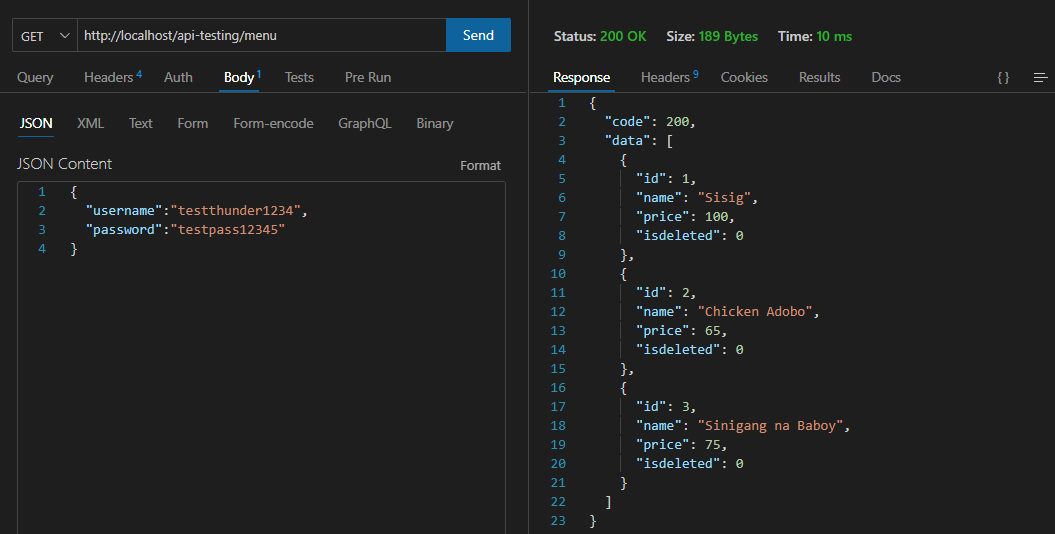
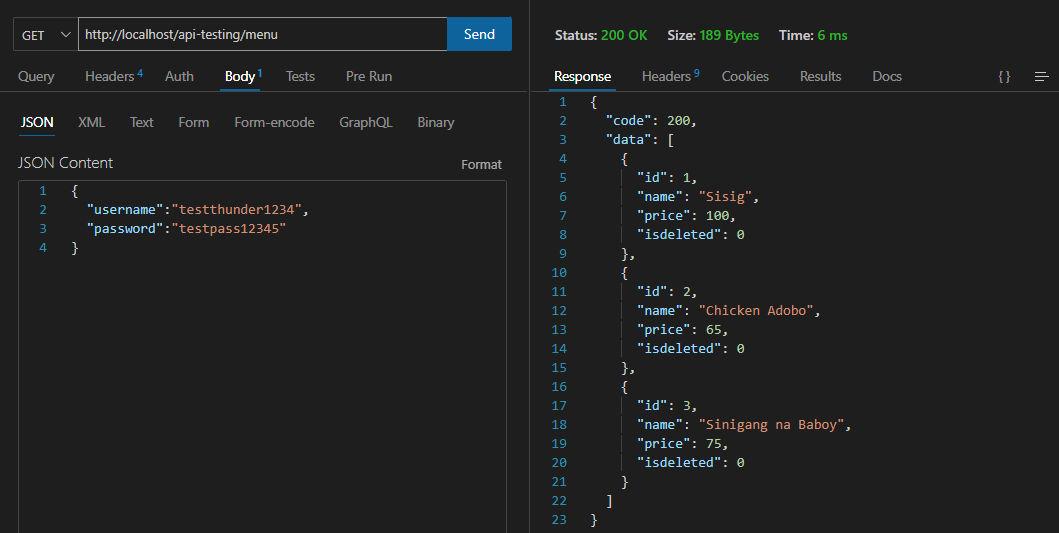
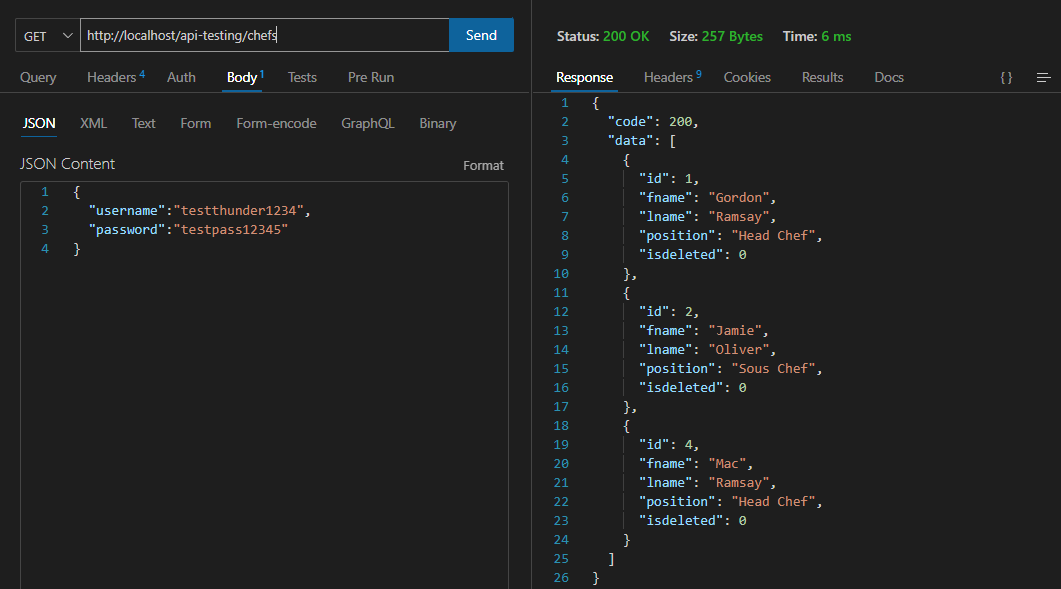
**5. Refactorization**

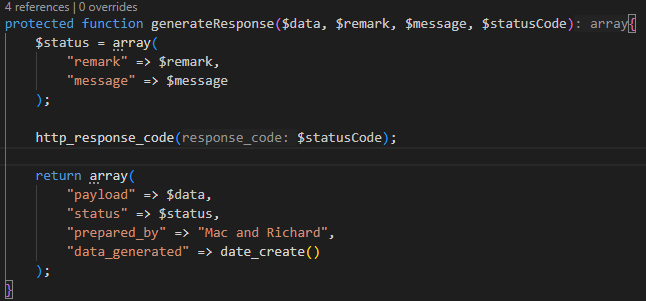
Create a case inside “GET” and name it Menu (you can create your own). Copy the whole code on case “CHEFS” and paste after it. Then, we will go to get.php and copy the whole function getChefs. We’re going to modify the name of the function and change it to getMenu() then we will also change the sql command to menu\_tbl (we will create that in database later). We will now create a new table in database named menu with fields of id, name, price and isdeleted. We will now try it in thunder client to see if the data will be retrieved.

We’re going to refactor the duplicate codes because the function for getChefs and getMenu are just the same. We’re create a single instance of the code and we’ll reuse it. We will create inside the modules folder a new file named Common.php. This file will be the refactor which makes the code easier to implement, reducing complexity, improving readability and overall quality of the code. We’re going to the code inside function getChefs and paste it inside the Common.php. We create a protected function getDataByTable and pass the variables $tableName, $condition, \PDO $pdo. We modify the sql command to become accessible to all functions (getChefs and getMenu) where we replace chefs\_tbl into $tableName and isdeleted = 0 into $condition. Inside the if condition located in try, we will remove the $this-> because we’re not calling pdo anymore so we just assign $pdo. Then, we will go back to get.php to remove all the codes inside the function leaving only the $sqlString and the if condition. We will replace the name $sqlString to $condition and we will also change the sql command leaving only isdeleted = 0. We will also rename the $sqlString variable inside if statement to $condition. Then, we will return the value by calling the instance using $this->getDataByTable(‘chefs\_tbl’, $condition, $this->pdo). Also apply this method to the menu function and test if it works.

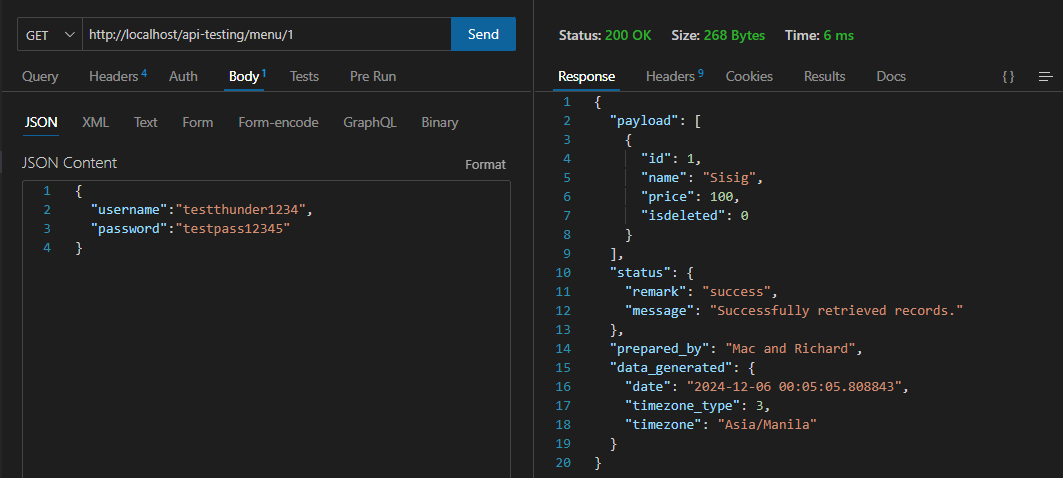
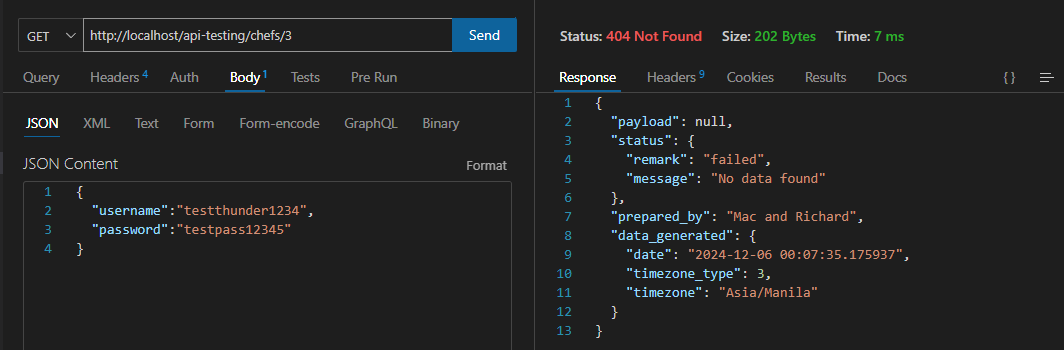




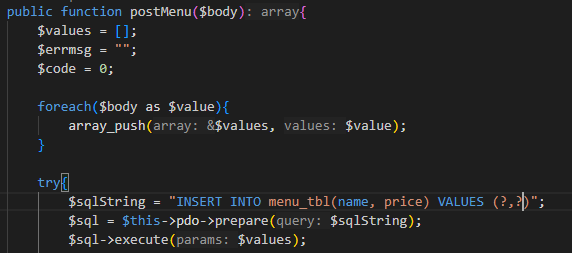
If the API has inner joining, we can create another version of function getDataBySQL. We will pass $sqlString to it instead of $tableName and remove $condition. We will remove $sqlString and we will pass sql string to be constructed with the string with inner join. (OPTIONAL)

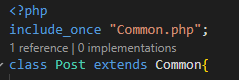
When we send request to thunder client, we may notice that even if we successfully retrieved or failed the status code returns 200 (which shouldn’t be because failed status code = 400). To fix that, we will create a function that will generateResponse. Create a protected function generateResponse($data, $remark, $message, $statusCode). Inside of it, we will create $status = array. Inside of it we will assign “remark” and “message”. Then we will create a return command with fields “payload”, “status”, “prepared\_by”, “date\_generated”. Last, we will add http\_response\_code($statusCode). This function will verify if the record input is existing, if it exists it will generate this payload in an array format. Nothing changed on status code as it is default at 200 (OK). If it doesn’t exist, it will still provide the same return value format yet the status code is changed to 404 not found.

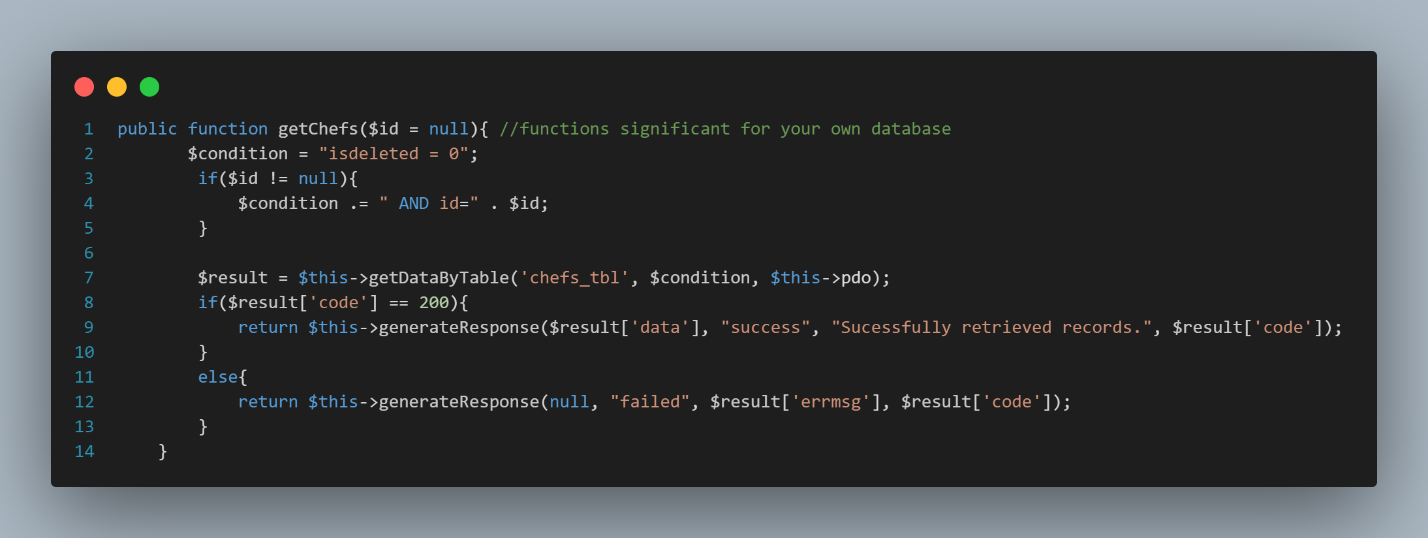
Once we’re done, we will go back to get.php and will do modifications. We will remove return command and change it to $result to store the records. We will create an if condition to check if the code = 200, it will return the data/records stored inside $result, remarks which is “success”, message which is “Successfully retrieved records.” and the status code. We will do it also in else condition, we will assign null for data since there are no records found, remark which is “failed”, result coming from ‘errmsg’ from the getDataByTable and status code. To test:



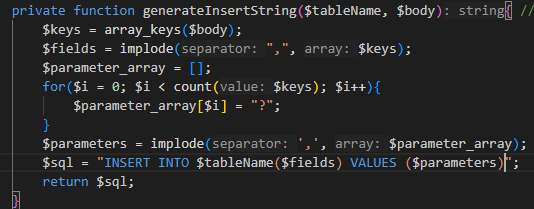
You can create separate routes for request methods (GET, POST, PATCH). (OPTIONAL)

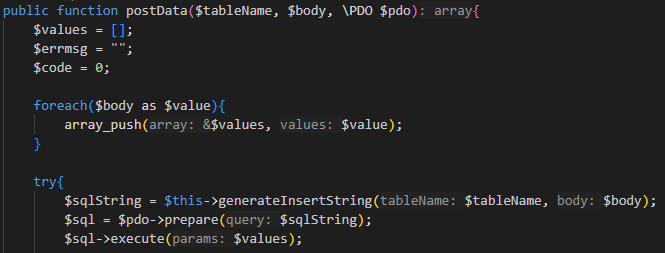
We will also add a case “MENU” in post. Just repeat the same process where we copy the function from postChefs and modify the name of the function together with the sql command.

When we refactor like we did in GET and POST we need to import the file Common.php to our request methods get.php and post.php. To do that, we will use include\_once “Common.php”. Also, we will insert extends Common in class because we are now using the concept of inheritance. (Forgot to mention)

Inside get.php code after refactorization. (Forgot to put)

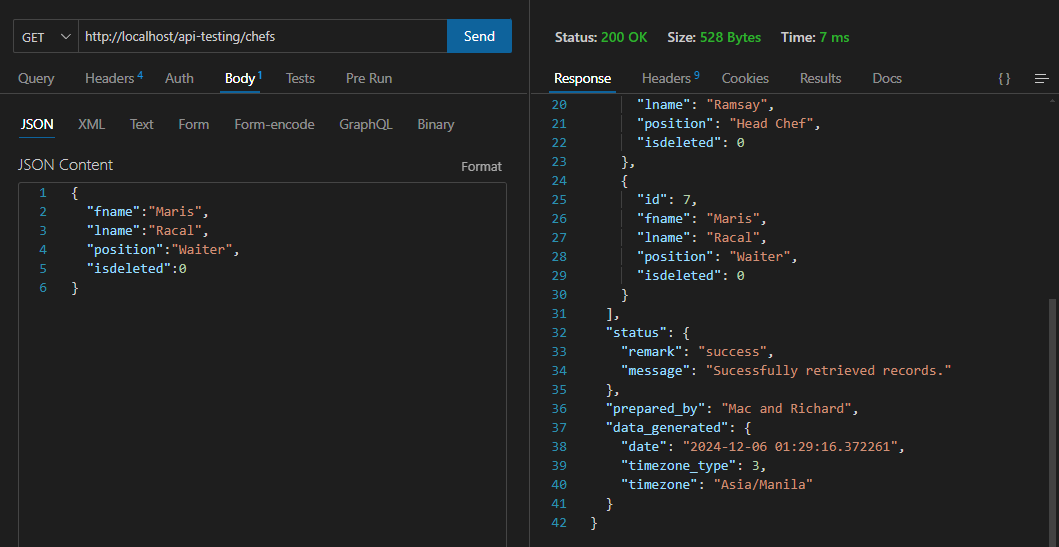
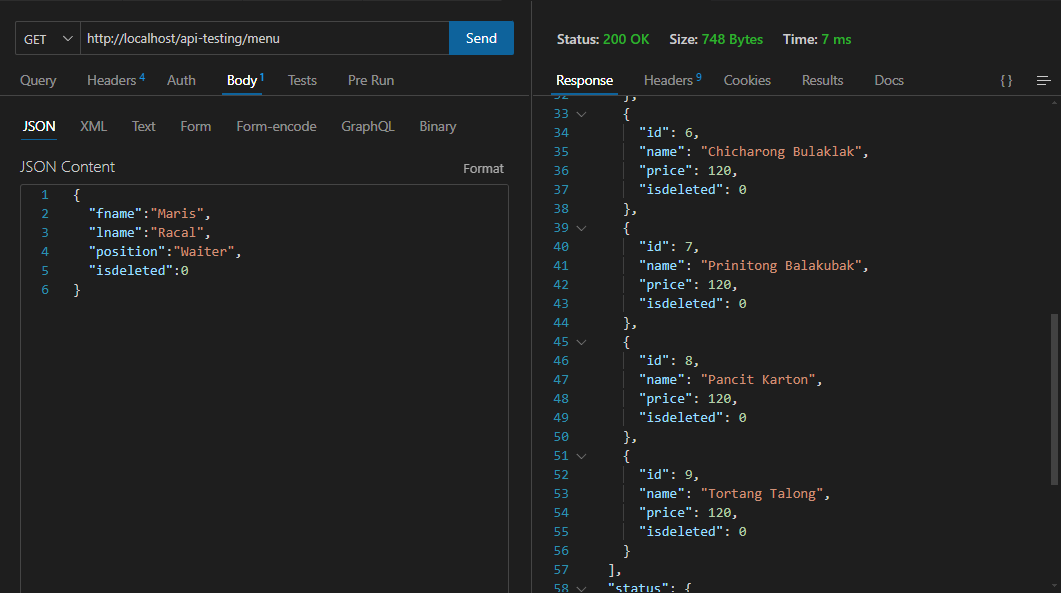
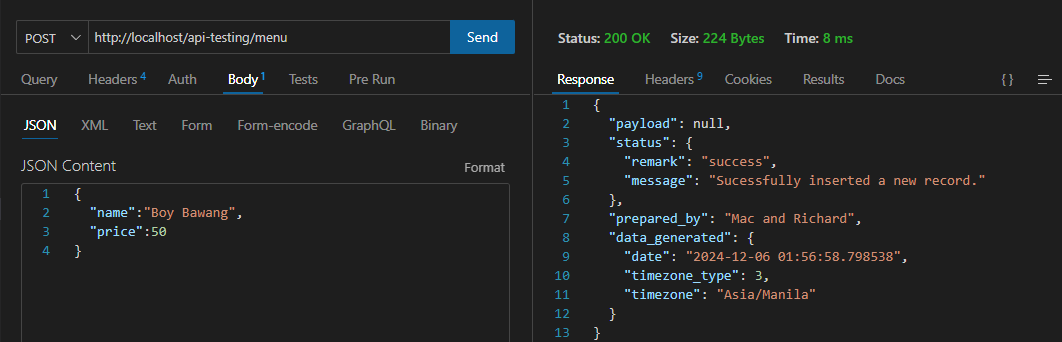
We will create a refactor code also for POST. To do that, just do the same thing that we did earlier in GET. We need to copy the code from post.php to common.php. We will put it in a public function that will be named postData($tableName, $body, \PDO $pdo). Since this will be different than GET because we have different values to put like fields and parameters coming from chefs\_tbl and menu\_tbl. To fix that, we will create a private function named generateInsertString that will work on the sql command for both postChefs and postMenu.

Array\_keys return all the keys or a subset of the keys of an array (numeric or string). Array\_key($body) returns [‘fname’, ‘lname’, ‘position’,’isdeleted’]. Then implode will then add commas to separate them like “fname,lname,position,isdeleted”. Meanwhile, we constructed a for loop in $parameters to know how many parameters are present in a table. When the loop determines how many it will generate question marks and that will be stored inside $parameters that separates the question marks with commas. Last, $sql is the sql command stored in string format that will be returned at the end. WE will also just add payload same from what we did in get. It will appear once we inserted a new record. To test:

Back in postData function, we are going to call in $sqlString the instance of the private function that we made which is generateInsertString. (also forgot to include) (last part is testing)

We will go to post.php to return postData function. Just copy the code in Get.php. It will just deploy the payload once we create a new record.



Results