Using Angular 2 with ASP.NET Core

## **Introduction**

### **Using Angular 2 with ASP.NET Core with TypeScript using Microsoft Visual Studio 2015**

This tutorial is aimed at beginners starting Angular 2 in ASP.NET Core using Visual Studio 2015. I have compiled the steps involved in starting to learn Angular 2. This is largely a step-by-step explanation, and you will feel much more confident at the end of the article.

## **Step 1: Creating an empty ASP.NET Core project**

Open Visual Studio 2015 Community Edition Update 3, Select New Web Project naming it “ngCoreContacts” and select “Empty” project template. Don’t forget to [install new web tools for ASP.NET Core 1.0](http://www.asp.net/core)

I used Visual Studio 2015 Community Edition Update 3, TypeScript 2.0, latest NPM, Gulp. For Visual Studio and TypeScript, I suggest that you use the same versions that I use in this guide.

## **Step 2: Configure ASP.NET Core to serve Static Files**

ASP.NET Core is designed as pluggable framework to include and use only necessary packages, instead of including too many modules initially.

Let's create an HTML file named index.html under wwwroot folder.

Right click on wwwroot folder, Add New Item and create the index.html file. This HTML page will act as default page.

1

2

3

4

5

6

7

8

9

<html>

<head>

<meta charset="utf-8" />

<title>Angular 2 with ASP.NET Core</title>

</head>

<body>

<h1>Demo of Angular 2 using ASP.NET Core with Visual Studio 2015</h1>

</body>

</html>

html

For ASP.NET Core to serve static files, we need to add StaticFiles middleware in the **Configure** method of the Startup.cs page. Ensure that packages are restored properly.

project.json is redesigned to make it better, we have StaticFiles middleware to serve static assets like HTML, JS files etc.

1

2

3

4

5

public void Configure(IApplicationBuilder app)

{

app.UseDefaultFiles();

app.UseStaticFiles();

}

C#

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

{

"dependencies": {

"Microsoft.NETCore.App": {

"version": "1.0.1",

"type": "platform"

},

"Microsoft.AspNetCore.Diagnostics": "1.0.0",

"Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",

"Microsoft.AspNetCore.Server.Kestrel": "1.0.1",

"Microsoft.Extensions.Logging.Console": "1.0.0",

"Microsoft.AspNetCore.StaticFiles": "1.0.0"

},

"tools": {

"Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final"

},

"frameworks": {

"netcoreapp1.0": {

"imports": ["dotnet5.6", "portable-net45+win8"]

}

},

"buildOptions": {

"emitEntryPoint": true,

"preserveCompilationContext": true,

"compile": {

"exclude": ["node\_modules"]

}

},

"runtimeOptions": {

"configProperties": {

"System.GC.Server": true

}

},

"publishOptions": {

"include": ["wwwroot", "web.config"]

},

"scripts": {

"postpublish": [

"dotnet publish-iis --publish-folder %publish:OutputPath% --framework %publish:FullTargetFramework%"

]

}

}

json

Interestingly Program.cs is an entry point of application execution just like main().

Run the application now. Notice that ASP.NET Core renders the static HTML page.

Delete this index.html page, we will be injecting this dynamically later. Till now you saw demonstration of “wwwroot“ as root folder for ASP.NET Core web apps.

## **Step 3: Angular 2 in ASP.NET Core**

Angular 2 has famously claimed to be the ONE Framework for mobile and desktop apps.

This tutorial refers to [5 MIN QUICK START](https://angular.io/docs/ts/latest/quickstart.html) for getting started. The 5-minute guide is more focused on other light-weight code editors. However, we are using Visual Studio 2015 Community Edition Update 3 for its built-in TypeScript tooling and other features.

We will be using Webpack as our module bundler. It’s an excellent alternative to the systemJS approach. To learn more about Webpack, read [Webpack and Angular 2](https://angular.io/docs/ts/latest/guide/webpack.html#!#configure-webpack)

The majority of Webpack scripting is based on AngularClass’s [angular2-webpack-starter](https://github.com/AngularClass/angular2-webpack-starter). I have modified it according to ASP.NET Core web apps.

ASP.NET Core follows Single Page Application (SPA) architecture rather than a Model-View-Controller (MVC) framework.

Why Webpack for Angular 2 in ASP.NET Core? I could have probably used [many other bundlers](https://www.slant.co/options/11602/alternatives/~webpack-alternatives).

However, Webpack does have many perks:

* It's much simpler to code, plugins based working with static files.
* Webpack 2 with dev-server runs application in memory, live reloading & compilation is easy. It provides tree shaking to eliminate unused code.
* Integrating with 3rd party packages like Angular Material 2, AngularFire, bootstrap, and more is short and sweet.
* The AngularClass webpack starter kit provides HMR (Hot Module Replacement) – maintain execution state while code gets modified.
* Webpack is adopted by Angular CLI, AngularService by Microsoft for making bundles much smaller.

## **Step 4: Adding NPM Configuration file for Angular 2 packages**

The Angular 2 team is pushing the code changes using NPM rather than CDN or any other source. Due to this, we need to add an NPM configuration file (package.json) to this ASP.NET Core application.

Right Click on “ngCoreContacts“, add new file “NPM Configuration File“; by default package.json is added to ASP.NET Core project. This acts Node Package Manager (NPM) file, a must-have for adding packages when using Angular 2.

From the Angular 2 Quick Start provided above, we need to add required dependencies. Copy-paste the below configuration in your package.json file

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

{

"version": "1.0.0",

"description": "ngcorecontacts",

"main": "wwwroot/index.html",

"scripts": {

"build:dev": "webpack --config config/webpack.dev.js --progress --profile",

"build:prod": "webpack --config config/webpack.prod.js --progress --profile --bail",

"build": "npm run build:dev",

"server:dev:hmr": "npm run server:dev -- --inline --hot",

"server:dev": "webpack-dev-server --config config/webpack.dev.js --progress --profile --watch --content-base clientsrc/",

"server:prod": "http-server dist --cors",

"server": "npm run server:dev",

"start:hmr": "npm run server:dev:hmr",

"start": "npm run server:dev",

"version": "npm run build",

"watch:dev:hmr": "npm run watch:dev -- --hot",

"watch:dev": "npm run build:dev -- --watch",

"watch:prod": "npm run build:prod -- --watch",

"watch:test": "npm run test -- --auto-watch --no-single-run",

"watch": "npm run watch:dev",

"webpack-dev-server": "webpack-dev-server",

"webpack": "webpack"

},

"dependencies": {

"@angular/common": "~2.0.1",

"@angular/compiler": "~2.0.1",

"@angular/core": "~2.0.1",

"@angular/forms": "~2.0.1",

"@angular/http": "~2.0.1",

"@angular/platform-browser": "~2.0.1",

"@angular/platform-browser-dynamic": "~2.0.1",

"@angular/router": "~3.0.1",

"@angular/upgrade": "~2.0.1",

"angular-in-memory-web-api": "~0.1.1",

"@angularclass/conventions-loader": "^1.0.2",

"@angularclass/hmr": "~1.2.0",

"@angularclass/hmr-loader": "~3.0.2",

"@angularclass/request-idle-callback": "^1.0.7",

"@angularclass/webpack-toolkit": "^1.3.3",

"assets-webpack-plugin": "^3.4.0",

"core-js": "^2.4.1",

"http-server": "^0.9.0",

"ie-shim": "^0.1.0",

"rxjs": "5.0.0-beta.12",

"zone.js": "~0.6.17",

"@angular/material": "^2.0.0-alpha.9",

"hammerjs": "^2.0.8"

},

"devDependencies": {

"@types/hammerjs": "^2.0.33",

"@types/jasmine": "^2.2.34",

"@types/node": "^6.0.38",

"@types/source-map": "^0.1.27",

"@types/uglify-js": "^2.0.27",

"@types/webpack": "^1.12.34",

"angular2-template-loader": "^0.5.0",

"awesome-typescript-loader": "^2.2.1",

"codelyzer": "~0.0.28",

"copy-webpack-plugin": "^3.0.1",

"clean-webpack-plugin": "^0.1.10",

"css-loader": "^0.25.0",

"exports-loader": "^0.6.3",

"expose-loader": "^0.7.1",

"file-loader": "^0.9.0",

"gh-pages": "^0.11.0",

"html-webpack-plugin": "^2.21.0",

"imports-loader": "^0.6.5",

"json-loader": "^0.5.4",

"parse5": "^1.3.2",

"phantomjs": "^2.1.7",

"raw-loader": "0.5.1",

"rimraf": "^2.5.2",

"source-map-loader": "^0.1.5",

"string-replace-loader": "1.0.5",

"style-loader": "^0.13.1",

"sass-loader": "^3.1.2",

"to-string-loader": "^1.1.4",

"ts-helpers": "1.1.1",

"ts-node": "^1.3.0",

"tslint": "3.15.1",

"tslint-loader": "^2.1.3",

"typedoc": "^0.4.5",

"typescript": "2.0.3",

"url-loader": "^0.5.7",

"webpack": "2.1.0-beta.22",

"webpack-dev-middleware": "^1.6.1",

"webpack-dev-server": "^2.1.0-beta.2",

"webpack-md5-hash": "^0.0.5",

"webpack-merge": "^0.14.1"

}

}

{

"version": "1.0.0",

"description": "ngcorecontacts",

"main": "wwwroot/index.html",

"scripts": {

"build:dev": "webpack --config config/webpack.dev.js --progress --profile",

"build:prod": "webpack --config config/webpack.prod.js --progress --profile --bail",

"build": "npm run build:dev",

"server:dev:hmr": "npm run server:dev -- --inline --hot",

"server:dev": "webpack-dev-server --config config/webpack.dev.js --progress --profile --watch --content-base clientsrc/",

"server:prod": "http-server dist --cors",

"server": "npm run server:dev",

"start:hmr": "npm run server:dev:hmr",

"start": "npm run server:dev",

"version": "npm run build",

"watch:dev:hmr": "npm run watch:dev -- --hot",

"watch:dev": "npm run build:dev -- --watch",

"watch:prod": "npm run build:prod -- --watch",

"watch:test": "npm run test -- --auto-watch --no-single-run",

"watch": "npm run watch:dev",

"webpack-dev-server": "webpack-dev-server",

"webpack": "webpack"

},

"dependencies": {

"@angular/common": "~2.0.1",

"@angular/compiler": "~2.0.1",

"@angular/core": "~2.0.1",

"@angular/forms": "~2.0.1",

"@angular/http": "~2.0.1",

"@angular/platform-browser": "~2.0.1",

"@angular/platform-browser-dynamic": "~2.0.1",

"@angular/router": "~3.0.1",

"@angular/upgrade": "~2.0.1",

"angular-in-memory-web-api": "~0.1.1",

"@angularclass/conventions-loader": "^1.0.2",

"@angularclass/hmr": "~1.2.0",

"@angularclass/hmr-loader": "~3.0.2",

"@angularclass/request-idle-callback": "^1.0.7",

"@angularclass/webpack-toolkit": "^1.3.3",

"assets-webpack-plugin": "^3.4.0",

"core-js": "^2.4.1",

"http-server": "^0.9.0",

"ie-shim": "^0.1.0",

"rxjs": "5.0.0-beta.12",

"zone.js": "~0.6.17",

"@angular/material": "^2.0.0-alpha.9",

"hammerjs": "^2.0.8"

},

"devDependencies": {

"@types/hammerjs": "^2.0.33",

"@types/jasmine": "^2.2.34",

"@types/node": "^6.0.38",

"@types/source-map": "^0.1.27",

"@types/uglify-js": "^2.0.27",

"@types/webpack": "^1.12.34",

"angular2-template-loader": "^0.5.0",

"awesome-typescript-loader": "^2.2.1",

"codelyzer": "~0.0.28",

"copy-webpack-plugin": "^3.0.1",

"clean-webpack-plugin": "^0.1.10",

"css-loader": "^0.25.0",

"exports-loader": "^0.6.3",

"expose-loader": "^0.7.1",

"file-loader": "^0.9.0",

"gh-pages": "^0.11.0",

"html-webpack-plugin": "^2.21.0",

"imports-loader": "^0.6.5",

"json-loader": "^0.5.4",

"parse5": "^1.3.2",

"phantomjs": "^2.1.7",

"raw-loader": "0.5.1",

"rimraf": "^2.5.2",

"source-map-loader": "^0.1.5",

"string-replace-loader": "1.0.5",

"style-loader": "^0.13.1",

"sass-loader": "^3.1.2",

"to-string-loader": "^1.1.4",

"ts-helpers": "1.1.1",

"ts-node": "^1.3.0",

"tslint": "3.15.1",

"tslint-loader": "^2.1.3",

"typedoc": "^0.4.5",

"typescript": "2.0.3",

"url-loader": "^0.5.7",

"webpack": "2.1.0-beta.22",

"webpack-dev-middleware": "^1.6.1",

"webpack-dev-server": "^2.1.0-beta.2",

"webpack-md5-hash": "^0.0.5",

"webpack-merge": "^0.14.1"

}

}

JSON

Right after saving this, ASP.NET Core starts restoring the packages. It will download all packages mentioned in the dependencies section of the package.json file.

The Visual Studio Solution Explorer might read ‘Dependencies – not installed’. Don’t worry; all the npm packages have been installed. This is simply a bug in the tooling.

## **Step 5: Add the TypeScript configuration file.**

You must do this for Angular 2 in ASP.NET Core apps that use TypeScript. The TypeScript Configuration file handles [transpiling our application](https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them) to JavaScript, loads modules, and meets ES5 standards.

Refer to [Getting Started with TypeScript](http://www.mithunvp.com/typescript-in-asp-net-5-using-visual-studio-2015/) if want a beginner intro on TypeScript configuration.

Add tsconfig.json in the project, and add in the configuration below.

“baseUrl” ensures that TypeScript files are transpiled to JavaScript from the ‘./clientsrc‘. This folder is virtual directory for TypeScript

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

{

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"moduleResolution": "node",

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"allowSyntheticDefaultImports": true,

"sourceMap": true,

"noEmitHelpers": true,

"strictNullChecks": false,

"baseUrl": "./clientsrc",

"paths": [],

"lib": [

"dom",

"es6"

],

"types": [

"hammerjs",

"node",

"source-map",

"uglify-js",

"webpack"

]

},

"exclude": [

"node\_modules",

"dist"

],

"awesomeTypescriptLoaderOptions": {

"forkChecker": true,

"useWebpackText": true

},

"compileOnSave": false,

"buildOnSave": false,

"atom": { "rewriteTsconfig": false }

{

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"moduleResolution": "node",

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"allowSyntheticDefaultImports": true,

"sourceMap": true,

"noEmitHelpers": true,

"strictNullChecks": false,

"baseUrl": "./clientsrc",

"paths": [],

"lib": [

"dom",

"es6"

],

"types": [

"hammerjs",

"node",

"source-map",

"uglify-js",

"webpack"

]

},

"exclude": [

"node\_modules",

"dist"

],

"awesomeTypescriptLoaderOptions": {

"forkChecker": true,

"useWebpackText": true

},

"compileOnSave": false,

"buildOnSave": false,

"atom": { "rewriteTsconfig": false }

}

JSON

##### It’s mandatory to install TypeScript 2.0 when working with Angular 2.

At present typings.json is not required because we are using [@types with TypeScript.](https://blogs.msdn.microsoft.com/typescript/2016/06/15/the-future-of-declaration-files/) However if your using any other packages which don’t have entries in @types then **typings.json** has to be added.

## **Step 6: Usage of Webpack as module bundler**

### **What is Webpack?**

Webpack is a powerful module bundler. A bundle is a JavaScript file that incorporate assets that belong together and should be served to the client in a response to a single file request. A bundle can include JavaScript, CSS, HTML, and almost any other language.

Webpack roams over your application source code, looking for import statements, building a dependency graph, and emitting one (or more) bundles. With plugin “loaders” Webpack can preprocess and minify different non-JavaScript files such as TypeScript, SASS, and LESS files.

In package.json, we have added “webpack“ packages as “devdependencies“. They will perform all bundling work.

webpack.config.js defines what Webpack does. As always, the applications are run in **Development**, **Test** and **Production** environment.

There are some common functionalities and some specific to environments. We will focus on the development and production environment.

The development environment should have source maps for debugging TypeScript files. However, minifying bundles of JS, CSS, etc. is not required. The production environment should minify bundles to reduce loading time. Webpack 2 also does tree-shaking, a method for eliminating unused code to shrink bundle sizes.

*The entire source code is on my*[*github repo*](https://www.github.com/syedikramshah)*, fork or clone to play with it.*

webpack.config.js – Based on environment set process.env.NODE\_ENV, it runs either dev or prod configurations.

Webpack.common.js before bundling environment specific files, it performs tasks meant to be used for both environment.

Webpack splits Angular 2 apps into 3 files polyfills (to maintain backward compatibility with older browsers), vendors (all JS, CSS, HTML, SCSS, images, JSON etc into one file) and boot (application specific files). This split is resolved based on various file extensions when Webpack itself doesn’t know what to do with a non-JavaScript file.

We teach it to process such files into JavaScript using loaders. For this, we have written loaders for TS, HTML, JSON, fonts, images, and more. Any static assets placed in clientsrc/assets will be copied to assets folder using CopyWebpackPlugin. CleanWebpackPlugin cleans wwwroot/dist folder every time we run it, so that we get fresh set of files.

I told you above to delete the index.html file, now the clientsrc/index.html will be moved to wwwroot using HtmlWebpackPlugin. Plus Webpack injects the bundle files i.e. polyfills, vendor, boot JS files and includes them in HTML script reference. Now let’s see webpack.dev.js for development purpose

Running webpack-dev-server – this runs the entire application in memory. Any changes to source file gets applied immediately. Loads application in debug mode with source map. Runs the application on localhost 3000 port. (The port can be changed as your convenience.)

Now let’s take a look at what webpack.prod.js does:

* Merges all the bundle files and copies to wwwroot.
* Minifies all files to load faster using UglifyJsPlugin.

Hopefully this clarifies some of Webpack's functionalities.

## **Step 7: Writing the Angular 2 application**

Until now we created ASP.NET Core app, added TSconfig file, webpack configuration. Now it’s time to write the Angular 2 application.

In the github repo, you can see the “clientsrc” folder. This contains the Angular 2 app which gets bundled into using webpack configurations we wrote

“Clientsrc“ folder has index.html, polyfills.browses.ts, vendor.browsers.ts and mostly importantly boot.ts.

We have app folder containing HTML, Angular 2 components and root level module (app.module.ts) which gets loaded while bootstrapping application.

Some of files might be not interesting now, will focus them in separate articles later.

## **Step 8: Running the application**

Before running make sure you have run command npm install. This might not be needed but still it will ensure all packages are installed.

##### Now let’s run the application in development mode

From command line (directory should be same as package.json), type npm start & hit enter. It will start running the webpack-dev-server which loads application and listens on localhost:3000. When the console reads bundle is now VALID, open a browser and navigate to http://localhost:3000 to see the loaded application Notice wwwroot folder, we don’t see any files copied because everything is running in memory.

Now that the application runs properly on browser, let’s understand how Angular 2 app loads.

When the browser starts rendering the index.html page, it encounters the Loading… tag. Then Angular’s module platformBrowserDynamic bootstraps clientsrc/app/AppModule through line platformBrowserDynamic().bootstrapModule(AppModule).

AppModule then loads the component app.component.ts which is mentioned in @NgModule as bootstrap entry. Clientsrc/src/Appcomponent then resolves the <my-app> tag as a selector in it and renders the UI with TypeScript code.

When we enter npm start in console to run the application, execution points scripts section of package.json to below code

1

webpack-dev-server --config config/webpack.dev.js --progress --profile --watch --content-base clientsrc/

bash

This invokes webpack-dev-server, runs the development config, and watches for any changes in clientsrc folder. Any changes in this folder will reload the application with changes.

*Here ASP.NET Core feels just like an HTML-based web app, so run using npm start to use AngularClass features of reloading, using webpack, Hot module replacement feature.*

Running the application in Production mode

Assuming the application is now ready to deployed, we need to have the PROD build. For this run command

1

2

//builds app and copies in wwwroot

npm run build:prod

bash

Now if you see wwwroot folder, we see the HTML, JS bundle files. This wwwroot folder can be deployed on any web server like IIS or nginx.

You can either do F5 to run from Visual Studio IDE or run command **npm run server:prod**

## **Conclusion**

Hopefully this tutorial helped you create your Angular 2 app using Visual Studio. Thank you for reading!