**Angular 2— Setting up your working environment (updated: using Beta 15)**

I am assuming that you already have *NPM (Node.js Package Manager)*installed on your computer (because you will need it - you can find more information on the [official web site](https://docs.npmjs.com/getting-started/installing-node)). It is important to double check that you have a recent version of Node.js (executing ***node --version***), something like 4.2+. I am using Windows to make this work.

**Related Links**:

. [Angular 2 — Our first component](https://medium.com/@menezes.carlos/angular-2-our-first-component-5f9ddbca4fe7#.7yellce24)

**Steps:**

1. Install [Visual Studio Code](https://www.visualstudio.com/en-us/products/code-vs.aspx) (IDE, but you can use pretty much anything you want; pick your favourite IDE: Atom, Webstorm…all of them have good support for TypeScript)
2. Install TypeScript
3. Create tsconfig.json
4. Create typings.json
5. Create package.json
6. Install Angular 2
7. Off you go! You are ready to write your first Angular 2 app.

**1) Installing Visual Studio Code**

As I mentioned before, you can use pretty much any IDE but my preference at this stage is Visual Studio Code.

**2) Installing TypeScript**

We are going to write our app in **TypeScript**, so we will have to install it via **npm**: (**-g** means that it will install it globally). As I said I am using Windows, so to run the ***npm*** you will have to run the cmd (command prompt). Go to your cmd and execute this command.

npm install -g typescript

*TypeScript is not required to write Angular 2, but I have chosen this option because of the type system it offers and if you are coming from C# it is really easy to get used to it, you can pretty much write your Angular 2 app using normal ECMAScript 5 (standard JavaScript where most of the apps are written), but I guarantee you that it will look really ugly!*

*Now we have installed typescript globally (you can see it here: %AppData%/npm/node\_modules/typescript), the next step is to create a new empty folder for our test app and use tsc (stands for TypeScript Compiler) from that new folder to start a project .tsc. Create a new folder whenever you want and open it from the Visual Studio Code. (File > Open Folder > Select Folder)*

**3) Creating tsconfig.json**

**What is the tsconfig.json file?**

**Answer**: <https://github.com/Microsoft/TypeScript/wiki/tsconfig.json>

…Basically it is a file in a directory that indicates that the directory is the root of a TypeScript project. We can compare it as a project solution (.sln) in a MVC project for example.

Make sure you run this command in your new folder directory using **cmd** (command prompt), because it will create for us the file *tsconfig.json* (after executing the command) that stores the TypeScript compilation options. You can see the *tsconfig.json* from the Visual Studio Code when created.

tsc --init --target es5 --sourceMap --experimentalDecorators --emitDecoratorMetadata

**Explaining the flags:**

***target es5****: Specify that we want to get our TypeScript code compiled to ECMAScript 5 and we can run our code in older browsers.****sourceMap****: Allows us to generate the source maps, i.e files that contain mappings between the generated ES5 code and the original TypeScript code.****experimentalDecorators & emitDecoratorMetada:****We need those flags to compile, transpile to ECMAScript 5 as we are using TypeScript with decorators (to annotate and modify classes and properties at design time, it basically says things like — ‘this is a custom property’ or ‘this class is a component (@Component, decorator)).*

The *tsconfig.json* file contains TypeScript compiler configuration parameters as we said previously and the file looks like this (after being generated):

{  
 "compilerOptions": {  
 "**target**": "es5",  
 "**sourceMap**": true,  
 "**experimentalDecorators**": true,  
 "**emitDecoratorMetadata**": true,  
 "**module**": "commonjs",  
 "**noImplicitAny**": false,  
 "**removeComments**": true,  
 "**moduleResolution**": "node",  
 "**outDir**": "built",  
 "**rootDir**": "."  
 },  
 "exclude": [  
 "node\_modules",  
 "typings/main",  
 "typings/main.d.ts"   
 ]  
}

As you can see above, the structure consists of two sections. The first section is named *compilerOptions* and contains command line options which are passed to the TypeScript compiler every time the program is executed. The second section *exclude* is an array of folders and files in the project directory which should not be processed by the TypeScript compiler and it makes sense to exclude the *node\_modules* directory.

**4) Creating typings.json**

Add **typings.json** file to the project folder and copy/paste the following:

{   
 "**ambientDependencies**":   
 {   
 "es6-shim": "github:DefinitelyTyped/DefinitelyTyped/es6-shim/es6-shim.d.ts#6697d6f7dadbf5773cb40ecda35a76027e0783b2"   
 }  
}

**5) Creating package.json**

To add Angular 2 library into our project we are going to download it using NPM; which will help us with dependencies. Preferably using NPM version 2 to avoid any problems -to check which version you are using you only need to type on the cmd (windows command prompt):

npm -v

If you have a more up to date / advanced version, you can install NPM version 2 typing:

npm install -g npm@2

Creating the the package.json:

npm init

…and answer ***Enter*** to every question. (Bear in mind that you might need to write the name of the project….)

And then….

**6) Installing Angular 2**

npm install angular2 --save

**then.....**

npm install systemjs --save

**We are going to install also other dependencies listed:**

npm install typings --save-dev

**Typings***: Angular uses [TypeScript type definition files](http://www.typescriptlang.org/Handbook" \l "writing-dts-files" \t "_blank) —*d.ts files*— to tell the compiler about the libraries loaded.*

npm install concurrently --save-dev

**Concurrently***: we can execute two npm commands at the same time. For Angular 2 this is great because we always need to perform two steps: run the TypeScript compiler and then execute the lite-server to deliver our application to the browser. Instead of having two terminals executing two commands separately, concurrently lets us to do this in one step by using the concurrent command.*

npm install lite-server --save-dev

***Lite-server:****A package that contains a lightweight development Node.js server that serves a web app, open it in the browser and refresh it when the html or JavaScript change.*

After running all those commands above, if you open the *package.json, your file*should look like this (*I will shortly explain the scripts section - I added it manually and it will be useful later*):

{  
 "**name**": "test",  
 "**version**": "1.0.0",  
 "**description**": "",  
 "**main**": "index.js",  
 "**scripts**": {  
 "**postinstall**": "npm run typings install",  
 "**tsc**": "tsc",  
 "**tsc:w**": "tsc -w",  
 "**lite**": "lite-server",  
 "**app**": "concurrent \"npm run tsc:w\" \"npm run lite\" ",  
 "**typings**": "typings",  
 },  
 "**author**": "",  
 "**license**": "ISC",  
 "**devDependencies**": {  
 "**concurrently**": "^2.0.0",  
 "**lite**-**server**": "^2.2.0",  
 "**typescript**": "^1.8.10",  
 "**typings**": "^0.7.12"  
 },  
 "**dependencies**": {  
 "**angular2**": "^2.0.0-beta.15",  
 "**es6**-**shim**": "^0.35.0",  
 "**reflect**-**metadata**": "^0.1.2",  
 "**rxjs**": "^5.0.0-beta.2",  
 "**zone**.js": "^0.6.10”,  
 "**systemjs**": "0.19.26"  
 }  
}

*If you are having problems installing the dependencies you can always copy and paste the structure of the file above, create a file called package.json in your new empty folder, open the cmd (command prompt) and point to the directory of the new folder and then run the follow command:****npm install****, which will install all the dependencies defined in****package.json****and will automatically install one by one in the****node\_modules****subfolder in the project. Doing that you can avoid all the steps I have just said and sometimes is nice to have a package.json file ready to set a new quick demo, test project.*

**Let’s have a quick look at some parts of the package.json:**

"**devDependencies**": {  
 "**concurrently**": "^2.0.0",  
 "**lite**-**server**": "^2.2.0",  
 "**typescript**": "^1.8.10",  
 "**typings**": "^0.7.12"  
 },  
 "**dependencies**": {  
 "**angular2**": "^2.0.0-beta.15",  
 "**es6**-**shim**": "^0.35.0",  
 "**reflect**-**metadata**": "^0.1.2",  
 "**rxjs**": "^5.0.0-beta.2",  
 "**zone**.js": "^0.6.10”,  
 "**systemjs**": "0.19.26"  
 }

* “**dependencies**” contains all dependencies used in our application to implement the functionality we want to have and we could add other entries for example for bootstrap or jquery.
* “**devDependencies**” section is listening the dependencies required at development time only.

**What about the scripts section?**

“scripts” section is where we are defining command shortcuts and it will be executed by *npm*command.

"**scripts**": {  
 "**postinstall**": "npm run typings install",  
 "**tsc**": "tsc",  
 "**tsc:w**": "tsc -w",  
 "**lite**": "lite-server",  
 "**app**": "concurrent \"npm run tsc:w\" \"npm run lite\" ",  
 "**typings**": "typings"  
 },

* **postinstall** — called by *npm* automatically *after* it successfully completes package installation. This script installs the [TypeScript definition files](https://angular.io/docs/ts/latest/quickstart.html" \l "typings" \t "_blank) this app requires.
* **tsc** — It runs one time the TypeScript compiler.
* **tsc:w** — “-w means watch” enables the watch mode and listen for file changes.
* **lite** — It runs the static file web server.
* **app** — It uses concurrently to run the tsc:w and the lite script at the same time. (*doing this we can avoid having two commands running in different windows*)
* **typings** — runs the *[typings](https://angular.io/docs/ts/latest/quickstart.html" \l "typings" \t "_blank)*[tool](https://angular.io/docs/ts/latest/quickstart.html" \l "typings" \t "_blank)

To run the command you have to make sure that the current directory is the project folder and then you only need to execute the*npm command* as follows:

npm run app

Running this command right now will not do anything because we did not create our first Angular 2 component. Let’s keep that for the next article…

**7) Off you go! You are ready to write your first Angular 2 app!**

That’s all really, no more steps needed! You are ready to create your first **Angular 2** app using **TypeScript.** In the [next article I will explain how to achieve this! If you want, you can always create a “.ts” file on this project and start playing with TypeScript.](https://medium.com/@menezes.carlos/angular-2-our-first-component-5f9ddbca4fe7#.7yellce24) :)

The next step will be to create an index.html and configure the SystemJS, then the boot.ts (Angular 2 bootstrap) and to finalise the Angular 2 component to make it work!

*Please let me know if you have any problems! :o)*

*Thanks for reading my first online article!*

# Angular 2 — Our first component (updated: using Beta 15)

Let’s create something really simple and basic to begin with, only because it is very important to understand all the steps. We can then progress to something that involves more work, like creating an app using all the CRUD operations, sub-components, services, pipes, directives, using dependency injection, structural directives, interpolation, property binding, event binding, variable declarations, forms, routing, etc, etc.

**Related Links:**

. [Setting up the working environment](https://medium.com/@menezes.carlos/angular-2-setting-up-your-working-environment-52b985d1d341#.g2tl1rfte)

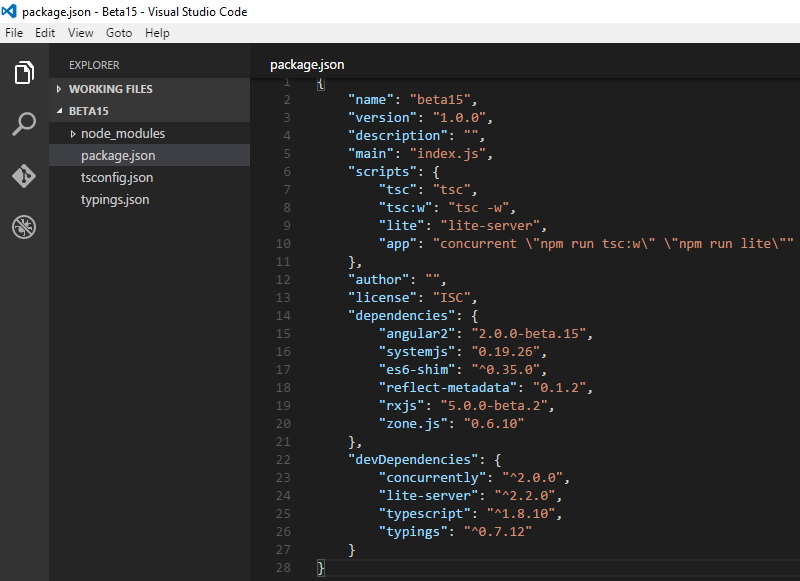
You can download the code discussed in this blog from my [**GitHub**](https://github.com/spipo/Angular2FirstComponent)**.**

#### ****Steps:****

1. **Let’s go! — Our first component**
2. **Bootstrapping the app**
3. **Creating the index.html**
4. **Running our first component**

### ****1. Let’s go! — Our first component****

If you have the [**working environment already set up**](https://medium.com/@menezes.carlos/angular-2-setting-up-your-working-environment-52b985d1d341#.2c0ij8wn2), your project should look like this:

https://cdn-images-1.medium.com/freeze/max/33/1*WWFBbvPM24-e9lCOKXxLqA.png?q=20<img class="progressiveMedia-noscript js-progressiveMedia-inner" src="https://cdn-images-1.medium.com/max/880/1\*WWFBbvPM24-e9lCOKXxLqA.png">

As you can see we have a project set up; I am using [**Visual Studio Code**](https://code.visualstudio.com/). Now we are ready to create our first simple component. To do this, create a new file called **app.component.ts**

It is important to have some knowledge about typescript language and the best place to acquire this knowledge is [here](http://www.typescriptlang.org/Handbook).

#### What’s a component?

A component is a combination of a view (the template) and some logic (our Typescript class .ts extension)

Let’s first create a class inside our **app.component.ts**:

export class **AppComponent** {  
 constructor() {}  
}

All classes in TypeScript have a constructor, whether you specify one or not. If you do not define the constructor, the compiler will automatically add one. The **constructor** is called before any other component lifecyle hook. If the component has any dependencies, the **constructor** is the best place to inject those dependencies. The **export** statement tells TypeScript that this is a module whose **AppComponent** class is public and accessible to other modules of the application.

More about [lifecycle hook](https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html) on Angular 2.

In Angular 2, components are the main way we build and specify elements and logic on the page, whilst in Angular 1 we can achieve this using **controllers** and **scope**; using Angular 2 all those concepts are combined into **Components**. Components are hierarchical way, so Angular 2 applications always start with one main component.

To identify to Angular that there is a component, we use the **@Component** decorator and to be able to use it, we have to import it:

**import {Component} from 'angular2/core';**

**@Component**()  
export class **AppComponent**{  
 constructor() {}  
}

The **@Component** decorator takes a required configuration object with the information Angular needs to create and present the component and its view.

**….we’re not finished yet!**

The **@Component** decorator is expecting only one property called: **selector**. The selector tells Angular what to look for in the HTML pages. Every time the selector is found in the HTML, Angular replaces it with our component. Let’s add the selector to our component:

import {Component} from 'angular2/core';

**@Component**({  
 **selector**: 'main-app'  
})  
export class **AppComponent**{  
 constructor() {}  
}

Every time our HTML contains the element like:

**<main-app></main-app>**

….Angular will instantiate a new instance of the class **AppComponent**.

**….and it is still not finished!**

### ****Previously we said…****

“A component is a combination of a view (the template)”

So, a component **must** have a template where you can inline it, or we can externalise the template in another file (using **templateUrl**), but for the moment let’s keep it simple.

import {Component} from 'angular2/core';

**@Component**({  
 **selector**: 'main-app',  
  **template:** '<h1>Hiya! First Component</h1>'  
})  
export class **AppComponent** {  
 constructor() {}  
}

We tell Angular that **AppComponent** is a component by attaching **metadata** to the class and the **metadata** tells Angular how to process a class.

### 2. Bootstrapping the app and loading components

The **bootstrap** method (available on Angular 2) is a subject that would take too long to write about now, therefore we will keep things simple at this stage. I address this subject in more detail [**here**](https://medium.com/@menezes.carlos/angular-2-an-introduction-of-bootstrap-and-providers-1c60ffbb7604#.uhg46974i).

We need to start our app and the bootstrap method will help us to get the job done. We have to import it from **angular2/platform/browser** module using a different location (not **angular2/core** for example) because Angular 2 supports server-side rendering or running in a Web Worker. Let’s create a new file called: **boot.ts** to separate the bootstrap logic.

import {**bootstrap**} from 'angular2/platform/browser';  
import {**AppComponent**} from './app.component';

bootstrap(**AppComponent**)  
 .**then** (success => console.log('Bootstrap successfully!'))   
 .**catch**(err => console.log(err));

A bit of background…

The bootstrap method is a promise object and the first parameter we define is the main class of our app - **the component**; that’s why we had to **import {AppComponent} from './app.component’.**

We were able to import this class because we used the ’**export**’ keyword which makes the class public across the project.As per the example above, inside the brackets **{}** is the name of our class and **from ‘’** is where the file of that class is located. Because we have all the files in the main root we don’t need to include any folder or sub folder to locate the component (ex: **‘. /app/app.component’**). At the moment we do not need to worry about the folder structure because it is a basic example; however, the second parameter of the bootstrap method is where we can have an array that contains a list of what we want to make available for injection.

The **import** statement tells the system it can get an **AppComponent** from a module namedapp.component located in a neighboring file. The module name (AKA module id) is often the same as the filename without its extension.

### 3. Creating the index.html

Create another file named: **index.html** and add the following html:

<html>

<head>  
 **<script src="node\_modules/angular2/bundles/angular2-polyfills.js"></script>**

**<script src="node\_modules/systemjs/dist/system.js"></script>**

<script>  
 **System.config({  
 defaultJSExtensions: true,  
 map: {  
 'angular2': 'node\_modules/angular2',  
 'rxjs': 'node\_modules/rxjs'  
 }  
 });  
 System.import('built/boot');**  
 </script>

</head>

<body>  
 **<main-app>**Loading…**</main-app>**  
</body>

</html>

In Angular 1.x adding scripts to our HTML files was simple, we just needed to add a script for angular.js, and a script for every JS file we wrote. Angular 2 is bundled in modules, and these modules can be loaded dynamically. To load our module we are relying on a tool **SystemJS** which is a small module loader. We added it statically into our HTML and it tells where the modules are located in our server. It then automatically figures out the dependencies between modules and downloads the ones used from our application.

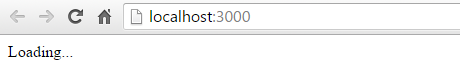
System.import('**built/boot**');

‘**build/boot**’ — Indicates where is located our **boot.js** and if you did not change your **tsconfig.json**, you should not have any problems loading the JS file.

### ****4. Running our first component****

We are pretty much done! **Hooray!** We only need to run our first component using Angular 2.

Open the **cmd** (command prompt) and go to your project directory and type: **npm run app**



Before loading the scripts you should be able to see “**Loading…**”

…and then the application should show “**Hiya! First Component**”

https://cdn-images-1.medium.com/freeze/max/33/1*ZrCq3flCqTAlsKQJCZQwTA.png?q=20<img class="progressiveMedia-noscript js-progressiveMedia-inner" src="https://cdn-images-1.medium.com/max/880/1\*ZrCq3flCqTAlsKQJCZQwTA.png">

If you open the developer tools from your browser, you should be able to see the **console.log(‘Bootstrap successfully’)** message…