HTTP

La mayoría de las aplicaciones front-end necesitan comunicarse con un servidor a través del protocolo HTTP para descargar o cargar datos y acceder a otros servicios back-end.

Angular proporciona una API para implementar un cliente HTTP para aplicaciones Angular, la clase de servicio HttpClient se encuentra en @angular/common/http

El servicio de cliente HTTP ofrece las siguientes características principales.

- •La capacidad de solicitar objetos de respuesta escritos
- •Manejo de errores simplificado
- •Características de capacidad de prueba
- •Intercepción de solicitudes y respuestas

Configuración para la comunicación del servidor mediante la clase HttpClient

Antes de poder usar HttpClient se debe importar el modulo HttpClientModule, la mayoría de las aplicaciones lo hacen en el modulo raíz de la aplicación, pero también se puede aplicar de manera modular.

Luego se puede inyectar el servicio HttpClient como una dependencia de la aplicación, como se muestra en el siguiente ejemplo.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Injectable()
export class ConfigService {
     constructor(private http: HttpClient) { }
}
```

HttpClient metodo get

Se utiliza el método HttpClient.get() para obtener datos de un servidor. El método asíncrono envía una solicitud HTTP y devuelve un Observable que emite los datos solicitados cuando se recibe la respuesta. El tipo de devolución varía según los valores observe y responseType que pase a la llamada.

El método get() toma dos argumentos; la URL del punto final desde la que se obtendrán los datos y un objeto de opciones que se utiliza para configurar la solicitud.

```
options: { headers?: HttpHeaders | {[header: string]: string | string[]},
          observe?: 'body' | 'events' | 'response',
          params?: HttpParams|{[param: string]: string | number |
boolean |                          ReadonlyArray<string | number |
boolean>},
          reportProgress?: boolean,
          responseType?: 'arraybuffer'|'blob'|'json'|'text',
          withCredentials?: boolean, }
```

Las opciones mas importantes son observe y responseType.

- La opción de observe especifica qué parte de la respuesta devolver, el valor predeterminado es body
- La opción responseType especifica el formato en el que se devolverán los datos, el valor predeterminado es json.

Manejo de errores de solicitud

Si la solicitud falla en el servidor, HttpClientdevuelve un objeto de error en lugar de una respuesta exitosa.

El mismo servicio que realiza las transacciones de su servidor también debe realizar la inspección, interpretación y resolución de errores.

Cuando ocurre un error, puede obtener detalles de lo que falló para informar a su usuario. En algunos casos, también puede volver a intentar la solicitud automáticamente .

Obtener detalles del error

Una aplicación debe brindar al usuario comentarios útiles cuando falla el acceso a los datos. Un objeto de error sin procesar no es particularmente útil como retroalimentación. Además de detectar que se ha producido un error, debe obtener los detalles del error y utilizarlos para redactar una respuesta fácil de usar.

Pueden ocurrir dos tipos de errores.

- El backend del servidor puede rechazar la solicitud y devolver una respuesta HTTP con un código de estado como 404 o 500. Estas son respuestas de error .

•Algo podría salir mal en el lado del cliente, como un error de red que impide que la solicitud se complete con éxito

HttpClient captura ambos tipos de errores en su archivo HttpErrorResponse.Se necesita revisar esa respuesta para identificar la causa del error.

El siguiente es un ejemplo de método para manejar errores

```
private handleError(error: HttpErrorResponse) {
     if (error.status === 0) { // A client-side or network error occurred. Handle it                                    //accordingly.
          console.error('An error occurred:', error.error);
     } else { // The backend returned an unsuccessful response code. // The response                    //body may contain clues as to what went wrong.
          console.error( `Backend returned code ${error.status}, body was: `,                              error.error); } // Return an observable with a user-
     //facing error message.
     return throwError(() => new Error('Something bad happened; please try again                                        later.'));
}
```

para ejecutar este manejador de errores se utiliza un metodo de la libreria RxJs llamado catchError y se puede utilizar como en el ejemplo siguiente

```
getConfig() {
     return this.http.get<Config>(this.configUrl)
                          .pipe(
                    catchError(this.handleError)
               );
}
```

Envío de datos a un servidor

Además de obtener datos de un servidor, HttpClientadmite otros métodos HTTP como PUT, POST y DELETE, que puede usar para modificar los datos remotos.

Hacer una solicitud POST

Las aplicaciones a menudo envían datos a un servidor con una solicitud POST al enviar un formulario.

Lo siguiente es un codigo de ejemplo para enviar una solicitud tipo post utilizando HttpClient.

```
addHero(hero: Hero): Observable<Hero> {
      return this.http.post<Hero>(this.heroesUrl, hero, httpOptions)
                        .pipe(
                  catchError(this.handleError('addHero', hero))
              );
}
```

El método HttpClient.post() es similar a get()que tiene un parámetro de tipo, que puede usar para especificar que tipo de dato se espera que el servidor devuelva. El método toma una URL de recurso y dos parámetros adicionales:

PARÁMETRO	DETALLES
cuerpo	Los datos para el cuerpo de la solicitud.
opciones	Un objeto que contiene opciones a aplicar en la solicitud, aquí también se especifican los encabezados requeridos.

Realizar una solicitud mediante el método delete

El método delete funciona de manera similar al método get, necesita un parámetro url y un parámetro de opciones a aplicar a la petición que es opcional

Lo siguiente es un ejemplo de este tipo de solicitud.

```
deleteHero(id: number): Observable<unknown> {
      const url = `${this.heroesUrl}/${id}`; // DELETE api/heroes/42
return    this.http.delete(url, httpOptions)
          .pipe(
               catchError(this.handleError('deleteHero'))
          );
}
```

Realizar una solicitud mediante el método put

El método put funciona de manera similar al método post, necesita un parámetro url , un parámetro body, en el cual se especifica los datos que se enviaran al servidor, y un parámetro opciones a aplicar a la petición que es opcional

Lo siguiente es un ejemplo de este tipo de solicitud.

```
updateHero(hero: Hero): Observable<Hero> {
     return this.http.put<Hero>(this.heroesUrl, hero, httpOptions)
               .pipe(
                    catchError(this.handleError('updateHero', hero))
          );
}
```

Agregar y actualizar encabezados

Muchos servidores requieren encabezados adicionales para las operaciones de guardado. Por ejemplo, un servidor puede requerir un token de autorización o un encabezado de "Tipo de contenido" para declarar explícitamente el tipo MIME del cuerpo de la solicitud.

Agregar encabezados

El siguiente código es un ejemplo de para modificar la cabecera de la petición, solo basta con agregar la constante httpOptions al parámetro de opciones de la petición que es realizará.

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

const httpOptions = {
     headers: new HttpHeaders({
          'Content-Type': 'application/json',
          Authorization: 'my-auth-token' })
};
```

Interceptar solicitudes y respuestas.

Se declaran interceptores para inspeccionar y transformar las solicitudes HTTP que viajaran desde la aplicación a un servidor. Los mismos interceptores también pueden inspeccionar y transformar las respuestas de un servidor en su camino de regreso a la aplicación.

Los interceptores pueden realizar una variedad de tareas *implícitas*, desde la autenticación hasta el registro, de manera rutinaria y estándar, para cada solicitud/respuesta HTTP.

Sin los interceptores, se tendrían que implementar estas tareas *explícitamente* para cada llamada de los métodos de HttpClient.

Escribir un interceptor

Para declarar un interceptor, basta con crear una clase de servicio e implementar la interfaz HttpInterceptor así como a su vez el método intercept, este método se encargara de interceptar u darle tratamiento a la solicitud.

Lo siguiente es el ejemplo de código de una clase interceptora:

```
export class NoopInterceptor implements HttpInterceptor {
     intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {        return next.handle(req);
     }
}
```

También se necesita definir que esta clase será una clase interceptora en el archivo module de la aplicación, esto se hace agregando a la sección providers la clase interceptora

```
providers: [
{
provide: HTTP_INTERCEPTORS,
useClass: AuthInterceptorService,
multi: true
}
],
```

la opción multi: true, significa que este interceptor puede trabajar en conjunto con otros interceptores.

Fuentes:
https://angular.io/guide/http

https://medium.com/@insomniocode/angular-autenticaci%C3%B3n-usando-interceptors-a26c167270f4