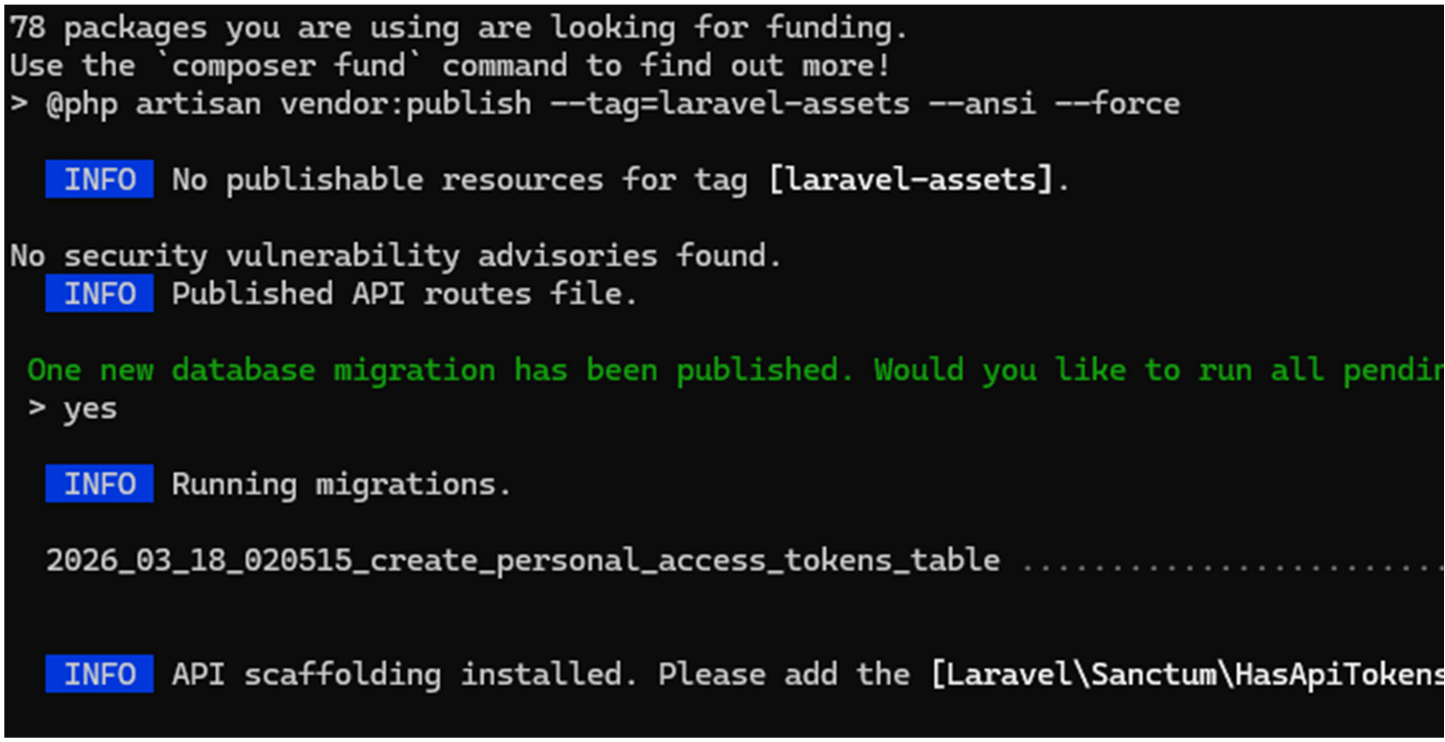
PKL-01
Laravel 13 REST API

Install Laravel API



Sejak **Laravel 11**, konfigurasi bawaan untuk API **tidak lagi otomatis** disertakan saat kita membuat proyek baru. Jadi, jika kita ingin menggunakan API di proyek Laravel, kita perlu menambahkannya secara manual.
Untuk itu, silakan jalankan perintah berikut di terminal:

```
php artisan install:api
```

Perintah ini akan menginstal modul-modul yang dibutuhkan untuk API, termasuk:

- File konfigurasi yang diperlukan,
- Middleware,
- Struktur rute untuk API.

Setelah dijalankan, terminal akan menampilkan proses seperti berikut:

```
./composer.json has been updated
Running composer update laravel/sanctum
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/sanctum (v4.3.1)
...
```

```
Generating optimized autoload files
> @php artisan package:discover --ansi
...
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
INFO  Published API routes file.
```

Pada akhir proses, Laravel biasanya akan menanyakan apakah Anda ingin menjalankan **migrasi tambahan**. Silakan ketik `yes` dan tekan *Enter*. Contohnya:

```
One new database migration has been published. Would you like to run all pending database migrations? (yes/no) [yes]:
> yes

INFO  Running migrations.
2026_03_18_020515_create_personal_access_tokens_table
.............................................. 55.22ms DONE
```

Migrasi ini akan membuat tabel yang dibutuhkan untuk API, misalnya **tabel tokens** jika kita menggunakan Laravel Sanctum untuk otentikasi API.
Terakhir, Laravel akan menampilkan informasi:

```
INFO  API scaffolding installed. Please add the [Laravel\Sanctum\HasApiTokens] trait to your User model.
```

Ingat, kita perlu menambahkan **trait** `HasApiTokens` pada model `User` agar fitur token API bisa digunakan.

Konfigurasi CSRF untuk API

Secara default, Laravel melindungi semua rute dengan **CSRF token**. Namun, untuk API, CSRF sering kali tidak diperlukan karena API biasanya diakses oleh **aplikasi eksternal** seperti frontend JavaScript atau aplikasi mobile.
Untuk mengecualikan token CSRF pada rute API, buka file `bootstrap/app.php` lalu tambahkan kode berikut:

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->PreventRequestForgery(except: [
        '/api/*',
    ]);
})
```

Kode ini memastikan semua rute yang diawali `/api/*` tidak akan dicek CSRF-nya, sehingga klien eksternal bisa melakukan request tanpa masalah.

Perubahan CSRF Middleware di Laravel 13

Pada Laravel 13, **middleware CSRF** mengalami perubahan nama dan perilaku:

1. **Nama Middleware Baru** Middleware sebelumnya dikenal sebagai:
    1. `VerifyCsrfToken`
    2. `ValidateCsrfToken` (alias lama)

    Sekarang **diganti menjadi**:

    ```
    PreventRequestForgery
    ```

Alias lama tetap ada untuk kompatibilitas, tapi sudah **deprecated**, jadi sebaiknya **jangan lagi digunakan** di kode baru.

2. **Fitur Baru: Request-Origin Verification** Laravel 13 menambahkan **cek origin request** menggunakan header:

```
Sec-Fetch-Site
```

Artinya, Laravel bisa memverifikasi apakah request berasal dari **sumber yang sama** atau bukan, sehingga perlindungan CSRF lebih kuat.

3. **Dampak pada Rute dan Test** Jika kalian sebelumnya menulis:

```
Route::middleware('VerifyCsrfToken')->group(function() {
    // rute
});
```

Maka di Laravel 13 sebaiknya **diubah** menjadi:

```
Route::middleware('PreventRequestForgery')->group(function() {
    // rute
});
```

Begitu juga pada **test**, jika ingin mengecualikan CSRF, gunakan `PreventRequestForgery` alih-alih `VerifyCsrfToken`.

Setelah itu, file `app.php` secara keseluruhan akan terlihat seperti ini:

```
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->PreventRequestForgery(except: [
            '/api/*',
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

Dengan begitu, API kita siap untuk digunakan.

Pada materi berikutnya, kita akan mulai membuat **Auth Controller** untuk mengelola registrasi, login, dan token API.

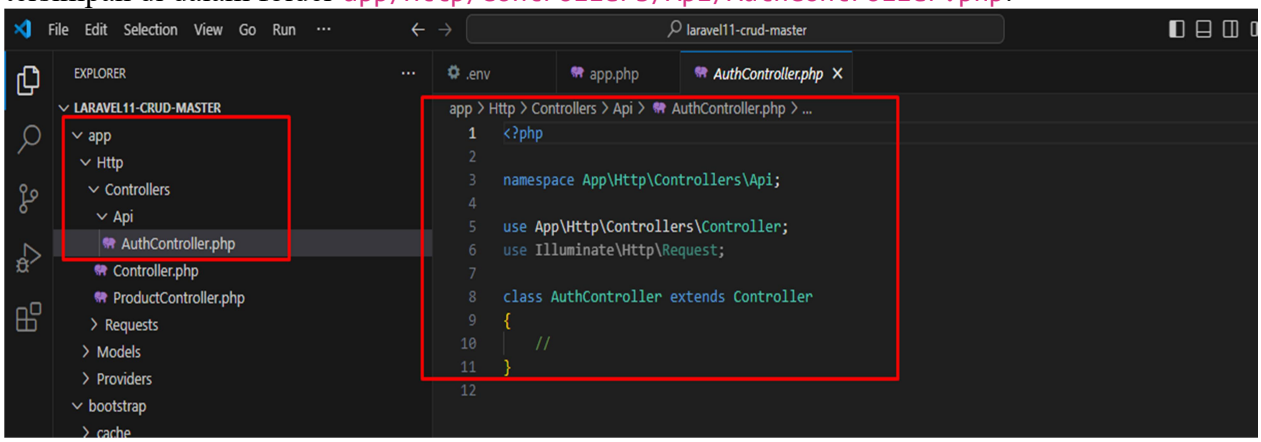
Membuat API Register & Validasi Input

Create Auth Controller

Pada materi kali ini, kita akan membuat sebuah API *controller* yang bertujuan untuk menangani proses *register*, *login*, dan *logout*.
Silakan masukkan *command line* berikut ini di dalam proyek Laravel Kita:

```
php artisan make:controller Api\AuthController
```

Perintah di atas akan menghasilkan file *controller* baru dengan nama `AuthController` yang tersimpan di dalam folder `app/Http/Controllers/Api/AuthController.php`.



Function Register

Mari kita buat *function* untuk menangani proses *register* pengguna. Silakan edit file *controller* yang baru saja dibuat sehingga akan terlihat seperti berikut:

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\User;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules\Password;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        // Validasi input
        $validator = Validator::make($request->all(), [
```

```
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users,email',
            'password' => ['required', 'confirmed', Password::defaults()],
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'errors' => $validator->errors(),
                'message' => 'Registration failed. Please check your input.',
            ], 422);
        }

        // Buat user baru
        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        // Login user dan buat token
        Auth::login($user);
        $responseData = [
            'token' => $user->createToken($user->name)->plainTextToken,
            'name' => $user->name,
        ];

        // Kirim respons sukses
        return response()->json([
            'success' => true,
            'data' => $responseData,
            'message' => 'User successfully registered.',
        ]);
    }
}
```

1. **Deklarasi Fungsi**

```
public function register(Request $request)
```

- Fungsi `register` ini adalah **method publik** di sebuah controller Laravel.
- Menerima satu parameter, yaitu `$request` yang merupakan instance dari **Illuminate\Http\Request**.
- `$request` berisi semua data yang dikirimkan oleh client (misalnya dari form registrasi).

---

2. **Validasi Input**

```
$validator = Validator::make($request->all(), [
```

```
    'name' => 'required|string|max:255',
    'email' => 'required|string|email|max:255|unique:users,email',
    'password' => ['required', 'confirmed', Password::defaults()],
]);
```

- Membuat objek validator untuk memeriksa data input user.
- `$request->all()` → mengambil semua input dari request.
- Aturan validasi:
    - `name`: wajib diisi (`required`), tipe string (`string`), maksimal 255 karakter.
    - `email`: wajib, string, format email, max 255 karakter, harus unik di kolom `email` tabel `users`.
    - `password`: wajib, harus dikonfirmasi (`confirmed` → memerlukan input `password_confirmation`), sesuai aturan default Laravel (`Password::defaults()`).

---

3. **Cek Hasil Validasi**

```
if ($validator->fails()) {
    return response()->json([
        'success' => false,
        'errors' => $validator->errors(),
        'message' => 'Registration failed. Please check your input.',
    ], 422);
}
```

- Jika validasi gagal:
    - `fails()` mengembalikan `true`.
    - Mengirimkan **JSON response**:
        - `success`: false → menandakan gagal.
        - `errors`: daftar error per field.
        - `message`: pesan umum untuk client.
    - Status HTTP **422 Unprocessable Entity** → artinya data yang dikirim tidak valid.

---

4. **Membuat User Baru**

```
$user = User::create([
    'name' => $request->name,
    'email' => $request->email,
    'password' => Hash::make($request->password),
]);
```

- `User::create(...)` → menyimpan data baru ke tabel `users`.
- `Hash::make(...)` → mengenkripsi password sebelum disimpan, agar **tidak disimpan dalam bentuk plain text** (praktik keamanan wajib).
- `$user` sekarang adalah instance user yang baru saja dibuat.

---

5. **Login dan Buat Token**

```
Auth::login($user);
$responseData = [
    'token' => $user->createToken($user->name)->plainTextToken,
    'name' => $user->name,
];
```

- `Auth::login($user)` → otomatis login user baru.
- `createToken($user->name)` → membuat **API token** (menggunakan Laravel Sanctum).
- `plainTextToken` → token yang dikirim ke client untuk autentikasi selanjutnya.
- `responseData` berisi data yang akan dikirim balik ke client: `token` + `name`.

---

6. **Kirim Respons Sukses**

```
return response()->json([
    'success' => true,
    'data' => $responseData,
    'message' => 'User successfully registered.',
]);
```

- Mengirimkan **JSON response** ke client:
  - `success`: true → menandakan registrasi berhasil.
  - `data`: token dan nama user.
  - `message`: pesan sukses.

Ringkasan Alur

1. Terima request registrasi.
2. Validasi input.
3. Jika gagal → kirim error 422.
4. Jika valid → buat user baru, hash password.
5. Login user dan buat token API.
6. Kirim JSON response sukses.

Model User

Silakan buka file `User.php` di dalam folder `app/Models`, kemudian tambahkan baris berikut ini:

```
use Laravel\Sanctum\HasApiTokens;
HasApiTokens
```

Dan tambahkan *trait* `HasApiTokens` ke dalam kelas `User`. Sehingga secara keseluruhan akan terlihat seperti berikut:

```
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Database\Factories\UserFactory;
use Illuminate\Database\Eloquent\Attributes\Fillable;
```

```
use Illuminate\Database\Eloquent\Attributes\Hidden;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Sanctum\HasApiTokens;
use Illuminate\Notifications\Notifiable;

#[Fillable(['name', 'email', 'password'])]
#[Hidden(['password', 'remember_token'])]
class User extends Authenticatable
{
    /** @use HasFactory<UserFactory> */
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
}
```

Set Register Route

Silakan buka file `api.php` yang terletak di dalam folder `routes`. Tambahkan rute *register* berikut ini:

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

Route::post('register', [AuthController::class, 'register']);
```

Sehingga, secara keseluruhan, file `api.php` akan terlihat seperti berikut:

```
<?php

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', [AuthController::class, 'getUser']);
});


Route::post('register', [AuthController::class, 'register']);
```

Cek Rute

Untuk memastikan bahwa rute *register* sudah terdaftar di dalam proyek kita, silakan masukkan dan jalankan *command line* berikut ini di CMD:

```
php artisan route:list
```

```
  GET|HEAD        /
  POST            api/register
  GET|HEAD        api/user
  GET|HEAD        images
  POST            images
  GET|HEAD        images/create
  GET|HEAD        images/{image}
  PUT|PATCH       images/{image}
  DELETE          images/{image}
  GET|HEAD        images/{image}/edit
  GET|HEAD        sanctum/csrf-cookie ..... sanctum.csrf-cookie › Larav
  GET|HEAD        storage/{path} storage.local › vendor/laravel/framework/src/Il
  PUT             storage/{path} storage.local.upload › vendor/laravel/framework
  GET|HEAD        up ..... vendor/laravel/framework/src/Illuminate/Foundation/Co
```

Terlihat bahwa rute *register* sudah terdaftar. Pada materi berikutnya, kita akan melakukan uji coba untuk membuat pengguna menggunakan API *register* yang baru saja kita buat.

Uji Coba API Register

Test API Register

Sebelum kita mencoba **API Register**, pastikan beberapa hal berikut sudah siap:

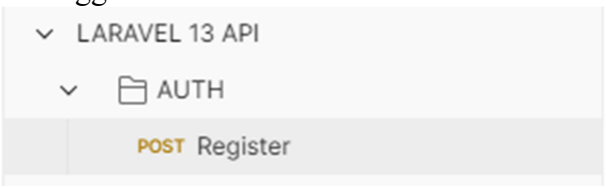
1. **Project Laravel berjalan** Jalankan project menggunakan:
   ```
   php artisan serve
   ```
2. **Server lokal sudah aktif** Pastikan MySQL dan Apache berjalan.
3. **Aplikasi Postman sudah terpasang** Jika belum, teman-teman bisa mengunduhnya di sini:
   ```
   https://www.postman.com/downloads/
   ```

Membuat Collection Baru

Langkah pertama di Postman adalah membuat **collection** untuk mengelompokkan semua request API kita.

1. Buat **collection baru** dengan nama:
   ```
   LARAVEL13-RESTAPI
   ```
2. Di dalam collection tersebut, buat **subfolder** bernama:
   ```
   AUTH
   ```
3. Tambahkan **request baru** di folder `AUTH` dengan nama:
   ```
   register
   ```

Sehingga struktur folder di Postman akan terlihat seperti ini:

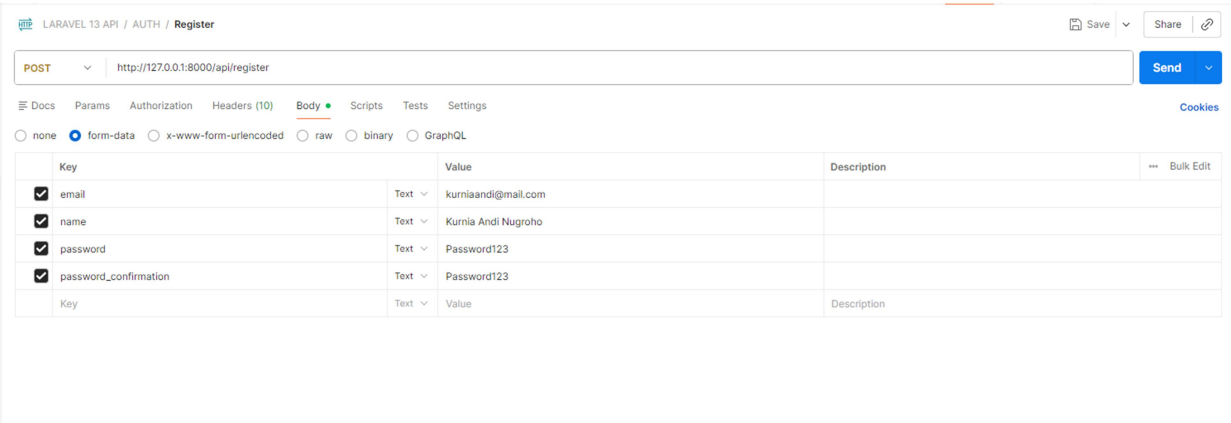


Konfigurasi Request Register

Untuk request `register`:

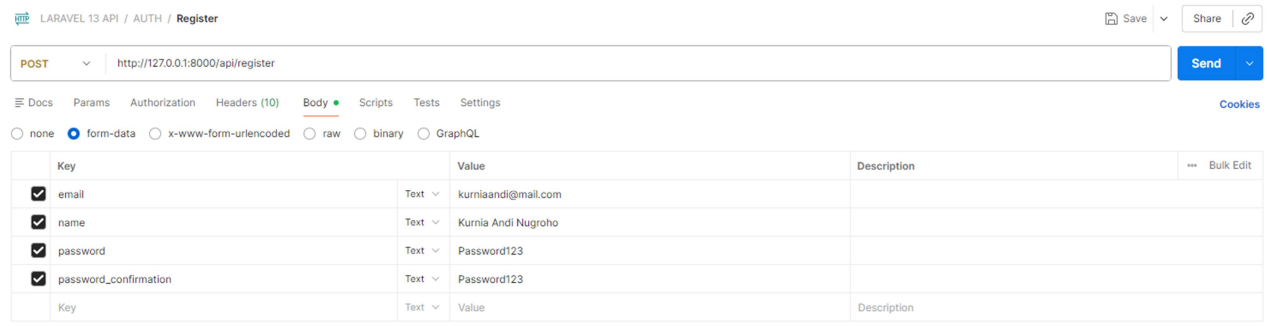
1. Masukkan URL endpoint API Register:
   ```
   http://127.0.0.1:8000/api/register
   ```
2. Pilih metode:
   ```
   POST
   ```
3. Pada bagian **Body → form-data**, tambahkan field yang diperlukan:
   1. `name`
   2. `email`
   3. `password`
   4. `password_confirmation`

Contohnya seperti pada gambar berikut:



Mengirim Request

Setelah semua field diisi, klik tombol **Send** di Postman.
Jika berhasil, kita akan menerima **response JSON** seperti ini:



Dari response tersebut, kita bisa melihat beberapa informasi penting:

- `success` → menandakan status proses register
- `message` → keterangan proses
- `data` → berisi `token` dan `name` pengguna yang baru terdaftar

Artinya, **endpoint API Register sudah berhasil dibuat dan dapat menerima request**.

Mendapatkan Info User Login & Testing API

Function Get User

Pembahasan kali ini, kita akan membuat sebuah *function* yang bertujuan untuk mendapatkan detail data *user* yang sedang login. Pada API *register* sebelumnya, ketika *user* berhasil melakukan proses registrasi, maka secara otomatis akan mendapatkan *token*. *Token* ini nantinya akan kita gunakan untuk mendapatkan detail data *user*. Silakan buka file **AuthController.php**, kemudian tambahkan *function* berikut ini:

```
public function getUser(Request $request)
    {
        return response()->json([
            'success' => true,
            'data' => $request->user(),
        ], 200);
    }
```

`$request->user()` Mengambil data *user* yang sedang terautentikasi berdasarkan *token* yang dikirimkan di *header*.

Sehingga secara keseluruhan, file **AuthController.php** saat ini akan tampak seperti berikut:

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\User;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules\Password;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        // Validasi input
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users,email',
            'password' => ['required', 'confirmed', Password::defaults()],
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation errors occurred.',
                'errors' => $validator->errors(),
```

```
            ], 422);
        }

        try {
            // Buat user baru
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
            ]);

            // Login user dan buat token
            Auth::login($user);
            $responseData = [
                'token' => $user->createToken($user->name)->plainTextToken,
                'name' => $user->name,
            ];

            // Kirim respons sukses
            return response()->json([
                'success' => true,
                'message' => 'User successfully registered.',
                'data' => $responseData,
            ], 201);
        } catch (\Exception $e) {
            // Tangani kesalahan server
            return response()->json([
                'success' => false,
                'message' => 'An error occurred during registration.',
                'errors' => ['server_error' => $e->getMessage()],
            ], 500);
        }
    }
    public function getUser(Request $request)
    {
        // Mengembalikan data pengguna yang sedang terotentikasi
        return response()->json([
            'success' => true,
            'data' => $request->user(),
        ], 200);
    }
}
```

Deklarasi Fungsi

```
public function getUser(Request $request)
```

- Fungsi `getUser` adalah **method publik** pada controller Laravel.
- Menerima parameter `$request` yang berisi objek **Illuminate\Http\Request**, termasuk informasi pengguna yang sedang login.
- Fungsi ini biasanya dipanggil oleh client untuk **mengambil data user yang sedang terotentikasi**.

2. **Mengembalikan Data User**

```
return response()->json([
    'success' => true,
```

```
    'data' => $request->user(),
], 200);
```

- `response()->json([...], 200)` → membuat **respon JSON** dengan status HTTP **200 OK**.
- Struktur JSON yang dikirim ke client:
    - `success: true` → menandakan request berhasil.
    - `data: $request->user()` → mengambil **user yang sedang login**, yaitu:
        - Laravel secara otomatis mengisi `$request->user()` dengan user dari token (jika menggunakan Sanctum atau Passport) atau session.
        - Berisi informasi user seperti `id`, `name`, `email`, dsb.

Ringkasan Alur

1. Client melakukan request ke endpoint `getUser`.
2. Laravel mengecek token atau session untuk mengetahui user yang login.
3. `$request->user()` mengembalikan objek user yang sedang terotentikasi.
4. Fungsi membungkus data ini ke JSON dan mengirim response dengan status 200.

Konfigurasi Rute Get User

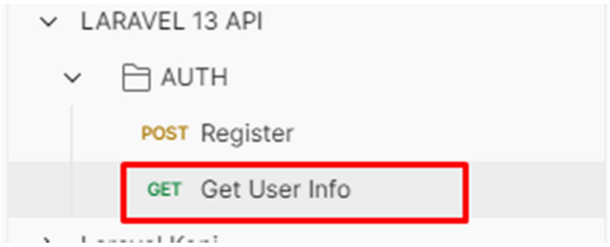
Silakan buka file rute yakni api.php, kemudian ubah menjadi seperti berikut ini:

```
<?php

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', [AuthController::class, 'getUser']);
});


Route::post('register', [AuthController::class, 'register']);
```

`Route::middleware('auth:sanctum')` Menentukan bahwa rute dalam grup ini hanya bisa diakses oleh *user* yang telah login dan memiliki *token* autentikasi (*middleware* `auth:sanctum`).

Test API

Endpoint	**Methode**
http://127.0.0.1:8000/api/user	`GET`



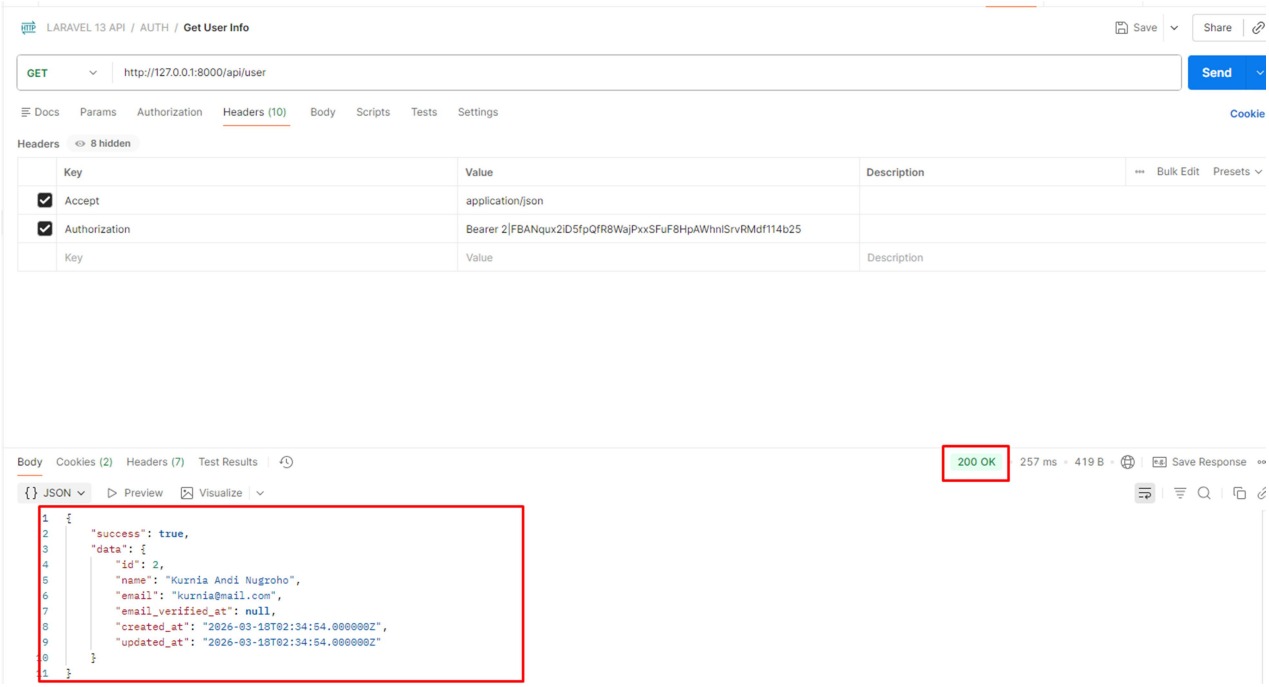
Silakan buat request baru dengan nama request `Get User Info` dengan menggunakan *methode* `GET` kemudian pada *header* silakan masukkan key dan value berikut ini:

Key	**Value**
Accept	application/json
Authorization	Bearer (TOKEN YANG DIDAPAT SETELAH REGISTER)

Catatan:

- *Gantilah (TOKEN YANG DIDAPAT SETELAH REGISTER) dengan token autentikasi yang Anda terima saat proses register berhasil.*
- *Dengan request ini, Kita akan mendapatkan detail data user yang sedang login dalam format JSON.*

Jika kita coba klik send maka akan muncul seperti berikut ini:



Terlihat kita sudah berhasil mendapatkan data user yang sudah teregistrasi dan mendapatkan token. pada materi berikutnya, kita akan menambahkan function Login dan Juga Log Out.

Logout API dengan Sanctum

Function Logout

Fungsi ini bertujuan untuk menghapus semua token yang terkait dengan user yang sedang login. Dengan demikian, saat mencoba mengakses rute yang dilindungi oleh middleware Sanctum, maka pengguna akan dianggap **unauthenticated**. Tambahkan fungsi berikut ke dalam **AuthController**:

```
public function logout(Request $request)
    {
        // Ambil pengguna yang sedang login
        $user = Auth::user();

        // Periksa apakah pengguna memiliki token
        if ($user->tokens->isEmpty()) {
            return response()->json([
                'success' => false,
                'message' => 'No active tokens found for this user.',
            ], 404);
        }

        // Hapus semua token yang terkait dengan pengguna
        $user->tokens->each(function ($token) {
            $token->delete();
        });

        // Kirim respons sukses
        return response()->json([
            'success' => true,
            'message' => 'Successfully logged out.',
        ], 200);
    }
```

1. **Deklarasi Fungsi**

```
public function logout(Request $request)
```

- Method `logout` bersifat **publik** pada controller.
- Parameter `$request` adalah objek **Illuminate\Http\Request**, berisi informasi request dari client.
- Fungsi ini digunakan untuk **menghapus token API user** sehingga mereka tidak bisa lagi melakukan request yang membutuhkan autentikasi.

---

2. **Ambil User yang Sedang Login**

```
$user = Auth::user();
```

- `Auth::user()` mengembalikan **user yang sedang terautentikasi** berdasarkan token atau session yang dipakai.
- `$user` sekarang berisi instance model `User` yang login.

---

3. **Cek Token yang Aktif**

```
if ($user->tokens->isEmpty()) {
    return response()->json([
        'success' => false,
        'message' => 'No active tokens found for this user.',
    ], 404);
}
```

- `$user->tokens` → relasi ke token API (Laravel Sanctum).
- `isEmpty()` → mengecek apakah user **tidak memiliki token aktif**.
- Jika kosong → kembalikan response JSON **404 Not Found**, karena tidak ada token yang bisa dihapus.
- Struktur JSON:
    - `success: false` → logout gagal.
    - `message` → informasi error.

---

4. **Hapus Semua Token**

```
$user->tokens->each(function ($token) {
    $token->delete();
});
```

- `$user->tokens->each(...)` → iterasi semua token yang terkait dengan user.
- `$token->delete()` → menghapus token dari database.
- Setelah ini, **user tidak lagi bisa menggunakan token lama untuk mengakses API**, artinya logout berhasil.

---

5. **Kirim Respons Sukses**

```
return response()->json([
    'success' => true,
    'message' => 'Successfully logged out.',
], 200);
```

- Mengirim JSON response ke client:
    - `success: true` → menandakan logout berhasil.
    - `message` → pesan sukses.
- Status HTTP **200 OK** → request berhasil.

Ringkasan Alur Logout

1. Ambil user yang sedang login.
2. Periksa apakah user punya token aktif.
    1. Jika tidak ada → kirim 404.

3. Hapus semua token aktif user.
4. Kirim response sukses.

Pada kode diatas terdapat kondisi, jika tidak ada token aktif, maka function logout akan menampilkan pesan response `No active tokens found for this user` dengan status code `404`. Apabila terdapat token aktif yang ditemukan maka akan muncul status code `200` dengan response `Successfully logged out`. Dengan kondisi ini, pengguna akan mendapatkan pesan yang lebih informatif jika mereka mencoba logout tanpa memiliki token aktif.

Konfigurasi Rute Logout

Silakan buka file rute yakni `api.php`, kemudian ubah menjadi seperti berikut ini:

```
<?php

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', [AuthController::class, 'getUser']);
    Route::post('/logout', [AuthController::class, 'logout']);
});


Route::post('register', [AuthController::class, 'register']);
```

Test API

Endpoint	**Methode**
http://127.0.0.1:8000/api/logout	`POST`

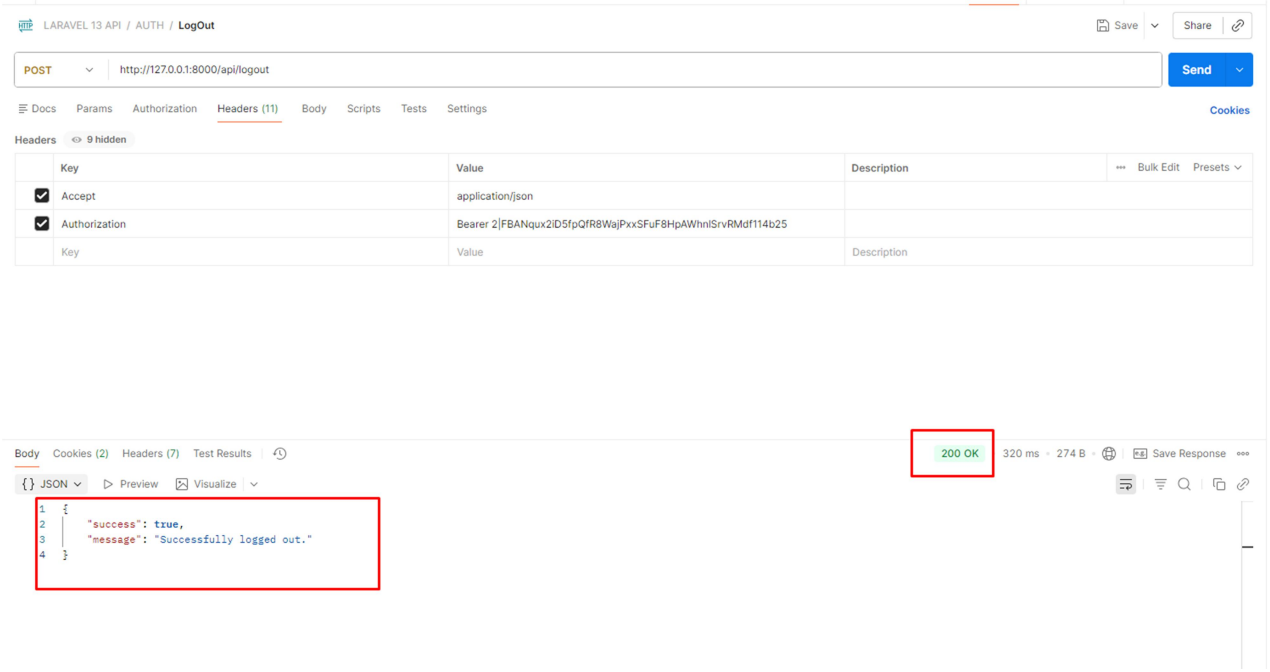
Silakan buat request baru dengan nama request `LogOut` dengan menggunakan *methode* `POST` . kemudian pada *header* silakan masukkan key dan value berikut ini:

Key	**Value**
Accept	application/json
Authorization	Bearer (TOKEN YANG DIDAPAT SETELAH REGISTER)

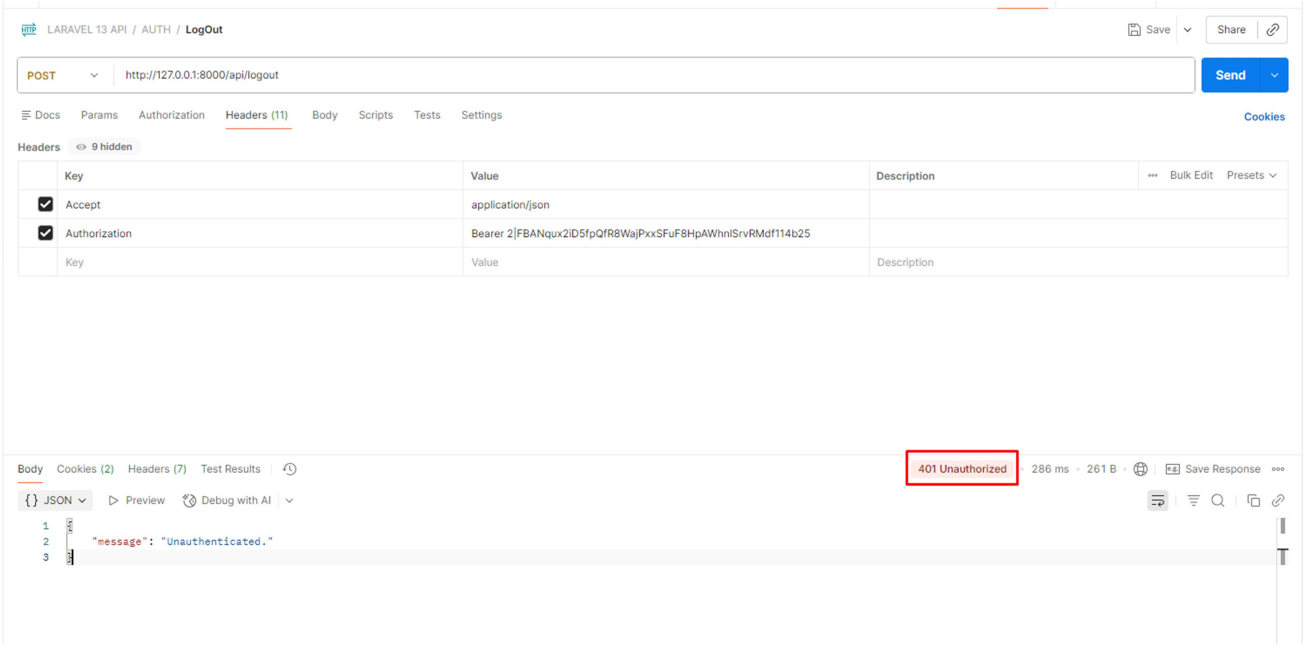
Catatan:

- *Gantilah (TOKEN YANG DIDAPAT SETELAH REGISTER) dengan token autentikasi yang Anda terima saat proses register berhasil.*

Lebih jelasnya bisa dilihat pada gambar berikut ini



Respon Ketika Tidak ada Token Aktif



Pada materi berikutnya, kita akan membuat satu fungsi lagi didalam AuthController, yakni fungsi Login.

Membuat API Login & Testing API

Function Login

Selain pada register, fungsi ini bertindak sebagai gerbang utama bagi pengguna untuk mendapatkan token. Token yang diberikan digunakan sebagai kunci autentikasi untuk mengakses URL yang memerlukan token sebagai bagian dari sistem keamanan API.

Penjelasan ini menekankan bahwa token berfungsi untuk autentikasi dan memberikan akses ke API yang dilindungi. Tambahkan fungsi login berikut ke dalam **AuthController**:

```
public function login(Request $request)
    {
        // Validasi input
        $validator = Validator::make($request->all(), [
            'email' => 'required|email',
            'password' => 'required',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation errors occurred.',
                'errors' => $validator->errors(),
            ], 422);
        }

        // Verifikasi kredensial
        if (Auth::attempt(['email' => $request->email, 'password' => $request->password])) {
            $user = Auth::user();
            $responseData = [
                'token' => $user->createToken($user->name)->plainTextToken,
                'name' => $user->name,
            ];

            // Kirim respons sukses
            return response()->json([
                'success' => true,
                'message' => 'Login successful.',
                'data' => $responseData,
            ], 200);
        }

        // Kredensial salah
        return response()->json([
            'success' => false,
            'message' => 'Invalid credentials.',
        ], 401);
    }
```

1. **Deklarasi Fungsi**

```
public function login(Request $request)
```

- Method `login` bersifat **publik** pada controller.
- Parameter `$request` berisi semua data yang dikirim client, seperti email dan password.

2. **Validasi Input**

```
$validator = Validator::make($request->all(), [
    'email' => 'required|email',
    'password' => 'required',
]);
```

- Membuat **validator** untuk memeriksa input user.
- Aturan validasi:
    - `email`: wajib (`required`) dan harus berformat email (`email`).
    - `password`: wajib diisi (`required`).

```
if ($validator->fails()) {
    return response()->json([
        'success' => false,
        'message' => 'Validation errors occurred.',
        'errors' => $validator->errors(),
    ], 422);
}
```

- Jika validasi gagal:
    - Kirim response JSON dengan status **422 Unprocessable Entity**.
    - `success: false` → menandakan gagal.
    - `errors` → daftar error per field.
    - `message` → pesan umum.

3. **Verifikasi Kredensial**

```
if (Auth::attempt(['email' => $request->email, 'password' => $request->password])) {
```

- `Auth::attempt(...)` → mencoba login user dengan **kombinasi email dan password**.
- Laravel otomatis membandingkan password dengan **hash yang tersimpan di database**.
- Jika kredensial valid → login berhasil.

```
$user = Auth::user();
$responseData = [
    'token' => $user->createToken($user->name)->plainTextToken,
    'name' => $user->name,
];
```

- `Auth::user()` → mengambil user yang baru saja login.
- `createToken($user->name)` → membuat **token API baru** (menggunakan Laravel Sanctum).
- `plainTextToken` → token dikirim ke client agar bisa digunakan di request selanjutnya.

```
return response()->json([
    'success' => true,
    'message' => 'Login successful.',
```

```
    'data' => $responseData,
], 200);
```

- Mengirim JSON sukses:
  - `success: true`
  - `message`: login berhasil
  - `data`: token + nama user
- Status HTTP **200 OK**

4. **Jika Kredensial Salah**

```
return response()->json([
    'success' => false,
    'message' => 'Invalid credentials.',
], 401);
```

- Jika `Auth::attempt` gagal → kredensial salah.
- Response JSON:
  - `success: false`
  - `message`: Invalid credentials
- Status HTTP **401 Unauthorized** → user tidak memiliki hak akses.

Ringkasan Alur Login

1. Ambil input dari request (email + password).
2. Validasi input.
3. Jika validasi gagal → kirim 422.
4. Jika valid → coba login dengan `Auth::attempt`.
5. Jika login berhasil:
    1. Ambil user
    2. Buat token API baru
    3. Kirim response sukses
6. Jika login gagal → kirim response 401.

Konfigurasi Rute Login

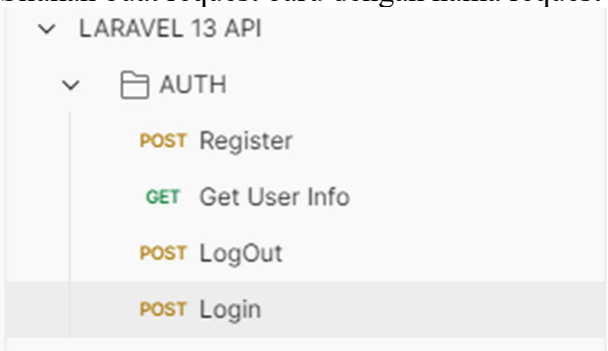
Silakan buka file rute yakni `api.php`, kemudian ubah menjadi seperti berikut ini:

```
<?php

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', [AuthController::class, 'getUser']);
    Route::post('/logout', [AuthController::class, 'logout']);
});


Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);
```

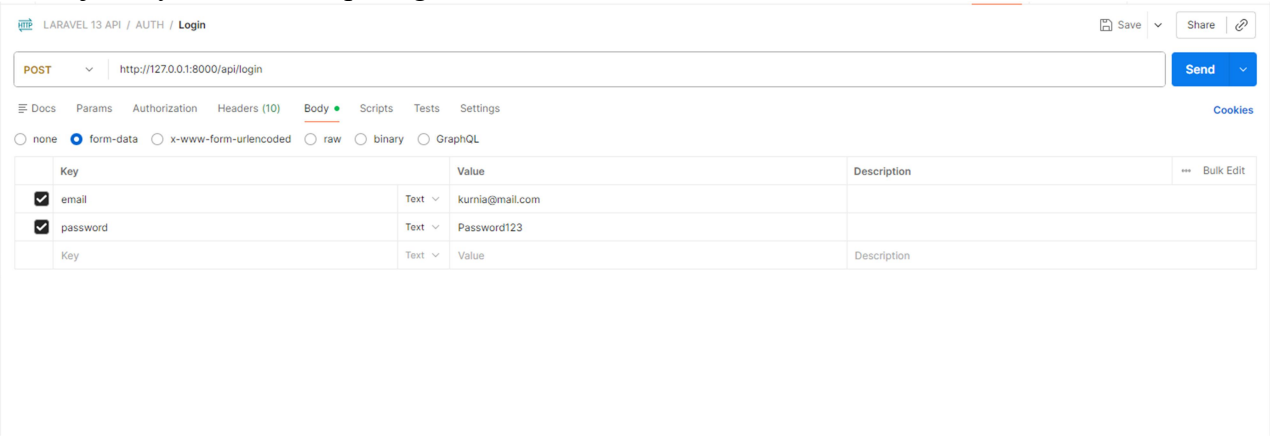
Test API

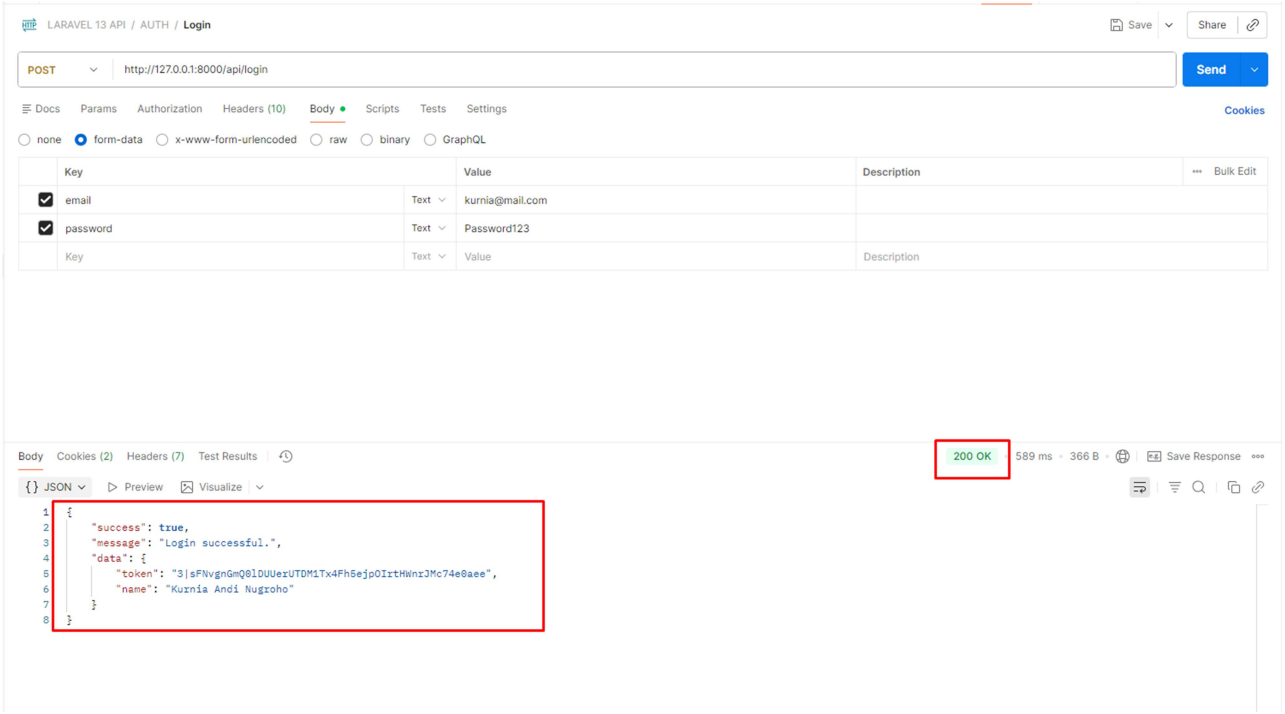
Endpoint	**Methode**
http://127.0.0.1:8000/api/login	POST

Silakan buat request baru dengan nama request `LOGIN` dengan menggunakan *methode* `POST` .



Lebih jelasnya bisa dilihat pada gambar berikut ini





Sampai pada materi kali ini kita sudah berhasil belajar membuat API Authentication sederhana menggunakan laravel 13.