Colecciones y API - Parte II

Introducción a APIs

Objetivos

- Conocer el concepto de request y response.
- Conocer la importancia de las API Rest para comunicar programas por internet.
- Utilizar Postman para realizar un request a una API.

Introducción a API

Una API es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Esta tecnología permite que aplicaciones escritas en distintos lenguajes se integren y se ejecuten en plataformas diferentes. Desarrollar con este estilo de arquitectura, otorga una manera sencilla de exponer servicios web y de entender cómo funcionan. API, acrónimo de Application Program Interface, es una interfaz de acceso que nos permite comunicar programas.

Existen diversos tipos de APIs, Particularmente hablaremos de una en especial llamada API REST y cuando utilicemos el concepto de API de ahora en adelante nos estamos refiriendo a una API REST.

API Rest

En términos muy simplificados una API REST es un programa para otro programa que sigue una serie de restricciones para estandarizar la interconexión (interfaz). Es decir, es un programa fácil de ser consultado desde cualquier programa. Esto es muy útil para integrar sistemas y comunicar distintas aplicaciones.

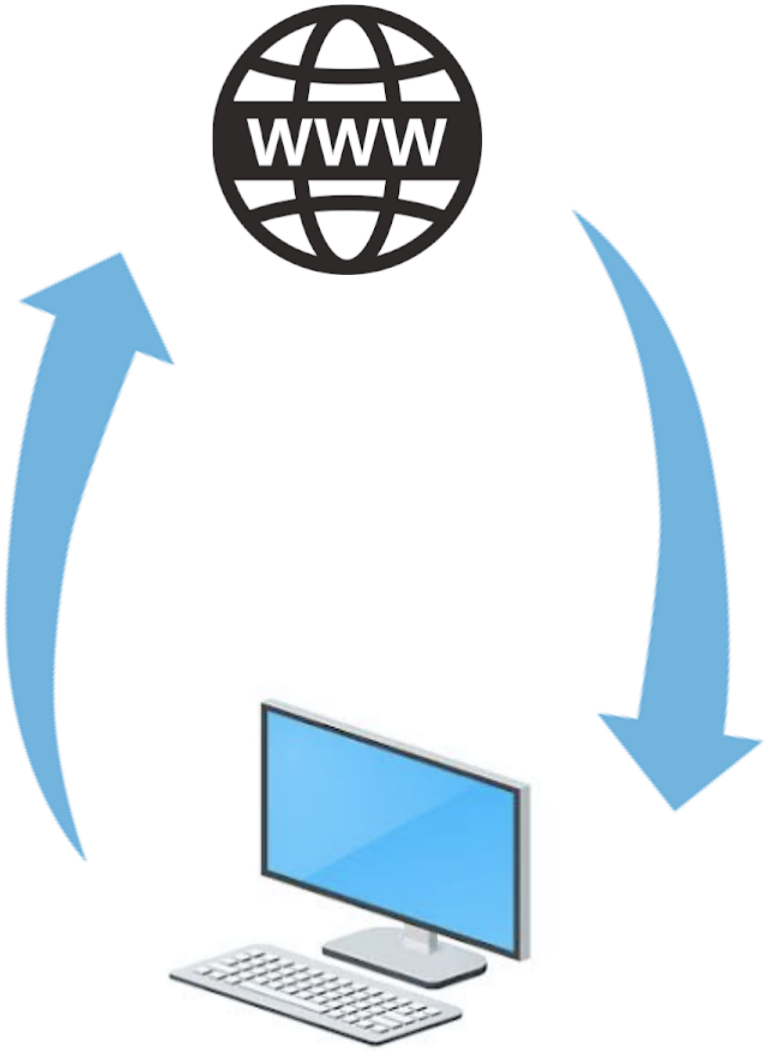

Imagen 1. Diagrama de conexión

Al programa que consulta se le denomina cliente. Al programa que entrega la respuesta se le suele llamar servidor.

Ejemplos de APIs existentes

Existen miles de APIs para distintos propósitos.

- Clima y temperatura.
- Cambio de monedas.
- Indicadores económicos.
- Servicios para subir archivos.
- Compra y venta de criptomonedas.
- Servicios de geolocalización como googlemaps.
- Y muchas más.

Cualquier persona puede construir una API y disponibilizarla a través de internet.

¿Cómo se usa una API?

Muy sencillo, a través de un cliente hacemos un request (solicitud) a una dirección y obtendremos como resultado un response (respuesta).

Un request (solicitud) es información que nosotros enviamos a un servidor. Para ser mas precisos desde ahora en adelante diremos un request a una URL(Uniform Resource Locator) y Response a la información que nos devuelve el servidor o servicio.

Aprenderemos a hacer el pedido y analizar la respuesta.

Otro concepto frecuente es el de Endpoint, que es lo mismo que una URL a un recurso.

En otras palabras, son los terminales que expone una API.

Usar los métodos HTTP correctos

Para indicar el tipo de operación que se llevará a cabo, existen métodos HTTP para cada caso, y de esta forma no tener verbos en las rutas. Es importante que el uso de GET es solo para realizar consultas, no modifica estados ni realiza cambios.

- GET : Se utiliza para obtener un recurso o una colección de recursos
- POST : Se utiliza para crear un recurso o una colección de recursos
- PUT : Se utiliza para actualizar un recurso o una colección de recursos
- DELETE : Se utiliza para eliminar un recurso o una colección de recursos

Probando una API

Existe una herramienta muy potente para probar APIs sin necesidad de programar y nos ayudará a comprender la idea. Esta Herramienta se llama Postman

Descargando Postman

Podemos descargar Postman desde la página oficial. https://www.getpostman.com

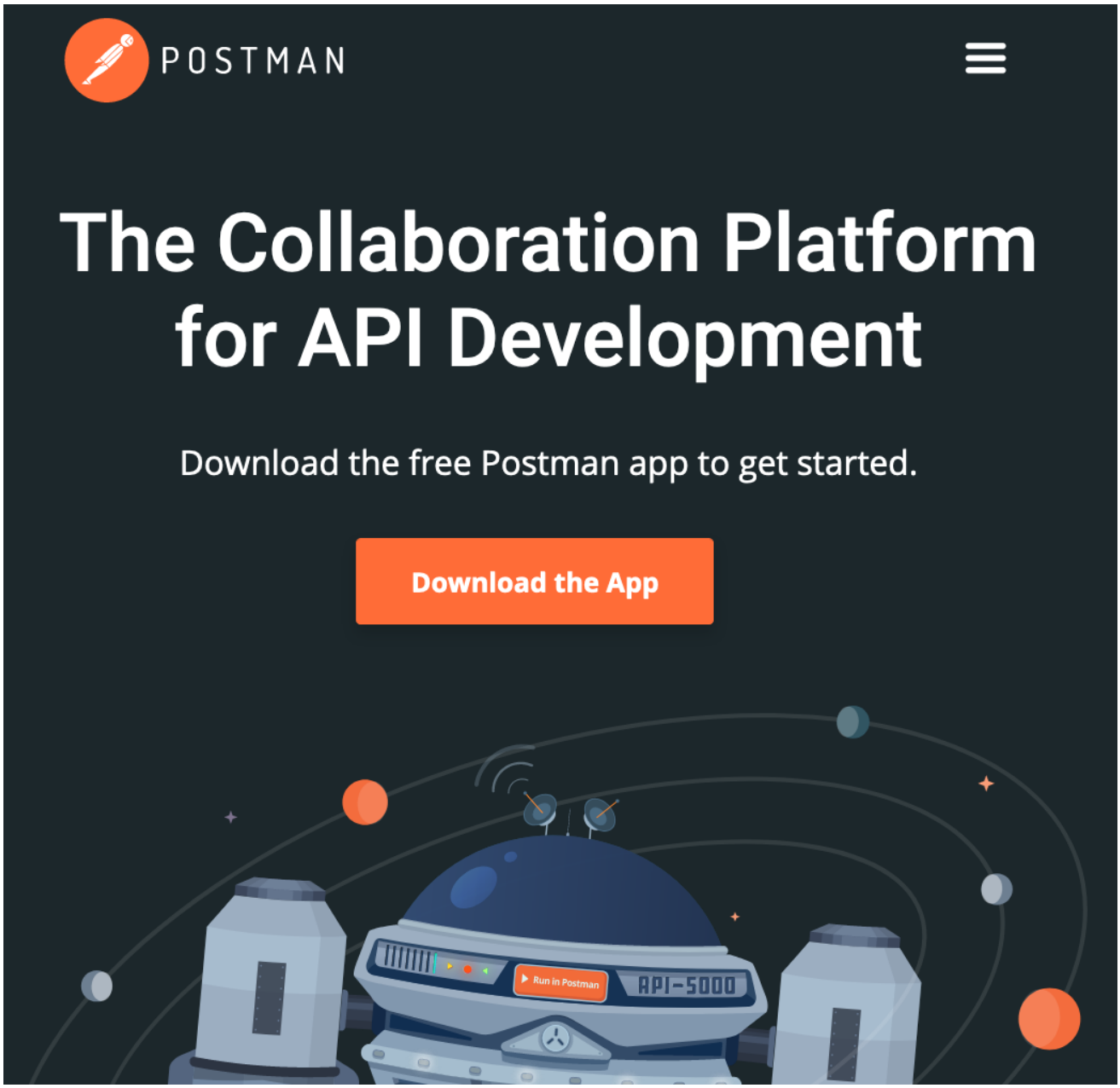

Imagen 2. Descargando Postman.

Se debe instalar y abrir la aplicación descargada, y se debe ver un panel como el siguiente.

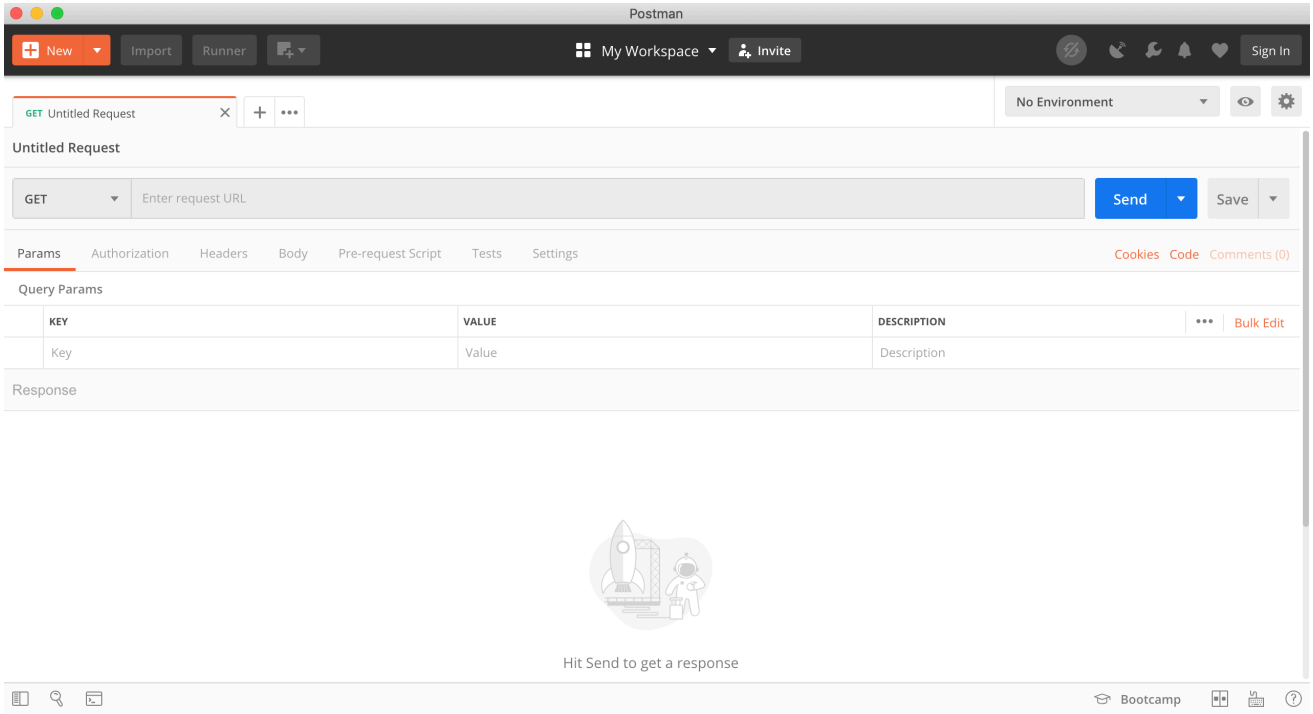

Imagen 3. Interfaz de Postman

Nuestra primera consulta a una API

Nuestra primera consulta será a una API llamada jsonplaceholder: https://jsonplaceholder.typicode.com/`

Es una API falsa en el sentido de que no tiene información real, pero es una API en todo otro sentido.

Primer request

Donde dice Request URL pondremos la siguiente URL:

```
https://jsonplaceholder.typicode.com/posts
```

Lo que haremos será realizar una petición GET, para obtener un recurso, por lo tanto se debe definir que es una petición de tipo GET para luego hacer clic donde dice send y obtendremos nuestra respuesta. La cuál será una lista de publicaciones. Que mas adelante tendremos que asignar a una clase Java.

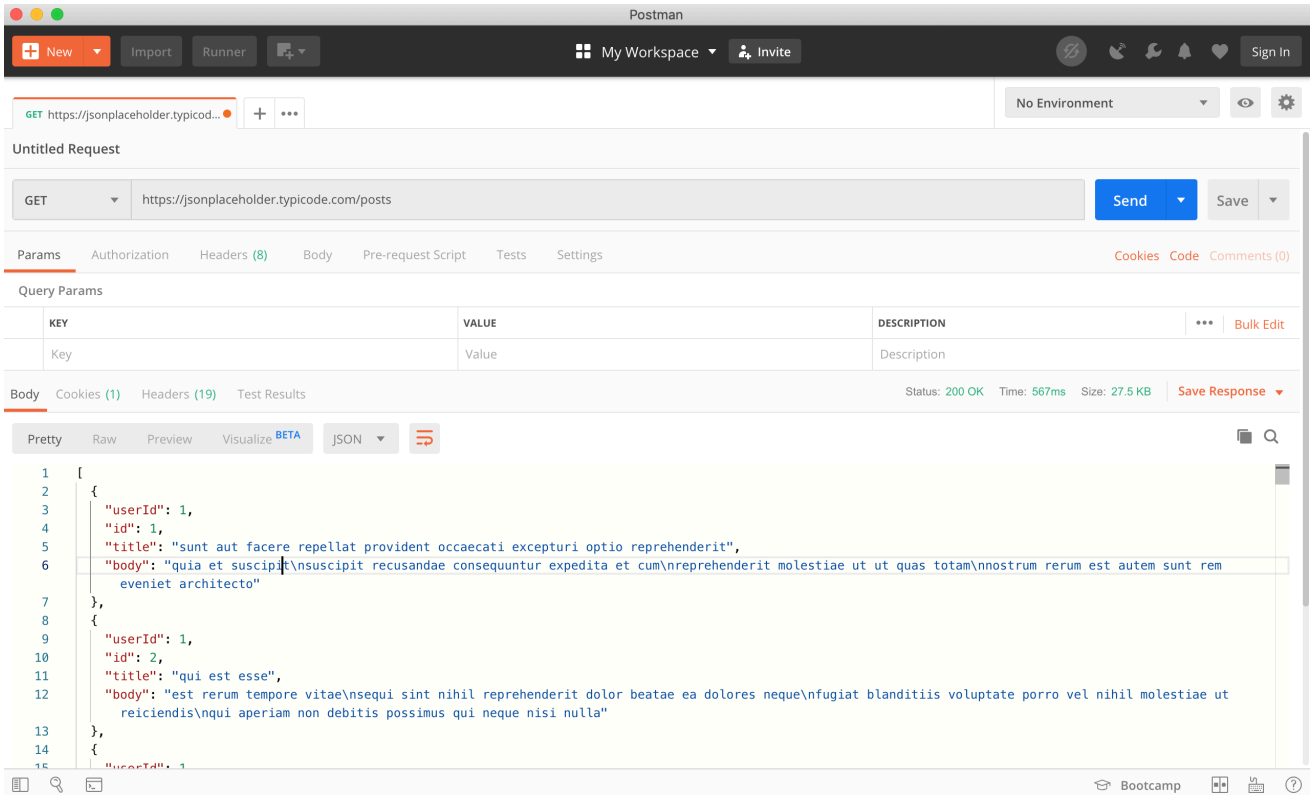

Imagen 4. Creando la primera request.

Luego de algunos milisegundos (o segundos en un mal caso) el servidor nos enviará una respuesta. Ese es el texto que aparece en la parte inferior de la imagen. La respuesta se parece mucho a estructuras que ya conovemos. Primero vemos la apertura de un arreglo y luego varios diccionarios. La respuesta está en un formato llamado JSON.

JSON

JSON es acrónimo de Javascript Object Notation. Es un formato para enviar información en texto plano, fácilmente legíble por humanos y fácilmente analizable por lenguajes de programación. Hoy en día es uno de los formato más utilizado para enviar información entre sistemas.

JSON no requiere de JavaScript

Que el acrónimo tenga la palabra Javascript no quiere decir que necesitemos saber javascript o utiliazar javascript. JSON es un formato plano que podemos utilizar desde cualquier lenguaje.

Consumiendo API desde Java

Introducción

Previamente, aprendimos a realizar un request a una URL utilizando Postman. A continuación aprenderemos a hacer el request directamente desde Java.

Objetivos

- Crear un programa para consumir API desde Java.
- Conocer y agregar dependencias a Maven.
- Utilizar Jersey como librería para desarrollar el servicio.

Gestores de ciclo de vida de la aplicación

Son sistemas de automatización de construcción, que tienen el objetivo de simplificar los procesos de build (limpiar, compilar y generar ejecutables del proyecto a partir del código fuente). Se abordará Apache Maven, Maven es una herramienta de gestión y comprensión de proyectos de software. Basándose en el concepto de un modelo de objeto de proyecto (POM), Maven puede gestionar la compilación, los informes y la documentación de un proyecto a partir de una información central.

El archivo pom.xml es el núcleo de la configuración de un proyecto en Maven. Es un archivo de configuración único que contiene la mayoría de la información necesaria para construir un proyecto de la forma que desee. El POM es enorme y puede ser desalentador en su complejidad, pero no es necesario entender todas las complejidades para usarlo de manera efectiva. Dentro del archivo pom.xml se pueden agregar dependencias (librerías externas) en formato xml, que Maven se encargará de añadir al proyecto y configurar por nosotros.

Para utilizar Maven, basta con instalarlo desde su pagina oficial, para distintos sistemas operativos.

Crear proyecto

A continuación, se necesita crear un nuevo proyecto Java, con la capacidad de extenderlo mediante dependencias externas, para ello se debe crear un nuevo proyecto Java con Maven, ejecutar el siguiente comando por consola, esto genera la estructura de carpetas en base a un artifact que no es mas que un template.

Los parametros que se necesita setear son los siguientes:

- **DgroupId**: Debe contener el nombre del dominio, el cual será el nombre de su paquete.
- **DartifactId**: Debe contener el nombre de su aplicación.
- **DarchetypeArtifactId**: Se pasa como valor el nombre de un "template" ya definido por Maven, en este caso maven-archetype-quickstart.
- **DarchetypeVersion**: Se agrega la version del template.

```
mvn archetype:generate -DgroupId=cl.desafiolatam -DartifactId=gs-consume-api -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```

El proyecto generado contiene esta estructura de carpetas.

```
gs-consume-api
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── cl
    │           └── desafiolatam
    │               └── App.java
    └── test
        └── java
            └── cl
                └── desafiolatam
                    └── AppTest.java
```

Se debe importar el proyecto usando, su editor de texto o IDE favorito.

Agregando dependencias al proyecto

Se debe agregar las dependencias de Jersey, la cual es una librería útil para consumir servicios REST, y para agregarlas al proyecto, es suficiente agregar lo siguiente dentro del archivo pom.xml:

```
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-client</artifactId>
  <version>2.29.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.inject</groupId>
  <artifactId>jersey-hk2</artifactId>
  <version>2.29.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.media</groupId>
  <artifactId>jersey-media-json-jackson</artifactId>
  <version>2.29.1</version>
</dependency>
```

Crear clase para representar los datos

Los datos son post con las propiedades userId, id, title y body. Se debe crear una clase para contener y mostrar estos datos.

```
package cl.desafiolatam;

public class Publicacion {
    private Integer userId;
    private Integer id;
    private String title;
    private String body;
    //getters y setters
}
```

Creando una instancia de un cliente

Lo primero que se necesita es crear una instancia de un cliente, eso se hará en la clase App.

```
Client client = ClientBuilder.newClient();
```

Creando un objetivo

Una vez que se tiene la instancia del Cliente, se puede crear un WebTarget utilizando el URI del recurso web objetivo, en este caso es la ruta: `https://jsonplaceholder.typicode.com/posts`. De la cual obtendremos las publicaciones (posts).

```
WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("posts");
```

Crear una petición HTTP

Se crea un generador de invocación de instancias con uno de los métodos de Jersey llamado WebTarget.request():

```
Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
```

Realizando un GET

Invocando GET, para consumir API. Método GET usado para obtener un recurso.

```
Response respuestaPublicaciones = invocationBuilder.get();
```

Para leer la respuesta y asignarla a una propiedad, se debe llamar al método readEntity.

```
List<Publicacion> listaPublicaciones = respuestaPublicaciones.readEntity(new
GenericType<List<Publicacion>>(){});
```

Se puede imprimir en pantalla el resultado, el método main completo luce así:

```
public static void main( String[] args ) {
    Client client = ClientBuilder.newClient();
    WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("posts");
    Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
    Response respuestaPublicaciones = invocationBuilder.get();
    List<Publicacion> listaPublicaciones =
respuestaPublicaciones.readEntity(new GenericType<List<Publicacion>>(){});
    System.out.println(listaPublicaciones.get(0));
}
```

El código de la respuesta

En caso de haber obtenido exitosamente la respuesta el código será 200. Existen varios códigos, no es necesario saberlos de memoria, podemos encontrarlos de forma muy rápida en internet.

Algunos de los códigos de respuesta mas relevantes son:

- 200: OK
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 500: Server error

Analizando la respuesta

Analizando código de respuesta

Al invocar get() desde invocationBuilder, se devuelve un objeto que contiene elementos claves dentro de la respuesta, el código de la respuesta y el cuerpo (readEntity). A continuación se obtiene el código de respuesta de la petición.

```
respuestaPublicaciones.getStatus();
```

Analizando los headers de la respuesta

Para acceder a los headers, los cuales contienen permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. Para acceder a los headers respuesta basta con realizar:

```
respuestaPublicaciones.getHeaders();
```

El contenido de la respuesta

El objeto response contiene la información que nos interesa. Esta información viene como una lista de Posts, la cual ya está mapeada y contenida dentro de una colección, lo que hace sencillo acceder y manipular los datos. En palabras sencillas, Jersey se encargó de mapear el objeto JSON a una estructura de datos previamente definida.

Trabajando con la respuesta

El resto del trabajo dependerá de como venga estructurada la respuesta. Por ejemplo en este caso tenemos una Lista, cada elemento se compone de un objeto Posts. La estructura como la siguiente:

```
[
  {
    userId=1,
    id=1,
    title=sunt aut facere repellat provident occaecati excepturi optio
reprehenderit,
    body=quia et suscipit
      suscipit recusandae consequuntur expedita et cum
      reprehenderit molestiae ut ut quas totam
      nostrum rerum est autem sunt rem eveniet architecto
  },
  {
    userId=1,
    id=2,
    title=qui est esse,
    body=est rerum tempore vitae
      sequi sint nihil reprehenderit dolor beatae ea dolores neque
      fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis
      qui aperiam non debitis possimus qui neque nisi nulla
  }
```

```
]
```

Accediendo a un elemento de la respuesta.

Podemos acceder a un elemento en particular de las respuestas utilizando el índice de la Lista, para acceder al primer elemento.

```
List<Publicacion> listaPublicaciones = respuestaPublicaciones.readEntity(new
GenericType<List<Publicacion>>(){});
System.out.println(listaPublicaciones.get(0));
```

Iterando los resultados

Podríamos mostrar todos los títulos iterando de los post iterando la lista, haciendo uso de .forEach u for.

```
listaPublicaciones.forEach(System.out::println);
```

Realizando un POST

Para crear una nueva Publicación, se debe postear al path "posts". Con el método POST utilizado para crear un recurso.

```
public static void main(String[] args) {
    Publicacion publicacion = new Publicacion();
    publicacion.setTitle("LoTR");
    publicacion.setBody("A ring");
    publicacion.setUserId(1);
    publicacion.setId(101);
    Client client = ClientBuilder.newClient();
    WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("posts");
    Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
    Response publicacionRespuesta =
invocationBuilder.post(Entity.entity(publicacion, MediaType.APPLICATION_JSON));
    System.out.println(publicacionRespuesta);
}
```

Analizando la respuesta

Para acceder a los datos de esta petición como los headers y el estado de la respuesta basta con llamar a los métodos getHeaders, getStatus y getStatusInfo:

```
publicacionRespuesta.getHeaders(); //detalle de los headers
publicacionRespuesta.getStatus(); //201
publicacionRespuesta.getStatusInfo(); //Created
```

Realizando un PUT

Para actualizar el post con id 1, debe ser pasado en el path. Método PUT utilizado para actualizar un recurso.

```
public static void main(String[] args) {
    Publicacion publicacion = new Publicacion();
    publicacion.setTitle("LoTR");
    publicacion.setBody("Three movies");
    publicacion.setUserId(1);
    publicacion.setId(101);
    Client client = ClientBuilder.newClient();
    WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("posts").path("1");
    Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
    Response publicacionRespuesta =
invocationBuilder.put(Entity.entity(publicacion, MediaType.APPLICATION_JSON));
    System.out.println(publicacionRespuesta);
}
```

Analizando la respuesta

Para acceder a los datos de esta petición como los headers y el estado de la respuesta basta con llamar a los métodos getHeaders, getStatus y getStatusInfo:

```
publicacionRespuesta.getHeaders(); //detalle de los headers
publicacionRespuesta.getStatus(); //201
publicacionRespuesta.getStatusInfo(); //Created
```

Realizando un DELETE

Para eliminar el post con id 1, debe ser pasado en el path. Método DELETE utilizado para eliminar un recurso.

```
public static void main(String[] args) {
    Client client = ClientBuilder.newClient();
    WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("posts").path("101")
;
    Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
    Response publicacionRespuesta = invocationBuilder.delete();
    System.out.println(publicacionRespuesta);
}
```

Analizando la respuesta

Para acceder a los datos de esta petición como los headers y el estado de la respuesta basta con llamar a los métodos getHeaders, getStatus y getStatusInfo:

```
publicacionRespuesta.getHeaders(); //detalle de los headers
publicacionRespuesta.getStatus(); //201
publicacionRespuesta.getStatusInfo(); //Created
```

Cierre

Palabras finales

Las API's están transformado los negocios y permitiendo la interconexión de muchos sistemas. Trabajar fluidamente con ellas es una habilidad muy útil para cualquier desarrollador en cualquier lenguaje o framework. Además, si complementamos con el manejo seguro de la información que estamos enviando mediante SSL y un adecuado manejo de credenciales estaremos creando aplicaciones robustas y preparadas para solucionar problemas en entornos productivos.

Temas interesantes que todavía nos falta aprender

Entender la arquitectura orientada a servicios siguiendo las restricciones REST nos permitirá hacer aplicaciones ordenadas y extensibles. Nos falta ahora aprender a desarrollar nuestras aplicaciones siguiendo estos principios y modelar nuestras soluciones con un enfoque en recursos y en acciones que haremos con ellos.