PHP REST API Part 2

Code improvements and Implementation of Reviews

.htaccess

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule . index.php [L]
```

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"]);

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

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

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

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

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

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

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

ProductGateway.php (inside src folder)

```php
<?php

class ProductGateway
{
    private PDO $conn;
    public function __construct(Database $database)
    {
        $this->conn = $database->getConnection();
    }

    public function getAll(): array
    {
        $sql = "SELECT * FROM products";
        $res = $this->conn->query($sql);
        $data = [];

        while ($row = $res->fetch(PDO::FETCH_ASSOC)) {
            $row["is_available"] = (bool) $row["is_available"];
            $data[] = $row;
        }

        return $data;
    }

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

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

    public function get(string $id)
    {
        $sql = "SELECT * FROM products WHERE id = :id";
        $res = $this->conn->prepare($sql);
        $res->bindValue(":id", $id, PDO::PARAM_INT);
        $res->execute();
        $data = $res->fetch(PDO::FETCH_ASSOC);

        if ($data !== false) {
            $data["is_available"] = (bool) $data["is_available"];

            $sqlReviews = "SELECT * FROM reviews where productid = :productid LIMIT 10";
            $resReviews = $this->conn->prepare($sqlReviews);
            $resReviews->bindValue(":productid", $id, PDO::PARAM_INT);
            $resReviews->execute();
            $dataReviews = $resReviews->fetchAll(PDO::FETCH_ASSOC);
            if ($dataReviews !== false) {
                $data['reviews'] = $dataReviews;
            }
```

```php
        }

        return $data;
    }

    public function update(array $current, array $new): int
    {
        $sql = "UPDATE products SET name = :name, size = :size, is_available = :is_available 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(":is_available", $new["is_available"] ?? $current["is_available"], PDO::PARAM_BOOL);
        $res->bindValue(":id", $current["id"], PDO::PARAM_INT);

        $res->execute();

        return $res->rowCount();
    }

    public function delete(string $id): int
    {
        $sql = "DELETE FROM products WHERE id = :id";
        $res = $this->conn->prepare($sql);
        $res->bindValue(":id", $id, PDO::PARAM_INT);
        $res->execute();

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

ProductController.php (inside src folder)

```php
<?php
class ProductController
{
    public function __construct(private ProductGateway $gateway)
    {
    }
    public function processRequest(string $method, ?string $id): void
    {
        if ($id) {
            $this->processResourcetRequest($method, $id);
        } else {
            $this->processCollectionRequest($method);

        }
    }


    private function processResourcetRequest(string $method, string $id): void
    {
        $product = $this->gateway->get($id);
        if (!$product) {
            http_response_code(404);
            echo json_encode(["message" => "Product not found"]);
            return;
        }

```

```
27.         switch ($method) {
28.             case "GET":
29.                 echo json_encode($product);
30.                 break;
31.
32.             case "PATCH":
33.                 $data = (array) json_decode(file_get_contents("php://input"), true);
34.                 $errors = $this->getValidationErrors($data, false);
35.
36.                 if (!empty($errors)) {
37.                     http_response_code(422);
38.                     echo json_encode(["errors" => $errors]);
39.                     break;
40.                 }
41.
42.                 $rows = $this->gateway->update($product, $data);
43.
44.                 echo json_encode([
45.                     "message" => "Product $id updated",
46.                     "rows" => $rows
47.                 ]);
48.                 break;
49.
50.             case "DELETE":
51.                 $rows = $this->gateway->delete($id);
52.                 echo json_encode([
53.                     "message" => "Product $id deleted",
54.                     "rows" => $rows
55.                 ]);
56.                 break;
57.
58.             default:
59.                 http_response_code(405);
60.                 header("Allow: GET, PATCH, DELETE");
61.
62.         }
63.     }
64.
65.     private function processCollectionRequest(string $method): void
66.     {
67.         switch ($method) {
68.             case "GET":
69.                 echo json_encode($this->gateway->getAll());
70.                 break;
71.
72.             case "POST":
73.                 $data = (array) json_decode(file_get_contents("php://input"), true);
74.                 $errors = $this->getValidationErrors(($data));
75.
76.                 if (!empty($errors)) {
77.                     http_response_code(422);
78.                     echo json_encode(["errors" => $errors]);
79.                     break;
80.                 }
81.
82.                 $id = $this->gateway->create($data);
83.
84.                 http_response_code(201);
85.                 echo json_encode([
86.                     "message" => "Product created",
87.                     "id" => $id
```

```
                ]);
                break;

            default:
                http_response_code(405);
                header("Allow: GET, POST");
        }
    }

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

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

            return $errors;
        }
    }
```

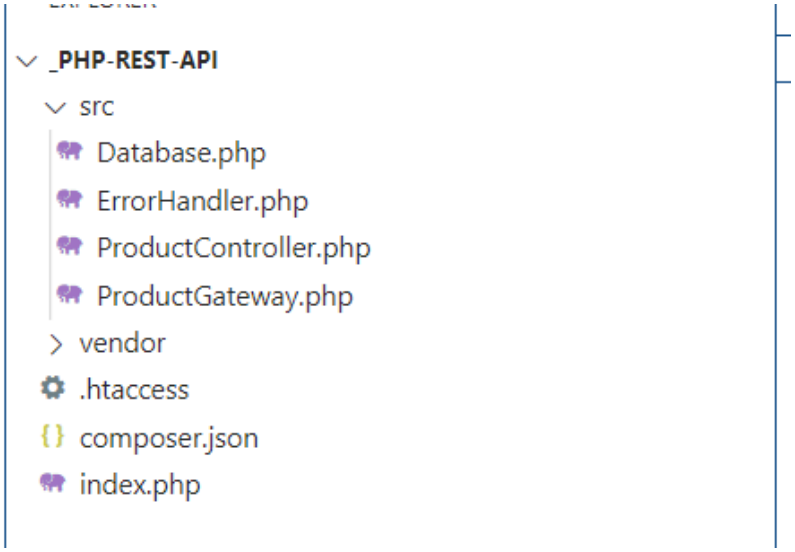
Download and install Composer on your computer, **https://getcomposer.org/download/
*url for windows setup.exe **(https://getcomposer.org/Composer-Setup.exe)**

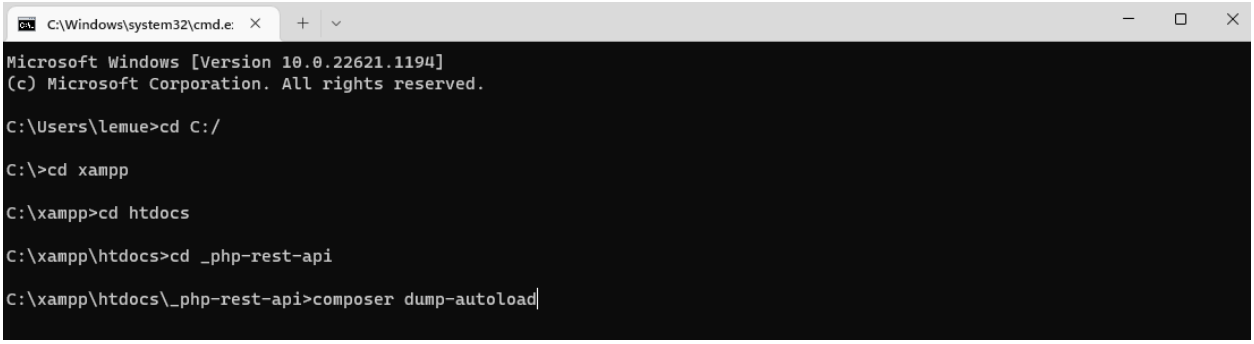
I also added a new file named **composer.json** for autoloading of files with the same class names.

After creating the composer.json with the code below, run this on command prompt inside your project folder to generate the vendor folder.

composer dump-autoload

```
{
    "autoload": {
        "psr-4": {
            "": "src/"
        }
    }
}
```





In my phpMyAdmin, create a new table named **reviews,** set the id index = primary, auto increment(A_I) to true.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
☐	1	**id**	int(11)			No	*None*		AUTO_INCREMENT	Change	Drop	More
☐	2	**productid**	int(11)			No	*None*			Change	Drop	More
☐	3	**name**	varchar(128)	utf8mb4_general_ci		No	*None*			Change	Drop	More
☐	4	**content**	text	utf8mb4_general_ci		No	*None*			Change	Drop	More
☐	5	**rate**	enum('1', '2', '3', '4', '5', '6', '7', '8', '9', ...	utf8mb4_general_ci		No	*None*			Change	Drop	More
☐	6	**datetime_created**	datetime			No	current_timestamp()			Change	Drop	More

User Authentication

Create a new table: **users**, set the id index as primary and auto increment(A_I) to true. Next, we need to have a unique email, set the index of email as primary.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	**id**	int(11)			No	*None*		AUTO_INCREMENT
☐	2	**email**	varchar(128)	utf8mb4_general_ci		No	*None*		
☐	3	**password**	varchar(60)	utf8mb4_general_ci		No	*None*		
☐	4	**firstName**	varchar(128)	utf8mb4_general_ci		No	*None*		
☐	5	**middleName**	varchar(128)	utf8mb4_general_ci		No	*None*		
☐	6	**lastName**	varchar(128)	utf8mb4_general_ci		No	*None*		
☐	7	**contactNo**	varchar(20)	utf8mb4_general_ci		No	*None*		

For user authentication, create a new file inside **src folder** and named it as **UserGateway.php.**

Set the $database inside public function __construct() so that we can pass the database to the UserGateway. Then create new function for **register** and **login.**

```php
<?php

class UserGateway
{
    private PDO $conn;
    public function __construct(Database $database)
    {
        $this->conn = $database->getConnection();
    }

    public function register(array $data): string
    {
        $sql = "INSERT INTO users (email, password, firstName, middleName, lastName, contactNo)
                VALUES (:email, :password, :firstName, :middleName, :lastName, :contactNo)";
        $res = $this->conn->prepare($sql);

        $password = md5($data["password"]);

        $res->bindValue(":email", $data["email"], PDO::PARAM_STR);
        $res->bindValue(":password", $password, PDO::PARAM_STR);
        $res->bindValue(":firstName", $data["firstName"], PDO::PARAM_STR);
        $res->bindValue(":middleName", $data["middleName"], PDO::PARAM_STR);
        $res->bindValue(":lastName", $data["lastName"], PDO::PARAM_STR);
        $res->bindValue(":contactNo", $data["contactNo"], PDO::PARAM_STR);
```

```php

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

    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 = $res->fetch(PDO::FETCH_ASSOC);
        unset($data['password']);
        return $data;
    }

    public function changePassword(array $current, array $new): int
    {
        $sql = "UPDATE users SET password = :password";
        $res = $this->conn->prepare($sql);
        $res->bindValue(":password", md5($new["password"]) ?? md5($current["password"]), PDO::PARAM_STR);

        $res->execute();

        return $res->rowCount();
    }

}
```

After creating the UserGate, create a new file inside **src** folder for **UserController.php**

```php
<?php
class UserController
{
    public function __construct(private UserGateway $gateway)
    {
    }

    public function processRequest(string $method, string $action): void
    {
        if ($method === 'POST') {
            switch ($action) {
                case "login":
                    $this->processLoginRequest();
                    break;

                case "register":
                    $this->processRegistrationRequest();
                    break;
            }
        } else {
            http_response_code(405);
            header("Allow: POST");
```

```
23.         }
24.     }
25.
26.     private function processLoginRequest(): void
27.     {
28.
29.         $data = (array) json_decode(file_get_contents("php://input"), true);
30.         $errors = $this->getLogInValidationErrors(($data));
31.
32.         if (!empty($errors)) {
33.             http_response_code(422);
34.             echo json_encode(["errors" => $errors]);
35.         }
36.
37.         $user = $this->gateway->login($data['email'], $data['password']);
38.
39.         if (!$user) {
40.             http_response_code(404);
41.             echo json_encode(["message" => "Incorrect email/password"]);
42.             return;
43.         }
44.
45.         echo json_encode($user);
46.
47.     }
48.
49.     private function processRegistrationRequest(): void
50.     {
51.
52.         $data = (array) json_decode(file_get_contents("php://input"), true);
53.         $errors = $this->getRegistrationValidationErrors(($data));
54.
55.         if (!empty($errors)) {
56.             http_response_code(422);
57.             echo json_encode(["errors" => $errors]);
58.         }
59.
60.         $id = $this->gateway->register($data);
61.
62.         http_response_code(201);
63.         echo json_encode([
64.             "message" => "User created",
65.             "id" => $id
66.         ]);
67.
68.     }
69.
70.     private function getLogInValidationErrors(array $data): array
71.     {
72.         $errors = [];
73.         if (empty($data["email"])) {
74.             $errors[] = "Email is requred";
75.         }
76.
77.         if (empty($data["password"])) {
78.             $errors[] = "Password is requred";
79.         }
80.
81.         return $errors;
82.     }
83.
```

```
    private function getRegistrationValidationErrors(array $data): array
    {
        $errors = [];
        if (empty($data["email"])) {
            $errors[] = "Email is requred";
        }

        if (empty($data["password"])) {
            $errors[] = "Password is requred";
        }

        if (empty($data["firstName"])) {
            $errors[] = "First name is requred";
        }

        if (empty($data["lastName"])) {
                $errors[] = "Last name is requred";
            }

            return $errors;
        }
    }
```

To use the UserController, modify the **index.php.** Add a new case inside the switch

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

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

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