



API REST CON PHP

## 1.0 DESCRIPCIÓN DEL PROYECTO

### 1.1 ¿Qué es la API REST?

Para definir "REST API", primero debemos saber qué es "REST" y qué es "API". Haré todo lo posible para explicarlo en términos simples porque REST tiene muchos conceptos dentro de él que [podrían significar muchas cosas](https://stackoverflow.com/questions/4663927/what-is-rest-slightly-confused) .

**REST** significa "Transferencia de estado representativa". Es un concepto o arquitectura para gestionar información a través de Internet. Los conceptos REST se denominan recursos. Una representación de un recurso debe ser apátrida. Generalmente está representado por [JSON](https://stackoverflow.com/questions/383692/what-is-json-and-why-would-i-use-it) . Vale la pena leer esta publicación: [¿Cómo le expliqué REST a mi esposa?](http://www.looah.com/source/view/2284)

**API** significa "interfaz de programación de aplicaciones". Es un conjunto de reglas que permite que una aplicación de software se comunique con otra. Esas "reglas" pueden incluir operaciones de creación, lectura, actualización y eliminación. Si desea obtener más información, mire el video a continuación y lea la [reseña de la cámara de YouTube de musiccritic](http://musiccritic.com/equipment/best-camera-for-youtube/) si está interesado en hacer algunos videos.

**REST API** permite que su aplicación coopere con una o varias aplicaciones diferentes utilizando conceptos REST. Si desea obtener más información, mire el video a continuación.

### 1.2 ¿Por qué necesitamos API REST?

En muchas aplicaciones, REST API es una necesidad porque esta es la forma más ligera de crear, leer, actualizar o eliminar información entre diferentes aplicaciones a través de Internet o el protocolo HTTP. Esta información se presenta al usuario en un instante, especialmente si utiliza JavaScript para representar los datos en una página web.

### 1.3 ¿Dónde se utiliza la API REST?

REST API puede ser utilizado por cualquier aplicación que pueda conectarse a internet. Si los datos de una aplicación se pueden crear, leer, actualizar o eliminar utilizando otra aplicación, generalmente significa que se utiliza una API REST.

### 1.4 REST API en nuestros tutoriales

Se necesita una API REST para nuestro [Tutorial AJAX CRUD](https://www.codeofaninja.com/2015/06/php-crud-with-ajax-and-oop.html) . Pero no te preocupes por ahora. Lo haremos paso a paso. No necesitas aprenderlo todo también. Simplemente elige lo que necesitas aprender.

Además, tenga en cuenta que esta API REST de PHP aún no está en su forma final. Todavía tenemos trabajo que hacer con .htaccess para obtener mejores URL y más.

Pero una cosa es segura, este código fuente es lo suficientemente bueno y funciona para nuestros tutoriales de JavaScript.

## ESTRUCTURA DE ARCHIVO 2.0

Al final de este tutorial, tendremos las siguientes carpetas y archivos.  
├─ api /  
├─── config /  
├────── core.php: archivo utilizado para la configuración principal  
├────── database.php: archivo utilizado para conectarse a la base de datos.  
├─── objetos /  
├────── producto.php - contiene propiedades y métodos para consultas de bases de datos "producto".  
├────── category.php: contiene propiedades y métodos para consultas de bases de datos de "categorías".  
├─── product /  
├────── create.php: archivo que aceptará los datos publicados del producto para guardarlos en la base de datos.  
├────── delete.php: archivo que aceptará una ID de producto para eliminar un registro de base de datos.  
├────── read.php: archivo que generará datos JSON basados ​​en registros de la base de datos de "productos".  
├────── read\_paging.php: archivo que generará datos JSON de "productos" con paginación.  
├────── read\_one.php: archivo que aceptará la identificación del producto para leer un registro de la base de datos.  
├────── update.php: archivo que aceptará una ID de producto para actualizar un registro de base de datos.  
├────── search.php: archivo que aceptará el parámetro de palabras clave para buscar en la base de datos de "productos".  
├─── category /  
├────── read.php: archivo que generará datos JSON basados ​​en los registros de la base de datos de "categorías".  
├─── shared /  
├────── utilities.php: archivo que devolverá la matriz de paginación.

## 3.0 CONFIGURAR LA BASE DE DATOS

Usando PhpMyAdmin, cree una nueva base de datos api\_db . Sí, api\_db es el nombre de la base de datos. Después de eso, ejecute las siguientes consultas SQL para crear nuevas tablas con datos de muestra.

### 3.1 Crear tabla de categorías

|  |
| --- |
| CREATE TABLE IF NOT EXISTS `categories` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `name` varchar(256) NOT NULL,    `description` text NOT NULL,    `created` datetime NOT NULL,    `modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,    PRIMARY KEY (`id`)  ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=19 ; |
|  |

### 3.2 Datos de volcado para la tabla de categorías

|  |
| --- |
| INSERT INTO `categories` (`id`, `name`, `description`, `created`, `modified`) VALUES  (1, 'Fashion', 'Category for anything related to fashion.', '2014-06-01 00:35:07', '2014-05-30 17:34:33'),  (2, 'Electronics', 'Gadgets, drones and more.', '2014-06-01 00:35:07', '2014-05-30 17:34:33'),  (3, 'Motors', 'Motor sports and more', '2014-06-01 00:35:07', '2014-05-30 17:34:54'),  (5, 'Movies', 'Movie products.', '0000-00-00 00:00:00', '2016-01-08 13:27:26'),  (6, 'Books', 'Kindle books, audio books and more.', '0000-00-00 00:00:00', '2016-01-08 13:27:47'),  (13, 'Sports', 'Drop into new winter gear.', '2016-01-09 02:24:24', '2016-01-09 01:24:24'); |

### 3.3 Tabla de productos

|  |
| --- |
| CREATE TABLE IF NOT EXISTS `products` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `name` varchar(32) NOT NULL,    `description` text NOT NULL,    `price` decimal(10,0) NOT NULL,    `category\_id` int(11) NOT NULL,    `created` datetime NOT NULL,    `modified` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,    PRIMARY KEY (`id`)  ) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO\_INCREMENT=65 ; |

### 3.4 Datos de volcado para la tabla de productos

|  |
| --- |
| INSERT INTO `products` (`id`, `name`, `description`, `price`, `category\_id`, `created`, `modified`) VALUES  (1, 'LG P880 4X HD', 'My first awesome phone!', '336', 3, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (2, 'Google Nexus 4', 'The most awesome phone of 2013!', '299', 2, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (3, 'Samsung Galaxy S4', 'How about no?', '600', 3, '2014-06-01 01:12:26', '2014-05-31 17:12:26'),  (6, 'Bench Shirt', 'The best shirt!', '29', 1, '2014-06-01 01:12:26', '2014-05-31 02:12:21'),  (7, 'Lenovo Laptop', 'My business partner.', '399', 2, '2014-06-01 01:13:45', '2014-05-31 02:13:39'),  (8, 'Samsung Galaxy Tab 10.1', 'Good tablet.', '259', 2, '2014-06-01 01:14:13', '2014-05-31 02:14:08'),  (9, 'Spalding Watch', 'My sports watch.', '199', 1, '2014-06-01 01:18:36', '2014-05-31 02:18:31'),  (10, 'Sony Smart Watch', 'The coolest smart watch!', '300', 2, '2014-06-06 17:10:01', '2014-06-05 18:09:51'),  (11, 'Huawei Y300', 'For testing purposes.', '100', 2, '2014-06-06 17:11:04', '2014-06-05 18:10:54'),  (12, 'Abercrombie Lake Arnold Shirt', 'Perfect as gift!', '60', 1, '2014-06-06 17:12:21', '2014-06-05 18:12:11'),  (13, 'Abercrombie Allen Brook Shirt', 'Cool red shirt!', '70', 1, '2014-06-06 17:12:59', '2014-06-05 18:12:49'),  (26, 'Another product', 'Awesome product!', '555', 2, '2014-11-22 19:07:34', '2014-11-21 20:07:34'),  (28, 'Wallet', 'You can absolutely use this one!', '799', 6, '2014-12-04 21:12:03', '2014-12-03 22:12:03'),  (31, 'Amanda Waller Shirt', 'New awesome shirt!', '333', 1, '2014-12-13 00:52:54', '2014-12-12 01:52:54'),  (42, 'Nike Shoes for Men', 'Nike Shoes', '12999', 3, '2015-12-12 06:47:08', '2015-12-12 05:47:08'),  (48, 'Bristol Shoes', 'Awesome shoes.', '999', 5, '2016-01-08 06:36:37', '2016-01-08 05:36:37'),  (60, 'Rolex Watch', 'Luxury watch.', '25000', 1, '2016-01-11 15:46:02', '2016-01-11 14:46:02'); |

### 3.5 Conectarse a la base de datos

El siguiente código muestra las credenciales de la base de datos y un método para obtener una conexión de base de datos mediante PDO. Si aún no está familiarizado con PDO, aprenda primero de nuestro [Tutorial PHP OOP CRUD](https://www.codeofaninja.com/2014/06/php-object-oriented-crud-example-oop.html) .

Crea una carpeta api . Abra la carpeta de la API . Crear carpeta de configuración . Abra la carpeta de configuración . Crear archivo database.php . Coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  class Database{        // specify your own database credentials      private $host = "localhost";      private $db\_name = "api\_db";      private $username = "root";      private $password = "";      public $conn;        // get the database connection      public function getConnection(){            $this->conn = null;            try{              $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db\_name, $this->username, $this->password);              $this->conn->exec("set names utf8");          }catch(PDOException $exception){              echo "Connection error: " . $exception->getMessage();          }            return $this->conn;      }  }  ?> |

## 4.0 LEER PRODUCTOS

### 4.1 Objeto del producto

El siguiente código muestra una clase llamada Producto con varias de sus propiedades. También muestra un método de constructor que aceptará la conexión de la base de datos.

Utilizaremos esta clase para leer datos de la base de datos. Abra la carpeta de la API . Crear carpeta de objetos . Abrir carpeta de objetos . Crear archivo product.php . Coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  class Product{        // database connection and table name      private $conn;      private $table\_name = "products";        // object properties      public $id;      public $name;      public $description;      public $price;      public $category\_id;      public $category\_name;      public $created;        // constructor with $db as database connection      public function \_\_construct($db){          $this->conn = $db;      }  }  ?> |

### 4.2 Crear archivo para leer productos

El siguiente código muestra encabezados sobre quién puede leer este archivo y qué tipo de contenido devolverá.

En este caso, nuestro archivo read.php puede ser leído por cualquier persona (asterisco \* significa todo) y devolverá datos en [formato JSON](https://www.json.org/) .

Abra la carpeta de la API . Crear carpeta de producto . Abra la carpeta del producto . Crea el archivo read.php . Coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");    // database connection will be here |

### 4.3 Conectarse a la base de datos y tabla de productos

En el siguiente código, incluimos los archivos database.php y product.php . Estos son los archivos que creamos anteriormente.

Necesitamos usar el método getConnection () de la clase Database para obtener una conexión a la base de datos. Pasamos esta conexión a la clase de Producto .

Reemplazar // la conexión de la base de datos será aquí comentario del archivo read.php con el siguiente código.

|  |
| --- |
| // include database and object files  include\_once '../config/database.php';  include\_once '../objects/product.php';    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    // initialize object  $product = new Product($db);    // read products will be here |

### 4.4 Leer productos de la base de datos

En el siguiente código, utilizamos el método read () de la clase Product para leer datos de la base de datos. A través de la variable $ num , verificamos si se encontraron registros.

Si hay encontraron registros, el ciclo que recorre usando el tiempo de bucle, añadir cada registro a la $ products\_arr matriz, establecer un 200 OK código de respuesta y mostrarla al usuario en formato JSON.

Reemplazar // leer productos estará aquí comentario del archivo read.php con el siguiente código.

|  |
| --- |
| // query products  $stmt = $product->read();  $num = $stmt->rowCount();    // check if more than 0 record found  if($num>0){        // products array      $products\_arr=array();      $products\_arr["records"]=array();        // retrieve our table contents      // fetch() is faster than fetchAll()      // <http://stackoverflow.com/questions/2770630/pdofetchall-vs-pdofetch-in-a-loop>      while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){          // extract row          // this will make $row['name'] to          // just $name only          extract($row);            $product\_item=array(              "id" => $id,              "name" => $name,              "description" => html\_entity\_decode($description),              "price" => $price,              "category\_id" => $category\_id,              "category\_name" => $category\_name          );            array\_push($products\_arr["records"], $product\_item);      }        // set response code - 200 OK      http\_response\_code(200);        // show products data in json format      echo json\_encode($products\_arr);  }    // no products found will be here |

### 4.5 Agregar el método "read ()" del producto

Utilizamos el método read () en la sección anterior, pero aún no existe en la clase Product . Necesitamos agregar este método read () . El siguiente código muestra la consulta para obtener registros de la base de datos.

Abrir carpeta de objetos . Abra el archivo product.php . Coloque el siguiente código dentro de la clase de Producto . Para asegurarse de haberlo agregado correctamente, coloque el código antes de la última llave de cierre.

|  |
| --- |
| // read products  function read(){        // select all query      $query = "SELECT                  c.name as category\_name, p.id, p.name, p.description, p.price, p.category\_id, p.created              FROM                  " . $this->table\_name . " p                  LEFT JOIN                      categories c                          ON p.category\_id = c.id              ORDER BY                  p.created DESC";        // prepare query statement      $stmt = $this->conn->prepare($query);        // execute query      $stmt->execute();        return $stmt;  } |
|  |

### 4.6 Indique al usuario que no se encontraron productos

Si la variable $ num tiene un valor de cero o negativo, significa que no hay registros devueltos desde la base de datos. Necesitamos decirle al usuario sobre esto.

En el siguiente código, configuramos el código de respuesta en 404: no encontrado y un mensaje que dice No se encontraron productos.

Reemplazar // no se encontraron productos, aquí estará el comentario del archivo read.php con el siguiente código.

|  |
| --- |
| else{        // set response code - 404 Not found      http\_response\_code(404);        // tell the user no products found      echo json\_encode(          array("message" => "No products found.")      );  } |

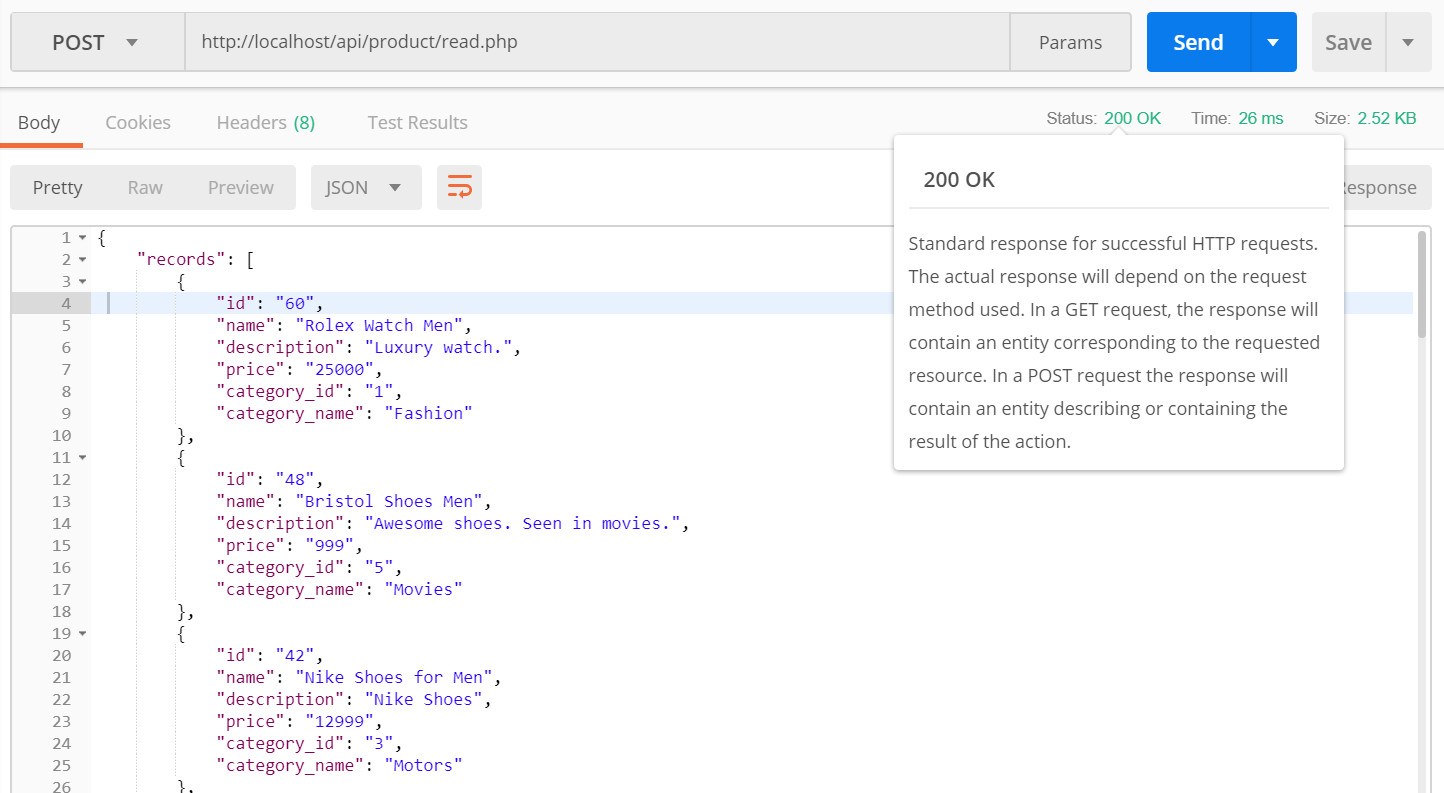
### 4.7 Salida

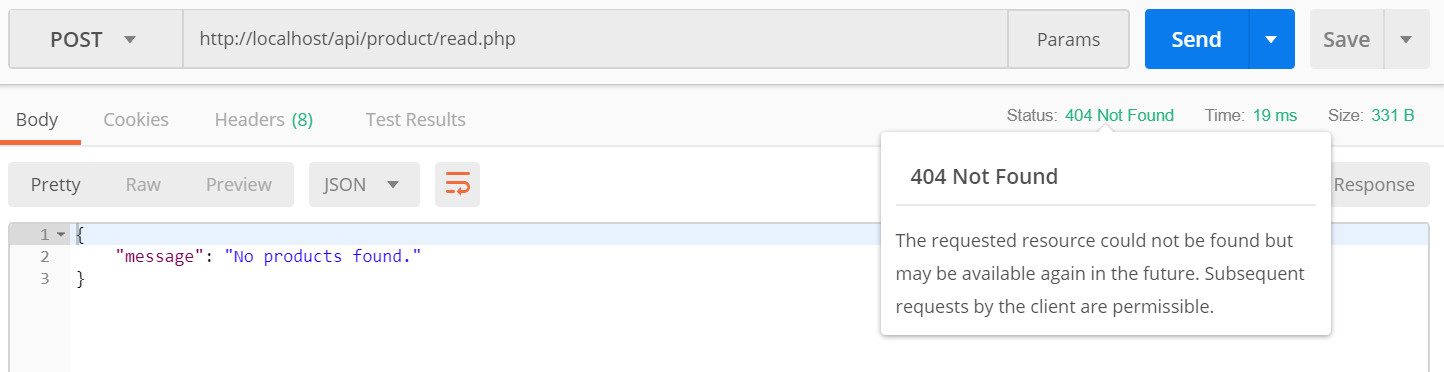
[Debe](https://www.getpostman.com/) usar [POSTMAN](https://www.getpostman.com/) para probar nuestra API. Descargue su versión de POSTMAN [aquí](https://www.getpostman.com/apps) .

Lanzamiento POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/product/read.php](http://localhost/api/product/read.php)

Haga clic en el botón azul "Enviar".

Salida si hay datos del producto.  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-products-200.jpg?ssl=1)

Salida si no hay datos del producto.  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-products-404.jpg?ssl=1)

## 5.0 CREAR PRODUCTO

### 5.1 Crear archivo create.php

Abra la carpeta del producto . Crea un nuevo archivo create.php . Abra ese archivo y coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // get database connection  include\_once '../config/database.php';    // instantiate product object  include\_once '../objects/product.php';    $database = new Database();  $db = $database->getConnection();    $product = new Product($db);    // get posted data  $data = json\_decode(file\_get\_contents("php://input"));    // make sure data is not empty  if(      !empty($data->name) &&      !empty($data->price) &&      !empty($data->description) &&      !empty($data->category\_id)  ){        // set product property values      $product->name = $data->name;      $product->price = $data->price;      $product->description = $data->description;      $product->category\_id = $data->category\_id;      $product->created = date('Y-m-d H:i:s');        // create the product      if($product->create()){            // set response code - 201 created          http\_response\_code(201);            // tell the user          echo json\_encode(array("message" => "Product was created."));      }        // if unable to create the product, tell the user      else{            // set response code - 503 service unavailable          http\_response\_code(503);            // tell the user          echo json\_encode(array("message" => "Unable to create product."));      }  }    // tell the user data is incomplete  else{        // set response code - 400 bad request      http\_response\_code(400);        // tell the user      echo json\_encode(array("message" => "Unable to create product. Data is incomplete."));  }  ?> |

### 5.2 Método create () del producto

Abrir carpeta de objetos . Abra el archivo product.php . La sección anterior no funcionará sin el siguiente código dentro de la clase Product (objects / product.php).

|  |
| --- |
| // create product  function create(){        // query to insert record      $query = "INSERT INTO                  " . $this->table\_name . "              SET                  name=:name, price=:price, description=:description, category\_id=:category\_id, created=:created";        // prepare query      $stmt = $this->conn->prepare($query);        // sanitize      $this->name=htmlspecialchars(strip\_tags($this->name));      $this->price=htmlspecialchars(strip\_tags($this->price));      $this->description=htmlspecialchars(strip\_tags($this->description));      $this->category\_id=htmlspecialchars(strip\_tags($this->category\_id));      $this->created=htmlspecialchars(strip\_tags($this->created));        // bind values      $stmt->bindParam(":name", $this->name);      $stmt->bindParam(":price", $this->price);      $stmt->bindParam(":description", $this->description);      $stmt->bindParam(":category\_id", $this->category\_id);      $stmt->bindParam(":created", $this->created);        // execute query      if($stmt->execute()){          return true;      }        return false;    } |

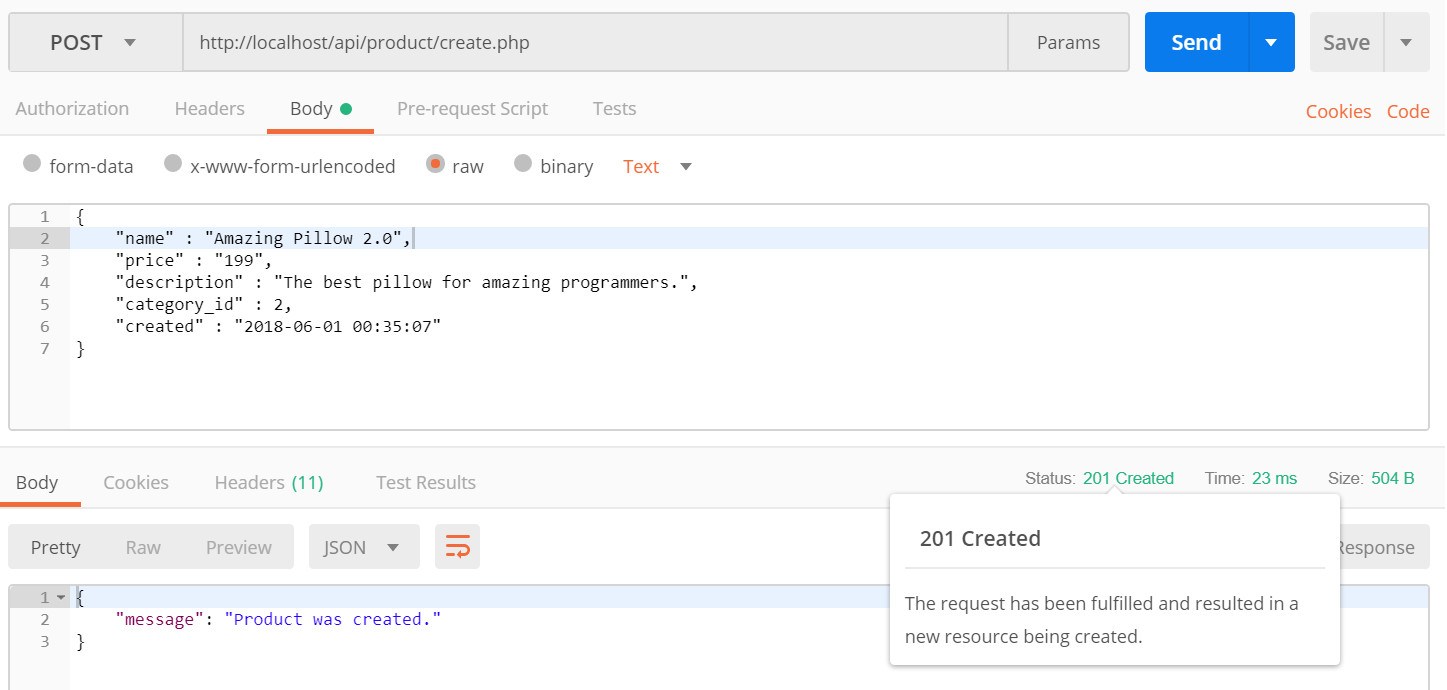
### 5.3 Salida

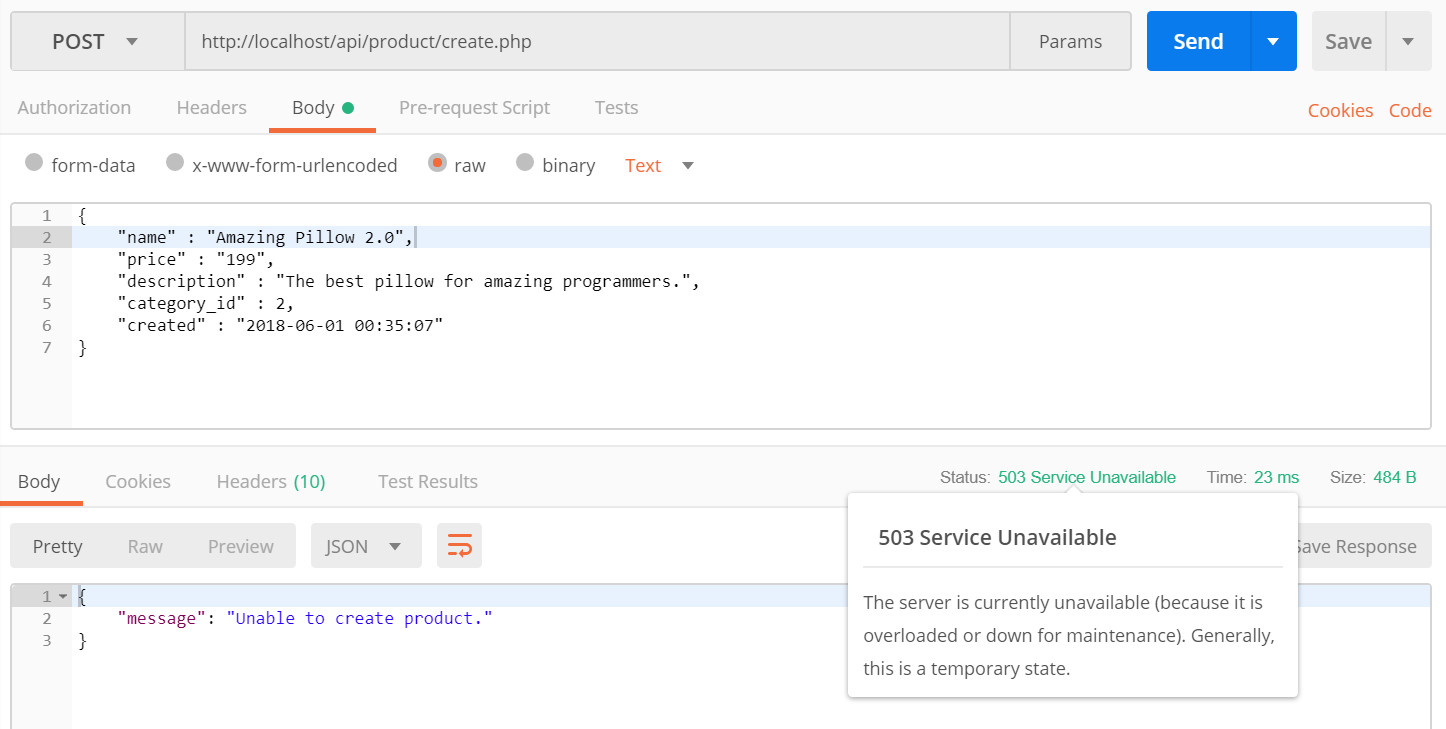
Para probar la creación exitosa de un producto, abra POSTMAN. Ingrese lo siguiente como la URL de solicitud

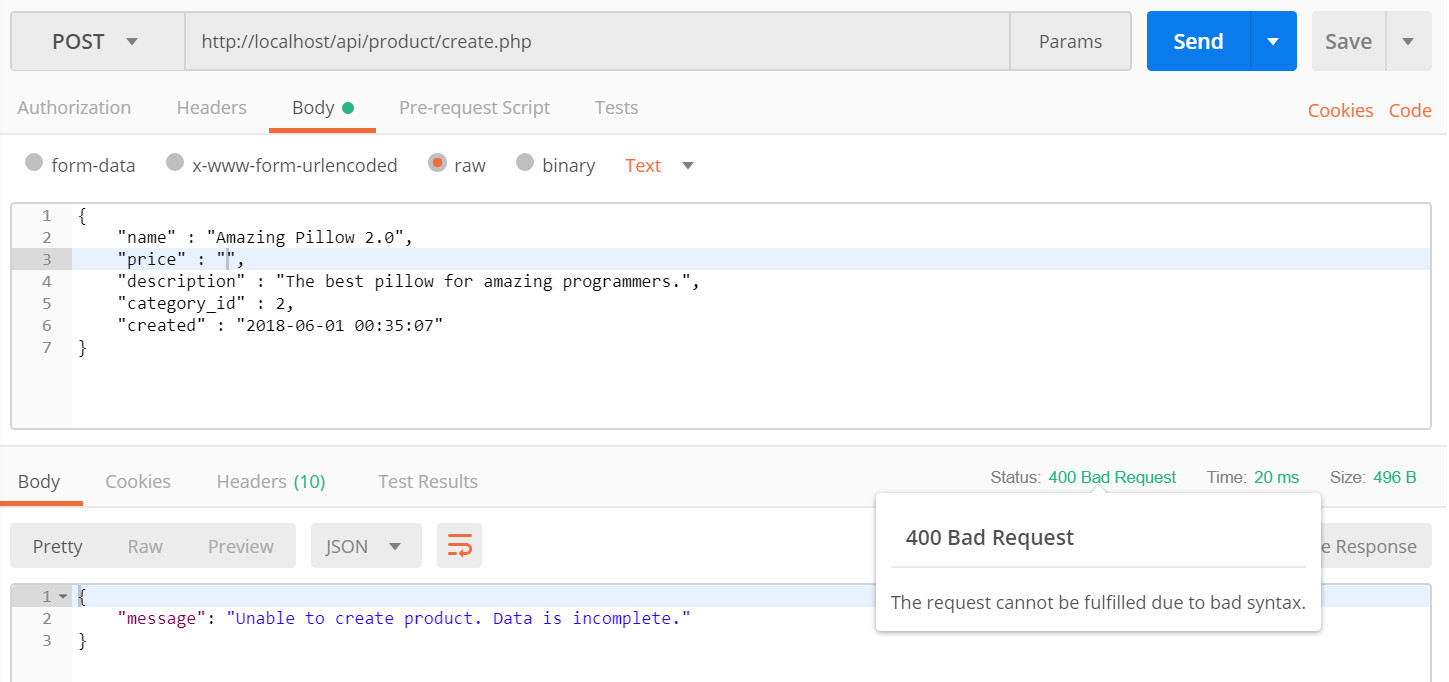
[http: //localhost/api/product/create.php](http://localhost/api/product/create.php)

Haga clic en la pestaña "Cuerpo". Haga clic en "crudo". Ingrese este valor JSON:

|  |
| --- |
| {      "name" : "Amazing Pillow 2.0",      "price" : "199",      "description" : "The best pillow for amazing programmers.",      "category\_id" : 2,      "created" : "2018-06-01 00:35:07"  } |

Debe tener un aspecto como este:  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/create-product-201.jpg?ssl=1)

Si el sistema no puede crear el producto, debería verse así:  
[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/create-product-503.jpg?ssl=1)

Si los datos enviados están incompletos, por ejemplo, faltan los datos del precio, la salida debería verse así:  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/create-product-400.jpg?ssl=1)

## 6.0 LEER UN PRODUCTO

### 6.1 Crear archivo read\_one.php

Abra la carpeta del producto . Crea un nuevo archivo read\_one.php . Abra ese archivo y ponga el siguiente código.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Access-Control-Allow-Headers: access");  header("Access-Control-Allow-Methods: GET");  header("Access-Control-Allow-Credentials: true");  header('Content-Type: application/json');    // include database and object files  include\_once '../config/database.php';  include\_once '../objects/product.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // prepare product object  $product = new Product($db);    // set ID property of record to read  $product->id = isset($\_GET['id']) ? $\_GET['id'] : die();    // read the details of product to be edited  $product->readOne();    if($product->name!=null){      // create array      $product\_arr = array(          "id" =>  $product->id,          "name" => $product->name,          "description" => $product->description,          "price" => $product->price,          "category\_id" => $product->category\_id,          "category\_name" => $product->category\_name        );        // set response code - 200 OK      http\_response\_code(200);        // make it json format      echo json\_encode($product\_arr);  }    else{      // set response code - 404 Not found      http\_response\_code(404);        // tell the user product does not exist      echo json\_encode(array("message" => "Product does not exist."));  }  ?> |

### 6.2 Método readOne () del producto

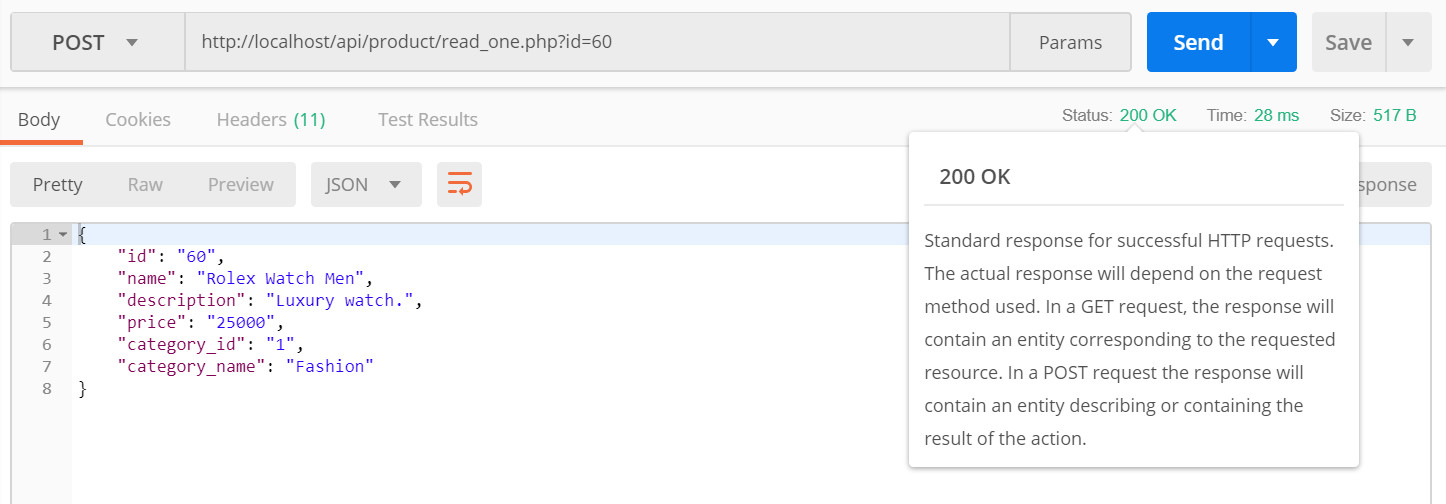
Abrir carpeta de objetos . Abra el archivo product.php . La sección anterior no funcionará sin el siguiente código dentro de la clase de Producto .

|  |
| --- |
| // used when filling up the update product form  function readOne(){        // query to read single record      $query = "SELECT                  c.name as category\_name, p.id, p.name, p.description, p.price, p.category\_id, p.created              FROM                  " . $this->table\_name . " p                  LEFT JOIN                      categories c                          ON p.category\_id = c.id              WHERE                  p.id = ?              LIMIT                  0,1";        // prepare query statement      $stmt = $this->conn->prepare( $query );        // bind id of product to be updated      $stmt->bindParam(1, $this->id);        // execute query      $stmt->execute();        // get retrieved row      $row = $stmt->fetch(PDO::FETCH\_ASSOC);        // set values to object properties      $this->name = $row['name'];      $this->price = $row['price'];      $this->description = $row['description'];      $this->category\_id = $row['category\_id'];      $this->category\_name = $row['category\_name'];  } |

### 6.3 Salida

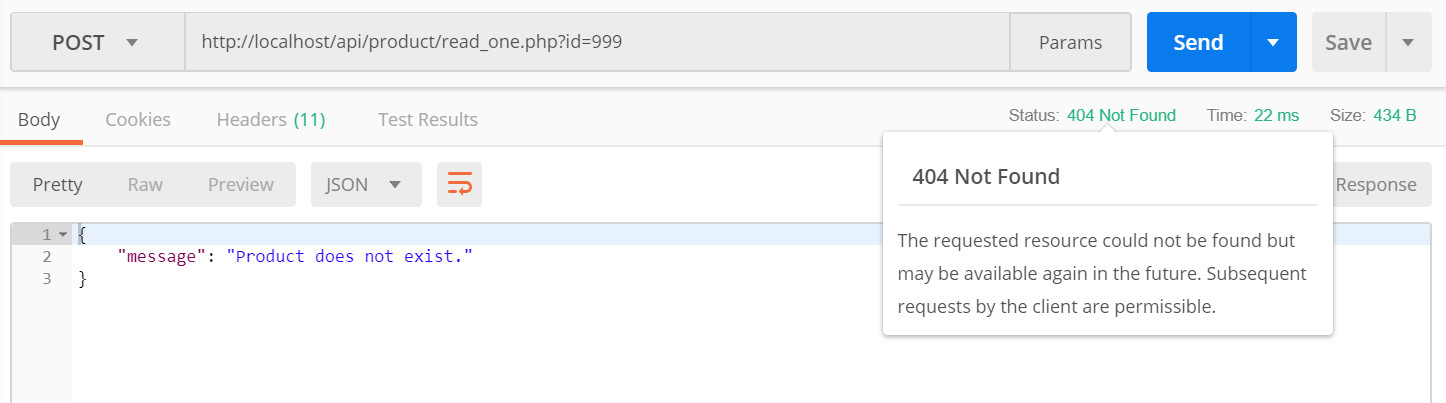
Primero, probaremos un producto que exista. Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud. Haga clic en el botón azul "Enviar".

[http: //localhost/api/product/read\_one.php? id = 60](http://localhost/api/product/read_one.php?id=60)

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-one-200.jpg?ssl=1)

A continuación, probaremos un producto que no existe. Ingrese lo siguiente como la URL de solicitud. Haga clic en el botón azul "Enviar".

[http: //localhost/api/product/read\_one.php? id = 999](http://localhost/api/product/read_one.php?id=999)

[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-one-404.jpg?ssl=1)

## 7.0 ACTUALIZAR PRODUCTO

### 7.1 Crear archivo "update.php"

Abra la carpeta del producto . Crear nuevo archivo update.php . Abra ese archivo y coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // include database and object files  include\_once '../config/database.php';  include\_once '../objects/product.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // prepare product object  $product = new Product($db);    // get id of product to be edited  $data = json\_decode(file\_get\_contents("php://input"));    // set ID property of product to be edited  $product->id = $data->id;    // set product property values  $product->name = $data->name;  $product->price = $data->price;  $product->description = $data->description;  $product->category\_id = $data->category\_id;    // update the product  if($product->update()){        // set response code - 200 ok      http\_response\_code(200);        // tell the user      echo json\_encode(array("message" => "Product was updated."));  }    // if unable to update the product, tell the user  else{        // set response code - 503 service unavailable      http\_response\_code(503);        // tell the user      echo json\_encode(array("message" => "Unable to update product."));  }  ?> |

### 7.2 Método de actualización del producto ()

Abrir carpeta de objetos . Abra el archivo product.php . La sección anterior no funcionará sin el siguiente código dentro de la clase de Producto.

|  |
| --- |
| // update the product  function update(){        // update query      $query = "UPDATE                  " . $this->table\_name . "              SET                  name = :name,                  price = :price,                  description = :description,                  category\_id = :category\_id              WHERE                  id = :id";        // prepare query statement      $stmt = $this->conn->prepare($query);        // sanitize      $this->name=htmlspecialchars(strip\_tags($this->name));      $this->price=htmlspecialchars(strip\_tags($this->price));      $this->description=htmlspecialchars(strip\_tags($this->description));      $this->category\_id=htmlspecialchars(strip\_tags($this->category\_id));      $this->id=htmlspecialchars(strip\_tags($this->id));        // bind new values      $stmt->bindParam(':name', $this->name);      $stmt->bindParam(':price', $this->price);      $stmt->bindParam(':description', $this->description);      $stmt->bindParam(':category\_id', $this->category\_id);      $stmt->bindParam(':id', $this->id);        // execute the query      if($stmt->execute()){          return true;      }        return false;  } |

### 7.3 Salida

Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/product/update.php](http://localhost/api/product/update.php)

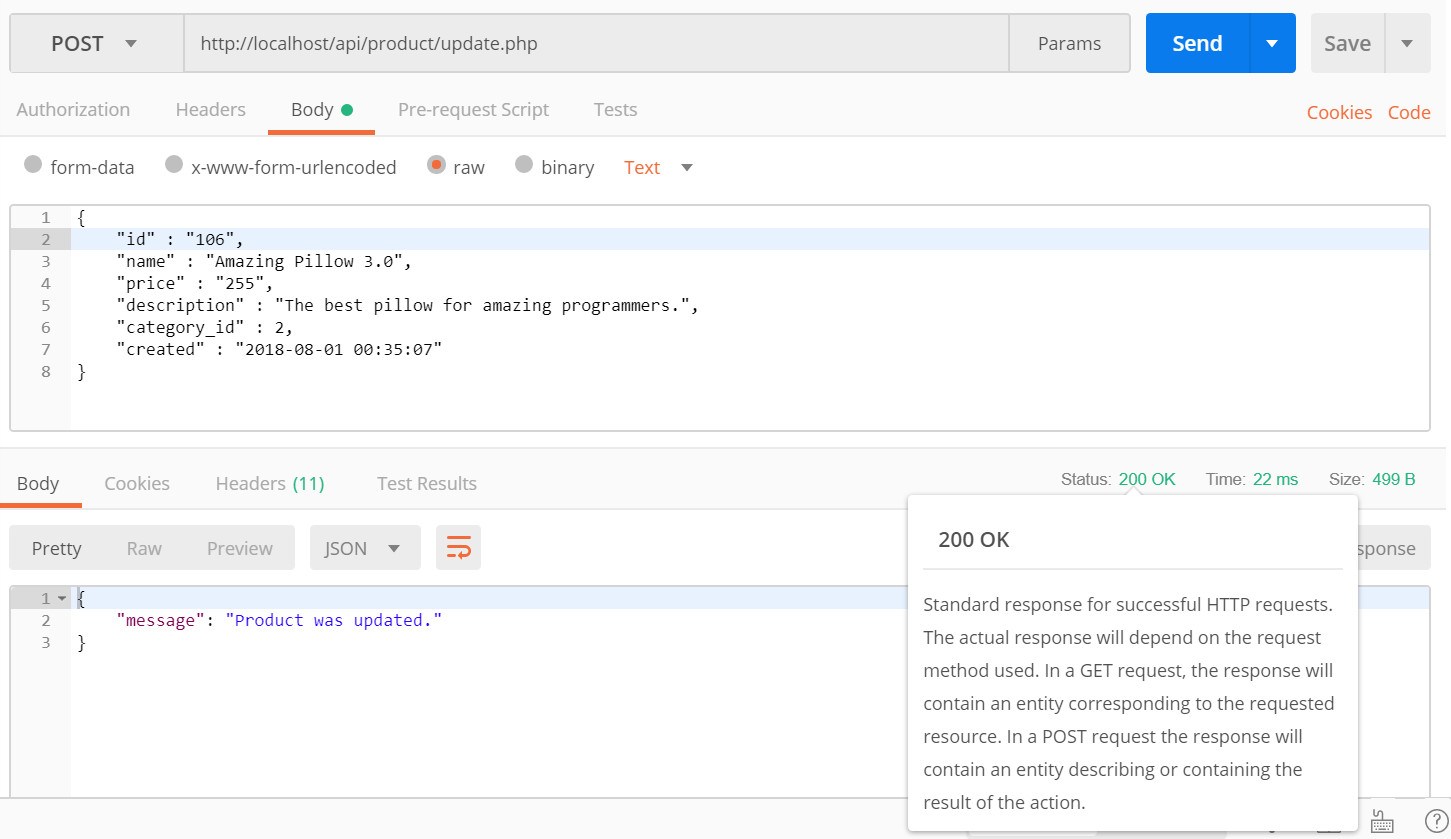
Haga clic en la pestaña "Cuerpo". Haga clic en "crudo". Ingrese el siguiente valor JSON (asegúrese de que el ID exista en su base de datos) y haga clic en el botón azul "Enviar".

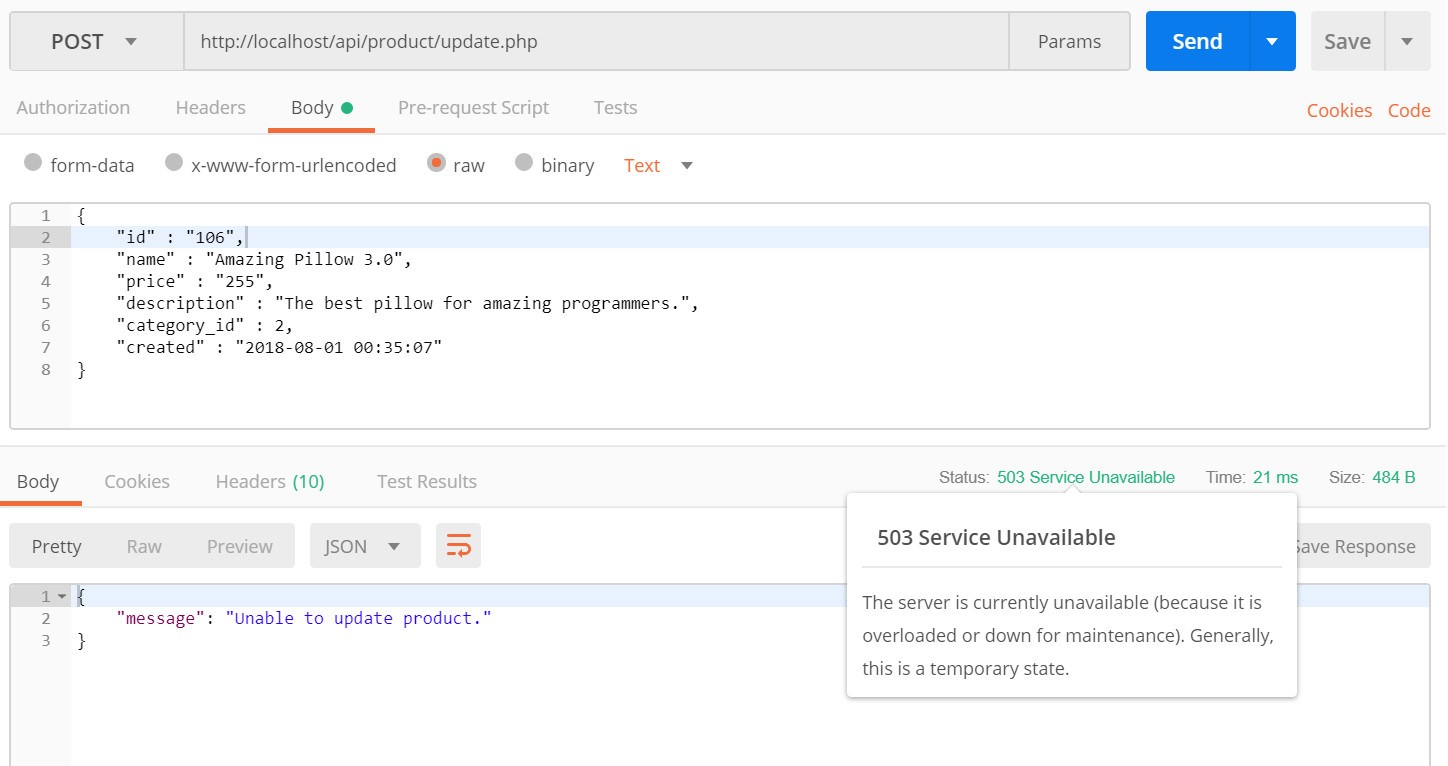
|  |
| --- |
| {      "id" : "106",      "name" : "Amazing Pillow 3.0",      "price" : "255",      "description" : "The best pillow for amazing programmers.",      "category\_id" : 2,      "created" : "2018-08-01 00:35:07"  } |

La identificación del producto 106 es solo un ejemplo. Debe especificar un ID de producto que exista en su base de datos.

Si especifica una ID que no existe en la base de datos, aún podría decir "el producto se actualizó. No actualiza nada en la base de datos, pero la consulta se ejecutó correctamente sin ningún error de sintaxis.

Para evitar esto, necesita una validación adicional donde verifique si existe una ID en la base de datos. Esta característica aún no forma parte de nuestro tutorial.

Si la actualización de un producto es exitosa, debería verse así:  
[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/update-product-200.jpg?ssl=1)

Si el sistema no puede actualizar el producto, la salida se verá así:  
[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/update-product-503.jpg?ssl=1)

## 8.0 ELIMINAR PRODUCTO

### 8.1 Crear archivo "delete.php"

Abra la carpeta del producto . Crear nuevo archivo delete.php . Abra ese archivo y coloque el siguiente código dentro de él.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // include database and object file  include\_once '../config/database.php';  include\_once '../objects/product.php';    // get database connection  $database = new Database();  $db = $database->getConnection();    // prepare product object  $product = new Product($db);    // get product id  $data = json\_decode(file\_get\_contents("php://input"));    // set product id to be deleted  $product->id = $data->id;    // delete the product  if($product->delete()){        // set response code - 200 ok      http\_response\_code(200);        // tell the user      echo json\_encode(array("message" => "Product was deleted."));  }    // if unable to delete the product  else{        // set response code - 503 service unavailable      http\_response\_code(503);        // tell the user      echo json\_encode(array("message" => "Unable to delete product."));  }  ?> |

### 8.2 Método de eliminación del producto ()

Abrir carpeta de objetos . Abra el archivo product.php . La sección anterior no funcionará sin el siguiente código dentro de la clase de Producto .

|  |
| --- |
| // delete the product  function delete(){        // delete query      $query = "DELETE FROM " . $this->table\_name . " WHERE id = ?";        // prepare query      $stmt = $this->conn->prepare($query);        // sanitize      $this->id=htmlspecialchars(strip\_tags($this->id));        // bind id of record to delete      $stmt->bindParam(1, $this->id);        // execute query      if($stmt->execute()){          return true;      }        return false;    } |

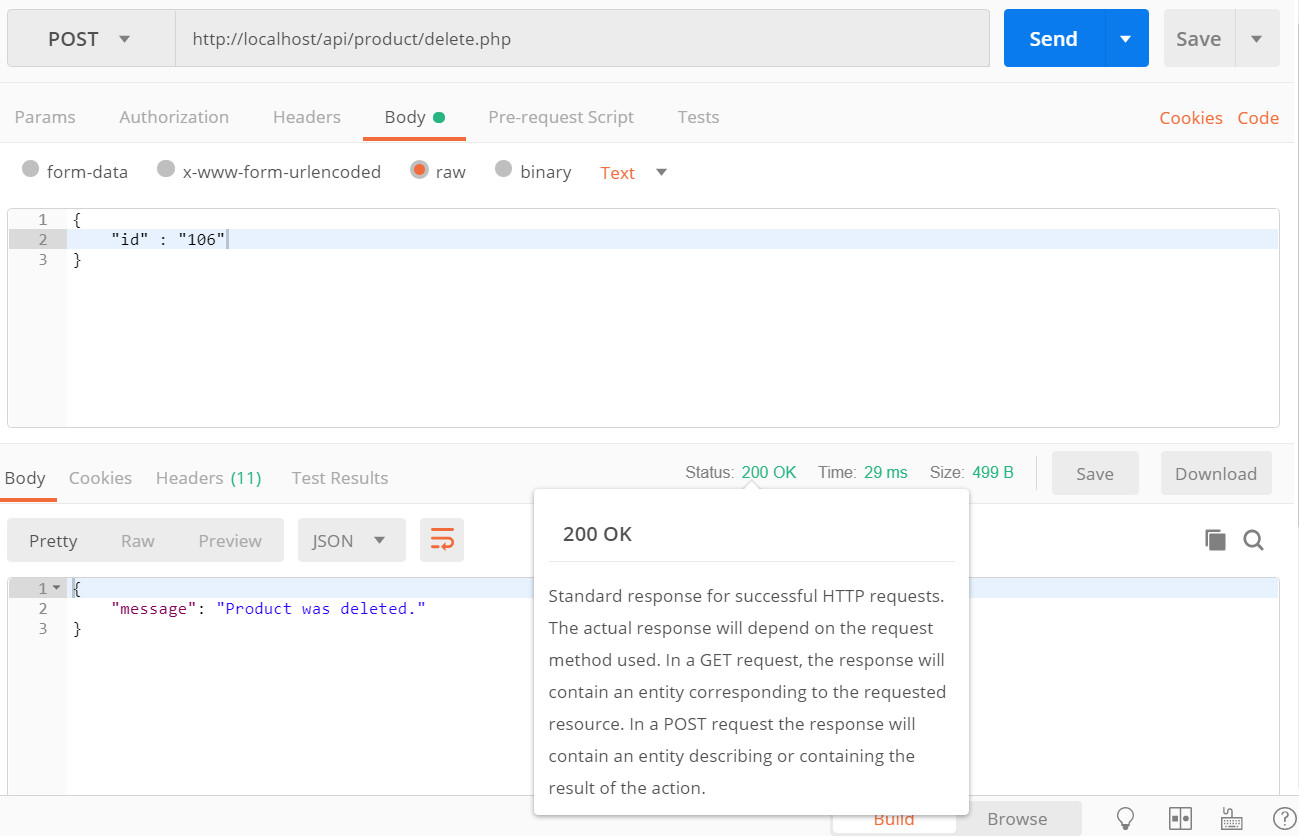
### 8.3 Salida

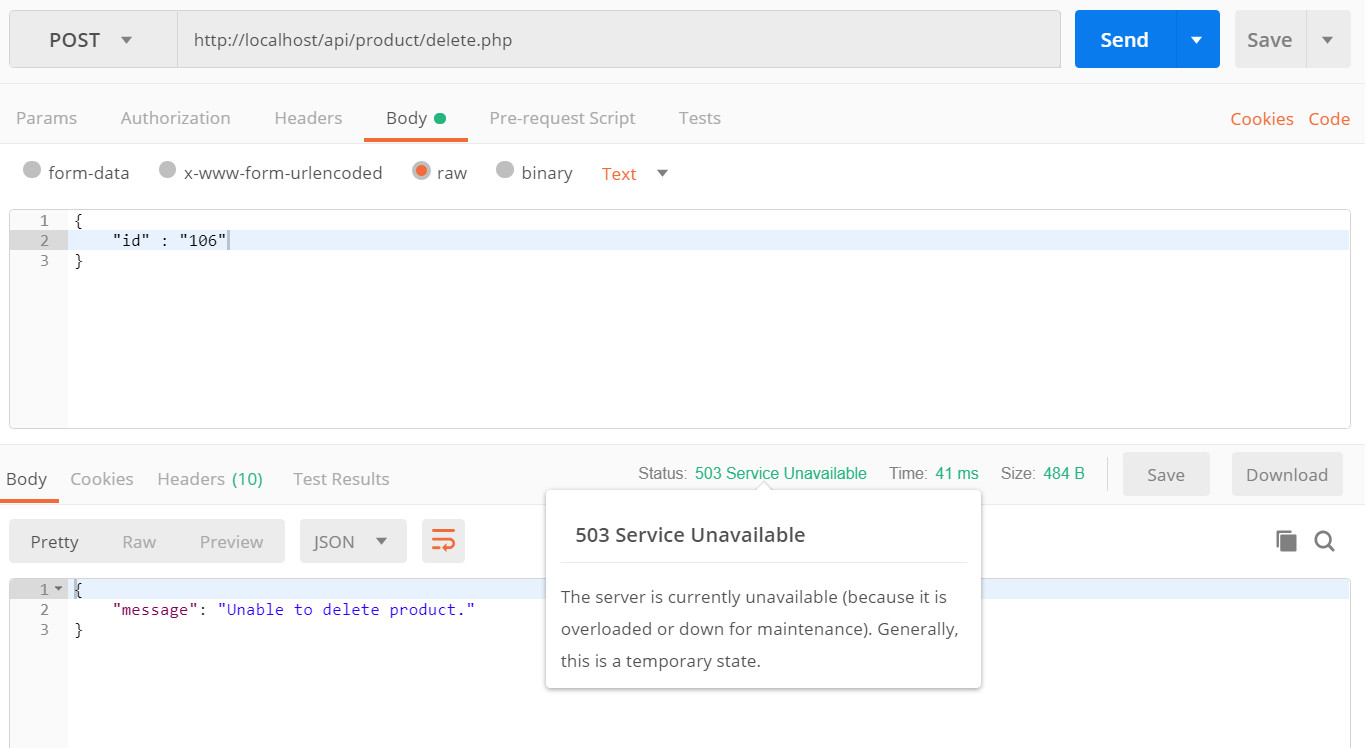
Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/product/delete.php](http://localhost/api/product/delete.php)

Haga clic en la pestaña "Cuerpo". Haga clic en "crudo". Ingrese el siguiente valor JSON (asegúrese de que el ID exista en su base de datos) y haga clic en el botón azul "Enviar".

|  |
| --- |
| {      "id" : "106"  } |

Si un producto se eliminó con éxito, debería verse así:  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/delete-200.jpg?ssl=1)

Si el sistema no puede eliminar el producto, la salida se verá así:  
[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/delete-503-1.jpg?ssl=1)

## 9.0 BUSCAR PRODUCTOS

### 9.1 Crear archivo "search.php"

Abra la carpeta del producto . Crea el archivo search.php . Abra ese archivo y coloque el siguiente código.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");    // include database and object files  include\_once '../config/core.php';  include\_once '../config/database.php';  include\_once '../objects/product.php';    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    // initialize object  $product = new Product($db);    // get keywords  $keywords=isset($\_GET["s"]) ? $\_GET["s"] : "";    // query products  $stmt = $product->search($keywords);  $num = $stmt->rowCount();    // check if more than 0 record found  if($num>0){        // products array      $products\_arr=array();      $products\_arr["records"]=array();        // retrieve our table contents      // fetch() is faster than fetchAll()      // <http://stackoverflow.com/questions/2770630/pdofetchall-vs-pdofetch-in-a-loop>      while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){          // extract row          // this will make $row['name'] to          // just $name only          extract($row);            $product\_item=array(              "id" => $id,              "name" => $name,              "description" => html\_entity\_decode($description),              "price" => $price,              "category\_id" => $category\_id,              "category\_name" => $category\_name          );            array\_push($products\_arr["records"], $product\_item);      }        // set response code - 200 OK      http\_response\_code(200);        // show products data      echo json\_encode($products\_arr);  }    else{      // set response code - 404 Not found      http\_response\_code(404);        // tell the user no products found      echo json\_encode(          array("message" => "No products found.")      );  }  ?> |

### 9.2 Crear método de búsqueda ()

Abrir carpeta de objetos . Abra el archivo product.php . Agregue el siguiente método de búsqueda ().

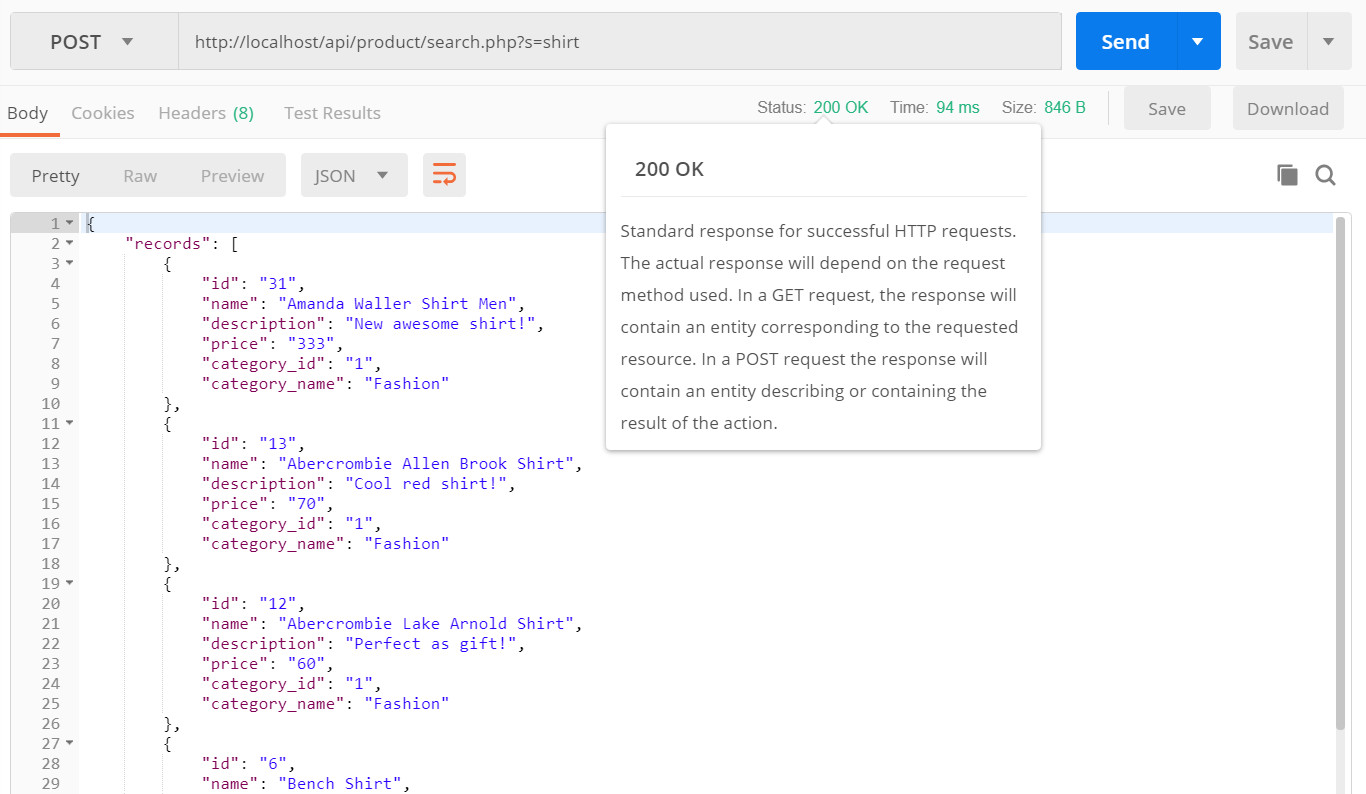
|  |
| --- |
| // search products  function search($keywords){        // select all query      $query = "SELECT                  c.name as category\_name, p.id, p.name, p.description, p.price, p.category\_id, p.created              FROM                  " . $this->table\_name . " p                  LEFT JOIN                      categories c                          ON p.category\_id = c.id              WHERE                  p.name LIKE ? OR p.description LIKE ? OR c.name LIKE ?              ORDER BY                  p.created DESC";        // prepare query statement      $stmt = $this->conn->prepare($query);        // sanitize      $keywords=htmlspecialchars(strip\_tags($keywords));      $keywords = "%{$keywords}%";        // bind      $stmt->bindParam(1, $keywords);      $stmt->bindParam(2, $keywords);      $stmt->bindParam(3, $keywords);        // execute query      $stmt->execute();        return $stmt;  } |

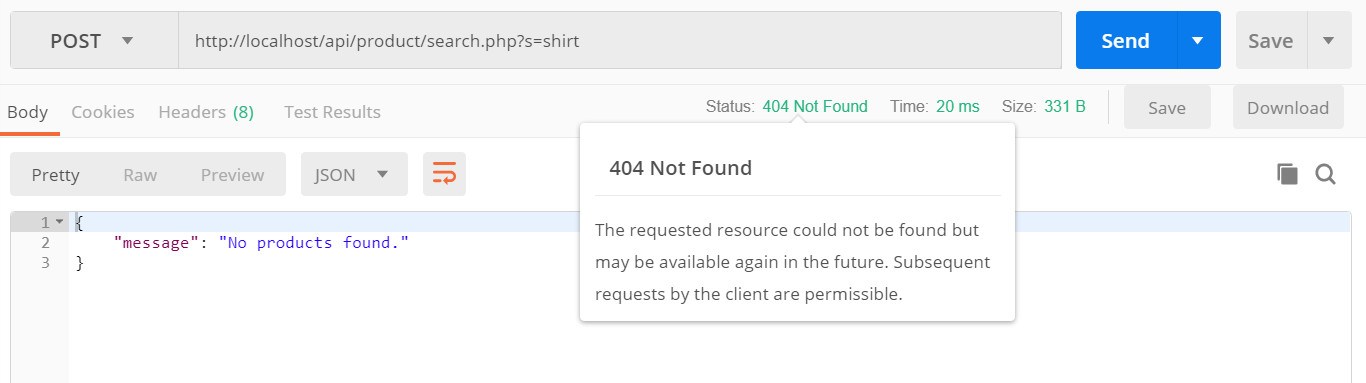
### 9.3 Salida

Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/product/search.php? s = shirt](http://localhost/api/product/search.php?s=shirt)

Haga clic en el botón azul "Enviar".

Si se encontró un producto, debería verse así:  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/search-200.jpg?ssl=1)

Si no se encuentran productos, la salida se verá así:  
[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/search-404.jpg?ssl=1)

## 10.0 PRODUCTOS PAGINATE

### 10.1 Crear el archivo "read\_paging.php"

Abra la carpeta del producto . Crea el archivo read\_paging.php .

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");    // include database and object files  include\_once '../config/core.php';  include\_once '../shared/utilities.php';  include\_once '../config/database.php';  include\_once '../objects/product.php';    // utilities  $utilities = new Utilities();    // instantiate database and product object  $database = new Database();  $db = $database->getConnection();    // initialize object  $product = new Product($db);    // query products  $stmt = $product->readPaging($from\_record\_num, $records\_per\_page);  $num = $stmt->rowCount();    // check if more than 0 record found  if($num>0){        // products array      $products\_arr=array();      $products\_arr["records"]=array();      $products\_arr["paging"]=array();        // retrieve our table contents      // fetch() is faster than fetchAll()      // <http://stackoverflow.com/questions/2770630/pdofetchall-vs-pdofetch-in-a-loop>      while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){          // extract row          // this will make $row['name'] to          // just $name only          extract($row);            $product\_item=array(              "id" => $id,              "name" => $name,              "description" => html\_entity\_decode($description),              "price" => $price,              "category\_id" => $category\_id,              "category\_name" => $category\_name          );            array\_push($products\_arr["records"], $product\_item);      }          // include paging      $total\_rows=$product->count();      $page\_url="{$home\_url}product/read\_paging.php?";      $paging=$utilities->getPaging($page, $total\_rows, $records\_per\_page, $page\_url);      $products\_arr["paging"]=$paging;        // set response code - 200 OK      http\_response\_code(200);        // make it json format      echo json\_encode($products\_arr);  }    else{        // set response code - 404 Not found      http\_response\_code(404);        // tell the user products does not exist      echo json\_encode(          array("message" => "No products found.")      );  }  ?> |

### 10.2 Crear archivo "core.php"

Este archivo contiene nuestra configuración principal, como la URL de inicio y las variables de paginación.

Abra la carpeta de configuración y cree el archivo core.php . Abra el archivo core.php y coloque el siguiente código.

|  |
| --- |
| <?php  // show error reporting  ini\_set('display\_errors', 1);  error\_reporting(E\_ALL);    // home page url  $home\_url="<http://localhost/api/>";    // page given in URL parameter, default page is one  $page = isset($\_GET['page']) ? $\_GET['page'] : 1;    // set number of records per page  $records\_per\_page = 5;    // calculate for the query LIMIT clause  $from\_record\_num = ($records\_per\_page \* $page) - $records\_per\_page;  ?> |

### 10.3 Crear el método "readPaging ()"

Abrir carpeta de objetos . Abra el archivo product.php . Agregue el siguiente método dentro de la clase de producto. Este método devolverá una lista de registros limitada a lo que establecimos en $ records\_per\_page de la sección anterior.

|  |
| --- |
| // read products with pagination  public function readPaging($from\_record\_num, $records\_per\_page){        // select query      $query = "SELECT                  c.name as category\_name, p.id, p.name, p.description, p.price, p.category\_id, p.created              FROM                  " . $this->table\_name . " p                  LEFT JOIN                      categories c                          ON p.category\_id = c.id              ORDER BY p.created DESC              LIMIT ?, ?";        // prepare query statement      $stmt = $this->conn->prepare( $query );        // bind variable values      $stmt->bindParam(1, $from\_record\_num, PDO::PARAM\_INT);      $stmt->bindParam(2, $records\_per\_page, PDO::PARAM\_INT);        // execute query      $stmt->execute();        // return values from database      return $stmt;  } |

### 10.4 Crear el método "count ()"

Aún en la clase de producto ( archivo product.php ), agregue el siguiente método. Las filas totales son necesarias para construir la matriz de paginación. Se incluye en el cálculo de 'paginación'.

|  |
| --- |
| // used for paging products  public function count(){      $query = "SELECT COUNT(\*) as total\_rows FROM " . $this->table\_name . "";        $stmt = $this->conn->prepare( $query );      $stmt->execute();      $row = $stmt->fetch(PDO::FETCH\_ASSOC);        return $row['total\_rows'];  } |

### 10.5 Obtener matriz de "paginación"

Crear carpeta compartida Abrir carpeta compartida .

Crear archivo utilities.php . Abra el archivo utilities.php y coloque el siguiente código.

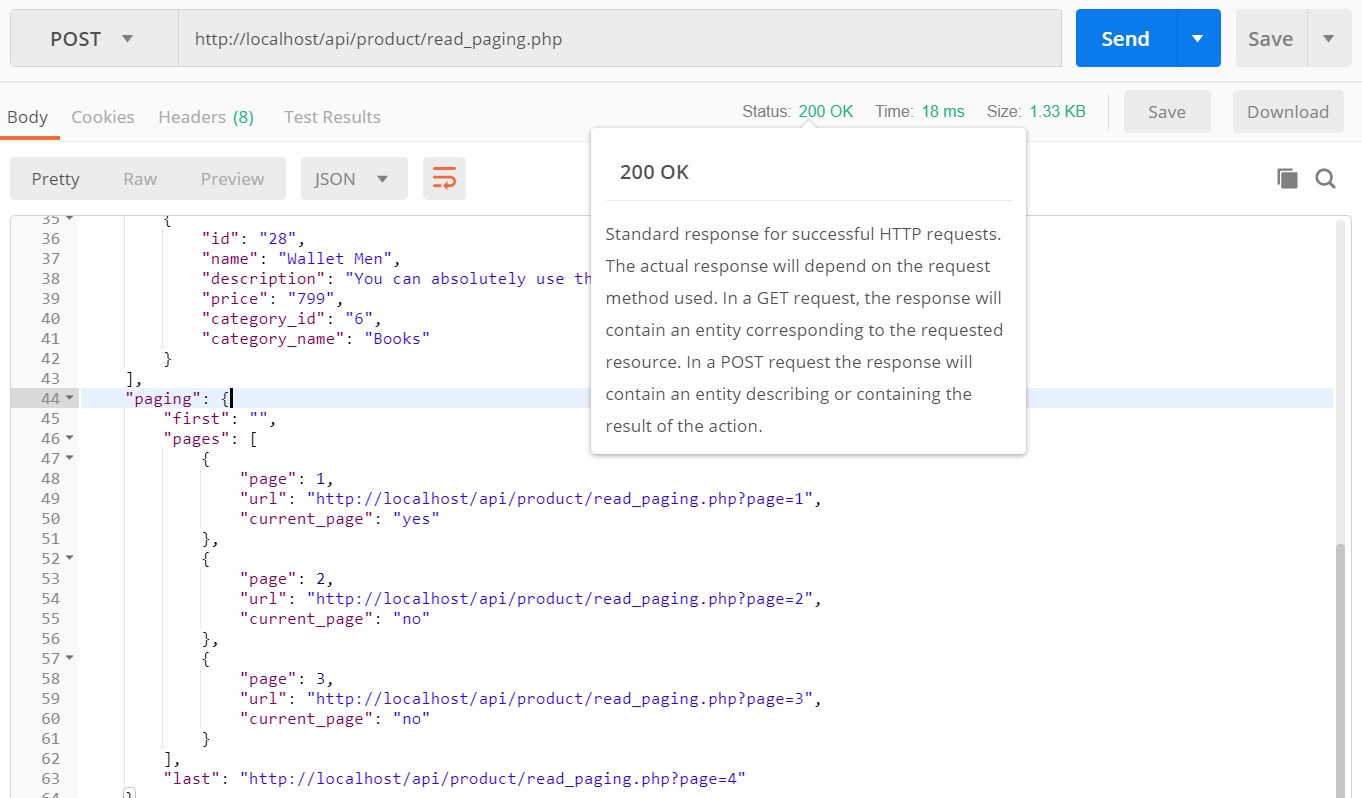
|  |
| --- |
| <?php  class Utilities{        public function getPaging($page, $total\_rows, $records\_per\_page, $page\_url){            // paging array          $paging\_arr=array();            // button for first page          $paging\_arr["first"] = $page>1 ? "{$page\_url}page=1" : "";            // count all products in the database to calculate total pages          $total\_pages = ceil($total\_rows / $records\_per\_page);            // range of links to show          $range = 2;            // display links to 'range of pages' around 'current page'          $initial\_num = $page - $range;          $condition\_limit\_num = ($page + $range)  + 1;            $paging\_arr['pages']=array();          $page\_count=0;            for($x=$initial\_num; $x<$condition\_limit\_num; $x++){              // be sure '$x is greater than 0' AND 'less than or equal to the $total\_pages'              if(($x > 0) && ($x <= $total\_pages)){                  $paging\_arr['pages'][$page\_count]["page"]=$x;                  $paging\_arr['pages'][$page\_count]["url"]="{$page\_url}page={$x}";                  $paging\_arr['pages'][$page\_count]["current\_page"] = $x==$page ? "yes" : "no";                    $page\_count++;              }          }            // button for last page          $paging\_arr["last"] = $page<$total\_pages ? "{$page\_url}page={$total\_pages}" : "";            // json format          return $paging\_arr;      }    }  ?> |

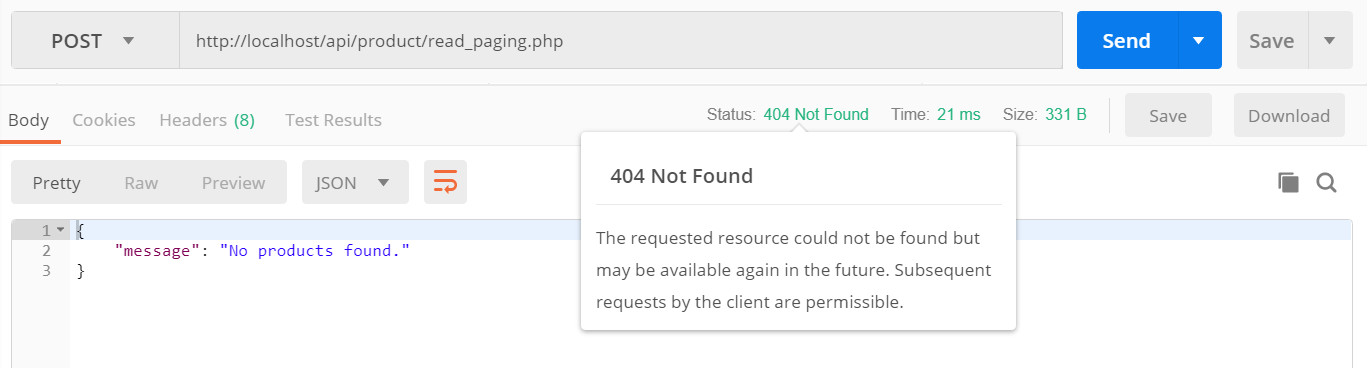
### 10.6 Salida

Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/product/read\_paging.php](http://localhost/api/product/read_paging.php)

Haga clic en el botón azul "Enviar".

Si se encuentran productos, desplácese hacia abajo para ver el nodo de paginación . Debe tener un aspecto como este:  
[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-paging-200.jpg?ssl=1)

Si no se encuentran productos, la salida se verá así:  
[](https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-paging-404.jpg?ssl=1)

## 11.0 LEER CATEGORÍAS

### 11.1 Crear archivo "category.php"

Abrir carpeta de objetos . Crear nuevo archivo category.php . Coloque el siguiente código dentro del archivo category.php .

|  |
| --- |
| <?php  class Category{        // database connection and table name      private $conn;      private $table\_name = "categories";        // object properties      public $id;      public $name;      public $description;      public $created;        public function \_\_construct($db){          $this->conn = $db;      }        // used by select drop-down list      public function readAll(){          //select all data          $query = "SELECT                      id, name, description                  FROM                      " . $this->table\_name . "                  ORDER BY                      name";            $stmt = $this->conn->prepare( $query );          $stmt->execute();            return $stmt;      }  }  ?> |

### 11.2 Crear archivo "read.php"

Crear nueva carpeta de categoría . Abra esa carpeta y cree un nuevo archivo read.php dentro de ella. Abra el archivo read.php y coloque el siguiente código.

|  |
| --- |
| <?php  // required header  header("Access-Control-Allow-Origin: \*");  header("Content-Type: application/json; charset=UTF-8");    // include database and object files  include\_once '../config/database.php';  include\_once '../objects/category.php';    // instantiate database and category object  $database = new Database();  $db = $database->getConnection();    // initialize object  $category = new Category($db);    // query categorys  $stmt = $category->read();  $num = $stmt->rowCount();    // check if more than 0 record found  if($num>0){        // products array      $categories\_arr=array();      $categories\_arr["records"]=array();        // retrieve our table contents      // fetch() is faster than fetchAll()      // <http://stackoverflow.com/questions/2770630/pdofetchall-vs-pdofetch-in-a-loop>      while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)){          // extract row          // this will make $row['name'] to          // just $name only          extract($row);            $category\_item=array(              "id" => $id,              "name" => $name,              "description" => html\_entity\_decode($description)          );            array\_push($categories\_arr["records"], $category\_item);      }        // set response code - 200 OK      http\_response\_code(200);        // show categories data in json format      echo json\_encode($categories\_arr);  }    else{        // set response code - 404 Not found      http\_response\_code(404);        // tell the user no categories found      echo json\_encode(          array("message" => "No categories found.")      );  }  ?> |

### 11.3 Agregar método de categoría "read ()"

Abrir carpeta de objetos . Abra el archivo category.php . El código de la sección anterior no funcionará sin el siguiente código dentro del archivo category.php . Agregue el siguiente método dentro de la clase Categoría .

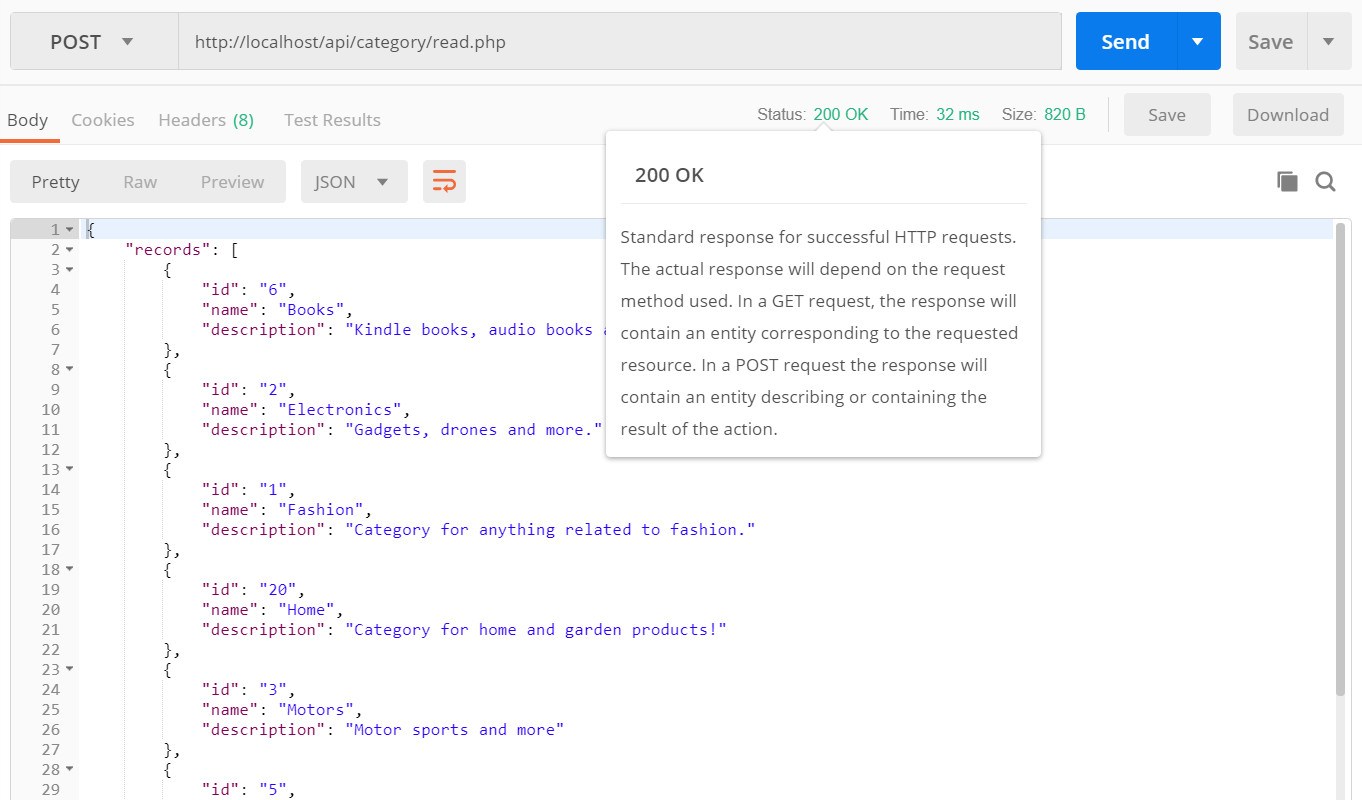
|  |
| --- |
| // used by select drop-down list  public function read(){        //select all data      $query = "SELECT                  id, name, description              FROM                  " . $this->table\_name . "              ORDER BY                  name";        $stmt = $this->conn->prepare( $query );      $stmt->execute();        return $stmt;  } |

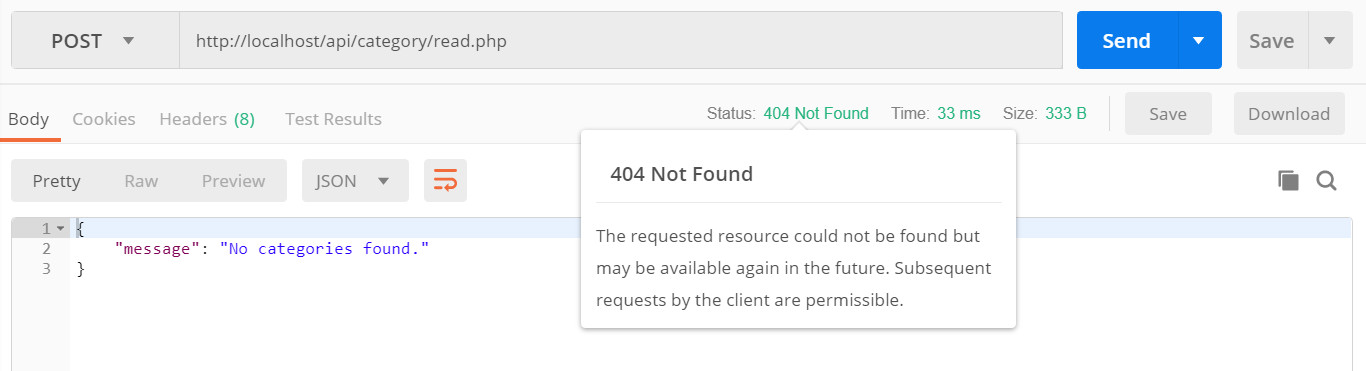
### 11.4 Salida

Abrir POSTMAN. Ingrese lo siguiente como la URL de solicitud.

[http: //localhost/api/category/read.php](http://localhost/api/category/read.php)

Haga clic en el botón azul "Enviar".

Si se encuentran categorías, debería verse así:  
[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-categories-200.jpg?ssl=1)

Si no se encuentran categorías, la salida se verá así:  
[](https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-categories-404.jpg?ssl=1)

## 12.0 DESCARGAR CÓDIGOS FUENTE

Le recomendamos encarecidamente que siga y estudie nuestro tutorial detallado y detallado paso a paso primero. Nada supera la experiencia cuando se trata de aprender.

Pero creemos que aprenderá más rápido si también ve el código fuente final. Lo consideramos como su guía adicional.

Imagine el valor o la mejora de habilidades que puede brindarle. El ingreso adicional que puede obtener de su trabajo, proyectos o negocio. El precioso tiempo que ahorras. ¿No es eso lo que quieres?

### 12.1 Descargar el código fuente LEVEL 1

|  |  |
| --- | --- |
| CARACTERISTICAS | NIVEL 1 |
| Crear producto | SI |
| Leer productos | SI |
| Leer un producto | SI |
| Actualizar producto | SI |
| Eliminar producto | SI |
| Buscar Productos | SI |
| Productos paginados | SI |
| Leer categorías | SI |
| Soporte por correo electrónico GRATIS por 3 meses | SI |
| Actualizaciones del código fuente por correo electrónico | SI |
| Principio del formulario   * NIVEL 1 Código fuente$ 20.00 * NIVEL 2 Código fuente$ 40.00   [DESCARGAR AHORA](https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html)  Final del formulario | |

### 12.2 Descargar el código fuente LEVEL 2

|  |  |
| --- | --- |
| CARACTERISTICAS | NIVEL 2 |
| Todas las características del código fuente LEVEL 1 | SI |
| Eliminar producto seleccionado | SI |
| Exportar producto CSV | SI |
| Leer productos por categoría | SI |
| Buscar productos con paginación | SI |
| Crear categoría | SI |
| Leer categorías | SI |
| Leer una categoría | SI |
| Actualizar categoría | SI |
| Eliminar categoría | SI |
| Buscar categorías | SI |
| Paginar categorías | SI |
| Eliminar categorías seleccionadas | SI |
| Categorías de exportación CSV | SI |
| Buscar categorías con paginación | SI |
| Soporte por correo electrónico GRATIS por 6 meses | SI |
| Actualizaciones del código fuente por correo electrónico | SI |
| Principio del formulario   * NIVEL 1 Código fuente$ 20.00 * NIVEL 2 Código fuente$ 40.00   [DESCARGAR AHORA](https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html)  Final del formulario | |

## LINKS PARA VER

<https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html>

<https://www.codeofaninja.com/2018/09/rest-api-authentication-example-php-jwt-tutorial.html>