database

Step7:

Create Controller:

CRUD using Simple steps:

Index show list data:

```
private readonly StudentDbContext studentDbContext;


public HomeController(StudentDbContext studentDbContext)
{
    this.studentDbContext = studentDbContext;
}
public IActionResult Index()
{
    var students = studentDbContext.Students.ToList();


    return View(student);
}
```

Create Student:

```
      [HttpPost]
      public IActionResult Create(Student student)
      {
         if (ModelState.IsValid)
         {
            studentDbContext.Students.add(student);
            studentDbContext.SaveChanges();
            return RedirectToAction("Index","Home");
         }
         return View(student);
      }
```

Details of Students:

```
      public IActionResult Details(int id)
      {
         var student  = studentDbContext.Students.FindorDefault(x => x.id = id);
         return View(student);
      }
```

Database First Approach

Steps:

1. Install Entity Framework
2. Scaffold-DbContext ‘server=ServerName;database=DbName;trusted_connection=true’ Microsoft.Entity.FrameworkCore.SqlServer-OutputDir Models
3. Add Connection string in startup file.

```
Var connection = “”;
Services.AddDbContext<StudentDbContext>(options=>
options.UseSqlServer(connection));
```

4. Add MVC Controller with Views.

How to Update Model Classes.

```
Scaffold-DbContext
“server=ServerName;database=DbName;trusted_connection=true”
Microsoft.Entity.FrameworkCore.SqlServer-OutputDir Models -Force
```

Creating Dropdown list:

```
<div class="form-group">
    @Html.DropDownList("Gender", Html.GetEnumSelectList<Gender>(), "Select Gender", new { @class = "form-control" })
</div>
```

Identity Setup:

1) Download identity Framework.
2) Change DBContext to IdentityDBContext.
3) Service Add:

```
services.AddIdentity<IdentityUser,
IdentityRole>().AddEntityFrameworkStores<bazarContext>();
        4) Add-migration
        5) If Error in Migration so Add in DB-Context
   protected override void OnModelCreating(ModelBuilder modelBuilder)
           {
               base.OnModelCreating(modelBuilder);
                  OnModel.Seed();
        }
```

6)

Sign UP Work

Sign Up Model:

```
public class SignupModel
    {


        [Required]
        [EmailAddress]
        public string Email{ get; set; }
        [Required]
        [DataType(DataType.Password)]
        public string Password{ get; set; }
        [Required]
        [DataType(DataType.Password)]
        [Compare("Password")]
        public string Confirmpassword { get; set; }

    }
```

Controller:

```
   private readonly UserManager<IdentityUser> userManager;
        private readonly SignInManager<IdentityUser> signInManager;

        public AccountController(UserManager<IdentityUser> userManager,
SignInManager<IdentityUser> signInManager)
        {
            this.userManager = userManager;
            this.signInManager = signInManager;
        }


public IActionResult Register()
        {
            return View();
        }
        [HttpPost]
        public async Task<IActionResult> Register(SignupModel model)
        {

            if (ModelState.IsValid)
            {

            var identity = new IdentityUser()
            {
                Email = model.Email,
                UserName = model.Email
            };

                var result = await userManager.CreateAsync(identity, 
model.Password);

                if (result.Succeeded)
                {
                    await signInManager.SignInAsync(identity,isPersistent : false);
                    return RedirectToAction("Index", "Home");
                }

                foreach(var error in result.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }

            return View();
        }
```

Logout:

```
  public IActionResult Logout(){

            signInManager.SignOutAsync();

            return RedirectToAction("Index", "Home");
        }
```

Login Work:

Login Model:

```
public class LoginModel
    {

        [Required]
        [EmailAddress]
        public string Email{ get; set; }
        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

    }
```

Controller:

```
        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }
        [HttpPost]
        public async Task<IActionResult> Login(LoginModel model)
        {


            if (ModelState.IsValid)
            {

                var result = await signInManager.PasswordSignInAsync(model.Email,
model.Password,false,false);

                if (result.Succeeded)
                {
                    return RedirectToAction("Index", "Home");
                }

                ModelState.AddModelError("", "Invalid Credentials !");

            }


            return View(model);
        }
```

Creating Roles:

Steps:

1) Create Controller
2) Create Model CreateRoleModel
3) Add RoleManager<IdentityRole> in Constructor
4) Create Role Method and View in Controller
5) Create Post Role Method

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

        [HttpPost]

        public async Task<IActionResult> Create(CreatingRoleModel model)
        {
            if (ModelState.IsValid)
            {
                IdentityRole user = new IdentityRole()
                {
                    Name = model.RoleName
                };

                IdentityResult result = await roleManager.CreateAsync(user);

                if (result.Succeeded)
                {
                    return RedirectToAction("RolesList", "Administrator");
                }

                foreach(IdentityError error in result.Errors)
                {
                    ModelState.AddModelError("", error.Description);
                }
            }
            return View(model);
        }
```

6) View Roles List Action Method:

```
[HttpGet]
        public IActionResult RolesList()
        {
            var roles = roleManager.Roles;

            return View(roles);
        }
```

7) View Roles List

```
<div class="container mt-3">
    <div class="row">

        @foreach(var role in Model)
        {

        <div class="col-md-12">
```

```
                <p class="bg-light">Role Id : @role.Id</p>

                <p>Role Name : @role.Name</p>

                <a class="btn btn-primary" asp-action="EditRole"
                asp-controller="Administrator" asp-route-
   id="@role.Id">edit</a>
                <a class="btn btn-danger">Delete</a>

            </div>
            }
     </div>

</div>
```

8) Delete Role:

Upload Single Image:

```
var folder = "";
                if (model.pimg != null)
                {
                    folder = "images/";

                    folder  = await MupImages(folder, model.pimg);

                }
```

Method :

```
public async Task<string> MupImages(string folderpath, IFormFile
file)
        {

            folderpath += Guid.NewGuid().ToString() + "_" +
file.FileName;

            string serverfolder =
Path.Combine(_webHostEnvironment.WebRootPath, folderpath);

            await file.CopyToAsync(new FileStream(serverfolder,
FileMode.Create));

            return "/" + folderpath;

        }
```

Upload Multiple Image:

```
                var gfolder = "";

if (model.GalleryImages != null)
                {
                    gfolder = "images/Gallery/";

                    model.Gallery = new List<ProductPictures>();

                    foreach (var file in model.GalleryImages)
                    {
                        var gallery = new ProductPictures()
                        {
                            Imgsrc = await MupImages(gfolder, file),
                            Productid = id
                        };

                        model.Gallery.Add(gallery);

                        await _context.SaveChangesAsync();
                    }


    public async Task<string> MupImages(string folderpath, IFormFile file)
        {

            folderpath += Guid.NewGuid().ToString() + "_" + file.FileName;

            string serverfolder = Path.Combine(_webHostEnvironment.WebRootPath,
folderpath);

            await file.CopyToAsync(new FileStream(serverfolder, FileMode.Create));

            return "/" + folderpath;

        }
```