How to Create a Angular 6 app from scratch using Webpack 4

1. Make a new project directory and cd into it:

   mkdir hello_angular

   cd hello_angular

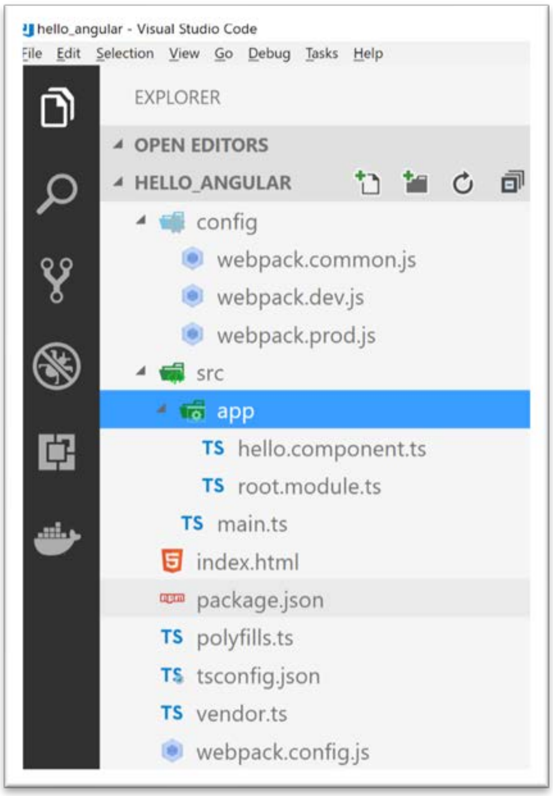
2. Create a package.json file:

   npm init -y

3. Open package.json file and copy paste the below contents in it:

```
{
  "name": "hello_angular",
  "version": "1.0.0",
  "description": "Hello World in Angular",
  "private": true,
  "scripts": {
    "start": "webpack-dev-server --mode development --hot",
    "build": "rimraf dist && webpack --config ./config/webpack.prod.js --
mode production"
  },
  "author": "Manish Sharma",
  "license": "ISC"
}
```

4. Open Folder in VSCode and create the files and folder as per below screen:



5. After the Folders are created, we must do package installations using the following commands:

   Production Dependencies

```
npm install --save rxjs zone.js @angular/core @angular/common @angular/compiler @angular/platform-browser @angular/platform-browser-dynamic core-js
```

   Development Dependencies

```
npm install --save-dev webpack webpack-cli webpack-dev-server webpack-merge
```

```
npm install --save-dev typescript@2.7
```

```
npm install --save-dev html-loader angular2-template-loader awesome-typescript-loader
```

```
npm install --save-dev html-webpack-plugin
```

```
npm install --save-dev rimraf
```

   We can also install, all the dev dependencies using one command as follows:

```
npm install --save-dev webpack webpack-cli webpack-dev-server webpack-merge typescript@2.7 html-loader angular2-template-loader awesome-typescript-loader html-webpack-plugin rimraf
```

Understanding of packages for development

webpack	webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.
webpack-cli	Webpack's Command Line Interface
webpack-dev-server	Serves a webpack app. Updates the browser on changes.
webpack-merge	webpack-merge provides a merge function that concatenates arrays and merges objects creating a new object. If functions are encountered, it will execute them, run the results through the algorithm, and then wrap the returned values within a function again. This behaviour is particularly useful in configuring webpack although it has uses beyond it. Whenever you need to merge configuration objects, webpack-merge can come in handy.
typescript	TypeScript is a language for application scale JavaScript development
angular2-template-loader	Angular2 webpack loader that inlines your angular2 templates and stylesheets into angular components.
awesome-typescript-loader	TypeScript loader for Webpack
html-loader	Exports HTML as string. HTML is minimized when the compiler demands.

html-webpack-plugin	The HtmlWebpackPlugin simplifies creation of HTML files to serve your webpack bundles. This is especially useful for webpack bundles that include a hash in the filename which changes every compilation. You can either let the plugin generate an HTML file for you, supply your own template using lodash templates, or use your own loader.
rimraf	The UNIX command rm -rf for node.

6. Open and copy paste the following code in each file, we have created earlier:

config/webpack.common.js

```
var webpack = require('webpack');
var HtmlWebpackPlugin = require('html-webpack-plugin');

const path = require('path');

module.exports = {
    entry: {
        'polyfills': './polyfills.ts',
        'vendor': './vendor.ts',
        'app': './src/main.ts',
    },

    resolve: {
        extensions: ['.js', '.ts']
    },

    module: {
        rules: [
            {
                test: /\.ts$/,
                loaders: [
                    {
                        loader: 'awesome-typescript-loader',
                        options: { configFileName: './tsconfig.json' }
                    }, 'angular2-template-loader'
                ]
            },
            {
                test: /\.html$/,
                use: [
                    {
                        loader: "html-loader",
                        options: { minimize: true }
                    }
                ]
            }
        ]
    },

    plugins: [
        new HtmlWebpackPlugin({
            template: './index.html'
        }),
        new webpack.ContextReplacementPlugin(
            /angular(\\|\/)core/,
            path.resolve(__dirname)
        )
    ],

    optimization: {
        splitChunks: {
            chunks: "all"
        }
    }
};
```

config/webpack.dev.js

```
var webpackMerge = require('webpack-merge');
var commonConfig = require('./webpack.common.js');

const path = require('path');
const rootDir = path.resolve(__dirname, '..');

module.exports = webpackMerge(commonConfig, {
    devtool: 'cheap-module-eval-source-map',

    output: {
        path: path.resolve(rootDir, 'dist'),
        publicPath: 'http://localhost:3000/',
        filename: '[name].js',
        chunkFilename: '[id].chunk.js'
    },

    devServer: {
        inline: true,
        port: 3000,
        historyApiFallback: true,
        stats: 'minimal'
    }
});
```

config/webpack.prod.js

```
var webpackMerge = require('webpack-merge');
var commonConfig = require('./webpack.common.js');

const path = require('path');
const rootDir = path.resolve(__dirname, '..');

module.exports = webpackMerge(commonConfig, {
    output: {
        path: path.resolve(rootDir, 'dist'),
        publicPath: './',
        filename: '[name].js',
        chunkFilename: '[id].chunk.js'
    }
});
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <hello>Loading Hello COmponent...</hello>
</body>
</html>
```

vendor.ts

```
import "@angular/core";
import "@angular/common";
import "@angular/compiler";
import "@angular/platform-browser";
import "@angular/platform-browser-dynamic";

import 'rxjs';
```

polyfills.ts

```
import "core-js/es6";
import "core-js/es7/reflect";
import 'zone.js/dist/zone';
```

webpack.config.js

```
module.exports = require('./config/webpack.dev.js');
```

Lets create a angular component and render it on the HTML page

1. Open src/app/hello.component.ts and write the following code.

```
import { Component } from "@angular/core";

@Component({
    selector: 'hello',
    template: `<h2>Hello World!</h2>`
})
export class HelloComponent { }
```

2. Open src/app/root.module.ts and write the following code.

```
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { HelloComponent } from "./hello.component";

@NgModule({
    imports: [BrowserModule],
    declarations: [HelloComponent],
    bootstrap: [HelloComponent]
})
export class RootModule { }
```

3. Open src/main.ts

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { RootModule } from "./app/root.module";

platformBrowserDynamic().bootstrapModule(RootModule);
```

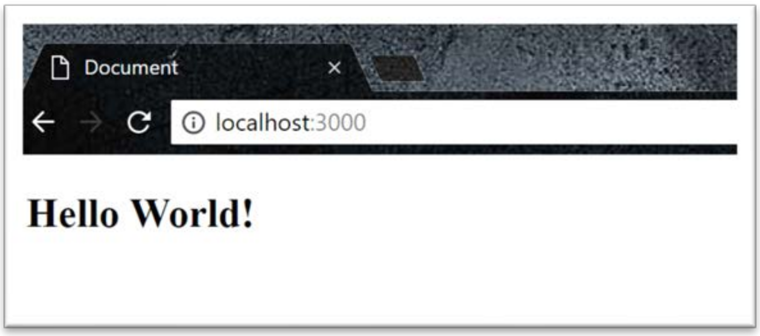
Let's run the application:

Open Command Prompt on the folder 'hello_react' and type **npm start,** after the following screen is shown

```
> webpack-dev-server --mode development --hot

i ｢wds｣: Project is running at http://localhost:3000/
i ｢wds｣: webpack output is served from http://localhost:3000/
i ｢wds｣: 404s will fallback to /index.html
i ｢wdm｣:    52 modules
i ｢wdm｣: Compiled successfully.
```

open browser and navigate to localhost:3000, you will see the following output



Hello World!

Production Build:

Open Command Prompt on the folder 'hello_angular' and type **npm run build,** after webpack processing is complete, you must see a folder named **dist,** which is the final deployment unit.