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

import {Response, RequestOptions } from '@angular/http';

import 'rxjs/add/observable/throw';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

import { Observable } from 'rxjs/Observable';

import { AppConstants, AppConstantService, ValidationMessageConstants } from '../constants/index';

import { AppConfig, ConfigService } from '../../../config/index';

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

import { LoggedError } from '../../models/logged-error';

export class HttpUtilities {

private validationMessages: ValidationMessageConstants;

private apiBaseUrl: string;

private loggedError: LoggedError;

public constructor(private http: HttpClient, private appConstantService: AppConstantService, private configService: ConfigService) {

let appConfig: AppConfig;

this.validationMessages = this.appConstantService.appConstants.validationMessages;

appConfig = this.configService.config;

this.apiBaseUrl = appConfig.apiBaseUrl;

}

public httpGet<T>(apiActionUrl: string, apiParams?: Array<[string, any]>): Observable<T> {

const apiUrl = this.generateCompleteUrl(apiActionUrl, apiParams);

if (apiUrl) {

return this.http.get<T>(apiUrl, {headers: this.getRequestOptions(), withCredentials: true})

.catch(this.handleError);

}

return Observable.throw(this.validationMessages.serverError);

}

public httpPost<T>(apiActionUrl: string, payLoadToPost: any, apiParams?: Array<[string, any]>): Observable<Response> {

const apiUrl = this.generateCompleteUrl(apiActionUrl, apiParams);

if (apiUrl) {

return this.http.post<T>(apiUrl, payLoadToPost, {headers: this.getRequestOptions(), withCredentials: true})

.catch(this.handleError);

}

return Observable.throw(this.validationMessages.serverError);

}

public httpPut<T>(apiActionUrl: string, payLoadToPut: any, apiParams?: Array<[string, any]>): Observable<T> {

const apiUrl = this.generateCompleteUrl(apiActionUrl, apiParams);

if (apiUrl) {

return this.http.put<T>(apiUrl, payLoadToPut, {headers: this.getRequestOptions(), withCredentials: true})

.catch(this.handleError);

}

return Observable.throw(this.validationMessages.serverError);

}

public httpDelete<T>(apiActionUrl: string, apiParams?: Array<[string, any]>): Observable<T> {

const apiUrl = this.generateCompleteUrl(apiActionUrl, apiParams);

if (apiUrl) {

return this.http.delete<T>(apiUrl, {headers: this.getRequestOptions(), withCredentials: true})

.catch(this.handleError);

}

return Observable.throw(this.validationMessages.serverError);

}

public generateCompleteUrl(apiActionUrl: string, apiParams?: [string, any][]) {

if (apiActionUrl) {

if (apiParams) {

apiParams.forEach(param => {

apiActionUrl = apiActionUrl.replace('{' + param[0] + '}', param[1]);

});

}

return this.apiBaseUrl + apiActionUrl;

}

return undefined;

}

private extractData(res: Response) {

let body;

if (res.text()) {

if (res.json() != null && res.json().data) {

body = res.json().data;

} else {

body = res.json();

}

}

return body || {};

}

public handleError(error: HttpErrorResponse) {

let errMsg = '';

console.log(errMsg);

if (error instanceof Error) {

const err = error.error;

this.loggedError = new LoggedError();

try {

this.loggedError.message = err.message;

this.loggedError.messageDetail = err.messageDetail;

} catch (error) {

this.loggedError.message = 'Error occurred';

this.loggedError.messageDetail = err;

}

} else {

if (error.status !== 0) {

errMsg = `${error.status} - ${error.statusText} ${error.error}`;

}

return Observable.throw(errMsg);

}

}

private getRequestOptions(): HttpHeaders {

const headers = new HttpHeaders();

headers.append('Content-Type', 'application/json');

headers.append('Accept', 'application/json');

return headers;

}

}