FITUR OTORISASI BERTINGKAT, CONTROLLER DAN VIEW

Pada bab sebelumnya, kita telah berhasil membangun fondasi yang kuat untuk aplikasi kita dengan mengimplementasikan fitur login atau autentikasi menggunakan Laravel Breeze. Fitur ini memungkinkan pengguna untuk masuk ke dalam sistem dan mengakses berbagai fungsionalitas yang disediakan. Namun, dalam banyak aplikasi, kita seringkali perlu membedakan level akses pengguna berdasarkan peran (role) mereka. Misalnya, pengguna dengan peran "admin" memiliki akses penuh ke semua fitur aplikasi, sedangkan pengguna dengan peran "user" hanya memiliki akses terbatas pada fitur tertentu. Untuk mencapai hal ini, kita perlu mengembangkan fitur otorisasi multi-role.

Sebelum kita mulai mengimplementasikan fitur otorisasi multi-role, penting untuk memahami konsep dasar otorisasi. Otorisasi adalah proses menentukan apakah pengguna yang sudah terautentikasi memiliki izin untuk melakukan tindakan tertentu pada suatu sumber daya. Dalam konteks Laravel, kita dapat menggunakan middleware, gates, dan policies untuk mengontrol akses pengguna.

Percobaan ketiga

Kita akan melanjutkan dari progres terakhir dari bab pembelajaran sebelumnya langkah-langkahnya adalah sebagai berikut :

1. Menambahkan field role pada tabel users

   Pada tabel users yang telah kita buat memalui migration, selanjutnya kita perlu menambahkan field role dengan menambahkannya menggunakan migration (kode pembuatan migration pada bab sebelumnya) dan diisi dengan kode seperti dibawah ini

2024_09_19_212106_update_users_table_role_field.php U

database > migrations > 2024_09_19_212106_update_users_table_role_field.php > class > up

```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('users', function (Blueprint $table) {
            $table->enum('role', ['root', 'admin', 'user'])->default('user');
        });
    }

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

Jika selesai menambahkan kode diatas, silahkan eksekusi migration file diatas sehingga menjadi field baru 'role' ditabel users.

Penjelasan sederhana sintaks diatas digunakan dalam migrasi database Laravel untuk mendefinisikan sebuah kolom baru pada tabel. Mari kita pecah sintaks ini menjadi bagian-bagian:

- $table->enum: Bagian ini menunjukkan bahwa kita sedang membuat sebuah kolom dengan tipe data enum. Tipe data enum artinya kolom ini hanya dapat menyimpan nilai-nilai yang sudah ditentukan sebelumnya.
- 'role': Ini adalah nama kolom yang kita buat. Jadi, dalam tabel kita akan ada sebuah kolom bernama role.

- ['root', 'admin', 'user']: Di sini kita mendefinisikan nilai-nilai yang diperbolehkan untuk kolom role. Artinya, kolom role hanya boleh berisi salah satu dari tiga nilai ini: root, admin, atau user.
- ->default('user'): Bagian ini mengatur nilai default untuk kolom role. Jika kita tidak memasukkan nilai apapun saat memasukkan data baru ke dalam tabel, maka kolom role secara otomatis akan diisi dengan nilai 'user'.

2. Membuat middleware baru.
   Untuk dapat menambahkan role user kita perlu membuat middleware yang akan memvalidasi user sesuai dengan role-nya. Silahkan membuat middleware dengan nama RoleCheck dengan sintaks seperti berikut

```
php artisan make:middleware RoleCheck
```

Sintaks diatas akan menambah file baru pada folder app/Http/Middleware/RoleCheck.php silahkan anda modifikasi kode pada file tersebut seperti berikut

```php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;
use Illuminate\Support\Facades\Auth;

class RoleCheck
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next, ...$roles): Response
    {
        foreach ($roles as $role) {
            if (Auth::check() && Auth::user()->role == $role) {
                return $next($request);
            }
        }
        Auth::logout();
        return redirect()->route('login')->with('status','You are not authorized to access this page.');
    }
}

```

Setelah kita membuat file middleware baru, selanjutnya kita perlu mendaftarkan middleware yang telah dibuat didalam file

/vendor/laravel/framework/src/Illuminate/Foundation/Http/Kernel.php pada array protected $routeMiddleware menjadi seperti berikut

```
74      protected $routeMiddleware = [
75          'RoleCheck' => \App\Http\Middleware\RoleCheck::class,
76      ];
77
```

Selanjutnya untuk membatasi routes agar hanya diakses oleh role tertentu kita perlu membatasinya dengan middleware beserta role yang dapat mengaksesnya, seperti gambar berikut

```
20  Route::get('/dashboard', function () {
21      return view('dashboard');
22  })->middleware(['auth', 'verified','RoleCheck:admin'])->name('dashboard');
```

Dari gambar diatas, routes /dasboard dibatasi harus login untuk mengakses dan hanya yang memiliki role admin, diatur pada bagian RoleCheck:admin

Membuat controller

Controller pada Laravel 11 adalah kelas PHP yang bertugas menangani logika aplikasi dan bertindak sebagai perantara antara model (database) dan tampilan (view), tetapi disisi lain kita dapat menempatkan logika sistem yang kita buat di dalam service, sehingga controller hanya benar-benar menghubungkan komponen-komponen yang dibutuhkan saja. Controller juga mengelola permintaan HTTP, mengakses data melalui model, dan mengembalikan respons yang sesuai, seperti tampilan atau data JSON. Berikut pembuatan controller di Laravel 11 beserta contohnya

```
php artisan make:controller (NamaController) - -resource
```

Contoh

```
php artisan make:controller ProductController --resource
```

Perintah diatas akan menghasilkan file controller di direktori app/Http/Controllers dengan nama ProductController.php, didalam class controller Product yang kita buat, akan terdapat Method yang dibuat otomatis dari option --resource seperti berikut

- Method index() Biasanya digunakan untuk menampilkan daftar data. Metode ini mengembalikan tampilan yang berisi data, yang biasanya diambil dari model.
- Method create() Menampilkan form untuk membuat entitas baru (misalnya, menambah mahasiswa).
- Method store(Request $request) Menangani logika penyimpanan data baru ke database. Parameter $request berisi data yang dikirimkan dari form.
- Method show($id) Menampilkan detail satu entitas berdasarkan ID-nya.
- Method edit($id) Menampilkan form untuk mengedit entitas yang ada.
- Method update(Request $request, $id) Menangani logika untuk memperbarui data entitas yang sudah ada berdasarkan ID-nya.
- Method destroy($id) Menghapus entitas yang ada berdasarkan ID.

Menghubungkan Routes dengan Controller

Untuk mengakses *Method* dalam controller melalui URL, Kita perlu mendefinisikan dan menghubungkan controller dengan routes yang berada di dalam file routes/web.php. Sebelum menghubungkan method controller kedalam routes, maka perlu diimport controller yang dimaksud dengan perintah use, Contoh untuk ProductController

```
6   use App\Http\Controllers\ProductController;
7
8   Route::get('/product', [ProductController::class, 'index']);
9   Route::get('/product/create', [ProductController::class, 'create']);
10  Route::post('/product', [ProductController::class, 'store']);
11  Route::get('/product/{id}', [ProductController::class, 'show']);
12  Route::get('/product/{id}/edit', [ProductController::class, 'edit']);
13  Route::put('/product/{id}', [ProductController::class, 'update']);
14  Route::delete('/product/{id}', [ProductController::class, 'destroy']);
15
```

Mengenal blade sebagai template engine di laravel

Blade adalah mesin template (template engine) dari Laravel, yang digunakan untuk mempermudah pembuatan antarmuka pengguna (UI) pada aplikasi web. Dengan Blade, Anda bisa menulis HTML bersama dengan sintaks khusus yang menyediakan fitur seperti looping, conditional statements, inheritance dan component creation secara lebih mudah dan efisien.

Untuk penjelasan dengan format vidio anda dapat mengunjungi materi yang sudah saya unggah di https://youtu.be/pCGY5mkec7I?si=ObiEJN4cRtrgOk3N dan https://youtu.be/VfhtUUlPxjI?si=INDFgnAs524hmklB

Fitur Utama Blade

- Sintaks yang Sederhana
  Blade menggunakan sintaks khusus yang sederhana dan mudah dipahami. Misalnya, menampilkan data menggunakan `{{ }}`: <h1>Hello, {{ $name }}!</h1> Di sini, variabel `$name` akan dieksekusi dan ditampilkan sebagai teks pada halaman HTML. untuk lebih detailnya anda dapat mengunjungi dokumentasi resmi tentang blade di https://laravel.com/docs/11.x/blade#blade-directives

- Template Inheritance
  Blade mendukung sistem pewarisan (inheritance) untuk membuat layout yang bisa digunakan kembali. Anda dapat mendefinisikan sebuah template induk (master layout) dan mengisi konten di dalamnya menggunakan `@yield` dan `@section`. Contoh

  Template Induk (`layouts/master.blade.php`):

```
<html>
    <head>
        <title>My Website - @yield('title')</title>
    </head>
    <body>
        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

  Template Turunan (`home.blade.php`):

```
@extends('layouts.master')

@section('title', 'Home Page')
```

```
@section('content')
    <h1>Welcome to the Home Page</h1>
@endsection
```

- Include File Blade
  Anda bisa menyertakan file Blade lainnya di dalam satu file menggunakan `@include`.

Membuat view blade dengan template engine

Didalam laravel jika kita akan membuat view atau *user interface* kita dapat menggunakan Blade sebagai template engine untuk membuat view yang dinamis dan modular. Dengan Blade, Anda bisa membuat layout dasar (template) dan menggabungkannya dengan konten yang spesifik untuk setiap halaman. Berikut adalah langkah-langkah membuat view Blade dengan template engine.

Percobaan keempat

Silahkan lakukan percobaan dalam vidio dibawah ini :

1. https://youtu.be/pCGY5mkec7I?si=P962UjCaIEyyID-m&t=153
2. https://youtu.be/VfhtUUlPxjI?si=5PeGgws07uDVc8bs

Menghubungkan controller dengan view

Setelah kita menghubungkan routes ke controller dan membuat view, selanjutnya kita perlu menghubungkan dari controller (ProductController.php pada direktory app\Http\Controllers\ProductController.php) ke view dengan menambahkan sintax di method index seperti gambar berikut

```
12      public function index()
13      {
14          return view('layouts-percobaan.app');
15      }
```

Pada gambar diatas, pada baris 14 kita memanggil view('layouts-percobaan.app') yang artinya secara default fungsi view( ) me-load file yang berada di folder resources\views\ jika dalam blade [titik] diartikan sebagai path folder, seperti contoh diatas 'layouts-percobaan[titik]app' yang artinya memanggil file app.blade.php yang berada

dalam folder layouts-percobaan sama seperti 'layouts-percobaan/app.blade.php' atau lebih jelasnya seperti gambar berikut

Pada ProductController.php	File view yang diload