ASP.NET CORE MVC MODULE 04
DEVELOPING CONTROLLERS

Summer 2021 – Web Development using ASP .Net Core MVC

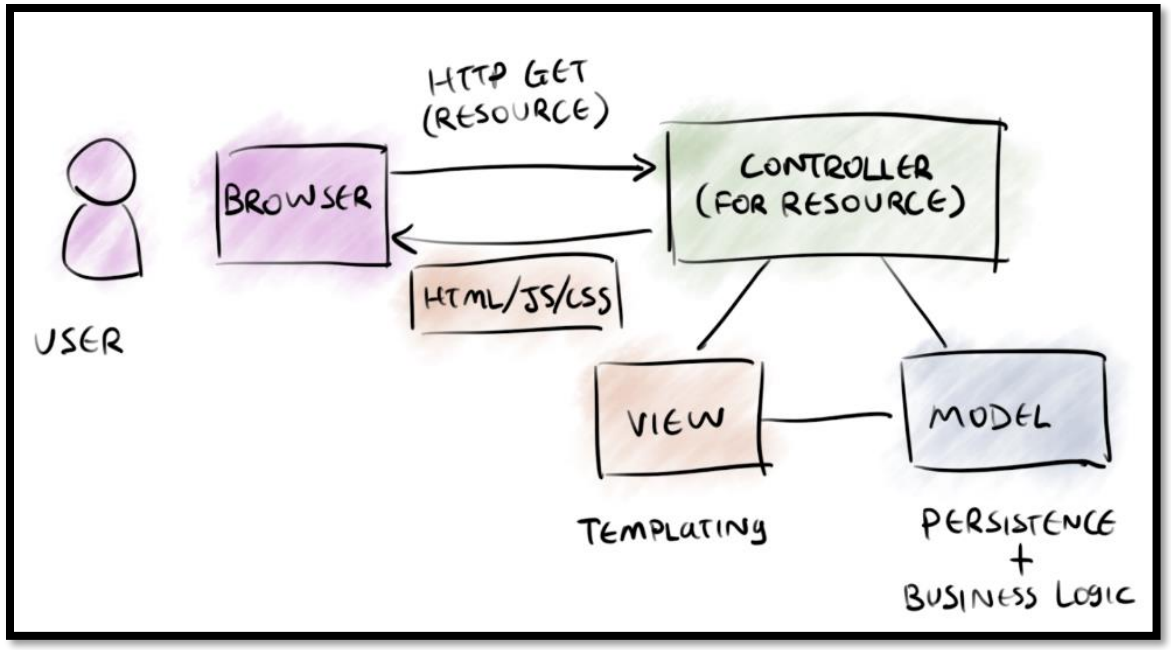
***This lesson is very important. Please ensure that you have a clear understanding of controllers & actions, and how they handle user requests.**

MAIN SOURCES FOR THESE SLIDED

- Unless otherwise specified, the main sources for these slides are:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications ←for homework
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0 ←for "textbook"
    - Especially pages 1621-1691

SOME ESSENTIAL MVC CONCEPTS

- Here is a great picture to help you visualize the MVC model



Naming is important: MVC relies on **convention** over **configuration**

- a **request** is sent from a **web browser**.
  - for example: http://www.mysite.com/student/show/1

- a **controller** object is instantiated to respond to this **request**.
  - for the above, the **StudentController** will be instantiated
  - the **URL routing** determines which **controller** & **action** will handler the request

---

- an **action** method is then called by the **controller**.
  - for the above, a **Show** action will be selected
  - a **model binder** determines the values passed to the **action** as **parameters** (e.g. **1**).
  - the **action** may create a new instance of a **model** class.
  - this **model** object may be passed to a **view** to display results.

- a **view** will produce the output that is sent back to the **browser**.
  - The output could be HTML+CSS+JS file (most often), or JSON, XML, text, files, …

- Side note: where does middleware fit into this diagram?

LET'S CREATE AN MVC WEB APPLICATION

- In this module we'll focus on **controllers**, but we'll briefly see **models** and **views** too.
- Source: https://www.microsoftpressstore.com/articles/article.aspx?p=2928202
- Let's start by creating an **empty** Web Application an MVC application …

- To add MVC framework to the web application you'll need the following:
  - Add the MVC Service ← services.AddMvc(x => x.EnableEndpointRouting = false);
  - Activate the MVC Service ← app.UseMvcWithDefaultRoute();

- **Notes:**
  - The actual code behind the **UseMvcWithDefaultRoute** method is (we'll see more details later):

```
app.UseMvc(routes =>
  {
     routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
  });
```

  - You may want to add the following before the **UseMvcWithDefaultRoute** call:

```
app.UseDeveloperExceptionPage(); //useful for debugging
```

  - For testing purposes, after the app.UseMvcWithDefaultRoute call, also add:

```
app.Run(async (context) =>
{
     await context.Response.WriteAsync("You did not match a configured route in here.");
})
```

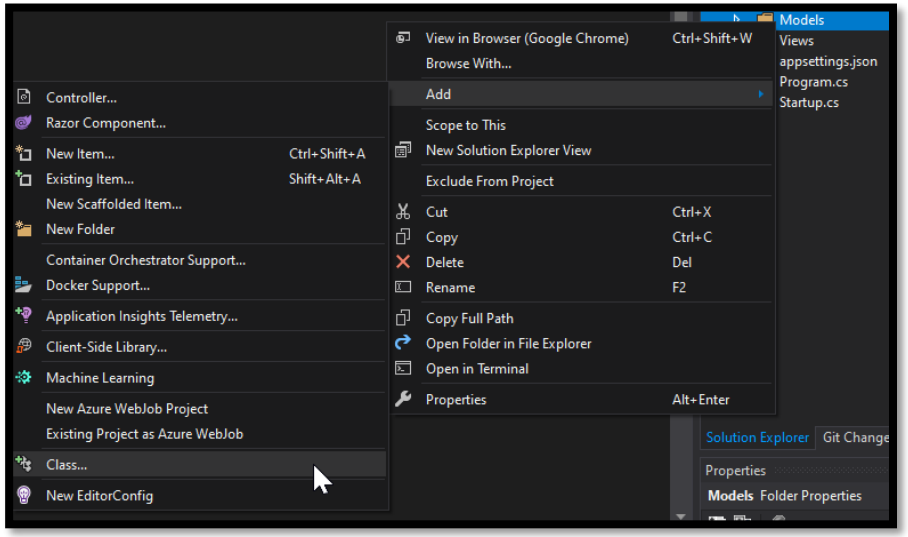
```
// This method gets called by the runtime. Use this method to add servic
// For more information on how to configure your application, visit https
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(x => x.EnableEndpointRouting = false);
}

// This method gets called by the runtime. Use this method to configure
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseMvcWithDefaultRoute();
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
              "You did not match a configured route in here");
    });
}
```

LET'S CREATE AN MVC WEB APPLICATION

- Let's create a **Student model class**.
  - A **model** class is simply a POCO class. Here is an example:

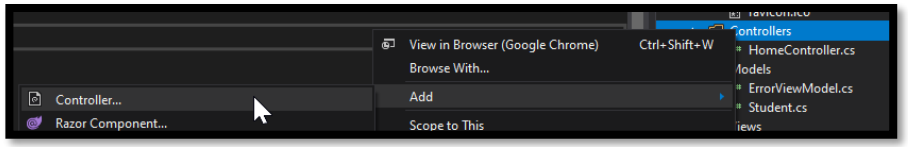
```
public class Student
{
    public string FirstName { get; set; }
    public string Major{ get; set; }
    public double GPA { get; set; }
}
```



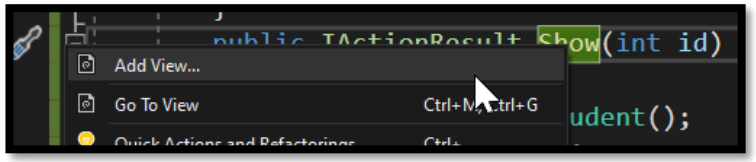
  - Then let's add a **controller class**.
    - Typically named **StudentController**, derived from **Controller**
    - **Actions**, are public methods defined inside a **controller** class.
    - This example has 2 **actions**:

```
public class StudentController : Controller
{
    public IActionResult Index()
    {
        return Content("No student selected");
    }
    public IActionResult Show(int id)
    {
        Student st = new Student();
        if (id == 1) { st.FirstName = "Alex"; st.Major = "IT"; st.GPA = 3.0; }
        else { st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0; }

        return View(st);
    }
}
```



  - Lastly, let's add a **view** for the second **action**.
    - Note **convention over configuration**




```
@model WebApplication2.Models.Student

<!DOCTYPE html>
<html>
<body>

    <h1>Student name is @Model.FirstName</h1>
    <p> Major: @Model.Major  and GPA: @Model.GPA</p>

</body>
</html>
```

- Next, let's test our application … to see how MVC works …

QUICK OVERVIEW OF MVC

- Next, let's test our application … to see how MVC works …
  - This is the code we have so far … ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔ ➔

- Users can make requests to a web application either:
  - By **typing an URL** in a browser, or
  - By **clicking on a link** or **button** within a webpage

- Let's have a quick overview of the MVC request lifecycle and routing
  - (more details on **routing** … see below…)
  - https://localhost:44394/ ← calls the default **Home** controller, **Index** action (if it exists …)
  - https://localhost:44394/Student ←calls the **Student** controller, **Index** action
  - https://localhost:44394/Student/Show ← calls the **Student** controller, **Show** action (no id!)
  - https://localhost:44394/Student/Show/1 ← calls the **Student** controller, **Show** action (id = 1)
    - A **parameter** is passed to the Show action
  - https://localhost:44394/Student/Show/2 ← calls the **Student** controller, **Show** action (id = 2)
    - A **parameter** is passed to the Show action

  - https://localhost:44394/Student/Show?id=1 ← calls the **Student** controller, **Show** action (id = 1)
    - A **query string** is used (the **model binder** passes this value to the **Show** action because it has a **parameter** called **id**)
  - https://localhost:44394/Student/Show?nm=1 ← calls the **Student** controller, **Show** action (nm = 1)
    - A **query string** is used (the **model binder does not** find a **Show** action with a parameter named **nm**)

ROUTING

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseDeveloperExceptionPage();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
            "You did not match a configured route in here");
    });
}
```

MODEL

```
public class Student
{
    public string FirstName { get; set; }
    public string Major{ get; set; }
    public double GPA { get; set; }
}
```

CONTROLLER

It has 2 actions

```
public class StudentController : Controller
{
    public IActionResult Index()
    {
        return Content("No student selected");
    }
    public IActionResult Show(int id)
    {
        Student st = new Student();
        if (id == 1) { st.FirstName = "Alex"; st.Major = "IT"; st.GPA = 3.0; }
        else { st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0; }

        return View(st);
    }
}
```

VIEW

```
@model WebApplication2.Models.Student

<!DOCTYPE html>
<html>
<body>

    <h1>Student name is @Model.FirstName</h1>
    <p> Major: @Model.Major  and GPA: @Model.GPA</p>

</body>
</html>
```

ACTION RESULTS IN ASP.NET MVC CORE

- Source (please review it!): https://tutexchange.com/action-results-in-asp-net-mvc-core/
- Above we saw two types of results returned by actions. There are several others. For example:
  - **ViewResult**
    - used to render a **View** as response (see above)
  - **ContentResult**
    - used when we want to send content like plain text, XML, html as response
  - **JsonResult**
    - used to send a JSON object as a result
  - **PartialViewResult**
    - we may see them later …
  - **RedirectResult**
    - used when we want to redirect to specific URL page
  - **RedirectToActionResult**
    - used when we want to redirect to another **Action** (from the same or another **Controller**)
  - **RedirectToPageResult**
    - …
  - **RedirectToRouteResult**
    - …
  - **StatusCodeResult**
    - when we want to send an HTTP status code as result
  - **ViewComponentResult**
    - we may see this later …

```
public IActionResult Show(int id)
{
    Student st = new Student();
    st.FirstName = "Alex"; st.Major = "IT"; st.GPA = 3.0;
    return View(st);
}
```

ViewResult Controller.View(object model) (+ 3 overloads)
Creates a ViewResult object by specifying a model to be rendered by the view.
Returns:
The created ViewResult object for the response.

```
public IActionResult Index()
{
    return Content("No student selected");
}
```

ContentResult ControllerBase.Content(string content) (+ 3 overloads)
Creates a ContentResult object by specifying a content string.
Returns:
The created ContentResult object for the response.

```
public IActionResult Index()
{
    return Redirect("http://www.google.com");
}
```

RedirectResult ControllerBase.Redirect(string url)
Creates a RedirectResult object that redirects (Microsoft.AspNetCore.Http.StatusCodes.Status302Found) to the specified url.
Returns:
The created RedirectResult for the response.

```
public IActionResult Index()
{
    return RedirectToAction("Show","Student", 1);
}
```

RedirectToActionResult ControllerBase.RedirectToAction(string actionName, string controllerName, object routeValues) (+ 6 overloads)
Redirects (Microsoft.AspNetCore.Http.StatusCodes.Status302Found) to the specified action using the specified actionName, controllerName, and routeValues.
Returns:
The created RedirectToActionResult for the response.

```
public IActionResult Index()
{
    return StatusCode(404);
}
```

StatusCodeResult ControllerBase.StatusCode(int statusCode) (+ 1 overload)
Creates a StatusCodeResult object by specifying a statusCode.
Returns:
The created StatusCodeResult object for the response.

- Note that we typically use **IActionResult** (this is an interface) as the return type.

WHAT IS MODEL BINDERS

- See page: 5649

- **Controllers** work with **data** that comes from **HTTP requests.**
  - There are various sources such as: **form fields, route data, query strings, uploaded/posted files**.
  - These values need to be passed to various **Action** parameters (and these can be of simple/complex types)

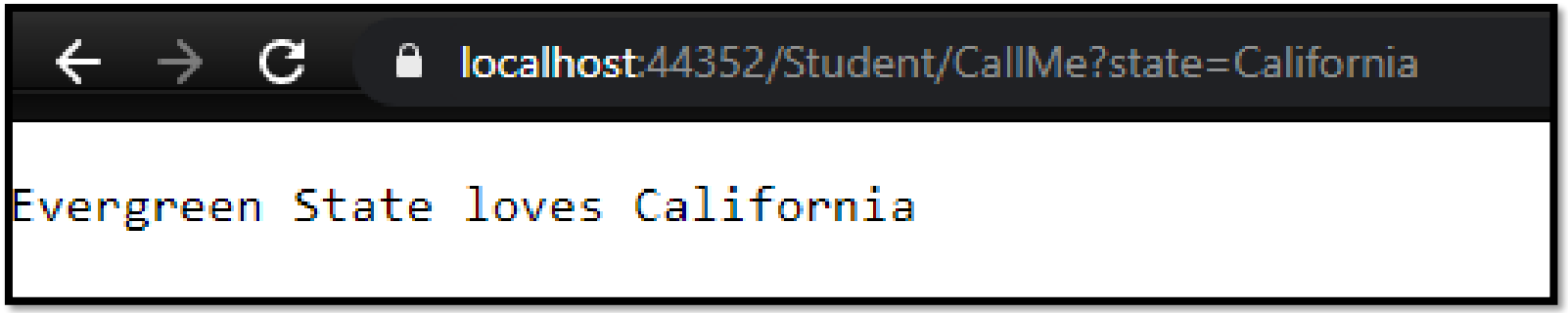
- **Model binders**:
  - **Retrieves** data from various sources mentioned above (sources searched in that order ^ …)
  - **Provides** this data to **controllers** in **method parameters** and **public properties**.
  - **Converts** string data to .NET types (we'll see this later).

- Examples (if time …)
  - https://localhost:44394/Student/Show/1 ← calls the **Student** controller, **Show** action (id = 1)
    - A **parameter** is passed to the Show action
  - https://localhost:44394/Student/Show/2 ← calls the **Student** controller, **Show** action (id = 2)
    - A **parameter** is passed to the Show action
  - https://localhost:44394/Student/Show?id=1 ← calls the **Student** controller, **Show** action (id = 1)
    - A **query string** is used (the **model binder** passes this value to the **Show** action because it has a **parameter** called **id**)
  - https://localhost:44394/Student/Show?nm=1 ← calls the **Student** controller, **Show** action (nm = 1)
    - A **query string** is used (the **model binder does not** find a **Show** action with a parameter named **nm**)

READ ABOUT HTTP ENDPOINTS – ON YOUR OWN

- See page 1632
- "Every *public* method in a **controller** is callable as an HTTP endpoint"

```
public class StudentController : Controller
{
    public string CallMe(string state)
    {
        return "Evergreen State loves " + state;
    }
```



VIEWBAG AND VIEWDATA

- In the earlier example we've seen how to pass a **model** object to a **view**.
  - To provide extra information one can use **ViewBag** and/or **ViewData** properties

```
public class StudentController : Controller
{
    public IActionResult Index()
    {
        return Content("No student selected");
    }
    public IActionResult Show(int id)
    {
        Student st = new Student();
        if (id == 1) { st.FirstName = "Alex"; st.Major = "IT"; st.GPA = 3.0; }
        else { st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0; }

        return View(st);
    }
}
```

CONTROLLER

It has 2 actions

- **ViewBag**:
  - Is a property that is a **dynamic object**, part of the **Controller** base class.
  - Set values in an **action** method, use those values inside a **view**.

```
public IActionResult Show(int id)
{
    Student st = new Student();
    st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0;
    ViewBag.EvergreenState = "Is Awesome";

    return View(st);
}
```

```
@model WebApplication2.Models.Student

<!DOCTYPE html>
<html>
<body>

    <h1>Student name is @Model.FirstName</h1>
    <p> Major: @Model.Major  and GPA: @Model.GPA</p>
    Life @ViewBag.EvergreenState
</body>
</html>
```

VIEW

```
@model WebApplication2.Models.Student

<!DOCTYPE html>
<html>
<body>

    <h1>Student name is @Model.FirstName</h1>
    <p> Major: @Model.Major  and GPA: @Model.GPA</p>

</body>
</html>
```

VIEW

- **ViewData**:
  - Is a property that is a **dictionary**, part of the **Controller** base class.
  - Set values in an **action** method, use those values inside a **view**.

```
public IActionResult Show(int id)
{
    Student st = new Student();
    st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0;
    ViewData["EvergreenState"] = "Is Awesome";

    return View(st);
}
```

```
@model WebApplication2.Models.Student

<!DOCTYPE html>
<html>
<body>

    <h1>Student name is @Model.FirstName</h1>
    <p> Major: @Model.Major  and GPA: @Model.GPA</p>
    Life @ViewData["EvergreenState"]
</body>
</html>
```

VIEW

RECOMMENDED READING AHEAD

- On your own, you may want to check out the following:
  - Page 1621: Get started with ASP.NET Core MVC
  - Page 1630: Part 2, add a controller to an ASP.NET Core MVC app
  - Page 1642: Part 3, add a view to an ASP.NET Core MVC app
  - Page 1658: Part 4, add a model to an ASP.NET Core MVC app
  - Page 1691: Part 5, work with a database in an ASP.NET Core MVC app
  - Page 1704: Part 6, controller methods and views in ASP.NET Core
  - Page 1718: Part 7, add search to an ASP.NET Core MVC app
  - Page 1729: Part 8, add a new field to an ASP.NET Core MVC app (… see migrations)
  - Page 1734: Part 9, add validation to an ASP.NET Core MVC app

IN-CLASS DEMO: HOW TO WRITE CONTROLLERS AND ACTIONS

Demonstration: How to Write Controllers and Actions

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD04_DEMO.md#demonstration-how-to-write-controllers-and-actions

ROUTING IN ASP .NET CORE

- Main sources for the next few slides:
  - See pages 1765+ … for the next few slides' source + more!
  - See also: https://dotnettutorials.net/lesson/routing-asp-net-core-mvc/

- ASP.NET Core **controllers** use the **Routing middleware** to match the URLs of incoming requests and map them to **actions**. For this we use **routing templates**.

- **Route templates**:
  - defined in **startup code** or **attributes**.
  - describe how **URL** are matched to **actions**.
  - used to generate URLs for links (seen later). These generated links are typically returned in responses.

- **Actions** are either **conventionally-routed** or **attribute-routed**
  - **Conventional routing** is centrally configured in Startup.cs
  - **Attribute routing** is locally configured using attributes

http://localhost:52190/Home/Details/2

```
public class HomeController : Controller
{
    public ViewResult Details(int id)
    {
        return View();
    }
}
```

DEFAULT ROUTING

- The actual code behind the **UseMvcWithDefaultRoute** method is:

```
app.UseMvc(routes =>
  {
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
  });
```

```
// This method gets called by the runtime. Use this method to add servic
// For more information on how to configure your application, visit http
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(x => x.EnableEndpointRouting = false);
}

// This method gets called by the runtime. Use this method to configure
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseMvcWithDefaultRoute();
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync(
                "You did not match a configured route in here");
    });
}
```

- **MapRoute** is used to create a single route
  - The first parameter is just a name (useful only to developers)
  - The second parameter is a template. In the example above, it has three **segments**:
    - {**controller**=Home} the first segment represents the controller name – here it also defines **Home** as the **default controller**
    - {**action**=Index} the second segment represents the action name – here it defines **Index** as the **default action**
    - {id?} the third segment represents a parameter named **id** – here ? specifies that **id** as **optional**

- **Default routing** work with many applications. What do each of the following URLs represent/call?
  - www.mysite.com/
  - www.mysite.com/Student
  - www.mysite.com/Home
  - www.mysite.com/Student/Show
  - www.mysite.com/Student/Show/2021
  - www.mysite.com/Show/Student/2021
  - www.mysite.com/Word1/Word2/Word3/10

WILL EACH OF THE FOLLOWING FAIL OR SUCCEED?

- www.mysite.com/Word1/Word2
- www.mysite.com/Word1
- www.mysite.com

```
app.UseMvc(routes =>
  {
    routes.MapRoute("route1", “{controller}/{article}");
  });
```

- ---

- www.mysite.com/Word1/Word2
- www.mysite.com/Word1
- www.mysite.com

```
app.UseMvc(routes =>
  {
    routes.MapRoute("route2", "{article}/{controller}");
  });
```

- ---

- www.mysite.com/Word1/Word2/Word3
- www.mysite.com/article/Word2/Word3
- www.mysite.com /article/Word2/

```
app.UseMvc(routes =>
  {
    routes.MapRoute("route3", "article/{controller}/{action}");
  });
```

- ---

- www.mysite.com/Word1/Word2
- www.mysite.com/WA/Word1
- www.mysite.com / WA/Word1/Word2/

```
app.UseMvc(routes =>
  {
    routes.MapRoute("route4", “WA/{action}"
          default: new{Controller = “Student”} );
  });
```

MULTIPLE ROUTES (CONVENTIONAL ROUTING)

- One can add multiple routes. For example:

```
app.UseMvc(routes =>
  {
    routes.MapRoute("blogs", "blog/{article}", defaults: new {controller = "Blog", action = "Show"});
    routes.MapRoute("students", "students/{action}/{id?}", defaults: new {controller = "Student", action = "Show"});
    routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
  });
```

- Notice we have three routes. The first route also uses a **named parameter** *defaults*.
  - What kind of URLs would the routes shown above match?

- When you have multiple routes, the are evaluated in the order in which they are added.
  - The first route matching a requested URL will be used.
  - Therefore, order is important! Add most general routes last

- See also: https://exceptionnotfound.net/routing-basics-in-asp-net-core-3-0/

ATTRIBUTE ROUTING

- One can define routes in the same file as the **controller** that the routes refer to.
  - Note: You can mix and match convention-based routing and attribute-based routing!

- What will the following return?
- https://localhost:44352/
- https://localhost:44352/Student
- https://localhost:44352/Student/WA
- https://localhost:44352/Student/Show
- https://localhost:44352/Student/Show/2020

```
// This method gets called by the runtime. Use this method
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(x => x.EnableEndpointRouting = false);
}

// This method gets called by the runtime. Use this method
public void Configure(IApplicationBuilder app)
{
    app.UseMvc();
}
```

```
public class StudentController : Controller
{
    [Route("")]
    [Route("Student")]
    [Route("Student/{state?}")]
    public string CallMe(string state)
    {
        return "Evergreen State loves " + state;
    }


    public IActionResult Show()
    {
        return Content("No student selected");
    }
    public IActionResult Show(int id)
    {
        Student st = new Student();
        st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0;

        return View(st);
    }
}
```

ATTRIBUTE ROUTING(2)

- See more: page 1771
- What will the following return?
- https://localhost:44352/
- https://localhost:44352/WA
- https://localhost:44352/WA/Student/Cali
- https://localhost:44352/WA/Show
- https://localhost:44352/WA/Show/2
- https://localhost:44352/ WA/Show/1/2

```
// This method gets called by the runtime. Use this method
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(x => x.EnableEndpointRouting = false);
}

// This method gets called by the runtime. Use this method
public void Configure(IApplicationBuilder app)
{
    app.UseMvc();
}
```

```
[Route("WA")]
public class StudentController : Controller
{
    [Route("")]
    [Route("Student")]
    [Route("Student/{state?}")]
    public string CallMe(string state)
    {
        return "Evergreen State loves " + state;
    }

    [NonAction]
    public IActionResult Show()
    {
        return Content("No student selected");
    }
    [Route("Show/{id}")]
    public IActionResult Show(int id)
    {
        Student st = new Student();
        st.FirstName = "Mary"; st.Major = "CS"; st.GPA = 4.0;

        return View(st);
    }
}
```

HTTP VERB ATTRIBUTES

- See more on page 1774

- One can use the **Http[Verb]** attributes instead of using the **Route** attribute.
  - **Http[Verb]** attributes - used when you want to limit access to the action to a specific **HTTP** verb.

- If you want an action to run only when the HTTP verb is **GET**, you can use the **HttpGet** attribute.
- If you want an action to run only when the HTTP verb is **POST**, you can use the **HttpPost** attribute.

- We'll talk about those verbs soon.
  - So far, all our requests were **GET** requests

- For example, what will the following do?
  - https://localhost:44352/WA/Student/Cali
  - https://localhost:44352/WA/Show/1

```
[Route("WA")]
public class StudentController : Controller
{
    [Route("")]
    [Route("Student")]
    [HttpGet("Student/{state?}")]
    public string CallMe(string state)
    {
        return "Evergreen State loves " + state;
    }

    [NonAction]
    public IActionResult Show()
    {
        return Content("No student selected");
    }
    [Http("Show/{id}")]
    p HttpDelete                sult Show(int id)
    { HttpGet
      HttpHead
      HttpOptions               new Student();
      HttpPatch                 = "Mary"; st.Major = "CS"; st.GPA = 4.0;
      HttpPost                  class Microsoft.AspNetCore.Mvc.HttpPostAttribute
      HttpPut                   Identifies an action that supports the HTTP POST method.
    } RequireHttps
}
```

IN-CLASS DEMO: HOW TO ADD ROUTES

Demonstration: How to Add Routes

- Source/Steps:
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD04_DEMO.md#demonstration-how-to-add-routes

LAB/HOMEWORK: EXPLORING ASP.NET CORE MVC

- **Module 04**
  - Exercise 1: Adding Controllers and Actions to an MVC Application
  - Exercise 2: Configuring Routes by Using the Routing Table
  - Exercise 3: Configuring Routes by Using Attributes
  - ~~Exercise 4: Adding an Action Filter~~ ← PLEASE SKIP(we did not cover it in class)

- You will find the **high-level** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD04_LAB_MANUAL.md

- You will find the **detailed** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD04_LAK.md

- For your homework submit one zipped folder with your complete solution.