

BAB VI DATABASES

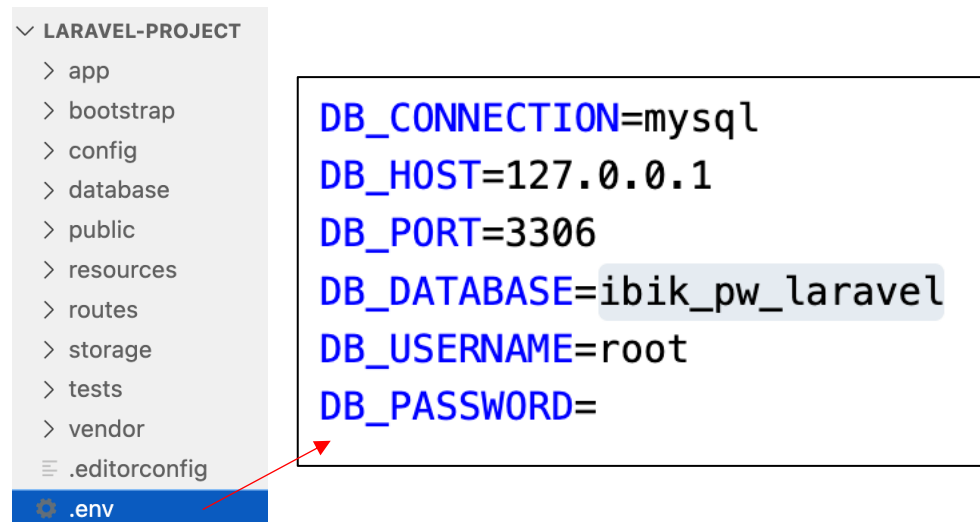
1. Migration

Migration seperti kontrol versi untuk database, memungkinkan untuk memodifikasi dan membagikan skema database aplikasi. Migrasi biasanya dipasangkan dengan pembuatan skema Laravel lainnya agar dapat membuat skema database aplikasi

Bagian depan Skema Laravel menyediakan dukungan agnostik database untuk membuat dan memanipulasi tabel di semua sistem database yang didukung Laravel.

1.1. Database configuration

Dalam membangun sebuah koneksi project Laravel dengan database terdapat hal-hal yang perlu diseting terlebih dahulu, pertama buatlah sebuah database bernama `ibik_pw_laravel`, bukalah file `.env` pada project Laravel dan ubahlah setingan environment database sesuai dengan koneksi MYSQL anda:



Gambar 1.1. Setting database file koneksi `.env`

1.2. Generate Migration

Untuk membuat sebuah table pada database anda dapat memanfaatkan Teknik migration yang dimiliki oleh Laravel. Dimana anda dapat membuat beberapa factory berdasarkan table-table database anda. Contoh berikut ini adalah akan membuat sebuah table bernama Products. Untuk membuat sebuah table Products hal yang perlu dilakukan ialah membuat file migration, bukalah project Laravel anda dengan terminal dan masukan syntax dibawah ini untuk membuat file migration table:

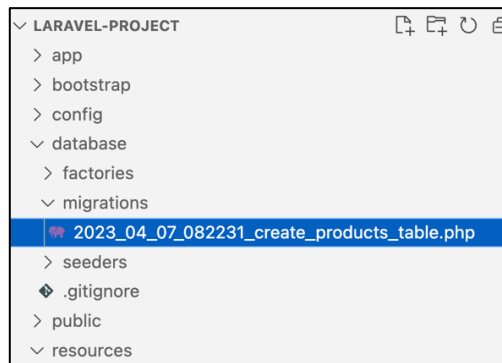
```
php artisan make:migration create_products_table
```

```
febryfairuz@Febrys-MacBook-Air laravel-project % php artisan make:migration create_products_table

[INFO] Migration [database/migrations/2023_04_07_082231_create_products_table.php] created successfully.

febryfairuz@Febrys-MacBook-Air laravel-project %
```

Setelah anda memasukan syntax migration tersebut maka akan menginformasikan letak directory path file migrations bernama *create_products_table*.

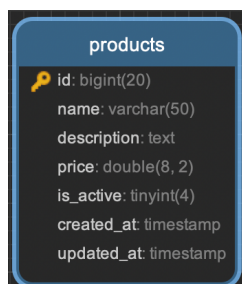


Gambar 1.2.1. Directory path file migrations

Bukalah file migration yang telah anda buat dan silakan anda buat field-field apa saja yang dibutuhkan, contohnya yaitu seperti dibawah ini:

```
public function up(): void
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name',50);
        $table->text('description');
        $table->float('price',8,2); //=> bernilai DOUBLE
        $table->tinyInteger('is_active'); //=> tiny int
        $table->timestamps();
    });
}
```

Jika melihat script diatas menandakan bahwa table product akan memiliki beberapa field seperti:



Field	Type	Value
id	bigint	20
name	varchar	50
description	text	
price	double	8,2
is_active	tinyint	4

A. Tipe Data

Beberapa Tipe data yang dapat digunakan untuk mendeklarasikan field pada table:

# bigIncrements()	# decimal()
# bigInteger()	# enum()
# boolean()	# float()
# char()	# id()
# dateTime()	# increments()
# date()	# integer()

B. Menjalankan Migration

Setelah menentukan file apa saja yang akan dibuat untuk mengeksekusi file migration anda dapat memasukkan syntax dibawah ini pada terminal project Laravel anda:

```
php artisan migrate
```

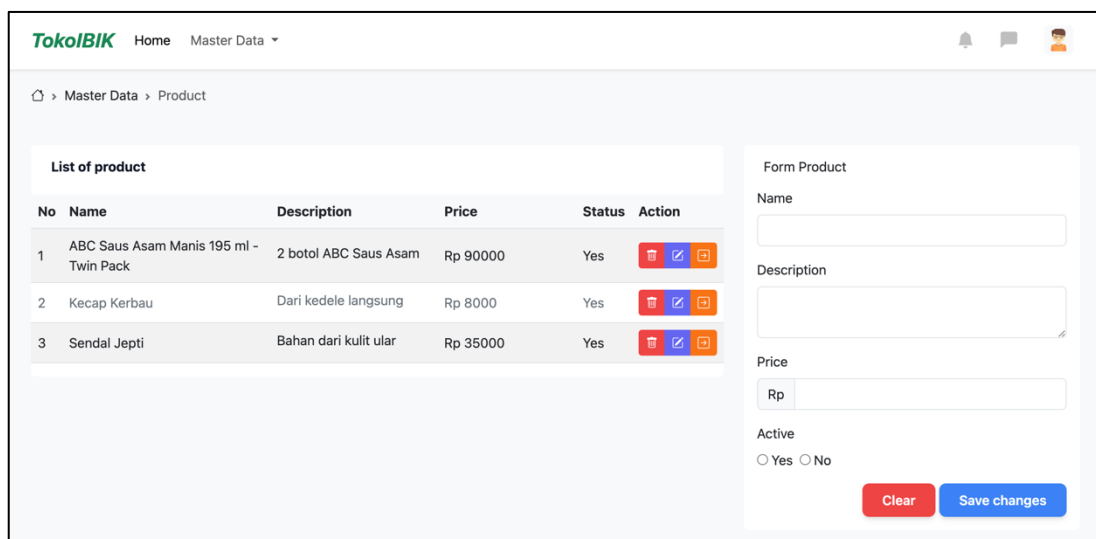
Namun jika ada perubahan pada table anda, setelah merubah file migrasinya lalu jalankan perintah ini:

```
php artisan migrate:refresh
```

Silakan anda cek kedalam database *ibik pw laravel* apakah table *products* yang telah anda buat dengan migration berhasil dibuat atau tidak.

2. Query Builder

Setelah berhasil membuat sebuah table *products*, contoh kasus kali ini akan membuat CRUD terhadap data-data yang akan dikelola oleh table *products*. Pada chapter sebelumnya, telah diterangkan bahwa konsep kerja dari PHP Frameworkd ialah MVC. Dimana file Model akan mengeksekusi logika database, file Controller akan mengeksekusi logika business process dan View pada blade akan menampilkan hasil yang telah dikelola di model dan controller dalam bentuk UI.



Gambar 2. Tampilan project Laravel dengan CRUD

Setelah membuat file migration buatlah file Controller dan Model untuk Product dengan menggunakan syntax php artisan:

```
php artisan make:controller ProductsController --resource --model=Products
```

Syntax php artisan diatas akan membuat sebuah file controller pada `./app/Http/controllers/` bernama *ProductsController* dan file model pada `./app/Http/models/` bernama *Products* yang berada pada directory path project Laravel.

Untuk mengimplementasikan Query Builder pada Laravel, terdapat dua buah cara yaitu dengan *Eloquent Factory Model* dan PDO parameters. Berikut ini akan memberikan contoh dari kedua Teknik query builder pada Laravel untuk melakukan CRUD.

1. Create dan Retrive

Pada section ini akan mencontohkan bagaimana melakukan insert dan retriive data kedalam sebuah database dengan konsep MVC dan Teknik query builder dengan *Eloquent Factory Model*.

Model: Products.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\DB;

class Products extends Model
{
    use HasFactory;

    protected $table = 'products';

    protected $fillable = [
        'id', 'name', 'description', 'price', 'is_active'
    ];

    public function storedData($data){
        $results = Products::create($data);
        return $results;
    }
}
```

Controller: ProductsController.php

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\UpdateProductsRequest;
use App\Models\Products;
use Illuminate\Http\Request;

class ProductsController extends Controller
{
    public function index()
    {
        $products = Products::latest()->paginate(5);
        return view('modules.master-data.products.index', compact('products'))
            ->with('i', (request()->input('page', 1) - 1) * 5);
    }
}
```

Pada sebuah fungsi bernama index pada *ProductsController* diatas, variable *\$products* menyimpan sebuah *Eloquent Factory* dari model products:

`Products::latest()->paginate(5);` → mengambil record pada database dari urutan 5 terbaru (pengambilan data secara descending).

Lalu nilai variable *\$products* dikirimkan ke file blade yang berada di `module.master-data.products.index`. Fungsi `paginate` merupakan bawaan eloquent factory dimana pada ui blade index sudah membentuk sebuah data dalam bentuk pagination dengan maximum nilai data sebanyak 5 buah.

Route: Web.php

```
<?php

use App\Http\Controllers\HomeController;
use App\Http\Controllers\ProductsController;
use Illuminate\Support\Facades\Route;

Route::get('/', [HomeController::class, 'index']);
Route::get('/home', [HomeController::class, 'index']);

Route::controller(ProductsController::class)->group(function () {
    Route::get('/master-data/products', 'index')->name('m_products');
    Route::post('/master-data/products', 'store');
    Route::get('/master-data/products/{id}', 'show')->name('m_products_detail');
    Route::get('/master-data/products/edit/{id}', 'edit')->name('m_products_edit');
    Route::get('/master-data/products/remove/{id}', 'destroy')->name('m_products_remove');
});
```

Blade View: index.blade.php

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

@section('content')
    <div class="row mt-5">
        <div class="col-lg-8 col-xxl-8">
            <div class="card border-0">
                <div class="card-header bg-white border-0 px-4 py-3">
                    <div class="card-title">
                        <h5 class="fw-bolder text-gray-900 m-0">List of product</h5>
                    </div>
                </div>
                <div class="card-body p-0">
                    <div class="table-responsive">
                        <table class="table table-striped table-hover align-middle">
                            <thead class="fs-6 fw-bolder bg-light">
                                <tr class="fs-7">
                                    <th>No</th>
                                    <th width="30%">Name</th>
                                    <th>Description</th>
                                    <th width="20%">Price</th>
                                    <th>Status</th>
                                    <th>Action</th>
                                </tr>
                            </thead>
                            <tbody class="fw-6 text-secondary">
                                <@if (count($products) > 0)>
                                    <@foreach ($products as $index => $product)>
                                        <tr>
                                            <td>{{ $index + 1 }}</td>
                                            <td>{{ $product->name }}</td>
                                            <td>
                                                <div style="height: 30px; overflow: hidden;">
                                                    {{ $product->description }}
                                                </div>
                                            </td>
                                            <td>Rp {{ $product->price }}</td>
                                            <td>{{ ($product->is_active) ? "Yes":"No" }}</td>
                                            <td>
                                                <div class="btn-group">
                                                    <button
                                                        class="btn btn-sm btn-icon bg-red-500 text-white hover:bg-red-700"
                                                        title="Remove" type="button">
                                                        <i class="bi bi-trash"></i>
                                                    </button>
                                                    <button
                                                        title="Edit"
                                                        class="btn btn-sm btn-icon bg-indigo-500 text-white hover:bg-indigo-700"
                                                        type="button">
                                                        <i class="bi bi-pencil-square"></i>
                                                    </button>
                                                    <a
                                                        title="Detail"
                                                        class="btn btn-sm btn-icon bg-orange-500 text-white hover:bg-orange-700"
                                                        type="button">
                                                        <i class="bi bi-arrow-right-square"></i>
                                                    </a>
                                                </div>
                                            </td>
                                        </tr>
                                    <@endforeach>
                                <@else>
                                    <tr>
                                        <td colspan="6">No record found</td>
                                    </tr>
                                <@endif>
                            </tbody>
                        </table>
                        <div class="text-center">
                            {{ $products->links() }}
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="col-lg-4 col-xxl-4">
            <@include('modules.master-data.products.create')>
        </div>
    </div>
</section>
```

- resources
 - css
 - js
 - sass
- views
 - auth
- modules
 - home
- master-data
 - products
 - create.blade.php
 - index.blade.php
 - show.blade.php

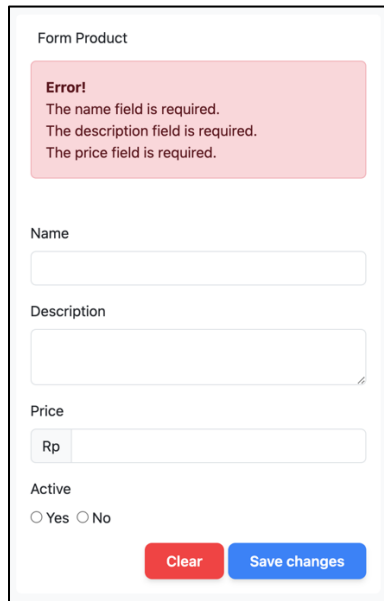
Paginate bawaan Laravel
10 dengan menggunakan
CSS TailWind

Blade View: create.blade.php

```
<div class="card border-0">
  <div class="card-header bg-white border-0 px-4 py-3">
    <h5>Form Product</h5>
  </div>
  <div class="card-body pt-0">
    @if ($errors->any())
      <div class="alert alert-danger mb-5">
        <strong>Error!</strong> <br>
        <ul>
          @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
          @endforeach
        </ul>
      </div>
    @endif
    @if ($message = Session::get('success'))
      <div class="alert alert-success mb-5">
        <p>{{ $message }}</p>
      </div>
    @endif

    <form action={{ url('/master-data/products') }} method="post" autocomplete="off" id="form-product">
      @csrf
      <div class="mb-3">
        <label class="form-label">Name</label>
        <input type="hidden" class="form-control id" name="id" />
        <input type="text" class="form-control name" name="name" />
        <div class="form-text text-danger"></div>
      </div>
      <div class="mb-3">
        <label class="form-label">Description</label>
        <textarea class="form-control description" name="description"></textarea>
        <div class="form-text text-danger"></div>
      </div>
      <div class="mb-3">
        <label class="form-label">Price</label>
        <div class="input-group flex-nowrap">
          <span class="input-group-text" id="addon-wrapping">Rp</span>
          <input type="text" class="form-control price" name="price" />
        </div>
        <div class="form-text text-danger"></div>
      </div>
      <div class="mb-3">
        <label class="form-label">Active</label>
        <div class="d-flex justify-content-start align-items-center">
          <label class="me-2">
            <input type="radio" class="is_active_Y" name="is_active" value="1" /> Yes
          </label>
          <label class="me-2">
            <input type="radio" class="is_active_N" name="is_active" value="0" /> No
          </label>
        </div>
        <div class="form-text text-danger"></div>
      </div>
      <div class="text-end">
        <button class="py-2 px-4 btn-danger" type="reset">Clear</button>
        <button class="py-2 px-4 btn-primary" type="submit">Save changes</button>
      </div>
    </form>
  </div>
</div>
```

Output dari script diatas akan menghasilkan program seperti dibawah ini dengan beberapa kondisi, seperti validasi dan informasi data berhasil atau gagal dieksekusi untuk melakukan inserting ataupun menampilkan data pada database.



Form Product

Error!
The name field is required.
The description field is required.
The price field is required.

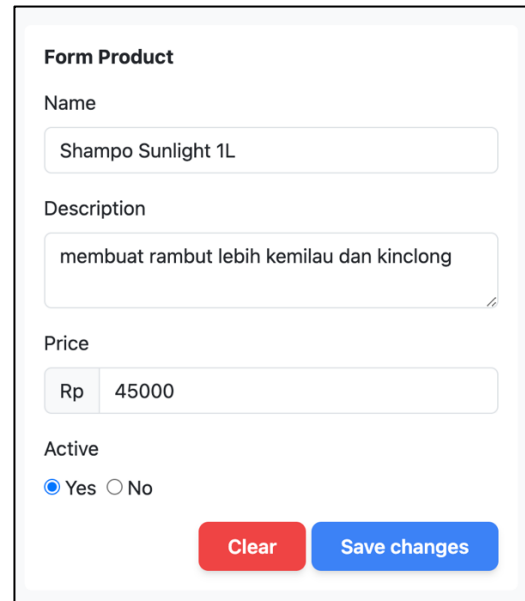
Name

Description

Price
Rp

Active
☐ Yes ☐ No

Gambar 2.1.1. Jika salah satu form isian field adalah kosong



Form Product

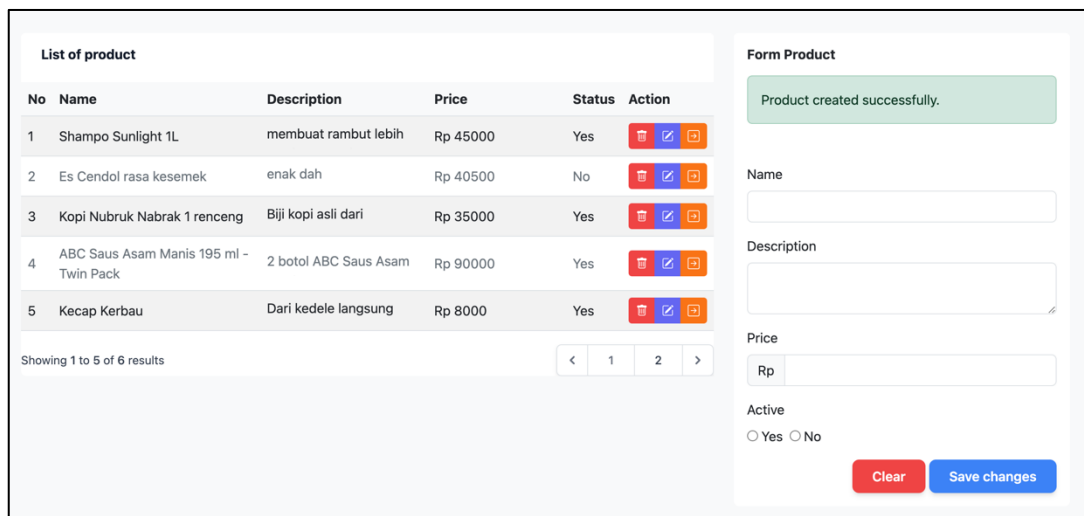
Name

Description

Price
Rp

Active
☒ Yes ☐ No

Gambar 2.1.2. Tampilan mengisi sebuah data baru



List of product

No	Name	Description	Price	Status	Action
1	Shampo Sunlight 1L	membuat rambut lebih	Rp 45000	Yes	<input type="button" value="Delete"/> <input type="button" value="Edit"/> <input type="button" value="Add"/>
2	Es Cendol rasa kesemek	enak dah	Rp 40500	No	<input type="button" value="Delete"/> <input type="button" value="Edit"/> <input type="button" value="Add"/>
3	Kopi Nubruk Nabrak 1 renceng	Biji kopi asli dari	Rp 35000	Yes	<input type="button" value="Delete"/> <input type="button" value="Edit"/> <input type="button" value="Add"/>
4	ABC Saus Asam Manis 195 ml - Twin Pack	2 botol ABC Saus Asam	Rp 90000	Yes	<input type="button" value="Delete"/> <input type="button" value="Edit"/> <input type="button" value="Add"/>
5	Kecap Kerbau	Dari kedele langsung	Rp 8000	Yes	<input type="button" value="Delete"/> <input type="button" value="Edit"/> <input type="button" value="Add"/>

Showing 1 to 5 of 6 results

Form Product

Product created successfully.

Name

Description

Price
Rp

Active
☐ Yes ☐ No

Gambar 2.1.3. Tampilan informasi berhasil menambahkan data baru

2. Retrive 1 Data

Pada section ini akan menampilkan data dalam jumlah 1 atau mencari data berdasarkan kondisi. Disini akan mencontohkan penggunaan query builder dalam bentuk PDO parameters. Dimana untuk mengakses sebuah table harus didefinisikan dengan menggunakan `DB::table('table_name')`.

Model: Products.php

```
public function getByCondition($condition){
    $results = DB::table($this->table)->where($condition);
    return $results;
}
```

Controller: ProductsController.php

```
public function show(Request $request)
{
    $products = new Products();
    $data['product'] = $products->getByCondition(array('id'=>$request->id))->first();

    return view('modules.master-data.products.show', $data);
}
```

Blade View: show.blade.php

```
@extends('templates.layouts')
@section('content')
    <div class="row">
        <div class="col-lg-6 col-xl-6">
            <div class="info">
                <p class="text-4xl mb-3">{{ $product->name }}</p>
                <p class="text-4xl mb-2">Rp {{ $product->price }}</p>
                <div class="border-top border-bottom p-2 mt-3">
                    <span class="text-1xl">Description</span>
                </div>
                <div class="description p-2">{{ $product->description }}</div>
            </div>
        </div>
        <div class="col-lg-2 col-xl-2">
            <div class="border rounded bg-white p-3">
                <div class="mb-3">
                    <p class="text-dark text-sm mb-2">Created at <br/>{{ $product->created_at }}</p>
                    <p class="text-dark text-sm">Last Updated at <br/>{{ $product->updated_at }}</p>
                </div>
                <div class="d-grid gap-2">
                    <a href="" class="btn btn-sm btn-warning">Edit</a>
                    <a href="" class="btn btn-sm btn-danger">Remove</a>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Jika melihat gambar 2.1.3 pada table of product terdapat 3 buah button di kolom Action, jika mengklik tombol ke tiga (button berwarna orange) yaitu detail maka akan dialihkan ke halaman dengan route end-point:

<http://localhost:8000/master-data/products/3>

jika melihat file route pada *Web.php* tipe link diatas akan diarahkan ke *ProductsController* dengan fungsi bernama *show()*.



Gambar 2.2. Tampilan halaman detail

3. Update dan Delete

Pada contoh section kali ini akan mengkombinasikan blade view dengan bentuk render data dengan JAVASCRIPT untuk melakukan UPDATE dan DELETE.

Model: Products.php

```
public function updatedData($data){
    $isExist = $this->getByCondition(array('id'=>$data['id']))->first();
    if(!empty($isExist)){
        unset($data['_token']);
        $results = DB::table($this->table)->where(array('id'=>$data['id']))->update($data);
        return $results;
    }else{
        return null;
    }
}

public function removeByCondition($condition){
    $results = Products::where($condition)->delete();
    return $results;
}
```

Query Builder PDO

Query Builder Eloquent Factory

Controller: ProductsController.php

```
public function store(Request $request)
{
    $products = new Products();
    if(!empty($request->id)){
        $request->validate([
            'id' => 'required',
            'name' => 'required',
            'description' => 'required',
            'price'=> 'required'
        ]);

        $results = $products->updatedData($request->all());
        return redirect()->route('m_products')->with('success',
            ($results) ? 'Product saved.' : 'Product failed save.');
```

```
    }else{
        $request->validate([
            'name' => 'required',
            'description' => 'required',
            'price'=> 'required'
        ]);
        $results = $products->storedData($request->all());
        return redirect()->route('m_products')->with('success',
            ($results)? 'Product created successfully.' : 'Product failed save.');
```

```
    }
}
```

Pada code sebelumnya di fase create, function store hanya menyimpan logika transaksi insert query saja. Namun kali ini dilakukan perubahan, jika data yang dilempar memiliki key bernama *id*, maka akan mengeksekusi logika transaksi update query sedangkan sebaliknya maka akan mengeksekusi logika transaksi insert query.

Pengecekan ini terjadi dikarenakan jika ingin melakukan update query data, memerlukan sebuah kondisi statement, pada contoh update disini akan mempergunakan key *id* sebagai salah satu update kondisi statement.

Blade View: index.blade.php

```
@extends('templates.layouts')
@section('content')
...
<div class="btn-group">
    <button
        class="btn btn-sm btn-icon bg-red-500 text-white hover:bg-red-700"
        title="Remove" type="button"
        onclick="RemoveItem({{ $product->id }})">
        <i class="bi bi-trash"></i>
    </button>
    <button title="Edit"
        class="btn btn-sm btn-icon bg-indigo-500 text-white hover:bg-indigo-700"
        type="button" onclick="EditItem({{ $product->id }})">
        <i class="bi bi-pencil-square"></i>
    </button>
    <a href="{{ route('m_products_detail', $product->id) }}" title="Detail"
        class="btn btn-sm btn-icon bg-orange-500 text-white hover:bg-orange-700"
        type="button">
        <i class="bi bi-arrow-right-square"></i>
    </a>
</div>
...
<script>
    const RemoveItem = (id) => {
        if (confirm("Are you sure wants to remove this item ?")) {
            const xmlhttp = new XMLHttpRequest();
            xmlhttp.onload = function() {
                var data = JSON.parse(this.response);
                alert(data.message);
                window.location.href = "{{ route('m_products') }}";
            }
            xmlhttp.open("GET", "{{ url('') }}/master-data/products/remove/" + id);
            xmlhttp.send();
        }
    }

    const EditItem = (id) => {
        var targetDiv = document.getElementById("form-product");
        let id_ = targetDiv.getElementsByClassName("id")[0];
        let name = targetDiv.getElementsByClassName("name")[0];
        let desc = targetDiv.getElementsByClassName("description")[0];
        let price = targetDiv.getElementsByClassName("price")[0];
        let is_active_Y = targetDiv.getElementsByClassName("is_active_Y")[0];
        let is_active_N = targetDiv.getElementsByClassName("is_active_N")[0];

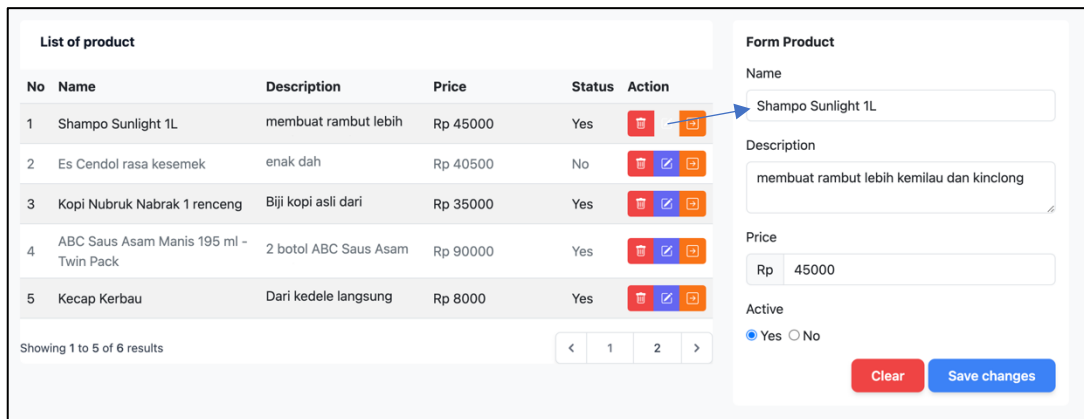
        const xmlhttp = new XMLHttpRequest();
        xmlhttp.onload = function() {
            var data = JSON.parse(this.response);
            id_.value = data.id;
            name.value = data.name;
            desc.value = data.description;
            price.value = data.price;
            if(data.is_active === 1){
                is_active_Y.checked = true;
            }else{
                is_active_N.checked = true;
            }
        }
        xmlhttp.open("GET", "{{ url('') }}/master-data/products/edit/" + id);
        xmlhttp.send();
    }
</script>
</endsection>
```











Hanya menambahkan event click

Menambahkan route link pada tombol

Menambahkan function action dari event click

Maka output dari proses transaksi logical diatas untuk UPDATE dan DELETE sebagai berikut:



No	Name	Description	Price	Status	Action
1	Shampo Sunlight 1L	membuat rambut lebih	Rp 45000	Yes	 
2	Es Cendol rasa kesemek	enak dah	Rp 40500	No	 
3	Kopi Nubruk Nabrak 1 renceng	Biji kopi asli dari	Rp 35000	Yes	 
4	ABC Saus Asam Manis 195 ml - Twin Pack	2 botol ABC Saus Asam	Rp 90000	Yes	 
5	Kecap Kerbau	Dari kedele langsung	Rp 8000	Yes	 

Showing 1 to 5 of 6 results

Form Product

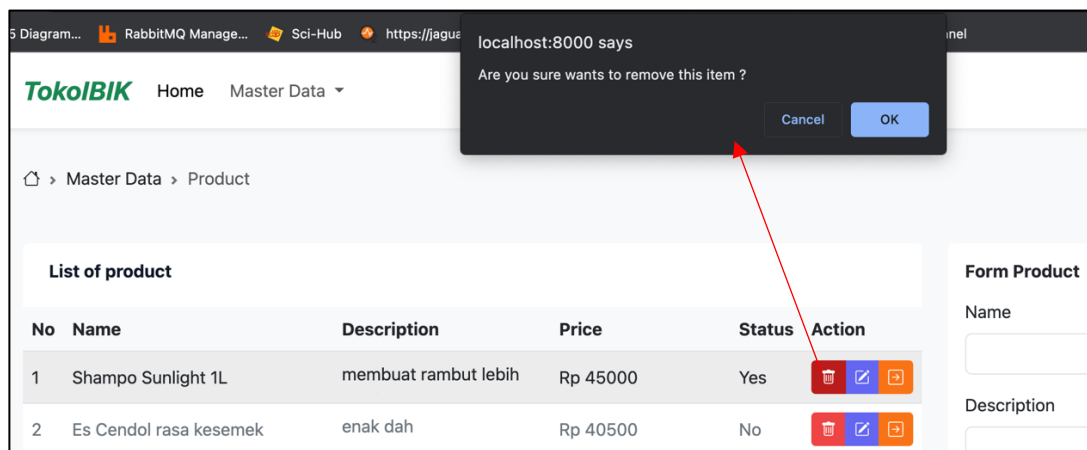
Name:

Description:

Price:

Active: ☒ Yes ☐ No





Gambar 2.3. Tampilan update salah satu data.



TokoIBIK Home Master Data

Are you sure wants to remove this item ?

List of product

No	Name	Description	Price	Status	Action
1	Shampo Sunlight 1L	membuat rambut lebih	Rp 45000	Yes	 
2	Es Cendol rasa kesemek	enak dah	Rp 40500	No	 

Form Product

Name:

Description:

Gambar 2.4. Tampilan delete salah satu data.

3. Latihan Praktikum

1. Buatlah package baru bernama praktikum-6
2. Buatlah sebuah CRUD untuk data **Users**, dimana pada tablenya memiliki structure dibawah ini:

Field	Tipe	Value
id*	bigint	20
email	varchar	20
fullname	varchar	100
address	text	8,2
birthdate	date	4
gender	enum	('M','F')
phone	varchar	14

3. Berdasarkan soal nomor 2, buatlah skema table **Users** dengan menggunakan MIGRATION.



IBIK

Matakuliah/Code

Dosen

Kelas

: Lab. Pemrograman Web / TIFA2Q0

: Irvan Rizky Ariansyah / Thesya Marcella

: TI-21-PA

4. Setelah berhasil membuat table dengan soal nomor 3, buatlah Tampilan untuk melakukan maintenance data Users dalam bentuk CRUD dengan memanfaatkan kerangka kerja MVC.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PW-TI-21-[PA/KA]-NPM. File source code disimpan sesuai nama package-praktikum dan masukan kedalam repositori tersebut. Buatlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository anda kepada:

@FebryFairuz dan (@IrvanRizkyAriansyah atau @thesyamarcella)