Práctica Angular – Parte 2

Películas

Diseño del componente *peliculas*

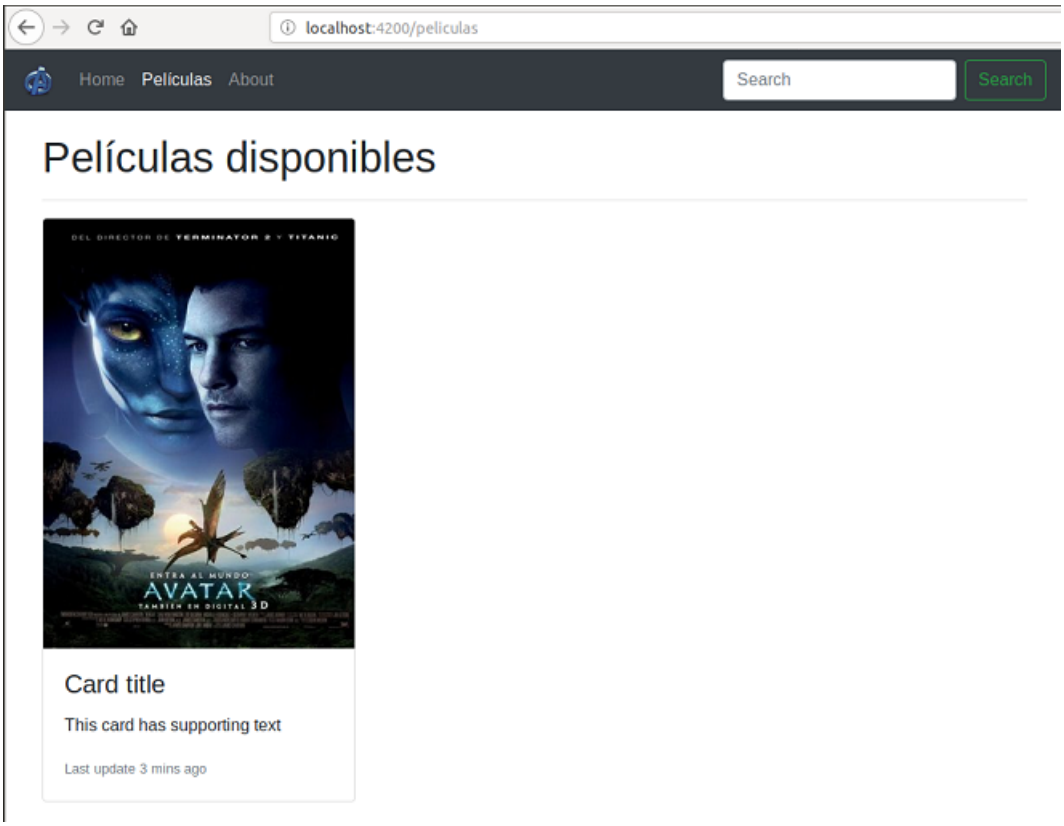
Para el diseño de este componente nos vamos a basar en un elemento llamado “Cards” de Bootstrap. Las “cards” son un tipo de contenedores que se adaptan a su contenido, de manera similar a como la web de Pinterest visualiza el contenido.

Partiremos del siguiente código que tenemos que pegar en el fichero `peliculas.component.html`, reemplazando cualquier contenido previo. De momento ponemos “hardcoded” la portada de la película “Avatar”, solamente para hacernos una idea de cómo se a visualizar cada “Card”:

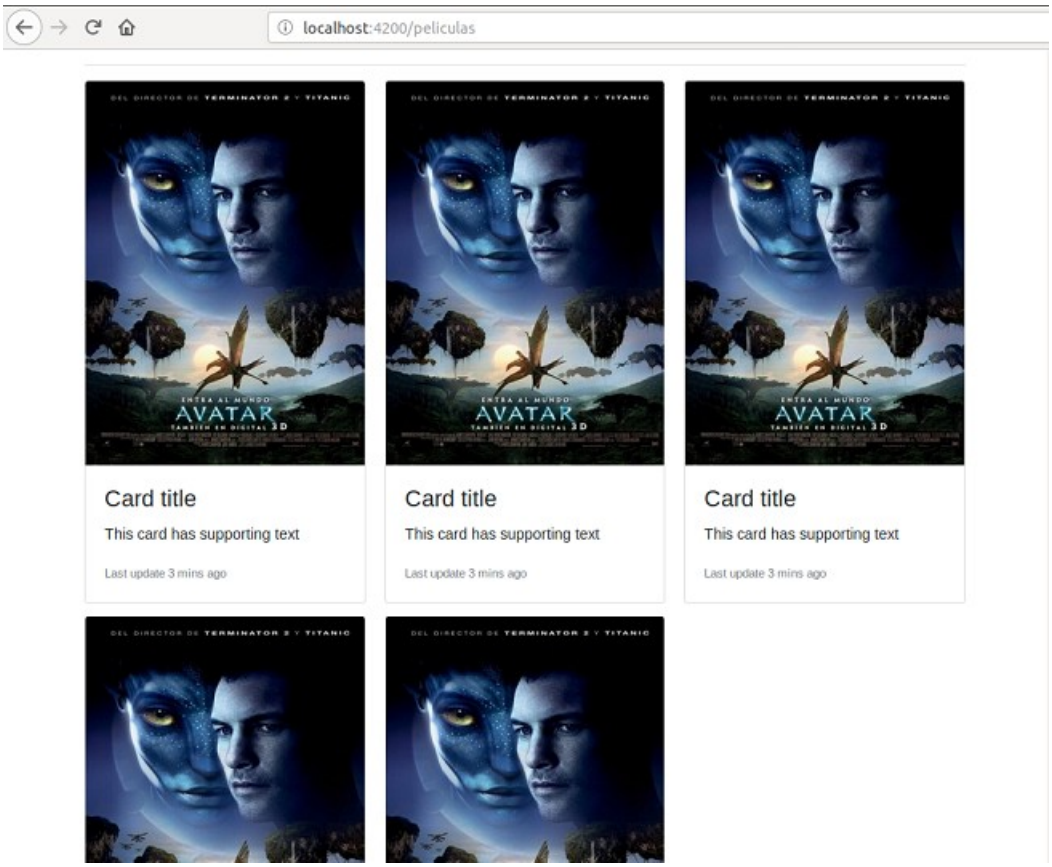
peliculas.component.html

```
<h1>Películas disponibles</h1>
<hr>
<div class="card-columns">
   <div class="card" >
      <img class="card-img-top"
           src="assets/img/avatar.jpg"
           alt="portada avatar">
      <div class="card-body">
         <h4 class="card-title">Card title</h4>
         <p class="card-text">This card has supporting text</p>
         <p class="card-text">
            <small class="text-muted">Last update 3 mins ago</small>
         </p>
      </div>
   </div>
</div>
```

Deberíamos ver algo como lo siguiente:



Si ahora, a modo de prueba, copiamos y pegamos 4 o 5 veces el `<div class="card">` y todo su contenido, obtendremos lo siguiente:

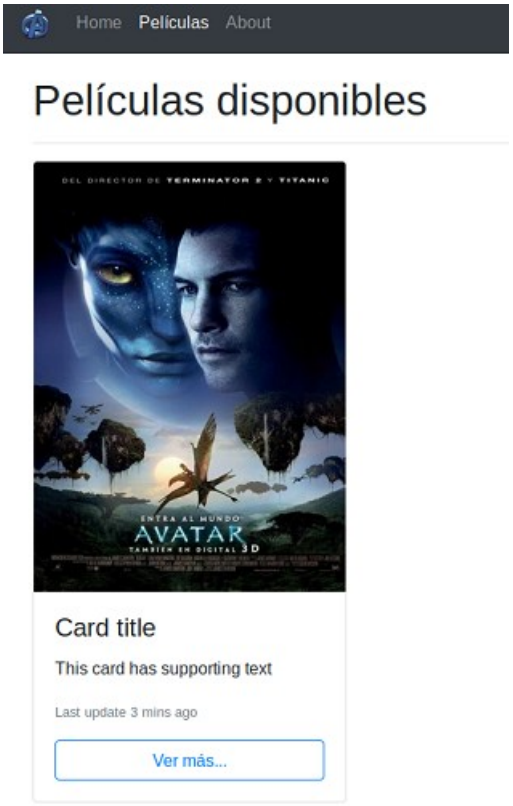


Lógicamente, cada película deberá presentar su propio contenido. Para obtener el contenido implementaremos en breve un servicio de Angular. Ya podemos deshacer el cambio anterior.

Continuamos. Desde la página de las películas queremos poder seleccionar una película en particular y ver información más detalla, como la sinopsis, por ejemplo. Para esto, vamos a añadir un botón a cada “card”:

peliculas.component.html

```
<h1>Películas disponibles</h1>
<hr>
<div class="card-columns">
  <div class="card">
        <img  class="card-img-top"  src="assets/img/avatar.jpg"  alt="portada avatar">
    <div class="card-body">
      <h4 class="card-title">Card title</h4>
      <p class="card-text">This card has supporting text</p>
      <p class="card-text">
        <small class="text-muted">Last update 3 mins ago</small>
      </p>
      <button type="button" class="btn btn-outline-primary btn-block">
        Ver más...
      </button>
    </div>
  </div>
</div>
```



Ahora pequeñas mejoras estéticas. Por un lado, cambiamos el color del botón de la barra del *navbar* para que se muestre el texto azul en lugar de verde. Para esto hemos de cambiar la clase del botón de `btn-outline-success` a `btn-outline-primary`:

navbar.component.html

```
...
      <input class="form-control mr-sm-2" type="search"
          placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-primary my-2 my-sm-0"
          type="submit">Search</button>
    </form>
  </div>
</nav>
```

Por otro lado, vamos a modificar la hoja de estilos general para que el contenido de las páginas no quede tan pegado al final del navegador. Para ello, añadimos al principio del fichero styles.css el siguiente estilo:

styles.css

```
body {
  padding-bottom: 50px;
}
```

Implementando un servicio

La página de películas necesita mostrar información relativa a las películas. En un primer momento podríamos pensar que tal información podría residir en esta página. Ahora bien, ¿qué pasaría si otras páginas también necesitaran esta misma información o un subconjunto de esta información? Lo que no es admisible es repetir información, ya que eso implicaría incumplir el principio de diseño DRY (Don't Repeat Yourself).

La solución en Angular pasa por la creación de un servicio. Un servicio tiene las siguientes características:

- Ofrecer objeto Singleton y mediante Inyección de Dependencias.

- Brindar información a quien la necesite.
- Realizar peticiones CRUD (create, read, update, delete), normalmente a un *backend* que exponga un API Rest.
- Gestionar la persistencia de los datos.
- Servir como recurso reutilizable en la aplicación.

Para crear un servicio:

```
ng g s services/peliculas --spec=false
```

Lo anterior habrá generado el siguiente fichero:

peliculas.services.ts

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

@Injectable({
  providedIn: 'root'
})
export class PeliculasService {
  constructor() { }
}
```

Modifiquemos el constructor anterior:

```
constructor() {
    console.log('Servicio listo para usar!');
}
```

Ahora hemos de informar de este servicio en el módulo principal de la aplicación:

app.module.ts

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

```
// Rutas
import { APP_ROUTING } from './app.routes';

// 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 { PeliculasComponent } from
'./components/peliculas/peliculas.component';

// Servicios
import { PeliculasService } from './services/peliculas.service';

@NgModule({
  declarations: [
    AppComponent,
    NavbarComponent,
    HomeComponent,
    AboutComponent,
    PeliculasComponent
  ],
  imports: [
    BrowserModule, APP_ROUTING
  ],
  providers: [PeliculasService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Finalmente tenemos que utilizar el servicio en los componentes que nos interesen. En estos momentos solamente nos interesa que Angular inyecte el servicio en la clase `PeliculasComponent`:

peliculas.component.ts

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

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

Si ahora vamos a la consola del navegador deberíamos ver el mensaje “Servicio listo para usar!”:

Ahora que tenemos la infraestructura del servicio implementada vamos a darle la funcionalidad adecuada. Para ello, necesitamos abrir el fichero `datos.txt`, el cual venía en el fichero de recursos que acompañaba esta práctica (recursos.zip). Una vez abierto el fichero anterior, seleccionamos todo el contenido y lo pegamos justo antes del constructor de nuestro servicio `PeliculasService`. El código que acabamos de pegar tenemos que asignarlo a un propiedad privada llamada `peliculas` y de tipo `any[]`. A continuación se muestra un resumen de como debería quedar esta propiedad:

peliculas.service.ts

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

@Injectable({
  providedIn: 'root'
})
export class PeliculasService {
```

```
  private peliculas: any[] = [
    {
      id: 1001,
      titulo: "Copland",
        sinopsis: "Babitch (Michael Rapaport), un policía de Nueva York,
conduce...",
      img: "assets/img/copland.jpg",
      anio: 1997,
      direccion: "James Mangold"
    },
    {
      id: 1002,
      titulo: "Cadena perpetua",
       sinopsis: "Acusado del asesinato de su mujer, Andrew Dufresne (Tim
Robbins)...",
      img: "assets/img/cadena_perpetua.jpg",
      anio: 1994,
      direccion: "Frank Darabont"
    },
    ...
    {
      id: 1015,
      titulo: "Avatar",
      sinopsis: "Año 2154. Jake Sully (Sam Worthington), un ex-marine...",
      img: "assets/img/avatar.jpg",
      anio: 2009,
      direccion: "James Cameron"
    }
  ];

  constructor() {
    console.log('Servicio listo para usar!');
  }
}
```

Ahora bien, dado que la propiedad `peliculas` es privada, no podemos acceder a ella desde fuera de esta clase. Así que vamos a crear un método público que retorno el array contenido en esta propiedad.

Lo creamos justo después del constructor:

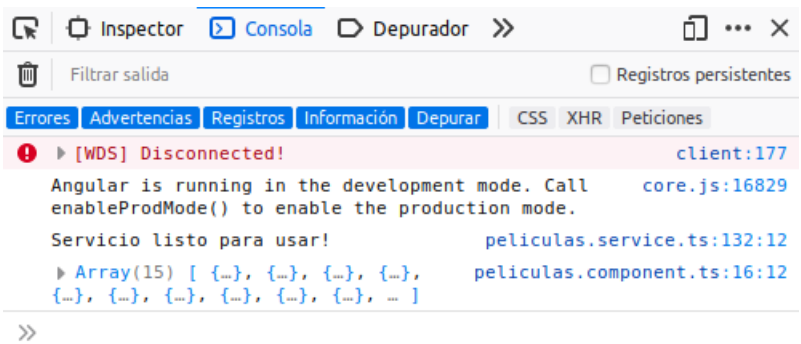
```
  getPeliculas() {
    return this.peliculas;
  }
```

Volvemos al componente de las películas y añadimos una propiedad llamada `peliculas` para almacenar la información devuelta por el servicio. Notad que se ha optado por poner el código que ejecuta el método `getPeliculas()` del servicio (y asigna su resultado a la propiedad `peliculas`) en el método `ngOnInit()`, que es un método del ciclo de vida de los componentes. Este método se ejecuta cuando la página está lista para mostrarse, lo cual sucede después que la ejecución del constructor.. En este caso en particular es indiferente poner el código en el constructor que en `ngOnInit()`.

peliculas.component.ts

```
import { Component, OnInit } from '@angular/core';
import { PeliculasService } from '../../services/peliculas.service';
@Component({
  selector: 'app-peliculas',
  templateUrl: './peliculas.component.html',
  styles: []
})
export class PeliculasComponent implements OnInit {
  peliculas: any[] = [];
  constructor(private peliculasService: PeliculasService) { }
  ngOnInit() {
    console.log(this.peliculas = this.peliculasService.getPeliculas());
  }
}
```

Ahora en el navegador tendríamos que ver que aparecen los 15 objetos en la consola:



Creación de una interfaz para las películas

En lugar de trabajar con el tipo `any` sería conveniente definir una interfaz que detallara en qué consiste un objeto de tipo película.

Creamos la interfaz dentro de la ruta de carpetas `models/peliculas`:

```
ng g i models/peliculas/pelicula --spec=false
```

La interfaz estará vacía. Definimos las siguientes propiedades, todas obligatorias:

pelicula.ts

```
export interface Pelicula {
  id: number;
  titulo: string;
  sinopsis: string;
  img: string;
  anio: number;
  direccion: string;
}
```

Ahora utilizaremos la interfaz `Pelicula` tanto en el servicio como en el componente de películas:

peliculas.service.ts

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

@Injectable({
  providedIn: 'root'
})
export class PeliculasService {
  private peliculas: Pelicula[] = [
    ...
  ];
  constructor() {
    console.log('Servicio listo para usar!');
```

```
  }
  getPeliculas(): Pelicula[] {
    return this.peliculas;
  }
}
```

peliculas.component.ts

```
import { PeliculasService } from '../../services/peliculas.service';
import { Pelicula } from '../../models/peliculas/pelicula';

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

Todo debería seguir funcionando como antes, aunque ahora tenemos un código de tipado más fuerte que anteriormente.

Modificar la plantilla de películas

Sabemos que en la clase `PeliculasComponent` disponemos de una propiedad llamada `peliculas`, de tipo array, que contiene todos los datos de las películas que necesitamos mostrar en las “Cards”. Esta información nos la ha proporcionado el servicio que creamos anteriormente.

Ahora se trata de modificar la plantilla del componente para pintar las “Cards” con la información del array. Para ello utilizaremos la directiva estructural `*ngFor` y el *databinding*.

La plantilla ha de quedar como sigue:

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>
      <button type="button" class="btn btn-outline-primary btn-block">
        Ver más...
      </button>
    </div>
  </div>
</div>
```

Deberíamos ver algo parecido a esto:

Creación de un Pipe personalizado

Mediante un pipe personalizado vamos a controlar la cantidad de caracteres a mostrar en las "Cards" para el texto de la sinopsis. De esta manera no aparecerá tanta información en esta página. Si el usuario quiere ver toda la sinopsis, entonces tendrá que acceder a la página de detalle.

Pasos a seguir:

1) Creamos el pipe:

```
ng g p pipes/smart-truncate –spec=false
```

Lo anterior habrá creado el pipe y lo habrá declarado en `app.module.ts`.

smart-truncate.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'smartTruncate'
})
export class SmartTruncatePipe implements PipeTransform {
  transform(value: any, args?: any): any {
    return null;
  }
}
```

app,module.ts

```
// Modulos
import { BrowserModule } from '@angular/platform-browser';
...
// Servicios
import { PeliculasService } from './services/peliculas.service';
// Pipes
import { SmartTruncatePipe } from './pipes/smart-truncate.pipe';

@NgModule({
  declarations: [
```

```
    AppComponent,
    NavbarComponent,
    HomeComponent,
    AboutComponent,
    PeliculasComponent,
    PeliculaComponent,
    SmartTruncatePipe
  ],
  imports: [
    BrowserModule, APP_ROUTING
  ],
  providers: [PeliculasService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2) Creamos código para el pipe:

El siguiente código permite que desde la plantilla que lo necesitemos, pasemos dos parámetros: el texto a recortar y la longitud a partir de la que hay que recortar:

smart-truncate.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'smartTruncate'
})
export class SmartTruncatePipe implements PipeTransform {
  transform(cadena: any, maxSize: any): any {
    const token = cadena.substr(0, parseInt(maxSize, 10));
    if (token.length < cadena.length) {
      return token.substr(0, token.lastIndexOf(' ')) + '...';
    }
    return cadena;
  }
}
```

3) Usar SmartTruncate en la plantilla de PeliculasComponent:

Indicaremos que solamente muestre los primeros 100 caracteres de la sinopsis.

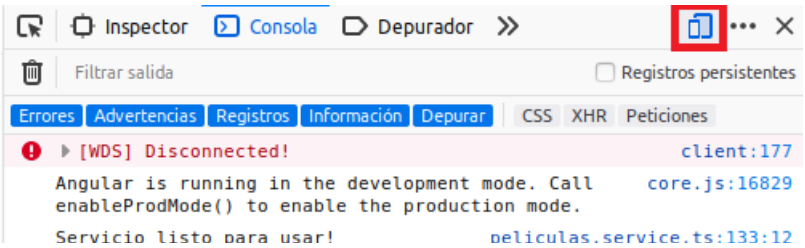
peliculas.component.html

```
<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 | 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>
```

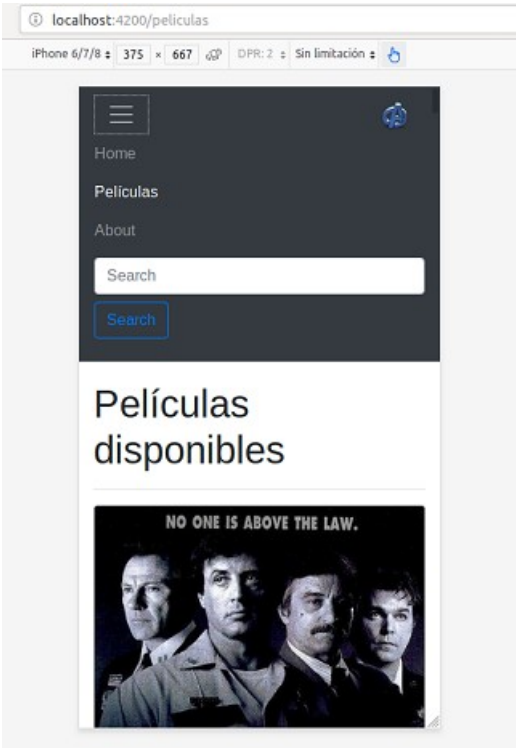
Deberíamos visualizar lo siguiente:

Visualización en dispositivo móvil

Si pulsamos el icono para visualizar en dispositivo móvil:



Veremos que gracias a Bootstrap y sus capacidades "responsive" seguiremos teniendo toda la funcionalidad del *navbar* y además todo el contenido se verá bien:



Conclusión

En esta segunda parte de la práctica hemos avanzado mucho. Hemos realizado lo siguiente:

- Diseño de la página de películas mediante las "Cards" de Bootstrap.
- Implementación de un servicio para disponer de datos sobre películas desde cualquier componente.
- Creación de un modelo de datos basado en la interfaz Pelicula.

- Creación de un Pipe personalizado.
- Hemos comprobado que mediante Bootstrap mantenemos capacidades *responsive*.

En la siguiente parte de esta práctica crearemos la página que muestra el detalle de una película al ser seleccionada desde la página que muestra todas las películas. También implementaremos la funcionalidad para buscar (filtrar) películas por título en base a un texto introducido por el usuario.