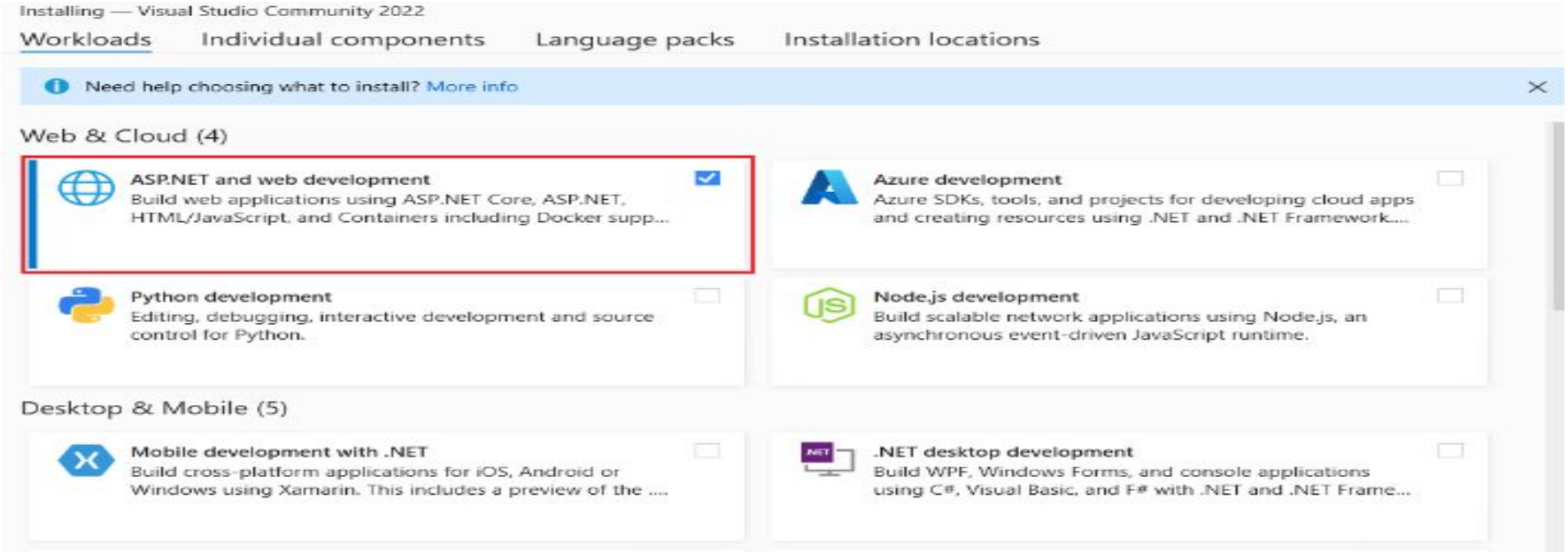
Backend Development Using Asp.Net

Unit-3

Installing Asp.Net and Web Development



Additional information

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

C# Linux macOS Windows Cloud Service Web

Framework

.NET 8.0 (Long Term Support)

Authentication type

None

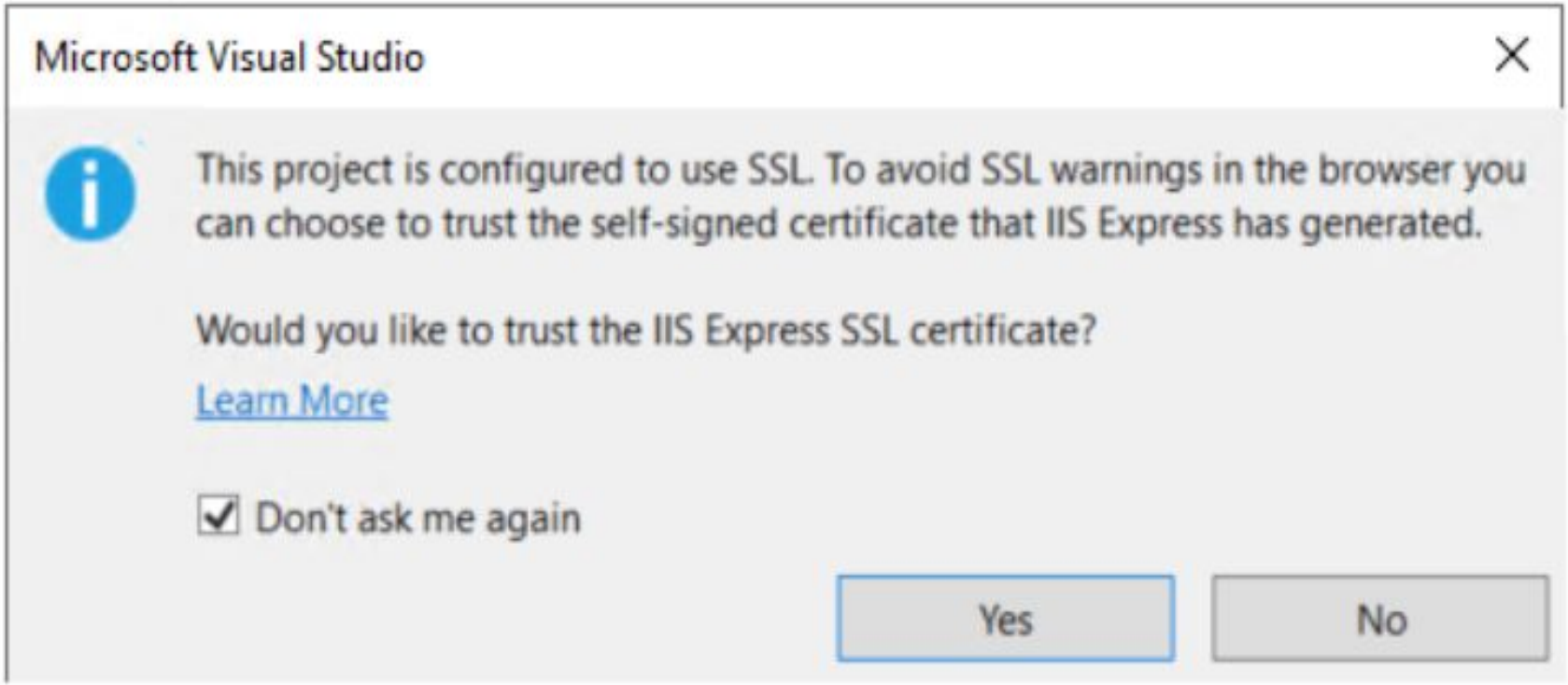
- [x] Configure for HTTPS
- [ ] Enable Docker

Docker OS

Linux

- [ ] Do not use top-level statements

Back Create


Microsoft Visual Studio
This project is configured to use SSL. To avoid SSL warnings in the browser you can choose to trust the self-signed certificate that IIS Express has generated.
Would you like to trust the IIS Express SSL certificate?
Learn More
Don't ask me again
Yes
No

Security Warning

You are about to install a certificate from a certification authority (CA) claiming to represent:

localhost

Windows cannot validate that the certificate is actually from "localhost". You should confirm its origin by contacting "localhost". The following number will assist you in this process:

Thumbprint (sha1): 35AAFB1F 70EC5F89 C7180FBD 61AFE491 94580874

Warning:
If you install this root certificate, Windows will automatically trust any certificate issued by this CA. Installing a certificate with an unconfirmed thumbprint is a security risk. If you click "Yes" you acknowledge this risk.

Do you want to install this certificate?

Yes No

Steps

- Start Visual Studio and select Create a new project.
- In the Create a new project dialog, select ASP.NET Core Web App (Model-View-Controller) > Next.
- In the Configure your new project dialog, enter `MvcMovie` for Project name. It's important to name the project *MvcMovie*. Capitalization needs to match each `namespace` when code is copied.
- Select Next.
- In the Additional information dialog:
    - Select .NET 8.0 (Long Term Support).
    - Verify that Do not use top-level statements is unchecked.
- Select Create.

What is MVC?

MVC-based apps contain:

- Models: Classes that represent the data of the app. The model classes use validation logic to enforce business rules for that data. Typically, model objects retrieve and store model state in a database. In this tutorial, a `Movie` model retrieves movie data from a database, provides it to the view or updates it. Updated data is written to a database.
- Views: Views are the components that display the app's user interface (UI). Generally, this UI displays the model data.
- Controllers: Classes that:
    - Handle browser requests.
    - Retrieve model data.
    - Call view templates that return a response.

Working of a MVC

User Interaction: A user interacts with the application by making a request to a URL, such as clicking a link or submitting a form.

Routing: The ASP.NET MVC routing system maps the incoming URL to a specific controller action method based on the route configuration defined in the application.

Controller Processing: The controller action method receives the request, performs any necessary processing (such as retrieving data from the database), and prepares the data to be displayed.

View Rendering: The controller action method selects an appropriate view to render and passes the data to the view.

HTML Generation: The view generates HTML dynamically based on the data provided by the controller and renders it to the user's browser.

User Interaction (Again): The user sees the rendered HTML in their browser and interacts with the application (e.g., clicking links, submitting forms).

Repeat: The process repeats for each user request, with the controller handling the request,

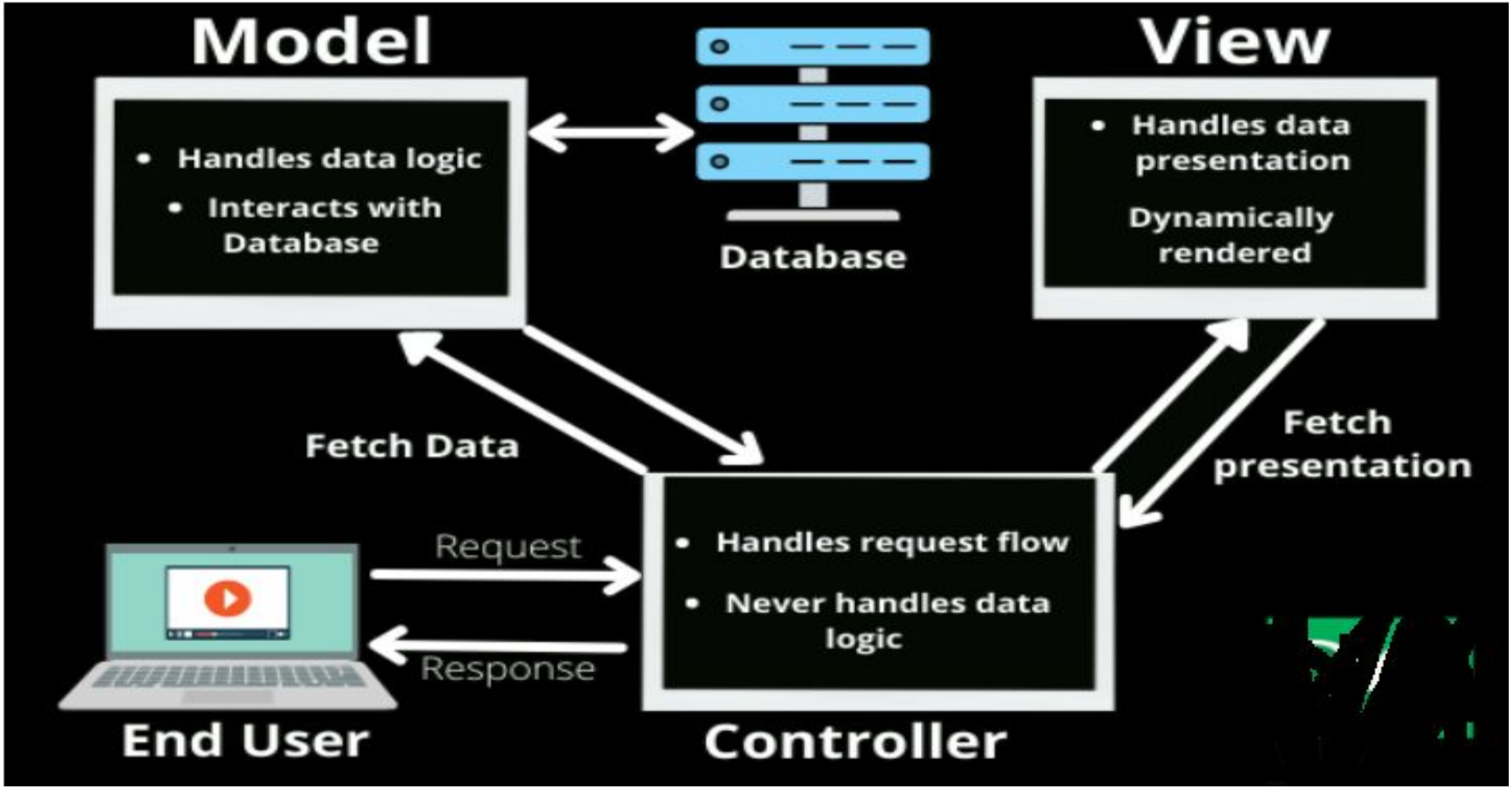
MVC?

MVC is Model, View, and Controller framework.

Model is the business layer of an application. It contains classes and application logic.

View is the front-end or interface through which a user interacts with our application.

Controller is the bridge between Model and View. It is used to handle requests.


Model
Handles data logic
Interacts with Database
Database
View
Handles data presentation
Dynamically rendered
Fetch Data
Fetch presentation
Request
Response
Handles request flow
Never handles data logic
End User
Controller

Communications

Model to Model

We can communicate from Model to Model via parameters / composition.

Model to View

To communicate form Model to View, you have to follow the path: *Model* > *Controller* > *View*

We can’t directly move from Model to View. First, the Model object is made in the Controller and then it is passed to View. We can pass the data or communicate from Model to View by these three steps:

- Take the object in the action of a Controller.
- Pass Model object as a parameter to View.
- Use @model to include Model on the View page.

Model to Controller

Create an object of Model class in Controller to access the Model in Controller.

View to Model

To communicate from View to Model, you have to follow the path: *View > Controller > Model*

You can’t directly move from View to Model. First, you have to submit data to the Controller and then pass it to Model. To pass the data from View to Model, you have to follow these three steps:

- Submit HTML form to a Controller.
- Create an object of Model in Controller.
- Pass values to the Model object.

View to Controller

We can move data from View to Controller by submitting forms from View to specific Controller or by -

- JSON
- AJAX Calls
- JavaScript
- Partial Views

Controller to Model

We can move from Controller to Model just like we move from Model to Controller - by creating an object of Model in Controller.

Controller to View

We can move from Controller to View the following ways:

- By using ViewBag
- ViewData
- TempData

Controller to Controller

We can move from one Controller to another by using RedirectToAction(); and then pass the name of the specific action.

View to view

Partial Views are used.

Razor View

Razor View engine is a markup syntax which helps us to write HTML and server-side code in web pages using C# or VB.NET. It is server-side markup language however it is not at all a programming language.

Razor is a templating engine and ASP.NET MVC has implemented a view engine which allows us to use Razor inside of an MVC application to produce HTML. However, Razor does not have any ties with ASP.NET MVC.

Now, Razor Syntax is compact which minimizes the characters to be used, however it is also easy to learn.

Some of Razor Syntax Rules for C# are given below.

- It must be always enclosed in @{ ... }
- Semicolon “;” must be used to ending statements
- Files have .cshtml extension.
- Variables are declared with var keyword
- Inline expressions (variables and functions) start with @
- C# code is case sensitive

Variables

```
// Using the var keyword:
var greeting = "Welcome to Razor";
var counter = 200;
var day = DateTime.Today;

// Using data types:
string greeting = "Welcome to Razor ";
int counter = 200;
DateTime day = DateTime.Today;
```

Conditions

If statement

It starts with code block and its condition is written in parenthesis. And the code which needs to be executed once condition gets satisfied is written inside braces.

Let’s understand with the below example.

```
@{var price=60;}
<html>
   <body>
      @if (price>50)
       {
          <p>The price is  greater than 50.</p>
       }
   </body>
</html>
```

If – Else statement

It starts with code block and its condition is written in parenthesis. And code which needs to be executed once the condition gets satisfied is written inside braces and if it does not gets satisfied then code written inside else block gets executed.

Let’s understand with the below example.

```
@{var price=60;}
<html>
<body>
@if (price>50)
    {
    <p>The price is  greater than 50.</p>
 }

else
  {
  <p>The price is less than 50.</p>
  }

</body>
</html>
```

How To Create a Controller in Asp.net MVC?

To create a controller in an ASP.NET MVC application, you typically follow these steps:

Open Visual Studio:

> Launch Visual Studio, and either create a new ASP.NET MVC project or open an existing one.

Add a Controller:

> Right-click on the "Controllers" folder within your project in the Solution Explorer. Then, choose "Add" -> "Controller".

Choose Controller Template:

> In the "Add Scaffold" dialog that appears, select "MVC Controller with views, using Entity Framework" or "MVC Controller" depending on your requirements.

Controller Creation

csharp

Copy code

```csharp
using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";

        return View();
    }

    public IActionResult Contact()
    {
        ViewData["Message"] = "Your contact page.";

        return View();
    }
```

Create Corresponding Views

To create corresponding views for the actions in the controller, follow these steps:

Index View:

- Right-click within the `Index()` action method in the controller.
- Select "Add View".
- In the "Add View" dialog, provide a name for the view (e.g., "Index").
- Choose the template for your view. For the `Index()` action, you might select "Empty" or "List".
- Click "Add".

About View:

- Right-click within the `About()` action method in the controller.
- Select "Add View".
- Provide a name for the view (e.g., "About").
- Choose the template as per your requirements.
- Click "Add".

Contact View:

- Right-click within the `Contact()` action method in the controller.
- Select "Add View".
- Provide a name for the view (e.g., "Contact").
- Choose the template.
- Click "Add".

1. **Index.cshtml**:

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

<h2>Welcome to our application!</h2>

<p>This is the home page.</p>
```

1. **About.cshtml**:

```html
@{
    ViewData["Title"] = "About";
}

<h2>About Us</h2>

<p>This is the about page.</p>
```

Layout,Sections and ViewStart

Layout-In the context of ASP.NET MVC or ASP.NET Core MVC, a layout is a shared template or master page that defines the common structure and appearance of multiple views within a web application. It allows you to define the common elements such as headers, footers, navigation bars, and stylesheets in a single file, which are then applied to all views in your application.

Here's how layouts work:

Shared Structure: A layout typically contains the HTML structure that is common across multiple pages of your website. This includes elements like the header, footer, navigation menus, and other common sections.
Content Placeholder: Within the layout, you define one or more content placeholders using the `@RenderBody()` directive (in Razor syntax). These placeholders indicate where the content of individual views should be inserted.
Consistent Appearance: By using a layout, you ensure that all pages in your application have a consistent appearance and structure. Any changes made to the layout are automatically reflected across all pages that use that layout.
Flexible Design: Layouts can also include additional placeholders for sections that may vary from page to page, such as a sidebar or a section for dynamic content. Views can then optionally override these sections as needed.

Creation of the Layout

1. First of all, You have to create a controller- I have created a controller named “HomeController1.cs”.
2. Create three methods named as Index,About and Contact.
3. Then create the corresponding views.
4. In Index.cshtml, write some content in h1 or paragraph tag.
5. In Contact.cshtml, write some content in h1 or paragraph tag.
6. In About.cshtml, write some content in h1 or paragraph tag.
7. We are using Layout, as we want a layout must be created once and it will be common for every action and their corresponding view

Creation of Action Methods and also create views of action methods

WebApplication7 | WebApplication7.Controllers.HomeCont | About()

```
4   {
        0 references
5       public class HomeController1 : Controller
6       {
            0 references
7           public IActionResult Index()
8           {
9               return View();
10          }
            0 references
11          public IActionResult About()
12          {
13              return View();
14          }
            0 references
15          public IActionResult Contact()
16          {
17              return View();
18          }
```

Search Solution Explorer (Ctrl+;)

- Solution 'WebApplication7' (1 of 1 project)
  - WebApplication7
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
      - HomeController1.cs
    - Models
    - Views
      - Home
      - Shared
        - _Layout.cshtml
        - _Layout1.cshtml
        - _ValidationScriptsPartial.cshtml
        - Error.cshtml
      - _Layout.cshtml
      - _ViewImports.cshtml
      - _ViewStart.cshtml
      - _ViewStart1.cshtml

Now, you have to create Layout file

Go inside views/shared and Right click on shared

View Git Project Build Debug Test Analyze Tools Extensions Window Help Search WebApplication7 Sign in

Debug Any CPU https

html About.cshtml Index.cshtml HomeController1.cs _Lay

```
1
2   @{
3       ViewData["Title"] = "Index";
4   }
5
6   <h1>So this is main page.....or we can sa
7
8
```

- View...
- Controller...
- Razor Component...
- New Item... Ctrl+Shift+A
- Existing Item... Shift+Alt+A
- New Scaffolded Item...
- New Folder
- Container Orchestrator Support...
- Docker Support...
- Application Insights Telemetry...
- Client-Side Library...
- Machine Learning Model...
- Class...
- New EditorConfig

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'WebApplication7' (1 of 1 project)
  - WebApplication7
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs

- View in Browser (Microsoft Edge) Ctrl+Shift+W
- Browse With...
- Add
- Collapse All Descendants Ctrl+Left Arrow
- Scope to This

Add New Item - WebApplication7

Installed

C#

General

ASP.NET Core

Search Results

Online

Sort by: Default

Razor Layout C#

layout

Type: C#

Razor View Layout Page

Name: Layout1.cshtml

Now _Layout1.cshtml file has been created

Contact.cshtml | About.cshtml | Index.cshtml | HomeController1.cs | _Layout1.cshtml* | Contact.cshtml

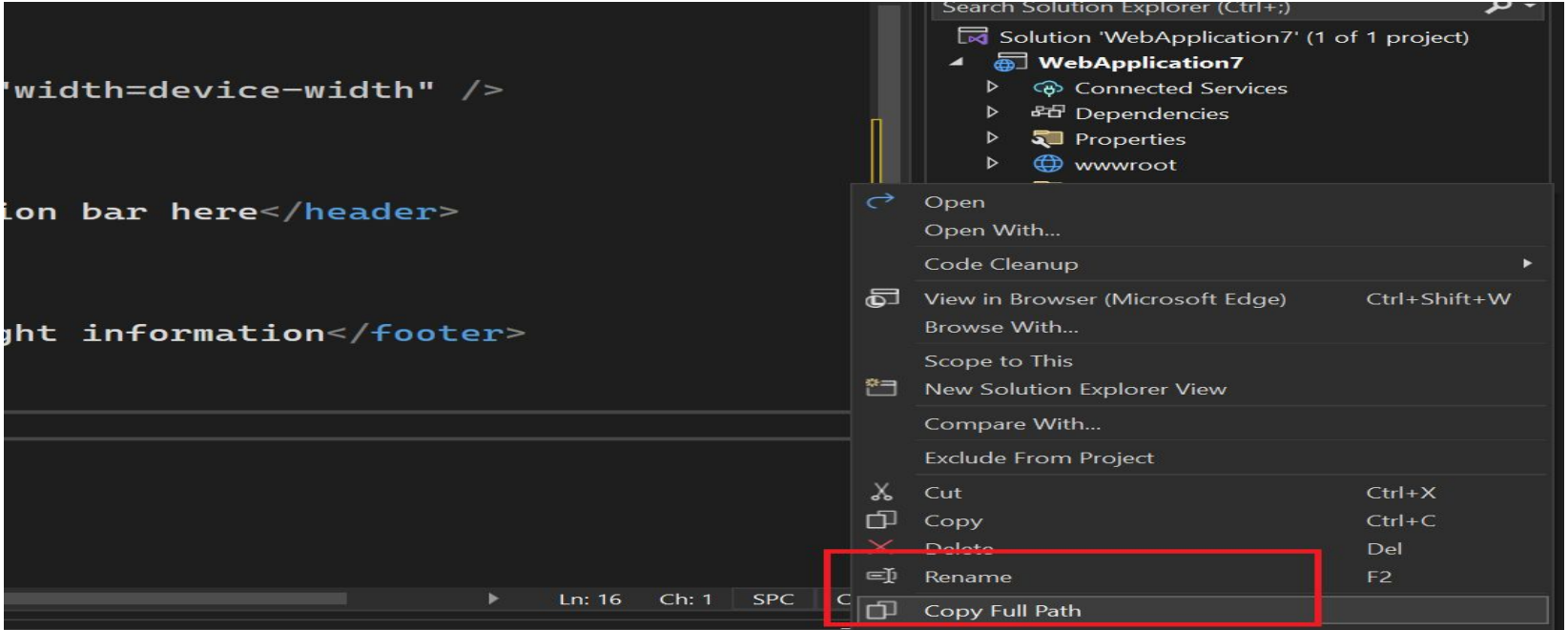
```
1   <!DOCTYPE html>
2
3   <html>
4   <head>
5       <meta name="viewport" content="width=device-width" />
6       <title>@ViewBag.Title</title>
7   </head>
8   <body>
9
10
11      @RenderBody()
12
13  </body>
14  </html>
15
```

Write something above and below @RenderBody

```
1   <!DOCTYPE html>
2
3   <html>
4   <head>
5       <meta name="viewport" content="width=device-width" />
6       <title>@ViewBag.Title</title>
7   </head>
8   <body>
9       <header>You can include navigation bar here</header>
10
11          @RenderBody()
12
13      <footer>You can include copyright information</footer>
14  </body>
15  </html>
16
```

Now you have to link your layout file to all action methods

Right click on _Layout1.cshtml file, then you will be able to see the copy full path option. Copy that path



Paste that path in Index/about/contact view

Layout=”~/Views/Shared/_Layout1.cshtml”

Make sure you have to use / instead of \. So corrected version is in the next slide

About.cshtml | Index.cshtml* | HomeController1.cs | _Layout1.cshtml* | Contact.cshtml | About.cshtml | Index.cshtml

```
@{
    ViewData["Title"] = "Index";
    Layout = "~\Views\Shared\_Layout1.cshtml"
}

<h1>So this is main page.....or we can say this is the index page</h1>
```

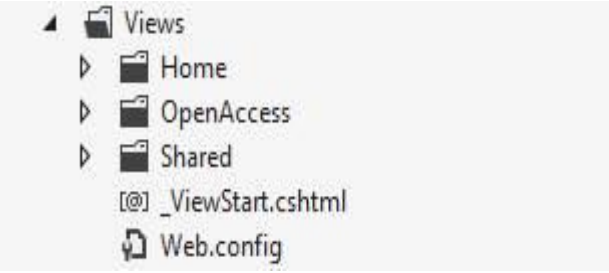
```
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout1.cshtml";
}

<h1>So this is main page.....or we can say this is the index page</h1
```

VIEWSTART

_Viewstate.cshtml plays an important and a tricky role in Razor views. It was introduced in MVC 3 along with Razor views. _Viewstart.cshtml is used to place common UI logic across the Views in the folder, where it is located. This means, the views in a single folder which is having _Viewstart.cshtml will be rendered along with it.

For example: If we observe the views folder of an MVC project, we will see _Viewstart.cshtml in the folder.



viewstart

1. Thus, the views in Home and OpenAccess will be rendered along with the UI in _Viewstart.cshtml. We need not to declare anything in the views. This will be done automatically by the framework.

Benefit

By doing so, we can change the layout at a single place only. Otherwise, we have to change the number of views.

HTML TAG HELPERS

In MVC, developers normally do not use any web content for their applications. Because, Microsoft introduced three helper objects (HtmlHelper, UrlHelper, and AjaxHelper) for generating web control in the application. These helper objects simply shorten the work of the developer for designing any application of web interface. In the MVC pattern, all the code of Razor views (including server-side) starts with the @ sign. In this way, the MVC framework always has a clear separation between the server-side code and client-side code.

Why Tag Helpers?

Microsoft introduced a new feature in the MVC Razor engine with the release of ASP.NET Core which is known as Tag Helpers. This feature helps web developers use their old conventional HTML tags in their web applications for designing presentation layers.

So, with the help of Tag Helpers, developers can replace the Razor cryptic syntax with the @ symbol, a more natural-looking HTML-like syntax. So, the first question always arises “Why do we need Tag Helpers?”. The simple answer is that Tag Helpers reduce the coding amount in HTML which we need to write and also create an abstracted layer between our UI and server-side code. We can extend the existing HTML tag elements or create custom elements just like HTML elements with the help of Tag Helpers.

Difference between HTML helpers and Tag Helpers

```
// HTML Helpers

@Html.ActionLink("Click", "Controller1", "CheckData", { @class="my-css-classname",
data_my_attr="my-attribute"})


//Tag Helpers

<a asp-controller="Controller1" asp-action="CheckData" class="my-css-classname"
my-attr="my-attribute">Click</a>
```

Built In Tag Helpers

The Anchor Tag Helpers extend the standard HTML anchor tag (<a>..) which provides some new attributes. By standard convention, the attribute name must start with asp-. This helper object contains the following attributes –

1. asp-controller - This attribute assigns the controller name which is used for generating the URL.
2. asp-action - This attribute is used to specify the controller action method name. If no value is assigned against this attribute name, then the default asp-action value in the controller will execute to render the view.
3. asp-route-{value} - This attribute enables a wildcard-based route prefix. Any value specified in the {value} placeholder is interpreted as a route parameter.
4. asp-route - This attribute is used for creating a direct URL linking to an actual route value.

Ways of linking

_ViewImports.cshtml | Program.cs | **Index.cshtml*** | Contact.cshtml | HomeController1.cs | HomeController.cs | WebApplication15: Overview

```
1
2   @{
3       ViewData["Title"] = "Index";
4   }
5
6   <h1>Index</h1>
7   <a href="/HomeController1/Contact">Contact-Hyperlink first way</a>
8   <br />
9   @Html.ActionLink("Contact Page 2","Contact","HomeController1");
10  <br />
11  <a href=" @Url.Action("Contact","HomeController1")">Third way</a>
12  <br />
13  <h2>Tag Helpers</h2>
14  <a asp-controller="HomeController1" asp-action="Contact">Contact Page 4</a>
```

Data Passing Techniques-ViewBag and View Data,Working with TempData

ViewBag, ViewData, and TempData are all objects in ASP.NET MVC, and these are used to pass the data in various scenarios.

The following are the scenarios where we can use these objects.

1. Pass the data from Controller to View.
2. Pass the data from one action to another action in the same Controller.
3. Pass the data in between Controllers.
4. Pass the data between consecutive requests

What is ViewBag?

ViewBag is a dynamic object to passes the data from the Controller to View. This will pass the data as a property of the object ViewBag. And we have no need to typecast to read the data or for null checking.

In controller

```
Public ActionResult Index()

{

    ViewBag.Title = “Welcome”;

    return View();

}
```

In view

Example View

```
<h2>@ViewBag.Title</h2>
```

What is ViewData?

ViewData is a dictionary object to pass the data from Controller to View, where data is passed in the form of a key-value pair. Typecasting is required to read the data in View if the data is complex, and we need to ensure a null check to avoid null exceptions. The scope of ViewData is similar to ViewBag, and it is restricted to the current request, and the value of ViewData will become null while redirecting.

Example Controller

```
Public ActionResult Index() { ViewData["Title"] = "Welcome"; return View(); }
```

C#Copy

Example View

```
<h2>@ViewData["Title"] </h2>
```

What is TempData?What is TempData?

TempData is a dictionary object to passes the data from one action to another action in the same Controller or different Controllers. Usually, the TempData object will be stored in a session object. Tempdata is also required to typecast and for null checking before reading data from it. TempData scope is limited to the next request, and if we want TempData to be available even further, we should use Keep and Peek.

Example Controller

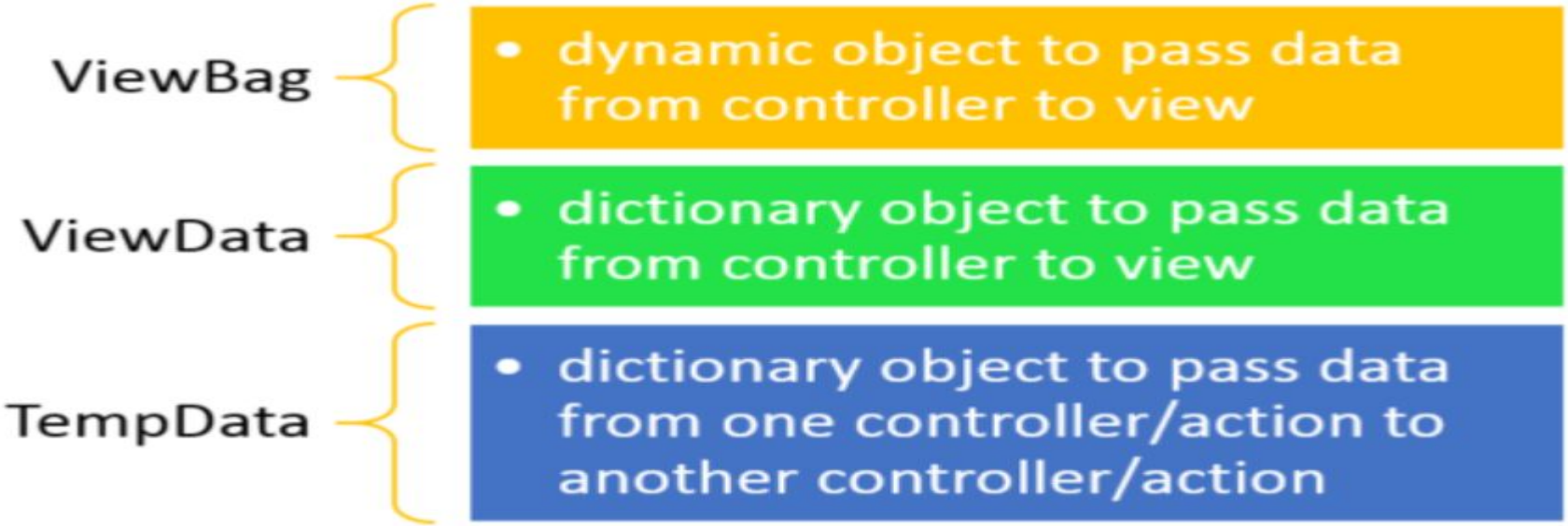
```
Public ActionResult Index() { TempData["Data"] = "I am from Index action"; return
View(); } Public string Get() { return TempData["Data"] ; }
```

SIMILARITIES AND DIFFERENCES BETWEEN VIEW DATA AND VIEW BAG IN MVC

SIMILARITIES

- Both ViewData and ViewBag are used to pass data from a controller to a view.
- Both Helps to maintain data when you move from controller to view.
- Short life means value becomes null when redirection occurs.
- ViewData and ViewBag are DataDictionary objects.
- Both ViewData and ViewBag does not provide compile time error checking. For Example- if you mis-spell keys you wouldn't get any compile time errors. You get to know about the error only at runtime.

ViewData, ViewBag, and TempData are used to pass data between controller, action, and views. To pass data from the controller to view, either ViewData or ViewBag can be used. To pass data from one controller to another controller,



Example of viewdata

Modelclass.cs | Index.cshtml | DataPassing.cs* | _ViewImports.cshtml | Program.cs | Index.cshtml | Contact.csh

WebApplication15 | WebApplication15.Controllers.DataPassing | Index()

```
1   using Microsoft.AspNetCore.Mvc;
2   using WebApplication15.Models;
3
4   namespace WebApplication15.Controllers
5   {
        0 references
6       public class DataPassing : Controller
7       {
            0 references
8           public IActionResult Index()
9           {
10              Modelclass model = new Modelclass()
11              {
12                  Id = 1,
13                  Name = "Palvi",
14                  Designation = "Assistant Professor",
15                  Experience = 3
16              };
17              ViewData["myvalues"]=model;
18              return View();
19          }
```

Model class

Modelclass.cs | Index.cshtml | **DataPassing.cs*** | _ViewImports.cshtml | Program.cs | Index.cshtml

WebApplication15 | WebApplication15.Controllers.DataPassing

```
1   using Microsoft.AspNetCore.Mvc;
2   using WebApplication15.Models;
3
4   namespace WebApplication15.Controllers
5   {
        0 references
6       public class DataPassing : Controller
7       {
            0 references
8           public IActionResult Index()
9           {
10              Modelclass model = new Modelclass()
11              {
12                  Id = 1,
13                  Name = "Palvi",
14                  Designation = "Assistant Professor",
```

Views

Modelclass.cs | **Index.cshtml** | DataPassing.cs* | _ViewImports.cshtml | Program.cs

```
1
2   @{
3       ViewData["Title"] = "Index";
4       var a = ViewData["myvalues"] as Modelclass;
5   }
6   @{
7       if(a!=null)
8       {
9           <h2>@a.Id</h2>
10          <h2>@a.Name</h2>
11          <h2>@a.Designation</h2>
12          <h2>@a.Experience</h2>
13      }
14  }
15
16
17
```

ViewBag-Controller

odelclass.cs | Index.cshtml* | DataPassing.cs* | _ViewImports.cshtml | Program.cs | Index.cshtml | Contact.cshtml

WebApplication15 | WebApplication15.Controllers.DataPassing | Index()

```
1   using Microsoft.AspNetCore.Mvc;
2   using WebApplication15.Models;
3
4   namespace WebApplication15.Controllers
5   {
        0 references
6       public class DataPassing : Controller
7       {
            0 references
8           public IActionResult Index()
9           {
10              Modelclass model = new Modelclass()
11              {
12                  Id = 1,
13                  Name = "Palvi",
14                  Designation = "Assistant Professor",
15                  Experience = 3
16              };
17          //   ViewData["myvalues"]=model;
18          ViewBag.myvalues= model;
19              return View();
20          }
21      }
```

ViewBag-Views

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search WebApplication15 Sign in

Debug Any CPU Continue

Process: [8] WebApplication15.exe Lifecycle Events Thread: Stack Frame:

Modelclass.cs | Index.cshtml | DataPassing.cs* | _ViewImports.cshtml | Program.cs | Index.cshtml | Contact.cshtml | HomeController1.cs

```
1
2   @{
3       ViewData["Title"] = "Index";
4      // var a = ViewData["myvalues"] as Modelclass;
5       var a = ViewBag.myvalues as Modelclass;
6   }
7   @{
8       if(a!=null)
9       {
10          <h2>@a.Id</h2>
11          <h2>@a.Name</h2>
12          <h2>@a.Designation</h2>
13          <h2>@a.Experience</h2>
14      }
15  }
16
17
18
19
```

Output

Output Error List

Solution Explorer Git Changes

Viewbag-model

Modelclass.cs | Index.cshtml | DataPassing.cs | _ViewImports.cshtml | Program.cs | Index.cshtml | Contact.cshtml | HomeController1.cs

WebApplication15 | WebApplication15.Models.Modelclass | Name

```
namespace WebApplication15.Models
{
    3 references
    public class Modelclass
    {
        2 references
        public int Id { get; set; }
        2 references
        public string ? Name { get; set; }
        2 references
        public string ? Designation { get; set; }
        2 references
        public int Experience { get; set; }
    }
}
```

For a list of items-Controller-Viewdata

Modelclass.cs | Index.cshtml* | DataPassing.cs | _ViewImports.cshtml | Program.cs | Index.cshtml | Contact.cshtml | HomeController1.cs

WebApplication15 | WebApplication15.Controllers.DataPassing | Index()

```
0 references
public class DataPassing : Controller
{
    0 references
    public IActionResult Index()
    {
        var a = new List<Modelclass>

        {

            new Modelclass{Id=1,Name = "palvi", Designation = "AP",Experience = 3 },
              new Modelclass{Id=1,Name = "palvi", Designation = "AP",Experience = 3 },
                new Modelclass{Id=1,Name = "palvi", Designation = "AP",Experience = 3 }
        };
        ViewData["myvalues"] = a;


        return View();
```

Contd views

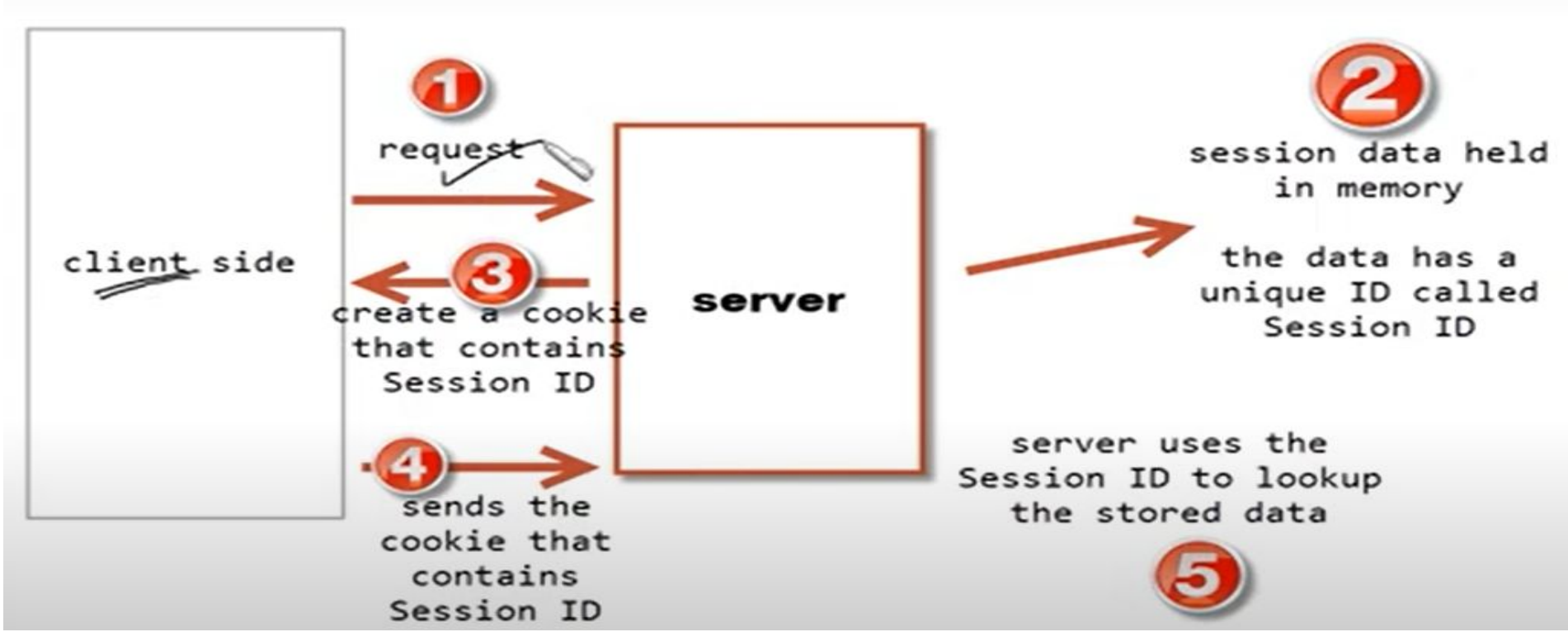
Modelclass.cs | **Index.cshtml*** | DataPassing.cs | _ViewImports.cshtml | Program.cs | Index.cshtm

```
1
2   @{
3       ViewData["Title"] = "Index";
4       // var a = ViewData["myvalues"] as Modelclass;
5       var a = ViewData["myvalues"] as List<Modelclass>;
6   }
7   @{
8
9       if(a!=null)
10      {
11          foreach(var item in a)
12          {
13              <h2>@item.Id</h2>
14              <h2>@item.Name</h2>
15              <h2>@item.Designation</h2>
16              <h2>@item.Experience</h2>
17          }
18          /* if(a!=null)
```

SESSIONS AND STATE MANAGEMENT

- Sessions provide a way to store user-specific data across requests.
- A session is a way to persist data between requests for the same user,
- even if they close their browser or leave the application.
- When user visits a website, server creates a unique session ID, and sends it to client in cookie.
- The client then includes this session ID in subsequent requests,
- allowing the server to associate the requests with a specific session.
- ASP.NET Core provides a built-in ***ISession*** interface that represents a session object.

Session-Contd



Basic Syntax to create session

In the controller, the session can be created inside your action method

- In the below example, we are setting a string value "**John**" in the session with key "**UserName**".

```
public IActionResult Index()
{
    HttpContext.Session.SetString("UserName", "John");
    return View();
}
```

Session-Open Program.cs file

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();
```

Add Session

Here you have to write the following lines of codes between builder.services.AddControllerwithViews and Var app=builder.Build

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

// Add session services
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(20);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

var app = builder.Build();
```

```
// Add session services
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(20);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});
```

Explanation

`builder.Services.AddSession(...)`: This line adds session support to the ASP.NET Core application's service collection. `builder` typically refers to an `IServiceCollection` object, which is used to register application services.

`options => {...}`: This is a lambda expression used to configure the session options.

`options.IdleTimeout = TimeSpan.FromSeconds(20);`: This line sets the idle timeout for the session. The session will expire if there is no activity within 20 seconds.

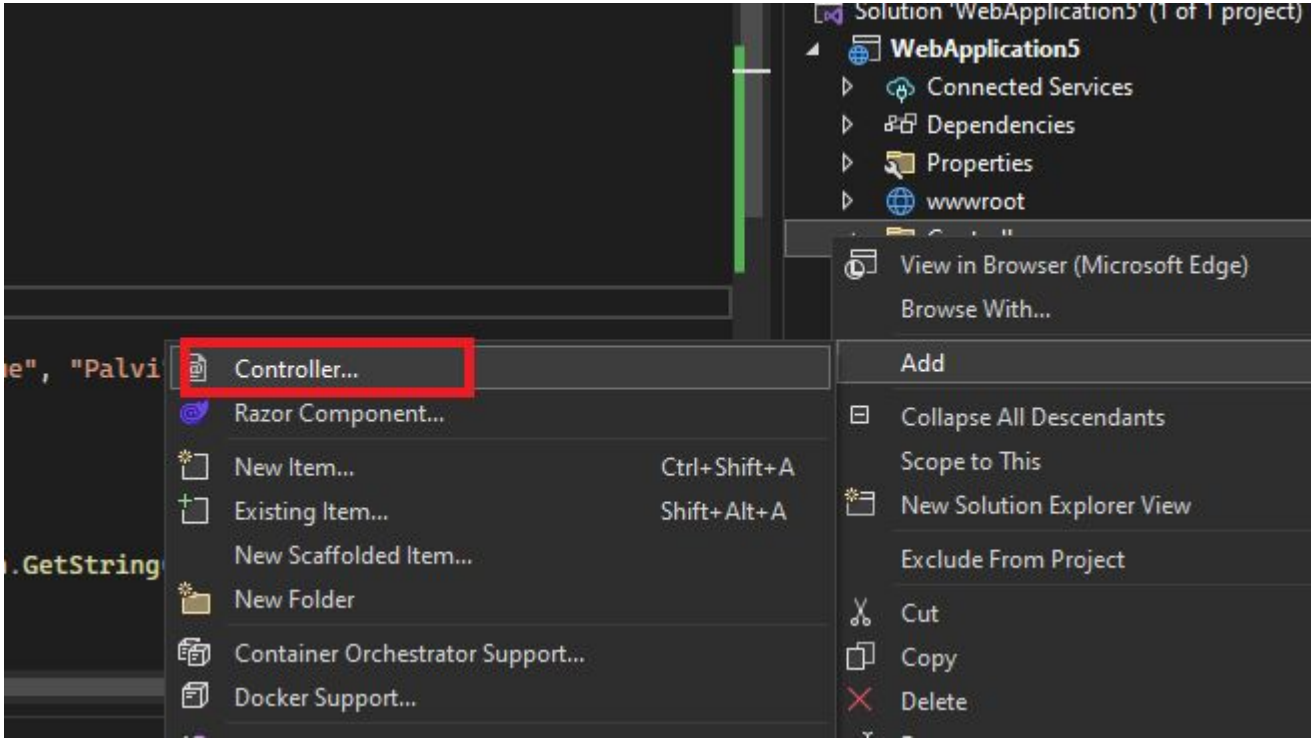
`options.Cookie.HttpOnly = true;`: This line sets the HttpOnly attribute of the session cookie to true. HttpOnly is a flag that, when set, prevents client-side scripts from accessing the cookie, thus enhancing security by mitigating certain types of XSS (cross-site scripting) attacks.

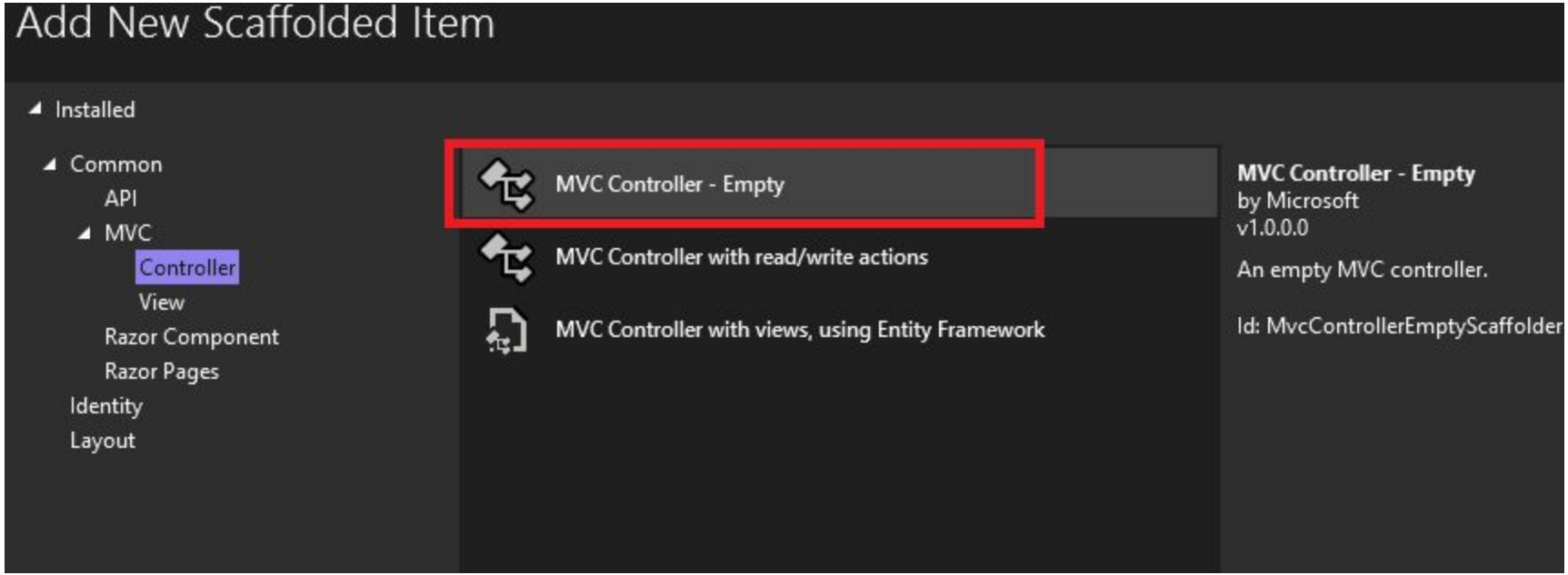
`options.Cookie.IsEssential = true;`: This line sets the IsEssential property of the session cookie to true. This indicates that the cookie is essential for the operation of the application, and as such, consent for its use is not required according to GDPR regulation

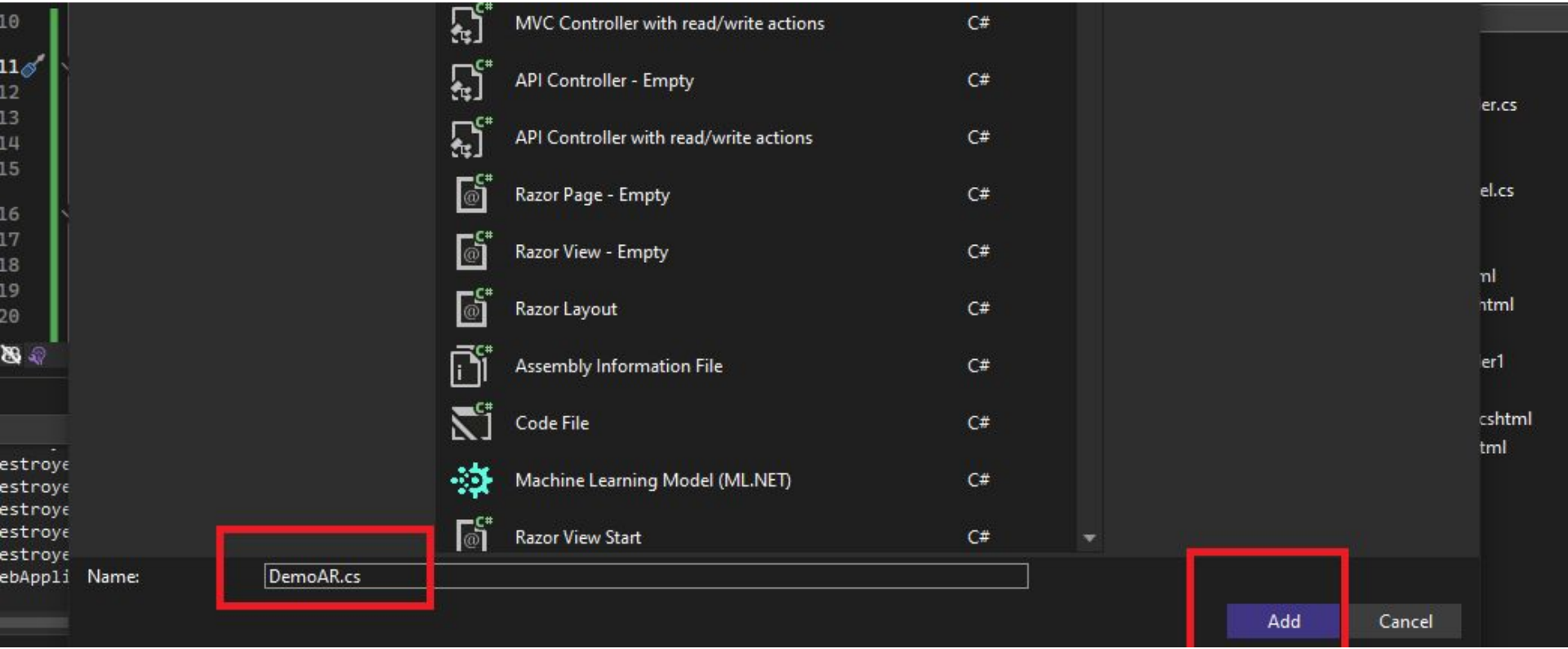
Add app.UseSession(); below app.UseAuthorization()

```
24  app.UseHttpsRedirection();
25  app.UseStaticFiles();
26
27  app.UseRouting();
28
29  app.UseAuthorization();
30  app.UseSession();
31
32  app.MapControllerRoute(
33      name: "default",
34      pattern: "{controller=Home}/{action=Index}/{id?}");
35
36  app.Run();
37
```

Now we need to create a Controller -DemoAR.cs




Add New Scaffolded Item
Installed
Common
API
MVC
Controller
View
Razor Component
Razor Pages
Identity
Layout
MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
MVC Controller - Empty
by Microsoft
v1.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder


MVC Controller with read/write actions C#
API Controller - Empty C#
API Controller with read/write actions C#
Razor Page - Empty C#
Razor View - Empty C#
Razor Layout C#
Assembly Information File C#
Code File C#
Machine Learning Model (ML.NET) C#
Razor View Start C#
Name:
DemoAR.cs
Add
Cancel

low, on DemoAR.cs controller, you have to create three Action methods called

ogin, Success and Logout.

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication5.Controllers
{
    0 references
    public class DemoAR : Controller
    {
        0 references
        public IActionResult Login()
        {
            return View();
        }
        0 references
        public IActionResult Success()
        {
            return View();
        }
        0 references
        public IActionResult Logout()
        {
            return View();
        }
    }
```

Now Add Corresponding View to every Action method

Tabs

WebApplication5
Su...ml

WebApplication5
Account.cs
Accou...shtml
Accounts.cs
Demo.cs
DemoAR.cs*
Index.cshtml
Program.cs
Succe...cshtml

[Tool Windows]
What's New?

WebApplication5 | WebApplication5.Controllers.DemoAR | Login()

```
1   using Microsoft.AspNetCore.Mvc;
2
3   namespace WebApplication5.Controllers
4   {
        0 references
5       public class DemoAR : Contro
6       {
            0 references
7           public IActionResult Log
8           {
9               return View();
10          }
            0 references
11          public IActionResult Suc
12          {
13              return View();
14          }
            0 references
15          public IActionResult Log
16          {
17              return View();
18          }
19      }
20  }
```

100 % | No issues found | Ln: 7 | Ch: 33 | SPC

Add View...	
Go To View	Ctrl+M, Ctrl+G
Quick Actions and Refactorings...	Ctrl+.
Rename...	Ctrl+R, Ctrl+R
Remove and Sort Usings	Ctrl+R, Ctrl+G
Peek Definition	Alt+F12
Go To Definition	F12
Go To Base	Alt+Home
Go To Implementation	Ctrl+F12
Find All References	Shift+F12
View Call Hierarchy	Ctrl+K, Ctrl+T
Track Value Source	
Create Unit Tests	
Breakpoint	
Run To Cursor	Ctrl+F10

Now Setstring must be deployed for setting the session data

```
0 references
public class DemoAR : Controller
{
    0 references
    public IActionResult Login()
    {
        HttpContext.Session.SetString("Username", "Palvi");
        return RedirectToAction("Success");
    }
    0 references
    public IActionResult Success()
    {
        return View();
    }
    0 references
    public IActionResult Logout()
    {
        return View();
```

Use ViewBag in Success Action Method for getting the value of username

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication5.Controllers
{
    0 references
    public class DemoAR : Controller
    {
        0 references
        public IActionResult Login()
        {
            HttpContext.Session.SetString("Username", "Palvi");
            return RedirectToAction("Success");
        }
        0 references
        public IActionResult Success()
        {
            //use viewbag to pass value
            ViewBag.Username = HttpContext.Session.GetString("Username");
            return View();
        }
        0 references
        public IActionResult Logout()
        {
            return View();
```

```
Output
Show output from: Code Generation Log
Finding the generator 'view'...
Running the generator 'view'...
Added View : \Views\DemoAR\Success.cshtml
```

Go to corresponding view of Success

WebApplication5 | WebApplication5.Controllers.DemoAR | Success()

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication5.Controllers
{
    0 references
    public class DemoAR : Control
    {
        0 references
        public IActionResult Logi
        {
            HttpContext.Session.S
            return RedirectToActi
        }
        0 references
        public IActionResult Succ
        {
            //use viewbag to pass
            ViewBag.Username = Ht
            return View();
        }
        0 references
        public IActionResult Logo
```

Add View...	
Go To View	Ctrl+M, Ctrl+G
Quick Actions and Refactorings...	Ctrl+.
Rename...	Ctrl+R, Ctrl+R
Remove and Sort Usings	Ctrl+R, Ctrl+G
Peek Definition	Alt+F12
Go To Definition	F12
Go To Base	Alt+Home
Go To Implementation	Ctrl+F12
Find All References	Shift+F12
View Call Hierarchy	Ctrl+K, Ctrl+T
Track Value Source	
Create Unit Tests	

Access the value of Username via ViewBag in success view

Tabs

WebApplication5

WebApplication5
Su...ml

WebApplication5
Account.cs
Accou...shtml
Accounts.cs
Demo.cs
DemoAR.cs*
Index.cshtml
Login.cshtml
Program.cs
Succe...cshtml

```
1
2   @{
3       ViewData["Title"] = "Success";
4   }
5
6   <h1>Success</h1>
7
8   @ViewBag.Username
9
10
```

Add Tag Helper to Logout Action Method in the Success View

Tabs

WebApplication5

WebApplication5
Su...ml

WebApplication5
Account.cs
Accou...shtml
Accounts.cs
Demo.cs
DemoAR.cs*
Index.cshtml
Login.cshtml
Program.cs
Succe...cshtml
Succe...shtml

[Tool Windows]

```
1
2   @{
3       ViewData["Title"] = "Success";
4   }
5
6   <h1>Success</h1>
7
8   @ViewBag.Username
9
10  <a asp-controller="DemoAR" asp-action="Logout">Logout</a>
```

Output

After 20 seconds of session, data passed in viewbag disappeared

You can use Logout on this session and redirect to some another method called bye

Create a corresponding view of bye and write something as per your

```
    0 references
    public IActionResult Logout()
    {
        HttpContext.Session.Remove("Username");
        return RedirectToAction("Bye");
    }
    0 references
    public IActionResult Bye()
    {
        return View();
    }
}
```

QueryString-Data Passing

QUERYSTRING

- A query string is a collection of characters input to a computer or web browser.
- A Query String is helpful when we want to transfer a value from one page to another.
- When we need to pass content between the HTML pages or aspx Web Forms,
- A Query String is very easy to use and the Query String follows a separating character, usually a Question Mark (?).
- It is basically used for identifying data appearing after this separating symbol.
- **Syntax:**

```
Request.QueryString(variable)[(index).count].
```

Create Action Method for Query String in DemoAR.cs Controller

Now create corresponding view to the Querytest method

```
    public IActionResult Querytest()
    {

        return View();
    }
}
```

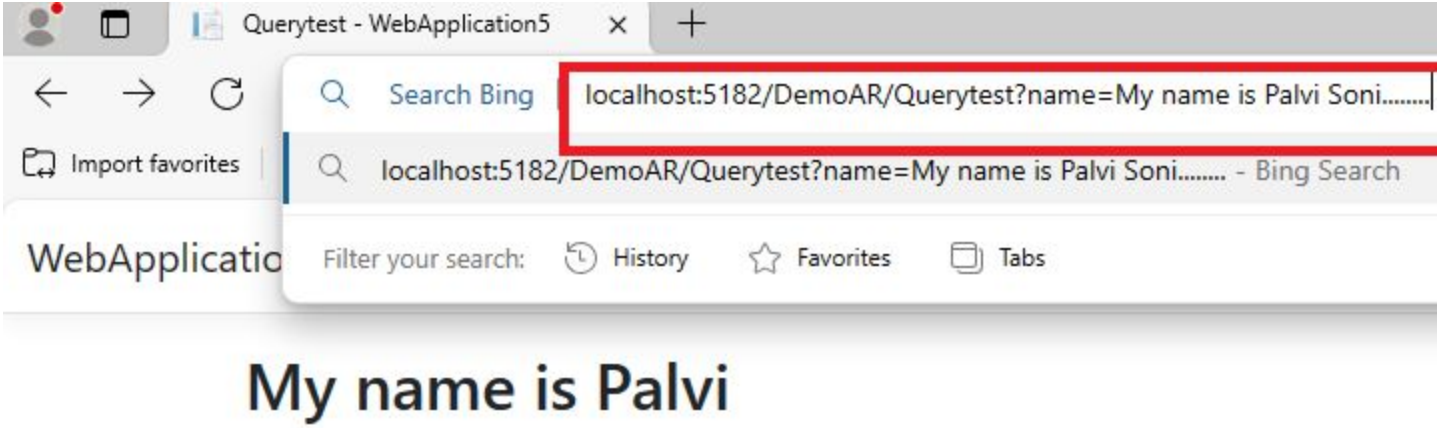
QueryString

Pass value in string having variable name

```
    }
    0 references
    public IActionResult Querytest()
    {
        string name = "My name is Palvi";
        if (!String.IsNullOrEmpty(HttpContext.Request.Query["name"]))
            name = HttpContext.Request.Query["name"];
        ViewBag.Name = name;
        return View();
    }
}
```

Output of QueryString

If you want to reassign the name



Updated output

Steps to Create Session-Program.cs

Introduction To Asp.Net Core Forms

Weakly Typed Forms-In weakly typed forms, you manually write HTML code for form elements like input fields, dropdowns, etc., without any direct connection to the model properties. For example:

```
<form action="/Controller/Action" method="post">

    <input type="text" name="username" />

    <input type="password" name="password" />

    <button type="submit">Submit</button>

</form>
```

Weakly Typed Form in Controller

In the controller action, you would retrieve form data using `Request.Form` or `Request["FieldName"]`.

```
[HttpPost]
public ActionResult Action()
{
    string username = Request.Form["username"];
    string password = Request.Form["password"];

    // Process form data
}
```

Drawbacks of Using Weakly Typed Forms in Asp.Net Mvc

Weakly typed forms are straightforward but may lead to potential issues such as spelling mistakes in field names, missing fields, and manual conversion of data types.

Action Method for Weakly Typed Login

After creating action method create a corresponding view

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

Create Another Method of Login Post

```
}
0 references
public IActionResult WeaklyTypedLogin()
{
    return View();
}
[HttpPost]
0 references
public IActionResult LoginPost(string Username, string Password)
{
    ViewBag.user = Username;
    ViewBag.pass = Password;
    return View();
}
```

Corresponding View of Weakly Typed Method

Tabs

WebApplication5

WebApplication5
Su...ml

WebApplication5
Account.cs
Accou...shtml
Accounts.cs
Bye.cshtml
Demo.cs
DemoAR.cs
Index.cshtml
Login.cshtml
Login...cshtml
Program.cs
Query...shtml
Succe...cshtml
Succe...cshtml
Weak...shtml

```
1
2   @{
3       ViewData["Title"] = "WeaklyTypedLogin";
4   }
5
6   <h1>WeaklyTypedLogin</h1>
7   @using(Html.BeginForm("LoginPost","DemoAR",FormMethod.Post))
8   {
9       <div>
10          Username:@Html.TextBox("Username")
11      </div>
12      <div>
13          Password:@Html.TextBox("Password")
14      </div>
15      <div>
16          <input type="submit"  value="login"/>
17      </div>
18  }
19  }
20
21
```

Corresponding View of LoginPost Method

WebApplication5

```
1
2   @{
3       ViewData["Title"] = "LoginPost";
4   }
5
6   <h1>Welcome to dashboard @ViewBag.user @ViewBag.Pass</h1>
7
8
```

Strongly Typed Form

WebApplication25.Models.Account

```
namespace WebApplication25.Models

{
    4 references
    public class Account
    {
        2 references
        public int Id { get; set; }
        2 references
        public string ?Password { get; set; }
    }
}
```

```
    For more information on enabling MVC for empty projects, visi
*@
@{
}
@using(Html.BeginForm("LoginSuccess","STF",FormMethod.Post))
{
    <div>
        Username : @Html.TextBoxFor(m=>m.Username)
    </div>
    <div>
    Password : @Html.TextBoxFor(m => m.Password)

    </div>
    <input type="submit" value="login" />
```

LoginSuccess.cshtml | Program.cs | appsettings.json | STV.cshtml | LoginView.cs

```
1
2   @{
3       ViewData["Title"] = "LoginSuccess";
4   }
5
6   <h1>LoginSuccess</h1>
7   @ViewBag.Username
8   @ViewBag.Password
9
10
```

```
public class STF : Controller
{
    0 references
    public IActionResult STV()
    {
        return View();
    }
    [HttpPost]
    0 references
    public IActionResult LoginSuccess(LoginView v)
    {
    ViewBag.Username=v.Username;
        ViewBag.Password = v.Password;
        return View();
    }
}
```

WebApplication25 Home Privacy

Username :

Password :

login

WebApplication25 Home Privacy

LoginSuccess

palvi 123456

Model Binding

MODEL BINDING

- This is a mechanism that extracts the data from an HTTP request and provides them to the controller action method parameters.
- The action method parameters may be simple types:
  - like integers, strings, etc., or complex types such as Student, Order, Product, etc.
- When a client makes a request to a controller action method,
- ASP.NET Core automatically maps the data in the request to the action method's parameters using model binding.
- Model binding is a powerful feature that can simplify your code and reduce boilerplate.

Create a model class

LoginViewModel class must be created, username and password must be passed there

Tabs

WebApplication8

HomeController.cs

HomeController1.cs

LoginViewModel.cs

Program.cs

Userdetails.cshtml

WebApplication8: Overview

WebApplication8 | WebApplication8.Models.LoginViewModel | Use

```
1  namespace WebApplication8.Models
2  {
       4 references
3      public class LoginViewModel
4      {
           2 references
5          public string? Username { get; set; }
           2 references
6          public string ?Password { get; set; }
7      }
8  }
9
```

Create a new controller and create an action method

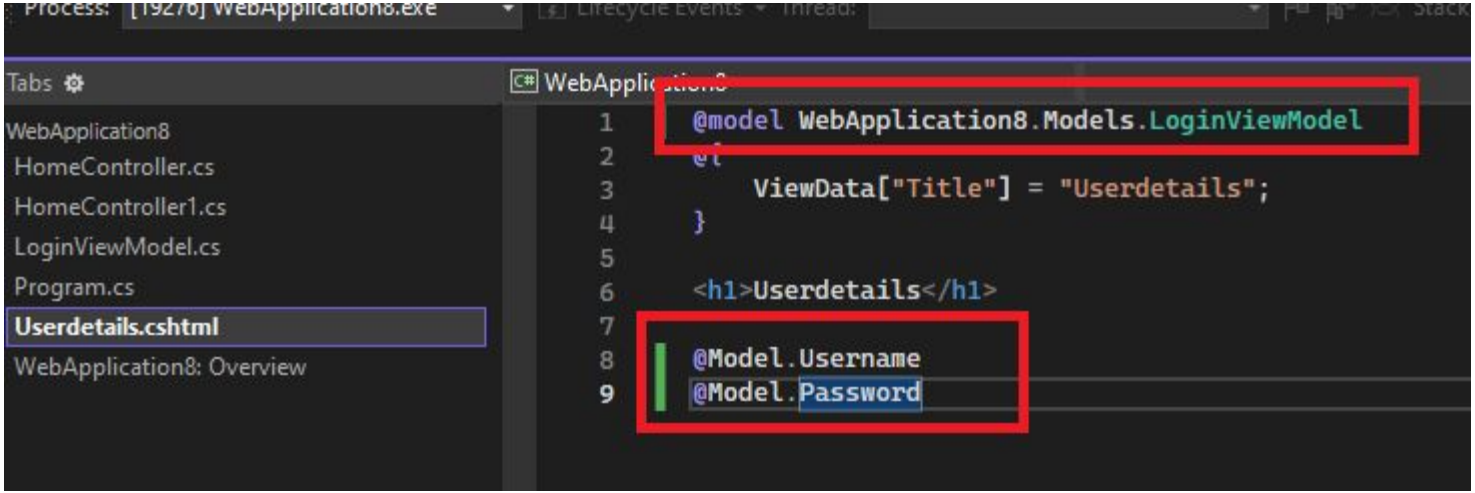
Here, you're creating an instance of the `LoginViewModel` class. This class is assumed to be a model class that holds data related to a user's login information.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using WebApplication8.Models;

namespace WebApplication8.Controllers
{
    0 references
    public class HomeController1 : Controller
    {
        0 references
        public IActionResult Userdetails()
        {
            var user = new LoginViewModel()
            {
                Username = "Palvi",
                Password = "123"

            };
            return View(user);
        }
    }
}
```

`@model WebApplication8.Models.LoginViewModel` : This directive specifies the model type for this Razor view, indicating that it expects to receive an instance of `LoginViewModel`.

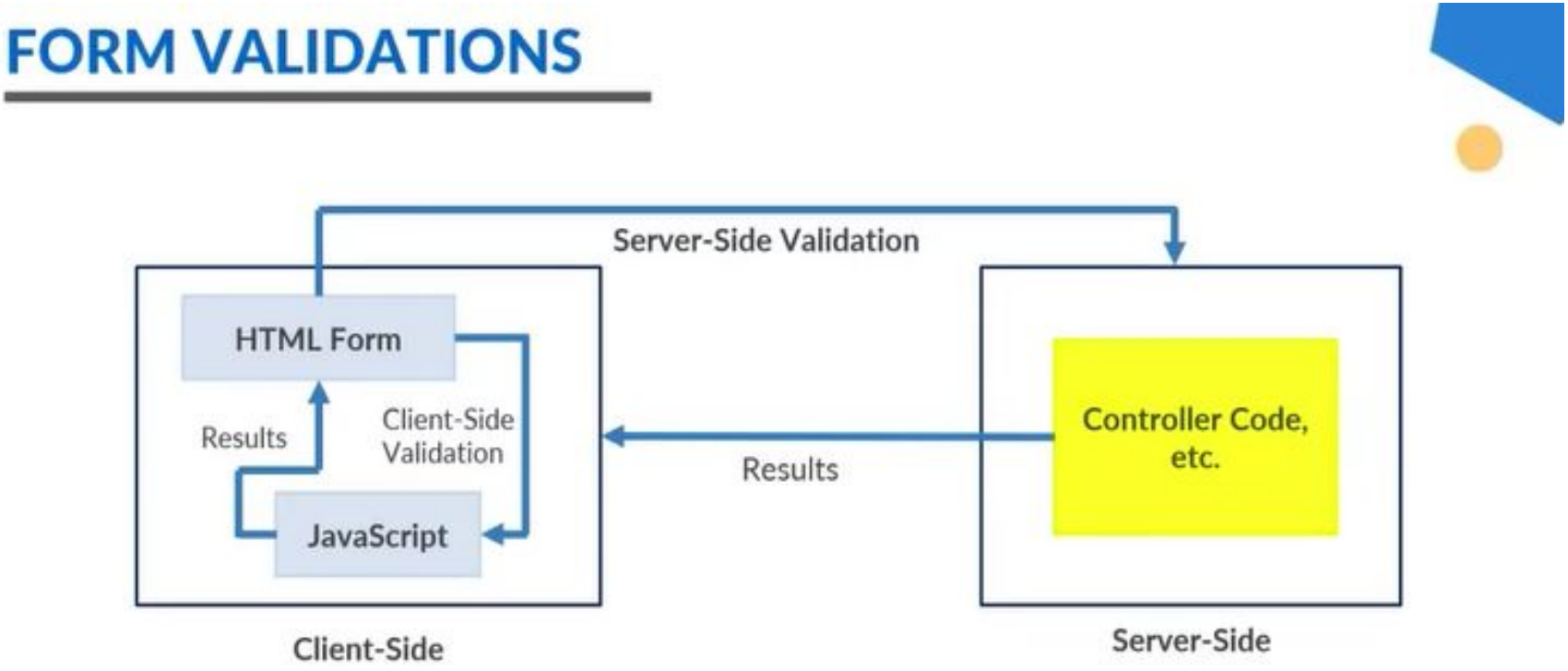


Form Validation

FORM VALIDATIONS

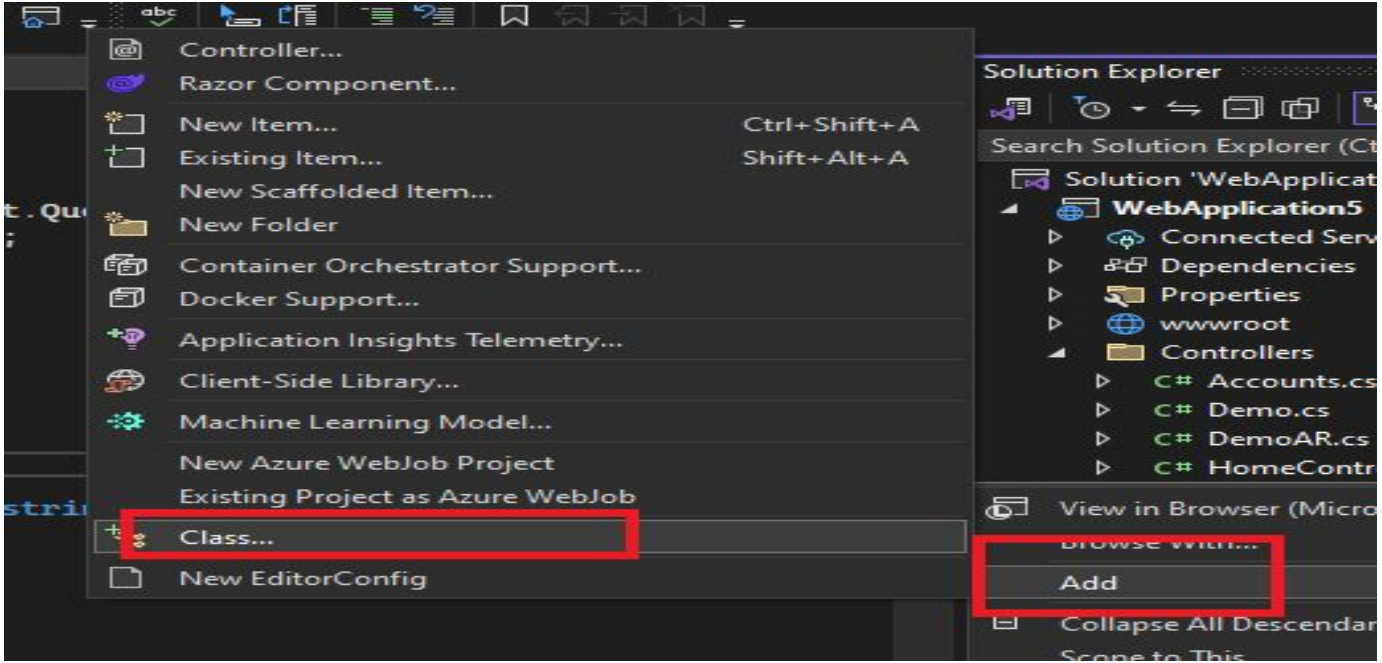
- Form validation is the process of ensuring that user input data submitted through a web form is valid, and meets certain criteria or rules.
- It helps to ensure that user input is accurate and consistent with the expected format.
- **There are different types of form validations:**
  - Server-side Form Validation
  - Client-side Form Validation
  - Custom Form Validation

Form Validation



Form Validation

Create model class



Model Class- Account.cs

```
stem.ComponentModel.DataAnnotations;

e WebApplication9.Models


ences
ic class Account

[Required]
3 references
public string ?username { get; set; }
[Required]
0 references
public string? password { get; set; }
[Required]
3 references
public string ?email { get; set; }
[Required]
0 references
public string? website { get; set; }
```

Controller

```
using Microsoft.AspNetCore.Mvc;
using WebApplication9.Models;

namespace WebApplication9.Controllers
{
    0 references
    public class AccountController1 : Controller
    {
        0 references
        public IActionResult GetAccount()
        {
            return View();
        }
        [HttpPost]
        0 references
        public IActionResult PostAccount(Account account)
        {
            if(ModelState.IsValid)
            {
                return View("success");
            }
            return RedirectToAction("GetAccount");
        }
    }
```

View of GetAccount Action Method

Tabs

WebApplication9
- Account.cs
- AccountController1.cs
- **GetAccount.cshtml**
- Program.cs
- success.cshtml
- WebApplication9: Overview

WebApplication9

```
1   @model WebApplication9.Models.Account
2
3   @{
4       ViewData["Title"] = "GetAccount";
5   }
6
7   <h1>GetAccount</h1>
8
9   <h4>Account</h4>
10  <hr />
11  <div class="row">
12      <div class="col-md-4">
13          <form asp-action="PostAccount" method="post">
14              <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15              <div class="form-group">
16                  <label asp-for="username" class="control-label"></label>
17                  <input asp-for="username" class="form-control" />
18                  <span asp-validation-for="username" class="text-danger"></span>
19              </div>
20              <div class="form-group">
21                  <label asp-for="password" class="control-label"></label>
22                  <input asp-for="password" class="form-control" />
23                  <span asp-validation-for="password" class="text-danger"></span>
```

Success view

Process: [19380] WebApplication9.exe Lifecycle Events Thread:

Tabs

WebApplication9
- Account.cs
- AccountController1.cs
- GetAccount.cshtml
- Program.cs
- **success.cshtml**
- WebApplication9: Overview

WebApplication9

```
1
2   @{
3       ViewData["Title"] = "success";
4   }
5
6   <h1>success</h1>
7   <h2>Home Page</h2>
8
9
```

Expected C

GetAccount

Account

---

username

password

email

website

Create

Back to List

GetAccount - WebApplication9 | GetAccount - WebApplication9

https://localhost:7290

Import favorites | Gmail | YouTube | Maps

WebApplication9 Home Privacy

GetAccount

Account

---

username

The username field is required.

password

The password field is required.

email

The email field is required.

website

The website field is required.

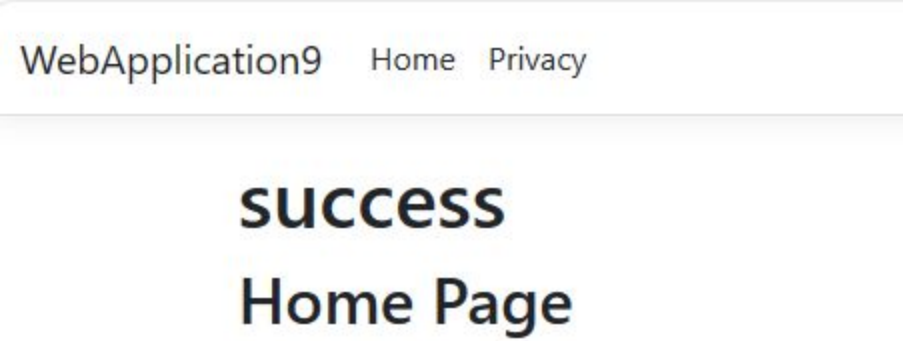
Create

Back to List

Type here to search

15°C Sunny ENG 07:46 20-03-2024

Successfully login



Changes in Program .cs= Routing

```
ticFiles();

ting();

horization();

trollerRoute(
"default",
n: "{controller=AccountController1}/{action=GetAccount}/{id?}");
```