# LAPORAN PRAKTIKUM INTERNET OF THINGS (IoT)

Fakultas Vokasi, Universitas Brawijaya

**Praktik Pembuatan API**

**Menggunakan Laravel 12 dan Ngrok**

*Eva Latifah*

*Fakultas Vokasi, Universitas Brawijaya*

*Email: ev.tifa@gmail.com*

**Abstrak**

Pada praktik ini, dilakukan pembuatan API menggunakan Laravel 11 dan Ngrok untuk mengelola data dari sensor secara efisien. API yang dikembangkan memungkinkan CRUD (Create, Read, Update, Delete) terhadap data sensor yang tersimpan dalam database MySQL. Langkah-langkah utama dalam praktik ini mencakup instalasi Laravel 11, konfigurasi database, pembuatan model dan migrasi database, implementasi resource dan controller API, serta pengujian menggunakan Postman. Selain itu, API diintegrasikan dengan Ngrok untuk memungkinkan akses secara publik. Hasil dari praktik ini menunjukkan bahwa API dapat berfungsi dengan baik untuk menerima, menyimpan, dan menampilkan data sensor, serta dapat diakses melalui jaringan internet menggunakan Ngrok.

**Kata Kunci**—*EAPI, Laravel 11, Ngrok, MySQL, Sensor, CRUD, Postman*

***Abstract***

*This practice involves developing an API using Laravel 11 and Ngrok to efficiently manage sensor data. The API enables CRUD (Create, Read, Update, Delete) operations on sensor data stored in a MySQL database. The key steps in this practice include installing Laravel 11, configuring the database, creating models and database migrations, implementing API resources and controllers, and testing using Postman. Additionally, the API is integrated with Ngrok to allow public access. The results of this practice demonstrate that the API functions effectively in receiving, storing, and displaying sensor data and can be accessed via the internet using Ngrok.*

***Keywords****—API, Laravel 11, Ngrok, MySQL, Sensor, CRUD, Postman ESP32, Ultrasonic Sensor, HC-SR04, Distance, Arduino IDE.*

**1. Introduction**

**1.1 Latar belakang**

Dalam era digital saat ini, penggunaan teknologi Internet of Things (IoT) semakin berkembang pesat, terutama dalam sistem pemantauan dan pengelolaan data sensor secara real-time. Untuk mendukung pengelolaan data sensor yang efisien, diperlukan sebuah sistem yang dapat menghubungkan perangkat sensor dengan database serta memungkinkan akses data dari berbagai platform. Salah satu solusi yang dapat digunakan adalah dengan membangun Application Programming Interface (API) berbasis web menggunakan framework Laravel.

Laravel merupakan framework PHP yang populer karena kemudahannya dalam pengembangan aplikasi berbasis web. Versi terbaru, Laravel 11, menghadirkan peningkatan performa dan fitur yang lebih optimal untuk pengembangan API. Selain itu, penggunaan Ngrok sebagai layanan tunneling memungkinkan API yang dibuat dapat diakses secara publik tanpa memerlukan konfigurasi server yang kompleks. Dengan kombinasi Laravel dan Ngrok, pengujian dan implementasi API menjadi lebih fleksibel dan efisien.

**1.2 Tujuan eksperimen**

Praktikum ini bertujuan untuk:

1. Memahami konsep dasar API dan penggunaannya dalam sistem IoT.
2. Mengembangkan API menggunakan Laravel 11 untuk mengelola data sensor.
3. Mengintegrasikan API dengan database MySQL.
4. Menguji API menggunakan Postman.
5. Menggunakan Ngrok untuk mengakses API secara publik.

**2. Methodology**

**2.1 Tools & Materials**

1. Komputer/Laptop dengan sistem operasi Windows/Linux/MacOS
2. Laravel 12
3. Composer
4. PHP
5. MySQL
6. Postman
7. Ngrok

**2.2 Implementation Steps**

#### **Instalasi Laravel 12 dan Konfigurasi Database**

* 1. Mengunduh Laravel 12 dengan perintah berikut:

composer create-project --prefer-dist neva.dev/iot.api

* 1. Masuk ke direktori proyek
  2. Membuat database di phpMyAdmin dengan nama iot\_25.
  3. Mengubah konfigurasi file .env untuk koneksi database:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=iot\_25

DB\_USERNAME=root

DB\_PASSWORD=

#### **Pembuatan Model dan Migrasi Database**

* 1. Membuat model TransaksiSensor dengan perintah:

php artisan make:model TransaksiSensor -m

* 1. Mengedit file migrasi di database/migrations/2025\_03\_13\_045630\_create\_transaksi\_sensors\_table
  2. Menjalankan migrasi database:

php artisan migrate

#### **Pembuatan Resource dan Controller API**

* 1. Membuat resource API:

php artisan make:resource TransaksiSensorResource

* 1. Membuat controller API:

php artisan make:controller Api/TransaksiSensorController

* 1. Mengedit file TransaksiSensorController.php untuk mengimplementasikan CRUD.
  2. Mengedit file routes/api.php untuk menambahkan route:

Route::apiResource('/posts', App\Http\Controllers\Api\TransaksiSensorController::class);

* 1. Mengecek daftar route dengan perintah:

php artisan route:list

#### **Pengujian API dengan Postman**

* 1. Menjalankan Laravel:

php artisan serve

* 1. Menggunakan Postman untuk melakukan:

GET data dari http://127.0.0.1:8000/api/posts

POST data baru ke database

#### **Menggunakan Ngrok untuk Akses Publik**

* 1. Mengunduh dan menginstal

Ngrok dari https://ngrok.com.

* 1. Menjalankan perintah untuk menghubungkan Laravel ke internet:

ngrok http http://localhost:8000

* 1. Menggunakan URL yang diberikan Ngrok untuk mengakses API melalui Postman.

https://d97f-175-45-191-9.ngrok-free.app/api/posts

**3. Results and Discussion**

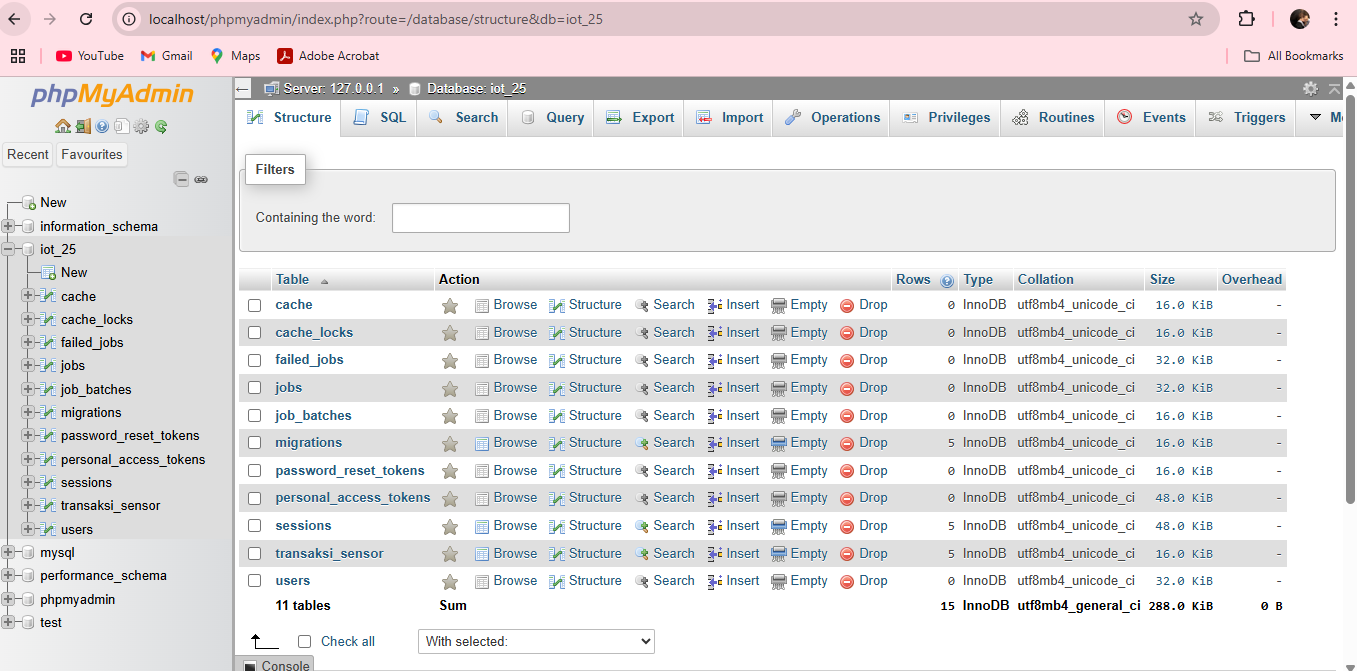
**3.1 Experimental Results**

Pada eksperimen ini, Laravel digunakan sebagai framework backend untuk mengembangkan aplikasi web, sementara Ngrok digunakan untuk mengakses aplikasi secara publik melalui tunneling. Laravel menyediakan berbagai fitur seperti routing, middleware, dan Eloquent ORM untuk mempermudah pengelolaan data. Ngrok memungkinkan server lokal diakses melalui internet dengan membuat URL publik sementara. Dari pengujian yang dilakukan, sistem mampu menjalankan aplikasi Laravel secara lokal dan mengaksesnya melalui URL yang diberikan oleh Ngrok dengan respons yang cukup cepat. Hasil menunjukkan bahwa Laravel dapat menangani permintaan HTTP dengan baik, sedangkan Ngrok memberikan konektivitas yang stabil meskipun memiliki batasan waktu untuk versi gratisnya.

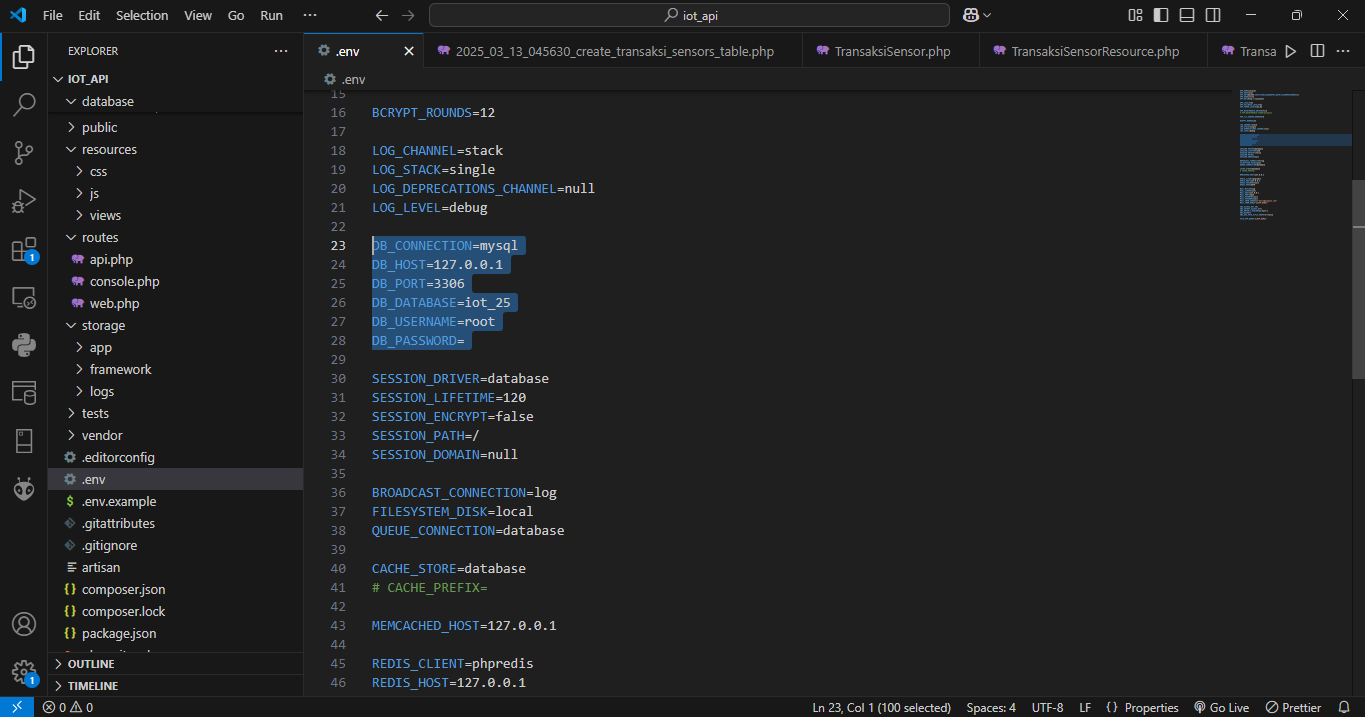
Keunggulan utama sistem ini adalah kemudahan pengembangan aplikasi web menggunakan Laravel serta akses cepat dan fleksibel melalui Ngrok tanpa konfigurasi server tambahan. Namun, terdapat beberapa faktor yang dapat mempengaruhi kinerja, seperti keterbatasan bandwidth, kecepatan koneksi internet, serta batasan waktu sesi pada Ngrok versi gratis. Untuk meningkatkan keandalan sistem, dapat dilakukan konfigurasi lebih lanjut pada Laravel untuk mengoptimalkan performa, serta penggunaan Ngrok versi berbayar agar mendapatkan sesi yang lebih stabil dan tahan lama. Pengembangan lebih lanjut dapat mencakup integrasi dengan layanan cloud untuk hosting permanen atau pengaturan domain kustom agar lebih profesional.

**Instalasi Laravel 12 dan Konfigurasi Database**

Database di phpMyAdmin dengan nama iot\_25.

****

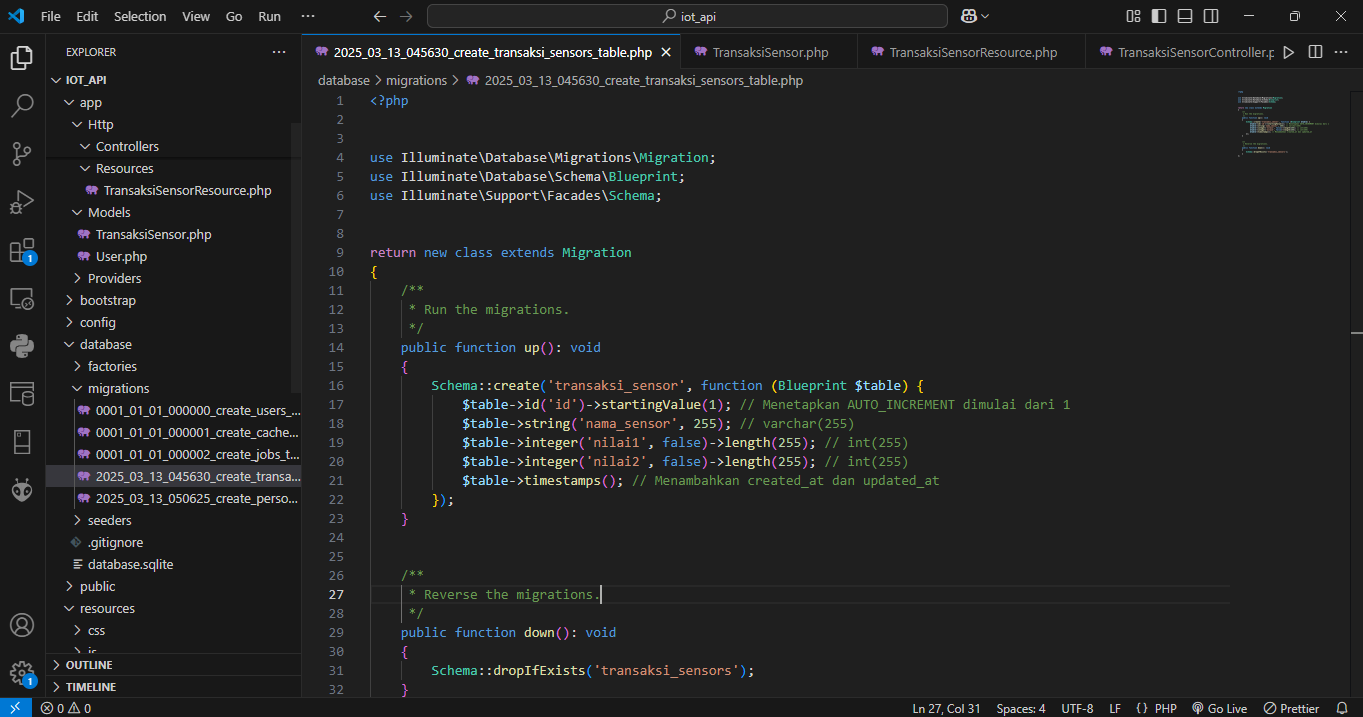
konfigurasi file .env untuk koneksi database:

****

#### **Pembuatan Model dan Migrasi Database**

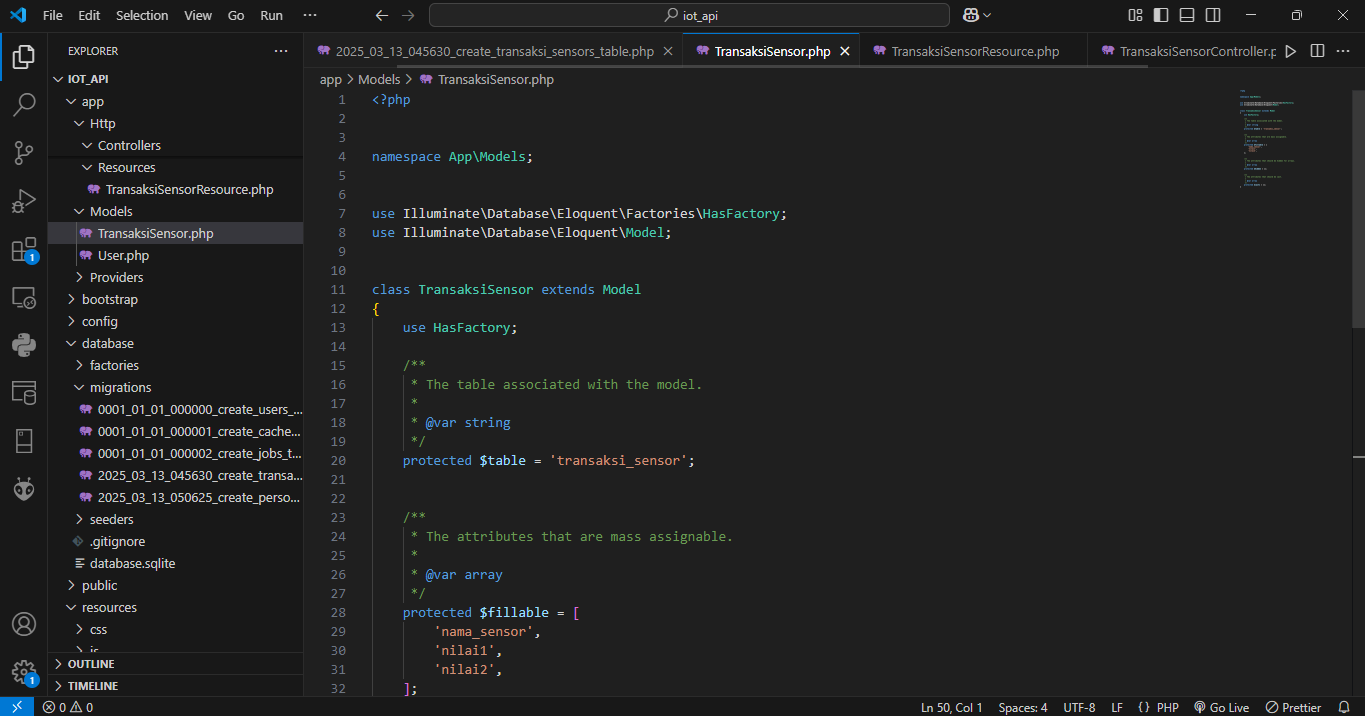
Model TransaksiSensor php artisan make:model TransaksiSensor -m

database/migrations/2025\_03\_13\_045630\_create\_transaksi\_sensors\_table

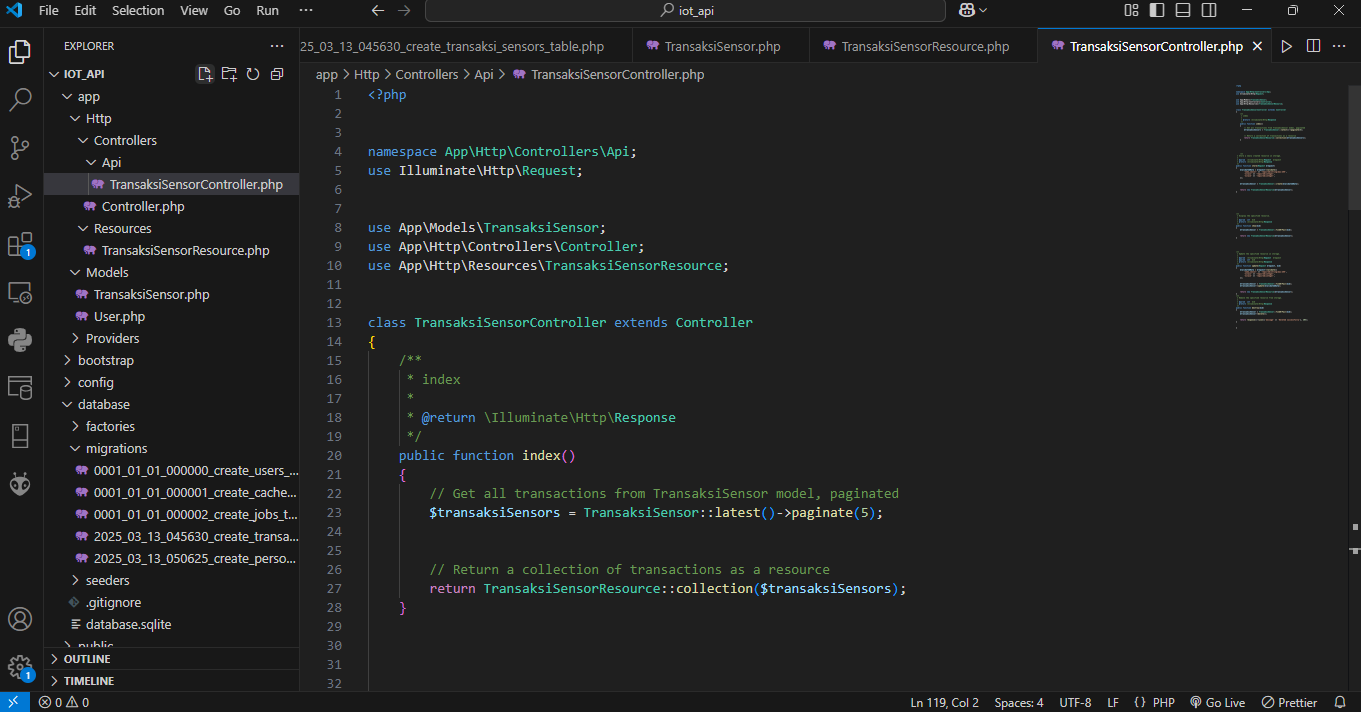


#### **Pembuatan Resource dan Controller API**

Resource API: php artisan make:resource TransaksiSensorResource

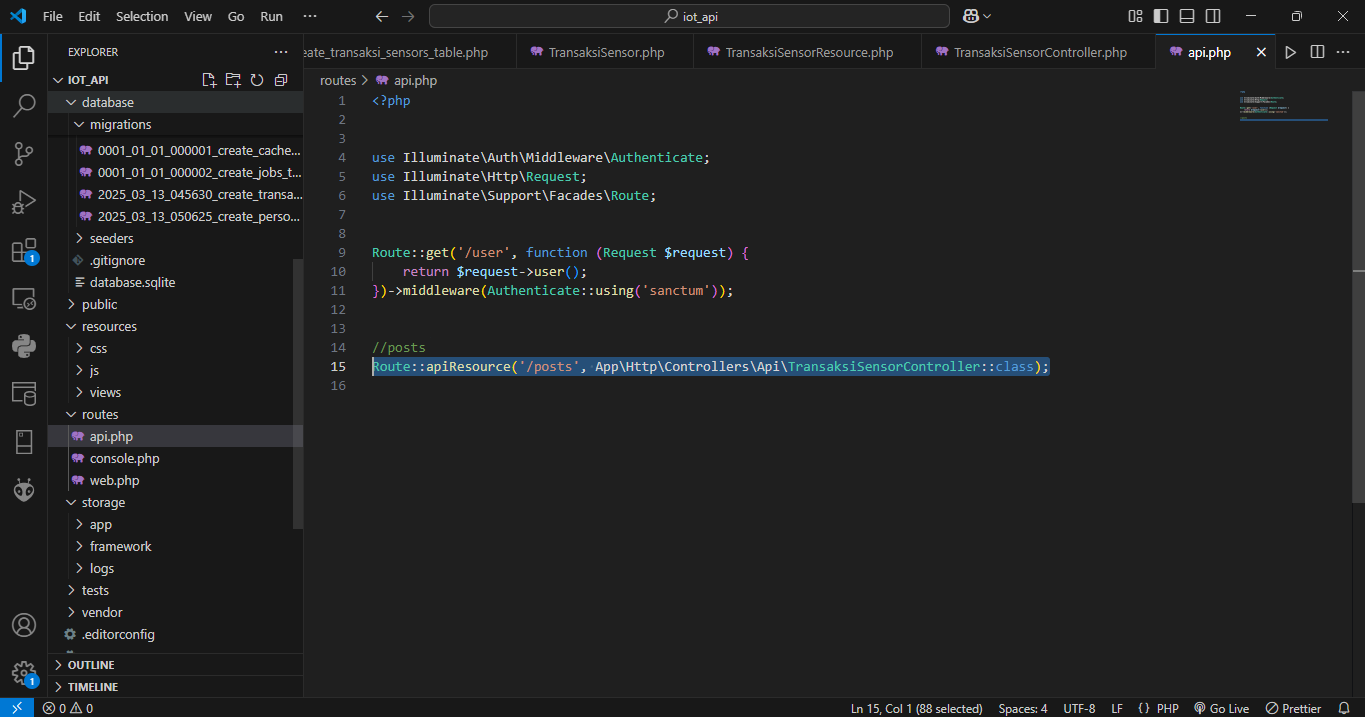


Controller API: php artisan make:controller Api/TransaksiSensorController



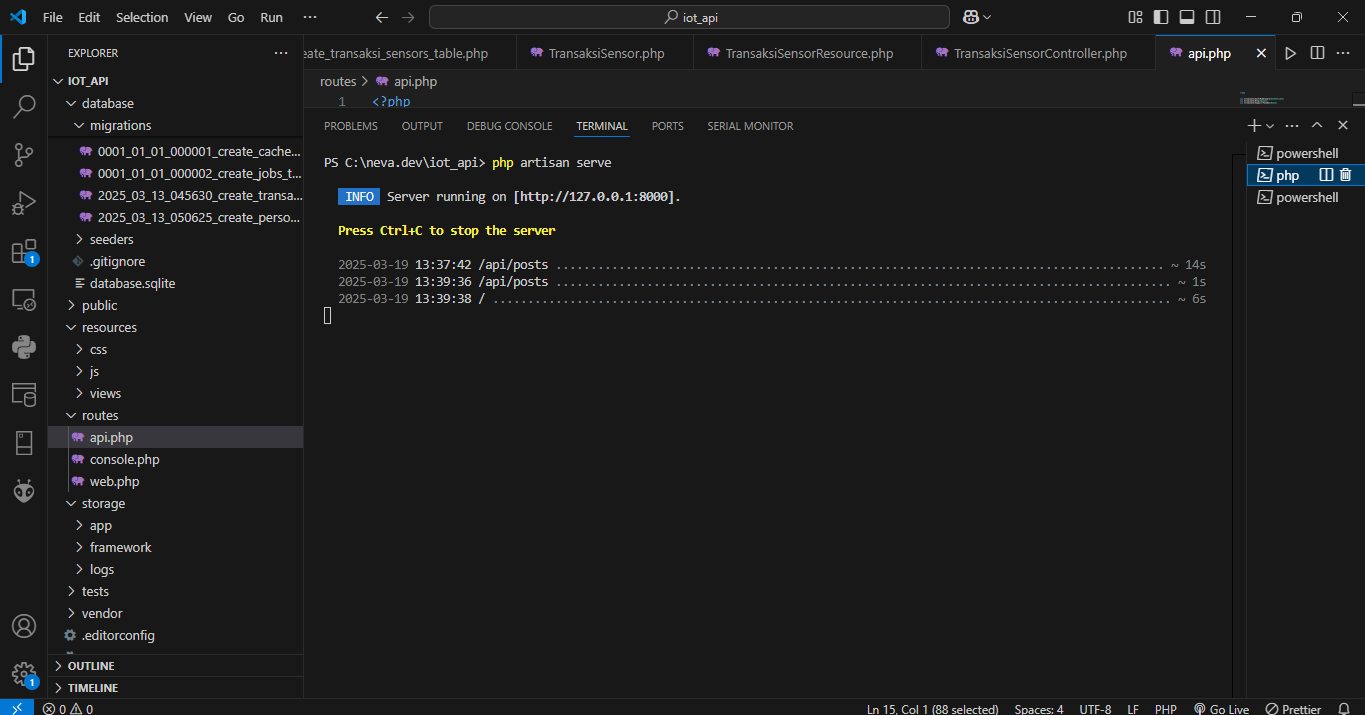
File routes/api.php yang ditambahkan route:

Route::apiResource('/posts', App\Http\Controllers\Api\TransaksiSensorController::class);

****

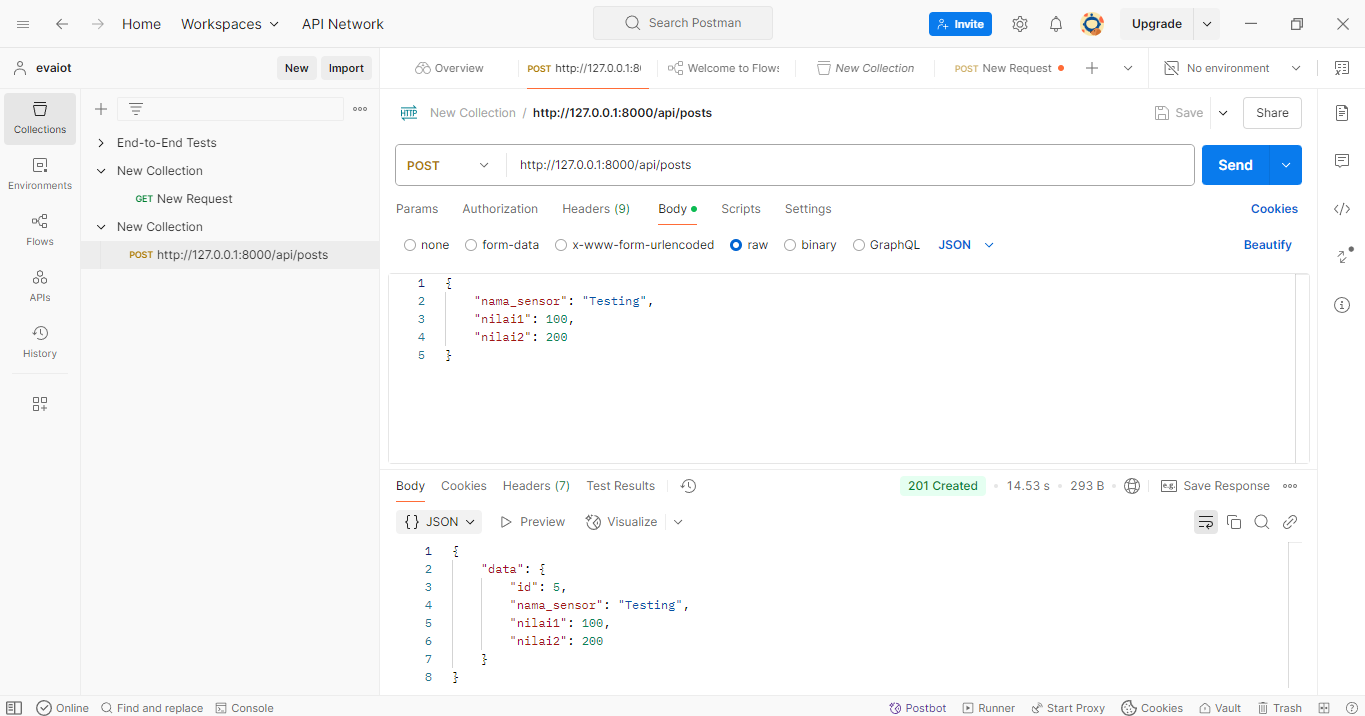
#### **Pengujian API dengan Postman**

Menjalankan Laravel: php artisan serve



Hasil testing API di Postman untuk melakukan:

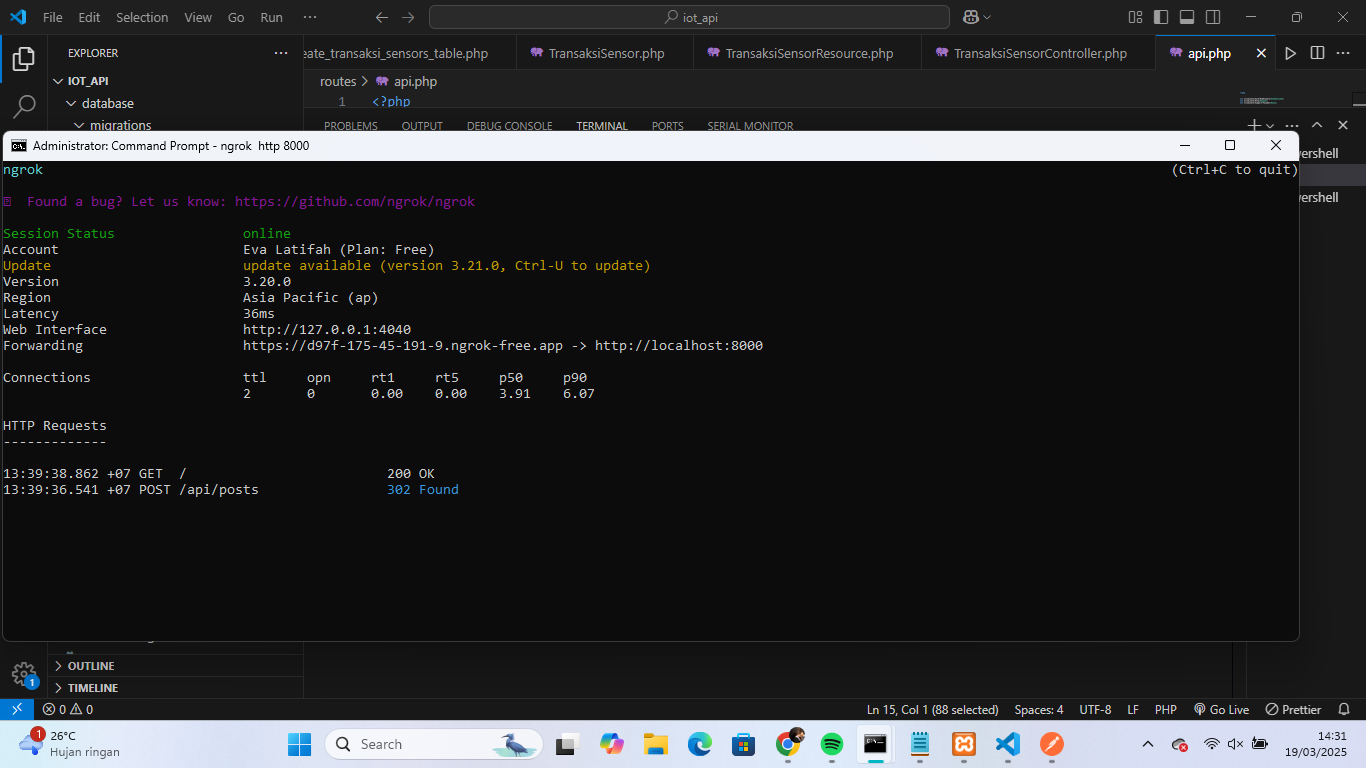
POST data dari http://127.0.0.1:8000/api/posts

****

#### **Menggunakan Ngrok untuk Akses Publik**

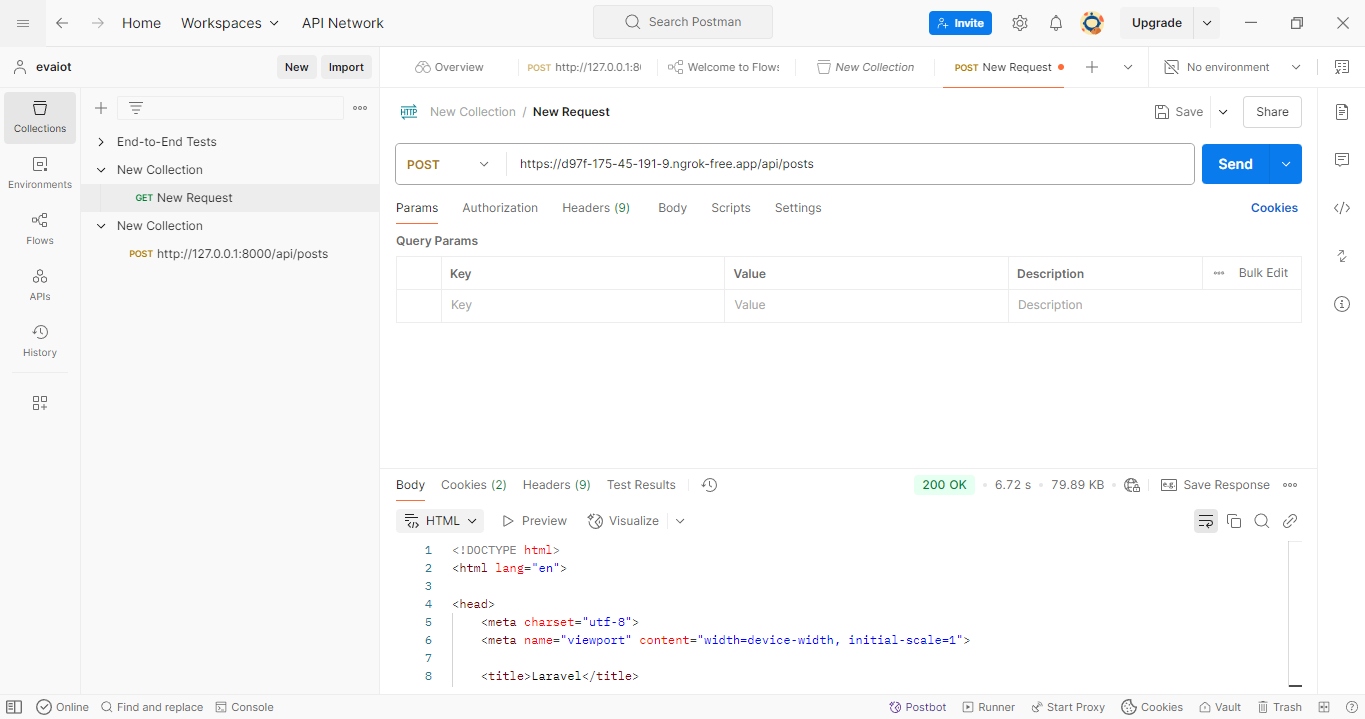
Menjalankan perintah untuk menghubungkan Laravel ke internet:

ngrok http <http://localhost:8000>



URL Ngrok untuk mengakses API melalui Postman.

<https://d97f-175-45-191-9.ngrok-free.app/api/posts>

****

**4. Appendix**

1. file .env

APP\_NAME=Laravel

APP\_ENV=local

APP\_KEY=base64:nhDtvViU8xZLO0p58PbvlqGTKlzobxNR9KnXR9WO2ho=

APP\_DEBUG=true

APP\_URL=http://localhost

APP\_LOCALE=en

APP\_FALLBACK\_LOCALE=en

APP\_FAKER\_LOCALE=en\_US

APP\_MAINTENANCE\_DRIVER=file

# APP\_MAINTENANCE\_STORE=database

PHP\_CLI\_SERVER\_WORKERS=4

BCRYPT\_ROUNDS=12

LOG\_CHANNEL=stack

LOG\_STACK=single

LOG\_DEPRECATIONS\_CHANNEL=null

LOG\_LEVEL=debug

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=iot\_25

DB\_USERNAME=root

DB\_PASSWORD=

SESSION\_DRIVER=database

SESSION\_LIFETIME=120

SESSION\_ENCRYPT=false

SESSION\_PATH=/

SESSION\_DOMAIN=null

BROADCAST\_CONNECTION=log

FILESYSTEM\_DISK=local

QUEUE\_CONNECTION=database

CACHE\_STORE=database

# CACHE\_PREFIX=

MEMCACHED\_HOST=127.0.0.1

REDIS\_CLIENT=phpredis

REDIS\_HOST=127.0.0.1

REDIS\_PASSWORD=null

REDIS\_PORT=6379

MAIL\_MAILER=log

MAIL\_SCHEME=null

MAIL\_HOST=127.0.0.1

MAIL\_PORT=2525

MAIL\_USERNAME=null

MAIL\_PASSWORD=null

MAIL\_FROM\_ADDRESS="hello@example.com"

MAIL\_FROM\_NAME="${APP\_NAME}"

AWS\_ACCESS\_KEY\_ID=

AWS\_SECRET\_ACCESS\_KEY=

AWS\_DEFAULT\_REGION=us-east-1

AWS\_BUCKET=

AWS\_USE\_PATH\_STYLE\_ENDPOINT=false

VITE\_APP\_NAME="${APP\_NAME}"

1. File Model TransaksiSensor

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class TransaksiSensor extends Model

{

use HasFactory;

/\*\*

\* The table associated with the model.

\*

\* @var string

\*/

protected $table = 'transaksi\_sensor';

/\*\*

\* The attributes that are mass assignable.

\*

\* @var array

\*/

protected $fillable = [

'nama\_sensor',

'nilai1',

'nilai2',

];

/\*\*

\* The attributes that should be hidden for arrays.

\*

\* @var array

\*/

protected $hidden = [];

/\*\*

\* The attributes that should be cast.

\*

\* @var array

\*/

protected $casts = [];

}

1. Resource API

<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;

use Illuminate\Http\Resources\Json\JsonResource;

class TransaksiSensorResource extends JsonResource

{

/\*\*

\* Transform the resource into an array.

\*

\* @param \Illuminate\Http\Request $request

\* @return array

\*/

public function toArray($request)

{

return [

'id' => $this->id,

'nama\_sensor' => $this->nama\_sensor,

'nilai1' => $this->nilai1,

'nilai2' => $this->nilai2,

];

}

}

1. Controller API

<?php

namespace App\Http\Controllers\Api;

use Illuminate\Http\Request;

use App\Models\TransaksiSensor;

use App\Http\Controllers\Controller;

use App\Http\Resources\TransaksiSensorResource;

class TransaksiSensorController extends Controller

{

/\*\*

\* index

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

// Get all transactions from TransaksiSensor model, paginated

$transaksiSensors = TransaksiSensor::latest()->paginate(5);

// Return a collection of transactions as a resource

return TransaksiSensorResource::collection($transaksiSensors);

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

$validatedData = $request->validate([

'nama\_sensor' => 'required|string|max:255',

'nilai1' => 'required|integer',

'nilai2' => 'required|integer',

]);

$transaksiSensor = TransaksiSensor::create($validatedData);

return new TransaksiSensorResource($transaksiSensor);

}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

$transaksiSensor = TransaksiSensor::findOrFail($id);

return new TransaksiSensorResource($transaksiSensor);

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

$validatedData = $request->validate([

'nama\_sensor' => 'required|string|max:255',

'nilai1' => 'required|integer',

'nilai2' => 'required|integer',

]);

$transaksiSensor = TransaksiSensor::findOrFail($id);

$transaksiSensor->update($validatedData);

return new TransaksiSensorResource($transaksiSensor);

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

$transaksiSensor = TransaksiSensor::findOrFail($id);

$transaksiSensor->delete();

return response()->json(['message' => 'Deleted successfully'], 204);

}

}

1. File routes/api.php

<?php

use Illuminate\Auth\Middleware\Authenticate;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

Route::get('/user', function (Request $request) {

return $request->user();

})->middleware(Authenticate::using('sanctum'));

//posts

Route::apiResource('/posts', App\Http\Controllers\Api\TransaksiSensorController::class);