Angular HTTP Error Handling

10 Comments / 9 minutes of reading / March 9, 2023

← Error Handling | Angular Tutorial | Angular CLI Tutorial →

In this guide, we learn about Angular HTTP Error Handling. Whenever the error occurs in an HTTP operation, the Angular wraps it in an `httpErrorResponse` Object before throwing it back. We catch the `httpErrorResponse` either in our component class or in the data service class or globally. The Global HTTP error handling is done using the Angular HTTP Interceptor.

Suggested Reading: Error Handling in Angular

Table of Contents

HttpErrorResponse

The `HttpClient` captures the errors and wraps it in the generic `HttpErrorResponse`, before passing it to our app. The `error` property of the `HttpErrorResponse` contains the underlying `error` object. It also provides additional context about the state of the HTTP layer when the error occurred.

The HTTP errors fall into two categories. The back end server may generate the error and send the error response. Or the client-side code may fail to generate the request and throw the error (`ErrorEvent` objects).

The server might reject the request for various reasons. Whenever it does it will return the `error response` with the HTTP Status Codes such as `Unauthorized` (401), `Forbidden` (403), `Not found` (404), `internal Server Error` (500), etc. The Angular assigns the `error response` to `error` property of the `HttpErrorResponse`.

The client-side code can also generate the error. The error may be due to a network error or an error while executing the HTTP request or an exception thrown in an RxJS operator. These errors produce JavaScript `ErrorEvent` objects. The Angular assigns the `ErrorEvent` object to `error` property of the `HttpErrorResponse`.

In both the cases, the generic `HttpErrorResponse` is returned by the HTTP Module. We will inspect the error property to find out the type of Error and handle accordingly.

Catching Errors in HTTP Request

We can catch the HTTP Errors at three different places.

1. Component
2. Service
3. Globally

Catch Errors in Component

Refer to our tutorial on Angular HTTP Get Request. We created a `GitHubService`, where we made a `GET` request to the GitHub API to get the list of Repositories. The following is the `getRepos()` method from the service. We have intentionally changed the URL (`uersY`) so that it will result in an error.

```
1
2 getRepos(userName: string): Observable<any> {
3   return this.http.get(this.baseURL + 'usersY/' + userName + '/repos')
4 }
5
```

We `subscribe` to the `httpClient.get` method in the component class

```
1
2  public getRepos() {
3    this.loading = true;
4    this.errorMessage = "";
5    this.githubService.getReposCatchError(this.userName)
6      .subscribe(
7        (response) => {                    //Next callback
8          console.log('response received')
9          this.repos = response;
10       },
11       (error) => {                       //Error callback
12         console.error('error caught in component')
13         this.errorMessage = error;
14         this.loading = false;
```

```
15
16          //throw error;   //You can also throw the error to a global error handler
17        }
18      )
19    }
20
```

The `subscribe` method has three callback arguments.

```
1
2 .subscribe(success, error, completed);
3
```

The observable invokes the first callback `success`, when the HTTP request successfully returns a response. The third call back `completed` is called when the observable finishes without any error.

The second callback `error`, is invoked when the HTTP Request end in an error. We handle error here by figuring out the type of error and handle it accordingly. It gets the `error` object which is of type `HttpErrorResponse`.

```
1
2       (error) => {                    //Error callback
3         console.error('error caught in component')
4         this.errorMessage = error;
5         this.loading = false;
6       }
7
```

Catch Errors in Service

We can also catch errors in the service, which makes the HTTP Request using the `catchError` Operator as shown below. Once you handle the error, you can re-throw it back to the component for further handling.

```
getRepos(userName: string): Observable<repos[]> {
  return this.http.get<repos[]>(this.baseURL + 'usersY/' + userName + '/repos')
    .pipe(
      catchError((err) => {
        console.log('error caught in service')
        console.error(err);

        //Handle the error here

        return throwError(err);    //Rethrow it back to component
      })
    )
}
```

Catch error globally using HTTP Interceptor

The type of error we may encounter vary. But some of those errors are common to every HTTP request. For Example

1. You are unauthorized to access the API Service,
2. You are authorized, but forbidden to access a particular resource
3. The API End Point is invalid or does not exist
4. Network error
5. Server down

We can check all these errors in the service or in component, but our app may contain many such service or components. Checking for common errors in each and every method is inefficient and error-prone.

The Right thing to do is to handle only the errors specific to this API call in this component/service and move all the common errors to one single place. This is where we use the HTTP Interceptor.

The HTTP Interceptor is a service, which we create and register it globally at the root module using the Angular Providers. Once defined, it will intercept all the HTTP requests passing through the app. It intercepts when we make the HTTP request and also intercepts when the response arrives. This makes it an ideal place to catch all the common errors and handle it

We create the Interceptor by creating a Global Service class, which implements the `HttpInterceptor` Interface. Then we will override the `intercept` method in that service.

The following code shows a simple `GlobalHttpInterceptorService`

```
1
2  import {Injectable} from "@angular/core";
3  import {HttpEvent, HttpHandler, HttpInterceptor,HttpRequest,HttpResponse,HttpErrorRespo
4  import {Observable, of, throwError} from "rxjs";
5  import {catchError, map} from 'rxjs/operators';
6
7  @Injectable()
8  export class GlobalHttpInterceptorService implements HttpInterceptor {
9
10   constructor(public router: Router) {
11   }
12
13   intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
14
15     return next.handle(req).pipe(
16       catchError((error) => {
17         console.log('error is intercept')
18         console.error(error);
```

```
      return throwError(error.message);
    })
  )
 }
}

```

The caching of the Error is done using the `catchError` RxJS operator. We then re-throw it to the subscriber using the `throwError`

The `catchError` is added to the request pipeline using the RxJs `pipe` operator . When the error occurs in the HTTP Request it is intercepted and invokes the `catchError` . Inside the `catchError` you can handle the error and then use `throwError` to throw it to the service.

We then register the Interceptor in the Providers array of the root module using the injection token `HTTP_INTERCEPTORS` . Note that you can provide more than one Interceptor `(multi: true)` .

```

providers: [
   GitHubService,
   { provide: HTTP_INTERCEPTORS, useClass: GlobalHttpInterceptorService, multi: true  }
]

```

HTTP Error Handling

Next, step is what to do with the errors

The server-side errors return status codes, we can take appropriate actions based on that. For Example for Status code 401 Unauthorized , we can redirect the user to the login page, for 408 Request Timeout, we can retry the operation, etc.

The following example code shows how to check for status codes 401 & 403 and redirect to the login page.

```
1
2  if (error instanceof HttpErrorResponse) {
3      if (error.error instanceof ErrorEvent) {
4          console.error("Error Event");
5      } else {
6          console.log(`error status : ${error.status} ${error.statusText}`);
7          switch (error.status) {
8              case 401:      //login
9                  this.router.navigateByUrl("/login");
10                 break;
11             case 403:     //forbidden
12                 this.router.navigateByUrl("/unauthorized");
13                 break;
14         }
15     }
16 } else {
17     console.error("some thing else happened");
18 }
19 return throwError(error);
20
21
```

For Server errors with status codes *5XX*, you can simply ask the user to retry the operation. You can do this by showing an alert box or redirect him to a special page or show the error message at the top of the page bypassing the error message to a special service AlertService.

For other errors, you can simply re-throw it back to the service.

```
return throwError(error);
```

You can further handle the error in service or throw it back to the component.

The component must display the error message to the user. You can also throw it back to a global error handler in Angular.

```
.subscribe(
  (response) => {
    this.repos = response;
  },
  (error) => {
    //Handle the error here
    //If not handled, then throw it
    throw error;
  }
)
```

HTTP Error handling example

The complete code of this example

app.component.html

```
<h1 class="heading"><strong>Angular HTTP</strong>Error Example</h1>

<div class="form-group">
  <label for="userName">GitHub User Name</label>
  <input type="text" class="form-control" name="userName" [(ngModel)]="userName">
</div>

<div class="form-group">
  <button type="button" (click)="getRepos()">Get Repos</button>
</div>

<div *ngIf="loading">loading...</div>

<div *ngIf="errorMessage" class="alert alert-warning">
  <strong>Warning!</strong> {{errorMessage | json}}
</div>

<table class='table'>
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>HTML Url</th>
      <th>description</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let repo of repos;">
      <td>{{repo.id}}</td>
      <td>{{repo.name}}</td>
      <td>{{repo.html_url}}</td>
      <td>{{repo.description}}</td>
    </tr>
  </tbody>
</table> -

<pre>{{repos | json}}</pre>

```

app.component.ts

```
1
2  import { Component } from '@angular/core';
3
4  import { GitHubService } from './github.service';
5  import { repos } from './repos';
6
7  @Component({
8    selector: 'app-root',
9    templateUrl: './app.component.html',
10 })
11 export class AppComponent {
12   userName: string = "tektutorialshub"
13   repos: repos[];
14
15   loading: boolean = false;
16   errorMessage;
17
18   constructor(private githubService: GitHubService) {
19   }
20
21   public getRepos() {
22     this.loading = true;
23     this.errorMessage = "";
24     this.githubService.getReposCatchError(this.userName)
25       .subscribe(
26         (response) => {                    //Next callback
27           console.log('response received')
28           this.repos = response;
29         },
30         (error) => {                       //Error callback
31           console.error('error caught in component')
32           this.errorMessage = error;
33           this.loading = false;
34
35           throw error;
36         }
37       )
38   }
39 }
40
```

github.service.ts

```
1
2  import { Injectable } from '@angular/core';
3
```

```
import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { map, catchError } from 'rxjs/operators';

import { repos } from './repos';

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

 baseURL: string = "https://api.github.com/";

 constructor(private http: HttpClient) {
 }

 //Any Data Type
 getRepos(userName: string): Observable<any> {
  return this.http.get(this.baseURL + 'usersY/' + userName + '/repos')
 }


 //With catchError
 getReposCatchError(userName: string): Observable<repos[]> {
  return this.http.get<repos[]>(this.baseURL + 'usersY/' + userName + '/repos')
   .pipe(
    catchError((err) => {
     console.log('error caught in service')
     console.error(err);
     return throwError(err);
    })
   )
 }

}

```

global-http-Interceptor.service.ts

```

import { Injectable } from "@angular/core";
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest, HttpErrorResponse } from '
import { Observable, of, throwError } from "rxjs";
import { catchError, map } from 'rxjs/operators';
import { Router } from '@angular/router';

@Injectable()
export class GlobalHttpInterceptorService implements HttpInterceptor {
```

```
10
11  constructor(public router: Router) {
12  }
13
14  //1.  No Errors
15  intercept1(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
16
17    return next.handle(req).pipe(
18      catchError((error) => {
19        console.log('error in intercept')
20        console.error(error);
21        return throwError(error.message);
22      })
23    )
24  }
25
26  //2. Sending an Invalid Token will generate error
27  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
28
29    const token: string = 'invald token';
30    req = req.clone({ headers: req.headers.set('Authorization', 'Bearer ' + token) });
31
32    return next.handle(req).pipe(
33      catchError((error) => {
34        console.log('error in intercept')
35        console.error(error);
36        return throwError(error.message);
37      })
38    )
39  }
40
41  intercept3(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
42
43    const token: string = 'invald token';
44    req = req.clone({ headers: req.headers.set('Authorization', 'Bearer ' + token) });
45
46    return next.handle(req).pipe(
47      catchError((error) => {
48
49        let handled: boolean = false;
50        console.error(error);
51        if (error instanceof HttpErrorResponse) {
52          if (error.error instanceof ErrorEvent) {
53            console.error("Error Event");
54          } else {
55            console.log(`error status : ${error.status} ${error.statusText}`);
56            switch (error.status) {
57              case 401:      //login
58                this.router.navigateByUrl("/login");
```

```
            console.log(`redirect to login`);
            handled = true;
            break;
          case 403:     //forbidden
            this.router.navigateByUrl("/login");
            console.log(`redirect to login`);
            handled = true;
            break;
        }
      }
    }
    else {
      console.error("Other Errors");
    }

    if (handled) {
      console.log('return back ');
      return of(error);
    } else {
      console.log('throw error back to to the subscriber');
      return throwError(error);
    }

  })
 )
 }
}

```

global-error-handler.service.ts

```

import { ErrorHandler, Injectable, Injector } from '@angular/core';
import { HttpErrorResponse } from '@angular/common/http';
import { throwError } from 'rxjs';

@Injectable()
export class GlobalErrorHandlerService implements ErrorHandler {

 constructor() {
 }


 handleError(error: Error | HttpErrorResponse) {
   console.log('GlobalErrorHandlerService')
   console.error(error);
```

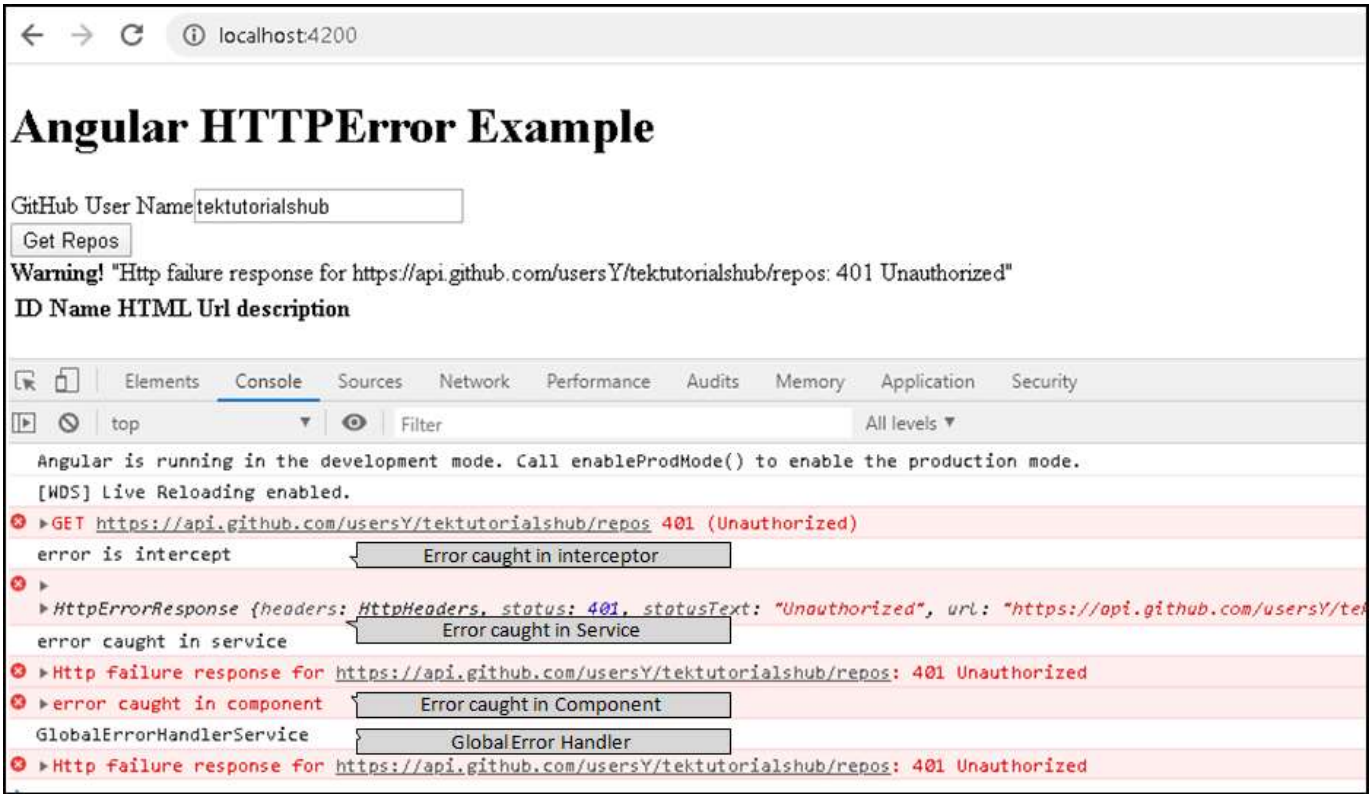
```
  }

}

```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule ,ErrorHandler } from '@angular/core';
import { HttpClientModule,HTTP_INTERCEPTORS} from '@angular/common/http';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { GlobalHttpInterceptorService} from './global-http-Interceptor.service';
import { AppRoutingModule } from './app-routing.module';
import { GlobalErrorHandlerService } from './global-error-handler.service';



@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    AppRoutingModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS,    useClass: GlobalHttpInterceptorService,    multi: tru
    { provide: ErrorHandler, useClass:GlobalErrorHandlerService}
],
  bootstrap: [AppComponent]
})
export class AppModule { }


```

app-routing.module.ts

```
1
2  import { NgModule } from '@angular/core';
3  import { Routes, RouterModule } from '@angular/router';
4
5
6  const routes: Routes = [];
7
8  @NgModule({
9    imports: [RouterModule.forRoot(routes)],
10   exports: [RouterModule]
11 })
12 export class AppRoutingModule { }
13
14
```



References

HttpErrorResponse

Summary

Using HTTP Interceptors you can catch HTTP Errors and handle it appropriately. Check the HTTP status codes and take appropriate actions like redirecting to the login page,