

# CRUD pada Aplikasi Web dengan Laravel 5.5

[Husni@Trunojoyo.ac.id](mailto:Husni@Trunojoyo.ac.id)

Ini merupakan tutorial **Laravel 5.5** bagi pengguna awal Laravel. Laravel baru saja meluncurkan versi barunya bernama **Laravel 5.5**, dan hadir dengan banyak fitur baru. Kali ini kita akan fokus pada pengembangan aplikasi web dengan fasilitas CRUD di dalam Laravel 5.5. Hanya tutorial membuat sebuah aplikasi kecil. **Laravel** adalah framework PHP yang elegan dan sangat fleksibel, sejauh ini, dan selalu hadir dengan fitur baru. Laravel merupakan perwakilan terbaik dari Bahasa **PHP Language**, menurut saya. Ini merupakan framework yang sangat lengkap fungsinya. Pembuatnya telah dengan serius membuat dan memeliharanya. Penggunaannya semakin banyak dan selalu nomor satu dalam 3 tahun terakhir.

## Daftar Isi:

- [Fitur baru Laravel 5.5.](#)
- [Kebutuhan Instalasi](#)
- [Contoh Praktis pembuatan aplikasi Web](#)
  - [Langkah 1: Instalasi Framework Laravel 5.5.](#)
  - [Langkah 2: Pengaturan koneksi ke database MySQL](#)
  - [Langkah 3: Pembuatan file Model & Migrasi](#)
  - [Langkah 4: Pembuatan File View \(form create\)](#)
  - [Langkah 5: Pembuatan Controller & Route](#)
  - [Langkah 6: Validasi Form di Laravel 5.5](#)
  - [Langkah 7: Pembuatan halaman Index](#)
  - [Langkah 8: Pembuatan View Edit dan Penanganan Update](#)
  - [Langkah 9: Penghapusan Produk](#)

## Fitur Baru Laravel 5.5

1. Custom validation rules object.
2. Returns response data from the validator.
3. Improvements with the default error views.
4. Great support for the custom error reporting.
5. Support for email themes in mailable.
6. We can render mailable in the browser.
7. Vendor packages have provider support.
8. It adds front end presets, which means we can use Vue, React.js or none of them if we want.
9. Laravel Migrate Fresh command.
10. Whoops, which was there in Laravel 4.2 is back in Laravel 5.5
11. Laravel Package Auto Discovery.

## Kebutuhan Instalasi

- PHP >= 7.0.0
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

**Catatan:** Laravel 5.5 mensyaratkan **PHP 7** atau versi di atasnya. Jika anda menggunakan **PHP versi lama seperti 5.4, 5.5, 5.6**, maka Laravel 5.5 tidak akan bekerja. Silakan upgrade sistem anda ke **PHP 7**

## Contoh Praktis pembuatan aplikasi Web

### Langkah 1: Instalasi Framework Laravel 5.5

```
composer create-project --prefer-dist laravel/laravel rai2017
```

Perintah ini akan membuat suatu direktori bernama rai2017 di bawah root directory web server (c:\xampp\htdocs) dan menginstall semua file yang diperlukan Laravel ke dalam folder tersebut.

Catatan: Di sini kita tidak akan secara rinci mengenai kebutuhan front-end seperti Vue.js atau React.js menggunakan Laravel Mix. Kita tidak akan menginstall paket Node.js saat ini.

### Langkah 2: Pengaturan Koneksi Database MySQL.

Konfigurasi koneksi ke database MySQL dilakukan di dalam file .env yang terletak di root directory dari aplikasi Laravel yang sedang di bangun, dalam contoh kita adalah rai2017 (lengkapannya c:\xampp\htdocs\rai2017).

```
// .env
```

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_rai2017
DB_USERNAME=root
DB_PASSWORD=
```

Selanjutnya adalah membuat database bernama db\_rai2017. Ini dapat dilakukan memanfaatkan tool seperti PHPMyAdmin atau MySQL Workbench. Kita hanya perlu membuat database, tabel-tabel yang di dalam database akan dibuat oleh Laravel.

Laravel 5.5, seperti juga versi sebelumnya, mempunyai perintah php artisan migrate. Perintah ini digunakan untuk membuat file PHP yang saat dieksekusi akan membuatkan tabel-tabel di dalam database yang Laravel tersambungkan. Sekarang, jalankan perintah ini di command line (pastikan anda berada di c:\xampp\htdocs\rai2017):

## php artisan migrate

Perintah ini akan membuatkan dua tabel di dalam database db\_rai2017, yaitu:

1. **users**
2. **password\_resets**

Table	Action	Rows	Type	Collation	Size	Overhead
migrations	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	16 KiB	-
password_resets	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
users	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
3 tables	Sum	2	InnoDB	latin1_swedish_ci	48 KiB	0 B

Apakah perintah migrate di atas menghasilkan error seperti di bawah ini?

```
c:\xampp\htdocs\rai2017>php artisan migrate
```

```
[Illuminate\Database\QueryException]
```

```
SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 767 bytes (SQ
```

```
L: alter table `users` add unique `users_email_unique`(`email`))
```

```
[PDOException]
```

```
SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 767 bytes
```

Bagaimana cara mengatasinya? Cara yang sudah berhasil sesuai panduan dari situs web Laravel adalah dengan mengedit file **AppServiceProvider.php** (di komputer saya di dalam C:\xampp\htdocs\rai2017\app\Providers ) dan di dalam metode boot atur panjang dari string default, seperti di bawah ini (ada 2 baris yang ditambahkan):

```
use Illuminate\Support\Facades\Schema;
```

```
public function boot() {  
    Schema::defaultStringLength(191);  
}
```

### Langkah 3: Pembuatan File Model dan Migrasi untuk Tabel Produk

Pembuatan model dilakukan memanfaatkan perintah php artisan make:model. Tuliskan perintah ini di Command line:

```
php artisan make:model Product -m
```

```
c:\xampp\htdocs\rai2017>php artisan make:model Product -m
```

```
Model created successfully.
```

```
Created Migration: 2017_10_08_134825_create_products_table
```

Perintah tersebut akan menghasilkan dua file berikut:

1. File model bernama: **Product.php**.
2. File migrasi bernama: **create\_products\_table**.

Kita perlu membuat skema untuk tabel products. Laravel telah menyiapkan skema minimal, kita dapat melengkapinya dengan informasi field-file yang harus ada di dalam tabel products. Sekarang, bukan file **create\_products\_table** yang terdapat di dalam folder **Rai2017** >> **database** >> **migrations**.

```
// create_products_table

public function up() {
    Schema::create('products', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->integer('price');
        $table->timestamps();
    });
}
```

Sehingga, tabel **products** sekarang mempunyai **5 field (kolom)**. Kolom **timestamps()** mempunyai 2 field, yaitu:

1. **created\_at**
2. **updated\_at**

Selanjutnya kita harus memigrasikan tabel tersebut dengan perintah berikut:

```
php artisan migrate
```

```
c:\xampp\htdocs\rai2017>php artisan migrate
```

```
Migrating: 2017_10_08_134825_create_products_table
```

```
Migrated: 2017_10_08_134825_create_products_table
```

Dalam database db\_rai2017 telah hadir tabel baru: **products**.

Table	Action	Rows	Type	Collation	Size	Overhead
migrations	Browse  Structure  Search  Insert  Empty  Drop	4	InnoDB	utf8mb4_unicode_ci	16 KiB	-
password_resets	Browse  Structure  Search  Insert  Empty  Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
products	Browse  Structure  Search  Insert  Empty  Drop	3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
users	Browse  Structure  Search  Insert  Empty  Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
4 tables	Sum	7	InnoDB	latin1_swedish_ci	64 KiB	0 B

#### Langkah 4: Pembuatan File View dan Form Penambahan Produk

Aplikasi web ini akan dapat digunakan untuk mengelola produk, mencakup penambahan, pengambilan informasi, perubahan dan penghapusannya. Kita dapat membuat folder bernama **v1** (anggap ini aplikasi versi pertama) dan buat file bernama create.blade.php di

alam folder **resources >> views >> v1**. File (form) ini akan digunakan untuk menerima input dari pengguna. Berikut ini kode HTMLnya:

```
<!-- create.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>...: RAI 2017 - Penerapan CRUD pada Laravel 5.5 :...</title>
    <link rel="stylesheet" href="{{asset('css/app.css')}}">
  </head>
  <body>
    <div class="container">
      <h2>Penambahan Produk</h2><br/>
      <form method="post">

        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="name">Nama:</label>
            <input type="text" class="form-control" name="name">
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="price">Harga:</label>
            <input type="text" class="form-control" name="price">
          </div>
        </div>
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <button type="submit" class="btn btn-success" style="margin-left:38px">Tambahkan Produk</button>
          </div>
        </div>

      </form>
    </div>
  </body>
</html>
```

### Langkah 5: Pembuatan Controller dan Route untuk Menghadirkan Form Create

Pada terminal tulislah perintah berikut:

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

```
c:\xampp\htdocs\rai2017>php artisan make:controller ProductController --resource
```

```
Controller created successfully
```

Perintah di atas akan membuat suatu file controller bernama **ProductController.php** dan di dalamnya sudah otomatis terdapat fungsi-fungsi CRUD yang dapat dilengkapi nantinya.

Perintah `make:controller` tersebut disertai parameter **resource**. Ini secara default akan memberikan kita pola-pola routing sesuai standard aplikasi CRUD modern. Rute sendiri kita tetapkan di dalam `web.php` (di dalam folder **routes**). Masukkan baris berikut ke dalam file `web.php`:

```
// web.php
```

```
Route::resource('v1', 'ProductController');
```

Rute atau URI apa saja yang sebenarnya ada di aplikasi CRUD kita? Beralihlah ke Terminal dan tuliskan perintah berikut untuk mengetahuinya:

```
php artisan route:list
```

Kita mendapatkan daftar seperti:

```
c:\xampp\htdocs\rai2017>php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	v1	v1.index	App\Http\Controllers\ProductController@index	web
	POST	v1	v1.store	App\Http\Controllers\ProductController@store	web
	GET HEAD	v1/create	v1.create	App\Http\Controllers\ProductController@create	web
	GET HEAD	v1/{v1}	v1.show	App\Http\Controllers\ProductController@show	web
	PUT PATCH	v1/{v1}	v1.update	App\Http\Controllers\ProductController@update	web
	DELETE	v1/{v1}	v1.destroy	App\Http\Controllers\ProductController@destroy	web
	GET HEAD	v1/{v1}/edit	v1.edit	App\Http\Controllers\ProductController@edit	web

Kita harus menulis sedikit kode di dalam file **ProductController.php**. Buka file ini dan tambahkan kode berikut ke dalam fungsi **create()**:

```
// ProductController.php
```

```
/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create() {
    return view('v1.create');
}
```

Sekarang saatnya melihat hasil pekerjaan kita. Jalankan server development:

```
php artisan serve
```

Buka web browser dan arahkan ke <http://localhost:8000/v1/create> atau jika menjalankan Apache (di bawah XAMPP), akses pada URL: <http://rai2017.dev/rai2017/public/v1/create>. Tampilan yang diperoleh adalah

## Langkah 6: Validasi Terhadap Form

Pertama, kita harus menerapkan suatu action terhadap form create produk sebelumnya.

```
<!-- create.blade.php -->
```

```
<form method="post" action="{{url('products')}}">
```

Selanjutnya adalah penanganan persoalan **CSRF**. Kita cukup meletakkan kode berikut di dalam form:

```
<!-- create.blade.php -->
```

```
{{csrf_field()}}
```

Kita juga perlu menangani **Mass Assignment Exception**. Silakan bukan file **Product.php** (di dalam folder App) dan masukkan properti **protected \$fillable** berikut ke dalamnya:

```
// Product.php
```

```
protected $fillable = ['name', 'price'];
```

Isi lengkap file Product.php menjadi:

```
<?php
```

```
namespace App;
```

```
use Illuminate\Database\Eloquent\Model;
```

```
class Product extends Model {
    protected $fillable = ['name', 'price'];
}
```

Jika kita melihat rute sumber daya (resource routes) maka di sana ada **post request** yang mempunyai rute **/v1** dan fungsi **store** dalam file **ProductController.php**. Sehingga kita perlu kode untuk fungsi store dalam rangka menyimpan data ke dalam database.

Perlu tetap diperhatikan bahwa kita perlu untuk memasukkan **namespace** dari model **Product.php** ke dalam file **ProductController.php**. Jadi, ketik baris berikut pada awal dari file **ProductController.php**:

```
use App\Product;
```

Juga, kita harus menempatkan validasi di sana:

```
// ProductController.php

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)    {
    $product = $this->validate(request(), [
        'name' => 'required',
        'price' => 'required|numeric'
    ]);

    Product::create($product);

    return back()->with('success', 'Product has been added');;
}
```

Sekarang, jika validasi gagal maka kita perlu menampilkan suatu pesan error. Silakan kembali buka file **create.blade.php** dan letakkan kode berikut setelah tag **h2**:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div><br />
@endif
@if (\Session::has('success'))
    <div class="alert alert-success">
        <p>{{ \Session::get('success') }}</p>
    </div><br />
@endif
```

Sehingga isi file **create.blade.php** menjadi:

```
<!-- create.blade.php -->

<!DOCTYPE html>
<html>
    <head>
```



```

<meta charset="utf-8">
<title>...: RAI 2017 - Penerapan CRUD pada Laravel 5.5 :...</title>
<link rel="stylesheet" href="{{asset('css/app.css')}}">
</head>
<body>
    <div class="container">
        <h2>Penambahan Produk</h2><br/>

        @if ($errors->any())
            <div class="alert alert-danger">
                <ul>
                    @foreach ($errors->all() as $error)
                        <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div><br />
        @endif
        @if (\Session::has('success'))
            <div class="alert alert-success">
                <p>{{ \Session::get('success') }}</p>
            </div><br />
        @endif

        <form method="post" action="{{url('v1')}}">
            {{csrf_field()}}
            <div class="row">
                <div class="col-md-4"></div>
                <div class="form-group col-md-4">
                    <label for="name">Nama:</label>
                    <input type="text" class="form-control" name="name">
                </div>
            </div>
            <div class="row">
                <div class="col-md-4"></div>
                <div class="form-group col-md-4">
                    <label for="price">Harga:</label>
                    <input type="text" class="form-control" name="price">
                </div>
            </div>
            <div class="row">
                <div class="col-md-4"></div>
                <div class="form-group col-md-4">
                    <button type="submit" class="btn btn-success" style="margin-left:38px">Tambahkan Produk</button>
                </div>
            </div>

        </form>
    </div>
</body>
</html>

```

Sekarang, silakan buka kembali URL localhost/rai2017/public/v1/create. Masukkan nama produk dan harganya. Kemudian klik Tambahkan Produk. Berikut ini adalah beberapa capture saat form tersebut disubmit:

**Nama dan Harga diisi dengan benar**

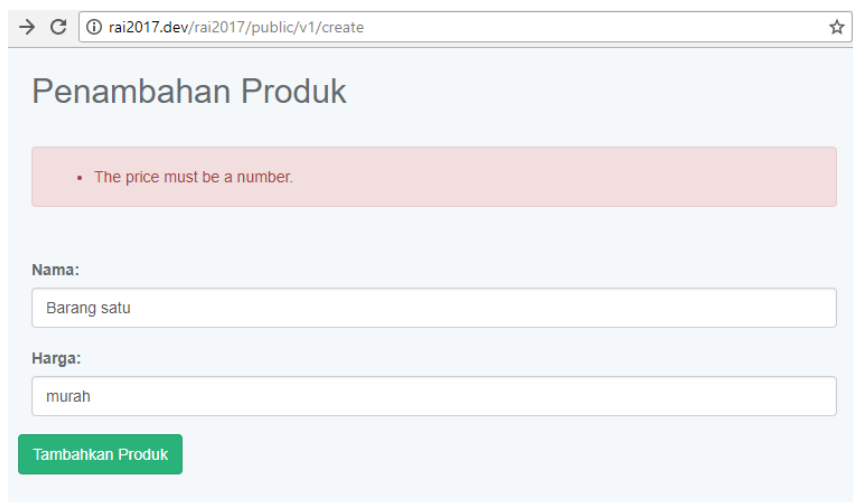
The first screenshot shows the 'Penambahan Produk' form with a green success message 'Product has been added'. The 'Nama:' field contains 'Barang satu' and the 'Harga:' field contains '100000'. A green 'Tambahkan Produk' button is visible at the bottom.

The second screenshot is identical to the first, showing the same form and success message.

**Nama dan Harga dikosongkan (padahal harus diisi):**

The screenshot shows the 'Penambahan Produk' form with a red error message box containing two bullet points: 'The name field is required.' and 'The price field is required.'

Nama diisi dengan benar, tetapi Harga salah isinya (harusnya angka, numerik):



→ [rai2017.dev/rai2017/public/v1/create](#)

### Penambahan Produk

- The price must be a number.

Nama:

Harga:

Tambahkan Produk

Jika kita mengisi semua nilai (dengan benar) maka akan diredirect ke halaman dengan pesan “sukses”, seperti dijelaskan oleh kode berikut:

// ProductController.php

```
public function store(Request $request) {
    $product = $this->validate(request(), [
        'name' => 'required',
        'price' => 'required|numeric'
    ]);
    Product::create($product);
    return back()->with('success', 'Product has been added');
}
```

Dalam Laravel 5.5 kita secara langsung memperoleh array nilai yang dikembalikan oleh fungsi validasi dan menggunakannya untuk menyisipkan ke dalam database, ini merupakan fitur baru di Laravel 5.5

### Langkah 7: Pembuatan Halaman Index

Bagaimana jika pengguna mengirimkan request tanpa parameter, hanya “v1/”? Aplikasi ini akan menampilkan semua data produk yang telah ada di dalam database, bersama dengan tombol “Ubah” dan “Hapus”. Pertama, kita perlu mengirimkan data itu ke **index.blade.php**. Jadi di dalam file **ProductController.php**, kita harus menuliskan kode untuk mengambil data dan menyerahkannya ke **view index**.

// ProductController.php

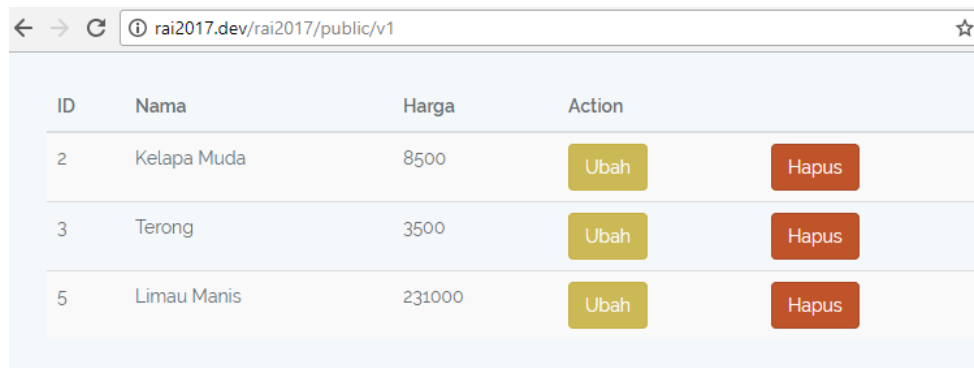
```
public function index() {
    $products = Product::all()->toArray();
    return view('v1.index', compact('products'));
}
```

Di dalam folder **resources >> views >> v1**, kita membuat file blade bernama **index.blade.php** dan isinya adalah sebagai berikut:

```
<!-- index.blade.php -->

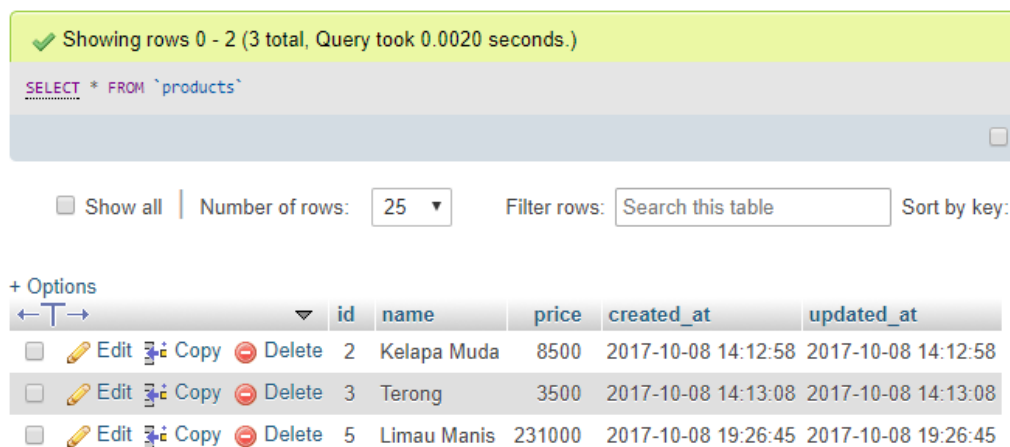
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>...: RAI 2017 - Penerapan CRUD pada Laravel 5.5 :...</title>
    <link rel="stylesheet" href="{{asset('css/app.css')}}">
  </head>
  <body>
    <div class="container">
      <br />
      @if (\Session::has('success'))
        <div class="alert alert-success">
          <p>{{ \Session::get('success') }}</p>
        </div><br />
      @endif
      <table class="table table-striped">
        <thead>
          <tr>
            <th>ID</th>
            <th>Nama</th>
            <th>Harga</th>
            <th colspan="2">Action</th>
          </tr>
        </thead>
        <tbody>
          @foreach($products as $product)
            <tr>
              <td>{{ $product['id'] }}</td>
              <td>{{ $product['name'] }}</td>
              <td>{{ $product['price'] }}</td>
              <td><a href="{{action('ProductController@edit', $product['id'])}}"
class="btn btn-warning">Ubah</a></td>
              <td>
                <form action="{{action('ProductController@destroy',
$product['id'])}}" method="post">
                  {{csrf_field()}}
                  <input name="_method" type="hidden" value="DELETE">
                  <button class="btn btn-danger" type="submit">Hapus</button>
                </form>
              </td>
            </tr>
          @endforeach
        </tbody>
      </table>
    </div>
  </body>
</html>
```

Bagaimana hasil dari perubahan pada 2 file di atas? Akses <http://rai2017.dev/rai2017/public/v1>:



ID	Nama	Harga	Action
2	Kelapa Muda	8500	<button>Ubah</button> <button>Hapus</button>
3	Terong	3500	<button>Ubah</button> <button>Hapus</button>
5	Limau Manis	231000	<button>Ubah</button> <button>Hapus</button>

Di dalam database, terdapat record-record berikut:



Showing rows 0 - 2 (3 total, Query took 0.0020 seconds.)

SELECT \* FROM `products`

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key:

+ Options

	id	name	price	created_at	updated_at
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	Kelapa Muda	8500	2017-10-08 14:12:58	2017-10-08 14:12:58
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	3	Terong	3500	2017-10-08 14:13:08	2017-10-08 14:13:08
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	5	Limau Manis	231000	2017-10-08 19:26:45	2017-10-08 19:26:45

## Langkah 8: Pembuatan View Edit dan Penanganan Update

Langkah selanjutnya adalah melakukan perubahan terhadap fungsi edit di dalam file **ProductController.php** sehingga dapat menampilkan form untuk perubahan informasi Produk. Berikut ini adalah kode untuk fungsi edit tersebut:

```
// ProductController.php

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id) {
    $product = Product::find($id);
    return view('v1.edit', compact('product', 'id'));
}
```

Terlihat jelas bahwa fungsi edit memanggil view bernama edit. Artinya kita harus membuat file **edit.blade.php** di dalam folder **resources >> views >> v1**.

```

<!-- edit.blade.php -->

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>...: RAI 2017 - Penerapan CRUD pada Laravel 5.5 :...</title>
    <link rel="stylesheet" href="{{asset('css/app.css')}}">
  </head>
  <body>
    <div class="container">
      <h2>Perubahan Produk</h2><br />
      @if ($errors->any())
        <div class="alert alert-danger">
          <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
          </ul>
        </div><br />
      @endif
      <form method="post" action="{{action('ProductController@update', $id)}}">
        {{csrf_field()}}
        <input name="_method" type="hidden" value="PATCH">
        <div class="row">
          <div class="col-md-4"></div>
          <div class="form-group col-md-4">
            <label for="name">Name:</label>
            <input type="text" class="form-control" name="name" value="{{ $product->name }}">
          </div>
          <div class="row">
            <div class="col-md-4"></div>
            <div class="form-group col-md-4">
              <label for="price">Price:</label>
              <input type="text" class="form-control" name="price" value="{{ $product->price }}">
            </div>
          </div>
          <div class="row">
            <div class="col-md-4"></div>
            <div class="form-group col-md-4">
              <button type="submit" class="btn btn-success" style="margin-left:38px">Update Produk</button>
            </div>
          </div>
        </form>
      </div>
    </body>
  </html>

```

Kemudian kita menentukan apa yang akan dilakukan saat form edit di atas diterima oleh ProductController. Ini ditangani oleh fungsi update.

```
// ProductController.php

public function update(Request $request, $id) {
    $product = Product::find($id);
    $this->validate(request(), [
        'name' => 'required',
        'price' => 'required|numeric'
    ]);

    $product->name = $request->get('name');
    $product->price = $request->get('price');
    $product->save();
    return redirect('v1')->with('success','Product has been updated');
}
```

### Langkah 9: Penghapusan Produk

Apa yang harusnya terjadi saat tombol “Hapus” diklik? Berikut ini kodenya:

```
// ProductController.php

public function destroy($id) {
    $product = Product::find($id);
    $product->delete();
    return redirect('v1')->with('success','Product has been deleted');
}
```

Demikianlah tutorial praktis membangun aplikasi web (dengan kemampuan CRUD) dasar menggunakan Framework PHP Laravel 5.5. Semoga bermanfaat.