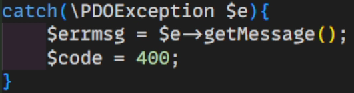
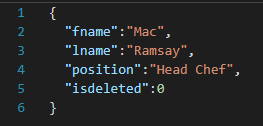
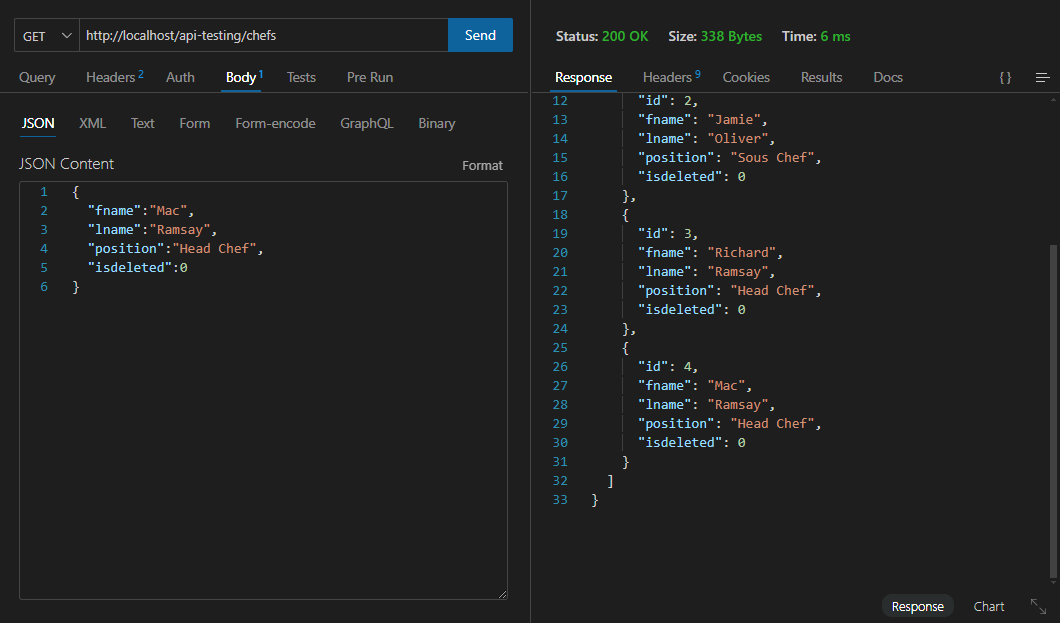
**2. Create, Update, Delete**

Post means adding records. Create first a case named “chefs” (desired name) inside the case “POST”. We will use JSON format to retrieve records. To do that, we will use the function file\_get\_contents. Place it in a variable named $body = json\_decode(file\_get\_contents(“php://input));. To check if it will create a record in case “chefs” add echo json\_encode($body);. It will be changed later as we will need to create a function inside post.php that will do this method that will be called later inside the routes.php. Rewrite the code inside case “chefs” echo json\_encode($post->postChefs($body));.

Create a public function postChefs($body). We need to create $sqlString = "INSERT INTO chefs\_tbl(fname, lname, position, isdeleted) VALUES (?,?,?,?)";. The purpose of this code is to insert records with the sql command INSERT INTO chefs\_tbl with the following fields fname, lname, position,isdeleted and assigning values marked as ? because that’s the binding parameter for user input. It acts as temporary storage for values that user is going to insert. We’re going to change the value retrieved as array. Using $values = []. We will use foreach($body as $value) to be iterated when $body variable will be stored inside $value. Then inside the foreach loop, we will add array\_push (same as append in javascript) to pass the value. To check change the return value from $body to $values to see if the code format to retrieve records was changed from just values and excluding fields names for each category. Also add $errmsg = “”; and $code = 0; We will also create try and catch to capture exceptions just like from the get.php. We will put $sqlString inside try. Inside of try, we will also add $sql = $this->pdo->prepare($sqlString). Prepare is a way for configuring for a preparation where it’s ready to pass values. It also protects sql from sql injection. A type of attack where attackers insert a sql statements to deploy all records. Then, we will execute $sql->($values); Also we will create $code = 200; and $data = null; Last is the return command return array(“data”=>$data, “code”=>$code); In catch, this will be the else condition.

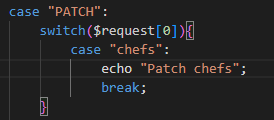
$errmsg will store $e->getMessage(); Get message will track if there’s invalid output and will display the error code equivalent to 400. Finally edit the return command at the last line. return array(“errmsg”=>$data, “code”=>code);.

To check if the code is successful. Go to thunderclient/postman and change it to post. Enter the values in fields just like this figure.

Once done, you can click the send and you’ve successfully added a new record to the database. To check if it was successful, you change the request method to get and it will display the added records there. Here’s an example. You can reuse the code to create a function that will also needed to insert records like menu or your own choice (just edit the fields).

**Put and Patch (Update)**

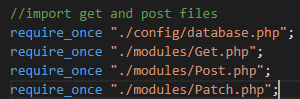
Patch is temporary update and put is permanent. We will use patch because we’re just partially updating. Create a case in routes.php named “PATCH” then add the following.

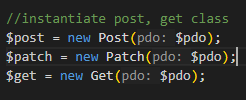


Update in database.php the header consisting of post, get and add patch onto it.

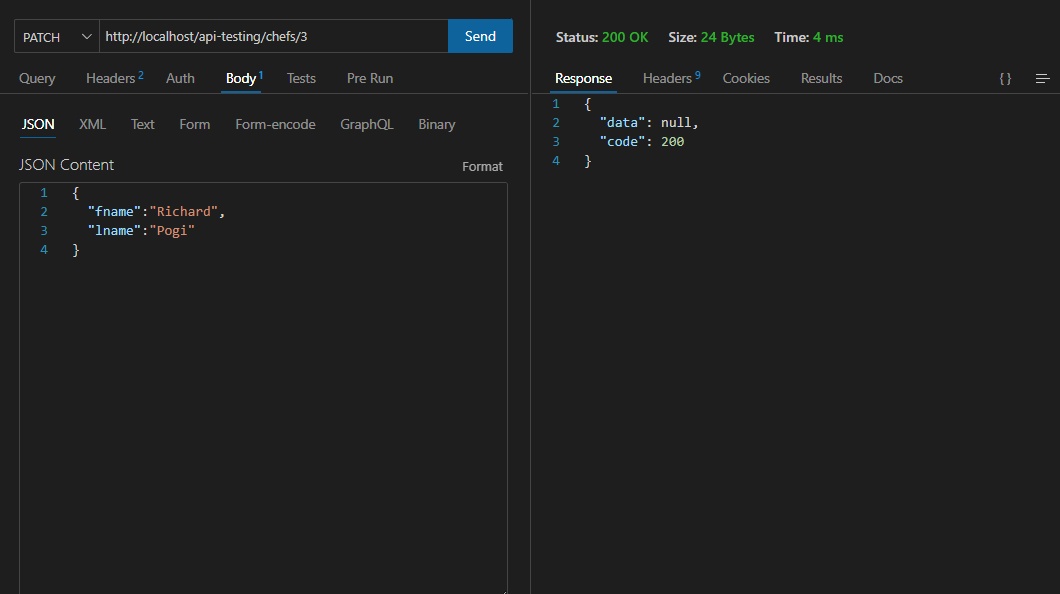
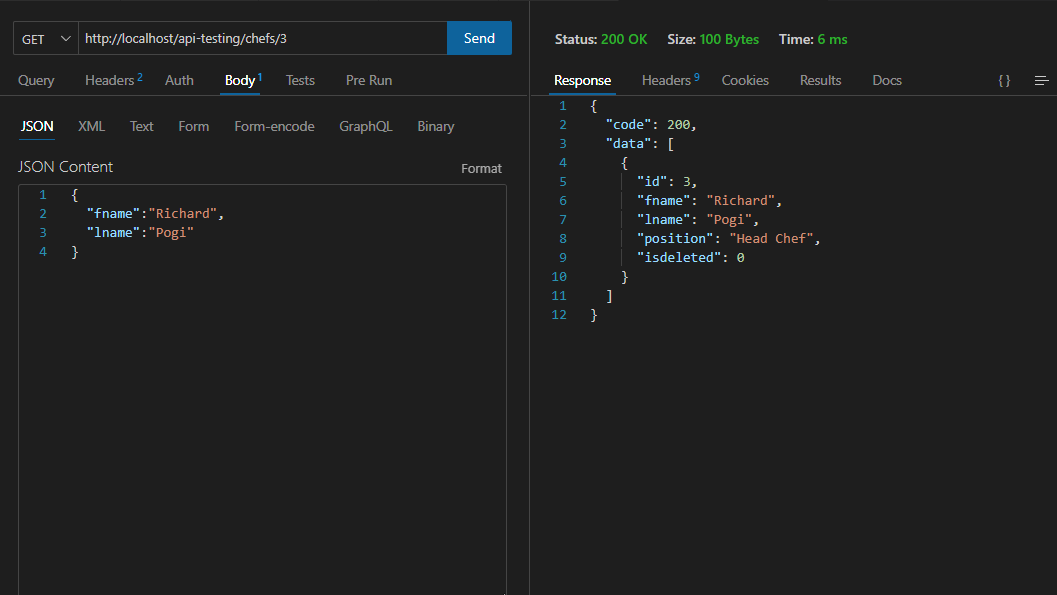
Create a file named patch.php inside the modules folder and copy the following code from post.php. Remove unnecessary codes but save the previous function that we did and just change the name from postChefs to patchChefs. We will just change the $sqlString command to this. The command for updating records is UPDATE chefs\_tbl then the fields. We should put the condition WHERE because it will change all fields if there’s no WHERE command. The $id will be passed on $values after the we process the whole loop and is now an array. We will add the $id, by using array\_push($values, $id). The process for updating is done and we just need to import it in routes.php.





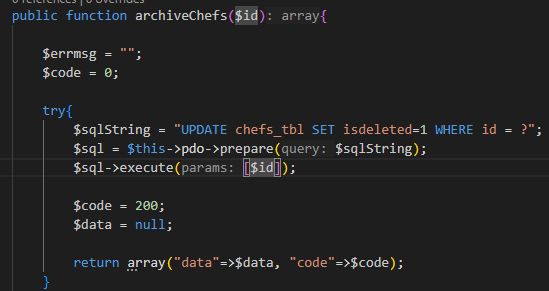
We will also create patch object

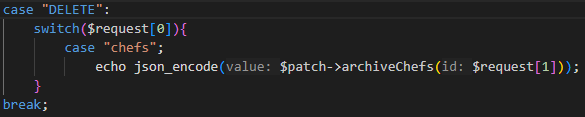
To check if we’ve successfully updated a record.

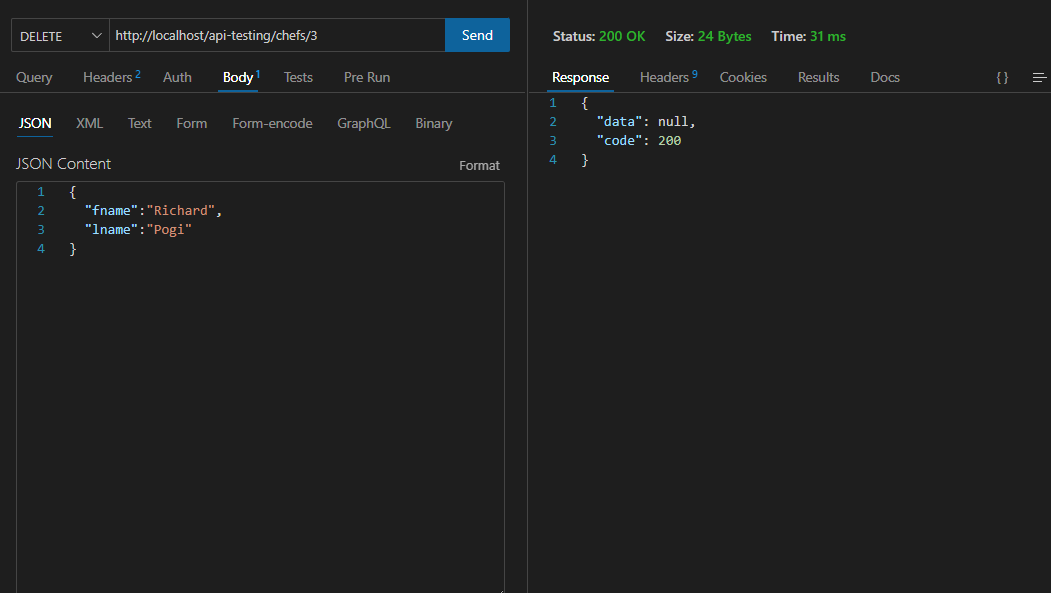
Then change the request method to get to see if it’s successfully updated.

**Archiving**

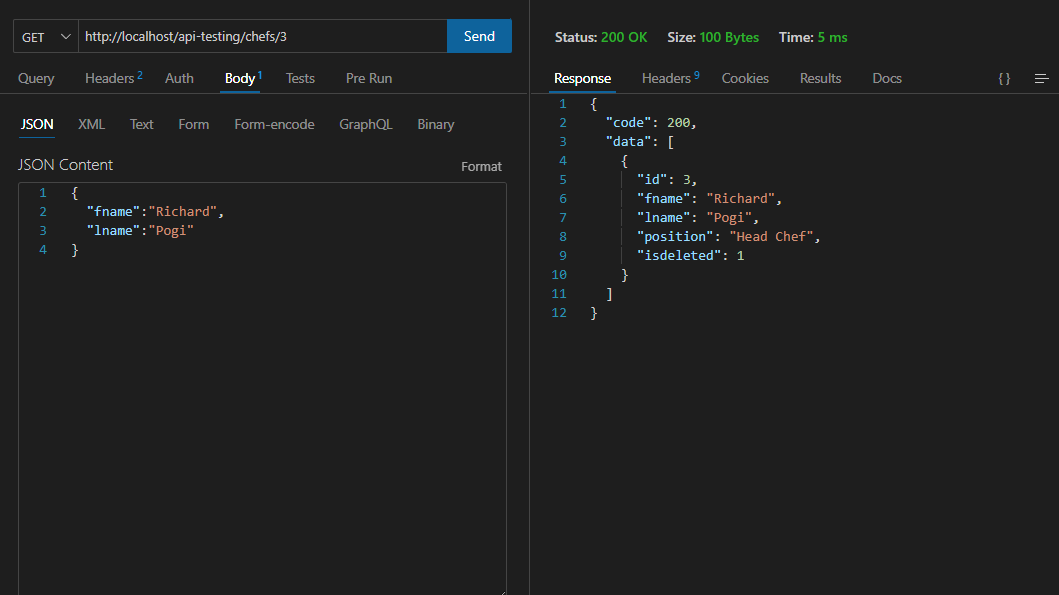
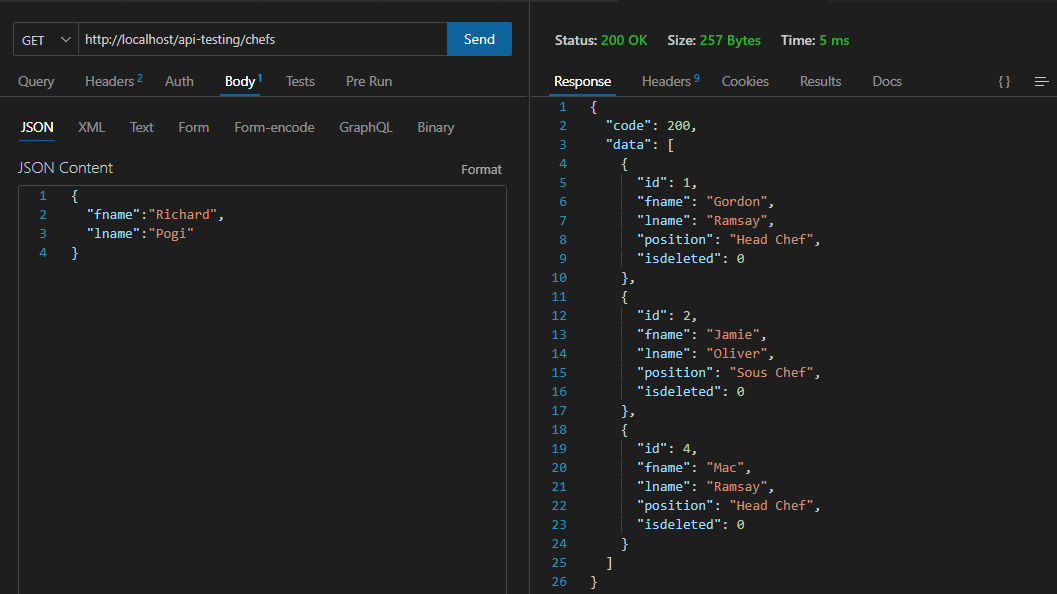
Copy the code for patchChefs. Remove the following, $body, $values = [], foreach condition, array\_push. Inside the $sqlString remove fname=?, lname=? and change it to isdeleted=1. In $sql->execute change the $values to [$id].



Next step is that we will create a case in routes.php and name it “DELETE”.

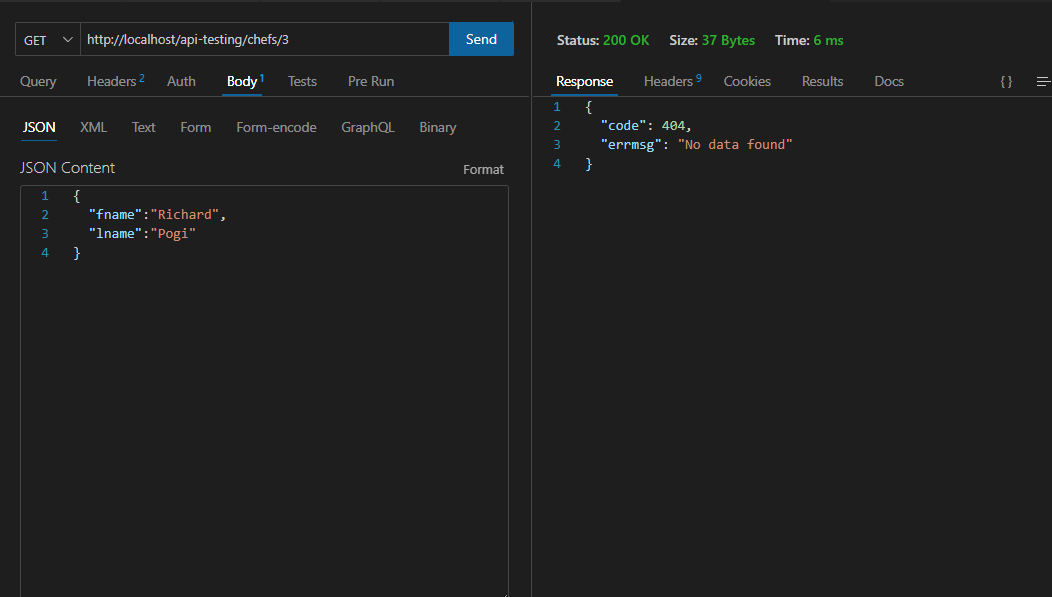
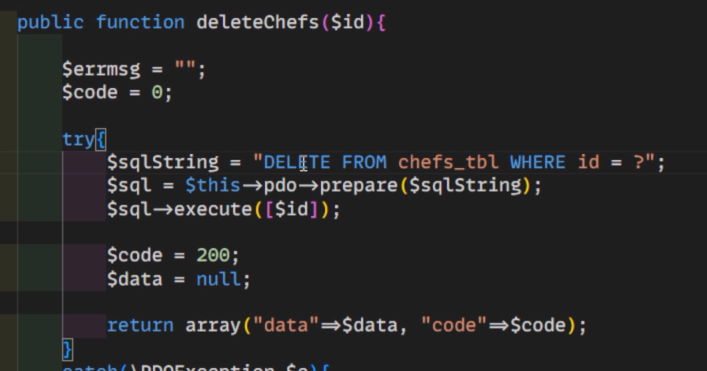
To check if the code is successful.

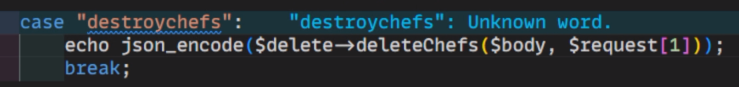
Then change the request method to get to see if the isdeleted value is now 1.

However, we must create a condition where once a record has been deleted it shouldn’t be existing or can be seen once we get the value. So, we will be modifying in get.php the condition. In get.php $sqlString we will add WHERE isdeleted=0 and change inside the if condition the WHERE to AND to prevent redundancy. We will check if it now filters the isdeleted.

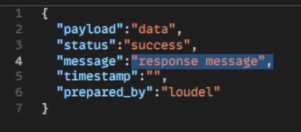
To check the deleted/archived record we will try to get it and it should display an error message saying that there’s no data found.

**(OPTIONAL)**

You can add Delete.php if you want. Just copy the code from patch the archiveChefs function only. You just need to change archiveChefs to deleteChefs then change the sql command UPDATE to DELETE FROM, remove isdeleted and leave WHERE id =?.

Then make a case inside “DELETE” and input the following code.

This is only optional and not advisable because once you deleted a record it’s not retrievable rather than just archiving it.

Next Topic.Added modifications for the response. Like this example instead of this default message. Log in, encryption and more.

