WEB PROGRAMMING ASP.NET MVC CORE

CONTROLLERS

Incoming requests are handled by controllers . In ASP.NET Core MVC, controllers are just C# classes (usually inheriting from the `Microsoft.AspNetCore.Mvc.Controller` class, which is the built-in MVC controller base class).

Controller classes handle communication from the user, overall application flow, and application-specific logic. Every public method in a controller is callable as an HTTP endpoint. The methods within a controller are called controller actions. Their job is to respond to URL requests perform the appropriate actions and return a response back to the browser or user that invoked the URL.

The MVC convention is to put controllers in the Controllers folder, which Visual Studio created when it set up the project.

CONVENTION OVER CONFIGURATION

"We know, by now, how to build a web application. Let's roll that experience into the framework so we don't have to configure absolutely everything again."

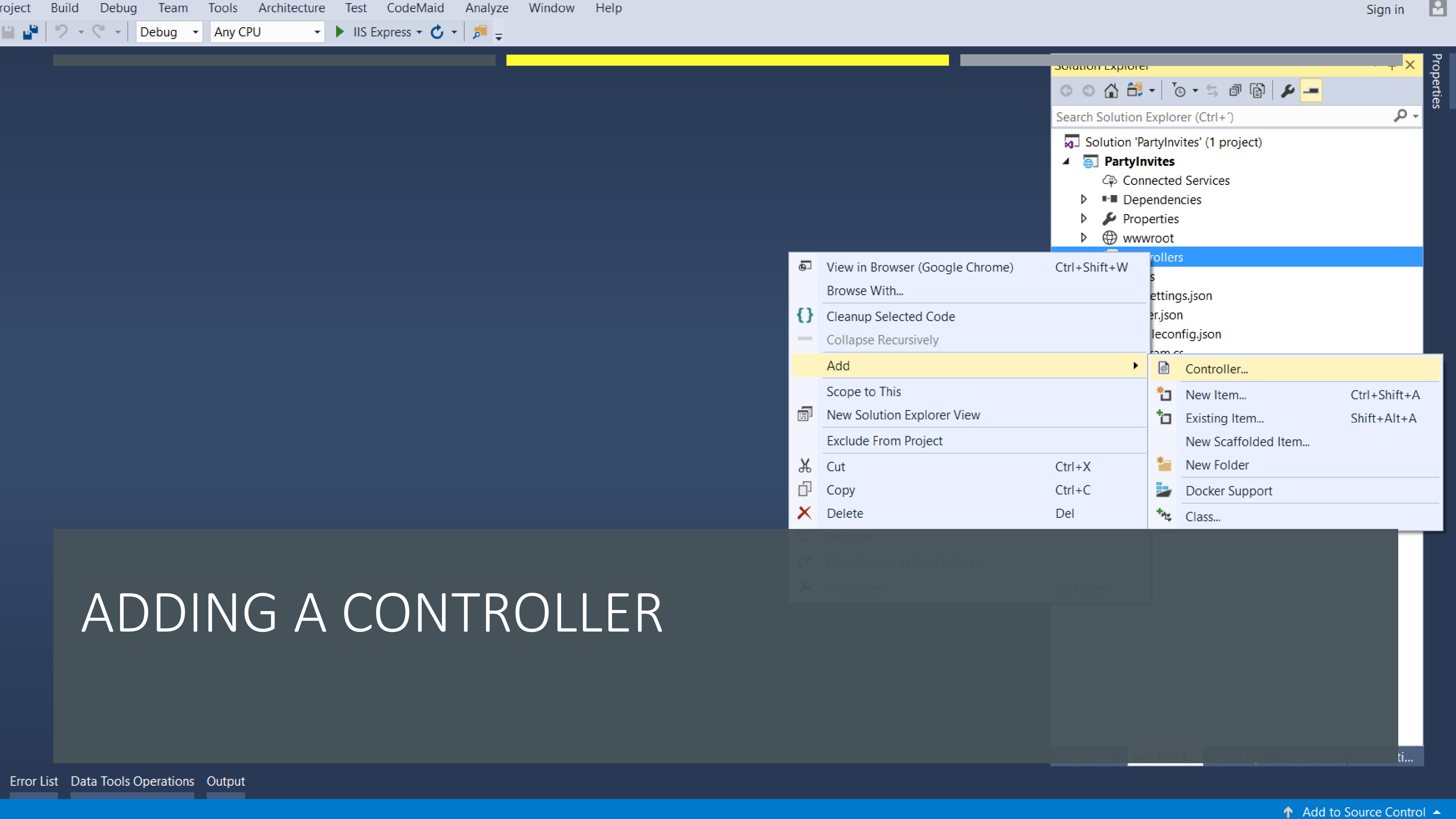
Each controller's class name ends with Controller (e.g. `ProductController`, `HomeController`) in the Controllers directory.

Views that controllers use live in a subdirectory of the Views main directory and are named according to the controller name (e.g. the view for the `ProductController` is `/Views/Product`)

CONTROLLERS

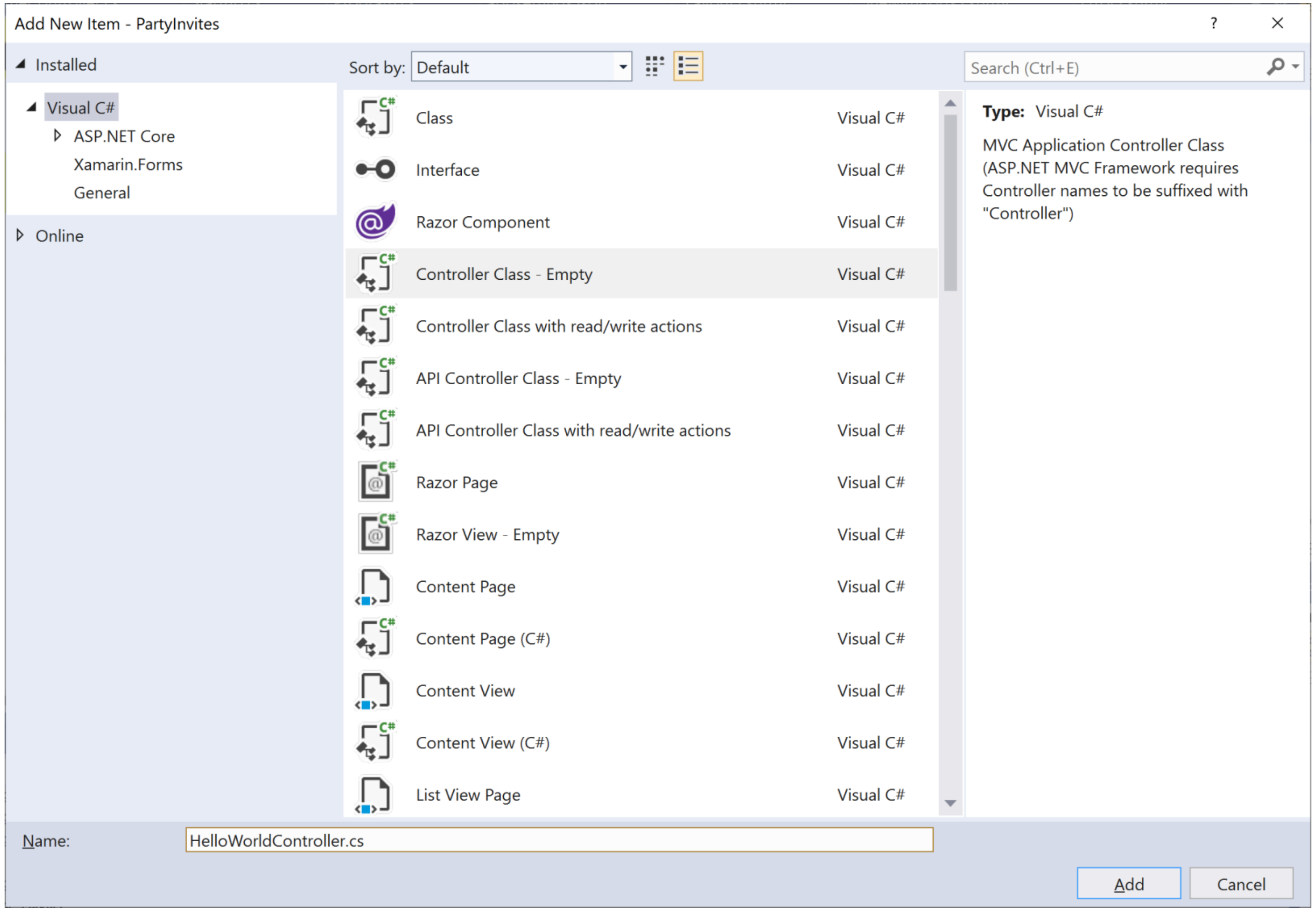
```
public class HomeController : Controller {
    public IActionResult Index() {
        return View();
    }

    public IActionResult Contact() {
        ViewData["Message"] = "Contacts ...";
        return View();
    }
}
```



ADDING A CONTROLLER

ADDING A CONTROLLER



Class	Visual C#
Interface	Visual C#
Razor Component	Visual C#
Controller Class - Empty	Visual C#
Controller Class with read/write actions	Visual C#
API Controller Class - Empty	Visual C#
API Controller Class with read/write actions	Visual C#
Razor Page	Visual C#
Razor View - Empty	Visual C#
Content Page	Visual C#
Content Page (C#)	Visual C#
Content View	Visual C#
Content View (C#)	Visual C#
List View Page	Visual C#

Type: Visual C#

MVC Application Controller Class (ASP.NET MVC Framework requires Controller names to be suffixed with "Controller")

Name: HelloWorldController.cs

CONTROLLER ACTIONS

```
public class HelloWorldController : Controller {
   //
   // GET: /HelloWorld/
   public string Index() {
       return "This is my default action...";
   }

   //
   // GET: /HelloWorld/Welcome/
   public string Welcome() {
       return "Welcome action method...";
   }
}
```

UNDERSTANDING ROUTES

- 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.
- The format for routing is defined in the `Startup.cs` file.

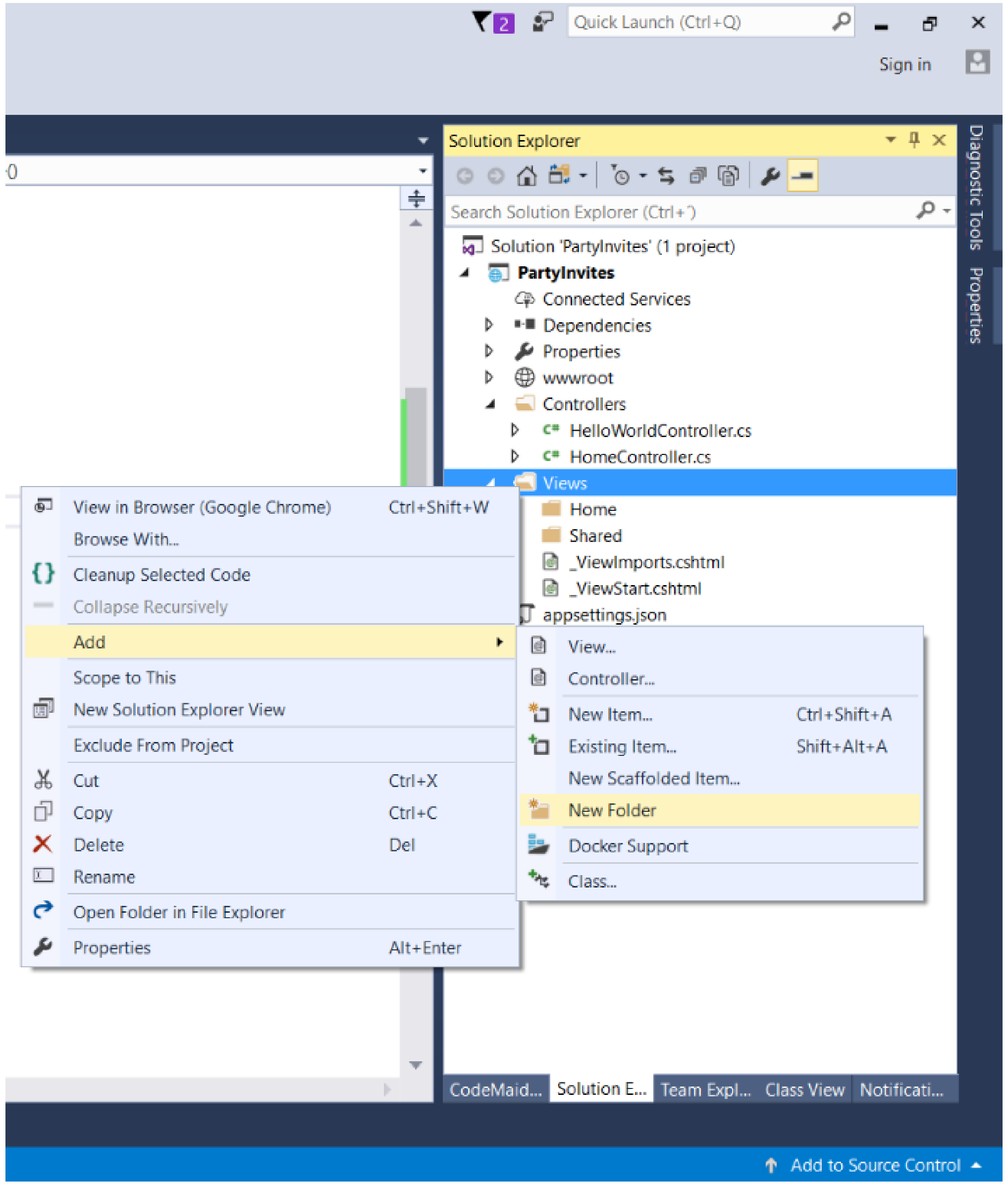
```
app.UseMvc(routes => {
    routes.MapRoute(
        name: "default",
        template:"{controller=Home}/{action=Index}/{id?}");
});
```

VIEWS

- The view is responsible for providing the user interface (UI) to the user. After the controller has executed the appropriate logic for the requested URL, it delegates the display to the view.
- Views are stored in the Views folder, organized into subfolders. Views that are associated with the Home controller, for example, are stored in a folder called Views/Home . Views that are not specific to a single controller are stored in a folder called Views/Shared .
- Visual Studio creates the Home and Shared folders automatically when the Web Application template is used and puts in some placeholder views to get the project started.

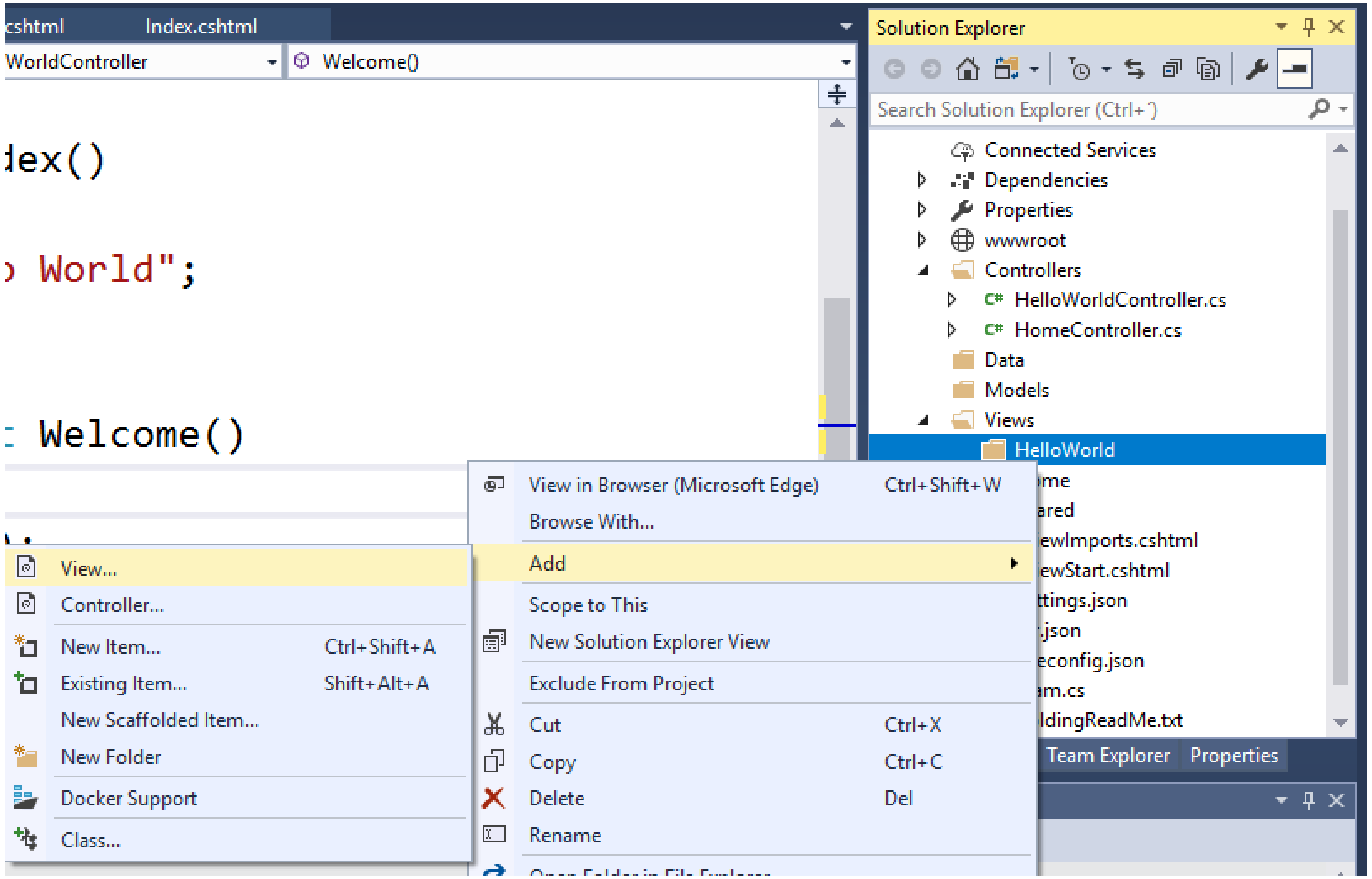
MODIFYING THE CONTROLLER TO RENDER A VIEW

```
public ViewResult Welcome()
{
    return View();
}
```

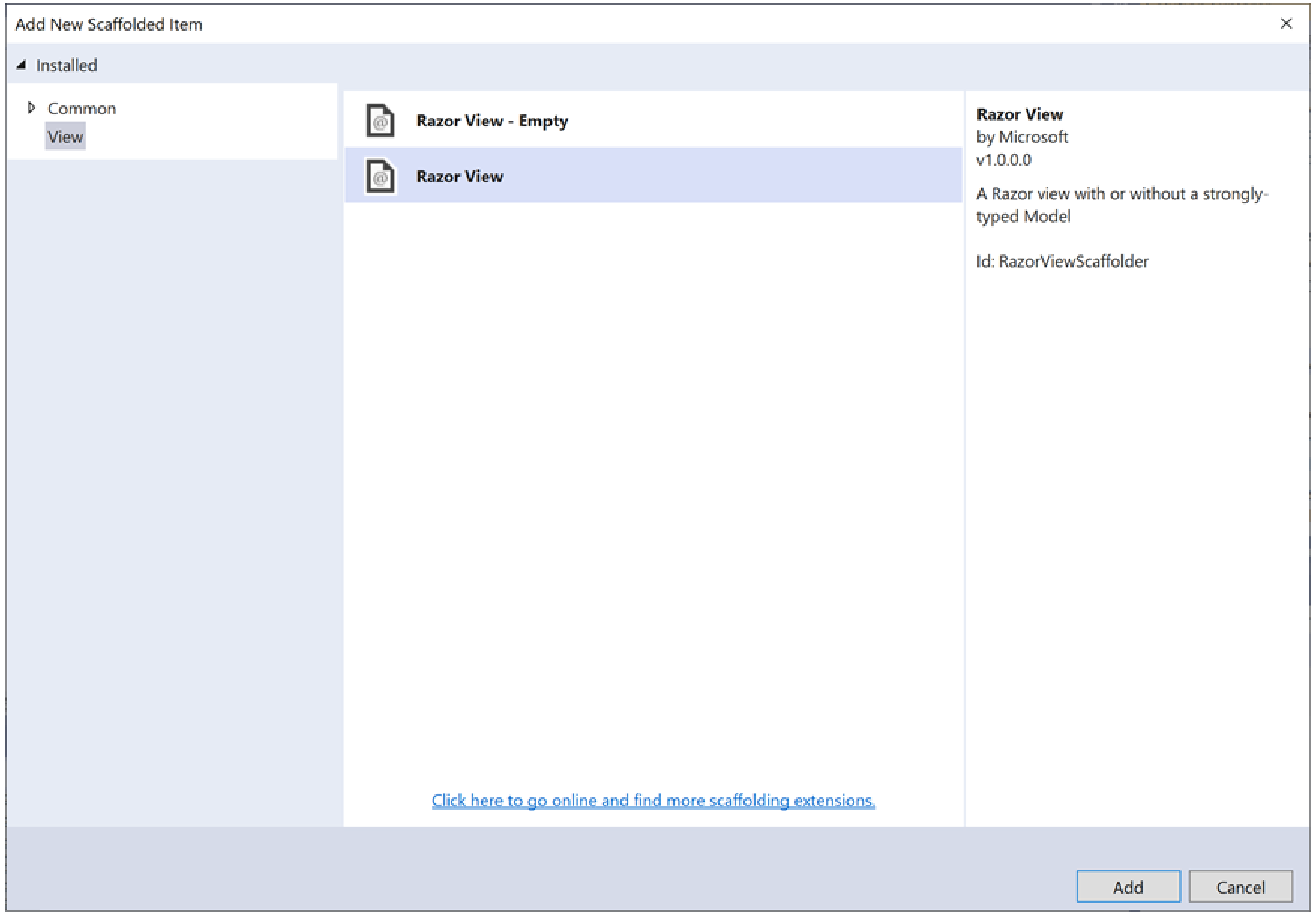


ADDING A VIEW

CREATING A VIEW



```
dex()
 World";
 Welcome()
```


Add New Scaffolded Item

Installed

Common

View

Razor View - Empty

Razor View

Razor View
by Microsoft
v1.0.0.0

A Razor view with or without a strongly-typed Model

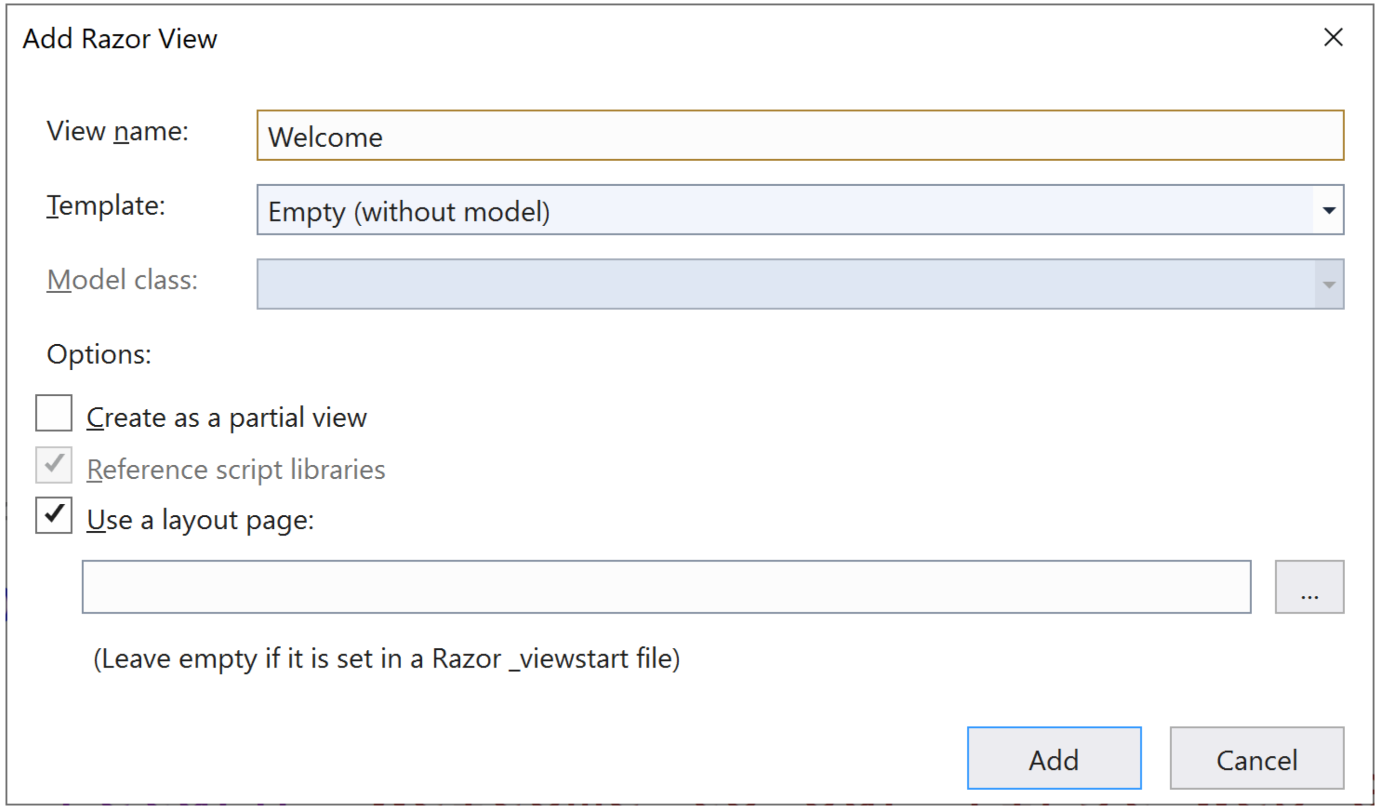
Id: RazorViewScaffolder

Click here to go online and find more scaffolding extensions.

Add Cancel

CREATING A VIEW

CREATING A VIEW


Add Razor View

View name: Welcome

Template: Empty (without model)

Model class:

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

VIEWS

MVC uses the naming convention to find the View automatically. The convention is that the view has the name of the action method and is contained in a folder named after the controller.

```
@{
   ViewData["Title"] = "Welcome";
}

<h2>Welcome</h2>

<p>Hello from Welcome action (Hello World
Controller)</p>
```

ACTION METHODS RETURN TYPES

- Besides strings and `ViewResult` objects action methods can return other results. For example, if the method returns a `RedirectResult` , the browser will be redirected to another URL. If it returns an `HttpUnauthorizedResult`, I force 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.

ADDING DYNAMIC OUTPUT

- The whole point of a web application platform is to construct and display *dynamic* output. In MVC, it is the controller's job to construct some data and pass it to the view, which is responsible for rendering it to HTML.
- One way to pass data from the controller to the view is by using the `ViewBag` object, which is a member of the Controller base class. `ViewBag` is a dynamic object to which you can assign arbitrary properties, making those values available in whatever view is subsequently rendered.

PASSING DATA FROM THE CONTROLLER TO THE VIEW

Data for the View is provided when the `ViewBag.Message` property is assigned a value.

The `Message` property didn't exist until the moment it was assigned a value. This allows to pass data from the controller to the view in a free and fluid manner, without having to define classes ahead of time.

```
public IActionResult Index() {
    int hour = DateTime.Now.Hour;

    string message;

    if (hour >= 7 && hour < 12) {
        message = "Good morning";
    } else if (hour >= 12 && hour < 20) {
        message = "Good afternoon";
    } else {
        message = "Good evening";
    }

    ViewBag.Message = message;

    return View();
}
```

RETRIEVING A VIEWBAG DATA VALUE IN THE VIEW

```
@{
   ViewData["Title"] = "Home";
}

<h2>Home</h2>
<p>@ViewData["Message"].</p>
<p>We are going to have an exciting party.</p>
```