

Perubahan dan Fitur baru di Laravel

Pengenalan Laravel 11

Laravel 11 melanjutkan perbaikan yang dilakukan di **Laravel 10.x** dengan memperkenalkan struktur aplikasi yang disederhanakan, *per-second rate limiting*, *health routing*, *graceful encryption key rotation*, *queue testing improvements*, [Resend](#) mail transport, *Prompt validator integration*, *new Artisan commands*, dan masih banyak lagi.

Selain itu, **Laravel** juga merilis sebuah *library* yang bernama **Laravel Reverb**. *Library* ini merupakan *server WebSocket* yang memberikan kemampuan *real-time* yang baik pada aplikasi yang kita buat.

PHP 8.2

Untuk **PHP** dengan versi `8.1` kini sudah tidak akan bisa digunakan lagi di **Laravel 11**. Untuk yang akan memulai membuat *project* baru di **Laravel 11**, maka harus menggunakan **PHP** versi `8.2` atau `8.3`.

Berikut ini informasi terkait dukungan **PHP 8.1** yang dihentikan di Laravel 11.

[\[11.x\] Drop PHP 8.1 support](#)

Kerangka Aplikasi yang lebih Sederhana

Laravel 11 akan memperkenalkan struktur *folder* yang lebih sederhana. Ketika membuat *project* baru di **Laravel 11**, *folder* `app` hanya akan berisi tiga *folder* utama, yaitu `Http`, `Models`, dan `Providers`, seperti yang terlihat pada ilustrasi berikut.

```
app
├── Http
│   └── Controllers
│       └── Controller.php
├── Models
│   └── User.php
└── Providers
    └── AppServiceProvider.php
```

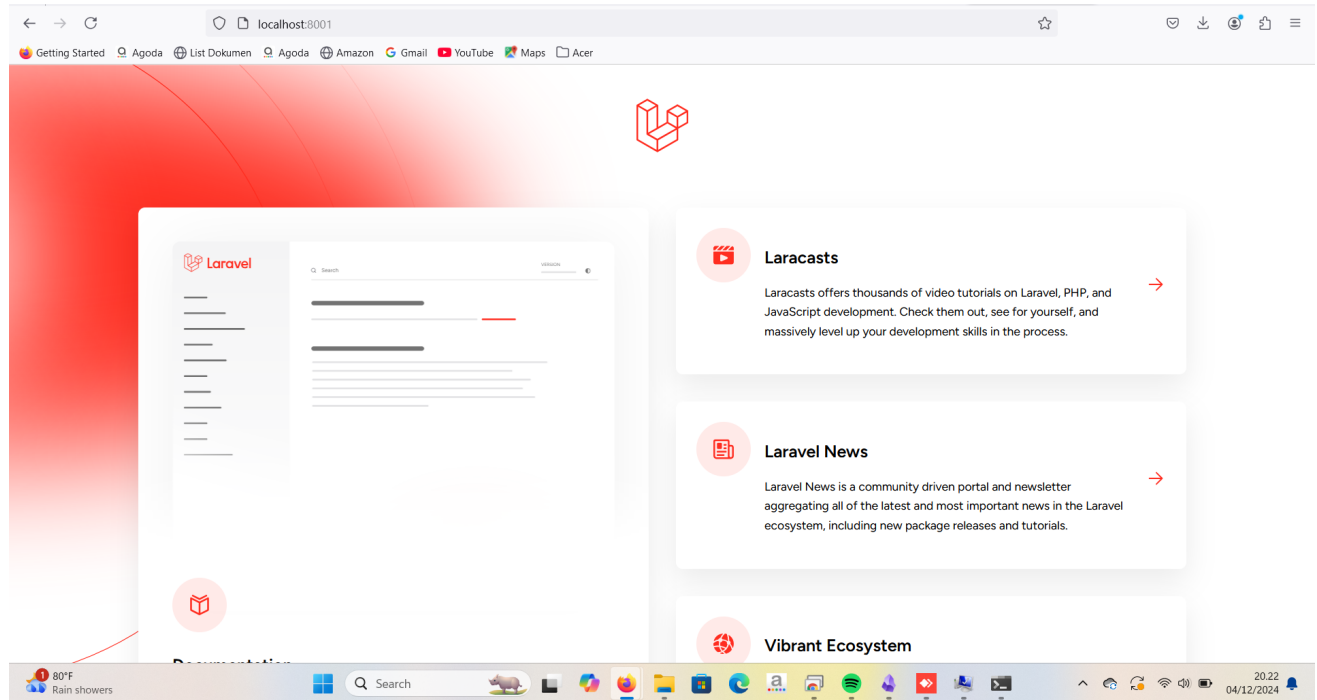
Struktur aplikasi yang baru ini dirancang untuk memberikan pengalaman yang lebih efisien dan *modern*, sambil tetap mempertahankan banyak konsep yang sudah dikenal oleh pengembang

Laravel sebelumnya.

INFORMASI: Bagi pengembang yang sebelumnya menggunakan **Laravel** versi **10** atau lebih rendah dan berencana untuk *upgrade* ke **Laravel 11**, tidak perlu mengadopsi struktur *project* baru ini. Struktur *project* **Laravel** versi lama masih tetap dapat digunakan.

Tampilan Welcome Screen Baru

Di **Laravel 11**, tampilan *welcome screen* telah diperbarui, dan kini tampilannya akan terlihat seperti ilustrasi berikut.



File Bootstrap Aplikasi

File `bootstrap/app.php` telah diperbarui untuk menjadi kode pertama yang dijalankan dalam aplikasi. Dari *file* ini, kita kini dapat mengonfigurasi *route* aplikasi, *middleware*, *service providers*, *exception handling*, dan berbagai pengaturan lainnya. *File* ini menyatukan berbagai pengaturan aplikasi yang sebelumnya tersebar di berbagai bagian struktur *file* aplikasi **Laravel**.

```
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        //
    })
```

```
->withExceptions(function (Exceptions $exceptions) {  
    //  
})->create();
```

Perubahan Route

Pada **Laravel 11**, untuk *default routes* hanya ada dua *file*, yaitu `console.php` dan `web.php`. Lalu, jika ingin membuat *route* untuk *REST API*, caranya sangat mudah, cukup jalankan perintah berikut ini.

```
php artisan install:api
```

Dengan cara tersebut, *file* `api.php` akan dibuat di dalam *folder* `routes`. Begitu pula dengan *route broadcasting*, untuk membuatnya, cukup jalankan perintah berikut ini.

```
php artisan install:broadcasting
```

Perintah Artisan Baru

Perintah Artisan baru telah ditambahkan untuk memudahkan pembuatan *class*, *enum*, *interface*, dan *traits* sebagai berikut:

```
php artisan make:class  
php artisan make:enum  
php artisan make:interface  
php artisan make:trait
```

Model Cast Improvements

Dulu, untuk mendefinisikan `cast`, kita harus menggunakan sebuah properti, contohnya seperti berikut ini.

```
protected $casts = [  
    'email_verified_at' => 'datetime',  
    'password' => 'hashed',  
];
```

Sekarang, di **Laravel 11**, untuk mendefinisikan `cast`, kita menggunakan sebuah `method`, sehingga hasilnya akan terlihat seperti berikut ini.

```
protected function casts(): array
{
    return [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
    ];
}
```

Default Database

Secara *default*, aplikasi **Laravel** yang baru akan menggunakan **SQLite** sebagai database untuk penyimpanan data, serta menggunakan *driver* database yang sama untuk *session*, *cache*, dan *queue*.

Laravel Reverb

[Laravel Reverb](#) menawarkan komunikasi *WebSocket real-time* yang cepat dan skalabel langsung ke aplikasi **Laravel**, serta menyediakan integrasi dengan *event broadcasting Laravel* yang sudah ada, seperti **Laravel Echo**.

Untuk informasi lebih lanjut tentang **Laravel Reverb**, kunjungi <https://laravel.com/docs/11.x/reverb>.

Trait Dumpable

Pada **Laravel 11**, semua kelas telah dilengkapi dengan *method* `dd` atau `dump`. Tujuannya adalah untuk menyederhanakan inti dari *framework* itu sendiri. Berikut ini adalah contohnya.

```
class Stringable implements JsonSerializable, ArrayAccess
{
    use Conditionable, Dumpable, Macroable, Tappable;

    str('foo')->dd();
    str('foo')->dump();
}
```

Migration yang lebih Sederhana

Sekarang di **Laravel 11**, *default migration* yang disertakan akan disederhanakan, sehingga hanya terdapat 3 *file migration* saja.

```
database/migrations
├─ 0001_01_01_000000_create_users_table.php
```

```
|— 0001_01_01_000001_create_cache_table.php
|— 0001_01_01_000002_create_jobs_table.php
```

0 directories, 3 files

Per-Second Rate Limiting

Pada versi sebelumnya, **Laravel** membatasi *rate limiter* menjadi "per menit". Kini, di **Laravel 11**, *rate limiting* mendukung pengaturan "per detik", termasuk untuk permintaan *HTTP* dan *queued jobs*.

```
RateLimiter::for('invoices', function (Request $request) {
    return Limit::perSecond(1);
});
```

Eager Loading Limit

Laravel 11 akan terintegrasi dengan *package* yang akan memudahkan pembuatan *query* dalam *eager loading*. Berikut adalah contohnya.

```
User::select('id', 'name')->with([
    'articles' => fn($query) => $query->limit(5)
])->get();
```

Kesimpulan

Bagi pemula, perubahan struktur di **Laravel 11** yang lebih minimalis mungkin terasa mengejutkan jika dibandingkan dengan versi sebelumnya. Namun, hal ini tidak perlu dikhawatirkan, karena inti dari **Laravel** sejak versi 5 hingga saat ini tetap mempertahankan konsep dasar yang sama.

Cara Install dan Menjalankan Laravel 11

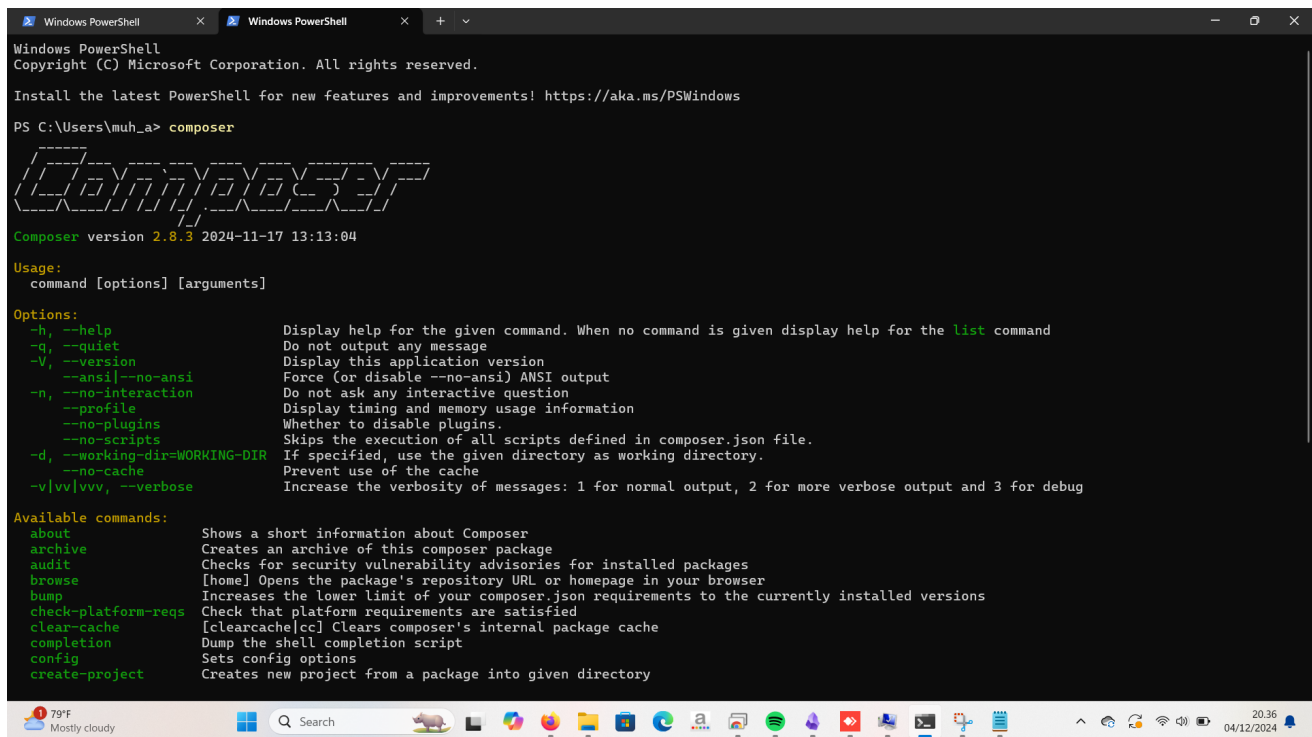
Instalasi Composer

Berikut adalah beberapa cara untuk melakukan instalasi **Composer** sesuai dengan sistem operasi yang digunakan:

- **Installation - Linux / Unix / macOS:** <https://getcomposer.org/doc/00-intro.md#installation-linux-unix-macos>
- **Installation - Windows:** <https://getcomposer.org/doc/00-intro.md#installation-windows>

Untuk memastikan apakah **Composer** berhasil diinstal di komputer, jalankan perintah berikut di terminal/CMD:

composer



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\muh_a> composer

Composer version 2.8.3 2024-11-17 13:13:04

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display help for the list command
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi|--no-ansi         Force (or disable --no-ansi) ANSI output
  -n, --no-interaction     Do not ask any interactive question
  --profile               Display timing and memory usage information
  --no-plugins             Whether to disable plugins.
  --no-scripts            Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache              Prevent use of the cache
  -v|vv|vvv, --verbose    Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:
  about                Shows a short information about Composer
  archive              Creates an archive of this composer package
  audit                Checks for security vulnerability advisories for installed packages
  browse               [home] Opens the package's repository URL or homepage in your browser
  bump                Increases the lower limit of your composer.json requirements to the currently installed versions
  check-platform-reqs Check that platform requirements are satisfied
  clear-cache           [clearcache|cc] Clears composer's internal package cache
  completion           Dump the shell completion script
  config               Sets config options
  create-project        Creates new project from a package into given directory
```

Membuat Project Laravel 11

INFORMASI: Minimal versi PHP yang digunakan adalah 8.2.

Langkah pertama, masuklah ke dalam *folder* tempat Anda ingin menyimpan *project*. Jika menggunakan XAMPP, biasanya *folder* tersebut berada di dalam direktori `htdocs`.

Kemudian, jalankan perintah berikut di dalam terminal atau CMD untuk membuat project **Laravel 11**:

```
composer create-project --prefer-dist laravel/laravel:^11.0 laravel-11
```

Perintah ini akan membuat *project* **Laravel 11** dengan nama `laravel-11`. Tunggu hingga proses instalasi selesai, dan pastikan koneksi internet Anda stabil selama proses ini.

```
PS C:\Users\muh_a> composer create-project --prefer-dist laravel/laravel:^11.0 laravel-11
Creating a "laravel/laravel:11.0" project at "./laravel-11"
Installing laravel/laravel (v11.0.0)
- Downloading laravel/laravel (v11.0.0)
- Installing laravel/laravel (v11.0.0): Extracting archive
Created project in C:\Users\muh_a\laravel-11
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 111 installs, 0 updates, 0 removals
- Locking brick/math (0.12.1)
- Locking carbonphp/carbon-doctrine-types (3.2.0)
- Locking dflydev/dot-access-data (v3.0.3)
- Locking doctrine/inflector (2.0.10)
- Locking doctrine/lexer (3.0.1)
- Locking dragonmantank/cron-expression (v3.4.0)
- Locking egulias/email-validator (4.0.2)
- Locking fakerphp/faker (v1.24.1)
- Locking filp/whoops (2.16.0)
- Locking fruitcake/php-conds (v1.3.0)
- Locking graham-campbell/result-type (v1.1.3)
- Locking guzzlehttp/guzzle (7.9.2)
- Locking guzzlehttp/promises (2.0.4)
- Locking guzzlehttp/psr7 (2.7.0)
- Locking guzzlehttp/uri-template (v1.0.3)
- Locking hamcrest/hamcrest-php (v2.0.1)
- Locking laravel/framework (v11.34.2)
- Locking laravel/pint (v1.18.3)
- Locking laravel/prompts (v0.3.2)
- Locking laravel/sail (v1.39.1)
- Locking laravel/serializable-closure (v2.0.0)
- Locking laravel/tinker (v2.10.0)
- Locking league/commonmark (2.5.3)
- Locking league/config (v1.2.0)
- Locking league/flysystem (3.29.1)
- Locking league/flysystem-local (3.29.0)
- Locking league/mime-type-detection (1.16.0)
- Locking mockery/mockery (1.6.12)
- Locking monolog/monolog (3.8.0)
- Locking myclabs/deep-copy (1.12.1)
- Locking nesbot/carbon (3.8.2)
```

Menjalankan Laravel 11

Setelah proses instalasi **Laravel 11** selesai, langkah selanjutnya adalah menguji coba menjalankan *project* **Laravel**.

1. Pertama, masuk ke dalam *folder project* yang telah dibuat dengan perintah berikut:

```
cd laravel-11
```

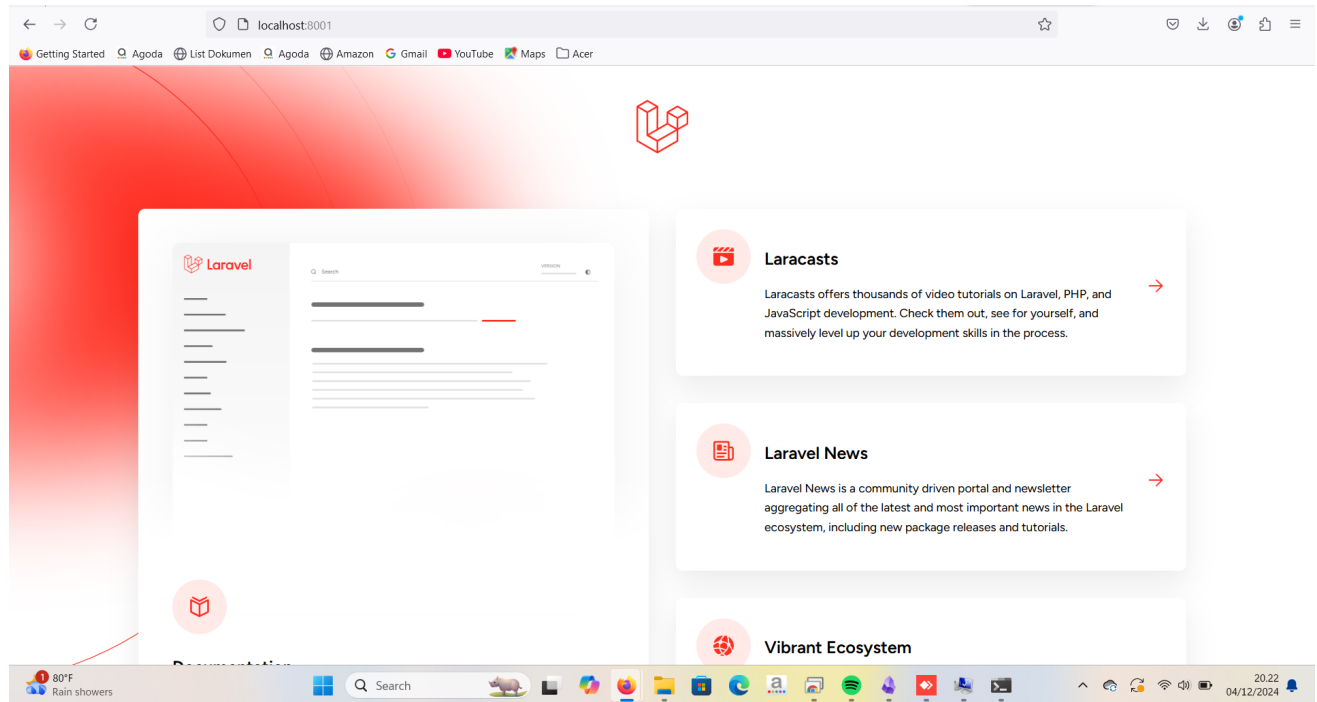
Perintah ini digunakan untuk masuk ke dalam *folder project* yang bernama `laravel-11`.

2. Setelah berhasil masuk ke dalam *project*, jalankan perintah berikut di terminal atau CMD untuk menjalankan *server* **Laravel**:

```
php artisan serve
```

3. Jika proses berhasil, *project* **Laravel** akan berjalan di *localhost* pada port `8000`. Buka *browser* dan akses <http://localhost:8000> untuk melihat hasilnya.

Jika berhasil, tampilan awal *project* **Laravel** akan terlihat seperti berikut.



Menjalankan Storage Link

Laravel hanya dapat mengakses *file* yang berada di dalam *folder public* , sementara *file* atau gambar yang diupload akan disimpan di dalam *folder storage* .

Lalu, bagaimana caranya agar **Laravel** bisa mengakses *file* yang sudah diupload? Solusinya adalah dengan menggunakan *symlink* atau *storage link*.

Perintah *storage link* ini akan membuat sebuah *link* dari *folder storage* ke *folder public* , sehingga **Laravel** dapat mengakses dan membaca *file* yang ada di dalam *folder storage* melalui *folder public* .

Untuk membuat *storage link*, jalankan perintah berikut di dalam terminal/CMD, pastikan Anda sudah berada di dalam *project Laravel*:

```
php artisan storage:link
```

Kesimpulan

Dengan mengikuti langkah-langkah di atas, Anda telah mempelajari cara memulai dan menjalankan *project* baru menggunakan **Laravel 11**. Di artikel berikutnya, kita akan belajar bersama bagaimana cara membuat *Model* dan *Migration* di dalam **Laravel 11**.

Membuat Model dan Migration

Konfigurasi Koneksi Database

Pertama-tama, mari kita konfigurasi koneksi antara **MySQL** dan **Laravel 11**. Proses ini cukup mudah, karena kita hanya perlu mengonfigurasinya di dalam *file* `.env`.

Silakan buka *file* `.env`, kemudian cari baris kode berikut ini:

```
DB_CONNECTION=sqlite
# DB_HOST=127.0.0.1
# DB_PORT=3306
# DB_DATABASE=laravel
# DB_USERNAME=root
# DB_PASSWORD=
```

kemudian ubah menjadi seperti berikut ini.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_laravel11
DB_USERNAME=root
DB_PASSWORD=
```

Di atas, pertama-tama ubah konfigurasi `DB_CONNECTION` yang sebelumnya menggunakan `sqlite` menjadi `mysql`.

Selanjutnya, hilangkan tanda `#` pada konfigurasi key, agar *variable* tersebut diaktifkan. Kemudian, atur nilai `DB_DATABASE` dengan nama database yang ingin digunakan, misalnya `db_laravel_11`.

Untuk `DB_USERNAME`, secara *default* isikan dengan `root`, dan untuk `DB_PASSWORD`, jika menggunakan **XAMPP**, biarkan kosong (kosongkan bagian tersebut).

Membuat Model dan Migration

Selanjutnya, mari kita lanjutkan dengan membuat *Model* dan *Migration* di dalam **Laravel 11**. Silakan jalankan perintah berikut di terminal/CMD, pastikan Anda sudah berada di dalam direktori *project Laravel*:

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

Jika perintah di atas berhasil dijalankan, maka kita akan mendapatkan 2 *file* baru yang berada di dalam *folder* :

1. `app/Models/Product.php`
2. `database/migrations/2024_01_31_080603_create_products_table.php`

INFORMASI : nama *file Migration* akan *random* sesuai dengan tanggal dibuat-nya.

Menambahkan Field / Kolom di dalam Migration

Setelah *file Migration* berhasil dibuat, langkah selanjutnya adalah menambahkan *field* atau kolom di dalamnya. Buka *file migration* tersebut, kemudian pada fungsi `up` , ubah kode-nya seperti berikut.

`database/migrations/2024_01_31_080603_create_products_table.php`

```
public function up(): void
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('image');
        $table->string('title');
        $table->text('description');
        $table->bigInteger('price');
        $table->integer('stock')->default(0);
        $table->timestamps();
    });
}
```

Dari perubahan kode di atas, kita menambahkan 5 *field*, yaitu :

FIELD / KOLOM	TIPE DATA	OPTION
image	string	-
title	string	-
description	text	-
price	bigInteger	-
stock	integer	nilai <i>default</i> 0

Menambahkan Mass Assignment

Mass Assignment merupakan sebuah properti *array* yang berisi *field-field* yang ada di dalam *table / migration*. *Mass Assignment* digunakan agar *field* yang sudah kita tambahkan di dalam *migration* dapat melakukan manipulasi ke dalam *database*.

Buka *file* `app/Models/Product.php`, lalu ubah kodenya menjadi seperti berikut.

`app/Models/Product.php`

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    /**
     * fillable
     *
     * @var array
     */
    protected $fillable = [
        'image',
        'title',
        'description',
        'price',
        'stock',
    ];
}
```

Di atas, kita menambahkan properti `$fillable` dengan jenis *array* dan di dalamnya terdapat *field/kolom* yang sudah kita buat sebelumnya di dalam *file migration*.

Menjalankan Proses Migrate

Sekarang kita akan belajar bagaimana menjalankan proses *migrate* di dalam **Laravel**, proses ini akan *men-generate database* dan *table* beserta *field/kolom* di dalamnya.

Silahkan jalankan perintah berikut ini di dalam terminal/CMD dan pastikan sudah berada di dalam *project Laravel*-nya.

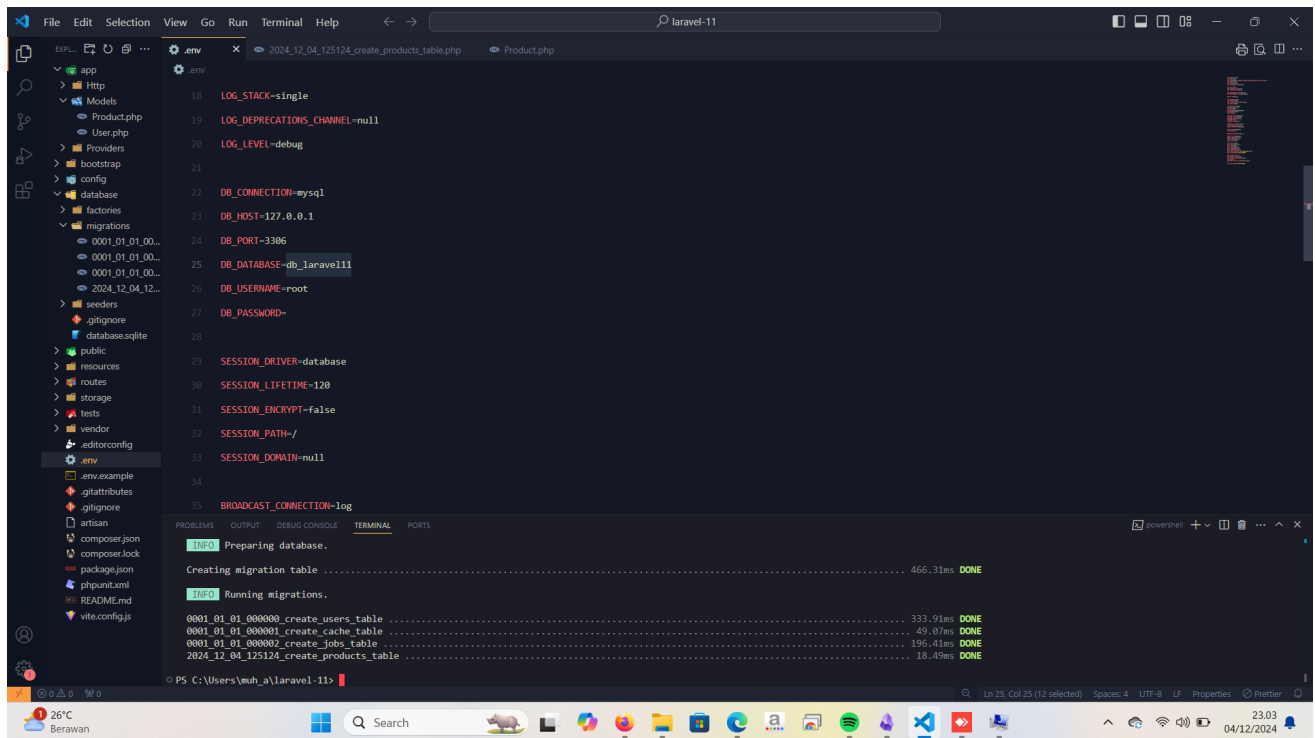
php artisan migrate

Jika keluar pertanyaan seperti berikut ini. Itu artinya kita belum memiliki *database* di dalam **MySQL** dengan nama `db_laravel11` dan apakah kita ingin membuatnya ? Silahkan pilih **Yes** dan **ENTER**.

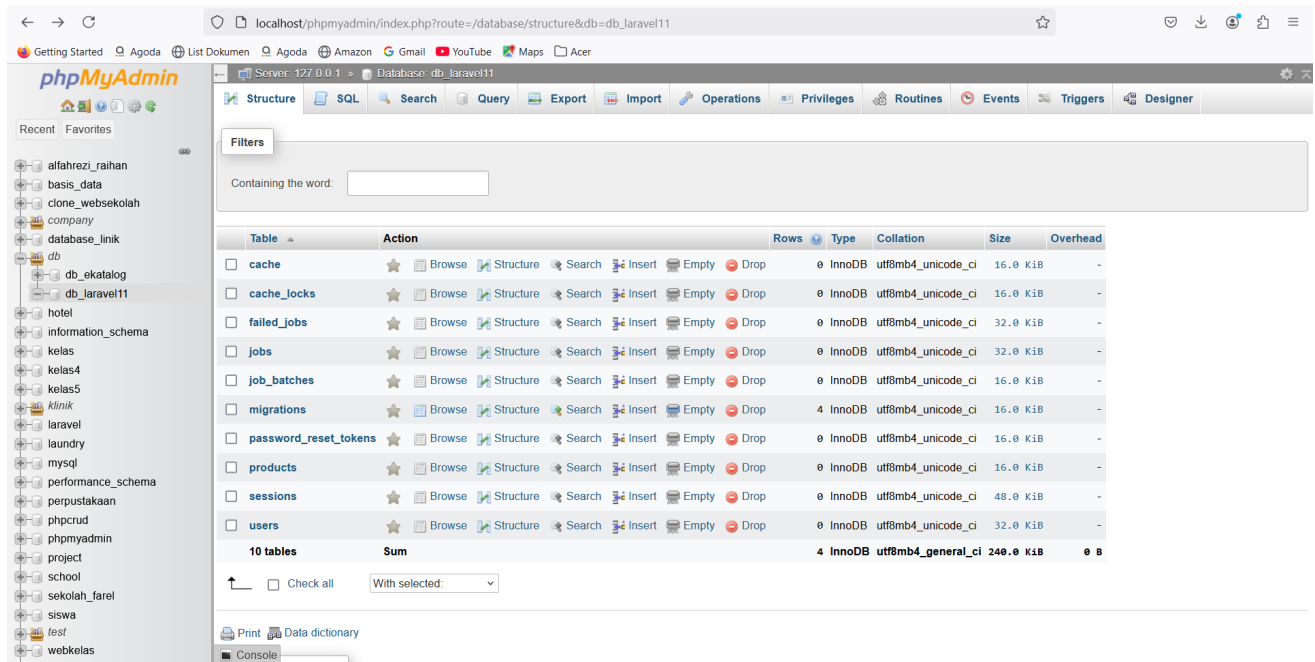
```
WARN The database 'db_laravel11' does not exist on the 'mysql' connection.
```

```
└─ Would you like to create it? ───
```

```
└─ ● Yes / ○ No
```



Dan jika cek pada **PhpMyAdmin** di <http://localhost/phpmyadmin>, maka *table-table* akan berhasil digenerate berserta *field-field* di dalamnya.



Kesimpulan

Sampai disini pembahasan kita terkait bagaimana cara membuat *Model* dan *Migration* di dalam **Laravel 11** dan kita juga belajar menambahkan *field* dan *mass assignment* beserta menjalankan proses *migrate*.

Pada artikel berikutnya, kita semua akan belajar bagaimana cara menampilkan data dari *database* di dalam **Laravel 11**.

Menampilkan Data dari Database

Membuat Controller Product

Pertama, kita akan membuat sebuah *controller* terlebih dahulu, *controller* ini yang bertugas untuk mengatur semua aksi-aksi di dalam *project*. Dan *controller* ini akan menghubungkan antara *model*, *view* dan juga *route*.

Silahkan jalankan perintah berikut ini di dalam terminal/CMD dan pastikan sudah berada di dalam *project Laravel*-nya.

```
php artisan make:controller ProductController
```

Jika perintah di atas berhasil dijalankan, maka kita akan mendapatkan 1 *file controller* baru yang berada di dalam *folder* `app/Http/Controllers/ProductController.php`.

Silahkan buka *file* tersebut, kemudian ubah semua kode-nya menjadi seperti berikut ini.

app/Http/Controllers/ProductController.php

```
<?php

namespace App\Http\Controllers;

//import model product
use App\Models\Product;

//import return type View
use Illuminate\View\View;

class ProductController extends Controller
{
    /**
     * index
     *
     * @return void
     */
    public function index() : View
    {
        //get all products
        $products = Product::latest()->paginate(10);

        //render view with products
        return view('products.index', compact('products'));
    }
}
```

Dari perubahan kode di atas, pertama kita harus melakukan *import Model* Product .

```
//import model product
use App\Models\Product;
```

kemudian kita *import juga return type* View .

```
//import return type View
use Illuminate\View\View;
```

Setelah itu, di dalam *class* ProductController kita menambahkan sebuah *method* baru dengan nama `index` .

```
public function index() : View
{

    //...

}
```

Di dalam *method* tersebut, hal pertama yang kita lakukan adalah memanggil data *products* dari *database* melalui *Model Product*.

```
//get all products
$products = Product::latest()->paginate(10);
```

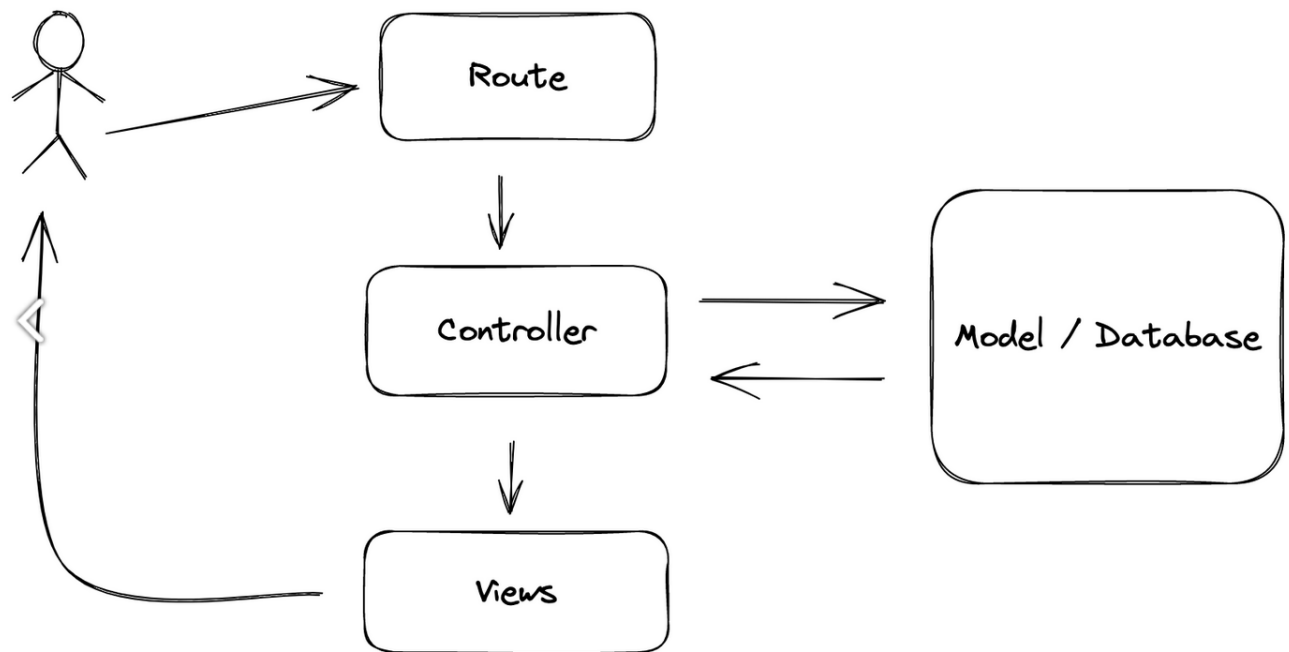
Setelah data berhasil di dapatkan dan ditampung di dalam *variable \$products*, maka langkah berikutnya adalah melakukan *render view* beserta mengirim data *product* yang ada di dalam *variable \$products*.

```
//render view with products
return view('products.index', compact('products'));
```

Di atas, kita gunakan `compact` untuk mengirim data dari *controller* ke dalam *view*.

Membuat Route Resource Product

Setelah *controller* berhasil dibuat, maka langkah berikutnya adalah membuat *route*-nya. *Route* ini akan bertugas sebagai penghubung antara pengguna / *user* dengan aplikasi. Kurang lebih ilustrasinya seperti berikut ini.



Silahkan buka *file* `routes/web.php` , kemudian ubah kode-nya menjadi seperti berikut ini.

`routes/web.php`

```
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

//route resource for products
Route::resource('/products', \App\Http\Controllers\ProductController::class);
```

Di atas, kita membuat sebuah *route* baru dengan *path* `/products` dan kita arahkan ke dalam *class* `ProductController` . Untuk *route* yang kita buat adalah jenis *resource*, artinya **Laravel** akan secara otomatis menghasilkan *route-route* untuk berbagai operasi **CRUD**, seperti:

METHOD	PATH	KETERANGAN
GET	<code>/products</code>	Menampilkan semua <i>products</i> .
GET	<code>/products/create</code>	Menampilkan <i>form</i> untuk membuat <i>product</i> baru.
POST	<code>/products</code>	Menyimpan <i>product</i> baru.
GET	<code>/products/{id}</code>	Menampilkan <i>detail</i> dari sebuah <i>product</i> .
GET	<code>/products/{id}/edit</code>	Menampilkan <i>form</i> untuk mengedit <i>product</i> .

METHOD	PATH	KETERANGAN
PUT/PATCH	/products/{id}	Memperbarui <i>product</i> yang ada.
DELETE	/products/{id}	Menghapus <i>product</i> .

Untuk memastikan apakah *route-route* tersebut sudah digenerate oleh **Laravel**, bisa menjalankan perintah berikut ini di dalam terminal/CMD.

```
php artisan route:list
```

```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 //route resource for products
10 Route::resource('/products', \App\Http\Controllers\ProductController::class);

```

```

GET|HEAD products ..... products.index > ProductController@index
POST products ..... products.store > ProductController@store
GET|HEAD products/create ..... products.create > ProductController@create
GET|HEAD products/{product} ..... products.show > ProductController@show
PUT|PATCH products/{product} ..... products.update > ProductController@update
DELETE products/{product} ..... products.destroy > ProductController@destroy
GET|HEAD products/{product}/edit ..... products.edit > ProductController@edit
up .....
Showing [12] routes

```

Membuat View untuk Menampilkan Data

Sekarang kita akan lanjutkan membuat *view*, yang nanti digunakan untuk menampilkan *list* data *products* dari *database*.

Silahkan buat *folder* baru dengan nama `products` di dalam *folder* `resources/views`, kemudian di dalam *folder* `products` tersebut silahkan buat *file* baru dengan nama `index.blade.php`, kemudian masukkan kode berikut ini di dalamnya.

```
resources/views/products/index.blade.php
```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Data Products - SantriKoding.com</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body style="background: lightgray">

    <div class="container mt-5">
        <div class="row">
            <div class="col-md-12">
                <div>
                    <h3 class="text-center my-4">Tutorial Laravel 11 untuk
Pemula</h3>
                    <h5 class="text-center"><a
href="https://santrikoding.com">www.santrikoding.com</a></h5>
                    <hr>
                </div>
                <div class="card border-0 shadow-sm rounded">
                    <div class="card-body">
                        <a href="{{ route('products.create') }}" class="btn
btn-md btn-success mb-3">ADD PRODUCT</a>
                        <table class="table table-bordered">
                            <thead>
                                <tr>
                                    <th scope="col">IMAGE</th>
                                    <th scope="col">TITLE</th>
                                    <th scope="col">PRICE</th>
                                    <th scope="col">STOCK</th>
                                    <th scope="col" style="width:
20%">ACTIONS</th>
                                </tr>
                            </thead>
                            <tbody>
                                @forelse ($products as $product)
                                    <tr>
                                        <td class="text-center">
                                            
                                        </td>
                                        <td>{{ $product->title }}</td>
                                        <td>{{ "Rp " . number_format($product->
price,2,',','.') }}</td>
                                        <td>{{ $product->stock }}</td>

```

```

        <td class="text-center">
            <form onsubmit="return
confirm('Apakah Anda Yakin ?');" action="{{ route('products.destroy',
$product->id) }}" method="POST">
                <a href="{{
route('products.show', $product->id) }}" class="btn btn-sm btn-dark">SHOW</a>
                <a href="{{
route('products.edit', $product->id) }}" class="btn btn-sm btn-
primary">EDIT</a>
                @csrf
                @method('DELETE')
                <button type="submit"
class="btn btn-sm btn-danger">HAPUS</button>
            </form>
        </td>
    </tr>
    @empty
        <div class="alert alert-danger">
            Data Products belum Tersedia.
        </div>
    @endforelse
</tbody>
</table>
{{ $products->links() }}
</div>
</div>
</div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min
.js"></script>
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

<script>
    //message with sweetalert
    @if(session('success'))
        Swal.fire({
            icon: "success",
            title: "BERHASIL",
            text: "{{ session('success') }}",
            showConfirmButton: false,
            timer: 2000
        });
    @elseif(session('error'))

```

```

        Swal.fire({
            icon: "error",
            title: "GAGAL!",
            text: "{{ session('error') }}",
            showConfirmButton: false,
            timer: 2000
        });
    @endif

</script>

</body>
</html>

```

Dari penambahan kode di atas, untuk menampilkan data kita akan gunakan direktif `@forelse`.

```

@forelse ($products as $product)

    //tampilkan data

@empty

    //Data products belum Tersedia.

@endforelse

```

Untuk *pagination*, kita cukup memanggilnya seperti berikut ini.

```

{{ $products->links() }}

```

kemudian kita menambahkan kondisi untuk memeriksa sebuah *session flash*, yang nantinya difungsikan untuk menampilkan *alert* atau notifikasi setelah berhasil melakukan proses *insert*, *update* dan *delete* data. Dan library yang kita gunakan adalah `Sweet Alert2`.

```

<script>
    //message with sweetalert
    @if(session('success'))
        Swal.fire({
            icon: "success",
            title: "BERHASIL",
            text: "{{ session('success') }}",
            showConfirmButton: false,
            timer: 2000
        });
    @endif
</script>

```

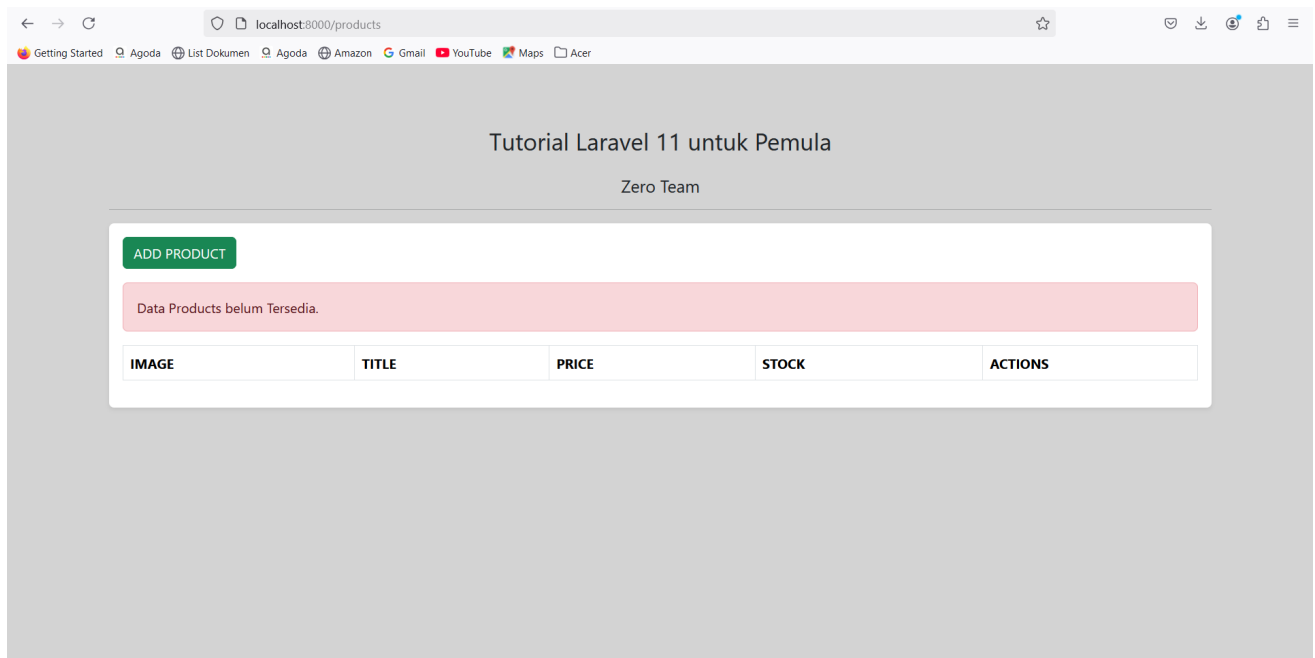
```

    @elseif(session('error'))
        Swal.fire({
            icon: "error",
            title: "GAGAL!",
            text: "{{ session('error') }}",
            showConfirmButton: false,
            timer: 2000
        });
    @endif
</script>

```

Uji Coba Menampilkan Data Products

Sekarang silahkan jalankan *project*-nya dengan mengakses <http://localhost:8000/products> di dalam *browser*, jika berhasil maka hasilnya kurang lebih seperti berikut ini.



Di atas masih menampilkan pesan `Data Products belum Tersedia`, itu karena memang kita masih belum memiliki data apapun di dalam *table* `products`.

Kesimpulan

Pada artikel ini kita telah belajar banyak hal, seperti membuat controller beserta *method*-nya, membuat dan memahami *routing* dan menampilkan data menggunakan *view*.

Pada artikel berikutnya kita semua akan belajar bagaimana cara melakukan proses *insert* data ke dalam *database* di **Laravel 11**.

Insert Data ke Dalam Database

Menambahkan Method `Create` dan `Store` di Controller

Karena akan membuat proses *insert* data, maka kita harus menambahkan 2 *method* baru di dalam *controller*, apa saja *method* tersebut? berikut ini penjelasannya.

METHOD	KETERANGAN
<code>create</code>	Digunakan untuk menampilkan halaman <i>form</i> tambah data
<code>store</code>	Digunakan untuk <i>insert</i> data ke dalam <i>database</i> dan melakukan <i>upload</i> gambar.

Silahkan buka *file* `app/Http/Controllers/ProductController.php`, kemudian ubah semua kode-nya menjadi seperti berikut ini.

`app/Http/Controllers/ProductController.php`

```
<?php

namespace App\Http\Controllers;

//import model product
use App\Models\Product;

//import return type View
use Illuminate\View\View;

//import return type redirectResponse
use Illuminate\Http\RedirectResponse;

//import Http Request
use Illuminate\Http\Request;

class ProductController extends Controller
{
    /**
     * index
     *
     * @return void
     */
    public function index() : View
    {
        //get all products
    }
}
```

```

    $products = Product::latest()->paginate(10);

    //render view with products
    return view('products.index', compact('products'));
}

/**
 * create
 *
 * @return View
 */
public function create(): View
{
    return view('products.create');
}

/**
 * store
 *
 * @param mixed $request
 * @return RedirectResponse
 */
public function store(Request $request): RedirectResponse
{
    //validate form
    $request->validate([
        'image'      => 'required|image|mimes:jpeg,jpg,png|max:2048',
        'title'       => 'required|min:5',
        'description' => 'required|min:10',
        'price'       => 'required|numeric',
        'stock'       => 'required|numeric'
    ]);

    //upload image
    $image = $request->file('image');
    $image->storeAs('public/products', $image->hashName());

    //create product
    Product::create([
        'image'      => $image->hashName(),
        'title'       => $request->title,
        'description' => $request->description,
        'price'       => $request->price,
        'stock'       => $request->stock
    ]);
}

```

```
        //redirect to index
        return redirect()->route('products.index')->with(['success' => 'Data
Berhasil Disimpan!']);
    }
}
```

Dari perubahan kode di atas, pertama kita *import return type* `RedirectResponse` .

```
//import return type redirectResponse
use Illuminate\Http\RedirectResponse;
```

kemudian kita *import juga* `Http Request`.

```
//import Http Request
use Illuminate\Http\Request;
```

Setelah itu, di dalam *class* `ProductController` kita menambahkan 2 *method* baru:

1. function create
2. function store

Function Create

Method ini akan kita gunakan untuk menampilkan halaman *form* tambah data *product*. Pada *method* ini kita hanya melakukan *return* ke dalam sebuah *view*.

```
public function create(): View
{
    return view('products.create');
}
```

Function Store

Method ini akan kita gunakan untuk melakukan proses *insert* data ke dalam *database* dan juga *upload* gambar.

```
public function store(Request $request): RedirectResponse
{
    //...

}
```


Di dalam *method* tersebut, hal pertama yang kita lakukan adalah membuat validasi, yaitu untuk memeriksa apakah data yang dikirimkan sudah sesuai atau belum.

```
//validate form
$request->validate([
    'image'      => 'required|image|mimes:jpeg,jpg,png|max:2048',
    'title'      => 'required|min:5',
    'description' => 'required|min:10',
    'price'      => 'required|numeric',
    'stock'      => 'required|numeric'
]);
```

Dari kode validasi di atas, berikut ini penjelasan lengkapnya.

KEY	VALIDATION	KETERANGAN
image	required	<i>field</i> wajib diisi.
	image	<i>field</i> harus berupa gambar
	mimes:jpeg,jpg,png	<i>field</i> harus memiliki extensi jpeg , jpg dan png .
	max:2048	<i>field</i> maksimal berukuran 2048 Mb / 2Mb.
title	required	<i>field</i> wajib diisi.
	min:5	<i>field</i> minimal memiliki 5 karakter/huruf.
description	required	<i>field</i> wajib diisi.
	min:10	<i>field</i> minimal memiliki 10 karakter/huruf.
price	required	<i>field</i> wajib diisi.
	numeric	<i>field</i> harus berupa angka
stock	required	<i>field</i> wajib diisi.
	numeric	<i>field</i> harus berupa angka

Setelah itu, kita melakukan *upload* gambar menggunakan method `storeAs` .

```
//upload image
$image = $request->file('image');
$image->storeAs('public/products', $image->hashName());
```

Setelah data berhasil di*upload*, maka langkah selanjutnya adalah melakukan proses *insert* data ke dalam *database* menggunakan *Model* `Product` .

```
//create product
Product::create([
    'image'      => $image->hashName(),
    'title'      => $request->title,
    'description' => $request->description,
    'price'      => $request->price,
    'stock'      => $request->stock
]);
```

Jika proses *insert* data berhasil dilakukan, maka kita akan *redirect* ke dalam route yang bernama `products.index` dengan mengirimkan sebuah *session flash* data.

```
//redirect to index
return redirect()->route('products.index')->with(['success' => 'Data Berhasil
Disimpan!']);
```

Membuat View Form Create Product

Setelah berhasil membuat *method* di dalam *controller*, maka langkah selanjutnya adalah membuat halaman *view* untuk menampilkan *form* tambah data *product*.

Silahkan buat *file* baru dengan nama `create.blade.php` di dalam *folder* `resources/views/products`, kemudian masukkan kode berikut ini di dalamnya.

`resources/views/products/create.blade.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Add New Products - SantriKoding.com</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body style="background: lightgray">

    <div class="container mt-5 mb-5">
        <div class="row">
            <div class="col-md-12">
                <div class="card border-0 shadow-sm rounded">
                    <div class="card-body">
```

```

        <form action="{{ route('products.store') }}"
method="POST" enctype="multipart/form-data">

        @csrf

        <div class="form-group mb-3">
            <label class="font-weight-bold">IMAGE</label>
            <input type="file" class="form-control
@error('image') is-invalid @enderror" name="image">

            <!-- error message untuk image -->
            @error('image')
                <div class="alert alert-danger mt-2">
                    {{ $message }}
                </div>
            @enderror
        </div>

        <div class="form-group mb-3">
            <label class="font-weight-bold">TITLE</label>
            <input type="text" class="form-control
@error('title') is-invalid @enderror" name="title" value="{{ old('title') }}"
placeholder="Masukkan Judul Product">

            <!-- error message untuk title -->
            @error('title')
                <div class="alert alert-danger mt-2">
                    {{ $message }}
                </div>
            @enderror
        </div>

        <div class="form-group mb-3">
            <label class="font-weight-
bold">DESCRIPTION</label>
            <textarea class="form-control
@error('description') is-invalid @enderror" name="description" rows="5"
placeholder="Masukkan Description Product">{{ old('description') }}</textarea>

            <!-- error message untuk description -->
            @error('description')
                <div class="alert alert-danger mt-2">
                    {{ $message }}
                </div>
            @enderror
        </div>

```

```

        <div class="row">
            <div class="col-md-6">
                <div class="form-group mb-3">
                    <label class="font-weight-
bold">PRICE</label>

                    <input type="number" class="form-
control @error('price') is-invalid @enderror" name="price" value="{{
old('price') }}" placeholder="Masukkan Harga Product">

                    <!-- error message untuk price -->
                    @error('price')
                        <div class="alert alert-danger mt-
2">

                            {{ $message }}
                        </div>
                    @enderror
                </div>
            </div>
            <div class="col-md-6">
                <div class="form-group mb-3">
                    <label class="font-weight-
bold">STOCK</label>

                    <input type="number" class="form-
control @error('stock') is-invalid @enderror" name="stock" value="{{
old('stock') }}" placeholder="Masukkan Stock Product">

                    <!-- error message untuk stock -->
                    @error('stock')
                        <div class="alert alert-danger mt-
2">

                            {{ $message }}
                        </div>
                    @enderror
                </div>
            </div>
        </div>

        <button type="submit" class="btn btn-md btn-
primary me-3">SAVE</button>
        <button type="reset" class="btn btn-md btn-
warning">RESET</button>

    </form>
</div>
</div>

```

```

        </div>
    </div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min
.js"></script>
<script src="https://cdn.ckeditor.com/4.13.1/standard/ckeditor.js">
</script>
<script>
    CKEDITOR.replace( 'description' );
</script>
</body>
</html>

```

Dari penambahan kode di atas, jika perhatikan pada bagian *form action*, kita arahkan ke dalam *route* yang bernama `products.store`. Artinya itu akan memanggil *method* yang bernama `store` yang berada di dalam *controller*.

```

<form action="{{ route('products.store') }}" method="POST"
enctype="multipart/form-data">

    //...

</form>

```

Pada bagian *form* kita tambahkan *attribute* `enctype="multipart/form-data"`. Ini ditambahkan jika ada sebuah *upload* file di dalam *form*-nya.

Kemudian di dalam JavaScript, bisa perhatikan kita melakukan inisialisasi *Rich Text Editor* menggunakan **CKEditor**. Dan kita akan terapkan ke dalam *textarea* yang memiliki *name* `description`.

```

<script>
    CKEDITOR.replace('description');
</script>

```

Uji Coba Insert Data Product

Silahkan klik *button* `ADD PRODUCT` pada halaman *products index* atau bisa juga ke *URL* berikut ini <http://localhost:8000/products/create>, jika berhasil maka akan menampilkan hasil seperti berikut ini.

IMAGE

Browse... No file selected.

TITLE

Masukkan Judul Product

DESCRIPTION

X Undo Redo Paste Insert Link Unlink Image Table List Bulleted Numbered Source |
B I S T Styles - Normal - ?

|

body p

PRICE

Masukkan Harga Product

STOCK

Masukkan Stock Product

SAVE RESET

Silahkan klik *button* **SAVE** tanpa mengisi data apapun, jika berhasil maka akan menampilkan validasi yang kurang lebih serti berikut ini.

IMAGE

Browse... No file selected.

The image field is required.

TITLE

Masukkan Judul Product

The title field is required.

DESCRIPTION

X Undo Redo Bold Italic Strikethrough Link Unlink List Bulleted List Numbered List Indent Decrease Indent Increase Outdent Source

B **I** **S** **I_x** **¶** **≡** **≡** **≡** Styles - Normal ?


Paragraph Format

body p

Sekarang, silahkan masukkan data di dalam *form* dan klik *button* **SAVE** , jika berhasil maka akan menampilkan hasil seperti berikut ini.

Tutorial Laravel 11 untuk Pemula

Zero Team

ADD PRODUCT				
IMAGE	TITLE	PRICE	STOCK	ACTIONS
	ZERO TEAM	Rp 150.000,00	200	SHOW EDIT HAPUS

Kesimpulan

Pada artikel ini kita telah belajar banyak hal, seperti membuat *method* `create` dan `store` untuk proses *insert* data, membuat fungsi *upload* gambar, membuat halaman *view* untuk form tambah data *product*.

Pada artikel berikutnya, kita semua akan belajar bagaimana cara menampilkan *detail* data *product* berdasarkan ID.

Menampilkan Detail Data By ID

Menambahkan Method `Show` di Controller

Sekarang kita akan menambahkan 1 *method* di dalam *controller* dengan nama `show` dan jika bertanya, apakah harus dengan nama tersebut? jawabannya adalah iya, karena sebelumnya kita menggunakan *route* dengan jenis *resource*, maka untuk penamaan *method-method* di dalam *controller* harus sesuai dengan aturan dari **Laravel**.

Silahkan buka *file* `app/Http/Controllers/ProductController.php`, kemudian ubah semua kode-nya menjadi seperti berikut ini.

```
app/Http/Controllers/ProductController.php
```

```
<?php
```

```
namespace App\Http\Controllers;
```

```
//import model product
use App\Models\Product;

//import return type View
use Illuminate\View\View;

//import return type redirectResponse
use Illuminate\Http\RedirectResponse;

//import Http Request
use Illuminate\Http\Request;

class ProductController extends Controller
{
    /**
     * index
     *
     * @return void
     */
    public function index() : View
    {
        //get all products
        $products = Product::latest()->paginate(10);

        //render view with products
        return view('products.index', compact('products'));
    }

    /**
     * create
     *
     * @return View
     */
    public function create(): View
    {
        return view('products.create');
    }

    /**
     * store
     *
     * @param mixed $request
     * @return RedirectResponse
     */
    public function store(Request $request): RedirectResponse
    {

```



```

//validate form
$request->validate([
    'image'          => 'required|image|mimes:jpeg,jpg,png|max:2048',
    'title'           => 'required|min:5',
    'description'      => 'required|min:10',
    'price'           => 'required|numeric',
    'stock'           => 'required|numeric'
]);

//upload image
$image = $request->file('image');
$image->storeAs('public/products', $image->hashName());

//create product
Product::create([
    'image'          => $image->hashName(),
    'title'           => $request->title,
    'description'      => $request->description,
    'price'           => $request->price,
    'stock'           => $request->stock
]);

//redirect to index
return redirect()->route('products.index')->with(['success' => 'Data
Berhasil Disimpan!']);
}

/**
 * show
 *
 * @param mixed $id
 * @return View
 */
public function show(string $id): View
{
    //get product by ID
    $product = Product::findOrFail($id);

    //render view with product
    return view('products.show', compact('product'));
}
}

```

Dari perubahan kode di atas, kita menambahkan *method* baru dengan nama `show` dan di dalam parameteranya kita berikan *variable* `$id`.

```
public function show(string $id): View
{

    //...

}
```

Di dalam *method* di atas, kita akan melakukan *get* data dari *database* melalui *Model* `Product` dengan *method* `findOrFail`.

```
//get product by ID
$product = Product::findOrFail($id);
```

Setelah data berhasil didapatkan, selanjutnya adalah dikirimkan ke dalam *view*.

```
//render view with product
return view('products.show', compact('product'));
```

Membuat View Detail Data Product

Setelah berhasil menambahkan *method*, maka langkah selanjutnya adalah membuat *view* untuk menampilkan data-nya.

Silahkan buat *file* baru dengan nama `show.blade.php` di dalam *folder* `resources/views/products`, kemudian masukkan kode berikut ini di dalamnya.

`resources/views/products/show.blade.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Show Products - SantriKoding.com</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body style="background: lightgray">

    <div class="container mt-5 mb-5">
        <div class="row">
```

```

        <div class="col-md-4">
            <div class="card border-0 shadow-sm rounded">
                <div class="card-body">
                    
                </div>
            </div>
        </div>
        <div class="col-md-8">
            <div class="card border-0 shadow-sm rounded">
                <div class="card-body">
                    <h3>{{ $product->title }}</h3>
                    <hr/>
                    <p>{{ "Rp " . number_format($product->price,2,',','.') }}</p>

                    <code>
                        <p>{!! $product->description !!}</p>
                    </code>
                    <hr/>
                    <p>Stock : {{ $product->stock }}</p>
                </div>
            </div>
        </div>
    </div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Dari penambahan kode di atas, kita menampilkan *detail* data-nya dengan cara memanggil objek dari `$product`.

1. Menampilkan Gambar

```



```

2. Menampilkan Title

```

{{ $product->title }}

```

3. Menampilkan Harga

```
{{ "Rp " . number_format($product->price,2,',','.') }}
```

Di atas, kita gunakan *function* `number_format` dari **PHP** untuk memformat harga *product*.

4. Menampilkan Description

```
{{ !! $product->description !! }}
```

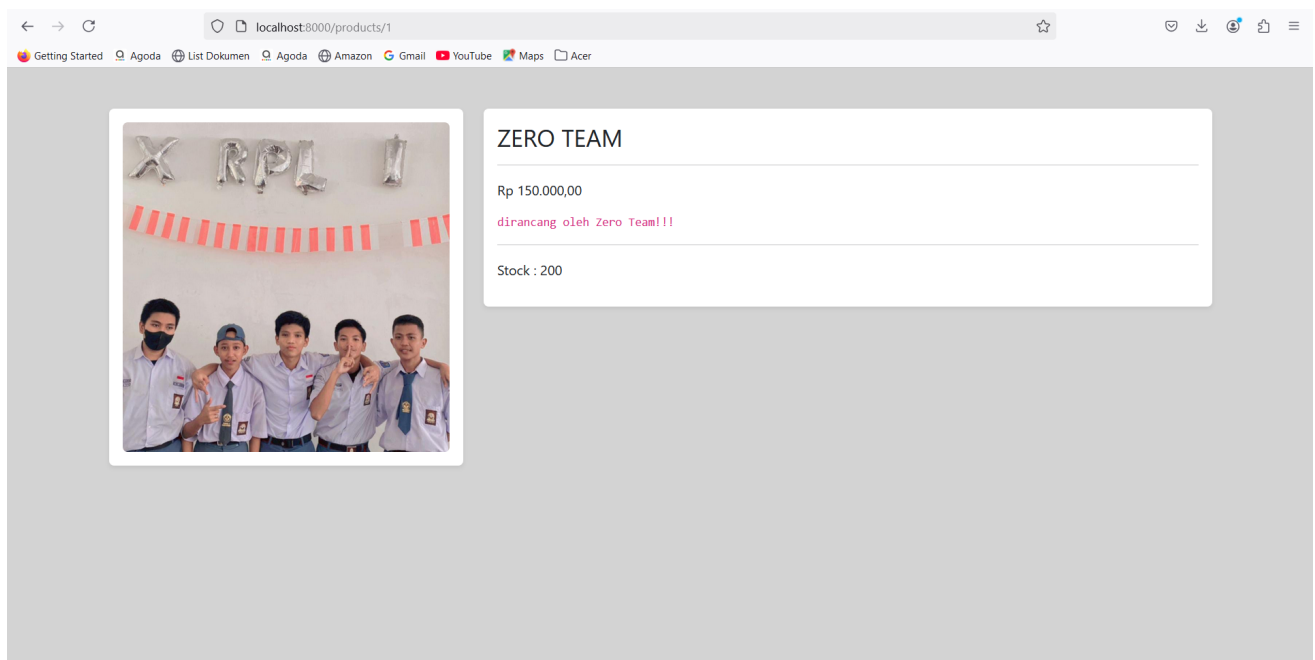
Karena di dalam *description* terdapat sintaks **HTML**, maka kita harus me-`_render_`nya menggunakan `{!! !!}`

5. Menampilkan Stock

```
{{ $product->stock }}
```

Uji Coba Menampilkan Detail Data Product

Silahkan klik *button* `SHOW` di data yang miliki, jika berhasil maka akan menampilkan hasil seperti berikut ini.



Kesimpulan

kita semua telah belajar bagaimana cara menampilkan *detail* data berdasarkan ID, dimana data **ID** tersebut akan diambil dari parameter di dalam *URL browser*.

Edit dan Update Data ke Database

Menambahkan Method `edit` dan `update` di Controller

Pada proses pembuatan *edit* dan *update*, kita membutuhkan 2 *method* tambahan di dalam *controller*, *method* tersebut bernama `edit` dan `update`. Berikut penjelasannya.

METHOD	KETERANGAN
<code>edit</code>	Digunakan untuk menampilkan halaman <i>form</i> edit data
<code>update</code>	Digunakan untuk update data ke dalam <i>database</i> dan melakukan <i>upload</i> gambar (opsional).

Sekarang, silahkan buka *file* `app/Http/Controllers/ProductController.php`, kemudian ubah semua kode-nya menjadi seperti berikut ini.

`app/Http/Controllers/ProductController.php`

```
<?php

namespace App\Http\Controllers;

//import model product
use App\Models\Product;

//import return type View
use Illuminate\View\View;

//import return type redirectResponse
use Illuminate\Http\Request;

//import Http Request
use Illuminate\Http\RedirectResponse;

//import Facades Storage
use Illuminate\Support\Facades\Storage;

class ProductController extends Controller
{
    /**
     * index
     *
     * @return void
```

```

*/
public function index() : View
{
    //get all products
    $products = Product::latest()->paginate(10);

    //render view with products
    return view('products.index', compact('products'));
}

/**
 * create
 *
 * @return View
 */
public function create(): View
{
    return view('products.create');
}

/**
 * store
 *
 * @param mixed $request
 * @return RedirectResponse
 */
public function store(Request $request): RedirectResponse
{
    //validate form
    $request->validate([
        'image'      => 'required|image|mimes:jpeg,jpg,png|max:2048',
        'title'       => 'required|min:5',
        'description' => 'required|min:10',
        'price'       => 'required|numeric',
        'stock'       => 'required|numeric'
    ]);

    //upload image
    $image = $request->file('image');
    $image->storeAs('public/products', $image->hashName());

    //create product
    Product::create([
        'image'      => $image->hashName(),
        'title'       => $request->title,
        'description' => $request->description,
    ]);
}

```

```

        'price'          => $request->price,
        'stock'          => $request->stock
    });

    //redirect to index
    return redirect()->route('products.index')->with(['success' => 'Data
Berhasil Disimpan!']);
}

/**
 * show
 *
 * @param mixed $id
 * @return View
 */
public function show(string $id): View
{
    //get product by ID
    $product = Product::findOrFail($id);

    //render view with product
    return view('products.show', compact('product'));
}

/**
 * edit
 *
 * @param mixed $id
 * @return View
 */
public function edit(string $id): View
{
    //get product by ID
    $product = Product::findOrFail($id);

    //render view with product
    return view('products.edit', compact('product'));
}

/**
 * update
 *
 * @param mixed $request
 * @param mixed $id
 * @return RedirectResponse
 */

```

```

public function update(Request $request, $id): RedirectResponse
{
    //validate form
    $request->validate([
        'image'          => 'image|mimes:jpeg,jpg,png|max:2048',
        'title'           => 'required|min:5',
        'description'      => 'required|min:10',
        'price'            => 'required|numeric',
        'stock'            => 'required|numeric'
    ]);

    //get product by ID
    $product = Product::findOrFail($id);

    //check if image is uploaded
    if ($request->hasFile('image')) {

        //upload new image
        $image = $request->file('image');
        $image->storeAs('public/products', $image->hashName());

        //delete old image
        Storage::delete('public/products/' . $product->image);

        //update product with new image
        $product->update([
            'image'          => $image->hashName(),
            'title'           => $request->title,
            'description'      => $request->description,
            'price'            => $request->price,
            'stock'            => $request->stock
        ]);
    } else {

        //update product without image
        $product->update([
            'title'           => $request->title,
            'description'      => $request->description,
            'price'            => $request->price,
            'stock'            => $request->stock
        ]);
    }

    //redirect to index
    return redirect()->route('products.index')->with(['success' => 'Data

```



```
Berhasil Diubah!'][];  
    }  
}
```

Dari perubahan kode di atas, pertama kita *import Facades Storage* dari **Laravel**. Ini akan kita gunakan untuk menghapus *file* gambar *product* lama saat gambar diperbarui.

```
//import Facades Storage  
use Illuminate\Support\Facades\Storage;
```

kemudian di dalam *class productController* , kita membuat 2 *method* baru, yaitu:

1. *function edit*
2. *function update*

Function Edit

Method ini akan kita gunakan untuk menampilkan halaman *form edit* data product dan di dalam *method* tersebut terdapat parameter *\$id* yang mana isinya akan diambil dari ID yang ada di dalam *URL browser*.

```
public function edit(string $id): View  
{  
  
    //...  
  
}
```

Di dalam *method* tersebut, kita melakukan *get* data ke dalam *database* menggunakan *Model* berdasarkan ID.

```
//get product by ID  
$product = Product::findOrFail($id);
```

Setelah data didapatkan, kita tinggal kirimkan data tersebut ke dalam *view* menggunakan *compact* .

```
//render view with product  
return view('products.edit', compact('product'));
```

Function Update

Method ini akan kita gunakan untuk proses *update* data *product* ke dalam *database*. Di dalam *method* ini terdapat 2 parameter, yaitu `$request` dan `$id`.

1. `$request` - digunakan untuk menerima request data yang dikirimkan oleh pengguna melalui *form*.
2. `$id` - merupakan **ID** data *product* yang akan akan dijadikan acuan *update* data.

```
public function update(Request $request, $id): RedirectResponse
{

    //...

}
```

Di dalam *method* di atas, pertama-tama kita membuat sebuah validasi terlebih dahulu.

```
//validate form
$request->validate([
    'image'      => 'image|mimes:jpeg,jpg,png|max:2048',
    'title'      => 'required|min:5',
    'description' => 'required|min:10',
    'price'      => 'required|numeric',
    'stock'      => 'required|numeric'
]);
```

Dari penambahan validasi di atas, kurang lebih ini penjelasannya.

KEY	VALIDATION	KETERANGAN
image	image	<i>field</i> harus berupa gambar
	mimes:jpeg,jpg,png	<i>field</i> harus memiliki extensi jpeg , jpg dan png .
	max:2048	<i>field</i> maksimal berukuran 2048 Mb / 2Mb.
title	required	<i>field</i> wajib diisi.
	min:5	<i>field</i> minimal memiliki 5 karakter/huruf.
description	required	<i>field</i> wajib diisi.
	min:10	<i>field</i> minimal memiliki 10 karakter/huruf.
price	required	<i>field</i> wajib diisi.
	numeric	<i>field</i> harus berupa angka
stock	required	<i>field</i> wajib diisi.
	numeric	<i>field</i> harus berupa angka

KEY	VALIDATION	KETERANGAN
-----	------------	------------

Setelah itu, kita melakukan get data ke dalam *database* menggunakan *Model* berdasarkan **ID**.

```
//get product by ID
$product = Product::findOrFail($id);
```

kemudian kita membuat kondisi untuk gambar, jika terdapat *request* gambar

```
//check if image is uploaded
if ($request->hasFile('image')) {

    //upload gambar baru

    //hapus gambar lama

    //update data product dengan gambar baru

} else {

    //update data product tanpa gambar

}
```

Di atas, jika ada sebuah *request* dengan nama `image`, maka kita akan melakukan beberapa aksi, yaitu:

1. Upload gambar baru

```
//upload new image
$image = $request->file('image');
$image->storeAs('public/products', $image->hashName());
```

2. Hapus gambar lama

```
//delete old image
Storage::delete('public/products/'.$product->image);
```

3. Update data *product*

```
//update product with new image
$product->update([
    'image'      => $image->hashName(),
    'title'      => $request->title,
    'description' => $request->description,
    'price'      => $request->price,
    'stock'      => $request->stock
]);
```

Tapi jika request `image` tidak ada, maka kita cukup *update* data *product* tanpa gambar.

```
//update product without image
$product->update([
    'title'      => $request->title,
    'description' => $request->description,
    'price'      => $request->price,
    'stock'      => $request->stock
]);
```

Setelah itu, kita tinggal *redirect* ke dalam sebuah *route* dengan nama `products.index` dengan menambahkan *flash* data.

```
//redirect to index
return redirect()->route('products.index')->with(['success' => 'Data Berhasil
Diubah!']);
```

Membuat View Form Edit Data Product

Langkah selanjutnya adalah membuat *view* untuk menampilkan halaman *form edit* data *product*. Silahkan buat *file* baru dengan nama `edit.blade.php` di dalam *folder* `resources/views/products`, kemudian masukkan kode berikut ini di dalamnya.

`resources/views/products/edit.blade.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Edit Products - SantriKoding.com</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
```

[illegible]

```
@error('description') is-invalid @enderror" name="description" rows="5"
placeholder="Masukkan Description Product">{{ old('description', $product-
>description) }}</textarea>
```

```
<!-- error message untuk description -->
@error('description')
    <div class="alert alert-danger mt-2">
        {{ $message }}
    </div>
@enderror
</div>
```

```
<div class="row">
    <div class="col-md-6">
        <div class="form-group mb-3">
            <label class="font-weight-
bold">PRICE</label>

            <input type="number" class="form-
control @error('price') is-invalid @enderror" name="price" value="{{
old('price', $product->price) }}" placeholder="Masukkan Harga Product">
```

```
<!-- error message untuk price -->
@error('price')
    <div class="alert alert-danger mt-
2">

        {{ $message }}
    </div>
@enderror
</div>
```

```
<div class="col-md-6">
    <div class="form-group mb-3">
        <label class="font-weight-
bold">STOCK</label>

        <input type="number" class="form-
control @error('stock') is-invalid @enderror" name="stock" value="{{
old('stock', $product->stock) }}" placeholder="Masukkan Stock Product">
```

```
<!-- error message untuk stock -->
@error('stock')
    <div class="alert alert-danger mt-
2">

        {{ $message }}
    </div>
@enderror
</div>
```

```

        </div>
    </div>

    <button type="submit" class="btn btn-md btn-
primary me-3">UPDATE</button>
    <button type="reset" class="btn btn-md btn-
warning">RESET</button>

    </form>
</div>
</div>
</div>
</div>
</div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min
.js"></script>
    <script src="https://cdn.ckeditor.com/4.13.1/standard/ckeditor.js">
</script>
    <script>
        CKEDITOR.replace( 'description' );
    </script>
</body>
</html>

```

Pada penambahan kode di atas, untuk mengisi *value* kita panggil object dari data *product* di dalam *helper* `old`. Contohnya seperti berikut.

```
{{ old('title', $product->title) }}
```

Dan jangan lupa, untuk halaman *form* edit, pastikan menambahakn *method* `PUT` di dalamnya. Ini menandakan bahwa *form* tersebut adalah *form edit* data.

```
@method('PUT')
```

Uji Coba Edit dan Update Product

Silahkan klik *button* `EDIT` di salah satu data yang dimiliki, jika berhasil maka akan menampilkan halaman *edit* data *product*, kurang lebih seperti berikut ini.

localhost:8000/products/1/edit

Getting Started Agoda List Dokumen Agoda Amazon Gmail YouTube Maps Acer

IMAGE
Browse... No file selected.

TITLE
ZERO TEAM

DESCRIPTION
dirancang oleh Zero Team!!!

body p

PRICE
150000

STOCK
200

UPDATE RESET

Silahkan sesuaikan isinya, kemudian klik *button* UPDATE , jika berhasil maka akan menampilkan halaman seperti berikut ini.

localhost:8000/products

Getting Started Agoda List Dokumen Agoda Amazon Gmail YouTube Maps Acer

Tutorial Laravel 11 untuk Pemula

Zero Team

ADD PRODUCT

IMAGE	TITLE	PRICE	STOCK	ACTIONS
	ZERO TEAM EDITED	Rp 15.000,00	20	SHOW EDIT HAPUS

Kesimpulan

Pada moudl ini kita telah belajar bagaimana cara membuat proses *edit* dan *update* data di **Laravel 11**, dengan cara menambahkan 2 *method* di dalam *controller*, yaitu `edit` dan `update` dan membuat halaman *view* untuk *form edit*-nya.

Delete Data dari Database

Menambahkan Method Destroy

Sekarang kita akan menambahkan *method* baru di dalam *controller*, method ini yang bertugas melakukan hapus data ke dalam *database* dan juga menghapus gambar di dalam *project* **Laravel**.

Silahkan buka *file* `app/Http/Controllers/ProductController.php`, kemudian ubah semua kode-nya menjadi seperti berikut ini.

`app/Http/Controllers/ProductController.php`

```
<?php

namespace App\Http\Controllers;

//import model product
use App\Models\Product;

//import return type View
use Illuminate\View\View;

//import return type redirectResponse
use Illuminate\Http\Request;

//import Http Request
use Illuminate\Http\RedirectResponse;

//import Facades Storage
use Illuminate\Support\Facades\Storage;

class ProductController extends Controller
{
    /**
     * index
     *
     * @return void
     */
    public function index() : View
    {
        //get all products
        $products = Product::latest()->paginate(10);

        //render view with products
    }
}
```

```

        return view('products.index', compact('products'));
    }

    /**
     * create
     *
     * @return View
     */
    public function create(): View
    {
        return view('products.create');
    }

    /**
     * store
     *
     * @param mixed $request
     * @return RedirectResponse
     */
    public function store(Request $request): RedirectResponse
    {
        //validate form
        $request->validate([
            'image'      => 'required|image|mimes:jpeg,jpg,png|max:2048',
            'title'       => 'required|min:5',
            'description' => 'required|min:10',
            'price'       => 'required|numeric',
            'stock'       => 'required|numeric'
        ]);

        //upload image
        $image = $request->file('image');
        $image->storeAs('public/products', $image->hashName());

        //create product
        Product::create([
            'image'      => $image->hashName(),
            'title'       => $request->title,
            'description' => $request->description,
            'price'       => $request->price,
            'stock'       => $request->stock
        ]);

        //redirect to index
        return redirect()->route('products.index')->with(['success' => 'Data
        Berhasil Disimpan!']);
    }

```

```

}

/**
 * show
 *
 * @param mixed $id
 * @return View
 */
public function show(string $id): View
{
    //get product by ID
    $product = Product::findOrFail($id);

    //render view with product
    return view('products.show', compact('product'));
}

/**
 * edit
 *
 * @param mixed $id
 * @return View
 */
public function edit(string $id): View
{
    //get product by ID
    $product = Product::findOrFail($id);

    //render view with product
    return view('products.edit', compact('product'));
}

/**
 * update
 *
 * @param mixed $request
 * @param mixed $id
 * @return RedirectResponse
 */
public function update(Request $request, $id): RedirectResponse
{
    //validate form
    $request->validate([
        'image'      => 'image|mimes:jpeg,jpg,png|max:2048',
        'title'       => 'required|min:5',
        'description' => 'required|min:10',
    ]);
}

```

```

        'price'          => 'required|numeric',
        'stock'          => 'required|numeric'
    ]);

    //get product by ID
    $product = Product::findOrFail($id);

    //check if image is uploaded
    if ($request->hasFile('image')) {

        //upload new image
        $image = $request->file('image');
        $image->storeAs('public/products', $image->hashName());

        //delete old image
        Storage::delete('public/products/'.$product->image);

        //update product with new image
        $product->update([
            'image'          => $image->hashName(),
            'title'          => $request->title,
            'description'     => $request->description,
            'price'          => $request->price,
            'stock'          => $request->stock
        ]);

    } else {

        //update product without image
        $product->update([
            'title'          => $request->title,
            'description'     => $request->description,
            'price'          => $request->price,
            'stock'          => $request->stock
        ]);
    }

    //redirect to index
    return redirect()->route('products.index')->with(['success' => 'Data
    Berhasil Diubah!']);
}

/**
 * destroy
 *
 * @param mixed $id

```

```

    * @return RedirectResponse
    */
    public function destroy($id): RedirectResponse
    {
        //get product by ID
        $product = Product::findOrFail($id);

        //delete image
        Storage::delete('public/products/'. $product->image);

        //delete product
        $product->delete();

        //redirect to index
        return redirect()->route('products.index')->with(['success' => 'Data
        Berhasil Dihapus!']);
    }
}

```

Dari perubahan kode di atas, kita menambahkan *method* baru dengan nama `destroy` dan pada parameternya kita berikan `$id`, yang nanti nilainya akan diambil dari *URL browser*.

```

public function destroy($id): RedirectResponse
{

    //...

}

```

Di dalam *method* tersebut, pertama, kita akan mencari data *product* berdasarkan ID.

```

//get product by ID
$product = Product::findOrFail($id);

```

Jika sudah ditemukan, maka kita akan melakukan *delete* gambar.

```

//delete image
Storage::delete('public/products/'. $product->image);

```

Setelah gambar berhasil dihapus, langkah selanjutnya adalah menghapus data yang terkait dari *database*.

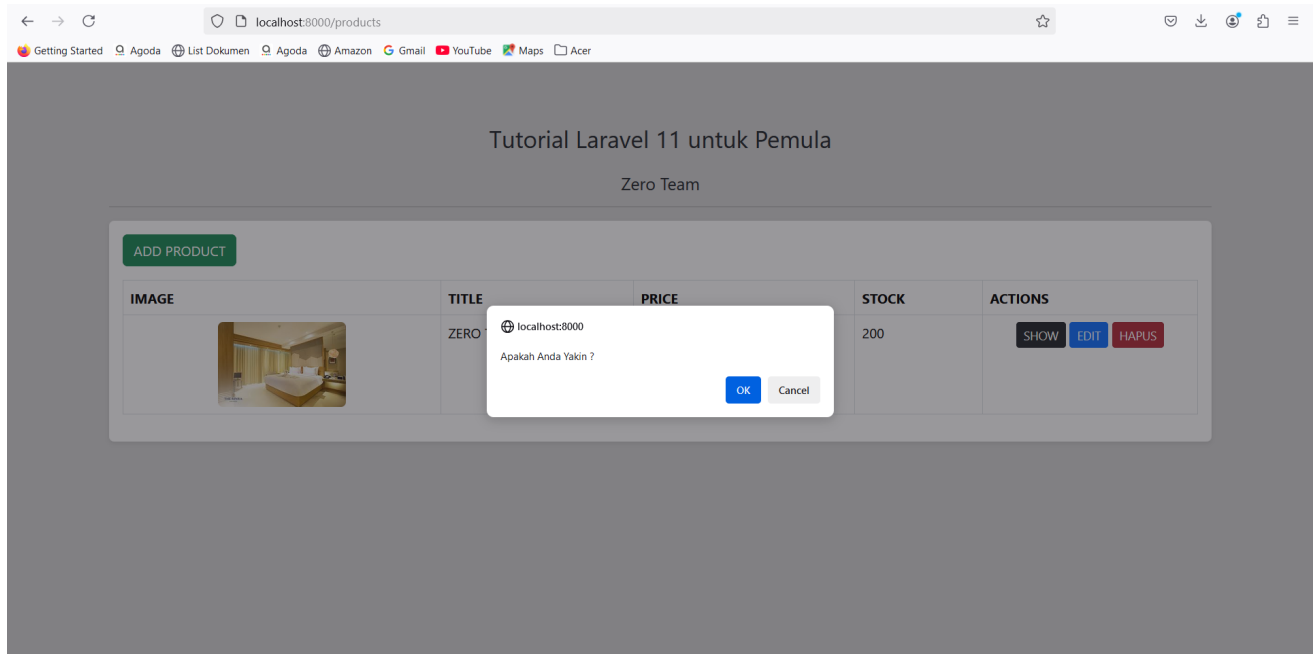
```
//delete product  
$product->delete();
```

Selanjutnya, dilakukan *redirect* ke route dengan nama `products.index` sambil mengirimkan data *flash session*.

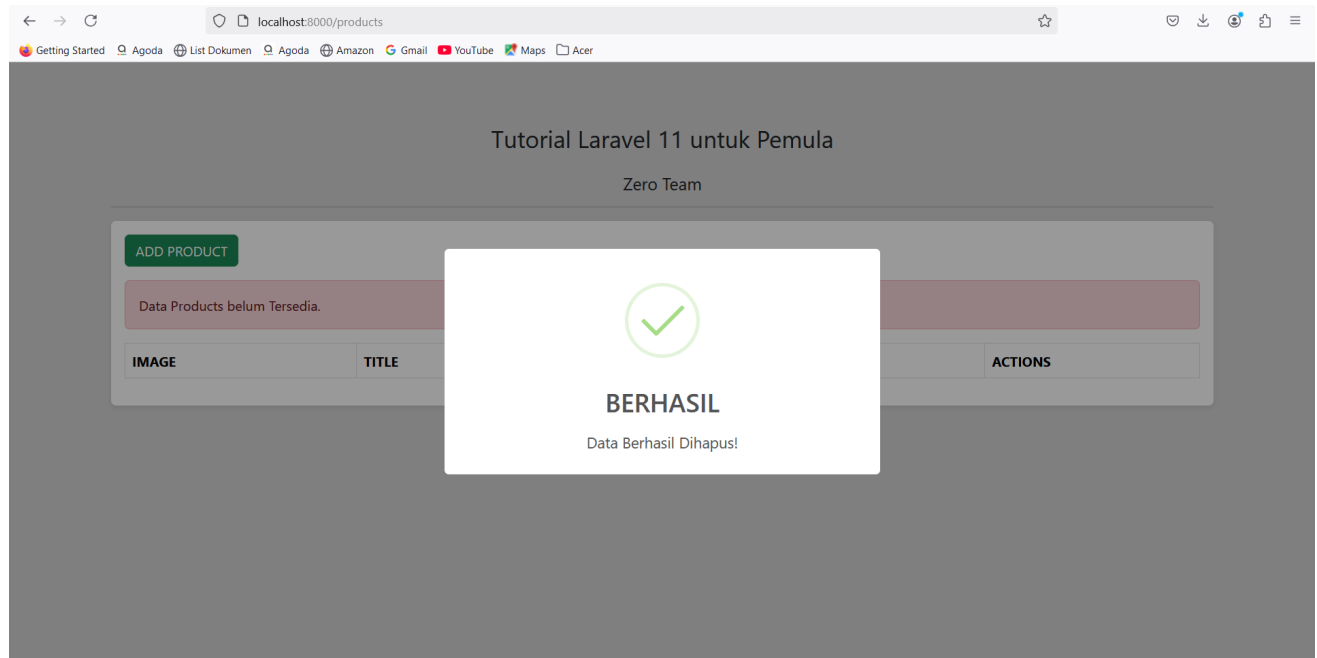
```
//redirect to index  
return redirect()->route('products.index')->with(['success' => 'Data Berhasil  
Dihapus!']);
```

Langkah 2 - Uji Coba Delete Data Product

Klik *button* DELETE pada salah satu data yang tersedia. Jika berhasil, akan muncul *alert* konfirmasi seperti ilustrasi berikut.



Jika tombol `OK` diklik, maka data akan terhapus, dan hasilnya akan terlihat seperti pada ilustrasi berikut.



Kesimpulan

Pada tahap ini kita telah belajar menambahkan *method* untuk proses *delete* data dari *database* dan menghapus gambar.