Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311

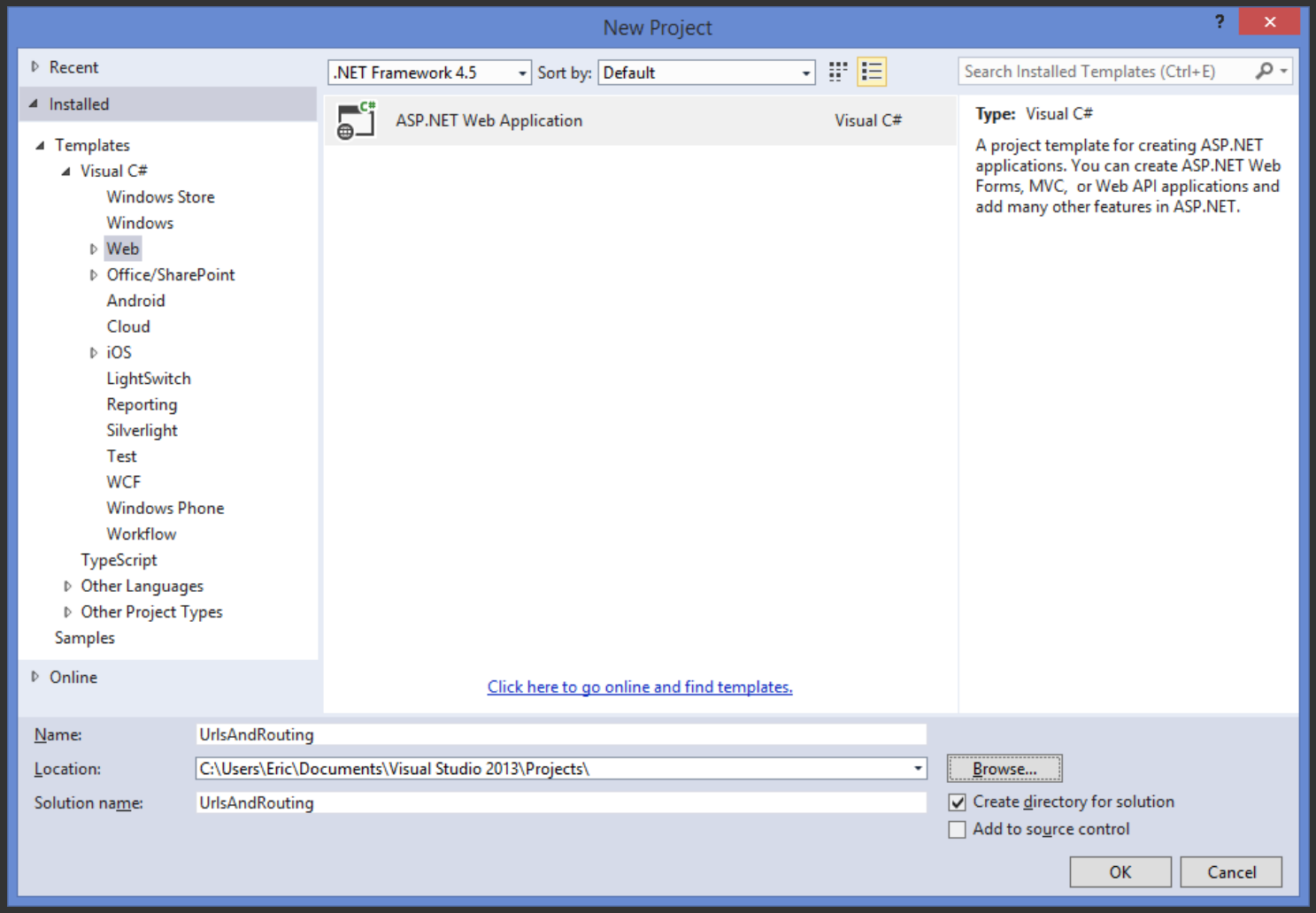
URL Routing in MVC

Software Craftsmanship Guild

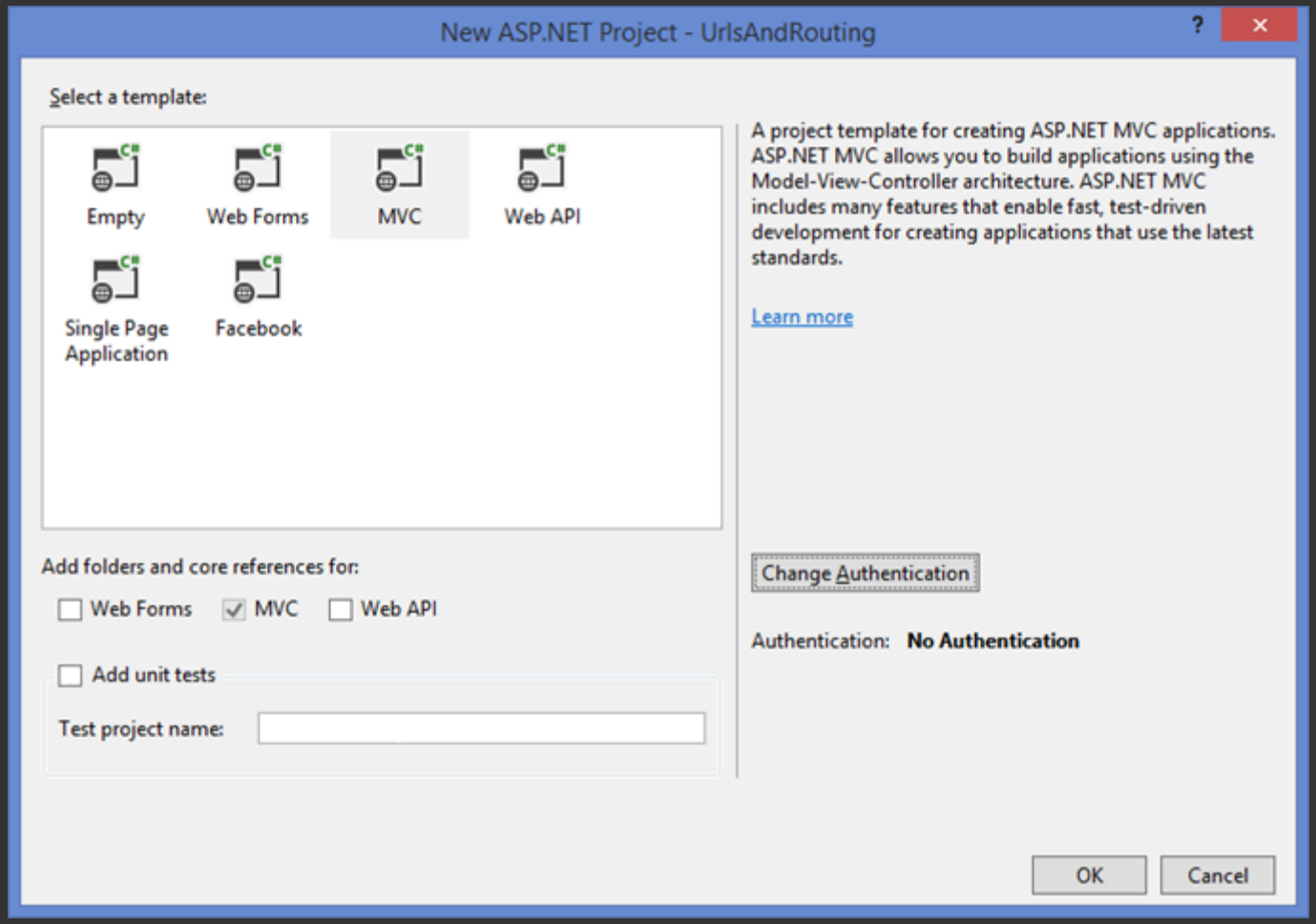
Lesson Goals

Learn how to set up and use the ASP.NET routing system to create flexible URL handling for your projects.

First, Create the Project



Second, Basic Template



Delete the existing controllers

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Controller = "Home";
        ViewBag.Action = "Index";
        return View("ActionName");
    }
}
```

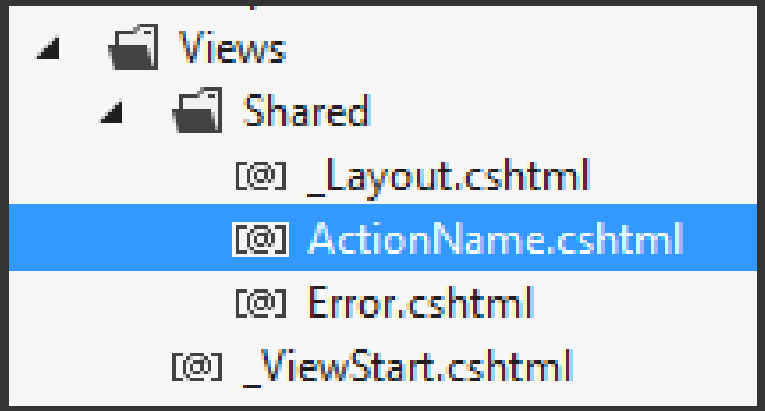
```
public class AdminController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Controller = "Admin";
        ViewBag.Action = "Index";
        return View("ActionName");
    }

}
```

```
public class CustomerController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Controller = "Customer";
        ViewBag.Action = "Index";
        return View("ActionName");
    }

    public ActionResult List()
    {
        ViewBag.Controller = "Customer";
        ViewBag.Action = "List";
        return View("ActionName");
    }
}
```

Add the View

For demonstration purposes, all of the actions go to the same view, and we're using the ViewBag dynamic object for displaying the output on the view.

Since all of the controllers use the same view, we'll create it in /shared



View cshtml

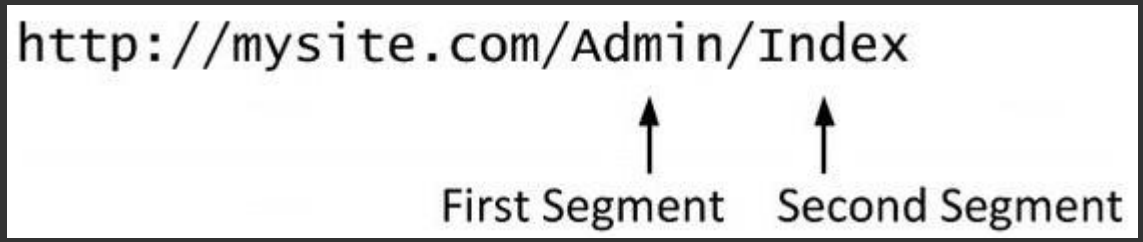
Let’s print out the controller and action name from the ViewBag

```
@{
    ViewBag.Title = "ActionName";
}

<h2>Action Name View</h2>

<p>The controller is: @ViewBag.Controller</p>
<p>The action is: @ViewBag.Action</p>
```

Url Patterns in MVC

The routing system in MVC allows us to compose a set of pattern matches to take browser URLs and map them to controller actions.

The routing system breaks URLs into *segments*.



The routing system when it receives a request first breaks the URL into all of its segments.

Remember RouteConfig?

Here is the starter code for a new MVC site in the RouteConfig.cs file:

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

Notice how the url specifies 3 segments. Controller is the first segment, Action is second, and id is the optional third segment

Matching URLs

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/banana/apples	Controller=banana Action=apples
http://mysite.com/admin	Controller=admin Action=index (index is the default in routeconfig)
http://mysite.com/admin/index/banana	Controller=admin Action=index Id=Banana

Registering a Route

Let's remove the default route and replace it with something more simple.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute("MyRoute", "{controller}/{action}");
}
```

Since we removed the default values, we can no longer take requests to the root URL.

Adding Route Defaults

If a segment currently isn't specified, the web site will error out. We can fix that by providing default values for the route:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

Now if we don't specify an action segment it will default to the Index action of the controller.

Let's Add a Controller Default Too

Now we can handle URLs that have zero, one, or two segments

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

Static URL Segments

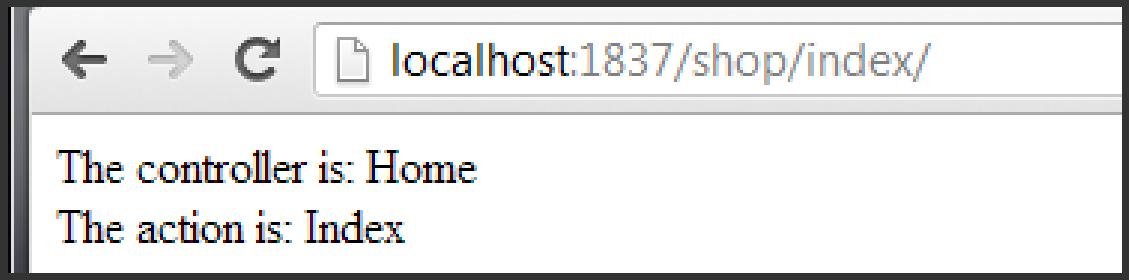
We can mix and match variables and static text in our routes. Static text is useful for serving up routes to external applications and for Search Engine Optimization (SEO).

As an example, imagine we used to have a controller called Shop that our business partners would pull data from and we wanted to remap it to the Home controller:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute("ShopSchema", "Shop/{action}",
        new { controller = "Home" });

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

Result

Now where the shop/index/ URL would have gone to the shop controller, the routing system captures the match and redirects it to the Home controller.



So we can safely remap our published URLs without breaking our customers.

Routes are Matched in Order

We tend to put our more specific routes first and our least specific routes last in the list. This is because the first valid pattern match that is found "wins" and that controller action will be called.

Try these:

```
routes.MapRoute("", "X{controller}/{action}");

routes.MapRoute("ShopSchema", "Shop/{action}",
    new { controller = "Home" });

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

```
routes.MapRoute("ShopSchema", "Shop/{action}",
    new { controller = "Home" });

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

routes.MapRoute("", "X{controller}/{action}");
```

SOFTWARE CRAFTSMANSHIP GUILD

Custom Segments

The {controller} and {action} segments have special meaning in MVC, but we can define our own variables as well. Let's create an id segment:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");


    routes.MapRoute("ShopSchema", "Shop/{action}",
        new { controller = "Home" });

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}",
        new { controller = "Home", action = "Index", id = "0"});
}
```

Add a New Action to HomeController

```
public ActionResult CustomVariable()
{
    ViewBag.Controller = "Home";
    ViewBag.Action = "CustomVariable";
    ViewBag.CustomVariable = RouteData.Values["id"];
    return View();
}
```

And a new view

```
<div>The controller is: @ViewBag.Controller</div>
<div>The action is: @ViewBag.Action</div>
<div>The custom variable is: @ViewBag.CustomVariable</div>
```

Custom Variables as Action Parameters

This is a neat feature of MVC:

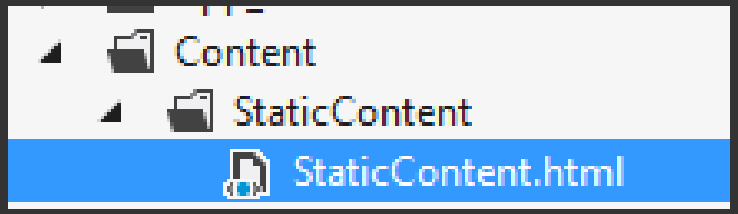
```
public ActionResult CustomVariable(string id)
{
    ViewBag.Controller = "Home";
    ViewBag.Action = "CustomVariable";
    ViewBag.CustomVariable = id;
    return View();
}
```

We can also make segments optional

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute("ShopSchema", "Shop/{action}",
        new { controller = "Home" });

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

Serving up Static Content

Static content like HTML files, pdf downloads, etc can be served up as normal URLs. MVC evaluates whether a file exists before going to the routing engine.




```
<!DOCTYPE html>
<html >
    <head><title>Static HTML Content</title></head>
    <body>
        This is the static html file (~/Content/StaticContent.html)
    </body>
</html>
```

URL Best Practices

- Design URLs to describe their content, not the implementation details of your application. Use /Articles/AnnualReport rather than /Website_v2/CachedContentServer/FromCache/AnnualReport.

- Prefer content titles over ID numbers. Use /Articles/AnnualReport rather than /Articles/2392. If you must use an ID number (to distinguish items with identical titles, or to avoid the extra database query needed to find an item by its title), then use both (/Articles/2392/AnnualReport). It takes longer to type, but it makes more sense to a human and improves search-engine rankings. Your application can just ignore the title and display the item matching that ID.

- Do not use file name extensions for HTML pages (for example, .aspx or .mvc), but do use them for specialized file types (such as .jpg, .pdf, and .zip). Web browsers do not care about file name extensions if you set the MIME type appropriately, but humans still expect PDF files to end with .pdf.

URL Best Practices

- Create a sense of hierarchy (for example, /Products/Menswear/Shirts/Red), so your visitor can guess the parent category's URL.
- Be case-insensitive (someone might want to type in the URL from a printed page). The ASP.NET routing system is case-insensitive by default.
- Avoid symbols, codes, and character sequences. If you want a word separator, use a dash (as in /my-great-article). Underscores are unfriendly, and URL-encoded spaces are bizarre (/my+great+article) or disgusting (/my%20great%20article).
- Do not change URLs. Broken links equal lost business. When you do change URLs, continue to support the old URL schema for as long as possible via permanent (301) redirections.
- Be consistent. Adopt one URL format across your entire application.

Gut Check

- What would be the best way to access a URL parameter in a controller action?
  - /users/100/common-friends/200

```
routes.MapRoute(
    name: "FriendsInCommon",
    url: "users/{userId}/common-friends/{friendId}",
    defaults: new { controller = "Users", action = "FriendsInCommon" }
);
```

```
public ActionResult FriendsInCommon(int userId, int friendId)
{
    // Setup data
    return View();
}
```

Conclusion

The MVC routing system's flexibility is pretty much limited only by your imagination!

We can customize routes for SEO, dynamically repoint deprecated routes to new routes, and serve up static content with little issue.