Sharepoint framework with angular 4

Step by step

Preparation:

1. Install NodeJS (Long Term Support)
2. Install Visual Studio Code
3. Install Yeoman

Getting started:

1. First we need to make a folder for our project.
2. Open cmd and go to the folder you just made.
3. Type yo @microsoft/sharepoint in cmd
4. Yeoman will ask some questions:

* What is your solution name? => Type name or choose default
* Use the current folder
* Which type of client-side component to create? => Webpart
* What is your Web part name? => Type name or choose default
* Which framework would you like to use? => no javascript framework
* To run the project => gulp serve

1. Open the project in Visual Studio code
2. Add following dependencies to package.json

"@angular/common": "^4.3.1",

"@angular/compiler": "^4.3.1",

"@angular/core": "^4.3.1",

"@angular/forms": "^4.3.1",

"@angular/http": "^4.3.1",

"@angular/platform-browser": "^4.3.1",

"@angular/platform-browser-dynamic": "^4.3.1",

"@angular/router": "^4.3.1",

"core-js": "^2.4.1",

"reflect-metadata": "^0.1.9",

"rxjs": "5.0.1",

"sp-pnp-js": "^2.0.6",

"systemjs": "0.19.40",

"zone.js": "0.8.12"

1. Run npm install
2. Go to config/config.json and add externals

"externals": {

"Zone": "https://cdnjs.cloudflare.com/ajax/libs/zone.js/0.8.12/zone.min.js"

},

1. Add following compileroptions to tsconfig.json

"skipLibCheck": true,

"emitDecoratorMetadata": true,

1. Add following to gulpfile.js

const path = require('path');

const webpack = require('webpack');

/\*\*

\* Fixing the "5644:15-36 Critical dependency: the request of a dependency is an expression" warning

\* Linked to an existing bug/problem in Angular https://github.com/angular/angular/issues/11580

\*/

build.configureWebpack.mergeConfig({

additionalConfiguration: (generatedConfiguration) => {

generatedConfiguration.plugins.push(

new webpack.ContextReplacementPlugin(

/angular(\\|\/)core/,

path.resolve(\_\_dirname, './src')

)

);

return generatedConfiguration;

}

});

1. Go to src/webparts/{app-name) and add a folder. Name the folder ‘app’
2. In the new folder ‘app’ add a folder ‘home’
3. In the new folder ‘home’ add a file: [home.component.ts](http://home.component.ts)

import { Component, Inject } from "@angular/core";

@Component({

selector: "home",

template: require("./home.template.html") as string

})

export class HomeComponent {

title: string;

constructor() {

this.title = "hello world";

}

}

1. In the folder ‘home’ add a file: [home.template.html](http://home.template.html)

<h1>{{title}}</h1>

1. In the folder ‘home’ add a file index.ts

export \* from "./home.component";

1. In the folder ‘app’ add a file app.routes.ts

import { Routes, RouterModule } from "@angular/router";

import { ModuleWithProviders } from "@angular/core";

import { HomeComponent } from "./home";

const routes: Routes = [

{ path: "", component: HomeComponent },

{ path: "\*\*", redirectTo: "" }

];

export const AppRoutes: ModuleWithProviders = RouterModule.forRoot(routes, { useHash: true });

1. In the folder ‘app’ add a file index.ts

export \* from "./app.routes";

1. In the folder src add a new folder: core
2. In the new folder ‘core’ add a new folder: components
3. In the new folder ‘components’ add a new file: app.component.ts

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

export class AppComponent {

public static getComponent(selectorId: string): any {

return Component({

selector: `angular-${selectorId}`,

template: `<router-outlet></router-outlet>`

})(class AppComponentInner {});

}

}

1. In the folder core add a new folder: webparts
2. In the new folder ‘webparts add a new file: base-angular.webpart.ts

/\*\*

\* @Copyright (c) Microsoft Corporation. All rights reserved.

\*

\* Angular2 web part base class

\*/

import "reflect-metadata";

require("Zone");

import {

BaseClientSideWebPart

} from "@microsoft/sp-webpart-base";

import { AppComponent } from "./../components/app.component";

import { NgModule, ApplicationRef, NgZone, ModuleWithProviders } from "@angular/core";

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

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

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

import { RouterModule } from "@angular/router";

import { HttpModule } from "@angular/http";

/\*\*

\* All Angular2 client side web parts should inherit from this class.

\*/

export abstract class BaseAngularWebPart<TProperties> extends BaseClientSideWebPart<TProperties> {

/\*\*

\* Reference to the root application.

\*/

private \_app: any;

/\*\*

\* Reference to the zone of the angular2 application to put execution (and all async tasks) in the Angular controlled zone.

\*/

private \_zone: any;

/\*\*

\* Reference to the root component.

\*/

private \_component: AppComponent;

/\*\*

\* Array of class references for the NgModule declarations.

\*/

protected abstract get appDeclarationTypes(): any;

/\*\*

\* Array of class references for the NgModule imports.

\*/

protected abstract get importDeclarationTypes(): any;

/\*\*

\* Array of class references for the NgModule declarations.

\*/

protected abstract get routes(): ModuleWithProviders;

/\*\*

\* Array of class references for the NgModule declarations.

\*/

protected abstract get providers(): any[];

/\*\*

\* On property change.

\*/

public onPropertyChange(propertyPath: string, newValue: any): void {

// trigger app/root-component refresh

this.\_zone.run(() => { console.log("Outside Done!"); });

}

/\*\*

\* Render the web part. This causes the Angular2 app to be bootstrapped which

\* in turn bootsraps the Angular2 web part root component.

\*/

public render(): void {

// @todo: most likely we need to make this width:100%

this.domElement.innerHTML = `<angular-${this.context.instanceId} />`;

this.\_bootStrapModule();

}

/\*\*

\* Bootstrap the root component of the web part.

\*/

private \_bootStrapModule(): void {

var self = this;

platformBrowserDynamic().bootstrapModule(self.\_getModule()).then(

ngModuleRef => {

if(self.\_app["\_rootComponents"] !== undefined && self.\_app["\_rootComponents"].length > 0) {

self.\_component = self.\_app['\_rootComponents'][0]['\_component'] as AppComponent;

self.\_zone.run(() => { console.log("Outside Done!"); });

}

}, err => {

console.log(err);

}

);

}

/\*\*

\* Get the NgModule reference that will act as the root of this web part.

\*/

private \_getModule(): any {

const component: any = AppComponent.getComponent(this.context.instanceId);

const declarations = this.appDeclarationTypes.concat(component);

const imports = this.importDeclarationTypes;

const routes = this.routes;

const providers = this.providers;

const webPart = this;

/\*\*

\* Our goal is to define a single module class definition to be instantiated for each

webpart (like instances of a class). When an instance of the module class is bootstrapped Angular2

will create an annotation and attach it to the module class. However, when multiple instances of the

same module class are bootstrapped, only the first annotation associated with the module class will be parsed.

This results in any other module class instances on the page to not function.

To allow multiple modules of the same class definitoin on one page to work, we need to define the

class in a closure to create a new environment for each instance class, so that each annotation

object will be parsed.

\*/

const AppModule = (() => {

function AppModule(applicationRef, ngZone) {

webPart.\_app = applicationRef; // applicationRef gives us a reference to the Angular2 component's properties

webPart.\_zone = ngZone;

}

// we now attach required metadata for Angular2 that is allowable within a clousure

const AppModule1 = Reflect.decorate([

NgModule({

imports: [BrowserModule, FormsModule, HttpModule, routes, imports],

declarations: declarations,

bootstrap: [component],

exports: [RouterModule],

providers: providers

}),

Reflect.metadata("design:paramtypes", [ApplicationRef, NgZone]) // this allows Angular2's DI to inject dependencies

], AppModule);

return AppModule1;

})();

return AppModule;

}

}

// export default BaseAngularWebPart;

1. Add following imports to the main ts file (HelloWorldWebPart.ts)

import { ModuleWithProviders} from "@angular/core";

import { BaseAngularWebPart } from "../../core/webparts/base-angular.webpart";

import pnp from "sp-pnp-js";

import { AppRoutes } from "./app";

import { HomeComponent } from "./app/home";

1. Delete the render function and add following code

protected importDeclarationTypes: any = [];

protected get appDeclarationTypes(): any[] {

return [

HomeComponent

];

}

protected get routes(): ModuleWithProviders {

return AppRoutes;

}

protected get providers(): any[] {

return [

];

}

public onInit(): Promise<void> {

return super.onInit().then(\_ => {

pnp.setup({

spfxContext: this.context

});

});

}

1. Change the implementation from the interface to following

export default class HelloWorldWebPartWebPart extends BaseAngularWebPart<IHelloWorldWebPartProps>

1. Run gulp serve in cmd

Extending the project: read a list from sharepoint

1. In the app-folder add a folder shared with a folder models and a folder services
2. Add a model in the folder models: item.ts

export interface IItemModel {

Title: string;

Picture: {};

Link: {};

}

export class ItemModel implements IItemModel {

public Title: string;

public Picture: {};

public Link: {};

constructor(obj?: IItemModel) {

if (obj) {

(<any>Object).assign(this, obj);

}

}

}

1. Add a file to models: index.ts

export \* from "./item";

1. Add a folder to the folder services: interfaces
2. Add a file to the new folder interfaces: items.service.ts

import { ItemModel } from "./../../models";

export interface IItemsService {

getItems(): Promise<ItemModel[]>;

}

1. Add a service: add a file to the folder services: items.service.ts

import { Injectable } from "@angular/core";

import pnp, { ODataEntityArray } from "sp-pnp-js";

import { IItemsService } from "./../services/interfaces/items.service";

import { ItemModel } from "./../models";

@Injectable()

export class ItemsService implements IItemsService {

public async getItems(): Promise<ItemModel[]> {

return await pnp.sp.web.lists.getByTitle("ReadListTest").items.getAs<ItemModel[]>();

}

}

1. To test the project locally we are going to add some mock-data
2. Add a folder mock in services and create file: items.service.ts

import { Injectable } from "@angular/core";

import { IItemsService } from "./../../services/interfaces/items.service";

import { ItemModel } from "./../../models";

@Injectable()

export class MockItemsService implements IItemsService {

private readonly MOCK\_DELAY: number = 1000;

constructor() {

}

public async getItems(): Promise<ItemModel[]> {

return new Promise<ItemModel[]>((resolve, reject) => {

const items: ItemModel[] = [

{ Title: "Mauris", Picture: {Url: "http://fakeimg.pl/300"}, Link: {Url: "www.google.be"} },

{ Title: "Sed", Picture : {Url: "http://fakeimg.pl/300"}, Link: {Url: "www.google.be"} },

{ Title: "Quisque", Picture: {Url: "http://fakeimg.pl/300"}, Link: {Url: "www.google.be"}}

];

setTimeout(() => {

resolve(items);

}, this.MOCK\_DELAY);

});

}

}

1. Add index.ts to folder services

export \* from "./items.service";

export \* from "./mock/items.service";

1. Add imports to main ts file (HelloWorldWebParts.ts)

import {

Environment,

EnvironmentType

} from "@microsoft/sp-core-library";

import { ItemsService, MockItemsService } from "./app/shared/services";

1. Change the provider function

protected get routes(): ModuleWithProviders {

return AppRoutes;

}

protected get providers(): any[] {

if (Environment.type === EnvironmentType.Local) {

return [

{ provide: ItemsService, useClass: MockItemsService },

];

} else if (Environment.type === EnvironmentType.SharePoint || Environment.type === EnvironmentType.ClassicSharePoint) {

return [

{ provide: ItemsService, useClass: ItemsService },

];

}

}

1. Change [home.component.ts](http://home.component.ts)

import { Component, Inject } from "@angular/core";

import { ItemsService } from "./../shared/services";

import { ItemModel } from "./../shared/models";

@Component({

selector: "home",

template: require("./home.template.html") as string,

styles: [`

a {

text-decoration: none;

}

img {

float: left;

}

.space {

float: left;

margin-left: 5px;

}

`]

})

export class HomeComponent {

private items: ItemModel[] = [];

constructor( @Inject(ItemsService) private itemsService: ItemsService) {

}

public ngOnInit() {

this.itemsService.getItems().then((items: ItemModel[]) => {

this.items = this.MakeArrayListItems(items);

});

}

private MakeArrayListItems(items: ItemModel[]): ItemModel[] {

let listWithItems: ItemModel[];

let newItem: ItemModel;

let counter: number;

listWithItems = [];

for (counter = 0; counter < items.length; counter++) {

const listItem = items[Object.keys(items)[counter]];

newItem = new ItemModel;

if (listItem.hasOwnProperty("Title")) {

newItem.Title = listItem.Title;

}

if (listItem.hasOwnProperty("Picture")) {

if (listItem.Picture !== null) {

if (listItem.Picture.hasOwnProperty("Url")) {

newItem.Picture = listItem.Picture.Url;

}

}

}

if (listItem.hasOwnProperty("Link")) {

if (listItem.Link !== null) {

if (listItem.Link.hasOwnProperty("Url")) {

newItem.Link = listItem.Link.Url;

}

}

}

listWithItems.push(newItem);

}

return listWithItems;

}

}

1. Change [home.template.html](http://home.template.html)

<div \*ngFor="let item of items" >

<br>

<br>

<a href={{item.Link}}>

<img src={{item.Picture}} alt="picture" height="50" width="50">

<div class="space">

{{item.Title}}

<br>

{{item.Link}}

</div>

</a>

<br>

</div>

<br>

1. If you run gulp serve you can view the result locally
2. Add a sharepoint list in a document library with Title, Picture, Link
3. Now to read the list from sharepoint go to config/write.manifests.json and change de cdn to the right path where your list is located

"cdnBasePath": "https:/{sharepoint-tenant} /deploy/HelloWorld"

1. Go into your browser to:

https://{sharepoint-tenant}/sites/.../\_layouts/15/workbench.aspx#/

1. You see the list
2. To deploy to your sharepoint tenant :

http://www.c-sharpcorner.com/article/build-and-deploy-the-client-side-web-part-spfx-in-sharepoint-online/