KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

Mata Kuliah : Pemrograman Web Lanjut (PWL)
Program Studi : D4 – Teknik Informatika / D4 – Sistem Informasi Bisnis
Semester : 4 (empat) / 6 (enam)
Pertemuan ke- : 10 (tujuh)

JOBSHEET 10
RESTFUL API

Sebelumnya kita sudah membahas mengenai *authentication*, *authorization*, dan *middleware* pada Laravel. Dimana kita telah membuat fungsi login, register, logout, serta pemilihan role dan penerapan session pada halaman web. Pada pertemuan kali ini, kita akan mempelajari penerapan RESTFUL API di dalam project Laravel.

Sebelum kita masuk materi, kita buat dulu project baru yang akan kita gunakan untuk membangun aplikasi sederhana dengan topik *Point of Sales (PoS)*, sesuai dengan `Studi Kasus PWL.pdf`.
Jadi kita bikin project Laravel 10 dengan nama **PWL_POS.**

Project **PWL_POS** akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajarai

A. RESTFUL API

Representational State Transfer (REST) adalah gaya arsitektur perangkat lunak yang mendefinisikan seperangkat prinsip untuk merancang jaringan aplikasi terdistribusi. RESTful API adalah aplikasi pemrograman antarmuka yang mengikuti prinsip-prinsip REST untuk mentransfer data antara klien dan server.

RESTful API adalah salah satu arsitektur dalam API (*Application Program Interface*) yang menggunakan request HTTP untuk mengakses data. Data diakses dengan menggunakan HTTP method GET, PUT, POST dan DELETE yang merujuk pada operasi pembacaan, pembaruan, pembuatan dan penghapusan pada resource. Selain HTTP method, dalam RESTful atau REST digunakan juga HTTP response untuk mendefinisikan respon data yang dikembalikan. Format respon yang umum digunakan berupa JSON (Javascript Object Notation).

B. JSON Web Token (JWT)

JWT adalah singkatan dari JSON Web Token. Ini adalah standar terbuka (RFC 7519) yang mendefinisikan format token yang kompak dan mandiri untuk mentransfer klaim antara dua pihak. JWT sering digunakan dalam otentikasi dan pertukaran informasi yang aman di lingkungan yang tidak terpercaya, seperti internet.

JWT terdiri dari tiga bagian yang dipisahkan oleh titik ("."): header, payload, dan signature. Setiap bagian ini terdiri dari data JSON yang dienkripsi menggunakan algoritma tertentu dan kemudian disatukan untuk membentuk token yang lengkap. Header berisi jenis token dan tipe algoritma yang digunakan untuk enkripsi. Payload berisi klaim atau informasi yang ingin disampaikan. Signature digunakan untuk memverifikasi bahwa token belum berubah dan datanya berasal dari sumber yang dipercayai.

JWT sering digunakan dalam sistem otentikasi dan otorisasi modern, seperti aplikasi web dan layanan web API, karena fleksibilitasnya dalam menyampaikan informasi terenkripsi secara ringkas.

Kita dapat menggunakan JWT untuk:

- Authentication

  Ketika pengguna melakukan authentication dan mendapatkan token, maka setiap permintaan berikutnya akan menyertakan token tersebut, dan memungkinkan pengguna untuk mengakses route, service, dan resources yang diizinkan.
- Pertukaran informasi

  JSON Web Token adalah cara yang baik untuk mengirimkan informasi antar pihak dengan aman. Dengan token yang sudah ditandatangani dengan algoritma RSA, maka kita bisa tahu siapa yang melakukan request tersebut.

  Berikut adalah cara kerja JWT :

JWT (JSON Web Token) adalah cara untuk mentransfer informasi antara dua pihak secara aman sebagai objek JSON. Ini terdiri dari tiga bagian: header, payload, dan signature. Setelah pengguna berhasil autentikasi, server menghasilkan token JWT yang disematkan dalam permintaan HTTP. Server kemudian memvalidasi token untuk memberikan akses ke sumber daya yang diminta. Ini memberikan autentikasi yang aman dan stateless tanpa memerlukan penyimpanan status sesi di server.

Praktikum 1 – Membuat RESTful API Register

1. Sebelum memulai membuat REST API, terlebih dahulu download aplikasi Postman di https://www.postman.com/downloads.
   Aplikasi ini akan digunakan untuk mengerjakan semua tahap praktikum pada Jobsheet ini.
2. Lakukan instalasi JWT dengan mengetikkan perintah berikut:
   ```
   composer require tymon/jwt-auth:2.1.1
   ```
   Pastikan Anda terkoneksi dengan internet.
3. Setelah berhasil menginstall JWT, lanjutkan dengan publish konfigurasi file dengan perintah berikut:
   ```
   php artisan vendor:publish --
   provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
   ```
4. Jika perintah di atas berhasil, maka kita akan mendapatkan 1 file baru yaitu config/jwt.php. Pada file ini dapat dilakukan konfigurasi jika memang diperlukan.
5. Setelah itu jalankan peintah berikut untuk membuat secret key JWT.
   ```
   php artisan jwt:secret
   ```
   Jika berhasil, maka pada file .env akan ditambahkan sebuah baris berisi nilai key JWT_SECRET.
6. Selanjutnya lakukan konfigurasi guard API. Buka config/auth.php. Ubah bagian ‘guards’ menjadi seperti berikut.

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],
    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
    ],
],
```

7. Kita akan menambahkan kode di model UserModel, ubah kode seperti berikut:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Illuminate\Foundation\Auth\User as Authenticatable;

class UserModel extends Authenticatable implements JWTSubject
{

    public function getJWTIdentifier(){
        return $this->getKey();
    }

    public function getJWTCustomClaims(){
        return [];
    }

    protected $table = 'm_user';
    protected $primaryKey = 'user_id';
```

```php
// Implementasi JWTSubject
0 references | 0 overrides
public function getJWTIdentifier(): mixed
{
    return $this->getKey();
}

0 references | 0 overrides
public function getJWTCustomClaims(): array
{
    return [];
}
```

8. Berikutnya kita akan membuat controller untuk register dengan menjalankan peintah berikut.

   ```
   php artisan make:controller Api/RegisterController
   ```

   Jika berhasil maka akan ada tambahan controller pada folder Api dengan nama RegisterController.
9. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Models\UserModel;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

class RegisterController extends Controller
{
    public function __invoke(Request $request)
    {
```

```
14          //set validation
15          $validator = Validator::make($request->all(), [
16              'username' => 'required',
17              'nama' => 'required',
18              'password' => 'required|min:5|confirmed',
19              'level_id' => 'required'
20          ]);
21
22          //if validations fails
23          if($validator->fails()){
24              return response()->json($validator->errors(), 422);
25          }
26
27          //create user
28          $user = UserModel::create([
29              'username' => $request->username,
30              'nama' => $request->nama,
31              'password'  => bcrypt($request->password),
32              'level_id' => $request->level_id,
33          ]);
34
35          //return response JSON user is created
36          if($user){
37              return response()->json([
38                  'success' => true,
39                  'user' => $user,
40              ], 201);
41          }
42
43          //return JSON process insert failed
44          return response()->json([
45              'success' => false,
46          ], 409);
47      }
48  }
```

```php
<?php

namespace App\Http\Controllers\Api;

use App\Models\UserModel;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

0 references | 0 implementations
class RegisterController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed
    {
        // Set validation
        $validator = Validator::make(data: $request->all(), rules: [
            'username' => 'required',
            'nama' => 'required',
            'password' => 'required|min:5|confirmed',
            'level_id' => 'required',
        ]);

        // If validation fails
        if ($validator->fails()) {
            return response()->json(data: $validator->errors(), status: 422);
        }

        // Create user
        $user = UserModel::create(attributes: [
            'username' => $request->username,
            'nama' => $request->nama,
            'password' => bcrypt(value: $request->password),
            'level_id' => $request->level_id,
        ]);

        // Return response JSON if user is created
        if ($user) {
            return response()->json(data: [
                'success' => true,
                'user' => $user,
            ], status: 201);
        }

        // Return JSON if insert failed
        return response()->json(data: [
            'success' => false,
        ], status: 409);
    }
}
```

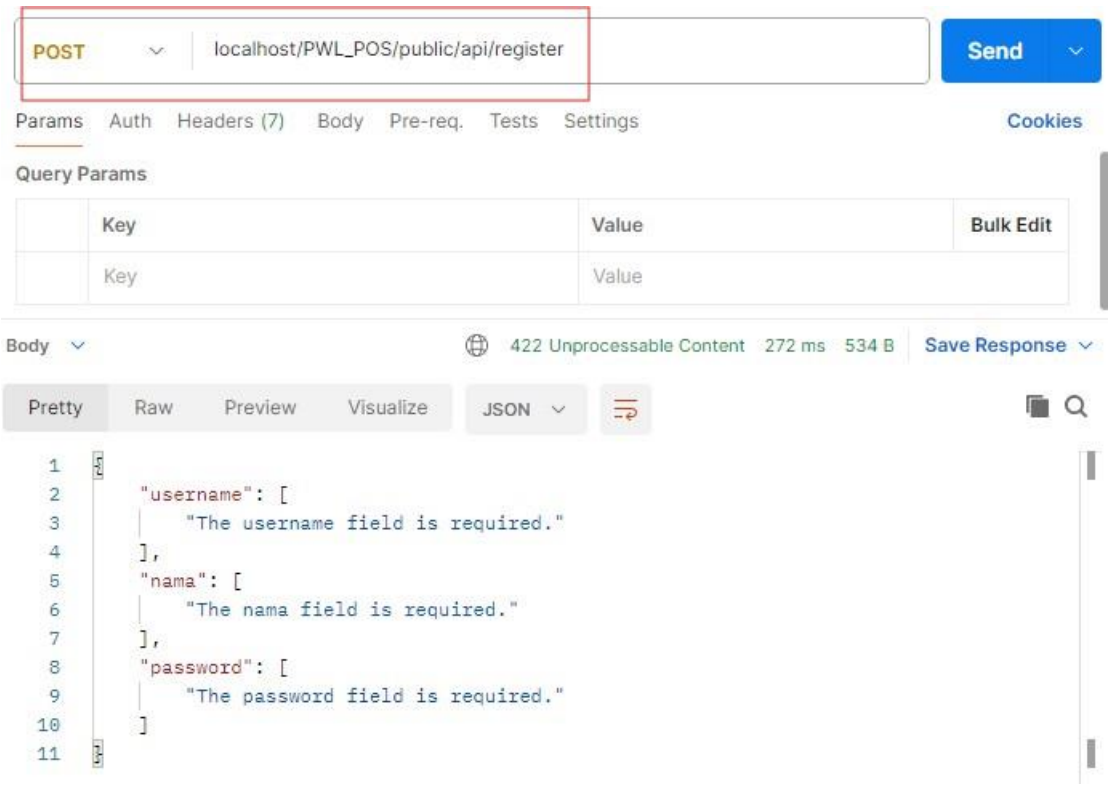
10. Selanjutnya buka routes/api.php, ubah semua kode menjadi seperti berikut.

```php
<?php

use App\Http\Controllers\Api\RegisterController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/

Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
```

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\RegisterController;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/

Route::post(uri: '/register', action: RegisterController::class)->name(name: 'register');

Route::middleware(middleware: 'auth:sanctum')->get(uri: '/user', action: function (Request $request): mixed {
    return $request->user();
});
```

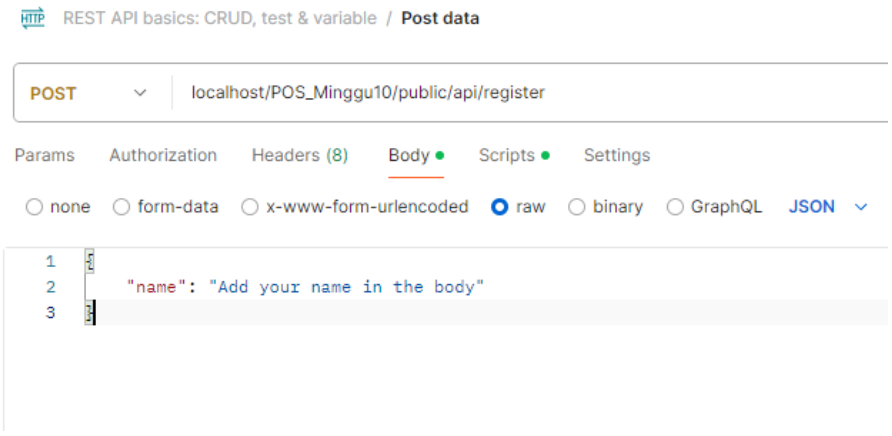
11. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman.

Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/register serta method POST. Klik Send.

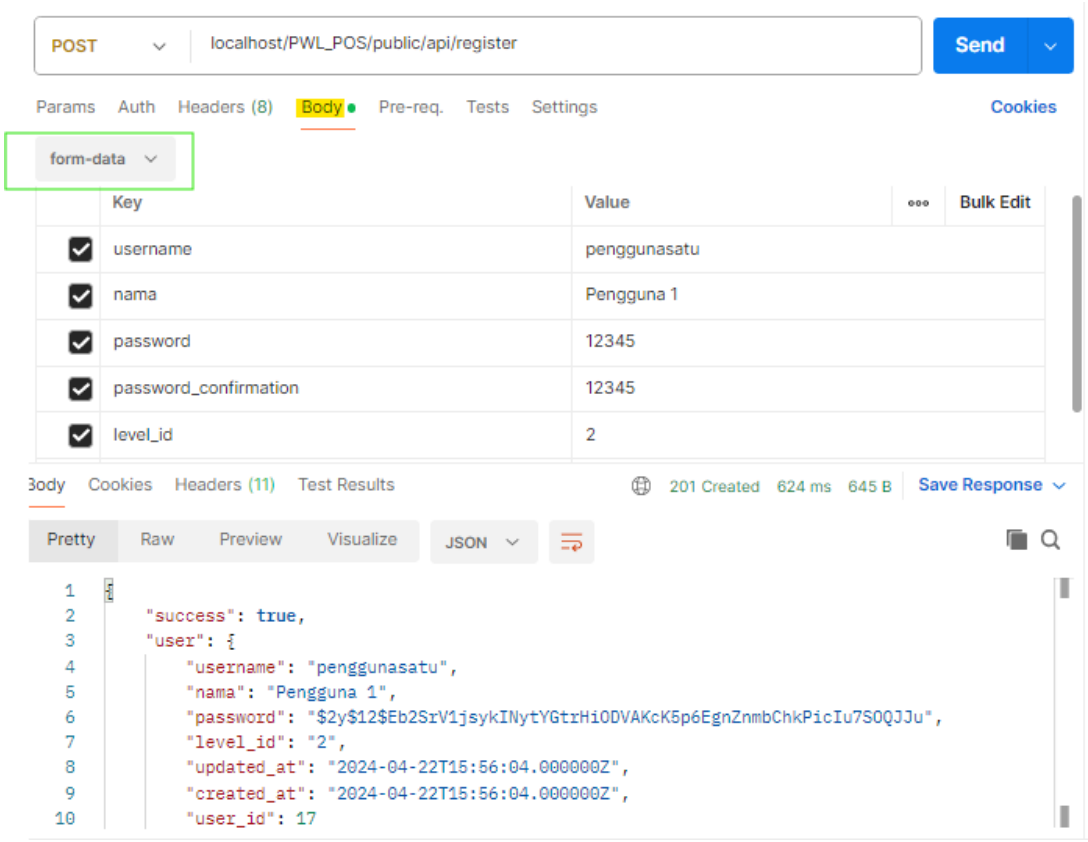


Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

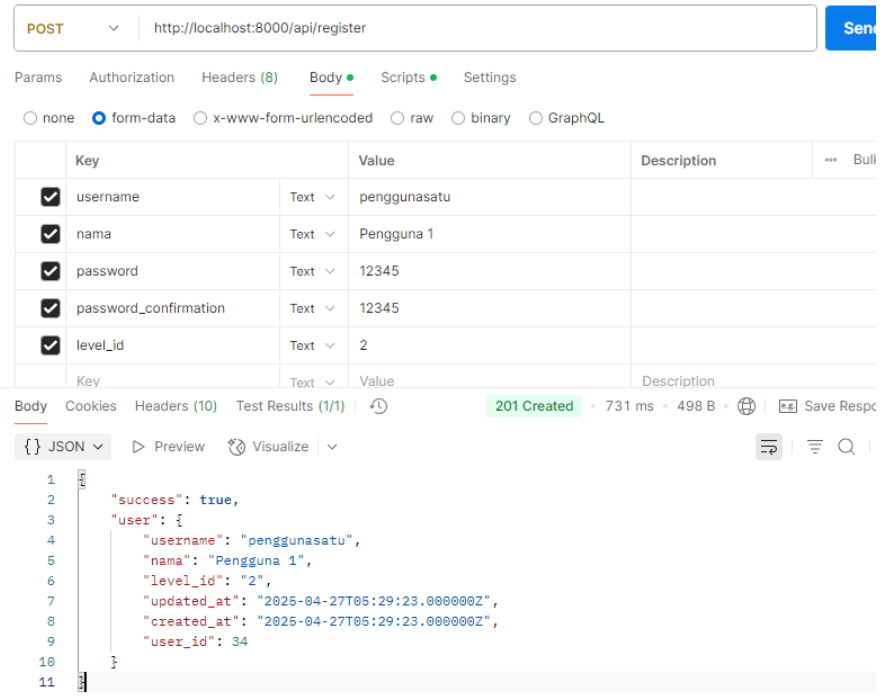


12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.



Setelah klik tombol Send, jika berhasil maka akan keluar pesan sukses seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



13. Lakukan commit perubahan file pada Github.

Praktikum 2 – Membuat RESTful API Login

1. Kita buat file controller dengan nama LoginController.
   ```
   php artisan make:controller Api/LoginController
   ```
   Jika berhasil maka akan ada tambahan controller pada folder Api dengan nama LoginController.
2. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

```

```php
class LoginController extends Controller
{
    public function __invoke(Request $request)
    {
        //set validation
        $validator = Validator::make($request->all(), [
            'username'     => 'required',
            'password'  => 'required'
        ]);

        //if validation fails
        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        //get credentials from request
        $credentials = $request->only('username', 'password');

        //if auth failed
        if(!$token = auth()->guard('api')->attempt($credentials)) {
            return response()->json([
                'success' => false,
                'message' => 'Username atau Password Anda salah'
            ], 401);
        }

        //if auth success
        return response()->json([
            'success' => true,
            'user'    => auth()->guard('api')->user(),
            'token'   => $token
        ], 200);
    }
}
```

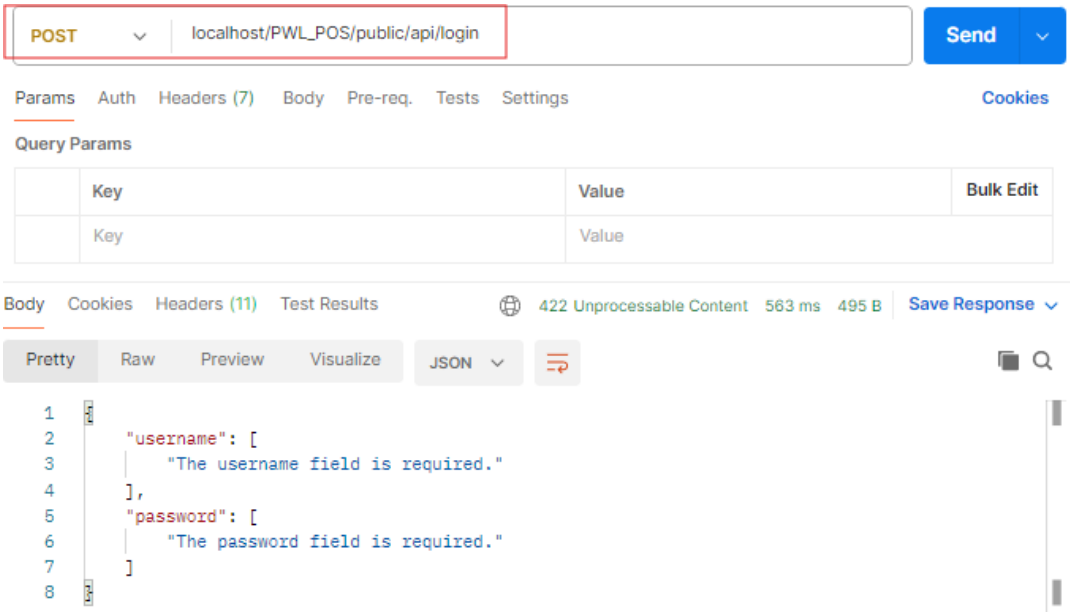
```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

0 references | 0 implementations
class LoginController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed
    {
        // set validation
        $validator = Validator::make(data: $request->all(), rules: [
            'username' => 'required',
            'password' => 'required',
        ]);

        // if validation fails
        if ($validator->fails()) {
            return response()->json(data: $validator->errors(), status: 422);
        }

        // get credentials from request
        $credentials = $request->only(keys: 'username', 'password');

        // if auth failed
        if (!$token = auth()->guard(name: 'api')->attempt(credentials: $credentials)) {
            return response()->json(data: [
                'success' => false,
                'message' => 'Username atau Password Anda salah'
            ], status: 401);
        }

        // if auth success
        return response()->json(data: [
            'success' => true,
            'user' => auth()->guard(name: 'api')->user(),
            'token' => $token
        ], status: 200);
    }
}
```

3. Berikutnya tambahkan route baru pada file api.php yaitu /login dan /user.

```php
use App\Http\Controllers\Api\LoginController;

Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
Route::post('/login', App\Http\Controllers\Api\LoginController::class)->name('login');
Route::middleware('auth:api')->get('/user', function (Request $request) {
    return $request->user();
});
```

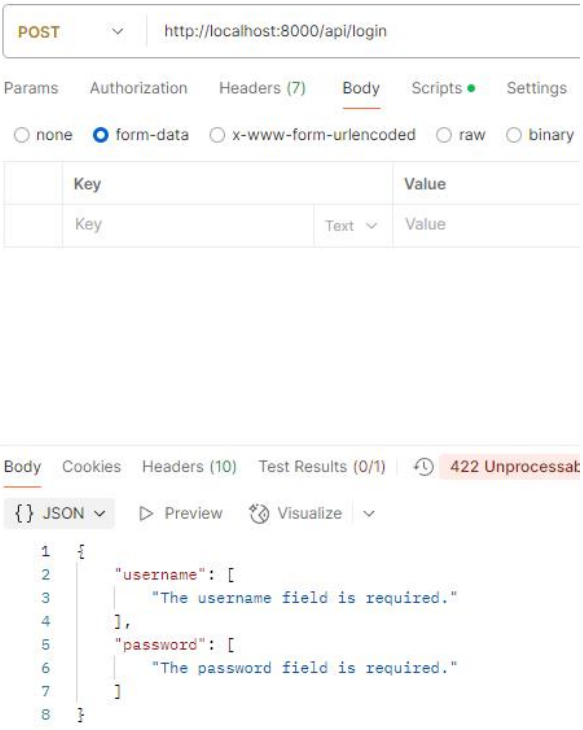
```php
// Route register dan login
Route::post(uri: '/register', action: RegisterController::class)->name(name: 'register');
Route::post(uri: '/login', action: LoginController::class)->name(name: 'login');
```

4. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/login serta method POST. Klik Send.

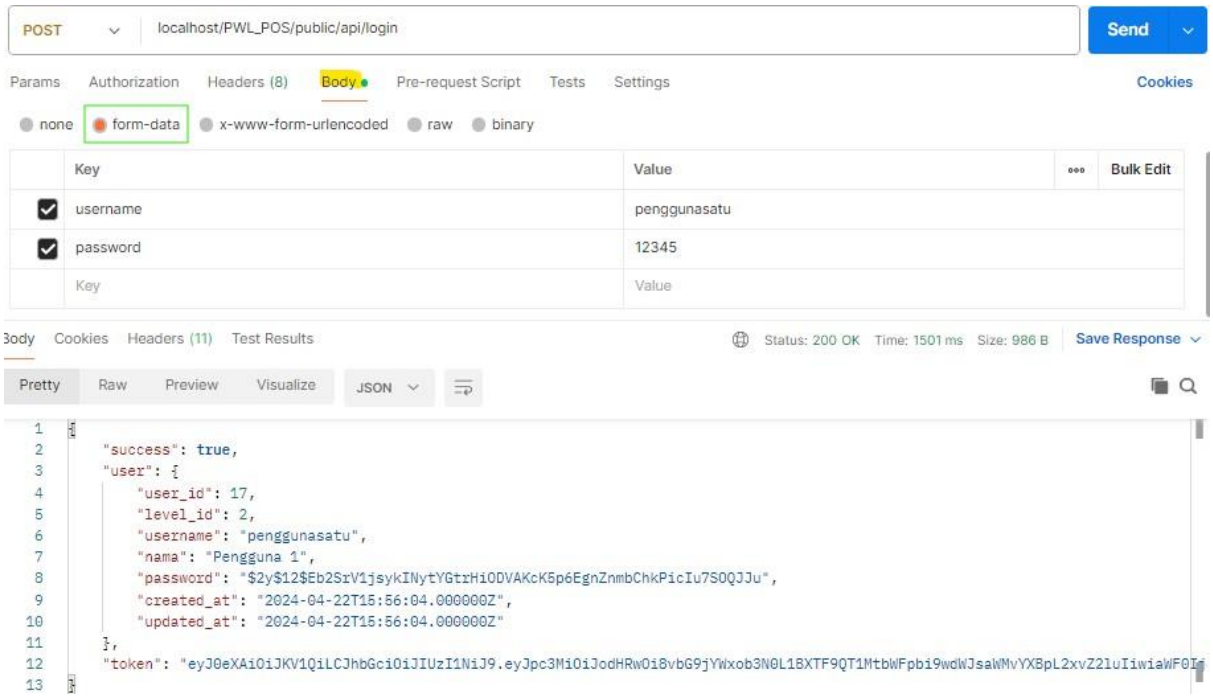


Jika berhasil akan muncul error validasi seperti gambar di atas.

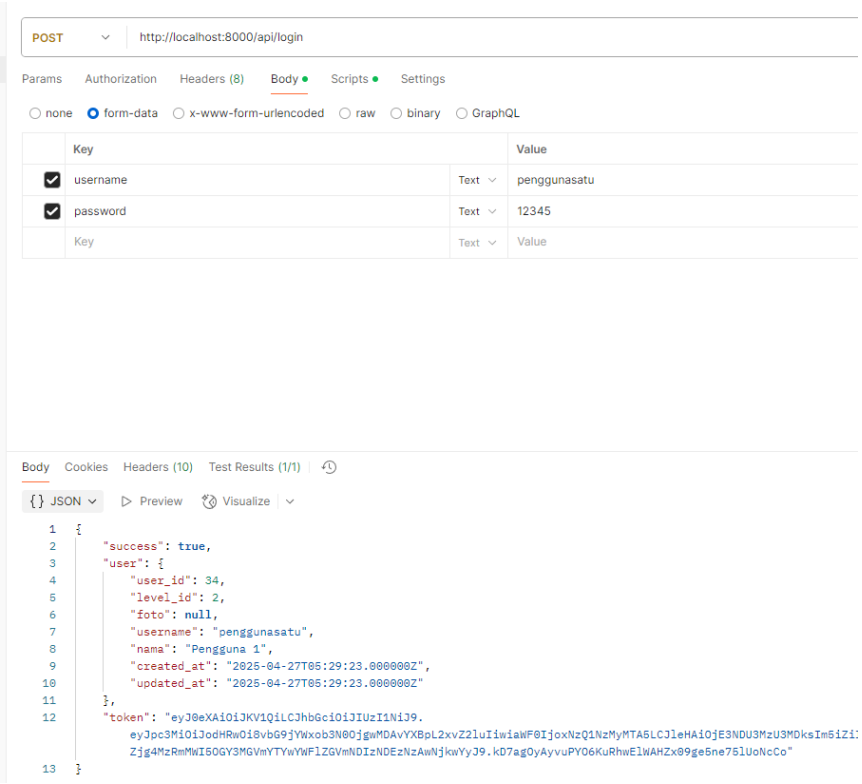
Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



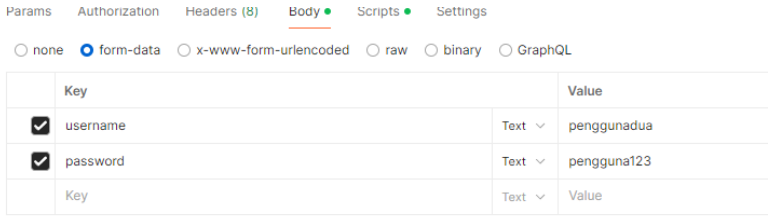
5. Selanjutnya, isikan username dan password sesuai dengan data user yang ada pada database. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data user. Klik tombol Send, jika berhasil maka akan keluar tampilan seperti berikut. Copy nilai token yang diperoleh pada saat login karena akan diperlukan pada saat logout.

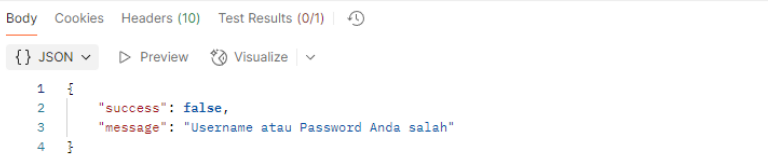

POST
localhost/PWL_POS/public/api/login
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
username
penggunasatu
password
12345
Key
Value
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 1501 ms
Size: 986 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"user": {
"user_id": 17,
"level_id": 2,
"username": "penggunasatu",
"nama": "Pengguna 1",
"password": "$2y$12$Eb2SrV1jsykINytYGtrHiODVAKcK5p6EgnZnmbChkPicIu7SOQJJu",
"created_at": "2024-04-22T15:56:04.000000Z",
"updated_at": "2024-04-22T15:56:04.000000Z"
},
"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0L18XTF9QT1MtbWFpbi9wdWJsaWMvYXBpL2xvZ2luIiwiaWF0Ij
}

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

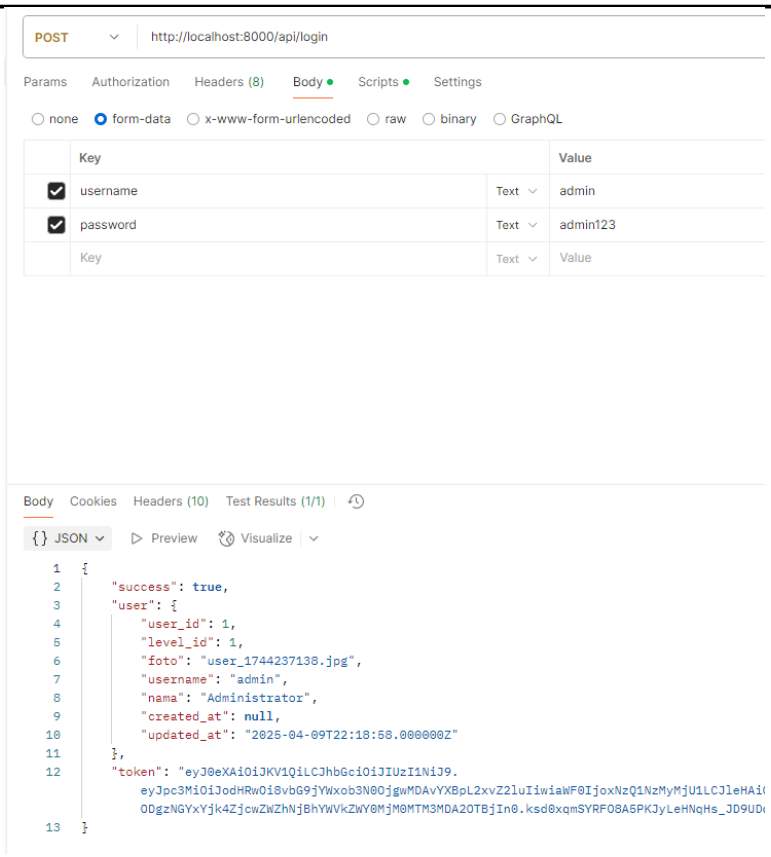


6. Lakukan percobaan yang untuk data yang salah dan berikan screenshoot hasil percobaan Anda.





7. Coba kembali melakukan login dengan data yang benar. Sekarang mari kita coba menampilkan data user yang sedang login menggunakan URL localhost/PWL_POS/public/api/user dan method GET. Jelaskan hasil dari percobaan tersebut.



8. Lakukan commit perubahan file pada Github.

Praktikum 3 – Membuat RESTful API Logout

1. Tambahkan kode berikut pada file .env

```
JWT_SHOW_BLACKLIST_EXCEPTION=true
```

2. Buat Controller baru dengan nama LogoutController.

```
php artisan make:controller Api/LogoutController
```

3. Buka file tersebut dan ubah kode menjadi seperti berikut.

```php
<?php

namespace App\Http\Controllers\Api;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Tymon\JWTAuth\Facades\JWTAuth;
use Tymon\JWTAuth\Exceptions\JWTException;
use Tymon\JWTAuth\Exceptions\TokenExpiredException;
use Tymon\JWTAuth\Exceptions\TokenInvalidException;

class LogoutController extends Controller
{
    public function __invoke(Request $request)
    {
        //remove token
        $removeToken = JWTAuth::invalidate(JWTAuth::getToken());

        if($removeToken) {
            //return response JSON
            return response()->json([
                'success' => true,
                'message' => 'Logout Berhasil!',
            ]);
        }
    }
}
```

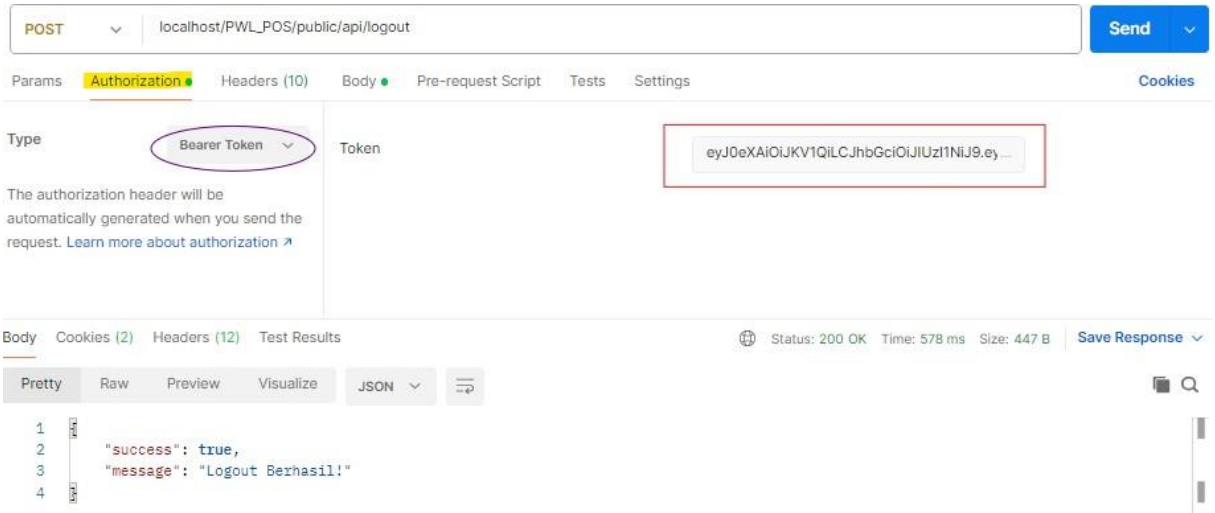
```php
<?php

namespace App\Http\Controllers\Api;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Tymon\JWTAuth\Facades\JWTAuth;
use Tymon\JWTAuth\Exceptions\JWTException;

0 references | 0 implementations
class LogoutController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed
    {
        try {
            // remove token
            $removeToken = JWTAuth::invalidate(JWTAuth::getToken());

            if ($removeToken) {
                // return response JSON
                return response()->json(data: [
                    'success' => true,
                    'message' => 'Logout Berhasil!',
                ]);
            }
        } catch (JWTException $e) {
            return response()->json(data: [
                'success' => false,
                'message' => 'Logout gagal!',
            ], status: 500);
        }
    }
}
```

4. Lalu kita tambahkan routes pada api.php

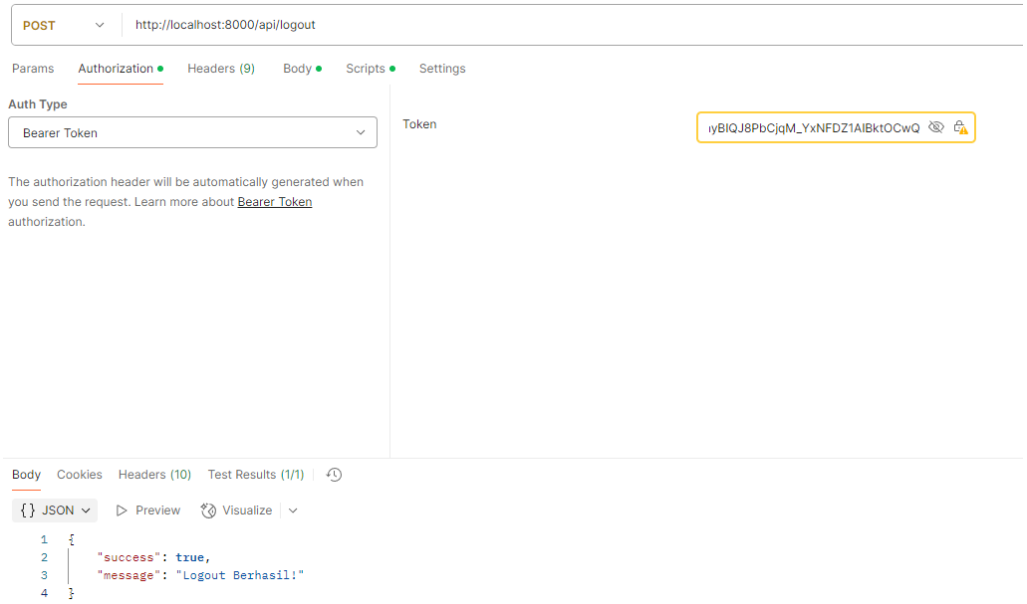
```
Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
```

```
// Route register dan login
Route::post(uri: '/register', action: RegisterController::class)->name(name: 'register');
Route::post(uri: '/login', action: LoginController::class)->name(name: 'login');
Route::post(uri: '/logout', action: App\Http\Controllers\Api\LogoutController::class)->name(name: 'logout');
```

5. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/logout serta method POST.
6. Isi token pada tab Authorization, pilih Type yaitu Bearer Token. Isikan token yang didapat saat login. Jika sudah klik Send.



Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



7. Lakukan commit perubahan file pada Github.

Praktikum 4 – Implementasi CRUD dalam RESTful API

Pada praktikum ini kita akan menggunakan tabel m_level untuk dimodifikasi menggunakan RESTful API.

1. Pertama, buat controller untuk mengolah API pada data level.
   ```
   php artisan make:controller Api/LevelController
   ```
2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya.

```php
namespace App\Http\Controllers\Api;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\LevelModel;

class LevelController extends Controller
{
    public function index()
    {
        return LevelModel::all();
    }
```

```
    public function store(Request $request)
    {
        $level = LevelModel::create($request->all());
        return response()->json($level, 201);
    }

    public function show(LevelModel $level)
    {
        return LevelModel::find($level);
    }

    public function update(Request $request, LevelModel $level)
    {
        $level->update($request->all());
        return LevelModel::find($level);
    }

    public function destroy(LevelModel $user)
    {
        $user->delete();

        return response()->json([
            'success' => true,
            'message' => 'Data terhapus',
        ]);
    }
}
```

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\LevelModel;

0 references | 0 implementations
class LevelController extends Controller
{
    // Menampilkan semua data level
    0 references | 0 overrides
    public function index(): Collection
    {
        return LevelModel::all();
    }

    // Menyimpan data level baru
    0 references | 0 overrides
    public function store(Request $request): JsonResponse|mixed
    {
        $level = LevelModel::create(attributes: $request->all());
        return response()->json(data: $level, status: 201);
    }

    // Menampilkan detail satu data level
    0 references | 0 overrides
    public function show($id): JsonResponse|mixed
    {
        $level = LevelModel::find(id: $id);
        return response()->json(data: $level);
    }

    // Mengupdate data level
    0 references | 0 overrides
    public function update(Request $request, $id): JsonResponse|mixed
    {
        $level = LevelModel::find(id: $id);
        $level->update(attributes: $request->all());
        return response()->json(data: $level);
    }

    // Menghapus data level
    0 references | 0 overrides
    public function destroy($id): JsonResponse|mixed
    {
        $level = LevelModel::find(id: $id);
        $level->delete();
        return response()->json(data: [
            'success' => true,
            'message' => 'Data terhapus',
        ]);
    }
}
```

3. Kemudian kita lengkapi routes pada api.php.

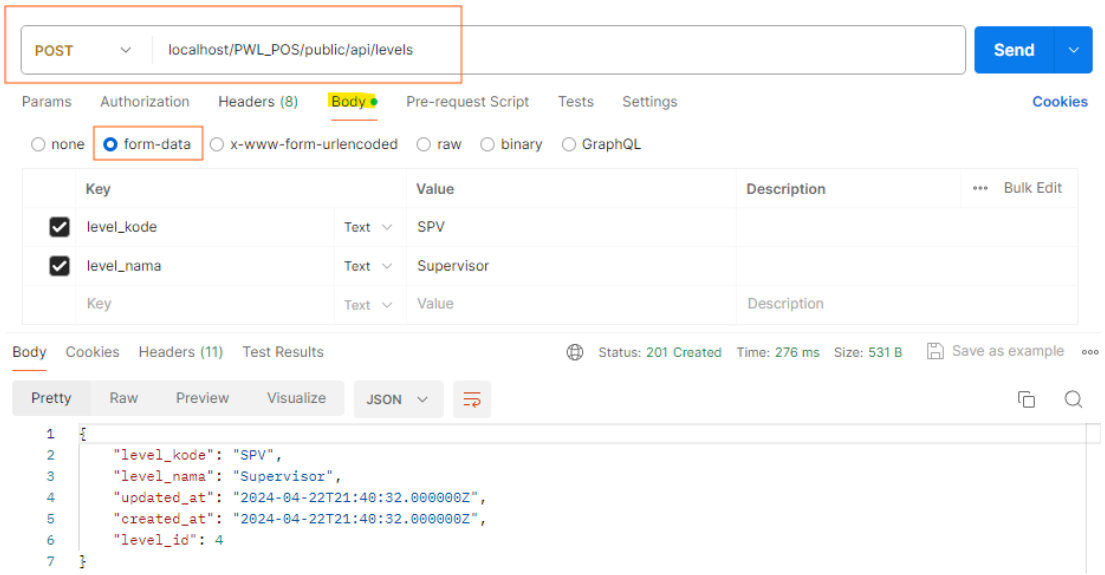
```php
use App\Http\Controllers\Api\LevelController;

Route::get('levels', [LevelController::class, 'index']);
Route::post('levels', [LevelController::class, 'store']);
Route::get('levels/{level}', [LevelController::class, 'show']);
Route::put('levels/{level}', [LevelController::class, 'update']);
Route::delete('levels/{level}', [LevelController::class, 'destroy']);
```

4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/levels dan method GET. Jelaskan dan berikan screenshoot hasil percobaan Anda.

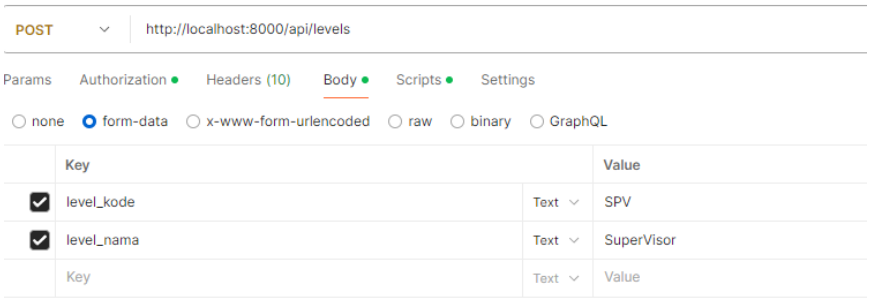
```
[
    {
        "level_id": 1,
        "level_kode": "ADM",
        "level_nama": "Administrator",
        "created_at": null,
        "updated_at": null
    },
    {
        "level_id": 2,
        "level_kode": "MNG",
        "level_nama": "Manager",
        "created_at": null,
        "updated_at": null
    },
    {
        "level_id": 3,
        "level_kode": "STF",
        "level_nama": "Staff/Kasir",
        "created_at": null,
        "updated_at": null
    },
    {
        "level_id": 5,
        "level_kode": "CUS",
        "level_nama": "Pelanggan",
        "created_at": "2025-03-02T14:45:16.000000Z",
        "updated_at": null
    },
]
```

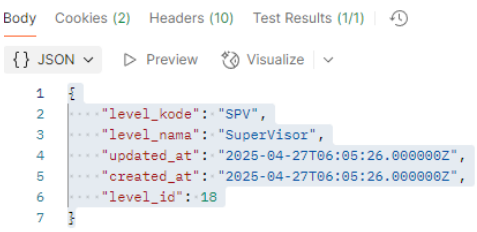

server mengembalikan data berisi daftar Level yang sudah ada di database.

5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POS-main/public/api/levels dan method POST seperti di bawah ini.



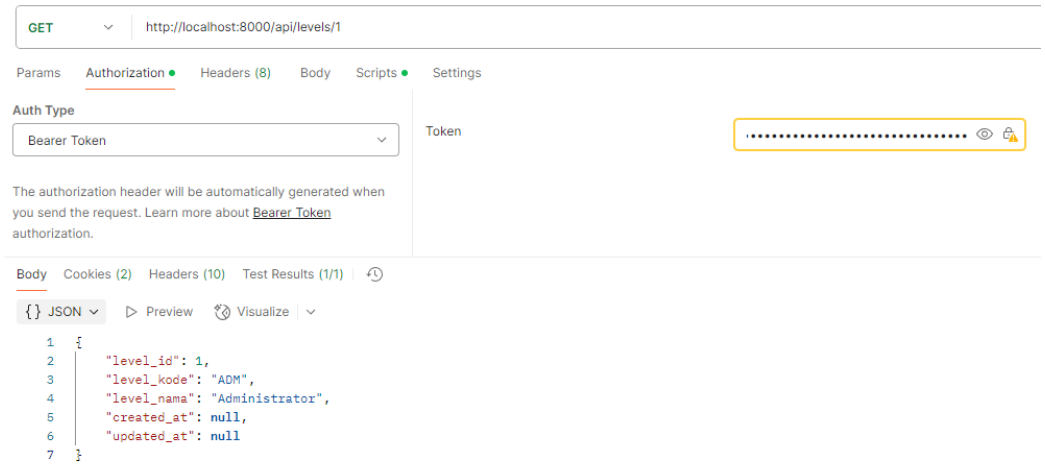
Jelaskan dan berikan screenshoot hasil percobaan Anda.





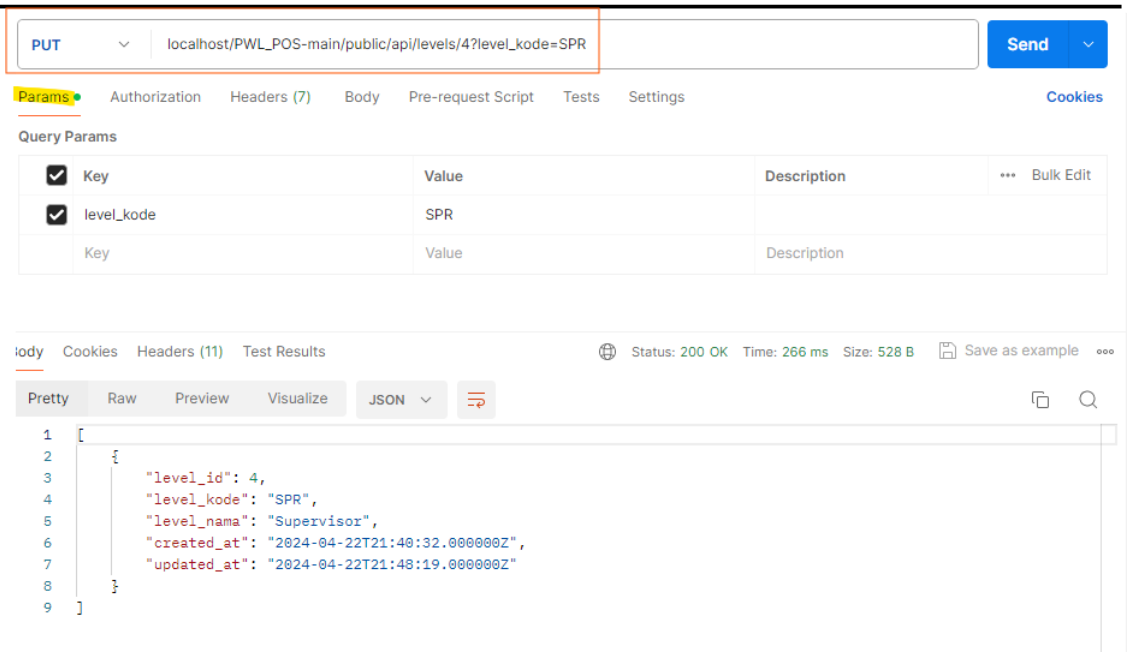
server menerima data baru dan menyimpan data Level ke dalam database, serta mengembalikan response JSON berisi data Level yang baru dibuat.

6. Berikutnya lakukan percobaan menampilkan detail data. Jelaskan dan berikan screenshoot hasil percobaan Anda.



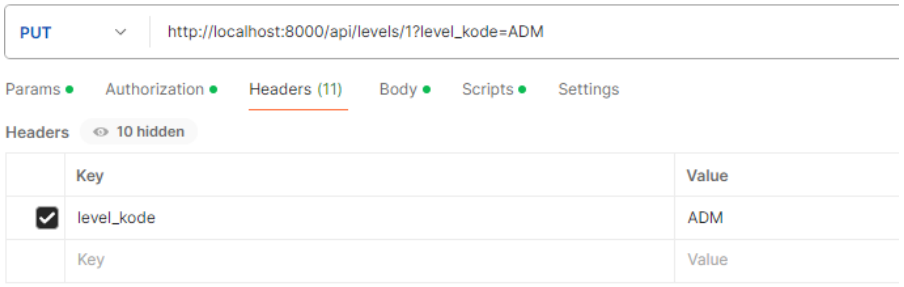
Data yang dikembalikan berisi informasi id, level_kode, level_nama, created_at, dan updated_at.

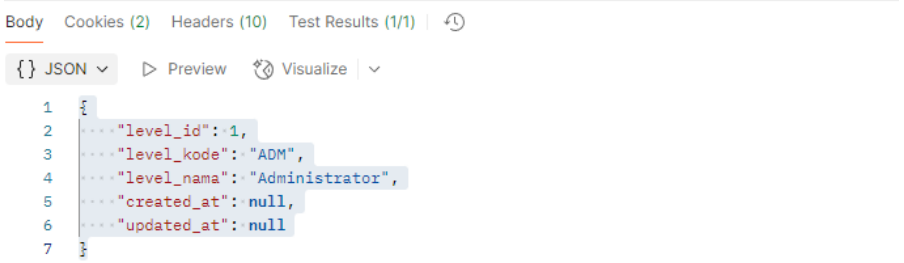
7. Jika sudah, kita coba untuk melakukan edit data menggunakan localhost/PWL_POS-main/public/api/levels/{id} dan method PUT. Isikan data yang ingin diubah pada tab Param.



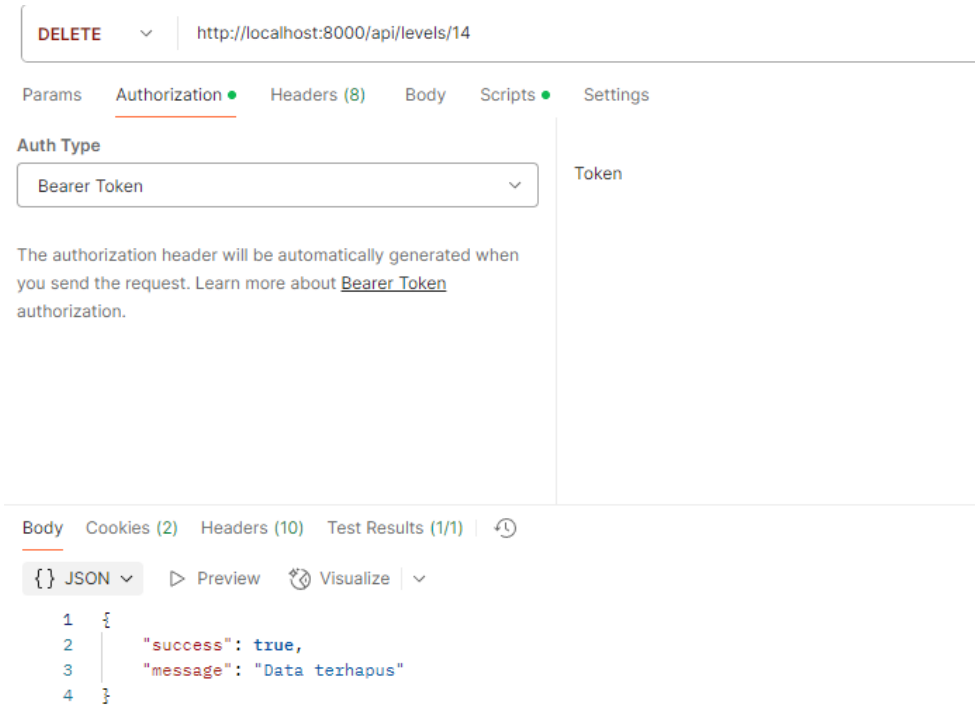
Jelaskan dan berikan screenshoot hasil percobaan Anda.

server berhasil mengupdate data Level yang sesuai ID





8. Terakhir lakukan percobaan hapus data. Jelaskan dan berikan screenshoot hasil percobaan Anda.



Server menghapus data Level sesuai ID yang dikirimkan.

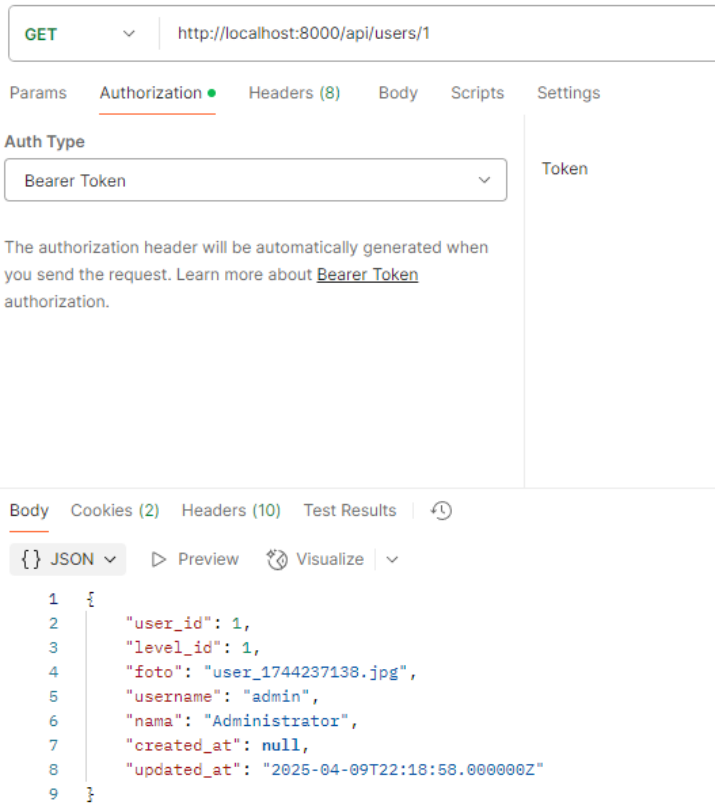
9. Lakukan commit perubahan file pada Github.

TUGAS

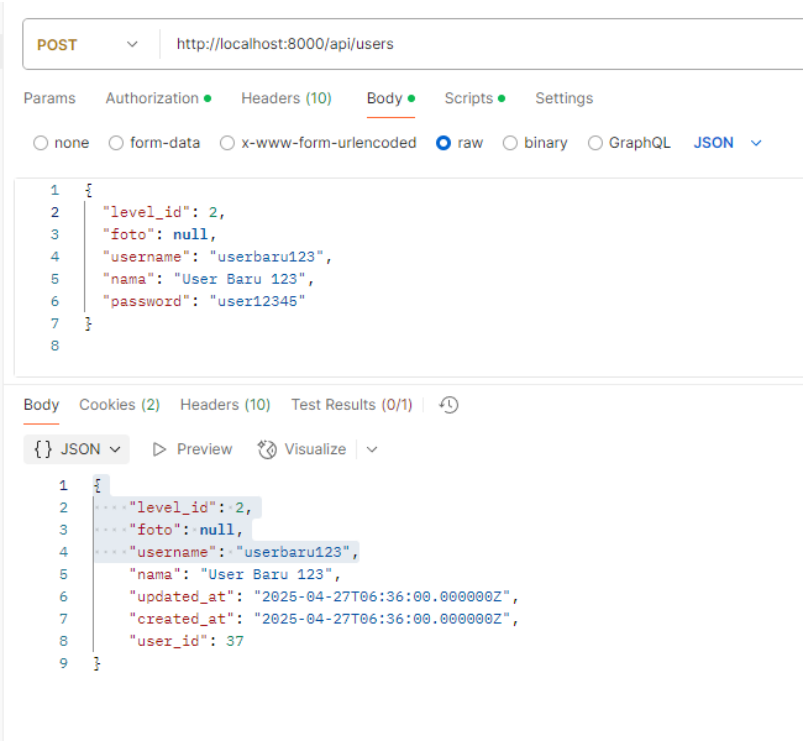
Implementasikan CRUD API pada tabel lainnya yaitu tabel m_user, m_kategori, dan m_barang

M_user

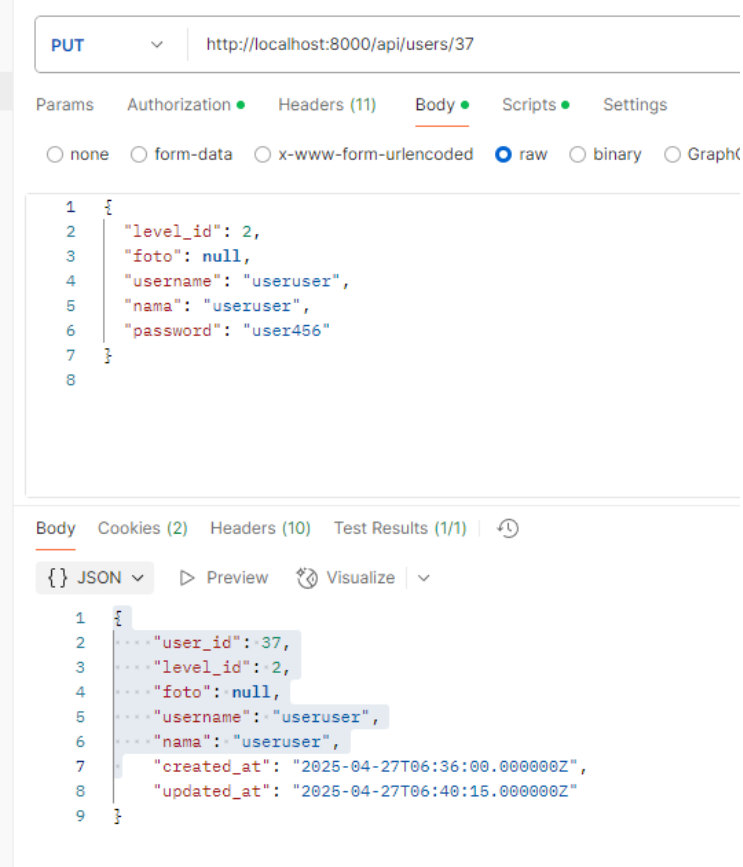
Get



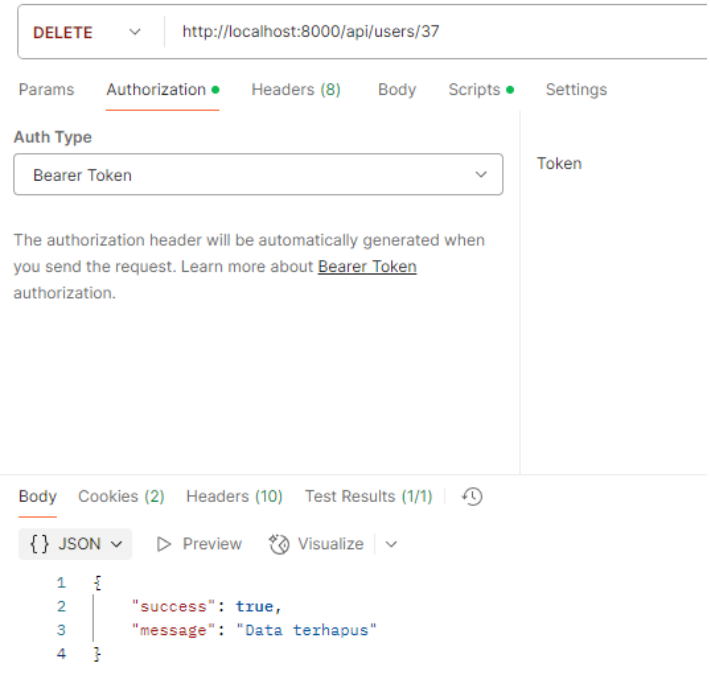
Post



Put

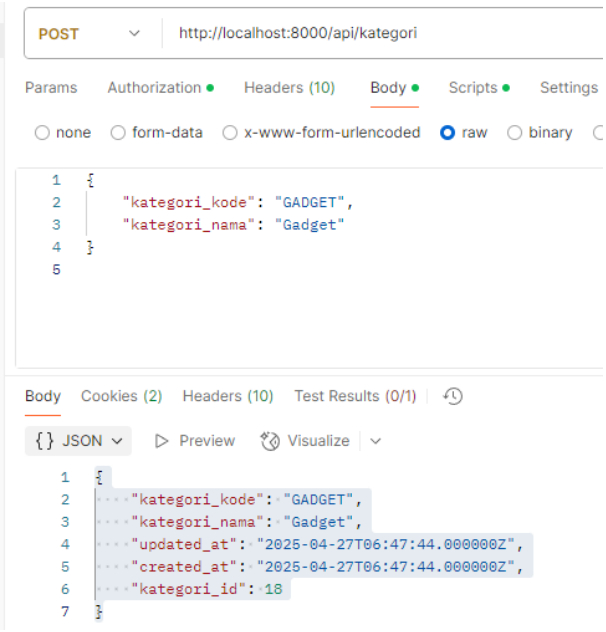

PUT
http://localhost:8000/api/users/37
Params
Authorization
Headers (11)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
{
"level_id": 2,
"foto": null,
"username": "useruser",
"nama": "useruser",
"password": "user456"
}
Body
Cookies (2)
Headers (10)
Test Results (1/1)
{} JSON
Preview
Visualize
{
"user_id": 37,
"level_id": 2,
"foto": null,
"username": "useruser",
"nama": "useruser",
"created_at": "2025-04-27T06:36:00.000000Z",
"updated_at": "2025-04-27T06:40:15.000000Z"
}

Delete

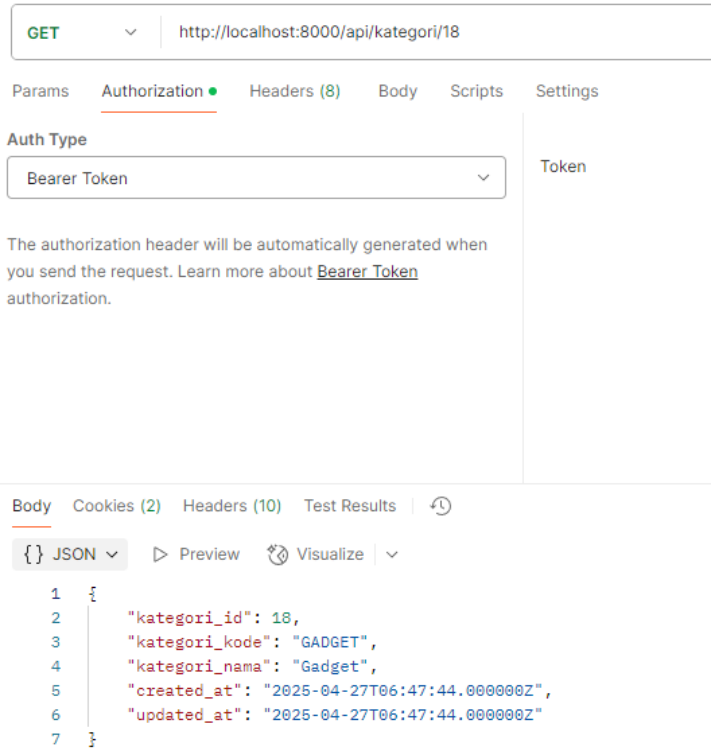

DELETE
http://localhost:8000/api/users/37
Params
Authorization
Headers (8)
Body
Scripts
Settings
Auth Type
Bearer Token
Token
The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.
Body
Cookies (2)
Headers (10)
Test Results (1/1)
{} JSON
Preview
Visualize
{
"success": true,
"message": "Data terhapus"
}

M_kategori

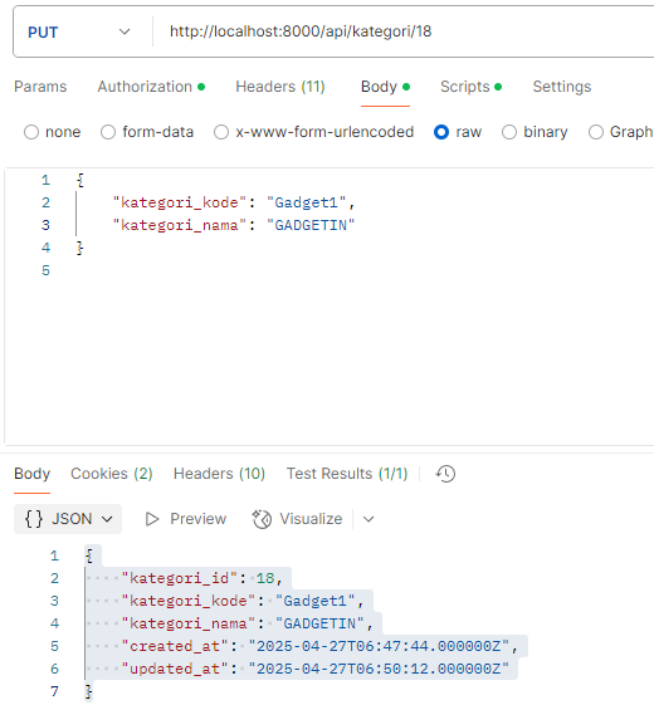
Post



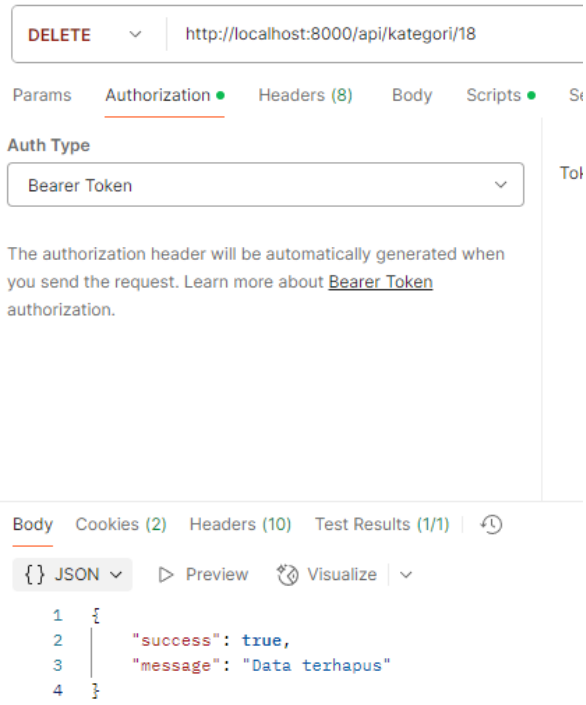
Get



Put

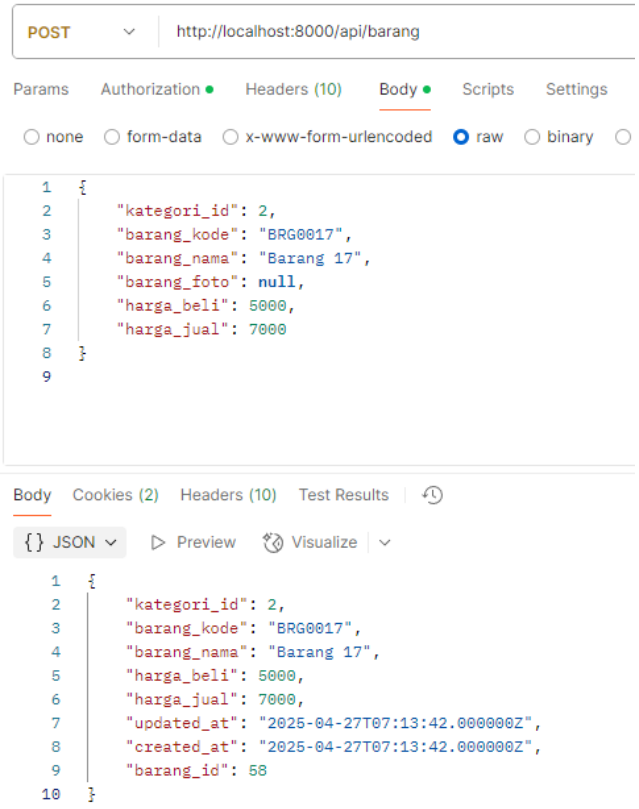


Delete

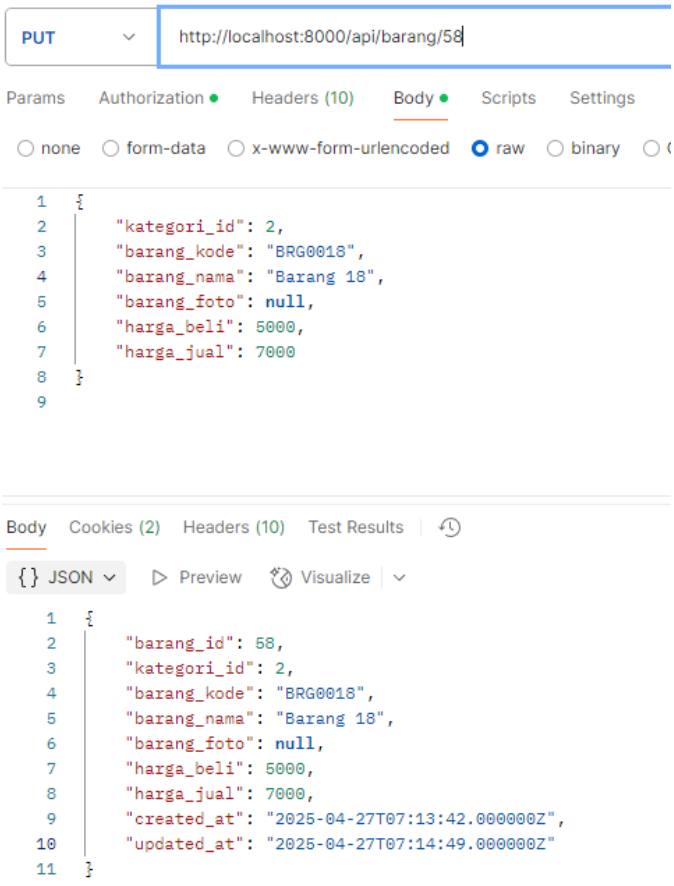


M_barang

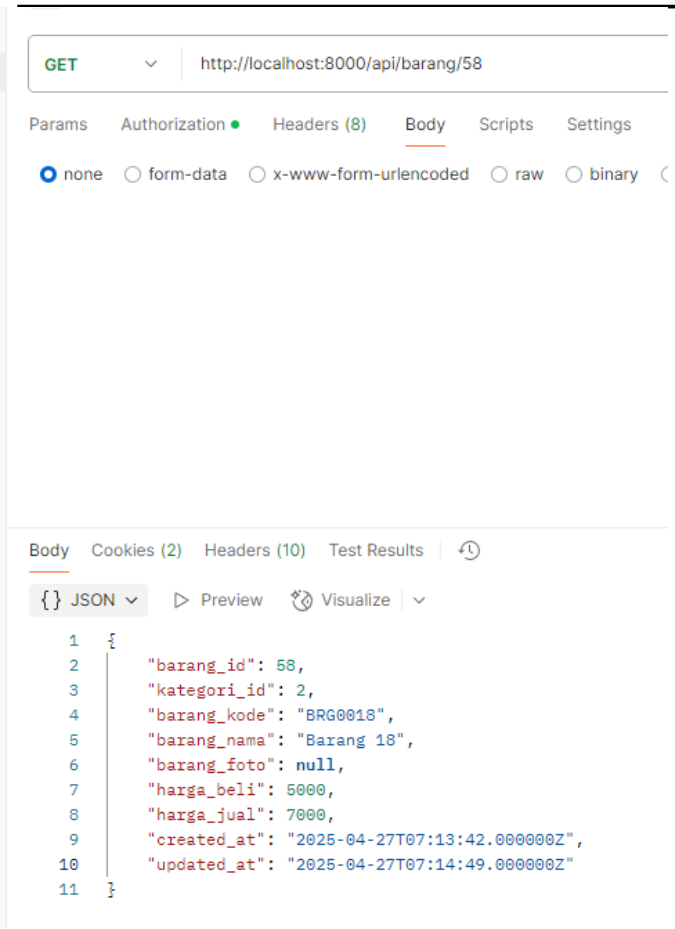
Post


POST
http://localhost:8000/api/barang
Params
Authorization
Headers (10)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
{
"kategori_id": 2,
"barang_kode": "BRG0017",
"barang_nama": "Barang 17",
"barang_foto": null,
"harga_beli": 5000,
"harga_jual": 7000
}
Body
Cookies (2)
Headers (10)
Test Results
{} JSON
Preview
Visualize
{
"kategori_id": 2,
"barang_kode": "BRG0017",
"barang_nama": "Barang 17",
"harga_beli": 5000,
"harga_jual": 7000,
"updated_at": "2025-04-27T07:13:42.000000Z",
"created_at": "2025-04-27T07:13:42.000000Z",
"barang_id": 58
}

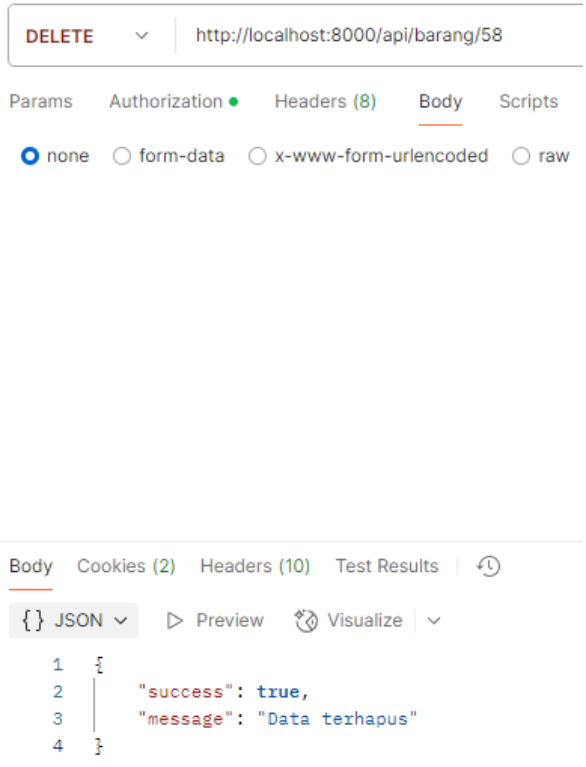
Put


PUT
http://localhost:8000/api/barang/58
Params
Authorization
Headers (10)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
{
"kategori_id": 2,
"barang_kode": "BRG0018",
"barang_nama": "Barang 18",
"barang_foto": null,
"harga_beli": 5000,
"harga_jual": 7000
}
Body
Cookies (2)
Headers (10)
Test Results
JSON
Preview
Visualize
{
"barang_id": 58,
"kategori_id": 2,
"barang_kode": "BRG0018",
"barang_nama": "Barang 18",
"barang_foto": null,
"harga_beli": 5000,
"harga_jual": 7000,
"created_at": "2025-04-27T07:13:42.000000Z",
"updated_at": "2025-04-27T07:14:49.000000Z"
}

Get


GET
http://localhost:8000/api/barang/58
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
Body
Cookies (2)
Headers (10)
Test Results
JSON
Preview
Visualize
{
"barang_id": 58,
"kategori_id": 2,
"barang_kode": "BRG0018",
"barang_nama": "Barang 18",
"barang_foto": null,
"harga_beli": 5000,
"harga_jual": 7000,
"created_at": "2025-04-27T07:13:42.000000Z",
"updated_at": "2025-04-27T07:14:49.000000Z"
}

Delete


DELETE
http://localhost:8000/api/barang/58
Params
Authorization
Headers (8)
Body
Scripts
none
form-data
x-www-form-urlencoded
raw
Body
Cookies (2)
Headers (10)
Test Results
JSON
Preview
Visualize
{
"success": true,
"message": "Data terhapus"
}

**** Sekian, dan selamat belajar ****