|  |  |  |
| --- | --- | --- |
| 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**

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

# 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.

# 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']);

});

# 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']);

}

}

# 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');

## Berbagi data dengan semua view

Terkadang, anda mungkin perlu berbagi data dengan semua tampilan yang dirender 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');

}

}

## 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.

### 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 dirender 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”.

### 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

## 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/.](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.

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

composer require laravel/ui

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

php artisan ui bootstrap

php artisan ui bootstrap --auth

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

npm install

* 1. 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

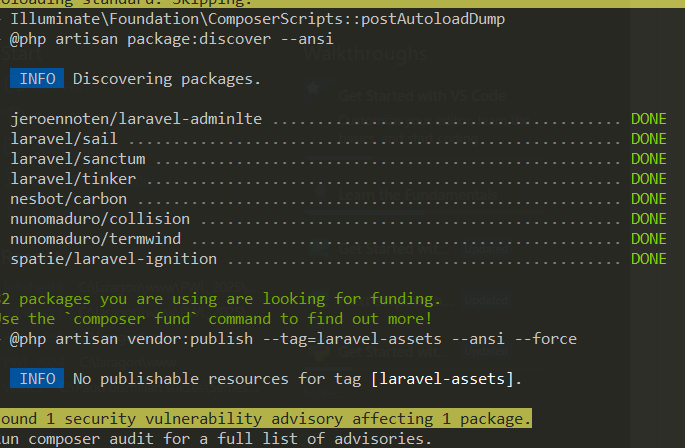
<script type="text/javascript" src="{{ asset('/js/app.js') }}"></script>

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

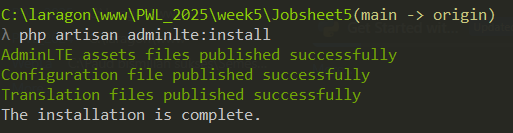
## Praktikum 1 – Integrasi Laravel dengan AdminLte3

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.

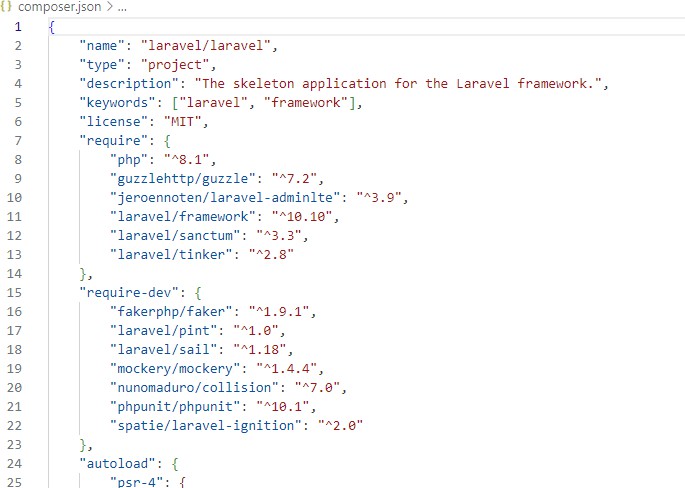
composer require jeroennoten/laravel-adminlte  


1. 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



1. 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

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



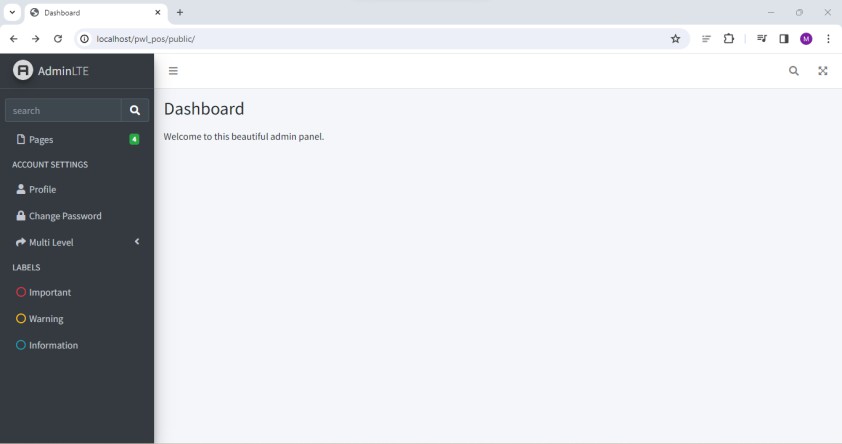
Untuk menggunakan blade template ini cukup dengan extend **AdminLTE layout** dengan cara @extends('adminlte::page').

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-](https://github.com/jeroennoten/Laravel-AdminLTE/wiki/Usage) [AdminLTE/wiki/Usage](https://github.com/jeroennoten/Laravel-AdminLTE/wiki/Usage)

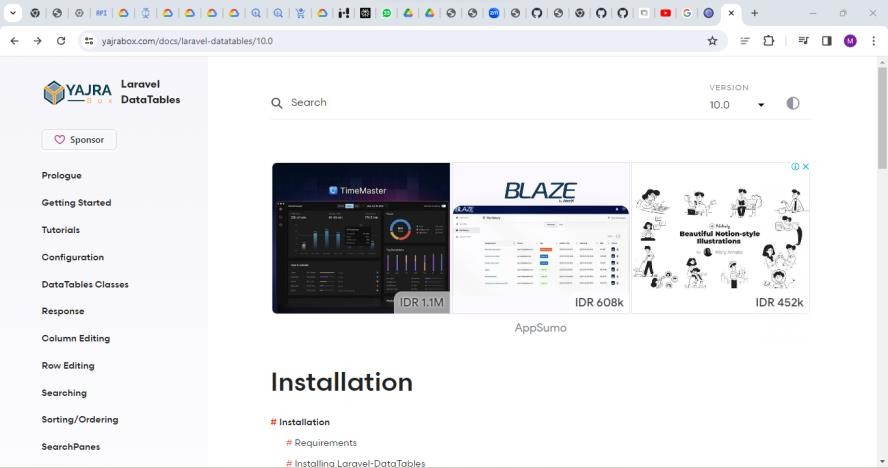
1. Kembali ke browser, menuju ke halaman awal.



## Praktikum 2 – Integrasi dengan DataTables

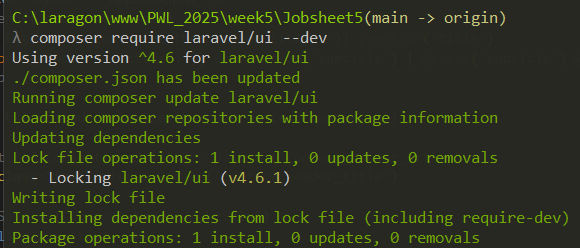
Datatables biasa digunakan untuk menampilkan list data dengan banyak fitur seperti searching, pagination, sorting dan lain-lainnya. Dalam laravel biasa digunankan yajra datatables karena didesain agar sesuai dengan laravel.

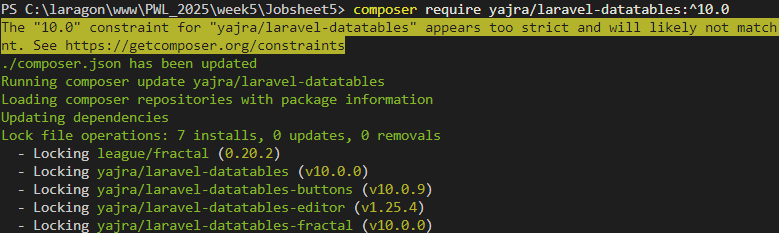
<https://yajrabox.com/docs/laravel-datatables/10.0>



Berikut langkah-langkah praktikum

1. Install Laravel DataTables

composer require laravel/ui –dev  


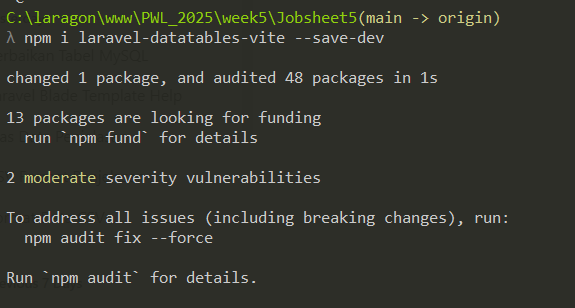
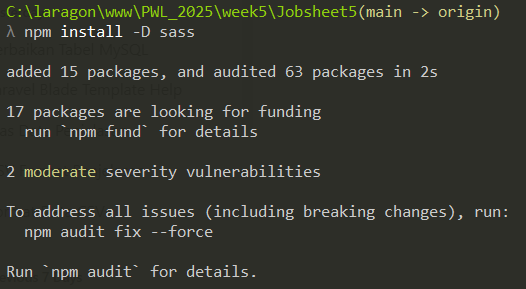
composer require yajra/laravel-datatables:^10.0  


1. Pastikan nodejs sudah terinstall, dengan perintah npm -v.

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

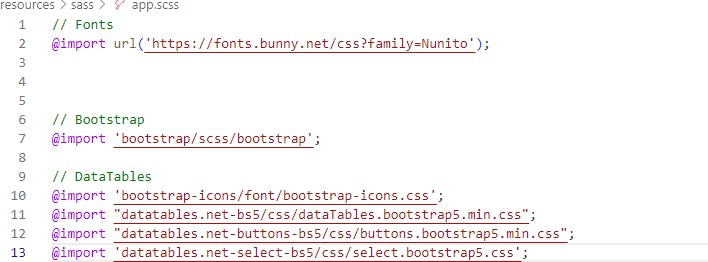
Npm digunakan untuk manajemen package javascript.

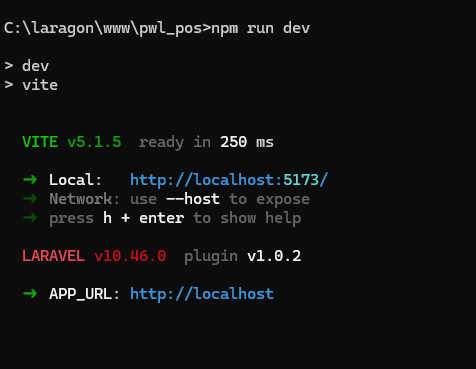
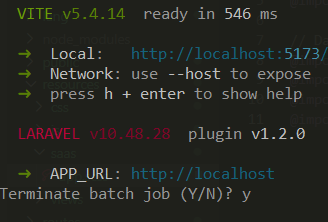
1. Install Laravel DataTables Vite dan sass

npm i laravel-datatables-vite --save-dev   
  
npm install -D sass  


1. Edit file resources/js/app.js



1. Buatlah file resources/saas/app.scss
2. Jalankan dengan npm run dev

1. Selanjutnya kita akan buat datatables untuk kategori

php artisan datatables:make Kategori





1. 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

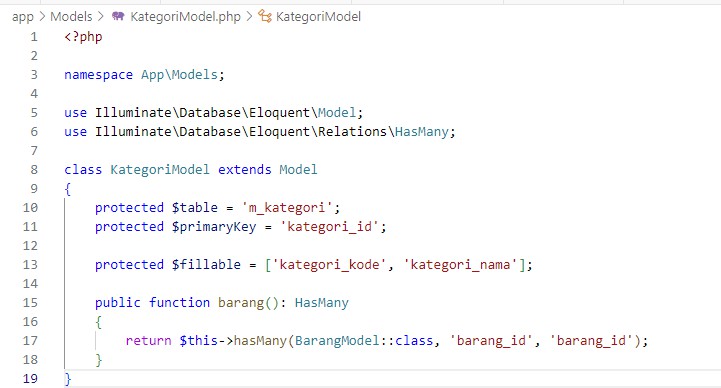
{

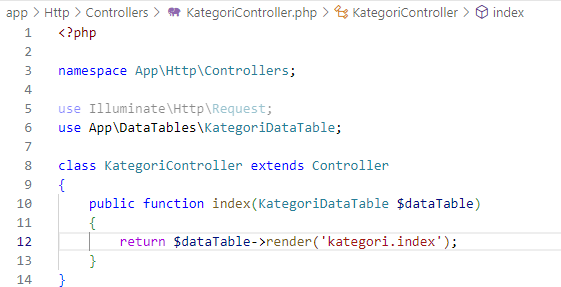
return 'Kategori\_' . date('YmdHis');

}

}

1. Ubah kategori model, sesuaikan seperti berikut



1. Ubah Kategori Controller sesuaikan dengan skrip berikut:
2. Buat folder kategori di dalam resources/view, kemudian buat view blade index untuk kategori resources/views/kategori/index.blade.php



1. Pastikan route untung kategori sudah tersedia



1. 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 {

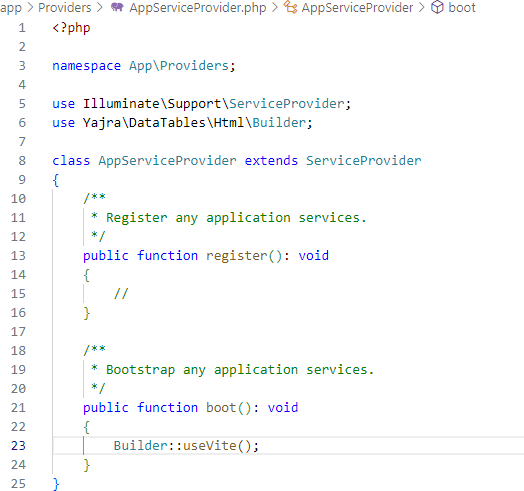
font-weight: 600;

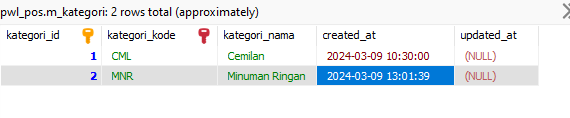
}

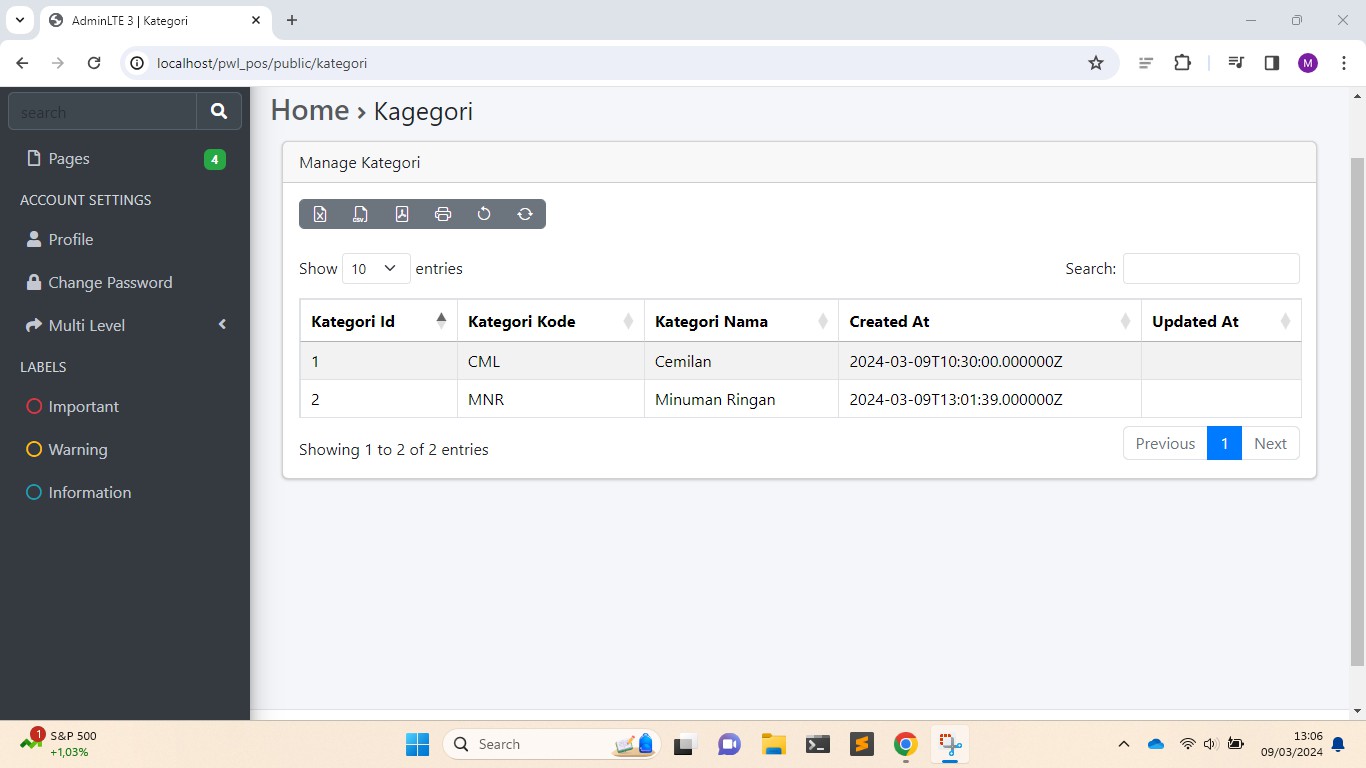
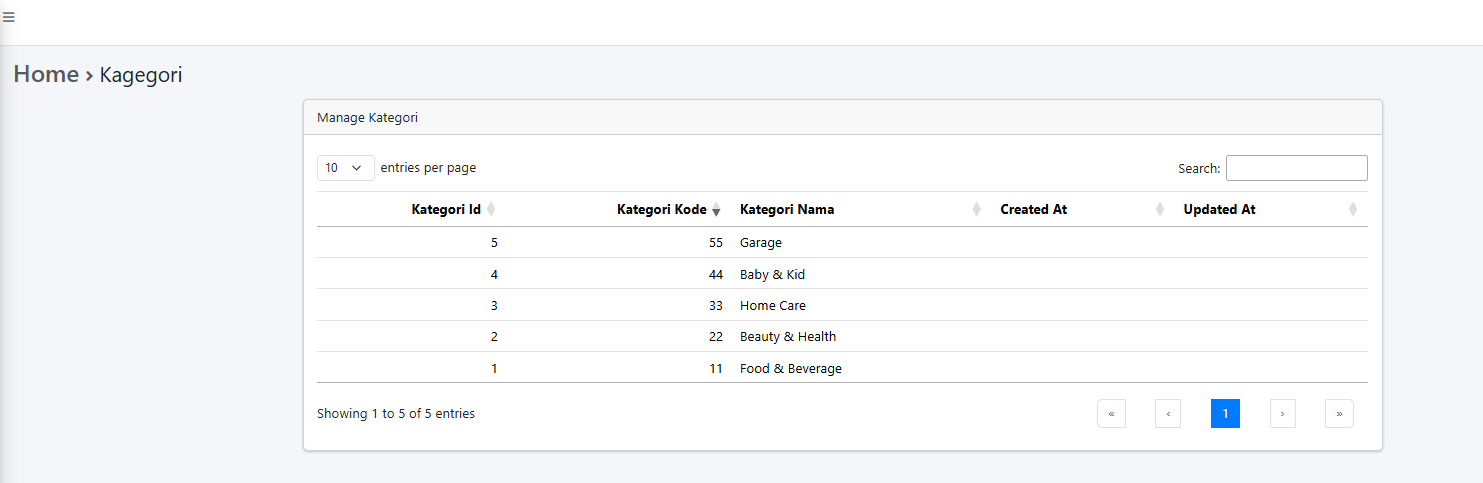
\*/

</style> @endpush

1. Menset ViteJs / script type defaults



1. Isikan beberapa data ke table kategori  
   
2. Datatables sudah dapat di load di /kategori



## Praktikum 3 – Membuat form kemudian menyimpan data dalam database

Langkah-langkahnya adalah sebagai berikut:

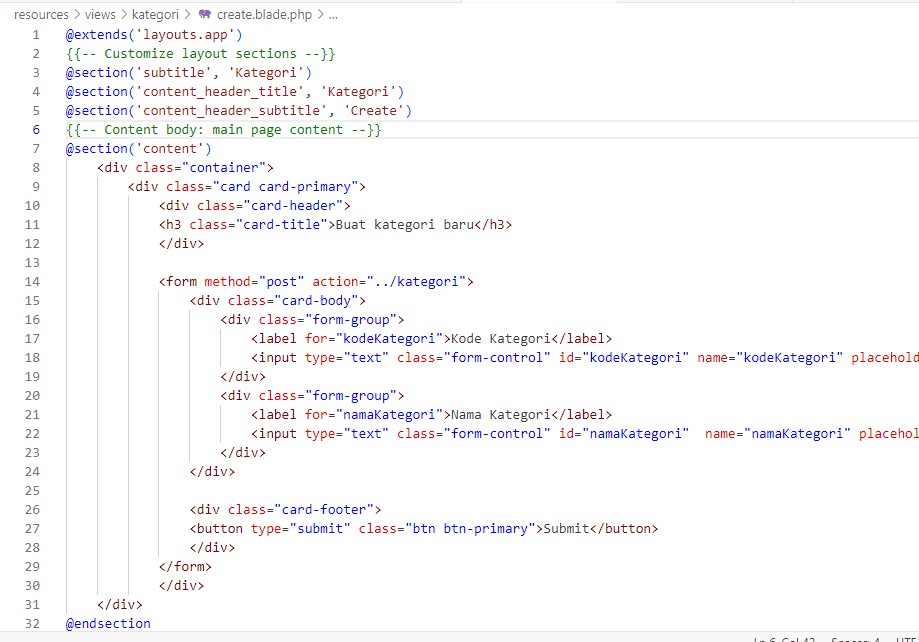
1. Menyesuaikan routing, tambahkan dua routing berikut



1. Tambahkan dua function berikut dalam KategoriController



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

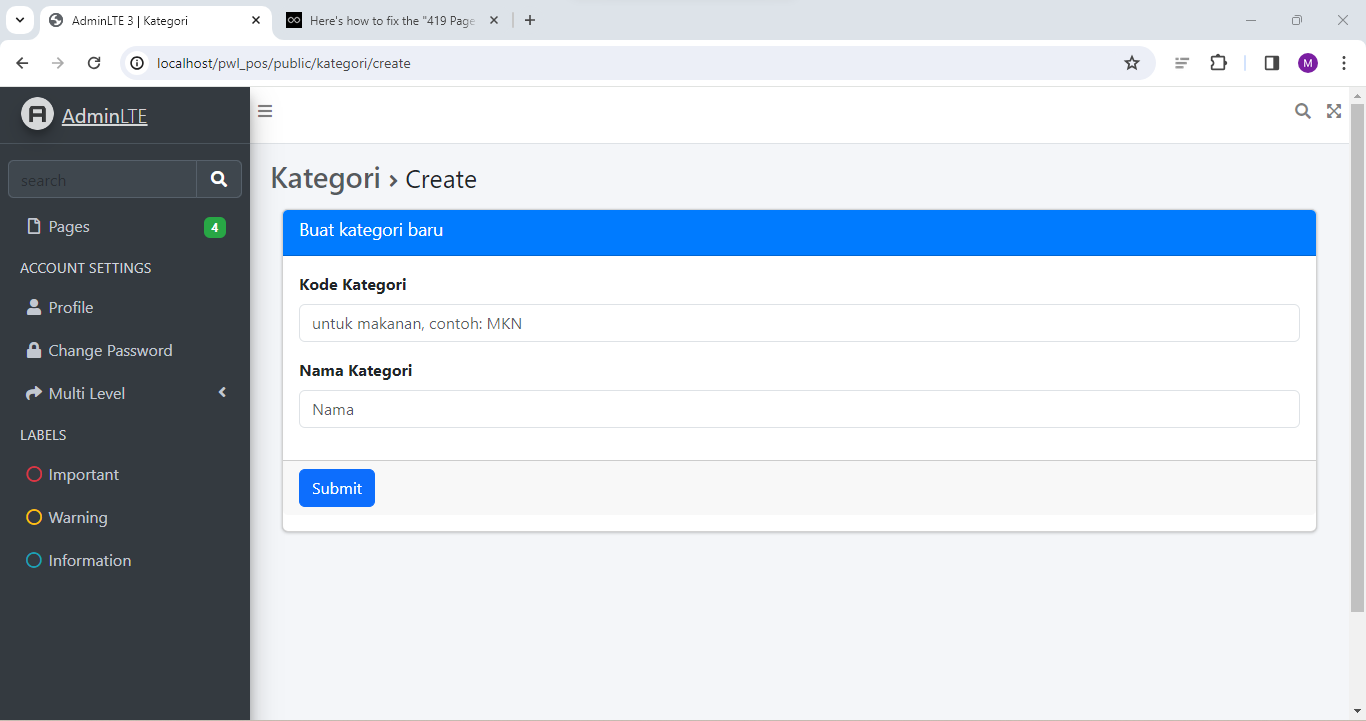
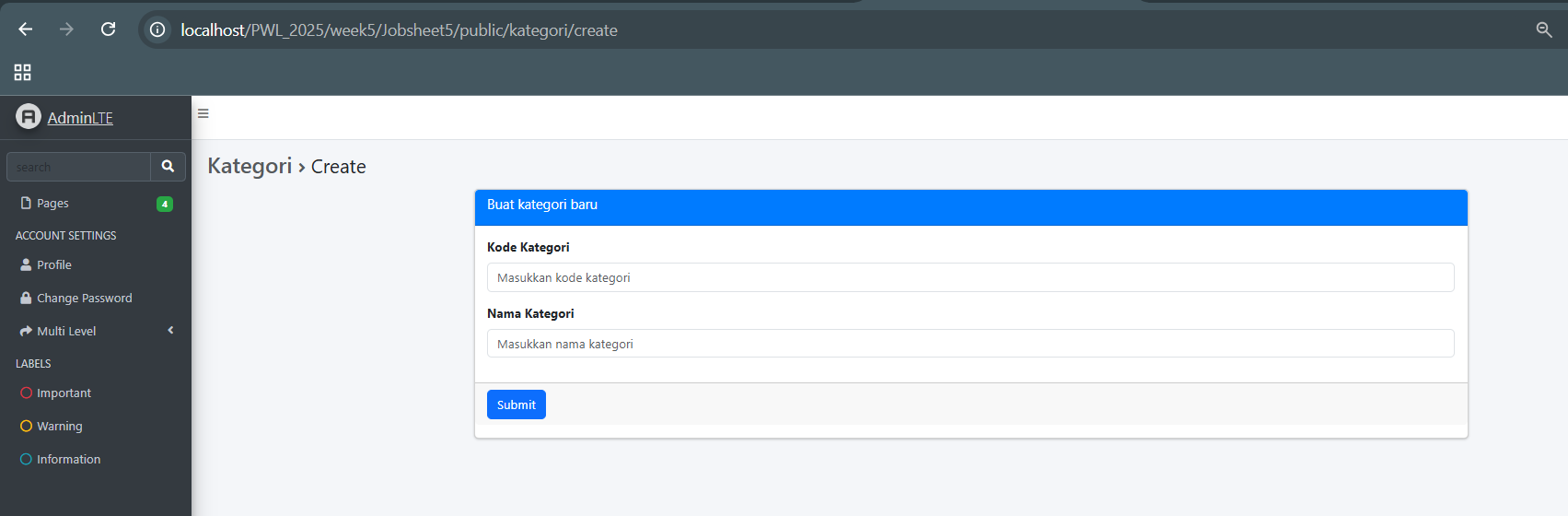


1. Kita lakukan pengecualian proteksi CsrfToken. Karena kita belum melakukan otentikasi

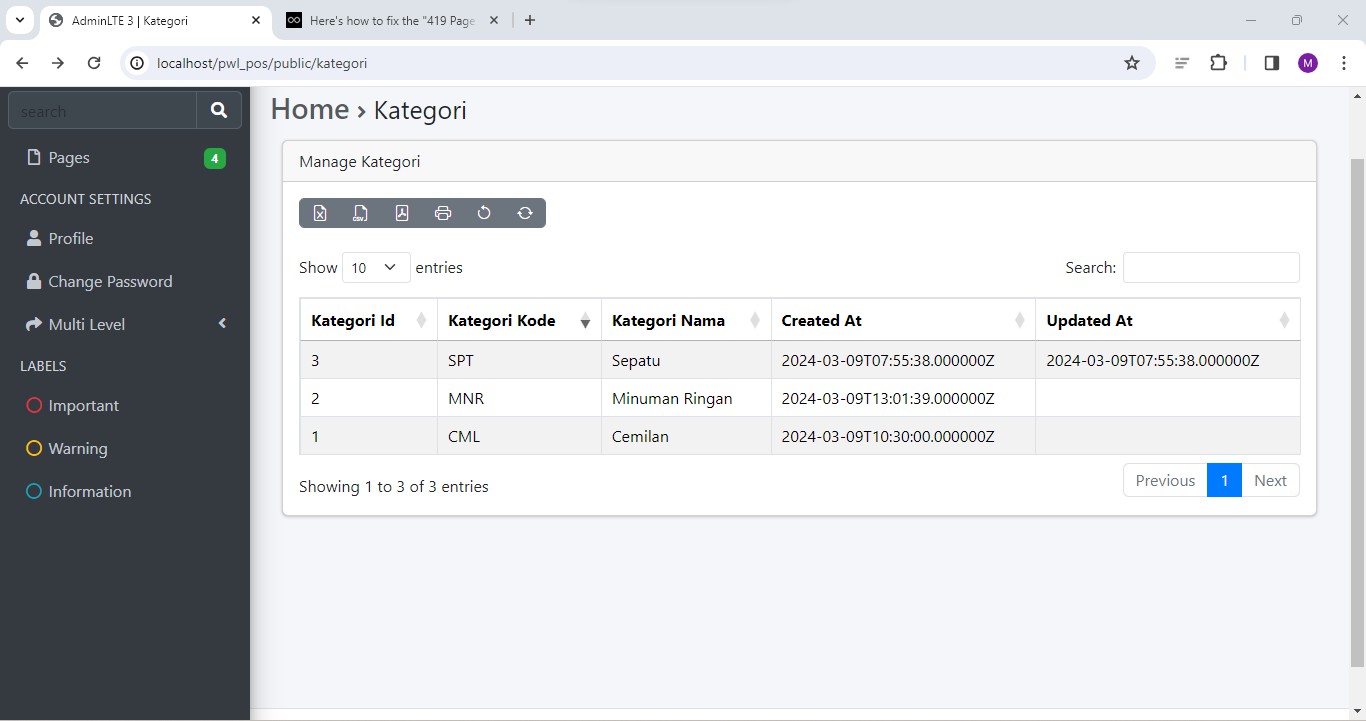
. Kita edit dalam file berikut:



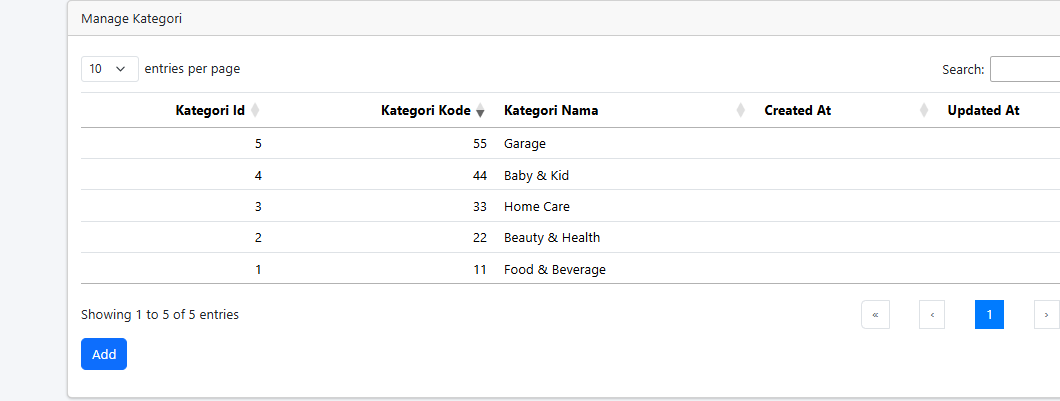
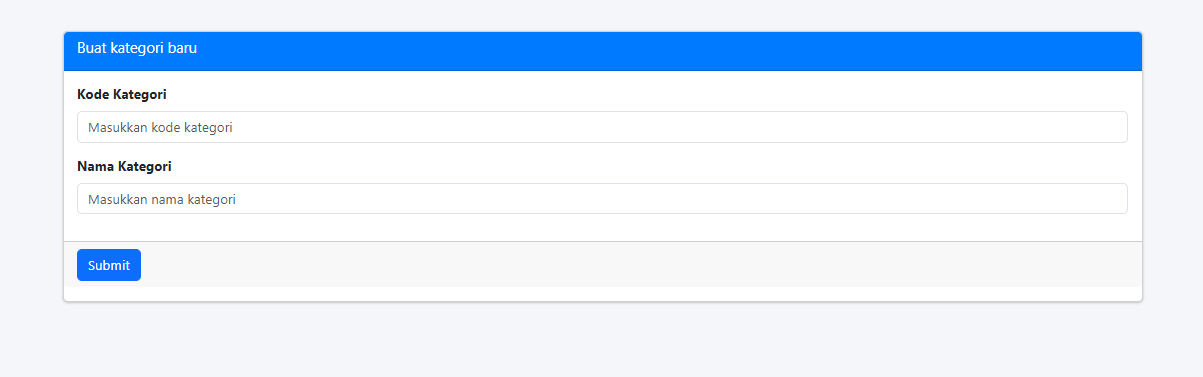
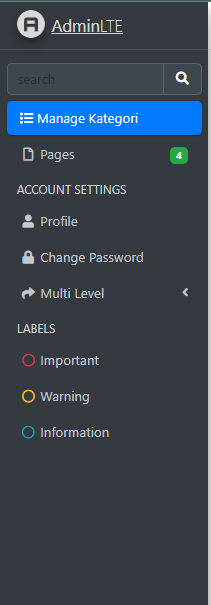
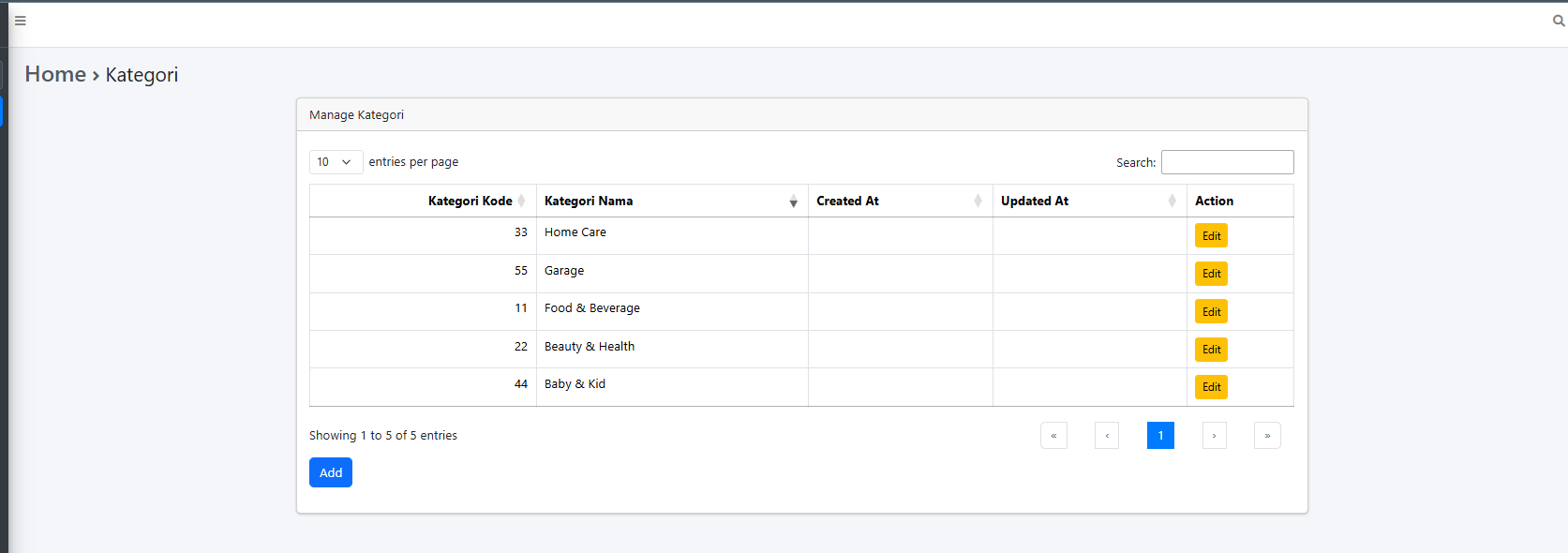
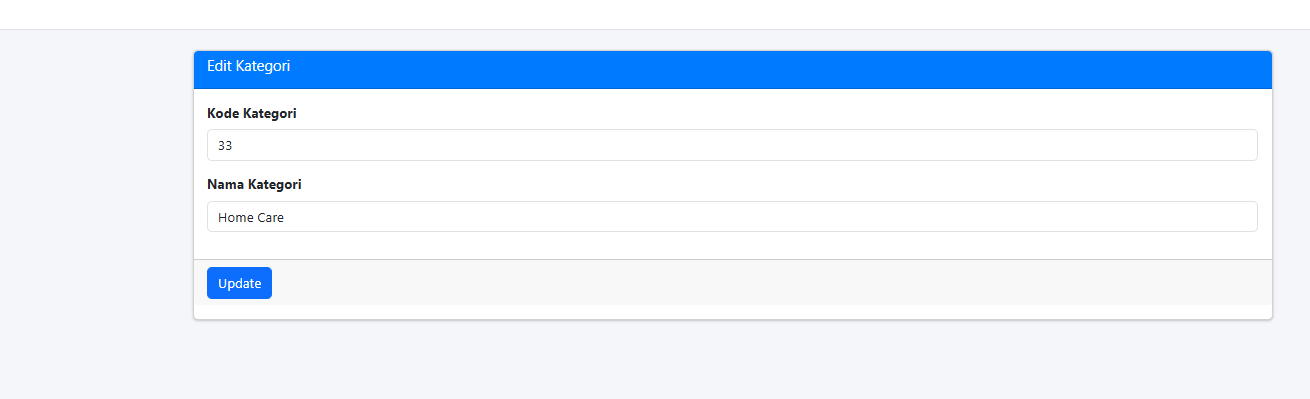
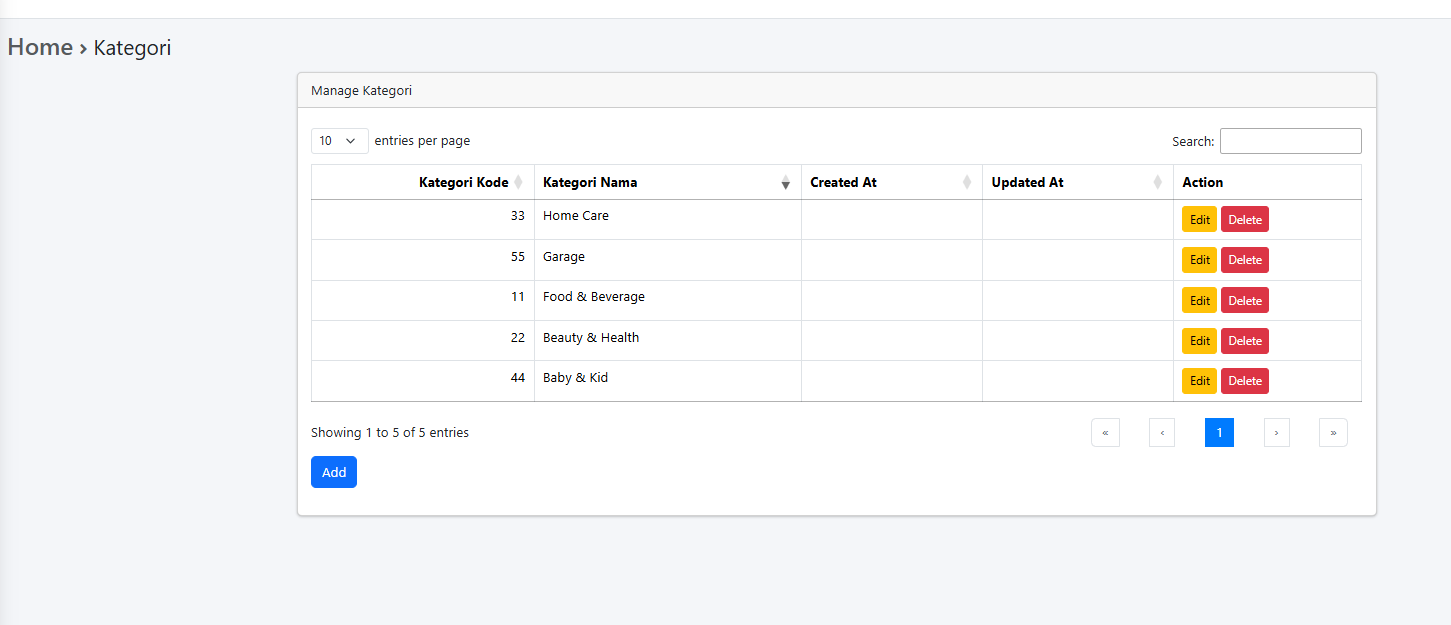
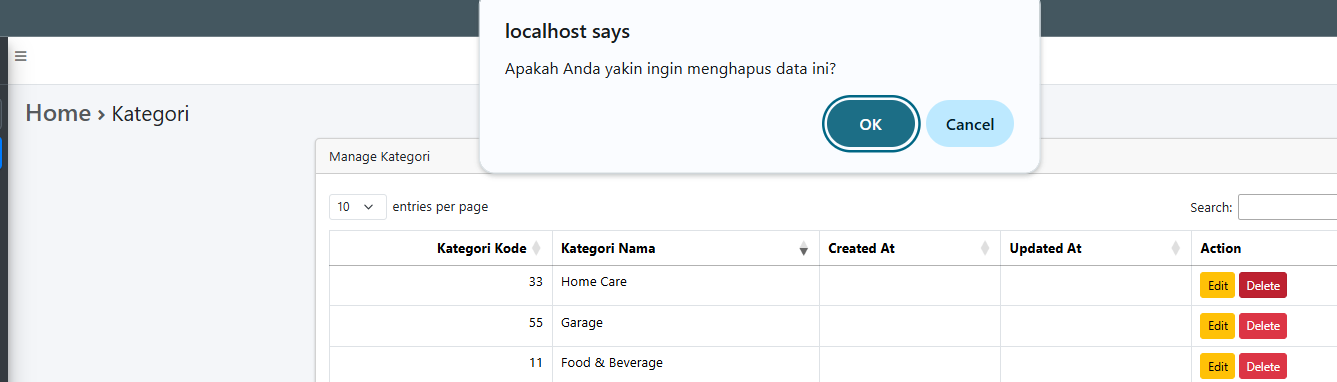
1. Akses kategori/create

1. Halaman kategori



## Tugas Praktikum

1. Tambahkan button Add di halam manage kategori, yang mengarah ke create kategori baru  
     
   
2. Tambahkan menu untuk halaman manage kategori, di daftar menu navbar  
   
3. Tambahkan action edit di datatables dan buat halaman edit serta controllernya  
     
   
4. Tambahkan action delete di datatables serta controllernya  
     
   

## References

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