



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

```
php artisan vendor:publish --  
provider="Tyson\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:



```
<?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';
}
```

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 }
```

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');
```

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.

POST localhost/PWL_POS/public/api/register Send

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

Query Params

Key	Value	Bulk Edit
Key	Value	

Body 422 Unprocessable Content 272 ms 534 B Save Response

Pretty Raw Preview Visualize JSON

```
1  [
2    "username": [
3      "The username field is required."
4    ],
5    "nama": [
6      "The nama field is required."
7    ],
8    "password": [
9      "The password field is required."
10   ]
11 ]
```

Jika berhasil akan muncul error validasi seperti gambar di atas.

Jawab:



POST

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (10) Test Results 422 Unprocessable Content 363 ms 507 B Save as example

Pretty Raw Preview Visualize JSON

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

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.



POST localhost/PWL_POS/public/api/register Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies

form-data

Key	Value
<input checked="" type="checkbox"/> username	penggunasatu
<input checked="" type="checkbox"/> nama	Pengguna 1
<input checked="" type="checkbox"/> password	12345
<input checked="" type="checkbox"/> password_confirmation	12345
<input checked="" type="checkbox"/> level_id	2

Body Cookies Headers (11) Test Results 201 Created 624 ms 645 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "user": {
4     "username": "penggunasatu",
5     "nama": "Pengguna 1",
6     "password": "$2y$12$Eb2SrV1jsykINytYGtrHi0DVAKcK5p6EgnZnmbChkPicIu7S0QJJJu",
7     "level_id": "2",
8     "updated_at": "2024-04-22T15:56:04.000000Z",
9     "created_at": "2024-04-22T15:56:04.000000Z",
10    "user_id": 17
11  }
```

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

Body Cookies Headers (10) Test Results 201 Created 1213 ms 496 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "user": {
4     "username": "penggunaSatu",
5     "nama": "Pengguna1",
6     "level_id": "2",
7     "updated_at": "2024-05-18T15:13:00.000000Z",
8     "created_at": "2024-05-18T15:13:00.000000Z",
9     "user_id": 24
10  }
```

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()->guard('api')->user(),
39             'token' => $token
40         ], 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();
});
```

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

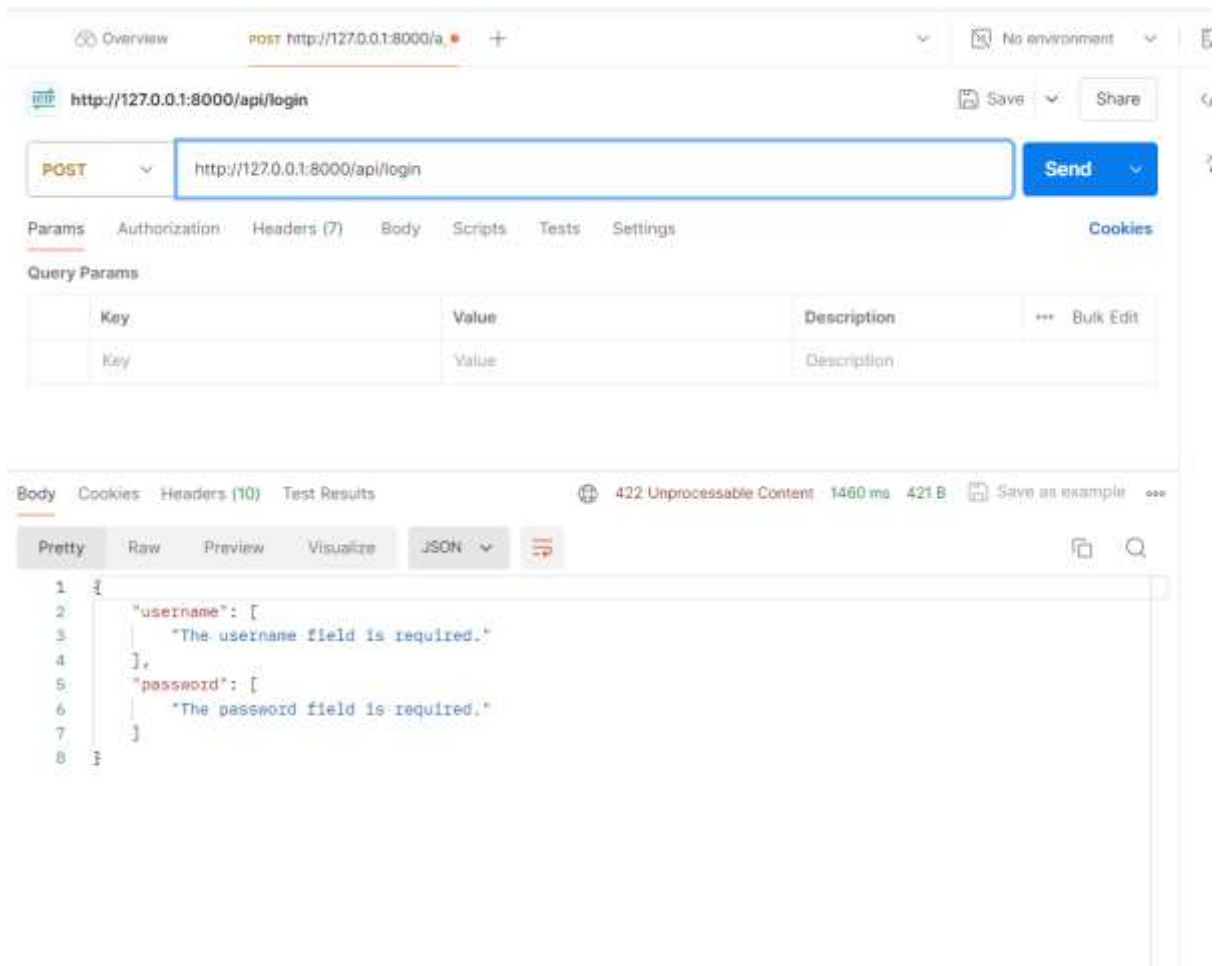
Pretty Raw Preview Visualize JSON

```
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 screenshot hasil percobaan Anda.

Jawab



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



POST localhost/PWL_POS/public/api/login Send

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

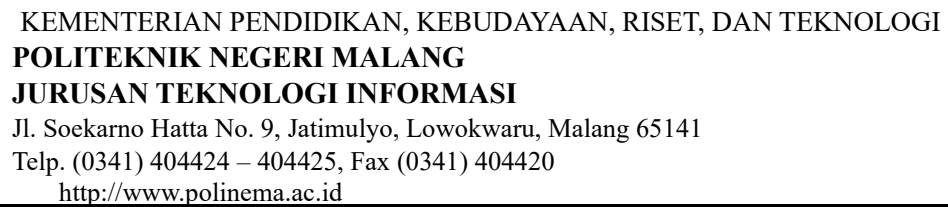
none **form-data** x-www-form-urlencoded raw binary

Key	Value	...	Bulk Edit
<input checked="" type="checkbox"/> username	penggunasatu		
<input checked="" type="checkbox"/> password	12345		
Key	Value		

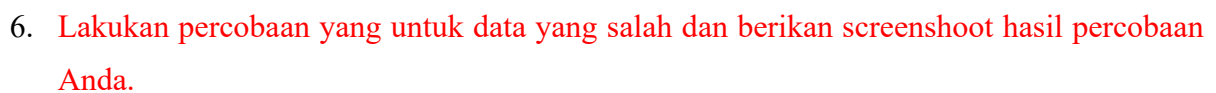
body Cookies Headers (11) Test Results Status: 200 OK Time: 1501 ms Size: 986 B Save Response

Pretty Raw Preview Visualize JSON

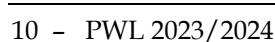
```
1  {
2    "success": true,
3    "user": {
4      "user_id": 17,
5      "level_id": 2,
6      "username": "penggunasatu",
7      "nama": "Pengguna 1",
8      "password": "$2y$12$Eb2SxV1jsykINytYGtzHi0DVAKcK5p6EgnZnmbChkPicIu7S0QJJJu",
9      "created_at": "2024-04-22T15:56:04.000000Z",
10     "updated_at": "2024-04-22T15:56:04.000000Z"
11   },
12   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYXRob3N0L1BXTF9QT1MtbnVpbi9wdWJsawMvYXBPL2xvZ21uIiwiaWF0Ij01"
13 }
```

Jawab:

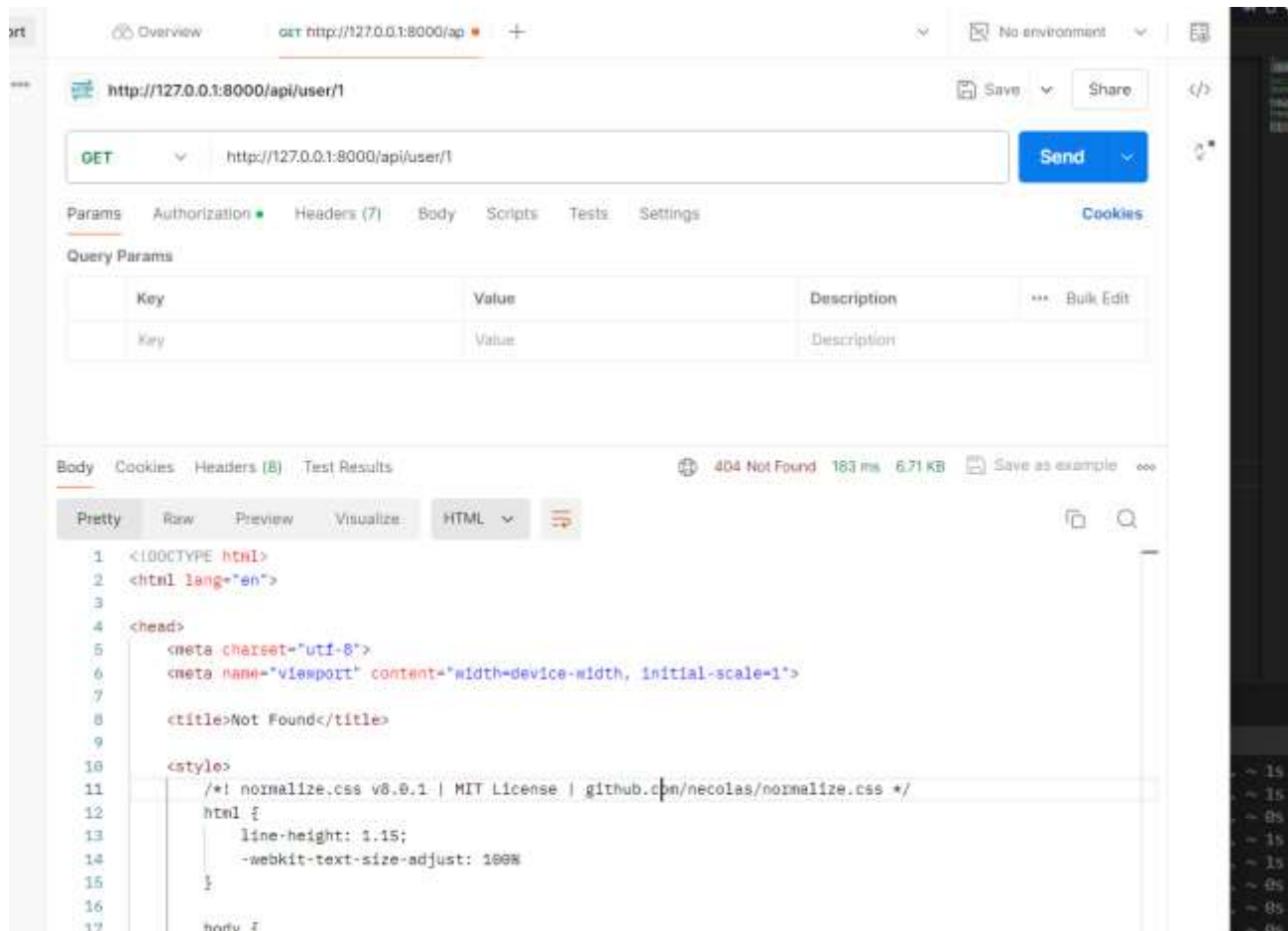


Jawab





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`
2. Buat Controller baru dengan nama `LogoutController`.
`php artisan make:controller Api/LogoutController`
3. Buka file tersebut dan ubah kode menjadi seperti berikut.

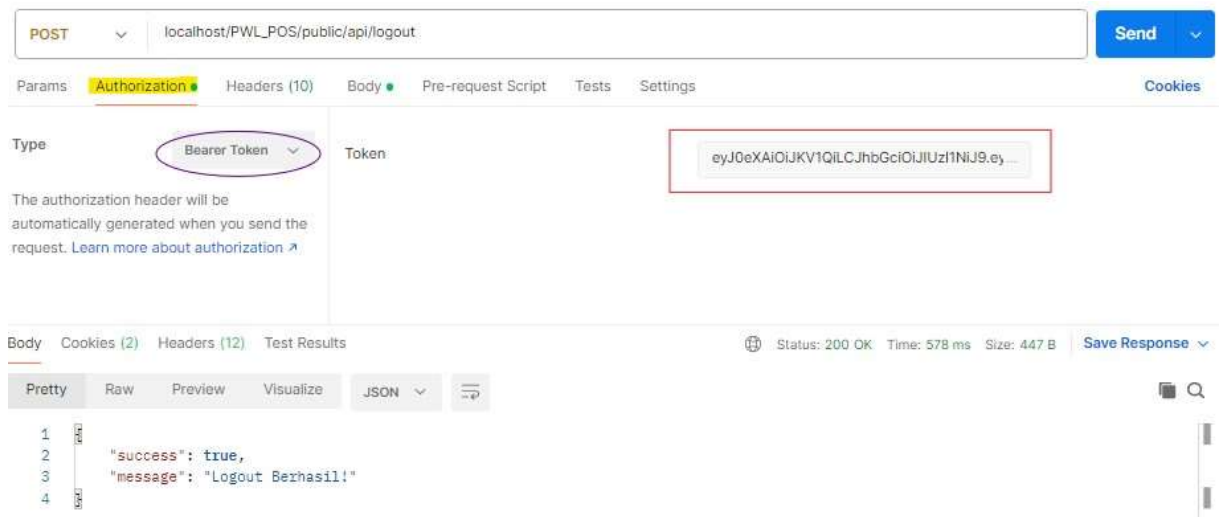


```
1  <?php
2
3  namespace App\Http\Controllers\Api;
4  use Illuminate\Http\Request;
5  use App\Http\Controllers\Controller;
6  use Tymon\JWTAuth\Facades\JWTAuth;
7  use Tymon\JWTAuth\Exceptions\JWTException;
8  use Tymon\JWTAuth\Exceptions\TokenExpiredException;
9  use Tymon\JWTAuth\Exceptions\TokenInvalidException;
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 }
```

4. Lalu kita tambahkan routes pada api.php

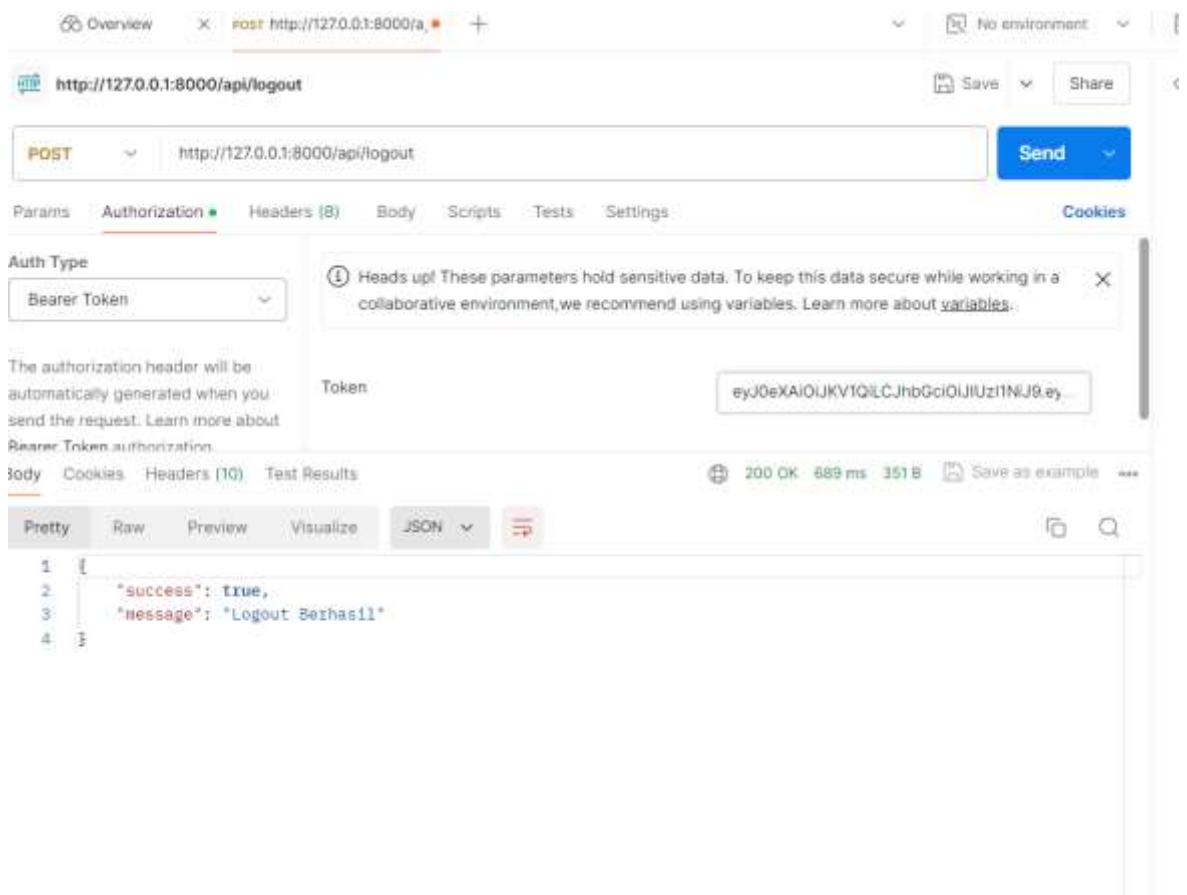
```
Route::post('/logout', App\Http\Controllers\Api\LogoutController::class)->name('logout');
```

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.



Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

Jawab



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;

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.

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

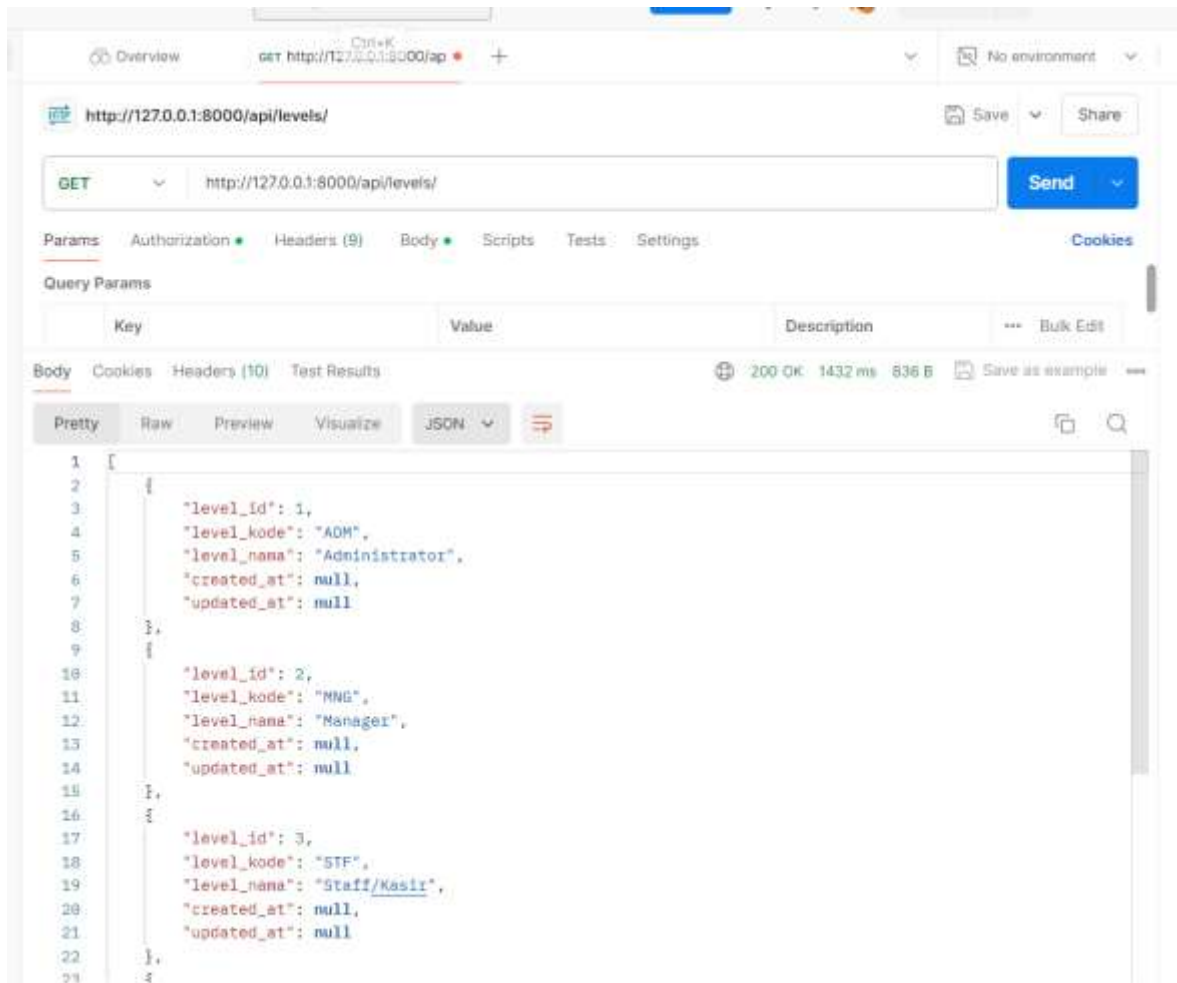
Route::get('levels', [LevelController::class, 'index']);
Route::post('levels', [LevelController::class, 'store']);
Route::get('levels/{level}', [LevelController::class, 'show']);
Route::put('levels/{level}', [LevelController::class, 'update']);
Route::delete('levels/{level}', [LevelController::class,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

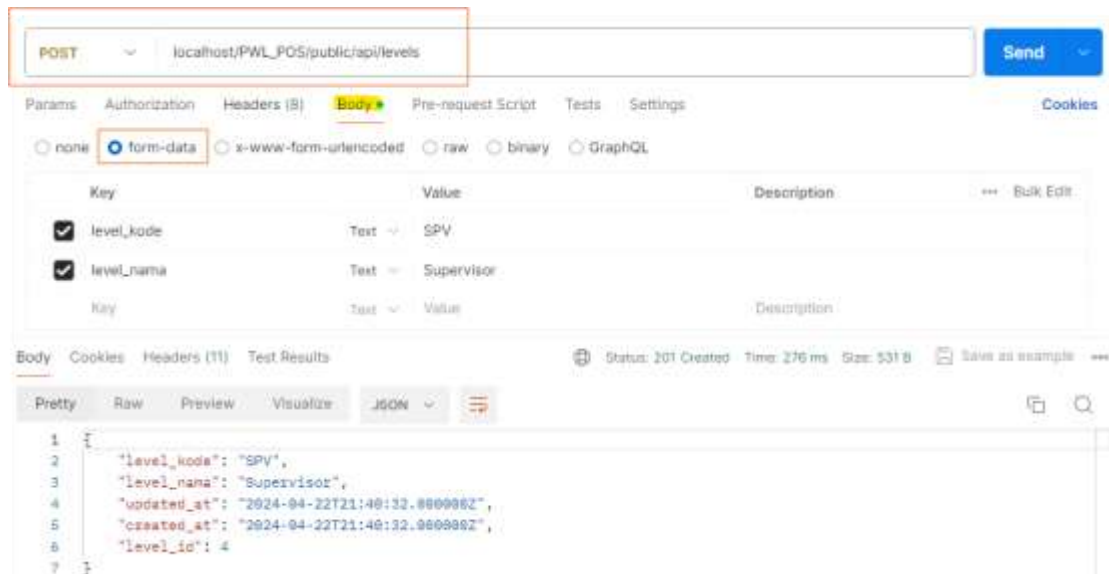
Jawab:



Method GET pada <http://127.0.0.1:8000/api/levels/> digunakan untuk mengambil data level dari server tanpa mengubah data tersebut



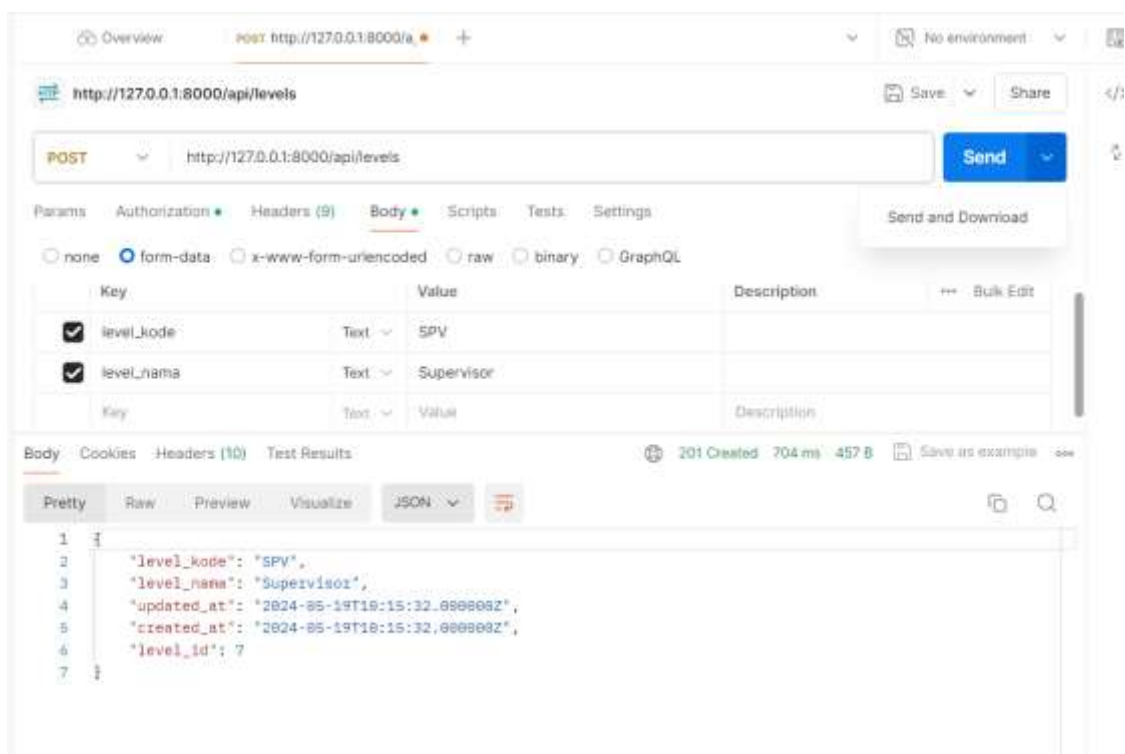
5. Kemudian, lakukan percobaan penambahan data dengan URL : localhost/PWL_POSmain/public/api/levels dan method POST seperti di bawah ini.



Jelaskan dan berikan screenshoot hasil percobaan Anda.

Jawab:

Method GET pada <http://127.0.0.1:8000/api/levels/> digunakan untuk mengambil data level dari server tanpa mengubah data tersebut

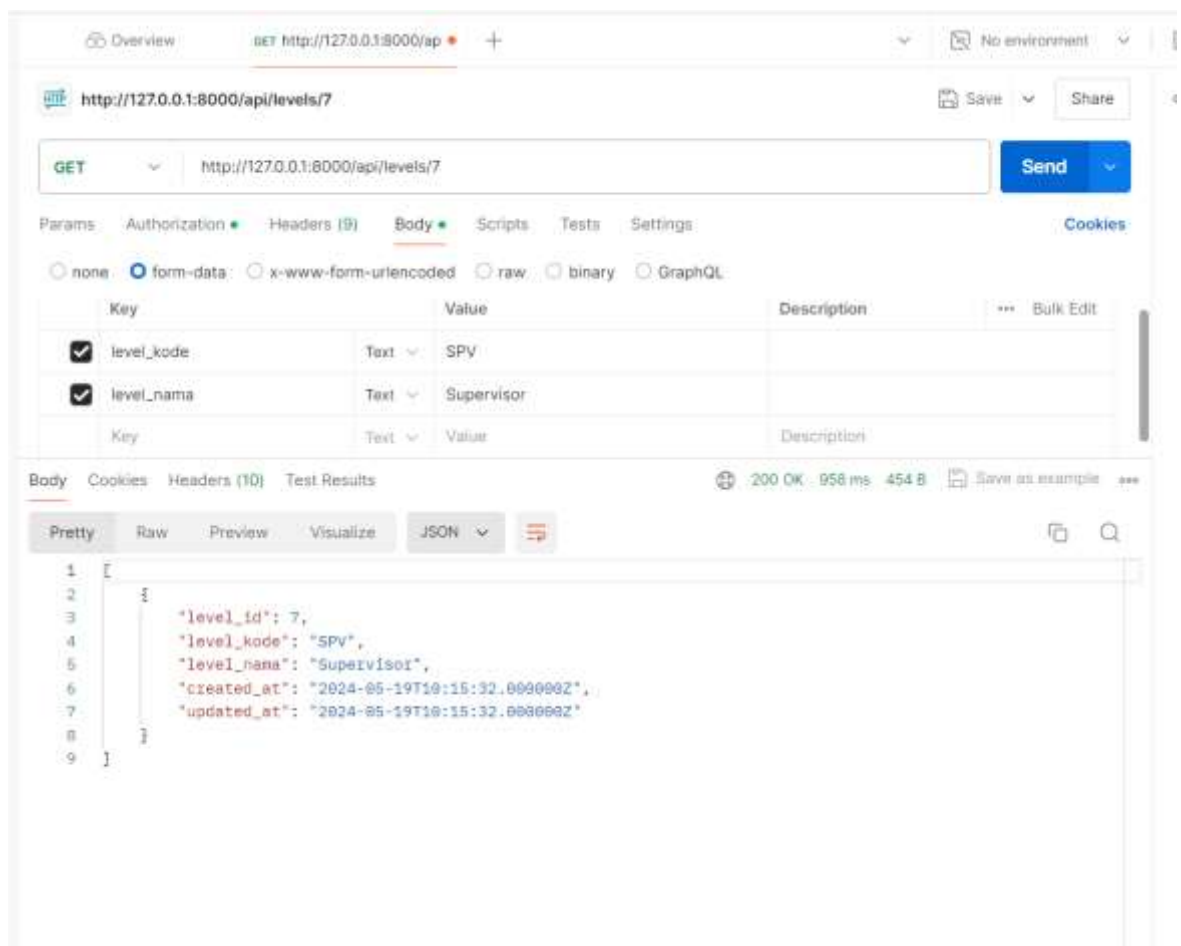




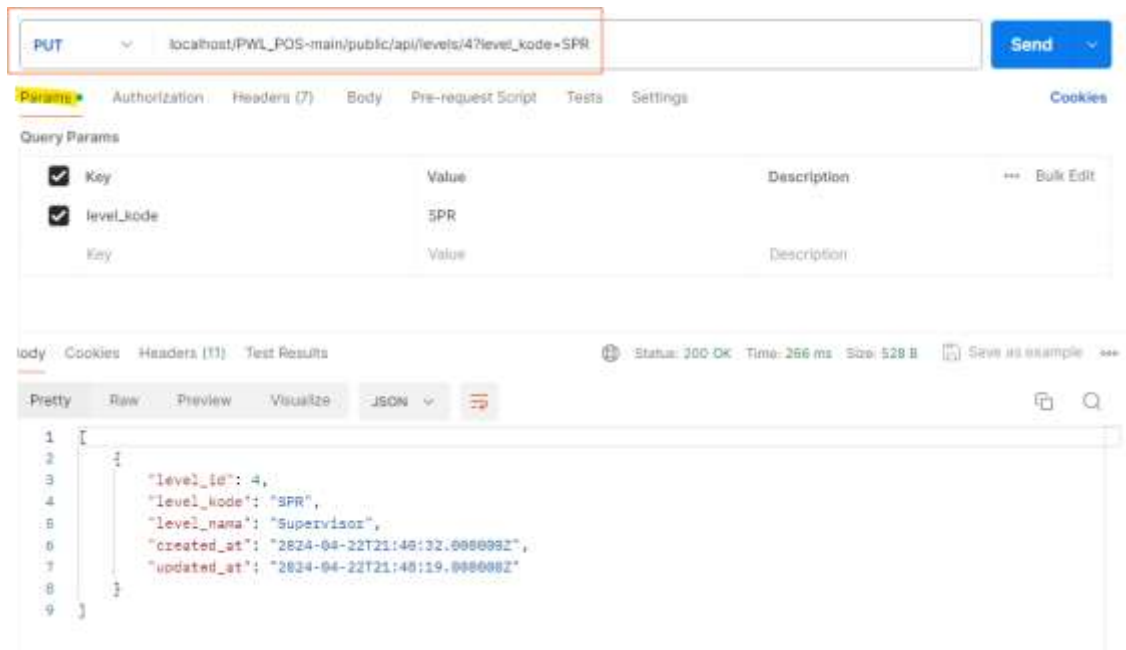
6. Berikutnya lakukan percobaan menampilkan detail data. **Jelaskan dan berikan screenshot hasil percobaan Anda.**

Jawab:

Method GET pada <http://127.0.0.1:8000/api/levels/> digunakan untuk mengambil data level dari server tanpa mengubah data tersebut, data yang diambil adalah data yang memiliki level_id 7



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.



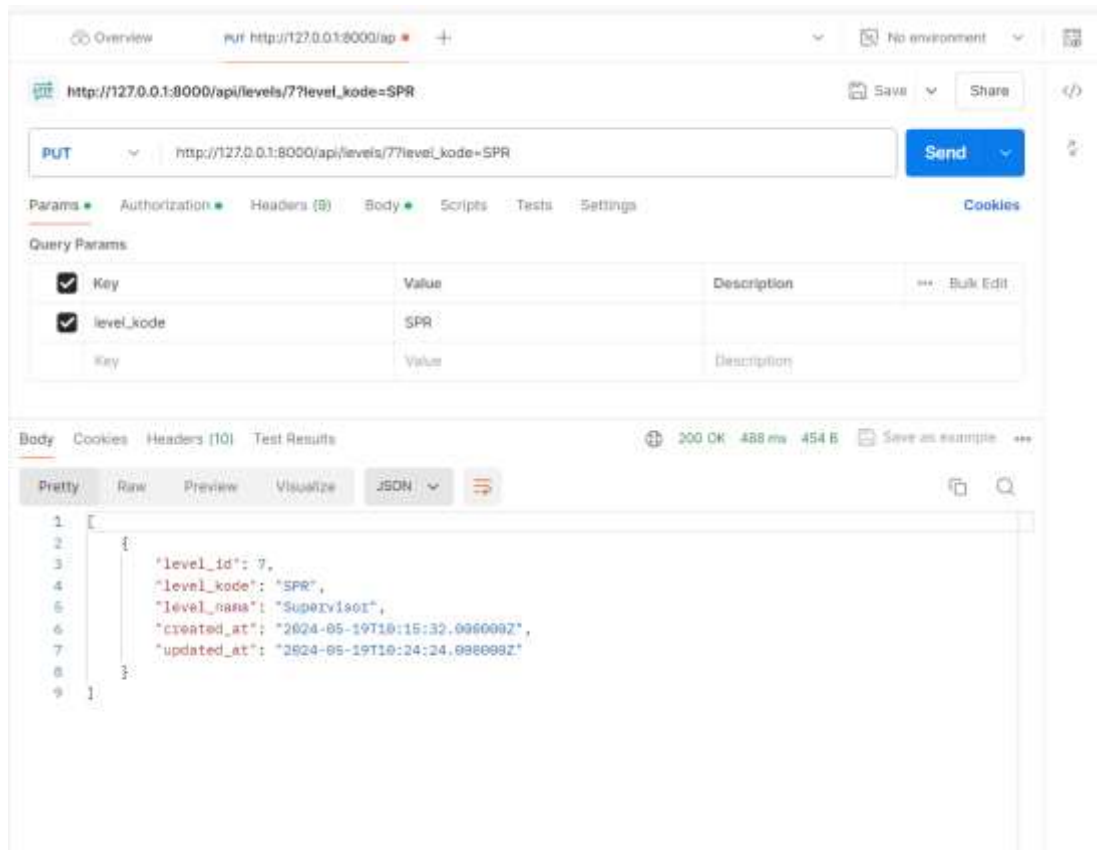
Jelaskan dan berikan screenshot hasil percobaan Anda.

Jawab

Permintaan tersebut digunakan untuk memperbarui data pada level dengan **level_id** 4. Berikut adalah penjelasan rinci dari elemen-elemen dalam gambar tersebut:

URL dan Method

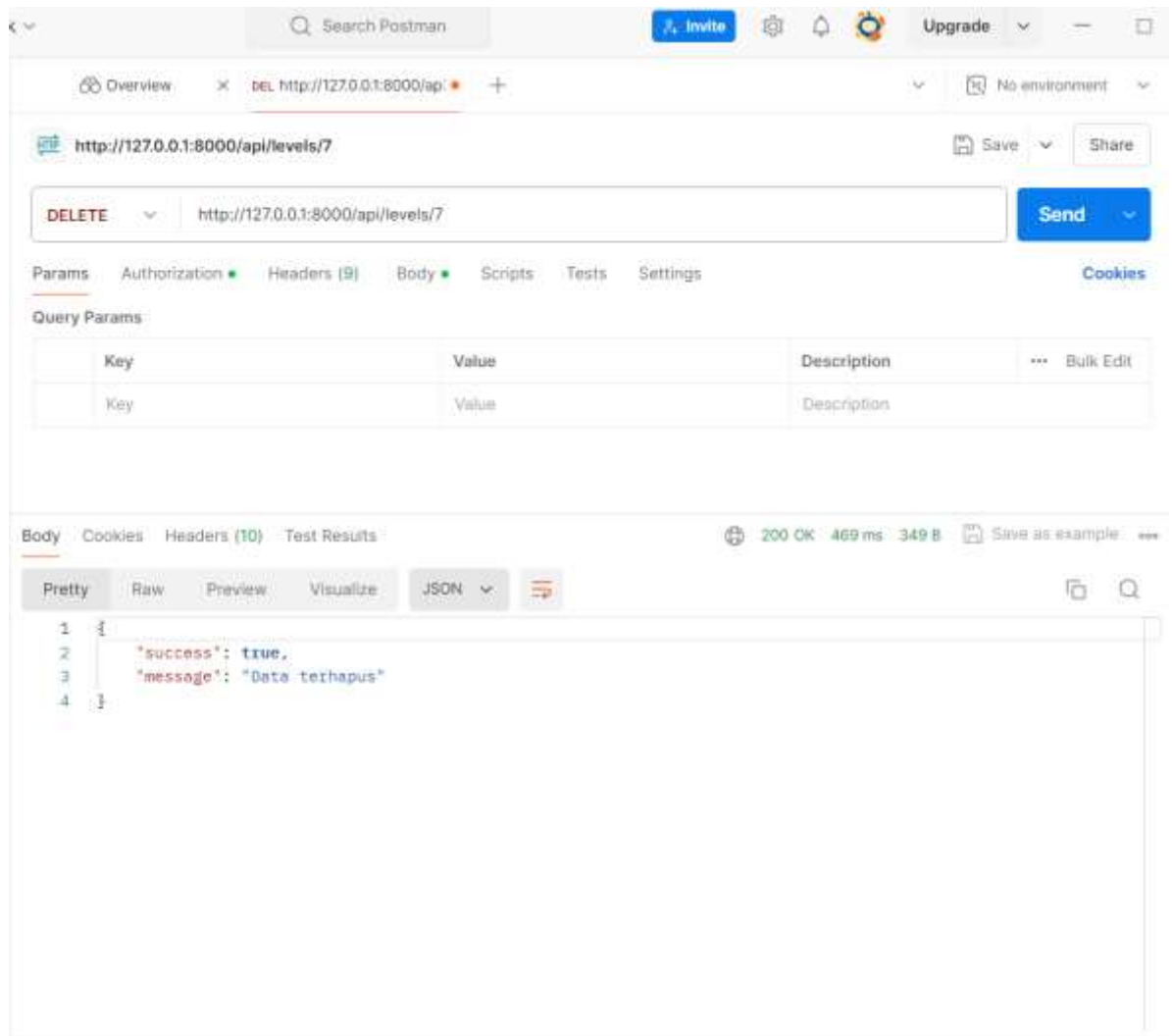
- **URL:** `localhost/PWL_POS-main/public/api/levels/4?level_kode=SPR`
 - `localhost/PWL_POS-main/public/api/levels/4`: Menunjukkan bahwa permintaan ini ditujukan untuk memperbarui data level dengan **level_id** 4.
 - `?level_kode=SPR`: Ini adalah parameter query yang dikirimkan bersama dengan permintaan, di mana **level_kode** diatur ke **SPR**.
- **Method:** PUT
 - Method HTTP PUT digunakan untuk memperbarui sumber daya yang ada dengan data baru. Dalam konteks ini, PUT digunakan untuk memperbarui informasi level dengan **level_id** 4.



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

Jawab:

Pada method delete pada postman untuk menghapus data dengan level__id 7



9. Lakukan commit perubahan file pada Github.

TUGAS

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

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

HASIL TABEL M_USER

- Post



http://127.0.0.1:8000/api/user

POST http://127.0.0.1:8000/api/user

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 2		
<input checked="" type="checkbox"/> username	Text newMember		
<input checked="" type="checkbox"/> nama	Text penggunabaru		
<input checked="" type="checkbox"/> password	Text 12345		
Key	Text Value	Description	

Body Cookies Headers (10) Test Results 201 Created 1150 ms 472 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "level_id": "2",
3   "username": "newMember",
4   "nama": "penggunabaru",
5   "updated_at": "2024-05-19T11:12:54.000000Z",
6   "created_at": "2024-05-19T11:12:54.000000Z",
7   "user_id": 25
8 }
```

- Get



Overview GET http://127.0.0.1:8000/ap + No environment

HTTP http://127.0.0.1:8000/api/user/25 Save Share

GET http://127.0.0.1:8000/api/user/25 Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	+++ Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results 200 OK 296 ms 494 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "user_id": 25,
3   "level_id": 2,
4   "username": "newMember",
5   "nama": "penggunabaru",
6   "created_at": "2024-05-19T11:12:54.000000Z",
7   "updated_at": "2024-05-19T11:12:54.000000Z",
8   "profil_img": null,
9   "status": 0
10 }
```




- Update

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

- Method:** PUT
- URL:** `http://127.0.0.1:8000/api/user/25?username=nm`
- Query Params:**

Key	Value	Description
username	nm	
- Response:**

```
12 {
13   "user_id": 25,
14   "level_id": 2,
15   "username": "nm",
16   "nama": "penggunabaru",
17   "created_at": "2024-05-19T11:12:54.000000Z",
18   "updated_at": "2024-05-19T11:15:21.000000Z",
19   "profil_img": null,
20   "status": 0
21 }
22 }
```



- Delete

Overview **DEL** <http://127.0.0.1:8000/api/> **No environment**

<http://127.0.0.1:8000/api/user/25> **Save** **Share**

DELETE <http://127.0.0.1:8000/api/user/25> **Send**

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	*** Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results **200 OK 474 ms 349 B** **Save as example**

Pretty Raw Preview Visualize **JSON**

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



HASIL TABEL M_KATEGORI

- Post

Overview POST http://127.0.0.1:8000/a. +

Save Share

POST http://127.0.0.1:8000/api/kategoris 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"/>	kategori_kode	Text spt		
<input checked="" type="checkbox"/>	kategori_nama	Text Sepatu		
	Key	Text Value	Description	

Body Cookies Headers (10) Test Results 201 Created 608 ms 462 B Save as example ***

Pretty Raw Preview Visualize JSON

```
1 {
2   "kategori_kode": "spt",
3   "kategori_nama": "Sepatu",
4   "updated_at": "2024-05-19T11:34:44.000000Z",
5   "created_at": "2024-05-19T11:34:44.000000Z",
6   "kategori_id": 8
7 }
```



- Update

The screenshot shows a Postman interface for a PUT request. The URL is `http://127.0.0.1:8000/api/kategoris/8?kategori_kode=spu`. The request body is a JSON object with the following fields:

```
{  "kategori_id": 8,  "kategori_kode": "spu",  "kategori_nama": "Sepatu",  "created_at": "2024-05-19T11:34:44.000000Z",  "updated_at": "2024-05-19T11:35:40.000000Z"}
```

The response status is 200 OK, with a response time of 366 ms and a body size of 459 B. The response body is displayed in JSON format.



- Get

Overview GET http://127.0.0.1:8000/ap + No environment

http://127.0.0.1:8000/api/kategoris/8 Save Share

GET http://127.0.0.1:8000/api/kategoris/8 Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (10) Test Results 200 OK 277 ms 459 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "kategori_id": 8,
4     "kategori_kode": "spu",
5     "kategori_nama": "Sepatu",
6     "created_at": "2024-05-19T11:34:44.000000Z",
7     "updated_at": "2024-05-19T11:35:40.000000Z"
8   }
9 ]
```



- Delete

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

- Method:** DELETE
- URL:** `http://127.0.0.1:8000/api/kategoris/8`
- Response Status:** 200 OK
- Response Time:** 433 ms
- Response Size:** 349 B
- Response Body (JSON):**

```
{
  "success": true,
  "message": "Data terhapus"
}
```



HASIL TABEL M_BARANG

- Post

Overview POST http://127.0.0.1:8000/a

http://127.0.0.1:8000/api/barangs

POST http://127.0.0.1:8000/api/barangs

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

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

Key	Value	Description
kategori_id	1	
barang_kode	ovr	
barang_nama	Overall	
harga_beli	55000	
harga_jual	100000	

Body Cookies Headers (10) Test Results

201 Created 477 ms 519 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "kategori_id": "1",
3   "barang_kode": "ovr",
4   "barang_nama": "Overall",
5   "harga_beli": "55000",
6   "harga_jual": "100000",
7   "updated_at": "2024-05-19T11:46:16.000000Z",
8   "created_at": "2024-05-19T11:46:16.000000Z",
9   "barang_id": 14
10 }
```




- Get

Overview x GET http://127.0.0.1:8000/ap +

http://127.0.0.1:8000/api/barangs/14 Save Share

GET http://127.0.0.1:8000/api/barangs/14 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"/> kategori_id	Text 1		
<input checked="" type="checkbox"/> barang_kode	Text ovr		
<input checked="" type="checkbox"/> barang_nama	Text Overall		
<input checked="" type="checkbox"/> harga_beli	Text 55000		
<input checked="" type="checkbox"/> harga_jual	Text 100000		
Key	Text Value	Description	

Body Cookies Headers (10) Test Results 200 OK 606 ms 693 B Save as example

Pretty Raw Preview Visualize JSON

```
12 {
13   "barang_id": 14,
14   "kategori_id": 1,
15   "barang_kode": "ovr",
16   "barang_nama": "Overall",
17   "harga_beli": 55000,
18   "harga_jual": 100000,
19   "created_at": "2024-05-19T11:46:16.000000Z",
20   "updated_at": "2024-05-19T11:46:16.000000Z"
21 }
22 }
```

Postbot Runner Start Proxy Cookies Vault Trash



- Update

Overview PUT http://127.0.0.1:8000/ap +

http://127.0.0.1:8000/api/barangs/14?barang_kode=ovl

PUT http://127.0.0.1:8000/api/barangs/14?barang_kode=ovl Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> barang_kode	ovl			
Key	Value	Description		

Body Cookies Headers (10) Test Results 200 OK 613 ms 693 B Save as example

Pretty Raw Preview Visualize JSON

```
12 {
13   "barang_id": 14,
14   "kategori_id": 1,
15   "barang_kode": "ovl",
16   "barang_nama": "Overall",
17   "harga_beli": 55000,
18   "harga_jual": 100000,
19   "created_at": "2024-05-19T11:46:16.000000Z",
20   "updated_at": "2024-05-19T11:48:59.000000Z"
21 }
22 }
```

Postbot Runner Start Proxy Cookies Vault Trash



- Delete

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

- URL:** `http://127.0.0.1:8000/api/barangs/14`
- Method:** DELETE
- Response:** 200 OK, 426 ms, 348 B
- Body (JSON):**

```
{
  "success": true,
  "message": "Data terhapus"
}
```