Práctica Angular – Parte 3

Películas

Diseño del componente *pelicula*

No confundir este nuevo componente que vamos a crear ahora con el componente de la práctica anterior. El que vamos a crear ahora, `PeliculaComponent`, será responsable de gestionar una película, mientras que el anterior, `PeliculasComponent`, se encargaba de mostrarlas todas. Quizá se entenderá esta diferencia en breve, durante el transcurso de esta parte de la práctica.

De momento vamos a proceder con la creación del componente:

```
ng g c components/peliculas/pelicula –flat -is –spec=false
```

`--flat` Indica que no queremos crear una carpeta específica para el componente. Por tanto, los dos ficheros que se van a generar (`pelicula.component.ts` y `pelicula.component.html`) se encontrarán dentro de la carpeta `components/peliculas`, junto a los otros dos que ya existen (`peliculas.component.ts` y `peliculas.component.html`).

Creación de la nueva ruta

Cuando el usuario pulse el botón “Ver más...” de una “Card” tenemos que mostrar la página que muestra la información detallada de esa película. Para esto necesitamos añadir una nueva ruta a nuestro fichero de rutas, teniendo en cuenta que necesitamos pasar el ID de la película que tenemos que mostrar:

routes,module.ts

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

```
import { AboutComponent } from './components/about/about.component';
import { PeliculasComponent } from
'./components/peliculas/peliculas.component';
import { PeliculaComponent } from
'./components/peliculas/pelicula.component';

const APP_ROUTES: Routes = [
  { path: 'home', component: HomeComponent},
  { path: 'about', component: AboutComponent},
  { path: 'peliculas', component: PeliculasComponent},
  { path: 'peliculas/:id', component: PeliculaComponent},
  { path: '**', pathMatch: 'full', redirectTo: 'home' }
];
export const APP_ROUTING = RouterModule.forRoot(APP_ROUTES);
```

Ahora tenemos que modificar la plantilla de `PeliculasComponent` para que al pulsar el botón "Ver más..." se muestre la página del nuevo componente que acabamos de crear:

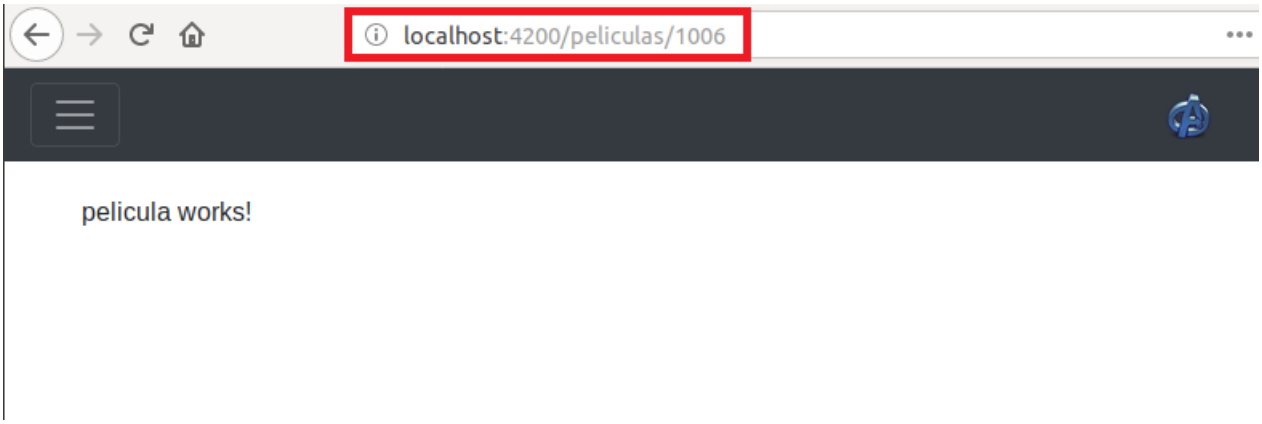
peliculas.component.html

```
<h1>Películas disponibles</h1>
<hr>
<div class="card-columns">
  <div class="card animated fadeIn fast" *ngFor="let pelicula of peliculas">
    <img class="card-img-top" [src]="pelicula.img" [alt]="pelicula.titulo">
    <div class="card-body">
      <h4 class="card-title">{{ pelicula.titulo }}</h4>
      <p class="card-text">{{ pelicula.sinopsis }}</p>
      <p class="card-text">{{ pelicula.direccion }}</p>
      <p class="card-text">
        <small class="text-muted">{{ pelicula.anio }}</small>
      </p>
      <a [routerLink]="['/peliculas', pelicula.id]"
            class="btn btn-outline-primary btn-block">
        Ver más...
      </a>
    </div>
  </div>
</div>
```

En el código anterior hay que advertir dos cosas:

- Para poder usar `routerLink` hemos cambiando el botón por una etiqueta `<a>`
- Que estamos pasando el ID de la película en el `routerLink`

Suponiendo que pulsamos sobre el enlace de la película “El resplandor” veremos que en la URL se muestra el ID 1006:



Alternativamente, si quisiéramos seguir usando el botón tendríamos que hacer lo siguiente:

En la clase de `PeliculasComponent` deberíamos crear un método que recibiese el ID del botón y programáticamente realice el enrutado, para lo cual se necesita invocar al método `navigate()` del objeto `Router` de Angular (lo cual implica inyectar `Router` en el constructor):

peliculas.component.ts

```
import { Component, OnInit } from '@angular/core';
import { PeliculasService } from '../../services/peliculas.service';
import { Pelicula } from '../../models/peliculas/pelicula';
import { Router } from '@angular/router';

@Component({
  selector: 'app-peliculas',
  templateUrl: './peliculas.component.html',
  styles: []
})
export class PeliculasComponent implements OnInit {
  peliculas: Pelicula[] = [];
  constructor(private peliculasService: PeliculasService,
```

```
            private router: Router) { }

  ngOnInit() {
    this.peliculas = this.peliculasService.getPeliculas();
  }

  verPelicula(id: number) {
    this.router.navigate(['/peliculas', id]);
  }
}
```

Y en la plantilla de `PeliculasComponent` capturamos la pulsación del botón, invocando al método `verPelicula()` para el ID de película en curso:

peliculas.component.html

```
<h1>Películas disponibles</h1>
<hr>
<div class="card-columns">

  <div class="card animated fadeIn fast" *ngFor="let pelicula of peliculas">
    <img class="card-img-top" [src]="pelicula.img" [alt]="pelicula.titulo">
    <div class="card-body">
      <h4 class="card-title">{{ pelicula.titulo }}</h4>
      <p class="card-text">{{ pelicula.sinopsis }}</p>
      <p class="card-text">{{ pelicula.direccion }}</p>
      <p class="card-text">
        <small class="text-muted">{{ pelicula.anio }}</small>
      </p>
        <!--<a [routerLink]="['/peliculas', pelicula.id]" class="btn btn-
outline-primary btn-block">
        Ver más...
      </a>-->
        <button (click)="verPelicula(pelicula.id)" type="button" class="btn
btn-outline-primary btn-block">
        Ver más...
      </button>
    </div>
  </div>
```

Recepción del parámetro

Veamos cómo recuperamos el identificador de la película desde la clase de `PeliculaComponent`:

Recordemos que en el fichero de rutas dijimos que el parámetro se llamaba `id`:

routes.module.ts

```
import { RouterModule, Routes } from '@angular/router';
...
const APP_ROUTES: Routes = [
    ...
  { path: 'peliculas/:id', component: PeliculaComponent},
  { path: '**', pathMatch: 'full', redirectTo: 'home' }
];
...
```

Por tanto, para recibir el valor de este identificador hemos de hacer lo siguiente en la clase de `PeliculaComponent`: inyectar el objeto `ActivatedRoute` de Angular en el constructor y suscribirnos a un parámetro llamado `id`:

pelicula.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-pelicula',
  templateUrl: './pelicula.component.html',
  styles: []
})
export class PeliculaComponent implements OnInit {
  constructor(private activatedRoute: ActivatedRoute) {
    this.activatedRoute.params.subscribe(params => {
      console.log(params['id']);
    });
```

```
  }
  ngOnInit() {
  }
}
```

Ahora necesitamos modificar nuestro servicio para añadir un método que a partir de un identificador devuelva un objeto película (y no todo el array):

peliculas.services.ts

```
  ...
  getPelicula(id: number): Pelicula {
      return this.peliculas.filter(peli => peli.id == id)[0];
  }
```

A continuación modificamos PeliculaComponent para invocar al nuevo método del servicio desde el constructor:

pelicula.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import {PeliculasService} from '../../services/peliculas.service';
import {Pelicula} from '../../models/peliculas/pelicula';

@Component({
  selector: 'app-pelicula',
  templateUrl: './pelicula.component.html',
  styles: []
})
export class PeliculaComponent implements OnInit {
  pelicula: Pelicula = {
    id: 0,
    titulo: '',
    sinopsis: '',
    img: '',
    anio: 0,
    direccion: ''
  };
```

```
  constructor(private activatedRoute: ActivatedRoute,
    private peliculasService: PeliculasService)
   {
    this.activatedRoute.params.subscribe(params => {
      console.log(params['id']);
     this.pelicula = this.peliculasService.getPelicula(params['id']);
    });
  }
  ngOnInit() {
  }
}
```

Modificamos ahora la plantilla:

pelicula.component.html

```
<h1 class="animated fadeIn">{{ pelicula.titulo | uppercase }}
<small>({{ pelicula.anio }})</small></h1>
<hr>
<div class="row animated fadeIn fast">
  <div class="col-md-4">
    <img [src]="pelicula.img" class="img-fluid" [alt]="pelicula.titulo">
    <br><br>
    <a [routerLink]="['/peliculas']" class="btn btn-outline-danger btn-
block">Regresar</a>
  </div>
  <div class="col-md-8">
    <h3>{{ pelicula.titulo }}</h3>
    <hr>
    <p>{{ pelicula.sinopsis }}</p>
    <hr>
    <p>{{ pelicula.direccion}}</p>
  </div>
</div>
```

Obtendremos una salida similar a la de la siguiente figura:

Implementar la funcionalidad de buscar

En el *navbar*, además del menú de opciones (Home, Películas, About), tenemos un caja de texto junto con un botón cuya utilidad es que el usuario pueda realizar una búsqueda (un filtro) de películas según el texto introducido. Ahora es el momento de implementar esa funcionalidad. Para ello, hacemos lo siguiente:

En la plantilla del *navbar*:

- Cambiar los textos literales a español.
- Al pulsar RETURN (ENTER) sobre la caja de texto que se ejecute un método llamado `buscarPelicula()`.
- Crear un alias para la caja de texto. El alias será: `#buscarTexto`. Este alias nos permite pasar el valor introducido en la caja de texto como argumento del método `buscarPelicula()`.
- El botón lo pasamos de tipo `submit` a tipo `button`, ya que no queremos que el formulario se envíe.
- Al pulsar el botón que se ejecute también el método `buscarPelicula()`.

navbar.component.html

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  ...
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    ...
    <form onsubmit="return false;" class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="search"
           placeholder="Buscar película"
           #buscarTexto (keyup.enter)="buscarPelicula(buscarTexto.value)"
           aria-label="Search">
      <button (click)="buscarPelicula(buscarTexto.value)"
              class="btn btn-outline-primary my-2 my-sm-0"
                type="button">Buscar</button>
    </form>
  </div>
</nav>
```

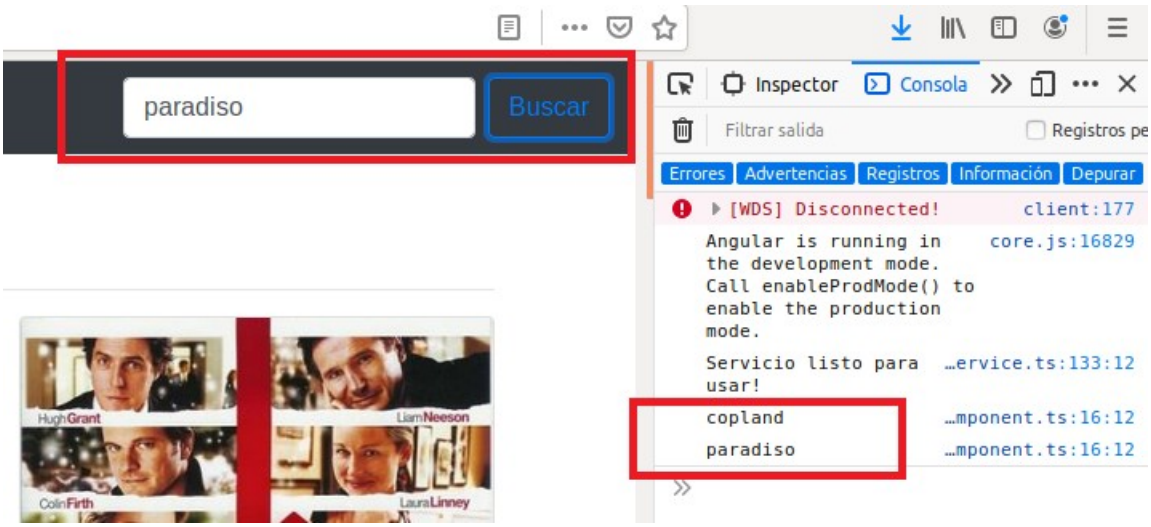
A continuación tenemos que crear el método `buscarPelicula()` en la clase del componente anterior. De momento la implementación consistirá en mostrar en la consola el mismo texto que haya escrito el usuario:

navbar.component.ts

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

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styles: []
})
export class NavbarComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
  buscarPelicula(terminoBusqueda) {
    console.log(terminoBusqueda);
  }
}
```

Ahora comprobamos que en la consola se muestra el mismo texto que introducimos en la caja de búsqueda, tanto cuando pulsamos *enter* sobre la caja como cuando pulsamos el botón:



El siguiente paso es implementar la funcionalidad del filtrado de películas en nuestro servicio. Para ello creamos el siguiente método en `PeliculasService`:

peliculas.services.ts

```
import { Injectable } from '@angular/core';
import { Pelicula } from '../models/peliculas/pelicula';

@Injectable({
  providedIn: 'root'
})
export class PeliculasService {
  private peliculas: Pelicula[] = [
      ...
  ];
  constructor() {
    console.log('Servicio listo para usar!');
  }
  getPeliculas(): Pelicula[] {
    return this.peliculas;
  }
  getPelicula(id: number): Pelicula {
      return this.peliculas.filter(peli => peli.id == id)[0];
```

```
  }
  buscarPeliculas(termino: string): Pelicula[] {
    return this.peliculas
.filter(peli => peli.titulo.toLowerCase().includes(termino.toLowerCase()));
  }
}
```

Consideraciones del método anterior:

- Mediante la función *filter* establecemos el criterio de búsqueda: solamente considerar los títulos que incluyan el término de búsqueda.
- Notad que para evitar que la búsqueda no tenga éxito a causa de diferencias entre mayúsculas y minúsculas, se decide compararlo todo a minúsculas.

A continuación tenemos que modificar el fichero de rutas para crear una nueva que reaccione con la caja y el botón de búsqueda. La idea es que el texto que escriba el usuario para buscar títulos (o subcadenas de títulos) se utilizará como un parámetro en la URL. Este parámetro lo recibirá la clase PeliculasComponent, quien en lugar de mostrar todas las películas solamente mostrará aquellas que pasen el filtro.

app.routes.ts

```
import { RouterModule, Routes } from '@angular/router';
...
const APP_ROUTES: Routes = [
  { path: 'home', component: HomeComponent},
  { path: 'about', component: AboutComponent},
  { path: 'peliculas', component: PeliculasComponent},
  { path: 'peliculas/:id', component: PeliculaComponent},
  { path: 'buscar/:termino', component: PeliculasComponent},
  { path: '**', pathMatch: 'full', redirectTo: 'home' }
];
...
```

Puesto que ya tenemos una ruta, ahoya ya podemos proporcionar una implementación correcta para el método `buscarPelicula()` de `NavbarComponent`:

navbar.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styles: []
})
export class NavbarComponent implements OnInit {
  constructor(private router: Router) { }
  ngOnInit() {
  }
  buscarPelicula(terminoBusqueda) {
    if (terminoBusqueda.trim().length > 0) {
      this.router.navigate(['/buscar', terminoBusqueda]);
    }
  }
}
```

Finalmente, tenemos que modificar `PeliculasComponent` para gestionar los dos casos siguientes:

- Si en la URL existe el parámetro `termino`, esto significa que el usuario está filtrando por algún criterio, por tanto la propiedad `peliculas` contendrá el resultado devuelto por el método `buscarPeliculas()` de nuestro servicio.
- Si en la URL no existe el parámetro `termino`, entonces la propiedad `peliculas` contendrá el resultado devuelto por el método `getPeliculas()` de nuestro servicio (que recordemos que las devuelve todas).

Implementar lo anterior conlleva bastantes cambios en esta clase. A destacar que se ha puesto el código nuevo en el método `ngOnInit()`, pues el sitio natural para hacer trabajo pesado y no en el constructor:

peliculas.component.ts

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

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

@Component({
  selector: 'app-peliculas',
  templateUrl: './peliculas.component.html',
  styles: []
})
export class PeliculasComponent implements OnInit {
  peliculas: Pelicula[] = [];
  constructor(private peliculasService: PeliculasService,
              private router: Router,
              private activatedRoute: ActivatedRoute)
  {  }

  ngOnInit() {
    this.activatedRoute.paramMap.subscribe( params => {
      if (params.get('termino') != null) {
        this.peliculas =
            this.peliculasService
                .buscarPeliculas(params.get('termino'));
      } else {
          this.peliculas = this.peliculasService.getPeliculas();
      }
    });
  }

  verPelicula(id: number) {
    this.router.navigate(['/peliculas', id]);
  }
}
```

Una vez realizado todo lo anterior, vamos a probar la funcionalidad de búsqueda. Por ejemplo, si buscamos por el texto “ci” se obtienen dos películas: “Cinema Paradiso” y “Ciudad de Dios”.

La siguiente figura muestra este caso:

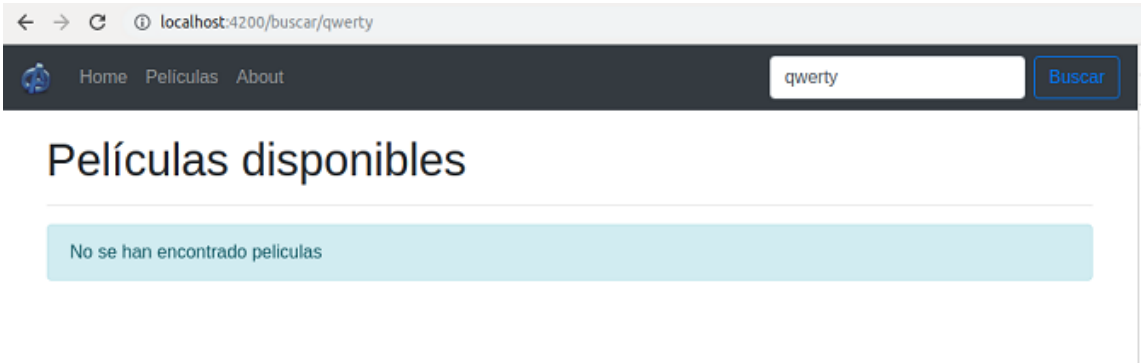
¿Y qué pasa si no se encuentran resultados para el criterio de búsqueda introducido? Pues es que tal como lo tenemos no aparecerá nada en la página. Vamos a cambiar esto. Una solución fácil es en la plantilla de PeliculasComponent mostrar un mensaje cuando la longitud del array de películas es 0:

peliculas.component.html

```
<h1>Películas disponibles</h1>
<hr>
<div class="row animated fadeIn fast" *ngIf="peliculas.length == 0">
  <div class="col-md-12">
    <div class="alert alert-info" role="alert">
      No se han encontrado peliculas
    </div>
  </div>
</div>
<div class="card-columns">
  <div class="card animated fadeIn fast" *ngFor="let pelicula of peliculas">
    <img class="card-img-top" [src]="pelicula.img" [alt]="pelicula.titulo">
    <div class="card-body">
      <h4 class="card-title">{{ pelicula.titulo }}</h4>
```

```
      <p class="card-text">{{ pelicula.sinopsis | smartTruncate: 100 }}</p>
      <p class="card-text">{{ pelicula.direccion }}</p>
      <p class="card-text">
        <small class="text-muted">{{ pelicula.anio }}</small>
      </p>
        <button (click)="verPelicula(pelicula.id)" type="button" class="btn
btn-outline-primary btn-block">
        Ver más...
      </button>
    </div>
  </div>
</div>
```

Ahora si escribimos en la caja de texto algo que sepamos que no va a coincidir con ningún título, obtendremos lo siguiente:



Conclusión

En esta tercera parte de la práctica hemos mejorado la funcionalidad de la aplicación, consiguiendo que el usuario pueda ir al detalle de una película y también que pueda realizar búsquedas por títulos o fragmentos de títulos.

En la siguiente parte, que es la final, realizaremos algunas refactorizaciones para obtener un mejor código.