Colecciones y API (Parte II)

Introducción a API

Competencias

- Comprender el uso de API mediante los conceptos de request y response.
- Comprender la importancia de las API Rest para comunicar programas desde internet.

Introducción

La tecnología avanza día a día en nuestras vidas, nos sorprende con descubrimientos o creaciones nuevas, sin embargo, este avance no sería posible si no existiera un elemento clave como lo es Internet. Este último elemento nos facilita la conexión, la generación de contenido y el acceso a otras personas. Es por ello que en esta unidad veremos cómo adquirir desde Internet aquellos datos tan relevantes para la toma de decisiones en cualquier rubro. Usaremos funciones para obtener respuestas mediante algo denominado API y buscaremos cómo acceder a ella mediante lenguaje Java.

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. La palabra API es un acrónimo de “Application Program Interface”, es decir, una interfaz de acceso que nos permite comunicar programas.

Existen diversos tipos de APIs, pero en esta ocasión hablaremos de API REST y, de ahora en adelante cuando utilicemos el concepto “API”, nos estamos refiriendo al concepto de API REST.

API Rest

En términos muy simplificados la palabra REST proviene de "Representational State Transfer", es decir, es un estilo de arquitectura que busca ordenar mediante reglas la creación de un servicio web. Es un programa que sigue una serie de restricciones para estandarizar la interconexión (interfaz), fácil de ser consultado y muy útil para integrar sistemas o comunicar distintas aplicaciones.

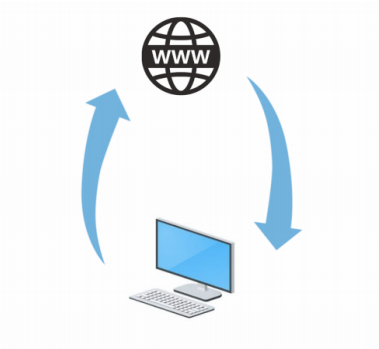


Imagen 1. Diagrama de conexión.
Fuente: Desafío Latam.

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 Google Maps, etc.

Inclusive, cualquier persona puede construir una API y disponibilizarla a través de Internet. Estas son las denominadas "Fake APIs".

¿Cómo se usa una API?

Para consumir o utilizar una API mediante un programa (cliente), debemos efectuar una acción llamada "request" o dicho de otra forma una solicitud hacia una dirección web específica (servidor). Esta solicitud nos traerá información de vuelta a nuestro programa en forma de respuesta o "response".

- Si la respuesta es correcta, nos desplegará la información que solicitamos.
- Si la respuesta no es correcta, nos dará un error que se desplegará en la consola.

Para ser más precisos, desde ahora en adelante llamaremos request a una dirección URL (Uniform Resource Locator) y response a la información que nos devuelve este servidor o servicio.

Otro concepto frecuente dentro del consumo de API es el denominado Endpoint. Esto es lo mismo que la parte final de la URL que buscamos como recurso. En otras palabras, son los terminales que expone una API.

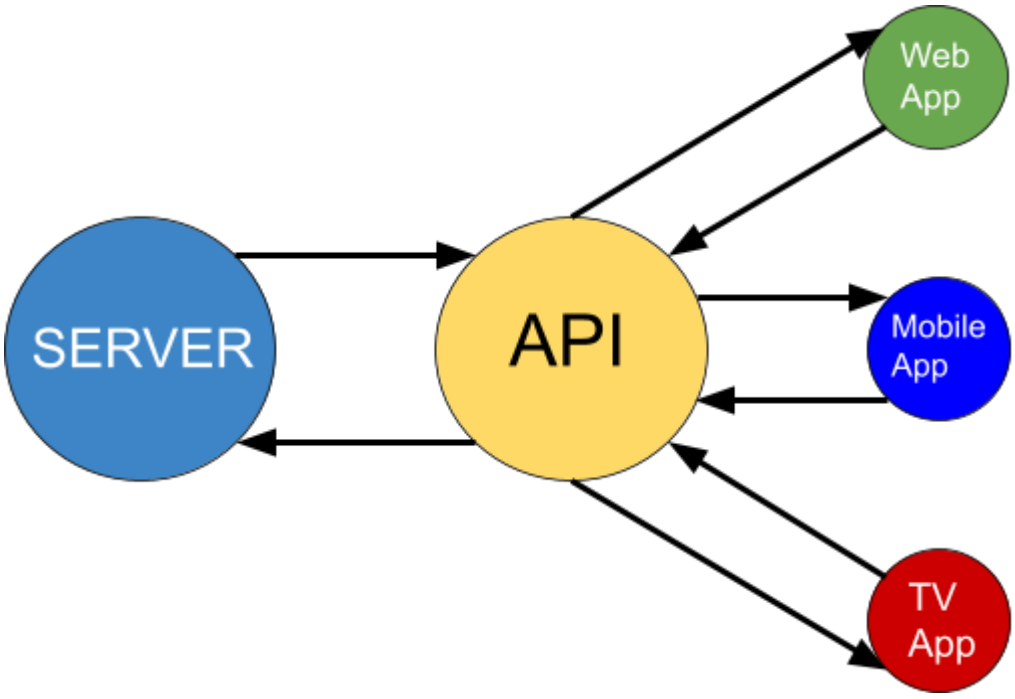


Imagen 2. Diagrama conexión API.
Fuente: Desafío Latam.

Usando los métodos HTTP correctos

Para indicar el tipo de operación que se llevará a cabo, existen diversos métodos HTTP. Estos buscan evitar la utilización de verbos en las rutas. Es importante destacar que el uso de GET es solamente para realizar consultas y no modifica estados ni realiza cambios.

Algunos de los métodos HTTP más usados son los siguientes:

- **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

Postman

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

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

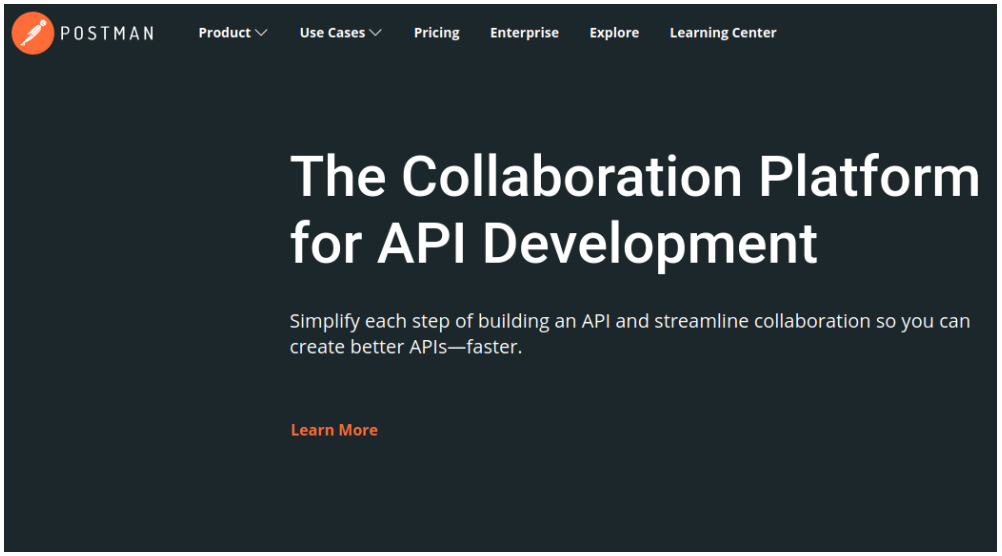


Imagen 3. Página oficial de Postman.
Fuente: Desafío Latam.

Es importante destacar que para instalar la aplicación debemos registrarnos en la página y luego buscar la opción de descarga, tal como se muestra en la imagen a continuación.

Imagen 4. Pantalla de descarga Postman.
Fuente: Desafío Latam.

Una vez instalada la aplicación, la abrimos y nos encontraremos con una interfaz semejante a la que se muestra en imagen.

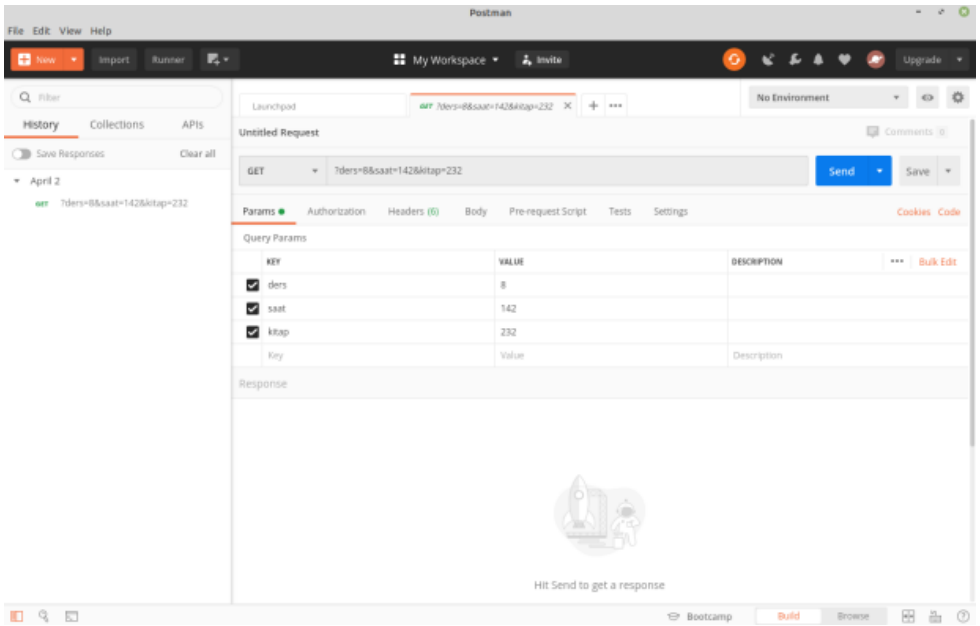


Imagen 5. Interfaz de Postman.
Fuente: Desafío Latam.

Nuestra primera consulta a una API

Para realizar nuestra primera consulta usaremos una API llamada jsonplaceholder:

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

Esta API contiene datos falsos y no tiene información real, pero nos sirve para darnos el primer acercamiento a este mundo. Funciona igual que el resto de las APIs y veremos un ejemplo a continuación.

Ejercicio Guiado: Primer Request

- **Paso 1:** Para comenzar a utilizar Postman, debemos colocar la siguiente URL en la casilla Request URL:

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

- **Paso 2:** Lo que haremos será realizar una petición GET para obtener un recurso, por lo tanto, se debe definir como tipo el método GET y luego hacer clic donde dice “send”. Obtendremos nuestra respuesta la cual será una lista de publicaciones.

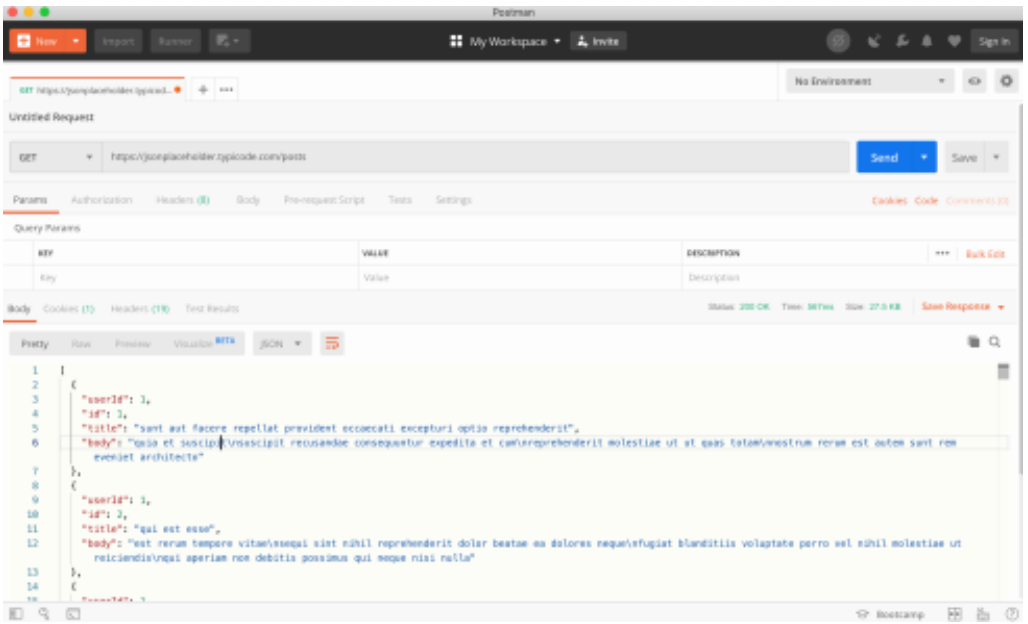


Imagen 6. Obteniendo URL.
Fuente: Desafío Latam.

- **Paso 3:** Luego de algunos milisegundos o segundos, el servidor nos enviará una respuesta en caso de ser exitosa. Aparecerá un texto en la parte inferior tal como se muestra en la imagen número 7.

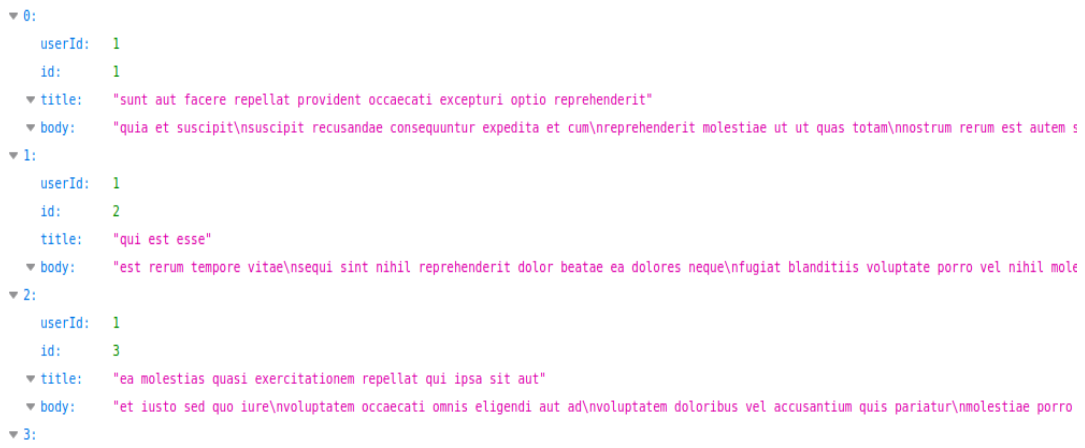


Imagen 7. Formato JSON visto desde Firefox.
Fuente: Desafío Latam.

La respuesta es el contenido que traemos directamente desde internet y se parece mucho a algunas estructuras que ya conocemos.

JSON

La respuesta está en un formato llamado **JSON** y se destaca por separar los objetos con las siguientes llaves { }, sin embargo es importante preguntarnos ¿Qué es JSON?

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

Que el acrónimo tenga la palabra Javascript no quiere decir que necesitemos saber del lenguaje Javascript o utilizar Javascript. El formato JSON es de tipo plano y lo podemos utilizar desde cualquier lenguaje.

Ejercicio Propuesto (1)

- Incorporar la siguiente URL en Postman y ver qué nos da como respuesta desde Internet.

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

Consumiendo API desde Java

Competencias

- Crear un programa para consumir API desde Java.
- Comprender dependencias Maven para utilizar Jersey como biblioteca.

Introducción

Previamente, aprendimos a realizar un request a una URL utilizando Postman. A continuación aprenderemos a hacer el request directamente desde una interfaz usando Java. Para esto definiremos algunos conceptos:

- Maven: es una herramienta de gestión y comprensión de proyectos de software, que se basa en el concepto de modelo objeto. El archivo pom.xml es el núcleo de la configuración de un proyecto en Maven.

- 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 (bibliotecas externas) en formato xml.

Ejercicio Guiado: Proyecto Posts

Paso 1: Se necesita crear un nuevo proyecto en Eclipse, con la capacidad de extenderlo mediante dependencias externas. Para ello crearemos un proyecto Java del tipo Maven de la siguiente manera.

```
File-> New -> Other..() -> Maven -> Maven Project
```

Paso 2: Agregamos las dependencias de Jersey (la cual es una librería útil para consumir servicios REST) y buscamos el archivo pom.xml para pegar lo siguiente en su interior.

```
<dependencies>
 <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>

 <dependency>
       <groupId>org.glassfish.jaxb</groupId>
       <artifactId>jaxb-runtime</artifactId>
       <version>2.3.2</version>
 </dependency>

 <dependency>
       <groupId>jakarta.xml.bind</groupId>
       <artifactId>jakarta.xml.bind-api</artifactId>
       <version>2.3.2</version>
 </dependency>
</dependencies>
```

Paso 3: Creamos la clase "Publicacion" para representar los datos provenientes desde la API. Los parámetros a definir al interior de la clase serán: **userId**, **id**, **title** y **body**. Además, debemos generar los getter and setter correspondientes y su toString().

```
package cl.desafiolatam;

      public class Publicacion {
          private Integer userId;
          private Integer id;
          private String title;
          private String body;
          //Generate getter, setter and toString()
          }
```

Paso 4: Para hacer el llamado de la API, nos dirigiremos a la clase Main y creamos una instancia de cliente.

```
            //Crear un cliente

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

Paso 5: 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

```
        //Cliente consume desde una API alguna información

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

Paso 6: Se crea un generador de invocación de instancias con uno de los métodos de Jersey llamado target.request().

```
        //WebTarget construye el Request

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

Paso 7: Invocamos GET para consumir API y obtener un recurso.

```
      //El builder tiene la información del request y le pedimos
ejecutar un get

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

Paso 8: Para leer la respuesta y asignarla a una propiedad, se debe llamar al método readEntity.

```
       //A la respuest le pedimos que lea la información

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

Paso 9: Se puede imprimir en pantalla el resultado de la posición cero o del listado completo.

```
       //Obtenemos todos los libros

       System.out.println(listaPublicaciones.get(0));

       --------------------------
```

```
       Impresión en pantalla:

       [Publicacion: [userId=1,id=1,title="sunt aut facere repellat
provident occaecati excepturi optio reprehenderit",body="quia et
suscipit\nsuscipit recusandae consequuntur expedita et
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem
sunt rem eveniet architecto"]]
```

En caso de haber obtenido exitosamente la respuesta del código será 200. Existen varios códigos, no es necesario saberlos de memoria y cada uno de ellos podemos encontrarlos de forma muy rápida en Internet.

Algunos de los códigos de respuesta más relevantes son:

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

Si deseas conocer todas las propiedades que acepta este objeto de configuración, revisa el documento - **Códigos de estado HTTP** ubicado en “Material Complementario”.

Métodos HTTP

Realizando un GET

Analizando código de respuesta

- Al invocar get() desde invocationBuilder, se devuelve un objeto que contiene elementos claves. Estos elementos son: un código de respuesta y el cuerpo o readEntity. A continuación se observa cómo se obtiene el código de respuesta de la petición.

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

Analizando los headers de la respuesta

- Los headers permiten enviar información entre el cliente y el servidor, junto a una petición o respuesta. Para acceder a los headers y su 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 cómo venga estructurada la respuesta. Por ejemplo, en este caso tenemos una Lista, cada elemento se compone de un objeto Posts. La estructura es 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, por ejemplo, para acceder al primer elemento podemos hacer lo siguiente.

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

Iterando en la respuesta

- Podríamos mostrar todos los títulos provenientes desde la API, iterando y haciendo uso de un método “forEach” o de un “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);
}
```

Realizando un PUT

Para actualizar el post con id 1, debemos pasarlo mediante el path. El método PUT es 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);
}
```

Realizando un DELETE

Para eliminar el post con id 1, debemos pasarlo en el path. El método DELETE puede ser 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(1);
     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(); //Código de respuesta “201”
publicacionRespuesta.getStatusInfo(); //Información “Created”
```

Ejercicio propuesto (2)

Obtener el listado completo de comments, dada la siguiente URL:

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

- Ver cual es el contenido que trae la API, ya sea por Postman o algún navegador.
- Crear la clase Comment con los parámetros vistos previamente.
- Obtener el listado de comments en el Main.

Palabras de cierre

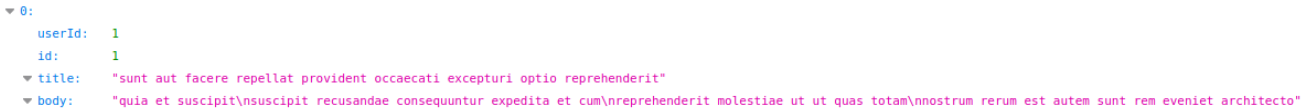
Hemos visto a través de este capítulo que existen diversas funciones para conectar nuestros programas con servidores de Internet y así obtener datos. Muchos sistemas de trabajo funcionan con APIs y esto les permite transformar sus negocios. Para cualquier desarrollador y en cualquier lenguaje o framework, trabajar con APIs es una habilidad muy útil para resolver problemas del día a día. Si a todo lo anterior le complementamos el manejo seguro de la información mediante SSL y un adecuado manejo de credenciales, estaremos creando aplicaciones robustas y preparadas para resolver necesidades.

Soluciones ejercicios propuestos

Solución Ejercicio Propuesto (1)

Contexto

1.- El contenido de la API nos debería arrojar la siguiente información



Solución Ejercicio Propuesto (2)

Contexto

1.- Crear la clase.

```
  public class Book {
      private Integer postId;
      private Integer id;
      private String name;
      private String email;
      private String body;
  //Efectuar los getter and Setter correspondientes, junto al toString()
}
```

2.-Obtener el listado de comments en el Main.

```
            //Crear un cliente
            Client client = ClientBuilder.newClient();
            //Cliente consume desde una API alguna información
            WebTarget target =
client.target("https://jsonplaceholder.typicode.com").path("comments");
            //WebTarget construye el Request
            Invocation.Builder invocationBuilder =
target.request(MediaType.APPLICATION_JSON);
            //Al builder, que es el que tiene la información del
request, le pedimos ejecutar un get
            Response respuestaPublicaciones = invocationBuilder.get();
            /A la respuest le pedimos que lea la información
            List<Comment> listaPublicaciones =
respuestaPublicaciones.readEntity(new
            GenericType<List<Comment>>() {});
            //Obtenemos todos los comments
            System.out.println(listaPublicaciones);
```