UNIT II: Controllers

2.1 Introduction to Controllers.
2.2 Returning from action methods
2.3 Parameters in Action methods.
2.4 Action Link.
2.5 URL Routing.
2.6 Parameters with Constraints.
2.7 Literals in URL.

2.1 Introduction to Controllers

Controller is a class that handles user requests. It retrieves data from the Model and renders view as response.

The ASP.NET MVC framework maps requested URLs to the classes that are referred to as controllers. Controller processes incoming requests, handle user input and interactions and executes appropriate business logic.

The ControllerBase class is a base class for all controller classes. It provides general MVC handling.

It locates for the appropriate action method to call and validate.

It gets the values to use as the action method's arguments.

It handles all errors that might occur during execution of the action.

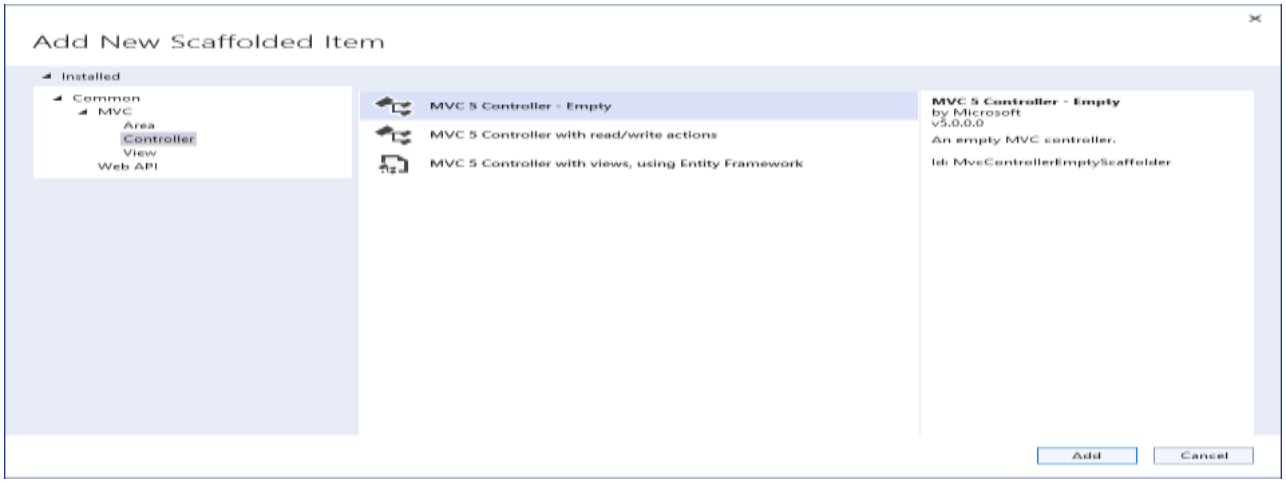
It uses the WebFormViewEngine class for rendering ASP.NET page.

Create a Controller

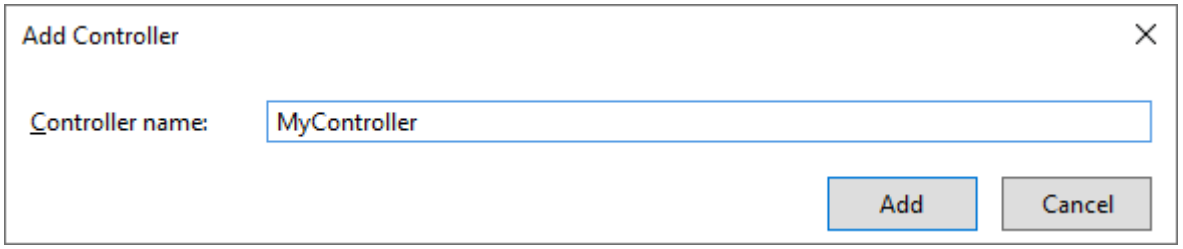
We can create controller for the application by adding a new item into the controller folder.

Just right click on the controller folder and click add -> controller.

This opens Add Scaffold dialog



Select "MVC 5 Controller - Empty" and click Add. It will open the Add Controller dialog.



In the Add Controller dialog, enter the name of the controller. Remember, the controller name must end with Controller.

Write MyController and click Add.

This will create the MyController class with the Index() method in MyController.cs file under the Controllers folder.

After adding this controller, as per the conventions project will create a folder with the same name as the controller name in the view folder to store the view files belongs to the controller.

You need to create index.cshtml file inside view -> my and add following code.

```
<div class="jumbotron">

   <h2>Welcome to the WebApp Demo.</h2>

</div>
```

Calling action methods through the browser

Open any browser and enter the URL like “DomainName/ControllerName” as follows.

Ex. localhost:port_no/Controller name/Index

To Change the default controller you need to open the **Routeconfig.cs** file from **App_start** folder.

Change the controller name to the "My" controller instead of the “Home” controller inside the Routeconfig file.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```

class System.String
Represents text as a sequence of UTF-16 code units.To browse the .NE

2.2 Returning from action methods

In MVC Controller file you have many action methods.

Each action method can return different return types of results like contentresult,javascript,json or view.

Actionresult is an abstract class and it is the base class for all types of action results.

1. ViewResult

If you want to return a view in action method , you should use View as return type of that method

Ex. public class EmployeeController : Controller

```
{           public ViewResult Index()

{  return View();}}
```

2. PartialViewResult

If you want to return partialview in action method,you should use partialviewresult as return type of that method.

Partialview is used to call inside our Normal view page.

We have to create partial view inside shared folder.

PartialViewexample.cshtml

```
<div style="border:1px solid blue;height:400px;width:300px;">

This is my partial view

<div>
```

MainPage.cshtml

@Html.Partial("partialViewExample");

<h2>This is the output of my MAIN PAGE. <h2>

3. ContentResult

If you want to return your content to the view then you should use Content as the return type of the action method. you can return htmlresult,javascript result etc.

Ex.

public ContentResult Index()

{ return Content("<h1><font color='blue'>I AM HAPPY WITH<font>, WHAT I HAVE<h1><script>alert('HI USMTECHWORLD , HOW R U?');<script>");

}

4. EmptyResult

This emptyresult returns nothing in the view page

public EmptyResult Index()

{ return new EmptyResult(); or return null;}

5. FileResult

If you want to return file to the view then you should use File as the return type of the action method

The FileREsult class has File method inside and has 2 parameters i.e Filename and ContentType.

```
public FileResult Index()

{return File("~/Aboutme.txt", "text/plain");}
```

Suppose if you don't want to show your file in the browser, but asked to download it then Just give filedownloadname as 3rd parameter in File return type.

2.3 Parameters in Action methods

Action parameters are the variables that are used to retrieve user requested values from the URL.

The parameters are retrieved from the request's data collection. It includes name/value pairs for form data, query string value etc.

The controller class locates for the parameters values based on the RouteData instance. If the value is present, it is passed to the parameter. Otherwise, exception is thrown.

The Controller class provides two properties Request and Response that can be used to get handle user request and response.

Syntax

```
Public retrun_type method_name(type parameter_name)

{ return “”;}
```

Ex.

```
public string ShowMsg(string MsgTitle)

{       return "You selected " + MsgTitle + " Ok"; }
```

In URL, we have to pass the parameter value.

Ex.

https://localhost:44377/My/ShowMsg?MsgTitle=user.

Output:

You selected user Thank You!

If you want to add multiple parameter in method just use (, comma) for separate other parameter.

Ex.

```
public string ShowMsg(string MsgTitle1, string MsgTitle2, string MsgTitle3, string MsgTitle4)

{          return "You selected " + MsgTitle1+" "+ MsgTitle2 +" "+MsgTitle3 +" "+MsgTitle4 + "\n Thank You!";     }
```

In URL, we have to pass the multiple parameter value by using && (and) Operator.

Ex.

https://localhost:44377/My/ShowMsg?MsgTitle1=Gopal**&&**MsgTitle2=Pundlik**&&**MsgTitle3=Shinde**&&**MsgTitle4=Latur.

Output:

You selected Gopal Pundlik Shinde Latur Thank You!

2.4 Action Link

Html.ActionLink is used for creating hyperlink. This action method renders hyperlink in html pages but it redirect to action method not directly to view pages.

The Html.ActionLink creates an anchor element based on parameters supplied.

We will also create multiple types of Action links

1. HtmlActionLink without controller name.

Ex. @Html.ActionLink("Click Here", "About")

2. HtmlActionLink with controller name.

Ex. @Html.ActionLink("Home Controller Page", "Index", "Home")

3. HtmlActionLink with parameters.

@Html.ActionLink("Msg Page", "ShowMsg", "My", new { MsgTitle1 = "Gopal" }, null)

Note: Do not forgot to add null as 5th parameter.

4. HtmlActionLink with Area.

@Html.ActionLink("Link Text", "ActionName", "ControllerName", new { Area = "AreaName" }, null)

Note: Do not forgot to add null as 5th parameter.

5. HtmlActionLink with CSS class

@Html.ActionLink("Click Me", "Index", "Home", null, new { @class = "myCustomLink" })

This is useful when we want to apply any CSS class to our anchor text.

2.5 URL Routing

In MVC, routing is a process of mapping the browser request to the controller action and return response back.

The need of URL Routing

Each MVC application has default routing for the default HomeController. We can set custom routing for newly created controller.

The **RouteConfig.cs** file is used to set routing for the application.

```
public static void RegisterRoutes(RouteCollection routes)

    {

       routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

       routes.MapRoute(
```

Default Parameter Values

```
          name: "Default",

          url: "{controller}/{action}/{id}",
```

Parameters in URL

```
          defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }

       );  }
```

According to this setup file, Index action of Home controller will be treated as default.

If we look at the address bar, it contains only localhost: 44377

There is no controller and action is specified because MVC router maps the controller from the RouteConfig.cs.

If we explicitly enter controller and action names in the address bar, it will redirect to the same action. The localhost: 44377/Home/Index will produce the same output to the browser.

2.6 Parameters with Constraints

Routing constraints let you restrict how the parameters in the route template are matched. It helps to filter out the input parameter and action method it can accept.

if the URL Parameter is restricted to have int value, the route engine will match the controller action having integer value in the parameter or restrict it.

How Route Constraints are applied:

Using constraint parameter in the MapControllerRoute at the application startup where the endpoints are defined; i.e. Inline Constraint

Route attribute at the controller or action method

endpoints.MapControllerRoute(

name: "default",

pattern: "controller=Home}/{action=Index}/{id:int?}");

When the route engine matches the incoming url, it invokes the routing constraint to the values in the url which match the pattern. In this case which is **seperated by :** restricts the value should be **int** or **null**.

Alpha - Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)

Bool - Matches a Boolean value.

Int - Matches a 32-bit integer value.

Long - Matches a 64-bit integer value.

Float - Matches a 32-bit floating-point value.

Double - Matches a 64-bit floating-point value.

Decimal - Matches a decimal value.

Length - Matches a string with the specified length or within a specified range of lengths.

Max - Matches an integer with a maximum value.

Maxlength - Matches a string with a maximum length.

Min - Matches an integer with a minimum value.

Minlength - Matches a string with a minimum length.

Range - Matches an integer within a range of values.

Regex - Matches a regular expression.

2.7 Literals in URL

URL rewriting is the act of modifying request URLs based on one or more predefined rules.

URL rewriting creates an abstraction between resource locations and their addresses so that the locations and addresses aren't tightly linked.

URL rewriting is valuable in several scenarios to:

Move or replace server resources temporarily or permanently and maintain stable locators for those resources.

Split request processing across different apps or across areas of one app.

Remove, add, or reorganize URL segments on incoming requests.

Optimize public URLs for Search Engine Optimization (SEO).

Permit the use of friendly public URLs to help visitors predict the content returned by requesting a resource.

Redirect insecure requests to secure endpoints.

Prevent hotlinking, where an external site uses a hosted static asset on another site by linking the asset into its own content.

URL rewriting can reduce the performance of an app. Limit the number and complexity of rules.

Example: pattern: "{controller=Home}/{action=Index}/{id?}");

Thank You!