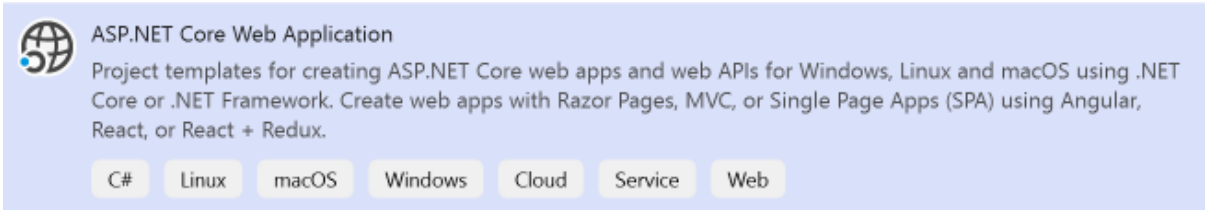
Web Development .NET
Creating an ASP.NET Web App With MVC

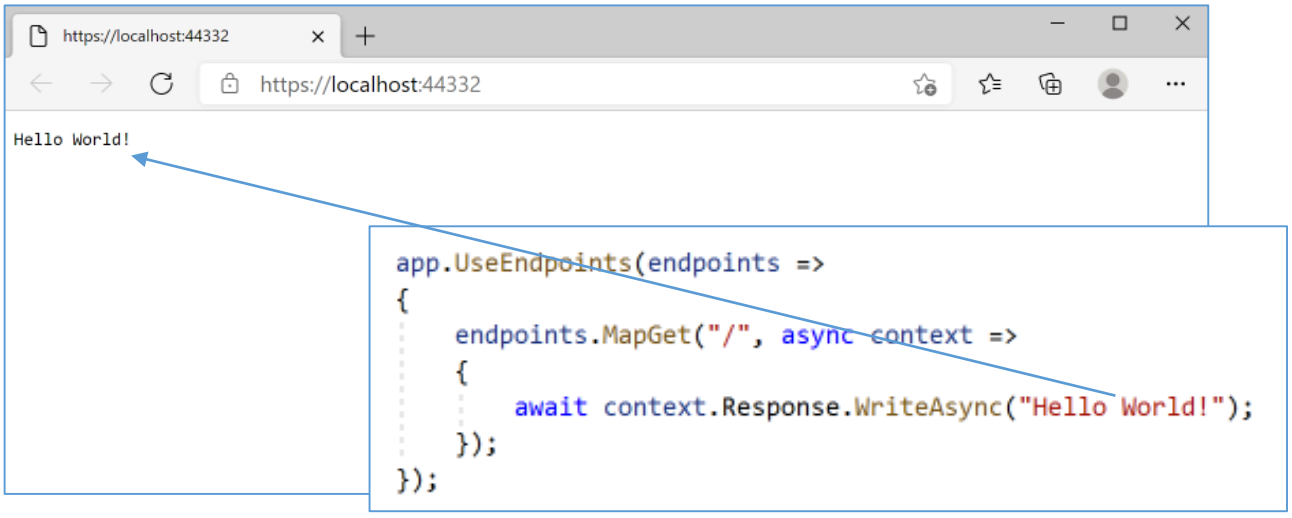
There is a Visual Studio template that allows us to create an ASP.NET Web Application with MVC built in, but we can learn more about how things work if we start off creating an Empty Web App and integrate MCV into it ourselves.

Create a new project.
Choose ASP.NET Core Web Application, enter a name and then choose Empty.
I chose to call this project Dogs as it will be a web application about dogs.



This creates a new project with just a few files.
The important files here are Program.cs and Startup.cs

Run the application.
See the message “Hello World!”
Where does this come from?



This comes from inside the Startup.cs file.
It is actually inside the Configure() method which does a lot of stuff we don’t understand yet.
So right now our Web Application does actually work (we see the “Hello World!” message).

Adding MVC Support

We need to add MVC support to it so that the web server knows how we want it to handle requests. Open Startup.cs and look for the ConfigureServices method. It starts off empty, but we need to add code to include MCV support.

```
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
}
```

Startup.cs

We also want to change our Configure() method.

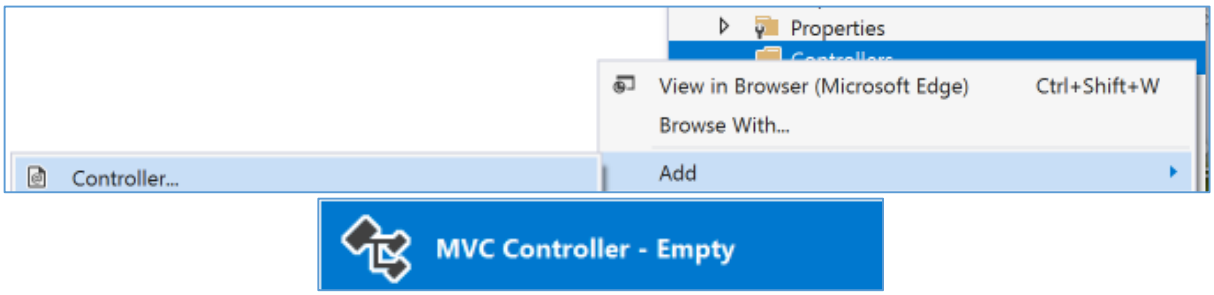
We want to tell it that instead of printing the words “Hello World!”, we want it to map incoming requests to the Controllers. This uses the concept of routes, so we need to add default routing to our application.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();

    });
}
```

Startup.cs

Adding A Controller

We should create folders to store our Models, Views, Controllers, etc. to keep our project organised. So create a Controllers folder from Solution Explorer, then Add a new controller inside this folder. Choose Empty because we want to learn how to do this from scratch.



We will start off creating a Controller for the homepage, so we will call it HomeController If we look at HomeController.cs it is basically a class which inherits the Controller class. This is how the .NET Framework understands that it must be treated as an MVC Controller.

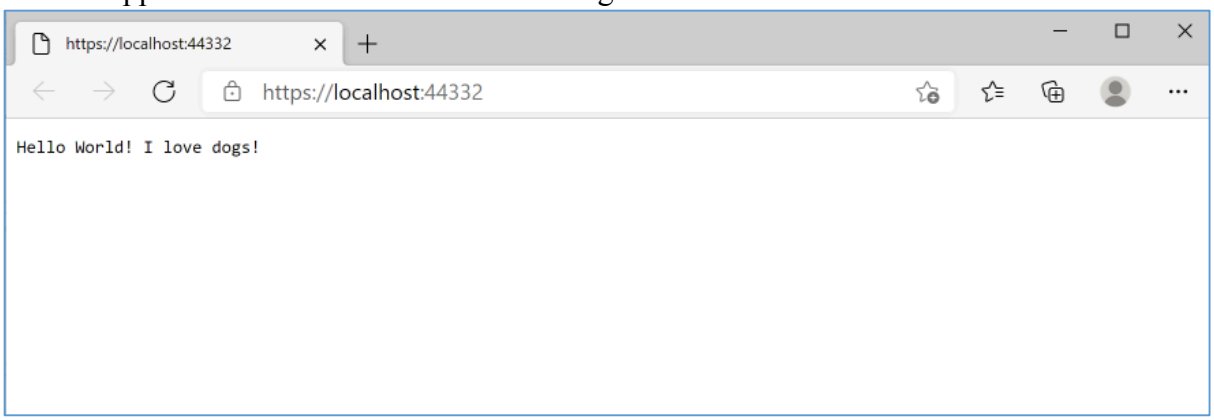
```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

It has one method Index() which returns the default View by calling the View() method.

We will change this to return some content because we have not looked at Views yet.

```
public IActionResult Index()
{
    // Remove this line
    // return View();
    // Add this line
    return Content("Hello World! I love dogs!");
}
```

Run the application and we will see this message.

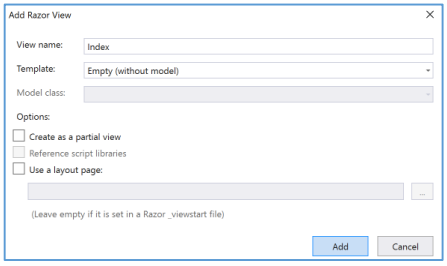


That shows that we created our first Controller (real projects will have many Controllers). Now we are going to add a View so that we have an actual web page being displayed.

Creating A View

We should have our views inside a Views folder, so create a new folder called Views.

It is also recommended to have one folder for each Controller, so inside the new Views folder we should create a Home folder. We call it Home because it will contain the view(s) that will be used by our Home controller.



Add a new Razor View.
Enter the name (Index) and click Add.
Make sure the options are as shown for now.
We are not creating a partial view or using a layout page.

A new View has been created.
It is a standard HTML document with a little bit of MVC-related code at the top.
Modify the HTML by adding a top-level heading and some paragraphs of text.

Index.cshtml

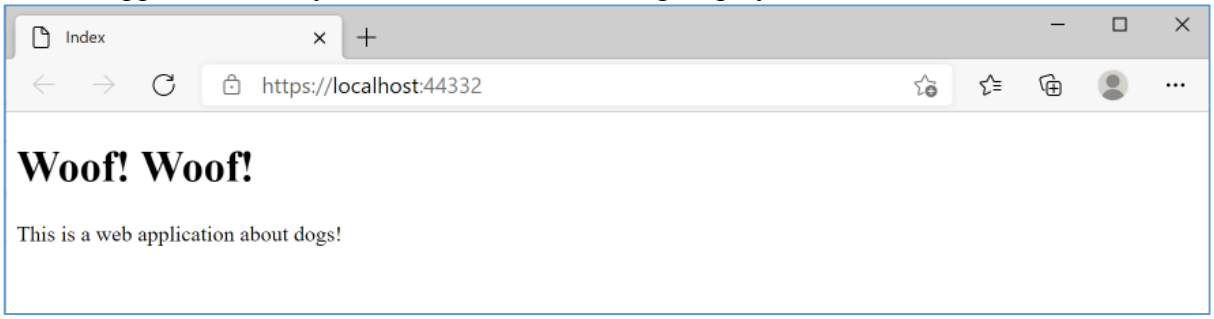
```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <h1>Woof! Woof!</h1>

    <p>This is a web application about dogs!</p>
</body>
</html>
```

Now go to the HomeController.cs and change the Index() method back to what it was initially so that the View is returned (instead of the message we changed it to).

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        // Put this line back
        return View();

    }
}
```

HomeController.cs

Run the application and you will see the View being displayed.



Creating A Model

In the MVC architecture, the Model is generated by the Controller and passed to the View.
A Model can be anything - a simple variable, or a class, or it could come from a database.

Create a folder called Models and add a new Class.
Since this web application is about dogs, I have created a Dog class.

```
public class Dog
{
    public string Name { get; set; }
    public string Breed { get; set; }
    public DateTime DateOfBirth { get; set; }

}
```

Dog.cs

Once our Model is ready, we need to instantiate it by the Controller.
So in HomeController.cs we can create a new Dog object and pass it to the View.

```
public IActionResult Index()
{
    // Create a new dog
    Models.Dog myDog = new Models.Dog()
    {
        Name = "Gypsy",
        Breed = "Golden Retriever",
        DateOfBirth = new DateTime(1999, 3, 24)
    };

    // Pass this dog to the View() method
    return View(myDog);

}
```

HomeController.cs

Once we have done this, we should tell the View that it is being given a Model and that we want it displayed.

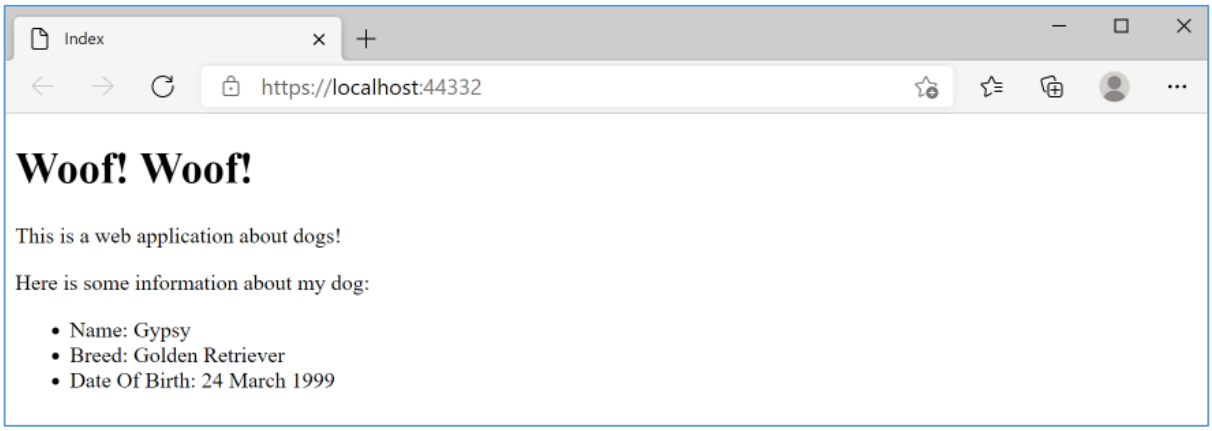
First we use the @model Razor directive (at the top of the View file) so that the View knows that it will be getting a model of the type Dog. This also helps Visual Studio provide IntelliSense for the Dog class.

We can now use Razor syntax to access the Dog object and its properties.

```
@model Dogs.Models.Dog

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <h1>Woof! Woof!</h1>

    <p>This is a web application about dogs!</p>

    <p>Here is some information about my dog:</p>

    <ul>
        <li>Name: @Model.Name</li>
        <li>Breed: @Model.Breed</li>
        <li>Date Of Birth: @Model.DateOfBirth.ToLongDateString()</li>
    </ul>

</body>
</html>
```

Index.cshtml

Run the application and we will see the View displaying the Model (which was passed by the Controller)



Next Steps:

- Understanding Razor syntax which is used to mix client-side markup and server-side code without having to explicitly jump in and out of both syntax types.