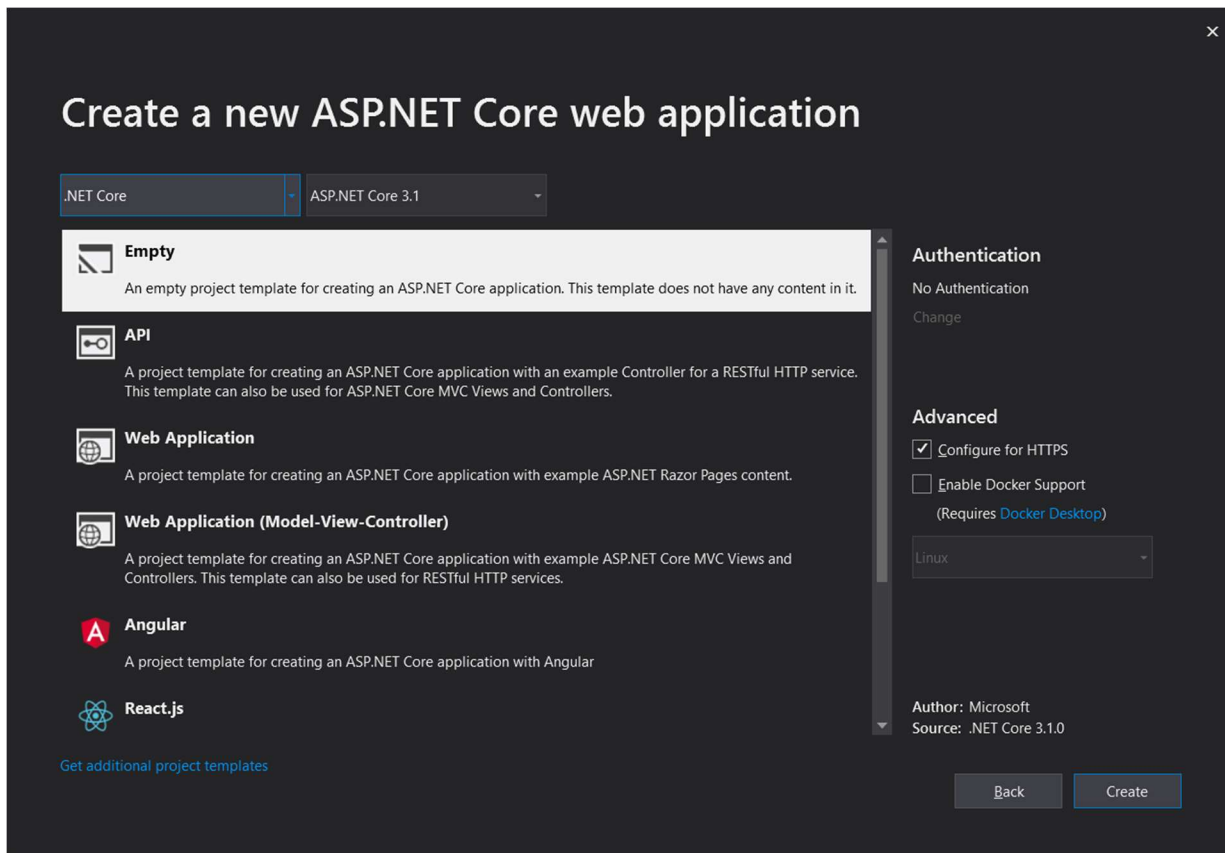


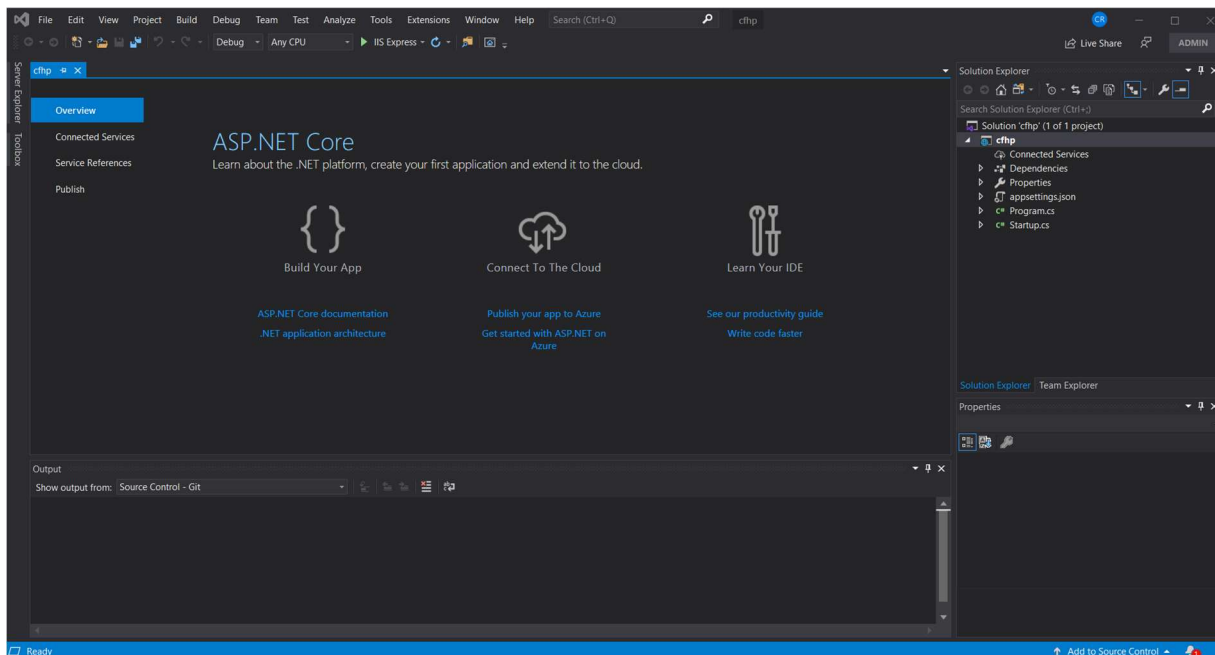
Empty Project

Starting Visual Studio 2019

Created empty project, accepted the default ASP.NET Core 3.1 at the top and the Source: .NET Core 3.1.0

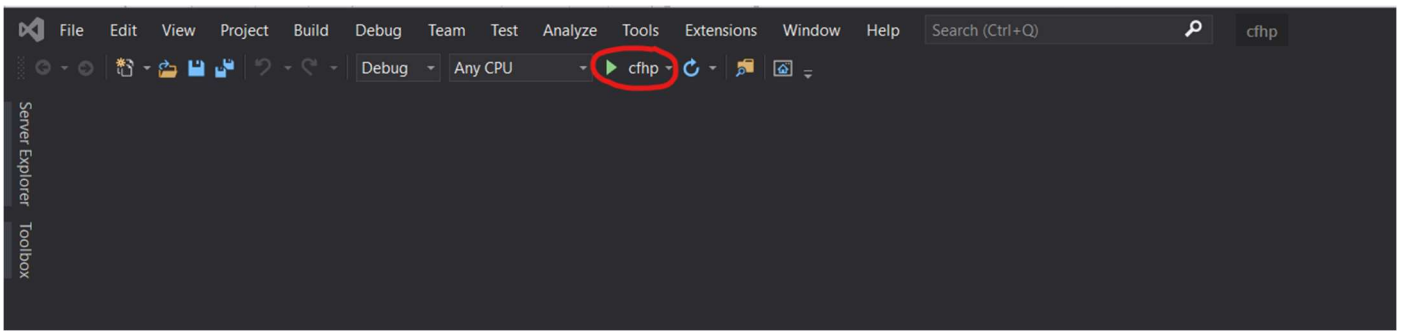


After clicking on Create in the bottom right, the next page shows up

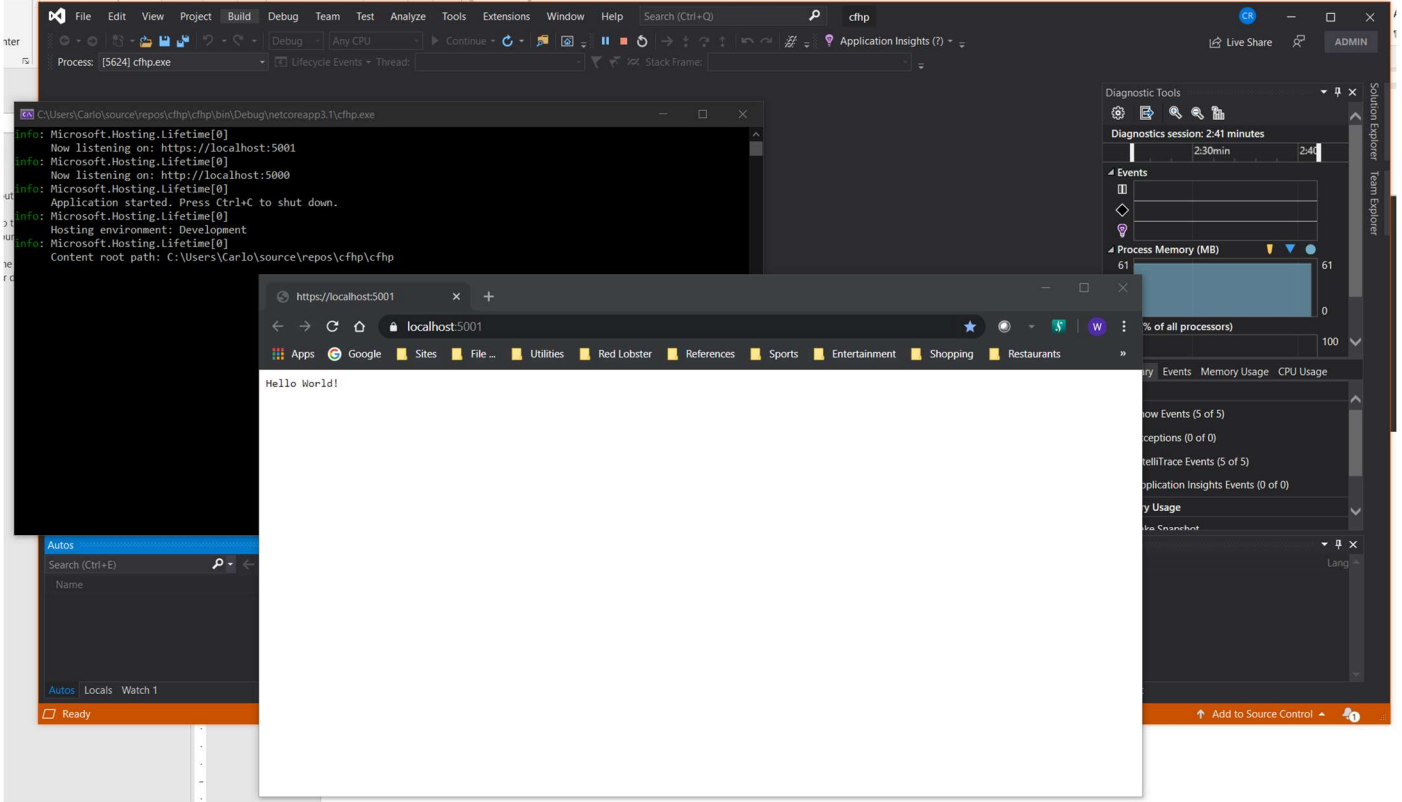


Running the Project

I then changed to run the project from the project mode, not IIS express

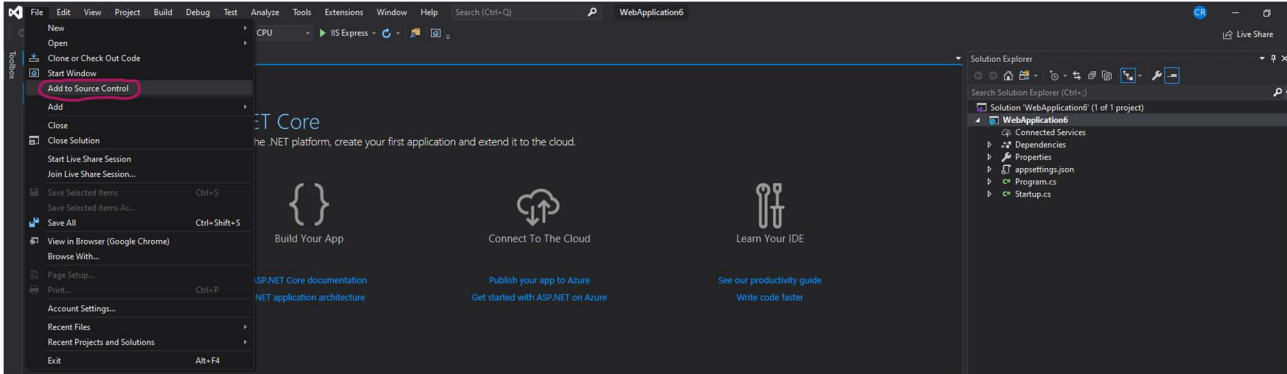


Running it pops up the command windows typical of the ASP.NET Core 2.X.X projects, HOWEVER, it now pops up a web page with Hello World instead of a DOS prompt with Hello World

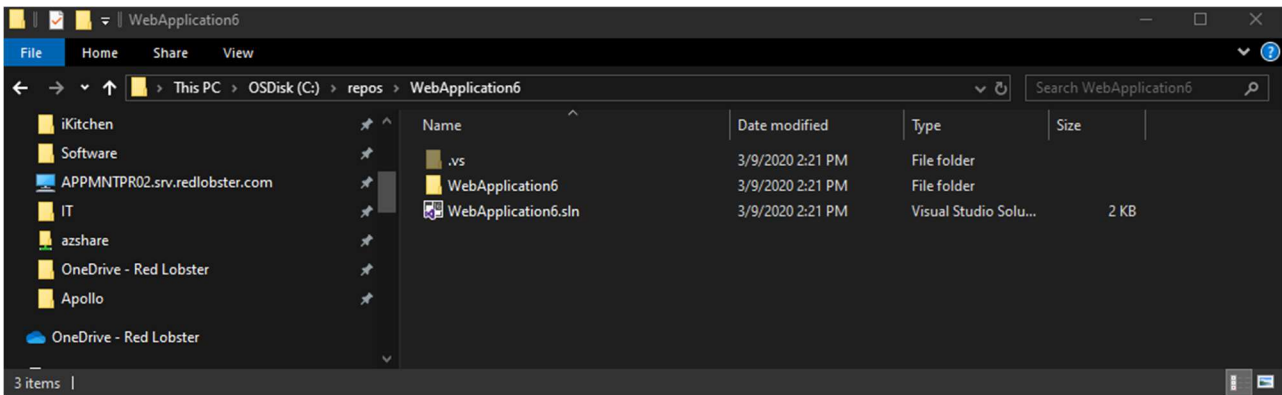


Create Local Git Repository

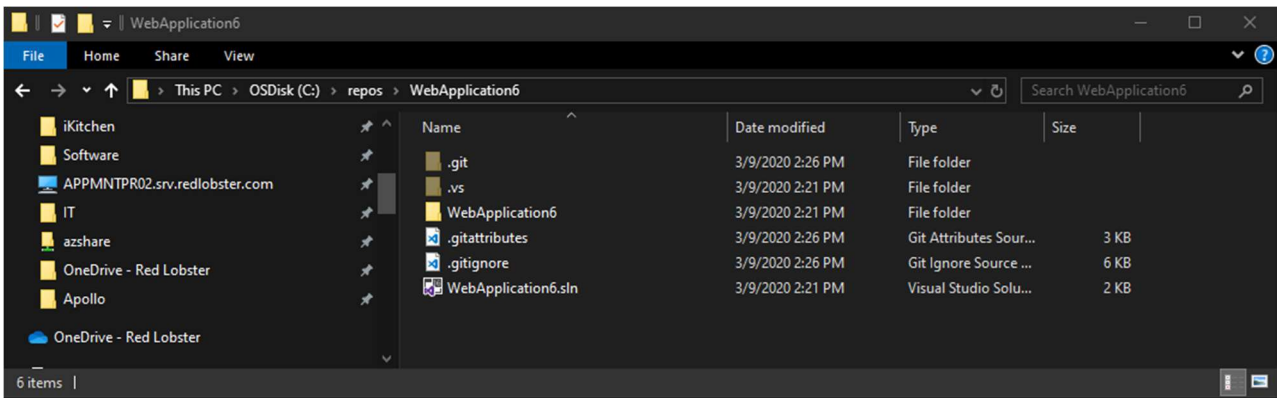
From solution folder right click on the solution and click on File menu to add to “Add to Source Control”. This is better than doing from the git init dos command because this will add the Visual Studio’s gitignore file for you.



Before adding to Git

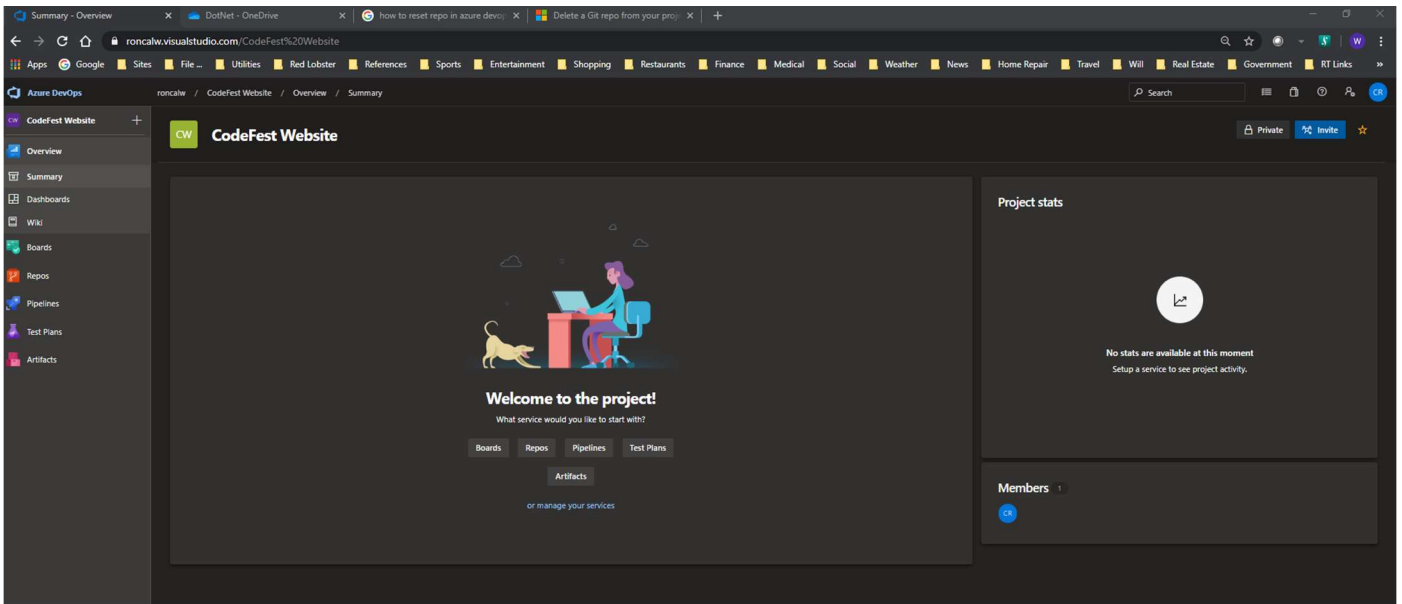
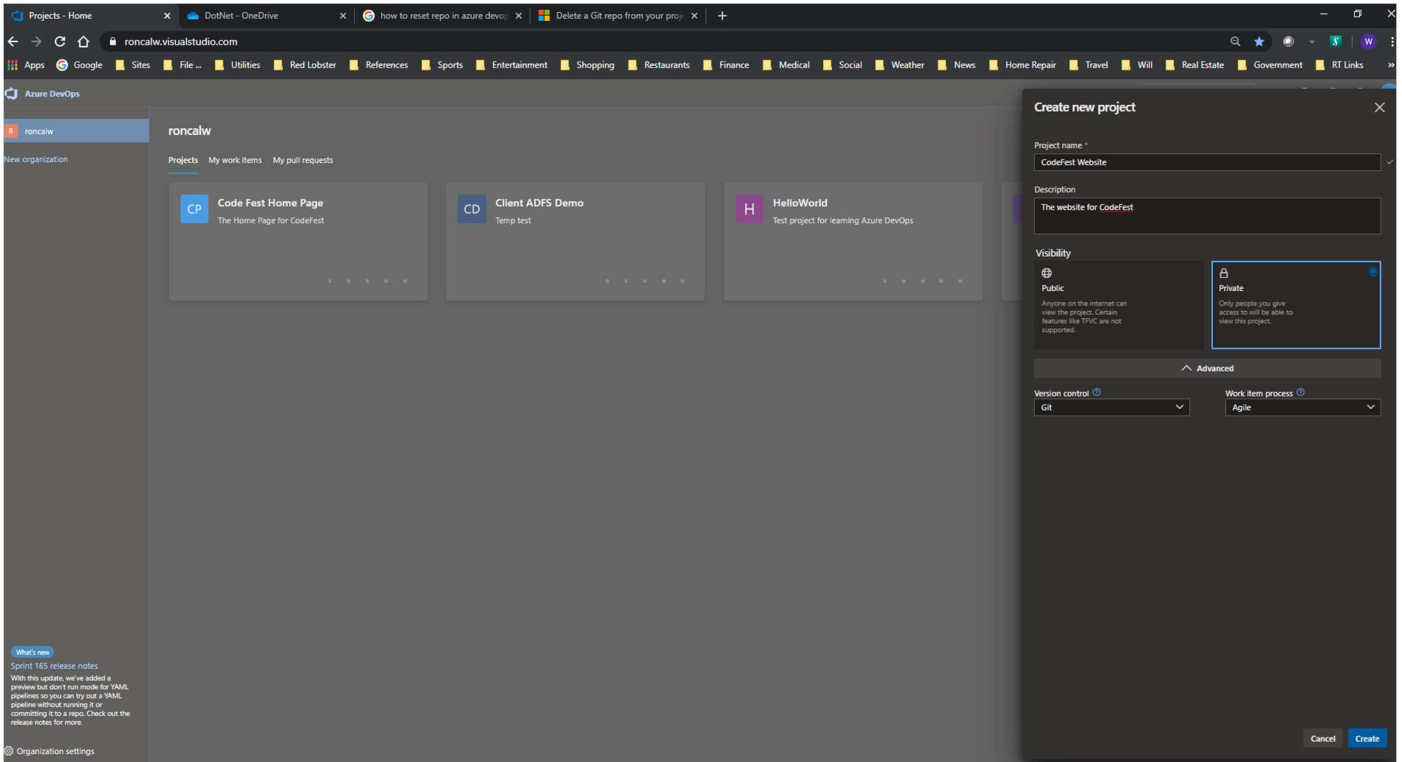


After Adding to Git



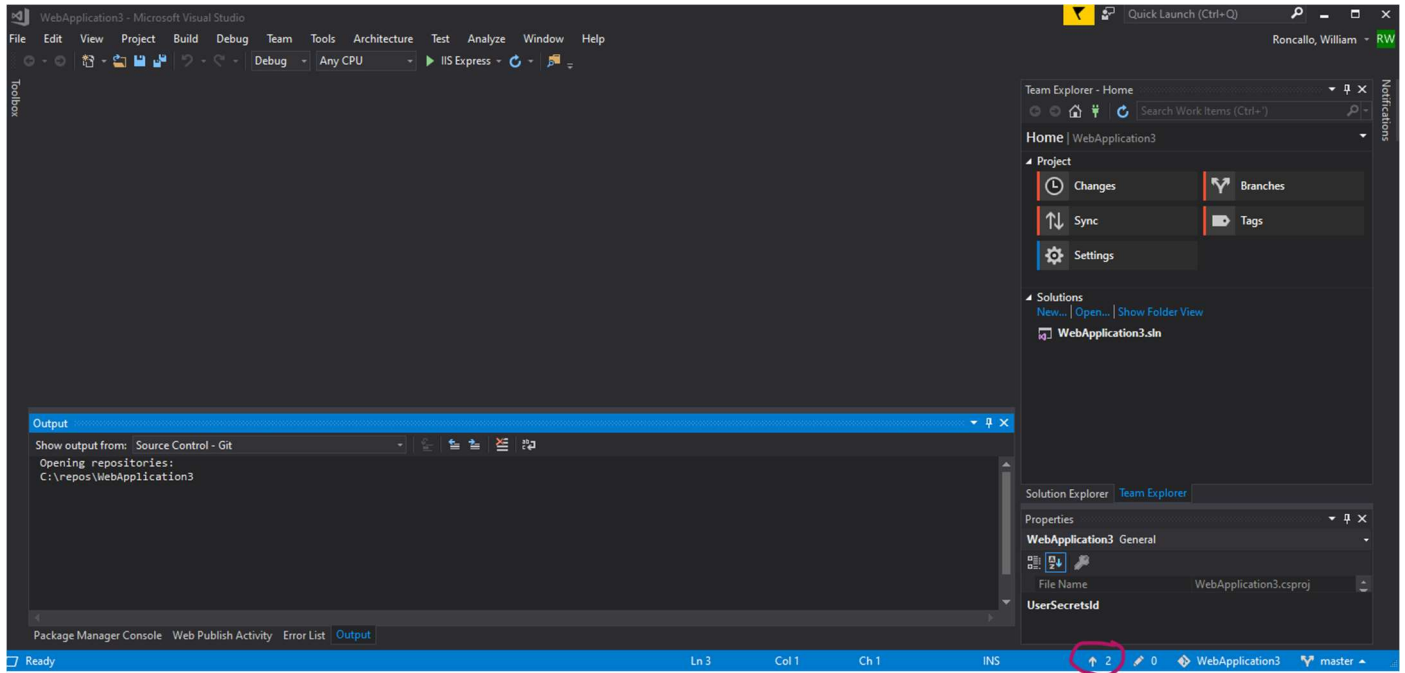
Add to Azure DevOps Project

Create the Azure DevOps Project

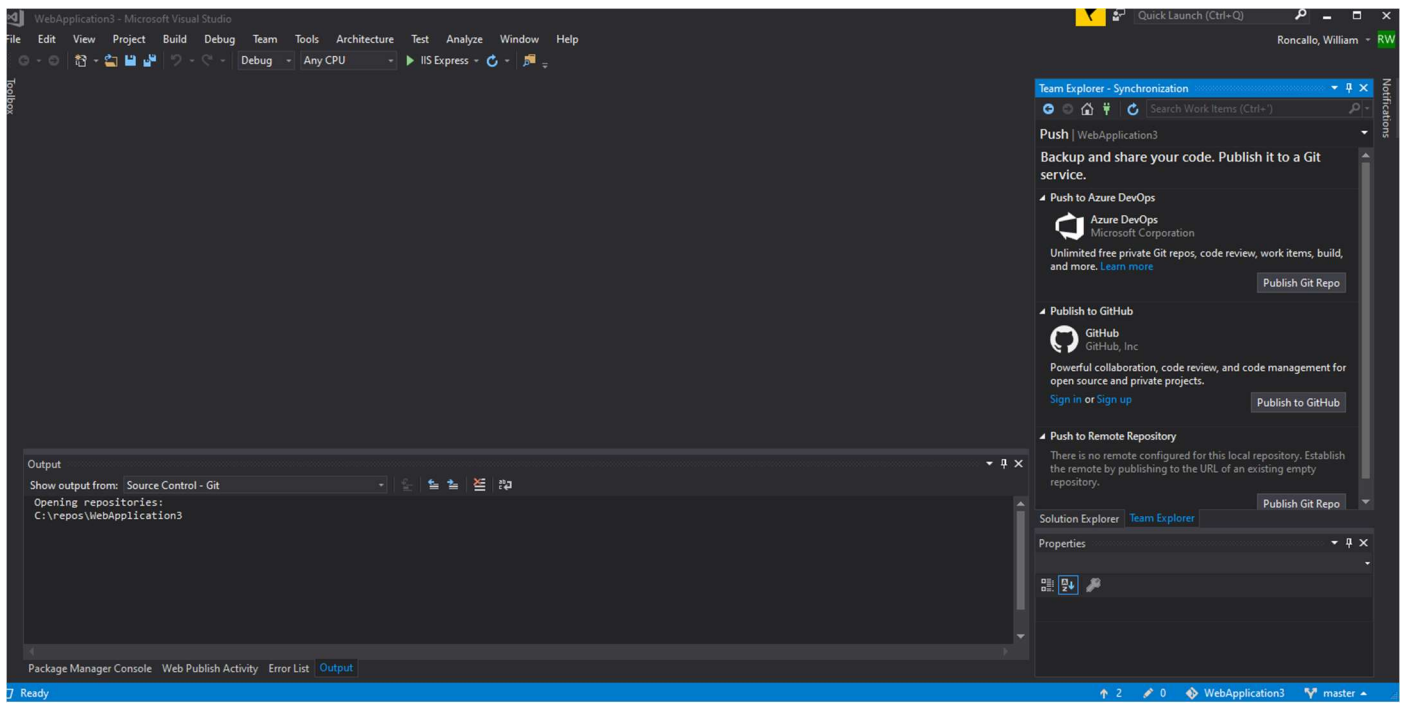


Push the Project to Azure DevOps

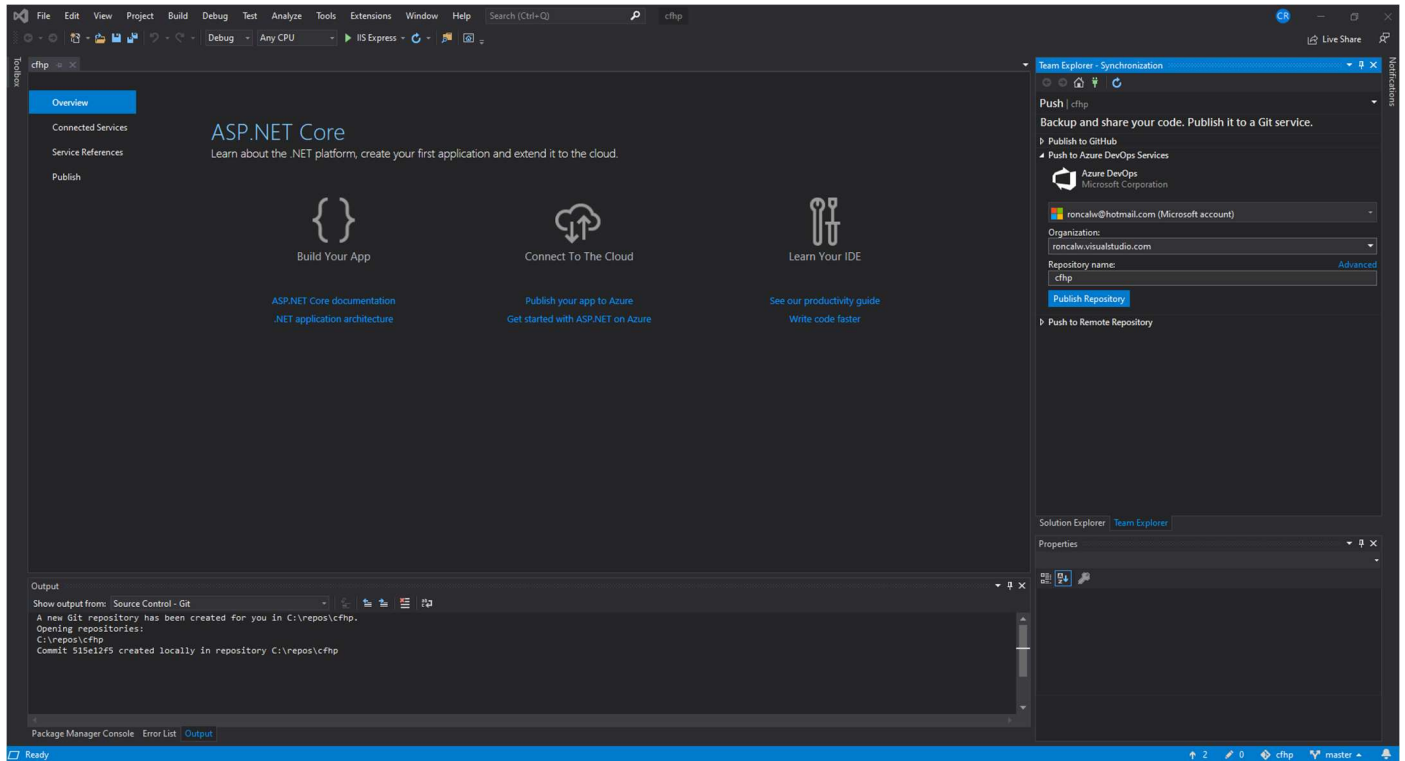
In the section above for Create Local Git Repository, after adding the local Git you get the below



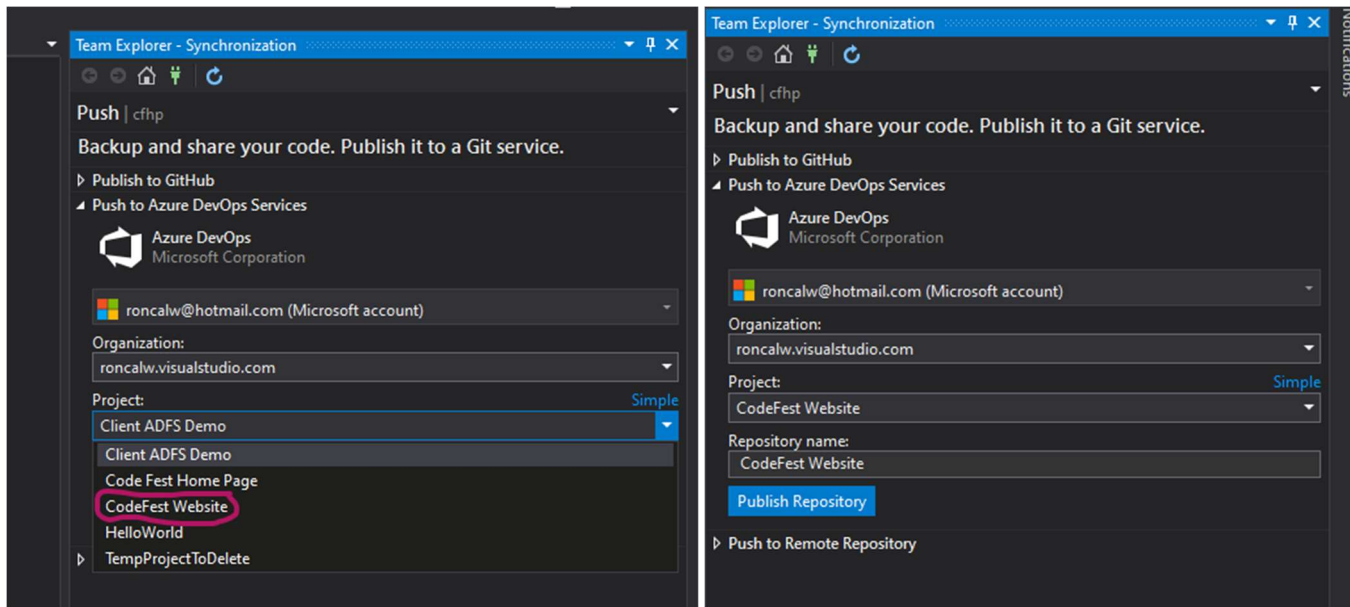
Click on the # 2 that is circled above to get this screen



Then, click on the Publish Git Repo button in the Azure DevOps section, this will bring you to this window

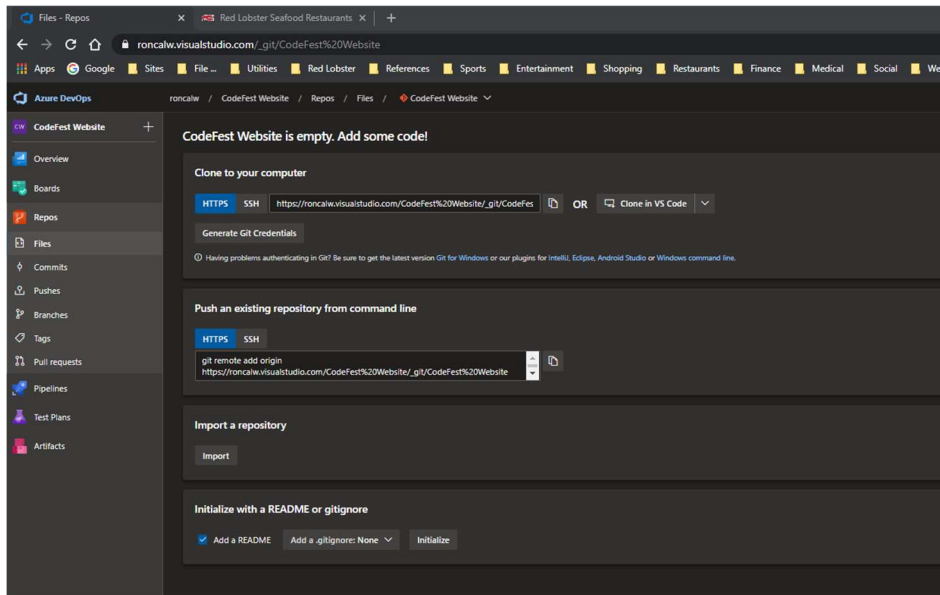


Click on Advanced to the right of Repository name to show list of projects, select the project name created above, and leave the default Repository name

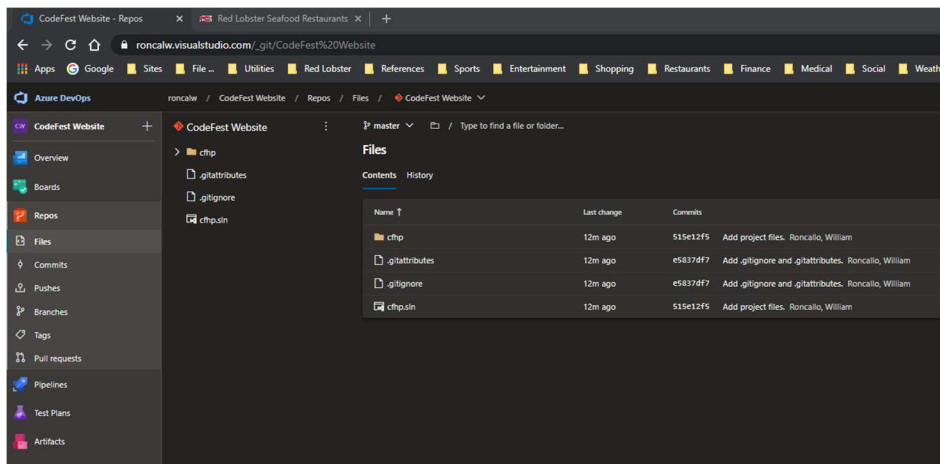


Click on Publish Repository to upload the Repository to the selected Azure DevOps project

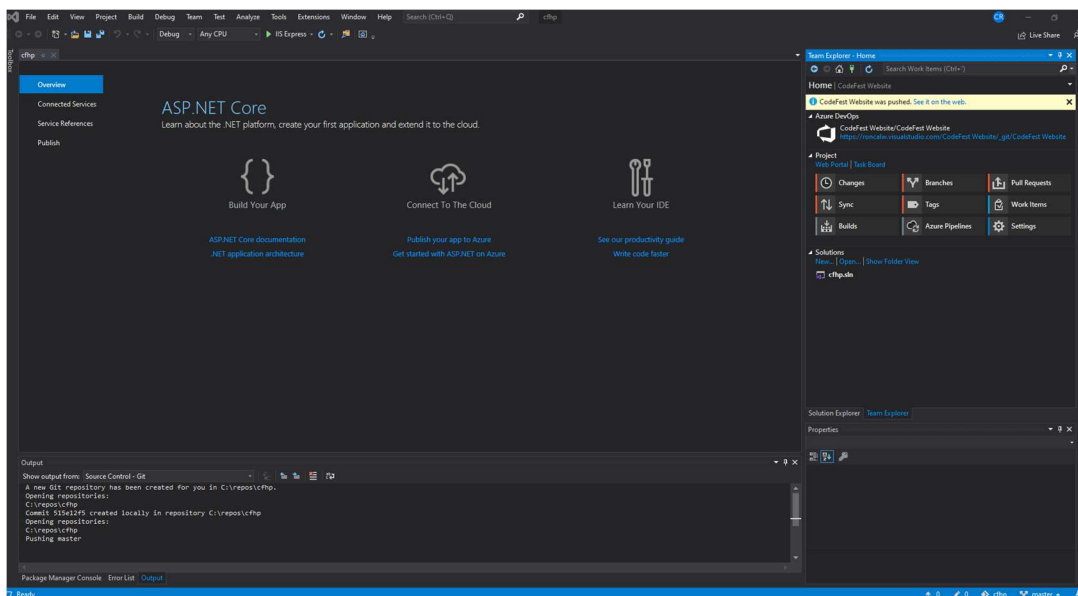
Before



After



After in Visual Studio



Default Pages

All of the default pages prior to making any changes

Project File

Where it says netcoreapp3.1 that is the moniker

```
cfhp.csproj  X launchSettings.json  appsettings.json  appsettings.Development
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2
3  <PropertyGroup>
4      <TargetFramework>netcoreapp3.1</TargetFramework>
5  </PropertyGroup>
6
7  </Project>
8
```

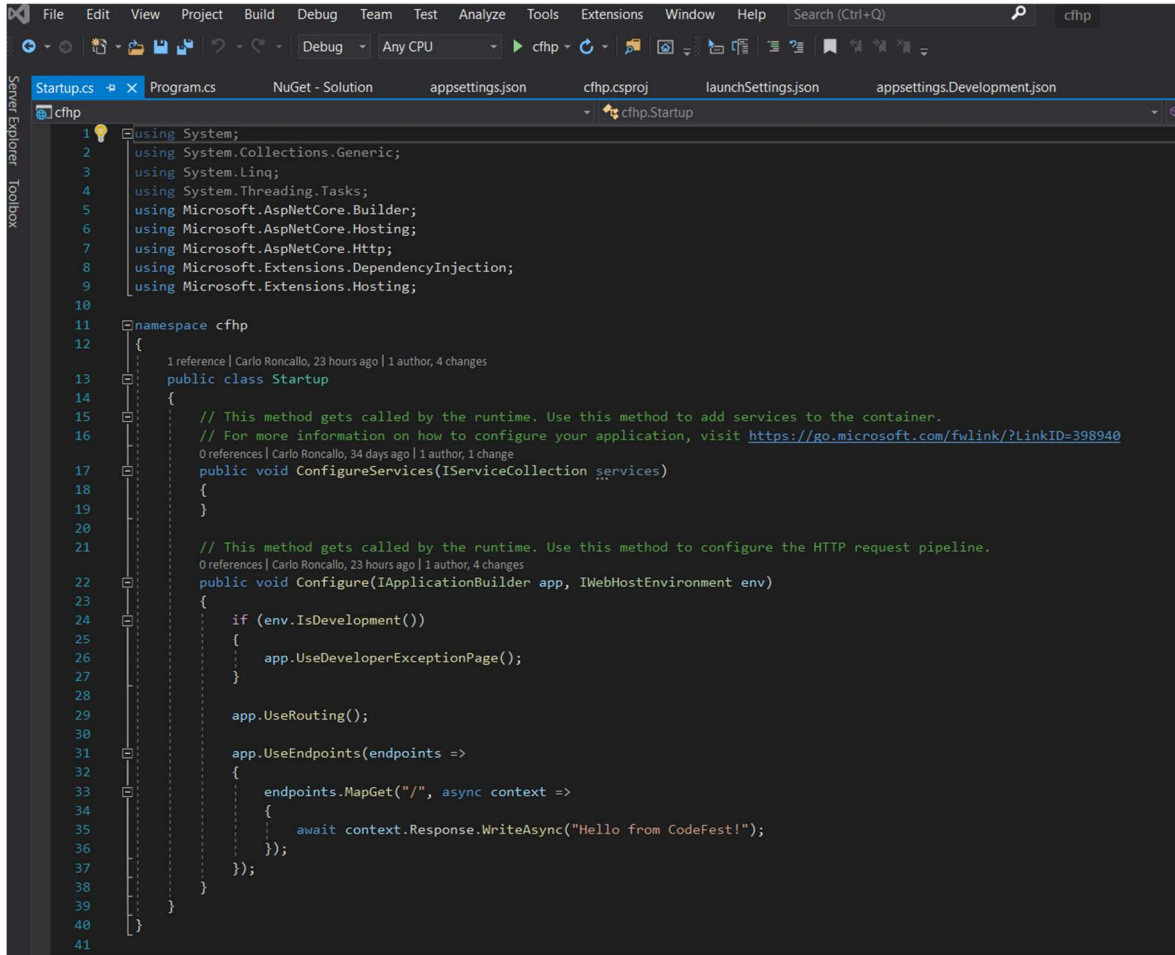
Launch Settings

```
cfhp.csproj  launchSettings.json  X  appsettings.json  appsettings.Development.json  NuGet - Sc
Schema: http://json.schemastore.org/launchsettings
1  {
2  "iisSettings": {
3      "windowsAuthentication": false,
4      "anonymousAuthentication": true,
5  "iisExpress": {
6      "applicationUrl": "http://localhost:52979",
7      "sslPort": 44386
8  }
9  },
10 "profiles": {
11 "IIS Express": {
12     "commandName": "IISExpress",
13     "launchBrowser": true,
14     "environmentVariables": {
15         "ASPNETCORE_ENVIRONMENT": "Development"
16     }
17 },
18 "cfhp": {
19     "commandName": "Project",
20     "launchBrowser": true,
21     "applicationUrl": "https://localhost:5001;http://localhost:5000",
22     "environmentVariables": {
23         "ASPNETCORE_ENVIRONMENT": "Development"
24     }
25 }
26 }
27 }
28 }
```



```
Program.cs x NuGet - Solution appsettings.json cfhp.csproj launchSettings.json appsett
cfhp
1 using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Threading.Tasks;
5     using Microsoft.AspNetCore.Hosting;
6     using Microsoft.Extensions.Configuration;
7     using Microsoft.Extensions.Hosting;
8     using Microsoft.Extensions.Logging;
9
10 namespace cfhp
11 {
12     0 references | Carlo Roncallo, 34 days ago | 1 author, 1 change
13     public class Program
14     {
15         0 references | Carlo Roncallo, 34 days ago | 1 author, 1 change
16         public static void Main(string[] args)
17         {
18             CreateHostBuilder(args).Build().Run();
19
20         1 reference | Carlo Roncallo, 34 days ago | 1 author, 1 change
21         public static IHostBuilder CreateHostBuilder(string[] args) =>
22             Host.CreateDefaultBuilder(args)
23                 .ConfigureWebHostDefaults(webBuilder =>
24                 {
25                     webBuilder.UseStartup<Startup>();
26                 });
27     }
28 }
```

Startup



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Builder;
6 using Microsoft.AspNetCore.Hosting;
7 using Microsoft.AspNetCore.Http;
8 using Microsoft.Extensions.DependencyInjection;
9 using Microsoft.Extensions.Hosting;
10
11 namespace cfhp
12 {
13     1 reference | Carlo Roncallo, 23 hours ago | 1 author, 4 changes
14     public class Startup
15     {
16         // This method gets called by the runtime. Use this method to add services to the container.
17         // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
18         0 references | Carlo Roncallo, 34 days ago | 1 author, 1 change
19         public void ConfigureServices(IServiceCollection services)
20         {
21         }
22
23         // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
24         0 references | Carlo Roncallo, 23 hours ago | 1 author, 4 changes
25         public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
26         {
27             if (env.IsDevelopment())
28             {
29                 app.UseDeveloperExceptionPage();
30             }
31
32             app.UseRouting();
33
34             app.UseEndpoints(endpoints =>
35             {
36                 endpoints.MapGet("/", async context =>
37                 {
38                     await context.Response.WriteAsync("Hello from CodeFest!");
39                 });
40             });
41         }
42     }
43 }
```

```
namespace cfhp
{
    public class Startup
    {
        // 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)
        {
        }

        // 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.MapGet("/", async context =>
                {
                    await context.Response.WriteAsync("Hello from CodeFest!");
                });
            });
        }
    }
}
```

Appsettings.JSON

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft": "Warning",
6       "Microsoft.Hosting.Lifetime": "Information"
7     }
8   },
9   "AllowedHosts": "*"
10 }
11
```

Nuget Packages

NuGet - Solution

appsettings.json cfhp.csproj launchSettings.json appsettings.Development.json Program.cs Startup.cs

Browse Installed Updates Consolidate

Search (Ctrl+L) Include prerelease

No packages found

Using Static Pages

Add MVC to Startup

- ConfigureServices
 - Add services.AddControllersWithViews
- Configure
 - Add app.UseStaticFiles
 - Change the section for app.UseEndpoints (see below)

```
public class Startup
{
    // 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.AddControllersWithViews();
    }

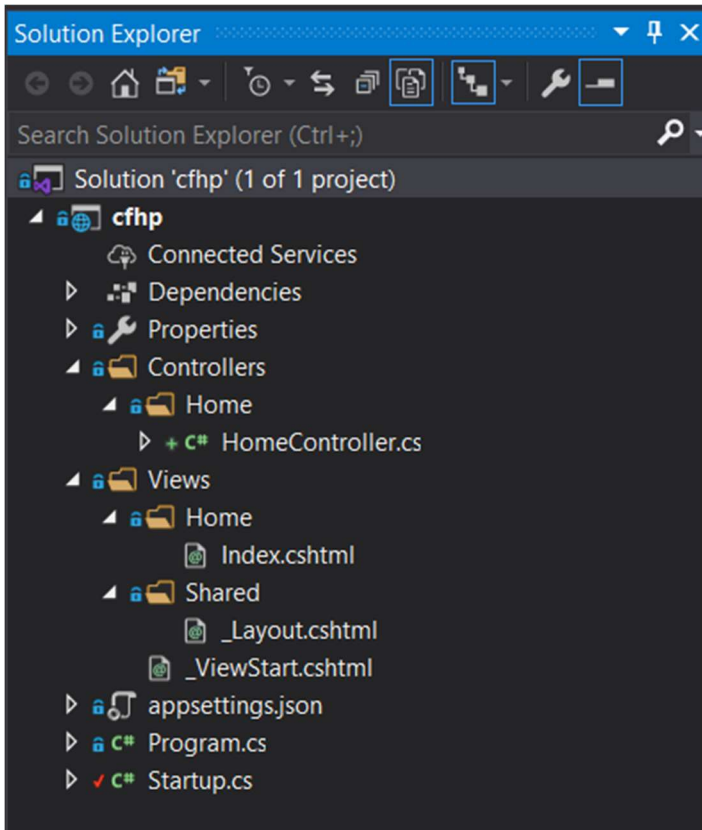
    // This method gets called by the runtime. Use this method to configure the HTTP
    // request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseStaticFiles();

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}
```

Add Controllers and Views Folders



Files are described below

Controllers

Added Controllers folder, then the file. Could not add a controller by right clicking and selecting Controller even though the controller was empty. Because when you do that it adds a nuget package that adds hundreds of files to the project. The Nuget package is some kind of codegenerator package.

So I copied a controller from Windows Explorer and pasted the file into the Controllers/Home folder that I made at the root of the project. I then changed the namespace's 1st section from other project name to this one.

For Views I copied over from the previous projects as well, the pic above is what it looks like when done.

HomeController.cs

Controller was copied in so changed the namespace's 1st section accordingly

```
namespace cfhp.Controllers.Home
{
    public class HomeController: Controller
    {
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult About()
        {
            return View();
        }
    }
}
```

Views

- Added Views Folder, in Views Folder added
 - Home folder, in Home folder, added
 - Index.cshtml (copied from another project pasted straight into the solution explorer)
 - Shared folder, in Shared folder, added
 - _Layout.cshtml (copied from another project pasted straight into the solution explorer0, this is called from the _ViewStart.cshtml page below)
 - _ViewStart.cshtml (copied from another project pasted straight into the solution explorer)
 - _ViewImports.cshtml (copied from another project pasted straight into the solution explorer)

Index.cshtml

To add this without using any wizard that then scaffolds, right click on Home folder, left click on Add item ..., then search for text, and then change the name from TextFile.txt to Index.cshtml. Although I probably could have selected "Razor View" from Add item to get the same thing without scaffolding as well

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

<div class="jumbotron">
    <h1>Minimal Configuration</h1>
</div>
<div class="row">
    <div class="col-md-6">
        <h2>Template with Minimal Configuration</h2>
        <p>
            This is a sample application that just has MVC and Configuration
        </p>
    </div>
    <div class="col-md-6">
        <h2>Documentation</h2>
        <p><a class="btn btn-default" href="https://docs.asp.net/en/latest/data/entity-framework-6.html">See the documentation &raquo;</a></p>
    </div>
</div>
```

`_Layout.cshtml`

This is the main outer page, the other views get loaded from here in the `RenderBody` method below

The script and css files will get added to the project in a later section. In the includes section below, the `~` represents "wwwroot" which is the folder created for static folders explained below

```
<!DOCTYPE html>
<html>
<head>
  <title>Code Fest: @ViewBag.Title</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="~/node_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
  <nav class="navbar navbar-expand-md navbar-dark bg-dark mb-4">
    <a class="navbar-brand" href="#">Top navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <!--Menu Bar-->-->
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" asp-controller="Home" asp-action="Index">Home <span
class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" asp-controller="Persons" asp-
action="Index">Persons</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" asp-controller="Home" asp-action="About">About</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#">Disabled</a>
        </li>
      </ul>
      <form class="form-inline mt-2 mt-md-0">
        <input class="form-control mr-sm-2" type="text" placeholder="Search" aria-
label="Search">
        <button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Search</button>
      </form>
    </div>
  </nav>

  <section class="container">
    <h2 class="text-center">@ViewBag.Title</h2>
    @RenderBody()
  </section>

  <footer class="container">
    <div class="text-center">Copyright 2018 Code Fest</div>
  </footer>

  <script src="~/node_modules/jquery/dist/jquery.min.js"></script>
  <script src="~/node_modules/popper.js/dist/popper.min.js"></script>
  <script src="~/node_modules/bootstrap/dist/js/bootstrap.min.js"></script>

</body>
</html>
```


[_ViewStart.cshtml](#)

This says to load the `_Layout.cshtml` file

```
@{  
    Layout = "_Layout";  
}
```

[_ViewImports.cshtml](#)

This says to import the TagHelpers on each and every view page. This is needed to work with the various TagHelpers such as `asp-controller="Home" asp-action="Index"` that is in the `_Layout.cshtml` example above. This makes it so that clicking on this link takes the user to the Index action of the HomeController.

Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files. For example, the built-in ImageTagHelper can append a version number to the image name. Whenever the image changes, the server generates a new unique version for the image, so clients are guaranteed to get the current image (instead of a stale cached image). There are many built-in Tag Helpers for common tasks - such as creating forms, links, loading assets and more - and even more available in public GitHub repositories and as NuGet packages. Tag Helpers are authored in C#, and they target HTML elements based on element name, attribute name, or parent tag. For example, the built-in LabelTagHelper can target the HTML `<label>` element when the LabelTagHelper attributes are applied.

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Adding Client Script Files

wwwroot Folder

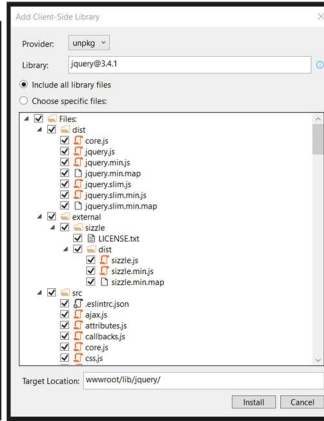
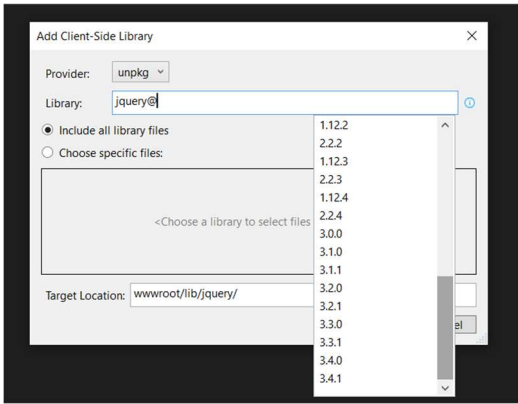
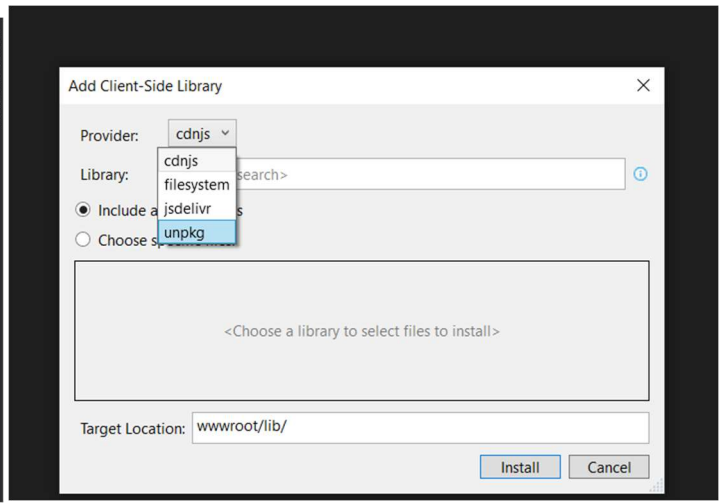
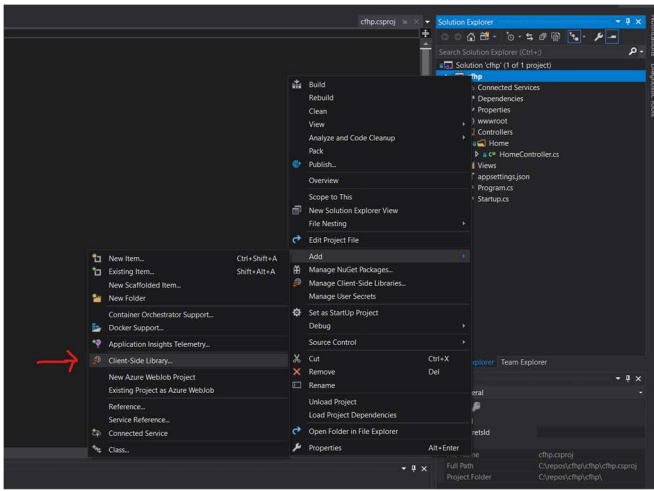
Add this folder at the root of the project. The `wwwroot` folder is new in ASP.NET 5 to store all of the static files in your project. Any files including HTML files, CSS files, image files, and JavaScript files which are sent to the users browser should be stored inside this folder.

Two Ways of Adding Script Libraries

Libman

From looking at the Pluralsight Video this looks like the best way to add scripts, since you don't have to worry about doing it via the Package.json file way (described below). Using Package.json uses NPM to download the entire library for each tool, like JQuery, or Bootstrap, when you do it this way their entire library gets downloaded into the same folder of the project file, but because they are static files you can't do anything with them there, they then have to be manually added over to the `wwwroot` folder.

Using Libman, though, the wizard, uses NPM (if that is what you say to use) to download what you say to download from the CDNs and it lets you determine whereb(such as in the `wwwroot` folder) as well. After doing it, the wizard creates a `libman.json` file that can then be configured manually.



Package.json File

Do this in another project and then copy them back in this project to the wwwroot folder above, so that you only what is necessary for the project. Do this to use NPM to download the full set of client side scripts for each type. Added these one by one; it immediately creates a node_modules folder, with a subfolder for each one.

```
{
  "version": "1.0.0",
  "name": "mypackage",
  "private": true,
  "devDependencies": {
    "jquery": "3.3.1",
    "popper.js": "1.14.3",
    "bootstrap": "4.1.2",
    "font-awesome": "4.7.0"
  }
}
```

Making the wwwroot folder and the Package.json file will create the ItemGroups in the project file that are below in the 1st box or if copying the script files over from another project instead it looks like the 2nd box below.

Either way, I then changed this section to just what I have in the 3rd box below

1st box - **FYI - The Folder tag below does not work when trying to publish as a self-contained application**, it will do nothing for copying anything in that wwwroot folder. Had to change it to Content

```
<ItemGroup>
  <Compile Remove="wwwroot\node_modules\*" />
  <Content Remove="wwwroot\node_modules\*" />
  <EmbeddedResource Remove="wwwroot\node_modules\*" />
  <None Remove="wwwroot\node_modules\*" />
</ItemGroup>

<ItemGroup>
  <Folder Include="wwwroot\" />
</ItemGroup>
```

2nd Box - If dropping the files in manually from another folder that ran the Package.json file you get this in the proj file

```
<ItemGroup>
  <Content Include="wwwroot\node_modules\bootstrap\dist\css\bootstrap.min.css" />
</ItemGroup>

<ItemGroup>
  <None Include="wwwroot\node_modules\bootstrap\dist\css\bootstrap.min.css.map" />
  <None Include="wwwroot\node_modules\bootstrap\dist\js\bootstrap.min.js" />
  <None Include="wwwroot\node_modules\bootstrap\dist\js\bootstrap.min.js.map" />
  <None Include="wwwroot\node_modules\jquery\dist\jquery.min.js" />
  <None Include="wwwroot\node_modules\jquery\dist\jquery.min.map" />
  <None Include="wwwroot\node_modules\popper.js\dist\popper.min.js" />
  <None Include="wwwroot\node_modules\popper.js\dist\popper.min.js.map" />
</ItemGroup>
```

In either case, I changed to the following. **HOWEVER** this would cause an error when I added an img folder to the wwwroot folder. Received the following error: Duplicate 'Content' items were included. The . NET SDK includes 'Content' items from your project directory by default. You can either remove these items from your project file, or set the 'EnableDefaultContentItems' property to 'false' if you want to explicitly include them in your project file.

```
<ItemGroup>
  <Content Include="wwwroot\**\*.*" CopyToOutputDirectory="PreserveNewest" />
</ItemGroup>
```

I thought I needed this to be able to publish to iis (it would skip the node_module folder), but I found out that the publish would work for images without it so I tried a new folder called lib, took out the section and it worked. It is as if the publish command looks for the img and lib folder by default for publishing the wwwroot folder contents

Project File

Added the OS line at the top, this is what the final version looked like

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <RuntimeIdentifiers>win10-x64</RuntimeIdentifiers>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="wwwroot\**\*.*" CopyToOutputDirectory="PreserveNewest" />
  </ItemGroup>

</Project>
```

Navigation Bar

ViewBags

ViewBag is a dynamic property and a wrapper around the ViewData. It is a dynamic object and any number of fields can be added into it dynamically. We can pass both strongly and weakly typed data using ViewBag. It doesn't require any typecasting for complex and strongly typed data.

ViewBag is also one of the properties of "ControllerBase" class so when we create a controller, that controller will automatically inherit the "Controller" abstract class and this "Controller" class inherits "ControllerBase" abstract class, that's why we can access this ViewBag property in each controller.

As ViewBag wraps the ViewData so it stores the data into ViewDataDictionary, eg. the Key (string) of ViewData (ViewData["employee"]) and property of ViewBag (ViewBag.employee) are the aliases of each other. Both will print the same output.



The reason why ViewBags are discussed in this section for Navigation is because it is changed by the code upon navigation and can be useful for assignments to then display differently based on what page you are rendering. This can be done in two ways, at the top of the razor page like in box 1 here or in the controller like box 2, I prefer box 2.

Box 1 – At the top of the razor page

```
@{
  ViewData["Title"] = "About Page";
}
```

Box 2 – In the controller, notice I added Title and NavIndex, I use Title to display what I want to show in the Title tag of the HTML and I use NavIndex to determine when I want the menu item in the nav bar to be active (show as highlighted)

```
public class HomeController: Controller
{
    public IActionResult Index()
    {
        ViewData["Title"] = "Home Page";
        ViewData["NavIndex"] = "one";
        return View();
    }
    public IActionResult About()
    {
        ViewData["Title"] = "About Page";
        ViewData["NavIndex"] = "two";
        return View();
    }
}
```

Navigation Bar

To make sure the nav bar highlights and stays highlighted when clicking on a menu link set the class of the li tag, (that holds the anchor tag), to active. This can be set by looking at the ViewBag.NavIndex when loading the page to see what the value is when loading, and if it is the same value show the “active” attribute (see highlight below).

In the box below when the Home controller loads the Index view, just before the controller loads it via the “return View();” command, I set the ViewData["NavIndex"] = "one"; I did that for when the controller loads the About page as well, except that I set that to “two”. I did all this in the controller class (see 2nd box below).

```
<ul class="navbar-nav mr-auto">
    <li class="nav-item @(ViewBag.NavIndex == "one" ? "active" : "")">
        <a class="nav-link" asp-controller="Home" asp-action="Index">
            Home <span class="sr-only">(current)</span>
        </a>
    </li>
    <li class="nav-item @(ViewBag.NavIndex == "two" ? "active" : "")">
        <a class="nav-link" asp-controller="Home" asp-action="About">About</a>
    </li>
    <li class="nav-item">
        <a class="nav-link disabled" href="#">Disabled</a>
    </li>
</ul>
```

Home controller that makes the above work

```
public class HomeController: Controller
{
    public IActionResult Index()
    {
        ViewData["Title"] = "Home Page";
        ViewData["NavIndex"] = "one";
        return View();
    }
    public IActionResult About()
    {
        ViewData["Title"] = "About Page";
        ViewData["NavIndex"] = "two";
        return View();
    }
}
```

Publishing

```
dotnet publish --framework netcoreapp3.1 --output C:\pub\wwwroot --configuration Release --self-contained -r win10-x64
```

Do the command above before trying CI/CD to make sure everything copies over.