

**Jefri Maruli**

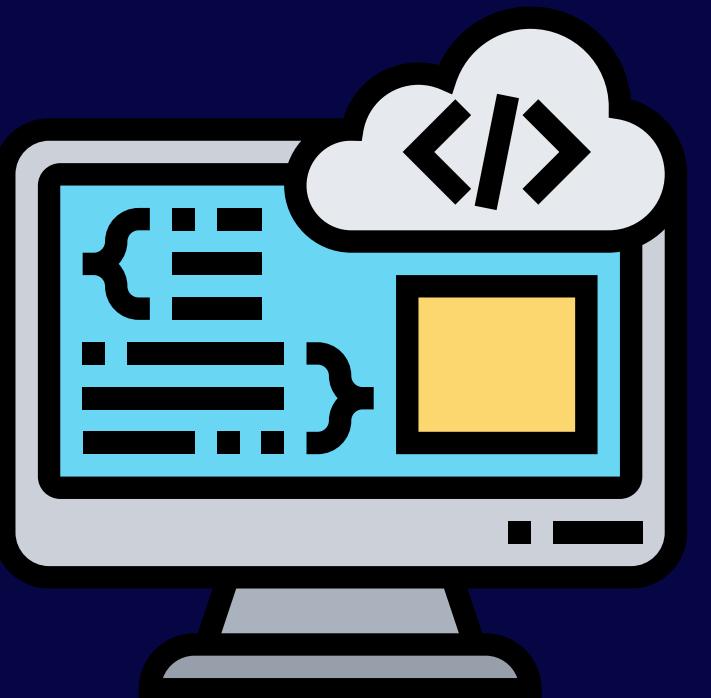
# Back-End Development

## Materi 6



# Agenda Belajar

- JWT Authentication
- MC Pattern (Model & Controller)



# JWT Authentication

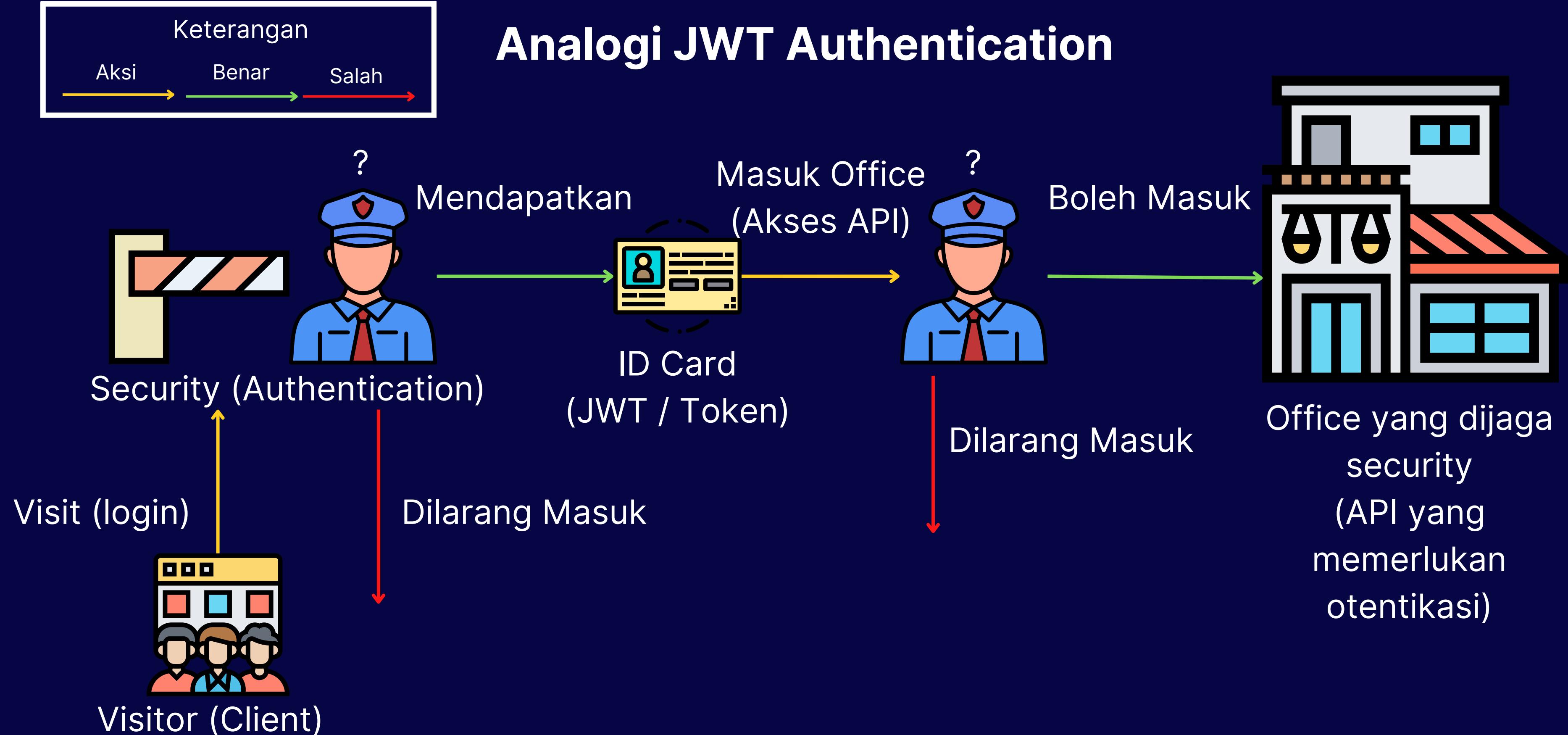
JWT atau disebut JSON Web Token adalah token yang menggunakan JSON (JavaScript Object Notation) berbentuk string random yang panjang, dan biasanya digunakan untuk aktivitas transfer informasi antar platform. JWT dapat berfungsi untuk sistem otentikasi dan juga untuk pertukaran informasi.

situs resmi : <https://jwt.io>

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG91IiwiaWF0IjoxNTE2MjM5MDIyfQ.SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

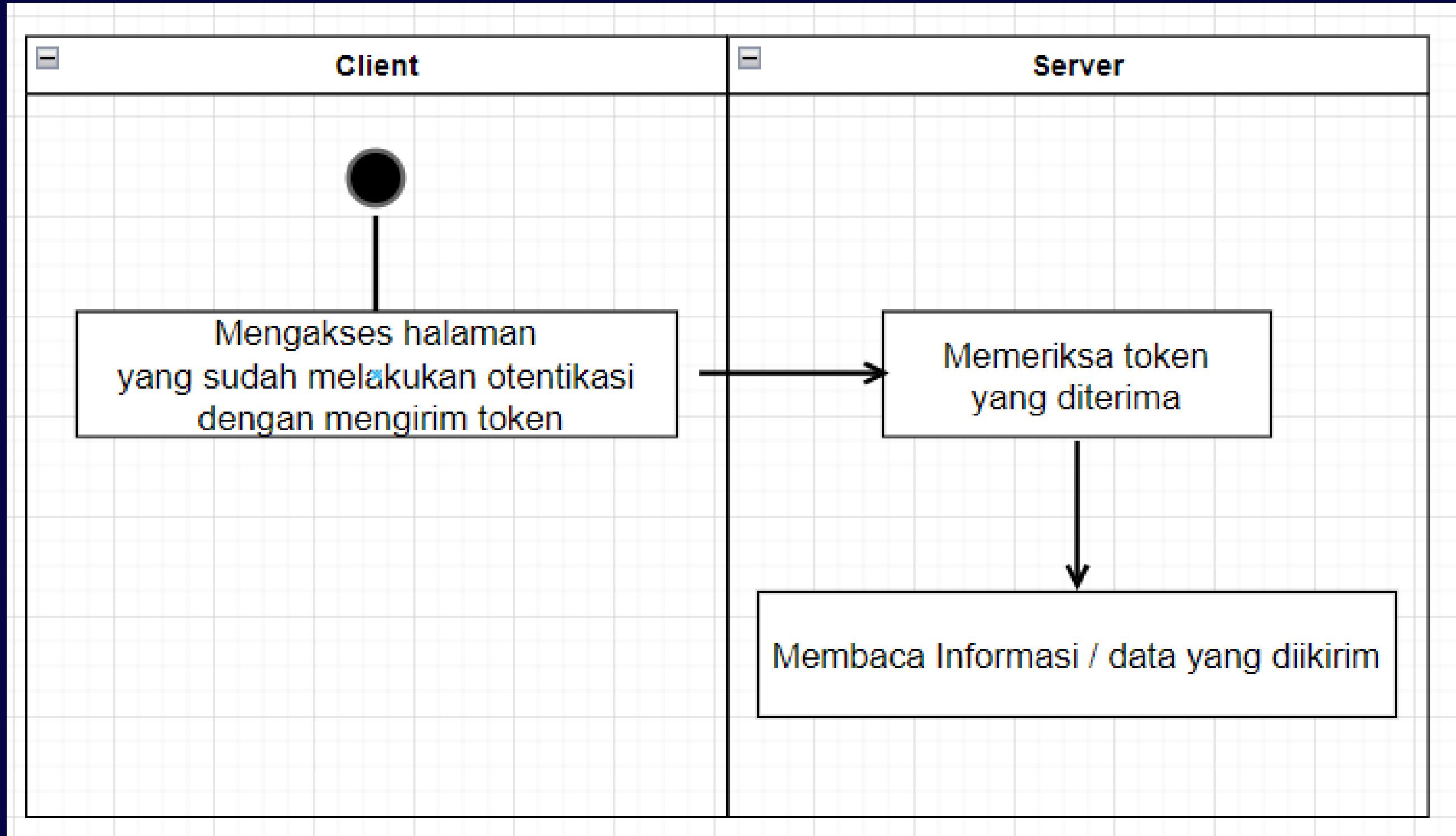


# Analogi JWT Authentication



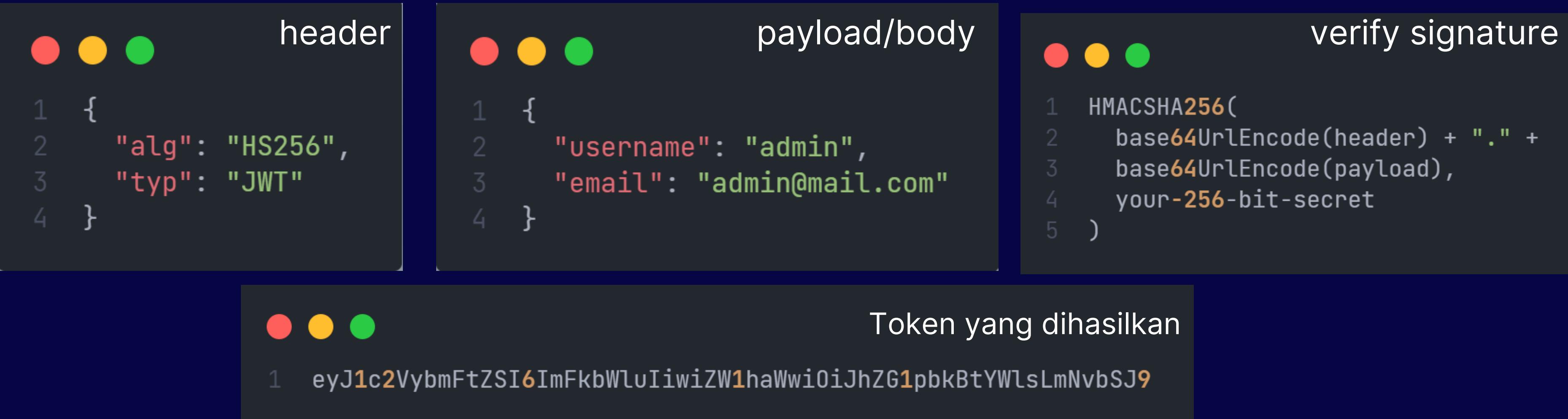
Setelah melakukan login server akan melakukan generate token (JWT), ketika ingin mengakses API yang memerlukan login, maka client harus mengirimkan token yang sudah digenerate oleh server, jika token tersebut valid maka client bisa mengakses API tersebut dan sebaliknya.

# Cara Kerja JWT



Server akan melakukan generate token yang mensertifikasi identitas user, dan mengirimkannya ke client. Client akan mengirim token kembali ke server untuk setiap request ke API, sehingga Server side akan bisa mengetahui asal dari request tersebut.

# Algoritma JWT



Algoritma untuk membuat Token khususnya JWT ada 3 komponen yaitu header sebagai jenis HASH yang akan digunakan seperti HS256 (HMAC SHA256), kemudian payload / body adalah sebagai data yang akan di enkripsi (jangan memasukkan data yang bersifat sensitif seperti password), dan verify signature yaitu kombinasi untuk membuat sebuah Token. Signature tidak mungkin dapat mendekripsi (decrypt), karena sudah dalam berbentuk hash atau enkripsi satu arah (tidak dapat dikembalikan ke nilai semula).

# Persiapan Implementasi JWT

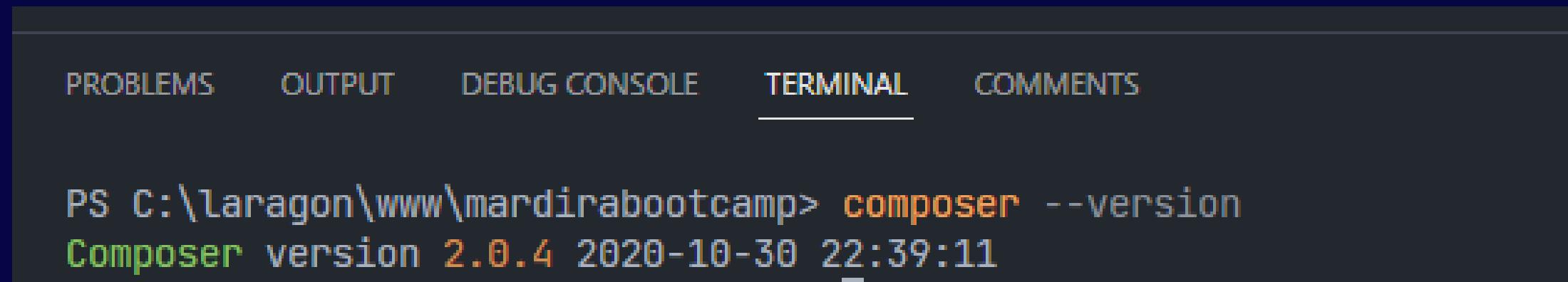
Tools yang dibutuhkan untuk implementasi JWT adalah menggunakan composer, composer sebagai dependency manager membuat pengembangan project jadi lebih mudah. Fitur-fitur yang dibutuhkan untuk sebuah proyek akan memakan waktu lama bila semua dibuat sendiri dari awal. Solusinya gunakan library yang sudah dibuat orang lain dengan bantuan composer.



Composer

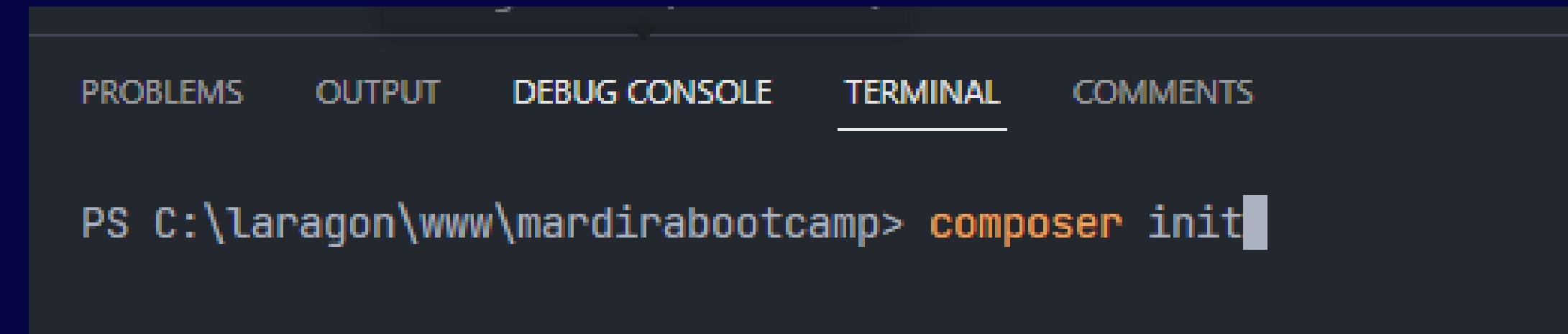
Download : <https://getcomposer.org/download/>

# Setup Composer Init



A screenshot of a terminal window with a dark background and light-colored text. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and COMMENTS. The terminal output shows the command 'composer --version' being run in a directory 'C:\Laragon\www\mardirabootcamp'. The output displays 'Composer version 2.0.4 2020-10-30 22:39:11'.

```
PS C:\Laragon\www\mardirabootcamp> composer --version
Composer version 2.0.4 2020-10-30 22:39:11
```



A screenshot of a terminal window with a dark background and light-colored text. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and COMMENTS. The terminal output shows the command 'composer init' being run in a directory 'C:\Laragon\www\mardirabootcamp'.

```
PS C:\Laragon\www\mardirabootcamp> composer init
```

Setelah install componernya, kemudian buka workspace project menggunakan text editor masing-masing, setelah itu pada bagian terminal jalankan perintah composer --version untuk melihat versi composer yang di install dan memastikan composer sudah terinstall dengan benar, lalu jalankan perintah composer init untuk melakukan setup composer.

# Konfigurasi Composer

```
PS C:\laragon\www\mardirabootcamp> composer init

Welcome to the Composer config generator

This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [dell/mardirabootcamp]: bootcamp/jwt
Description []: Membuat otentikasi dengan JWT
Author [                gmail.com], n to skip]:
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []:
License []:
```

Setelah menjalankan init pada composer, langkah selanjutnya ialah mengisi Package name, Description, Author, untuk yang lainnya boleh dikosongkan saja dengan cara menekan tombol enter.

# Konfigurasi Selanjutnya

```
Define your dependencies.
```

```
Would you like to define your dependencies (require) interactively [yes]? no
```

```
Would you like to define your dev dependencies (require-dev) interactively [yes]? no
```

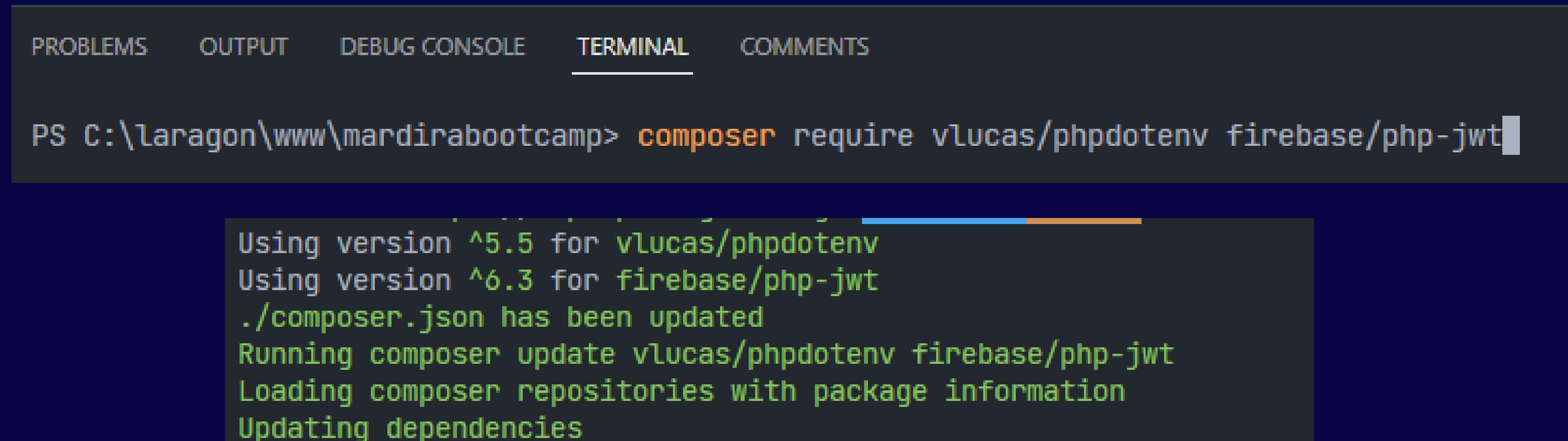
```
{
  "name": "bootcamp/jwt",
  "description": "Membuat otentikasi dengan JWT",
  "authors": [
    {
      "name": "        ",
      "email": "        .com"
    }
  ],
  "require": {}
}
```

```
Do you confirm generation [yes]? _
```

{--> **composer.json**

Konfigurasi selanjutnya untuk bagian dependencies ketik perintah no untuk kedua konfigurasi dependencies, setelah itu bagian terakhir tekan enter untuk menyelesaikan konfigurasi composer yang sudah dibuat dan akan membuat file composer.json.

# Instalasi Library



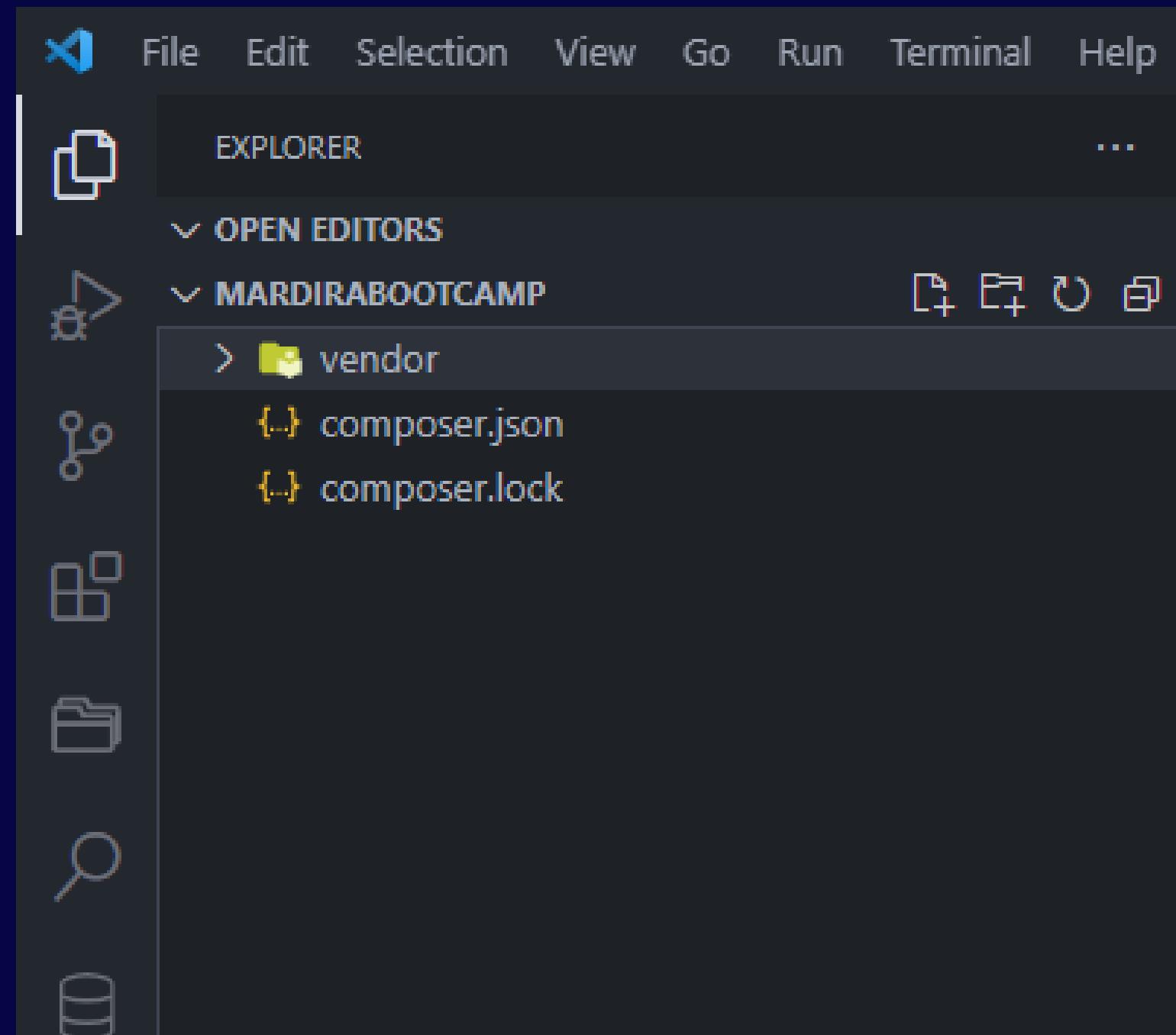
A screenshot of a terminal window titled "TERMINAL". The window shows the command "composer require vlucas/phpdotenv firebase/php-jwt" being run in a directory "C:\Laragon\www\mardirabootcamp". The output of the command is displayed below, showing the installation of dependencies:

```
PS C:\Laragon\www\mardirabootcamp> composer require vlucas/phpdotenv firebase/php-jwt
Using version ^5.5 for vlucas/phpdotenv
Using version ^6.3 for firebase/php-jwt
./composer.json has been updated
Running composer update vlucas/phpdotenv firebase/php-jwt
Loading composer repositories with package information
Updating dependencies
```

Setelah konfigurasi composer selesai, kemudian ketik perintah composer require dan nama library karena kita memerlukan library dotenv dan JWT untuk menggunakan otentikasi berbasis JWT. Alasan menggunakan dotenv untuk keamanan referensi ada di tautan bawah ini:

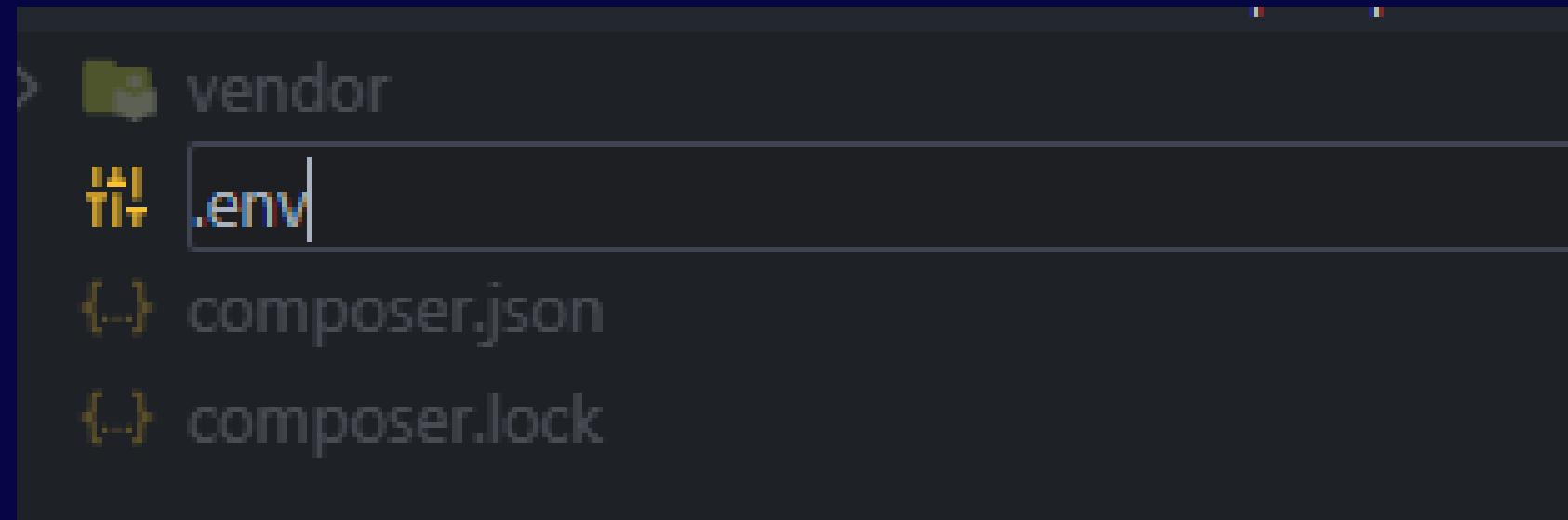
Belajar Nodejs #10: Gunakan Environment Variable untuk Mengamankan Aplikasimu ([petanikode.com](http://petanikode.com))

# Hasil Instalasi Library



Hasil setelah instalasi library berada di folder vendor

# Membuat .env File



A screenshot of a terminal window. At the top, there are three colored circular progress indicators: red, yellow, and green. Below them, the terminal output shows two lines of code:

```
1 ACCESS_TOKEN_SECRET_KEY=
2 REFRESH_TOKEN_SECRET_KEY=
```

Buat file dengan nama .env kemudian tambahkan 2 baris kode untuk ACCESS\_TOKEN\_SECRET\_KEY dan REFRESH\_TOKEN\_SECRET\_KEY

# Value .env File

Tidak disarankan



```
1 ACCESS_TOKEN_SECRET_KEY=mardira-bootcamp-access-token  
2 REFRESH_TOKEN_SECRET_KEY=mardira-bootcamp-refresh-token
```

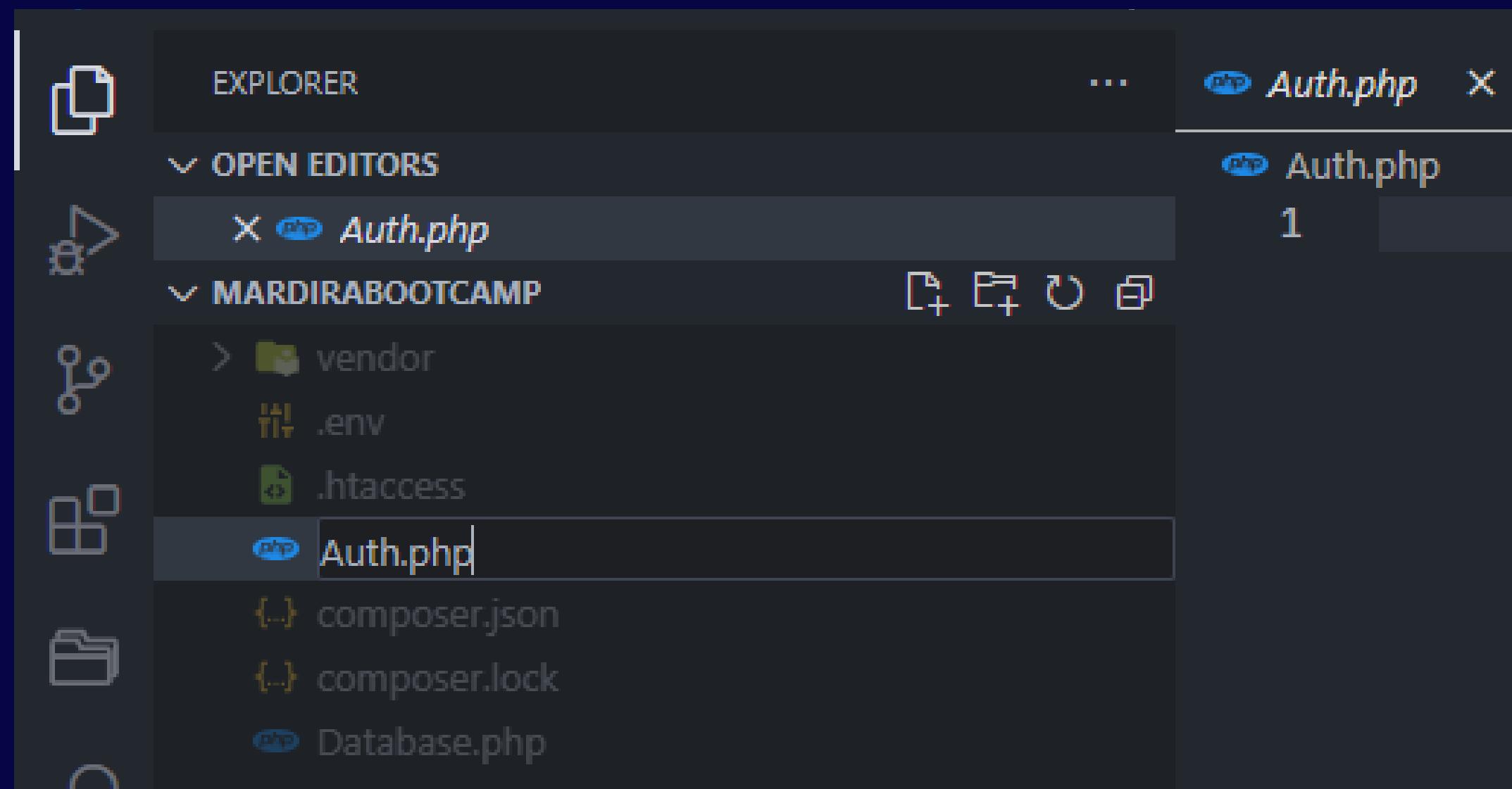
Rekomendasi



```
1 ACCESS_TOKEN_SECRET_KEY=xsffz9c53gm5t54xvgbuvxk6rd9j9hv  
2 REFRESH_TOKEN_SECRET_KEY=uv2a7fqpre2drs7wzkf42d4swshm6frt
```

Untuk kedua secret key value bisa diatur sesuai keinginan, tetapi kedua secret key ini bersifat rahasia, dan kedua valuenya tidak boleh sama. Alangkah lebih baik menggunakan secret key yang rumit, untuk generate password klik tautan di bawah ini.

# Implementasi JWT dengan Database



Buatlah file dengan nama Auth.php, untuk koneksi database masih menggunakan studi kasus materi sebelumnya.

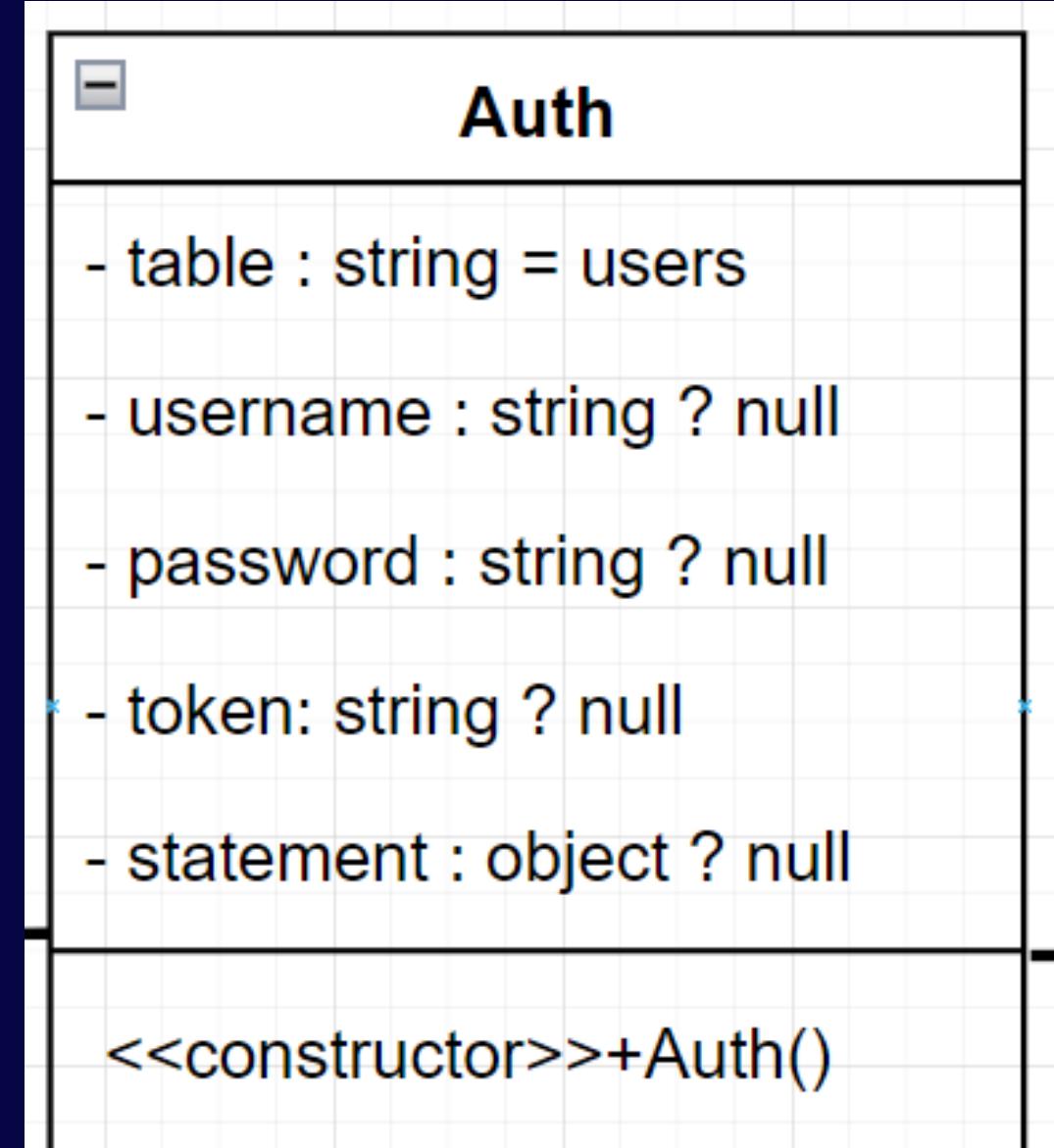
# Menggunakan Library

```
● ● ●  
1 <?php  
2  
3 // import script  
4 require_once('./Database.php');  
5 require_once('./vendor/autoload.php');  
6  
7 // menggunakan library JWT dan DotENV  
8 use Firebase\JWT\JWT;  
9 use Dotenv\Dotenv;  
10  
11 // menggunakan header untuk mengakses API  
12 header('Content-Type: application/json');  
13 header('Access-Control-Allow-Origin: *');  
14 header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');  
15 header('Access-Control-Allow-Headers: X-Requested-With');  
16 header('Access-Control-Allow-Headers: Content-Type, Authorization');
```

Untuk menggunakan library JWT dan Dotenv, kita menggunakan kata kunci use setelah load script autoload.php pada composer.

# Deklarasi Property dan Constructor Class Auth

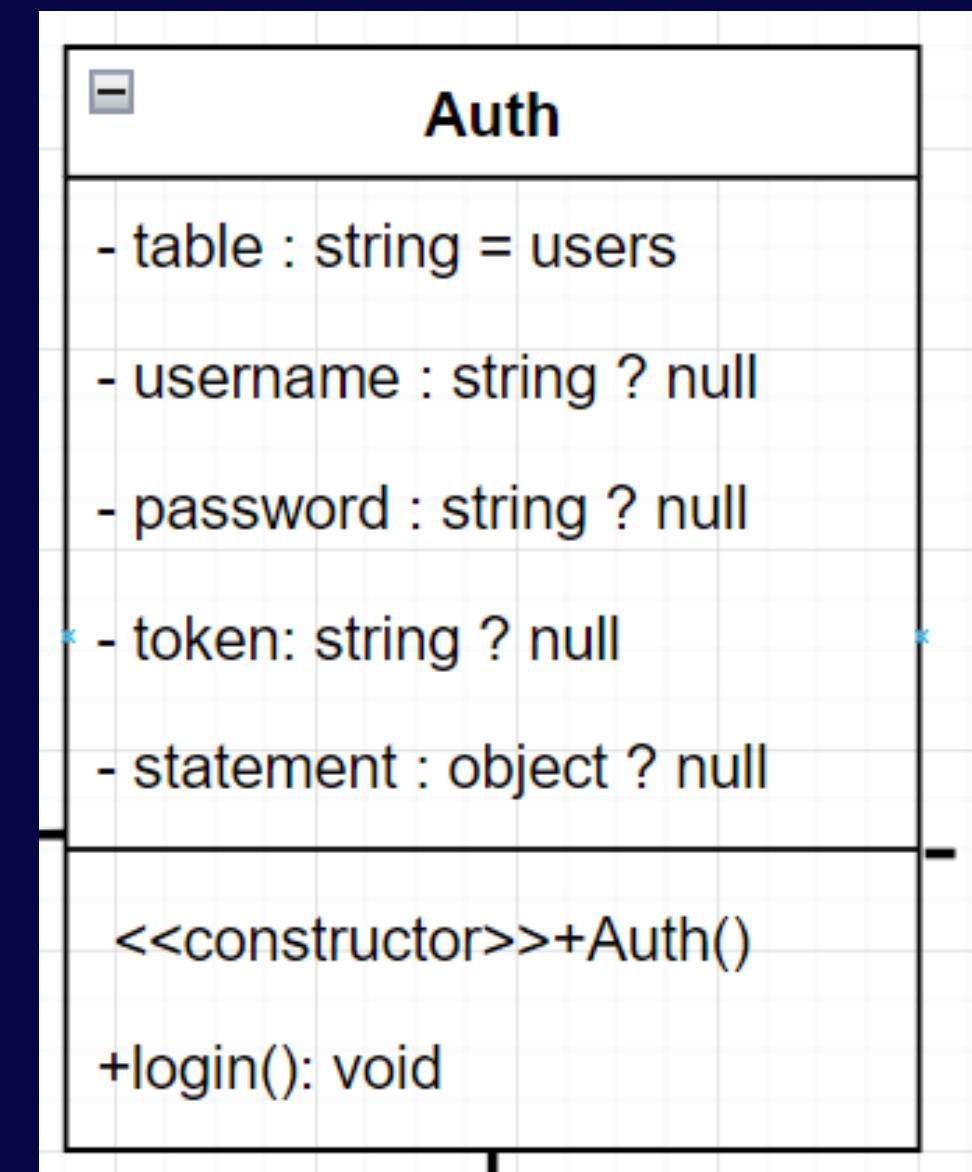
```
1 class Auth
2 {
3     private string $table = 'users';
4     private ?string $username;
5     private ?string $password;
6     private ?string $token;
7     private ?object $statement;
8
9     public function __construct()
10    {
11         // membuat instance dari class Database
12         $this->statement = new Database();
13         $this->statement = $this->statement->connection;
14         $this->username = $_POST['username'] ?? null;
15         $this->password = $_POST['password'] ?? null;
16         $this->token = getallheaders()['Authorization'] ?? null;
17         // membuat instance dari class DotENV
18         $dotenv = Dotenv::createImmutable(__DIR__);
19         $dotenv->load();
20    }
21 }
```



Kemudian pada class Auth tambahkan property, dan constructor seperti gambar di atas.

# Membuat Method Login

```
● ● ●  
1 public function login(): void  
2 {  
3     // membuat query untuk mengecek username dan password  
4     $query = "SELECT * FROM {$this→table} WHERE username = :username AND password = :password";  
5     $statement = $this→statement→prepare($query);  
6     // binding data  
7     $password = md5($this→password);  
8     $statement→bindParam(':username', $this→username);  
9     $statement→bindParam(':password', $password);  
10    // eksekusi query  
11    $statement→execute();  
12 }
```



Setelah membuat property dan constructor, kemudian membuat method login.

# Statement Row Count

```
● ● ●  
1 public function login(): void  
2 {  
3     // membuat query untuk mengecek username dan password  
4     $query = "SELECT * FROM {$this→table} WHERE username = :username AND password = :password";  
5     $statement = $this→statement→prepare($query);  
6     // binding data  
7     $password = md5($this→password);  
8     $statement→bindParam(':username', $this→username);  
9     $statement→bindParam(':password', $password);  
10    // eksekusi query  
11    $statement→execute();  
12    // jika data ditemukan  
13    if ($statement→rowCount() > 0) {  
14        // blok diatas  
15    } else {  
16        // blok dibawah  
17    }  
18 }
```

Statement row count fungsinya untuk menghitung jumlah data, jika jumlah data ditemukan lebih besar dari 0 atau 1 ditemukan maka akan eksekusi block diatas, dan sebaliknya.

# Baris Kode Setiap Block Statement

```
● ● ●

1 if ($statement->rowCount() > 0) {
2     $user = $statement->fetch(PDO::FETCH_OBJ);
3     // expired time 15 * 60 (detik) = 15 menit
4     $expired_time = time() + (15 * 60);
5     // membuat token
6     $payload = [
7         'exp' => $expired_time,
8         'data' => [
9             'id' => $user->user_id,
10            'username' => $user->username,
11        ]
12    ];
13    // membuat token dengan library JWT
14    $jwt = JWT::encode($payload, $_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256');
15    // membuat response
16    http_response_code(200);
17    $response = [
18        'message' => 'Login Berhasil',
19        'token' => $jwt,
20        'expired_time' => date('d-m-Y H:i:s', $expired_time),
21    ];
22    echo json_encode($response, JSON_PRETTY_PRINT);
23 } else {
24     // membuat response
25     $response = [
26         'message' => 'Login Gagal'
27     ];
28     http_response_code(401);
29     echo json_encode($response, JSON_PRETTY_PRINT);
30 }
```

Setelah menentukan control statement untuk membuat token, jika baris data ditemukan maka akan membuat token dengan, menentukan expired time setiap token yang akan di generate.

# Expiry Time JWT



```
1 // expired time 15 * 60 (detik) = 15 menit  
2 $expired_time = time() + (15 * 60);
```

Expiry time JWT atau batas kadaluwarsa token bisa diatur sesuai dengan keinginan misal 30, 60, atau 90 menit, dengan rumus `time()` (waktu sekarang) + `(15 * 60)` detik hasilnya sama dengan 15 menit.

# Membuat JWT

```
● ● ●  
1 $payload = [  
2     'exp' => $expired_time,  
3     'data' => [  
4         'id' => $user->user_id,  
5         'username' => $user->username,  
6     ]  
7 ];  
8 // membuat token dengan library JWT  
9 $jwt = JWT::encode($payload, $_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256');
```

Untuk membuat JWT yaitu menggunakan Class Library JWT dan menggunakan method static encode dengan parameter payload berisi array data yang akan di konversi, parameter kedua adalah secret key dari .env dan yang terakhir adalah jenis hash yang akan digunakan.

# Method Login Secara Keseluruhan

```
● ● ●

1 public function login(): void
2 {
3     // membuat query untuk mengecek username dan password
4     $query = "SELECT * FROM {$this→table} WHERE username = :username AND password = :password";
5     $statement = $this→statement→prepare($query);
6     // binding data
7     $password = md5($this→password);
8     $statement→bindParam(':username', $this→username);
9     $statement→bindParam(':password', $password);
10    // eksekusi query
11    $statement→execute();
12    // jika data ditemukan
13    if ($statement→rowCount() > 0) {
14        $user = $statement→fetch(PDO::FETCH_OBJ);
15        // expired time 15 * 60 (detik) = 15 menit
16        $expired_time = time() + (15 * 60);
17        // membuat token
18        $payload = [
19            'exp' => $expired_time,
20            'data' => [
21                'id' => $user→user_id,
22                'username' => $user→username,
23            ]
24        ];
25        // membuat token dengan library JWT
26        $jwt = JWT::encode($payload, $_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256');
27        // membuat response
28        http_response_code(200);
29        $response = [
30            'message' => 'Login Berhasil',
31            'token' => $jwt,
32            'expired_time' => date('d-m-Y H:i:s', $expired_time),
33        ];
34        echo json_encode($response, JSON_PRETTY_PRINT);
35    } else {
36        // membuat response
37        $response = [
38            'message' => 'Login Gagal'
39        ];
40        http_response_code(401);
41        echo json_encode($response, JSON_PRETTY_PRINT);
42    }
43 }
```

# Membuat Object Class Auth

```
● ● ●  
1 $auth = new Auth();  
2 switch ($_SERVER['REQUEST_METHOD']) {  
3     case 'POST':  
4         $auth->login();  
5         break;  
6     case 'GET':  
7         $auth->check();  
8         break;  
9     case 'DELETE':  
10        $auth->logout();  
11        break;  
12    default:  
13        http_response_code(405);  
14        $response = [  
15            'message' => 'Method Not Allowed'  
16        ];  
17        echo json_encode($response, JSON_PRETTY_PRINT);  
18        break;  
19    }
```

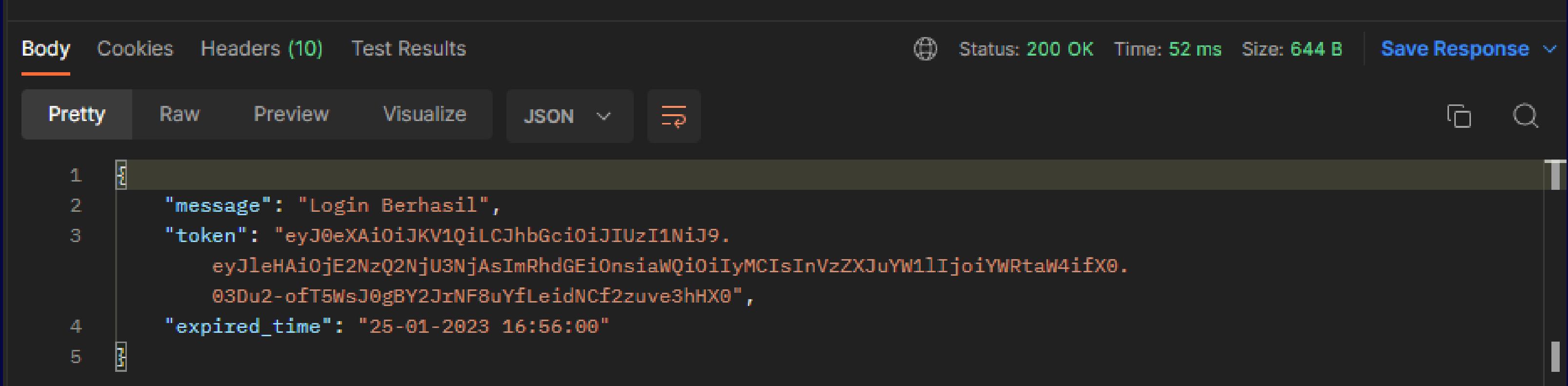
Ada 3 method yang digunakan yaitu login [POST], check [GET], dan logout[DELETE].

# Testing Login Menggunakan Postman

The screenshot shows the Postman application interface. On the left, the sidebar displays a project structure under 'Bootcamp' with 'Users' and 'Auth' sections. The 'Auth' section contains a single item, 'POST login', which is highlighted with a green border. The main workspace shows a POST request to 'http://localhost/mardirabootcamp/auth/'. The 'Body' tab is selected, showing a 'form-data' structure with two fields: 'username' (value: 'admin') and 'password' (value: 'admin'). Below the request, there's a placeholder message 'Click Send to get a response' accompanied by a cartoon rocket icon.

Selanjutnya akan melakukan testing menggunakan Postman yaitu menambahkan folder Auth dan request login dengan method POST, setelah itu gunakan body untuk mengirim data username dan password.

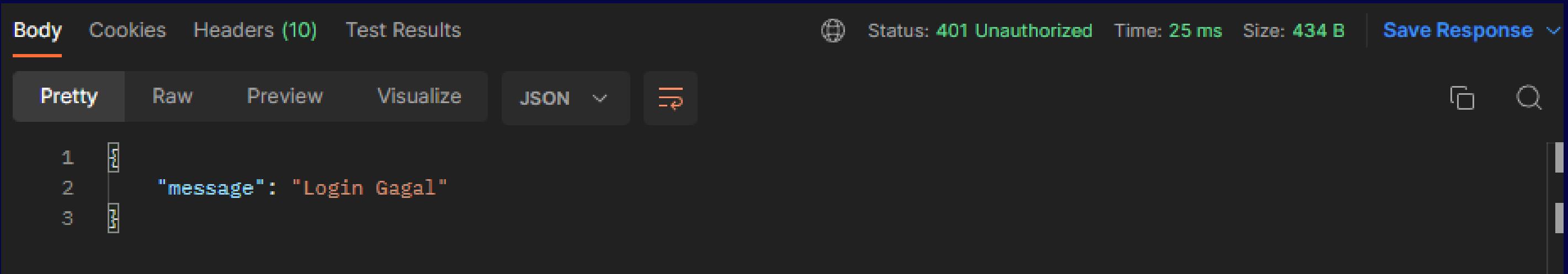
# Response Data Login



Body Cookies Headers (10) Test Results Status: 200 OK Time: 52 ms Size: 644 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Login Berhasil",
3   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
4       eyJleHAiOjE2NzQ2NjU3NjAsImRhdGEiOnsiaWQiOiIyMCIiInVzZXJuYW1lIjoiYWRtaW4ifX0.
5       03Du2-ofT5WsJ0gBY2JrNF8uYfLeidNCf2zuve3hHX0",
6   "expired_time": "25-01-2023 16:56:00"
7 }
```



Body Cookies Headers (10) Test Results Status: 401 Unauthorized Time: 25 ms Size: 434 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Login Gagal"
3 }
```

Response jika berhasil login maka akan muncul token yang akan digunakan oleh client/FE untuk disimpan untuk mengakses API yang memerlukan login.

# Setting Timezone untuk Expiry Time

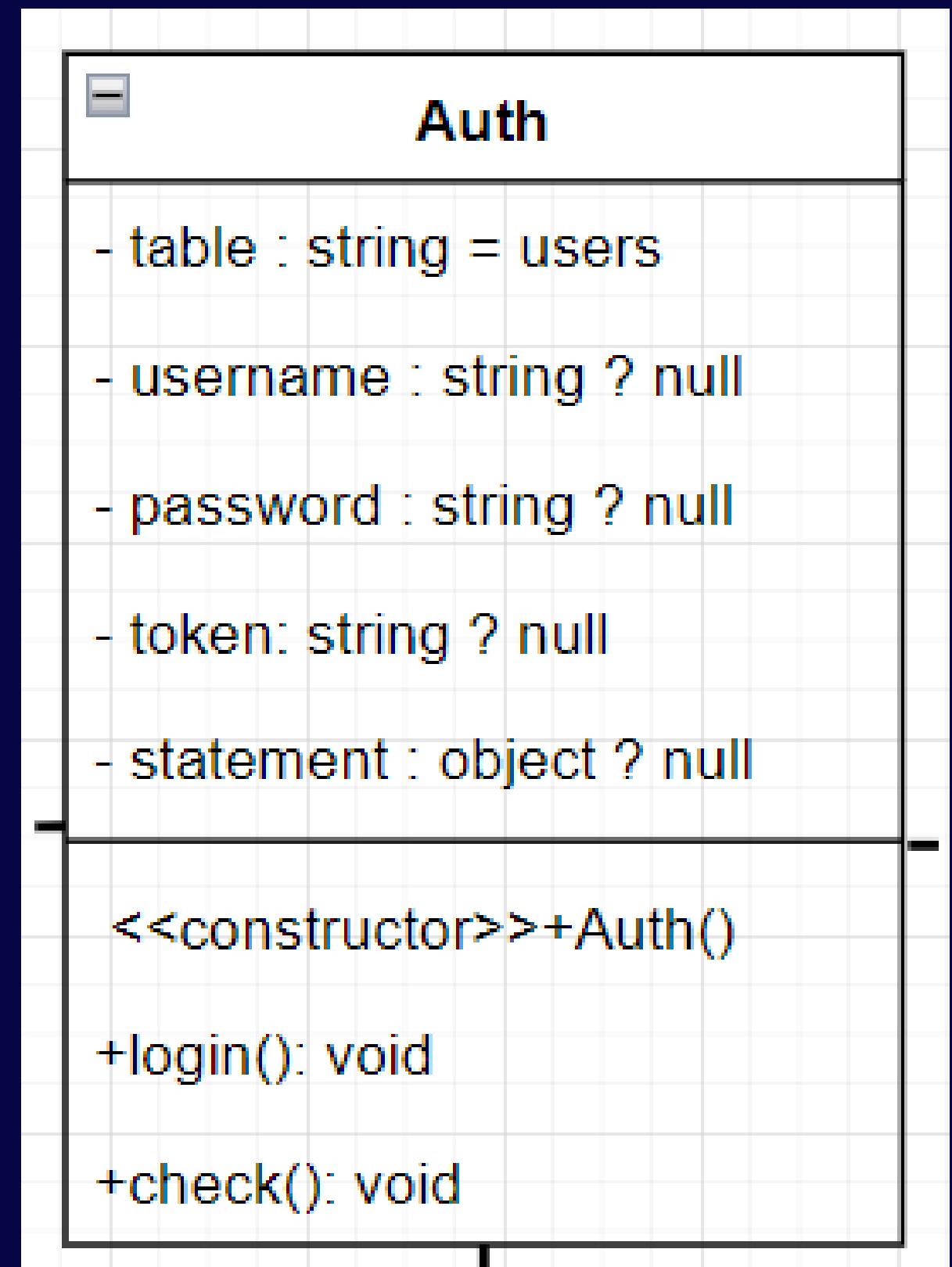
```
● ● ●  
1 <?php  
2  
3 // import script  
4 require_once('./Database.php');  
5 require_once('./vendor/autoload.php');  
6  
7 // menggunakan library JWT dan DotENV  
8 use Firebase\JWT\JWT;  
9 use Firebase\JWT\Key;  
10 use Dotenv\Dotenv;  
11  
12 // menggunakan header untuk mengakses API  
13 header('Content-Type: application/json');  
14 header('Access-Control-Allow-Origin: *');  
15 header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');  
16 header('Access-Control-Allow-Headers: X-Requested-With');  
17 header('Access-Control-Allow-Headers: Content-Type, Authorization');  
18  
19 date_default_timezone_set('Asia/Jakarta');
```

Supaya waktu kadaluwarsa sesuai dengan real time atau waktu yang asli dengan cara menambahkan fungsi `date_default_timezone_set('Asia/Jakarta')`.

# Membuat Method Check Login

```
1 public function check(): void  
2 {  
3  
4 }
```

Setelah membuat method login, selanjutnya yaitu membuat method check untuk melakukan cek login.



# Cek Token Pada Method Check Login



```
1 // jika token tidak ada
2 if (!$this->token) {
3     // membuat response
4     $response = [
5         'message' => 'Token tidak ada'
6     ];
7     http_response_code(401);
8     echo json_encode($response, JSON_PRETTY_PRINT);
9     exit;
10 }
```

Pada method check login, tambahkan statement untuk mengecek token tersebut ada atau tidak

# Memecah Token & Cek Kesesuaian Token



```
1 // memecah token
2 $token = explode(' ', $this→token);
3 // jika token tidak sesuai
4 if ($token[0] != 'Bearer') {
5     // membuat response
6     $response = [
7         'message' => 'Token tidak sesuai'
8     ];
9     http_response_code(401);
10    echo json_encode($response, JSON_PRETTY_PRINT);
11    exit;
12 }
```

Setelah cek token tersebut, tambahkan statement untuk memecah token dan cek format token tersebut sesuai atau tidak.

# Fungsi Explode & Penjelasan



```
1 var_dump($this->token); die;
```



```
1 $token = explode(' ', $this->token);
2 var_dump($token); die;
```



```
1 if ($token[0] != 'Bearer') {
```

```
string(166) "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJleHAiOjE2NzQ2NjY2NTcsImRhdGEiOnsiaWQiOiIyMCIsInVzZXJuYW1lIjoiaGFvZ2VuZyJ9fQ.
tb25q0nIWdYhGRJoFfmIl94C0RpmqNQjjOuvwxi3HQZY"
```

```
array(2) {
    [0]=>
    string(6) "Bearer"
    [1]=>
    string(159) "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJleHAiOjE2NzQ2NjY2NTcsImRhdGEiOnsiaWQiOiIyMCIsInVzZXJuYW1lIjoiaGFvZ2VuZyJ9fQ.
tb25q0nIWdYhGRJoFfmIl94C0RpmqNQjjOuvwxi3HQZY"
}
```



```
1 $jwt = JWT::decode($token[1]
```

Alasan menggunakan explode karena token pada property this->token ada kata "Bearer" di bagian value nya, untuk memisahkan kata "Bearer" dengan JWT menggunakan fungsi explode dimana string akan pecah / split menjadi array terindeks, dimana index ke 0 adalah "Bearer" sedangkan index ke 1 adalah JWT.

# Dekripsi / Decrypt Token



```
1 // dekripsi / decrypt token
2 try {
3     $jwt = JWT::decode($token[1], new Key($_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256'));
4 } catch (\Exception $e) {
5     // membuat response
6     $response = [
7         'message' => $e->getMessage()
8     ];
9     http_response_code(401);
10    echo json_encode($response, JSON_PRETTY_PRINT);
11    exit;
12 }
```

Selanjutnya pada method check tambahkan baris kode selanjutnya yaitu decrypt token, dimana token yang sudah di pecah. Kemudian token tersebut akan didekripsi menjadi data semula, dengan menggunakan static method decode dan parameternya yaitu token, secret key serta jenis hash yang digunakan.

# Validasi Data ke Database



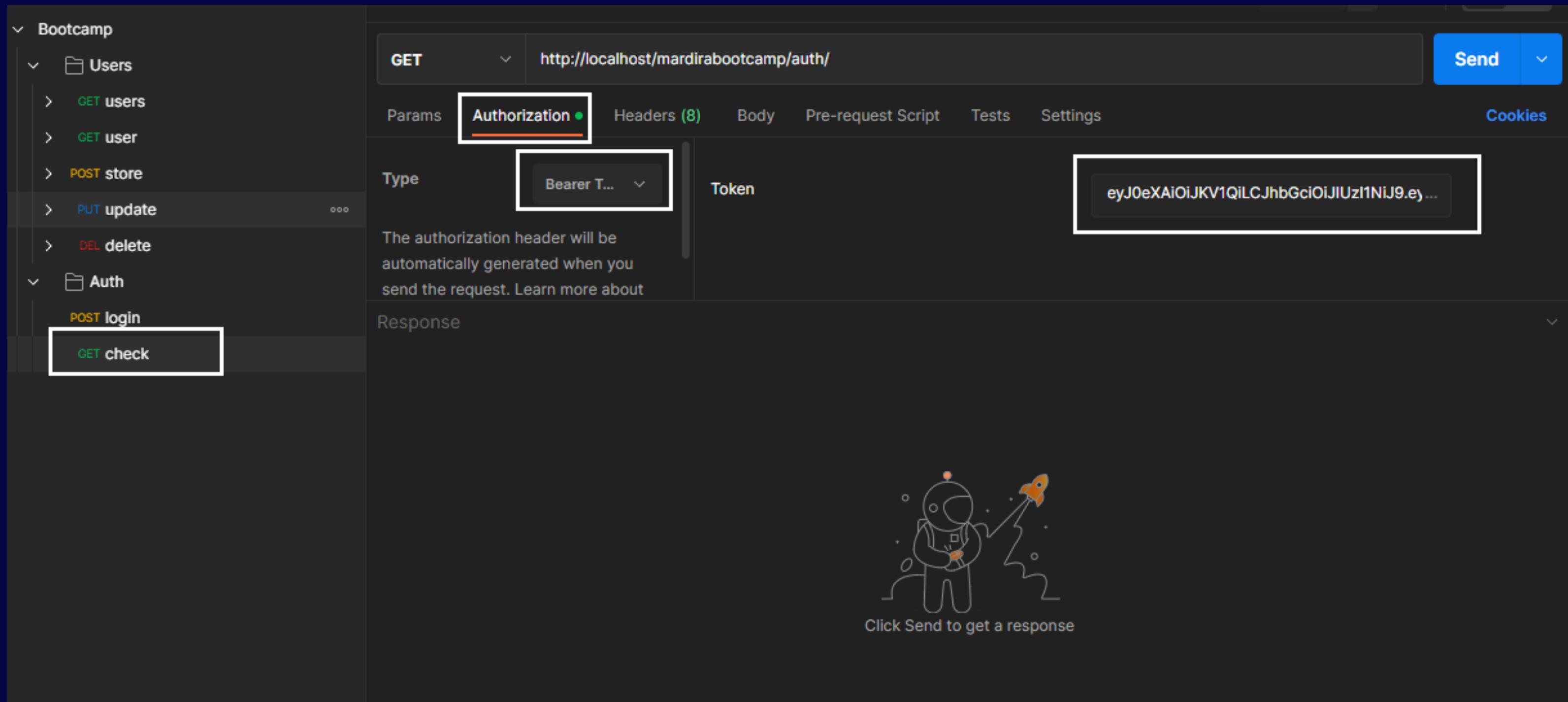
```
1 // cek jika data sesuai dengan database
2 $query = "SELECT * FROM {$this→table} WHERE username = :username";
3 $statement = $this→statement→prepare($query);
4 // binding data
5 $statement→bindParam(':username', $jwt→data→username);
6 // eksekusi query
7 $statement→execute();
8 // jika data tidak ditemukan
9 if ($statement→rowCount() === 0) {
10     // membuat response
11     $response = [
12         'message' => 'Token tidak valid'
13     ];
14     http_response_code(401);
15     echo json_encode($response, JSON_PRETTY_PRINT);
16     exit;
17 }
```

Ketika sudah melakukan dekripsi token menjadi data semula, kemudian cek jika data semula yang telah dikonversi terdata di database.

```
1 public function check(): void
2 {
3
4     // jika token tidak ada
5     if (!$this->token) {
6         // membuat response
7         $response = [
8             'message' => 'Token tidak ada'
9         ];
10        http_response_code(401);
11        echo json_encode($response, JSON_PRETTY_PRINT);
12        exit;
13    }
14
15    // memecah token
16    $token = explode(' ', $this->token);
17
18    // jika token tidak sesuai
19    if ($token[0] != 'Bearer') {
20        // membuat response
21        $response = [
22            'message' => 'Token tidak sesuai'
23        ];
24        http_response_code(401);
25        echo json_encode($response, JSON_PRETTY_PRINT);
26        exit;
27    }
28
29    // dekripsi / decrypt token
30    try {
31        $jwt = JWT::decode($token[1], new Key($_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256'));
32    } catch (\Exception $e) {
33        // membuat response
34        $response = [
35            'message' => $e->getMessage()
36        ];
37        http_response_code(401);
38        echo json_encode($response, JSON_PRETTY_PRINT);
39        exit;
40    }
41
42    // cek jika data sesuai dengan database
43    $query = "SELECT * FROM {$this->table} WHERE username = :username";
44    $statement = $this->statement->prepare($query);
45    // binding data
46    $statement->bindParam(':username', $jwt->data->username);
47    // eksekusi query
48    $statement->execute();
49    // jika data tidak ditemukan
50    if ($statement->rowCount() == 0) {
51        // membuat response
52        $response = [
53            'message' => 'Token tidak valid'
54        ];
55        http_response_code(401);
56        echo json_encode($response, JSON_PRETTY_PRINT);
57        exit;
58    }
59
60    // membuat response
61    $response = [
62        'message' => 'Token valid',
63        'data' => $jwt->data
64    ];
65    http_response_code(200);
66    echo json_encode($response, JSON_PRETTY_PRINT);
67 }
```

# Method Check Secara Keseluruhan

# Testing Check Login Menggunakan Postman



The screenshot shows the Postman interface with a dark theme. On the left, the 'Bootcamp' collection is expanded, showing 'Users' and 'Auth' sections. Under 'Auth', 'POST login' and 'GET check' are listed. The main workspace shows a 'GET' request to 'http://localhost/mardirabootcamp/auth/'. The 'Authorization' tab is selected in the header section, which contains a 'Type' dropdown set to 'Bearer T...' and a 'Token' input field containing a long JWT token. Other tabs in the header include 'Params', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. A note below the header states: 'The authorization header will be automatically generated when you send the request. Learn more about'. At the bottom right, there is a cartoon character of a person launching a rocket, with the text 'Click Send to get a response'.

Untuk melakukan cek login, pilih tab Authorization, kemudian pilih type Bearer Token, dan paste JWT yang telah digenerate setelah melakukan login.

# Response Check Login

```
1  "message": "Token tidak ada"
2
3
```

```
1  "message": "Expired token"
2
3
```

```
1  "message": "Token tidak sesuai"
2
3
```

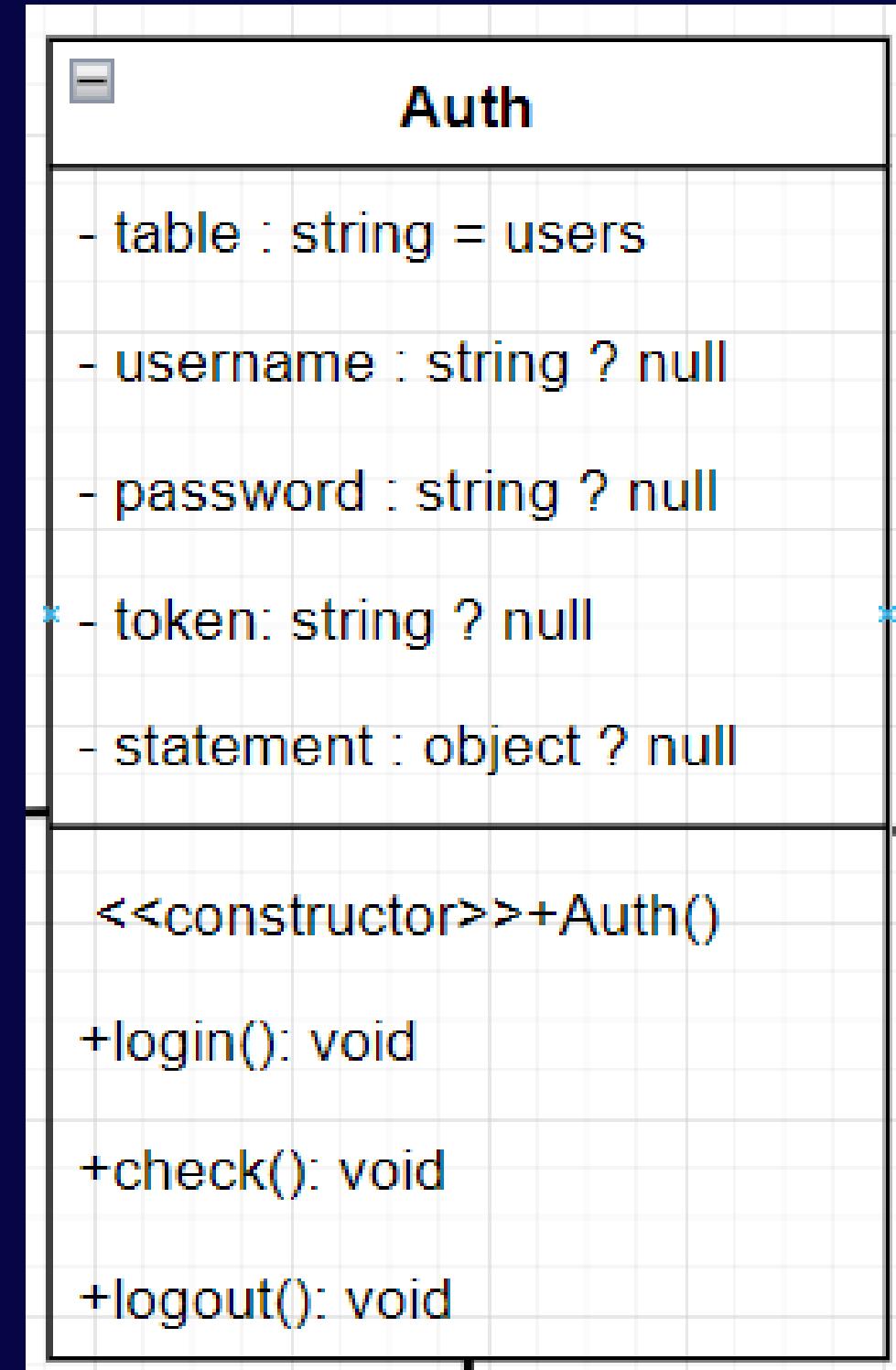
```
1  "message": "Token tidak valid"
2
3
```

```
1  "message": "Token valid",
2  "data": {
3    "id": "20",
4    "username": "admin"
5  }
6
7
```

Beberapa response token ketika melakukan check login sesuai dengan kondisi.

# Method Logout

```
● ● ●
1 public function logout(): void
2 {
3     // cek token bearer
4     if (!$this->token) {
5         // membuat response
6         $response = [
7             'message' => 'Token tidak ada'
8         ];
9         http_response_code(401);
10        echo json_encode($response, JSON_PRETTY_PRINT);
11        exit;
12    }
13
14    $token = explode(' ', $this->token);
15    // decode data token
16    try {
17        $jwt = JWT::decode($token[1], new Key($_ENV['ACCESS_TOKEN_SECRET_KEY'], 'HS256'));
18    } catch (\Exception $e) {
19        // membuat response
20        $response = [
21            'message' => $e->getMessage()
22        ];
23        http_response_code(401);
24        echo json_encode($response, JSON_PRETTY_PRINT);
25        exit;
26    }
27
28    // cek jika data sesuai dengan database
29    $query = "SELECT * FROM {$this->table} WHERE username = :username";
30    $statement = $this->statement->prepare($query);
31    // binding data
32    $statement->bindParam(':username', $jwt->data->username);
33    // eksekusi query
34    $statement->execute();
35    // jika data tidak ditemukan
36    if ($statement->rowCount() === 0) {
37        // membuat response
38        $response = [
39            'message' => 'Token tidak valid'
40        ];
41        http_response_code(401);
42        echo json_encode($response, JSON_PRETTY_PRINT);
43        exit;
44    }
45
46    // membuat response
47    $response = [
48        'message' => 'Logout Berhasil'
49    ];
50    http_response_code(200);
51    echo json_encode($response, JSON_PRETTY_PRINT);
52 }
```



# Testing Logout Menggunakan Postman

The screenshot shows the Postman application interface. On the left, the 'Collection' sidebar displays a tree structure with 'Bootcamp' expanded, showing 'Users' and 'Auth' folders. Under 'Auth', there are three items: 'POST login', 'GET check', and 'DEL logout'. The main workspace shows a DELETE request for 'http://localhost/mardirabootcamp/auth/'. The 'Authorization' tab is selected, showing a 'Bearer Token' field with a placeholder 'Token' and a long JWT token. Below the authorization section, a note states: 'The authorization header will be automatically generated when you send the request. Learn more about'. The 'Body' tab is selected, showing a JSON response with the message: "message": "Logout Berhasil" (Logout Successful). The status bar at the bottom indicates a 200 OK status, 28 ms time, and 428 B size.

Penerapan method logout hampir sama dengan method check login.

# Controller & Model Pattern

Controller model pattern atau yang sering dikenal **MVC (Model View Controller)** adalah sebuah pola arsitektur dalam membuat sebuah aplikasi dengan cara memisahkan kode menjadi 3 bagian yang terdiri dari:

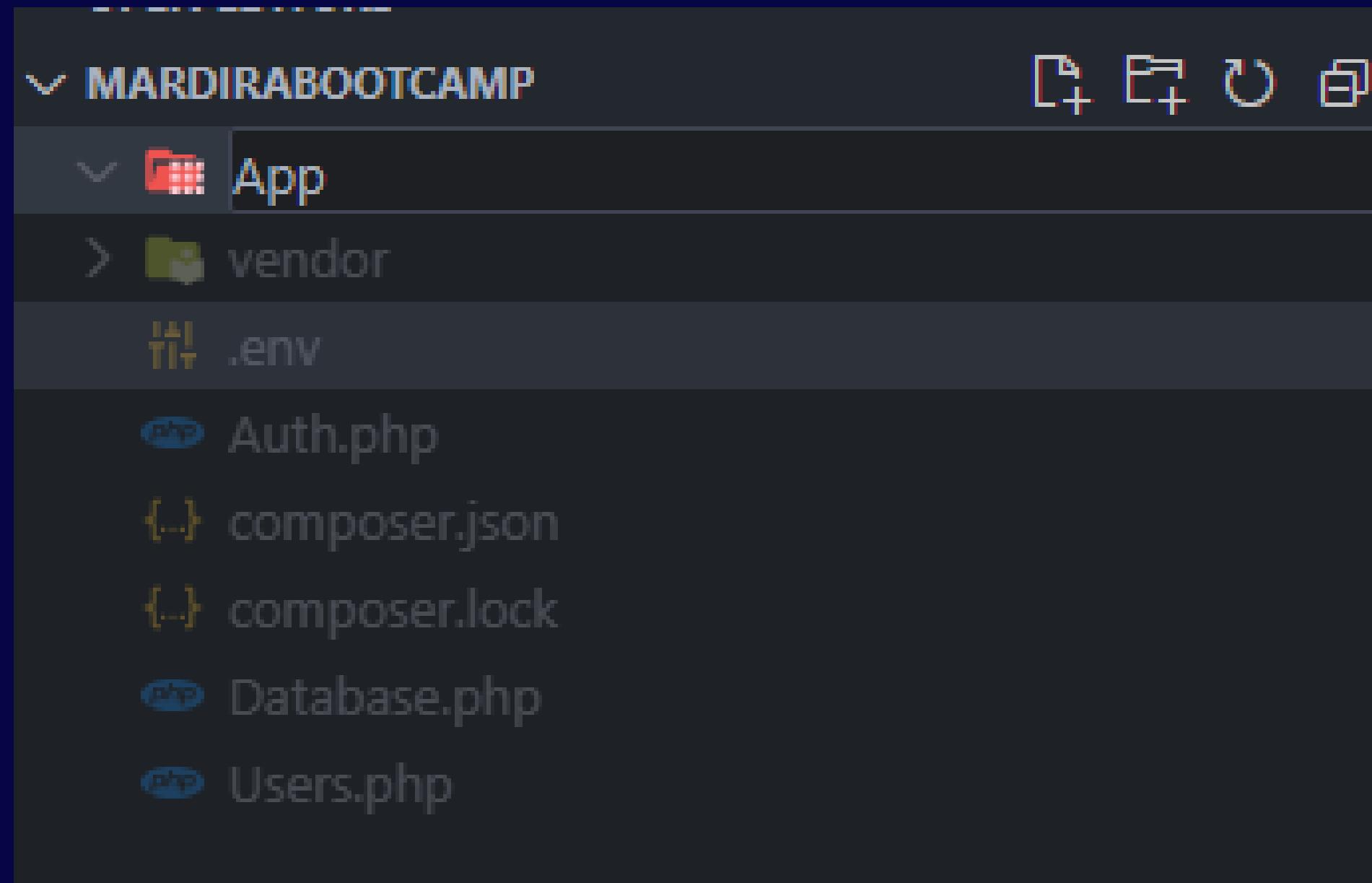
- Model : Bagian yang bertugas untuk menyiapkan, mengatur, memanipulasi, dan mengorganisasikan data yang ada di database.
- View : Bagian yang bertugas untuk menampilkan informasi dalam bentuk Graphical User Interface (GUI).
- Controller : Bagian yang bertugas untuk menghubungkan serta mengatur model dan view agar dapat saling terhubung.

Karena tidak menggunakan view jadi yang hanya digunakan hanyalah Controller & Model saja.

# Diagram Controller & Model



# Persiap Membuat Asitektur MC (Model Controller)



Buatlah folder dengan nama App untuk tempat direktori Model dan Controller

# Setup Autoload Dengan PSR4

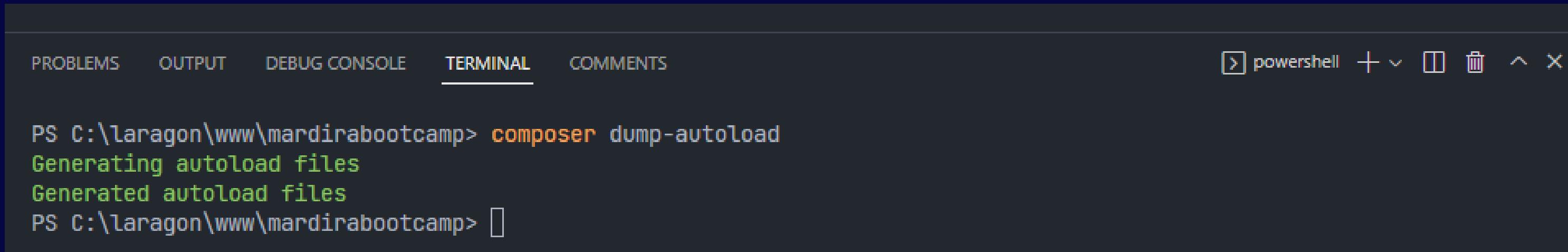


```
1 {  
2     "name": "bootcamp/jwt",  
3     "description": "Membuat otentikasi dengan JWT",  
4     "authors": [  
5         {  
6             "name": "Muhammad Iqbal",  
7             "email": "iqbal.1995@gmail.com"  
8         },  
9     ],  
10    "require": {  
11        "vlucas/phpdotenv": "^5.5",  
12        "firebase/php-jwt": "^6.3"  
13    },  
14    "autoload": {  
15        "psr-4": {  
16            "App\\": "App/"  
17        }  
18    }  
19 }  
20 }
```

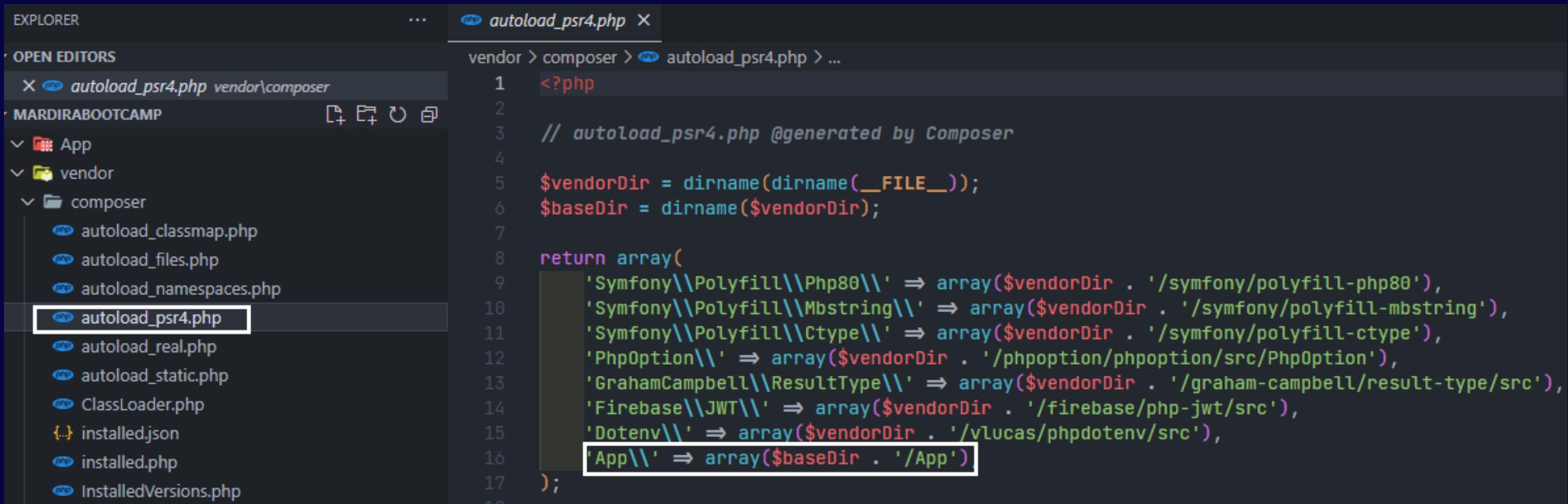
Setup autoload dengan PSR4 (PHP Standards Recommendations) ke-4, yaitu standarisasi penulisan programmer sehingga mudah untuk melakukan kolaborasi atau kerja sama, dimana fungsinya adalah meminimalisir menggunakan fungsi require / include untuk memanggil file, karena harus include satu per satu file dan besar kemungkinan bakal banyak kode redundant / repetitif, PSR4 biasanya digunakan oleh framework populer seperti Laravel, CI, Yii, dll.

Pada file composer.json tambahkan baris kode json autoload, dengan menyebutkan direktori "App" yang telah dibuat.

# Menjalankan Perintah Autoload



PS C:\laragon\www\mardirabootcamp> composer dump-autoload  
Generating autoload files  
Generated autoload files  
PS C:\laragon\www\mardirabootcamp>

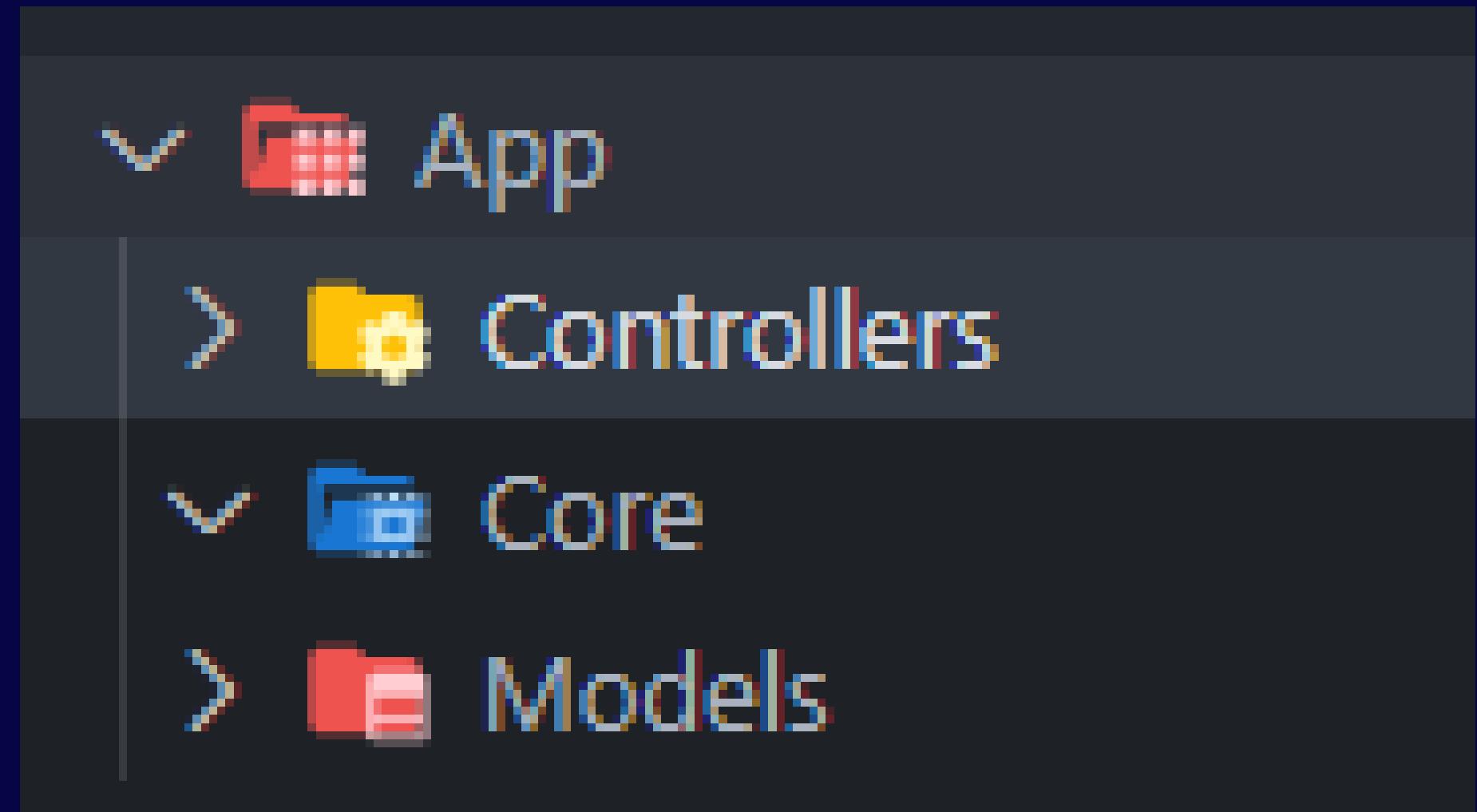


The screenshot shows a code editor with the 'autoload\_psr4.php' file open. The file is a PHP script that generates an array of class mappings for autoloading. The code is as follows:

```
1 <?php
2
3 // autoload_psr4.php @generated by Composer
4
5 $vendorDir = dirname(dirname(__FILE__));
6 $baseDir = dirname($vendorDir);
7
8 return array(
9     'Symfony\\Polyfill\\Php80\\' => array($vendorDir . '/symfony/polyfill-php80'),
10    'Symfony\\Polyfill\\Mbstring\\' => array($vendorDir . '/symfony/polyfill-mbstring'),
11    'Symfony\\Polyfill\\Ctype\\' => array($vendorDir . '/symfony/polyfill-ctype'),
12    'PhpOption\\' => array($vendorDir . '/phoption/phoption/src/PhpOption'),
13    'GrahamCampbell\\ResultType\\' => array($vendorDir . '/graham-campbell/result-type/src'),
14    'Firebase\\JWT\\' => array($vendorDir . '/firebase/php-jwt/src'),
15    'Dotenv\\' => array($vendorDir . '/vlucas/phpdotenv/src'),
16    'App\\' => array($baseDir . '/App')
17);
```

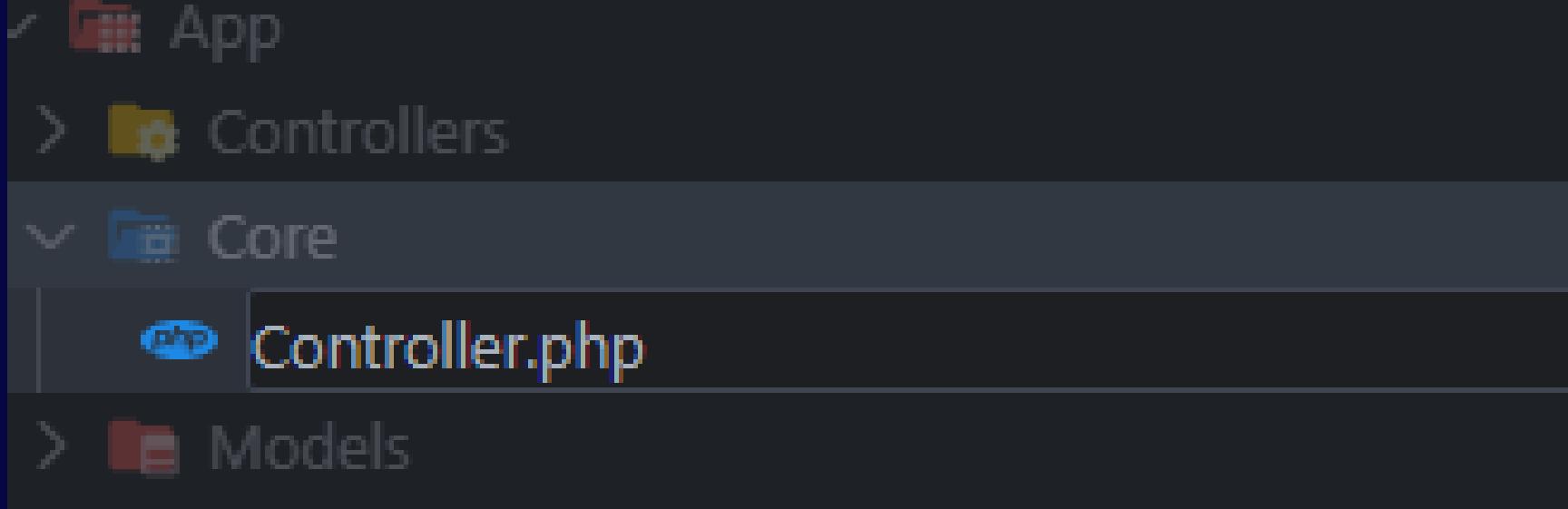
Kemudian jalankan perintah composer dump-autoload dimana fungsinya akan membuat file autoload\_psr4.php di folder vendor/composer/autoload\_psr4.php

# Struktur Folder MC Pattern



Struktur folder MC (Model Controller) pattern, terdiri dari 3 direktori yaitu Controllers fungsinya sebagai tempat untuk membuat file controller, Models untuk tempat membuat file models, dan core adalah tempat untuk kumpulan beberapa file atau sistem utama dimana kita bisa menggunakan controller dan model tersebut.

# Membuat Controller Utama



```
1 <?php
2
3 namespace App\Core;
4
5 header('Content-Type: application/json');
6 header('Access-Control-Allow-Origin: *');
7 header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
8 header('Access-Control-Allow-Headers: X-Requested-With');
9
```

Setelah membuat struktur folder, selanjutnya tambahkan header dan namespace agar controller utama dapat digunakan file yang berada di folder controllers.

# Membuat Class Controller



```
1 <?php
2
3 namespace App\Core;
4
5 header('Content-Type: application/json');
6 header('Access-Control-Allow-Origin: *');
7 header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
8 header('Access-Control-Allow-Headers: X-Requested-With');
9
10
11 class Controller
12 {
13
14 }
15
```

Membuat class controller utama sama layaknya kita membuat class pada OOP (Object Oriented Programming)

# Method Response Controller

```
● ● ●  
1 namespace App\Core;  
2  
3 class Controller  
4 {  
5     public function response(int $status_code, $response): void  
6     {  
7         http_response_code($status_code);  
8         header('Content-Type: application/json');  
9         echo json_encode($response);  
10        exit;  
11    }  
12 }
```

Membuat response pada class controller gunanya untuk memanggil response ketika ingin membuat response API.

# Perbedaan Pemanggilan Response



```
1 $response = [  
2     'message' => 'Data User',  
3     'data' => $user  
4 ];  
5 http_response_code(200);  
6 echo json_encode($response, JSON_PRETTY_PRINT);
```

Menjadi



```
1 $users = [  
2     'message' => 'Data User',  
3     'data' => $user  
4 ];  
5 $this->response(200, $users);
```

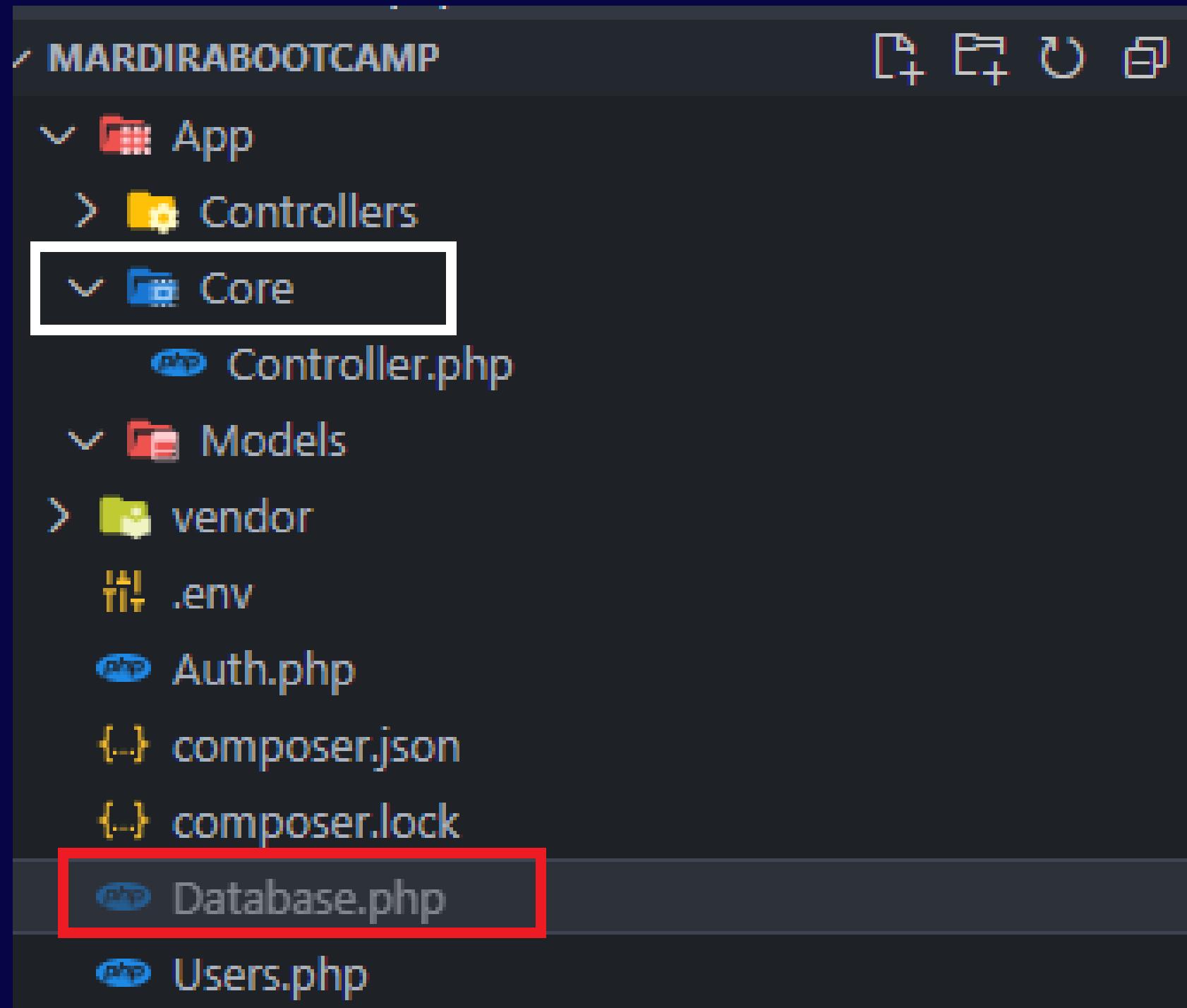
Untuk penerapan pemanggilan response API yang biasanya menggunakan cara pertama seperti gambar di sebelah kiri, menjadi lebih simple seperti gambar di sebelah kanan.

# Method Model Controller Utama

```
● ● ●  
1 <?php  
2  
3 namespace App\Core;  
4  
5 header('Content-Type: application/json');  
6 header('Access-Control-Allow-Origin: *');  
7 header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');  
8 header('Access-Control-Allow-Headers: X-Requested-With');  
9  
10  
11 class Controller  
12 {  
13     public function response(int $status_code, $response): void  
14     {  
15         http_response_code($status_code);  
16         header('Content-Type: application/json');  
17         echo json_encode($response);  
18         exit;  
19     }  
20  
21     public function model(string $model): object  
22     {  
23         if (is_array($model)) {  
24             $model = array_map(function ($model) {  
25                 $model = 'App\\Models\\' . $model;  
26                 $model = $model . 'Models';  
27                 return new $model;  
28             }, $model);  
29             return $model;  
30         }  
31         $model = 'App\\Models\\' . $model;  
32         return new $model;  
33     }  
34 }
```

Method model berfungsi untuk memanggil model saat pembuatan controller di folder App\Controllers.

# Memindahkan File Database.php



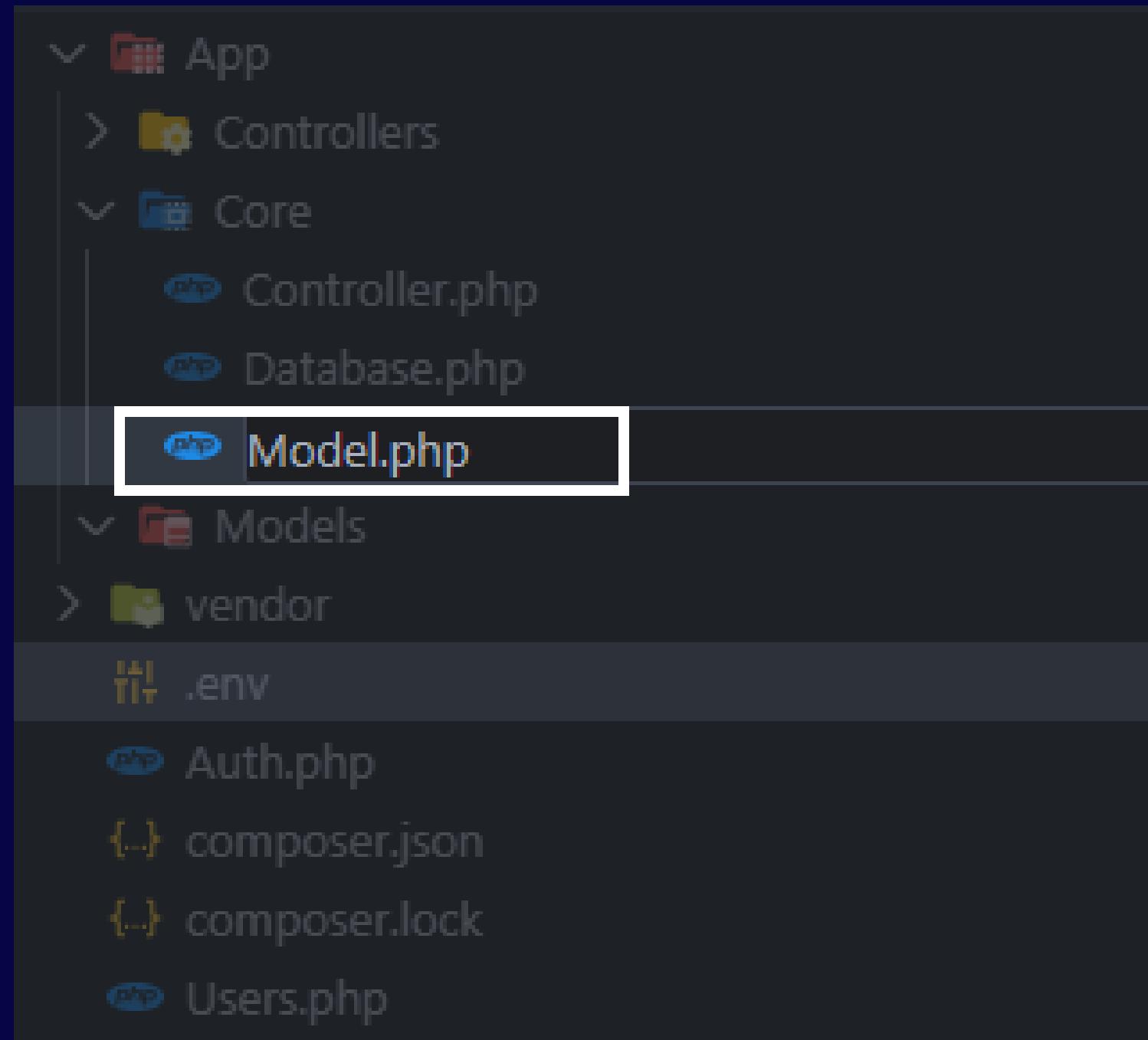
Pindahkan file Database.php ke folder Core.

# Namespace Database.php

```
● ● ●  
1  <?php  
2  
3  namespace App\Core;  
4  
5  use PDO;  
6  use PDOException;  
7  
8  class Database  
9  {  
10     private const DB_HOST = 'localhost';  
11     private const DB_USER = 'root';  
12     private const DB_PASS = '';  
13     private const DB_NAME = 'pdo';  
14     private const DSN = 'mysql:host=' . self::DB_HOST . ';dbname=' . self::DB_NAME . '';  
15     public ?object $connection = null;  
16  
17     public function __construct()  
18     {  
19         try {  
20             $this->connection = new PDO(  
21                 self::DSN,  
22                 self::DB_USER,  
23                 self::DB_PASS  
24             );  
25             $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
26         } catch (PDOException $e) {  
27             http_response_code(500);  
28             echo $e->getMessage();  
29         }  
30         return $this->connection;  
31     }  
32 }
```

Tambahkan namespace dan use PDO, supaya bisa digunakan pada model utama dengan keyword "use".

# Membuat Model Utama



```
1 <?php
2
3 namespace App\Core;
4
5 use App\Core\Database;
6 use PDO;
```

Buatlah file Model.php di folder core, dan setelah itu gunakan keyword untuk memanggil class Database dan PDO

# Membuat Class Model

```
1 <?php
2
3 namespace App\Core;
4
5 use App\Core\Database;
6 use PDO;
7
8 class Model
9 {
10     protected object $statement;
11     protected string $table;
12     protected string $primaryKey;
13
14     public function __construct()
15     {
16         $this→statement = new Database();
17         $this→statement = $this→statement→connection;
18     }
19
20     public function all(): array
21     {
22         $query = "SELECT * FROM {$this→table}";
23         $statement = $this→statement→prepare($query);
24         $statement→execute();
25         return $statement→fetchAll(PDO::FETCH_OBJ);
26     }
27 }
```

Selanjutnya membuat class Model dan membuat method all untuk menampilkan semua data.

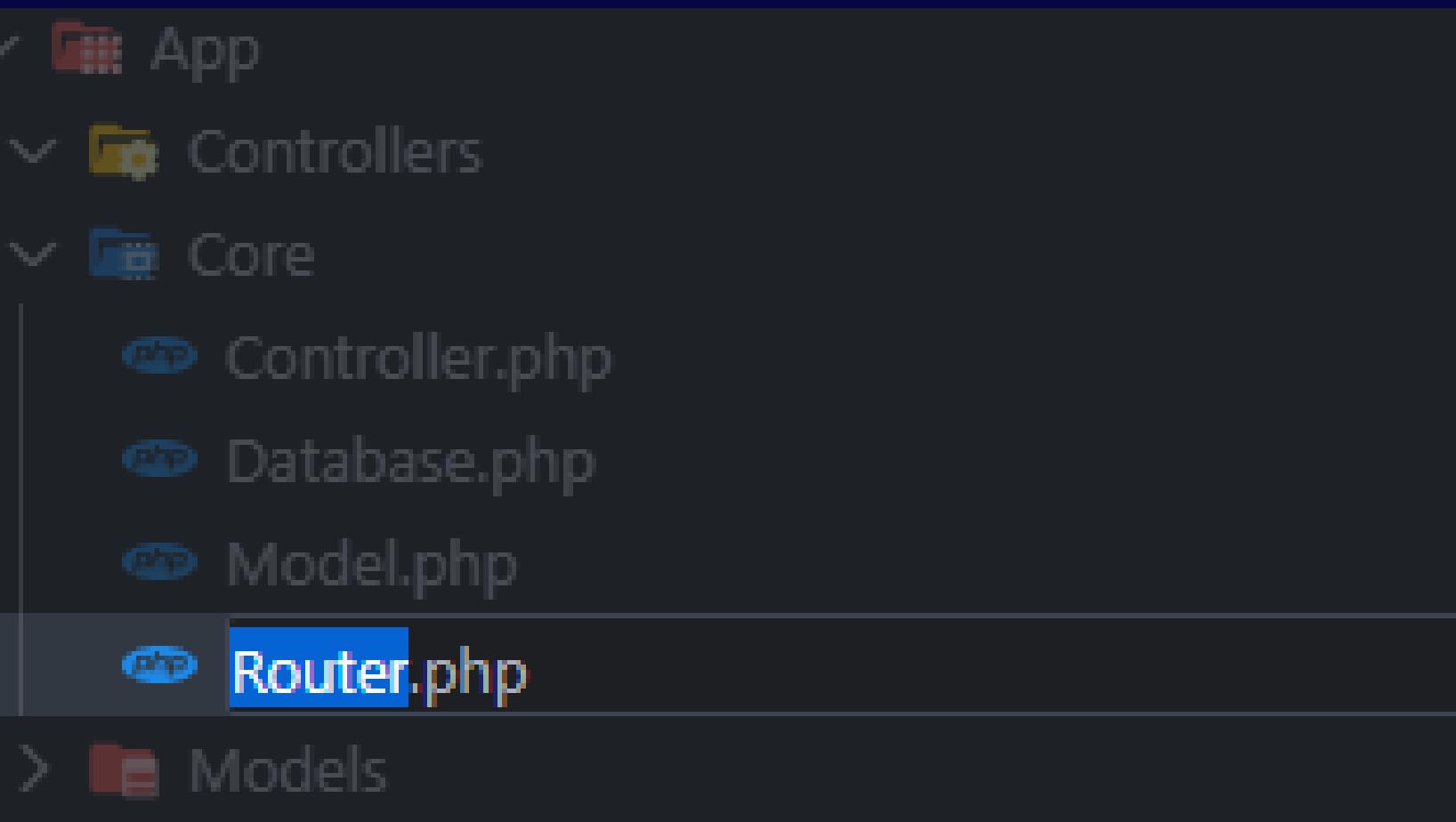
# Membuat Class Model

```
1 <?php
2
3 namespace App\Core;
4
5 use App\Core\Database;
6 use PDO;
7
8 class Model
9 {
10     protected object $statement;
11     protected string $table;
12     protected string $primaryKey;
13
14     public function __construct()
15     {
16         $this→statement = new Database();
17         $this→statement = $this→statement→connection;
18     }
19
20     public function all(): array
21     {
22         $query = "SELECT * FROM {$this→table}";
23         $statement = $this→statement→prepare($query);
24         $statement→execute();
25         return $statement→fetchAll(PDO::FETCH_OBJ);
26     }
27 }
```

Selanjutnya membuat class Model dan membuat method all untuk menampilkan semua data.

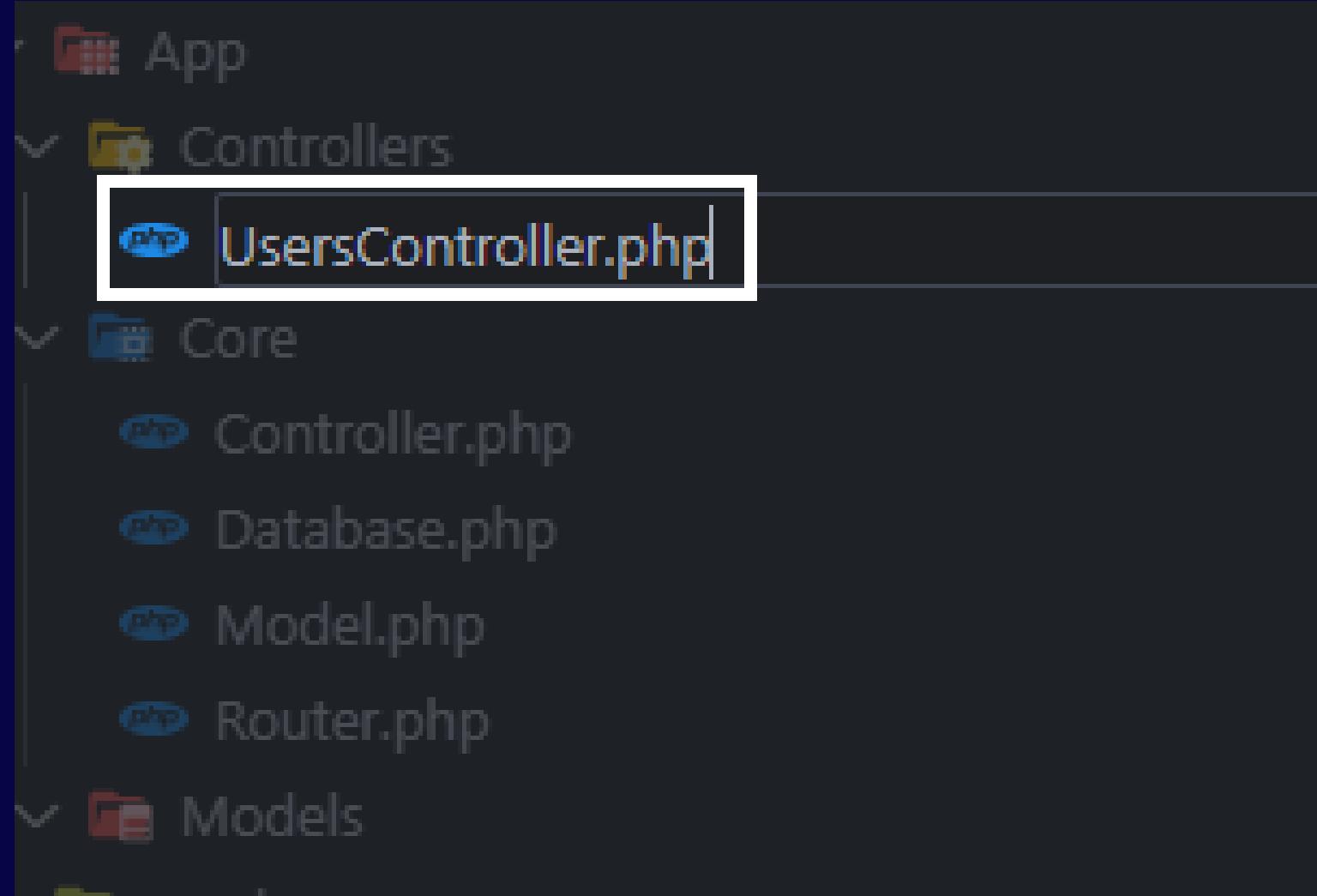
# Membuat Router

```
1 <?php
2
3 namespace App\Core;
4
5 class Router
6 {
7     private static $routes = [];
8
9     public static function add(
10         string $method,
11         string $path,
12         string $controller,
13         string $function,
14         array $middlewares = []
15     ): void {
16         self::$routes[] = [
17             'method' => $method,
18             'path' => $path,
19             'controller' => $controller,
20             'function' => $function,
21             'middleware' => $middlewares
22         ];
23     }
24
25     public static function run(): void
26     {
27         $requestMethod = $_SERVER['REQUEST_METHOD'];
28         $requestUri = $_SERVER['REQUEST_URI'];
29
30         foreach (self::$routes as $route) {
31             if ($route['method'] === $requestMethod) {
32                 $path = $route['path'];
33                 $path = str_replace('/', '\\/', $path);
34                 $path = preg_replace('/\{[a-zA-Z0-9]+\}/', '([a-zA-Z0-9]+)', $path);
35                 $path = '/^' . $path . '$/';
36                 if (preg_match($path, $requestUri, $matches)) {
37                     $controller = $route['controller'];
38                     $function = $route['function'];
39                     $controller = new $controller;
40                     $controller->$function(...array_slice($matches, 1));
41                     return;
42                 }
43             }
44         }
45         http_response_code(404);
46         $response = [
47             'status' => 'error',
48             'message' => 'Not Found'
49         ];
50         header('Content-Type: application/json');
51         echo json_encode($response);
52     }
53 }
```



Langkah selanjutnya yaitu membuat Router, dimana router itu berfungsi untuk membuat jalur http, saat melakukan request melalui URL seperti GET localhost/users/

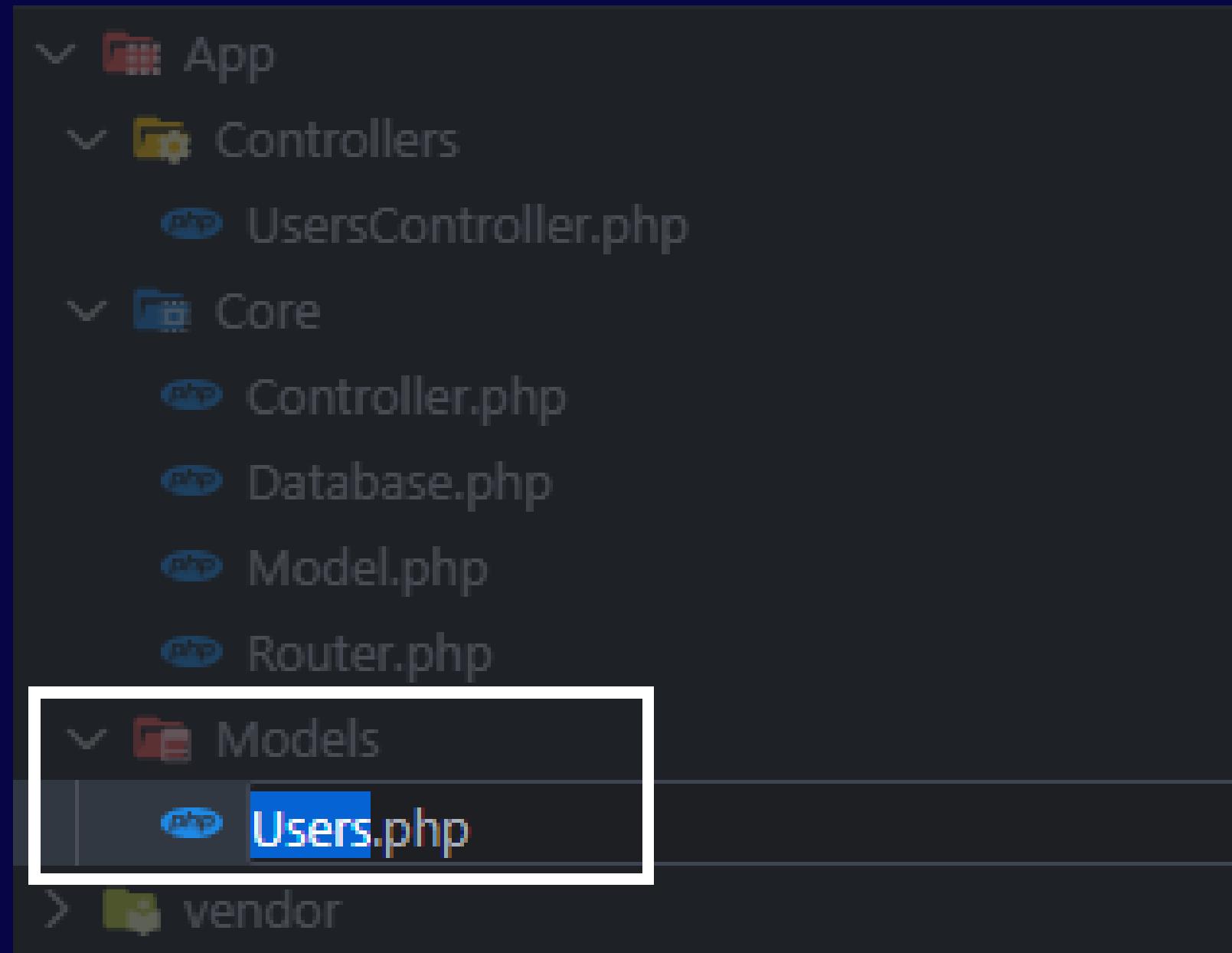
# Membuat Class Controller Users



```
1 <?php
2
3 namespace App\Controllers;
4
5 use App\Core\Controller;
6
7 class UsersController extends Controller
8 {
9
10 }
11
```

Untuk membuat file controller, dengan nama UsersController di folder Controllers, dan setelah itu buat class dengan metode pewarisan (Inheritance) pada class parent Controller.

# Membuat Model Users



```
● ● ●
1 <?php
2
3 namespace App\Models;
4
5 use App\Core\Model;
6
7 class Users extends Model
8 {
9     protected string $table = 'users';
10    protected string $primaryKey = 'user_id';
11 }
```

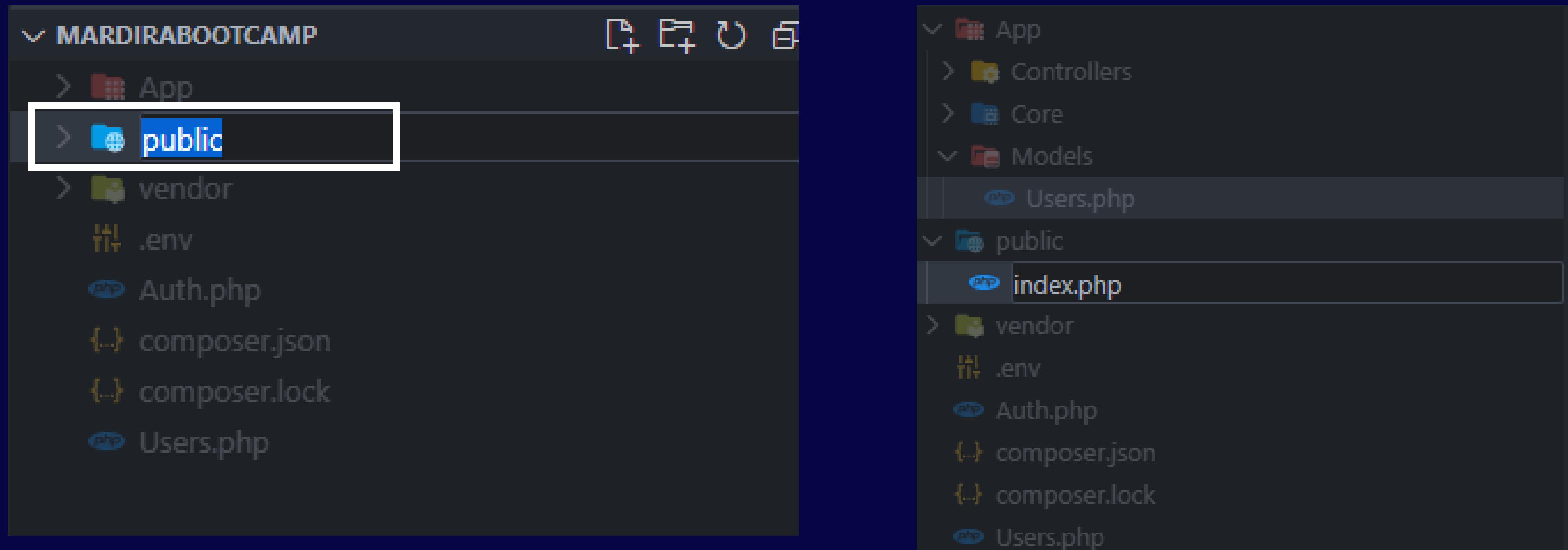
Setelah membuat controller, untuk membuat model Users dengan nama file Users yang berada di folder Models

# Menambahkan Method Controller

```
● ● ●  
1 <?php  
2  
3 namespace App\Controllers;  
4  
5 use App\Core\Controller;  
6  
7 class UsersController extends Controller  
8 {  
9     public function index() : void  
10    {  
11        $users = $this->model('Users')->all();  
12        $this->response(200, $users);  
13    }  
14 }  
15
```

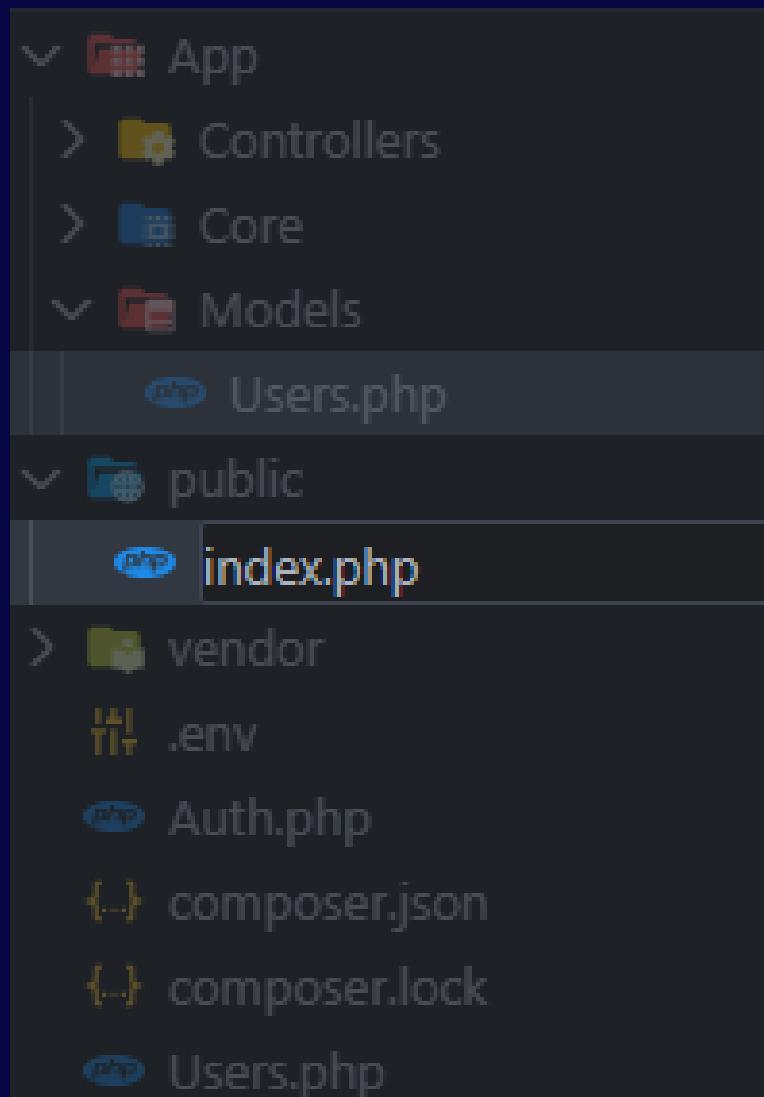
Selanjutnya buat method index untuk menampilkan semua data users dengan memanggil model Users yang telah dibuat.

# Membuat Folder Public & Index



Membuat folder public dan index bertujuan untuk menjalankan aplikasi melalui direktori public, agar tidak langsung memanggil file yang ada di Direktori App, metode ini sering digunakan oleh framework seperti Laravel, dan CI4

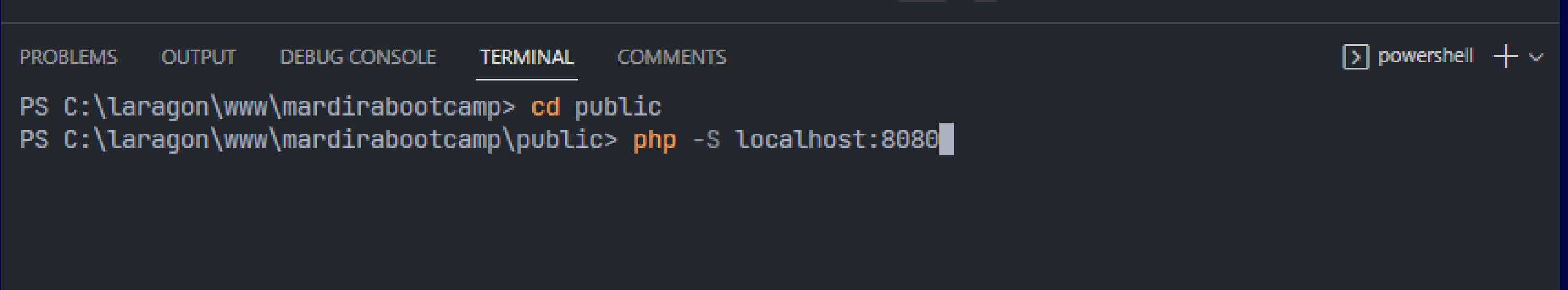
# Menjalankan Router



```
● ● ●  
1 <?php  
2  
3 require_once __DIR__ . '/../vendor/autoload.php';  
4  
5 use App\Core\Router;  
6  
7 use App\Controllers\UsersController;  
8  
9 Router::add('GET', '/users', UsersController::class, 'index');  
10  
11 Router::run();
```

Setelah membuat file index.php, dengan cara menggunakan keyword require\_once untuk memanggil autoload.php yang ada di composer, setelah itu untuk menggunakan route ada 3 argument yaitu HTTP Verbs, Controllers dan method yang akan digunakan.

# Menjalankan Aplikasi



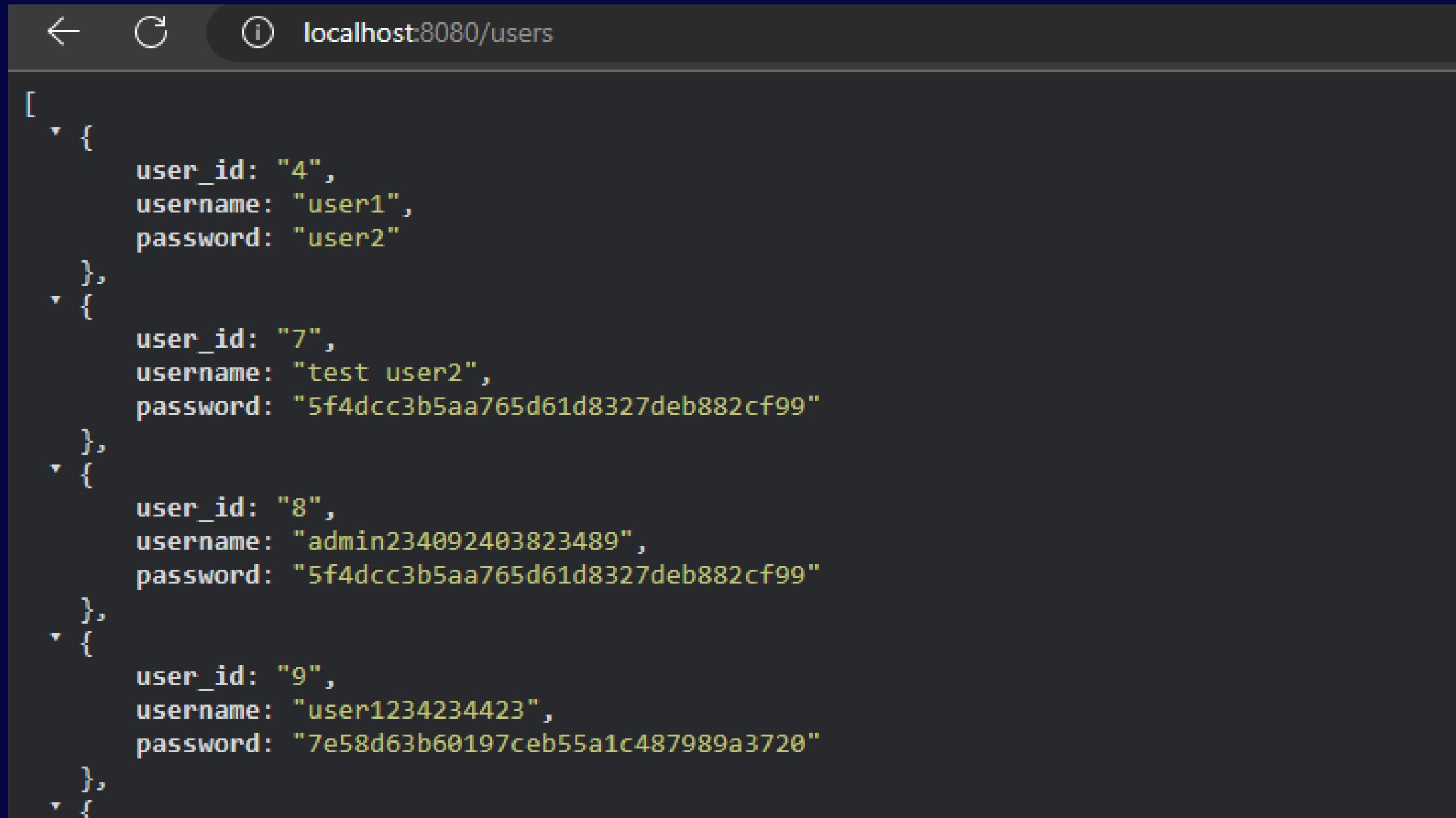
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

[powershell]

```
PS C:\Laragon\www\mardirabootcamp> cd public
PS C:\Laragon\www\mardirabootcamp\public> php -S localhost:8080
```

Langkah terakhir adalah menjalankan aplikasi dengan perintah cd (change directory) ke folder public dan perintah php -S localhost:8080

# Testing Aplikasi

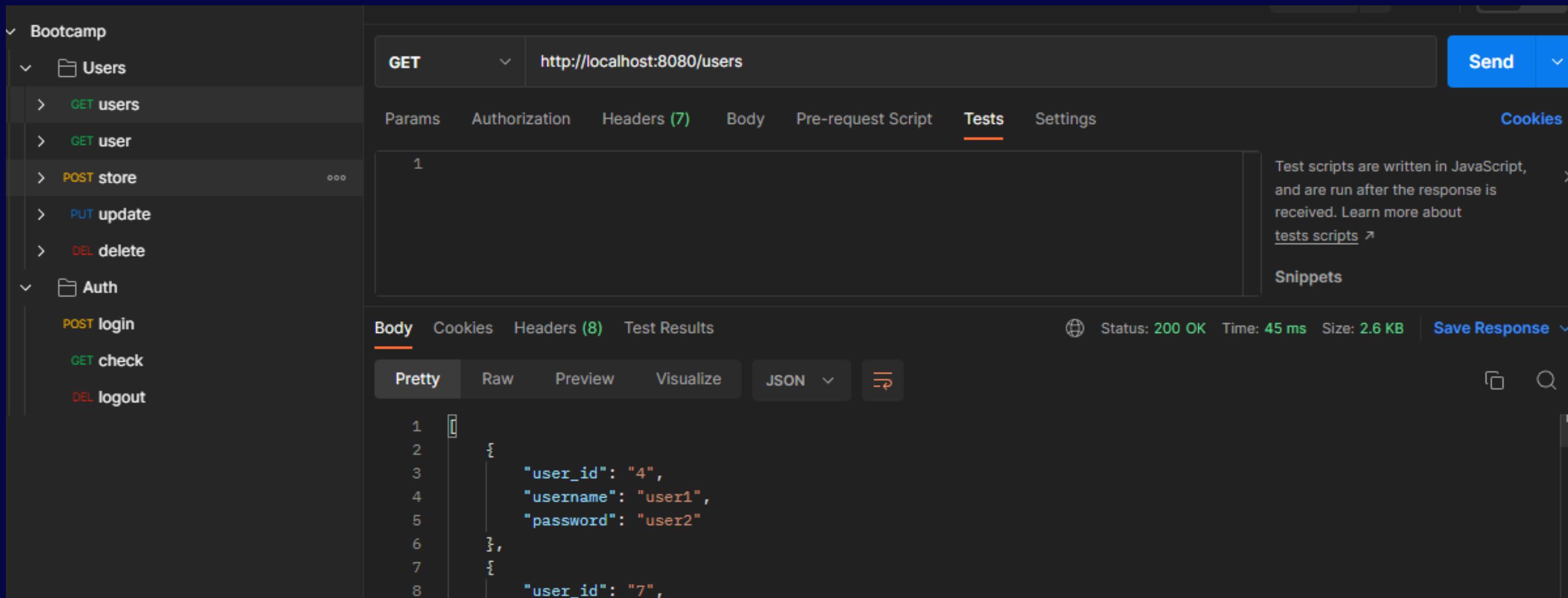


A screenshot of a terminal window with a dark background. The title bar shows the URL `localhost:8080/users`. The main area displays a JSON array of user objects:

```
[{"user_id": "4", "username": "user1", "password": "user2"}, {"user_id": "7", "username": "test user2", "password": "5f4dcc3b5aa765d61d8327deb882cf99"}, {"user_id": "8", "username": "admin234092403823489", "password": "5f4dcc3b5aa765d61d8327deb882cf99"}, {"user_id": "9", "username": "user1234234423", "password": "7e58d63b60197ceb55a1c487989a3720"}]
```

Untuk melakukan testing masukkan url `localhost:8080/users` untuk melihat data users.

# Testing Aplikasi Menggunakan Postman



The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API collections: 'Bootcamp' is expanded, showing 'Users' (with 'GET users', 'GET user', 'POST store', 'PUT update', 'DEL delete') and 'Auth' (with 'POST login', 'GET check', 'DEL logout'). The main area shows a request configuration for a 'GET' request to 'http://localhost:8080/users'. The 'Tests' tab is selected, containing the following JavaScript code:

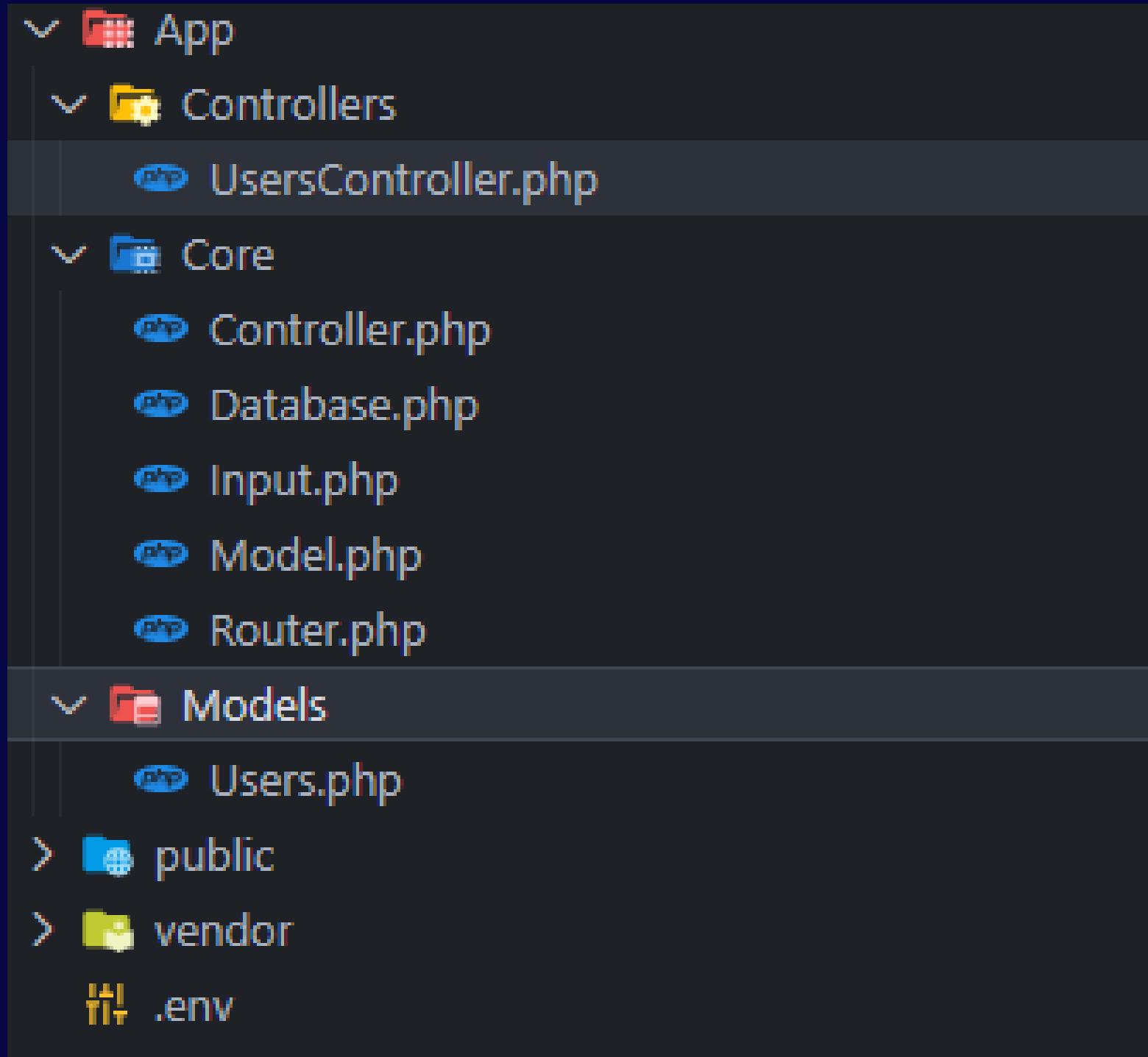
```
1
```

Below the request configuration, the response details are shown: Status: 200 OK, Time: 45 ms, Size: 2.6 KB. The 'Body' tab is selected, displaying the JSON response in 'Pretty' format:

```
1 [ {  
2   "user_id": "4",  
3   "username": "user1",  
4   "password": "user2"  
5 },  
6   {  
7     "user_id": "7",  
8   } ]
```

Testing aplikasi bisa diterapkan menggunakan Postman.

# Struktur Folder Framework Selengkapnya



Struktur folder pembuatan framework PHP sederhana berbasis Model Controller (MC) selengkapnya ada di tautan dibawah ini:

**Terima kasih**