

WEB APPLICATION DEVELOPMENT



MODUL 5

CRUD DENGAN LARAVEL TANPA AUTHENTICATION

Tim Penyusun

Nama	Kode Asisten
Afif Priandhika Izzulhaq	AFF
Akbar Affaruk Khuzaimi Ahmadani	AFK
Arddhana Zhafran Amanullah	ZAF
Daris Raihan Dityo	DAR
Dewi Astuti	DEA
Disa Hagai Tarigan	GAI
Evan Reswara	EVN
Farhan Anggara Putra Pratama	RTM
Galih Rashif Husaini	GHR
Henry Augusta Harsono	NRY
Immanuel Arya	NUE
Krisna Dwi Permana	KRI
M. Faiz Triputra	LOL
Muhammad Afif Rizqi	MAF
Muhammad Hanif Zahran	MHZ
Muhammad Raihan Januar	COK
Nadya Zahra	NDY
Rafi Adinegoro	ADI
Rizalrasyd Dwiselia Ridwanah	ZAY
Thasya Ummul Kulsum	TYU

DAFTAR ISI

DAFTAR ISI	2
TUJUAN PRAKTIKUM	3
KEBUTUHAN PRAKTIKUM	3
Pengenalan Laravel	4
Syntax	7
Migration	8
Seed	10
MVC (Model-View-Controller)	11
Model	11
View	14
Controller	17
CRUD (Create-Read-Update-Delete)	21
CONTOH SOAL	25
DAFTAR PUSTAKA	33

TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan dan memahami Laravel
2. Mampu memahami *model*, *view* dan *controller* pada framework Laravel
3. Mampu mengimplementasikan *model*, *view* dan *controller* pada framework Laravel
4. Mampu mengimplementasikan *create*, *read*, *update*, *delete* menggunakan framework Laravel

KEBUTUHAN PRAKTIKUM

1. XAMPP
2. Composer
3. Laravel 8
4. Visual Studio Code
5. Git Bash

LARAVEL

1. Pengenalan Laravel

Laravel merupakan salah satu Framework PHP yang *open source* dengan tujuan mengembangkan aplikasi berbasis web berarsitektur MVC (*Model-View-Controller*). Fitur yang dimiliki oleh laravel yaitu desain yang modular, memiliki berbagai cara yang bisa digunakan untuk mengakses database sehingga memudahkan *developer* dalam mengembangkan aplikasinya atau *maintenance* hanya dengan sintaks yang pendek dan mudah dipahami. Fitur utama yang ada pada Laravel:

Blade Template Engine	Blade adalah <i>template engine</i> yang digunakan untuk mendesain <i>layout</i> yang unik, dapat digunakan di tampilan lain sehingga terdapat konsistensi desain dan struktur selama proses pengembangan. Kelebihan yang dimiliki blade yaitu tidak adanya batasan penggunaan kode PHP biasa bagi pengembang dalam tampilan dan desain tampilan blade akan berada di- <i>cache</i> hingga adanya modifikasi.
CSRF	Laravel membantu melindungi aplikasi dari serangan lintas situs (CSRF) dimana perintah yang tidak sah dilakukan atas nama pengguna yang di autentifikasi. "Token" CSRF otomatis dihasilkan oleh Laravel untuk memverifikasi pengguna benar-benar membuat permintaan ke aplikasi. Pada blade dapat menggunakan arahan <code>@csrf</code> Blade untuk menghasilkan bidang token: <pre><form method="POST" action="/profile"> @csrf ... </form></pre>
Routing	<i>Request</i> yang ada pada Laravel dipetakan dengan bantuan <i>route</i> . Routing merupakan pemetaan <i>route</i> request ke kontroler terkait, dapat mempermudah dalam mengembangkan website dan meningkatkan performanya. Ada tiga kategori pada <i>routing</i> : <i>basic routing</i> , <i>route parameters</i> , dan <i>named routes</i> .
Modularity	Pada Laravel terdapat kumpulan modul dan <i>library</i> yang terkait dengan composer untuk membantu pengembang dalam menyempurnakan dan meningkatkan fungsionalitas website yang dibangun, serta mempermudah proses update.
Testability	Laravel memiliki fitur proses pengecekan yang cukup lengkap. Adanya PHPUnit dan file <code>phpunit.xml</code> yang membantu proses pengecekan dengan menyesuaikan aplikasi web yang sedang dibangun. Laravel dibangun menggunakan metode pembantu yang nyaman sehingga memungkinkan pengembang untuk melakukan pengujian

	<i>website</i> secara ekspresif.
Query Builder and ORM (Object Relational Mapping)	Laravel <i>database query builder</i> menyediakan antarmuka untuk membuat dan menjalankan <i>database query</i> , dapat digunakan untuk menjalankan berbagai operasi <i>database</i> di dalam <i>website</i> , serta mendukung berbagai sistem <i>database</i> .
Authentication	Seluruh proses konfigurasi otentikasi yang ada pada Laravel sudah berjalan secara otomatis. Bisa dilihat pada file konfigurasi otentikasi atau 'config/auth.php', di dalamnya terdapat beberapa opsi otentifikasi yang sudah terdokumentasikan dengan baik dan dapat disesuaikan dengan kebutuhan sistem.
Schema Builder	<i>Class Laravel Schema</i> menyediakan <i>database agnostic</i> untuk memanipulasi tabel. <i>Schema</i> ini berjalan baik di berbagai tipe <i>database</i> yang didukung Laravel dan mempunyai API yang sama di seluruh sistem.
Configuration Management Features	Seluruh file konfigurasi Laravel disimpan di dalam direktori config. Setiap opsi didokumentasikan dengan baik sehingga dapat mengubah setiap konfigurasi yang tersedia.
E-mail Class	Laravel menyediakan API beberapa <i>library</i> SwiftMailer yang cukup populer dengan koneksi ke SMTP, Postmark, Mailgun, SparkPost, Amazon SES, dan sendmail yang memungkinkan untuk mengirimkan email dengan cepat melalui aplikasi lokal atau melalui layanan <i>cloud</i> .
Redis	Server struktur data yang dapat menyimpan <i>key</i> dengan tipe <i>strings</i> , <i>hashes</i> , <i>lists</i> , <i>sets</i> , dan <i>sorted set</i> atau dikenal dengan redis merupakan aplikasi <i>open source</i> yang menyimpan <i>key-value</i> , menghubungkan antara sesi yang sudah ada dengan <i>cache general-purpose</i> . Redis terkoneksi dengan session secara langsung.
Event and Command Bus	Laravel <i>Command Bus</i> menyediakan metode pengumpulan tugas yang dibutuhkan aplikasi supaya dapat berjalan secara simpel dan perintah yang mudah dimengerti.

Pada Laravel terdapat *root directory* yang didalamnya terdapat beberapa file, diantaranya:

```
|- app/
  |- Http/
    |- Controllers/
    |- Middleware/
  |- Providers/
  |- Account/
    |- Console/
    |- Exceptions/
    |- Events/
    |- Jobs/
    |- Listeners/
    |- Models/
      |- User.php
      |- Role.php
      |- Permission.php
    |- Repositories/
    |- Presenters/
    |- Transformers/
    |- Validators/
    |- Auth.php
    |- Acl.php
  |- Merchant/
  |- Payment/
  |- Invoice/
|- resources/
|- routes/
```

Sumber: <https://laraveldaily.com/wp-content/uploads/2017/10/Screen-Shot-2017-10-10-at-3.55.01-PM-515x1024.png>

Git ignore	Merupakan file teks yang memberitahu Git file atau folder mana yang harus diabaikan pada suatu proyek, biasanya ditempatkan pada direktori root proyek.
.env	Digunakan untuk mengatur konfigurasi dari <i>environment</i> . Jika tidak ada, dapat menyalin file .env.example lalu diberi nama .env dan pastikan sudah terdapat variabel APP_KEY. Jika belum ada dapat melakukan <i>generate key</i> atau php artisan key:generate
.htaccess	Merupakan file konfigurasi yang digunakan untuk mengubah pengaturan <i>default</i> dari Apache
Artisan	Merupakan perintah-perintah yang dijalankan dalam command line/command prompt, digunakan untuk membuat <i>Controller</i> , <i>Model</i> , <i>Middleware</i> , dsb.
composer.json	Merupakan file yang berisi pendefinisian <i>library (Third Party)</i>

composer.lock	File yang mencatat versi package yang terinstal
composer.phar	Merupakan composer binary. PHAR (PHP ARCHIVE) yaitu format <i>archive</i> untuk PHP yang dijalankan pada baris perintah
index.php	File untuk file index yang berisi script PHP
package.json	File yang berisi keterangan <i>project</i> Javascript.
package-lock.json	File yang menyimpan data npm, dependency tree, dan info update package lainnya
phpunit.xml	Sebagai konfigurasi test
server.php	Menampilkan data dari salah satu server nya, mengetahui alamat IP user, jenis browser dan OS yang digunakan user, alamat url yang diakses oleh user, url referal yang digunakan user untuk mengakses situs
webpack.mix.js	Merupakan titik masuk untuk semua kompilasi asset. Dapat dianggap sebagai pembungkus konfigurasi ringan di sekitar Webpack.
yarn.lock	Memudahkan user dalam menambahkan modul baru. <i>Package management</i> yarn.lock adalah yarn.

2. Syntax

Syntax pada Laravel dapat dilihat dengan mengetikkan atau menulis **php artisan list** pada terminal Visual Studio Code. Terdapat beberapa syntax yang sering digunakan, diantaranya:

php artisan serve	Digunakan untuk menjalankan proyek Laravel di localhost.
php artisan make:migration	Digunakan untuk membuat migrasi.
php artisan make:model	Digunakan untuk membuat modal
php artisan make:controller	Digunakan untuk membuat controller
php artisan migrate	Digunakan untuk mengatur skema atau relasi tabel pada database sesuai dengan <i>folder</i> migration

3. Migration

Migration dapat dikatakan sebagai kontrol versi *database* karena memungkinkan untuk mengubah skema atau relasi *database* aplikasi sesuai dengan apa yang ada di *folder* migration. Untuk membuat *migration* dapat menggunakan command artisan:

```
PS C:\xampp\htdocs\Modul5> php artisan make:migration create_namatabel_table
Created Migration: 2020_11_24_150229_create_namatabel_table
```

Migration akan ditempatkan di direktori **database/migrations** dengan setiap file yang dibuat memiliki *timestamp*. Tampilan dari *file migration* adalah:

```
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateNamatabelTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('namatabel', function (Blueprint $table) {
17             $table->id();
18             $table->timestamps();
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      *
25      * @return void
26      */
27     public function down()
28     {
29         Schema::dropIfExists('namatabel');
30     }
31 }
```

Selain artisan command untuk membuat migration, terdapat beberapa artisan command yang lain terkait dengan *migration*, diantaranya:

php artisan make:migration create_namatabel_table --path	Membuat migration dengan menentukan jalurnya
php artisan schema:dump	Menekan migration menjadi satu file SQL
php artisan migrate	Menjalankan semua migration
php artisan migrate --force	Memaksa menjalankan migration
php artisan migrate:rollback	Mengembalikan sebuah migration atau kumpulan migration yang terakhir dibuat.
php artisan migrate:reset	Mengembalikan semua migration
php artisan migrate:refresh	Mengembalikan semua migration dan

	memuat ulang seluruh migration atau mererefresh kembali database
php artisan migrate:fresh	Menghapus semua tabel dari database dan menjalankan migration

Mohon diperhatikan dalam penamaan migration, harus plural, contohnya untuk database buku, dinamai books, apabila dalam menamakan database di migration menggunakan bahasa selain bahasa inggris, kita bisa melakukan deklarasi nama tabel di model yang kita gunakan.

Penting : Nama Tabel yang disepakati dalam *Laravel Naming Convention* harus berbentuk plural dan menggunakan bahasa inggris, semisalnya anda ingin membuat tabel book dengan migration, maka command yang dijalankan adalah “php artisan make:migration create_**books**_table” (book menjadi books).

Sedangkan untuk penamaan Model, nama yang disepakati adalah berbentuk Singular, sehingga untuk tabel buku tadi, nama kelas model yang dibuat adalah Book, tiap Model dalam laravel mencerminkan atau mewakili nama tabel yang ada di database, dengan pengaturan bawaan sebagai berikut : Jika nama class Model adalah Book, maka tabel yang diwakili oleh class Model tersebut adalah tabel dengan nama books (Nama class model + huruf s plural).

Jika terdapat kasus dimana nama model tidak sesuai dengan diatas (misal nama table di database adalah siswa tanpa tambahan s plural, dan class model yang terbentuk adalah Siswa, tambahkan baris berikut di protected \$table = “siswa”.

```
class Siswa extends Model
{
    protected $table = "siswa";
}
```

4. Seed

Laravel memberikan metode sederhana untuk melakukan seeding atau membuat *data dummy* ke database menggunakan *seed classes*. Semua *seed classes* disimpan di direktori `database/seeder`. Pada folder `seeds` terdapat class `DatabaseSeeder` yang bisa menggunakan metode `call` untuk menjalankan *class seed* yang ada. Untuk membuat seeder dilakukan dengan cara menuliskan command di terminal Visual Studio Code:

```
PS C:\xampp\htdocs\Modul5> php artisan make:seeder NamaSeeder
Seeder created successfully.
```

Hasilnya akan ada di direktori **database/seeder**. Contoh dengan isinya seperti pada gambar berikut:

```
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use Illuminate\Support\Facades\DB;
7  use Illuminate\Support\Str;
8
9  class NamaSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      *
14      * @return void
15      */
16     public function run()
17     {
18         DB::table('produk')->insert([
19             'nama' => Str::random(10),
20             'deskripsi' => 'Ini produk',
21             'harga' => 15000,
22         ]);
23     }
24 }
```

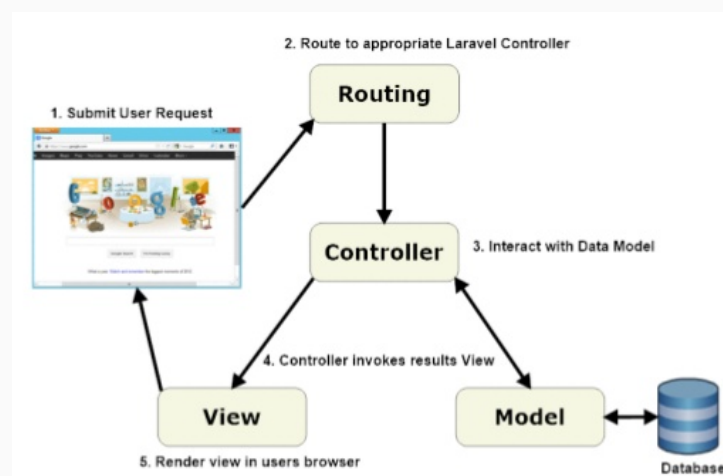
Adapun artisan command yang bisa digunakan terkait dengan seed:

php artisan db:seed --class=NamaSeeder	Menjalankan seeder class tertentu
php artisan migrate:fresh --seed	Menghapus semua tabel dan menjalankan kembali atau membangun kembali database
php artisan db:seed --force	Memaksa menjalankan perintah <i>seeding</i> terhadap database dengan menerima segala resikonya

5. MVC (Model-View-Controller)

MVC (*Model-View-Controller*) merupakan metode yang digunakan dalam membuat aplikasi dengan cara memisahkan pengembangan aplikasi berdasarkan komponen utama yang membangun aplikasi tersebut, seperti manipulasi data, antarmuka pengguna, dan kontrol utama aplikasi.

Proses berjalannya aplikasi pada *framework* Laravel dimulai dengan *user* mengakses *url* pada *browser*, *url* yang diakses akan mencari *route*, jika *url* sama dengan *route* maka *method* yang ada di *controller* akan dieksekusi dan mengambil data dari *model* sesuai dengan *database* yang digunakan, jika *url* tidak ditemukan maka *url* akan menampilkan *error 404 not found*. Setelah *method* dieksekusi oleh *controller* dan mengambil data dari *model*, *controller* akan meneruskan data ke *view* untuk ditampilkan di halaman *browser user*. Proses berjalannya aplikasi dapat dilihat pada gambar berikut:



Sumber: <https://images.app.goo.gl/pMkTeJTFihWx6eHB6>

A. Model

Model adalah bagian pada laravel yang berhubungan langsung dengan *database* yang digunakan untuk memanipulasi data seperti *input*, *update*, *delete*, dll. *Model* tidak bisa berhubungan langsung dengan *view*.

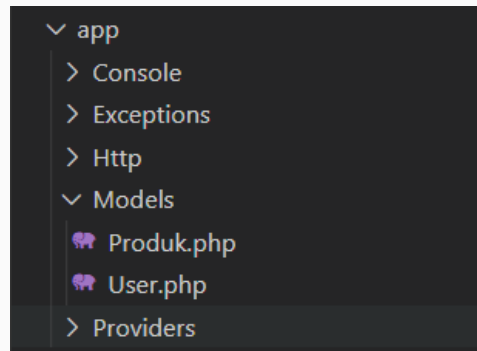
Contoh syntax membuat *model* pada Laravel dengan menjalankan perintah pada *command line* atau terminal yang ada pada Visual Studio Code:

```
C:\xampp\htdocs\Modul5> php artisan make:model Produk -m
```

Jika berhasil, akan muncul konfirmasi seperti pada gambar dibawah ini pada *command line* atau terminal:

```
Model created successfully.
Created Migration: 2020_11_11_162356_create_produk_table
```

Secara otomatis file akan dibuat di *folder app/Models*:



Tampilan dari *file model* yang dibuka akan seperti gambar berikut:

```
app > Models > Produk.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Produk extends Model
9  {
10     use HasFactory;
11 }
12
```

Sebuah *model* merepresentasikan data pada tabel dan relasi antar tabel. Contoh *syntax* pembuatan relasi di *model*:

```
app > Models > Produk.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Produk extends Model
9  {
10     public function kategori (){
11         return $this->hasOne('App\Models\Kategori');
12     }
13
14
15     use HasFactory;
16
17     //model ini mewakili tabel produk
18     protected $table = 'Produk';
19
20 }
21
```

Dari contoh diatas, *code program* terlihat seperti kerangka sebuah *class*, tapi tidak demikian. *Framework* akan mengekstrak sebagian besar logika *model* dan membuat logika *model* menjadi mudah dan lebih rapi.

Laravel mempunyai *Active Record Implementation* atau dapat disebut *Eloquent ORM* (*Object Relational Mapping*). *Eloquent ORM* menawarkan *syntax* yang intuitif dalam melakukan tugas yang berhubungan dengan *database*. Sederhananya, *Eloquent* merupakan sebuah fitur yang ada pada Laravel yang bisa digunakan untuk mengelola data di *database* hanya dengan satu buah *model*, mulai dari *insert*, *update*, *delete* dan *read data* serta memanggil relasi antar tabel.

Terdapat beberapa relasi yang ada pada *eloquent*, yaitu:

1. **Relasi One To One.** Relasi yang paling dasar, contohnya satu produk hanya memiliki satu kategori. Relasi yang ditulis pada model produk yaitu **hasOne**.

```
class Produk extends Model
{
    public function kategori (){
        return $this->hasOne('App\Models\Kategori');
    }
}
```

2. **Relasi Inverse.** Definisikan relasi yang telah dibuat sebelumnya, dimana satu kategori dimiliki banyak produk, relasi yang ditulis pada model kategori yaitu **belongsTo**.

```
class Kategori extends Model
{
    public function produk()
    {
        return $this->belongsTo('App\Models\Produk');
    }
}
```

3. **Relasi One-to-many.** Relasi dimana tabel A bisa memiliki banyak aksi di tabel B. Contohnya, relasi antara satu pembeli dilayani satu kasir dan satu kasir melayani banyak pembeli. Relasi yang ditulis pada model kasir yaitu **hasMany**.

```
class Kasir extends Model
{
    public function Pembeli(){
        return $this->hasMany('App\Models\Pembeli');
    }
    use HasFactory;
}
```

4. **Relasi Many-to-many.** Relasi yang kompleks, contohnya seperti satu produk memiliki satu pembeli dan satu pembeli dimiliki banyak produk. Relasi yang ditulis pada model pembeli yaitu **belongsToMany**.

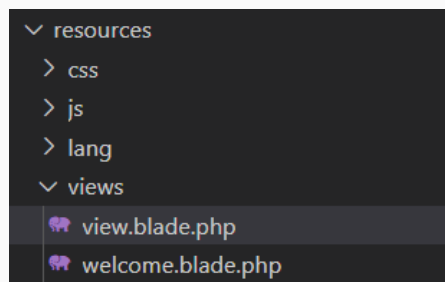
```
class Pembeli extends Authenticatable
{
    public function Produk(){
        return $this->belongsToMany('App\Models\Produk');
    }
}
```

B. View

View merupakan bagian pada Laravel yang menerima dan mempresentasikan data ke pengguna, serta mengatur tampilan aplikasi untuk dilihat oleh pengguna. Biasanya berupa *file template* HTML yang diatur oleh *controller*. *View* tidak bisa mengakses langsung bagian *model*. *Folder views* pada Laravel digunakan sebagai penyimpanan *file-file* PHP untuk keperluan tampilan (*front-end*) aplikasi.

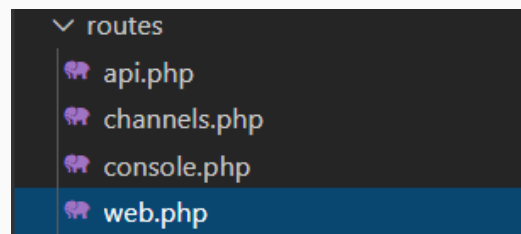
- **Membuat view**

Untuk membuat *view* harus dipastikan *file* yang dibuat dan disimpan pada folder *resources/views* menggunakan ekstensi **.blade.php**. *File* dengan ekstensi ini merupakan isi dari baris kode yang akan digunakan dalam membuat tampilan (*frontend*) seperti *css*, *html*, dll. Contohnya disini membuat *view* bernama 'view'.



- **Memanggil view**

Untuk memanggil *view* dapat dilakukan dengan cara masuk ke direktori *resources/routes*, kemudian pilih *route web.php*.



Contohnya memanggil *blade* 'view' :

```
Route::get('/view', function () {  
    //view disini merupakan nama file view.blade.php  
    return view('view');  
});
```

Pemanggilan ini dilakukan dengan cara menulis 'view' pada *address bar browser* `http://127.0.0.1:8000/view` sehingga tidak perlu menuliskan ekstensi *.blade.php*, cukup dengan menuliskan nama *view* nya saja. Maka tampilan dari 'view' tersebut seperti dibawah ini:



Halo, Selamat Datang di Modul 5!

- **Penggunaan yield, section, show**

@Yield dan **@section** .. **@show** digunakan untuk ditimpa secara opsional setiap kali Anda memperpanjang templat blade. Segala sesuatu yang dapat Anda lakukan dengan **@yield** juga dapat dilakukan dengan **@section** .. **@show** tetapi tidak sebaliknya. Inilah yang mereka lakukan:

@yield ('utama')

Dapat diganti dengan @ bagian ('utama') .. @ bagian

Dapat disediakan dengan string default tetapi tanpa HTML! String default akan ditampilkan di sub-blade-template ketika tidak ada @ bagian ('utama') .. @ bagian disediakan.

@section ('main') .. @show

Dapat diganti dengan @ bagian ('utama') .. @ bagian

Dapat diberikan kode HTML default. Kode HTML default akan ditampilkan di sub-blade-template ketika tidak ada @ bagian ('utama') disediakan.

Dapat diganti dengan **@section ('main') @ parent .. @endsection** dan juga menunjukkan kode HTML default.

Berikut beberapa contoh: **test.blade.php**


```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8">
6     <title>Test</title>
7 </head>
8
9 <body>
10     <h1>This is a test</h1>
11
12     @yield('mainA')
13     @yield('mainB', 'This is the alternative 1')
14     @yield('mainC', '<p>This is the alternative 2</p>')
15     @yield('mainD', 'This is the alternative 3')
16
17     @section('testA')
18     @show
19
20     @section('testB')
21     This is the alternative 4
22     @show
23
24     @section('testC')
25     <p>This is the alternative 5</p>
26     @show
27
28     @section('testD')
29     <p>This is the alternative 6</p>
30     @show
31
32
33 </body>
34
35 </html>
```

di sini ada file lain bernama **testA.blade.php** yang memperluas file berbilang lainnya:

```
1  @extends('test')
2
3  @section('mainD')
4  <div>
5      <p>First replacement!</p>
6      <hr>
7  </div>
8  @endsection
9
10 @section('testC')
11 <div>
12     <p>Second replacement!</p>
13     <hr>
14 </div>
15 @endsection
16
17 @section('testD')
18 @parent
19 <div>
20     <p>Additional content</p>
21     <hr>
22 </div>
23 @endsection
```

C. Controller

Controller merupakan bagian pada Laravel yang mengatur hubungan antara bagian *model* dengan bagian *view*, berfungsi sebagai penerima *request* dan data dari *user* lalu menentukan proses mana yang akan dilakukan aplikasi. Folder *controller* digunakan untuk menyimpan kelas-kelas PHP *controller* yang telah dibuat. *Controller* dapat digunakan untuk memisahkan logika aplikasi ke dalam beberapa kelas PHP.

- **Mengarahkan *routes* ke *controller***

Pengimplementasian *routes* dilakukan untuk menjalankan sebuah *function* di *Controller*, semua proses akan dilakukan di *controller* sesuai dengan kaidah MVC.

Routes

Berfungsi untuk menangani *request* dari aplikasi, lalu mengarahkan aplikasi tersebut ke halaman atau *resource* tertentu untuk dipanggil. Seluruh perintah *routing* berada di dalam *file web.php* yang berada di folder **app/routes/web.php**. Untuk melihat *routing* yang telah didefinisikan dapat membuka *file web.php* atau dengan menggunakan perintah artisan:

```
C:\xampp\htdocs\Modul5> php artisan route:list
```

Hasil dari perintah tersebut adalah sebagai berikut:

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api auth:api
	GET HEAD	history	history	App\Http\Controllers\HistoryController@index	web
	GET HEAD	history/{index}	detailHistory	App\Http\Controllers\HistoryController@detailHistory	web
	GET HEAD	order	order	App\Http\Controllers\OrderController@index	web
	GET HEAD	order/{index}	detailOrder	App\Http\Controllers\OrderController@detailOrder	web
	POST	order/{index}	orderProduct	App\Http\Controllers\OrderController@orderProduct	web
	GET HEAD	product	product	App\Http\Controllers\ProductController@index	web
	GET HEAD	product/add	addIndex	App\Http\Controllers\ProductController@addIndex	web
	POST	product/add	addProduct	App\Http\Controllers\ProductController@addProduct	web
	GET HEAD	product/delete/{index}	deleteProduct	App\Http\Controllers\ProductController@deleteProduct	web
	GET HEAD	product/update/{index}	updateIndex	App\Http\Controllers\ProductController@updateIndex	web
	POST	product/update/{index}	updateProduct	App\Http\Controllers\ProductController@updateProduct	web

Terkadang, kita perlu mengambil URI yang ada di route, misalnya ingin mengambil ID pengguna dari URL. Maka, kita bisa menggunakan parameter pada *route*

```
// Using PHP callable syntax...
Route::get('page/{parameter}', [nameController::class, 'index']);

// Using string syntax...
Route::get('page/{parameter}', 'App\Http\Controllers\nameController@index');
```

route juga bisa tanpa parameter:

```
// Using PHP callable syntax...
Route::get('page', [nameController::class, 'index']);

// Using string syntax...
Route::get('page', 'App\Http\Controllers\nameController@index');
```

Contoh penggunaan dari *route*:

```
// Using PHP callable syntax...
Route::get('/users', [UserController::class, 'index']);

// Using string syntax...
Route::get('/users', 'App\Http\Controllers\UserController@index');
```

Dengan syarat sudah meng-*import* class controller nya.

```
use App\Http\Controllers\nameController;
```

Method routes

Dalam membuat *routing* di Laravel, berbagai macam *http method* dapat didefinisikan. Diantaranya:

- GET
Berfungsi untuk mengambil halaman *view* atau mengirimkan data ke *controller*
- POST
Berfungsi untuk menerima data dari *http body* untuk di proses oleh *controller*
- PUT
Berfungsi untuk menerima data, bertujuan memproses perubahan data dengan logika program diserahkan ke *controller*
- DELETE
Berfungsi menerima data, lalu perintah penghapusan dikirimkan ke *controller*
- PATCH
Memiliki fungsi yang sama dengan *method PUT*

Cara penggunaannya lebih ke fungsi, format kode *method routes* adalah sebagai berikut:

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);
```

HTML *forms* tidak mendukung *method PUT*, *DELETE*, dan *PATCH* sehingga harus melakukan *method spoofing*. *Method spoofing* merupakan penambahan *hidden field* pada form dengan nama `_method`

```
<form action="/foo/bar" method="POST">  
  <input type="hidden" name="_method" value="PUT">  
  <input type="hidden" name="_token" value="{{ csrf_token() }}">  
</form>
```

- Cara membuat **Controller**

Membuat *controller* dapat dilakukan melalui command prompt (CMD) pada direktori *project* Laravel atau melalui terminal Visual Studio Code, lalu menulis perintah artisan:

```
PS C:\xampp\htdocs\Modul5> php artisan make:controller NamaController  
Controller created successfully.
```

Controller akan dibuat secara otomatis dan tampilan dari *controller* adalah sebagai berikut:

```
app > Http > Controllers >>NamaController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class NamaController extends Controller
8  {
9      //
10 }
```

- **Syntax Eloquent pada *controller***

Terdapat beberapa contoh syntax eloquent:

1. **Insert.** Digunakan untuk menambahkan data ke *database*.
2. **Update.** Digunakan untuk mengubah data pada *database*.
3. **Read.** Digunakan untuk menampilkan data dari *database*.
4. **Delete.** Digunakan untuk menghapus data di *database*.

Contoh Syntax Eloquent pada *controller* Pembeli:

```
app > Http > Controllers > > PembeliController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4  use Illuminate\Http\Request;
5  use App\Models\Pembeli;
6
7  class PembeliController extends Controller
8  {
9      public function insert(Request $request)
10     {
11         $Pembeli = new Pembeli();
12         $Pembeli->name = $request->name;
13         $Pembeli->email = $request->email;
14         $Pembeli->password = $request->password;
15         $Pembeli->save();
16     }
17
18     public function updates(Request $request)
19     {
20         $Pembeli = Pembeli::find($request->id);
21         $Pembeli->name = $request->title;
22         $Pembeli->gender = $request->gender;
23         $Pembeli->password = $request->password;
24         $Pembeli->save();
25     }
26
27     public function deletes(Request $request)
28     {
29         $Pembeli = Pembeli::find($request->id);
30         $Pembeli->delete();
31     }
32
33     public function read()
34     {
35         $Pembeli = Pembeli::all();
36         return view('view', ['pembeli'=>$Pembeli]);
37     }
38 }
```

Pada saat membuat controller, pastikan sudah “use” Model nya, contohnya untuk **PembeliController** menggunakan Model **Pembeli**, dan apabila model **Pembeli** belum ada, bisa cek ke sub bab sebelumnya

6. CRUD (Create-Read-Update-Delete)

CRUD merupakan singkatan dari *Create*, *Read*, *Update*, dan *Delete*. Operasi CRUD adalah manipulasi data dasar untuk database. CRUD pada Laravel berbentuk objek yang membantu pengembang aplikasi dari segi mengeksekusi query database. Fungsi ini digunakan untuk menambahkan data, melihat data, menghapus data, serta mengupdate data.

A. Cara melakukan Migration

Buat model terlebih dahulu

```
PS C:\xampp\htdocs\latihan5\latihan5> php artisan make:model student -m
Model created successfully.
Created Migration: 2020_11_11_174434_create_students_table
```

Pada file migration diisi dengan *code* seperti ini:

```
Schema::create('students', function (Blueprint $table) {
    $table->id();
    $table->string('nama');
    $table->string('nim');
    $table->text('alamat');
    $table->timestamps();
});
```

Lalu melakukan migration dengan mengetikkan perintah cmd menggunakan code “php artisan migrate”

```
PS C:\xampp\htdocs\latihan5\latihan5> php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (349.97ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (251.84ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (255.50ms)
Migrating: 2020_11_11_174434_create_students_table
Migrated: 2020_11_11_174434_create_students_table (94.58ms)
```

B. Cara membuat *Create* pada Laravel:

- View

File ini dibuat di direktori resource > views > layouts > addStudent.blade.php

```
<form method="post" enctype="multipart/form-data" action="{{route('addStudent')}}">
    @csrf
    <div class="form-group">
        <label>Nama Mahasiswa</label>
        <input type="text" name="nama">
    </div>
    <div class="form-group">
        <label>NIM</label>
        <input type="text" name="nim">
    </div>
    <div class="form-group">
        <label>Alamat</label>
        <textarea name="alamat"></textarea>
    </div>
    <input type="submit" value="Submit">
</form>
```

- Route

File ini dibuat di direktori routes > web.php

```
Route::post('/student/add',[StudentController::class, 'addStudent'])->name('addStudent');
```

- Controller

File ini dibuat di direktori app > Http > Controllers > StudentController.php

```
public function addStudent(Request $request){
    $student = new Student();
    $student->nama = $request->nama;
    $student->nim = $request->nim;
    $student->alamat = $request->alamat;
    $student->save();

    return redirect(route('student'));
}
```

C. Cara Membuat *Read* pada Laravel:

- View

File ini dibuat di direktori resource > views > layouts > student.blade.php

```
@foreach ($students as $index=> $student)
    <tr>
        <td>{{ $student->nama }}</td>
        <td>{{ $student->nim }}</td>
        <td>{{ $student->alamat }}</td>
        <td>
            <a href="{{ route('updateIndex', $student->id) }}">Edit
            <button class="btn btn-primary"><i class="fa fa-pencil"></button>
            </a>

            <a href="{{ route('deleteStudent', $student->id) }}">Delete
            <button class="btn btn-danger"><i class="fa fa-trash"></button>
            </a>
        </td>
    </tr>
</foreach>
```

- **Route**

File ini dibuat di direktori routes > web.php

```
Route::get('/student', [StudentController::class, 'index'])->name('student');
```

- **Controller**

File ini dibuat di direktori app > Http > Controllers > StudentController.php

```
public function index(){
    $students = Student::all();
    return view('student',compact('students'));
}
```

D. Cara Membuat *Update* pada Laravel

- **View**

File ini dibuat di direktori resource > views > layouts > updateStudent.blade.php

```
<form method="post" enctype="multipart/form-data" action="{{ route('updateStudent', ['index'=> $students->id]) }}>
    <div class="form-group">
        <label>Nama Mahasiswa</label>
        <input type="text" name="nama">
    </div>
    <div class="form-group">
        <label>NIM</label>
        <input type="text" name="nim">
    </div>
    <div class="form-group">
        <label>Alamat</label>
        <textarea name="alamat"></textarea>
    </div>
    @csrf
    <input type="submit" value="Submit">
</form>
```

- **Controller**

File ini dibuat di direktori app > Http > Controllers > StudentController.php


```
public function updateStudent($index, Request $request){  
    $student = Student::all();  
    $student = $student->find($index);  
    $student->nama = $request->nama;  
    $student->nim = $request->nim;  
    $student->alamat = $request->alamat;  
    $student->save();  
  
    return redirect(route('student'));  
}
```

E. Cara membuat *Delete* pada Laravel Controller:

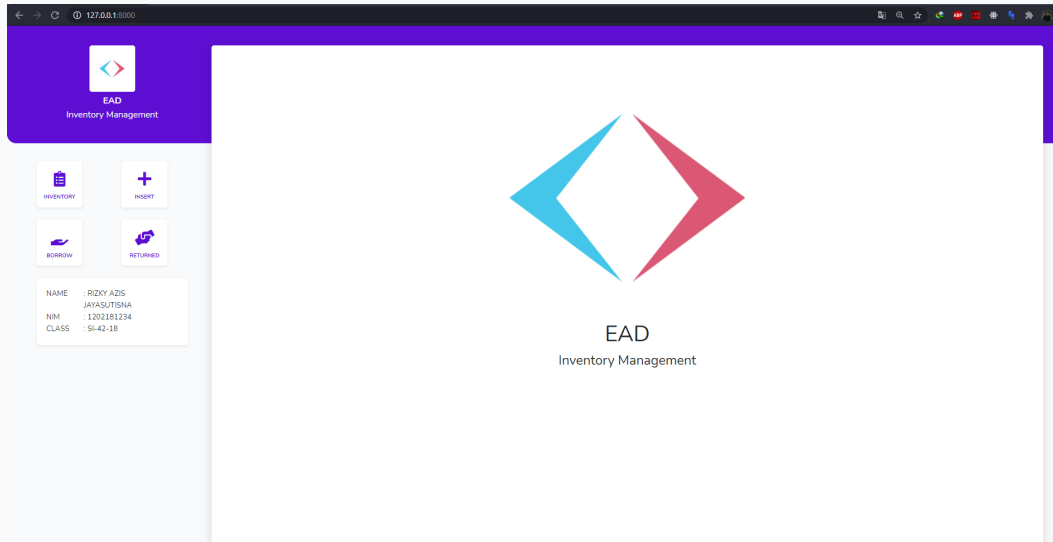
File ini dibuat di direktori app > Http > Controllers > StudentController.php

```
public function deleteStudent($index){  
    $student = Student::find($index);  
    $student->delete();  
  
    return redirect(route('student'));  
}
```


-Pemanggilan template (resources/views/index.blade.php)

```
1  {{-- Menunjukkan bahwa akan mengambil template dari sidebar.blade.php --}}
2  @extends('layouts/sidebar')
3
4  {{-- pengisian dalam template dimulai @section dan nama yieldnya --}}
5  @section('content')
6  <div class="text-center">
7      
8      <h1>EAD</h1>
9      <h4>Inventory Management</h4>
10 </div>
11 @endsection
12 {{-- pengisian template ditutup dengan @endsection --}}
13
```

- Hasil



2. Pembuatan model (app/models/inventory.php)

```
<?php

namespace App\Models;

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

class Inventory extends Model
{
    use HasFactory;

    //menunjukkan model ini mewaliki tabel inventories
    protected $tabel = 'inventories';

    //menunjukkan bahwa 3 kolom ini yang dapat diisi
    protected $fillable = [
        'name', 'photo', 'quantity'
    ];
}
```

3. Pembuatan migration

- table inventories

(database/migrations/####_##_#####_create_invetories_table.php)

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateInventoriesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('inventories', function (Blueprint $table) {
            $table->id();
            $table->string('photo');
            $table->string('name');
            $table->integer('quantity');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('inventories');
    }
}
```

4. Pembuatan fungsi create -routes (routes/web.php)

```
//menampilkan view add inventory melalui InventoryController fungsi addIndex
Route::get('/add',[InventoryController::class, 'addIndex']->name('inventory.add.index'));

//menjalankan add inventory melalui InventoryController fungsi add
Route::post('/add',[InventoryController::class, 'add']->name('inventory.add'));
```

-controller (app/Http/Controllers/InventoryController.php)

```
public function updateIndex($id){
    //mencari data dalam tabel inventory dengan id = $id
    $inventory = Inventory::find($id);

    //mengembalikan view inventory-update.blade.php dengan membawa data inventory yang telah didapatkan
    return view('inventory-update',compact('inventory'));
}

public function add(Request $request){
    //set file dengan nama baru
    $photo = time().'.'.$request->photo->extension();

    //memindahkan file kedalam folder public/images/upload dengan nama yang sudah di set
    $request->photo->move(public_path('images/upload'), $photo);

    //membuat data baru pada tabel inventory
    $inventory = new Inventory();

    //pengisian data pada setiap kolom
    $inventory->name = $request->name;
    $inventory->quantity = $request->quantity;
    $inventory->photo = $photo;

    //menyimpan data
    $inventory->save();

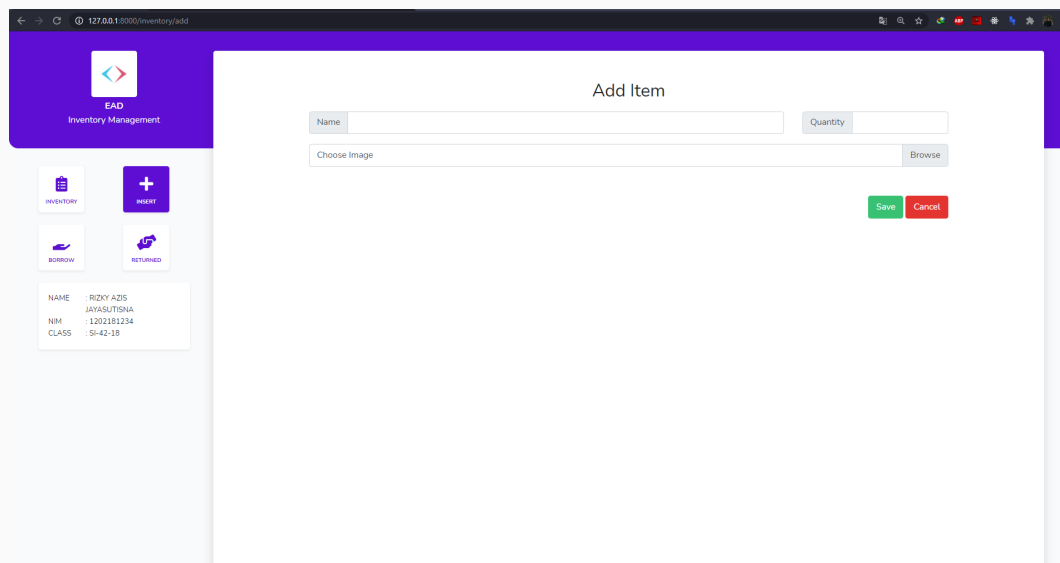
    //mengalihkan kepada route dengan nama inventory.index
    return redirect(route('inventory.index'));
}
```

-view (resources/views/inventory-add.php)

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

@section('content')
<h2 class="text-center">
    Add Item
</h2>
<div class="container mt-3 pl-5 pr-5">
    <form action="{{route('inventory.add')}}" method="post" enctype="multipart/form-data">
        @csrf
        <div class="row">
            <div class="col-9">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Name</span>
                    </div>
                    <input type="text" name="name" class="form-control" >
                </div>
            </div>
            <div class="col-3">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Quantity</span>
                    </div>
                    <input type="number" name="quantity" class="form-control" >
                </div>
            </div>
        </div>
        <div class="input-group">
            <div class="custom-file">
                <input type="file" name="photo" class="custom-file-input" id="inputGroupFile" aria-describedby="inputGroupFileAddon">
                <label class="custom-file-label" for="inputGroupFile">Choose Image</label>
            </div>
        </div>
        <div class="pt-5 text-right">
            <button type="submit" class="btn btn-success">
                Save
            </button>
            <a href="{{route('inventory.index')}}" class="btn btn-danger">Cancel</a>
        </div>
    </form>
</div>
@endsection
```

- Hasil



5. Pembuatan fungsi read

-routes (routes/web.php)

```
//menampilkan view inventory index melalui InventoryController fungsi index
Route::get('/',[InventoryController::class, 'index']->name('inventory.index'));
```

-controller (app/Http/Controllers/InventoryController.php)

```
public function index(){
    //mengambil semua data dari tabel Inventory melalui model
    $inventory = Inventory::all();

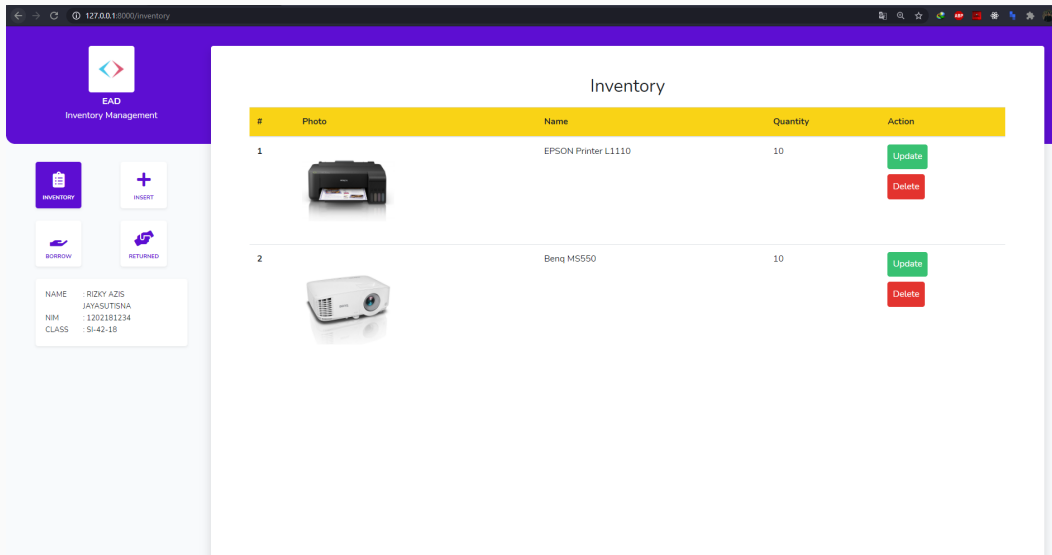
    //mengembalikan view inventory.blade.php dengan membawa data inventory yang telah didapatkan
    return view('inventory',compact('inventory'));
}
```

-view (resources/views/inventory.php)

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

@section('content')
<h2 class="text-center">
    Add Item
</h2>
<div class="container mt-3 pl-5 pr-5">
    <form action="{{route('inventory.update',$inventory->id)}}" method="post" enctype="multipart/form-data">
        @csrf
        <div class="row">
            <div class="col-9">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Name</span>
                    </div>
                    <input value="{{ $inventory->name }}" type="text" name="name" class="form-control" >
                </div>
            </div>
            <div class="col-3">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Quantity</span>
                    </div>
                    <input value="{{ $inventory->quantity }}" type="number" name="quantity" class="form-control" >
                </div>
            </div>
        </div>
        <div class="input-group">
            <div class="custom-file">
                <input type="hidden" value="{{ $inventory->photo }}" name="photo">
                <input type="file" name="photoChange" class="custom-file-input" id="inputGroupFile" aria-describedby="inputGroupFileAddon">
                <label class="custom-file-label" for="inputGroupFile">Choose Image</label>
            </div>
        </div>
        <div class="pt-5 text-right">
            <button type="submit" class="btn btn-success">
                Save
            </button>
            <a href="{{route('inventory.index')}}" class="btn btn-danger">Cancel</a>
        </div>
    </form>
</div>
@endsection
```

- Hasil



6. Pembuatan fungsi update -routes (routes/web.php)

```
//menampilkan view update melalui InventoryController fungsi updateIndex
Route::get('/update/{id}',[InventoryController::class, 'updateIndex'])->name('inventory.update.index');

//menjalankan update inventory melalui InventoryController fungsi update
Route::post('/update/{id}',[InventoryController::class, 'update'])->name('inventory.update');
```

-controller (app/Http/Controllers/InventoryController.php)

```
public function updateIndex($id){
    //mencari data dalam tabel inventory dengan id = $id
    $inventory = Inventory::find($id);

    //mengembalikan view inventory-update.blade.php dengan membawa data inventory yang telah didapatkan
    return view('inventory-update',compact('inventory'));
}

public function update($id,Request $request){
    //mencari data dalam tabel inventory dengan id = $id
    $inventory = Inventory::find($id);

    //pengisian data pada setiap kolom
    $inventory->name = $request->name;
    $inventory->quantity = $request->quantity;

    //mengecek apakah input file telah di isi
    if ($request->photoChange == null){
        //ketika tidak di isi mengeset dengan value yang lama
        $inventory->photo = $request->photo;
    }else{
        //set file dengan nama baru
        $photo = time().'.'.$request->photoChange->extension();
        //memindahkan file kedalam folder public/images/upload dengan nama yang sudah di set
        $request->photoChange->move(public_path('images/upload'), $photo);
        //pengisian data pada tabel
        $inventory->photo = $photo;
    }

    //menyimpan data
    $inventory->save();

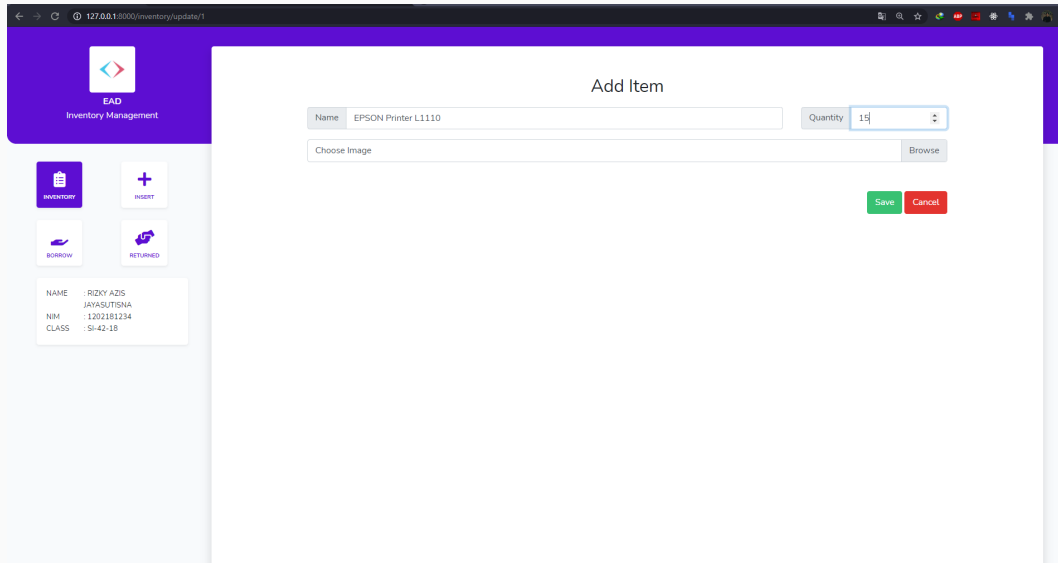
    //mengalihkan kepada route dengan nama inventory.index
    return redirect(route('inventory.index'));
}
```

-view (resources/views/inventory-update.php)

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

@section('content')
<h2 class="text-center">
    Add Item
</h2>
<div class="container mt-3 pl-5 pr-5">
    <form action="{{route('inventory.update',$inventory->id)}}" method="post" enctype="multipart/form-data">
        @csrf
        <div class="row">
            <div class="col-9">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Name</span>
                    </div>
                    <input value="{{ $inventory->name }}" type="text" name="name" class="form-control" >
                </div>
            </div>
            <div class="col-3">
                <div class="input-group mb-3">
                    <div class="input-group-prepend">
                        <span class="input-group-text">Quantity</span>
                    </div>
                    <input value="{{ $inventory->quantity }}" type="number" name="quantity" class="form-control" >
                </div>
            </div>
        </div>
        <div class="input-group">
            <div class="custom-file">
                <input type="hidden" value="{{ $inventory->photo }}" name="photo">
                <input type="file" name="photoChange" class="custom-file-input" id="inputGroupFile" aria-describedby="inputGroupFileAddon">
                <label class="custom-file-label" for="inputGroupFile">Choose Image</label>
            </div>
        </div>
        <div class="pt-5 text-right">
            <button type="submit" class="btn btn-success">
                Save
            </button>
            <a href="{{route('inventory.index')}}" class="btn btn-danger">Cancel</a>
        </div>
    </form>
</div>
@endsection
```

-Hasil



7. Pembuatan fungsi delete

-routes (routes/web.php)

```
//menjalankan delete inventory melalui InventoryController fungsi delete
Route::post('/delete',[InventoryController::class, 'delete']->name('inventory.delete'));
```

-controller (app/Http/Controllers/InventoryController.php)

```
public function delete(Request $request){
    //mencari data dalam tabel inventory dengan id = $id
    $inventory = Inventory::find($request->id);

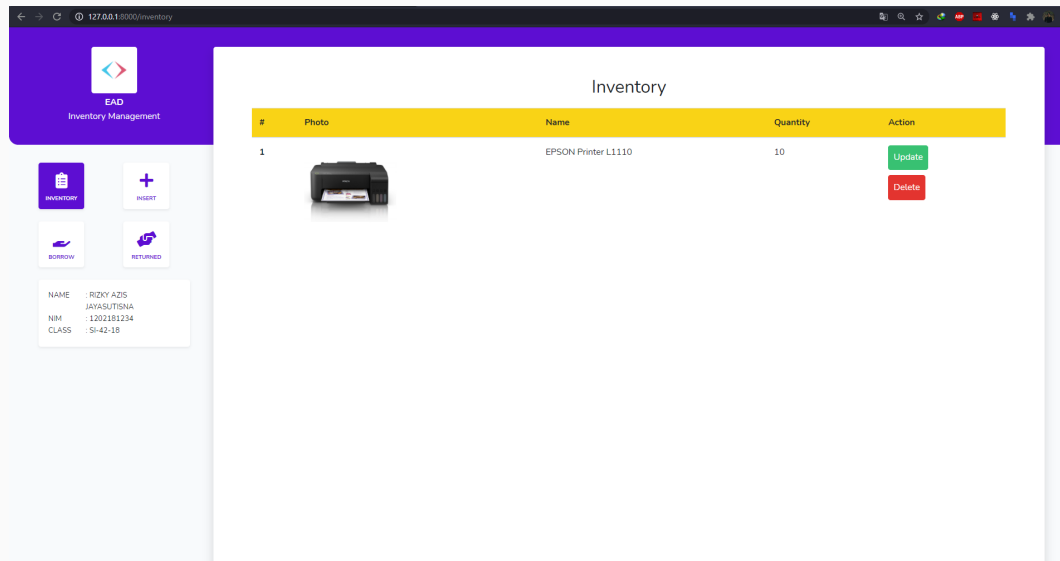
    //menghapus data yang berhasil dicari
    $inventory->delete();

    //mengalihkan kepada route dengan nama inventory.index
    return redirect(route('inventory.index'));
}
```

-view (resources/views/inventory.php)

```
<form action="{{route('inventory.delete')}}" class="mt-2" method="post">
    @csrf
    <input type="hidden" value="{{ $item->id }}" name="id">
    <button class="btn btn-danger p-2">Delete</button>
</form>
```


- Hasil



8. Hasil akhir file routes (routes/web.php)

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

//membuat grouping untuk semua route inventory
Route::prefix('inventory')->group(function(){

    //menampilkan view inventory index melalui InventoryController fungsi index
    Route::get('/',[InventoryController::class, 'index'])->name('inventory.index');

    //menampilkan view add inventory melalui InventoryController fungsi addIndex
    Route::get('/add',[InventoryController::class, 'addIndex'])->name('inventory.add.index');

    //menjalankan add inventory melalui InventoryController fungsi add
    Route::post('/add',[InventoryController::class, 'add'])->name('inventory.add');

    //menampilkan view update melalui InventoryController fungsi updateIndex
    Route::get('/update/{id}',[InventoryController::class, 'updateIndex'])->name('inventory.update.index');

    //menjalankan update inventory melalui InventoryController fungsi update
    Route::post('/update/{id}',[InventoryController::class, 'update'])->name('inventory.update');

    //menjalankan delete inventory melalui InventoryController fungsi delete
    Route::post('/delete',[InventoryController::class, 'delete'])->name('inventory.delete');
});
```

DAFTAR PUSTAKA

1. Modul Praktikum Web Application Development, (2017). Bandung: Laboratorium EAD
2. Laravel. Retrieved from Laravel: <https://laravel.com/>
3. <https://www.it-swarm-id.com/id/laravel/laravel-perbedaan-antara-yield-dan-section/1052792436/>