Configuring ASP.NET Core

Web Api Project with Angular 2

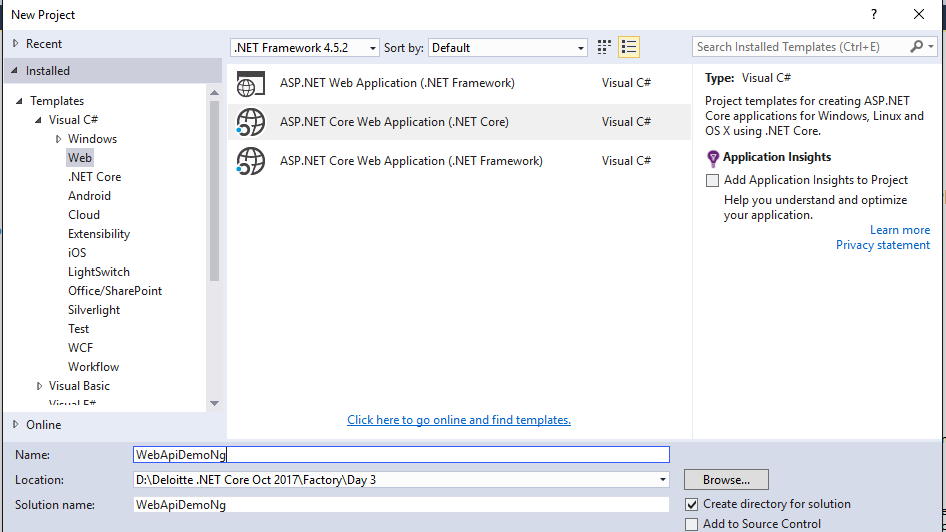
# Setup NodeJS and NPM

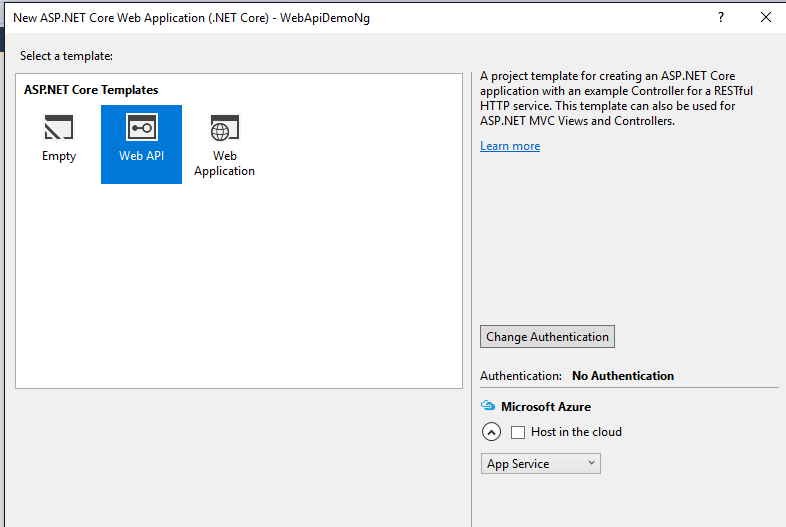
Download the Install the latest version of NodeJS and NPM

# Setup TypeScript for Visual Studio 2015

<https://www.microsoft.com/en-us/download/details.aspx?id=48593>

# Setting up the Project





A Web API project can't serve static files like JavaScripts, CSS styles, images, or even HTML files. Therefore we need to add a reference to Microsoft.AspNetCore.StaticFiles in the project.json:

"Microsoft.AspNetCore.StaticFiles": "1.0.0 ",

And in the startup.cs, we need to add the following line, just before the call of `UseMvc()

app.UseStaticFiles();

## Providing Support for Angular2 Routing in Startup

Another important thing we need to do in the startup.cs, is to support the Routing of Angular2. If the Browser calls a URL which doesn't exists on the server, it could be a Angular route. Especially if the URL doesn't contain a file extension.

This means we need to handle the 404 error, which will occur in such cases. We need to serve the index.html to the client, if there was an 404 error, on requests without extensions. To do this we just need a simple lambda based MiddleWare, just before we call UseStaticFiles():

app.Use(async (context, next) =>

{

await next();

if (context.Response.StatusCode == 404

&& !Path.HasExtension(context.Request.Path.Value))

{

context.Request.Path = "/index.html";

await next();

}

});

## Editing launchSettings.Json

Inside the properties folder we'll find a file called launchSettings.json. This file is used to configure the behavior of visual Studio 2015, when we press F5 to run the application. Remove all strings "api/values" from this file. Because Visual Studio will always call that specific Web API, every time you press F5.

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:33831/",

"sslPort": 0

}

},

"profiles": {

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"WebApiDemoNg": {

"commandName": "Project",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

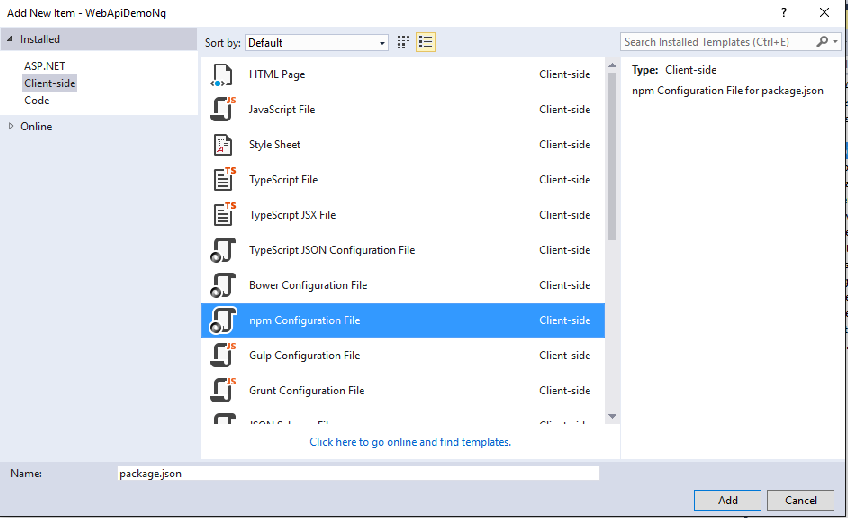
}

}

}

# Configuring Angular2

## Adding package.json



package.json

{

"name": "angular-quickstart",

"version": "1.0.0",

"scripts": {

"start": "tsc && concurrently \"tsc -w\" \"lite-server\" ",

"lite": "lite-server",

"postinstall": "typings install && gulp restore",

"tsc": "tsc",

"tsc:w": "tsc -w",

"typings": "typings"

},

"licenses": [

{

"type": "MIT",

"url": "https://github.com/angular/angular.io/blob/master/LICENSE"

}

],

"dependencies": {

"@angular/common": "2.0.2",

"@angular/compiler": "2.0.2",

"@angular/core": "2.0.2",

"@angular/forms": "2.0.2",

"@angular/http": "2.0.2",

"@angular/platform-browser": "2.0.2",

"@angular/platform-browser-dynamic": "2.0.2",

"@angular/router": "3.0.2",

"@angular/upgrade": "2.0.2",

"angular-in-memory-web-api": "0.1.5",

"bootstrap": "3.3.7",

"core-js": "2.4.1",

"reflect-metadata": "0.1.8",

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

"systemjs": "0.19.39",

"zone.js": "0.6.25"

},

"devDependencies": {

"concurrently": "3.0.0",

"lite-server": "2.2.2",

"gulp": "^3.9.1",

"typescript": "2.0.3",

"typings": "1.4.0"

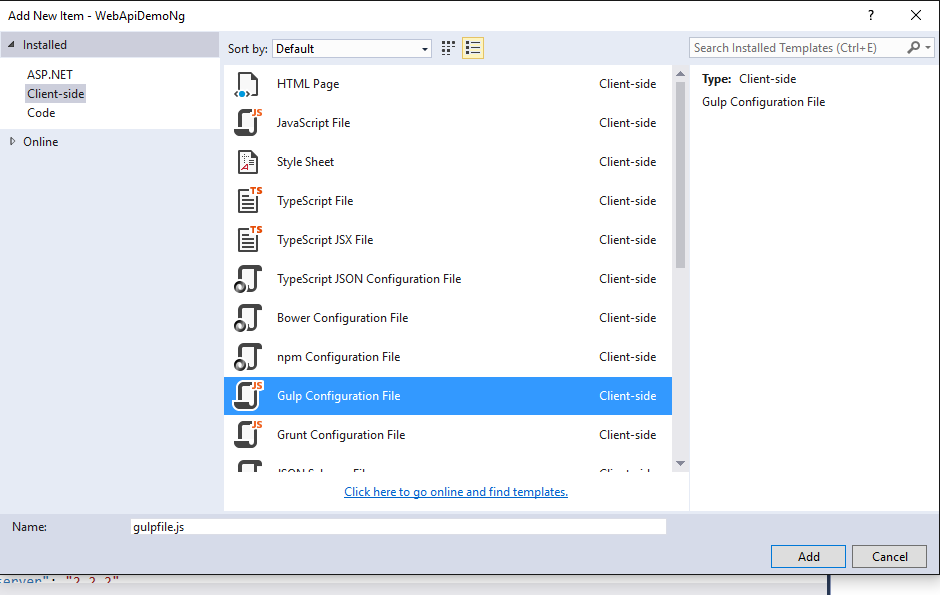
}

}

After the file is saved, Visual Studio tryies to load all the packages. This works, but VS shows a yellow exclemation mark because of any arror. Until yet, I didn't figure out what is going wrong here. To be sure all packages are propery installed, use the console, change directory to the current project and type npm install

The post install will possibly fail because gulp is not yet configured. We need gulp to copy the dependencies to the right location inside the wwwroot folder, because static files will only be loaded from that location. This is not part of the tutorial on angular.io, but is needed to fit the client stuff into Visual Studio. Using Visual Studio we need to create a new "gulp Configuration file" with the name gulpfile.js:

## Configuring Gulp Task Runner - Gulpfile.js



var gulp = require('gulp');

var libs = './wwwroot/libs/';

gulp.task('default', function () {

// place code for your default task here

});

gulp.task('restore:core-js', function () {

gulp.src([

'node\_modules/core-js/client/\*.js'

]).pipe(gulp.dest(libs + 'core-js'));

});

gulp.task('restore:zone.js', function () {

gulp.src([

'node\_modules/zone.js/dist/\*.js'

]).pipe(gulp.dest(libs + 'zone.js'));

});

gulp.task('restore:reflect-metadata', function () {

gulp.src([

'node\_modules/reflect-metadata/reflect.js'

]).pipe(gulp.dest(libs + 'reflect-metadata'));

});

gulp.task('restore:systemjs', function () {

gulp.src([

'node\_modules/systemjs/dist/\*.js'

]).pipe(gulp.dest(libs + 'systemjs'));

});

gulp.task('restore:rxjs', function () {

gulp.src([

'node\_modules/rxjs/\*\*/\*.js'

]).pipe(gulp.dest(libs + 'rxjs'));

});

gulp.task('restore:angular-in-memory-web-api', function () {

gulp.src([

'node\_modules/angular-in-memory-web-api/\*\*/\*.js'

]).pipe(gulp.dest(libs + 'angular-in-memory-web-api'));

});

gulp.task('restore:angular', function () {

gulp.src([

'node\_modules/@angular/\*\*/\*.js'

]).pipe(gulp.dest(libs + '@angular'));

});

gulp.task('restore:bootstrap', function () {

gulp.src([

'node\_modules/bootstrap/dist/\*\*/\*.\*'

]).pipe(gulp.dest(libs + 'bootstrap'));

});

gulp.task('restore', [

'restore:core-js',

'restore:zone.js',

'restore:reflect-metadata',

'restore:systemjs',

'restore:rxjs',

'restore:angular-in-memory-web-api',

'restore:angular',

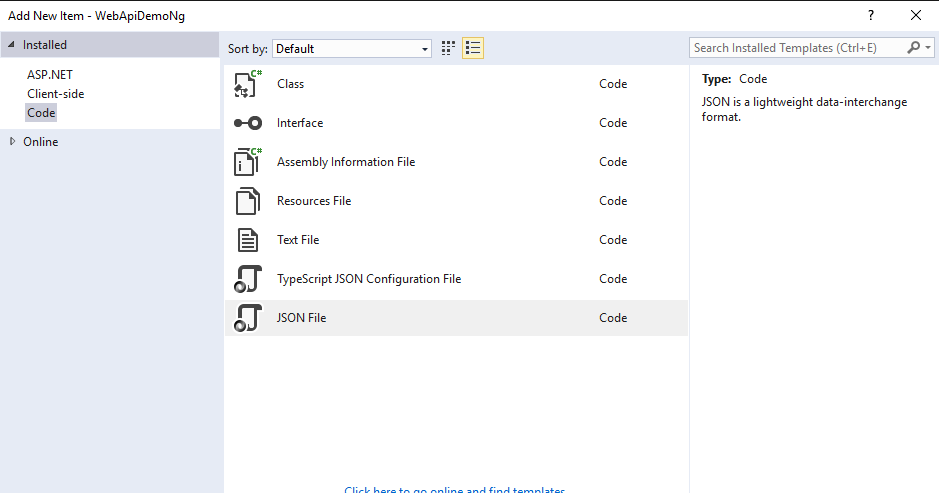
'restore:bootstrap'

]);

The task restore, copies all the needed files to the Folder ./wwwroot/libs

## Configuring Typings

TypeScript needs some type definitions to get the types and API definitions of the libraries, which are not written in TypeScript or not available in TypeScript. To load this, we use another tool, called "typings". This is already installed with NPM. This tool is a package manager for type definition files. We need to configure this tool with a typings.json



### Typings.json

{

"globalDependencies": {

"core-js": "registry:dt/core-js#0.0.0+20160725163759",

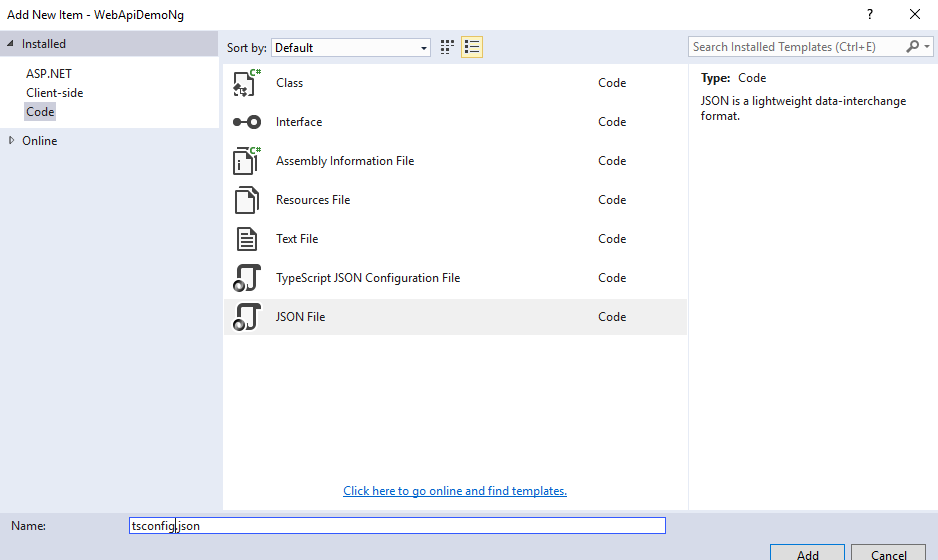
"jasmine": "registry:dt/jasmine#2.2.0+20160621224255",

"node": "registry:dt/node#6.0.0+20160909174046"

}

}

### tsconfig.json



**Tsconfig.json**

{

"compileOnSave": true,

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"moduleResolution": "node",

"sourceMap": true,

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"removeComments": false,

"noImplicitAny": false

},

"exclude": [

"node\_modules"

]

}

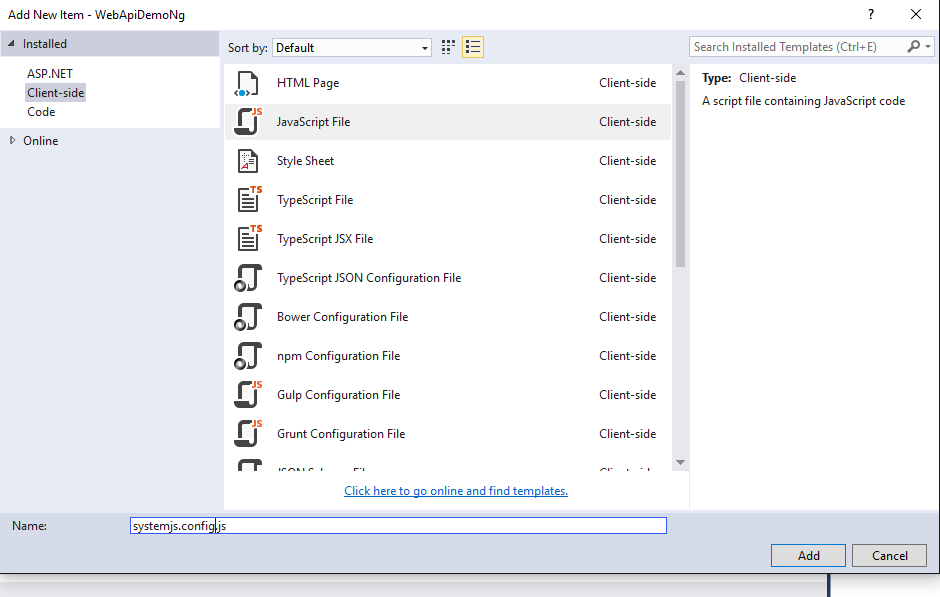
# Preparing for the first AngularJS App

## Index.html

The first step is to create a index.html in the folder wwwroot

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Angular QuickStart</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="styles.css">  <!-- 1. Load libraries -->  <!-- Polyfill(s) for older browsers -->  <script src="libs/core-js/shim.min.js"></script>  <script src="libs/zone.js/zone.js"></script>  <script src="libs/reflect-metadata/Reflect.js"></script>  <script src="libs/systemjs/system.src.js"></script>  <!-- 2. Configure SystemJS -->  <script src="systemjs.config.js"></script>  <script>  System.import('app').catch(function (err) { console.error(err); });  </script>  </head>  <!-- 3. Display the application -->  <body>  <my-app>Loading...</my-app>  </body>  </html> |

## SystemJS.Config.js

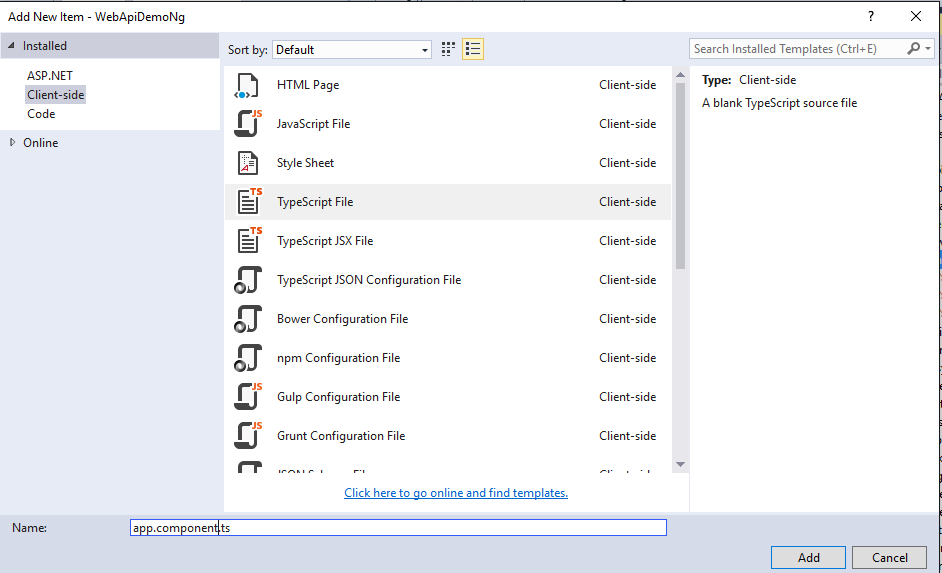


|  |
| --- |
| /\*\*  \* System configuration for Angular samples  \* Adjust as necessary for your application needs.  \*/  (function (global) {  System.config({  paths: {  // paths serve as alias  'npm:': 'libs/'  },  // map tells the System loader where to look for things  map: {  // our app is within the app folder  app: 'app',  // angular bundles  '@angular/core': 'npm:@angular/core/bundles/core.umd.js',  '@angular/common': 'npm:@angular/common/bundles/common.umd.js',  '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',  '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',  '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',  '@angular/http': 'npm:@angular/http/bundles/http.umd.js',  '@angular/router': 'npm:@angular/router/bundles/router.umd.js',  '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',  // other libraries  'rxjs': 'npm:rxjs',  'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',  },  // packages tells the System loader how to load when no filename and/or no extension  packages: {  app: {  main: './main.js',  defaultExtension: 'js'  },  rxjs: {  defaultExtension: 'js'  },  'angular-in-memory-web-api': {  main: './index.js',  defaultExtension: 'js'  }  }  });  })(this); |

This file also defines a main entry point which is a main.js. This file is the transpiled TypeScript file main.ts we need to create in the next step.

# Coding the AngularJS App

## app.component.ts



|  |
| --- |
| import { Component } from '@angular/core';  @Component({  selector: 'my-app',  template: '<h1>My First Angular App</h1>'  })  export class AppComponent { } |

## app.module.ts

|  |
| --- |
| import { NgModule } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { AppComponent } from './app.component';  @NgModule({  imports: [BrowserModule],  declarations: [AppComponent],  bootstrap: [AppComponent]  })  export class AppModule { } |

## main.ts

|  |
| --- |
| import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  import { AppModule } from './app.module';  const platform = platformBrowserDynamic();  platform.bootstrapModule(AppModule); |