Evitando el callback hell 15/23

RECURSOS MARCADORES

Uno de los principales problemas de los observables es el **Callback Hell**. La anidación de N cantidad de suscripciones, una dentro de la otra, vuelve tu código muy difícil de mantener y de leer.

Cómo solucionar el Infierno de Callbacks

Utilizando promesas, puedes resolver este problema fácilmente con `async/await`. Pero si hablamos de observables, nuestra mejor amiga, la librería **RxJS**, llega para aportar su solución.

Un ejemplo común de esta problemática en Angular es como la siguiente.

```
readAndUpdate(): void {
  // Ejemplo de callback hell
  this.apiService.getProduct(1)
    .subscribe(res => {
      this.apiService.updateProduct(1, { name: 'Nuevo nombre del producto' })
        .subscribe(res2 => {
          // ...
        });
```

```
    });
}
```

Donde se está realizando una petición para la lectura de un producto e inmediatamente se está actualizando el mismo. Generando un `subscribe` dentro de otro.

¿Cómo resolver este problema de Callback Hell?

Tal vez hasta dos subscribe es aceptable, pero no se recomienda continuar con esa estructura de código y es posible resolverlo de la siguiente manera.

```
import { switchMap } from 'rxjs/operators';
readAndUpdate(): void {
  // Solución callback hell
  this.apiService.getProduct(1)
    .pipe(
      switchMap(products => this.apiService.updateProduct(1, { name: 'Nuevo nombre' }) )
    )
    .subscribe(res => {
      // Producto actualizado
    });
}
```

Importando `switchMap` desde `rxjs/operators` , lo que hace esta función es recibir el dato que emite un observable, y utilizarlo como input para el segundo. De esta manera, el código queda más limpio y profesional.

Conoce más sobre RxJS

Otra alternativa que brinda **RxJS** es la posibilidad de manipular varios observables al mismo tiempo. Con las promesas, puedes hacer uso de `Promise.all([])` para realizar N procesamientos asincrónicos en paralelo y obtener sus resultados.
De forma muy similar, en **RxJS** puedes hacer lo siguiente.

```
import { zip } from 'rxjs';
readAndUpdate(): void {
  // Agrupando observables en un mismo subscribe
  zip(
    this.apiService.getProduct(1),
    this.apiService.updateProductPATCH(1, { name: 'Nuevo nombre' })
  )
  .subscribe(res => {
    const get = res[0];
    const update = res[1];
  });
}
```

Importando la función `zip`, puedes crear un *array* de observables y suscribirte a todos ellos al mismo tiempo. Diferenciando por el índice de cada uno el resultado de la operación que ejecutaron para obtenerlo.

En este ejemplo, el índice 0 posee el resultado de la lectura y el índice 1 el del update.

Contribución creada por: *Kevin Fiorentino.*

Archivos de la clase