Lección 10 – Documentación Practica

1. El alumno debe crear un componente que vaya a tener un listado

Primero creamos el componente “Games” y creamos una variable llamada “gameList” el cual contendrá un array de la interfaz “Game”

```
export class GamesComponent implements OnInit {
  public gamesList!: Game[]
```

\src\app\pages\games\games.component.ts

Para poder hacer esto, necesitamos hacer también la interfaz comentada, esto creando la carpeta “Model” y añadiendo el archivo “game.interface.ts

```
export interface Game{
    name: string;
    released : string;
    background_image: string;
    metacritic: string;
}
```

\src\app\model\game.interface.ts

Dentro de esta interfaz debemos indicar y tipar los datos que vamos a querer utilizar de la respuesta que nos devolverá la API.

Una vez creado, desde el componente Games nos permitirá importar la interfaz.

2. Se creará un componente card que contenga la información de cada item.

Dentro del componente creado, indicaremos dentro del archivo HTML vamos a recorrer la lista que hemos creado con el atributo *ngFor

```
<article *ngFor="let game of gamesList" appClicGame>
```

Una vez hecho esto, dentro del contenedor, podemos ir añadiendo los campos que hemos mapeado en la interfaz y que recogemos con la llamada a la API.

```
<article *ngFor="let game of gamesList" appClicGame>
    <h4>{{game.name}}</h4>
    <img src="{{game.background_image}}" alt="{{game.name}}">
    <div>
        <p>{{game.released}} / Metacritic: {{game.metacritic}}</p>
    </div>
</article>
```

\src\app\pages\games\games.component.html

3. Se debe de realizar una petición desde el componente list a un servicio, y que este se traiga el array con los ítems necesarios.

Primero creamos el servicio con el comando “ng g s game”. Una vez creado importamos las librerías “Observable”, “HttpClient” y la interfaz “Game”

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Game } from '../model/game.interface';
import { Observable } from 'rxjs';
```

\src\app\servicios\game.service.ts

Una vez hecho esto, dentro del constructor, inicializamos la variable “httpClient”

```
  constructor(
    private httpClient: HttpClient
  ) { }
```

\src\app\servicios\game.service.ts

Una vez esto, creamos la función “findGame()” que devolverá un Observable de la interfaz que tenemos y dentro, realizamos la petición GET a la API.

```
public findGame(): Observable<{results: Game[]}>{
    return this.httpClient.get<{results:
Game[]}>("https://api.rawg.io/api/games?key=f3a3f996b19e406eb2b1a9b9a86a3
a08")
  }
```

\src\app\servicios\game.service.ts

Una vez hecho esto, desde el componente que tenemos para la lista de juegos, creamos la función “getGames” donde nos suscribiremos al servicio “gameService” y guardaremos la respuesta en la variable “gameList”

```
private getGames(): void{
    this.gameService.findGame().subscribe({
      next: ( resp ) => {
        this.gamesList = resp.results
      }, error: (err) => {
        console.log(err)
      }

    })
  }
```

\src\app\pages\games\games.component.ts

Una vez hecho esto, con la platilla que hemos creado en HTML ya te debería de aparecer los valores recuperados de la API

4. Cuando pulsemos sobre un card, el componente padre (el listado) debe de mostrar quien se ha seleccionado

Para realizar esto, podemos crear una directiva que aplique los eventos “MouseOver”, “MouseOut” y “Click”.

En mi caso, he utilizado los dos primeros para cambiar el fondo y al pulsar sobre el juego, saltar una alerta con el nombre de dicho juego.

```
constructor(
  private elementRef : ElementRef
) { }

@HostListener('mouseover')
public onMouseOver(){
  this.elementRef.nativeElement.style.background='orange'
}

@HostListener('mouseout')
public onMouseOut(){
  this.elementRef.nativeElement.style.background='white'
}

@HostListener('click')
public onClick(){
  alert("Has pulsado sobre el juego " + this.elementRef.nativeElement.children[0].innerHTML)
}
```

\src\app\directivas\clic-game.directive.ts

*Realizando la actividad vi los comentarios sobre las practicas de las lecciones anteriores y he modificado los puntos que me comentaste:

```
<nav>
    <li>
        <a href="" [routerLink]="['home']"><ul appRedColor>Inicio</ul></a>
        <a href="" [routerLink]="['formulario']"><ul appRedColor>Formulario</ul></a>
        <a href="" [routerLink]="['curriculum-vitae']"><ul appRedColor>CV</ul></a>
        <a href="" [routerLink]="['games']"><ul appRedColor>Games</ul></a>
        <a href="" [routerLink]="['login']"><ul appRedColor>LogIn</ul></a>
    </li>
</nav>
```

Quite los href del nav para que no cargue de nuevo la pagina y solo cambie el contenido del router-outlet

Del apartado del login, no se si se llego a subir una versión anterior y por esa razón no salía. Lo he revisado y he añadido un alert para cuando no se cumplen y he revisado la parte de que se muestren los datos si se cumplen las condiciones:

```
public validate(){
  let regexp = new RegExp(/\S+@\S+\.\S+/)
  if (this.datos.password.length > 8 && regexp.test(this.datos.email) ) { this.isValid = true}
  if (!this.isValid) {alert('Uno de los datos introducidos no tiene el formato correcto')}
}
```

```
<div id="datos" *ngIf="isValid">
    <p *ngIf="datos.email">Email : {{datos.email}}</p>
    <p *ngIf="datos.password">Password : {{datos.password}}</p>
</div>
```