

**PEMROGRAMAN WEB LANJUT**  
**JOBSHEET 10 - RESTFUL API**



**Disusun oleh:**  
Stefanus Ageng Budi Utomo (2241720126)

**JURUSAN TEKNOLOGI INFORMASI**  
**POLITEKNIK NEGERI MALANG**  
**TAHUN AJARAN 2023/2024**

## Praktikum 1 - Membuat Restful API Register

1. Sebelum memulai membuat REST API, terlebih dahulu download aplikasi Postman di

<https://www.postman.com/downloads>.

Aplikasi ini akan digunakan untuk mengerjakan semua tahap praktikum pada Jobsheet ini.

2. Lakukan instalasi JWT dengan mengetikkan perintah berikut:

```
composer require tymon/jwt-auth:2.1.1
```

Pastikan Anda terkoneksi dengan internet.

```
PS D:\Abi\kuliah\semester 4\PBL\Code\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 (2.2.0)
- Locking lcobucci/jwt (4.0.4)
- Locking stella-maris/clock (0.1.7)
- Locking tymon/jwt-auth (2.1.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 4 installs, 0 updates, 0 removals
- Downloading stella-maris/clock (0.1.7)
- Downloading lcobucci/clock (2.2.0)
- Downloading lcobucci/jwt (4.0.4)
- Downloading tymon/jwt-auth (2.1.1)
- Installing stella-maris/clock (0.1.7): Extracting archive
- Installing lcobucci/clock (2.2.0): Extracting archive
- Installing lcobucci/jwt (4.0.4): Extracting archive
```

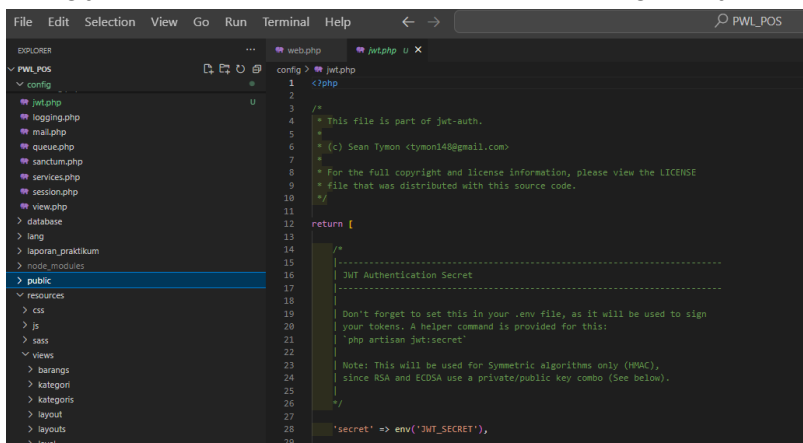
3. Setelah berhasil menginstall JWT, lanjutkan dengan publish konfigurasi file dengan perintah berikut:

```
php artisan vendor:publish
```

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

```
PS D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS> php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
INFO Publishing assets.
Copying file [D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS\vendor\tymon\jwt-auth\config\config.php] to [D:\Abi\kuliah\semester 4\PBL\Code\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.



```
File Edit Selection View Go Run Terminal Help
PWL_POS
EXPLORER
PWL_POS
  config
    jwt.php
  logging.php
  mail.php
  queue.php
  sanctum.php
  services.php
  session.php
  view.php
  database
  lang
  laporan_praktikum
  node_modules
  public
  resources
  css
  js
  sass
  views
  barangs
  kategori
  kategori
  layout
  layouts
  level
  web.php
  jwt.php
config > jwt.php
1 <?php
2
3 /*
4  * This file is part of jwt-auth.
5  *
6  * (c) Sean Tymon <tymon148@gmail.com>
7  *
8  * For the full copyright and license information, please view the LICENSE
9  * file that was distributed with this source code.
10 */
11
12 return [
13
14     /*
15      * JWT Authentication Secret
16      *
17      * Don't forget to set this in your .env file, as it will be used to sign
18      * your tokens. A helper command is provided for this:
19      *
20      * 'php artisan jwt:secret'
21      *
22      * Note: This will be used for Symmetric algorithms only (HMAC),
23      * since RSA and ECDSA use a private/public key combo (see below).
24      */
25     'secret' => env('JWT_SECRET'),
26 ]
```

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

```

58 VITE_PUSHER_SCHEME="${PUSHER_SCHEME}"
59 VITE_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
60
61 JWT_SECRET=VOCE3Aw5Vte6gZEDc1kx9XTuu30ktC9gds879kXz3o9RVxy1f3m333NIAhId
62
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB
PS D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS> php artisan vendor:publish --provider="Tyron\JwtAuth\Providers\LaravelServiceProvider"
[INFO] Publishing assets.
Copying file [D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS\vendor\tyron\jwt-auth\config\config.php] to [D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS\config\jwt.php] DONE
PS D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS> php artisan jwt:secret
Jwt-auth secret [VOCE3Aw5Vte6gZEDc1kx9XTuu30ktC9gds879kXz3o9RVxy1f3m333NIAhId] set successfully.
PS D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS>

```

- 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'
    ]
],

```

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

```

class UserModel extends Authenticatable
{
    use HasFactory;

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

    public function getJWTCustomClaims()
    {
        return [];
    }
}

```

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

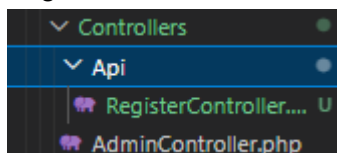
`php artisan make:controller Api/RegisterController`

```

PS D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS> php artisan make:controller Api/RegisterController
[INFO] Controller [D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS\app\Http\Controllers\Api\RegisterController.php] created successfully.

```

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



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

```

class RegisterController extends Controller
{
    //
    public function __invoke(Request $request)
    {
        // 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);
    }
}

```

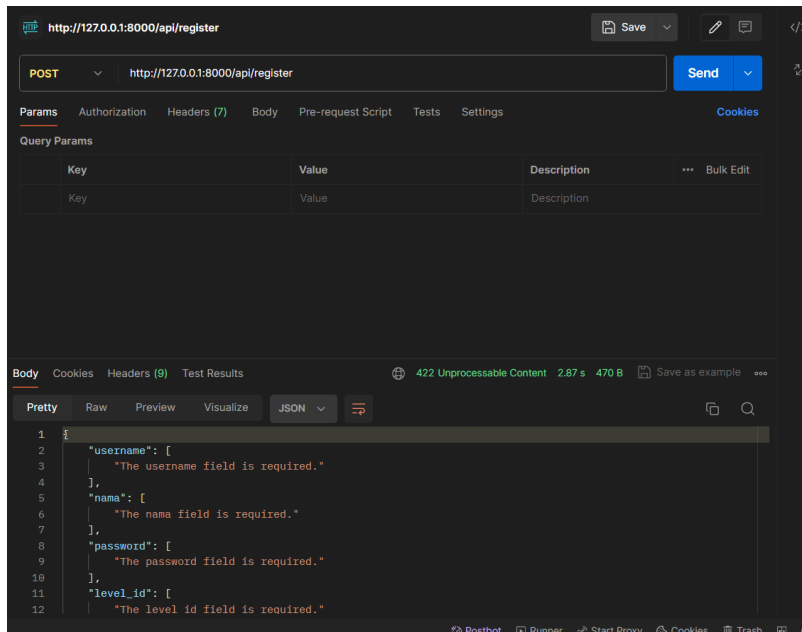
10. Selanjutnya buka routes/api.php, ubah semua kode menjadi seperti berikut

```

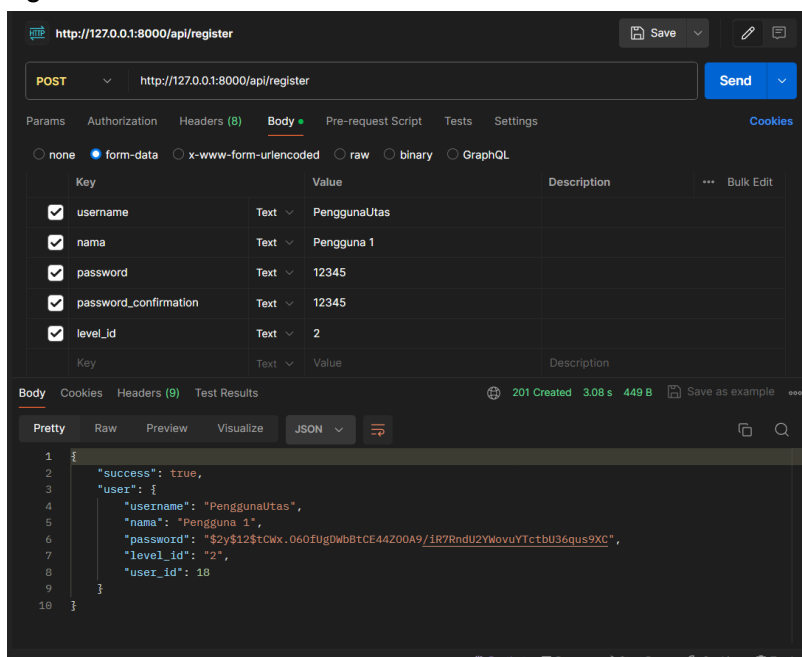
routes > api.php > ...
1  <?php
2
3  use Illuminate\Http\Request;
4  use Illuminate\Support\Facades\Route;
5
6  /*
7  |-----
8  | API Routes
9  |-----
10 |
11 | Here is where you can register API routes for your application. These
12 | routes are loaded by the RouteServiceProvider and all of them will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

```

11. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL\_POS/public/api/register serta method POST. Klik Send.



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.



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](#)

```
D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS>php artisan make:controller Api/LoginController

INFO Controller [D:\Abi\kuliah\semester 4\PBL\Code\PWL_POS\app\Http\Controllers\Api\LoginController.php] created successfully.
```

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

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

```
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');

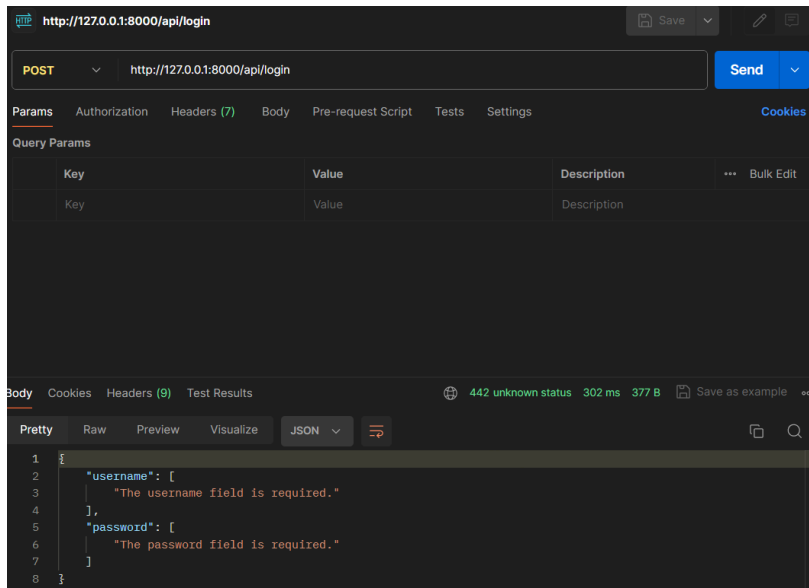
        // if auth failed
        if (!$token = auth()->guard('api')->attempt($credentials))
            return response()->json([
                'success' => false,
                'message' => 'Username atau password anda salah',
            ], 401);

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

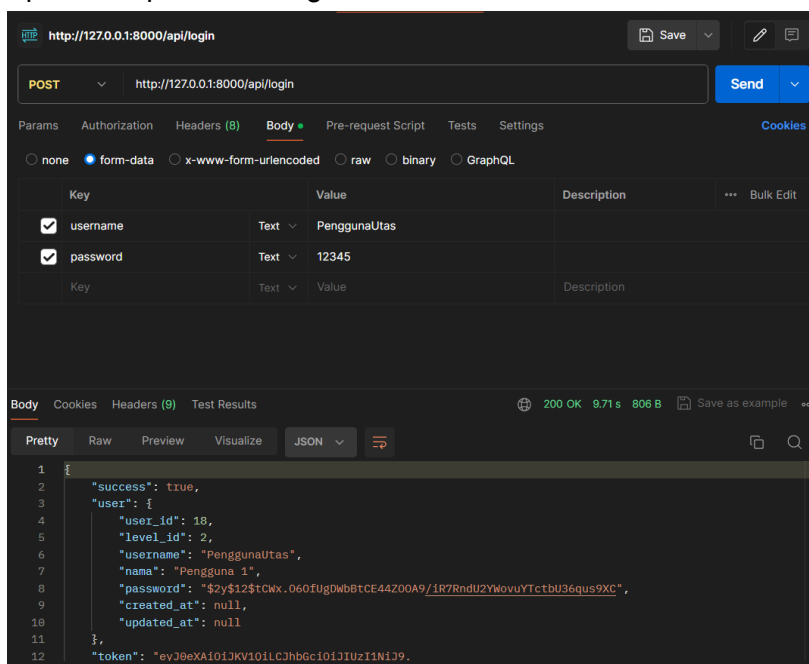
3. Berikutnya tambahkan route baru pada file api.php yaitu /login dan /user.

```
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();
});
```

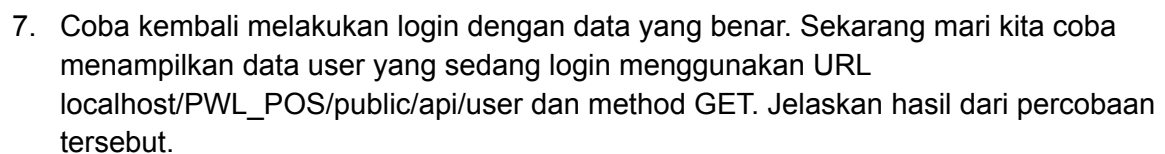
4. Jika sudah, kita akan melakukan uji coba REST API melalui aplikasi Postman. Buka aplikasi Postman, isi URL localhost/PWL\_POS/public/api/login serta method POST. Klik Send.



- 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 untuk data yang salah dan berikan screenshot hasil percobaan Anda.



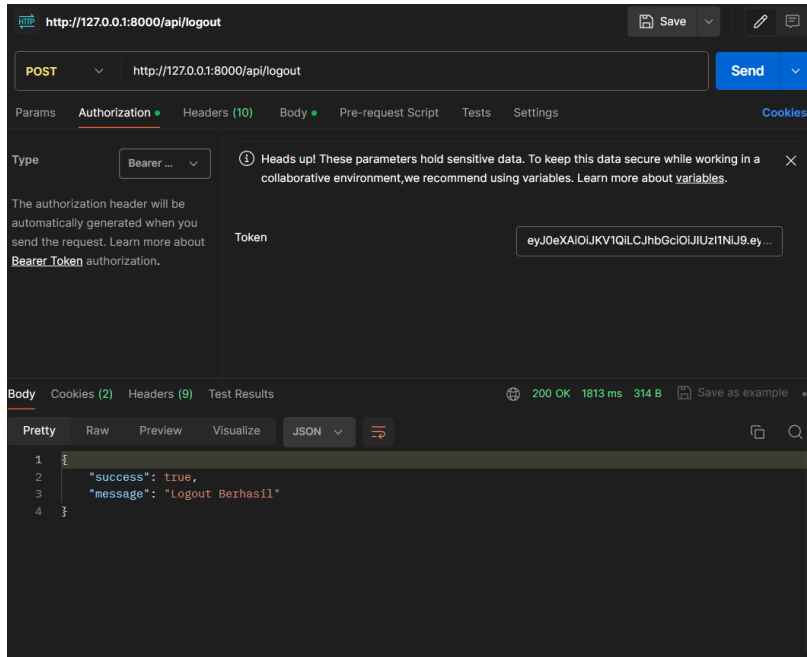
1. Tambahkan kode berikut pada file.env  
`JWT_SHOW_BLACKLIST_EXCEPTION=true`
2. Buat Controller baru dengan nama LogoutController.  
`php artisan make:controller Api/LogoutController`



3. Buka file tersebut dengan ubah kode menjadi seperti berikut.
4. Lalu kita tambahkan routes pada api.php

```
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



7. Lakukan commit perubahan file pada Github.

## Praktikum 4 - Implementasi CRUD dalam RESTful API

Pada praktikum ini kita akan menggunakan tabel m\_level untuk dimodifikasi menggunakan RESTful API.

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

```

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',
        ]);
    }
}

```

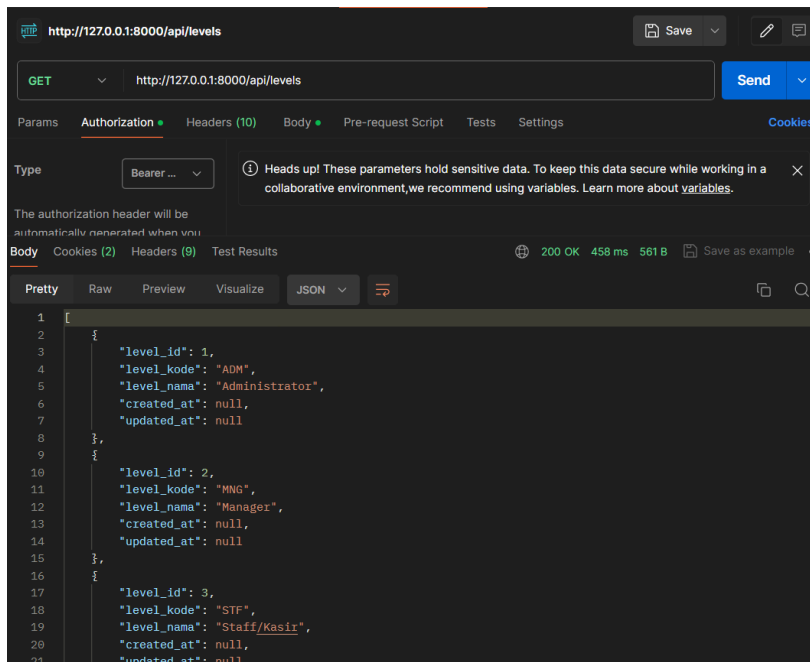
3. Kemudian kita lengkapi routes pada api.php.

```

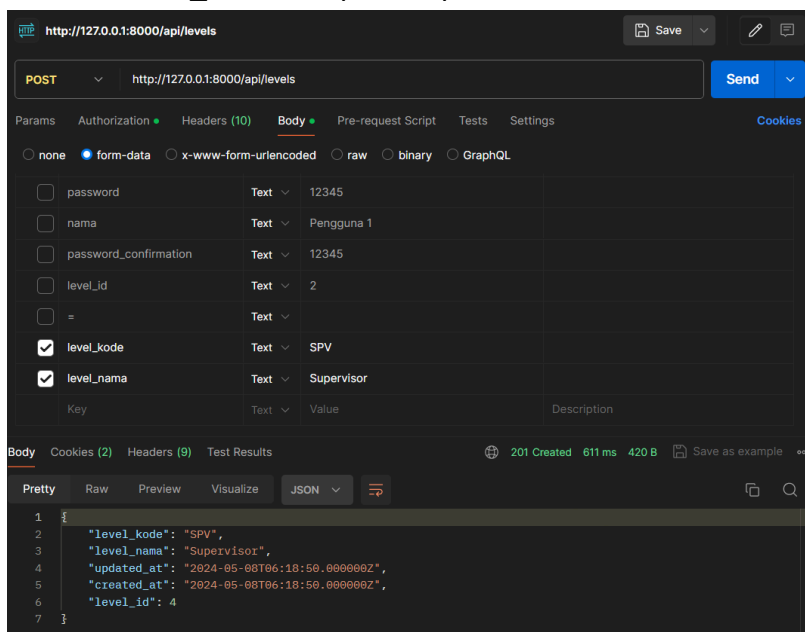
Route::get('levels', [LevelController::class, 'index']);
Route::post('levels', [LevelController::class, 'store']);
Route::get('levels/{level}', [LevelController::class, 'show']);
Route::put('levels/{level}', [LevelController::class, 'update']);
Route::delete('levels/{level}', [LevelController::class,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

```

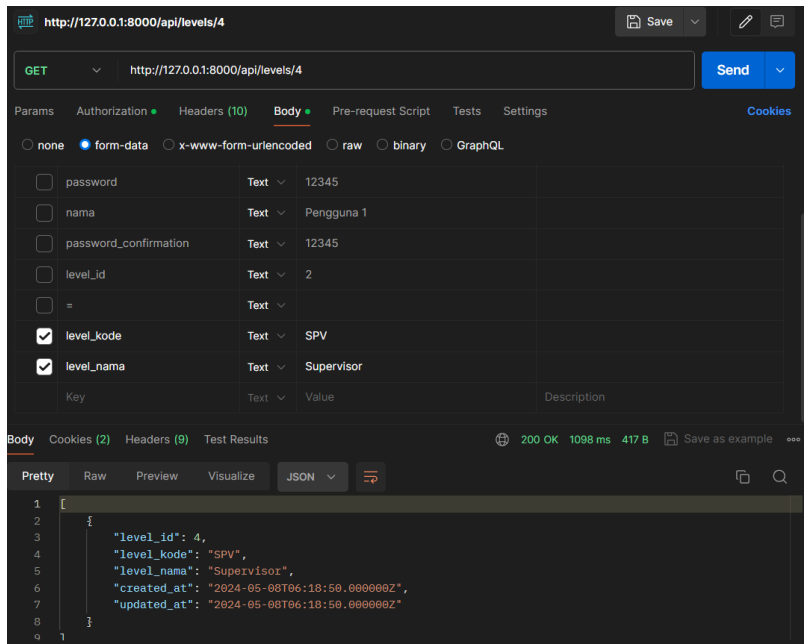
4. Jika sudah. Lakukan uji coba API mulai dari fungsi untuk menampilkan data. Gunakan URL: localhost/PWL\_POS-main/public/api/levels dan method GET.



5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL\_POSmain/public/api/levels dan method POST seperti di bawah ini.

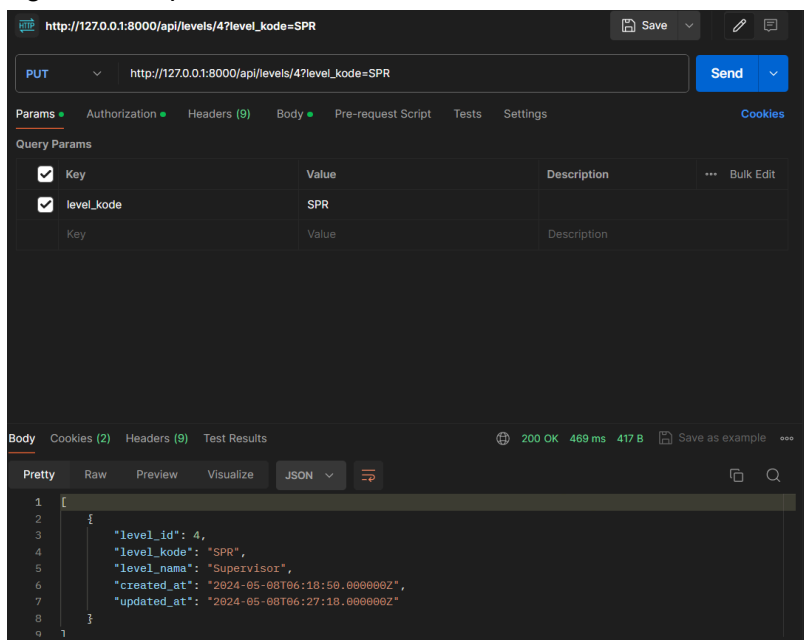


6. Berikutnya lakukan percobaan menampilkan detail data.



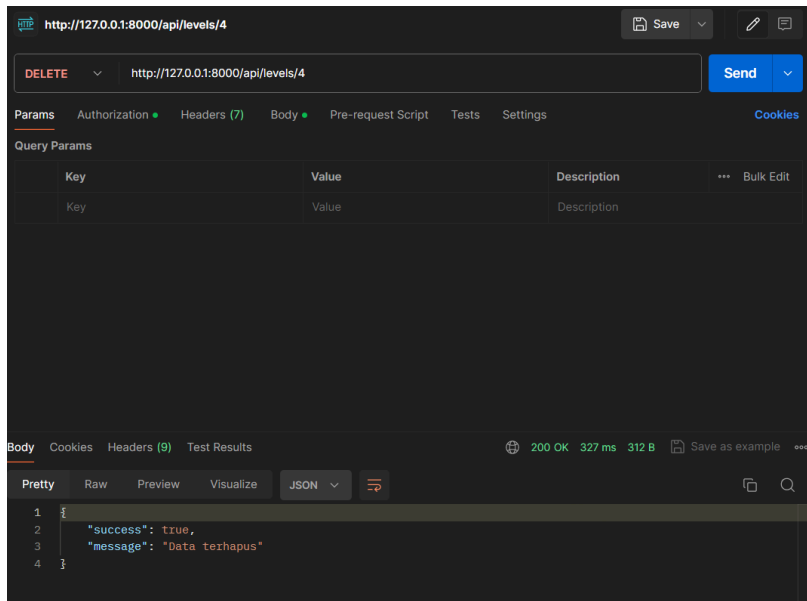
- untuk mengakses detailnya setelah levels kita perlu menambahkan level\_id pada url nya `levels/{level_id}` sesuai dengan yang ingin ditampilkan. seperti pada contoh diatas, ingin menampilkan data yang baru saja ditambahkan.

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



- Method put digunakan untuk mengupdate data sesuai dengan apa yang kita inginkan. seperti contoh di atas merubah level\_kode dari SPV menjadi SPR dengan menambahkan "level\_kode=SPR" pada parameter url.

8. Terakhir lakukan percobaan hapus data.



- Untuk menghapus data, dapat menggunakan method delete dengan mengirimkan level\_kode sebagai parameter data yang ingin dihapus.
9. Lakukan commit perubahan file pada Github.

# Tugas

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

```
// user
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']);

// kategori
Route::get('kategoris', [KategoriController::class, 'index']);
Route::post('kategoris', [KategoriController::class, 'store']);
Route::get('kategoris/{id}', [KategoriController::class, 'show']);
Route::put('kategoris/{id}', [KategoriController::class, 'update']);
Route::delete('kategoris/{id}', [KategoriController::class, 'destroy']);

// barang
Route::get('barangs', [BarangController::class, 'index']);
Route::post('barangs', [BarangController::class, 'store']);
Route::get('barangs/{id}', [BarangController::class, 'show']);
Route::put('barangs/{id}', [BarangController::class, 'update']);
Route::delete('barangs/{id}', [BarangController::class, 'destroy']);
```

## UserController

```
class UserController extends Controller
{
    //
    public function index()
    {
        return UserModel::all();
    }

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

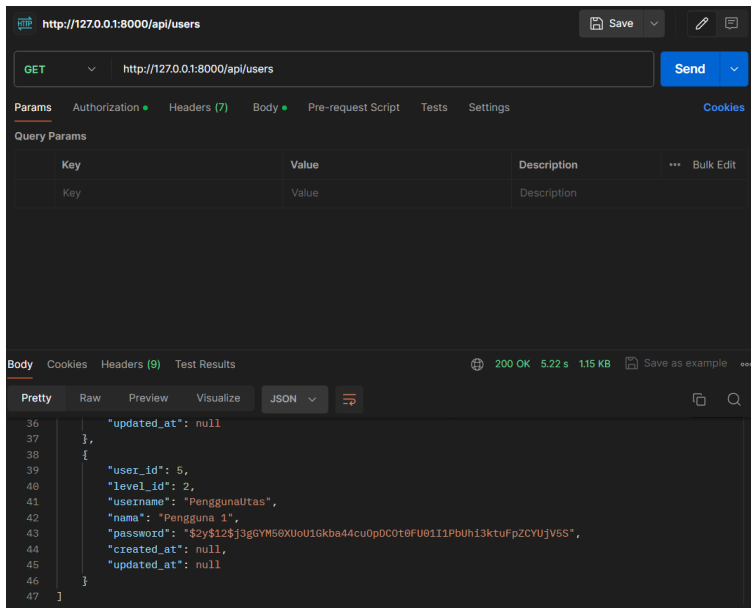
    public function show($id)
    {
        return UserModel::find($id);
    }

    public function update(Request $request, $id)
    {
        $user = UserModel::find($id);
        $user->update($request->all());
        return UserModel::find($id);
    }

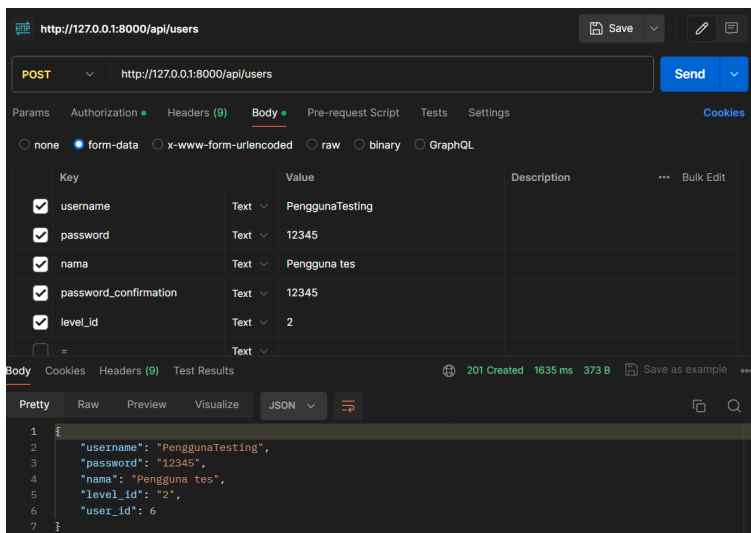
    public function destroy($id)
    {
        $user = UserModel::find($id);
        $user->delete();

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

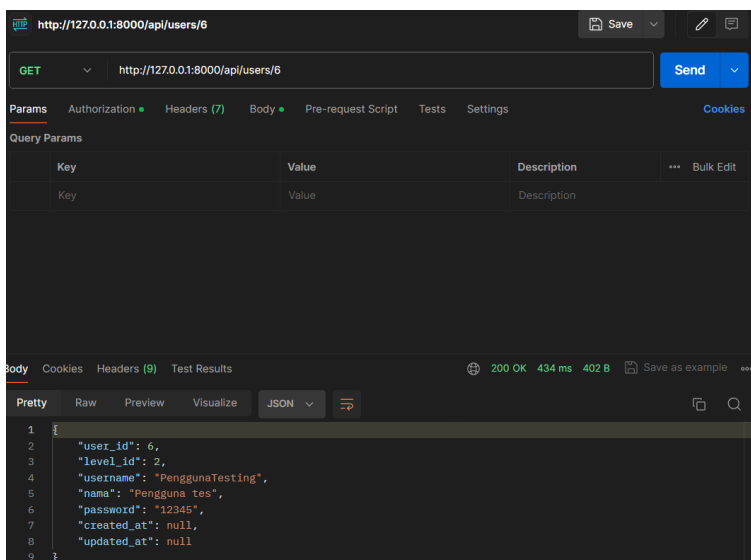
index user



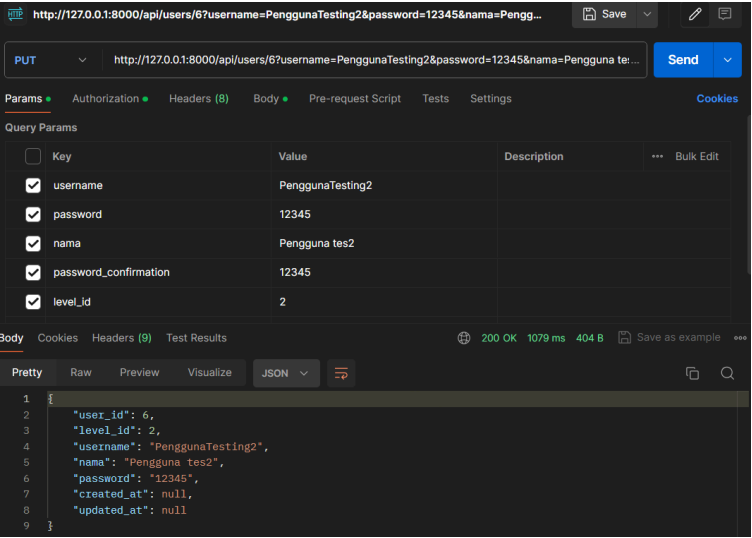
store user



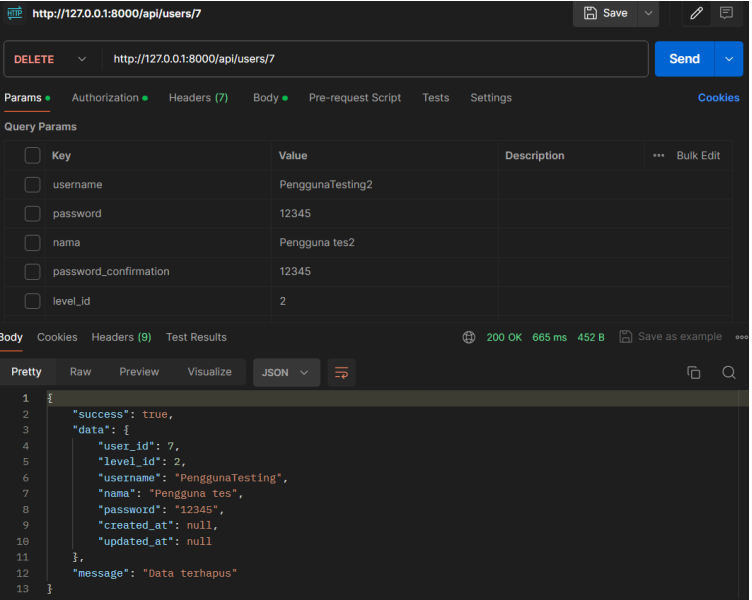
show user



update user



# delete user





## Kategori Controller

```
class KategoriController extends Controller
{
    //
    public function index()
    {
        return KategoriModel::all();
    }

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

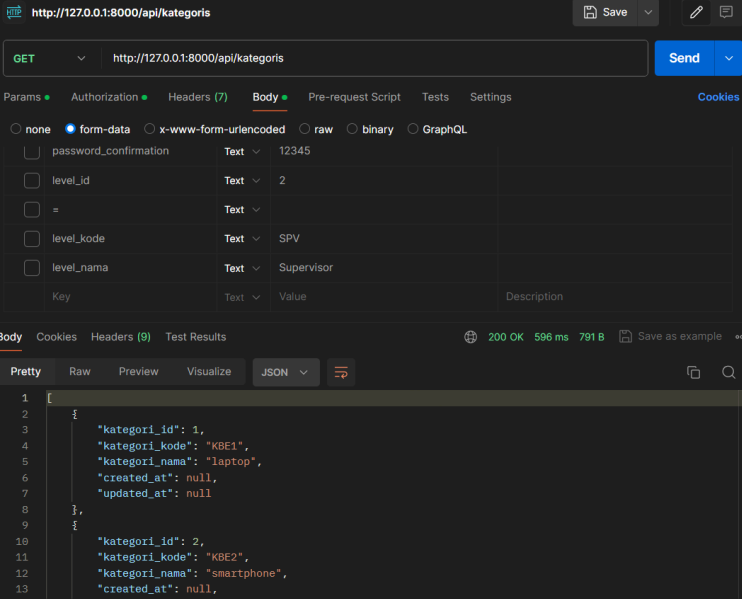
    public function show($id)
    {
        return KategoriModel::find($id);
    }

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

    public function destroy($id)
    {
        $ktg = KategoriModel::find($id);
        $ktg->delete();

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

## index kategori

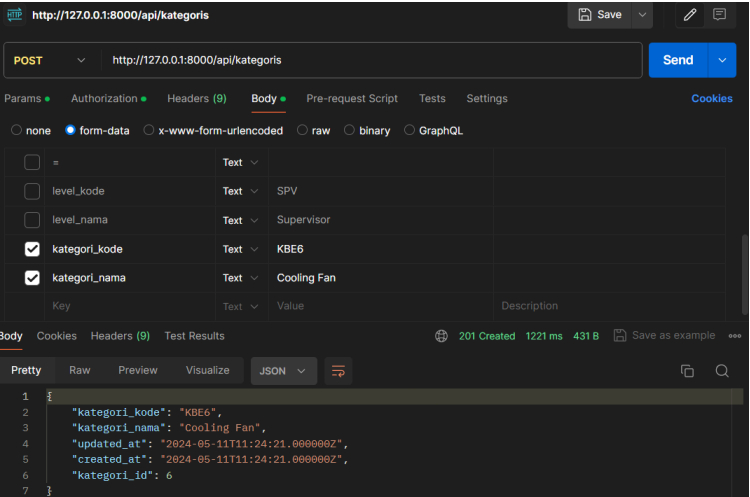


The screenshot shows a REST client interface with the following details:

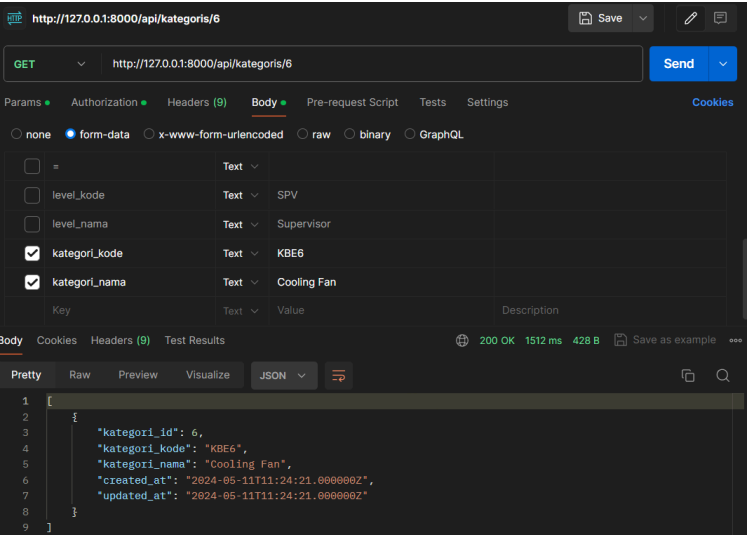
- URL:** `http://127.0.0.1:8000/api/kategoris`
- Method:** `GET`
- Body:** The response is a JSON array of two category objects:

```
1 {
2   {
3     "kategori_id": 1,
4     "kategori_kode": "KBE1",
5     "kategori_nama": "laptop",
6     "created_at": null,
7     "updated_at": null
8   },
9   {
10    "kategori_id": 2,
11    "kategori_kode": "KBE2",
12    "kategori_nama": "smartphone",
13    "created_at": null,
```
- Status:** `200 OK`, `596 ms`, `791 B`

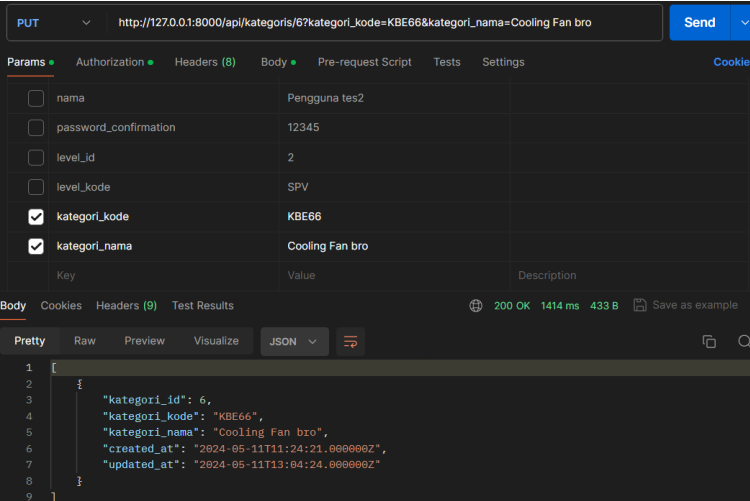
## store kategori



show kategori



update kategori



delete kategori

http://127.0.0.1:8000/api/kategori/6

Save

Send

DELETE

http://127.0.0.1:8000/api/kategori/6

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

☐

nama

Pengguna tes2

☐

password\_confirmation

12345

☐

level\_id

2

☐

level\_kode

SPV

☐

kategori\_kode

KBE66

☐

kategori\_nama

Cooling Fan bro

Key

Value

Description

Body

Cookies

Headers (9)

Test Results

200 OK

1303 ms

481 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "success": true,
3   "data": {
4     "kategori_id": 6,
5     "kategori_kode": "KBE66",
6     "kategori_nama": "Cooling Fan bro",
7     "created_at": "2024-05-11T11:24:21.000000Z",
8     "updated_at": "2024-05-11T13:04:24.000000Z"
9   },
10  "message": "Data terhapus"
11 }
```

## BarangController

```
class BarangController extends Controller
{
    //

    public function index()
    {
        return BarangModel::all();
    }

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

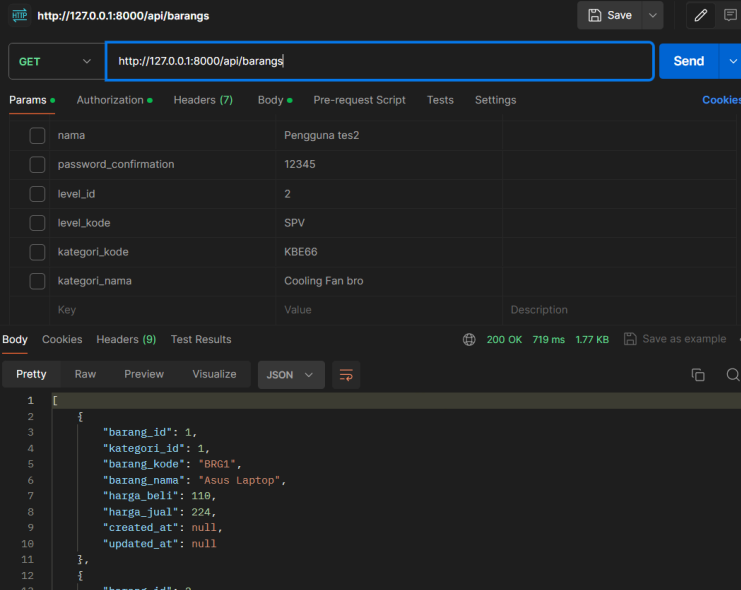
    public function show($id)
    {
        return BarangModel::find($id);
    }

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

    public function destroy($id)
    {
        $brg = BarangModel::find($id);
        $brg->delete();

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

## index barang



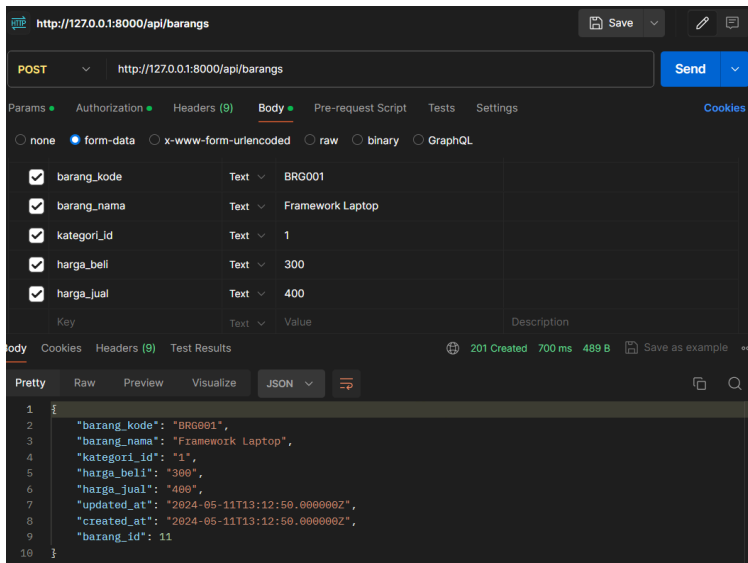
The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8000/api/barangs`
- Method:** `GET`
- Params:** A table with the following data:

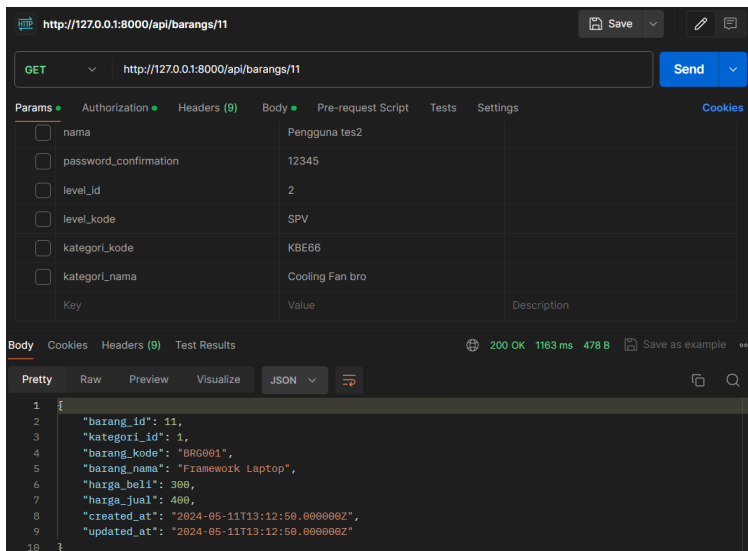
Key	Value
nama	Pengguna tes2
password_confirmation	12345
level_id	2
level_kode	SPV
kategori_kode	KBE66
kategori_nama	Cooling Fan bro
- Body:** The response is a JSON array of two items:

```
[{"barang_id": 1, "kategori_id": 1, "barang_kode": "BRG1", "barang_nama": "Asus Laptop", "harga_beli": 110, "harga_jual": 224, "created_at": null, "updated_at": null}, {"barang_id": 2, ...}]
```

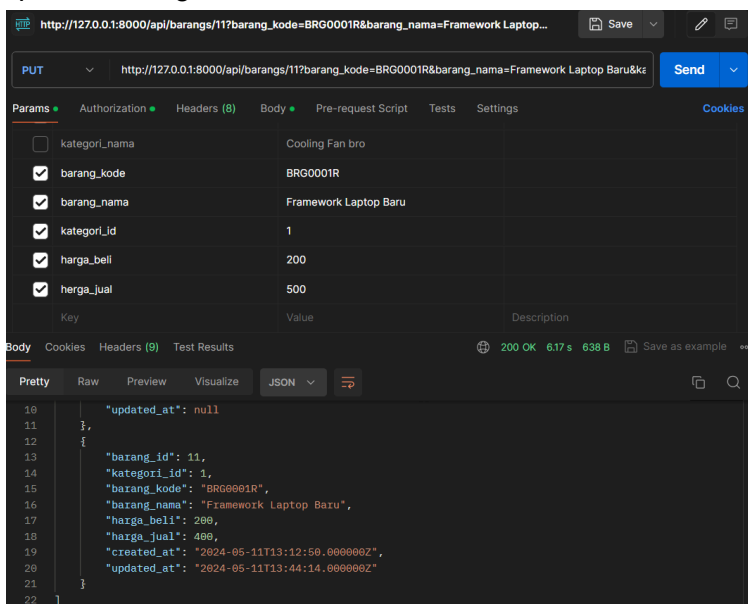
## store barang



show barang



update barang



delete barang

http://127.0.0.1:8000/api/barangs/11

DELETE

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

<input type="checkbox"/>	kategori_nama	Cooling Fan bro	
<input type="checkbox"/>	barang_kode	BRG0001R	
<input type="checkbox"/>	barang_nama	Framework Laptop new	
<input type="checkbox"/>	kategori_id	1	
<input type="checkbox"/>	harga_beli	200	
<input type="checkbox"/>	harga_jual	500	
	Key	Value	Description

body

Cookies

Headers (9)

Test Results

200 OK

1007 ms

535 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "success": true,
3   "data": {
4     "barang_id": 11,
5     "kategori_id": 1,
6     "barang_kode": "BRG0001R",
7     "barang_nama": "Framework Laptop Baru",
8     "harga_beli": 200,
9     "harga_jual": 400,
10    "created_at": "2024-05-11T13:12:50.000000Z",
11    "updated_at": "2024-05-11T13:44:14.000000Z"
12  },
13  "message": "Data terhapus"
```