How To Create A Simple REST API in PHP? Step By Step Guide!



Previously, we learned how to create, read, update and delete database records (CRUD operations) with our [PHP, MySQL & OOP CRUD Tutorial](https://www.codeofaninja.com/2014/06/php-object-oriented-crud-example-oop.html).

Today, before we go to JavaScript programming, we will learn how to create a simple REST API in PHP. Enjoy our step-by-step tutorial below!

**Contents**[[hide](https://codeofaninja.com/create-simple-rest-api-in-php/)]

* [1 Project Overview](https://codeofaninja.com/create-simple-rest-api-in-php/#Project_Overview)
  + [1.1 What is REST API?](https://codeofaninja.com/create-simple-rest-api-in-php/#What_is_REST_API)
  + [1.2 Why do we need REST API?](https://codeofaninja.com/create-simple-rest-api-in-php/#Why_do_we_need_REST_API)
  + [1.3 Where REST API is used?](https://codeofaninja.com/create-simple-rest-api-in-php/#Where_REST_API_is_used)
  + [1.4 REST API in our tutorials](https://codeofaninja.com/create-simple-rest-api-in-php/#REST_API_in_our_tutorials)
* [2 File structure](https://codeofaninja.com/create-simple-rest-api-in-php/#File_structure)
* [3 Setup the database](https://codeofaninja.com/create-simple-rest-api-in-php/#Setup_the_database)
  + [3.1 Create categories table](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_categories_table)
  + [3.2 Dump data for categories table](https://codeofaninja.com/create-simple-rest-api-in-php/#Dump_data_for_categories_table)
  + [3.3 Products table](https://codeofaninja.com/create-simple-rest-api-in-php/#Products_table)
  + [3.4 Dump data for products table](https://codeofaninja.com/create-simple-rest-api-in-php/#Dump_data_for_products_table)
  + [3.5 Connect to database](https://codeofaninja.com/create-simple-rest-api-in-php/#Connect_to_database)
* [4 Read products](https://codeofaninja.com/create-simple-rest-api-in-php/#Read_products)
  + [4.1 Product object](https://codeofaninja.com/create-simple-rest-api-in-php/#Product_object)
  + [4.2 Create file to read products](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_file_to_read_products)
  + [4.3 Connect to database and products table](https://codeofaninja.com/create-simple-rest-api-in-php/#Connect_to_database_and_products_table)
  + [4.4 Read products from the database](https://codeofaninja.com/create-simple-rest-api-in-php/#Read_products_from_the_database)
  + [4.5 Add product “read()” method](https://codeofaninja.com/create-simple-rest-api-in-php/#Add_product_8220read8221_method)
  + [4.6 Tell the user no products found](https://codeofaninja.com/create-simple-rest-api-in-php/#Tell_the_user_no_products_found)
  + [4.7 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output)
* [5 Create Product](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_Product)
  + [5.1 Create create.php file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_createphp_file)
  + [5.2 Product create() method](https://codeofaninja.com/create-simple-rest-api-in-php/#Product_create_method)
  + [5.3 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-2)
* [6 Read One Product](https://codeofaninja.com/create-simple-rest-api-in-php/#Read_One_Product)
  + [6.1 Create read\_one.php file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_read_onephp_file)
  + [6.2 Product readOne() method](https://codeofaninja.com/create-simple-rest-api-in-php/#Product_readOne_method)
  + [6.3 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-3)
* [7 Update product](https://codeofaninja.com/create-simple-rest-api-in-php/#Update_product)
  + [7.1 Create “update.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220updatephp8221_file)
  + [7.2 Product update() method](https://codeofaninja.com/create-simple-rest-api-in-php/#Product_update_method)
  + [7.3 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-4)
* [8 Delete Product](https://codeofaninja.com/create-simple-rest-api-in-php/#Delete_Product)
  + [8.1 Create “delete.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220deletephp8221_file)
  + [8.2 Product delete() method](https://codeofaninja.com/create-simple-rest-api-in-php/#Product_delete_method)
  + [8.3 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-5)
* [9 Search Products](https://codeofaninja.com/create-simple-rest-api-in-php/#Search_Products)
  + [9.1 Create “search.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220searchphp8221_file)
  + [9.2 Create search() method](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_search_method)
  + [9.3 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-6)
* [10 Paginate Products](https://codeofaninja.com/create-simple-rest-api-in-php/#Paginate_Products)
  + [10.1 Create “read\_paging.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220read_pagingphp8221_file)
  + [10.2 Create “core.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220corephp8221_file)
  + [10.3 Create “readPaging()” method](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220readPaging8221_method)
  + [10.4 Create “count()” method](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220count8221_method)
  + [10.5 Get “paging” array](https://codeofaninja.com/create-simple-rest-api-in-php/#Get_8220paging8221_array)
  + [10.6 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-7)
* [11 Read Categories](https://codeofaninja.com/create-simple-rest-api-in-php/#Read_Categories)
  + [11.1 Create “category.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220categoryphp8221_file)
  + [11.2 Create “read.php” file](https://codeofaninja.com/create-simple-rest-api-in-php/#Create_8220readphp8221_file)
  + [11.3 Add Category “read()” method](https://codeofaninja.com/create-simple-rest-api-in-php/#Add_Category_8220read8221_method)
  + [11.4 Output](https://codeofaninja.com/create-simple-rest-api-in-php/#Output-8)
* [12 Download Source Codes](https://codeofaninja.com/create-simple-rest-api-in-php/#Download_Source_Codes)
  + [12.1 Download LEVEL 1 source code](https://codeofaninja.com/create-simple-rest-api-in-php/#Download_LEVEL_1_source_code)
  + [12.2 Download LEVEL 2 source code](https://codeofaninja.com/create-simple-rest-api-in-php/#Download_LEVEL_2_source_code)
* [13 What’s Next?](https://codeofaninja.com/create-simple-rest-api-in-php/#What8217s_Next)
* [14 Related Tutorials](https://codeofaninja.com/create-simple-rest-api-in-php/#Related_Tutorials)
* [15 Notes](https://codeofaninja.com/create-simple-rest-api-in-php/#Notes)
  + [15.1 Found An Issue?](https://codeofaninja.com/create-simple-rest-api-in-php/#Found_An_Issue)
  + [15.2 Subscribe to CodeOfaNinja](https://codeofaninja.com/create-simple-rest-api-in-php/#Subscribe_to_CodeOfaNinja)
  + [15.3 Thank You!](https://codeofaninja.com/create-simple-rest-api-in-php/#Thank_You)
  + [15.4 Share our tutorial](https://codeofaninja.com/create-simple-rest-api-in-php/#Share_our_tutorial)

**Project Overview**

**What is REST API?**

To define “REST API”, we have to know what is “REST” and what is “API” first. I’ll do my best to explain it in simple terms because REST has a lot of concepts inside of it that [could mean a lot of things](https://stackoverflow.com/questions/4663927/what-is-rest-slightly-confused).

**REST** stands for “REpresentational State Transfer”. It is a concept or architecture for managing information over the internet. REST concepts are referred to as resources. A representation of a resource must be stateless. It is usually represented by [JSON](https://stackoverflow.com/questions/383692/what-is-json-and-why-would-i-use-it). This post is worth reading: [How I Explained REST to My Wife?](http://www.looah.com/source/view/2284)

**API** stands for “Application Programming Interface”. It is a set of rules that allows one piece of software application to talk to another. Those “rules” can include the create, read, update and delete operations.

**REST API** enables your application to cooperate with one or several different applications using REST concepts. If you want to learn more, watch the video below.

**Why do we need REST API?**

In many applications, REST API is a need because this is the lightest way to create, read, update or delete information between different applications over the internet or HTTP protocol. This information is presented to the user in an instant especially if you use JavaScript to render the data on a webpage.

**Where REST API is used?**

REST API can be used by any application that can connect to the internet. If data from an application can be created, read, updated or deleted using another application, it usually means a REST API is used.

**REST API in our tutorials**

A REST API is needed for our [AJAX CRUD Tutorial](https://www.codeofaninja.com/2015/06/php-crud-with-ajax-and-oop.html). But don’t mind it for now. We will do it one step at a time. You don’t need to learn all of it as well. Just choose what you need to learn.

But one thing is for sure, this source code is good enough and works for our JavaScript tutorials.

**File structure**

At the end of this tutorial, we will have the following folders and files.  
├─ api/  
├─── config/  
├────── core.php – file used for core configuration  
├────── database.php – file used for connecting to the database.  
├─── objects/  
├────── product.php – contains properties and methods for “product” database queries.  
├────── category.php – contains properties and methods for “category” database queries.  
├─── product/  
├────── create.php – a file that will accept posted product data to be saved to the database.  
├────── delete.php – a file that will accept a product ID to delete a database record.  
├────── read.php – a file that will output JSON data based on “products” database records.  
├────── read\_paging.php – a file that will output “products” JSON data with pagination.  
├────── read\_one.php – a file that will accept product ID to read a record from the database.  
├────── update.php – a file that will accept a product ID to update a database record.  
├────── search.php – a file that will accept keywords parameter to search “products” database.  
├─── category/  
├────── read.php – a file that will output JSON data based on “categories” database records.  
├─── shared/  
├────── utilities.php – a file that will return pagination array.

**Setup the database**

Using PhpMyAdmin, create a new api\_db database. Yes, api\_db is the database name. After that, run the following SQL queries to create new tables with sample data.

**Create categories table**

|  |
| --- |
| **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 ; |

**Dump data for categories table**

|  |
| --- |
| **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'); |

**Products table**

|  |
| --- |
| **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 ; |

**Dump data for products table**

|  |
| --- |
| **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'); |

**Connect to database**

The code below shows the database credentials and a method to get a database connection using PDO. If you’re not yet familiar with PDO, please learn from our [PHP OOP CRUD Tutorial](https://www.codeofaninja.com/2014/06/php-object-oriented-crud-example-oop.html) first.

* Create api folder. Open api folder.
* Create config folder. Open config folder.
* Create a database.php file. Place the following code inside it.

|  |
| --- |
| <?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;      }  }  ?> |

**Read products**

**Product object**

The code below shows a class named Product with several of its properties. It also shows a constructor method that will accept the database connection.

We will use this class to read data from the database.

* Open api folder.
* Create objects folder.
* Open objects folder.
* Create product.php file.
* Place the following code inside it.

|  |
| --- |
| <?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;      }  }  ?> |

**Create file to read products**

The code below shows headers about who can read this file and which type of content it will return.

In this case, our read.php the file can be read by anyone (asterisk \* means all) and will return a data in [JSON format](https://www.json.org/).

* Open api folder.
* Create product folder.
* Open product folder.
* Create read.php file.
* Place the following code inside it.

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

**Connect to database and products table**

In the code below, we included the database.php and product.php files. These are the files we created earlier.

We need to use the getConnection() method of the Database class to get a database connection. We pass this connection to the Product class.

Replace of // database connection will be here comment of read.php file with the following code.

|  |
| --- |
| // 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 |

**Read products from the database**

In the code below, we used the read() method of Product class to read data from the database. Through the $num variable, we check if there are records found.

If there are records found, we loop through it using the while loop, add each record to the $products\_arr array, set a 200 OK response code and show it to the user in JSON format.

Replace of // read products will be here comment of read.php file with the following code.

|  |
| --- |
| // 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 |

**Add product “read()” method**

We used the read() method in the previous section but it does not exist yet in the Product class. We need to add this read() method. The code below shows the query to get records from the database.

* Open objects folder.
* Open product.php file.
* Place the following code inside the Product class.
* To make sure you added it correctly, place the code before the last closing curly brace.

|  |
| --- |
| // 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;  } |

**Tell the user no products found**

If the $num variable has a value of zero or negative, it means there are no records returned from the database. We need to tell the user about this.

On the code below, we set the response code to 404 - Not found and a message that says No products found.

Replace of // no products found will be here comment of read.php file with the following code.

|  |
| --- |
| **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.")      );  } |

**Output**

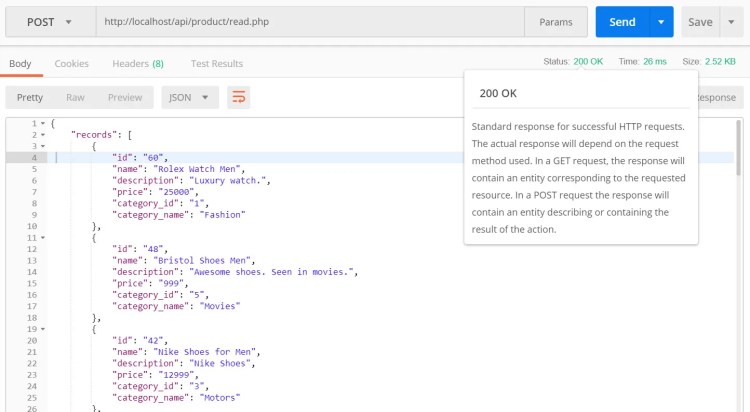
You need to use [POSTMAN](https://www.getpostman.com/) to test our API. Download your version of POSTMAN [here](https://www.getpostman.com/apps).

Launch POSTMAN. Enter the following as the request URL.

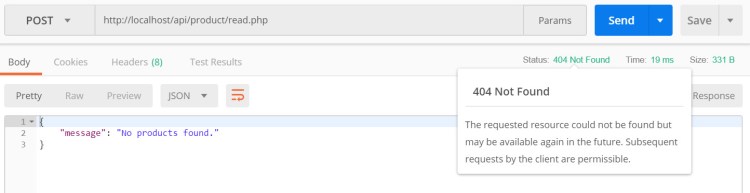
|  |
| --- |
| <http://localhost/api/product/read.php> |

Click the blue “Send” button.

* Output if there are product data.

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

* Output if there are no product data.

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-products-404.jpg?ssl=1)

**Create Product**

**Create create.php file**

* Open product folder.
* Create a new create.php file.
* Open that file and put the following code inside it.

|  |
| --- |
| <?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."));  }  ?> |

**Product create() method**

* Open objects folder.
* Open product.php file.
* The previous section will not work without the following code inside the Product (objects/product.php) class.

|  |
| --- |
| // 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;    } |

**Output**

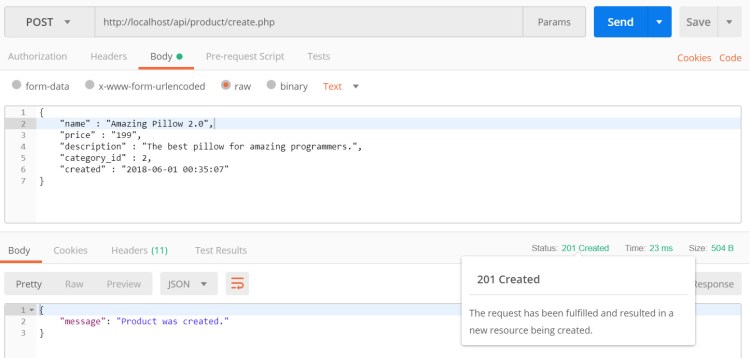
To test for the successful creation of a product, open POSTMAN. Enter the following as the request URL

|  |
| --- |
| <http://localhost/api/product/create.php> |

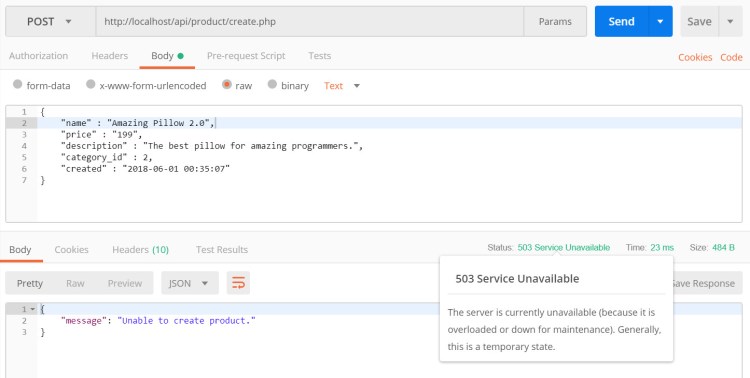
* Click the “Body” tab.
* Click “raw”.
* Enter this JSON value:

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

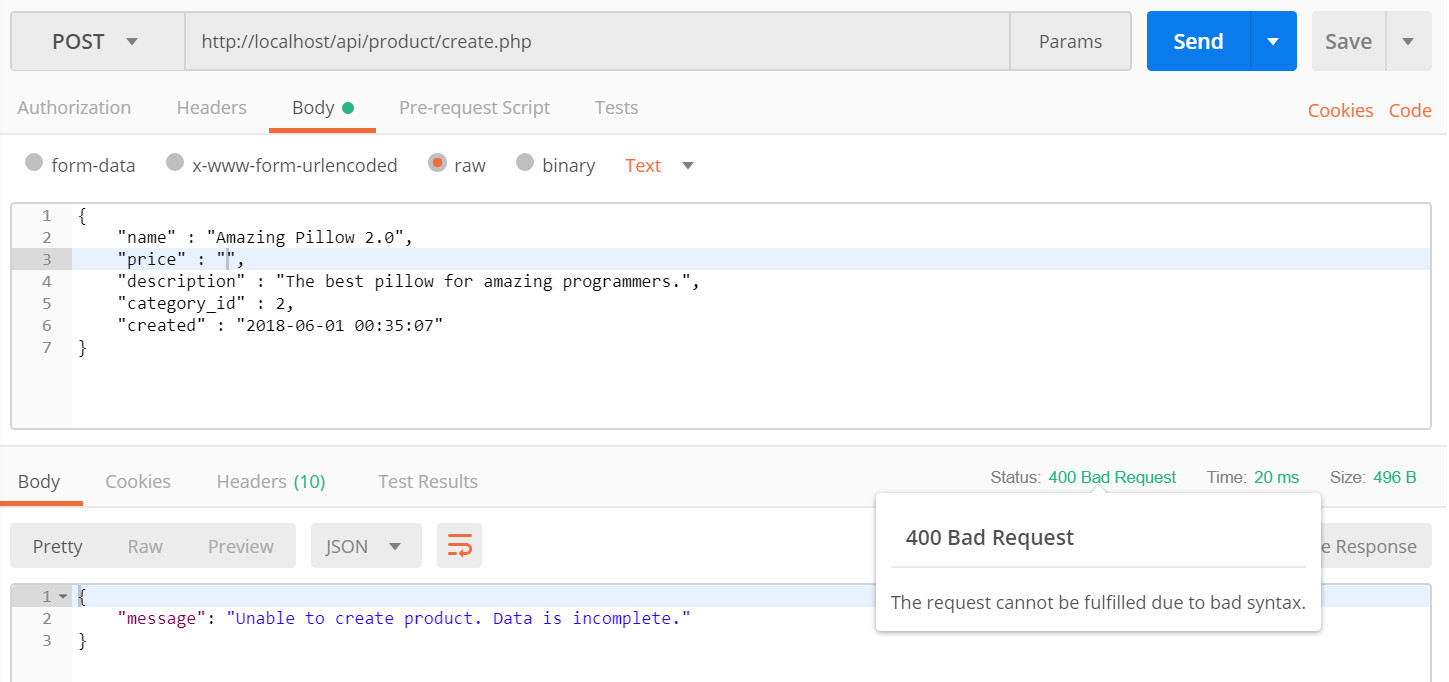
* It should look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/create-product-201.jpg?ssl=1)

* If the system is unable to create the product, it should look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/create-product-503.jpg?ssl=1)

* If the sent data is incomplete, for example, it is missing the price data, the output should look like this:



**Read One Product**

**Create read\_one.php file**

* Open product folder.
* Create a w read\_one.php file.
* Open that file and put the following code.

|  |
| --- |
| <?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."));  }  ?> |

**Product readOne() method**

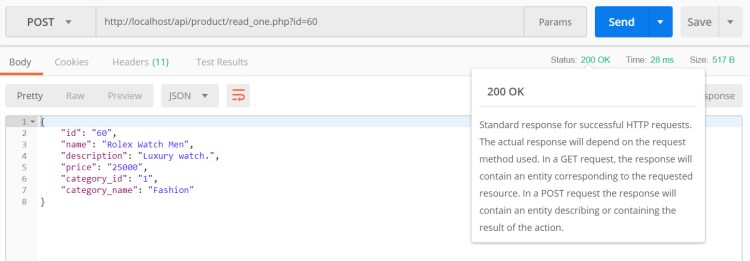
* Open objects folder.
* Open product.php file.
* The previous section will not work without the following code inside the Product class.

|  |
| --- |
| // 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'];  } |

**Output**

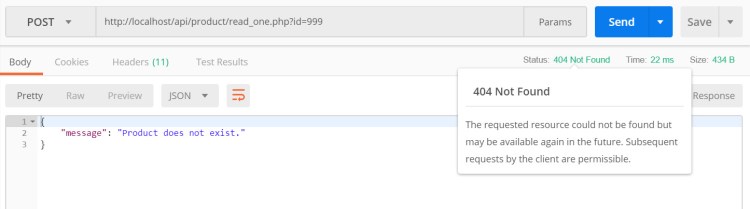
* First, we will test for a product that exists. Open POSTMAN. Enter the following as the request URL. Click the blue “Send” button.

|  |
| --- |
| <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)

* Next, we will test for a product that does not exist. Enter the following as the request URL. Click the blue “Send” button.

|  |
| --- |
| <http://localhost/api/product/read_one.php?id=999> |

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-one-404.jpg?ssl=1)

**Update product**

**Create “update.php” file**

* Open product folder.
* Create a new update.php file.
* Open that file and put the following code inside it.

|  |
| --- |
| <?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."));  }  ?> |

**Product update() method**

* Open objects folder.
* Open product.php file.
* The previous section will not work without the following code inside the Product class.

|  |
| --- |
| // 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;  } |

**Output**

Open POSTMAN. Enter the following as the request URL.

|  |
| --- |
| <http://localhost/api/product/update.php> |

* Click the “Body” tab.
* Click “raw”.
* Enter the following JSON value.
* Make sure the ID exists in your database.
* Click the blue “Send” button.

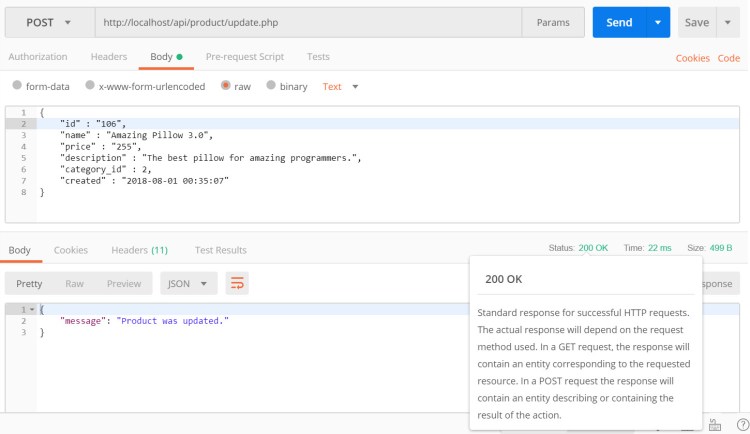
|  |
| --- |
| {      "id" : "106",      "name" : "Amazing Pillow 3.0",      "price" : "255",      "description" : "The best pillow for amazing programmers.",      "category\_id" : 2,      "created" : "2018-08-01 00:35:07"  } |

The product ID 106, is just an example. You need to specify a product ID that exists in your database.

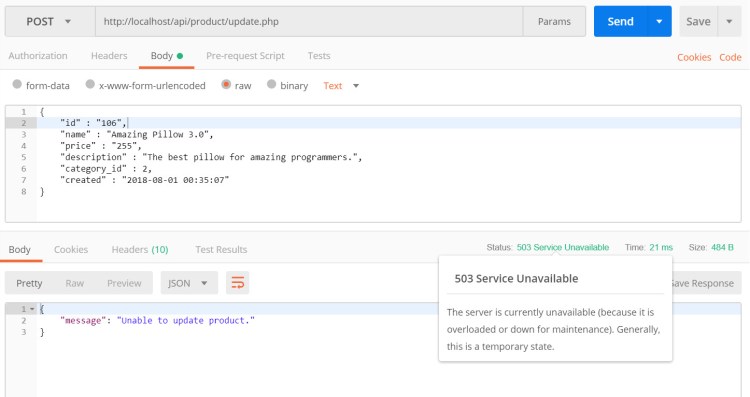
If you specify an ID that does not exist in the database, it might still say that the product was updated. It does not update anything on the database but the query was executed successfully without any syntax errors.

To prevent this, you need an extra validation where you check if an ID exists in the database. This feature is not yet part of our tutorial.

* If updating a product is successful, it should look like this:

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

* If the system fails to update the product, the output will look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/update-product-503.jpg?ssl=1)

**Delete Product**

**Create “delete.php” file**

* Open product folder.
* Create new delete.php file.
* Open that file and put the following code inside it.

|  |
| --- |
| <?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."));  }  ?> |

**Product delete() method**

* Open objects folder.
* Open product.php file.
* The previous section will not work without the following code inside the Product class.

|  |
| --- |
| // 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;  } |

**Output**

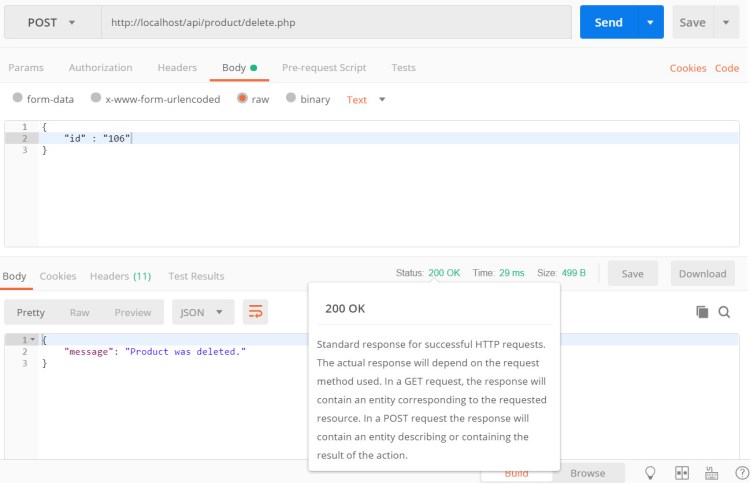
Open POSTMAN. Enter the following as the request URL.

|  |
| --- |
| <http://localhost/api/product/delete.php> |

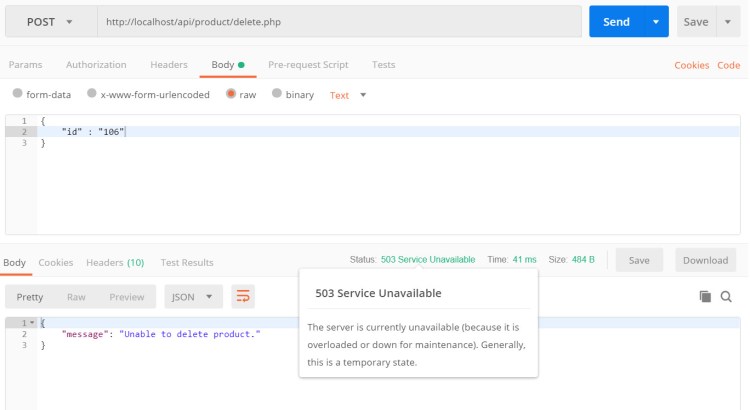
* Click the “Body” tab.
* Click “raw”.
* Enter the following JSON value.
* Make sure the ID exists in your database.
* Click the blue “Send” button.

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

* If a product was successfully deleted, it should look like this:

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

* If the system fails to delete the product, the output will look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/delete-503-1.jpg?ssl=1)

**Search Products**

**Create “search.php” file**

* Open product folder.
* Create a search.php file.
* Open that file and place the following code.

|  |
| --- |
| <?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.")      );  }  ?> |

**Create search() method**

* Open objects folder.
* Open product.php file.
* Add the following search() method.

|  |
| --- |
| // 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;  } |

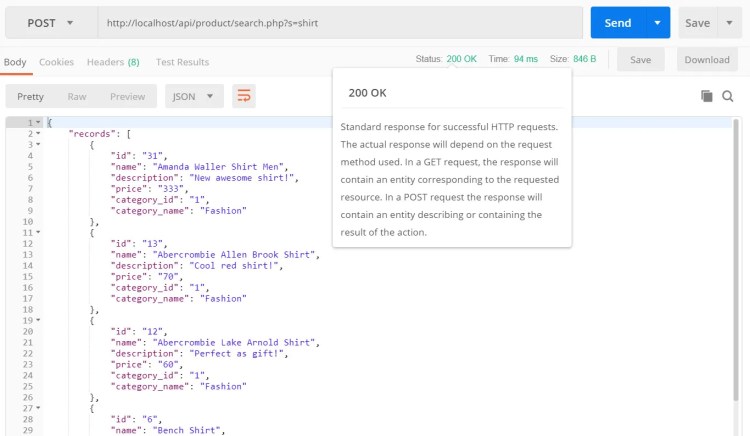
**Output**

Open POSTMAN. Enter the following as the request URL.

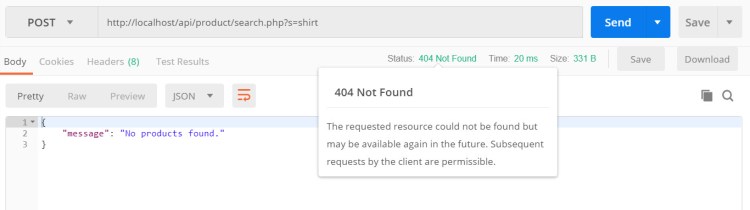
|  |
| --- |
| <http://localhost/api/product/search.php?s=shirt> |

Click the blue “Send” button.

* If there was a product found, it should look like this:

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

* If there are no products found, the output will look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/search-404.jpg?ssl=1)

**Paginate Products**

**Create “read\_paging.php” file**

* Open product folder.
* Create read\_paging.php file.

|  |
| --- |
| <?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.")      );  }  ?> |

**Create “core.php” file**

This file holds our core configuration like the home URL and pagination variables.

* Open the config folder.
* Create core.php file.
* Open core.php file.
* Place the following code.

|  |
| --- |
| <?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;  ?> |

**Create “readPaging()” method**

* Open objects folder.
* Open product.php file.
* Add the following method inside the product class.
* This method will return a list of records limited to what we set in $records\_per\_page of the previous section.

|  |
| --- |
| // 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;  } |

**Create “count()” method**

Still in the product class (product.php file), add the following method. The total rows are needed to build the pagination array. It is included in the ‘paging’ computation.

|  |
| --- |
| // 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'];  } |

**Get “paging” array**

* Create shared folder.
* Open shared folder.
* Create utilities.php file.
* Open utilities.php file and place the following code.

|  |
| --- |
| <?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;      }    }  ?> |

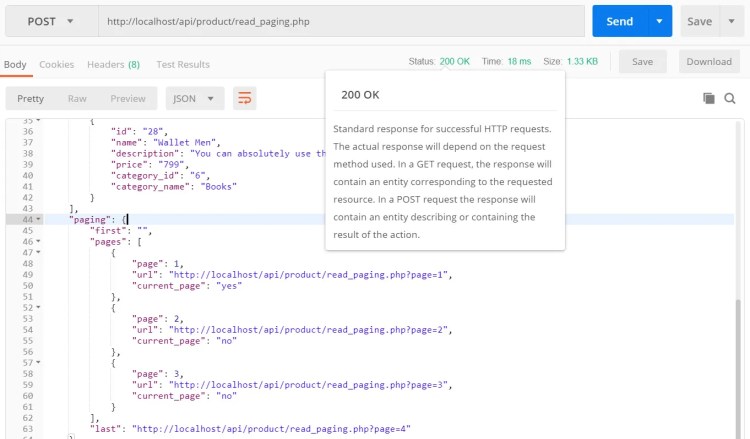
**Output**

Open POSTMAN. Enter the following as the request URL.

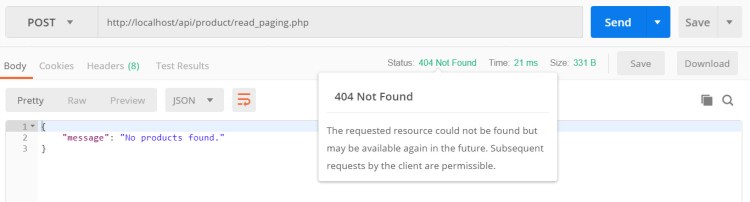
|  |
| --- |
| <http://localhost/api/product/read_paging.php> |

Click the blue “Send” button.

* If there are products found, scroll down to see the paging node. It should look like this:

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

* If there are no products found, the output will look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-paging-404.jpg?ssl=1)

**Read Categories**

**Create “category.php” file**

* Open objects folder.
* Create new category.php file.
* Place the following code inside the category.php file.

|  |
| --- |
| <?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;      }  }  ?> |

**Create “read.php” file**

* Create new category folder.
* Open that folder and create new read.php file inside it.
* Open read.php file and place the following code.

|  |
| --- |
| <?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.")      );  }  ?> |

**Add Category “read()” method**

* Open objects folder.
* Open category.php file.
* The previous section’s code will not work without the following code inside the category.php file.
* Add the following method inside the Category class.

|  |
| --- |
| // 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;  } |

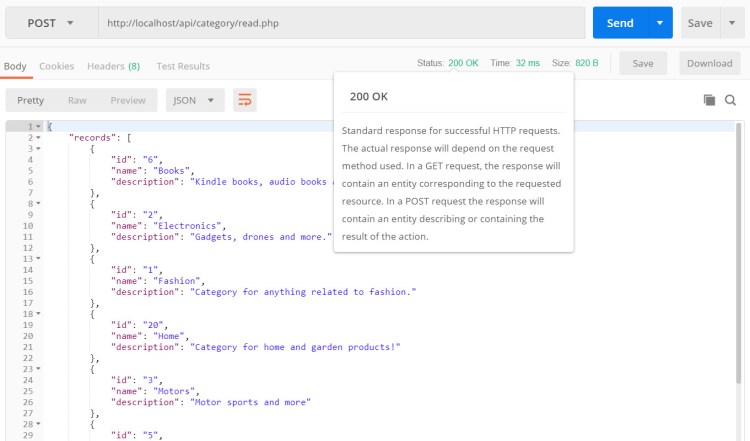
**Output**

Open POSTMAN. Enter the following as the request URL.

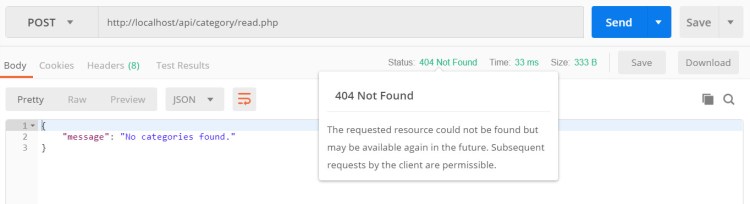
|  |
| --- |
| <http://localhost/api/category/read.php> |

Click the blue “Send” button.

* If there are categories found, it should look like this:

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

* If there are no categories found, the output will look like this:

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2017/02/read-categories-404.jpg?ssl=1)