Agenda: Deep Dive into Controllers

- Controllers Overview
- Action Methods and IActionResult object
- Passing data from Controller to View
- Action Selectors
- Dependency Injection to Controller

Controller Overview

The ASP.NET MVC framework maps URLs to classes that are referred to as controllers.

Controllers

- Process incoming requests
- Handle user input and interactions and
- Execute appropriate application logic.

By convention, controller classes:

- Reside in the project's root-level ***Controllers*** folder
- Inherit from Microsoft.AspNetCore.Mvc.**Controller**

A controller is an instantiable class in which at least one of the following conditions is true:

- The class name is **suffixed** with "**Controller**"
- The class inherits from a class whose name is **suffixed** with "**Controller**"
- The class is decorated with the **[Controller]** attribute

The Controller class is responsible for the following processing stages:

1. Locating the appropriate action method to call and validate that it can be called.
2. Getting the values to be used as the action method's parameters.
3. Handling all errors that might occur during the execution of the action method.
4. Providing the View for rendering ASP.NET pages to browser.

ActionMethods and IActionResult object

ASP.NET MVC application is organized around controllers and action methods. The controller defines action methods. Controllers can include as many action methods as needed.

- Action methods typically have a one-to-one mapping with user interactions. Examples of user interactions include entering a URL into the browser, clicking a link, and submitting a form. Each of these user interactions causes a request to be sent to the server. In each case, the URL of the request includes information that the MVC framework uses to invoke an action method.

- Most action methods return an instance of a class that derives from IActionResult. IActionResult is simply the interface and an ActionResult is a generic implementation of that same interface. The ActionResult class is the base for all action results. However, there are different action result types, depending on the task that the action method is performing. For example, the most common action is to call the View method. The View method returns an instance of the ViewResult class, which is derived from ActionResult.
- We can create action methods that return an object of any type, such as a string, an integer or a Boolean value. These return types are wrapped in an appropriate IActionResult type before they are rendered to the response stream.

In Controller

```
public string SayHello(string name)
{
        return "Hello " + name;
}
```

In View

```
@Html.ActionLink("Say Hello", "SayHello", new { name = "Sandeep" })
```

The following table shows the built-in action result types and the action helper methods that return them:

Classes Implementing IActionResult	Action Method	Description
BadRequestObjectResult BadRequestResult	BadRequest	Produces a Bad Request (400) response.
ChallengeResult	Challenge	Invokes AuthenticationManager.ChallengeAsync.
ContentResult	Content	Returns a user-defined content type.
CreatedAtActionResult CreatedAtRouteResult CreatedResult	CreatedAtAction CreatedAtRoute Created	Returns a Created (201) response with a Location header.
EmptyResult	---	An ActionResult that when executed will do nothing.

FileResult FileContentResult FileStreamResult VirtualFileResult	File	Write a file as the response. Write a binary file to the response. Write a file from a stream to the response. Writes the file specified using a virtual path to the response.
ForbidResult	Forbid	Invokes AuthenticationManager.ForbidAsync.
LocalRedirectResult	LocalRedirect	Returns a redirect to the supplied local URL.
NoContentResult	NoContent	Produces an HTTP response with the given response status code.
NotFoundObjectResult NotFoundResult	NotFound	Produces a Not Found (404) response.
ObjectResult	---	Produces formatting response data.
OkObjectResult OkResult	Ok	Produces an object/empty status 200 OK response.
PhysicalFileResult	PhysicalFile	On execution will write a file from disk to the response.
RedirectResult	Redirect RedirectPermanent	Redirects to another action method by using its URL.
RedirectToActionResult RedirectToRouteResult	RedirectToAction RedirectToRoute	Redirects to another action method.
SignInResult	SignIn	On execution invokes AuthenticationManager.SignInAsync.
SignOutResult	SignOut	On execution invokes AuthenticationManager.SignOutAsync.
StatusCodeResult	StatusCode	Returns a specified HTTP status code.
UnauthorizedResult	Unauthorized	Returns an Unauthorized response.
UnsupportedMediaTypeResult	---	A StatusCodeResult that when executed will produce a UnsupportedMediaType (415) response.
JsonResult	Json	Returns a serialized JSON object.
PartialViewResult	PartialView	Renders a partial view, which defines a section of a view that can be rendered inside another view.
ViewComponentResult	ViewComponent	Renders a view component to the response.
ViewResult	View	Renders a view as a Web page.

Example of ViewResult

By default, the Controller actions will return the IActionResult object. We can return various types of results as IActionResult, which will decide how the output needs to render on the browser.

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

Example of ContentResult

```
public IActionResult Index()
{
    return Content("Hello from Index action in Sample Controller");
}
public IActionResult RenderXML()
{
    return Content("<Demo>This is Test</Demo>", "text/xml", System.Text.Encoding.Unicode);
}
```

Example of Redirect

```
public IActionResult SayHello(string name)
{
  return Redirect("~/Home/RenderXml");
}
```

Example of RedirectToAction: Depending on the input values, we can redirect to another Action.

```
public IActionResult Index()
{
     // Following Redirect's to Verify action inside the Sample Controller
     return RedirectToAction("Verify", "Sample");
}
```

Example of RedirectToRoute

When we need to redirect to a route defined in ***Startup.cs***, we will use the **RedirectToRoute** object.

In **Startup.cs**:

```
        routes.MapRoute("RenderXml", // Route name
                                  "Home/RenderXml"); //URL with parameters
```

In **Home Controller**:

```
public IActionResult Index()
{
        return RedirectToRoute("RenderXml");
}
```

Example of File

PhysicalFile is used to return the content of a file to the browser.

To get the application path, IHostingEnvironment is added to the application's services by the framework, we can simply inject the service into the constructor of controller and the built-in dependency injection system will resolve it for us. IHostingEnvironment dependency is automatically injected by framework.

```
private IHostingEnvironment _hostEnv;
public HomeController(IHostingEnvironment hostEnv)
{
   _hostEnv = hostEnv;
}
public IActionResult Index()
{
    string file = System.IO.Path.Combine(_hostEnv.ContentRootPath, "Demo.xml");
    return PhysicalFile(file, "text/xml");
}
```

Note: PhysicalFileResult loads the file and renders the content to the browser without actually redirecting to the URL of mentioned file.

Example of JSON

We can render the text to the result page or can send it as a file to the client using JSON notation.

```
public IActionResult Index()
{
    Person p = new Person();
    p.FirstName = "Sandeep";
    p.LastName = "Soni";
    return Json(p);
}
```

Non-Action Methods

By default, the MVC framework treats all public methods of a controller class as action methods. If your controller class contains a public method and you do not want it to be an action method, you must mark that method with the NonActionAttribute attribute.

Example:

```
[NonAction]
public void DoSomething()
{
   // Method logic.
}
```

Action Selectors

When the MVC Framework is selecting one of the controller public methods to invoke as an action, it will use any action selector attribute that might be present to define the correct action to invoke.

1. ActionName
2. AcceptVerbs
   - HttpPost
   - HttpGet

ActionName: When we apply this to a controller action it will specify the action name for that method.

For the Index method we have bellow we no longer reach this method as action name "Index". We have to reach this method as "**Start**". (http://localhost:123/Home/Start)

```
[ActionName("Start")]
public IActionResult Index()
{
   ViewBag.Message = "Welcome to Deccansoft!";
   return View();
}
```

Note: A view by name "Start" must be added. Index view will not work.

*Can be used if the URL has "_" or "-" and method doesn't have the same.

*Also useful when method names and parameters are same for both HTTPGet and HTTPPost.

Also

```
[ActionName("GetXmlContent")]
public ActionResult GetXml()
{
    return new ContentResult()
    {
        Content="<item>Some Text</item>",
        StatusCode = 200,
        ContentType = "application/xml"
    };
}
```

AcceptVerbs: Http verbs allow us to reach a particular action we can say action method is reachable with:

1. [HttpGet] or [AcceptVerbs(HttpVerbs.Get)]
2. [HttpPost] or [AcceptVerbs(HttpVerbs.Post)]
3. **[RequireHttps]** – Forces the Http request to be resent over Https.

HttpGet: Only get request will be served by action method.

```
[HttpGet]
public IActionResult LogOn()
{
    return View();
}
```

HttpPost: Only post request will be served by the action method.

In View:

```
@{ Html.BeginForm(); }
    @ViewBag.Greetings<br />
    Enter your name: <input name="name" /><br />
    <input type="submit" value="Submit" />
@{ Html.EndForm(); }
```

In Controller:

```
[HttpPost]
public IActionResult LogOn(string name)
{
    ViewBag.Greetings = "Hello, " + name + "!";
    return View();
}
```