



CREACIÓN del PROYECTO

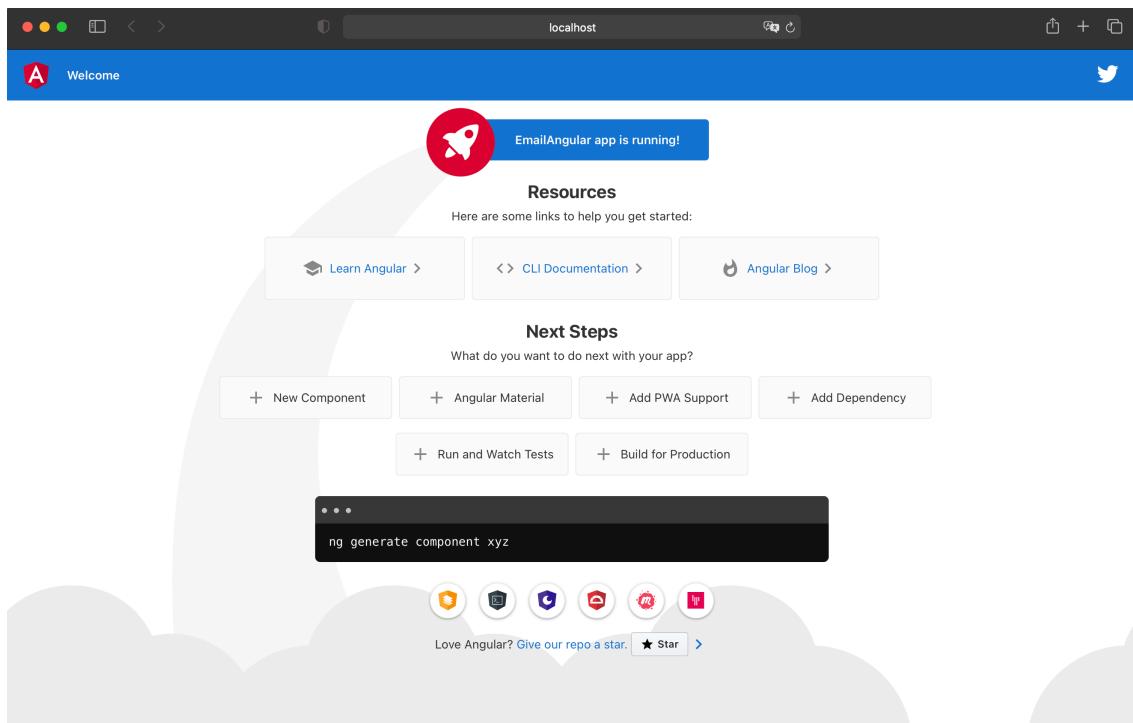
Creamos un nuevo proyecto que nos va a permitir conectarnos con nuestro usuario de google a Gmail, consultar los correos recibidos, responder o crear correos nuevos.

`ng new EmailAngular`

Esta vez queremos:

- Que genere él las rutas.
- Que los ficheros de estilos tengan el formato SCSS.

Después arrancamos el servidor: `ng serve -o` (con -o arrancamos el navegador) desde la ruta del proyecto.



HTML en ANGULAR

- HTML es la base de los **templates** en Angular.
- Podemos seguir utilizando las etiquetas de siempre.
- Podemos utilizar también directivas `<app-root></app-root>`
- Podemos encontrar también nuevos caracteres:



Nuevos caracteres

- {{ variable }}
- (evento)
- [(ngModel)]
- #etiqueta
- Etc ...

Directivas, “hook” entre el HTML de los templates y TypeScript

- Con doble llave insertamos los valores de las variables de TypeScript.
 - Con paréntesis, vamos a ver los eventos, etc.
- Usaremos la librería **Material** muy bien integrada con Angular.

“MATERIAL”: Elementos con funcionalidades extra al simple html.

Ejemplo:

- <input matInput>
- <mat-checkbox>Check me!</mat-checkbox>

- Podremos integrar Bootstrap, en este ejemplo para la colocación: sistema de grid, etc. Usaremos importación directa en el **index.html**. También podríamos descargarlo como librería.

TYPESCRIPT

- Lenguaje tipado.
- Se transpila.
- Ejemplo:



```
TS correo.component.ts ✘

1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-correo',
5   templateUrl: './correo.component.html',
6   styleUrls: ['./correo.component.scss']
7 })
8 export class CorreoComponent implements OnInit {
9
10   title: string;
11   public body: string;
12   private isStarted = false;
13
14   constructor() {
15     this.title = 'Correo';
16     this.body = 'Cuerpo del correo';
17   }
18
19   ngOnInit() {
20     this.isStarted = true;
21   }
22
23   private clickBoton() {
24     console.log("Click en botón");
25   }
26
27 }
28
```

- Mediante *import* se incluirán los componentes, librerías, etc que se van a usar.
OnInit lo usamos en la clase y Component lo necesitamos siempre.
- El decorador @Component ya lo vimos en el pdf anterior.
- A continuación, viene la cabecera con la clase:
 - o Se declaran las variables indicando el tipo.
 - o Con modificadores (si no ... automáticamente cogería *public*)
 - o Si no establecemos tipo, cogería *any*.
 - o En el constructor, se inicializan las variables. Es la parte más importante de componentes, servicios, etc. Es la primera función que se ejecuta cuando se crea el elemento. También aquí se importarán las clases que necesitaremos que se autoinyecten en la clase.
 - o ngOnInit se ejecuta después del constructor. Ahí le establecemos los valores reales de las propiedades de los elementos.
 - o El siguiente ejemplo es la creación de una función privada de la clase.



- Hay otros tipos de objetos fundamentales en TypeScript:
 - o Objetos: {}
 - Las clases, al ser exportadas a otras, se transforman en objetos JavaScript.
 - Mediante interfaces o clases podemos crear nuestros tipos de datos para usar en la aplicación.
 - Modelar nuestros objetos tiene muchas ventajas.
 - o Arrays: []
 - Tipos de datos iterables
 - Contienen listas de objetos
 - La base para cualquier transmisión de datos

NPM

- Software de control de librerías.
- Provee utilidades para nuevos desarrollos.
- Además de instalar Angular, nos permite instalar librerías externas
- Mantiene las dependencias entre paquetes
- Para descargar librerías en Angular:
 - o Sólo necesitaremos conocer su nombre si es público
 - o Mediante buscadores en internet
 - o Mediante web especializadas gratuitas como: <https://www.npmjs.com>
 - o Si por ejemplo nos descargamos un proyecto de Git, para que reconozca las librerías las descargue y las instale tenemos que hacer: `npm install` y posteriormente ya podremos hacer `ng serve`.

CLI de ANGULAR

- CLI es Angular
- Es una librería npm
- Genera código
- Configura, compila y ejecuta nuestra aplicación

Arranque

- `ng serve [options]`

Usaremos la opción '`-o`' para abrir el navegador al iniciar el proyecto

Es una contracción del comando '`--open=true`'



Creación

- `ng generate component nombre`
- `ng g directive nombre`
- `ng g interface nombre`
- `ng g module nombre`
- `ng g pipe nombre`
- `ng g service nombre`

<https://cli.angular.io/>

ESTRUCTURA

- Ficheros de Angular

- Al crear un proyecto en Angular se generan algunos ficheros:
 - **index.html**: Primer fichero en ejecución
Contiene la cabecera con las importaciones e inicia el contenido principal de la aplicación
 - **styles.scss**: Los estilos principales de la aplicación
- **app/app.component.ts**: Primer componente en ser ejecutado tras index.html
- **app/app.module.ts**: Indica al sistema el conjunto de ficheros que se tendrán en cuenta en la aplicación y deben estar disponibles

El fichero app.component.spec.ts es para test.

- Estructura de carpetas.

- Crearemos una serie de carpetas para almacenar los distintos ficheros que generemos:
 - **Interfaces**: Estos elementos nos ayudan en TypeScript a la hora de interactuar con objetos
 - **Components**: Los componentes de nuestra aplicación; todo aquello que se pueda usar como tal



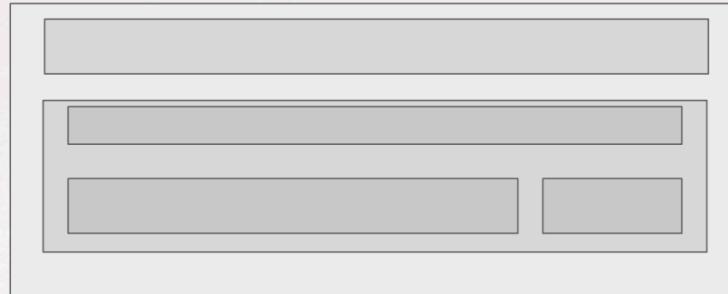
- **Services:**
 - Servicios web
 - Lógica de negocio
 - Interconexión de componentes
- **Views:** Componentes partes del menú
Tienen el contenido y es donde se importan los componentes
- **Menu:** Al igual que las vistas, es una buena opción incluir aquí los componentes que constituyen el menú de la aplicación

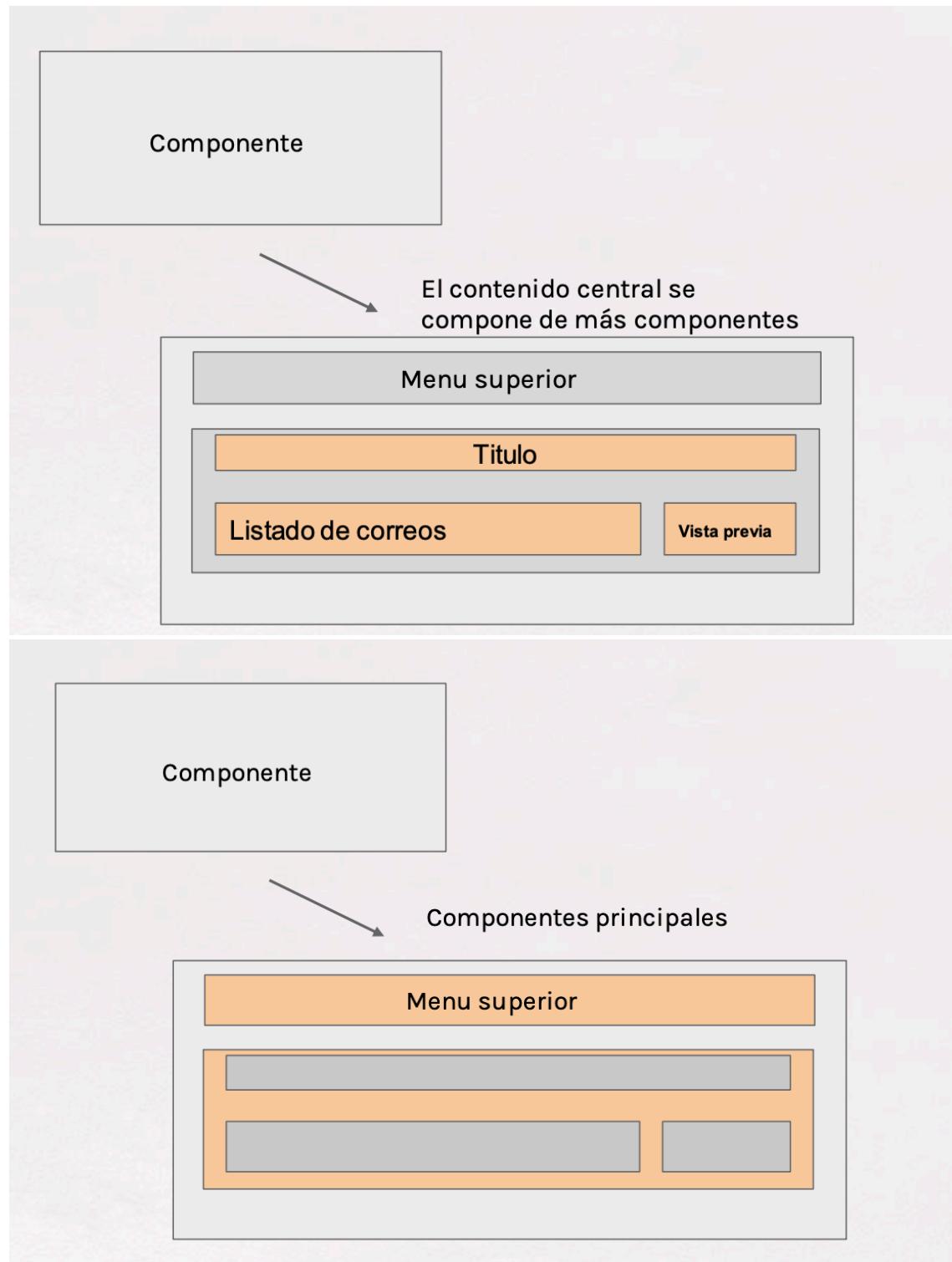
COMPONENTES

- Unidad mínima, pieza.

Componente

Componentes en una vista simple





- Reutilización de componentes.

CREACIÓN DEL COMPONENTE CORREO

- Eliminamos el contenido de la vista app.component.html.



```
app.component.html — EmailAngular
...
<app.component.html> X TS app.component.ts
src > app > <app.component.html> > router-outlet
1   <router-outlet></router-outlet>
```

- Crearemos el componente correo y lo mostraremos por pantalla.
- Puedo ejecutar un terminal desde Visual Studio Code.

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with the project structure:

- OPEN EDITORS
- EMAILANGULAR
- e2e
- node_modules
- src
- app
- app-routing.module.ts
- app.component.html M
- app.component.scss
- app.component.spec.ts
- app.component.ts
- app.module.ts
- assets
- environments
- favicon.ico
- index.html
- main.ts
- polyfills.ts
- styles.scss
- test.ts
- .browserslistrc
- .editorconfig
- .gitignore
- angular.json
- karma.conf.js
- package-lock.json
- package.json
- README.md
- tsconfig.app.json

On the right, the main editor shows the app.component.html file with the router-outlet component. Below it is the Terminal tab, which displays the output of a build process:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node
polyfills.js | polyfills | 141.34 kB
main.js | main | 10.91 kB
runtime.js | runtime | 6.15 kB
styles.css | styles | 118 bytes
Initial Total | 2.80 MB
Build at: 2020-12-09T10:54:53.405Z - Hash: 8fcf68621251869ce7a5 - Time: 5153ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Compiled successfully.
```

- Creo dentro de **app** una carpeta **components** y dentro voy a crear un componente abriendo otro terminal (BDR-> Open terminal): *ng g c correo*

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh
inma@MacBook-Pro-de-Inmaculada components % ng g c correo
CREATE src/app/components/correo/correo.component.scss (0 bytes)
CREATE src/app/components/correo/correo.component.html (21 bytes)
CREATE src/app/components/correo/correo.component.spec.ts (626 bytes)
CREATE src/app/components/correo/correo.component.ts (276 bytes)
UPDATE src/app/app.module.ts (486 bytes)
inma@MacBook-Pro-de-Inmaculada components %
```

- Nos actualiza *app.module.ts*



```
app.module.ts — EmailAngular
...
... < app.component.html < TS app.module.ts > ...
src > app > TS app.module.ts > ...
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { CorreoComponent } from './components/correo/correo.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     CorreoComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
```

- En *correo.component.html* podemos ver un párrafo de prueba.
- En *correo.component.ts* podemos ver el selector para usarlo.
- En *app.component.html*, incluimos el componente.

```
app.component.html — EmailAngular
...
... < app.component.html > < TS correo.component.ts >
src > app > < app.component.html > < app-correo >
1   <router-outlet></router-outlet>
2   <app-correo></app-correo>
```

localhost:4200

Aplicaciones Portal de Educaci... Educamos

correo works!

- Vamos a modificar el componente para añadir algo de lógica de prueba.



```
↳ app.component.html      TS correo.component.ts ×
src > app > components > correo > TS correo.component.ts > CorreoComponent > ☰
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-correo',
5     templateUrl: './correo.component.html',
6     styleUrls: ['./correo.component.scss']
7   })
8   export class CorreoComponent implements OnInit {
9
10   correo: any;
11
12   constructor() {
13     this.correo = {
14       titulo: "Titulo del Email",
15       cuerpo: `Cuerpo del email, Cuerpo del Email, Cuerpo del Email,
16       Cuerpo del Email,Cuerpo del Email,Cuerpo del Email,
17       Cuerpo del Email,Cuerpo del Email,Cuerpo del Email`,
18       emisor: 'correoEmisor@prueba.com',
19       destinatario: 'correoReceptor@prueba.com'
20     }
21   }
22
23   ngOnInit(): void {
24   }
25
26 }
27

↳ app.component.html      TS correo.component.ts      ↳ correo.component.html ×      ☰
src > app > components > correo > ↳ correo.component.html > ☰ p
1   <h1>{{correo.titulo}}</h1>
2   <p>De: {{correo.emisor}}</p>
3   <p>A: {{correo.destinatario}}</p>
4   <p>{{correo.cuerpo}}</p>
5
```



Titulo del Email

De: correoEmisor@prueba.com

A: correoReceptor@prueba.com

Cuerpo del email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email,Cuerpo del Email,Cuerpo del Email, Cuerpo del Email,Cuerpo del Email,Cuerpo del Email

DIRECTIVAS DE ANGULAR

- Son marcas en los “templates” (HTML)
- Permiten editar el árbol DOM de la página
- Angular integra muchas de ellas

`{{ variable }}`

Nos sirve para mostrar el contenido de una variable

Ejemplo: `{{ titulo }}`

`[(ngModel)]`

Enlaza el valor a variables TypeScript

Ejemplo: `<input [(ngModel)]="name">`



[(ngModel)] permite enlazar variables de TypeScript con contenido HTML. Veremos después un ejemplo aunque ahora se usan más los formularios reactivos.

(event)

Captura y asigna una función a un evento de un elemento

Ejemplo:

```
<button (click)="mostrar= !mostrar">Mostrar</button>
```

<app-component></app-component>

Incluye un componente Angular

[variable]

Incluir información a componentes

Ejemplo: <spinner [color]="warn"></spinner>

Nos sirve para incluir información en una variable. Lo veremos después

*ngFor o *ngIf

Servirán como if y for en los “templates”

Estas directivas son estructurales y las veremos a continuación.

DIRECTIVAS ESTRUCTURALES



Nglf

- Funciona como la instrucción IF
- Espera una variable boolean para evaluar
- Controla la existencia de elementos o componentes
- Aplicable a cualquier elemento

***ngIf="conditionVar"**

```
<div *ngIf="isVisible">  
    <p> Soy visible si isVisible es true  
</p>  
</div>
```

<https://angular.io/api/common/NgIf>

Si un elemento no existe, no podemos acceder a él, no es que no sea visible, es que directamente no existirá.



NgFor

- Funciona como la instrucción FOR
- Espera una variable iterable
- Usado normalmente para generar elementos dinámicamente
- Aplicable a cualquier elemento

```
*ngFor="let elemento of lista"  
*ngFor="let elemento of lista; index as i;"
```

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

<https://angular.io/api/common/NgForOf>

COMPONENTE LISTA DE CORREO

- Siguiendo con nuestro ejemplo crearemos un componente de lista de correos.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh +   
inma@MacBook-Pro-de-Inmaculada:~/components % ng g c lista-correos  
CREATE src/app/components/lista-correos/lista-correos.component.scss (0 bytes)  
CREATE src/app/components/lista-correos/lista-correos.component.html (28 bytes)  
CREATE src/app/components/lista-correos/lista-correos.component.spec.ts (669 bytes)  
CREATE src/app/components/lista-correos/lista-correos.component.ts (303 bytes)  
UPDATE src/app/app.module.ts (605 bytes)  
inma@MacBook-Pro-de-Inmaculada:~/components %
```



The screenshot shows a code editor with three tabs open:

- correo.component.ts**:

```
src > app > components > lista-correos > correo.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-lista-correos',
5   templateUrl: './lista-correos.component.html',
6   styleUrls: ['./lista-correos.component.scss']
7 })
8 export class ListaCorreosComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14
15 }
```
- correo.component.html**:

```
src > app > correo.component.html > ...
1 <router-outlet></router-outlet>
2 <app-lista-correos>
```
- lista-correos.component.ts**:

```
src > app > components > lista-correos > lista-correos.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-lista-correos',
5   templateUrl: './lista-correos.component.html',
6   styleUrls: ['./lista-correos.component.scss']
7 })
8 export class ListaCorreosComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14
15 }
```

Below the code editor is a browser window showing the application running at localhost:4200. The address bar also shows `localhost:4200`.

Navigation icons: back, forward, refresh.

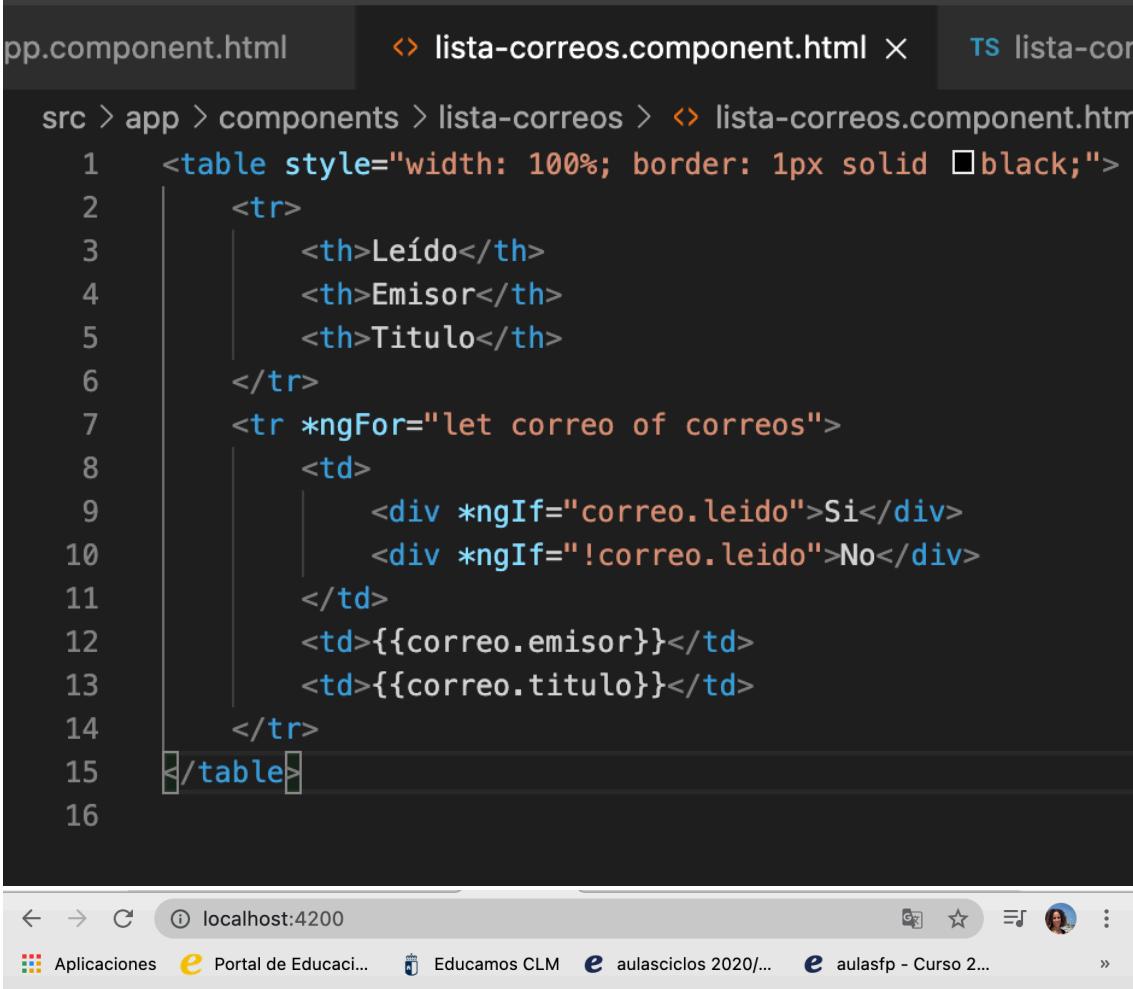
Links in the header: Aplicaciones, Portal de Educaci..., Educamos.

lista-correos works!

Vamos a modificarlo:



```
pp.component.html | lista-correos.component.html | lista-correos.component.ts | correo.component.html
src > app > components > lista-correos > lista-correos.component.ts > ListaCorreosComponent > correo.component.html
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-lista-correos',
5   templateUrl: './lista-correos.component.html',
6   styleUrls: ['./lista-correos.component.scss']
7 })
8 export class ListaCorreosComponent implements OnInit {
9
10   correos: any[];
11
12   constructor() {
13     const correo1 = [
14       titulo: "Titulo del 1",
15       cuerpo: `Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email,
16       Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo
17       Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo
18       emisor: 'correoEmisor1@openWebinar.inv',
19       destinatario: 'correoReceptor@openWebinar.inv',
20       leido: true,
21     ];
22     const correo2 = {
23       titulo: "Titulo del 2",
24       cuerpo: `Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email,
25       Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo del Email, Cuerpo
26       emisor: 'correoEmisor2@openWebinar.inv',
27       destinatario: 'correoReceptor@openWebinar.inv',
28       leido: false,
29     };
30     this.correos = [];
31     this.correos.push(correo1);
32     this.correos.push(correo2);
33   }
34
35   ngOnInit(): void {
36   }
37
38 }
```



```
pp.component.html < lista-correos.component.html < lista-correos.ts

src > app > components > lista-correos > lista-correos.component.html
1   <table style="width: 100%; border: 1px solid black;">
2     <tr>
3       <th>Leído</th>
4       <th>Emisor</th>
5       <th>Titulo</th>
6     </tr>
7     <tr *ngFor="let correo of correos">
8       <td>
9         <div *ngIf="correo.leido">Si</div>
10        <div *ngIf="!correo.leido">No</div>
11       </td>
12       <td>{{correo.emisor}}</td>
13       <td>{{correo.titulo}}</td>
14     </tr>
15   </table>
16
```

Leído	Emisor	Titulo
Si	correoEmisor1@openWebinar.inv	Titulo del 1
No	correoEmisor2@openWebinar.inv	Titulo del 2

Vamos a settear la información de leído y no leído mediante un radio button usando **ngModel**.

Si miramos la documentación de ngModel, nos indica que tenemos que incorporar módulos.



```
app.module.ts — EmailAngular
pp.component.html    ◊ lista-correos.component.html    TS app.module.ts ×    TS lista-correos.component.html
src > app > TS app.module.ts > AppModule
  1 import { BrowserModule } from '@angular/platform-browser';
  2 import { NgModule } from '@angular/core';
  3 import { FormsModule } from '@angular/forms';
  4
  5 import { AppRoutingModule } from './app-routing.module';
  6 import { AppComponent } from './app.component';
  7 import { CorreoComponent } from './components/correo/correo.component';
  8 import { ListaCorreosComponent } from './components/lista-correos/lista-correos.compone
  9
 10 @NgModule({
 11   declarations: [
 12     AppComponent,
 13     CorreoComponent,
 14     ListaCorreosComponent
 15   ],
 16   imports: [
 17     BrowserModule,
 18     AppRoutingModule,
 19     FormsModule
 20   ],
 21   providers: [],
 22   bootstrap: [AppComponent]
 23 })
 24 export class AppModule { }
 25

pp.component.html    ◊ lista-correos.component.html ×    TS app.module.ts    TS lista-correos.component.ts    TS
src > app > components > lista-correos > ◊ lista-correos.component.html > ...
  1 <table style="width: 100%; border: 1px solid black;">
  2   <tr>
  3     <th>Leído</th>
  4     <th>Emisor</th>
  5     <th>Título</th>
  6   </tr>
  7   <tr *ngFor="let correo of correos; index as i;">
  8     <td>
  9       <!--Para cada correo, genera un name para esos dos radiobuttons-->
 10       <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=true> Si
 11       <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=false> No
 12     </td>
 13     <td>{{correo.emisor}}</td>
 14     <td>{{correo.titulo}}</td>
 15   </tr>
 16 </table>
 17
 18 <!--Podemos ver cómo se modifica dinámicamente-->
 19 <h2>Estado de lectura del correo 1:</h2>
 20 <h3 *ngIf="correos[0].leido">Si</h3>
 21 <h3 *ngIf="!correos[0].leido">No</h3>
```



Leído	Emisor	Titulo
<input checked="" type="radio"/> Si <input type="radio"/> No	correoEmisor1@openWebinar.inv	Titulo del 1
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor2@openWebinar.inv	Titulo del 2

Estado de lectura del correo 1:

Si

USO DE FORMULARIOS REACTIVOS

- Vamos a crear un componente para poder crear correos nuevos desde un formulario y con algo de validación.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: zsh

inma@MacBook-Pro-de-Inmaculada components % ng g c nuevo-correo
CREATE src/app/components/nuevo-correo/nuevo-correo.component.scss (0 bytes)
CREATE src/app/components/nuevo-correo/nuevo-correo.component.html (27 bytes)
CREATE src/app/components/nuevo-correo/nuevo-correo.component.spec.ts (662 bytes)
CREATE src/app/components/nuevo-correo/nuevo-correo.component.ts (299 bytes)
UPDATE src/app/app.module.ts (783 bytes)
inma@MacBook-Pro-de-Inmaculada components %

nt.html | <> lista-correos.component.html | ↗ lista-correos.component.scss | TS app.module.ts ×

src > app > TS app.module.ts > AppModule
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4
5 import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7 import { CorreoComponent } from './components/correo/correo.component';
8 import { ListaCorreosComponent } from './components/lista-correos/lista-correos.component';
9 import { NuevoCorreoComponent } from './components/nuevo-correo/nuevo-correo.component';
10
11 @NgModule([
12   declarations: [
13     AppComponent,
14     CorreoComponent,
15     ListaCorreosComponent,
16     NuevoCorreoComponent
17   ],
18   imports: [
19     BrowserModule,
20     AppRoutingModule,
21     FormsModule,
22     ReactiveFormsModule
23   ],
24   providers: [],
25   bootstrap: [AppComponent]
26 ])
27 export class AppModule { }
```



```
↳ app.component.html ×   ↳ lista-correos.component.html    ↳ lista-correos.component.html
src > app > ↳ app.component.html > app-nuevo-correo
1   <router-outlet></router-outlet>
2   <app-nuevo-correo></app-nuevo-correo>

↳ app.component.html      TS nuevo-correo.component.ts ●   ↳ lista-correos.component.html
src > app > components > nuevo-correo > TS nuevo-correo.component.ts > NuevoCorreoComponent
1   import { Component, OnInit } from '@angular/core';
2   import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3
4   @Component({
5     selector: 'app-nuevo-correo',
6     templateUrl: './nuevo-correo.component.html',
7     styleUrls: ['./nuevo-correo.component.scss']
8   })
9   export class NuevoCorreoComponent implements OnInit {
10
11   nuevoCorreo: FormGroup;
12   submitted = false;
13
14   constructor(private formBuilder: FormBuilder) { }
15
16   ngOnInit() {
17     this.nuevoCorreo = this.formBuilder.group({
18       titulo: ['', [Validators.required, Validators.minLength(3)]],
19       cuerpo: ['', [Validators.required, Validators.minLength(10)]],
20       destinatario: ['', [Validators.required, Validators.email]],
21     });
22   }
23 }
```



```
24     |     get formulario() { return this.nuevoCorreo.controls; }
25
26     |     onSubmit() {
27     |         this.submitted = true;
28
29         |         if (this.nuevoCorreo.invalid) {
30         |             return;
31         |         }
32
33         let correo = this.nuevoCorreo.value;
34         correo.leido= false;
35         correo.emisor= 'correoEmisor1@prueba.com';
36
37         alert("Correo Enviado \nEliminamos el formulario");
38         this.onReset();
39     }
40
41     onReset() {
42         this.submitted = false;
43         this.nuevoCorreo.reset();
44     }
45
46
47 }
```

```
app.component.html      ts nuevo-correo.component.ts      nuevo-correo.component.html      lista-correos.component.html
src > app > components > nuevo-correo > nuevo-correo.component.html > form > div > div > div > div
1   <form [formGroup]="nuevoCorreo" (ngSubmit)="onSubmit()">
2     <div>
3       <div>
4         <label>Titulo</label>
5         <input type="text" formControlName="titulo" />
6         <div *ngIf="submitted && formulario.titulo.errors">
7           <div>El título: </div>
8           <div *ngIf="formulario.titulo.errors.required">No puede estar vacío</div>
9           <div *ngIf="formulario.titulo.errors.minLength">Deber contener más de 3 caracteres</div>
10
11
12       <div>
13         <label>Cuerpo</label>
14         <input type="text" formControlName="cuerpo" />
15         <div *ngIf="submitted && formulario.cuerpo.errors">
16           <div>El cuerpo: </div>
17           <div *ngIf="formulario.cuerpo.errors.required">No puede estar vacío</div>
18           <div *ngIf="formulario.cuerpo.errors.minLength">Deber contener más de 10 caracteres</div>
19
20
21     </div>
22   <div>
23     <label>Destinatario</label>
24     <input type="text" formControlName="destinatario" />
25     <div *ngIf="submitted && formulario.destinatario.errors">
26       <div *ngIf="formulario.destinatario.errors.required">El destinatario no puede estar vacío</div>
27       <div *ngIf="formulario.destinatario.errors.email">El destinatario debe ser un email válido</div>
28
29
30     <div>
31       <button>Enviar</button>
32       <button type="reset" (click)="onReset()">Cancelar</button>
33     </div>
34   </div>
35 </form>
```



← → ⌂ ⓘ localhost:4200

Aplicaciones Portal de Educaci... Educamos C

Titulo
Cuerpo
Destinatario

← → ⌂ ⓘ localhost:4200

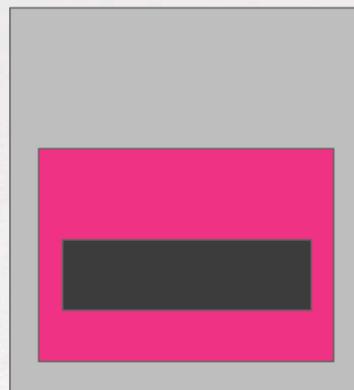
Aplicaciones Portal de Educaci...

Titulo
Cuerpo
El cuerpo:
Deber contener más de 10 caracteres
Destinatario
El destinatario debe ser un email válido

DATA BINDING (Comunicación padre-hijo)



Los componentes, al igual que los elementos html, tienen una jerarquía



Componente

Padre de



Abuelo de



Preparación del componente hijo

- Componente hijo debe disponer de variables
- Para usar un “template” para transmitir la información, necesitamos indicarlo en nuestro componente hijo
- Para indicar que una variable quiere usarse en el “template”
- Usamos el decorador `@Input()`
- Un **decorador** es una herramienta de Angular para indicar alguna configuración a tener en cuenta de forma dinámica (como al definir un componente con `@Component`)



- Declaramos una variable con el decorador delante, en nuestro componente hijo, para indicar qué esperamos recibir

```
@Input() variable: tipo;
```

- Declaramos una variable o método en nuestro componente hijo para poder acceder desde el padre

```
variable: tipo;
```

```
funcionDelHijo() {}
```

<https://angular.io/api/core/Input>

Formas de comunicación Padre → Hijo

- Usando el “template” con el propio HTML
- Usando TypeScript en el padre



En “template”

Vamos a editar la variable título de nuestro componente hijo, desde el “template” del componente padre:

```
<componente-hijo></componente-hijo>
```

Usando el valor directamente:

```
<componente-hijo  
    titulo="Mi Titulo">  
</componente-hijo>
```

Ahora en lugar de un valor, le indicaremos lo mismo pero usando una variable:

```
<componente-hijo  
    [titulo]="variableTitulo">  
</componente-hijo>
```



Mediante TypeScript

Lo usamos cuando:

- Queremos o necesitamos usar TypeScript para ello
- Necesitamos controlar el momento de enviar la información

Necesitamos una variable en el padre para poder acceder al hijo; para ello:

- Usamos el decorador `@ViewChild(selector)`
- Usamos el `selector` del componente hijo

`<componente-hijo></componente-hijo>`

Creamos la variable en el componente padre para poder acceder al hijo:

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

Ya podemos editar las variables en el componente hijo, con una función por ejemplo:

```
enviarTituloAlHijo(){
    this.componenteHijo.titulo = "Mi Titulo"
}
```



También podemos ejecutar métodos del componente hijo:

```
enviarTituloAlHijo(){  
    this.componenteHijo.funcionDelHijo();  
}
```

Vamos a ver cómo usar la comunicación entre componentes para crear.

Vamos a integrar el componente de nuevo correo con el componente de lista de correo que hemos creado anteriormente.

Vamos a crear la funcionalidad de enviar un correo mediante respuesta rápida, la diferencia con nuevo-correo será que como estamos respondiendo a un correo concreto, esa información de ese correo nos servirá para completar la información del nuevo correo (título, destinatario, ...).

- Vamos a modificar el componente de la lista, añadiendo un botón responder a cada correo de la lista.

```
src > app > components > lista-correos > lista-correos.component.html > ...  
1   <table style="width: 100%; border: 1px solid black;">  
2     <tr>  
3       <th>Leído</th>  
4       <th>Emisor</th>  
5       <th>Título</th>  
6       <th>Responder</th>  
7     </tr>  
8     <ng-container *ngFor="let correo of correos; index as i;">  
9       <tr>  
10         <td>  
11           <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=true disabled> Si  
12           <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=false disabled> No  
13         </td>  
14         <td>{{correo.emisor}}</td>  
15         <td>{{correo.titulo}}</td>  
16         <td>  
17           <button (click)="clickResponder(correo)">Responder</button>  
18         </td>  
19       </tr>  
20       <tr *ngIf="correo.responder">  
21         <app-nuevo-correo [correo]="correo"></app-nuevo-correo>  
22       </tr>  
23     </ng-container>  
24   </table>
```

- **ng-container** es un componente de Angular que en este caso me va a permitir iterar entre los correos y generar para cada correo dos filas, una con la información del correo y otra sólo en el caso de que se haya pulsado *Responder*.
Ng-container no genera ningún elemento en el DOM.



- En el .ts hemos añadido un par de correos más y nos hemos creado el método **clickResponser(correo)** que lo único que hace es cambiar el estado de la variable booleana creada para visualizar o no el formulario de respuesta.

```
src > app > components > lista-correos > lista-correos.component.ts > ListaCorreosComponent > constructor > emisor
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-lista-correos',
5   templateUrl: './lista-correos.component.html',
6   styleUrls: ['./lista-correos.component.scss']
7 })
8 export class ListaCorreosComponent implements OnInit {
9
10   correos: any[];
11   responder: boolean;
12
13   constructor() {
14     const correo1 = {
15       titulo: "Título Email 1",
16       cuerpo: `Cuerpo del email, Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
17       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
18       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email `,
19       emisor: 'correoEmisor@prueba.com',
20       receptor: 'correoReceptor@prueba.com',
21       leido: true
22     };
23     const correo2 = {
24       titulo: "Título Email 2",
25       cuerpo: `Cuerpo del email, Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
26       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
27       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email `,
28       emisor: 'correoEmisor@prueba.com',
29       receptor: 'correoReceptor@prueba.com',
30       leido: false
31     };
32
33     this.correos = [];
34     this.correos.push(correo1);
35     this.correos.push(correo2);
36
37     this.correos.push({
38       titulo: "Título Email 3",
39       cuerpo: `Cuerpo del email, Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
40       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
41       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email `,
42       emisor: 'correoEmisor3@prueba.com',
43       receptor: 'correoReceptor@prueba.com',
44       leido: false
45     });
46
47     this.correos.push([
48       titulo: "Título Email 4",
49       cuerpo: `Cuerpo del email, Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
50       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,
51       Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email,Cuerpo del email `,
52       emisor: 'correoEmisor3@prueba.com',
53       receptor: 'correoReceptor@prueba.com',
54       leido: false
55     ]);
56     this.responder = false;
57   }
58
59   ngOnInit(): void {
60   }
61
62   clickResponser(correo) {
63     correo.responder = !correo.responder;
64   }
65
66 }
67 }
```

- En *nuevo-correo.component.ts* crearemos una variable **correo** con **@Input** para poder recibir la información del padre, es decir del componente **lista-correos** y



así rellenar los valores correspondientes al **titulo** y al **destinatario** en el formulario de nuevo correo.

- Por otro lado, modificamos también el **onReset** para poner responder a false para que desaparezca el formulario una vez terminado de enviar o cuando pulsemos cancelar.

```
TS nuevo-correo.component.ts X  <--> nuevo-correo.component.html  TS lista-correos.component.ts
src > app > components > nuevo-correo > TS nuevo-correo.component.ts > NuevoCorreoComponent >
1 | import { Component, Input, OnInit } from '@angular/core';
2 | import { FormBuilder, FormGroup, Validators} from '@angular/forms';
3 |
4 | @Component({
5 |   selector: 'app-nuevo-correo',
6 |   templateUrl: './nuevo-correo.component.html',
7 |   styleUrls: ['./nuevo-correo.component.scss']
8 | })
9 | export class NuevoCorreoComponent implements OnInit {
10 |
11 |   nuevoCorreο: FormGroup;
12 |   submitted = false;
13 |   @Input() correo: any;
14 |
15 |   constructor(private formBuilder: FormBuilder) { }
16 |
17 |   ngOnInit(): void {
18 |     this.nuevoCorreο = this.formBuilder.group({
19 |       titulo: ['', [Validators.required, Validators.minLength(3)]],
20 |       cuerpo: ['', [Validators.required, Validators.minLength(10)]],
21 |       destinatario: ['', [Validators.required, Validators.email]]
22 |     });
23 |
24 |     if(this.correo != undefined) {
25 |       console.log("A", this.correo);
26 |       this.nuevoCorreο.patchValue({
27 |         titulo: 'Re: '+this.correo.titulo,
28 |         destinatario: this.correo.emisor
29 |       });
30 |     }
31 |   }
32 |
33 |   get formulario() { return this.nuevoCorreο.controls; }
34 | }
```



```
35  |     onSubmit() {
36  |       this.submitted = true;
37  |       if(this.nuevoCorreo.invalid) {
38  |         return;
39  |       }
40
41  |       let correo = this.nuevoCorreo.value;
42  |       correo.leido = false;
43  |       correo.emisor = "correoEmisor@prueba.com";
44
45  |       alert("Correo enviado\nEliminamos el formulario");
46  |       this.onReset();
47
48  |     }
49
50  |   onReset() {
51  |     this.submitted = false;
52  |     this.nuevoCorreo.reset();
53  |   }
54 }
```

Leído	Emisor	Título	Responder
<input checked="" type="radio"/> Si <input type="radio"/> No	correoEmisor@prueba.com	Título Email 1	<input type="button" value="Responder"/>
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor@prueba.com	Título Email 2	<input type="button" value="Responder"/>
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor3@prueba.com	Título Email 3	<input type="button" value="Responder"/>
Título Re: Título Email 3 Cuerpo Destinatario correoEmisor3@prueba. Enviar Cancelar <input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor3@prueba.com	Título Email 4	<input type="button" value="Responder"/>

COMUNICACIÓN HIJO-PADRE

- Usaremos Eventos para realizar la comunicación entre hijo y padre
- Veremos otros métodos más adelante



- **Eventos**

- Uso de eventos en el hijo
- Uso de eventos en el padre

Eventos

- Herramientas para “avisar” a otros
 - Requieren que previamente estén “escuchando”
 - Podemos transmitir información con ellos
-
- Angular dispone de eventos predefinidos para usar, como **click**, **change**, etc ...
 - También nos permite crear nuestros eventos personalizados con el decorador **@Output()** y el tipo **EventEmitter**



- Vamos a emitir un “aviso” en el hijo
- Vamos a escuchar el aviso del hijo en el padre con una función
- Cuando el hijo “avise”, se “dispara” la función del padre

<https://angular.io/api/core/Output>

<https://angular.io/api/core/EventEmitter>

- Eventos
- Uso de eventos en el hijo
- Uso de eventos en el padre

Componente hijo

Creamos un evento personalizado usando `@Output` y el tipo `EventEmitter`

`@Output()` hijoAvisando:

```
EventEmitter<any> = new EventEmitter();
```



“Disparamos” el evento usando la función `emit(value?: T)`

```
let mensaje = "hola";
this.hijoAvisando.emit(mensaje);
```

- Eventos
- Uso de eventos en el hijo
- **Uso de eventos en el padre**

Componente Padre

Primero crearemos una función en TypeScript para asignar al evento y que se active al “dispararse”

```
mensajeRecibido(mensaje: string){
    console.log(mensaje);
}
```

A continuación, en el “template”, asignamos la función del padre al evento del hijo

```
<componente-hijo
    (hijoAvisando)="mensajeRecibido($event)">
</componente-hijo>
```

- Vamos a modificar nuestro ejercicio para que cuando se envíe el correo o se cancele se cierre. Para ello lo vamos a hacer mediante eventos, es decir, el hijo



se comunicará con el padre mediante eventos (hay otras formas de hacerlo pero para practicar esto).

- Modificaremos el .ts de nuevo-correo añadiendo la variable **@Output** de tipo **EventEmitter**, posteriormente en el método **onReset**, tendremos que lanzar el evento (una vez que se ha enviado el formulario o cancelado porque pasa por ahí en ambos casos).

```
pp.component.html      TS nuevo-correo.component.ts ×  ➔ nuevo-correo.component.html  TS li
src > app > components > nuevo-correo > TS nuevo-correo.component.ts > NuevoCorreoComponent
1 | import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
2 | import { FormBuilder, FormGroup, Validators} from '@angular/forms';
3 |
4 | @Component({
5 |   selector: 'app-nuevo-correo',
6 |   templateUrl: './nuevo-correo.component.html',
7 |   styleUrls: ['./nuevo-correo.component.scss']
8 | })
9 | export class NuevoCorreoComponent implements OnInit {
10 |
11 |   nuevoCorreο: FormGroup;
12 |   submitted = false;
13 |   @Input() correo: any;
14 |   @Output() accionRealizada: EventEmitter<any> = new EventEmitter();
15 |
16 |   constructor(private formBuilder: FormBuilder) { }
17 |
18 |   ngOnInit(): void {
19 |     this.nuevoCorreο = this.formBuilder.group({
20 |       titulo: ['', [Validators.required, Validators.minLength(3)]],
21 |       cuerpo: ['', [Validators.required, Validators.minLength(10)]],
22 |       destinatario: ['', [Validators.required, Validators.email]]
23 |     });
24 |
25 |     if(this.correo != undefined) {
26 |       console.log("A", this.correo);
27 |       this.nuevoCorreο.patchValue({
28 |         titulo: 'Re: '+this.correo.titulo,
29 |         destinatario: this.correo.emisor
30 |       });
31 |     }
32 |   }
33 | }
```



```
34     get formulario() { return this.nuevoCorreo.controls; }
35
36     onSubmit() {
37         this.submitted = true;
38         if(this.nuevoCorreo.invalid) {
39             return;
40         }
41
42         let correo = this.nuevoCorreo.value;
43         correo.leido = false;
44         correo.emisor = "correoEmisor@prueba.com";
45
46         alert("Correo enviado\nEliminamos el formulario");
47         this.onReset();
48     }
49
50
51     onReset() {
52         this.submitted = false;
53         this.nuevoCorreo.reset();
54         this.accionRealizada.emit();
55     }
56 }
```

- Modificaremos también el .html de lista-correos, para que cuando añada el componente hijo nuevo-correo, asocie el evento (**accionRealizada**) con el método que lo manejará, en este caso **accionRespuestaRapida** que se implementará en el .ts del padre (lista-correos).

```
src > app > components > lista-correos > lista-correos.component.html > ...
1   <table style="width: 100%; border: 1px solid black;">
2     <tr>
3       <th>Leído</th>
4       <th>Emisor</th>
5       <th>Titulo</th>
6       <th>Responder</th>
7     </tr>
8     <ng-container *ngFor="let correo of correos; index as i;">
9       <tr>
10         <td>
11           <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=true disabled> Si
12           <input type="radio" name="leido-{{i}}" [(ngModel)]="correo.leido" [value]=false disabled> No
13         </td>
14         <td>{{correo.emisor}}</td>
15         <td>{{correo.titulo}}</td>
16         <td>
17           <button (click)="clickResponder(correo)">Responder</button>
18         </td>
19       </tr>
20     <tr *ngIf="correo.responder">
21       <app-nuevo-correo [correo]="correo" (accionRealizada)="accionRespuestaRapida(correo)"></app-nuevo-correo>
22     </tr>
23   </ng-container>
24 </table>
```

- En el fichero .ts de lista-correos, lo único que añadimos es el método **accionRespuestaRapida** que recibe el correo sobre el que estamos trabajando.

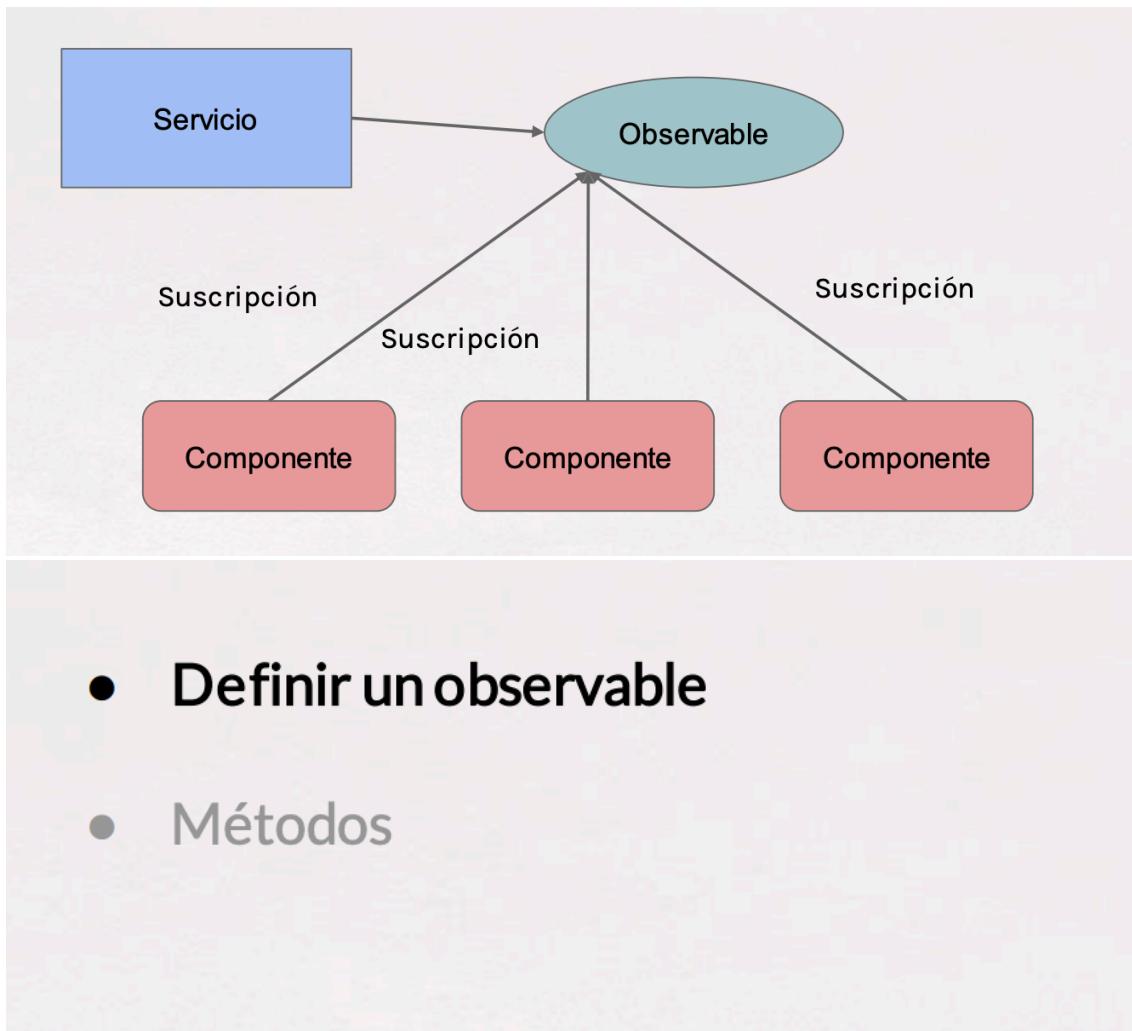


```
53 |     clickResponder(correo) {
54 |       correo.responder = !correo.responder;
55 |
56 |
57 |     accionRespuestaRapida(correo) {
58 |       correo.responder=false;
59 |     }
60 |
61 |   }
62 |
63 | }
```

OBSERVABLES (Rxjs)

Observables

- Tipo de dato
 - Pertenece a la librería Rxjs
 - “Canal” de comunicación, similar a los eventos
 - Permite suscribirse para recibir “avisos” y nueva información
-
- TypeScript (y JavaScript) son asíncronos
 - Realizar peticiones http, cargar svg, imágenes, componentes, librerías, etc ...
 - Nos permite sincronizar estas acciones



- **Definir un observable**
- Métodos

Definir un Observable

- Son tipos de datos muy comunes en Angular 8
- Devueltos por multitud de funciones de Angular y TypeScript (timers, servicios http, carga de elementos)
- Se usa **\$** al final de los nombres de variables para advertir su tipo



Declaración:

myObservable\$ = Observable<any[]>;

Aplicando un valor con un array:

myObservable\$ = of(1, 2, 3);

Observables

- Definir un observable
- Métodos

Suscripciones

Para indicar que acciones realizar, nos suscribimos a un observable:

```
myObservable$.subscribe(  
  (datos) => {  
    // acción al recibir datos  
    // Obligatoria  
  },  
  (error) => {},  
  () => {}  
);
```



Para indicar qué hacer en caso de error:

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

Para ejecutar cuando se finaliza el observable:

```
myObservable$.subscribe(  
    (datos) => {},  
    (error) => {},  
    () => {  
        // acción al finalizar  
        // Opcional  
    },  
);
```

Pipes

Nos permite aplicar cambios al resultado del Observable, antes de enviarse a los suscriptores

Nos ayudamos de la función **map** de Rxjs



```
myObservable$.pipe(  
    map(  
        data => {  
            // cambios a los datos  
        }  
    )  
);
```

Terminar suscripción

- Obligatorio en el ciclo de vida de componentes
- Siempre que nos suscribimos, debemos terminar la suscripción o se mantendrá activa, generando errores

Para terminar la suscripción usamos:

```
const mySubscription = myObservable$.subscribe(...);  
  
if(!mySubscription.closed){  
    mySubscription.unsubscribe();  
}
```

<https://angular.io/guide/observables>

<https://angular.io/guide/pipes>



SERVICIOS

- Forma de organizar código
- Engloba varios usos
- Describle como un componente sin “template” asociado

Son usados para:

- Obtener información (Proveedores)
- Compartir información (Compartir)
- Contener lógica (Inteligentes)

Para generar un nuevo servicio usando CLI de Angular, usaremos el comando:

`ng generate service <name> [options]`

O simplificado:

`ng g s <name> [options]`

<https://angular.io/cli/generate#service>

- Vamos a añadir servicios a nuestra práctica.
- Vamos a su vez a comenzar a organizar nuestras vistas y vamos a crear la vista de correo recibido que a su vez tiene el componente de lista de correo y éste a su vez el de nuevo correo.
- Vamos a crear un nuevo componente **avisos** que se comunicará con nuevo correo mediante un **servicio**.
- No se van a comunicar como anteriormente habíamos hecho con jerarquías de padres e hijos.



- En primer lugar lo que tenemos que hacer es crear el componente **avisos**, un **servicio** vacío y una nueva **vista** para incluir dichos componentes.
- Dentro de **app** debo tener las carpetas: **components, services y views**.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

inma@MacBook-Pro-de-Inmaculada components % ng g c avisos
CREATE src/app/components/avisos/avisos.component.scss (0 bytes)
CREATE src/app/components/avisos/avisos.component.html (21 bytes)
CREATE src/app/components/avisos/avisos.component.spec.ts (626 bytes)
CREATE src/app/components/avisos/avisos.component.ts (276 bytes)
UPDATE src/app/app.module.ts (921 bytes)
inma@MacBook-Pro-de-Inmaculada components %

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

inma@MacBook-Pro-de-Inmaculada services % ng g s avisos
CREATE src/app/services/avisos.service.spec.ts (357 bytes)
CREATE src/app/services/avisos.service.ts (135 bytes)
inma@MacBook-Pro-de-Inmaculada services %

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh

inma@MacBook-Pro-de-Inmaculada views % ng g c correos-recibidos
CREATE src/app/views/correos-recibidos/correos-recibidos.component.scss (0 bytes)
CREATE src/app/views/correos-recibidos/correos-recibidos.component.html (32 bytes)
CREATE src/app/views/correos-recibidos/correos-recibidos.component.spec.ts (697 bytes)
CREATE src/app/views/correos-recibidos/correos-recibidos.component.ts (319 bytes)
UPDATE src/app/app.module.ts (1051 bytes)
inma@MacBook-Pro-de-Inmaculada views %
```

- Este es el aspecto del .ts del servicio inicialmente:

```
src > app > services > TS avisos.service.ts > ...
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class AvisosService {
7
8   constructor() { }
9 }
10
```

- Vamos a integrar todos los componentes en la vista y posteriormente lo incluimos en **app.component.html**.



Vista Correos Recibidos

Leído	Emisor	Título	Responder
<input checked="" type="radio"/> Si <input type="radio"/> No	correoEmisor@prueba.com	Título Email 1	<button>Responder</button>
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor@prueba.com	Título Email 2	<button>Responder</button>
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor3@prueba.com	Título Email 3	<button>Responder</button>
<input type="radio"/> Si <input checked="" type="radio"/> No	correoEmisor3@prueba.com	Título Email 4	<button>Responder</button>

avisos works!

- Completamos la parte del componente avisos para que muestre un mensaje y luego lo oculte.

```

ent.html      TS lista-correos.component.ts      < avisos.component.html > 1
src > app > components > avisos > avisos.component.html > div
1   <div *ngIf="visible">
2     <p>Aviso:</p>
3     <p>{{mensaje}}</p>
4   </div>

```



```
-correos.component.ts    ◊ avisos.component.html    TS avisos.component.ts X ◊
src > app > components > avisos > TS avisos.component.ts > AvisosComponent > hideMessage()
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-avisos',
5   templateUrl: './avisos.component.html',
6   styleUrls: ['./avisos.component.scss']
7 })
8 export class AvisosComponent implements OnInit {
9
10   mensaje: string;
11   visible: boolean;
12
13   constructor() {
14     this.mensaje = "Correo enviado";
15     this.visible = false;
16   }
17
18   ngOnInit(): void {
19     this.showMessage('Correo Enviado');
20   }
21
22   showMessage(mensaje: string) {
23     this.mensaje = mensaje;
24     this.visible = true;
25     this.waitToHide();
26   }
27
28   hideMessage(){
29     this.visible = false;
30     this.mensaje = '';
31   }
32
33   waitToHide() {
34     setTimeout(() => {
35       this.hideMessage();
36     }, 2000);
37   }
38 }
```

- Ahora que hemos probado que funciona, pasamos toda la lógica al servicio de avisos, lo importamos con los constructores de avisos y nuevo-correo para que puedan acceder a él. Ahora al simular el envío de un correo, aparece el aviso.



```
 avisos.component.html      TS avisos.component.ts      TS avisos.service.ts ×  
src > app > services > TS avisos.service.ts > AvisosService > constructor  
1   import { Injectable } from '@angular/core';  
2  
3   @Injectable({  
4     providedIn: 'root'  
5   })  
6   export class AvisosService {  
7  
8     mensaje: string;  
9     visible: boolean;  
10  
11    constructor() {  
12      this.mensaje = '';  
13      this.visible = false;  
14    }  
15  
16    showMessage(mensaje: string) {  
17      this.mensaje = mensaje;  
18      this.visible = true;  
19      this.waitToHide();  
20    }  
21  
22    hideMessage(){  
23      this.visible = false;  
24      this.mensaje = '';  
25    }  
26  
27    waitToHide() {  
28      setTimeout(() => {  
29        this.hideMessage();  
30      }, 2000);  
31    }  
32  }  
33
```



```
aviso.component.html TS avisos.component.ts X TS avisos.service.ts
src > app > components > avisos > TS avisos.component.ts > ...
2 import { AvisosService } from 'src/app/services/avisos.service';
3
4 @Component({
5   selector: 'app-avisos',
6   templateUrl: './avisos.component.html',
7   styleUrls: ['./avisos.component.scss']
8 })
9 export class AvisosComponent implements OnInit {
10
11
12   constructor(private servicioAvisos:AvisosService) {}
13
14   ngOnInit(): void {}
15
16 }
17
```

```
TS nuevo-correo.component.ts X <--> nuevo-correo.component.html TS lista-correos.component.ts <--> av...
src > app > components > nuevo-correo > TS nuevo-correo.component.ts > NuevoCorreoComponent > ngOnInit()
1 import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
2 import { FormBuilder, FormGroup, Validators} from '@angular/forms';
3 import { AvisosService } from 'src/app/services/avisos.service';
4
5 @Component({
6   selector: 'app-nuevo-correo',
7   templateUrl: './nuevo-correo.component.html',
8   styleUrls: ['./nuevo-correo.component.scss']
9 })
10 export class NuevoCorreoComponent implements OnInit {
11
12   nuevoCorreο: FormGroup;
13   submitted = false;
14   @Input() correo: any;
15   @Output() accionRealizada: EventEmitter<any> = new EventEmitter();
16
17   constructor(private formBuilder: FormBuilder, private servicioAvisos:AvisosService) { }
18
19   ngOnInit(): void {
20     this.nuevoCorreο = this.formBuilder.group({
21       titulo: ['', [Validators.required, Validators.minLength(3)]],
22       cuerpo: ['', [Validators.required, Validators.minLength(10)]],
23       destinatario: ['', [Validators.required, Validators.email]]
24     });
25
26     if(this.correo != undefined) {
27       this.nuevoCorreο.patchValue([
28         { titulo: 'Re: '+this.correo.titulo,
29         destinatario: this.correo.emisor
30       });
31     }
32   }
33
34   get formulario() { return this.nuevoCorreο.controls; }
35 }
```



```
36  onSubmit() {
37      this.submitted = true;
38      if(this.nuevoCorreo.invalid) {
39          return;
40      }
41
42      let correo = this.nuevoCorreo.value;
43      correo.leido = false;
44      correo.emisor = "correoEmisor@prueba.com";
45
46      this.onReset();
47      this.servicioAvisos.showMessage("Correo enviado")
48
49  }
50
51  onReset() {
52      this.submitted = false;
53      this.nuevoCorreo.reset();
54      this.accionRealizada.emit();
55  }
56 }
```

```
-correos.component.ts    avisos.component.html ×  TS avisos.compone
src > app > components > avisos > avisos.component.html > div
1   <div *ngIf="servicioAvisos.visible">
2     <p>Aviso:</p>
3     <p>{{servicioAvisos.mensaje}}</p>
4   </div>
```

- Finalmente incluimos una función para cuando cancelamos el envío de correo e incluimos el destinatario en el mensaje a enviar.



```
TS nuevo-correo.component.ts ×  < nuevo-correo.component.html  TS lista-corre  ⏴
src > app > components > nuevo-correo > TS nuevo-correo.component.ts > ✎ NuevoCorreoComponent
  1   import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
  2   import { FormBuilder, FormGroup, Validators} from '@angular/forms';
  3   import { AvisosService } from 'src/app/services/avisos.service';
  4
  5   @Component({
  6     selector: 'app-nuevo-correo',
  7     templateUrl: './nuevo-correo.component.html',
  8     styleUrls: ['./nuevo-correo.component.scss']
  9   })
10  export class NuevoCorreoComponent implements OnInit {
11
12    nuevoCorreo: FormGroup;
13    submitted = false;
14    @Input() correo: any;
15    @Output() accionRealizada: EventEmitter<any> = new EventEmitter();
16
17  constructor(private formBuilder: FormBuilder, private servicioAvisos:A
18
19  ngOnInit(): void {
20    this.nuevoCorreo = this.formBuilder.group({
21      titulo: ['', [Validators.required, Validators.minLength(3)]],
22      cuerpo: ['', [Validators.required, Validators.minLength(10)]],
23      destinatario: ['', [Validators.required, Validators.email]]
24    });
25
26    if(this.correo != undefined) {
27      this.nuevoCorreo.patchValue({
28        titulo: 'Re: '+this.correo.titulo,
29        destinatario: this.correo.emisor
30      });
31    }
32  }
33
34  get formulario() { return this.nuevoCorreo.controls; }
35
```



```
35
36     onSubmit() {
37         this.submitted = true;
38         if(this.nuevoCorreo.invalid) {
39             return;
40         }
41
42         let correo = this.nuevoCorreo.value;
43         correo.leido = false;
44         correo.emisor = "correoEmisor@prueba.com";
45
46         this.onReset();
47         this.servicioAvisos.showMessage(`Correo enviado a ${correo.emisor}`);
48
49     }
50
51     onReset() {
52         this.submitted = false;
53         this.nuevoCorreo.reset();
54         this.accionRealizada.emit();
55     }
56
57     cancel() {
58         this.onReset();
59         this.servicioAvisos.showMessage("Envio cancelado");
60     }
61 }
62 }
```



```
TS nuevo-correo.component.ts      <-- nuevo-correo.component.html x   TS lista-correos.component.ts    <-- avisos
src > app > components > nuevo-correo > nuevo-correo.component.html > form
  1  <form [formGroup]="nuevoCorreo" (ngSubmit)="onSubmit()">
  2    <div>
  3      <div>
  4        <label>Título</label>
  5        <input type="text" formControlName="titulo" />
  6        <div *ngIf="submitted && formulario.titulo.errors">
  7          <div>El título: </div>
  8          <div *ngIf="formulario.titulo.errors.required">No puede estar vacío</div>
  9          <div *ngIf="formulario.titulo.errors.minLength">Debe contener más de 3 caracteres</div>
10        </div>
11      </div>
12      <div>
13        <label>Cuerpo</label>
14        <input type="text" formControlName="cuerpo" />
15        <div *ngIf="submitted && formulario.cuerpo.errors">
16          <div>El cuerpo: </div>
17          <div *ngIf="formulario.cuerpo.errors.required">No puede estar vacío</div>
18          <div *ngIf="formulario.cuerpo.errors.minLength">Debe contener más de 10 caracteres</div>
19        </div>
20      </div>
21    </div>
22    <div>
23      <label>Destinatario</label>
24      <input type="text" formControlName="destinatario" />
25      <div *ngIf="submitted && formulario.destinatario.errors">
26        <div *ngIf="formulario.destinatario.errors.required">El destinatario no puede estar vacío</div>
27        <div *ngIf="formulario.destinatario.errors.email">El destinatario debe ser un email válido</div>
28      </div>
29    </div>
30    <div>
31      <button>Enviar</button>
32      <button type="reset" (click)="cancel()">Cancelar</button>
33    </div>
34  </form>
```

- En el botón reset hemos añadido que al hacer click se llame a la función cancel().

OBTENER TOKEN PARA GOOGLE Y GMAIL

- Necesitamos el identificador de google para que nuestra aplicación se pueda autenticar con Google. Que nuestros usuarios puedan entrar con su usuario de Google en nuestra aplicación.
- Tenemos que tener una cuenta de correo de Gmail.
- Tenemos que entrar en la URL: <https://console.developers.google.com/>
- Lo primero será crear un proyecto.



⚠ Te quedan 24 projects en la cuota. Solicita un aumento o elimina proyectos. [Más información](#)

[MANAGE QUOTAS](#)

Nombre de proyecto * ?

ID del proyecto: miproyectoangular11. No se puede cambiar más adelante. [EDITAR](#)

Ubicación * [EXPLORAR](#)

Carpeta u organización principal

[CREAR](#) [CANCELAR](#)

- Ahora tenemos que habilitar APIs y servicios. Tenemos que habilitar el API de GMAIL para usar Gmail desde nuestra cuenta.

APIs MiProyectoAngular11 ▾ 🔍 ? 📲

Servicios	APIs y servicios	+ HABILITAR APIs Y SERVICIOS
Control	! Todavía no tienes ninguna API que usar. Para empezar, haz clic en "Activar API" o ve a la biblioteca de APIs .	
S		
consentimiento ...	Buscar <input type="text" value="gmail"/> 🔍	
por	2 resultados	
CORÍA	Gmail API Google Flexible, RESTful access to the user's inbox	
electrónico (1)	Gmail Postmaster Tools API Google The Gmail Postmaster API is a RESTful API that provides programmatic access to the Gmail Postmaster interface.	
(1)		

- Esto es para conectarnos con Google y que nos permita pedir al usuario permisos para poder realizar acciones.
- Vamos ahora a crear las pantallas de consentimiento.



≡ **Google APIs** • MiProyectoAngular11 ▾

PI APIs y servicios	Pantalla de consentimiento de OAuth
❖ Panel de control	Elige cómo quieras configurar y registrar la aplicación, incluidos los usuarios objetivo. Solo puedes vincular una aplicación a tu proyecto.
☰ Biblioteca	
▷ Credenciales	
🕒 Pantalla de consentimiento ...	User Type
☒ Verificación del dominio	<input type="radio"/> Internos ?
≡ Acuerdos de uso de páginas	Solo estará disponible para los usuarios que pertenezcan a tu organización. No hará falta que envíes tu aplicación para verificarla.
	<input checked="" type="radio"/> Externos ?
	Estará a disposición de cualquier usuario de prueba con una cuenta de Google. La aplicación se iniciará en modo de prueba y solo estará disponible para los usuarios que añadas a la lista de usuarios de prueba. Cuando la aplicación esté lista para enviarla a producción, es posible que tengas que verificarla .
	CREAR

Google APIs • MiProyectoAngular11 ▾

APIs y servicios	Editar registro de aplicación 🔗
Panel de control	1 Pantalla de consentimiento de OAuth — 2 Permisos —
Biblioteca	3 Usuarios de prueba — 4 Resumen
Credenciales	
Pantalla de consentimiento ...	Información de la aplicación
Verificación del dominio	Aparece en la pantalla de consentimiento y ayuda a que los usuarios finales te identifiquen y se pongan en contacto contigo
Acuerdos de uso de páginas	<p>Nombre de la aplicación * <input type="text" value="MiProyectoAngular11"/></p> <p>Nombre de la aplicación que solicita el consentimiento</p> <p>Correo electrónico de la asistencia al usuario * <input type="text" value="inmagijon@gmail.com"/></p> <p>Los usuarios lo utilizarán para ponerte en contacto contigo y resolver dudas sobre su consentimiento</p> <p>Logotipo de la aplicación EXPLORAR</p> <p>Sube una imagen que no ocupe más de 1 MB en la pantalla de consentimiento para ayudar a los usuarios a reconocer tu aplicación. Los formatos de imagen admitidos son JPG, PNG y BMP. Para conseguir unos resultados óptimos, los logotipos deben ser cuadrados y de 120x120 píxeles.</p>

- Guardamos y al darle a siguiente establecemos los permisos:

- Gmail.labels
- Gmail.readonly
- Gmail.send



			Interactúa con el complemento
<input type="checkbox"/>	Gmail API	.../auth/gmail.addons.current.message.action	Consultar mensajes de correo electrónico cuando se interactúa con el complemento
<input checked="" type="checkbox"/>	Gmail API	.../auth/gmail.readonly	Consultar tus mensajes de correo electrónico y la configuración
<input type="checkbox"/>	Gmail API	.../auth/gmail.metadata	Ver los metadatos de un mensaje de correo electrónico, como las etiquetas y los encabezados, pero no el cuerpo
<input type="checkbox"/>	Gmail API	.../auth/gmail.insert	Introducir correo en el buzón
<input type="checkbox"/>	Gmail API	.../auth/gmail.addons.current.message.metadata	Ver los metadatos de un mensaje de correo electrónico cuando se está ejecutando el complemento
<input type="checkbox"/>	Gmail API	.../auth/gmail.addons.current.message.readonly	Consultar los mensajes de correo electrónico cuando se está ejecutando el complemento
<input checked="" type="checkbox"/>	Gmail API	.../auth/gmail.send	Enviar correo electrónico en tu nombre
<input type="checkbox"/>	Gmail API	.../auth/gmail.full_access	New Service: https://www.googleapis.com/auth/gmail.full_access
<input checked="" type="checkbox"/>	Gmail API	.../auth/gmail.labels	Gestionar etiquetas de buzón
<input type="checkbox"/>	Gmail API	.../auth/gmail.settings.basic	Administrar la configuración del correo básico

3 Usuarios de prueba — 4 Resumen

Usuarios de prueba

Mientras el estado de publicación aparezca como "En prueba", solo podrán acceder a la aplicación los usuarios de prueba. El límite de usuarios permitido antes de proceder a la verificación de la aplicación es de 100 y se calcula a lo largo de la vida de la aplicación. [Más información](#)

[+ ADD USERS](#)

2 usuarios (2 usuarios de prueba, 0 de otro tipo) / límite de usuarios: 100



Filtrar tabla



⚠ Para evitar un uso inadecuado, se pueden añadir usuarios, pero no quitarlos

Información del usuario

igijoninf@cifpvirgendegracia.com

igijoninf@gmail.com



The screenshot shows the Google APIs console interface. At the top, there's a navigation bar with the Google logo, the text "APIs", a dropdown menu "MiProyectoAngular11", a search icon, and a help icon. Below the navigation is a sidebar with links like "Mis servicios", "Gmail API", "Visión general", "Credenciales", "Solicitudes", and "Historial". The main content area has a title "Visión general" and a button "INHABILITAR API". A callout box contains the text "Es posible que necesites credenciales para usar esta API. Haz clic en credenciales para empezar." with a "CREAR CREDENCIALES" button. To the right, there's a section titled "Tráfico por código de respuesta" with a chart and the text "Solicitudes por segundo (promedio de 2 h)".

- En dominios autorizados debemos añadir localhost:4200 que es nuestra url de desarrollo.



APIs MiProyectoAngular11 ▾

🔍 ? 📡 ⋮

Servicios	Crear ID de cliente de OAuth
Control	<p>necesitará un ID de cliente en cada una de ellas. Para obtener más información, consulta este artículo sobre cómo configurar OAuth 2.0.</p> <p>Tipo de aplicación *</p> <p>Aplicación web</p> <p>Más información sobre los tipos de clientes de OAuth</p> <p>Nombre *</p> <p>MiProyectoAngular11</p> <p>El nombre de tu cliente de OAuth 2.0. Este nombre solo se utiliza para identificar el cliente en la consola y no se mostrará a los usuarios finales.</p> <p>i Los dominios de los URI que especifiques más abajo se añadirán automáticamente a tu pantalla de consentimiento de OAuth como dominios autorizados.</p>
consentimiento ...	
del dominio	
uso de páginas	

Orígenes de JavaScript autorizados

Para usarse en solicitudes desde un navegador

URIs

http://localhost:4200

+ AÑADIR URI

URI de redirección autorizados

Para usarse con las solicitudes de un servidor web

+ AÑADIR URI

> environments

- Esto nos generará un id de cliente que utilizaremos a continuación.

CONECTAR CON GOOGLE USANDO OAUTH

- La conexión con OAuth es muy común y vamos a utilizar una librería para ello: **ng-gapi**
- Nos ayuda a que se establezca la conexión con Google y obtener el token del usuario para que podamos obtener información de ese usuario.
- **npm install ng-api –save** (para que se register en nuestro proyecto como módulo necesario)

<https://www.npmjs.com/package/ng-gapi>



- Vamos a iniciar la integración con OAuth, creando para su uso un servicio llamado login y un componente con el mismo nombre.