

API REST CON XMYSQL



REST API

INTRODUCCION

Vamos a ver como generar una API REST directamente desde una base de datos MySQL/MariaDB utilizando una extensión super potente llamada **xmysql** que instalaremos desde GITHUB utilizando npm:

<https://github.com/o1lab/xmysql>

NOTA: Para instalar npm tendremos que instalarnos nodeJS:

<https://nodejs.org/es>

INTRODUCCIÓN REST

QUE ES UNA API

La palabra viene de **A**pplication **P**rogramming **I**nterface, y no es más que una aplicación que permite que otros programas se comuniquen con un programa en específico, por ejemplo Facebook.

A diferencia de los web services, las API no necesariamente deben comunicarse entre una red, pueden usarse entre dos aplicaciones en una misma computadora aunque lo habitual es que utilicen también la red.

REST

REST es una arquitectura para aplicaciones basadas en el intercambio de datos a través de la red (protocolo HTTP), sus siglas significan ***RE***presentational ***State Transfer***.

Es un protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla

VENTAJAS

Ventajas que ofrece REST para el desarrollo:

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos.
- Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas.
- La API REST siempre es independiente del tipo de plataformas o lenguajes ya que el intercambio de información se realiza siempre con el estándar JSON o XML

METODOS HTTP API REST

Las acciones o métodos que podemos utilizar con una API REST son los siguientes:

Recuperar un recurso:

GET

Crear un recurso

POST

Modificar parcialmente un recurso

PATCH

Reemplazar un recurso

PUT

Borrar un recurso

DELETE

LLAMADA A UNA API REST

Las llamadas al API se implementan como peticiones HTTP, en las que:

La URL representa el recurso. Ejemplo

<http://midominio.com/api/cursos/1>

El método representa la operación:

[GET http://midominio.com/api/cursos/1](http://midominio.com/api/cursos/1)

RESPUESTA DE UNA API REST

La respuesta siempre contendrá, a parte de la información solicitada, una cabecera HTTP con el estado (y que representa el resultado de la petición).

Ejemplo:

200 OK HTTP/1.1

404 NOT FOUND HTTP/1.1

RECURSOS

Nombre de los recursos:

- Plural mejor que singular, para lograr uniformidad:
- Obtenemos un listado de clientes: GET /api/clientes
- Obtenemos un cliente en particular: GET /api/clientes/1234
- Url's lo más cortas posibles
- Evita guiones y guiones bajos
- Deben ser semánticas para el cliente
- Utiliza nombres y no verbos
- Estructura jerárquica para indicar la estructura:
/api/clientes/1234/pedidos/203

TIPOS DE MENSAJES HTTP

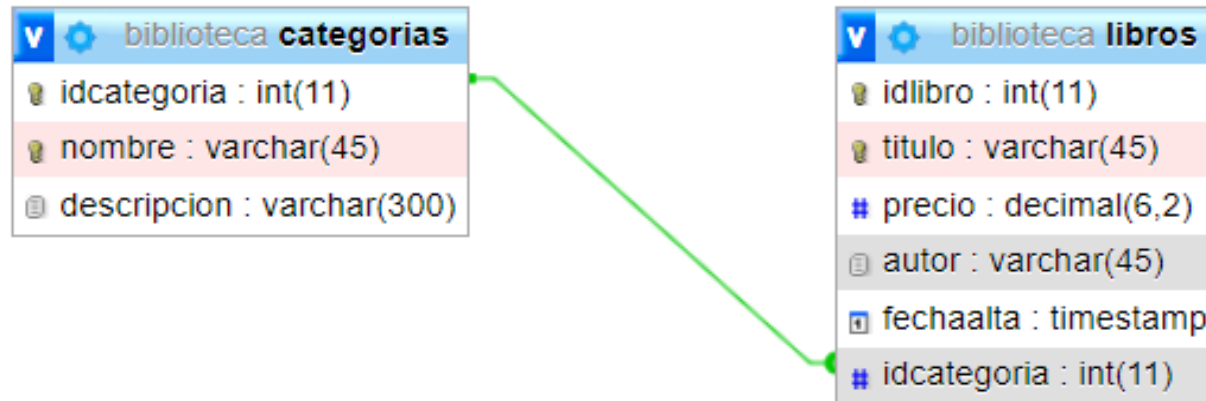
Mensajes descriptivos:

HTTP Verbs	GET, PUT, POST, DELETE, ...
HTTP Status Codes	200, 201, 404, 500, ...
HTTP Authentication	Usuario, password

EJEMPLO API REST CON XMYSQL

CREAR BASE DE DATOS

Necesitaremos una base de datos **biblioteca** con una tabla **libros** y **categorias** que utilizaremos para nuestra API



Utilizaremos MySQL WorkBench o phpMyAdmin para crear la base de datos y añadir algunos libros de prueba

INSTALAR LA LIBRERIA XMYSQL

Instalaremos la librería **xmysql** desde GITHUB siguiendo las instrucciones que se indican en:

<https://github.com/o1lab/xmysql>

Para ello necesitaremos instalarnos previamente el gestor de dependencias de NodeJS npm

<https://nodejs.org/en>

Y una vez instalado, desde la linea de comandos:

```
npm install -g xmysql
```

CONEXION A LA BASE DE DATOS

Desde el terminal de VSC o el cmd del sistema utilizamos el siguiente comando de xmysql para conectarnos a la base de datos

```
xmysql -h localhost -o 3306 -u root -p root -d biblioteca
```

Donde:

-h corresponde al host donde nos conectamos

-o corresponde al puerto de la conexión (por defecto 3306)

-u es el nombre de usuario (normalmente es root)

-p es la contraseña (normalmente es root o sin informar)

-d es el nombre de la base de datos que hemos creado antes

CONEXION A LA BASE DE DATOS

Si todo ha ido correctamente veremos

```
Generating REST APIs at the speed of your thought..  
-----  
Database           : rest_libreria  
Number of Tables   : 6  
  
REST APIs Generated : 123  
  
Xmysql took        : 2.2 seconds  
API's base URL     : localhost:3000  
-----
```

Y ya podemos probar la api

PROBANDO LA API CON REQBIN

REQBIN

Los servicios RESTful requieren que hagamos peticiones por medio de los métodos GET, POST, DELETE, etc.

Para esto podemos utilizar una extensión para Chrome llamada **reqbin** que podemos instalar desde

<https://reqbin.com/>

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making A validators. Load test your API with hundreds of simulated concurrent connections. Generate c online.

 File ▾  Generate Code ▾  Tools ▾

POST ▾

EXT ▾

Send

Authorization

Content (1)

Headers

Raw

JSON (application/json)

```
{"title": "hola mundo"}
```

GET
POST
PUT
PATCH
DELETE
HEAD
OPTIONS

Status: 200 (OK) Time: 75 ms Size: 0.01 kb

Content (1)

Headers (11)

Raw (13)

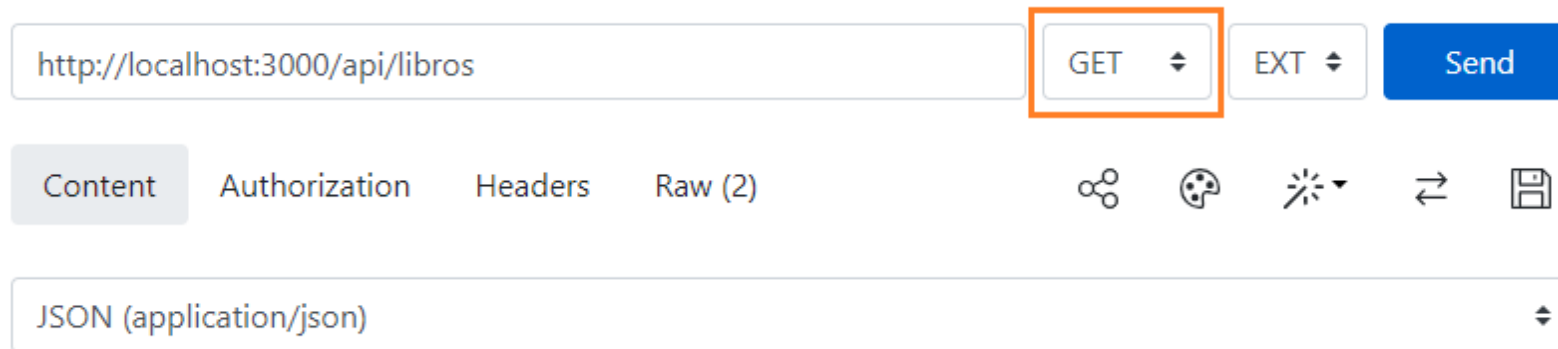
HTML

hola mundo

REQBIN: PETICIÓN GET

Vamos a probar la consulta de todos los libros de la base de datos utilizando la petición GET.

Para ello tenemos que activar el servidor con **php artisan serve** y, desde el panel de Reqbin:



NOTA: Fijémonos que estamos utilizando en la url **api/libros**

REQBIN: PETICIÓN GET

Y en el panel de resultados veremos el código de respuesta http y el listado de libros en formato json

Status: 200 (OK) Time: 31 ms Size: 0.55 kb

Content (29)

Headers (9)

Raw (11)

JSON

```
[{
  "idlibro": 2,
  "titulo": "Vincent Vega: historias de un matón",
  "precio": 55,
  "autor": "Arch Stanton",
  "fechaalta": "2023-09-26T20:51:46.000Z",
  "idcategoria": 1
}, {
  "idlibro": 107,
  "titulo": "Tirando a dar",
  "precio": 90,
  "autor": "desconocido",
```

REQBIN: PETICIÓN GET mas de 20 libros

Por defecto la librería xmysql nos entrega un máximo de 20 resultados.

Si queremos ampliarlo tendremos que utilizar el siguiente parámetro adicional en la llamada a la API:

http://localhost:3000/api/libros?_size=50

Donde **_size** es el parámetro utilizado para indicar el número de resultados que deseamos recibir (en este ejemplo 50)

REQBIN: PETICIÓN GET filas ordenadas

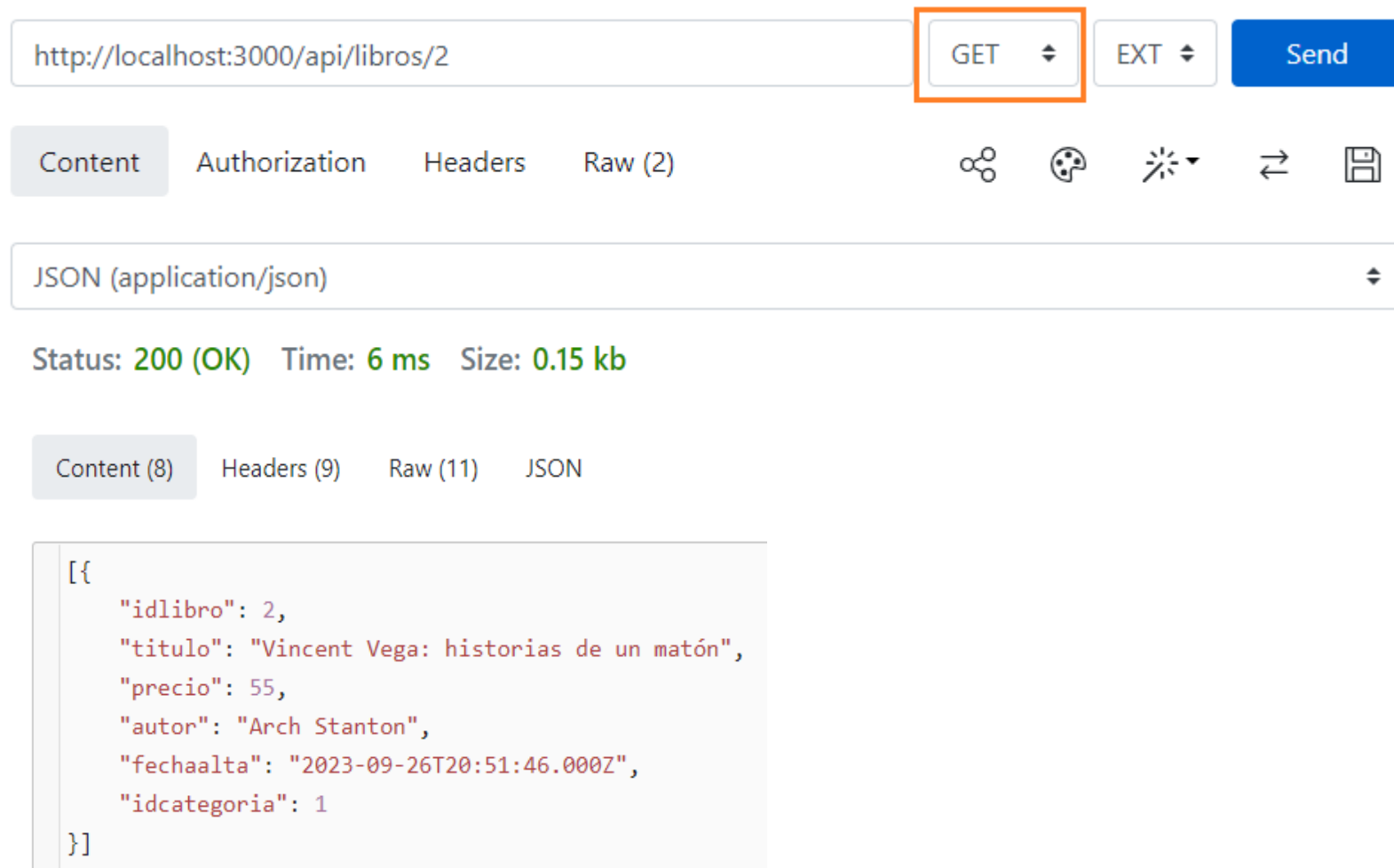
Y, si además, queremos que la consulta se devuelva ordenada por alguna columna de la tabla (por ejemplo título del libro) utilizaríamos un segundo parámetro en la petición

http://localhost:3000/api/libros?_size=50&_sort=titulo

Donde **_sort** es el parámetro utilizado para indicar la columna de ordenación

REQBIN: PETICIÓN GET

Vamos a probar la consulta de un libro utilizando el identificador de recurso.



http://localhost:3000/api/libros/2 GET EXT Send

Content Authorization Headers Raw (2)

JSON (application/json)

Status: 200 (OK) Time: 6 ms Size: 0.15 kb

Content (8) Headers (9) Raw (11) JSON

```
[{
  "idlibro": 2,
  "titulo": "Vincent Vega: historias de un matón",
  "precio": 55,
  "autor": "Arch Stanton",
  "fechaalta": "2023-09-26T20:51:46.000Z",
  "idcategoria": 1
}]
```

REQBIN: PETICIÓN POST

Vamos a probar el alta de un recurso utilizando el método POST.



The screenshot shows the ReqBin web interface for configuring a POST request. The URL field contains "localhost:3000/api/libros". The method dropdown is set to "POST". The content type dropdown is set to "JSON (application/json)". The request body field contains a JSON object: {"titulo": "Inventos con gaseosa", "precio": 40, "autor": "Luca Brasi", "idcategoria": 1}.

localhost:3000/api/libros

POST EXT Send

Content (1) Authorization Headers Raw (6)

JSON (application/json)

```
{"titulo": "Inventos con gaseosa", "precio": 40, "autor": "Luca Brasi", "idcategoria": 1}
```

NOTA 1: Informamos url del recurso y método a utilizar (1a línea)

NOTA 2: Informamos el formato de envío de los datos del recurso (2a línea)

NOTA 3: Informamos los datos de recurso a crear en formato json (3a línea)

REQBIN: PETICIÓN POST

Deberíamos ver una respuesta correcta:

Status: 200 (OK) Time: 40 ms Size: 0.13 kb

Content (10)

Headers (9)

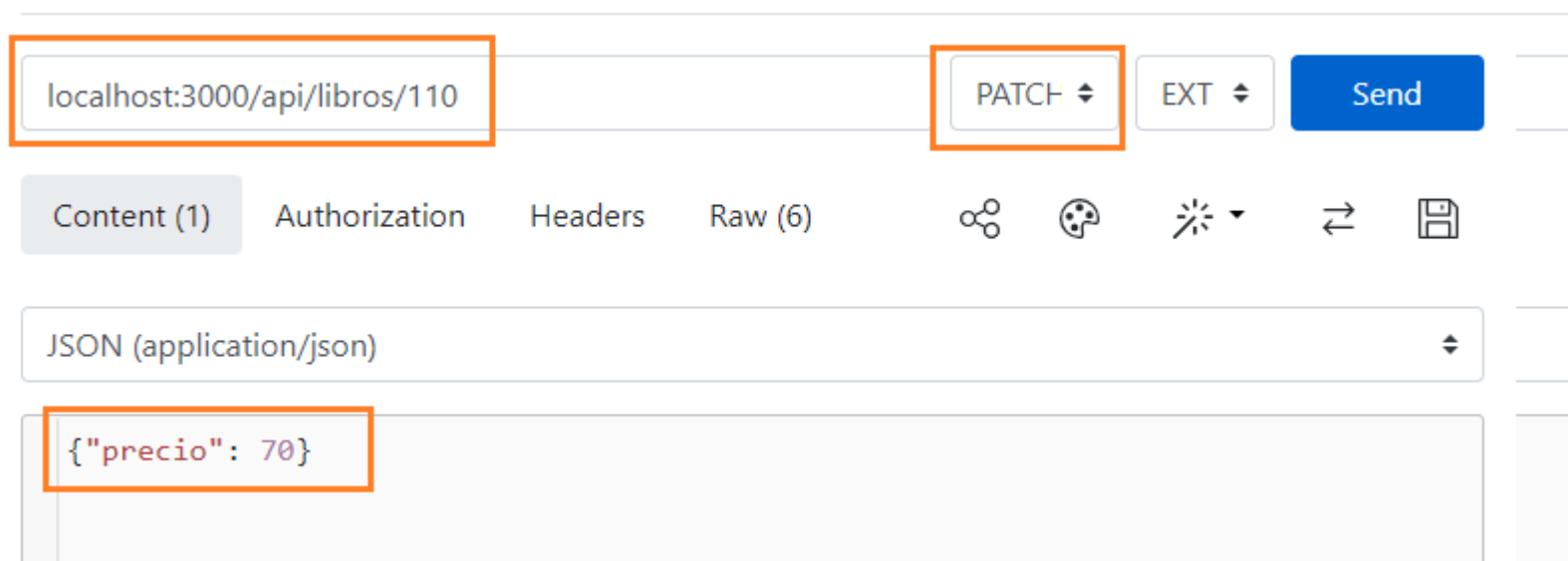
Raw (11)

JSON

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 110,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "",
  "protocol41": true,
  "changedRows": 0
}
```

REQBIN: PETICIÓN PUT/PATCH

Vamos a probar la modificación de un recurso utilizando el método PATCH.



The screenshot shows the REQBIN tool interface. The URL field contains `localhost:3000/api/libros/110`. The method dropdown is set to `PATCH`. The content type dropdown is set to `JSON (application/json)`. The request body field contains `{"precio": 70}`. The interface includes tabs for Content (1), Authorization, Headers, and Raw (6), along with various icons for sharing, debugging, and saving.

NOTA 1: Informamos url del recurso con el identificador del recurso a modificar y método a utilizar (1a línea)

NOTA 2: Informamos el formato de envío de los datos del recurso (2a línea)

NOTA 3: Informamos los datos de recurso a modificar en formato json (3a línea)

REQBIN: PETICIÓN PUT/PATCH

En el panel de resultados veremos el resultado de la petición

Status: 200 (OK) Time: 8 ms Size: 0.16 kb

Content (10)

Headers (9)

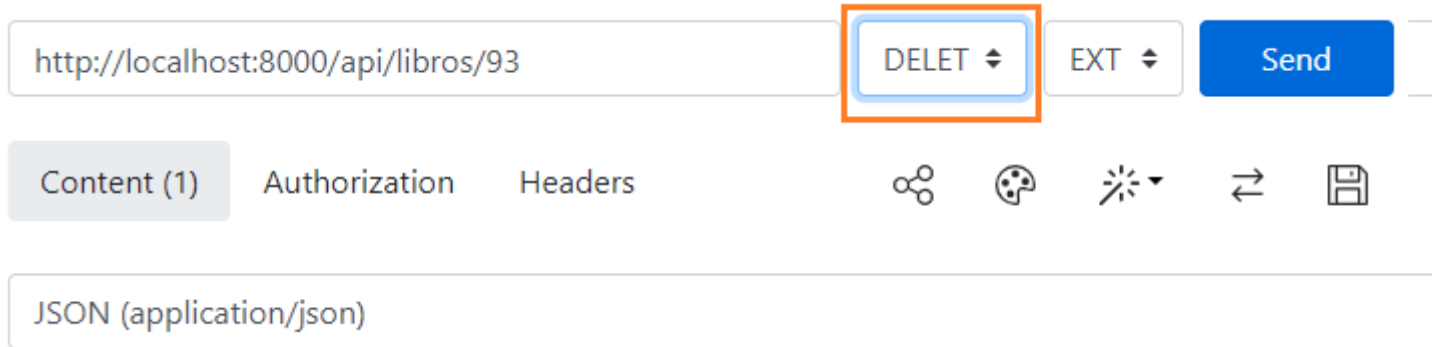
Raw (11)

JSON

```
{
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 0,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "(Rows matched: 1  Changed: 1  Warnings: 0",
  "protocol41": true,
  "changedRows": 1
}
```

REQBIN: PETICIÓN DELETE

Vamos a probar el borrado de un recurso utilizando el método DELETE.



The screenshot shows the REQBIN tool interface. At the top, there is a text input field containing the URL `http://localhost:8000/api/libros/93`. To the right of this field are two dropdown menus: the first is set to 'DELETE' and is highlighted with an orange border, and the second is set to 'EXT'. To the right of these dropdowns is a blue 'Send' button. Below the URL field, there are three tabs: 'Content (1)', 'Authorization', and 'Headers'. The 'Content (1)' tab is selected and highlighted. To the right of the tabs are several icons: a share icon, a palette icon, a sun icon with a dropdown arrow, a double arrow icon, and a save icon. Below the tabs, there is a text input field containing the content type `JSON (application/json)`.

NOTA 1: Informamos url del recurso con el identificador del recurso a borrar y método a utilizar (1a linea)

REQBIN: PETICIÓN DELETE

En el panel de resultados veremos el resultado de la petición

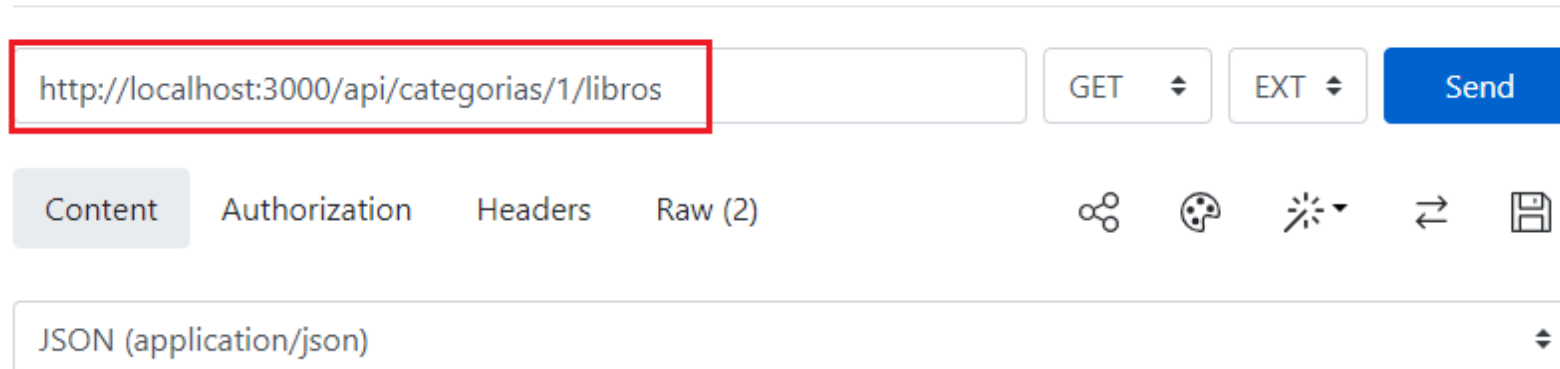
```
{  
  "fieldCount": 0,  
  "affectedRows": 1,  
  "insertId": 0,  
  "serverStatus": 2,  
  "warningCount": 0,  
  "message": "",  
  "protocol41": true,  
  "changedRows": 0  
}
```

REQBIN: PETICIÓN GET CON JOIN

Podemos consultar también todos los libros que pertenecen a una categoría determinada junto con los datos de la categoría:

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.



http://localhost:3000/api/categorias/1/libros

GET EXT Send

Content Authorization Headers Raw (2)

JSON (application/json)

REQBIN: PETICIÓN GET CON JOIN

Y veremos como se muestra el libro o libros que pertenecen a la categoría seleccionada

Status: 200 (OK) Time: 7 ms Size: 0.15 kb

Content (8)

Headers (9)

Raw (11)

JSON

```
[{
  "idlibro": 2,
  "titulo": "Vincent Vega: historias de un matón",
  "precio": 55,
  "autor": "arch stanton",
  "fechaalta": "2023-09-26T20:51:46.000Z",
  "idcategoria": 1
}]
```

PROBANDO LA API DESDE UN FRONTEND WEB CON AJAX

API REST CON AJAX

Vamos a probar nuestra API utilizando un servicio ajax con JavaScript.

Pare ello necesitaremos el siguiente documento html:

```
<main>
  <h2>Petición api rest laravel con ajax</h2>
  <button onclick="rest('A')">Alta de libro</button>
  <button onclick="rest('T')">Consulta de libros</button>
  <button onclick="rest('C')">Consulta de libro</button>
  <button onclick="rest('M')">Modificación de libro</button>
  <button onclick="rest('B')">Baja de libro</button>
</main>
<script> ... </script>
```

API REST CON AJAX. PETICION I

Paso 1: Crear la llamada ajax

Vamos a confeccionar, en un fichero JS que incorporaremos a un documento html, una función con la llamada ajax utilizando **fetch()** y con un **json** como respuesta:

```
fetch(api, parametros)
.then((resp) => { return resp.json() })
.then((mensaje) => { console.log(mensaje) })
.catch((error) => { alert(error) })
```

API REST CON AJAX. PETICION II

Paso 2: GET → Consulta de un recurso

Variable api (indicamos el recurso a consultar):

```
let api = 'http://localhost:3000/api/libros/20';
```

Parameros petición:

```
let parametros = {  
    method: 'GET',  
    mode: 'cors' //en caso que la API no pertenezca al dominio  
}
```

Respuesta esperada:

```
created_at: "2021-11-25T10:03:02.000000Z"  
id: 20  
precio: "4.02"  
titulo: "Facilis nulla aliquam vel voluptatem."  
updated_at: "2021-11-25T10:03:02.000000Z"
```

API REST CON AJAX. PETICION III

Paso 3: GET → Consulta de todos los recursos

Variable api:

```
let api = 'http://localhost:3000/api/libros';
```

Parameros petición:

```
let parametros = {  
    method: 'GET',  
    mode: 'cors' //en caso que la API no pertenezca al dominio  
}
```

Respuesta esperada:

```
▶0: {id: 2, titulo: 'Quo ea non est consectetur.', precio:  
▶1: {id: 3, titulo: 'Aut corporis corrupti modi facere rei  
▶2: {id: 4, titulo: 'Nam laborum labore delectus rerum et.  
▶3: {id: 5, titulo: 'Cupiditate soluta illo nemo et.', pre  
▶4: {id: 6, titulo: 'Impedit ut provident numquam sit expl  
▶5: {id: 7, titulo: 'Dolores id corporis maiores porro opt
```

API REST CON AJAX. PETICION IV

Paso 4: POST → Alta de un recurso

Variable api:

```
let api = 'http://localhost:3000/api/libros';
```

Datos a enviar en la petición:

```
let datos = {  
  'titulo': 'Testaferria avanzada',  
  'precio': 60.00  
}
```

API REST CON AJAX. PETICION V

Paso 4: POST → Alta de un recurso (cont.)

Parameros petición:

```
let parametros = {  
  method: 'POST',  
  body: JSON.stringify(datos), //convertir el objeto datos a JSON  
  mode: 'cors', //en caso que la API no pertenezca al dominio  
  headers: { //Cabecera indicando el tipo de datos que se envia  
    'Content-Type': 'application/json'  
  }  
}
```

Respuesta esperada:

```
affectedRows: 1  
changedRows: 0  
fieldCount: 0  
insertId: 96  
message: ""  
protocol41: true  
serverStatus: 2  
warningCount: 0
```

API REST CON AJAX. PETICION V

Paso 4: POST → Alta de un recurso duplicado

Si intentamos volver a dar de alta el recurso anterior obtendríamos un error en la respuesta

Respuesta esperada:

```
"error": {  
  "code": "ER_DUP_ENTRY",  
  "errno": 1062,  
  "sqlMessage": "Duplicate entry 'Ronaldo Macaón: vida y milagros deluxe' for key 'titulo'",  
  "sqlState": "23000",  
  "index": 0,  
  "sql": "INSERT INTO `libros` SET `titulo` = 'Ronaldo Macaón: vida y milagros deluxe', `precio` = 160"  
}
```

API REST CON AJAX. PETICION VI

Paso 5: PATCH → Modificación de un recurso

Variable api (indicamos el recurso a modificar):

```
let api = 'http://localhost:3000/api/libros/96';
```

Datos a enviar en la petición:

```
let datos = {  
  'titulo': 'Testaferria avanzada deluxe',  
  'precio': 40.00  
}
```


API REST CON AJAX. PETICION VII

Paso 5: PATCH → Modificación de recurso

Parameros petición:

```
let parametros = {  
  method: 'PATCH',  
  body: JSON.stringify(datos), //convertir el objeto datos a JSON  
  mode: 'cors', //en caso que la API no pertenezca al dominio  
  headers: { //Cabecera indicando el tipo de datos que se envia  
    'Content-Type': 'application/json'  
  }  
}
```

Respuesta esperada:

```
affectedRows: 1  
changedRows: 1  
fieldCount: 0  
insertId: 0  
message: "(Rows matched: 1 Changed: 1 Warnings: 0"  
protocol41: true  
serverStatus: 2  
warningCount: 0
```

API REST CON AJAX. PETICION VIII

Paso 6: DELETE → Borrado de un recurso

Variable api (indicamos el recurso a borrar):

```
let api = 'http://localhost:3000/api/libros/96';
```

Parameros petición:

```
let parametros = {  
  method: 'DELETE',  
  mode: 'cors', //en caso que la API no pertenezca al dominio  
}
```

Respuesta esperada:

```
affectedRows: 1  
changedRows: 0  
fieldCount: 0  
insertId: 0  
message: ""  
protocol41: true  
serverStatus: 2  
warningCount: 0
```

Aviso Legal

Los derechos de propiedad intelectual sobre el presente documento son titularidad de David Alcolea Martinez Administrador, propietario y responsable de www.alcyon-it.com El ejercicio exclusivo de los derechos de reproducción, distribución, comunicación pública y transformación pertenecen a la citada persona.

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro).

La ley que ampara los derechos de autor establece que: “La introducción de una obra en una base de datos o en una página web accesible a través de Internet constituye un acto de comunicación pública y precisa la autorización expresa del autor”. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multa, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaran, distribuyeren o comunicaren públicamente, en todo o en parte, o su transformación, interpretación o ejecución fijada en cualquier tipo de soporte o comunicada a través de cualquier medio.

El usuario que acceda a este documento no puede copiar, modificar, distribuir, transmitir, reproducir, publicar, ceder, vender los elementos anteriormente mencionados o un extracto de los mismos o crear nuevos productos o servicios derivados de la información que contiene.

Cualquier reproducción, transmisión, adaptación, traducción, modificación, comunicación al público, o cualquier otra explotación de todo o parte del contenido de este documento, efectuada de cualquier forma o por cualquier medio, electrónico, mecánico u otro, están estrictamente prohibidos salvo autorización previa por escrito de David Alcolea. El autor de la presente obra podría autorizar a que se reproduzcan sus contenidos en otro sitio web u otro soporte (libro, revista, e-book, etc.) siempre y cuando se produzcan dos condiciones:

1. Se solicite previamente por escrito mediante email o mediante correo ordinario.
2. En caso de aceptación, no se modifiquen los textos y se cite la fuente con absoluta claridad.

David Alcolea
david-alcolea@alcyon-it.com
www.alcyon-it.com



Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro) así como su distribución sin la autorización explícita y por escrito de su autor.