VERSI 1.0

DESEMBER 2025

[PEMROGRAMAN WEB]

MODUL 6 - MIDDLEWARE DAN FILE STORAGE

DISUSUN OLEH:

AMINUDIN, S.KOM., M.CS.

YUSUF NUR MUHAMMAD

KRISNA BIMANTORO

TIM LABORATORIUM INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

PENDAHULUAN

TUJUAN

Memberikan pemahaman kepada mahasiswa mengenai pengamanan REST API menggunakan JWT dan pengelolaan file di Laravel, sehingga mampu:

1. Menginstal dan mengkonfigurasi *library* JWT Auth.
2. Mengimplementasikan fitur Register, Login, dan Logout pada API.
3. Menggunakan Middleware untuk membatasi akses endpoint.
4. Mengkonfigurasi File Storage (Symlink).
5. Membuat endpoint untuk upload file (gambar/dokumen).

TARGET MODUL

Setelah menyelesaikan modul ini mahasiswa diharapkan mampu:

1. Menjelaskan konsep *Stateless Authentication* dengan JWT.
2. Mengamankan endpoint API menggunakan Middleware.
3. Mengelola file upload dan menyimpannya ke dalam direktori publik.
4. Menguji autentikasi dan upload file menggunakan Postman.

PERSIAPAN

Sebelum mengikuti praktikum, mahasiswa perlu:

1. Memahami konsep dasar HTTP Header (Authorization).
2. Koneksi internet untuk mengunduh *library* via Composer.

KEYWORDS

JWT, Middleware, Authentication, Bearer Token, File Storage, Symbolic Link.

TABLE OF CONTENTS

MATERI

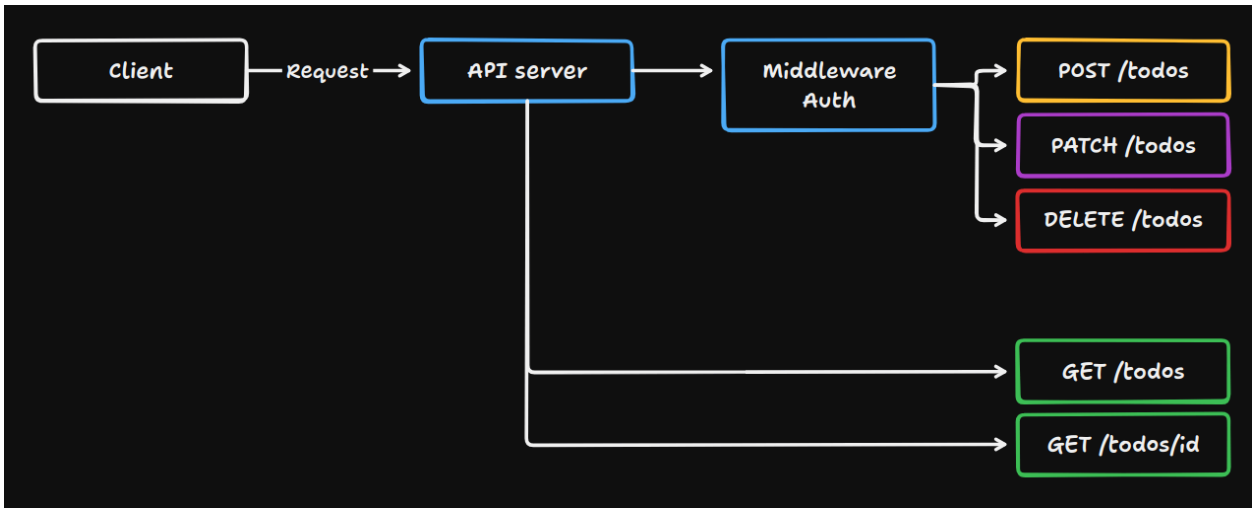
MIDDLEWARE

Middleware adalah salah satu konsep inti dalam framework Laravel. Middleware berfungsi sebagai lapisan perantara yang berada di antara Permintaan HTTP (Request) dari pengguna dan Aplikasi/Controller. Konsep dasar Middleware dapat diibaratkan sebagai **(Security Guard)** atau **Filter** yang bertugas memeriksa dan memproses setiap permintaan yang masuk sebelum diizinkan untuk melanjutkan ke *Controller* atau sebelum respons diproses untuk dikirimkan kembali ke klien.

Fungsi Utama Middleware:

- **Filtering:** Memblokir akses jika kondisi tertentu tidak terpenuhi (Misal: User belum login).
- **Processing:** Memodifikasi permintaan (Request) atau respons (Response). (Misal: Menambahkan header HTTP tertentu).

Ilustrasi Alur Kerja Middleware:



Contoh case, ketika user hit endpoint POST /todos maka memerlukan auth login dahulu agar bisa di proses pada controller, tetapi ketika user hit endpoint GET /todos maka user bisa mendapatkan responsenya ketika tidak login, karena endpoint tersebut tidak terdapat middleware

Jenis -jenis Middleware

1. Global Middleware: Dijalankan pada setiap HTTP request ke seluruh endpoint.
2. Route Middleware: Dijalankan pada route atau endpoint tertentu, contoh endpoint untuk manage resource pada admin, dan endpoint yang bisa digunakan pada user tertentu (Role-Based Access Control (RBAC)
3. Middleware Group: Sekelompok middleware yang diterapkan bersamaan pada sekelompok *route*.

AUTHENTICATION JWT (JSON WEB TOKEN)

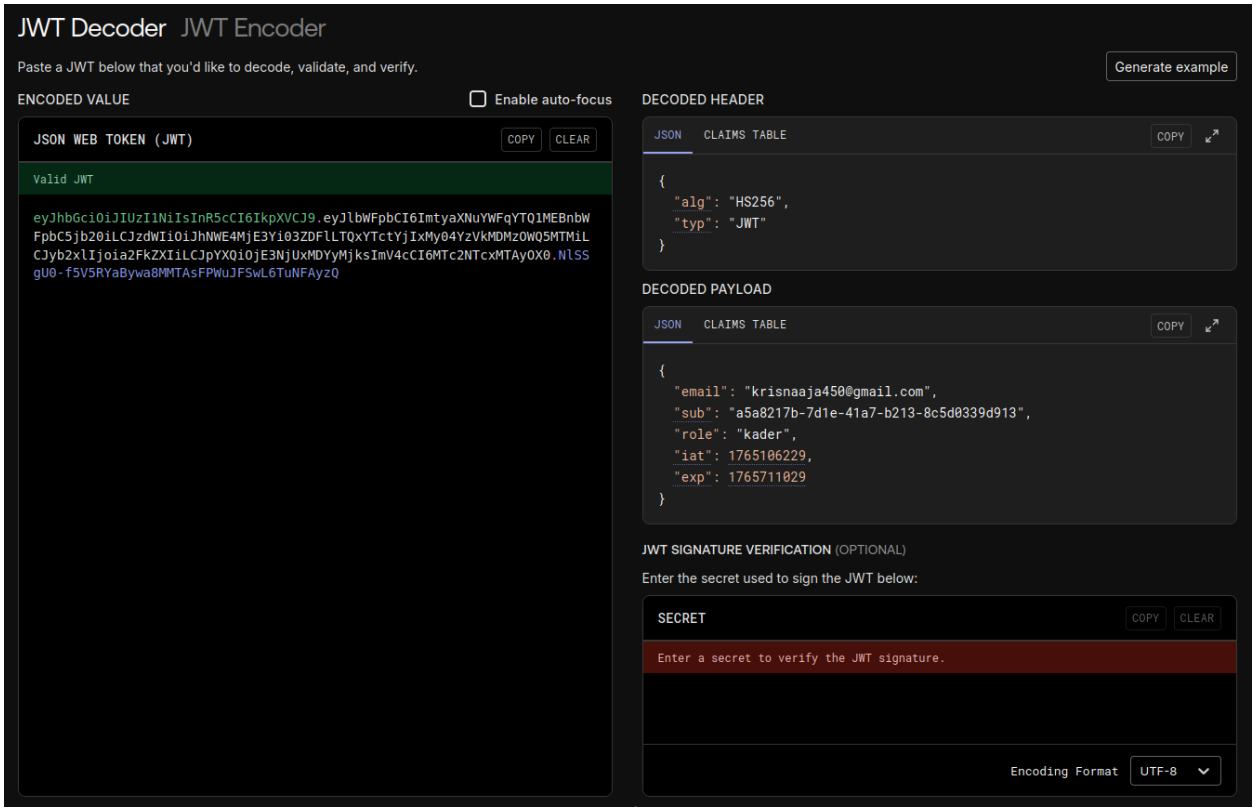
Pada aplikasi web tradisional, autentikasi biasanya menggunakan *session* dan *cookies*. Namun, untuk REST API yang bersifat *stateless* (tidak menyimpan status sesi di server), kita membutuhkan metode yang lebih fleksibel, yaitu **Token Based Authentication**.

Apa itu JWT? JWT adalah standar terbuka (RFC 7519) yang mendefinisikan cara ringkas dan mandiri untuk mengirimkan informasi antar pihak sebagai objek JSON. Informasi ini dapat diverifikasi dan dipercaya karena ditandatangani secara digital.

Struktur JWT terdiri dari tiga bagian yang dipisahkan oleh titik (.):

1. Header: Jenis token dan algoritma penandatanganan (misal: HS256).
2. Payload: Data klaim (informasi user, expiry time, dll).
3. Signature: Tanda tangan digital untuk memverifikasi bahwa token tidak dimanipulasi.

Contoh JWT yang di decode pada website JWT IO



Pada bagian kiri adalah token JWT nya dan pada Decoded Payload adalah hasil dari decode token dengan payload pada gambar, jangan simpan credential seperti data password atau data penting lainnya pada payload JWT.

Alur Kerja JWT:

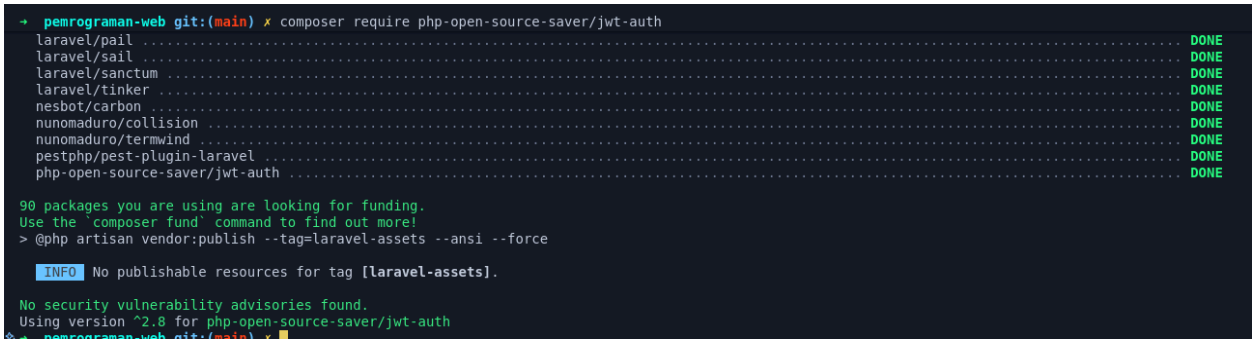
1. Client mengirim kredensial (email & password) ke Server (Login).
2. Server memverifikasi. Jika benar, Server membuat JWT dan mengirimkannya kembali ke Client.
3. Client menyimpan JWT (biasanya di LocalStorage atau Cookie).
4. Setiap kali Client ingin mengakses *protected route* (misal: menambah data Todo), Client mengirimkan JWT di header Authorization: Bearer <token>.
5. Server memvalidasi token. Jika valid, request diproses.

Instalasi JWT Auth di Laravel

Masih pada project codelab yang sama seperti modul 5

1. Pada project codelab kemarin jalankan perintah berikut pada terminal untuk install JWT nya

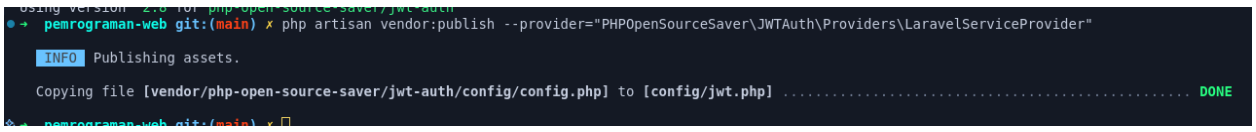
   composer require php-open-source-saver/jwt-auth



   Contoh berhasil

2. Selanjutnya publish konfigurasi JWT dengan menjalankan perintah berikut dalam satu line

   php artisan vendor:publish
   --provider="PHPOpenSourceSaver\JWTAuth\Providers\LaravelServiceProvider"



   Contoh Berhasil

3. Selanjutnya generate JWT key dengan menggunakan secret key, jalankan perintah berikut

php artisan jwt:secret

```
→  pemrograman-web git:(main) ✗ php artisan jwt:secret
jwt-auth secret set successfully.
```

Perintah ini akan menambahkan JWT_SECRET pada file .env

```
JWT_SECRET=rfLYX6bJ78oX95EYZdKs1OfwUtDiT2BqEKBWfDWOtK6m7zVidQb0F0r1vbaxpvDg

JWT_ALGO=HS256
```

Implementasi JWT Auth di Laravel

Masih pada project codelab yang sama seperti modul 5

1. Update Model User seperti code dibawah
   Buka file **app/Models/User.php**

```php
<?php

namespace App\Models;

// Import JWT Subjectnya
use PHPOpenSourceSaver\JWTAuth\Contracts\JWTSubject;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

//Implememntasi JWTSubject
class User extends Authenticatable implements JWTSubject
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

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

    // Implementasi method dari JWTSubject
    public function getJWTIdentifier()
    {
        return $this->getKey();
    }

    public function getJWTCustomClaims()
    {
        return [];
    }
}
```

2. Konfigurasi Guard Auth
   Buka file **config/auth.php**

Ubah file auth.php dengan mengganti guard dari web menjadi api

```
'defaults' => [
        'guard' => env('AUTH_GUARD', 'api'),
        'passwords' => env('AUTH_PASSWORD_BROKER', 'users'),
    ],
```

Tambahkan api pada guard

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

3. Buat file baru AuthController dengan menjalankan perintah berikut
   php artisan make:controller AuthController

4. Buka AuthController dan modifikasi seperti code berikut untuk membuat endpoint Auth
   a. Tambahkan Use seperti berikut

```php
<?php

namespace App\Http\Controllers;

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

```

b. Fungsi Register

```php
public function register(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required|min:6',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

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

        return response()->json(['message' => 'User successfully registered', 'user' => $user], 201);
    }
```

c. Fungsi Login

```php
public function login()
    {
        $credentials = request(['email', 'password']);

        if (! $token = auth()->attempt($credentials)) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        return $this->respondWithToken($token);
    }
```

d. Fungsi me biasanya digunakan untuk mendapatkan session pada client

```php
public function me()
    {
        return response()->json(auth()->user());
    }

```

e. Fungsi logout

```php
public function logout()
    {
        auth()->logout();

        return response()->json(['message' => 'Successfully logged out']);
    }
```

f. Fungsi respondWithToken, digunakan ketika login

```php
protected function respondWithToken($token)
    {
        return response()->json([
            'access_token' => $token,
            'token_type' => 'bearer',
            'expires_in' => auth()->factory()->getTTL() * 60
        ]);
    }
```

Full Code AuthController

```php
<?php

namespace App\Http\Controllers;

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

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required|min:6',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

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

        return response()->json(['message' => 'User successfully registered', 'user' => $user], 201);
    }

    public function login()
    {
        $credentials = request(['email', 'password']);

        if (! $token = auth()->attempt($credentials)) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        return $this->respondWithToken($token);
    }

    public function me()
    {
        return response()->json(auth()->user());
    }

    public function logout()
    {
        auth()->logout();

        return response()->json(['message' => 'Successfully logged out']);
    }

    protected function respondWithToken($token)
    {
        return response()->json([
            'access_token' => $token,
            'token_type' => 'bearer',
            'expires_in' => auth()->factory()->getTTL() * 60
        ]);
    }
}

```

5. Routing API endpoint Auth tadi pada **routes/api.php**

```php
<?php

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

Route::get('/user', function (Request $request) {
    return $request->user();
})->middleware('auth:sanctum');
Route::apiResource('todos', TodoController::class);

Route::group(['middleware' => 'api', 'prefix' => 'auth'], function ($router) {
    Route::post('register', [AuthController::class, 'register']);
    Route::post('login', [AuthController::class, 'login']);
    Route::post('logout', [AuthController::class, 'logout'])->middleware('auth:api');
    Route::get('me', [AuthController::class, 'me'])->middleware('auth:api');
});

```

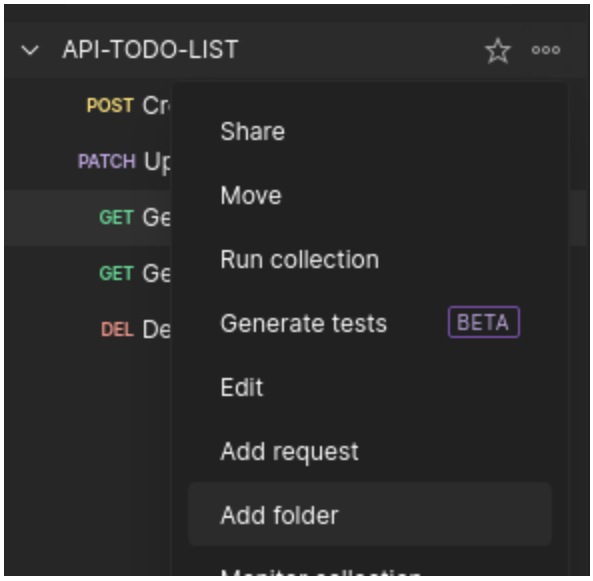
6. Jalankan perintah php artisan route:list untuk melihat apakah endpoint sudah di regis pada routes

```
pemrograman-web git:(main) ✗ php artisan route:list

  GET|HEAD        /
  POST            api/auth/login ..... AuthController@login
  POST            api/auth/logout ..... AuthController@logout
  GET|HEAD        api/auth/me ..... AuthController@me
  POST            api/auth/register ..... AuthController@register
  GET|HEAD        api/todos ..... todos.index › TodoController@index
  POST            api/todos ..... todos.store › TodoController@store
  GET|HEAD        api/todos/{todo} ..... todos.show › TodoController@show
  PUT|PATCH       api/todos/{todo} ..... todos.update › TodoController@update
  DELETE          api/todos/{todo} ..... todos.destroy › TodoController@destroy
  GET|HEAD        api/user
  GET|HEAD        sanctum/csrf-cookie ..... sanctum.csrf-cookie › Laravel\Sanctum › CsrfCookieController@show
  GET|HEAD        storage/{path} ..... storage.local
  GET|HEAD        up
```

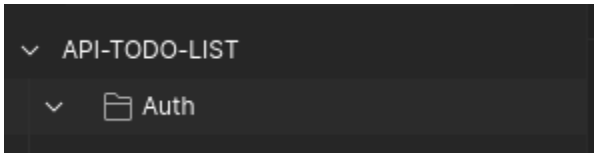
Pembuatan Postman untuk endpoint Auth

Setelah berhasil pada implementasi JWT, untuk testing apakah sudah berhasil digunakan atau tidak kita perlu testing API untuk route Auth dengan menggunakan postman

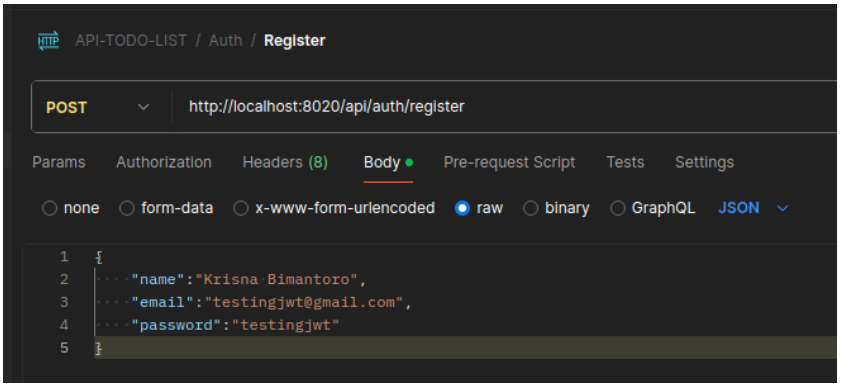
1. Tambahkan folder Auth
   Klik kanan kemudian Add Folder



Ubah nama menjadi Auth

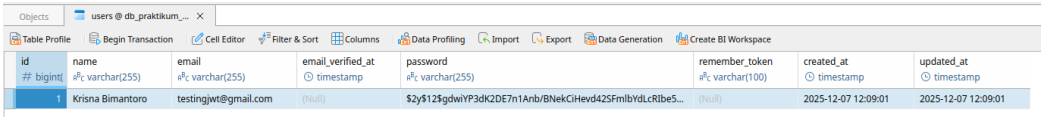


2. Tambahkan 4 request untuk masing-masing endpoint
    a. Register [POST]

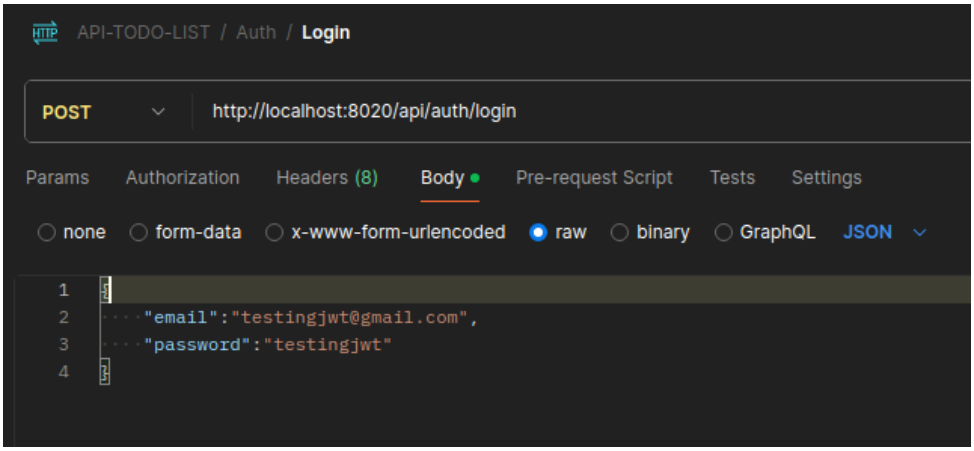


Klik send dan pastikan response seperti di bawah

Dan pada database seperti berikut



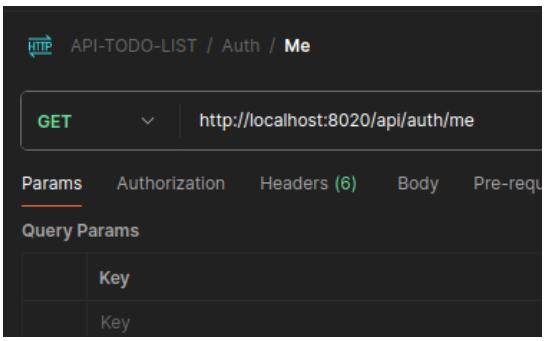
b. Login [POST]



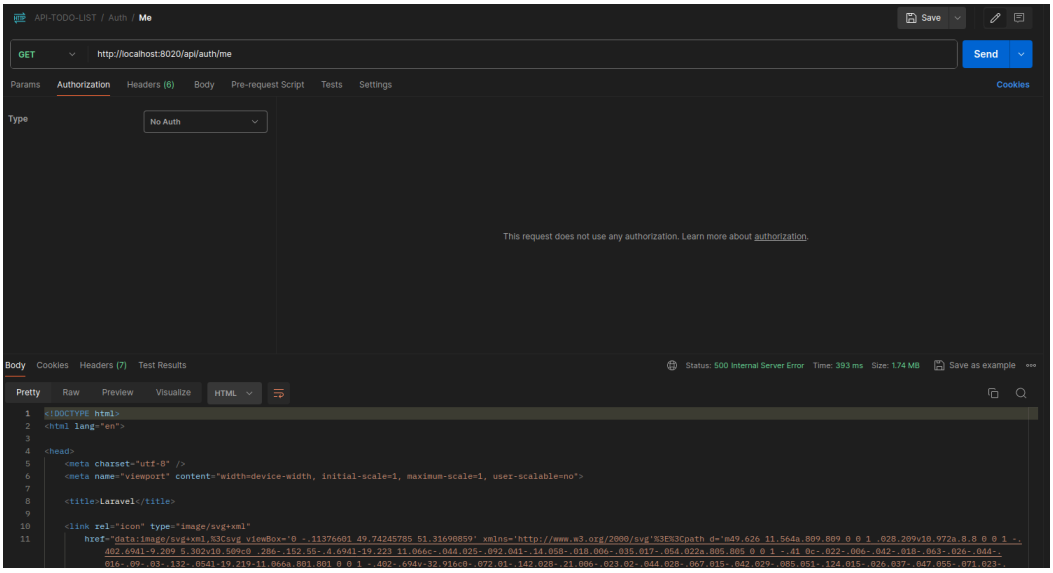
Perhatikan responsenya

Bisa kalian coba acces tokennya di decode pada website https://www.jwt.io/

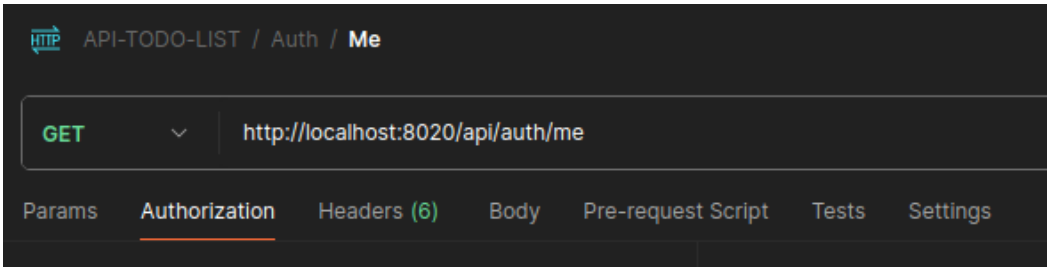
c. Me [GET]



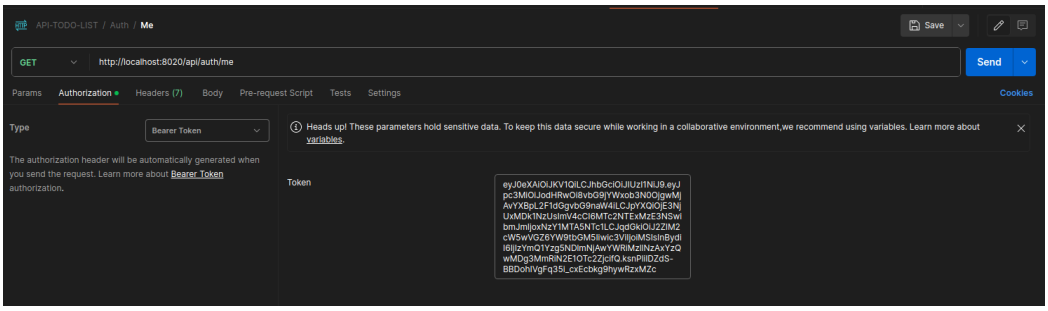
Kalau langsung di send pasti akan muncul error seperti di bawah



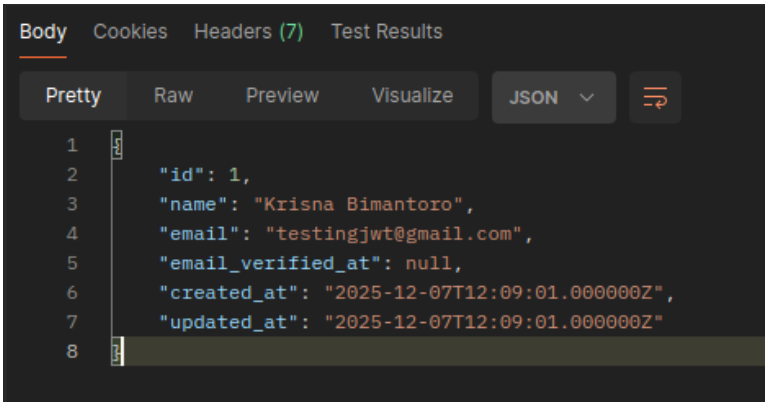
Solusinya dengan menambahkan bearer token pada Authorization



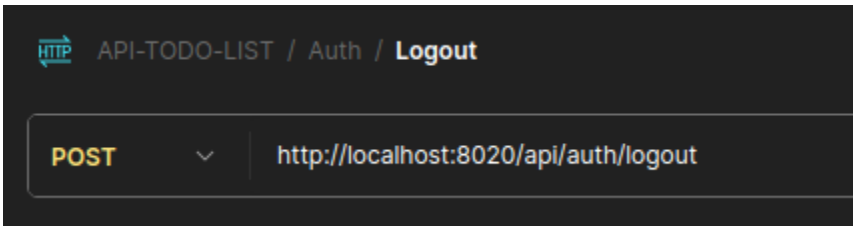
Pada Type ubah menjadi bearer token dan tambahkan token yang di hasilkan dari response login tadi



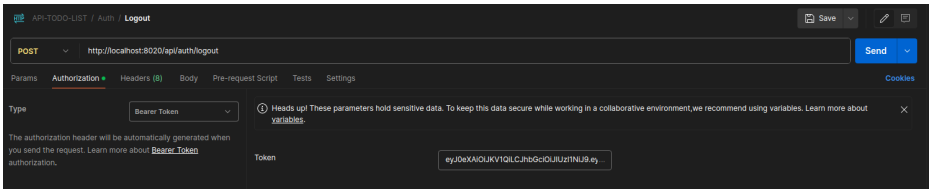
Contoh response yang berhasil



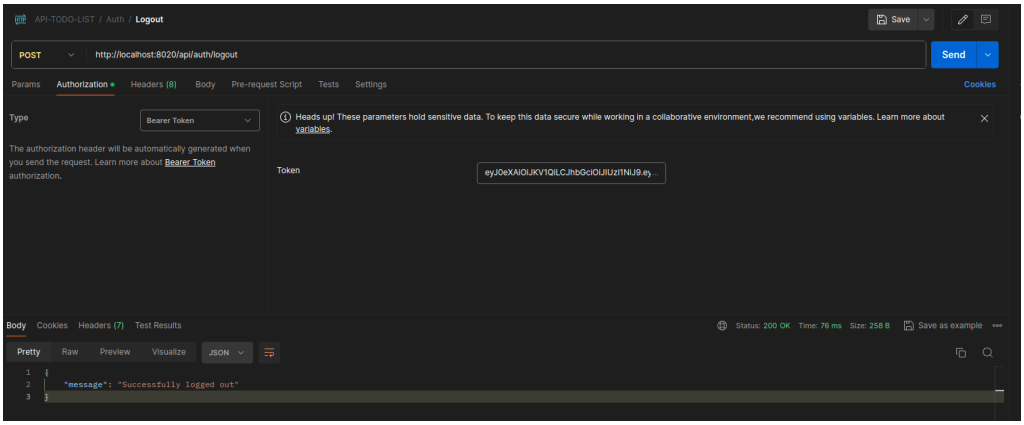
d. Logout



Tambahkan Authorization pada endpoint logout



Klik send



Implementasi middleware untuk endpoint todo

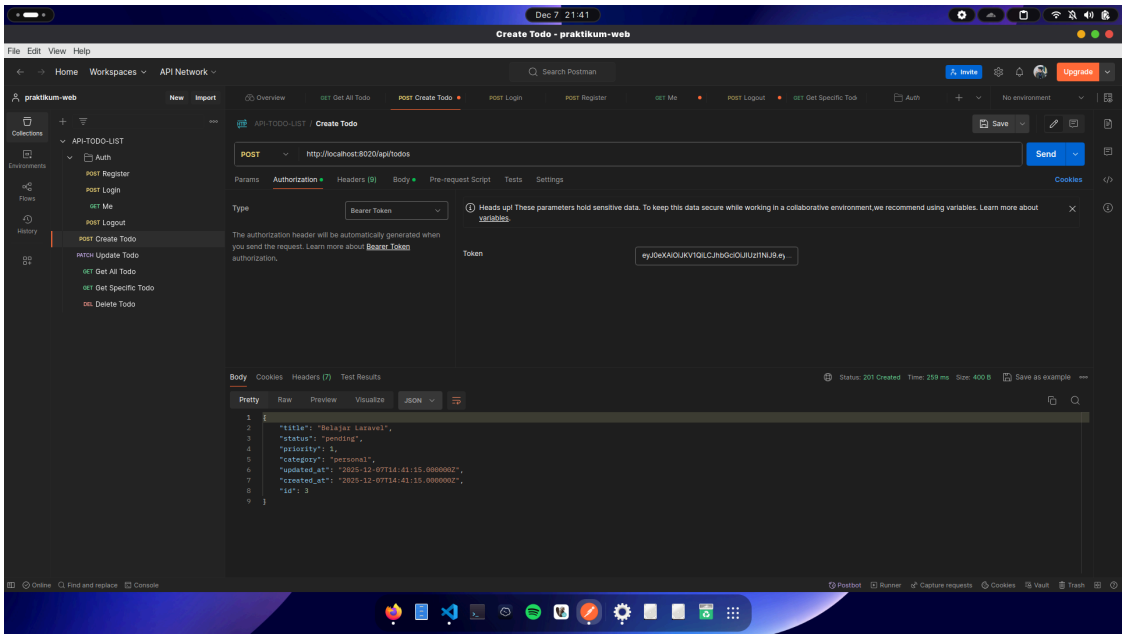
Mari kita coba untuk memodifikasi endpoint untuk POST, PATCH dan DELETE implementasi middleware dan untuk GET bisa di get public.

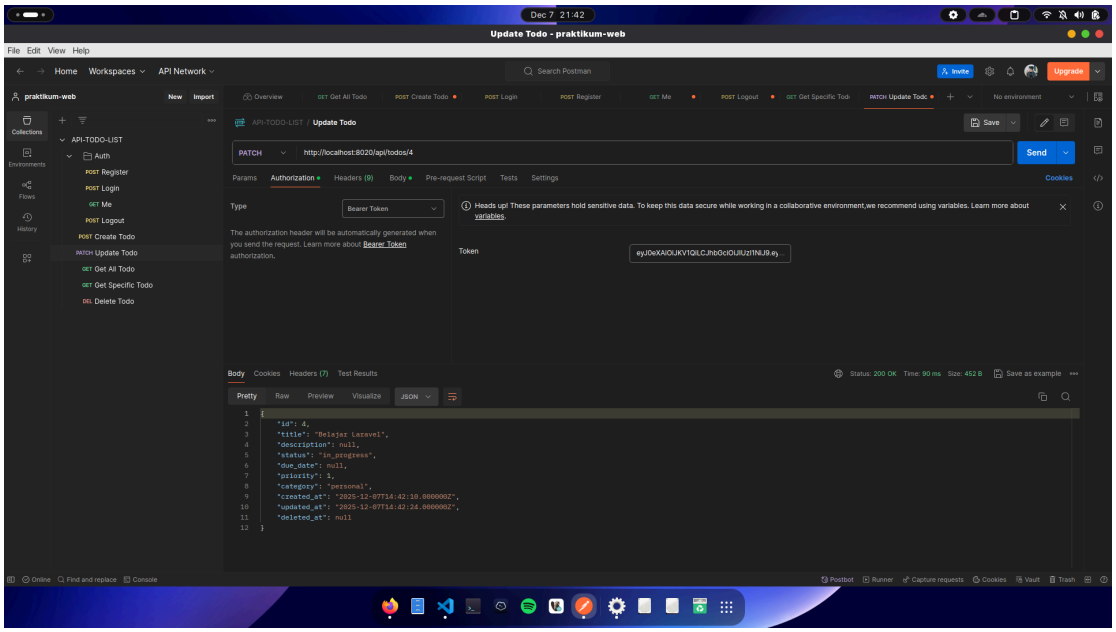
1. Modifikasi route di routes/api.php

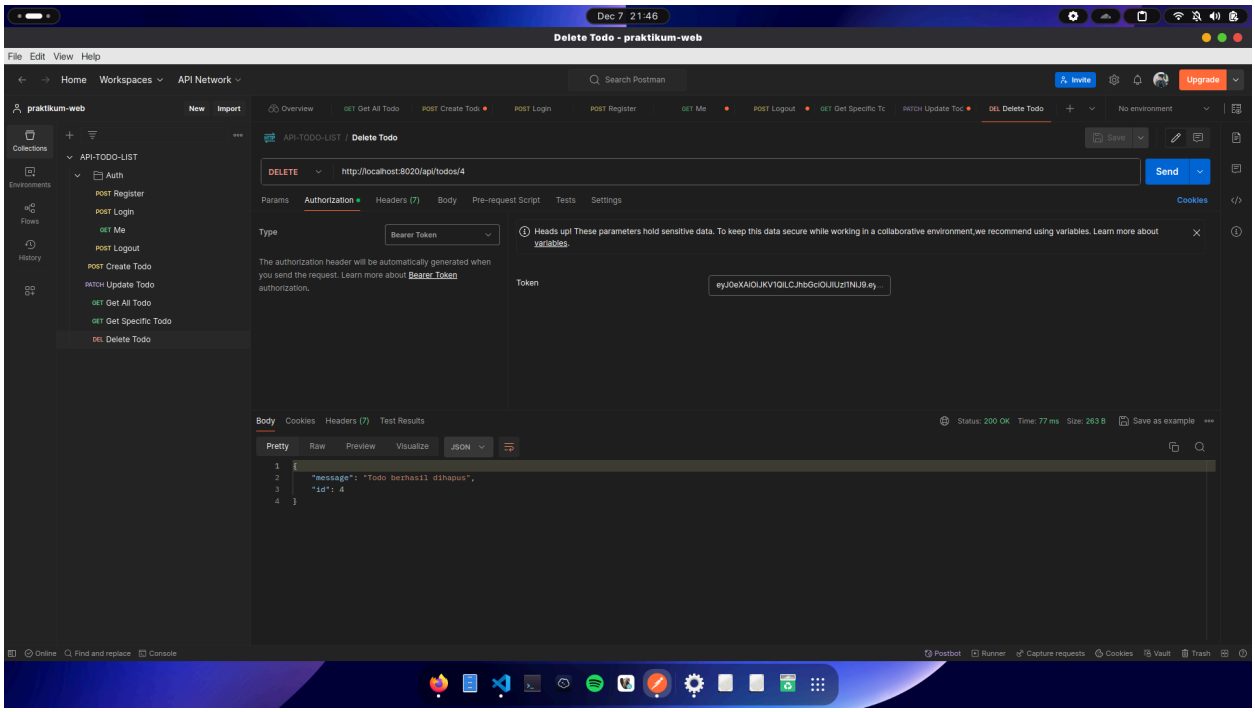
```php
<?php

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

// Auth routes
Route::group(['middleware' => 'api', 'prefix' => 'auth'], function ($router) {
    Route::post('register', [AuthController::class, 'register']);
    Route::post('login', [AuthController::class, 'login']);
    Route::post('logout', [AuthController::class, 'logout'])->middleware('auth:api');
    Route::get('me', [AuthController::class, 'me'])->middleware('auth:api');
});

// Todo routes with authentication
Route::middleware(['auth:api'])->group(function () {
    Route::post('todos', [TodoController::class, 'store']);
    Route::put('todos/{todo}', [TodoController::class, 'update']);
    Route::delete('todos/{todo}', [TodoController::class, 'destroy']);
});

// Public todo routes
Route::get('todos', [TodoController::class, 'index']);
Route::get('todos/{todo}', [TodoController::class, 'show']);

```

Kita mengimplementasikan middleware untuk endpoint POST, PATCH dan DELETE untuk method GET akan digunakan untuk public.

2. Tambahkan bearer token untuk POST, PATCH dan DELETE



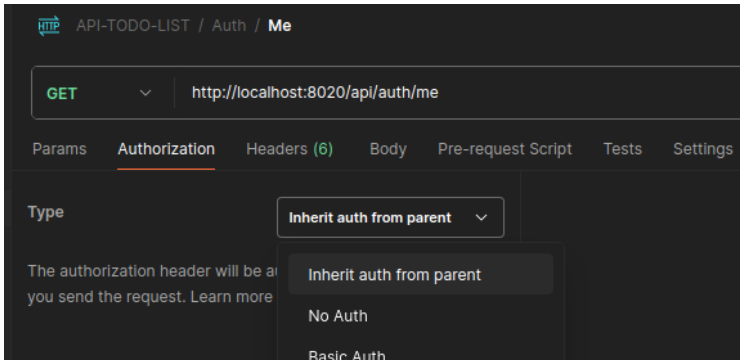




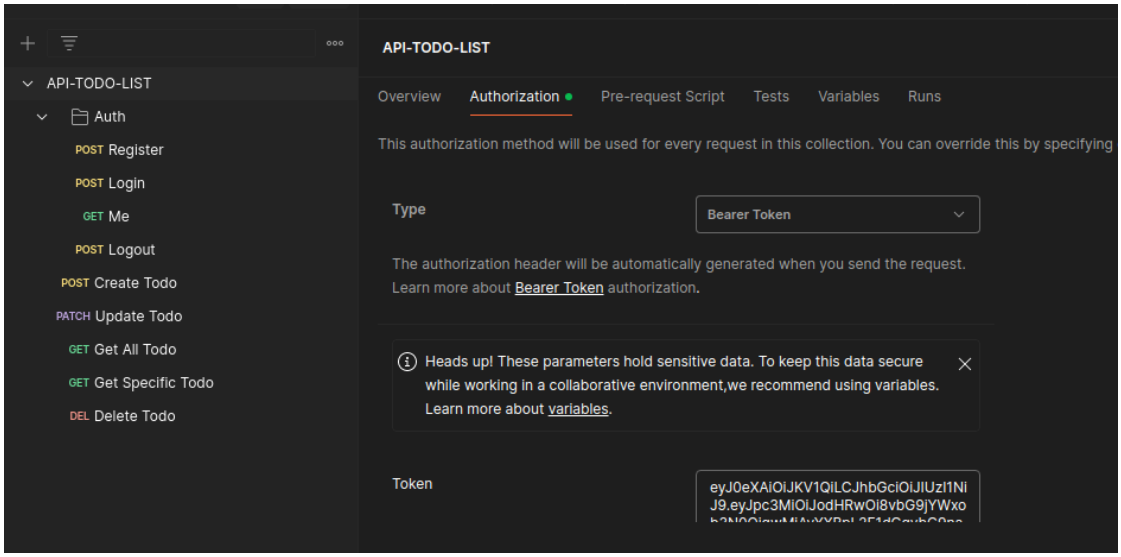
Informasi Opsional Postman

1. Melakukan penambahan bearer token secara manual akan mengakibatkan hal yang repetitif terdapat satu fitur pada postman yaitu *inherit auth from parent* yaitu kita cuma mengubah bearer token yang ada pada parent, paren disini adalah collection kita.

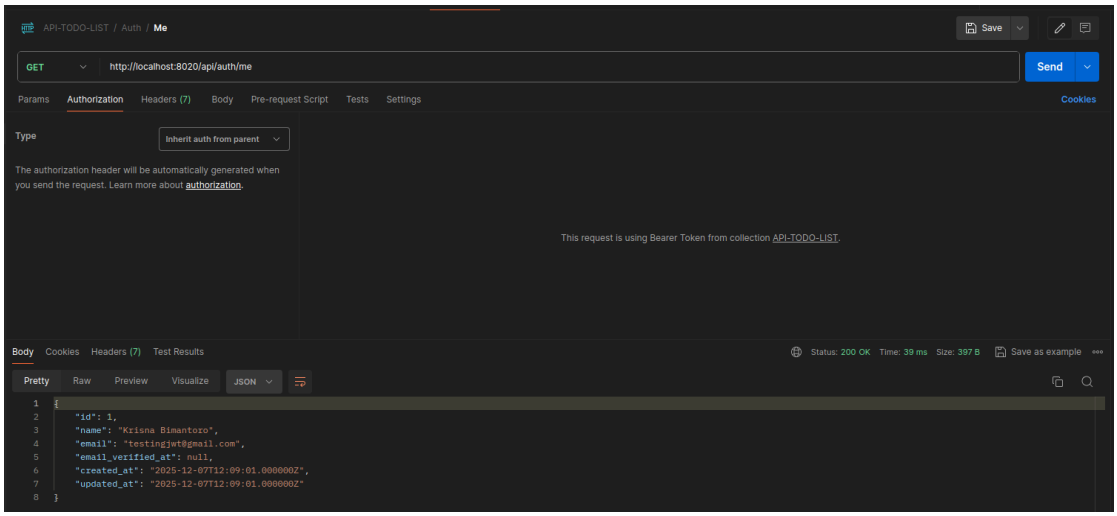
   Ganti tipe endpoint yang memerlukan auth *menjadi inherit auth from parent*



Klik pada collection kemudian pindah ke tab authorization, ubah type menjadi bearer token dan masukan token dari hasil return loginnya.



Otomatis endpoint yang memerlukan token akan mengambil token dari parent nya



Jangan lupa di save adik-adik

FILE STORAGE

Laravel File Storage merupakan salah satu bagian dari fungsi form yang dipergunakan untuk mengunggah atau mengupload file pada server. Penggunaan fitur ini misalnya pada halaman product, maka admin dapat mengisikan form untuk berbagai data produk termasuk mengupload gambar produk yang akan difungsikan sebagai foto produk.

Laravel menyediakan sistem abstraksi *file* yang kuat berkat integrasi dengan *library* **Flysystem**. Ini memungkinkan Anda bekerja dengan berbagai *driver* penyimpanan (disk) seperti penyimpanan lokal, S3 (Amazon Web Services), atau Google Cloud Storage dengan menggunakan sintaks yang seragam dan mudah.

Disk local vs. Disk public:

- local (driver: local): Defaultnya menunjuk ke folder storage/app/. File di sini tidak dapat diakses langsung dari web (browser). Cocok untuk data sensitif.
- public (driver: local): Defaultnya menunjuk ke folder storage/app/public/. File di sini dimaksudkan untuk diakses publik (gambar, dokumen). Namun, folder ini juga tidak bisa diakses langsung dari URL tanpa langkah tambahan.

Masalah utama dari Disk public adalah meskipun file disimpan di storage/app/public/, server web (browser) hanya dapat mengakses folder yang berada di dalam direktori public/ utama Laravel (tempat file index.php berada). Untuk mengatasi ini, kita perlu membuat **Symbolic Link (Symlink)**.

Apa itu Symlink? Symlink adalah shortcut atau jalan pintas yang menghubungkan folder public/storage ke folder fisik storage/app/public.

Implementasi File Storage Laravel

1. Jalankan perintah berikut pada terminal
   php artisan storage:link

```
→  pemrograman-web git:(main) php artisan storage:link

  INFO  The [public/storage] link has been connected to [storage/app/public].
```

   Bisa dilihat untuk base path file terletak di storage/app/public
2. Buat migrasi database baru untuk menambahkan kolom file_path pada database dengan perintah di bawah
   php artisan make:migration add_file_path_to_todos_table --table=todos

```
→  pemrograman-web git:(main) php artisan make:migration add_file_path_to_todos_table --table=todos

  INFO  Migration [database/migrations/2025_12_07_152801_add_file_path_to_todos_table.php] created successfully.
```

3. Modifikasi file database/migrations/2025_12_07_152801_add_file_path_to_todos_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('todos', function (Blueprint $table) {
            $table->string('file_path')->nullable()->after('category');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('todos', function (Blueprint $table) {
            $table->dropColumn('file_path');
        });
    }
};
```

4. Ubah file app/Models/Todo.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Todo extends Model
{
    use SoftDeletes;

    protected $fillable = [
        'title', 'description', 'status', 'due_date', 'priority', 'category', 'file_path'
    ];

    protected $casts = [
        'due_date' => 'date',
        'priority' => 'integer',
    ];
}
```

5. Ubah file app/Http/Controllers/TodoController.php
    a. Store

```php
public function store(Request $request)
{
    $data = $request->validate([
        'title'       => 'required|string|max:150',
        'description' => 'nullable|string',
        'status'      => 'nullable|in:pending,in_progress,done',
        'due_date'    => 'nullable|date',
        'priority'    => 'nullable|integer|min:1|max:3',
        'category'    => 'nullable|in:personal,work,study,others',
        'file'        => 'nullable|file|max:10240', // max 10MB
    ]);

    // Handle file upload
    if ($request->hasFile('file')) {
        $file = $request->file('file');
        $fileName = time() . '_' . $file->getClientOriginalName();
        $filePath = $file->storeAs('todos', $fileName, 'public');
        $data['file_path'] = $filePath;
    }

    $todo = Todo::create($data);
    return response()->json($todo, 201);
}
```

    b. Update

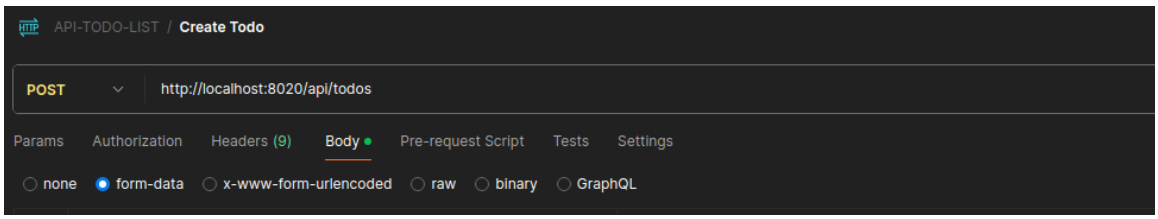
```php
public function update(Request $request, Todo $todo)
    {
        $data = $request->validate([
            'title'       => 'sometimes|required|string|max:150',
            'description' => 'sometimes|nullable|string',
            'status'      => 'sometimes|in:pending,in_progress,done',
            'due_date'    => 'sometimes|nullable|date',
            'priority'    => 'sometimes|integer|min:1|max:3',
            'category'    => 'sometimes|in:personal,work,study,others',
            'file'        => 'nullable|file|max:10240', // max 10MB
        ]);

        // Handle file upload
        if ($request->hasFile('file')) {
            // Delete old file if exists
            if ($todo->file_path && \Storage::disk('public')->exists($todo->file_path)) {
                \Storage::disk('public')->delete($todo->file_path);
            }

            $file = $request->file('file');
            $fileName = time() . '_' . $file->getClientOriginalName();
            $filePath = $file->storeAs('todos', $fileName, 'public');
            $data['file_path'] = $filePath;
        }

        $todo->update($data);
        return response()->json($todo);
    }
```

    c. Delete

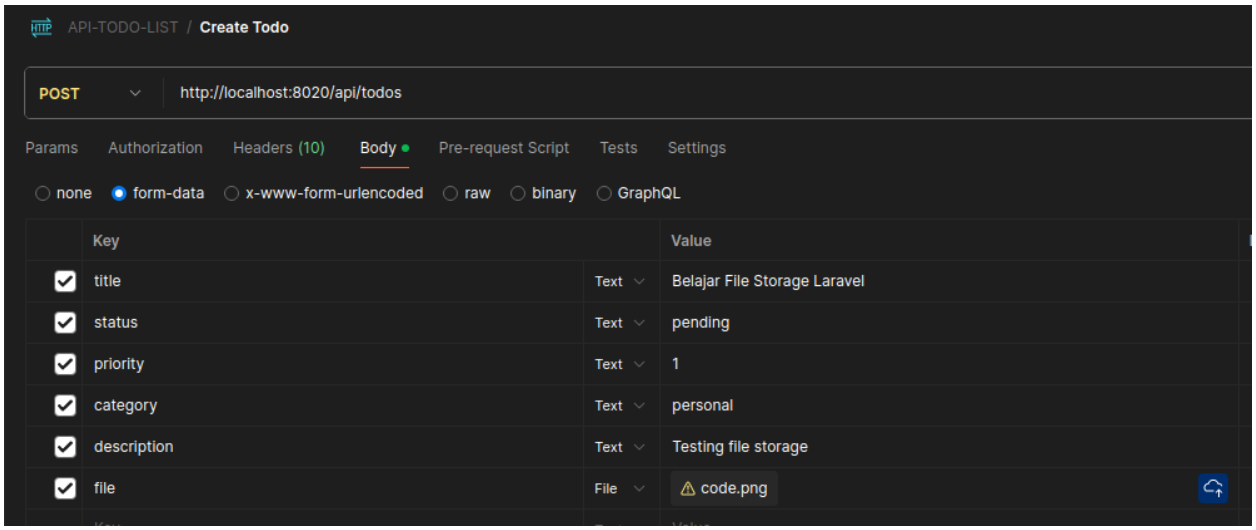
Untuk method ini sebenarnya tidak perlu diimplementasikan delete file storage, karena soft delete, jika kita merestore dan menghapus file maka file tidak akan terlihat.

Implementasi Pada Postman

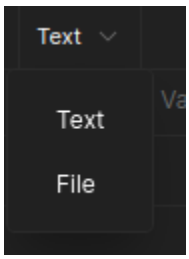
1. Ubah tipe requestnya menjadi form data



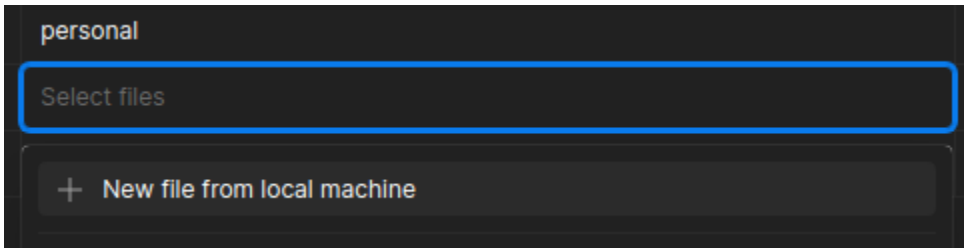
2. Isi key sesuai dengan pada json sebelumnya, dan tambahkan key file



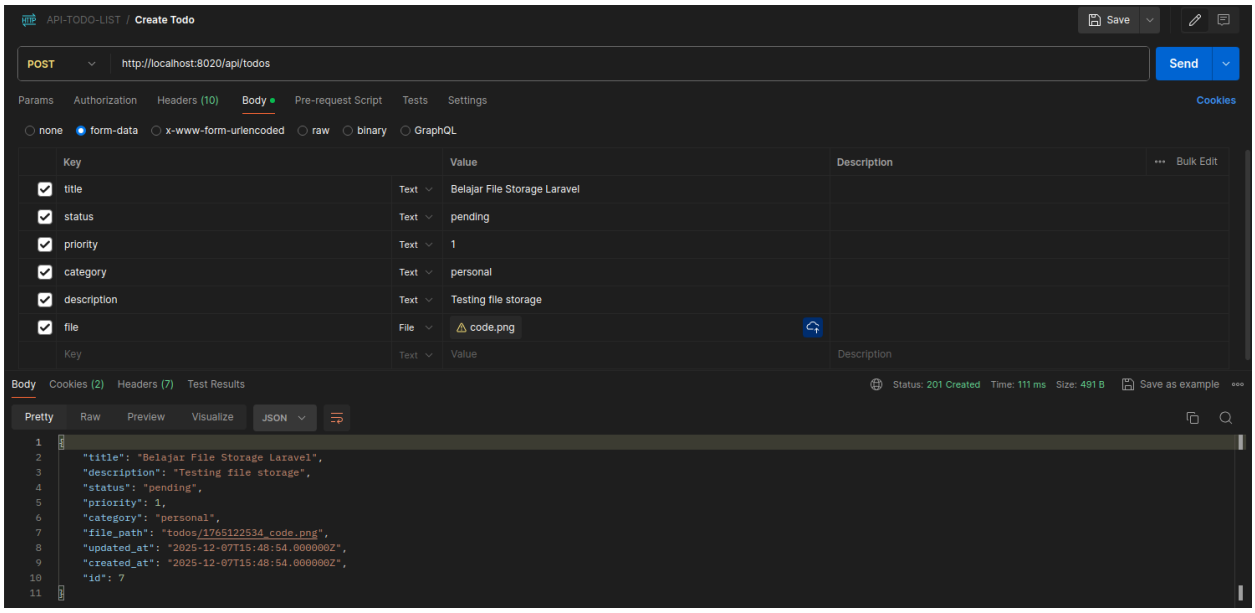
Ubah Text menjadi File



Klik select files dan pilih new file from local machine

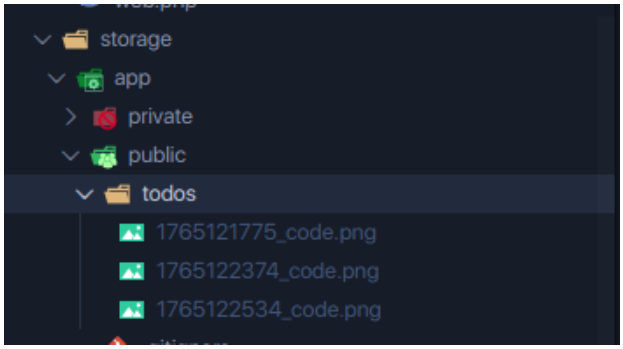


3. Klik send

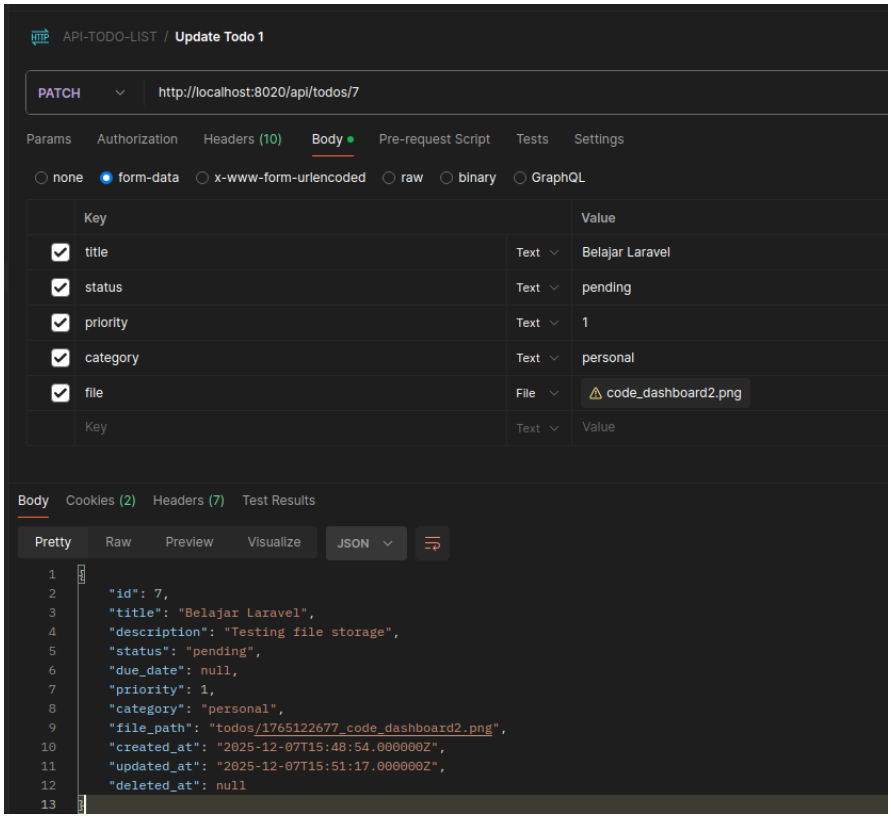


Bisa dilihat file_path memiliki ke path directory di public

4. Validasi pada local path



5. Implementasikan juga pada patch



Cara akses image melalui url local
http://localhost:8020/storage/todos/file_path (pada response)

Contoh
http://localhost:8020/storage/todos/1765122677_code_dashboard2.png

CODELAB

1. Lakukan modifikasi pada codelab sebelumnya untuk mengimplementasikan middleware dan file storage.
2. Lakukan instalasi dan implementasi middleware auth seperti pada modul (Instalasi JWT Auth di Laravel dan Implementasi JWT Auth di Laravel)
3. Lakukan instalasi dan implementasi file storage seperti pada modul (Implementasi File Storage Laravel)
4. Modifikasi collection postman dengan contoh seperti pada modul.
5. Penilaian Codelab:
    a. Jika memungkinkan selesai pada pekan materi akan mendapatkan nilai 100
    b. Jika tidak memungkinkan selesai pada pekan materi, silahkan di tunjukan pada pekan demo dan mendapatkan nilai 70.

TUGAS

Implementasikan middleware auth dan file storage dengan tema yang yang kalian gunakan dan implementasikan dengan ketentuan seperti berikut:

1. Buat endpoint untuk Login, Register, Logout dan Me, seperti pada modul.
2. Modifikasi 5 endpoint yang sudah dibuat sebelumnya dengan ketentuan berikut
    a. Get All -> Public/tanpa middleware
    b. Get Detail > Public/tanpa middleware
    c. Create -> middleware auth
    d. Update dengan param id -> middleware auth
    e. Delete dengan param id -> middleware auth
3. Implementasi file storage pada endpoint Create dan Update, ketentuan file
    a. File harus dibawah <5 MB
4. Modifikasi Collection Postman dan request http untuk masing-masing endpoint.

KRITERIA & DETAIL PENILAIAN

Kriteria Penilaian	Persentase Penilaian
Codelab	**Total 15%**
Tugas	
Mengimplementasikan Middleware Auth	20%
Mengimplementasikan File Storage	15%
Modifikasi Collection Postman	10%
Pemahaman	
Kemampuan menjelaskan tugas dan menjawab pertanyaan	40%
Total	**Total 100%**