

DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

AD
Acceso a Datos

UD 5
Programación de componentes
de acceso a datos
(Parte 3)

IES BALMIS
Dpto Informática
Curso 2022-2023
Versión 1 (11/2022)

UD5 – Programación de componentes de acceso a datos

ÍNDICE

9. Uso de API Rest

9.1 Introducción

9.2 Probar el funcionamiento de un servicio API Rest

9.3 Utilizar un servicio API Rest desde Java

9.4 Instalación de un API Rest con PHP

9. Uso de API Rest

9.1 Introducción

API (Application Programming Interface o Interfaz de Programación de Aplicaciones) es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca (librería de software) para ser utilizado por otro software como una capa de abstracción.

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP.

REST es un modelo de arquitectura web basado en el protocolo HTTP para mejorar las comunicaciones cliente-servidor que utiliza un conjunto de restricciones con las que podemos crear un estilo de arquitectura software.

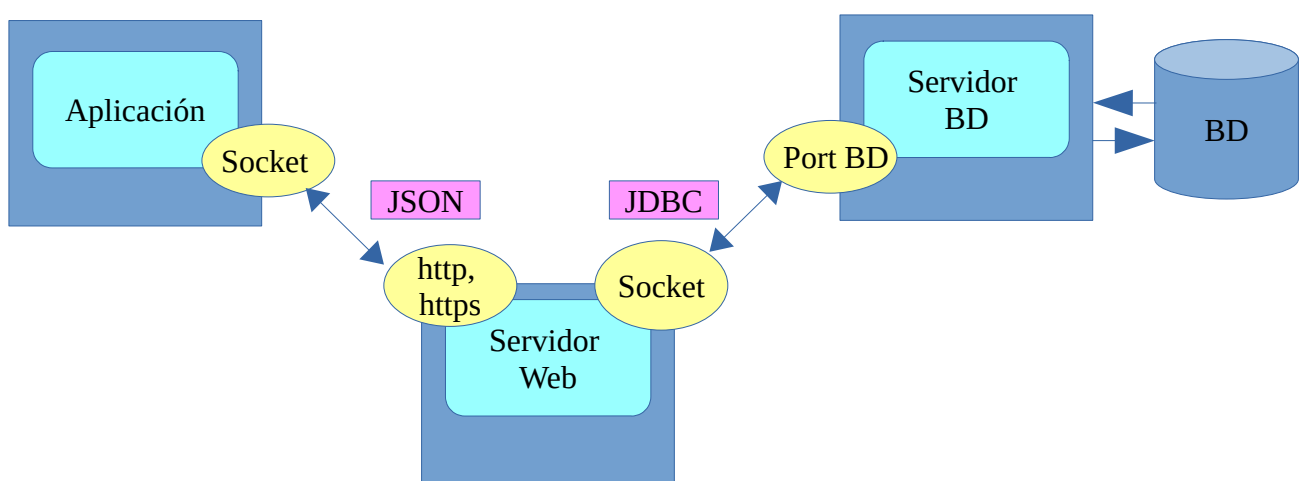
API Rest o Restful es un servicio web que implementa la arquitectura REST y que responde a diferentes peticiones utilizando generalmente un intercambio de datos en formato JSON o XML

Algunos frameworks con los que podremos implementar nuestras API Rest son:

- JAX-RS y Spring Boot para Java,
- Django REST framework para Python,
- Laravel para PHP o
- Restify para Node.js

Hoy en día la mayoría de las empresas utilizan API REST para crear servicios. Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web.

Cuando ofrecemos este servicio conectando con un servidor de BD tenemos el siguiente esquema:



Como vimos en la UD1, existen diferentes tecnologías para ofrecer datos usando conexión TCP como:

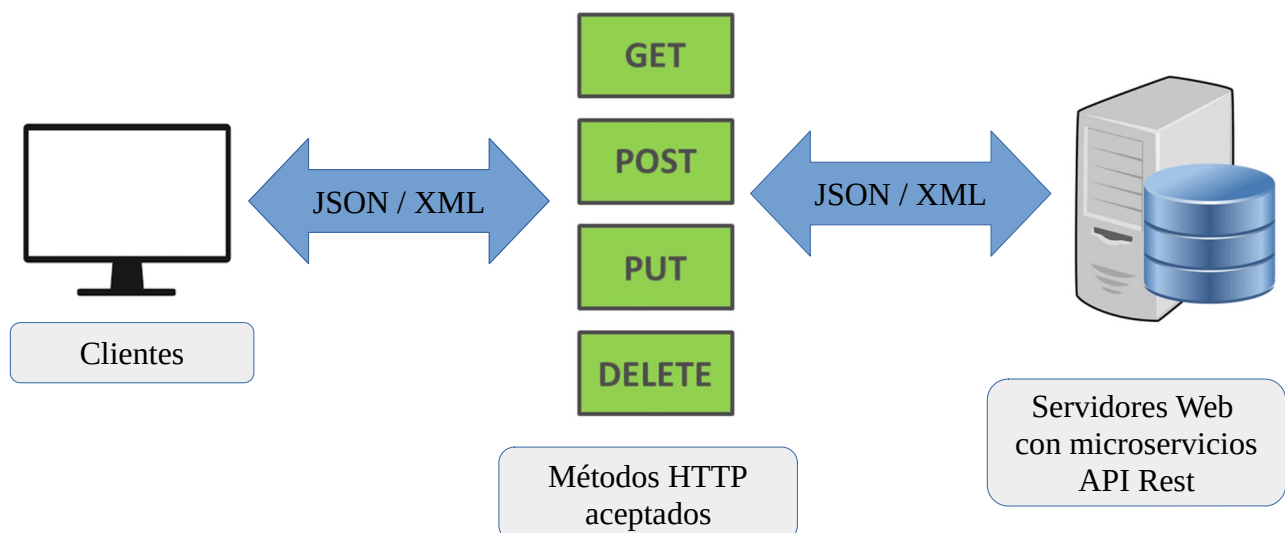
- SOAP (1998)
- API REST (2000)
- GraphQL (2015)

El concepto de **Open Data (Datos abiertos)** es una filosofía y práctica que persigue que determinados tipos de datos estén disponibles de forma libre para todo el mundo, sin restricciones de derechos de autor, de patentes o de otros mecanismos de control. Tiene una ética similar a otros movimientos y comunidades abiertos, como el software libre, el código abierto (open source) y el acceso libre (open access).

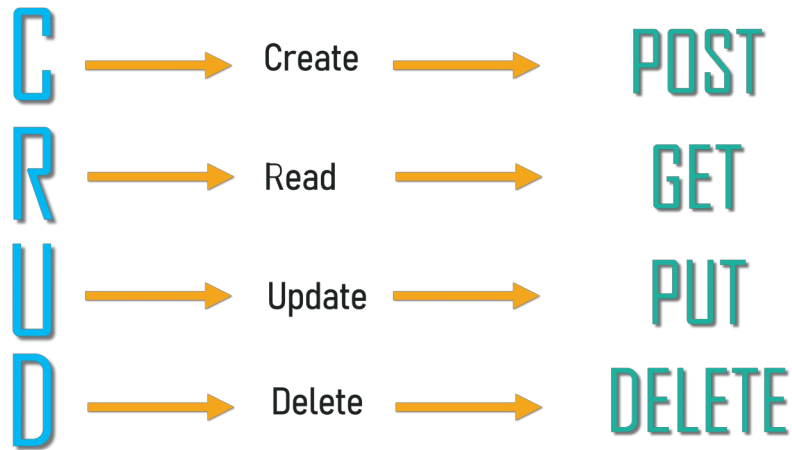
Hay muchos servicios de Open Data y la mayoría se ofrecen a través de servicios API Rest. Algunos ejemplos importantes:

- **OpenStreetMap** => <https://wiki.openstreetmap.org/wiki/API>
 - Ejemplo: <https://www.openstreetmap.org/api/0.6/way/81131976>
- **Aemet OpenData** => <https://opendata.aemet.es/centrodedescargas/inicio>
 - Especificación de métodos:
 - https://opendata.aemet.es/AEMET_OpenData_specification.json
 - Ejemplo de link en Aemet con JSON:
 - https://opendata.aemet.es/opendata/api/maestro/municipios/?api_key=XXX
- **Open Food** => <https://es.openfoodfacts.org/> y <https://es.openfoodfacts.org/data>
 - Ejemplo de link en Open Food con JSON con el código de barra de Nutela:
 - <https://es.openfoodfacts.org/producto/80135463/nutella-ferrero>
 - <https://world.openfoodfacts.org/api/v0/product/80135463.json>

Las operaciones más importantes que nos permitirán manipular los recursos son:.



Cada método se usa para realizar una acción en la base de datos:



Arquitectura API Rest

<https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html>

Algunos de estos servicios se ofrecen con un **API KEY**, es decir, se necesita registrarse para que el sistema conozca las peticiones de cada usuario. Esto no implica identificación de usuario y contraseña, sino control de peticiones de cada registro.

Algunos ejemplos

API Rest para practicar

- <https://jsonplaceholder.typicode.com>
- <https://reqres.in/>
- <https://gorest.co.in/>
- <https://www.mockable.io/>
- <https://designer.mocky.io/>
- http://httpbin.org/#/Request_inspection

API Rest Públicos (la mayoría solo GET)

- <https://openlibra.com/es/page/public-api>
- <https://datos.gob.es/es/apidata>
- <https://dog.ceo/dog-api/documentation/>
- <https://api.poems.one/>
- <https://es.openfoodfacts.org/data>
 - (<https://es.openfoodfacts.org/>)
- <https://jikan.moe/>
- <https://api.nasa.gov/> (con api_key=DEMO_KEY)
- <https://openweathermap.org/api>
- <https://api.mymemory.translated.net>
- <http://www.omdbapi.com/> (<https://es.omdb.org/>)

→ <https://www.youtube.com/watch?v=x0sqjAyIWwI>
 dog.ceo, jikan, jsonplaceholder,
 regres.in, omdb, openweathermap

Repositorios de API REST Públicos (la mayoría solo GET)

- <https://ichi.pro/es/una-lista-seleccionada-de-100-api-publicas-geniales-y-divertidas-para-inspirar-su-proximo-proyecto-8109114517939>
- <https://any-api.com/>

Herramientas gráficas para probar un API REST

- <https://es.sensedia.com/post/rest-api-understand-the-step-by-step-to-perform-tests>

Para probar podemos usar la utilidad **curl**. Puedes consultar algunos ejemplos prácticos de CURL en:

- <https://geekflare.com/es/curl-command-usage-with-example/>
- <https://terminalcheatsheet.com/es/guides/curl-rest-api>

Por ejemplo, abriendo una ventana de línea de comandos CMD:

```
C:\> curl -X GET https://jsonplaceholder.typicode.com/albums/2
C:\> curl https://jsonplaceholder.typicode.com/albums/2
{
  "userId": 1,
  "id": 2,
  "title": "sunt qui excepturi placeat culpa"
}
```

```
C:\> curl https://jsonplaceholder.typicode.com/users/2
{
  "id": 2,
  "name": "Ervin Howell",
  "username": "Antonette",
  "email": "Shanna@melissa.tv",
  "address": {
    "street": "Victor Plains",
    "suite": "Suite 879",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771",
    "geo": {
      "lat": "-43.9509",
      "lng": "-34.4618"
    }
  },
  "phone": "010-692-6593 x09125",
  "website": "anastasia.net",
  "company": {
    "name": "Deckow-Crist",
    "catchPhrase": "Proactive didactic contingency",
    "bs": "synergize scalable supply-chains"
  }
}
```

9.2 Probar el funcionamiento de un servicio API Rest

Entraremos en la URL Base del API Rest de ejemplo:

<http://riconet.es/fp/apirest>

Probar con el navegador el método GET

Sin instalar ningún plugin en el navegador, podemos probar el método **GET**. Basta con introducir la URL de los ejemplos:

<http://riconet.es/fp/apirest/datos>
<http://riconet.es/fp/apirest/libros>
<http://riconet.es/fp/apirest/libros/3>
<http://riconet.es/fp/apirest/libros/count>

El formato obtenido por defecto es HTML para la primera y JSON para el resto.

Probar API Rest con Postman

Para probar un API Rest, lo ideal es utilizar una aplicación específica que nos permita parametrizar todos los datos necesarios y **Postman** es una de las más utilizadas.

Postman – Web oficial

<https://www.postman.com/downloads/>

Generalmente la información que vamos a utilizar en el envío son:

Método	Aunque hay más, se suelen usar GET, POST, PUT y DELETE
URL Base	Es la parte inicial de la URL de tipo http que contiene hasta la carpeta donde instalamos nuestro API Rest
URL RewriteRule	Es la parte final de la URL que se reescribe y procesa para procesar la petición (REQUEST). A este proceso también se le llama "URL Amigable".
Datos de QUERY	Es la parte de la URL posterior al carácter '?' donde se añaden parejas de parámetro=valor separadas por el carácter '&'
Content-Type	Es el tipo de los datos de envío. En API Rest se suele usar json o xml y se indica con el valor " application/json " y " application/xml " respectivamente
Datos de envío	Es una cadena de caracteres en el formato indicado por " Content-Type "
Accept	Es el formato en el se desea que el servidor nos devuelva los datos

La respuesta contendrá al menos la siguiente información:

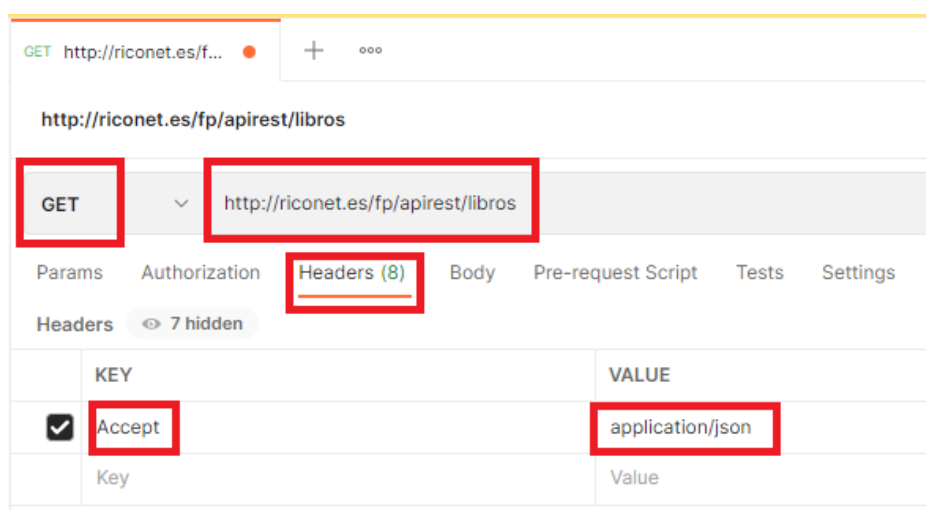
Datos de respuesta	Es una cadena de caracteres en el formato indicado por " Accept " en la petición (REQUEST)
Response	<p>Código de estado de la respuesta (HTTP Status Code).</p> <p>Es un número entero de 3 dígitos donde el primer dígito del Código de estado define la clase de respuesta y los dos últimos dígitos el tipo de respuesta</p> <p>https://www.tutorialspoint.com/http/http_status_codes.htm</p>

Veamos el formato general:

http://dominio/pathApiRest/rewriteRule?queryString

Probaremos algunos ejemplos con **Postman** para comprobar el funcionamiento:

Obtener en JSON todos los registros de la tabla libros	
Método	GET
URL	http://riconet.es/fp/apirest/libros
Headers => Content-Type	
Datos de envío	
Headers => Accept	application/json



Obtener en XML el registro con id=3 de la tabla libros	
Método	GET
URL	http://riconet.es/fp/apirest/libros/3
Headers => Content-Type	
Datos de envío	
Headers => Accept	application/xml

Insertar en la tabla libros el equivalente a: INSERT INTO libros (id, titulo, autor) VALUES (0, 'El instituto', 'Stephen King'); enviando los datos en JSON y recibiendo la respuesta en XML	
Método	POST
URL	http://riconet.es/fp/apirest/libros
Headers => Content-Type	application/json
Datos de envío	Body => Raw => JSON <pre>{ "id":0, "titulo":"El instituto", "autor":"Stephen King" }</pre>
Headers => Accept	application/xml

POST <http://riconet.es/fp/apirest/libros>

Method: POST, URL: <http://riconet.es/fp/apirest/libros>

Headers (10):

KEY	VALUE
<input checked="" type="checkbox"/> Accept	application/xml
<input checked="" type="checkbox"/> Content-Type	application/json

Body (11):

```
raw JSON
1 {
2   "id":0,
3   "titulo":"El instituto",
4   "autor":"Stephen King"
5 }
```

Se puede consultar la información de especificación completa del API Rest de ejemplo pulsando en el botón de "Descargar PDF":

<http://riconet.es/fp/apirest>

9.3 Utilizar un servicio API Rest desde Java

Para poder usar un servicio de API Rest desde Java usaremos JAX-RS.

Tenemos varias opciones de proyecto:

- 1) Usando las **librerías de Netbeans**
- 2) Usando **Maven**
- 3) Usando los **jar** descargados y copiados en una carpeta **lib**

Opción 1) Con Netbeans tendremos que usar dos librerías:

JAX-RS 2.0	Es la librería principal que nos permite conectar con el servicio de API Rest
Jersey 2.5.1 (Jax-RS RI)	Es una librería necesaria (dependency) para que funcione JAX-RS

Es posible que Netbeans no tenga todas las librerías JAR necesarias y por lo tanto muestre algún error.

Opción 2) Con Maven tendremos que indicar la dependencia:

```
org.glassfish.jersey.bundles : jaxrs-ri  
2.35 [zip] - central
```

Otra posibilidad es añadir la dependencia en el archivo pom.xml:

```
<dependency>  
  <groupId>org.glassfish.jersey.bundles</groupId>  
  <artifactId>jaxrs-ri</artifactId>  
  <version>2.35</version>  
  <type>zip</type>  
</dependency>
```

También podríamos usar la nueva versión de JAXRS 3.0.3 pero nos cambiaría todos los import de **javax.ws.rs.*** a **jakarta.ws.rs.*** pero el código de java sería el mismo.

Opción 3) La **opción recomendada** es usando los **jar** descargados y copiados en una carpeta **lib**. Estos archivos los proporciona el profesor y nos permite tener un proyecto sin depender de la instalación de Netbeans y ni de Internet.

El proyecto a crear usará un objeto **ClientBuilder** de JAX-RS que podrá conectar y obtener respuesta (REQUEST) de un servicio API Rest.

API Rest – Clases

<https://javaee.github.io/javaee-spec/javadocs/javax/ws/rs/client/Client.html>
<https://javaee.github.io/javaee-spec/javadocs/javax/ws/rs/client/WebTarget.html>
<https://javaee.github.io/javaee-spec/javadocs/javax/ws/rs/client/Invocation.Builder.html>
<https://javaee.github.io/javaee-spec/javadocs/javax/ws/rs/core/Response.html>

Probaremos a crear un pequeño proyecto que conecte con nuestro ejemplo de API Rest para obtener los datos en **JSON** de **libro con id='4'**.

ApiRest01Get

```
// Objetos para realizar la petición
Client client = ClientBuilder.newClient();

WebTarget webTargetBase;
WebTarget webTargetSolicitud;
Invocation.Builder invbuilder;
Response response;

// Indicar la URL del API Rest
webTargetBase = client.target("http://riconet.es/fp/apirest");
webTargetSolicitud = webTargetBase.path("/libros/4");

// Aportar los formatos y datos a la llamada
invbuilder = webTargetSolicitud.request();
invbuilder.header("Content-Type", "application/json");
invbuilder.accept(MediaType.APPLICATION_JSON);

// Ejecutar el método
response = invbuilder.get();
```

Los métodos ejecutados son:

target	Permite asignar la URL Base del servicio API Rest
path	Es la parte de la URL que permite parametrizar el servicio requerido
request	Crea un objeto para invocar el método
header	Asigna variables de cabecera como "Content-Type" para indicar el formato de los datos enviados
accept	Indicar el formato de los datos que vamos a recibir
get	Invoca el método

Una vez ejecutado obtenemos un objeto Response que nos proporciona es código de estado (response status code) y los datos de la respuesta.

Los códigos de estado están tipificados en la especificación de API Rest

API Rest – Status Code

<https://restfulapi.net/http-status-codes/>

Para mostrar el resultado probaremos con el siguiente código:

```
// Mostrar la respuesta obtenida
System.out.println("Status:");
System.out.println("=====");
System.out.println(response.getStatus());
System.out.println();

System.out.println("Data Body:");
System.out.println("=====");
System.out.println(response.readEntity(String.class));
System.out.println();
```

Para crear un código más simple, podemos encadenar los métodos hasta obtener el objeto **response**:

ApiRest02Get

```
// Objetos para realizar la petición
Client client = ClientBuilder.newClient();

Response response = client
    .target("http://riconet.es/fp/apirest")
    .path("/libros/4")
    .request()
    .header("Content-Type", "application/json")
    .accept(MediaType.APPLICATION_JSON)
    .get();

// Mostrar la respuesta obtenida
System.out.println("Status:");
System.out.println("=====");
System.out.println(response.getStatus());
System.out.println();

System.out.println("Data Body:");
System.out.println("=====");
System.out.println(response.readEntity(String.class));
System.out.println();
```

Para recibir los datos en XML bastará con cambiar el parámetro en el método **accept**:

ApiRest03GetXML

```
...
Response response = client
    .target("http://riconet.es/fp/apirest")
    .path("/libros/4")
    .request()
    .header("Content-Type", "application/json")
    .accept(MediaType.APPLICATION_XML)
    .get();
...
```

Si queremos **añadir** un nuevo libro, necesitaremos enviar los datos del nuevo libro al servidor.

Como hemos visto anteriormente con Postman, podemos enviar los datos en varios formatos:

- raw => JSON (application/json)
- raw => XML (application/xml)
- raw => TEXT (x-www-form-urlencoded)

Para nuestro proyecto de insertar un libro, calcularemos el ID obteniendo el número de libros y le sumándole uno. El resto de campos los pedimos al usuario.

ApiRest04Post

```
// DECLARACIÓN DE VARIABLES
String body;
JsonReader jsonReader;
JsonObject json;

int id;
String titulo;
String autor;

Scanner teclado = new Scanner(System.in);

Response responsePost;

// PROCESO DE POST
Client client = ClientBuilder.newClient();

// *****
// DATOS DEL LIBRO
System.out.print("Introduzca título: ");
titulo = teclado.nextLine();

System.out.print("Introduzca autor: ");
autor = teclado.nextLine();
```

```

// *****
// JSON DEL LIBRO
String libro = "{ 'id': 0, 'titulo':'" + titulo + "'," +
                "'autor':'"+autor+"' }";
libro = libro.replaceAll("'", "\\");
System.out.println(libro);

// *****
// POST DEL LIBRO
responsePost = client.target("http://riconet.es/fp/apirest")
                    .path("/libros/")
                    .request()
                    .header("Content-Type", "application/json")
                    .accept(MediaType.APPLICATION_JSON)
                    .post(Entity.json(libro));
if (responsePost.getStatus() == 201){
    body = responsePost.readEntity(String.class);
    // System.out.println("Response: " + body);

    jsonReader = Json.createReader(new StringReader(body));
    json = jsonReader.readObject();
    System.out.println(json.getString("mensaje"));
} else {
    body = responsePost.readEntity(String.class);
    // System.out.println("Response: " + body);

    jsonReader = Json.createReader(new StringReader(body));
    json = jsonReader.readObject();
    if (json.containsKey("mensaje")) {
        System.out.println("ERROR: "+json.getString("mensaje"));
    }
    if (json.containsKey("sqlError")) {
        System.out.println(json.getString("sqlError"));
    }
}
}

```

También podríamos hacer un proyecto que obtuviera el número de registro con `/libros/count` y sumar uno para obtener el id. El problema es que si eliminamos un registro podríamos obtener un id que ya exista.

Si los datos los queremos enviar en formato XML deberíamos cambiar la llamada al Post:

ApiRest05PostXML

```
...
// *****
// XML DEL LIBRO
String libro = "<?xml version='1.0' encoding='UTF-8' standalone='yes'?>\n"+
               "<libro>\n" +
               "    <id>"+id+"</id>\n" +
               "    <titulo>"+titulo+"</titulo>\n" +
               "    <autor>"+autor+"</autor>\n" +
               "</libro> ";
libro = libro.replaceAll("'", "\\'");
System.out.println(libro);

// *****
// POST DEL LIBRO
responsePost = client.target("http://riconet.es/fp/apirest")
                    .path("/libros")
                    .request()
                    .header("Content-Type", "application/xml")
                    .accept(MediaType.APPLICATION_JSON)
                    .post(Entity.xml(libro));
...
```

Al usar los servicios de API Rest también podemos realizar **Mapping** a objetos.

En nuestro ejemplo, debemos crear la clase **Libros** y utilizarla al leer del objeto **response**.

Por ejemplo, partiendo del ejemplo **ApiRest02Get** tendríamos que indicarle a **readEntity** el tipo de Mapeo que queremos en vez de un **String.class**:

ApiRest06GetArray

```
...
// Mostrar la respuesta obtenida
System.out.println("Status:");
System.out.println("=====");
System.out.println(response.getStatus());
System.out.println();

System.out.println("Data Body:");
System.out.println("=====");

ArrayList<Libros> listaLib;
listaLib = response.readEntity(new GenericType<ArrayList<Libros>>() {});
for (Libros lib : listaLib) {
    System.out.println(lib);
}
System.out.println();
...
```

A la hora de realizar un Post, también cambiaríamos el **Entity**. Partiendo del ejemplo **ApiRest04Post** se podría cambiar creando el objeto libro en vez de el JSON:

ApiRest07PostEntity

```
...
// *****
// OBJETO DEL LIBRO
Libros libro = new Libros(id, titulo, autor);
System.out.println(libro);

// *****
// POST DEL LIBRO
responsePost = client.target("http://riconet.es/fp/apirest")
    .path("/libros/")
    .request()
    .header("Content-Type", "application/json")
    .accept(MediaType.APPLICATION_JSON)
    .post(Entity.entity(libro, MediaType.APPLICATION_JSON));
...
```

Tanto **put** como **delete**, son métodos más sencillos de usar que get y post, ya que son variantes.

ANEXO – CONEXIÓN CON PROXY

Para poder utilizar una URL de internet utilizando el proxy, bastará con cambiar el ClientBuilder

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

por otro con configuración:

```
ClientConfig config = new ClientConfig();
config.property(ClientProperties.PROXY_URI, "10.100.0.1:8080");
Client client;
client = ClientBuilder.newClient(config);
```

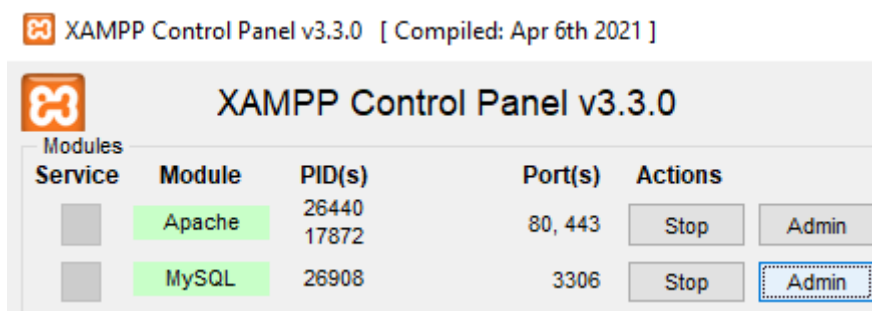

9.4 Instalación de un API Rest con PHP

El profesor entregará el código desarrollado de un **API Rest en PHP** que funciona son un servidor web **Apache + PHP** como el incluido en **XAMPP**.

Este código analiza la base de datos con la que se conecta y muestra un sencillo **API** para acceder a la información de cada tabla de forma independiente, es decir, sin relaciones. Para la **instalación** seguiremos los siguientes pasos:

Paso 1) Crear la BD en MariaDB con phpMyAdmin

En nuestro ejemplo, deberemos iniciar los servicios con **xampp-control.exe**



y crear la BD **bibliotecah2** ejecutando las siguientes instrucciones con **phpMyAdmin**:

```
CREATE SCHEMA bibliotecah2 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;

USE bibliotecah2;

CREATE TABLE autores (
  cod      VARCHAR(5) PRIMARY KEY,
  nombre   VARCHAR(60)
);

CREATE TABLE libros (
  id        INTEGER PRIMARY KEY AUTO_INCREMENT,
  titulo    VARCHAR(60),
  codautor  VARCHAR(5),
  FOREIGN KEY (codautor) REFERENCES autores(cod)
);

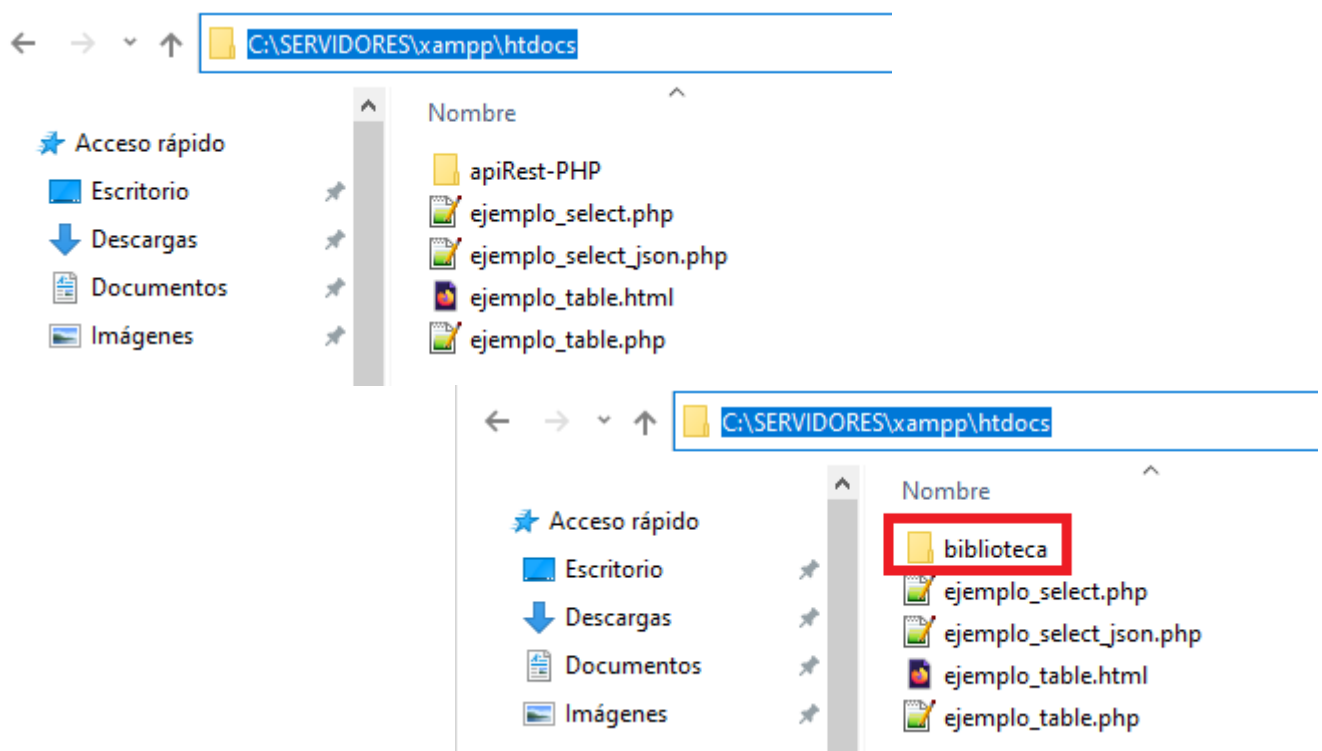
INSERT INTO autores (cod, nombre) VALUES
  ('WSHAK', 'William Shakespeare'),
  ('MCERV', 'Miguel de Cervantes'),
  ('FROJA', 'Fernando de Rojas');
```

```
INSERT INTO libros (id, titulo, codautor) VALUES
(1, 'Macbeth', 'WSHAK'),
(2, 'La Celestina (Tragicomedia de Calisto y Melibea)',
'FROJA'),
(3, 'Don Quijote de la Mancha', 'MCERV'),
(4, 'La tempestad', 'WSHAK'),
(5, 'La Galatea', 'MCERV'),
(6, 'Los trabajos de Persiles y Sigismunda', 'MCERV'),
(7, 'Novelas ejemplares', 'MCERV');

SELECT *
FROM autores, libros
WHERE autores.cod = libros.codautor;
```

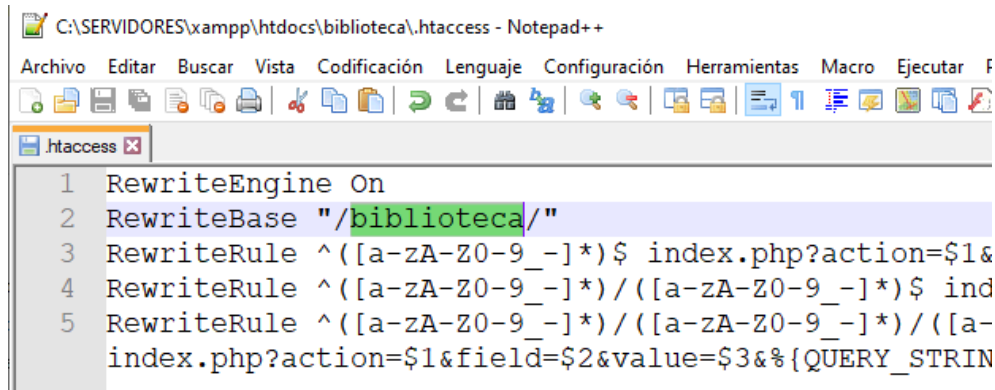
Paso 2) Copiar la aplicación API Rest en Apache

Copiaremos el código proporcionado por el profesor en la carpeta htdocs y cambiaremos su nombre al **Context Path**, por ejemplo **biblioteca**:



Paso 3) Configurar API Rest - Context Path

Editaremos el fichero **.htaccess** incluido en la aplicación, donde configuraremos el **Context Path**, que debe coincidir con el nombre de la carpeta, que en nuestro ejemplo es biblioteca:



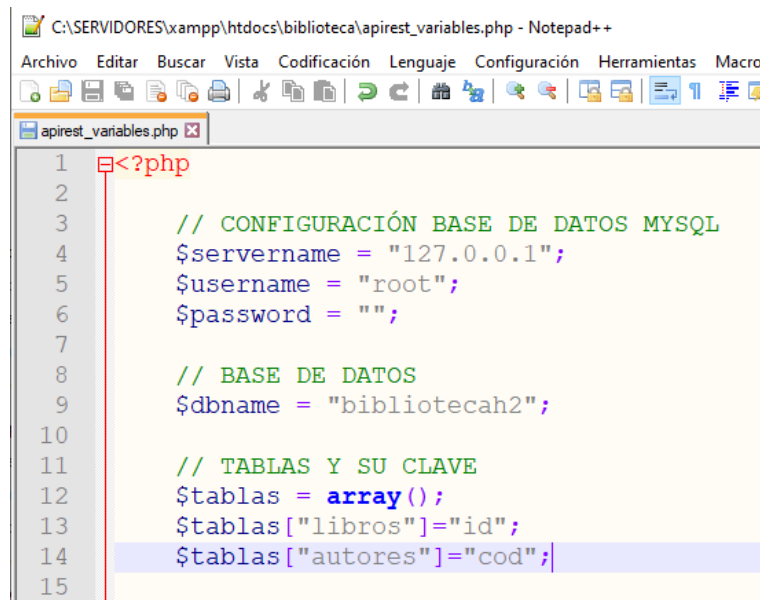
```

1 RewriteEngine On
2 RewriteBase "/biblioteca/"
3 RewriteRule ^([a-zA-Z0-9_-]*)$ index.php?action=$1&
4 RewriteRule ^([a-zA-Z0-9_-]*)/([a-zA-Z0-9_-]*)$ ind
5 RewriteRule ^([a-zA-Z0-9_-]*)/([a-zA-Z0-9_-]*)/([a-
  index.php?action=$1&field=$2&value=$3&{%QUERY_STRIN
  
```

Paso 4) Configurar API Rest - Conexión a MariaDB/MySQL

Editaremos el fichero **apirest_variables.php** y completaremos los datos. El archivo contiene 3 secciones:

- Datos de conexión a la BD
- Base de datos a mostrar por el API Rest
- Las tablas indicando su clave



```

1 <?php
2
3 // CONFIGURACIÓN BASE DE DATOS MYSQL
4 $servername = "127.0.0.1";
5 $username = "root";
6 $password = "";
7
8 // BASE DE DATOS
9 $dbname = "bibliotecah2";
10
11 // TABLAS Y SU CLAVE
12 $tablas = array();
13 $tablas["libros"]="id";
14 $tablas["autores"]="cod";
15
  
```

Paso 5) Probar funcionamiento

Abrir en el navegador la dirección del API Rest:

<http://localhost/biblioteca/>