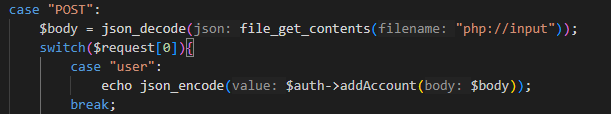
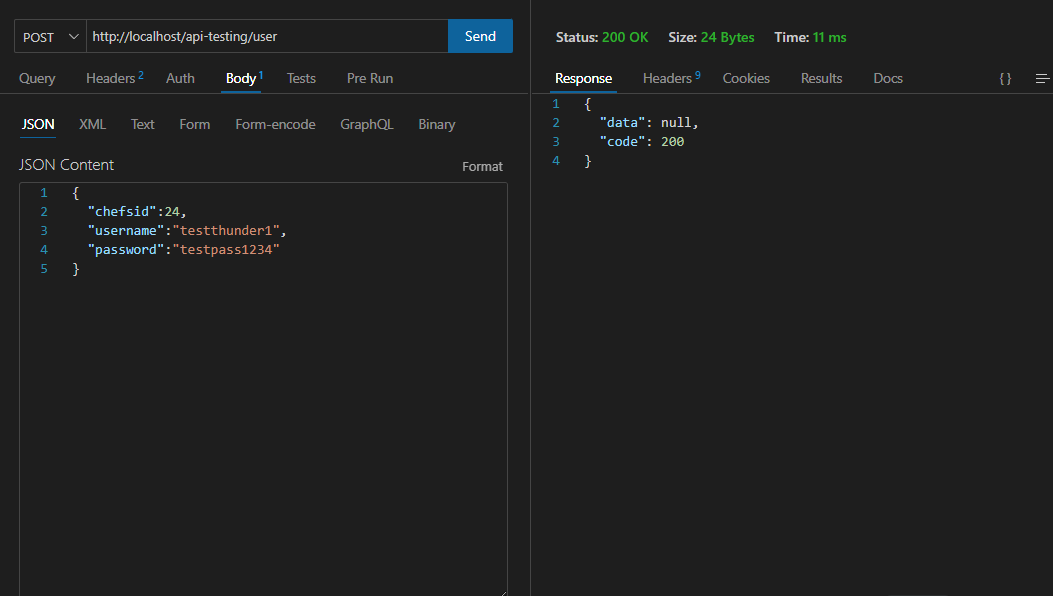
**3. Login, Password Hashing, Password Validation version A**

In phpMyAdmin, create a table named accounts\_tbl. Inside of it, there should be fields id, chefsid, username, password and token. ID must be auto\_increment, both chefsid and username must be unique so that there will be no repeated input.

Create a new class named Auth.php in modules folder. Copy the values from post.php and remove unnecessary codes. Change the name of the function to addAccount. It will be responsible for inserting the values. Modify the $sqlString by changing the table name from chefs\_tbl to accounts\_tbl then change the fields. Chefsid will be computer generated and will be implementing sort of algorithm to generate chefsid.

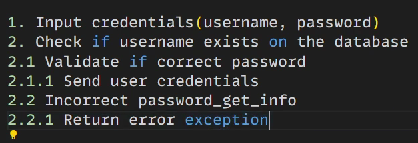
Import the Auth.php to routes.php and also add the object.

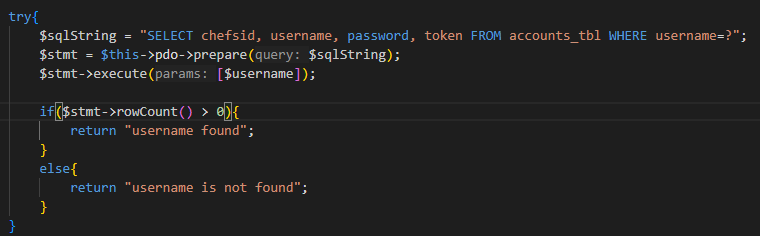
Create a case named “user” inside the case “POST”. This is to call the class authentication and its function.

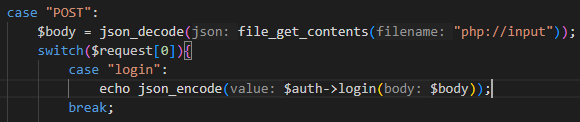
Trying to insert now to see if the code is successfully working.

Check in the database if it’s successfully added.

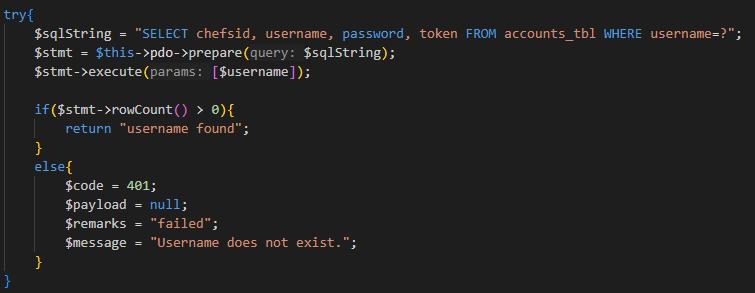
Sequential cycle of the program.



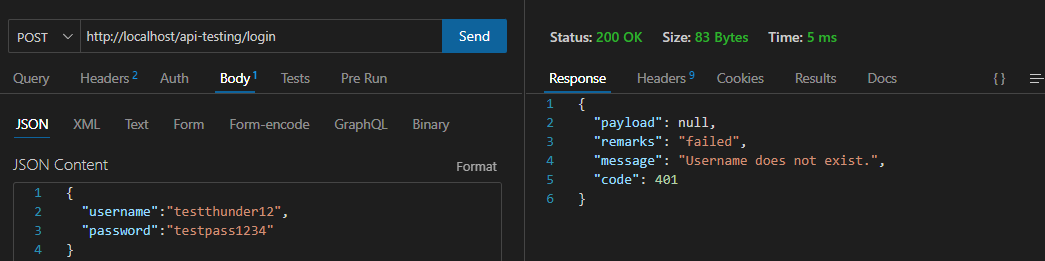
Create a function inside the Auth.php named login. Remove from $values to foreach loop. Define $username and $password. Input the $code, $payload, $remarks and $message. Copy the try and catch function from the addAccount and modify it to SELECT chefsid, username, password, token FROM accounts\_tbl WHERE username=?. Pass inside the sql execute the $username. We will create a function to check if the user exists but first it should have to pull a record. We create an if-else condition where it will verify by checking if the rowCount value is greater than 0 meaning once an account is existing it will return that a username was found and the else block if the user is not found.

We will also create a case inside “POST” named “login” just the same process as what we did with the case named “user”, we just have to change the values inside the parenthesis.

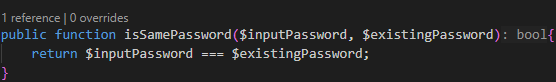
To check if it was successful, we go to thunderclient. We will be going to input our registered username and password before to see if it exists.

You can modify what will be the response in else once the username was not found just like this:

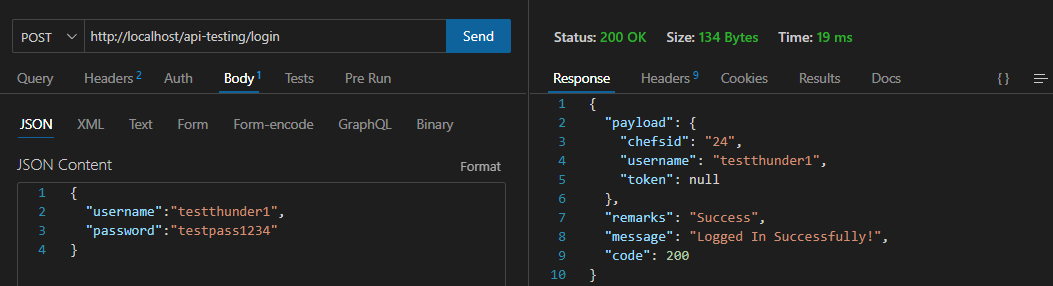
We will add a return command so that it will be echoed once we run it. (After catch condition)

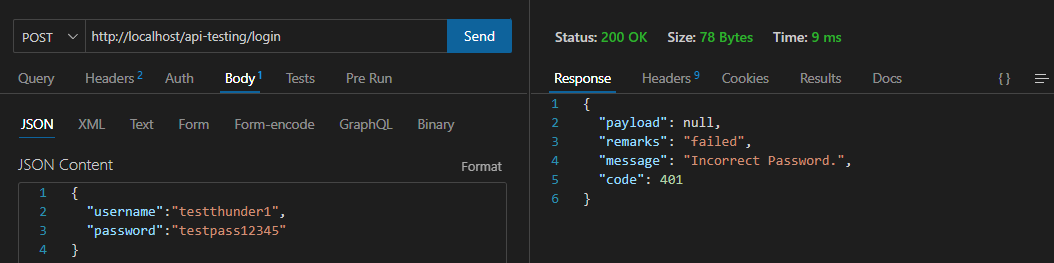
Here’s the result:

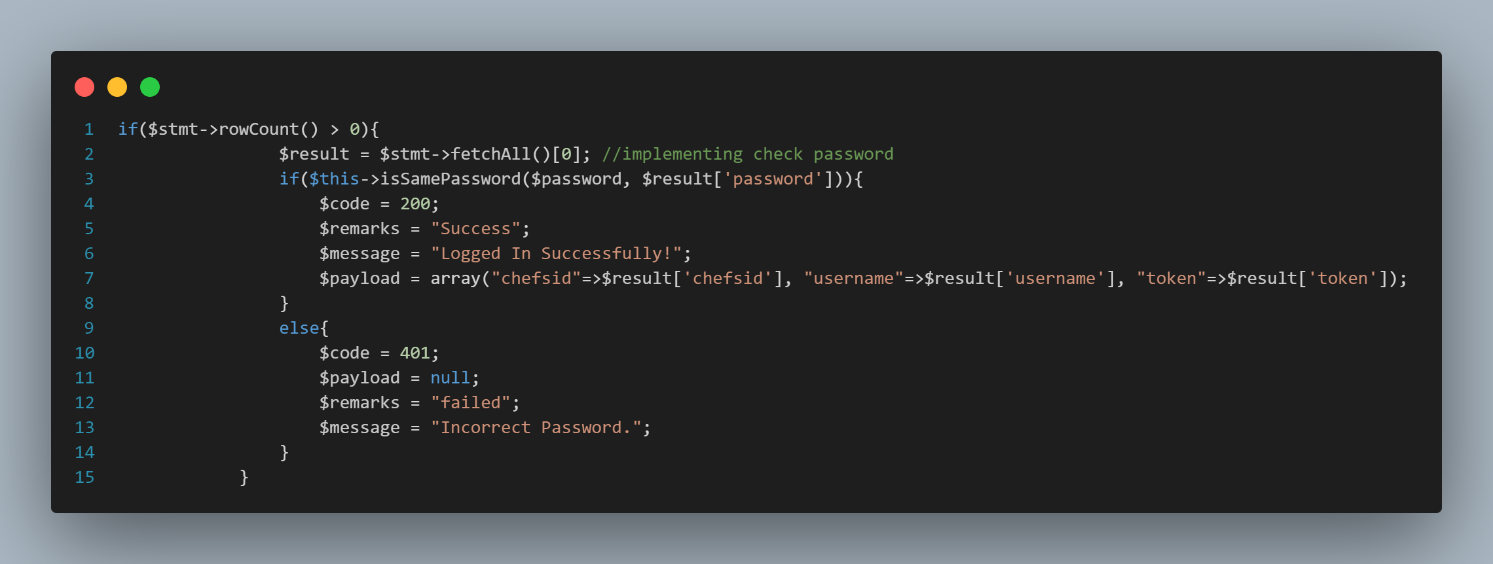
**Password Validation**

We created a public function that will check if the input password is the same with the existing password. (in advance, change it to private function because it should be available in authentication section/class)

After that, we modified inside the if condition because we will add the validator to check if the password is same. In the if block we have the code equivalent to 200 which means successful/okay, remarks, message and added array where it will display chefsid, username, and token of the user. In the else block, it will be executed when the input password doesn’t match the existing password of the user.

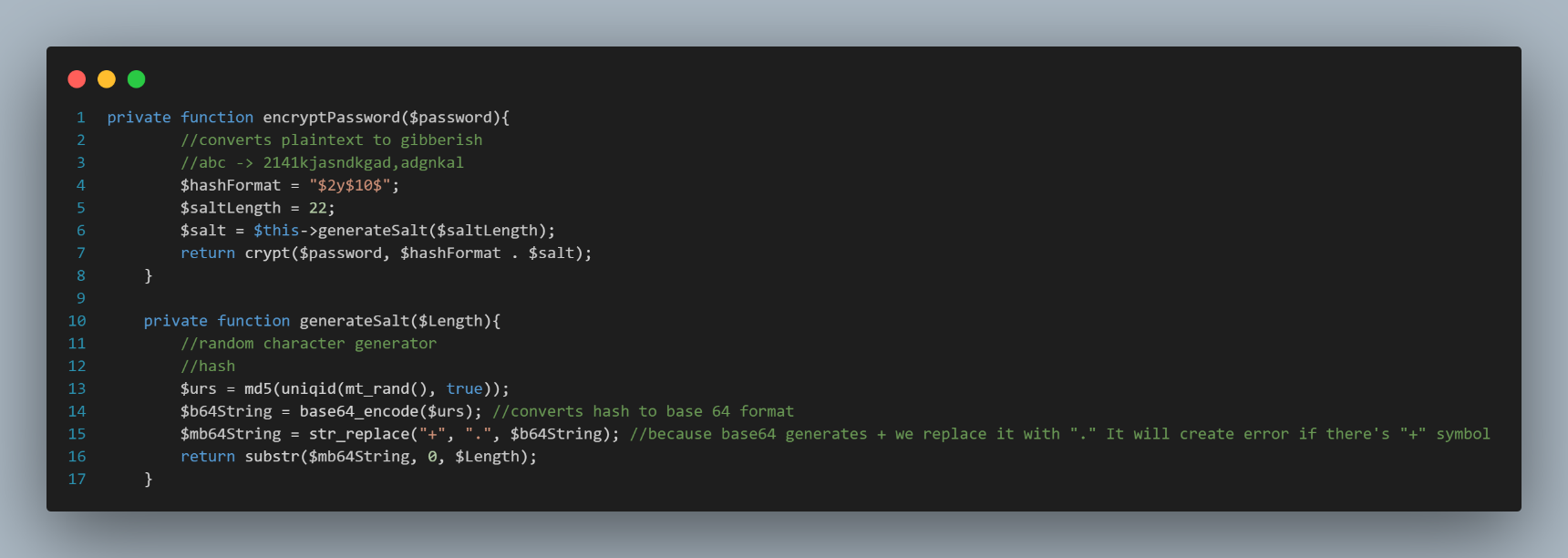
Example when the input password matches the existing password:

Example when the input password doesn’t match with the existing password:

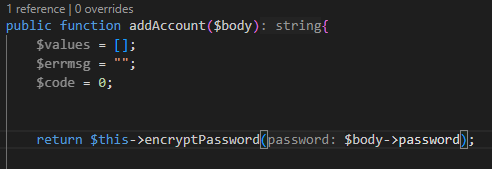
The modified code for the password validator

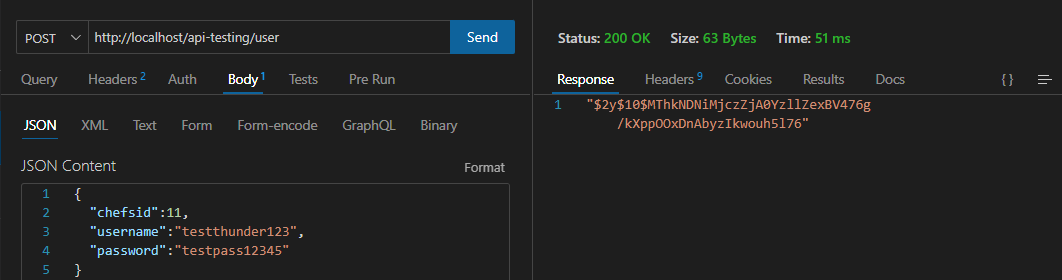
Token is ticket to access resources. (Will be done in future discussions)

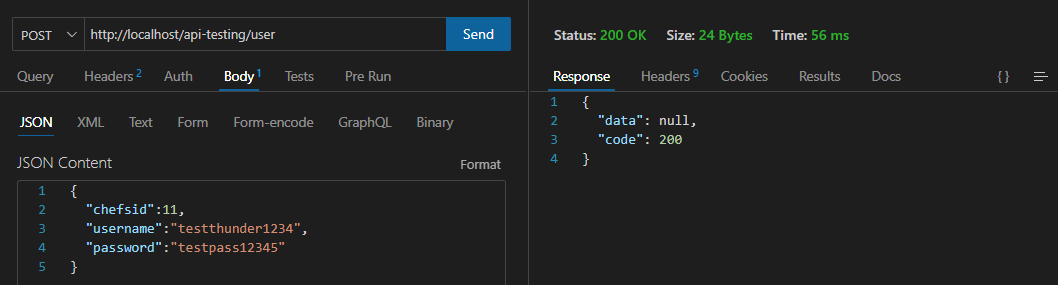
**Hashing**

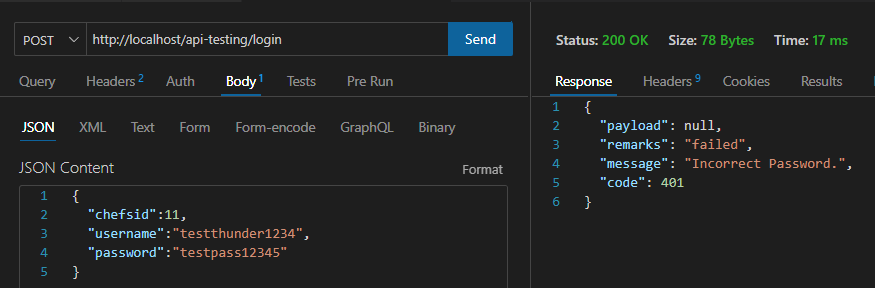
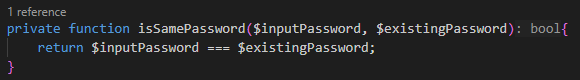
Since a database is prone to hacking. Once the hacker accesses the table, it can see all the information of the user and will be able to breach the account. To prevent that from happening, we will use a method to change the password to a word that is gibberish yet still stores the same value from the registered password. To do that, we will create a private function named encryptPassword($password). We will use the implementation called blowfish (to be discussed in the future). Inside the function is we will the hash format we were going to assign together with the saltLength (to be further discussed). Also adding the variable salt that will store the value from the generate salt function with the passed length format. It will be returned as encrypted password already. Then, we will create the private function generateSalt($length). This will create a random character generator or the hash process itself. We will use the md5 hashing. We will pass inside the parenthesis uniqid, mt\_rand() and true. This will generate uniqid that will be converted using md5 function then converted into base 64 string format. We will use the base64 \_encode($urs). Then, we will remove the “+” characters because it will create an error if it’s randomly included in the encryption. Create $mb64String = str\_replace(“+”, “.”, $b64String). Last, we will add the return function where it will return the $mb64String with the length assigned from the $saltLength which is valued at 22.

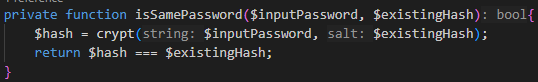
To check if the process is right we will insert return $this->encryptPassword($body->password) inside the addAccount.

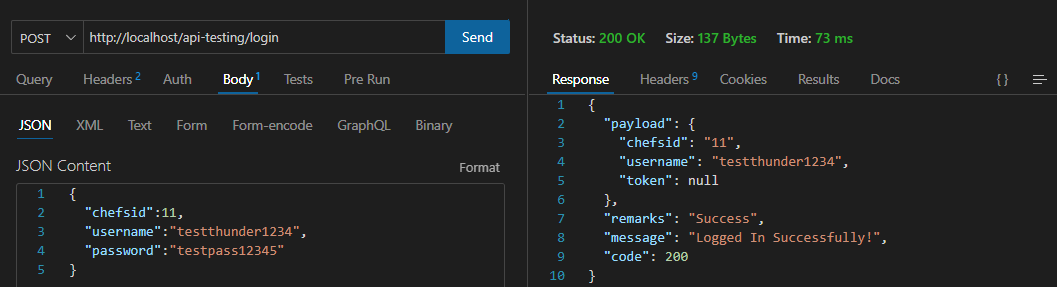


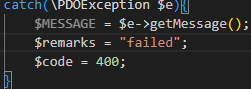
We will put the value of hashed password as the password before we save into the database/table. We will assign the value of $body->password the encrypted password.

Let’s check inside our database table:

Since the encrypted password is already set. When we try to input the password, we will get an error because the hashed password is not equivalent to the stringed password that was set before. So, we will modify the code from this:

To this:

We will change the value as $existingHash because it was already changed. Then, we will add the variable hash containing the command crypt (returns hashed string as string) and we will call the $inputPassword and the $existingHash. Last, we will modify the return and rename it to $hash === $existingHash. It’s now accepting the input/registered password the same as the hashed password.

Modified this line of code inside login class. Added $remarks = “failed” and change the $errmsg to $MESSAGE. This will be executed once the sql command is not correct.