URL Routing

ASP.NET MVC – REVIEW

AUG 31, 2016

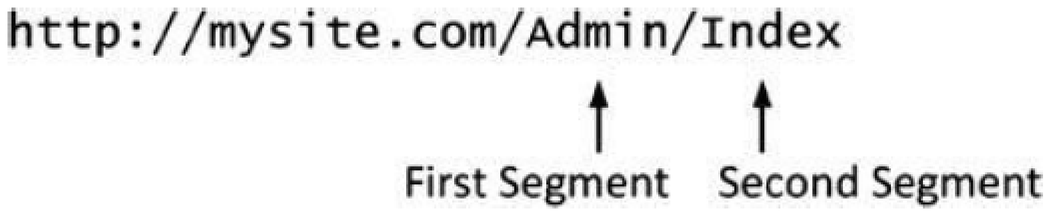
PLAN

- Default Routes
- Static, Custom, Optional segments
- Namespace priority
- Route constraints
- Attribute routing

URL Patterns

URL : http://mysite.com/Admin/Index

Pattern :



URL Patterns

- Default MVC routing pattern: {controller}/{action}

By default, a URL pattern will match any URL that has the correct number of segments

Request URL	Segment Variables
http://mysite.com/Admin/Index	controller = Admin action = Index
http://mysite.com/Index/Admin	controller = Index action = Admin
http://mysite.com/Apples/Oranges	controller = Apples action = Oranges
http://mysite.com/Admin	No match - too few segments
http://mysite.com/Admin/Index/Soccer	No match - too few segments

Note The routing system does not have any special knowledge of controllers and actions. It just extracts values for the segment variables.

Static URL Segments

You can add static segments to you URL schema:

- To parse only URLs with prefix e.g. ‘Public’
```
  routes.MapRoute("", "Public/{controller}/{action}",
        new { controller = "Home", action = "Index" });
```
- To parse only URLs that contains both static and variable
```
  routes.MapRoute("", "X{controller}/{action}");
```
- **To** create an alias for a specific URL
```
  routes.MapRoute("ShopSchema", "Shop/{action}",
        new { controller = "Home" });
```

Custom URL Segments

The controller and action segment variables a default MVC functionalities. But you are free to add additional segment variables.

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}",
    new { controller = "Home", action = "Index",
    id = "DefaultId" });
```

Caution! Some names are reserved and not available for custom segment variable names. These are *controller*, *action*, and *area*. The meaning of the first two is obvious, and I will explain areas in the next chapter.

Custom URL Segments

You can retrieve custom variable value by:

- Using RouteData.Values["id"] syntax
- Using Action Method parameters

It is possible to define custom URL variable as optional:

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}",
new { controller = "Home", action = "Index",
id = UrlParameter.Optional });
```

Custom URL Segments

- Optional custom variables are frequently used to Enforce Separation of Concerns.

The overall difference is that the default value for the id segment variable is defined in the controller code and not in the routing definition. But never the less you are able to remove the code which deals with the null value.

Variable-Length Routes

This URL pattern accepts a variable number of URL segments. This allows you to route URLs of arbitrary lengths in a single route. You define support for variable segments by designating one of the segment variables as a *catchall*, done by prefixing it with an asterisk (*)

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
    new { controller = "Home", action = "Index",
    id = UrlParameter.Optional });
```

Variable-Length Routes

URL : /
Route: controller = Home, action = Index

URL: /Customer
Route: controller = Customer, action = Index

URL: /Customer/List
Route: controller = Customer, action = List

URL: /Customer/List/All
Route: controller = Customer, action = List, id = All

URL: /Customer/List/All/Delete
Route: controller = Customer, action = List, id = All, catchall = Delete

URL: /Customer/List/All/Delete/Perm
Route: controller = Customer, action = List, id = All, catchall = Delete/Perm

Namespace priority

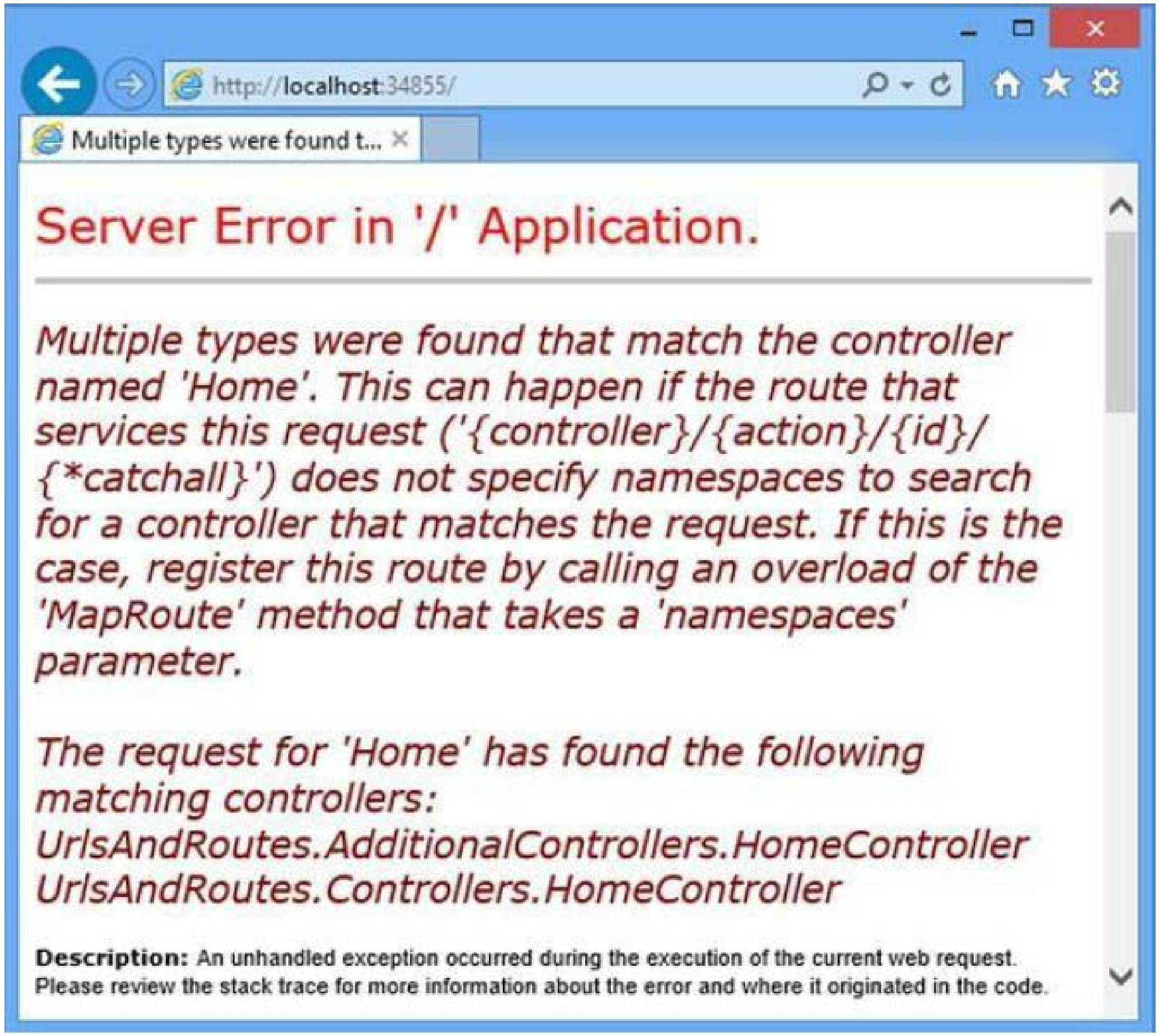
When an incoming URL matches a route, the MVC Framework takes the value of the controller variable and looks for the appropriate name. Controller name is an unqualified class name, which means that the MVC Framework doesn’t know what to do if there are two or more classes called ‘HomeController’.

Example:

- Old legacy solution.
- Project RoutesAndUrls.Controllers contains class HomeController
- Project RoutesAndUrls.AdditionalControllers contains class HomeController

Result?

Namespace priority



Server Error in '/' Application.

*Multiple types were found that match the controller named 'Home'. This can happen if the route that services this request ('{controller}/{action}/{id}/{*catchall}') does not specify namespaces to search for a controller that matches the request. If this is the case, register this route by calling an overload of the 'MapRoute' method that takes a 'namespaces' parameter.*

The request for 'Home' has found the following matching controllers:
UrlsAndRoutes.AdditionalControllers.HomeController
UrlsAndRoutes.Controllers.HomeController

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Namespace priority

Solution:

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
new { controller = "Home", action = "Index", id =
UrlParameter.Optional} ,
new[] { "URLsAndRoutes.AdditionalControllers" });
```

If a suitable controller cannot be found in that namespace, then the MVC Framework will default to its regular behavior and look in all of the available namespaces.

Note. The namespaces added to a route are given equal priority. The MVC Framework does not check the first namespace before moving on to the second and so forth.

Namespace priority

If you have to give preference to a single controller in one namespace, but have all other controllers resolved in another namespace, you need to create multiple routes.

```
routes.MapRoute("AddContollerRoute", "Home/{action}/{id}/{*catchall}",
new { controller = "Home", action = "Index", id = UrlParameter.Optional },
new[] { "URLsAndRoutes.AdditionalControllers" });

routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
new { controller = "Home", action = "Index",id = UrlParameter.Optional },
new[] { "URLsAndRoutes.Controllers" });
```

Constraining Routes

- Constraining a Route Using a Regular Expression
- Constraining a Route to a Set of Specific Values
- Constraining a Route Using HTTP Methods
- Constraining a Route using Type and Value Constraints
- Constraining a Route using a Custom Constraints

Constraining a Route Using a RegExp

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
        new { controller = "Home", action = "Index", id = UrlParameter.Optional },
        new { controller = "^H.*" }, new[] { "URLsAndRoutes.Controllers" });
```

Note. Default values are applied before constraints are checked. So, for example, if I request the URL /, the default value for controller, which is Home, is applied. The constraints are then checked, and since the controller value begins with H, the default URL will match the route.

Constraining a Route to a Set of Specific Values

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
        new { controller = "Home", action = "Index", id = UrlParameter.Optional },
        new { controller = "^H.*", action = "^Index$|^About$" },
        new[] { "URLsAndRoutes.Controllers" });
```

This constraint will allow the route to match only URLs where the value of the action segment is Index or About.

Constraints are applied together, so the restrictions imposed on the value of the action variable are combined with those imposed on the controller variable.

Constraining a Route Using HTTP Methods

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
    new { controller = "Home", action = "Index", id = UrlParameter.Optional },
    new { controller = "^H.*", action = "Index|About",
    httpMethod = new HttpMethodConstraint("GET") },
    new[] { "URLsAndRoutes.Controllers" });
```

- It does not matter what name is given to the property, as long as it is assigned to an instance of the ‘HttpMethodConstraint’ class.
- The ability to constrain routes by HTTP method is unrelated to the ability to restrict action methods using attributes such as ‘HttpGet’ and ‘HttpPost’.
- The route constraints are processed earlier in the request pipeline, determine the name of the controller and action required to process a request.
- The action method attributes are used to determine which specific action method will be used to service a request by the controller.

Using Type and Value Constraints

```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
    new { controller = "Home", action = "Index", id = UrlParameter.Optional },
    new { controller = "^H.*", action = "Index|About", httpMethod = new
    HttpMethodConstraint("GET"), id = new RangeRouteConstraint(10, 20)},
    new[] { "URLsAndRoutes.Controllers" });
```

- The MVC Framework contains a number of built-in constraints that can be used to restrict the URLs that a route matches based on the type and value of segment variables.

Using Type and Value Constraints

- AlphaRouteConstraint()

  Matches alphabet characters, irrespective of case (A–Z, a–z)
- BoolRouteConstraint()

  Matches a value that can be parsed into a bool
- DateTimeRouteConstraint()

  Matches a value that can be parsed into a DateTime
- DecimalRouteConstraint()

  Matches a value that can be parsed into a decimal
- DoubleRouteConstraint()

  Matches a value that can be parsed into a double
- FloatRouteConstraint()

  Matches a value that can be parsed into a float

Using Type and Value Constraints

- IntRouteConstraint()

  Matches a value that can be parsed into an int
- LengthRouteConstraint(len) LengthRouteConstraint(min, max)

  Matches a value with the specified number of characters or that is between min and max characters in length.
- LongRouteConstraint()

  Matches a value that can be parsed into a long
- MaxRouteConstraint(val)

  Matches an int value if the value is less than val
- MaxLengthRouteConstraint(len)

  Matches a string with no more than len characters
- MinRouteConstraint(val)

  Matches an int value if the value is more than val

Using Type and Value Constraints

You can combine different constraints for a single segment variable by using the ‘CompoundRouteConstraint’ class, which accepts an array of constraints as its constructor argument.

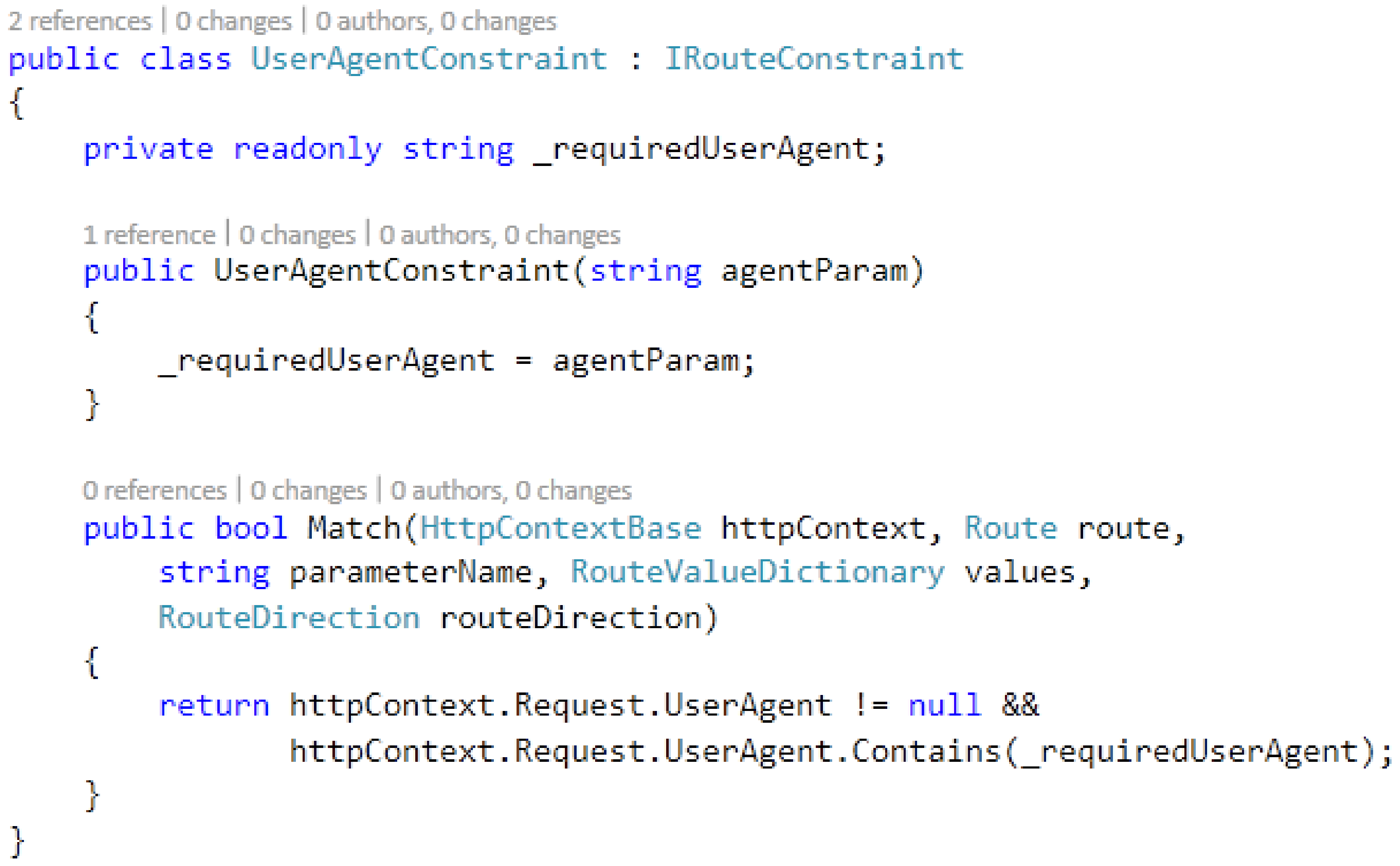
```
routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{*catchall}",
    new { controller = "Home", action = "Index", id = UrlParameter.Optional },
    new { controller = "^H.*", action = "Index|About", httpMethod = new
    HttpMethodConstraint("GET"),
    id = new CompoundRouteConstraint(new IRouteConstraint[]
    { new AlphaRouteConstraint(), new MinLengthRouteConstraint(6) })},
    new[] { "URLsAndRoutes.Controllers" });
```

Custom Constraints

If the standard constraints are not sufficient for your needs, you can define your own custom constraints by implementing the ‘IRouteConstraint’ interface.

The ‘IRouteConstraint’ interface defines the ‘Match’ method, which an implementation can use to indicate to the routing system if its constraint has been satisfied.

Custom Constraints

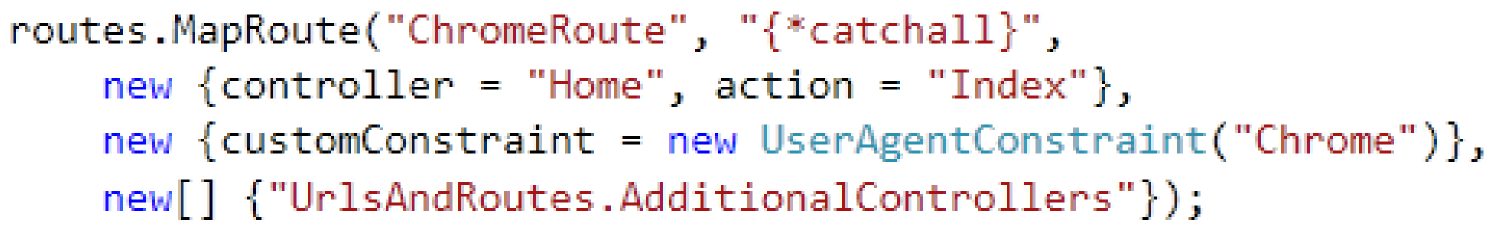


```
2 references | 0 changes | 0 authors, 0 changes
public class UserAgentConstraint : IRouteConstraint
{
    private readonly string _requiredUserAgent;

    1 reference | 0 changes | 0 authors, 0 changes
    public UserAgentConstraint(string agentParam)
    {
        _requiredUserAgent = agentParam;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public bool Match(HttpContextBase httpContext, Route route,
        string parameterName, RouteValueDictionary values,
        RouteDirection routeDirection)
    {
        return httpContext.Request.UserAgent != null &&
                httpContext.Request.UserAgent.Contains(_requiredUserAgent);
    }
}
```

Custom Constraints



```
routes.MapRoute("ChromeRoute", "{*catchall}",
    new {controller = "Home", action = "Index"},
    new {customConstraint = new UserAgentConstraint("Chrome")},
    new[] {"UrlsAndRoutes.AdditionalControllers"});
```

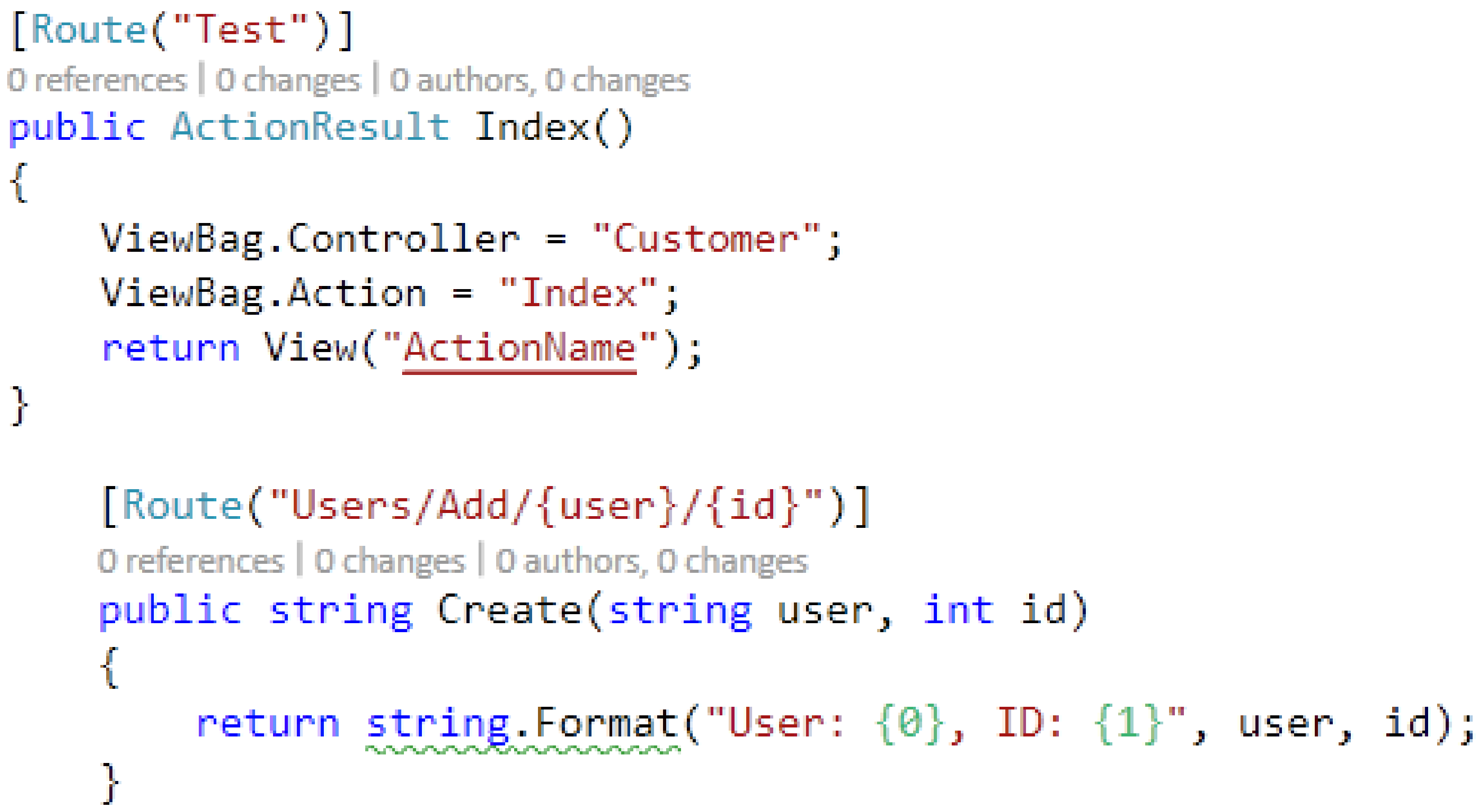
Attribute Routing

Attribute routing is disabled by default and is enabled by the ‘MapMvcAttributeRoutes’ extension method.

```
routes.MapMvcAttributeRoutes();
```

Calling the MapMvcAttributeRoutes method causes the routing system to inspect the controller classes in the application and look for attributes that configure routes.

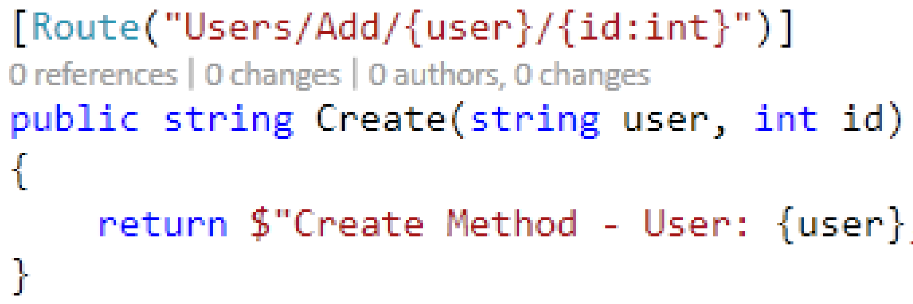
Attribute Routing



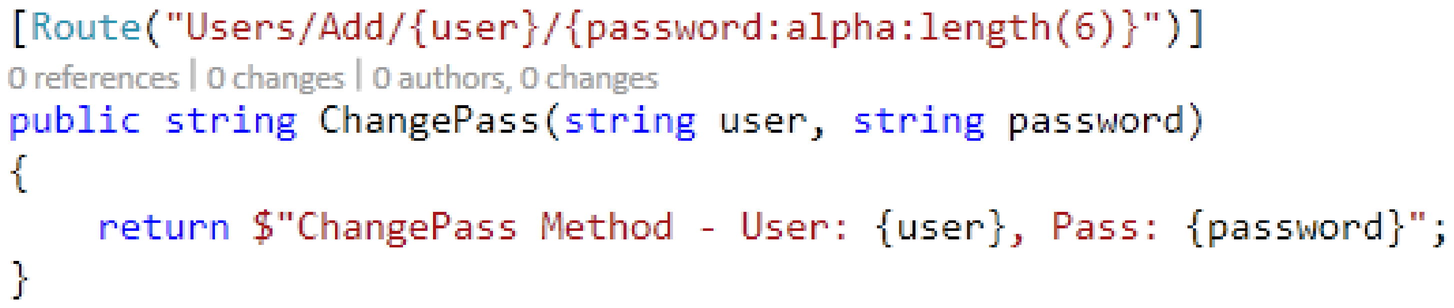
```
[Route("Test")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Index()
{
    ViewBag.Controller = "Customer";
    ViewBag.Action = "Index";
    return View("ActionName");
}

    [Route("Users/Add/{user}/{id}")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Create(string user, int id)
    {
        return string.Format("User: {0}, ID: {1}", user, id);
    }
```

Attribute Routing - Applying Route Constraints

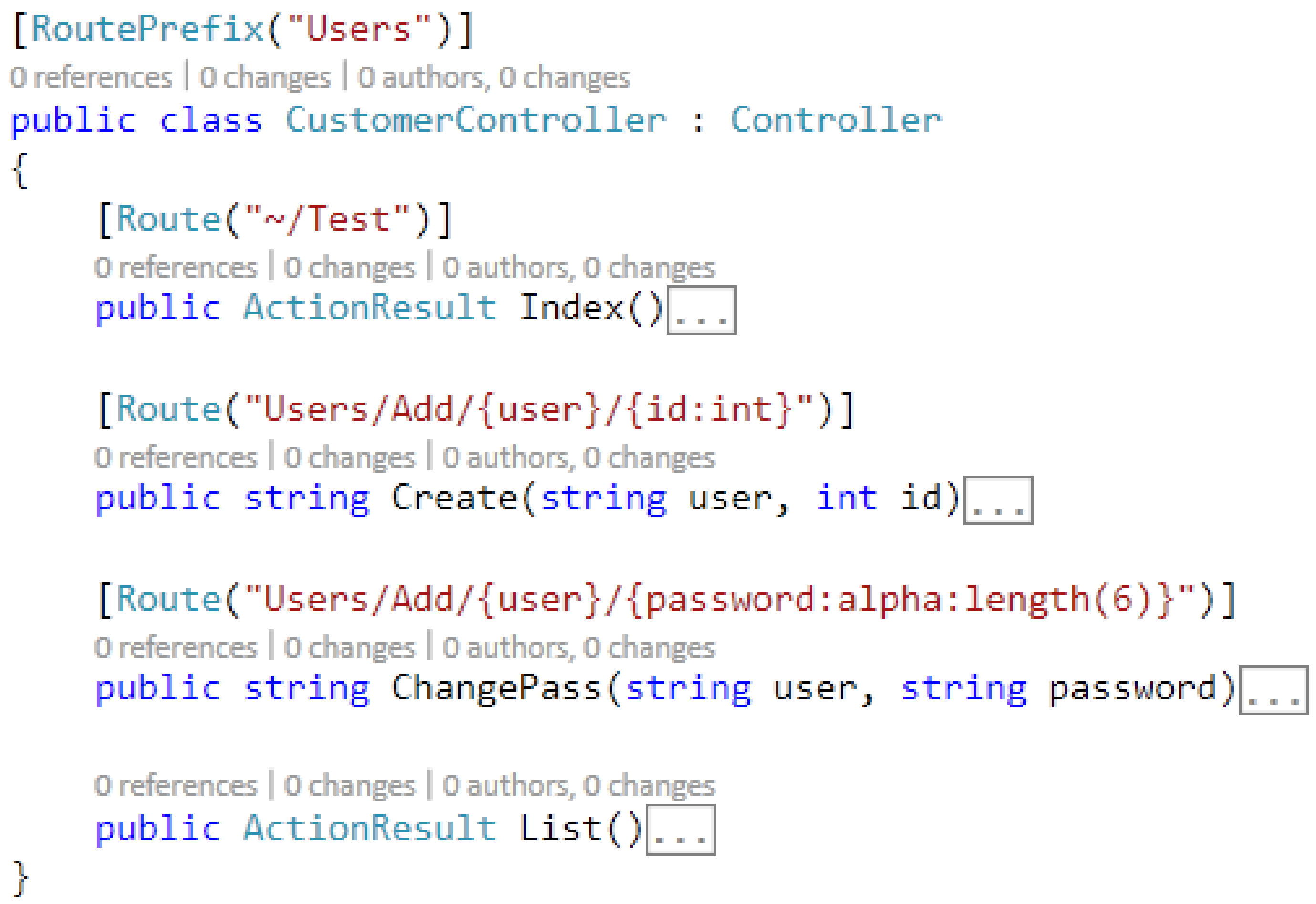


```
[Route("Users/Add/{user}/{id:int}")]
0 references | 0 changes | 0 authors, 0 changes
public string Create(string user, int id)
{
    return $"Create Method - User: {user}, ID: {id}";
}
```



```
[Route("Users/Add/{user}/{password:alpha:length(6)}")]
0 references | 0 changes | 0 authors, 0 changes
public string ChangePass(string user, string password)
{
    return $"ChangePass Method - User: {user}, Pass: {password}";
}
```

Attribute Routing – Applying Prefix



```
[RoutePrefix("Users")]
0 references | 0 changes | 0 authors, 0 changes
public class CustomerController : Controller
{
    [Route("~/Test")]
    0 references | 0 changes | 0 authors, 0 changes
    public ActionResult Index()...

    [Route("Users/Add/{user}/{id:int}")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Create(string user, int id)...

    [Route("Users/Add/{user}/{password:alpha:length(6)}")]
    0 references | 0 changes | 0 authors, 0 changes
    public string ChangePass(string user, string password)...

    0 references | 0 changes | 0 authors, 0 changes
    public ActionResult List()...
}
```

Homework

1. Create a simple ASP.NET MVC project.
2. Implement all of routing approaches defined below:
    1. Default Routes
    2. Static, Custom, Optional segments
    3. Namespace priority
        a. Add library with Home controller and Index action.
        b. Index action should return JSON result.
    4. Route constraints
        a. CompoundRouteConstraint should be presented.
        b. Custom Constraint is optional.
    5. Attribute routing is optional.
3. Add unit test to verify your routing. You should use Machine.Specifications.