



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 pelajari

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.

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> composer require tymon/jwt-auth:2.1.1
./composer.json has been updated
Running composer update tymon/jwt-auth
Loading composer repositories with package information
Updating dependencies
Lock file operations: 4 installs, 0 updates, 0 removals
- Locking lcobucci/clock (2.2.0)
- Locking lcobucci/jwt (4.0.4)
- Locking stella-maris/clock (0.1.7)
- Locking tymon/jwt-auth (2.1.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 4 installs, 0 updates, 0 removals
- Downloading stella-maris/clock (0.1.7)
- Downloading lcobucci/clock (2.2.0)
- Downloading lcobucci/jwt (4.0.4)
- Downloading tymon/jwt-auth (2.1.1)
- Installing stella-maris/clock (0.1.7): Extracting archive
- Installing lcobucci/clock (2.2.0): Extracting archive
- Installing lcobucci/jwt (4.0.4): Extracting archive
- Installing tymon/jwt-auth (2.1.1): Extracting archive
Generating optimized autoload files
> illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

barryvdh/laravel-dumpdf ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
laravel/ui ..... DONE
nesbot/carbon ..... DONE
```

3. Setelah berhasil menginstall JWT, lanjutkan dengan publish konfigurasi file dengan perintah berikut:

```
php artisan vendor:publish --
```

```
provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
```

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan vendor:publish --provider="Tymon\JWTAuth\Pr
viders\LaravelServiceProvider"

 INFO  Publishing assets.

Copying file [D:\Apps\laragon\www\PWL2025\week 10\jobsheet\vendor\tymon\jwt-auth\config\config.php] to
[D:\Apps\laragon\www\PWL2025\week 10\jobsheet\config\jwt.php] DONE
```

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 perintah 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.



```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan jwt:secret  
jwt-auth secret [MaIzmITYBINSHvQPk7XvVgXwBUfQfXPu8bJzp7lSIkLungLoQM0Nb6zG1ET2RpRx] set successfully.  
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet>
```

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

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



```
<?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';
}

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

28 references | 0 implementations | You, 1 second ago | 1 author (You)
class UserModel extends Authenticatable implements JWTSubject
{
    0 references | 0 overrides
    public function getJWTIdentifier(): string|int
    {
        return $this->getKey();
    }

    0 references | 0 overrides
    public function getJWTCustomClaims(): array You, 1 second ago
    {
        return [];
    }
}
```

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

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

Jika berhasil maka akan ada tambahan controller pada folder Api dengan nama RegisterController.



```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan make:controller Api/RegisterController
```

INFO Controller [D:\Apps\laragon\www\PWL2025\week 10\jobsheet\app\Http\Controllers\Api/RegisterController.php] created successfully.

9. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Models\UserModel;
6  use App\Http\Controllers\Controller;
7  use Illuminate\Http\Request;
8  use Illuminate\Support\Facades\Validator;
9
10 class RegisterController extends Controller
11 {
12     public function __invoke(Request $request)
13     {
```

```
<?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
    {
```



```
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 }
```



```
$validator = Validator::make($request->all(), [
    'username' => 'required',
    'nama' => 'required',
    'password' => 'required|min:5|confirmed',
    'level_id' => 'required'
]);

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

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

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

//return JSON process insert failed
return response()->json([
    'success' => false,
], 409);
}
```

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



```
<?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!
|
*/

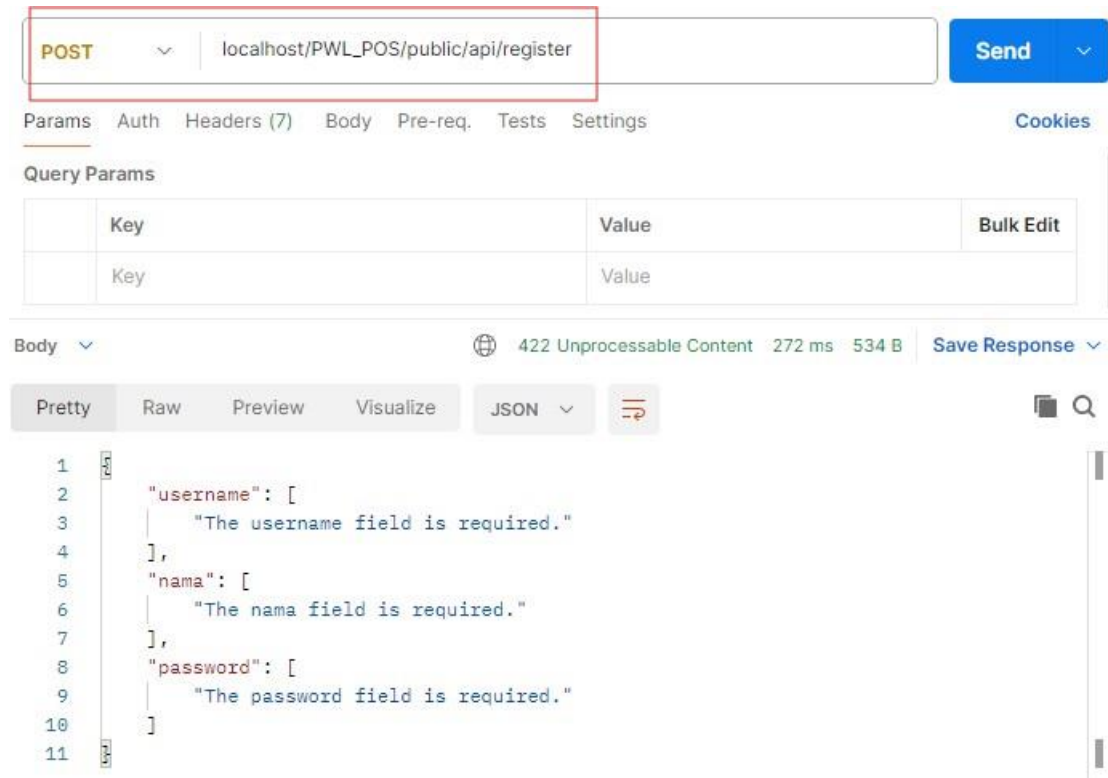
Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');

routes > api.php
1  <?php
2
3  use Illuminate\Http\Request;
4  use Illuminate\Support\Facades\Route;
5
6  /*
7  |-----
8  | API Routes
9  |-----
10 |
11 | Here is where you can register API routes for your application. These
12 | routes are loaded by the RouteServiceProvider and all of them will
13 | be assigned to the "api" middleware group. Make something great!
14 |
15 */
16
17 Route::post('/register', \App\Http\Controllers\Api\RegisterController::class)->name('register');
```

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

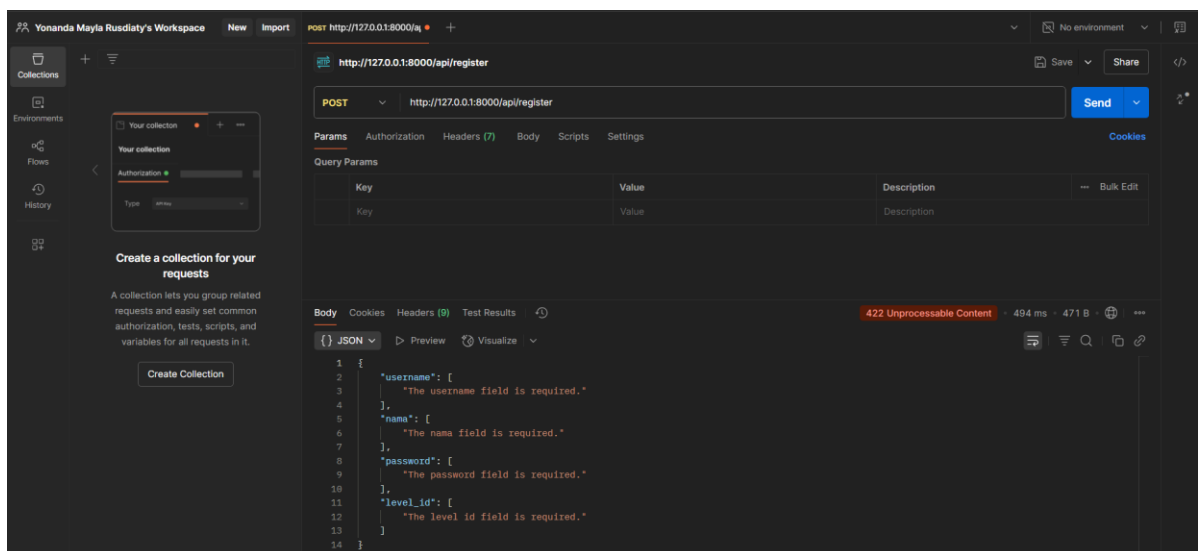


Buka aplikasi Postman, isi URL localhost/PWL_POS/public/api/register serta method POST. Klik Send.

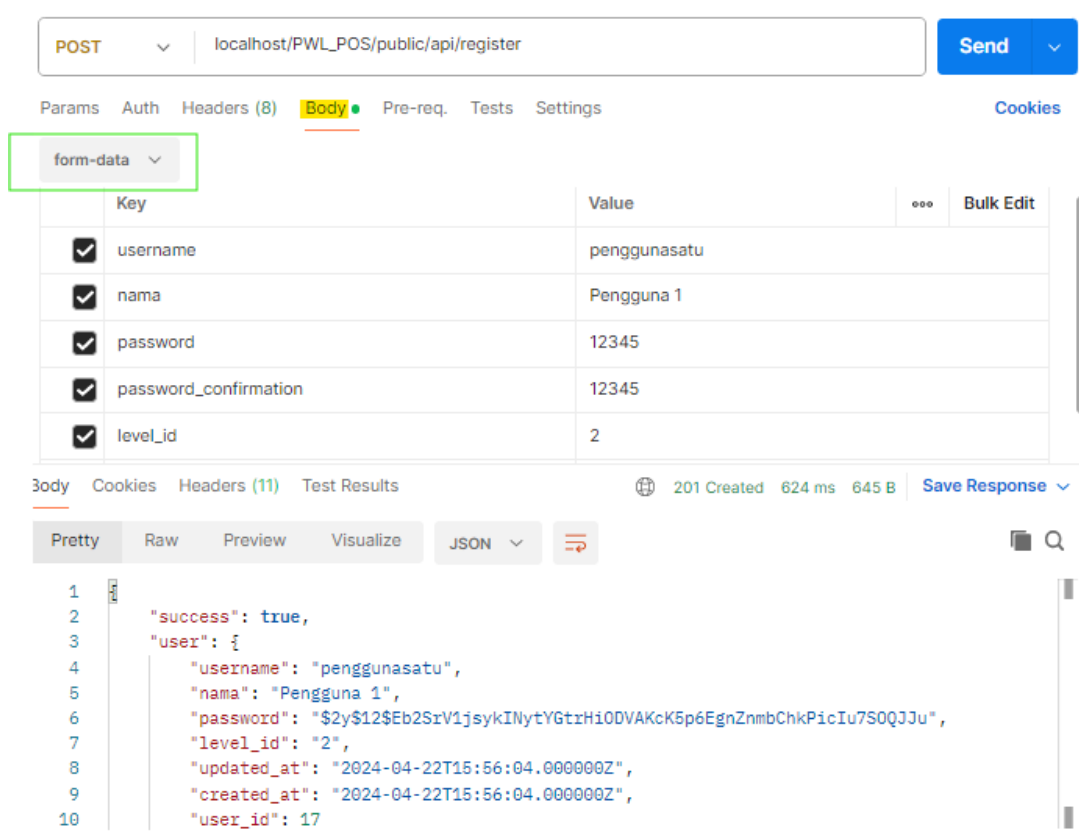


Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

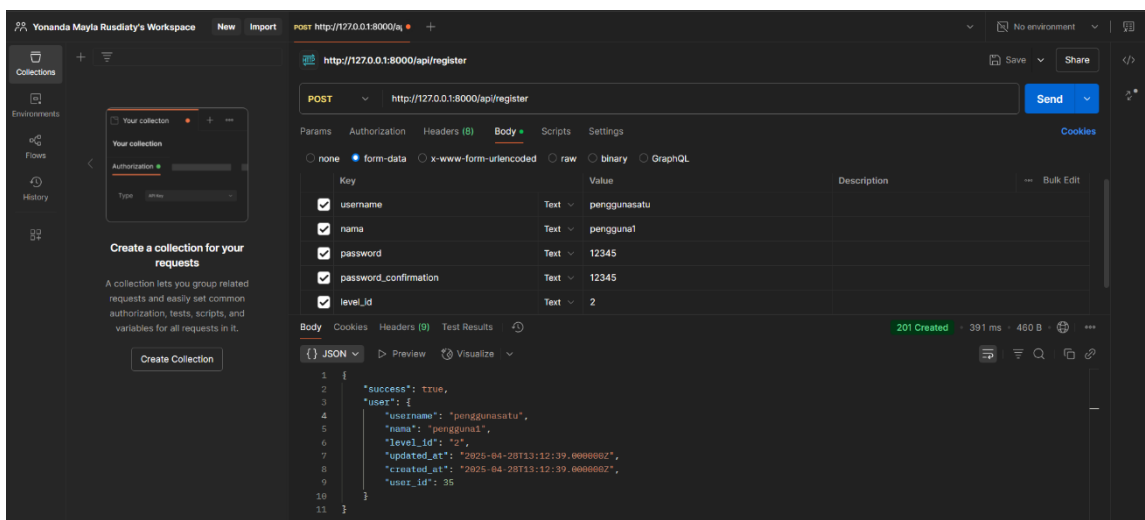


12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.



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.



Jawab: Jika kita mengirim data (seperti username, password) ke alamat /api/register dengan metode POST, maka aplikasi akan memproses pendaftaran pengguna baru.

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.

```
PS D:\Apps\laragon\www\PWL2025\week-10\jobsheet> php artisan make:controller Api/LoginController  
INFO Controller [D:\Apps\laragon\www\PWL2025\week-10\jobsheet\app\Http\Controllers\Api>LoginController.php] created successfully.
```

2. Buka file tersebut, dan ubah kode menjadi seperti berikut.

```
1  <?php  
2  
3  namespace App\Http\Controllers\Api;  
4  
5  use App\Http\Controllers\Controller;  
6  use Illuminate\Http\Request;  
7  use Illuminate\Support\Facades\Validator;  
8
```

```
<?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  
{
```



```
9  class LoginController extends Controller
10 {
11     public function __invoke(Request $request)
12     {
13         //set validation
14         $validator = Validator::make($request->all(), [
15             'username' => 'required',
16             'password' => 'required'
17         ]);
18
19         //if validation fails
20         if ($validator->fails()) {
21             return response()->json($validator->errors(), 422);
22         }
23
24         //get credentials from request
25         $credentials = $request->only('username', 'password');
26
27         //if auth failed
28         if (!$token = auth()->guard('api')->attempt($credentials)) {
29             return response()->json([
30                 'success' => false,
31                 'message' => 'Username atau Password Anda salah'
32             ], 401);
33         }
34
35         //if auth success
36         return response()->json([
37             'success' => true,
38             'user' => auth()->guard('api')->user(),
39             'token' => $token
40         ], 200);
41     }
42 }
```



```
0 references | 0 implementations
class LoginController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed
    {
        //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-only username, password
        $credentials = $request->only('username', 'password');

        //if auth failed
        if (!$token = auth()->guard('api')->attempt(credentials: $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);
    }
}
```

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

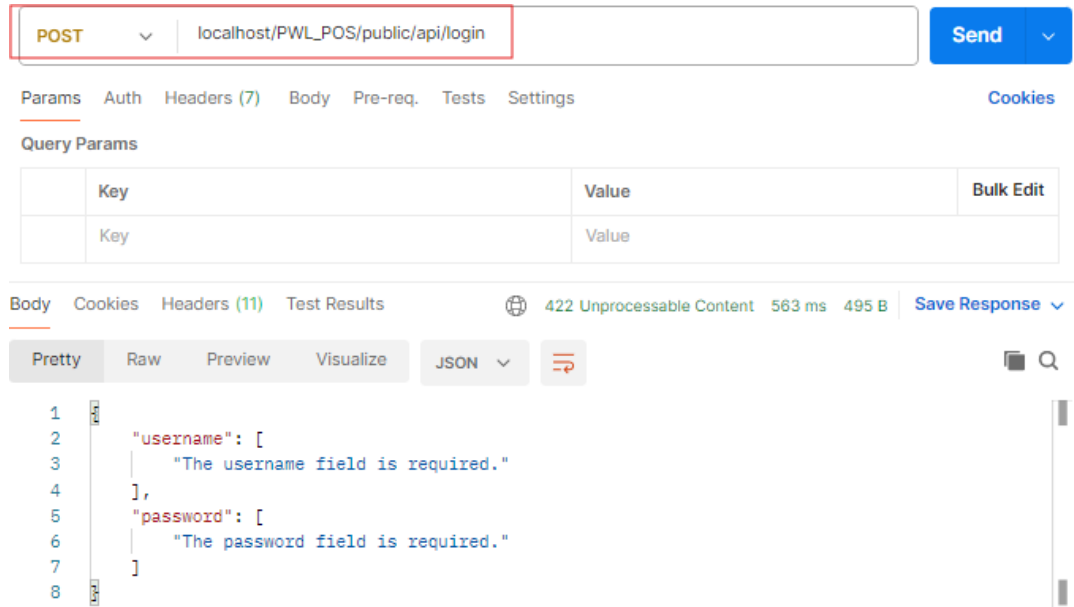
```
use App\Http\Controllers\Api\LoginController;

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();
});
```

```
routes > api.php > ...
You, 27 seconds ago | 1 author (You)
<?php
1
2
3 use Illuminate\Http\Request;
4 use Illuminate\Support\Facades\Route;
5
6 /*
7 |-----
8 | API Routes
9 |-----
10 |
11 | Here is where you can register API routes for your application. These
12 | routes are loaded by the RouteServiceProvider and all of them will
13 | be assigned to the "api" middleware group. Make something great!
14 |
15 */
16
17 Route::post('/register', App\Http\Controllers\Api\RegisterController::class)->name('register');
18 Route::post('/login', App\Http\Controllers\Api\LoginController::class)->name('login');
19 Route::middleware('auth:api')->get('/user', function (Request $request): mixed {
20     return $request->user();
21 });
You, 1 second ago * Uncommitted changes
```

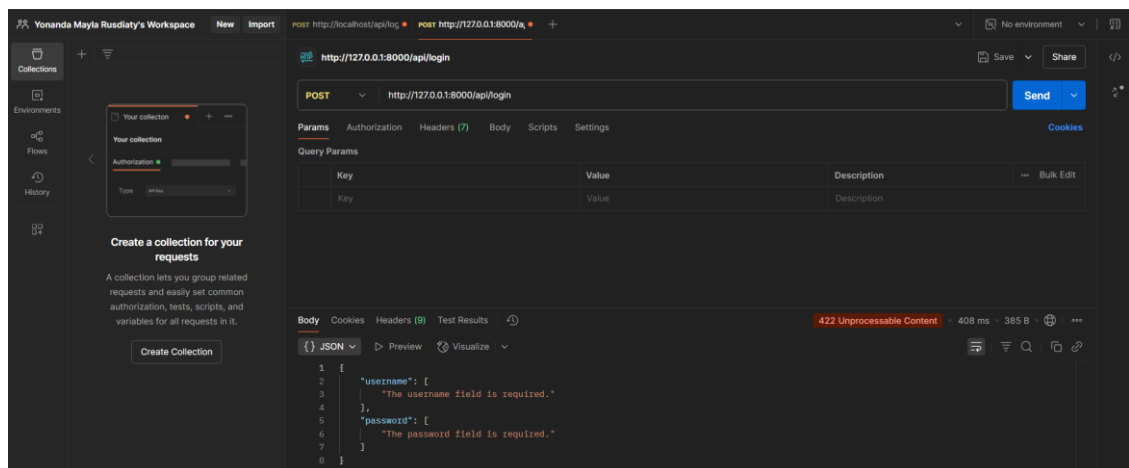


4. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL `localhost/PWL_POS/public/api/login` serta method POST. Klik Send.



Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshot 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

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none **form-data** x-www-form-urlencoded raw binary

Key	Value
username	penggunasatu
password	12345

Body Cookies Headers (11) Test Results Status: 200 OK Time: 1501 ms Size: 986 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "user": {
4     "user_id": 17,
5     "level_id": 2,
6     "username": "penggunasatu",
7     "nama": "Pengguna 1",
8     "password": "$2y$12$Eb2SxV1jsyKINyYgTzH10DVAKcK5p6EgnZnmbChkPicIu7S0QJJU",
9     "created_at": "2024-04-22T15:56:04.000000Z",
10    "updated_at": "2024-04-22T15:56:04.000000Z",
11  },
12  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ3c3MiOiJodHRwOi8vbG9jYXRob3N0L1BXTF9QT1MtbnFpb19wdWJsakMvYXpl2xvZ2luIiwiaWF0Ij0i"
13 }
```

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

POST http://127.0.0.1:8000/api/login

Params Authorization Headers (8) **Body** Scripts Settings Cookies

none **form-data** x-www-form-urlencoded raw binary GraphQL

Key	Value	Description
username	penggunasatu	
password	12345	

Body Cookies Headers (8) Test Results 200 OK · 397 ms · 801 B

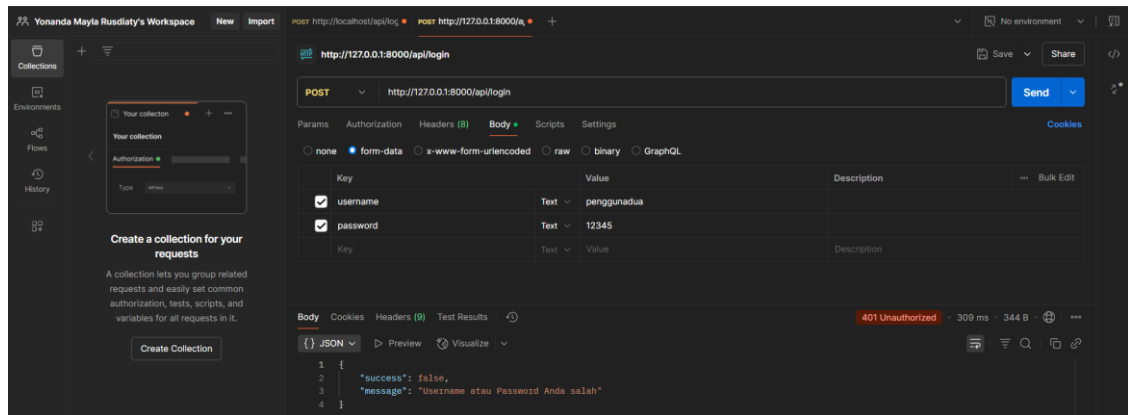
JSON Preview Visualize

```
1 {
2   "success": true,
3   "user": {
4     "user_id": 35,
5     "level_id": 2,
6     "foto_profile": null,
7     "username": "penggunasatu",
8     "nama": "pengguna1",
9     "created_at": "2025-04-28T13:12:39.000000Z",
10    "updated_at": "2025-04-28T13:12:39.000000Z",
11  },
12  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ3c3MiOiJodHRwOi8vbG9jYXRob3N0L1BXTF9QT1MtbnFpb19wdWJsakMvYXpl2xvZ2luIiwiaWF0Ij0i"
13 }
```

Jika kita mengirim username dan password ke alamat /api/login dengan metode POST, aplikasi akan memeriksa data tersebut dan memberikan token akses jika benar.

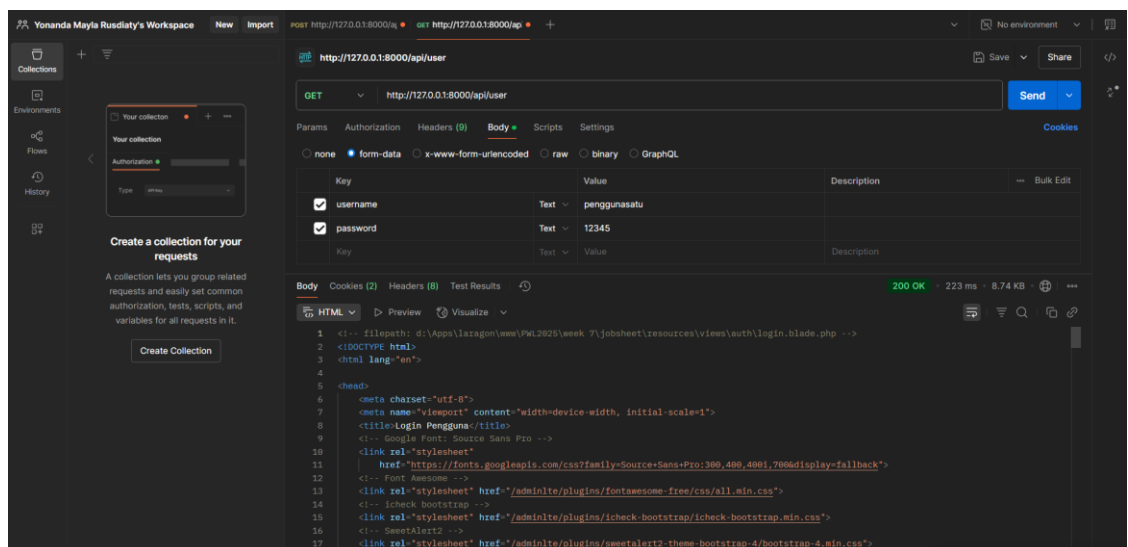


6. Lakukan percobaan yang untuk data yang salah dan berikan screenshoot hasil percobaan Anda.



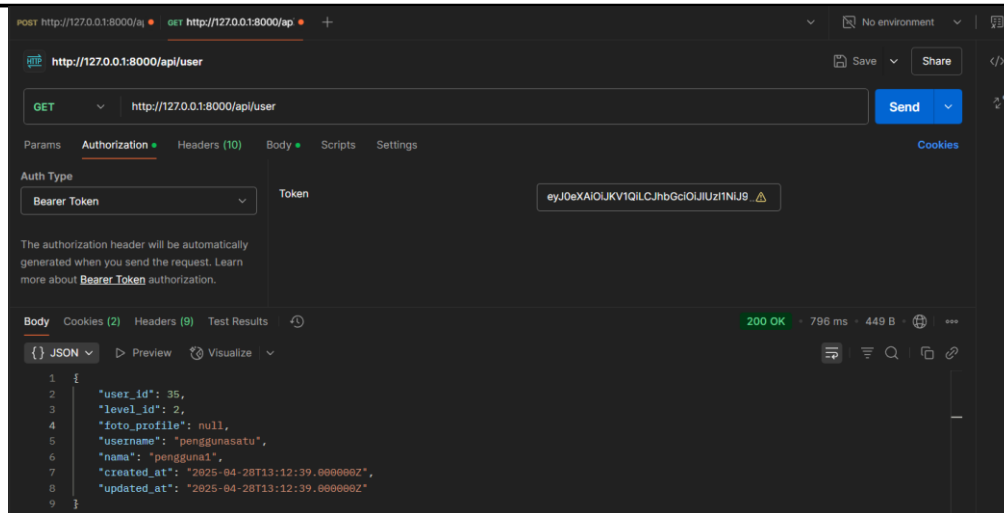
7. Coba kembali melakukan login dengan data yang benar. Sekarang mari kita coba menampilkan data user yang sedang login menggunakan URL `localhost/PWL_POS/public/api/user` dan method GET.

Jelaskan hasil dari percobaan tersebut.



Jawab:

Token menyimpan informasi identitas pengguna yang telah berhasil login. Saat kita mengakses method get pada url <http://127.0.0.1:8000/api/user>, hal ini berarti kita ingin menampilkan data diri sendiri di alamat /api/user tetapi harus sudah login dulu. Namun di return postmannya hanya bisa menampilkan halaman login karena token belum dimasukkan. Begini tampilan jika kita sudah memasukkan tokennya:



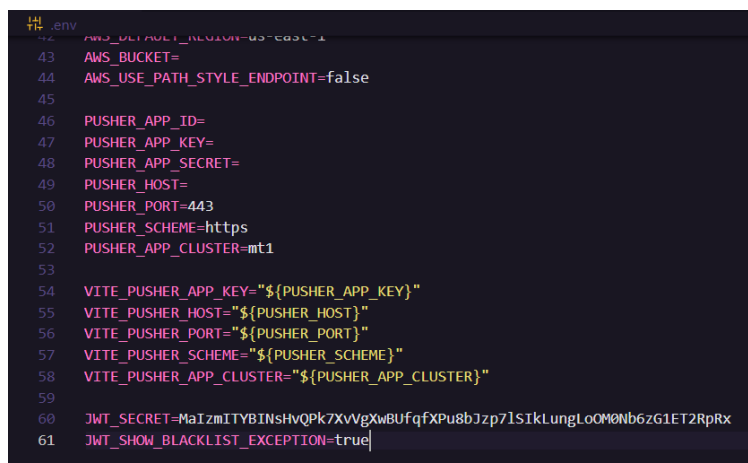
Setelah login, token disimpan dan ditambahkan ke header request berikutnya. Tanpa token, kita akan mendapatkan error 401 Unauthorized saat mencoba mengakses endpoint terproteksi

8. Lakukan commit perubahan file pada Github.

Praktikum 3 – Membuat RESTful API Logout

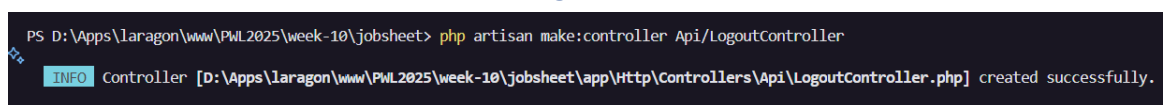
1. Tambahkan kode berikut pada file .env

`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.



```
1  <?php
2
3  namespace App\Http\Controllers\Api;
4  use Illuminate\Http\Request;
5  use App\Http\Controllers\Controller;
6  use Tymon\JWTAuth\Facades\JWTAuth;
7  use Tymon\JWTAuth\Exceptions\JWTException;
8  use Tymon\JWTAuth\Exceptions\TokenExpiredException;
9  use Tymon\JWTAuth\Exceptions\TokenInvalidException;
10
11 class LogoutController extends Controller
12 {
13     public function __invoke(Request $request)
14     {
15         //remove token
16         $removeToken = JWTAuth::invalidate(JWTAuth::getToken());
17
18         if($removeToken) {
19             //return response JSON
20             return response()->json([
21                 'success' => true,
22                 'message' => 'Logout Berhasil!',
23             ]);
24         }
25     }
26 }
```

app > Http > Controllers > Api > LogoutController.php > PHP Intelephense > LogoutController > __invoke

```
1  <?php
2
3  namespace App\Http\Controllers\API;
4
5  use Illuminate\Http\Request;
6  use App\Http\Controllers\Controller;
7  use Tymon\JWTAuth\Facades\JWTAuth;
8  use Tymon\JWTAuth\Exceptions\JWTException;
9  use Tymon\JWTAuth\Exceptions\TokenExpiredException;
10 use Tymon\JWTAuth\Exceptions\TokenInvalidException;
11
12 0 references | 0 implementations
13 class LogoutController extends Controller
14 {
15     0 references | 0 overrides
16     public function __invoke(Request $request): JsonResponse|mixed
17     {
18         //remove token
19         $removeToken = JWTAuth::invalidate(JWTAuth::getToken());
20
21         if($removeToken) {
22             //return response JSON
23             return response()->json([
24                 'success' => true,
25                 'message' => 'Logout Berhasil!',
26             ]);
27         }
28     }
29 }
```

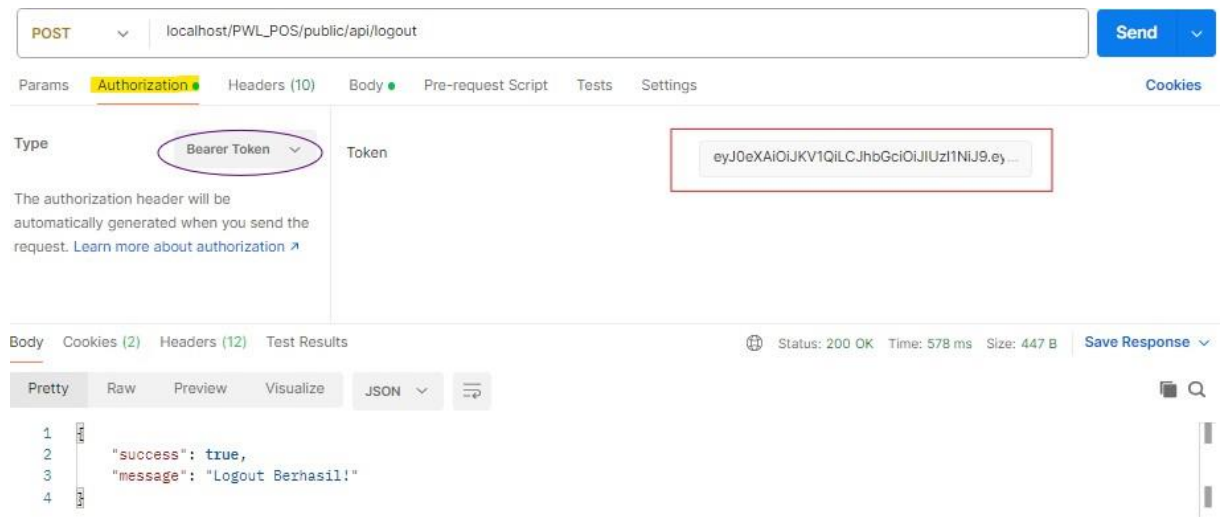


4. Lalu kita tambahkan routes pada api.php

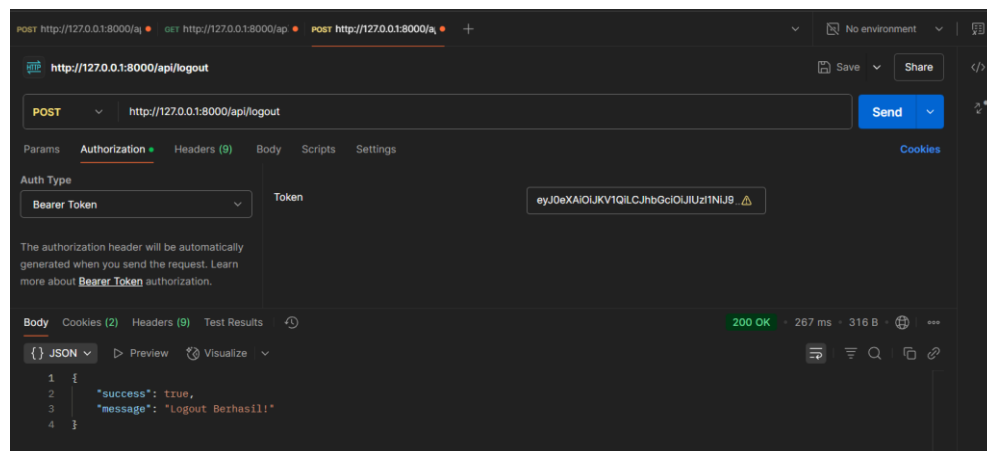
```
Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');

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): mixed {
    return $request->user();
});
Route::post('/logout', \App\Http\Controllers\Api\LogoutController::class)->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.



Jawab: Jika kita ingin keluar dari sistem, kirim permintaan ke alamat /api/logout dengan metode POST. Ini akan membuat token kita sebelumnya tidak bisa digunakan lagi. JWT_SHOW_BLACKLIST_EXCEPTION=true, mencegah penggunaan token yang sudah tidak valid



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
```

```
PS D:\Apps\laragon\www\PWL2025\week-10\jobsheet> php artisan make:controller Api/LevelController  
INFO Controller [D:\Apps\laragon\www\PWL2025\week-10\jobsheet\app\Http\Controllers\Api\LevelController.php] created successfully.
```

2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya.

```
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();  
    }  
}
```

```
<?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  
{  
    0 references | 0 overrides  
    public function index(): Collection  
    {  
        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',
    ]);
}
```

```
0 references | 0 implementations
9  class LevelController extends Controller
10
11  0 references | 0 overrides
12  public function index(): Collection
13  {
14      return LevelModel::all();
15  }
16
17  0 references | 0 overrides
18  public function store(Request $request): JsonResponse|mixed
19  {
20      $level = LevelModel::create($request->all());
21      return response()->json($level, 201);
22  }
23
24  0 references | 0 overrides
25  public function show(LevelModel $level): array|Collection|LevelModel|Model|null
26  {
27      return LevelModel::find($level);
28  }
29
30  0 references | 0 overrides
31  public function update(Request $request, LevelModel $level): array|Collection|LevelModel|Model|null
32  {
33      $level->update($request->all());
34      return LevelModel::find($level);
35  }
36
37  0 references | 0 overrides
38  public function destroy(LevelModel $user): JsonResponse|mixed
39  {
40      $user->delete();
41      return response()->json([
42          'success' => true,
43          'message' => 'Data terhapus',
44      ]);
45  }
46  }
```

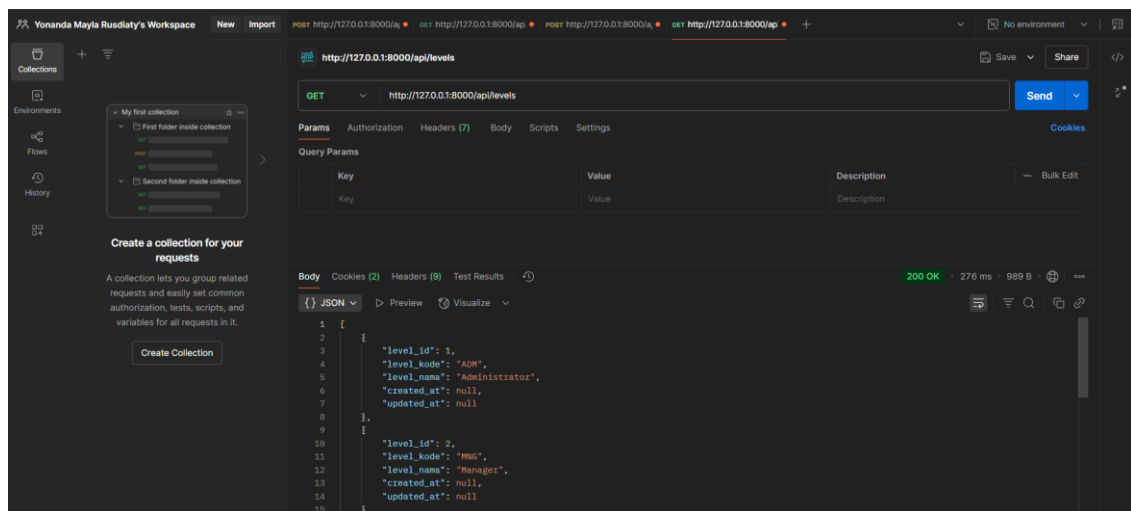


3. Kemudian kita lengkapi routes pada api.php.

```
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.



The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:8000/api/levels`. The response is a 200 OK status with a JSON body containing an array of level data.

	level_id	level_kode	level_nama	created_at	updated_at
0	1	ADM	Administrator	null	null
1	2	MNG	Manager	null	null
2	3	STF	Staff/Kasir	null	null
3	4	CUS	Customer	null	null
4	50	GUE	Guest	2025-03-29 13:38:37	null
5	51	SUP	Supervisor	2025-03-29 13:38:37	null
6	52	TRN	Trainee	2025-03-29 13:38:37	null

Jawab: Hasil yang didapatkan dari Postman saat mengakses endpoint GET /api/levels adalah data dalam format JSON yang menampilkan semua level pengguna dalam sistem

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



POST localhost/PWL_POS/public/api/levels

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> level_kode	Text SPV			
<input checked="" type="checkbox"/> level_nama	Text Supervisor			
Key	Text Value	Description		

Body Cookies Headers (11) Test Results

Status: 201 Created Time: 276 ms Size: 531 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "level_kode": "SPV",
3   "level_nama": "Supervisor",
4   "updated_at": "2024-04-22T21:40:32.000000Z",
5   "created_at": "2024-04-22T21:40:32.000000Z",
6   "level_id": 4
7 }
```

Jelaskan dan berikan screenshoot hasil percobaan Anda.

Yonanda Mayla Rusdlaty's Workspace

http://127.0.0.1:8000/api/levels

POST http://127.0.0.1:8000/api/levels

Params Authorization Headers (9) **Body** Scripts Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> level_kode	Text SPV			
<input checked="" type="checkbox"/> level_nama	Text Supervisor			
Key	Text Value	Description		

Body Cookies (2) Headers (9) Test Results

201 Created · 296 ms · 336 B

JSON Preview Visualize

```
1 {
2   "level_kode": "SPV",
3   "level_nama": "Supervisor",
4   "level_id": 55
5 }
```

Jawab: Disini kita sedang mengirimkan permintaan (request) ke API untuk membuat data baru di endpoint /api/levels, yaitu dengan kode SPV as a Supervisor

6. Berikutnya lakukan percobaan menampilkan detail data.

Jelaskan dan berikan screenshoot hasil percobaan Anda.

Yonanda Mayla Rusdlaty's Workspace

http://127.0.0.1:8000/api/levels/55

GET http://127.0.0.1:8000/api/levels/55

Params Authorization Headers (9) **Body** Scripts Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> level_kode	Text SPV			
<input checked="" type="checkbox"/> level_nama	Text Supervisor			
Key	Text Value	Description		

Body Cookies (2) Headers (9) Test Results

200 OK · 562 ms · 369 B

JSON Preview Visualize

```
1 {
2   "level_id": 55,
3   "level_kode": "SPV",
4   "level_nama": "Supervisor",
5   "created_at": null,
6   "updated_at": null
7 }
```

Disini kita sedang menampilkan detail data dengan endpoint /api/levels/ {level}.

Karena saya ingin menampilkan detail data dari supervisor, maka endpointnya adalah



<http://127.0.0.1:8000/api/levels/55>. Dan hasilnya seperti pada gambar.

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.

PUT localhost/PWL_POS-main/public/api/levels/4?level_kode=SPR

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

Query Params

Key	Value	Description	Bulk Edit
level_kode	SPR		
Key	Value	Description	

body Cookies Headers (11) Test Results Status: 200 OK Time: 266 ms Size: 528 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "level_id": 4,
4     "level_kode": "SPR",
5     "level_nama": "Supervisor",
6     "created_at": "2024-04-22T21:40:32.000000Z",
7     "updated_at": "2024-04-22T21:48:19.000000Z"
8   }
9 ]
```

Jelaskan dan berikan screenshoot hasil percobaan Anda.

PUT http://127.0.0.1:8000/api/levels/55?level_kode=SPR

Params Authorization Headers (9) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
level_kode	SPR		
Key	Value	Description	

Body Cookies (2) Headers (9) Test Results 200 OK 568 ms 369 B

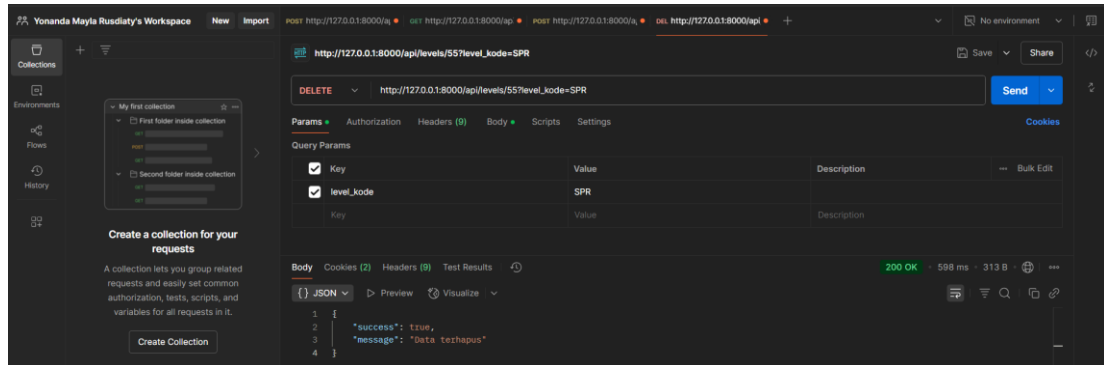
JSON

```
1 [
2   {
3     "level_id": 55,
4     "level_kode": "SPR",
5     "level_nama": "Supervisor",
6     "created_at": null,
7     "updated_at": null
8   }
9 ]
```

Data di Query Params bersifat terlihat di URL dan terbatas panjangnya. Params biasanya digunakan untuk GET seperti filter, dan searching.



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



Disini saya menghapus level/55 menggunakan endpoint http://127.0.0.1:8000/api/levels/55?level_kode=SPR dengan method DELETE dan menggunakan query params karena yg dibutuhkan hanya id saja

9. Lakukan commit perubahan file pada Github.

TUGAS

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

JAWAB

➤ Tabel : m_user

1. Pertama, buat controller untuk mengolah API pada data user

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan make:controller Api/UserController

INFO Controller [D:\Apps\laragon\www\PWL2025\week 10\jobsheet\app\Http\Controllers\Api\UserController.php] created successfully.
```

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

6 references | 0 implementations
class UserController extends Controller
{
    /**
     * Create a new AuthController instance.
     *
     * @return void
     */
    0 references | 0 overrides
    public function __construct()
    {
        $this->middleware('auth:api', ['except' => []]);
    }

    /**
     * Display a listing of users.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    1 reference | 0 overrides
    public function index(): JsonResponse|mixed
    {
        $users = UserModel::with('level')->get();
    }
}
```

```
return response()->json([
    'success' => true,
    'message' => 'Daftar data user',
    'data' => $users
]);

/**
 * Store a newly created user.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\JsonResponse
 */
1 reference | 0 overrides
public function store(Request $request): JsonResponse|mixed
{
    $validator = Validator::make($request->all(), [
        'username' => 'required|string|min:3|unique:m_user,username',
        'nama' => 'required|string|max:100',
        'password' => 'required|min:6',
        'level_id' => 'required|integer|exists:m_level,level_id',
    ]);

    if ($validator->fails()) {
        return response()->json([
            'success' => false,
            'message' => 'Validasi gagal',
            'errors' => $validator->errors()
        ], 422);
    }

    $user = UserModel::create([
        'username' => $request->username,
        'nama' => $request->nama,
        'password' => Hash::make($request->password),
        'level_id' => $request->level_id
    ]);

    return response()->json([

```



```
        return response()->json([
            'success' => true,
            'message' => 'User berhasil ditambahkan',
            'data' => $user
        ], 201);
    }

    /**
     * Display the specified user.
     *
     * @param int $id
     * @return \Illuminate\Http\JsonResponse
     */
    // 1 reference | 0 overrides
    public function show($id): JsonResponse|mixed
    {
        $user = UserModel::with('level')->find($id);

        if (!$user) {
            return response()->json([
                'success' => false,
                'message' => 'User tidak ditemukan'
            ], 404);
        }

        return response()->json([
            'success' => true,
            'message' => 'Detail user',
            'data' => $user
        ]);
    }

    /**
     * Update the specified user.
     *
     * @param \Illuminate\Http\Request $request
     * @param int $id
     * @return \Illuminate\Http\JsonResponse
     */
```

```
class UserController extends Controller
{
    // 1 reference | 0 overrides
    public function update(Request $request, $id): JsonResponse|mixed
    {
        $user = UserModel::find($id);

        if (!$user) {
            return response()->json([
                'success' => false,
                'message' => 'User tidak ditemukan'
            ], 404);
        }

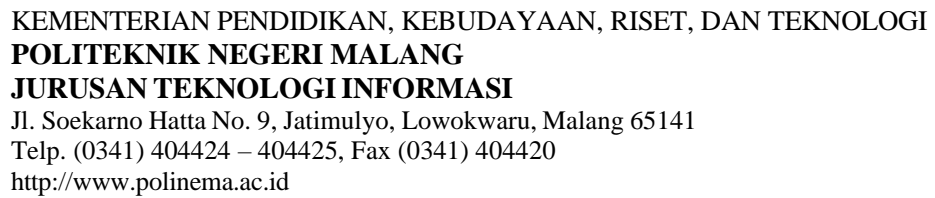
        $validator = Validator::make($request->all(), [
            'username' => 'required|string|min:3|unique:m_user,username,' . $id . ',user_id',
            'nama' => 'required|string|max:100',
            'password' => 'nullable|min:6',
            'level_id' => 'required|integer|exists:m_level,level_id',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validasi gagal',
                'errors' => $validator->errors()
            ], 422);
        }

        $userData = [
            'username' => $request->username,
            'nama' => $request->nama,
            'level_id' => $request->level_id
        ];

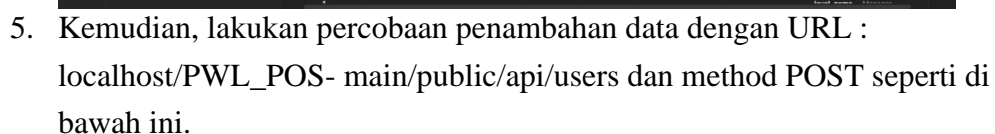
        // Hanya update password jika diisi
        if ($request->filled('password')) {
            $userData['password'] = Hash::make($request->password);
        }

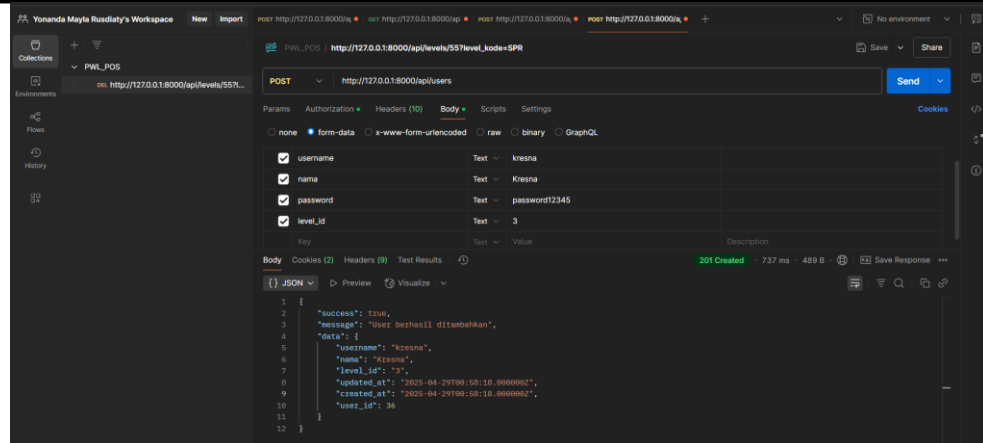
        $user->update($userData);
    }
}
```



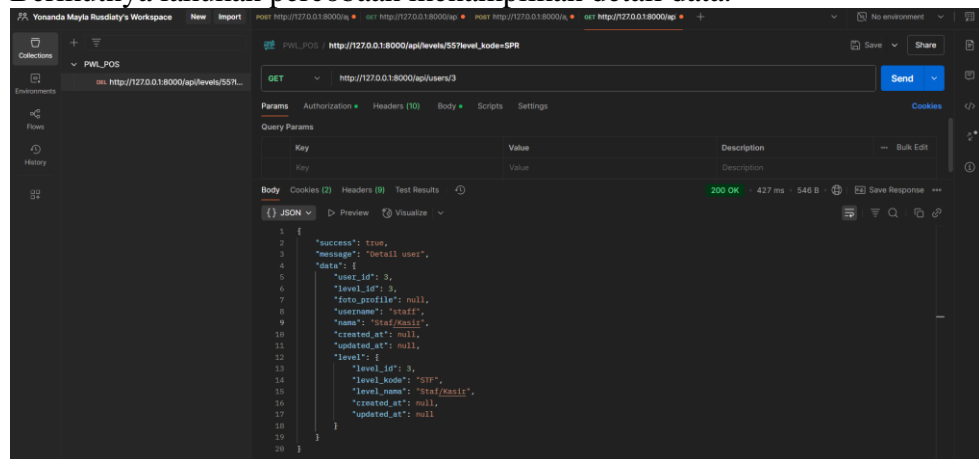
3. Kemudian kita lengkapi routes pada api.php.

4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/users dan method GET.

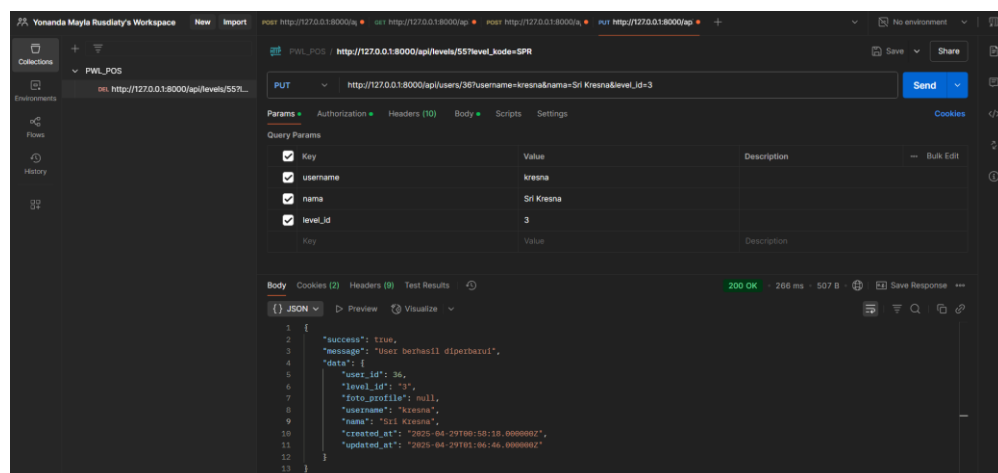




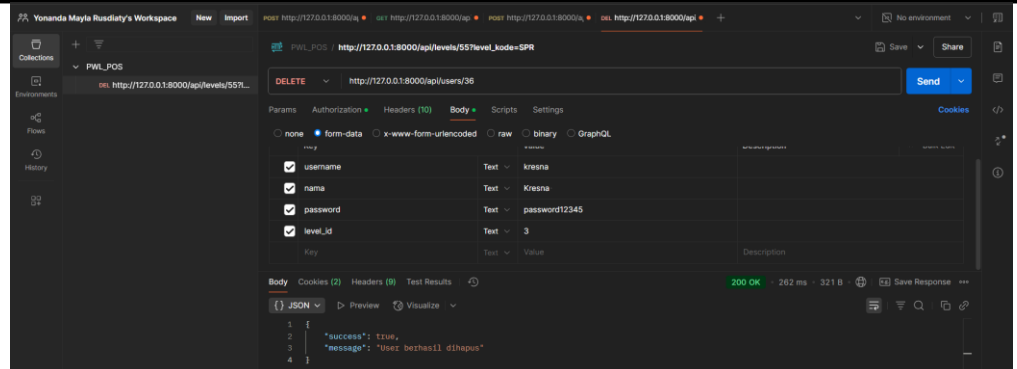
6. Berikutnya lakukan percobaan menampilkan detail data.



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.



8. Terakhir lakukan percobaan hapus data



➤ **Tabel: m_kategori**

1. Pertama, buat controller untuk mengolah API pada data kategori

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan make:controller Api/KategoriController
INFO Controller [D:\Apps\laragon\www\PWL2025\week 10\jobsheet\app\Http\Controllers\Api\KategoriController.php] created successfully.
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet>
```

2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya



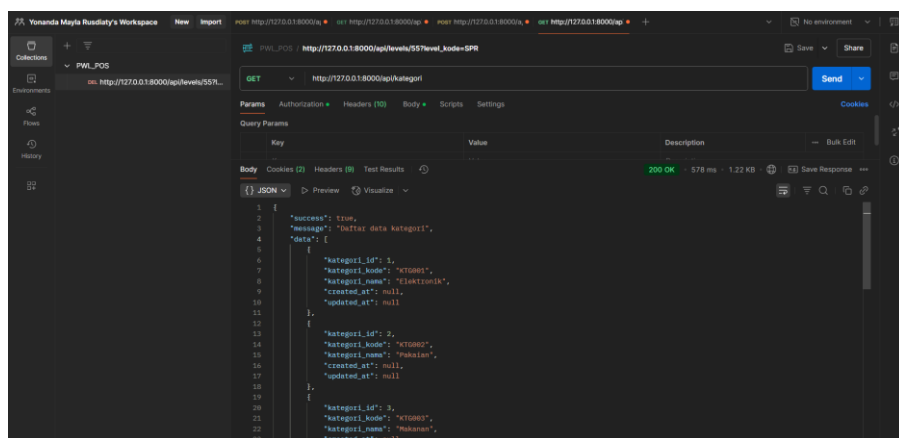
```
1 <http
2
3 namespace app\Http\Controllers;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\Validator;
8
9 class KategoriController extends Controller
10 {
11     /**
12      * Create a new controller instance.
13      *
14      * @return void
15      */
16     public function __construct()
17     {
18         $this->middleware('auth:api');
19     }
20
21     /**
22      * Display a listing of the resource.
23      *
24      * @return \Illuminate\Http\Response
25      */
26     public function index()
27     {
28         $kategori = KategoriModel::all();
29
30         return response()->json([
31             'success' => true,
32             'message' => 'Data dari kategori',
33             'data' => $kategori
34         ]);
35     }
36
37     /**
38      * Store a newly created resource in storage.
39      *
40      * @param \Illuminate\Http\Request $request
41      * @return \Illuminate\Http\Response
42      */
43     public function store(Request $request)
44     {
45         $validator = Validator::make($request->all(), [
46             'kategori_kode' => 'required|string|max:10|unique:kategori,kategori_kode',
47             'kategori_nama' => 'required|string|max:100',
48         ]);
49
50         if ($validator->fails()) {
51             return response()->json([
52                 'success' => false,
53                 'message' => 'Validasi gagal',
54                 'errors' => $validator->errors()
55             ], 422);
56         }
57
58         $kategori = KategoriModel::create([
59             'kategori_kode' => $request->kategori_kode,
60             'kategori_nama' => $request->kategori_nama
61         ]);
62
63         return response()->json([
64             'success' => true,
65             'message' => 'Kategori berhasil ditambahkan',
66             'data' => $kategori
67         ], 201);
68     }
69
70     /**
71      * Display the specified resource.
72      *
73      * @param int $id
74      * @return \Illuminate\Http\Response
75      */
76     public function show($id)
77     {
78         $kategori = KategoriModel::find($id);
79
80         if (!$kategori) {
81             return response()->json([
82                 'success' => false,
83                 'message' => 'Kategori tidak ditemukan'
84             ], 404);
85         }
86
87         return response()->json([
88             'success' => true,
89             'message' => 'Detail kategori',
90             'data' => $kategori
91         ]);
92     }
93
94     /**
95      * Update the specified resource in storage.
96      *
97      * @param \Illuminate\Http\Request $request
98      * @param int $id
99      * @return \Illuminate\Http\Response
100      */
101     public function update(Request $request, $id)
102     {
103         $kategori = KategoriModel::find($id);
104
105         if (!$kategori) {
106             return response()->json([
107                 'success' => false,
108                 'message' => 'Kategori tidak ditemukan'
109             ], 404);
110         }
111
112         $validator = Validator::make($request->all(), [
113             'kategori_kode' => 'required|string|max:10|unique:kategori,kategori_kode',
114             'kategori_nama' => 'required|string|max:100',
115         ]);
116
117         if ($validator->fails()) {
118             return response()->json([
119                 'success' => false,
120                 'message' => 'Validasi gagal',
121                 'errors' => $validator->errors()
122             ], 422);
123         }
124
125         $kategori->update([
126             'kategori_kode' => $request->kategori_kode,
127             'kategori_nama' => $request->kategori_nama
128         ]);
129
130         return response()->json([
131             'success' => true,
132             'message' => 'Kategori berhasil diperbarui',
133             'data' => $kategori
134         ]);
135     }
136
137     /**
138      * Remove the specified resource from storage.
139      *
140      * @param int $id
141      * @return \Illuminate\Http\Response
142      */
143     public function destroy($id)
144     {
145         $kategori = KategoriModel::find($id);
146
147         if (!$kategori) {
148             return response()->json([
149                 'success' => false,
150                 'message' => 'Kategori tidak ditemukan'
151             ], 404);
152         }
153
154         try {
155             $kategori->delete();
156
157             return response()->json([
158                 'success' => true,
159                 'message' => 'Kategori berhasil dihapus'
160             ]);
161         } catch (Exception $e) {
162             return response()->json([
163                 'success' => false,
164                 'message' => 'Kategori tidak dapat dihapus karena masih digunakan',
165                 'error' => $e->getMessage()
166             ], 409);
167         }
168     }
169 }
```



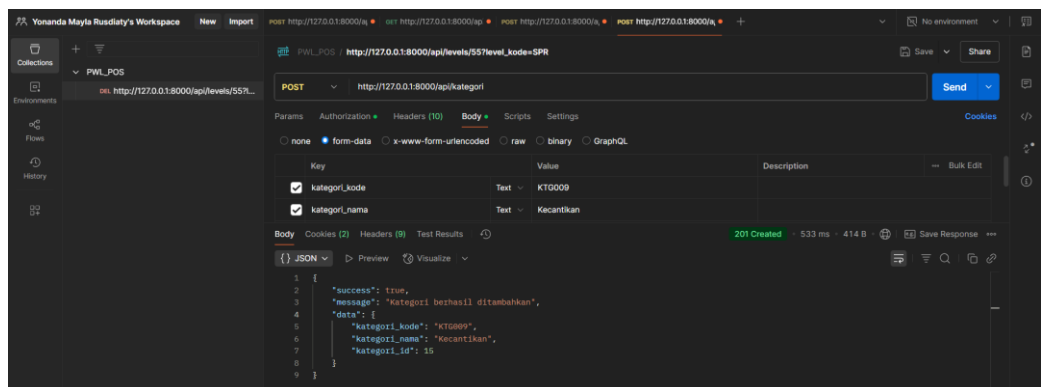

3. Kemudian kita lengkapi routes pada api.php.

```
// kategori
Route::middleware('auth:api')->group(function(): void {
    Route::get('/kategori', [KategoriController::class, 'index']);
    Route::post('/kategori', [KategoriController::class, 'store']);
    Route::get('/kategori/{id}', [KategoriController::class, 'show']);
    Route::put('/kategori/{id}', [KategoriController::class, 'update']);
    Route::delete('/kategori/{id}', [KategoriController::class, 'destroy']);
});
```

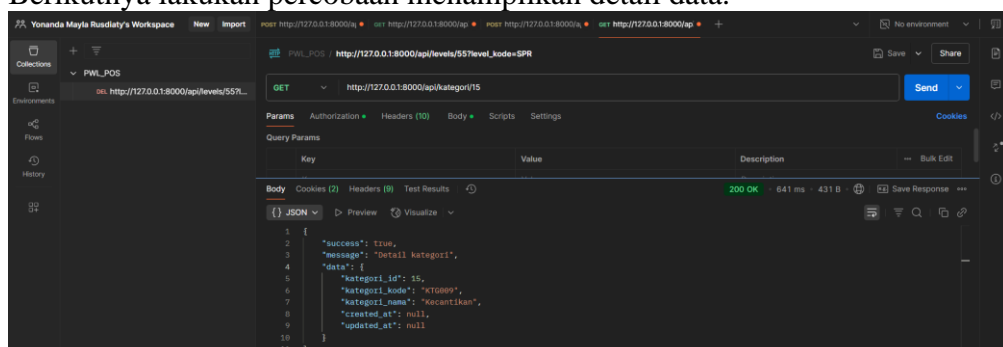
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/kategori dan method GET.



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

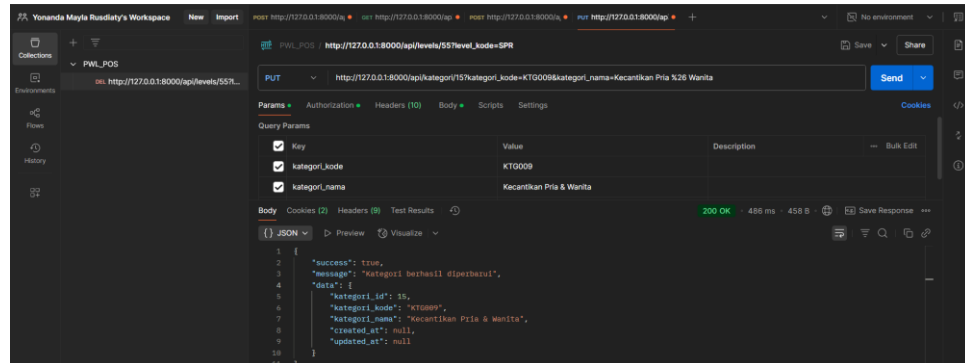


6. Berikutnya lakukan percobaan menampilkan detail data.

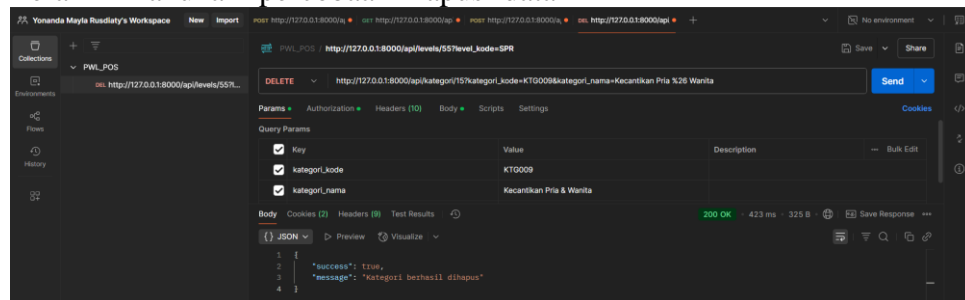




7. Jika sudah, kita coba untuk melakukan edit data menggunakan `localhost/PWL_POS-main/public/api/levels/{id}` dan method PUT. Isikan data yang ingin diubah pada tab Param.



8. Terakhir lakukan percobaan hapus data



➤ Tabel: m_barang

1. Pertama, buat controller untuk mengolah API pada data barang

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan make:controller Api\BarangController
```

INFO Controller [D:\Apps\laragon\www\PWL2025\week 10\jobsheet\app\Http\Controllers\Api\BarangController.php] created successfully.

2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya



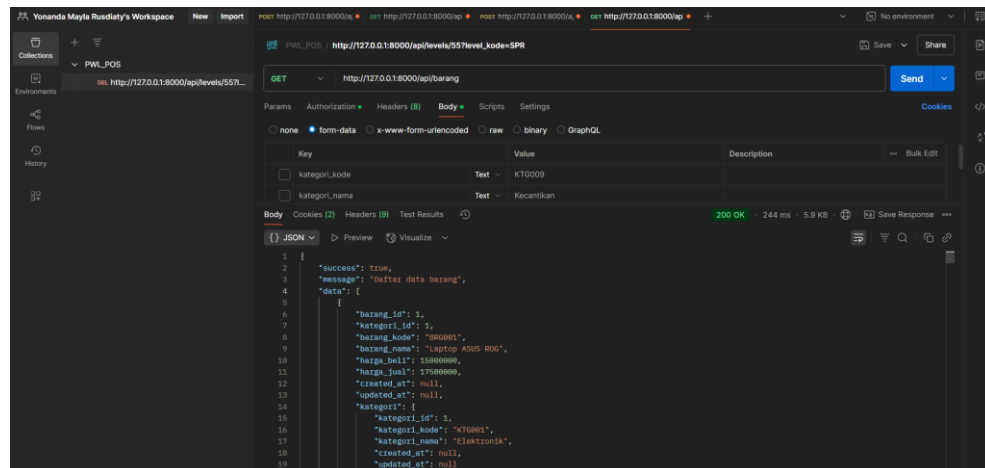
```
1 </do>
2
3 namespace App\Http\Controllers{
4
5     use App\Http\Controllers\Controller;
6     use Illuminate\Http\Request;
7     use Illuminate\Support\Facades\Validator;
8
9     class BarangController extends Controller
10     {
11         /**
12          * Create a new controller instance.
13          *
14          * @return void
15          */
16         public function __construct()
17         {
18             $this->middleware('auth:api');
19         }
20
21         /**
22          * Display a listing of the resource.
23          *
24          * @return \Illuminate\Http\Response
25          */
26         public function index()
27         {
28             $barang = Barang::select('kategori')->get();
29
30             return response()->json([
31                 'success' => true,
32                 'message' => 'Data data barang',
33                 'data' => $barang
34             ]);
35         }
36
37         /**
38          * Store a newly created resource in storage.
39          *
40          * @param \Illuminate\Http\Request $request
41          * @return \Illuminate\Http\Response
42          */
43         public function store(Request $request)
44         {
45             $validator = Validator::make($request->all(), [
46                 'barang_kode' => 'required|string|max:100|unique=barang,barang_kode',
47                 'barang_nama' => 'required|string|max:100',
48                 'harga' => 'required|numeric',
49                 'stok' => 'required|integer|min:0',
50                 'kategori_id' => 'required|exists=categories,kategori_id',
51             ]);
52
53             if ($validator->fails()) {
54                 return response()->json([
55                     'success' => false,
56                     'message' => 'Validasi gagal',
57                     'errors' => $validator->errors()
58                 ], 422);
59             }
60
61             $barang = Barang::create([
62                 'barang_kode' => $request->barang_kode,
63                 'barang_nama' => $request->barang_nama,
64                 'harga' => $request->harga,
65                 'stok' => $request->stok,
66                 'kategori_id' => $request->kategori_id,
67             ]);
68
69             return response()->json([
70                 'success' => true,
71                 'message' => 'Barang berhasil ditambahkan',
72                 'data' => $barang
73             ], 201);
74         }
75
76         /**
77          * Display the specified resource.
78          *
79          * @param int $id
80          * @return \Illuminate\Http\Response
81          */
82         public function show($id)
83         {
84             $barang = Barang::select('kategori')->find($id);
85
86             if (!$barang) {
87                 return response()->json([
88                     'success' => false,
89                     'message' => 'Barang tidak ditemukan'
90                 ], 404);
91             }
92
93             return response()->json([
94                 'success' => true,
95                 'message' => 'Detail barang',
96                 'data' => $barang
97             ]);
98         }
99
100         /**
101          * Update the specified resource in storage.
102          *
103          * @param \Illuminate\Http\Request $request
104          * @param int $id
105          * @return \Illuminate\Http\Response
106          */
107         public function update(Request $request, $id)
108         {
109             $barang = Barang::find($id);
110
111             if (!$barang) {
112                 return response()->json([
113                     'success' => false,
114                     'message' => 'Barang tidak ditemukan'
115                 ], 404);
116             }
117
118             $validator = Validator::make($request->all(), [
119                 'barang_kode' => 'required|string|max:100|unique=barang,barang_kode', 'id' => 'required',
120                 'barang_nama' => 'required|string|max:100',
121                 'harga' => 'required|numeric',
122                 'stok' => 'required|integer|min:0',
123                 'kategori_id' => 'required|exists=categories,kategori_id',
124             ]);
125
126             if ($validator->fails()) {
127                 return response()->json([
128                     'success' => false,
129                     'message' => 'Validasi gagal',
130                     'errors' => $validator->errors()
131                 ], 422);
132             }
133
134             $barang->update([
135                 'barang_kode' => $request->barang_kode,
136                 'barang_nama' => $request->barang_nama,
137                 'harga' => $request->harga,
138                 'stok' => $request->stok,
139                 'kategori_id' => $request->kategori_id,
140             ]);
141
142             return response()->json([
143                 'success' => true,
144                 'message' => 'Barang berhasil diperbarui',
145                 'data' => $barang
146             ], 200);
147         }
148
149         /**
150          * Remove the specified resource from storage.
151          *
152          * @param int $id
153          * @return \Illuminate\Http\Response
154          */
155         public function destroy($id)
156         {
157             $barang = Barang::find($id);
158
159             if (!$barang) {
160                 return response()->json([
161                     'success' => false,
162                     'message' => 'Barang tidak ditemukan'
163                 ], 404);
164             }
165
166             try {
167                 $barang->delete();
168             } catch (Exception $e) {
169                 return response()->json([
170                     'success' => false,
171                     'message' => 'Barang berhasil dihapus'
172                 ], 200);
173             }
174
175             return response()->json([
176                 'success' => true,
177                 'message' => 'Barang tidak dapat dihapus karena masih digunakan',
178                 'error' => $e->getMessage()
179             ], 400);
180         }
181     }
182 }
```



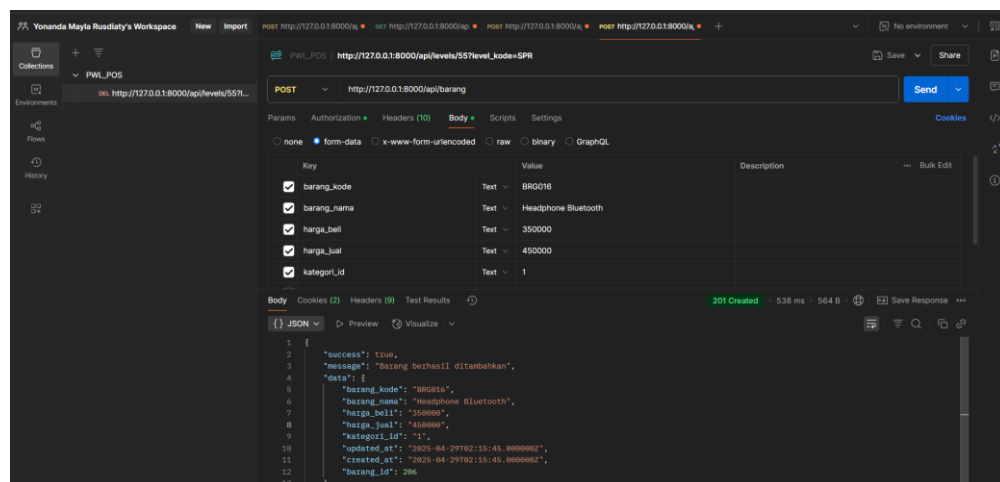
3. Kemudian kita lengkapi routes pada api.php.

```
Route::middleware('auth:api')->group(function(): void {  
    Route::get('/barang', [BarangController::class, 'index']);  
    Route::post('/barang', [BarangController::class, 'store']);  
    Route::get('/barang/{id}', [BarangController::class, 'show']);  
    Route::put('/barang/{id}', [BarangController::class, 'update']);  
    Route::delete('/barang/{id}', [BarangController::class, 'destroy']);  
});
```

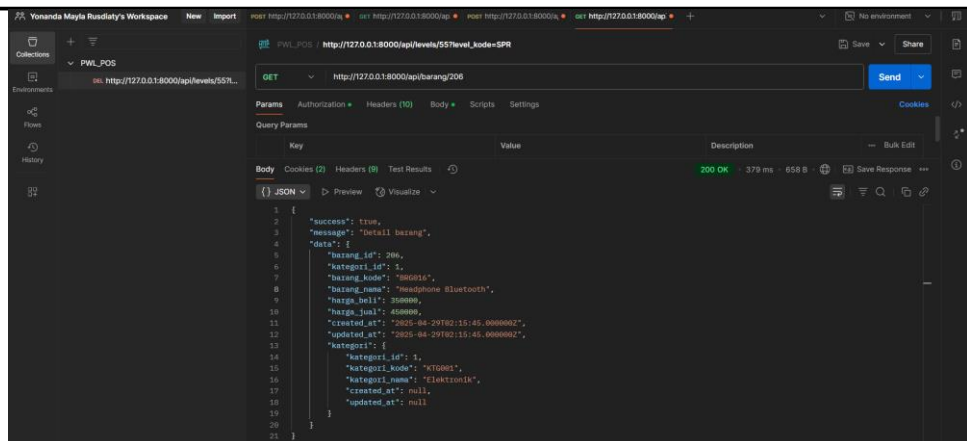
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data.
Gunakan URL: localhost/PWL_POS-main/public/api/barang dan method GET.



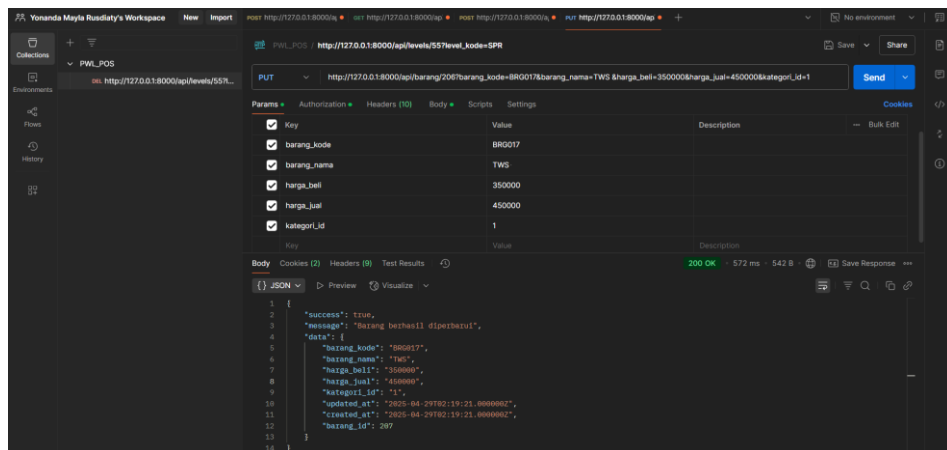
5. Kemudian, lakukan percobaan penambahan data dengan URL :
localhost/PWL_POS- main/public/api/barang dan method POST seperti di
bawah ini.



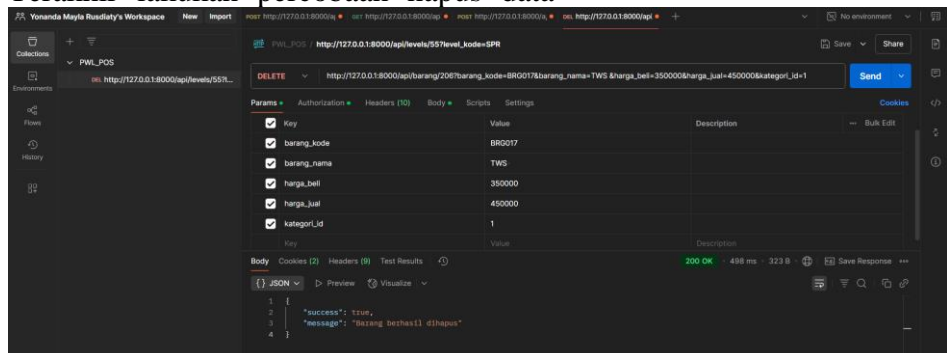
6. Berikutnya lakukan percobaan menampilkan detail data.



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



8. Terakhir lakukan percobaan hapus data



*** Sekian, dan selamat belajar ***