MODUL PRAKTIKUM

PEMROGRAMAN FRAMEWORK

Oleh

Rahmat Robi Waliyansyah, M.Kom.

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK DAN INFORMARIKA

UNIVERSITAS PGRI SEMARANG

DAFTAR ISI

MODUL I
MENGENAL DAN INSTALASI LARAVEL
(Pertemuan 1)

Tujuan :

1. Mahasiswa dapat mengetahui framework php
2. Mahasiswa dapat mengetahui *software* pendukung dalam menggunakan framework php
3. Mahasiwa dapat memahami proses instalasi framework php
4. Mahasiswa dapat memahami Konsep Model, View dan Controller (MVC)
5. Mahasiswa dapat memahami struktur folder dalam framework php

DASAR TEORI

Laravel merupakan salah satu dari sekian banyak framework PHP yang dapat digunakan secara gratis. Laravel dikembangkan oleh programmer asal amerika yang bernama Taylor Otwell pada tahun 2011. Framework sendiri dapat diartikan sebagai kumpulan kode-kode program yang akan selalu digunakan pada setiap pembuatan aplikasi. Karena selalu digunakan maka kode-kode tersebut dikumpulkan dan disusun secara rapi pada folder-folder agar mudah digunakan dan jadilah sebuah framework.

SOFTWARE PENDUKUNG LARAVEL

1. **TEXT EDITOR**

   Text editor menjadi kebutuhan wajib yang harus dimiliki untuk menulis suatu program. Beberapa text editr yang biasa digunakan oleh para programmer di antaranya Notepad++, Sublime Text, ATOM dan sebagainya, namun pada modul ini akan menggunakan text editor Sublime Text.

2. **WEB SERVER**

   Web server sebagai penyedia layanan web pada komputer lokal. Laravel mendukung web server Apache maupun Ngix. Pada modul ini akan menggunakan web server XAMPP yang didalamnya menggunakan server Apache. Paket aplikasi ini dapat diunduh dari website apachefriends.org. adapun versi yang digunakan pada modul ini adalah versi 3.2.2 yang telah mendukung PHP dengan versi 5.6.30 untuk menjalankan laravel versi 5.4.

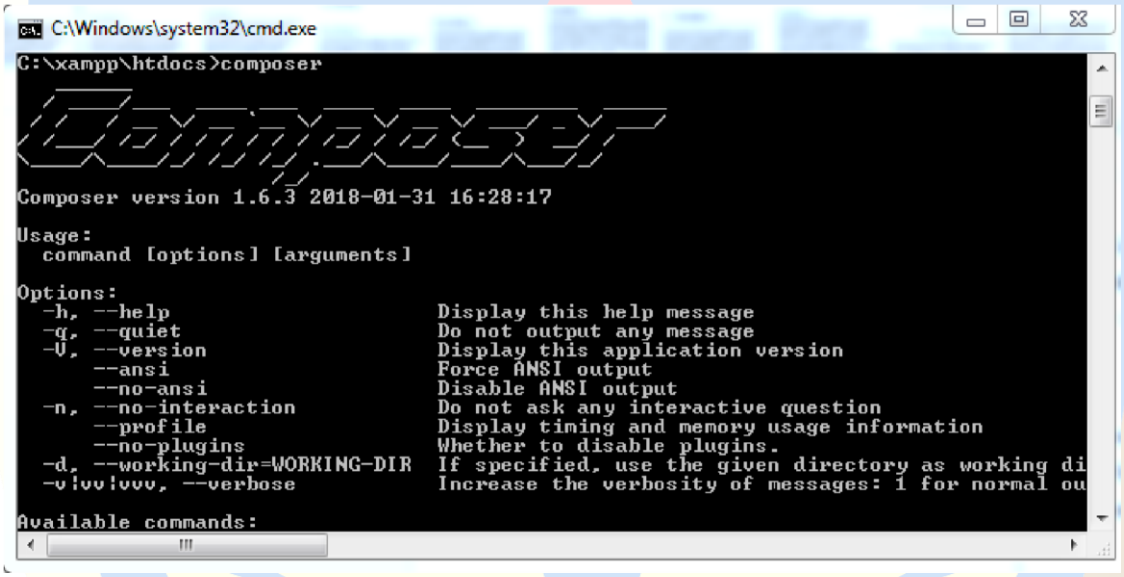
3. **COMPOSER**

   Composer digunakan untuk memudahkan instalasi Laravel dan mendapatkan library yang dibutuhkan dari internet. Jadi sebelum melakukan instalasi Laravel, komputer atau laptop atau

mesin yang digunakan harus sudah terinstal Composer. Aplikasi ini dapat diunduh dari website getcomposer.org.

KEGIATAN PRAKTIKUM INSTALASI LARAVEL

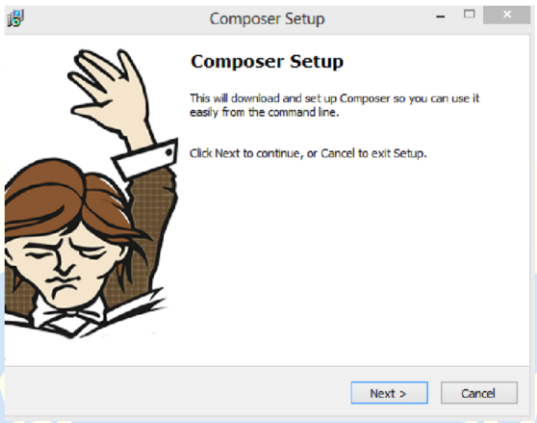
Pada dasarnya ada dua cara untuk melakukan instalasi projek laravel, yaitu menggunakan *laravel installer* atau menggunakan via *composer create project*. Namun cara kedua yang paling umum digunakan. Sebelum memulai melakukan instalasi projek Laravel alangkah baiknya kita memeriksa terlebih dahulu apakah mesin atau komputer yang kita gunakan sudah terinstal composer atau belum dengan cara membuka comand prompt atau cmd dan mengetikan **composer** dan menekan enter. Jika muncul tulisan composer seperti pada gambar dibawah maka composer sudah terinstal.



Gambar 1. 1 Pengecekan Composer

Jika tulisan **composer** tidak muncul maka ikutilah langkah-langkah berikut untuk menginstal composer:

1. Unduh composer di https://getcomposer.org/ , pada menu windows installer download composersetup.exe
2. Klik dua kali tersebut sehingga muncul dialog setup install composer kemudian klik next

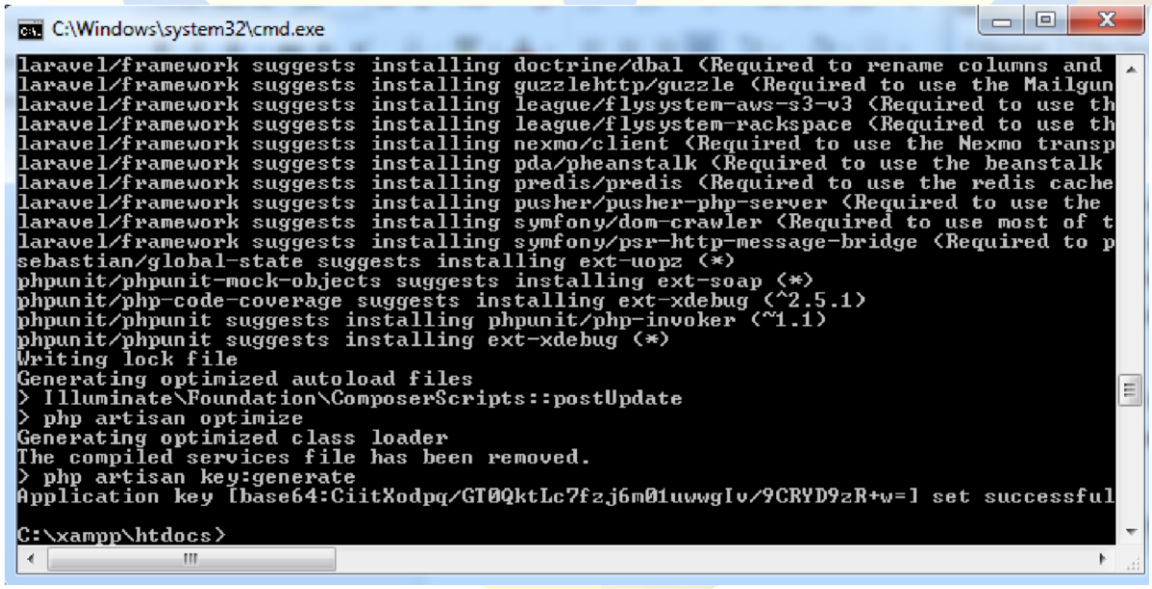


3. Cek versi php. Pada tombol “browse”, kemudian masukkan path php yang sudah diinstal di komputer. disini dicontohkan path php yaitu “C:/xampp/php.exe” kemudian klik next
4. Ikuti proses instalasi, jika sudah berhasil menginstal composer, untuk mengecek composer sudah berjalan dengan baik lakukan kembali langkah sebelumnya yaitu masuk ke comand prompt lalu ketikan **composer** dan kemudian tekan enter.

Untuk memulai menginstal Laravel silahkan masuk ke dalam cmd atau comand prompt (cmd) pada Windows atau terminal pada Linux. Lalu arahkan ke dalam folder htdocs pada xamp. Selanjutnya ketikan script berikut pada cmd atau terminal:

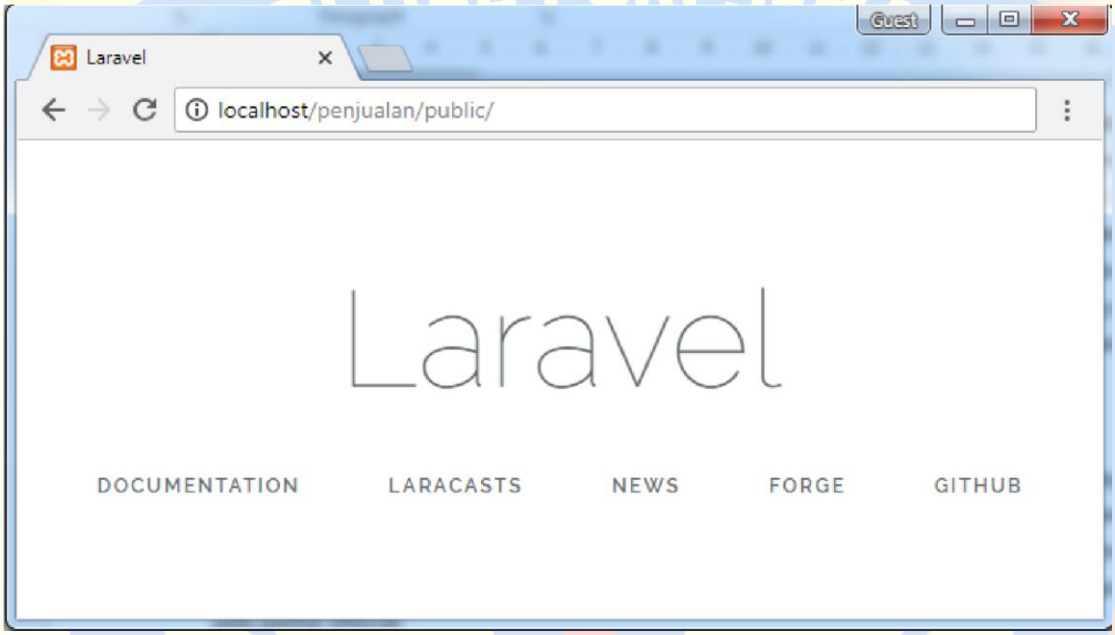
```
composer create-project --prefer-dist laravel/laravel penjualan "5.4.*"
```

Kata penjualan yang ada pada script dapat akan menjadi nama folder sekaligus nama projek laravel nantinya. Proses instalasi laravel dapat dilihat pada gambar dibawah.

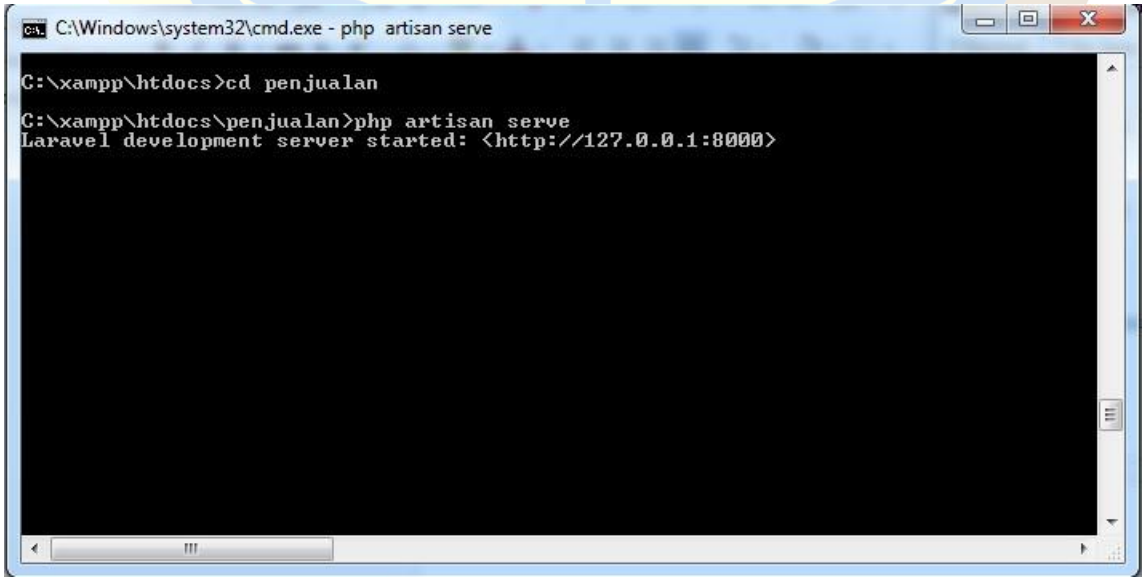


Jika proses instalasi sudah selesai maka pada folder htdocs yang ada pada XAMPP akan ada satu folder baru yang bernama penjualan atau nama projek yang ditulis pada saat menjalankan script

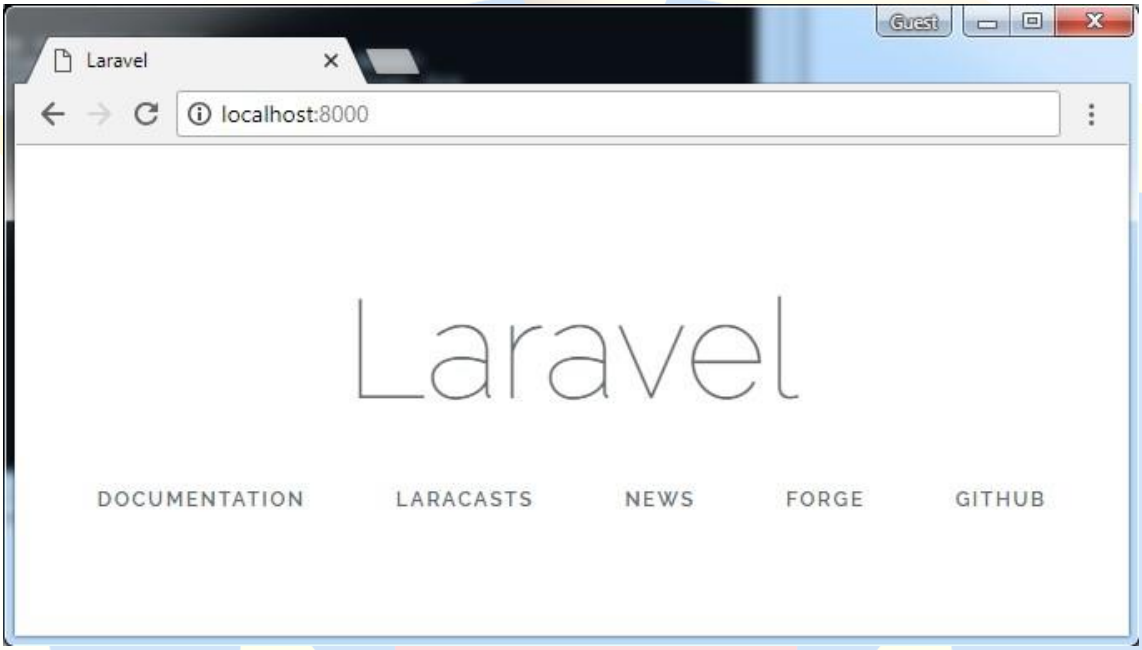
composer create project, folder tersebut merupakan hasil dari script istalasi laravel. Untuk menjalan projek laravel tersebut terdapat dua cara yaitu menggunakan server dari laravel sendiri atau menggunakan server dari xampp. Untuk menjalankan projek laravel dari server XAMPP harus menuliskan alamat seperti berikut localhost/nama_projek/public/ pada link browser yang digunakan. Namun sebelum menjalankannya pastikan terlebih dahulu xampp sudah dalan keadaan aktif dan apache dan mysql sudah running. untuk contoh dari projek laravel yang telah dibuat bisa dilihat pada gambar dibawah.



Atau untuk menjalankan projek laravel bisa menggunakan server dari laravel itu sendiri yang bernama **server artisan** yang bisa dijalankan melalui cmd atau terminal. Untuk menjalankan server artisan pastikan cmd atau terminal sudah berada di dalam folder projek laravel yang telah dibuat, setelah itu untuk menjalankan server dari laravel ketikan script "*php artisan serve"* pada cmd atau terminal seperti pada gambar dibawah.

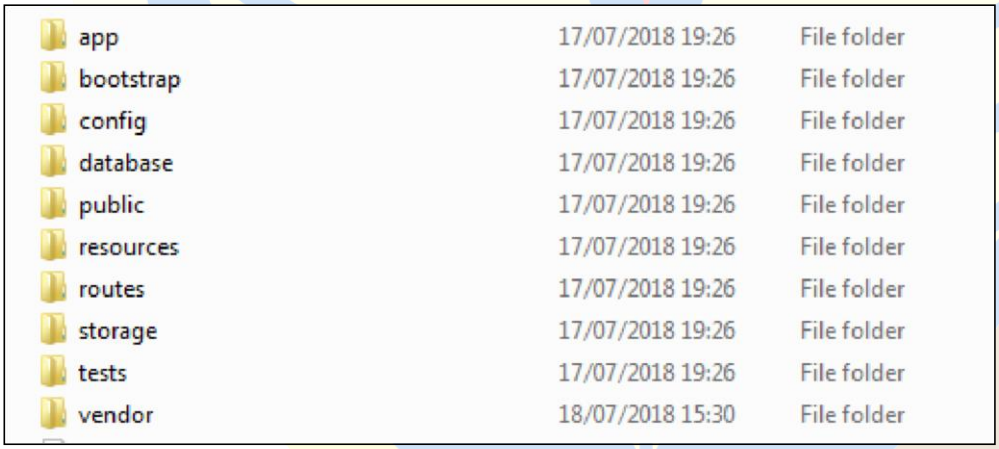


Pada gambar comand prompt diatas terdapat satu baru yang bertuliskan **cd penjualan**, sintak ini adalah untuk membuat comaand prompt masuk ke dalam satu folder yaitu folder penjualan. Setelah menuliskan sintak **php artisan serve** maka projek laravel bisa diakses dengan mengetikan "localhost:8000' pada link browser yang digunakan seperti gambar 1.3 dibawah:



MEMAHAMI STRUKTUR FOLDER LARAVEL

Pada folder hasil instal Laravel terdapat beberapa folder yang penting untuk kita ketahui. Struktur folder tersebut dapat dilihat pada gambar 1.3 dibawah.

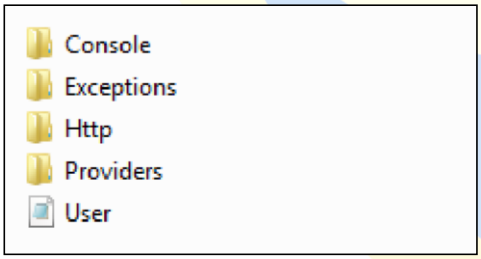


Pada gambar diatas tampak 10 folder yang ada di dalam projek laravel yang telah dibuat. Fungsi folder-folder tersebut yaitu sebabagi berikut:

1. Folder **app** merupakan folder yang paling banyak mendapatkan perhatian karena hampir semua script aplikasi yang kita buat ditaruh pada folder ini. Di dalam folder ini berisi banyak folder yang akan dibahas pada modul-modul selanjutnya.

2. Folder **bootstrap** merupakan folder yang berisi file app.php yang mengendalikan framework Laravel dan file autoload.php yang mengkonfigurasikan autoloading. Folder ini juga berisi folder cache yang berisi file-file cache untuk meningkatkan kecepatan aplikasi.
3. Folder **config** merupakan folder yang berisi file-file konfigurasi aplikasi. Sebaiknya kita memahami setiap file yang ada di dalam folder ini beserta pengaturan-pengaturan yang harus diberikan di dalamnya.
4. Folder **database** merupakan folder yang berisi database migration dan seeds. Migration dan seeds akan dibahas pada modul akan dibahas pada modul-modul selanjutnya.
5. Folder **public** folder public merupakan folder yang berisi file index.php yang merupakan file utama sebagai pintu masuk semua request atau permintaan pada aplikasi yang dibangun. Folder ini juga tempat menyimpan semua aset aplikasi seperti gambar dan file javascript atau css external.
6. Folder **resources** merupakan folder yang berisi file-file aset yang berlum dikompilasi seperti file LESS, SASS atau javascript. Folder ini juga sebagai tempat semua file bahasa.
7. Folder **routes** merupakan folder yang berisi semua route yang kita definisikan pada aplikasi. Untuk materi route akan dibahas lebih detail pada modul selanjutnya.
8. Folder **storage** merupakan folder yang berisi file file yang dibuat oleh framework. Folder ini berisi tiga folder di dalamnya, yaitu app, freamework dan logs. Folder app digunakan untuk menyimpan file yang dibuat oleh aplikasi, folder framework digunakan untuk menyimpan file yang dibuat oleh framework. Sedangkan folder logs digunakan untuk menyimpan file logs.
9. Folder **tests** merupakan folder-folder yang berisi file-file pengetesan.
10. Folder **vendor** merupakan folder yang berisi file-file dependency yang diperoleh dari composer.

MEMAHAMI STRUKTUR FOLDER APP



Seperti telah dijelaskan sebelumnya, ketika membuat suatu aplikasi menggunakan Laravel, maka kita akan banyak bekerja pada folder **app.** Untuk itu kita perlu memahami lebih detail isi dari folder **app**. Secara default folder ini hanya berisi empat folder seperti yang terlihat pada gambar 1.4 diatas.

Sebenarnya ada beberapa folder lagi yang terdapat dalam folder app, namun folder-folder tersebut tidak tampil secara default. Folder tersebut akan ada ketika kita menjalankan perintah artisan *make*. Mengenai perintah-perintah atisan akan dibahas pada modul-modul selanjutnya.

Pada gambar 1.4 terlihat 4 folder. Fungsi dari folder-folder tersebut yaitu sebagai berikut:

1. Folder **Console** merupakan folder yang berisi perintah artisan custom untuk aplikasi yang akan dibuat.
2. Folder **Exception** merupakan folder yang berisi exception handler dari aplikasi yang kita buat.
3. Folder **Http** merupakan folder yang berisi *controller*, *middleware* dan *form request*.
4. Folder **Providers** merupakan folder yang berisi semua *service providers* untuk aplikasi yang akan dibuat.

Selain empat folder diatas, sebenarnya ada beberapa folder lagi yang hanya ada jika kita menjalankan perintah artisan yang ada pada tanda kurng, yaitu, Event (make:event), Jobs (make:job), Listeners (make:listener), Mail (make:mail), notifications (make:notification) dan policies (make:policy).

TUGAS PRAKTIKUM

1. Lakukan proses instalasi framework
2. Buatlah projek pertama laravel dan tampilkan dalam browser

MODUL II

FITUR PADA LARAVEL

(Pertemuan 2)

Tujuan :

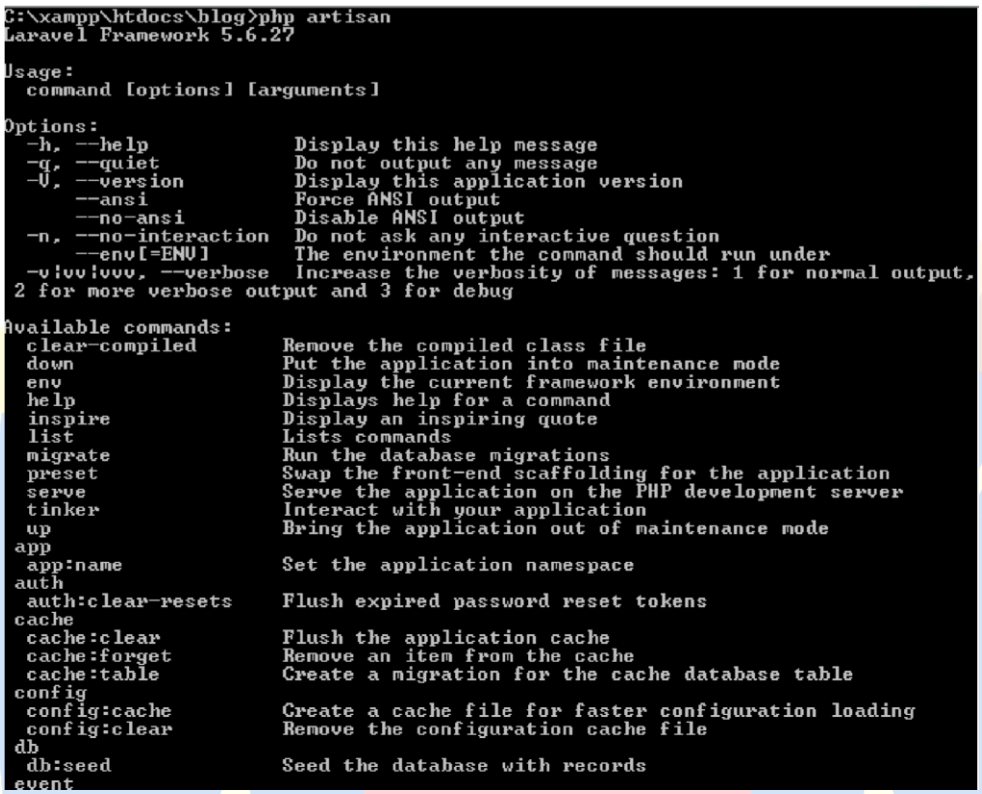
1. Mahasiswa dapat mengetahui fitur dalam framework php
2. Mahasiswa dapat memahami perintah *artisan* pada framework php
3. Mahasiswa dapat memahami perintah *migration* pada framework php
4. Mahasiswa dapat memahami perintah *model* pada framework php
5. Mahasiswa dapat memahami perintah *seeder* pada framework php

DASAR TEORI

Artisan merupakan *comand-line interface* atau perintah-perintah yang diketikan pada command prompt untuk melakukan tugas tertentu saat proses pembuatan suatu apliasi pada Laravel. Adapun cara untuk menggunakan perintah artisan adalah masuk ke dalam command prompt atau cmd pada windows atau terminal pada linux, lalu arahkan ke dalam folder projek laravel yang telah dibuat. Selanjutnya kita dapat mengetikan perintah-perintah artisan.

ARTISAN

Untuk melihat perintah-perintah artisan yang dapat digunakan, kita dapat mengetikan perintah `php artisan list` pada cmd atau command promt, maka akan muncul list dari perintah-perintah artisan yang disediakan oleh laravel seperti pada gambar dibawah:



Pada gambar di atas dapat dilihat sebagian perintah dari perintah-perintah yang disediakan oleh artisan berikut dengan penjelasnya. Kita dapat menampilkan penjelasan yang lebih detail dari sebuah perintah artisan dengan mengetikan perintah dalam format php artisan help (nama perintah) yang pada contohnya nyatanya untuk melihat penjelasan dari perintah migrate yang ada pada perintah artisan , maka perintahnya adalah php artisan help migrate.

KEGIATAN PRAKTIKUM MEMBUAT MIGRATION

Migration dapat dikatan sebagai versi control untuk skema database pada sebuah aplikasi. Dengan bantuan migration, skema database dapat diperbaharui sehingga tetap up to date. Migration juga memungkinkan kita tidak perlu membuka database client seperti PHPMyadmin untuk mengubah struktur database. Cukup mengubah kode pada migation lalu menjalankan perintah artisan, maka skema database akan secara otomatis akan berubah. Kelebihan lain dari fitur ini adalah ketika ingi pindah database, misalnya dari Mysql ke Sqllite, kita tidak perlu membuat struktur database dari awal namun hanya perlu mengubah konfigurasi database Laravel, lalu menjalan perintah artisan migration.

Untuk dapat mempraktekan fitur migration pada Laravel, terlebih dahulu kita harus membuat sebuah database melalui PHPMyAdmin misalnya dalam contoh ini database yang bernama "**penjualan**"**.** Selanjutnya ubah konfigurasi database dalam projek laravel pada file **.env** yang berada pada folder **penjualan/.env** menjadi seperti berikut:

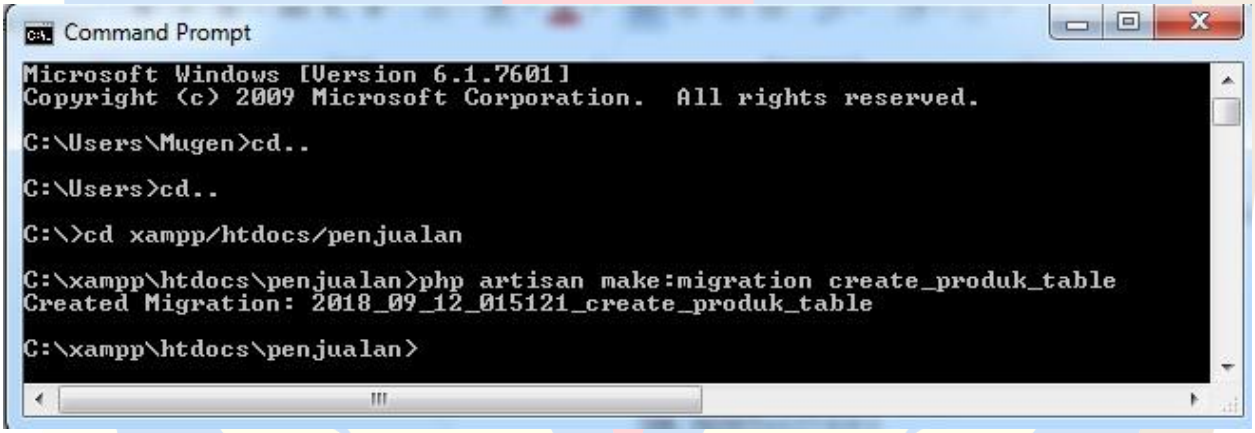
```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=penjualan
DB_USERNAME=root DB_PASSWORD=
```

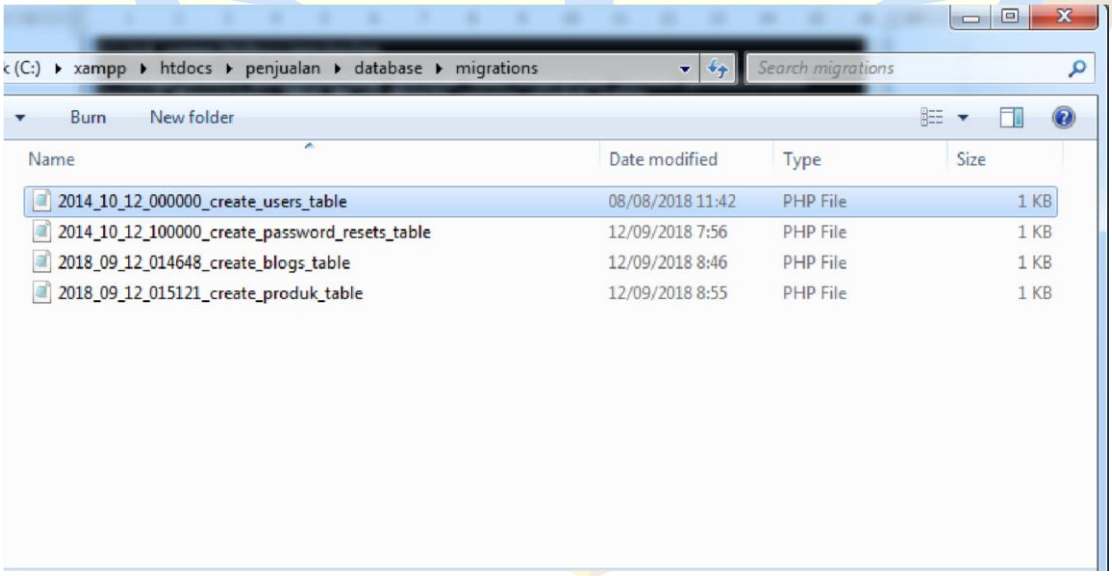
MEMBUAT TABEL

Untuk membuat migration, kita menggunakan artisan, jadi silahkan buka comand prompt dan arahkan ke folder Laravel yang telah diinstal yang dalam contoh pada modul ini adalah projek Laravel dengan nama penjualan yang telah dibuat pada pertemuan sebelumnya, lalu ketikan script berikut:

```
php artisan make:migration create_produk_table
```



Hasil dari perintah artisan tersebut adalah akan ada file migration baru denagn nama **2018_09_12_015121_create_produk_table** pada folder database/migration sebagai berikut:



Selanjutnya, buka file **2018_09_12_015121_create_produk_table**, lalu ubah scriptnya menjadi seperti berikut:

```
<?php  use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateProdukTable extends Migration
{
   public function up()
   {
      Schema::create('produks', function (Blueprint $table) {
         $table->increments('id');
         $table->string('nama');
         $table->integer('id_kategori');
         $table->integer('qty');
         $table->integer('harga_beli');
         $table->integer('harga_jual');
         $table->timestamps();
      });     }
   public function down()
   {
      Schema::dropIfExists('produks');
   }
}
```

Pada file migration selalu ada dua method atau fungsi, yaitu up() dan down(). Method up() biasanya diisi script untuk membuat tabel, kolom, atau index pada database. Sedangkan method down() diisi script untuk mengembalikan operasi yang dilakukan oleh method up().

Pada script diatas kita membuat tabel dengan nama produks. Untuk membuat kolom tabelnya. Kita sebutkan dulu tipe datanya baru menuliskan nama kolomnya di dalam tanda kurung (dengan tanda petik karena merupakan string).

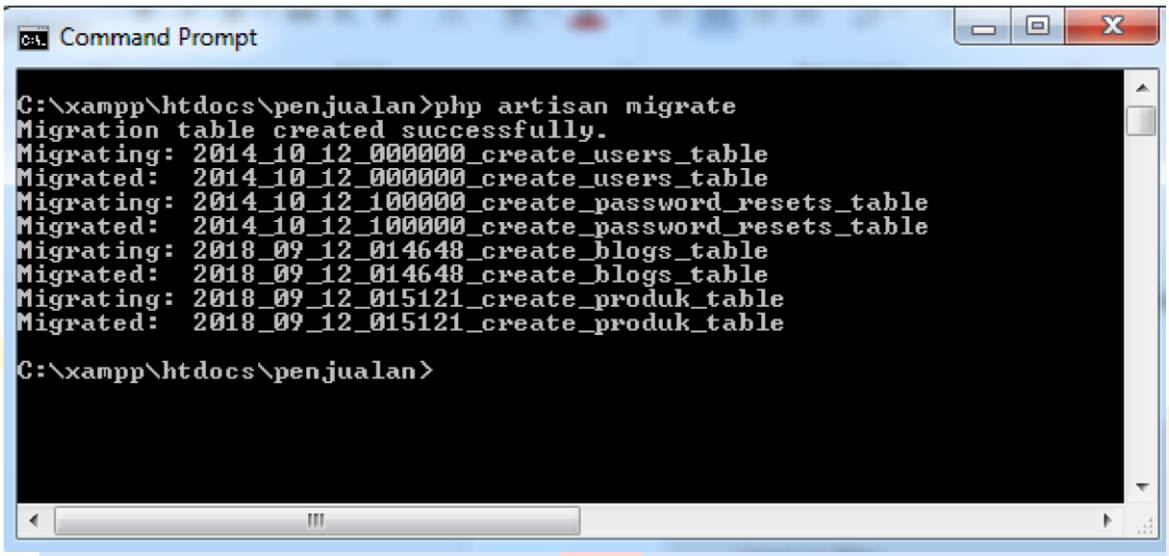
Perintah increment() digunakan untuk membuat kolom yang menjadi primary key da diset auto increments (bertambah otomatis), perintah string untuk membuat kolom dengan tipe string atau varchar, perintah integer() untuk membuat kolom dengan tipe data integer, perintah timestamps()

untuk membuat kolom pendanda waktu yang mengasilkan dua kolom yakni created_at dan updated_at.

Sebelum menjalankan perintah migration kita harus melakukan sedikit konfigurasi pada file **database.php** yang ada pada folder projek laravel yang baru dibuat atau pada folder **penjualan/config/database.php** pada baris ke 53 ubahlah value strict dari true menjadi false seperti pada script dibawah :

```
'mysql' => [
    'driver' => 'mysql',
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
           'strict' => false,
    'engine' => null,
],
```

Untuk membuat tabel dari file migration diatas ketikan perintah artisan **php artisan migrate** pada commad prompt, dan hasil dari perintah artisan tersebut dapat dilihat pada gambar 2.3 dibawah.



Perintah di atas akan menjalankan method run() pada file migration sehingga akan menghasilkan tabel pada database penjualan. Untuk membuktikan hasilnya silahkan buka http://localhost/phpmyadmin/ pilih database penjualan maka hasilnya akan tampak seperti gambar dibawah.

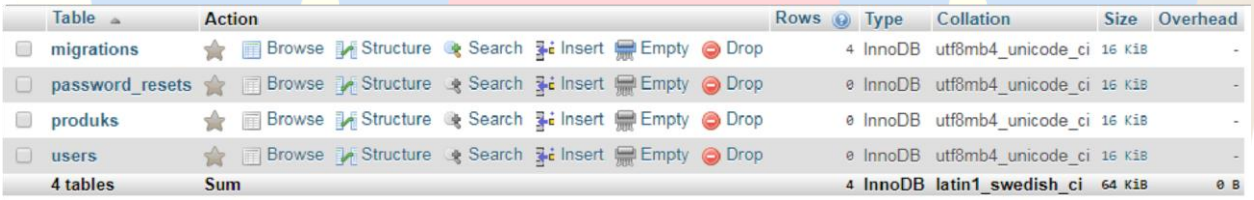


Table	Action	Rows	Type	Collation	Size	Overhead
migrations	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	16 KiB	-
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
produks	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci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

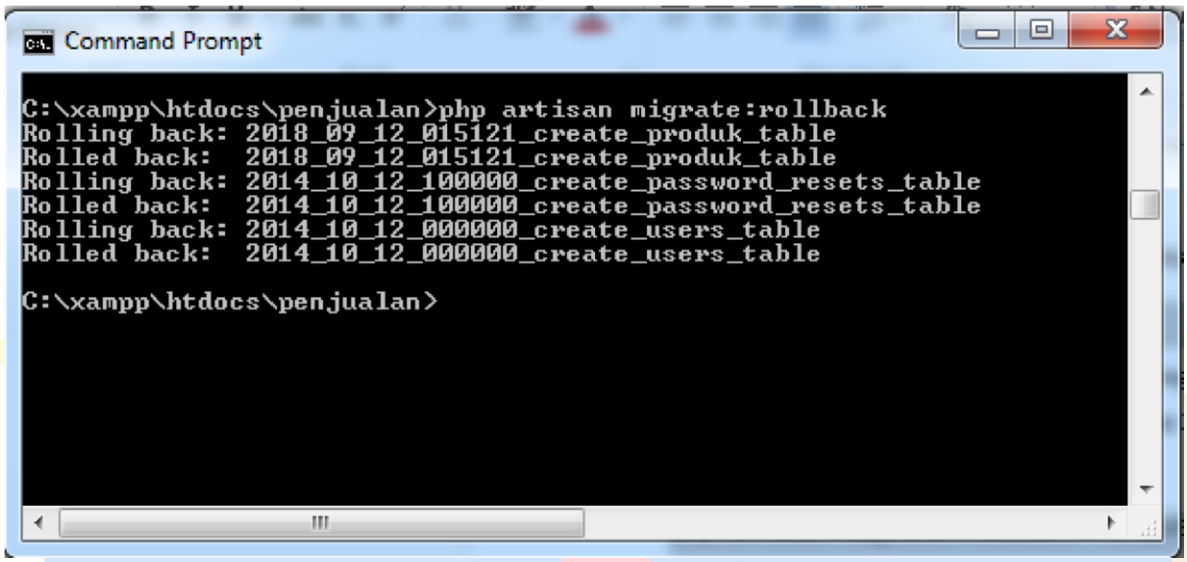
Akan ada 4 table baru yang salah satunya adalah tabel produk yang dibuat dari file migration yang baru saja dibuat. Untuk tabel migrations, password_reset dan users adalah tabel-tabel default yang telah disediakan oleh laravel untuk melakukan autentikasi yang akan dijelaskan pada bagian selanjutnya.

ROLLING BACK MIGRATION

Untuk menghapus tabel, kita dapat menjalankan perintah berikut:

```
php artisan migrate:rollback
```

perintah artisan tersebut akan atau mengembalikan satu operasi atau operasi terakhir yang telah dilakukan seperti pada contoh



Jika diperiksa pada **localhost/phpmyadmin** maka tabel yang baru saja dibuat menggunakan fitur migration akan terhapus. Untuk mengembalikan tabel-tabel tersebut jalankanlah perintah artisan **php artisan migrate** sekali lagi.

misalkan kita ingin mengembalikan ke 2 operasi terakhir, maka kita dapat menambahkan opsi –step dibelakangnya atau seperti contoh:

```
php artisan migrate:rollback –step=5
```

kita juga daoat me-roll back seluruh operasi migration, yaitu dengan perintah:

```
php artisan migrate:reset
```

atau ada kalanya kita ingin me-rollback seluruh operasi migration dan langsung menjalankan migration. Untuk keperluan seperti itu kita dapat menjalankan perintah

```
php artisan migrate:refresh.
```

MODIFIKASI TABEL

Sebelumnya kita telah membuat tabel dengan migration menggunakan perintah **schema::create**(). Selain membuat tabel, kita juga dapat mengubah nama tabel menggunakan perintah **schema::rename**(). Untuk mencobanya, kita harus membuat migration dengan nama ubah_tabel_produk dengan script artisan sebagai berikut:

```
php artisan make:migration ubah_tabel_produk
```

Selanjutnya ubah method up() pada migration yang baru dibuat menjadi seperti berikut:

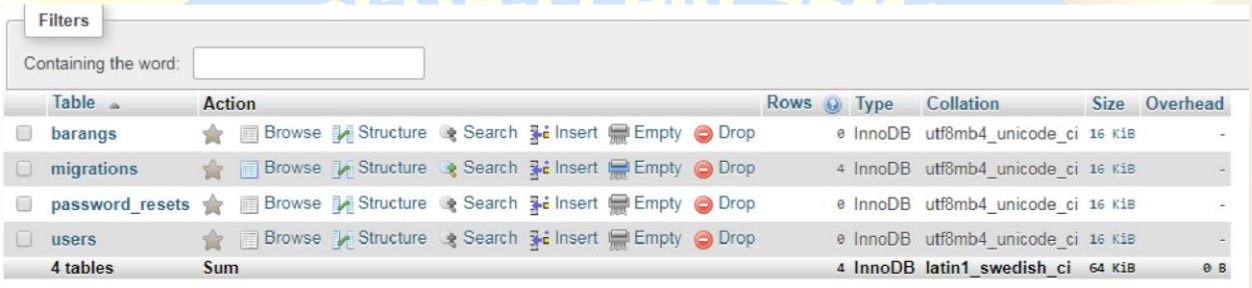
```
public function up()
{
```

```
    Schema::rename('produks', 'barangs');     }
```

Pada script diatas akan, akan mengubah nama tabel produks menjadi tabel barangs. Untuk mengeksekusinya, jalankan perintah berikut:

```
php artisan migrate
```

Sekarang nama tabel produks sudah berubah menjadi barang. Bisa dilihat pada gambar dibawah.



Selain method create(), drop() dan rename(), masih ada method lain yang dapat digunakan untuk memodifikasi tabel, antara lain dapat dilihat pada tabel 2.1 dibawah.

Tabel 2. 1 Method-method Untuk Memodifkasi Tabel Pada Migration

Contoh Sciprt	**Keterangan**

If(Schema::hasTable(‘produk’)){
 	
}
	Untuk mengecek keberadaan suatu tabel

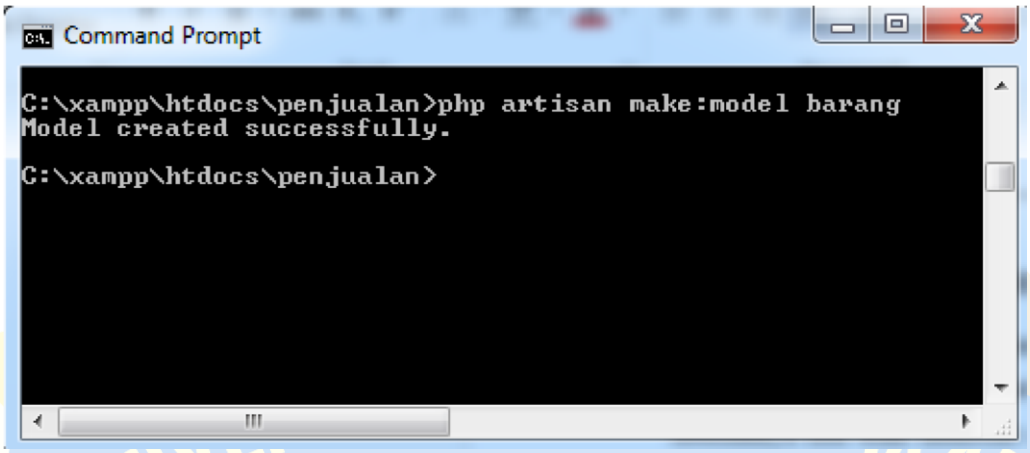
If(Schema::hasColumn(‘produk,’nama’)){
 	
}
	Untuk mengecek keberadaan kolom pada suatu tabel

Schema::dropIfExists(‘produk’);	Untuk menghapus nama tabel ketika nama tabel ditemukan.

KEGIATAN PRAKTIKUM MEMBUAT MODEL

Model merupakan bagian dari konsep MVC pada Laravel yang fungsinya untuk berinteraksi dengan database. Model dibuat berdasarkan nama tabel pada aplikasi yang dibangun. Misalnya sebelumnya kita telah membuat satu tabel yang bernama barangs. Untuk membuat model untuk table barangs kita dapat menjalankan perintah artisan dengan sciprt sebagai berikut:

```
php artisan make:model barang
```



hasil dari perintah di atas, akan ada satu file baru pada folder **penjualan/app** dengan nama **barang.php**. hasil dari perintah artisan di atas dapat dilihat pada gambar dibawah.

Laravel mengamsumsikan bahwa nama dari file model yang dibuat berarti nama tabel yang ada dalam database adalah plural dari nama model tersebut yang berarti jika nama file model disini adalah barang maka nama tabel yang ada di dalam database adalah barangs. Namun hal tersebut dapat dikonfigurasi dengan cara membuka file model **barang.php** yang baru saja dibuat dan tambahkan script berikut:

```
<?php  namespace
App;
use  Illuminate\Database\Eloquent\Model;  class  barang
extends Model
```

```
{
   protected $table = 'barangs'; }
```

Di dalam class model barang kita mendefinsikan satu variabel bernama table, value dari variabel ini akan dianggap sebagai nama tabel yang diwakili oleh model ini.

SEEDER

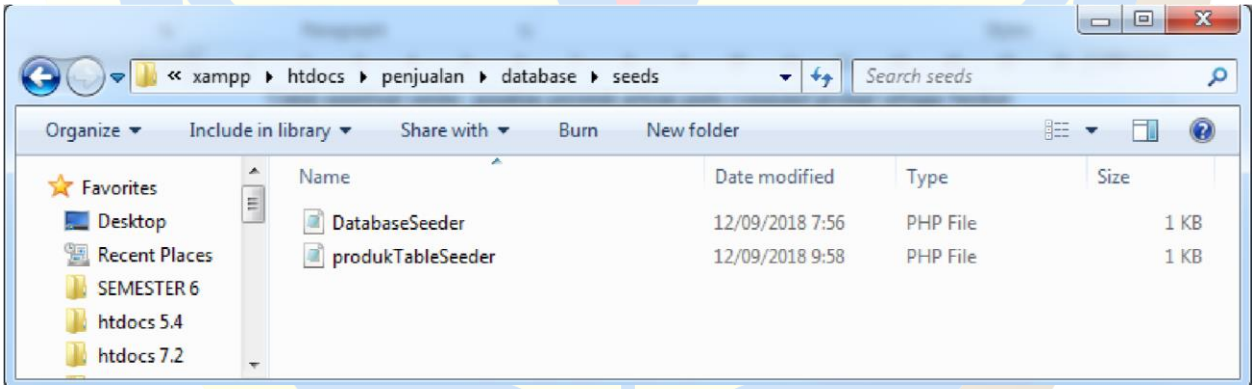
Seeder digunakan untuk membuat contoh data pada database. Fitur ini sangat bermanfaat saat melakukan pengembangan suatu sistem yang dimana kita memerlukan suatu contoh data. Apalagi ketika membutuhkan contoh data yang banyak, fitur ini dapat membantu dari pada memasukan data satu persatu scara manual melalui PhpMyAdmin.

KEGIATAN PRAKTIKUM MEMBUAT SEEDER

Untuk membuat seeder, gunakan perintah artisan pada command prompt sebagai berikut:

```
php artisan make:seeder produkTableSeeder
```

Perintah tersebut akan menghasilkan satu file baru pada folder database/seeds dengan nama **produkTableSeeder.php** atau yang diketikan pada perintah. Hasil dari perintah artisan diatas dapat dilihat pada gambar dibawah.



Selanjutnya buka file produkTableSeeder yang telah dibuat dan tambahkan script pada method run() sehingga menjadi seperti berikut:

```
<?php
use Illuminate\Database\Seeder;
class produkTableSeeder extends Seeder
{
   public function run()    {
```

```
        DB::table('produks')->insert(array(['nama' => 'Meja', 'id_kategori' => '1', 'qty' => '12',
'harga_beli' => '50000', 'harga_jual' => '540000', ],['nama' => 'Kursi', 'id_kategori'
=> '1', 'qty' => '12', 'harga_beli' => '40000', 'harga_jual' => '450000', ] ));
    }
}
```

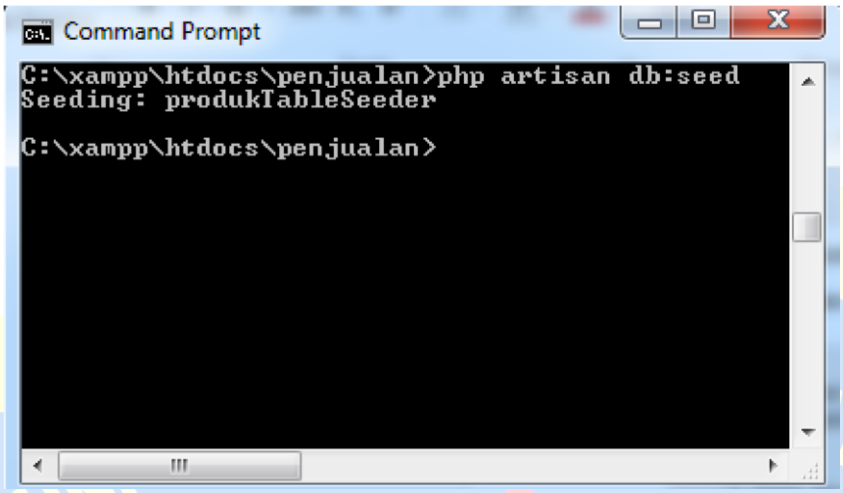
Langkah berikutnya, buka file DatabaseSeeder.php yang ada pada folder database/seeds dan panggil seeder yang baru dibuat pada method run dengan menambahkan script sebagai berikut:

```
?php
use Illuminate\Database\Seeder; class
DatabaseSeeder extends Seeder
{
    /**
* Seed the application's database.
*
* @return void
     */
    public function run()
    {
        $this->call(produkTableSeeder::class);
    }
}
```

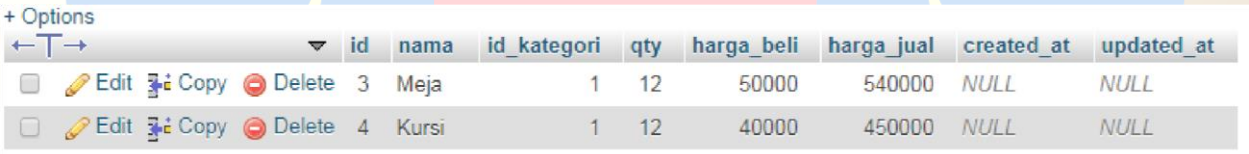
Sekarang kita bisa menjalankan perintah atisan pada comand prompt untuk mengeksekusi seeder yang telah dibaut dengan perintah sebagai berikut:

```
php artisan db:seed
```

atau seperti pada contoh dibawah:



Untuk melihat hasil dari seeder yang dibuat bukalah **localhost/phpmyadmin** pada browser lalu pilih database penjualan dan table produks maka hasilnya akan ada 2 data baru yang dihasilkan dari pembuatan seeder yang telah kita melakukan seperti pada gambar dibawah:



TUGAS

1. Exam
2. Exam

MODUL III

ROUTING

(Pertemuan 3)

Tujuan :

1. Mahasiswa dapat memahami konsep *routing* dalam framework ph
2. Mahasiswa dapat memahami cara pembuatan dan penulisan route dalam framework
3. Mahasiswa dapat mengetahui jenis-jenis route dalam framework

DASAR TEORI

Routing pada Laravel merupakan cara mengakses suatu halaman pada aplikasi melalui URL. Misalnya untuk membuka halaman awal aplikasi dapat dilakukan dengan mengetik URL **localhost:8000**. Berarti dalam menentukan route, kita menentukan bagaimana struktur URL untuk mengakses halaman tertentu. File yang digunakan untuk melakukan penyetingan route terdapat pada folder **routes**. Pada folder ini terdapat empat file php untuk pembuatan aplikasi, file yang digunakan untuk pembuatan route adalah **web.php**.

CARA MEMBUAT ROUTE

Untuk pembuatan suatu aplikasi, route dibuat pada file **routes/web.php.** kita tinggal menambahkan scipt route pada bagian bawah file tersebut. Adapun format penulisan route yaitu sebagai berikut:

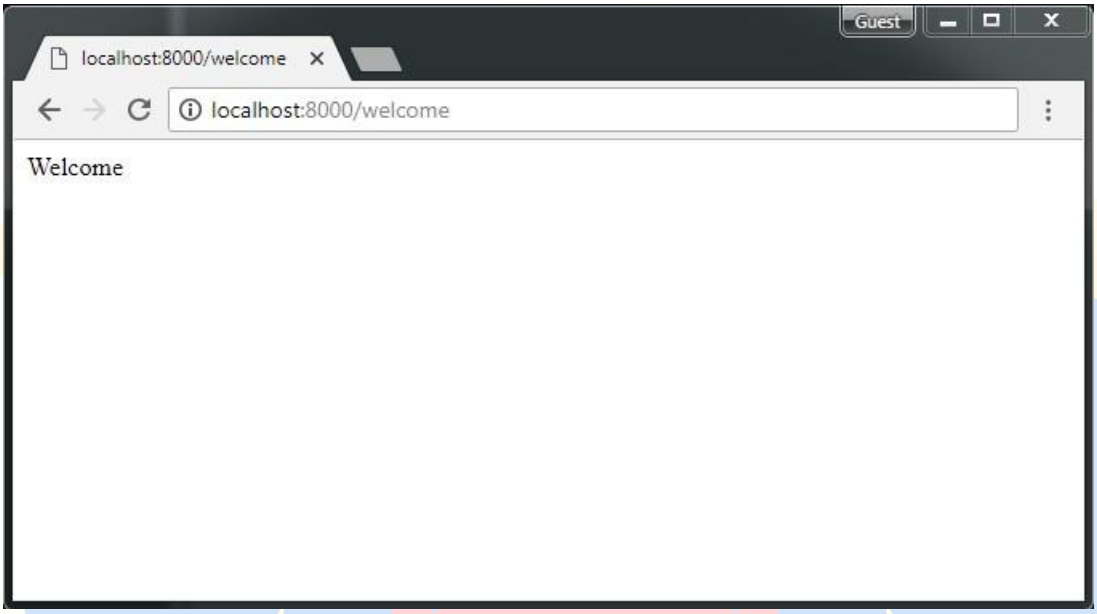
```
Route::get($url,$callback);
```

Pada format penulisan di atas, $url diisi dengan format URL dari route, sedangkan $callback diisi dengan script yang akan dijalankan ketika membuka URL tersebut. Callback dapat berupa controller atau fungsi. Contoh penulisan route sederhana adalah sebagai berikut:

```
Route::get('/welcome', function () {          echo
"Welcome";

});
```

Untuk menguji route diatas pertama-pertama pastikan server artisan sudah berjalan, untuk menjalankan server artisan itu sendiri sudah dibahas pada modul sebelumnya yaitu melalui cmd atau comand promt dengan cara mengarahkan direktori ke dalam projek laravel yang dalam kasus ini adalah projek penjualan dan kemudian mengetikan **php artisan serve**.

Lalu jika kita mengetikan pada browser **localhost:8000/welcome** maka hasilnya akan seperti gambar dibawah:



JENIS-JENIS ROUTE METHOD

Ada berbagai method dalam route yang dapat digunakan sesuai dengan HTTP request. Pada contoh berikut ada beberapa contoh route yang menggunakan route method:

1. Untuk menampilkan atau mengambil data dapat menggunakan method GET. Route dengan method ini dapat langsung kita akses melalui link pada browser. Contohnya dapat dilihat pada latihan dibawah:

```
Route::get('/index', function () {     echo "Uji Coba route
dengan method GET";
});
```

2. Untuk mengirim data dari form dengan dapat menggunakan method POST, biasasnya digunakan untuk menambah data. Adapun penulisan dari route ini adalah sebagai berikut:

```
Route::post('/store', function () {
    // sintak untuk menyimpan data
});
```

3. Untuk mengirim data dari form dengan tujuan untuk memperbaharui data dapat menggunakan method PUT, contoh penulisan route dengan method PUT adalah sebagai berikut:

```
Route::put('/update', function () {
    // sintak untuk upadte data
```

```
});
```

4. Untuk mengirim data dari form dengen tujuan untuk menghapus data dapat menggunakan method DELETE. Contoh dari penulisan route dengen method delete adalah sebagai berikut:

```
Route::delete('/delete', function () {
    // sintak untuk menghapus data
});
```

5. Untuk route yang dapat merespons beberapa HTTP request, kita dapat menggunakan method **match()**.

```
Route::match(['get','post'],'/welcome' function () {
    //
});
```

6. Sedangkan untuk route yang dapat merespons semua HTTP request, dapat menggunakan method **any()**. Contohnya pada script berikut:

```
Route::any('/welcome' function () {
    //
});
```

Untuk contoh dari penggunakan masing-masing method route diatas akan akan dibahas pada modul selanjutnya.

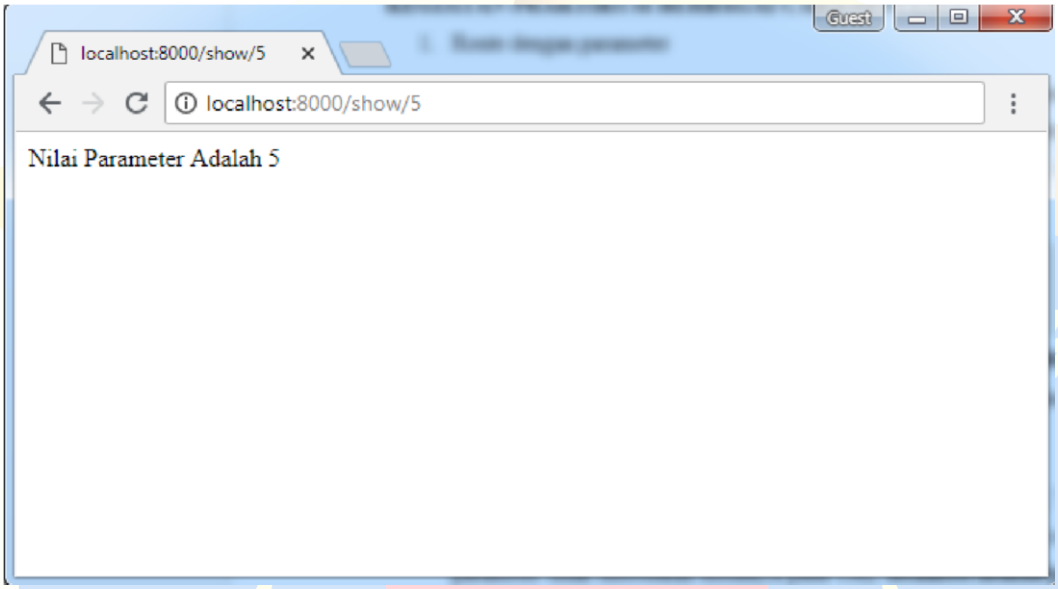
KEGIATAN PRAKTIKUM BERBAGAI CARA PENULISAN ROUTE

1. Route dengan parameter

   Parameter pada route ditandai dengan tanda { }. Artinya kata pada URL yang berada pada posisi ini akan menjadi nilai dari parameter. Untuk penulisanya buatlah satu route baru pada file **web.php** yang ada pada folder **routes/web.php** dengan skrip sebagai berikut:

```
Route::get('/show/{id}', function ($id) {       echo "Nilai
Parameter Adalah ".$id;
});
```

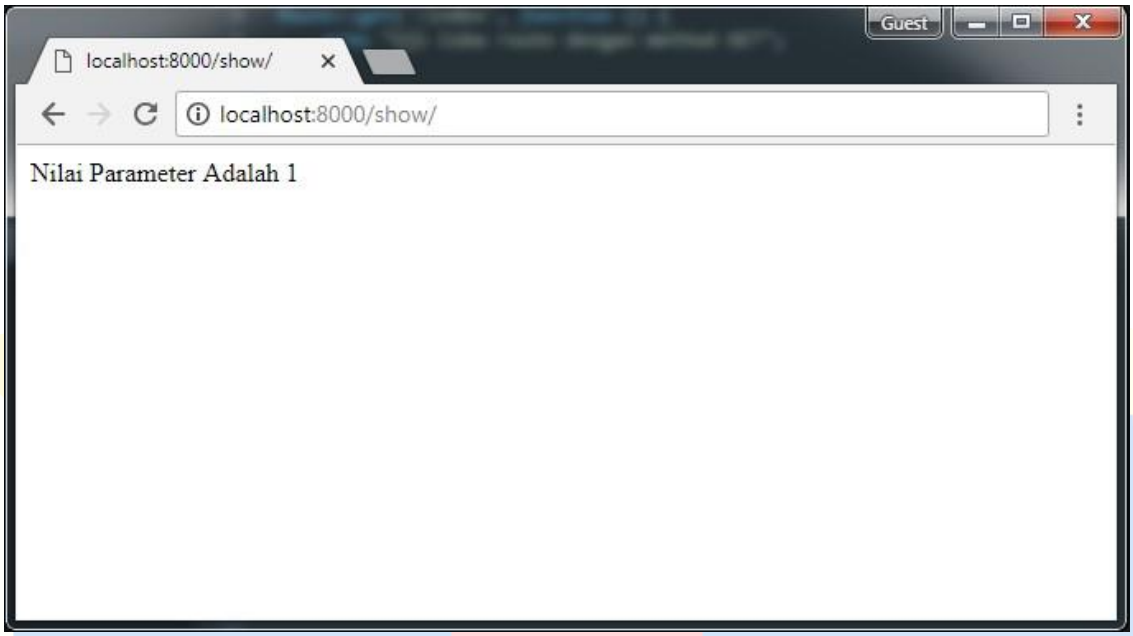
Kemudian bukalah browser dan ketikan pada link **localhost:8000/show/5** angka 5 tersebut akan menjadi parameter dari route yang dibuat yang artinya bebas diinputkan berapa saja dan jika dilihat pada browser maka hasilnya akan menjadi seperti gambar dibawah:



Jika parameter tidak wajib diisi maka tambahkan tanda tanya (**?**) dibelakang nama parameter dan pada fungsi kita beri default parameter pada contoh dibawah jika parameter tidak disebutkan misalnya pada URL **localhost:8000/show** maka parameter **$id** akan dianggap bernilai 1 sesuai dengan default parameter yang diberikan.

```
Route::get('/show/{id?}', function ($id=1) {        echo "Nilai
Parameter Adalah ".$id;
});
```

Jika route diatas dijalankan maka hasilnya akan menjadi seperti gambar dibawah

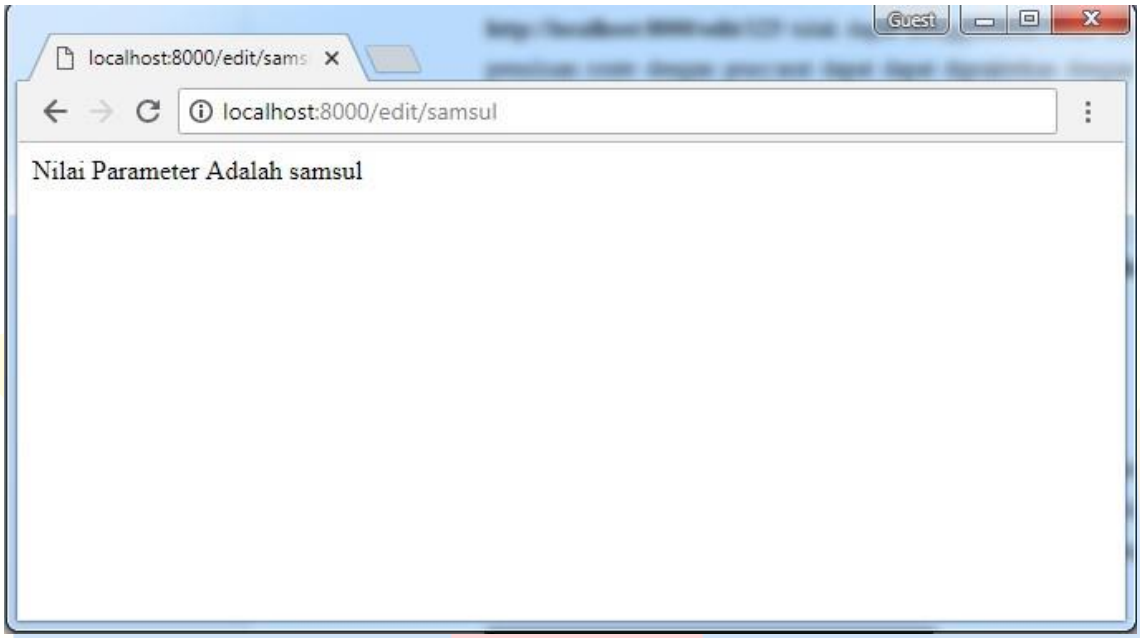


2. Route dengan reguler expression

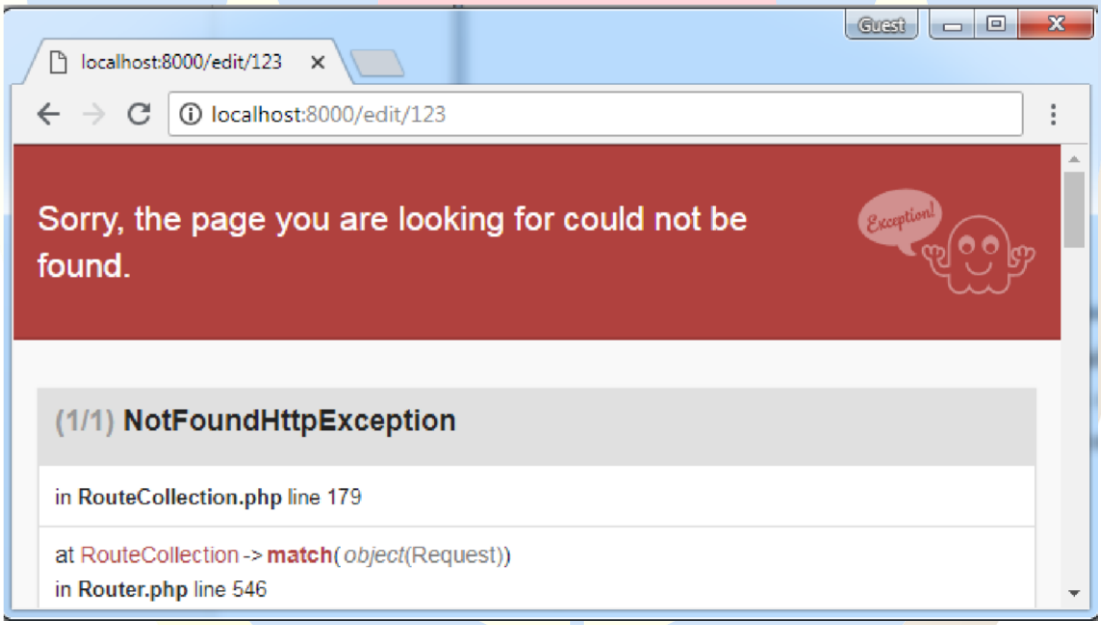
Route ini akan dijalankan hanya jika nilai dari parameter pada URL memenuhi syarat tertentu. Misalnya pada baris pertama contoh di bawah, parameter judul harus berupa karakter A sampai Z atau a sampai z. Dengan demikian, maka URL **localhost:8000/edit/samsul** memenuhi syarat dari route tersebut, sedangkan URL **http://localhost:8000/edit/123** tidak dapat menggunakan route tersebut. Adapun contoh penulisan route dengan prasyarat dapat dapat dipraktekan dengan menambahkan route dibawah pada file **web.php** pada folder **routes/web.php.**

```
Route::get('/edit/{nama}', function ($nama) {       echo "Nilai
Parameter Adalah ".$nama;
})->where('nama','[A-Za-z]+');
```

Kemudian jika dijalankan pada browser dengan alamat link **localhost:8000/edit/samsul** maka hasilnya akan menjadi seperti berikut:



Sedangkan jika nilai pada bagian parameternya kita ganti menjadi angka maka Laravel tidak akan mengeksekusi route tersebut. Dapat dicoba dengan mengetikan pada link browser alamat **http://localhost:8000/edit/123** maka laravel akan mengeluar error seperti berikut:



Error diatas bertuliskan NotFoundHttpException yang berarti Route untuk link yang diakses tidak ditemukan.

3. Route dengan nama

Route dengan pendefinisian nama atau identitas memiliki kelebihan yang dapat digunakan hanya dengan menyebutkan nama yang diberikan. Contohnya buatlah 2 route baru pada file **web.php** yang ada pada folder **routes/web.php**

```
Route::get('/index', function () {
```

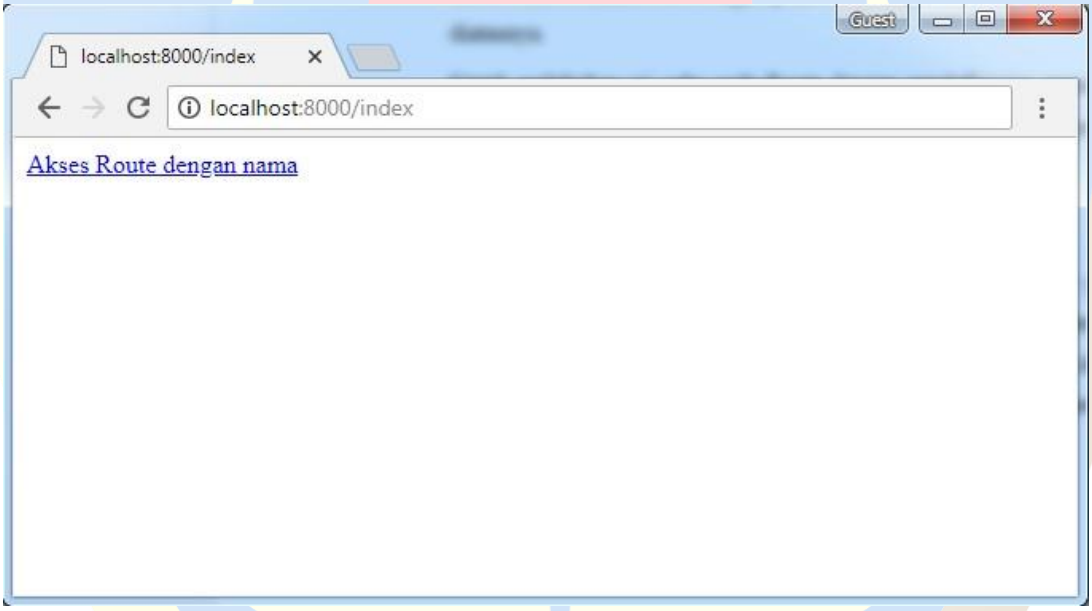
```
    echo "<a href="'.route('create').'">Akses Route dengan nama </a>";
});
```

Route diatas akan digunakan untuk menapilkan tag <a> dalam atribut href nya akan diisikan nama dari route yang akan diakses.

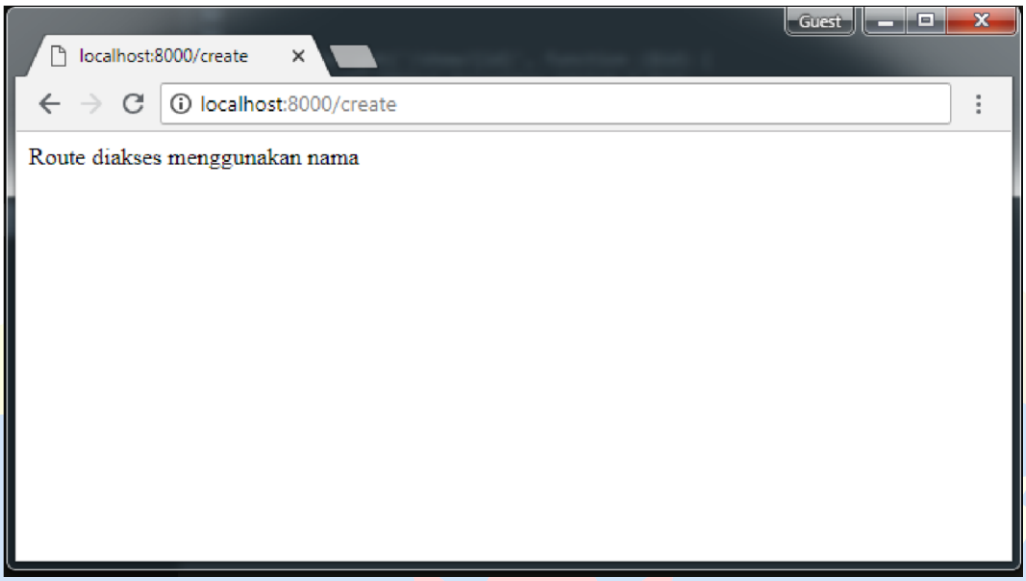
```
Route::get('/create', function () {        echo "Route
diakses menggunakan nama";
})->name('create');
```

Route diatas adalah route dengan pendefinisan nama yang akan coba kita akses dari route diatasnya.

Untuk melakukan uji coba pada Route dengan pendefinisian nama bukalah browser dan ketikan link **localhost:8000/index** maka akan muncul halaman seperti berikut:



Jika link **Akses route dengan nama di klik** maka kita akah diarahkan ke halaman dengan route name create sebagai berikut:

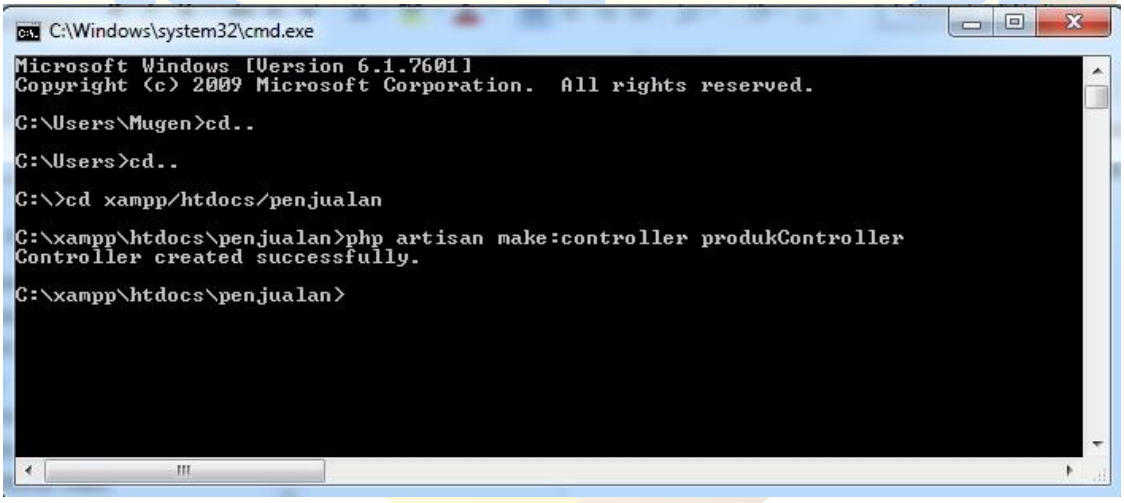


4. Route dengan aksi controller

Route seperti ini ketika dijalankan akan mengakses controller yang disebutkan pada parameter kedua. Jika ingin mengakses method atau function tertentu pada controller, maka antara nama controller dengan nama fungsi dipisahkan dengan tanda **@**. Untuk mempraktekan route ini buatlah satu controller dengan menggunakan perintah artisan sebagai berikut:

```
php artisan make:controller:produkController
```

Jalankan perintah artisan diatas pada cmd yang sudah terarah pada folder projek Laravel yang digunakan atau dalam modul ini adalah projek dengan nama penjualan seperti pada gambar dibawah:



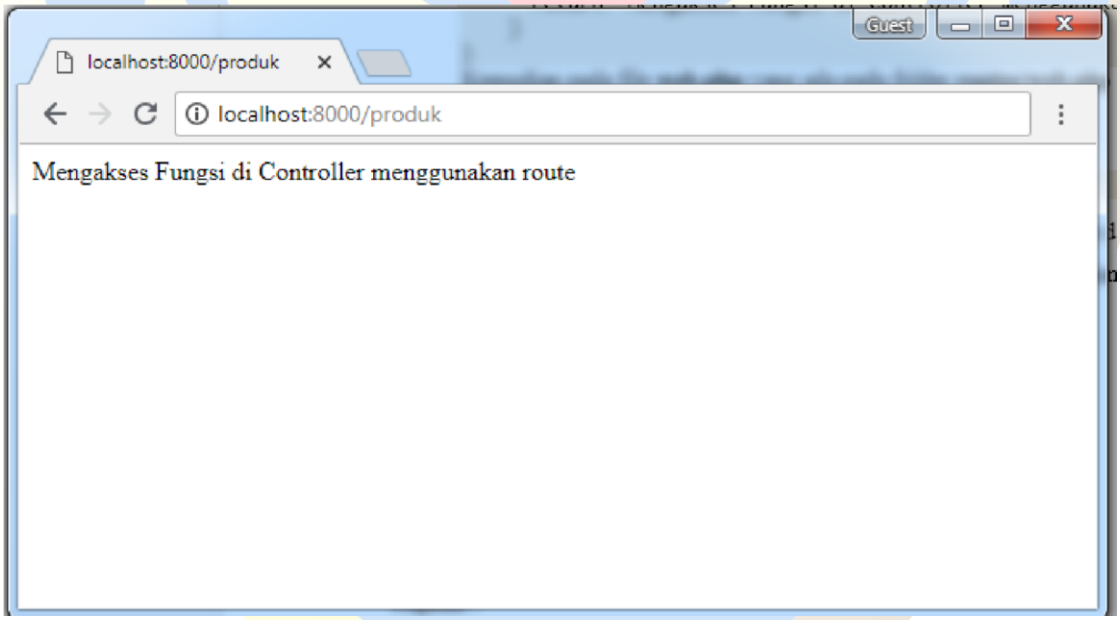
Perintah artisan diatas akan menghasilkan satu file baru bernama **produkController.php** yang terletak di folder **app\Http\Controller.** Bukalah file controller tersebut dan tambahkan satu fungsi bernama index seperti pada controh skrip dibawah:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class produkController extends Controller
{
    public function index()
    {
     return 'Mengakses Fungsi di Controller menggunakan route';     }
}
```

Kemudian pada file **web.php** yang ada pada folder **routes/web.php** tambahkan satu route baru dengen bentuk seperti berikut:

```
Route::get('/produk', 'produkController@index');
```

Lakukan uji coba pada route diatas dengen cara membuka browser dan ketikan localhost:8000/produk pada link browser yang digunakan maka akan muncul halaman seperti berikut:



Tugas

- Buatlah route sesuai dengan projek anda

MODUL IV

VIEW DAN BLADE TEMPALTE

(Pertemuan 4 dan 5)

Tujuan :

1. Mahasiswa mampu memahami konsep view dalam framework
2. Mahasiswa mampu memahami konsep blade dalam framework
3. Mahasiswa mampu membuat view dan blade
4. Mahasiswa mampu memahami kontrol struktur dengan blade template
5. Mahasiswa mampu mengetahui master template blade (layout)

DASAR TEORI

View merupakan bagian yang menampilkan informasi untuk disampaikan kepada users. View terdiri dari script HTML dengan bantuan CSS dan javascript. Sesuai dengan aturan konsep MVC, di dalam view tidak boleh ada script logika maupun script untuk mengakses database. Di dalam laravel view diletekan di dalam folder **resources/views/**.

Sedangkan blade template merupakan engine yang disediakan laravel untuk memudahkan developer dalam menampilkan data pada view. Dengan blade tempalte ini kita tidak perlu lagi menggunakan **<?php echo $data ?>** untuk menampilkan variabel data pada view. Untuk menggunakan blade tempalete, file view harus disimpan dengan akhiran **.blade.php** dan disimpan pada folder **resourcess/views**.

KEGIATAN PRAKTIKUM 1 MEMBUAT VIEW

Untuk mempraktekan pembuatan view, terlebih dahulu kita membuat sebuah route untuk mengakses view tersebut. Route yang dibuat akan langsung mengarah ke controller **produkController.php** yang telah dibuat pada latihan minggu. Route yang dibuat adalah seperti pada sintak dibawah:

```
Route::get('/produk', 'produkController@index');
```

Pada route diatas, jika pada browser dituliskan alamat **localhost:8000/produk** maka kita akan diarahkan ke class produkController ke dalam fungsi index. Selanjutnya bukalah file **produkController.php** yang ada di dalam folder **app\Http\Controller** dan kemudian modifikasi script nya menjadi seperti berikut:

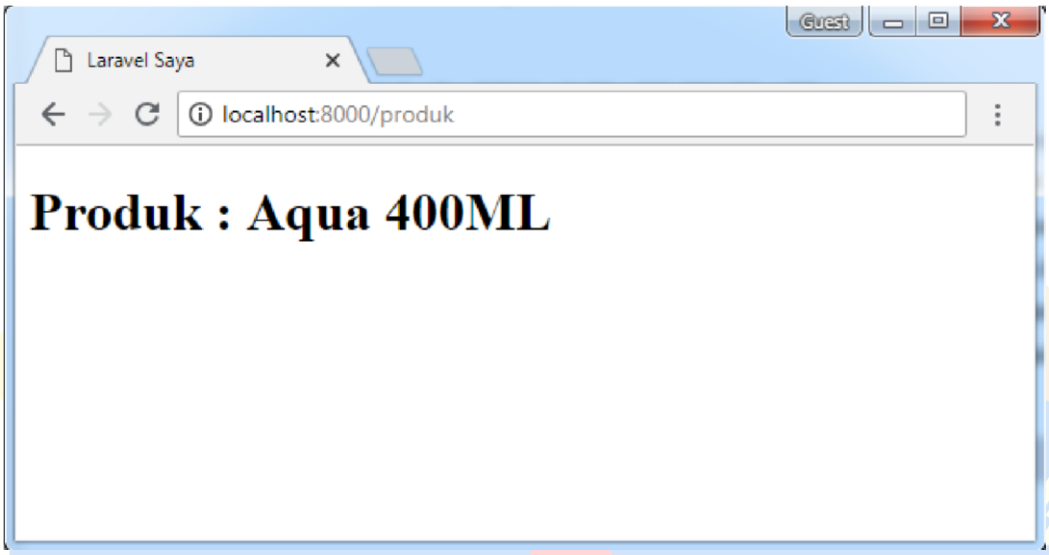
```
<?php
```

```
namespace App\Http\Controllers; use
Illuminate\Http\Request;
class produkController extends Controller
{
   public function index()
   {
    $produk = 'Aqua 400ML';
    return view('produk/index',compact('produk'));        }
}
```

Di dalam fungsi index() yang telah dibuat kita mendefinisikan satu variabel bernama **produk** yang langsung kita ikut sertakan lempar bersama dengan return view yang dimana view yang akan dicari berada pada folder projek Laravel yaitu **penjualan/resources/views/produk/** dengan nama **index.blade.php**. karena view tersebut belum dibuat maka buatlah satu file baru pada folder **penjualan/resources/views/produk/** dan beri nama **index.blade.php** dan isikan skript sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
        <title>Laravel Saya</title>
</head>
<body>
        <h1>Produk : {{ $produk }}</h1>
</body>
</html>
```

Pada script diatas kita menampilkan value dari variabel produk dengan hanya menggunakan tanda **{{** dan ditutup dengan tanda **}}**. Fitur yang memungkinkan untuk menampilkan data dengan cara tersebut yaitu **blade template** yang akan dibahas pada materi selanjutnya. Dan jika kita membuka link **localhost:8000/produk** maka akan muncul halaman seperti berikut:

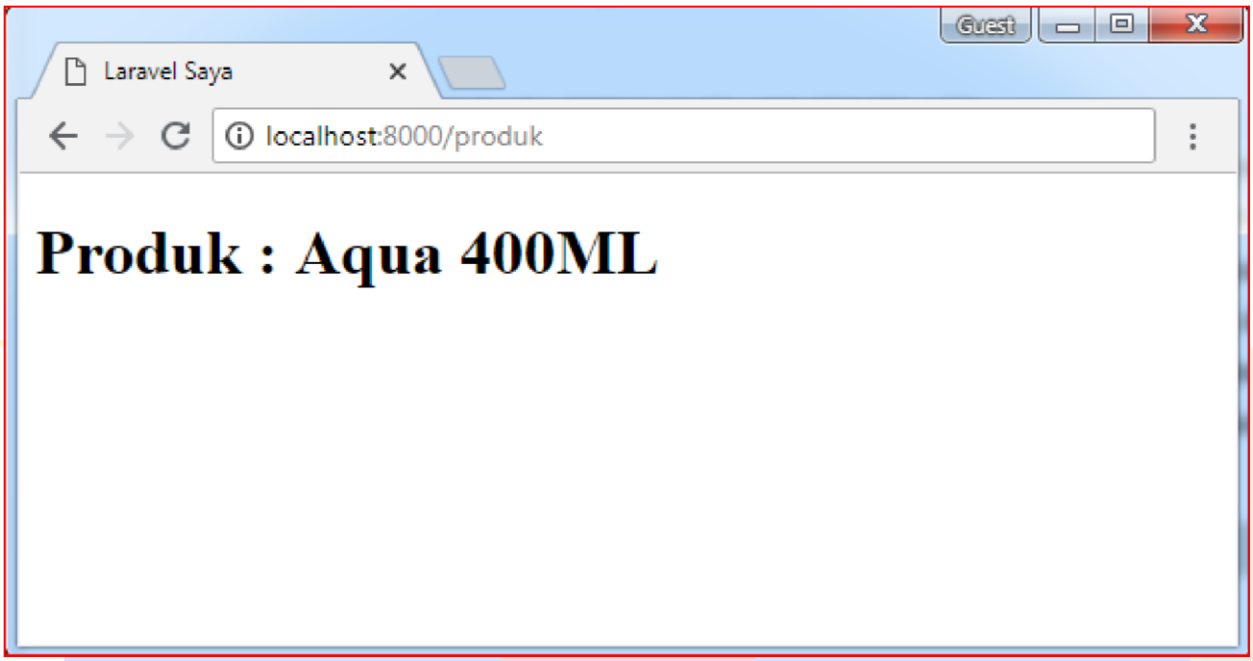


KEGIATAN PRAKTIKUM 2 MENAMPILKAN DATA DENGAN BLADE TEMPLATE

Biasanya untuk menampilkan variabel PHP pada HTML kita menggunakan script **<?php echo $data ?>**, dengan blade template kita cukup menulisnya dengan script **{{ $data }}**. Cara ini juga sudah dilengkapi dengan pencegahan XXS attacks (hacker yang memanfaatkan javascript untuk mencuri informasi). Untuk melihat contoh dari penggunaan data tersebut dapat dilihat pada contoh sebelumnya. Pada kasus lain, kadang kita menampilkan variabel dengan mengecek terlebih dahulu apakah variabel tersebut telah dideklarasikan atau belum. Dengan blade template, kita dapat memodifikasi sintak yang ada pada file **index.blade.php** yang berada pada folder **penjualan/resources/views/produk/** menjadi seperti berikut:

```
<!DOCTYPE html>
<html>
<head>
        <title>Laravel Saya</title>
</head> <body>
        <h1>Produk : {{ isset($produk) ? $produk : 'Produk Kosong' }}</h1>
</body>
</html>
```

Dan jika dijalankan kembali hasilnya adalah seperti berikut:



Pada hasil diatas memang tidak terdapat perbedaan dari sebelumnya, namun sebernarnya dengan cara tersebut, jika variabel produk telah dideklarasikan, maka akan ditampilkan nilai variabel produk, sedangkan jika belum dideklarasikan maka akan menampilkan text default ‘produk kosong’.

KEGIATAN PRAKTIKUM 3 CONTROL STRUKTUR PADA BLADE TEMPLATE

Blade template juga menyediakan cara tersendiri untuk melakukan logika percabangan maupun perulangan. Untuk melakukan percabangan dengan blade template dapat menggunakan **@if.. @else.. @endif.** Untuk mencobanya buatlah satu route baru pada file **web.php** yang berada pada folder **routes/web.php** dengan sintak sebagai berikut:

```
Route::get('/produk/show, 'produkController@show);
```

Kemudian pada class controller **produkController.php** yang berada pada folder **app\Http\Controller** tambahkan satu fungsi baru bernama **show()** dan isikan script sebagai berikut:

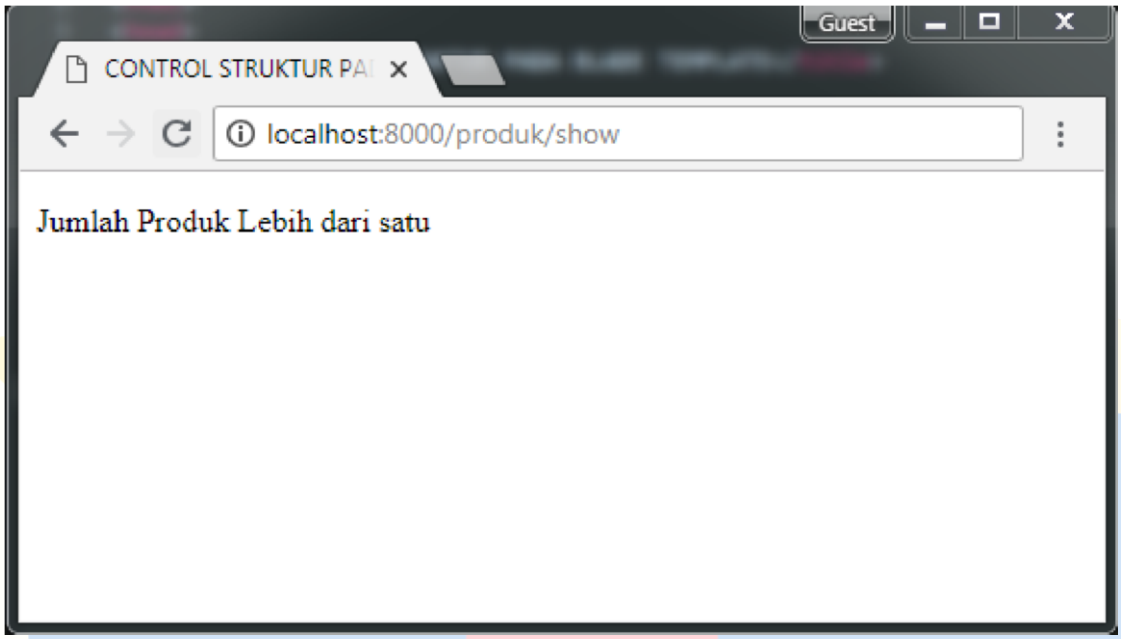
```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class produkController extends Controller
{
   public function index()
   {
    $produk = 'Aqua 400ML';
```

```
    return view('produk/index',compact('produk'));     }
   public function show()     {
    $produk = ['Aqua 115 ML','Minuman Bersoda','Buku Sejarah','Mouse','CPU'];       return
view('produk/show',compact('produk'));     }
}
```

Selanjutnya buatlah view baru dengan nama **show.blade.php** pada folder **resources/views/produk** dan isikan skript seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
        <title>CONTROL        STRUKTUR        PADA        BLADE
TEMPLATE</title> </head>
<body>
   @if (count($produk)==1)
        <p>Jumlah Produk adalah satu</p>
   @elseif(count($produk)>1)
        <p>Jumlah Produk Lebih dari satu</p>
   @else
        <p>Tidak Ada Data Produk</p>
   @endif
</body>
</html>
```

Jika dicoba dijalankan dengan mengakses link **localhost:8000/produk/show** maka tampilannya akan menjadi seperti pada gambar dibawah:



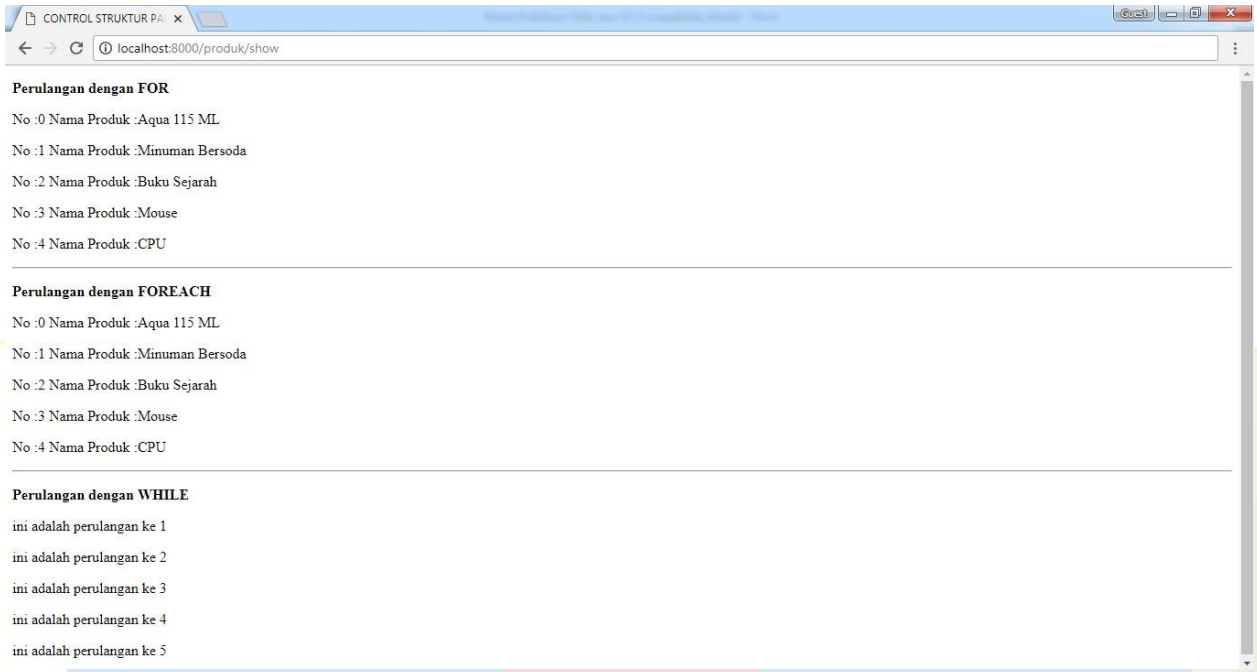
Sedangkan untuk perulangan pada blade template mendukung berbagai jenis perulangan, dengan contoh sebagai berikut:

Perulangan dengan @for, @foreach dan @while, untuk mempraktekan penggunaan perulangan menggunakan blade template kita masih akan menggunakan route **localhost:8000/produk/show.** Pada class Controller **produkController.php** di dalam method show() kita telah mendifisikan satu array bernama produk, pada view **show.blade.php** yang berada pada folder **resources/views/produk** modifikasi koding di dalamnya menjadi seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
        <title>CONTROL STRUKTUR PADA BLADE
TEMPLATE</title> </head>
<body>
        @if (count($produk)==1)
                <p>Jumlah Produk adalah satu</p>
        @elseif(count($produk)>1)
                <p><b>Perulangan dengan FOR</b></p>
                @for ($i = 0; $i < count($produk); $i++)
                        <p>{{ 'No :'.$i.' Nama Produk :'.$produk[$i] }}</p>
                @endfor
                <hr>
```

```
                <p><b>Perulangan dengan FOREACH</b></p>
                @foreach ($produk as $x => $v)
                        <p>{{ 'No :'.$x.' Nama Produk :'.$v }}</p>
                @endforeach

                <hr>
                <p><b>Perulangan dengan WHILE</b></p>
                @php
                        $no = 1;
                @endphp
                @while ($no<=5)
                        <p>ini adalah perulangan ke {{ $no }}</p>
                        @php
                                $no++;
                        @endphp
                  @endwhile
        @else
                <p>Tidak Ada Data Produk</p>
        @endif
</body>
</html>
```

Jika halaman pada link **localhost:8000/produk/show** di refresh atau dimuat ulang maka halamanya akan nampak seperti pada gambar dibawah;



KEGIATAN PRAKTIKUM 4 LAYOUT DENGAN BLADE TEMPLATE

Untuk membuat layout pada template aplikasi, Laravel menggunakan cara yang mudah menggunakan blade template. Untuk mempraktekannya buatlah satu folder baru yang bernama layout pada folder **resourcess/views**, lalu bautlah file dengan nama **layout.blade.php** di dalamnya dan isikan script sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
	<title>@yield('title')</title>
</head>
<body>
	<header>
		@include('layout.header')
	</header>
	<ul>
		@section('sidebar')
			<li>HTML</li>
			<li>CSS</li>
			<li>JS</li>
		@show
	</ul>
```

```
        <div class="container">
                @yield('content')
        </div>
</body>
</html>
```

Keterangan dari masing-masing script diatas adalah sebagai berikut:

1. **@yield('title')** diartikan kita menyediakan tempat yang datanya akan diisi oleh file lain menggunakan **@section('title')**. Begitu juga dengan script **yield('content')** akan diisi oleh file blade lain dengan **@section('content')**. Data data bagian ini bersifat dinamis sesuai dengan halaman yang sedang dibuka.
2. **@inlcude('layout.header')** diartikan kita memanggil file **header.blade.php** pada folder layout untuk ditampilkan pada halaman tersebut.
3. **@section('sidebar)** ... **@show** diartikan kita menyediakan tempat yang isi dapat ditambahkan oleh file lain dengan menggunakan section **@section('sidebar').**

Selanjutnya buatlah satu file baru pada folder yang sama yaitu **resources/views/layout** dengan nama **header.blade.php** dan isikan script sebagai berikut:

```
<h1>Layout Dengan Blade Template Laravel</h1>
```

Selanjutnya untuk membuat contoh halaman yang menggunakan layout. Buatlah satu folder baru dan berinama **konten** pada folder **resourcess/views** untuk menyimpan contoh halaman yang akan kita buat. Misalnya kita buat satu file dengan nama **halaman.blade.php** dengan script sebagai berikut:

```
@extends('layout.layout')
@section('title',$title)
@section('sidebar')
        @parent
        <li>PHP</li>
@endsection
@section('content')
        <p>{{ $konten }}</p>
@stop
```

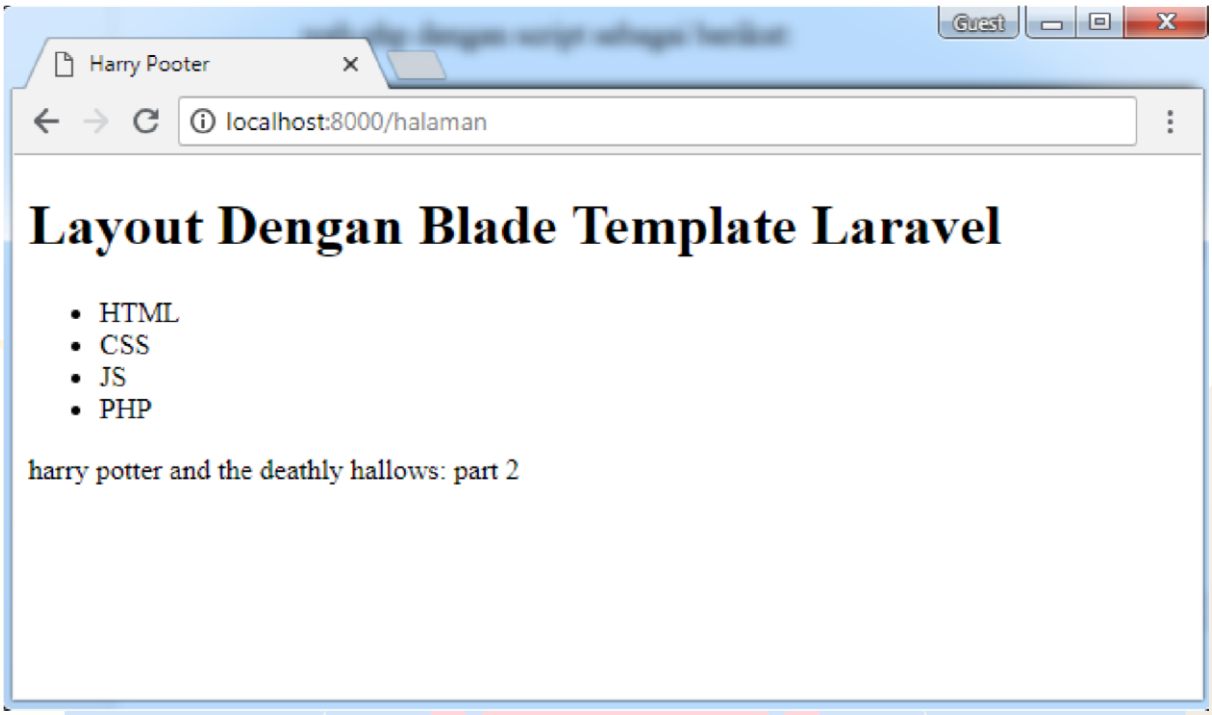
Keterangan dari masing-masing script diatas adalah sebagai berikut:

1. **@extends('layout.layout)** berarti kita mewarisi semua yang ada pada file layout.blade.php pada folder layout sebagai file master template.
2. **@section('title',$title)** berarti kita menyiapkan data untuk ditampilkan di bagian **yield('title')** pada file master yaitu **layout.blade.php**. data **$title** dikirim dari controller atau route yang memanggil view dengan fungsi view.
3. **@section('sidebar')** ... **@endsection** berarti kita menggabungkan isi sidebar di atas dengan sidebar yang ada pada file **layout.blade.php**, dimana sidebar dari file layout.blade.php diletakan pada bagian @**parent**.
4. **@section('content') ..... @stop** diartikan kita menyiapkan data untuk ditampilkan di bagian **@yield('content')** pada file master. data **$content** dikirim dari controller atau route yang memanggil view dengan fungsi view.

Untuk membuktikan hasil dari pembuatan layout diatas, silahkan buat route pada file **web.php** pada file **routes/web.php** dengan script sebagai berikut:

```
Route::get('/halaman',function(){
   $title = 'Harry Pooter';
   $konten = 'harry potter and the deathly hallows: part 2';       return
view('konten.halaman',compact('title','konten')); });
```

Pada script diatas kita membuat variabel $title dan $konten yang akan dikirimkan ke view dengan fungsi compact(). Jika kita membuka **localhost:8000/halaman** maka hasilnya akan nampak seperti gambar dibawah:



Tugas

- Buatlah view dan blade untuk projek anda
- Buatlah layout dengan master template blade untuk projek anda

MODUL V

CONTROLLER

(Pertemuan 6 dan 7)

Tujuan :

1. Mahasiswa mampu memahami konsep controller dalam framework
2. Mahasiswa mampu mengetahui penulisan controller dalam framework
3. Mahasiswa mampu memahami pembuatan controller dalam framework

DASAR TEORI

Controller merupakan bagian dari konsep MVC yang bertugas untuk memproses semua request untuk ditampilkan kembali melalui view. Bisa dikatakan controller merupakan jembatan penghubung antara model dan view. Jika sebelumnya kita masih ada mengatur proses pada route, kini tidak saatnya lagi karena itu merupakan tugas dari controller. Pada aplikasi yang akan dibuat nanti semua route diarahkan ke controller dan tidak ada lagi proses seperti mengambil data dari database atau memanggil method view pada route. Pada modul sebelumnya kita telah menggunakan satu class controller yaitu produkContrller, namun pada modul ini kita akan lebih mendalami cara penggunaan controller.

Untuk membuat Controller, kita tidak perlu membuatnya secara manual, karena Laravel memiliki fitur artisan yang sudah dibahas pada bagian sebelumnya untuk mempermudah pembuatan Controller. Silahkan jalankan cmd atau comand prompt dan arahkan ke projek laravel yang dibuat kemudian tuliskan script berikut (masih dengan projek Laravel **penjualan**):

```
php artisan make:controller pelangganController
```

Perintah tersebut akan menghasilkan file controller dengan nama **pelangganController.php** pada folder **penjualan/app/Http/Controller**. Untuk contoh penggunaan controller pada proses Laravel pertama siapkan sebuah route pada file **web.php** yang ada di folder **routes/web.php** seperti pada gambar dibawah:

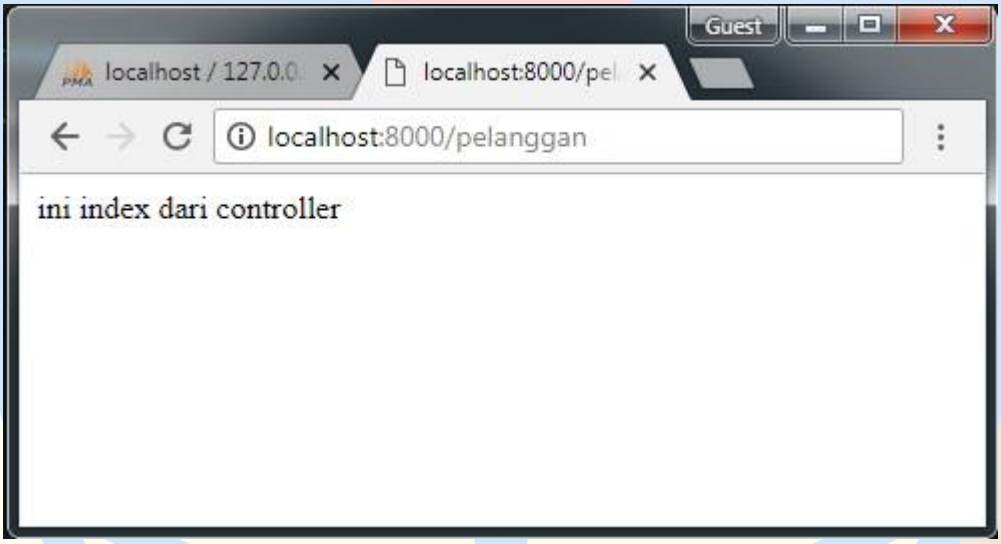
```
Route::get('/pelanggan', 'pelangganController@index');
```

Kemudian pada file **pelangganController.php** yang ada pada folder **app\Http\controller** yang baru saja dibuat buatlah satu function atau method yang bernama index() dan isikan script seperti berikut:

```
<?php
```

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class pelangganController extends Controller
{
   public function index()
   {
    echo "ini index dari controller";
   } }
```

Untuk mengetes apakah pembuatan fungsi pada controller sudah berhasil maka bukalah alamat route yang baru saja dibuat yaitu **localhost:8000/pelanggan** (pastikan server artisan telah aktif) maka hasilnya akan nampak seperti berikut:



Atau pada funtion index yang ada pada **pelangganController.php** kita dapat langsung melakukan return view atau mengarahkan route yang diakses ke suatu halaman, ubahlah function index yang ada pada pelangganController lalu buatlah menjadi seperti berikut:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class pelangganController extends Controller
{
   public function index()
   {
    return view('pelanggan.index');
```

```
    }
}
```

Kemudian buatlah satu folder dalam folder **resourcess/view** bernama pelanggan dan buat satu file yang bernama **index.blade.php** dan isikan script seperti berikut:

```
<!DOCTYPE html>
<html>
<head>
        <title>Pelanggan</title>
</head>
<body>
        <h1>Mengakses View Pelanggan dari Controller</h1>
</body>
</html>
```

Kemudian akses kembali route **localhost:8000/pelanggan** atau reload halaman yang telah dibuka sebelumnya, maka tampilan nya akan menjadi seperti berikut:

Pada gambar diatas adalah halaman view yang diakses dari route **locahost:800/pelanggan** yang route tersebut mengarahkan kita ke **pelangganController.php** dengan function yang dituju adalah

function index(). Di dalam function index yang ada di dalam **pelangganController.php** halaman diarahkan dengan menggunakan method view ke dalam view **index.blade.php** yang ada pada folder folder **resourcess/views/pelanggan**.

MENGAKSES FUNGSI BERBEDA DALAM SATU CONTROLLER

Di dalam konsep laravel kita dapat memanggil suatu fungsi dari fungsi lain di dalam laravel menggunakan methid **this**. Method **this** ini menandakan kita akan memanggil fungsi yang ada di dlam satu controller yang sama dengan bentuk pemanggilan adalah untuk fungsi yang tidak melakukan return value:

```
$this->nama_fungsi();
```

Sedangkan untuk fungsi yang melakukan return value kita perlu mendefinisikan suatu variabel sebelum memanggil fungsi tersebut untuk tempat menyimapan return value dari fungsi yang diakses. Untuk contoh pemanggilan fungsi yang memiliki return value dapat dilihat pada contoh dibawah:

```
$data = $this->nama_fungsi();
```

Untuk contoh penggunaan dari pemanggilan fungsi dalam satu controller yang sama kita masih akan menggunakan route **localhost:8000/pelanggan** namun pada **pelangganController.php** tambahkan satu fungsi bernama dataPelanggan() dan isikan script seperti dibawah:

```
public function dataPelanggan()
{
 $pelanggan = ['Ina','Ani','Ita','Indra'];      return $pelanggan;
}
```

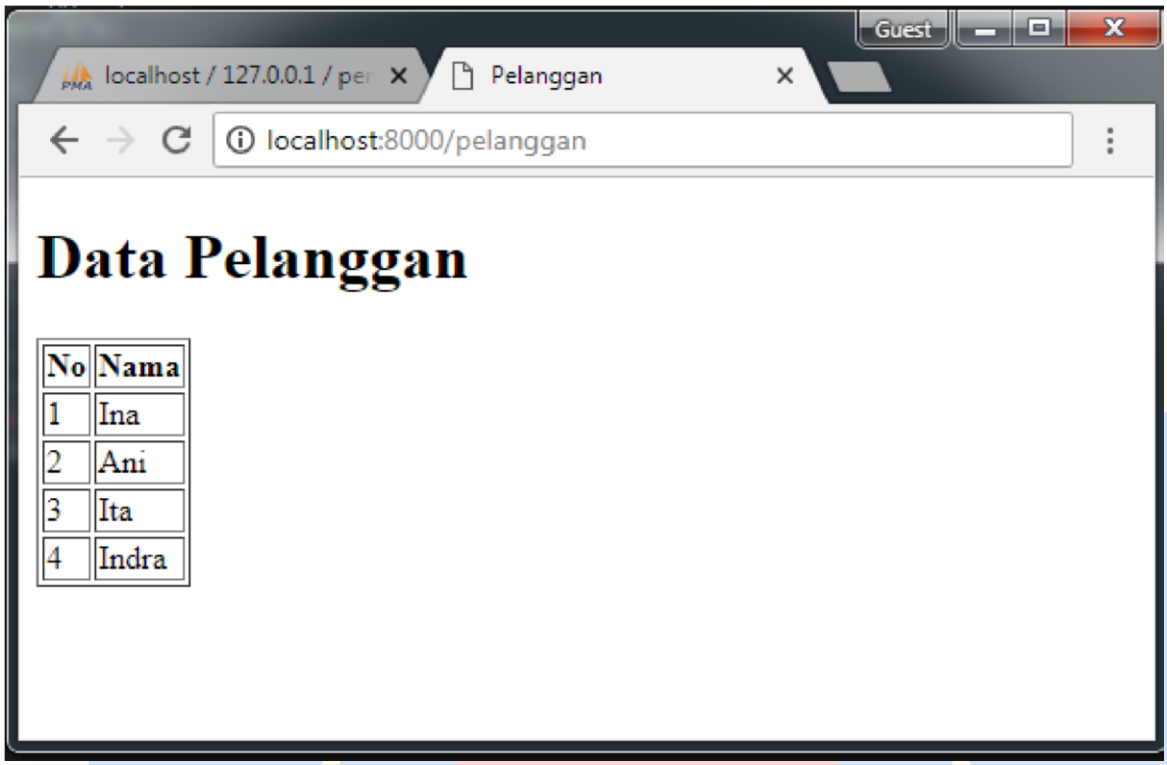
Pada fungsi dataPelanggan() diata kita mendefinisikan satu variabel array bernama pelanggan dan langsung melakukan return value variabel pelanggan tersebut, kemudian pada fungsi index() yang pada controller yang sama yaiut **pelangganController.php** kita modifikasi sedikit scriptnya menjadi seperti berikut:

```
public function index()
{
 $pelanggan = $this->dataPelanggan();
 return view('pelanggan/index',compact('pelanggan'));     }
```

Pada fungsi index() diatas kita memanggil fungsi dataPelanggan() menggunakan method **this** dan menerima nilai dari return value yang kita tampung kedalam variabel pelanggan yang ada di dalam fungsi index(). Untuk menampilkan data pelanggan yang telah kita terima dari fungsi dataPelanggan(), modifikasi view **index.blade.php** yang berada pada folder **resources/views/pelanggan** menjadi seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
	<title>Pelanggan</title>
</head>
<body>
	<h1>Data Pelanggan</h1>
	<table border="1">
		<thead>
			<tr>
				<th>No</th>
				<th>Nama</th>
			</tr>
		</thead>
		<tbody>
			@foreach ($pelanggan as $i => $v)
				<tr>
					<td>{{ $i+1 }}</td>
					<td>{{ $v }}</td>
				</tr>
			@endforeach
		</tbody>
	</table>
</body>
</html>
```

Kemudian jalankan kembali route **localhost:8000/pelanggan** maka hasilnya akan nampak seperti pada gambar dibawah:



Untuk menghasilkan controller yang lengkap, buatlah satu controller baru dengan nama produkController tapi tambahkan kata **--resource** dibelakangnya sehingga menjadi seperti berikut:

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

perintah tersebut akan menghasilkan file produkController yang lengkap dengan 7 fungsi yang dibutuhkan untuk melakukan proses CRUD pada laravel seperti berikut:

```
<?php
namespace App\Http\Controllers; use
Illuminate\Http\Request;
class produkController extends Controller
{
   public function index()
   {
      //    }
   public function create()
   {
      //
   }
   public function store(Request $request)
```

```
    {
        //
    }
    public function show($id)
    {
        //     }
    public function edit($id)
    {
        //
    }
    public function update(Request $request, $id)
    {
        //
    }
    public function destroy($id)
    {
        //
    }
}
```

Adapun 7 kegunaan dari 7 fungsi tersebut adalah sebagai berikut:

1. **index()** merupakan fungsi utama controller yang akan dipanggil ketika action tidak disebutkan. Bisanya digunakan untuk menaruh script untuk menampilkan data.
2. **create()** merupakan fungsi untuk menampilkan form tambah data.
3. **store()** merupakan fungsi untuk menaruh script untuk menyimpan data yang dikirim dari form tambah data untuk disimpan ke database.
4. **show()** merupakan funsgi untuk menaruh script yang menampilkan data lebih detail dari sebuah record.
5. **edit()** merupakan funsi untuk menaruh script untuk mengarahkan ke form edit.
6. **update()** fungsi ini digunakan untuk menauh script yang akan digunakan untuk memperbaharui data dari database yang diterima dari form edit.
7. **destroy()** untuk menaruh script yang digunakan untuk menghapus data dari database.

Kelebihan lain yang di dapat dari pembuatan controller dengan –resource adalah kita cukup membuat satu route yang mengarah ke controller tanpa harus membuat route masing-masing fungsi. Untuk membuktikannya tambah route berikut ke file web.php yang ada di folder route.

```
Route::resource('produk','produkConttroller');
```

Dengan mendefinisikan satu route di atas, Laravel akan membuatkan route untuk setiap fungsi pada controller. Untuk membuktikannya, silahkan ketikan skrip artisan berikut pada cmd atau command promt dan jangan lupa arahkan cmd ke folder projek laravel yang dibuat.

```
php artisan route:list
```

berikut dafar URL yang dihasilkan dari route diatas:

Method	URL	Action (Fungsi)
GET	/produk	index()
GET	/produk/create	create()
POST	/produk	store(Request $request)
GET	/produk/{id}	show($id)
GET	/produk/{id}/edit	edit($id)
PUT/PATCH	/produk/{$id}	update(Request $request, $id)
DELETE	/produk/{id}	destroy($id)

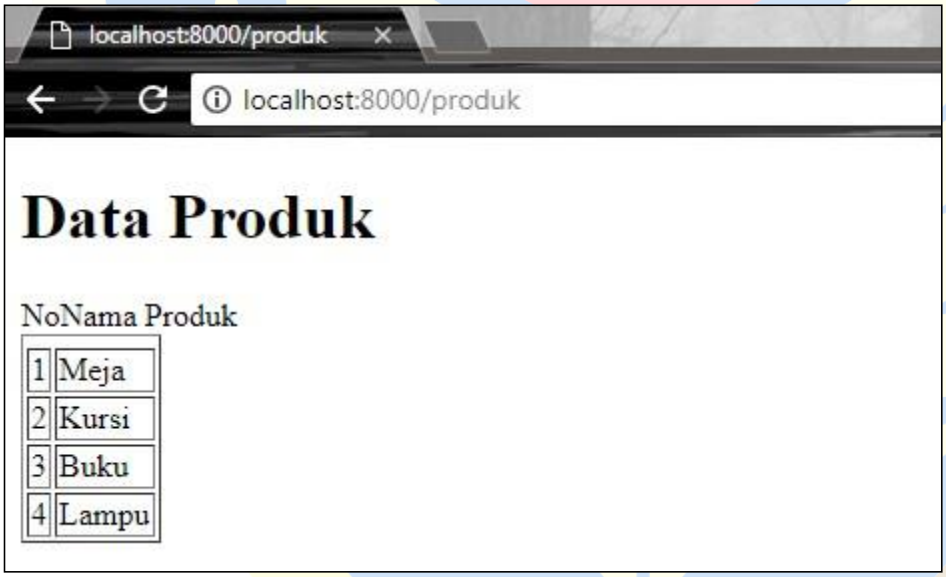
Contoh penggunaan controlller, misalnya pada fungsi **index()** dapat kita isi dengan script sebagai berikut:

```
public function index()
{
    $produk = ['Meja','Kursi','Buku','Lampu'];
    return view('produk.index',compact('produk'));
}
```

Kemudian buatlah satu view baru di folder resourcess/views/produk (jika folder produk tidak ada silahkan dibuat) yang bernama index.blade.php dan isikan script sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
<h1>Data Produk</h1>
<table border="1">
    <thead>
        <tr>No</tr>
        <tr>Nama Produk</tr>
    </thead>
    <tbody>
        @foreach ($produk as $i => $v)
            <tr>
                <td>{{ $i+1 }}</td>
                <td>{{ $v }}</td>
            </tr>
        @endforeach
    </tbody>
</table>
</body>
</html>
```

Kemudian silahkan buka **localhost:8000/produk**, maka hasilnya akan nampak seperti berikut:



Tugas

- Buatlah controller yang diperlukan untuk projek anda

MODUL VI

INTERAKSI DATABASE DENGAN QUERY BUILDER

(Pertemuan 8 dan 9)

Tujuan :

1. Mahasiswa mampu memahami interaksi database dengan framework
2. Mahasiswa mampu mengetahui penulisan *query builder* dalam interaksi database

DASAR TEORI

Query Builder merupakan salah satu cara untuk menjalankan query database dengan lebih mudah, Query Builder juga telah dilengkapi dengan fitur keamanan untuk mencegah terjadinya SQL Injextion (adalah sebuah aksi hacking yang dilakukan di aplikasi client dengan cara memodifikasi perintah SQL yang ada di memori aplikasi client). Selain itu kita dapat menggunakan query builder tanpa harus membuat model terlebih dahulu.

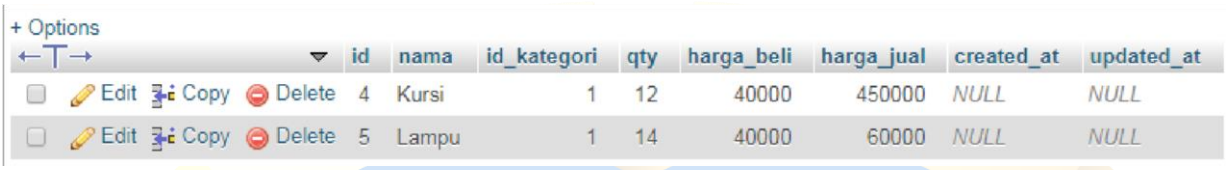
Sricpt query bulder cukup mudah dipahami, misalnya untuk menampilkan data dari tabel produk yang dibuat sebelumnya pertama-tama kita harus mendefinisikan penggunaan class dari QUERY BUILDER dengan cara menambahkan script **use DB;** pada bagian atas controller atau seperti contoh dibawah:

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request; use
DB;
class produkController extends Controller
{

}
```

Dengan pendefinisian mengunakan script **use DB;** kita telah menambahkan class QUERY BUILDER ke dalam controller produkController. Lalu untuk menampilkan data dari tabel produks yang telah dibuat sebelumnya bisa menggunakan script berikut:

```
DB::table('produks')->get();
```

Pada script diatas kita akan mengambil data dari tabel produk yang berada di dalam database penjualan yang telah dibuat sebelumnya, adapun isi dari tabel produk tersebut adalah sebagai berikut:



+ Options

	id	nama	id_kategori	qty	harga_beli	harga_jual	created_at	updated_at
Edit Copy Delete	4	Kursi	1	12	40000	450000	NULL	NULL
Edit Copy Delete	5	Lampu	1	14	40000	60000	NULL	NULL

KEGIATAN PRAKTIKUM 1 MENAMPILKAN DATA DENGAN QUERY BUILDER

Untuk mempraktekan contoh dari latihan menampilkan data dengan QUERY BUILDER pertama kita harus memeriksa file **.env** apakah projek sudah terhubung dengan database atau belum. File ini terletak pada bagian luar projek laravel yang dibuat, dan pastikan pada bagian mysql sudah terkonfigurasi seperti pada gambar dibawah:

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

Database yang akan kita gunakan adalah database dengan nama **penjualan**, yang telah kita buat pada latihan pembuatan migration sebelumnya. Kemudian pada file **produkController.php** yang berada pada folder **app\Http\Controller** telah dibuat pada latihan sebelumnya buatlah satu fungsi yang bernama **index()** atau modifikasi jika sudah ada dan tambahkan script sebagai berikut:

```
<?php
namespace App\Http\Controllers;
use   Illuminate\Http\Request;
use DB;
class produkController extends Controller
{
   public function index()
   {
    $produk  =  DB::table('produks')->get();          return
view('produk/index',compact('produk'));
   }
```

```
}
```

Fungsi index() diatas akan mengambil semua data yang ada pada tabel produk dan kemudian akan mengirimkannya ke view **index.blade.php** yang ada di dalam folder **resources/views/produk**. Jadi pada view **index.blade.php** yang ada pada folder **resourcess/views/produk** (jika belum ada silahkan dibuat) modifikasi script nya menjadi seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
	<title>Laravel Saya</title>
</head>
<body>
	<h3><b>Data Produk</b></h3>
	<table border="1">
	 <thead>
		<tr>
			<th>No</th>
			<th>Nama</th>
			<th>Kategori</th>
			<th>Qty</th>
			<th>Harga Beli</th>
			<th>Harga Jual</th>
		</tr>
	</thead>
	<tbody>
		<tr>
			@foreach ($produk as $i => $p)
				<tr>
					<td>{{ $i+1 }}</td>
					<td>{{ $p->nama }}</td>
					<td>{{ $p->id_kategori }}</td>
					<td>{{ $p->qty }}</td>
					<td>{{ $p->harga_beli }}</td>
```

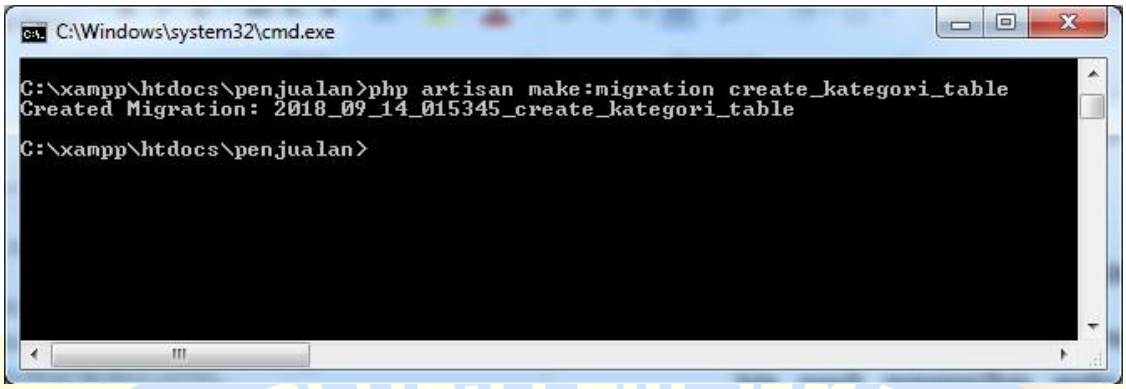
```
                                    <td>{{ $p->harga_jual }}</td>
                                </tr>
                            @endforeach
                        </tr>
                    </tbody>
                </table>
</body>
</html>
```

Sebelum mencoba menjalankan skript diatas pertama-tama cobalah untuk memeriksa route yang ada pada file **web.php** yang ada pada folder **routes/web.php** dan periksa apakah route Route::get('/produk', 'produkController@index'); sudah didefinisikan atau belum. Jika sudah maka aktifkan server artisan pada cmd lalu akses **http://localhost:8000/produk** maka hasilnya akan nampak seperti pada gambar dibawah:

KEGIATAN PRAKTIKUM 2 JOIN TABEL DENGAN QUERY BUILDER

Jika diperhatikan pada tampilan data produk pada tabel diatas pada kolom id_kateogri kita masih menampilkan angka atau bukan data dari suatu kategori produk. Untuk memperbaikinya buatlah satu tabel baru dengan nama kateogri menggunakan fasilitas migration pada Laravel dengan tahapan sebagai berikut:

1. Jalankan perintah artisan berikut pada cmd atau comand promt yang sudah terarah ke dalam directori projek laravel yang digunakan seperti pada contoh gambar:

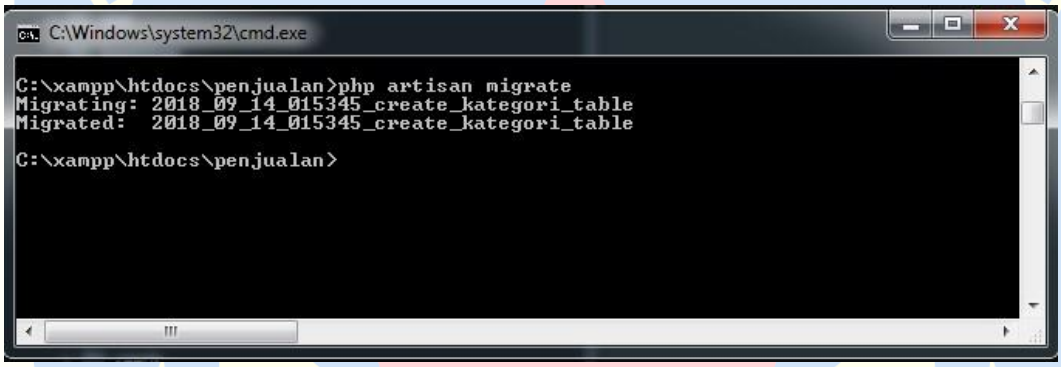


2. Langkah kedua bukalah file **2018_09_14_015345_create_kategori_table.php** yang ada folder **database/migration** dan modifikasi koding di dalamnya menjadi seperti dibawah:

```
<?php  use Illuminate\Support\Facades\Schema; use
Illuminate\Database\Schema\Blueprint;          use
Illuminate\Database\Migrations\Migration;


class CreateKategoriTable extends Migration
{
   /**
* Run the migrations.
*
* @return void
    */
   public function up()
   {
      Schema::create('kategori', function (Blueprint $table) {
         $table->increments('id');
         $table->string('kategori');
         $table->string('keterangan');
         $table->timestamps();
      });
   }
   /**
* Reverse the migrations.
*
```

```
* @return void
    */
    public function down()
    {
        Schema::dropIfExists('kategori');
    }
}
```

3. Jalankan perintah php artisan migrate seperti pada gambar dibawah.



4. Tambahkan beberapa data ke dalam tabel kateogri secara manual melalui phpMyAdmin sehingga akan tambak seperti pada gambar dibawah:

Setelah membuat tabel kategori beserta dengan 1 contoh datanya sekarang bukalah class Controller **produkController.php** yang ada pada folder **app\Http\Controller** dan modifikasi fungsi index() yang ada di dalam controller tersebut menjadi seperti dibawah:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request; use
DB;
class produkController extends Controller
{
    public function index()
    {
```

```
    $produk = DB::table('produks')
            ->join('kategori','produks.id_kategori','=','kategori.id')
            ->get();
    return view('produk/index',compact('produk'));     }
}
```

Kemudian pada view index.blade.php yang ada pada folder **resources/views/produk/** modifikasi kodingnya menjadi seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
        <title>Laravel Saya</title>
</head>
<body>
        <h3><b>Data Produk</b></h3>
        <table border="1">
         <thead>
                <tr>
                        <th>No</th>
                        <th>Nama</th>
              <th>Kategori</th>
                        <th>Qty</th>
                        <th>Harga Beli</th>
                        <th>Harga Jual</th>
                </tr>
        </thead>
        <tbody>
                <tr>
                        @foreach ($produk as $i => $p)
                                <tr>
                                        <td>{{ $i+1 }}</td>
                                        <td>{{ $p->nama }}</td>
                                        <td>{{ $p->kategori }}</td>
```

```
                                    <td>{{ $p->qty }}</td>
                                    <td>{{ $p->harga_beli }}</td>
                                    <td>{{ $p->harga_jual }}</td>
                                </tr>
                            @endforeach
                        </tr>
                    </tbody>
                </table>
</body>
</html>
```

Selain menggunakan cara diatas, ada berbagai variasi dalam menampilkan data menggunakan query builder. Berikut ini adalah berbagai script untuk menampilkan data dengan Query Builder:

QUERY BUILDER	KETERANGAN
count();	Menampilkan jumlah data
max('nama_kolom');	Menampilkan nilai maksimal dari sebuah kolom. Dengan cara yang sama juga kita dapat menggunakan min(), avg(), sum().
select('kolom1','kolom2','kolom3 as kolomketiga','kolom ke n')->get();	Memilih beberaa kolom dan membuat alias untuk kolom
select(db::raw('count(*) as total'))>get();	Menggunakan row expression
where('nama_kolom','=,'value')	Menampilkan data dengan kondisi tertentu, selain menggunakan = kita juga dapat menggunakan > , < , >= , <=, <> dan like
where(['kolom1','>,'valie'], ['kolom2','<','value'])->get();	Menampilkan data dengan beberapa kondisi.
where('kolom1','='value')->orwhere('kolom2','=','value')->get();	Menggunakan or dalam menampikan data dengan beberapa kondisi.
whereBetween ('nama_kolom',[parameter1,parameter2]) ->get();	Menampilkan data dengan nilai suatu kolom di antara 2 batas nilai.
whereNotBetween	

(‘nama_kolom’,[parameter1,parameter2]) ->get();	Menampilkan data dengan nilai suatu kolom tidak di antara 2 batas nilai.
WhereIn(‘nama_kolom’,[‘parameter1’,’	Menampilkan data dengan nilai suatu kolom
parameter2’,’ parameter3’])->get();	sesuai yang disebutkan. Untuk kebalikanya dapat menggunakan whereNotIn().
WhereNull(‘nama_kolom’)->get();	Menampilkan data dengan nilai suatu kolom null. Untuk kebalikanya dapat menggunakan whereNotNull().
WhereDate(‘nama_kolom_tanggal’,201712-12)->get();	Menampilkan data dengan membandingkan nilai tanggal. Juga terdapat pilihan lain seperti whereMonth(), whereYear(), dan whereDay().
whereColumns(‘kolom1’,’<’,’kolom2’)>get();	Menampilkan data dengan membandingkan dua kolom.
orderBy(‘nama_kolom’,’desc’)->get();	Menampilkan data dengan urutan.
inRandomOrder()->get();	Menampilkan data dengan urutan acak.
latest()->get();	Menampilkan data terbaru. Kebalikanya dapat menggunakan oldest().
groupBy()->get();	Menampilkan data menggunakan grup
limit(10)->offset(5)->gey();	Menampilan data dengan batas dan offset tertentu. Alternatif lain dapat menggunakan take() dan skip().
join(‘nama_table2’,’ nama_tabel1.primary_key’,’=’,’ nama_tabel2.foreginkey’)->get();	Menampilkan tabel dengan join.
leftjoin(‘nama_table2’,’ nama_tabel1.primary_key’,’=’,’ nama_tabel2.foreginkey’)->get();	Menampilkan data dengan leftjoin, terdapat juga pilihan rightjoin().

MENAMBAH DATA DENGAN QUERY BUILDER

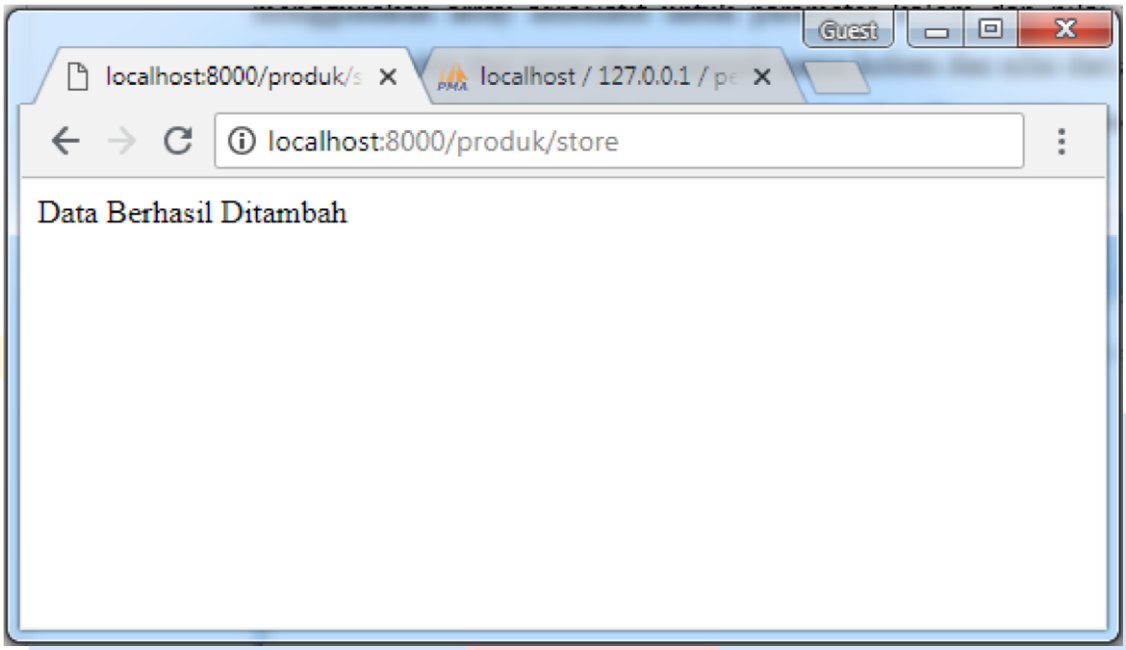
Untuk menambahkan data menggunakan Query Builder buatlah satu route baru pada file **web.php** yang terletak pada folder **routes/web.php** dengan script sebagai berukut:

```
Route::get('/produk/store', 'produkController@store');
```

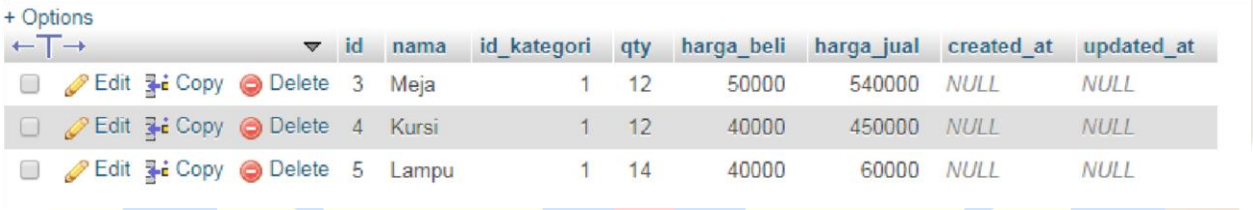
Pada route diatas akan menjalankan fungsi store yang ada di dalam class Controller produkController.php, untuk itu bukalah file produkController yang ada pada folder **App/Http/Controller** dan tambahkan satu fungsi yang bernama **store()** dan isikan script sebagai berikut:

```
  public function store()
  {
     DB::table('produks')
     ->insert([
        'nama' => 'Lampu',
        'id_kategori' => 1,
        'qty' => 14,
        'harga_beli' => 40000,
        'harga_jual' => 60000,
     ]);

     echo "Data Berhasil Ditambah";
}
```

Pada script diatas adalah script untuk menambah data ke tabel yang ada di dalam database, pada kasus diatas tabel yang akan ditambah datanya adalah tabel produks. Laravel sendiri menggunakan array assosiatif untuk parameter kolom dan nilai yang akan ditambah dengan ketentuan index dari array akan menjadi nama kolom dan nilai dari array akan menjadi nilai yang akan disimapn ke kolom. Jika kita menjalankan **localhost:8000/produk/store** pada browser akan muncul tampilan seperti berikut:



Dan bila kita memerika tabel produk yang ada di dalam database penjualan maka akan ada satu data baru seperti pada gambar dibawah:



MEMPERBAHARUI DATA QUERY BUILDER

Untuk memperbaharui (update) data, kita harus menentukan terlebih dahulu data mana yang akan di update. Pada contoh ini kita akan memperbaharui data yang baru saja ditambahkan yaitu data dengan id 5, pertama buatlah satu route baru di fiile **web.php** di folder **routes/web.php** dengan skrip sebagai berikut:

```
Route::get('/produk/update', 'produkController@update');
```

Pada route diatas jika kita membuka **localhost:8000/produk/update** maka kita akan dibawa ke function **update()** yang ada pada file produkController.php, maka dari itu buatlah satu function baru bernama update di file **produkController.php** yang ada pada folder **App/Http/Controller** dengan script sebagai berikut:

```
public function update()
{
   DB::table('produks')->where('id',3)
   ->update([
```

```
        'nama' => 'Bola Lampu',
        'qty' => 20,
        'harga_beli' => 45000,
        'harga_jual' => 55000,
    ]);
    echo "Data Berhasil Diperbaharui";
}
```

Pada script diatas merupakan script untuk memperbaharui data menggunakan Query Builder. Pertama kita harus menentukan data mana yang akan diperbaharui dengan menggunakan fungsi where() dan diikuti dengan fungsi update() untuk memperbaharui data yang ada pada kolom. Dan jika dilihat pada tabel produk yang ada pada database penjualan data dengan id 3 akan berubah:

+ Options

	id	nama	id_kategori	qty	harga_beli	harga_jual	created_at	updated_at
Edit Copy Delete	3	Bola Lampu	1	20	45000	55000	NULL	NULL
Edit Copy Delete	4	Kursi	1	12	40000	450000	NULL	NULL
Edit Copy Delete	5	Lampu	1	14	40000	60000	NULL	NULL

MENGHAPUS DATA QUERY BUILDER

Untuk menghapus data kita perlu menentukan terlebih dahulu data mana yang akan dihapus lalu diikuti dengan fungsi **delete().** Pertama buatlah route seperti berikut pada file web.php yang ada pada folder routes:

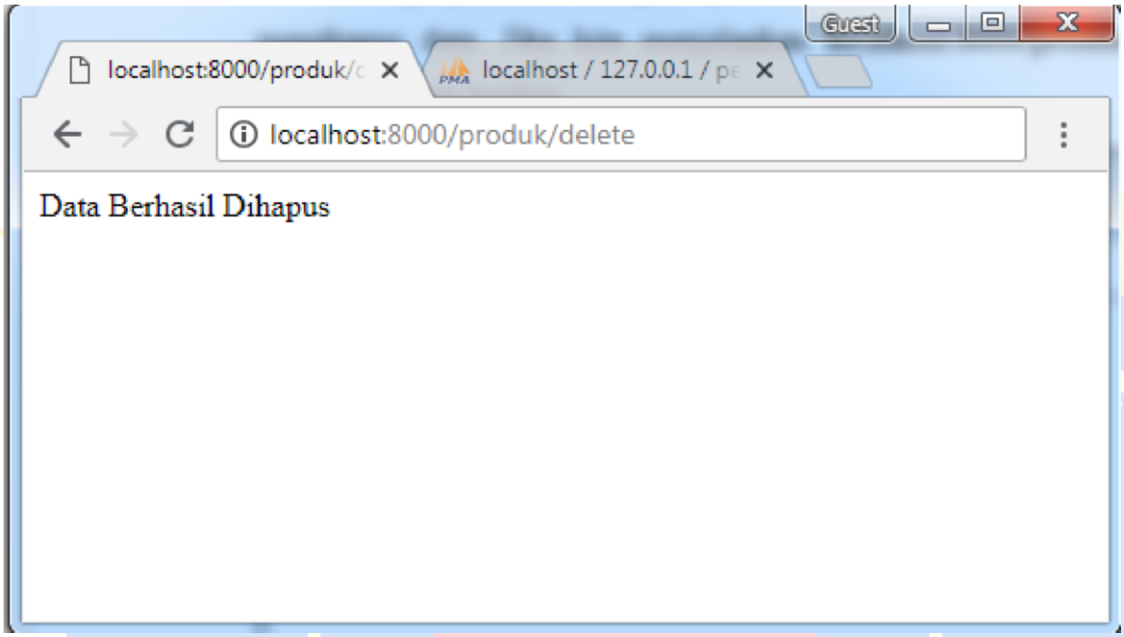
```
Route::get('/produk/delete', 'produkController@delete');
```

Pada route diatas jika kita mengakses **localhost:8000/produk/delete** pada browser maka kita akan diarahkan ke fungsi delete yang ada pada file produkController, maka dari itu buatlah fungsi yang bernama delete pada file **produkController.php** dan isikan script sebagai berikut:

```
  public function delete()
  {
     DB::table('produks')->where('id',3)->delete();          echo "Data
Berhasil Dihapus";     }
```

Pada script diatas adalah script untuk menghapus data menggunakan Query Builder. Pertama kita harus menentukan terlebih dahulu data mana yang akan dihapus menggunakan fungsi where() untuk

menghapus data dengan syarat tertentu. Lalu diikuti dengan fungsi delete() untuk menghapus data. Jika kita menjalankan **localhost:8000/produk/delete** maka akan muncul halaman seperti berikut:



Dan jika dilihat pada tabel produks data dengan id 3 telah dihapus:

+ Options

	id	nama	id_kategori	qty	harga_beli	harga_jual	created_at	updated_at
Edit Copy Delete	4	Kursi	1	12	40000	450000	NULL	NULL
Edit Copy Delete	5	Lampu	1	14	40000	60000	NULL	NULL

Tugas

- Implementasikan query builder dalam projek anda

MODUL VII

INTERAKSI DATABASE DENGAN ELOQUENT ORM

(Pertemuan 10)

Tujuan :

1. Mahasiswa mampu memahami interaksi database dengan framework
2. Mahasiswa mampu mengetahui penggunaan *eloquent* dalam interaksi database

DASAR TEORI

Eloquent ORM (Object Relational Mapping) merupakan teknik untuk memetakan basis data relasional ke model objek. Berinteraksi dengan database seperti menampilkan, menambah, mengubah atau menghapus data menggunakan Eloquent ORM lebih disarankan walaupun kita dapat menggunakan Query Builder tanpa membuat model terlebih dahulu.

Hal yang perlu diperhatikan sebelum menggunakan Qloquent ORM untuk berintaksi dengan database adalah model, secara default eloquent akan menganggap nama class model adalah sebagai nama tabel dengan menambahkan ‘s’ dibelakangnya. Secara default eloquent juga akan beramsumi kolom primary key dari tabel akan bernama id, serta eloquent juga kan beramsumi kita ingin mengetahui kapan suatu data ditambah dan diperbaharui maka dari itu kita perlu menyiapkan dua kolom tambahan yang bernama **created_at** dan **updated_at** yang bertipe datetime. Pada contoh ini kita akan menggunakan tabel kategori yang telah dibuat sebelumnya.

Pertama-tama periksalah model dari tabel kateogri pada folder **app/** jika model belum dibuat maka buatlah menggunakan perintah artisan make:model atau seperti berikut:

```
php artisan make:model kategori
```

perintah tersebut akan menghasilkan file model baru yang bernama **kategori.php** pada folder **app/**. Bukalah dan modifikasi file model **kateogri.php** tersebut menjadi seperti berikut:

```
<?php
namespace App;
 use Illuminate\Database\Eloquent\Model;
 class kategori extends Model
{
   protected $guarded = ['id','created_at','updated_at'];        public
$table = 'kategori'; }
```

ELOQUNET mengaharuskan kita mendefinisikan kolom-kolom yang kita gunakan, namun kita tidak perlu mendefinisikan semua kolom, hanya kolom yang boleh diisi oleh pengguna (fillable) atau kolom yang tidak boleh diisi oleh pengguna (guarded). Pada script diatas kita mendifinisikan kolom guarded atau kolom yang tidak boleh diisi oleh pengguna dan secara otomatis sisa kolom yang tidak didefinisikan akan termasuk ke kolom fillable atau boleh diisi oleh pengguna. Selain itu kita juga mendefinisikan satu variabel bernama **table** dengan nilai variabel tersebut adalalah kategori, hal ini dilakukan karena tabel dengan nama kategori yang telah kita buat tidak memenuhi syarat nama tabel dengan menggunakan ELOQUENT maka dari itu kita bisa mengakalinya dengan mendefinisikan variabel tersebut di dalam model.

KEGIATAN PRAKTIKUM MENAMPILKAN DATA DENGAN ELOQUENT ORM

Untuk menampilkan data kategori produk dengan ELOQUENT pertama-tama kita harus membuat sebuah route pada file **web.php** yang ada pada folder **routes/web.php** dengan skript sebagai berikut:

```
Route::get('/kategori', 'kategoriController@index');
```

Pada route diatas jika kita mengakses link **localhost:8000/kategori** maka kita akan diarahkan ke fungsi **index()** yang ada pada file **kategoriController.php**. Jadi karena controller kategoriController.php belum dibuat maka buatlah dengan menggunakan perintah artisan berikut pada cmd atau comand prompt:

```
php artisan make:controller kategoriController
```

selanjutnya bukalah **file kategoriController.php** yang berada pada folder **app\Http\Controller** dan pada bagian atas controller tambahkan script **use App\kategori;** menambahkan skript tersebut berarti kita telah menambakan class model kateogri ke dalam class controller kategoriController yang artinya untuk berinteraksi dengan tabel kateogri kita hanya perlu memanggil nama dari modelnya saja.

Untuk menampilkan data kateogri produk buatlah satu fungsi bernama **index()** pada **kategoriController.php** dengan skript sebagai berikut:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request; use
App\kategori;
class kategoriController extends Controller
{
   public function index()
```

```
    {
     $kategori_produk = kategori::all();
     return view('kategori/index',compact('kategori_produk'));
    }
}
```

Fungsi index() diatas akan menampilkan view **index.blade.php** yang berada pada folder **resources/views/kategori**, karena view ini belum dibuat maka buatlah terlebih dahulu dengan cara membuat file baru pada folder **resources/views/kategori** dengan nama **index.blade.php** dan isikan skript seperti dibawah:

```
<!DOCTYPE html>
<html>
<head>
        <title>Laravel</title>
</head>
<body>
<h2><b>Data Kategori Produk</b></h2>
<table border="1">
        <thead>
                <tr>
                        <th>No</th>
                        <th>Nama Kategori</th>
                        <th>Keterangan</th>
                        <th>Dibuat</th>
                        <th>Terakhir Diperbaharui</th>
                </tr>
        </thead>
        <tbody>
                @foreach ($kategori_produk as $i => $k)
                        <tr>
                                <td>{{ $i+1 }}</td>
                                <td>{{ $k->kategori }}</td>
                                <td>{{ $k->keterangan }}</td>
```

```
                        <td>{{ $k->created_at }}</td>
                        <td>{{ $k->updated_at }}</td>
                    </tr>
                @endforeach
            </tbody>
</table>
</body>
</html>
```

Sekarang jika kita mengakses link **localhost:8000/kategori** (pastikan server artisan sudah berjalan) maka halaman browser akan menampilkan halaman seperti berikut:

Pada Eloqunet ORM untuk melakukan operasi tambah, edit dan menghapus data tedapat dua cara yang pertama menggunakan Eloqunet ORM biasa atau menggunakan Eloquent ORM Mass Assignment. Perbedaan dari kedua cara ini terlihat pada penulisan syntaknya.

MENAMBAH DATA

Untuk menambahkan data menggunakan Eloquent ORM langkah-langkahnya sama dengan Query Builder. Pertama siapkan route seperti gambar dibawah.

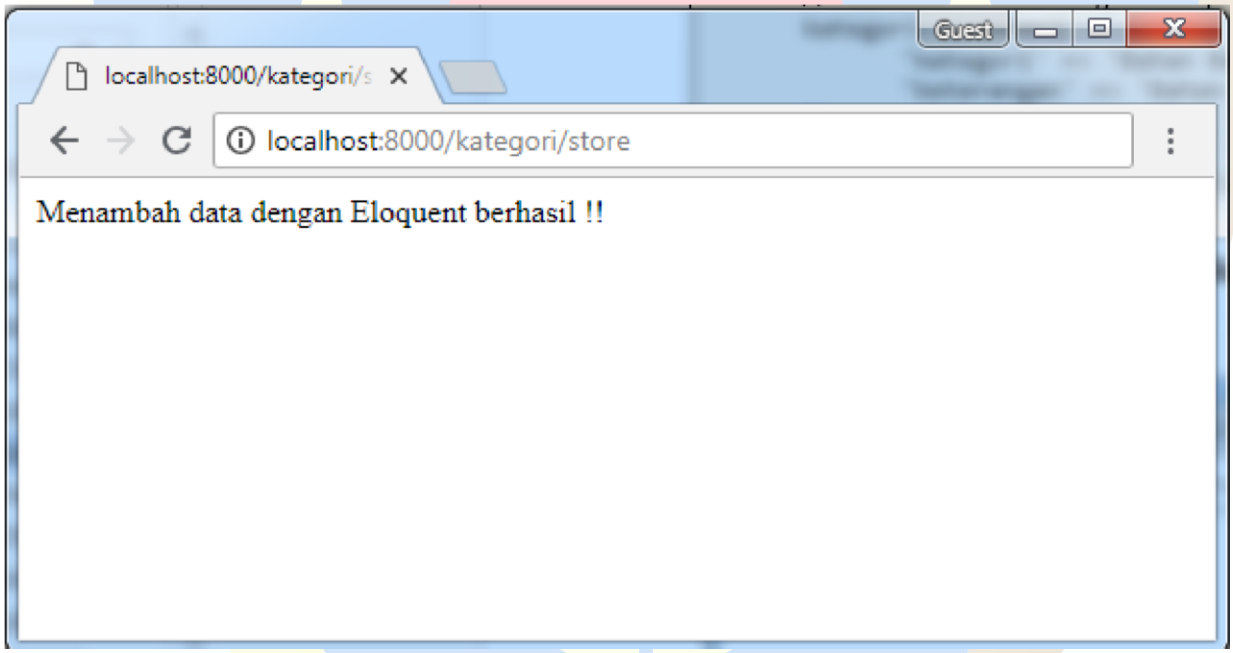
```
Route::get('/kategori/store', 'kategoriController@store');
```

Lalu pada file kategoriController.php buatlah satu fungsi baru bernama **store()** dan tambahkan koding dengan script berikut:

```
    public function store()
```

```
{
 // menambah data dengan Eloquent ORM biasa
 $kategori = new kategori;
 $kategori->kategori = 'Alat Dapur';
 $kategori->keterangan = 'Alat Daput 1';
 $kategori->save();
 // menambah data dengan Eloquent ORM Mass Assignment      kategori::create([
       'kategori' => 'Bahan Bangunan',
       'keterangan' => 'Bahan Banguanan Atas',
 ]);
 echo "Menambah data dengan Eloquent berhasil !!";     }
```

Kemudian cobalah jalankan route **localhost:8000/kategori/store** maka tampilanya akan menjadi seperti berikut:



Dan jika dilihat pada database penjualan di tabel kategori maka akan ada 2 data baru seperti pada gambar dibawah:

	id	kategori	keterangan	created_at	updated_at
Edit Copy Delete	1	Perlengkapan Rumah Tangga	Perlengkapan Rumah Tangga	NULL	NULL
Edit Copy Delete	2	Alat Dapur	Alat Daput 1	2018-09-14 07:49:25	2018-09-14 07:49:25
Edit Copy Delete	3	Bahan Bangunan	Bahan Banguanan Atas	2018-09-14 07:49:25	2018-09-14 07:49:25

Dapat dilihat pada data dengan id no 2 dan 3 adalah hasil dari script menyimpan data menggunakan ELOQUENT. Kelebihan dari Eloquent sendiri kolom created_at dan updated_at akan terisi otomatis tergantung dari kapan data itu dibuat dan kapan data itu diperbaharui.

MEMPERBAHARUI DATA

Untuk memperbaharui data menggunakan ELOQUENT langkah-langkahnya sama dengan Query Builder yaitu pertama kita harus mendefinisikan data mana yang akan di update dan diikuti oleh nilai baru dari kolom.

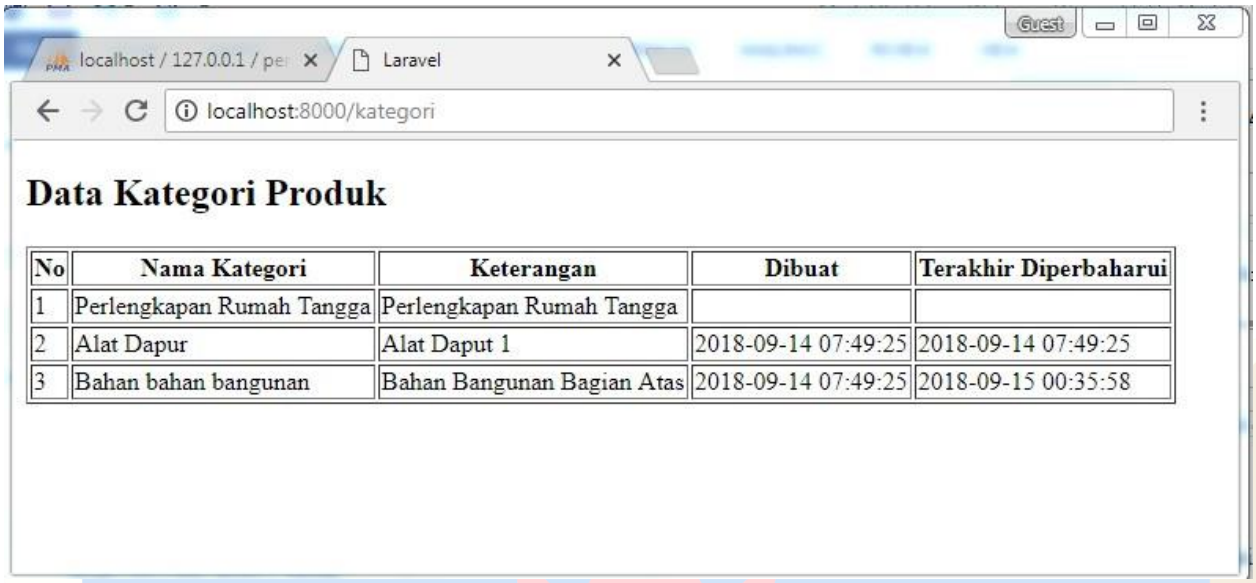
Untuk mempraktekannya buatlah satu route baru pada file **web.php** yang ada pada folder **routes** seperti berikut:

```
Route::get('/kategori/update', 'kategoriController@update');
```

Route diatas akan mengarahkan kita ke fugsi **update**() jika kita mengakses link **localhost:8000/kateogri/update** maka dari itu masuklah ke file **kategoriController.php** yang berada pada folder **app\Http\Controller** dan buat satu fungsi baru bernama update() dan isikan script seperti berikut:

```
public function update()
{
 // update data dengan Eloquent ORM biasa
 $kategori = kategori::where('id',3)->first();
 $kategori->kategori = 'Bahan bahan bangunan';
 $kategori->keterangan = 'Bahan Bangunan Bagian Atas';
 $kategori->save();


 return redirect('/kategori');     }
```

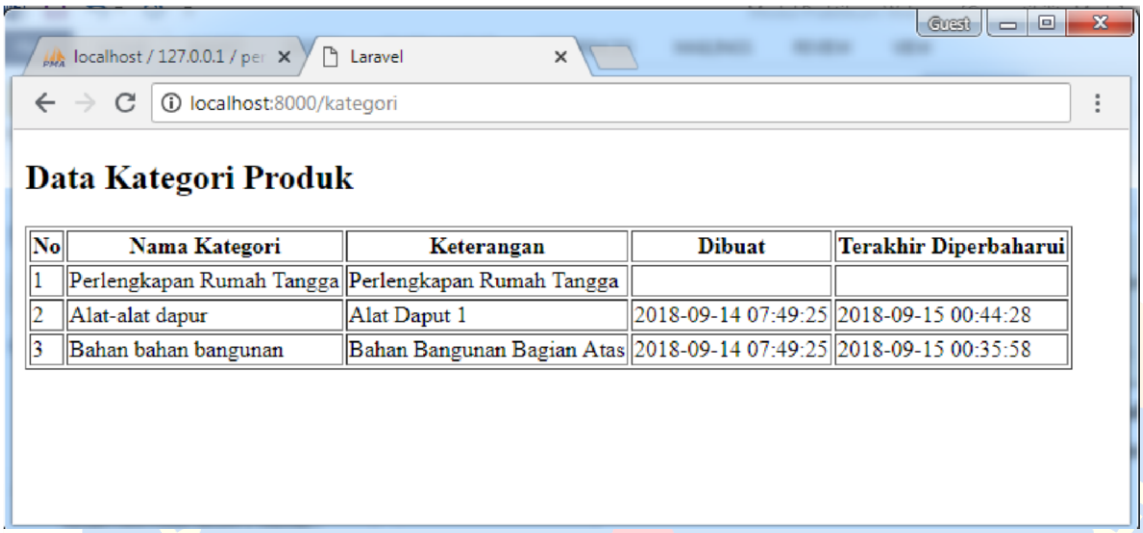
Pada script diatas adalah sintak untuk memperbaharui data menggunakan Eloquent biasa, dan jika diperhatikan kita langsung melakukan **redirect** atau langsung mengalihkan halaman ke route **localhost:8000/kateogri**. sekarang jika kita menjalankan route **localhost:8000/kateogri/update** maka hasilnya kita akan dibawa kembali ke halaman kateogri dengan data kategori yang telah diperbaharui seperti pada gambar dibawah:



Update data menggunakan Eloquent juga memiliki metode Mass Assignment yatu dimulai dari mendefinisikan data mana yang akan di update yang diikuti oleh array assosiatif sebagai nama kolom dan value yang akan diperbaharui. Untuk mencobanya ubahlah fungsi **update()** yang ada pada file **kategoriContoller.php** menjadi seperti berikut:

```
   public function update()
   {
    // update data dengan Eloquent ORM biasa
    // $kategori = kategori::where('id',3)->first();
    // $kategori->kategori = 'Bahan bahan bangunan';
    // $kategori->keterangan = 'Bahan Bangunan Bagian Atas';              //
$kategori->save();
    // update data dengan Eloquent ORM Mass Assignment      kategori::where('id',2)
    ->update([
          'kategori' => 'Alat-alat dapur',
    ]);
    return redirect('/kategori');     }
```

Selanjutnya jalankan kembali route **localhost:8000/kategori/update**. Maka kita akan diarahkan kembali ke halaman kategori dengan data nama kategori dengan id 2 yang telah diperbaharui seperti pada gambar dibawah:



MENGHAPUS DATA

Untuk menghapus data menggunakan Eloquent kita harus mendefinisikan terlebih dahulu data mana yang akan dihapus dan diikuti dengan fungsi **delete**() untuk menghapus data tersebut dari tabel. Untuk mencobanya buatlah satu route baru di file **web.php** dengan script sebagai berikut:

```
Route::get('/kategori/delete', 'kategoriController@delete');
```

Pada route diatas kita akan diarahkan ke fungsi delete() yang ada pada file **kateogriController.php** jika kita mengakses route **localhost:8000/kategori/delete**. Selanjutnya buatlah satu fungsi baru bernama **delete()** pada file **kateogriController.php** dan isikan script seperti dibawah.

```
public function delete()
{
 $kategori = kategori::where('id',1)->first();
 $kategori->delete();
 return redirect('/kategori');     }
```

Kemudian jalankanlah route **localhost:8000/kateogri/delete** untuk mengesekusi method tersebut maka kita akan diarahkan ke kembali ke route **localhost:8000/kategori** dengan data dengan kateogri dengan id 1 yang telah dihapus seperti pada gambar dibawah:

Dan jika diperiksa pada tabel kategori yang ada pada penjualan maka data dengan id 1 telah dihapus.

Untuk menghapus data ELOQUENT juga memiliki metode Mass Assignment dimana sintaknya diawali dengan penulisan model dan diikuti dengan mendefinisikan data mana yang dihapus dan diakhiri dengan fungsi delete(). Untuk mencobanya ubahlah fungsi **delete**() yang ada pada file **kategoriController.php** menjadi seperti berikut:

```
public function delete()
{
 // $kategori = kategori::where('id',1)->first();
 // $kategori->delete();
 // update data dengan Eloquent ORM Mass Assignment
kategori::where('id',2)->delete();  return redirect('/kategori');  }
```

Lalu cobalah jalankan route **localhost:8000/kategori/delete** sekali lagi. Script diatas akan menghapus data yang ada pada tabel kategori yang memiliki id 2. Jika dilihat pada tabel kategori yang ada pada database penjualan maka akan data dengan id 2 telah dihapus:

Tugas

- Gunakan eloquent dalam projek anda

MODUL VIII

OPERASI CRUD PADA LARAVEL

(Pertemuan 11,12,13)

Tujuan :

1. Mahasiswa mampu memahami proses Create, Read, Update, Delete (CRUD) data pada framework
2. Mahasiswa mampu memahami konfigurasi database
3. Mahasiwa mampu memahami pengunaan *authentication*
4. Mahasiwa mampu memahami pembuatan tabel dengan *migration*
5. Mahasiswa mampu memahami pembuatan model dengan *eloquent*
6. Mahasiwa mampu mengimplementasikan *controller* dan *route*

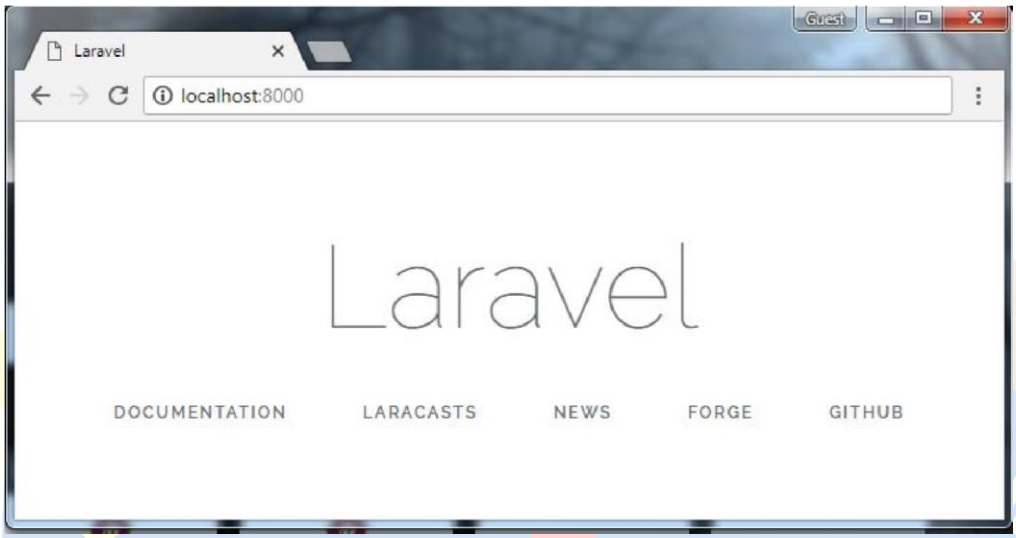
DASAR TEORI

Pada materi-materi sebelumnya, kita telah mempelajari fitut-fitur penting pada Laravel. Agar lebih paham pada bagian ini akan membahas tentang contoh CRUD (create, read, update dan detele) pada Laravel. Dengan pembahasan ini diharapkan para mahasiswa akan semakin mengerti hubungan anatara masing-masing komponen yang telah dipelajari pada bagian bagian sebelumnya.

Contoh CRUD yang akan dibuat adalah tabel produk yang berada pada database penjualan. Tabel produk itu sendiri akan berelasi dengan tabel kategori produk. Untuk membuat contoh CRUD pada tabel produk kita perlu membuat route, model, controller, dan view. Untuk memulainya silahkan buat satu projek laravel baru dan beri nama penjualan dengan cara menjalan script berikut pada cmd atau comand prompt yang sudah terarah ke folder htodocs yang ada pada xampp.

```
composer create-project --prefer-dist laravel/laravel latihan_penjualan "5.4.*"
```

Kemudian jika proses instalasi telah selesai cobalah jalankan projek laravel yang baru dibuat dengan nama latihan_penjualan menggunakan server artisan dengan menjalankan script berikut pada cmd **php artisan serve**. Sebelum menjalankan script **php artisan serve** pastika bahwa directory cmd sudah berada pada folder projek latihan_penjualan laravel yang telah dibuat, kemudian pada browser akses lah link **localhost:8000** dan pastikan projek laravel sudah tampil seperti pada gambar dibawah:



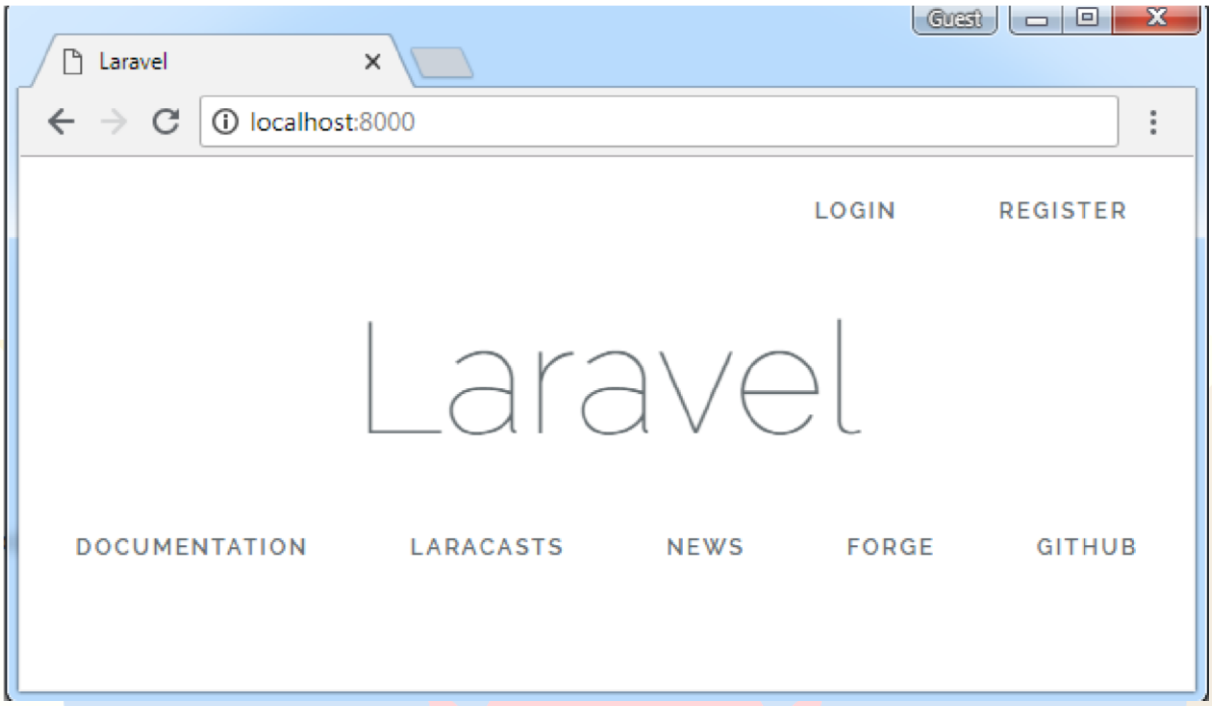
MEMBUAT AUTHENTICATION PADA LARAVEL

Authentication merupakan fitur yang disediakan laravel untuk membatasi akses penggunaan aplikasi melalui login. Fitur ini juga telah dilengkapi dengan fasilitas register dan reset password. Dengan fitur ini kita tidak perlu lagi membuat tampilan halaman login dari awal, cukup modifikasi pada bagian yang diperlukan terutama pada bagian desain.

Untuk mengaktifkan fitur ini cukup jalankanlah perintah atisan dibawah pada comand prompt yang sudah diarahkan ke projek laravel penjualan.

```
php artisan make:auth
```

untuk membuktikan pengaktifan fitur Authentication telah berhasil, silahkan buka halaman awal Laravel dengan terlebih dahulu menjalankan perintah `php artisan serve` dan bukalah alamat localhost:8000 pada browser dan hasilnya akan menjadi seperti berikut:



Hasilnya ialah akan ada tombol login ataupun register disebelah kanan atas pada halaman awal Laravel. Jika tulisan login atau register di klik maka kita akan diarahkan ke form register ataupun login, namun login atau register tersebut belum dapat digunakan karna belum ada tabel untuk menyimpan data pengguna. Maka dari itu ikuti dulu materi selanjutnya tentang membuat tabel dengan migration dibawah.

MEMBUAT TABEL DENGAN MIGRATION

Untuk membuat tabel produk dan kategori dengan migration langkah pertama ialah membuat file migration itu sendiri. Pertama kita akan membuat file migration untuk tabel produk dengan menjalankan script berikut pada cmd:

```
php artisan make:migration create_produks_table
```

kemudian modifikasi file migration dengan nama create_produks_table.php yang ada pada folder database/migrations menjadi seperti berikut:

```
<?php
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateProduksTable extends Migration
```

```
{
    /**
 * Run the migrations.
 *
 * @return void
     */
    public function up()
    {
        Schema::create('produks', function (Blueprint $table) {
            $table->increments('id');
            $table->string('nama');
            $table->integer('id_kategori');
            $table->integer('qty');
            $table->integer('harga_beli');
            $table->integer('harga_jual');
            $table->timestamps();
        });
    }
    /**
 * Reverse the migrations.
 *
 * @return void
     */
    public function down()
    {
        //
    }
}
```

Kemudian buatlah satu file migration baru untuk tabel kategori produk dengan menggunakan perintah artisan sebagai berikut:

```
php artisan make:migration create_kategori_table
```

kemudian modifikasi file migration dengan nama create_kategori_table.php yang ada pada folder database/migrations menjadi seperti berikut:

```
<?php
use Illuminate\Support\Facades\Schema; use
Illuminate\Database\Schema\Blueprint; use
Illuminate\Database\Migrations\Migration;
class CreateKategoriTable extends Migration
{
   /**
* Run the migrations.
*
* @return void
    */
   public function up()
   {
      Schema::create('kategoris', function (Blueprint $table) {
         $table->increments('id');
         $table->string('nama');
         $table->timestamps();
      });
   }

   /**
* Reverse the migrations.
*
* @return void
    */
   public function down()
   {
      //
   }
}
```

Sebelum menjalankan perintah **php artisan migrate** untuk melakukan migrasi rancangan database ke database sql buatlah satu database baru dengan nama latihan_penjualan dan konfigurasi database pada file .env yang ada pada projek Laravel **latihan_penjualan/.env** menjadi seperti pada gambar dibawah:

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

Jika database sudah terkonfigurasi Sebelum menjalankan perintah migration kita harus melakukan sedikit konfigurasi pada file **database.php** yang ada pada folder projek laravel yang baru dibuat atau pada folder **latihan_penjualan/config/database.php** pada baris ke 53 ubahlah value strict dari true menjadi false seperti pada script dibawah :

```
        'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
```

Kemudian sekarang jalankan perintah **php artisan migrate** pada cmd atau comand prompt yang sudah terarah pada projek laravel latihan_penjualan yang telah dibuat, maka rancangan database yang

dibuat dengan migration tadi akan dibawa ke database mysql dan jika dilihat pada localhost/phpmyadmin akan ada tabel produks dan tabel kategori.

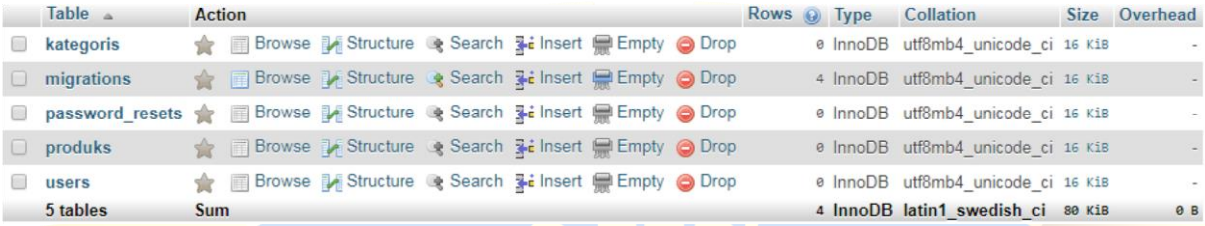


Table	Action	Rows	Type	Collation	Size	Overhead
kategoris	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
migrations	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	16 KiB	-
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
produks	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
users	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16 KiB	-
5 tables	Sum	4	InnoDB	latin1_swedish_ci	80 KiB	0 B

Dapat dilihat pula tabel users, migrations dan password resets, yang dimana tabel users tersebut adalah tabel berfungsi untuk menyimpan data pengguna yang melakukan register menggunakan fitur Authentication yang dibuat pada tabel sebelumnya dan sekarang fitur login dan register dari laravel siap digunakan. Lakukanlah register untuk membuat akun baru agar bisa mengakses halaman home untuk admin.

MEMBUAT MODEL ELOQUENT

Pada modul sebelumnya kita telah mempelajari fitur-fitur Laravel yang salah satunya adalah pembuatan model untuk mengakses database menggunakan Eloquent ORM. Karena pada kasus ini kita akan menggunakan tabel produks dan tabel kategori maka kita harus mendefinisikan 2 model yaitu untuk tabel produks dan tabel kategoris.

Pertama jalankanlah perintah artisan berikut pada cmd yang sudah terarah ke projek laravel penjualan:

```
php artisan make:model produk
```

perintah artisan diatas akan menghasilkan file model **produk.php** yang terletak pada folder app/ bukalah file tersebut dan modifikasi menjadi seperti gambar dibawah:

```
<?php  namespace App;
use Illuminate\Database\Eloquent\Model;
class produk extends Model
{
    protected $guarded = ['id','created_at','upadated_at'];
    public function kategori()
    {
     return $this->hasOne('App\kategori', 'id', 'id_kategori');
    }
```

```
}
```

Pada script produk.php diatas ada satu fungsi bernama kateogri, fungsi ini berfungsi untuk merelasikan tabel produk dengan tabel kateogri.

Langkah kedua adalah membuat model untuk tabel kategoris. Jalankan perintah artisan dibawah untuk membuat model tersebut:

```
php artisan make:model kategori
```

kemudian bukalah file model **kateogri.php** dan modifikasi menjadi seperti berikut:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class kategori extends Model
{
   protected $guarded = ['id','created_at','upadated_at']; }
```

Langkah selanjutnya adalah membuat controller produkController, namun sebelum membuat controller tersebut tambahkanlah beberapa data ke dalam tabel kateogri menggunakan fitur seeder dari laravel dengan cara mengeksekusi perintah artisan berikut:

```
php artisan make:seeder kategoriTableSeeder
```

perintah diatas akan menghasilkan file dengan nama **kategoriTableSeeder.php** yang terletak di folder **database/seeds.** bukalah file tersebut dan modifikasi menjadi seperti dibawah:

```
<?php
use    Illuminate\Database\Seeder;    use
App\kategori;
class kategoriTableSeeder extends Seeder
{
   /**
* Run the database seeds.
*
* @return void
    */
```

```
    public function run()
    {
     kategori::create([
            'nama' => 'Makanan'
     ]);
     kategori::create([
            'nama' => 'Perlengkapan Rumah Tangga'
     ]);
     kategori::create([
            'nama' => 'Alat Belajar'
     ]);
    }
}
```

Sebelum mengeksekusi seeder diatas bukalah file **DatabaseSeeder.php** yang berada pada folder yang sama dengan tabel seeder yang baru saja dibuat dan modifikasi fungsi **up()** menjadi seperti berikut:

```
<?php
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    /**
*  Run the database seeds.
*
*  @return void
     */
    public function run()
    {
        $this->call(kategoriTableSeeder::class);
    }
}
```

Kemudian sekarang barulah jalankan perintah artisan php artisan db:seed untuk mengeksekusi seeder yang baru saja dibuat. Dan jika dilihat pada database latihan_penjualan tabel kateogri maka akan muncul contoh data baru seperti pada gambar dibawah:

	id	nama	created_at	updated_at
Edit Copy Delete	5	Makanan	2018-08-12 14:11:25	2018-08-12 14:11:25
Edit Copy Delete	6	Perlengkapan Rumah Tangga	2018-08-12 14:11:25	2018-08-12 14:11:25
Edit Copy Delete	7	Alat Belajar	2018-08-12 14:11:25	2018-08-12 14:11:25

MEMBUAT CONTROLLER DAN ROUTES

Langkah selanjutnya adalah membuat sebuah controller untuk melakukan proses CRUD untuk tabel produk. Pada controller ini kita akan membuat controller dengan tag –resource yang akan menghasilkan controller dengan fungsi-fungsi secara otomatis. Untuk membuat controller tersebut jalankanlah perintah artisan dibawah pada comand prompt:

```
php artisan make:controller produkController --resources
```

perintah artisan diatas akan menghasilkan controller **produkController.php** pada folder **app\Http\Controller** lengkap dengan 7 fungsi yang umum digunakan pada proses CRUD.

Bukalah file controller tersebut dan tambahkan script berikut pada bagian atas controler.

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\produk;
7
```

Script **use App\produk;** tersebut berfungsi untuk mengkoneksikan produkContoller.php yang baru saja dibuat dengan model produk untuk tabel produks yang ada pada database.

Setelah membuat controller selanjutnya adalah mendefinisikan route untuk mengakses tiap-tiap fungsi yang ada pada controller produkController. Karena kita menggunakan –tag resource maka kita hanya perlu mendefinisikan satu route untuk mengakses semua fungsi yang ada pada controller produkController. Untuk mendefinisikannya bukalah file **web.php** yang ada pada folder **routes** dan tambahkan script seperti dibawah untuk mendefinisikan route untuk mengakses fungsi-fungsi yang ada pada controller produkController:

```php
<?php

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

Route::resource('/produk', 'produkController');
```

MENAMPILKAN DATA

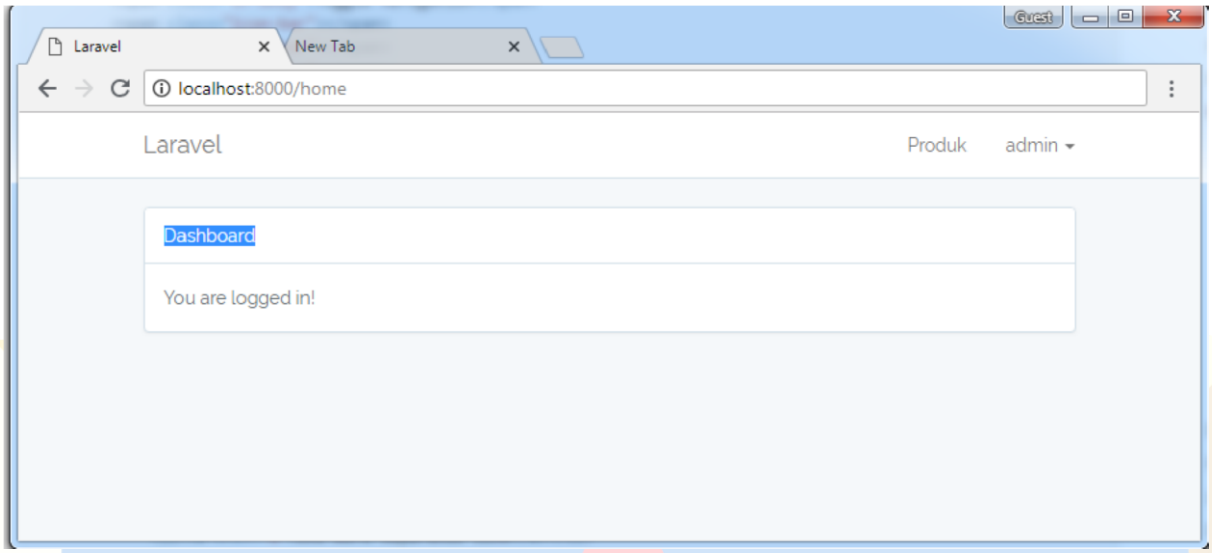
Untuk menampilkan data yang ada pada tabel bukalah file produkController.php dan modifikasi fungsi **index()** menjadi seperi berikut:

```php
public function index()
{
    $produk = produk::all();
    return view('produk.index');
}
```

Setelah memodifikasi fungsi index yang ada pada controller langkah selanjutnya adalah memodifikasi tampilan Dashboard setelah login untuk menambah menu produk agar kita bisa mengakses menu untuk melakukan proses CRUD untuk tabel produk dengan cara bukalah file app.blade.php yang ada pada folder **resources/views/layouts/** dan modifikasi pada bagian class collapse navbar-collapse menjadi seperti berikut :

```
<div class="collapse navbar-collapse" id="app-navbar-collapse">
    <!-- Left Side Of Navbar -->
    <!-- Right Side Of Navbar -->
    <ul class="nav navbar-nav navbar-right">

        <!-- Authentication Links -->
        @if (Auth::guest())
            <li><a href="{{ route('login') }}">Login</a></li>
            <li><a href="{{ route('register') }}">Register</a></li>
        @else
            <ul class="nav navbar-nav">
                <li><a href="{{ route('produk.index') }}">Produk<span class="sr-only"></span></a></li>
            </ul>
            <li class="dropdown">
                <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-expanded="false">
                    {{ Auth::user()->name }} <span class="caret"></span>
                </a>

                <ul class="dropdown-menu" role="menu">
                    <li>
                        <a href="{{ route('logout') }}"
                            onclick="event.preventDefault();
                                     document.getElementById('logout-form').submit();">
                            Logout
                        </a>
```

Dan jika dijalankan pada browser akan menjadi seperti berikut:



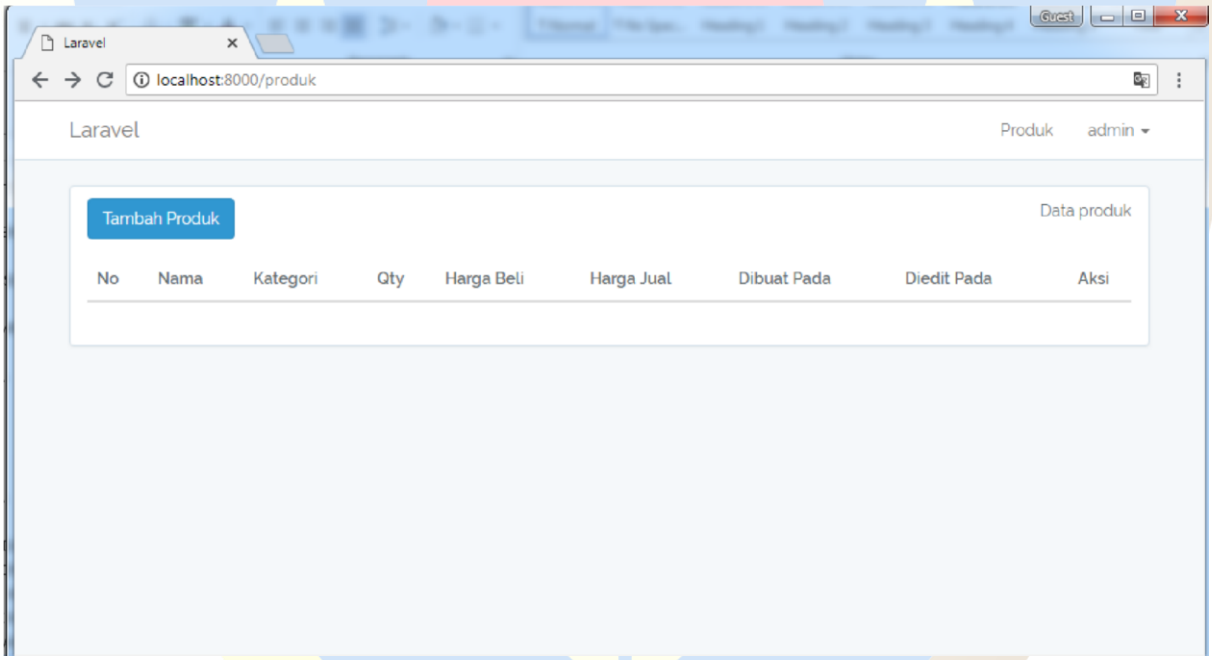
Kemudian buatlah folder baru di dalam folder **resources/views** dengan nama **produk** dan buatlah file dengan nama **index.blade.php** yang berisikan script sebagai berikut:

```
@extends('layouts.app')
@section('content')
<div class="container">
   <div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
                <div class="panel-head container-fluid" style="margin-top: 10px;">
                                <a href="{{ route('produk.create') }}"              class="btn     btn-
primary">Tambah              <div class="pull-right">
Produk</a>                              <p>Data produk</p>
                                </div>

                                ass="panel-body">
                        </div>  <table class="table table-striped">
                        <div
                        cl                      <thead>
                                                <tr>
                                                        <th>No</th>
                                                        <th>Nama</th>
                                                        <th>Kategori</th>
                                                        <th>Qty</th>
                                                        <th>Harga Beli</th>
```

```
                    <th>Harga Jual</th>
                    <th>Dibuat Pada</th>
                    <th>Diedit Pada</th>
                    <th     colspan="3"
style="text-align:
center;">Aksi</th>
 </tr>
 </thead>
 <tbody>
 @foreach ($produk as $i => $p)
        <tr>
        <td>{{ $i+1 }}</td>
        <td>{{ $p->nama }}</td>
        <td>{{ $p->Kategori->nama }}</td>
        <td>{{ $p->qty }}</td>
        <td>{{ $p->harga_beli }}</td>
        <td>{{ $p->harga_jual }}</td>
        <td>{{ $p->created_at }}</td>
         <td>{{ $p->updated_at }}</td>
          <td><a href="{{
route('produk.show,$p->id) }}" class="btn btn-
warning"> Detail</a></td>

        <td><a          class="btn      btn-
success"  href="{{  route('produk.edit',$p->id)
}}"> Edit</a></td>
                <td>
                <form  method="post"
action="{{ route('produk.destroy',$p->id) }}">
                {{ csrf_field() }}
                        <input      type="hidden"
name="_method" value="DELETE">
        <button         class="btn    btn-danger"
type="submit">Hapus</button>
```

```
                    </form>
                </td>
                </tr>
                @endforeach
                                </tbody>
                            </table>
                    </div>
                </div>
            </div>
        </div>
    </div>
    @endsection
```

Maka ketika kita memilih menu produk yang ada pada halaman dashboard akan muncul halaman seperti berikut:



MENAMBAH DATA

Untuk melakukan penambahan data produk langkah pertama yang harus kita lakukan adalah menampilkan form untuk menambah data, maka dari itu bukalalah controller produkController dan modifikasi fungsi **create()** yang ada di dalamnya menjadi seperti berikut:

```
public function create()
{
```

```
    $kategori = \App\kategori::all();
    return view('produk.create',compact('kategori'));     }
```

Kemudian buatlah satu file baru pada folder **resources/views/produk/** dengan nama **create.blade.php** dan isikan script sebagai berikut:

```
@extends('layouts.app') @section('content')
<div class="container">
   <div class="row">
    <div class="col-md-12">
         <div class="panel panel-default">
                  <div class="panel-head container-fluid" style="margin-top: 10px;">
                           <p>Tambah Data produk</p>
                  </div>
                  <div class="panel-body">
      <form class="form-horizontal" action="{{ route('produk.store')
}}" method="post">
                                    {{ csrf_field() }}
                                    <div class="form-group">
                                    <label class="control-label col-sm-2">Nama
Produk</label>
                                     <div class="col-sm-10">
                                    <input type="text" class="form-control" name="nama">
                                     </div>
                                    </div>
                                    <div class="form-group">
                                   <label class="control-label col-sm-2">Kategori
Produk</label>
                                     <div class="col-sm-10">
                                     <select class="form-control" name="kategori">
                                     <option value="">Pilih
Kateogri</option>
                                     @foreach ($kategori as $k)
                                    <option value="{{ $k->id
```

```
}}">{{ $k->nama }}</option>
                                    @endforeach
                                                </select>
                                    </div>
                                    </div>
                                    <div class="form-group">
                                    <label class="control-label col-sm-2">Qty
Awal</label>
                                    <div class="col-sm-10">
                                    <input type="text" class="form-control" name="qty">
                                    </div>
                                    </div>
                                    <div class="form-group">
                                    <label class="control-label col-sm-2">Harga
Jual</label>
                                    <div class="col-sm-10">
                                    <input type="text" class="form-control" name="jual">
                                    </div>
                                    </div>
                                    <div class="form-group">
                                    <label class="control-label col-sm-2">Harga
Beli</label>
                                    <div class="col-sm-10">
                                    <input type="text" class="form-control" name="beli">
                                    </div>
                                    </div>
                                    <div class="form-group">
                                    <div class="col-sm-offset-2 col-sm-10">
                                                <button type="submit" class="btn
btnprimary">Simpan</button>
                                        </div>
                                    </div>
                            </form>
```

```
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

Jika diperhatikan pada tag form pada bagian action terdapat script yang bertuliskan {{route('produk.store')}}, script tersebut merupakan fungsi yang digunakan untuk mengakses suatu route berdasarkan **name** yang telah diberikan pada suatu route. Sedangkan route **produk.store** dihasilkan dari penggunaan **Route::resource** daftar route dan namanya dapat dilihat menggunakan cmd dengan perintah artisan **php artisan route::list**.

Jika diperhatikan juga terdapat script bertuliskan **{{ csrf_field() }}**, script ini digunakan untuk membuat hidden field yang berisi token **CRSF**. Fungsinya untuk memberikan keamanan dengan token pada setiap form, pada Laravel token ini wajib disiikan pada setiap form.

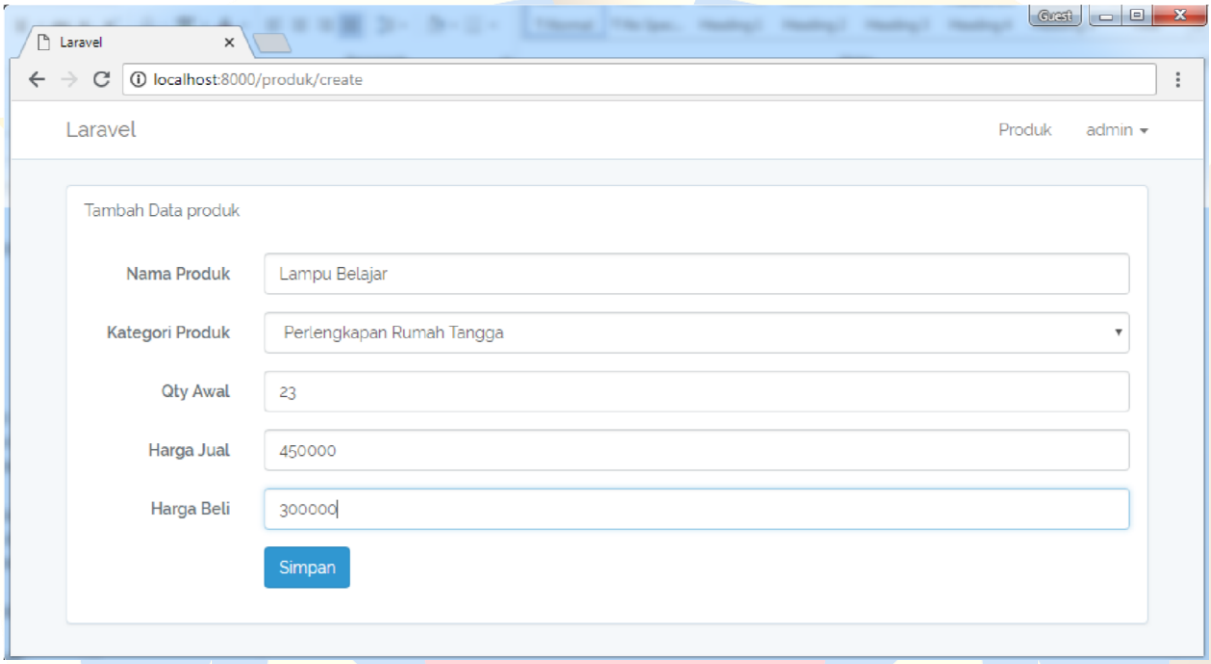
Langkah selanjutnya adalah memberikan script untuk menyimpan data produk ke dalam database. Pada atribut action yang ada pada form tambah produk kita menuliskan **{{route('produk.store')}}** route tersebut akan membara data yang diinputkan di form ke dalam **produkController.php** ke dalam fungsi store, jadi bukalah controller ProdukController kemudian modifikasi fungsi store menjadi seperti berikut:

```
public function store(Request $request)
{
    produk::create([
        'nama' => $request->nama,
        'id_kategori' => $request->kategori,
        'qty' => $request->qty,
        'harga_beli' => $request->beli,
        'harga_jual' => $request->jual,
    ]);
    return redirect()->route('produk.index');     }
```

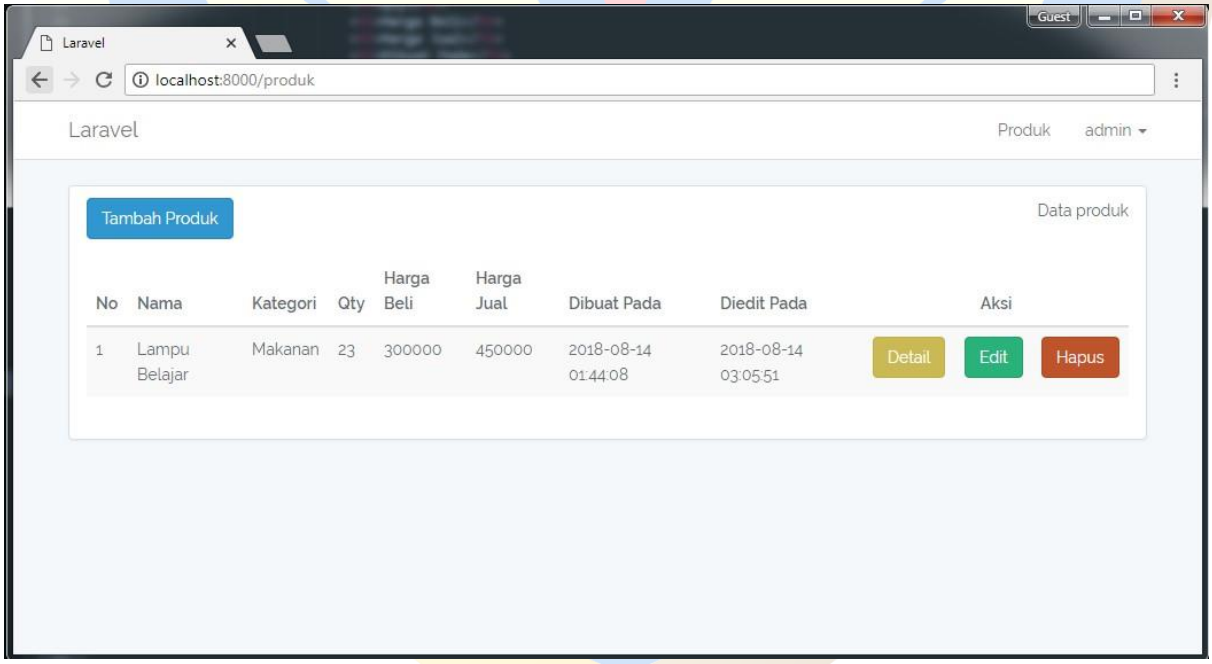
Pada fungsi store di bagian parameter kita memanggil fungsi bernama ***Request***, fungsi tersebut digunakan untuk menangkap data inputan dari form lalu akan disimpan ke dalam **$request**. Untuk script menambah data kita menggunakan **Eloquent ORM**, pada bagian akhir fungsi kita melakukan

return nilai yakni **redirect()->route('produk.index');** script tersebut adalah fasilitas yang disediakan Laravel untuk mengarahkan kita secara otomatis ke dalam route tertentu dan pada kasus diatas kita akan diarahkan ke kembali ke halaman index dari produk dan.

Sekarang form tambah produk sudah siap digunakan, silahkan lakukan uji coba pada form.



Pada bagian diatas jika kita menekan tombol simpan maka halaman akan langsung diarahkan ke halaman utama produk atau halaman index produk dengan satu data baru di tabelnya.



MEMPERBAHARUI DATA

Untuk dapat memperbahari data pada Laravel sama dengan menambah data kita harus menampilkan form edit data terlebih dahulu, namun perbedaannya kita harus menampilkan nilai lama dari data yang ingin diedit, maka dari itu bukalah file controller produkController dan modifikasi pada bagian fungsi **edit()** menjadi seperti berikut:

```
public function show($id)    {
$produk = produk::where('id',$id)->first();                return
view('produk.show',compact('produk'));    }
```

Pada script fungsi edit() diatas kita mengambil data produk dari tabel produk menggunakan model Eloquent dengan kondisi id dari produk sama dengan parameter yang diberikan pada link edit pada tombol edit yang ada di halaman utama produk dan kemudan kita melakukan return view untuk menampilkan view **edit.blade.php** yang ada di dalam folder **resourece/views/produk/.** Karena file tersebut belum ada maka buatlah satu file baru dengan nama **edit.blade.php** pada folder **resourece/views/produk/** dan isikan script sebagai berikut:

```
@extends('layouts.app') @section('content')
<div class="container">
   <div class="row">
    <div class="col-md-12">
         <div class="panel panel-default">
                  <div class="panel-head container-fluid" style="margin-top: 10px;">
                           <p>Tambah Data produk</p>
                  </div>
                  <div class="panel-body">
                           <form  class="form-horizontal"        action="{{
route('produk.update',$produk->id) }}" method="post">
                                    {{ csrf_field() }}
                                    <input type="hidden" name="_method" value="PUT">
                                    <div class="form-group">
        <label class="control-label col-sm-2">Nama
Produk</label>
                                             <div class="col-sm-10">
```

```
                                        <input type="text" class="form-control"
name="nama" value="{{ $produk->nama }}">
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Kategori
Produk</label>
                                    <div class="col-sm-10">
                                        <select class="form-control"
name="kategori">
          <option value="">Pilih
Kateogri</option>
                                            @foreach ($kategori as $k)
          <option value="{{ $k->id }}"
@if ($produk->id_kategori==$k->id) selected @endif>{{ $k->nama }}</option>
                                            @endforeach
                                        </select>
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Qty
Awal</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="qty" value="{{ $produk->qty }}">
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Harga
Jual</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="jual" value="{{ $produk->harga_jual }}">
```

```
</div>
                                    </div>
                                    <div class="form-group">
        <label class="control-label col-sm-2">Harga
Beli</label>
                                        <div class="col-sm-10">
                                            <input type="text" class="form-control"
name="beli" value="{{ $produk->harga_beli }}">
                                        </div>
                                    </div>
                                    <div class="form-group">
                                        <div class="col-sm-offset-2 col-sm-10">
                                            <button type="submit" class="btn
btnprimary">Perbaharui Data</button>
                                        </div>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
</div>
@endsection
```

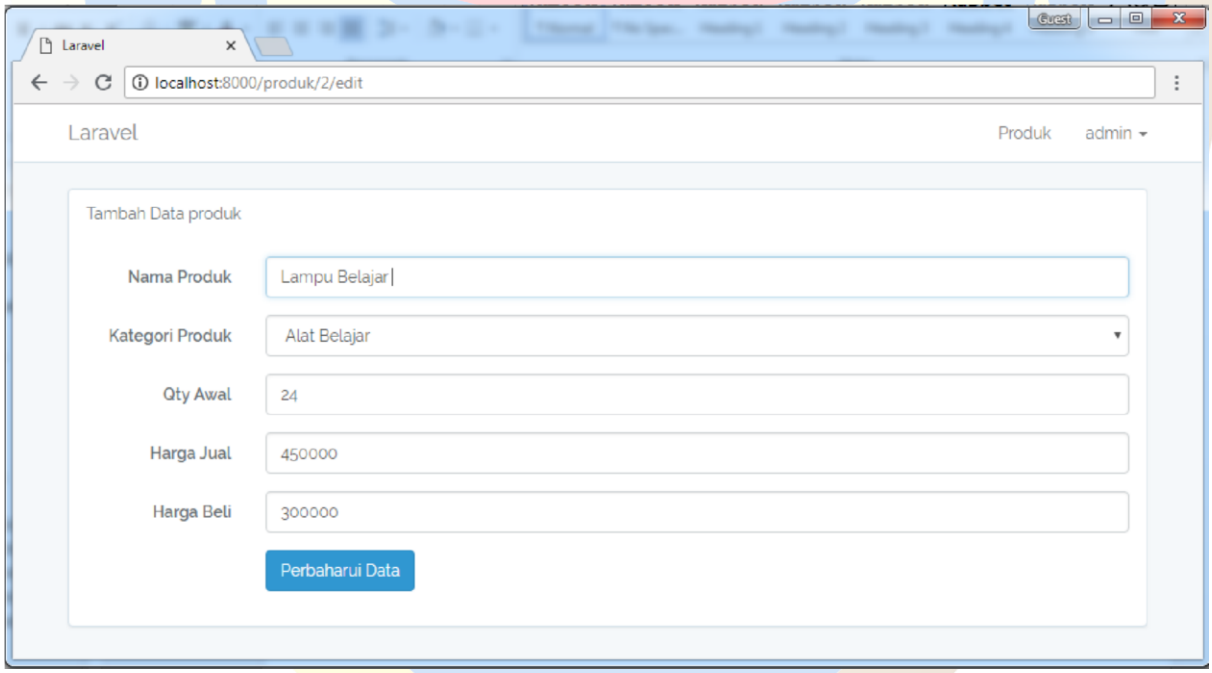
Jika diperhatikan di dalam tag form terdapat script <input type="hidden" name="_method" value="PUT"> field ini digunakan untuk menandakan bahwa data pada form akan dibawa ke dalam route produk.upadate yang method nya adalah PUT yang dimana route tersebut telah disediakan secara otomatis karena kita memakai Route::resources. Selanjutnya bukalah file controller **produkController.php** dan modifikasi pada bagian fungsi **update()**; menjadi seperti berikut:

```
    public function update(Request $request, $id)
    {
        produk::where('id',$id)
        ->update([
            'nama' => $request->nama,
```

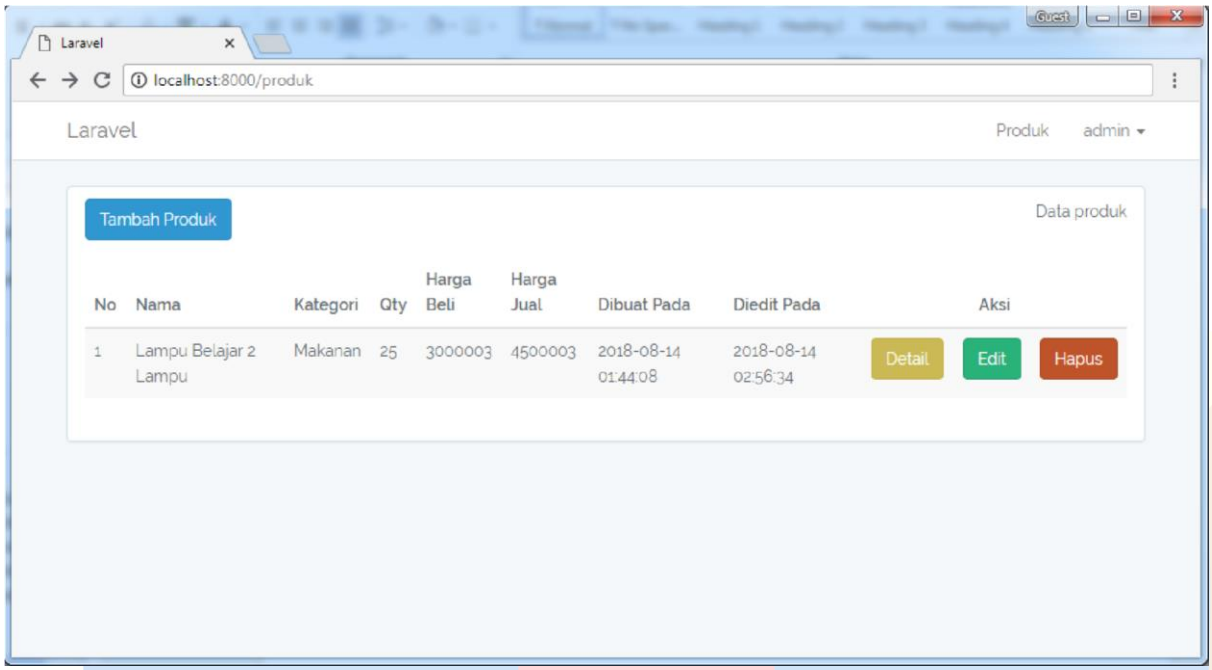
```
        'id_kategori' => $request->kategori,
        'qty' => $request->qty,
        'harga_beli' => $request->beli,
        'harga_jual' => $request->jual,
    ]);
    return redirect()->route('produk.index');     }
```

Pada fungsi update kita menerima parameter berupa data dari form edit yang kita tangkap menggunakan fungsi *Request* yang kemuidan juga kita simpan ke dalam variabel request serta kita juga mengirim data id dari produk yang kita edit yang juga kita tangkap pada bagian paramter fungsi edit yang kita beri nama $id. Setelah itu kita menyimpan nilai baru dari data produk menggunakan eloquent ORM. Setelah menyimpan nilai baru tersebut kita melakukan return **redirect()->route('produk.index');** yang akan secara otomatis langsung mengarahkan kita ke route produk.index atau halaman utama produk.

Langkah selanjutnya adalah melakukan uji coba untuk memperbaharui data. Silakan pilih menu edti dari data produk yang baru ditambah tadi. Kemudian akan muncul form seperti pada gambar dibawah:



Kemudian lakukan perubahan pada masing-masing kolom dan klik tombol perbaharui data. Maka data perubahan akan disimpan dan kita akan diarahkan kembali ke halaman utama produks dengan data produk yang telah diperbaharui:



MENAMPILKAN DETAIL DATA

Untuk menampilkan detail dari suatu data kita bisa menggunakan fungsi show() pada controller produkController, jika perhatikan kita telah menyiapkan satu tombol untuk mengakses fungsi show tersebut yang bernama detail. Tombol tersebut akan mengarahkan kita ke route produk.show dengan satu parameter yaitu id produk. Dan untuk menampilkan detail dari produk tersebut modifikasi fungi show() tersebut menjadi seperti berikut:

```
   public function show($id)
{
      $produk = produk::where('id',$id)->first();           return
view('produk.show',compact('produk'));     }
```

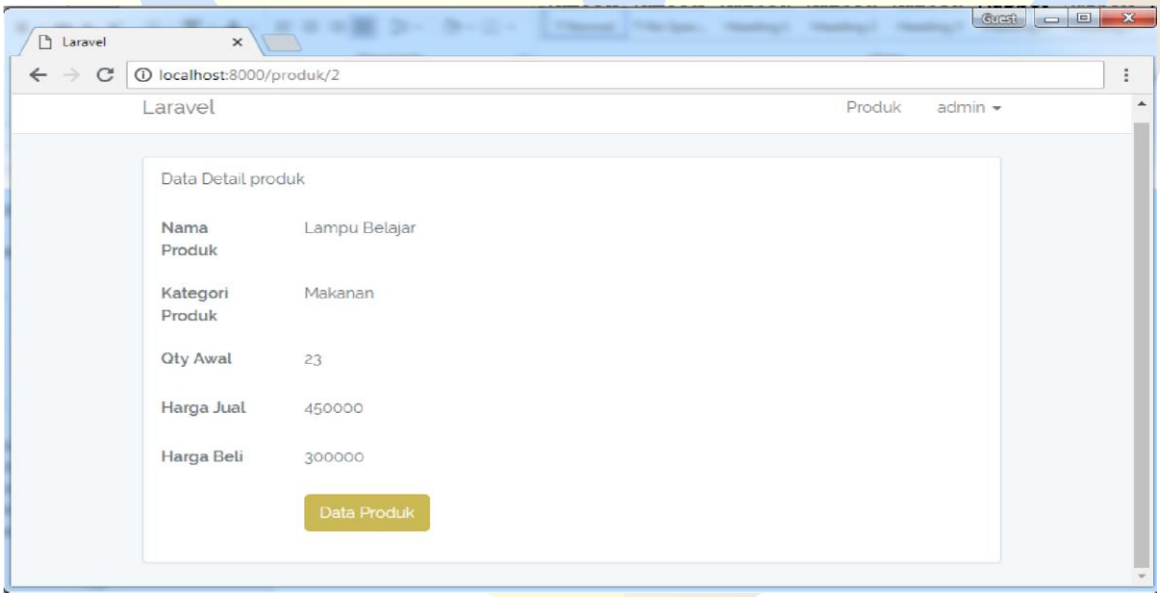
Kemudian buatlah satu file baru pada folder **resourece/views/produk/** dengan nama **show.blade.php** dan isi script sebagai berikut:

```
@extends('layouts.app') @section('content')
<div class="container">
   <div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
                <div class="panel-head container-fluid" style="margin-top: 10px;">
                    <p>Data Detail produk</p>
```

```
            </div>
            <div class="form-horizontal">
                <div class="panel-body">
                    {{ csrf_field() }}
                    <input type="hidden" name="_method" value="PUT">
                    <div class="form-group">
                        <label class="col-sm-2">Nama Produk</label>
                        <div class="col-sm-10">
                            <p>{{ $produk->nama }}</p>
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2">Kategori Produk</label>
                        <div class="col-sm-10">
                            {{ $produk->kategori->nama }}
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2">Qty Awal</label>
                        <div class="col-sm-10">
                            <p>{{ $produk->qty }}</p>
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2">Harga Jual</label>
                        <div class="col-sm-10">
                            <p>{{ $produk->harga_jual }}</p>
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2">Harga Beli</label>
                        <div class="col-sm-10">
```

```
                                        <p>{{ $produk->harga_beli }}</p>
                                    </div>
                                </div>
                                <div class="form-group">
                                    <div class="col-sm-offset-2 col-sm-10">
                                        <a href="{{ route('produk.index') }}" class="btn
btn-warning">Data Produk</a>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
@endsection
```

Selanjutnya lakukan uji coba dengan mengklik tombol detail yang ada pada salah satu data di tabel produk maka kita akan diarahkan ke halaman detail produk yang baru saja dibuat seperti pada gambar dibawah:



MENGHAPUS DATA

Pada bagian menampilkan data produk kita telah menyiapkan form untuk melakukan proses penghapusan data di bagian tag td:

```
<td>
    <form method="post" action="{{ route('produk.destroy',$p->id) }}">
        {{ csrf_field() }}
        <input type="hidden" name="_method" value="DELETE">
        <button class="btn btn-danger" type="submit">Hapus</button>
    </form>
</td>
```

Form tersebut akan mengirimkan data ke route produk.destroy dengan parameter $id dari data produk. Karena ini merupakan penghapusan data dan kita menggunakan route resource makan kita perlu mendifinsikan _method yang akan kita gunakan yang kita simpan dalam tag input type hidden, selanjutnya bukalah controller produkController dan modifikasi fungsi destroy() menjadi seperti berikut:

```
public function destroy($id)
{
    produk::where('id',$id)->delete();          return
redirect()->route('produk.index');    }
```

Kemudian lakukan uji coba dengan memilih menu hapus pada salah satu data produk yang ada pada tabel produk.

Tugas

- Buatlah CRUD sederhana untuk projek anda

MODUL IX

VALIDASI FORM PADA LARAVEL

(Pertemuan 14)

Tujuan :

1. Mahasiswa dapat memahami validasi form dalam framework

DASAR TEORI

Laravel menyediakan beberapa pendekatan berbeda untuk memvalidasi data masuk ke dalam controller. Secara default, kelas kontroler dasar Laravel menggunakan sifat ValidatesRequests yang menyediakan metode yang nyaman untuk memvalidasi permintaan HTTP yang masuk dengan berbagai aturan validasi yang kuat.

VALIDASI FORM SEDERHANA

Untuk memulai contoh pembuatan validasi pertama kita memerlukan sebuah form, yang pada contoh ini kita akan menggunakan form CRUD untuk tabel produk yang baru kita buat pada modul sebelumnya. Langkah pertama bukalah file view **create.blade.php** yang ada pada folder **resources/views/produk/** dan tambahan script:

```
@if ($errors->has('name_field'))
    <span class="label label-danger">{{ $errors->first('name_field') }}</span> @endif
```

Pada setiap tag input yang ada pada file create.blade.php di folder **resources/views/produk/** script tersebut berfungsi untuk melakukan pengecekan apabila pada saat halaman dibuka apakah $error terisi atau tidak, dan jika terisi maka error akan ditampilkan melalui tag span. Adapun script create.blade.php pada folder **resources/views/produk/** akan menjadi seperti berikur:

```
@extends('layouts.app') @section('content')
<div class="container">
  <div class="row">
   <div class="col-md-12">
       <div class="panel panel-default">
              <div class="panel-head container-fluid" style="margin-top: 10px;">
                    <p>Tambah Data produk</p>
              </div>
```

```
                <div class="panel-body">
      <form class="form-horizontal" action="{{ route('produk.store')
}}" method="post">
                                {{ csrf_field() }}
<div class="form-group">
       <label class="control-label col-sm-2">Nama
Produk</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="nama">
                                            @if ($errors->has('nama'))
                                        <span class="label
        labeldanger">{{ $errors->first('nama') }}</span>
                                            @endif
                                    </div>
                                </div>
<div class="form-group">
       <label class="control-label col-sm-2">Kategori
Produk</label>
                                    <div class="col-sm-10">
                                        <select class="form-control"
name="kategori">
        <option value="">Pilih
Kateogri</option>
                                            @foreach ($kategori as $k)
         <option value="{{ $k->id
}}">{{ $k->nama }}</option>
                                            @endforeach
                                        </select>
                                            @if ($errors->has('kategori'))
                                            <span class="label
        labeldanger">{{ $errors->first('kategori') }}</span>
                                            @endif
```

```
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Qty
Awal</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="qty">
 @if ($errors->has('qty'))                                                    <span
          class="label    labeldanger">{{ $errors->first('qty') }}</span>
                                                    @endif
                                    </div>
                                </div>

                                <div class="form-group">
        <label class="control-label col-sm-2">Harga
Jual</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="jual">
                                                    @if ($errors->has('jual'))
                                                <span class="label
          labeldanger">{{ $errors->first('jual') }}</span>
                                                    @endif
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Harga
Beli</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="beli">
```

```
                                                    @if ($errors->has('beli'))
                                                <span class="label
        labeldanger">{{ $errors->first('beli') }}</span>
                                                    @endif
                                        </div>
                                    </div>
                                    <div class="form-group">
                                        <div class="col-sm-offset-2 col-sm-10">
                                            <button type="submit" class="btn
btnprimary">Simpan</button>
                                        </div>
                                    </div>
        </form>
                </div>
        </div>
    </div>
   </div>
</div>
@endsection
```

Kemudian pada controller produkController.php yang ada pada folder **app\Http\Controller** di dalam fungsi store modifikasi scriptnya menjadi seperti berikut:

```
   public function store(Request $request)
   {
      $aturan = [
         'nama' => 'required',
         'kategori' => 'required',
         'qty' => 'required|numeric',
         'beli' => 'required|numeric',
         'jual' => 'required|numeric',
      ];
      $pesan = [
        'required' => 'Data ini Wajib Diisi !',
```

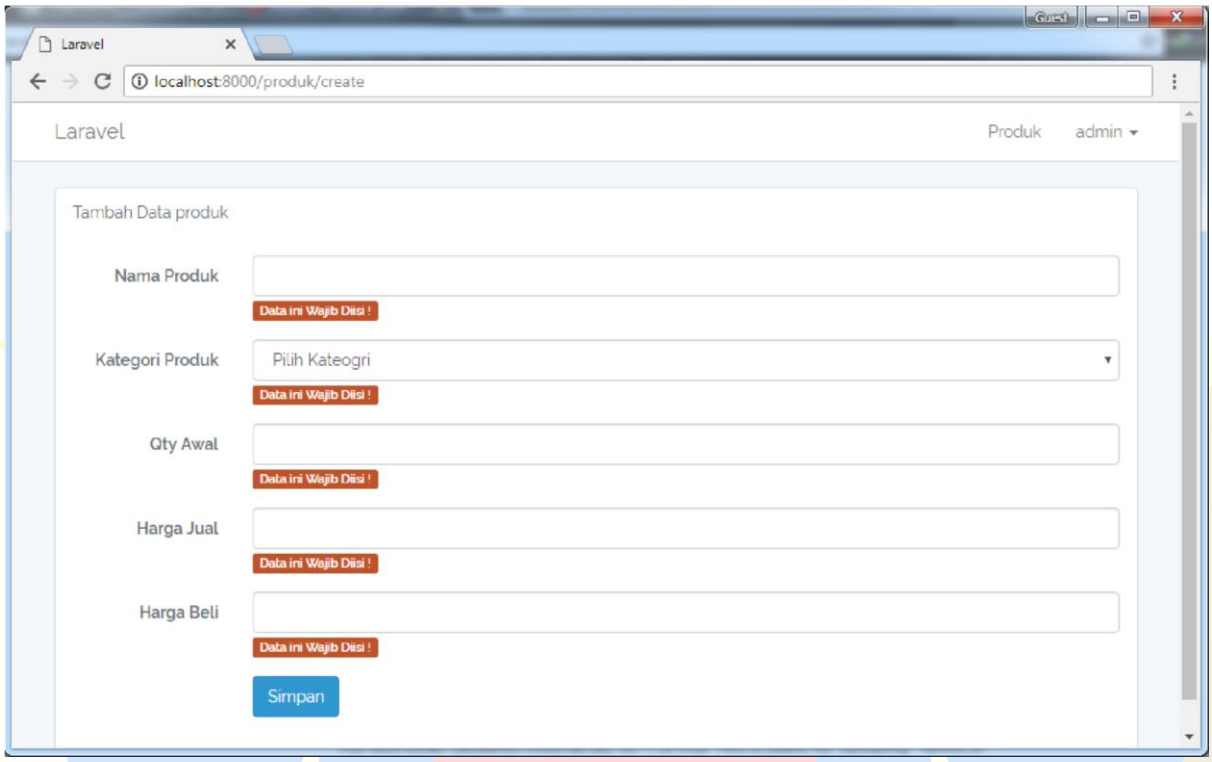
```
    'numeric' => 'Mohon isi dengan angka'
];
$this->validate($request,$aturan,$pesan);
produk::create([
    'nama' => $request->nama,
    'id_kategori' => $request->kategori,
    'qty' => $request->qty,
    'harga_beli' => $request->beli,
    'harga_jual' => $request->jual,
]);
return redirect()->route('produk.index');     }
```

Adapun penjelasan dari modifikasi script tersebut adalah pertama kita mendefinisikan variabel aturan yang akan digunakan untuk memvalidasi fom tambah produk. Variabel tersebut bertipe array assosiatif yang dimana index dari array tersebut merupakan name dari input type yang ada pada form dan nilai dari masing-masing index merupakan aturan-aturan yang berlaku untuk masing-masing input type pada form. Terdapat banyak aturan yang dapat digunakan untuk memvalidasi form yang disediakan oleh Laravel yang dapat dilihat pada situs resmi Laravel https://laravel.com/docs/5.4/validation .

Kemudian kita juga mendifinisikan variabel pesan yang bertipe array assosiatif yang akan menyimpan data pesan jika salah satu aturan tidak terpenuhi. Format dari variabel pesan ini adalah index dari array akan menjadi nama aturan yang digunakan dan nilai dari array tersebut berisi dengan pesan yang akan muncul jika kondisi aturan tidak terpenuhi. Pada bagian akhir kita memanggil fungsi validate() dengan urutan parameter

validate(data yang ingin di validasi, aturan untuk data,pesan yang akan dimunculkan untuk setiap aturan);

Sekarang lakukan uji coba pada form tambah data produk. Dengan mengosongkan semua field atau salah satu field dan klik tombol simpan, maka halaman akan dibawa kembali ke halaman tambah data produk dengan pesan error seperti berikut:



MENGATUR NILAI LAMA PADA VALIDASI FORM LARAVEL

Jika diperhatikan pada validasi form yang dibuat sebelumnya jika salah satu nilai form tidak memenhi aturan validasi maka kita akan dikembalikan ke halaman tambah data produk atau ke halaman awal kita melakukan tambah data. Namun jika salah satu nilai form memenuhi aturan nilai tersebut akan hilang pada form. Hal ini dapat kita cegah dengan menambahkan script berikut pada masing-masing tag input:

```
{{ old('name_field') }}
```

Script tersebut menyimpan nilai lama dari masing-masing input type yang memenuhi aturan. Pada form tambah data produk modifikasi kembali dengan menambahkan value pada setiap input type sehingga menjadi seperti berikut:

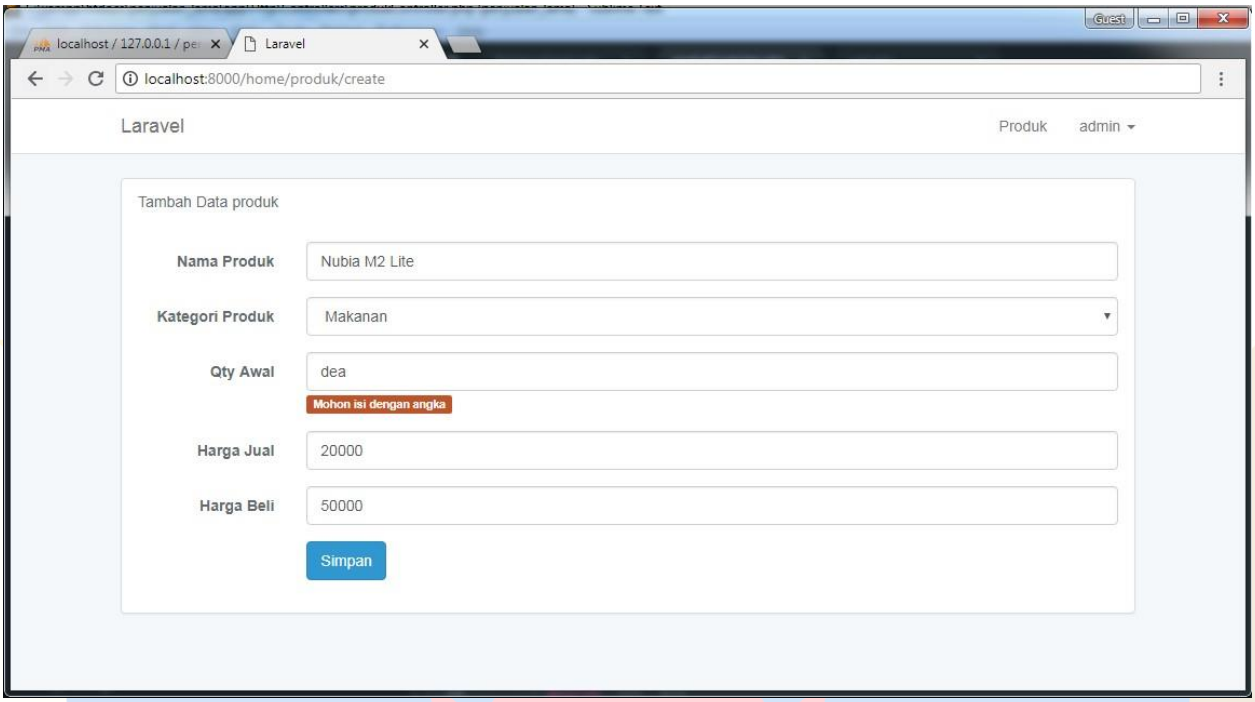
```
@extends('layouts.app') @section('content')
<div class="container">
   <div class="row">
    <div class="col-md-12">
          <div class="panel panel-default">
                <div class="panel-head container-fluid" style="margin-top: 10px;">
                      <p>Tambah Data produk</p>
                </div>
```

```
                <div class="panel-body">
      <form class="form-horizontal" action="{{ route('produk.store')
}}" method="post">
                                {{ csrf_field() }}
                                <div class="form-group">
        <label class="control-label col-sm-2">Nama
Produk</label>
                                        <div class="col-sm-10">
                                                <input          type="text"          class="form-control"
name="nama" value="{{ old('nama') }}">
                                                        @if ($errors->has('nama'))
                                                           <span          class="label
          labeldanger">{{ $errors->first('nama') }}</span>
                                                        @endif
                                        </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Kategori
Produk</label>
                                        <div class="col-sm-10">
                                                <select          class="form-control"
name="kategori">
          <option value="">Pilih
Kateogri</option>
                                                        @foreach ($kategori as $k)
           <option value="{{ $k->id }}"
@if (old('kategori')==$k->id) selected @endif>{{ $k->nama }}</option>
                                                        @endforeach
                                                </select>
                                                        @if          ($errors->has('kategori'))
                                                           <span          class="label
          labeldanger">{{ $errors->first('kategori') }}</span>
                                                        @endif
```

```
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Qty
Awal</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="qty" value="{{ old('qty') }}">
                                            @if ($errors->has('qty'))
                                        <span class="label
        labeldanger">{{ $errors->first('qty') }}</span>
                                            @endif
                                    </div>
                                </div>

                                <div class="form-group">
        <label class="control-label col-sm-2">Harga
Jual</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="jual" value="{{ old('jual') }}">
                                            @if ($errors->has('jual'))
                                        <span class="label
        labeldanger">{{ $errors->first('jual') }}</span>
                                            @endif
                                    </div>
                                </div>
                                <div class="form-group">
        <label class="control-label col-sm-2">Harga
Beli</label>
                                    <div class="col-sm-10">
                                        <input type="text" class="form-control"
name="beli" value="{{ old('beli') }}">
```

```
                                                @if ($errors->has('beli'))
                                                <span class="label
        labeldanger">{{ $errors->first('beli') }}</span>
                                                @endif
                                        </div>
                                    </div>
                                    <div class="form-group">
                                        <div class="col-sm-offset-2 col-sm-10">
                                            <button type="submit" class="btn
btnprimary">Simpan</button>
                                        </div>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>
                </div> </div>
@endsection
```

Kemudian lakukan uji coba dengan memberikan nilai salah pada salah satu input field dan klik tombol simpan, maka pesan error akan dimunculkan dengan nilai lama dari form masih terdapat pada setiap field input seperti pada contoh:



Tugas

- Lakukan validasi form pada projek anda

MODUL X

MENAMPILKAN DATA DALAM GRAFIK

(Pertemuan 15)

Tujuan :

1. Mahasiswa mampu memahami penggunaan grafik dalam framework

DASAR TEORI

Grafik adalah penyajian data yang terdapat dalam table yang ditampilkan ke dalam bentuk gambar. Selain itu grafik juga dapat diartikan sebagai suatu kombinasi data-data baik berupa angka, huruf, simbol, gambar, lambang, perkataan, lukisan, yang disajikan dalam sebuah media dengan tujuan memberikan gambaran tentang suatu data dari penyaji materi kepada para penerima materi dalam proses menyampaikan informasi.

Untuk membuat data grafik dari laravel kita akan menggunakan data dari tabel produk yang telah kita buat pada latihan-latihan sebelumnya. Adapun grafik yang akan kita buat adalah grafik yang menampilkan jumlah produk yang ada per kategori produk.

Adapun dependency yang akan kita gunakan adalah dependency dari situs **http://www.chartjs.org/.** Pada situs http://www.chartjs.org/ terdapat banyak sekali pilihan bentuk bentuk grafik, namun pada materi kali ini kita akan mencoba membuat grafik batang.

KEGIATAN PRAKTIKUM 1 MEMBUAT GRAFIK BATANG

Langkah pertama yang harus dilakukan saat membuat sebuah grafik adalah menyiapkan data dari grafik yang akan dibuat. pada contoh kali ini kita akan menampilkan grafik pada halaman dashboard setelah user/pengguna login. Langkah pertama yang harus dilakukan adalah bukalah file **HomeController.php** yang berada pada folder **app\Http\Controller** dan modifikasi fungsi index() yang ada di dalam controller tersebut menjadi seperti dibawah:

```
public function index()
{
    $kateogri = DB::table('kategoris')->get();
    $data = [];
    $label = [];

    foreach ($kateogri as $i => $v) {
        $value[$i] = DB::table('produks')->where('id_kategori',$v->id)->count();
```

```
        $label[$i] = $v->nama;
    }
    return view('home')
        ->with('value',json_encode($value))
        ->with('label',json_encode($label));     }
```

pada fungsi index() diatas pertama-tama kita mengambil data semua kategori yang tersimpan lalu kita melakukan perulangan untuk menghitung jumlah dari produk berdasarkan kategorinya yang kita simpan ke dalam variabel array bernama value, sedangkan untuk label dari grafiknya nanti kita menampungnya dalam variabel array bernama label. Kedua variabel tersebut kita lempar bersama ke dalam view home.blade.php yang ada pada folder **resources/views/** dengan bentuk json_encode. Json ecode sendiri merupakan fungsi php yang membuat bentuk array menjadi string dengan bentuk alaminya [value1,value,value3].

Kemudian pada file home.blade.php yang berada pada folder **resources/views/** bukalah dan tambahkan script berikut sebelum tag @endsection:

```
<script type="text/javascript" src="http://www.chartjs.org/dist/2.7.2/Chart.bundle.js"></script>
<script type="text/javascript" src="http://www.chartjs.org/samples/latest/utils.js"></script>
```

Skript tersebut merupakan script dari situs http://www.chartjs.org/ yang akan membantu kita membuat grafik nantinya. Kemudian modifikasi koding html yang pada file **home.blade.php** menjadi seperti dibawah:

```
@extends('layouts.app') @section('content')
<div class="container">
   <div class="row">
      <div class="col-md-8 col-md-offset-2">
         <div class="panel panel-default">
            <div class="panel-heading">Dashboard</div>
            <div class="panel-body">
               @if (session('status'))
                  <div class="alert alert-success">
                     {{ session('status') }}
                  </div>
```

```
                @endif
                You are logged in!
            </div>
            <div id="container" style="width: 100%;">
                <canvas id="canvas"></canvas>
            </div>
        </div>
    </div>
  </div>
</div>
@endsection
```

Pada script diatas kita menambahkan satu div dengan id bernilai canvas. Pada div ini nanti kita akan menaruh grafik jumlah barang per produk. Langkah selanjutnya adalah menambahkan javascript untuk mengambil kelas dari script yang telah kita include kan dari situs http://www.chartjs.org/ tadi dengan kode seperti berikut:

```
<script type="text/javascript"> var
color = Chart.helpers.color; var
barChartData = {     labels: {!! $label
!!},    datasets: [{
      label: 'Produk Per Kategori',
      backgroundColor:                color(window.chartColors.red).alpha(0.5).rgbString(),
borderColor: window.chartColors.red,       borderWidth: 1,       data: {!! $value !!},
  }],
};
 window.onload = function() {
  var ctx = document.getElementById('canvas').getContext('2d');       window.myBar = new
Chart(ctx, {       type: 'bar',       data: barChartData,       options: {
        responsive: true,          legend: {
          position: 'top',
        },          title: {              display: true,
          text: 'Grafik Data Produk'
        }
```

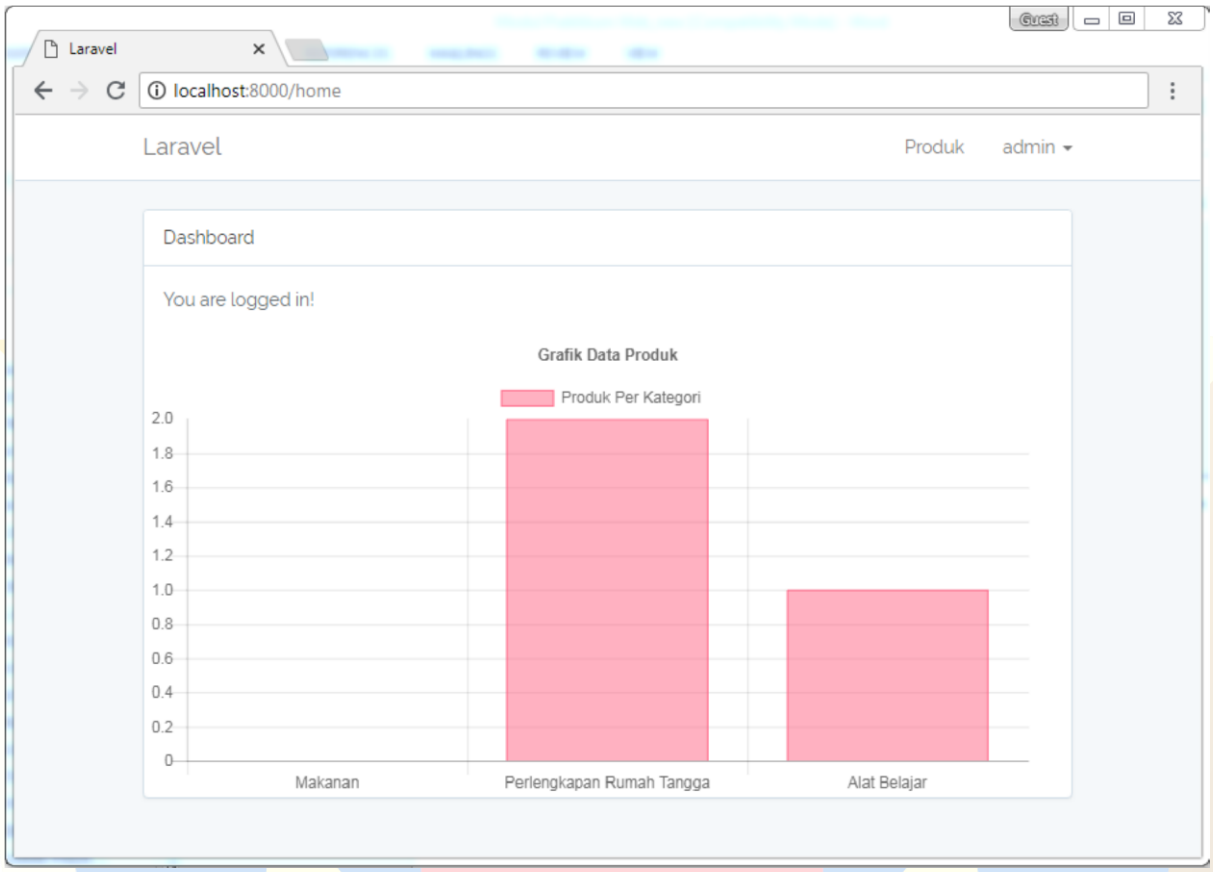
```
        }
    });
};
</script>
```

Jika dilihat secara keseluruhan bentuk kode yang ada di dalam file home.blade.php ini adalah seperti berikut:

```
@extends('layouts.app')
@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-8 col-md-offset-2">
            <div class="panel panel-default">
                <div class="panel-heading">Dashboard</div>
                <div class="panel-body">
                    @if (session('status'))
                        <div class="alert alert-success">
                            {{ session('status') }}
                        </div>
                    @endif

                    You are logged in!
                </div>
                <div id="container" style="width: 100%;">
                    <canvas id="canvas"></canvas>
                </div>
            </div>
        </div>
    </div>
</div>
<script           type="text/javascript"
src="http://www.chartjs.org/dist/2.7.2/Chart.bundle.js"></script>
```

```
<script type="text/javascript"
src="http://www.chartjs.org/samples/latest/utils.js"></script>
<script type="text/javascript"> var
color = Chart.helpers.color; var
barChartData = { labels: {!! $label
!!}, datasets: [{
        label: 'Produk Per Kategori',
        backgroundColor: color(window.chartColors.red).alpha(0.5).rgbString(),
borderColor: window.chartColors.red, borderWidth: 1, data: {!! $value !!},
}],
};
window.onload = function() {
    var ctx = document.getElementById('canvas').getContext('2d'); window.myBar = new
Chart(ctx, { type: 'bar', data: barChartData, options: {
            responsive: true, legend: {
                position: 'top',
            }, title: { display: true,
                text: 'Grafik Data Produk'
            }
        }
    });
};
</script>
@endsection
```

Lalu jika kita membuka halaman dashboard setelah login maka tampilannya akan nampak seperti pada gambar dibawah:



Tugas

- Gunakan grafik untuk menampilkan data pada projek anda