Catching errors in Angular HttpClient

Asked 2 years, 1 month ago Active 2 months ago Viewed 153k times

78

34

I have a data service that looks like this:

```
@Injectable()
export class DataService {
    baseUrl = 'http://localhost'
        constructor(
        private httpClient: HttpClient) {
    }
    get(url, params): Promise<Object> {

        return this.sendRequest(this.baseUrl + url, 'get', null, params)
            .map((res) => {
                return res as Object
            })
            .toPromise();
    }
    post(url, body): Promise<Object> {
        return this.sendRequest(this.baseUrl + url, 'post', body)
            .map((res) => {
                return res as Object
            })
            .toPromise();
    }
    patch(url, body): Promise<Object> {
        return this.sendRequest(this.baseUrl + url, 'patch', body)
            .map((res) => {
                return res as Object
            })
            .toPromise();
    }
    sendRequest(url, type, body, params = null): Observable<any> {
        return this.httpClient[type](url, { params: params }, body)
    }
}
```

If I get an HTTP error (i.e. 404), I get a nasty console message: **ERROR Error: Uncaught (in promise): [object Object]** from angular angular-httpclient

edited Jul 10 '18 at 19:37

asked Sep 3 '17 at 2:26
LastTribunal
2,119 6 27 56

11 Answers

163

You have some options, depending on your needs. If you want to handle errors on a per-request basis, add a `catch` to your request. If you want to add a global solution, use `HttpInterceptor` .

Open **here the working demo plunker** for the solutions below.

tl;dr

In the simplest case, you'll just need to add a `.catch()` or a `.subscribe()` , like:

```
import 'rxjs/add/operator/catch'; // don't forget this, or you'll get a runtime error
this.httpClient
      .get("data-url")
      .catch((err: HttpErrorResponse) => {
        // simple logging, but you can do a lot more, see below
        console.error('An error occurred:', err.error);
      });

// or
this.httpClient
      .get("data-url")
      .subscribe(
        data => console.log('success', data),
        error => console.log('oops', error)
      );
```

But there are more details to this, see below.

If you need to handle errors in only one place, you can use `catch` and return a default value (or empty response) instead of failing completely. You also don't need the `.map` just to cast, you can use a generic function. Source: [Angular.io - Getting Error Details](#).

So, a generic `.get()` method, would be like:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from "@angular/common/http";
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/of';
import 'rxjs/add/observable/empty';
import 'rxjs/add/operator/retry'; // don't forget the imports

@Injectable()
export class DataService {
    baseUrl = 'http://localhost';
    constructor(private httpClient: HttpClient) { }

    // notice the <T>, making the method generic
    get<T>(url, params): Observable<T> {
      return this.httpClient
          .get<T>(this.baseUrl + url, {params})
          .retry(3) // optionally add the retry
          .catch((err: HttpErrorResponse) => {

            if (err.error instanceof Error) {
              // A client-side or network error occurred. Handle it accordingly.
              console.error('An error occurred:', err.error.message);
            } else {
              // The backend returned an unsuccessful response code.
              // The response body may contain clues as to what went wrong,
              console.error(`Backend returned code ${err.status}, body was:
${err.error}`);
            }

            // ...optionally return a default fallback value so app can continue (pick
one)
            // which could be a default value
            // return Observable.of<any>({my: "default value..."});
            // or simply an empty observable
            return Observable.empty<T>();
          });
     }
}
```

This per-request solution is good mostly when you want to return a specific default response to each method. But if you only care about error displaying (or have a global default response), the better solution is to use an interceptor, as described below.

Run the **working demo plunker here**.

Advanced usage: Intercepting all requests or responses

Once again, Angular.io guide shows:

> A major feature of `@angular/common/http` is interception, the ability to declare interceptors which sit in between your application and the backend. When your application makes a request, interceptors transform it before sending it to the server, and the interceptors can transform the response on its way back before your application sees it. This is useful for everything from authentication to logging.

Which, of course, can be used to handle errors in a very simple way (**demo plunker here**):

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest, HttpResponse,
         HttpErrorResponse } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/of';
import 'rxjs/add/observable/empty';
import 'rxjs/add/operator/retry'; // don't forget the imports

@Injectable()
export class HttpErrorInterceptor implements HttpInterceptor {
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(request)
      .catch((err: HttpErrorResponse) => {

        if (err.error instanceof Error) {
          // A client-side or network error occurred. Handle it accordingly.
          console.error('An error occurred:', err.error.message);
        } else {
          // The backend returned an unsuccessful response code.
          // The response body may contain clues as to what went wrong,
          console.error(`Backend returned code ${err.status}, body was: ${err.error}`);
        // which could be a default value (which has to be a HttpResponse here)
        // return Observable.of(new HttpResponse({body: [{name: "Default value..."}]}));
        // or simply an empty observable
        return Observable.empty<HttpEvent<any>>();
      });
  }
}
```

Providing your interceptor: Simply declaring the `HttpErrorInterceptor` above doesn't cause your app to use it. You need to wire it up in your app module by providing it as an interceptor, as follows:

```
import { NgModule } from '@angular/core';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { HttpErrorInterceptor } from './path/http-error.interceptor';

@NgModule({
  ...
  providers: [{
    provide: HTTP_INTERCEPTORS,
    useClass: HttpErrorInterceptor,
    multi: true,
  }],
  ...
})
export class AppModule {}
```

Note: If you have *both* an error interceptor and some local error handling, naturally, it is likely that no local error handling will ever be triggered, since the error will always be handled by the interceptor *before* it reaches the local error handling.

Run the **working demo plunker here**.

edited Dec 30 '17 at 19:38 answered Sep 3 '17 at 2:50

acdcjunior
92.1k 23 220 214

2 well, if he wants to be fully fancy he would leave his service fully clear: `return this.httpClient.get<type>(...)` . and then have `catch...` somewhere out of the service where he actually consumes it because thats where he will be building observables flow and can handle it best. – dee zg Sep 3 '17 at 3:17

1 I agree, maybe an optimal solution would be to have the `Promise<Object>` 's client (the caller of the `DataService` 's methods) to handle the error.

1 @YakovFain If you want a default value in the interceptor, it must be a `HttpEvent` , such as a `HttpResponse` . So, for instance, you could use: `return Observable.of(new HttpResponse({body: [{name: "Default value..."}]}));` . I have updated the answer to make this point clear. Also, I created a working demo plunker to show everything working: plnkr.co/edit/ulFGp4VMzrbaDJeGqc6q?p=preview – acdcjunior Dec 30 '17 at 18:24

1 @acdcjunior Thank you! This works. – Yakov Fain Dec 30 '17 at 20:18

1 @acdcjunior, you are a gift that keeps on giving :) – LastTribunal Apr 7 '18 at 1:11

45

With the arrival of the `HTTPClient` API, not only was the `Http` API replaced, but a new one was added, the `HttpInterceptor` API.

AFAIK one of its goals is to add default behavior to all the HTTP outgoing requests and incoming responses.

So assumming that you want to add a **default error handling behavior**, adding `.catch()` to all of your possible http.get/post/etc methods is ridiculously hard to maintain.

This could be done in the following way as example using a `HttpInterceptor` :

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest, HttpErrorResponse,
HTTP_INTERCEPTORS } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { _throw } from 'rxjs/observable/throw';
import 'rxjs/add/operator/catch';

/**
 * Intercepts the HTTP responses, and in case that an error/exception is thrown, handles
it
 * and extract the relevant information of it.
 */
@Injectable()
export class ErrorInterceptor implements HttpInterceptor {
    /**
     * Intercepts an outgoing HTTP request, executes it and handles any error that could
be triggered in execution.
     * @see HttpInterceptor
     * @param req the outgoing HTTP request
     * @param next a HTTP request handler
     */
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(req)
            .catch(errorResponse => {
```

```
JSON.stringify(errorResponse.error);
                    errMsg = `${errorResponse.status} - ${errorResponse.statusText ||
''} Details: ${err}`;
                } else {
                    errMsg = errorResponse.message ? errorResponse.message :
errorResponse.toString();
                }
                return _throw(errMsg);
            });
    }
}

/**
 * Provider POJO for the interceptor
 */
export const ErrorInterceptorProvider = {
    provide: HTTP_INTERCEPTORS,
    useClass: ErrorInterceptor,
    multi: true,
};
```

// app.module.ts

```
import { ErrorInterceptorProvider } from 'somewhere/in/your/src/folder';

@NgModule({
   ...
   providers: [
    ...
    ErrorInterceptorProvider,
    ....
   ],
   ...
})
export class AppModule {}
```

Some extra info for OP: Calling http.get/post/etc without a strong type isn't an optimal use of the API. Your service should look like this:

```
// These interfaces could be somewhere else in your src folder, not necessarily in your
service file
export interface FooPost {
 // Define the form of the object in JSON format that your
 // Define the form of the object in JSON format that your
 // expect from the backend on patch
}

export interface FooGet {
 // Define the form of the object in JSON format that your
 // expect from the backend on get
}

@Injectable()
export class DataService {
    baseUrl = 'http://localhost'
    constructor(
        private http: HttpClient) {
    }

    get(url, params): Observable<FooGet> {

        return this.http.get<FooGet>(this.baseUrl + url, params);
    }

    post(url, body): Observable<FooPost> {
        return this.http.post<FooPost>(this.baseUrl + url, body);
    }

    patch(url, body): Observable<FooPatch> {
        return this.http.patch<FooPatch>(this.baseUrl + url, body);
    }
}
```

Returning `Promises` from your service methods instead of `Observables` is another bad decision.

And an extra piece of advice: if you are using **TYPE**script, then start using the type part of it. You lose one of the biggest advantages of the language: to know the type of the value that you are dealing with.

If you want a, in my opinion, good example of an angular service, take a look at the following gist.

edited Feb 15 '18 at 20:51
BSMP
2,903 5 27 36

answered Sep 3 '17 at 9:01
Jota.Toledo
15.5k 7 38 56

Comments are not for extended discussion; this conversation has been moved to chat. – deceze ♦ Sep 4 '17 at 12:34

The selected answer appears to now be more complete. – Chris Haines Dec 12 '17 at 12:55

2 The gist link does not work anymore ... – Jette Jun 27 '18 at 8:31

34

Let me please update the acdcjunior's answer about using HttpInterceptor with the latest RxJs features(v.6).

```
import { Injectable } from '@angular/core';
import {
  HttpInterceptor,
  HttpRequest,
  HttpErrorResponse,
  HttpHandler,
  HttpEvent,
  HttpResponse
} from '@angular/common/http';

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

@Injectable()
export class HttpErrorInterceptor implements HttpInterceptor {
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    return next.handle(request).pipe(
      catchError((error: HttpErrorResponse) => {
        if (error.error instanceof Error) {
          // A client-side or network error occurred. Handle it accordingly.
          console.error('An error occurred:', error.error.message);
        } else {
          // The backend returned an unsuccessful response code.
          // The response body may contain clues as to what went wrong,
          console.error(`Backend returned code ${error.status}, body was:
${error.error}`);
        }

        // If you want to return a new response:
        //return of(new HttpResponse({body: [{name: "Default value..."}]}));

        // If you want to return the error on the upper level:
        //return throwError(error);

        // or just return nothing:
    }
  }
```

answered Nov 19 '18 at 17:20

MegaCasper
494 4 6

6 This needs to get upvoted more. acdcjunior's answer is unusable as of today – Paul Kruger Mar 22 at 9:20

4

For Angular 6+ , .catch doesn't work directly with Observable. You have to use

```
.pipe(catchError(this.errorHandler))
```

Below code:

```
import { IEmployee } from './interfaces/employee';
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

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

  private url = '/assets/data/employee.json';

  constructor(private http: HttpClient) { }

  getEmployees(): Observable<IEmployee[]> {
    return this.http.get<IEmployee[]>(this.url)
                    .pipe(catchError(this.errorHandler));  // catch error
  }

  /** Error Handling method */
```

```
        } else {
          // The backend returned an unsuccessful response code.
          // The response body may contain clues as to what went wrong,
          console.error(
            `Backend returned code ${error.status}, ` +
            `body was: ${error.error}`);
        }
        // return an observable with a user-facing error message
        return throwError(
          'Something bad happened; please try again later.');
      }
    }
```

For more details, refer to the Angular Guide for Http

edited Jul 31 at 5:49
Paul Rooney
13.8k 7 31 47

answered May 3 at 16:58
Udith Indrakantha
118 6

2

You probably want to have something like this:

```
this.sendRequest(...)
.map(...)
.catch((err) => {
//handle your error here
})
```

It highly depends also how do you use your service but this is the basic case.

answered Sep 3 '17 at 2:51
dee zg
5,913 5 20 42

Fairly straightforward (in compared to how it was done with the previous API).

Source from (copy and pasted) the Angular official guide

```
http
  .get<ItemsResponse>('/api/items')
  .subscribe(
    // Successful responses call the first callback.
    data => {...},
    // Errors will call this callback instead:
    err => {
      console.log('Something went wrong!');
    }
  );
```

answered Nov 15 '17 at 23:01

Tomer
847 6 12

1

Following @acdcjunior answer, this is how I implemented it

service:

```
  get(url, params): Promise<Object> {

            return this.sendRequest(this.baseUrl + url, 'get', null, params)
                .map((res) => {
                    return res as Object
                }).catch((e) => {
                    return Observable.of(e);
                })
                .toPromise();
        }
```

caller:

```
this.dataService.get(baseUrl, params)
            .then((object) => {
                if(object['name'] === 'HttpErrorResponse') {
                            this.error = true;
                           //or any handle
                } else {
```

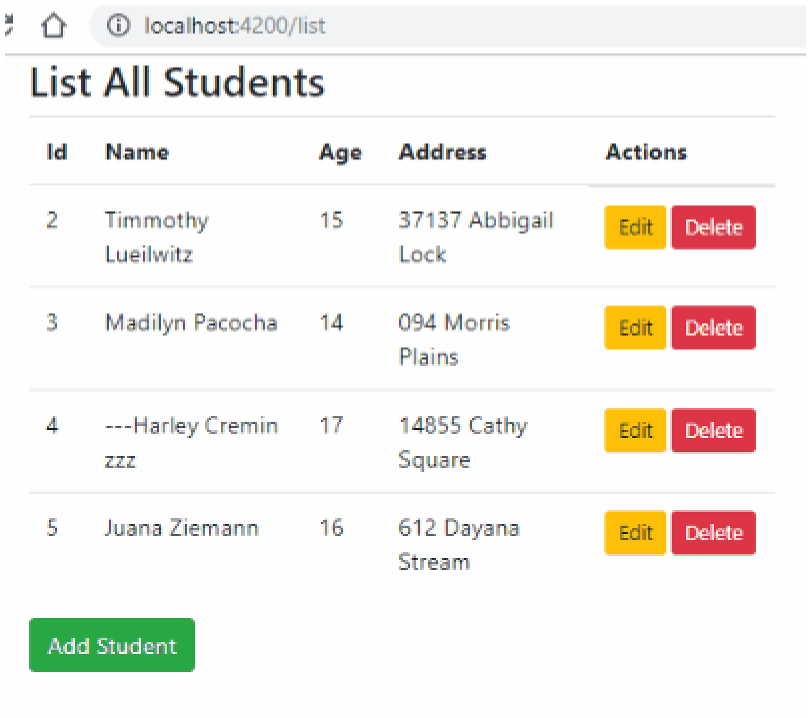
answered Sep 3 '17 at 4:02

LastTribunal

2,119 6 27 56

1

Angular 8 HttpClient Error Handling Service Example



List All Students

Id	Name	Age	Address	Actions
2	Timmothy Lueilwitz	15	37137 Abbigail Lock	Edit Delete
3	Madilyn Pacocha	14	094 Morris Plains	Edit Delete
4	---Harley Cremin zzz	17	14855 Cathy Square	Edit Delete
5	Juana Ziemann	16	612 Dayana Stream	Edit Delete

api.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpErrorResponse } from '@angular/common/http';
import { Student } from '../model/student';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';

@Injectable({
  // API path
  base_path = 'http://localhost:3000/students';

  constructor(private http: HttpClient) { }

  // Http Options
  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  }

  // Handle API errors
  handleError(error: HttpErrorResponse) {
    if (error.error instanceof ErrorEvent) {
      // A client-side or network error occurred. Handle it accordingly.
      console.error('An error occurred:', error.error.message);
    } else {
      // The backend returned an unsuccessful response code.
      // The response body may contain clues as to what went wrong,
      console.error(
        `Backend returned code ${error.status}, ` +
        `body was: ${error.error}`);
    }
    // return an observable with a user-facing error message
    return throwError(
      'Something bad happened; please try again later.');
  };


  // Create a new item
  createItem(item): Observable<Student> {
    return this.http
      .post<Student>(this.base_path, JSON.stringify(item), this.httpOptions)
      .pipe(
        retry(2),
        catchError(this.handleError)
      )
  }

 ........
 ........

}
```

Code Spy
4,372 2 25 27

0

If you find yourself unable to catch errors with any of the solutions provided here, it may be that the server isn't handling CORS requests.

In that event, Javascript, much less Angular, can access the error information.

Look for warnings in your console that include `CORB` or `Cross-Origin Read Blocking`.

Also, the syntax has changed for handling errors (as described in every other answer). You now use pipe-able operators, like so:

```
this.service.requestsMyInfo(payload).pipe(
    catcheError(err => {
        // handle the error here.
    })
);
```

answered Aug 20 '18 at 14:57

Kevin Beal
4,595 9 39 58

0

By using Interceptor you can catch error. Below is code:

```
@Injectable()
export class ResponseInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    //Get Auth Token from Service which we want to pass thr service call
    const authToken: any = `Bearer ${sessionStorage.getItem('jwtToken')}`
    // Clone the service request and alter original headers with auth token.
    const authReq = req.clone({
      headers: req.headers.set('Content-Type', 'application/json').set('Authorization', 
authToken)
    });

    const authReq = req.clone({ setHeaders: { 'Authorization': authToken, 'Content-
Type': 'application/json'} });
```

```
        if (event instanceof HttpResponse) {
          console.log("Service Response thr Interceptor");
        }
      }, (err: any) => {
        if (err instanceof HttpErrorResponse) {
          console.log("err.status", err);
          if (err.status === 401 || err.status === 403) {
            location.href = '/login';
            console.log("Unauthorized Request - In case of Auth Token Expired");
          }
        }
      });
    }
  }
```

You can prefer this blog..given simple example for it.

edited Jul 31 at 5:51
Paul Rooney
13.8k 7 31 47

answered Mar 12 at 14:02
Prashant M Bhavsar
984 6 12

0

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

const PASSENGER_API = 'api/passengers';

getPassengers(): Observable<Passenger[]> {
  return this.http
    .get<Passenger[]>(PASSENGER_API)
    .pipe(catchError((error: HttpErrorResponse) => throwError(error)));
}
```

edited Jul 31 at 5:51
Paul Rooney
13.8k 7 31 47

answered Mar 12 at 6:51
sun sreng
11 3