Angular 8

Desarrollo de una aplicación web conectada a Gmail

Anastasia Macovei - 5 de junio de 2020

Introducción

Desarrollo de una aplicación web en la que nos conectaremos con nuestro usuario google a Gmail. Vamos a consultar correos recibidos, verlos en detalle y crear nuevos correos (enviándolos con nuestra cuenta de gmail).

Enlace a GitHub del proyecto

https://github.com/anmacovei/Enviar-email-con-Angular

Instalación

Para un proyecto angular necesitamos:
- Node -> instalamos descargando desde la página oficial
- CLI angular -> npm install -g @angular/cli

Creamos el proyecto con:

```
ng new nombre-proyecto
```

Arrancamos el proyecto en el navegador con:

```
ng serve -o
```

Introducción a HTML dentro de Angular

La parte HTML de angular es lo que llamamos **template**. Pero también tiene nuevas etiquetas como :

```
<app-component></app-component>
```

También tiene nuevos caracteres como:
- {{variable}}
- (evento)
- [(NgModel)]

Otros elementos html que usaremos es: 'MATERIAL' con funcionalidades extras.
También usaremos Bootstrap para la colocación ya que los estilos los haremos con *material*.

Introducción a SCSS

Es un superconjunto de CSS. Incluye más funcionalidades que CSS. Tal y como lo hacemos en CSS tenemos un #identificador-elemento{} -> por lo que ese elemento será único en el DOM. Es recomendable poner un identificador único para no hacer referencia solo al primero.
- Para hacer referencia a más de un elemento usaremos .class.

Tenemos un orden en cascada: por lo que tendremos padres, hijos, hermanos.

Themes de material:
- hay que tener claro que usaremos los colores que tiene
- Los aspectos
- La apariencia

BootStrap:
- se introducirá dentro del índex. Se usará solo para la grid system. Colocación de elementos sin crear css.

Introducción a Typescript

Typescript es un lenguaje nuevo. Es un lenguaje tipado. También compila y después se transforma en javascript.
- orientado a objetos

En nuestros ficheros ts tendremos:
- **import** -> con las librerías, servicios que usemos.
- En **@Component** tenemos:
  - Selector: de nuestro componente
  - templateURL: donde escribiremos nuestro html
  - styleURL: con su fichero scss.
- en la cabecera tendremos **export Class** y el nombre de nuestra clase -> así se genera automáticamente ese componente. Por defecto siempre implementa el método **onInit**.

Después tendremos la variables que usaremos:

```
      activo = true;
      private mensaje: string;
      public visible: boolean; //especificamos el tipo y nombre y además le
decimos el valor que tendrá por defecto
```

La siguiente parte es el **constructor**:
- es la primera función que se ejecuta y normalmente dentro importamos las clases que necesitamos que se autoinyecten. Después lo idea sería ponerle las variables que necesitemos.

A continuación el método **ngOnInit()**
- aquí irian los valores reales de nuestra variables.

Funciones:
- pueden ser tanto públicas como privadas
- Puede ser asociada a un evento en un template

Objetos y Arrays:
Las **clases** al ser exportadas a otras se transforman en objetos Javascript.
Los **Arrays** son un conjunto de cualquier cosa. Tienen tipos. Se pueden recorrer.

Introducción a Git

Git es un software de control de versiones, que almacena nuestros cambios de código.
En Git guardaremos nuestro código y nuestras copias de seguridad y en nuestro repositorio tendremos nuestro proyecto. El repositorio para cada una de las ramas tendrá diferentes clases.

Git es un sistema distribuido, eso quiere decir que cada usuario tiene una copia.

Para este proyecto se ha utilizado Git para el control de versiones y en cada lección trabajar con una rama diferente para no entrar en conflicto con la rama 'master'. Por lo que hemos subido toda la aplicación a un repositorio de Git.

Introducción a NPM

NPM es un software de control de librerías. Es con el cual hemos instalado Angular. A parte de esta, podemos usar muchas otras librerías que no necesariamente tengan que ser Angular. En nuestro caso NPM se encarga de instalar Angular para todas las librerías que necesita.
Siempre es recomendable ir a nmpjs.com para descargar librerías. Con NPM Install sabe qué librerías necesita descargar para mantener el proyecto.

Introducción a CLI/angular

CLI -> es angular. Es una librería de NPM que nos ayudará a generar por ejemplo, el proyecto. Sobretodo se usa para generar código que sea complicado y agrupar ciertos pasos.

Comandos básicos:
- ng serve [options] -> se usa para iniciar el proyecto con -o que es —open=true.
- ng generate component nombre_componente -> para generar un componente
- ng g directive nombre
- ng g interface nombre

En la página web de angular tenemos más información sobre los comandos con todas sus opciones y su uso.

Estructura de Angular

Un proyecto consta de muchos ficheros que se generan automáticamente y otros los generamos nosotros.
- index.html
- app/app.component.ts

Todos los ficheros principales son: app.component -> es el que genera Angular. Así como el fichero de index.html. Lo que trae es la página de prueba. Aquí pondríamos nuestro menú.

El fichero que más tocaremos es **app.module.ts** -> todos los ficheros que se utilicen en la aplicación tendrán que estar importados aquí.

Styles.scss -> aquí es donde pondremos nuestros estilos para toda nuestra aplicación. Angular tiene una serie de configuraciones que si no se le pide, no dejará que unos estilos de unos componentes afecten fuera de este.

Estructura de carpetas:
- será una estructura inventada donde incluiremos el código
- Tendremos la carpeta **interface**
- Carpeta **components** -> con todos los componentes de la aplicación
- Carpeta **service**: pondremos todos los servicios que conecten por ejemplo con api's, lógica de código etc.
- Carpeta **views** -> estarán las vistas y en su interior tendrá otros componentes

- **Menú** -> es mejor hacer una carpeta por separado porque el menú tendrá otras subpartes de otros componentes. Es donde creamos la barra de navegación para ir moviéndonos de un sitio a otro.

Componente

Componente: es así como la unidad mínima, la parte más pequeña que podamos hacer, donde todo se construye en base a los componentes.
Por ejemplo, en una vista se compondría de varios componentes: menú, sección que a su vez tenga otro componente con una tabla. O componente con el título. Para empezar se tienen un componente disponible pero después se pueden crear muchos más.

En el curso haremos un componente de: correo
- otro para hacer un nuevo correo, por ejemplo.

Creación de un componente:
Dentro de **app** -> creamos **Components ->** a su vez -> ng g(de generate) c (de component) correo

Para utilizar el componente que acabamos de crear:
- en **app.component.html** -> incluimos la etiqueta con el componente:

```
<app-correo></app-correo>
```

- para poder visualizar el contenido en **correo.component.ts** creamos una variable para mostrarla en la vista principal:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-correo',
  templateUrl: './correo.component.html',
  styleUrls: ['./correo.component.scss']
})
export class CorreoComponent implements OnInit {
  //creamos una variable correo dentro del componente
  correo: any;

  constructor() {
    this.correo = {
      titulo: "Título del correo",
      cuerpo: "Cuerpo del correo",
      emisor: 'anotherlittlebook@gmail.com',
      destinatario: 'correoReceptor@gmail.com'
    }
  }

  ngOnInit(): void {
  }
}
```

- y para printar su contenido en nuestra página html en **correo.component.html**:

```
<h1>{{correo.titulo}}</h1>
<p>De:{{correo.emisor}}</p>
<p>{{correo.cuerpo}}</p>
```

Estamos utilizando {{correo.}} para poder acceder al contenido de la variable que hemos creado con typescript y al ser un objeto mediante . accedemos a donde nosotros queremos.

Estructura de Angular

Son marcas que se usan en los **templates (HTML).**
- permiten editar el árbol DOM de la página, es decir, sus etiquetas y elemntos. Cuando usamos directivas estamos editando el DOM.
- La directiva más fácil sería usar **{{variable}}** -> para mostrar el contenido de una variable
- Otra directiva puede ser: [(**ngModel)**] -> de manera que se enlaza el valor a las variables typescript. Hoy en día se usan más los formularios relativos.
- **eventos**() -> que captaremos y asignaremos esta función a un elemento
- La directiva de **[]** la usamos para incluir información en una variable.
- Directivas **ngFor** o **ngIf** -> son directivas estructurales

Las directivas estructurales tienen más importancia ya que son fundamentales en Angular.
Las principales son:
- ngIf
- ngFor

Actúan como el if y el for de la programación.

ngIf -> necesita para funcionar un boolean.
- controla la existencia o no en el DOM de los elementos que aplicamos. Cuando se lo ponemos a un elemento a false, significa que ese elemento no existe y en caso contrario sí existe. Por lo que no encontraremos ese elemento si intentáramos buscarlo.

Ejemplo:

```
<div *ngIf="esVisible">Soy visible si esVisible es true.</div>
```

ngFor -> funciona como una instrucción for
- espera una variable que sea un iterable y lo recorrerá

- Normalmente usado para generar elementos dinámicamente
- Se puede aplicar a cualquier elemento

Ejemplo:

```
<div *ngFor="let correo of listaCorreos; index as i;">
   <p> {{ correo.titulo }} <span>Número {{ i }}</span></p>
  <p> {{ correo.cuerpo }} </p>
</div>
```

Para utilizar ngModel lo que tendremos que hacer es:
- mirar la documentación oficial para saber si tenemos que importar algo más para nuestra aplicación.

Validadores

Angular tiene sus propios **validadores** para por ejemplo enviar un formulario, de manera que no tengamos que programar si el email es válido, o si el campo está vacío y el usuario envíe así el formulario.

Comunicación entre componentes

Los componentes tienen una jerarquía, al igual que los htmls. Cuando queremos pasar la información a un componente hijo, hay que preparar el componente hijo con una variable. La variable abarcará algún valor. La comunicación se puede hacer de dos maneras. Una de ellas es:
- para indicar que una variable quiere usarse en el template se usa **@Input()** -> así Angular entiende que vendrá la variable desde fuera.

Los decoradores (@Input() por ejemplo) son una herramienta de Angular que indica alguna configuración a tener en cuenta de forma dinámica.
En el componente hijo pondríamos **@Input() variable: tipo;**

También podemos hacerlo desde la parte de typescript. En vez de decoradores definimos la variable y una función.

Una vez preparado el componente haremos la comunicación, editando el template, dentro podremos poner nuestra variable directamente con su valor. O en su lugar, en vez de ponerle un valor haremos **[titulo] = "Mi título"**.

Por el otro lado, si no queremos hacerlo desde la parte de template, tendremos que hacerlo mediante typescript. Para ello una vez preparada la función, la podremos llamar desde el padre. Tendremos que usar el decorador en el padre:

- **@ViewChild(selector)**
- A continuación usaremos el selector del componente hijo.

Ejemplo:

```
    @ViewChild(componente-hijo)
    variableComponenteHijo: HijoComponent;
```

Cuando hagamos el envío, se llamará la función del hijo y mediante this. podremos usarlo.

Con un **ng-container** -> tenemos la diferencia de que no se genera un elemento en el DOM. Por lo que no es ningún div, y esto nos ayuda a generar código porque de otra manera tendríamos muchos divs que nos impedirían ver la información porque saldría descolocada.

Comunicación entre hijo y el padre

Se realiza a partir de los eventos. Existen otros métodos que veremos más adelante. Los eventos, entre otros, como el evento click, pero también nos ayudan a que se comuniquen el componente hijo con el componente padre, ya que uno "avisa" a otros. Además se avisa de que realice algo.

Es muy simular a los eventos de los botones como *click*, *change*. Pero en nuestro caso, en Angular haremos nuestros propios eventos con **@OutPut()** -> con el tipo **EventEmitter**.

El uso de los eventos en el hijo, lo que haríamos sería poner el OutPut() y después emitir ese evento. Ejemplo:

```
@Output() hijoAvisando: EventEmitter<any> = new EventEmitter();
```

En este caso podremos ejecutar la emisión del evento. Esta emisión tiene la función **emit**(que dentro tendrá un parámetro). En el hijo una vez que hemos hecho el new **EventMitter**, tenemos la función de **emit** (y dentro pondremos lo que queremos enviarle).

Desde el hijo estamos emitiendo pero el padre todavía no hace nada. Lo que se hace a continuación es que se crea una función en el padre y como el click necesita una función para cuando se ejecute, se necesita disparar. Así que lo que haremos será una función para

asignar al evento y que se active al dispararse. A continuación en el template, le asignaremos la función del padre al evento del hijo.

Los observables

Son un tipo de datos que pertenecen a la librería Rxjs.
- son un canal de comunicación, similar a la de los eventos
- Tanto typscrit como javascript al ser lenguajes asíncronos, el lenguaje no se ejecuta de forma lineal, porque si hacemos peticiones y a continuación otras, ambas se hacen simultáneamente. Los observables permiten que podamos suscribirnos y podremos escribir la respuesta a continuar escribiendo con nuestros datos.
- Son muy comunes en Angular
- Todas las funciones que son asíncronas nos devolverán un observable
- Por una convención se utiliza un $ al final de la variable. Ejemplo:

```
    myObservable$ = Observable<any[]>;
```

El principal método que tiene es:
- subscribe: nos conectaremos con ese observable y cuando termine la petición haremos una acción al recibir los datos que es obligatoria. Ejemplo:

```
myObservable$.subscribe(
  (datos) => {
    // acción al recibir datos
  },
  (error) => {
    // acción solo para el caso de error
  },
  () => {
    // acción al finalizar
  }
);
```

Tenemos que hacer que cuando hagamos los subscribe tenemos que finalizarlos porque de lo contrario, se mantiene activo y generará errores. Para ello usamos:

```
const mySubscription = myObservable$.subscribe();

if(!mySubscription.closed){
  mySubscription.unsubscribe();
}
```

Servicios de Angular 8

Tenemos son una forma de organizar el código y es como si fuera un componente pero no tiene un template asociado.

¿Para qué podemos usarlo?
- se engloban 3 tipos de usos:
    - Obtener información de proveedores
    - Compartir información -> conectamos varios componentes que no tienen relación de parentesco
    - Contener lógica -> esto quiere decir aquella lógica que tiene nuestra aplicación.

Para generarlo haremos:

```
ng generate service <name> [options]
```

Rutas

Angular es una aplicación de una sola página. Cuando cambiamos de ruta se sustituye un componente por otro.
Se pueden generar rutas con:
- cli/angular
- Crearlas posteriormente con:

```
ng generate module rutas --routing --flat --module=app
```

Cuando usamos este comando autogenera dos ficheros:
- routing -> la que se ha generado automáticamente cuando creamos el proyecto, por ejemplo
- Rutas.module.ts -> podemos eliminarlo

En nuestro fichero de rutas podemos usar esta configuración básica:

```
{ path: string, component: Component }
```

En estas rutas también podemos pasar parámetros, uno o varias. O también mediante otros métodos distintos que no sean los parámetros.

Las rutas son fáciles de incluir ya que Angular lo hace prácticamente todo porque sabe dónde colocar los componentes para cambiar. El menú suele ir fuera de las rutas siempre

porque nunca cambia. Normalmente se crea el componente menú y lo escribiremos en **app.component.html**.
Para utilizar una ruta haremos un:

```
    <a [routerLink]="['/mail', identificadorMail]"> Home </a>
```

Si hemos hecho el envío de una ruta, ese componente recoge los datos enviados con **ActivatedRoute** y **Router**.

Otra utilidad es navegar a otra pantalla y eso querrá decir otra ruta por ejemplo así:

```
          irAHome() {
                this.router.navigate(['/home']);
          }
```

Material

Material es una librería con componentes listos para usar. Son elementos que solemos ver en Google. Estas librerías ya están desarrolladas con muchas utilidades.
- podemos copiar, editar o solo usarlos si quisiéramos
- Sustituye a bootstrap en todo, salvo en la colocación de elementos en la parte responsive.

En su página oficial tenemos los componentes que queramos:
- https://material.angular.io/components/categories
Tendremos en la página HTML, TS y CSS.

Para instalar Material en angular -> a partir de Angular 6 simplemente ejecutamos un comando y tendremos toda la parte de Material.

```
    ng add @angular/material
```

A través de este comando podremos elegir el tema que queramos de los que aparecen.

Themes

Son una forma para controlar los colores. Se puede cambiar dinámicamente el tema que se esté usando.
- los temas disponen de paletas de colores que incluye Angular
- Se combinan de la forma que elijamos
- A parte tenemos los tipos de estilos (que son un estilo claro u oscuro)
- Podemos usar los temas al instalarlo o si queremos podemos hacer nuestro propio tema.

Los tipos de colores importantes son:
- Primary
- Accent
- Warn

Para un tema tendremos que configurar todos estos colores especificados arriba.

También es muy fácil para crear nuestros temas, creando una carpeta y un nuevo dichero scss. Tendremos que crearlo dentro de la ruta **src/Themes/TemaBlanco.scss**, por ejemplo. Al inicio de fichero tenemos que incluir:

```
@import '~@angular/material/theming';
@include mat-core();
```

Así como las tres variables de las paletas de colores:

```
$palette-primary: mat-palette($mat-grey, 500);
$palette-accent: mat-palette($mat-light-blue);
$palette-warn: mat-palette($mat-green, 800);
```

La paleta es una gama de colores, no únicamente un color.

Cuando tengamos nuestro archivo y queramos usar el tema entraremos en src/styles/scc: Aquí importaremos el tema que hayamos creado.

```
@import 'Themes/myTema.scss';
@include angular-material-theme($light-theme);
```

También tenemos la posibilidad de crear varios temas a la vez, creando los mismos ficheros mencionado anteriormente. Para importarlo haremos:

```
.other-theme {
      @include angular-material-theme($other-theme);
}
```

Interface

Son como unos objetos qué nombres y qué propiedades tendrán. Así podremos usar más adelante ese objeto en nuestro código.

Pipes

Las pipes se usan en los **templates** para convertir datos de una forma a otra, en este curso haremos un ejemplo usando una pipe para poner en mayúsculas el nombre del usuario en la pantalla principal.
En esta práctica vamos a usar las pipes para simplificar el código que usamos en las llamadas para recoger los mensajes, de manera que pasen por una "pipe" (tubería) y cuando lleguen al componente ya tengan forma de "correo". Este uso de pipe pertenece a la parte operadores de observables de Angular.

Inyección de dependencias en Angular 8

La inyección de dependencias se da en angular principalmente en los servicios, esto es debido a que son los principales elementos que se comparten por la aplicación de manera transversal.
Lo importante es que todos los componentes comparten el mismo servicio, no se copia del mismo servicio y por ello se comparte y persiste la información contenida en ellos.

Se utiliza el Decorador **@Injectable** -> para que un servicio sea inyectable.
- lo usamos para compartirlo con el resto de la aplicación sin necesidad de incluirlos en **app.module**.
- En Angular 8 esta utilidad está muy consolidada

Lo usaremos así:

```
    constructor(public servicioAvisos: AvisosService) { }
```

En cuanto a la jerarquía, no entraremos en detalle pero sí vamos a ver que Angular va a resolver nuestra petición de una dependencia de manera jerárquica. Dependiendo de cómo hayamos registrado ese servicio, va a estar disponible o no. Por ejemplo, en **Root** siempre lo estará porque **Root** es el primer nivel.

Ciclo de vida de Angular

El ciclo de vida de un elemento de angular son las fases por las que pasa un elemento desde que se crea hasta que se destruye.

Determinadas acciones no estarán disponibles en algunos momentos del ciclo de vida de un componente. Por ejemplo, si usamos un pasamos información de un componente padre a un componente hijo, no podremos recuperar esa información en el constructor. Necesitaremos hacerlo después de que el componente se inicie.

Angular nos provee de unas funciones que se activan cuando se pasa por ese momento del ciclo de vida, para poder realizar las acciones que necesitemos de manera fácil.

Ciclo de vida simplificado:

- "constructor()"
- Todo empieza con él, se ejecuta lo primero.
- "ngOnInit()"
- Se ejecuta tras recibir las propiedades de entrada, aquí solemos inicializar todo
- "ngAfterContentInit()"
- Se ejecuta después de que Angular proyecte contenido externo en la vista del componente (ng-content).
- "ngAfterContentChecked()"
- Se ejecuta tras comprobar el contenido proyectado en el paso anterior.
- "ngAfterViewInit()"
- Se ejecuta tras iniciar todas las vistas y subvistas del elemento.
- "ngOnDestroy()"
- Este es el último en ejecutarse, justo antes de destruir el elemento. Aquí hay que tener especial atención porque debemos destruir todas las subscripciones a observables que se iniciaran en el elemento, de lo contrario, seguirán ejecutándose.

CURSO FINALIZADO