Unit 4

Creating ASP.NET Core MVC Applications

Creating a Web App & Run

- From the Visual Studio, select Create a new project.
- Select ASP.NET Core Web Application and then select Next.
- Name the project as you like or WebApplicationCoreS1
- Choose the location path to save your project.
- Click Create
- Select Web Application(Model-View-Controller), and then select Create.
- Now, To run the App,
  - Select **Ctrl-F5** to run the app in non-debug mode, Or
  - Select IIS Express Button. 

Setting up the Environment

ASP.NET Core wwroot 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 ASP.NET Core, 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, we find static files such as JavaScript, CSS, Images, library scripts etc. in the wwwroot folder
- You can access static files with base URL and file name. For example, for css folder, we can access via http://localhost:<port>/css/app.css

Setting up the Environment

ASP.NET Core – Program.cs Class

- ASP.NET Core web application project starts executing from the entry point - public static void Main() in Program class.

ASP.NET Core – Startup.cs Class

- It is like Global.asax in the traditional .NET application. As the name suggests, it is executed first when the application starts.
- The startup class can be configured at the time of configuring the host in the Main() method of Program class.

Add a controller

- In Solution Explorer, right-click Controllers > Add > Controller
- In the Add Scaffold dialog box, select Controller Class – Empty
- In the Add Empty MVC Controller dialog, enter HelloWorldController and select Add.
- Replace the contents of Controllers/HelloWorldController.cs

```
public string Index() {
        return "This is my default action...";
}
public string Welcome() {
        return "This is the Welcome action method...";
}
```

Add a controller

- MVC invokes controller classes (and the action methods within them) depending on the incoming URL.
- The default URL routing logic used by MVC uses a format like this:

  /[Controller]/[ActionName]/[Parameters]

- The routing format is set in the Configure method in Startup.cs file.

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

- Run your app & Check with these url in your browser
  - https://localhost:{PORT}/HelloWorld
  - https://localhost:{PORT}/HelloWorld/Index
  - https://localhost:{PORT}/HelloWorld/Welcome
- Make changes for Welcome Method like this:

```
// GET: /HelloWorld/Welcome/
// Requires using System.Text.Encodings.Web;
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Hello {name}, NumTimes is: {numTimes}");
}
```

- Check on your browser with these:
  - https://localhost:{PORT}/HelloWorld/Welcome?name=AAA&numtimes=4

- Make changes again for Welcome Method with following code

```
public string Welcome(string name, int ID = 1) {
    return HtmlEncoder.Default.Encode($"Hello {name},  ID is: {numTimes}");
}
```

- Check on your browser with these:
  - http://localhost:{PORT}/HelloWorld/Welcome/3?name=AAA
- Here, the third URL segment matched the route parameter id. The Welcome method contains a parameter id that matched the URL template in the MapControllerRoute method in Startup.cs file. The trailing ? (in id?) indicates the id parameter is optional.

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

Add a view

- In your Project, Right click on the *Views* folder, and then **Add > New Folder** and name the folder *HelloWorld*.
- Right click on the *Views/HelloWorld* folder, and then **Add > New Item**.
- In the **Add New Item** dialog
  - In the search box in the upper-right, enter *view*
  - Select **Razor View**
  - Keep the **Name** box value, *Index.cshtml*.
  - Select **Add**
- Replace the contents of the Views/HelloWorld/Index.cshtml Razor view file with the following

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

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

Your Controller and View

```
public class HelloWorldController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }
```

Index.cshtml

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

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

- Navigate to https://localhost:{PORT}/HelloWorld

Change views and layout pages

- In page, Select the menu links (WebApplicationCoreS1, Home, Privacy)
- Each page shows the same menu layout.
- The menu layout is implemented in the Views/Shared/_Layout.cshtml file.
- Open the Views/Shared/_Layout.cshtml file.
- Layout templates allow you to specify the HTML container layout of your site in one place and then apply it across multiple pages in your site.
- Find the @RenderBody() line. RenderBody is a placeholder where all the view-specific pages you create show up, wrapped in the layout page.
- For example, if you select the Privacy link, the Views/Home/Privacy.cshtml view is rendered inside the RenderBody method.

Change the title, footer, and menu link in the layout file

- Replace the content of the Views/Shared/_Layout.cshtml file with the following markup. The changes are highlighted:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplicationCoreS1</title>


<div class="container">
    <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">WebApplicationCoreS1</

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2020 - WebApplicationCoreS1 - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy<
```

Passing Data from the Controller to the View

- Controllers are responsible for providing the data required in order for a view template to render a response.
- In HelloWorldController.cs, change the Welcome method to add a Message and NumTimes value to the ViewData dictionary.
- The ViewData dictionary is a dynamic object, which means any type can be used; the ViewData object has no defined properties until you put something inside it. The MVC model binding system automatically maps the named parameters (name and numTimes) from the query string in the address bar to parameters in your method.
- Ex looks like :

Passing Data from the Controller to the View

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

    public IActionResult Welcome(string name, int numTimes = 1)
    {
        ViewData["Message"] = "Hello " + name;
        ViewData["NumTimes"] = numTimes;

        return View();
    }
}
```

Passing Data from the Controller to the View

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

<h2>Welcome</h2>

<ul>
    @for (int i = 0; i < (int)ViewData["NumTimes"]; i++)
    {
        <li>@ViewData["Message"]</li>
    }
</ul>
```