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

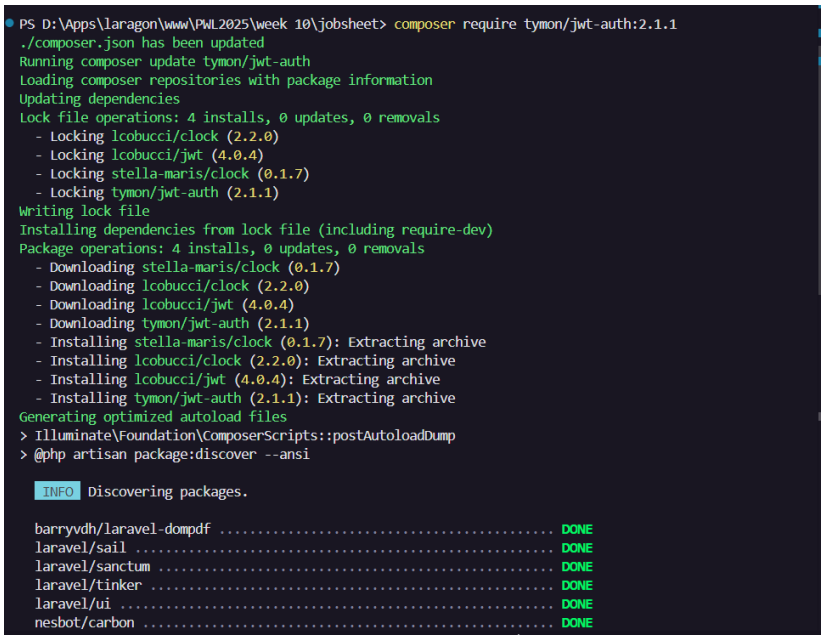
  JSON Web Token adalah cara yang baik untuk mengirimkan informasi antar pihak dengan aman. Dengan token yang sudah ditandatangani dengan algoritma RSA, maka kita bisa tahu siapa yang melakukan request tersebut.

  Berikut adalah cara kerja JWT :

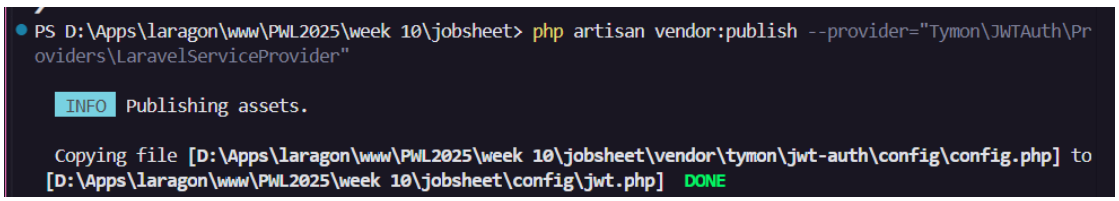
JWT (JSON Web Token) adalah cara untuk mentransfer informasi antara dua pihak secara aman sebagai objek JSON. Ini terdiri dari tiga bagian: header, payload, dan signature. Setelah pengguna berhasil autentikasi, server menghasilkan token JWT yang disematkan dalam permintaan HTTP. Server kemudian memvalidasi 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.

```
PS D:\Apps\laragon\www\PWL2025\week 10\jobsheet> php artisan jwt:secret
jwt-auth secret [MaIzmITYBINsHvQPk7XvVgXwBUfqfXPu8bJzp7lSIkLungLoOM0Nb6zG1ET2RpRx] 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
<?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
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 secon
    {
        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.

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

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

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)
    {
```

```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
        $validator = Validator::make($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);
    }
}
```

```
        $validator = Validator::make($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.

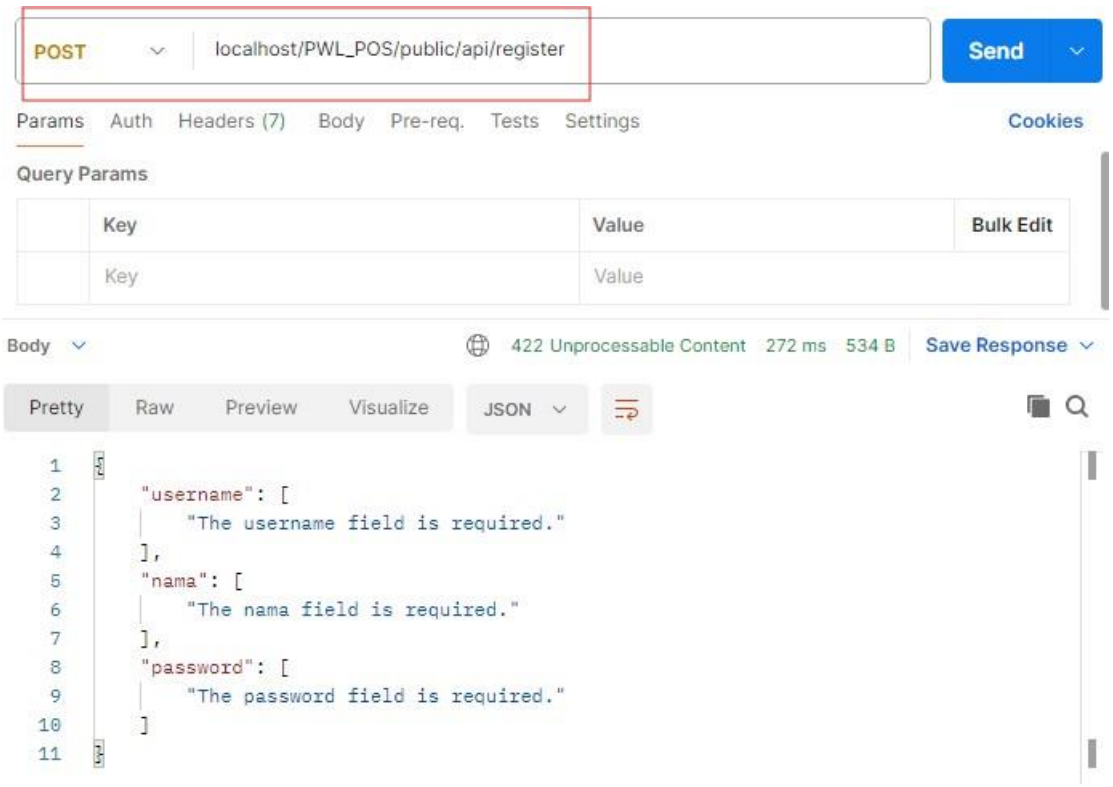
```php
<?php

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

```php
<?php

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');
```

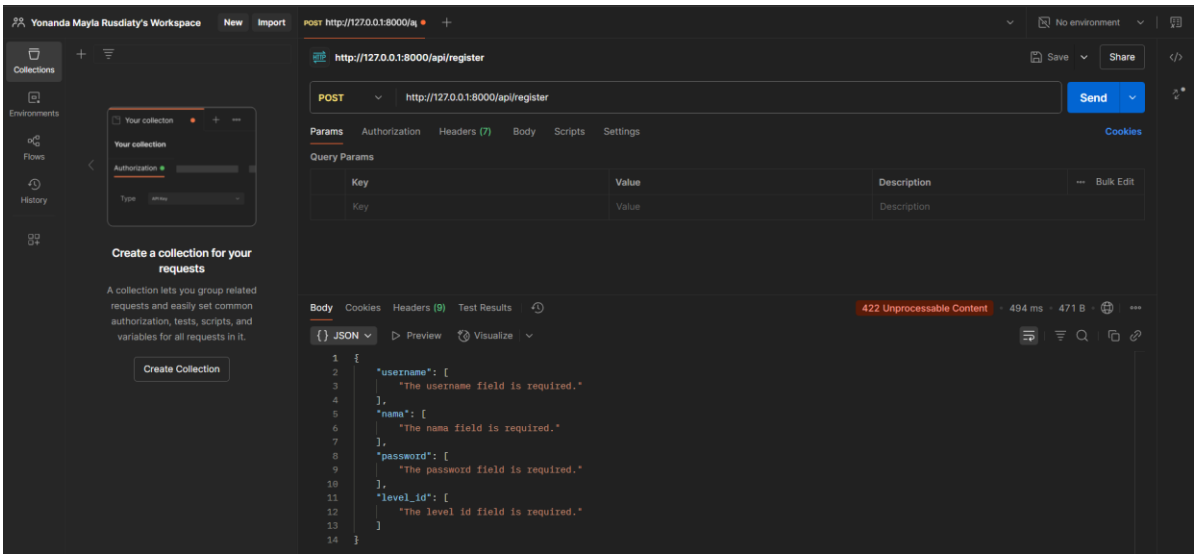
11. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman.

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

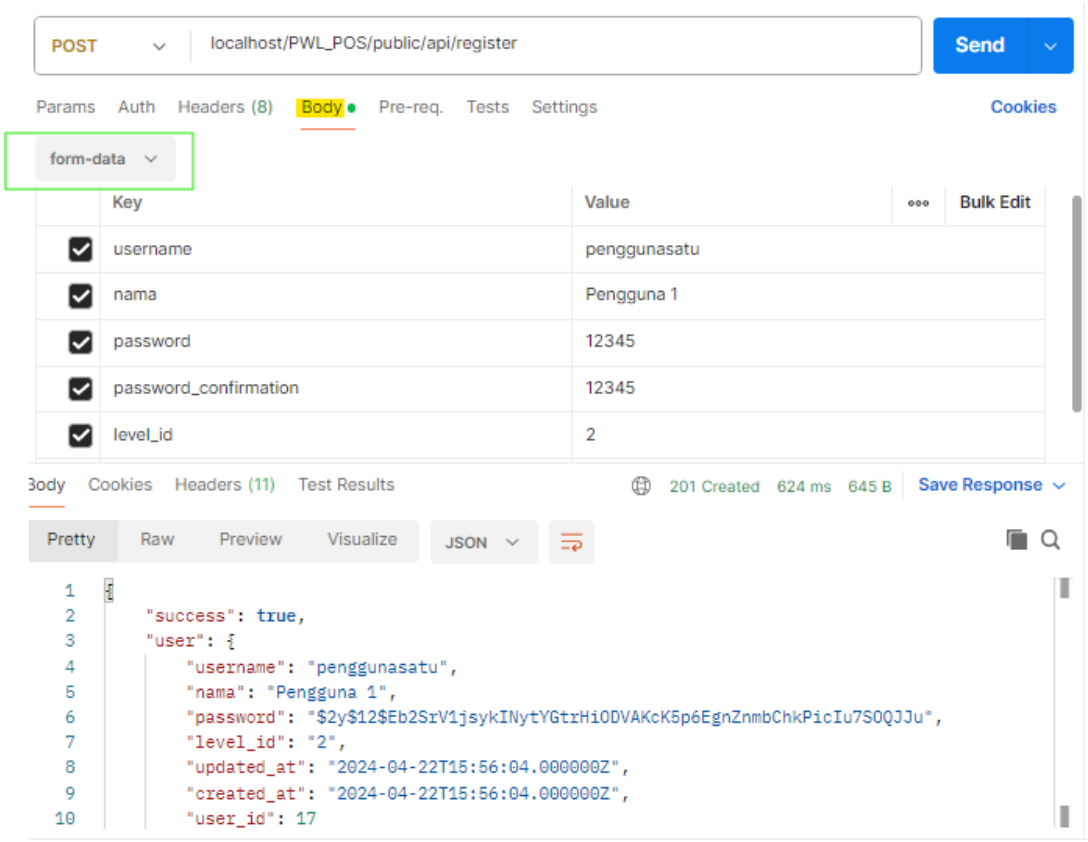


Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

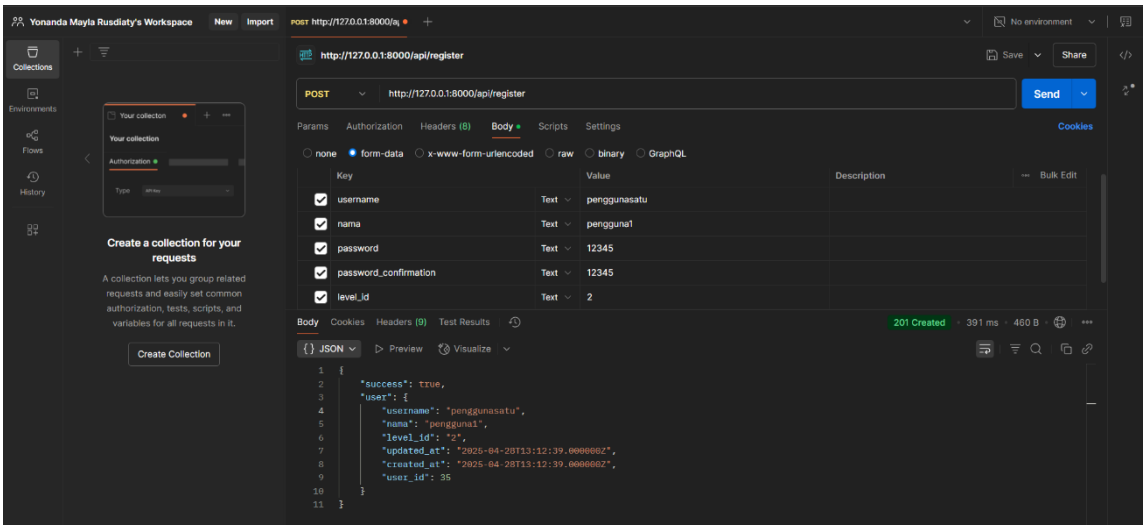


12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.



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.

   ```php
   <?php

   namespace App\Http\Controllers\Api;

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

   ```

   ```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
   {
   ```

```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
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.

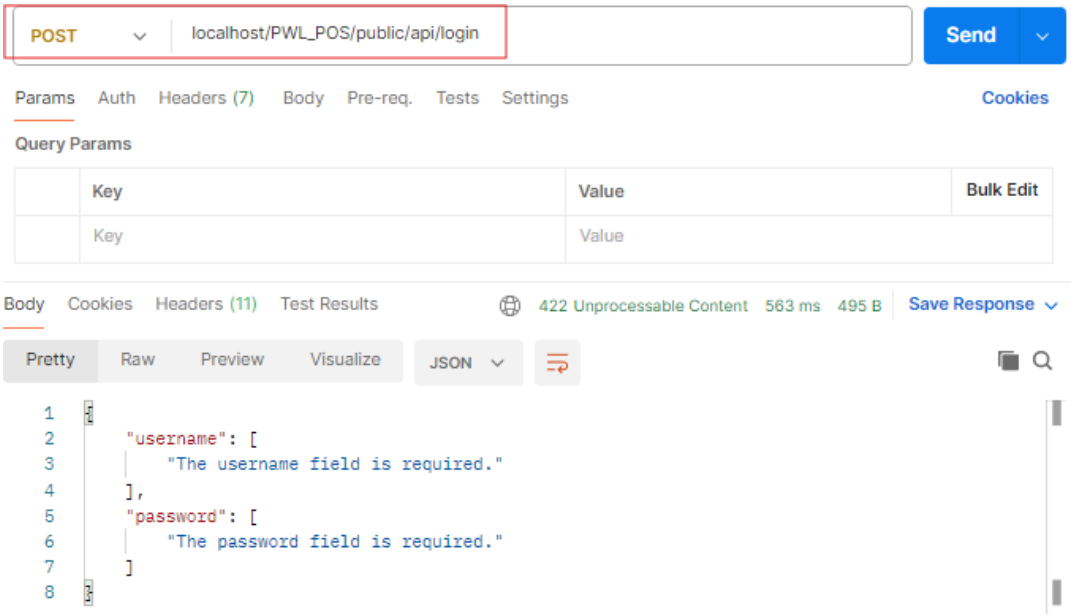
```php
use App\Http\Controllers\Api\LoginController;

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();
});
```

routes > api.php > ...
You, 27 seconds ago | 1 author (You)

```php
<?php

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

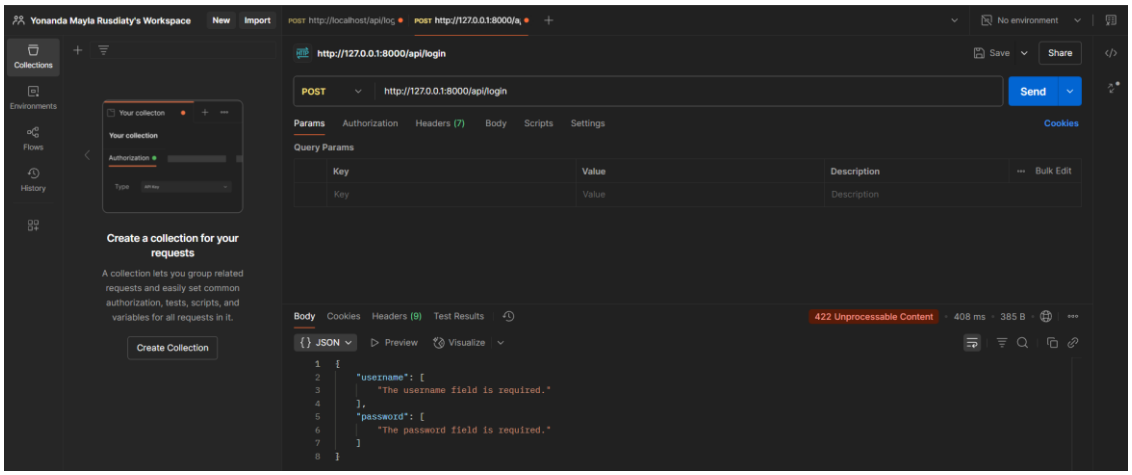
You, 1 second ago • Uncommitted changes

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.

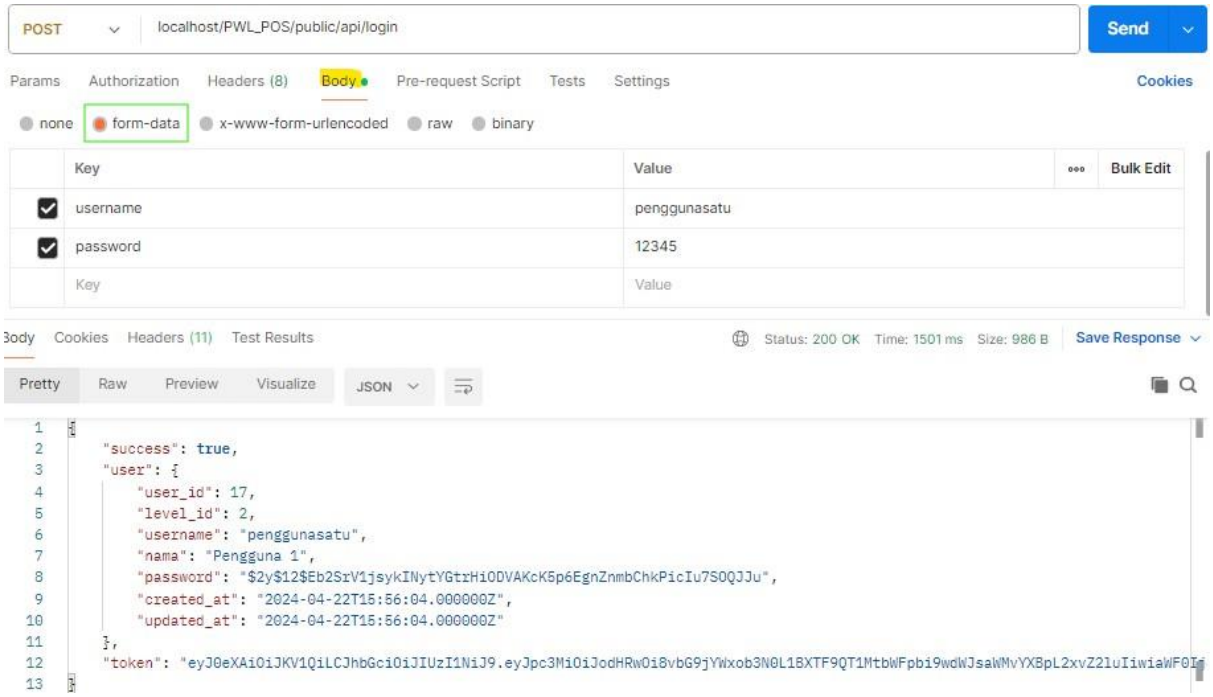


Jika berhasil akan muncul error validasi seperti gambar di atas.

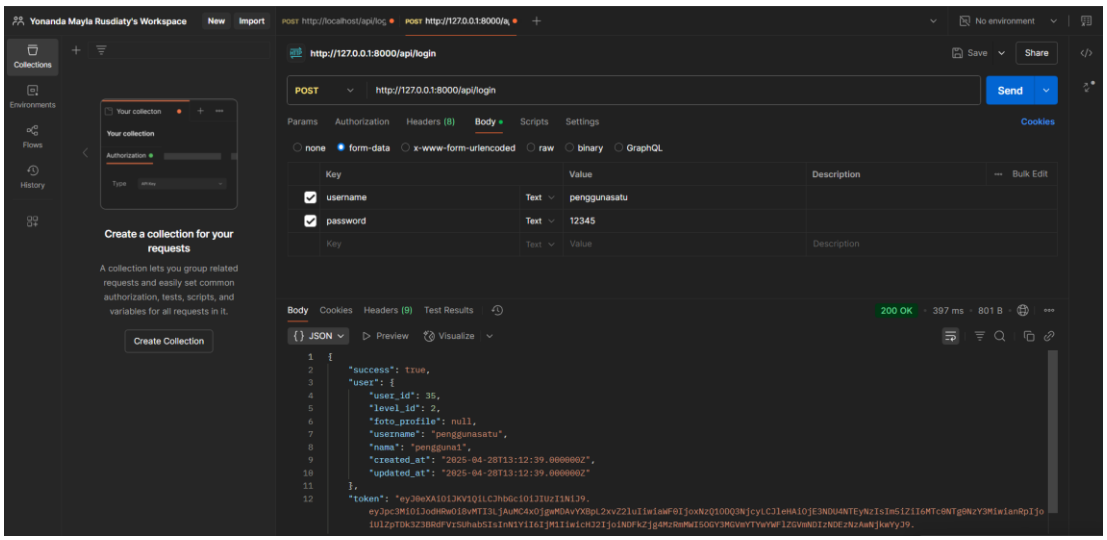
Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



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.

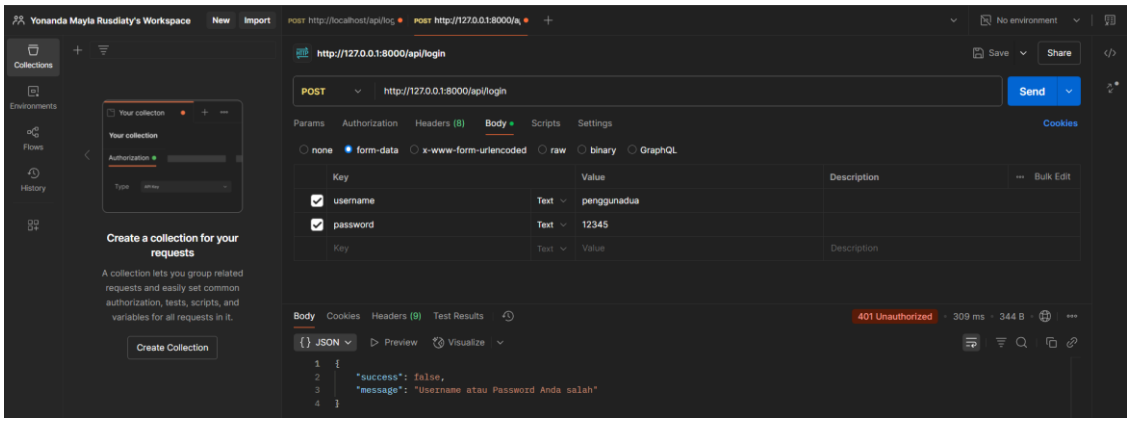


Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

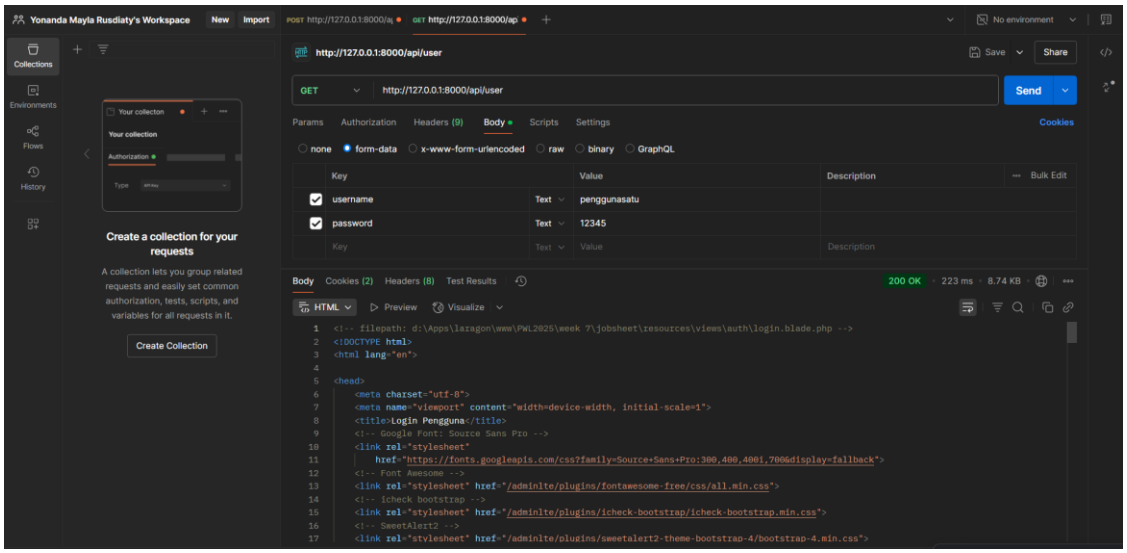


Jika kita mengirim username dan password ke alamat /api/login dengan metode POST, aplikasi akan memeriksa data tersebut dan memberikan token akses jika benar.

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

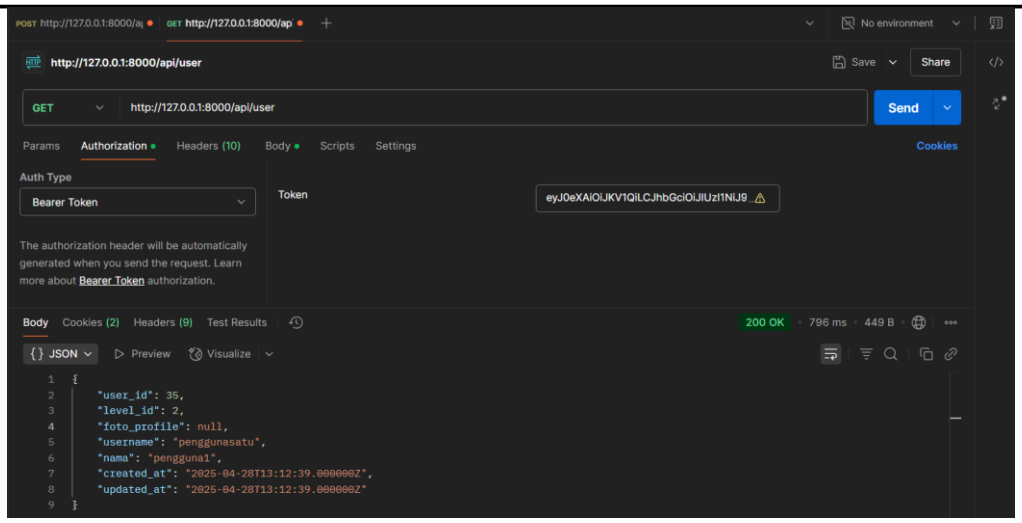


7. Coba kembali melakukan login dengan data yang benar. Sekarang mari kita coba menampilkan data user yang sedang login menggunakan URL localhost/PWL_POS/public/api/user dan method GET.
Jelaskan hasil dari percobaan tersebut.



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
```

```
.env
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
PUSHER_APP_CLUSTER=mt1

VITE_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
VITE_PUSHER_HOST="${PUSHER_HOST}"
VITE_PUSHER_PORT="${PUSHER_PORT}"
VITE_PUSHER_SCHEME="${PUSHER_SCHEME}"
VITE_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"

JWT_SECRET=MaIzmITYBINsHvQPk7XvVgXwBUfqfXPu8bJzp7lSIkLungLoOM0Nb6zG1ET2RpRx
JWT_SHOW_BLACKLIST_EXCEPTION=true
```

2. Buat Controller baru dengan nama LogoutController.

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

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

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

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!',
            ]);
        }
    }
}
```

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

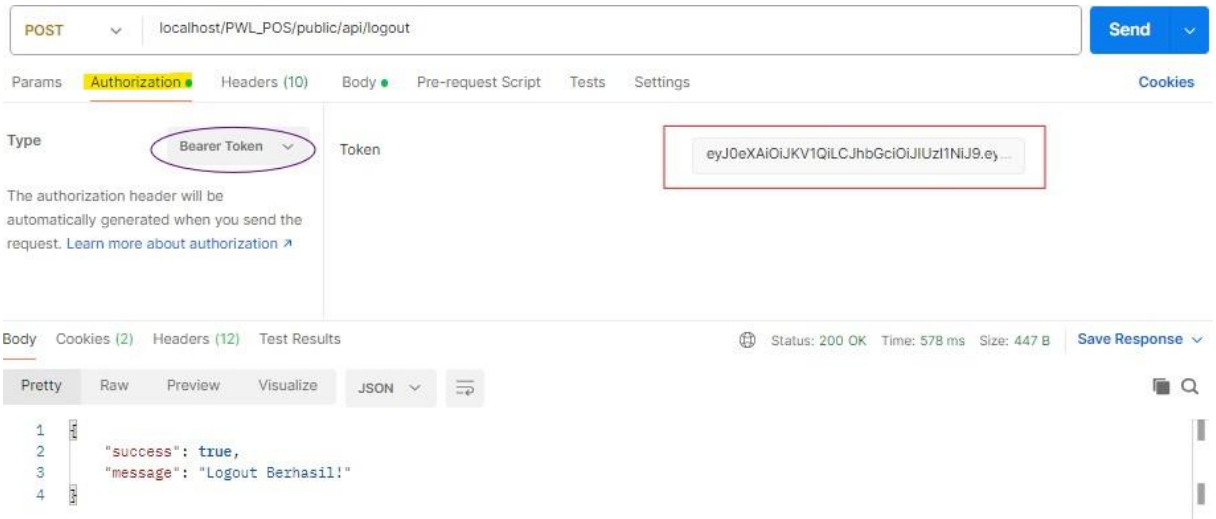
```php
<?php

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;

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

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

4. Lalu kita tambahkan routes pada api.php

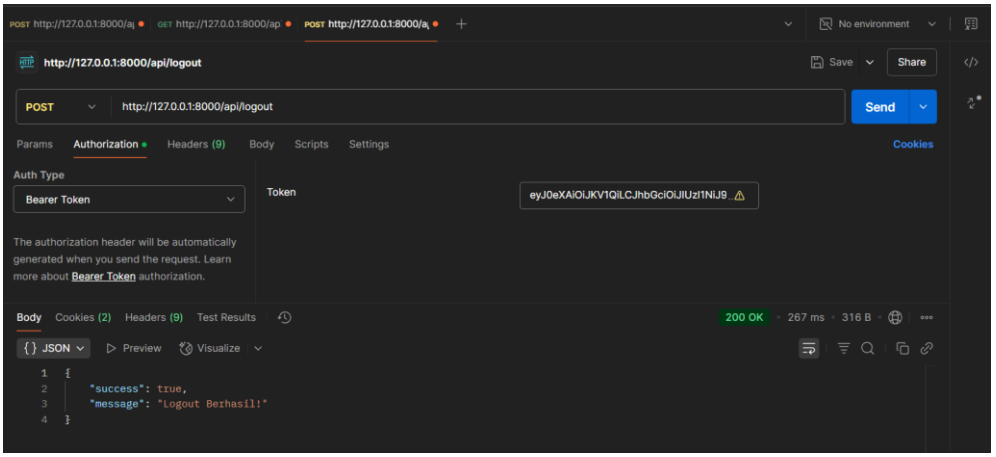
```
Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
```

```
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.

```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();
    }
```

```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
{
    0 references | 0 overrides
    public function index(): Collection
    {
        return LevelModel::all();
    }
```

```php
    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',
        ]);
    }
}
```

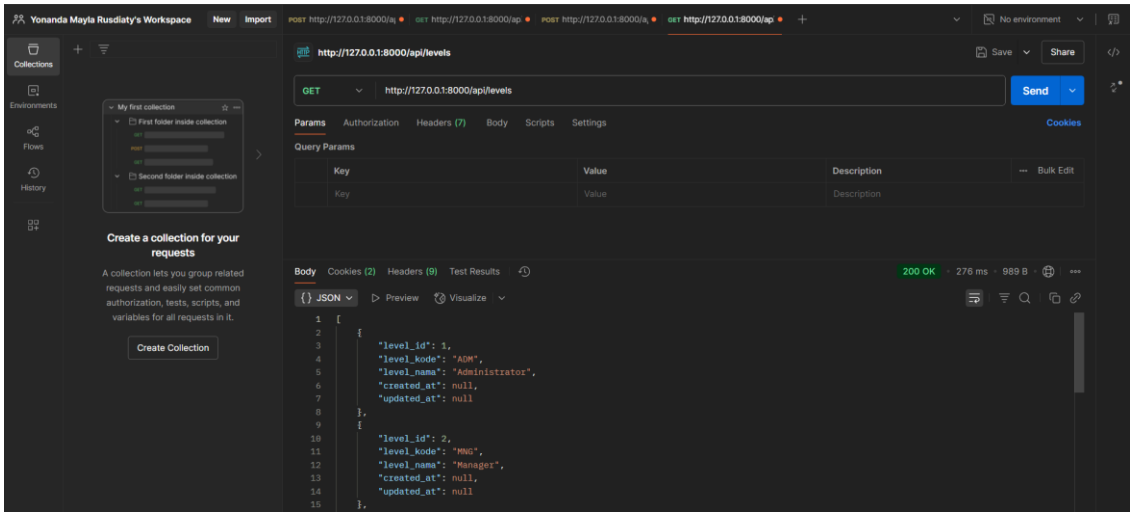
```php
0 references | 0 implementations
class LevelController extends Controller
{
    0 references | 0 overrides
    public function index(): Collection
    {
        return LevelModel::all();
    }

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

    0 references | 0 overrides
    public function show(LevelModel $level): array|Collection|LevelModel|Model|null
    {
        return LevelModel::find($level);
    }

    0 references | 0 overrides
    public function update(Request $request, LevelModel $level): array|Collection|LevelModel|Model|null
    {
        $level->update($request->all());
        return LevelModel::find($level);
    }

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

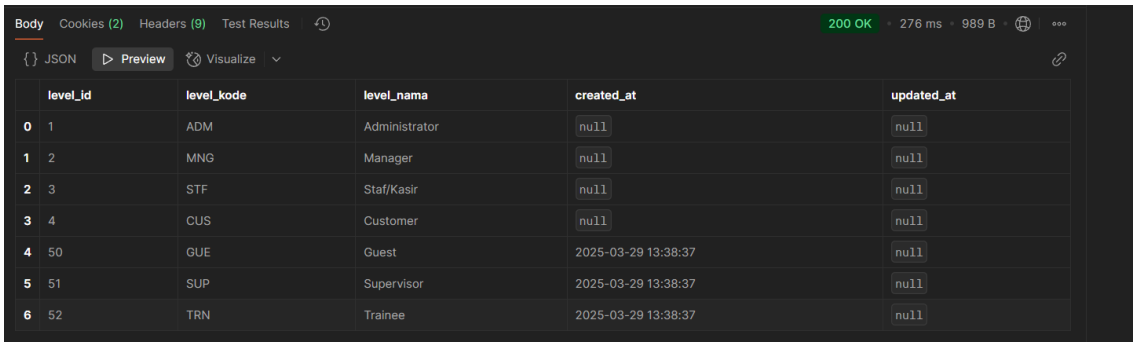
3. Kemudian kita lengkapi routes pada api.php.

```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.

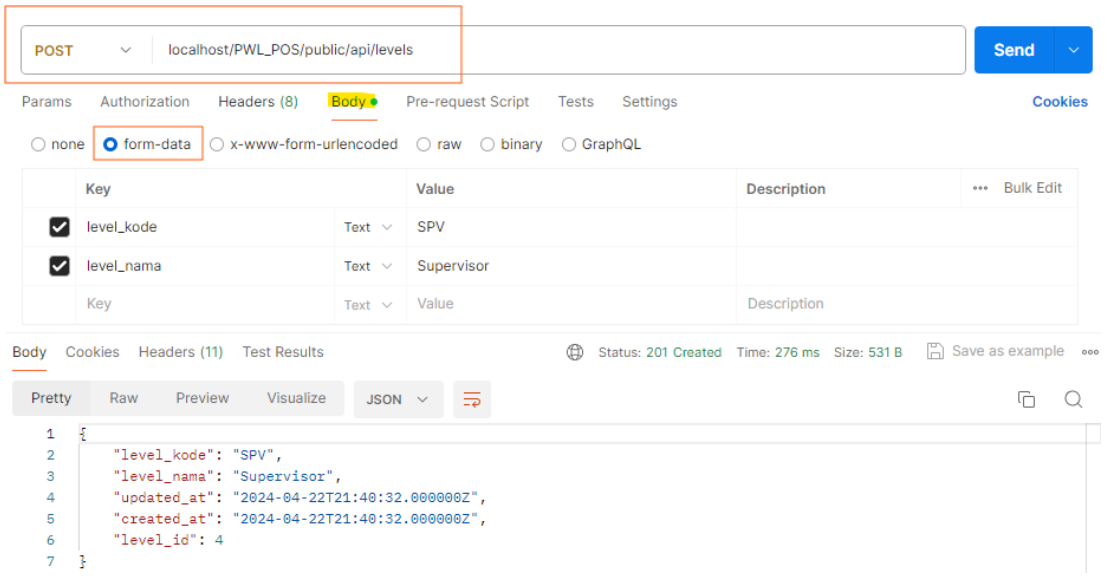




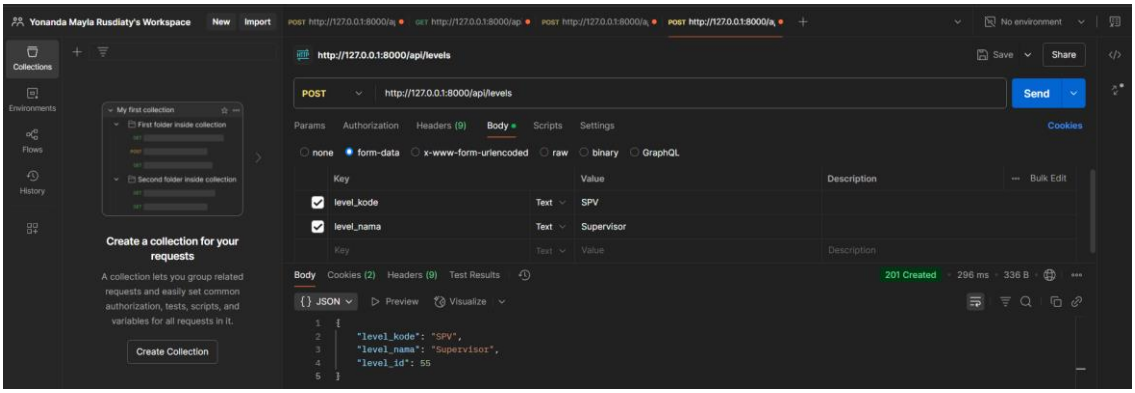
	level_id	level_kode	level_nama	created_at	updated_at
0	1	ADM	Administrator	null	null
1	2	MNG	Manager	null	null
2	3	STF	Staf/Kasir	null	null
3	4	CUS	Customer	null	null
4	50	GUE	Guest	2025-03-29 13:38:37	null
5	51	SUP	Supervisor	2025-03-29 13:38:37	null
6	52	TRN	Trainee	2025-03-29 13:38:37	null

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.



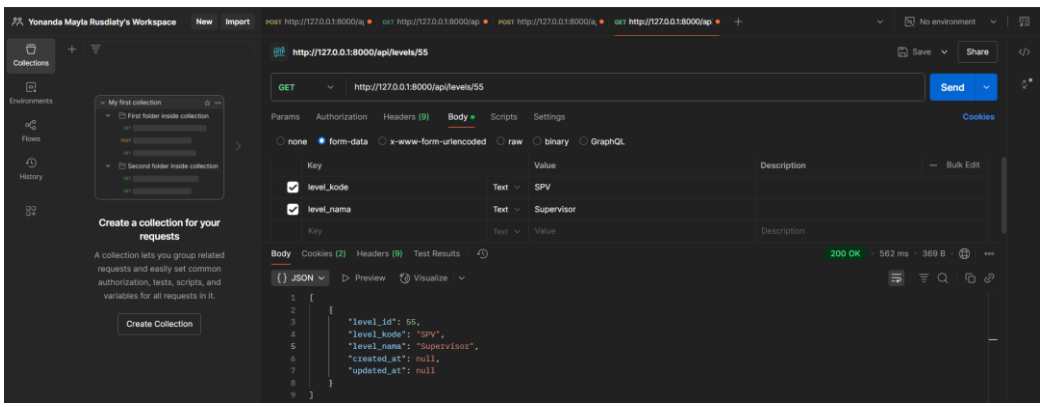
Jelaskan dan berikan screenshoot hasil percobaan Anda.



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.

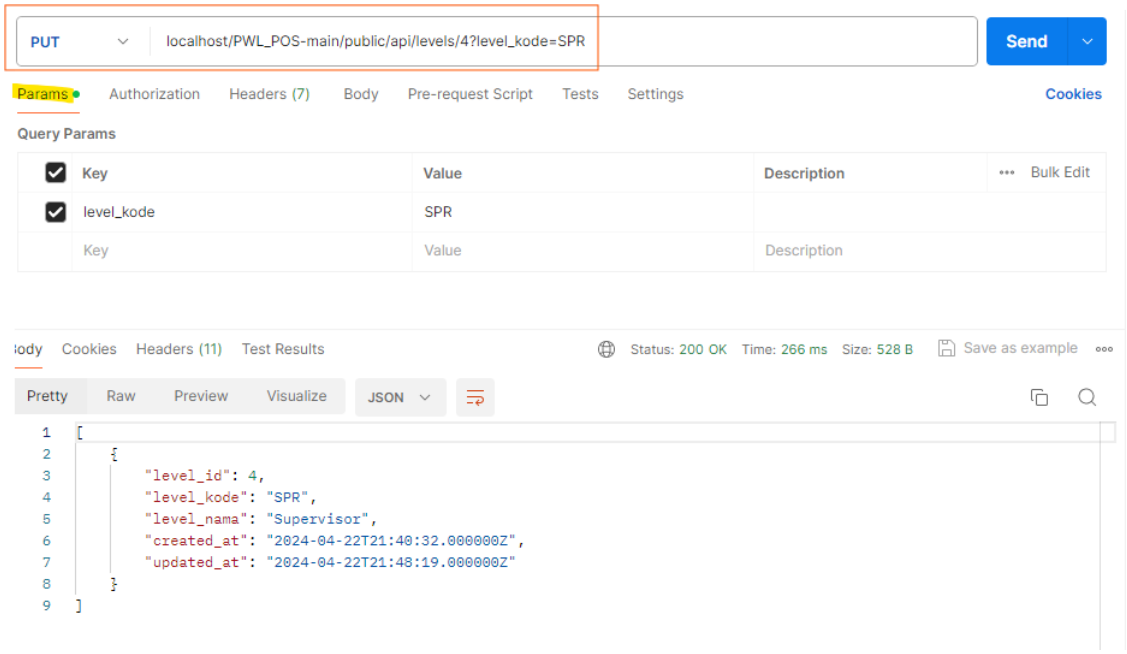


Disini kita sedang menampilkan detail data dengan endpoint /api/levels/ /{level}.

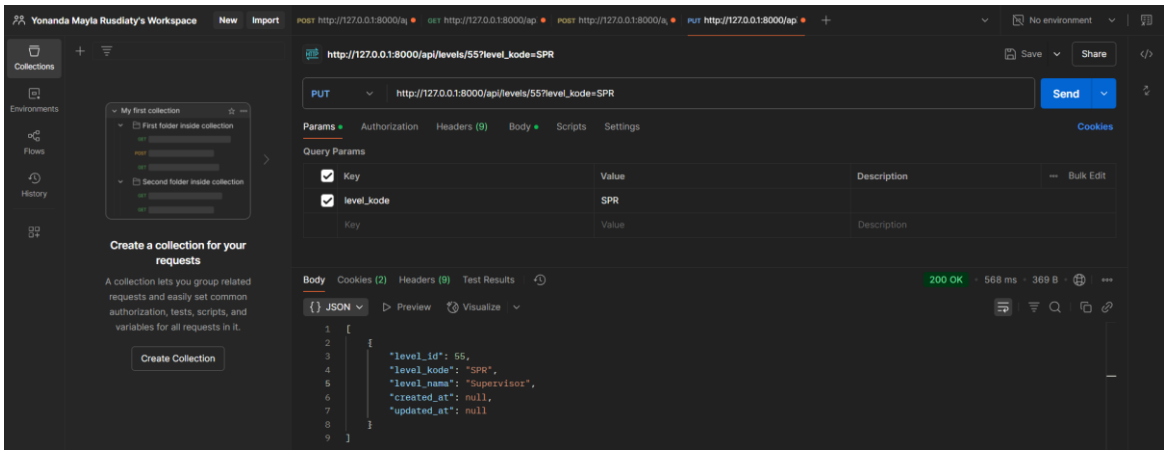
Karena saya ingin menampilkan detail data dari supervisor, maka endpointnya adalah

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.

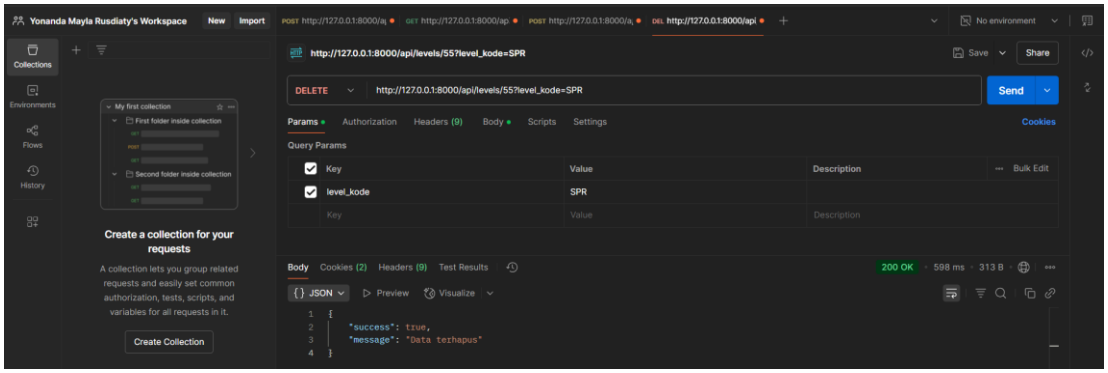


Jelaskan dan berikan screenshoot hasil percobaan Anda.



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:control
ler Api/UserController

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

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

```php
<?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();
```

```php
        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
     */
```

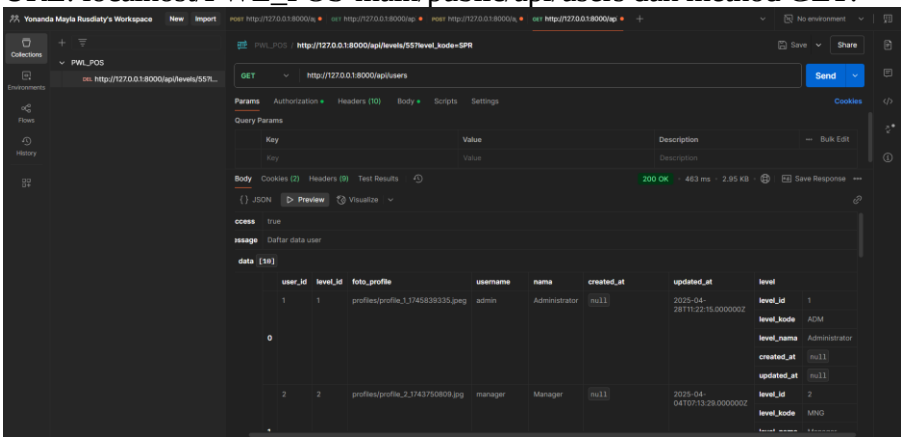
```
class UserController extends Controller
{
    1 reference | 0 overrides
    public function update(Request $request, $id): JsonResponse|mixed
    {
        $user = UserModel::find($id);

        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);
```

```php
public function update(Request $request, $id): JsonResponse|mixed

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

/**
 * Remove the specified user.
 *
 * @param  int  $id
 * @return \Illuminate\Http\JsonResponse
 */
1 reference | 0 overrides
public function destroy($id): JsonResponse|mixed
{
    $user = UserModel::find($id);

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

    $user->delete();

    return response()->json([
        'success' => true,
        'message' => 'User berhasil dihapus'
    ]);
}
}
```

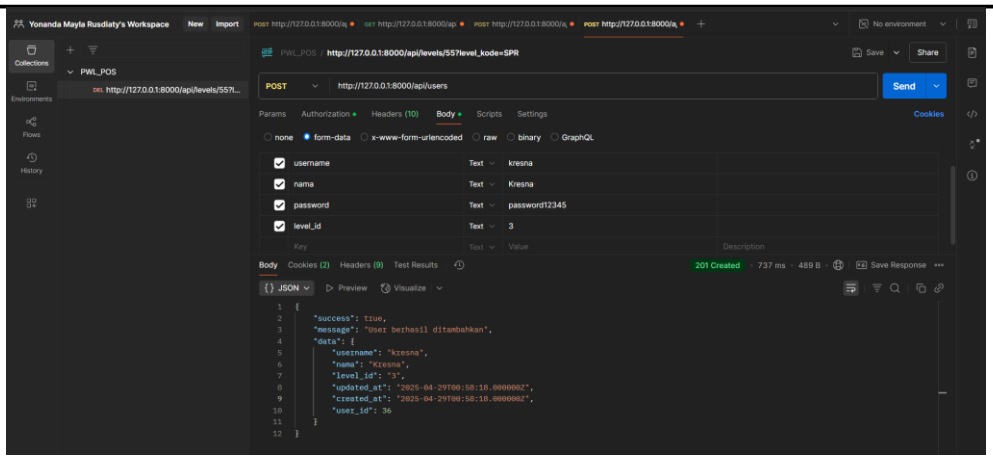
3. Kemudian kita lengkapi routes pada api.php.

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

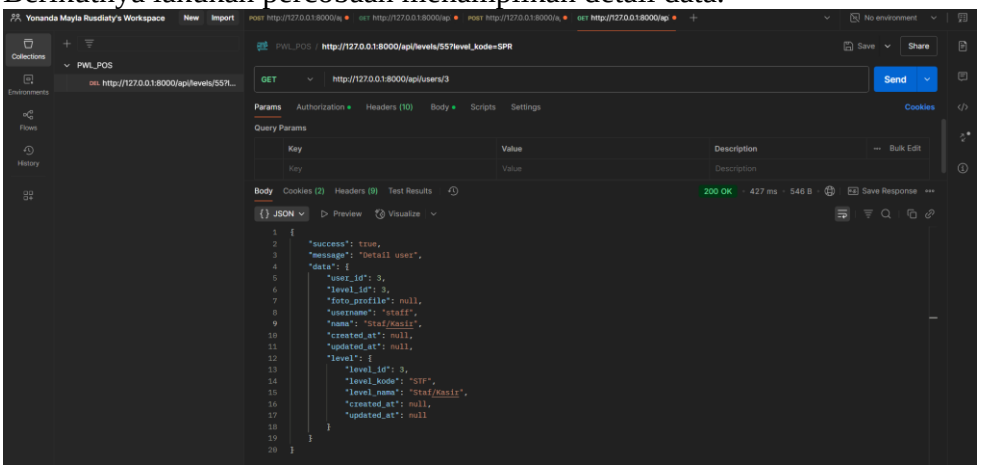
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/users dan method GET.



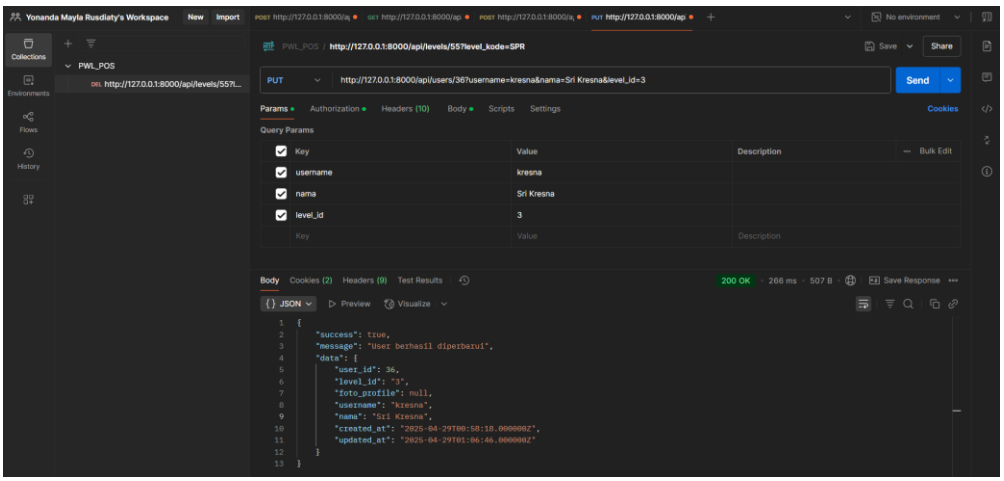
5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POS- main/public/api/users dan method POST seperti di bawah ini.



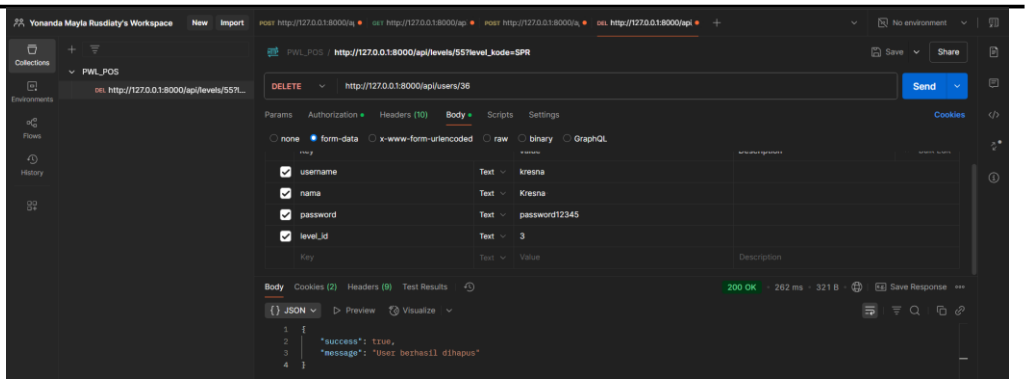
6. Berikutnya lakukan percobaan menampilkan detail data.



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_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>
```

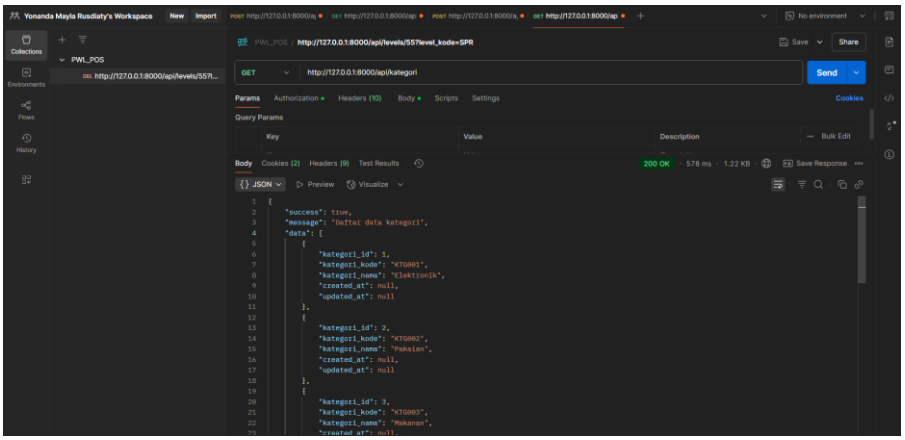
  2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya

```php
<?php

namespace App\Http\Controllers\Api;

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

class KategoriController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth:api');
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function index()
    {
        $kategori = KategoriModel::all();

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

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\JsonResponse
     */
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'kategori_kode' => 'required|string|max:10|unique:m_kategori,kategori_kode',
            'kategori_nama' => 'required|string|max:100',
        ]);

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

        $kategori = KategoriModel::create([
            'kategori_kode' => $request->kategori_kode,
            'kategori_nama' => $request->kategori_nama
        ]);

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

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function show($id)
    {
        $kategori = KategoriModel::find($id);

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

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

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function update(Request $request, $id)
    {
        $kategori = KategoriModel::find($id);

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

        $validator = Validator::make($request->all(), [
            'kategori_kode' => 'required|string|max:10|unique:m_kategori,kategori_kode,' . $id . ',kategori_id',
            'kategori_nama' => 'required|string|max:100',
        ]);

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

        $kategori->update([
            'kategori_kode' => $request->kategori_kode,
            'kategori_nama' => $request->kategori_nama
        ]);

        return response()->json([
            'success' => true,
            'message' => 'Kategori berhasil diperbarui',
            'data' => $kategori
        ]);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function destroy($id)
    {
        $kategori = KategoriModel::find($id);

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

        try {
            $kategori->delete();

            return response()->json([
                'success' => true,
                'message' => 'Kategori berhasil dihapus'
            ]);
        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => 'Kategori tidak dapat dihapus karena masih digunakan',
                'error' => $e->getMessage()
            ], 409);
        }
    }
}
```

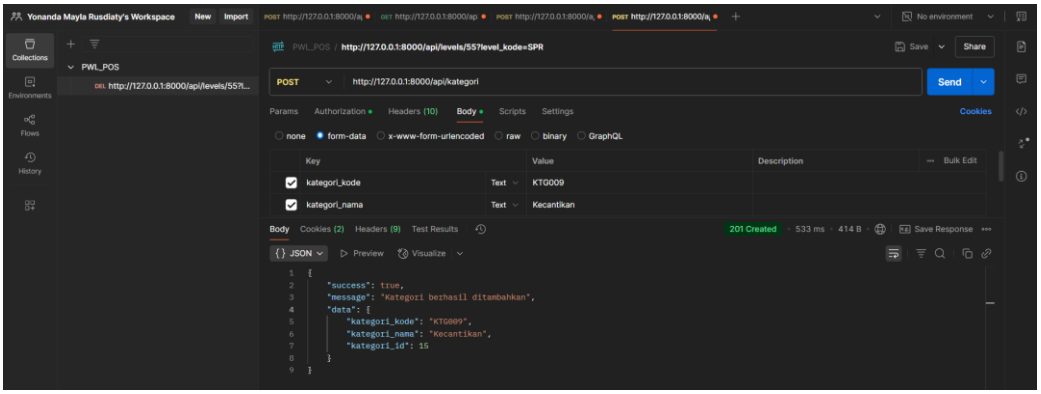
3. Kemudian kita lengkapi routes pada api.php.

```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']);
});      You, 22 minutes ago • feat(api): implement user CRUD operations via R…
```

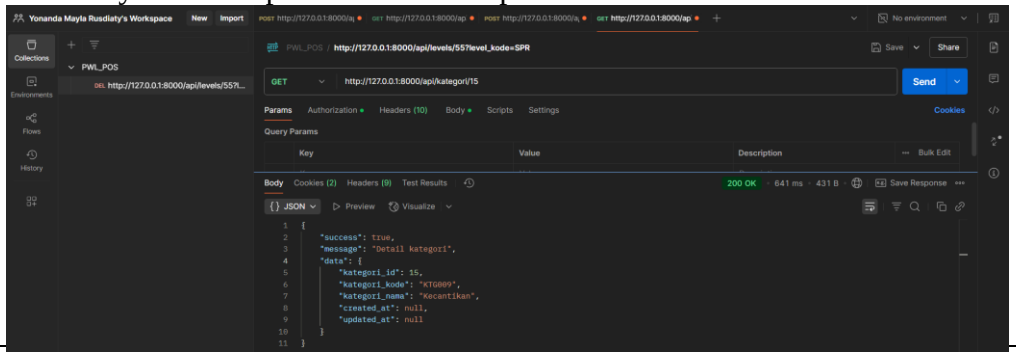
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/kategori dan method GET.



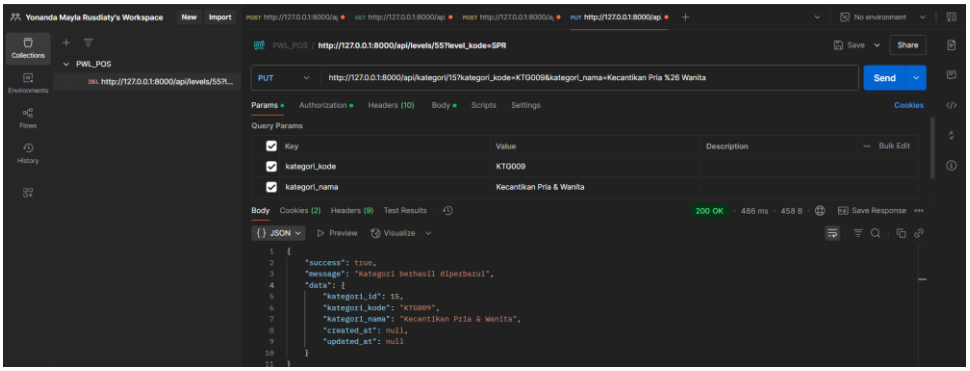
5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POS- main/public/api/kategori dan method POST seperti di bawah ini.



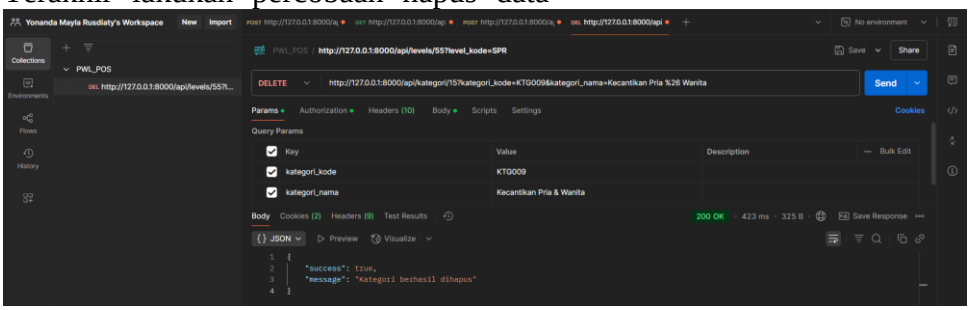
6. Berikutnya lakukan percobaan menampilkan detail data.



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.
```

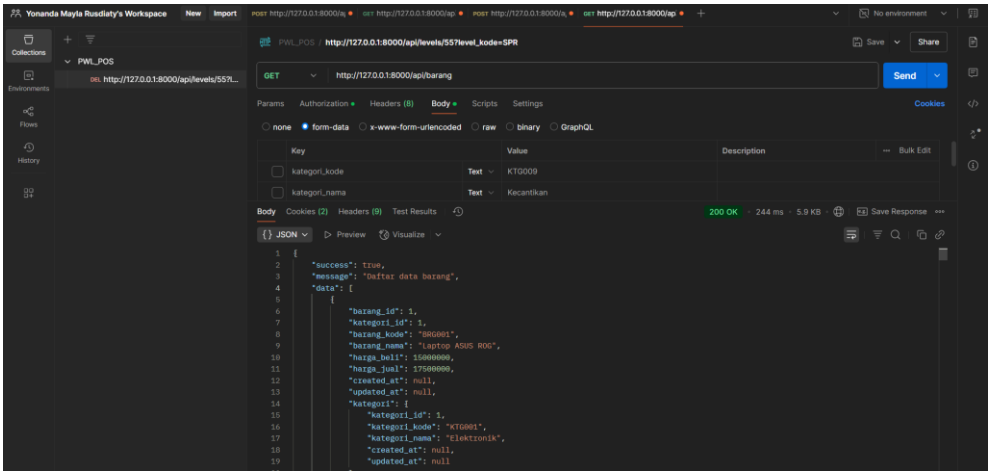
2. Setelah berhasil, buka file tersebut dan tuliskan kode seperti berikut yang berisi fungsi CRUDnya

```php
<?php

namespace App\Http\Controllers\Api;

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

class BarangController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth:api');
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\JsonResponse
     */
    public function index()
    {
        $barang = BarangModel::with('kategori')->get();

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

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\JsonResponse
     */
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'barang_kode' => 'required|string|max:10|unique:m_barang,barang_kode',
            'barang_nama' => 'required|string|max:100',
            'harga' => 'required|numeric|min:0',
            'stok' => 'required|integer|min:0',
            'kategori_id' => 'required|exists:m_kategori,kategori_id',
        ]);

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

        $barang = BarangModel::create([
            'barang_kode' => $request->barang_kode,
            'barang_nama' => $request->barang_nama,
            'harga' => $request->harga,
            'stok' => $request->stok,
            'kategori_id' => $request->kategori_id,
        ]);

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

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function show($id)
    {
        $barang = BarangModel::with('kategori')->find($id);

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

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

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function update(Request $request, $id)
    {
        $barang = BarangModel::find($id);

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

        $validator = Validator::make($request->all(), [
            'barang_kode' => 'required|string|max:10|unique:m_barang,barang_kode,' . $id . ',barang_id',
            'barang_nama' => 'required|string|max:100',
            'harga' => 'required|numeric|min:0',
            'stok' => 'required|integer|min:0',
            'kategori_id' => 'required|exists:m_kategori,kategori_id',
        ]);

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

        $barang->update([
            'barang_kode' => $request->barang_kode,
            'barang_nama' => $request->barang_nama,
            'harga' => $request->harga,
            'stok' => $request->stok,
            'kategori_id' => $request->kategori_id,
        ]);

        return response()->json([
            'success' => true,
            'message' => 'Barang berhasil diperbarui',
            'data' => $barang
        ]);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function destroy($id)
    {
        $barang = BarangModel::find($id);

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

        try {
            $barang->delete();

            return response()->json([
                'success' => true,
                'message' => 'Barang berhasil dihapus'
            ]);
        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => 'Barang tidak dapat dihapus karena masih digunakan',
                'error' => $e->getMessage()
            ], 409);
        }
    }
}
```

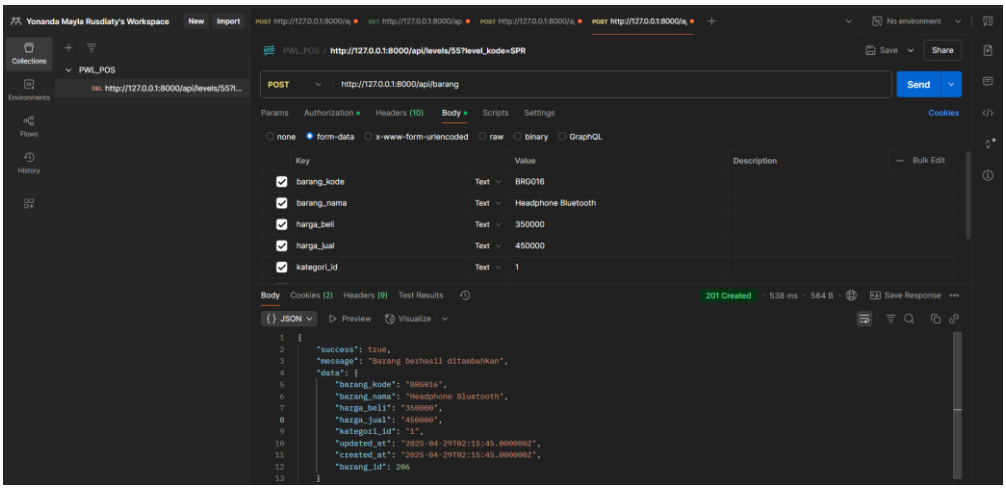
3. Kemudian kita lengkapi routes pada api.php.

```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']);
});
```

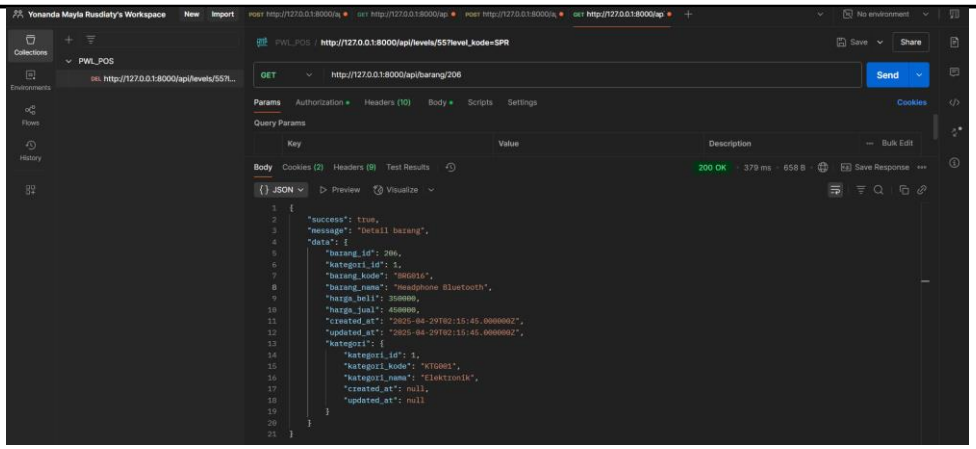
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL_POS-main/public/api/barang dan method GET.



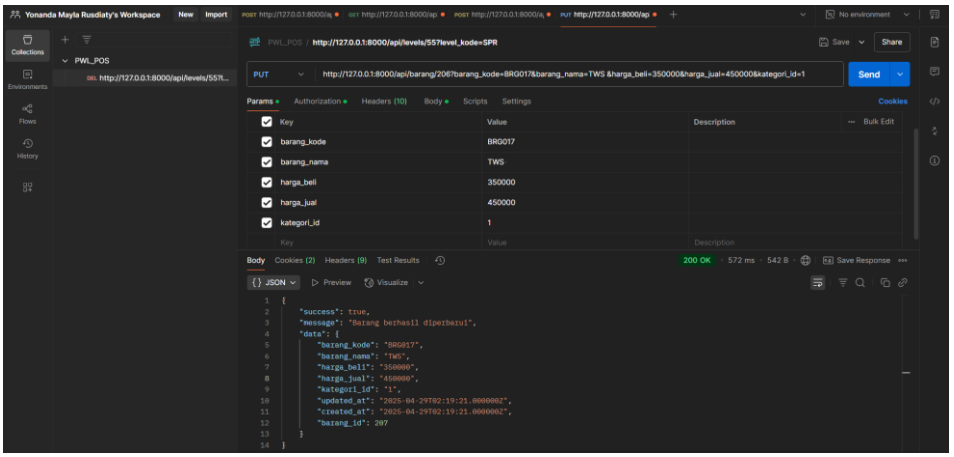
5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POS- main/public/api/barang dan method POST seperti di bawah ini.



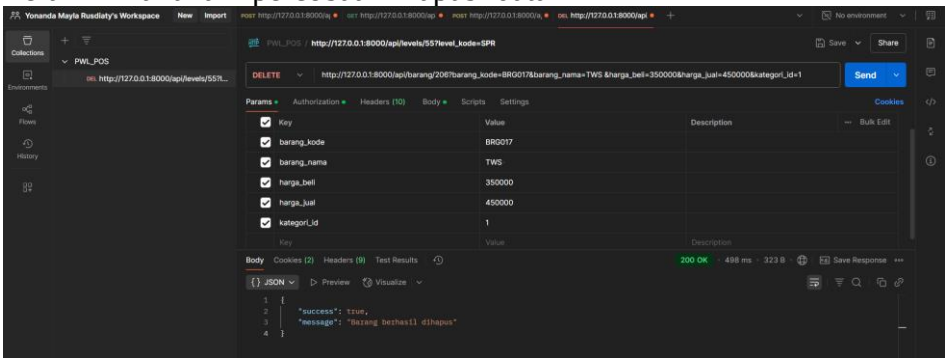
6. Berikutnya lakukan percobaan menampilkan detail data.



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 ****