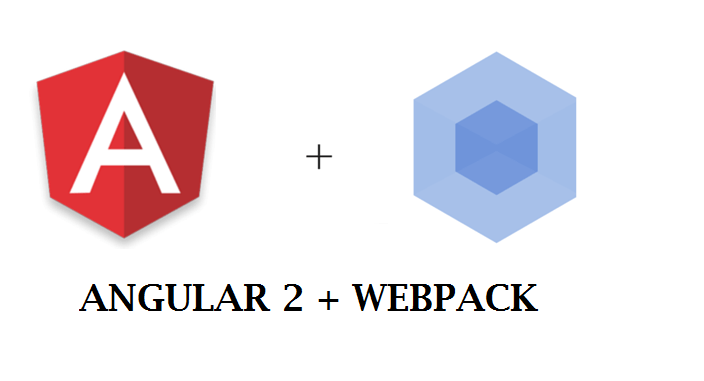
**Sample Angular 2 application with WebPack**



Make sure you have downloaded and installed [VS Code](http://bitly.com/dncvscode).

**Step 1:**Create a folder on the drive with the name NG2WebPack. Open VSCode and using the File menu, open this folder. This folder is a workspace for our application. In this folder, add a new html file of the name index.html. This html page will be used to load Angular 2 template which contains UI for the application.

**Step 2:**This step will explain the installation of the necessary modules required for the application. Right-click on index.html and select **open in command prompt**. Run the following command from the command prompt

|  |
| --- |
| npm init |

This command will create a package.json file for the project. This command expects to specify the application information e.g. name, version, description, main, author, license.

Since the project uses the Webpack, the following commands must be executed on the command prompt:

|  |
| --- |
| npm install webpack --save-dev  npm install webpack-dev-server --save-dev  npm install webpack-merge --save-dev  npm install typescript    npm install jquery bootstrap-sass bootstrap-loader css-loader node-sass resolve-url-loader sass-loader style-loader url-loader --save-dev |

In all of the above commands - -save-dev switch is used. This means that these packages will be saved in the package.json file as dependencies and devdepenedencies.

*package.json is a a manifest file at the root of the extension folder in VS Code. This file provides an overview of the structure of that file and the mandatory fields.*

The **webpack** represents the Webpack bundler, the **webpack-dev-server**represents dev tools using which a port can be defined to run the application. The **webpack-merge**is used to manage the object merging which are used in configuration file. The **style-loader**,**css-loader**are used for css and bootstrap styles. The package.json file must also specify Angular 2 components and its dependencies as shown in the following file

|  |
| --- |
| {    "name": "ng2-webpack-app",    "version": "1.0.0",    "description": "The Application for WebPack",    "main": "boot.js",    "scripts": {      "start": "webpack-dev-server --inline --progress --port 9090"    },    "keywords": [      "NG2-WebPack"    ],    "author": "MS",    "license": "ISC",    "dependencies": {      "@angular/common": "2.1.2",      "@angular/compiler": "2.1.2",      "@angular/core": "2.1.2",      "@angular/forms": "2.1.2",      "@angular/http": "2.1.2",      "@angular/platform-browser": "2.1.2",      "@angular/platform-browser-dynamic": "2.1.2",      "@angular/router": "3.1.2",      "core-js": "2.4.1",      "rxjs": "5.0.0-rc.1",      "zone.js": "0.6.26",      "webpack": "^1.13.3",      "bootstrap": "3.3.7"    },    "devDependencies": {      "@types/core-js": "0.9.34",      "@types/node": "6.0.46",      "angular2-template-loader": "0.6.0",      "awesome-typescript-loader": "2.2.4",      "bootstrap-loader": "^1.3.0",      "bootstrap-sass": "^3.3.7",      "bootstrap-webpack": "0.0.5",      "css-loader": "^0.25.0",      "extract-text-webpack-plugin": "1.0.1",      "file-loader": "0.9.0",      "html-loader": "0.4.4",      "html-webpack-plugin": "2.24.1",      "jquery": "^3.1.1",      "node-sass": "^3.11.2",      "raw-loader": "0.5.1",      "resolve-url-loader": "^1.6.0",      "rimraf": "2.5.4",      "sass-loader": "^4.0.2",      "style-loader": "^0.13.1",      "typescript": "2.0.7",      "url-loader": "^0.5.7",      "webpack": "^1.13.3",      "webpack-dev-server": "^1.16.2",      "webpack-merge": "^0.15.0"    }  } |

The above configuration contains webpack-dev-server in **scripts**section, this registers port 9090 for the application to run.

From the command prompt, run the following command to install all packages

|  |
| --- |
| npm install |

This will install all packages required for the application.

From the VS Code command prompt (Ctrl+`), run the following command to create tsconfig.json

|  |
| --- |
| tsc –init |

This file provides settings for module conde generation, ES7 decorators, etc. as shown in the following code

|  |
| --- |
| {    "compilerOptions": {      "target": "es5",      "module": "commonjs",      "moduleResolution": "node",      "sourceMap": true,      "emitDecoratorMetadata": true,      "experimentalDecorators": true,      "removeComments": false,      "noImplicitAny": true,      "suppressImplicitAnyIndexErrors": true    }  } |

**Step 3:**To the project, add folders of name **app**and **deps.**These folders will contains required application files.

**Step 4:**In the app folder, add a new file product.model.ts with Product class in it as shown in the following code:

|  |
| --- |
| export class Product{      constructor(          public productId:number,          public productName:string,          public productPrice:number,          public category:string      ){        }  } |

The above class will be used as a model class for managing product information.

**Step 5:**In the app folder, add a new file of name app.component.ts, this file contains Angular 2 component which exposes properties and functions to be exposed to view. The code for it is as follows:

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import {Product} from './product.model';    @Component({      selector: 'product-data',      templateUrl: './product.html'  })  export class ProductComponent{      product:Product;      products:Array< Product >;        constructor() {          this.product =new Product(0,'',0,'');          this.products = new Array< Product >();          this.products.push(new Product(101,'Laptop',98000,'IT'));          this.products.push(new Product(102,'Desktop',48000,'IT'));          this.products.push(new Product(103,'TV',18000,'Electronics'));       }       save(){           this.products.push(this.product);       }         clear(){           this.product =new Product(0,'',0,'');       }  } |

The above code imports @angular/core module which provides **Component**TypeDecorator. This is used to decorate typescript class as Angular 2 component. The file imports Product class from product.model file. The Component class provides **selector** property which is used to define the HTML selector to render the HTML which is defined using **templateUrl**property. This property is assigned using product.html file.

The ProductComponent class contains product object and products array of the type Product. The constructor initializes these objects with default values. The save() and clear() functions are used to push data in products array, and re-initialize product object respectively.

**Step 6:**In the app folder, add a new file product.html with the following HTML markup in it:

|  |
| --- |
| <table class="table table-stripped table-bordered">      <tr>          <td>            <table class="table table-stripped table-bordered">              <tr>                  <td>Product Id:</td>                  <td>                      <input type="text"                      [(ngModel)]='product.productId'                       class="form-control">                  </td>              </tr>              <tr>                  <td>Product Name:</td>                  <td>                      <input type="text"                      [(ngModel)]='product.productName'                      class="form-control">                  </td>              </tr>              <tr>                  <td>Product Price:</td>                  <td>                      <input type="text"                      [(ngModel)]='product.productPrice'                       class="form-control">                  </td>              </tr>              <tr>                  <td>Category:</td>                  <td>                      <input type="text"                      [(ngModel)]='product.category'                      class="form-control">                  </td>              </tr>              <tr>                  <td colspan="2">                      <input type="button" value="clear" (click)="clear()" class="btn btn-default"/>                      <input type="button" value="save" (click)="save()" class="btn btn-success"/>                  </td>              </tr>            </table>          </td>      </tr>      <tr>          <td>              <table class="table table-stripped table-bordered">                  <thead>                      <tr>                          <td>Product Id</td>                          <td>Product Name</td>                          <td>Product Price</td>                          <td>Product Category</td>                      </tr>                  </thead>                  <tbody>                       <tr \*ngFor="let p of products">                         <td>{{p.productId}}</td>                         <td>{{p.productName}}</td>                         <td>{{p.productPrice}}</td>                         <td>{{p.category}}</td>                       </tr>                  </tbody>              </table>          </td>      </tr>  </table> |

The above markup contains the databinding expressions.

**Step 7:**In the app folder, add a new file of name boot.ts. This file will contain code for bootstrapping the AppModule. The code in the file is as follows:

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

The above bootstrap code uses @NgModule decorator which imports BrowserModule and FormsModule. The BrowserModule is the module for the browser, where as the FormsModule is used for executing DataBinding.

**Step 7:**Since the WebPack is a module bundler and loader, the application needs to create a separate file where **import** statement for all required modules for the application are defined. This separate file is also needed for standard polyfills to run Angular 2 application in all standard browsers. Add a file of the name stdpkgs.ts in deps folder with following import statements in it:

|  |
| --- |
| import 'rxjs';  import '@angular/core';  import '@angular/common';  import '@angular/compiler';  import '@angular/forms';  import '@angular/http';  import '@angular/router';  import '@angular/platform-browser';  import '@angular/platform-browser-dynamic';  import 'jquery';  import 'bootstrap-loader'; |

Here the code imports all angular modules and other dependencies for the application. The importance of this is the Webpack scans the application source code and looks for **import**statement, and builds a dependency graph for all the modules.

Add a new file of the name polyfills.ts with following code in it:

|  |
| --- |
| import 'core-js/es6';  import 'core-js/es7/reflect';  require('zone.js/dist/zone');  Error['stackTraceLimit'] = Infinity;  require('zone.js/dist/long-stack-trace-zone'); |

The above code imports zone, core-js dependencies, which will be added in index.html at runtime.

**Step 8:**At the root of the application, add a file of the name webpack.config.js. This file is the most important part of the application. This file contains required configuration for loaders, entry, plugins and output. The code is as follows:

|  |
| --- |
| var webPack = require('webpack');  var htmlWebPack = require('html-webpack-plugin');  var extractTextWebPackPlugin = require('extract-text-webpack-plugin');  var path = require('path');  var config = {    entry:{      'app':'./app/boot.ts',      'libs':'./deps/stdpkgs.ts',      'polyfills':'./deps/polyfills.ts'    },    resolve:{        extensions:['','.ts','.js','css']    },    module:{        loaders:[{             test:/\.ts$/,             loaders:['awesome-typescript-loader','angular2-template-loader']        },        {          test: /\.html$/,          loader: 'html'        },        {          test: /\.css$/,          exclude: path.resolve('deps', 'app'),         // loader: extractTextWebPackPlugin.extract('style', 'css?sourceMap')         loader:"style-loader!css-loader?root=."        },         { test: /\.scss$/, loaders: ['style', 'css', 'postcss', 'sass'] },        { test: /\.(woff2?|ttf|eot|svg)$/, loader: 'url?limit=10000' },        { test: /bootstrap\/dist\/js\/umd\//, loader: 'imports?jQuery=jquery' }        ]    },   plugins: [      new webPack.optimize.CommonsChunkPlugin({             name: ['app', 'libs', 'polyfills']      }),        new htmlWebPack({        template: './index.html'      }),      new webPack.ProvidePlugin({          jQuery: 'jquery',          $: 'jquery',          jquery: 'jquery'      })    ],    output: {        publicPath: 'http://localhost:9090/',      filename: '[name].js',      },  };  module.exports = config; |

The file loads webpack, the html-webpack-plugin. This plugin is used to add < script > and < link > tags in index.html.

The **entry**section contains all files so that from these, fine dependencies can be searched. Currently this section contains app, libs and polyfills which uses files boot.ts, stdpkgs.ts and pollyfills.ts respectively. The **output**section defines that all files in entry will be output in the file specified using **finename**property. The **resolve**section tells Webpack to resolve file requests by looking at files based on extensions defined in this section. The **loaders**section defines loaders for typesctipt, html, css.

The **plugins**section defines the following features:

CommonsChunkPlugin - defines that the output file only contains code from stdpkgs.js, application code from app.component.js.

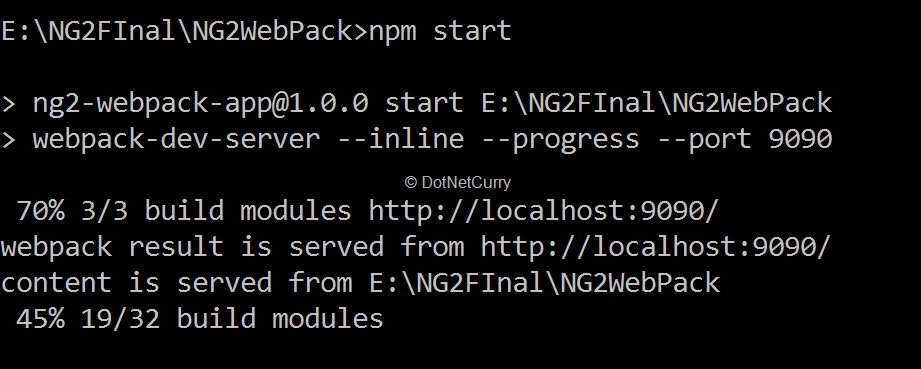
HtmlWebPackPlugIn - generates various js and css files. These will be inserted in the index.html, using this the manual job of adding js and css references in index.html can be eliminated.

The jQuery plugin is used for bootstrapping.

Build the project using ctrl+shift+B to generate the .js files. Open the VS Code command prompt (Ctrl+`), and enter the following command:

|  |
| --- |
| npm start |

This will start building modules as shown in the following image:



Once all modules are built, the result can be seen by entering the following url in the browser http://localhost:9090