¿Qué son los servicios?

Si cada componente se encarga de obtener datos en sus ficheros .ts, ¿qué pasaría si varios componentes necesitaran los mismos datos? Este problema se podría solucionar si el componente llamara a una función externa al componente, que traiga esos datos a través de apis. De esta forma, varios componentes podrían llamar a dicha función para obtener dichos datos y cada uno luego tratarlos en cada componente. Estas consultas de base de datos se harían en servicios. Esos servicios serán proporcionados a esos componentes que lo soliciten, lo que se conoce como 'inyección de dependencias': "Se inyecta el servicio en el componente".

- Cualquier lógica de negocio que no tiene interfaz de usuario: almacenamiento de datos, comunicaciones, cálculos, etc.. las implementamos en servicios.
- Se implementan como clases.
- Se ubican en una carpeta /**services**, dentro de app

Ej: Servicio Logger. Creamos un fichero logger.service.ts. (por convención es nombre.service.ts)

```
export class Logger {
        log(msg: any) { console.log(msg); }
        error(msg: any) { console.error(msg); }
        warn(msg: any) { console.warn(msg); }
}
```

Para crear un servicio se utiliza:

```
ng generate service services/nombreServicio
                    o
ng g s services/nombreServicio
```

Los servicios tienen el decorador @Injectable. ¿Qué significa? Que angular hace automáticamente el new del servicio. El servicio permitirá conectarnos a las diferentes apis, a través del objeto HttpClient. Es decir, HttpClient es el mecanismo de Angular para comunicarse con un servidor remoto a través de HTTP.
Con el objeto httpClient podremos acceder a los métodos: get/ put/post/delete/...

Todos los métodos de HttpClient devuelven un RxJS Observable de algo. Los observables son flujos de datos, una colección de eventos que en algún momento se pueden emitir. Te permite la manipulación asíncrona de datos (programación reactiva)

HttpCliente:
Nos permite acceder a internet para hacer las llamadas a las API. Pasos:

1.- Inyectar en el constructor de nuestro servicio:

constructor (**private http: HttpClient**)

Se importará: import { HttpClient } from '@angular/common/http';

2.- IMPORTANTE: En el fichero: app.config.ts hay que importar la clase provideHttpClient:

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

Eso importará: `import { provideHttpClient } from '@angular/common/http';`

3.- Como en el servicio hemos inyectado el HttpClient en el constructor, ya podremos utilizar los métodos para conectarnos a la api:

Ejemplo:

```
getListUsuarios(){
        this.http.get(this.url);
}
```

4.- Para usar un servicio en un componente, se inyecta el servicio en el constructor. En el componente decimos que implemente la interfaz OnInit. Y necesitamos el constructor y el método ngOnInit.
En el constructor inyectamos el servicio como parámetro:

```
export class AppComponent implements OnInit {
  pokemonList$ = new Observable<pokemonList>();

  constructor(private pokemonService: PokemonService) {}

  ngOnInit(): void {
    this.pokemonList$ = this.pokemonService.getpokemonListParam(0, 20);
  }
```

Donde la variable que devuelve el servicio es un Observable. (Importar las operaciones para Observables (dependencia): import 'rxjs/operator)

5.- AsyncPipe. Una vez que ya tenemos el observable, se pinta el resultado en la vista usando la pipe Async:

Ejemplo:

```
<li *ngFor="let item of pokemonList$ | async" >
```

- Se puede usar también un alias: Ejemplo:

Ejemplo:

```
<li *ngIf="variableUser$ | async as userData" >
        <h2>{{ userData.name }}</h2>
</li>
```

- Async se usa sobre el observable. Si queremos obtener algún dato, necesitaremos usar el paréntesis: Ejemplo:

```
<h2 *ngIf="(userData$ | async)?.fullName">
```

NOTA: para generar los tipos en typeScript y crear las interfaces, es útil utilizar Quicktype.io.

Ejercicios

1.- Crear un proyecto que se llame pokemon1 que muestre en el componente principal los nombres de los pokemon. Para ello nos conectaremos a la api: https://pokeapi.co/api/v2/pokemon?offset=10&limit=20.

RECORDAMOS PASOS:

- Crear servicio en una carpeta services/
- En el servicio, creamos constructor e inyectamos el objeto HttpClient.
- En el servicio, crearemos los métodos que queramos para llamar a this.http.get
- En app.config.ts importamos provideHttpClient()
- Crear un modelo de datos, en interfaces, en la carpeta Model.
- El componente debe implementar la interfaz onInit() inyectamos el servicio en el constructor .
- En el método ngOnInit llamamos el método de nuestro servicio. - En la vista del componente, accedemos al observable y con async obtenemos datos

2.- Crear un proyecto que se llame planetas1 para mostrar por pantalla el nombre de los planetas que nos devuelve la api de star wars : https://swapi.dev/api/planets/?page=1
Crear el servicio en: services/planetas: ng g service services/planetas

Crear una clase **planet.ts** que tendrá todos los atributos del planeta, desde name hasta population(en una carpeta model) Tendrá todos los atributos que devuelve la api: export interface Planet{ .....}

Dentro del servicio vamos a crear un método: getPlanets() que devuelve un un listado de planetas. Recibe un número de página. Por defecto a 1 si viene vacío.

Crear un componente **PlanetList** para mostrar la lista de planetas.

Crear otro componente **PlanetDetail** que mostrará el detalle de un planeta, por ejemplo el planeta Ambos componentes serán llamados desde el componente principal.

3.- Crear una aplicación de personajes del señor de los anillos que permita mostrar un listado de personajes, agregar y eliminar. Necesitamos crear un servicio: services/anillos

- Para cada personaje guardar el nombre y raza. Crear una *interfaz* '**Personaje**' con los atributos id, nombre y raza. (en carpeta model)
- Crear un array de Personaje: personajes con al menos 3 en el *servicio*: **personaje.service**
- El *componente principal* tendrá un título centrado e invocará a un componente **main**.
- El *componente main* tendrá un título y estará formado por dos columnas:
  - Primera columna: componente *list-personaje*. Muestra la lista de los personajes id, nombre, raza y un botón para eliminar del array dicho personaje. (Para mostrar los personajes se llamará a un método getPersonajes; para borrarlo: eliminarPersonaje)
  - Segunda columna: componente *add-personaje*. Tendrá dos input de tipo text para agregar el nombre y la raza de un personaje. También un botón agregar que al pulsar agregue el personaje a la lista y deje los campos de los input vacíos.

4.- Crear una aplicación de usuarios. El componente principal cargará en un componente ListadoUsuarios, el listado de usuarios que devuelve la api: https://jsonplaceholder.typicode.com/users. Mostrar el nombre, apellido, correo , teléfono y el nombre de la calle.