cursos.mejorcodigo.net

Crear REST API con Slim - PHP y MySQL

Mejor Código

10-13 minutu

Última actualización: 10-12-2017

Actualmente existe un gran numero de frameworks disponibles para poder crear un REST API. Para este artículo se escogió Slim. Slim en un micro framework que te permite escribir aplicaciones web y API's de forma rápida y eficiente.

Así mismo, en este artículo, crearemos el REST API utilizando MySQL. Veremos el servicio REST con ejemplos de todos los métodos disponibles como lo son: POST, GET, PUT y DELETE. La respuesta que nos devolverá el REST API será en formato JSON.

Con esto dicho, vamos a empezar.

Métodos HTTP:

- GET: se utiliza para retornar y buscar información en la base de

datos.

- POST: se utiliza para insertar información a la base de datos.
- PUT: se utiliza para actualizar información en la base de datos.
- DELETE: se utiliza para eliminar información de la base de datos.

Como configurar Slim en XAMPP ó WAMP

Para configurar Slim necesitas descargar el framework. Una vez descargado lo descomprimes y lo renombras de `Slim-Skeleton-master` a `mi-api`. Ahora copiamos el framework y lo pegamos en `/xampp/htdocs` en el caso de XAMPP ó `/wamp/www` en el caso de WAMP.

Ahora tenemos una estructura de proyecto así:

`/xampp/htdocs/mi-api` en el caso de XAMPP

`/wamp/www/mi-api` en el caso de WAMP

Para este artículo asumiremos que tienes instalado Composer. Ahora abrimos la terminal (cmd) y navegamos al directorio de nuestro proyecto.

```
# cd /xampp/htdocs/mi-api //En caso de xampp
```

ó

```
# cd /wamp/www/mi-api //En caso de wamp
```

Y ejecutamos el siguiente comando:

```
# composer update
```

Creamos Un Host Vitual en XAMPP

Paso 1: Abrimos el archivo `C:\XAMPP\apache\conf\extra\httpd-vhosts.conf` y agregamos las siguientes lineas:

```
<VirtualHost *:80>

    DocumentRoot "C:\XAMPP\htdocs\mi-api\public"

    ServerName mi-api
```

```
  <Directory "C:\XAMPP\htdocs\mi-api\public">

    Order allow,deny

    Allow from all

  </Directory>

</VirtualHost>
```

También debemos descomentar la linea `NameVirtualHost *:80` de este archivo.

Nota: cambia el disco C al disco donde tengas instalado XAMPP. Si tienes instalado XAMPP en el disco local C ignora esta nota.

Paso 2: Ahora abrimos el archivo `C:\Windows\System32\drivers\etc\hosts` y agregamos la siguiente linea:

```
127.0.0.1       mi-api
```

Si abrimos nuestro navegador a `http://mi-api` nos debe redireccionar a la página de bienvenida de Slim (archivo `index.php` dentro de `C:\XAMPP\htdocs\mi-api\public`).

Conexión A MySQL Desde Slim

Primero debemos crear la base de datos. En mi caso la nombraré `api_db`. Ahora en phpMyAdmin nos metemos a la sección de SQL en la parte superior e ingresamos la siguiente consulta:

```
CREATE TABLE IF NOT EXISTS `empleados` (

  `id` int(11) NOT NULL COMMENT 'primary key',

  `nombre` varchar(255) NOT NULL COMMENT
'employee name',

  `salario` double NOT NULL COMMENT 'employee
```

```
salary',

  `edad` int(11) NOT NULL COMMENT 'employee age'

) ENGINE=InnoDB AUTO_INCREMENT=58 DEFAULT
CHARSET=latin1 COMMENT='datatable demo table';
```

Crearemos la conexión en el archivo `index.php`. Ahora abrimos el archivo `mi-api/public/index.php` y agregamos lo siguiente:

```
function getConnection() {

    $dbhost="127.0.0.1";

    $dbuser="root";

    $dbpass="";

    $dbname="api_db";

    $dbh = new PDO("mysql:host=$dbhost;
dbname=$dbname", $dbuser, $dbpass);

    $dbh->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    return $dbh;

}
```

Como ya sabemos estos son los parámetros de la conexión:

- **dbhost**: es la dirección ip o dominio donde se encuentra alojada nuestra base de datos.
- **dbuser**: es el nombre de usuario para poder conectarnos a la base de datos. Si trabajas en un servidor local, el nombre de usuario por lo general es `root`.

- **dbpass**: es la contraseña ligada al usuario. Si trabajas en un servidor local, la contraseña por lo general esta vacia.
- **dbname**: el nombre de la base de datos a la que nos queremos conectar. En nuestro caso es `api_db`.

  Hasta ahorita si abrimos `http://api/` y enseña la página de bienvenida de Slim todo va bien y ya estas conectado a la base de datos. En otro caso, saldrá un error.

Servicio REST con HTTP

Nosotros crearemos el servicio REST con GET, POST, PUT y DELETE. A continuación se muestra la tabla de funcionamiento:

# Ruta	Método	Tipo	Ruta Completa	Descripción
1 /empleados	GET	JSON	http://tudominio.com /api/v1/empleados/	Obtener información de todos los empleados.
2 /empleado/{id}	GET	JSON	http://tudominio.com /api/v1/empleado/1	Obtener información de un solo empleado.
3 /crear	POST	JSON	http://tudominio.com /api/v1/crear/	Crear un nuevo empleado en la base de datos.
4 /actualizar/{id}	PUT	JSON	http://tudominio.com /api/v1/actualizar/1	Actualizar información de un empleado.
5 /eliminar/{id}	DELETE	JSON	http://tudominio.com /api/v1/eliminar/1	Eliminar información de un empleado.

Crear Rutas HTTP En Slim

Debemos crear las rutas de la tabla anterior en Slim. Para esto abrimos el archivo `mi-api/src/routes.php` y escribimos las siguientes rutas:

```
 // Routes

// Grupo de rutas para el API

$app->group('/api', function () use ($app) {

  // Version group

  $app->group('/v1', function () use ($app) {

    $app->get('/empleados', 'obtenerEmpleados');

    $app->get('/empleado/{id}',
'obtenerEmpleado');

    $app->post('/crear', 'agregarEmpleado');

    $app->put('/actualizar/{id}',
'actualizarEmpleado');

    $app->delete('/eliminar/{id}',
'eliminarEmpleado');

  });

});
```

En las rutas creamos dos grupos `/api` y `/v1`. Esto lo hicimos ya que podrá ser útil a un futuro para manejar diferentes API's ó manejar diferentes versiones de las API's existentes.

Así mismo, dentro de nuestras rutas estalecemos que función será ejecutada cuando haya una petición recibida. Por ejemplo, una petición GET a `/empleados` llamará la función `obtenerEmpleados()`.

Obtener Todos Los Empleado de la Base de Datos

Crearemos la function `obtenerEmpleados()` en el archivo `mi-api/public/index.php`. Esta función retornará un objeto JSON con todos los empleados de la base de datos.

```
function obtenerEmpleados($response) {

    $sql = "SELECT * FROM empleados";

    try {

        $stmt = getConnection()->query($sql);

        $employees =
$stmt->fetchAll(PDO::FETCH_OBJ);

        $db = null;


        return json_encode($employees);


    } catch(PDOException $e) {

        echo '{"error":{"text":'.
$e->getMessage() .'}}';

    }

}
```

Ahora si enviamos una petición GET a `http://mi-api/api/v1/empleados` nos mostrará un objeto JSON con todos los empleados registrados.

Agregar Empleado a la Base de Datos

Crearemos un nuevo método para poder agregar empleados una

vez que recibamos una petición POST.

```
function agregarEmpleado($request) {

    $emp = json_decode($request->getBody());


    $sql = "INSERT INTO empleados (nombre,
salario, edad) VALUES (:nombre, :salario,
:edad)";



    try {

        $db = getConnection();

        $stmt = $db->prepare($sql);

        $stmt->bindParam("nombre", $emp->nombre);

        $stmt->bindParam("salario",
$emp->salario);

        $stmt->bindParam("edad", $emp->edad);

        $stmt->execute();

        $emp->id = $db->lastInsertId();

        $db = null;

        echo json_encode($emp);

    } catch(PDOException $e) {

        echo '{"error":{"text":'.
$e->getMessage() .'}}';
```

```
    }

}
```

Ahora si realizamos una petición POST a `http://mi-api/api/v1/crear` utilizando cualquier cliente REST como Postman, nos retornará la información del empleado agregado en formato JSON.

Actualizar Empleado en la Base de Datos

Ahora crearemos un método para una petición PUT. Esta nos servirá para actualizar información de un empleado existente.

```
function actualizarEmpleado($request) {

    $emp = json_decode($request->getBody());

    $id = $request->getAttribute('id');

    $sql = "UPDATE empleados SET nombre=:nombre,
salario=:salario, edad=:edad WHERE id=:id";

    try {

        $db = getConnection();

        $stmt = $db->prepare($sql);

        $stmt->bindParam("nombre", $emp->nombre);

        $stmt->bindParam("salario",
$emp->salario);

        $stmt->bindParam("edad", $emp->edad);

        $stmt->bindParam("id", $id);

        $stmt->execute();
```

```
        $db = null;

        echo json_encode($emp);

    } catch(PDOException $e) {

        echo '{"error":{"text":'.
$e->getMessage() .'}}';

    }

}
```

Ahora podemos enviar una petición PUT a `http://mi-api/api/v1/actualizar/1` con Postman y nos actualizara el empleado con el ID correspondiente e imprimira la información actualizada del empleado en formato JSON.

Eliminar Empleado de la Base de Datos

El último método que debemos crear es para eliminar la información de un empleado con el método DELETE.

```
function eliminarEmpleado($request) {

    $id = $request->getAttribute('id');

    $sql = "DELETE FROM empleados WHERE id=:id";

    try {

        $db = getConnection();

        $stmt = $db->prepare($sql);

        $stmt->bindParam("id", $id);

        $stmt->execute();
```

```
        $db = null;

        echo '{"error":{"text":"se elimino el
empleado"}}';

    } catch(PDOException $e) {

        echo '{"error":{"text":'.
$e->getMessage() .'}}';

    }

}
```

Ahora si envias una petición DELETE a `http://mi-api/api/v1/eliminar/1` se eliminará el empleado con la ID correspondiente.

Arreglar Errores Cross-Origin

Es posible que podamos tener un error cross-origin (solicitud HTTP de origen cruzado) ya que estaremos realizando nuestras peticiones, GET, POST, PUT y DELETE, desde otros servidores. Para resolver este error debemos permitir el uso de estos métodos desde diferentes servidores. Para esto tendremos que hacer uso del plugin `corsslim`. Simplemente incluimos `"palanik/corsslim": "dev-slim3"` en nuestro archivo de composer y ejecutamos el comando `composer update` en el cmd. Ahora escribimos la configuración para este plugin en el archivo `mi-api/public/index.php`.

```
$corsOptions = array(

    "origin" => "*",

    "exposeHeaders" => array("Content-Type", "X-
Requested-With", "X-authentication", "X-client"),
```

```
    "allowMethods" => array('GET', 'POST', 'PUT',
'DELETE', 'OPTIONS')

);

$cors = new \CorsSlim\CorsSlim($corsOptions);
```

¡Y LISTO! Ya estas listo para hacer uso de tu REST API hecha con Slim. Puedes leer más acerca de REST API's aquí.

Etiquetas

Slim Framework PHP MySQL REST API

¿Te gustó el articulo? Ayudanos compartiendo.

Compartir en Facebook Compartir en Twitter