

**LAPORAN PRAKTIKUM  
PEMROGRAMAN WEB LANJUT**

**JOBSCHEET - 10 : RESTFUL API**



**Disusun Oleh :**  
**Ghoffar Abdul Ja'far**  
**2341720035/TI2F**

**JURUSAN TEKNOLOGI INFORMASI  
POLITEKNIK NEGERI MALANG  
2024/2025**



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 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 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 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.

6. Selanjutnya lakukan konfigurasi guard API. Buka config/auth.php. Ubah bagian ‘guards’ menjadi seperti berikut.

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

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



```
use Illuminate\Container\Attributes\Auth;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Tymon\JWTAuth\Contracts\JWTSubject;

Windsurf: Refactor | Explain
class UserModel extends Authenticatable implements JWTSubject
{
    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function getJWTIdentifier()
    {
        return $this->getKey();
    }
    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function getJWTCustomClaims(): array
    {
        return [];
    }
}
```

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

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

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

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

```
namespace App\Http\Controllers\Api;

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

Windsurf: Refactor | Explain
class RegisterController extends Controller
{
    Windsurf: Refactor | Explain | Generate Function Comment | X
    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 validation fails
if ($validator->fails()) {
    return response()->json([
        $validator->errors(),
    ], 422);
}
// create user
$user = UserModel::create([
    'username' => $request->username,
    'nama' => $request->nama,
    'password' => bcrypt($request->password),
    'level_id' => $request->level_id,
]);
// return response JSON user is created
if ($user) {
    return response()->json([
        'success' => true,
        'user' => $user,
    ], 201);
}

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



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

```
<?php

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

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

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

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

The screenshot shows the Postman interface with a failed API call. The URL is `http://127.0.0.1:8000/api/register`. The response code is `422 Unprocessable Content`. The response body is a JSON object with validation errors:

```
1 [2 {3   "username": [4     "The username field is required."5   ],6   "nama": [7     "The nama field is required."8   ],9   "password": [10     "The password field is required."11   ],12   "level_id": [13     "The level_id field is required."14   ]15 }16 ]
```

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.

The screenshot shows the Postman interface with a successful API call. The URL is `http://127.0.0.1:8000/api/register`. The request method is POST. The body is set to form-data, containing the following fields:

Key	Type	Description
username	Text	pengunasatu
nama	Text	Pengguna 1
password	Text	12345
password_confirmation	Text	12345
level_id	Text	2

The response status is 201 Created. The JSON response is:

```
1 {  
2     "success": true,  
3     "user": {  
4         "username": "pengunasatu",  
5         "nama": "Pengguna 1",  
6         "level_id": "2",  
7         "updated_at": "2025-05-31T08:19:13.000000Z",  
8         "created_at": "2025-05-31T08:19:13.000000Z",  
9         "user_id": 5  
10    }  
11 }
```

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

Lakukan percobaan yang sama dan berikan screenshot 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.

```
namespace App\Http\Controllers\Api;

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

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

        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);
    }
}
```



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

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

/*
|--------------------------------------------------------------------------
| 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', RegisterController::class)->name('register');
Route::post('/login', 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.

The screenshot shows a Postman interface with the following details:

- Method: POST
- URL: http://127.0.0.1:8000/api/login
- Headers (8): Content-Type: application/json
- Body (raw): { "username": "", "password": "" }
- Response status: 422 Unprocessable Content
- Response body:

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

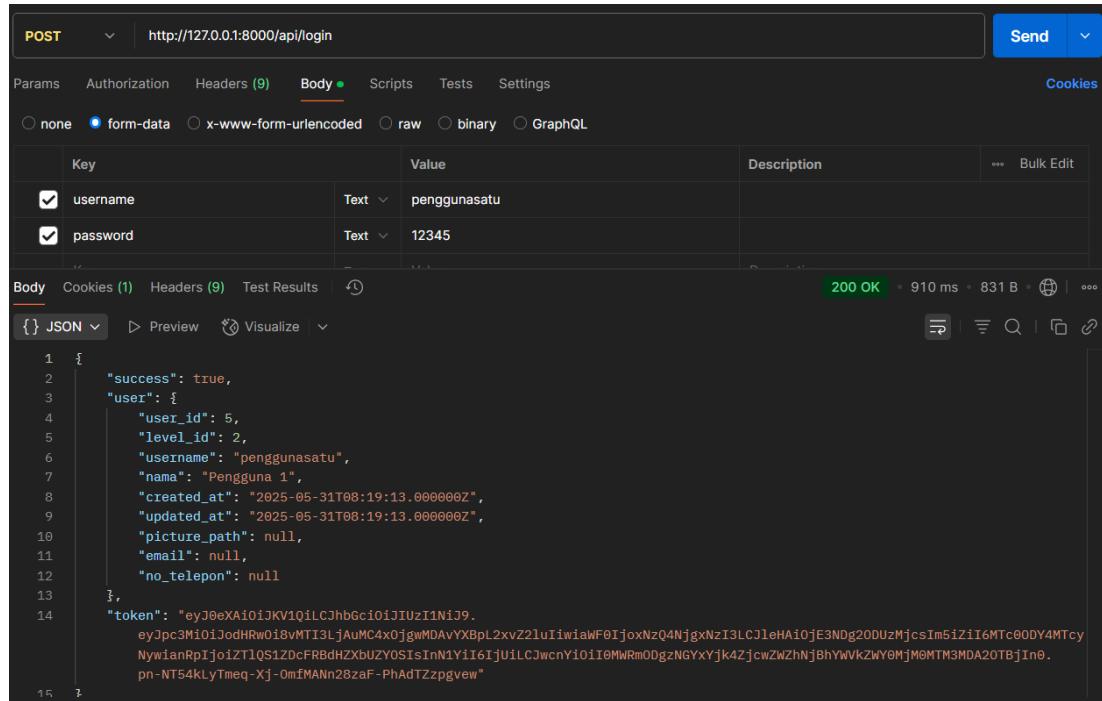
Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshot hasil percobaan Anda.

5. Selanjutnya, isikan username dan password sesuai dengan data user yang ada pada



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

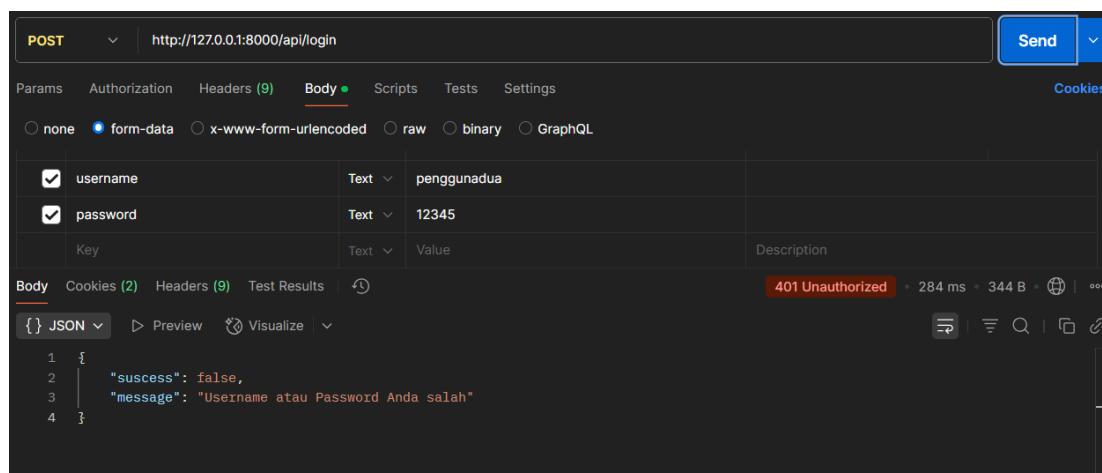


The screenshot shows a POST request to `http://127.0.0.1:8000/api/login`. The 'Body' tab is selected with 'form-data' chosen. Two fields are present: 'username' (text value: penggunasatu) and 'password' (text value: 12345). The response status is 200 OK, with a response time of 910 ms and a body size of 831 B. The JSON response is:

```
1 {  
2     "success": true,  
3     "user": {  
4         "user_id": 5,  
5         "level_id": 2,  
6         "username": "pengunasatu",  
7         "nama": "Pengguna 1",  
8         "created_at": "2025-05-31T08:19:13.000000Z",  
9         "updated_at": "2025-05-31T08:19:13.000000Z",  
10        "picture_path": null,  
11        "email": null,  
12        "no_telepon": null  
13    },  
14    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
eyJpc3MiOiJodHRwOi8vMTI3LjAuMC4xOjgwMDAvYXBpL2xvZ2luIiwiaWF0IjoxNjgxNzI3LCJleHAIoJE3NDg2ODUzMjcsIm5iZi6MTc0ODY4MTcy  
NyrianRpIjoIzTlQ51ZdcFRBdHZxbUZOStsInN1YiI6IjUiLCJwcnyIOiIOMWRmODgzNGYxYjk4ZjcwZWZhNjBhYwVkJM0MTM3MDA20TBjIn0.  
pn-NT54kLyTmeq-Xj-0mfMANn28zaF-PhAdTZzpgvew"
```

Lakukan percobaan yang sama dan berikan screenshot hasil percobaan Anda.

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



The screenshot shows a POST request to `http://127.0.0.1:8000/api/login`. The 'Body' tab is selected with 'form-data' chosen. Two fields are present: 'username' (text value: penggunadua) and 'password' (text value: 12345). The response status is 401 Unauthorized, with a response time of 284 ms and a body size of 344 B. The JSON response is:

```
1 {  
2     "success": false,  
3     "message": "Username atau Password Anda salah"  
4 }
```

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.



The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: <http://127.0.0.1:8000/api/user>
- Authorization tab selected, showing "Bearer Token" selected.
- Headers tab (8):
  - Auth Type: Bearer Token
- Token field contains: zGA0YZZbftj5ZF3YF\_5t09lWGd5-kjbk
- Body tab:
  - JSON response:
  - Preview shows a JSON object with fields: user\_id, level\_id, username, nama, created\_at, updated\_at, picture\_path, email, and no\_telepon.
  - Raw code (JSON):
- Response details: 200 OK, 689 ms, 480 B

```
1 {  
2   "user_id": 5,  
3   "level_id": 2,  
4   "username": "pengguna satu",  
5   "nama": "Pengguna 1",  
6   "created_at": "2025-05-31T08:19:13.000000Z",  
7   "updated_at": "2025-05-31T08:19:13.000000Z",  
8   "picture_path": null,  
9   "email": null,  
10  "no_telepon": null  
11 }
```

8. Lakukan commit perubahan file pada Github.



### Praktikum 3 – Membuat RESTful API Logout

1. Tambahkan kode berikut pada file .env

```
JWT_SHOW_BLACKLIST_EXCEPTION=true
```

2. Buat Controller baru dengan nama LogoutController.

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

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

```
namespace App\Http\Controllers\Api;

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

Windsurf: Refactor | Explain
class LogoutController extends Controller
{
    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function __invoke(Request $request)
    {
        // remove token
        $removeToken = JWTAuth::invalidate(JWTAuth::getToken());
        if ($removeToken) {
            return response()->json([
                'status' => 'success',
                'message' => 'Logout Berhasil!'
            ]);
        }
    }
}
```

4. Lalu kita tambahkan routes pada api.php

```
Route::post('/logout', 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.



The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://127.0.0.1:8000/api/logout
- Authorization:** Bearer Token (selected)
- Token:** y6m4mcnUHwjzaGNvzEtBjbM\_1jc9U
- Body:** JSON (empty)
- Headers:** (9 items)
- Test Results:** 200 OK, 580 ms, 316 B
- JSON Response:**

```
1 {  
2     "success": true,  
3     "message": "Logout Berhasil!"  
4 }
```

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

7. Lakukan commit perubahan file pada Github.



## Praktikum 4 – Implementasi CRUD dalam RESTful API

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

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

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

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

```
namespace App\Http\Controllers\Api;

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

Windsurf: Refactor | Explain
class LevelController extends Controller
{
    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function index()
    {
        return LevelModel::all();
    }

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

    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function show(LevelModel $level)
    {
        return LevelModel::find($level);
    }

    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function update(Request $request, LevelModel $level)
    {
        $level->update($request->all());
        return LevelModel::find($level);
    }

    Windsurf: Refactor | Explain | Generate Function Comment | X
    public function destroy(LevelModel $level)
    {
        $level->delete();
        return response()->json([
            'success' => true,
            'message' => 'Data terhapus',
        ]);
    }
}
```



- 
3. Kemudian kita lengkapi routes pada api.php.

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

/*
|--------------------------------------------------------------------------
| 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::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 screenshot hasil percobaan Anda.**

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: http://127.0.0.1:8000/api/levels
- Headers: Authorization (No Auth)
- Body: JSON (2 items)
- Test Results: 200 OK (351 ms, 706 B)
- Body content (JSON):

```
1  [
2   {
3     "level_id": 1,
4     "level_kode": "ADM",
5     "level_nama": "Administrator",
6     "created_at": null,
7     "updated_at": null
8   },
9   {
10    "level_id": 2,
11    "level_kode": "MNG",
12    "level_nama": "Manager",
13  }
]
```

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



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

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body (9) Headers (9) Test Results

201 Created 403 ms 421 B

Key	Value	Description	Bulk Edit
level_kode	SPV		
level_nama	Supervisor		

{ "level\_kode": "SPV", "level\_nama": "Supervisor", "updated\_at": "2025-05-31T09:51:59.000000Z", "created\_at": "2025-05-31T09:51:59.000000Z", "level\_id": 5 }

Jelaskan dan berikan screenshoot hasil percobaan Anda.

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

GET http://127.0.0.1:8000/api/levels/5

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Auth Type

No Auth

No Auth

This request does not use any authorization.

200 OK 371 ms 418 B

Key	Value	Description	Bulk Edit
level_id	5		
level_kode	SPV		
level_nama	Supervisor		
created_at	"2025-05-31T09:51:59.000000Z"		
updated_at	"2025-05-31T09:51:59.000000Z"		

[ { "level\_id": 5, "level\_kode": "SPV", "level\_nama": "Supervisor", "created\_at": "2025-05-31T09:51:59.000000Z", "updated\_at": "2025-05-31T09:51:59.000000Z" } ]

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



PUT http://127.0.0.1:8000/api/levels/5?level\_kode=SPR

Params: Authorization, Headers (9), Body (checked), Scripts, Tests, Settings

Body: form-data, x-www-form-urlencoded, raw, binary, GraphQL

Key	Value	Description
level_kode	SPR	

Body: Cookies (2), Headers (9), Test Results | 200 OK | 291 ms | 418 B | ⏱ | ⚡ | 🔍 | ⌂ | ⌂

{ } JSON | Preview | Visualize | ↻

```
1 [  
2 {  
3   "level_id": 5,  
4   "level_kode": "SPR",  
5   "level_nama": "Supervisor",  
6   "created_at": "2025-05-31T09:51:59.000000Z",  
7   "updated_at": "2025-05-31T09:54:22.000000Z"  
8 }]  
9 ]
```

Jelaskan dan berikan screenshot hasil percobaan Anda.

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

DELETE http://127.0.0.1:8000/api/levels/5

Params: Authorization, Headers (9), Body (checked), Scripts, Tests, Settings

Body: form-data, x-www-form-urlencoded, raw, binary, GraphQL

Key	Value	Description
level_kode	SPR	

Body: Cookies (2), Headers (9), Test Results | 200 OK | 599 ms | 313 B | ⏱ | ⚡ | 🔍 | ⌂ | ⌂

{ } JSON | Preview | Visualize | ↻

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

9. Lakukan commit perubahan file pada Github.

## TUGAS

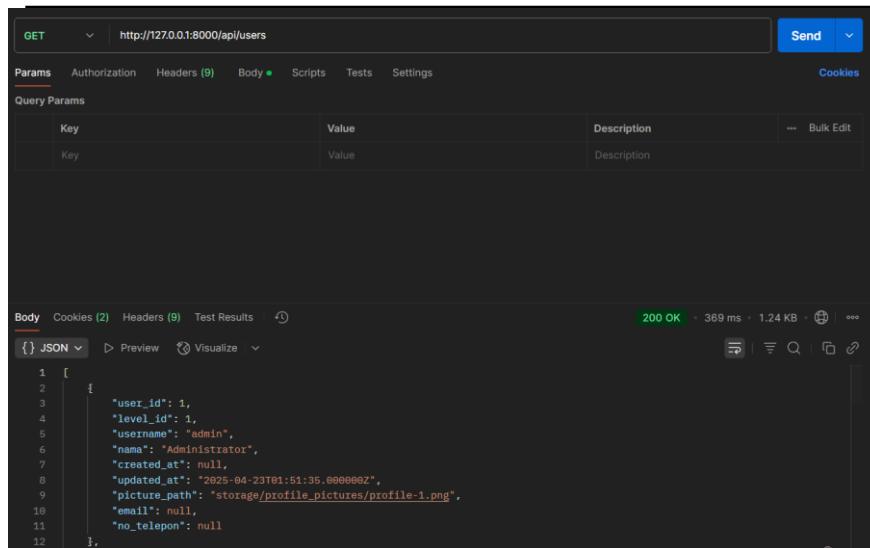
Implementasikan CRUD API pada tabel lainnya yaitu tabel m\_user, m\_kategori, dan m\_barang

**User**

index:



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>

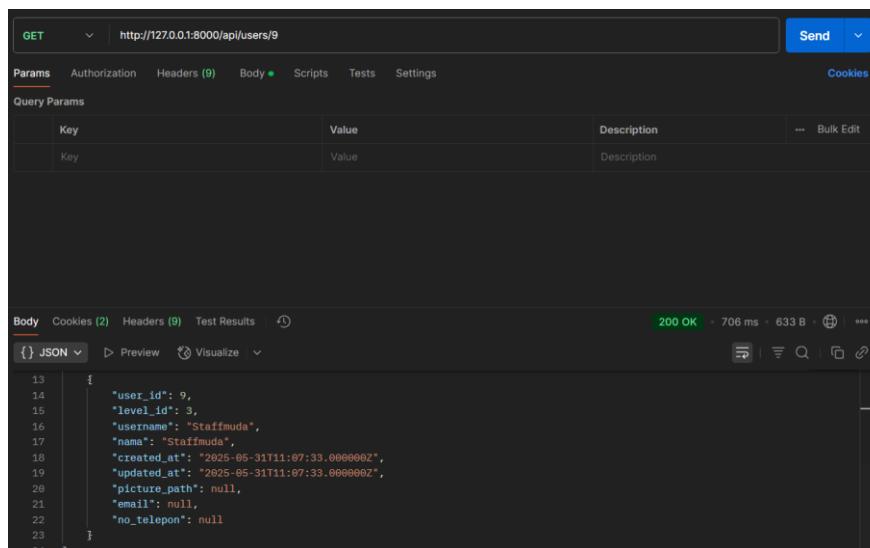


Body Cookies (2) Headers (9) Test Results ⏱

{ } JSON ▾ D Preview ⏷ Visualize ▾

```
1 [  
2   {  
3     "user_id": 1,  
4     "level_id": 1,  
5     "username": "admin",  
6     "nama": "Administrator",  
7     "created_at": null,  
8     "updated_at": "2025-04-23T01:51:35.000000Z",  
9     "picture_path": "storage/profile_pictures/profile-1.png",  
10    "email": null,  
11    "no_telepon": null  
12  },  
13 ]
```

Show:

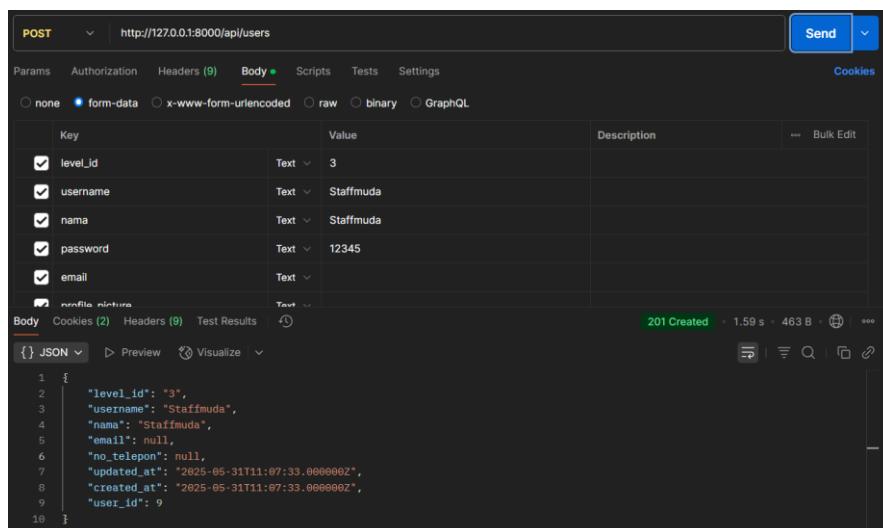


Body Cookies (2) Headers (9) Test Results ⏱

{ } JSON ▾ D Preview ⏷ Visualize ▾

```
13 {  
14   "user_id": 9,  
15   "level_id": 3,  
16   "username": "Staffmuda",  
17   "nama": "Staffmuda",  
18   "created_at": "2025-05-31T11:07:33.000000Z",  
19   "updated_at": "2025-05-31T11:07:33.000000Z",  
20   "picture_path": null,  
21   "email": null,  
22   "no_telepon": null  
23 }
```

Create:



POST http://127.0.0.1:8000/api/users Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

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

Key	Value	Description	... Bulk Edit
<input checked="" type="checkbox"/> level_id	Text ▾ 3		
<input checked="" type="checkbox"/> username	Text ▾ Staffmuda		
<input checked="" type="checkbox"/> nama	Text ▾ Staffmuda		
<input checked="" type="checkbox"/> password	Text ▾ 12345		
<input checked="" type="checkbox"/> email	Text ▾		
profile picture	Text ▾		

Body Cookies (2) Headers (9) Test Results ⏱

{ } JSON ▾ D Preview ⏷ Visualize ▾

```
1 {  
2   "level_id": "3",  
3   "username": "Staffmuda",  
4   "nama": "Staffmuda",  
5   "email": null,  
6   "no_telepon": null,  
7   "updated_at": "2025-05-31T11:07:33.000000Z",  
8   "created_at": "2025-05-31T11:07:33.000000Z",  
9   "user_id": 9  
10 }
```



Update:

The screenshot shows a POST request to <http://127.0.0.1:8000/api/users/9?nama=Staff1>. The 'Body' tab contains the following JSON payload:

```
{  
    "user_id": 9,  
    "level_id": 3,  
    "username": "Staffmuda",  
    "nama": "Staff1",  
    "created_at": "2025-05-31T11:07:33.000000Z",  
    "updated_at": "2025-05-31T11:11:04.000000Z",  
    "picture_path": null,  
    "email": null,  
    "no_telepon": null  
}
```

The response status is 200 OK with a response time of 290 ms and a body size of 630 B.

Delete:

The screenshot shows a DELETE request to <http://127.0.0.1:8000/api/users/9>. The 'Body' tab contains an empty JSON object: {}.

The response status is 200 OK with a response time of 273 ms and a body size of 313 B. The response message is: {"suscess": true, "message": "data terhapus"}

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