



Florida
Universitària

Cliente HTTP en Angular

Curso 2025/2026

Paco Segura

Cliente HTTP

- Angular dispone de un cliente para realizar las peticiones HTTP.
- Para poder utilizarlo a lo largo de nuestra aplicación, haremos uso de dos herramientas que nos provee Angular:
 - Servicios.
 - Inyección de dependencias.

Servicios

- Se utilizan para implementar funcionalidades que queremos utilizar a lo largo de la aplicación.
- Se definen en un archivo y luego se *inyectan* en los distintos componentes a utilizar.
- En este caso utilizaremos un servicio para implementar las funciones necesarias para realizar las peticiones HTTP.

Inyección de dependencias

- Técnica muy empleada en *frameworks*. Permite implementar funcionalidades que luego podremos inyectar en nuestros componentes.
- En el componente que queramos utilizar el servicio no será necesario “requerirlo”, bastará con inyectarlo en la clase del componente para poder utilizarlo.
- En nuestro caso inyectaremos el servicio donde tengamos implementadas las peticiones HTTP en todos los componentes en que necesitemos usar el cliente HTTP.

Ejemplo: Cliente HTTP

- Vamos a ver cómo crear un servicio donde hacer uso del Cliente HTTP para luego poder inyectarlo y utilizarlo en nuestros componentes.
- Pasos:
 - **1º Creación del servicio**
 - `ng generate service services/nombre_servicio`
 - Genera dos archivos:
 - `nombre_servicio.spec.ts`
 - `nombre_servicio.ts`

Ejemplo: Cliente HTTP

- Para poder utilizar el Cliente HTTP de Angular en nuestro servicio, primero tenemos que proveerlo desde la raíz del proyecto.
- En archivo **app.config.ts** importamos **provideHttpClient** y **withFetch**:

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideHttpClient, withFetch } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideRouter(routes),
    provideHttpClient(withFetch())
  ]
};
```

Ejemplo: Cliente HTTP

- En archivo **nombre_servicio.ts** importamos el cliente HTTP y lo inyectamos.

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

```
@Injectable({  
  providedIn: 'root',  
})  
export class Request {  
  private http = inject(HttpClient);
```

Ejemplo: Cliente HTTP

- En archivo **nombre_servicio.ts** implementamos las funciones con las peticiones para luego hacerlas servir en nuestro componente.
- Las peticiones HTTP se realizan mediante `this.http.peticion`.
- Importamos Observable, que es similar a Promises, para gestionar la respuesta asíncrona del servidor.

Ejemplo: Cliente HTTP

```
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { Observable } from 'rxjs';

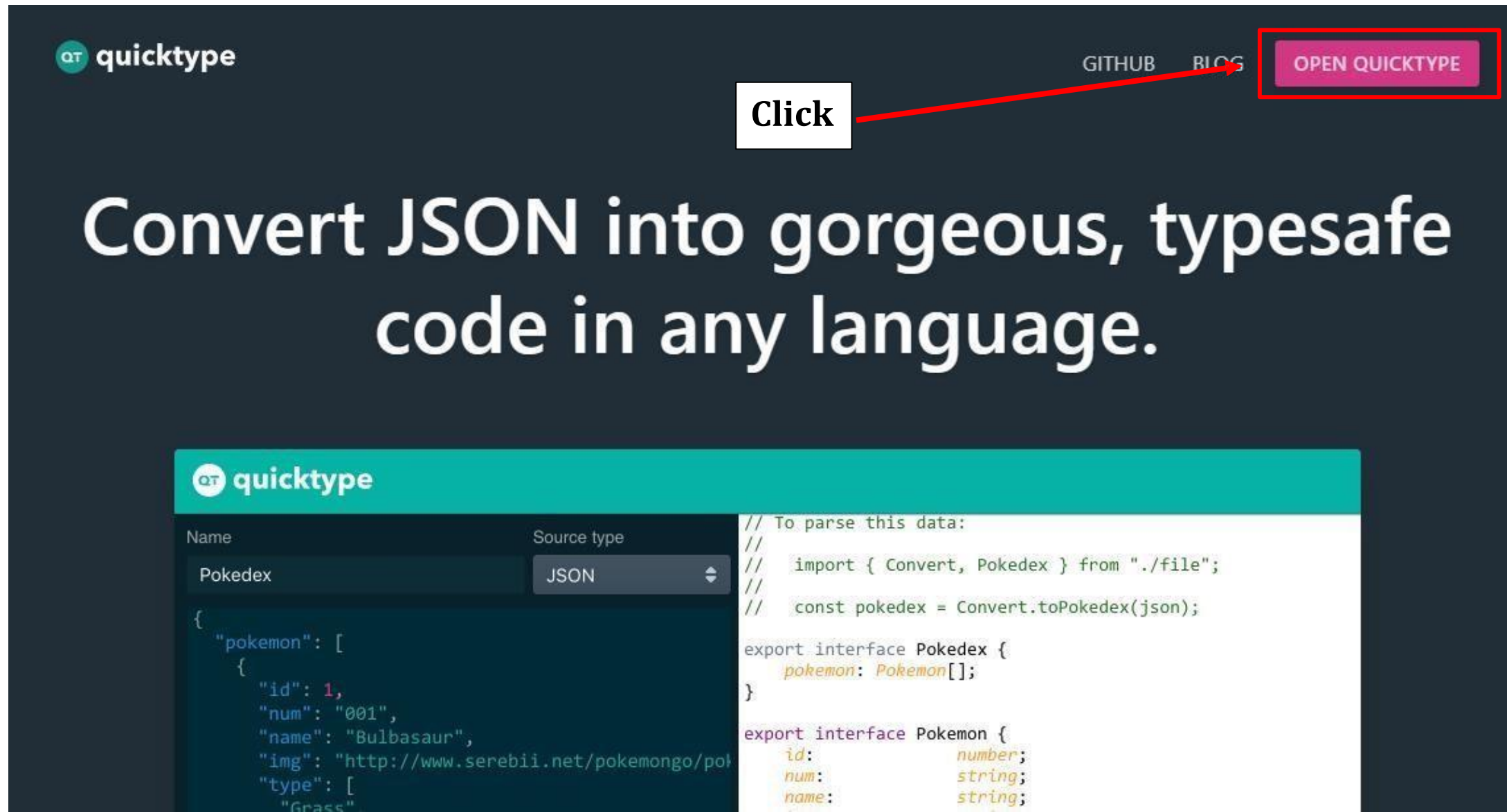
@Injectable({
  providedIn: 'root',
})
export class Request {
  private http = inject(HttpClient);

  public getData(): Observable<any> {
    return this.http.get<any>('https://rickandmortyapi.com/api/character');
  }
}
```

Ejemplo: Cliente HTTP

- Para evitar tener una respuesta de tipo 'any', utilizaremos una interface que nos permitirá gestionar mejor la respuesta a la petición HTTP.
- Para ello generamos una interface (response.interface.ts) para la respuesta a la petición HTTP en una carpeta de nombre 'models' o 'interfaces'.
- Para crear el tipado de la interface de forma rápida podemos utilizar <https://quicktype.io/>

Ejemplo: Cliente HTTP



The image shows the Quicktype website and a preview of its interface. The website header includes the Quicktype logo, links to GitHub and Blog, and a prominent 'OPEN QUICKTYPE' button highlighted with a red box. A red arrow points from a 'Click' label to this button. The main headline reads 'Convert JSON into gorgeous, typesafe code in any language.' Below this, a preview of the Quicktype application is shown. It features a teal header with the logo, a sidebar with 'Name' and 'Source type' filters (set to 'Pokedex' and 'JSON'), and a main area displaying JSON input on the left and the corresponding TypeScript code on the right.

QT quicktype

GITHUB BLOG

OPEN QUICKTYPE

Click

Convert JSON into gorgeous, typesafe code in any language.

QT quicktype

Name: Pokedex Source type: JSON

```
{
  "pokemon": [
    {
      "id": 1,
      "num": "001",
      "name": "Bulbasaur",
      "img": "http://www.serebii.net/pokemongo/po",
      "type": [
        "Grass"
      ]
    }
  ]
}
```

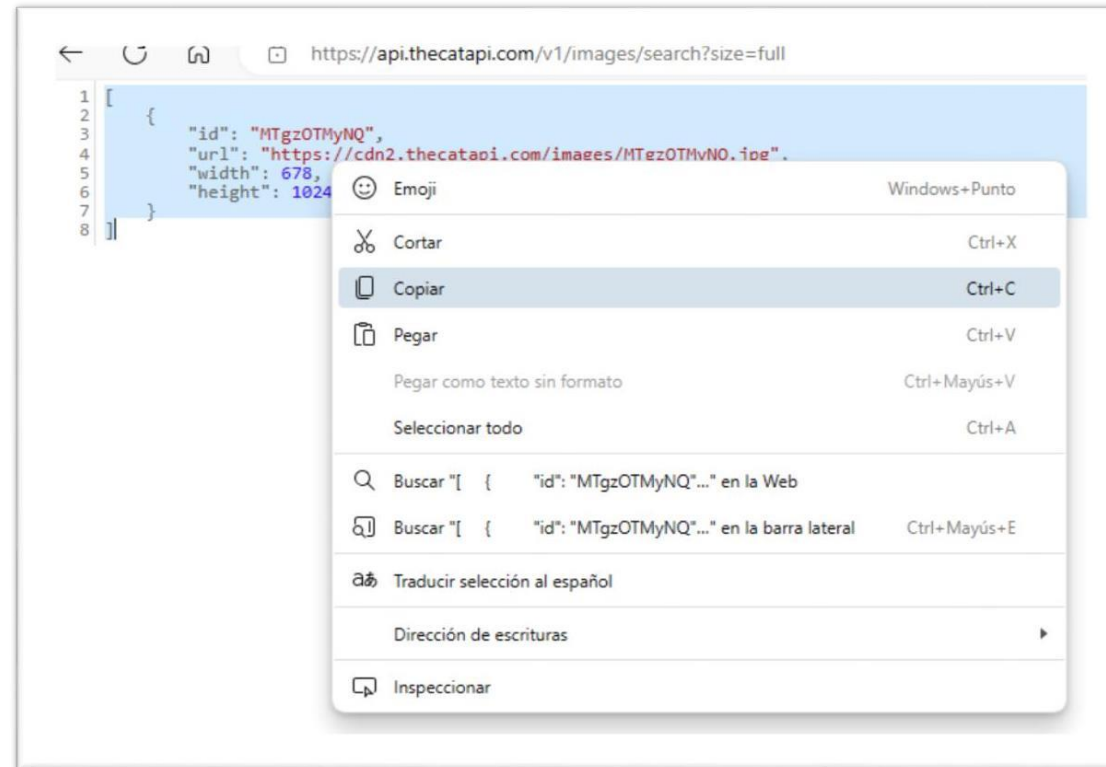
```
// To parse this data:
//
//   import { Convert, Pokedex } from "./file";
//
//   const pokedex = Convert.toPokedex(json);

export interface Pokedex {
  pokemon: Pokemon[];
}

export interface Pokemon {
  id: number;
  num: string;
  name: string;
}
```

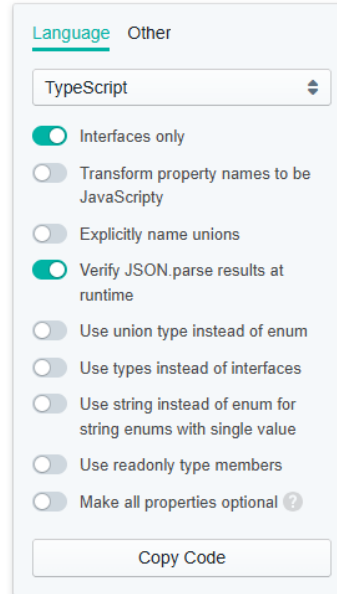
Ejemplo: Cliente HTTP

- A continuación cargamos la respuesta de la url de la API REST pública que queremos utilizar y copiamos los datos que devuelve. Ejemplo para [The Cat API](https://api.thecatapi.com/v1/images/search?size=full):



Ejemplo: Cliente HTTP

- En la web de QuickType seleccionamos del menú lateral derecho TypeScript y añadimos la opción 'Interfaces only':



The screenshot shows the QuickType web interface. At the top, there are two tabs: 'Language' (which is underlined in green) and 'Other'. Below the tabs is a dropdown menu currently showing 'TypeScript'. Underneath the dropdown is a list of toggle switches. The first toggle, 'Interfaces only', is turned on (green). The other toggles are turned off (grey): 'Transform property names to be JavaScripty', 'Explicitly name unions', 'Verify JSON.parse results at runtime', 'Use union type instead of enum', 'Use types instead of interfaces', 'Use string instead of enum for string enums with single value', 'Use readonly type members', and 'Make all properties optional'. At the bottom of the panel is a 'Copy Code' button.

Language Other

TypeScript

☒ Interfaces only

☐ Transform property names to be JavaScripty

☐ Explicitly name unions

☒ Verify JSON.parse results at runtime

☐ Use union type instead of enum

☐ Use types instead of interfaces

☐ Use string instead of enum for string enums with single value

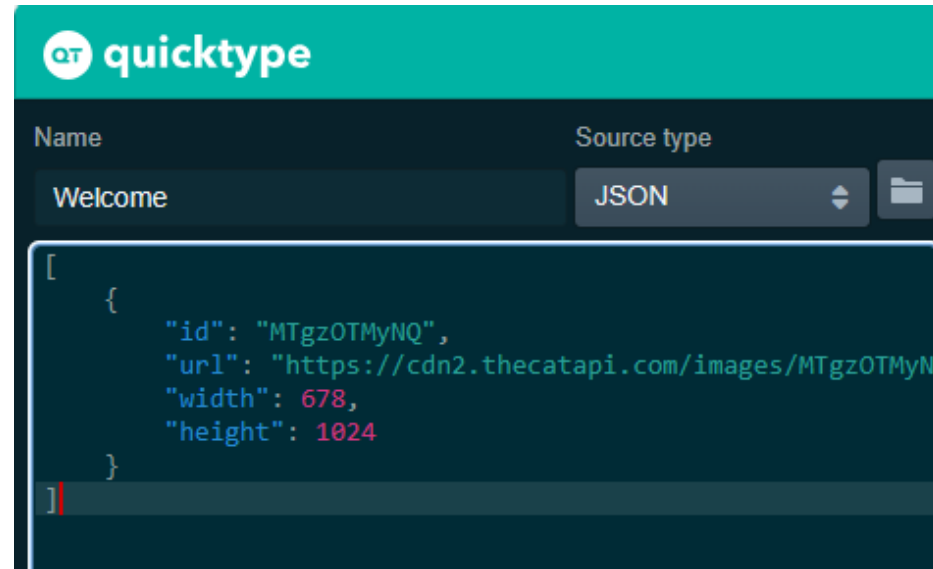
☐ Use readonly type members

☐ Make all properties optional ?

Copy Code

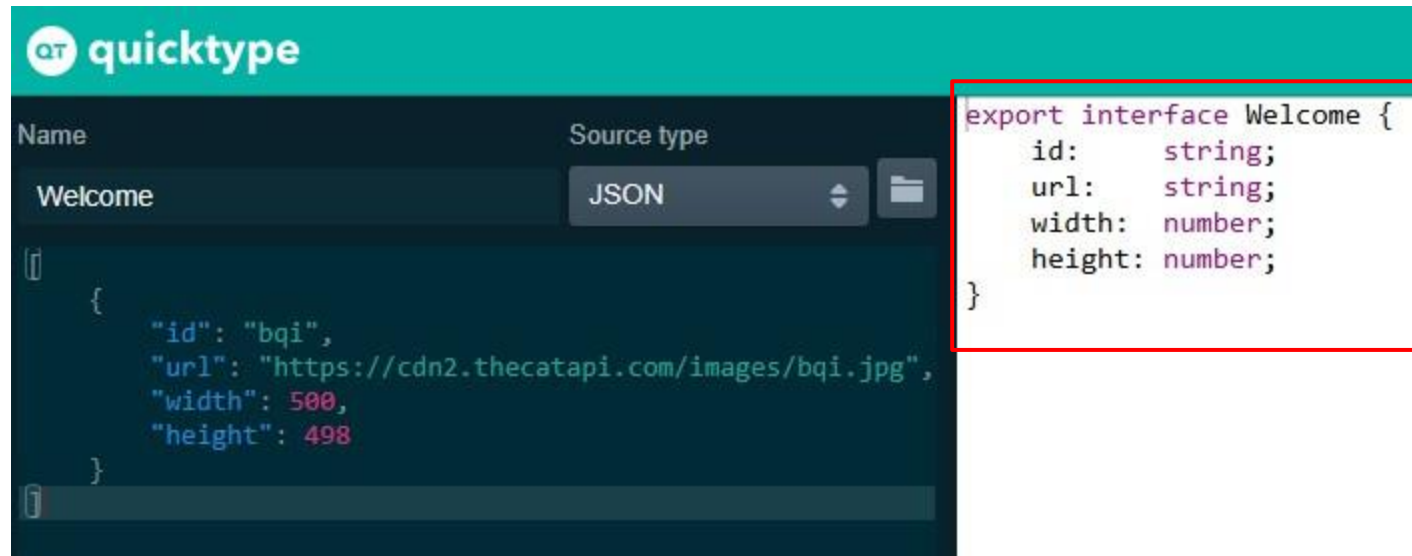
Ejemplo: Cliente HTTP

- Pegamos el código con la respuesta en la parte lateral izquierda de QuickType:



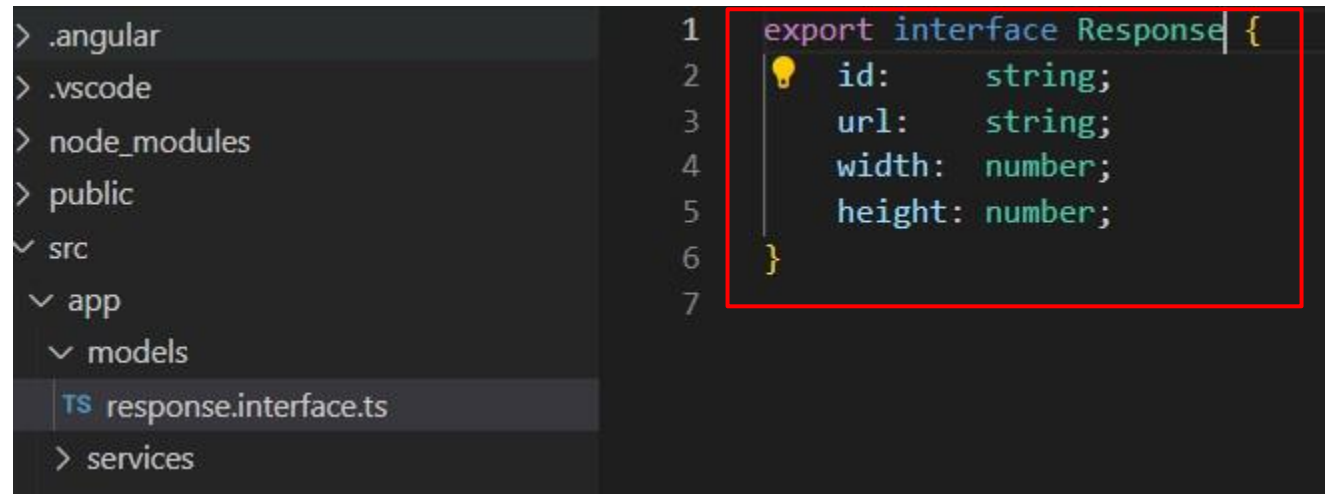
Ejemplo: Cliente HTTP

- En el centro tenemos la Interface generada automáticamente:



Ejemplo: Cliente HTTP

- Pegamos la interface en nuestro archivo **response.interface.ts** y la renombramos a **Response**:



```
1 export interface Response {  
2     id: string;  
3     url: string;  
4     width: number;  
5     height: number;  
6 }  
7
```

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file 'response.interface.ts' is selected under the 'models' folder. The main editor area displays the following TypeScript code, which is highlighted with a red rectangle:

Ejemplo: Cliente HTTP

- Volvemos a nuestro archivo ***.ts** y *tipamos* con la interface creada la respuesta de la petición HTTP:

```
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { Response } from '../interfaces/response.interface';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class Request {
  private http = inject(HttpClient);

  public getData(): Observable<Response> {
    return this.http.get<Response>('https://rickandmortyapi.com/api/character');
  }
}
```

Ejemplo: Cliente HTTP

- Pasos:
 - **2º Inyectar el servicio en el componente o componentes donde queramos hacer la petición HTTP.**

Ejemplo: Cliente HTTP

- Para ello, en nuestro componente inyectamos el servicio, que a su vez contiene inyectado el Cliente HTTP. De este modo podemos llamar a las peticiones HTTP desde varios componentes, inyectándoles el servicio.

```
import { NgClass, NgStyle } from '@angular/common';
import { Component, inject } from '@angular/core';
import { Request } from '../services/request';

@Component({
  selector: 'app-main',
  imports: [NgClass, NgStyle],
  templateUrl: './main.html',
  styleUrls: ['./main.css'],
})
export class Main {
  public data = inject(Request);
```

Ejemplo: Cliente HTTP


- Desde el componente llamamos a la función del **servicio** que queremos utilizar.

```
import { NgClass, NgStyle } from '@angular/common';
import { Component, inject } from '@angular/core';
import { Request } from '../services/request';

@Component({
  selector: 'app-main',
  imports: [NgClass, NgStyle],
  templateUrl: './main.html',
  styleUrls: ['./main.css'],
})
export class Main {
  public data = inject(Request);

  public getResponse(): void {
    this.data.getData().subscribe((response) => {
      console.log(response);
    });
  }
}
```

```
public getData(): Observable<Response> {
  return this.http.get<Response>('https://rickandmortyapi.com/api/character');
}
```



Ejemplo: Cliente HTTP

- Finalmente, implementamos la función **ngOnInit()** para que la petición se realice al cargarse la aplicación.

```
import { NgClass, NgStyle } from '@angular/common';
import { Component, inject } from '@angular/core';
import { Request } from '../../services/request';

@Component({
  selector: 'app-main',
  imports: [NgClass, NgStyle],
  templateUrl: './main.html',
  styleUrls: ['./main.css'],
})
export class Main {
  public data = inject(Request);

  public getResponse(): void {
    this.data.getData().subscribe((response) => {
      console.log(response);
    });
  }

  public ngOnInit(){
    this.getResponse();
  }
}
```