Introducción Angular

Jesús Rodríguez – Desarrollador Aplicaciones

INDICE

1. Componentes y directivas estructurales
2. Crear entorno local en Angular
3. Estructura del proyecto
4. Bootstrap 4 en Angular
5. Separar HTML del componente
6. Crear componentes de forma automática
7. Directivas *ngIf y *ngFor
8. Routes
9. Servicios

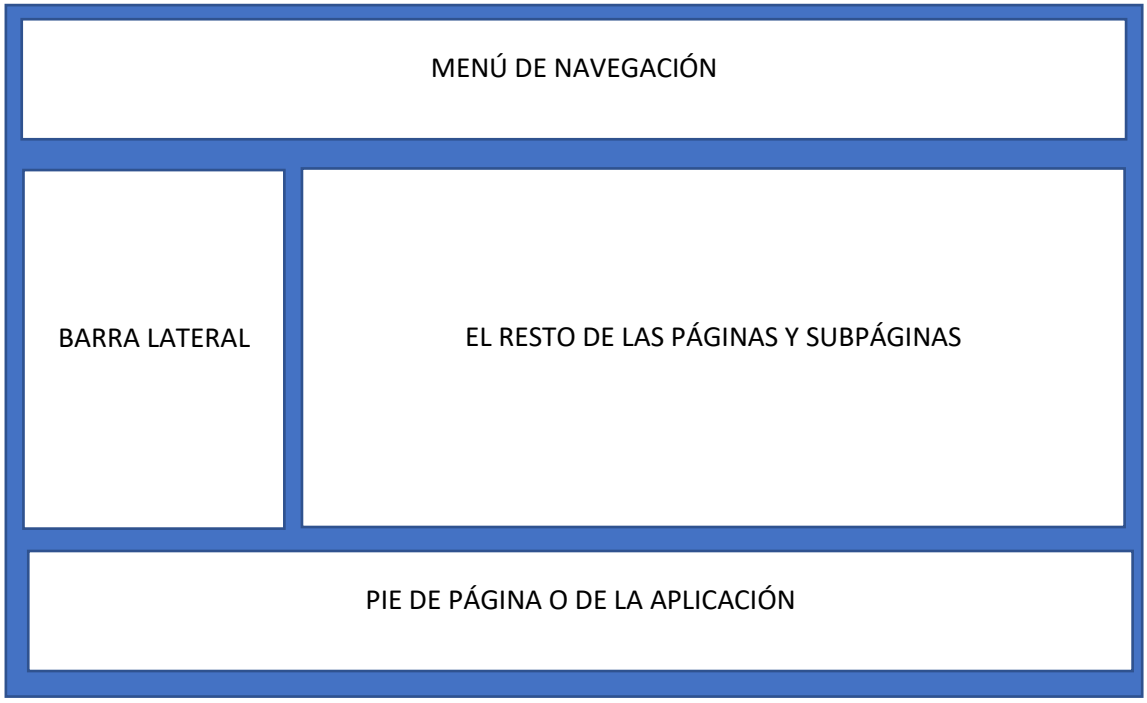
Componentes y directivas estructurales

Antes de nada, debemos saber dos diferencias sobre Angular:

- Cuando se habla de **AngularJS** se habla de la primera versión que salió.
- Cuando se dice **Angular** estamos mencionando a cualquier versión actual de **Angular** a partir de la 2.

Una buena **aplicación desarrollada en Angular**, es aquella que está formada por **múltiples componentes**.

Los **componentes** son pequeñas **clases** que cumple una **funcionalidad específica**. Por ejemplo, un componente podría ser em menú de navegación, a continuación, veremos una imagen formada por varios componentes.



Componentes:

- **Menú de navegación**
- **Barra lateral:** dónde podemos encontrar nuestras opciones de menú, por ejemplo.
- **El resto de las páginas y subpáginas:** Sería el componente que se encargaría de mostrar la información que queremos de nuestra aplicación, es decir, sería el encargado de mostrar una perspectiva u otra de la aplicación en relación con las funcionalidades que nosotros queramos adoptar
- **Pie de página o de la aplicación**

La **idea** de **reestructurar** la pantalla a partir de **componentes** debe de responder por la **funcionalidad** específica que se ha **desarrollado** al realizar una acción **sobre el componente en el que estemos interactuando**.

Además, tenerlo **estructurado por componentes no hace más fácil el desarrollo y cambios a futuro**, debido a que, si debemos realizar un cambio o nuevo desarrollo en el menú/barra lateral, iremos a dicho componente y solo se desarrollada o modificará dicho fichero.

Un **componente** en **Angular** es **igual** que una **clase normal**, lo único que lo diferencian es que **tienen un decorar específico**.

Las **directivas estructurales** en Angular son instrucciones que se indican en el **HTML**, es decir, a partir de estas directivas el HTML actuará de una manera u otra. Ejemplos de directivas:

- **ngIf:** Es la que se encarga de ocultar o mostrar elementos HTML en la página web. **¡OJO!** Esta directiva tiene una particularidad, si la directiva ngIf está a **true**, el elemento se va a mostrar en la página web, pero en caso de valer **false**, el elemento ni siquiera va a existir en el **HTML** cuando visualicemos la página, pero **Angular** seguirá manteniendo una relación con dicho elemento para saber cuándo debe mostrarlo y cuando no.
- **ngFor:** Es la encargada de realizar repeticiones de elementos **HTML** en nuestra página. Es decir, una tabla está formada por filas dónde se deben mostrar **X elementos**, para ello utilizamos un **ngFor** asociado a un elemento array, entonces la tabla se irá montando en base a los elementos que tengamos en dicho **array**.

A continuación, podemos ver el decorador @Component en una imagen dónde explicaremos como se comopone:

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent  {
  name = 'Angular 6';
}
```

Podemos ver, que una de la propiedad del componente es selector definido como **my-app** podemos encontrarlo en el **index.html**. Esto lo que realiza básicamente es **informar al navegador** que cuando esté **cargando nuestra página web** y encuentre la etiqueta **my-app renderice** todo lo que se encuentre en la imagen anterior, es decir, la clase que define el **componente**.

Como hemos dicho antes un componente es igual que una clase, solo que se define el decorador **@Component**, ya que si borramos todo se quedaría como una clase normal.

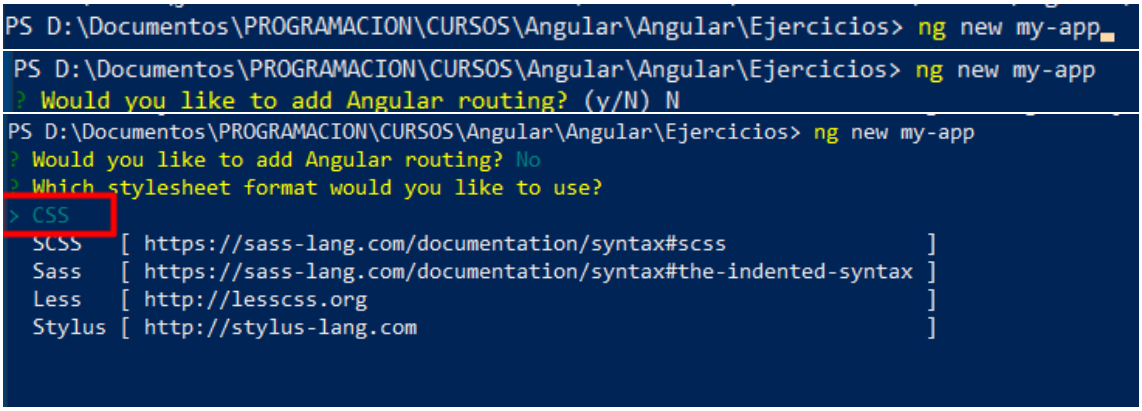
Como configuramos un componente, como veremos a continuación es bastante sencillo, en relación a la imagen anterior, un componente está formado por:

El **selector** que es el encargado de informar a **Angular** que debe renderizar lo que se encuentre en el html que está indicado en **templateUrl** y el **styleUrls** son los estilos que se van a aplicar únicamente a este apartado **.html**.

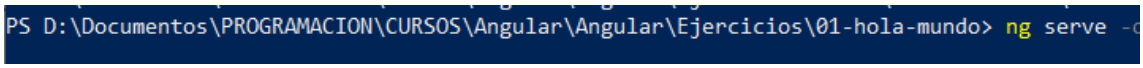
Crear entorno local en Angular

Lo primero que debemos de hacer es acceder a la siguiente página https://angular.io/start.

1. Debemos instalar Angular CLI en caso de no tenerlo instalado, entrando en la siguiente página https://angular.io/cli.
2. Una vez tenemos instalado Angular CLI, podemos proceder a crear nuestro nuevo proyecto, para ello debemos ejecutar el siguiente comando:

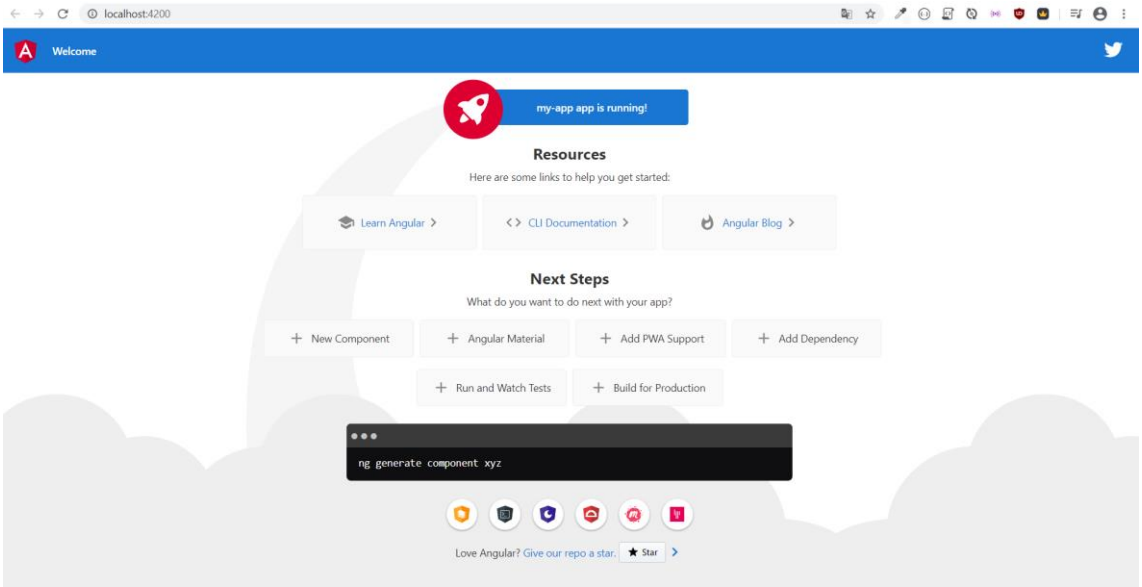


   Con esto ya tenemos nuestro proyecto, con su paquetería y módulos descargado, por lo tanto, ya podríamos comenzar nuestro proyecto.
3. Ahora procedemos a arrancar nuestro proyecto de Angular, para ello ejecutamos el siguiente comando **¡MUY IMPORTANTE!** Para arrancarlo debemos situarnos en la raíz de nuestro proyecto:
   a. **ng serve** es el comando que debemos ejecutar para arrancar nuestro proyecto, este comando levantará la aplicación en local en el puerto **4200** por defecto, en caso de que queramos especificar el puerto debemos ejecutar el siguiente comando **ng serve -p NºPUERTO**, si es la primera vez que ejecutamos el comando ejecutamos **ng serve -o**.



```
PS D:\Documentos\PROGRAMACION\CURSOS\Angular\Angular\Ejercicios\01-hola-mundo> ng serve -o
? Would you like to share anonymous usage data about this project with the Angular Team at
Google under Google's Privacy Policy at https://policies.google.com/privacy? For more
details and how to change this setting, see http://angular.io/analytics. Yes

Thank you for sharing anonymous usage data. Would you change your mind, the following
command will disable this feature entirely:

    ng analytics project off

0% compiling
Compiling @angular/core : es2015 as esm2015

Compiling @angular/common : es2015 as esm2015

Compiling @angular/platform-browser : es2015 as esm2015

Compiling @angular/platform-browser-dynamic : es2015 as esm2015

chunk {main} main.js, main.js.map (main) 57.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 140 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.75 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.7 MB [initial] [rendered]
Date: 2020-02-29T13:53:23.008Z - Hash: 35d79d56b8c8e1e375bb - Time: 24466ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.

Date: 2020-02-29T13:53:25.741Z - Hash: 35d79d56b8c8e1e375bb
5 unchanged chunks

Time: 2123ms
: Compiled successfully.
```

4. Ya tenemos levantando nuestra página en local en el puerto 4200 y está todo listo para poder empezar nuestro desarrollo.



Estructura del proyecto

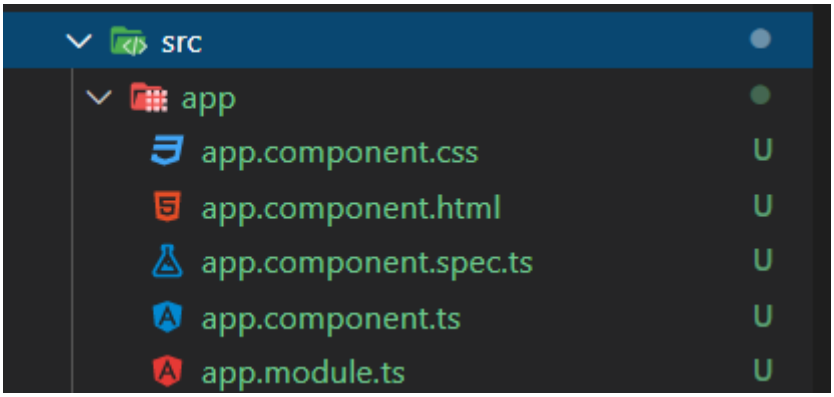
La estructura de nuestro proyecto es la siguiente:

La carpeta **e2e**, es quella que está destinada al manejo de funcionalidad de pruebas de extremo a extremo, es decir, **end to end**. Esta carpeta es configurada de forma automática por el **AngularCLI**.

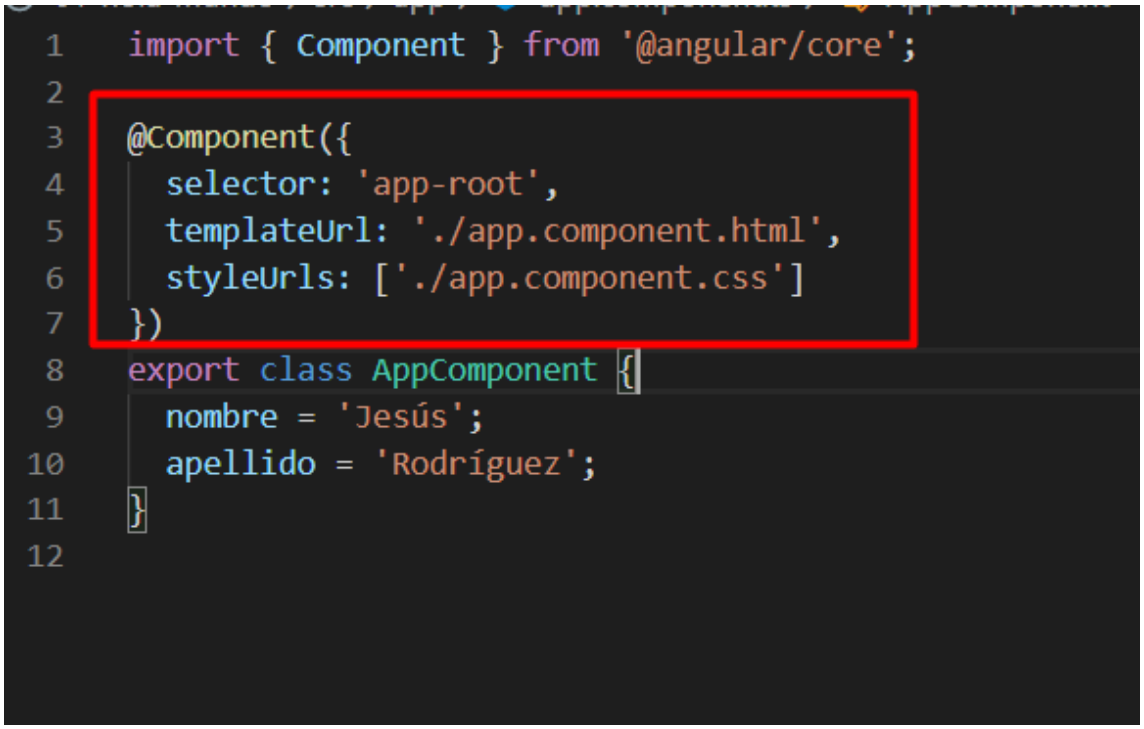
La carpeta **node_modules**, es la cantidad de package que se instalan cuando se crea un proyecto y sirven para muchísimas utilidades, como, por ejemplo:

- Escuchar los cambios de nuestra aplicación.
- Montar un observer.
- Todas estas funcionalidades se instalan con el fin de desarrollo. (Esta carpeta no debe subirse a los repositorios, ya que es bastante pesada). Lo único que se debe hacer cuando la vuelvas a descargar en otro dispositivo es lanzar el **comando npm install**.
- El fichero **.editroconfig** simplemente son configuraciones del editor y cuando se carga la aplicación coge un par de tips de dichos fichero.
- El fichero **.gitignore** es para omitir todos aquellos archivos que no deseamos subir a nuestro repositorio. Por lo tanto, todos los ficheros que no son de nuestra aplicación, como, por ejemplo, puede ser la carpeta /node_modules no se subiría a dicho repositorio.

- El fichero **angular.json**, es el fichero que le dice a angular como es nuestra aplicación y como funciona. Hay algunos parámetros que vamos a modificar como por ejemplo assets, styles y scripts.
- El fichero **package-lock.json** le dice a nuestra aplicación de node como fue creado el fichero **package.json**, este fichero no lo vamos a modificar ya que se configura de forma automática. Además, conforme nosotros vayamos creando cosas se va a ir volcando a este fichero.
- El fichero **package.json** es muy importante y no se modifica, en caso de hacerlo se debe estar muy seguro de lo que se está modificando.
- El fichero **README.md**, normalmente se crea de forma automática y explica como funciona la aplicación. Aquí es dónde nosotros podemos crear documentación de la aplicación. Este fichero se puede modificar como queramos.
- El fichero **tsconfig.json**, es el fichero que le indica a typescript en que estándar necesita trabajar la aplicación.
- El fichero **tslint.json**, este fichero normalmente no se toca y nos informa de cualquier error que tengamos, esto nos facilita realizar un código más limpio.
- En la carpeta **src**, es dónde se encuentra nuestra aplicación y dónde pasaremos la mayor parte del tiempo.
    - El fichero **index.html** es una pagina web común y corriente, la única peculiaridad es que contiene la etiqueta **app-root,** la definición de la etiqueta **app-root** se encuentra en el fichero **app.component.ts**.
    - El fichero **app.component** es el primer fichero que nuestra aplicación va a cargar.



        - Como vemos nos encontramos 4 ficheros que son **app.component**.
            - El **css** son solos estilos que se aplican al html de la misma nomenclatura, es decir, en el **app.component.html**. No tienen porque denominarse igual, ya que esto se define en el **app.component.ts**, pero normalmente se denominan igual. A continuación, podemos ver dónde se especifica:



```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = 'Jesús';
  apellido = 'Rodríguez';
}

```

        - Cualquier fichero **spect.ts** es un archivo de pruebas automáticas.
    - El fichero **app.module.ts** este archivo se puede manipular tanto manualmente como automáticamente. El decorador más importante de esta clase es el **@NgModule**.
- La carpeta **assets** como normal general es utilizada para alojar los recursos estáticos como por ejemplo imágenes.
    - Está formado por un fichero automático que se denomina **.gitkeep** este fichero no tiene nada y solo sirve para informa que cuando se suba el proyecto a un repositorio se **mantenga esa carpeta aunque esté vacía**, ya que git cuando subes una carpeta vacía la ignora.
- El fichero **karma.conf.js** es el fichero de configuración para las pruebas de karma.
- El fichero **main.js** es el primer código que angular va a lanzar para arrancar la aplicación. Todo lo que hay dentro de este fichero se hace de forma automática.
- El fichero **polyfills.ts** simplemente se utiliza para funciones que ayudan a la compatibilidad entre versiones viejas.
- El fichero **styles.css**, es un fichero de estilos que son globales para la aplicación.
- El fichero **tsconfig.app.json** es un fichero dónde se indican las especificaciones propias de la aplicación de TypeScript.
- El fichero **tsconfig.spec.json** contiene la configuración para realizar las pruebas.
- El fichero **tslint.json** incluye las reglas de configuración de nuestra aplicación para que nos muestre los errores cuando estemos codificando.

Bootstrap 4 en Angular

Para aplicar Bootstrap a nuestra aplicación de Angular, accedemos a la siguiente url https://getbootstrap.com/docs/4.4/getting-started/download/ y vamos a la siguiente sección:

BootstrapCDN

Skip the download with BootstrapCDN to deliver cached version of Bootstrap's compiled CSS and JS to your project.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1u
```

Copy

If you're using our compiled JavaScript, don't forget to include CDN versions of jQuery and Popper.js before it.

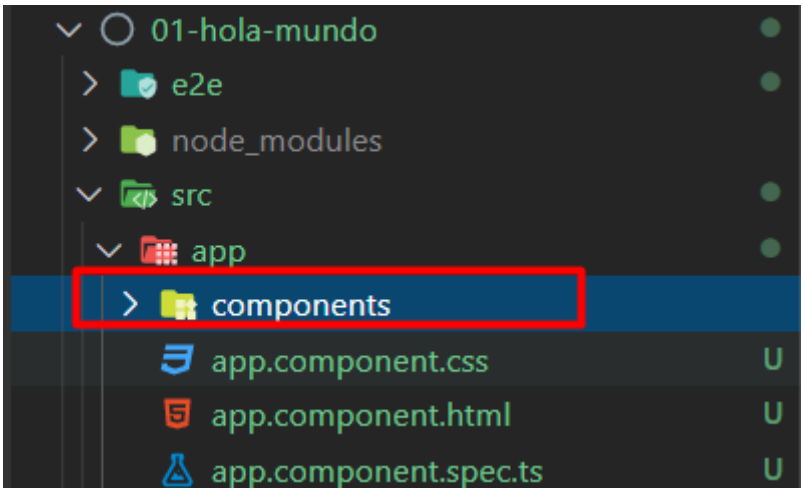
```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEj
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft
```

Copy

Copiamos el primer bloque de código y nos vamos a nuestro fichero **index.html** y lo colocamos dentro de la etiqueta **head**.

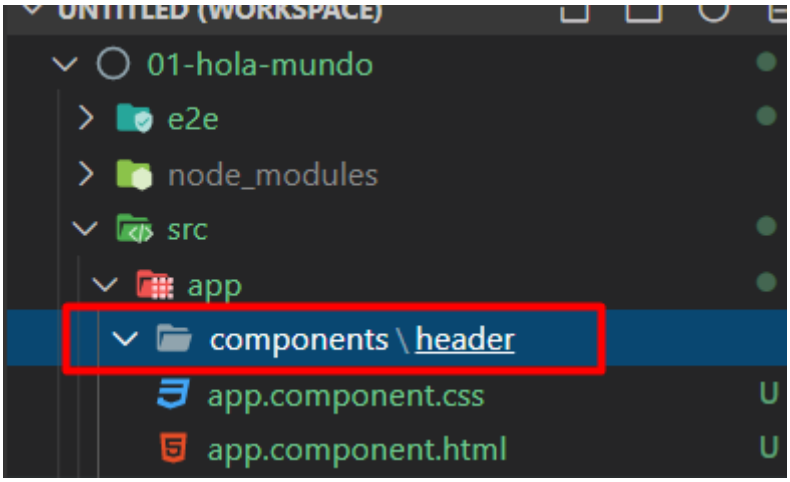
```
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>MyApp</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vko
</head>

<body>
    <app-root></app-root>
</body>

</html>
```

Ahora procedemos a utilizar Bootstrap en nuestra aplicación, lo que debemos de realizar es una nueva carpeta denominada **components** dentro de la carpeta **src/app**.

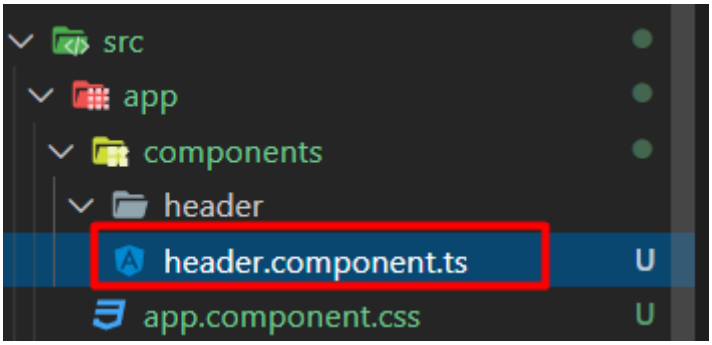


Esta carpeta que hemos creado es dónde normalmente se crean los componentes personalizados de la aplicación, pero todo esto depende de la estructura del proyecto.

Dentro de la carpeta que hemos creado (**components**) procedemos a crear una nueva carpeta denominada **header**.



En esta nueva carpeta que hemos creado (**components\header**) es dónde vamos a definir nuestro nuevo componente.



La extensión **.ts** indica que es un fichero de **typeScript,** el **.component** no es obligatorio pero es recomendable que lo tenga porque así podemos identificar de una manera más fácil que es un componente y además debido a la **nomenclatura** que le hemos dado sabemos que contiene este fichero ya que es el **header** de la página. (Por lo tanto, hemos creado el componente que se va a encargar de manejar el header de la página).

Ahora procedemos a crear el contenido de dicho fichero que hemos creado:

```
1  import { Component } from '@angular/core';
2
3  @Component ({
4      selector: 'app-header',
5      template: `<h1>Header Component</h1>`
6  })
7
8  export class HeaderComponent {
9
10 }
```

Ya tenemos configurado nuestro componente, aún nos falta añadir propiedades a la clase, pero para que el componente sea identificado como un componente debemos especificarlo en el siguiente fichero (**app.module.ts**):

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

import { AppComponent } from './app.component';
import { HeaderComponent } from './components/header/header.component';

@NgModule({
  declarations: [AppComponent, HeaderComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Ya tenemos configurado nuestro componente, ahora solo nos faltaría mostrarlo en nuestra página web, para ello debemos asignar el componente en el fichero **app.component.html**.

app.component.html | app.component.ts

01-hola-mundo > src > app > app.component.html

```
<app-header></app-header>

<ul>
    <li>Nombre: {{ nombre }}</li>
    <li>Apellidos: {{ apellido }}</li>
</ul>
```

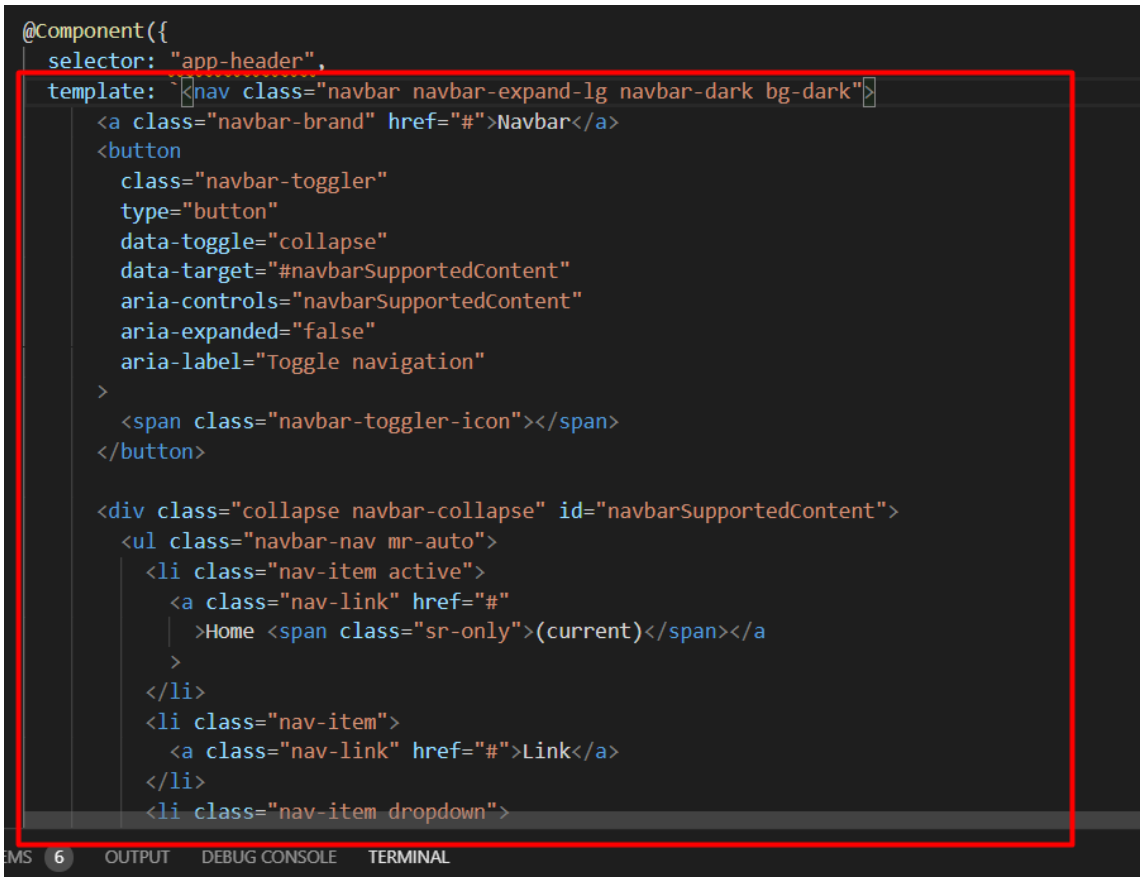
Si queremos que nuestro header aplique bootsrap podemos coger el componente nacbar y aplicarlo en nuestro header.component.ts en el atributo template del decorador @Component.

Navbar Home Link Dropdown Disabled Search Search

- Nombre: Jesús
- Apellidos: Rodríguez

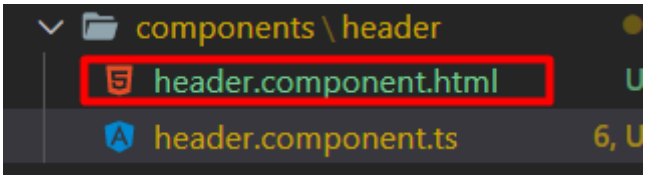
Separar HTML del componente

Actualmente, tenemos la lógica del **html** del componente del **header** de la siguiente manera:



Esta forma no es la buena práctica, por lo que vamos a proceder a separarlo y explicar cómo se debe realizar.

Lo primero que debemos de hacer es crear nuestro nuevo componente html para el header.



Una vez que hemos creado dicho componente en su interior debemos copiar el html que tenemos en nuestro parámetro template de la etiqueta **@Component y quitarlo de aquí**. Si ahora cargamos la página no nos mostrará dicho header.

Ahora lo que debemos de hacer es especificar en nuestro **header.component.ts** en la etiqueta **@Component** cual es la plantilla del **html** de dicho componente, para ello debemos sustituir el parámetro **template** por **templateUrl** y especificar dónde se encuentra el fichero **header.component.html**.

```
import { Component } from '@angular/core';

@Component({
  selector: "app-header",
  templateUrl: './header.component.html'
})
export class HeaderComponent {}
```

De esta manera, estamos importando dicho archivo indicando que será el html de este componente.

Crear componentes de forma automática

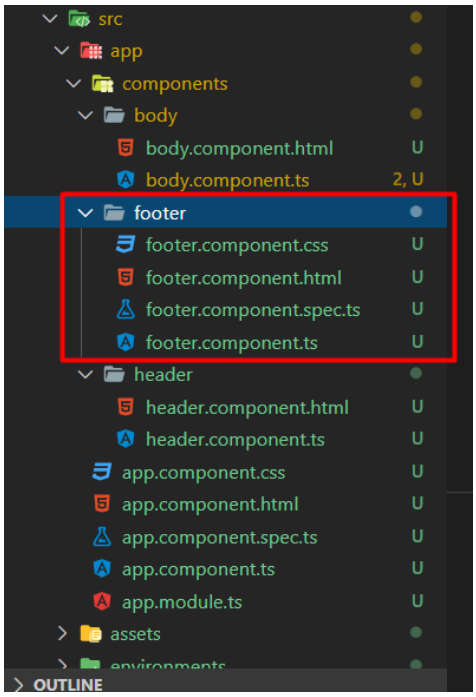
Para crear un componente de forma automática debemos ejecutar el siguiente comando, se realiza con angular CLI:

```
ng g c components/footer
```

g: significa generate

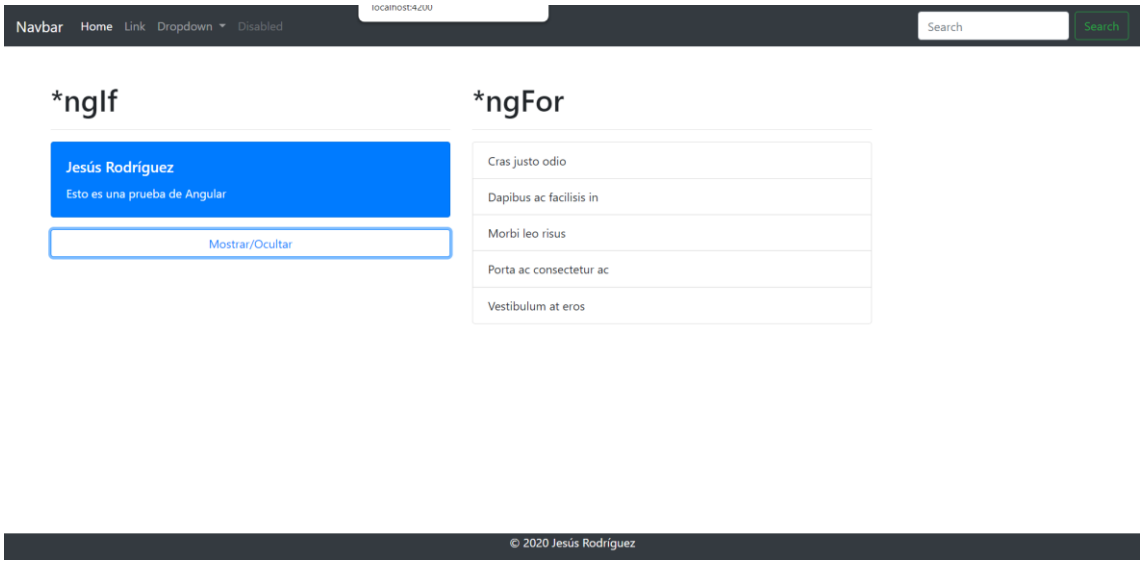
c: significa component

Una vez que hemos generado dicho componente de forma automática, podemos ver que se nos ha creado en la ruta que le hemos especificado de nuestro proyecto:



Esto nos ha creado ya el componente y se ha importando en app.module.ts

Directivas *ngIf y *ngFor

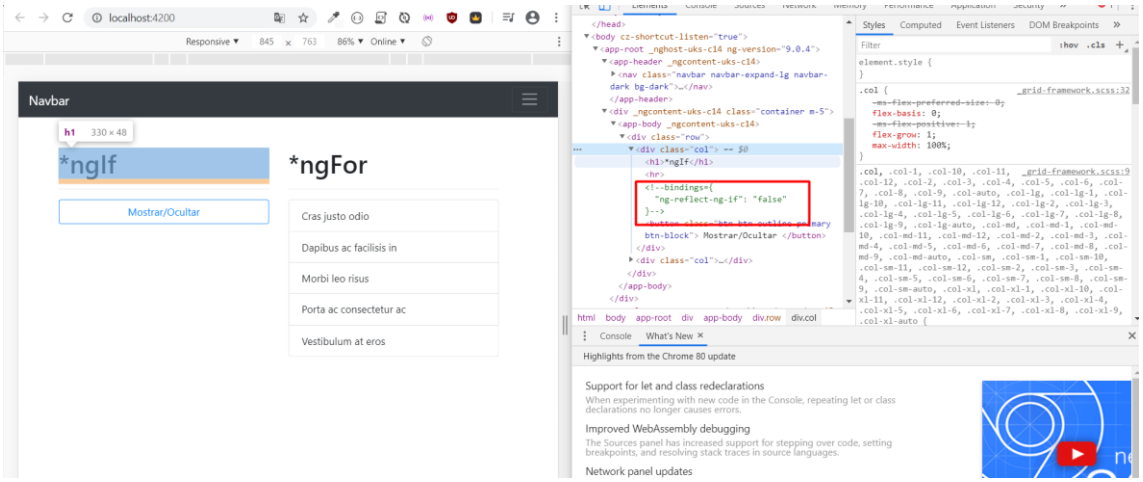


Actualmente nuestra aplicación es la siguiente:

1. Vamos a ver como aplicar la directiva *ngIf para que cuando se pulse el botón se oculte la card de color azul.

```
<div class="row">

    <div class="col">
        <h1>*ngIf</h1>
        <hr>
        <div *ngIf="false" class="card text-white bg-primary mb-3" style="width: 100%;">
            <div class="card-body">
                <h5 class="card-title">{{ frase.autor }}</h5>
                <p class="card-text">{{ frase.mensaje }}</p>
            </div>
        </div>
        <button class="btn btn-outline-primary btn-block">
            Mostrar/Ocultar
        </button>
    </div>
    <div class="col">
        <h1>*ngFor</h1>
        <hr>
        <ul class="list-group">
            <li class="list-group-item">Cras justo odio</li>
            <li class="list-group-item">Dapibus ac facilisis in</li>
            <li class="list-group-item">Morbi leo risus</li>
            <li class="list-group-item">Porta ac consectetur ac</li>
            <li class="list-group-item">Vestibulum at eros</li>
        </ul>
    </div>
</div>
```

Si le asignamos la variable false al elemento que queremos aplicar dicha directiva, este desaparece de la página web, ni siquiera se ve en el código fuente de inspección de elemento de la página.



Como podemos ver, se ha eliminado el elemento de la página y solo queda un comentario en la inspección de dicho elemento, pero este elemento es tratado por angular de forma automática para saber qué es lo que tiene que realizar cuando cambie el estado. **La diferencia de poner true o false es simplemente que destruye o crea el elemento**. Es decir, si ponemos **true** nos va a volver a aparecer el elemento cuando recarguemos la página.

Ahora vamos a aplicar la directiva ***ngIf**, para ello nos creamos una variable en el ts de nuestro componente:

```
export class BodyComponent {
    mostrar = true;

    frase: any = {
        mensaje: 'Esto es una prueba de Angular',
        autor: 'Jesús Rodríguez'
    };
}
```

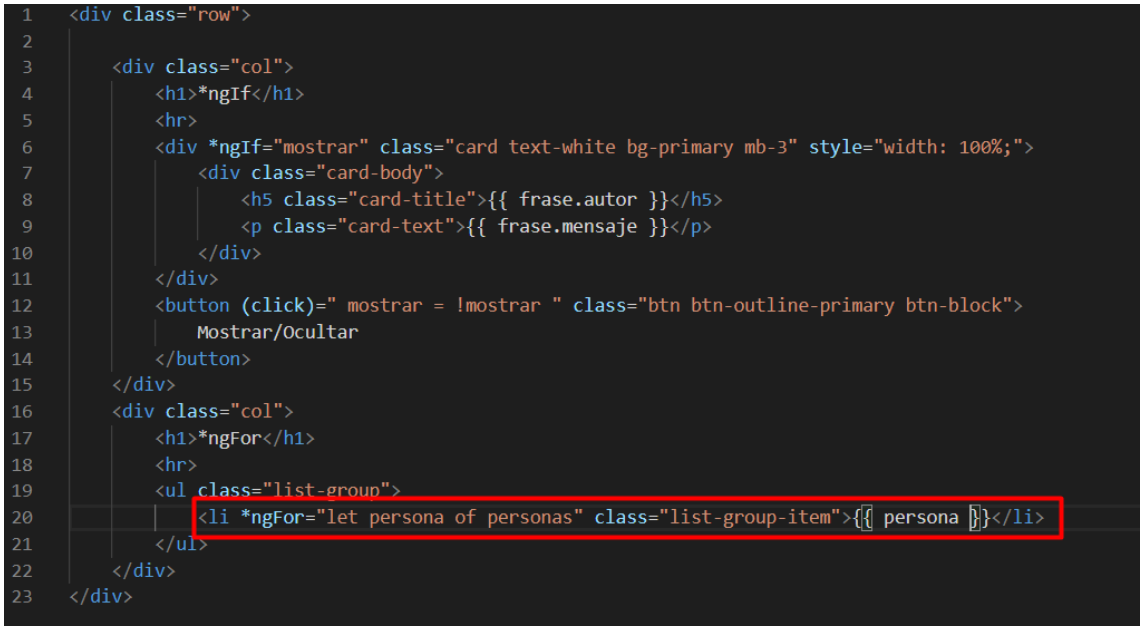
Una vez lo hemos creado el elemento que queremos ocultar o mostrar debe tomar el valor de dicha variable que hemos creado. Además, debemos añadir la funcionalidad de **click** a nuestro botón y que cada vez que se haga click en él cambie el valor de la variable que hemos creado. A continuación, se puede ver un ejemplo:

```
<div class="row">

    <div class="col">
        <h1>*ngIf</h1>
        <hr>
        <div *ngIf="mostrar" class="card text-white bg-primary mb-3" style
            <div class="card-body">
                <h5 class="card-title">{{ frase.autor }}</h5>
                <p class="card-text">{{ frase.mensaje }}</p>
            </div>
        </div>
        <button (click)=" mostrar = !mostrar " class="btn btn-outline-prim
            Mostrar/Ocultar
        </button>
    </div>
    <div class="col">
```

Ahora vamos a proceder a aplicar la directiva ***ngFor**, tenemos que destacar que esta directiva trabaja con arrays, por lo tanto, lo primero que debemos hacer es definir un array en nuestro **component.ts**.

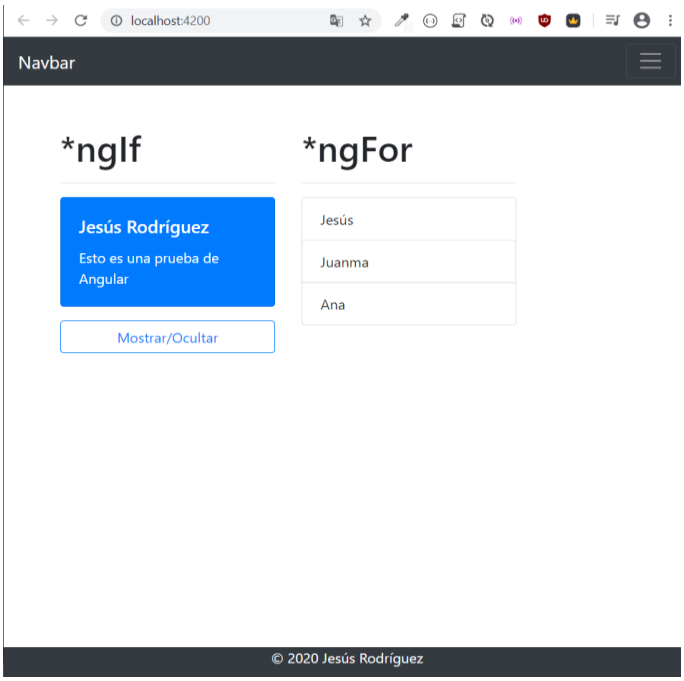
```
import { Component } from '@angular/core';

@Component ({
    selector: 'app-body',
    templateUrl: './body.component.html'
})

export class BodyComponent {
    mostrar = true;

    frase: any = {
        mensaje: 'Esto es una prueba de Angular',
        autor: 'Jesús Rodríguez'
    };

    personas: string[] = ['Jesús', 'Juanma', 'Ana'];
}
```

Una vez tenemos definido nuestro array que queremos mostrar a partir de la directiva ***ngFor**, procedemos a aplicarlo en esta misma, como vamos a ver a continuación en un ejemplo:



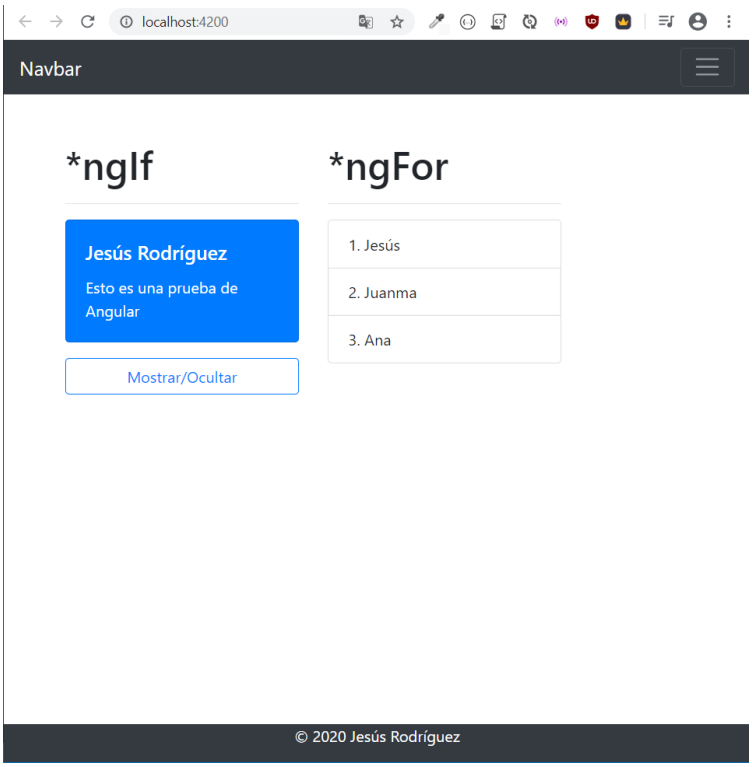
```
<div class="row">

    <div class="col">
        <h1>*ngIf</h1>
        <hr>
        <div *ngIf="mostrar" class="card text-white bg-primary mb-3" style="width: 100%;">
            <div class="card-body">
                <h5 class="card-title">{{ frase.autor }}</h5>
                <p class="card-text">{{ frase.mensaje }}</p>
            </div>
        </div>
        <button (click)=" mostrar = !mostrar " class="btn btn-outline-primary btn-block">
            Mostrar/Ocultar
        </button>
    </div>
    <div class="col">
        <h1>*ngFor</h1>
        <hr>
        <ul class="list-group">
            <li *ngFor="let persona of personas" class="list-group-item">{{ persona }}</li>
        </ul>
    </div>
</div>
```

Si mostramos nuestra página web ya deben de aparecer el array de personas:



Si queremos enumerar la lista, cuando declaramos la directiva ***ngFor**, después de declarar la variable persona referenciado al array de personas añadimos un **;** para declarar otra variable que será nuestro índice. No obstante, a continuación, os dejo un ejemplo:

```
<div class="row">

    <div class="col">
        <h1>*ngIf</h1>
        <hr>
        <div *ngIf="mostrar" class="card text-white bg-primary mb-3" style="width: 100%;">
            <div class="card-body">
                <h5 class="card-title">{{ frase.autor }}</h5>
                <p class="card-text">{{ frase.mensaje }}</p>
            </div>
        </div>
        <button (click)=" mostrar = !mostrar " class="btn btn-outline-primary btn-block">
            Mostrar/Ocultar
        </button>
    </div>
    <div class="col">
        <h1>*ngFor</h1>
        <hr>
        <ul class="list-group">
            <li *ngFor="let persona of personas; let enumerador = index" class="list-group-item">{{ enumerador+1 }}. {{ persona }}</li>
        </ul>
    </div>
</div>
```



Routes

A continuación, vamos a ver como se utilizan las rutas en Angular, para ello, tenemos que crear el fichero **app.routes.ts**, aquí es dónde vamos a definir la ruta de cada página de nuestras web. Un ejemplo de fichero:

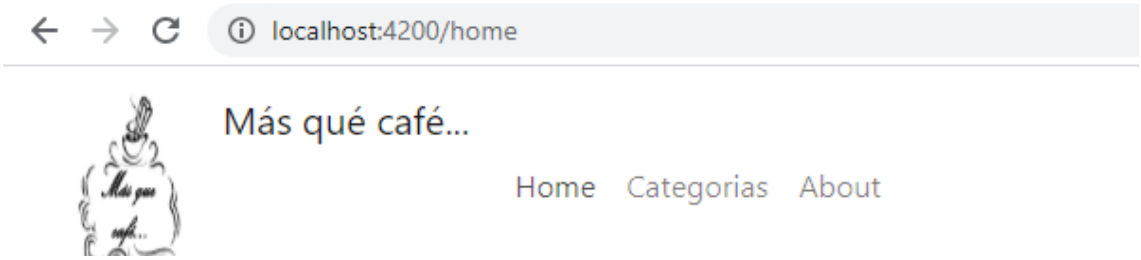
```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
import { AboutComponent } from './components/about/about.component';
import { CategoriasComponent } from './components/categorias/categorias.component';

const APP_ROUTES: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'about', component: AboutComponent },
    { path: 'categorias', component: CategoriasComponent },
    { path: '**', pathMatch: 'full', redirectTo: 'home' }
];

export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);
```

Como vemos, tenemos 3 páginas homre, about y categorías. El último path, lo que realiza es que si se intenta acceder a una página que no existe te redirecciona al home.

Para definir cada path en la página correspondiente, nos vamos a dónde queramos que se aplique dicha ruta, en nuestro caso al componente navbar, y se define de la siguiente manera.

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
        <!-- active es la clase que hace que se vea una capa gris-->
        <li class="nav-item">
            <a class="nav-link" [routerLink]="['home']">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" [routerLink]="['categorias']">Categorias</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" [routerLink]="['about']">About</a>
        </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
        <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
    </form>
```

Cuando accedemos a nuestra página web, podemos ver que la ruta aparece tal y como hemos definido.



Si queremos utilizar **routerLink** añadiéndole parámetros, debemos de realizar lo siguiente:

En nuestro **app.routes.ts** debemos de añadir un nuevo **path**, como veremos a continuación.

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
import { AboutComponent } from './components/about/about.component';
import { CategoriasComponent } from './components/categorias/categorias.component';
import { CategoriaComponent } from './components/categoria/categoria.component';


const APP_ROUTES: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'about', component: AboutComponent },
    { path: 'categorias', component: CategoriasComponent },
    { path: 'categoria/:id/:nombreCategoria', component: CategoriaComponent },
    { path: '**', pathMatch: 'full', redirectTo: 'home' }
];

export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);
```

Y en nuestro **html** dónde queramos realizar el evento de **routerLink** debemos colocar el código siguiente:

```
<h1>Categorias <small>Productos Más Qué Café...</small></h1>
<hr>

<div class="card-columns">
    <div class="card animated fadeIn fast" *ngFor="let categoria of categorias" ng-dblclick="dbClickCategoria()">
        <img [src]="categoria.img" class="card-img-top img-fluid" [alt]="categoria.nombreCategoria">
        <div class="card-block">
            <h4 class="card-title">{{ categoria.nombreCategoria }}</h4>
            <p class="card-text">{{ categoria.descripcionCategoria }}</p>
            <p class="card-text"><small class="text-muted">{{ categoria.lema }}</small></p>

            <a [routerLink]="['/categoria', categoria.id, categoria.nombreCategoria]" class="btn btn-outline-dark">Editar Categoría</a>
        </div>
    </div>
</div>
```

Como vemos se está pasando el id de la categoría y el nombre de la categoría como parámetros a dicha url.

Servicios

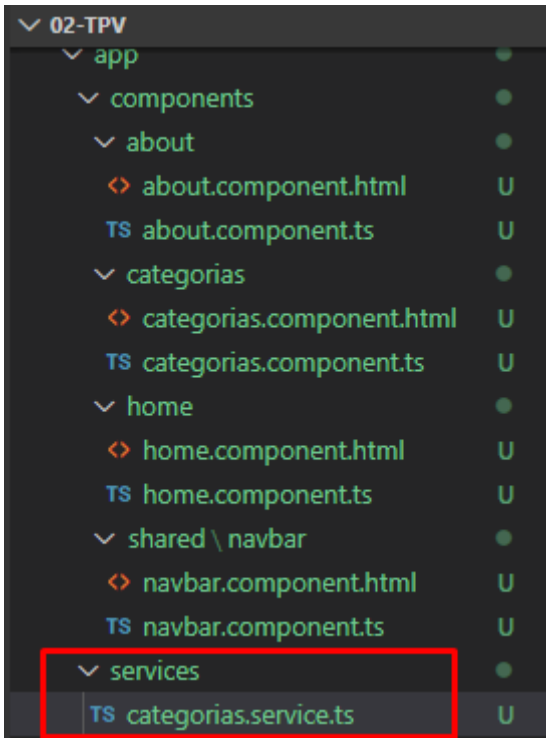
Los servicios en Angular se utilizan para evitar la duplicidad de código.

Por lo que si por ejemplo nosotros tenemos un componente categorías que es utilizado desde varios sitios lo correcto sería crearse un servicio denominado categorías. Este servicio será el encargado de mostrar la información o de manejar la información referente a las categorías de nuestra aplicación.

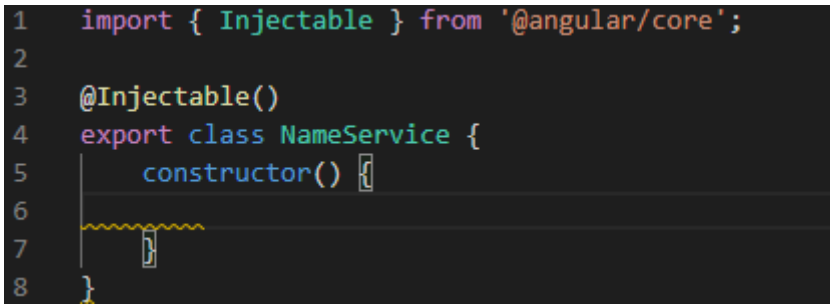
Características principales de los servicios:

- Dar información a quien lo necesite.
- Realizar peticiones CRUD (Create, read, update, delete). Angular no puede realizar inserciones directamente en bases de datos, por lo que necesita un intermediario.
- Mantener los datos de forma persistente.
- Dar como recurso re-utilizable para nuestra aplicación. Es decir, que varios componentes puedan acceder a la misma información o al mismo servicio.

La estructura que debe seguir la creación de un servicio en Angular es la siguiente:



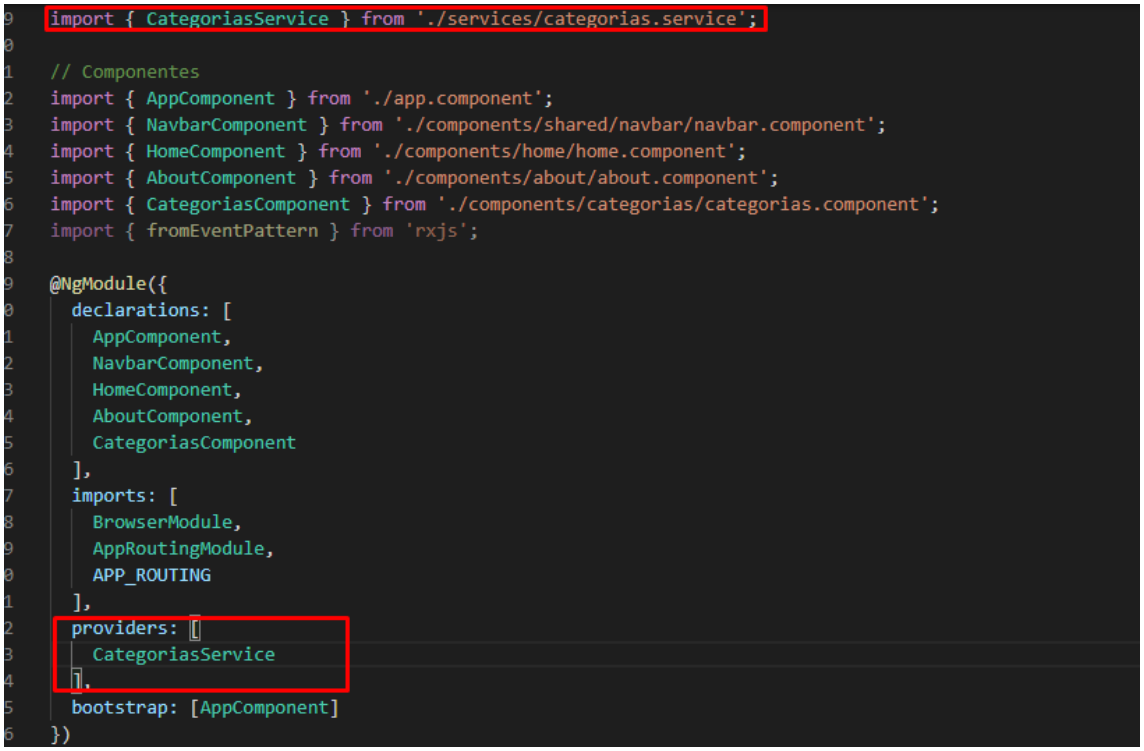
Si tenemos instalados los **snippets** correctamente con escribir **ng-servive**, se nos creará en el fichero **categorias.service.ts** los datos necesarios para poder empezar a codificar en él, se vería de la siguiente manera:



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

@Injectable()
export class NameService {
    constructor() {

    }
}
```

En **NameService**, la variable **Name** debería correspondes con el nombre de nuestro servicio, en mi caso sería **CategoriasService**.

Una vez que ya tenemos creado este servicio, tenemos que indicarle a Angular que dispone de dicho servicio para poder utilizarlo en el resto de la aplicación, esto se indica en el fichero **app.module.ts**, a continuación podemos ver un ejemplo:

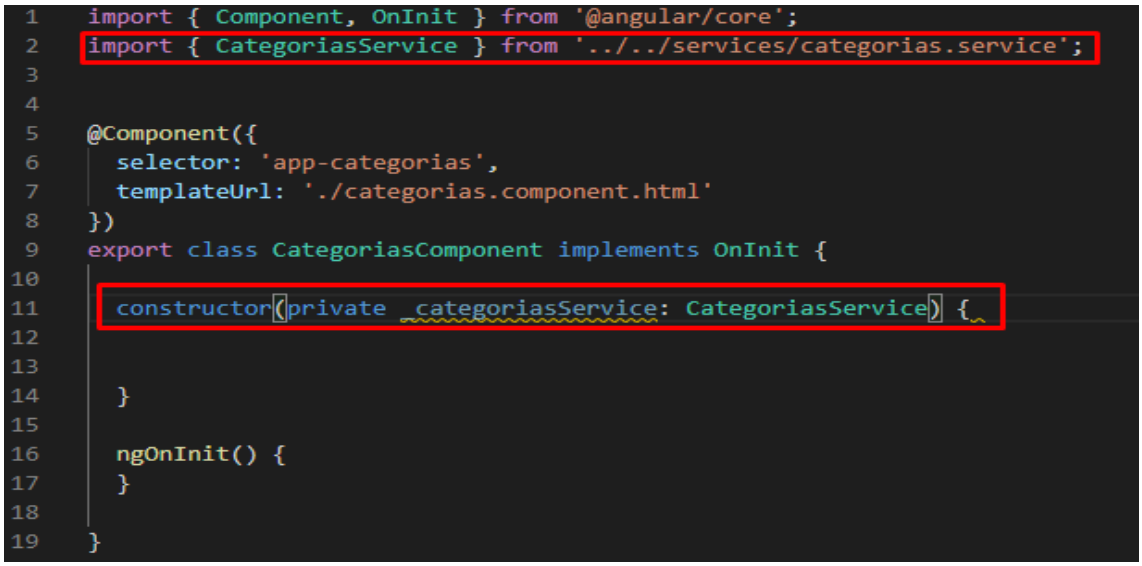


```
import { CategoriasService } from './services/categorias.service';

// Componentes
import { AppComponent } from './app.component';
import { NavbarComponent } from './components/shared/navbar/navbar.component';
import { HomeComponent } from './components/home/home.component';
import { AboutComponent } from './components/about/about.component';
import { CategoriasComponent } from './components/categorias/categorias.component';
import { fromEventPattern } from 'rxjs';

@NgModule({
  declarations: [
    AppComponent,
    NavbarComponent,
    HomeComponent,
    AboutComponent,
    CategoriasComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    APP_ROUTING
  ],
  providers: [
    CategoriasService
  ],
  bootstrap: [AppComponent]
})
```

Todos nuestros servicios se tienen que declarar en el bloque de providers.

Para poder utilizar nuestro servicio, nos tenemos que ir al **fichero .ts** del componente que deseamos utilizarlo, en mi caso voy a ir a **categorias.component.ts**, a continuación dejo un ejemplo de como quedaría el uso de dicho servicio que hemos creado con anterioridad.

1. Para importar nuestro servicio debemos de realizar lo siguiente:



```
import { Component, OnInit } from '@angular/core';
import { CategoriasService } from '../../services/categorias.service';


@Component({
  selector: 'app-categorias',
  templateUrl: './categorias.component.html'
})
export class CategoriasComponent implements OnInit {

  constructor(private _categoriasService: CategoriasService) {


  }

  ngOnInit() {
  }

}
```

Declarar dicho servicio en el constructor del componente, lo que realiza es que automáticamente lanza el código que tenemos en el constructor del servicio que hemos generado anteriormente.

probar código de angular podemos utilizar Plunker o StackBlitz