

**HLC**

***2º DAM***

**I.E.S. POLITÉCNICO H. LANZ  
JOSÉ MARÍA MOLINA**

**php**

**TEMA 2 – APIREST**

# TEMA 2 – APIREST



## Vamos a crear una APIRest con PHP

- ✓ 1 Clases y Métodos
- ✓ 2 Estructura API
- ✓ 3 Json Encode y Decode
- ✓ 4 Headers
- ✓ 5 Comprobación de variables
- ✓ 6 Respuestas HTTP
- ✓ 7 Ejemplo de API
- ✓ 8 POSTMAN

Vamos a ver estos apuntes que muestran toda la teoría necesaria para montar una API.

Una vez acabados, vemos código y vamos paso por paso repasando todo desde el principio.

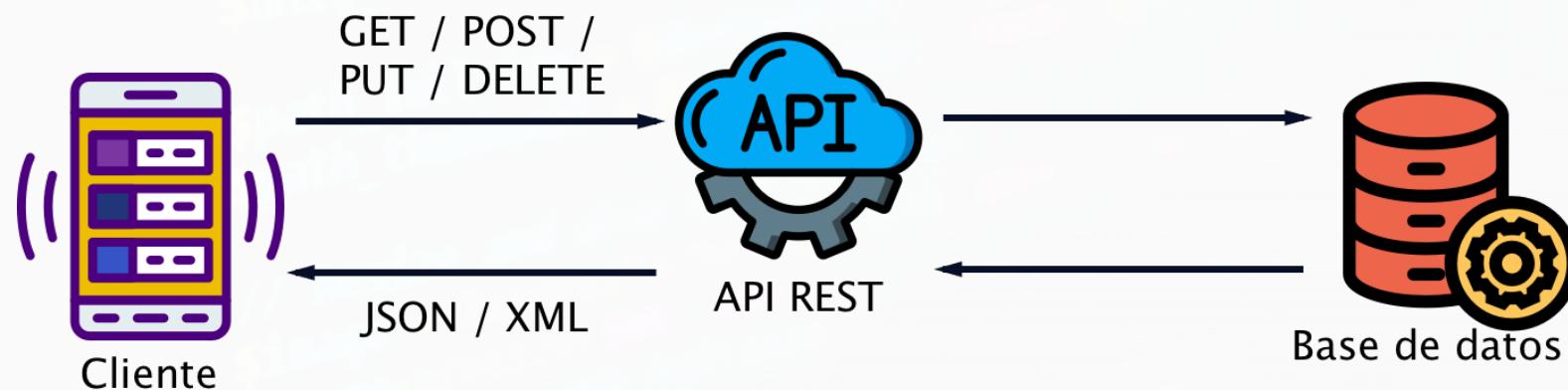
# TEMA 2 – APIREST

php

✓ **API Rest:** API es una Interfaz de Programación de Aplicaciones, es decir, una forma de que otras aplicaciones interactúen con los datos de la primera. **Rest** significa que es una arquitectura que se ejecuta sobre HTTP/HTTPS.

✓ Formas de probar una API:

- Conjunto de Formularios
- APP hecha con Móvil(Android/IOS) o APP de Escritorio
- POSTMAN

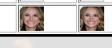


# TEMA 2 - APIREST

php

WEB

INSERTAR

ID	Nombre	Edad	Activo	Foto URL	Foto Base64	Acciones
92	Tom Hanks	65	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
93	Al Pacino	81	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
94	Brad Pitt	58	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
95	Tim Robbins	63	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
96	Morgan Freeman	84	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
97	Marlon Brando	80	No			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
99	Robert De Niro	78	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
101	Leonardo DiCaprio	47	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
102	Keanu Reeves	57	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
103	Matt Damon	51	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
104	Anne Hathaway	39	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
105	Cate Blanchett	52	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
106	Natalie Portman	40	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
107	Michelle Pfeiffer	63	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>
108	Julia Roberts	55	Sí			<a href="#">Editar</a> <a href="#">Subir foto</a> <a href="#">Borrar</a>

localhost/APIRest\_BBDDActoresV8/crud/leer.php

JSON Datos sin procesar Cabeceras  
Guardar Copiar Contraseña Expander todo Filtrar JSON

92: id: 92 nombre: "Tom Hanks" edad: 65 activo: 1 fotourl: "" fotocodif: ""

93: id: 93 nombre: "Al Pacino" edad: 81 activo: 1 fotourl: "" fotocodif: ""

94: id: 94 nombre: "Brad Pitt" edad: 58 activo: 1 fotourl: "" fotocodif: ""

95: id: 95 nombre: "Tim Robbins" edad: 63 activo: 1 fotourl: "" fotocodif: ""

96: id: 96 nombre: "Morgan Freeman" edad: 84 activo: 1 fotourl: "" fotocodif: ""

97: id: 97 nombre: "Marlon Brando" edad: 80 activo: 0 fotourl: "" fotocodif: ""

99: id: 99 nombre: "Robert De Niro" edad: 78 activo: 1 fotourl: "" fotocodif: ""

101: id: 101 nombre: "Leonardo DiCaprio" edad: 47 activo: 1 fotourl: "" fotocodif: ""

102: id: 102 nombre: "Keanu Reeves" edad: 57 activo: 1 fotourl: "" fotocodif: ""

103: id: 103 nombre: "Matt Damon" edad: 51 activo: 1 fotourl: "" fotocodif: ""

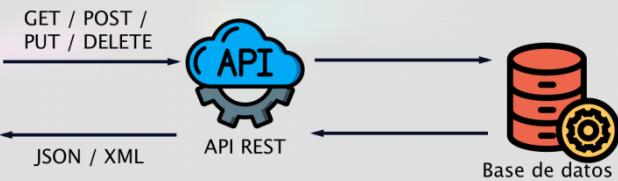
104: id: 104 nombre: "Anne Hathaway" edad: 39 activo: 1 fotourl: "" fotocodif: ""

105: id: 105 nombre: "Cate Blanchett" edad: 52 activo: 1 fotourl: "" fotocodif: ""

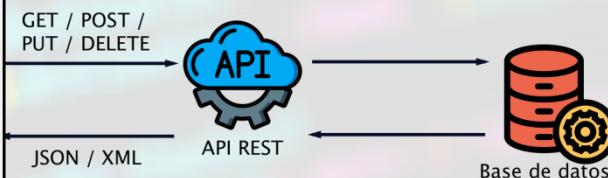
106: id: 106 nombre: "Natalie Portman" edad: 40 activo: 1 fotourl: "" fotocodif: ""

107: id: 107 nombre: "Michelle Pfeiffer" edad: 63 activo: 1 fotourl: "" fotocodif: ""

108: id: 108 nombre: "Julia Roberts" edad: 55 activo: 1 fotourl: "juliac...".jpg" fotocodif: "108"



## ESCRITORIO



PMDM



# *1 CLASES Y MÉTODOS*



- ✓ Uso básico de Clases y métodos:
  - Visibilidad atributos y métodos (public/private/protected) tienen el mismo funcionamiento que en Java
  - Métodos: function
  - Constructor: function \_\_construct(){}
  - Acceso a variables de clase y funciones mediante ->
  - Nuevos objetos con new

# 1 CLASES Y MÉTODOS



```
<?php
class Employee
{
    private $first_name;
    private $last_name;
    private $age;

    public function __construct($first_name, $last_name, $age)
    {
        $this->first_name = $first_name;
        $this->last_name = $last_name;
        $this->age = $age;
    }

    public function getFirstName()
    {
        return $this->first_name;
    }

    public function getLastname()
    {
        return $this->last_name;
    }

    public function getAge()
    {
        return $this->age;
    }
}
?>
```

<https://code.tutsplus.com/es/basics-of-object-oriented-programming-in-php--cms-31910t>

```
<?php
$objEmployee = new Employee('Bob', 'Smith', 30);

echo $objEmployee->getFirstName(); // print 'Bob'
echo $objEmployee->getLastname(); // prints 'Smith'
echo $objEmployee->getAge(); // prints '30'
?>
```

<https://www.php.net/manual/es/language.oop5.basic.php>

## 2 ESTRUCTURA API



### -Estructura:

- Conexión (utilizamos librería mysqli)
- Creación de Tablas/Datos (sobre mysql/mariadb. Ej: phpmyadmin de LAMP)
- CRUD y páginas de envío/recepción de datos (sin formularios)

-**Tipos de Llamadas:** No confundir los ***métodos de envío POST/GET*** que ya conocemos con los ***métodos HTTP POST/GET***.

- Los primeros son formas de enviar datos desde un HTML a un servidor PHP
- Los segundos son formas de funcionamiento del servidor con respecto a los datos que se reciben. Además son estándar, utilizados por cualquier lenguaje de programación.
- Métodos: GET, POST, DELETE, PUT, PATCH,...
- ¿Por qué usar métodos distintos? Símil de las cajas rápidas en los supermercados o los carriles lentos.. No obligatorio, sí recomendable y además se puede “capar” (muy importante esto).

## 2 ESTRUCTURA API



- ✓ **GET:** recupera información de un recurso en el servidor. No debe cambiar el estado del servidor (*solo consulta*).
- ✓ **POST:** envía datos al servidor para que este realice “*alguna acción que no es consulta*”, por ejemplo, para crear algo nuevo. Es el “comodín”
- ✓ **DELETE:** Se utiliza para *borrar* un recurso del servidor.
- ✓ **PUT:** *actualiza* un recurso en el servidor. La solicitud debe contener la representación completa del recurso.
- ✓ **PATCH:** Se utiliza para realizar una *actualización parcial* de un recurso en el servidor. En lugar de enviar la representación completa del recurso, solo se envían los cambios que se deben aplicar.

## 3 JSON

- ✓ **JSON** es un formato estructurado para intercambio de información vía HTTP. Ejemplo:

```
{  
    "id": 92,  
    "nombre": "Tom Hanks",  
    "edad": 65,  
    "activo": 1,  
    "fotourl": "",  
    "fotocodif": ""  
},  
,  
{  
    "id": 93,  
    "nombre": "Al Pacino",  
    "edad": 81,  
    "activo": 1,  
    "fotourl": "",  
    "fotocodif": ""  
},  
}
```

### Por qué JSON??

Formato de  
intercambio de datos  
**UNIVERSAL** (facilita  
traducciones  
Multilenguaje )

- ✓ A nosotros nos hará falta pasar JSON→PHP (decodificar tras consulta) y PHP→JSON (codificar para INS/UPD/DEL)

## 3 JSON



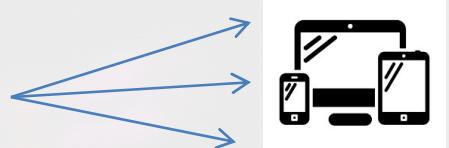
✓ **json\_decode()**: se usa cuando queremos **CONVERTIR JSON→variables PHP (Utilizado en cada operación CRUD)**. Convierte una cadena JSON en una variable PHP para su uso. Si utilizamos POST obtendremos los datos enviados en el cuerpo de la solicitud POST, que de forma normal tendrán un formato JSON. Si usamos GET se convierte automáticamente a JSON. Se utiliza junto a la función:

**file\_get\_contents("php://input")**: de forma normal esta función lee el contenido de un archivo, pero este parámetro indica que se lee el cuerpo de la solicitud HTTP actual. Hará falta decodificar el JSON por lo que deberá estar dentro de un json\_decode(). Ej:

```
$datos = json_decode(file_get_contents("php://input"));
```

✓ **json\_encode()**: **se usa cuando queremos dar una respuesta tipo JSON desde el servidor**. Convierte una variable PHP en una cadena JSON. Ej: Crear un array asociativo para codificarlo en una cadena json con un mensaje de información:

```
json_encode(array("info" => "Actor borrado!"));
```



## 4 HEADERS

- ✓ **Cabeceras HTTP (Headers)**: permiten configurar seguridad y funcionamiento de la petición http. Las más importantes son las cabeceras CORS.
- ✓ **CORS (Cross-Origin Resource Sharing)** : CORS es un mecanismo de seguridad implementado en los navegadores web para evitar problemas de seguridad relacionados con solicitudes HTTP entre diferentes dominios.
- ✓ Vamos a suponer que la web (API) de nuestro servidor alojado en nuestra casa (dominio origen) se le hace una solicitud HTTP desde un origen distinto (está en un dominio distinto, por ejemplo fuera de nuestra casa). En estos casos deberíamos poder configurar las condiciones: desde dónde se aceptan las peticiones, el método de envío y las cabeceras permitidas (tipos de datos, codificación, etc..)

## 4 HEADERS

- ✓ Ejemplos de cabeceras:

-header("Access-Control-Allow-Origin: \*"); //Permite hacer petición desde cualquier lugar

-header("Access-Control-Allow-Origin: <http://midominio>,  
<http://otrodominio> "); //Permite hacer petición desde estos dos dominios.

- ✓ La cabecera anterior implica el uso de otras:

-header("Access-Control-Allow-Methods: POST"); //Solo permite método POST desde el exterior.

-header("Access-Control-Allow-Headers: Content-Type, Authorization"); //Son los tipos de cabeceras que se aceptan en la petición a la API. Se podría configurar tipo de contenido como ficheros de subida o tokens de autorización entre otros.

# 4 HEADERS

- OTRAS:
- header("Content-Type: application/json; charset=UTF-8");  
//Solicitud CORS que indica que la solicitud tiene datos en Json y  
además el conjunto de caracteres utilizado es UTF-8.
- header("Access-Control-Max-Age: 3600"); //Indica que la  
respuesta se almacenará 1hora en caché.

Vamos a pensar los pasos necesarios  
para crear la API...  
Quiero enviar un actor para que se  
inserte...  
Qué pasos concretos hemos de seguir??

# 5 COMPROBACIONES DE VARIABLES



## ✓ Paso 1: Captura datos JSON de entrada antes de Insertar/Borrar/Actualizar

✓ \$datos = json\_decode(file\_get\_contents("php://input"));

-En este punto se harían comprobaciones locales con el objeto datos

## ✓ Paso 2: Distintas comprobaciones (según operación):

-ha pasado variables? isset(\$\_GET['id'])

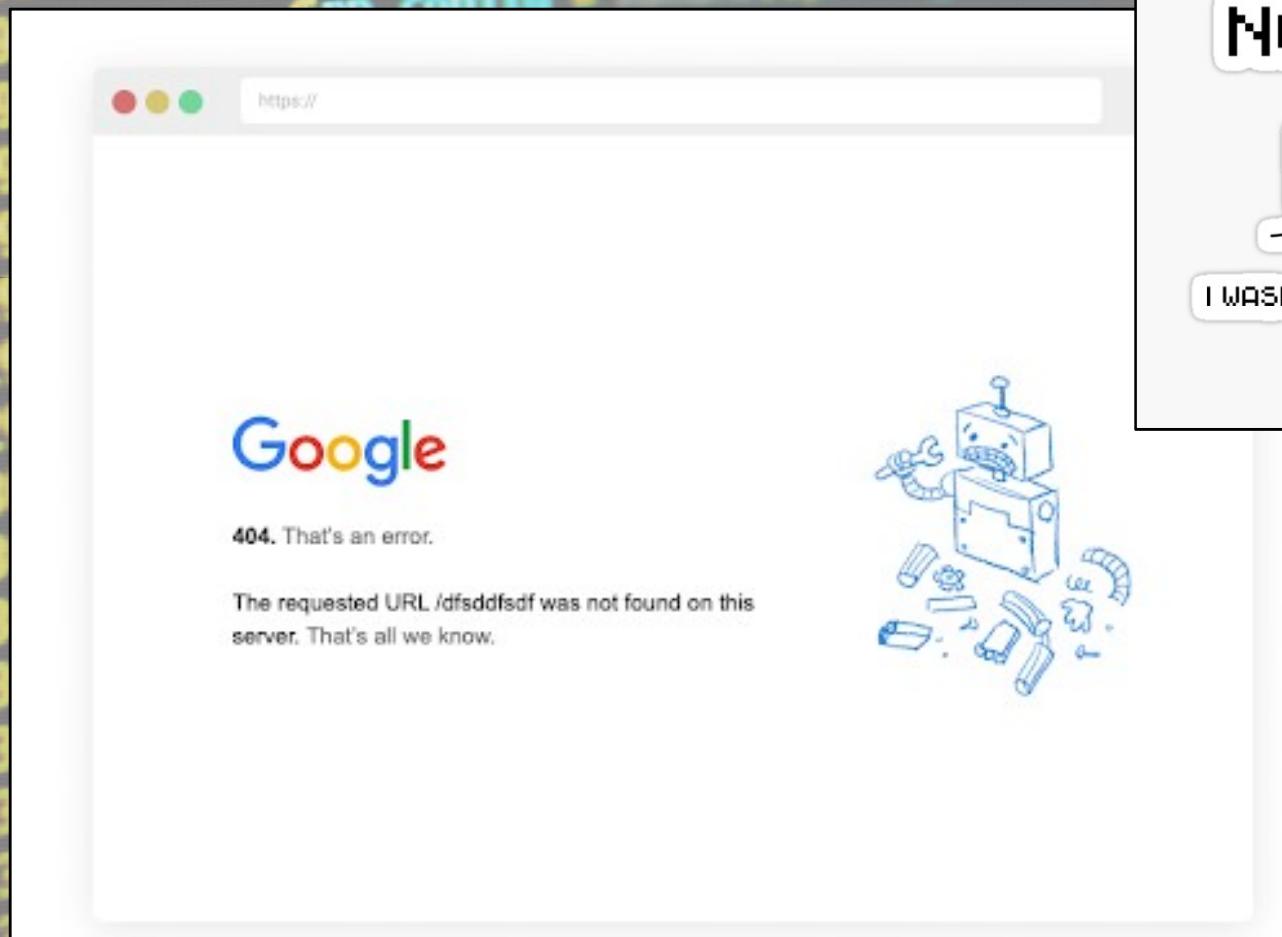
-número de filas leído es correcto?: \$result->num\_rows > 0

## ✓ Paso 3 y 4 : Operación (CRUD) y respuesta personalizada.

```
304  
305 // Resize and compress image  
306 $image = imagecreatefromjpeg($img);  
307 $image->resize(500, 500);  
308 $image->save('sport.jpg');
```

# 6 RESPUESTAS HTTP

php



# 6 RESPUESTAS HTTP



- ✓ Los servidores suelen enviar códigos de respuesta **universales** para indicar situaciones tras consulta u operación sobre los datos.
  - Primer código indica si el problema está del lado del cliente (4) o del servidor (5), los que empiezan por 1 son códigos informativos, los que empiezan por 2 son códigos de éxito,
- ✓ Permite que el cliente, es decir, una App(web/móvil/escritorio) puede saber lo que ha ocurrido y personalizar respuesta.
- ✓ <https://es.semrush.com/blog/codigos-de-estado-http/>
- ✓ Y como son **respuestas universales** universales, hay que cumplirlas...

306  
307  
308

\$image  
\$image->resiz...  
\$image->save(Sp...)

# 6 RESPUESTAS HTTP



## ✓ INSERT

- 201 **OK (Creado)**
- 503 No se puede insertar (repetido?)
- 400 Faltan datos para inserción o son incorrectos

## ✓ UPDATE/DELETE

- 200 **OK (update/delete/select correcto = operación correcta)**
- 503 No se puede borrar (por problemas en el servidor)
- 400 Faltan datos o son incorrectos

## ✓ SELECT

- 200 **OK (update/delete/select correcto = operación correcta)**
- 404 No hay datos (No se encuentra el recurso solicitado)

## 7 EJEMPLO DE API



1. Montamos LAMP (Docker)
2. Copiamos la API a la carpeta de Apache enlazada al contenedor (por ejemplo /home/usuario/app/apirest/ o en la carpeta compartida de C: si usamos docker para windows)
3. Accedemos a PHPMyadmin (utilizando las credenciales de inicio de LAMP)
4. Creamos BBDD
5. Ejecutamos carga de datos actores.sql
6. Modificamos datos de conexión a la base de datos con usuario y pwd proporcionados por LAMP (ver logs en docker)
7. Probamos la API : no se pueden enviar datos para insertar/borrar/actualizar por un formulario html normal porque necesita JSON!

## 8 POSTMAN

php

- ✓ Aplicación muy útil que sirve para probar API Rest
- ✓ <https://www.postman.com/downloads/>
- ✓ Hay que indicar método, url y los datos necesarios
- ✓ Un método inadecuado desde el mismo dominio no evita que se ejecute el php a no ser que lo evitemos mediante los headers.  
Ej: DELETE al LEER

The screenshot shows the Postman application interface. At the top, there is a dropdown menu set to 'DELETE', a URL input field containing 'http://localhost/crud/leer.php', and a 'Send' button. Below the header, there is a sidebar with method buttons: GET, POST, PUT, PATCH, DELETE (which is highlighted in grey), and HEAD. To the right of the sidebar, there are tabs for Headers (6), Body, Pre-request Script, Tests, Settings, and Cookies. Under the Headers tab, there are two rows, each with a 'Value' column and a 'Bulk Edit' button. At the bottom of the interface, there is a 'Test Results' section showing a single entry with a status of '200 OK'. On the far right, there are buttons for 'Save Response' and a dropdown menu.

# 8 POSTMAN



✓ LEER: Se pasan los parámetros desde Params: Key y Value. Si no se le pasa nada devuelve todo (porque así está programado)

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** `http://localhost/crud/leer.php?id=99`
- Params Tab:** The "Params" tab is highlighted with a red box. It contains a table with one row, where the "id" key has a checked checkbox and the value is "99".
- Body Tab:** The response body is displayed in Pretty format:

```
1  [
2   "act": [
3     {
4       "id": 99,
5       "nombre": "Robert De Niro",
6       "edad": 78,
7       "activo": 1
8     }
9   ]
10 ]
```
- Headers Tab:** Headers (7) are listed, but no specific headers are shown.
- Test Results:** No test results are present.
- Status:** 200 OK, 14 ms, 309 B
- Save Response:** A button to save the response.

## 8 POSTMAN



- ✓ LEER: Se pasan los parámetros desde Params: Key y Value. Si no se le pasa nada devuelve todo (porque así está programado)
- ✓ **Prueba de métodos CORS:** Intentamos un acceso cruzado desde un dominio distinto al del API: accedemos a SWAPI con distintos métodos desde POSTMAN: POST, GET, PUT.....
- ✓ <https://swapi.dev/api/people>
- ✓ ¿qué ocurre en cada caso?

A screenshot of the Postman application. At the top, there's a header bar with the Postman logo and some icons. Below it, the main interface shows a request configuration. On the left, there's a dropdown menu with 'POST' selected. In the center, the URL 'https://swapi.dev/api/people' is entered. To the right of the URL are several tabs: 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. Under the 'Body' tab, there's a dropdown menu with 'Text' selected. At the bottom of the interface, there are two large buttons: 'Send' on the right and 'Cancel' on the left.

# 8 POSTMAN

php

✓ INSERTAR: Se pasan los parámetros desde Body utilizando formato raw. No necesita id porque es autoincrementado!!

The screenshot shows the Postman interface for an insert operation. The URL is `http://localhost/crud/insertar.php`. The 'Body' tab is selected, and the 'raw' option is chosen. The JSON payload is:

```
1 {"nombre": "Torrente",  
2 "edad": 55 ,  
3 "activo": 1}
```

The response status is 201 Created, with a response body:

```
1 {"info": "Actor/Actriz Creado!"}
```

# 8 POSTMAN

php

- ✓ BORRAR: Se pasan los parámetros desde Body utilizando formato raw

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** `http://localhost/crud/borrar.php?id=114`
- Body (raw):** `{"id": 114}`
- Response Status:** 200 OK
- Response Body:** `114{"info": "Actor borrado!"}`

# 8 POSTMAN

php

- ✓ ACTUALIZAR: Se pasan los parámetros desde Body utilizando formato raw. Obviamente necesita el id para saber sobre qué actor se está haciendo el update.

The screenshot shows the Postman interface for an update request. The URL in the header is `http://localhost/crud/actualizar.php`. The 'Body' tab is selected, showing raw JSON data:

```
1 {  
2   "id": 114,  
3   "nombre": "Zendaya",  
4   "edad": 27,  
5   "activo": 1,  
6 }  
7
```

The response status is 200 OK with a response body:

```
1 {  
2   "info": "Actor/Actriz actualizado"  
3 }  
4
```

## 8 POSTMAN

✓ BIND de Mysqli

Especificación del tipo de caracteres

Carácter	Descripción
i	la variable correspondiente es de tipo entero
d	la variable correspondiente es de tipo double
s	la variable correspondiente es de tipo string
b	la variable correspondiente es un blob y se envía en paquetes

Ahora ya sí, podemos ver el código de la API, volviendo a los apuntes si es necesario