**Introduction**

If, like me, you've been a C#/MVC developer for a while, working in Visual Studio. You've probably been adding Jquery snippets to your views, leveraging Jquery-ui or Bootstrap and writing javascript functions to improve the responsiveness of your sites. Perhaps, as a good dev should, you've moved on to writing javascript libraries and including these in your MVC views.

Being conscious that your sites still have to support IE 11 (yes, even in 2021!) your javascript has been ES5 compliant.

But something's been nagging at you for a while now. What about React? How can I take the next step and start producing React front-ends? How can I leverage modern javascript techniques?

But you look at your existing code-base and read the getting-started articles on the web and think: *I don't want to re-invent my whole site. Can't someone just explain to me how to add React to the next view I have to design and incorporate into my existing MVC site.*

This article is for you!

**Step 1 - Install Node**

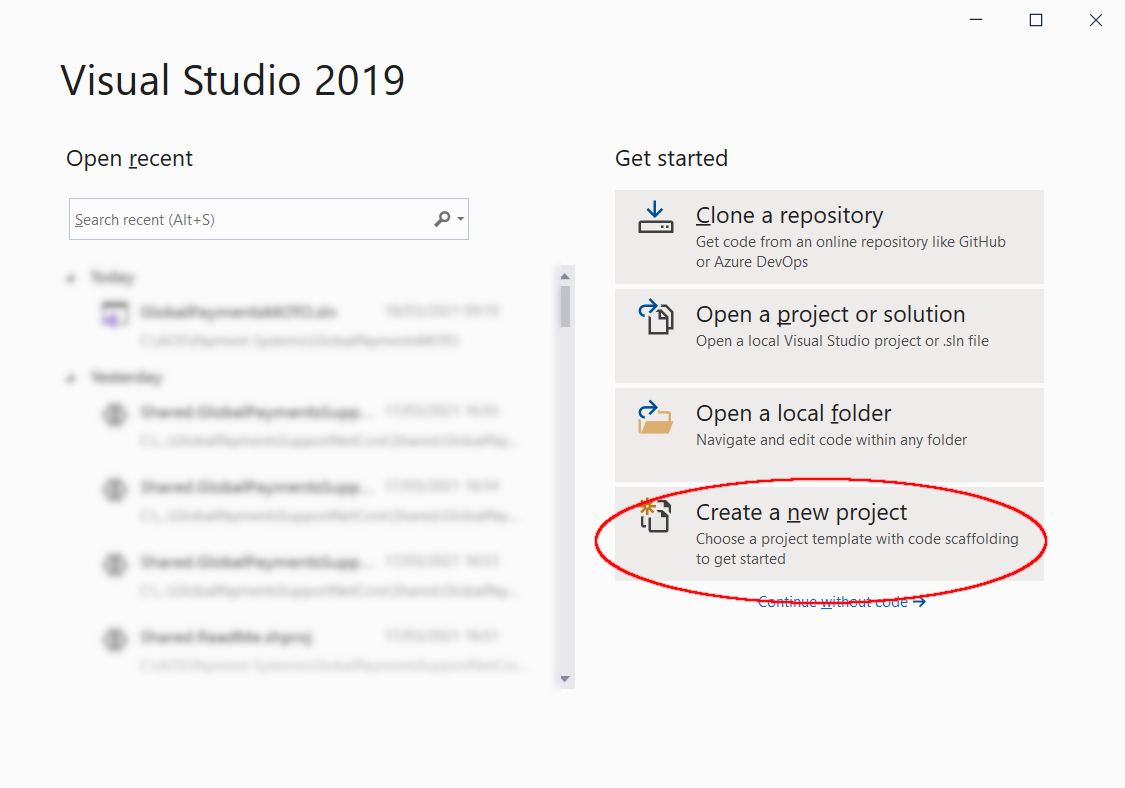
We'll need Node. Node is a package manager like Nuget but for javascript. Before you do anything else you need to install [Node](https://nodejs.org/en/).

So go ahead and do that then come back here. The LTS (Long Term Support) version will be just fine.

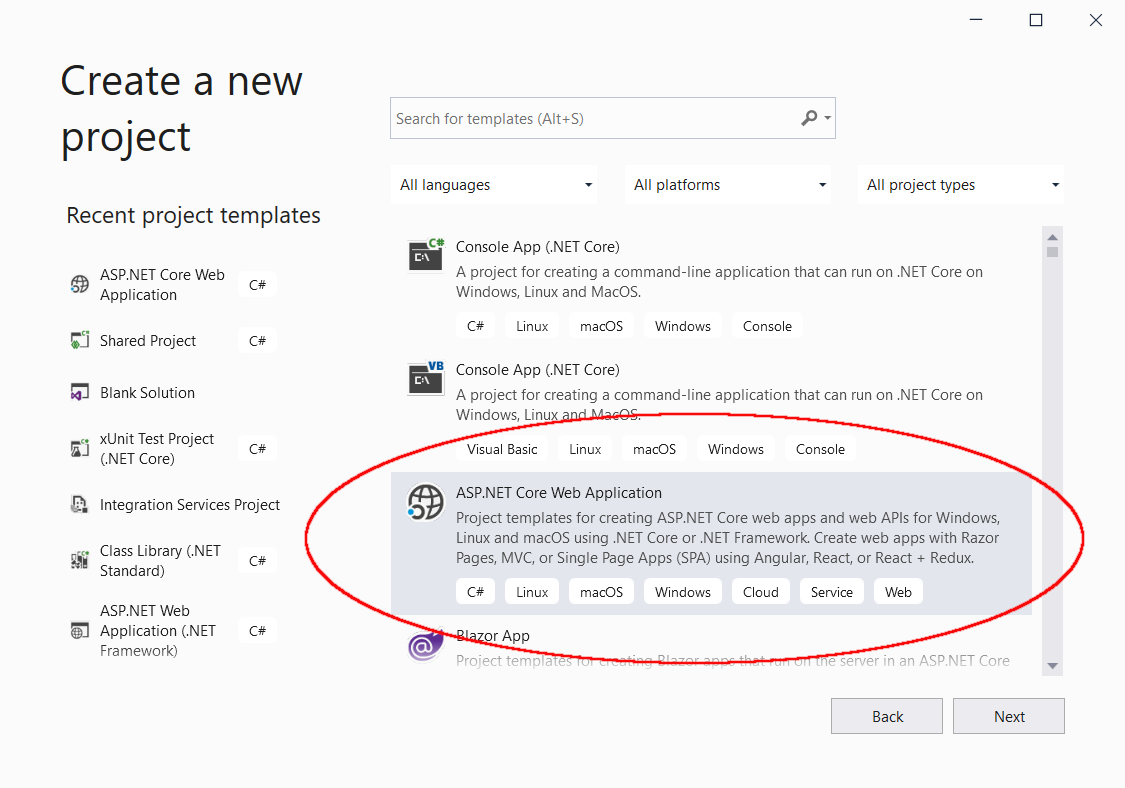
**Step 2 - Create a new site.**

As I don't have access to *your* MVC site I'm going to create a new mvc app in Visual Studio

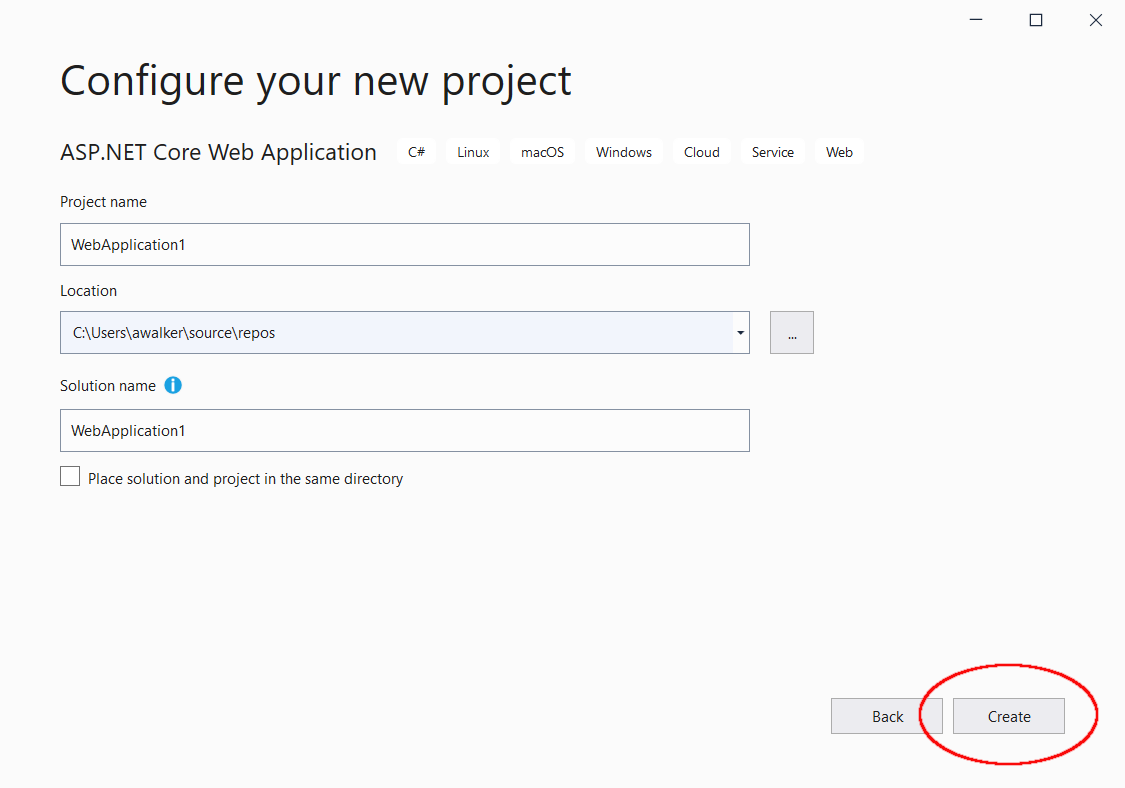
1. Start Visual Studio and click *Create a New Project*



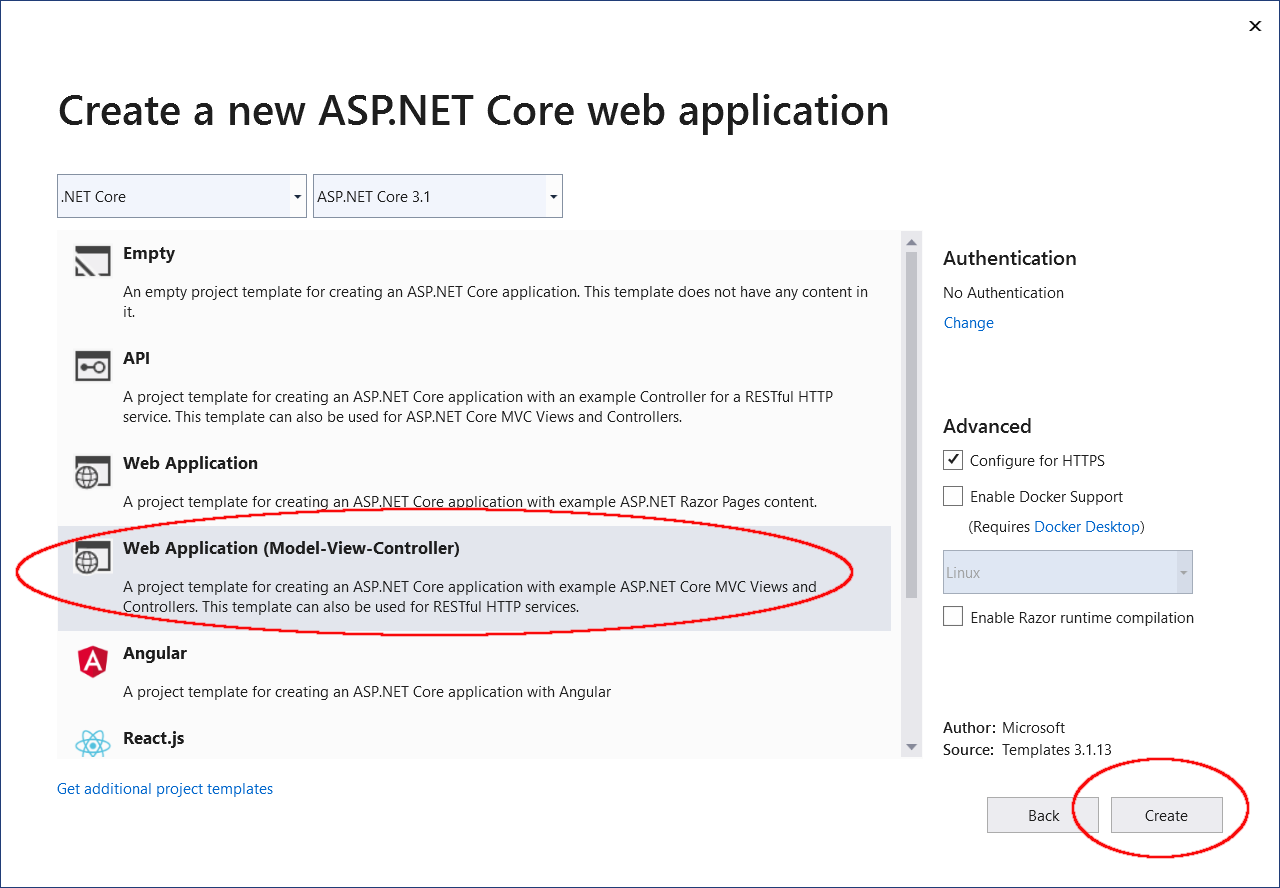
2. Create a new ASP.NET Core Web Application



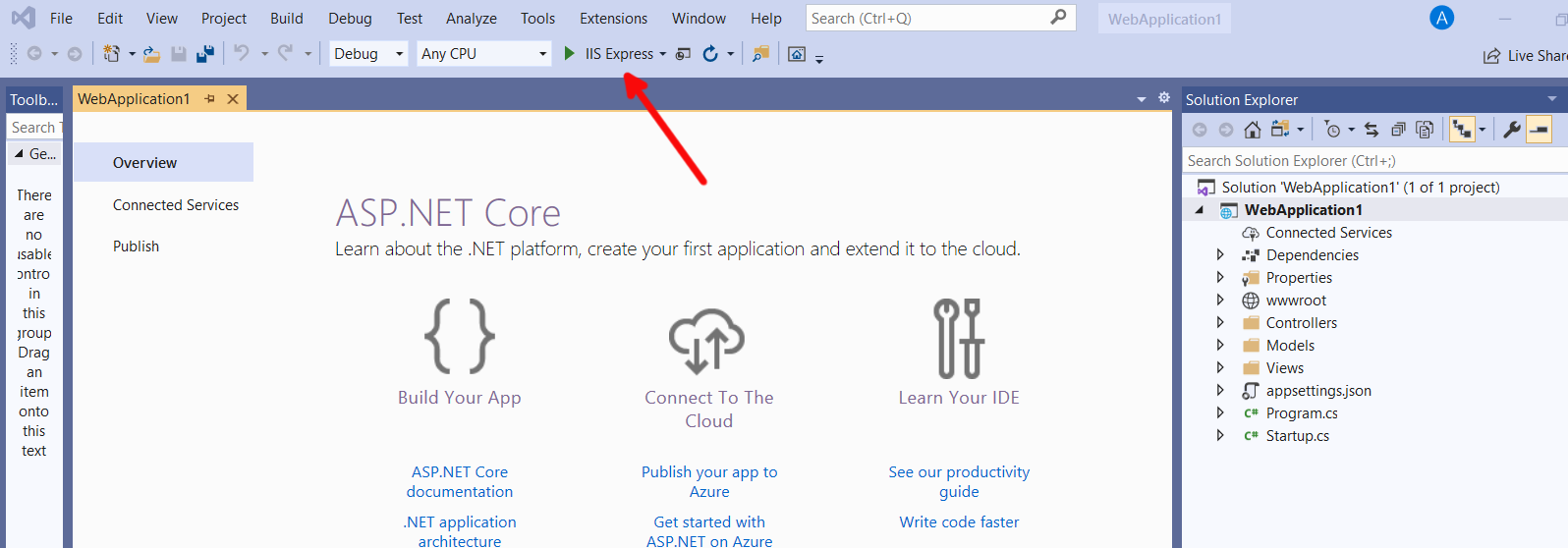
3. Name the project then click the *Create* button.



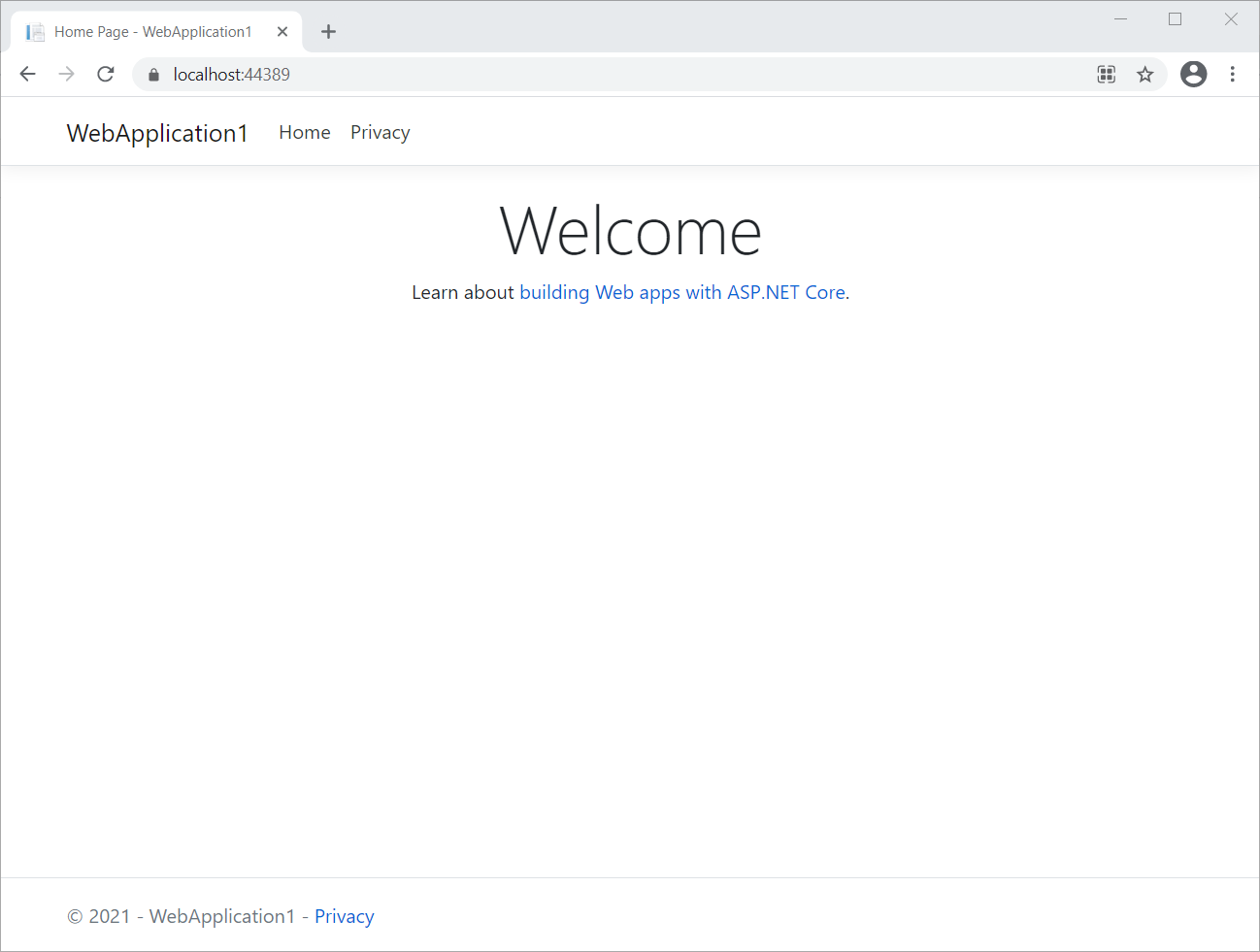
4. Finally, tell Visual Studio you're creating an MVC app, then click *Create*.



5. As a check that all is well run the application by clicking the *Run* button



The new website will load.

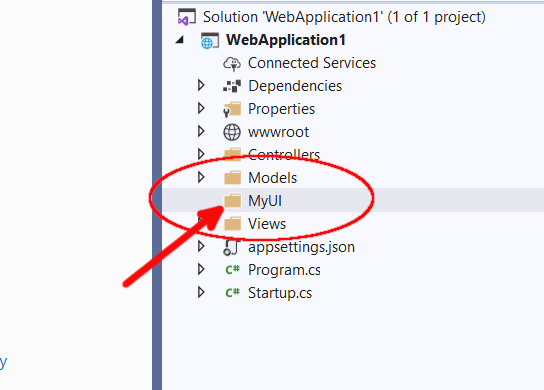


**Step 3 - Add package.json**

In this step we're going to add the required libraries to get our view to work with React. We're also going to carry out the appropriate set up steps to ensure our React components can also be rendered by IE 11.

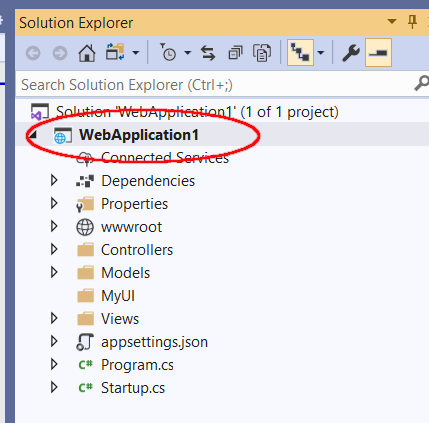
Working with React will mean a bit more command line activity than many .NET developers are accustomed but it soon becomes the new normal.

1. Create a folder in your project to hold the React source files. Call this folder *MyUI*.

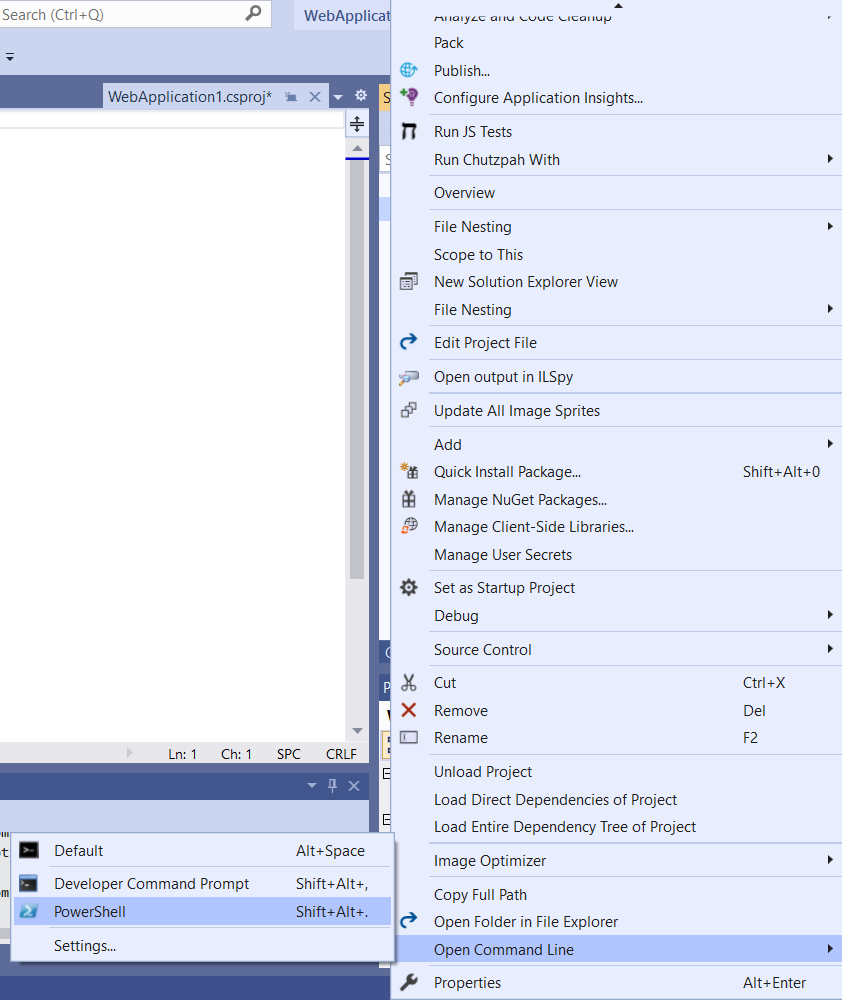


2. Add the package.json file by following these steps.

a. Right-click the project root to call up the context menu.



b. Select *Open Command Line -> Powershell* from the context menu. (You can use the command line shell instead if you wish.)



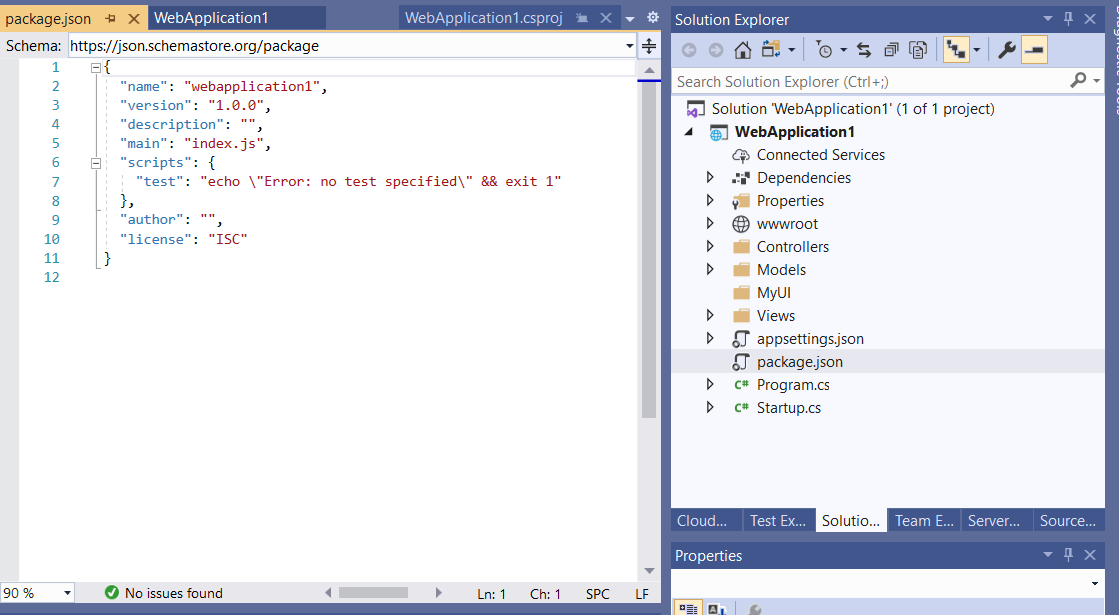
c. At the powershell prompt type:

npm init

d. Continue pressing return (and thus accepting the default response) for each of the questions asked by the node package manager (npm) until all the questions are answered.

At this point the node package manager creates the package.json file in the root folder of our MVC app. We will have to modify one or two of the responses later but for now we're good.

e. Confirm the package.json file exists by opening it right there in Visual Studio.



**Step 4 - Installing packages**

We will ask the node package manager to install the packages we need. We will need packages for:

* React
* Webpack - this is the bundler so that only the javascript we actually used from the node modules we install is added to our mvc site.
* Babel - this is a transpiler. We will use it to transpile the modern javascript we will write to a form that can be interpreted by older browsers - specifically IE 11.

Let's get started once more with the command line we opened earlier. After each command you can review package.json to see which dependencies have now been added.

1. Open (or return to) the powershell command line.
2. Install React by entering the following command

npm install react react-dom

3. Install babel by entering the following command

npm install --save-dev babel-core babel-loader @babel/preset-react @babel/preset-env

4. Install the babel-polyfill by entering the following command

npm install babel-polyfill

5. Install webpack by entering the following command

npm install --save-dev webpack webpack-cli

6. Upgrade babel-core. My npm installs a lower version (6.x.x) but I need version 7.x.x to work nicely with my webpack. So I manually edited my package.json changing

  "@babel/core": "^6.26.3",

to

  "@babel/core": "^7.12.3",

before running

npm install

to get npm to install the correct version

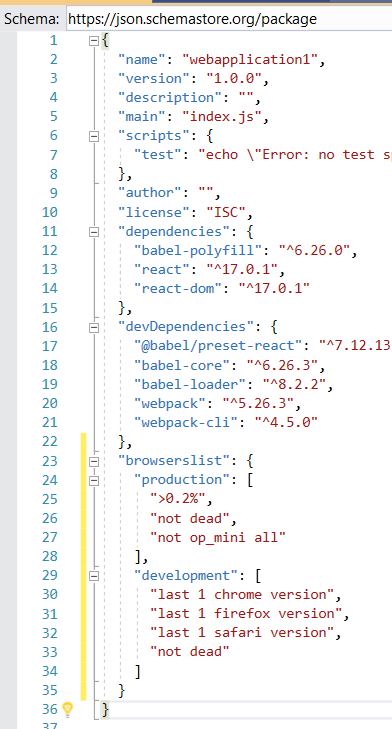
**Step 5 - Getting Webpack and Babel working together**

To get webpack and babel working together we have to update our package.json file and to add a couple of config files.

There are multiple ways on the web using different techniques and differing versions of these files. I have to confess that this was the step I found most difficult to do when figuring this out

1. Tell Webpack not to ignore Internet Explorer in your development environment.

This is achieved by updating the package.json file. The browserslist section should be added as a new entry to our package.json



2. Next add a new blank file called *webpack.config.js* to the root of our project in Visual Studio.

3. Add the following text to the new webpack.config.js file

const path = require('path');

module.exports = {

  mode: 'development',

  devtool: 'inline-source-map',

  entry: ['babel-polyfill', './MyUI/Components/MyUIMain.js'],

  output: {

    filename: 'myui-bundle.js',

    path: path.join(\_\_dirname, 'wwwroot/dist')

  },

  module: {

    rules: [{

      loader: 'babel-loader',

      test: /\.js$/,

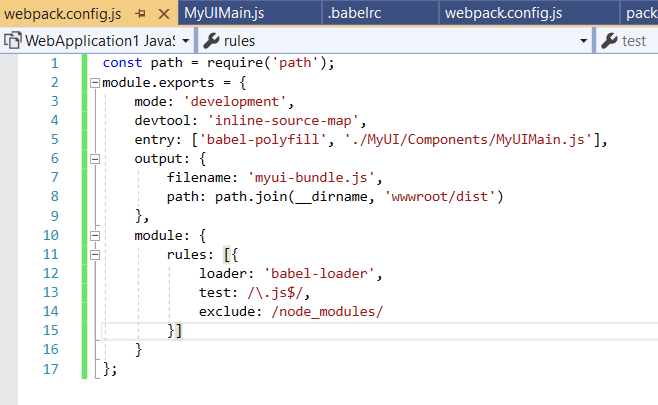
      exclude: /node\_modules/

    }]

  }

};

Here's a picture of the file in Visual Studio



4. Add a .babelrc file by following these steps.

a. In Visual Studio, add a new file called *.babelrc* - note the full stop at the beginning.

add the following text to the file, then save.

{

  "presets": [

    "@babel/preset-env",

    "@babel/preset-react"

  ]

}

**Step 6 - Add a React Component**

Back in Visual Studio for this step.

1. Add a new folder in MyUI and call it *Components*.
2. In this Components folder add a new javascript file called *MyUIMain.js*.
3. Enter the following code into this file.

import React from 'react';

export const MotoUIMain = () => {

  return (

    <div>Hello from React</div>

    )

}

ReactDOM.render(

  <MyUIMain />,

  document.getElementById('myuimain'),

);

**Step 7 - Tell Webpack to produce a distribution bundle**

In this section we're going to edit the scripts section of our package.json.

1. Add the following item to the scripts section of our package,json file. Simply edit the file directly in Visual Studio.

*"wpb": "webpack",*

*Our scripts section will now look like this:*



2. Tell webpack to generate the distribution bundle by running the following command

npm run wpb

3. Confirm the javascript bundle now exists by looking in the Visual Studio solution explorer in the folder www/dist. You'll see the javascript file called *myui-bundle.js.*

**Step 8 - The final stretch**

The last step is to tell our Index view to use this javascript bundle.

1. Open the Views\Home\Index.cshtml view file.
2. Add the following code to the bottom of the file.

<div id="myuimain"></div>

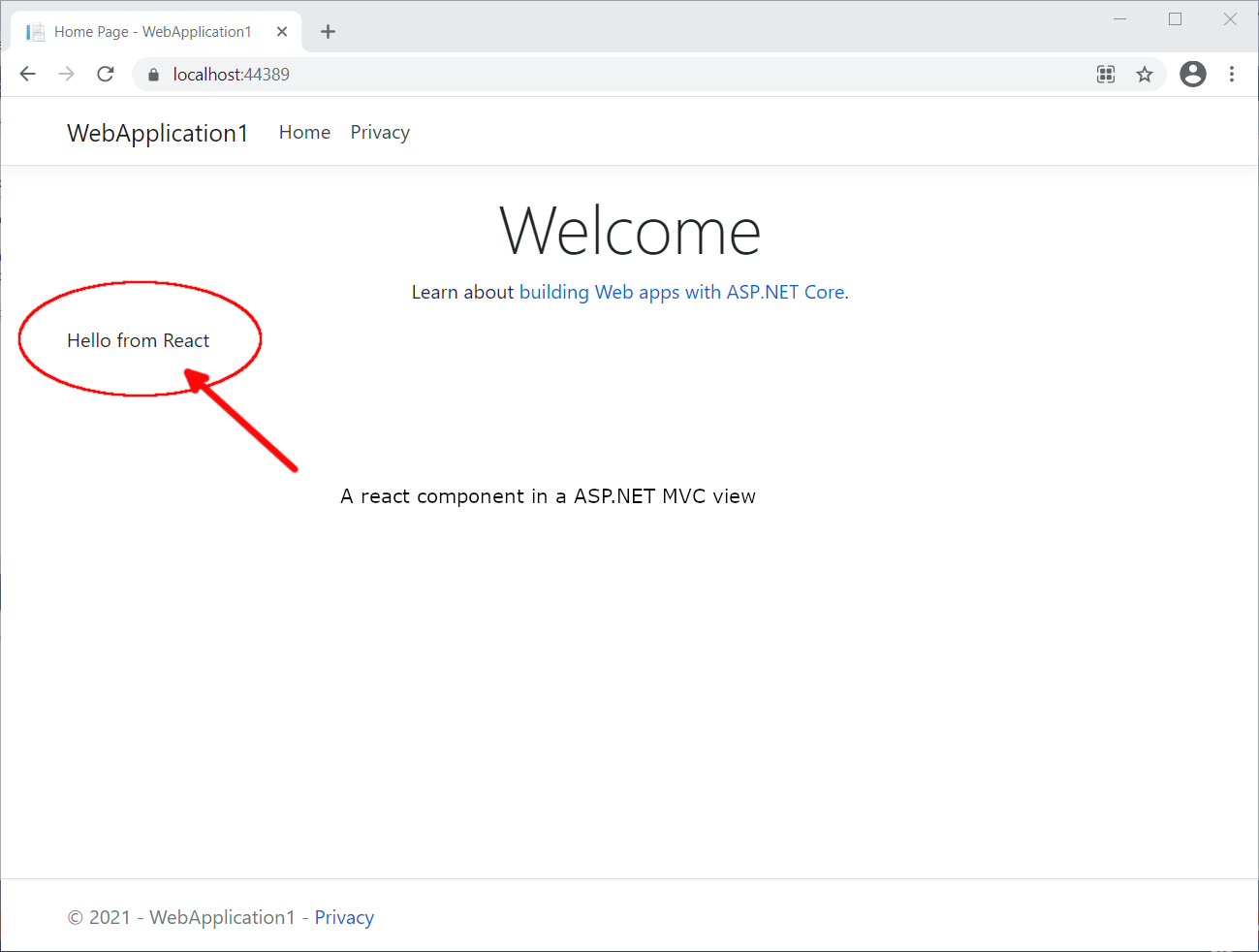
@section Scripts {

  <script src="~/dist/myui-bundle.js"></script>

}

3. Save the file.

4. Run the project. The home page now looks like this.



A React component working nicely and independent of your other MVC views. If you look at the \_Layout file you see it's also working on the same page as a loaded jquery and a loaded bootstrap.