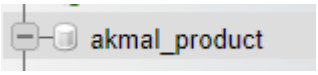
Nama : Akmal Muhamad Firdaus

NIM : 1301204188

Jurnal Praktikum Modul 7

1. **Instalasi Laravel**
   Karena dimodul sebelumnya sudah melakukan instalasi, jadi pada praktikum modul 7 ini akan menggunakan codebase pada modul 6 dan akan di modifikasi sesuai kebutuhan modul praktikum.
2. **Konfigurasi dan Skema**
   Jalankan xampp (Apache & MySQL), lalu masuk ke http://localhost/phpmyadmin dan buat sebuah database baru.



Hal pertama yang harus dilakukan agar aplikasi dapat terhubung dengan database adalah mengatur konfigurasi koneksi. Secara detil, konfigurasi database berada pada file config/database.php. Tetapi untuk konfigurasi standar, kita dapat mengubahnya di file .env pada baris yang berisi DB_CONNECTION hingga DB_PASSWORD. Ubah nilainya dengan pengaturan yang kita inginkan.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=akmal_product
DB_USERNAME=root
DB_PASSWORD=
```

Langkah selanjutnya adalah membuat membuat migration dengan perintah *php artisan make:migration nama_table*

```
C:\xampp\htdocs\ABP\Modul 7>php artisan make:migration product

  INFO  Migration [C:\xampp\htdocs\ABP\Modul 7\database\migrations/2023_04_01_223501_product.php] created successfully.
```

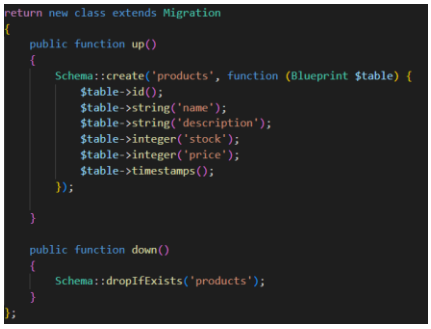
"*php artisan make:migration*" adalah perintah di dalam framework Laravel yang digunakan untuk membuat file migrasi database baru. Migrasi database digunakan untuk mengelola perubahan skema database pada aplikasi Laravel, seperti membuat, menghapus, atau memodifikasi tabel dan kolom di dalam database. Laravel akan secara otomatis membuat file migrasi baru di direktori database/migrations dalam proyek Larave. Kemudian dapat membuka file tersebut dan menentukan perubahan skema database yang ingin dilakukan menggunakan metode bawaan Laravel seperti create, drop, table, addColumn, renameColumn, dan lain-lain.

Di dalam Laravel, setiap file migrasi database terdiri dari dua method utama, yaitu up() dan down().

Method up() digunakan untuk mendefinisikan perubahan yang akan dilakukan pada skema database, seperti membuat tabel baru, menambahkan kolom baru, atau mengubah tipe data kolom. Ketika file migrasi dijalankan dengan perintah *php artisan migrate*, method up() akan dijalankan dan perubahan skema database akan diterapkan.

Sebaliknya, method down() digunakan untuk mendefinisikan operasi yang harus dilakukan untuk mengembalikan skema database ke keadaan semula sebelum perubahan dilakukan. Misalnya, jika ingin membuat kolom baru di dalam tabel menggunakan method up(), maka di dalam method down() harus menuliskan operasi untuk menghapus kolom tersebut. Method down() akan dijalankan ketika menjalankan perintah *php artisan migrate:rollback* untuk membatalkan migrasi database yang telah dijalankan sebelumnya.

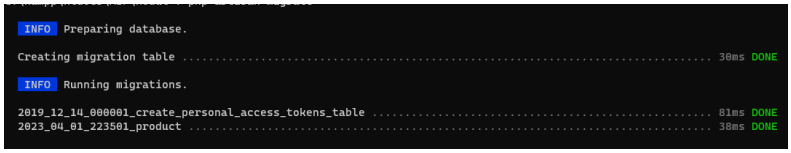
Dengan adanya method up() dan down(), kita dapat membuat migrasi database secara terstruktur dan dapat dengan mudah memperbarui atau membatalkan perubahan pada skema database. Ubah function **up** dan **down** menjadi seperti berikut:



```
return new class extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('description');
            $table->integer('stock');
            $table->integer('price');
            $table->timestamps();
        });

    }

    public function down()
    {
        Schema::dropIfExists('products');
    }
};
```

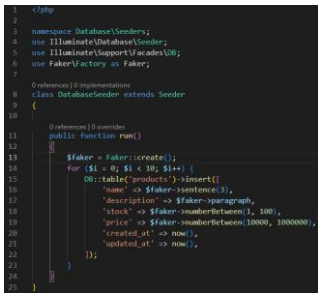
- Fungsi id() yaitu kita mendefinisikan kolom id sebagai PK dengan tipe int dan auto-increment.
- Fungsi string(‘name’) yaitu kita mendefinisikan kolom name dengan tipe varchar
- Fungsi string(‘description’) yaitu kita mendefinisikan kolom description dengan tipe varchar
- Fungsi integer(‘stock’) yaitu kita mendefinisikan kolom stock dengan tipe int
- Fungsi integer(‘price’) yaitu kita mendefinisikan kolom price dengan tipe int
- Fungsi timestamps() yaitu kita mendefinisikan kolom created_at yang akan terisi jika data dibuat melalui ORM dan updated_at yang akan terisi jika data diubah melalui ORM

Lalu jalankan perintah *php artisan migrate*



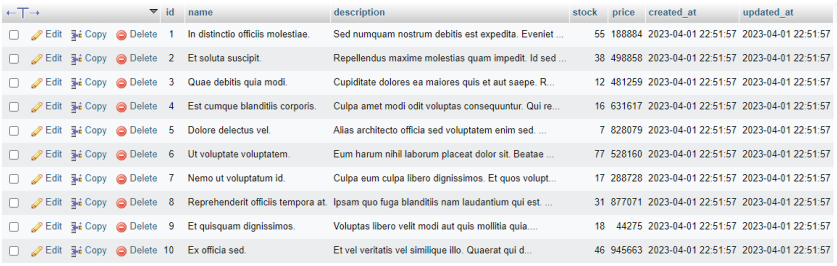
```
INFO  Preparing database.

Creating migration table ..... 30ms DONE

INFO  Running migrations.

2019_12_14_000001_create_personal_access_tokens_table ..... 81ms DONE
2023_04_01_223501_product ..... 38ms DONE
```

Selanjutnya adalah mencoba membuat seeder pada tabel products. Seeder adalah fitur di dalam Laravel yang digunakan untuk mengisi data ke dalam database secara otomatis. Seeder dapat digunakan untuk membuat data dummy atau sample data di dalam database agar mudah digunakan dalam pengembangan dan pengujian aplikasi. Modifikasi file yang berada pada database/seeders/DatabaseSeeder.php, lalu tambahkan seeder sebagai berikut.



```
<?php

namespace Database\Seeders;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Faker\Factory as Faker;

0 references | 0 implementations
class DatabaseSeeder extends Seeder
{

    0 references | 0 overrides
    public function run()
    {
        $faker = Faker::create();
        for ($i = 0; $i < 10; $i++) {
            DB::table('products')->insert([
                'name' => $faker->sentence(3),
                'description' => $faker->paragraph,
                'stock' => $faker->numberBetween(1, 100),
                'price' => $faker->numberBetween(10000, 1000000),
                'created_at' => now(),
                'updated_at' => now(),
            ]);
        }
    }
}
```

Untuk menjalankannya, gunakan perintah *php artisan db:seed,* maka secara otomatis, tabel products akan diisi dengan data dummy atau sample.



	id	name	description	stock	price	created_at	updated_at
Edit Copy Delete	1	In distinctio officiis molestiae.	Sed numquam nostrum debitis est expedita. Eveniet ...	55	188884	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	2	Et soluta suscipit.	Repellendus maxime molestias quam impedit. Id sed ...	38	498858	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	3	Quae debitis quia modi.	Cupiditate dolores ea maiores quis et aut saepe. R...	12	481259	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	4	Est cumque blanditiis corporis.	Culpa amet modi odit voluptas consequuntur. Qui re...	16	631617	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	5	Dolore delectus vel.	Alias architecto officia sed voluptatem enim sed. ...	7	828079	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	6	Ut voluptate voluptatem.	Eum harum nihil laborum placeat dolor sit. Beatae ...	77	528160	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	7	Nemo ut voluptatum id.	Culpa eum culpa libero dignissimos. Et quos volupt...	17	288728	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	8	Reprehenderit officiis tempora at.	Ipsam quo fuga blanditiis nam laudantium qui est. ...	31	877071	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	9	Et quisquam dignissimos.	Voluptas libero velit modi aut quis mollitia quia....	18	44275	2023-04-01 22:51:57	2023-04-01 22:51:57
Edit Copy Delete	10	Ex officia sed.	Et vel veritatis vel similique illo. Quaerat qui d...	46	945663	2023-04-01 22:51:57	2023-04-01 22:51:57

3. Model

Model adalah fitur di dalam Laravel yang digunakan untuk mengakses dan memanipulasi data di dalam database. Model merupakan representasi objek dari tabel di dalam database, sehingga setiap operasi yang dilakukan pada model akan diimplementasikan ke dalam tabel yang sesuai.

Laravel memberikan tiga cara untuk mengakses ataupun manipulasi data ke database, yaitu query langsung, query builder, dan ORM. Query langsung dan query builder menggunakan library Illuminate (Illuminate\Support\Facades\DB) sedangkan ORM menggunakan Eloquent, yang merupakan kelas extend dari Illuminate. File Model diletakkan pada folder app/Models. Untuk membuat Model dalam Laravel, dapat dilakukan dengan manual dengan menambahkan namespace “App\Models” dan meng-extend kelas base Model dari Eloquent (Illuminate\Database\Eloquent\Model) ataupun di-generate oleh Laravel dari command prompt / console / terminal dengan perintah: *php artisan make:model nama_model*

```
C:\xampp\htdocs\ABP\Modul 7>php artisan make:model ProductModel

  INFO  Model [C:\xampp\htdocs\ABP\Modul 7\app/Models/ProductModel.php] created successfully.
```

Jika sebelumnya belum membuat Controller atau file migration-nya, perintah generate Model ini dapat ditambahkan parameter sehingga bisa sekaligus me-generate file Controller-nya (-c), file migration-nya (-m), ataupun keduanya langsung (-cm).

Eloquent memiliki beberapa konvensi / aturan yang harus diperhatikan yaitu:

- Nama sebuah Model “X” secara otomatis merepresentasikan tabel database bernama “xs”. Contoh, Model Product akan merepresentasikan tabel **products**. Jika tabel yang direpresentasikan berbeda, maka harus ditambahkan atribut **protected $table = 'nama_tabel’;** pada kelas Model
- Kolom PK pada tabel bernama “id”. Jika kolom PK bukan “id”, maka harus ditambahkan atribut **protected $primaryKey = 'nama_kolom_PK';** pada kelas Model.
- Kolom PK pada tabel di-set auto-increment. Jika kolom PK tidak auto-increment, maka harus ditambahkan atribut **public $incrementing = false;** pada kelas Model.
- Kolom PK pada tabel bertipe integer. Jika kolom PK bukan integer, maka harus ditambahkan atribut **protected $keyType = ‘string’;** pada kelas Model.
- Ada kolom “created_at” dan “updated_at” pada tabel. Jika tidak ada, maka harus ditambahkan atribut public $timestamps = false; pada kelas Model.

4. **Controller**

Setelah Model siap, kita tinggal memanggilnya pada Controller. Berikut kode lengkapnya dalam ProductController

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Products;
class ProductController extends Controller
{
    //Author: Akmal Muhamad Firdaus - 1301204188
    public function index()
    {
        $products = Products::get();
        //dd($products);
        return view('product.index', compact('products'));
    }

    public function create()
    {
        return view('product.form', [
            'title' => 'Tambah',
            'method' => 'POST',
            'action' => 'product'
            ]);
    }

    public function store(Request $request)
    {
        $prod = new Products;
        $prod->name = $request->name;
        $prod->description = $request->description;
        $prod->price = $request->price;
        $prod->stock = $request->stock;
        $prod->save();
        return redirect('/products')->with('msg', 'Berhasil menambahkan product!');
    }

    public function show($id)
    {
        return Products::find($id);
    }

    public function edit($id)
    {
        return view('product.form', [
            'title' => 'Edit',
            'method' => 'PUT',
            'action' => "product/$id",
            'data' => Products::find($id)
        ]);
    }

    public function update(Request $request, $id)
    {
        $prod = Products::find($id);
        $prod->name = $request->name;
        $prod->description = $request->description;
        $prod->price = $request->price;
        $prod->stock = $request->stock;
        $prod->save();
        return redirect('/product')->with('msg', 'Berhasil mengubah product!');
    }

    public function destroy($id)
    {
        Products::destroy($id);
        return redirect('/product')->with('msg', 'Berhasil menghapus product!');
    }
}
```

5. **View**

index.blade.php

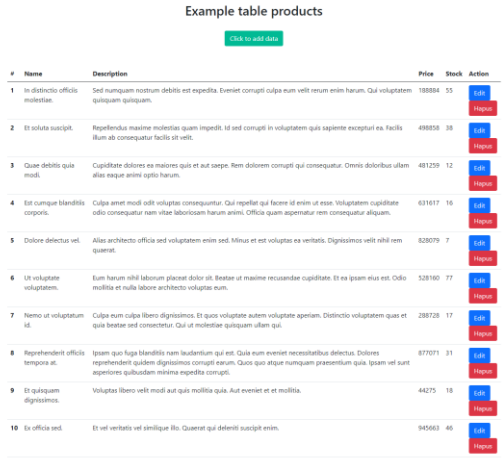
form.blade.php

Dikarenakan tampilan dari edit dan tambah (add) itu sama, jadi cukup menggunakan 1 file view saja tetapi diberikan kondisi pada viewnya.

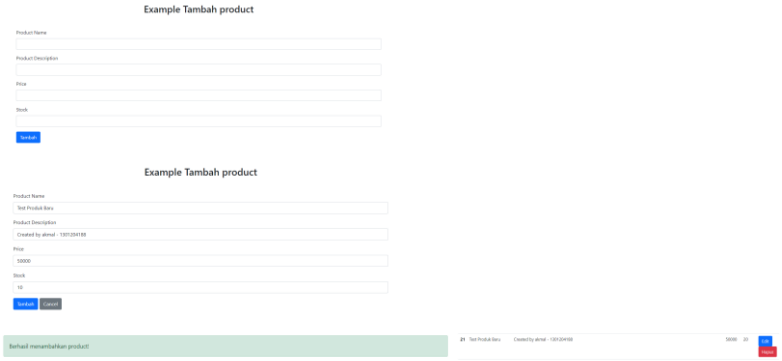
```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <link href="{{ asset('custom.css') }}" rel="stylesheet">
    <title>Modul 7!</title>
  </head>
  <body>
    <section class="bg-white pt-10 py-10">
        <div class="container px-5">
            <div class="text-center mb-5">
                <h2>Example {{ $title }} product</h2>
            </div>
            <div class="row gx-5 y-1">
              <div class="col-md-12">
                <form method="POST" action="/{{$action}}">
                    @csrf
                    @method($method)
                    <div class="mb-3">
                        <label for="name" class="form-label">Product Name</label>
                        <input type="text" class="form-control" id="name" name="name" value="
{{isset($data)?$data->name:'' }}" >
                    </div>
                    <div class="mb-3">
                        <label for="description" class="form-label">Product Description</label>
                        <input type="text" class="form-control" id="description" name="description"
value="{{isset($data)?$data->description:'' }}">
                    </div>
                    <div class="mb-3">
                        <label for="price" class="form-label">Price</label>
                        <input type="number" class="form-control" id="price" name="price" value="
{{isset($data)?$data->price:'' }}">
                    </div>
                    <div class="mb-3">
                        <label for="stock" class="form-label">Stock</label>
                        <input type="number" class="form-control" id="stock" name="stock" value="
{{isset($data)?$data->stock:'' }}">
                    </div>
                    <button type="submit" class="btn btn-primary">{{ $title }} </button>
                    <a class="btn btn-secondary" href="/products"> Cancel </a>
                </form>
              </div>
            </div>
        </div>
    </section>
    <script src="https://code.jquery.com/jquery-3.6.4.min.js" integrity="sha256-
oP6HI9z1XaZNBrJURtCoUT5SUnxFr8s3BzRl+cbzUq8=" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
  </body>
</html>
```

6. Tampilan Halaman

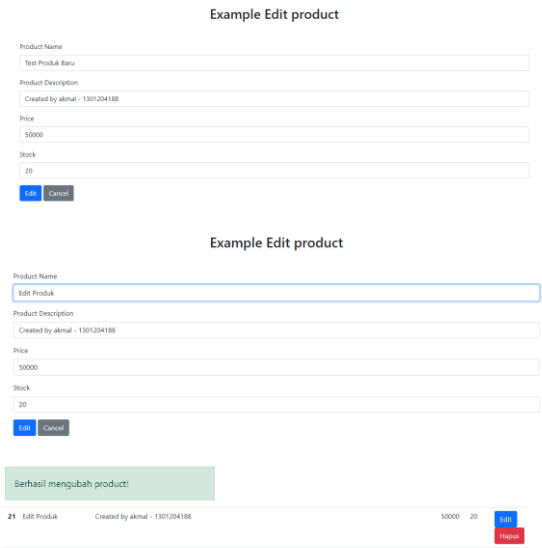
1. http://127.0.0.1:8000/products



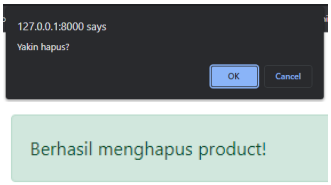
2. http://127.0.0.1:8000/products/create



3. http://127.0.0.1:8000/products/21/edit



4. http://127.0.0.1:8000/products/1 (Delete)



7. Templating Halaman

Templating adalah fitur yang digunakan untuk memisahkan logika tampilan (view) dari logika aplikasi (controller). Pada Laravel, templating dikenal sebagai Blade Templating Engine, yang menyediakan sintaks khusus untuk membuat tampilan yang terstruktur dan mudah dibaca.

Dengan menggunakan Blade, kita dapat membuat tampilan HTML yang dapat digunakan ulang di seluruh aplikasi. Blade menyediakan fitur seperti template inheritance, sections, dan directives, yang memudahkan pengembang untuk membuat tampilan yang konsisten dan mudah dipelihara.

Beberapa fitur utama dari Blade Templating Engine di Laravel antara lain:

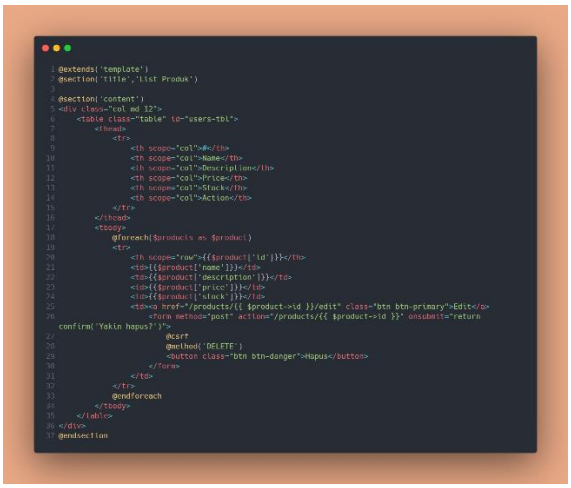
- Template Inheritance: Kita dapat membuat layout utama (master template) yang dapat digunakan ulang di seluruh tampilan aplikasi, sehingga memudahkan untuk membuat tampilan yang konsisten.
- Sections: Kita dapat menggunakan section untuk mengisi konten spesifik pada halaman tertentu. Hal ini memudahkan untuk membuat halaman yang dinamis dan dapat digunakan kembali.
- Directives: Blade menyediakan direktif (directive) seperti @if, @foreach, @include, dan lain-lain, yang memudahkan pengembang untuk membuat tampilan yang kompleks dan dinamis.

Langkah pertama dalam templating adalah membuat file template-nya, yang dimiliki oleh tiap halaman. Untuk contoh kasus pada modul ini, kita membuat file template dengan nama template.blade.php dengan isi:

```blade
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
        integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <link href="{{ asset('custom.css') }}" rel="stylesheet">
    <title>Modul 7!</title>
</head>

<body>
    <section class="bg-white pt-10 py-10">
        <div class="container px-5">
            <div class="text-center mb-5">
                <h2>Example table products</h2>
                <a class="btn btn-teal mt-4" class="btn btn-primary" href="/products/create">Click to
add data</a>
            </div>
            <div class="row gx-5 z-1">
                @if(session('msg'))
                    <span class="alert alert-success mb-2">{{ session('msg') }}</span>
                @endif
                @if(session('error'))
                    <span class="alert alert-danger mb-2">{{ session('error') }}</span>
                @endif

                @yield('content')
            </div>
        </div>
    </section>
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"
        integrity="sha256-oP6HI9z1XaZNBrJURtCoUT5SUnxFr8s3BzRl+cbzUq8=" crossorigin="anonymous">
</script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-MrcW6/MFYlzcLA8Nl+NtUVF0sA/MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous">
    </script>
</body>

</html>
```

Bagian index.blade.php dan form.blade.php dimodifikasi menjadi menggunakan template menjadi sebagai berikut:

Index



```blade
@extends('template')
@section('title','List Produk')

@section('content')
<div class="col md 12">
    <table class="table" id="users-tbl">
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">Name</th>
                <th scope="col">Description</th>
                <th scope="col">Price</th>
                <th scope="col">Stock</th>
                <th scope="col">Action</th>
            </tr>
        </thead>
        <tbody>
            @foreach($products as $product)
            <tr>
                <th scope="row">{{$product['id']}}</th>
                <td>{{$product['name']}}</td>
                <td>{{$product['description']}}</td>
                <td>{{$product['price']}}</td>
                <td>{{$product['stock']}}</td>
                <td><a href="/products/{{ $product->id }}/edit" class="btn btn-primary">Edit</a>
                    <form method="post" action="/products/{{ $product->id }}" onsubmit="return
confirm('Yakin hapus?')">
                        @csrf
                        @method('DELETE')
                        <button class="btn btn-danger">Hapus</button>
                    </form>
                </td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
@endsection
```

Form

```blade
@extends('template')

@section('content')
<div class="col-md-12">
    <form method="POST" action="/{{$action}}">
        @csrf
        @method($method)
        <div class="mb-3">
            <label for="name" class="form label">Product Name</label>
            <input type="text" class="form-control" id="name" name="name" value="{{isset($data)?$data-
>name:'' }}">
        </div>
        <div class="mb 3">
            <label for="description" class="form-label">Product Description</label>
            <input type="text" class="form-control" id="description" name="description"
                value="{{isset($data)?$data->description:'' }}">
        </div>
        <div class="mb-3">
            <label for="price" class="form-label">Price</label>
            <input type="number" class="form-control" id="price" name="price" value="{{isset($data)?
$data->price:'' }}">
        </div>
        <div class="mb 3">
            <label for="stock" class="form-label">Stock</label>
            <input type="number" class="form control" id="stock" name="stock" value="{{isset($data)?
$data->stock:'' }}">
        </div>
        <button type="submit" class="btn btn-primary">{{ $title }} </button>
        <a class="btn btn-secondary" href="/products"> Cancel </a>
    </form>
</div>
@endsection
```

Login

```
@extends('template')
@section('title','Login Page')

@section('content')
<div class="col-md-12">
    <form method="POST" action="/auth">
        @csrf
        <div class="mb-3">
            <label for="email" class="form-label">Email</label>
            <input type="email" class="form-control @error('email') is-invalid @enderror" id="email"
name="email"
                value="{{ old('email') }}">
            @error('email')
            <div class="invalid feedback">{{ $message }}</div>
            @enderror

        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input type="password" class="form-control @error('password') is-invalid @enderror"
id="password"
                name="password" value="{{old('password') }}">
            @error('password')
                <div class="invalid feedback">{{ $message }}</div>
            @enderror
        </div>
        <button type="submit" class="btn btn-primary"> Login </button>
    </form>
</div>
@endsection
```

Directive @extend digunakan untuk menentukan file template mana yang digunakan oleh halaman ini. Sedangkan directive @section digunakan untuk mengisi halaman sesuai dengan nama yield yang di-define dalam file template. Directive @section dapat diisi dengan string ataupun tag HTML.

8. Form Validation

Form validation adalah fitur yang memungkinkan kita untuk memvalidasi input yang diterima dari pengguna sebelum data tersebut disimpan ke dalam database. Validasi dilakukan untuk memastikan bahwa data yang disubmit oleh pengguna memenuhi kriteria yang diinginkan, sehingga menghindari data yang tidak valid dari masuk ke dalam sistem.

Laravel menyediakan fitur validasi yang kuat dan mudah digunakan. Validasi dilakukan pada sisi server, sehingga memastikan data yang masuk ke dalam sistem telah melewati validasi yang benar dan aman. Beberapa fitur utama dari validasi pada Laravel antara lain:

- Rule: Laravel menyediakan berbagai macam rule yang dapat digunakan untuk validasi input, seperti required, numeric, email, unique, dan lain-lain.
- Custom Error Messages: Kita dapat membuat pesan error kustom untuk setiap aturan validasi yang diterapkan, sehingga memudahkan pengguna untuk memahami kesalahan yang terjadi.
- Form Request Validation: Laravel menyediakan fitur Form Request Validation, yang memungkinkan kita untuk memisahkan validasi dari logika controller. Hal ini memungkinkan kita untuk membuat kode yang lebih mudah dibaca dan dipelihara.
- Validasi pada Controller: Kita dapat melakukan validasi input pada logika controller dan menampilkan pesan error jika ada masalah pada input yang diterima.

Pada controller, sebelum kita memanggil fungsi untuk menambahkan (fungsi store) / mengubah (fungsi update), kita dapat menambahkan kode:

```php
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required|min:4',
        'description' => 'required|min:4',
        'price' => 'required|integer|min:1000',
        'stock' => 'required|integer|min:1'
    ]);
    $prod = new Products;
    $prod->name = $request->name;
    $prod->description = $request->description;
    $prod->price = $request->price;
    $prod->stock = $request->stock;
    $prod->save();
    return redirect('/products')->with('msg', 'Berhasil menambahkan product!');
}
```

```php
public function update(Request $request, $id)
{
    $this->validate($request, [
        'name' => 'required|min:4',
        'description' => 'required|min:4',
        'price' => 'required|integer|min:1000',
        'stock' => 'required|integer|min:1'
    ]);
    $prod = Products::find($id);
    $prod->name = $request->name;
    $prod->description = $request->description;
    $prod->price = $request->price;
    $prod->stock = $request->stock;
    $prod->save();
    return redirect('/products')->with('msg', 'Berhasil mengubah product!');
}
```

Di kode validasi ini, kita mengatur nama dan description tidak boleh kosong dan minimal 4 karakter, harga tidak boleh kosong, harus integer, dan nilai minimal 1.000, sedangkan stock tidak boleh kosong, harus integer, dan minimal 1. Kemudian di halaman form.blade.php kita harus menambahkan directive @error seperti ini:

```php
<form method="POST" action="/{{$action}}">
    @csrf
    @method($method)
    <div class="mb-3">
        <label for="name" class="form-label">Product Name</label>
        <input type="text" class="form-control @error('name') is-invalid @enderror" id="name" name="name" value="{{isset($data)?$data->name:old('name') }}">
        @error('name')
          <div class="invalid-feedback">{{ $message }}</div>
        @enderror

    </div>
    <div class="mb-3">
        <label for="description" class="form-label">Product Description</label>
        <input type="text" class="form-control @error('description') is-invalid @enderror" id="description" name="description"
            value="{{isset($data)?$data->description:old('description') }}">
            @error('description')
              <div class="invalid-feedback">{{ $message }}</div>
            @enderror
    </div>
    <div class="mb-3">
        <label for="price" class="form-label">Price</label>
        <input type="number" class="form-control @error('price') is-invalid @enderror" id="price" name="price" value="{{isset($data)?$data->price:old('price') }}">
        @error('price')
          <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>
    <div class="mb-3">
        <label for="stock" class="form-label">Stock</label>
        <input type="number" class="form-control @error('stock') is-invalid @enderror" id="stock" name="stock" value="{{isset($data)?$data->stock:old('stock') }}">
        @error('stock')
          <div class="invalid-feedback">{{ $message }}</div>
        @enderror
    </div>
    <button type="submit" class="btn btn-primary">{{ $title }} </button>
    <a class="btn btn-secondary" href="/products"> Cancel </a>
</form>
```

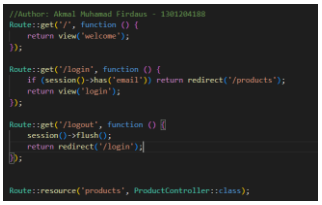
Directive @error diletakkan di atribut class pada tag <input> untuk validasinya dan diletakkan di bawah tag <input> untuk menampilkan pesan validasinya. Kemudian agar data pada form di-repopulate, maka isikan atribut value pada tag <input> dengan old('value-di-atribut-name').

Contoh ouput form validation

9. Session

Session memiliki fungsi menyimpan suatu variabel pada server dan variabel ini dapat diakses dan diubah dimanapun: Routing, Controller, ataupun halaman manapun. Session default pada Laravel dikelola sendiri oleh Laravel dan disimpan dalam bentuk file. Selain file, Session pada Laravel juga dapat diubah menjadi disimpan ke database atau menggunakan mekanisme lain.

Laravel memiliki dua jenis Session, yaitu Session biasa dan Session flash. Session biasa akan selalu ada selama belum dihapus, time-out, atau browser ditutup. Sedangkan Session flash adalah session yang hanya berlaku untuk satu request saja, setelah itu Laravel akan menghapusnya secara otomatis. Contoh Session flash sudah pernah kita gunakan melalui fungsi with('x', y) yang digunakan bersamaan dengan fungsi redirect. Untuk Session biasa, berikut contoh penggunaannya di Routing:



```
//Author: Akmal Muhamad Firdaus - 130120418
Route::get('/', function () {
    return view('welcome');
});

Route::get('/login', function () {
    if (session()->has('email')) return redirect('/products');
    return view('login');
});

Route::get('/logout', function () {
    session()->flush();
    return redirect('/login');
});

Route::resource('products', ProductController::class);
```

Fungsi session()->has('x') digunakan untuk memeriksa apakah ada Session bernama "x". Sedangkan session()->flush() digunakan untuk menghapus semua Session. Contoh lain, berikut penggunaan Session di SiteController:

```
class SiteController extends Controller
{
    0 references | 0 overrides
    public function auth(Request $req) {
        $u = User::where([
            ['email', $req->em],
            ['password', $req->pwd],
        ])->first();
        if (isset($u)) {
            session()->put('email', $u->email);
            session()->put('name', $u->name);
            return "<script>
            alert('Welcome, " . session('name') . "');
            location.href='/product';
            </script>";
        }
        return redirect('/login')->with('msg', 'Email / password salah');
    }
}
```

Fungsi session()->put('x', y) digunakan untuk membuat Session bernama "x" dengan nilai y. Sedangkan session('x') digunakan untuk mengambil nilai dari Session "x".

10. Middleware

Middleware pada Laravel adalah suatu mekanisme untuk menyaring request yang masuk ke suatu Routing. Sebagai contoh, kita bisa membuat Middleware untuk memverifikasi apakah seorang user sudah terautentikasi atau memiliki hak akses untuk mengakses URL. Jika user belum terautentikasi atau tidak memiliki akses, maka Middleware dapat me-redirect user tersebut ke URL lain. Sebaliknya jika user terautentikasi atau memiliki hak akses, maka Middleware akan meneruskan request ke aksi yang dipetakan di Routing.

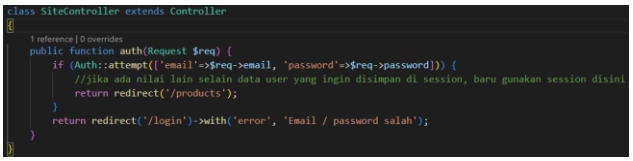
Middleware dapat melakukan hal lain selain untuk autentikasi, misalnya untuk menulis log atau error yang terjadi. Dan Middleware dapat kita atur mekanisme yang dilakukannya, yaitu bisa sebelum request diteruskan atau sesudah request diteruskan. Middleware yang kita buat harus berada di folder **app/Http/Middleware.**

Sekarang kita akan membuat autentikasi dengan menggunakan library Auth. Library ini sudah termasuk di dalam paket instalasi Laravel, sehingga kita tidak perlu menambahkannya lagi melalui composer. Dalam library ini sudah mencakup semua fitur untuk autentikasi, registrasi, dan middleware-nya. Dengan menggunakan Auth, autentikasi menjadi lebih simpel, aman, dan bisa menjadi alternatif pengganti yang lebih baik dari Session.

Langkah pertama adalah pengaturan pada Routing. Ganti Routing untuk login, logout, dan product:

```php
//Author: Akmal Muhamad Firdaus - 1301204188
Route::get('/', function () {
    return view('welcome');
});

Route::get('/login', function () {
    if (Auth::check()) return redirect('/product');
    return view('login');
})->name('login');


Route::get('/logout', function () {
    Auth::logout();
    return redirect('/login');
});

Route::post('/auth', [SiteController::class, 'auth']);


Route::resource('products', ProductController::class)->middleware('auth');
```

Fungsi Auth::check() digunakan untuk memeriksa apakah user telah terautentikasi. Routing login kita berikan nama alias karena kebutuhan dari Middleware auth. Sedangkan Auth::logout() digunakan untuk menghapus autentikasi. Untuk product, Middleware kita ganti dengan Middleware auth yang berkolaborasi dengan library Auth.
Lalu langkah kedua ubah isi dari fungsi auth di SiteController:



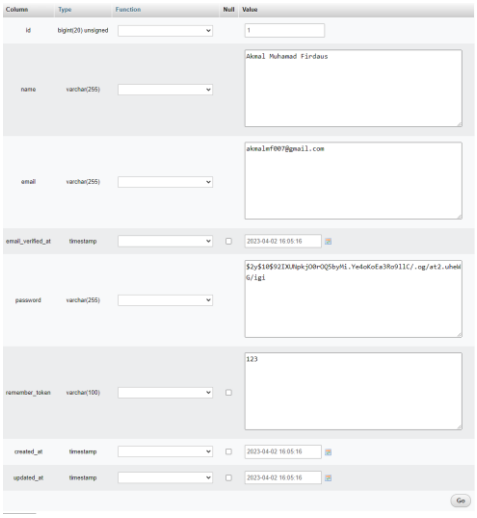
```php
class SiteController extends Controller
{
    1 reference | 0 overrides
    public function auth(Request $req) {
        if (Auth::attempt(['email'=>$req->email, 'password'=>$req->password])) {
            //jika ada nilai lain selain data user yang ingin disimpan di session, baru gunakan session disini
            return redirect('/products');
        }
        return redirect('/login')->with('error', 'Email / password salah');
    }
}
```

Fungsi Auth::attempt() digunakan untuk proses autentikasi, yaitu apakah email dan password yang di-submit ada dalam database pada tabel users. Jika email dan password cocok maka akan menghasilkan true. Hal yang harus diperhatikan adalah ketika menambahkan data User ke database, pastikan isi dari kolom password di-hash dengan metode Bcrypt agar dapat menggunakan library Auth ini. Laravel sudah menyediakan fungsi bcrypt('x') untuk mempermudahnya

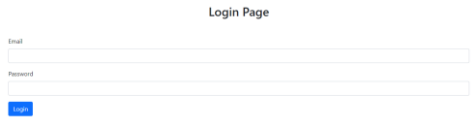
Langkah ketiga, dalam View yang sebelumnya menggunakan @if(session('x')) dapat kita ganti menjadi directive @auth untuk pengecekan apakah user telah terautentikasi, dan dapat menggunakan Auth::user() untuk mengakses data user yang login. Berikut contohnya dalam template.blade.php:

```
@auth
<div class="row justify-content-end" style="margin-top:2%">
    <div class="col-3">
        {{ Auth::user()->name }}
        <a href="/logout" class="btn btn-warning">Logout</a>
    </div>
</div>
@endauth
```

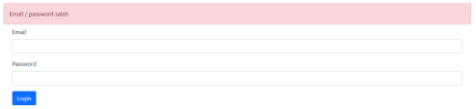
Untuk melakukan pengecekan bahwa middleware auth kita sudah benar, kita dapat langsung saja mencobanya dengan menambahkan data barus secara manual pada DBMS kita. Perlu diingat kolom password gunakan enkripsi bcrypt.



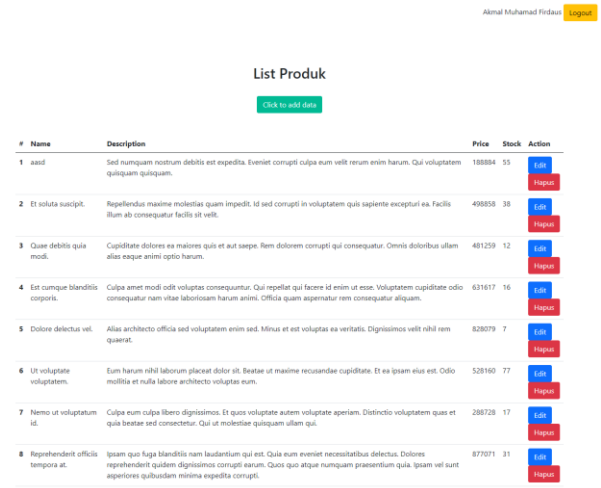
Untuk uji coba, berdasarkan data user diatas adalah : Email : **akmalmf007@gmail.com** password : **password**



Tampilan jika login gagal



Tampilan setelah login berhasil



11. Model Relasi

Model Eloquent menyediakan fasilitas agar dua Model yang berelasi dapat langsung memanggil satu sama lain. Kita akan mencoba salah satunya yaitu relasi one to many, dengan menggunakan contoh kasus relasi Product dengan Variant. Tiap Product dapat memiliki banyak Variant, tetapi tiap Variant hanya dimiliki oleh satu Product. Langkah awal yaitu membuat Model dan file migration-nya, maka perintahnya:

```
C:\xampp\htdocs\ABP\Modul 7>php artisan make:model Variant -m

  INFO  Model [C:\xampp\htdocs\ABP\Modul 7\app/Models/Variant.php] created successfully.

  INFO  Migration [C:\xampp\htdocs\ABP\Modul 7\database\migrations/2023_04_02_091615_create_variants_table.php] created
successfully.
```

Setelah itu edit file xxx_create_variants_table.php pada folder database/migrations untuk mendefinisikan kolomnya:

```php
return new class extends Migration
{
    public function up()
    {
        Schema::create('variants', function (Blueprint $table) {
            $table->id();
            $table->string('processor');
            $table->integer('memory');
            $table->integer('storage');
            $table->foreignId('product_id')->constrained();
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('variants');
    }
};
```

Untuk kebutuhan foreign key, kita menggunakan foreignId dan constrained jika tabel entitas yang diacu mengikuti konvensi Model Eloquent (PK bernama “id”, tabel database ditambah akhiran “s”, dsb). Jika berbeda, maka pendefinisian foreign key harus menggunakan foreign, references, dan on. Contoh:

```php
return new class extends Migration
{
    public function up()
    {
        Schema::create('variants', function (Blueprint $table) {
            $table->id();
            $table->string('processor');
            $table->integer('memory');
            $table->integer('storage');
            $table->bigInteger('product_id')->unsigned();
            $table->foreign('product_id')->references('id')->on('products');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('variants');
    }
};
```

Kemudian generate tabel variants dengan perintah seperti sebelumnya (php artisan migrate). Selanjutnya edit file Model Variants, tambahkan fungsi berikut:

```php
class Variant extends Model
{
    use HasFactory;

    0 references | 0 overrides
    public function product() {
        return $this->belongsTo(Product::class);
    }
}
```

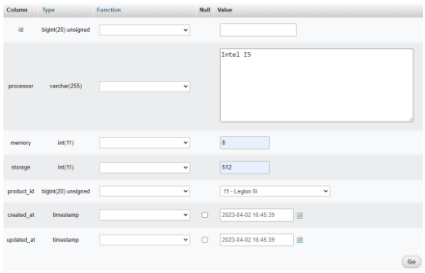
Fungsi belongsTo() secara otomatis akan memanggil object Product yang berelasi dengan dirinya berdasarkan product_id. Kemudian tambahkan juga fungsi ke dalam Model Product:

```php
class Products extends Model
{
    use HasFactory;
    0 references
    protected $fillable = [
        'name',
        'description',
        'stock',
        'price',
    ];
    0 references | 0 overrides
    public function variants() {
        return $this->hasMany(Variant::class, "id");
    }
}
```

Fungsi hasMany() secara otomatis akan memanggil semua object Variant yang berelasi dengan dirinya berdasarkan product_id. Untuk mencobanya kita akan menambahkan satu kolom pada tampilan tabel di index.blade.php untuk menampilkan data variant dari tiap product:

```
<tbody>
    @foreach($products as $product)
    <tr>
        <th scope="row">{{$product['id']}}</th>
        <td>{{$product['name']}}</td>
        <td>{{$product['description']}}</td>
        <td>{{$product['price']}}</td>
        <td>{{$product['stock']}}</td>
        <td>
            @foreach($product->variants()->get() as $variant)
                <li> Processor: {{$variant->processor}}</li>
                <li> Memory: {{$variant->memory}}</li>
                <li> Storage: {{$variant->storage}}</li>
            @endforeach
        </td>
        <td><a href="/products/{{ $product->id }}/edit" class="btn btn-primary">Edit</a>
            <form method="post" action="/products/{{ $product->id }}" onsubmit="return confirm('Yakin hapus?')">
                @csrf
                @method('DELETE')
                <button class="btn btn-danger">Hapus</button>
            </form>
        </td>
    </tr>
    @endforeach
</tbody>
```

Tambahkan data secara manual pada DBMS untuk melakukan pengecekan relasi yang kita buat sudah benar



11	Legion 5	Berkompetisi seperti profesional dengan laptop gaming Legion 5i Gen 6 (15" Intel). Dilengkapi dengan semua yang Anda butuhkan untuk mendominasi lawan dari prosesor Intel® Core™ Generasi ke-11 yang bertenagahingga grafis NVIDIA® GeForce RTX™ 30 Series yang	15000000	5	• Processor: Intel I5 • Memory: 8 • Storage: 512	Edit Hapus

Relasi tersebut sudah sesuai dan masuk sesuai kolom data yang diberikan.

Full Source code: https://github.com/codernewbie04/CII3H4-ABP-Praktikum