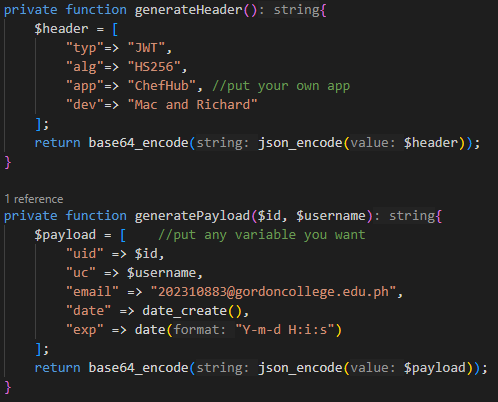
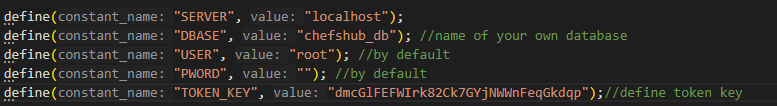
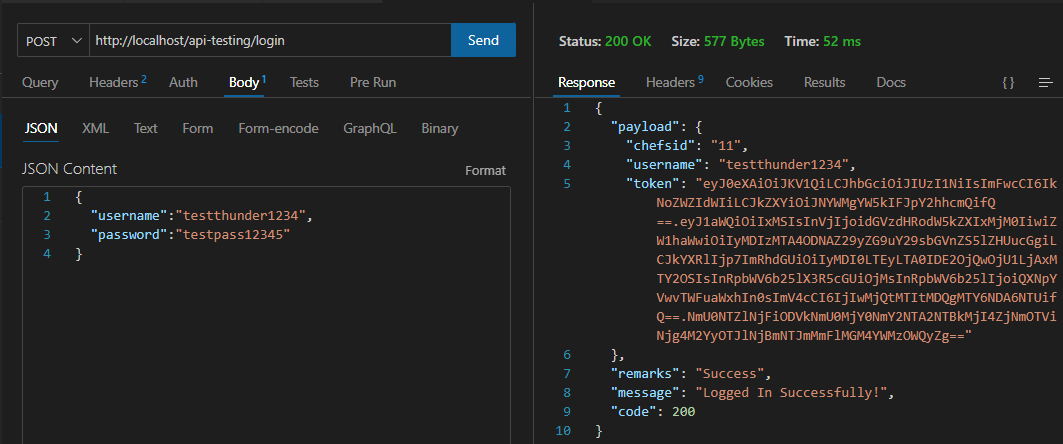
**4. Hashing, JWT Token Version A**

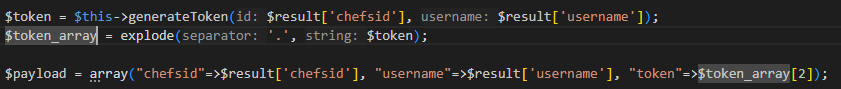
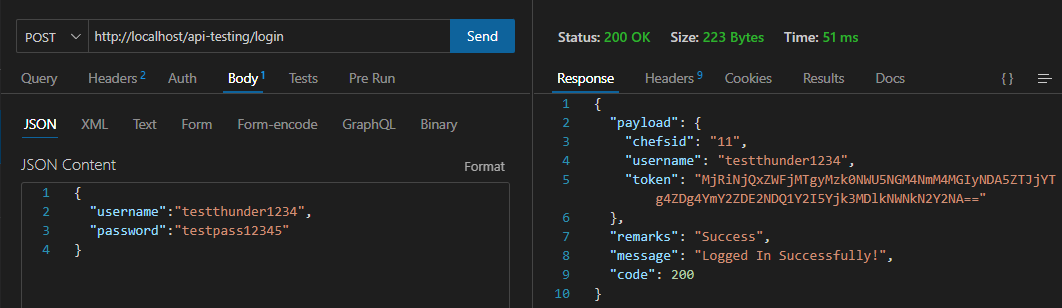
Token is something saved from database. We will use JWT also known as JSON Web Token. Go to jwt.io to know more about JWT. As we can see in the site, it has three parts: header, payload and verify signature. When you login, we will be going to generate a new token before returning the information to the user we will need to generate token and that token will be included with the information passed with user. We will be going to implement some function/method that will generate the token. We will be generating 3 functions: function for generating token, function for generating payload, and function for generating header.

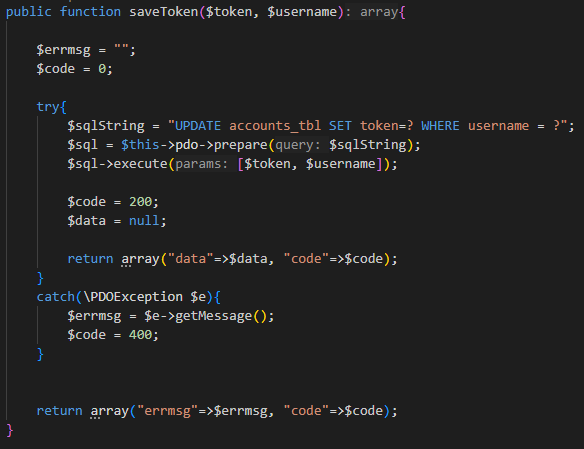
We’re create 3 private functions namely $generateHeader, $generatePayload, and $generateToken. The contents for both header and payload function are just the same. We just have to customize according to our desired output. Yet, we’re going to call $id and $username since we’re going to display it in the payload.

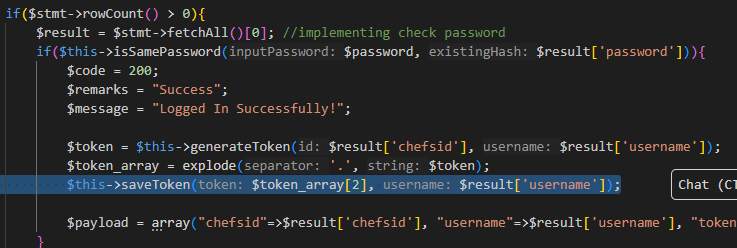


In generateToken we just have to call the functions from header and payload then create a new variable named signature that will be hashed in hmac with the algorithm “sha256”. The data will be the concatenated version of $header and $payload and we store it inside “TOKEN\_KEY”. We’re going to return the concatenated version of $header and $payload and the concatenated base64 value of $signature. Next, we’re going to define the TOKEN\_KEY inside database.php. Go to <https://acte.ltd/utils/randomkeygen> this site that generates a 256 encrpytion key as example or you can create your own. Put it inside the define command to assign the value of “TOKEN\_KEY”.

Inside the function login, we’re going to call the token by creating $token = $this->generatedToken($result[‘chefsid’, $result[‘username’]; Also change the calling function for token to $token. Test it to check if the code was successful. Voila!

Next is we have to save the token but only the signature part. There will be 2 versions saved: one copy for client browser and one copy for database. When the user wants access to the API, we need to check if the token coming from the client browser matches the token saved from database. When it doesn’t match, we will not authorize the user to access the API. We’re going to create a variable named token\_array = explode(‘.’, $token). The purpose of explode is to separate the strings, since there are 3 functions that are returned and they are separated by ‘.’ We use explode to cut them because we’re only returning the signature encryption. Also, we modified the $payload so that it will only return the 3rd element which is the signature equivalent to index 2 in array.

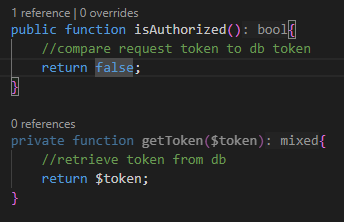
To save the returned value for signature we have to update the token field. To do that, we just have to go to patch.php then we’re going to copy the whole function archiveChefs. Rename the function to saveToken and pass $token and $username. Modify the sql command to UPDATE accounts\_tbl SET token=? WHERE username = ?. Then, we’re changing the execute value by passing the $token and $username. Then, go to login function and create the instance $this->saveToken($token\_array[2], $result[‘username’]).

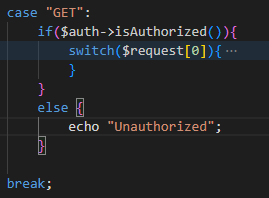


We will try to send the request to see if it’s successful. Refresh database and you can see it was changed.

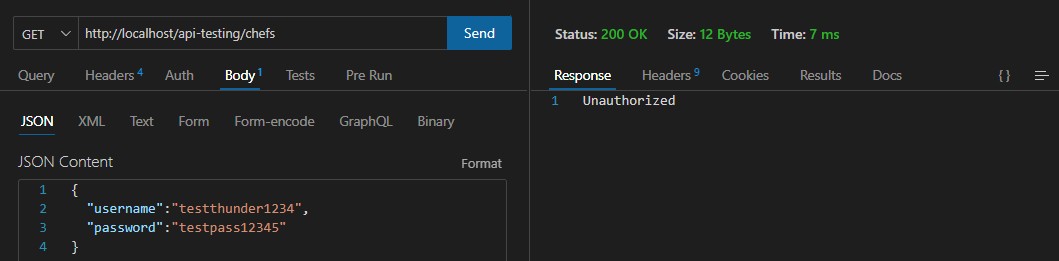
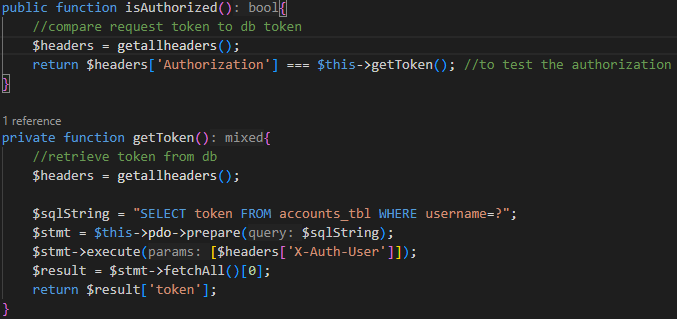
After the process of saving the token, we will now use the token. The token is use to restrict the access on certain functionalities in back-end or API. Example, in getChefs anyone can send request to retrieve records of the chefs\_table. We can restrict who can use/retrieve records in the database. Even insert records in database and more. That’s the purpose of token.

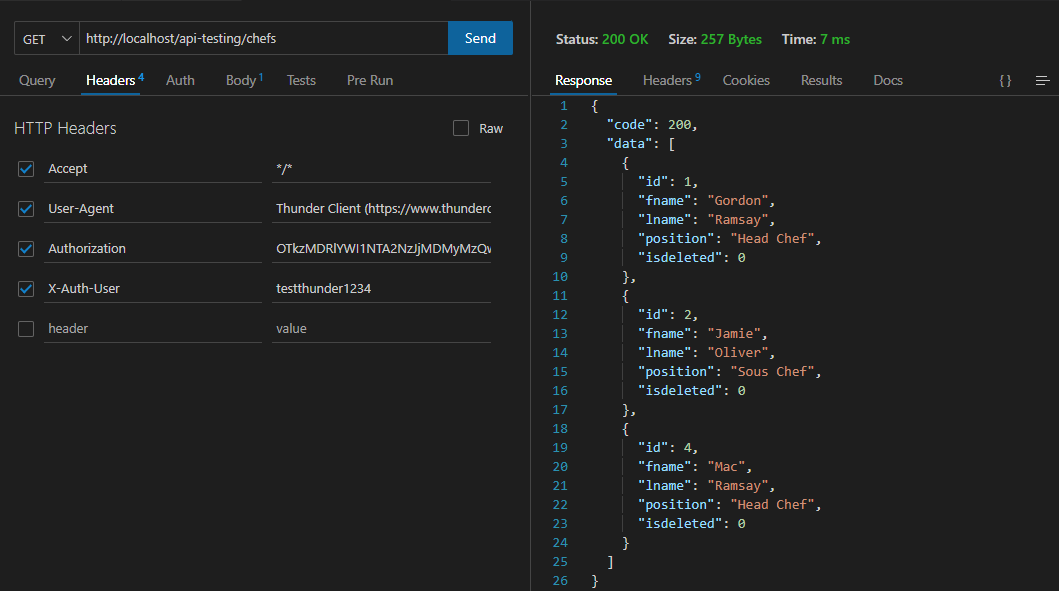
We will create 2 functions public function isAuthorized() and private function getToken($token). We will input return true (turn later to false to test the code) in isAuthorized and $token in getToken.



In order to apply authorization, we go to case “GET” in routes.php. We will see the whole switch command and were going to copy that. Then, we create an if condition($auth->isAuthorized) then we will paste the whole switch command inside and create an else condition that will return when the user is not authorized (can decide on what to put in else condition whether redirect or just simple text. To test that we will go to thunderclient.

Before we proceed, we will go to the headers section in thunder client and add a new header named Authorization, we will change the request method to Post and Login. Once log in is successful we will copy the token and that will be the value inside Authorization. Then, we will add new header named X-Auth-User and assign the username of the user.

We’re going to modify our created 2 functions isAuthorized and getToken. In isAuthorized() we will get all the headers from the HTTP Headers we modified in thunderclient. Then, we will return it $headers[‘Authorization’]. The token saved in headers (encrypted token) is stored in $headers[‘Authorization’]. Then, we will compare it with the instance $this->getToken(). In getToken function we can copy the code from the login function and modify the sql command to select token from accounts\_tbl where it gets the username of the user. Then, we also modify the execute where we pass the $headers[‘X-Auth-User’] which is equal to the input username. Last, we will return the $result[‘token’].

To test: