



MODUL PRAKTIKUM WEB

**LARAVEL (N+1 Problem, Auth, CRUD,
Middleware)**

TEKNIK INFORMATIKA
FAKULTAS TEKNIK UNIVERSITAS PASUNDAN
2025/2026

DAFTAR ISI

A. Data dan Relasi Laravel 12	1
1. Relasi One to One	1
2. Relasi One to Many	1
3. N+1 Problem	3
B. Autentikasi Laravel 12	5
C. Dashboard CRUD (Create & Read)	8
D. File Upload (Storage)	19
1. Storage (Penyimpanan)	19
2. Symbolic Link (Symlink)	19
E. Validasi Data	22
TUGAS PRAKTEK	23
REFERENSI	24

A. Data dan Relasi Laravel 12

Laravel menyediakan sistem Eloquent ORM (Object Relational Mapping) untuk mempermudah pengelolaan data di database. Dengan Eloquent, setiap tabel di database diwakili oleh satu model dalam kode PHP. Salah satu fitur utama Eloquent adalah kemampuannya dalam mengatur relasi antar data.

Relasi adalah cara untuk menghubungkan satu tabel dengan tabel lainnya. Contohnya: satu user memiliki satu profile, atau satu user memiliki banyak posts. Laravel menyediakan berbagai jenis relasi, dan dua di antaranya yang paling umum adalah:

1. Relasi One to One

Relasi one to one (satu ke satu) adalah hubungan di mana satu baris data pada tabel pertama berhubungan dengan tepat satu baris pada tabel kedua. Misalnya, satu user memiliki satu profile.



The screenshot shows a code editor with a dark theme. At the top, there are three circular icons. To the right of them, the text "Contoh Relasi One to One" is displayed. The code itself is written in PHP and defines two methods: `profile()` in the `User.php` model and `user()` in the `Profile.php` model. Both methods return a relationship object using Eloquent's `hasOne` and `belongsTo` methods respectively.

```
1 // User.php
2 public function profile()
3 {
4     return $this->hasOne(Profile::class);
5 }
6
7 // Profile.php
8 public function user()
9 {
10    return $this->belongsTo(User::class);
11 }
```

2. Relasi One to Many

Relasi one to many (satu ke banyak) adalah hubungan di mana satu baris data di tabel pertama dapat memiliki banyak baris terkait di tabel kedua. Contohnya, satu user memiliki banyak posts.

Contoh Relasi One to Many

```
1 // User.php
2 public function posts()
3 {
4     return $this->hasMany(Post::class);
5 }
6
7 // Post.php
8 public function user()
9 {
10    return $this->belongsTo(User::class);
11 }
```

Mari kita praktek!

1. Silahkan buka Model “User” untuk menambahkan relasi User ke Post (One to Many)

User.php

```
public function posts()
{
    return $this->hasMany(Post::class);
}
```

2. Begitu juga dengan Model ‘Post’, kita menambahkan relasi dari Post ke User (Many to One) dan menambahkan relasi ke Category (Many to One)

Post.php

```
public function category()
{
    return $this->belongsTo(Category::class);
}

// Relasi belongsTo untuk User (author)
public function author()
{
    return $this->belongsTo(User::class, 'user_id');
}

// Tetap ada untuk backward compatibility
public function user()
{
    return $this->belongsTo(User::class);
}
```

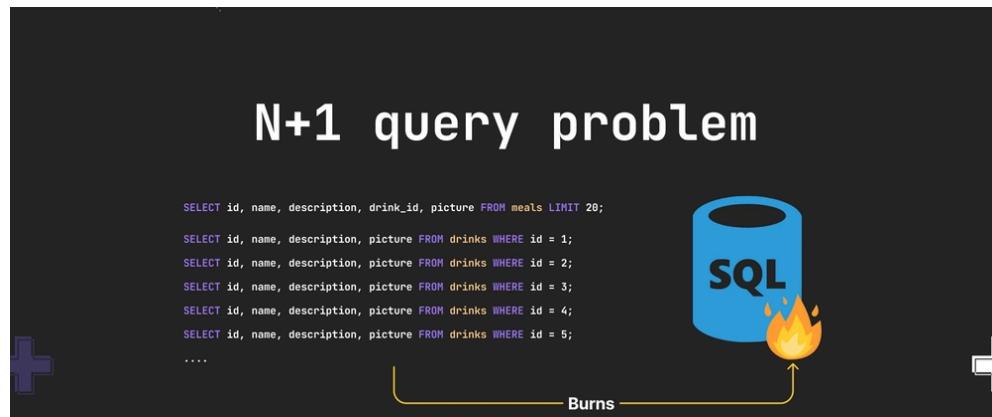
3. Begitu juga dengan ‘Category’, kita akan menambahkan relasi ke Post (One to Many)

```
Category.php

public function posts()
{
    return $this->hasMany(Post::class);
}
```

3. N+1 Problem

N+1 Problem adalah masalah di database yang membuat aplikasi Anda menjadi lambat saat Anda mengakses data relasi (seperti mengambil nama Author dari setiap Post)



Apa yang terjadi? Bayangkan Anda ingin menampilkan daftar 10 Posts beserta nama penulisnya:

1. Kueri (The "1"): Laravel mengambil 10 daftar Posts.
2. N Kueri (The "N"): Untuk setiap satu Post (total 10 Post), Laravel menjalankan satu kueri database tambahan untuk mendapatkan nama penulisnya.

Total kueri yang dijalankan adalah $1 + 10 = 11$ kueri. Jika ada 100 Post, maka akan ada 101 kueri. Ini tidak efisien dan membebani server database Anda.

Lalu Solusinya? Solusi (Cepat) Gunakan Eager Loading dengan method `with()` di Eloquent.

```
Post::with(['author', 'category'])->get();
```

Untuk melihat langsung berapa banyak kueri yang Anda jalankan dan memantau performa, **instal Laravel Debugbar** sekarang:

“composer require barryvdh/laravel-debugbar”

Mari kita praktek!

1. Silahkan install package PHP ini “composer require barryvdh/laravel-debugbar”, lalu lihat di Tab Queries. Pastikan didalam database terdapat beberapa dummy data
2. Kita akan modifikasi tugas pertemuan sebelumnya dengan N+1 Problem
3. “PostController” lalu kita perbaiki N+1 Problem menggunakan Eager Loading (`with()`) dan Lazy Eager Loading (`load()`)

```
PostController.php

public function index()
{
    // Menggunakan with() untuk mengatasi N+1 Problem
    $posts = Post::with(['author', 'category'])->get();
    return view('posts', compact('posts'));
}

// Route Model Binding untuk single post page
public function show(Post $post)
{
    // Menggunakan with() untuk mengatasi N+1 Problem
    $post->load(['author', 'category']);
    return view('post', compact('post'));
}
```

4. Lalu lihat kembali di Tab Queries laravel debug, seharusnya N+1 Problem telah teratas

B. Autentikasi Laravel 12

Autentikasi (Auth) adalah fitur wajib yang memverifikasi identitas pengguna (proses **Login**) dan mencatat informasi mereka (**Registrasi/Daftar**). Di Laravel, proses Auth dibangun di atas beberapa pilar utama:

Komponen	Tujuan
Guard	Penjaga yang menentukan bagaimana pengguna diautentikasi (misalnya, menyimpan sesi di browser setelah login).
User Provider	Mekanisme untuk mengambil data pengguna (seperti nama, <i>password</i>) dari <i>database</i> (umumnya dari tabel <code>users</code>).
Facades Auth	Jalan pintas (kelas statis) yang mempermudah Anda melakukan operasi Auth, seperti <code>Auth::attempt()</code> untuk mencoba login atau <code>Auth::user()</code> untuk mendapatkan data pengguna yang sedang login.
Middleware auth	Pagar pembatas yang melindungi <i>route</i> (halaman) tertentu agar hanya bisa diakses oleh pengguna yang sudah login.

- **Inti Konsep (Tanpa Starter Kit)**

Untuk membuat sistem Auth secara manual, Anda hanya perlu berinteraksi dengan komponen di atas:

1. **Registrasi:** Simpan data pengguna baru ke tabel `users` (User Provider).
2. **Login:** Gunakan `Auth::attempt()` untuk memverifikasi email/username dan *password* pengguna (Guard).
3. **Logout:** Gunakan `Auth::logout()` untuk menghapus sesi pengguna (Guard)..

Mari kita praktek!

1. Buat routes baru di web.php, routes login, register, dashboard, logout

```
// Protect posts dan categories dengan auth middleware
// Route dari laraveltest-main
Route::get('/posts', [PostController::class, 'index'])->middleware('auth')->name('posts.index');

// Route Model Binding dengan slug
Route::get('/posts/{post:slug}', [PostController::class, 'show'])->middleware('auth')->name('posts.show');

// Route untuk register - middleware guest (hanya untuk yang belum login)
Route::get('/register', [RegisterController::class, 'showRegistrationForm'])->middleware('guest')->name('register');
Route::post('/register', [RegisterController::class, 'register'])->middleware('guest');

// Route untuk login - middleware guest (hanya untuk yang belum login)
Route::get('/login', [LoginController::class, 'showLoginForm'])->middleware('guest')->name('Login');
Route::post('/login', [LoginController::class, 'login'])->middleware('guest');

// Route logout - hanya untuk yang sudah login
Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
```

2. Kita akan membuat controller LoginController, RegisterController dan DashboardController menggunakan perintah:

```
php artisan make:controller RegisterController
php artisan make:controller LoginController
```

3. Silahkan buat halaman form login.blade.php (terdapat input email, password) dan form register.blade.php (terdapat input nama, email, password, konfirmasi password)
4. Setelah dibuat, maka buka LoginController, dan kita buat method login dan logout didalamnya

```
<?php

namespace App\Http\Controllers;

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

class LoginController extends Controller
{

    public function showLoginForm()
    {
        return view('login');
    }

    public function login(Request $request)
    {
        // Validasi input
        $request->validate([
            'email' => 'required|email',
            'password' => 'required',
        ]);

        // Logic login dengan Auth::attempt
        if (Auth::attempt($request->only('email', 'password'))) {
            // Login berhasil
            $request->session()->regenerate();
            return redirect()->intended('/posts');
        }

        // Login gagal
        return back()->withErrors([
            'email' => 'Email atau password salah.',
        ]);
    }

    public function logout(Request $request)
    {
        Auth::logout();
        $request->session()->invalidate();
        $request->session()->regenerateToken();
        return redirect('/');
    }
}
```

5. Setelah dibuat, maka buka RegisterController, dan kita buat method register didalamnya

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class RegisterController extends Controller
{
    // MODUL 2-2 START - Authentikasi Manual Sederhana
    public function showRegistrationForm()
    {
        return view('register');
    }

    public function register(Request $request)
    {
        // Validasi input
        $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8|confirmed',
        ]);

        // Logic register: validasi, hash password, User::create
        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        // Redirect ke login setelah register berhasil
        return redirect('/login')->with('success', 'Registrasi berhasil! Silakan login.');
    }
    // MODUL 2-2 END
}
```

6. Setelah dibuat, mari kita buat

Silakan daftar akun dan login terlebih dahulu. Setelah itu, coba akses halaman *posts*. Halaman tersebut seharusnya sudah bisa diakses dan aman karena telah dilindungi oleh *middleware auth*.

C. Dashboard CRUD (Create & Read)

Dashboard CRUD adalah halaman utama atau panel kontrol di aplikasi web yang menyediakan antarmuka untuk mengelola seluruh data dalam sebuah tabel (misalnya, tabel *Posts* atau *Categories*). CRUD sendiri adalah singkatan dari :

- Create (Membuat data baru)
- Read (Melihat daftar data)
- Update (Mengubah data)

- Delete (Menghapus data)

Dibagian ini kita akan fokus pada **Create & Read** terlebih dahulu:

1. **Create (Buat)**: Pengguna bisa mengisi formulir dan membuat Postingan baru, di mana Postingan baru ini akan otomatis terhubung dengan ID pengguna yang sedang login.
2. **Read (Baca)**: Pengguna bisa melihat semua Postingan yang *hanya* mereka buat (bukan postingan pengguna lain).

Mari kita praktek!

1. Kita akan membuat PostDashboardController Terlebih dahulu PostDashboardController menggunakan perintah:

```
console  
php artisan make:controller DashboardPostController --resource
```

Kenapa menggunakan --resouce di ujung perintah nya? agar semua fungsi CRUD (index, create, store, edit, update, destroy) sudah disiapkan.

2. Setelah itu kita akan membuat layout untuk dashboard nya

```
console  
php artisan make:component dashboard-layout
```

lalu copy paste isi kode yang ada di layout.blade.php

3. Setelah di buat copy paste isi kode yang ada di layout.blade.php lalu simpan di dashboard-layout.blade.php
4. Lalu jalan kan perintah ini di terminal yang baru untuk instal flowbite
<https://flowbite.com/docs/getting-started/laravel/>:

```
Terminal  
npm install flowbite --save
```

- Setelah itu kita akan menambahkan @import, @plugin, dan @source dari flowbite dan simpan di file app.css:

```

app.css

@import 'tailwindcss';
@import "flowbite/src/themes/default";

@plugin "flowbite/plugin";

@source
'../../vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php';
@source '../../storage/framework/views/*.php';
@source '../**/*.blade.php';
@source '../**/*.js';
@source "../../node_modules/flowbite";

@theme {
    --font-sans: 'Instrument Sans', ui-sans-serif, system-ui, sans-serif, 'Apple Color Emoji',
    'Segoe UI Emoji',
        'Segoe UI Symbol', 'Noto Color Emoji';
}

```

- Lalu tambahkan script cdn dari flowbite dan simpan di dashboard-layout.blade.php

```

dashboard-layout.blade.php

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    {{-- Tambahkan slot baru dengan nama $title --}}
    <title>{{ $title }}</title>
    @vite('resources/css/app.css')
    {{-- flowbite --}}
    <script src="https://cdn.jsdelivr.net/npm/flowbite@0.1.0/dist/flowbite.min.js"></script>
</head>

```

- Lalu buat folder dashboard di dalam view, setelah itu buat file index dengan nama file index.blade.php

```

dashboard.blade.php

<x-dashboard-layout>
    <x-slot:title>
        Dashboard
    </x-slot:title>

    <h1 class="text-4xl font-bold text-gray-800 mb-4">Welcome, {{ auth()>user()>name }}</h1>

    @include('components.table')
</x-dashboard-layout>

```

- Sekarang kita modifikasi class index di DashboardPostController.php



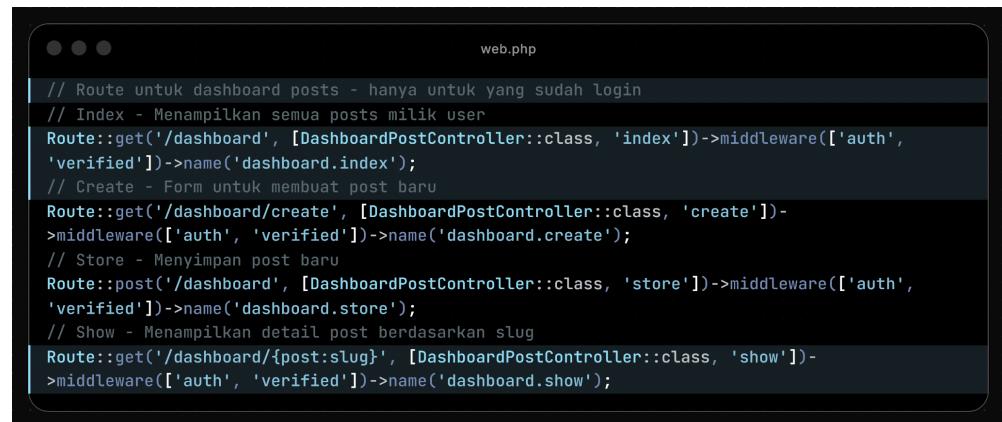
```
DashboardPostController.php

public function index()
{
    // menggunakan user_id dari user yang sedang login
    $posts = Post::where('user_id', auth()->user()->id);

    // fitur search
    if (request('search')) {
        $posts->where('title', 'like', '%' . request('search') . '%');
    }

    // menampilkan 5 data per halaman dengan pagination
    return view('dashboard.index', ['posts' => $posts->paginate(5)->withQueryString()]);
}
```

9. Tambahkan route untuk dashboard di web.php



```
web.php

// Route untuk dashboard posts - hanya untuk yang sudah login
// Index - Menampilkan semua posts milik user
Route::get('/dashboard', [DashboardPostController::class, 'index'])->middleware(['auth', 'verified'])->name('dashboard.index');
// Create - Form untuk membuat post baru
Route::get('/dashboard/create', [DashboardPostController::class, 'create'])->middleware(['auth', 'verified'])->name('dashboard.create');
// Store - Menyimpan post baru
Route::post('/dashboard', [DashboardPostController::class, 'store'])->middleware(['auth', 'verified'])->name('dashboard.store');
// Show - Menampilkan detail post berdasarkan slug
Route::get('/dashboard/{post:slug}', [DashboardPostController::class, 'show'])->middleware(['auth', 'verified'])->name('dashboard.show');
```

10. Sekarang kita akan membuat components untuk table yang ada di dashboard menggunakan template dari flowbite

<https://flowbite.com/docs/components/tables/> setelah paste template tersebut ke file teable.blade.php yang ada di folder component, silahkan untuk modifikasi table tersebut dengan kolom (No, Title, Author, Category, Published At, Action).

```

table.blade.php

{{-- Table --}}
<div class="relative overflow-x-auto bg-neutral-primary-soft shadow-xs rounded-base border border-default">
    <table class="w-full text-sm text-left rtl:text-right text-body">
        <thead class="text-sm text-body bg-neutral-secondary-soft border-b rounded-base border-default">
            <tr>
                <th scope="col" class="px-6 py-3 font-medium">
                    No
                </th>
                <th scope="col" class="px-6 py-3 font-medium">
                    Title
                </th>
                <th scope="col" class="px-6 py-3 font-medium">
                    Category
                </th>
                <th scope="col" class="px-6 py-3 font-medium">
                    Published At
                </th>
                <th scope="col" class="px-6 py-3 font-medium">
                    Actions
                </th>
            </tr>
        </thead>
        <tbody>
            @forelse($posts as $post)
                <tr class="bg-neutral-primary border-b border-default">
                    <td class="px-6 py-4">
                        {{ $posts->firstItem() + $loop->index }}
                    </td>
                    <th scope="row" class="px-6 py-4 font-medium text-heading whitespace nowrap">
                        {{ $post->title }}
                    </th>
                    <td class="px-6 py-4">
                        {{ $post->category->name ?? 'Uncategorized' }}
                    </td>
                    <td class="px-6 py-4">
                        {{ $post->created_at->format('d M Y') }}
                    </td>
                    <td class="px-6 py-4">
                        <a href="{{ route('dashboard.show', $post->slug) }}"
                            class="text-blue-600 hover:underline">View</a>
                    </td>
                </tr>
            @empty
                <tr>
                    <td colspan="6" class="px-6 py-12 text-center text-gray-500">
                        No posts yet. <a href="{{ route('dashboard.create') }}"
                            class="text-blue-600 hover:underline">Create one</a>
                    </td>
                </tr>
            @endforelse
        </tbody>
    </table>
</div>

```

11. Setelah itu di atas table tersebut tambahkan Search dan Button untuk Menambahkan Post di bagian header:

```
table.blade.php

{{-- Header with Search and Add Post Button --}}
<div class="px-6 py-4 border-b border-gray-200 flex justify-between items-center gap-4 bg-gradient-to-r from-blue-50 to-indigo-50">
    <form method="GET" action="{{ route('dashboard.index') }}" class="flex-1 max-w-md">
        <label for="search" class="sr-only">Search</label>
        <div class="relative">
            <div class="absolute inset-y-0 start-0 flex items-center ps-3 pointer-events-none">
                <svg class="w-4 h-4 text-body" aria-hidden="true"
                    xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" viewBox="0 0 24 24">
                    <path stroke="currentColor" stroke-linecap="round" stroke-width="2"
                        d="m21 21-3.5-3.5M17 10a7 7 0 1 1-14 0 7 7 0 0 1 14 0Z"/>
                </svg>
            </div>
            <input type="search" name="search" id="search" value="{{ request('search') }}"
                class="block w-full p-3 ps-9 bg-neutral-secondary-medium border border-default-medium text-heading text-sm rounded-base focus:ring-brand focus:border-brand shadow-xs placeholder:text-body" placeholder="Search posts..." />
            <button type="submit" class="absolute end-1.5 bottom-1.5 text-white bg-brand
                hover:bg-brand-strong box-border border-transparent focus:ring-4 focus:ring-brand-medium shadow-xs font-medium leading-5 rounded text-xs px-3 py-1.5 focus:outline-none">Search</button>
        </div>
    </form>

    <a href="{{ route('dashboard.create') }}"
        class="inline-flex items-center px-4 py-2 bg-blue-600 hover:bg-blue-700 text-white
        font-medium rounded-lg shadow-sm transition-colors duration-200 whitespace nowrap">
        <svg class="w-5 h-5 mr-2" fill="none" stroke="currentColor" viewBox="0 0 24 24">
            <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12
                4v16m8-8H4"/>
        </svg>
        Add Post
    </a>
</div>
```

12. Lalu tambahkan pagination menggunakan template dari flowbite:

```
table.blade.php

{{-- Pagination --}}
@if($posts->hasPages())
    <div class="px-6 py-4 border-t border-gray-200">
        <nav aria-label="Page navigation">
            <ul class="flex -space-x-px text-sm">
                {{-- Previous Button --}}
                    @if($posts->onFirstPage())
                        <li>
                            <span class="flex items-center justify-center text-gray-400 bg-gray-100 box-border border-gray-300 cursor-not-allowed font-medium rounded-s-base text-sm px-3 h-10">Previous</span>
                        </li>
                    @else
                        <li>
                            <a href="{{ $posts->previousPageUrl() }}" class="flex items-center justify-center text-body bg-neutral-secondary-medium box-border border-default-medium hover:bg-neutral-tertiary-medium hover:text-heading font-medium rounded-s-base text-sm px-3 h-10 focus:outline-none">Previous</a>
                        </li>
                    @endif
                {{-- Page Numbers --}}
                    @foreach($posts->getUrlRange(1, $posts->lastPage()) as $page => $url)
                        @if($page == $posts->currentPage())
                            <li>
                                <a href="{{ $url }}" aria-current="page" class="flex items-center justify-center text-fg-brand bg-neutral-tertiary-medium box-border border-default-medium hover:text-fg-brand font-medium text-sm w-10 h-10 focus:outline-none">{{ $page }}</a>
                            </li>
                        @else
                            <li>
                                <a href="{{ $url }}" class="flex items-center justify-center text-body bg-neutral-secondary-medium box-border border-default-medium hover:text-heading font-medium text-sm w-10 h-10 focus:outline-none">{{ $page }}</a>
                            </li>
                        @endif
                    @endforeach
                {{-- Next Button --}}
                    @if($posts->hasMorePages())
                        <li>
                            <a href="{{ $posts->nextPageUrl() }}" class="flex items-center justify-center text-body bg-neutral-secondary-medium box-border border-default-medium hover:bg-neutral-tertiary-medium hover:text-heading font-medium rounded-e-base text-sm px-3 h-10 focus:outline-none">Next</a>
                        </li>
                    @else
                        <li>
                            <span class="flex items-center justify-center text-gray-400 bg-gray-100 box-border border-gray-300 cursor-not-allowed font-medium rounded-e-base text-sm px-3 h-10">Next</span>
                        </li>
                    @endif
                
            </nav>
        </div>
    @endif
```

13. Selanjutnya kita akan membuat halaman show.blade.php di dalam folder dashboard, isi nya copy paste saja dari file detailpost.blade.php lalu modifikasi isinya:

```
show.blade.php

<x-dashboard-layout>
    <x-slot:title>
        {{ $post->title }} - Dashboard
    </x-slot:title>

    <article class="max-w-4xl mx-auto">
        <header class="mb-8">
            <h1 class="text-4xl font-bold text-gray-800 mb-4">{{ $post->title }}</h1>

            <div class="flex items-center text-sm text-gray-600 mb-4">
                <span class="mr-4">By {{ $post->author->name ?? auth()->user()->name }}</span>
                <span class="mr-4">Category: {{ $post->category->name ?? 'Uncategorized' }}</span>
            </div>

            <span>{{ $post->created_at->format('d M Y') }}</span>
        </header>

        <div class="prose prose-lg max-w-none">
            <p class="text-xl text-gray-600 mb-6">{{ $post->excerpt }}</p>
            <div class="text-gray-800 leading-relaxed">
                {!! nl2br(e($post->body)) !!}
            </div>
        </div>

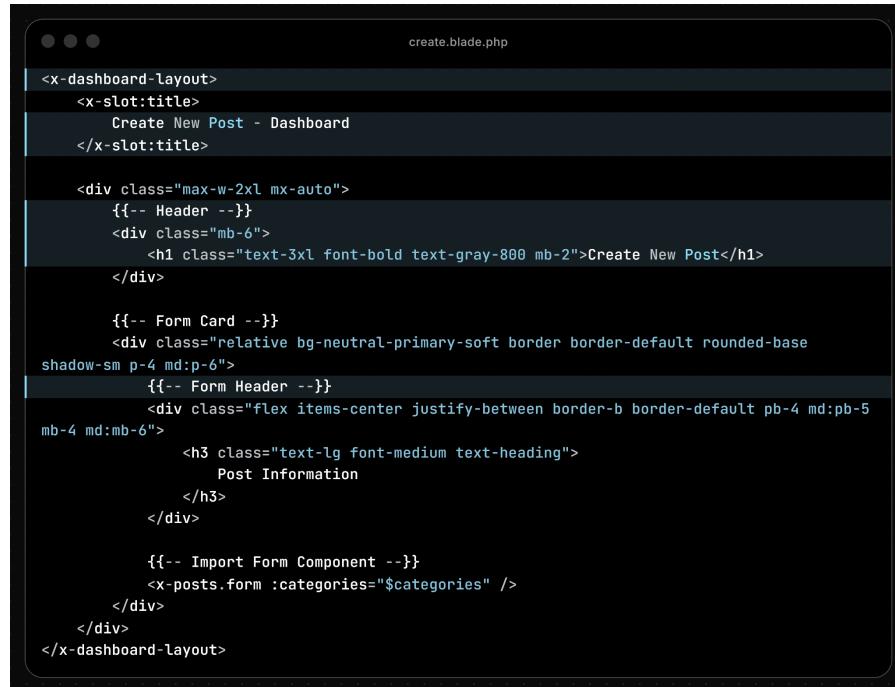
        <footer class="mt-8 pt-8 border-t border-gray-200">
            <div class="flex justify-between items-center">
                <a href="{{ route('dashboard.index') }}"
                    class="inline-flex items-center text-blue-600 hover:text-blue-800 font-medium">
                    ← Back to Dashboard
                </a>
            </div>
        </footer>
    </article>
</x-dashboard-layout>
```

14. Setelah itu modifikasi DashboardPostController.php method show:

```
DashboardPostController.php

public function show(Post $post)
{
    return view('dashboard.show', ['post' => $post]);
}
```

15. Selanjutnya membuat halaman untuk menambah Post, buat terlebih dahulu file create.blade.php di dalam folder dashboard:



```
<x-dashboard-layout>
    <x-slot:title>
        Create New Post - Dashboard
    </x-slot:title>

    <div class="max-w-2xl mx-auto">
        {{-- Header --}}
        <div class="mb-6">
            <h1 class="text-3xl font-bold text-gray-800 mb-2">Create New Post</h1>
        </div>

        {{-- Form Card --}}
        <div class="relative bg-neutral-primary-soft border border-default rounded-base shadow-sm p-4 md:p-6">
            {{-- Form Header --}}
            <div class="flex items-center justify-between border-b border-default pb-4 md:pb-5 mb-4 md:mb-6">
                <h3 class="text-lg font-medium text-heading">
                    Post Information
                </h3>
            </div>

            {{-- Import Form Component --}}
            <x-posts.form :categories="$categories" />
        </div>
    </div>
</x-dashboard-layout>
```

16. Setelah itu buat file form.blade.php di direktori components/posts, buka <https://flowbite.com/docs/components/modal/> untuk template form nya, dan modifikasi form nya:

```

    form.balde.php.php

@props(['categories'])

{{-- Form Body --}}
<form action="{{ route('dashboard.store') }}" method="POST">
    @csrf
    <div class="grid gap-4 grid-cols-2">
        {{-- Title --}}
        <div class="col-span-2">
            <label for="title" class="block mb-2.5 text-sm font-medium text-
heading">Title</label>
            <input type="text" name="title" id="title" value="{{ old('title') }}" class="bg-
neutral-secondary-medium border border-default-medium text-heading text-sm rounded-base
focus:ring-brand focus:border-brand block w-full px-3 py-2.5 shadow-xs placeholder:text-body"
placeholder="Enter post title">
        </div>

        {{-- Category --}}
        <div class="col-span-2">
            <label for="category_id" class="block mb-2.5 text-sm font-medium text-
heading">Category</label>
            <select name="category_id" id="category_id" class="block w-full px-3 py-2.5 bg-
neutral-secondary-medium border border-default-medium text-heading text-sm rounded-base
focus:ring-brand focus:border-brand shadow-xs placeholder:text-body">
                <option value="">Select category</option>
                @foreach($categories as $category)
                    <option value="{{ $category->id }}>{{ old('category_id') == $category->id
? 'selected' : '' }}>{{ $category->name }}</option>
                @endforeach
            </select>
        </div>

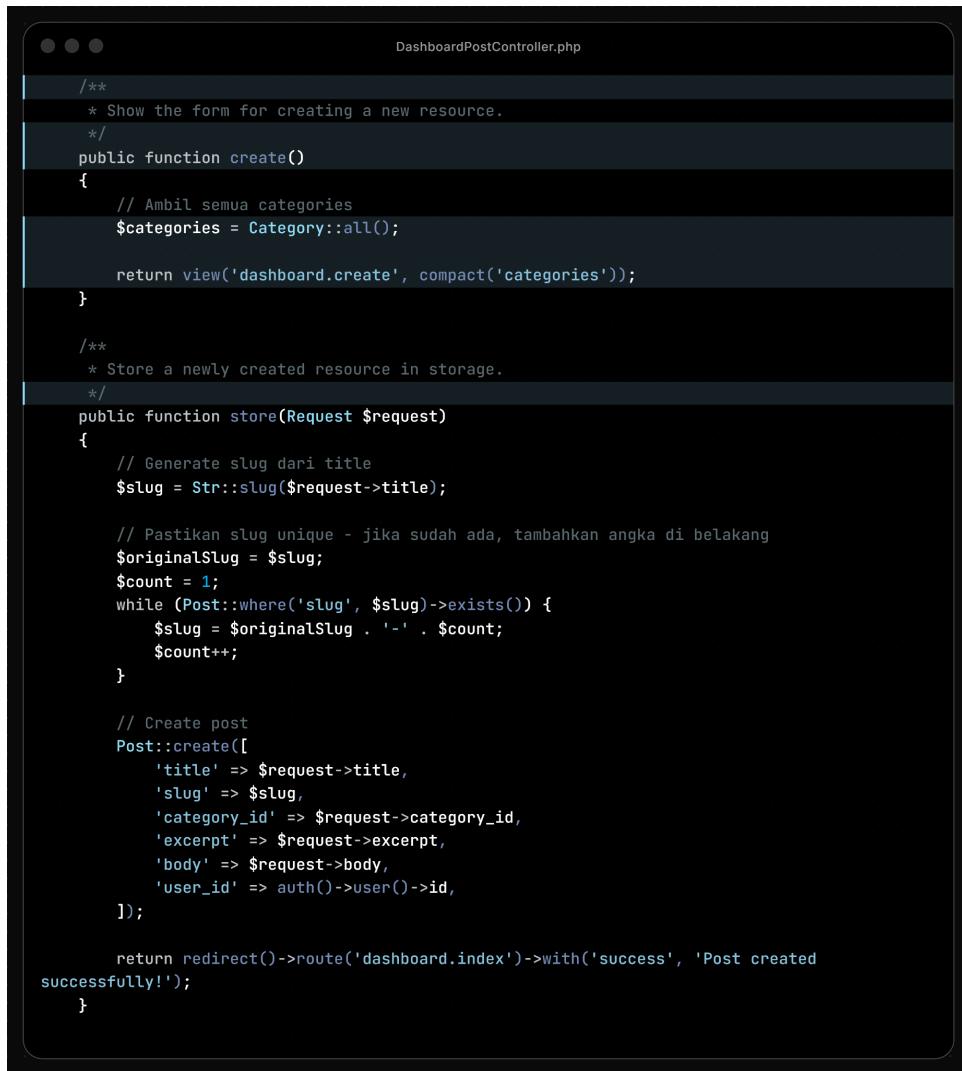
        {{-- Excerpt --}}
        <div class="col-span-2">
            <label for="excerpt" class="block mb-2.5 text-sm font-medium text-
heading">Excerpt</label>
            <textarea name="excerpt" id="excerpt" rows="3" class="block bg-neutral-secondary-
medium border border-default-medium text-heading text-sm rounded-base focus:ring-brand
focus:border-brand w-full p-3.5 shadow-xs placeholder:text-body" placeholder="Write a short
excerpt or summary">{{ old('excerpt') }}</textarea>
        </div>

        {{-- Body --}}
        <div class="col-span-2">
            <label for="body" class="block mb-2.5 text-sm font-medium text-
heading">Content</label>
            <textarea name="body" id="body" rows="8" class="block bg-neutral-secondary-medium
border border-default-medium text-heading text-sm rounded-base focus:ring-brand focus:border-
brand w-full p-3.5 shadow-xs placeholder:text-body" placeholder="Write your post content
here">{{ old('body') }}</textarea>
        </div>

        {{-- Form Footer --}}
        <div class="flex items-center space-x-4 border-t border-default pt-4 md:pt-6 mt-4 md:mt-
6">
            <button type="submit" class="inline-flex items-center text-white bg-brand hover:bg-
brand-strong box-border border border-transparent focus:ring-4 focus:ring-brand-medium shadow-
xs font-medium leading-5 rounded-base text-sm px-4 py-2.5 focus:outline-none">
                Create Post
            </button>
            <a href="{{ route('dashboard.index') }}" class="text-body bg-neutral-secondary-medium
box-border border border-default-medium hover:bg-neutral-tertiary-medium hover:text-heading
focus:ring-4 focus:ring-neutral-tertiary shadow-xs font-medium leading-5 rounded-base text-sm
px-4 py-2.5 focus:outline-none">
                Cancel
            </a>
        </div>
    </form>

```

17. sekarang kita modifikasi method create dan store di
DashboardPostController.php



```
DashboardPostController.php

/*
 * Show the form for creating a new resource.
 */
public function create()
{
    // Ambil semua categories
    $categories = Category::all();

    return view('dashboard.create', compact('categories'));
}

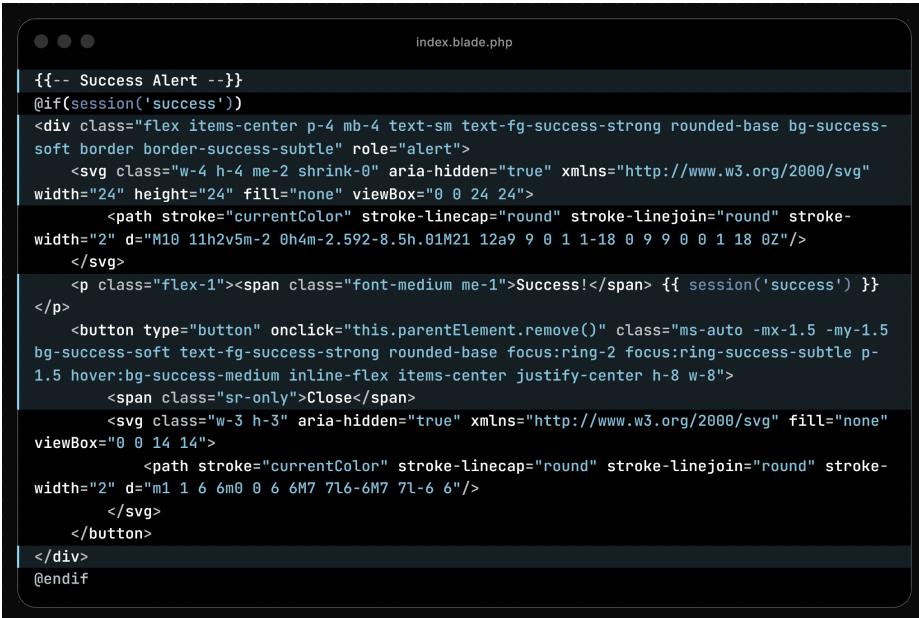
/*
 * Store a newly created resource in storage.
 */
public function store(Request $request)
{
    // Generate slug dari title
    $slug = Str::slug($request->title);

    // Pastikan slug unique - jika sudah ada, tambahkan angka di belakang
    $originalSlug = $slug;
    $count = 1;
    while (Post::where('slug', $slug)->exists()) {
        $slug = $originalSlug . '-' . $count;
        $count++;
    }

    // Create post
    Post::create([
        'title' => $request->title,
        'slug' => $slug,
        'category_id' => $request->category_id,
        'excerpt' => $request->excerpt,
        'body' => $request->body,
        'user_id' => auth()->user()->id,
    ]);

    return redirect()->route('dashboard.index')->with('success', 'Post created successfully!');
}
```

18. selanjutnya tambahkan error message di file index.blade.php yang berada dalam folder dashboard, untuk template nya di
<https://flowbite.com/docs/components/alerts/>



A screenshot of a web browser window titled "index.blade.php". The content shows a success alert message with an "X" button to close it. The code in the browser's developer tools shows a PHP conditional block (@if(session('success'))), followed by HTML and SVG code for the alert box, and a button to close it.

```
index.blade.php

{{-- Success Alert --}}
@if(session('success'))
<div class="flex items-center p-4 mb-4 text-sm text-fg-success-strong rounded-base bg-success-soft border border-success-subtle" role="alert">
    <svg class="w-4 h-4 me-2 shrink-0" aria-hidden="true" xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" viewBox="0 0 24 24">
        <path stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M10 11h2v5m-2 0h4m-2.592-8.5h.01M21 12a9 9 0 1 1-18 0 9 9 0 0 1 18 0z"/>
    </svg>
    <p class="flex-1"><span class="font-medium me-1">Success!</span> {{ session('success') }}</p>
    <button type="button" onclick="this.parentElement.remove()" class="ms-auto -mx-1.5 -my-1.5 bg-success-soft text-fg-success-strong rounded-base focus:ring-2 focus:ring-success-subtle p-1.5 hover:bg-success-medium inline-flex items-center justify-center h-8 w-8">
        <span class="sr-only">Close</span>
        <svg class="w-3 h-3" aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 14 14">
            <path stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="m1 1 6m0 0 6M7 7l6-M7 7l-6 6"/>
        </svg>
    </button>
</div>
@endif
```

Silahkan coba untuk menambah postingan terlebih dahulu, dan cek apakah postingannya bertambah yang di route/dashboard lalu cek detail postingan yang baru di buat.

D. File Upload (Storage)

1. Storage (Penyimpanan)

Di Laravel, *Storage* adalah sistem yang mengatur tempat file-file yang di-*upload* pengguna akan diletakkan. Secara *default*, file akan disimpan di folder **storage/app/public**. Folder ini terlindungi dan tidak bisa diakses langsung melalui URL publik, yang meningkatkan keamanan aplikasi Anda.

2. Symbolic Link (Symlink)

Ini adalah *jembatan* atau *pintasan* virtual. Karena file Anda aman di folder **storage/app/public** (yang tidak bisa diakses publik), Anda perlu membuat tautan agar *browser* bisa mengambil file tersebut.

- **Peran:** Perintah `php artisan storage:link` membuat *symbolic link* dari **public/storage** yang mengarah ke **storage/app/public**.
- **Hasil:** Anda bisa mengakses gambar Anda melalui URL publik, seperti https://nama_project/storage/namafile.jpg.

Mari kita praktek!

1. Melanjutkan file form.blade.php yang sebelumnya, kita tambahkan attribute `encType="multipart/formdata"` pada form, karena data yang dikirim ada data teks dan data file yang dikirim melalui tipe data file.
2. setelah ditambahkan sekarang tambahkan form upload dari flowbite <https://flowbite.com/docs/forms/file-input/>.
3. selanjutnya kita akan menambahkan kode dari flowbite tersebut setelah field input "Content", lalu modifikasi kodennya agar sesuai dengan apa yang kita akan buat:

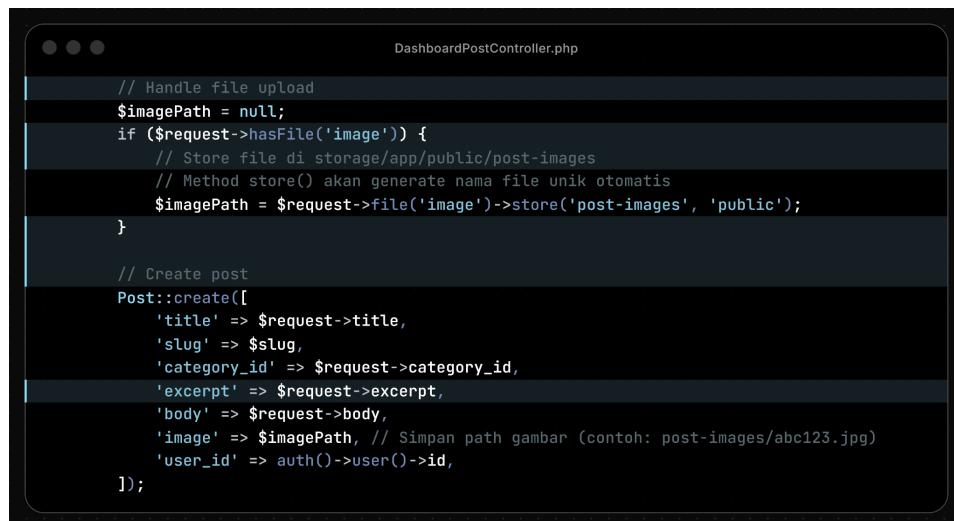


```
form.blade.php.php
{{-- Image Upload --}}


<label for="image" class="block mb-2.5 text-sm font-medium text-heading">Upload Image</label>
    <input type="file" name="image" id="image" accept="image/png, image/jpeg, image/jpg" class="cursor-pointer bg-neutral-secondary-medium border text-heading text-sm rounded-base focus:ring-brand focus:border-brand block w-full shadow-xs placeholder:text-body">


```

4. lalu sekarang modifikasi method store di DashboardPostController.php



```
DashboardPostController.php
// Handle file upload
$imagePath = null;
if ($request->hasFile('image')) {
    // Store file di storage/app/public/post-images
    // Method store() akan generate nama file unik otomatis
    $imagePath = $request->file('image')->store('post-images', 'public');
}

// Create post
Post::create([
    'title' => $request->title,
    'slug' => $slug,
    'category_id' => $request->category_id,
    'excerpt' => $request->excerpt,
    'body' => $request->body,
    'image' => $imagePath, // Simpan path gambar (contoh: post-images/abc123.jpg)
    'user_id' => auth()->user()->id,
]);
```

5. selanjutnya kita akan jalankan perintah ini di terminal:

terminal

```
php artisan storage:link
```

6. sebelum itu kita akan download gambar terlebih dahulu di <https://pin.it/1cd2nHjfX> lalu simpan dengan nama file preview.jpg
7. simpan gambar preview.jpg di folder public/images
8. lalu modifikasi table.blade.php untuk menambahkan kolom tabel agar menampilkan semua gambar di setiap post yang telah dibuat

table.blade.php

```
// tambahkan table head image setelah table head Nomor
<th scope="col" class="px-6 py-3 font-medium">
    Image
</th>

// tambahkan table data image setelah table data Nomor
<td class="px-6 py-4">
    @if($post->image)
        title }}" class="w-16 h-16 rounded-base object-cover">
    @else
        
    @endif
</td>
```

9. Setelah itu file show.blade.php modifikasi juga agar menampilkan gambar apabila post tersebut memiliki gambar

show.blade.php

```
<header class="mb-8">
    <h1 class="text-4xl font-bold text-gray-800 mb-4">{{ $post->title }}</h1>

    <div class="flex items-center text-sm text-gray-600 mb-4">
        <span class="mr-4">By {{ $post->author->name ?? auth()->user()->name }}</span>
        <span class="mr-4">Category: {{ $post->category->name ?? 'Uncategorized' }}</span>
        <span>{{ $post->created_at->format('d M Y') }}</span>
    </div>

    @if($post->image)
        title }}"
            class="w-full h-64 object-cover rounded-lg mb-6">
    @endif
</header>
```

10. Setelah semuanya dilakukan, silahkan coba untuk membuat postingan dengan upload gambar, lalu periksa ke file halaman dashboard dan detailnya.

E. Validasi Data

Validasi adalah proses krusial untuk memastikan data yang diterima dari *form pengguna (Request)* bersih, lengkap, dan aman sebelum disimpan ke *Database*. Validasi dilakukan di *method store(Request \$request)* menggunakan Facade Validator.

Mari kita praktik!

1. Sekarang buka file DashboardPostController.php, yang nantinya akan di menambahkan Facade Validator di method store

```

DashboardPostController.php

// Validasi input dengan custom messages
$validator = Validator::make($request->all(), [
    'title' => 'required|max:255',
    'category_id' => 'required|exists:categories,id', // Memastikan ID ada di tabel categories
    'excerpt' => 'required',
    'body' => 'required',
    // Aturan untuk Image: Opsional (nullable), harus gambar, format tertentu, max 2MB
    'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
],
[
    // Custom Messages
    'title.required' => 'Field Title wajib diisi',
    'title.max' => 'Field Title tidak boleh lebih dari 255 karakter',
    'category_id.required' => 'Field Category wajib dipilih',
    'category_id.exists' => 'Category yang dipilih tidak valid',
    'excerpt.required' => 'Field Excerpt wajib diisi',
    'body.required' => 'Field Content wajib diisi',
    'image.image' => 'File harus berupa gambar',
    'image.mimes' => 'Format gambar harus jpeg, png, jpg, atau gif',
    'image.max' => 'Ukuran gambar maksimal 2MB',
]
);

```

2. selanjutnya tambahkan Failure Handling di method store

DashboardPostController.php

```

DashboardPostController.php

// Jika validasi gagal, redirect kembali dengan error
if ($validator->fails()) {
    return redirect()->back()
        ->withErrors($validator) // Mengirimkan semua pesan error kembali
        ->withInput(); // Mengirimkan semua data yang sudah diinput (old data)
}

```

3. lalu kita akan menampilkan pesan apabila terjadi error di setiap fiel di file form.blade.php

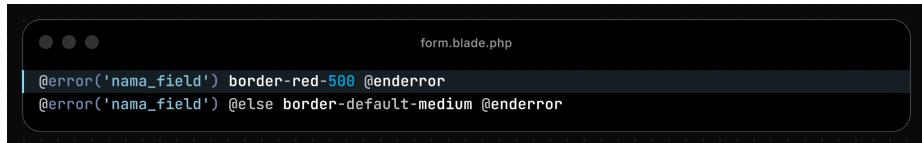


```
form.blade.php

{{-- contoh --}}
<div>
    @error('nama_field')
        <p class="mt-1 text-sm text-red-600">{{ $message }}</p>
    @enderror
</div>

{{-- implementasi pada field tittle --}}
<div class="col-span-2">
    <label for="title" class="block mb-2.5 text-sm font-medium text-heading">Title</label>
    <input type="text" name="title" id="title" value="{{ old('title') }}" class="bg-neutral-secondary-medium border border-default-medium text-heading text-sm rounded-base focus:ring-brand focus:border-brand block w-full px-3 py-2.5 shadow-xs placeholder:text-body" placeholder="Enter post title">
    @error('title')
        <p class="mt-1 text-sm text-red-600">{{ $message }}</p>
    @enderror
</div>
```

4. setelah itu tambahkan juga dalam atribut **class** dari setiap *field input*



```
form.blade.php

@error('nama_field') border-red-500 @enderror
@error('nama_field') @else border-default-medium @enderror
```

setelah menambahkan validasi dan pesan ketika error, silahkan untuk mencoba isi form dengan data yang tidak sesuai dengan validasi. seharusnya ketika dicoba akan memunculkan pesan dari validasi dari method store.

TUGAS PRAKTEK

1. Buatkan **CRUD** Category
2. Selesaikan Fungsionalitas **Update Posts**.
3. Selesaikan Fungsionalitas **Delete Posts**.

REFERENSI

Playlist Laravel	https://www.youtube.com/watch?v=00o1vJYTp4I&list=PLFIM0718LjIWIXb7cVj7LdAr32ATDQMdr
WPUCourse - Belajar Laravel	https://wpucourse.id/course/belajar-laravel