DWC

(Desarrollo Web en entorno cliente)

Índice

1.- El router 1
2.- Navegación mediante links 1
3.- Navegación mediante objeto router 3
4.- Paso de parámetros en ruta 4
5.- Paso de parámetros en queryString 6
6.- Protección de las rutas 9

1.- El router

Para poder navegar entre componentes Angular nos ofrece la herramienta router. Lo que vamos a realizar es asociar a cada ruta que definamos en nuestra aplicación el componente que se cargara.

La creación del router ha ido variando en las diferentes versiones de Angular, a partir de la 17 se crea en la carpeta **app** un fichero llamado **app.routes.ts**

Es en este fichero es donde se realiza la **asociación de rutas y componentes**. Para ello Angular utilizará una variable array de tipo Route denominada **routes**

```
import { Routes } from '@angular/router';

export const routes: Routes = [];
```

Inicialmente este array está vacío. Para gestionar el router deberemos ir rellenando el array con nuestras rutas y componentes.

Cada objeto de tipo ruta tiene varias propiedades, en principio vamos a utilizar 2, las propiedades **path** y **component**

El formato seria el siguiente:

```
export const routes: Routes = [
  { path: '', component: InicioComponent },
  { path: 'articulos', component: ArticulosComponent },
  { path: '**', component: ErrorComponent }
]
```

Para hacer uso de los componentes en la variable routes tendremos que importar los componentes.

```
import { Routes } from '@angular/router';
import { InicioComponent } from './Components/inicio/inicio.component';
import { ArticulosComponent } from './Components/articulos/articulos.component';
import { ErrorComponent } from './Components/error/error.component';

export const routes: Routes = [
    { path: '', component: InicioComponent },
    { path: 'articulos', component: ArticulosComponent },
    { path: '**', component: ErrorComponent }
  ]
```

La propiedad path hacer referencia a la ruta en nuestra aplicación, inicialmente partiendo de /. En este caso hemos definido:

- una ruta ‘ ‘ para el inicio de nuestra app (http://localhost:4200)
- una ruta **articulos** para la gestión de articulos (http://localhost:4200/articulos)
- una ruta **‘**’**para la gestión de cualquier otra ruta que no este definida en el array, es decir para gestionar los errores en la ruta

En cada ruta se cargará el componente indicado en la propiedad **component**

Es importante el orden en las rutas, ya que una vez encontrada una ruta ya no se gestionan las demás. Cuidado con la ruta **, debería ser la última.

Para poder utilizar el enrutamiento en nuestra aplicación deberemos importar en el app-config.ts el provideRouter

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

Ya hemos visto como se gestionan la ruta, es decir ya podemos asignar a cada ruta de nuestra app un componente que lo gestiones. **Nos falta saber dónde se cargarán los componentes indicados**.

Los componentes se cargarán en la etiqueta **<router-outlet></router-outlet>.** Esta etiqueta es la que pone angular en el componente **app-component.html**, esto lo que indica es que cada componente llamado por el router será cargado en esa etiqueta.

Lo habitual sería en el app-component.html incluir un componente header con la barra de navegación principal de nuestra app y luego dejar la etiqueta **<router-outlet></router-outlet>** para cargar los componentes, de esta manera tendremos siempre la barra de navegación disponible en nuestra app

```
<app-header></app-header>
<router-outlet></router-outlet>
```

Una vez visto la asociación de rutas y componente, y cómo funciona la carga del componente, **nos falta por ver como se llama a las rutas en nuestra app.**

Para esta función tenemos dos opciones:

- Navegar por enlaces, es decir desde la plantilla (el archivo html)
- Navegar desde código, es decir desde el controlador (el archivo ts)

2.- Navegación mediante links

Vamos a definir la barra de navegación con los enlaces adecuados para **Articulos** e **Inicio**, utilizando el componente de bootstrap Navbar

Para poder utilizar el router desde el html deberemos utilizar la etiqueta **routerLink** indicando la ruta a la que queremos navegar.

Para poder utilizar routerLink en el html deberemos importarlo en el ts

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

@Component({
  selector: 'app-header',
  standalone: true,
  imports: [RouterLink],
  templateUrl: './header.component.html',
  styleUrl: './header.component.css'
})
export class HeaderComponent {

}
```

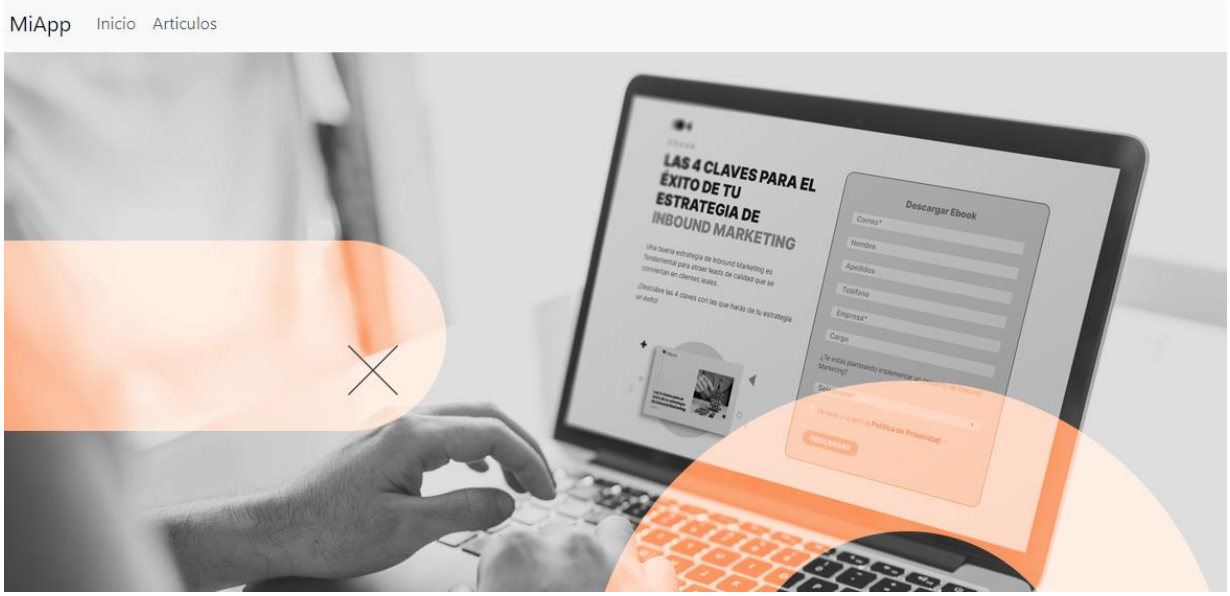
Una vez importado ya se podrá utilizar en el html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" routerLink="/"> MiApp</a>
        <button class="navbar-toggler" type="button">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
          <ul class="navbar-nav">
            <li class="nav-item">
               <a class="nav-link" routerLink="/">Inicio</a>
             </li>
              <li class="nav-item">
               <a class="nav-link" routerLink="/articulos">Articulos</a>
              </li>
          </ul>
        </div>
    </div>
</nav>
```

De esta manera estamos indicando que cuando pulsen sobre el enlace se llame a la ruta articulos, una vez llamada a esa ruta, angular ira mirar si existe esa ruta en el array de routes y cargará el componente indicado en la propiedad component (destacar que

hemos puesto /articulos y en el array de routes se define articulos, es decir hemos de poner la barra / en la llamada desde routerLink)

El resultado sería:



En un botón sería similar

```
<button class="btn btn-primary" routerLink="/articulos"> Articulos
</button>
```

También podemos utilizar otra forma para llamar desde la plantilla, esta forma es utilizar un **property binding** con la propiedad **routerLink.** En este caso se pone la propiedad routerlink entre [] y la ruta entre [] y entrecomillada

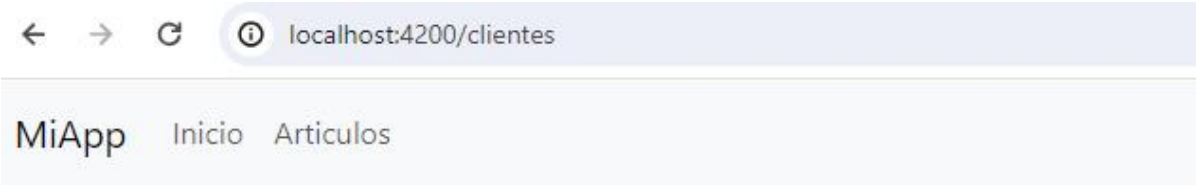
Para el enlace y el botón anterior

```
<li class="nav-item">
          <a class="nav-link" [routerLink]="[ '/articulos']">Articulos</a>
</li>
```

```
<button class="btn btn-success" [routerLink]="['/articulos']"> Articulos
</button>
```

Al pulsar sobre el enlace de artículos se cargará nuestro componente artículos.

Si alguien nos pusiera una ruta en el navegador el funcionamiento sería el mismo que si hubieran pulsado sobre un enlace. Si intentamos acceder a la ruta /clientes como no está definida se cargará el componente error, ya que es lo que le hemos dicho en el array routes con la ruta **



3.- Navegación mediante el objeto Router

En ocasiones necesitaremos navegar desde alguna instrucción de nuestro código, en estos casos no podemos utilizar la etiqueta routerLink. Cuando queramos navegar desde nuestro código deberemos utilizar **el objeto router** que nos proporciona angular.

Para ello deberemos importar el objeto router en nuestro componente.

```
import { Router } from '@angular/router';
```

Inyectarlo en el componente a través del constructor

```
constructor(private router:Router){}
```

Esto al igual que con los servicios lo que hará es definir una variable llamada router en el componente y que será accesible para todos los métodos de nuestro componente.

A partir de ese momento ya podemos utilizar el objeto router. **Para navegar utilizaremos el método navigate** indicándole entre **([ ])** la ruta entre comillas

Por ejemplo, supongamos que una vez cargada la página de artículos queramos volver a la página de inicio mediante código.

```
volver(){
    this.router.navigate(["/articulos"])
  }
```

4.- Paso de parámetros en ruta

Angular nos permite poder **pasar parámetros en la ruta** de una forma **muy similar a las peticiones REST**, es decir podría definir una ruta para poder ver el articulo con id m1 de la misma forma que haríamos en una petición mediante API REST.

Para realizar lo anterior deberíamos definir una ruta de este estilo **/articulo/m1** que en nuestra aplicación sería **http://localhost:4200/articulo/m1**

Con esta forma de trabajar deberemos saber **cómo definir los parámetros** y **como se van a recibir** posteriormente.

Para definir los parámetros en la ruta deberemos indicarlo en el array routes Para ello deberemos indicar el parámetro en la ruta mediante **:/parámetro** donde parámetro será el nombre en cual pasaremos el dato

```
{ path: 'articulo/:id', component: ArticuloComponent }
```

De esta manera angular sabe que en la ruta /articulo se va a pasar un dato en la variable id, la llamada seria así:

Con routerLink

```
<button class="btn btn-success" routerLink="/articulo/m2">Ver Articulo m2
<button>
```

Con [routerLink]

```
<button class="btn btn-success" [routerLink]="['/ articulo/m2']"> Articulo
m2 <button>
```

Con objeto router

```
this.router.navigate(["/articulo/m2"])
```

En todos los casos anteriores lo que estamos indicando es que queremos navegar a la ruta **articulo/m2**. Hemos definido en las rutas que la ruta **articulo/:id** la gestiona **el componente Articulo.**

Ahora angular cuando detecte una ruta del estilo articulo/id cargara el componente articulo.

Podemos añadir un botón para cada artículo en la tabla para ver sus datos.

```
@for(articulo of articulos; track articulo.id){
        <tr>
            <td>{{articulo.id}}</td>
            <td>{{articulo.nombre}}</td>
            <td>{{articulo.descripcion}}</td>
            <td>{{articulo.precio}}</td>
            <td>{{articulo.unidades}}</td>
            <td>
              <button class="btn btn-success"
                routerLink = "/articulo/{{articulo.id}}">Ver Articulo
              </button>
            </td>
        </tr>
    }
```

El resultado

Cuando pulsemos sobre el botón de ver se completará la url y se cargará el componente articulo.

Como vemos, siempre se carga el componente articulo, ahora lo que tenemos que hacer, es que el componente articulo sepa cuál es el parámetro que se le ha pasado en la ruta para poder mostrar el articulo adecuado en función del parámetro

Para recuperar en un componente el parámetro pasado en la ruta deberemos en el controlador del componente utilizar el objeto **activatedRoute** que es el que nos da acceso a la ruta activa en ese momento.

```
import { Router, ActivatedRoute } from '@angular/router';
```

Una vez importado lo inyectaremos en el constructor:

```
constructor(private miRutaActiva:ActivatedRoute){}
```

Una vez inyectado ya lo podemos utilizar en el componente. Lo más normal es **recuperar el dato pasado en el componente durante la carga del componente**, es decir en el **ngOnInit**, de forma similar a lo que hacíamos en la inyección de servicios para poder rellenar la lista de datos para poder pintarlos en el componente. Para ello definiremos en el componente una variable en la cual recogeremos el dato de la ruta en el ngOnInit.

Como vamos obtener los datos de un articulo en concreto vamos a utilizar el servicio que creamos para gestionar los artículos, para ello deberemos también importarlo e inyectarlo en el componente.

Para acceder al **parámetro pasado en la url** deberemos utilizar **una suscripción al objetos params**.

La suscripción nos devolverá un objeto del cual deberemos obtener el parámetro pasado, la forma seria:

```
activatedRoute.params.subscribe(params=>variable=params[“nombreParametro”])
```

En nuestro caso:

```
export class ArticuloComponent {
  articulo!:Articulo
  constructor(private miRutaActiva:ActivatedRoute,
              private miServicio:ArticulosService
  ){}

  ngOnInit(){
    this.miRutaActiva
    .params
    .subscribe(params=>this.articulo=this.miServicio.getArticulo(params["id"]))
  }

}
```

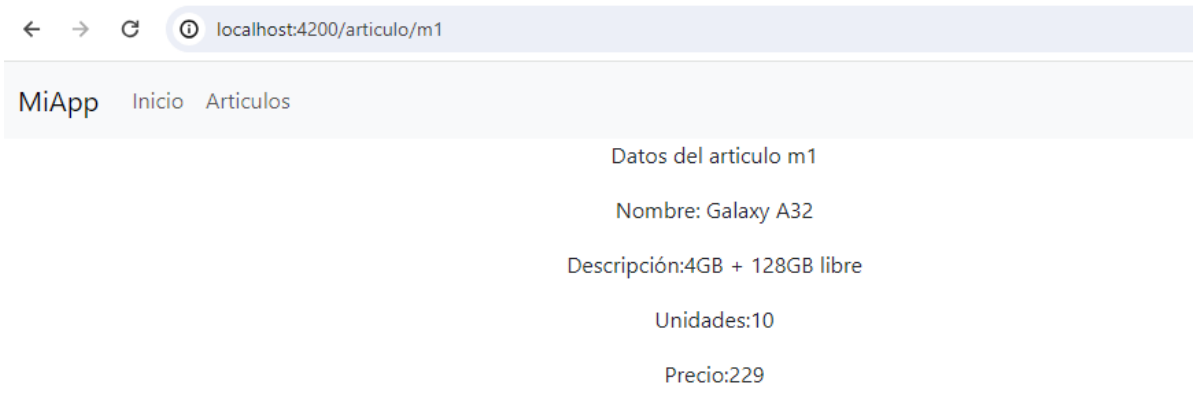
En nuestro caso hemos utilizado el nombre del parámetro definido que era **id**

Lo que acabamos de hacer es llamar a la ruta /articulo/m1, al recibir esa ruta angular carga el componente articulo y utilizando activatedRoute hemos recuperado el dato m1, **lo que deberíamos hacer ahora con ese dato seria acceder realmente al articulo con id m1**. Para ello deberíamos **utilizar un servicio** que nos proporcionara esa información, con lo cual deberíamos trabajar con un **servicioArticulos** e inyectarlo en el constructor y en el ngOnInit recuperar los datos del articulo con id que habremos recuperado con el activatedRoute.

Al recuperar los datos del articulo los guardamos en la variable articulo que hemos definido en el componente que es la que utilizaremos para interpolar en la vista

```
<div class="text-center">
    <p>Datos del articulo {{articulo.id}}</p>
    <p>Nombre: {{articulo.nombre}}</p>
    <p>Descripción:{{articulo.descripcion}}</p>
    <p>Unidades:{{articulo.unidades}}</p>
    <p>Precio:{{articulo.precio}}</p>
</div>
```

Resultado



5.- Paso de parámetros en queryString

Angular nos permite pasar más de un parámetro en la url, para ello deberemos de utilizar **queryParams.** De esta manera podemos pasar parámetros en la url al igual que podemos hacer desde JavaScript mediante los conocidos query string.

El formato es el siguiente:

url?parámetro1=valor1?parametro2=valor2

Suelen ser parámetros para realizar consultas sobre el objeto de la url. Por ejemplo podríamos consultar artículos de un precio mayor.a un valor determinado e indicar que se muestren en orden ascendente

http://localhost:4200/verArticulos?precio=300&orden=ascendente

En este ejemplo estamos pasando por la url dos parámetros precio con valor 300 y orden con valor ascendente

Podemos pasar los parámetros desde el routerLink (en el HTML) y desde el objeto router (en el TS).

Para hacer este ejemplo deberemos:

- Crear el componente verArticulos
- Modifcar la variable routes para añadir la nueva ruta y su componente.
- En el header añadir los enlaces a la ruta verArticulos con paso de parámetros con queryStriing
- Añadir un el servicio un metodo para filtrar articulos

Uso de queryparams desde el routerLink (HTML)

Para realizar esta opción deberemos utilizar la sintaxis de property binding [routerLink] y utilizar otro property binding para queryparams.

Html del componente header

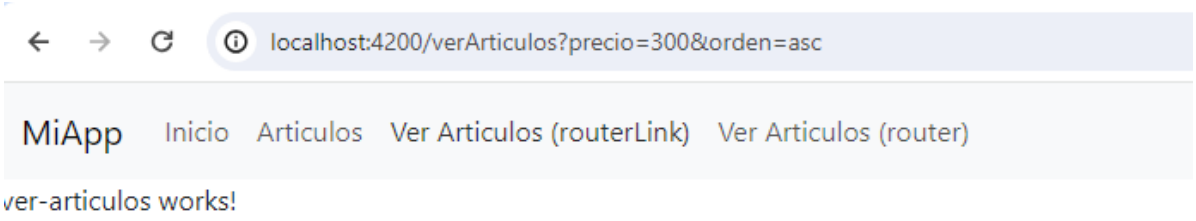
```
<li class="nav-item">
  <a class="nav-link"
     routerLink="/verArticulos"
     [queryParams]="{precio:300,orden:'asc'}">Ver Articulos (routerLink)
  </a>
</li>
```

La ruta a la que llamamos es la siguiente:

http://localhost:4200/verArticulos?precio=300&orden=’asc’

Si quisiéramos pasar más parámetros, deberemos completar el objeto asociado a queryparams con los pares, nombre:valor.

El resultado es:



Uso de queryparams desde el router (TS)

También podemos utilizar el paso de parámetros con queryparams desde **el objeto Router** utilizado en los componentes a través de los métodos que definamos para poder

navegar desde forma programática. Para ello utilizaremos el método navigate con la ruta deseada y el objeto queryparams. Quedaría de la siguiente forma:

router.navigate(“[ruta]”,{queryparams:{parametro1:’valor1’, parametro2:’valor2’}})

En nuestro caso:

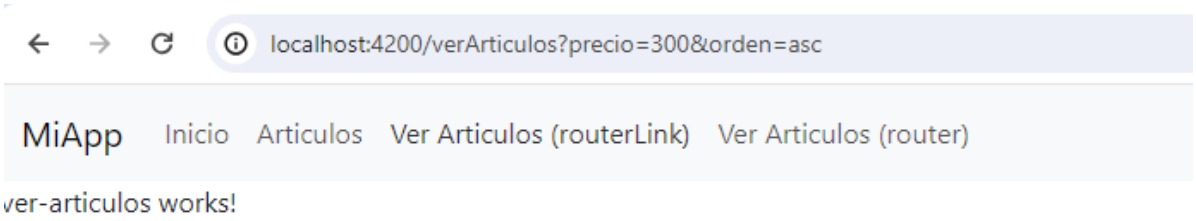
Html del componente header

```
<li class="nav-item">
  <a class="nav-link" (click)="verArticulos()">Ver Articulos (router)</a>
</li>
```

En el ts del componente header

```
constructor(private miRouter:Router){}

verArticulos(){
 this.miRouter.navigate(["/verArticulos"],
                        {queryParams:{precio: 300,orden:'asc'}})
}
```

El resultado es el mismo que antes:



Recepción de queryparams

Hemos visto cómo podemos pasar los parámetros tanto desde el HTML como desde el TS. Ahora nos falta ver cómo podremos recoger estos parámetros cuando se carga el componente indicado en la ruta, para ello deberemos utilizar el objeto **ActivatedRoute**

En este caso utilizaremos un mecanismo de **subscripción** al método queryParams que nos devolverá un **observable** con los parametros

Esto es un mecanismo similar a las promesas, el método queryParams nos devolverá los parámetros pasados en el queryParams, pero no se realizara de forma inmediata, pueden tardar, por modificaciones o tiempos extra. Para esperar la recepción de los parámetros nos subscribiremos y cuando estén disponibles los datos los utilizaremos.

```
activatedRoute.queryParams.subscribe(params=>{variable1=params["parametro1"]
                                              variable2=params[“parametro2”]})
```

Lo que estamos realizando es subscribirnos al método queryParams que nos devolverá el objeto params y de ahí sacaremos los parámetros, para ello deberíamos volcarlos en variables del componente para poder utilizarlas más adelante.

En nuestro caso en el ejemplo de paso de parámetros con queryparams de precio y orden vamos a definir el método **filtrarArticulos(precio,orden)** en el servicio

```
filtroArticulos(precio:number,orden:string){
    let articulosFiltrados:Articulo[]
    articulosFiltrados=this.articulos.filter(a=>a.precio>precio)
    if(orden=='asc')
      return articulosFiltrados.sort((a,b)=>a.precio-b.precio)
    else
      return articulosFiltrados.sort((a,b)=>a.precio-b.precio)
  }
```

Ahora ya podemos completar el ts del componente verArticulos

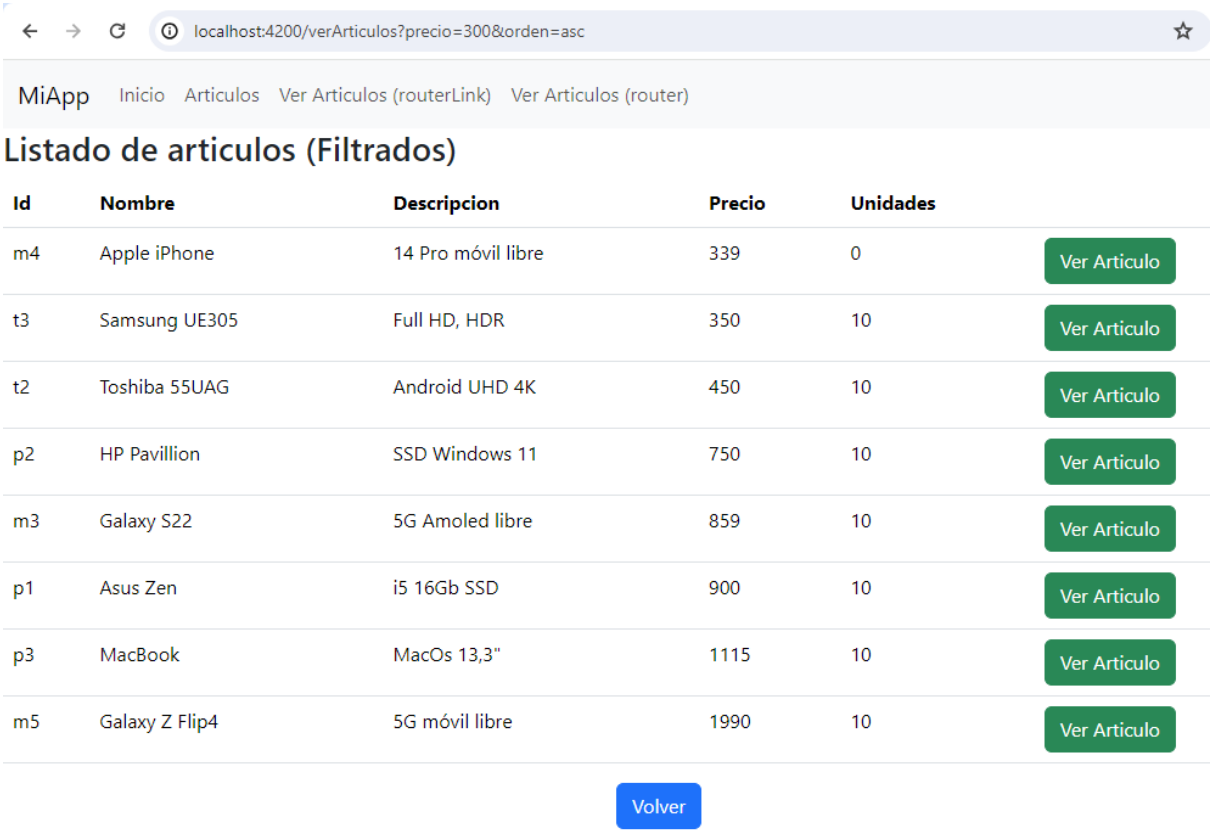
```
import { Component } from '@angular/core';
import { ActivatedRoute, RouterLink } from '@angular/router';
import { Articulo } from '../../Modelos/articulo';
import { ArticulosService } from '../../Services/articulos.service';

@Component({
  selector: 'app-ver-articulos',
  standalone: true,
  imports: [RouterLink],
  templateUrl: './ver-articulos.component.html',
  styleUrl: './ver-articulos.component.css'
})
export class VerArticulosComponent {

articulos!:Articulo[]

constructor(private miRutaActiva:ActivatedRoute,
            private miServicio:ArticulosService){}

ngOnInit(){
 let precio:number
 let orden:string

 this.miRutaActiva
     .queryParams
      subscribe(params=>{precio=params["precio"]
                         orden=params["orden"]
                         this.articulos=this.miServicio
                                        .filtroArticulos(precio,orden)})
  }
}
```

Para poder realizar todo el proceso de forma correcta hemos realizado los siguientes pasos:

- Definir en el servicio el metodo filtrarArticulos(precio,orden)r
- Importar ActivatedRoute y el Servicio de Articulos
- Inyectar al constructor ActivatedRoute y el Servicio
- Crear una variable articulos para interpolarla en la vista
- En ngOnInit
  - Definir las variables precio y orden
  - Suscribirnos a queryParams de activatedRoute
  - Obtener los parametros preco y orden
  - Rellenar la variable articulos con los valores devueltos por el metodo filtarArticulos del servicio

El html del componente verArticulos es mostrar una tabla con los datos de la variable artículos igual que hemos hecho en el componete artículos.

El resultado final



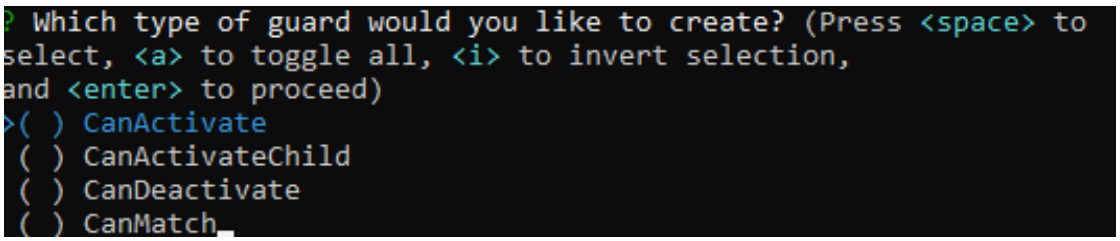
6.- Protección de las rutas

Hasta ahora hemos visto cómo podemos navegar entre componentes y pasar parámetros. Vamos a ver cómo podemos **proteger una ruta**, es decir cómo podemos hacer para cuando se intente acceder a una ruta, decidamos si dejamos que se cargue el componente o por el contrario denegamos la carga. Para realizar esta tarea Angular nos ofrece los **guards**.

Para crear un guard utilizaremos el CLI con el siguiente comando

ng g g nombreGuarda

Angular dispone de varios tipos de guarda, cada una se aplica en un caso diferente, al crear la guarda nos preguntara de que tipo queremos que sea.



Los guards en Angular, son de alguna manera: middlewares que se ejecutan antes de cargar una ruta y determinan si se puede cargar dicha ruta o no. Existen 4 tipos diferentes de Guards (o combinaciones de estos) que son los siguientes:

1. (*CanActivate*) Antes de cargar los componentes de la ruta.
2. (*CanActivateChild*) Antes de cargar las rutas hijas de la ruta actual.
3. (*CanDeactivate*) Antes de intentar salir de la ruta actual (usualmente utilizado para evitar salir de una ruta, si no se han guardado los datos).
4. (CanMatch) se ejecuta antes de que el enrutador intente activar cualquier ruta. Esto te permite evaluar si la ruta debe ser considerada en primer lugar

La más utilizada es la guarda de tipo CanActivate

Como middleware, estos componentes se ejecutan de manera intermedia antes de determinadas acciones y **si retorna true la ruta seguiría su carga normal**, **en caso negativo, el guard retornaría false y la ruta no se cargaría**. Generalmente en caso de que no se cumpla la condición del guard, **se suele hacer una redirección a la ruta anterior o a una ruta definida como la interfaz de autenticación**.

Si vamos a la carpeta de Guards veremos que se ha creado el fichero **guarda1.guard.ts**

El contenido del fichero de la guarda es el siguiente:

```
import { CanActivateFn } from '@angular/router';

export const guarda1Guard: CanActivateFn = (route, state) => {
  return true;
};
```

Este es el fichero inicial de la guarda. Como podemos ver **define una funcion** que **recibe dos parámetros route y state**, en el código de la función aparece la instrucción **return true**

Para determinar si nuestra guarda1 va a permitir activar una ruta o no lo va a permitir dependerá del return que se realice, por defecto no se implementa nada y siempre se permite la activación de la ruta. Si devolviéramos false no se permitiría el acceso a la ruta indicada.

Para poder utilizar el guard en nuestras rutas deberemos indicarlo en el módulo del router asignándolo como un valor más de cada objeto de tipo route.

```
{ path: 'articulos',
      component: ArticulosComponent,
      canActivate:[guarda1Guard]
    },
```

De esta manera, cada vez que se llame a la ruta /artículos se comprobara si esa ruta tiene asociado un guard, en ese caso se ejecutara el guard y si devuelve true se cargara el componente, si devuelve false no se cargara.

En nuestro ejemplo se va a cargar el componente articulos porque el guard asociado guarda1 devuelve true.

Si modificáramos el valor de return a false en guarda1.ts, ya no se cargaría el componente trabajadores

En el guard deberemos definir la lógica necesaria para poder determinar si permitimos la activación de la ruta o no. Generalmente haremos alguna comprobación previa para determinar el resultado del guard.

Vamos a modificar guarda1 para que permita el acceso si la hora del día es mayor o igual a las 10h, en este caso en función de la hora, si son las 10h o posterior se devolverá true y se cargará el componente asociado. Si aún no son las 10h se navegará a la ruta de inicio y devolverá false para que no se cargue el componente. Para poder navegar deberemos haber importado el Router y haberlo inyectado en el constructor del guard

Es muy habitual utilizar los guards inyectando el objeto Router y Servicios en el constructor para poder realizar la lógica del guard en función de datos del servicio y usar el router para en caso de no activar la ruta navegar a una ruta de error.