Part 3

Creating access token

Create new file: **JWTCodec.php** inside **src** folder.

Create a new **function encode(array $payload)** for generating the JWT token and **base64UrlEncode(string $text)**

JWTCodec.php

```php
<?php
class JWTCodec
{
    public function encode(array $payload): string
    {
        $header = json_encode([
            "typ" => "JWT",
            "alg" => "HS256"
        ]);

        $header = $this->base64urlEncode($header);

        $payload = json_encode($payload);
        $payload = $this->base64urlEncode($payload);

        $private_key =
"357538782F413F4428472B4B6150645367566B5970337336763979244226452 9";
        $signature = hash_hmac("sha256", $header . "." . $payload, $private_key, true);
        $signature = $this->base64urlEncode($signature);

        return $header . "." . $payload . "." . $signature;
    }


    private function base64urlEncode(string $text): string
    {
        return str_replace(
            ['+', "/", "="],
            ["-", "_", ""],
            base64_encode($text)
        );
    }

}
```

Modify **login() function** inside **UserGateway.php.** We need to return the generated access token when a user login their account and use the token in every API request.

UserGateway.php

```php
public function login(string $email, string $password)
{
    $sql = "SELECT * FROM users WHERE email = :email AND password = :password";
    $res = $this->conn->prepare($sql);
    $res->bindValue(":email", $email, PDO::PARAM_STR);
    $res->bindValue(":password", md5($password), PDO::PARAM_STR);

    $res->execute();
    $data = $res->fetch(PDO::FETCH_ASSOC);


    if ($data !== false) {

        $payload_response = array(
            "sub" => $data["id"],
            "email" => $data["email"],
            "firstName" => $data["firstName"],
            "middleName" => $data["middleName"],
            "lastName" => $data["lastName"],
            "contactNo" => $data["contactNo"]
        );
        $codec = new JWTCodec;
        $access_token = $codec->encode($payload_response);

        return ["access_token" => $access_token];
    }
}
```

After creating the encode, we need to create the decode function. Back to the **JWTCodec.php**, create a new function for **decode()** and **base64Decode().**

```php
public function decode(string $token): array
    {
        if (preg_match("/^(?<header>.+)\.(?<payload>.+)\.(?<signature>.+)$/", $token, $matches) !== 1) {
            throw new InvalidArgumentException("Invalid token format");

        }

        $private_key = "357538782F413F4428472B4B6150645367566B5970337336763979244226452 9";
        $signature = hash_hmac("sha256", $matches["header"] . "." . $matches["payload"], $private_key, true);
        $signature_from_token = $this->base64UrlDecode($matches["signature"]);

        if (!hash_equals($signature, $signature_from_token)) {
            throw new Exception("Signature doesn't match");
        }

        $payload = json_decode($this->base64UrlDecode($matches["payload"]), true);
        return $payload;
    }
```

```php
private function base64urlDecode(string $text): string
{
    return base64_decode(
        str_replace(
            ["-", "_"],
            ["+", "/"],
            $text
        )
    );
}
```

We need a way to decode and authenticate the access token. Create a new file inside **src/Auth.php** and add a new function **authenticateAccessToken()**

```php
<?php

class Auth
{
    private int $user_id;

    public function __construct(
        private UserGateway $user_gateway,
        private JWTCodec $codec
    )
    {
    }

    public function getUserID(): int
    {
        return $this->user_id;
    }

    public function authenticateAccessToken(): bool
    {
        if (!preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches))
{
            http_response_code(400);
            echo json_encode(["message" => "incomplete authorization header"]);
            return false;
        }

        try {
            $data = $this->codec->decode($matches[1]);

        } catch (Exception $e) {

            http_response_code(400);
            echo json_encode(["message" => $e->getMessage()]);
            return false;
        }

        $this->user_id = $data["sub"];

        return true;
    }
}
```

PHP doesn’t include the header for Authorization, we need to change the **.htaccess** and add the code below to include the Authorization in the header request.

.htaccess

```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

Create a new file inside **src** folder with file name **InvalidSignatureException.php**

```php
<?php
class InvalidSignatureException extends Exception
{

}
```

Modify the **JWTCodec.php, decode()** function.

```php
public function decode(string $token): array
{
    if (preg_match("/^(?<header>.+)\.(?<payload>.+)\.(?<signature>.+)$/", $token, $matches) !== 1) {
        throw new InvalidArgumentException("Invalid token format");

    }

    $private_key = "357538782F413F4428472B4B6150645367566B5970337336763979244226452 9";
    $signature = hash_hmac("sha256", $matches["header"] . "." . $matches["payload"], $private_key, true);
    $signature_from_token = $this->base64UrlDecode($matches["signature"]);

    if (!hash_equals($signature, $signature_from_token)) {
        ~~throw new Exception("Signature doesn't match");~~

        throw new InvalidSignatureException("Signature doesn't match");
    }

    $payload = json_decode($this->base64UrlDecode($matches["payload"]), true);
    return $payload;
}
```

Modify **index.php**

```php
<?php
declare(strict_types=1);
require __DIR__ . "/vendor/autoload.php";

set_error_handler("ErrorHandler::handleError");
set_exception_handler("ErrorHandler::handleException");

header("Content-type: application/json; charset=UTF-8");
$parts = explode("/", $_SERVER["REQUEST_URI"]);

$database = new Database("localhost", "ecommercedb", "root", "");
$database->getConnection();
$user_gateway = new UserGateway($database);

$codec = new JWTCodec;

$auth = new Auth($user_gateway, $codec);
if ($parts[2] !== 'user' && $_SERVER["REQUEST_METHOD"] === "POST") {

    if (!$auth->authenticateAccessToken()) {
        exit;
    }
}

switch ($parts[2]) {
    case 'products':
        $id = $parts[3] ?? null;

        $gateway = new ProductGateway($database);

        $controller = new ProductController($gateway, $auth);
        $controller->processRequest($_SERVER["REQUEST_METHOD"], $id);
        break;

    case 'reviews':
        $id = $parts[3] ?? null;
        $productid = $parts[5] ?? null;

        $gateway = new ReviewGateway($database);

        $controller = new ReviewController($gateway);
        $controller->processRequest($_SERVER["REQUEST_METHOD"], $id, $productid);
        break;

    case 'user':
        $action = $parts[3] ?? null;
        if ($action === null) {
            http_response_code(404);
        }

        $gateway = new UserGateway($database);

        $controller = new UserController($gateway);
        $controller->processRequest($_SERVER["REQUEST_METHOD"], $action);
        break;

    default:
        http_response_code(404);
        exit;
}
```

We need to relate the product that are created by user. I also added new columns on products table for price, image, userid. Modify **ProductGateway.php**

Create function

```
public function create(array $data): string
{
    $sql = "INSERT INTO products (name, size, price, is_available, image, userid)
            VALUES (:name, :size, :price, :is_available, :image, :userid)";
    $res = $this->conn->prepare($sql);
    $res->bindValue(":name", $data["name"], PDO::PARAM_STR);
    $res->bindValue(":size", $data["size"] ?? 0, PDO::PARAM_INT);
    $res->bindValue(":price", $data["price"], PDO::PARAM_STR);
    $res->bindValue(":is_available", (bool) $data["is_available"] ?? false,
PDO::PARAM_BOOL);
    $res->bindValue(":image", $data["image"], PDO::PARAM_STR);
    $res->bindValue(":userid", $data["userid"], PDO::PARAM_INT);

    $res->execute();
    return $this->conn->lastInsertId();
}
```

Update function

```
public function update(array $current, array $new): int
   {
       $sql = "UPDATE products SET name = :name, size = :size, is_available =
:is_available, price =:price, description = :description, image = :image WHERE id
=:id";
       $res = $this->conn->prepare($sql);
       $res->bindValue(":name", $new["name"] ?? $current["name"], PDO::PARAM_STR);
       $res->bindValue(":size", $new["size"] ?? $current["size"], PDO::PARAM_INT);
       $res->bindValue(":price", $new["price"] ?? $current["price"], PDO::PARAM_STR);
       $res->bindValue(":description", $new["description"] ?? $current["description"],
PDO::PARAM_STR);
       $res->bindValue(":is_available", $new["is_available"] ??
$current["is_available"], PDO::PARAM_BOOL);
       $res->bindValue(":image", $new["image"] ?? $current["image"], PDO::PARAM_STR);
       $res->bindValue(":id", $current["id"], PDO::PARAM_INT);

       $res->execute();

       return $res->rowCount();
   }
```

Included the **Auth** on __construct() of **ProductController.php**

```
<?php
class ProductController
{
    public function __construct(private ProductGateway $gateway, private Auth $auth)
    {
    }
```

For file uploading, modify the case “POST” under **processCollectionRequest** function

```
case "POST":
                // $data = (array) json_decode(file_get_contents("php://input"), true);
                $data = $_POST;
                $errors = $this->getValidationErrors(($data));
                if (!empty($_FILES['file']['name'])) {
                    $file_name = $_FILES['file']['name'];
                    $temp_path = $_FILES['file']['tmp_name'];
                    $file_size = $_FILES['file']['size'];
                    $temp = explode(".", $_FILES["file"]["name"]);
                    $new_file_name = round(microtime(true)) . '.' . end($temp);

                    $upload_path = "uploads/";
                    $file_ext = strtolower(pathinfo($file_name, PATHINFO_EXTENSION));

                    $valid_extensions = array("jpeg", "jpg", "png", "gif");
                    if (in_array($file_ext, $valid_extensions)) {
                        if (!file_exists($upload_path . $new_file_name)) {
                            if ($file_size < 5000000 && empty($errors)) {
                                $data['image'] = $upload_path . $new_file_name;
                                move_uploaded_file($temp_path, $upload_path . $new_file_name);
                            } else {
                                $errors[] = "File size is too large, maximum file size is 5Mb";
                            }
                        } else {
                            $errors[] = "file already exists in upload folder";
                        }
                    } else {
                        $errors[] = "Invalid file format";
                           }
                        } else {
                            if (empty($file_name)) {
                                $errors[] = "Image is required";
                            }
                        }

                        if (!empty($errors)) {
                            http_response_code(422);
                            echo json_encode(["errors" => $errors]);
                            break;
                        }
                        $data['userid'] = $this->auth->getUserID();
                        $id = $this->gateway->create($data);

                        http_response_code(201);
                        echo json_encode([
                            "message" => "Product created",
                            "id" => $id
                        ]);
                        break;
```

Add the price on **getValidationErrors**

```
    private function getValidationErrors(array $data, bool $is_new = true): array
    {
        $errors = [];
        if ($is_new && empty($data["name"])) {
            $errors[] = "name is required";
        }

        if (array_key_exists("size", $data)) {
            if (filter_var($data["size"], FILTER_VALIDATE_INT) === false) {
                $errors[] = "size must be an integer";
            }
        }

        if ($is_new && empty($data["price"])) {
            $errors[] = "price is required";
        }

        return $errors;
    }
```