

# Routing in ASP.NET Core

3 Comments

← [Controllers](#)

[Attribute Routing](#) →

One of the most important parts of the MVC Architecture is the Routing engine. It is the Routing engine that decides to invoke controller for a particular request. In this Tutorial, we will look at How the Routing Works in ASP.NET Core app.

## Table of Content


[What is Routing](#)

[How Routing Works](#)

[What is a Route](#)

[What is a Route Collection](#)

[What is a Route Handler](#)

 [MVCRouteHandler](#)

## How to Set up Routes

Convention-based routing

Attribute routing

Convention Based Routing

URL Patterns

URL Matching

Routing in Action

Register the Route

Parameters to Controller Action Method

Route Defaults

Multiple Routes

Order matters. First Route wins

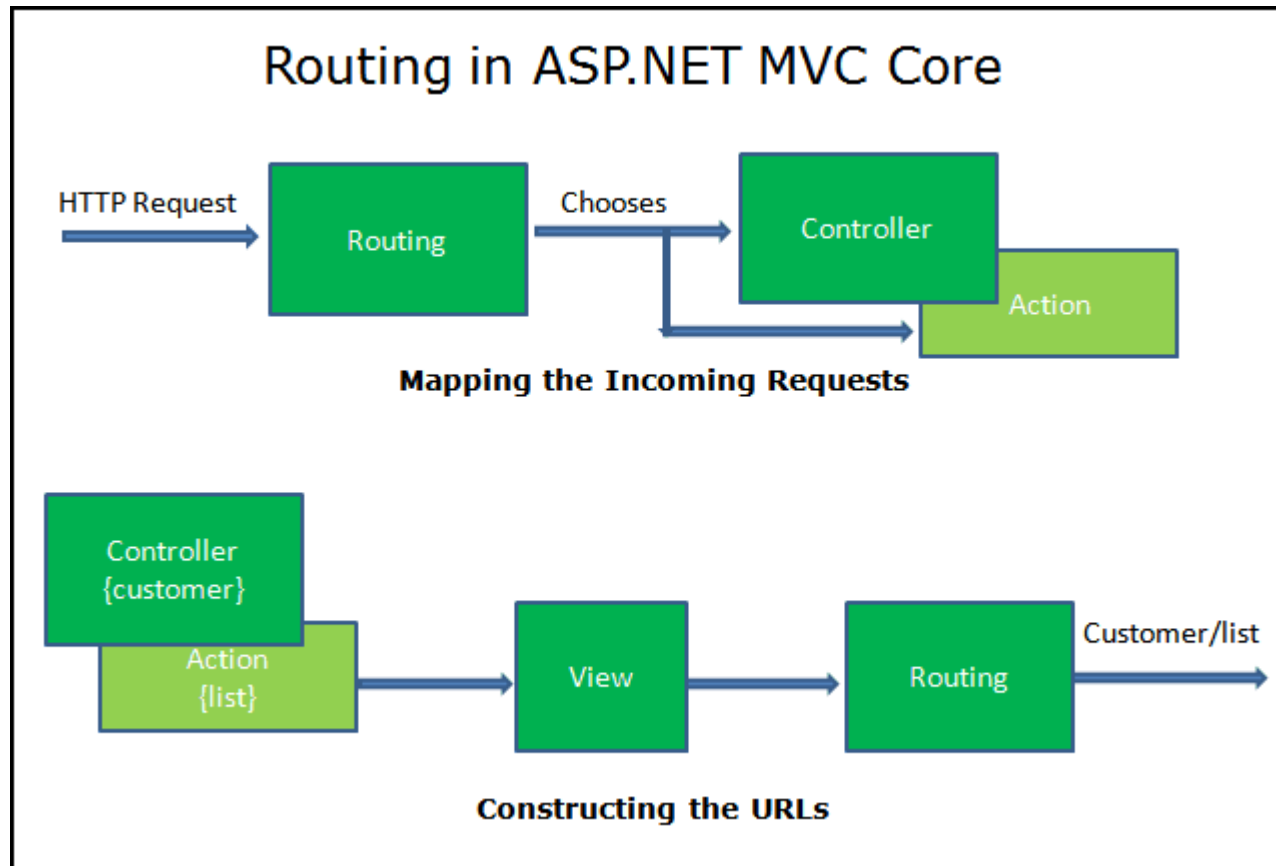
Summary

## What is Routing

The **Routing** is the Process by which ASP.NET Core inspects the incoming URLs and maps them to Controller Actions. It also used to generate the outgoing URLs. This process is handled by the Routing Middleware. The Routing Middleware is available in [Microsoft.AspNetCore.Routing](https://www.nuget.org/packages/Microsoft.AspNetCore.Routing) Namespace.

The Routing has two main responsibilities:

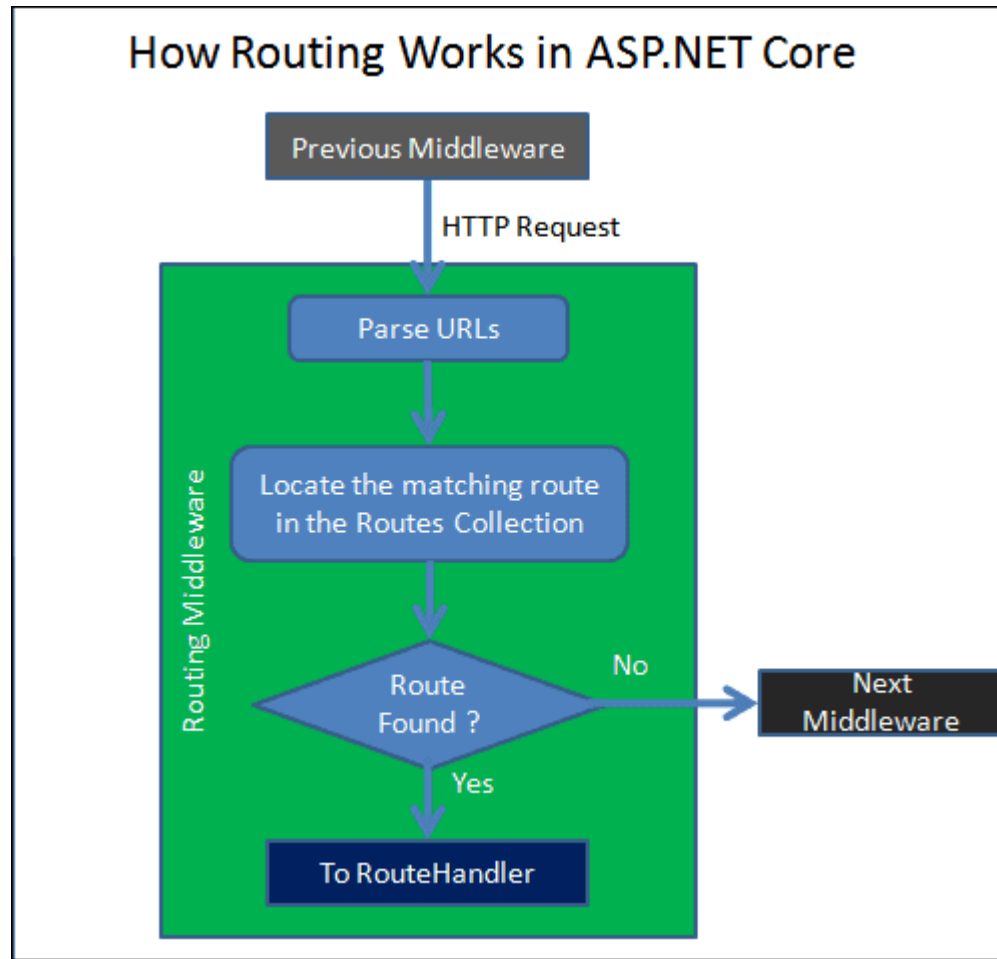
1. It maps the incoming requests to the Controller Action
2. Generate an outgoing URLs that correspond to Controller actions.



## How Routing Works

The following diagram shows how the Routing Works in ASP.NET Core apps.





When the Request arrives at the Routing Middleware it does the following.

1. It Parses the URL.
2. Searches for the Matching Route in the RouteCollection.
3. If the Route found then it passes the control to RouteHandler.
4. If Route not found, it gives up and invokes the next Middleware.



## What is a Route

The `Route` is similar to a roadmap. We use a roadmap to go to our destination. Similarly, the ASP.NET Core Apps uses the `Route` to go to the controller action.

The Each `Route` contains a Name, URL Pattern (Template), Defaults and Constraints. The `URL Pattern` is compared to the incoming URLs for a match. An example of URL Pattern is `{controller=Home}/{action=Index}/{id?}`

The [`Route`](#) is defined in the `Microsoft.AspNetCore.routing` namespace.


## What is a Route Collection

The `Route Collection` is the collection of all the `Routes` in the Application. An app maintains a single in-memory collection of `Routes`. The `Routes` are added to this collection when the application starts. The Routing Module looks for a `Route` that matches the incoming request URL on each available `Route` in the `Route collection`.

The [`Route Collection`](#) is defined in the namespace `Microsoft.AspNetCore.routing`.

## What is a Route Handler

The Route Handler is the Component that decides what to do with the route.

When the routing Engine locates the `Route` for an incoming request, it invokes the associated `RouteHandler` and passes the `Route` for further processing. The `Route handler` is the class which implements the `IRouteHandler` rface.

In the ASP.NET Core, the `Routes` are handled the `MvcRouteHandler`.

## MVCRouteHandler

This is the Default `Route Handler` for the ASP.NET Core MVC Middleware.

The `MVCRouteHandler` is registered when we register the MVC Middleware in the Request Pipeline. You can override this and create your own implementation of the `Route Handler`.

The [MVCRouteHandler](#) is defined in the namespace `Microsoft.AspNetCore.Mvc`.

The `MVCRouteHandler` is responsible for invoking the `Controller Factory`, which in turn creates the instance of the `Controller` associated the `Route`. The `Controller` then takes over and invokes the `Action method` to generate the `View` and Complete the Request.

## How to Set up Routes

There are two different ways by which we can set up routes.

1. Convention-based routing
2. Attribute routing

### Convention-based routing



The Convention based Routing creates routes based on a series of conventions, defined in the ASP.NET Core `Startup.cs` file.

## Attribute routing

Creates routes based on `attributes` placed on `controller actions`.

The two routing systems can co-exist in the same system. Let's first look at Convention-based routing. We will look at the Attribute Based Routing in another tutorial

## Convention Based Routing

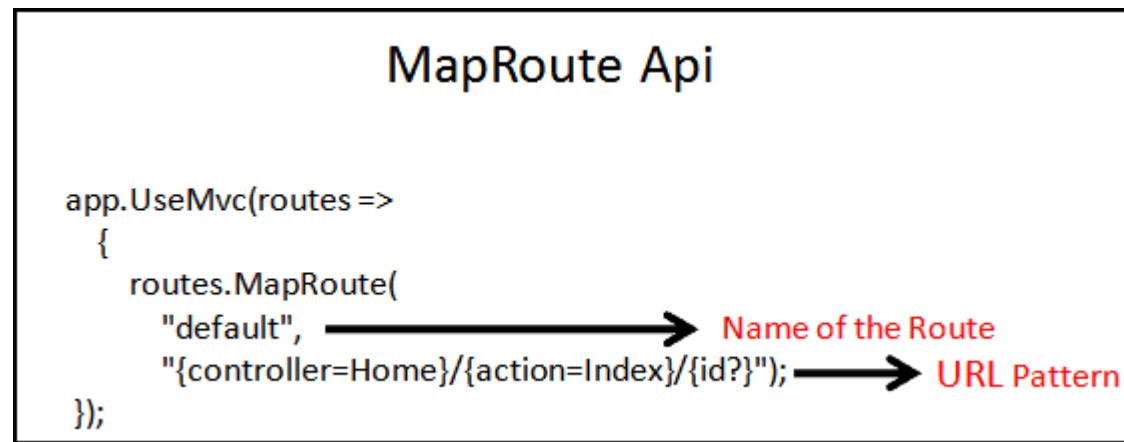
The Convention based Routes are configured in the `Configure` method of the `Startup` class.

The Routing in is handled by the Router Middleware. ASP.NET MVC adds the routing Middleware to the Middleware pipeline when using the `app.UseMvc` or `app.UseMvcWithDefaultRoute`.

The `app.UseMvc` method gets an instance of the [RouteBuilder](#) class. The `RouteBuilder` has an extension method [MapRoute](#) which allows us to add `Route` to the `Routes Collection` as shown below.

The Routing engine is given a `Route` to follow using `routes.MapRoute` API





In the above example, the `MapRoute` creates a single route, which is named as `default` and the `URL Pattern` of the route is `{controller=Home}/{action=Index}/{id?}`


## URL Patterns

The Each route must contain a `URL pattern`. This Pattern is compared to an incoming URL. If the pattern matches the URL, then it is used by the routing system to process that URL.

Each URL Pattern consists of one or more "segments." The Segments are delimited by the forward slash character

Each segment can be either a `Constant` (literal) or `Route Parameter`.

The `Route Parameters` are wrapped in curly braces for example `{controller}`, `{action}`.

 `Route Parameters` can have default value like `{controller=Home}`, where `Home` is the default value for the controller. An equals `=` sign followed by a value after the `Route Parameter` name defines a default value for the

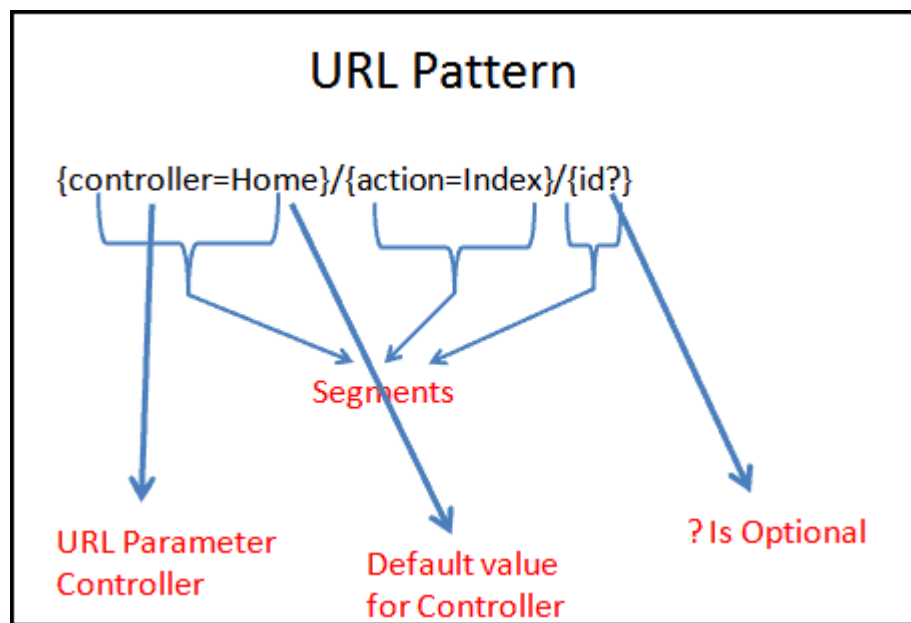


parameter.

You can also have the Constant segments as shown in this route. `admin/{controller=Home}/{action=Index}/{id?}`. Here `admin` is a Constant and must present in the requested URL.

The `?` in `{id?}` indicates that it is optional. A question mark `?` after the route parameter name defines the parameter as optional

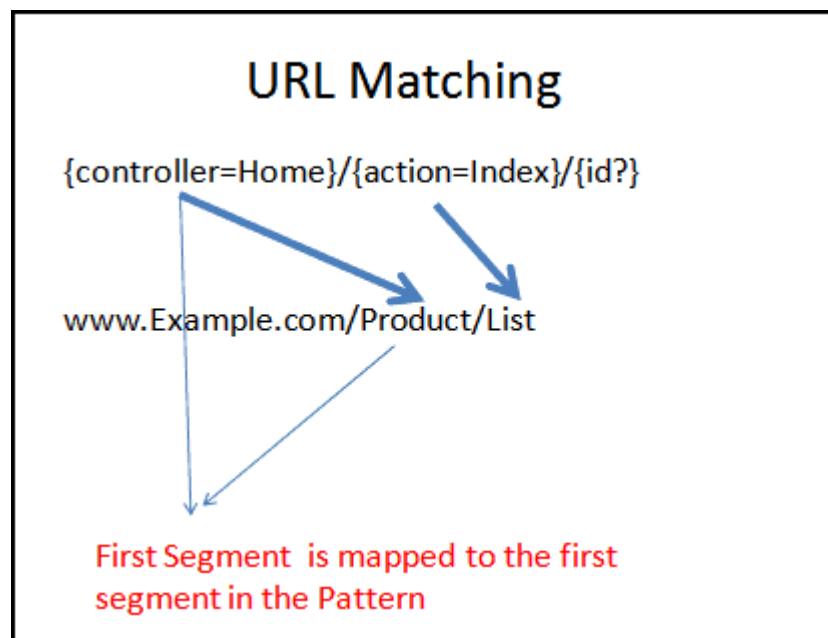
The URL Pattern `{controller=Home}/{action=Index}/{id?}` Registers route where the first part of the URL is Controller, the second part is the action method to invoke on the controller. The third parameter is an additional data in the name of `id`.



The Each segment in the incoming URL is matched to the corresponding segment in the URL Pattern. The Route `{controller=Home}/{action=Index}/{id?}` has three segments. The last one is optional.

Consider the Example URL `www.example.com/product/list`. This URL has two segments. This URL Matches with the above pattern as the third segment in the URL Pattern is Optional

The Routing engine will parse as `{controller}= Product` & `{action}= List`



The URL `www.example.com/product` also matches with the above URL Pattern, although it has only one segment. This is because the action placeholder has a default value of the Index. if there is no corresponding segments are in the URL and segment has a default value in the Pattern, then the default value is chosen by the Routing Engine.



Hence this URL Is parsed as `{controller}=Product` and `{action}=Index`

The `www.example.com` also matches with the above URL Pattern as the first segment controller has the default value of Home. This URL is Parsed as `{controller}=Home` and `{action}=Index`

The URL `www.example.com/product/list/10` is parsed as `{controller}=Home`, `{action}=Index` and `{id}=10`.

The URL `www.example.com/product/list/10/detail` fails. That is because this URL has four segments and the URL Pattern has three.

*The Previous versions of the ASP.NET returned a match, even if the Controller Action method did not exist and application returned a 404 error.*

*The ASP.NET Core routing engine checks of the existence of the Controller & Action method for the each route checked. If there is no corresponding controller and action method in the application then matching will return negative, even if the Route exists.*

## Routing in Action

Lets us build an example ASP.NET Core app to check how the routing works.



Create a new ASP.NET Core App using Framework Version ASP.NET Core 2.0. Choose the Empty Template and name the Project as MVCController.

Open the startup.cs and add open the ConfigureServices method and register the MVC Services as follows

```
1
2     public void ConfigureServices(IServiceCollection services){
3         services.AddMvc();
4     }
5
```

Now, go to the Configure method in the startup.cs and register the MVC middleware using `app.UseMvc`.

```
1
2 public void Configure(IApplicationBuilder app, IHostingEnvironment env) {
3     if (env.IsDevelopment()) {
4         app.UseDeveloperExceptionPage();
5     }
6     app.UseMvc();
7     app.Run(async (context) => {
8         await context.Response.WriteAsync("Failed to Find Route");
9     });
10 }
11
```

The `app.UseMvc` does not register any routes.



The `app.Run` is a Terminating Middleware, which in the above example displays when "Failed to Find Route". This Middleware will run only when the `app.UseMvc` fails to detect any route.

Now, Create a `Controllers` folder in the project root. Right, Click on the Controller folder, click on Add Controller. Select "MVC – Controller Empty". Enter the name of the controller as `HomeController`.

Now. Open the HomeController and change the index method to as follows

```
1  
2 public string Index() {  
3     return "Hello from Index method of Home Controller";  
4 }  
5
```

You can refer to the tutorial [Build your First ASP.NET Core MVC Application](#) on how to create a new asp net core app.

Run this app and try the following URLs. `/`, `/Home`, `/Home/Index`.

In all the above cases you will receive the "Fails to Find a Route" message. This is because we have not registered any Route.

## Register the Route

Go to the Configure method of the `Startup.cs` class and change `app.UseMvc` as follows.



```
1
2 app.UseMvc(routes => {
3     routes.MapRoute("default",
4         "{controller}/{action}");
5 });
6
```

We have not specified any defaults. Hence both the controller and action to be present in the URL.

URL	Match ?	Parsed As
/	No	
/Home	No	
/Home/Index	Yes	Controller=Home Action=Index

Now, Try this route

```
1
2 app.UseMvc(routes => {
3     routes.MapRoute("default",
4         "{controller}/{action=index}");
5 });
6
```

Now, we have a default value for action URL Parameter.



URL	Match ?	Parsed As
/	No	
/Home	Yes	Controller=Home Action=Index
/Home/Index	Yes	Controller=Home Action=Index

Now, Add the default for the Controller

```
1  
2 routes.MapRoute("default", "{controller=Home}/{action=Index}");  
3
```

URL	Match ?	Parsed As
/	Yes	Controller=Home Action=Index
/Home	Yes	Controller=Home Action=Index
/Home/Index	Yes	Controller=Home Action=Index

Now, Try this Route



```
2 routes.MapRoute("default", "{admin}/{controller=Home}/{action=Index}");  
3
```

We have added Route Parameter admin to the route. Note that Route Parameter are enclosed in Curly braces. Now, test it with the following URLs.

URL	Match ?	Parsed As
/	No No defaults for admin. Hence first segment is mandatory	
/Home	Yes  The First segment Home matches to the Admin	Admin=Home Controller=Home Action=Index
/Abc	Yes	Admin=Abc Controller=Home Action=Index
/Home/Index	No  Admin=Home Controller=Index  There is No IndexController, Hence it fails	
/Xyz/Home	Yes	Admin=Xyz Controller=Home Action=Index






URL	Match ?	Parsed As
/Admin/Home	Yes	Admin=Admin Controller=Home Action=Index

Now, Try this Route

```
1
2 routes.MapRoute("default",
3     "admin/{controller=Home}/{action=Index}");
4 });
5
```

The difference between above route and this route is that the admin is now defined as **Constant** (without curly braces). This means the first segment of the URL must contain the word admin.

URL	Match ?	Parsed As
/	No because First segment is mandatory	
/Home	No The first segment must contain the word Admin	
/Abc	No The first segment must contain the word Admin	
 Admin	Yes	Controller=Home Action=Index

URL	Match ?	Parsed As
/Admin/Home	Yes	Controller=Home Action=Index

## Parameters to Controller Action Method

Now, Consider the following Route. This is the default Route that gets registered when we use the `app.UseMvcWithDefaultRoute`.

```
1
2 app.UseMvc(routes => {
3     routes.MapRoute("default",
4         "{controller=Home}/{action=Index}/{id?}");
5 });
6
```

It has the third segment named `id`, which is Optional.

The `id` can be passed as the parameter to the Controller Action method.

*Any route parameters except `{controller}` and `{action}` can be passed as parameters to the action method*



Change the Index method of the HomeController as shown below.

```
1
2 public string Index(string id) {
3     if (id != null) {
4         return "Received " + id.ToString();
5     } else {
6         return "Received nothing";
7     }
8 }
9
```

A request for `/Home/Index/10` will match the above route and the value 10 is passed as id parameter to the Index action.

## Route Defaults

Route Defaults can be specified in two ways.

The one way is to use the equal sign (`{controller=Home}`) as shown in the previous examples.

Another way is to use the third argument of the `MapRoute` method.

```
1
2 routes.MapRoute("default",
3     "{controller}/{action}",
4     new { controller = "Home", action = "Index" });
5
```

We, create an instance of an anonymous type, which contains properties that represents the URL Parameter. Values of those properties become the default values of the URL Parameter.

The above Route is similar to the this route `{controller=Home}/{action=Index}`

## Multiple Routes

In the examples above, we used only one route

You can configure the ASP.NET Core to handle any no of routes as shown below

```
1
2 app.UseMvc(routes => {
3     routes.MapRoute("secure",
4         "secure",
5         new { Controller = "Admin", Action = "Index" });
6
7     routes.MapRoute("default",
8         "{controller=Home}/{action=Index}");
9 });
10
```

The example above has two routes. Each route must have a Unique Name. We have named it as `secure` and `default`.

The first route is interesting. It has only one segment. We have setup defaults values for `controller` and `action method` on this route. The Controller & Action defaults to Index method of the AdminController.

Create an empty AdminController.cs under Controllers folder. Change the Index method as shown below

```
1
2 public string Index() {
3     return "Hello from Index method of Admin Controller";
4 }
5
```

Now run the app and try out the following URLs.

URL	Match ?	Parsed As
/	Yes	Controller=Home Action=Index
/Secure	Yes	Controller=Admin Action=Index
/Secure/Test	No	
/Admin	Yes	Controller=Admin Action=Index

This goes to the AdminController not through the first route, but through the second route

## Order matters. First Route wins

Order in which routes are registered is very important.



URL Matching starts at the top and enumerates the collection of Routes searching for a match. It stops when it finds the first match.

Now consider this route

```
1
2 routes.MapRoute("Home",
3     "{home}",
4     new { Controller = "Home", Action = "Index" });
5
6 routes.MapRoute("secure",
7     "secure",
8     new { Controller = "Admin", Action = "Index" });
9
```

The first route has URL Parameter {home} and matches everything. The second route has constant "secure", which is a more specific route.

When you use the URL "/secure", it does not invoke the AdminController, but instead the HomeController. This is because the first "Secure" matches the first route.

To make this route work, move the secure route ahead of the Home Route

```
1
2 routes.MapRoute("secure",
3     "secure",
4     new { Controller = "Admin", Action = "Index" });
5
```

```
6 routes.MapRoute("Home",  
7 "{home}",  
8 new { Controller = "Home", Action = "Index" });  
9
```

## Summary

In this article, We learned how the Routing Works in ASP.NET Core app. We need to register the Routes using the `MapRoute` method of the Routing Middleware. The Routing Engine then matches the incoming URL with the `Routes` collection and when it finds a match, it invokes the `RouteHandler` ( which is by default `MVCRouteHandler`). The `MVCRouteHandler` then invokes the Controller associated with the `Route`.

[← Controllers](#)[Attribute Routing →](#)

## 3 thoughts on "Routing in ASP.NET Core"

**POONAM**

MARCH 28, 2019 AT 4:52 PM

Is it possible to route based on url ?

My url has "abc.xyz.com" where xyz.com is my domain and abc is my subdomain then it should use the

route with area else if the url is xyz.com then the route should use default routes. To be more specific here are my two routes

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

```
routes.MapRoute(  
    name: "Admin",  
    template: "{area=Admin}/{controller=Home}/{action=Index}/{id?}/{id1?}");
```

[Reply](#)

**CHANDRANATH SHETTY**

MARCH 30, 2019 AT 10:35 PM

No

abc.xyz.com or xyz.com is your domain name

The Routing engine will not look at the domain name while deciding to select the route.

[Reply](#)



**PRASHANT DHABSUHE**

NOVEMBER 11, 2017 AT 6:43 PM

Excellent article very very useful information for me good job

[Reply](#)

## Leave a Comment

Your email address will not be published.

Type here..



This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)



Copyright © 2020 TekTutorialsHub |

