MVC – Controller

.NET CORE

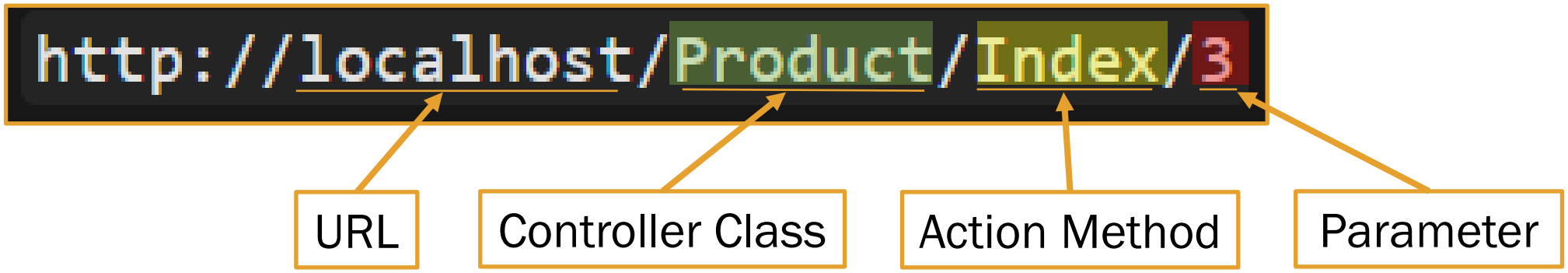
*MVC **controllers** are responsible for responding to requests made against an ASP.NET MVC website. Each browser request is mapped to a particular **controller**.*

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs#understanding-controllers

ASP.NET URL Routing

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs

The default route in an ASP.NET Core application maps the first segment of a URL to a controller name, the second segment of a URL to a controller action, and the third segment to a parameter named id. The below Url equates to calling *Product.Index(3)*



The routing configuration includes defaults for all three parameters. If a controller is not supplied, the controller defaults 'Home'. If an action is not supplied, the action defaults to 'Index'. If there's no id, the id parameter defaults to an empty string.

Controllers

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs

If the URL, ***http://localhost/Product/Index/3***, is entered into the browser, a ***controller*** named Product***Controller*** is invoked. The Product***Controller*** is responsible for generating the response to the browser request. Convention (and default) is to name the ***controller*** the same as the Url.

A ***controller*** is a regular C# class that derives from the base ***System.Web.Mvc.Controller*** class.

The ***controller*** inherits useful methods from the ***Controller*** base class.

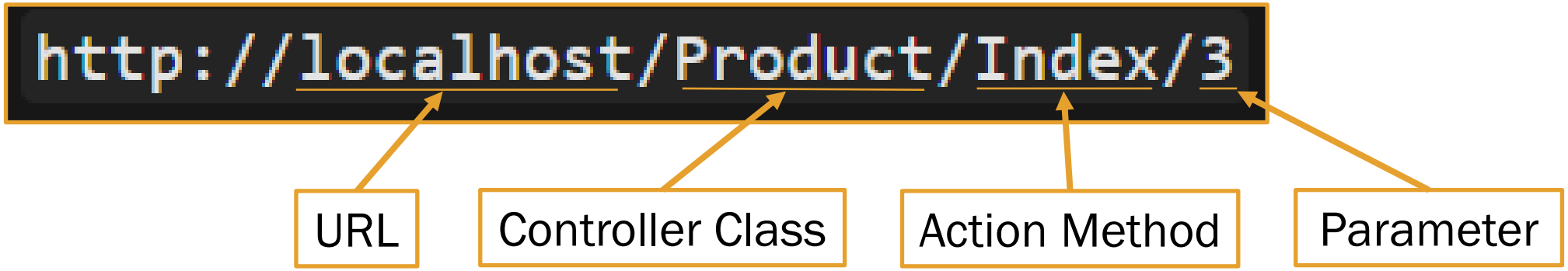
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;

namespace MvcApplication1.Controllers
{
    public class ProductController : Controller
    {
        //
        // GET: /Products/

        public ActionResult Index()
        {
            // Add action logic here
            return View();
        }
    }
}
```

Controller Actions

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs#understanding-controller-actions

---

An ***action*** is a method on a ***controller*** class that gets called when you enter a URL with the ***action*** specified. If you make a request for URL, ***http://localhost/Product/Index/3***, the product ***controller*** is called and the Index ***action*** is invoked with the parameter "3".



A ***controller action*** must be a ***public method*** of a ***controller*** class. Any ***public method*** in a ***controller*** class is exposed as a ***controller action.*** A ***method*** used as a ***controller action*** cannot be overloaded.

ActionResults

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs#understanding-action-results

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult Index(string id)
    {
        return View();
    }
}
```

```
public class StatusController : Controller
{
    public ActionResult Index()
    {
        return Content("Hello World!");
    }
}
```

When a browser makes a request, a ***controller action*** returns a response in the form of an ***ActionResult***. All ***ActionResults*** inherit from the ***ActionResult*** Class. Conventionally, an ***ActionResult*** is returned through a method of the ***Controller*** base class instead of returning an ***action result*** directly.

ActionResult	Controller Base Class Method	Purpose
ViewResult	Return View();	Returns a HTML ***View*** to the browser
EmptyResult	Return new EmptyResult()	Returns empty (void)
RedirectResult	Return RedirectToAction(Index);	Calls another ***Action*** Method.
ContentResult	Content("Hello World!");	Returns text to the browser

```
public class HomeController : Controller
    {
        // GET: for main view
        public EmptyResult EmptyData()
        {
        return new EmptyResult();
        }

    }
```

Filter Attributes

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs#the-different-types-of-filters
https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-filters#setting-an-authentication-filter

---

The ASP.NET MVC framework supports four different types of filters, listed below in order of execution.

- Authorization filters – Implements the IAuthorizationFilter attribute. Always executes first.
- Action filters – Implements the IActionFilter attribute.
- Result filters – Implements the IResultFilter attribute.
- Exception filters – Implements the IExceptionFilter attribute. Always executes last. Can be used for logging.

To implement any filter, create a class that inherits the base ***Filter*** class and implements one or more of the ***IAuthorizationFilter***, ***IActionFilter***, ***IResultFilter***, or ***IExceptionFilter*** interfaces.

```
[Authorize] // Require authenticated requests.
public class HomeController : ApiController
{
    public IHttpActionResult Get() { . . . }

    [IdentityBasicAuthentication] // Enable Basic authentication for this action.
    public IHttpActionResult Post() { . . . }
}
```

Action Filters

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/understanding-action-filters-cs

An ***action filter*** is an ***attribute*** that you can apply to a ***controller action*** or an entire ***controller*** that modifies the way in which the ***action*** is executed. The ASP.NET MVC framework includes several ***action filters:***

- OutputCache – caches the output of a controller action for a specified amount of time.
- HandleError – handles errors raised when a controller action executes.
- Authorize – enables you to restrict access to a user or role.
- You can also create custom filters.

You can apply more than one action filter at a time.

```
using System;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class DataController : Controller
    {
        [OutputCache(Duration=10)]
        public string Index()
        {
            return DateTime.Now.ToString("T");
        }
    }
}
```

This ***OutputCache*** filter causes the returned value to be cached for 10 seconds. If you repeatedly invoke the Index() ***action*** by hitting the Refresh button, you will see the same time for 10 seconds.

Request Parameters

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/creating-a-route-constraint-cs

The last part of the Url is the query parameter. This value is passed in the ***Action*** Method where it can be acted upon.

To prevent an invalid value from being passed to the ***Action*** Method, you can

- define constraints when defining routes.
- check the value of the parameter and ***RedirectToAction()*** to handle the input correctly.

```
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class CustomerController : Controller
    {
        public ActionResult Details(int? id)
        {
            if (!id.HasValue)
                return RedirectToAction("Index");

            return View();
        }

        public ActionResult Index()
        {
            return View();
        }

    }
}
```

HTTP Verbs in ASP.NET MVC

https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api

Instead of using the naming convention for ***HTTP verbs***, you can explicitly specify the ***HTTP verb*** for an ***action*** by decorating the ***action*** method with one of the following ***attributes***:

[HttpGet]	[HttpPut]
[HttpDelete]	[HttpOptions]
[HttpPatch]	[HttpPost]
[HttpHead]	

Use ***attributes*** to specify the allowed ***HTTP verbs***. You can override the ***action*** name given in the URL by using the ***[ActionName] attribute***. ***api/products/thumbnail/id*** maps to both of the below ***Action*** Methods.

```
public class ProductsController : ApiController
{
    [HttpGet]
    [ActionName("Thumbnail")]
    public HttpResponseMessage GetThumbnailImage(int id);

    [HttpPost]
    [ActionName("Thumbnail")]
    public void AddThumbnailImage(int id);
}
```

ASP.NET Core MVC Tutorial

Complete the **ASP.NET Core MVC** tutorial here