Pasos para conectarse a un servidor REST en angular2:

En el archivo app.module.ts añadir los import necesarios: HttpModule, JsonpModule, algo asi:

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

import { routing } from './app.routing';

import { PostComponent } from './posts.component';

import { UserComponent } from './users.component';

import { HomeComponent } from './home.component';

import { NavbarComponent } from './navbar.component';

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

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

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

import { HttpModule, JsonpModule } from '@angular/http';

@NgModule({

declarations: [

AppComponent,

NavbarComponent,

HomeComponent,

UserComponent,

PostComponent

],

imports: [

BrowserModule,

routing,

HttpModule,

JsonpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Nota, por buenas practicas nuestros componentes solo se encargan de la parte de la vista para cualquier otra cosa como llamadas al servidor, etc se debe de crear un servicio.

En nuestro servicio añadir los siguientes imports:

import { Observable } from 'rxjs/Observable';

import { Http } from '@angular/http';

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

import 'rxjs/add/operator/map';

Observable sirve para ejecutar las llamadas al servidor, Injectable para que nuestra clase pueda ser usada en nuestro componente, http sirve para conectarse al servidor y el ultimo que es map sirve para poder maper la respuesta del servidor en json.

Luego crear un constructor masomenos asi:

constructor(private http: Http) { }

las variables privadas de preferencia empiezan con \_ (guion bajo) es por convención además de que si se declaran privadas en los parámetros del constructor ya no es necesario declararlas dentro de la clase.

Ahora creamos un método get para recibir usurios:

getUser() {

return this.\_http.get(this.\_url).map(users => users.json());

}

Llamamos al método get, le pasamos la url que previamente definimos, y mapeamos la respuesta a json. OJO aquí aun no mandamos a llamar al servidor, solo definimos los pasos a seguir pero en ningún momento se ejecuta una llamada al servidor.

Otra vez en nuestro app.module.ts ahora añadimos el servicio previamente creado para que funcione Dependency Injection, específicamente en la parte de providers :

@NgModule({

declarations: [

AppComponent,

NavbarComponent,

HomeComponent,

UserComponent,

PostComponent

],

imports: [

BrowserModule,

routing,

HttpModule,

JsonpModule

],

providers: [UserService],

bootstrap: [AppComponent]

})

export class AppModule { }

Nos dirigimos al componente con el que estábamos trabajando previamente, importamos el service que vamos a usar:

import { UserService } from './users.service';

luego creamos un constructor con un parametro privado que se llame igual q nuestro servicio para identificarlo fácilmente:

export class UserComponent {

constructor(private \_userServie: UserService) {}

}

NOTA ADICIONAL: en el constructor no se debe de mandar a llamar nada al servidor, el constructor debe de ser lo más ligero posible. En vez de usar el constructor tenemos que implementar una interfaz y su método.

La interfaz se llama : OnInit

La importamos:

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

Y el metodo de la intefaz se llama ngOnInit, basicamente la diferencia es q OnInit se ejecuta después de que se inicializo correctamente el componente.

En el método ngOnInit ya es donde se hace la llamada al servidor:

ngOnInit() {

this.\_userServie.getUser()

.subscribe(answer => {

this.user = answer;

console.log(answer);

},

error => console.log(error),

() => console.log('ya acabe')

);

}

Mandamos a llamar a nuestro servicio, el método get para recibiri la lista de usuarios, el método subscribe sirve para ejecutar la llamada al servidor, dentro de esta podemos poner cualquier variable, solo que para identificarla fácilmente le puse answer y le asignamos la respuesta hacia nuestra variable local que se llama user, nota: en el ejemplo puse:

answer => {

this.user = answer;

console.log(answer);

},

No es necesario poner las llaves, solo son necesarias para cuando quieres hacer mas de una cosa por ejemplo logearlo en consola, pero igual esta correcto:

answer =>this.user = answer,

el metodo subscribe tiene 3 posibles metodos:

subscribe(respuesta=>this.user=respuesta,error=>console.log(error),()=>consolog(‘ya acabe’));

Estos son:

Asignar la respuesta a una variable,

En caso de error hacer algo

El ultimo método se ejecuta cuando termina de ejecutarse el observable