¿Qué es un API?

Un API es una interfaz que publica una serie de funciones o procedimientos para que puedan ser utilizadas por otro software que se abstrae de cómo esté implementada dicha interfaz. Se utilizan habitualmente para reutilizar código, librerías o para integrar diferentes aplicaciones.

https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

¿Qué es un Web Service?

Es una aplicación web que está diseñado para atender peticiones de uno o varios clientes (otras aplicaciones) con los que se comunica a través de una red. Esto permite que entre dos máquinas, sistemas o aplicaciones pueda haber una comunicación enviando y recibiendo información. La parte que solicita o envía información en primer término será el CLIENTE. La parte que atiende o responde las peticiones será el SERVIDOR.

Un API basada en un Web Service es un API que está publicada como tal. Es decir, el API es el servidor que atiende peticiones a través de la red de uno o varios clientes.

¿Qué es un API REST?

Es un API basado en un Web Service. Los Web Services se pueden implementar de muchas maneras. Decimos que el API es REST cuando el Web Service está implementado basándose en la arquitectura REST.

¿Cómo se invoca un servicio REST?

El API publicada se invoca mediante una petición HTTP. Las peticiones HTTP pueden ser de varios tipos. Normalmente, el servicio REST se construye para que, en función de lo que haga cada método, se invoque utilizando un método u otro. La lógica que se sigue es la siguiente:

- Consultar (uno o varios): Usan el método GET
- Crear: Usan el método POST
- Actualizar: Usan el método PUT (actualización completa) o PATCH (actualización parcial)
- Borrar: Usan el método DELETE

Para invocarlo, podemos hacerlo directamente desde un navegador web. Aunque lo más sencillo es utilizar aplicaciones para probar servicios REST. Por ejemplo, Postman o SOAP UI.

Si desde una aplicación tenemos que invocar a un servicio Rest, necesitaremos construir previamente un cliente de ese servicio.

¿Cómo crear un servicio REST utilizando Spring Boot?

1. Crear un proyecto Java
2. Convertir nuestro proyecto a Maven Project
3. Añadir dependencia de spring
   (poner exclusión del logging para evitar incompatibilidad con slf4j)

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
	<version>3.1.4</version>
	<exclusions>
		<exclusion>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-logging</artifactId>
		</exclusion>
	</exclusions>
</dependency>
```

4. Crear una clase con un main que lance SpringApplication:

```
@SpringBootApplication
@EnableAutoConfiguration
public class App {

	public static void main(String[] args) {
		SpringApplication.run(App.class, args);
	}

}
```

5. Crear la clase del servicio como RestController:

```
@RestController
public class SampleService {
```

6. Añadir los métodos que queramos al servicio como queramos. En la cabecera del método indicaremos las anotaciones:
   a. **@GetMapping**(“/url”) → Lo usamos cuando estamos **consultando** algo
   b. **@PostMapping**(“/url”) → Lo usamos cuando estamos **creando** algo
   c. **@PutMapping**(“/url”) → Lo usamos cuando estamos **actualizando** algo
   d. **@PatchMapping**(“/url”) → Lo usamos cuando estamos **actualizando** algo de manera **selectiva**.
   e. **@DeleteMapping**(“/url”) → Lo usamos cuando estamos **borrando** algo

7. Tendremos que indicar los parámetros que vayamos a recibir:
    a. Si son parámetros que vengan en la URL del tipo "?nombreParametro=valor", entonces deberán indicarse en el método con la anotación "@RequestParam" Si queremos que un parámetro de este tipo no sea obligatorio, podemos incluir la anotación así:

```
@RequestParam(required = false) String ejemplo
```

    b. A los parámetros que lleguen en el body de la petición, les pondremos delante la anotación "@RequestBody".

    c. A los que lleguen como parte de la URL, les pondremos la anotación "@PathVariable", e incluiremos el nombre del parámetro en la misma URL entre llaves, por ejemplo, "/test/{nombreParametro}". Ejemplo:

```
@GetMapping("/cliente/{id}")
public Cliente consultarCliente(@PathVariable Long id) thro
    try {
```

¿Cómo devolver errores? (Códigos HTTP)

Un servicio REST, aparte de poder devolver al cliente una entidad en forma de JSON, también devuelve un código HTTP en la cabecera de la respuesta. Ese código tiene un significado que indica si todo ha ido bien o si ha habido errores y de qué tipo. Los principales códigos son:

https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

Nosotros podremos lanzar excepciones como siempre. En la clase de la Excepción podremos incluir la anotación "@ResponseStatus(value = XXX)"

Por ejemplo:

```
5
6 @ResponseStatus(value = HttpStatus.NOT_FOUND)
7 public class NotFoundException extends Exception {
8
```

¿Cómo crear un cliente para un servicio REST?

El cliente para poder consumir o invocar a un servicio REST lo realizaremos utilizando la tecnología apropiada según a donde vayamos a utilizarlo. Por ejemplo, si queremos invocar al servicio REST desde una APP móvil de Android, lo implementaremos para Android. Si lo queremos invocar desde una aplicación web en javascript, lo haremos para Javascript.

Si quisiéramos implementarlo en Java para utilizarlo desde una aplicación Java, hay muchos frameworks y herramientas para implementar un cliente de forma rápida y sencilla.

Con Spring, podemos utilizar la clase RestTemplate

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

Algunos ejemplos de uso:

https://www.baeldung.com/rest-template

http://www.profesor-p.com/clase-resttemplate-3/

https://howtodoinjava.com/spring-boot2/resttemplate/spring-restful-client-resttemplate-example/

Existen otros muchos Frameworks:

- RestEasy
- Jersey
- RestLet
- Etc.

CLIENTE REST CON RestTemplate de SPRING

Para crear un cliente REST, normalmente creamos una clase independiente por cada servicio REST que queramos "atacar". Por ejemplo, si queremos crear un cliente REST para poder invocar a un servicio REST de animales, nos vamos a crear una clase AnimalesClienteREST.

Para instanciar un cliente, siempre deberíamos indicar dos parámetros:

- La url base. Esta es la URL donde está alojado el servicio REST.
- El timeout. Es el tiempo máximo que queremos configurar para que el cliente espere la respuesta de cada petición.

```
public class AnimalRestClient {

    private String urlBase;
    private RestTemplate restTemplate;

    public AnimalRestClient(String urlBase, Integer msTimeout) {
        this.urlBase = urlBase;
        HttpComponentsClientHttpRequestFactory requestFactory = new HttpComponentsClientHttpRequestFactory();
        requestFactory.setConnectTimeout(msTimeout);
        this.restTemplate = new RestTemplate(requestFactory);
    }
```

URL Base que siempre será la misma

Objeto RestTemplate que usaremos para invocar al API Rest

Así creo que objeto RestTemplate

Para poder utilizar RestTemplate, en nuestro proyecto tendremos que incluir estas librerías:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>3.1.4</version>
        <exclusions>
                <exclusion>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
        </exclusions>
</dependency>

<dependency>
        <groupId>org.apache.httpcomponents.client5</groupId>
        <artifactId>httpclient5</artifactId>
        <version>5.2.1</version>
</dependency>
```

Con esto estamos en disposición de darle forma a nuestra clase con los métodos que necesitemos invocar. Lo normal es crear en nuestro cliente un método por cada método que tenga el servicio REST. Es decir, si mi servicio REST tiene un método para consultar un animal por su ID, en mi cliente crearé otro método que me permita invocar al anterior.

Veamos cómo podemos invocar al servicio REST utilizando el RestTemplate:

- Para hacer un GET con parámetros en el path de la URL

```
String url = urlBase + "/animal/{id}";
Animal respuesta = restTemplate.getForObject(url, Animal.class, id);
return respuesta;
```

En la URL ponemos el parámetro así
Indicamos la clase de la respuesta
Pasamos los parámetros aquí

- Para hacer un GET con parámetros en la URL (no en el paht)

```
String url = urlBase + "/animal?id=" + id;
Animal respuesta = restTemplate.getForObject(url, Animal.class);
return respuesta;
```

- Para hacer un GET que devuelva una lista

```
String url = urlBase + "/animales?especie=" + especie;
Animal[] respuesta = restTemplate.getForObject(url, Animal[].class);
return Arrays.asList(respuesta);
```

Convertimos el array en una lista luego
El tipo de la respuesta será un array

- Para hacer un POST enviando un objeto en el request body

```
String url = urlBase + "/animal";
animal = restTemplate.postForObject(url, animal, Animal.class);
return animal;
```

Este es el objeto que enviamos en el request body
Este es el tipo de objeto de la respuesta

- Para hacer un PATCH enviando un objeto en el request body (igual que el POST)

```
String url = urlBase + "/animal";
animal = restTemplate.patchForObject(url, animal, Animal.class);
return animal;
```

Similar al POST

- Para hacer un PUT enviando un objeto en el request body

```
String url = urlBase + "/animal";
restTemplate.put(url, animal);
```

Un PUT no debería devolver ningún objeto

- Para hacer un DELETE pasando parámetro en el PATH de la URL

```
String url = urlBase + "/animal/{id}";
restTemplate.delete(url, id);
```

- Para hacer un DELETE pasando parámetro en la URL (no en el PATH)

```
String url = urlBase + "/animal?id="+id;
restTemplate.delete(url);
```

¿Cómo trataríamos las excepciones?

Como sabemos, nuestro servicio REST puede devolver excepciones con códigos de error HTTP. La forma de tratarlas sería:

1. Capturamos HttpStatusCodeException
2. Miramos qué código de error nos ha llegado
3. Actuamos en consecuencia. Por ejemplo, lanzando una excepción nuestra "entendible" para quien vaya a utilizar el cliente REST

Ejemplo:

Cuando invocamos al GET de nuestro servicio REST de animales, puede devolvernos un 404 si el animal con ese ID no existe. ¿Cómo tratamos esa excepción en el cliente?

```
public Animal getAnimal(Long id) throws AnimalNotFoundException {
    try {
        String url = urlBase + "/animal/{id}";
        Animal respuesta = restTemplate.getForObject(url, Animal.class, id);
        return respuesta;
    }
    catch(HttpStatusCodeException e) {
        if (e.getStatusCode() == HttpStatus.NOT_FOUND) {
            throw new AnimalNotFoundException("No existe animal con id " + id);
        }
        throw e;
    }
}
```

1. Capturamos HttpStatusCodeException
2. Miramos si el código de error es 404
Si es así, lanzamos una excepción nuestra más amigable
Si no, entonces será un error que desconocemos, y lo lanzamos de nuevo tal cual sin hacer nada con él.