# Angular 4

Angular 4 (de ahora en adelante solo Angular) es un framework para construir aplicaciones cliente en HTML y un lenguaje bien sea Javascript, o un lenguaje que compile a Javascript tales como TypeScript o Dart.

Escribimos la aplicación componiendo templates HTML con etiquetas de Angular, escribiendo clases componentes para controlar dichos templates, agregando lógica de la aplicación en servicios, y agrupando los componentes y servicios en módulos.

## Angular CLI

Es un plugin para NodeJS que simplifica la generación y gestión de proyectos de Angular 4.

Para comenzar instalaremos el plugin de manera global con el siguiente comando:

npm install -g @angular/cli

## Hola Mundo Angular

A continuación, crearemos nuestra primera aplicación Angular usando Angular CLI.

1. Creamos el proyecto con Angular CLI ejecutando el siguiente comando:

ng new contacts-ui

El anterior comando creara el proyecto configurando las dependencias para trabajar con Typescript y Angular 4; así como, las dependencias de desarrollo necesarias para ejecutar las pruebas de desarrollo y automáticas de la aplicación.

1. Para iniciar el proyecto ejecutamos el siguiente nos movemos al directorio de la aplicación (cd contacts-ui) y ejecutamos el siguiente comando:

ng serve

El anterior comando inicia un servidor de desarrollo que nos permitirá acceder a la aplicación en el URL: <http://localhost:4200>, es posible cambiar el puerto de escucha del servidor pasando como parámetro --port <puerto> o la forma abreviada –p <puerto>.

Ahora que hemos visto “Welcome to app!!” en nuestro navegador WEB vamos a seguir con los conceptos de Angular y como creamos una aplicación para el mundo real.

Para este proyecto de ejemplo usaremos Angular Material que es una librería de componentes de interface gráfica para Angular.

Para instalar y configurar la libraría realizaremos los siguientes pasos:

1. Instalamos las dependencias

npm install --save @angular/material hammerjs

1. En src/styles.css importamos el tema y las fuentes usadas para los iconos

@import '~@angular/material/prebuilt-themes/deeppurple-amber.css';

@import '~https://fonts.googleapis.com/icon?family=Material+Icons';

1. En el módulo importamos los módulos de animaciones, material y hammerjs

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

import { MaterialModule } from '@angular/material';

import 'hammerjs';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule,

MaterialModule

],

providers: [],

bootstrap: [AppComponent]

})

Desarrollaremos la aplicación de contactos (para la cual ya desarrollamos la API de servicios REST)

Angular CLI brinda la facilidad de generar automáticamente módulos, componentes, directivas, pipes, servicios, clases, guards, interfaces y enumeraciones usando el comando ng generate <blueprint> <nombre> o la versión abreviada ng g <blueprint> <nombre>. A continuación, la descripción de cada uno de los blueprints soportados por Angular CLI y lo que representa cada uno de ellos al interior del framework.

## Módulo

Es una clase adornada con el decorador ***@NgModule***, pasamos como parámetro al decorador un objeto metadata que indica a Angular como compilar y ejecutar el módulo.

Ejemplo:

ng g module authorization

## Componente

Es una clase adornada con el decorador ***@Component***, que controla parte de la pantalla de la aplicación llamada vista, pasamos al decorador como parámetro un objeto metadata que indica a Angular el comportamiento del componente, como debe inicializarse, el selector que indica donde debe ser inyectado en el DOM, el template que representa el HTML renderizado por el componente, el archivo de estilos usado por el componente.

Ejemplo:

ng g component contact-list

## Servicio

Es una clase adornada con el decorador @Injectable y provee funcionalidades comunes a múltiples componentes o módulos (reutilización de funcionalidades).

Ejemplo:

ng g service oauthv2-token-handler

## Templates

Usamos los templates para definir la representación gráfica de los diferentes componentes usando HTML regular, excepto por las directivas de Angular que usamos para controlar como Angular interactúa en la renderización del componente. El template puede ser definido en la metadata del decorador del componente inline usando la propiedad template o en un archivo independiente usando la propiedad templateURL.

## Data binding

Es un mecanismo soportado por Angular que permite coordinar la interacción de los datos asociados al componente y el template o viceversa. Agregamos etiquetas de binding a los templates HTML para indicar a Angular como conectar ambos extremos.

## Directivas

Son clases adornadas con el decorador ***@Directive***, pasamos como parámetro al decorador un objeto metadata para indicar a Angular como la directiva interactúa con el DOM para transformarlo. Existen directivas estructurales que alteran la distribución de los elementos del DOM, agregando, eliminando y reemplazando, y directivas de atributo que alteran la apariencia de un elemento existente.

Ejemplo:

ng g directive contacts-map

## contacts-ui

Creamos la aplicación y accedemos al directorio creado por el CLI:

ng new contacts-ui

cd contacts-ui

Ahora nos integramos con el OAuth v2 provider que implementamos en la API de servicios REST:

Generamos un guard que se encargará de verificar que el token sea válido:

ng g guard auth-guard --flat false

Generamos el servicio encargado de proveer la lógica de verificación del token:

ng g service oauthv2-token-handler --flat false

Importamos el servicio en el guard:

import { Oauthv2TokenHandlerService } from '../oauthv2-token-handler/oauthv2-token-handler.service';

Creamos el constructor para el guard e inyectamos el servicio:

constructor(private tokenHandlerService: Oauthv2TokenHandlerService){}

Creamos el método isValidToke en el servicio (este método será el encargado de verificar que el token sea válido, para el ejemplo solo validaremos que el token exista en el localStorage y que tenga el atributo access\_token.value):

isValidToken(): boolean {

let currentTokenStr = localStorage.getItem('oauth-token');

if (currentTokenStr) {

let token = JSON.parse(currentTokenStr);

return token && token.access\_token && token.access\_token.value;

} else {

return false;

}

}

Implementamos en el guard el metodo canActivate (para el ejemplo este método retorna boolean, debido a que delega la responsabilidad en el método isValidToken del servicio, el cual es un método síncrono que retorna boolean), en caso que el token no sea válido redirigimos el usuario a la página de autorización del provider de OAuth:

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

if (this.tokenHandlerService.isValidToken()) {

return true;

}

let accessCode = this.tokenHandlerService.getAccessCode();

if (accessCode) {

return this.tokenHandlerService.exchangeToken(accessCode).map((token) => true);

}

location.href = 'http://localhost:3000/oauth2/authorize?client\_id=TestApp&response\_type=code&redirect\_uri=http://localhost:4200';

}

Creamos el archivo de configuración de las rutas para la aplicación routes.ts inicialmente con solo la ruta principal de la aplicación que usaremos para configurar el guard:

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

import { AuthGuard } from './auth-guard/auth-guard.guard';

export const routeConfig: Routes = [

{

path: '',

canActivate: [AuthGuard],

children: [

]

}

];

Agregamos la configuración de las rutas a nuestro modulo:

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

import { routeConfig } from './routes';

…

imports: [

BrowserModule,

BrowserAnimationsModule,

RouterModule.forRoot(routeConfig, {useHash: true}),

MaterialModule,

MdButtonModule

],

Agregamos a Oauthv2TokenHandlerService getAccessCode que obtendrá el code entregado en la URL por el OAuth provider:

getAccessCode(): string {

let url = window.location.href;

var regex = new RegExp("[?&]code(=([^&#]\*)|&|#|$)"),

results = regex.exec(url);

if (!results) return null;

if (!results[2]) return '';

return decodeURIComponent(results[2].replace(/\+/g, " "));

}

Importamos las dependencias que requiere el servicio para comunicarse con el servidor y gestionar los eventos que puedan ocurrir durante la comunicación:

import { Http, Response, URLSearchParams, Headers } from '@angular/http';

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map';

import { Token, AccessToken } from './token';

Agregamos el método exchangeToken para intercambiar con el OAuth provider el access code por token,, este método requiere otros métodos utilitarios que también serán agregados y un stand-alone function que usaremos para mapear el token retornado por el servidor a una instancia de la interface token:

exchangeToken(accessCode: string): Observable<Token> {

let formData = this.getAuthorizationData(accessCode);

let token = this.http

.post('http://localhost:3000/oauth2/token', formData, { headers: this.getHeaders() })

.map(this.saveToken)

.catch(this.handleError);

return token;

}

private getHeaders(): Headers {

let headers = new Headers();

headers.append('Accept', 'application/json');

let clientId = 'TestApp';

let clientSecret = '123456';

headers.append('Authorization', 'Basic ' + btoa(`${clientId}:${clientSecret}`));

headers.append('Content-Type', 'application/x-www-form-urlencoded; charset=UTF-8');

return headers;

}

private getAuthorizationData(accessCode: string): URLSearchParams {

let formData = new URLSearchParams();

formData.append('code', accessCode);

formData.append('grant\_type', Oauthv2TokenHandlerService.GRANT\_TYPE);

formData.append('redirect\_uri', Oauthv2TokenHandlerService.REDIRECT\_URI);

return formData;

}

handleError(error: any) {

let errorMsg = error.message || 'Error no especificado tratando obtener el token'

console.error(errorMsg);

return Observable.throw(errorMsg);

}

saveToken(response: Response): Token {

let serverToken = response.json();

localStorage.setItem(Oauthv2TokenHandlerService.TOKEN\_KEY, JSON.stringify(serverToken));

return toClientToken(serverToken);

}

Stand-alone function toClientToken

function toClientToken(serverToken: any): Token {

let clientToken = <Token>({

access\_token: <AccessToken>({

\_\_v: serverToken.access\_token.\_\_v,

value: serverToken.access\_token.value,

clientId: serverToken.access\_token.clientId,

userId: serverToken.access\_token.userId,

\_id: serverToken.access\_token.\_id

}),

token\_type: serverToken.serverToken

});

return clientToken;

}

Agregamos el módulo http de Angular al módulo de nuestra aplicación, para poder usarlo en nuestro servicio:

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

…

imports: [

BrowserModule,

BrowserAnimationsModule,

RouterModule.forRoot(routeConfig, {useHash: true}),

HttpModule,

MaterialModule,

MdButtonModule

],

Agregamos el servicio y el guard como providers en nuestro módulo:

import { AuthGuard } from './auth-guard/auth-guard.guard';

import { Oauthv2TokenHandlerService } from './oauthv2-token-handler/oauthv2-token-handler.service';

…

providers: [AuthGuard, Oauthv2TokenHandlerService],

Ahora vamos a modificar el componente creado por el CLI y comenzar a construir la interfaz gráfica para la API de contactos:

Modificamos el archivo app.component.html, para que su contenido sea solo un router-outlet donde vamos a poner el contenido de la aplicación:

<router-outlet></router-outlet>

Creamos el componente para la ventana principal:

ng g component main-window

Modificamos el archivo main-window.component.html, para crear una barra de herramientas, agregarle el menú de la aplicación y un router-outlet que nos permitirá cambiar el contenido de la ventana a medida que navegamos en la aplicación:

<md-toolbar color="primary">

<span>Contacts</span>

<span class="fill-remaining-space"></span>

<button md-icon-button [mdMenuTriggerFor]="menu">

<md-icon>more\_vert</md-icon>

</button>

<md-menu #menu="mdMenu">

<button md-menu-item routerLink="/home">

<md-icon>home</md-icon>

<span>Inicio</span>

</button>

</md-menu>

</md-toolbar>

<router-outlet></router-outlet>

Creamos el componente contact-list:

ng g component contact-list

Modificamos el archivo de rutas para agregar una ruta a home que nos redirigirá al componente que acabamos de crear:

import { MainWindowComponent } from './main-window/main-window.component';

import { ContactListComponent } from './contact-list/contact-list.component';

…

{

path: '',

component: MainWindowComponent,

canActivate: [AuthGuard],

children: [

{ path: 'home', component: ContactListComponent }

]

}