## Manejo de errores 13/23

RECURSOS MARCADORES

Las **peticiones HTTP** que tu aplicación realiza pueden fallar por una u otra razón. Recuerda que una aplicación profesional tiene que contemplar estos escenarios y estar preparada para cuando esto suceda.

### Manejo de Errores con RxJS

Ya hemos visto anteriormente la función `retry()` de **RxJS** para reintentar una petición en caso de fallo. Pero si el error persiste, veamos cómo es posible manejar los mismos.

#### 1. Imports desde RxJS

Presta atención a las siguientes importaciones desde **RxJS** y `@angular/common/http` que utilizaremos.

```
// service/api.service.ts
import { HttpErrorResponse, HttpStatusCode } from '@angular/common/http';
import { throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

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

```
    // ...
}
```

### 2. Manejo de errores con catchError()

Agrega a los métodos que realizan las solicitudes HTTP un `.pipe()` para manipular los datos que el observable emita antes de enviarlos al componente.

Aquí estamos utilizando `catchError` de **RxJS** para capturar los errores y le pasamos un método personalizado llamado `handleErrors()` para centralizar el manejo de los mismos (Así no repetimos el mismo código en todos los pipes).

```
// service/api.service.ts
getProduct(idProduct: number): Observable<Product> {
  return this.http.get<Product>(`https://example.com/api/productos/${idProduct}`)
    .pipe(
      catchError((err: HttpErrorResponse) => {
        return this.handleErrors(err)
      })
    );
}

handleErrors(error: HttpErrorResponse): Observable<never>  {
  if (error.status == HttpStatusCode.Forbidden)
    return throwError('No tiene permisos para realizar la solicitud.');
  if (error.status == HttpStatusCode.NotFound)
    return throwError('El producto no existe.');
  if (error.status == HttpStatusCode.InternalServerError)
    return throwError('Error en el servidor.');
```

```
    return throwError('Un error inesperado ha ocurrido.');
  }
```

### 3. Identifica el tipo de error

Dicho método recibe un parámetro del tipo `HttpErrorResponse` que Angular nos provee para tipar errores HTTP. Utilizando el enumerado `HttpStatusCode` puedes determinar qué tipo de error se produjo. Si fue un error 403, 404, 500, etc. Fácilmente, puedes verificarlo y devolver un error al componente con `throwError` y un mensaje personalizado.

> *NOTA: Más allá de este enum que Angular nos facilita, es muy importante que como desarrollador de software sepas diferenciar lo que significa un 401, 403, 404, 500, 501, entre otros estados HTTP. Poco a poco lo irás aprendiendo.*

### 4. Captura los errores en los componentes

En tu componente, captura los errores en las suscripciones de la siguiente manera:

```
// components/catalogo/catalogo.component.ts
this.apiService.getProducts()
  .subscribe(res => {
    this.productos = res;
  }, err => {
    alert(err);     // Aquí se emitirá el alerta con el mensaje que `throwError`
devuelva.
  });
```

Así, ya puedes diferenciar los errores y comunicarle al usuario si algo salió mal de la manera más apropiada, mostrándole un mensaje o una alerta.

---

***Contribución creada por:*** *Kevin Fiorentino.*

## Archivos de la clase

/

- tsconfig.app.json
- package.json
- angular.json
- .eslintrc.json
- package-lock.json
- .browserslistrc
- .gitignore
- obs-promises.js
- tsconfig.spec.json
- proxy.config.json

tsconfig.json

README.md

.editorconfig

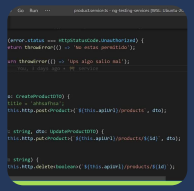
karma.conf.js

src

## Lecturas recomendadas

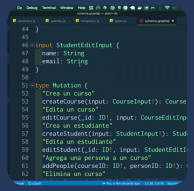
Angular
https://angular.io/api/common/http/HttpStatusCode

## Clases relacionadas



Pruebas a errores

Curso de Angular: Unit Testing para Servicios



Errores

Curso Práctico de GraphQL con JavaScript