**Why we need Node.js and npm to create Angular 2 Apps?**

Technically, Node.js and NPM are not needed to create Angular 2 apps. It does ease things though. Here are the main reasons behind this choice:

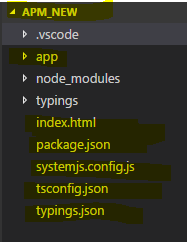
* *TypeScript*: Since we are going to develop our application using TypeScript, we need to *transpile* our *.ts* files to get them into *.js*, which can be done on-the-fly easily with Node.js and NPM.
* *Web Server*: Having Node.js helps in serving your Angular SPA from a “real” albeit light web server.
* *Download Packages*: NPM is a packet manager that allows us to download libraries and packages to use in Angular 2. Just type *npm install* in the project folder to install dependencies and get our angular project going.

**What is transpile? How is it different from Compile?**

Transpiling is a specific kind of compiling. Compiling is the general term for taking source code written in one language and transforming into another. Transpiling is a specific term for taking source code written in one language and transforming into another language that has a similar level of abstraction.

So when you compile C#, your method bodies are transformed by the compiler into IL. This cannot be called transpiling because the two languages are very different levels of abstraction.

When you compile TypeScript, it is transformed by the compiler into JavaScript. These are very similar levels of abstraction, so you could call this transpiling.

* The next step is to create a project folder. Once the folder is created, add the following package definition and configuration files to it. Make the folder structure as shown:
* 

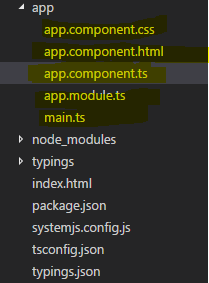
|  |  |  |
| --- | --- | --- |
| Package.json | A package.json file contains meta data about our app. Most importantly, it includes the list of dependencies to install from npm when running npm install. |  |
| tsconfig.json | We need to transpile our *.ts* files to *.js* files and*tsconfig.json*is a TypeScript configuration file that guides the compiler as it generates JavaScript files |  |
| Typings.js | Many JavaScript libraries such as jQuery, the Jasmine testing library, and Angular itself, extend the JavaScript environment with features and syntax that the TypeScript compiler doesn’t recognize natively. When the compiler doesn’t recognize something, it throws an error. We use [TypeScript type definition files](http://www.typescriptlang.org/Handbook#writing-dts-files) — *d.ts files* — to tell the compiler about the libraries we load. |  |
| Systemjs.config.js | We use [SystemJS](https://github.com/systemjs/systemjs) to load application and library modules. There are alternatives that work just fine including the well-regarded [webpack](https://angular.io/docs/ts/latest/guide/webpack.html). SystemJS happens to be a good choice. All module loaders require configuration and this is what we use this file for. |  |
| Index.html |  |  |

We install the packages listed in **package.json** using **npm**. From programs, go to Node.js and open the command window. Next, navigate to your project folder.

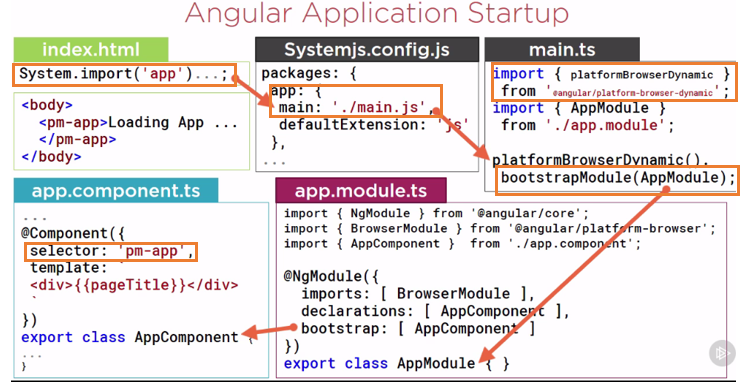
Enter the following command:



Add all the other code to main.js, app.module.ts, app.component.ts and app.component.html under the app folder.



|  |  |
| --- | --- |
| app.module.ts | import {NgModule} from '@angular/core';  import {AppComponent} from './app.component';  import {BrowserModule} from '@angular/platform-browser';  @NgModule({  imports : [BrowserModule],  declarations: [AppComponent],  bootstrap: [AppComponent]  })  export class AppModule{  } |
| app.component.ts | import {Component} from '@angular/core';  @Component ({  selector : 'pm-app',  template : `<h1> Acme Products </h1>`  })  export class AppComponent {  } |
| Main.ts | import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  import { AppModule } from './app.module';  platformBrowserDynamic().bootstrapModule(AppModule); |



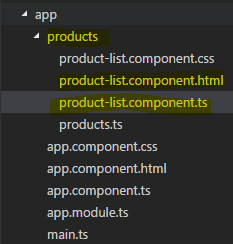
We import the *app folder* and app has main.js as the entry point in the *Systemjs.config.js* file. Main.js file bootstraps the root module (*AppModule*), which in turn bootstraps the *AppComponent* which has the selector and the template.

After writing the simple starter code, we use the following command to run the application on localhost.



* A class become an angular component when we give it Component metadata.

Now create another folder inside App, by the name products, and add two files - product-list.component.html and related ts file – product-list.component.ts



* Product-list.component.html is a simple html file with divs containing a table of products.
* Now product-list.component.ts is the related ts file.

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

@Component({

selector: 'pm-products',

templateUrl: `app/products/product-list.component.html`

})

export class ProductListComponent {

}

Add the selector as the template in root component (app.component.ts) and in the root module (app.module.ts)

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

@Component ({

Selector: 'pm-app',

template: `<pm-products> </pm-products>`

})

export class AppComponent {

}

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

import {FormsModule} from '@angular/forms';

import {AppComponent} from './app.component';

import {ProductListComponent} from './products/product-list.component';

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

@NgModule({

imports : [BrowserModule],

declarations: [AppComponent,

ProductListComponent],

bootstrap: [AppComponent]

})

export class AppModule{

}