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

import {

HttpClient,

HttpErrorResponse,

HttpParams,

} from '@angular/common/http';

import { Observable, of, throwError } from 'rxjs';

import { catchError, delay, map, timeout } from 'rxjs/operators';

import { Update } from '@ngrx/entity';

import { DataServiceError } from './data-service-error';

import { DefaultDataServiceConfig } from './default-data-service-config';

import {

EntityCollectionDataService,

HttpMethods,

QueryParams,

RequestData,

} from './interfaces';

import { HttpUrlGenerator } from './http-url-generator';

/\*\*

\* A basic, generic entity data service

\* suitable for persistence of most entities.

\* Assumes a common REST-y web API

\*/

export class DefaultDataService<T> implements EntityCollectionDataService<T> {

protected \_name: string;

protected delete404OK: boolean;

protected entityName: string;

protected entityUrl: string;

protected entitiesUrl: string;

protected getDelay = 0;

protected saveDelay = 0;

protected timeout = 0;

get name() {

return this.\_name;

}

constructor(

entityName: string,

protected http: HttpClient,

protected httpUrlGenerator: HttpUrlGenerator,

config?: DefaultDataServiceConfig

) {

this.\_name = `${entityName} DefaultDataService`;

this.entityName = entityName;

const {

root = 'api',

delete404OK = true,

getDelay = 0,

saveDelay = 0,

timeout: to = 0,

} = config || {};

this.delete404OK = delete404OK;

this.entityUrl = httpUrlGenerator.entityResource(entityName, root);

this.entitiesUrl = httpUrlGenerator.collectionResource(entityName, root);

this.getDelay = getDelay;

this.saveDelay = saveDelay;

this.timeout = to;

}

add(entity: T): Observable<T> {

const entityOrError =

entity || new Error(`No "${this.entityName}" entity to add`);

return this.execute('POST', this.entityUrl, entityOrError);

}

delete(key: number | string): Observable<number | string> {

let err: Error | undefined;

if (key == null) {

err = new Error(`No "${this.entityName}" key to delete`);

}

return this.execute('DELETE', this.entityUrl + key, err).pipe(

// forward the id of deleted entity as the result of the HTTP DELETE

map((result) => key as number | string)

);

}

getAll(): Observable<T[]> {

return this.execute('GET', this.entitiesUrl);

}

getById(key: number | string): Observable<T> {

let err: Error | undefined;

if (key == null) {

err = new Error(`No "${this.entityName}" key to get`);

}

return this.execute('GET', this.entityUrl + key, err);

}

getWithQuery(queryParams: QueryParams | string): Observable<T[]> {

const qParams =

typeof queryParams === 'string'

? { fromString: queryParams }

: { fromObject: queryParams };

const params = new HttpParams(qParams);

return this.execute('GET', this.entitiesUrl, undefined, { params });

}

update(update: Update<T>): Observable<T> {

const id = update && update.id;

const updateOrError =

id == null

? new Error(`No "${this.entityName}" update data or id`)

: update.changes;

return this.execute('PUT', this.entityUrl + id, updateOrError);

}

// Important! Only call if the backend service supports upserts as a POST to the target URL

upsert(entity: T): Observable<T> {

const entityOrError =

entity || new Error(`No "${this.entityName}" entity to upsert`);

return this.execute('POST', this.entityUrl, entityOrError);

}

protected execute(

method: HttpMethods,

url: string,

data?: any, // data, error, or undefined/null

options?: any

): Observable<any> {

const req: RequestData = { method, url, data, options };

if (data instanceof Error) {

return this.handleError(req)(data);

}

let result$: Observable<ArrayBuffer>;

switch (method) {

case 'DELETE': {

result$ = this.http.delete(url, options);

if (this.saveDelay) {

result$ = result$.pipe(delay(this.saveDelay));

}

break;

}

case 'GET': {

result$ = this.http.get(url, options);

if (this.getDelay) {

result$ = result$.pipe(delay(this.getDelay));

}

break;

}

case 'POST': {

result$ = this.http.post(url, data, options);

if (this.saveDelay) {

result$ = result$.pipe(delay(this.saveDelay));

}

break;

}

// N.B.: It must return an Update<T>

case 'PUT': {

result$ = this.http.put(url, data, options);

if (this.saveDelay) {

result$ = result$.pipe(delay(this.saveDelay));

}

break;

}

default: {

const error = new Error('Unimplemented HTTP method, ' + method);

result$ = throwError(error);

}

}

if (this.timeout) {

result$ = result$.pipe(timeout(this.timeout + this.saveDelay));

}

return result$.pipe(catchError(this.handleError(req)));

}

private handleError(reqData: RequestData) {

return (err: any) => {

const ok = this.handleDelete404(err, reqData);

if (ok) {

return ok;

}

const error = new DataServiceError(err, reqData);

return throwError(error);

};

}

private handleDelete404(error: HttpErrorResponse, reqData: RequestData) {

if (

error.status === 404 &&

reqData.method === 'DELETE' &&

this.delete404OK

) {

return of({});

}

return undefined;

}

}

/\*\*

\* Create a basic, generic entity data service

\* suitable for persistence of most entities.

\* Assumes a common REST-y web API

\*/

@Injectable()

export class DefaultDataServiceFactory {

constructor(

protected http: HttpClient,

protected httpUrlGenerator: HttpUrlGenerator,

@Optional() protected config?: DefaultDataServiceConfig

) {

config = config || {};

httpUrlGenerator.registerHttpResourceUrls(config.entityHttpResourceUrls);

}

/\*\*

\* Create a default {EntityCollectionDataService} for the given entity type

\* @param entityName {string} Name of the entity type for this data service

\*/

create<T>(entityName: string): EntityCollectionDataService<T> {

return new DefaultDataService<T>(

entityName,

this.http,

this.httpUrlGenerator,

this.config

);

}

}