

ANGULAR 5: Conexión con Firebase, conexión HttpClient y creación de formulario de productos

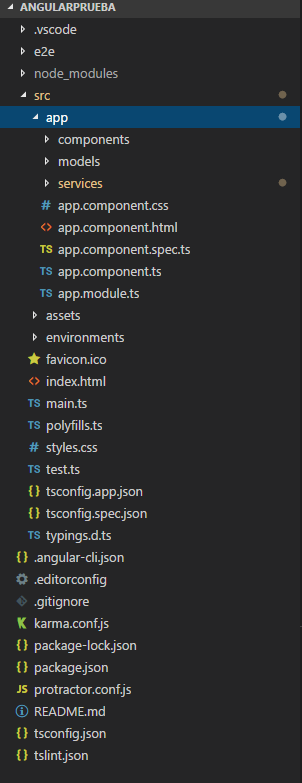
**Carlos López Nieto**

**24/04/2018**

***Índice***

1. Estructura de la aplicación
2. Conexión con Firebase
3. Conexión HttpClient
4. Formulario de Productos

***1. Estructura de la aplicación***

Para crear esta aplicación uso el editor de código fuente Visual Studio Code.

Siguiendo la documentación de la página web [angular.io](https://angular.io/guide/quickstart), creo un nuevo proyecto al que añado 3 nuevas carpetas (components, models, services) dentro del directorio /app quedando una estructura del proyecto de la siguiente manera:

Una vez tengo la estructura del proyecto creada, añado bootstrap4 y font awesome 5, para ello en el fichero *index.html* incluyo las siguientes líneas dentro de las etiquetas <head> del fichero:

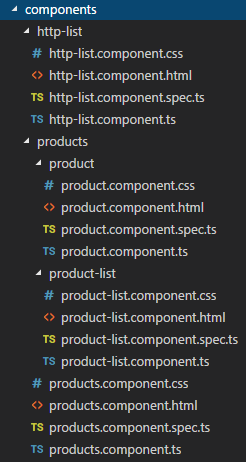
<!--BOOTSTRAP 4-->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css" integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4" crossorigin="anonymous">

<!--FONT AWESOME 5-->

<script defer src="https://use.fontawesome.com/releases/v5.0.10/js/all.js" integrity="sha384-slN8GvtUJGnv6ca26v8EzVaR9DC58QEwsIk9q1QXdCU8Yu8ck/tL/5szYlBbqmS+" crossorigin="anonymous"></script>

Como he comentado anteriormente he creado 3 nuevas carpetas en las que incluyo lo siguiente:

**Components:**

En esta carpeta es donde incluyo todos los componentes que voy creando (algunos componentes se añaden dentro de otros).

En mi caso tengo 4 componentes:

* http-list: en este componente añado y obtengo datos que serán mostrados en una tabla haciendo uso de una conexión HttpClient con un servidor remoto.
* products: simplemente es un componente que engloba otros componentes, en este caso a product y product-list. Haciendo uso de estos componentes puedo añadir y obtener datos mediante un formulario.
* product: es un componente que me permite crear nuevos productos y modificar otros ya creados.
* product-list: es un componente que muestra una lista con todos los productos que tenemos almacenados en Firebase.

**Models:**

En esta carpeta tan solo hay un fichero llamado *product.ts*, el cual incluye la definición de la clase Product y sus atributos de la siguiente manera:

export class Product {

id: string;

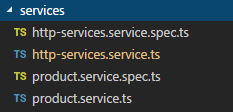
name: string;

category: string;

location: string;

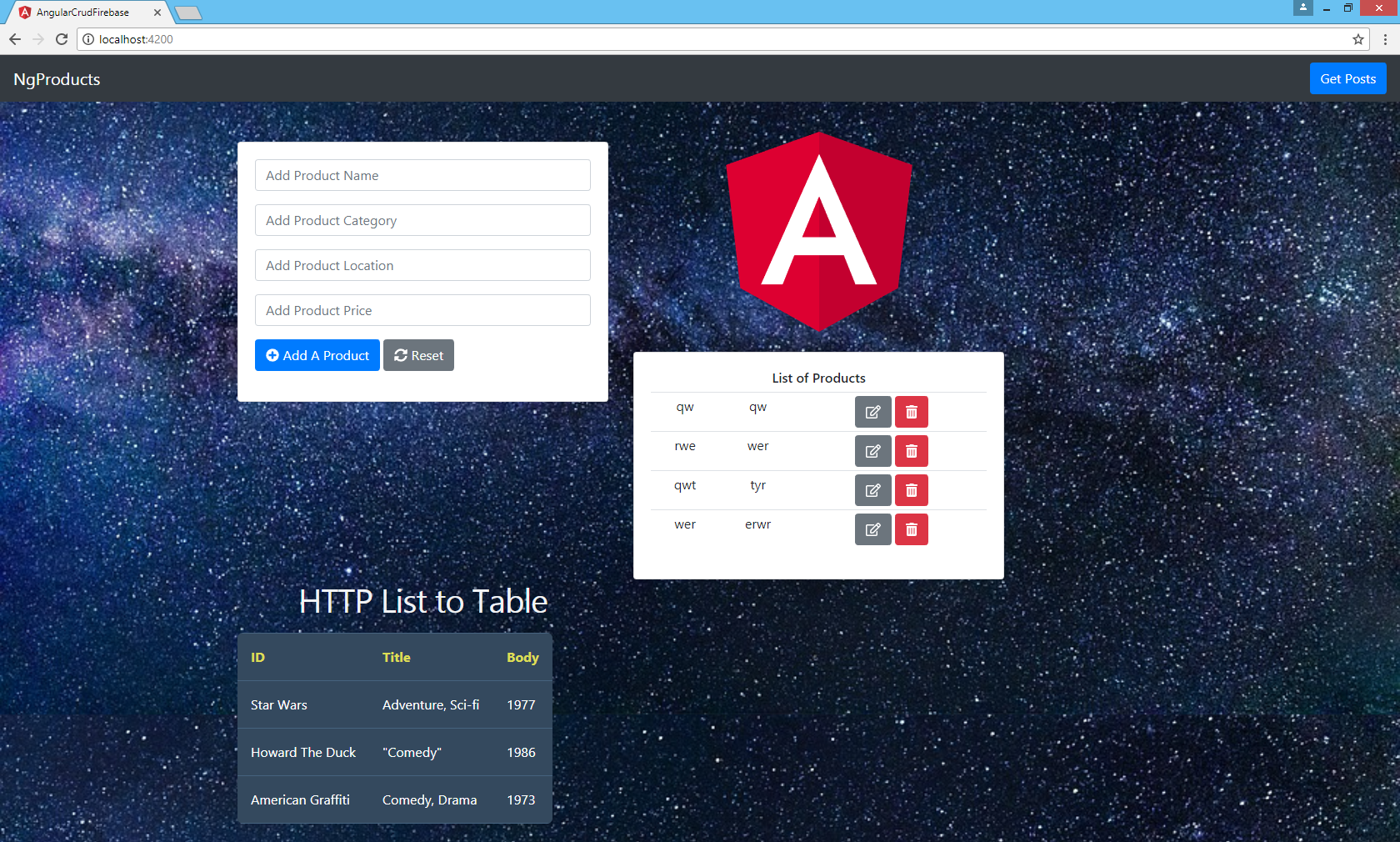
price: number;

}

**Services**

En esta carpeta es donde creo los servicios, los cuales usaré para poder realizar las distintas conexiones. En este ejemplo hago uso de dos servicios, uno para la conexión HttpClient y otro para la conexión con Firebase usando los ficheros http-services.service.ts y product.service.ts respectivamente.

Un vistazo al resultado final de esta aplicación:



***2. Conexión con Firebase***

Para poder crear una conexión con Firebase debemos instalarlo en nuestro proyecto para ello ejecutamos en siguiente comando (en caso de usar VisualStudio Code, podemos hacer uso de su terminal):

* ***npm install firebase angularfire2***

Una vez se haya instalado, necesitaremos incluir en nuestra aplicación los datos del proyecto que tengamos creado en Firebase y al cual queramos acceder, para ello en el fichero *environment.ts* añadimos tal información quedando por tanto así en este caso:

export const environment = {

production: false,

firebase: {

apiKey: "AIzaSyDdjgiQSPAA9stqcASSw2Jd-2p4yM3Id6Y",

authDomain: "angular-crud-firebase-c0805.firebaseapp.com",

databaseURL: "https://angular-crud-firebase-c0805.firebaseio.com",

projectId: "angular-crud-firebase-c0805",

storageBucket: "angular-crud-firebase-c0805.appspot.com",

messagingSenderId: "207574685869"

}

};

Tras haber realizado lo anterior, ya podemos empezar a crear los servicios que nos permitirán interactuar con Firebase. Estos servicios los crearemos en el fichero ***product.service.ts***, para ello lo primero que debemos hacer es importar lo siguiente en dicho fichero:

import { AngularFireDatabase, AngularFireList } from 'angularfire2/database';

import { Product } from '../models/product'

Con el segundo import estamos importando el modelo Product que habíamos creado anteriormente, ya que será esta clase la que usaremos para crear objetos de este tipo. En el constructor del servicio debemos añadir lo siguiente:

constructor(private firebase: AngularFireDatabase) { }

Para **obtener** todos los productos almacenados en nuestro proyecto de firebase crearemos un método llamado *getProducts()* qué nos los devolverá:

getProducts(){

return this.productList = this.firebase.list('products');

}

“productList es una variable de tipo AngularFireList<any> donde obtendremos los productos”

“La cadena ‘products’ que le pasamos al método list es el nombre de la tabla donde tenemos almacenados nuestros productos en Firebase”

Si queremos **insertar** un nuevo producto hacemos uso del método *insertProducts*, al cual le pasamos un objeto de tipo Product de la siguiente manera:

insertProducts(product: Product){

this.productList.push({

name: product.name,

category: product.category,

location: product.location,

price: product.price

})

}

Para **actualizar**  producto ya insertado usamos el método *updateProduct* el cual también recibe un Producto

updateProduct(product: Product){

this.productList.update(product.id,{ //Con esto busca el producto por la clave

name: product.name,

category: product.category,

location: product.location,

price: product.price

});

}

Por último para **eliminar** un producto le pasamos la clave de aquel producto que queramos eliminar al método *deleteProduct*:

deleteProduct($key: string){

this.productList.remove($key);

}

Si se quiere una información más detallada de cómo realizar esta tarea, yo he seguido el siguiente tutorial:

<https://youtu.be/WND9mw3HiBM>

***3. Conexión con HttpClient***

En este caso, para poder realizar conexiones mediante HttpClient no es necesario realizar ninguna instalación, Angular ya nos provee de recursos para poder hacer uso de ellas, por tanto en este caso tan solo debemos ir al fichero ***http-services.service.ts***, que será el servicio encargado de establecer estos métodos para la conexión http. Cabe destacar que para este ejemplo no se hará uso de cabeceras para la creación de peticiones.

Lo primero que debemos hacer es importar lo siguiente

import { HttpClient } from '@angular/common/http';

En el constructor debemos añadir lo siguiente:

constructor(

public http: HttpClient, private toastr: ToastrService

) { }

“Los toastr serán usado para mostrar mensajes emergentes al usuario”

Para **obtener todos los datos** almacenados en el servidor vamos a realizar una petición del tipo GET con el método httpGetProducts:

httpGetProducts() {

try {

this.http.get<Product[]>

('http://192.168.1.68:8087/product/getall').subscribe(x => {

this.products = x;

console.log("Products length: " + this.products.length);

for (let index = 0; index < this.products.length; index++) {

var table: HTMLTableElement = <HTMLTableElement>document.getElementById("tablecontent");

var row = table.insertRow(table.rows.length);

var cell1 = row.insertCell(0);

var cell2 = row.insertCell(1);

var cell3 = row.insertCell(2);

cell1.innerHTML = this.products[index].name;

cell2.innerHTML = this.products[index].category;

cell3.innerHTML = "" + this.products[index].id;

}

});

this.toastr.success('Successfull Operation', 'Successfull Operation');

} catch (e) {

console.log(e);

this.toastr.error('Failure Operation', 'Failure Operation');

}

}

Como se puede apreciar con este método realizamos la petición GET al servidor con dirección 'http://192.168.1.68:8087/product/getall’ ,el cual nos devolverá todos los productos que tenga almacenados y los iremos introduciendo en una tabla para ser mostrados de forma dinámica.

Para **introducir un nuevo producto** debemos realizar una petición POST y pasarle los distintos valores del objeto como parámetros, en este ejemplo , los datos son introducido manualmente, es decir, los datos no se recogen de ninguna parte, todo esto se lleva a cabo en el método *httpPostExample*:

httpPostExample() {

this.product = new Product();

this.product.name = 'Carlos3';

this.product.category = 'prueba';

this.product.location = 'dialogo';

this.product.price = 13;

this.http.post("http://192.168.1.68:8087/product/new",

{

name: this.product.name,

category: this.product.category,

location: this.product.location,

price: this.product.price

})

.subscribe(

(val) => {

console.log("POST call successful value returned in body",

val);

this.httpGetProducts();

},

response => {

console.log("POST call in error", response);

},

() => {

console.log("The POST observable is now completed.");

}

);

}

Para poder llevar acabo la ejecución de estos métodos he añadido un botón en la parte superior derecha de la aplicación llamado **Get Posts,** el cual al pulsarlo realiza la llamada al método *getPosts*, que a su vez llama a los dos métodos mencionados anteriormente.

getPosts() {

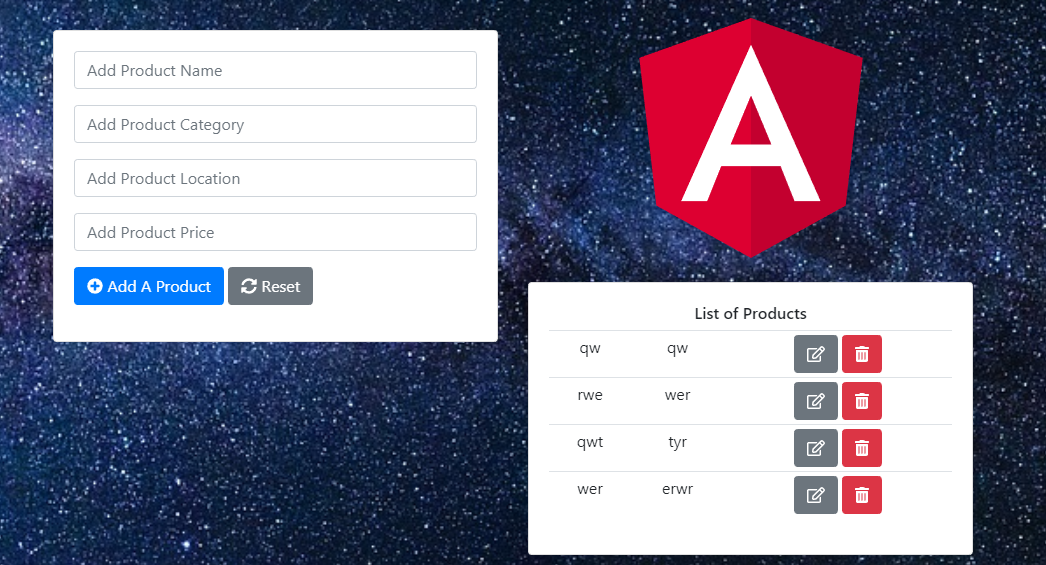
this.httpServices = new HttpServicesService(this.http, this.toastr);

this.httpServices.httpPostExample();

this.httpServices.httpGetProducts();

}

***4. Formulario de productos***



Como se puede apreciar en la imagen anterior este sería el formulario de productos, compuesto por los componentes product y product list ya mencionados. En la parte superior izquierda se encuentra el formulario para que el usuario de la aplicación pueda introducir los valores, ya sean para crear un nuevo producto o actualizar un producto ya creado.  
  
El botón **Add a Product** obtiene los valores de los campos anteriores y crea un nuevo producto, que posteriormente será pasado como parámetro al método insertProducts del servicio ***product.service.ts***. Este es su código incluido en el archive ***product.component.ts***:

onSubmit(productForm: NgForm){

if(productForm.value.$key == null){

this.productService.insertProducts(productForm.value);

}else{

this.productService.updateProduct(productForm.value);

}

this.resetForm(productForm);

this.toastr.success('Successful Operation', 'Successful Operation');

}

Como se puede ver este método servirá también para actualizar un producto en caso de que se vaya a realizar dicha modificación sobre alguno ya existente.

El botón **Reset** tan sólo limpia los valores que puedan haber en los campos para la introducción de valores.

Por otro lado, en la parte derecha inferior del formulario podemos ver aparece una lista con los objetos ya creados, en esta lista aparecen dos iconos los cuales nos permiten, editar un objeto ya creado o bien eliminándolo. Para ello y al igual que ocurría con el método de añadir, se hace una llamada a los métodos correspondientes para cada acción ya que habíamos creado en el servicio para la conexión con Firebase.

Por último, es importante añadir todos los valores importados en toda nuestra aplicación en el fichero ***app.module.ts:***

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

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

import { FormsModule } from '@angular/forms'

import { BrowserAnimationsModule } from '@angular/platform-browser/animations'

import { ToastrModule } from 'ngx-toastr'

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

//firebase

import { AngularFireModule } from 'angularfire2';

import { AngularFireDatabaseModule } from 'angularfire2/database';

import { environment } from '../environments/environment';

//components

import { ProductsComponent } from './components/products/products.component';

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

import { ProductComponent } from './components/products/product/product.component';

//services

import { ProductService } from './services/product.service';

//Importar HttpClientModule

import { HttpClientModule } from '@angular/common/http';

import { HttpListComponent } from './components/http-list/http-list.component';

@NgModule({

declarations: [

AppComponent,

ProductsComponent,

ProductListComponent,

ProductComponent,

HttpListComponent,

],

imports: [

BrowserModule,

AngularFireModule.initializeApp(environment.firebase),

AngularFireDatabaseModule,

FormsModule,

ToastrModule.forRoot(), //forRoot() hace que se ejecute en toda la aplicacion

BrowserAnimationsModule,

HttpClientModule //Para peticiones http

],

providers: [

ProductService

],

bootstrap: [AppComponent]

})

export class AppModule { }