Developing an Angular2 app with Visual Studio

In the following tutorial we will create a development environment for Angular2 applications on a windows machine using Visual Studio 2015 as an IDE. As a side note I would highly recommend working with open source technologies with a linux based machine so if possible I would recommend developing Angular2 on an Ubuntu or MacOS X.
However if you still insist on using windows, this guide will help you get started with windows 10 and Visual Studio 2015

Install Visual Studio 2015

Go to the following link and download the Visual Studio (**VS**) community edition.
After you finished downloading install **VS** , when asked **choose the type of installation** you can choose to install the **default**.
While installing **VS** you can continue to install **NodeJS**
After installation is done launch **VS**

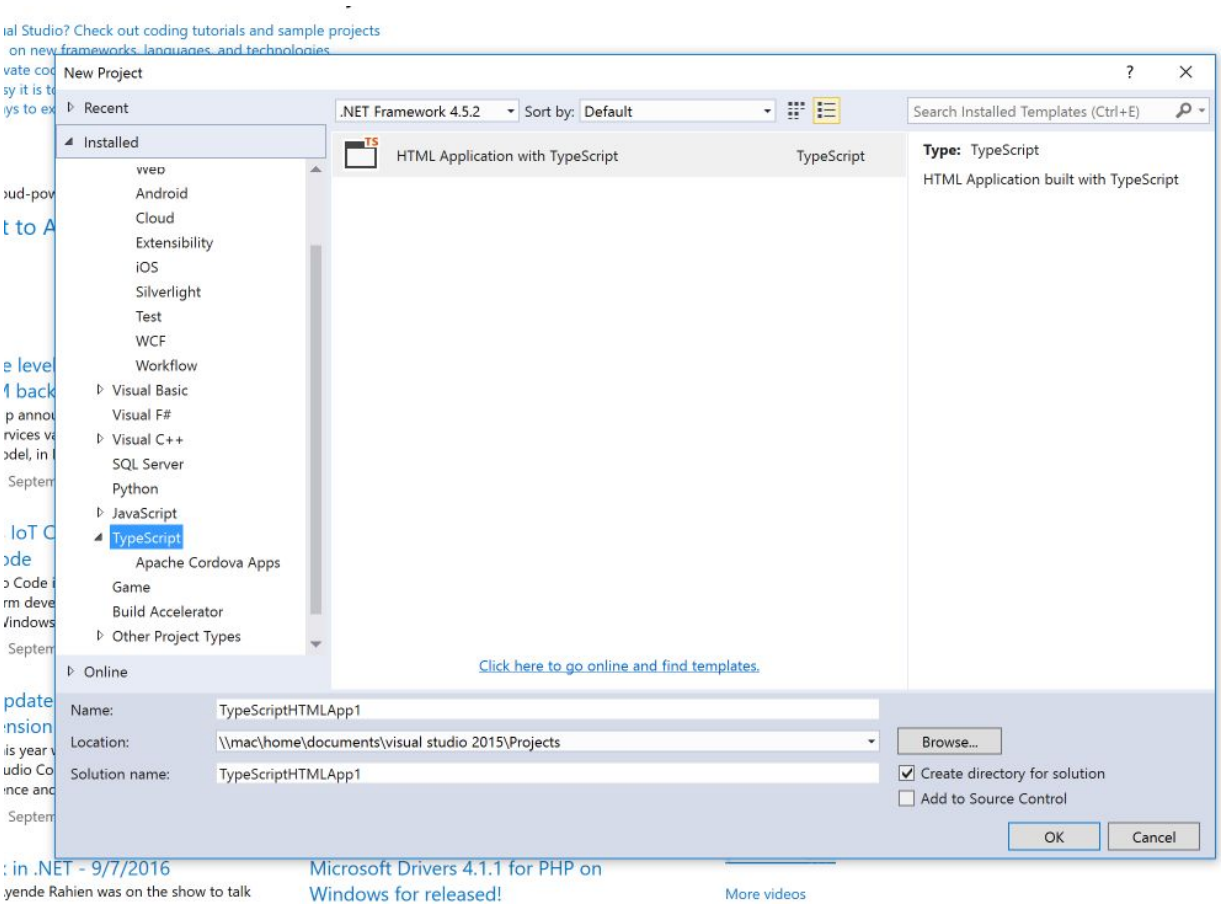
Install NodeJS

We will need to install NodeJs and NPM on our windows machine so do the following steps:

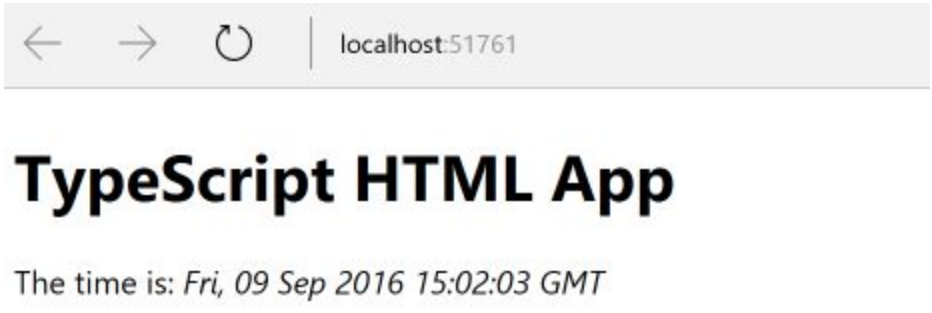
- Go to the following link and download **NodeJs** version 6.5.0.
- Install the file you downloaded.
- Verify that node is installed correctly by going to **CMD** - open **CMD** by clicking the **window start** button and typing **CMD** . after **CMD** is open type:
  > node -v
  You should see the version of node that is currently installed. Also type:
  > npm -v
  You should see the version of **npm** that is installed

VS - New Project

After you launched you **VS** start a new project.
You should see a window opens with the project templates to choose from. In the list of templates you should see a template named: **HTML Application with TypeScript.**
Choose this template and in the name of the project type: **HelloWorld**



After **VS** finished preparing your project run it in the browser
You should see the following screen:



Great Job! You just finished launching your project template, now let's install webpack

Installing WebPack

Webpack is used as a module bundler, our web application is made out of many files:

Style files (css, less or sass), image files, font files, TypeScript files etc.
WebPack is our tool for dealing with all our files and creating the final files for our project.
Among the things webpack can do for us:

- Compile our TypeScript files
- Compile our SCSS files
- Embed images as Base64
- Embed fonts as Base64
- Seperate our code to chunks, for example combining all the third party packages we use to a single vendor file and uploading it to CDN
- And more…

Webpack is a must tool for every modern web developer.
We will now install webpack in our project.
Complete the following steps:

- Open your **CMD**
- **cd** into you root project folder and then cd into **HelloWorld** folder inside your root project folder, (at the end of your path there should be 2 directories names **HelloWorld**)
- In your **CMD** type: **> npm init**
- When asked type the name: **helloworld** and answer the rest of the questions as the default answer
- If all went well you should see that a **package.json** file is created.
- To install **webpack** simply type in your **CMD:**
  > npm install webpack --save-dev

Webpack is now installed but we still need to configure it. So this will be our next step

Configuring Webpack

In order to configure **webpack** we need to place a **JS** file called: **webpack.config.js**
Inside you project create that file and place the following code:

```
/**
 * Configuration file for webpack
 *
 * Created September 9th, 2016
 * @author: ywarezk
 * @version: 1.0
 * @copyright: Nerdeez Ltd
 */
var path = require('path');
module.exports = {
        entry: './src/app.module.ts',
        output: {
```

```
    filename: 'app.js',
    path: path.resolve(__dirname, 'dist')
  },
  module: {
    loaders: [
      {test: /\.ts$/, loader: 'ts-loader'}
    ]
  }
}
```

In the following file we are telling webpack to do the following:

- The Entry point file, the file you are compiling is called **app.module.ts** and it is placed in the relative directory of **src**
- The compilation should output a file called **app.js** and that file should be located at the **dist** directory.
- We added a loader to deal with the TypeScript files called **ts-loader** and we are activating this loader for every file webpack stumbles on with an ending of **.ts**

Before we can test our webpack is installed properly Let's install our loader and TypeScript

Install loaders & Test Webpack

We will now install **typescript** and **ts-loader** which **webpack** requires. In your **CMD** type the following:
> npm install typescript ts-loader --save-dev

When we started our **TypeScript** project, **VS** automatically created a file called **app.ts**. Webpack is searching the **entry** file in the **src** folder and it's looking for a file called **app.module.ts** so let's move the file **app.ts** to the proper place and also rename that file.

Before we are ready to see if our configuration works we will want the **CMD** commands to search also in the local **node_modules/.bin** folder so do the following:

- Hit the Start button and type: **system**
- On the left click **advanced system settings**
- On the next screen click on **Environment Variables…**
- In the **System Variables** click on **Path** and click **Edit**
- Add a new entry add the bottom with the following value: **.\node_modules\.bin**

This will make sure when we type commands our cmd will also look for them in the local **node_modules** folder.

Since we are using TypeScript we will need to add to our project root the typescript configuration file named: **tsconfig.json** we will create the file and put the following configuration:

```
{
  "compileOnSave": false,
  "compilerOptions": {
        "emitDecoratorMetadata": true,
        "experimentalDecorators": true
  }
}
```

Now open a new **CMD** window and cd back to the folder of the project.
Now in that folder type: **webpack** .
You should see a green indicators and if all went well then you should see a **dist** folder and the app.js file created for us.

Test our app

When we created the project, **VS** automatically added an **index.html** file, modify the file so it will load the **app.js** file created in the **dist** folder:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="utf-8" />
        <title>TypeScript HTML App</title>
        <link rel="stylesheet" href="app.css" type="text/css" />
        <script src="dist/app.js"></script>
</head>
<body>
        <h1>TypeScript HTML App</h1>
        <div id="content"></div>
</body>
</html>
```

If you launch your app now from **VS** you should see same screen when you first created the project.
Now that we are handling the compile from webpack you can go ahead and delete the **JS** files that **VS** automatically created for you in the **src** folder.

Typings

Currently if we try to import into our app **angular** libraries, we will see a lot of errors from the **TypeScript compiler**
error TS2304: Cannot find name 'Map'.

The reason those errors appear is because angular is using certain libraries that **TypeScript** doesn't recognize, to introduce to **TypeScript** certain libraries that are not developed using **TypeScript** we can use **Typings**. The community prepared interfaces to libraries that are unknown to **TypeScript** and to install those interfaces we use a tool called **Typings**.

To install **typings** type in your **cmd** at the root project folder:
> npm install typings --save-dev

This will install typings and place the package reference inside **package.json** in the **dev** dependencies.

Working with typings will look very familiar, it's very similar to working with **NPM**. let's init **typings** in the root project type the following:
> typings init

You should see that the command created a file named: **typings.json**
Let's install the interface for a library angular needs called **core-js**. In the **CMD** type the following:
> typings install dt~core-js --save --global
> typings install dt~require --save --global

If this command succeeded you should see a folder called **typings** is created. And now if you import angular libraries there won't be any errors.

Hello World

Finally we are ready to develop our hello world app, so let's install the packages we need.
In your **CMD** type the following:
> npm install @angular/core zone.js rxjs@5.0.0-beta.12 --save
> npm install @angular/platform-browser-dynamic @angular/common @angular/compiler @angular/core @angular/platform-browser --save
> npm install reflect-metadata --save

Now inside the file **src/app.module.ts** type the following:

```
/**
 * Root module & bootstrap
 *
 * Created September 10th, 2016
 * @version: 1.0
 * @copyright: Nerdeez Ltd
 * @author: ywarezk
 */
require('zone.js');
require('reflect-metadata/Reflect');
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
import {AppComponent} from './app.component.ts';
@NgModule({
        imports: [BrowserModule],
        declarations: [AppComponent],
        exports: [],
        bootstrap: [AppComponent]
})
class AppModule { }
platformBrowserDynamic().bootstrapModule(AppModule);
```

In this file we are creating the **Root Module** of our app, this module will be bootstrapped to the **DOM**

Create another file next to the **app.module.ts** that is called: **app.component.ts**. That file looks like this:

```
/**
 * root component
 *
 * Created September 10th, 2016
 * @author: ywarezk
 * @version: 1.0
 * @copyright: Nerdeez Ltd
 */
import {Component} from '@angular/core';
@Component({
        selector: 'app',
        template:
        `
        <h1>Hello world</h1>
```

```
    `
})
export class AppComponent { }
```

We define here our **Root Component** which doesn't do much except print an **Hello World** message on the screen.

Change our **index.html** file to look like this:

```
<!--
    Index file for our hello world angular2 app

    Created September 10th, 2016
    @author: ywarezk
    @version: 1.0
    @copyright: Nerdeez Ltd
-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Hello World</title>
</head>
<body>
    <app></app>
    <script src="dist/app.js"></script>
</body>
</html>
```

This will load our destination compiled by bootstrap **app.js** bundled script. We are also placing the **selector** we choose for the **AppComponent**.
Launch your app and you should see the hello world message.