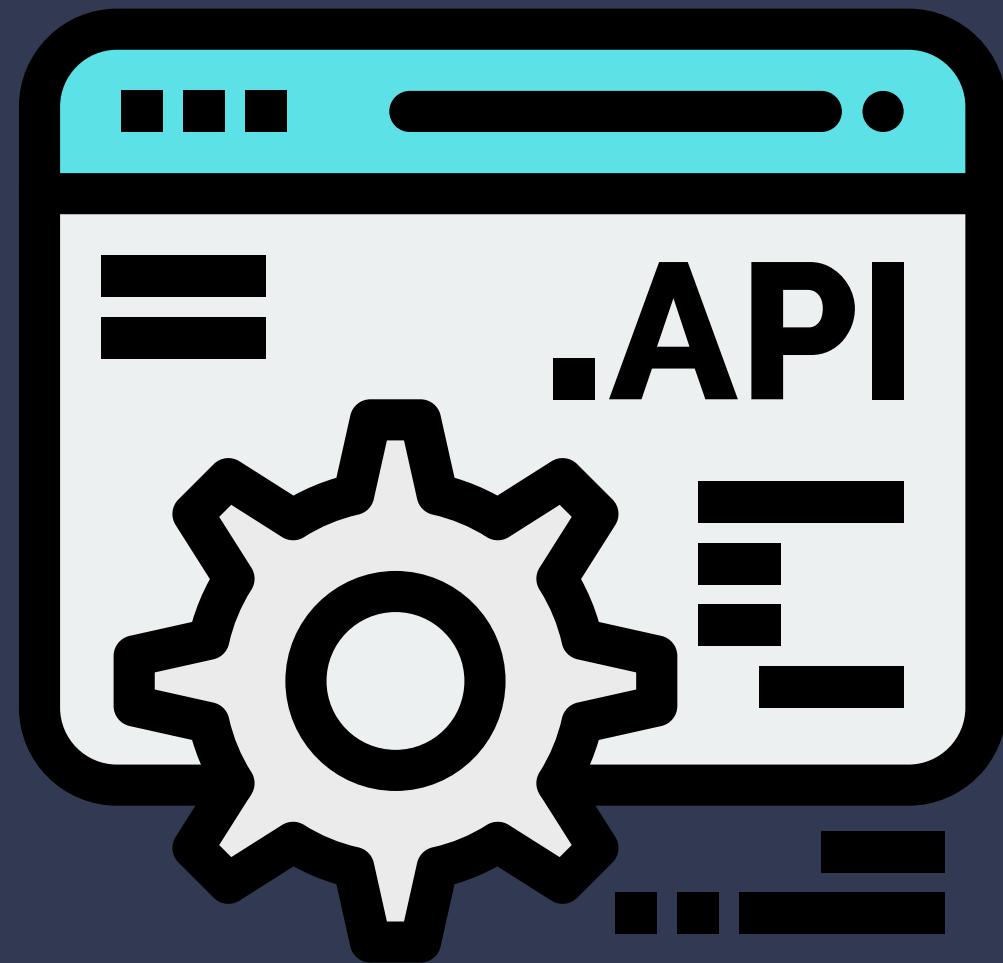


Jefri Maruli

Back-End Development

Materi 5



Agenda Belajar

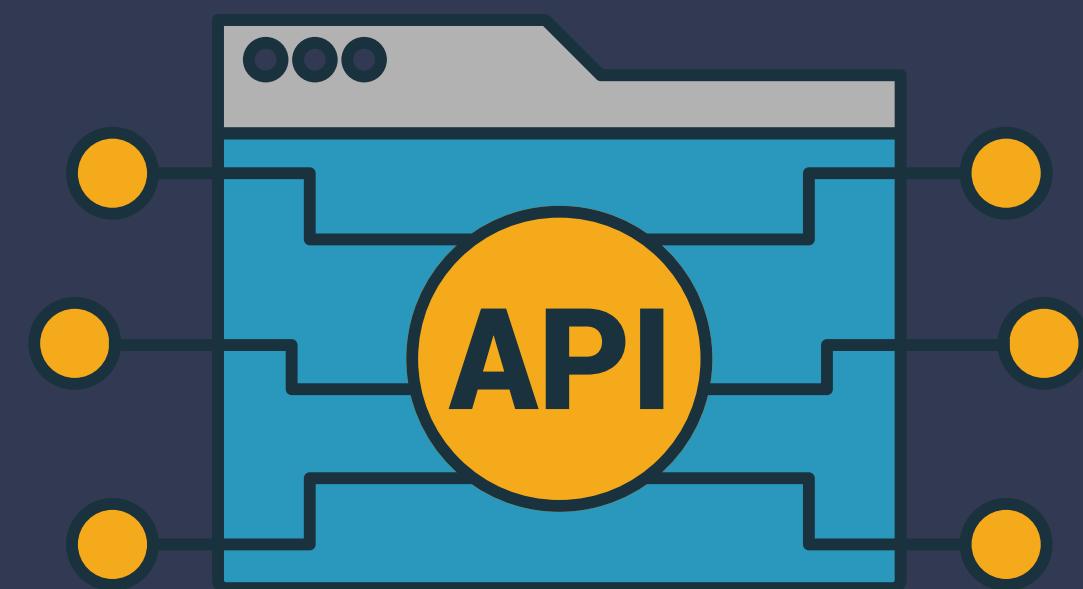
- REST API (Representational State Transfer Application Programming Interface)

referensi : <https://belajaraplikasi.com/membuat-rest-api-dengan-php-dan-mysql>

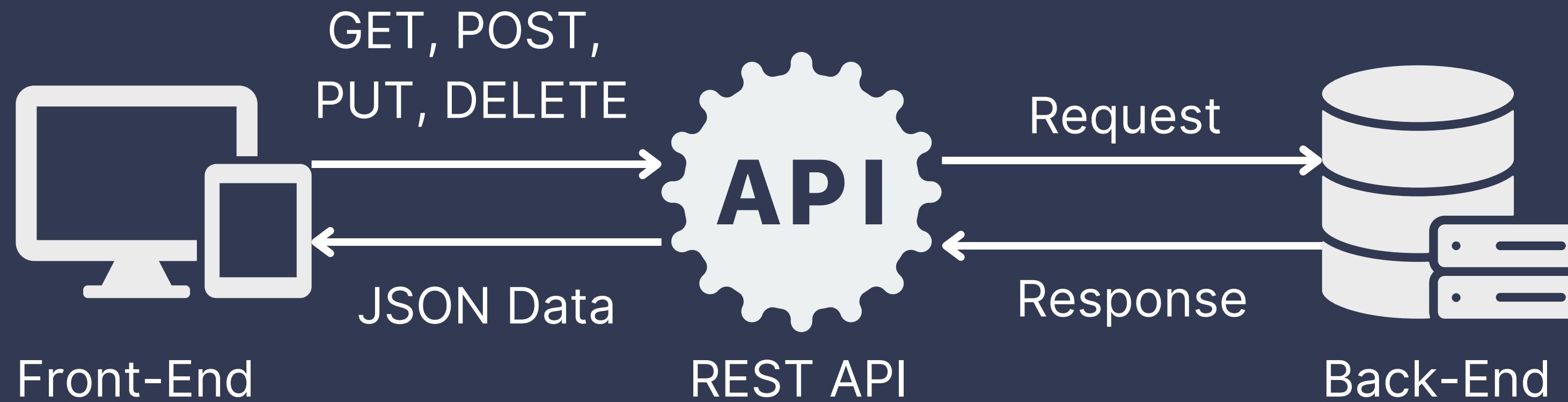


REST API

REST API (Representational State Transfer Application Programming Interface) merupakan implementasi dari API yang memiliki suatu arsitektur metode komunikasi dengan menggunakan protokol HTTP untuk pertukaran Data. Metode ini sering diterapkan dalam pengembangan aplikasi yang bertujuan untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk dikembangkan terutama dalam pertukaran dan komunikasi data.

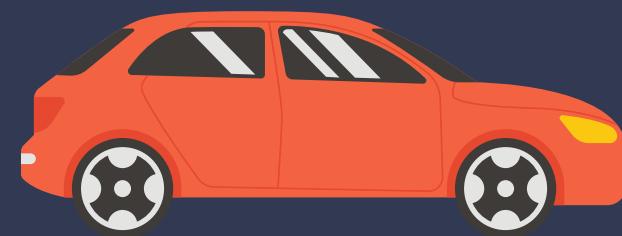


Konsep REST API



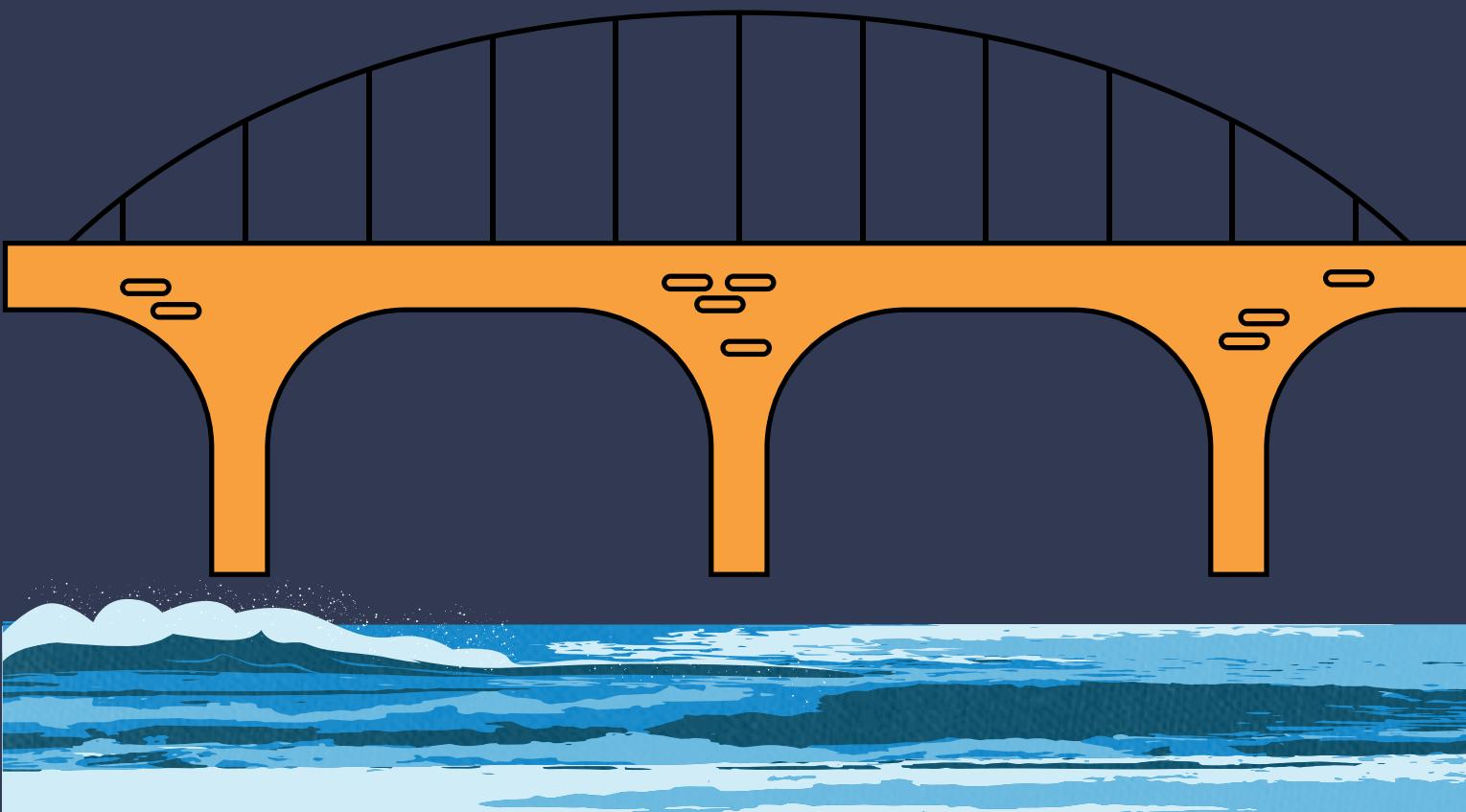
Analogi REST API

Kota A (FE)



Mobil (Request)

Jembatan (API)



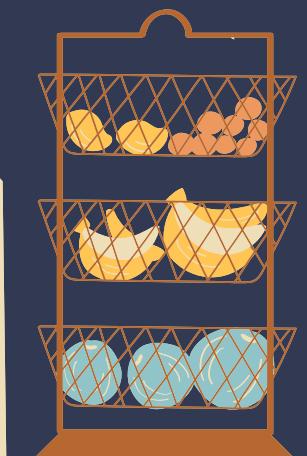
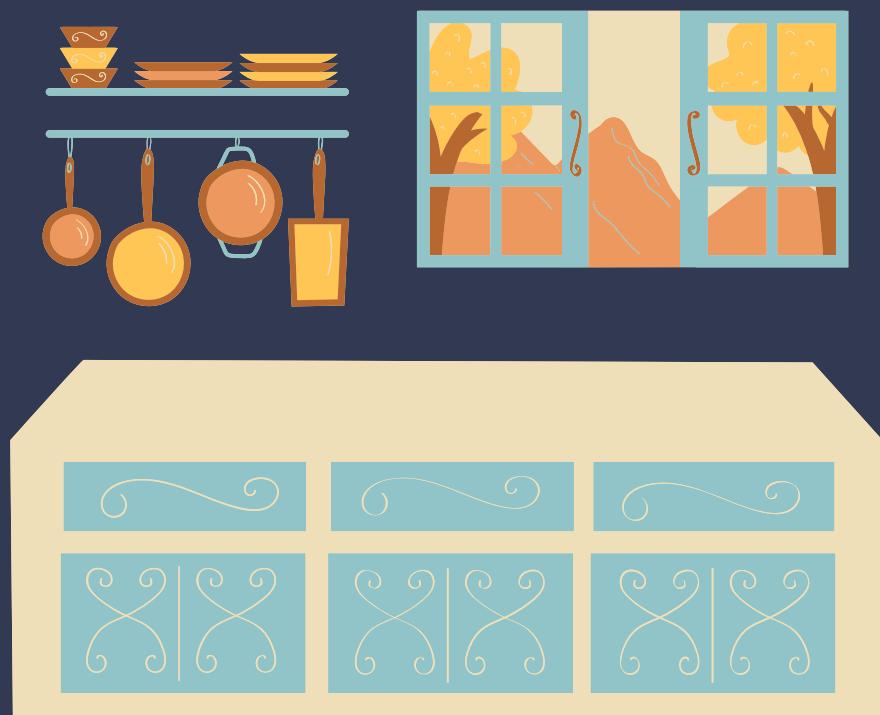
Kota B (BE)



Analogi Development API

Koki

Kompor
(Server)



Pelayan
(FE Developer)

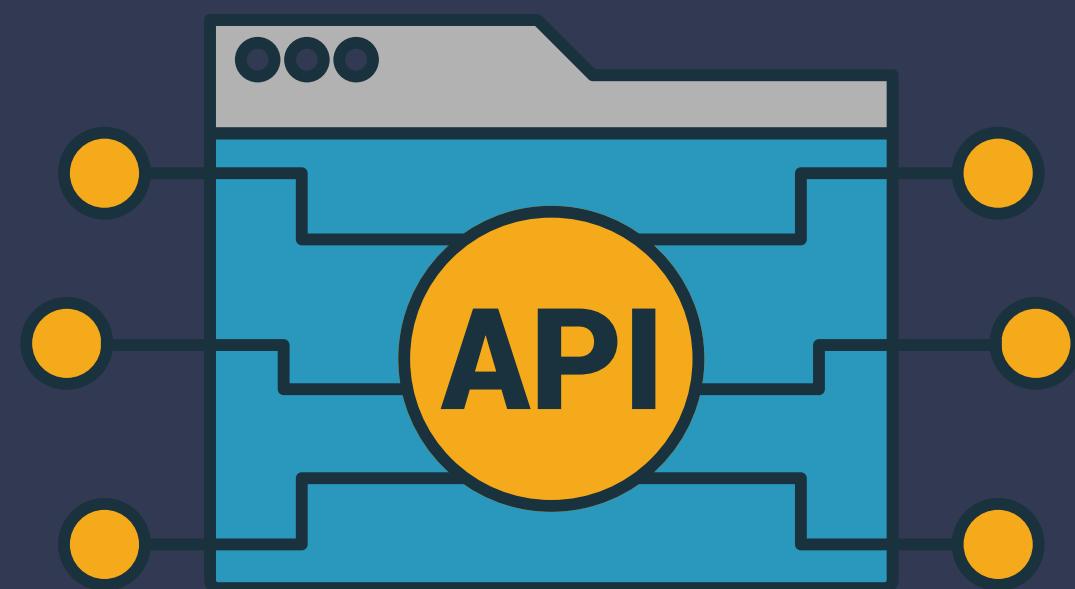


```
1  {
2    "name": "Sandwich",
3    "type": "Hidangan yang terbuat dari dua potong roti atau lebih dengan satu atau lebih isian",
4    "varieties": [
5      {
6        "name": "Grilled Cheese",
7        "description": "Sandwich yang terbuat dari satu atau lebih jenis keju di atas roti yang biasanya diolesi mentega dan dipanggang"
8      },
9      {
10        "name": "Philly Cheesesteak",
11        "description": "Sandwich yang terbuat dari steak yang diiris tipis dan keju leleh di atas gulungan panjang"
12      }
13    ]
14 }
```

Komponen REST API

REST API memiliki komponen penting sebagai syarat utama mengimplementasikan REST API, dan akan disebut RESTful API jika melakukan semua komponen yang ada di bawah ini:

- URL Design
- HTTP Verbs
- HTTP Response Code
- Format Response



Perbedaan API , REST API & RESTful API

Komponen	API	REST API	RESTful API
URL Design	Tidak	Tidak	Ya
HTTP Verbs	Tidak	Tidak	Ya
HTTP Response Code	Tidak	Ya	Ya
Format Response	Ya	Ya	Ya

URL Design

REST API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti developer. URL API biasa disebut endpoint dalam pemanggilannya. Contoh penamaan URL / endpoint yang baik adalah seperti berikut:

/users

/users/1234

/users/store/

/users/update

`http://localhost/mardirabootcamp/users`

Perbandingan URL Design

Tidak Menggunakan URL Design

```
http://localhost/mardirabootcamp/PDO/Users.php
```

Menggunakan URL Design

```
http://localhost/mardirabootcamp/users
```

HTTP Verbs

Setiap request yang dilakukan terdapat metode yang dipakai agar server mengerti apa yang sedang di request client atau dari front-end, diantaranya yang umum dipakai adalah:

HTTP	Deskripsi	Contoh
GET	GET adalah metode HTTP Request yang paling simpel, metode ini digunakan untuk membaca atau mendapatkan data dari sumber	/users
POST	POST adalah metode HTTP Request yang digunakan untuk membuat data baru dengan menyisipkan data dalam body saat request dilakukan.	/users/store
PUT	PUT adalah metode HTTP Request yang biasanya digunakan untuk melakukan update data resource.	/users/update/:id
DELETE	DELETE adalah metode HTTP Request yang digunakan untuk menghapus suatu data pada resource.	/users/delete/:id

HTTP Response Code

HTTP response code adalah kode standarisasi dalam menginformasikan hasil request kepada client / Front-End. Secara umum terdapat 3 kelompok yang biasa kita jumpai pada RESTful API yaitu:

- 2xx : adalah response code yang menampilkan bahwa request berhasil
- 4xx : adalah response code yang menampilkan bahwa request mengalami kesalahan pada sisi client / front-end
- 5xx : adalah response code yang menampilkan bahwa kesalahan pada sisi server / back-end

Contoh HTTP Response Code

-  **200 (OK)** : Response code berhasil.
-  **201 (Created)** : Response code data telah berhasil dibuat (POST/PUT).
-  **400 (Bad Request)** : Response code request yang dibuat salah / tidak ada.
-  **401 (Unauthorized)** : Response code request yang dibuat membutuhkan authentication (login) sebelum mengakses endpoint tersebut.
-  **403 (Forbidden)** : Response code yang bukan memiliki hak untuk mengakses endpoint tersebut meskipun sudah melakukan login.
-  **404 (Not Found)** : Response code request yang dicari tidak ditemukan.
-  **405 (Method Not Allowed)** : Response code request endpoint ada tetapi HTTP verbs yang digunakan tidak diizinkan, contoh : endpoint yang dibuat menggunakan HTTP GET tetapi saat di request menggunakan HTTP POST

Contoh HTTP Response Code



- **409 (Conflict)** : Response code request yang dibuat terdapat duplikasi, biasanya data yang dikirim sudah ada sebelumnya.



- **500 (Internal Server Error)** : Response code ini menandakan bahwa request yang dilakukan terdapat kesalahan pada sisi server atau resource.



- **503 (Service Temporarily Unavailable)** : Response code untuk menandakan bahwa server sedang mengalami perbaikan (maintenance).

Referensi status code : https://id.wikipedia.org/wiki/Daftar_kode_status_HTTP

Format Response

Setiap request yang dilakukan client akan menerima data response dari server, response tersebut biasanya berupa data XML (eXtensible Markup Language) ataupun JSON (JavaScript Object Notation). Setelah mendapatkan data response tersebut barulah client bisa menggunakannya dengan cara parsing data tersebut dan diolah sesuai dengan kebutuhan.



Format JSON

```
1  {
2    "data": [
3      {
4        "id": 1,
5        "name": "John Doe",
6        "email": "johndoe@gmail.com"
7      },
8      {
9        "id": 2,
10       "name": "Jane Doe",
11       "email": "janedoe@gmail.com"
12     }
13   ]
14 }
```



Format XML

```
1 <users>
2   <user>
3     <id>1</id>
4     <name>John Doe</name>
5     <email>johndoe@gmail.com</email>
6   </user>
7   <user>
8     <id>2</id>
9     <name>Jane Doe</name>
10    <email>janedoe@gmail.com</email>
11  </user>
12 </users>
```

Pembuatan Format Response API



XML

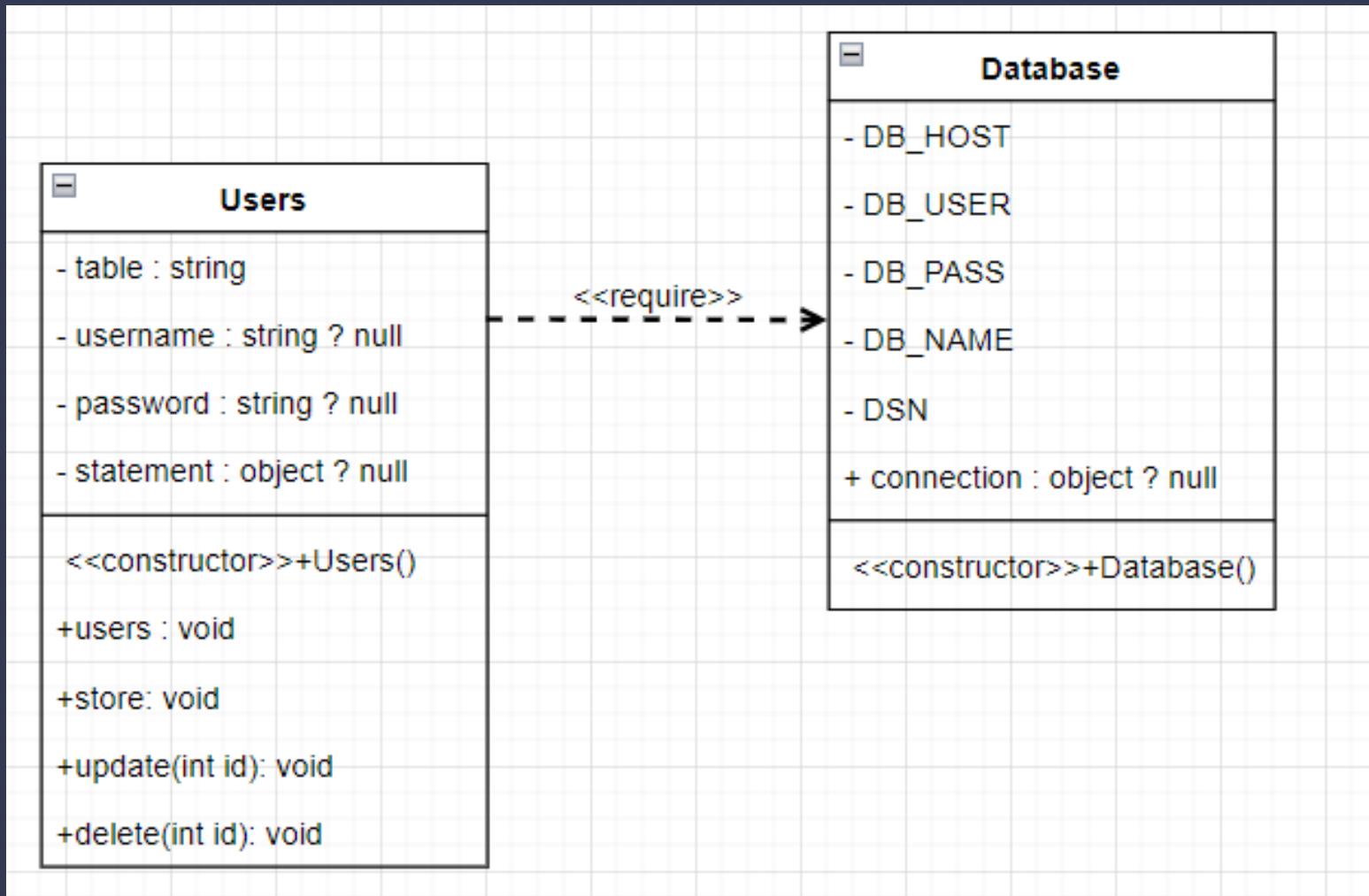
```
1 <?php
2 $users = [
3     [
4         'id' => 1,
5         'name' => 'John Doe',
6         'email' => 'johndoe@gmail.com'
7     ],
8     [
9         'id' => 2,
10        'name' => 'Jane Smith',
11        'email' => 'janedoe@gmail.com'
12    ]
13];
14
15 function array_to_xml($array, &$xml_user_info)
16 {
17     foreach ($array as $key => $value) {
18         if (is_array($value)) {
19             if (!is_numeric($key)) {
20                 $subnode = $xml_user_info->addChild("$key");
21                 array_to_xml($value, $subnode);
22             } else {
23                 $subnode = $xml_user_info->addChild("$key");
24                 array_to_xml($value, $subnode);
25             }
26         } else {
27             $xml_user_info->addChild("$key", htmlspecialchars("$value"));
28         }
29     }
30 }
31
32 $xml = new SimpleXMLElement("<?xml version=\"1.0\"?><users></users>");
33
34 //function call to convert array to xml
35 array_to_xml($users,$xml);
36
37 echo $xml->asXML();
38
```



JSON

```
1 <?php
2 $users = [
3     [
4         'id' => 1,
5         'name' => 'John Doe',
6         'email' => 'johndoe@gmail.com'
7     ],
8     [
9         'id' => 2,
10        'name' => 'Jane Smith',
11        'email' => 'janedoe@gmail.com'
12    ]
13];
14
15 echo json_encode($users);
```

Implementasi RESTful API



Untuk mengimplementasi RESTful API, kita tetap menggunakan studi kasus pada materi sebelumnya yang sudah dirancang. Tugas utama untuk membuat RESTful API adalah:

- URL Design
- HTTP Verbs
- HTTP Response Code
- Format Response

No	Field	Type	Size	Index
1	user_id	int	11	Primary Key
2	username	varchar	25	
3	password	varchar	64	

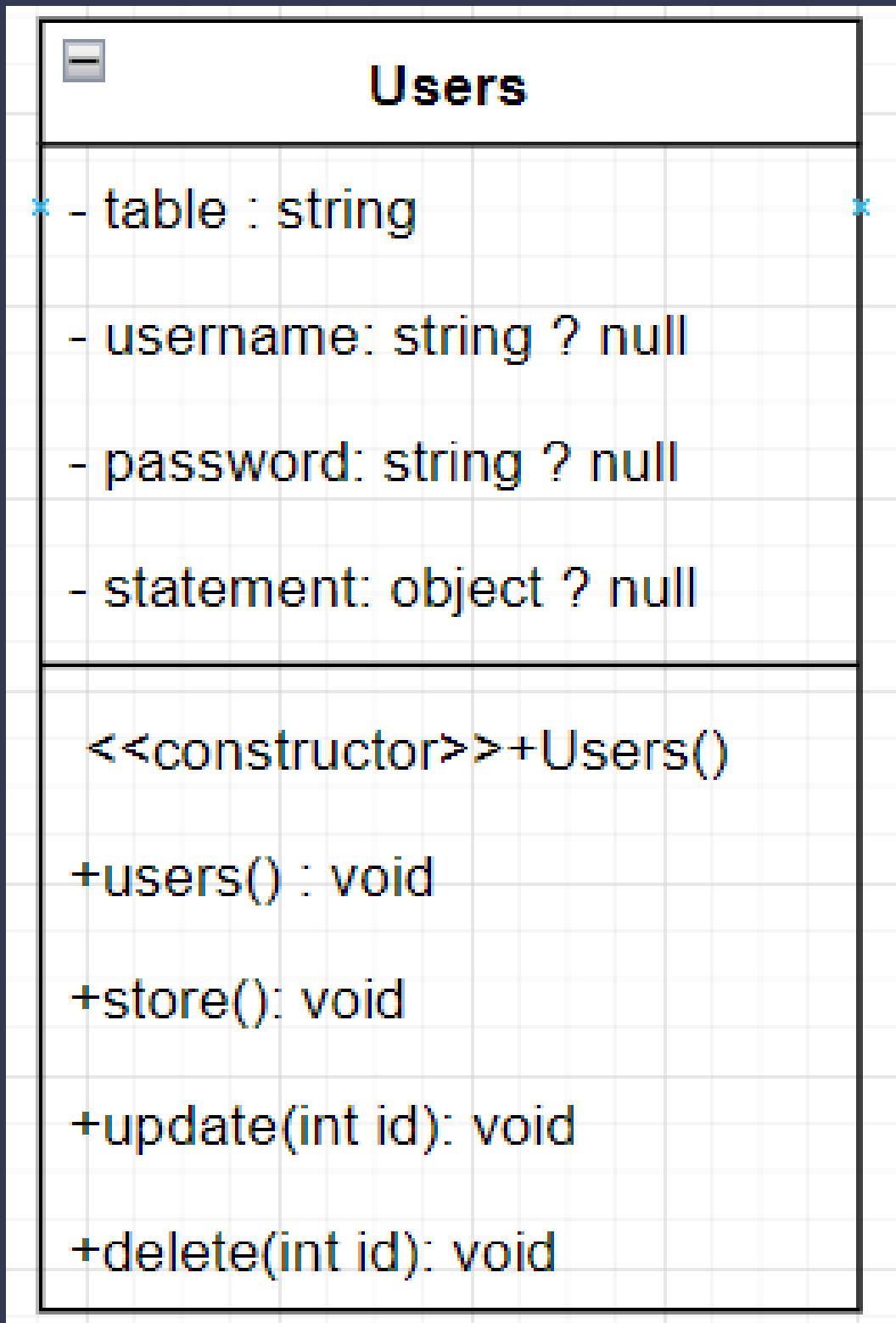
Penerapan Koneksi

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

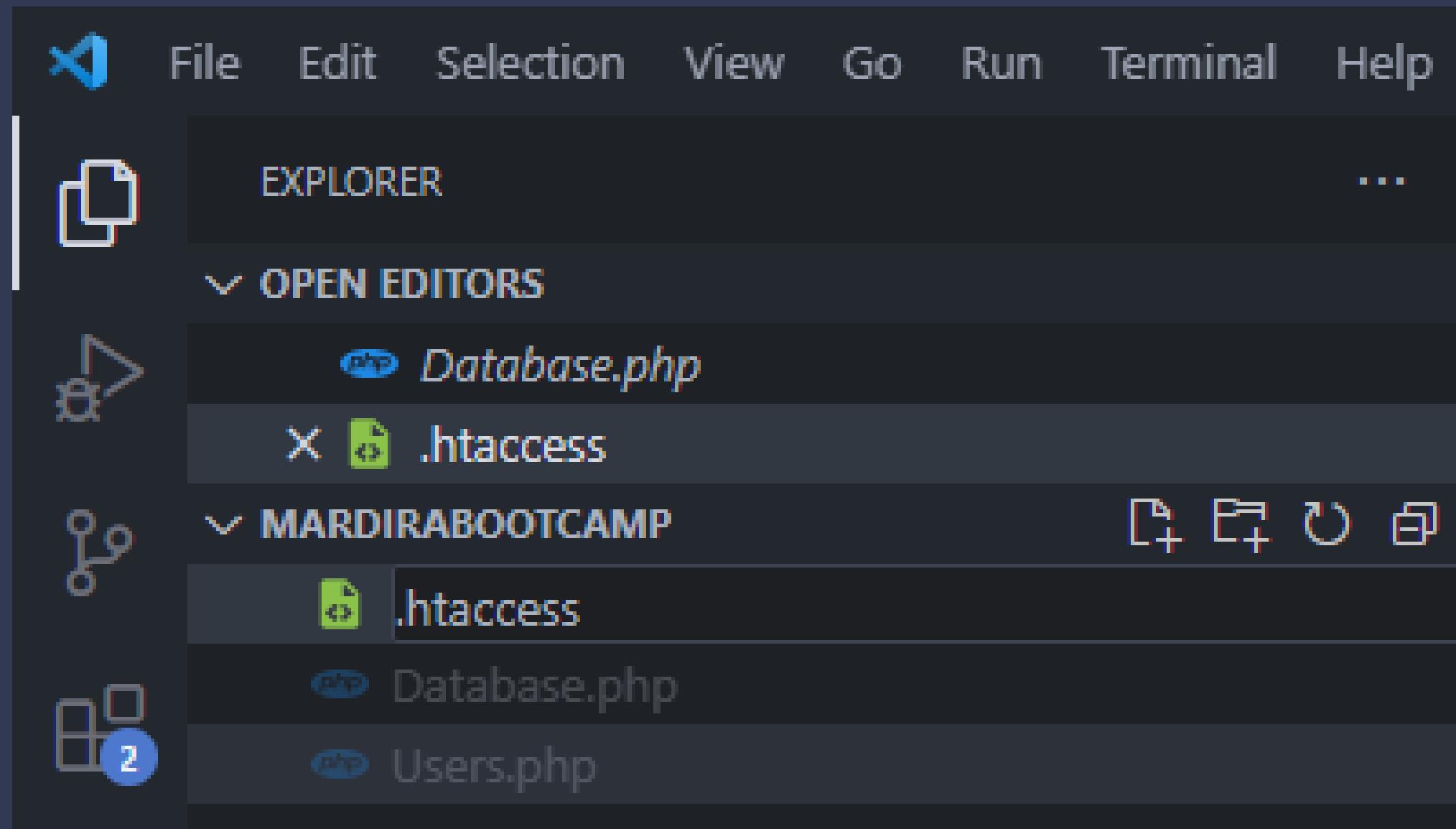
Penerapan Class Users Dari Class Diagram

```
<?php  
require_once('Database.php');  
  
class Users  
{  
    private string $table = 'users';  
    private ?object $statement;  
    private ?string $username;  
    private ?string $password;  
  
    public function __construct()  
    {  
        $this->statement = new Database();  
        $this->statement = $this->statement->connection;  
        $this->username = $_POST['username'] ?? null;  
        $this->password = $_POST['password'] ?? null;  
    }  
  
    public function users(): void  
    {  
        $query = "SELECT * FROM $this->table";  
        $statement = $this->statement->prepare($query);  
        $statement->execute();  
        $users = $statement->fetchAll(PDO::FETCH_OBJ);  
        $total = $statement->rowCount();  
        $response = [  
            'message' => 'Data User',  
            'total' => $total,  
            'data' => $users  
        ];  
        echo json_encode($response, JSON_PRETTY_PRINT);  
    }  
  
    public function store(): void  
    {  
        $query = "SELECT * FROM {$this->table} WHERE username = :username";  
        $statement = $this->statement->prepare($query);  
        $statement->bindParam(':username', $this->username);  
        $statement->execute();  
        if ($statement->rowCount() > 0) {  
            $response = [  
                'message' => 'Data User Sudah Ada!',  
            ];  
            http_response_code(409);  
        } else {  
            $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";  
            $statement = $this->statement->prepare($query);  
            $password = md5($this->password);  
            $statement->bindParam(':username', $this->username);  
            $statement->bindParam(':password', $password);  
            $statement->execute();  
            $getUsers = "SELECT * FROM users WHERE username = :username";  
            $statement = $this->statement->prepare($getUsers);  
            $statement->bindParam(':username', $this->username);  
            $statement->execute();  
            $user = $statement->fetch(PDO::FETCH_OBJ);  
            $response = [  
                'message' => 'Data User Berhasil Disimpan',  
                'user' => $user,  
            ];  
        }  
        echo json_encode($response);  
    }  
  
    public function update(int $id): void  
    {  
        $query = "UPDATE {$this->table} SET username = :username, password = :password WHERE user_id = :user_id";  
        $statement = $this->statement->prepare($query);  
        $password = md5($this->password);  
        $statement->bindParam(':username', $this->username);  
        $statement->bindParam(':password', $this->password);  
        $statement->bindParam(':user_id', $id);  
        $statement->execute();  
        $response = [  
            'message' => 'Data User Berhasil Diubah',  
        ];  
        echo json_encode($response);  
    }  
  
    public function delete(int $id): void  
    {  
        $query = "DELETE FROM {$this->table} WHERE user_id = :user_id";  
        $statement = $this->statement->prepare($query);  
        $statement->bindParam(':user_id', $id);  
        $statement->execute();  
        $response = [  
            'message' => 'Data User Berhasil Dihapus',  
        ];  
        echo json_encode($response);  
    }  
  
    $users = new Users();  
    $users->users();  
}
```

Class Users secara keseluruhan dari class diagram



Menerapkan URL Design Dengan .htaccess



Todo

- URL Design
- HTTP Verbs
- HTTP Response Code
- Format Response

Tambahkan file .htaccess atau hypertext access untuk mengatur bagaimana mengatur sebuah script php yang berada dalam sebuah server/hosting

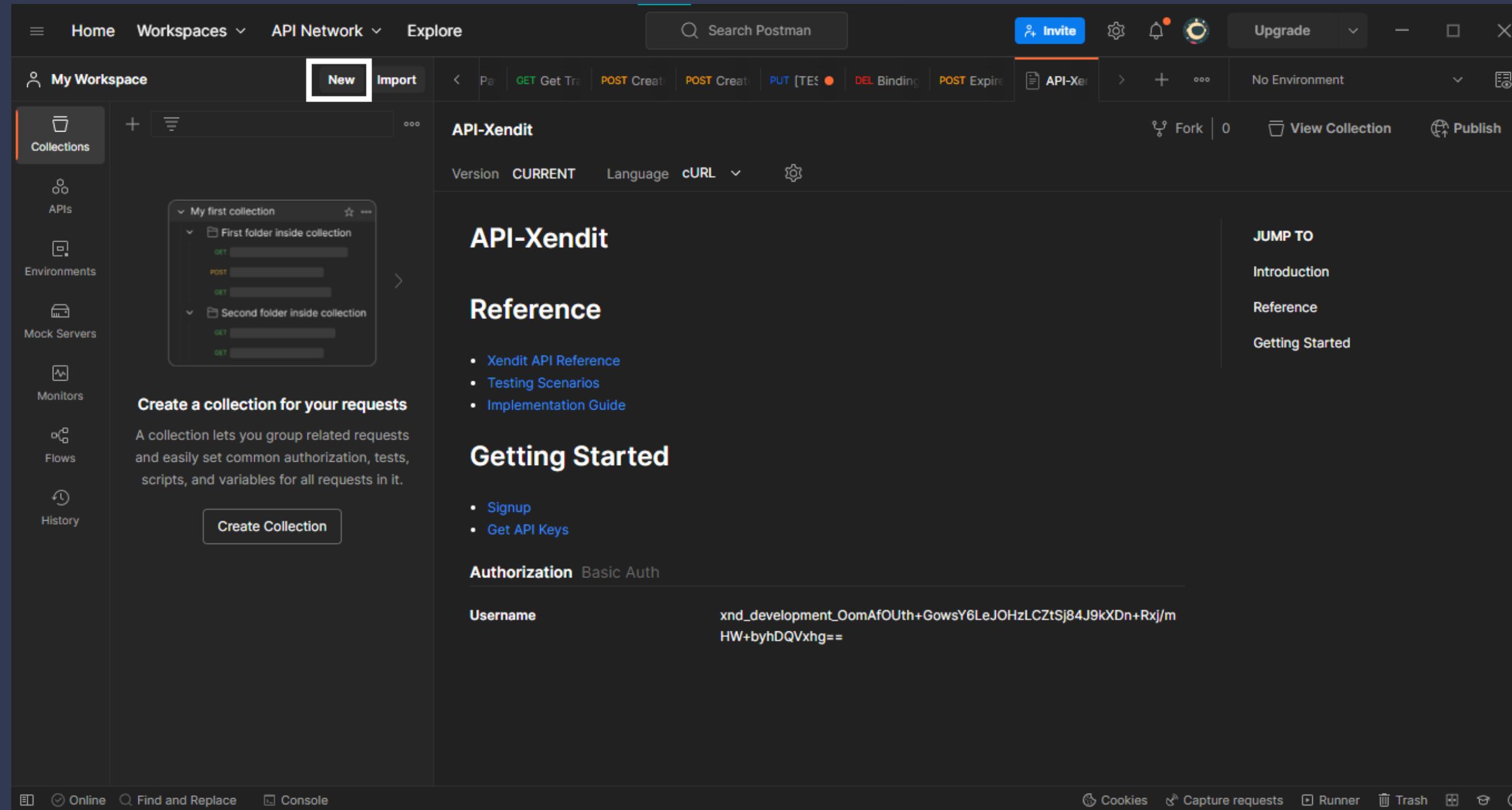
Isi File .htaccess



```
1 RewriteEngine On # Mengaktifkan modul mod_rewrite
2 # Mengarahkan URL ke file Users.php
3 RewriteRule ^users/?$ Users.php [NC,L]
4 # Mengarahkan URL ke file Users.php dengan parameter id
5 RewriteRule ^Users/([0-9]+)/?$ users.php?id=$1 [NC,L]
```

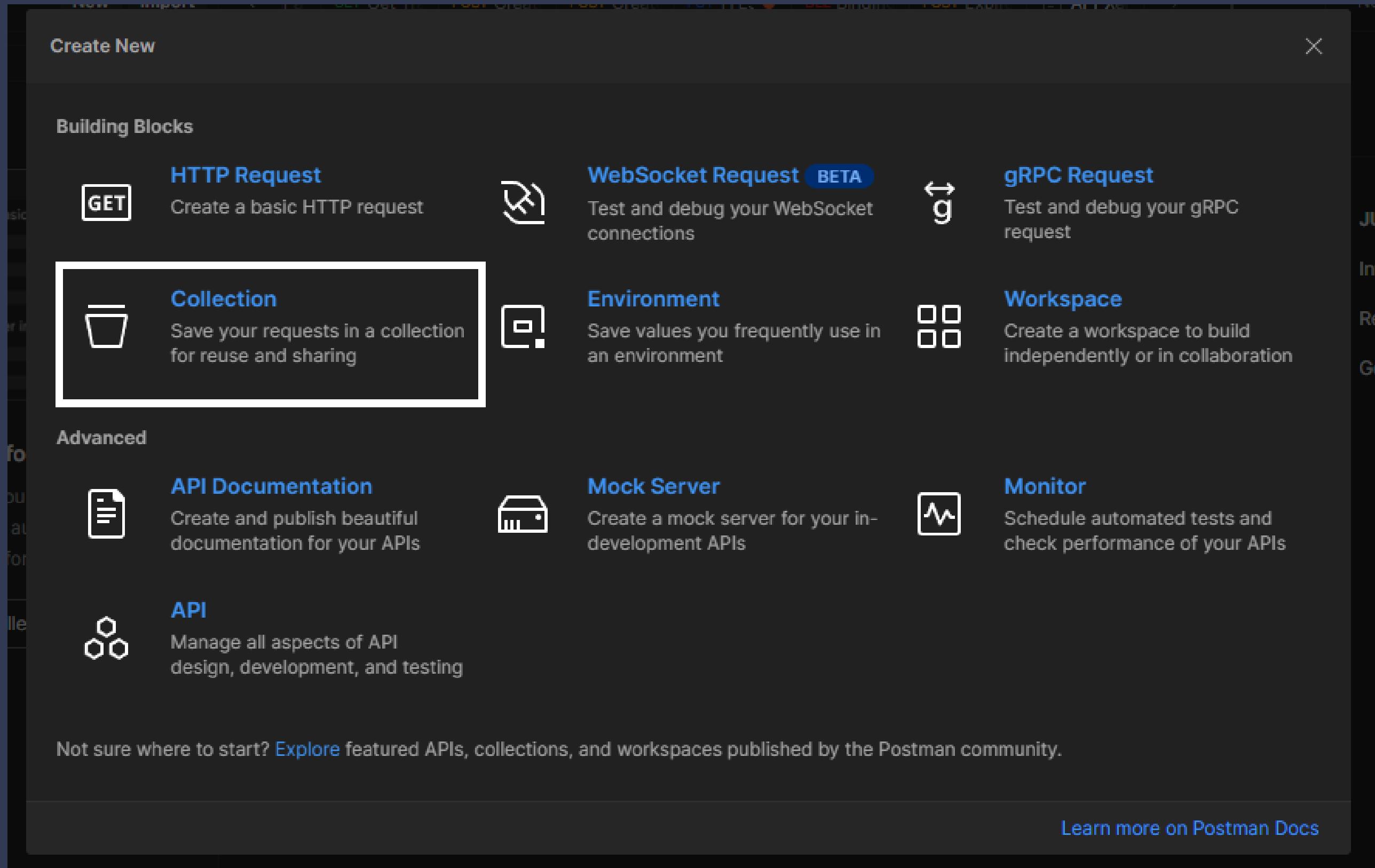
Tambahkan baris kode di atas untuk file .htaccess supaya bisa menggunakan clean URL design pada REST API.

Menggunakan Postman



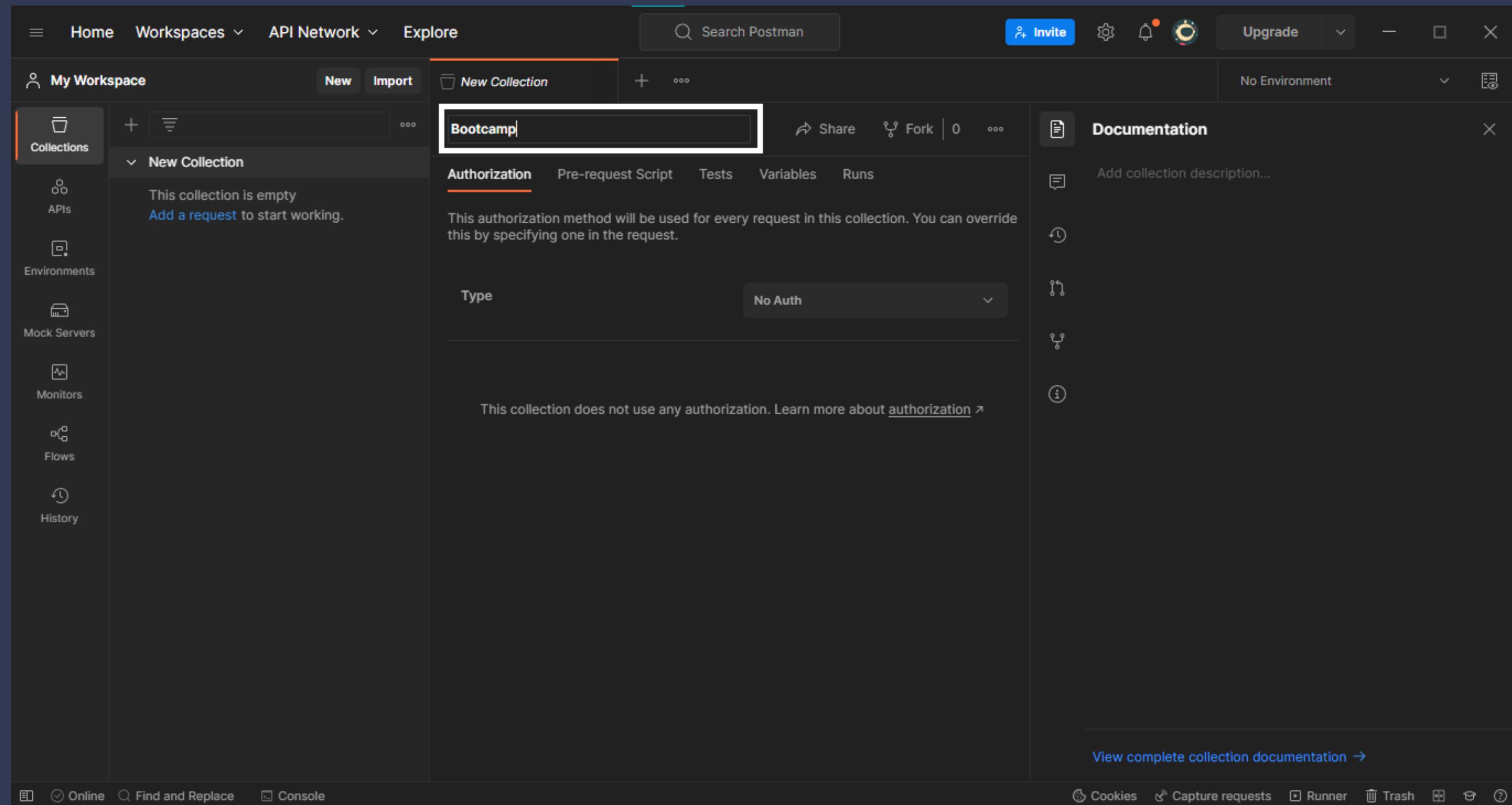
Kita akan menggunakan Postman untuk melakukan testing API yang telah dibuat, pertama setelah membuka Postman pilih tombol new

Menambahkan Collection



Setelah memilih tombol new kemudian pilih "Collection".

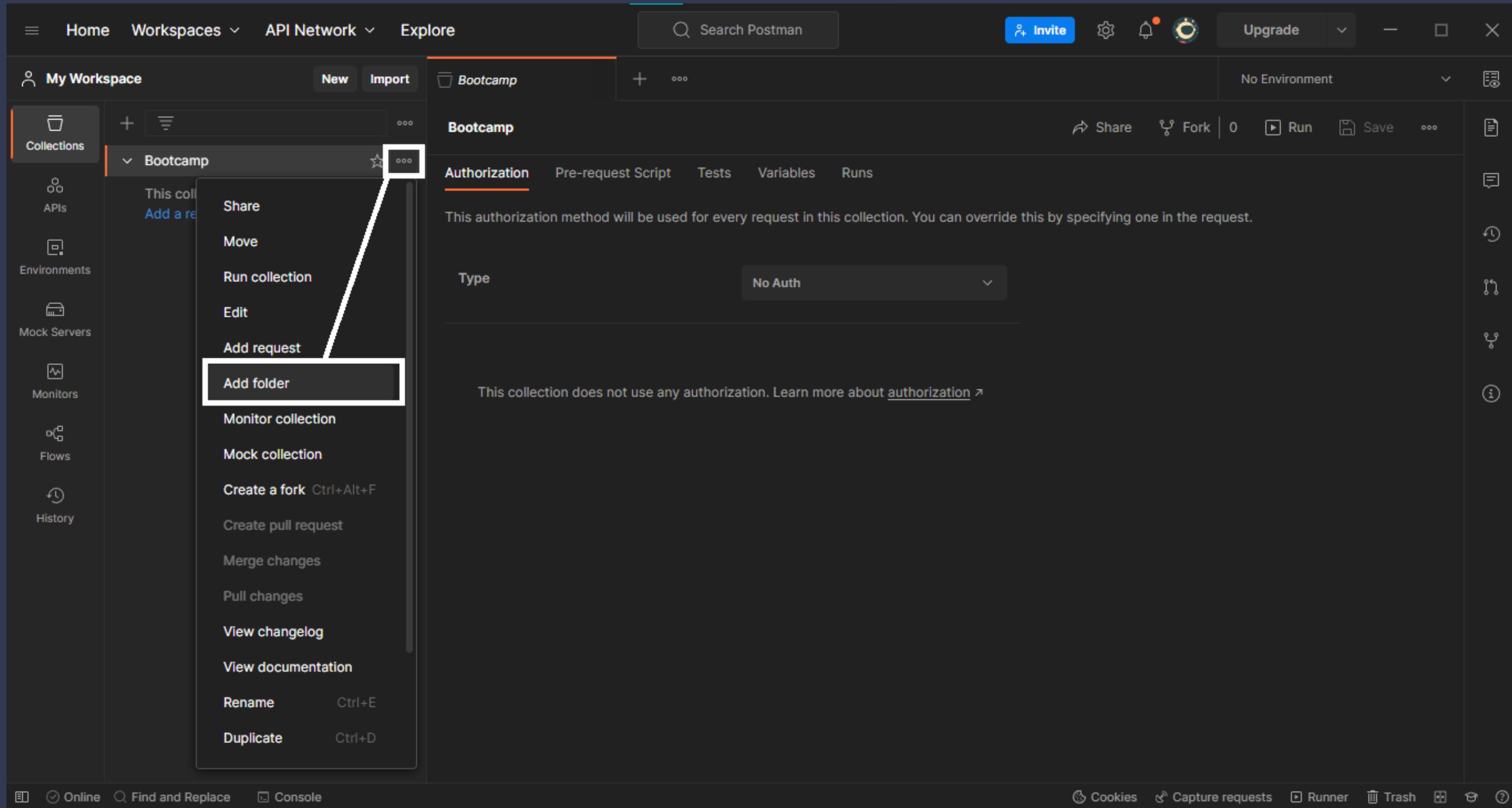
Mengubah Nama Collection



The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'API Network', and 'Explore'. A search bar says 'Search Postman' and there are 'Invite', 'Settings', 'Notifications', and 'Upgrade' buttons. Below the navigation is a header for 'My Workspace' with 'New' and 'Import' buttons, and a 'New Collection' button. A collection titled 'Bootcamp' is selected, highlighted with a red border. To the right of the collection title are 'Share', 'Fork', and 'Runs' buttons. The main workspace shows a 'New Collection' section with the message 'This collection is empty' and a link 'Add a request to start working.' Below this is an 'Authorization' tab, which is currently active, showing 'No Auth' selected from a dropdown. A note below the tab says 'This collection does not use any authorization. Learn more about [authorization](#)'. On the right side, there's a 'Documentation' panel with a 'Description' field containing 'Add collection description...' and several icons for sharing, forks, and information. At the bottom, there are buttons for 'Online', 'Find and Replace', 'Console', 'Cookies', 'Capture requests', 'Runner', 'Trash', and help icons.

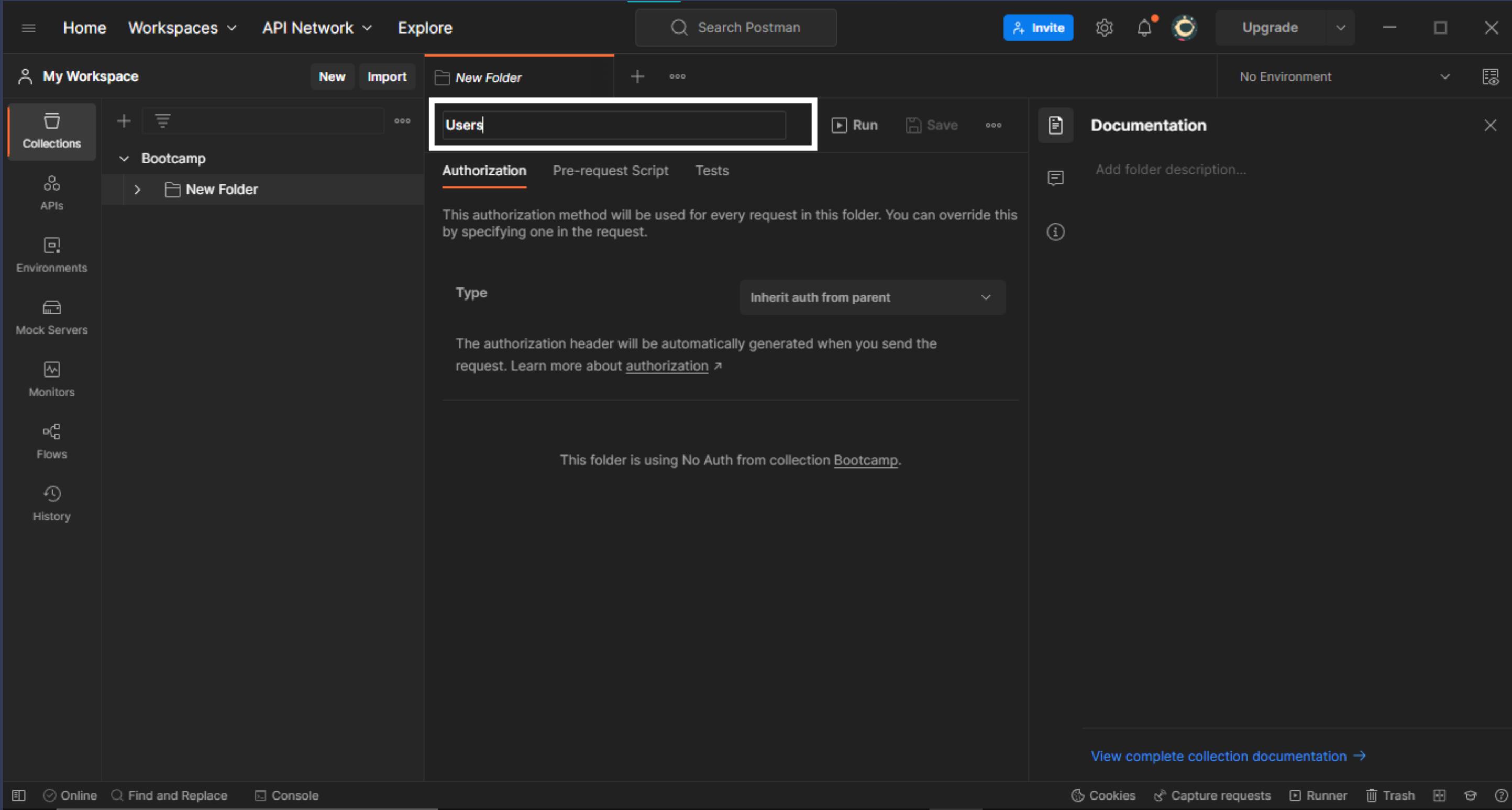
Selanjutnya rename nama collection menjadi "Bootcamp".

Menambahkan Folder



Untuk menambahkan folder pilih tombol icon triple dot "...", setelah itu pilih "Add folder"

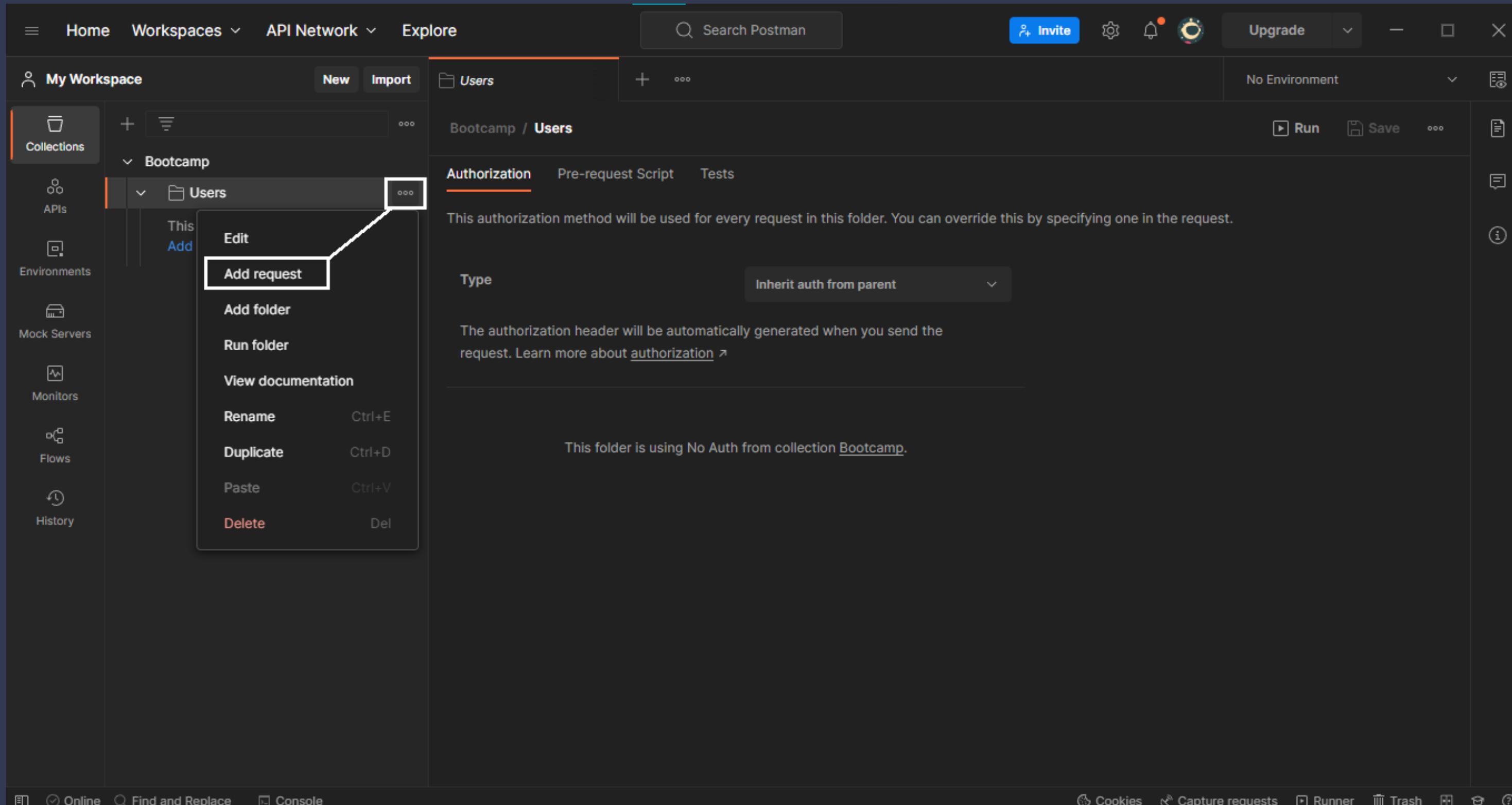
Mengubah Nama Folder



The screenshot shows the Postman application interface. At the top, there is a navigation bar with links for Home, Workspaces, API Network, and Explore. A search bar labeled "Search Postman" is also present. On the left side, there is a sidebar with icons for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace shows a "My Workspace" section with a "New Folder" button and a "Users" folder highlighted with a red border. The "Users" folder has a "Run", "Save", and "More" button. Below the folder, there are tabs for "Authorization", "Pre-request Script", and "Tests". A note states: "This authorization method will be used for every request in this folder. You can override this by specifying one in the request." Under the "Type" dropdown, it says "Inherit auth from parent". Another note below states: "The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)". A message at the bottom of the folder area says: "This folder is using No Auth from collection [Bootcamp](#)". To the right of the folder, there is a "Documentation" panel with a "Add folder description..." field and an "Info" icon. At the bottom of the interface, there are buttons for Online, Find and Replace, Console, Cookies, Capture requests, Runner, Trash, and Help.

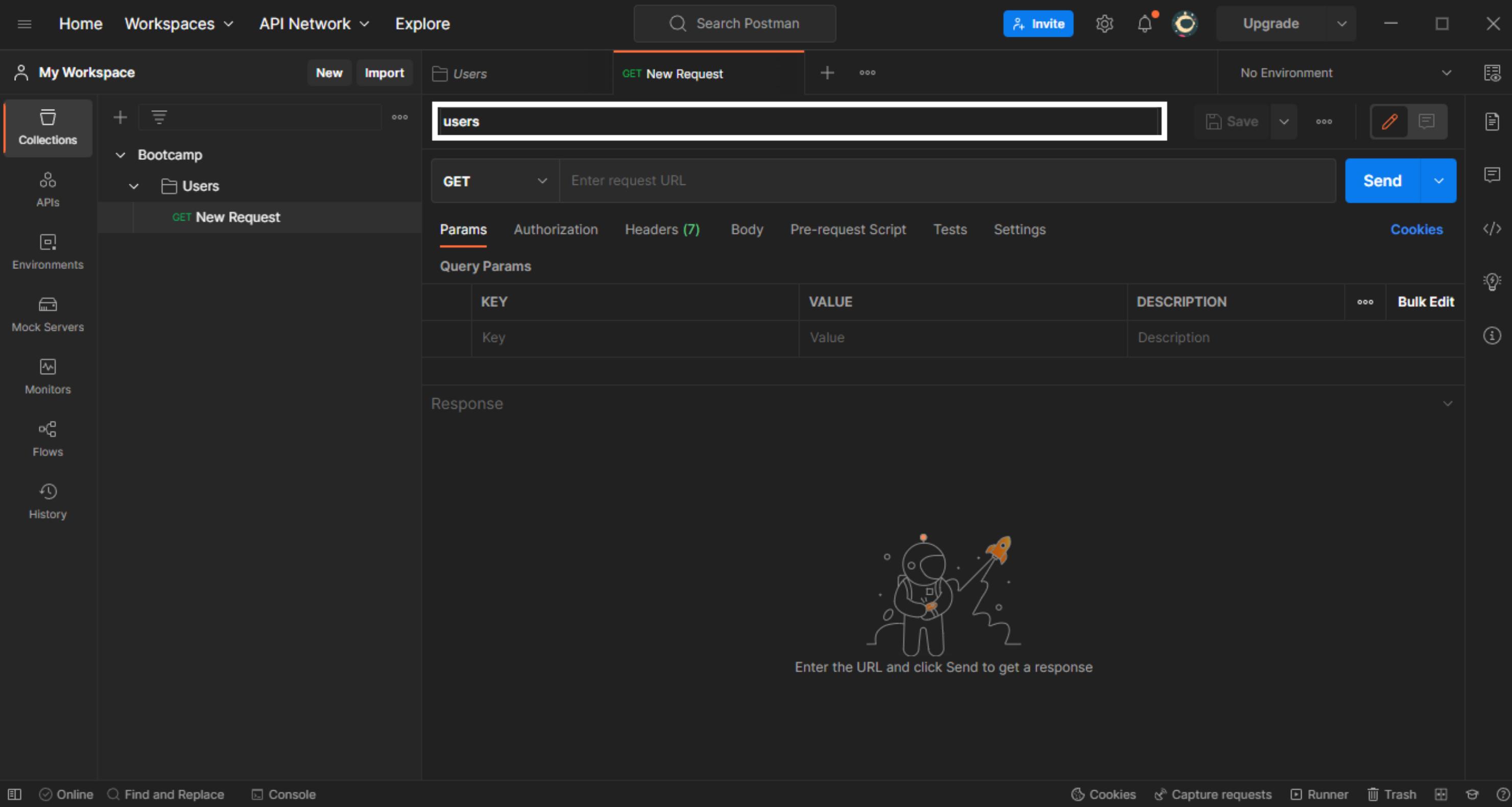
Rename folder sesuai dengan nama class yang telah dibuat yaitu "Users".

Menambahkan Request



Langkah selanjutnya yaitu menambahkan request atau disebut dengan endpoint pilih tombol icon triple dot "..." kemudian pilih "Add Request".

Mengubah Nama Request



The screenshot shows the Postman application interface. In the center, there is a 'New Request' dialog box with the title 'GET New Request'. The URL field contains 'users'. Below the URL field, the method is set to 'GET' and the 'Enter request URL' field is empty. To the right of the URL field is a 'Send' button. Above the URL field, the search bar says 'Search Postman' and has a magnifying glass icon. To the right of the search bar are several small icons: 'Invite', 'Settings', 'Notifications', and 'Profile'. Further to the right are 'Upgrade', a window control bar (minimize, maximize, close), and a 'X' button. On the left side of the interface is a sidebar titled 'My Workspace' with a 'Collections' tab selected. Under 'Collections', there is a 'Bootcamp' folder which contains a 'Users' folder. Inside the 'Users' folder, there is a 'GET New Request' item. Other items in the 'Users' folder include 'GET Get User', 'POST Create User', 'PUT Update User', and 'DELETE Delete User'. Below the 'Users' folder, there are other sections like 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. At the bottom of the interface, there are several status indicators: 'Online' (green), 'Find and Replace' (blue), 'Console' (grey), 'Cookies' (blue), 'Capture requests' (blue), 'Runner' (grey), 'Trash' (grey), and a question mark icon. A large, friendly cartoon character of an astronaut with a rocket on his back is positioned in the center of the interface, pointing upwards.

Untuk rename request sesuaikan dengan nama method yang telah dibuat

Menambahkan Endpoint URL

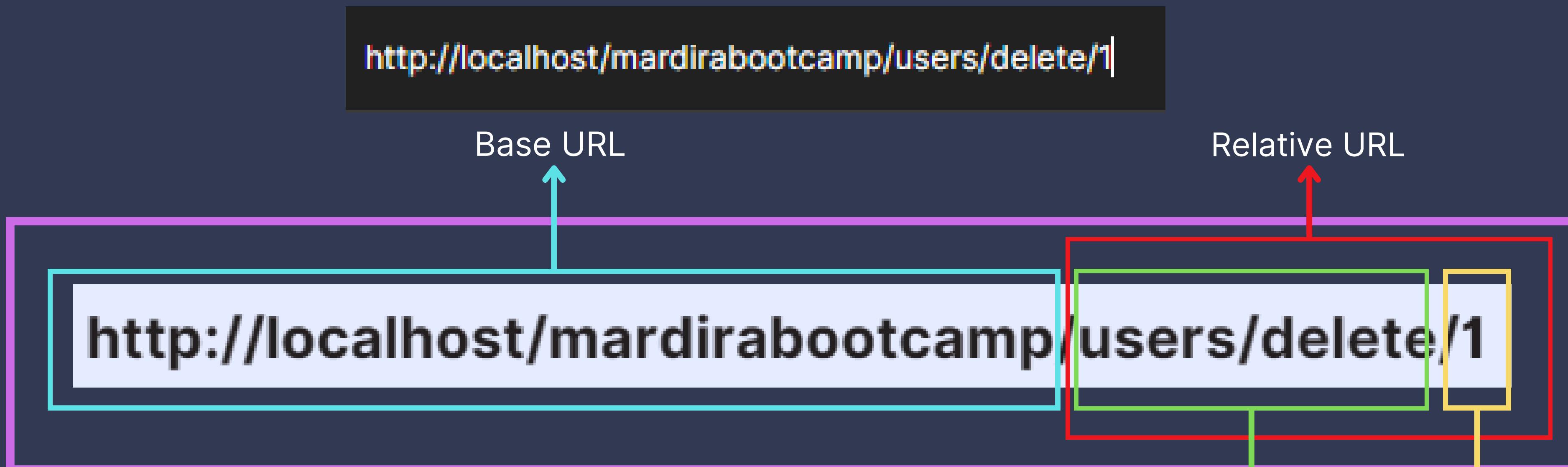
The screenshot shows the Postman application interface. In the top navigation bar, 'Workspaces' is selected, and the workspace is named 'My Workspace'. A collection named 'Bootcamp' contains a 'Users' folder with a 'GET users' request. The request method is 'GET' and the URL is 'http://localhost/mardirabootcamp/users'. The 'Send' button is highlighted with a red box. Below the request, the 'Params' tab is active, showing a table for 'Query Params' with columns for KEY, VALUE, DESCRIPTION, and Bulk Edit. The 'Body' tab is selected, showing a JSON response with 15 numbered lines. The response body is:

```
1  {
2    "message": "Data User",
3    "total": 20,
4    "data": [
5      {
6        "user_id": "1",
7        "username": "admin",
8        "password": "ad8c0863e80c639ed29473efb54ea4fc"
9      },
10     {
11       "user_id": "2",
12       "username": "user1",
13       "password": "7e58d63b60197ceb55a1c487989a3720"
14     },
15   ]
```

The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 44 ms, Size: 3.15 KB.

Kita tidak perlu lagi menambahkan ekstensi .php untuk URL nya karena sudah diatur oleh .htaccess itu sendiri, untuk menambahkan Endpoint URL cukup menambahkan prefix nya saja "/users", setelah itu pilih tombol send untuk melakukan request.

URL Pattern / Digital Object Identifier



- Absolute URL
 - Base URL
 - Relative URL
 - Prefix
 - Suffix

Absolute URL

Relative URL

Prefix

Suffix

Response Endpoint

The screenshot shows the Postman application interface. In the top navigation bar, the 'Explore' tab is selected. On the left sidebar, under 'My Workspace', there is a 'Collections' section with 'Bootcamp' expanded, showing a 'Users' folder containing a 'GET users' request. The main workspace displays this 'GET users' request. The method is set to 'GET' and the URL is 'http://localhost/mardirabootcamp/users'. Below the URL, the 'Params' tab is active, showing a table for 'Query Params' with one row: 'Key' (Value) and 'Description' (Description). The 'Body' tab is also visible. At the bottom of the request card, it says 'Status: 200 OK Time: 10 ms Size: 3.15 KB'. The response body is displayed in a code editor with tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize', with 'Pretty' selected. The response JSON is:

```
    "message": "Data User",
    "total": 20,
    "data": [
        {
            "user_id": "1",
            "username": "admin",
            "password": "ad8c0863e80c639ed29473efb54ea4fc"
        },
        {
            "user_id": "2",
            "username": "user1",
            "password": "7e58d63b60197ceb55a1c487989a3720"
        }
    ]
```

Untuk melihat response pada endpoint yang telah direquest pilih satu tab yaitu pretty, raw, preview.

Menerapkan HTTP Verbs

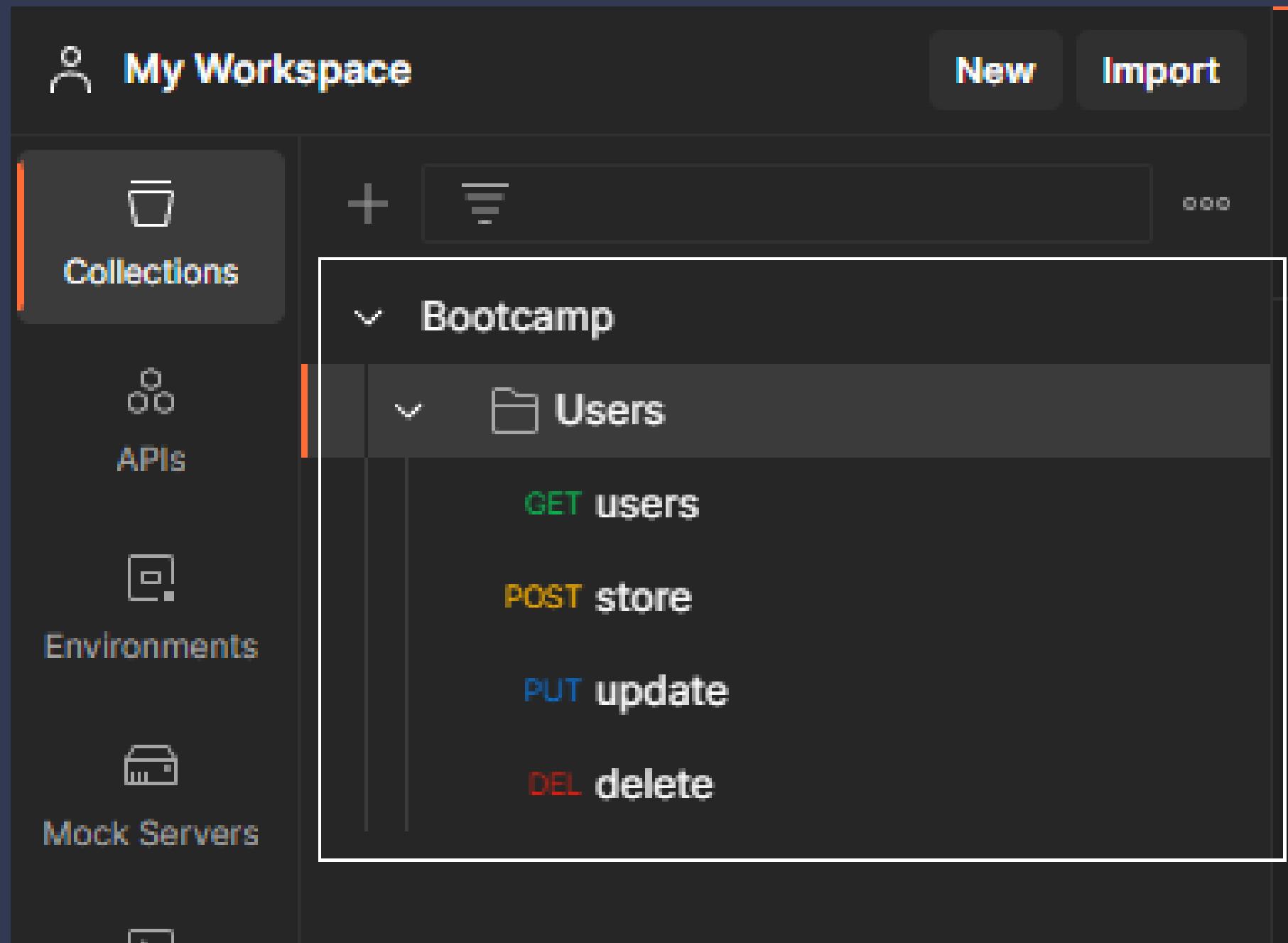
Route	Request Method	Body Request / Payload	Description
/users	GET	-	Menampilkan Semua Data Users
/users/:id	GET	-	Menampilkan satu data users sesuai id
/users	POST	<ul style="list-style-type: none">• username• password	Menyimpan data User
/users/:id	PUT	<ul style="list-style-type: none">• id• username• password	Mengubah data user sesuai dengan id
/users/:id	Delete	-	Menghapus data user sesuai dengan id

Todo

- URL Design 
- HTTP Verbs 
- HTTP Response Code 
- Format Response 

Setelah membuat URL Design, langkah selanjutnya ialah menerapkan HTTP verbs tujuannya untuk menggunakan semua method yang telah didefinisikan.

Menambahkan Endpoint Lainnya



Tambahkan endpoint lainnya dengan HTTP verbs sesuai dengan di atas.

Control Statement Switch Untuk HTTP Verbs



```
1 $users = new Users();
2
3 switch ($_SERVER['REQUEST_METHOD']) {
4     case 'GET':
5         $users->users();
6         break;
7     case 'POST':
8         $users->store();
9         break;
10    case 'PUT':
11        $users->update($users->id);
12        break;
13    case 'DELETE':
14        $users->delete($users->id);
15        break;
16    default:
17        $response = [
18            'message' => 'Method Not Allowed',
19        ];
20        http_response_code(405);
21        echo json_encode($response);
22        break;
23 }
```

Setelah menambahkan endpoint di Postman, agar bisa menggunakan semua method untuk setiap requestnya dengan cara menggunakan control statement switch.

Menambahkan Property Class Users

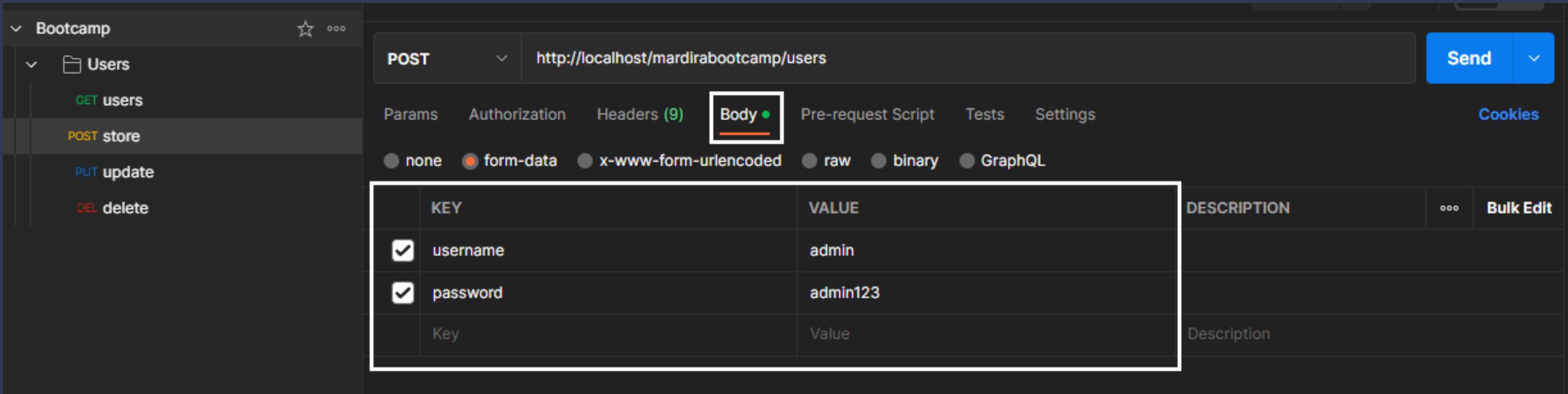
```
Users
- table : string
- username : string ? null
- password : string ? null
* + id: int ? null
- statement : object ? null
```

```
case 'DELETE':
    $users->delete($users->id);
break;
```

```
● ● ●
1 class Users
2 {
3     private string $table = 'users';
4     private ?string $username;
5     private ?string $password;
6     public ?int $id;
7     private ?object $statement;
8
9     public function __construct()
10    {
11         $this->statement = new Database();
12         $this->statement = $this->statement->connection;
13         $this->username = $_POST['username'] ?? null;
14         $this->password = $_POST['password'] ?? null;
15         $this->id = $_GET['id'] ?? null;
16    }
}
```

Kemudian tambahkan public property untuk id, supaya bisa digunakan diluar class untuk method delete dan update

Store Data Menggunakan Postman



The screenshot shows the Postman application interface. On the left, there's a sidebar with a project named 'Bootcamp' containing a collection 'Users' with methods: 'GET users', 'POST store' (which is selected), 'PUT update', and 'DEL delete'. The main area shows a POST request to 'http://localhost/mardirabootcamp/users'. The 'Body' tab is selected, showing 'form-data' selected. A table is present with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. Two rows are visible: one for 'username' with value 'admin' and one for 'password' with value 'admin123'. A third row is partially visible with 'Key' and 'Value' columns.

KEY	VALUE	DESCRIPTION	Bulk Edit
<input checked="" type="checkbox"/> username	admin		
<input checked="" type="checkbox"/> password	admin123		
Key	Value	Description	

Untuk melakukan store data pada Postman, pilih tab body lalu pilih form-data kemudian masukkan key dan value sesuaikan dengan field/column yang ada di table yang akan ditambahkan.

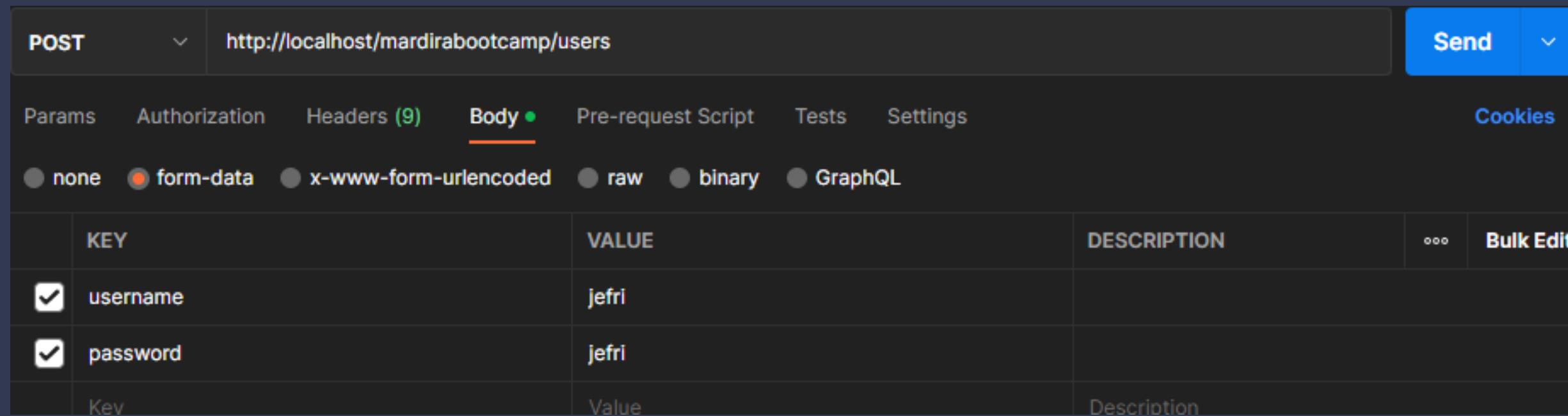
Proses Menyimpan Data Dari Postman



```
1 public function store(): void
2 {
3     $query = "SELECT * FROM {$this->table} WHERE username = :username";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':username', $this->username);
6     $statement->execute();
7     if ($statement->rowCount() > 0) {
8         $response = [
9             'message' => 'Data User Sudah Ada!',
10            ];
11    } else {
12        $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
13        $statement = $this->statement->prepare($query);
14        $password = md5($this->password);
15        $statement->bindParam(':username', $this->username);
16        $statement->bindParam(':password', $password);
17        $statement->execute();
18        $getUsers = "SELECT * FROM users WHERE username = :username";
19        $statement = $this->statement->prepare($getUsers);
20        $statement->bindParam(':username', $this->username);
21        $statement->execute();
22        $user = $statement->fetch(PDO::FETCH_OBJ);
23        $response = [
24            'message' => 'Data User Berhasil Disimpan',
25            'user' => $user,
26            ];
27    }
28    echo json_encode($response, JSON_PRETTY_PRINT);
29 }
```

Sebelum melakukan penyimpanan data akan di cek terlebih dahulu untuk memastikan data tersebut bersifat duplikat atau tidak dengan menggunakan query rowCount untuk menghitung data. Lalu menggunakan control statement untuk menjalankan salah satu aksi / response yang akan ditampilkan.

Response Store Data



POST http://localhost/mardirabootcamp/users

Body (form-data)

KEY	VALUE	DESCRIPTION	...	Bulk Edit
username	jefri			
password	jefri			

Body Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {"message": "Data User Berhasil Disimpan", "user": { "user_id": "23", "username": "jefri", "password": "c710857e9b674843afc9b54b7ae2032d" }}
```

Data berhasil disimpan

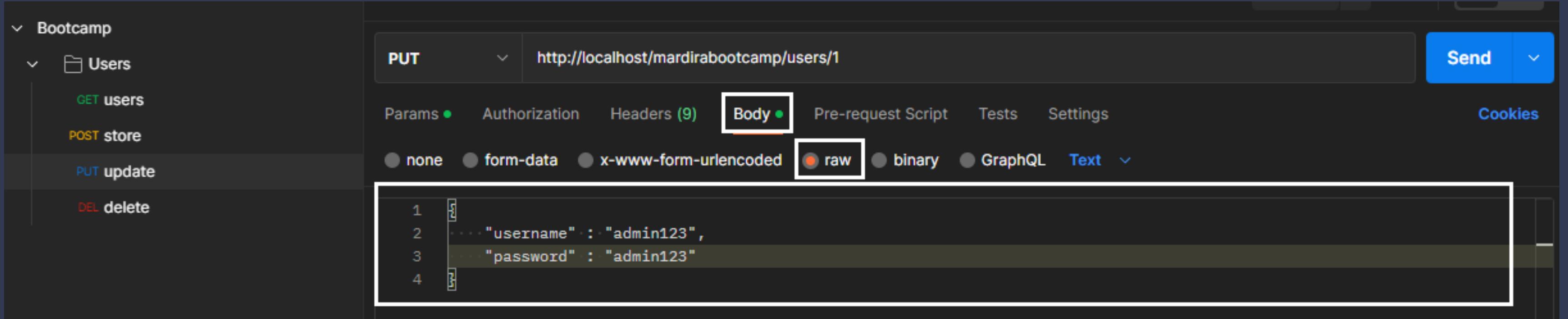
Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {"message": "Data User Sudah Ada!"}
```

Duplikat data

Update Data Menggunakan Postman



Saat melakukan update data (PUT) dari Postman berbeda dengan melakukan store (POST), karena PUT tidak bisa menggunakan form-data seperti POST. Jadi untuk melakukan perubahan data pilih tab body lalu pilih raw, kemudian masukkan data berupa format JSON, dan cantumkan id yang ingin diubah datanya. Posisi id berada di suffix setelah prefix pada endpoint "**users/1**"

Proses Mengubah Data Dari Postman



```
1 public function update(int $id): void
2 {
3
4     $data = file_get_contents("php://input");
5     $_PUT = json_decode($data, true);
6     $this->username = $_PUT['username'] ?? null;
7     $this->password = $_PUT['password'] ?? null;
8
9     $query = "SELECT * FROM {$this->table} WHERE user_id = :user_id";
10    $statement = $this->statement->prepare($query);
11    $statement->bindParam(':user_id', $id);
12    $statement->execute();
13
14    if ($statement->rowCount() == 0) {
15        $response = [
16            'message' => 'Data User Tidak Ditemukan!',
17        ];
18        echo json_encode($response, JSON_PRETTY_PRINT);
19        exit;
20    }
21
22    $query = "UPDATE {$this->table} SET username = :username, password = :password WHERE user_id = :user_id";
23    $statement = $this->statement->prepare($query);
24    $password = md5($this->password);
25    $statement->bindParam(':username', $this->username);
26    $statement->bindParam(':password', $password);
27    $statement->bindParam(':user_id', $id);
28    $statement->execute();
29    $response = [
30        'message' => 'Data User Berhasil Diubah',
31    ];
32    echo json_encode($response, JSON_PRETTY_PRINT);
33 }
```

Proses untuk mengubah data di method update, untuk menangkap payload/request data yang sifatnya raw (mentah), menggunakan fungsi file_get_contents, sebelum melakukan update data untuk memastikan data tersebut ada atau tidak dengan cara melakukan cek data dengan id, jika tidak ada maka akan memberi response tidak ada dan json_decode yang berfungsi untuk mengubah dari JSON menjadi array php.

Json Encode & Json Decode

Array PHP

```
● ○ ●  
1 [  
2   'username' => 'admin123',  
3   'password' => 'admin123',  
4 ];
```

json_encode

JSON

konversi

```
● ○ ●  
1 {  
2   "username": "admin123",  
3   "password": "admin123"  
4 }
```

JSON

json_decode

Array PHP

konversi

```
● ○ ●  
1 {  
2   "username": "admin123",  
3   "password": "admin123"  
4 }
```

```
● ○ ●  
1 [  
2     'username' => 'admin123',  
3     'password' => 'admin123',  
4   ];
```

Response Update Data

A screenshot of the Postman application interface. The request method is set to PUT, and the URL is `http://localhost/mardirabootcamp/users/1`. The Body tab is selected, showing a raw JSON payload:

```
1
2   ...
3     "username" : "admin123",
4     "password" : "admin123"
```

The response status is 200 OK, with a message: "Data User Berhasil Diubah".

Data berhasil diubah

A screenshot of the Postman application interface. The request method is set to PUT, and the URL is `http://localhost/mardirabootcamp/users/1234`. The Body tab is selected, showing a raw JSON payload:

```
1
2   ...
3     "username" : "admin123",
4     "password" : "admin123"
```

The response status is 200 OK, with a message: "Data User Tidak Ditemukan!".

Data tidak ditemukan

Delete Data Menggunakan Postman

The screenshot shows the Postman application interface. On the left, there is a sidebar with a tree structure under 'Bootcamp' containing 'Users' with various methods: GET users, POST store, PUT update, and DEL delete. The 'DEL delete' method is selected. The main workspace shows a 'DELETE' request to the URL 'http://localhost/mardirabootcamp/users/1'. The 'Params' tab is active, displaying a table with one row: 'Key' (empty) and 'Value' (empty). Other tabs include 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Body' (which is also the active tab). At the bottom right, the status bar shows 'Status: 200 OK', 'Time: 34 ms', 'Size: 509 B', and a 'Save Response' button.

Delete data menggunakan Postman cukup menambahkan suffix id pada endpoint delete.

Proses Menghapus Data Dari Postman



```
1 public function delete(int $id): void
2 {
3     $query = "SELECT * FROM {$this->table} WHERE user_id = :user_id";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':user_id', $id);
6     $statement->execute();
7
8     if ($statement->rowCount() == 0) {
9         $response = [
10             'message' => 'Data User Tidak Ditemukan!',
11         ];
12         echo json_encode($response, JSON_PRETTY_PRINT);
13         exit;
14     }
15
16     $query = "DELETE FROM {$this->table} WHERE user_id = :user_id";
17     $statement = $this->statement->prepare($query);
18     $statement->bindParam(':user_id', $id);
19     $statement->execute();
20     $response = [
21         'message' => 'Data User Berhasil Dihapus',
22     ];
23     echo json_encode($response, JSON_PRETTY_PRINT);
24 }
```

Untuk method delete tambahka select data where id untuk menentukan data yang dihapus terdaftar ditable users.

Response Delete Data

A screenshot of the Postman application interface. The top bar shows a 'DELETE' method and the URL 'http://localhost/mardirabootcamp/users/3'. Below the URL are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected. Under 'Body', there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The JSON response is displayed in a code editor-like format:

```
1 "message": "Data User Berhasil Dihapus"
```

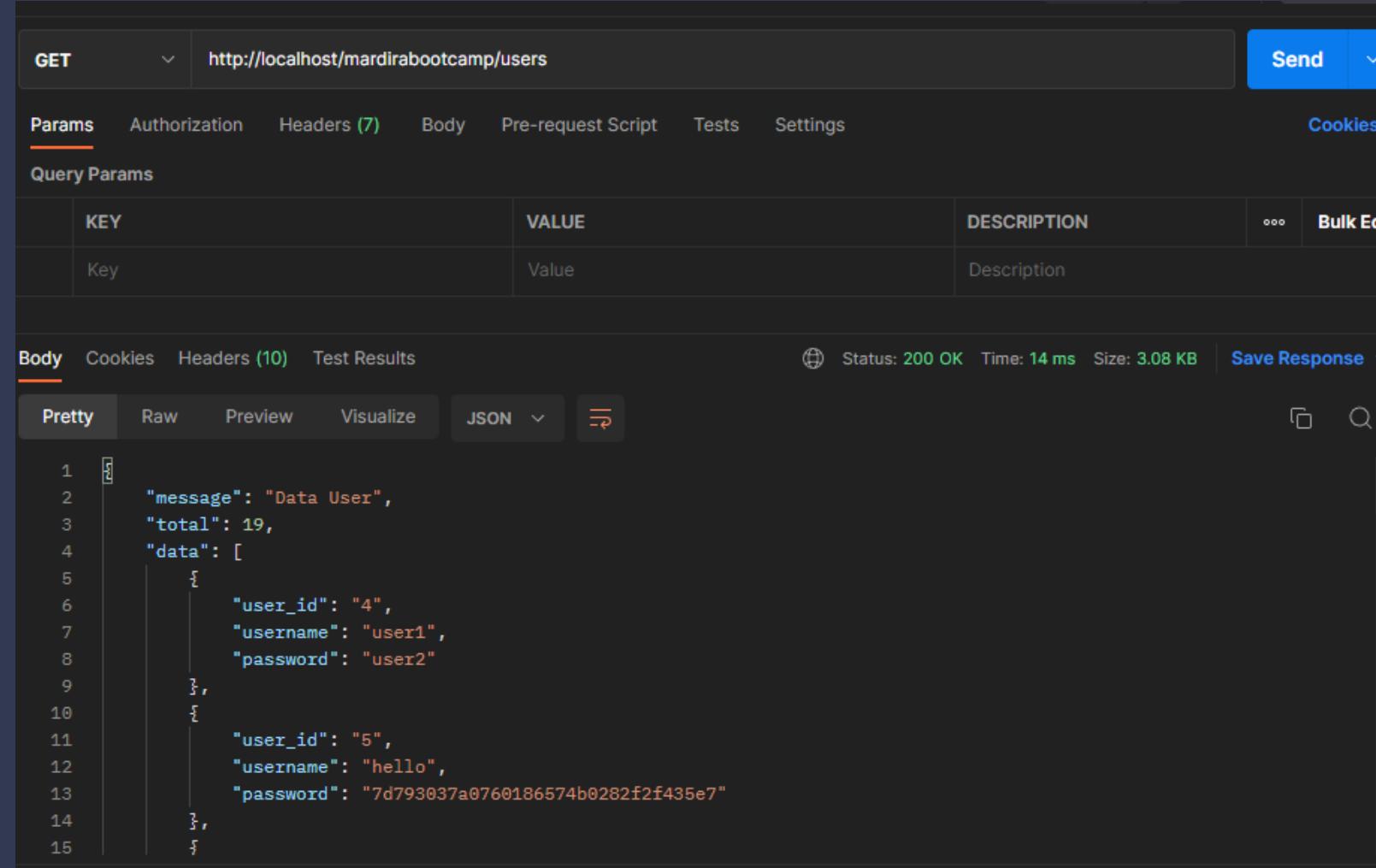
Data berhasil dihapus

A screenshot of the Postman application interface, identical in layout to the first one. The top bar shows a 'DELETE' method and the URL 'http://localhost/mardirabootcamp/users/3'. The 'Body' tab is selected. The JSON response is displayed in a code editor-like format:

```
1 "message": "Data User Tidak Ditemukan!"
```

Data tidak ditemukan

Menampilkan Data Berdasarkan ID



GET http://localhost/mardirabootcamp/users

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

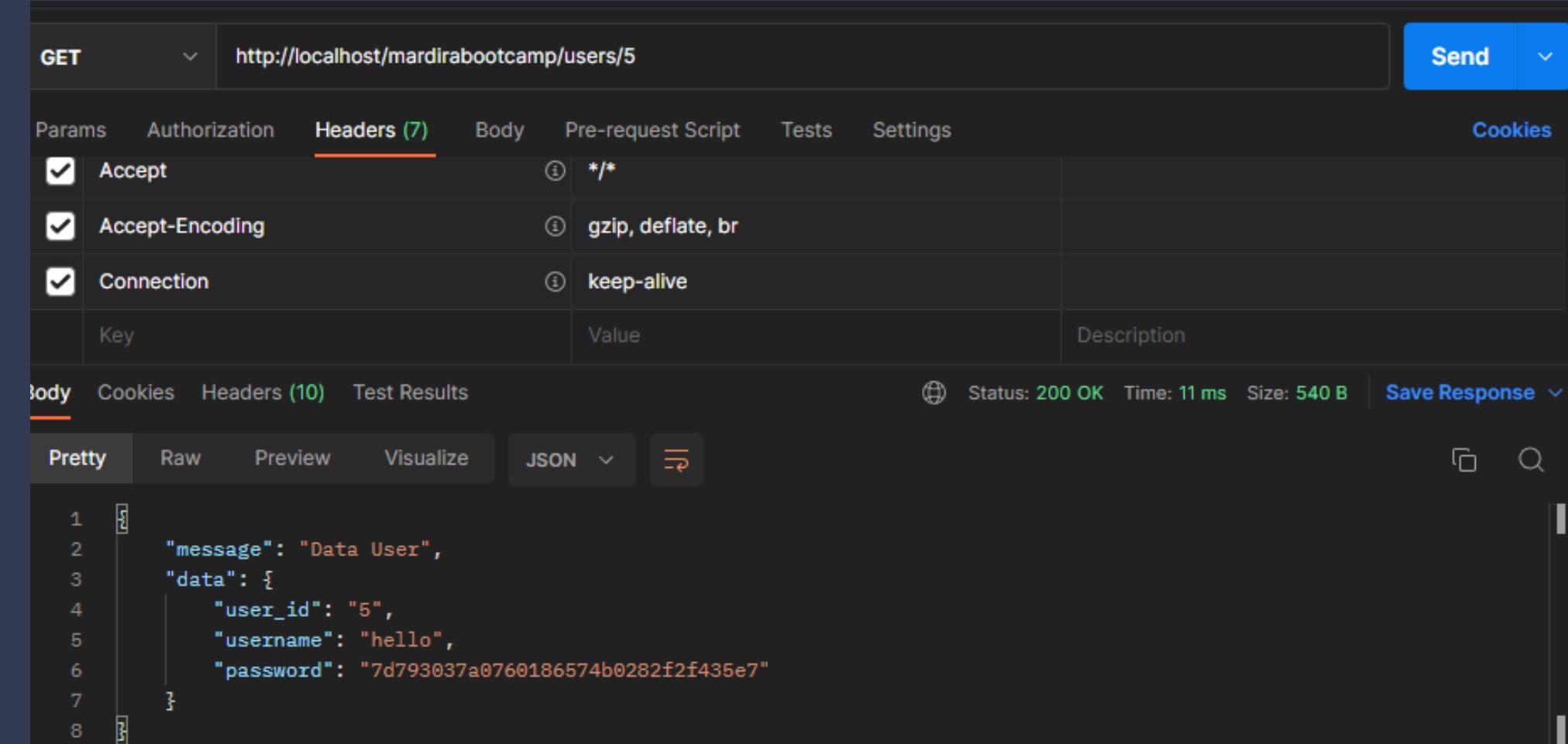
Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 "message": "Data User",
2 "total": 19,
3 "data": [
4   {
5     "user_id": "4",
6     "username": "user1",
7     "password": "user2"
8   },
9   {
10    "user_id": "5",
11    "username": "hello",
12    "password": "7d793037a0760186574b0282f2f435e7"
13  },
14 ]
15 }
```



GET http://localhost/mardirabootcamp/users/5

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Headers

Key	Value	Description
Accept	*	*
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Key	Value	Description

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 "message": "Data User",
2 "data": {
3   "user_id": "5",
4   "username": "hello",
5   "password": "7d793037a0760186574b0282f2f435e7"
6 }
```

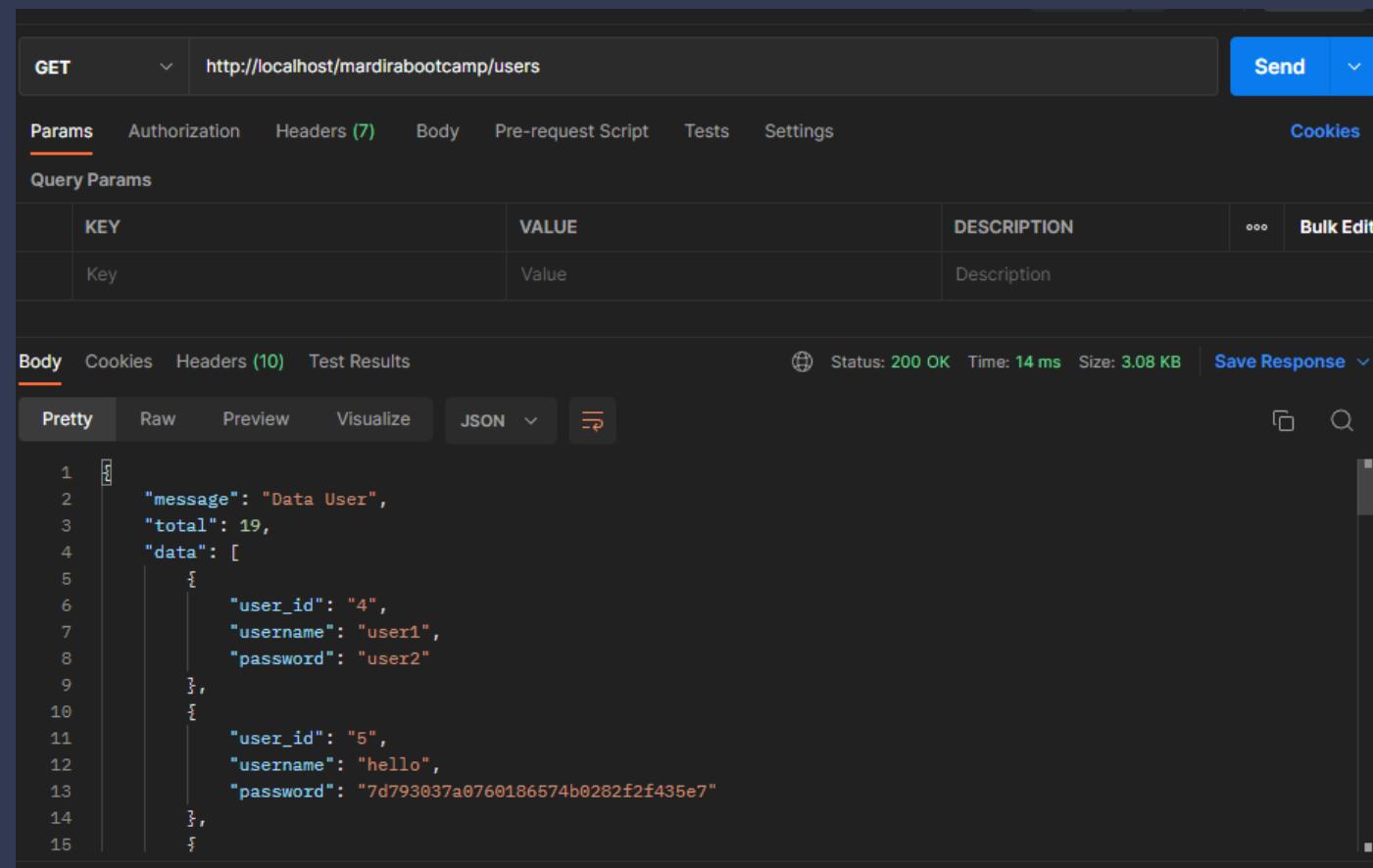
Bagaimana caranya menampilkan semua data dan menampilkan data berdasarkan id yang dipilih, dalam satu endpoint?

Method Khusus Untuk Get Data

```
1 public function users(): void
2 {
3     if ($this->id) {
4         $user = $this->getUser($this->id);
5         $response = [
6             'message' => 'Data User',
7             'data' => $user
8         ];
9         echo json_encode($response, JSON_PRETTY_PRINT);
10    } else {
11        $users = $this->getUsers();
12        $total = count($users);
13        $response = [
14            'message' => 'Data User',
15            'total' => $total,
16            'data' => $users
17        ];
18        echo json_encode($response, JSON_PRETTY_PRINT);
19    }
20}
21
22 private function getUsers(): array
23 {
24     $query = "SELECT * FROM {$this->table}";
25     $statement = $this->statement->prepare($query);
26     $statement->execute();
27     $users = $statement->fetchAll(PDO::FETCH_OBJ);
28     return $users;
29}
30
31 private function getUser(int $id): object
32 {
33     $query = "SELECT * FROM {$this->table} WHERE user_id = :user_id";
34     $statement = $this->statement->prepare($query);
35     $statement->bindParam(':user_id', $id);
36     $statement->execute();
37     if ($statement->rowCount() == 0) {
38         $response = [
39             'message' => 'Data User Tidak Ditemukan!',
40         ];
41         http_response_code(404);
42         echo json_encode($response, JSON_PRETTY_PRINT);
43         exit;
44     }
45     $user = $statement->fetch(PDO::FETCH_OBJ);
46     return $user;
47 }
```

Agar bisa menampilkan (hybrid data) semua data atau data berdasarkan id saja, dengan cara menambahkan private method, kemudian gunakan control statement jika id yang di input ada ataupun tidak.

Response Get Data



GET http://localhost/mardirabootcamp/users

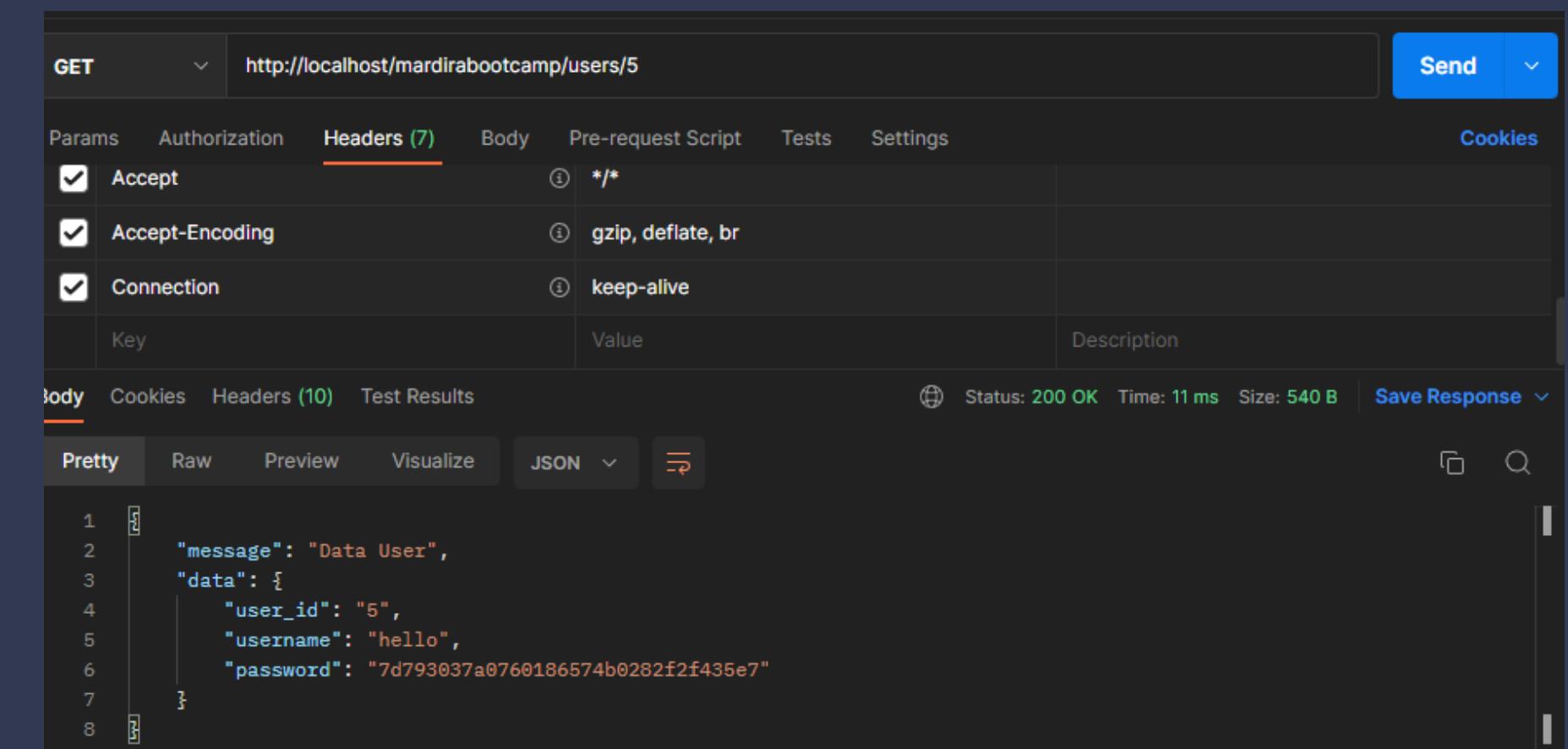
Headers (7)

Key	Value	Description
Key	Value	Description

Body (Pretty)

```
1 "message": "Data User",
2 "total": 19,
3 "data": [
4   {
5     "user_id": "4",
6     "username": "user1",
7     "password": "user2"
8   },
9   {
10    "user_id": "5",
11    "username": "hello",
12    "password": "7d793037a0760186574b0282f2f435e7"
13  },
14 ],
15 }
```

Menampilkan semua data



GET http://localhost/mardirabootcamp/users/5

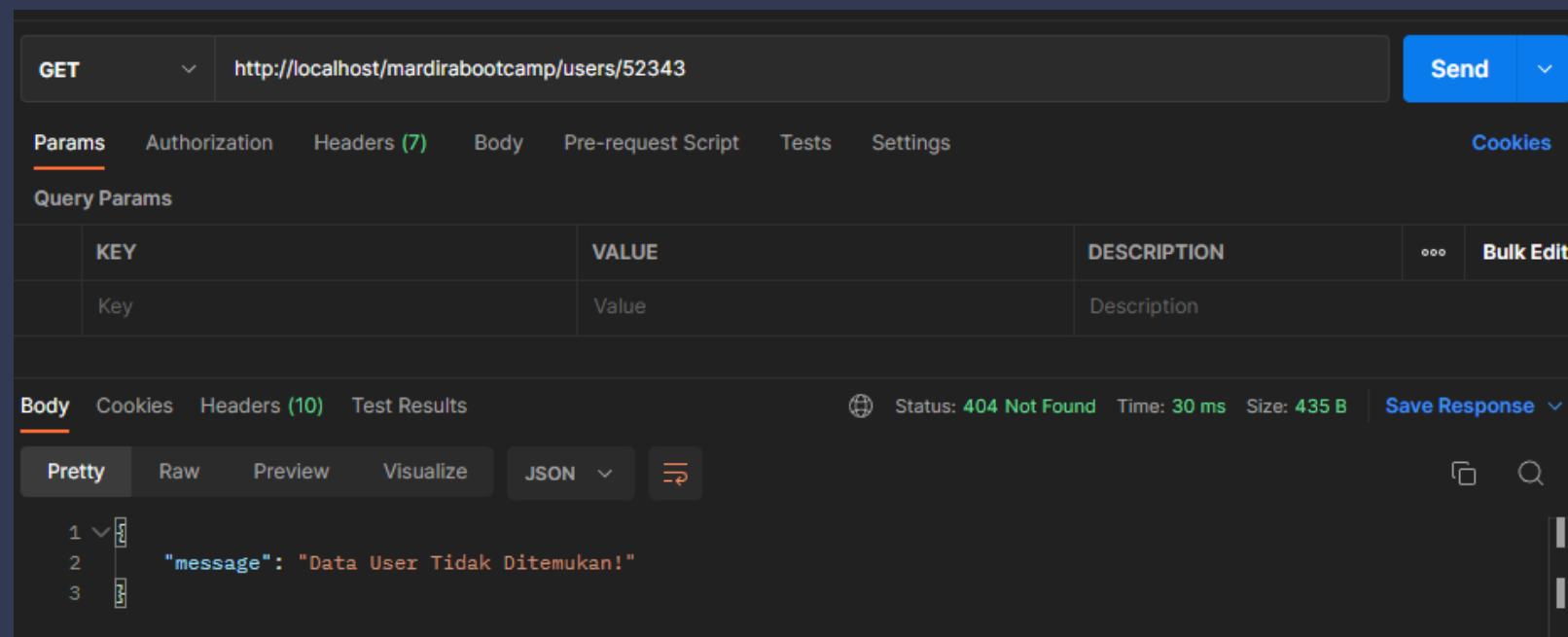
Headers (7)

Key	Value	Description
Accept	/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Key	Value	Description

Body (Pretty)

```
1 "message": "Data User",
2 "data": {
3   "user_id": "5",
4   "username": "hello",
5   "password": "7d793037a0760186574b0282f2f435e7"
6 }
```

Menampilkan data berdasarkan id



GET http://localhost/mardirabootcamp/users/52343

Headers (7)

Key	Value	Description
Key	Value	Description

Body (Pretty)

```
1 "message": "Data User Tidak Ditemukan!"
```

Data tidak ditemukan

Menerapkan HTTP Response Code



200 (OK)



201 (Created)



400
(Bad Request)
401
(Unauthorized)



401



500
(ISE)



403

(Forbidden)



404

(Not Found)



405
(Method
Not
Allowed)



409

(Conflict)
503
(Service
Unavailable)



503

Todo

- URL Design
- HTTP Verbs
- HTTP Response Code
- Format Response



Setelah menentukan HTTP verbs, langkah selanjutnya adalah menentukan HTTP Response code setiap response dari endpoint yang sudah ditentukan.

Melihat HTTP Response Code

The screenshot shows the Postman application interface. At the top, there is a header bar with a dropdown menu set to "DELETE", a URL input field containing "http://localhost/mardirabootcamp/users/3", and a blue "Send" button. Below the header, a navigation bar includes tabs for "Params", "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", "Settings", and "Cookies". The "Headers" tab is currently selected. Under the "Headers" tab, there is a section titled "Query Params" with a table:

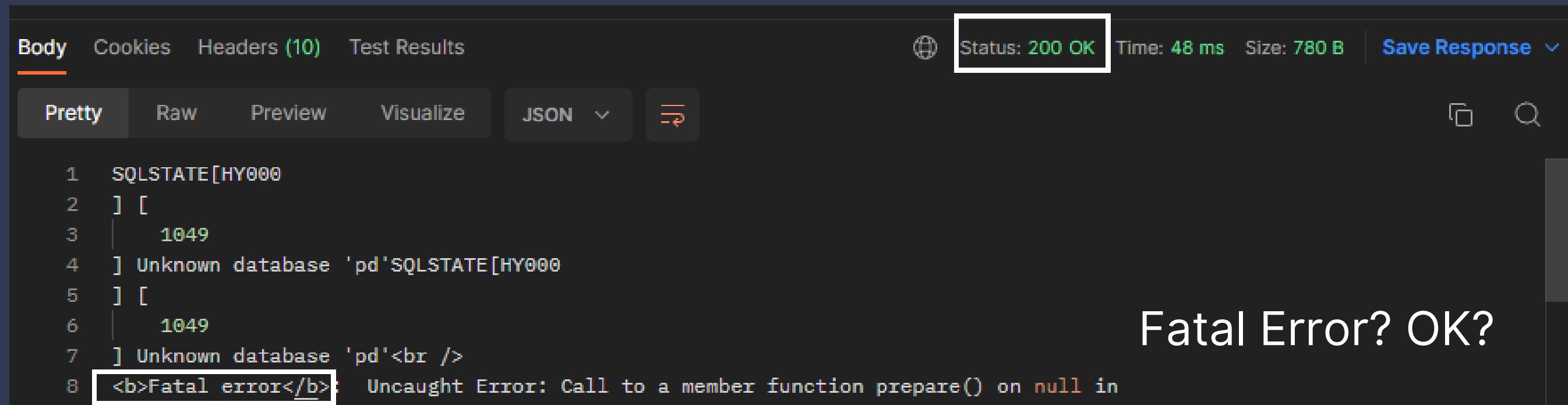
	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Below the table, there is a summary bar with tabs for "Body", "Cookies", "Headers (10)", and "Test Results". The "Body" tab is selected. On the right side of the summary bar, it displays "Status: 200 OK", "Time: 11 ms", and "Size: 427 B". A "Save Response" button is also present. Under the "Body" tab, there are four viewing options: "Pretty" (selected), "Raw", "Preview", and "Visualize". The "Pretty" view shows the JSON response:

```
1 {  
2   "message": "Data User Berhasil Dihapus"  
3 }
```

Setelah melakukan send request akan muncul status code / response code seperti di atas.

Jika Tidak Menerapkan HTTP Response Code

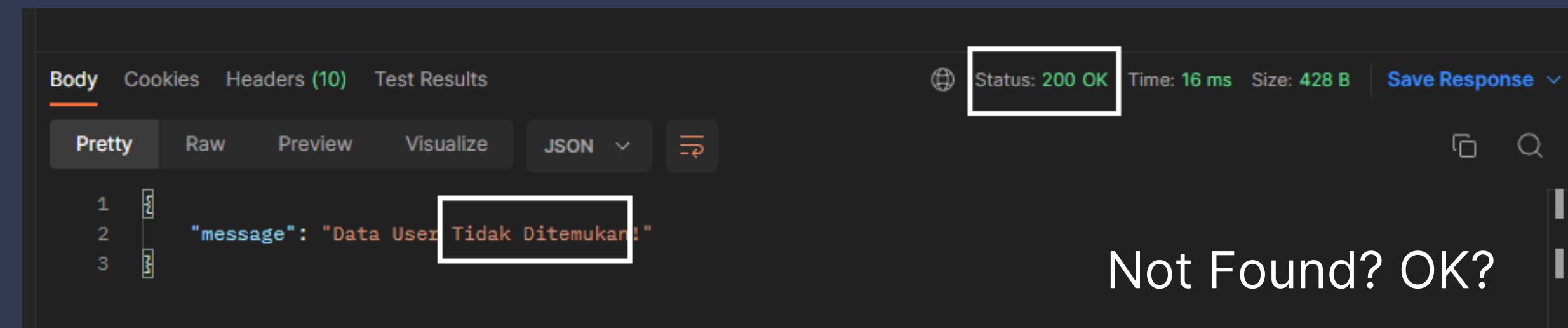


The screenshot shows a Postman test result. The status bar at the top right indicates "Status: 200 OK". The body of the response is displayed in a code editor-like interface. The text is as follows:

```
1 SQLSTATE[HY000]
2 ] [
3   1049
4 ] Unknown database 'pd'SQLSTATE[HY000
5 ] [
6   1049
7 ] Unknown database 'pd'<br />
8 <b>Fatal error</b>: Uncaught Error: Call to a member function prepare() on null in
```

A red box highlights the status code "200 OK" in the status bar, and another red box highlights the error message "Fatal error" in the response body.

Fatal Error? OK?



The screenshot shows a Postman test result. The status bar at the top right indicates "Status: 200 OK". The body of the response is displayed in a code editor-like interface. The text is as follows:

```
1 {
2   "message": "Data User Tidak Ditemukan!"
3 }
```

A red box highlights the status code "200 OK" in the status bar, and another red box highlights the error message "Data User Tidak Ditemukan!" in the response body.

Not Found? OK?

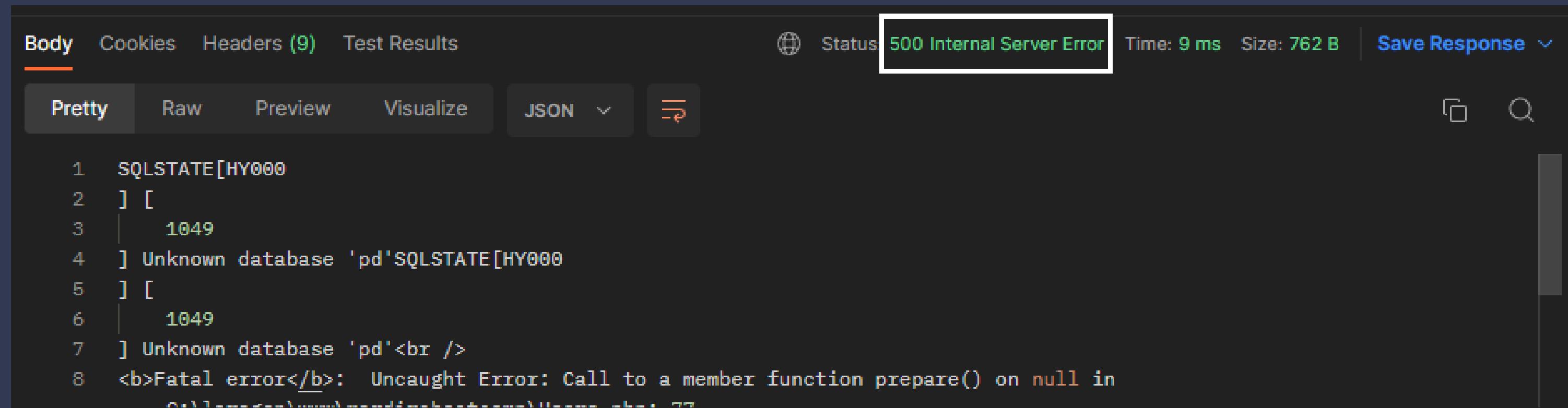
Secara default jika kita tidak menggunakan status code, status code yang akan muncul akan slalu 200 (OK), dan kalau tidak menerapkan status code justru akan menyulitkan client / Front-End yang ingin berkolaborasi dengan kalian sebagai Back-End Developer

Implementasi HTTP Response Code

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

Untuk menetapkan status code pada setiap response, dengan cara menambahkan `http_response_code(kode)` dengan parameter kodennya, untuk contoh jika ada kesalahan di server dengan menggunakan status code 500 (Internal Server Error).

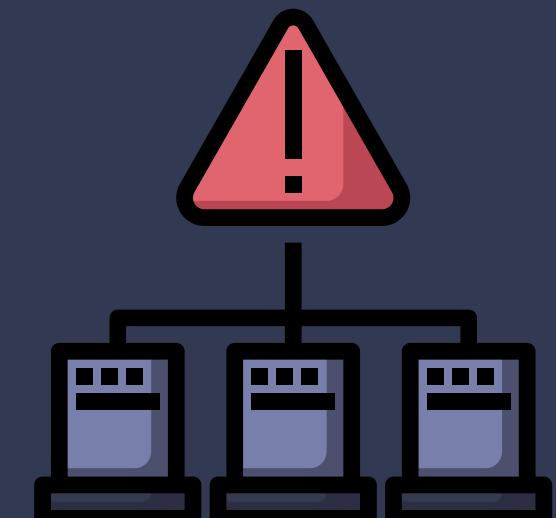
Hasil HTTP Response Code



The screenshot shows the Postman application interface. At the top, there are tabs for 'Body', 'Cookies', 'Headers (9)', and 'Test Results'. The 'Status' bar indicates a '500 Internal Server Error'. Below the tabs, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The main content area displays the following JSON response:

```
1  SQLSTATE[HY000]
2  ] [
3  |   1049
4  ] Unknown database 'pd'SQLSTATE[HY000
5  ] [
6  |   1049
7  ] Unknown database 'pd'<br />
8  <b>Fatal error</b>:  Uncaught Error: Call to a member function prepare() on null in
```

Setelah menetapkan HTTP Response code maka hasilnya akan muncul setelah melakukan send request menggunakan Postman.



Implementasi HTTP Response Code Lainnya



```
1 private function getUser(int $id): object
2 {
3     $query = "SELECT * FROM {$this->table} WHERE user_id = :user_id";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':user_id', $id);
6     $statement->execute();
7     if ($statement->rowCount() == 0) {
8         $response = [
9             'message' => 'Data User Tidak Ditemukan!',
10        ];
11        http_response_code(404);
12        echo json_encode($response, JSON_PRETTY_PRINT);
13        exit;
14    }
15    $user = $statement->fetch(PDO::FETCH_OBJ);
16    return $user;
17 }
```

Status: 404 Not Found

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {<br>2   "message": "Data User Tidak Ditemukan!"<br>3 }
```

Jika data yang dicari tidak ada maka status code yang digunakan adalah 404 (Not Found).



Implementasi HTTP Response Code Lainnya



```
1 public function store(): void
2 {
3     $query = "SELECT * FROM {$this->table} WHERE username = :username";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':username', $this->username);
6     $statement->execute();
7     if ($statement->rowCount() > 0) {
8         $response = [
9             'message' => 'Data User Sudah Ada!',
10            ];
11        http_response_code(409);
12    } else {
13        $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
14        $statement = $this->statement->prepare($query);
15        $password = md5($this->password);
16        $statement->bindParam(':username', $this->username);
17        $statement->bindParam(':password', $password);
18        $statement->execute();
19        $getUsers = "SELECT * FROM users WHERE username = :username";
20        $statement = $this->statement->prepare($getUsers);
21        $statement->bindParam(':username', $this->username);
22        $statement->execute();
23        $user = $statement->fetch(PDO::FETCH_OBJ);
24        $response = [
25            'message' => 'Data User Berhasil Disimpan',
26            'user' => $user,
27        ];
28    }
29    echo json_encode($response, JSON_PRETTY_PRINT);
30 }
```

Status: 409 Conflict

Body	Cookies	Headers (10)	Test Results
<p>Pretty</p> <p>Raw</p> <p>Preview</p> <p>Visualize</p> <p>JSON</p> <p>Copy</p> <pre>1 { 2 "message": "Data User Sudah Ada!" 3 }</pre>			

Untuk data yang bersifat duplikat gunakan 409 (Conflict)



Implementasi HTTP Response Code Lainnya



```
1
2 switch ($_SERVER['REQUEST_METHOD']) {
3     case 'GET':
4         $users->users();
5         break;
6     case 'POST':
7         $users->store();
8         break;
9     case 'PUT':
10        $users->update($users->id);
11        break;
12    case 'DELETE':
13        $users->delete($users->id);
14        break;
15    default:
16        $response = [
17            'message' => 'Method Not Allowed',
18        ];
19        http_response_code(405);
20        echo json_encode($response);
21        break;
22 }
```

The screenshot shows a Postman interface with a PATCH request to `http://localhost/mardirabootcamp/users/`. The Headers tab shows 9 items. The Body tab is selected and set to `form-data`, containing two fields: `username` and `password`, both with the value `admin`. The status bar at the bottom shows `Status: 405 Method Not Allowed`.

405 (Method Not Allowed) untuk HTTP verbs yang tidak digunakan tidak diizinkan karena hanya bisa GET, PUT , POST, dan DELETE saja.



Implementasi HTTP Response Code Lainnya



```
1 public function store(): void
2 {
3     $query = "SELECT * FROM {$this->table} WHERE username = :username";
4     $statement = $this->statement->prepare($query);
5     $statement->bindParam(':username', $this->username);
6     $statement->execute();
7     if ($statement->rowCount() > 0) {
8         $response = [
9             'message' => 'Data User Sudah Ada!',
10        ];
11        http_response_code(409);
12    } else {
13        $query = "INSERT INTO {$this->table} (username, password) VALUES (:username, :password)";
14        $statement = $this->statement->prepare($query);
15        $password = md5($this->password);
16        $statement->bindParam(':username', $this->username);
17        $statement->bindParam(':password', $password);
18        $statement->execute();
19        $getUsers = "SELECT * FROM users WHERE username = :username";
20        $statement = $this->statement->prepare($getUsers);
21        $statement->bindParam(':username', $this->username);
22        $statement->execute();
23        $user = $statement->fetch(PDO::FETCH_OBJ);
24        $response = [
25            'message' => 'Data User Berhasil Disimpan',
26            'user' => $user,
27        ];
28        http_response_code(201);
29    }
30 }
31 echo json_encode($response, JSON_PRETTY_PRINT);
```

The screenshot shows a browser developer tools interface with the Network tab selected. A single request is listed, showing the following details:

- Body:** The response body is displayed in a JSON pretty-printed format:

```
1   "message": "Data User Berhasil Disimpan",
2   "user": {
3       "user_id": "24",
4       "username": "stafff",
5       "password": "1253208465b1efa876f982d8a9e73eeef"
6   }
7
8 }
```
- Headers (10):** Shows 10 headers, though they are not explicitly visible in the screenshot.
- Status:** Status: 201 Created

Untuk berhasil melakukan input data gunakan status code 201 (Created)



Menerapkan Format Response

```
{  
  message: "Data User",  
  data: {  
    user_id: "5",  
    username: "hello",  
    password: "7d793037a0760186574b0282f2f435e7"  
  }  
}
```

Todo

- URL Design 
- HTTP Verbs 
- HTTP Response Code 
- Format Response 

Komponen yang terakhir adalah format response dimana JSON adalah format yang sering digunakan untuk berinteraksi antara FE & BE

Format Reponse

konversi



Todo

- URL Design ✓
- HTTP Verbs ✓
- HTTP Response Code ✓
- Format Response ✓

Secara tidak langsung JSON (JavaScript Object Notation) adalah kumpulan beberapa array yang dikonversi oleh bahasa pemrograman PHP supaya FE bisa mengambil data yang telah dikonversi ke bahasa yang sifatnya client-side cth : JavaScript, TypeScript dll.

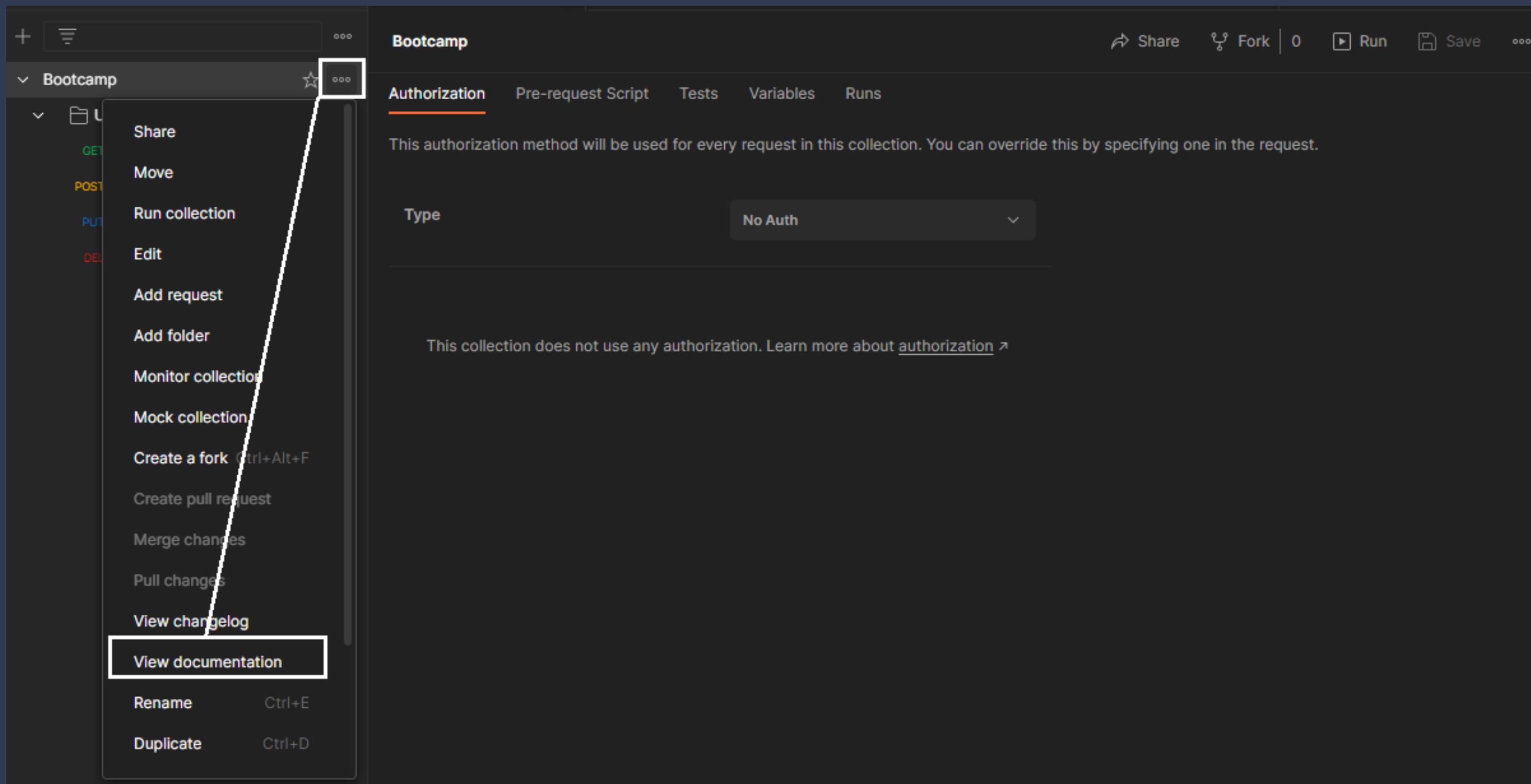
Langkah Terakhir Membuat Restful API



```
1 <?php  
2  
3 require_once('Database.php');  
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 class Users  
11 {  
12     private string $table = 'users';  
13     private ?string $username;  
14     private ?string $password;  
15     public ?int $id;  
16     private ?object $statement;  
17 }
```

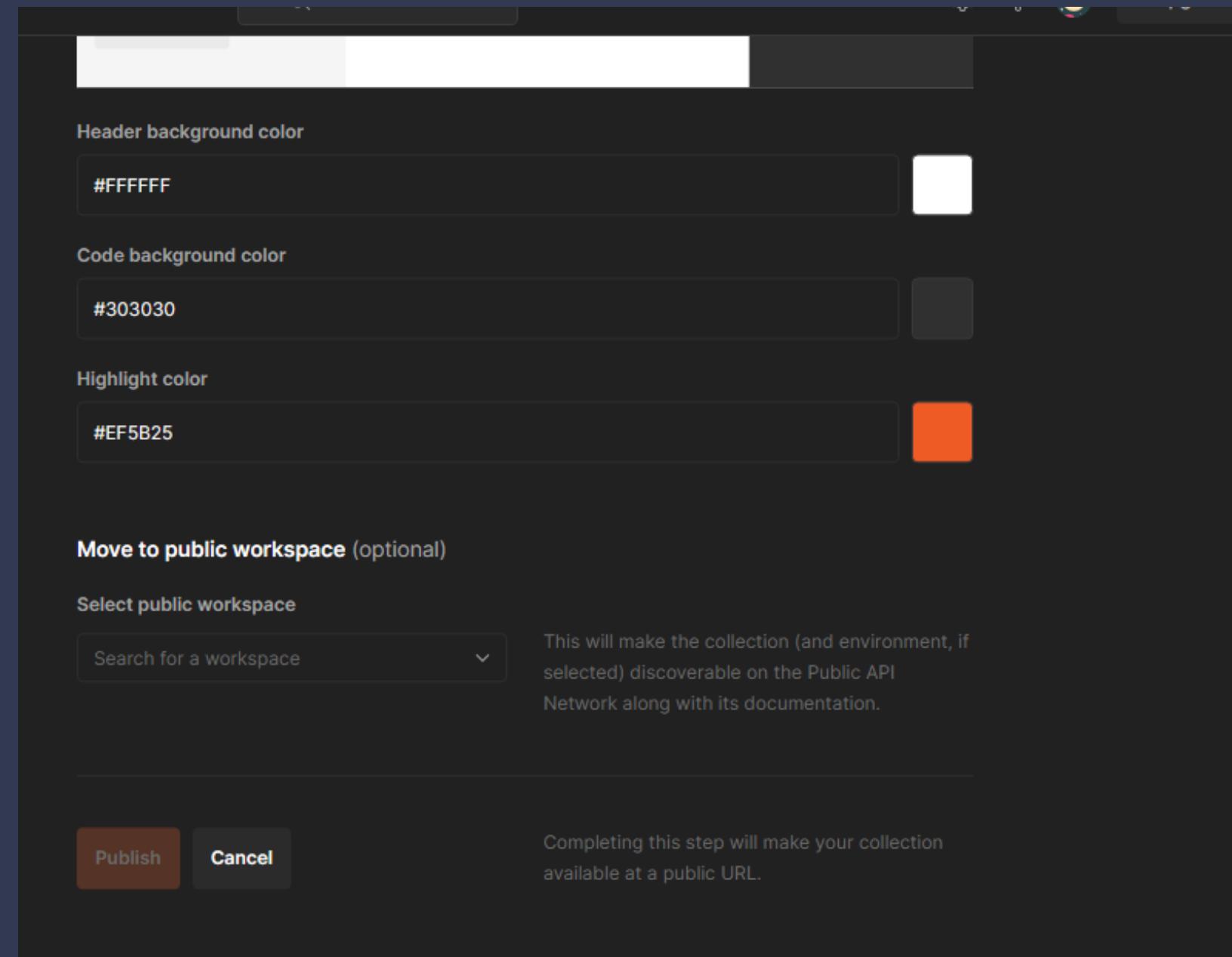
Setelah melalui 4 komponen penting dalam membuat RESTful API, selanjutnya dengan menambahkan beberapa header sebelum class Users supaya tidak terjadi error CORS (Cross-Origin Resource Sharing) kepada Front-End Developer, hal ini sering terjadi jika BE tidak mengijinkan untuk melakukan request method tertentu kepada FE Developer

Membuat Dokumentasi API Menggunakan Postman



Pilih tombol icon triple dot kemudian pilih view documentation

Publish Dokumentasi API



Setelah itu , kemudian pilih tombol publish untuk mendapatkan URL dokumentasi API yang untuk diserahkan kepada FE supaya RESTful API yang telah kita buat bisa diintegrasikan oleh FE Developer.

Kesimpulan

Setelah mempelajari beberapa materi back-end development, mulai dari fundamental hingga membuat sebuah API dengan konsep REST/RESTful API yang bertujuan untuk memudahkan kolaborasi antara back-end dan front-end developer, tetap semangat dan berlatih terus tentang back-end development. Sekian dari penulis, mohon maaf jika ada kesalahan dalam penulisan atau istilah asing buat yang awam, materi bootcamp ini akan terus disempurnakan seiring berkembangnya waktu.



Terimakasih