Creación de un servicio REST en PHP paso a paso

Sitio: Aula virtual do IES Pazo da Mercé
Curso: Desenvolvemento Web Contorno Servidor (24-25)
Libro: Creación de un servicio REST en PHP paso a paso

Impreso por: Lara Comesaña Varela
Data: mércores, 15 de xaneiro de 2025, 12:00 PM

Táboa de contidos

1. Redirigir todas las peticiones a index.php

Esta es una posible metodología para cumplir los requisitos. Recuerda que podrías utilizar otras siempre que le funcionamiento sea correcto y que sepas justificar el porqué de tu decisión.

En los diferentes pasos se muestra un ejemplo de como se podría ir codificando la aplicación. En clase, se dará tiempo para la realización del código. Es recomendable que intentes codificar tú una solución y luego veas la propuesta en el curso. Recuerda, en la programación se mejora con la práctica.

Partiremos de la siguiente versión del framework.

Lo primero que debemos hacer es crear dentro de la carpeta public un fichero .htaccess que obligue a redirigir todas las peticiones al fichero index.php

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

Para poder probar esta configuración, creamos en el directorio /etc/apache2/sites-available el fichero 007-api1.conf con el siguiente contenido:

```
<VirtualHost *:80>

    DocumentRoot /ruta/hasta/public
    ServerName api-demo.localhost

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
    <Directory /ruta/hasta/public/>
          Options +Indexes
          AllowOverride All
    </Directory>
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Escribimos ahora los siguientes comandos:

```
sudo a2ensite 007-api1.conf

sudo systemctl restart apache2
```

En la carpeta public, creamos un fichero php con el siguiente contenido

```
<?php

echo $_SERVER['PATH_INFO']
```

Abrimos en el navegador http://api1.localhost/hola/test y debería mostrar por pantalla **/hola/test**

2. Configuración del router

Modificar funcionamiento de FrontController

Cuando trabajamos con servicios rest es tan importante a URL de petición como el método de la misma. Es por ello que debemos ser muy cuidadosos configurando nuestro FrontController.El servicio REST que vamos a configurar es el siguiente:

URL	METHOD	Parámetros	Resultado
/categoria	GET	*Ninguno*	***200***. *Listado con todas las categorías.* ***500***. *Error indeterminado en el lado del servidor.*
/categoría/$id	GET	*Ninguno*	***200***. *Todos los datos de la categoría con id = $id* ***404***. *Si no existe la categoría con id = $id*
/categoria	POST	id_padre: int. Idenficador del padre nombre_categoria: string. Nombre de la categoría. No vacía.	***201***. *En caso de alta satisfactoria. Necesita recibir un id_padre válido o nulo y categoría es string no vacía.*
/categoria/$id	DELETE	*Ninguno*	**200**. Si se borra correctamente **404**. Si no existe la categoría a borrar.
/categoria/$id	PUT	id_padre: int. Idenficador del padre nombre_categoria: string. Nombre de la categoría. No vacía.	**200**. Si se modifica correctamente. **404**. Si no existe la categoría a modificar.

En un servicio REST es muy importante que establezcamos bien las rutas del FrontController y que seamos muy escrupulosos estableciendo el requisito del método de acceso.

3. Creación de vista

Todas las peticiones REST van a devolver las mismas cabeceras y si devuelven información, devolverán esa información en formato json. Cuando llamemos a la vista siempre le vamos a pasar un objeto de la clase Respuesta con un status y el campo data si es necesario. Por lo tanto una posible solución podría ser esta:

/media/rafa/DATOS/htdocs/22_23-DWES/api-demo/app/Views/json.view.php

```php
<?php

/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
//var_dump($respuesta);
header('Access-Control-Allow-Origin: http://localhost');
header("Access-Control-Allow-Headers: X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Request-Method");
header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE");
header("Allow: GET, POST, PUT, DELETE");
http_response_code($respuesta->getStatus());

if($respuesta->hasData()){
    header('Content-Type: application/json; charset=utf-8');
    echo json_encode($respuesta->getData(), TRUE);
}
```

json.view.php

La cabecera **Access-Control-Allow-Origin** contiene las URLs que pueden realizarnos peticiones. Podemos poner * si quisiéramos aceptar todas.

La cabecera **Access-Control-Allow-Headers** especifica las cabeceras que aceptamos en las peticiones.

La cabecera **Access-Control-Allow-Methods** especifica qué methods aceptamos en las peticiones.

Después enviamos el código de respuesta que hayamos obtenido durante la ejecución. Si vamos a enviar contenido, avisamos de que éste va en formato json y lo printamos por pantalla.

Enlace a la vista

4. Codificación del controlador

Codificación del controlador

En este ejemplo vamos a codificar el controlador de categorias. Ofrecerá los siguientes métodos:

URL	METHOD	Parámetros	Resultado
/categoria	GET	*Ninguno*	***200***. *Listado con todas las categorías.* ***500***. *Error indeterminado en el lado del servidor.*
/categoría/$id	GET	*Ninguno*	***200***. *Todos los datos de la categoría con id = $id* ***404***. *Si no existe la categoría con id = $id*
/categoria	POST	id_padre: int. Idenficador del padre nombre_categoria: string. Nombre de la categoría. No vacía.	***201***. *En caso de alta satisfactoria. Necesita recibir un id_padre válido o nulo y categoría es string no vacía.*
/categoria/$id	DELETE	*Ninguno*	**200**. Si se borra correctamente **404**. Si no existe la categoría a borrar.
/categoria/$id	PUT	id_padre: int. Idenficador del padre nombre_categoria: string. Nombre de la categoría. No vacía.	**200**. Si se modifica correctamente. **404**. Si no existe la categoría a modificar.

Si se hace una petición en la que falten parámetros o en la que estos sean incorrectos, se devuelve un código 400.

4.1. Creación de esqueleto para determinar el tipo de petición

Vamos a crear el esqueleto de la aplicación para ellos nos basaremos en una solución parecida a la que se realizamos en la tarea Detectar tipo de petición de la lección anterior. Discriminaremos las peticiones en base al tipo de las mismas GET, POST, PUT...

Primero crearemos un constructor que reciba como parámetro el method de la petición (string) y un parámetro opcional que sería el id de la categoría con la que queremos trabajar (?int).

Posteriormente creamos el método processRequest() que será el punto de entrada para todas las peticiones y que discrimina la petición en base al método.

Una posible codificación podría ser la siguiente.

/home/rafa/www/ud7-api-demo-categoria/app/Core/FrontController.php

```php
<?php

namespace Com\Daw2\Core;

use Steampixel\Route;
use Ahc\Jwt\JWT;
use Com\Daw2\Helpers\JwtTool;

class FrontController {

    static function main() {
        /*
         * Métodos controlador Categoría
         */
        $user = JwtTool::getBearerUser();
        Route::add('/categoria',
                function(){
                    $controller = new \Com\Daw2\Controllers\CategoriaController();
                    $controller->respuestaToJson($controller->getAll());
                },
                'get');
        Route::add('/categoria/([0-9]+)',
                function($id) {
                    $controller = new \Com\Daw2\Controllers\CategoriaController();
                    $controller->respuestaToJson($controller->getCategoria($id));
                },
                'get');
        //Ruta de autenticación
        Route::add('/login',
                fn() => (new \Com\Daw2\Controllers\SessionController('post'))->login(),
                'post');
        //Sólo podemos usar esta rutas si estamos logueados. Ver tema de JWT.
        if(!is_null($user)){
            Route::add('/categoria/([0-9]+)',
                    function($id) {
                        $controller = new \Com\Daw2\Controllers\CategoriaController();
                        $controller->respuestaToJson($controller->deleteCategoria($id));
                    },
                    'delete');
            Route::add('/categoria/([0-9]+)',
                    function($id) {
                        $controller = new \Com\Daw2\Controllers\CategoriaController();
                        $controller->respuestaToJson($controller->editCategoria($id));
                    },
                    'put');
            Route::add('/categoria',
                    function() {
                        $controller = new \Com\Daw2\Controllers\CategoriaController();
                        $controller->respuestaToJson($controller->insertCategoria());
                    },
                    'post');
        }
        //Podemos hacer una versión sin este else si preferimos que el usuario no logueado no sepa que exi
//        else{
//            Route::add('/categoria/([0-9]+)',
//                function() {
//                    $respuesta = new \Com\Daw2\Helpers\Respuesta(403);
//                    require $_ENV['folder.views'].'/json.view.php';
//                },
//                ['delete', 'put']);
//            Route::add('/categoria',
//                    function() {
//                        $respuesta = new \Com\Daw2\Helpers\Respuesta(403);
//                        require $_ENV['folder.views'].'/json.view.php';
//                    },
//                    'post');
        //}

        /*
         * Métodos generales (fallback cuando no encontramos ruta
         */
        Route::pathNotFound(
            function() {
                $respuesta = new \Com\Daw2\Helpers\Respuesta(404);
                require $_ENV['folder.views'].'/json.view.php';
            }
        );
        Route::methodNotAllowed(
            function() {
```

4.2. Creación de métodos GET

Vamos a centrarnos en la codificación de los métodos GET:

URL	METHOD	Parámetros	Resultado
/categoria	GET	*Ninguno*	***200***. *Listado con todas las categorías.* ***500***. *Error indeterminado en el lado del servidor.*
/categoría/$id	GET	*Ninguno*	***200***. *Todos los datos de la categoría con id = $id* ***404***. *Si no existe la categoría con id = $id*

Como podemos observar, todas las peticiones recibirán un código de respuesta y opcionalmente cuerpo de respuesta (JSON) con el contenido. Para facilitar las cosas crearemos en Helpers la **clase Respuesta** que tendrá un constructor (int) y un método que nos permitirá rellenar el cuerpo de la respuesta. Además crearemos el método hasData():bool que nos permitirá saber si la Respuesta tiene contenido en el cuerpo o no.

Ahora debemos discriminar dentro de GET si posición 1 del array de petición contiene un int o no.

- Si no es un int mostraremos un listado como éste de ejemplo. **Fíjate que se muestra la información de los padres de categoría**.
- Si es un int mostraremos una sola categoría o 404 si el id no existe. Ejemplo.

Para poder dar solución a este requisito debemos modificar FrontController poniendo dos reglas, una para las peticiones a las peticiones de la primera fila y otra para las peticiones del tipo de la segunda fila. En las primeras crearemos categoría controller sin id y en las segundas con el id recibido. Posible solución: FrontController.php (GET).

Ahora podemos modificar CategoriaControlle para trabajar con los supuestos, recuerda que es necesario crear CategoriaModel e implementar los métodos necesarios para obtener el listado y la carga de la categoría (puedes revisar en el repositorio si no logras hacer estos métodos) y por último modificar la vista en la que mostraremos los resultados. Esta podría ser una solución para CategoríaController.php

Vídeo explicativo

4.3. Implementación de PUT (Update)

Ahora que tenemos la vista y el modelo ya creado, el trabajo que nos queda para implementar esta operación es muy sencilla. Recordemos los requisitos de nuestro API para delete:

URL	METHOD	Parámetros	Resultado
/categoria/$id	PUT	id_padre: int. Idenficador del padre nombre_categoria: string. Nombre de la categoría. No vacía.	**200**. Si se modifica correctamente. **404**. Si no existe la categoría a modificar.

Como podemos observar, la única petición válida será la petición categoria/id con los parámetros categoría que será una string e id_padre que es opcional pero que si se envía debe ser un int que será el **id de una categoría existente en el sistema**.

Primero creamos la ruta en el FrontController. Fíjate que sólo creamos una ruta en la que obligamos a que nos pasen como parámetro un id (si quieres testear este código sin loguearte comenta `if(!is_null($user)){` y su cierre:

```
40              Route::add('/categoria/([0-9]+)',
41                      function($id) {
42                          $controller = new \Com\Daw2\Controllers\CategoriaController();
43                          $controller->respuestaToJson($controller->editCategoria($id));
44                      },
45                      'put');
```

Los parámetros de una petición PUT no lo recibimos en un array como $_GET o $_POST. Los parámetros de las peticiones PUT se envían en el cuerpo de la petición. Usualmente pueden ser enviados en formato JSON o en formato var1=valor1&var2=valor2... Vamos a crear un método en **BaseController** que cargue el contenido del cuerpo de la petición y en base a si es JSON o no parsee el cuerpo con el método adecuado:

```php
/**
 * Parsea el contenido del body de petición. Usualmente con PUT, PATCH.
 * Si recibe como CONTENT_TYPE = 'application/json' parsea el body como un json. En caso contrario asume formato var1=valor1&var2=valor2
 * Almacena el contenido parseado en atributo privado para tenerlo accesible durante toda la vida de la clase
 * @return array Devuelve un array clave=>valor con los parámetros recibidos en el cuerpo de la petición
 */
private function initBodyData() : array{
    $request = file_get_contents("php://input");
    if(!empty($request)){
        $contentType = $_SERVER["CONTENT_TYPE"] ?? 'plain/text';

        if($contentType === 'application/json'){
            $postVars = json_decode($request, true);
        }
        else{
            parse_str($request,$postVars);
        }
        return $postVars;
    }
    else{
        return array();
    }
}
```

Es recomendable cargar la petición solo una vez. Por eso hemos hecho este método privado y creamos un método público que se encarga de llamar a initBodyData() la primera vez que es llamado y guardar el contenido en un atributo privado para poder devolver ese atributo en las llamadas sucesivas:

```php
/**
 * Devuelve los valores enviados en el cuerpo de una petición. Si recibe como CONTENT_TYPE = 'application/json'
 * parsea el body como un json. En caso contrario asume formato var1=valor1&var2=valor2
 * Con esta función leemos sólo una vez la petición que recibimos ya que si la hemos leído con anterioridad,
 * tendremos contenido en atributo bodyData y no será necesario reparsea la petición
 * @return array Devuelve un array clave=>valor con los parámetros recibidos en el cuerpo de la petición
 */
public function getBodyData() : array{
    if(is_null($this->bodyData)){
        $this->bodyData = $this->initBodyData();
    }
    return $this->bodyData;
}
```

Estado final de BaseController.php

En CategoriaController.php creamos el método editCategoria(int)

Ahora dentro de nuestro código PUT podemos cargar un array con los parámetros recibidos en la petición:

```php
private function editCategoria(int $idcategoria): Respuesta{
    $params = $this->getBodyData();
```

Ahora simplemente debemos comprobar que recibimos el id de la categoría a editar y realizar las comprobaciones necesarias para hacer el update o mostrar los errores pertinentes.

En el modelo, si se ejecuta el update debemos devolver el contenido que tenemos en la base de datos tras el update:

```php
    public function editCategoria(int $idcategoria): Respuesta{
        try{
            $params = $this->getBodyData();
            $model =  new \Com\Daw2\Models\CategoriaModel();
            $categoriaEditar = $model->loadCategoria($idcategoria);

            //Si la categoría no existe, debemos devolver un 404
            if(is_null($categoriaEditar)){
                $respuesta = new Respuesta(404);
            }
            //Si existe la categoría
            else{
                if($this->checkInsertUpdateParams($params)){
                    $idPadre = isset($params['id_padre']) ? (int)$params['id_padre'] : null;
                    $data = $model->updateCategoria($idcategoria, $params['nombre_categoria'],$idPadre);
                    $respuesta = new Respuesta(200);
                    $respuesta->setData($data);
                }
                else{
                    //Parámetros pasados no son correctos
                    $respuesta = new Respuesta(400);
                }
            }
            return $respuesta;
        }
        catch(\PDOException $ex){
            $respuesta = new Respuesta(500);
            throw $ex;
        }
        catch(\Exception $ex){
            $respuesta = new Respuesta(500);
        }
    }
```

El resultado de esta respuesta se muestra en el json.view.php. Para no repetir código tenemos respuestoToJson definido en CategoriaController. éste método se debería subir a una superclase si va a haber más Controladores que respondan en formato json:

```php
public function respuestaToJson(Respuesta $respuesta){
    $this->view->show('json.view.php', array('respuesta' => $respuesta));
}
```

4.4. Resumen pasos realizados

En el siguiente vídeo hacemos un breve repaso de los pasos que hemos seguido para realizar nuestros primeros métodos en un API Rest:

https://youtu.be/9uojY0UrF7U

Se enlaza también el resto del API para categoría finalizado.