Aplicaciones SPA Lección 10: Servicios, petición httpClient

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

```
➜  ejercicios git:(leccion10) ✗ ng g c components/card-list
CREATE src/app/components/card-list/card-list.component.scss (0 bytes)
CREATE src/app/components/card-list/card-list.component.html (24 bytes)
CREATE src/app/components/card-list/card-list.component.spec.ts (641 bytes)
CREATE src/app/components/card-list/card-list.component.ts (287 bytes)
UPDATE src/app/app.module.ts (496 bytes)
```

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

```
➜  ejercicios git:(leccion10) ✗ ng g c components/card
CREATE src/app/components/card/card.component.scss (0 bytes)
CREATE src/app/components/card/card.component.html (19 bytes)
CREATE src/app/components/card/card.component.spec.ts (612 bytes)
CREATE src/app/components/card/card.component.ts (268 bytes)
UPDATE src/app/app.module.ts (581 bytes)
```

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, deberemos crear el servicio:

```
➜  ejercicios git:(leccion10) ✗ ng g service services/cocktail
CREATE src/app/services/cocktail.service.spec.ts (367 bytes)
CREATE src/app/services/cocktail.service.ts (137 bytes)
```

Tras analizar la API, deberemos crear las interfaces que nos sean necesarias para modelar la dicha API de formato JSON a formato de interfaz de TS:

```
import Cocktail from "./Cocktail";

export default interface CocktailList {

    drinks: Cocktail[];
}
```

```typescript
export default interface Cocktail {

    strDrink: String;
    idDrink: String;

    strGlass: String;

    strInstructions: String;

    strDrinkThumb: String;

    strIngredient1: String;
    strIngredient2: String;
```


```
{
  "drinks": [
    {
      "idDrink": "11007",
      "strDrink": "Margarita",
      "strDrinkAlternate": null,
      "strTags": "IBA,ContemporaryClassic",
      "strVideo": null,
      "strCategory": "Ordinary Drink",
      "strIBA": "Contemporary Classics",
      "strAlcoholic": "Alcoholic",
      "strGlass": "Cocktail glass",
      "strInstructions": "Rub the rim of the glass with the lime slice to make the salt stick to it. Take care to moisten only the ou
      "strInstructionsES": null,
      "strInstructionsDE": "Reiben Sie den Rand des Glases mit der Limettenscheibe, damit das Salz daran haftet. Achten Sie darauf, d
und vorsichtig in das Glas geben.",
      "strInstructionsFR": null,
      "strInstructionsIT": "Strofina il bordo del bicchiere con la fetta di lime per far aderire il sale.\r\nAvere cura di inumidire
nel bicchiere.",
      "strInstructionsZH-HANS": null,
      "strInstructionsZH-HANT": null,
      "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/5noda61589575158.jpg",
      "strIngredient1": "Tequila",
      "strIngredient2": "Triple sec",
```


(no sé por qué esta API contiene elementos en variables singulares en vez de en arrays)

También deberemos añadir las librerías en el archivo [servicio].service.ts para hacerlo inyectable, para hacer la propia request y para observar sus datos cuando se descarguen:

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
```

Importamos la interfaz que acabamos de crear y añadimos los métodos que nos hacen falta para obtener diferentes datos de la API:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import CocktailList from '../interfaces/cocktails/CocktailList';

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

  private cocktailUrl: String = 'https://thecocktaildb.com/api/json/v1/1';


  constructor(private httpClient: HttpClient) {

  }

  getRandomCocktail(): Observable<CocktailList> {
    return this.httpClient.get<CocktailList>(`${this.cocktailUrl}/random.php`)
  }

  getAlcoholicDrinks(): Observable<CocktailList> {
    return this.httpClient.get<CocktailList>(`${this.cocktailUrl}/filter.php?a=Alcoholic`)
  }

  getDrinkById(id: String): Observable<CocktailList> {
    return this.httpClient.get<CocktailList>(`${this.cocktailUrl}/lookup.php?i=${id}`)
  }

}
```

Deberemos añadir nuestro servicio y el módulo de peticiones HTTP a nuestro archivo app.modules.ts en imports y providers respectivamente:

```
@NgModule({
  declarations: [
    AppComponent,
    CardListComponent,
    CardComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [CocktailService],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

En el componente padre, deberemos hacer una llamada al servicio para conseguir el observable del array de elementos de la API en el método ngOnInit:

```
export class CardListComponent implements OnInit {

  public cocktailList!: Observable<CocktailList>;
  public cocktailError: String;
  public cocktailClicked?: Cocktail;
  constructor(private cocktailService: CocktailService) {
    this.cocktailError = "";
  }

  ngOnInit(): void {
    this.cocktailList = this.cocktailService.getAlcoholicDrinks();
  }

  public showDrinkClicked(cocktailClicked: Cocktail) {
    this.cocktailService.getDrinkById(cocktailClicked.idDrink).subscribe(
      res => { this.cocktailClicked = res.drinks[0] },
      error => { this.cocktailError = error }
    )
  }
}
```

Esta lista de elementos se mostrará en el html padre mediante una directiva ngFor (y una pipe asíncrona) creando un elemento card (hijo) que tenga la propiedad que corresponde (“drink”):

```
<div id="card-list-root" *ngIf="cocktailError.length == 0">
    <ng-container *ngIf="cocktailList | async as list">

        <app-card *ngFor="let cocktail of list.drinks" [drink]="cocktail" (drinkClicked)="showDrinkClicked($event)">
        </app-card>
    </ng-container>


</div>
```

(el atributo “drinkClicked” es para atrapar la card hija clicada)

En el componente hijo deberemos atrapar la propiedad mediante el decorador @Input([nombre de la propiedad]):

```
export class CardComponent implements OnInit {

  @Input('drink') drink!: Cocktail;
```

Deberemos sacar este elemento en nuestro HTML:

```
<div class="drink" (click)="clickDrink()">
    <img src="{{drink.strDrinkThumb}}" alt="{{drink.strDrink}}" height="200" width="300" />
    <p class="strDrink">{{drink.strDrink}}</p>

</div>
```

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

Cuando se genera un card, se le asigna una propiedad de clicado el cual enviará el elemento que tenga el hijo al padre mediante el decorador @Output:

```
<div class="drink" (click)="clickDrink()">
    <img src="{{drink.strDrinkThumb}}" alt="{{drink.strDrink}}" height="200" width="300" />
    <p class="strDrink">{{drink.strDrink}}</p>

</div>

export class CardComponent implements OnInit {

  @Input('drink') drink!: Cocktail;

  @Output() drinkClicked: EventEmitter<Cocktail> = new EventEmitter<Cocktail>();

  constructor() {}

  ngOnInit(): void {
  }

  public clickDrink() {
    this.drinkClicked.emit(this.drink);
  }

}
```

Esto se “atrapa” en el padre con un gestor de eventos, en mi caso se llama “showDrinkClicked” y le paso mi evento que contiene mi elemento hijo:

```
<div id="card-list-selected">
    <pre>
        {{cocktailClicked | json}}
    </pre>
</div>

<div id="card-list-root" *ngIf="cocktailError.length == 0">
    <ng-container *ngIf="cocktailList | async as list">
        <app-card *ngFor="let cocktail of list.drinks" [drink]="cocktail" (drinkClicked)="showDrinkClicked($event)">
        </app-card>
    </ng-container>
</div>

<div id="card-list-error" *ngIf="cocktailError.length != 0">
    {{cocktailError | json}}
</div>
```

En el padre, hago una búsqueda del elemento que me corresponde por ID para conseguir la información completa del elemento clicado:

```
public showDrinkClicked(cocktailClicked: Cocktail) {
  this.cocktailService.getDrinkById(cocktailClicked.idDrink).subscribe(
    res => { this.cocktailClicked = res.drinks[0] },
    error => { this.cocktailError = error }
  )
}
```

El resultado final es:


```
{
  "idDrink": "15395",
  "strDrink": "1-900-FUK-MEUP",
  "strDrinkAlternate": null,
  "strTags": null,
  "strVideo": null,
  "strCategory": "Shot",
  "strIBA": null,
  "strAlcoholic": "Alcoholic",
  "strGlass": "Old-fashioned glass",
  "strInstructions": "Shake ingredients in a mixing tin filled with ice cubes. Strain into a rocks glass.",
  "strInstructionsES": null,
  "strInstructionsDE": "Die Zutaten in einem mit Eiswürfeln gefüllten Mischgefäß schütteln. In ein Rocks Glas abseihen.",
  "strInstructionsFR": null,
  "strInstructionsIT": "Shakerare gli ingredienti in una teglia piena di cubetti di ghiaccio.\r\nFiltrare in un bicchiere alto.",
  "strInstructionsZH-HANS": null,
  "strInstructionsZH-HANT": null,
  "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/uxywyw1468877224.jpg",
  "strIngredient1": "Absolut Kurant",
  "strIngredient2": "Grand Marnier",
  "strIngredient3": "Chambord raspberry liqueur",
  "strIngredient4": "Midori melon liqueur",
  "strIngredient5": "Malibu rum",
  "strIngredient6": "Amaretto",
  "strIngredient7": "Cranberry juice",
  "strIngredient8": "Pineapple juice",
  "strIngredient9": null,
  "strIngredient10": null,
  "strIngredient11": null,
  "strIngredient12": null,
  "strIngredient13": null,
  "strIngredient14": null,
  "strIngredient15": null,
  "strMeasure1": "1/2 oz ",
  "strMeasure2": "1/4 oz ",
  "strMeasure3": "1/4 oz ",
  "strMeasure4": "1/4 oz ",
  "strMeasure5": "1/4 oz ",
  "strMeasure6": "1/4 oz ",
  "strMeasure7": "1/2 oz ",
  "strMeasure8": "1/4 oz ",
  "strMeasure9": null,
  "strMeasure10": null,
  "strMeasure11": null,
  "strMeasure12": null,
  "strMeasure13": null,
  "strMeasure14": null,
  "strMeasure15": null,
  "strImageSource": null,
  "strImageAttribution": null,
  "strCreativeCommonsConfirmed": "No",
  "dateModified": "2016-07-18 22:27:04"
}
```


1-900-FUK-MEUP

110 in the shade