Angular S5: Servicios - Peticiones Apis

Servicios para hacer peticiones a APIS

Lo primero como siempre nos crearemos un proyecto para poder seguir la clase con un ejemplo:

```
ng new htttp-app

cd htttp-app

cd src/app

ng generate component request-example
```

Por fin hemos llegado a una de las partes más importantes de Angular, la comunicación con el servidor. Para ello tenemos que trabajar algunos conceptos como el HttpClient.

Angular HttpClient

Hagamos algunas peticones rápidas para que conozcamos los conceptos de antemano y podamos avanzar sin preocupaciones.

En primer lugar, importamos el **HttpClientModule** en nuestro **app.module.ts**:

```
import { HttpClientModule } from '@angular/common/http';
```

Y lo importamos:

```
imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
```

El HttpClientModule es un modulo de Angular que nos ayudará a realizar peticiones contra una API, lo bueno de utilizar Frameworks como angular es que parte del trabajo nos lo da hecho.

¡Ahora podemos crear servicios llenos de peticiones! Para ello vamos a utilizar una API pública, y en este caso utilizaremos la API de Rick y Morty. Porque... ¿a quién no le gusta Rick y Morty?

Nuestro EndPoint sobre el que atacaremos es:

```
https://rickandmortyapi.com/api/character/
```

Y en caso de que queramos solicitar datos "paginados" para que no vengan muchos resultados, podemos hacer:

```
https://rickandmortyapi.com/api/character/?page=2
```

Después de tener nuestra estructura de proyecto crearemos un Servicio dentro de nuestro componente ya que de momento soló lo consumiremos desde ahí:

```
cd request-example
ng generate service services/request-example
```

Y comenzamos a trabajar con nuestro **request-example.service.ts** para manejar una sola solicitud a los personajes:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

//EndPoint base sobre el que atacaremos
const baseUrl = 'https://rickandmortyapi.com/api/';

//La petición character
const characterUrl = this.baseUrl + 'character';

@Injectable()
export class RequestExampleService {

  constructor(private http: HttpClient) { }

  getCharacters() {
    return this.http.get(characterUrl);
  }
}
```

Y recordad lo importamos en nuestro app.module.ts en los providers:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RequestExampleComponent } from './request-example/request-example.component';
import { HttpClientModule } from '@angular/common/http';
```

```
import { RequestExampleService } from './request-example/services/request-example.service';

@NgModule({
  declarations: [
    AppComponent,
    RequestExampleComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [RequestExampleService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Fácil eh? Importamos el módulo **HttpClient** allí y solo buscamos usando **http.get** desde el módulo inyectado.

¿Qué nos devolverá esta petición? Un **observable**! Esta es una buena descripción de ellos:

> Observables provide support for passing messages between publishers and subscribers in your application. Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values.

Es prácticamente una forma súper genial y declarativa de codificar reactivamente. Pero no os preocupéis que en la próxima lección nos adentraremos dentro del mundo de los observables, por ahora nos centraremos en como recibir esa info.

Entonces, ¿cómo podemos obtener el resultado de esa búsqueda? Usando el método **.subscribe**, que permite indicar una función a ejecutar en caso de éxito en la llamada, y otra función en caso de error.

La respuesta en el caso de la petición realizada tiene este formato:

```
{
  "info": {},
  "results": [{
    "id": 1,
    "name": "Rick Sanchez",
    "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg"
  }]
}
```

Por lo tanto, podemos crear interfaces para la respuesta esperada y el objeto de carácter recibido. Y es lo primero que vamos a hacer antes de trabajar en nuestro componente. Siempre es bueno tipar la entrada y salida de datos para controlar el flujo de información de nuestra aplicación.

Por ello lo primero será crear una carpeta dentro de nuestro request-example con el nombre de models, y dentro de esta definir nuestra interfaz. Algo como:

```
export interface CharacterInterface {
  id: number;
  name: string;
  image: string;
}

export interface CharacterResponseInterface {
  info: {
    count: number;
    next: string;
    pages: number;
    prev: string;
  };
  results: CharacterInterface[];
}
```

Ahora cambiemos nuestro **request-example.component.ts**:

```
import { Component, OnInit } from '@angular/core';
import { CharacterInterface, CharacterResponseInterface } from './models/character.interface'
import { RequestExampleService } from './services/request-example.service';

@Component({
  selector: 'app-request-example',
  templateUrl: './request-example.component.html',
  styleUrls: ['./request-example.component.scss']
})
export class RequestExampleComponent implements OnInit {
  // declaramos la variable donde almacenamos nuestro resultado
  characterList: CharacterInterface[] = [];

  // Llamamos a nuestro servicio o inicializamos servicio
  constructor(private requestExampleService: RequestExampleService) {}

  // Al arrancar nuestra aplicación:
  ngOnInit() {
    // Utilizamos la función getCharacters para guardar nuestros resultados:
    this.requestExampleService.getCharacters()
    .subscribe((data: CharacterResponseInterface) => {
      const results: CharacterInterface[] = data.results;

      const formattedResults = results.map(({ id, name, image }) => ({
        id,
        name,
        image,
      }));
      this.characterList = formattedResults;
    });
  }
}
```

Vamos a explicar qué ha sucedido aquí:

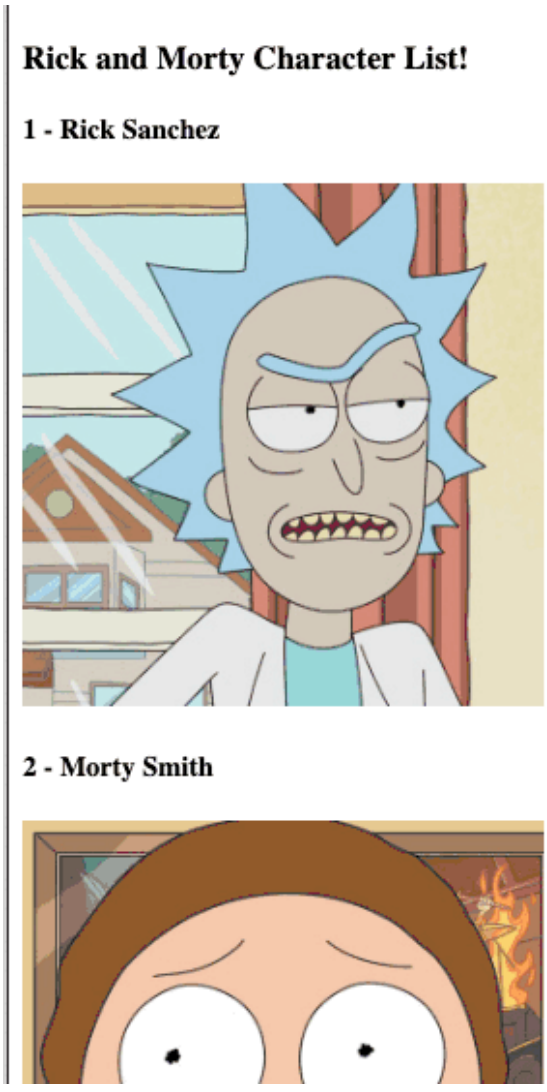
1. Hemos importado el **RequestExampleService para** que podamos acceder a él más adelante desde nuestras funciones.
2. En el método **ngOnInit**, llamamos al método **requestExampleService.getCharacters()** y nos suscribimos al observable devuelto. Para ello le indicamos la función a ejecutar en caso de que la respuesta del servidor sea correcta, en ese caso lo que haremos es guardar el Array de resultados dentro de nuestra variable characterList.

Ahora cambiemos el archivo HTML para mostrar los caracteres:

```
<div *ngIf="characterList">
  <h3>Rick and Morty Character List!</h3>
  <div *ngFor="let character of characterList">
    <h4>{{ character.id }} - {{ character.name }}</h4>
    <img [src]="character.image" [alt]="character.name">
  </div>
</div>
```

Pintamos contenido solo cuando se define **characterList** , y luego iteramos a través de la lista para crear divs de caracteres con la id, el nombre y la imagen.

Obteniendo el siguiente resultado:



¡Esto esta encendido! 🔥😎

Podríamos seguir usando solicitudes POST o paginación, pero mantengamos eso en espera. Una vez aprendamos bien a usar el módulo de router completaremos una aplicación desde cero.