ASP.NET Core

Day 3

ASP.NET Core MVC –

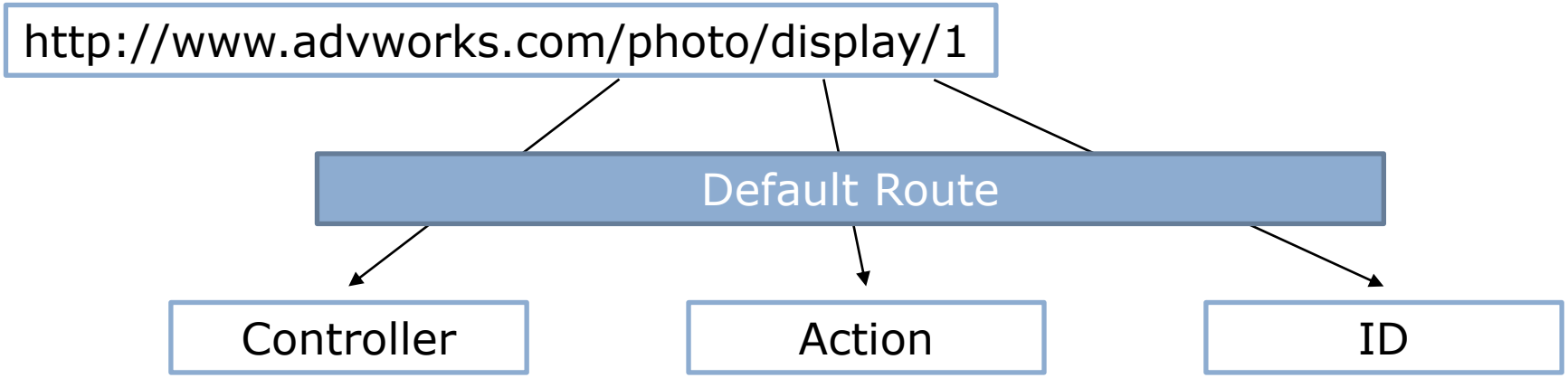
Razor View Engine, Routing

Session Overview

- Introduction to Routing
- Defining Routes
- Attribute Routing
- Need of Attribute Routing
- Understanding Razor View Engine
- Razor Syntax
- Razor Statements, Loops etc.

The ASP.NET Routing Engine

- The default route:



- Custom routes:
  - To make URLs easier for site visitors to understand
  - To improve search engine rankings
- Controller factories and routes

Adding and Configuring Routes

- Understand the properties of a route:
  - Includes Name, URL, Constraints and Defaults
- Analyze the default route code:
  - Specifies **Name**, **URL** ,and **Defaults** properties
- Create Custom Routes:
  - Involves calling the **routes.MapHttpRoute()** method
- Understand the precedence of routes:
  - Add routes to the **RouteTable.Routes** collection in the appropriate order

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

```
routes.MapRoute(
   name: "PhotoRoute",
   url: "photo/{id}",
   defaults: new {
      controller = "Photo",
      action = "Details" },
   constraints: new {
      id = "[0-9]+" }
);
```

Using Routes to Pass Parameters

- You can access the values of these variables by:
  - Using the **RouteData.Values** collection
  - Using the model binding to pass appropriate parameters to actions

```
public void ActionMethod Display (int PhotoID)
{
   return View(PhotoID);
}
```

- You can use optional parameters to match a route, regardless of whether parameter values are supplied

```
routes.MapRoute(
   name: "ProductRoute",
   url: "product/{id}/{color}",
   defaults: new { color = UrlParameter.Optional }
)
```

Understanding Routing

```
HomeController
public Index()
public Error()
public About()
```

```
BlogController
public Index()
public Create()
public Post()
```

```
HomeController
public Index()
public Error()
public About()
```

Understanding Routing

```
HomeController
  public Index()
  public Error()
  public About()

BlogController
  public Index()
  public Create()
  public Post(id)

AccountController
  public Login()
  public Logout()
```

Routing Table

Controller	Action	Route
Home	Index	/home/index
Home	Error	/home/error
Home	About	/home/about
Blog	Index	/blog/index
Blog	Create	/blog/create
Blog	Post	/blog/post/{id}
Account	Log in	/account/login
Account	Log out	/account/logout

Understanding Routing

```
"{controller=Home}/{action=Index}/{id?}"
```

/blog/post/123

Controller: Blog

Action: Post

ID: 123

Routing Table

Controller	Action	Route
Home	Index	/home/index
Home	Error	/home/error
Home	About	/home/about
Blog	Index	/blog/index
Blog	Create	/blog/create
Blog	**Post**	**/blog/post/123**
Account	Log in	/account/login
Account	Log out	/account/logout

Concept of Razor Engines

- Razor identifies server-side code by looking for the **@** symbol.
- In Razor syntax, the **@** symbol has various uses. You can:
  - Use **@** to identify server-side C# code.
  - Use **@@** to render an @ symbol in an HTML page.
  - Use **@:** to explicitly declare a line of text as content and not code.
  - Use **<text>**to explicitly declare several lines of text as content and not code.
- To render text without HTML encoding, you can use the **Html.Raw()** helper.

Features of Razor Syntax

- A sample code block displaying the features of Razor.

```
@* Some more Razor examples *@

<span>
Price including Sale Tax: @Model.Price * 1.2
</span>
<span>
Price including Sale Tax: @(Model.Price * 1.2)
</span>

@if (Model.Count > 5)
{
 <ol>
   @foreach(var item in Model)
   {
      <li>@item.Name</li>
   }
 </ol>
}
```

Binding Views to Model Classe and Displaying Properties

- You can use strongly-typed views and include a declaration of the model class. Visual Studio helps you with additional IntelliSense feedback and error-checking as you write the code.
- Binding to Enumerable Lists:

```
@model IEnumerable<MyWebSite.Models.Product>

<h1>Product Catalog</h1>

@foreach (var Product in Model)

{

   <div>Name: @Product.Name</div>

}
```

- You can use dynamic views to create a view that can display more than one model class.

Rendering Accessible HTML

- You can ensure that your content is accessible to the broadest range of users by adhering to the following guidelines:
  - Provide **alt** attributes for visual and auditory content
  - Do not rely on color to highlight content
  - Separate content from structure and presentation code:
    - Only use tables to present tabular content
    - Avoid nested tables
    - Use ** ** elements and positional style sheets to lay out elements on the page
    - Avoid using images that include important text
    - Put all important text in HTML elements or **alt** attributes