# Asp

Asp uses MVC as a main component of it. The views folder has the UI elements.

The controller folder has the routing elements. How it works is that; inside the controller folder there are ‘pages’ like homecontroller. They store libraries of functions that basically link the HTTP requests the user puts in the browser to the particular views.

It does this via an ‘action result’. The action result is quite clever, it will assume the page is the name of the function if it’s not specified.

To show the page you say:  
return View(“pagename”).



This describes how the controllers work. Home is a controller. The most general one is described last. Eg

{controller}/{action}.

Because MVC is essentially a framework it comes with a routeconfig class in app\_start folder. It has in there the code that makes the routing work.

Configuring the routing engine is done by copy and pasting the default and modyfing it:

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

Each route maps a ‘pattern’ or family of webpages as per my understanding so far.

The URL is quite nice because we can insert our own wildcards like:

routes.MapRoute(

name: "Serial Number",

url: "serial/{letterCase}",

defaults: new { controller = "Home", action = "Serial", letterCase="upper"}

Thus, it takes ‘serial/lettercase’. Where lettercase can be upper or lower. By default it’s upper.

The routes are executed sequentially. So move them up or in the order you want things to be accessed. If you notice, the first route specified defaults that actually work. So even though you type in webpage/serial/uppercase. It will default to index?

So we finished the first exercise getting routing functional.

The big hurdle was the routing order. Any time a website is causing a page not found. It’s because of a routing error.

**Action types:**

The ActionResult function we keep calling can return a variety of things.

It can return preset error messages, it can return ‘content’ which is just text. It can return full or partial webpages, and it can return JSON, it can ‘redirect to action or redirect to route.

Eg.

return Json(new { name = "serial", value = serial }, JsonRequestBehavior.AllowGet);

This constructs a new json object, specifying the internals name, and value. It also has the argument jsonrequestbehaviour.allowget. (This seems to be a security thing. Json by default doesn’t allow get?

**Action Selector:**

Short and long term memory inside of those actionResults.

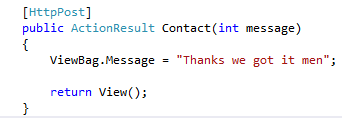
You can use ViewBag and functions of viewbag eg. ViewBag.Message = “bla” to store strings temporarily. These don’t survive redirects.

You can use TempData[“bla”]=”bla”. To survive redirects.

I think how this is to be used is like a ‘field kit’ for the view. The view you call (direct webpage) can reach into the bag and pull out anything you packed for it.

There’s also a [httpPost] you can put above functions in the controller. What this does is it indicates that the function will be executed if a post arrives from that page.

The HttpPost method, takes whatever the user posts and stores it in a variable that is an argument of the function that was called:



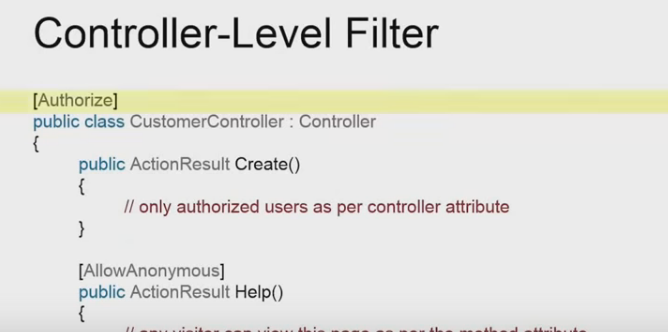
In this case message (which we made an int for lulz and testing) is returned.

# Filters

These are little tags you can add infront of the actionresult functions that have effects.

For example [Authorize] lets you control who can use the function as per the inbuilt authorization system.

This doesn’t have to just be on the action results, it can be on the whole controller class:



This can be reverted with [Allowanonymous] on individual funcitons.

Filters can be also more expansive, executing code before or after a route is taken.

# Razor Syntax

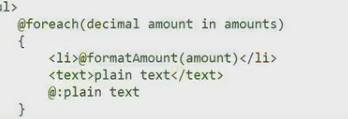
Razor is similar to the blade engine of laravel.

Use @ at any point in the html file to access C#, You can use this to get things like browser details.

For bigger blocks of C you can use @helper functionname.

It’s important to realise you can access csharp anytime inside of html with the @ sign, but you cannot do the reverse. Accessing HTMl inside fo a C# code block/line.

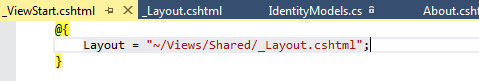
To do that you need to do things like:



Notice how the text element has a @: plaintext?

To comment @\* <code to be commented> \*@

# Layout

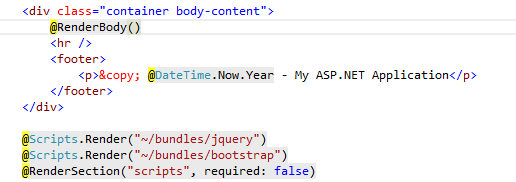


Quick way to apply the layout to all views.

This viewstart can also be copied and pasted into any view subfolder. (it’s currently in shared).

This defined the layout for anything in that folder.

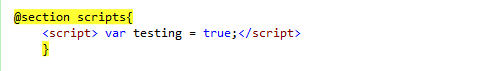
In the actual layout.cshtml file we have:



This @renderbody part is where the actual view data is ‘injected’.

The rendersection allows you to insert parts of the view for instance, the part of the view called ‘scripts’, in if it exists(required:false).

Inside the actual view file:



# HTML helpers

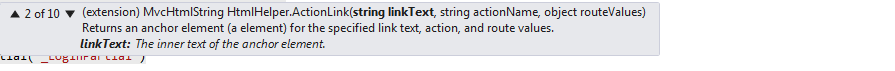
Seems to do something to make code less error prone>?

This seems to be a nice way to do things such as actually put a button in your website.

Done as follows:

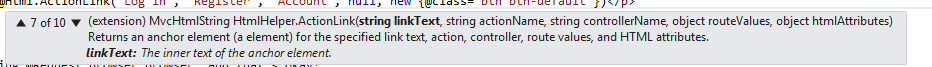


Actionlink has a lot of different overloads:



The above is telling it the link should be called ‘home’, it should call the method “action” inside the controller “Home”.

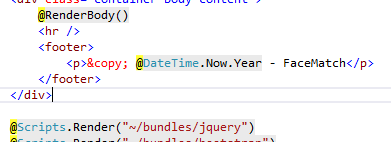
We can also do things like configure what the link looks like through CSS:

Another thing we can do is the “partial” section.



This is a lot like:



Render body.

It renders whatever’s in the arugment as a partial view:

In this case \_LoginPartial which is in the shared folder.

We can set up forms:



Logoff is the action, Account is the ‘controller’. FormMethod.post is the HTTP method.

We can: