Comenzamos creando un nuevo proyecto:

npm install Typescript

npm i rxjs

npm i @types/node

Creamos un proyecto:

tsc init

Y configuramos tsconfig.json:

{

"compilerOptions": {

/\* Basic Options \*/

"target": "es6",

"module": "commonjs",

"outDir": "./dist",

"strict": false,

"resolveJsonModule": true,

"esModuleInterop": true

},

"include": [

"./src/\*\*/\*.ts",

"./tests/\*\*/\*.ts",

]

}

Instalamos un módulo que nos permite realizar peticiones Http mediante NodeJS y de esta forma podemos ejecutarlo el cliente desde nuestro servidor. Esto es debido a qué el objeto Ajax de RxJS es solo compatible con XMLHttpRequest de los navegadores.

npm install xhr2

#### Código fuente del cliente

El archivo client.ts define la clase que desarrolla al cliente del servicio. Su código es:

/\*\*

\* File: client.ts

\* Description: Client for API RestFul Service (CRUD)

\* @autor: Carlos Serrano Sánchez for Developodo

\*/

import { Observable, of } from "rxjs";

import { ajax } from "rxjs/ajax";

import { map, catchError } from "rxjs/operators";

/\*\*

\* The only class to be exported: clientRestful

\*/

export class clientRestful {

private endPoint: string; //url of service

private XHR2: any; //method to implement XMLHttpRequest on Web and NodeJS

/\*\*

\* @param endP url of api restful, if not present it sets an endpoint by default

\*/

constructor(endP?: string) {

if (endP) {

this.endPoint = endP;

} else {

this.endPoint = "http://localhost/items";

}

//On NodeJS doesn't exist XMLHttpRequest, so if we want to create a client on server side we have to

//provide the xhr2 library: https://www.npmjs.com/package/xhr2

this.XHR2 =

typeof XMLHttpRequest !== "undefined" ? XMLHttpRequest : require("xhr2"); //hack to emulate XMLHttpRequest on NodeJS

}

/\*\*

\* It sends POST request with title and description fields: CREATE

\* @param newItemTitle title to new Item

\* @param newItemDescription description to new Item

\* @returns Observable<any> the response from API

\*/

createItem(

newItemTitle: string,

newItemDescription: string

): Observable<any> {

return ajax({

url: this.endPoint,

createXHR: () => new this.XHR2(),

method: "POST",

crossDomain: true,

hasContent: true,

headers: {

"Content-Type": "application/x-www-form-urlencoded; charset=utf-8" //application/json

},

body: {

title: newItemTitle,

description: newItemDescription

}

}).pipe(

map(response => {

return response.response;

}),

catchError(error => {

return of(error);

})

);

}

/\*\*

\* it calls getItemById without any parameters: reading all items

\*/

getItems() : Observable<any> {

return this.getItemById();

}

/\*\*

\* It sends a GET request to Server to READ items

\* @param id optional parameter to get items.If it is ommited, all items are retrieved.

\*/

getItemById(id?: any): Observable<any> {

return ajax({

url: id ? this.endPoint + "?id=" + id : this.endPoint,

createXHR: () => new this.XHR2(),

crossDomain: true,

async: true,

headers: {

"Content-Type": "application/x-www-form-urlencoded; charset=utf-8" //application/json

},

method: "GET"

}).pipe(

map(response => {

return response.response; //it maps the output, getting the response field of the object

}),

catchError(error => {

return of(error); //it captures the error and return an observable (according to function definition) of error message

})

);

}

/\*\*

\* Similar to getItemById but filtering by title field

\* @param title title parameter to filter items

\*/

getItemByTitle(title: any): Observable<any> {

return ajax({

url: this.endPoint + "?title=" + title,

createXHR: () => new this.XHR2(),

crossDomain: true,

method: "GET"

}).pipe(

map(response => {

return response.response;

}),

catchError(error => {

return of(error);

})

);

}

/\*\*

\* It sends a PUT request to UPDATE the item

\* @param id id of item to be updated

\* @param newItemTitle new title to update selected item

\* @param newItemDescription new description to update selected item

\*/

updateItem(

id: any,

newItemTitle: string,

newItemDescription: string

): Observable<any> {

return ajax({

url:

this.endPoint +

"/" +

id +

"?title=" +

newItemTitle +

"&description=" +

newItemDescription,

createXHR: () => new this.XHR2(),

method: "PUT",

crossDomain: true,

headers: {

"Content-Type": "application/x-www-form-urlencoded; charset=utf-8" //application/json

}

}).pipe(

map(response => {

return response.response;

}),

catchError(error => {

return of(error);

})

);

}

/\*\*

\* It sends a DELETE request to DELETE the selected item

\* @param id id of item to be removed

\*/

removeItem(id: any): Observable<any> {

return ajax({

url: this.endPoint + "/" + id,

createXHR: () => new this.XHR2(),

crossDomain: true,

async: true,

method: "DELETE",

headers: {

"Content-Type": "application/x-www-form-urlencoded; charset=utf-8" //application/json

}

}).pipe(

map(response => {

return response.response;

}),

catchError(error => {

return of(error);

})

);

}

}

Ahora creamos un archivo index.ts que nos sirve para hacer uso del cliente:

import { clientRestful } from "./client";

import { forkJoin } from 'rxjs';

let client=new clientRestful("http://localhost/items/");

let tasks$ = [];

console.log("creating...");

tasks$.push(client.createItem("test1","test1"));

tasks$.push(client.createItem("test2","test2"));

tasks$.push(client.createItem("test3","test3"));

forkJoin(...tasks$).subscribe(

results => { console.log(results);

console.log("All is created, now time to read");

readAll();

})

function readAll(){

client.getItems().subscribe((res)=>{

console.log(res);

})

}