Handling Exceptions Using the Angular HttpClient Service

## **Introduction**

HTTP requests are extremely error-prone because there are generally a lot of unknowns surrounding them. Is the server you are trying to reach running properly? Is the API returning the payload you are expecting? In this guide, you will learn what to do when these questions are not answered in the appropriate way. We will discuss some of the best ways to manage HTTP exceptions within your app while using Angular's HttpClient service.

## **Getting Started Handling HTTP Exceptions**

When talking about handling HTTP exceptions in Angular, it is nearly impossible not to talk about RxJs. The Angular framework is heavily bought into the RxJs library—a library that brings reactive programming into the JavaScript ecosystem. Angular's own HttpClient makes use of RxJs under the hood to ensure that the client has an observable API.

There are a few RxJs operators which you will need to be familiar with in order to handle exceptions properly. The primary operators we will concern ourselves with in this guide are the catchError and throwError operators.

## **Catching HTTP Exceptions**

Using the catchError operator gives you the ability to catch errors that may occur within an observable stream. Let's check out how we can use this operator in an example app.

Here is an example service for fetching books from the server. It looks like this:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

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

import { Injectable } from '@angular/core';

import { Book } from 'app/models/book';

@Injectable()

export class BooksService {

private booksApiUrl = '/api/books';

constructor(private http: HttpClient) { }

getBooks(): Observable<Book[]> {

return this.http.get(this.booksApiUrl);

}

}

typescript

And here is an example component that uses BooksService:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

import { Component } from '@angular/core';

import { BooksService } from 'app/services/books';

import { Book } from 'app/models/book';

@Component({

selector: 'app-books',

template: `

<ul \*ngFor="let book of books$">

<li>{{ book.title }}</li>

</ul>

`

})

export class BooksComponent {

books$: Observable<Book[]>;

constructor(private booksService: BooksService) {

this.books$ = this.booksService.getBooks();

}

}

typescript

Above, the BooksComponent communicates with BooksService in order to fetch a payload of books from the server. What if something goes wrong during this request? Of course, you will need to handle this exception in some way.

Next, add the catchError operator to BooksComponent to handle exceptions:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

import { Component } from '@angular/core';

import { BooksService } from 'app/services/books';

import { Book } from 'app/models/book';

import { catchError } from 'rxjs/operators';

import { of } from 'rxjs';

@Component({

selector: 'app-books',

template: `

<error-alert \*ngIf="errorMsg" [msg]="errorMsg"></error-alert>

<ul \*ngFor="let book of books$">

<li>{{ book.title }}</li>

</ul>

`

})

export class BooksComponent {

books$: Observable<Book[]>;

errorMsg: string;

constructor(private booksService: BooksService) {

this.books$ =

this.booksService

.getBooks()

.pipe(

catchError(error => {

if (error.error instanceof ErrorEvent) {

this.errorMsg = `Error: ${error.error.message}`;

} else {

this.errorMsg = `Error: ${error.message}`;

}

return of([]);

})

);

}

}

typescript

You have altered BooksComponent to catch any exceptions that may occur from the HTTP request! You are also using the of observable helper function to ensure that when an error occurs, the observable returns an empty array. Now, when an error occurs, the user can easily see the message.

But what if you want a little more custom control over the error message that is shown? In the next section, you will learn how you can use the throwError operator to do just that.

## **Re-throwing HTTP Exceptions**

Odds are, you don't want to show the user the exact error message that you get from Angular HttpClient when an exception occurs. A great way to customize this message is by throwing your own custom exceptions using throwError. First, update your BooksService to throw a custom error depending on the final status of the HTTP request.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

import { HttpClient, HttpErrorResponse } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Book } from 'app/models/book';

import { catchError, throwError } from 'rxjs/operators';

@Injectable()

export class BooksService {

private booksApiUrl = '/api/books';

constructor(private http: HttpClient) { }

getBooks(): Observable<Book[]> {

return this.http

.get(this.booksApiUrl)

.pipe(

catchError(error => {

let errorMsg: string;

if (error.error instanceof ErrorEvent) {

this.errorMsg = `Error: ${error.error.message}`;

} else {

this.errorMsg = this.getServerErrorMessage(error);

}

return throwError(errorMsg);

})

);

}

private getServerErrorMessage(error: HttpErrorResponse): string {

switch (error.status) {

case 404: {

return `Not Found: ${error.message}`;

}

case 403: {

return `Access Denied: ${error.message}`;

}

case 500: {

return `Internal Server Error: ${error.message}`;

}

default: {

return `Unknown Server Error: ${error.message}`;

}

}

}

}

typescript

The above code moved some of the error-handling logic from BooksComponent to the service. Through the use of both the throwError operator and the new getServerErrorMessage method, you are now throwing a custom error from service for HTTP errors.

Now, simplify your BooksComponent.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

import { Component } from '@angular/core';

import { BooksService } from 'app/services/books';

import { Book } from 'app/models/book';

import { catchError } from 'rxjs/operators';

import { of } from 'rxjs';

@Component({

selector: 'app-books',

template: `

<error-alert \*ngIf="errorMsg" [msg]="errorMsg"></error-alert>

<ul \*ngFor="let book of books$">

<li>{{ book.title }}</li>

</ul>

`

})

export class BooksComponent {

books$: Observable<Book[]>;

errorMsg: string;

constructor(private booksService: BooksService) {

this.books$ =

this.booksService

.getBooks()

.pipe(

catchError(error => {

this.errorMsg = error.message;

return of([]);

})

);

}

}

typescript

It is a best practice to keep components as simple as possible. In this case, you have been able to appropriately move the logic of your error handling from BooksComponent to BooksService.

## **Conclusion**

In this guide, you learned how to manage HTTP exceptions within your app. You have seen how the RxJs catchError operator can be used to catch any HTTP exceptions that occur while using the Angular HttpClientservice. You have also seen how to throw your own errors intentionally or re-throw caught errors using the RxJs throwError operator. By using these two techniques, you are well equipped to handle unexpected errors within any HTTP-driven application.