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)
Nama	:	Karina Ika Indasa

JOBSHEET 10
RESTFUL API

Sebelumnya kita sudah membahas mengenai *authentication, authorization,* dan *middleware* pada Laravel. Dimana kita telah membuat fungsi login, register, logout, serta pemilihan role dan penerapan session pada halaman web. Pada pertemuan kali ini, kita akan mempelajari penerapan RESTFUL API di dalam project Laravel.

Sebelum kita masuk materi, kita buat dulu project baru yang akan kita gunakan untuk membangun aplikasi sederhana dengan topik *Point of Sales (PoS)*, sesuai dengan `Studi Kasus PWL.pdf`. Jadi kita bikin project Laravel 10 dengan nama **PWL_POS.**

Project **PWL_POS** akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajarai

A. RESTFUL API

Representational State Transfer (REST) adalah gaya arsitektur perangkat lunak yang mendefinisikan seperangkat prinsip untuk merancang jaringan aplikasi terdistribusi. RESTful API adalah aplikasi pemrograman antarmuka yang mengikuti prinsip-prinsip REST untuk mentransfer data antara klien dan server.

RESTful API adalah salah satu arsitektur dalam API (*Application Program Interface*) yang menggunakan request HTTP untuk mengakses data. Data diakses dengan menggunakan HTTP method GET, PUT, POST dan DELETE yang merujuk pada operasi pembacaan, pembaruan, pembuatan dan penghapusan pada resource. Selain HTTP method, dalam RESTful atau REST digunakan juga HTTP response untuk mendefinisikan respon data yang dikembalikan. Format respon yang umum digunakan berupa JSON (Javascript Object Notation).

B. JSON Web Token (JWT)

JWT adalah singkatan dari JSON Web Token. Ini adalah standar terbuka (RFC 7519) yang mendefinisikan format token yang kompak dan mandiri untuk mentransfer klaim antara dua pihak. JWT sering digunakan dalam otentikasi dan pertukaran informasi yang aman di lingkungan yang tidak terpercaya, seperti internet.

JWT terdiri dari tiga bagian yang dipisahkan oleh titik ("."): header, payload, dan signature. Setiap bagian ini terdiri dari data JSON yang dienkripsi menggunakan algoritma tertentu dan kemudian disatukan untuk membentuk token yang lengkap. Header berisi jenis token dan tipe algoritma yang digunakan untuk enkripsi. Payload berisi klaim atau informasi yang ingin disampaikan. Signature digunakan untuk memverifikasi bahwa token belum berubah dan datanya berasal dari sumber yang dipercayai.

JWT sering digunakan dalam sistem otentikasi dan otorisasi modern, seperti aplikasi web dan layanan web API, karena fleksibilitasnya dalam menyampaikan informasi terenkripsi secara ringkas.

Kita dapat menggunakan JWT untuk:

- Authentication

  Ketika pengguna melakukan authentication dan mendapatkan token, maka setiap permintaan berikutnya akan menyertakan token tersebut, dan memungkinkan pengguna untuk mengakses route, service, dan resources yang diizinkan.
- Pertukaran informasi

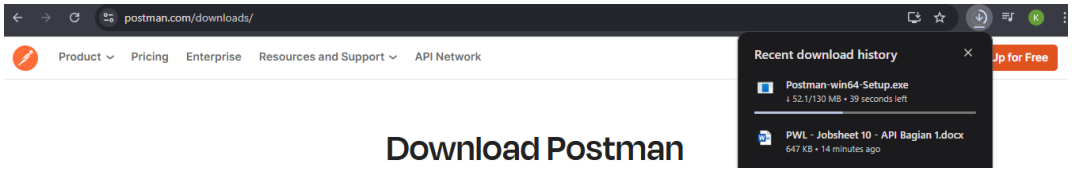
  JSON Web Token adalah cara yang baik untuk mengirimkan informasi antar pihak dengan aman. Dengan token yang sudah ditandatangani dengan algoritma RSA, maka kita bisa tahu siapa yang melakukan request tersebut.

  Berikut adalah cara kerja JWT :

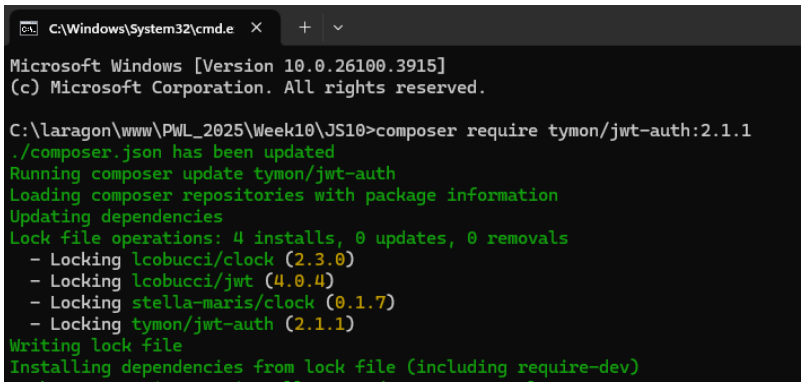
JWT (JSON Web Token) adalah cara untuk mentransfer informasi antara dua pihak secara aman sebagai objek JSON. Ini terdiri dari tiga bagian: header, payload, dan signature. Setelah pengguna berhasil autentikasi, server menghasilkan token JWT yang disematkan dalam permintaan HTTP. Server kemudian memvalidasi token untuk memberikan akses ke sumber daya yang diminta. Ini memberikan autentikasi yang aman dan stateless tanpa memerlukan penyimpanan status sesi di server.

Praktikum 1 – Membuat RESTful API Register

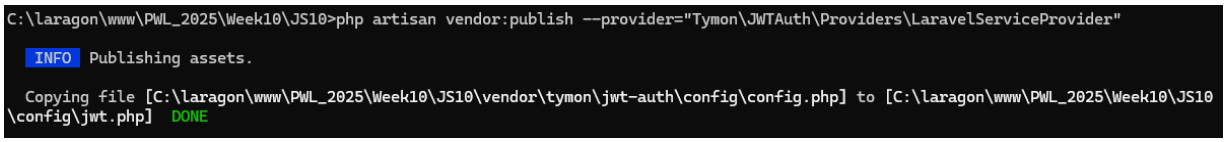
1. Sebelum memulai membuat REST API, terlebih dahulu download aplikasi Postman di https://www.postman.com/downloads.
   Aplikasi ini akan digunakan untuk mengerjakan semua tahap praktikum pada Jobsheet ini.



2. Lakukan instalasi JWT dengan mengetikkan perintah berikut:
   ```
   composer require tymon/jwt-auth:2.1.1
   ```
   Pastikan Anda terkoneksi dengan internet.



3. Setelah berhasil menginstall JWT, lanjutkan dengan publish konfigurasi file dengan perintah berikut:
   ```
   php artisan vendor:publish --
   provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
   ```



4. Jika perintah di atas berhasil, maka kita akan mendapatkan 1 file baru yaitu config/jwt.php. Pada file ini dapat dilakukan konfigurasi jika memang diperlukan.
5. Setelah itu jalankan peintah berikut untuk membuat secret key JWT.
   ```
   php artisan jwt:secret
   ```
   Jika berhasil, maka pada file .env akan ditambahkan sebuah baris berisi nilai key JWT_SECRET.

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',
    ],
],
```

```
38      'guards' => [
39          'web' => [
40              'driver' => 'session',
41              'provider' => 'users',
42          ],
43          'api' => [
44              'driver' => 'jwt',
45              'provider' => 'users',
46          ]
47      ],
```

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';
```

```
4   use Illuminate\Database\Eloquent\Model;
5   use Illuminate\Database\Eloquent\Factories\HasFactory;
6   use Illuminate\Database\Eloquent\Relations\BelongsTo;
7   use Tymon\JWTAuth\Contracts\JWTSubject;
8   use Illuminate\Foundation\Auth\User as Authenticatable; // im
9
        35 references | 0 implementations
10  class UserModel extends Authenticatable
11  {
            0 references | 0 overrides
12      public function getJWTIdentifier(): mixed
13      {
14          return $this->getKey();
15      }
            0 references | 0 overrides
16      public function getJWTCustomClaims(): array
17      {
18          return [];
19      }
```

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

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

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

```
C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/RegisterController

  INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\RegisterController.php] created successfully.
```

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

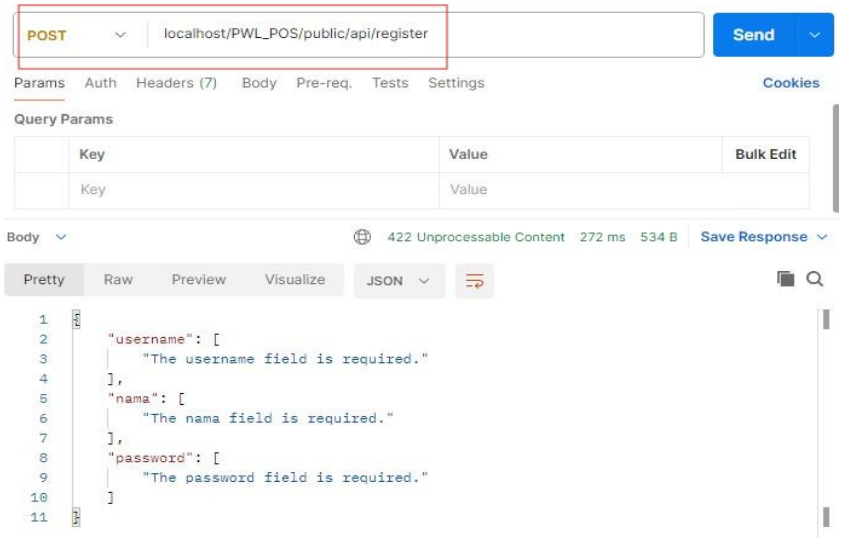
JS10 > app > Http > Controllers > Api > RegisterController.php > PHP Intelephense > RegisterController

```php
<?php

namespace App\Http\Controllers\Api;

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

0 references | 0 implementations
class RegisterController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed
    {
        //set validation
        $validator = Validator::make(data: $request->all(), rules: [
            'username' => 'required',
            'nama' => 'required',
            'password' => 'required min:5|confirmed',
            'level_id' => 'required'
        ]);

        //if validations fails
        if ($validator->fails()) {
            return response()->json(data: $validator->errors(), status: 422);
        }

        //create user

        $user = UserModel::create(attributes: [
            'username' => $request->username,
            'nama' => $request->nama,
            'password' => bcrypt(value: $request->password),
            'level_id' => $request->level_id,
        ]);

        //return response JSON user is created
        if ($user) {
            return response()->json(data: [
                'success' => true,
                'user' => $user,
            ], status: 201);
        }

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

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

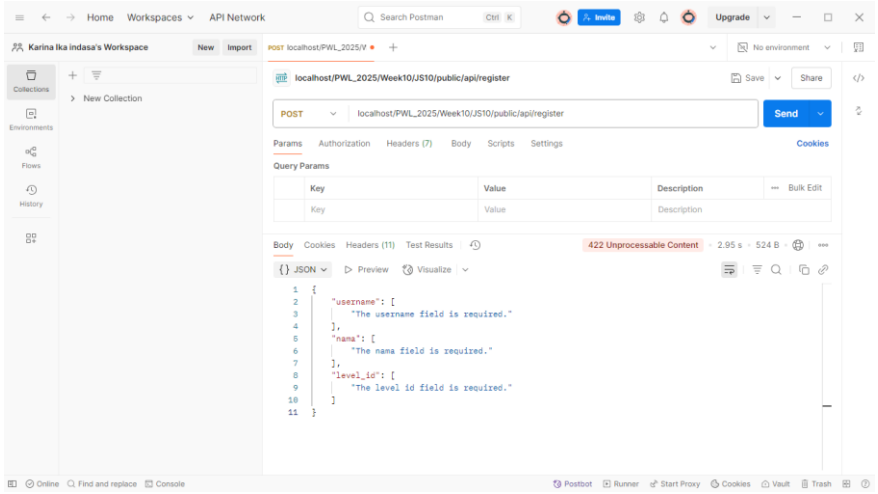
```
JS10 > routes > api.php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

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

Route::post(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.

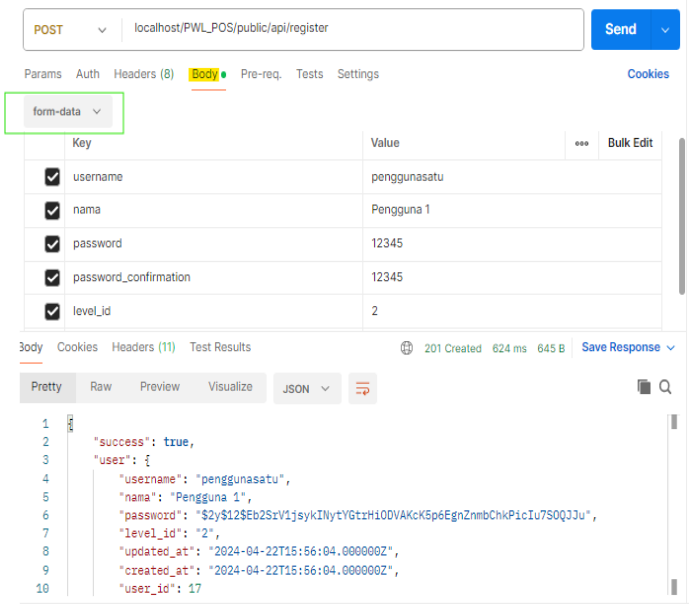


Jika berhasil akan muncul error validasi seperti gambar di atas.

Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.

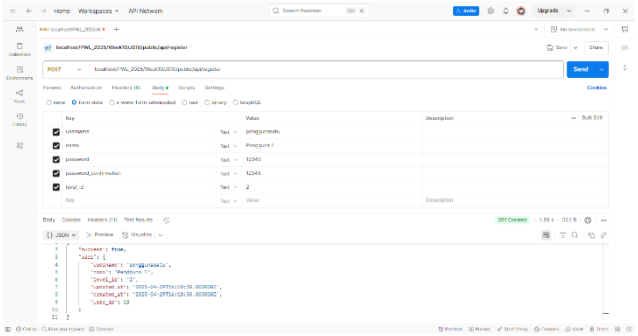


12. Sekarang kita coba masukkan data. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data registrasi menggunakan nilai yang Anda inginkan.



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.

   ```
   C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/LoginController

     INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\LoginController.php] created successfully.
   ```

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

JS10 > app > Http > Controllers > Api > LoginController.php > PHP > LoginController

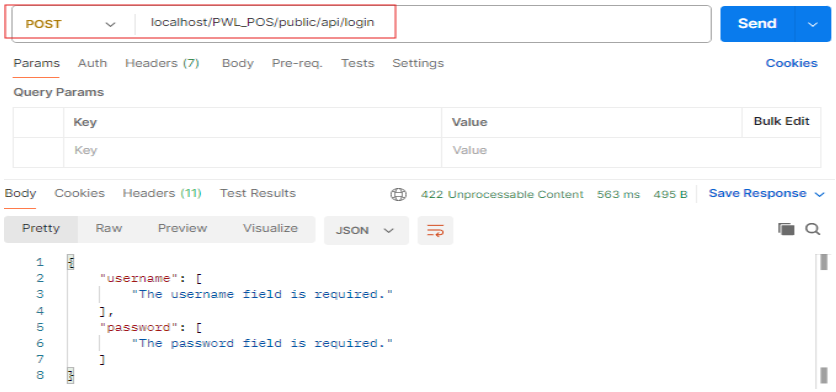
```php
<?php

namespace App\Http\Controllers\Api;

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

0 references | 0 implementations
class LoginController extends Controller
{
    0 references | 0 overrides
    public function __invoke(Request $request): JsonResponse|mixed{

        //set validation
        $validator = Validator::make(data: $request->all(), rules: [
            'username' => 'required',
            'password' => 'required'
        ]);

        //if validation fails
        if ($validator->fails()) {
            return response()->json(data: $validator->errors(), status: 422);
        }

        //get credentials from request
        $credentials = $request->only(keys: 'username', 'password');

        //if auth failed
        if (!$token = auth()->guard(name: 'api')->attempt(credentials: $credentials)) {
            return response()->json(data: [
                'success' => false,
                'message' => 'Username atau Password Anda salah'
            ], status: 401);
        }

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

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

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

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

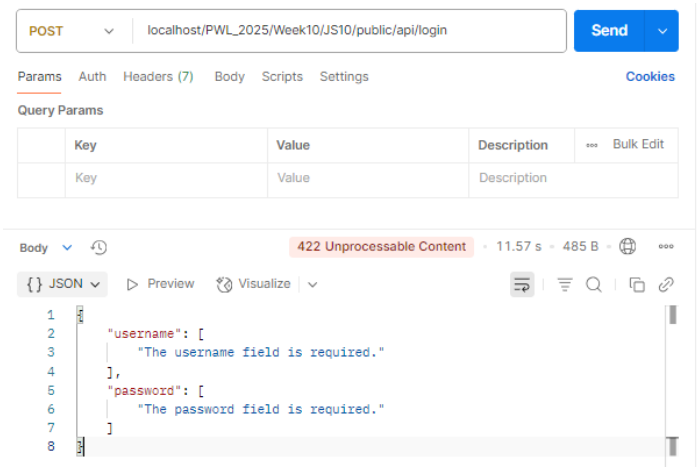
```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\LoginController;

/*
|--------------------------------------------------------------------------
| 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(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): mixed{
    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.

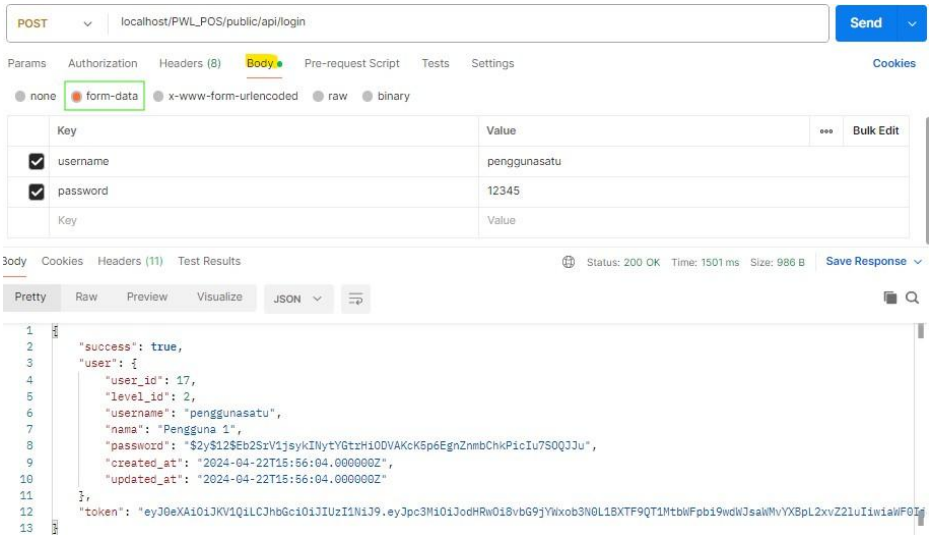


Jika berhasil akan muncul error validasi seperti gambar di atas.

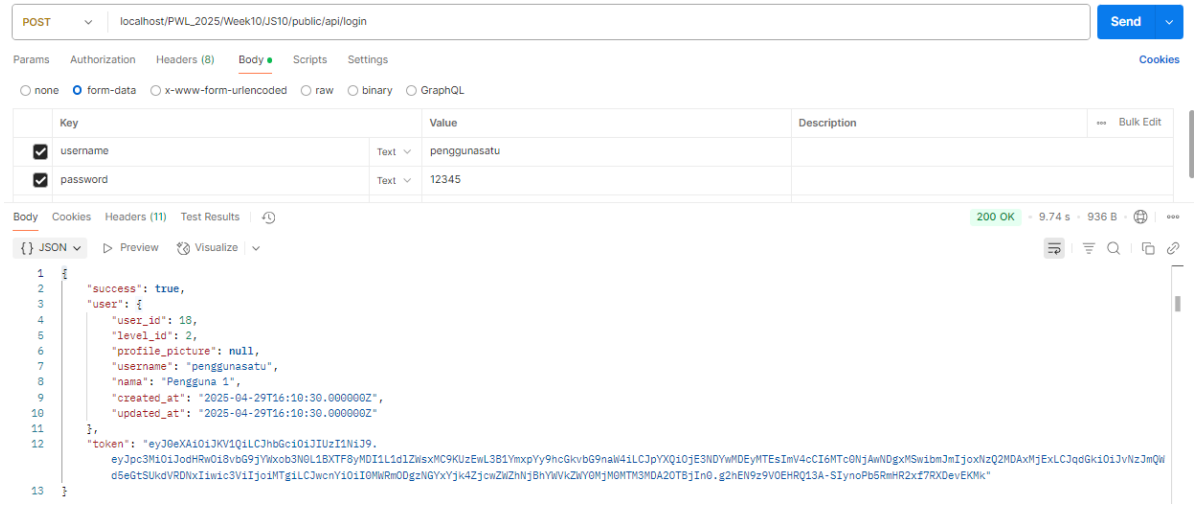
Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



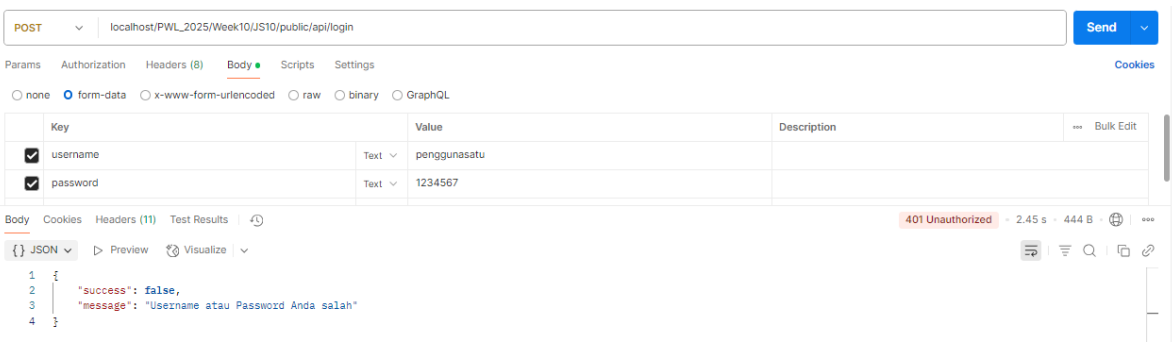
5. Selanjutnya, isikan username dan password sesuai dengan data user yang ada pada database. Klik tab Body dan pilih form-data. Isikan key sesuai dengan kolom data, serta isikan data user. Klik tombol Send, jika berhasil maka akan keluar tampilan seperti berikut. Copy nilai token yang diperoleh pada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



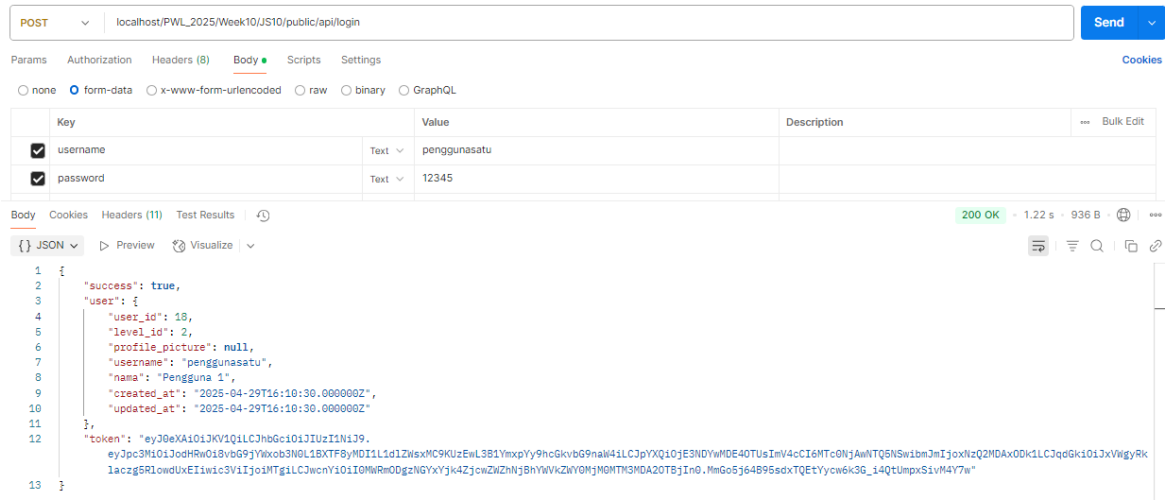
Lakukan percobaan yang sama dan berikan screenshoot hasil percobaan Anda.



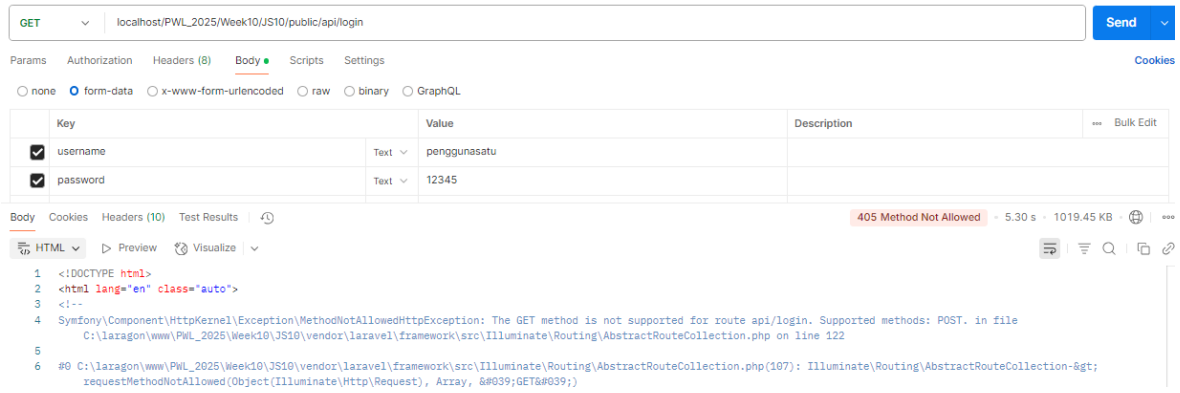
Lakukan percobaan yang untuk data yang salah dan berikan screenshoot hasil percobaan Anda.



6. 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.



- Login dengan method GET



Login harus pakai method POST, karena POST untuk mengirim data yang aman dan bersifat perubahan (seperti autentikasi), sedangkan GET hanya untuk mengambil/melihat data.

7. Lakukan commit perubahan file pada Github.

Praktikum 3 – Membuat RESTful API Logout

1. Tambahkan kode berikut pada file .env

   `JWT_SHOW_BLACKLIST_EXCEPTION=true`

```
62
63  JWT_SHOW_BLACKLIST_EXCEPTION=true
64
```

2. Buat Controller baru dengan nama LogoutController.

   `php artisan make:controller Api/LogoutController`

```
C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/LogoutController

  INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\LogoutController.php] created successfully.
```

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

JS10 > app > Http > Controllers > Api > LogoutController.php > PHP > LogoutController

```php
<?php

namespace App\Http\Controllers\Api;

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

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

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

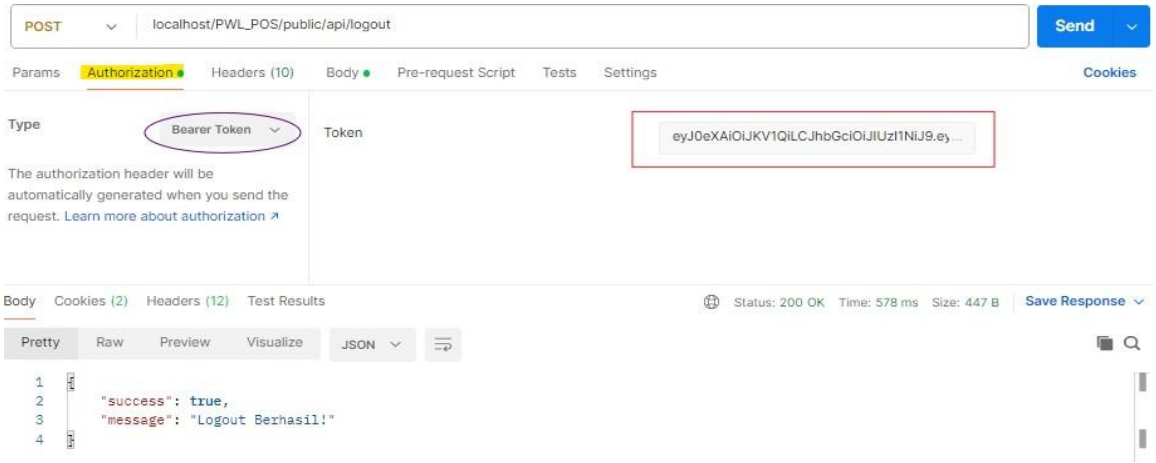
4. Lalu kita tambahkan routes pada api.php

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

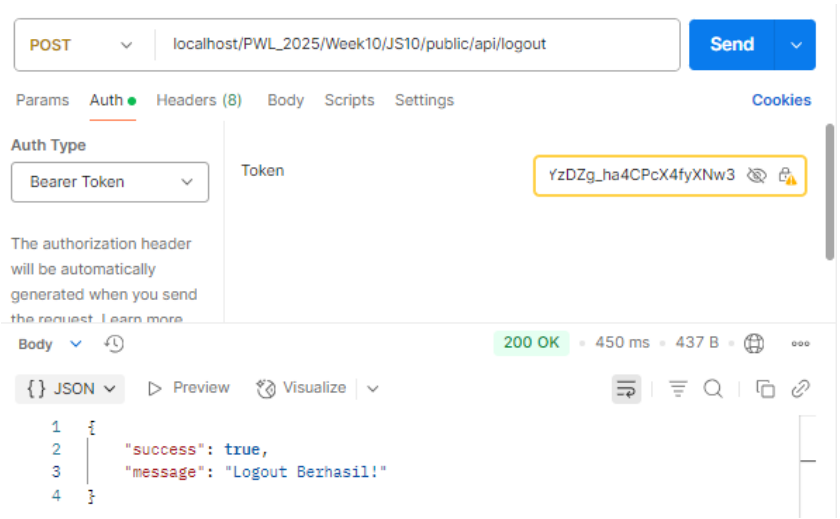
```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\LoginController;
use App\Http\Controllers\Api\LogoutController;

/*
|--------------------------------------------------------------------------
| 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(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): mixed{
    return $request->user();
});
Route::post(uri: '/logout', action: 'App\Http\Controllers\Api\LogoutController')->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 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
   ```

   ```
   C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/LevelController

     INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\LevelController.php] created successfully.
   ```

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

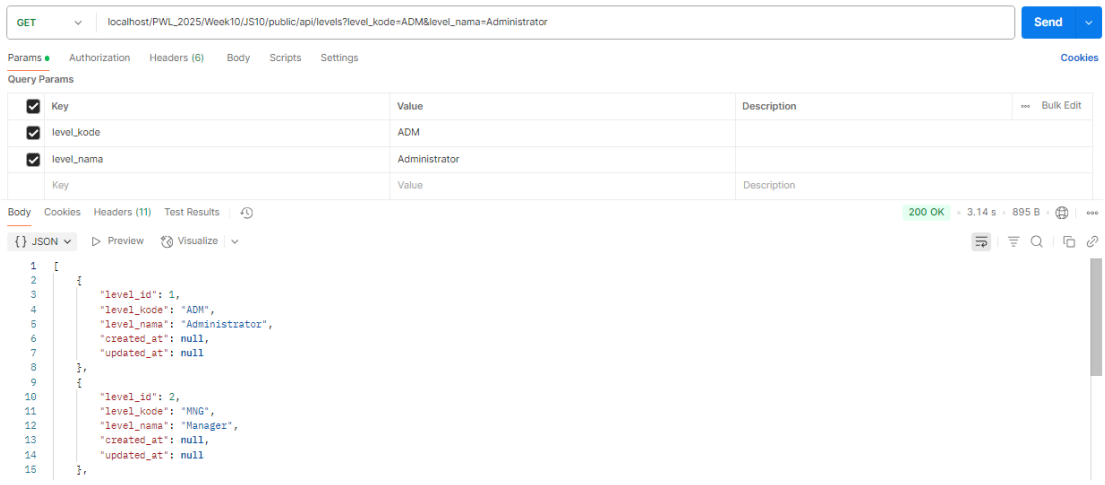
JS10 > app > Http > Controllers > Api > LevelController.php > PHP > LevelController

```php
<?php

namespace App\Http\Controllers\Api;

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

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

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

    0 references | 0 overrides
    public function show(LevelModel $level): LevelModel
    {
        return $level;
    }

    0 references | 0 overrides
    public function update(Request $request, LevelModel $level): LevelModel
    {
        $level->update(attributes: $request->all());
        return $level;
    }

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

3. Kemudian kita lengkapi routes pada api.php.

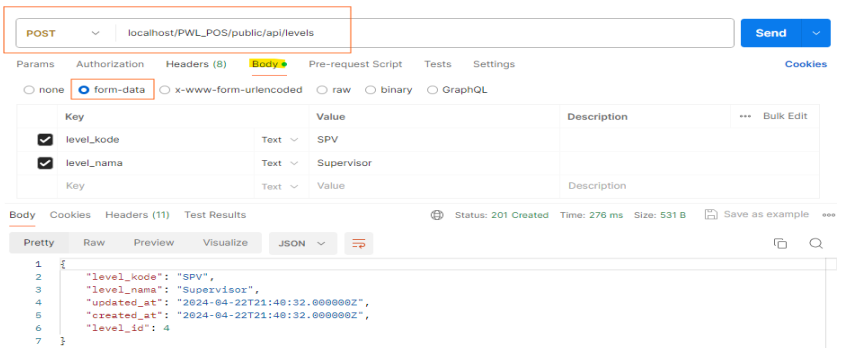
```php
use App\Http\Controllers\Api\LevelController;

/*
|--------------------------------------------------------------------------
| 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::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): mixed{
    return $request->user();
});
Route::post(uri: '/logout', action: 'App\Http\Controllers\Api\LogoutController')->name(name: 'logout');

// Route Level
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 screenshoot hasil percobaan Anda.

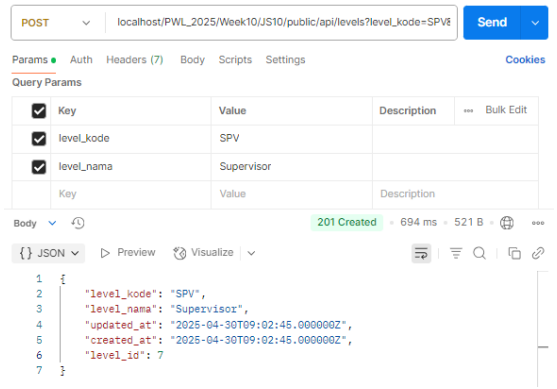


Method GET digunakan untuk mengambil atau menampilkan data dari server.

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

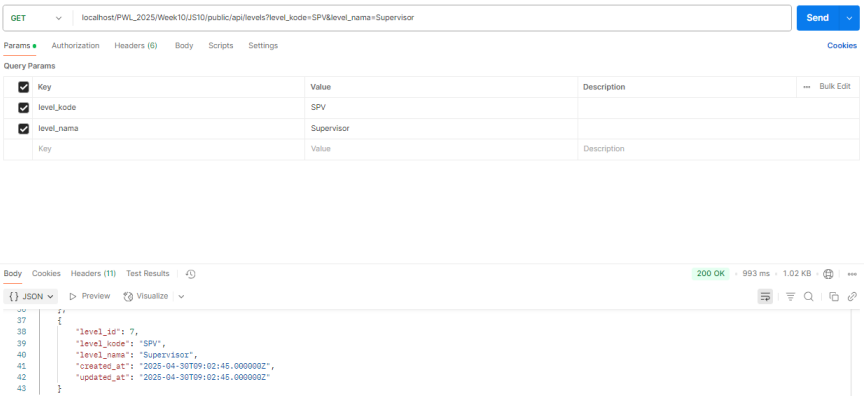


Jelaskan dan berikan screenshoot hasil percobaan Anda.



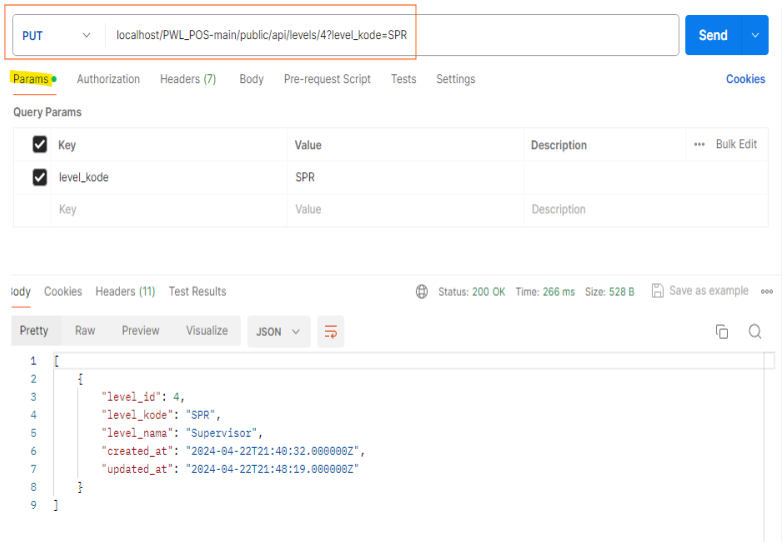
Method POST digunakan untuk mengirim data ke server.

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

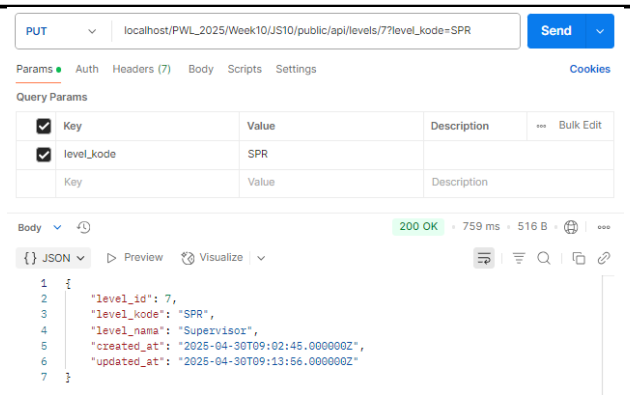


GET untuk menampilkan detail data.

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

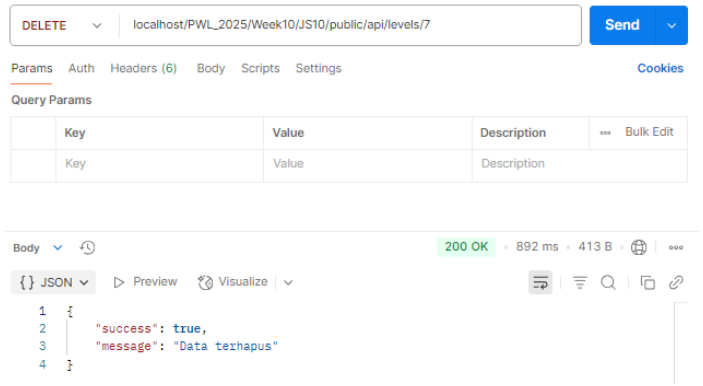


Jelaskan dan berikan screenshoot hasil percobaan Anda.



Method **PUT** digunakan saat kita ingin mengupdate data berdasarkan ID.

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



Method **DELETE** dipakai ketika kita ingin menghapus suatu data dari server, berdasarkan ID.

9. Lakukan commit perubahan file pada Github.

TUGAS

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

1. **Implementasi CRUD API pada tabel m_user**
   - Pertama, buat controller untuk mengolah API pada data user.
     `php artisan make:controller Api/UserController`

```
C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/UserController

  INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\UserController.php]
created successfully.
```

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

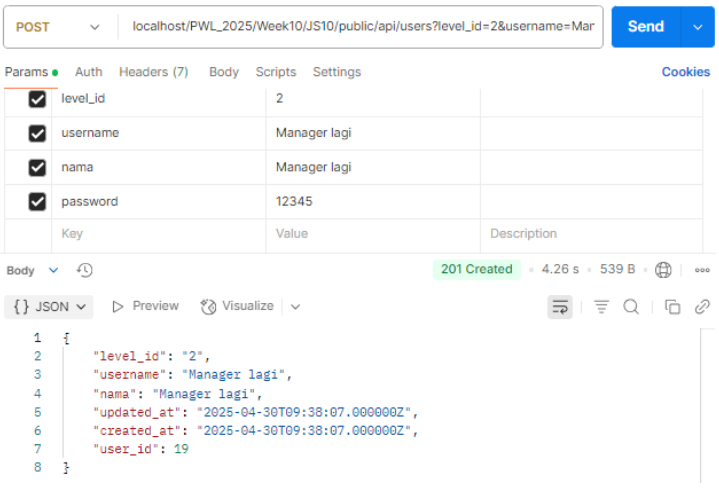
```php
<?php

namespace App\Http\Controllers\Api;

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

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

    public function store(Request $request): JsonResponse|mixed
    {
        $user = UserModel::create(attributes: $request->all());
        return response()->json(data: $user, status: 201);
    }

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

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

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

- Kemudian kita lengkapi routes pada api.php.

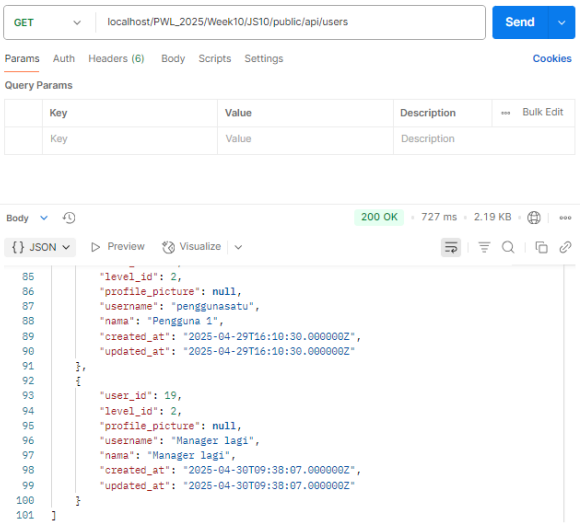
```php
use App\Http\Controllers\Api\UserController;
```

```php
// Route User
Route::get(uri: '/users', action: [UserController::class, 'index']);
Route::post(uri: '/users', action: [UserController::class, 'store']);
Route::get(uri: '/users/{user}', action: [UserController::class, 'show']);
Route::put(uri: '/users/{user}', action: [UserController::class, 'update']);
Route::delete(uri: '/users/{user}', action: [UserController::class, 'destroy']);
```

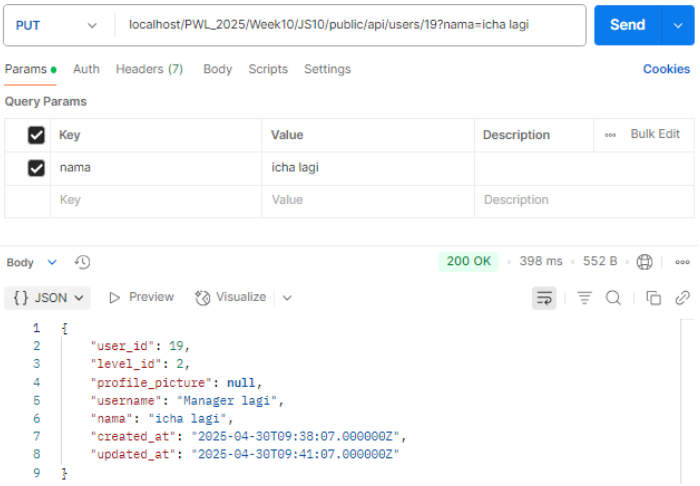
- Jika sudah. Lakukan uji coba API
  - POST



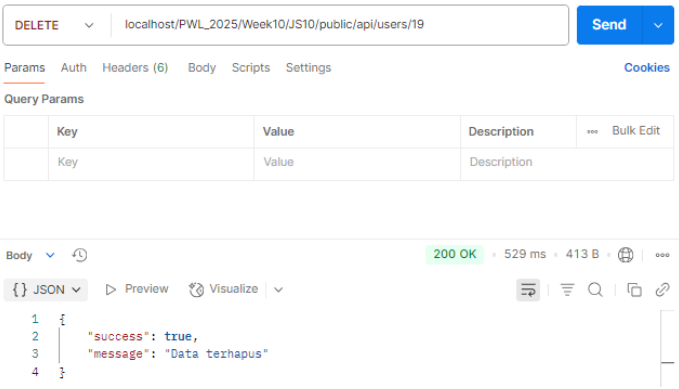
- GET



- PUT



- DELETE



2. Implementasi CRUD API pada tabel m_kategori

- Pertama, buat controller untuk mengolah API pada data kategori.

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

```
C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/KategoriController

  INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\KategoriController.php] created successfully.
```

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

JS10 > app > Http > Controllers > Api > KategoriController.php > PHP > KategoriController

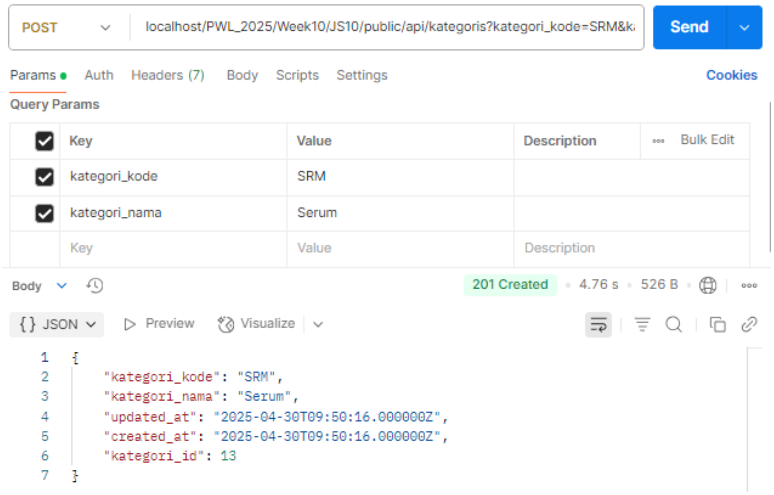
```php
<?php

namespace App\Http\Controllers\Api;

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

0 references | 0 implementations
class KategoriController extends Controller
{
    0 references | 0 overrides
    public function index(): Collection
    {
        return KategoriModel::all();
    }

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

    0 references | 0 overrides
    public function show(KategoriModel $kategori): KategoriModel
    {
        return $kategori;
    }

    0 references | 0 overrides
    public function update(Request $request, KategoriModel $kategori): KategoriModel
    {
        $kategori->update(attributes: $request->all());
        return $kategori;
    }

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

- Kemudian kita lengkapi routes pada api.php.

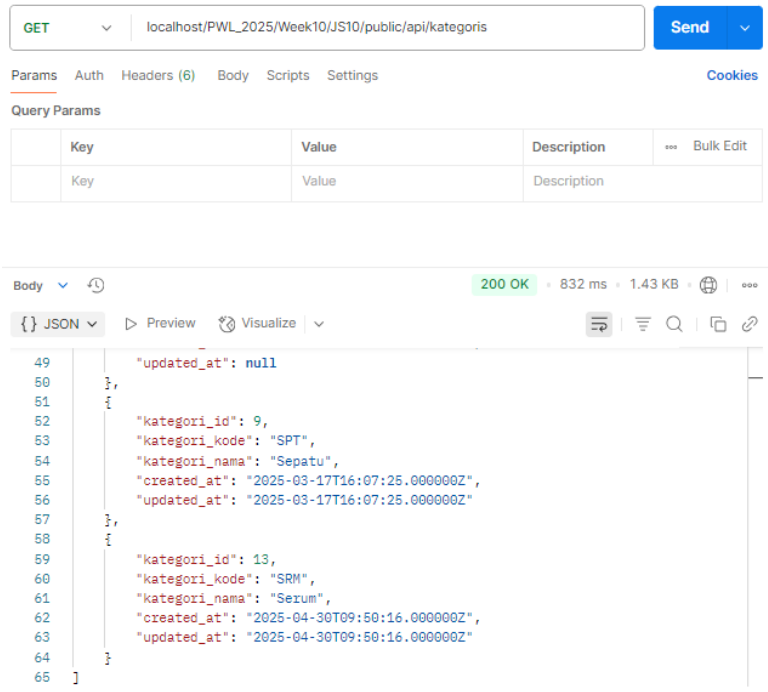
```php
use App\Http\Controllers\Api\KategoriController;
```

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

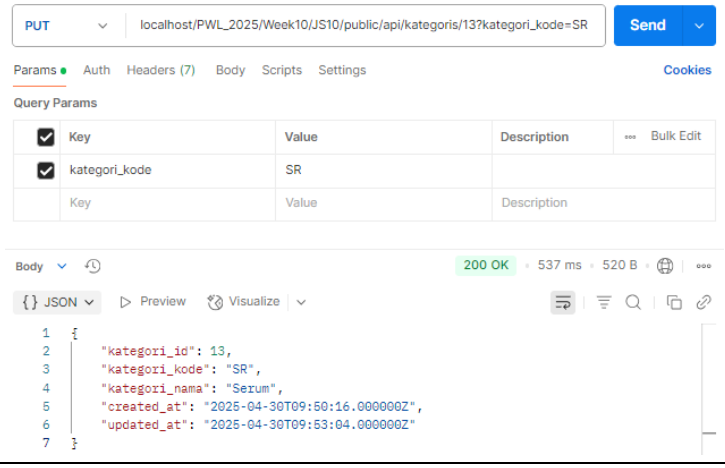
- Jika sudah. Lakukan uji coba API
  - POST


POST
localhost/PWL_2025/Week10/JS10/public/api/kategoris?kategori_kode=SRM&k
Send
Params
Auth
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
kategori_kode
SRM
kategori_nama
Serum
Body
201 Created
4.76 s
526 B
JSON
Preview
Visualize
{
"kategori_kode": "SRM",
"kategori_nama": "Serum",
"updated_at": "2025-04-30T09:50:16.000000Z",
"created_at": "2025-04-30T09:50:16.000000Z",
"kategori_id": 13
}

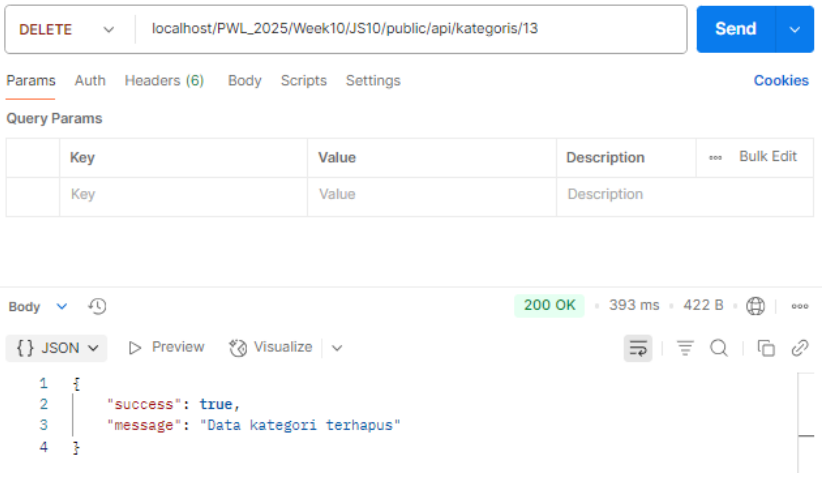
- GET


GET
localhost/PWL_2025/Week10/JS10/public/api/kategoris
Send
Params
Auth
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
200 OK
832 ms
1.43 KB
JSON
Preview
Visualize
"updated_at": null
},
{
"kategori_id": 9,
"kategori_kode": "SPT",
"kategori_nama": "Sepatu",
"created_at": "2025-03-17T16:07:25.000000Z",
"updated_at": "2025-03-17T16:07:25.000000Z"
},
{
"kategori_id": 13,
"kategori_kode": "SRM",
"kategori_nama": "Serum",
"created_at": "2025-04-30T09:50:16.000000Z",
"updated_at": "2025-04-30T09:50:16.000000Z"
}
]

- PUT


PUT
localhost/PWL_2025/Week10/JS10/public/api/kategoris/13?kategori_kode=SR
Send
Params
Auth
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
kategori_kode
SR
Body
200 OK
537 ms
520 B
JSON
Preview
Visualize
{
"kategori_id": 13,
"kategori_kode": "SR",
"kategori_nama": "Serum",
"created_at": "2025-04-30T09:50:16.000000Z",
"updated_at": "2025-04-30T09:53:04.000000Z"
}

- DELETE



3. Implementasi CRUD API pada tabel m_barang

- Pertama, buat controller untuk mengolah API pada data barang.

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

```
C:\laragon\www\PWL_2025\Week10\JS10>php artisan make:controller Api/BarangController

  INFO  Controller [C:\laragon\www\PWL_2025\Week10\JS10\app\Http\Controllers\Api\BarangController.php
] created successfully.
```

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

JS10 > app > Http > Controllers > Api > BarangController.php > PHP > BarangController

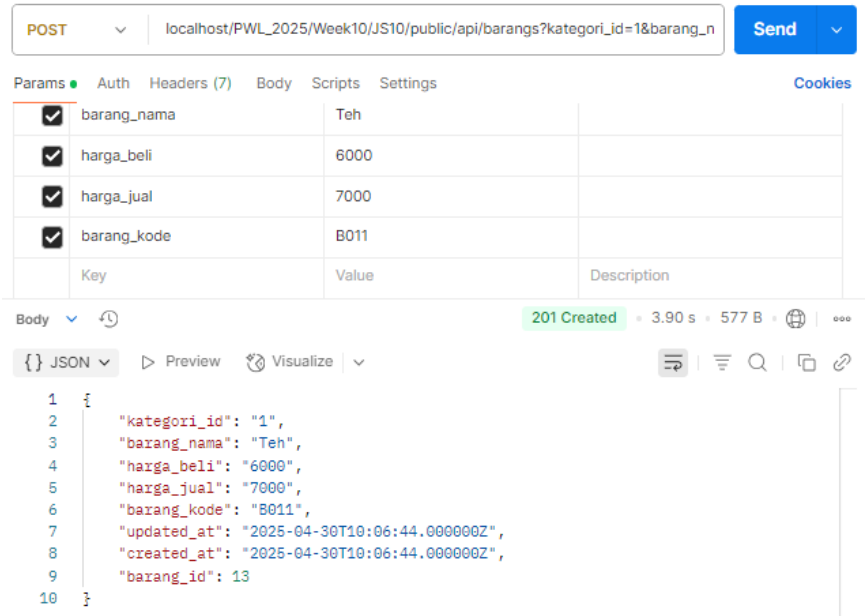
```php
<?php

namespace App\Http\Controllers\Api;

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

6 references | 0 implementations
class BarangController extends Controller
{
    1 reference | 0 overrides
    public function index(): Collection
    {
        return BarangModel::all();
    }

    1 reference | 0 overrides
    public function store(Request $request): JsonResponse|mixed
    {
        $barang = BarangModel::create(attributes: $request->all());
        return response()->json(data: $barang, status: 201);
    }

    1 reference | 0 overrides
    public function show(BarangModel $barang): BarangModel
    {
        return $barang;
    }

    1 reference | 0 overrides
    public function update(Request $request, BarangModel $barang): BarangModel
    {
        $barang->update(attributes: $request->all());
        return $barang;
    }

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

- Kemudian kita lengkapi routes pada api.php.

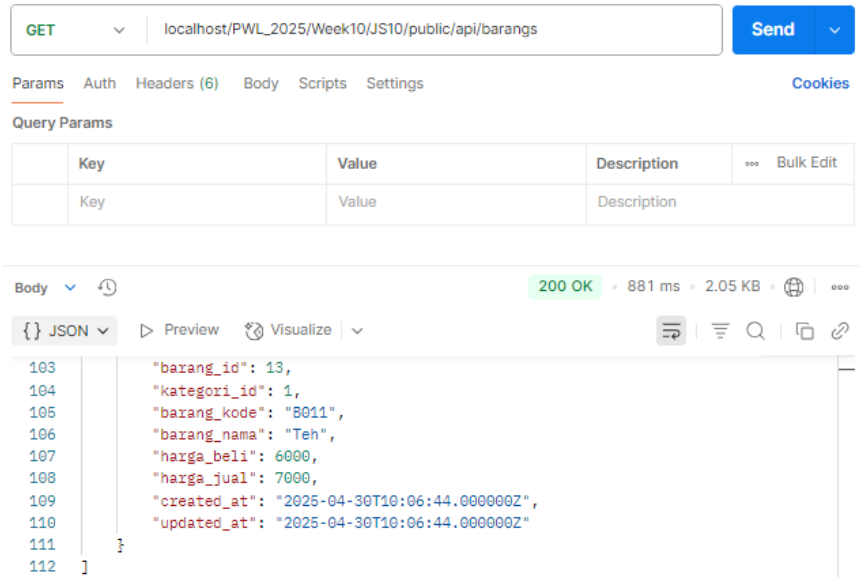
```php
use App\Http\Controllers\Api\BarangController;
```

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

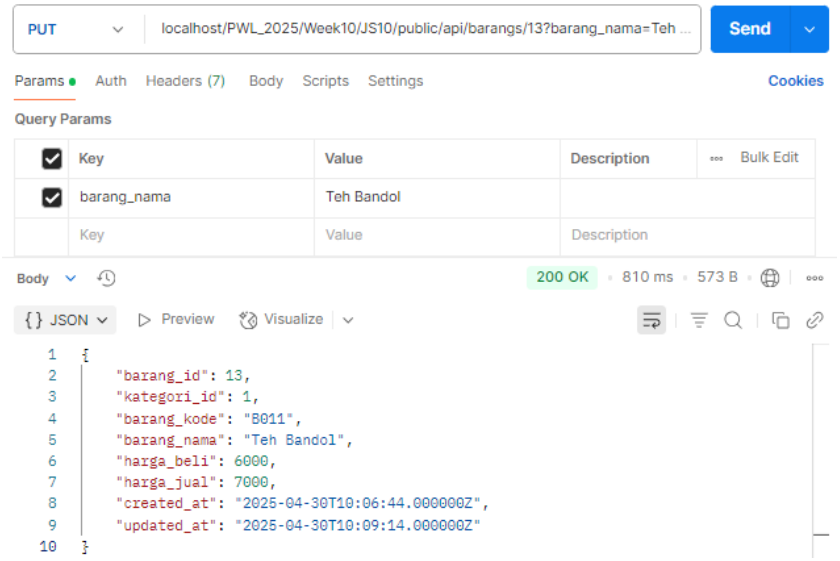
- Jika sudah. Lakukan uji coba API
  - POST



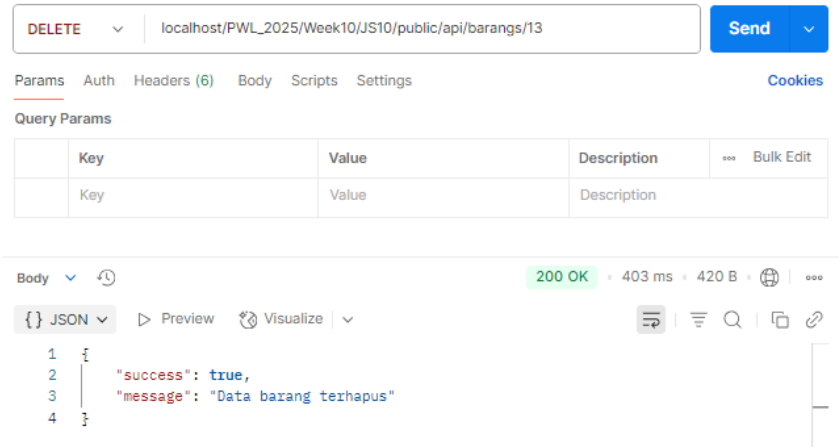
  - GET



- PUT



- DELETE



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