KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

Mata Kuliah	**:**	Pemrograman Web Lanjut (PWL)
Program Studi	**:**	D4 – Teknik Informatika / D4 – Sistem Informasi Bisnis
Semester	**:**	4 (empat) / 6 (enam)
Pertemuan ke-	**:**	5 (satu)

JOBSHEET 05

Blade View, Web Templating(AdminLTE), Datatables

Nama : Reza Angelina Febriyanti
Kelas / No Absen : SIB 2A / 27
NIM : 2341760015

View merupakan tempat bagi kita untuk meletakkan kode-kode HTML. Sehingga dalam laravel tidak menggunakan lagi file berformat .html. Kita masih menggunakan tagging HTML, tapi kita perlu menghandle tampilan dengan lebih canggih. Menampilkan data yang diberikan oleh controller. Untuk itu kita akan menggunakan templating engine, yaitu Blade.

Blade merupakan templating engine bawaan Laravel. Berguna untuk mempermudah dalam menulis kode tampilan. Dan juga memberikan fitur tambahan untuk memanipulasi data di view yang dilempar dari controller. Blade juga memungkinkan penggunaan plain PHP pada kode View.

Sesuai dengan `Studi Kasus PWL.pdf`.
Jadi project Laravel 10 kita masih sama dengan menggunakan repositori **PWL_POS.**

Project **PWL_POS** akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajarai

A. View

Karena Laravel menggunakan templating engine bawaan Blade, maka setiap file View diakhiri dengan .blade.php. Misal: index.blade.php, home.blade.php, product.blade.php. Contoh sederhana dari view dengan nama *file* hello.blade.php adalah sebagai berikut.

```
<!-- View pada resources/views/hello.blade.php -->
<html>
	<body>
	<h1>Hello, {{ $name }}</h1>
	</body>
</html>
```

View tersebut dapat dijalankan melalui Routing, dimana route akan memanggil View sesuai dengan nama file tanpa 'blade.php'

```
Route::get('/hello', function () {
	return view('hello', ['name' => 'Andi']);
});
```

Setiap kita membuat view, kita mungkin akan menampilkannya melalui router atau controller menggunakan global view helper. Selain menggunakan global view helper, kita juga dapat menggunakan view facade untuk menampilkan view.

```
Use Illuminate\Support\Facades\View;
return View::make('hello', ['name' => 'Andi']);
```

Seperti yang kita lihat, argumen pertama yang diteruskan ke view helper sesuai dengan nama file view di direktori resources / views. Argumen kedua adalah larik data yang harus tersedia untuk tampilan. Dalam hal ini, laravel meneruskan variabel nama yang ditampilkan dalam tampilan menggunakan sintaks Blade.

1. View di dalam direktori

Jika di dalam direktori `resources/views` terdapat direktori lagi untuk menyimpan *file* view, sebagai contoh hello.blade.php ada di dalam direktori blog, maka kita bisa menggunakan "dot" notation untuk mereferensikan direktori, sehingga syntax dalam route akan seperti berikut

```
Route::get('/hello', function () {
	return view('blog.hello', ['name' => 'Andi']);
});
```

2. Menampilkan view dari controller

View dapat dipanggil melalui Controller. Sehingga Routing akan memanggil Controller terlebih dahulu, dan Controller akan me-return view yang dimaksud

```
Route::get('/hello,[WelcomeController::class, 'hello']);
```

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class WelcomeController extends Controller
{
	public function hello() {
		return view('blog.hello', ['name' => 'Andi']);
	}
}
```

3. Meneruskan data ke view

Seperti yang anda lihat di contoh sebelumnya, anda dapat meneruskan data array ke view agar data tersebut tersedia untuk view

```
return view('hello', ['name' => 'Andi']);
```

Saat meneruskan informasi dengan cara ini, data harus berupa array dengan pasangan kunci / nilai. Setelah memberikan data ke view, anda kemudian dapat mengakses setiap nilai dalam view menggunakan kunci data seperti: <?php echo $name; ?> atau {{$name}}.

Sebagai alternatif untuk meneruskan array data lengkap ke fungsi view helper, anda dapat menggunakan metode with untuk menambahkan bagian data individual ke view. Metode with mengembalikan instance view objek sehingga anda dapat melanjutkan rangkaian metode sebelum mengembalikan tampilan

```
return view('hello')
              ->with('name','Andi')
              ->with('occupation','Astronaut');
```

4. Berbagi data dengan semua view

Terkadang, anda mungkin perlu berbagi data dengan semua tampilan yang direnderer oleh aplikasi anda. anda dapat melakukannya dengan menggunakan metode berbagi tampilan fasad. Biasanya, anda harus melakukan panggilan ke share methodi dalam metode boot penyedia layanan. anda bebas menambahkannya ke kelas `App\Providers\AppServiceProvider` atau membuat penyedia layanan terpisah untuk menampungnya

```
      <?php
namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
```

```
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        View::share('key', 'value');
    }
}
```

B. Blade Layout, Section, dan Component

Dalam membuat suatu tampilan, seringkali ada beberapa bagian yang sama dan selalu ditampilkan di setiap halaman view. Bagian-bagian ini dapat dibuat template sehingga tidak perlu dibuat berulang kali di setiap halaman view.

1. Layout dan Section

Pada laravel, layout digunakan untuk membuat master view yang akan selalu ditampilkan oleh view-view child yang menggunakannya. Dalam sebuah layout kita bisa memberikan tempat-tempat yang bisa digunakan oleh child view. Tempat-tempat tersebut adalah section. Misalnya, dalam layout utama, kita definisikan section sidebar, main_content, dan footer. Selanjutnya, setiap child view yang menggunakan layout utama dapat menempatkan kode view di masing-masing section yang tersedia di layout utama. Pada layout, setiap kode html akan digunakan oleh child view. Sehingga child view tidak perlu mendefinisikan tag html, head, title, dll pada tiap-tiap view. Terdapat beberapa istilah yang digunakan untuk menerapkan layout.

@yield

@yield("nama_section") digunakan untuk mendefinisikan bagian dari layout yang akan digunakan dan diisi oleh child view.

@section

@section digunakan selain untuk mendefinisikan sebuah section, juga bisa untuk mengisi section yang diharapkan oleh parent view / layout melalui @yield. Diakhiri dengan @endsction.

@parent

Dalam child view kita bisa menampilkan juga konten yang ada pada parent dalam section tertentu, hal tersebut dilakukan dengan @parent.

@extends

Extends digunakan pada setiap child view yang ingin menggunakan sebuah view sebagai parent / layout.

Berikut adalah contoh dari sebuah layout dari master/parent view.

```
<!-- Disimpan di resources/views/layouts/app.blade.php -->
 <html>
       <head>
       <title> Halaman @yield('title')</title>
       </head>
       <body>
       @section('sidebar')
              Ini adalah master sidebar.
       @show
       <div class="container">
              @yield('content')
       </div>
       </body>
</html>
```

Kode yang merupakan layout global, mendefinisikan title, sidebar, dan content yang dapat diisi oleh child view yang menggunakannya.

Berikut adalah contoh child view yang menggunakan layout app.blade.php.

```
<!—Disimpan di resources/views/child.blade.php -->
 @extends('layouts.app')
 @section('title', 'Profil')
 @section('sidebar')
       @parent
       <p>Sidebar halaman Profil.</p>
 @endsection
 @section('content')
            <p>Ini adalah bagian konten. NIM - Nama</p>
 @endsection
```

Keterangan: @extends(“layouts.app”) digunakan untuk menjadikan file view app.blade.php. sebagai master view. Kemudian kita isi section title dengan “Profil” yang akan direnderr sebagai <title> Halaman Profil </title>. Kemudian kita juga mengisi section sidebar menggunakan directive @parent. Sehingga ditampilkan konten sidebar parent/master serta menampilkan kalimat “Sidebar halaman Profil”. Setelah itu kita juga isi section “content” dengan tulisan “Ini adalah bagian konten. NIM - Nama”.

2. Component

Component berfungsi untuk membuat view yang dapat kita gunakan berulang kali. Berbeda dengan layout yang bertindak sebagai master, component dapat dianggap sebagai child view yang bisa kita pakai di view lain yang membutuhkannya. Misalnya dalam pengembangan sebuah aplikasi kita akan membutuhkan view untuk alert yang memberikan notifikasi kepada pengguna aplikasi terkait informasi, peringatan ataupun pesan error. Alert ini akan digunakan berulang kali di aplikasi. Oleh karena itu kita bisa membuatnya sebagai component yang bisa digunakan di view lainnya. Berikut adalah contoh component untuk membuat alert.

```
<!-- Disimpan di resources/views/components/alert.blade.php -->
 <div class="alert alert-danger">
{{ $slot }}
</div>
```

Di bawah ini adalah contoh view yang menerapkan component alert.

```
@extends('layouts.app')
// kode…
@component("alert")
<b>Tulisan ini akan mengisi variabel $slot</b>
@endcomponent
```

C. Bootstrap pada Laravel

Laravel menyediakan titik awal dasar menggunakan Bootstrap. Secara default, Laravel menggunakan NPM untuk menginstal paket frontend ini. Untuk menggunakannya, dapat dilakukan langkah-langkah berikut:

Dalam jobsheet ini instalasi bootstrap dilakukan dibagian praktikum. Berikut ini hanya pengetahuan cara instalasi umumnya.

1. Melakukan instalasi Node.js. Installer dapat diperoleh dari https://nodejs.org/en/. Sesuaikan dengan sistem operasi yang digunakan.
2. Untuk memastikan instalasi Node.js dan NPM berjalan lancar, Anda dapat memeriksanya dengan menjalankan dua perintah berikut melalui Command Prompt:

   node -v

   npm -v

   Setelah Anda mengetikkan kedua perintah tersebut, Command Prompt akan menunjukkan versi Node.js dan NPM yang ter-install di komputer.

3. Kemudian melalui command prompt, ubah direktori ke dalam direktori project laravel yang telah dibuat sebelumnya.
4. Scaffolding Bootstrap yang disediakan oleh Laravel terletak di dalam paket Composer, yang dapat diinstal menggunakan Composer:laravel/ui.

   ```
   composer require laravel/ui
   ```

5. Setelah paket diinstal, kita dapat menginstal scaffolding frontend menggunakan perintah Artisan:laravel/ui

```
php artisan ui bootstrap

php artisan ui bootstrap --auth
```

6. Sebelum mengompilasi CSS, install dependensi frontend proyek Anda menggunakan Node package manager (NPM) :

```
npm install
```

7. Setelah paket diinstal, Anda dapat menggunakan perintah berikut untuk mengompilasi aset Anda .

```
npm run dev
```

Menggunakan Template Bootstrap di Laravel

Sebenarnya pada project laravel, sudah ada file css bootstrap secara default pada pertama kali kita menginstall laravel. Letaknya ada pada file app.css dalam folder css. Anda dapat langsung menggunakannya dengan menghubungkan file app.css tersebut dengan file view anda.

Untuk mengubungkan file css ke laravel anda dapat menggunakan syntax berikut:

```
<link rel="stylesheet" type="text/css" href="/css/style.css">
```

Atau

```
<link rel="stylesheet" type="text/css" href="{{ asset('/css/app.css') }}">
```

Dan file JSnya:

```
<script type="text/javascript" src="/js/app.js"></script>
```

Atau

```
<script type="text/javascript" src="{{ asset('/js/app.js') }}"></script>
```

Secara default linknya dimulai dari folder public. Jadi anda bisa meletakan file css dan js didalam folder public.

Praktikum 1 – Integrasi Laravel dengan AdminLte3

Cara paling cepat dan clean dapat dilakukan menggunakan repository dalam tautan berikut https://github.com/jeroennoten/Laravel-AdminLTE/wiki/Installation.

1. Dalam root folder project lakukan command berikut, untuk mendefinisikan requirement project.

```
composer require jeroennoten/laravel-adminlte
```

2. Melakukan instalasi requirement project di atas dengan command berikut:

```
php artisan adminlte:install
```

Perintah di atas akan meng-install:

- **AdminLTE distribution files** dan dependensinya (`Bootstrap`, `jQuery`, etc.) dalam folder `public/vendor` .
- Konfigurasi package di file `config/adminlte.php`
- Paket translasi di folder `lang/vendor/adminlte/`
- Dalam composer.json akan otomatis ditambahkan require untuk laravel-adminlte

composer.json > ...

```
{
    "name": "laravel/laravel",
    "type": "project",
    "description": "The skeleton application for the Laravel framework.",
    "keywords": ["laravel", "framework"],
    "license": "MIT",
    "require": {
        "php": "^8.1",
        "guzzlehttp/guzzle": "^7.2",
        "jeroennoten/laravel-adminlte": "^3.9",
        "laravel/framework": "^10.10",
        "laravel/sanctum": "^3.3",
        "laravel/tinker": "^2.8"
    },
    "require-dev": {
        "fakerphp/faker": "^1.9.1",
        "laravel/pint": "^1.0",
        "laravel/sail": "^1.18",
        "mockery/mockery": "^1.4.4",
        "nunomaduro/collision": "^7.0",
        "phpunit/phpunit": "^10.1",
        "spatie/laravel-ignition": "^2.0"
    },
    "autoload": {
        "psr-4": {
```

3. Buat file resources/views/layout/app.blade.php. Isi dengan kode berikut.

```
@extends('adminlte::page')

{{-- Extend and customize the browser title --}}

@section('title')
    {{ config('adminlte.title') }}
    @hasSection('subtitle') | @yield('subtitle') @endif
@stop

{{-- Extend and customize the page content header --}}

@section('content_header')
    @hasSection('content_header_title')
        <h1 class="text-muted">
            @yield('content_header_title')

            @hasSection('content_header_subtitle')
                <small class="text-dark">
                    <i class="fas fa-xs fa-angle-right text-muted"></i>
                    @yield('content_header_subtitle')
                </small>
            @endif
        </h1>
    @endif
@stop

{{-- Rename section content to content_body --}}

@section('content')
    @yield('content_body')
@stop

{{-- Create a common footer --}}

@section('footer')
    <div class="float-right">
        Version: {{ config('app.version', '1.0.0') }}
    </div>

    <strong>
```

```
        <a href="{{ config('app.company_url', '#') }}">
            {{ config('app.company_name', 'My company') }}
        </a>
    </strong>
@stop

{{-- Add common Javascript/Jquery code --}}

@push('js')
<script>

    $(document).ready(function() {
        // Add your common script logic here...
    });

</script>
@endpush

{{-- Add common CSS customizations --}}

@push('css')
<style type="text/css">

    {{-- You can add AdminLTE customizations here --}}
    /*
    .card-header {
        border-bottom: none;
    }
    .card-title {
        font-weight: 600;
    }
    */
</style>
@endpush
```

4. Edit `resources/views/welcome.blade.php`, kemudian replace seluruh kodenya dengan kode berikut

resources > views > welcome.blade.php > ...

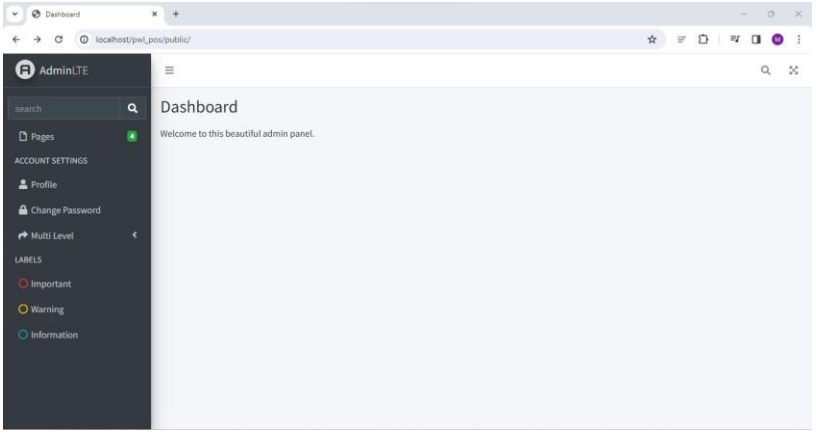
```
1  @extends('layouts.app')
2
3  {{-- Customize layout sections --}}
4
5  @section('subtitle', 'Welcome')
6  @section('content_header_title', 'Home')
7  @section('content_header_subtitle', 'Welcome')
8
9  {{-- Content body: main page content --}}
10
11 @section('content_body')
12     <p>Welcome to this beautiful admin panel.</p>
13 @stop
14
15 {{-- Push extra CSS --}}
16
17 @push('css')
18     {{-- Add here extra stylesheets --}}
19     {{-- <link rel="stylesheet" href="/css/admin_custom.css"> --}}
20 @endpush
21
22 {{-- Push extra scripts --}}
23
24 @push('js')
25     <script> console.log("Hi, I'm using the Laravel-AdminLTE package!"); </script>
26 @endpush
```

Untuk menggunakan blade template ini cukup dengan extend **AdminLTE layout** dengan cara `@extends('adminlte::page').`
Template yields di beberapa bagian terklasifikasi dalam 2 yield:

- **main**: Biasa digunakan untuk extending the layout.
- **misc**: untuk kasus yang tidak biasa, atau hanya situasi tertentu.

Dokumentasi lebih detail terdapat di link berikut: https://github.com/jeroennoten/Laravel-AdminLTE/wiki/Usage

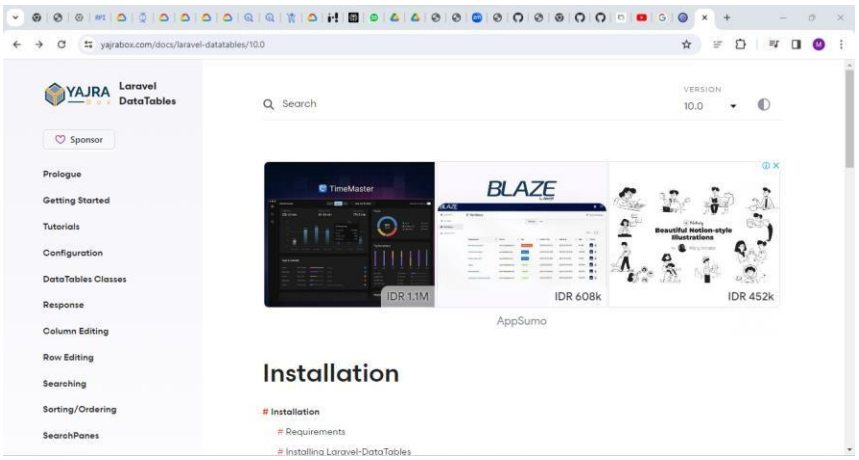
5. Kembali ke browser, menuju ke halaman awal.



Praktikum 2 – Integrasi dengan DataTables

Datatables biasa digunakan untuk menampilkan list data dengan banyak fitur seperti searching, pagination, sorting dan lain-lainnya. Dalam laravel biasa digunankan yajra datatables karena didesain agar sesuai dengan laravel.

https://yajrabox.com/docs/laravel-datatables/10.0



Berikut langkah-langkah praktikum

1. Install Laravel DataTables

```
composer require laravel/ui --dev
```

```
composer require yajra/laravel-datatables:^10.0
```

2. Pastikan nodejs sudah terinstall, dengan perintah `npm -v`.

Jika belum install nodejs package manager dari https://nodejs.org/dist/v20.11.1/node-v20.11.1-x64.msi

Npm digunakan untuk manajemen package javascript.

3. Install Laravel DataTables Vite dan sass

```
npm i laravel-datatables-vite --save-dev
```

```
npm install -D sass
```

4. Edit file resources/js/app.js

resources > js > JS app.js

```
import './bootstrap';
import "../sass/app.scss";
import 'laravel-datatables-vite';
```

5. Buatlah file resources/saas/app.scss

resources > sass > app.scss

```
// Fonts
@import url('https://fonts.bunny.net/css?family=Nunito');


// Bootstrap
@import 'bootstrap/scss/bootstrap';

// DataTables
@import 'bootstrap-icons/font/bootstrap-icons.css';
@import "datatables.net-bs5/css/dataTables.bootstrap5.min.css";
@import "datatables.net-buttons-bs5/css/buttons.bootstrap5.min.css";
@import 'datatables.net-select-bs5/css/select.bootstrap5.css';
```

6. Jalankan dengan `npm run dev`

```
C:\laragon\www\pwl_pos>npm run dev

> dev
> vite


  VITE v5.1.5  ready in 250 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help

  LARAVEL v10.46.0  plugin v1.0.2

  ➜  APP_URL: http://localhost
```

7. Selanjutnya kita akan buat datatables untuk kategori

`php artisan datatables:make Kategori`

```
C:\laragon\www\pwl_pos>php artisan datatables:make Kategori

  INFO  DataTable [C:\laragon\www\pwl_pos\app\DataTables\KategoriDataTable.php] created successfully.
```

8. Kita edit KategoriDatable untuk mengatur kolom apasaja yang ingin ditampilkan Sesuaikan dengan kodeprogram berikut

```
<?php
```

```
namespace App\DataTables;

use App\Models\KategoriModel;
use Illuminate\Database\Eloquent\Builder as QueryBuilder;
use Yajra\DataTables\EloquentDataTable;
use Yajra\DataTables\Html\Builder as HtmlBuilder;
use Yajra\DataTables\Html\Button;
use Yajra\DataTables\Html\Column;
use Yajra\DataTables\Html\Editor\Editor;
use Yajra\DataTables\Html\Editor\Fields;
use Yajra\DataTables\Services\DataTable;

class KategoriDataTable extends DataTable
{
    /**
     * Build the DataTable class.
     *
     * @param QueryBuilder $query Results from query() method.
     */
    public function dataTable(QueryBuilder $query): EloquentDataTable
    {
        return (new EloquentDataTable($query))
/*              ->addColumn('action', 'kategori.action') */
            ->setRowId('id');
    }

    /**
     * Get the query source of dataTable.
     */
    public function query(KategoriModel $model): QueryBuilder
    {
        return $model->newQuery();
    }

    /**
     * Optional method if you want to use the html builder.
     */
    public function html(): HtmlBuilder
    {
        return $this->builder()
                    ->setTableId('kategori-table')
                    ->columns($this->getColumns())
                    ->minifiedAjax()
                    //->dom('Bfrtip')
                    ->orderBy(1)
                    ->selectStyleSingle()
                    ->buttons([
```

```
                        Button::make('excel'),
                        Button::make('csv'),
                        Button::make('pdf'),
                        Button::make('print'),
                        Button::make('reset'),
                        Button::make('reload')
                    ]);
    }

    /**
     * Get the dataTable columns definition.
     */
    public function getColumns(): array
    {
        return [
    /*          Column::computed('action')
                    ->exportable(false)
                    ->printable(false)
                    ->width(60)
                    ->addClass('text-center'), */
            Column::make('kategori_id'),
            Column::make('kategori_kode'),
            Column::make('kategori_nama'),
            Column::make('created_at'),
            Column::make('updated_at'),
        ];
    }

    /**
     * Get the filename for export.
     */
    protected function filename(): string
    {
        return 'Kategori_' . date('YmdHis');
    }
}
```

9. Ubah kategori model, sesuaikan seperti berikut

   Sebelum itu buat file KategoriModel.php dan BarangModel.php melalui sintaks berikut

```
  INFO  Model [C:\laragon\www\PWL_2025\Minggu5\Jobsheet5\app\Models\KategoriModel.php] created successfully.

PS C:\laragon\www\PWL_2025\Minggu5\Jobsheet5> php artisan make:model BarangModel

  INFO  Model [C:\laragon\www\PWL_2025\Minggu5\Jobsheet5\app\Models\BarangModel.php] created successfully.
```

app > Models > KategoriModel.php > KategoriModel

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class KategoriModel extends Model
{
    protected $table = 'm_kategori';
    protected $primaryKey = 'kategori_id';

    protected $fillable = ['kategori_kode', 'kategori_nama'];

    public function barang(): HasMany
    {
        return $this->hasMany(BarangModel::class, 'barang_id', 'barang_id');
    }
}
```

10. Ubah Kategori Controller sesuaikan dengan skrip berikut:

app > Http > Controllers > KategoriController.php > KategoriController > index

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\DataTables\KategoriDataTable;

class KategoriController extends Controller
{
    public function index(KategoriDataTable $dataTable)
    {
        return $dataTable->render('kategori.index');
    }
}
```

11. Buat folder kategori di dalam resources/view, kemudian buat view blade index untuk kategori `resources/views/kategori/index.blade.php`

resources > views > kategori > index.blade.php > ...

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

{{-- Customize layout sections --}}

@section('subtitle', 'Kategori')
@section('content_header_title', 'Home')
@section('content_header_subtitle', 'Kagegori')

@section('content')
    <div class="container">
        <div class="card">
            <div class="card-header">Manage Kategori</div>
            <div class="card-body">
                {{ $dataTable->table() }}
            </div>
        </div>
    </div>
@endsection

@push('scripts')
    {{ $dataTable->scripts() }}
@endpush

```

12. Pastikan route untung kategori sudah tersedia

```
Route::get('/kategori', [KategoriController::class, 'index']);
```

13. Sesuaikan app layout seperti kode berikut

```
@extends('adminlte::page')

{{-- Extend and customize the browser title --}}

@section('title')
    {{ config('adminlte.title') }}
    @hasSection('subtitle') | @yield('subtitle') @endif
@stop

@vite('resources/js/app.js')

{{-- Extend and customize the page content header --}}

@section('content_header')
    @hasSection('content_header_title')
        <h1 class="text-muted">
            @yield('content_header_title')

            @hasSection('content_header_subtitle')
```

```
                <small class="text-dark">
                    <i class="fas fa-xs fa-angle-right text-muted"></i>
                    @yield('content_header_subtitle')
                </small>
            @endif
        </h1>
    @endif
@stop

{{-- Rename section content to content_body --}}

@section('content')
    @yield('content_body')
@stop


{{-- Create a common footer --}}

@section('footer')
    <div class="float-right">
        Version: {{ config('app.version', '1.0.0') }}
    </div>

    <strong>
        <a href="{{ config('app.company_url', '#') }}">
            {{ config('app.company_name', 'My company') }}
        </a>
    </strong>
@stop


{{-- Add common Javascript/Jquery code --}}



@push('js')
<script src="https://cdn.datatables.net/2.0.2/js/dataTables.js"></script>

@endpush

@stack('scripts')


{{-- Add common CSS customizations --}}

@push('css')
```

```
<link                                                                    rel="stylesheet"
href="https://cdn.datatables.net/2.0.2/css/dataTables.dataTables.css"  />

<style type="text/css">

    {{-- You can add AdminLTE customizations here --}}
    /*
    .card-header {
        border-bottom: none;
    }
    .card-title {
        font-weight: 600;
    }
    */

</style>

@endpush
```

14. Menset ViteJs / script type defaults

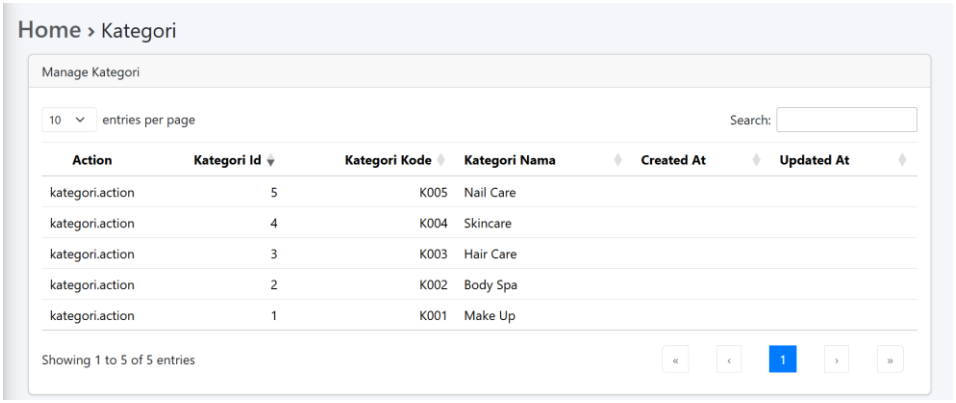
app > Providers > AppServiceProvider.php > AppServiceProvider > boot

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Yajra\DataTables\Html\Builder;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     */
    public function register(): void
    {
        //
    }

    /**
     * Bootstrap any application services.
     */
    public function boot(): void
    {
        Builder::useVite();
    }
}
```

15. Isikan beberapa data ke table kategori

	kategori_id	kategori_kode	kategori_nama	created_at	updated_at
Edit Copy Delete	1	K001	Make Up	NULL	NULL
Edit Copy Delete	2	K002	Body Spa	NULL	NULL
Edit Copy Delete	3	K003	Hair Care	NULL	NULL
Edit Copy Delete	4	K004	Skincare	NULL	NULL
Edit Copy Delete	5	K005	Nail Care	NULL	NULL

16. Datatables sudah dapat di load di `/kategori`



Praktikum 3 – Membuat form kemudian menyimpan data dalam database

Langkah-langkahnya adalah sebagai berikut:

1. Menyesuaikan routing, tambahkan dua routing berikut

```
Route::get('/kategori/create', [KategoriController::class, 'create']);
Route::post('/kategori', [KategoriController::class, 'store']);
```

2. Tambahkan dua function berikut dalam KategoriController

app > Http > Controllers > KategoriController.php > KategoriController > store

```
<?php

namespace App\Http\Controllers;
use App\Models\KategoriModel;
use Illuminate\Http\Request;
use App\DataTables\KategoriDataTable;

class KategoriController extends Controller
{
    public function index(KategoriDataTable $dataTable)
    {
        return $dataTable->render('kategori.index');
    }

    public function create()
    {
        return view('kategori.create');
    }

    public function store(Request $request)
    {
        KategoriModel::create([
            'kategori_kode' => $request->kodeKategori,
            'kategori_nama' => $request->namaKategori,

        ]);
        return redirect('/kategori');
    }
}
```

3. Dalam folder `views/kategori`, buatlah file dengan nama `create.blade.php`

resources > views > kategori > create.blade.php > ...

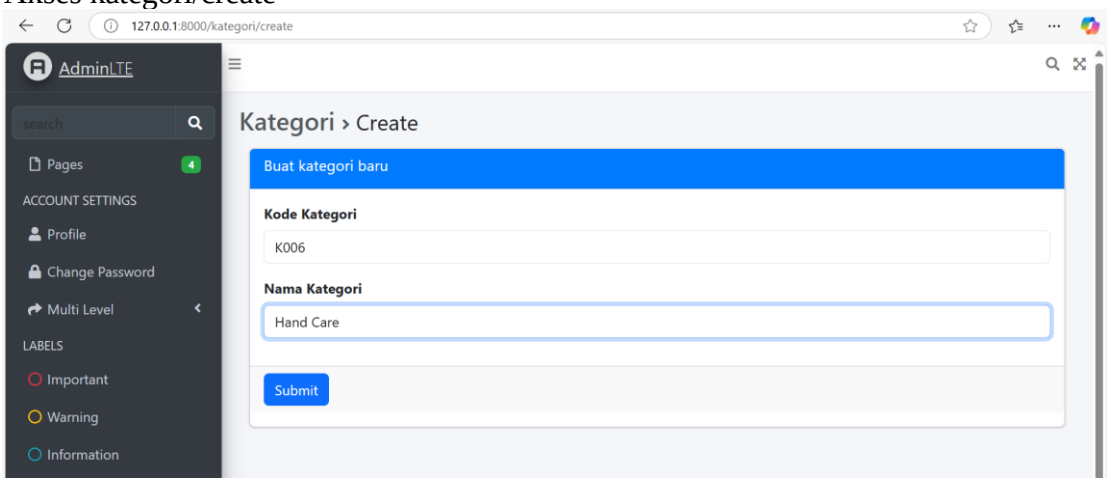
```
@extends('layouts.app')
{{-- Customize layout sections --}}
@section('subtitle', 'Kategori')
@section('content_header_title', 'Kategori')
@section('content_header_subtitle', 'Create')
{{-- Content body: main page content --}}
@section('content')
    <div class="container">
        <div class="card card-primary">
            <div class="card-header">
            <h3 class="card-title">Buat kategori baru</h3>
            </div>

            <form method="post" action="../kategori">
                <div class="card-body">
                    <div class="form-group">
                        <label for="kodeKategori">Kode Kategori</label>
                        <input type="text" class="form-control" id="kodeKategori" name="kodeKategori" placehold
                    </div>
                    <div class="form-group">
                        <label for="namaKategori">Nama Kategori</label>
                        <input type="text" class="form-control" id="namaKategori"  name="namaKategori" placehol
                    </div>
                </div>

                <div class="card-footer">
                <button type="submit" class="btn btn-primary">Submit</button>
                </div>
            </form>
            </div>
    </div>
@endsection
```

4. Kita lakukan pengecualian proteksi CsrfToken. Karena kita belum melakukan otentikasi . Kita edit dalam file berikut:

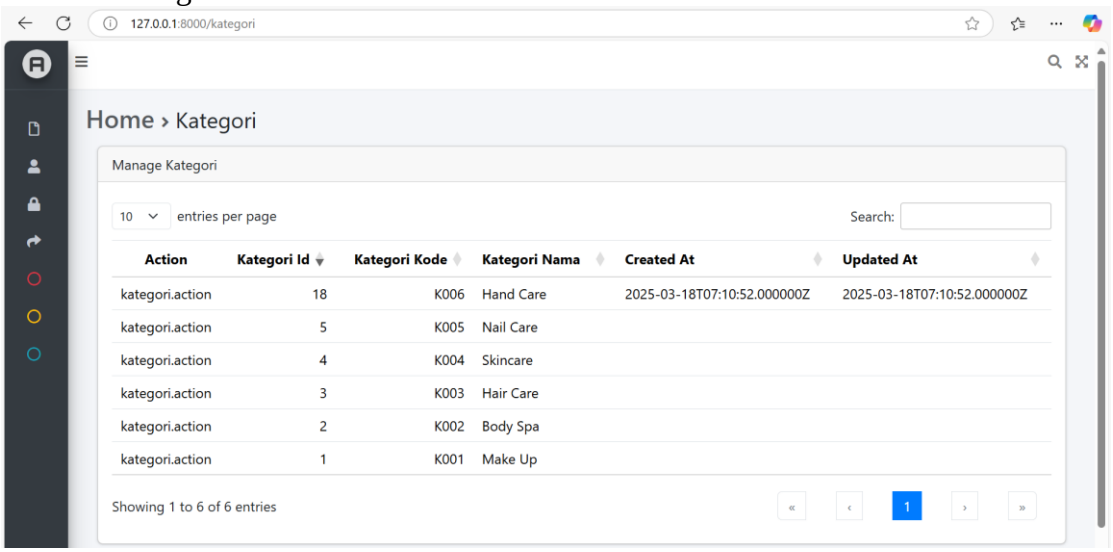
app > Http > Middleware > VerifyCsrfToken.php > VerifyCsrfToken > $except

```
<?php

namespace App\Http\Middleware;

use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;

class VerifyCsrfToken extends Middleware
{
    /**
     * The URIs that should be excluded from CSRF verification.
     *
     * @var array<int, string>
     */
    protected $except = [
        '/kategori'
    ];
}

```

5. Akses kategori/create



6. Halaman kategori



Tugas Praktikum

1. Tambahkan button Add di halam manage kategori, yang mengarah ke create kategori baru

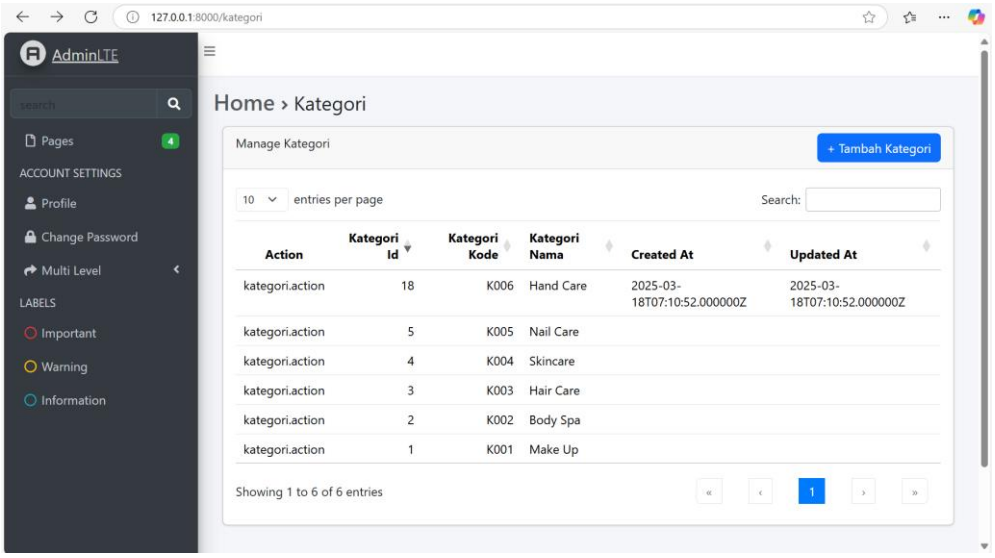
   Buat tombol add di index.blade.php

```
resources > views > kategori > index.blade.php > div.container > div.card > div.card-header
1  @extends(view: 'layouts.app')
2
3  {{-- Customize layout sections --}}
4
5  @section(section: 'subtitle', content: 'Kategori')
6  @section(section: 'content_header_title', content: 'Home')
7  @section(section: 'content_header_subtitle', content: 'Kategori')
8
9  @section(section: 'content')
10     <div class="container">
11         <div class="card">
12             <div class="card-header">Manage Kategori</div>
13             <a href="{{ route(name: 'kategori.create') }}" class="btn btn-primary float-right">Add Data</a>
14             <div class="card-body">
15                 {{ $dataTable->table() }}
16             </div>
17         </div>
18     </div>
19 @endsection
20
21 @push('scripts')
22     {{ $dataTable->scripts() }}
23 @endpush
```

   Buat Route di web.php seperti berikut

   Route::get('/kategori/create', [KategoriController::class, 'create'])->name('kategori.create');

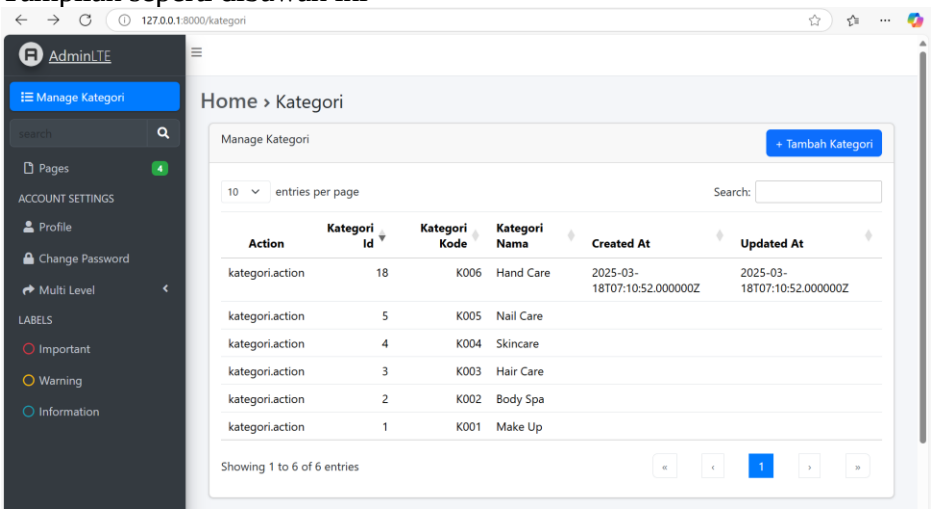
   Maka tampilan seperti dibawah ini



2. Tambahkan menu untuk halaman manage kategori, di daftar menu navbar Edit file adminlte.php di folder

```
'menu' => [
    // Navbar items:
    //Tugas2 Edit navbar items
    [
        'text' => 'Manage Kategori',
        'url' => 'kategori',
        'icon' => 'fas fa-list',
    ],
    [
        'type' => 'navbar-search',
        'topnav_right' => true,
    ],
```

Tampilan seperti dibawah ini



3. Tambahkan action edit di datatables dan buat halaman edit serta controllernya

Buat fungsi update dan edit pada file KategoriController.php

```
public function edit($id): Factory|View
{
    $data = KategoriModel::findOrFail(id: $id);
    return view(view: 'kategori.edit', data: ['kategori' => $data]);
}

0 references | 0 overrides
public function update(Request $request, $id): Redirector|RedirectResponse
{
    $request->validate(rules: [
        'kodeKategori' => 'required',
        'namaKategori' => 'required'
    ]);

    //cari data berdasarkan id
    $kategori = KategoriModel::findOrFail(id: $id);

    //update data
    $kategori->kategori_kode = $request->kodeKategori;
    $kategori->kategori_nama = $request->namaKategori;
    $kategori->save();

    return redirect(to: '/kategori');
}
```

Modifikasi fungsi dataTable dan getColums file KategoriDataTable

```php
public function dataTable(QueryBuilder $query): EloquentDataTable
{
    return (new EloquentDataTable($query))
    // ->addColumn('action', 'kategori.action')
    ->addColumn('action', function($id): string{
        $edit = route(name: 'kategori.edit', parameters: $id);
        return '<a href="'.$edit.'" class="btn btn-warning btn-sm">Edit</a>';

    })
    ->setRowId('id');
}
```

```php
public function getColumns(): array
{
    return [
        /* Column::computed('action')
              ->exportable(false)
              ->printable(false)
              ->width(60)
              ->addClass('text-center'), */
        Column::make('kategori_id'),
        Column::make('kategori_kode'),
        Column::make('kategori_nama'),
        Column::make('created_at'),
        Column::make('updated_at'),

        //Tugas3
        Column::computed('action')
            ->exportable(false)
            ->printable(false)
            ->width(60)
            ->addClass('text-center')
    ];
}
```

Buat file edit.blade.php di views/kategori

```blade
@extends('layouts.app')

@section('content')
    <div class="container-fluid">
        <div class="row">
            <div class="col-md-12">
                <div class="card card-primary">
                    <div class="card-header">
                        <h3 class="card-title">Edit Kategori</h3>
                    </div>

                    <form action="{{ route('kategori.update', $kategori->kategori_id) }}" method="POST">
                        @csrf
                        @method('PUT')

                        <div class="card-body">
                            <div class="form-group">
                                <label for="kodeKategori">Kode Kategori</label>
                                <input type="text" class="form-control" name="kodeKategori" id="kodeKategori"
                                    value="{{ $kategori->kategori_kode }}">
                            </div>

                            <div class="form-group">
                                <label for="namaKategori">Nama Kategori</label>
                                <input type="text" class="form-control" name="namaKategori" id="namaKategori"
                                    value="{{ $kategori->kategori_nama }}">
                            </div>
                        </div>

                        <div class="card-footer">
                            <button type="submit" class="btn btn-primary">Simpan</button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
@endsection
```

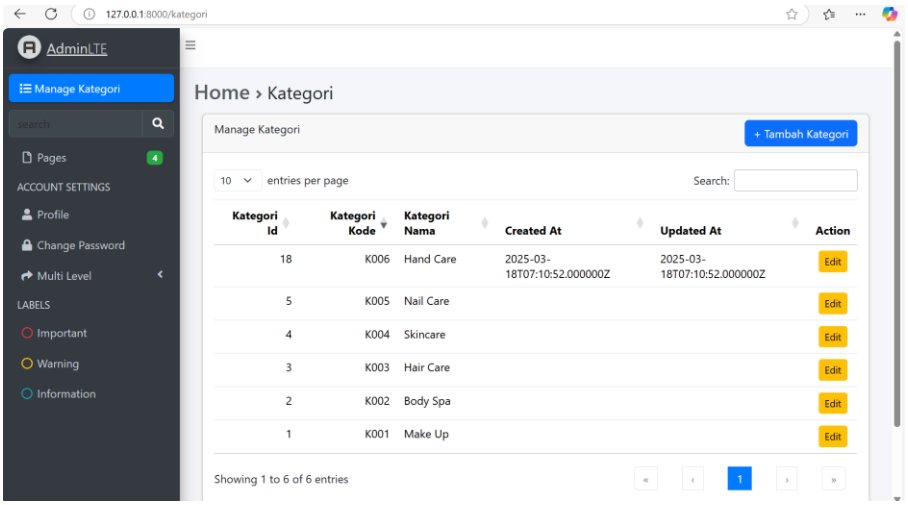
Tambahkan kode program berikut pada file index.blade.php

```
@push('scripts')
    {{ $dataTable->scripts() }}

    <script>
        $(document).ready(function () {
            $('#kategoriTable').on('draw.dt', function () {
                $('.btn-edit').on('click', function () {
                    var id = $(this).data('id');
                    window.location.href = "{{ url(path: 'kategori') }}/" + id + "/edit";
                });
            });
        });
    </script>

@endpush
```

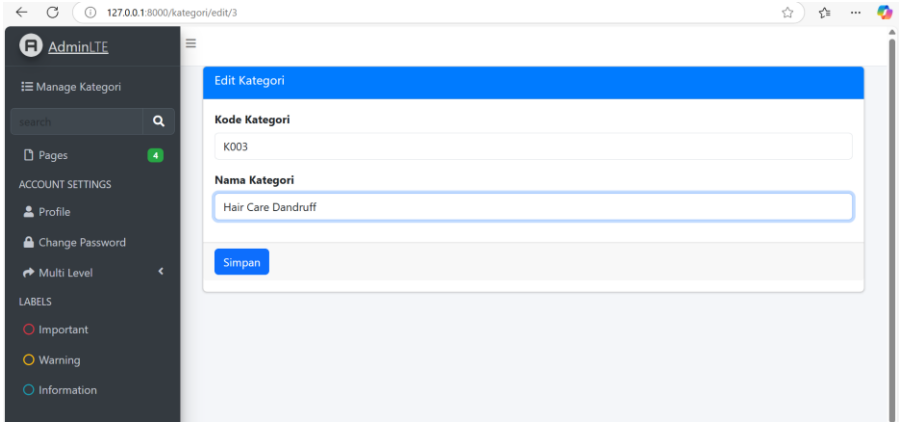
Tambahkan route berikut pada web.php

```
//Tugas3
Route::get('/kategori/edit/{id}', [KategoriController::class, 'edit'])->name('kategori.edit');
Route::put('/kategori/{id}', [KategoriController::class, 'update'])->name('kategori.update');
```

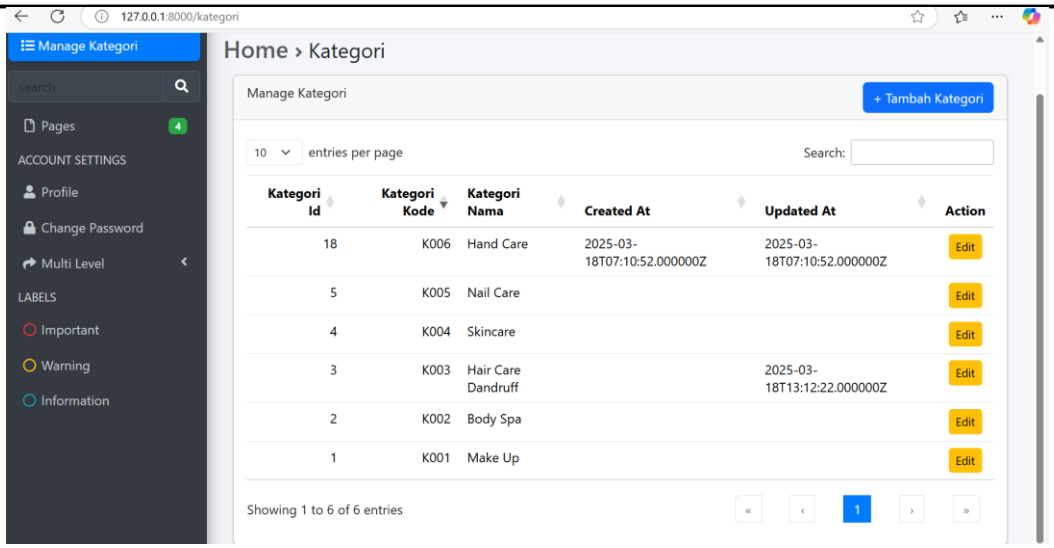
Maka hasilnya seperti dibawah ini



Saya ingin mengubah Hair Care menjadi



Hasilnya menjadi



4. Tambahkan action delete di datatables serta controllernya

   Menambahkan action delete pada KategoriDataTable.php

```php
public function dataTable(QueryBuilder $query): EloquentDataTable
{
    return (new EloquentDataTable($query))
    // ->addColumn('action', 'kategori.action')
    ->addColumn('action', function($id): string{
        $edit = route(name: 'kategori.edit', parameters: $id);
        $delete = route(name: 'kategori.delete', parameters: $id);
        return '
            <a href="'.$edit.'" class="btn btn-warning btn-sm">Edit</a>
            <a href="'.$delete.'" class="btn btn-danger btn-sm" onclick="return confirm(\'Apakah Anda yakin ingin menghapus kategori
            ini? Tindakan ini tidak dapat dibatalkan.\')">Delete</a>
        ';
    })
    ->setRowId('id');
}
```

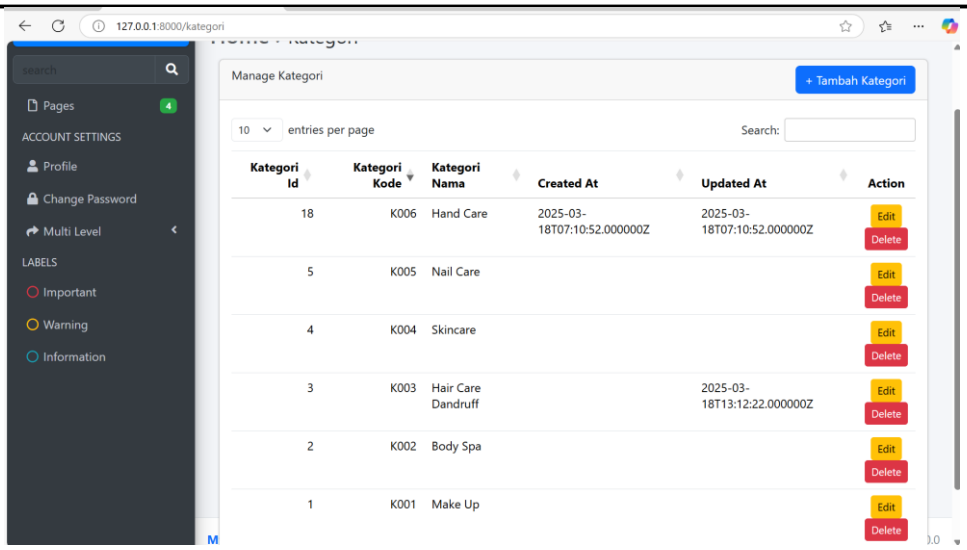
   Buat fungsi delete pada KategoriController.php

```php
1 reference | 0 overrides
public function delete($id): Redirector|RedirectResponse {
    KategoriModel::where('kategori_id', $id)->delete();
    return redirect(to: '/kategori');
}
```

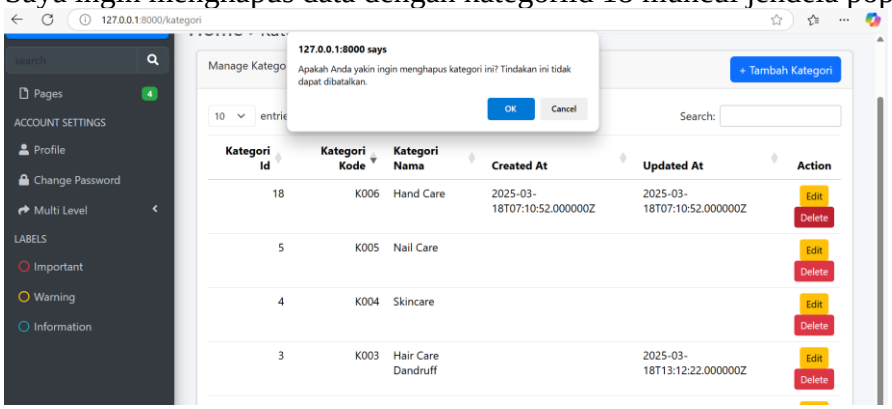
   Tambahkan Route delete seperti gambar dibawah ini di web.php

```php
//Tugas4
Route::get('/kategori/delete/{id}', [KategoriController::class, 'delete'])->name('kategori.delete');
```

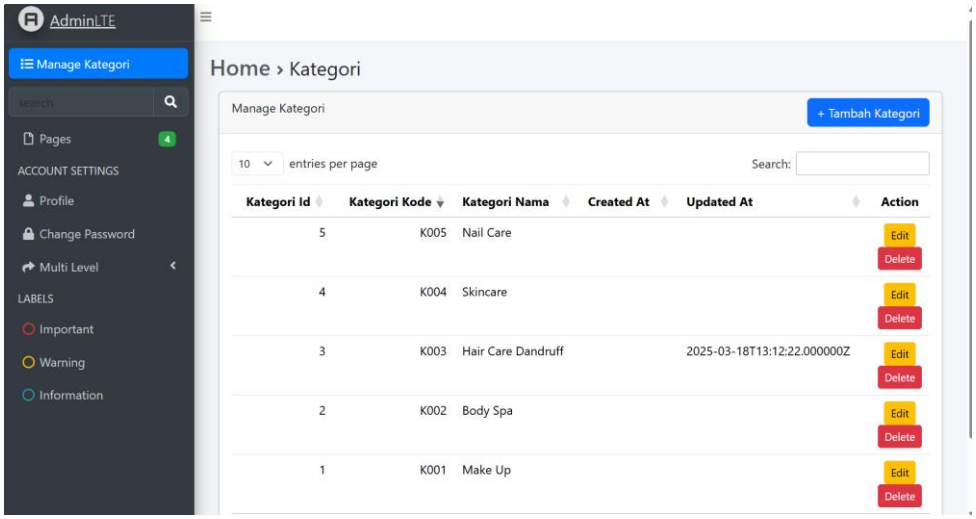
   Tampilan seperti dibawah ini



Saya ingin menghapus data dengan kategoriid 18 muncul jendela pop up



Saat klik oke maka tampillannya



References

1. https://github.com/jeroennoten/Laravel-AdminLTE/wiki/Usage
2. https://yajrabox.com/docs/laravel-datatables/10.0/quick-starter