

# **JOBSHEET 10**

## **LAPORAN TUGAS**

**Diajukan untuk Memenuhi Tugas  
Pemograman Web Lanjut**

**Jurusan:**

**TEKNOLOGI INFORMASI**

**Program Studi:**

**TEKNIK INFORMATIKA**

**Disusun oleh**

**1. Afrizal Dwi Septian (2241720122) / 01 / 2H**



**POLITEKNIK NEGERI MALANG  
TEKNIK INFORMATIKA**

**2024**



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 dipercaya.

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 :

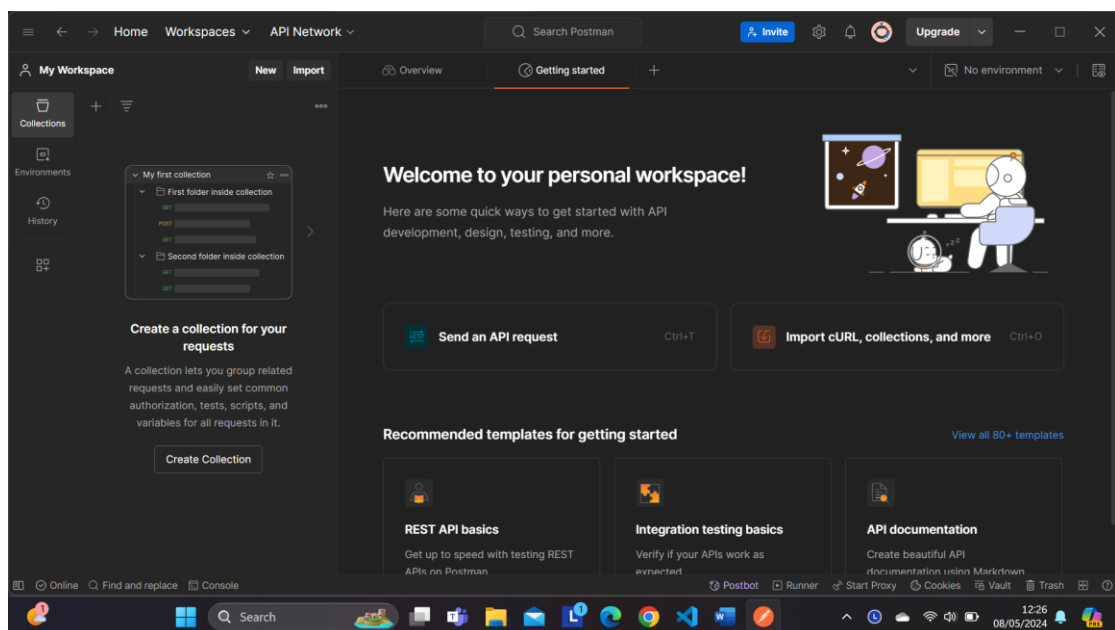
JWT (JSON Web Token) adalah cara untuk mentransfer informasi antara dua pihak secara aman sebagai objek JSON. Ini terdiri dari tiga bagian: header, payload, dan signature. Setelah pengguna berhasil autentikasi, server menghasilkan token JWT yang disematkan dalam permintaan HTTP. Server kemudian memvalidasi 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 tyson/jwt-auth:2.1.1` Pastikan Anda terkoneksi dengan internet.



```
C:\laragon\www\PWL_POS>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 (2.3.0)
  - 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 (2.3.0)
  - 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 (2.3.0): Extracting archive
  - Installing lcobucci/jwt (4.0.4): Extracting archive
  - Installing tymon/jwt-auth (2.1.1): Extracting archive
Generating optimized autoload files
Class App\Http\Controllers\POSController located in C:/laragon/www/PWL_POS/app/Http/Controllers/POSController.php does not comply with psr-4 autoloading standard. Skipping.
Class App\Models\penjualanDetailModel located in C:/laragon/www/PWL_POS/app/Models/PenjualanDetailModel.php does not comply with psr-4 autoloading standard. Skipping.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

jeroennoten/laravel-adminlte ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
laravel/ui ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE
tymon/jwt-auth ..... DONE
yajra/laravel-datatables-buttons ..... DONE
yajra/laravel-datatables-editor ..... DONE
yajra/laravel-datatables-fractal ..... DONE
yajra/laravel-datatables-html ..... DONE
yajra/laravel-datatables-oracle ..... DONE

92 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

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

```
php artisan vendor:publish --
provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
```

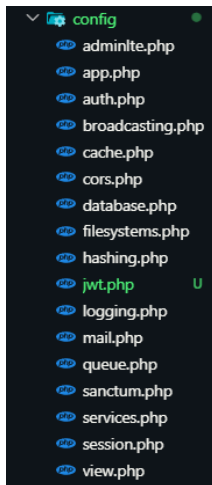
```
C:\laragon\www\PWL_POS>php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\LaravelServicePr
vider"

 INFO  Publishing assets.

Copying file [C:\laragon\www\PWL_POS\vendor\tymon\jwt-auth\config\config.php] to [C:\laragon\www\PWL
_POS\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.

```
C:\laragon\www\PWL_POS>php artisan jwt:secret
jwt-auth secret [IhHh2ixPTX7jHN16GbFZIyeC81c2cZemhkViB4E03eR4cnVWkIgrunYzIWUE4GVH] set successfully.

61  JWT_SECRET=IhHh2ixPTX7jHN16GbFZIyeC81c2cZemhkViB4E03eR4cnVWkIgrunYzIWUE4GVH
62
```

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';  
}
```

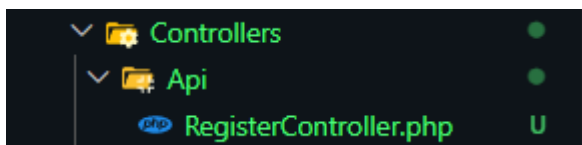


```
app > Models > UserModel.php > UserModel
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\Relations\BelongsTo;
8  use App\Models\LevelModel;
9  use Illuminate\Foundation\Auth\User as userAuthentication;
10 use Tymon\JWTAuth\Contracts\JWTSubject;
11
12 class UserModel extends userAuthentication implements JWTSubject
13 {
14     public function getJWTIdentifier(){
15         return $this->getKey();
16     }
17
18     public function getJWTCustomClaims(){
19         return [];
20     }
21 }
```

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.



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     {
```





```
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 }
```



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

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');
```



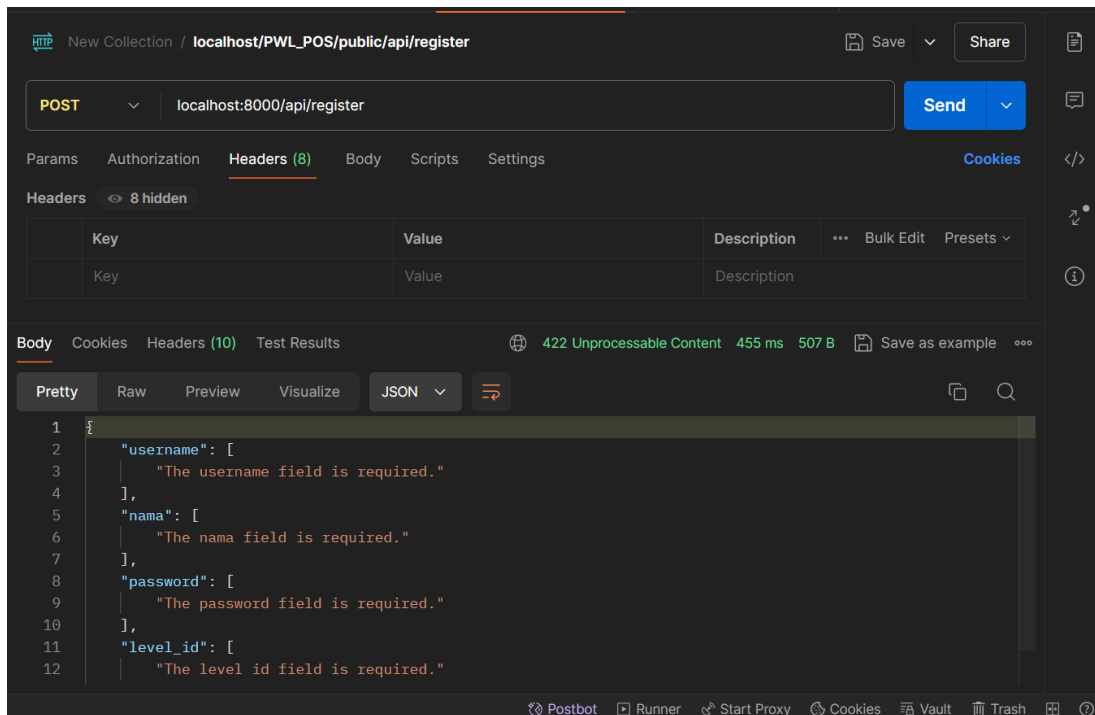
```
routes > api.php
1  <?php
2
3  use Illuminate\Http\Request;
4  use Illuminate\Support\Facades\Route;
5  use App\Http\Controllers\Api\RegisterController;
6
7  /*
8  |-----
9  | API Routes
10 |-----
11 |
12 | Here is where you can register API routes for your application. These
13 | routes are loaded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

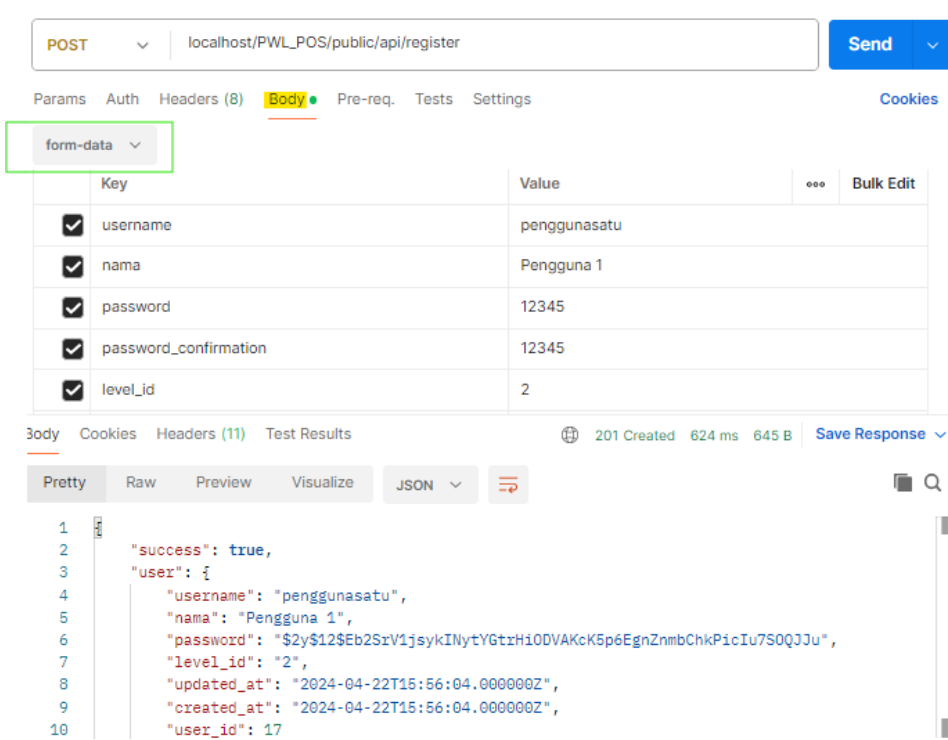


Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



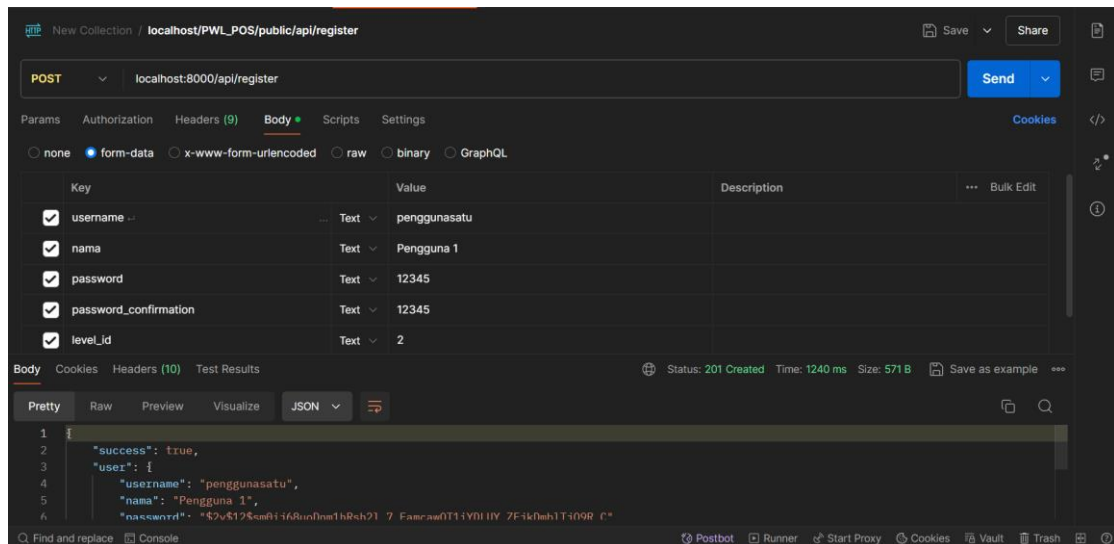
12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.





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.

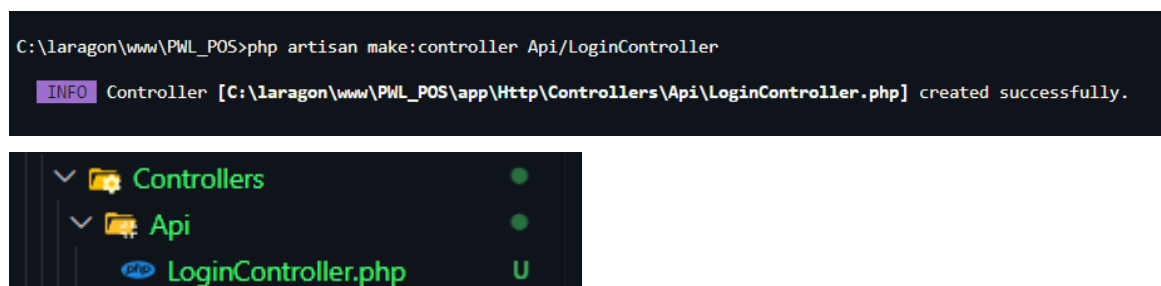


13. Lakukan commit perubahan file pada Github.

## Praktikum 2 – Membuat RESTful API Login

1. Kita buat file controller dengan nama LoginController. `php artisan make:controller Api/LoginController`

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





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

```
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
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()->user(),
39             'token' => $token
40         ], 200);
41     }
42 }
```





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

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();
});

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

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.



POST localhost/PWL\_POS/public/api/login Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body Cookies Headers (11) Test Results 422 Unprocessable Content 563 ms 495 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "username": [
3     "The username field is required."
4   ],
5   "password": [
6     "The password field is required."
7   ]
8 }
```

Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

localhost:8000/api/login Save Share

POST localhost:8000/api/login Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results Status: 422 Unprocessable Content Time: 220 ms Size: 421 B Save as example

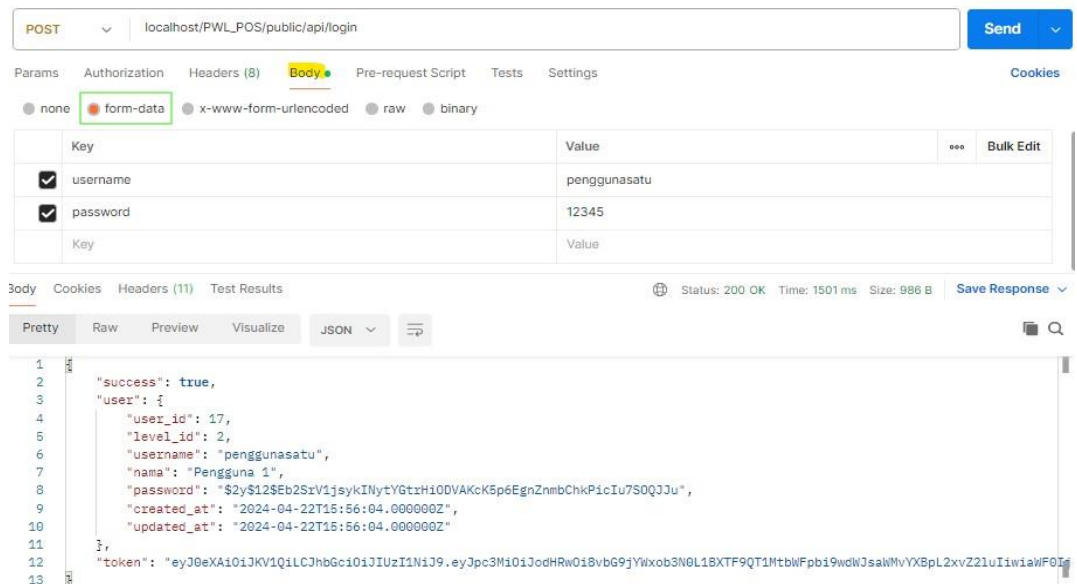
Pretty Raw Preview Visualize JSON

```
1 {
2   "username": [
3     "The username field is required."
4   ],
5   "password": [
6     "The password field is required."
7   ]
8 }
```

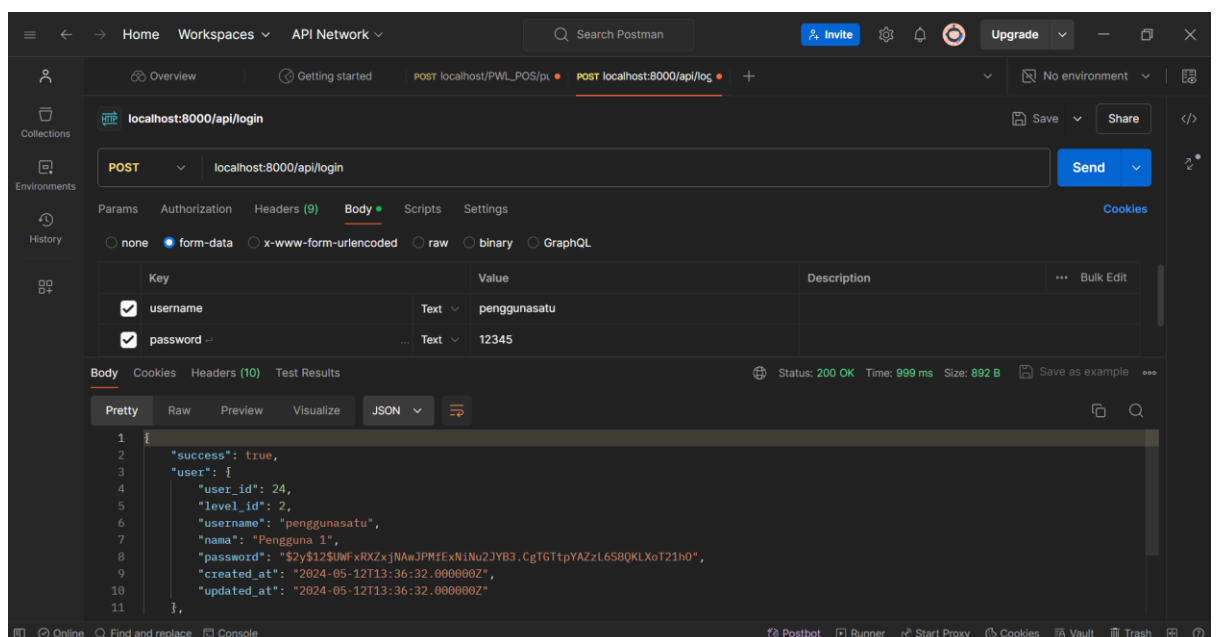




5. Selanjutnya, isikan username dan password sesuai dengan data user yang ada pada database. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data user. Klik tombol Send, jika berhasil maka akan keluar tampilan seperti berikut. Copy nilai token yang diperoleh pada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

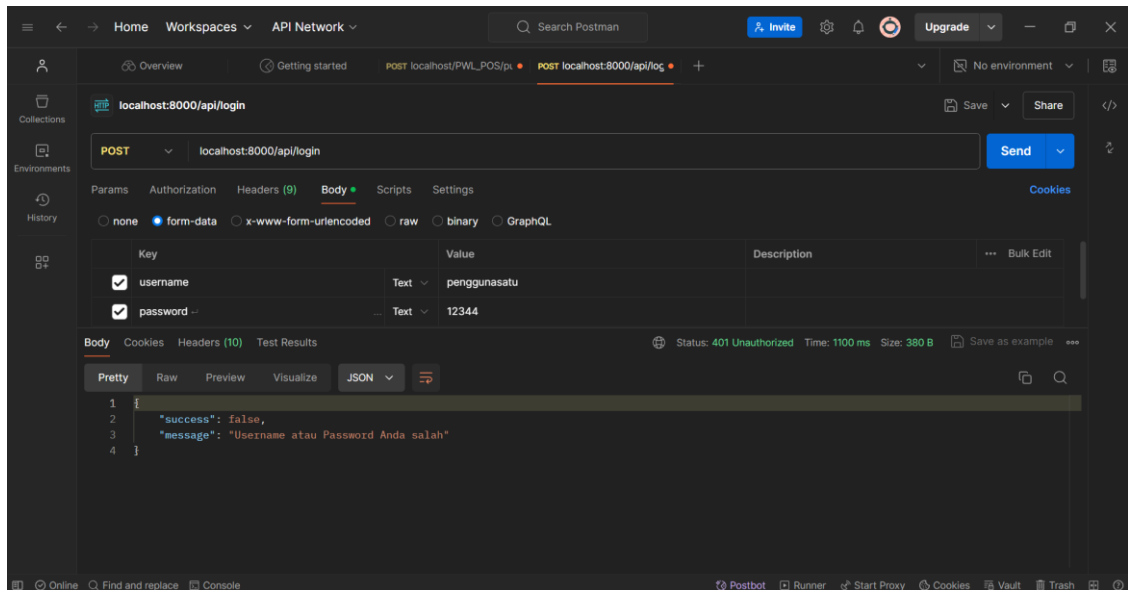


Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

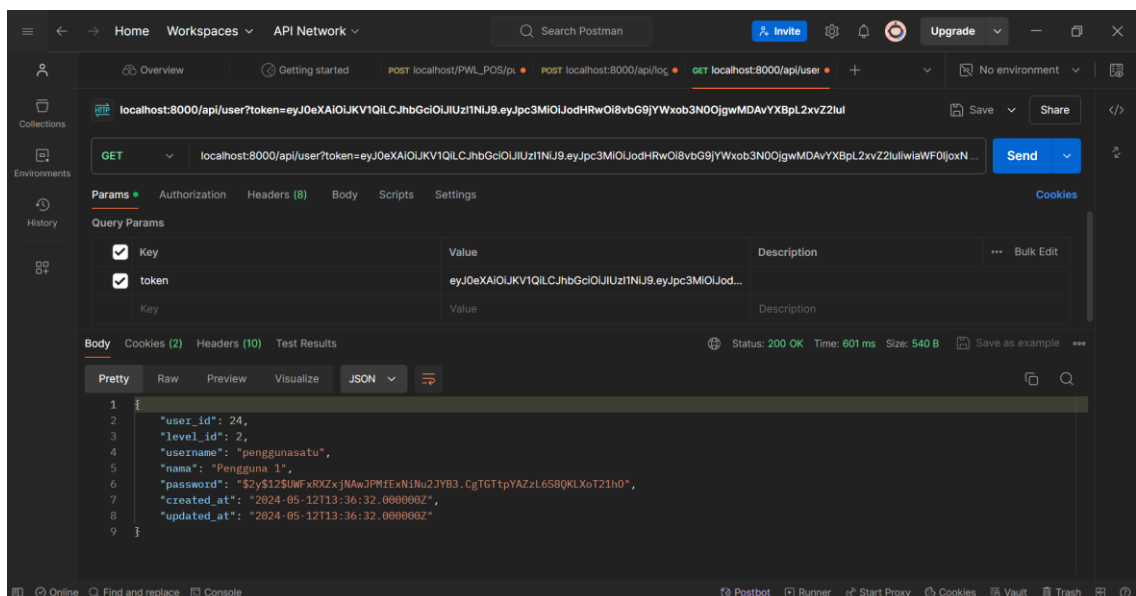




6. Lakukan percobaan yang untuk data yang salah dan berikan screenshoot hasil percobaan Anda.



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



8. Lakukan commit perubahan file pada Github.



### Praktikum 3 – Membuat RESTful API Logout

1. Tambahkan kode berikut pada file .env

`JWT_SHOW_BLACKLIST_EXCEPTION=true`

```
.env
61 JWT_SECRET=IhHh2ixPTX7jHn16GbFZIyeC81c2cZemhkViB4E03eR4cnVwkIgrunYzIWUE4GVH
62 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 > ...
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Http\Request;
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
13 class LogoutController extends Controller
14 {
15     public function __invoke(Request $request)
16     {
17         //remove token
18         $removeToken = JWTAuth::invalidate(JWTAuth::getToken());
19
20         if ($removeToken) {
21             //return response JSON
22             return response()->json( data: [
23                 'success' => true,
24                 'message' => 'Logout Berhasil!',
25             ]);
26         }
27     }
28 }
```

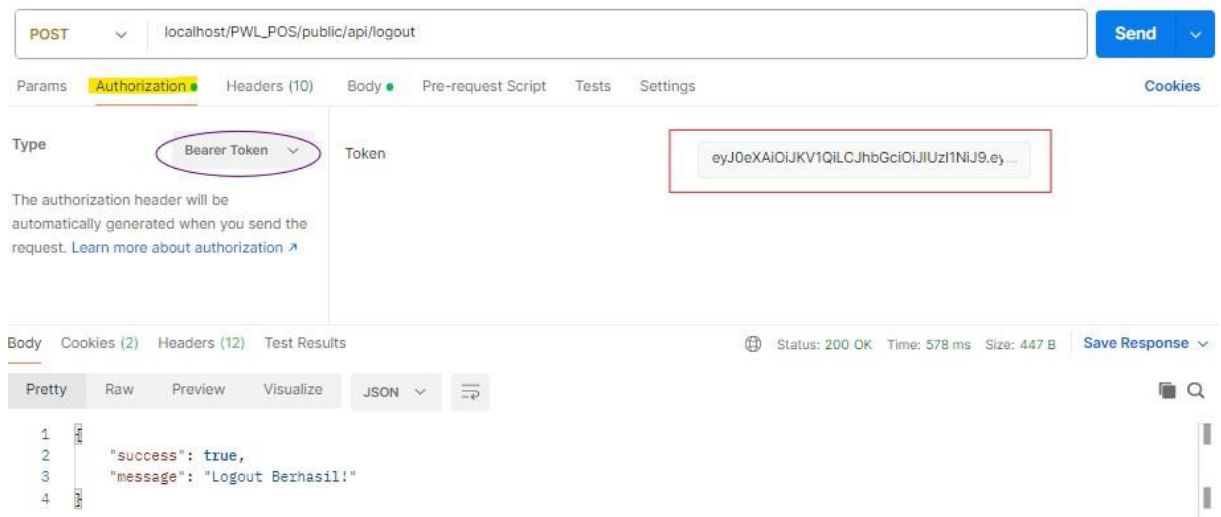
4. Lalu kita tambahkan routes pada api.php

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

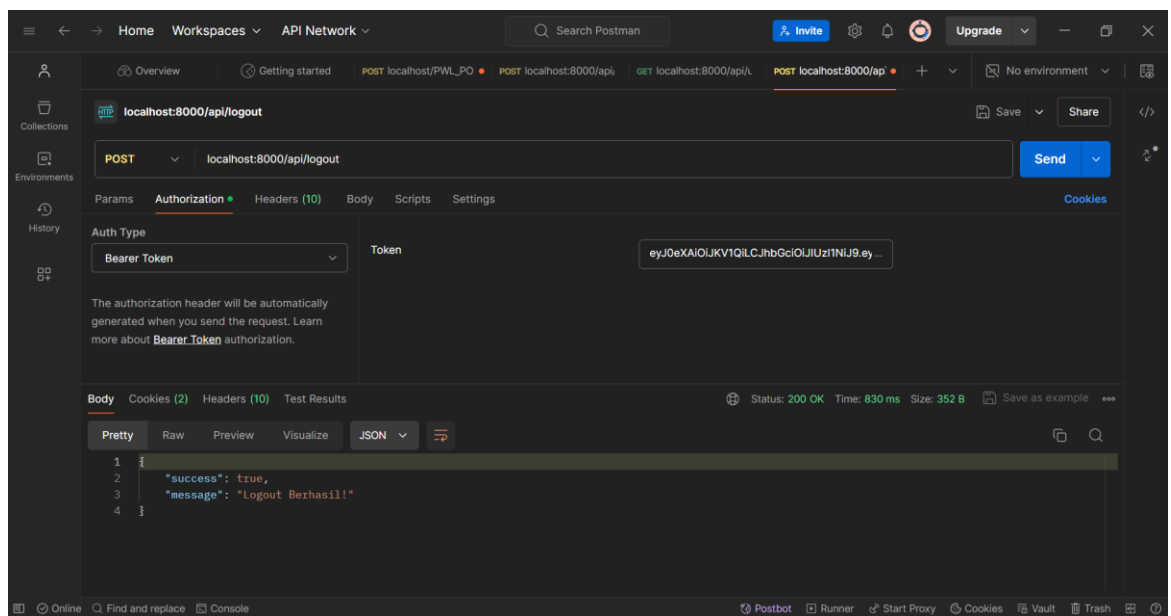
Route::post(uri: '/register', action: App\Http\Controllers\Api\RegisterController::class)->name(name: 'register');
Route::post(uri: '/login', action: App\Http\Controllers\Api\LoginController::class)->name(name: 'login');
Route::middleware(middleware: 'auth:api')->get(uri: '/user', action: function (Request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

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

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



Lakukan percobaan yang sama dan berikan screenshot hasil percobaan Anda.



7. Lakukan commit perubahan file pada Github.



## Praktikum 4 – Implementasi CRUD dalam RESTful API

---

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

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

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

```
C:\laragon\www\PWL_POS>php artisan make:controller Api/LevelController
```

```
INFO Controller [C:\laragon\www\PWL_POS\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();
    }
}
```



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

```
app > Http > Controllers > Api > LevelController.php > LevelController > destroy
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7 use App\Models\LevelModel;
8
9 class LevelController extends Controller
10 {
11     public function index()
12     {
13         return LevelModel::all();
14     }
15
16     public function store(Request $request)
17     {
18         $level = LevelModel::create($request->all());
19         return response()->json($level, 201);
20     }
21
22     public function show(LevelModel $level)
23     {
24         return response()->json($level, 200);
25     }
26
27     public function update(Request $request, LevelModel $level)
28     {
29         $level->update($request->all());
30         return response()->json($level, 200);
31     }
32
33     public function destroy(LevelModel $level)
34     {
35         $level->delete();
36
37         return response()->json([
38             'success' => true,
39             'message' => 'Data terhapus',
40         ]);
41     }
42 }
```





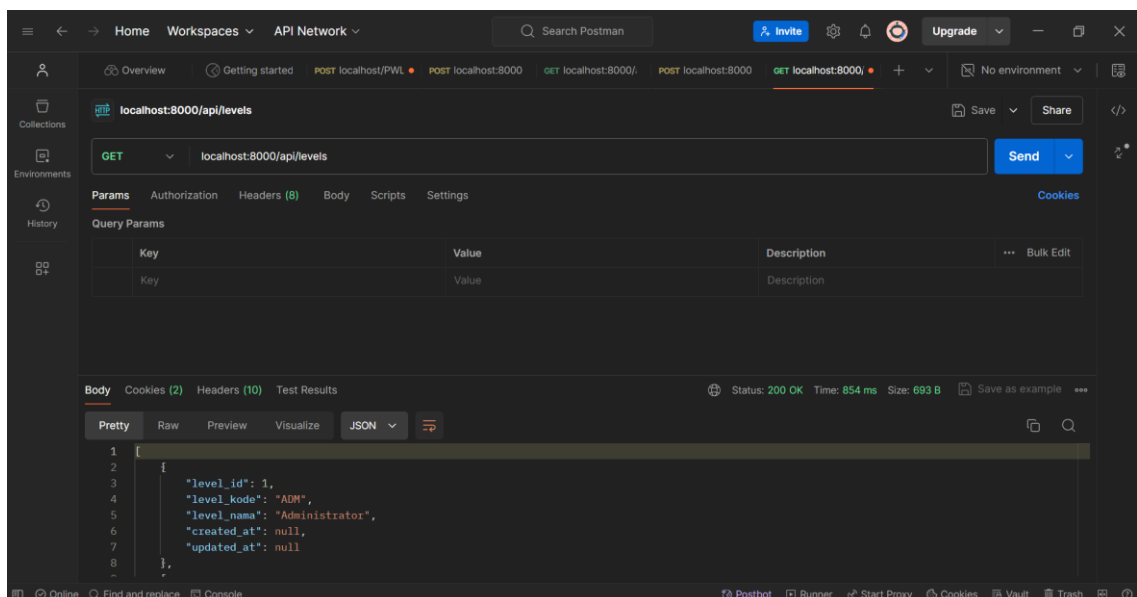
3. Kemudian kita lengkapi routes pada api.php.

```
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']);

Route::get(uri: 'levels', action: [LevelController::class, 'index']);
Route::post(uri: 'levels', action: [LevelController::class, 'store']);
Route::get(uri: 'levels/{level}', action: [LevelController::class, 'show']);
Route::put(uri: 'levels/{level}', action: [LevelController::class, 'update']);
Route::delete(uri: 'levels/{level}', action: [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 screenshot hasil percobaan Anda.**



Jawab: Menampilkan data dari database pada tabel level menggunakan metode get





5. Kemudian, lakukan percobaan penambahan data dengan URL :  
localhost/PWL\_POSmain/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 screenshot hasil percobaan Anda.

POST localhost:8000/api/levels

Params Authorization Headers (10) **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 (10) Test Results Status: 201 Created Time: 472 ms Size: 457 B Save as example

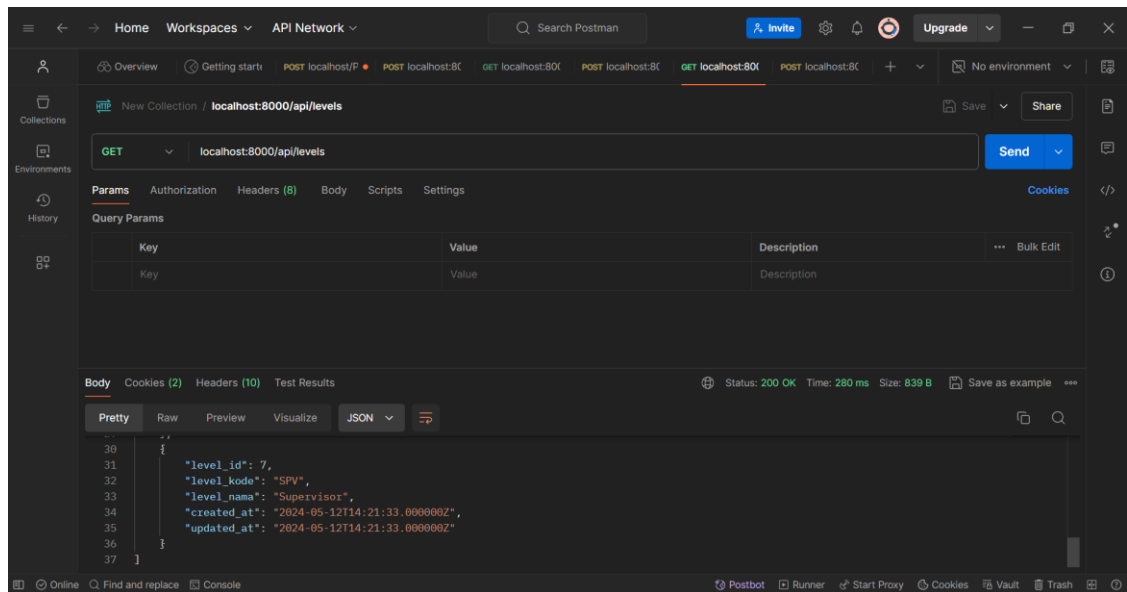
Pretty Raw Preview Visualize JSON

```
1 {
2   "level_kode": "SPV",
3   "level_nama": "Supervisor",
4   "updated_at": "2024-05-12T14:21:33.000000Z",
5   "created_at": "2024-05-12T14:21:33.000000Z",
6   "level_id": 7
7 }
```

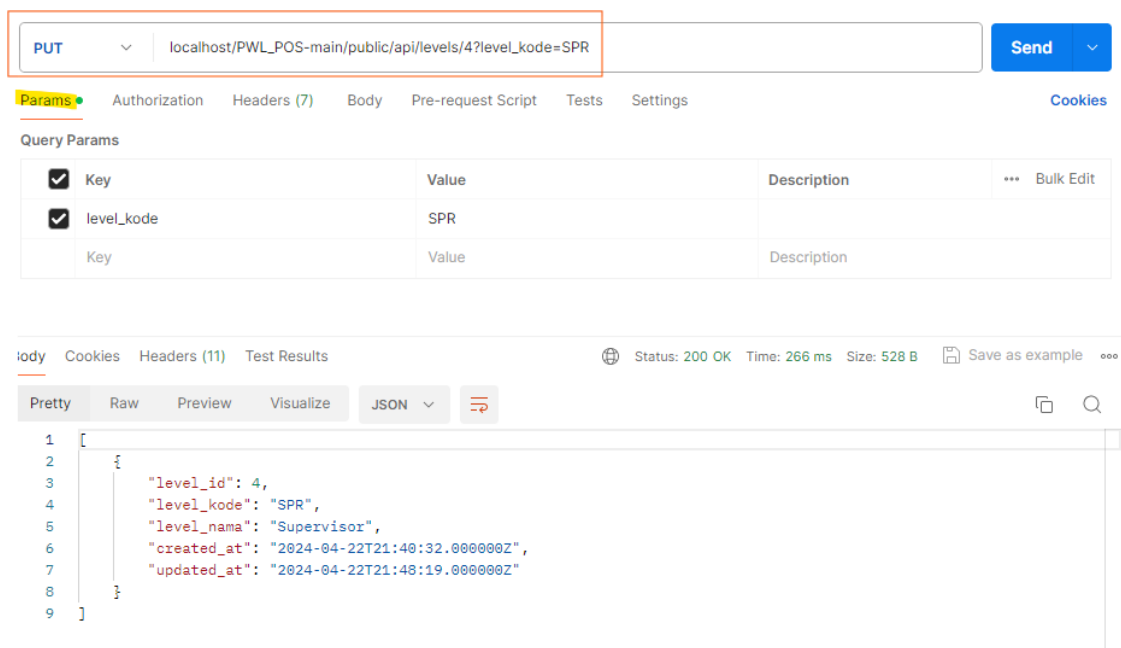
Jawab: Menambahkan data SPV ke dalam database level menggunakan metode POST



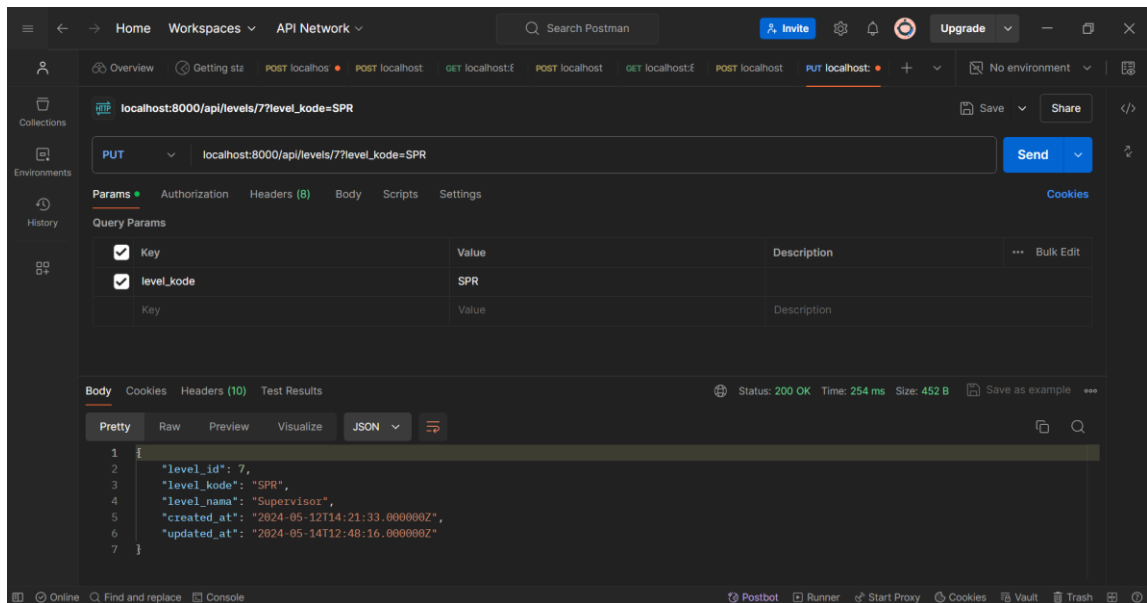
6. Berikutnya lakukan percobaan menampilkan detail data. **Jelaskan dan berikan screenshot hasil percobaan Anda.**



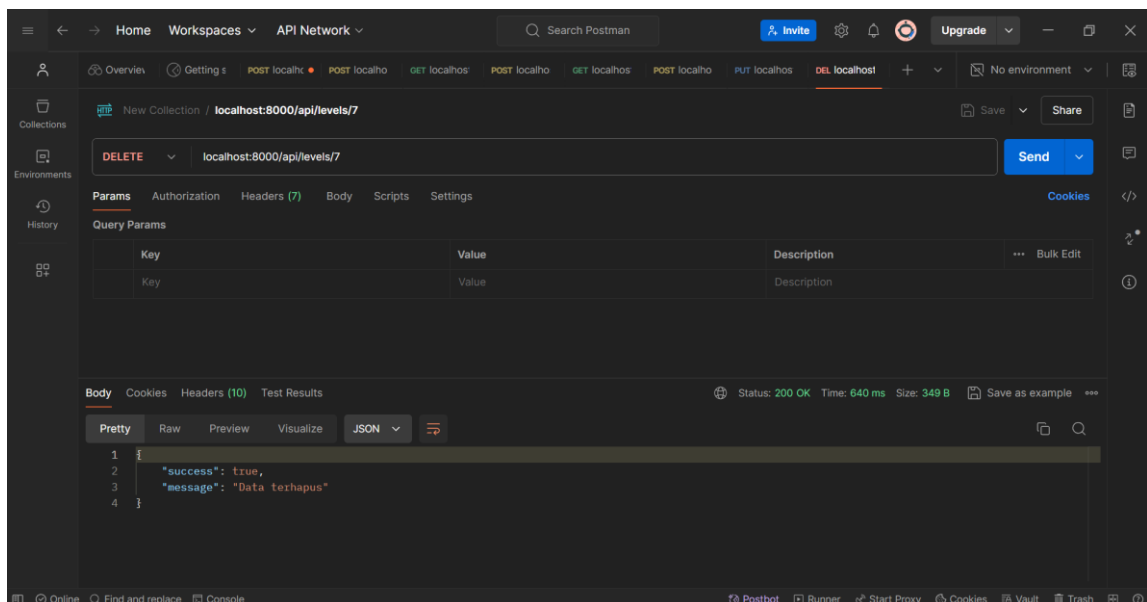
7. Jika sudah, kita coba untuk melakukan edit data menggunakan `localhost/PWL_POSmain/public/api/levels/{id}` dan method PUT. Isikan data yang ingin diubah pada tab Param.



**Jelaskan dan berikan screenshot hasil percobaan Anda.**



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

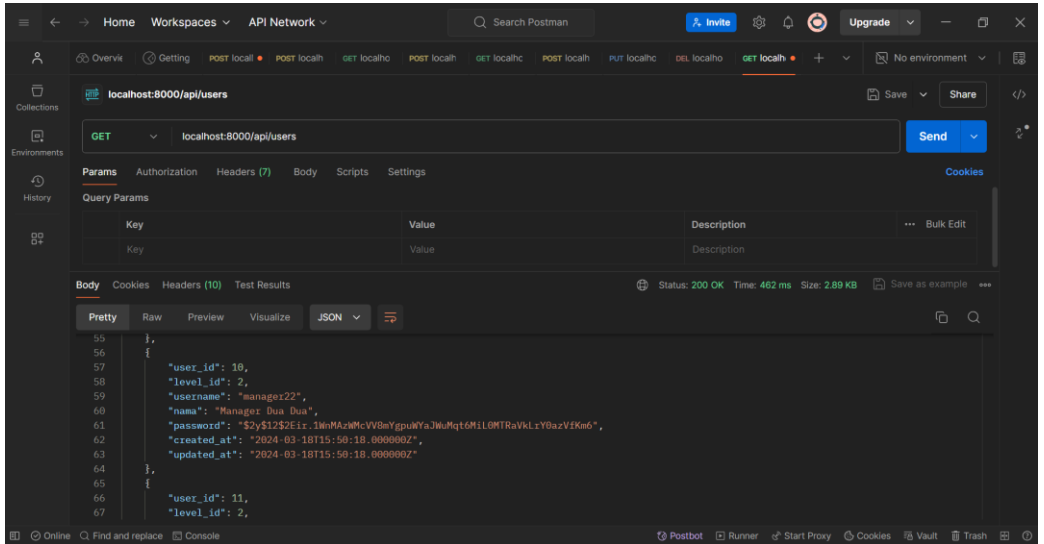
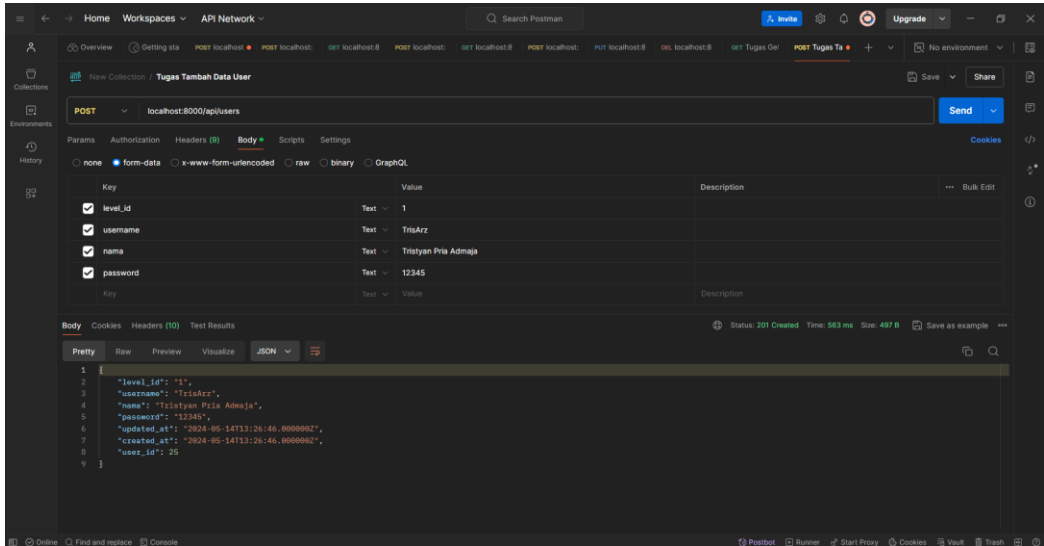


9. Lakukan commit perubahan file pada Github.



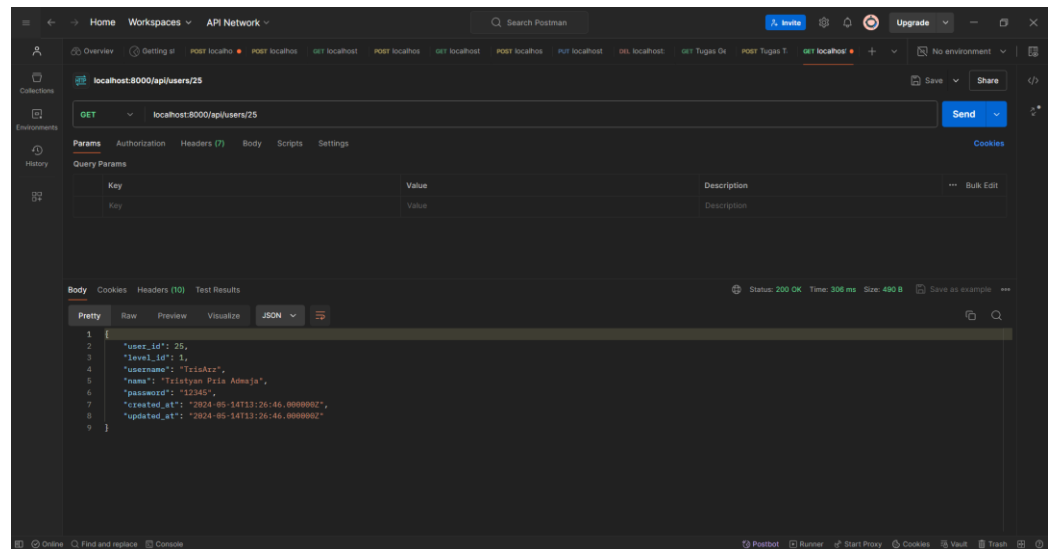
## TUGAS

Implementasikan CRUD API pada tabel lainnya yaitu tabel m\_user, m\_kategori, dan m\_barang

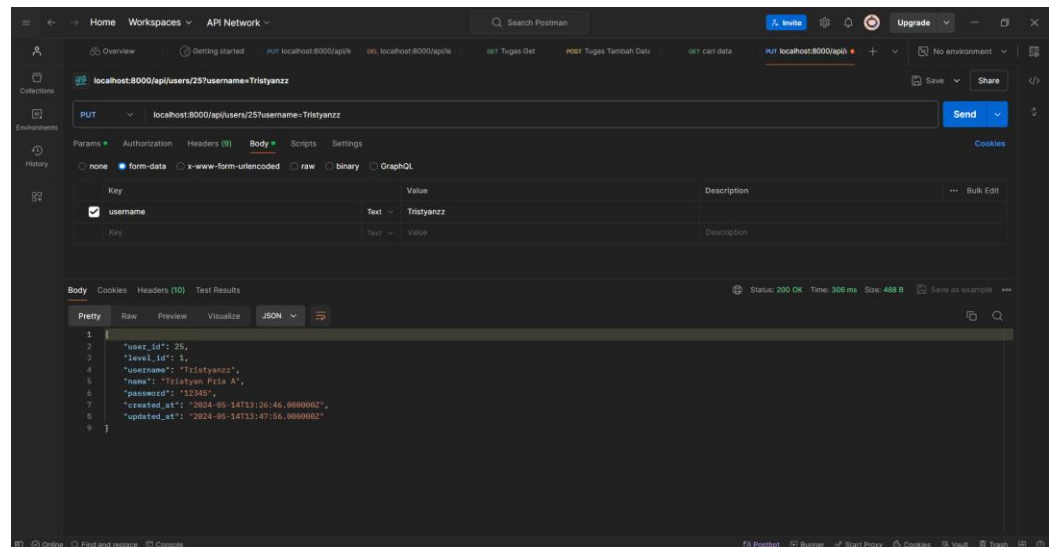
No	Screenshoot Tugas
1	<div><h3>Data User</h3><h4>Get Sema Data</h4><h4>Tambah Data</h4></div>



## Cari Data

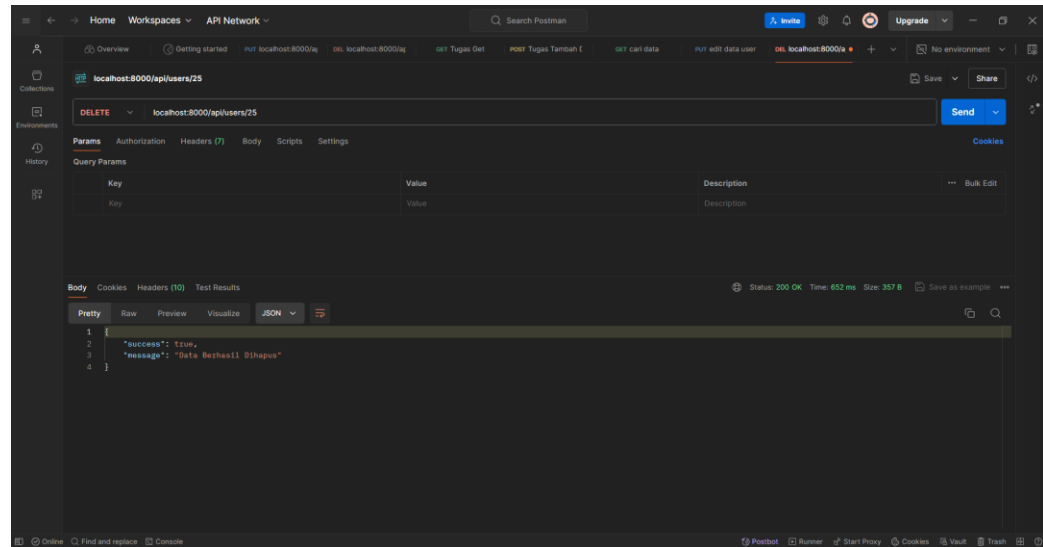


## Edit Data





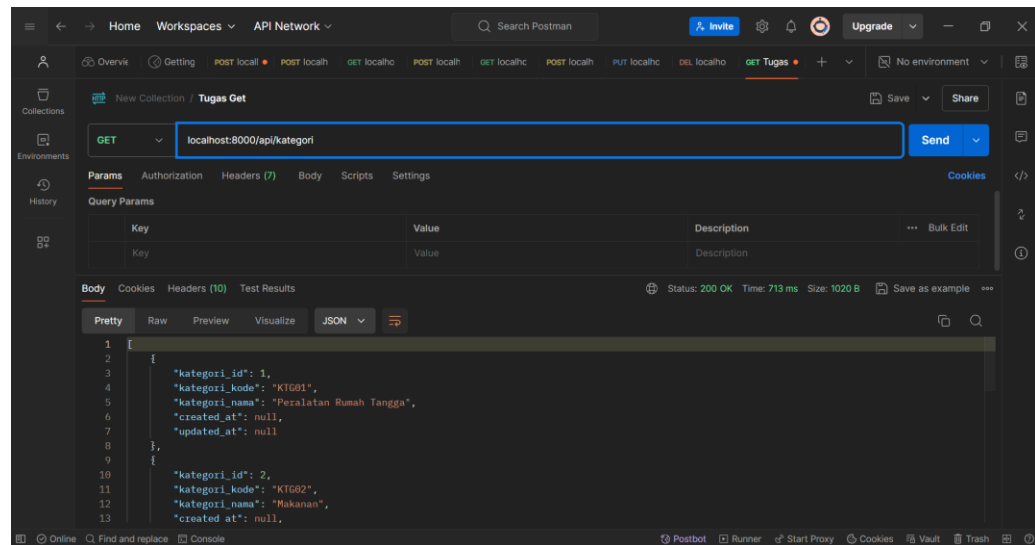
## Hapus Data



2

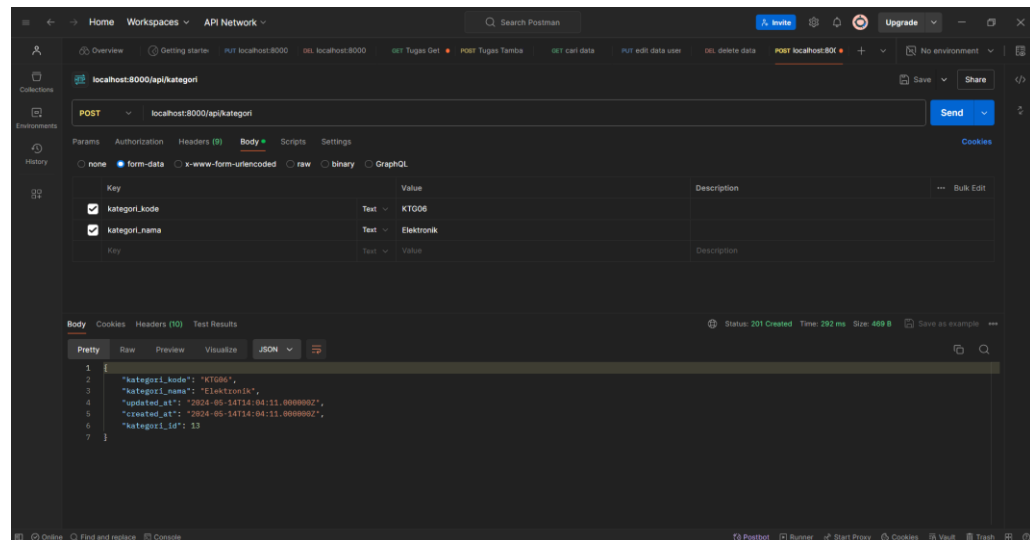
## Kategori Barang

### Get Semua Data

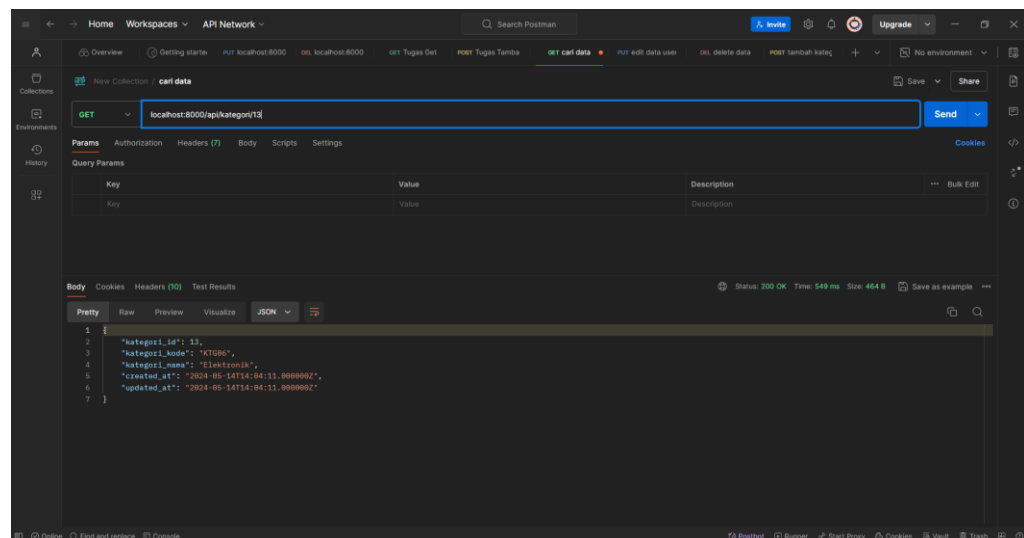




## Tambah Data



## Cari Data





## Edit Data

The screenshot shows the Postman interface for a PUT request. The URL is `localhost:8000/api/kategori/13` with the path parameter `kategori_nama=Alat Masak`. The request body is a JSON object:

```
1 {
2   "kategori_id": 13,
3   "kategori_kode": "KT006",
4   "kategori_nama": "Alat Masak",
5   "created_at": "2024-05-14T14:04:11.000000Z",
6   "updated_at": "2024-05-14T14:09:28.000000Z"
7 }
```

The response status is 200 OK, with a time of 336 ms and a size of 464 B.

## Hapus Data

The screenshot shows the Postman interface for a DELETE request. The URL is `localhost:8000/api/kategori/13`. The response status is 200 OK, with a time of 227 ms and a size of 357 B. The response body is a JSON object:

```
1 {
2   "success": true,
3   "message": "Data Berhasil Dihapus"
4 }
```

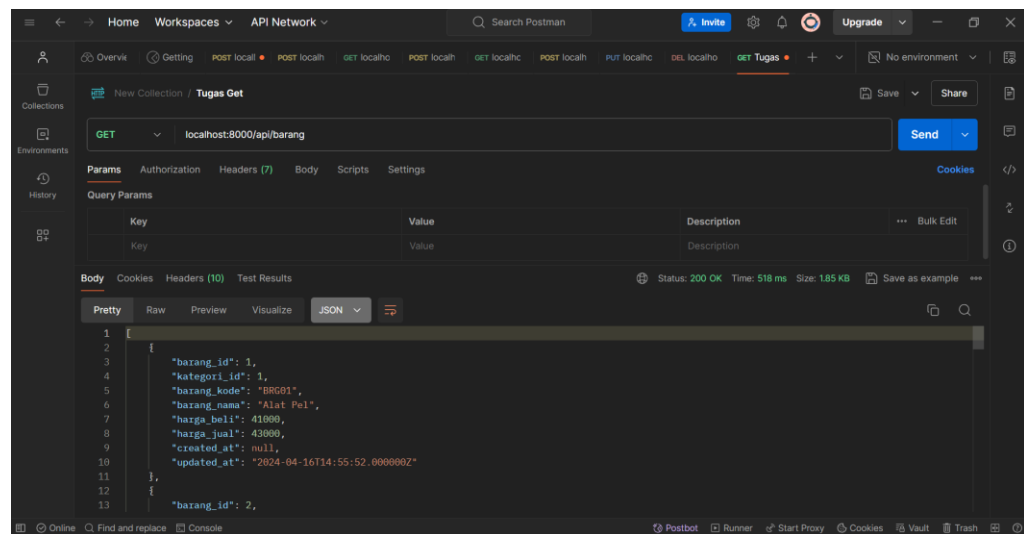




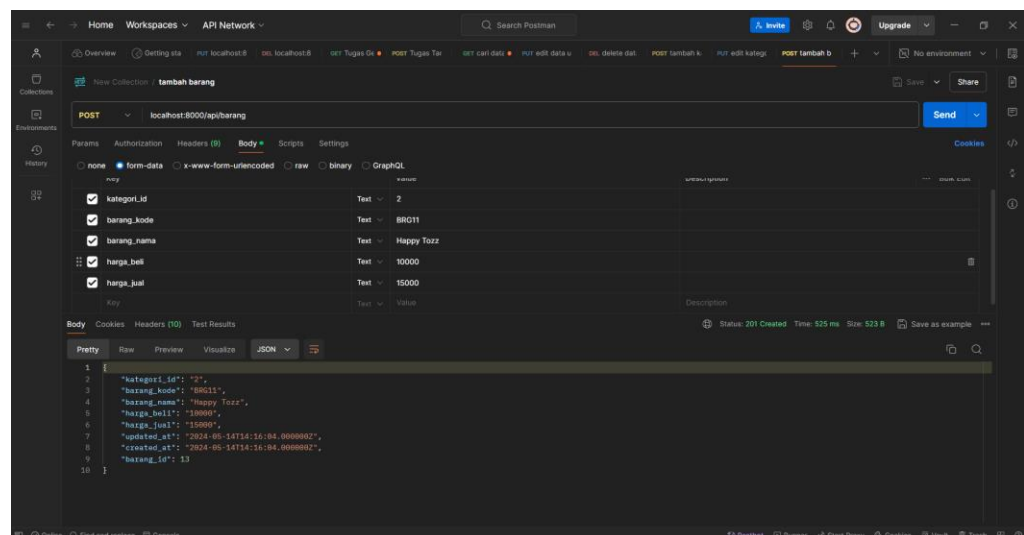
3

## Data Barang

### Get Semua Data

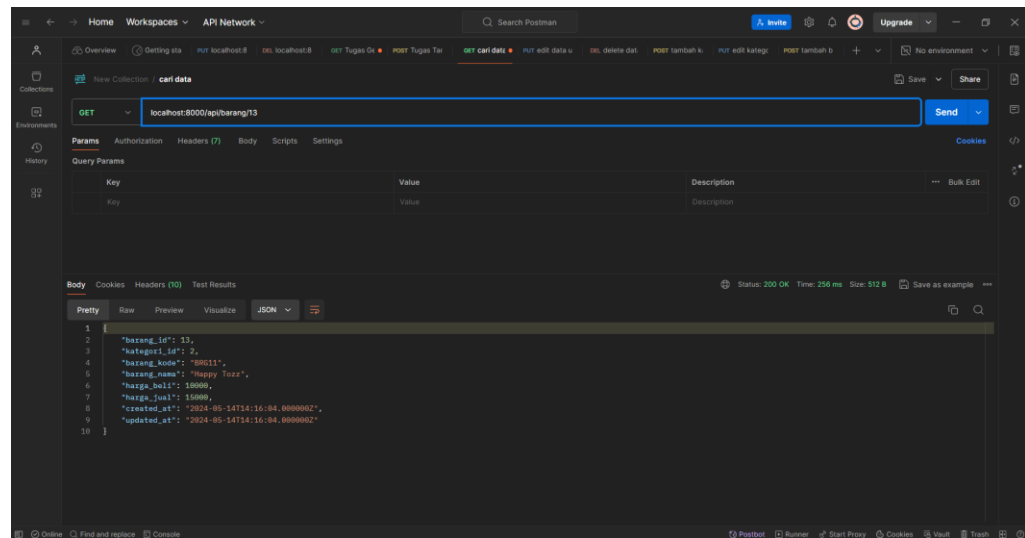


### Tambah Data

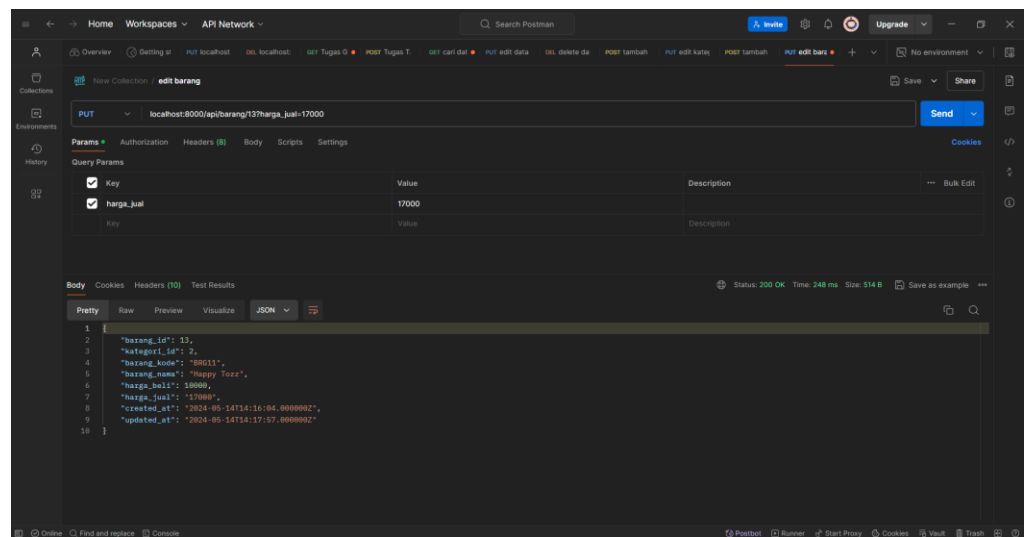




## Cari Data

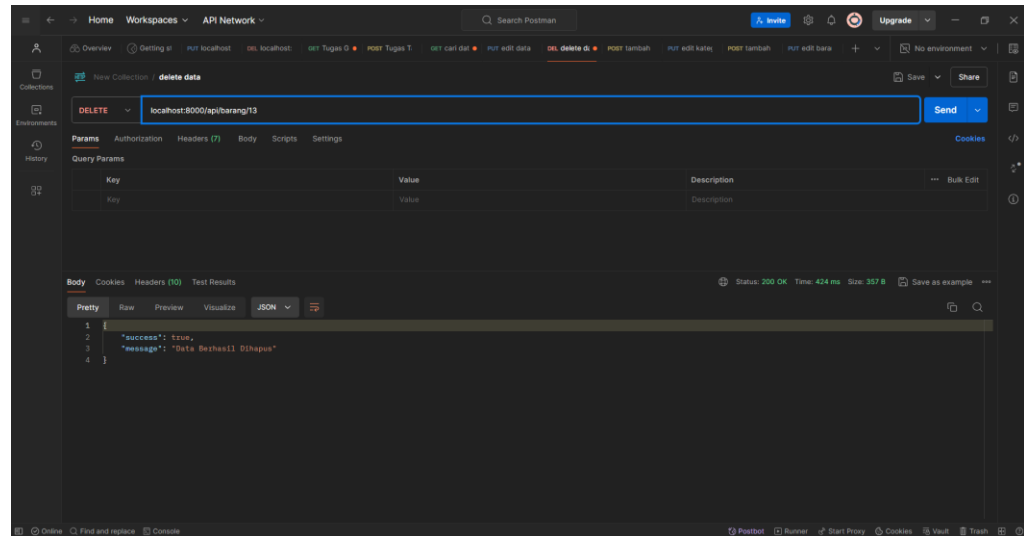


## Edit Data





## Hapus Data



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