Understanding Routes

As well as models, views, and controllers, MVC applications use the ASP.NET routing system, which decides

how URLs map to controllers and actions.

A route is a rule that is used to decide how a request is handled.

When Visual Studio creates the MVC project, it adds some default routes to get you started. You can request

any of the following URLs, and they will be directed to the Index action on the HomeController:

• /

• /Home

• /Home/Index

So, when a browser requests *http://yoursite/* or *http://yoursite/Home*, it gets back the output

from HomeController’s Index method. You can try this yourself by changing the URL in the browser. At the

moment, it will be http://localhost:57628/, except that the port part may be different. If you append /

Home or /Home/Index to the URL, you will see the same Hello World result from the MVC application. ok

This is a good example of benefiting from following conventions implemented by ASP.NET Core MVC.

In this case, the convention is that I will have a controller called HomeController and it will be the starting

point for the MVC application. The default configuration that Visual Studio creates for a new project assumes

I will follow this convention. Since I did follow the convention, I automatically got support for the URLs in

the preceding list. If I had not followed the convention, I would need to modify the configuration to point to

whatever controller I had created instead. For this simple example, the default configuration is all I need. Ok

* Rendering Web Pages

The output from the previous example wasn’t HTML—it was just the string Hello World.

To produce an HTML response to a browser request, I need a view, which tells MVC how to generate a response to a request

from a browser.

using Microsoft.AspNetCore.Mvc;

namespace PartyInvites.Controllers {

public class HomeController : Controller {

public ViewResult Index() { //Render a View

return View("MyView"); // Name of the view “MyView

}

}

}

When I return a ViewResult object from an action method, I am instructing MVC to render a view. I

create the ViewResult object by calling the View method, specifying the name of the view that I want to use,

which is MyView.

* Creating View

To create the view needed for this example,

1. expand the Views folder in the Solution Explorer, right-click

the Home folder.

1. Select Add ➤ New Item
2. Drill down to the ASP.NET Core ➤ Web ➤ ASP.NET category using the left pane
3. Select the *MVC View Page*
4. *MyView.cshtml* and click the Add button to create the view ok

Copy and paste

This is an expression that will be interpreted by the Razor view engine, which processes the contents of

views and generates HTML that is sent to the browser.

This is a simple Razor expression, and it tells Razor that I chose not to use a layout, which is like a template for the HTML that will be sent to the browser

I am going to ignore Razor for the moment and come back to it later.

@{  
    Layout = null;  
}

Html normal  
<!DOCTYPE html>  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <div>  
        Hello World (from the view)  
    </div>  
</body>  
</html>

When I first edited the Index action method, it returned a string value.

This meant that MVC did nothing except pass the string value as is to the browser. Now, that the Index method returns a ViewResult, MVC renders a view and returns the HTML it produces. I told MVC which view should be used, so it used

the naming convention to find it automatically. The convention is that the view has the name of the action

method and is contained in a folder named after the controller: /Views/Home/MyView.cshtml.

https://localhost:5001/home/index/MyView.cshtml

I can return other results from action methods besides strings and ViewResult objects.

For example, if I return a RedirectResult, the browser will be redirected to another URL. If I return an

*HttpUnauthorizedResult*, I can prompt the user to log in. These objects are collectively known as action

results. The action result system lets you encapsulate and reuse common responses in actions.