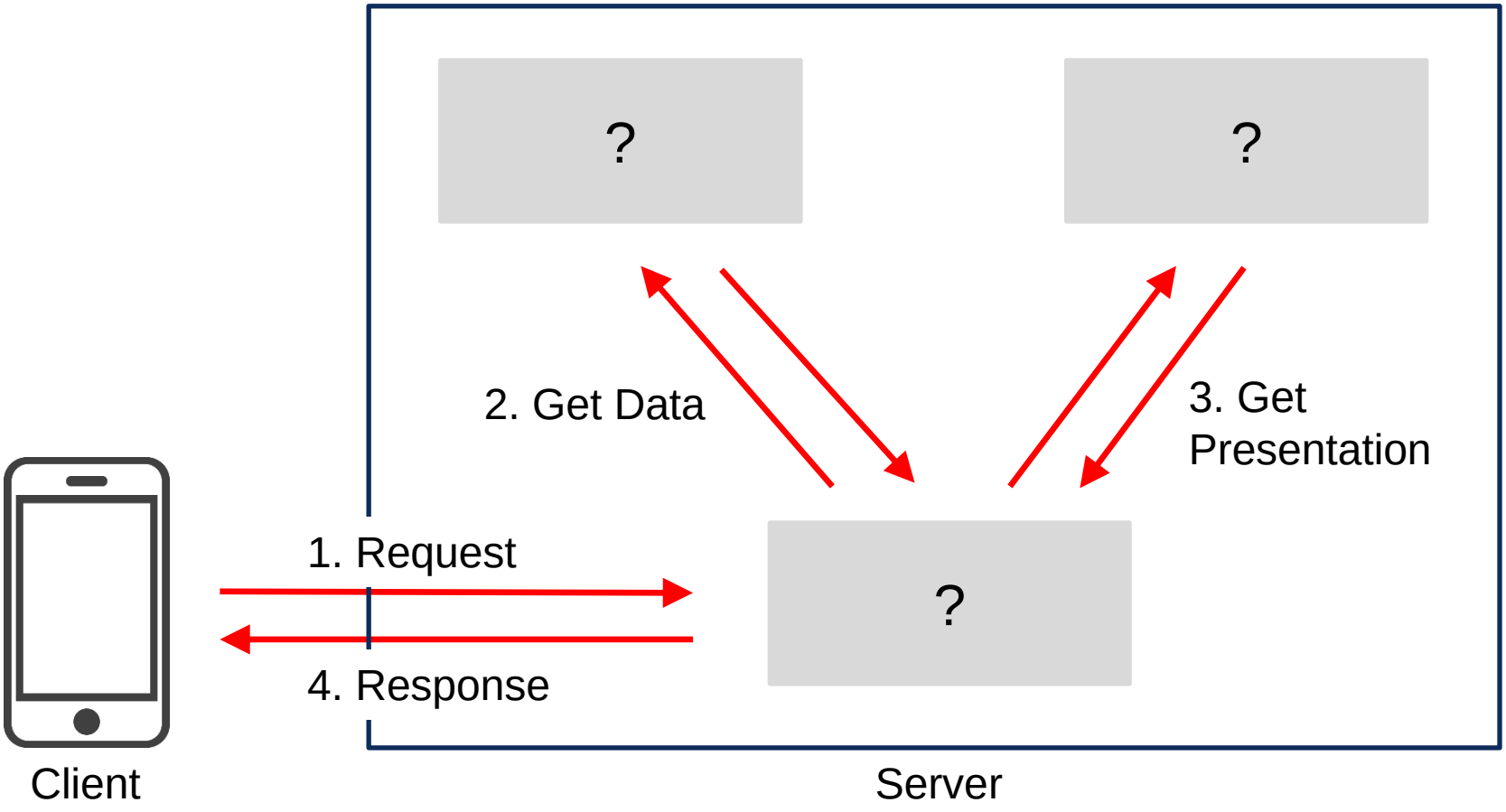
ASP.NET CORE

ROUTING

issntt@nus.edu.sg

Review Question

In ASP.NET Core, when a request reaches the **MVC Middleware**, what component will take care of the request **first**?



Objectives

At the end of this lesson, students will be able to

- Describe Controllers and Action Methods in ASP.NET Core web apps and their responsibilities
- Describe the roles of Routing in ASP.NET Core
- Describe different parts of a URL and how URL path is used for Routing
- Describe the concepts of routing templates, placeholders, fragments, literal values, routing parameters, route values, optional parameters, default route values, catch-all parameters, HTTP verbs and make use of each of them to configure Routing
- (Most important) Configure Routing for ASP.NET Core web apps using Conventional and Attributed Routings

Topics

- **Controllers and Action Methods revisit**
- URLs and Configuring Conventional Routes
- Routing Templates and Routing Parameters
- Attribute-Based Routing (Self-Study)
- Conventional vs Attributed Routing (Self-Study)

Controllers

Controllers are classes that **can be instantiated** and have a **name ending** with "*Controller*" and/or **inherit** from class ***Controller*** or class ***ControllerBase***

```
public class CourseController : Controller {
    public ActionResult Index() {
       List<Course> courses = CourseData.GetAllCourses();
       ViewData["courses"] = courses;

       return View();
    }
    public ActionResult CourseDetails(string sessionId, int courseId) {
       Course course = CourseData.GetCourseDetailsByCourseId(courseId);
       ViewData["course"] = course;

       return View();
    }
}
```

Do we need to **register** a **new controller** before it can handle requests?

Action Methods

An action method is a method **inside** a controller. It may run to **handle** requests

```
public class HelloWorldController : Controller {
    public IActionResult Index() {
        return View();
    }
    public string Welcome(
            string name, int numTimes = 1, int ID=1) {
        return HtmlEncoder.Default.Encode(
            $"Hello {name}, ID {ID}, NumTimes is: {numTimes}");
    }
}
```

Action Methods

Controllers provide a mechanism to **logically group** actions and so **apply** a **common set of rules** to them

```
public class MoviesController : Controller {
    private readonly MvcMovieContext _context;
    public MoviesController(MvcMovieContext context) {
        _context = context;
    }
    public IActionResult Search(string movieGenre, string searchString) {
        // method implementation
    }
    public IActionResult Details(int? id) {
        // method implementation
    }
    public IActionResult Create() {
        // method implementation
    }
}
```

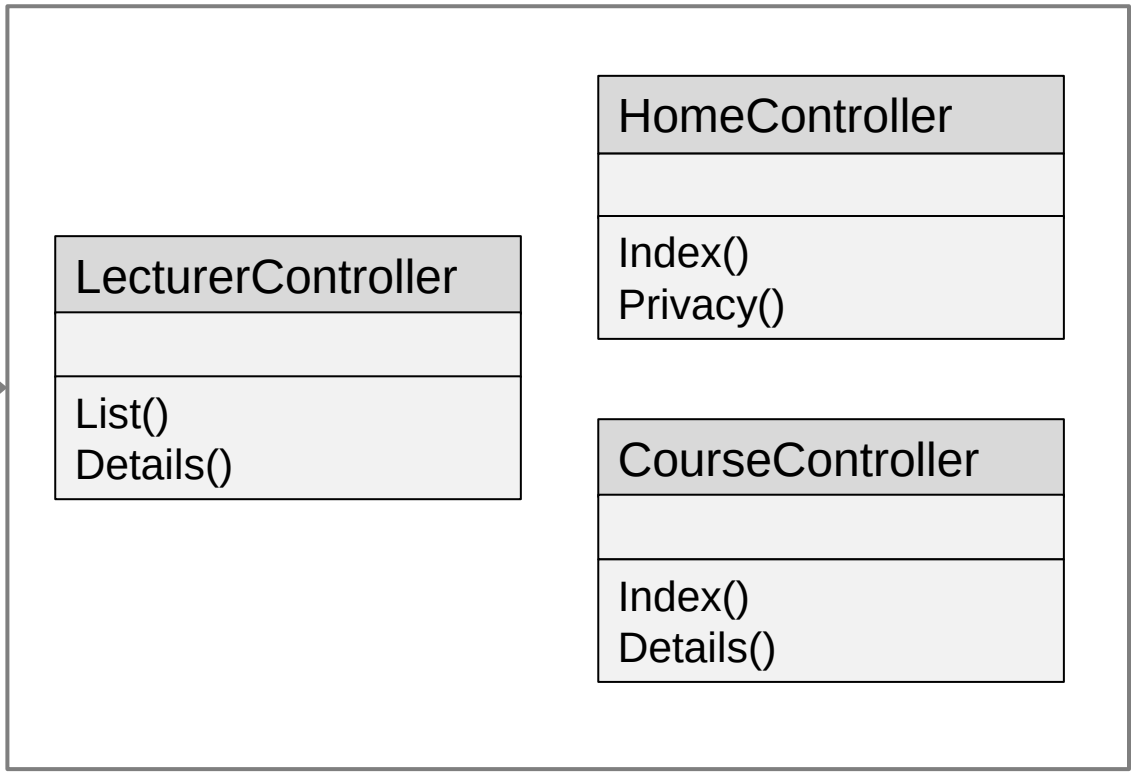
What do ***Search()***, view ***Details()***, and ***Create()*** movies **commonly need**?

Problem

HTTP Request

```
http://localhost
:56563/Lecturer/
Details/3
```

?



Given a request, how does ASP.NET Core know **which controller** and **which action** to handle?

Topics

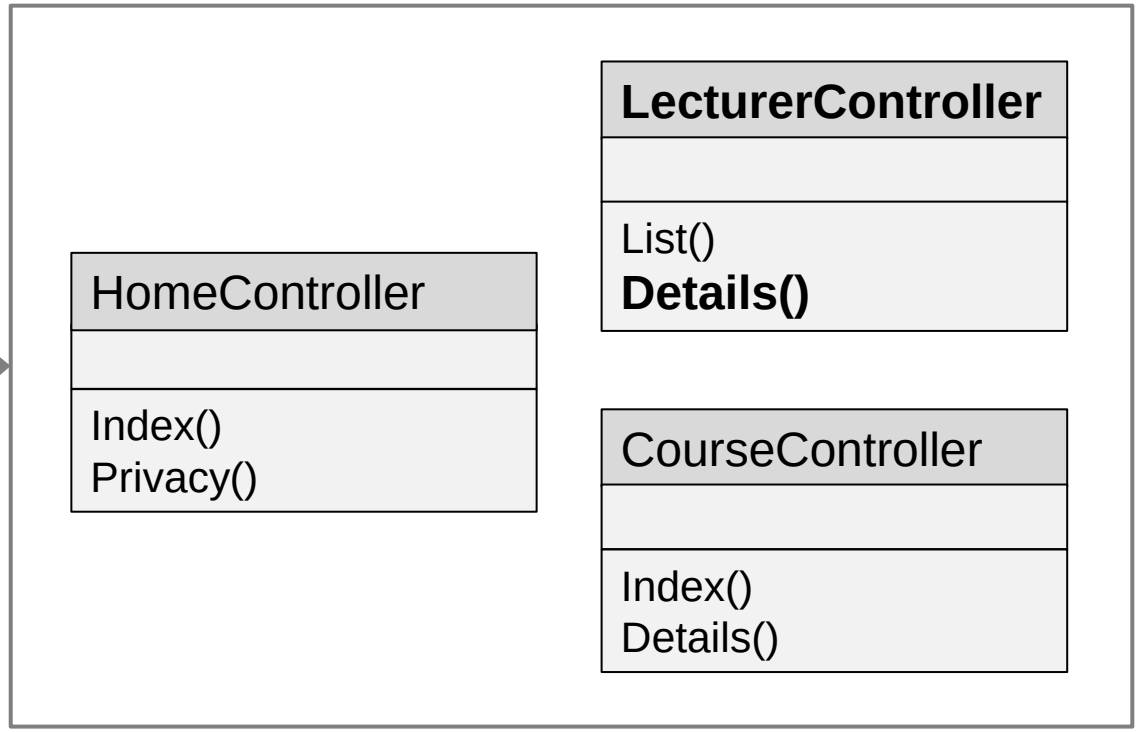
- Controllers and Action Methods revisit
- **URLs and Configuring Conventional Routes**
- Routing Templates and Routing Parameters
- Attribute-Based Routing (Self-Study)
- Conventional vs Attributed Routing (Self-Study)

Routing

Routing in ASP.NET Core is the process of **mapping** an incoming **HTTP request** to an **action method**

HTTP Request

`http://localhost:56563/Lecturer/Details/3`



We want to map the given request to *`Details()`* method. How to do that?

URL Anatomy

1 2 3 4

http://www.example.com:56563/Lecturer/Details/3

1. **Protocol**: we focus on HTTP and HTTPS (secured HTTP)
2. **Domain and subdomain**: decide **which server** to send the request to
3. **Port**: decide **which app** to send the request to
   - Because one server can run **multiple apps** (e.g., 2 .NET Core apps and 1 MS SQL app)
4. **Path**: specify the **resource path**

https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

URL Anatomy

ASP.NET Core **uses path for routing**

http://www.example.com:56563**/Lecturer/Details/3**

This part decides **which action method** will run

Protocol, domain/subdomain and port help the request to **reach** our .NET Core app

Because of that, from here, **only path** is displayed

How can we configure routing?

One simple way is to call *UseRouting()* and then *MapControllerRoute()* to configure one or more routes

```
1 app.UseRouting();
2
app.MapControllerRoute(
  a name: "default",
  b pattern: "{controller=Home}/{action=Index}/{id?}");
```

routing template

Program.cs

Call *MapControllerRoute()* to configure **each route**, providing 2a) a name and 2b) a **template**

Configure multiple routes

Multiple routes can be configured. .NET Core tries to match to the routes in **sequence**, so the **order matters**

```
app.MapControllerRoute(
    name: "default",
    pattern: "{action}/{controller}/{id}");

app.MapControllerRoute(
    name: "extra",
    pattern: "{controller}/{action}");
```

CourseController
Index() Details()

If the request URL path is **`/Course/Details/1`**, what action method will be executed? How about **/Course/Details**?

Topics

- Controllers and Action Methods revisit
- URLs and Configuring Conventional Routes
- **Routing Templates and Routing Parameters**
- Attribute-Based Routing (Self-Study)
- Conventional vs Attributed Routing (Self-Study)

Route templates

Route templates define the **patterns** of the known URLs, with **placeholders** for **parts** that may **vary**

```
{controller=Home}/{action=Index}/{id?}
{controller}/{action}/{id?}
api/{controller}/{action}/{id:customName}
```

Segments

A route template is **split** into a number of **segments**. A segment is typically **separated** by the **slash /**

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

How many segments are there in each template?

Segments

For each segment, we can define:

- **Literal values**: specific, **expected** strings
- **Route parameters: variable** segments of the URL
- **Optional values: optional** segments of a URL
- **Default values: default values** when an optional isn't provided
- …

```
api/{controller}/{action=Index}/{id?}
```

Literal values

Literal values must be **matched exactly** (ignore case) by the request URL

```
api/{controller}/{action}/{id}
```

Literal segment is defined, you know…, literally ☺

Route parameters

Route parameters are segments that **may vary** but **still match** the template

```
api/{controller}/{action}/{id}
```

Route parameters are defined by giving them a **name** and placing them in **curly brackets {}**

Game Time

Image by Joseph Samson from Pixabay

Quiz

Given this template *`about/contact`*, which URL(s) would match?

A) `about`

B) `about-us/contact`

C) `about/contact/1`

D) `about/contact/email`

Quiz

Given this template *{controller}/{action}*, which URL(s) would match?

A) `staffs/search`

B) `students`

C) `departments/list`

D) `modules/view/OOPCS`

Route values

When a request URL **matches** a route template, the **values associated** with the respective **route parameters** are **captured**, called route values

```
api/{controller}/{action}/{id}
```

```
https://localhost/api/Course/Details/1
```

Route values

Route values are stored in a **dictionary** of key (parameter) / value pairs. At least, **values** for **`controller`** and **`action`** are **required**

```
api/{controller}/{action}/{id}
```

```
https://localhost/api/Course/Details/1
```

Key	Value
controller	Course
action	Details
id	1

Optional parameters

Route parameters can be declared **optional**. Their respective **values** are **captured only if** they **present**

```
api/{controller}/{action}/{id?}
```

Optional are defined by giving them a **name** with a **question mark ?** and placing them in **curly brackets {}**

Optional parameters with default values

For **optional parameters**, **default values** can be specified

```
api/{controller}/{action=Index}/{id?}
```

Values of optional parameters are specified with **= operator**

Quiz

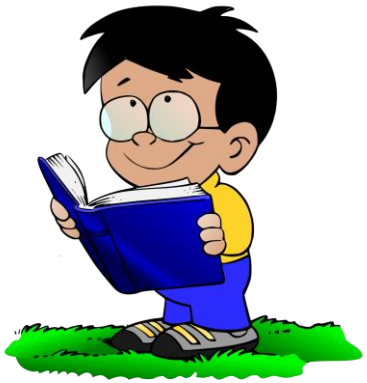
Given the template *{controller}/{action}*, what are route values for each case?

A) `/staffs/search`

B) `/students/`

C) `/departments/list`

D) `/modules/view/OOPCS`



Quiz

Given the template *{controller}/{action=Index}/{id?}*, what are route values for each case?

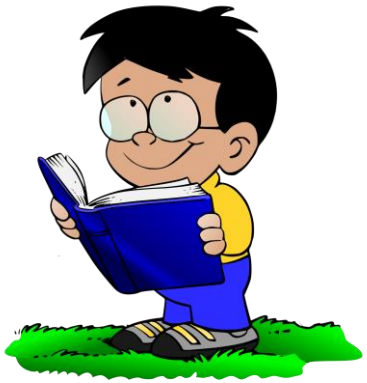
A) `/staffs/search/2`

B) `/students/view`

C) `/modules/edit/OOPCS`

D) `/departments/index`

E) `/departments/`



Another option for default values

Self study

Default values and can also be defined using **anonymous objects**

```
app.MapControllerRoute(
  name: "with_defaults",
  pattern: "{controller}/{id}/{action}/",
  defaults: new
   {
     id = 1,
     action = "Details"
   }
);
```

The default values to use when a parameter is missing

Anonymous objects https://www.youtube.com/watch?v=u8C9iO_4yIQ

Next

Let's say that we have 2 related scenarios:

1. **Display** a Login form for users from URL */Account/Login*
2. **Submit** a Login form that users have filled to the **same URL**

```
public class AccountController :
                              Controller {
    public IActionResult Login() {
        return View();
    }

    public IActionResult Login(
        string email,
        string password) {
        /* method implementation */
    }
}
```

Given the **same URL**, how to match to two **different action methods** in different scenarios?

Handle multiple matching actions

Use **HTTP verbs**, such as *[HttpPost]*, to constrain matching to a type of HTTP Request only, such as HTTP Post

```
public class AccountController : Controller {
  1 [HttpGet]
    public IActionResult Login() {
        // method implementation
        return View();
    }

  2 [HttpPost]
    public IActionResult Login(string email, string password) {
        // method implementation
        return null;
    }

}
```

For a matched GET request, which action will handle?

HTTP Verb Templates

Convention Based Summary

Up to now, the routing we declare is called **Convention Based**

One or more **global routes** are defined and MVC will try to map to **all** incoming **request** URLs

The routes are **defined** in **one place**

```
app.UseRouting();

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

Program.cs

Problem

Now, consider this scenario

```
public class BookCategoriesController : Controller {
    public IActionResult ListAllCurrentCategoriesWithProducts()
    {
        // Method implementation
    }
}
```

Using the Convention Based, the action is likely mapped to */BookCategories/ListAllCurrentCategoriesWithProducts*

The mapped URL is **too long**. How to make it **shorter**?

This problem is especially **common** when **developing Web API** apps

Topics

- Controllers and Action Methods revisit
- URLs and Configuring Conventional Routes
- Routing Templates and Routing Parameters
- **Attribute-Based Routing (Self-Study)**
- Conventional vs Attributed Routing (Self-Study)

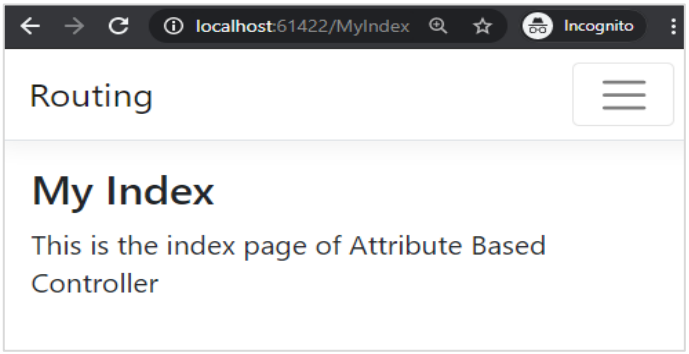
Attribute-Based Routing

Self study

Ties a given **URL pattern** to a **specific action method** by **placing** *[Route]* attributes on the **method** itself

```
public class AttributeBasedController
                        : Controller {
   [Route("MyIndex")]
   public IActionResult Index() {
      return View();
   }
   // ...
}
```

Map to **/MyIndex**




```
public class AttributeBasedController
                        : Controller {
   [Route("Hello/World")]
   public IActionResult HelloWorld() {
      return View();
   }
   // ...
}
```

Map to **/Hello/World**

Attribute-Based Routing

Self study

A **single** action method can be mapped to **multiple URLs**

```
public class CarController : Controller
{
   [Route("car/start")]
   [Route("car/ignition")]
   [Route("start-car")]
   public IActionResult Start()
   {
      // method implementation
   }
   // ...
}
```

Map to
/Car/Start
/Car/Ignition
/Start-Car

Is this a good idea ?

In general, no. We really need to know why we provide the options

Attribute-Based Routing

Self study

A Route Attribute can **contain route parameters**, just like in Convention-Based

```
public class CarController : Controller
{
   [Route("car/speed/{speed}")]
   public IActionResult SetCarSpeed(int speed)
   {
      // method implementation
   }
   // ...
}
```

Route parameters are **handled** in the **same way** as for Convention-Based

Next

Consider this scenario

```
public class TodoController : Controller {
    [Route("api/todo/list")]
    [Route("list-todo")]
    public IActionResult Index() {
        // method implementation
    }
    [Route("api/todo/search")]
    public IActionResult Search(string term) {
        // method implementation
    }
    [Route("api/todo/details/{id}")]
    public IActionResult Details(int id) {
        // method implementation
    }
}
```

Except "**list-todo**", all routes start with "**api/todo**"

Can we **remove** the duplicate?

Attribute-Based Routing

Self study

Apply *[Route]* to **controllers** to **combine** it with the routes in action methods

```
1 [Route("api/todo")]
public class TodoController : Controller
{
2 [Route("list")]
3 [Route("/list-todo")]
   public IActionResult Index() {
      // method implementation
   }

2 [Route("search")]
   public IActionResult Search(string term) {
      // method implementation
   }

2 [Route("details/{id}")]
   public IActionResult Details(int id) {
      // method implementation
   }
}
```

1. When we add Route Attribute to the Controller**,**
2. For Route Attribute in a method that **does not start** with a **slash /**, the overall route template is **combined**
3. For Route Attribute in a method that **starts** with slash /, **only** route templates in **action methods counts**

Which action method will match for each case?

`/list`

`/list-todo`

`/api/todo/details/2`

none, Index(), Details()

Handle multiple matching actions

Self study

Use **HTTP verbs** with templates to distinguish the actions where **same URL** is matched

```
[Route("api/todoItem")]
public class TodoItemController : Controller {
   [HttpGet]
   public IActionResult GetTodoItems() {
      // method implementation
   }
   [HttpGet("{id}")]
   public IActionResult GetTodoItem() {
      // method implementation
   }
   [HttpPut("{id}")]
   public IActionResult UpdateTodoItem() {
      // method implementation
   }
   [HttpDelete("{id}")]
   public IActionResult DeleteTodoItem() {
      // method implementation
   }
}
```

```
or these two lines:
[HttpGet]
[Route("{id}")]
```

```
or these two lines:
[HttpPut]
[Route("{id}")]
```

```
or these two lines:
[HttpDelete]
[Route("{id}")]
```

Which action method will match: `/api/todoItem/4`

Depending on the Request method

Topics

- Controllers and Action Methods revisit
- URLs and Configuring Conventional Routes
- Routing Templates and Routing Parameters
- Attribute-Based Routing (Self-Study)
- **Conventional vs Attributed Routing (Self-Study)**

Conventional and Attribute

Self study

Convention-Based and Attribute-Based routing can be **used together** in the **same** app

1
```
app.MapControllerRoute(
  name: "default",
  pattern:
      "{controller=Home}/{action=Index}/{id?}");
```

1
```
public class HomeController :
                    Controller
{
  public IActionResult Index() {
     // method implementation
  }

  public IActionResult Privacy()
  {
     // method implementation
  }
}
```

2
```
[Route("api/todoItem")]
public class TodoItemController :
                          Controller {
  [HttpGet("{id}")]
  public IActionResult GetTodoItem() {
     // method implementation
  }
  [HttpPut("{id}")]
  public IActionResult UpdateTodoItem() {
     // method implementation
  }
}
```

Which action will match?

/Home/Privacy

/TodoItem/GetTodoItem

```
HomeController::Privacy()
None
```

Conventional and Attribute

Self study

Attribute routing on an action or a controller will **make** the **action unreachable** by **conventional routing**

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

```
public class HomeController :
                    Controller {
   [Route("view-index")]
   public IActionResult Index()
   {
      // method implementation
   }

   public IActionResult Privacy()
   {
      // method implementation
   }
}
```

Which actions?

/Home/Privacy

/Home/Index

/view-index

```
HomeController::Privacy()
None
HomeController::Index()
```

Best Practices

1. Use **conventional** routing for **Web MVC** controllers
2. Use **attribute** routing for **Web API** controllers
3. Add a **prefix** such as "**`api`**" to the Web API route templates
   - to **separate** Web API **URL space** from MVC URL space

Readings

- ASP.NET Core in Action, Chapter 5 & 9, *Andrew Lock*
- Routing in ASP.NET Core https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-6.0