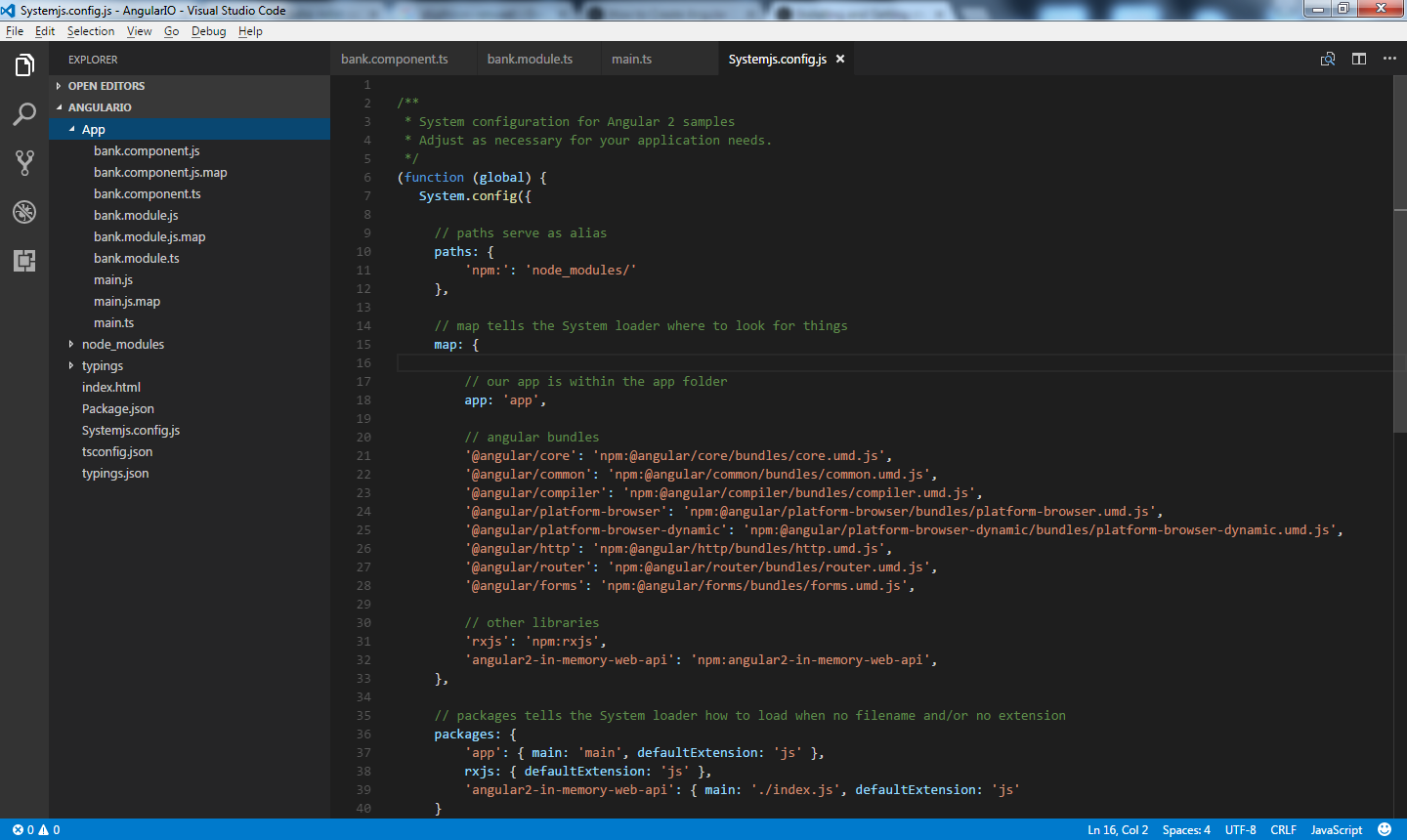
**Sample Angular 2 Application: Folder Structure View**



**Reference:**

<https://www.tektutorialshub.com/how-to-create-angular-2-application-using-system-js/>

Please install Node JS and typescript on your machine first before start working on this tutorial project.

**Tools Used: Visual Studio Code**

**Steps:**

1. Create an Application folder [Root Folder]
2. Create Configuration file
3. Install Angular 2 , Typescript, System.JS and other dependencies
4. Create First Component
5. Create a root Angular module
6. Bootstrap our application
7. Create the index.html
8. Run the application

## Create an Application Folder

Create a folder AngularIO. Then open Visual Studio Code and open the folder.

## Create Configuration file

Before we start to code, we need to configure the npm, Typescript & System.JS.

**NPM Configuration file ( Package.json )**

**Package.json** file contains the metadata about modules required for our Angular Application. It contains a list of external dependencies are used in our application. The npm ( node package manager) uses these files to install the required dependencies.

The Sample Package.json is listed below. Create file named **Package.json** in the **root folder** and copy the following content.

{

"name": "abc-bank-ltd",

"version": "1.0.0",

"scripts": {

"start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",

"lite": "lite-server",

"postinstall": "typings install",

"tsc": "tsc",

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

"typings": "typings"

},

"license": "ISC",

"dependencies": {

"@angular/common": "2.0.0",

"@angular/compiler": "2.0.0",

"@angular/core": "2.0.0",

"@angular/forms": "2.0.0",

"@angular/http": "2.0.0",

"@angular/platform-browser": "2.0.0",

"@angular/platform-browser-dynamic": "2.0.0",

"@angular/router": "3.0.0",

"@angular/upgrade": "2.0.0",

"core-js": "^2.4.1",

"reflect-metadata": "^0.1.3",

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

"systemjs": "0.19.27",

"zone.js": "^0.6.23",

"angular2-in-memory-web-api": "0.0.20",

"bootstrap": "^3.3.6"

},

"devDependencies": {

"concurrently": "^2.0.0",

"lite-server": "^2.2.2",

"typescript": "^2.0.2",

"typings":"^1.3.2"

}

}

**Typescript Configuration file: tsconfig.json**

**Typescript** requires two configuration files. One is **tsconfig.json**, which must be present in the root folder. The second file is **typings.json** which contains the Typescript definition file

Go to the **root folder of the project** and create the **tsconfig.json file**. Copy the following content.

{

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"moduleResolution": "node",

"sourceMap": true,

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"removeComments": false,

"noImplicitAny": false

}

}

Next, create the **typings.json** file and copy paste the following

{

"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"

}

}

**Systemjs.config.js (Module loader):**

The Angular 2 Applications needs module loader to load the application & associates modules dynamically. This is done using the SystemJs. The SystemJs has its own configuration file, which it uses to load the application

Create **Systemjs.config.js in the root folder** of the application and copy the following.

/\*\*

\* System configuration for Angular 2 samples

\* Adjust as necessary for your application needs.

\*/

(function (global) {

System.config({

// paths serve as alias

paths: {

'npm:': 'node\_modules/'

},

// 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',

'angular2-in-memory-web-api': 'npm:angular2-in-memory-web-api',

},

// packages tells the System loader how to load when no filename and/or no extension

packages: {

'app': { main: 'main', defaultExtension: 'js' },

rxjs: { defaultExtension: 'js' },

'angular2-in-memory-web-api': { main: './index.js', defaultExtension: 'js'

}

}

});

})(this);

## Install the Angular2 libraries, Typescript & SystemJs

The final step is to run the npm package manager to install the dependencies. Right click and select Open the Command prompt. You can also Open the command prompt directly and go to the root folder of the application

Type the following command.

npm install

If the installation is successful, you will see the **Node\_modules** & **typings folders** under the root folder. You may see many warning & errors ignore them. You can also run the npm installer again in case any issues.

If the typings folder does not appear then just run the following command in npm shell.

npm run typings install

If you still face problems in installing then run the following command in npm shell

npm cache clean

Then try running the npm install again.

Above command may fail while installing typings because of the proxy setting, we will have to change by below commands:

export HTTP\_PROXY=http://proxy.pershing.com:8080

export HTTPS\_PROXY=http://proxy.pershing.com:8080

npm config set proxy http://proxy.pershing.com:8080

npm config set https-proxy <http://proxy.pershing.com:8080>

## Create your first Component

The next step is to create a Component file. First Create an **App Folder** under the root folder. Create **bank.component.ts** inside the app directory. Copy the following code

import { Component } from '@angular/core';

@Component({

selector: 'bank-app',

template: '<h1>Hello & Welcome to ABC Bank Ltd. </h1>'

})

export class BankComponent

{

}

The Component is the most important part of the angular 2. It controls the region of the screen or View. It consists of three main parts one is class, a class decorator, and an import statement.

**Component class**

A component is a simple class. The class is defined using the Export keyword so that it can be used in other parts of the application

export class BankComponent

{

}

**@Component decorator**

The BankComponent class is then, **decorated with @Component decorator attribute**. The class decorator provides Metadata about the component class. The Angular uses this Metadata to create the view

@Component({

selector: 'bank-app',

template: '<h1>Hello & Welcome to ABC Bank Ltd. </h1>'

})

The Metadata above has two fields. The selector & template.

**Template**

The template field is plain HTML that tells angular what to display. In the example, above it displays “**Hello &** Welcome **to ABC Bank Ltd**” inside H1 tag

**Selector**

The selector tells angular, where to display. In the example above selector is “**bank-app**”.  The selector (bank-app) is replaced by the HTML template, when Angular renders the view.

**Import**

Since we are using the @Component decorator, we need to tell the Angular , where to find it. The @component decorator is available in @angular/core module. Hence we need to refer it our class. This is done using the import method. As follows.

import { Component } from '@angular/core';

## Create the root Angular Module

The Angular 2 applications are modular. Each feature of the application must be developed using small packets based on a feature and are grouped into modules. The every angular application must have at least one module called as root Module.

Create a file under the app folder with the name bank.module.ts. This is the main module of our application.

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { BankComponent } from './bank.component';

@NgModule({

imports: [ BrowserModule ],

declarations: [ BankComponent ],

bootstrap: [ BankComponent ]

})

export class BankModule{ }

First, create a BankModule class. Note that we are using the export keyword so that this module can be used in other modules. You can include any relevant codes in the class, but right now we leave it blank.

**@NgModule**

We need to tell angular that this class is an Angular Module. We can do this by decorating the class with @NgModule Decorator as shown below.

@NgModule({

imports: [ BrowserModule ],

declarations: [ BankComponent ],

bootstrap: [ BankComponent ]

})

**@NgModule** passes Metadata to angular by using the fields imports, declarations & bootstrap.

Imports**Metadata tells angular, the modules required by this module. The BankModule requires BrowserModule**

**Declaration** Metadata lists the components, directives , services etc that are part of this module. We have only one component in our example, hence we list it here.

Bootstrap Metadata identifies the root component for the module. Angular loads this component, when it loads the module. In our example, BankComponent must be loaded when the application is loaded. Hence that is listed here.

## Bootstrap our application: main.ts

Now we need to tell angular to load our root module. This requires us to create another typescript module which is a main typescript file with the following code.

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { BankModule } from './bank.module';

platformBrowserDynamic().bootstrapModule(BankModule);

In the above code, we import platformBrowserDynamic . This function contains necessary methods to bootstrap the angular application. The bootstrapper must know the location of our BankModule. Hence in the next line, we import the BankModule. Finally, we invoke the bootstrapModule(BankModule) to start our BankModule Application

## Create the Index.html

Finally, we need to create our root an HTML file, i.e. index.html. Create the index.html file in the root folder and copy the content, this will be outside of the App folder.

<!DOCTYPE html>

<html>

<head>

<title>Angular 2 Tutorial from TekTutorialsHub.com - Getting Started</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- 1. Load libraries -->

<!-- Polyfill(s) for older browsers -->

<script src="node\_modules/core-js/client/shim.min.js"></script>

<script src="node\_modules/zone.js/dist/zone.js"></script>

<script src="node\_modules/reflect-metadata/Reflect.js"></script>

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

<!-- 2. Bootstrap -->

<link href="node\_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />

<!-- 3. Configure SystemJS -->

<script src="systemjs.config.js"></script>

<script>

System.import('app').catch(function(err){ console.error(err); });

</script>

</head>

<!-- 4. Display the application -->

<body>

<div class='container'>

<bank-app>Loading...</bank-app>

</div>

</body>

</html>

**bank-app CSS Selector**

You will see that we have added bank-app CSS Selector inside the body tag of our index.html

<bank-app>Loading...</bank-app>

This is where Angular 2 loads our application. Scroll back to and take a look at the **bank.controller.ts**. In the **@Component decorator**, we have used ‘bank-app’ in the selector field. The Html inside the template field is placed inside the ‘bank-app’ CSS selector inside the index.html

@Component({

selector: 'bank-app',

template: '<h1>Hello & Welcome to ABC Bank Ltd. </h1>'

})

**System.js**

<!-- 3. Configure SystemJS -->

<script src="systemjs.config.js"></script>

<script>

System.import('app').catch(function(err){ console.error(err); });

</script>

</head>

Note that we have loaded the **SystemJs javascript**. SystemJs is responsible loading the required javascript file required by our angular Application. This is done by invoking the import method and bootstrapping the SystemJS.

## Run and Application

Finally, we are ready to roll. Right click on and select **Open the command Prompt** and enter the following command

**npm start**

