Consumiendo API desde Java

¡Comencemos!

¿Qué aprenderás?

- 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.

¡Vamos con todo!

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”
```

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.