# Gestione degli errori

La gestione degli errori rilevati nella parte back-end deve poter prevedere due modalità :

* Errore standard
* Errore custom

Nella modalità standard l’errore che viene prodotto è sotto forma di html.

Nella modalità custom, lato bech-end, è stato realizzato un oggetto così fatto :

1. ApiException : Exception
2. La sua struttura è la seguente :
   * 1. HttpStatusCode StatusCode
     2. List<ErrorData> ErrorDataList

L’oggetto ErrorDataList è una list di ErrorData, così strutturata :

* Field
* ErrorCode
* ErrorDescription

Di seguito riporto un esempio di utilizzo di errore custom all’interno di un blocco catch:

List<ErrorData> errorDataList = new List<ErrorData>();

ErrorData errore1 = new ErrorData("field", "999", err.Message);

errorDataList.Add(errore1);

ErrorData errore2 = new ErrorData("field", "998", "Fine Messaggio");

errorDataList.Add(errore2);

throw new ApiException(HttpStatusCode.BadRequest, errorDataList);

Lato front-end gli errori verranno catturati da un interceptor dedicato a questa situazione.

Un ErrorInterceptor è una classe ts che implementa HttpInterceptor e che deve essere caricata nel modulo app.module.ts come indicato di seguito :

…

import { HttpClient, HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

import { ErrorInterceptor } from './http-interceptor/ErrorInterceptor';

export const httpInterceptorProviders = [

        { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },

  ];

Nei providers :

    providers: [

        httpInterceptorProviders,

L’interceptor invece è il seguente :

import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';

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

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

import { ErrorDetailsModel } from '../model/error/errorDetails';

import { ErrorDialogConfig } from '../pages/error-dialog/error-dialog.component';

import { ErrorDialogService } from '../pages/error-dialog/error-dialog.service';

import { ToastService } from '../service/toast/toast.service';

/\*\* Pass untouched request through to the next request handler. \*/

@Injectable()

export class ErrorInterceptor implements HttpInterceptor {

    public \_toastSVC : ToastService;

    public \_errDialogSVC: ErrorDialogService;

    constructor(private \_inj: Injector) {

            this.\_toastSVC = this.\_inj.get(ToastService);

            this.\_errDialogSVC = this.\_inj.get(ErrorDialogService);

    }

    showError: boolean;

    intercept( req: HttpRequest<any>, next: HttpHandler ):

        Observable<HttpEvent<any>> {

        console.debug( `Http Request, method: ${req.method} - url: ${req.url} - params: ${req.params}` );

       return next.handle( req ).pipe(

            catchError( error =>  this.handleError(error) )

        );

    }

    private handleError( error: any ) {

}

Nella funzione handleError posso scrivere la gestione degli errori.

## Tipologia degli errori

Non tutti gli errori devono avere la stessa evidenza perché posso essere di gravità diverse.

E’ possibile dividerli in due grandi gruppi :

* Notifiche di attenzione
* Notifiche di errore

I “Warning” non devono essere “invasivi” quindi ho deciso di utilizzare i pup-up di tipo “Toast” per la notifica.

Le notifiche “Toast” posso notificare i seguenti tipi di errore :

1. Error
2. Warning
3. Info
4. Success

Mentre gli “Error”, che rappresentano gli errori con maggiore gravità, utilizzo un dialog che elencherà le anomalie che ha registrato.