

Muhammad Azamuddin  
Hafid Mukhlasin

5.7

# Laravel

The PHP Framework For Web Artisans

---

# Lisensi & Informasi Versi

---

## Lisensi

Ebook ini hanya boleh digunakan oleh pemilik email yang tertera di header buku ini. Penggunaan buku oleh selain pemilik email tersebut merupakan tindakan yang tidak diperbolehkan.

Siapapun (termasuk pemilik buku) tidak memiliki hak untuk menyalin dan atau menyebarkan buku ini tanpa seizin penulis.

## Versi ebook

versi	tanggal	author	keterangan
1.0.0	10 Oktober 2018	Muhammad Azamuddin	Rilis pertama
1.0.1	15 Oktober 2018	Muhammad Azamuddin	Fix typo di konten dan daftar isi
1.0.2	08 November 2018	Muhammad Azamuddin	Daftar isi baru + bookmark
1.0.3	13 November 2018	Muhammad Azamuddin	GET method lebih aman dibanding POST method, seharusnya adalah POST method yang lebih aman dibanding GET method. Pada bab Route.

# Persembahan

---

Bismillahirrahmanirrahim. Ucapan tanpa batas untuk Yang Maha Kuasa, Allah SWT atas setiap nafasku dan keberkahanNya. Shalawat serta salam bagi junjunganku, Nabi Muhammad SAW atas teladannya.

Terima kasih kepada Bapak dan Ibu saya, Yusuf dan Khadliroh atas cinta dan doa tulus yang tak pernah putus.

Dalam luasnya ruang dan besarnya waktu, adalah kebahagiaanku untuk mengarungi planet dan zaman dengan Istriku tercinta, Fadhilah Rimandani. Dan juga mentariku kecil yang selalu menemani Ayah dan menjadi penyejuk hati, Hana humaira.

Kepada adik-adikku, Nadia dan Johan, setiap langkahku juga tak lepas dari kenangan di saat kecil bersama kalian. Kita telah tumbuh bersama dan selalu ada rasa rindu kepada masa-masa itu.

Kemudian buku ini juga saya persembahkan kepada orang-orang atas inspirasi yang luar biasa, Sonny Lazuardi, Jihad Dzikri Waspada, Irfan Maulana, Peter J Kambey, Anne Regina, Bos naufal, Deroh, Luphy dan seluruh teman-teman komunitas programming lainnya yang tidak dapat saya sebutkan satu persatu.

Bung Danar terima kasih atas diskusi-diskusi yang sering kita lakukan sehingga saya mau tidak mau terus belajar.

Secangkir kopi panas kedai Reman yang selalu menemani hari-hari saya, luar biasa!

Dan juga tentu saja buku ini saya persembahkan tidak lain tidak bukan untuk Anda para pembaca, karena saya menulis untuk Anda dan berharap dapat membawa banyak manfaat untuk para pembaca dalam meningkatkan skill web programming. Terima kasih dan semoga senantiasa kesuksesan bersama kita semua, di dunia dan akhirat. Aaamiin

# Kata Pengantar

---

Puji syukur kehadirat Allah SWT atas limpahan rahmat dan karunianya sehingga Buku Panduan Laravel ini telah dapat diselesaikan. Buku panduan ini merupakan edisi pertama, sebagai pedoman bagi siapa saja yang ingin belajar membuat aplikasi berbasis web dengan framework Laravel.

Terima kasih disampaikan kepada mas Hafid Mukhasin selaku rekan penulis, guru dan juga sekaligus mentor saya selama ini, sebuah kehormatan mendapatkan kesempatan untuk berkolaborasi. Terima kasih juga kepada segenap kolega saya yang atas kontribusi dalam penyempurnaan buku ini.

Menjadi seorang web developer tentunya memerlukan banyak ilmu yang harus dikuasai. Dan untuk menggapainya tidaklah mudah, akan tetapi justru hal itu yang akan menempa seseorang untuk menjadi developer yang handal.

Meskipun tidak mudah tetapi bukan berarti kita harus menyerah. Salah satu usaha yang bisa dilakukan adalah dengan berusaha meningkatkan skill untuk menguasai framework-framework bagus baik dari segi kualitas maupun komunitas. Tentu ada banyak framework yang bisa kita pilih, dengan berbagai pertimbangan pada kesempatan yang awal ini, kami memutuskan untuk menulis panduan tentang framework laravel ke dalam sebuah buku dengan tujuan untuk memberikan referensi yang lengkap disertai studi kasus agar pembaca dapat mengaplikasikan ilmu yang didapatk setelah membaca buku ini.

Kami menyadari masih terdapat kekurangan dalam buku ini untuk itu kritik dan saran terhadap penyempurnaan buku ini sangat diharapkan. Semoga buku ini dapat memberi manfaat bagi para pembaca budiman.

Jakarta, 3 Oktober 2018

Muhammad Azamuddin

# Daftar Isi

---

<b>Lisensi &amp; Informasi Versi .....</b>	2
Lisensi .....	2
Versi ebook .....	2
<b>Persembahan .....</b>	3
<b>Kata Pengantar .....</b>	4
<b>Mengenal Laravel .....</b>	13
Sekilas Laravel .....	13
Sejarah .....	13
Fitur Unggulan .....	14
Mengapa Laravel .....	17
Open Source .....	17
Ekosistem Bagus .....	17
Mature .....	17
Kenyamanan dan Kemudahan .....	18
Secure .....	18
Modern .....	18
Kesimpulan .....	18
<b>Instalasi &amp; Konfigurasi .....</b>	20
Persiapan Lingkungan Kerja .....	20
Persyaratan Sistem .....	20
Untuk Pengguna Windows 7 atau 8 (Menggunakan XAMPP) .....	21
Install cmder Full (cmder + git) .....	21
Install XAMPP .....	21
Setup System Variable .....	21
Install Composer .....	22
Memastikan Sistem telah Memenuhi Persyaratan .....	22
Untuk Pengguna Linux & MacOS (atau Windows 10 Professional) .....	23
Instalasi Git .....	24
Instalasi Git di Linux .....	24
Instalasi Git di Debian-based Linux .....	24
Instalasi Git di Red Hat-based Linux .....	24
Instalasi Git di Mac .....	25
Instalasi di Windows .....	25
Instalasi Docker .....	25
Install Docker Compose .....	26
Menyiapkan Lingkungan Kerja Laradock .....	27
Membuat Project Laravel Baru .....	28
Masuk ke Workspace .....	28
Buat Project Menggunakan Installer Laravel .....	29
Buat Project dengan Composer create-project .....	29
Konfigurasi Awal .....	30
Folder Public .....	30
File Konfigurasi .....	30
Hak Akses Folder .....	30
Application Key .....	30
Konfigurasi Tambahan .....	30
Konfigurasi Web Server .....	31
Pretty URLs .....	31
Apache (XAMPP) .....	31
Nginx (Laradock) .....	31

Menghilangkan public di URL .....	31
Apache (XAMPP) .....	31
Nginx (Laradock) .....	32
Variabel Lingkungan .....	33
Hello World .....	34
Kesimpulan .....	34
<b>Arsitektur Laravel .....</b>	<b>36</b>
Konsep MVC .....	36
MVC di Laravel .....	36
Visualisasi MVC di Laravel .....	37
Pengenalan Routing .....	38
Di mana kita tuliskan definisi routing? .....	40
Pengenalan Model .....	40
Membuat Model .....	40
Pengenalan Controller .....	41
Membuat controller .....	41
Controller action .....	42
Pengenalan View .....	44
Kesimpulan .....	44
<b>Route &amp; Controller .....</b>	<b>45</b>
Route .....	45
Tipe-tipe route .....	45
Web route .....	45
Api route .....	45
Console route .....	45
Channel route .....	46
Mendefinisikan route .....	47
Route Parameter .....	47
Optional Route Parameter .....	49
Route Berdasarkan Jenis HTTP Method .....	49
GET Method .....	49
POST Method .....	50
PUT Method .....	50
DELETE Method .....	50
Mengizinkan lebih dari satu HTTP method .....	51
Named Route .....	51
Route Group .....	52
Route View .....	53
Controller .....	53
Anatomi controller .....	53
Membuat controller .....	54
Controller Resource .....	55
Action resource .....	55
index .....	55
create .....	55
store .....	55
show .....	56
edit .....	56
update .....	56
destroy .....	56
Membuat Controller Resource .....	56
Membuat Route Resource .....	58
Membaca Input & Memberikan Response .....	59
Input & Query String .....	59

Membaca Input & Query String .....	60
Membaca semua input & query string .....	61
Mengecualikan input & query string .....	62
Memberikan Response .....	62
Response dasar .....	62
Response redirect ke halaman lain .....	62
Response redirect ke website lain .....	63
Response view .....	63
Kesimpulan.....	63
<b>Database .....</b>	<b>64</b>
Intro .....	64
Konfigurasi Koneksi database .....	64
Metode dalam Bekerja dengan database .....	66
Code First .....	66
Database First .....	66
Migration .....	66
Membuat File Migration .....	66
Mengeksekusi Migration .....	68
Migration Rollback .....	68
Schema Builder .....	68
Membuat Tabel .....	68
Memilih Tabel .....	69
Mengecek apakah column / field sudah ada .....	69
Memilih koneksi .....	70
Mengubah settingan tabel .....	70
Mengubah nama column .....	70
Menghapus column .....	70
Menghapus tabel .....	71
Operasi untuk Column .....	71
Column Modifiers .....	73
Mengubah Column Attribute .....	74
Seeding .....	75
Membuat seeder .....	75
Menjalankan Seeder .....	76
Menjalankan Multiple Seeder Class .....	77
Latihan Praktik .....	78
A. Membuat Tabel dan Strukturnya dengan Migration .....	78
Kesimpulan .....	81
<b>Model &amp; Eloquent .....</b>	<b>82</b>
Model .....	82
Konvensi Model .....	82
Model Attribute / properti .....	83
Mengganti Tabel pada Model .....	83
Menggunakan Koneksi Selain Default pada Model .....	83
Mengubah Primary Key Model .....	84
Mass-assignment .....	84
Mengizinkan Operasi Mass-assignment pada Properti Model .....	85
Memproteksi Properti Model dari Operasi Mass-assignment .....	86
Mencatat Kapan Data Model Dibuat dan Diupdate .....	86
Eloquent .....	87
Query Record .....	88
Menampilkan seluruh record .....	88
Mencari record berdasarkan primaryKey .....	88
Mendapatkan record pertama saja .....	88

findOrFail()	88
Aggregates	89
count()	89
max()	89
min()	89
sum()	89
avg()	89
Insert Record	89
save()	90
create()	90
Update Record	91
save()	91
update()	91
Delete Record	91
Menghapus data model	92
Menghapus satu atau lebih data model berdasarkan primaryKey sekaligus	92
Soft Deletes	92
trashed()	93
restore()	94
withTrashed()	94
onlyTrashed()	94
forceDelete()	95
Data Pagination	95
simplePaginate()	95
paginate()	95
Perbedaan simplePaginate() dan paginate()	96
Menampilkan link pagination di view	97
Latihan Praktik	97
Kesimpulan	101
<b>View</b>	102
Intro	102
Menampilkan view dari controller	102
Memberikan data ke view	102
Menampilkan data	103
Menampilkan unescaped data	104
Komentar	104
Control Structure	105
@if	105
@unless	105
@switch	105
Mengecek apakah sebuah variabel tersedia	106
Mengecek apakah data kosong	106
Mengecek apakah pengguna sudah login	106
Mengecek apakah pengguna belum login	107
Menampilkan Kumpulan Data	107
@foreach	107
@forelse	107
@for	108
@while	108
Nested Loop	108
Loop Variable	108
@php	109
Blade Layout, Section & Component	109
Layout & Section	109

@yield .....	110
@section .....	110
@parent .....	110
@extends .....	110
Component .....	111
Latihan praktik .....	112
Kesimpulan .....	119
<b>Relationship .....</b>	<b>120</b>
Intro .....	120
Pengenalan relationship .....	120
One to one .....	120
One to many .....	122
Many to many .....	124
Apa yang kita pelajari? .....	126
Relationship di Laravel .....	126
Overview .....	126
Struktur tabel dan model .....	128
One to one relationship .....	128
Menyiapkan tabel dengan migration .....	128
Mendefinisikan di Model .....	129
Query one to one relationship .....	130
Query one-to-one relationship .....	130
One to many relationship .....	131
Menyiapkan tabel dengan migration .....	131
Mendefinisikan di Model .....	132
Many to many relationship .....	132
Menyiapkan tabel dengan migration .....	133
Mendefinisikan di Model .....	134
Menghubungkan relationship .....	135
save() .....	135
create() .....	135
associate() .....	136
createMany() .....	136
attach() .....	137
Menghapus relationship .....	138
dissociate() .....	138
detach() .....	138
Sinkronisasi relationship .....	138
Querying relationship tingkat lanjut .....	138
Mendapatkan data model hanya yang memiliki relation tertentu .....	139
Mendapatkan data berdasarkan query terhadap relationship model .....	139
Mendapatkan hanya data model yang tidak memiliki relation tertentu .....	139
Menghitung jumlah relationship .....	139
Foreign Key Constraint .....	140
Tipe-tipe constraint .....	140
SET NULL .....	140
RESTRICT .....	141
CASCADE .....	141
NO ACTION .....	141
Menerapkannya dalam migration .....	141
Kustomisasi definisi relationship .....	142
Mengubah foreign key di relationship .....	142
Mengubah pivot tabel pada many-to-many relationship .....	143
Mengubah foreign key pada many-to-many relationship .....	143

Kesimpulan .....	143
<b>Studi Kasus Manajemen Toko .....</b>	<b>145</b>
Gambaran Studi Kasus .....	145
Desain database .....	146
Membuat Project Laravel Baru .....	146
Pengguna XAMPP .....	146
Pengguna Laradock .....	147
laravel new project .....	147
Konfigurasi database .....	149
Ubah .env .....	149
User Authentication .....	150
Mengenal Authentication .....	150
User Authentication di Laravel .....	150
Scaffolding .....	150
Menyesuaikan struktur table users .....	154
Seeding user administrator .....	157
Hashing .....	158
Layout, Halaman dan Styling .....	159
Membuat layout aplikasi .....	159
Link Logout .....	162
Menggunakan template bootstrap .....	162
Menyesuaikan halaman login .....	165
Menghapus fitur registrasi .....	168
Menjadikan halaman awal sebagai login page .....	169
Manage Users .....	169
CRUD User .....	169
Membuat user resource .....	169
Fitur create user .....	170
Buat form untuk create users .....	170
Menangkap request dan menyimpan ke database .....	175
Menghandle file upload .....	176
Menyimpan user ke database .....	178
Membaca session / flash message .....	179
Fitur list user .....	182
Mendapatkan users dari database .....	182
Membuat view untuk list users .....	182
Fitur edit user .....	185
Mengambil data user yang akan diedit lalu lempar ke view .....	186
Membuat form edit .....	187
Menangkap request edit dan mengupdate ke database .....	194
Fitur delete user .....	197
Tambahkan link delete di list user .....	197
Menangkap request delete dan menghapus user di database .....	197
Menampilkan detail user .....	198
Mencari user dengan id tertentu .....	199
Membuat view untuk detail user .....	199
Filter User berdasarkan Email .....	201
Fitur filter by status user .....	206
Menampilkan pagination dan nomor urut .....	212
Manage Category .....	213
Membuat migration table categories .....	213
Membuat model Category .....	214
CRUD Category .....	215
Membuat resource category .....	215

Fitur create category .....	215
Membuat view untuk create category .....	215
Menangkap request create categories .....	218
Fitur category list .....	220
Filter kategori berdasarkan nama .....	222
Menangkap request filter by name .....	223
Fitur edit category .....	224
Menangkap request untuk update .....	227
Fitur show detail category .....	231
Fitur delete category .....	232
Menyesuaikan model Category .....	232
Delete .....	233
Menangkap request delete .....	234
Show soft deleted category .....	235
Restore .....	239
Delete permanent .....	241
Manage Book .....	245
Membuat migration .....	246
Membuat migration untuk tabel books .....	246
Membuat migration untuk relationship ke tabel categories .....	247
Membuat model Book .....	249
Mendefinisikan relationship .....	249
Mendefinisikan relationship di model Book .....	249
Mendefinisikan relationship di model Category .....	250
CRUD Book .....	250
Fitur create book .....	251
Menangkap request create user di controller action .....	253
Fitur pilih kategori buku .....	256
Fitur list book .....	258
Fitur edit book .....	260
Fitur soft delete book .....	266
Aktifkan soft delete pada model Book .....	266
Trash .....	266
Show soft deleted book / show trash .....	268
Show published or draft .....	270
Menampilkan navigasi all, publish, draft dan trash .....	271
Restore .....	272
Delete permanent .....	273
Filter book by title .....	275
Manage Order .....	282
Membuat migration .....	282
Membuat migration untuk tabel orders .....	282
Membuat migration untuk relationship ke tabel books .....	284
Membuat model Order .....	286
Mendefinisikan relationship .....	286
Relationship di model Order .....	286
CRUD Order .....	289
Persiapan database .....	289
Fitur list orders .....	290
Fitur edit status order .....	293
Fitur pencarian order .....	295
Validasi Form .....	299
Validasi form create user .....	300
Menampilkan pesan error di form .....	302

Validasi form edit user .....	306
Validasi form create category .....	309
Validasi form edit category .....	311
Validasi form create book .....	315
Validasi form edit book .....	318
Daftar rule validasi keseluruhan .....	320
<b>Kustomisasi Error Page .....</b>	<b>325</b>
Membuat halaman 401 / Unauthorized .....	326
Membuat halaman 403 / Forbidden .....	327
Membuat halaman 404 / Notfound .....	327
<b>Gate Authorization .....</b>	<b>327</b>
Mendefinisikan Gate di AuthServiceProvider .....	329
Otorisasi UserController .....	332
Otorisasi Category Controller .....	333
Otorisasi BookController .....	334
Otorisasi OrderController .....	334
Source Code .....	335
<b>Deployment .....</b>	<b>336</b>
Deployment ke shared hosting .....	336
Persiapan .....	336
Export database (.sql) dari development .....	336
Mengubah konfigurasi di .env .....	341
Upload project zip ke hosting .....	342
Menjalankan perintah artisan storage:link di server hosting .....	348
Menghapus &#34;public&#34; dari URL .....	351
Deployment ke VPS .....	352
Tambahkan remote di repository laradock lokal .....	353
Tambahkan remote di repository larashop lokal .....	355
Persiapkan VPS di Digital Ocean .....	357
Siapkan VPS agar bisa diakses via SSH .....	360
Mengarahkan custom domain ke VPS .....	362
Siapkan laradock dan larashop di VPS .....	365
Menjalankan webserver nginx .....	366
Clone source code larashop .....	367
Tips saat create droplet tidak bisa access console .....	369
Tips jika gagal menjalankan &#34;docker-compose up nginx&#34; .....	370
Kesimpulan .....	370

# Mengenal Laravel

## Sekilas Laravel

Laravel merupakan salah satu framework PHP yang sangat populer saat ini. Menurut beberapa survei, salah satunya di Coderseye, Laravel merupakan framework PHP modern terpopuler di tahun 2018 diikuti oleh Phalcon, CodeIgniter dan Symfony. Hal ini terbukti apabila kita melakukan pengecekan di Google Trends, seperti terlihat di gambar berikut ini, meskipun urutan kedua dan seterusnya berbeda dengan hasil survei Coderseye, tetapi peringkat pertama sama-sama dipegang oleh Laravel.



Sangat mencengangkan bukan? Bagaimana popularitas Laravel melesat tinggi meninggalkan framework lainnya, bahkan yang di peringkat kedua. Perhatikan garis berwarna biru tersebut. Laravel sebetulnya dibangun di atas framework Symfony, akan tetapi popularitas Symfony sendiri kalah jauh dari Laravel. Karena Laravel telah banyak membuat fitur-fitur yang memudahkan developer daripada langsung menggunakan Symfony. Lalu seperti apa sih sejarah Laravel ini? Kita akan membahasnya setelah ini.

## Sejarah

Laravel merupakan proyek *open source* yang dirintis oleh Taylor Otwell bertujuan untuk mengembangkan aplikasi berbasis web dengan arsitektur MVC (Model-View-Controller). Beberapa fitur laravel antara lain desain yang modular, beberapa cara untuk mengakses database yang memudahkan developer dalam pengembangan maupun *maintenance*. Semua itu mengarah kepada sintaks yang pendek dan mudah dipahami (*syntactic sugar*). Fitur-fitur tersebut akan kita bahas lebih detil di bahasan Fitur.

Laravel awalnya dibuat oleh Taylor Otwell untuk menyediakan framework alternatif yang lebih canggih dibandingkan CodeIgniter, saat itu CodeIgniter tidak memiliki beberapa fitur penting, seperti otentifikasi dan otorisasi bawaan. Rilis pertama Laravel dilakukan pada 9 Juni 2011 dengan versi beta, dan diikuti dengan rilis Laravel 1 pada bulan yang sama.

Laravel 1 memiliki fitur bawaan seperti otentikasi, localisation, models, views, sessions, routing dan mekanisme lainnya. Akan tetapi belum benar-benar memiliki controllers sehingga belum bisa disebut MVC. Sebetulnya bisa saja disebut MVC tetapi bukan merupakan MVC yang ideal. Laravel 2 dirilis pada September 2011, dengan membawa beberapa perbaikan dari pembuatnya maupun dari komunitas Laravel. Fitur baru yang tersedia di versi ini antara lain, *controllers*, yang membuat Laravel kini menjadi framework MVC yang ideal, dukungan terhadap prinsip *inversion of control* (IoC), dan templating system yaitu Blade. Sebagai akibatnya, dukungan terhadap *library* pihak ketiga dihapuskan di Laravel 2

Laravel 3 dirilis pada bulan Februari 2012 dengan fitur-fitur baru seperti *command-line-interface* (CLI) bernama Artisan, dukungan bawaan untuk beberapa manajemen *database*, *database migration* sebagai pengelola versi untuk skema *database*, dukungan untuk menghandle *events*, dan sistem pengelola paket (*packaging system*) bernama Bundles. Peningkatan *userbase* Laravel dan peningkatan popularitas terjadi dengan rilisnya Laravel 3.

Laravel 4, codename Illuminate --bukan Illuminati ya--, dirilis pada bulan Mei 2013. Versi ini merupakan penulisan ulang dari *codebase* framework Laravel, memindahkan beberapa sistem *dependency* ke *package* terpisah melalui Composer yang menjadi pengelola package di tingkat aplikasi. Fitur lainnya adalah database seeding untuk mengisi data awal, biasanya berguna saat fase pengembangan aplikasi. Selain itu juga mendukung berbagai cara untuk mengirim email, mendukung penundaan penghapusan pada *database* atau yang lebih dikenal dengan istilah "soft delete".

## Fitur Unggulan

- Sintaks yang Ringkas dan Elegan

Laravel memiliki sintaks yang ringkas dan elegan, tidak percaya? Mari kita lihat bersama, misalnya kita ingin melakukan penyimpanan data User ke database. Kita cukup melakukannya seperti ini:

```
$id = 1;

// cari user berdasarkan primaryKey
$user = User::find($id);

// ubah field dengan nilai baru
$user->name = "Muhammad Azamuddin";

// simpan ke database
$user->save();
```

Kode diatas sangat ringkas dan mudah dipahami. Kita mencari user dengan ID 1 lalu kemudian mengubah nilai *property* *name* menjadi "Muhammad Azamuddin". Dan kita menyimpannya dengan satu baris kode yaitu *\$user->save()*.

- Eloquent ORM

Apa yang baru saja kita lihat di poin sebelumnya merupakan operasi untuk mendapatkan dan mengolah data dari *database*. Apa kamu menyadari bahwa kode tersebut berbeda sekali dari PHP *native* yang mungkin sebelumnya sering kamu pakai.

Misal

```
$id = 1;

$command = "UPDATE user SET user.name = 'Muhammad Azamuddin' where
user.id = $id";

// $conn asumsi variabel koneksi ke database
if($conn->query($command)){
    // berhasil menyimpan
} else {
    // gagal menyimpan
}
```

Bandingkan lagi dengan kode berikut, jauh lebih singkat mana?

```
$id = 1;

$user = User::find($id);
$user->name = "Muhammad Azamuddin";

$user->save();
```

Meskipun kita tetap bisa menggunakan *SQL Command*, tetapi kita bisa memanfaatkan Eloquent ORM untuk operasi *database* agar lebih singkat dan cepat dan juga meminimalisasi kesalahan penulisan *command*.

ORM memiliki kepanjangan *Object Relational Mapping*, dengan ORM kita bisa melakukan operasi CRUD ke *database* tanpa harus menulis *SQL Command*. ORM inilah yang bertugas untuk menulis *SQL Command* untuk kita. Laravel menggunakan ORM yang bernama Eloquent.

Keunggulan menggunakan Eloquent ORM selain sintaks yang lebih ringkas antara lain

## 1. Mudah ganti *Database*

Dengan Eloquent ORM apabila kita ingin mengganti *database* dari MySQL ke MSSQL atau Oracle atau yang lain kita tidak perlu khawatir perbedaan sintaks yang bisa membuat error. Karena kita tidak perlu mengubah kode yang kita tulis dengan Eloquent ORM. Eloquent ORM tersebut yang akan menggenerate *SQL Command* sesuai dengan *database* baru yang kita pakai

## 2. Mudah mengelola *relationship* antar tabel

Dengan Eloquent ORM membuat, mendapatkan dan memanipulasi *relationship* antar tabel menjadi lebih menyenangkan. Jika sebelumnya kita memerlukan *SQL Command* yang cukup

Licensed to Adhitya Venancius - adhityavenancius@yahoo.com - 087882568270 at 20/11/2018 17:12:45  
panjang dan rawan kesalahan. Dengan Eloquent kita tidak perlu khawatir lagi. Lebih detail akan di bahas di Bab Model & Eloquent.

### 3. Memudahkan pemula dalam *SQL Command*

Jika Kamu tergolong pemula dalam *SQL Command*, kamu akan sangat terbantu dengan adanya Eloquent ORM. Karena kamu tidak perlu menghafal *SQL Command* yang rumit untuk bisa membuat fitur canggih.

- Mau Pake *SQL Command*? Bisa!

Misalnya, kamu lebih suka langsung menggunakan *SQL Command*, tidak perlu khawatir. Kamu juga bisa melakukannya di Laravel dengan fitur *facade DB*.

- Templating Engine

Laravel memanfaatkan *templating engine* yaitu Blade. Dengan Blade kamu bisa melakukan banyak hal yang jika hanya menggunakan HTML tidak bisa.

Menghindari penulisan kode HTML berulangkali untuk tampilan yang sama dengan *Layout* dan *Component*. Mengontrol tampilan dimunculkan atau disembunyikan. Dan juga melakukan *looping* dalam tampilan. Semua itu akan kita pelajari pada Bab View.

- *Migration*

*Migration* memudahkan kita mengelola struktur *database* dan tabel aplikasi kita. Jika kita terbiasa membuat tabel atau field menggunakan GUI semisal PHPMyAdmin atau SQL Management Studio, maka dengan *Migration* kita bisa langsung melakukannya dari teks editor tanpa berpindah aplikasi.

Keunggulan lainnya adalah *versioning*. Dengan *migration* kita bisa menyimpan riwayat perubahan struktur *database* kita dari waktu ke waktu sesuai dengan file *migration* yang kita buat. Tentu hal ini juga berarti kita bisa melakukan *rollback* ke poin-poin tertentu.

Selain itu kita juga bisa melakukan sharing struktur *database* ke rekan kerja kita. Sehingga struktur yang ada akan tersinkronisasi dengan baik hanya dengan menjalankan *migration* terhadap file-file *migration* yang kita buat.

- *Policy & Gate*

Dalam aplikasi Laravel kita bisa lebih mudah untuk menetapkan hak akses ke aplikasi atau ke fitur-fitur tertentu dengan *Policy & Gate*. Umumnya jika hak akses yang akan kita terapkan tergolong sederhana, maka menggunakan *Gate* cukup. Namun bila hak akses aplikasi kita tergolong rumit kita bisa menggunakan fitur *Policy*. Kenapa kita menggunakan *Policy*? Agar pengelolaan hak akses tersebut lebih rapi dan tertata meskipun *rulenya kompleks*. Kita akan mempelajari materi ini pada Bab Studi Kasus. Jadi, persiapkan dirimu!

- Built-in JSON

Di era sekarang ini, eranya *web service*. Kamu bisa mengembangkan aplikasi dengan view Blade ataupun menggunakan frontend javascript seperti VueJS, ReactJS atau yang lain. Dan kabar gembiranya, dengan Laravel, akan sangat mudah untuk menghasilkan *output* berupa JSON yang bisa digunakan oleh frontend javascript kita.

## Mengapa Laravel

Dengan melihat fitur-fitur unggulan yang ada sebetulnya kita sudah bisa mendapatkan gambaran bagus mengapa kita memilih Laravel. Namun selain hal-hal teknis tersebut, ada beberapa alasan lagi yang sulit kita pungkiri, mengapa kita sebaiknya memilih Laravel jika kita ingin menggunakan bahasa PHP. Apa saja alasannya? Berikut beberapa alasannya.

### Open Source

Laravel dikembangkan sebagai sebuah proyek *open source*. Hal ini merupakan hal yang positif. Itu artinya Laravel bisa kita modifikasi sesuai kebutuhan kita, bisa kita distribusikan, atau bisa kita gunakan untuk membuat proyek baik untuk personal atau *commercial*. Selain itu meskipun *open source* bukan berarti selalu gratis, tetapi Laravel termasuk ke dalam proyek *open source* yang selain **BEBAS** tapi juga **GRATIS**.

Perlu diluruskan apabila ada yang berpandangan bahwa *open source* berarti gratisan, dan gratisan berarti jelek. Ini merupakan pandangan kuno dan keliru, jangan diteruskan, karena tadi, *open source* bukan selalu bermakna gratis. Dan juga perlu dipahami bahwa gratis != gratisan (gratis tidak sama dengan gratisan).

Tidak perlu diragukan lagi kualitas Laravel. Selain *open source* Laravel merupakan framework berkualitas tinggi.

### Ekosistem Bagus

Dalam sebuah perbincangan di sebuah perusahaan, seorang pegawai yang tidak pernah menggunakan *open source* berceloteh bahwa kita harus memilih tools & framework yang dibuat oleh perusahaan besar. Karena akan membutuhkan *support* nya ketika terjadi masalah.

Menurut hemat saya, pegawai tersebut mungkin belum menjiwai atau menyelami era development saat ini. Kita software developer bukan user, kalo kita user, ya kita mungkin perlu *support* langsung dari perusahaan IT yang mengembangkan software.

Apa iya setiap kali kita menemukan kesulitan atau bug dalam aplikasi yang kita kembangkan akan menelpon *support*? Coba bayangkan berapa lama yang akan kita habiskan untuk menunggu jawaban dari mereka? selain jumlah *support* di perusahaan itu pasti terbatas, belum lagi kalau ternyata skill programming mereka kurang bagus, programming itu luas sekali. Itupun kalo kita di perusahaan yang mampu membayar untuk *support* semacam itu.

Jawaban yang kita butuhkan sebetulnya bukanlah *support* dari perusahaan dibalik teknologi yang kita pakai. Tapi justru dari komunitas yang besar yang menggunakan teknologi tersebut.

Laravel jelas memiliki *userbase* yang besar, dari trends saja sudah terlihat Laravel no 1. Framework paling populer dan ini artinya jika kita mengalami kendala kita bisa bertanya ke komunitas yang besar itu.

Selain itu juga *package-package* dukungan untuk Laravel juga banyak. Kita bisa memilih sesuai kebutuhan dari aplikasi kita. Dan inilah inti dari Ekosistem yang Bagus yang dimiliki oleh Laravel.

Dengan menggunakan Laravel, kamu berada di komunitas terbesar developer PHP.

### Mature

Saat ini Laravel sudah sampai pada versi 5, telah banyak perkembangan dan perbaikan framework ini sejak versi pertama diluncurkan beberapa tahun silam. Laravel bukanlah framework baru yang belum jelas masa depannya. Laravel telah didukung oleh *author* berkompeten yaitu Taylor Otwell dan juga oleh kontributor yang solid.

Fitur dan juga API dari Laravel selalu diperbarui dan diupdate mengikuti perkembangan di dunia web development. Bahkan kini Laravel memiliki sistem versioning yang memungkinkan penggunanya untuk memprediksi kapan Laravel versi baru akan muncul karena jadwal yang jelas.

Laravel juga memiliki versi LTS atau *Long Term Support* yaitu versi Laravel yang akan didukung untuk *bug fixes* selama 2 tahun dan terkait keamanan untuk jangka waktu kurang lebih 3 tahun sejak dirilis.

## Kenyamanan dan Kemudahan

Laravel merupakan framework yang sangat memerhatikan faktor kenyamanan dalam pengembangan aplikasi (*developer experience*) sesuai dengan *tagline* mereka yaitu "*The PHP Framework for Web Artisans*"

Kenyamanan dan kemudahan ini tercermin dari dokumentasi yang bagus, sintaks yang ringkas dan elegan, penataan file yang rapi serta struktur aplikasi yang modular.

Selain itu juga di Laravel kamu bisa memilih untuk menggunakan ORM atau *SQL command facade* DB, memilih menggunakan view atau menggunakan frontend javascript. Kita juga diberikan opsi Policy atau Gate dalam mengelola hak akses. Selain itu juga ada middleware yang bisa kita gunakan.

Di Laravel 5.7, kita juga bisa membuat fitur otentikasi pengguna secara bawaan dengan sangat mudah.

Selain itu Laravel memiliki dukungan *Command-Line Interface* yang sangat membantu dalam pengembangan seperti untuk menggenerate *Controller*, *Model*, *Migration*, dll.

## Secure

Tidak dipungkiri faktor security merupakan faktor yang sangat penting. Terutama apabila kita berada di lingkungan perusahaan yang mengutamakan kemanan dari pengguna yang akan memakai aplikasi kita.

Sebagai contoh, saya pribadi mengembangkan sebuah aplikasi untuk perusahaan besar di mana ada Tim khusus untuk menguji kelayakan dan keamanan dari aplikasi sebelum masuk ke tahap *live / production*. Aplikasi tersebut dilakukan pengujian oleh tim tersebut, meskipun awalnya tidak lolos karena kelalaian saya mengubah konfigurasi dan validasi (bukan karena Laravel). Akhirnya, aplikasi tersebut lolos uji kelayakan dan keamanan.

## Modern

Tidak diragukan lagi bahwa Laravel merupakan framework yang modern. Laravel senantiasa menyesuaikan fitur-fitur mereka dengan perkembangan saat ini, ketika framework lain masih belum kepikiran untuk menerapkannya.

Selain itu Laravel versi 5.7 juga sudah menggunakan PHP versi 7 yang berarti banyak fitur-fitur terbaru dari PHP yang bisa digunakan di Laravel.

## Kesimpulan

Laravel merupakan framework yang mengutamakan *developer experience*. Hal ini sangat cocok untuk Kamu yang menginginkan keindahan kode, kecepatan dan keluwesan dalam pembuatan aplikasi yang aman dan modern.

Kami berharap kamu sudah semakin paham keunggulan Laravel sehingga Kamu lebih bersemangat untuk menguasai framework ini. Sudah tidak sabar untuk segera belajar? Mari kita lanjutkan ke Bab berikutnya.

# Instalasi & Konfigurasi

---

## Persiapan Lingkungan Kerja

Sebelum menggunakan Laravel kita harus menyiapkan terlebih dahulu lingkungan development kita. Setelah itu kita akan belajar membuat project laravel baru dan melakukan konfigurasi awal terhadapnya.

Langkah pertama yang kita lakukan adalah menyiapkan lingkungan development kita. Hal ini terkait dengan persyaratan minimal sistem yang dibutuhkan oleh Laravel dan instalasi tool-tool pendukung sesuai dengan sistem operasi kita. Sudah siap? mari kita mulai.

### Persyaratan Sistem

Laravel memiliki beberapa persyaratan sistem yang harus dipenuhi agar bisa berjalan. Apa saja persyaratannya? Persyaratan yang harus dipenuhi antara lain:

- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension.

Selain persyaratan di atas, untuk memudahkan kita dalam pengembangan aplikasi berbasis web, kita sebaiknya menginstall tool-tool berikut ini:

- Cmder (khusus pengguna windows);
- Git;
- Composer.

Kemudian kita juga akan membutuhkan hal-hal berikut ini:

- Nginx sebagai web server;
- MySQL sebagai database engine;
- PhpMyadmin untuk mengelola database menggunakan Web GUI;
- Redis untuk mengelola cache.

Selanjutnya kita akan melakukan setup lingkungan development kita supaya persyaratan-persyaratan di atas terpenuhi. Selain itu, kita juga menginginkan proses development yang nyaman. Ada 2 alternatif yang kami berikan panduannya yaitu:

### 1. Menggunakan XAMPP

Bagi pengguna Windows. Khusus pengguna Windows 7, 8 atau di versi sebelumnya hanya bisa memilih opsi ini.

Sebetulnya bagi pengguna non-Windows juga bisa memilih cara ini. Namun, di buku ini kami hanya memberikan contoh instalasi XAMPP di Windows. Meskipun demikian, bagi pengguna non-Windows yang ingin XAMPP bisa mengikuti langkah-langkahnya karena secara garis besar panduannya sama. Cukup pilih installer sesuai dengan OS masing-masing ketika mendownload XAMPP nya ya.

## 2. Menggunakan Laradock (Docker environment)

Bagi pengguna Mac, Linux dan Windows 10 Profesional. Kita akan menggunakan Laradock yang membutuhkan instalasi Docker.

Jadi setelah ini silahkan kamu baca panduan sesuai dengan selera mu ya. Tidak perlu mengikuti keduanya.

## Untuk Pengguna Windows 7 atau 8 (Menggunakan XAMPP)

### Install cmder Full (cmder + git)

Seperti dijelaskan di atas, untuk pengguna Windows disarankan menginstall cmder. Buka <https://cmder.net> lalu download cmder versi full (sudah termasuk Git) bukan yang mini.

### Install XAMPP

Untuk pengguna Windows 7 atau 8, kita akan menggunakan XAMPP untuk local development, kita terlebih dahulu menginstall XAMPP. Perlu diperhatikan persyaratan sistemnya ya, pilih XAMPP dengan versi PHP 7.1.3 ke atas. XAMPP bisa didownload di <https://www.apachefriends.org/index.html>

Download versi XAMPP 7.2.6 atau versi di atasnya, lalu instal dengan mengikuti langkah-langkah di wizard. Setelah berhasil instal, kita akan membuat project-project kita nantinya di folder **htdocs** yang secara default berada di **C:\xampp\htdocs**.

### Setup System Variable

Saat development nanti kita akan memerlukan eksekusi perintah **php** melalui cmd / cmder. Agar perintah **php** tersebut bisa diakses melalui cmd / cmder kita harus mensetting system variable terlebih dahulu.

Sebelum itu, coba kamu buka cmder, lalu ketik **php -v** dan tekan enter. Seharusnya kamu akan menjumpai error kira-kira seperti ini

```
`php` is not recognized as an internal or external command, operable program or batch file.
```

Untuk mengatasi error ini, kita set system variable terlebih dahulu. Caranya, pada **Computer** klik kanan pilih **Properties**. Setelah itu klik **Advanced system settings**.

Setelah muncul popup seperti di bawah ini

Klik **Environment variables**, Lalu di box yang bawah cari property **Path** atau **PATH**, double klik lalu pada nilainya kita tambahkan kode ini ;**C:\xampp\php**, ingat ditambahkan dibelakang bukan menghapus

nilai yang ada ya. Baris yang baru saja kita tambahkan adalah path ke PHP yang sudah kita install sebelumnya menggunakan XAMPP.

Untuk memastikan bahwa konfigurasi di atas berhasil, tutup cmder lalu buka kembali dan ketikan lagi `php -v` seharusnya sekarang kita mendapatkan informasi versi PHP yang ada di sistem kita. Jika demikian maka kita telah siap untuk step selanjutnya.

## Install Composer

Composer merupakan package manager untuk PHP. Kita akan sangat memerlukan Composer saat development dengan Laravel.

Buka <https://getcomposer.org/download> lalu kita bisa memilih instalasi menggunakan Windows Installer, download `Composer-Setup.exe` link langsung ada di <https://getcomposer.org/Composer-Setup.exe>.

Setelah berhasil, cek dengan membuka cmder baru, lalu ketika perintah `composer`, jika muncul daftar `command` yang tersedia, maka instalasi sudah berhasil. Good!

## Memastikan Sistem telah Memenuhi Persyaratan

Setelah kita menginstall XAMPP, maka kita harus memastikan bahwa sistem kita sudah memenuhi semua persyaratan sistem Laravel. Untuk melakukannya ikuti petunjuk berikut ini:

1. Download / copy konten dari file ini <https://raw.githubusercontent.com/GastonHeim/Laravel-Requirement-Checker/master/check.php>
2. Simpan sebagai file `check.php`, letakkan di folder `C:\xampp\htdocs`. Lalu buka browser Kamu dan buka url `http://localhost/check.php`, pilih versi Laravel yang akan kita gunakan yaitu 5.7, jika kamu melihat semua persyaratan memiliki centang, artinya semua persyaratan telah terpenuhi! Yay!



# Server Requirements.

PHP >= 7.1.3 ✓  
OpenSSL PHP Extension ✓  
PDO PHP Extension ✓  
Mbstring PHP Extension ✓  
Tokenizer PHP Extension ✓  
XML PHP Extension ✓  
CTYPE PHP Extension ✓  
JSON PHP Extension ✓

Seharusnya jika Kamu sudah instal XAMPP versi 7.2.6 maka semua persyaratan sudah terpenuhi.

Untuk Pengguna Linux & MacOS (atau Windows 10 Professional)

Jika kamu pengguna Linux / MacOS, kami sarankan menggunakan Laradock. Oia, jika kamu menggunakan Windows 10 Profesional, kamu boleh memilih menggunakan XAMPP atau menggunakan Laradock juga.

Kenapa Laradock? Karena Laradock telah menyiapkan lingkungan development PHP yang lengkap. Fitur-fitur Laradock antara lain:

- **Yang paling penting!** Workspace image yang sudah siap dengan tool-tool berikut ini: **PHP CLI - Composer - Git - Linuxbrew - Node - V8JS - Gulp - SQLite - xDebug - Envoy - Deployer - Vim - Yarn - SOAP - Drush**
- Mudah ganti versi PHP: 7.2, 7.1, 5.6 ...
- Bisa memilih *database engine*: MySQL, Postgres, MariaDB ...
- Jalankan kombinasi software: Memcached, HHVM, Beanstalkd ...
- Setiap software berjalan di container yang terpisah: PHP-FPM, NGINX, PHP-CLI ...
- Mudah untuk mengkostumisasi container dengan mengubah Dockerfile
- Semua image dibuat berdasarkan image official (*Trusted base Images*)
- Telah dikonfigurasi otomatis untuk development menggunakan NGINX, bisa digunakan untuk single project atau multiple project
- Mudah untuk menginstall atau mengunisntall software di *container* menggunakan *environment variable*

- Memiliki Dockerfile yang rapi
- Semuanya bisa diedit
- Image yang cepat dibuild.

Untuk menggunakan Laradock kita perlu terlebih dahulu menginstall Docker. Itulah sebabnya bagi pengguna Windows, hanya versi Windows 10 Profesional ke atas yang bisa menikmatinya. Karena Docker for Windows tidak menyediakan untuk versi Windows sebelumnya.

Jika sistem operasimu belum memiliki Git install Git terlebih dahulu.

## Instalasi Git

Jika sistem operasimu belum memiliki Git, maka kamu harus menginstalnya terlebih dahulu. Berikut panduan untuk menginstall Git di Linux, MacOS dan Windows

### Instalasi Git di Linux

Pertama kamu tentukan Linux yang kamu pakai itu base nya apa. Jika kamu menggunakan ubuntu atau linuxmint, itu artinya kamu menggunakan Debian-based.

#### Instalasi Git di Debian-based Linux

Langkah menginstal Git adalah dengan membuka terminal, lalu ketika kode di bawah ini. Satu per satu ya tiap baris.

```
// pertama update  
sudo apt-get update
```

```
// lalu jika perlu lakukan upgrade distro sekalian  
sudo apt-get upgrade
```

```
// setelah itu kamu install git begini  
sudo apt-get install git
```

Jika sudah, maka kamu sekarang bisa menggunakan Git.

#### Instalasi Git di Red Hat-based Linux

Buka terminal, lalu ketikan perintah ini

```
sudo yum upgrade
```

```
sudo yum install git
```

Sekarang kamu bisa menggunakan Git.

### Instalasi Git di Mac

Instalasi Git di Mac akan lebih baik jika menggunakan Homebrew, jika di Mac mu belum ada Homebrew, install terlebih dahulu dengan mengetikan kode berikut ini di terminal.

```
ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Jika sudah lalu ketik perintah ini untuk mengecek kondisi brew di sistem kita.

```
brew doctor
```

Selanjutnya kita install Git dengan mengetikan ke terminal perintah ini

```
brew install git
```

Jika proses selesai, kamu bisa mulai menggunakan Git. Bila perintah git belum tersedia di terminal, coba untuk menutup terminal lalu buka kembali.

### Instalasi di Windows

Cara termudah adalah dengan menginstall cmder full version yang sudah ada Git di dalamnya di <https://cmder.net>. Sekaligus kamu juga mendapatkan cmder (Terminal emulator untuk Windows).

### Instalasi Docker

Docker memiliki dua versi yaitu Community Edition (CE) dan Enterprise Edition (EE). Kita akan menggunakan versi Community Edition.

- Untuk Mac <https://store.docker.com/editions/community/docker-ce-desktop-mac>
- Untuk Ubuntu <https://store.docker.com/editions/community/docker-ce-server-ubuntu>

Petunjuk detail instalasi di ubuntu <https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-using-the-repository>

- Untuk OS lainnya silahkan lihat langsung di <https://www.docker.com/community-edition>.

Bagi pengguna Mac, setelah selesai proses download, silahkan install Docker sesuai dengan langkah-langkah yang ada pada wizard instalasi. Bagi pengguna Linux, ikuti petunjuk detail instalasi sesuai distro masing-masing. Jika proses instalasi sudah selesai, buka terminal, lalu ketik perintah `docker`. Jika sudah muncul daftar perintah yang bisa dijalankan oleh Docker, maka kita siap untuk ke langkah selanjutnya.

```
➜ docker <ruby-2.4.1>
Usage: docker COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                       "/Users/azamuddin/.docker")
  -D, --debug          Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
                        ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string   Trust certs signed only by this CA (default
                       "/Users/azamuddin/.docker/ca.pem")
  --tlscert string     Path to TLS certificate file (default
                       "/Users/azamuddin/.docker/cert.pem")
  --tlskey string       Path to TLS key file (default
                       "/Users/azamuddin/.docker/key.pem")
  --tlsverify          Use TLS and verify the remote
  -v, --version         Print version information and quit

Management Commands:
  checkpoint  Manage checkpoints
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network    Manage networks
  node        Manage Swarm nodes
```

## Install Docker Compose

Khusus untuk pengguna Linux, kita memerlukan instalasi docker-compose. Sementara untuk pengguna Mac tidak perlu melakukannya, karena docker-compose sudah included di installer Docker for Mac.

1. Buka terminal dan ketik

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.21.2/docker-
compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
```

Gunakan versi terbaru, contoh di atas menggunakan versi 1.21.2, kamu juga boleh mengecek versi terbaru yang tersedia di link ini <https://github.com/docker/compose/releases>

2. Setelah itu izinkan docker-composer tadi menjadi executable

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. Cek apakah instalasi berhasil dengan perintah berikut

```
docker-compose --version
```

Jika berhasil akan muncul output di terminal semacam ini,

```
docker-compose version 1.21.2, build 1719ceb
```

## Menyiapkan Lingkungan Kerja Laradock

Nah karena sekarang Docker dan Git sudah terinstal di sistem kita, selanjutnya kita siap menggunakan Laradock.

Silahkan buka terminal lalu buat sebuah folder dan beri nama **laravel-projects**, caranya dengan mengetikkan perintah

```
mkdir laravel-projects
```

Folder ini akan kita gunakan untuk menaruh project-project Laravel kita nantinya.

Lalu pindah ke direktori untuk meletakkan project Laravel kita dengan perintah

```
cd laravel-projects
```

Jika sudah, kita ketik perintah berikut:

1. Clone laradock

```
git clone https://github.com/Laradock/laradock.git
```

2. Setelah *cloning* berhasil, masuk ke folder laradock.

```
cd laradock
```

3. Ubah env-example menjadi .env

```
cp env-example .env
```

4. Nyalakan container laradock

```
docker-compose up -d nginx mysql phpmyadmin redis workspace
```

Minimal kita perlu menyalakan service nginx, mysql, phpmyadmin dan redis karena kita akan memerlukan ke empat service tersebut.

5. Buka file .env lalu tambahkan kode berikut

```
DB_HOST=mysql  
REDIS_HOST=redis  
QUEUE_HOST=beanstalkd
```

6. Buka <http://localhost>, laradockmu telah siap!

## Membuat Project Laravel Baru

Untuk membuat project baru pertama kita buka terminal / cmder. Mulai dari materi ini, kita akan menggunakan istilah terminal ya. Jadi kalo buka terminal bagi pengguna cmder silahkan buka cmder. Sepakat?

### Masuk ke Workspace

Buka terminal, lalu **masuk ke workspace** kita.

- Bagi pengguna XAMPP silahkan masuk folder **htdocs** yang defaultnya ada di **C:\xampp\htdocs** via terminal
- Bagi pengguna Laradock
  1. Masuk ke folder laradock yang sudah kita buat sebelumnya

```
cd laravel-projects/laradock
```

2. Lalu setelah itu jalankan perintah ini untuk masuk ke workspace

```
docker-compose exec --user=laradock workspace bash
```

Mulai sekarang setiap kami menginstruksikan untuk **masuk ke workspace**, lakukan langkah di atas yang sesuai dengan setup yang kalian gunakan, XAMPP atau Laradock.

## Buat Project Menggunakan Installer Laravel

Laravel memanfaatkan Composer untuk mengelola dependency. Dan di bahasan sebelumnya kita telah menginstall Composer dan tool lain yang kita butuhkan.

Download Laravel installer menggunakan composer, caranya ketik di terminal perintah berikut.

```
composer global require "laravel/installer"
```

Setelah berhasil menjankan perintah di atas, perintah laravel bisa kita gunakan. Untuk membuat aplikasi laravel baru dengan nama toko-online, jalankan perintah berikut di terminal.

```
laravel new toko-online
```

Setelah berhasil masuk ke direktori aplikasi yang baru saja kita install menggunakan terminal / cmder. Lalu ketik perintah berikut untuk menginstall dependency project kita:

```
composer install
```

## Buat Project dengan Composer **create-project**

Alternatif lain adalah dengan menggunakan perintah composer create-project di terminal, seperti berikut

```
composer create-project --prefer-dist laravel/laravel toko-online
```

Setelah kamu jalankan perintah untuk membuat project baru, Laravel akan menginstall dependency yang dibutuhkan. Tunggu sampai proses instalasi selesai. Jika sudah, silahkan coba dengan membuka <http://localhost/toko-online/public>

Seharusnya kamu akan melihat homepage seperti ini

# Laravel

DOCUMENTATION

LARACASTS

NEWS

FORGE

GITHUB

## Konfigurasi Awal

### Folder Public

Setelah menginstal Laravel, kita harus menjadikan public sebagai document / web root. File index.php di folder public menjadi controller utama yang meresponse setiap HTTP request.

### File Konfigurasi

Setiap konfigurasi framework laravel disimpan di folder config. Setiap settingan memiliki dokumentasi, jadi kita bisa melihat-lihat file tersebut untuk memahami settingan apa saja yang tersedia.

### Hak Akses Folder

Setelah menginstall Laravel, kita harus mengecek beberapa hak akses. Folder di dalam folder storage dan bootstrap/cache harus bisa ditulis (writable) oleh web server kita jika tidak, Laravel tidak bisa berjalan.

### Application Key

Setelah itu, yang harus kita lakukan setelah menginstall Laravel adalah memberikan nilai pada application key dengan string acak. Jika kita menginstall Laravel menggunakan Composer atau installer Laravel, key ini sudah diset untuk kita oleh perintah php artisan key:generate.

Umumnya, string untuk application key memiliki 32 karakter. Key ini juga bisa ditaruh di file .env. Jika kamu belum melakukan rename pada file .env.example menjadi .env, lakukan sekarang. Jika key tidak diset, session dan semua data yang dienkripsi di aplikasi kita tidak akan aman.

### Konfigurasi Tambahan

Laravel hampir tidak memerlukan konfigurasi tambahan. Kita bisa mulai mengembangkan aplikasi kita, akan tetapi, mungkin kita ingin melihat kembali file config/app.php dan dokumentasinya. File tersebut memiliki beberapa settingan seperti timezone dan locale yang mungkin ingin kita ubah.

## Konfigurasi Web Server

### Pretty URLs

#### Apache (XAMPP)

Laravel menyertakan file public/.htaccess yang digunakan untuk membuat URL tanpa index.php. Sebelum menjalankan Laravel dengan Apache, pastikan untuk mengaktifkan modul mod\_rewrite sehingga file .htaccess dibaca oleh server.

Jika file .htaccess yang disertakan laravel tidak berfungsi di instalasi Apache kita, coba alternatif ini:

```
Options +FollowSymLinks  
RewriteEngine On  
  
RewriteCond %{REQUEST_FILENAME} !-d  
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteRule ^ index.php [L]
```

#### Nginx (Laradock)

Jika kita menggunakan Nginx standalone, directive berikut ini akan mengarahkan semua request ke index.php.

Bagi yang menggunakan Laradock, bisa skip step ini karena sudah otomatis terkonfigurasi.

```
location / {  
    try_files $uri $uri/ /index.php?$query_string;  
}
```

### Menghilangkan **public** di URL

Jika kamu perhatikan saat mengakses aplikasi yang baru saja kita buat, kita menggunakan URL <http://localhost/nama-project/public>, bagaimana jika kita ingin mengakses sebagai <http://localhost/nama-project>? Kita bisa melakukan hal itu.

#### Apache (XAMPP)

Untuk pengguna Apache langkahnya seperti ini

1. Ubah nama server.php di project laravel kita menjadi index.php
2. Copy file .htaccess dari folder public di project laravel kita ke root project laravel kita

## Nginx (Laradock)

1. Assign hostname di /etc/hosts Edit file di `/etc/hosts/`, caranya ketik di terminal

```
sudo nano /etc/hosts
```

Lalu tambahkan virtual domain yang akan kita pakai, misalnya untuk project `toko-online` yang baru kita buat kita akan assign domain `toko-online.test` maka tambahkan kode ini

```
127.0.0.1 toko-online.test
```

2. Buat file konfigurasi nginx baru

1. Masuk folder `laradock/nginx/sites`
2. copy `laravel.conf.example` menjadi `toko-online.conf`

```
cp laravel.conf.example toko-online.conf
```

3. Edit file `toko-online.conf` tersebut

Ubah kode ini

```
root /var/www/laravel/public;
```

menjadi

```
root /var/www/toko-online/public;
```

4. Restart laradock container kita

Masuk ke folder `laradock` lalu jalankan perintah ini

```
docker-compose restart nginx
```

Dengan begitu setelah ini kamu cukup mengakses di browser dengan alamat <http://toko-online.test> tidak perlu lagi menggunakan <http://localhost/toko-online/public>.

Ketika membuat virtual domain, sebaiknya gunakan akhiran `.test`. Hindari penggunaan `.dev` karena sudah tidak didukung.

## Variabel Lingkungan

Umumnya kita memerlukan konfigurasi berbeda pada tahap pengembangan aplikasi dengan pada tahap produksi. Misalnya kita menggunakan driver cache yang berbeda antara server lokal dengan server produksi.

Untuk memudahkan pengelolaan konfigurasi lingkungan, Laravel memanfaatkan DotEnv, sebuah pustaka PHP karya Vance Lucas. Pada saat pertama kali kita membuat proyek Laravel baru, di direktori root aplikasi kita terdapat file .env.example. Jika kita menginstall menggunakan composer, file ini otomatis diubah nama menjadi .env. Jika belum, kita harus mengubah namanya secara manual.

Contoh isi dari file .env

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:vrRH1xjAwY8uJ2Mctkx68gfeMTnFA4tYH1wrbX160zA=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=toko-online
DB_USERNAME=root
DB_PASSWORD=root

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

File tersebut berisi konfigurasi aplikasi Laravel yang bisa kita ganti sesuai kebutuhan. Misalnya pada file di atas terdapat konfigurasi database mysql seperti host, port, nama database, username dan password.

Untuk mengubah kita akan menggunakan koneksi apa dan ke host mana cukup lakukan perubahan di file ini. Kita akan belajar melakukan konfigurasi database pada bab database.

## Hello World

Setelah kita berhasil membuat project baru, silahkan buka kembali project kita di browser. Lalu coba buka <http://localhost/toko-online/helo> atau <http://toko-online.test/helo>, seharusnya Anda akan menjumpai pesan error seperti ini “Sorry, the page you are looking for could not be found.”

Itu artinya URL yang kita akses tidak ditemukan. Nah kita akan menjadikan jika URL tadi diakses, maka akan muncul tulisan “Hello World”. Caranya adalah dengan mengedit file “routes/web.php”, lalu tambahkan di bagian akhir kode dengan kode berikut ini:

```
Route::get('/helo', function(){
    return "Hello World";
});
```

Sekarang coba buka kembali URL <http://toko-online.test/helo> atau <http://localhost/toko-online/helo>, kini muncul tulisan “Hello World” seperti yang kita definisikan di file `routes/web.php`.

## Kesimpulan

Untuk memulai development project Laravel kita harus melakukan setup lingkungan development terlebih dahulu. Ada 2 alternatif lingkungan development yang bisa kita pakai yaitu:

1. Menggunakan XAMPP
2. Menggunakan Laradock

Setiap opsi ada kekurangan dan kelebihannya, silahkan pilih yang sesuai dengan kenyamananmu. Buku ini memberikan panduan untuk setup kedua lingkungan di atas.

Adapun secara umum yang harus dipastikan adalah hal-hal berikut ini:

1. Penuhi persyaratan sistem Laravel;
2. Install web server baik Nginx atau Apache;
3. Install database engine seperti MySQL;
4. Install phpmyadmin bila perlu untuk web GUI database mysql kita;
5. Install cmder khusus pengguna Windows;
6. Install git;
7. Install composer;
8. Konfigurasi project Laravel seperti menghilangkan `index.php` dan `public` di URL.

Demikianlah sekelumit panduan instalasi dan konfigurasi Laravel. Kita cukup melakukannya instalasi sekali saja. Jika ingin membuat project baru kita tidak perlu melakukan setup lingkungan development dari awal. Cukup jalankan perintah `composer create-project` atau `laravel new`. Lalu lakukan konfigurasi awal bila perlu.

Setelah ini kita akan membahas tentang Arsitektur Laravel. Seperti apa sih arsitektur Laravel yang membuatnya populer dan menjadi framework bagus? Hal itu akan kita bahas di Bab berikutnya.

# Arsitektur Laravel

---

## Konsep MVC

MVC merupakan kependekan dari *Model View Controller* dan merupakan sebuah pola yang sudah teruji dalam pengembangan aplikasi. Awalnya, MVC digunakan untuk pengembangan GUI desktop, tapi kini telah banyak diadopsi oleh framework-framework aplikasi berbasis web. Jika kita mengembangkan aplikasi tanpa pola MVC, kita berkecenderungan untuk mencampur adukkan kode logika kita dengan kode tampilan serta kode untuk mengambil data ke database. Benar? Nah dengan MVC tidak begitu, karena dengan pola MVC, kita membagi komponen aplikasi kita menjadi 3 bagian yang terpisah namun saling berkaitan, yaitu *Model*, *View* dan *Controller*. Apa guna masing-masing komponen tersebut?

### 1. Model

Merupakan komponen dalam aplikasi kita yang bertanggungjawab mengelola akses langsung dengan sumber data dan logika pengelolaan data tersebut

### 2. View

Merupakan komponen dalam aplikasi kita yang bertanggungjawab untuk membuat tampilan / interface untuk pengguna. Sumber data didapat dari model yang didapatkan melalui controller. Tidak berinteraksi langsung dengan database. View juga menangkap interaksi dari pengguna yang akan diteruskan ke aplikasi.

### 3. Controller

Merupakan komponen dalam aplikasi kita yang bertanggungjawab untuk menerima input dan memberikan output, atau dalam dunia web kita lebih mengenal dengan istilah request dan response. Controller bertugas untuk menerima request, kemudian memprosesnya dengan memberikan response baik berupa data atau view berisi data dari model.

## MVC di Laravel

MVC di laravel secara konsep tidak jauh berbeda dengan konsep MVC secara umum. Setiap komponen di MVC telah disediakan folder khusus di aplikasi laravel kecuali model.

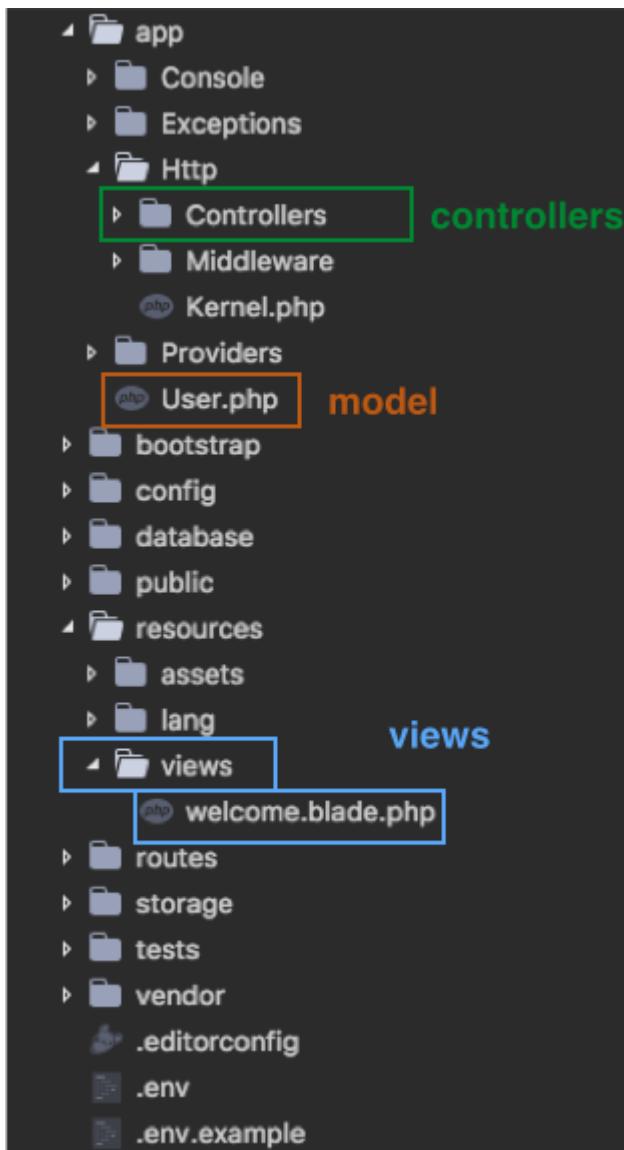
### 1. Controller berada di folder `app/Http/Controllers`

### 2. View berada di folder `resources/views`

View berfungsi untuk menaruh kode tampilan ke pengguna aplikasi. Di file view ini lah kita letakkan kode html, css dan javascript bukan di controller, route atau model. File view bisa mengakses variabel yang dilempar dari controller action seperti pada bahasan Controller.

Pada aplikasi Laravel baru, tersedia satu file view yaitu `welcome.blade.php`, silahkan dibuka pada `resources/views/welcome.blade.php`. File tersebut berisi kode html, css, javascript dan beberapa sintaks blade.

3. Khusus model tidak memiliki folder khusus, tetapi kita bisa meletakannya di folder app, atau di folder lainnya sesuai kebutuhan.

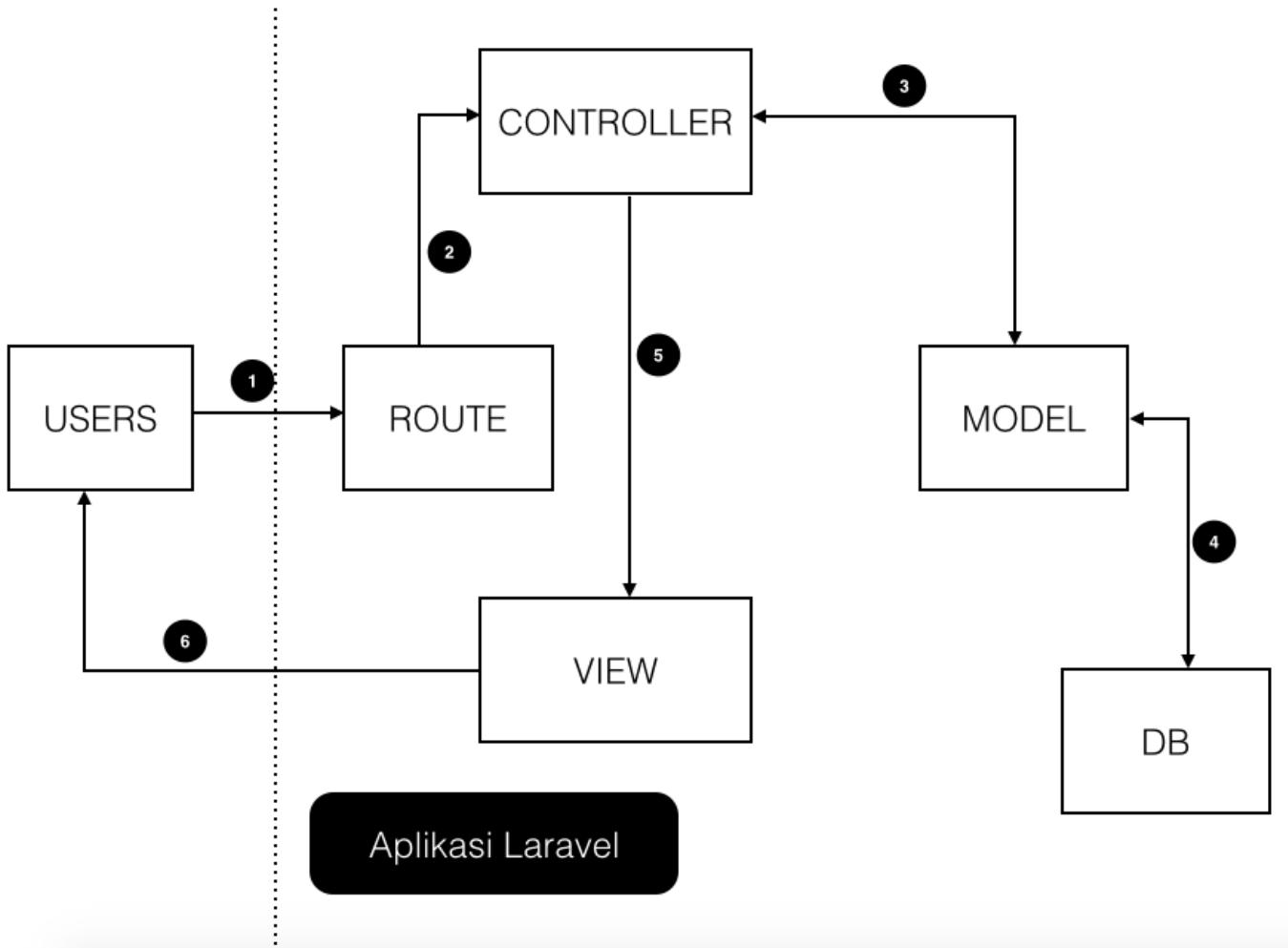


Mengapa model tidak memiliki folder sendiri?

Sebetulnya, pada versi Laravel sebelumnya, setiap komponen memiliki folder khusus, termasuk model. Akan tetapi, banyak yang memiliki interpretasi berbeda terkait model, ada yang menerjemahkan model sebagai entitas yang berinteraksi dengan database saja, ada yang menerjemahkan model sebagai entitas dalam ranah proses bisnis aplikasi saja.

Dengan adanya interpretasi yang berbeda terkait model, akhirnya Laravel membiarkan developer untuk memanfaatkan model sesuai dengan interpretasinya masing-masing dan menghapus folder model. Akan tetapi dengan tidak adanya folder model, Laravel tetap menggunakan model, hanya letaknya saja yang lebih fleksibel, dan dengan begitu Laravel tetaplah mengusung konsep MVC secara utuh.

## Visualisasi MVC di Laravel



Penjelasan:

1. User mengakses aplikasi melalui route tertentu
2. Route tersebut oleh aplikasi telah dipetakan ke controller action
3. Controller action akan menggunakan model untuk mengakses data. Atau langsung mengembalikan view tanpa data (langsung ke step 5)
4. Model berinteraksi ke database untuk mendapatkan data atau menyimpan data
5. Setelah berhasil mendapatkan data melalui model, controller akan mengembalikan sebuah view sekaligus data jika ada.
6. View tersebut pada akhirnya yang dilihat oleh user.

Setelah ini kita akan coba untuk membahas lebih detail ketiga komponen MVC di Laravel. Tetapi sebelum itu, karena kami mencoba untuk menyusun buku ini sesuai dengan alur pembelajaran yang paling memudahkan dalam memahami framework ini, kita akan bahas terlebih dahulu mengenai pengenalan routing, pengenalan model, pengenalan controller dan pengenalan view. Kemudian masing-masing komponen tersebut akan dibahas lebih detil di bab-bab tersendiri.

Pertama kami akan mengenalkan routing karena sebelum diolah oleh MVC, *request* dari user tidak bisa masuk jika tidak ada routing.

## Pengenalan Routing

Jika kita terjemahkan secara bebas, route berarti jalur. Dan hal itu bisa kita bayangkan sebagai konsep routing di Laravel, yaitu jalur URL yang bisa diakses oleh pengguna aplikasi dan ke mana jalur itu diproses.

Contoh pada bahasan hello world sebelumnya, kita telah mendefinisikan sebuah routing sederhana '/helo', route ini bisa diakses di <http://localhost/helo> dan akan menampilkan string "Hello World" sesuai dengan perintah dalam Closure function yang kita tulis sebagai callback. Sintak paling dasar dari routing di Laravel adalah sebagai berikut:

```
Route::verb("/path", callback);
```

Kode di atas dapat kita jelaskan sebagai berikut

1. Facade **Route**.

2. Verb

Verb merupakan HTTP verb, terdiri dari get, post, put, delete, options, patch.

3. Path

Merupakan path URL setelah domain aplikasi kita yang kita izinkan untuk diakses oleh pengguna aplikasi.

4. Callback

Callback bisa berupa *closure* function atau Controller action yang ingin kita eksekusi ketika path tertentu diakses oleh pengguna aplikasi.

*Closure function* merupakan istilah lain untuk function yang tidak memiliki nama dan seringkali digunakan sebagai callback. Jangan bingung ya.

Kita coba ambil contoh dari routing yang kita buat di poin hello world. Kita menuliskan sebuah routing sebagai berikut

```
Route::get("/helo", function(){
    return "Hello World";
});
```

Route di atas bisa kita baca seperti ini, ketika pengguna mengakses URL <http://toko-online.test/helo>, eksekusi closure function di parameter kedua, closure function tersebut menghasilkan string "Hello World". Sangat sederhana dan mudah dicerna, sehingga pengguna akan mendapatkan string Hello World di layar browser mereka.

Akan tetapi penggunaan closure function sebagai route callback jarang sekali kita pakai dalam pembuatan aplikasi sesungguhnya, karena kita memiliki controller.

Ingat, callback selain berupa closure function bisa juga kita isi dengan action dari controller. Sehingga misalnya kita memiliki WelcomeController dengan method beriSalam, kita bisa menulis ulang Route kita sebagai berikut

```
Route::get("/hello", "WelcomeController@beriSalam");
```

Dengan pemisalan method `beriSalam` akan menghasilkan “Hello World”, maka route di atas akan menghasilkan string yang sama ke user seperti halnya ketika menggunakan definisi route yang kita tulis sebelumnya.

Route ke controller perlu diketahui terlebih dahulu, supaya kita paham saat membahas controller, untuk apa menulis controller dan action yang ada di dalamnya. Yaitu untuk kita isikan sebagai *callback* di router, sebagian besar penggunaan controller adalah untuk hal tersebut.

Di mana kita tuliskan definisi routing?

Definisi route bisa kita tulis dengan menuliskan format routing di file `routes/web.php` apabila kita ingin mendefinisikan jalur akses untuk pengguna web standard. Namun jika kita menuliskan definisi routing untuk web service / api service, kita bisa menuliskannya di `routes/api.php`. Sementara itu `routes/console.php` kita gunakan apabila kita ingin membuka jalur akses melalui command line, dan `routes/channel.php` kita gunakan untuk memberikan jalur akses untuk broadcast channel melalui websocket.

Umumnya aplikasi yang kita buat, cukup dengan `routes/web.php` dan `routes/api.php`. Bahkan jika aplikasi kita tidak perlu menyediakan web servie, kita hanya perlu menggunakan `routes/web.php` saja.

## Pengenalan Model

Sebelumnya telah kami jelaskan bahwa ada dua interpretasi berbeda mengenai model, yaitu sebagai entitas yang berinteraksi dengan sumber data dan satu lagi interpretasi model sebagai representasi model di proses bisnis. Nah dalam buku ini, kami mengambil interpretasi **model sebagai entitas yang berinteraksi dengan sumber data**.

Model bertugas untuk query ke database, insert data baru, update, atau hapus record di database. Semua itu dilakukan dengan ORM (Object Relational Mapping) sehingga pada banyak kasus, kita tidak perlu menuliskan kode SQL secara langsung, akan tetapi langsung menggunakan method bawaan dari ORM seperti, `find`, `findOrFail`, `create`, `update`, dll. Dan ORM bawaan yang dipakai oleh Laravel adalah Eloquent. Tentang Eloquent akan kita bahas pada bagian tersendiri di buku ini.

Apa itu ORM? Masih ingat? ORM telah kita bahas pada bab Mengenal Laravel.

## Membuat Model

Untuk membuat model ketikan perintah berikut ini pada terminal

```
php artisan make:model NamaModel
```

Contoh, kita akan membuat model Product, maka perintahnya menjadi seperti ini

```
php artisan make:model Product
```

Dengan begitu, Laravel akan membuatkan sebuah file model kita yang terletak di "app/Product.php". Folder app merupakan folder default untuk pembuatan model. File tersebut apabila dibuka akan terlihat seperti di bawah ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    // code di sini
}
```

Kesepakatan (Konvensi) dalam penulisan model adalah bentuk tunggal dan menggunakan CamelCase dengan kapital di awal, misalnya Product, ProductItem, dll. Setiap model tersebut akan merepresentasikan tabel dengan nama jamak. Dengan demikian model Product yang sudah kita generate merepresentasikan tabel `products` pada database. Kesepakatan tersebut bisa kita ubah, pembahasannya akan kita pelajari bersama di bab Model & Eloquent.

## Pengenalan Controller

Setelah kita mendefinisikan model Product pada sub bab sebelumnya, lalu apa selanjutnya? model Product tersebut belum bisa diakses oleh pengguna aplikasi kita.

Ingat, pengguna bisa menggunakan aplikasi melalui jalur yang kita definisikan pada route. Pada bahasan pengenalan routing, kita telah mempelajari bagaimana cara membuka akses ke aplikasi dengan definisi route. Misalnya,

```
Route::get('/hello', function(){
    return "Hello World";
});
```

Kemudian kita belajar bahwa selain Closure sebagai callback, pada aplikasi sesungguhnya kita akan lebih sering memberikan Controller action sebagai callback route seperti ini

```
Route::get('/products', 'ProductController@index');
```

Pada definisi route di atas, ProductController merupakan nama controller sedangkan index merupakan action pada controller. Tentu jika kita definisikan seperti itu sekarang, dan kita akses maka aplikasi akan error, karena ProductController belum kita buat. Untuk itu mari kita buat terlebih dahulu controller tersebut.

Membuat controller

Untuk membuat controller melalui artisan generator, masuk ke workspace, lalu masuk ke folder project kita, yaitu toko-online, lalu ketik perintah berikut

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

Dengan perintah tersebut maka sekarang kita memiliki file pada [app/Http/Controllers/ProductController.php](#). Dan ini merupakan controller pertama yang kita buat.

Kesepakatan penamaan controller adalah dengan StudlyCaps dan memberikan akhiran Controller pada nama filenya. Misalnya ProductController, UserController, ProductItemController dll.

Kita coba lihat file ProductController.php maka akan kita dapati kode seperti ini

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    //
}
```

ProductController ini belum memiliki action apa-apa. Bisa dikatakan, controller ini masih tidak berguna sekarang, agar controller tersebut berfungsi mari kita tambahkan action.

### Controller action

Controller action merupakan method yang berfungsi untuk melakukan operasi logika. Dalam operasi logika tersebut kita bisa mengambil, mengupdate, menghapus data melalui model lalu mengembalikan sebuah response kepada pengguna aplikasi, baik berupa JSON atau berupa view. Contoh:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function showAll() {
        // kode logika untuk mengambil dan menampilkan semua data
    }

    public function saveNew(Request $request) {
        // kode logika untuk menyimpan data baru
    }
}
```

```
// kode logika untuk menyimpan product baru  
}  
}
```

Pada contoh kode di atas, ProductController memiliki dua action method yaitu showAll dan saveNew. Selanjutnya, kita bisa pakai action tersebut pada definisi Route kita seperti ini

```
Route::get('/product/display', 'ProductController@showAll');
```

```
Route::post('/product/save', 'ProductController@saveNew');
```

Nah sekarang dengan definisi route di atas, apabila pengguna aplikasi mengakses `http://toko-online.test/product/display` maka action showAll pada ProductController akan dieksekusi.

Lalu, pada action tersebut kita bisa gunakan model yang sudah kita buat sebelumnya untuk mengambil data, dan memerintahkan action showAll untuk mengembalikan data ke pengguna aplikasi dalam bentuk view yang berisi data, seperti ini:

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
use \App\Product;  
  
class ProductController extends Controller  
{  
    public function showAll()  
    {  
        $dataProductDariModel = Product::all();  
        return view('product.display', ["products" =>  
$dataProductDariModel]);  
    }  
  
    // action-action lainnya  
}
```

Perhatikan action showAll diatas, pertama kita mengambil seluruh data product menggunakan model Product dengan cara memanggil static method all() seperti ini `Product::all()`; method all() berfungsi untuk mengambil seluruh data pada model tertentu. Method-method model lainnya akan kita bahas pada bahasan Eloquent.

Setelah mengambil data dengan `Product::all()` kita assign ke variable `$dataProductDariModel`, lalu, kita perintahkan action showAll untuk mengembalikan sebuah view yang terletak pada folder `resources/views/product/display.blade.php`, tapi kita cukup menuliskan 'product/display' saja, karena semua views terletak pada folder `resources/views` dan berakhiran dengan `.blade.php`. Lebih detail tentang views akan dibahas di bagian Views di buku ini.

Selain mengembalikan views, kita juga melempar data bernama "products" yang bernilai `$dataproductDariModel`, perhatikan ["products" => `$dataproductDariModel`]. Kita lempar data products, agar file view yang akan kita pakai memiliki variable `$products` untuk kemudian ditampilkan ke pengguna aplikasi sesuai dengan desain yang kita inginkan. Karena belum ada data apa-apa, maka variable `$products` ini berisi empty array.

Coba kamu akses <http://toko-online.test/product/display>, seharusnya kamu akan menjumpai error. Error tersebut berkaitan karena view tidak ditemukan. Lihat controller action `showAll` yang digunakan oleh route tadi. Action tersebut mengembalikan sebuah view `product.display` tetapi kita belum membuat view tersebut. Supaya error tersebut hilang, kita akan berkenalan dengan view setelah ini.

## Pengenalan View

View bertanggungjawab untuk memberikan tampilan ke user. Jika kita ingin meletakkan kode html, css dan javascript di view lah tempatnya. Laravel membuat view lebih powerful dengan memanfaatkan templating engine.

Karena Laravel menggunakan templating engine bawaan Blade, maka file view diakhiri dengan `.blade.php`. Misal `product.blade.php`, `product-list.blade.php`, `product-detail.blade.php`. Dan kita akan mempelajari lebih lanjut di Bab View.

Melanjutkan sub bab sebelumnya, kita memerlukan view `product.display` agar route <http://toko-online.test/product/display> tidak error. Maka kita buat sebuah file baru di `resources/views/product/display.blade.php`. Lalu coba tambahkan text atau kode html, misalnya `<b> View display product siap </b>`. Jika sudah silahkan coba untuk mengakses route yang tadi error. Taraaa... Apa yang terjadi? view berhasil menampilkan pesan.

**Penting!** view `product.display` itu secara default Laravel mengharapkan ada sebuah file di `resources/views/product/display.blade.php`

**Contoh lagi** jika di controller memanggil `view("orders.list")` itu berarti Laravel akan mencari file `resources/views/orders/list.blade.php`. Mudah dimengerti, bukan?

## Kesimpulan

Sampai di sini kita sudah semakin paham alur dari konsep MVC di Laravel, pertama kita definisikan jalur akses URL melalui route, route tersebut kita arahkan ke controller dan action tertentu, action tersebut akan menggunakan model dan query eloquent untuk mengambil data, lalu action tersebut mengembalikan views dengan data data dari model bila diperlukan.

Konsep ini harus kamu pegang dengan baik karena inti dari Laravel adalah MVC. Kamu akan berurusan dengan hal-hal itu, baik untuk aplikasi sederhana maupun kompleks. Dan pada bab-bab selanjutnya kita akan membahas masing-masing komponen MVC dengan lebih detail. Semakin semangat? ^^

# Route & Controller

---

## Route

Nah, setelah di bab sebelumnya kita membahas arsitektur MVC di Laravel. Kini waktunya kita untuk membahas lebih detail komponen MVC. Pertama, kita akan membahas Route dan Controller karena keduanya sangat berkaitan dalam praktiknya.

Pada bab sebelumnya juga kita telah melihat visualisasi MVC, di situ kita melihat bahwa pengguna aplikasi kita akan mengakses aplikasi kita melalui jalur yang kita izinkan. Jalur tersebut adalah route yang akan kita definisikan di file-file route. Jadi, pada bab ini kita akan mulai dengan bahasan Route.

Kita perlu perkuat lagi pemahaman kita, bahwa route ini menjadikan aplikasi kita bisa diakses oleh pihak luar, baik pengguna atau sistem lain jika menggunakan API. Kemudian, route yang didefinisikan akan menggunakan atau memanggil action tertentu dari sebuah controller. Betul? yup, saya yakin kamu sudah memahaminya dari penjelasan di bab arsitektur Laravel.

### Tipe-tipe route

Laravel 5.7 mengenal 4 jenis route, yaitu web route, api route, console route dan channel route. Apa bedanya masing-masing route tersebut? dan kenapa harus ada 4?

#### Web route

Definisi web route ini terletak di `routes/web.php`

Web route digunakan untuk memberikan akses yang dapat diakses melalui HTTP request dan merupakan route yang paling sering digunakan dan kita akan banyak memanfaatkannya. Pada bab pengenalan routing web route ini yang kita bahas, dan di bab ini pun kita akan fokus untuk membahas web route disamping API route.

#### Api route

Definisi API route ini terletak di `routes/api.php`

API route ini mirip dengan web route yaitu sama-sama bisa diakses menggunakan HTTP request. Lalu kenapa mesti dibedakan antara web route dan api route? Itu karena perlakuan untuk mereka memang perlu dibedakan. Contohnya adalah jika kita menggunakan middleware, maka ada 2 middleware group. Intinya mereka menggunakan dua middleware yang berbeda sesuai kebijakan kita untuk masing-masing jenis route tersebut.

#### Console route

Definisi console route ini terletak di `routes/console.php`.

Console route kita gunakan untuk mendefinisikan jalur akses ke aplikasi kita melalui Command Line. Berbeda dengan web route atau api route yang diakses melalui HTTP request, kita bisa menjalankan perintah-perintah tertentu melalui command-line-interface.

Misalnya, kita tambahkan kode berikut ini di file `routes/console.php`

```
Artisan::command('say:hi', function($name){
    $this->info("Hi Fullstack Developer!");
});
```

Maka kini kita memilih sebuah command baru yaitu `say:hi`, jika dieksekusi akan akan menampilkan teks di terminal bernilai "Hi Fullstack Developer!". Cara eksekusinya adalah seperti ini.

```
php artisan say:hi
```

setelah diklik Enter, maka akan muncul tulisan

```
Hi Fullstack Developer!
```

Untuk mendefinisikan route console kita menggunakan facade `Artisan` dan memanggil method command. Kita tidak akan menggunakan command route pada study kasus di buku ini.

## Channel route

Definisi channel route terletak di file `routes/channel.php`

Channel route berfungsi untuk mengotorisasi chanel broadcast di aplikasi kita. Berbeda dengan tiga route sebelumnya di mana callback digunakan untuk logika kode dan bisa menghasilkan view atau data, callback di channel route ini berfungsi untuk mengecek apakah user aktif berhak mengakses channel tertentu.

Itulah kenapa callback di channel route ini selalu mendapatkan argument `$user` sebagai argument pertamanya. Contoh

```
Broadcast::channel('App.User.{id}', function ($user, $id) {
    return (int) $user->id === (int) $id;
});
```

Perhatikan `function($user, $id)`, setiap callback akan mendapatkan `$user` lalu diikuti oleh params yang kita definisikan, contoh di atas adalah `$id`.

Untuk mendefinisikan sebuah route kita gunakan facade `Broadcast` dan memanggil channel method.

Kita telah mengetahui empat jenis route yang ada di Laravel. Kita akan berfokus pada dua route yaitu web route dan api route. Memang, kedua route ini yang paling sering digunakan untuk pembuatan aplikasi berbasis web karena keduanya merupakan pintu akses menggunakan HTTP request.

Cara penggunaan web route dan api route kurang lebih sama dan kita akan bahas keduanya. Dan kita sepakati bahwa istilah route untuk dalam pembahasan-pembahasan selanjutnya merujuk ke web route atau

## Mendefinisikan route

Buka file `routes/web.php` lalu tambahkan kode berikut,

```
Route::get("/pintu-masuk", function(){
    return "Selamat datang di pintu masuk";
});
```

Maka sekarang kita punya akses melalui <http://toko-online.test/pintu-masuk>.

Contoh di atas menggunakan `callback`, selain cara tersebut kita juga bisa menggunakan controller action. Untuk itu, kita buat terlebih dahulu sebuah controller, kita beri nama `TestController`. Caranya dengan menjalankan perintah berikut:

```
php artisan make:controller TestController
```

Lalu buka file yang baru saja digenerate oleh perintah di atas, dan ubah agar menjadi seperti ini:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TestController extends Controller
{
    public function pintuMasuk(Request $request){
        return "Selamat datang di pintu masuk (dari controller)";
    }
}
```

Setelah itu modifikasi definisi route yang baru kita buat menjadi seperti ini:

```
Route::get("/pintu-masuk", "TestController@pintuMasuk");
```

Lalu coba buka kembali <http://toko-online.test/pintu-masuk>, maka kini kamu akan mendapatkan sebuah teks "Selamat datang di pintu masuk (dari controller)". Mudah dipahami bukan cara pendefinisian route baik dengan callback ataupun dengan controller?

## Route Parameter

Sampai di sini, route yang kita definisikan belum menggunakan parameter. Apa itu route param? yaitu variabel tambahan di URL yang ingin kita tangkap di callback atau controller action.

Misalnya, kita ingin melihat sebuah produk dengan id 1, umumnya URL yang kita gunakan adalah semacam ini <http://toko-online.test/products/20>. Ketika mengakses route tersebut, kita menginginkan untuk melihat produk dengan ID 20. Nah ID tersebut kan berubah-ubah sesuai dari pengguna mau melihat produk dengan ID berapa. Maka, ID tersebut bisa kita jadikan route param yang akan kita tangkap nilainya.

Untuk memberikan param pada route caranya adalah dengan menuliskan nama parameter dengan diapit kurung kurawal seperti ini `{id}` atau `{slug}` atau `{user_id}`. Contoh definisi route <http://toko-online.test/products/20> adalah seperti ini:

```
Route::get("products/{id}", "ProductController@show");
```

Lalu pada `ProductController` action `show` kita dapat mengakses nilai `$id` yang diinput oleh user seperti ini.

File `ProductController.php`

```
// ...kode lain di atasnya

// action show
public function show($id){
    /** Nah sekarang kita bisa menggunakan $id
     * untuk misalnya query ke DB product where id == $id
     */
}

//...kode lain di bawahnya
```

Misalnya user mengakses <http://toko-online.test/products/33> maka nilai `$id` adalah 33. Masuk akal bukan?

Urutan parameter menentukan cara aksesnya ya, misalnya kita mengubah di definisi route menjadi seperti ini

```
Route::get("/product/{product_id}", "ProductController@show");
```

Maka, apakah di controller action kita perlu mengubah `$id` menjadi `$product_id`. Jawabnya tidak harus, yang penting urutannya sama. Karena route di atas hanya memiliki 1 params, maka nama apapun di controller action akan merujuk ke param tersebut.

Contoh, route ini

```
Route::get("users/{user_id}/comments/{comment_id}",
"UserController@showComment");
```

Route tersebut memiliki dua params yaitu {user\_id} dan {comment\_id}, maka kita bisa mengaksesnya di **UserController** action **showComment** dalam argument. Argument pertama adalah untuk mendapatkan nilai user\_id dan argument kedua adalah untuk mendapatkan nilai comment\_id, apapun namanya.

```
public function showComment($argument1, $argument2){
    /**
     * $argument1 = nilai dari param {user_id}
     * $argument2 = nilai dari param {comment_id}
     * apapun nama yang kita berikan yaa
     * yang berpengaruh adalah uruatannya.
    */
}
```

Akan tetapi agar kode kita lebih enak dibaca, sebaiknya kita samakan dengan nama route param yang kita definisikan, yaitu \$user\_id dan \$comment\_id.

**Perhatikan** **ProductController** dan **UserController** hanya contoh, kita belum membuatnya, jadi kalo diakses ya pasti error. Sekarang kita belajar definisi routenya terlebih dahulu.

Definisi dengan format seperti {id}, {slug}, {user\_id} atau {nama\_lain} merupakan parameter yang harus ada. Jika ada pengguna mengakses <http://toko-online.test/products>, maka aplikasi akan error karena URL tersebut kurang route parameter. Harusnya kan setelah products ada angka seperti ini [products/40](http://toko-online.test/products/40).

## Optional Route Parameter

Nah untuk membuatnya menjadi opsional, artinya parameter tersebut boleh dikosongkan kita cukup tambahkan **?** di belakang definisi route parameter kita. Contoh:

```
Route::get("/products/{product_id?}", "ProductController@show");
```

Nah, kini param **{product\_id?}** boleh dikosongkan sehingga jika kita mengakses <http://toko-online.test/products> tetap akan memanggil controller bersangkutan, tanpa error.

## Route Berdasarkan Jenis HTTP Method

Sejauh ini kita hanya baru belajar membuat route yang bisa diakses dengan HTTP request method "GET". Tapi pada pengembangan aplikasi kita akan paling minimal membutuhkan empat method HTTP, yaitu GET, POST, PUT, DELETE. Sebelum belajar membuat definisi route untuk keempat method tersebut, bagi yang masih asing dengan empat tipe tersebut, akan kami jelaskan terlebih dahulu.

### GET Method

GET method merupakan method paling dasar dan umum digunakan. Jika kita mengakses alamat URL sebuah di browser, maka itu termasuk contoh penggunaan GET method.

## POST Method

POST method umumnya digunakan untuk melakukan proses submit data ke aplikasi. Sehingga sering kita temukan penggunaannya pada form. Dan memang POST method adalah method yang direkomendasikan untuk mengirim data dari form karena lebih aman daripada GET method.

## PUT Method

PUT method mirip dengan POST method yaitu digunakan juga untuk mengirimkan data ke aplikasi baik dari form ataupun ajax request. Selain itu juga dari segi keamanan dan perilaku hampir sama dengan POST method. Bedanya PUT method lebih sering digunakan untuk mengirimkan data yang bersifat UPDATE bukan CREATE. Misalnya di form edit.

Selain PUT method, method lain yang digunakan untuk mengupdate adalah PATCH.

## DELETE Method

DELETE method sesuai namanya digunakan untuk melakukan penghapusan sebuah data di aplikasi tertentu. Biasanya kita akan mengirimkan ID data tertentu yang ingin kita hapus baik melalui form atau ajax request. Kita mengirimkan data tersebut dengan method DELETE.

Nah setelah kita memahami keempat method dasar di atas, maka kini kita akan gunakan masing-masing method tersebut sesuai kegunaannya ya. Dan Laravel pun akan membedakan request yang datang itu menggunakan method apa, untuk itulah kita pun perlu menuliskan di definisi route kita, sebuah jalur aplikasi kita boleh diakses menggunakan method apa.

Hal ini untuk mencegah pengiriman data sensitif menggunakan method GET misalnya. Sekarang kita akan belajar membuat route untuk masing-masing method tersebut.

Method GET untuk melihat daftar produk pada alamat <http://toko-online.test/products/list> dapat didefinisikan seperti ini:

```
Route::get("/products/list", "ProductController@list");
```

Method POST untuk menyimpan produk baru dari sebuah form yang akan dikirimkan ke alamat misalnya <http://toko-online.test/product/create>, maka definisi routenya seperti ini:

```
Route::post("/products/create", "ProductController@create");
```

Kemudian, sebuah fitur untuk mengupdate produk yang diedit dari sebuah form dikirimkan ke alamat misalnya <http://toko-online.test/products/1> dengan method PUT. Definisi routenya seperti ini

```
Route::put("/products/{id}", "ProductController@update");
```

**Perhatian!** Jika kita memiliki 2 route yang sama-sama menggunakan url "products" misalnya, tapi http methodnya berbeda, maka ini dihitung 2 route yang berbeda!

Contoh:

```
Route::get("products", "ProductController@index");
```

dan

```
Route::post("products", "ProductController@store");
```

Dua route di atas merupakan route yang berbeda meskipun cara aksesnya sama-sama `http://toko-online.test/products` akan tetapi yang pertama boleh diakses menggunakan GET method dan akan memanggil controller action `index` pada `ProductController` dan yang bawah hanya boleh diakses melalui method POST dan akan memanggil controller action `store` pada `ProductController`

Mengizinkan lebih dari satu HTTP method

Misalnya kita ingin route `products/43` bisa diakses dengan method PUT dan PATCH, maka kita bisa menuliskan definisi routenya seperti ini:

```
Route::match(["PUT", "PATCH"], "/products/{id}",
"ProductController@update");
```

Atau jika kita ingin mengizinkan semua HTTP method untuk sebuah route maka kita bisa menulisnya seperti ini:

```
Route::any("/products/{id}", "ProductController@action");
```

Named Route

Kita juga memberikan nama untuk route yang kita definisikan. Caranya adalah seperti ini

```
Route::get("/products/list", "ProductController@list")->name("products");
```

Dengan begitu kini route dengan path `/products/list` mempunyai nama "products". Route yang memiliki nama ini bisa kita gunakan dengan helper lainnya. Misalnya dengan url generator seperti ini,

```
$url = route("products"); // -> http://toko-online.test/products/list
```

atau redirect ke route tersebut cukup dengan

```
redirect()->route("products") // akan redirect ke http://toko-
online.test/products/list
```

## Route Group

Dalam pembuatan aplikasi terkadang kita memiliki route yang punya banyak kesamaan. Misalnya, perhatikan url-url di bawah ini:

- <http://toko-online.test/products/all>
- <http://toko-online.test/products/bag>
- <http://toko-online.test/products/latest>
- <http://toko-online.test/products/discounts>

Adakah kesamaan yang bisa kamu lihat? Ya, semua route di atas memiliki "/products" di URL mereka. Alih-alih mendefinisikannya seperti ini:

```
Route::get("/products/all", "ProductController@all");
Route::get("/products/bag", "ProductController@bag");
Route::get("/products/latest", "ProductController@latest");
Route::get("/products/discounts", "ProductController@discounts");
```

Kita bisa membuat kelompok route (Route Group) seperti ini:

```
Route::group(["prefix"=>"products"], function(){
    Route::get("/all", "ProductController@all");
    Route::get("/bag", "ProductController@bag");
    Route::get("/latest", "ProductController@latest");
    Route::get("/discounts", "ProductController@discounts");
});
```

Definisi di atas akan berfungsi sama seperti definisi route sebelumnya. Kapan kita memerlukan route group? Jawabannya adalah ketika kita tidak ingin menuliskan sesuatu yang sama berulang kali. Selain itu juga kita bisa menerapkan policy melalui middleware tertentu ke dalam semua route di dalam sebuah route group.

Cukup dengan menambahkan key middleware ke dalam array config seperti ini

```
Route::group(["prefix"=>"admin", "middleware"=>"mustAdmin"], function(){
    Route::get("/dashboard", "DashboardController@index");
    Route::get("/orders", "DashboardController@index");
    // definisi route lainnya di sini
});
```

Route di atas akan membuka jalur akses URL yaitu pada

- <http://toko-online.test/admin/dashboard>
- <http://toko-online.test/admin/orders>

keduanya hanya bisa diakses oleh admin karena diterapkan middleware `mustAdmin` dalam definisi route groupnya. Lebih jauh tentang middleware akan kita bahas nanti. Kini kamu mengerti kenapa route group bisa sangat bermanfaat.

## Route View

Bisa juga kita melewati controller dari route, hal ini bisa kita lakukan jika kita ingin menampilkan view tanpa data yang kompleks, atau view yang bersifat statis. Caranya adalah seperti ini:

```
Route::view("/path", "nama.view");
```

**Penjelasan kode:** Dengan kode di atas kita bisa mengakses <http://toko-online.test/path> dan view `nama.view` yang terletak pada `resources/views/nama/view.blade.php` akan ditampilkan tanpa melalui controller.

## Controller

### Anatomi controller

Controller hanya sebuah class yang mengextends class `Controller`, akan tetapi akan lebih mudah jika kita dapat melihat anatomi dari controller ini. Anatomi paling dasar agar kita bisa menggunakan controller hanyalah action. Yup, controller harus memiliki action. Berikut adalah anatomi controller kosong yang belum bisa kita gunakan.

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class ItemController extends Controller  
{  
  
}
```

Kode diatas merupakan definisi class `ItemController`, karena controller tersebut tidak memiliki action apapun maka controller tersebut belum bisa digunakan. Agar kita bisa mulai memanfaatkannya, mari kita tambahkan sebuah action misalnya kita beri nama `showAllItems`, maka kita controller tersebut akan menjadi seperti ini:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ItemController extends Controller
{
    public function showAllItems(){
        // logika kode untuk mengambil dan memberikan semua items
        return "Tunjukan semua items"
    }
}
```

Nah, kini **ItemController** memiliki satu action yaitu **showAllItems** yang nantinya bisa kita gunakan untuk mengambil semua item dari database lalu melemparnya ke view atau melemparnya sebagai JSON. Sementara kita lemparkan dulu sebuah string.

Karena sekarang **ItemController** memiliki sebuah action maka kita bisa menggunakan dalam definisi route kita, misalnya seperti ini

```
Route::get("show-all-items", "ItemController@showAllItems");
```

Sehingga ketika pengguna aplikasi kita mengakses <http://toko-online.test/show-all-items> dia akan mendapatkan tulisan "Tunjukan semua items".

Semakin paham betapa route dan controller saling berkaitan bukan?

**Tips** Kita bisa menambahkan banyak action selain **showAllItems** misalnya **showOnlyDiscountedItem** atau lainnya. Bebas, tipsnya taruh action controller sesuai dengan controller yang sesuai ya. Misalnya, kan ga masuk akal kalo kita membuat action **showAllUsers** (menampilkan semua pengguna) pada **ItemController**.

## Membuat controller

Tentu kita bisa membuat controller secara manual dengan membuat file pada folder **app/Http/Controllers** lalu menuliskan definisi class controller kita. Akan tetapi akan sangat merepotkan kalo kita berulang-kali menulis kode tersebut setiap kali ingin membuat sebuah controller.

Cara paling mudah dan direkomendasikan adalah menggunakan perintah artisan, seperti yang telah kita pelajari pada bab pengenalan controller. Yaitu dengan menjalankan perintah berikut:

```
php artisan make:controller NamaController
```

misalnya

```
php artisan make:controller ItemController
```

Perintah tersebut akan membuatkan kita sebuah controller baru pada path `app/Http/Controllers/ItemController.php` dan mengisinya dengan anatomi controller kosong.

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class ItemController extends Controller  
{  
  
}
```

## Controller Resource

Pola paling dasar dari sebuah aplikasi adalah CRUD (Create, Read, Update, Delete). Yaitu pengguna aplikasi dapat membuat data baru, membaca / melihat data, mengupdate data tertentu dan menghapusnya.

Untuk mengakomodasi hal tersebut, sebuah controller harus memiliki beberapa action standard. Action tersebut antara lain index, create, show, edit, update dan delete. Istilah untuk menyebut kesemua fungsionalitas tersebut adalah **resource** sehingga semua action tadi kita sebut sebagai **actions resource** dan controller yang memiliki beberapa action tersebut kita sebut **controller resource**.

### Action resource

Nah apa sih kegunaan dari masing-masing action resource yang dimiliki oleh controller. Mari kita bahas satu per satu.

#### index

Action **index** digunakan untuk menampilkan list atau daftar data dari sebuah controller. Misalnya `ProductController` menggunakan action **index** untuk mengambil daftar products dari database lalu menampilkannya ke view.

#### create

Action **create** digunakan untuk menampilkan view yaitu berupa form untuk membuat / insert data baru. Misalnya `ProductController` menggunakan action **create** untuk menampilkan sebuah form isian produk.

#### store

Action **store** digunakan untuk menerima data yang diinput melalui form dari action **create** sebelumnya. Tugas dari action **store** adalah untuk menerima data lalu menyimpannya ke tabel tertentu, misalnya tabel

product.

#### show

Action **show** digunakan untuk menampilkan data tertentu, bukan daftar. Misalnya **ProductController** akan menggunakan action **show** untuk menampilkan detail product dengan ID tertentu. Sehingga action ini akan memerlukan route param yang berisi ID untuk query ke database.

#### edit

Action **edit** seperti halnya action **create** digunakan untuk menampilkan form isian untuk data tertentu. Bedanya form edit akan diisi dengan data tertentu bukan form kosong. Karena tujuan dari edit adalah untuk melakukan edit data yang sebelumnya sudah ada.

Di action **edit** kita juga memerlukan route param yang berisi ID data tertentu yang akan diedit. Misalnya **ProductController** akan menggunakan action **edit** untuk mencari data produk dengan ID bernilai dari route param, lalu mengembalikan sebuah view beserta data produk yang akan diedit.

#### update

Action **update** serupa dengan action **store** yaitu berfungsi untuk menyimpan data tertentu ke database. Sekali lagi, bedanya adalah **update** menerima data isian yang dikirim dari action **edit** bukan **create**.

Action ini juga memerlukan route param yang berisi ID tertentu yang akan diupdate dengan data yang diterima dari action **edit**

#### destroy

Action **destroy** digunakan untuk menghapus data tertentu dari database berdasarkan ID tertentu. Oleh karenanya action ini membutuhkan route param yang bernilai ID data tertentu yang akan didelete.

### Membuat Controller Resource

Kita bisa membuat controller resource dengan command seperti pembuatan controller pada umumnya. Bedanya, kita tambahkan **--resource** agar anatomi controller yang digenerate tidak kosong tetapi memiliki semua action resource. Coba jalankan perintah pembuatan controller resource seperti ini:

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

Jika sudah kini kita memiliki controller **CategoryController** yang sudah siap dengan action resource. Kode yang digenerate adalah seperti ini:

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;
```

```
class CategoryController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

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

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }
}
```

```

}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    //
}
}

```

Perhatikan, jika menggunakan `--resource` controller yang dihasilkan tidak kosongan lagi, tetapi sudah siap dengan `action resource`. Dan tugasmu tinggal menuliskan kode logika di masing-masing action itu. Kita akan berlatihnya pada bab Model & Eloquent nanti.

### Membuat Route Resource

Setelah kita memiliki `CategoryController` di atas dengan `action resource` lalu bagaimana kita menghubungkannya dengan route? apakah kita definisikan seperti sebelumnya yaitu seperti ini:

```

Route::get("/categories", "CategoryController@index");

Route::get("/categories/create", "CategoryController@create");
Route::post("/categories", "CategoryController@store");

Route::get("/categories/{id}", "CategoryController@show");

Route::get("/categories/{id}/edit", "CategoryController@edit");
Route::put("/categories/{id}", "CategoryController@update");

Route::delete("/categories/{id}", "CategoryController@destroy");

```

Cara tersebut boleh-boleh saja dan akan berfungsi dengan baik. Perhatikan bahwa tiap-tiap action controller menggunakan HTTP Method yang berbeda ya. Jika dieprhatikan mappingnya adalah seperti ini:

Action	HTTP Method
index	GET
create	GET
store	POST
edit	GET
update	PUT / PATCH
destroy	DELETE

Okay ya sekarang kita sudah tahu gimana menghubungkan route dengan controller resource, sama aja sebetulnya, karena controller resource ya hanyalah controller.

Tapi tunggu dulu, ada sesuatu yang keren, jika kita memiliki controller resource ada cara cepat untuk mendefinisikan route kita, tidak perlu satu per satu untuk tiap-tiap action seperti yang kita lakukan di definisi route di atas. Cara yang direkomendasikan dan lebih ringkas adalah seperti ini:

```
Route::resource("namaresource", "NamaControllerResource");
```

Misalnya untuk controller resource **CategoryController** yang sudah kita buat, maka kita cukup mendefinisikan route kita seperti ini:

```
Route::resource("categories", "CategoryController");
```

Dan **BOOM!** secara otomatis kita memiliki route sejumlah **action resource** yang ada pada **CategoryController**, sama persis seperti jika kita definisikan satu per satu seperti sebelumnya. Simpel dan singkat bukan?

## Membaca Input & Memberikan Response

Kini mari kita kembali berfokus pada controller action, karna di sinilah kita bisa membaca input dari pengguna maupun memberikan response kepada request pengguna.

### Input & Query String

Sebelum kita membaca input & query string, mari sedikit refresh apa itu input dan query string. Sebelumnya kita belajar bahwa ada beberapa HTTP method yang bisa kita gunakan dalam request. Dan kita juga belajar bagaimana POST method cenderung lebih aman ketimbang menggunakan GET method. Meskipun demikian keduanya memiliki kegunaannya masing-masing. Misalnya perhatikan URL google search, kamu akan menemukan URL kurang lebih seperti ini

```
https://google.co.id?q=buku+laravel+lengkap+bagus
```

Request ke search engine tersebut menggunakan method GET, kenapa? yang pertama karena kita bisa langsung mengetikkannya di address bar browser kita. Kedua, karena data yang kita kirim yaitu keyword dalam bentuk **query string** perhatikan bagian `?q=buku+laravel+lengkap+bagus` itu merupakan **query string**. Di contoh tersebut terdapat **query string** bernama `q` dan bernilai `buku laravel lengkap bagus`. Inilah salah satu penggunaan GET method dan **query string** nya, dan ketika kita mengubah langsung di browser keyword, maka tidak ada masalah. Sampai di sini kita sudah paham apa itu **query string** bukan?

Sekarang bayangkan jika form login aplikasi kita menggunakan method GET, itu berarti kita mengirim username dan password melalui **query string** seperti ini:

```
http://toko-online.test/login?username=azamuddin&password=forzalaravel
```

Kan tidak lucu dan tidak aman kalo password yang notabene sensitif dikirim menggunakan **query string** seperti itu. Inilah kenapa pada data yang sensitif kita jangan menggunakan method GET, tapi gunakan method POST yang lebih aman, karena data yang dikirim ke server tidak tampak di URL tetapi terbungkus dalam payload. Nah, kita menyebutnya data tersebut dengan **input**. Jadi sekarang kita paham yang kita maksud dengan **query string** dan **input** yaitu:

1. **query string**

Data yang dikirim menggunakan method GET.

2. **input**

Data yang dikirim menggunakan method POST atau PUT atau PATCH. Kenapa ketiganya sekaligus? karena sifat mereka sama dalam mengirimkan data.

Jadi jika menggunakan method POST maka url login kita akan seperti ini:

```
http://toko-online.test/login
```

Hanya seperti itu, akan tetapi terdapat input berupa **username** dan **password** yang dikirim ke URL tersebut yang tidak terlihat di URL.

Nah, di Laravel tugas menangkap **input** ataupun **query string** akan kita letakkan di **controller action**. Dan kita akan belajar bagaimana caranya.

## Membaca Input & Query String

Misalnya dalam **ProductController** action `create` kita ingin menangkap data yang diinput oleh user untuk membuat produk baru. Kita gunakan bantuan `$request->get("nama_input")` seperti ini:

```
// file ProductController.php  
  
public function create(Request $request){
```

```
$product_name = $request->get("product_name");
$product_stock = $request->get("stock");
$product_desc = $request->get("description");
$product_price = $request->get("price");
}
```

Ketika pengguna mengirimkan / submit data product dari sebuah form misalnya ke URL <http://toko-online.test/products> dengan method POST. Maka dengan kode di atas kita menangkap input dari user yaitu berupa `product_name`, `stock`, `description` dan `price` lalu kita simpan ke dalam variabel masing-masing. Dengan begitu kita siap untuk menyimpan produk baru ke database dengan data yang baru saja kita dapatkan.

Untuk menangkap query string kita juga menggunakan fungsi yang sama yaitu `$request->get("nama_query_string")`. Sehingga misalnya kita memiliki fitur search pada URL ini:

```
http://toko-online.test/search?product_name=Sepatu
```

Url tersebut kita konfigurasi ke route berikut

```
Route::get("search", "ProductController@search");
```

Maka di action search kita akan menangkap query string `product_name` seperti ini:

```
// File ProductController.php

public function search(Request $request){
    $keyword = $request->get("product_name");

    // Lalu kita akan mencari ke tabel product berdasarkan product.name ==
    $keyword
}
```

**TIPS!** Selain menggunakan `$request->get` kita bisa menggunakan `$request->input`.

#### Membaca semua input & query string

Seandainya kita memiliki 20 input yang harus ditangkap, tampaknya menangkap satu per satu terlalu melelahkan. Bagaimana kalo kita tangkap saja semuanya dan menyimpannya dalam variabel. Bisa! Seperti ini:

```
$data = $request->all();
```

Jika ada 20 input, enakan begini kan daripada harus satu per satu? `$data` berisi seluruh input dari request yang dikirimkan user.

### Mengecualikan input & query string

Setelah kita belajar cara mudah menangkap seluruh input, kita punya masalah, misalnya dari seluruh input ada input yang ingin kita kecualikan. Contohnya kita ingin mengecualikan input `_token` karena kita tidak butuh input tersebut untuk disimpan ke database. Dan justru akan error jika tidak dikecualikan. Maka kita bisa melakukannya seperti ini:

```
$data = $request->except(["_token"]);
```

Kini `$data` berisi seluruh input kecuali input bernama `_token`. Dan kita bisa mengecualikan lebih dari satu field seperti ini `$request->except(["_token", "field_lain"])`.

### Memberikan Response

Setelah kita menangkap input dari user pada controller maka dalam praktiknya kita akan memproses input tersebut baik menginsert data baru atau mengupdate data yang sudah ada. Lalu setelah proses itu kita memberikan response kembali ke user baik menginfokan bahwa insert berhasil, atau redirect ke halaman lain jika error dan lain-lain. Oleh karena itu maka, kini kita akan belajar memberikan response dari action controller.

### Response dasar

Cara paling dasar untuk memberikan reponse adalah dengan `return` dari controller action seperti ini, kita gunakan `CategoryController` yang sudah kita buat ya.

```
// file CategoryController
public function index(){
    return "DAFTAR CATEGORY";
}
```

Buka `http://toko-online.test/categories` maka akan muncul tulisan "DAFTAR CATEGORY".

Cukup mudah dan kita sudah mempelajarinya di bab pengenalan.

### Response redirect ke halaman lain

Kita bisa melakukan redirect ke halaman lain dengan bantuan Facade `\Redirect::to("path")` seperti ini:

```
public function index(){
    return \Redirect::to("/categories/10/edit");
}
```

Dengan kode di atas maka bila kita akses <http://toko-online.test/categories>, kita akan diarahkan ke halaman <http://toko-online.test/categories/10/edit>.

### Response redirect ke website lain

Facade `\Redirect::to` untuk meredirect ke halaman lain dalam aplikasi kita tetapi tidak bisa untuk meredirect ke URL di luar aplikasi kita. Kita memerlukan method lain untuk melakukannya yaitu `\Redirect::away("URL")` seperti ini:

```
public function index(){
    return \Redirect::away("https://google.co.id");
}
```

### Response view

Pada praktiknya kita akan meresponse dengan memberikan sebuah view untuk ditampilkan ke user apliksi. Untuk melakukannya kita bisa menggunakan helper `view()` seperti ini:

```
public function index(){
    return view("category/index"); // -> harus ada file
resources/views/category/index.blade.php
}
```

Maka jika kita mengakses <http://toko-online.test/categories> kita akan mendapatkan tampilan sesuai dengan isi file view di `resources/views/category/index.blade.php`

Lebih lanjut tentang view akan kita bahas di bab tersendiri nanti.

## Kesimpulan

Route dan controller merupakan *backbone* atau tulang punggung dari aplikasi kita karena dengannya kita bisa menghubungkan user aplikasi melalui jalur-jalur yang didefinisikan pada route ke kode logika aplikasi yang kita taruh dalam controller action.

Dengan route kita memberikan jalur akses ke pengguna yang kemudian request mereka akan diproses oleh controller. Kita belajar bagaimana menangkap data yang diinput oleh pengguna melalui request. Kita juga telah belajar lebih detail tentang tipe-tipe route dan seluk beluknya. Juga kita telah memahami controller dan controller resource.

Pengetahuan tentang route dan controller di bab ini sangatlah cukup untuk kita beranjak ke materi selanjutnya.

# Database

---

## Intro

Setelah kita belajar tentang route dan controller maka apa selanjutnya yang harus kita pelajari? Untuk menjawab pertanyaan ini coba kita ingat-ingat lagi bagaimana kita telah belajar untuk menangkap input dari user dan memberikan response dari controller. Akan tetapi kita belum melakukan apapun terhadap input-input tersebut.

Secara logika, maka selanjutnya adalah saatnya kita melakukan interaksi ke dalam database. Yakni dengan input dari user aplikasi, kita bisa membuat data baru, mengupdate atau menghapus data yang sudah ada. Cara termudah dan yang direkomendasikan sesuai dengan arsitekur MVC di Laravel adalah menggunakan Model dan memanfaatkan Eloquent.

Namun sebelum kita bisa menggunakan Model dan Eloquent, terlebih dahulu kita harus melakukan persiapan database terlebih dahulu. Untuk itulah, sebelum belajar tentang Model dan Eloquent, terlebih dahulu pada bab ini kita akan belajar tentang database. Siap? Ayo kita lanjutkan!

## Konfigurasi Koneksi database

Agar kita bisa bekerja dengan database sebelumnya kita harus melakukan konfigurasi koneksi database terlebih dahulu. Untuk melakukannya kita cukup isi variabel yang tersedia pada file konfigurasi database yang terletak pada config/database.php. Misalnya kita akan menghubungkan aplikasi Laravel kita dengan mysql, maka kita langsung menuju konfigurasi mysql seperti berikut pada file tersebut:

File config/database.php

```
'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' => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix' => '',
    'strict' => true,
    'engine' => null,
],
```

**Penjelasan kode:** File tersebut berfungsi untuk menyimpan konfigurasi koneksi database aplikasi Laravel kita, di file tersebut ada beberapa opsi driver database apa yang akan digunakan seperti mysql, pgsql untuk postgreSQL, sqlite dan sqlsrv untuk MSSQL. Kode di atas merupakan konfigurasi untuk mysql, isikan host dengan host server mysql kita, misalnya 127.0.0.1 atau localhost, kemudian isikan user dan password untuk mysql di host tersebut, kemudian isi nama database yang akan kita gunakan.

Tetapi bila kita perhatikan ternyata file tersebut menggunakan fungsi `env("DB_HOST", "127.0.0.1")`, itu artinya file itu akan menggunakan nilai yang ada pada variabel lingkungan (telah kita pelajari pada bab instalasi dan konfigurasi). Maka, kita isikan konfigurasi koneksi kita pada file `.env` di root direktori aplikasi kita. Coba kita buka file `.env` tersebut maka kita dapatkan blok kode seperti ini:

```
// File .env
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root
```

**Penjelasan Kode:** File ini merupakan file untuk menyimpan variabel lingkungan, di sini kita bisa mendefinisikan konfigurasi koneksi database kita. `DB_CONNECTION` merupakan settingan global settingan konfigurasi apa yang akan dipakai untuk *connect* ke database, nilainya mysql karena kita akan menggunakan konfigurasi mysql pada file `config/database.php`. Kemudian, alih-alih kita mengisi langsung detail koneksi seperti `DB_HOST` langsung di `config/database.php`, kita menaruhnya di file `.env`. Sehingga kita isikan semua detail koneksi di sini seperti `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`. Selanjutnya konfigurasi ini akan diakses oleh file konfigurasi `config/database.php` menggunakan fungsi `env(DB_HOST, 'nilai default')`, `env(DB_PORT, 'nilai default')` dst.

Baik, untuk yang develop menggunakan XAMPP silahkan buka file `.env` yang berada di root folder aplikasi kita. Lalu isikan kode ini:

```
// File .env
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=toko-online
DB_USERNAME=root
DB_PASSWORD=root
```

Sementara itu untuk yang lingkungan developmentnya menggunakan Laradock silahkan isikan konfigurasi seperti ini:

```
// File .env
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=toko-online
DB_USERNAME=root
DB_PASSWORD=root
```

Sedikit perbedaan bagi pengguna laradock, yaitu `DB_HOST = mysql` itu karena kita menggunakan docker service mysql dari laradock.

## Metode dalam Bekerja dengan database

Bekerja dengan database menggunakan Laravel agak sedikit berbeda apabila dibandingkan dengan pengembangan aplikasi tanpa framework. Terutama bila kita sebelumnya langsung menggunakan SQL sintaks pada kode PHP. Secara umum alur untuk bekerja dengan database tanpa Laravel adalah seperti ini.

Buat desain DB -> Buat Table & Struktur melalui GUI -> Connect -> Query SQL

Hampir sama sebetulnya, akan tetapi dengan Laravel kita memiliki tools yang memudahkan kita dalam mengelola database, bahkan tanpa harus berpindah dari teks editor favorit kita. Kita tidak perlu bolak-balik pindah aplikasi. Cukup dengan kode yang kita punya. Saya kelompokkan dua metode atau cara dalam bekerja dengan database di Laravel yaitu Code First dan Database First.

### Code First

Dengan metode ini, kita tidak perlu mendesain, membuat database bahkan melakukan insert data awal / dummy melalui aplikasi yang biasa kita gunakan seperti phpMyAdmin, Adminer, dll. Karena kita cukup mendesain dan membuat desain database langsung dari kode kita. Untuk membuat skema database kita memanfaatkan fitur Laravel yaitu Migration, sedangkan untuk mengisi data awal atau data dummy kita bisa menggunakan Seeding. Keduanya akan kita bahas di bab ini.

### Database First

Metode kedua adalah database first, hal ini jika kita sudah memiliki database, tabel dan skemanya, dan kita langsung ingin menggunakan tanpa membuat database baru. Metode ini juga bisa diterapkan jika kita lebih suka mendesain skema tabel langsung dari phpMyAdmin misalnya. Dengan metode ini, kita tidak perlu menggunakan Migration.

Karena metode Database first sudah sering kita gunakan, maka pada buku ini kita akan belajar menggunakan metode Code First. Kita akan belajar tentang migration dan seeding.

## Migration

Migration berfungsi sebagai version control database kita. Dengan migration kita bisa membuat, mengubah atau menghapus struktur tabel dan field database tanpa harus membuka aplikasi GUI database management. Dengan Migration inilah, kita menggunakan metode Code First.

### Membuat File Migration

Pertama kita perlu membuat sebuah file Migration, file ini nantinya bisa dieksekusi agar dapat mengubah database sesuai perintah yang kita tuliskan di file tersebut. Untuk membuat file Migration kita gunakan perintah ini:

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

**Penjelasan Kode:** Ketikan perintah itu pada terminal di root direktori aplikasi Laravel kita. Dengan begitu sebuah file akan dibuat di direktori `database/migrations`. File tersebut berakhiran dengan nama yang kita tulis di perintah tadi yaitu `create_products_table.php` dan diawali dengan stempel waktu kapan file tersebut dibuat, misalnya `2018_03_19_074052_create_products_table.php`. Stempel waktu itu penting karena dipakai oleh Laravel untuk menentukan urutan migrasi saat kita mengeksekusi Migration tersebut.

Selanjutnya, coba kita buka file yang baru saja kita buat maka akan kita dapati seperti ini

```
<?php

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

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

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

**Penjelasan Kode:** File Migration ini memiliki 2 method yaitu `up()` dan `down()`. Method `up()` digunakan untuk menuliskan perintah pembuatan atau pengubahan struktur database. Method `down()` digunakan untuk menuliskan kode yang membatalkan apa yang telah dieksekusi di method `up()`.

Pada method `up()`, pertama kita buat sebuah tabel dengan nama `products`, hal itu dilakukan melalui static method dari facade `Schema`, yaitu `Schema::create("products")`, setelah itu kita definisikan field apa saja yang akan dibuat, secara default perintah `make:migration` akan membuat 2 field untuk kita yaitu `id` yang merupakan `key` dan `autoIncrement`, kemudian `timestamps` yang akan membuat field `created_at` dan `updated_at` pada tabel `products`.

Sementara itu method `down()`, menggunakan perintah `Schema::dropIfExists("products")` untuk mengecek jika tabel product ada di database kita maka jalankan perintah drop, atau hapus tabel products.

Dengan tergeneratenya file Migration kita ini, database kita belum berubah, untuk mengubahnya kita perlu mengeksekusinya.

## Mengeksekusi Migration

Untuk mengeksekusi migration, pertama kita buat dulu file Migration. Setelah itu buka terminal dan masuk ke root path aplikasi Laravel kita, jalankan perintah ini:

```
php artisan migrate
```

**Penjelasan Kode:** Dengan perintah di atas, maka file-file Migration yang berisi perintah pengubahan database akan dijalankan, hanya kode di method `up()` yang dieksekusi. Untuk melihat perubahannya, kita bisa melihat langsung tabel di database kita.

Coba jalankan perintah migrate di atas setelah kamu berhasil menggenerate file migration. Setelah berhasil menjalankan, coba kamu buka database kamu, bisa melalui PHPMyadmin untuk mengecek apakah benar tabel sudah berhasil dibuat.

Buka localhost/phpmyadmin bagi yang menggunakan xampp dan localhost:8080 bagi yang menggunakan Laradock. Lalu cek database `toko-online` dan lihat apakah sudah ada tabel `product?` Sudah bukan? padahal kita tidak membuatnya dengan GUI.

## Migration Rollback

Migration rollback digunakan untuk mengembalikan database kita ke versi sebelum menjalankan perintah migrate. Misalnya kita banyak memiliki file migration, maka rollback akan mengembalikan database kita ke versi terakhir sebelum perintah migrate terakhir kita jalankan. Ketika melakukan rollback, maka method `down()` pada masing-masing file Migration yang dieksekusi, itulah kenapa ada 2 method `up()` dan `down()`.

## Schema Builder

Schema Builder merupakan facade yang bisa kita gunakan untuk mengubah struktur database kita. Umumnya kita gunakan pada file Migration, dan kita sudah melihat contohnya pada bahasan sebelumnya. Perhatikan kode `$table->increments("id")` dan `$table->timestamps()`, keduanya kita lihat pada file Migration yang kita generate sebelumnya, dan berfungsi untuk mengubah struktur database melalui kode. Dan kedua method itulah salah satu contoh method dari `Schema Builder`. Termasuk pula kode `Schema::create("products")` yang berfungsi untuk membuat tabel baru dengan nama sesuai parameter yang diberikan. Apa saja method di Schema Builder yang penting kita ketahui untuk mengubah struktur database?

## Membuat Tabel

Untuk membuat tabel baru kita gunakan perintah `Schema::create("nama_tabel", callback)` seperti ini:

```
Schema::create("products", function(Blueprint $table){
    // kode untuk mengubah kolom di sini
});
```

**Penjelasan kode:** Kode ini jika dijalankan oleh perintah php artisan migrate, akan membuat tabel baru dengan nama products, kemudian callback pada argument kedua digunakan untuk selanjutnya menulis kode untuk mengubah struktur tabel.

## Memilih Tabel

Selain membuat tabel baru, file Migration kita juga bisa melakukan perubahan struktur column dari suatu tabel yang sudah ada. Untuk melakukannya kita gunakan method table, buka create seperti ini

```
Schema::table("products", function(Blueprint $table){
    // kode untuk mengubah kolom di sini
});
```

**Penjelasan kode:** Kode ini jika dijalankan tidak akan membuat tabel baru, tetapi memilih tabel products yang sebelumnya sudah ada. Syaratnya adalah memang tabel yang kita pilih sudah ada di database kita. Jika belum, maka kita gunakan Schema::create();

## Mengecek apakah column / field sudah ada

Setelah menuliskan kode perintah membuat tabel, selanjutnya kita ingin menambah sebuah column dengan nama description, akan tetapi asumsikan kita tidak tahu apakah tabel tersebut sebelumnya pernah dibuat dan sudah memiliki column tersebut. Untuk mengecek apakah sebuah column sudah ada, kita gunakan method hasColumn("nama\_kolom\_untuk\_dicek") pada Schema table seperti ini:

```
Schema::table("products", function(Blueprint $table){
    $table->increments("id");

    $descriptionSudahAda = $table->hasColumn("description")
    if($descriptionSudahAda){
        // ternyata sudah ada
    } else {
        // tabel belum memiliki column description
        // buat column description dengan tipe string
        $table->string("description");
    }

    $table->timestamps();
});
```

**Penjelasan kode:** Perhatikan kode \$table->hasColumn("description"), kode ini akan menghasilkan Boolean, yaitu true jika tabel products sudah memiliki column description dan akan menghasilkan false jika tabel products belum memiliki column description.

## Memilih koneksi

Ketika menggunakan schema builder, kita menggunakan koneksi database global yang telah di setting pada file .env("DB\_CONNECTION"). Untuk mengganti koneksi yang akan digunakan pada file Migration, kita gunakan method connection()

```
Schema::connection('pgsql')->create('products', function (Blueprint $table)
{
    $table->increments('id');
});
```

**Penjelasan kode:** Sebelum membuat tabel products, kita memilih koneksi yang akan digunakan yaitu pgsql, koneksi ini harus tersedia pada konfigurasi config/database.php. Dengan begitu sekarang, tabel products akan dibuat pada database postgresql, bukan pada koneksi mysql.

## Mengubah settingan tabel

Kita juga dapat mengubah perilaku dari database melalui Schema Builder seperti ini:

Operasi	Penjelasan
\$table->engine = 'InnoDB';	Tentukan storage engine yang akan dipakai (MySQL)
\$table->charset = 'utf8';	Tentukan charset yang akan dipakai (MySQL).
\$table->collation = 'utf8';	Tentukan collation yang akan dipakai (MySQL).
\$table->temporary();	Buat tabel sementara (kecuali SQL Server).

## Mengubah nama column

Untuk mengubah nama tabel kita gunakan method renameColumn().

```
public function up(){
    Schema::table("products", function(Blueprint $table){
        $table->renameColumn("nama", "nama_baru");
    });
}
```

## Menghapus column

Untuk mengubah column kita gunakan method dropColumn()

```
public function up(){
    Schema::table("products", function(Blueprint $table){
        $table->dropColumn("nama_column");
    });
}
```

## Menghapus tabel

Sementara itu, untuk menghapus tabel kita bisa menggunakan method dari Schema Builder yaitu drop() dan dropIfExists(), ingat kita memakai fungsi ini pada method down() dari file class Migration kita

```
//File 2018_03_19_074052_create_products_table.php
public function down()
{
    // Drop tabel jika tabel products ditemukan
    Schema::dropIfExists('products');
}
```

## Operasi untuk Column

Sebelumnya kita telah belajar untuk membuat fields untuk tabel dengan method increments("nama\_column") dan timestamps() untuk membuat field created\_at dan updated\_at. Tentu kita membutuhkan lebih banyak method untuk memanipulasi column dari tabel kita. Kabar baiknya Schema Builder telah menyediakan banyak method yang bisa kita pakai, antara lain:

Perintah / Method	Penjelasan
\$table->bigIncrements('id');	Buat column bertipe auto-increment UNSIGNED BIGINT (primary key) dengan nama "id"
\$table->bigInteger('votes');	Buat bertipe BIGINT dengan nama "votes"
\$table->binary('data');	Buat column bertipe BLOB dengan nama "data".
\$table->boolean('confirmed');	Buat column bertipe BOOLEAN dengan nama "confirmed".
\$table->char('name', 100);	Buat column bertipe CHAR dengan panjang (length) 100. Parameter kedua yaitu length bersifat opsional.
\$table->date('created_at');	Buat column bertipe DATE dengan nama "created_at".
\$table->dateTime('created_at');	Buat column bertipe DATETIME dengan nama "created_at".
\$table->dateTimeTz('created_at');	Buat column bertipe DATETIME (dengan timezone) bernama "created_at"
\$table->decimal('amount', 8, 2);	Buat column bertipe DECIMAL dengan nama "amount". Parameter kedua merupakan total digits (presisi), dan parameter ketiga adalah decimal digits (decimal digit).
\$table->double('amount', 8, 2);	Buat column bertipe DOUBLE dengan presisi 8 dan decimal digits 2 bernama "amount".
\$table->enum('level', ['easy', 'hard']);	Buat column bertipe ENUM dengan nama "level" dan dua opsi yaitu "easy" dan "hard".

Perintah / Method	Penjelasan
\$table->float('amount', 8, 2);	Buat column bertipe FLOAT dengan presisi 8 dan decimal digits 2.
\$table->geometry('positions');	Buat column bertipe setara dengan GEOMETRY bernama "positions"
\$table->geometryCollection('positions');	Buat column bertipe setara dengan GEOMETRYCOLLECTION bernama
\$table->increments('id');	Buat column bertipe setara dengan UNSIGNED INTEGER (primary key)
\$table->integer('votes');	Buat column bertipe setara INTEGER bernama "votes"
\$table->ipAddress('visitor');	Buat column bertipe seperti alamat IP bernama visitor.
\$table->json('options');	Buat column bertipe seperti JSON bernama "options".
\$table->jsonb('options');	Buat column bertipe seperti JSONB bernama "options".
\$table->lineString('positions');	Buat column bertipe seperti LINESTRING bernama "positions".
\$table->longText('description');	Buat column bertipe seperti / setara dengan LONGTEXT bernama
\$table->macAddress('device');	Buat column bertipe seperti / setara dengan MAC Address
\$table->mediumIncrements('id');	Buat column bertipe setara UNSIGNED MEDIUMINT (primary key)
\$table->mediumInteger('votes');	Buat column bertipe setara dengan MEDIUMINT bernama "votes".
\$table->mediumText('description');	Buat column bertipe setara dengan MEDIUMTEXT bernama
\$table->morphs('taggable');	Menyisipkan column "taggable_id" dengan tipe setara UNSIGNED
\$table->multiLineString('positions');	Buat column bertipe MULTILINESTRING bernama "positions".
\$table->multiPoint('positions');	Buat column bertipe setara MULTIPOINT bernama "positions".
\$table->multiPolygon('positions');	Buat column bertipe MULTIPOLYGON bernama "positions".
\$table->nullableMorphs('taggable');	Sisipkan attribute nullable untuk column morphs().
\$table->nullableTimestamps();	Nama lain (alias) untuk timestamps().
\$table->point('position');	Buat column bertipe setara POINT dengan nama "position"
\$table->polygon('positions');	Buat column bertipe seperti POLYGON bernama "positions".
\$table->rememberToken();	Sisipkan column nullable bertipe VARCHAR(100) dengan nama
\$table->smallIncrements('id');	Buat column bertipe setara UNSIGNED SMALLINT (primary key)

Perintah / Method	Penjelasan
\$table->smallInteger('votes');	Buat column bertipe SMALLINT bernama "votes".
\$table->softDeletes();	Tambahkan column "deleted_at" bertipe TIMESTAMP dan memiliki
\$table->softDeletesTz();	Sama seperti softDeletes() tetapi dengan timezone.
\$table->string('name', 100);	Buat column bertipe setara VARCHAR dengan panjang (length)
\$table->text('description');	Buat column bertipe TEXT dengan nama "description".
\$table->time('sunrise');	Buat column bertipe TIME bernama "sunrise".
\$table->timeTz('sunrise');	Buat column bertipe TIME (dengan timezone) bernama "sunrise".
\$table->timestamp('added_on');	Buat column bertipe setara TIMESTAMP bernama "added_on".
\$table->timestampTz('added_on');	Buat column bertipe setara TIMESTAMP (dengan timezone)
\$table->timestamps();	Buat column untuk timestamps, yaitu column "created_at" dan
\$table->timestampsTz();	Sama seperti timestamps() tetapi menyertakan timezone
\$table->tinyIncrements('id');	Buat column bertipe setara UNSIGNED TINYINT (primary key)
\$table->tinyInteger('votes');	Buat column bertipe setara TINYINT bernama "votes".
\$table->unsignedBigInteger('votes');	Buat column bertipe setara UNSIGNED BIGINT dengan nama "votes"
\$table->unsignedDecimal('amount', 8, 2);	Buat column bertipe UNSIGNED DECIMAL dengan presisi 8 dan 2;
\$table->unsignedInteger('votes');	Buat column bertipe setara UNSIGNED INTEGER bernama "votes".
\$table->unsignedMediumInteger('votes');	Buat column bertipe setara UNSIGNED MEDIUMINT bernama "votes".
\$table->unsignedSmallInteger('votes');	Buat column bertipe setara UNSIGNED SMALLINT bernama "votes".
\$table->unsignedTinyInteger('votes');	Buat column bertipe setara UNSIGNED TINYINT bernama "votes".
\$table->uuid('id');	Buat column bertipe setara UUID dengan nama "id"
\$table->year('birth_year');	Buat column bertipe setara YEAR dengan nama "birth_year".

## Column Modifiers

Selain kita bisa menentukan tipe data dari column, dengan schema builder kita juga bisa mengubah / menambah perilaku dari column tersebut. Misalnya, membuat column menjadi nullable, menambahkan column setelah column tertentu. Kemampuan untuk melakukan hal itu disebut dengan modifier, apa saja column modifier yang tersedia pada Schema Builder, mari kita lihat:

Modifier	Penjelasan
->after('email')	Letakkan column yang akan dibuat setelah column "email"
->autoIncrement()	Set column bertipe INTEGER agar menjadi auto-increment (primary key)
->charset('utf8')	Tentukan charset utf8 untuk column (MySQL).
->collation('utf8_unicode_ci')	Tentukan collation utf8_unicode_ci untuk sebuah column (MySQL / SQL Serv
->comment('my comment')	Tambah komentar untuk column (MySQL)
->default(\$value)	Tentukan nilai default untuk column.
->first()	Letakkan column sebagai urutan pertama dalam
->nullable(\$value = true)	Izinkan nilai NULL untuk dimasukan ke column
->storedAs(\$expression)	Buat stored generated column (MySQL)
->unsigned()	Jadikan column INTEGER sebagai UNSIGNED (MySQL)
->useCurrent()	Jadikan column bertipe TIMESTAMP untuk menggunakan CURRENT_TIMESTAMP sebagai nilai default.
->virtualAs(\$expression)	Buat virtual generated column (MySQL)

Demikian beberapa column modifier yang tersedia, cara pakainya adalah seperti ini. Misalkan kita mau membuat column email, kita ingin meletakkannya setelah column password, dan mau kita beri komentar. Maka pada migration kita tuliskan seperti ini File migration

```
public function up()
{
    Schema::create("users", function(Blueprint $table){
        $table->increments();

        // begini cara pakai modifier
        $table->string("email", 255)->after("password")-
>comment("Email pengguna utama pengguna");
    })
}
```

## Mengubah Column Attribute

Untuk mengubah atribut kolom / field, terlebih dahulu kita membutuhkan package **doctrine/dbal**

Install dengan perintah berikut di aplikasi Laravel kita:

```
composer require doctrine/dbal
```

Lalu kita bisa menggunakan sebuah method bernama `change()`. Contohnya adalah kita ingin mengubah column "name" yang tadinya tidak menerima NULL / tidak boleh NULL ingin kita ubah agar menjadi nullable.

```
Schema::table('users', function (Blueprint $table) {
    $table->string('name', 50)->nullable()->change();
});
```

## Seeding

Apa itu seeding? Seeding jika diartikan secara harfiah bermakna memberi benih. Dalam konteks pengembangan aplikasi, Seeding adalah memberikan data awal ke database. Hal ini biasanya dilakukan saat pengembangan terutama jika kita ingin menguji apakah fitur tertentu telah berjalan sesuai ekspektasi menggunakan live data.

Kita bisa saja memberikan data awal secara manual dengan melakukan insert melalui DBMS, akan tetapi bayangkan jika kita akan menguji 100 data atau lebih, tentu akan sangat membosankan dan memakan waktu jika kita lakukan Seeding secara manual. Cara yang lebih pintar adalah dengan memanfaatkan fitur Seeding di Laravel.

### Membuat seeder

Pertama, kita buat terlebih dahulu file seeder untuk tabel tertentu. Misalnya, kita akan menguji fitur product list, itu berarti kita ingin menyiapkan data products terlebih dahulu ke database. Data ini hanyalah data dummy. Untuk membuat file seeder kita jalankan perintah berikut

```
php artisan make:seeder ProductTableSeeder
```

Setelah kita jalankan perintah di atas, maka sebuah file akan dibuat yaitu `app/database/seeds/ProductTableSeeder.php` dengan isi seperti ini:

File `app/database/seeds/ProductTableSeeder.php`

```
<?php

use Illuminate\Database\Seeder;

class ProductTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
    }
}
```

```

    }
}
```

**Penjelasan kode:** Kode di atas merupakan class seeder dengan nama **ProductTableSeeder**. Class tersebut memiliki satu method bernama **run()**. Pada method inilah kita akan menulis kode untuk mengisi data dummy untuk table products. Kita melakukan insert menggunakan **Query Builder**.

Selanjutnya, kita akan coba insert data ke tabel **products**. Maka kita tuliskan Query Builder kita pada method run() seperti ini:

```

<?php

use Illuminate\Database\Seeder;

class ProductTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Product::create([
            "name" => "Produk A",
            "description" => "Deskripsi Produk A created by model"
        ]);
    }
}
```

**Penjelasan kode:** Kita menginsert satu row ke table products. Data yang kita insert adalah name dan description. Untuk melakukan insert kita gunakan Model Product karena kita akan menginsert ke tabel products, ingat model Product yang telah kita buat sebelumnya pada bahasan Model. Dengan method **create()** kita menginsert sebuah row.

## Menjalankan Seeder

Setelah itu, kita akan eksekusi Seeder yang baru saja kita buat. Untuk melakukannya kita cukup jalankan perintah

```
php artisan db:seed --class=ProductTableSeeder
```

Dengan perintah di atas kita menjalankan Seeding menggunakan class **ProductTableSeeder**. Setelah itu data akan tersedia di tabel products.

Sebelum menjalankan perintah di atas pastikan tabel **products** sudah memiliki field **description** ya. Kamu bisa membuatnya secara manual atau belajar menggunakan file migration seperti pada sub bab sebelumnya.

## Menjalankan Multiple Seeder Class

Misalnya kita memiliki lebih dari satu file seeder, `ProductTableSeeder`, `CategoryTableSeeder`, `UserTableSeeder`. Maka kita bisa menjalankannya bersamaan tidak perlu satu per satu seperti sebelumnya.

Untuk melakukannya, pertama kita buka file `DatabaseSeeder.php` yang terletak di direktori `database/seeds`. Lalu jalankan `$this->call()` dan isi parameter dengan class table yang kita ingin eksekusi. Seperti ini:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);

        $this->call(ProductTableSeeder::class);

        $this->call(CategoryTableSeeder::class);
    }
}
```

Pastikan semua class Seeder yang akan dipanggil sudah kita buat ya. Lalu tinggal jalankan perintah berikut:

```
php artisan db:seed
```

Tanpa argument `--class` seperti pada cara sebelumnya.

Apabila ketika menjalankan perintah `db:seed` muncul error `class doesn't exist`. Jalankan terlebih dahulu perintah `composer dump-autoload` lalu jalankan lagi perintah `php artisan db:seed`

Selain menggunakan cara `$this->call(UserTableSeeder::class)`, kita juga bisa menggunakan sintak alternatif seperti ini

```
$this->call("UserTableSeeder");
```

Sintaks ini akan sama-sama memanggil class `UserTableSeeder`.

## Latihan Praktik

Untuk melanjutkan ke bab berikutnya, kita perlu untuk mempersiapkan data-data awal di database. Oleh karenanya, mari kita lakukan persiapan tersebut di sini. Inilah yang akan kita kerjakan:

1. Membuat migration untuk tabel categories dengan field-field berikut:
  1. `id`: Integer {PK} {AutoIncrement}
  2. `name`: String
  3. `created_at`: Timestamps
  4. `updated_at`: Timestamps
2. Membuat 10 untuk table categories di atas.

Langkah-langkahnya adalah sebagai berikut. Pertama kamu masuk ke `workspace` masing-masing dulu ya di terminal (ingat caranya di bab instalasi dan konfigurasi).

### A. Membuat Tabel dan Strukturnya dengan Migration

1. Buat file migration untuk tabel categories

```
php artisan make:migration create_categories_table
```

2. Buka file yang baru digenerate (nama file berakhiran `create_categories_table.php`) di direktori `database/migrations` lalu pastikan kode nya menjadi seperti ini:

```
<?php

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

class CreateCategoriesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            // buat column id -> integer, autoIncrement dan primary
            $table->increments('id');

            // buat column name -> string
            $table->string("name");
        });
    }

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

```
// buat column created_at dan updated_at
$table->timestamps();
});

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    // hapus jika tabel categories exist.
    // untuk migrate:rollback
    Schema::dropIfExists('categories');
}
}
```

### 3. Jalankan migration

```
php artisan migrate
```

Dengan demikian maka kamu bisa cek di database **toko-online** kini sudah memiliki tabel **categories** dengan field-field seperti yang kita definisikan pada file migration kita.

Selanjutnya kita akan mengisi 10 data awal untuk tabel categories.

#### B. Mengisi data awal dengan Seeder

##### 1. Buat file seeder untuk tabel categories

```
php artisan make:seeder CategoryTableSeeder
```

2. Sebelum kita bisa insert data di Seeder kita perlu model category. Buat dulu Model Category, pembahasan lebih lanjut akan kita pelajari di bab setelah ini.

```
php artisan make:model Category
```

3. Lalu buka file **CategoryTableSeeder.php** pada direktori **database/seeds**, dan isi kode berikut ini:

**Perhatikan** sebelum bisa menggunakan **new Category** kita menambahkan baris **use App\Category;** di bagian atas file untuk mengimport model **Category** yang terletak di **app/Category.php**.

```
<?php

use Illuminate\Database\Seeder;
use App\Category;

class CategoryTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $kategori1 = new Category;
        $kategori1->name = "Sepatu";
        $kategori1->save();

        $kategori2 = new Category;
        $kategori2->name = "Tas";
        $kategori2->save();

        $kategori3 = new Category;
        $kategori3->name = "Elektronik";
        $kategori3->save();

        $kategori4 = new Category;
        $kategori4->name = "Kemeja";
        $kategori4->save();

        $kategori5 = new Category;
        $kategori5->name = "Celana";
        $kategori5->save();

        $kategori6 = new Category;
        $kategori6->name = "Buku";
        $kategori6->save();

        $kategori7 = new Category;
        $kategori7->name = "Kosmetik";
        $kategori7->save();

        $kategori8 = new Category;
        $kategori8->name = "Komputer";
        $kategori8->save();

        $kategori9 = new Category;
        $kategori9->name = "Sandal";
        $kategori9->save();

        $kategori10 = new Category;
        $kategori10->name = "Furniture";
        $kategori10->save();
    }
}
```

```
}
```

- Setelah itu eksekusi seed di atas dengan perintah ini:

```
php artisan db:seed
```

- Selesai, silahkan buka phpmyadmin lalu buka table `categories` di database `toko-online` seharusnya kamu akan menemukan data yang baru saja kita insert di Seeder. Seperti ini misalnya:

+ Options

			<a href="#">id</a>	<a href="#">name</a>	<a href="#">created_at</a>	<a href="#">updated_at</a>
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	1	Sepatu	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	2	Tas	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	3	Elektronik	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	4	Kemeja	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	5	Celana	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	6	Buku	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	7	Kosmetik	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	8	Komputer	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	9	Sandal	2018-07-03 06:32:47	2018-07-03 06:32:47	
<input type="checkbox"/>	<a href="#"></a> Edit <a href="#"></a> Copy <a href="#"></a> Delete	10	Furniture	2018-07-03 06:32:47	2018-07-03 06:32:47	

Check all    With selected:  Edit  Copy  Delete  Export

## Kesimpulan

Dalam bekerja dengan database kita bisa memilih menggunakan code first atau database first, apapun pilihanmu keduanya bisa dilakukan di Laravel. Kita telah belajar fitur-fitur dari `Schema Builder` untuk memanipulasi struktur tabel kita melalui fitur `Migration`.

Kemudian kita juga belajar bagaimana melakukan mengisi data awal ke database menggunakan `QueryBuilder` yang kita tulis pada class-class Seeder.

Ini berarti kita telah siap untuk belajar materi selanjutnya yaitu Model dan Eloquent.

# Model & Eloquent

## Model

Sebelum memulai bab ini, pastikan kamu sudah melakukan latihan praktik pada bab database. Jika belum, silahkan lakukan sekarang lalu kembali lagi.

Pada latihan praktik di bab sebelumnya kita belajar membuat tabel baru dan strukturnya dengan migration. Lalu kita mengisi 10 data awal ke tabel `categories`. Pada latihan tersebut kita telah belajar membuat model, dan di bab ini kita akan melihat lebih jauh tentang model dan konfigurasi apa saja yang bisa kita ubah.

Kami berharap kamu masih ingat bagaimana mudahnya membuat model dengan perintah `php artisan make:model Nama`. Misalnya dengan perintah `php artisan make:model Category` dan lalu akan tergenerate sebuah file model pada path `app/Category.php`. Coba kamu buka file tersebut, maka kamu akan melihat sebuah file Model default. Seperti ini:

```
<?php  
  
namespace App;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Category extends Model  
{  
    //  
}
```

Dengan model `Category` kita telah bisa menginsert 10 data awal. Dan hal itu kita lakukan di file `CategoryTableSeeder`, salah satu penggunaannya untuk membuat data kategori baru adalah seperti ini:

```
$kategori = new Category;  
$kategori->name = "Sepatu";  
$kategori->save();
```

**Penjelasan Kode:** Membuat model `Category` dengan perintah `new Category` disimpan sebagai variabel `$kategori` lalu kita ubah field `name` menjadi "Sepatu" dan kita simpan ke database dengan perintah `$kategori->save()`

Sampai di situ kamu sekarang paham apa kegunaan model, bukan? Mulai semakin terlihat jelas?

Selain di file Seeder seperti yang telah kita lakukan di latihan praktik, kita juga akan banyak menggunakan Model pada controller. Yup, betul ini sejalan dengan visualisasi arsitektur MVC yang telah kita lihat di bab arsitektur Laravel.

## Konvensi Model

Konvensi penamaan model adalah hufuf pertama menggunakan kapital misalnya, **Category**, **Product**, **User** dan jika terdiri dari dua kata maka ditulis seperti ini **ProductList**. Lebih baik jika sebuah model cukup menggunakan satu kata.

Konvensi berikutnya adalah setiap nama tabel akan merepresentasikan nama tabel di database dengan nama jamak. Misalnya model **Category** secara otomatis merepresentasikan tabel **categories**, model **User** otomatis merepresentasikan tabel **users**. Perhatikan ya **bentuk jamak** nama tabel di database.

Bagaimana jika kita ingin menggunakan nama tabel yang berbeda dari kesepakatan? Setelah ini kita akan membahas properti apa saja yang bisa kita gunakan untuk mengubah model kita.

## Model Attribute / properti

### Mengganti Tabel pada Model

Kita bisa melakukannya dengan menambahkan properti **protected \$table** pada file model kita seperti berikut:

```
<?php  
  
namespace App;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Category extends Model  
{  
    protected $table = "custom_nama_table";  
}
```

Dengan definisi di atas, maka sekarang model **Category** merepresentasikan tabel **custom\_nama\_table** di database.

### Menggunakan Koneksi Selain Default pada Model

Koneksi untuk sebuah model secara default menggunakan settingan global di file konfigurasi database kita. Akan tetapi, misalnya untuk model tertentu kita ingin menggunakan koneksi selain default, maka kita gunakan properti **\$connection** pada model tersebut untuk menentukan koneksi mana yang ingin dipakai di model yang bersangkutan.

```
<?php  
  
namespace App;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Category extends Model  
{  
    protected $table = "custom_nama_table";
```

```
protected $connection = "mongo_stage_1";
}
```

Apabila settingan default koneksi di aplikasi laravel kita menggunakan settingan bernama "mysql\_production", maka dengan kode di atas, khusus untuk model Product akan menggunakan "mongo\_stage\_1" bukan "mysql\_production".

## Mengubah Primary Key Model

Ketika kita membuat model baru, maka secara default dianggap tabel dari model tersebut memiliki primary key dengan field `id`. Melanjutkan ilustrasi sebelumnya, bila kita ingin mengubah field primaryKey untuk model Product kita gunakan properti `$primaryKey` seperti ini

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    protected $table = "custom_nama_tabel";
    protected $primaryKey = "identifier";
}
```

Dengan kode di atas, maka sekarang model Product menggunakan field `identifier` pada tabel `custom_nama_tabel` sebagai `primaryKey`.

## Mass-assignment

Apa itu mass-assignment? dan kenapa penting? Mass-assignment merupakan cara untuk menginsert data melalui model dengan sekali perintah. Misalnya seperti ini:

```
$kategori = Category::create($request->all());
```

Perhatikan kita menginsert langsung ke tabel `categories` menggunakan model `Category` method `create` dan mengisikan seluruh data yang kita ambil dari `$request->all()` (telah kita pelajari di bab route & controller)

Bukankah cara ini berbeda dengan yang telah kita praktikan pada latihan sebelumnya yaitu dengan kode ini

```
$kategori = new Category;
$kategori->name = "Sepatu";
$kategori->save();
```

Cara yang ini memang agak bertele-tele karena kita mengassign nilai ke setiap field satu per satu, bayangkan jika ada banyak field? Bukankah lebih cepat dengan mass-assignment? Yaitu menggunakan `NamaModel::create($data)`

Ya, memang lebih cepat dengan `mass-assignment`, akan tetapi kita harus mempertimbangkan faktor keamanan. Kenapa begitu? karena dengan mass-assignment kita membuka celah bagi user untuk mengakali aplikasi kita.

Misalnya begini, kita punya model `User` yang merepresentasikan tabel `users` di database. Tabel `users` memiliki field antara lain `name`, `email`, `address` dan `role`. Field `role` akan diisi dengan `customer` atau `administrator`. Bayangkan jika kita menggunakan `mass-assignment`, maka orang yang tidak bertanggung jawab bisa menginject ke form untuk melakukan update terhadap akun dirinya agar menjadi admin, dia cukup menambahkan field `role` pada form diisi dengan nilai `administrator`.

Di kode kita, kita menggunakan mass-assignment seperti ini:

```
$user = User::create($request->all());
```

Siapa yang tahu ternyata `$request->all()` berisi data tambahan yaitu `role` bernilai `administrator`, padahal field tersebut seharusnya defaultnya adalah `customer`.

Nah untuk mencegah hal tersebut kita harus memberikan nilai ke field `role` pada kode kita seperti ini:

```
$user->role = "CUSTOMER";  
$user->save();
```

Cara ini akan mencegah kerentanan yang telah dijelaskan tadi. Oleh karena itu secara default Laravel model memproteksi seluruh field dari operasi mass-assignment. Akan tetapi mass-assignment ini sebenarnya sangat membantu mempercepat kita sebagai developer tanpa harus satu per satu memberi nilai ke tiap-tiap field. Yup, mass-assignment boleh dilakukan, terutama untuk data-data yang bukan bersifat sensitif dan krusial.

Dan Laravel pun mengizinkan hal ini, maka dari itu kita bisa melakukan konfigurasi pada model field mana saja yang boleh untuk dilakukan mass-assignment dan field mana saja yang diproteksi dari operasi tersebut.

### Mengizinkan Operasi Mass-assignment pada Properti Model

Properti `$fillable` kita gunakan untuk mengizinkan fields-fields apa saja yang diperbolehkan untuk operasi mass assignment. Properti ini harus diassignkan dengan nilai array yang berisi nama field yang diperbolehkan untuk mass assignment. Misalnya dari model `User` tadi kita ingin mengizinkan properti `email`, `name` dan `address` tapi membiarkan field `role` terlindungi, maka kita tambahkan properti `$fillable` di model `User` seperti ini:

```
<?php  
  
namespace App;
```

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

class User extends Model
{
    protected $table = "custom_nama_table";
    protected $primaryKey = "identifier";
    protected $fillable = ["name", "email", "address"];
}
```

Dengan kode di atas, kini fields `name`, `email`, dan `address` diperbolehkan untuk operasi mass-assignment, sementara field `role` tidak diizinkan.

### Memproteksi Properti Model dari Operasi Mass-assignment

properti `$guards` memiliki fungsi sebaliknya dengan `$fillable`, yaitu untuk mendeklarasikan field apa saja yang tidak diperbolehkan untuk mass-assignment. Contohnya adalah seperti ini

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    protected $table = "custom_nama_table";
    protected $primaryKey = "identifier";
    protected $guards = ["role"];
}
```

Dengan kode di atas, maka kini kita mengizinkan semua field pada model User untuk operasi mass-assignment, kecuali yang dilindungi (*guarded*) yaitu field `role`.

### Mencatat Kapan Data Model Dibuat dan Diupdate

Secara default model mengasumsikan tabel kita memiliki field `created_at` dan `updated_at`. Kedua field tersebut merupakan penanda kapan suatu record tersebut dibuat dan kapan terakhir diupdate. Keduanya akan otomatis diinsert atau diupdate ketika kita memanipulasi record menggunakan Eloquent.

Jika tabel kita tidak memiliki kedua field tersebut akan muncul error. Untuk menghindari error bila kita memang tidak memerlukan field `created_at` dan `updated_at`, maka kita set properti `$timestamps` ke `false` seperti ini:

```
<?php

namespace App;

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

```
class Category extends Model
{
    //
    protected $timestamps = false;
}
```

Selanjutnya, misalnya kita ingin menggunakan timestamps tapi menggunakan field lain kita bisa ubah dengan const CREATED\_AT dan UPDATED\_AT seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    const CREATED_AT = "tanggal_dibuat";
    const UPDATED_AT = "tanggal_diupdate";
}
```

## Eloquent

Ketika kita membuat model di Laravel, sebetulnya kita membuat **Eloquent Model**. Atau dengan kata lain semua model di Laravel merupakan turunan dari Eloquent model. Untuk mengeceknya silahkan buka salah satu file model yang pernah kita buat sebelumnya, yaitu app/Category.php

File app/Category.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{}
```

**Penjelasan kode:** Lihat kita menulis model Product dengan mengekstend Model, dari mana Model ini? Lihat baris di atasnya, yaitu **use Illuminate\Database\Eloquent\Model**. Kini kita tahu bahwa seluruh model di Laravel merupakan turunan dari **Illuminate\Database\Eloquent\Model** atau kita sebut **Eloquent Model**. Nah, lalu apa dampaknya? Dampaknya adalah kini setiap model memiliki method-method yang akan kita bahas pada bab Eloquent ini. Apa saja itu? Ada banyak, mari kita bahas satu per satu karena method ini akan sering kita pakai dalam pengembangan aplikasi Laravel.

## Query Record

### Menampilkan seluruh record

Method `all()` digunakan untuk mengambil semua data model dari tabel database yang bersangkutan. Misalnya model Product yang merepresentasikan tabel products, jika kita panggil method all seperti ini `Product::all()`, maka kita mengambil seluruh row pada tabel products.

Hasil dari method `all()` adalah berupa data bertipe Array, dan jika tabel masih kosong maka array kosong yang kita dapatkan.

### Mencari record berdasarkan primaryKey

Method `find()` digunakan untuk mencari row berdasarkan primaryKey, misalnya seperti ini:

```
Product::find(1)
```

**Penjelasan kode:** Kode di atas mencari row di tabel products dengan id 1. Hasil dari method `find()` adalah single record, yang berarti sebuah instance dari Model, bukan Array. Dengan begitu jangan coba-coba melakukan perulangan (looping) ya karena akan error. Apabila tidak ada row ditemukan, maka null yang kita dapatkan.

### Mendapatkan record pertama saja

Selain method `find()`, kita juga bisa mencari single row dengan method `first()`, bedanya first tidak menerima parameter apapun. Biasanya digunakan setelah kita melakukan filtering dengan method `where()` seperti ini:

```
Product::where("status", "active")->first();
```

**Penjelasan kode:** Kode di atas mencari row di tabel products dengan kriteria column status bernilai "active" dan hanya ambil satu row pertama. Hasil dari `first()` juga merupakan instance dari model, bukan Array.

### `findOrFail()`

Method `findOrFail()` bekerja mirip dengan `find()`, bedanya saat tidak ada row yang ditemukan maka akan otomatis melempar `Error Model Not Found` alias `404`, berbeda dengan `find()` yang mengembalikan nilai `NULL` jika row tidak ditemukan dengan id sesuai parameter.

```
Product::findOrFail(1);
```

**Penjelasan kode:** Kode ini akan mencari row di tabel products dengan id 1, apabila tidak ada row yang ditemukan dengan id 1 maka otomatis berikan error `404` ke user. Method ini kita gunakan jika kita tidak ingin mengecek secara manual apakah row ditemukan atau tidak.

## Aggregates

Method-method aggregates adalah method yang kita gunakan untuk menghitung beberapa rows dengan criteria tertentu dan mengembalikan sebuah nilai. Berikut daftar aggregates method yang tersedia.

### count()

Menghitung jumlah row.

### max()

Dapatkan nilai tertinggi dari kumpulan row berdasarkan field tertentu.

```
Product::all()->max("price");
```

**Penjelasan kode:** Dapatkan nilai price tertinggi dari semua row di tabel products.

### min()

Dapatkan nilai terendah dari kumpulan row berdasarkan field tertentu.

```
Product::all()->min("price");
```

**Penjelasan kode:** Dapatkan nilai price terendah dari semua row di tabel products.

### sum()

Jumlahkan nilai field tertentu.

```
Product::all()->sum("stock");
```

**Penjelasan kode:** Jumlahkan semua nilai stock dari tabel products, dengan begitu kita bisa mendapatkan keseluruhan stock dari product yang ada di tabel.

### avg()

Dapatkan nilai rerata berdasarkan field tertentu pada sebuah tabel.

```
Product::all()->avg("price");
```

**Penjelasan kode:** Dapatkan nilai rerata harga product dari seluruh data di tabel products.

Kita telah mempelajari beberapa method yang ada di Eloquent, akan tetapi dari semua yang kita bahas sebelumnya belum ada method untuk mengubah database. Kita telah belajar bagaimana membaca data (*query*) dari database, dan sekarang kita akan mulai belajar untuk memanipulasi data ke database. Operasi pertama yang kita pelajari adalah insert record baru.

Ada dua cara untuk melakukan insert record ke dalam database yaitu dengan method `save()` dan `create()`

### `save()`

Dengan method `save()` kita bisa menginsert record baru, caranya adalah kita buat model baru terlebih dahulu baru kita panggil method `save()`. Contoh kodennya seperti ini

```
$product_baru = new Product;
$product_baru->name = "Sepatu Coder";
$product_baru->description = "Deskripsi produk sepatu coder";
$product_baru->save();
```

**Penjelasan kode:** Untuk menggunakan method `save()` pertama kita buat terlebih dahulu model baru, karena kita akan menginsert ke tabel `products`, maka kita buat model `Product` dengan `new Product` dan kita simpan sebagai variabel `$product_baru`.

Setelah itu kita isi properti yang ada di model `products`, properti ini mewakili column yang ada di tabel `products`, yaitu `name` dan `description`. Kita mengisinya dengan `$product_baru->name = "Sepatu Coder"; $product_baru->description = "Deskripsi sepatu coder";`

Kita sudah membuat instance dari model `Product`, tetapi data yang kita tuliskan belum tersimpan ke database kita. Untuk menyimpannya kita panggil method `save()` di instance dari model `Product` yang baru kita buat tadi seperti ini:

```
$product_baru->save()
```

Dengan begini maka sekarang tersimpan data baru di tabel `products` dengan data `name` “Sepatu Coder” dan `description` “Deskripsi sepatu coder”.

Method `save()` mengembalikan instance model yang berhasil di simpan di database beserta dengan nilai field ID / primary key.

### `create()`

Cara lain untuk membuat record baru di database adalah dengan method `create()`. Cara penggunaannya sedikit berbeda dengan `save()`:

```
$product_tas = Product::create([
    "name" => "Tas Selempang Army",
```

```
"description" => "Deskripsi tas selempang army"
```

```
]);
```

**Penjelasan kode:** Insert ke tabel products dengan method create pada model Product, method `create()` membutuhkan sebuah parameter bertipe array yang berisi data yang akan diinsert ke database.

**PENTING!** Apa yang dilakukan oleh method `create()` adalah menggunakan cara mass-assignment, method create bisa kita pakai, kita harus definisikan dulu properti apa saja dari model `Product` yang boleh untuk operasi mass-assignment. Kita telah mempelajarinya pada sub bab mass-assignment.

## Update Record

### `save()`

Method `save()` selain bisa kita gunakan untuk menginsert record baru ke database juga bisa kita gunakan untuk mengupdate record tertentu. Ketika menyimpan kita membuat model baru dengan `new NamaModel` misalnya `new Product`, sedangkan untuk mengupdate data, pertama kita *query* dulu record yang akan kita update.

```
// select dari tabel products dengan ID 20
$produk_untuk_diupdate = Product::findOrFail(20);

// ubah nilai description dari record yang kita dapatkan
$produk_untuk_diupdate->description = "Deskripsi kita ubah di sini";

// simpan perubahan ke database untuk product dengan ID 20 tadi
$product_untuk_diupdate->save();
```

**Penjelasan kode:** Mengupdate nilai column description untuk produk dengan ID 20 ke database.

### `update()`

Selain method `save()`, kita juga bisa mengupdate record dengan method `update()`; Method ini fungsinya mirip dengan `create()` untuk membuat data. Perbedaannya tentu saja, method update untuk mengupdate bukan membuat / menginsert record baru.

```
App\Product::where("status", "active")->update(["status" => "inactive"]);
```

**Penjelasan kode:** Ubah status menjadi “inactive” dari semua data di tabel products yang saat ini memiliki status bernilai “active”

Method `update()` sama seperti method `create()` membutuhkan sebuah parameter bertipe array yang berisi data untuk diupdate ke database. Selain itu, untuk menggunakan method ini kita juga perlu mendefinisikan field / column apa saja yang boleh untuk operasi mass-assignment.

## Delete Record

Ada dua cara untuk menghapus record di database kita ketika menggunakan Eloquent, yaitu dengan method `delete()` dan `destroy()`

## Menghapus data model

Method ini kita panggil dari instance model tertentu. Misalnya pertama kita query terlebih dahulu sebuah model berdasarkan primary key seperti ini

```
$product = Product::findOrFail(21);
```

Kita cari product dengan id 21 dan simpan sebagai variabel \$product. Lalu kita hapus model tersebut dengan memanggil method delete seperti ini

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

## Menghapus satu atau lebih data model berdasarkan primaryKey sekaligus

Destroy adalah cara lain untuk menghapus record / model. Method `destroy()` membutuhkan parameter yang bisa berupa string / integer mewakili nilai primary key atau Array yang berisi satu atau lebih nilai primary key seperti ini

```
// hapus product yang memiliki ID 22
App\Product::destroy(22);

// hapus product dengan ID 23,25,31
App\Product::destroy([23,25,21]);

// cara ini bekerja persis seperti cara sebelumnya
App\Product::destroy(23,25,21);
```

**Perhatikan!** mungkin kamu bertanya-tanya, kenapa sekarang kita menggunakan `App\Product`, padahal sebelumnya kita cuma pakai `Product`. Apa bedanya? Tenang, keduanya merujuk ke model yang sama yaitu model `Product`. Apabila ingin menggunakan `Product` saja, pastikan sudah import model tersebut di bagian atas file model `Product` dengan kode ini `use App\Product;`

## Soft Deletes

Soft delete merupakan sebuah cara untuk menghapus sebuah record tanpa langsung menghapusnya dari database. Dengan kata lain sebetulnya data yang disoft delete masih ada di record tabel, akan tetapi column `deleted_at` di record tertentu akan diisi sebuah nilai yaitu timestamp kapan model tersebut dihapus.

Jika column `deleted_at` tidak null alias ada isiannya, maka model tersebut statusnya adalah sedang dihapus.

Jika kita ingin menggunakan fitur soft deletes, pertama kita harus buat sebuah column di tabel kita bernama `deleted_at`, jika kita menggunakan file Migration kita bisa membuat field tersebut dengan memanggil method `softDeletes()` dari Schema Builder (lihat kembali bahasan Migration).

Kemudian kita harus menggunakan trait `Illuminate\Database\Eloquent\SoftDeletes` pada model yang ingin kita aktifkan soft delete. Sehingga model kita akan tampak seperti ini

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

// PERHATIKAN INI!
use Illuminate\Database\Eloquent\SoftDeletes;

class Category extends Model
{
    // DAN INI
    use SoftDeletes;
}
```

Jika tanpa menggunakan soft delete, ketika kita memanggil `delete()` atau `destroy()` maka record yang kita pilih akan langsung dihapus dari tabel. Sedangkan jika menggunakan soft delete tidak seperti itu. Saat kita mengaktifkan soft delete maka ingat, kita memiliki column `deleted_at` pada tabel, column ini defaultnya bernilai NULL. Dan NULL berarti record tersebut dalam kondisi tidak dihapus, kemudian saat kita memanggil method `delete()` atau `destroy()` maka record yang kita pilih memiliki nilai pada column `deleted_at`, artinya column `deleted_at` tidak lagi bernilai NULL, sehingga kini record-record tersebut statusnya soft deleted (dihapus), tetapi tetap tersimpan di tabel. Jika kita query menggunakan eloquent, record-record ini tidak akan muncul karena dianggap telah dihapus.

Analogi lain untuk memudahkan kamu memahami soft delete adalah, setiap model yang dihapus masuk terlebih dahulu ke tong sampah, artinya masih kita bisa `restore` kembali jika kita perlukan dengan mengambilnya dari tong sampah. Untuk membuagnya secara permanen maka kita harus keluarkan dari tong sampah dan membakarnya. Begitu.

Dengan penjelasan di atas, kamu mungkin bertanya, berarti model yang sudah disoft delete bisa direstore kembali. Yup, betul sekali, itulah gunanya soft delete, kita bisa merestorenya kembali jika ternyata kita masih membutuhkan model yang sudah disoft delete.

Untuk memahami lebih lanjut apa saja yang bisa kita lakukan dengan fitur soft delete di Laravel sebaiknya kita bahas satu per satu method apa saja yang tersedia

### `trashed()`

Method `trashed()` digunakan untuk mengecek apakah sebuah model dalam kondisi di tempat sampah alias soft deleted. Seperti ini:

```
$product = App\Product::findOrFail(23);
if($product->trashed()){
    // ya product sedang berada di tempat sampah / soft deleted
}
```

**Penjelasan kode:** Melakukan pengecekan apakah record di tabel products dengan ID 23 sedang di tong sampah. Method `trashed()` mengembalikan nilai Boolean, `true` jika model tersebut soft deleted dan `false` jika model tersebut dalam kondisi tidak disoft deleted.

#### restore()

Method `restore()` kita gunakan untuk mengembalikan model yang sebelumnya berada di tong sampah / soft deleted agar kembali aktif atau tidak soft deleted. Melanjutkan contoh kode sebelumnya, kita ingin merestore `product` dengan id tertentu apabila statusnya sekarang soft deleted, maka kita lakukan seperti ini:

```
// cari produk di tabel products dengan ID == 23
// jika gagal lempar error 404
$product = App\Product::findOrFail(23);

// jika ketemu dan statusnya soft deleted (trashed)
// restore
if($product->trashed()){
    $product->restore();
}
```

**Penjelasan kode:** Dapatkan model products dengan id 23, kemudian cek menggunakan method `trashed()` apakah sedang berada di tong sampah / soft deleted, jika iya maka restore model tersebut.

#### withTrashed()

Kami tadi menjelaskan bahwa ketika kita melakukan *query* terhadap model yang mengaktifkan soft delete, maka semua record yang berstatus soft deleted tidak ikut terquery. Tetapi terkadang kita ingin menquery keduanya, baik soft deleted atau tidak. Maka, untuk mengikutsertakan hasil dari model yang berstatus soft deleted kita gunakan method `withTrashed()` pada query kita, misalnya seperti ini:

```
$product = App\Product::where("owner", 12)->withTrashed()->get();
```

**Penjelasan kode:** Dapatkan record dari tabel products yang nilai column ownernya adalah 12, ikutsertakan juga record yang berstatus soft deleted.

#### onlyTrashed()

Berbeda dengan `withTrashed()`, `onlyTrashed()` justru mengembalikan record-record yang berstatus soft deleted saja. Tanpa menyertakan record yang belum disoft delete.

```
$product_trashed = App\Product::where("owner", 12)->onlyTrashed()->get();
```

### **forceDelete()**

**forceDelete()** kita gunakan untuk menghapus sebuah record secara permanen meskipun kita menggunakan softDelete pada model yang terkait.

## Data Pagination

Dengan menggunakan Eloquent untuk memanipulasi data, kita jadi lebih mudah untuk melakukan paging. Fitur pagination bawaan Laravel mendukung baik jika kita menggunakan Query Builder atau Eloquent. Ada dua metode yang bisa kita gunakan untuk melakuka pagination di laravel yaitu **simplePaginate()** dan **paginate()**.

### **simplePaginate()**

Method **simplePaginate()** memiliki sebuah parameter opsional yaitu berapa item yang ingin kita tampilkan per halamannya. Misalnya kita ingin menampilkan data dari tabel products per halaman 25 product, maka kita gunakan **simplePaginate(25)**.

File app\Http\Controllers\ProductController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function index()
    {
        return view("products.list",
        ["products"=>\App\Product::simplePaginate(25)]);
    }
}
```

### **paginate()**

Method **paginate()** bekerja seperti halnya **simplePaginate()** yaitu memiliki sebuah parameter opsional berupa jumlah item yang ingin kita tampilkan per halaman. Untuk menggunakannya cukup tuliskan **paginate(25)** misalnya.

File app\Http\Controllers\ProductController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProductController extends Controller
{
    public function index()
    {
        return view("products.list",
["products"=>\App\Product::paginate(25)]);
    }
}
```

**Perhatikan:** Method `simplePaginate()` dan `paginate()` apabila tidak diberikan parameter maka default data per halaman adalah 15.

### Perbedaan `simplePaginate()` dan `paginate()`

Perbedaan pertama antara simplePaginate dengan paginate adalah, simplePaginate kita gunakan jika kita hanya ingin menampilkan tombol “next” dan “prev” tanpa perlu menampilkan informasi pagination lainnya seperti halaman yang aktif, jumlah halaman, halaman terakhir, dll. Menggunakan simplePaginate akan lebih efisien untuk data yang besar.

Perbedaan kedua adalah, ketika kita memanggil `simplePaginate()` dan `paginate()` maka kita akan bisa mengakses beberapa method berkaitan dengan pagination di view kita, namun ada 2 method yang tidak tersedia di `simplePaginate()` tapi tersedia jika menggunakan `paginate()`. Apa saja method-method tersebut?

Method berkaitan dengan hasil Pagination	Deskripsi
count()	Dapatkan jumlah data set dari seluruh halaman
currentPage()	Dapatkan halaman ke berapa yang sedang aktif
firstItem()	Dapatkan item pertama dari seluruh data set
hasMorePages()	Cek apakah masih ada halaman selanjutnya
lastItem()	Dapatkan item terakhir dari seluruh data set
lastPage()	Tidak tersedia di <code>simplePaginate()</code> . Dapatkan halaman terakhir
nextPageUrl()	Dapatkan url halaman selanjutnya, misalnya
perPage()	Dapatkan berapa jumlah item per halaman.
previousPageUrl()	Dapatkan url halaman sebelumnya,

Method berkaitan dengan hasil Pagination	Deskripsi
total()	Tidak tersedia di simplePaginate(). Dapatkan total halaman dari pagination.
url(\$page)	Generate link ke halaman tertentu (\$page). Berguna untuk fitur lompat ke halaman.

### Menampilkan link pagination di view

Method `simplePaginate()` dan `paginate()` berfungsi untuk mengambil data dari database sesuai jumlah item per halaman yang kita inginkan. Tugas selanjutnya bagi developer adalah untuk menampilkan link / tombol pagination di view agar pengguna aplikasi dapat melakukan navigasi ke halaman tertentu.

Untuk menampilkan link / tombol pagination pada sebuah view tentu langkah pertama yang harus kita lakukan adalah dengan menggunakan method `simplePaginate()` / `paginate()` di controller yang menggunakan view tersebut.

Bila kita perhatikan contoh kode sebelumnya yaitu pada `ProductController.php`, kita melakukan paginate pada model Product, dan melemparkan ke view “products.list” yang berarti file view tersebut terletak pada `resources/views/products/list.blade.php`. Data tersebut dilempar sebagai variabel `$products`, lihat bagian `["products" => \App\Product::paginate(25)]`.

Berarti, kini pada file view tersedia variabel `$products` yang berisi 25 data products pertama jika misalnya ada 300 data di database. Maka kita bisa menampilkan products tersebut dengan Blade melalui looping. Selanjutnya, kita ingin menampilkan tombol pagination di bawah daftar products, maka kita gunakan sebuah method pagination yaitu `links()` seperti ini

File `resources/views/products/list.blade.php`

```
<ul>
@foreach($products as $p)
    <li>{{$p->name}}</li>
@endforeach
</ul>

{{$products->links()}}
```

Dengan begitu maka akan muncul tombol markup pagination yang sudah mendukung Bootstrap CSS Framework secara otomatis.

Dan pembahasan mengenai cara menampilkan link di View ini menjadi penutup bab Eloquent dan Model sekaligus pengantar bagi bab berikutnya yaitu View.

### Latihan Praktik

Kita sudah sampai pada penghujung bab ini di mana kita akan kembali berlatih dengan materi yang sudah kita pelajari. Praktik ini akan melanjutkan praktik sebelumnya.

Seperti yang telah kita lakukan di praktik sebelumnya, kita sudah membuat tabel categories dan mengisinya dengan 10 data kategori. Kini, saatnya kita mencoba untuk melakukan query terhadap data-data tersebut ya.

Latihan:

1. Tampilkan semua kategori pada route <http://toko-online.test/latihan/kategori/all>
2. Search kategori berdasarkan name <http://toko-online.test/latihan/kategori/search?name=buku>
3. Hapus data kategori dengan ID tertentu di <http://toko-online.test/latihan/kategori/3/delete> lalu kembali lihat [kategori/all](http://toko-online.test/latihan/kategori/all) memastikan tidak ada lagi kategori yang baru dihapus
4. Restore data yang telah dihapus <http://toko-online.test/latihan/kategori/3/restore> dan lihat kembali [kategori/all](http://toko-online.test/latihan/kategori/all)
5. Force delete / hapus permanen data kategori <http://toko-online.test/latihan/kategori/3/permanent-delete>

Langkah-langkah:

1. Buat controller bernama **CategoryController** dengan action `index`, `search`, `delete`, dan `restore`

```
php artisan make:controller CategoryController
```

Lalu buka <app/Http/Controllers/CategoryController.php> dan buat action controlernya. Seperti ini:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class CategoryController extends Controller
{
    /**
     * Action untuk menampilkan semua kategori
     */
    public function index(){
        return "TODO: menampilkan semua kategori dari DB";
    }

    /**
     * Action untuk mencari kategori berdasarkan nama
     */
    public function search(Request $request){
        return "TODO: menampilkan hasil search berdasarkan nama kategori";
    }
}
```

```

/**
 * Action untuk delete kategori
 */
public function delete($id){
    return "TODO: menghapus (soft delete) kategori berdasarkan ID";
}

/**
 * Action untuk merestore kategori yang telah didelete
 */
public function restore($id){
    return "TODO: merestore kategori yang statusnya dihapus (soft
delete)";
}

/**
 * Action untuk permanent delete dari database (tidak bisa direstore)
*/
public function permanentDelete($id){
    return "TODO: hapus secara permanent sebuah kategori dari DB. Tidak
bisa direstore";
}

```

2. Buat route untuk masing-masing fitur latihan;

buka `routes/web.php` lalu tambahkan kode ini:

```

Route::group(["prefix" => "latihan"], function(){
    Route::get("/kategori/all", "CategoryController@index");
    Route::get("/kategori/search", "CategoryController@search");
    Route::get("/kategori/{id}/delete", "CategoryController@delete");
    Route::get("/kategori/{id}/restore",
"CategoryController@restore");
    Route::get("/kategori/{id}/permanent-delete",
"CategoryController@permanentDelete");
});

```

Silahkan dicoba untuk mengakses masing-masing route untuk menguji apakah sudah berhasil. Harusnya masing-masing route akan menampilkan pesan string yang kita return dari masing-masing action controller.

Definisi route di atas menghasilkan route-route berikut ini:

- <http://toko-online.test/latihan/kategori/all>
- <http://toko-online.test/latihan/kategori/search>
- <http://toko-online.test/latihan/kategori/1/delete>
- <http://toko-online.test/latihan/kategori/1/restore>

- <http://toko-online.test/latihan/kategori/1/permanent-delete>

3. Siapkan soft delete pada model **Category**; Buka file **app/Category.php** dan kita tambahkan fitur **softDeletes** seperti ini kodenya:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Category extends Model
{
    use SoftDeletes;
}
```

4. Koding masing-masing controller action.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class CategoryController extends Controller
{
    /**
     * Action untuk menampilkan semua kategori
     */
    public function index(){
        return \App\Category::all();
    }

    /**
     * Action untuk mencari kategori berdasarkan nama
     */
    public function search(Request $request){
        // dapatkan keyword dari querystring ?name=keyword
        $keyword = $request->get("name");

        // cari kategori where name == keyword dari querystring
        return \App\Category::where("name", "LIKE", "%$keyword%")->get();
    }

    /**
     * Action untuk delete kategori
     */
    public function delete($id){
```

```

$category = \App\Category::findOrFail($id);

if(!$category->trashed()){
    $category->delete();
    return "Kategori $category->name berhasil dihapus";
}
}

/**
 * Action untuk merestore kategori yang telah didelete
 */
public function restore($id){
    $category = \App\Category::withTrashed()->findOrFail($id);

    if(!$category->trashed()){
        return "Kategori tidak perlu direstore";
    }
    return "Kategori $category->name berhasil direstore";
}

/**
 * Action untuk permanent delete dari database (tidak bisa direstore)
 */
public function permanentDelete($id){
    $category = \App\Category::withTrashed()->findOrFail($id);
    $category->forceDelete();

    return "Kategori $category->name berhasil dihapus permanent. Tidak bisa direstore";
}
}

```

## Kesimpulan

Kita telah belajar mengenai model dan seperti apa konvensinya. Meskipun memiliki konvensi tetapi model di Laravel sangat fleksibel karena kita bisa mengubah perilakunya dengan properti-properti khusus seperti mengubah nama tabel default, mengganti koneksi mass-assignment dan menggunakan soft delete.

Kita juga belajar mengenai mass-assignment, kenapa diperlukan dan bagaimana mengizinkan operasi ini pada field tertentu atau sebaliknya melindungi field tertentu.

Setelah itu kita belajar tentang Eloquent untuk berinteraksi dengan database tanpa menuliskan sintaks SQL. Kita belajar query, insert, update dan delete dengan Eloquent ini.

Kemudian di akhir bab kita melakukan latihan praktik terhadap materi yang baru saja kita pelajari di bab ini. Namun ternyata kita belum menggunakan View sama sekali, karena di latihan kita hanya menghasilkan data-data dalam format JSON.

Oleh karena itu bab selanjutnya kita akan belajar tentang View agar aplikasi kita memiliki interface yang cantik dan menarik bagi user. Tertarik? Siap? Yuk kita lanjutkan belajarnya!

# View

---

## Intro

View merupakan tempat bagi kita untuk meletakkan kode-kode HTML. Kita tidak akan menggunakan lagi file `.html` ya. Tapi kita tidak hanya menggunakan HTML karena kita perlu menghandle tampilan dengan lebih canggih. Menampilkan data yang diberikan oleh controller. Untuk itu kita akan menggunakan templating engine, yaitu Blade.

Blade merupakan templating engine bawaan Laravel. Berguna untuk mempermudah dalam menulis kode tampilan. Dan juga memberikan fitur tambahan untuk memanipulasi data di view yang dilempar dari controller. Apa saja fitur yang ditawarkan oleh Blade?

## Menampilkan view dari controller

Untuk menampilkan view dari controller pertama kita buat view terlebih dahulu. Coba buka file `index.blade.php` di path `resources/views/kategori/index.blade.php`. Lalu isi dengan kode sederhana ini:

```
<div>
    <b>TODO: list daftar kategori di view ini</b>
</div>
```

Lalu kita gunakan `CategoryController` yang telah kita buat pada latihan praktik sebelumnya. Buka file `CategoryController.php` lalu ubah action index sehingga menjadi seperti ini:

```
public function index(){
    return view("kategori.index");
}
```

**Penjelasan kode:** kode di atas akan menghasilkan view `kategori.index` yang baru kita buat ditampilkan di layar. Kita `return` view dari `CategoryController` action `index`.

Coba buka `http://toko-online.test/latihan/kategori/all` kita akan mendapat tampilan seperti ini:

**TODO: list daftar kategori di view ini**

## Memberikan data ke view

Ingat kembali di bab Model dan Eloquent kita telah praktik untuk membuat route dan melakukan manipulasi data menggunakan model `Category`.

Data-data di praktik tersebut ditampilkan ke browser masih dalam bentuk **JSON**. Jika untuk keperluan **web service** maka tidak mengapa, akan tetapi jika kita ingin menggunakan view Laravel, maka seharusnya kita **return** view dari controller plus dengan data.

Jika tadi kita belajar **return** view tanpa data. Maka untuk **return** view dari action controller beserta dengan data tertentu caranya mudah, yaitu tambahkan data yang akan dikirim sebagai parameter kedua dalam helper **view** dalam bentuk array, seperti ini:

### CategoryController

```
//...
public function index(){
    $daftar_kategori = \App\Category::all();

    return view("kategori.index", ["daftar_kategori" => $daftar_kategori]);
}
//...
```

**Penjelasan Kode:** Dari controller action **index** pada **CategoryController** kita return sebuah view **kategori.index**. View ini harus sudah dibuat pada path **resources/views/kategori/index.blade.php**. Selain **return** view kita juga memberikan data "daftar\_kategori" agar di view tersebut bisa diakses, nilainya adalah daftar kategori hasil *query* dengan perintah **\App\Category::all()**;

Lalu buka kembali file view **kategori/index.blade.php**, dan ubah isinya menjadi seperti ini:

```
<div>
    Nama Kategori: {{$daftar_kategori[0]->name}}
</div>
```

**Penjelasan Kode:** Menampilkan kategori pertama (index 0) dari array **\$daftar\_kategori** yang diperoleh dari **CategoryController** action **index**.

Kita baru saja belajar mengenai dasar view, dan secara tidak sadar kita juga telah menggunakan blade untuk menampilkan data yang dikirim oleh controller. Setelah ini kita akan bahas fitur-fitur Blade yang ada.

## Menampilkan data

Untuk menampilkan data kita cukup tuliskan variabel PHP seperti biasa tetapi diapit oleh dua kurung kurawal seperti ini **{{ \$namaVariabel }}**. Tentu saja **\$namaVariabel** harus sebelumnya dikirim dari controller action.

File app/routes/web.php

```
Route::get('/ucapkan-salam', 'SalamController@beriSalam');
```

Mendefinisikan route `/ucapkan-salam` yang akan mengeksekusi action `beriSalam` pada `SalamController`

File app/Http/Controllers/SalamController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class SalamController extends Controller
{
    public function beriSalam(){
        return view("salam.index", ["kalimat" => "Halo Selamat Datang"]);
    }
}
```

`SalamController` memiliki action `beriSalam` yang mengembalikan sebuah view `salam/index` dan melempar data “kalimat” yang bernilai “Halo Selamat Datang”

File resources/views/salam/index.blade.php

```
<div>
    {{ $kalimat }}
</div>
```

View yang dipakai oleh action `beriSalam` terletak pada `resources/views/salam/index.blade.php`. Karena action `beriSalam` pada `SalamController` melemparkan data “kalimat” maka kini tersedia variabel `$kalimat` pada view “`salam.index`” dan ditampilkan ke user dalam div dengan cara `{{ $kalimat }}`

## Menampilkan unescaped data

Secara default semua variabel yang tersedia di view dan ditampilkan dengan kode `{{ $namaVariabel }}` otomatis dilempar dengan fungsi `htmlspecialchars` untuk menghindari serangan XSS. Akan tetapi, terkadang kita memerlukan view untuk menampilkan unescaped data, alias tanpa `htmlspecialchars`, untuk melakukannya kita gunakan kode seperti ini:

```
{{!! $dataUnescaped !!}}
```

Sebisa mungkin kita hindari menampilkan unescaped data agar aplikasi kita terlindungi dari serangan XSS, terutama ketika kita menampilkan data dari pengguna aplikasi.

## Komentar

Seperti halnya html yang memiliki tag khusus untuk komentar, kita juga bisa memberi komentar pada view yang berguna untuk dokumentasi atau catatan sementara. Untuk melakukannya kita gunakan sintak blade

```
{{-- komentar di sini --}}
```

Dengan begitu tulisan “komentar di sini” tidak akan muncul ke layar pengguna aplikasi.

## Control Structure

### @if

Blade memiliki fitur-fitur yang sering dipakai ketika mengembangkan sebuah aplikasi. Salah satunya adalah if statement, dengan blade kita bisa menuliskan kode kondisional dengan lebih singkat dan rapi.

```
@if($showSidebar === "left")
{{-- munculkan sidebar kiri --}}
<div> SIDEBAR KIRI! </div>

@if($showSidebar === "right")
{{-- munculkan sidebar kanan --}}
<div> SIDEBAR KANAN! </div>

@else
{{-- jangan munculkan sidebar --}}
<div> TIDAK ADA SIDEBAR DITAMPILKAN </div>
@endif
```

Kode di atas tidaklah jauh berbeda dengan if statement pada PHP akan tetapi penulisannya lebih memudahkan karena tidak perlu kurung kurawal di setiap block kode statement. Jika nilai `$showSidebar` bernilai “left” maka tampilkan sidebar kiri, dan jika `$showSidebar` bernilai “right” tampilkan sidebar kanan, jika `$showSidebar` bernilai bukan “left” atau “right” tidak ada sidebar yang ditampilkan.

**Perhatikan!** kode yang diawali dengan “@” seperti @if, @endif dll disebut dengan “directive”.

### @unless

`@unless` akan mengeksekusi kode block setelahnya, kecuali criteria yang dijadikan parameter terpenuhi.

```
@unless( $status == "keren" )
    Kamu belum keren
@endunless
```

Kode blade di atas akan selalu memunculkan “Kamu belum keren” di layar kecuali nilai `$status` adalah “keren”. Akhiri blok kode `@unless` dengan `@endunless`.

### @switch

Seperti **switch** pada PHP directive **@switch** digunakan untuk menuliskan kode kondisional dengan banyak kriteria.

```
@switch($vote_status)
@case(0)
    <div> Tidak Setuju </div>
    @break

@case(1)
    <div> Setuju </div>
    @break

@default
    <div> Anda belum melakukan voting </div>
@endswitch
```

Mengecek apakah sebuah variabel tersedia

Gunakan **@isset** untuk mengecek apakah sebuah variabel telah didefinisikan dan tidak bernilai NULL.

```
@isset($productList)
// kode view untuk menampilkan daftar produk jika $productList telah
// didefinisikan dan tidak bernilai NULL
@endisset
```

Akhiri blok kode **@isset** dengan **@endisset**.

Mengecek apakah data kosong

Gunakan **@empty** untuk mengecek apakah sebuah variabel bernilai kosong (empty).

```
@empty($productList)
// kode jika $productList empty
@endempty
```

Akhiri blok kode **@empty** dengan **@endempty**.

Mengecek apakah pengguna sudah login

Gunakan **@auth** untuk mengecek apakah pengguna yang sedang mengakses sudah login ke aplikasi Laravel kita.

```
@auth
    Selamat datang!
@endauth
```

Akhiri `@auth` dengan `@endauth`. Kode yang ada di dalam block `@auth` hanya akan ditampilkan jika user telah login ke aplikasi.

## Mengecek apakah pengguna belum login

Gunakan `@guest` untuk mengecek apakah pengguna yang sedang mengakses tidak login ke aplikasi. Kode yang ada didalam block `@guest` hanya akan ditampilkan ke user yang belum login.

```
@guest
    Hai, silahkan login terlebih dahulu jika sudah punya akun.
@endguest
```

Akhiri `@guest` dengan `@endguest`

**Penjelasan:** Kita melakukan switch terhadap variabel `@vote_status`, apabila variabel tersebut bernilai 0, maka tampilkan div dengan teks "Tidak Setuju", jika variabel `@vote_status` bernilai 1 tampilkan div dengan teks "Setuju" dan jika selain itu, defaultnya menampilkan div dengan teks "Anda belum melakukan voting".

## Menampilkan Kumpulan Data

Menampilkan kumpulan data dilakukan dengan perulangan dari variabel yang berupa array. Digunakan misalnya bila kita memiliki variabel `$productList` yang berisi 10 data product. Lalu kita ingin menampilkan masing-masing produk sebagai item.

### `@foreach`

Pertama kita bisa gunakan directive `@foreach`. Directive ini membutuhkan variabel bertipe iterable, misalnya array dan minimal alias untuk masing-masing item didalamnya.

```
@foreach($productList as $product)
// sekarang kita punya akses ke masing-masing produk sebagai $product
// di dalam $productList

// tampilkan masing-masing nama produk dari array $productList
{{ $product->name }}
@endforeach
```

Akhiri directive `@foreach` dengan `@endforeach`

### `@forelse`

Directive ini hampir mirip dengan `@foreach`, bedanya kita bisa menyisipkan directive `@empty` di tengah-tengah blok kode untuk menampilkan sesuatu jika ternyata arranya kosong.

```
@forelse($productList as $product)
// sekarang kita punya akses ke masing-masing produk sebagai $product
```

```
// di dalam $productList

// tampilkan masing-masing nama produk dari array $productList
{{ $product->name }}

@empty
// jika kosong
<div>Belum ada produk</div>

@endforelse
```

Akhiri directive **@forelse** dengan **@endforelse**.

## @for

For digunakan untuk melakukan perulangan seperti halnya for pada PHP.

```
@for($i = 0; $i < 10 ; $++)
    // kita akses nilai $i di sini
    // nilai $i bernilai 1 sampai 10;
@endfor
```

## @while

Seperti halnya for, directive **@while** juga memiliki fungsi yang sama dengan while pada PHP biasa.

## Nested Loop

Nested loop merupakan perulangan di dalam perulangan. Hal ini seringkali terjadi ketika data yang dilakukan perulangan memiliki property berupa array yang bisa dilakukan perulangan lagi.

```
@foreach($productList as $product)
    // nested tingkat 1 (parent)
    {{$product->nama}}

    Kategori:
    @foreach($product->categories as $category)
        // nested selanjutnya
        {{$category->name}}
    @endforeach
@endforeach
```

## Loop Variable

Ketika kita melakukan operasi perulangan (looping), Blade menyediakan variabel yang bisa diakses dalam setiap perulangan (iteration). Variabel tersebut antara lain

Variable	Penjelasan
\$loop->index	Index perulangan saat ini. Mulai dari 0
\$loop->iteration	Perulangan keberapa saat ini. Mulai dari 1
\$loop->remaining	Berapa perulangan lagi sampai perulangan berhenti
\$loop->count	Berapa jumlah data yang dilakukan perulangan. Bernilai sama seperti panjang array.
\$loop->first	Mengecek apakah ini perulangan pertama
\$loop->last	Mengecek apakah ini perulangan terakhir
\$loop->depth	Tingkat nested loop. Apabila merupakan nested loop.
\$loop->parent	Ketika menggunakan nested loop. Akses parent loop.

## @php

Suatu ketika, kita mungkin perlu menyisipkan kode PHP seperti biasa pada template blade. Untuk melakukannya, kita bisa menggunakan directive @php seperti ini

```
@php
    //kode PHP di sini
@endphp

Akhiri directive @php dengan @endphp
```

## Blade Layout, Section & Component

Dari semua fitur Blade yang telah kita bahas, ada tiga fitur yang belum kita bahas. Dan tiga fitur ini merupakan fitur yang penting ketika menggunakan Blade, yaitu Layout, Section dan Component.

### Layout & Section

Layout digunakan untuk membuat view master yang akan selalu ditampilkan oleh view-view child yang menggunakannya. Dalam sebuah layout kita bisa memberikan tempat-tempat yang bisa digunakan oleh child view. Tempat-tempat tersebut adalah section. Misalnya, dalam layout utama, kita definisikan section sidebar, main\_content, dan footer. Selanjutnya, setiap child view yang menggunakan layout utama dapat menempatkan kode view di masing-masing section yang tersedia di layout utama.

File resources/views/layouts/app.blade.php

```
<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
```

```

@show

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

```

Pada layout view di atas, setiap kode html akan digunakan oleh child view layout tersebut tanpa harus menulisnya lagi. Dengan demikian kita tidak perlu mendefinisikan tag html, head, title, dll pada tiap-tiap view. Dan dari layout view tersebut dapat kita baca bahwa layout itu menyediakan section **sidebar**, **title**, dan **content** yang didefinisikan menggunakan directive **@yield** dan **@section**.

### **@yield**

Dengan directive **@yield("nama\_section")** sebuah layout mengharapkan konten dari child view. Untuk mengisinya, child view akan menggunakan **@section** dengan parameter nama yang sama dengan **@yield** seperti ini **@section("nama\_section")**.

### **@section**

Directive **@section** digunakan selain untuk mendefinisikan sebuah section, juga bisa untuk mengisi section yang diharapkan oleh parent view / layout melalui **@yield**.

### **@parent**

Dalam child view kita bisa menampilkan juga konten yang ada pada parent dalam section tertentu, hal tersebut dilakukan dengan directive **@parent**.

### **@extends**

Extends digunakan pada setiap child view yang ingin menggunakan sebuah view sebagai parent / layout. Mari kita coba bahas semua directive di atas dengan contoh kode.

File resources/views/layouts/app.blade.php

```

<html>
<head>
    <title>App Name - @yield('title')</title>
</head>
<body>
    @section('navbar')
        Navbar dari Layout
        <hr>
    @show

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

```

```
</body>
</html>
```

**Penjelasan kode:** Ini merupakan kode yang kita jadikan sebagai layout global aplikasi Laravel kita. Layout tersebut mengharapkan title, sidebar, dan content. Kita akan mengisinya dari child view yang mengekstend layout ini.

File resources/views/pages/about.blade.php

```
@extends("layouts.app")

@section("title")
    Tentang Kami
@endsection

@section("sidebar")
    @parent
    Sidebar dari halaman tentang kami
@endsection

@section("content")
    Kami adalah Fullstack Developer yang penuh passion untuk memecahkan
    masalah yang Anda miliki melalui aplikasi yang kami bangun!
@endsection
```

**Penjelasan Kode:** File ini pertama melakukan `@extends("layouts.app")` untuk menjadikan file view `resources/views/layouts/app.blade.php`. Kemudian kita konten untuk section title dengan "Tentang Kami" yang akan dirender sebagai `<title> App Name - Tentang Kami </title>` sesuai dengan file layout. Kemudian kita juga mengisi konten untuk section sidebar, akan tetapi kita menggunakan directive `@parent` agar selain menampilkan kalimat "Sidebar dari halaman tentang kami" juga menampilkan konten sidebar dari parent view / layout yaitu "Sidebar utama". Setelah itu kita juga isi konten untuk section bernama "content" dengan tulisan "Kami adalah Fullstack Developer dst..."

## Component

Component berfungsi untuk membuat view yang dapat kita gunakan berulang kali. Berbeda dengan layout yang bertindak sebagai master yang dapat digunakan berulang, Component justru kebalikannya, yaitu ibarat child view yang bisa kita pakai di view lain yang membutuhkannya.

Misalnya, dalam pengembangan sebuah aplikasi kita akan membutuhkan view untuk alert. Berfungsi untuk memberikan notifikasi kepada pengguna aplikasi terkait informasi, peringatan ataupun pesan error. Alert tentu saja akan digunakan berulangkali di aplikasi, bukan? Oleh karena itu kita bisa membuatnya sebagai component yang bisa digunakan di view lainnya.

File `resources/views/components/alert.blade.php`

```
<div class="alert alert-danger">
    {{ $slot }}
</div>
```

Penjelasan kode: Kode di atas akan kita jadikan sebagai component, untuk melakukannya kita perlu menampilkan variabel `$slot`, variabel ini dari mana? Dari view lainnya yang ingin menggunakan component ini.

Misalnya, sebuah view ingin menggunakan component alert tersebut. Seperti di bawah ini

File `resources/views/about.blade.php`

```
@extends('layouts.app')
// kode...

@Component("alert")
    <b>Tulisan ini akan mengisi variabel $slot</b>
@endcomponent
```

**Penjelasan kode:** File view `about.blade.php` di atas memanfaatkan component alert, caranya adalah dengan menuliskan directive `@component("alert")`. Kemudian, teks yang ada di block component itulah yang akan mengisi variabel `$slot` yang ditulis pada `alert.blade.php`.

## Latihan praktik

1. Latihan layout & section Buat file layout di `resources/views/layouts/app.blade.php` lalu isikan kode berikut:

```
<html>
    <head>
        <title>App Name - @yield('title')</title>

        <!-- BOOTSTRAP CSS -->
        <link rel="stylesheet"
            href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
            crossorigin="anonymous">

    </head>
    <body>
        @section('navbar')
        Navbar dari Layout
        <hr>
        @show

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

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

Setelah itu buat file view yang akan menggunakan layout di atas pada `resources/views/child.blade.php` dan isikan kode berikut:

```
@extends("layouts.app")

@section("title")
    Aplikasi Toko Online
@endsection

@section("content")
    Konten dari child view
@endsection
```

Setelah itu buka file `routes/web.php` dan cukup tambahkan route view pada route group "latihan" yang pernah kita buat pada praktik bab Eloquent & Model.

```
Route::group(["prefix" => "latihan"], function(){
    // KODE KODE ROUTE SEBELUMNYA
    // .....

    // Tambahkan ini
    Route::view("layouts", "child");
});
```

Dengan route tersebut maka kini kita bisa mengakses halaman `http://toko-online.test/latihan/layouts` dan kamu akan melihat konten yang berasal dari view `resources/views/layouts/app.blade.php` dan konten dari `resources/views/child.blade.php`.

Jika kamu sudah berhasil menampilkannya maka ini adalah bukti bahwa `resources/views/child.blade.php` menggunakan layouts `resources/views/layouts/app.blade.php` dan mengisi section yang disediakan di layouts yaitu `title` dan `content`.

## 2. Latihan component

- Buat file view yang akan kita gunakan sebagai component pada path `resources/views/alert.blade.php` lalu isikan kode berikut ini:

```
<div class="alert alert-warning">
    {{$slot}}
```

&lt;/div&gt;

- Buka kembali file `child.blade.php` lalu kita tambahkan kode untuk menggunakan component alert yang baru kita buat

```
@component("alert")
    Alert - Latihan berhasil
@endcomponent
```

Sehingga jika kamu buka kembali `http://toko-online.test/latihan/layouts` maka kamu akan menjumpai alert telah berhasil ditampilkan seperti ini:

Navbar dari Layout



Alert - Latihan berhasil

Konten dari child view

Tapi tunggu dulu, alert tersebut selalu menampilkan warna kuning atau type warning dari bootstrap. Gimana caranya kalo kita ingin mengganti ke type lain misalnya "success". Mari kita ubah kode component kita supaya support fitur tersebut.

- Buat component alert dapat diubah tipe alertnya. Buka file `alert.blade.php` lalu ubah kodennya menjadi seperti ini:

```
@if(isset($type))
    <div class="alert alert-{{$type}}">
        {{$slot}}
    </div>
@else
    <div class="alert alert-warning">
        {{$slot}}
    </div>
@endif
```

**Penjelasan Kode:** Kini kita memperkenalkan variabel tambahan yaitu `$type`. Variabel ini bisa digunakan nantinya oleh `child.blade.php` seperti ini:

```
@component("alert", ["type"=>"success"])
// ...
```

- Ubah kembali file `child.blade.php` sehingga memanfaatkan component alert untuk membuat alert success seperti ini:

```
@extends("layouts.app")  
  
@section("title")  
    Aplikasi Toko Online  
@endsection  
  
@section("content")  
@component("alert", ["type"=>"success"])  
    Alert - Latihan berhasil  
@endcomponent  
    Konten dari child view  
@endsection
```

Lalu buka kembali `http://toko-online.test/latihan/layouts` maka kini warna alert akan berubah menjadi hijau.

Navbar dari Layout

Alert - Latihan berhasil

Konten dari child view

Dengan demikian kini kita bisa menggunakan component alert di view manapun yang kita inginkan, dan kita bisa menyesuaikan tipe alertnya "success", "warning", "info" atau "danger" tanpa perlu membuat component sendiri untuk masing-masing tipe. Component merupakan cara yang baik untuk membuat view yang bisa digunakan berulang kali.

### 3. Latihan data pagination

Pada bab Eloquent dan Model kita telah belajar tentang melakukan pagination menggunakan model dengan fungsi `simplePaginate()` dan `paginate()`. Di situ juga telah dijelaskan perbedaan keduanya. Nah sekarang kita akan belajar untuk menampilkan pagination di view.

- Kita akan melanjutkan latihan praktik menggunakan `CategoryController` dan view `kategori/index`.
- Buka `CategoryController` dan pastikan action `index` memiliki kode berikut:

```
public function index(){
    $daftar_kategori = \App\Category::all();

    return view("kategori.index", ["daftar_kategori" =>
    $daftar_kategori]);

}
```

- Setelah itu buka view `resources/views/kategori/index.blade.php` dan ubah kodennya agar menjadi seperti ini:

```
@extends("layouts.app")

@section("content")

<ul>
    @foreach($daftar_kategori as $kategori)
        <li>{{ $kategori->name }}</li>
        <br/>
    @endforeach
</ul>

@endsection
```

**Penjelasan Kode:** Kita kembali menggunakan layout app ("layouts.app") yang terletak di `resources/views/layouts/app.blade.php` lalu mengisi section `content` dengan looping variable `$daftar_kategori` yang dikirim oleh action `index` di `CategoryController` untuk menampilkan masing-masing nama kategori.

Jika kamu akses kembali `http://toko-online.test/latihan/kategori/all` maka kamu akan menjumpai tampilan seperti ini:

## Navbar dari Layout

---

- Sepatu
- Tas
- Kemeja
- Celana
- Buku
- Kosmetik
- Komputer
- Sandal
- Furniture

- Selanjutnya kita ingin menampilkan per halaman adalah 3 item. Maka ubah kembali action `index` pada `CategoryController` menjadi seperti ini:

```
public function index(){  
    $daftar_kategori = \App\Category::paginate(3);  
  
    return view("kategori.index", ["daftar_kategori" =>  
    $daftar_kategori]);  
}
```

**Penjelasan Kode:** Kita mengubah `\App\Category::all()` menjadi `\App\Category::paginate(3)` untuk menampilkan 3 item kategori sekali tampil (per halaman)

Buka kembali `http://toko-online.test/latihan/kategori/all` kamu kini hanya melihat 3 kategori teratas seperti ini:

## Navbar dari Layout

---

- Sepatu
- Tas
- Kemeja

- Nah kini kita ingin menampilkan link pagination di view kita. Buka file `resources/views/kategori/index.blade.php` dan tambahkan kode untuk menampilkan pagination sehingga secara keseluruhan kode viewnya menjadi seperti ini:

```
@extends("layouts.app")

@section("content")

<ul>
    @foreach($daftar_kategori as $kategori)
        <li>{{ $kategori->name }}</li>
        <br/>
    @endforeach
</ul>

<hr>
<!-- INI MERUPAKAN KODE UNTUK MENAMPAILKAN PAGINATION -->
{{ $daftar_kategori->links()}}

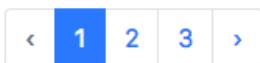
@endsection
```

Dengan kode di atas maka jika kamu buka kembali `http://toko-online.test/latihan/kategori/all` kamu akan memiliki link pagination seperti ini:

## Navbar dari Layout

---

- Sepatu
  - Tas
  - Kemeja
- 



Dan kamu boleh mengeklik ke halaman 2 atau 3 semuanya sudah berfungsi! Keren!

## Kesimpulan

Kita telah belajar tentang view dan templating enginnya yaitu Blade. Ada banyak fitur Blade yang memudahkan kita menulis kode tampilan aplikasi. Tidak hanya html tapi ada banyak directive yang sangat membantu kita sebagai developer.

Selain itu juga kita belajar tentang layouts dan section dan juga bagaimana membuat komponen yang bisa digunakan berulangkali dan dikonfigurasi dengan component. Pada akhirnya kita juga mempraktikan langung apa yang kita pelajari.

Kini kamu sudah bisa mulai untuk membuat aplikasi sederhana berbasis CRUD. Setelah ini kita akan belajar tentang Relationship. Kita akan belajar bahwa membuat relationship dan mendapatkan data relation antar model itu sangat menyenangkan di Laravel.

# Relationship

---

## Intro

Jika kamu pernah mengembangkan sebuah aplikasi sebelumnya meskipun tanpa framework saya yakin kamu pernah membuat relationship antar tabel. Dan bahkan kamu mungkin berpikir melakukan query relationship dan SQL command caranya cukup panjang. Tapi jangan kuatir, dengan Laravel semua itu akan menjadi mudah.

Dalam membuat tabel kita tidak mungkin menyimpan semua data hanya dalam satu tabel terlebih jika terdapat banyak entitas. Misalnya kita memiliki tabel-tabel berikut ini:

- user
- order
- book

Dimana setiap **user** bisa memiliki banyak **order** sementara itu **order** hanya boleh dimiliki oleh satu **user**, dengan demikian bisa dikatakan relationship antara tabel **user** dengan tabel **order** adalah **one-to-many**.

Sementara itu setiap **order** berisi banyak **book** dan setiap **books** bisa juga berada pada **order** lainnya. Berarti relationship antara **order** dengan **book** merupakan **many-to-many** relationship.

Baik **one-to-one**, **one-to-many** atau **many-to-many** relationship, semua itu bisa kita lakukan di Laravel dengan cara yang elegan. Relationship tersebut akan dilakukan dengan Model.

Sebelum kita mempelajari bagaimana melakukan setup relationship di Laravel, kita akan membahas relationship secara umum terlebih dahulu. Terutama dari sisi database, hal ini penting bagi kita agar kita bisa memahami setup relationship dengan Eloquent.

## Pengenalan relationship

### One to one

**One to one** relationship terjadi jika data A hanya boleh memiliki satu data B (A has one B) dan data B hanya boleh dimiliki oleh data A (B belongs to A).

Dalam database, kita cukup memberikan foreign key di tabel B yang akan diisi dengan ID dari data di tabel A. ID tersebut merepresentasikan data A yang memiliki data di tabel B. Kita akan menggunakan ilustrasi relation antara tabel **user** dengan tabel **phone**. Setiap user hanya boleh memiliki 1 nomor handphone dan 1 nomor handphone hanya boleh dimiliki oleh 1 user. Maka tabel keduanya adalah seperti ini:

#### **users**

---

id: Int {PK}

---

username: String

---

fullname: String

---

city: Text

Dan tabel nomor hape seperti ini:

### **phones**

---

id: Int {PK}

---

phone\_num: String

Kedua tabel tersebut belum bisa dikatakan memiliki relation, agar **one-to-one** relationship terbentuk maka kita tambahkan 1 field di **phones** sebagai foreign key yang mereferensikan ID di tabel **users** dan harus memiliki constraint unique {U} seperti ini:

### **phones**

---

id: Int {PK}

---

phone\_num: String

---

user\_id: Int {FK} {U}

Field **user\_id** akan diisi dengan user id dari tabel **users** sehingga kini kedua tabel tersebut bisa memenuhi syarat **one-to-one** relationship.

Coba kita isi masing-masing tabel seperti ini misalnya:

### **users**

<b>id</b>	<b>username</b>	<b>fullname</b>	<b>city</b>
1	johan	Johan Ziddan	Pemalang
2	nadia	Nadia Nurul Mila	Jakarta
3	hafidz	Hafid Mukhlasin	Jakarta
4	fadhilah	Fadilah Rimandani	Jakarta
5	azam	Muhammad Azamuddin	Jakarta
6	adi.lukman	Adi Lukmanul	Garut

### **phones**

#### **id**    **phone\_num**    **user\_id**

---

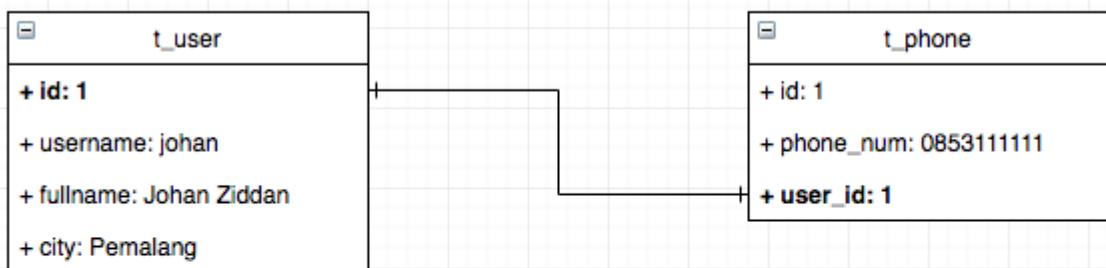
1    0873111111    1

---

2    0873222222    5

Dengan contoh data di atas kita dapat simpulkan bahwa user dengan **id** 1 yaitu Johan Ziddan memiliki nomor handphone dengan nomor 0873111111 dan user dengan **id** 5 yaitu azam memiliki nomor handphone 0873222222.

Field **user\_id** pada tabel **phones** bersifat unique sehingga tidak boleh ada nilai yang sama.



## One to many

Untuk one-to-many struktunya hampir sama, misalnya kita memiliki tabel `users` dan `orders`. Struktur tabel untuk masing-masing tabel dengan relation `one-to-many` adalah sebagai berikut:

### `users`

<code>id: Int {PK}</code>
<code>username: String</code>
<code>fullname: String</code>
<code>city: Text</code>

Dan untuk tabel `orders` adalah sebagai berikut:

### `orders`

<code>id: Int {PK}</code>
<code>total_price: Int</code>
<code>invoice_number: String</code>
<code>status: Enum</code>
<code>user_id: Int {FK}</code>

Struktur tabel di atas sudah memenuhi agar `users` memiliki relation `one-to-many` dengan `orders`. Perhatikan `orders` memiliki foreign key `user_id` yang mereferensikan `users` field `id`.

Sepintas struktur tabel `one-to-many` kok mirip dengan `one-to-one`, yup, bedanya foreign key pada `one-to-many` tidak memiliki constraint unique. Sehingga dibolehkan ada nilai yang sama untuk field `user_id` oleh karenanya relationshipnya disebut `one-to-many`. Sampai di sini cukup jelas bukan?

Kita coba buat kembali visualisasi dari data di kedua tabel di atas seperti ini:

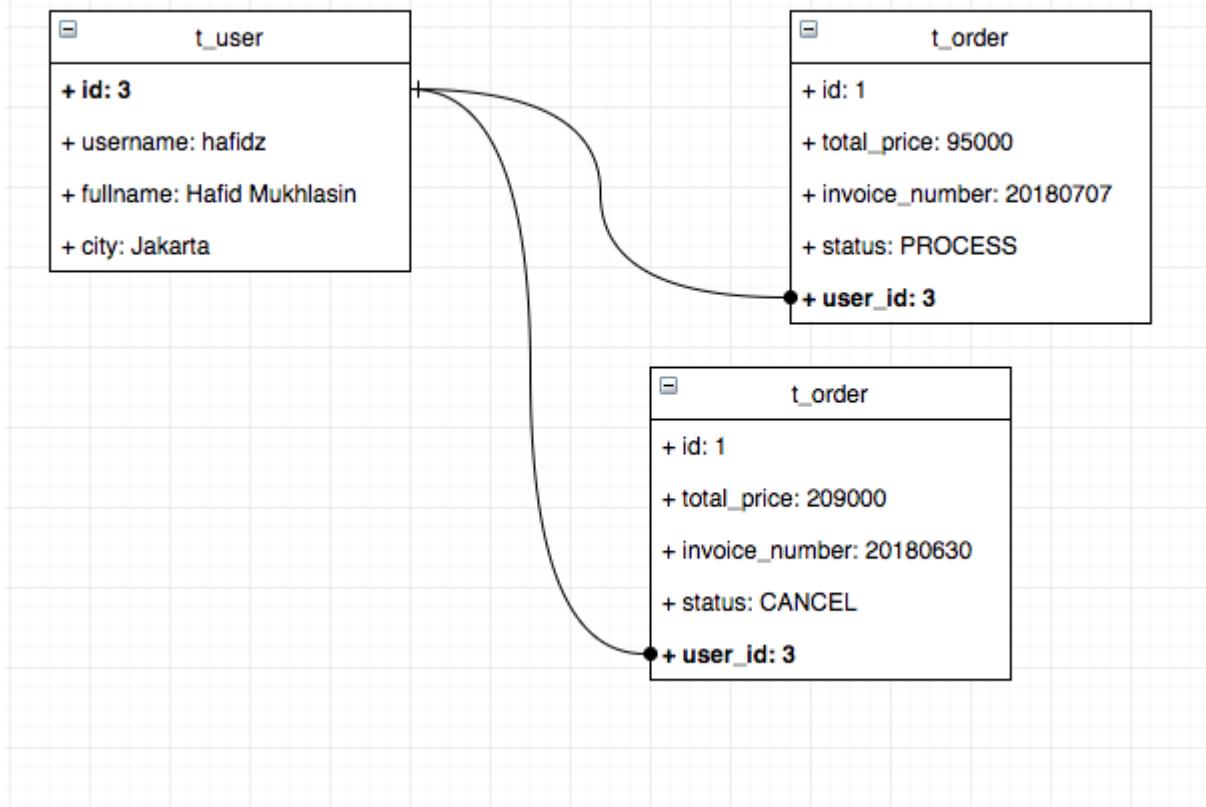
**users**

<b>id</b>	<b>username</b>	<b>fullname</b>	<b>city</b>
1	johan	Johan Ziddan	Pemalang
2	nadia	Nadia Nurul Mila	Jakarta
3	hafidz	Hafid Mukhlasin	Jakarta
4	fadhilah	Fadilah Rimandani	Jakarta
5	azam	Muhammad Azamuddin	Jakarta
6	adi.lukman	Adi Lukmanul	Garut

**orders**

<b>id</b>	<b>total_price</b>	<b>invoice_number</b>	<b>status</b>	<b>user_id</b>
1	95000	20180707	PROCESS	3
2	124000	20180705	DELIVER	4
3	209000	20180630	CANCEL	3
4	100000	20180629	FINISH	5

Perhatikan di tabel **orders** terdapat dua order yang memiliki **user\_id** bernilai 3, itu berarti user dengan **id** 3 yaitu Hafidz memiliki dua order yaitu order dengan **id** 1 dan 3. Satu user memiliki lebih dari 1 order (user has many order), sementara itu masing-masing order hanya bisa dimiliki oleh 1 user (order belongs to one user). ini berarti relationship **one-to-many** terpenuhi antara kedua tabel ini.



## Many to many

Many to many relationship sedikit berbeda strukturnya dibandingkan dua relationship yang telah kita pelajari. Ini karena data A boleh memiliki banyak data B (A has many B), dan sebaliknya data B boleh dimiliki oleh banyak data A (B belongs to many A). Kita tidak bisa menghubungkan keduanya hanya dengan tabel A dan tabel B, harus ada tabel lain yang menghubungkan keduanya, tabel tersebut kita sebut dengan istilah **pivot table** atau table penghubung.

Kita ambil sebuah ilustrasi dengan table `orders` dan tabel `books`, relationship keduanya haruslah **many-to-many** karena sebuah order boleh memiliki banyak buku yang diorder, dan buku bisa masuk ke banyak order. Maka mari kita buat struktur tabel untuk masing-masing tabel kurang lebih seperti ini:

### `orders`

---

`id: Int {PK}`

---

`total_price: Int`

---

`invoice_number: String`

---

`status: Enum`

---

`user_id: Int {FK}`

dan tabel `books`

### `books`

---

`id: Int {PK}`

---

## books

---

title: String

---

slug: String {U}

Sekarang mari kita berpikir, apakah struktur di atas cukup untuk membuat relationship **many-to-many**? Tentu tidak? apakah kita perlu menambahkan foreign key **order\_id** misalnya di **books**? Tentu tidak juga karena ini bukan **one-to-many**.

Caranya adalah dengan membuat 1 tabel lagi sebagai penghubung kedua tabel di atas, mari kita sebut dengan **book\_order** dengan struktur seperti ini:

## book\_order

---

id: Int {PK}

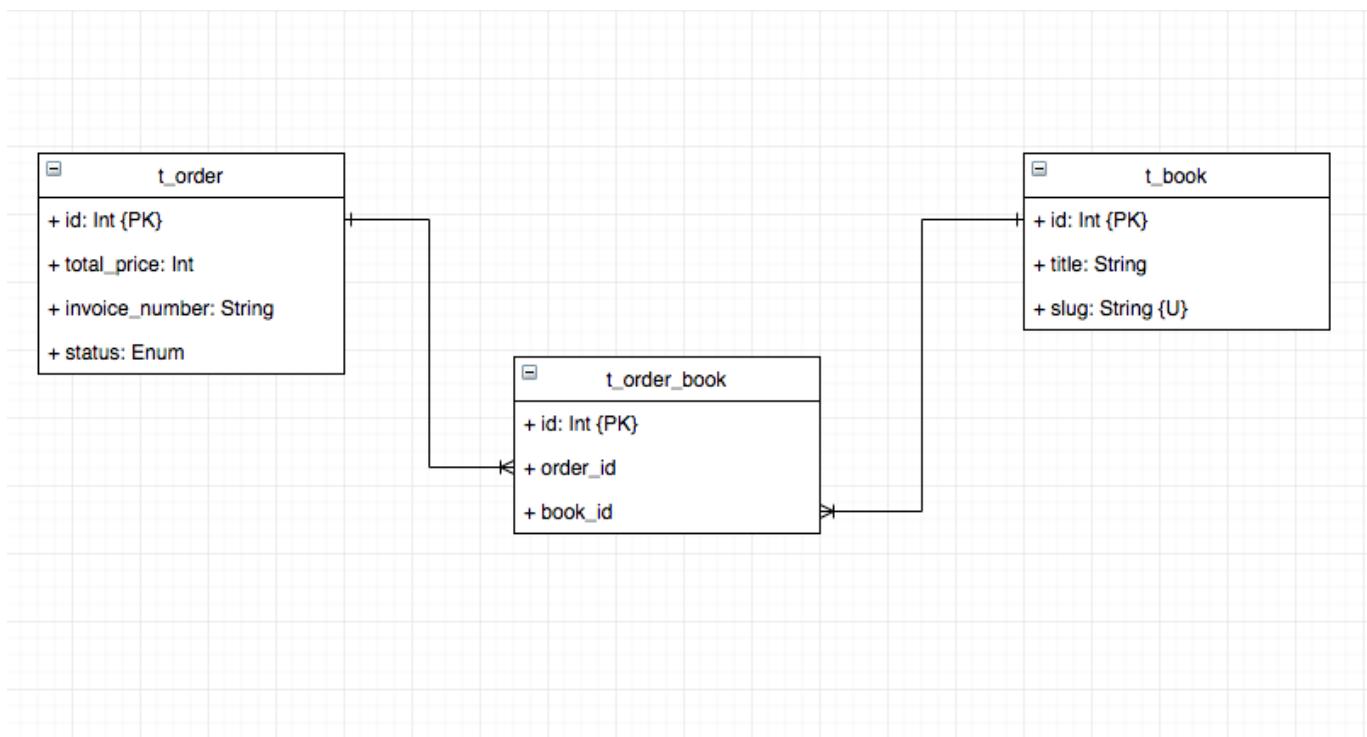
---

order\_id: Int {FK}

---

book\_id: Int {FK}

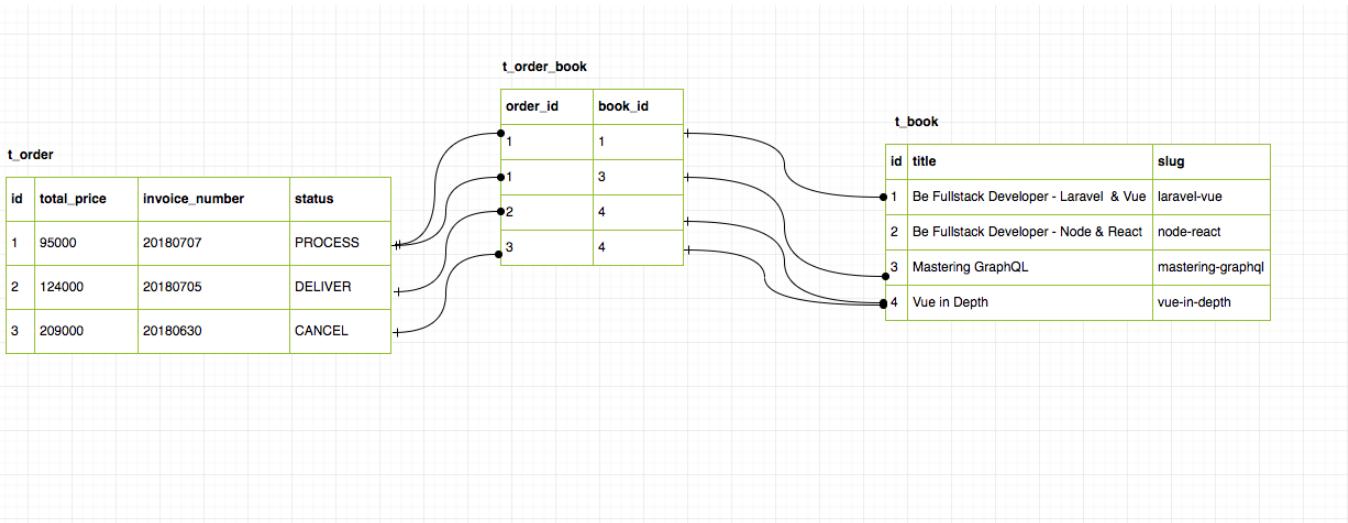
Apakah sekarang kamu sudah menangkapnya? Jika belum mari lihat visualisasi dengan diagramnya seperti ini:



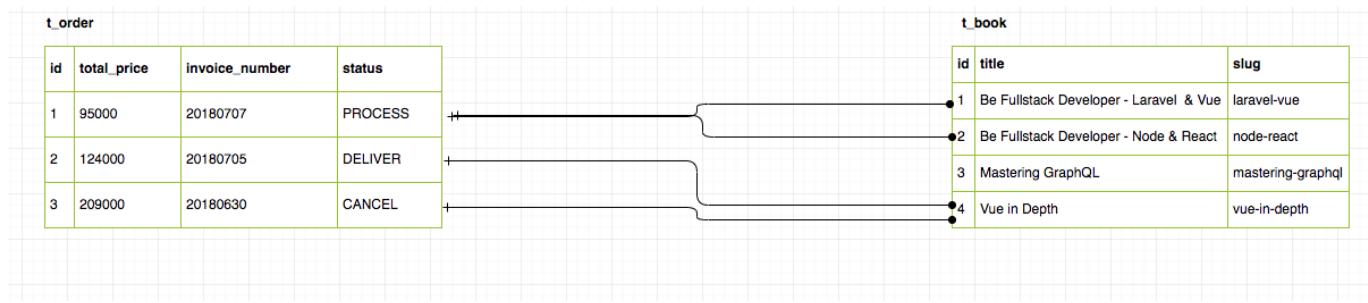
Penjelasan gambar: Diagram di atas merupakan gambaran dari relation **one-to-many** antara **orders** dengan **book\_order** melalui pivot table **book\_order**. Jika kita lihat dari diagram tersebut sebetulnya bisa kita pecah seperti ini:

- **orders** memiliki banyak (has many) **book\_order**
- **books** juga memiliki (has many) **book\_order**
- **book\_order** hanya bisa dimiliki oleh 1 (belongs to one) **orders** melalui field **order\_id**
- **book\_order** hanya bisa dimiliki oleh 1 (belongs to one) **books** melalui field **book\_id**

Itu berarti hubungan antara **orders** dengan **book\_order** adalah **one-to-many** begitu juga antara **books** dengan **book\_order**. Sehingga bila digabungkan ketiganya yang terjadi adalah **orders** bisa memiliki relation **many-to-many** dengan **books**.



Dari visualisasi di atas kita bisa coba untuk berpikir menghilangkan table penghubung dalam penghilatan kita sehingga akan terlihat lebih jelas seperti apa relation **many-to-many** antara table **orders** dengan **books** seperti di bawah ini:



Dengan melihat gambar di atas kini kita tahu bahwa order dengan **id** 1 memiliki 2 **book** yaitu buku dengan **id** 1 dan 2. Sementara itu **book** dengan **id** 4 dimiliki oleh 2 **order** yaitu **order** dengan **id** 2 dan 3. Terbukti bahwa relation antara tabel **orders** dengan **book** adalah **many-to-many** melalui tabel penghubung **books\_order**

Apa yang kita pelajari?

- one-to-one membutuhkan foreign key dengan constraint unique
- one-to-many membutuhkan foreign key TANPA constraint unique
- many-to-many membutuhkan tabel penghubung

## Relationship di Laravel

### Overview

Materi sebelumnya yaitu pengenalan relationship sangat penting untuk dipahami agar kita lebih mudah memahami bagaimana menggunakan fitur relationship di Laravel.

Karena kita menggunakan Eloquent untuk melakukan interaksi dengan database, maka kita memerlukan setup di model kita agar bisa bekerja dengan relationship.

Secara garis besar langkah-langkah yang akan kita lakukan untuk bisa menggunakan relationship di Laravel adalah sebagai berikut:

1. membuat struktur tabel untuk relationship

Untuk menyiapkan struktur tabel kita bisa menggunakan migration kembali, dan kita akan belajar bagaimana melakukannya. Di bab sebelumnya kita telah praktik menggunakan migration akan tetapi belum ada pembahasan bagaimana menyiapkan relationship dengan migration.

2. melakukan konfigurasi terhadap model model yang terkait

Setelah struktur tabel siap untuk relationship, maka langkah selanjutnya adalah kita konfigurasi model-model kita supaya bisa kita gunakan fitur-fitur Eloquent terkait relationship

Seperti apa sih gambaran seandainya model model kita sudah dikonfigurasi untuk relationship. Supaya kamu lebih paham mengapa kita perlu melakukan konfigurasi di atas, mari kita ambil contoh di pengenalan relationship. Misalnya `orders` diwakili oleh sebuah model bernama `Order` dan tabel `books` diwakili oleh model bernama `Book`. Maka kita ketahui bersama bahwa relasi kedua model tersebut sesuai contoh sebelumnya adalah `many-to-many` relationship.

Jika tanpa menggunakan model, kita perlu menuliskan kode SQL yang panjang seandainya ingin mendapatkan buku dari sebuah order. Tapi dengan model yang sudah kita konfigurasi relationshipnya maka kita bisa melakukan hal-hal ini:

- Mendapatkan buku-buku dari sebuah order

```
Order::find(1)->books;
```

- Mendapatkan order-order yang memasukan buku tertentu

```
Book::find(3)->orders;
```

- Menambahkan buku ke dalam order

```
// order dengan id 1
$order = Order::find(1);

// tambahkan buku dengan id 1,3,4
$order->attach([1,3,4]);
```

**Penjelasan kode:** Kita menambahkan 3 buku sekaligus yaitu `Book` dengan `id` 1,3 dan 4 ke dalam `Order` dengan `id` 1.

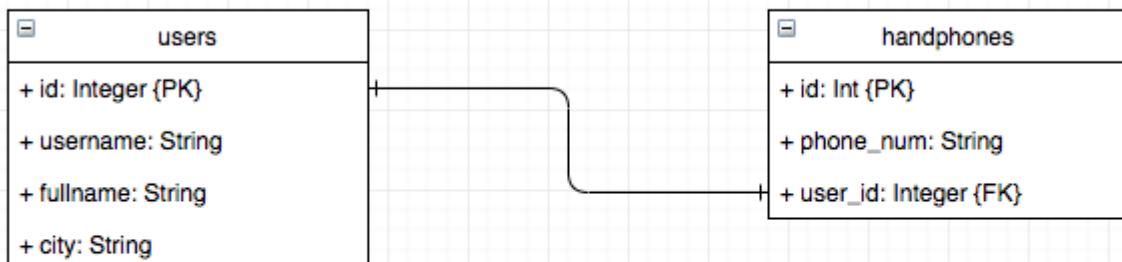
Begini mudah dan ringkas dibaca bukan? Nah tentu masih banyak lagi fitur-fitur terkait relationship yang akan segera kita pelajari. Paling tidak 3 contoh sederhana di atas membuat kamu paham manfaat dari kita melakukan konfigurasi relationship di Laravel.

Dan sebelum kita bisa menggunakan fitur-fitur Eloquent relationship yang ada, mari kita belajar untuk mempersiapkan tabel dan model kita agar mendukung relationship di Laravel.

## Struktur tabel dan model

### One to one relationship

Kita akan menggunakan pemisalan antara model **User** dan **Handphone**. Kedua tabel tersebut secara berurutan mewakili tabel **users** dan **handphones** di database. Kita akan menggunakan struktur tabel seperti ini:



### Menyiapkan tabel dengan migration

```

function up(){
    Schema::create("users", function(Blueprint $table){
        $table->primary("id");
        $table->string("username")->unique();
        $table->string("fullname");
        $table->string("city");
    });
}
  
```

**Penjelasan kode:** Kode migration untuk membuat struktur tabel **users**. Tidak ada yang khusus, menggunakan Schema builder yang pernah kita pelajari.

Karena ini konsepnya **one-to-one** tabel **users** ke **handphones** kita perlu membuat relation di migration tabel **handphones**

```

function up(){
    Schema::create("handphones", function(Blueprint $table){
        $table->primary("id");
    });
}
  
```

```
$table->string("phone_num")->unique();
$table->integer("user_id")->unsigned()->unique();

$table->foreign("user_id")->references("id")->on("users");
});

}
```

**Penjelasan kode:** File migration untuk membuat struktur tabel **handphones**. Perhatikan kita membuat foreign key **user\_id** di tabel ini yang mereferensikan field **id** di tabel **users** dengan kode ini:

```
$table->foreign("user_id")->references("id")->on("users")
```

Perlu diketahui juga sebelum kita bisa menggunakan kode di atas, ada hal yang perlu dilakukan yaitu:

1. field **user\_id** harus sudah ada terlebih dahulu, makanya jika belum ada kita buat dengan

```
$table->integer("user_id");
```

2. Field **user\_id** harus merupakan **unsigned** oleh karena itu dalam pembuatan kita juga menggunakan method **unsigned()**

```
$table->integer("user_id")->unsigned();
```

3. Field **user\_id** harus memiliki tipe yang sama dengan referensi di tabel tujuan, itu berarti tipe **integer** di field **user\_id** harus sama dengan tipe field **id** di tabel **users**. Field **id** di tabel **users** dibuat dengan method **primary()** yang secara otomatis memiliki tipe **integer**, ini berarti sudah sama. Good!

4. Karena kita akan membuat **one-to-one** relationship, maka foreign key **user\_id** harus kita tambahkan constraint **unique** dengan method **unique()** seperti ini:

```
$table->integer("user_id")->unsigned()->unique();
```

## Mendefinisikan di Model

Setelah tabel kita siap untuk keperluan relationship, maka langkah terakhir yang harus kita lakukan adalah konfigurasi dua model yang akan kita hubungkan yaitu model **User** dan model **Handphone**;

Pada model **User** kita perlu menambahkan kode berikut ini:

```
public function handphone(){
    return $this->hasOne("App\Handphone");
```

}

**Penjelasan kode:** Dengan kode ini kita mengatakan kepada model bahwa model ini (model **User**) **hasOne** atau boleh memiliki satu **Handphone**.

Sehingga nantinya kita bisa mendapatkan handphone yang dimiliki oleh user tertentu dengan cara ini:

```
// cari user dengan ID 1
$user = User::find(1);

// dapatkan model handphone dari user di atas
// akan mengembalikan satu data dengan tipe model Handphone
$handphone = $user->handphone;
```

Kemudian pada model **Handphone** kita perlu menambahkan kode ini:

```
public function user(){
    return $this->belongsTo("App\User");
}
```

**Penjelasan kode:** Dengan kode ini kita mengatakan kepada model ini (model **Handphone**) **belongsTo** atau dimiliki oleh sebuah model **User**.

Sehingga kita bisa mencari tahu siapa pemiliki nomor handpone tertentu dengan cara ini:

```
// dapatkan model handphone yang memiliki phone_num = 0852111111
$handphone = Handphone::where("phone_num", "0852111111")->first();

// cari model user yang merupakan pemilik no hape di atas
$pemilik = $handphone->user;
```

### Query one to one relationship

### Query one-to-one relationship

Kita melanjutkan perumpamaan yang kita pakai saat mendefinisikan model one-to-one relationship sebelumnya, yaitu relationship antara model **User** dengan model **Handphone**.

Seperti yang telah kita definisikan bahwa **User** memiliki 1 **Handphone** yang kita definisikan di model dalam method **handphone()**. Untuk mencari handphone dari seorang **User** maka kita cukup gunakan kode semacam ini:

```
// cari user dengan ID 34
$user = User::find(34);

// dapatkan handphone dari user dengan ID 34
$handphone = $user->handphone;

// dapatkan nomor hape dari model handphone yang baru saja didapatkan
$no_hape_user = $handphone->phone_num;
```

**Penjelasan:** User dengan `id` 34 misalnya memiliki sebuah handphone, maka bisa diakses dalam model `User` sebagai properti `handphone`

`$user->handphone` tidak perlu menggunakan `$user->handphone()` karena relation di Laravel merupakan sebuah `dynamic properties` jadi seolah-olah properti bukan method.

Misalnya kita balik, kita mencari handphone terlebih dahulu kemudian ingin mencari pemilik dari nomor tersebut juga bisa, caranya seperti ini:

```
// cari handphone dengan nomor 085211111 di DB dan dapatkan data pertama
`first()`
$handphone = Handphone::where("phone_num", "085211111")->first();

// cari siapa pemilik / user dari nomor handphone tersebut
$pemilik = $handphone->user;

// dapatkan nama lengkap dari pemilik nomor di atas
$nama_pemilik = $pemilik->fullname;
```

## One to many relationship

Kita akan menggunakan contoh sebelumnya, yaitu relation antara `User` dengan `Handphone` bedanya kini kita mengizinkan satu `User` memiliki lebih dari satu `Handphone`.

### Menyiapkan tabel dengan migration

One to many relationship memiliki struktur tabel yang mirip dengan `one-to-one` relationship, bedanya apa? sudah kita pelajari di pengenalan relationship, bedanya adalah foreign key tidak memiliki constraint unique. Jadi kita bisa menggunakan contoh migration sebelumnya bedanya kita hapus method `unique()` saat membuat foreign key `user_id` di tabel `handphones`.

### Migration `create_handphones_table`

```
function up(){
    Schema::create("handphones", function(Blueprint $table){
        $table->primary("id");
        $table->string("phone_num")->unique();
        $table->integer("user_id")->unsigned();
```

```
$table->foreign("user_id")->references("id")->on("users");
});  
}
```

Sementara itu migration untuk `create_users_table` masih sama seperti pada `one-to-one` relation.

#### Mendefinisikan di Model

Agar relationship antara model `User` dan `Handphone` menjadi `one-to-many` maka kita tambahkan kode ini di masing-masing model:

Model User

```
public function handphones(){
    return $this->hasMany("App\Handphone");
}
```

**Penjelasan kode:** Mirip dengan definisi sebelumnya, bedanya kita gunakan `hasMany` bukan `hasOne` karena kita ingin model `User` boleh memiliki lebih dari satu `Handphone`. Selain itu juga kita ubah nama relation dari bentuk tunggal `handphone` menjadi bentuk jamak yaitu `handphones`, perubahan nama ini tidak wajib hanya supaya lebih mudah dipahami bahwa user punya banyak handphone.

Sehingga kita bisa mengakses `Handphones` yang dimiliki oleh user dengan cara ini:

```
// cari user dengan id 1
$user = User::find(1);

// dapatkan handphones dari user di atas
// sekarang bentuknya adalah array of handphone bukan 1 handphone
// jadi tidak hanya 1, tapi bisa lebih dari satu Handphone
$user->handphones;
```

Untuk model `Handphone` tidak ada perubahan dari contoh di `one-to-one` relationship, yaitu konfigurasinya seperti ini:

```
public function user(){
    return $this->belongsTo("App\User");
}
```

**Penjelasan kode:** Dengan kode ini kita tetap mengatakan bahwa setiap model `Handphone` hanya dimiliki oleh satu `User`. Tidak ada sebuah `Handphone` yang dimiliki oleh beberapa `User`.

#### Many to many relationship

Kita akan menggunakan contoh pada pengenalan relationship **many-to-many** yaitu antara tabel **orders** dengan tabel **books**. Dimana **orders** diwakili oleh model **Order** dan **books** diwakili oleh model **Book**. Sebuah **Order** bisa memiliki beberapa item **Book** di dalamnya, dan sebaliknya sebuah **Book** bisa masuk ke dalam beberapa **Order**.

#### Menyiapkan tabel dengan migration

Kita telah memahami bahwa **many-to-many** berbeda karena membutuhkan pivot table. Oleh karenanya selain membuat migration untuk tabel **orders** dan **books** kita juga perlu membuat satu tabel lagi dan kita beri nama **book\_order**.

#### Migration `create_orders_table`

```
public function up(){
    Schema::create("orders", function(Blueprint $table){
        $table->primary("id");
        $table->integer("total_price");
        $table->string("invoice_number");
        $table->enum("status", ["SUBMIT", "PROCESS", "CANCEL", "FINISH"]);
    });
}
```

**Penjelasan:** Kode migration untuk membuat struktur tabel **orders**, kita tidak memberikan foreign key di tabel **orders** ke tabel **books** di sini.

#### Migration `create_books_table`

```
public function up(){
    Schema::create("books", function(Blueprint $table){
        $table->primary("id");
        $table->string("title");
        $table->string("slug")->unique();
    });
}
```

**Penjelasan:** Kode migration untuk membuat struktur tabel **books**. Kita juga tidak memberikan foreign key ke tabel **orders** di tabel ini.

#### Migration `create_book_order` table (pivot table)

```
public function up(){
    Schema::create("book_order", function(Blueprint $table){
        $table->primary("id");
        $table->integer("book_id")->unsigned();
        $table->integer("order_id")->unsigned();

        $table->foreign("book_id")->references("id")->on("books");
    });
}
```

```
$table->foreign("order_id")->references("id")->on("order");
});
```

**Penjelasan:** Kode migration untuk membuat pivot tabel yang akan kita gunakan untuk menghubungkan relation **many-to-many** antara tabel **orders** dengan tabel **books**, perhatikan kita mendefinisikan foreign key di tabel ini. Ada dua foreign key yaitu:

- **book\_id** yang mereferensikan **id** di tabel **books**
- **order\_id** yang mereferensikan **id** di tabel **orders**

**Perhatian!** konvensi penamaan untuk pivot tabel agar Laravel otomatis mendeteksi pada saat mendefinisikan **many-to-many** relationship di model adalah sebagai berikut:

- nama tabel merupakan bentuk tunggal dari dua tabel yang dihubungkan;

Misalnya tabel **orders** dan **books** maka nama pivot tabelnya adalah kombinasi antara **order** dan **book**

- kombinasi nama tabel tersebut diurutkan berdasarkan alfabet, dihubungkan dengan underscore.

Misalnya tabel **orders** dan **books** tadi memiliki pivot tabel kombinasi antara **order** dan **book** berarti nama pivot tabelnya adalah **book\_order** bukan **order\_book** karena **book** / awalan "b" secara alfabet lebih dulu dibandingkan **order** yang berawalan "o"

## Mendefinisikan di Model

### Model Order

```
public function books(){
    return $this->belongsToMany("App\Book");
}
```

**Penjelasan:** Mendefinisikan di model **Order** bahwa model ini bisa mempunyai banyak **Book**, kita memberi nama relation tersebut sebagai **books** dengan method **\$this->belongsToMany( "App\Book" );** sehingga kita bisa mengaksesnya dari model **Order** seperti ini:

```
// dapatkan order dengan ID 1
$order = Order::find(1);

// dapatkan buku dari orderan di atas
$buku_dipesan = $order->books;
```

Perhatikan bahwa untuk mendefinisikan **many-to-many** relationship di kedua model kita mendefinisikan relationship menggunakan **\$this->belongsToMany()**

### Model Book

```
public function orders(){
    return $this->belongsToMany("App\Order");
}
```

**Penjelasan:** Mendefinisikan model **Book** bahwa model ini bisa dimiliki oleh banyak **Order**, kita definisikan relation ke model **Order** sebagai **orders** dengan `$this->belongsToMany("App\Order");`. Sehingga kita bisa mengakses order apa saja yang memuat buku tertentu seperti ini:

```
// dapatkan buku dengan id 4
$buku = Book::find(4);

// cari order yang memiliki buku dengan id 4
$orderan_yang_memuat_buku_ini = $buku->orders;
```

## Menghubungkan relationship

Untuk menghubungkan sebuah model **User** dengan **Handphone** dalam **one to one** relationship kita bisa gunakan beberapa cara

### save()

Yang pertama adalah **save()** method. Kita gunakan dengan cara seperti ini:

```
$handphone_baru = new Handphone(["phone_num"=>"099999999"]);

$user = User::find(1);

$user->handphone()->create($handphone_baru);
```

**Penjelasan:** Untuk menggunakan **save()** pertama kita buat terlebih dahulu data handphone dengan `new Handphone(["..."])` (ini belum menyimpan ke database), lalu kita cari user tertentu misalnya dengan **id** 1 simpan sebagai **\$user**. Setelah itu kita create handphone tadi dan sekaligus tambahkan ke user tadi dengan cara `$user->handphone()->create($handphone_baru)`

**Perhatikan** Sekarang kita menggunakan `$user->handphone()` bukan `$user->handphone` seperti contoh sebelumnya. Itu karena kita ingin melakukan **chaining**, yaitu ingin menggunakan method lain dalam hal ini `create()`. Sama halnya jika kita ingin menggunakan method-method lainnya. Tapi jika hanya ingin mendapatkan handphone dari user maka cukup gunakan `$user->handphone`

### create()

Cara berikutnya adalah menggunakan **create()** method. Kita gunakan dengan cara yang hampir sama seperti sebelumnya:

```
$user = User::find(1);  
  
$user->handphone()->create(["phone_num"=>"0877777777"]);
```

**Penjelasan:** Mirip dengan cara `save()` method, kita menggunakan `create()` untuk membuat handphone baru untuk user dengan `id` 1. Bedanya kita langsung memberikan properti yang akan diinsert ke tabel `handphones` tanpa perlu melakukan `new Handphone` seperti method `save()`

### Perbedaan `create()` dan `save()`

- Method `save()` mengharapkan sebuah argument yaitu instance object dari model yang akan ditambahkan, dalam hal ini model `Handphone` makanya kita gunakan terlebih dahulu `new Handphone()` untuk membuat instance.
- Sementara itu, method `create()` mengharapkan sebuah argument yaitu `Array` dari properti yang akan diinsert ke tabel relationship, dalam hal ini tabel `handphone`, misalnya `["phone_num" => "08777777"]` atau `["phone_num"=>"0811111111"]`

### `associate()`

Cara ketiga adalah dengan menggunakan method `associate()` seperti ini:

```
$user = User::find(1);  
  
$handphone = Handphone::where("phone_num", => "0877777777");  
  
$user->handphone()->associate($handphone);
```

**Penjelasan:** Method `associate()` mengharapkan sebuah argument yaitu instance dari model relationship yang akan ditambahkan, dalam hal ini model `Handphone`. Karena method ini tidak mencreate data baru ke database kita bisa menggunakan method ini untuk menghubungkan handphone yang sudah ada di database ke user tertentu seperti pada kode di atas, kita tidak mengcreate handphone baru tapi melakukan query ke tabel handphone yang memiliki `phone_num == "0877777777"` lalu menambahkan hanphone tersebut ke `User` dengan `id` 1.

### `createMany()`

Create many berfungsi sama seperti `create()` akan tetapi digunakan untuk mengcreate banyak relationship data sekaligus. Method ini tidak bisa digunakan untuk `one-to-one` relationship.

Method ini mengharapkan sebuah parameter yaitu multidimensional Array berisi properti-properti relationship yang akan diinsert.

Misalnya kita bayangkan sebuah relationship `many to many` antara model `Product` dengan `Category`. Di mana model `Post` memiliki banyak `Category` dengan relationship properti `categories`. Kita bisa membuat kategori-kategori baru sekaligus menghubungkannya ke `Product` tertentu seperti ini:

```
$post = Post::find(23);

$post->categories()->createMany([
    [
        "name" => "Sepatu"
    ],
    [
        "name" => "Fashion Pria"
    ]
]);
```

**Penjelasan:** Membuat 2 kategori baru yaitu sepatu dan fashion pria sekaligus menambahkannya sebagai kategori dari **Post** dengan **id** 23.

Dengan begitu, maka jika kita menggunakan kode berikut ini setelahnya:

```
$post->categories()
```

Maka akan menghasilkan 2 **Category** yaitu "Sepatu" dan "Post". Seperti ini:

```
[
    [
        "id" => "ID_DARI_DB",
        "name" => "Sepatu"
    ],
    [
        "id" => "ID_DARI_DB",
        "name" => "Fashion Pria",
    ]
]
```

## attach()

Khusus untuk **many to many** relationship, disediakan method tersendiri agar lebih mudah untuk mengelola relationship jenis ini. Untuk menambahkan kita relationship kita gunakan method **attach()** seperti ini:

```
$product = Product::find(231);

$product->categories()->attach([1,2,4]);
```

**Penjelasan:** Kita menghubungkan **Product** 231 dengan **Category** yang memiliki **id** 1,2 dan 4. Seandainya sebelumnya **Product** ini sudah memiliki kategori dengan id lain yang terhubung, misalnya 10 dan 3, maka attach akan menambahkannya. Maka sekarang **Product** tadi memiliki kategori dengan id 1,2,3,4 dan 10.

## Menghapus relationship

### dissociate()

Untuk menghapus relationship kita gunakan method `dissociate()` seperti ini:

```
$user = User::find(1);  
  
$user->handphone()->dissociate();
```

**Penjelasan:** Menghapus relationship handphone yang dimiliki oleh `User` dengan `id` 1. Dengan begitu kini data handphone yang sebelumnya dimiliki oleh user tersebut kini akan diupdate dengan field `user_id` sebagai foreign key bernilai null, bukan lagi bernilai 1.

### detach()

Method `detach()` merupakan cara lain untuk menghapus relationship dan merupakan kebalikan dari `attach()`.

```
$product = Product::find(231);  
  
$product->categories()->detach([10,3]);
```

**Penjelasan:** Menghapus relationship `Category` dengan `id` 10 dan 3 dari `Product` 231.

## Sinkronisasi relationship

Sinkronisasi digunakan untuk memudahkan kita mengelola `many to many` relationship. Method yang digunakan adalah `sync()` dan method ini merupakan gabungan antara method `attach()` sekaligus `detach()`. Dengan kata lain, menggunakan method ini kita menambahkan relationship dengan id-id tertentu sekaligus menghapus relationship dengan id-id yang lain jika sebelumnya ada.

Contoh menggunakan ilustrasi `many to many` antara model `Product` dan `Category`. Di mana sebuah `Product` dengan id 231 saat ini memiliki `Category` 1,2,3,4,10,23, yaitu berjumlah 6 kategori. Lalu kita mengeksekusi kode berikut ini:

```
$product = Product::find(231);  
  
$product->categories()->sync([1,3]);
```

**Penjelasan:** Melakukan sinkronisasi relationship antara `Product` 231 dan `Category` `id` 1, `id` 3. Dengan kode tersebut maka kini `Product` tadi hanya akan memiliki `Category` 1 dan 3, sementara `Category` lainnya yang sebelumnya terhubung yaitu 2,4,10 dan 23 akan otomatis dihapus dari `Product` 231.

## Querying relationship tingkat lanjut

## Mendapatkan data model hanya yang memiliki relation tertentu

Untuk mendapatkan sebuah data yang memiliki relationship tertentu kita gunakan method `has()`

```
$categorized_products = Product::has("categories")->get();
```

Penjelasan: Mendapatkan semua product yang memiliki kategori.

```
$orders = Order::has("books", ">=", 4)->get();
```

Penjelasan: Mendapatkan semua order yang didalamnya terdapat 4 buku atau lebih.

## Mendapatkan data berdasarkan query terhadap relationship model

Kita juga bisa mendapatkan data berdasarkan query terhadap relationship menggunakan method `whereHas()` seperti ini:

```
$orders = Order::whereHas("books", function($query){
    $query->where("title", "Advanced Fullstack Developer");
})->get();
```

Penjelasan: Mendapatkan semua order yang memiliki buku berjudul "Advanced Fullstack Developer"

## Mendapatkan hanya data model yang tidak memiliki relation tertentu

Sebaliknya kita juga bisa melakukan query terhadap model yang tidak mempunyai relation tertentu, kita gunakan method `doesntHave()`.

```
$uncategorized_products = Product::doesntHave("categories")->get();
```

Penjelasan: Dapatkan semua produk yang tidak memiliki kategori.

Selain itu kita juga bisa menambahkan query lebih lanjut terhadap relationship menggunakan method `whereDoesntHave()` misalnya:

```
$orders = Order::whereDoesntHave("books", function($query){
    $query->where("title", "MS Access");
});
```

Penjelasan: Dapatkan semua order yang tidak memiliki buku berjudul "MS Access" di dalamnya.

## Menghitung jumlah relationship

```
$order = Order::withCount("books")->first();  
  
$jumlah_buku = $order->books_count
```

Penjelasan: Dapatkan order pertama dan dapatkan juga jumlah buku di orderan tersebut. Laravel akan otomatis memberikan properti dengan format {relationship}\_count seperti contoh di atas `books_count`.

Contoh lain

```
$product = Product::withCount("categories")->first();  
  
$jumlah_kategori = $product->categories_count;
```

Penjelasan: Dapatkan product pertama dan dapatkan juga jumlah kategorinya.

Method `withCount()` juga bisa kita berikan query lebih lanjut misalnya seperti ini:

```
$products = Product::withCount(["categories" => function($query){  
    $query->where("name", "like", "%anak%");  
}])
```

Penjelasan: Dapatkan produk-produk dan jumlah kategori yang memiliki kata-kata "anak" di dalamnya.

Lebih lanjut `withCount()` juga bisa digunakan untuk multiple relationship seperti ini:

```
$products = Product::withCount(["categories", "comments"])->get();  
  
$jumlah_kategori = $products->categories_count;  
$jumlah_komentar = $products->comments_count;
```

Penjelasan: Dapatkan produk-produk beserta dengan jumlah kategori dan jumlah komentarnya.

## Foreign Key Constraint

Cascading merupakan fitur yang memungkinkan kita untuk mendefinisikan perilaku terhadap tabel-tabel yang saling berhubungan dalam relationship. Perilaku tersebut bisa kita definisikan ketika terjadi perubahan terhadap salah satu data di salah satu tabel (on update) atau jika data di salah satu tabel dihapus (on delete). Ada 4 yaitu SET NULL, CASCADE, NO ACTION dan RESTRICT.

### Tipe-tipe constraint

SET NULL

Jika ada data di tabel parent yang dihapus atau diupdate, maka foreign key di tabel-tabel yang berkaitan akan otomatis diupdate dengan NULL.

Jika kamu menggunakan SET NULL, pastikan foreign key di tabel child telah menggunakan NULLABLE, atau di migration menggunakan `nullable()`

#### RESTRICT

Menolak penghapusan atau update terhadap data di parent tabel. Ini merupakan default di MySQL.

#### CASCADE

Jika terjadi penghapusan atau perubahan data di parent model maka data di child model akan mengikuti.

Itu berarti jika data di parent tabel dihapus, maka seluruh data di child tabel yang mereferensikan data di parent tabel akan dihapus juga.

#### NO ACTION

Secara fungsi sebetulnya sama saja dengan RESTRICT

### Menerapkannya dalam migration

Kita bisa menerapkan foreign key constraint menggunakan migration. Caranya adalah seperti ini saat mendefinisikan relationship antar tabel. Kita ambil contoh tabel `handphones` yang memiliki foreign key `user_id` mereferensikan `id` di tabel `users`.

```
function up(){
    Schema::create("handphones", function(Blueprint $table){
        $table->primary("id");
        $table->string("phone_num")->unique();
        $table->integer("user_id")->unsigned()->unique();

        $table->foreign("user_id")->references("id")->on("users")-
        >onDelete("cascade");
    });
}
```

Penjelasan: Mengatur foreign key constraint antara tabel `hanphones` dan `users` menjadi "CASCADE" menggunakan method `onDelete("cascade")`. Dengan demikian maka jika sebuah data user dihapus maka record handphone yang berkaitan di tabel `handphones` juga akan ikut dihapus.

Untuk menggunakan restriction lain cukup masukan parameter di function `onDelete()`, misalnya `onDelete("restrict")`, `onDelete("set null")`. Dan untuk menerapkan on update constraint gunakan method `onUpdate()`.

Misalnya seperti ini:

```
$table->foreign("user_id")->references("id")->on("users")-
>onDelete("cascade")->onUpdate("cascade");
```

## Kustomisasi definisi relationship

Pada subbab ini kita akan melakukan kustomisasi terhadap definisi relationship di model. Bab ini umumnya tidak perlu juga kamu mengikuti konvensi-konvensi penamaan tabel untuk relationship di laravel.

Tapi terkadang kamu memiliki struktur tabel yang berbeda, apalagi jika bukan kamu yang membuat databasenya. Atau kamu ingin mengubah database yang sudah ada dengan aplikasi Laravel yang sebelumnya bukan.

Intinya, terkadang kamu tidak mengikuti konvensi penamaan di Laravel tapi kamu ingin memanfaatkan fitur relationship tanpa harus mengubah struktur tabel yang sudah ada. Mari kita pelajari

### Mengubah foreign key di relationship

Model `User` memiliki relation dengan `Handphone` di model `User` kita mendefinisikan sebagai relationship `handphone()`, cukup seperti ini:

```
public function handphone(){
    return $this->hasOne("App\Handphone");
}
```

Kita tidak perlu menuliskan foreign key apa yang digunakan oleh relationship ini, karena ini model `User` maka Laravel mengasumsikan ada field `user_id` diambil dari `{model}_id`. Hal ini tidak masalah jika di tabel `handphones` memang terdapat field `user_id`, seandainya ternyata bukan field `user_id` tapi ternyata di tabel `handphones` menggunakan tabel `owner_id` maka kita mendefinisikan relationship kita di model `User` seperti ini:

Model `User`

```
public function handphone(){
    return $this->hasOne("App\Handphone", "owner_id");
}
```

Dan lebih lanjut lagi, ternyata tabel `users` kita menggunakan primaryKey bukan field `id` tetapi field dengan nama `customer_id` padahal Laravel mengasumsikan foreign key `owner_id` mereferensikan field `id` di `users`, kita bisa memberitahu Laravel bahwa foreign key kita mereferensikan `customer_id` bukan `id` seperti ini:

```
public function handphone(){
    return $this->hasOne("App\Handphone", "owner_id", "customer_id");
}
```

## Mengubah pivot tabel pada many-to-many relationship

Kita juga telah belajar konvensi penamaan tabel untuk pivot pada **many to many** relationship di bab ini juga. Tapi kita juga bisa memberitahu Laravel bahwa kita akan menggunakan custom nama pivot tabel misalnya **book\_order\_pivot** dalam relationship antara **Order** dan **Books** maka kita definisikan relationshipnya seperti ini:

Model **Order**

```
public function books(){
    return $this->belongsToMany("App\Book", "book_order_pivot");
}
```

Kemudian di model **Book** Model **Book**

```
public function orders(){
    return $this->belongsToMany("App\Order", "book_order_pivot");
}
```

## Mengubah foreign key pada many-to-many relationship

Lebih lanjut kita juga bisa melakukan perubahan terhadap field-field apa saja yang kita gunakan di tabel pivot tabel kita seperti ini:

Model **Order**

```
public function books(){
    return $this->belongsToMany("App\Book", "book_order_pivot",
        "customer_id", "book_id");
}
```

## Kesimpulan

Di bab ini kita belajar intense terkait model relationship. Dan memang model relationship ini membuka kemampuan kita untuk mengembangkan aplikasi yang kompleks dengan lebih mudah. Karena banyak sekali fitur-fitur yang bisa kita gunakan untuk mengelola relationship baik itu **one to one**, **one to many** atau **many to many** relationship.

Selain itu kita juga telah belajar pengenalan masing-masing tipe relationship di atas. Pengenalan tersebut memahamkan kita mengenai konsep relationship dasar dalam relationship database management system. Itu berarti pemahaman tersbeut bisa kamu pakai tidak peduli apapun frameworknya.

Dan di bab ini juga kita belajar menerapkan pemahaman kita mengenai relationship ke dalam aplikasi Laravel mulai dari menyusun struktur tabel kita menggunakan migration kemudian melakukan definisi relationship di

Licensed to Adhitya Venancius - adhityavenancius@yahoo.com - 087882568270 at 20/11/2018 17:12:45  
model. Dan terakhir kita banyak belajar mengenai fitur-fitur eloquent relationship.

Di bab ini tidak ada latihan praktik, namun kamu diharapkan untuk memahami apa yang ada di dalamnya karena kita akan langsung mempraktikan ilmu yang kita pelajari ini di bab study kasus nanti.

# Studi Kasus Manajemen Toko

---

## Gambaran Studi Kasus

Dalam studi kasus ini kita akan membuat manajemen toko menggunakan Laravel (tanpa VueJS). Sementara itu untuk front store akan menggunakan Laravel Web Service + VueJS yang kan dibahas di studi kasus berikutnya.

Untuk styling kita akan menggunakan Bootstrap v4 dengan menggunakan template karya salah satu penulis buku ini, yaitu polished template [Polished Template](#).

Adapun manajemen toko ini akan menggunakan contoh toko yang menjual produk buku. Diharapkan pembelajar setelah mengikuti studi kasus ini akan bisa membuat sistem manajemen toko menggunakan Laravel. Adapun fitur-fitur yang akan dikembangkan antara lain:

- Authentication User
- Otorisasi User
- Manajemen User
- Manajemen Buku
- Manajemen Kategori
- Manajemen Order

Pada studi kasus ini kita akan menerapkan pengetahuan yang kita pelajari di bab-bab sebelumnya seperti:

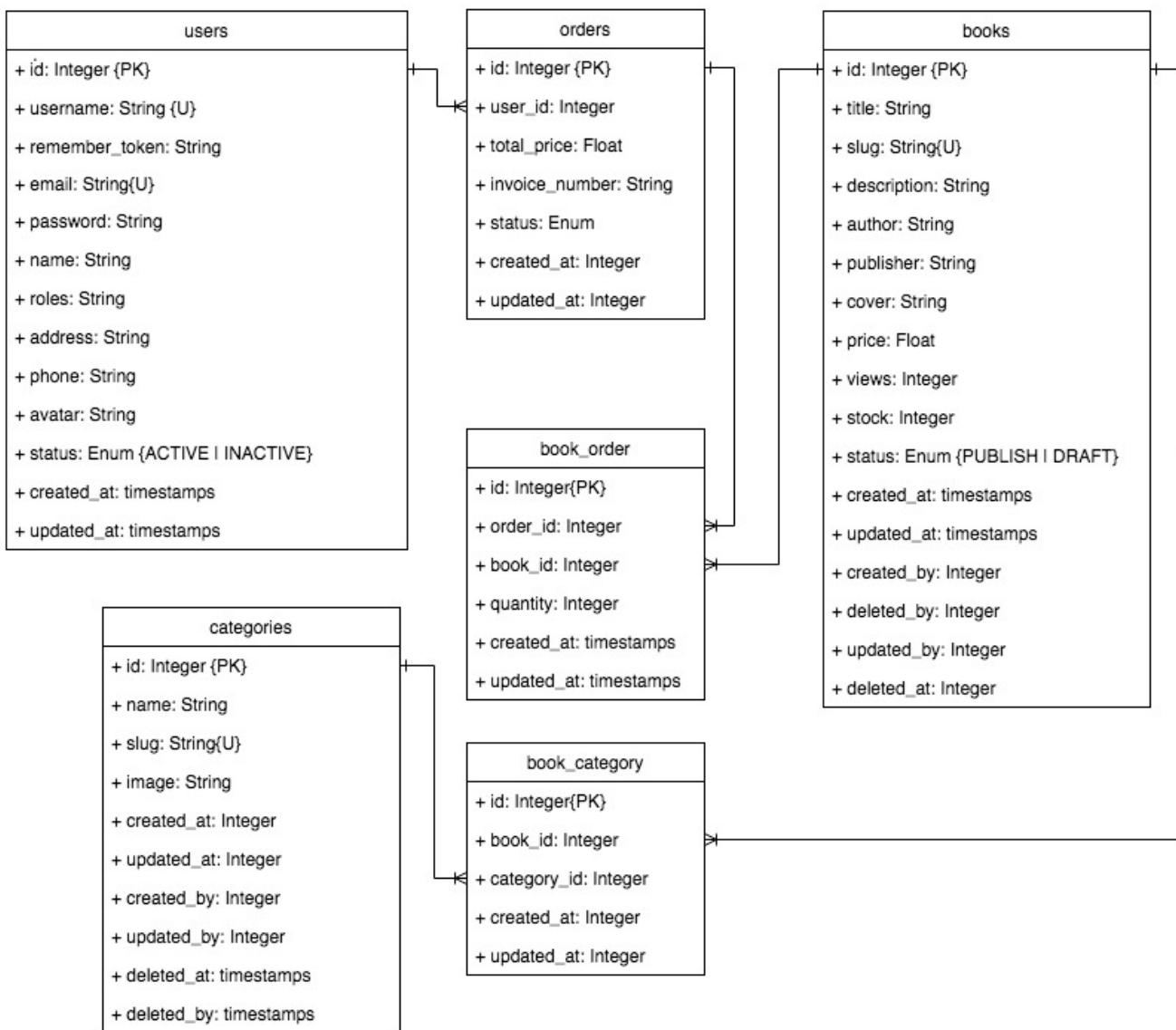
- Arsitektur Laravel
- Perintah-perintah artisan
- Migration
- Database Seeding
- Routing
- Controller
- Model & Eloquent
- View & Blade
- Membuat Pagination
- Model Relationship
- Soft Deletes
- Layouts

Selain itu kita juga akan belajar hal-hal yang belum kita bahas sebelumnya seperti:

- Authentication dengan Auth Scaffolding
- Hashing
- Integrasi dengan Bootstrap
- Membuat UI yang eye-catching dengan Polished Template
- Implementasi Form di Laravel
- Validasi form dengan Form Request
- Otorisasi dengan Gate
- Implementasi Fitur Search

## Desain database

Langkah pertama adalah kita harus mendesain database kita untuk mengakomodasi fitur-fitur yang akan kita bangun. Adapun desain database untuk studi kasus ini adalah sebagai berikut:



Untuk tabel orders field status bertipe ENUM dengan pilihan sebagai berikut:

- SUBMIT
- PROCESS
- FINISH
- CANCEL

Selanjutnya kita akan secara bertahap membuat masing-masing tabel di atas menggunakan migration. Ikuti langkah-langkahnya ya 😊

## Membuat Project Laravel Baru

### Pengguna XAMPP

Pertama kita buat sebuah project baru di direktori development kita, untuk pengguna XAMPP berarti di direktori **htdocs**. Buka terminal lalu pindah ke **htdocs**

```
cd C:/xampp/htdocs
```

## Pengguna Laradock

Untuk pengguna laradock buka direktori yang telah kita pilih untuk meletakkan project-project laravel kita, yaitu di **laravel-projects** sesuai dengan apa yang kita lakukan di bab Instalasi dan Konfigurasi. Kemudian pastikan **Docker** sudah berjalan dan pastikan service **nginx**, **mysql**, **phpmyadmin** dan **redis** juga sudah berjalan. Jika service-service tadi belum berjalan, jalankan perintah ini:

```
cd laravel-project/laradock
```

```
docker-compose up -d nginx mysql phpmyadmin redis
```

Lalu kembali ke direktori project

```
cd ..
```

Lalu masuk ke workspace kita dengan perintah:

```
docker-compose exec --user=laradock workspace bash
```

**laravel new project**

Setelah itu baik pengguna XAMPP ataupun Laradock jalankan perintah untuk membuat project baru:

```
laravel new larashop
```

Dengan command di atas kita sekarang memiliki sebuah project laravel baru di folder **larashop**.

Setelah itu masuk ke direktori aplikasi baru tadi dengan terminal dan jalankan

```
composer install
```

Jika sudah, maka **Untuk pengguna XAMPP** silahkan langsung buka <http://localhost/larashop/public>. Bila perlu lakukan konfigurasi untuk menghilangkan path **public** dari URL agar selanjutnya bisa mengakses <http://localhost/larashop> (telah kita pelajari juga di bab Instalasi dan Konfigurasi).

Untuk pengguna Laradock, setelah berhasil membuat project baru lakukan hal ini:

1. buat virtual host baru

buka file hosts Untuk mac atau linux gunakan perintah ini:

```
sudo nano /etc/hosts
```

untuk Pengguna Windows 10 file tersebut ada di <c:\Windows\System32\Drivers\etc\hosts>

Setelah itu tambahkan virtual host baru:

```
127.0.0.1 larashop
```

2. buat konfigurasi nginx baru

Setelah itu kita buat file baru dan beri nama <larashop.conf> di folder nginx laradock <laradock/nginx>.

Isinya adalah sebagai berikut:

```
server {

    listen 80;
    listen [::]:80;

    server_name larashop.test;
    root /var/www/larashop/public;
    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri /index.php =404;
        fastcgi_pass php-upstream;
        fastcgi_index index.php;
        fastcgi_buffers 16 16k;
        fastcgi_buffer_size 32k;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        #fixes timeouts
        fastcgi_read_timeout 600;
        include fastcgi_params;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

```
}

location /.well-known/acme-challenge/ {
    root /var/www/letsencrypt/;
    log_not_found off;
}

error_log /var/log/nginx/laravel_error.log;
access_log /var/log/nginx/laravel_access.log;
}
```

Jika sudah kini pengguna laradock bisa mengakses di browser [larashop.test](#)

Jika kamu mengalami error `No application encryption key has been specified.` Jangan lupa pastikan sudah punya file `.env`, jika belum rename file `.env-example` menjadi `.env`. Lalu jalankan perintah

```
php artisan key:generate
```

## Konfigurasi database

Kita terlebih dahulu perlu menyiapkan database yang akan kita gunakan.

Buka phpmyadmin lalu create database baru bernama `larashop`

- Pengguna XAMPP buka <http://localhost/phpmyadmin>
- Pengguna laradock buka <http://localhost:8080>

Ubah `.env`

buka `.env` di folder project larashop kita. Jika belum ada `.env`, cukup rename file `.env-example` menjadi `.env`. Lalu isikan konfigurasi database.

File `.env`

```
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=larashop
DB_USERNAME=root
DB_PASSWORD=root
```

Untuk pengguna XAMPP ganti `DB_HOST` dari `mysql` menjadi `localhost` seperti ini:

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=larashop
DB_USERNAME=root
DB_PASSWORD=root
```

## User Authentication

### Mengenal Authentication

Authentication berasal dari kata authentic, yang berarti asli, original. Secara istilah maksudnya adalah proses untuk memastikan keaslian atau kebenaran. Sehingga user authentication dapat diartikan sebagai proses untuk memastikan keaslian atau kebenaran pengguna yang mengakses aplikasi. Sederhananya, proses ini sering kamu jumpai dalam bentuk pengecekan menggunakan user dan password. Ini merupakan bentuk user authentication yang lazim dipakai saat ini terutama untuk aplikasi berbasis web. Apapun caranya, tujuannya adalah sama yaitu memastikan keaslian atau kebenaran pengguna yang mengakses aplikasi.

Dalam sebuah aplikasi, user authentication merupakan bagian yang harus selalu ada. Terlebih aplikasi yang membutuhkan pembatasan akses terhadap fitur-fiturnya. user authentication juga merupakan syarat bisa dilakukan otorisasi pengguna. Proses ini seringkali memakan waktu, padahal hal-hal ini seharusnya tidak terlalu menyita waktu developer dalam membuat aplikasi. Bayangkan jika setiap membuat aplikasi baru kita harus menghabiskan waktu dan energi untuk membuat sistem Authentication? Kabar baiknya, dengan Laravel kamu tidak perlu menghabiskan banyak waktu hanya untuk mempersiapkan sistem user authentication.

Dengan laravel, kamu cukup jalankan satu command dan aplikasi Laravelmu langsung siap untuk melakukan user authentication, lengkap dengan registrasi, login, forgot password.

### User Authentication di Laravel

#### Scaffolding

Kita akan membuat fitur Authentication untuk proyek toko online ini menggunakan fitur bawaan Laravel. Yaitu dengan perintah `artisan make:auth`. Buka terminal / cmd lalu ketik perintah ini:

```
php artisan make:auth
```

Apabila berhasil dijalankan maka kamu akan melihat tampilan seperti di bawah ini:

```
Authentication scaffolding generated successfully.
laradock@da3152cad38f:/var/www/online-shop$
```

Dengan begitu maka file-file yang diperlukan untuk Authentication termasuk halaman login akan digenerate.

Selanjutnya kita cukup jalankan perintah `artisan migrate` yang akan menjalankan dua file migrations yang terkait dengan Authentication yaitu file yang berakhiran dengan nama `create_users_table.php` dan `create_password_resets_table.php`. File ini sudah ada sejak pertama kali kita membuat project Laravel baru.

Langsung saja kita jalankan migration yang akan mengeksekusi dua file tadi. Kembali buka terminal / cmd mu lalu ketik

```
php artisan migrate
```

Jika berhasil dijalankan maka akan muncul tampilan seperti di bawah ini di terminal / cmd

```
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
laradock@da3152cad38f:/var/www/online-shop$
```

Apabila perintah tersebut berhasil dijalankan, maka kini aplikasimu sudah memiliki fitur user authentication termasuk halaman login, register dan reset password.

**Perhatian!** Apabila ketika menjalankan perintah `php artisan migrate` kamu mengalami error terkait key too long, itu artinya kamu harus sedikit menyesuaikan settingan Schema builder. Untuk melakukannya buka file `app/Providers/AppServiceProvider.php` lalu tambahkan kode `Schema::defaultStringLength(191);` pada method `boot()` seperti ini

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

Dengan perintah di atas kini kamu bisa melakukan registrasi dengan membuka di url <http://larashop.test/register>. Setelah kamu submit, kini kamu bisa mencoba untuk melakukan Authentication dengan cara login ke aplikasi pada url <http://larashop.test/login>.

Pahami bahwa yang dimaksud <http://larashop.test> Bagi pengguna XAMPP adalah:

- <http://localhost/larashop/public> atau
- <http://localhost/larashop> bila sudah dikonfigurasi untuk menghilangkan `public` di url

Buka <http://larashop.test> maka kamu akan mendapatkan dua menu baru di atas seperti ini tampilannya:

[LOGIN](#) [REGISTER](#)

# Laravel

[DOCUMENTATION](#) [LARACASTS](#) [NEWS](#) [FORGE](#) [GITHUB](#)

**Taraaa**, semudah itu untuk membuat user authentication di Laravel, ga sampe semenit langsung jadi. Tapi mungkin kamu bertanya bagaimana keajaiban ini terjadi? Tenang, ini bukanlah *magic*, untuk itu mari kita bahas lebih detail apa yang terjadi setelah kita menjalankan perintah `php artisan make:auth`. Supaya kita paham cara kerja Authenticationnya terutama jika kita ingin memodifikasi Authenticationnya.

Ingat, setiap fitur yang ada di Laravel sebetulnya hanyalah kombinasi antara MVC, model view controller. Apa yang dilakukan oleh perintah di atas hanyalah membuat komponen MVC tersebut untukmu. Perintah di atas akan **mengenerate** file-file / kode berikut ini

1. File views antara lain:

- `resources/views/layouts/app.blade.php`;
- `resources/views/auth/passwords/email.blade.php`;
- `resources/views/auth/passwords/reset.blade.php`;
- `resources/views/auth/login.blade.php`;
- `resources/views/auth/register.blade.php`;
- `resources/views/auth/home.blade.php`.

2. Sebuah controller yaitu `HomeController.php`

3. Kode routing untuk Authentication pada file `routes/web.php` yaitu

```
Auth::routes();
```

Apa saja fungsi-fungsi dari file yang digenerate tadi?

**File / Kode**

**Penjelasan**

File / Kode	Penjelasan
resources/views/layouts/app.blade.php	Merupakan file layout view yang dapat digunakan oleh view lainnya. Pada layout ini, terdapat sebuah kode untuk menghasilkan top navigation atau navigasi atas yang akan menampilkan link login dan register jika pengguna belum login. Navigasi ini akan berubah menjadi nama lengkap pengguna dan link logout jika pengguna sudah login ke aplikasi.
resources/views/auth/passwords/email.blade.php	Merupakan view yang digunakan untuk menampilkan form email saat forgot password.
resources/views/auth/login.blade.php	Merupakan view yang digunakan untuk menampilkan form login.
resources/views/auth/register.blade.php	Merupakan view yang digunakan untuk menampilkan form registrasi pengguna.
resources/views/auth/home.blade.php	Merupakan view yang digunakan sebagai contoh halaman khusus pengguna yang sudah login ke aplikasi.
Kode <code>Auth::routes();</code> pada file routes/web.php	Digunakan untuk menyediakan route user authentication antara lain: <code>/login, /register, /password/reset</code> .
app/Http/Controllers/HomeController.php	Berisi sebuah action <code>index</code> untuk menampilkan view home. Controller ini menggunakan middleware <code>auth</code> sehingga hanya bisa diakses oleh user yang sudah login ke aplikasi

Selain kode yang digenerate di atas, proses user authentication juga membutuhkan file-file yang sudah ada sejak pertama kali kita membuat project Laravel baru, yaitu:

File / kode	Penjelasan
app/User.php	Merupakan model user, di sini kita bisa mengubah konfigurasi dari model user aplikasi.
app/Http/Controllers/Auth/ForgotPasswordController.php	Merupakan controller untuk mengelola request berkaitan dengan lupa password.

File / kode	Penjelasan
app/Http/Controllers/Auth/LoginController.php	Merupakan controller untuk mengelola request berkaitan dengan login pengguna
app/Http/Controllers/Auth/RegisterController.php	Merupakan controller untuk mengelola request berkaitan dengan registrasi pengguna
app/Http/Controllers/Auth/ResetPasswordController.php	Merupakan controller untuk mengelola request berkaitan dengan reset password
xxxxxx_create_users_table.php	Migration untuk membuat tabel <b>users</b> dan strukturnya
xxxxx_create_password_resets_table.php	Migration untuk membuat tabel <b>password_resets</b> dan strukturnya.

## Menyesuaikan struktur table users

Apabila kita lihat struktur tabel **users** bawaan dari Laravel adalah seperti ini:

#	Name	Type
1	<b>id</b> 	int(10)
2	<b>name</b>	varchar(255)
3	<b>email</b> 	varchar(255)
4	<b>password</b>	varchar(255)
5	<b>remember_token</b>	varchar(100)
6	<b>created_at</b>	timestamp
7	<b>updated_at</b>	timestamp

Sementara itu table users sesuai desain database kita adalah sebagai berikut:

users	
+ id:	Integer {PK}
+ username:	String {U}
+ remember_token:	String
+ email:	String{U}
+ password:	String
+ name:	String
+ roles:	String
+ address:	String
+ phone:	String
+ avatar:	String
+ status:	Enum {ACTIVE   INACTIVE}
+ created_at:	timestamps
+ updated_at:	timestamps

Kita akan menyesuaikan tabel **users** agar sesuai dengan struktur tabel kita dengan migration. Untuk itu buat sebuah migration dengan perintah ini:

```
php artisan make:migration penyesuaian_table_users
```

Lalu buka file migration yang baru dibuat, cari di direktori **database/migrations**. Nama filenya adalah diakhiri **penyesuaian\_table\_users.php** Lalu isikan kode ini:

```
<?php

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

class PenyesuaianTableUsers extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string("username")->unique();
            $table->string("roles");
            $table->text("address");
            $table->string("phone");
            $table->string("avatar");
            $table->enum("status", ["ACTIVE", "INACTIVE"]);
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn("username");
            $table->dropColumn("roles");
            $table->dropColumn("address");
            $table->dropColumn("phone");
            $table->dropColumn("avatar");
            $table->dropColumn("status");
        });
    }
}
```

```

        } );
    }

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn("username");
        $table->dropColumn("roles");
        $table->dropColumn("address");
        $table->dropColumn("phone");
        $table->dropColumn("avatar");
        $table->dropColumn("status");
    });
}
}

```

Setelah itu jalankan perintah di terminal:

```
php artisan migrate
```

Jika sudah maka kini tabel `users` di database kita akan sesuai dengan desain database awal project kita.

#	Name	Type
1	<code>id</code>	int(10)
2	<code>name</code>	varchar(255)
3	<code>email</code>	varchar(255)
4	<code>password</code>	varchar(255)
5	<code>remember_token</code>	varchar(100)
6	<code>created_at</code>	timestamp
7	<code>updated_at</code>	timestamp
8	<code>username</code>	varchar(255)
9	<code>roles</code>	varchar(255)
10	<code>address</code>	text
11	<code>phone</code>	varchar(255)
12	<code>avatar</code>	varchar(255)
13	<code>status</code>	enum('ACTIVE', 'INACTIVE')

Lihat bagaimana kita telah berhasil mengubah struktur tabel di database tanpa menggunakan web gui phpmyadmin.

## Seeding user administrator

Kita akan menghilangkan fitur registrasi bawaan dari Laravel authentication untuk itu untuk membuat users baru kita tidak menggunakan fitur registrasi akan tetapi menggunakan fitur manage users.

Seperti kita lihat ada 3 role sesuai desain database kita yaitu "ADMIN", "STAFF" dan "CUSTOMER". Manage users ini hanya bisa diakses oleh users dengan role administrator.

Oleh karenanya kita perlu membuat akun administrator. Akun tersebut detailnya adalah seperti ini:

- email: administrator@larashop.test
- username: administrator
- password: larashop

Kita buat seeder dengan nama **AdministratorSeeder** dengan perintah:

```
php artisan make:seeder AdministratorSeeder
```

Maka file **AdministratorSeeder.php** akan dibuat di direktori **database/seeds**. Buka file tersebut dan kita buat akun administrator dengan kode ini:

```
<?php

use Illuminate\Database\Seeder;

class AdministratorSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $administrator = new \App\User;
        $administrator->username = "administrator";
        $administrator->name = "Site Administrator";
        $administrator->email = "administrator@larashop.test";
        $administrator->roles = json_encode(["ADMIN"]);
        $administrator->password = \Hash::make("larashop");

        $administrator->save();

        $this->command->info("User Admin berhasil diinsert");
    }
}
```

**Penjelasan:**

Kode di atas merupakan file seeder untuk menginsert data baru ke tabel `users`. Data tersebut merupakan data user administrator yang akan kita gunakan untuk login ke aplikasi. Pertama kita memanfaatkan model `User` dengan membuat instance baru `new \App\User` sebagai variabel `$administrator`. Lalu kita tambahkan properti-properti yang user admin tersebut meliputi `username` sampe `password`. Untuk mengisi field `roles` kita gunakan PHP native function `json_encode`, kenapa? karena kita ingin menyimpan tipe array ke dalam database sebagai string / teks, ingat field `roles` di desain database kita bertipe teks untuk menyimpan array roles. Artinya setiap user nantinya bisa memiliki beberapa role baik ADMIN, STAFF atau CUSTOMER secara bersamaan dalam array.

## Hashing

Selain itu juga kita menggunakan Laravel facade `\Hash` dan kita gunakan method `make` untuk menghashing password kita agar password tersebut di database tidak disimpan sebagai plain teks. Coba perhatikan dalam seeder di atas kita mengisikan password kita sebagai "larashop" dengan `\Hash::make`, maka nanti setelah kamu jalankan perintah `artisan migrate` kamu akan dapati string yang tidak bisa dibaca (misalnya seperti ini `$2y$10$CaHcu3RjHs2Yr1c.hgFOVeM77aJ2soN7JUBLyLyL1NZGMY2UqY3Vq`). Dengan begini password masing-masing user di aplikasi kita lebih aman seandainya akses database kita jatuh ke tangan hacker.

Jalankan perintah:

```
php artisan migrate
```

Setelah berhasil melakukan migration, coba buka <http://larashop.test/login> dan login menggunakan

- email: `administrator@larashop.test`
- password: `larashop`

Maka kamu akan masuk ke aplikasi sebagai administrator dan melihat tampilan ini:

The screenshot shows a web application interface. At the top left is the word "Laravel". At the top right is a dropdown menu labeled "Site Administrator". The main content area has a header "Dashboard". Below it is a message box containing the text "You are logged in!". The background is light gray.

Tampilan di atas merupakan homepage (<http://larashop.test/home>) hanya untuk user yang telah login. Halaman tersebut telah digenerate saat kita menjalankan perintah `php artisan make:auth`. Coba logout setelah itu akses kembali halaman tadi <http://larashop.test/home> maka kamu akan diredirect ke halaman login. Ini membuktikan bahwa fitur Authentication di aplikasi kita telah berjalan.

## Layout, Halaman dan Styling

Selanjutnya kita akan belajar untuk mengintegrasikan Bootstrap v4.1 ke dalam proyek kita. Dan kita menyesuaikan tampilan dari halaman-halaman yang digenerate oleh `make:auth`.

### Membuat layout aplikasi

Kita membutuhkan sebuah layout global yang nantinya bisa dipakai oleh view-view lainnya. Meskipun kita sudah memiliki satu layouts di `resources/views/layouts/app.blade.php` hasil generate dari perintah `artisan make:auth`, kita tidak menggunakan layout ini. Oleh karenanya mari kita buat layout dengan nama `global.blade.php` di direktori `resources/views/layouts`. Buka file layout tadi dan isikan kode berikut ini:

```
<!DOCTYPE html>
<!--[if IE 9]> <html class="ie9 no-js" lang="en"> <![endif]-->
<!--[if (gt IE 9)|(IE)]><!-->
<html class="no-js" lang="en">
<!--<![endif]-->

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Larashop @yield("title")</title>
    <link rel="stylesheet" href="{{asset('polished/polished.min.css')}}">
    <link rel="stylesheet" href="{{asset('polished/iconic/css/open-iconic-bootstrap.min.css')}}">

    <style>
        .grid-highlight {
            padding-top: 1rem;
            padding-bottom: 1rem;
            background-color: #5c6ac4;
            border: 1px solid #202e78;
            color: #fff;
        }

        hr {
            margin: 6rem 0;
        }

        hr+.display-3,
        hr+.display-2+.display-3 {
            margin-bottom: 2rem;
        }
    </style>
```

```

<script type="text/javascript">
    document.documentElement.className =
document.documentElement.className.replace('no-js', 'js') +
(document.implementation.hasFeature("http://www.w3.org/TR/SVG11/feature#BasicStructure", "1.1") ? ' svg' : ' no-svg');
</script>
</head>

<body>

    <nav class="navbar navbar-expand p-0">
        <a class="navbar-brand text-center col-xs-12 col-md-3 col-lg-2 mr-0"
href="index.html"> Larashop </a>
        <button class="btn btn-link d-block d-md-none" data-toggle="collapse"
data-target="#sidebar-nav" role="button" >
            <span class="oi oi-menu"></span>
        </button>

        <input class="border-dark bg-primary-darkest form-control d-none d-
md-block w-50 ml-3 mr-2" type="text" placeholder="Search" aria-
label="Search">
        <div class="dropdown d-none d-md-block">
            @if(\Auth::user())
                <button class="btn btn-link btn-link-primary dropdown-toggle"
id="navbar-dropdown" data-toggle="dropdown">
                    {{Auth::user()->name}}
                </button>
            @endif
            <div class="dropdown-menu dropdown-menu-right" id="navbar-
dropdown">
                <a href="#" class="dropdown-item">Profile</a>
                <a href="#" class="dropdown-item">Setting</a>
                <div class="dropdown-divider"></div>
                <li>
                    <form action="{{route("logout")}}" method="POST">
                        @csrf
                        <button class="dropdown-item" style="cursor:pointer">Sign
Out</button>
                    </form>
                </li>
            </div>
        </div>
    </nav>

    <div class="container-fluid h-100 p-0">
        <div style="min-height: 100%" class="flex-row d-flex align-items-
stretch m-0">
            <div class="polished-sidebar bg-light col-12 col-md-3 col-lg-2 p-0
collapse d-md-inline" id="sidebar-nav">

                <ul class="polished-sidebar-menu ml-0 pt-4 p-0 d-md-block">
                    <input class="border-dark form-control d-block d-md-none mb-
4" type="text" placeholder="Search" aria-label="Search" />
                    <li><a href="/home"><span class="oi oi-home"></span> Home</a>

```

```

</li>

        <div class="d-block d-md-none">
            <div class="dropdown-divider"></div>
            <li><a href="#"> Profile</a></li>
            <li><a href="#"> Setting</a></li>
            <li>
                <form action="{{route("logout")}}" method="POST">
                    @csrf
                    <button class="dropdown-item"
style="cursor:pointer">Sign Out</button>
                </form>
            </li>
        </div>
    </ul>
    <div class="pl-3 d-none d-md-block position-fixed"
style="bottom: 0px">
        <span class="oi oi-cog"></span> Setting
    </div>
</div>
<div class="col-lg-10 col-md-9 p-4">
    <div class="row ">
        <div class="col-md-12 pl-3 pt-2">
            <div class="pl-3">
                <h3>@yield("pageTitle")</h3>
                <br/>
            </div>
        </div>
    </div>
    @yield("content")

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

<script
src="https://code.jquery.com/jquery-3.3.1.min.js"
integrity="sha256-FgpCb/KJQ1LNf0u91ta32o/NMZxltwRo8QtmkMRdAu8="
crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"
integrity="sha384-
cs/chFZiN24E4KMATLdqvsezGxaGsi4hLG0z1Xwp5UZB1LY//20VyM2taTB4QvJ"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
integrity="sha384-
uefMccjFJAIV6A+rW+L4AHf99KvxDjWSu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm"
crossorigin="anonymous"></script>
</body>
</html>

```

## Penjelasan:

Perhatikan bahwa layouts ini menggunakan file css `polished.min.css` dan kita menggunakan helper `asset()` untuk mereferensikan file tersebut yang terletak di direktori `public/polished/polished.min.css` seperti ini:

```
<link rel="stylesheet" href="{{asset('polished/polished.min.css')}}">
```

Selain css tersebut kita juga memanggil satu file css lainnya untuk keperluan icon, yaitu file `public/polished/iconic/css/open-iconic-bootstrap.min.css`

```
<link rel="stylesheet" href="{{asset('polished/iconic/css/open-iconic-bootstrap.min.css')}}">
```

Helper `asset()` akan menghasilkan URL untuk mengakses file-file yang terdapat di direktori `public` sehingga misalnya ada file `public/images/logo.png` kita bisa mereferensikan di file view kita dengan `asset("images/logo.png")`

## Link Logout

Route authentication yang dibuat dengan `artisan make:auth` adalah `http://larashop.test/logout`, tetapi route tersebut hanya menerima method POST. Sehingga kita tidak bisa melakukan logout hanya dengan mengakses url tadi di browser. Akan tetapi kita harus membuat form dengan method POST dan mengirim data kosong ke url tadi.

Oleh karenanya, pada layouts `global.blade.php` kita menggunakan kode berikut untuk link logoutnya.

```
<form action="{{route("logout")}}" method="POST">
    @csrf
    <button class="dropdown-item" style="cursor:pointer">Sign Out</button>
</form>
```

Form tersebut kita letakan di dropdown menu kanan.

## Menggunakan template bootstrap

Tapi kamu mungkin menyadari bahwa kedua file css di atas tidak ada di direktori `public` project kita, yup betul. Oleh karena itu kita perlu copy terlebih dahulu folder `polished` yang ada di file assets buku ini ke direktori `public` project larashop kita, sehingga kita memiliki direktori `polished` dan seisinya di `public/polished`.

Atau kamu bisa mendownloadnya langsung dari github <https://github.com/azamuddin/polished-template> lalu rename folder `dist` di hasil download kita menjadi `polished` dan letakkan di direktori `public` project larashop kita.

Polished template merupakan bootstrap v4 admin template open source dan gratis yang dikembangkan oleh Muhammad Azamuddin.

Kini, kita memiliki sebuah layout baru yang siap digunakan di `resources/views/layouts/global.blade.php` selanjutnya di view-view lainnya kita bisa menggunakan layout ini dengan cara:

```
@extends("layouts.global")
```

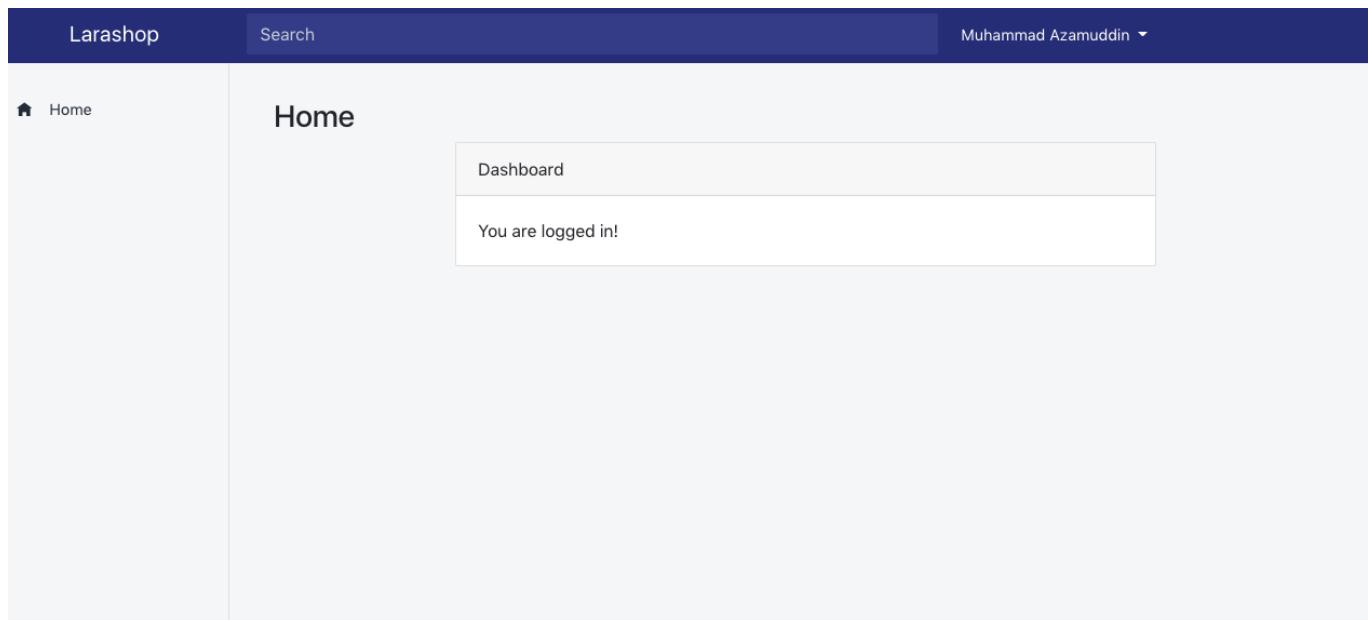
Untuk mencobanya silahkan buka file `resources/views/home.blade.php` dan ganti baris ini:

```
@extends("layouts.app")
```

Menjadi

```
@extends("layouts.global")
```

Dengan begitu kini view `home` tidak lagi menggunakan layout `app` melainkan menggunakan layout `global`. Coba kamu login ke aplikasi kita, dan lihat perubahan terhadap halaman home. Seharusnya kamu akan melihat tampilan seperti ini:



Selain itu di layout `global` kita juga menyediakan dua slot yaitu untuk `title` dan untuk `content` hal ini kita tulis di file `global.blade.php`, perhatikan line:

```
@yield("title")
```

dan

```
@yield("content")
```

Itulah kenapa ketika kita mengakses [http://larashop.test/home](http://larashop.test/home) kita bisa melihat tampilan "You are logged in" itu karena di view tersebut menggunakan slot `@content` dengan kode berikut:

```
@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">Dashboard</div>

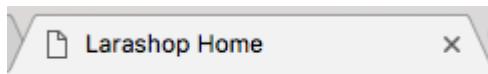
                <div class="card-body">
                    @if (session('status'))
                        <div class="alert alert-success" role="alert">
                            {{ session('status') }}
                        </div>
                    @endif

                    You are logged in!
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

di view `home` kita belum menggunakan slot `title` yang disediakan oleh layout `global`. Mari gunakan slot tersebut dengan menambahkan kode berikut setelah kode `@extends("layouts.global")`

```
@section("title") Home @endsection
```

Maka jika kamu refresh `http://larashop.test/home` kamu akan melihat di title bar dengan tulisan "Larashop Home" seperti ini:



Itu karena slot title pada layouts diletakkan di html tag title seperti ini:

```
<title>Larashop @yield("title")</title>
```

Selain tab title, judul halaman juga akan mengikuti seperti yang kita lihat di screenshot.

## Menyesuaikan halaman login

Kita ingin menyesuaikan tampilan login bawaan dari perintah `artisan make:auth`. Ada dua file yang harus kita ubah, yang pertama adalah layout `app.blade.php` yang ada di `resources/views/layouts`. Karena kita ingin tampilan kita konsisten menggunakan polished template, kita hapus semua kode yang ada lalu ganti dengan kode berikut:

File `resources/views/layouts/app.blade.php`

```
<!DOCTYPE html>
<!--[if IE 9]> <html class="ie9 no-js" lang="en"> <![endif]-->
<!--[if (gt IE 9)|(IE)]><!-->
<html class="no-js" lang="en">
<!--<![endif]-->

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Larashop @yield("title")</title>
    <link rel="stylesheet" href="{{asset('polished/polished.min.css')}}">
    <link rel="stylesheet" href="{{asset('polished/iconic/css/open-iconic-bootstrap.min.css')}}">

    <style>
        .grid-highlight {
            padding-top: 1rem;
            padding-bottom: 1rem;
            background-color: #5c6ac4;
            border: 1px solid #202e78;
            color: #fff;
        }

        hr {
            margin: 6rem 0;
        }

        hr+.display-3,
        hr+.display-2+.display-3 {
            margin-bottom: 2rem;
        }
    </style>
    <script type="text/javascript">
        document.documentElement.className =
        document.documentElement.className.replace('no-js', 'js') +
        (document.implementation.hasFeature("http://www.w3.org/TR/SVG11/feature#BasicStructure", "1.1") ? ' svg' : ' no-svg');
    </script>
</head>

<body>
```

```

<nav class="navbar navbar-expand p-0">
    <a class="navbar-brand text-center col-xs-12 col-md-3 col-lg-2 mr-0"
    href="index.html"> Larashop </a>
        <button class="btn btn-link d-block d-md-none" data-toggle="collapse"
    data-target="#sidebar-nav" role="button" >
            <span class="oi oi-menu"></span>
        </button>
    </nav>

    <div class="container-fluid h-100 p-0">
        <div style="min-height: 100%" class="flex-row d-flex align-items-stretch m-0">
            <div class="col-lg-12 col-md-12 p-4">
                @yield("content")
            </div>
        </div>
    </div>

    <script
    src="https://code.jquery.com/jquery-3.3.1.min.js"
    integrity="sha256-FgpCb/KJQ1LNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
    crossorigin="anonymous"></script>
    <script
    src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"
    integrity="sha384-"
    cs/chFZiN24E4KMATLdqvsezGxaGsi4hLG0z1Xwp5UZB1LY//20VyM2taTB4QvJ"
    crossorigin="anonymous"></script>
    <script
    src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
    integrity="sha384-"
    uefMccjFJAIV6A+rW+L4AHf99KvxDjWSu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm"
    crossorigin="anonymous"></script>
</body>

</html>

```

### Penjelasan:

Pada dasarnya isi dari file `app.blade.php` yang baru adalah mirip seperti yang ada di file `layout/global.blade.php` yaitu kita menginsert file-file css polished template dan javascript yang dibutuhkan oleh template tersebut. Bedanya tidak ada kode untuk sidebar dan top menu search.

Setelah itu file berikutnya yang kita ubah adalah file `login.blade.php` yaitu di `resources/views/auth/login.blade.php`. Jadikan kodennya terlihat seperti ini:

```

@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-4"></div>

```

```

<div class="col-md-4">
    <div class="card bg-white border">
        {{-- <div class="card-header bg-transparent border-0">{{
__('Login') }}</div> --}}
    <div class="card-body">
        <form method="POST" action="{{ route('login') }}" aria-
label="{{ __('Login') }}">
            @csrf
            <div class="form-group row">
                <div class="col-md-12">
                    <label for="email" class="col-sm-12 col-
form-label pl-0">{{ __('E-Mail Address') }}</label>
                    <br>
                    <input id="email" type="email" class="form-
control{{ $errors->has('email') ? ' is-invalid' : '' }}" name="email"
value="{{ old('email') }}" required autofocus>

                    @if ($errors->has('email'))
                        <span class="invalid-feedback">
                            <strong>{{ $errors->first('email') }}</strong>
                        </span>
                    @endif
                </div>
            </div>

            <div class="form-group row">
                <div class="col-md-12">
                    <label for="password" class="col-md-4 col-
form-label text-md-left pl-0">{{ __('Password') }}</label>
                    <input id="password" type="password"
class="form-control{{ $errors->has('password') ? ' is-invalid' : '' }}"
name="password" required>

                    @if ($errors->has('password'))
                        <span class="invalid-feedback">
                            <strong>{{ $errors-
>first('password') }}</strong>
                        </span>
                    @endif
                </div>
            </div>

            <div class="form-group row">
                <div class="col-md-12">
                    <div class="checkbox">
                        <input type="checkbox" id="remember"
name="remember" {{ old('remember') ? 'checked' : '' }}> <label
for="remember">{{ __('Remember Me') }}</label>
                </div>
            </div>
        </form>
    </div>
</div>

```

```

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

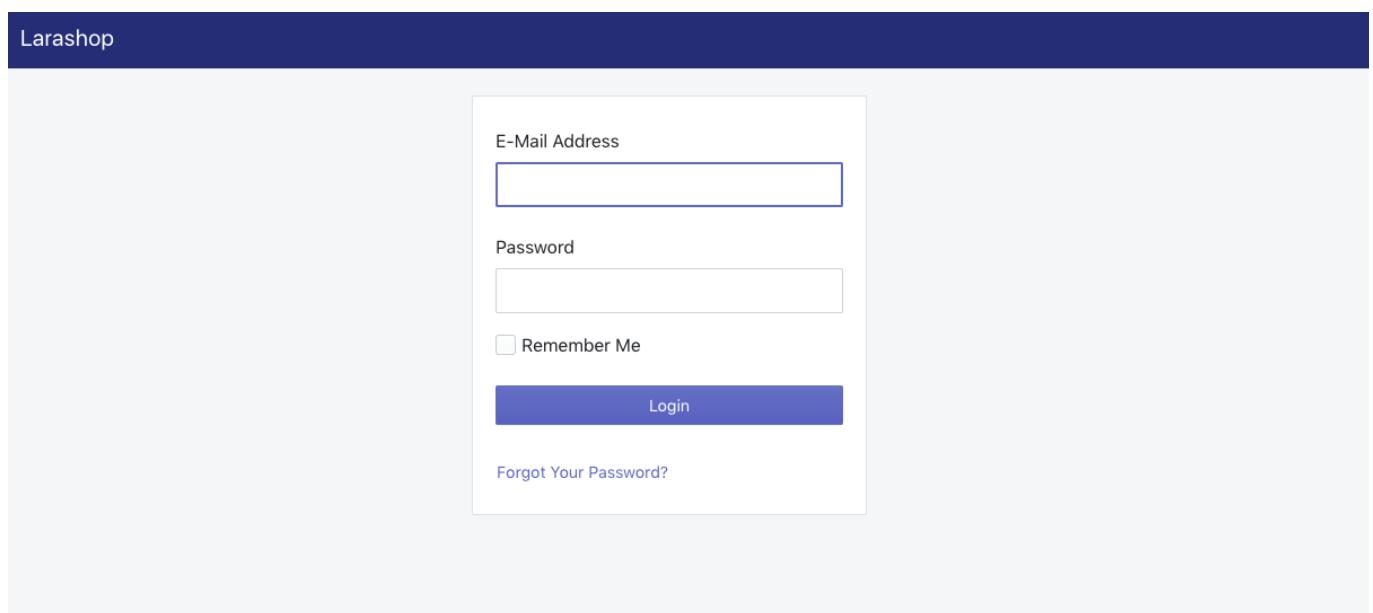
<div class="form-group row mb-0">
    <div class="col-md-12">
        <button type="submit" class="btn-block btn
btn-primary">
            {{ __('Login') }}
        </button>
        <br>

        <a class="btn btn-link pl-0" href="{{
route('password.request') }}">
            {{ __('Forgot Your Password?') }}
        </a>
    </div>
</div>
</form>
</div>
</div>
</div>
</div>
</div>
@endsection

```

#### Penjelasan:

Kita tidak banyak mengubah kode yang berkaitan dengan laravel, hanya menyesuaikan kode html saja agar tampilan login menjadi seperti ini:



Menghapus fitur registrasi

Seperti yang telah kita bahas di gambaran studi kasus bahwa apa yang akan kita buat di studi kasus I ini adalah manajemen toko online sehingga fitur registrasi tidak relevan, untuk itu kita akan hapus route tersebut.

Buka file `routes/web.php` lalu kita override route `/register` yang dihasilkan oleh `Auth::routes()`. Caranya, letakkan kode berikut ini setelah `Auth::routes()`

```
Route::match(["GET", "POST"], "/register", function(){
    return redirect("/login");
})->name("register");
```

Kini jika route `register` diakses maka akan diredirect ke `login`. Silahkan coba buka <http://larashop.test/register>.

Menjadikan halaman awal sebagai login page

Untuk menjadikan halaman awal sebagai halaman login cukup ganti kode berikut ini di file `routes/web.php`

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

Menjadi seperti ini:

```
Route::get('/', function(){
    return view('auth.login');
});
```

## Manage Users

CRUD User

### Membuat user resource

Untuk kepentingan fitur CRUD ini kita akan menggunakan controller dan route resource. Pertama kita buat terlebih dahulu `UserController`

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

Setelah kamu jalankan perintah di atas, kini kita memiliki sebuah file controller pada `app/Http/Controllers/UserController.php`.

Setelah itu mari kita hubungkan controller resource tadi ke dunia luar dengan menambahkan route resource di `routes/web.php` seperti ini:

```
Route::resource("users", "UserController");
```

Dengan begitu kini kita siap untuk mengembangkan fitur-fitur CRUD untuk user. Ingat dengan resource route seperti di atas kita memiliki pemetaan route dan controller seperti berikut:

method & path	keterangan	controller action	named route
GET /users/index	untuk list users	index	users.index
GET /users/create	untuk form create user	create	users.create
POST /users	untuk mengirim input dari form create	store	users.store
GET /users/{id}/edit	untuk form edit user	edit	users.edit
PUT /users/{id}	untuk mengirim input dari form edit user	update	users.update
GET /users/{id}	untuk menampilkan user dengan id tertentu	show	users.show
DELETE /users/{id}	untuk menghapus user dengan id tertentu	destroy	users.destroy

## Fitur create user

Route yang akan dipakai untuk menampilkan form create user adalah `/users/create`. Untuk itu kita akan mengubah kode controller action `create` agar menampilkan view `users.create`. Buka file `UserController` lalu di action `create()` jadikan seperti ini:

```
public function create(){
    return view("users.create");
}
```

Kini jika `http://larashop.test/users/create` diakses maka aplikasi akan berusaha menampilkan file view di `resources/views/users/create.blade.php`. Sayangnya file tersebut belum ada, supaya aplikasi tidak error mari kita buat filenya.

## Buat form untuk create users

Buat folder `users` di `resources/views` lalu sebuah file dengan nama `create.blade.php` di dalam folder `users` tadi. Sehingga kita memiliki sebuah file view di `resources/views/users/create.blade.php`. View ini akan kita gunakan untuk menaruh kode html form untuk menginput user baru.

Buka file `create.blade.php` tadi dan masukan kerangka berikut ini:

```
@extends("layouts.global")  
  
@section("title") Create User @endsection  
  
@section("content")  
  
@endsection
```

Penjelasan: View ini menggunakan layout global yang sebelumnya kita buat

`@extends('layouts.extend')`. Kita akan mengisi slot "title" dengan konten "Create User" kemudian kode form nantinya akan kita letakan di antara kode `@section("content")` dan `@endsection`

Selanjutnya mari kita isi dengan kerangka kode html form.

```
<form action="{{route('users.store')}}" method="POST">  
    @csrf  
</form>
```

Form ini menggunakan action `http://larashop.test/users` dengan method POST yang telah digenerate oleh route resource. Route tersebut juga memiliki nama bawaan (named route) yaitu `users.store` sehingga di kode di atas kita menggunakan helper `route()` untuk menggenerate full url. Sehingga bisa kita simpulkan bahwa

```
route('users.store') == 'http://larashop.test/users'
```

Route `users.store` hanya mengizinkan metode "POST" sehingga set form ini menggunakan method "POST".

Kemudian yang perlu diperhatikan adalah kita menggunakan blade helper `@csrf`. Helper ini berfungsi untuk menggenerate token csrf seperti ini:

```
<input  
    type="hidden"  
    name="_token"  
    value="HPoyywkM2LR0RM1L4qHfe9TwPAi75IgvXeL9PX4b">
```

Kode input hidden ini berfungsi untuk mengecek bahwa request yang dikirimkan adalah request yang valid. Jika kita tidak menyertakan helper `@csrf` pada form, maka kita akan mendapatkan error saat melakukan submit form.

Kemudian kita juga set form attribute `enctype` bernilai "multipart/form-data" karena kita akan mengunggah (mengupload) file dari form ini.

```
<form enctype="multipart/form-data" ...>
```

Kode untuk form secara keseluruhan menjadi seperti ini:

```
<form enctype="multipart/form-data" class="bg-white shadow-sm p-3" action="{!! route('users.store') !!}" method="POST">
```

Setelah memahami dasar dari penggunaan form, selanjutnya kita isikan kode html biasa untuk masing-masing field, seperti username, email, name, roles, password sehingga hasil akhir dari kode view `create.blade.php` akan menjadi seperti ini:

File `resources/views/users/create.blade.php`

```
@extends("layouts.global")  
  
@section("title") Create New User @endsection  
  
@section("content")  
  
    <div class="col-md-8">  
  
        <form  
            enctype="multipart/form-data"  
            class="bg-white shadow-sm p-3"  
            action="{!! route('users.store') !!}"  
            method="POST">  
  
            @csrf  
  
            <label for="name">Name</label>  
            <input  
                class="form-control"  
                placeholder="Full Name"  
                type="text"  
                name="name"  
                id="name"/>  
            <br>  
  
            <label for="username">Username</label>  
            <input  
                class="form-control"  
                placeholder="username"  
                type="text"  
                name="username"  
                id="username"/>  
            <br>  
  
            <label for="">Roles</label>
```

```
<br>
<input
  type="checkbox"
  name="roles[]"
  id="ADMIN"
  value="ADMIN">
  <label for="ADMIN">Administrator</label>

<input
  type="checkbox"
  name="roles[]"
  id="STAFF"
  value="STAFF">
  <label for="STAFF">Staff</label>

<input
  type="checkbox"
  name="roles[]"
  id="CUSTOMER"
  value="CUSTOMER">
  <label for="CUSTOMER">Customer</label>
<br>

<br>
<label for="phone">Phone number</label>
<br>
<input
  type="text"
  name="phone"
  class="form-control">

<br>
<label for="address">Address</label>
<textarea
  name="address"
  id="address"
  class="form-control"></textarea>

<br>
<label for="avatar">Avatar image</label>
<br>
<input
  id="avatar"
  name="avatar"
  type="file"
  class="form-control">

<hr class="my-3">

<label for="email">Email</label>
<input
  class="form-control"
  placeholder="user@mail.com"
  type="text">
```

```
    name="email"
    id="email"/>
<br>

<label for="password">Password</label>
<input
    class="form-control"
    placeholder="password"
    type="password"
    name="password"
    id="password"/>
<br>

<label for="password_confirmation">Password Confirmation</label>
<input
    class="form-control"
    placeholder="password confirmation"
    type="password"
    name="password_confirmation"
    id="password_confirmation"/>
<br>

<input
    class="btn btn-primary"
    type="submit"
    value="Save"/>
</form>
</div>

@endsection
```

Buat kamu bertanya-tanya tentang class yang ada semisal `col-md-8`, `btn btn-primary`, `form-control` dll itu hanyalah class-class yang disediakan oleh bootstrap css agar input-input form langsung menggunakan styling dari template polished.

Silahkan kamu buka <http://larashop.test/users/create> maka kamu akan melihat tampilan form yang cantik seperti ini

## Create New User

Name

Full Name

Username

username

Roles

Administrator  Staff  Customer

Phone number

Address

### Menangkap request dan menyimpan ke database

Form untuk create user telah siap untuk mengirimkan data ke route POST

<http://larashop.test/users> yang akan mengarahkannya ke action `store()` pada `UserController`. Langkah selanjutnya adalah kita akan menangkap data yang dikirimkan oleh form create user tadi dan menyimpan datanya ke tabel `users`. Untuk itu kita perlu menaruh kode logika untuk melakukannya di `UserController@store` (baca: action `store()` pada `UserController`);

Pertama kita perlu buat instance dari model `User` dengan kode ini:

```
$new_user = new \App\User;
```

Lalu kita set properti dari user dengan nilai yang berasal dari data yang dikirim oleh form create user seperti ini:

```
$new_user->name = $request->get('name');  
$new_user->username = $request->get('username');  
$new_user->roles = json_encode($request->get('roles'));
```

```
$new_user->name = $request->get('name');
$new_user->address = $request->get('address');
$new_user->phone = $request->get('phone');
$new_user->email = $request->get('email');
$new_user->password = \Hash::make($request->get('password'));
```

### Menghandle file upload

Setelah itu kita handle untuk upload file avatar dari user seperti ini:

```
if($request->file('avatar')){
    $file = $request->file('avatar')->store('avatars', 'public');

    $new_user->avatar = $file;
}
```

Kode di atas mengecek apakah request memiliki file dengan nama **avatar**, jika ada maka simpan file tersebut menggunakan method **\$request->file()->store()**.

```
$request->file('avatar');
```

Mendapatkan file avatar yang dikirimkan oleh input bertipe file yang attribute namanya bernilai 'avatar', kita memiliki input tersebut di form yang kita buat yaitu input ini:

```
<input id="avatar" name="avatar" type="file" class="form-control">
```

Setelah itu kita simpan di folder **avatars** yang kita masukan sebagai parameter pertama method **store()** dan kita set visibilitynya menjadi **public** agar bisa diakses oleh siapa saja menggunakan url.

```
store('avatars', 'public');
```

Method **store()** akan menghasilkan path dari file yang kita simpan. Sehingga kode di atas akan menghasilkan string path di mana lokasi file berada, kita memerlukan string tersebut untuk disimpan sebagai nilai 'avatar' di kolom tabel users, seperti ini:

```
$file = $request->file('avatar')->store('avatars', 'public')
$new_user->avatar = $file
```

Maka itu sama saja nilai dari **\$new\_user->avatar** adalah = '**avatars/randomstring.png**' misalnya. **avatars** merupakan direktori yang kita masukan sebagai argument pertama di method 'store', sementara

"randomstring" adalah autogenerate nama file oleh Laravel karena kita menggunakan `store()`. File ini akan disimpan di direktori `storage/app/public`.

Misal kita upload sebuah file `.png` dengan method `store('avatars', 'public')` seperti di atas, maka akan ada file baru contoh:

- `storage/app/public/avatars/xadfasfasfjaslkfasj.png`, extension dari file mengikuti file yang diupload, `avatars` adalah direktori yang kita masukan di method `store('avatars', ...)` (jika belum ada akan otomatis dibuat) sementara itu `xadfasfasfjaslkfasj` merupakan randomstring yang digenerate. Dan karena kita menggunakan visibility "public" di method `store(..., 'public')` maka filenya tersimpan di `app/public/{direktori}/{namafile}` bukan `app/{direktori}{namafile}`

Kemudian jika kita mengupload lagi file lain, maka akan muncul lagi file di lokasi yang sama dengan nama file random baru, misalnya:

- `storage/app/public/avatars/sfaresdaflajkrearan.png`

Nah ada satu langkah lagi yang harus kita lakukan, file ini sudah ada di server kita, akan tetapi belum bisa diakses menggunakan url. Padahal seharusnya bisa diakses seperti ini:

`http://larashop.test/storage/avatars/sfaresdaflajkrearan.png`

Sayangnya file-file yang ada di folder `storage` memang tidak bisa diakses menggunakan URL meskipun di bawah `storage/app/public`, loh? padahal file upload akan otomatis masuk ke folder storage. Sementara itu file-file yang bisa diakses oleh URL harus diletakkan di dalam folder `public` aplikasi kita. Untuk itu kita perlu membuat direktori virtual di `public/storage` yang isinya sama persis (mirroring) dengan direktori `storage/app/public`. Dan kabar gembiranya, Laravel menyediakan perintah untuk memudahkan proses itu, caranya kamu jalankan perintah ini:

```
php artisan storage:link
```

Maka kini akan terbentu virtual folder di `public/storage` yang isinya mirror dari `storage/app/public` dan file-file di sini bisa diakses menggunakan url seperti ini:

Lokasi file	Mirror file
<code>storage/app/public/avatars/abc.png</code>	<code>public/storage/avatars/abc.png</code>
<code>storage/app/public/def.pdf</code>	<code>public/storage/def.pdf</code>
<code>storage/app/public/covers/book1.jpg</code>	<code>public/storage/covers/book1.jpg</code>
Mirror file	URL Web
<code>public/storage/avatars/abc.png</code>	<code>http://larashop.test/storage/avatars/abc.png</code>
<code>public/storage/def.pdf</code>	<code>http://larashop.test/storage/def.pdf</code>
<code>public/storage/covers/book1.jpg</code>	<code>http://larashop.test/storage/covers/book1.jpg</code>

Kita juga bisa menggunakan `asset()` helper, `asset()` helper merupakan cara yang bisa kita pakai untuk menghasilkan URL dari sebuah file tanpa perlu mengetik `http://larashopt.test` secara manual, cara ini lebih direkomendasikan digunakan pada saat menampilkan file di view, misalnya seperti ini:

```

```

Kode di atas sama saja seperti kita tuliskan:

```

```

#### Menyimpan user ke database

Kembali lagi ke `UserController@store`, selanjutnya kita simpan model User baru tadi ke database dengan method `save()` seperti ini:

```
$new_user->save();
```

Setelah berhasil menyimpan kita ingin arahkan user kembali ke form create, akan tetapi kini kita ingin memberikan pesan bahwa status dari operasi penyimpanan berhasil. Kita akan memanfaatkan helper `redirect()` untuk melakukan redirect dan method `with()` untuk memberikan pesan seperti ini:

```
return redirect()->route('users.create')->with('status', 'User successfully created.');
```

Karena kita akan menggunakan `named route` yaitu `users.create` maka kita gunakan fungsi `redirect()->route()` kemudian kita berikan pesan dengan fungsi `with('status', 'User successfully created.')`

Seperti halnya `named route users.store` yang mereferensikan route dengan method POST ke `http://larashop.test/users`, `named route users.create` juga mereferensikan ke sebuah url yaitu `http://larashop.test/users/create`.

Selain dengan `named route` kita juga bisa melakukan redirect menggunakan url seperti ini:

```
return redirect('users/create')->with(...dst);
```

Itu sama saja seperti

```
return redirect()->route('users.create')->with(...dst);
```

Okai, jadi hasil akhir dari action `store()` di `UserController` akan menjadi seperti ini:

```
public function store(Request $request)
{
    $new_user = new \App\User;
    $new_user->name = $request->get('name');
    $new_user->username = $request->get('username');
    $new_user->roles = json_encode($request->get('roles'));
    $new_user->address = $request->get('address');
    $new_user->phone = $request->get('phone');
    $new_user->email = $request->get('email');
    $new_user->password = \Hash::make($request->get('password'));

    if($request->file('avatar')){
        $file = $request->file('avatar')->store('avatars', 'public');
        $new_user->avatar = $file;
    }

    $new_user->save();
    return redirect()->route('users.create')->with('status', 'User successfully created');
}
```

Kita belum melakukan validasi dari input yang disubmit oleh user, kita akan melakukannya nanti di poin tersendiri.

#### Membaca session / flash message

Fungsi create user kita sekarang sudah berhasil. Tapi kita perlu membaca pesan bahwa pembuatan user baru berhasil. Untuk itu kita kembali buka file view `users/create.blade.php` dan tambahkan kode berikut ini di atas form

```
@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif
```

Kode tersebut mengecek apakah ada session variabel dengan nama `status`, jika ada tampilkan sebagai alert.

File view `users/create.blade.php` kita terlihat seperti ini:

```
@extends("layouts.global")
@section("title") Create New User @endsection
```

```
@section("content")  
  
<div class="col-md-8">  
  
    @if(session('status'))  
        <div class="alert alert-success">  
            {{session('status')}}  
        </div>  
    @endif  
  
    <form  
        enctype="multipart/form-data"  
        class="bg-white shadow-sm p-3"  
        action="{{route('users.store')}}"  
        method="POST">  
  
        @csrf  
  
        <label for="name">Name</label>  
        <input  
            class="form-control"  
            placeholder="Full Name"  
            type="text"  
            name="name"  
            id="name"/>  
        <br>  
  
        <label for="username">Username</label>  
        <input  
            class="form-control"  
            placeholder="username"  
            type="text"  
            name="username"  
            id="username"/>  
        <br>  
  
        <label for="">Roles</label>  
        <br>  
        <input  
            type="checkbox"  
            name="roles[]"  
            id="ADMIN" value="ADMIN">  
            <label for="ADMIN">Administrator</label>  
  
        <input  
            type="checkbox"  
            name="roles[]"  
            id="STAFF" value="STAFF">  
            <label for="STAFF">Staff</label>  
  
        <input  
            type="checkbox"  
            name="roles[]"  
            id="CUSTOMER" value="CUSTOMER">
```

```
<label for="CUSTOMER">Customer</label>
<br>

<br>
<label for="phone">Phone number</label>
<br>
<input
  type="text"
  name="phone"
  class="form-control">

<br>
<label for="address">Address</label>
<textarea
  name="address"
  id="address"
  class="form-control"></textarea>

<br>
<label for="avatar">Avatar image</label>
<br>
<input
  id="avatar"
  name="avatar"
  type="file"
  class="form-control">

<hr
  class="my-3">

<label for="email">Email</label>
<input
  class="form-control"
  placeholder="user@mail.com"
  type="text"
  name="email"
  id="email"/>
<br>

<label for="password">Password</label>
<input
  class="form-control"
  placeholder="password"
  type="password"
  name="password"
  id="password"/>
<br>

<label for="password_confirmation">Password Confirmation</label>
<input
  class="form-control"
  placeholder="password confirmation"
  type="password"
```

```
    name="password_confirmation"
    id="password_confirmation"/>
<br>

<input
    class="btn btn-primary"
    type="submit"
    value="Save"/>
</form>
</div>

@endsection
```

Kini fitur create user sudah selesai. Selanjutnya mari kita buat fitur untuk menampilkan daftar user yang ada.

## Fitur list user

Untuk membuat fitur list user kita perlu pahami bahwa kita akan menggunakan route bermethod GET <http://larashop.test/users>, route itu menggunakan `UserController@index`.

### Mendapatkan users dari database

Maka mari kita buka file `UserController.php`, pertama kita perlu mendapatkan data user dari database menggunakan model `User` dan tampilkan per halaman 10 users dengan method `paginate()` seperti ini:

```
$users = \App\User::paginate(10);
```

Setelah itu cukup kita lempar view `users.index` dengan data users yang berisi daftar users seperti ini:

```
return view('users.index', ['users' => $users]);
```

Sehingga action `index` di `UserController` kita terlihat seperti ini:

```
public function index()
{
    $users = \App\User::paginate(10);

    return view('users.index', ['users' => $users]);
}
```

### Membuat view untuk list users

Selanjutnya kita perlu membuat view untuk menampilkan daftar user. View tersebut kita buat sebagai `index.blade.php` di direktori `resources/views/users`. Silahkan buat file tersebut.

Seperti sebelumnya kita isi terlebih dahulu dengan kerangka view seperti ini:

File `resources/views/users/index.blade.php`

```
@extends("layouts.global")  
  
@section("title") Users list @endsection  
  
@section("content")  
  
Daftar user di sini  
  
@endsection
```

Tidak perlu dijelaskan lagi ya? karena kita pernah melihat kerangka ini sebelumnya. Nah kita akan menampilkan daftar user di dalam section content.

Ingat bahwa view ini memiliki variabel `$users` yang dilempar dari `UsersController@index` dan bernilai array atau daftar user. Untuk menampilkannya kita gunakan Blade helper `@foreach` loop. Seperti ini:

```
@foreach($users as $user)  
- {{$user->email}} <br/>  
@endforeach
```

Coba kamu letakkan kode di atas di dalam section content seperti ini:

```
@section("content")  
  
@foreach($users as $user)  
- {{$user->email}} <br/>  
@endforeach  
  
@endsection
```

Setelah itu coba kamu buka <http://larashop.test/users> maka kamu akan melihat daftar email user yang ada di database, misalnya seperti ini:

## Users List

- administrator@larashop.test
- mas.azamuddin@gmail.com
- azamuddin@live.com
- user1@gmail.com

Kamu juga boleh simulasikan untuk menambah user baru menggunakan form yang kita buat sebelumnya lalu lihat kembali halaman user list untuk melihat user yang baru dibuat.

Tampilan tersebut masih sangat sederhana, mari kita perbaiki tampilan daftar user ini. Buka kembali file view `users/index.blade.php` dan jadikan kodennya seperti in:

```
@extends("layouts.global")

@section("title") Users list @endsection

@section("content")



| <b>Name</b>      | <b>Username</b>      | <b>Email</b>      | <b>Avatar</b>                                                                             | <b>Action</b>                               |
|------------------|----------------------|-------------------|-------------------------------------------------------------------------------------------|---------------------------------------------|
| {{\$user->name}} | {{\$user->username}} | {{\$user->email}} | @if(\$user->avatar)  @else N/A @endif | <a href="#">Edit</a> <a href="#">Delete</a> |


```

```

        </td>
        <td>
            [TODO: actions]
        </td>
    </tr>
@endforeach
</tbody>
</table>

@endsection

```

Dengan begitu tampilan user list kita sekarang seperti ini:

Name	Username	Email	Avatar	Action
Site Administrator	administrator	administrator@larashop.test	N/A	[TODO: actions]
Muhammad Azamuddin	azamuddin91	azamuddin@live.com	N/A	[TODO: actions]
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	[TODO: actions]
Hana Humaira	hana	hana@humaira.com	N/A	[TODO: actions]
User Empat	user4	user4@gmail.com	N/A	[TODO: actions]
User Lima	user5	user5@gmail.com	N/A	[TODO: actions]
User Enam	user6	user6@gmail.com	N/A	[TODO: actions]
Ridwan Mutaffaq	ridwan	ridwan@gmail.com	N/A	[TODO: actions]
Test	test	mas.azamuddin@gmail.com		[TODO: actions]

Khusus untuk avatar kita melakukan pengecekan, jika user telah memiliki avatar image maka kita tampilkan image avatarnya, jika belum punya avatar, maka tampilkan 'N/A', itu kita lakukan menggunakan kode ini:

```

@if($user->avatar)
    
@else
    N/A
@endif

```

## Fitur edit user

Fitur form edit users akan menggunakan url <http://larashop.test/users/{id}/edit> untuk menampilkan form, dan url <http://larashop.test/users/{id}> dengan method PUT untuk mengirim data edit.

Pertama kita buat sebuah tombol di tampilan list user yang sudah kita buat. Buka kembali file [users/index.blade.php](#) yang baru saja kita selesaikan. Tambahkan kode berikut ini:

```
<a class="btn btn-info text-white btn-sm" href="{{route('users.edit', ['id'=>$user->id])}}>Edit</a>
```

Menggantikan baris ini

```
[TODO: actions]
```

Sehingga jika kamu buka daftar user kamu akan melihat tombol edit di masing-masing baris, seperti ini:

Name	Username	Email	Avatar	Action
Site Administrator	administrator	administrator@larashop.test	N/A	<button>Edit</button>
Muhammad Azamuddin	azamuddin91	azamuddin@live.com	N/A	<button>Edit</button>
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	<button>Edit</button>
Hana Humaira	hana	hana@humaira.com	N/A	<button>Edit</button>
User Empat	user4	user4@gmail.com	N/A	<button>Edit</button>
User Lima	user5	user5@gmail.com	N/A	<button>Edit</button>
User Enam	user6	user6@gmail.com	N/A	<button>Edit</button>
Ridwan Mutaffaq	ridwan	ridwan@gmail.com	N/A	<button>Edit</button>
Test	test	mas.azamuddin@gmail.com		<button>Edit</button>

Kode link yang baru saja kita tambahkan tadi lah yang berfungsi untuk membuat tombol itu, kalo diperhatikan attribute href kita menggunakan helper `route()` seperti ini:

```
route('users.edit', ['id'=>$user->id])
```

Kode di atas menghasilkan string seperti ini `http://larashop.test/users/{id}/edit`, di mana `{id}` nilainya berasal dari `$user->id` yaitu ID dari masing-masing user. `users.edit` merupakan named route dari `/users/{id}/edit`, telah kita jelaskan di tabel saat mendefinisikan route resource untuk `UserController`.

Jika kamu klik salah satu tombol edit, maka kamu akan diarahkan ke halaman baru, halaman tersebut masih kosong, di halaman itulah kita akan membuat edit form.

**Mengambil data user yang akan diedit lalu lempar ke view**

Fitur edit kita yang menggunakan path `users/{id}/edit` dihandle oleh `UserController@edit` untuk itu pertama kita perlu mencari User yang akan diedit berdasarkan route parameter `id` lalu lempar ke view. Kita melakukannya di `UserController@edit`

Buka file `UserController.php` lalu ubah action `edit()` agar menjadi seperti ini:

```
public function edit($id)
{
    $user = \App\User::findOrFail($id);

    return view('users.edit', ['user' => $user]);
}
```

Kode tersebut mencari user dengan id bernilai sesuai variabel `$id`, variabel tersebut berasal dari route parameter `id`. Kita menggunakan method `findOrFail()` bukan `find()` agar seandainya user tidak ditemukan, Laravel akan otomatis memberikan tampilan error model not found.

Setelah itu jika user ditemukan, kita lempar view `user.edit` dengan data `user` yang bernilai data user yang tadi kita cari dengan `findOrFail()`

#### Membuat form edit

Selanjutnya kita perlu buat sebuah file view baru untuk meletakkan form edit kita. View ini kita beri nama `edit.blade.php` dan kita letakkan di direktori `resources/views/users/`. Buka file tersebut dan seperti biasa kita isikan kerangka view seperti ini:

File `resources/views/users/edit.blade.php`

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

@section('title') Edit User @endsection

@section('content')
    user yang akan diedit adalah {{$user->email}}
@endsection
```

Jika sudah, silahkan coba kembali ke halaman list users dan klik salah satu tombol edit, misalnya kamu mengeklik user dengan id 1, maka kamu akan diarahkan ke halaman <http://larashop.test/users/1/edit> dan kamu akan melihat tampilan seperti ini:

## Edit User

user yang akan diedit adalah mas.azamuddin@gmail.com

Kita telah berhasil membaca email dari user yang akan diedit menggunakan kode

```
{{$user->email}}
```

Ini karena sebelumnya kita telah melempar data `user` dari action controller yang menggunakan view ini.

Selanjutnya kita tinggal buat form untuk mengedit field-field user, mirip dengan create form, bedanya adalah setiap input sudah diisi dengan data dari user yang akan diedit.

Pertama kita buat kerangka dasar dari form edit seperti ini:

```
<form enctype="multipart/form-data" class="bg-white shadow-sm p-3" action="{{route('users.update', ['id'=>$user->id])}}" method="POST">
    @csrf
    <input type="hidden" value="PUT" name="_method">
</form>
```

Form ini akan dikirimkan ke url `http://larashop.test/users/{id}` dengan method PUT. Untuk itu pertama kita set url tersebut sebagai `action` dari form html kita. Kita tidak menuliskan secara langsung urlnya, akan tetapi menggunakan helper `route()` dengan mengisikan named route untuk url di atas yaitu

users.update dan mengisikan route parameter `id` dengan nilai id dari user yang akan diedit `$user->edit` seperti ini:

```
route('users.update', ['id' => $user->id])
```

Kamu mungkin bertanya, katanya kita akan menggunakan method "PUT" kok di formnya ditulis method "POST"??? Penjelasannya adalah begini:

Untuk menggunakan method selain GET dan POST, maka kita harus memberikan nilai method pada form sebagai POST, lalu kita tambahkan input bertipe hidden dengan nama `_method` bernilai nama method yang akan digunakan. Misalnya untuk mengirim data dengan method PUT kita menggunakan input hidden ini:

```
<input name="_method" name="PUT">
```

Dan untuk menggunakan method DELETE tinggal kita ganti valuenya menjadi seperti ini:

```
<input name="_method" name="DELETE">
```

Setelah itu kita perlu kembali menggunakan helper `@csrf` supaya request kita bisa diterima oleh Laravel.

Setelah ini kita buat kode html input untuk properti yang bisa diedit. Field yang bisa diedit adalah name, roles, phone, avatar dan address.

Pada setiap input kita isikan value sesuai dengan data dari variabel `$user` misalnya field `name` kita set value `{{$user->name}}` seperti ini:

```
<input  
    value="{{$user->name}}"  
    class="form-control"  
    placeholder="Full Name"  
    type="text"  
    name="name"  
    id="name"/>
```

Untuk form edit kita juga menambahkan fitur untuk mengubah status user dari ACTIVE menjadi INACTIVE atau sebaliknya. Kita tidak melakukannya di form create user karena secara default user yang dibuat kita anggap sebagai user ACTIVE sehingga di database pun defaultnya ACTIVE. Akan tetapi selanjutnya kita bisa mengubah status dari user melalui menu edit ini, itulah kenapa kita menambahkan kode html berikut ini:

```
<label for="">Status</label>  
<br/>  
<input {{$user->status == "ACTIVE" ? "checked" : ""}}  
    value="ACTIVE" type="radio"
```

```

class="form-control"
id="active"
name="status">
<label for="active">Active</label>

<input {{$user->status == "INACTIVE" ? "checked" : ""}}>
value="INACTIVE"
type="radio"
class="form-control"
id="inactive"
name="status">
<label for="inactive">Inactive</label>
<br><br>
```

Untuk field `username` dan `email` tetap kita tampilkan tapi disabled. Tidak bisa diubah.

Untuk roles kita akan mencentang input checkbox di tiap-tiap role yang dimiliki user, kita melakukannya bantuan php function `in_array()` seperti ini:

```
in_array('ADMIN', json_encode($user->roles));
```

Kode di atas mengecek apakah `$user->roles` didalamnya terdapat string "ADMIN". Kita menggunakan `json_decode()` untuk mengubah dari string JSON Array kembali ke PHP Array.

Kita implementasikan pengecekan di atas di masing-masing checkbox role seperti ini:

```

<input
type="checkbox"
{{in_array("ADMIN", json_decode($user->roles)) ? "checked" : ""}}
name="roles[]"
id="ADMIN"
value="ADMIN">
<label for="ADMIN">Administrator</label>

<input
type="checkbox"
{{in_array("STAFF", json_decode($user->roles)) ? "checked" : ""}}
name="roles[]"
id="STAFF"
value="STAFF">
<label for="STAFF">Staff</label>

<input
type="checkbox"
{{in_array("CUSTOMER", json_decode($user->roles)) ? "checked" : ""}}
name="roles[]"
id="CUSTOMER"
value="CUSTOMER">
<label for="CUSTOMER">Customer</label>
```

Jangan bingung ya, kode di atas menggunakan ternary operator yaitu bentuk singkat dari if else seperti ini:

```
in_array("ADMIN", json_decode($user->roles)) ? "checked" : ""
```

Kode di atas logikanya sama saja if else di PHP seperti ini:

```
if(in_array("ADMIN", json_decode($user->roles))){
    // tampilkan "checked"
} else {
    // tampilkan "" (empty string)
}
```

Khusus untuk menampilkan avatar kita lakukan pengecekan, jika user memiliki avatar kita tampilkan image avatarnya, jika tidak ada avatar kita tampilkan teks "No avatar"

```
@if($user->avatar)

<br>
@else
    No avatar
@endif
```

Selain itu pada input file avatar kita berikan keterangan untuk mengabaikan field input ini jika tidak ingin mengubah avatar

```
<small class="text-muted">Kosongkan jika tidak ingin mengubah
avatar</small>
```

Kemudian kita perlu tambahkan juga kode untuk membaca apakah ada pesan session dengan nama status, jika ada tampilkan menggunakan kode ini:

```
@if(session('status'))
<div class="alert alert-success">
    {{session('status')}}
</div>
@endif
```

Kode tersebut untuk menampilkan flash message yang nanti akan kita kirim dari controller saat update user berhasil.

Kini kode view `users/edit.blade.php` kita terlihat seperti ini:

```

@extends('layouts.global')

@section('title') Edit User @endsection

@section('content')
<div class="col-md-8">

@if(session('status'))
<div class="alert alert-success">
{{session('status')}}
</div>
@endif

<form
enctype="multipart/form-data"
class="bg-white shadow-sm p-3"
action="{{route('users.update', ['id'=>$user->id])}}" method="POST">

@csrf

<input
type="hidden"
value="PUT"
name="_method">

<label for="name">Name</label>
<input
value="{{ $user->name }}"
class="form-control"
placeholder="Full Name"
type="text"
name="name"
id="name"/>
<br>

<label for="username">Username</label>
<input
value="{{ $user->username }}"
disabled
class="form-control"
placeholder="username"
type="text"
name="username"
id="username"/>
<br>

<label for="">Status</label>
<br/>
<input {{$user->status == "ACTIVE" ? "checked" : ""}}>
value="ACTIVE"
type="radio"
class="form-control"

```

```

        id="active"
        name="status">
        <label for="active">Active</label>

        <input {{$user->status == "INACTIVE" ? "checked" : ""}}>
        value="INACTIVE"
        type="radio"
        class="form-control"
        id="inactive"
        name="status">
        <label for="inactive">Inactive</label>
<br><br>

        <label for="">Roles</label>
        <br>
        <input
            type="checkbox" {{in_array("ADMIN", json_decode($user->roles)) ? "checked" : ""}}>
            name="roles[]"
            id="ADMIN"
            value="ADMIN">
            <label for="ADMIN">Administrator</label>

        <input
            type="checkbox" {{in_array("STAFF", json_decode($user->roles)) ? "checked" : ""}}>
            name="roles[]"
            id="STAFF"
            value="STAFF">
            <label for="STAFF">Staff</label>

        <input
            type="checkbox" {{in_array("CUSTOMER", json_decode($user->roles)) ? "checked" : ""}}>
            name="roles[]"
            id="CUSTOMER"
            value="CUSTOMER">
            <label for="CUSTOMER">Customer</label>

        <br>

        <br>
        <label for="phone">Phone number</label>
        <br>
        <input
            type="text"
            name="phone"
            class="form-control"
            value="{{$user->phone}}">

        <br>
        <label for="address">Address</label>
        <textarea
            name="address">

```

```

        id="address"
        class="form-control"><{$user->address}>
</textarea>
<br>

<label for="avatar">Avatar image</label>
<br>
Current avatar: <br>
@if($user->avatar)
 }})

```

**Menangkap request edit dan mengupdate ke database**

Form edit user sudah bisa digunakan, akan tetapi jika kamu mencoba untuk submit kamu akan mendapatkan blank page. Itu karena kita belum menulis kode untuk menangkap data dari form edit. Form edit mengirimkan

data ke name route `users.store` alias path `/users/{id}` dengan method PUT, route tersebut menggunakan `UserController@store` untuk itu kita perlu edit action `store` tersebut.

Pertama, kita cari terlebih dahulu user yang akan diedit dengan kode ini:

```
$user = \App\User::findOrFail($id);
```

Lalu kita ubah nilai properti user di atas dengan nilai yang berasal dari form seperti ini:

```
$user->name = $request->get('name');
$user->roles = json_encode($request->get('roles'));
$user->address = $request->get('address');
$user->phone = $request->get('phone');
$user->status = $request->get('status');
```

Untuk field `roles` sebelum menyimpan ke database kita ubah dulu dari PHP Array menjadi JSON Array menggunakan fungsi `json_encode()` agar dapat disimpan ke database.

Lalu kita `UserController@store` kita tambahkan kode untuk menangkap

Setelah itu kita cek jika terdapat request bertipe file dengan nama 'avatar', kita handle file tersebut:

```
if($request->file('avatar')){
    if($user->avatar && file_exists(storage_path('app/public/' . $user->avatar))){
        \Storage::delete('public/' . $user->avatar);
    }
    $file = $request->file('avatar')->store('avatars', 'public');
    $user->avatar = $file;
}
```

Jika terdapat file upload 'avatar' maka kita cek lagi, apakah user yang akan diedit ini memiliki file avatar dan apakah file tersebut ada di server kita, jika ada maka kita hapus file tersebut.

```
if($user->avatar && file_exists(storage_path('app/public/' . $user->avatar))){
    \Storage::delete('public/' . $user->avatar);
}
```

Untuk menghapus file kita gunakan `\Storage::delete()` argument yang dimasukan adalah path relatif dari `storage/app` tidak perlu full path. Maka dari itu kita menggunakan '`public/' . $user->avatar` yang akan menghapus file di folder `storage/app/public/avatars/fileavataruser.png` misalnya.

Setelah itu kita simpan file yang diupload ke folder "avatars" seperti sebelumnya menggunakan method `store()` seperti ini:

```
$file = $request->file('avatar')->store('avatars', 'public');
```

Lalu kita set field 'avatar' user dengan path baru dari image yang diupload tadi

```
$user->avatar = $file;
```

Kemudian kita update ke database dengan method `save()` seperti ini:

```
$user->save();
```

Setelah itu kita redirect ke form edit user dengan status bahwa berhasil melakukan update dengan kode ini:

```
return redirect()->route('users.edit', ['id' => $id])->with('status', 'User successfully updated');
```

Sehingga hasil akhir dari kode action `UserController@store` adalah seperti ini:

```
public function update(Request $request, $id)
{
    $user = \App\User::findOrFail($id);

    $user->name = $request->get('name');
    $user->roles = json_encode($request->get('roles'));
    $user->address = $request->get('address');
    $user->phone = $request->get('phone');

    if($user->avatar && file_exists(storage_path('app/public/' . $user->avatar))){
        \Storage::delete('public/' . $user->avatar);
        $file = $request->file('avatar')->store('avatars', 'public');
        $user->avatar = $file;
    }

    $user->save();

    return redirect()->route('users.edit', ['id' => $id])->with('status', 'User successfully updated');
}
```

Sekarang fitur edit user telah siap kita gunakan. Silahkan coba mengedit salah satu user yang ada.

## Fitur delete user

Tambahkan link delete di list user

Buka view `resources/views/users/index.blade.php` lalu tambahkan kode untuk tombol delete seperti ini:

```
<form
  onsubmit="return confirm('Delete this user permanently?')"
  class="d-inline"
  action="{{route('users.destroy', ['id' => $user->id ])}}"
  method="POST">

  @csrf

  <input
    type="hidden"
    name="_method"
    value="DELETE">

  <input
    type="submit"
    value="Delete"
    class="btn btn-danger btn-sm">

</form>
```

Letakkan kode di tersebut tepat setelah kode untuk menampilkan tombol edit.

Tombol untuk delete menggunakan `<form>` dengan nilai `method` adalah POST karena kita akan menggunakan method DELETE untuk mengirimkan request ke path `/users/{id}` alias named route `users.destroy`. Kita set form action menggunakan `route()` helper ke named route tadi dan memberikan parameter `id` bernilai `$user->id`. Lalu kita tambhkan helper `@csrf` agar request kita valid. Kita juga menambahkan `<input>` bertipe `hidden` dengan nama `_method` bernilai "DELETE" yang mengindikasikan bahwa form ini akan mengirimkan data menggunakan method DELETE.

Tapi akan terlalu berisiko jika user sekali klik langsung menghapus, bagaimana kalau ternyata user salah klik? untuk itu kita perlu menggunakan dialog konfirmasi sebelum benar-benar mengirim request delete ini, untuk kepentingan latihan kita gunakan built in confirmation di javascript, maka kita menambahkan nya pada kode form saat submit, perhatikan kode ini di form di atas:

```
<form onsubmit="return confirm('Delete this user permanently?')" ...>
```

Kode di atas akan menyebabkan dialog konfirmasi muncul jika user mengeklik tombol delete.

Selanjutnya kita tangkap permintaan delete ini pada `UserController@destroy`.

**Menangkap request delete dan menghapus user di database**

Ingat bahwa request delete dikirimkan ke path `users/1` dengan method delete alias named route `users.destroy` yang menggunakan `UserController@destroy`. Mari kita tangkap permintaan tersebut dan hapus user di database, pertama kita cari user yang akan dihapus berdasarkan route parameter id seperti ini:

```
$user = \App\User::findOrFail();
```

Setelah itu kita hapus user tersebut dengan kode ini:

```
$user->delete();
```

Lalu kita redirect kembali ke halaman list user dengan pesan bahwa delete telah berhasil seperti ini:

```
return redirect()->route('users.index')->with('status', 'User successfully deleted');
```

Sehingga `UserController@destroy` akan terlihat seperti ini:

```
public function destroy($id)
{
    $user = \App\User::findOrFail($id);

    $user->delete();

    return redirect()->route('users.index')->with('status', 'User successfully deleted');
}
```

Setelah itu agar halaman list users bisa membaca pesan session kita, kita tambahkan di atas kode form di file `users/index.blade.php` dengan kode ini:

```
@if(session('status'))
<div class="alert alert-success">
    {{session('status')}}
</div>
@endif
```

## Menampilkan detail user

Satu fitur lagi di CRUD user yang akan kita buat, yaitu fitur untuk menampilkan detail user dengan id tertentu.

Pertama kita buat tombolnya di halaman list user. Buka file view `users/index.blade.php` lalu tambahkan kode ini setelah kode untuk tombol edit:

```
<a  
    href="{{route('users.show', ['id' => $user->id])}}"  
    class="btn btn-primary btn-sm">Detail</a>
```

#### Mencari user dengan id tertentu

Buka file `UserController` dan edit action `show`, tambahkan kode ini:

```
$user = \App\User::findOrFail($id);
```

Kemudian data tersebut kita lempar ke view `users.show` seperti ini:

```
return view('users.show', ['user' => $user]);
```

Sehingga action `show()` kita akan terlihat seperti ini:

```
public function show($id)  
{  
    $user = \App\User::findOrFail($id);  
  
    return view('users.show', ['user' => $user]);  
}
```

#### Membuat view untuk detail user

Action show telah melempar ke view `users.show`, mari kita buat file tersebut di direktori `resources/view/users` dengan nama `show.blade.php`, Lalu isikan kode ini:

```
@extends('layouts.global')  
  
@section('title') Detail user @endsection  
  
@section('content')  
  
@endsection
```

Nah kita akan menampilkan detail user di section content. Untuk itu mari kita tambahkan kode berikut:

```
<div class="col-md-8">
    <div class="card">
        <div class="card-body">
            <b>Name:</b> <br/>
            {{$user->name}}
            <br><br>

            @if($user->avatar)
                
            @else
                No avatar
            @endif

            <br>
            <br>
            <b>Username:</b><br>
            {{$user->email}}

            <br>
            <br>
            <b>Phone number</b> <br>
            {{$user->phone}}

            <br><br>
            <b>Address</b> <br>
            {{$user->address}}

            <br>
            <br>
            <b>Roles:</b> <br>
            @foreach (json_decode($user->roles) as $role)
                &middot; {{$role}} <br>
            @endforeach
        </div>
    </div>
</div>
```

Jika kamu klik salah satu tombol detail di halaman list user kamu akan mendapati tampilan detail user kurang lebih seperti ini:

## Detail user

**Name:**

Muhammad Azamuddin

**Username:**

azamuddin@live.com

**Phone number**

0871111111

**Address**

Jalan Haji Sarmili

**Roles:**

- ADMIN
- CUSTOMER

### Filter User berdasarkan Email

Kita akan membuat filter user berdasarkan email di halaman list users. Untuk itu mari kita buka kembali file view `users/index.blade.php` lalu kita tambahkan form untuk melakukan filter, form ini kita letakkan di atas tabel. Kodenya adalah sebagai berikut:

```
<div class="row">
    <div class="col-md-6">
        <form action="{{route('users.index')}}">
            <div class="input-group mb-3">
                <input
                    value="{{Request::get('keyword')}}"
                    name="keyword"
                    class="form-control col-md-10"
                    type="text"
                    placeholder="Filter berdasarkan email"/>
```

```
<div class="input-group-append">
    <input
        type="submit"
        value="Filter"
        class="btn btn-primary">
</div>
</div>
</form>
</div>
</div>
```

Form tersebut akan mengirimkan keyword pencarian / keyword filter ke route `users.index`. Route ini menggunakan `UserController@index`, oleh krenanya mari kita tangkap request filter ini di `UserController`.

Seperti kita lihat bahwa action `index` belum menggunakan `$request`, saat ini kodennya terlihat seperti ini:

```
public function index(){
    // kode ...
}
```

Kita harus melakukan injection `$request` agar kita bisa menggunakannya, ubah agar action `index` memiliki bentuk seperti ini:

```
public function index(Request $request){
    // kode ...
}
```

Jangan hapus kode di dalam action `index()` cukup ganti definisi function nya saja. Setelah itu mari kita tangkap terlebih dahulu request variabel bernama 'keyword` seperti ini:

```
$filterKeyword = $request->get('keyword');
```

Data yang ditangkap berasal dari apa yang diketikan oleh user melalui inputan ini:

```
<input
    name="keyword"
    class="form-control col-md-10"
    type="text"
    placeholder="Filter berdasarkan email"/>
```

Setelah itu kita check jika ada `$filterKeyword` maka kita query `User` yang emailnya memiliki sebagian dari keyword seperti ini:

```
if($filterKeyword){
    $users = \App\User::where('email', 'LIKE', "%$filterKeyword%");
}
```

Ingat kita tidak menghapus kode yang sebelumnya sudah ada di action ini, akan tetapi kita memodifikasi agar action ini mampu mendeteksi bahwa ada request untuk filter user berdasarkan email tertentu. Jika ada, maka kita cari menggunakan model `User` method `where()`, dan kita gunakan operator `LIKE` dan nilai dari keyword kita bungkus dalam `'%'` sehingga query ini akan mencari `User` yang memiliki email mengandung keyword tertentu, bukan exact match.

Maka kini action `index` di `UserController` akan terlihat seperti ini:

```
public function index(Request $request)
{
    $users = \App\User::paginate(10);

    $filterKeyword = $request->get('keyword');

    if($filterKeyword){
        $users = \App\User::where('email', 'LIKE', "%$filterKeyword%")-
>paginate(10);
    }

    return view('users.index', ['users' => $users]);
}
```

Dengan begitu, fitur filter telah selesai dan bisa kita gunakan silahkan coba buka kembali <http://larashop.test/users> dan cari user berdasarkan email.

## Users List

Filter berdasarkan email			Filter	
Name	Username	Email	Avatar	Action
Site Administrator	administrator	administrator@larashop.test		<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Muhammad Azamuddin	azamuddin91	azamuddin@live.com		<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Hana Humaira	hana	hana@humaira.com	N/A	<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
User Empat	user4	user4@gmail.com	N/A	<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>

Telah selesai fitur CRUD untuk model **User**. Kita akan melanjutkan ke beberapa fitur CRUD untuk model-model lainnya, kamu perlu memahami apa yang telah kita pelajari di CRUD **User** karena setelah ini saya tidak akan mengulangi terlalu detail hal-hal yang sudah dijelaskan ya. Supaya lebih singkat dan kamu semakin fokus untuk memahami pembuatan CRUD.

Sebelum kita melangkah ke CRUD berikutnya, kita tambahkan dulu tombol untuk Create User di halaman daftar user. Buka file `resources/views/users/index.blade.php` lalu tambahkan kode berikut tepat di atas kode `<table>`:

```

<div class="row">
    <div class="col-md-12 text-right">
        <a
            href="{{route('users.create')}}"
            class="btn btn-primary">Create user</a>
    </div>
</div>
<br>

```

Sehingga akan muncul di halaman daftar user sebuah tombol di kanan atas tabel seperti ini:

## Users List

Filter berdasarkan email			Filter	Create User		
Name	Username	Email	Avatar	Action		
Site Administrator	administrator	administrator@larashop.test		<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
Muhammad Azamuddin	azamuddin91	azamuddin@live.com		<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>

Lalu kita buat juga menu "Manage users" di sidebar kiri, caranya kita buka layout `resources/views/layouts/global.blade.php` lalu tambahkan kode berikut ini:

```
<li>
  <a href="{{route('users.index')}}">
    <span class="oi oi-people"></span> Manage Users
  </a>
</li>
```

Tepat setelah atau dibawah baris ini:

```
<li><a href="/home"><span class="oi oi-home"></span> Home</a></li>
```

Sehingga kini tampilan aplikasi kita setelah login adalah seperti ini:

The screenshot shows the Larashop application's user management interface. At the top, there is a dark blue header with the 'Larashop' logo, a search bar, and a dropdown menu for 'Site Administrator'. On the left, a sidebar menu includes 'Home' and 'Manage Users' (which is currently selected). The main content area is titled 'Users List' and contains a table of user data. The table has columns for Name, Username, Email, Avatar, and Action (with 'Edit', 'Detail', and 'Delete' buttons). Five users are listed: Site Administrator (administrator@larashop.test), Muhammad Azamuddin (azamuddin@live.com), Nadia Nurul Mila (nadia@gmail.com), and Hana Humaira (hana@humaira.com). Each user row also has a 'Create User' button at the bottom right.

Name	Username	Email	Avatar	Action
Site Administrator	administrator	administrator@larashop.test		<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Muhammad Azamuddin	azamuddin91	azamuddin@live.com		<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>
Hana Humaira	hana	hana@humaira.com	N/A	<a href="#">Edit</a> <a href="#">Detail</a> <a href="#">Delete</a>

## Fitur filter by status user

Selain fitur filter berdasarkan nama, kita juga ingin bisa melakukan filter berdasarkan status user, ACTIVE atau INACTIVE.

Baik, pertama kita ingin menampilkan status user di halaman list user. Untuk itu buka kembali file view `users/index.blade.php`, kita tambahkan kolom status di tabel seperti ini:

```
<th><b>Name</b></th>
<th><b>Username</b></th>
<th><b>Email</b></th>
<th><b>Avatar</b></th>
<th><b>Status</b></th>
<th><b>Action</b></th>
```

Lalu tambahkan kode berikut ini untuk menampilkan status user aplikasi kita, jika ACTIVE kita tampilkan sebagai `badge-success` berwarna hijau, INACTIVE kita tampilkan sebagai `badge-danger` berwarna merah.

```
<td>
@if($user->status == "ACTIVE")

{{ $user->status}}

@else

{{ $user->status}}

@endif
</td>
```

View `users/index.blade.php` sekarang menjadi seperti ini:

```
@extends("layouts.global")

@section("title") Users list @endsection

@section("content")



<form action="{{ route('users.index') }}">
<div class="input-group mb-3">

<input
value="{{ Request::get('keyword') }}"
name="keyword"
class="form-control col-md-10"


```

```
        type="text"
        placeholder="Filter berdasarkan email"/>

    <div class="input-group-append">
        <input
            type="submit"
            value="Filter"
            class="btn btn-primary">
    </div>
</div>
</form>
</div>
</div>

<br>

@if(session('status'))
<div class="alert alert-success">
    {{session('status')}}
</div>
@endif

<div class="row">
    <div class="col-md-12 text-right">
        <a href="{{route('users.create')}}" class="btn btn-primary">Create user</a>
    </div>
</div>
<br>
<table class="table table-bordered">
    <thead>
        <tr>
            <th><b>Name</b></th>
            <th><b>Username</b></th>
            <th><b>Email</b></th>
            <th><b>Avatar</b></th>
            <th><b>Status</b></th>
            <th><b>Action</b></th>
        </tr>
    </thead>
    <tbody>
        @foreach($users as $user)
        <tr>
            <td>{{$user->name}}</td>
            <td>{{$user->username}}</td>
            <td>{{$user->email}}</td>
            <td>
                @if($user->avatar)
                
                @else
                N/A
                @endif
            </td>
        </tr>
    @endforeach
</tbody>
</table>
```

```

<td>
    @if($user->status == "ACTIVE")
        <span class="badge badge-success">
            {{$user->status}}
        </span>
    @else
        <span class="badge badge-danger">
            {{$user->status}}
        </span>
    @endif
</td>
<td>
    <a
        class="btn btn-info text-white btn-sm"
        href="{{route('users.edit', ['id'=>$user->id])}}>Edit</a>

    <a
        href="{{route('users.show', ['id' => $user->id])}}"
        class="btn btn-primary btn-sm">Detail</a>

    <form
        onsubmit="return confirm('Delete this user permanently?')"
        class="d-inline"
        action="{{route('users.destroy', ['id' => $user->id ])}}"
        method="POST">

        @csrf

        <input
            type="hidden"
            name="_method"
            value="DELETE">

        <input
            type="submit"
            value="Delete"
            class="btn btn-danger btn-sm">
    </form>
</td>
</tr>
@endforeach
<tfoot>
    <tr>
        <td colspan=10>
            {{$users->links()}}
        </td>
    </tr>
</tfoot>
</tbody>
</table>

@endsection

```

Dan jika kita refresh halaman list user maka kini kita bisa melihat status user di tabel seperti ini:

Name	Username	Email	Avatar	Status	Action
Site Administrator	administrator	administrator@larashop.test		ACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
Muhammad Azamuddin	azamuddin91	azamuddin@live.com		ACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	INACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
Hana Humaira	hana	hana@humaira.com	N/A	INACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
User Empat	user4	user4@gmail.com	N/A	ACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
User Lima	user5	user5@gmail.com	N/A	ACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
User Enam	user6	user6@gmail.com	N/A	ACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>

Selanjutnya kita tambahkan form control untuk filter berdasarkan status seperti:

```
<form action="{{route('users.index')}}">
    <div class="row">
        <div class="col-md-6">
            <input
                value="{{Request::get('keyword')}}"
                name="keyword"
                class="form-control"
                type="text"
                placeholder="Masukan email untuk filter..."/>
        </div>
        <div class="col-md-6">
            <input {{Request::get('status') == 'ACTIVE' ? 'checked' : ''}}>
                value="ACTIVE"
                name="status"
                type="radio"
                class="form-control"
                id="active">
            <label for="active">Active</label>

            <input {{Request::get('status') == 'INACTIVE' ? 'checked' : ''}}>
                value="INACTIVE"
                name="status"
                type="radio"
                class="form-control"
                id="inactive">
            <label for="inactive">Inactive</label>

            <input
                type="submit"
                value="Filter">
        </div>
    </div>
</form>
```

```

        class="btn btn-primary">
    </div>
</div>
</form>

```

Kode di atas menambahkan dua input radio baru dengan nama `status` untuk dikirim ke route `users.index` yang menggunakan `UserController@index`. Dua input radio tersebut memungkinkan user untuk memilih ACTIVE atau INACTIVE.

Kode tersebut kita taruh di file `users/index.blade.php` untuk menggantikan kode ini:

```

<div class="row">
    <div class="col-md-6">
        <form action="{{route('users.index')}}">
            <div class="input-group mb-3">
                <input
                    value="{{Request::get('keyword')}}"
                    name="keyword"
                    class="form-control col-md-10"
                    type="text"
                    placeholder="Filter berdasarkan email"/>
            <div class="input-group-append">
                <input
                    type="submit"
                    value="Filter"
                    class="btn btn-primary">
            </div>
        </div>
    </div>
</form>
</div>
</div>

```

Sehingga kini kita bisa melihat filter baru berdasarkan status di halaman list users seperti ini

## Users List

Selanjutnya kita sesuaikan `UserController@index` agar menerima filter berdasarkan status dan memprosesnya sebagai query di model `User`

di `UserController@index` kita buat sebuah variabel bernama `$status` dengan nilai berasal dari request bernama 'status' seperti ini:

```
$status = $request->get('status');
```

Lalu kita cek jika `$status` memiliki nilai maka kita gunakan untuk query model `User` berdasarkan status, jika tidak maka query model `User` tanpa status, seperti ini:

```
$status = $request->get('status');

if($status){
    $users = \App\User::where('status', $status)->paginate(10);
} else {
    $users = \App\User::paginate(10);
}
```

Kemudian kita juga perlu mengubah kode jika terdapat request filter berdasarkan email yang tadinya seperti ini:

```
if($filterKeyword){
    $users = \App\User::where('email', 'LIKE', "%$filterKeyword%")-
>paginate(10);
}
```

menjadi seperti ini:

```
if($filterKeyword){
    if($status){
        $users = \App\User::where('email', 'LIKE', "%$filterKeyword%")
            ->where('status', $status)
            ->paginate(10);
    } else {
        $users = \App\User::where('email', 'LIKE', "%$filterKeyword%")
            ->paginate(10);
    }
}
```

Perubahan di atas kita lakukan supaya query ke model `User` berdasarkan email juga mengikutkan query berdasarkan status juga. Artinya jika ada filter berdasarkan email 'mas.azamuddin@gmail.com' dan status INACTIVE, maka kedua filter tersebut akan menjadi query ke model `User` dengan operator AND. Dengan kata lain, cari User yang emailnya mengandung keyword 'mas.azamuddin@gmail.com' dan statusnya INACTIVE.

Jika sudah, maka kini kita bisa melakukan selain berdasarkan email juga filter berdasarkan status user, ini contohnya:

## Users List

Name	Username	Email	Avatar	Status	Action
Nadia Nurul Mila	nadia	nadia@gmail.com	N/A	INACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>
Hana Humaira	hana	hana@humaira.com	N/A	INACTIVE	<button>Edit</button> <button>Detail</button> <button>Delete</button>

### Menampilkan pagination dan nomor urut

Apakah kamu sadar masih ada yang kurang dari tampilan list users kita? Yup, kita belum menampilkan link pagination, padahal di controller kita telah menggunakan method `paginate(10)` untuk menampilkan per halaman 10 users.

Kita buka kembali file view `users/index.blade.php` lalu table footer setelah `</tbody>` seperti ini:

```
<tfoot>
  <tr>
    <td colspan=10>
      {{$users->links()}}
    </td>
  </tr>
</tfoot>
```

Kita tampilkan link pagination dengan  `{{$users->links()}}` sehingga jika ada lebih dari 10 users di tabel `users` akan terlihat tampilan link pagination seperti ini:

User Empat	user4	user4@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
User Lima	user5	user5@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
User Enam	user6	user6@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
Ridwan Mutaffaq	ridwan	ridwan@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
Test 1	test	mas.azamuddin@gmail.com		<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
Habib Asagaf	habib	habib@gmail.com		<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
Iqbal Kholis	iqbal	iqbal@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>
User ABC	userabc	userabc@gmail.com	N/A	<span>ACTIVE</span>	<a href="#">Edit</a>	<a href="#">Detail</a>	<a href="#">Delete</a>

< 1 2 >

Namun, jika kita melakukan filtering dan hasilnya lebih dari 1 halaman lalu kita klik ke halaman kedua maka filter tersebut akan hilang. Kita tidak ingin hal itu terjadi, oleh karena itu dalam setiap link pagination kita juga ingin menambahkan query string filter yang sedang aktif, caranya mudah cukup ubah kode untuk menampilkan link pagination agar menjadi seperti ini:

```
{{ $users->appends(Request::all())->links()}}
```

Baik, kita telah menyelesaikan fitur CRUD users, selanjutnya kita akan belajar untuk membuat CRUD untuk model-model lainnya. Sebagian besar materi terkait CRUD telah kita pelajari di poin CRUD users ini sehingga setelah ini kita akan belajar dengan lebih cepat tanpa perlu banyak mengulang detail yang telah kita pelajari di poin CRUD users ini. Oleh karenanya, diharapkan kamu telah benar-benar memahami apa yang telah kita pelajari dalam pembuatan CRUD users. Jika belum, ada baiknya kamu membaca ulang masing-masing poin dan berusaha memahaminya kembali. Setelah itu lanjutkan ke poin studi kasus berikutnya.

## Manage Category

Membuat migration table categories

Pertama kita perlu membuat tabel `categories` kita lakukan dengan migration, untuk itu kita buat terlebih dahulu sebuah migration seperti ini:

```
php artisan make:migration create_table_categories
```

Setelah menjalankan perintah di atas buka file yang baru saja dibuat di direktori `database/migrations`

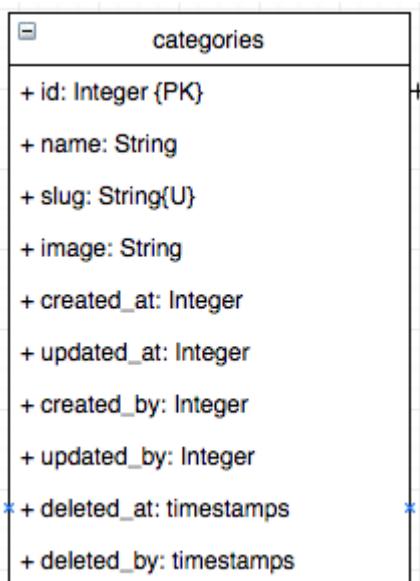
Kemudian kita tambahkan kode ini untuk membuat struktur tabel `categories` di function `up()`

```

public function up(){
    Schema::create('categories', function (Blueprint $table) {
        $table->increments('id');
        $table->string("name");
        $table->string("slug")->unique();
        $table->string("image")->comment("berisi nama file image saja tanpa
path");
        $table->integer("created_by");
        $table->integer("updated_by")->nullable();
        $table->integer("deleted_by")->nullable();
        $table->softDeletes();
        $table->timestamps();
    });
}

```

Kita membuat struktur tabel di atas merujuk pada desain database awal tabel `categories` ini:



Kemudian untuk keperluan rollback, kita tambahkan kode untuk menghapus table di method `down()` file migration tadi:

```

public function down(){
    Schema::dropIfExists('categories');
}

```

Jika sudah jangan lupa untuk menjalankan perintah migrate di projek kita:

```
php artisan migrate
```

Membuat model Category

Langkah berikutnya adalah membuat model **Category**, pada CRUD Users kita tidak perlu membuat tabel **User** karena sejak instalasi Laravel telah menyediakan model **User** untuk kita. Akan tetapi untuk model-model selain **User** kita perlu membuatnya, maka jalankan perintah untuk membuat model ini:

```
php artisan make:model Category
```

Setelah kamu dijalankan sebuah file model baru akan dibuat yaitu **app/Model.php**. Untuk sementara kita biarkan saja apa adanya model **Category** tersebut.

## CRUD Category

### Membuat resource category

Setelah tabel dan model **Category** siap, selanjutnya kita perlu membuat route dan controller. Seperti sebelumnya kita akan menggunakan fitur route resource dan controller resource. Buka file **routes/web.php** lalu tambahkan route resource untuk **categories** seperti ini:

```
Route::resource('categories', 'CategoryController');
```

Tapi tunggu dulu, route di atas menggunakan **CategoryController** tapi kita tidak memiliki file **app/Http/Controller/CategoryController.php**, kita perlu membuatnya terlebih dahulu dan menjadikannya sebagai controller resource. Buat **CategoryController** resource dengan perintah ini:

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

Dengan begitu kini resource **Category** telah siap kita gunakan, coba kamu akses <http://larashop.test/categories/create> maka kamu tidak akan mengalami error, hanya blank page saja karena kita belum memberikan kode di **CategoryController@create**.

### Fitur create category

Fitur pertama dari CRUD Category yang akan kita buat adalah fitur create category. Untuk itu mari kita ubah kode **CategoryController@create** agar menjadi seperti ini:

```
public function create(){
    return view('categories.create');
}
```

### Membuat view untuk create category

Kalo kamu coba untuk akses <http://larashop.test/categories/create> kamu akan mendapatkan error view not found. Itu karena kita belum membuat view **categories.create**. Maka, kita buat terlebih dahulu file

tersebut yaitu `resources/views/categories/create.blade.php`, silahkan dibuat terlebih dahulu file viewnya. Buat folder baru `categories` di `resources/views` lalu buat file `create.blade.php` di folder `categories` tadi. Setelah itu buka filenya dan kita isikan kerangka view kita ini:

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

@section('title') Create Category @endsection

@section('content')

TODO: Form create category

@endsection
```

Kini jika kamu merefresh halaman <http://larashop.test/categories/create>, error view not found tidak akan muncul lagi dan kita mendapati tampilan yang berasal dari view `categories/create.blade.php`.

Setelah itu di bagian section content mari kita berikan form dan inputan untuk membuat category baru

```
<form
  enctype="multipart/form-data"
  class="bg-white shadow-sm p-3"
  action="{{route('categories.store')}}"
  method="POST">

  @csrf

</form>
```

Kode ini merupakan kerangka form kita, kita mengeset atribut `enctype` bernilai `multipart/form-data` karena kita akan melakukan upload file. Form ini akan mengirimkan data ke `route('categories.store')` alias <http://larashop.test/categories> dengan method POST untuk itu kita set attribute `method` di form ini bernilai POST. Kemudian kita tambahkan styling template bootstrap 4 pada attribute `class="bg-white shadow-sm p-3"`. Dan yang paling penting jangan ketinggal helper `@csrf` supaya request yang akan kita kirim melalui form ini valid.

Setelah itu kita tambahkan input untuk category name dan upload image category seperti ini:

```
<label>Category name</label><br>
<input
  type="text"
  class="form-control"
  name="name">
<br>

<label>Category image</label>
<input
```

```
type="file"
class="form-control"
name="image"

<br>

<input
  type="submit"
  class="btn btn-primary"
  value="Save">
```

Sehingga kini view `categories/create.blade.php` menjadi seperti ini:

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

@section('title') Create Category @endsection

@section('content')

<div class="col-md-8">
  <form
    enctype="multipart/form-data"
    class="bg-white shadow-sm p-3"
    action="{{route('categories.store')}}"
    method="POST">

    @csrf

    <label>Category name</label><br>
    <input
      type="text"
      class="form-control"
      name="name"/>
    <br>

    <label>Category image</label>
    <input
      type="file"
      class="form-control"
      name="image"/>

    <br>

    <input
      type="submit"
      class="btn btn-primary"
      value="Save"/>

  </form>
</div>

@endsection
```

### Menangkap request create categories

Form create categories telah siap untuk mengirimkan data category akan tetapi controller kita belum melakukan apa-apa, kita perlu menangkap request data dari form create categories untuk kemudian membuat kategori baru di database:

```
public function store(Request $request){
    $name = $request->get('name');

    $new_category = new \App\Category;
    $new_category->name = $name;

    if($request->file('image')){
        $image_path = $request->file('image')
            ->store('category_images', 'public');

        $new_category->image = $image_path;
    }

}
```

Kode di atas menangkap request dengan nama 'name' ke dalam variabel `$name` kemudian membuat sebuah instance dari model `Category` baru dan memberikan nilai name = `$name`.

Setelah itu melakukan pengecekan apakah ada request bertipe file dengan nama 'image', kita lakukan dengan kode `if($request->file('image'))`. Jika ada maka kita simpan file tersebut ke dalam folder `storage/app/public/category_images`, penyimpanan tersebut kita lakukan dengan kode `$request->file('image')->store('category_images', 'public');` Setelah itu path dari file yang baru saja diupload tersebut kita jadikan nilai dari field 'image' pada kategori baru tadi. Kategori baru tersebut belum tersimpan didatabase selama kita belum memanggil `$new_category->save()`. Jangan kita save terlebih dahulu karena kita ingin memberikan nilai pada `created_by`, field ini mengindikasikan siapa pembuat category ini, nilai dari `created_by` adalah ID dari user yang melakukan pembuatan category. Maka kita perlu mengambil id dari user yang sedang login saat ini, untuk itulah sebelum menyimpan kita tambahkan kode berikut ini:

```
$new_category->created_by = \Auth::user()->id;
```

Kode di atas mengambil nilai id dari user yang sedang login dan berikan ke field `created_by`,

Ada satu lagi field yang tidak ada di form akan tetapi diperlukan di database, yaitu field `slug` dan kita bisa menggenerate slug dengan helper `str_slug()` yang ada di Laravel. Maka kita buat terlebih dahulu slug berdasarkan nama kategori seperti ini:

```
$new_category->slug = str_slug($name, '-');
```

Slug merupakan karakter yang tidak menyalahi aturan URL, artinya tidak ada spasi, huruf besar kecil, dsb. Misalnya nama kategori "Sepatu Olahraga" maka dengan `str_slug("Sepatu Olahraga", "-")` akan menghasilkan "sepatu-olahraga"

Baru setelah itu kita simpan model kategori baru tersebut ke database dengan kode ini:

```
$new_category->save();
```

Setelah itu kita redirect kembali ke halaman create category beserta dengan pesan bahwa pembuatan category telah berhasil:

```
return redirect()->route('categories.create')->with('status', 'Category successfully created');
```

Sehingga `CategoryController@store` kita terlihat seperti ini:

```
public function store(Request $request)
{
    $name = $request->get('name');

    $new_category = new \App\Category;
    $new_category->name = $name;

    if($request->file('image')){

        $image_path = $request->file('image')
            ->store('category_images', 'public');

        $new_category->image = $image_path;
    }

    $new_category->created_by = \Auth::user()->id;

    $new_category->slug = str_slug($name, '-');

    $new_category->save();

    return redirect()->route('categories.create')->with('status', 'Category successfully created');
}
```

Dan satu lagi jangan lupa menambahkan kode berikut ini di view `categories/create.blade.php` tepat di atas kode pembuka form kita agar pesan flash dapat terbaca:

```
@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif
```

Fitur create category telah berhasil kita selesaikan, silahkan coba untuk membuat beberapa kategori baru. Ingat pastikan kamu sudah login ya supaya tidak error (ingat kita membaca id user yang sedang login di `CategoriesController@store`), meskipun kita sekarang belum membatasi akses ke create category hanya untuk user yang login.

Tampilan setelah berhasil membuat kategori baru:

## Create Category

Category successfully created

Category name

Category image

No file chosen

## Fitur category list

Untuk membuat fitur category list pertama kita buat file view `categories/index.blade.php` di `resources/views`, lalu kita sesuaikan `CategoryController@index` seperti ini:

```
public function index(){
    $categories = \App\User::paginate(10);

    return view('categories.index', ['categories' => $categories]);
}
```

Setelah itu buka kembali file view `categories/index.blade.php` dan kita isikan kerangka view kita:

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

@section('title') Category list @endsection

@section('content')

@endsection
```

Kemudian kita berikan kode di section 'content' untuk menampilkan daftar kategori seperti ini:

```
<div class="row">
    <div class="col-md-12">
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th><b>Name</b></th>
                    <th><b>Slug</b></th>
                    <th><b>Image</b></th>
                    <th><b>Actions</b></th>
                </tr>
            </thead>

            <tbody>
                @foreach ($categories as $category)
                <tr>
                    <td>{{$category->name}}</td>
                    <td>{{$category->slug}}</td>
                    <td>
                        @if($category->image)
                            
                        @else
                            No image
                        @endif
                    </td>
                    <td>
                        [TODO: actions]
                    </td>
                </tr>
                @endforeach
            </tbody>
            <tfoot>
                <tr>
                    <td colSpan="10">
                        {{$categories->appends(Request::all()->links())}}
                    </td>
                </tr>
            </tfoot>
        </table>
```

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

Akses <http://larashop.test/categories> kamu akan mendapati tampilan seperti ini:

## Category List

Name	Slug	Image	Actions
Sports	sports		[TODO: actions]
Sepatu Olahraga	sepatu-olahraga		[TODO: actions]

### Filter kategori berdasarkan nama

Kita perlu membuat input untuk filter di atas tabel category list, buka file view `categories/index.blade.php` dan tambahkan kode berikut di bagian paling atas section 'content':

```
<div class="row">
  <div class="col-md-6">
    <form action="{{route('categories.index')}}">

      <div class="input-group">
        <input
          type="text"
          class="form-control"
          placeholder="Filter by category name"
          name="name">
        <div class="input-group-append">
          <input
            type="submit"
            value="Filter"
            class="btn btn-primary">
        </div>
      </div>

    </form>
  </div>
</div>
<hr class="my-3">
```

**Menangkap request filter by name**

Kita tahu bahwa kode input filter yang baru saja kita tambahkan memuat input dengan 'name' bernilai "name" juga, dan request tersebut yang akan kita jadikan keyword untuk filter model **Category** berdasarkan nama. Buka **CategoryController** dan inject variable **\$request** ke dalam action **index** seperti ini:

```
public function index(Request $request)
```

Setelah itu kita bisa menangkap request dari form filter, kita tangkap request dengan nama 'name':

```
$filterKeyword = $request->get('name');
```

Kemudian kita cek jika **\$filterKeyword** memiliki nilai maka kita gunakan variable tersebut untuk memfilter model **Category** yang akan kita lempar ke view seperti ini:

```
if($filterKeyword){
    $categories = \App\Category::where("name", "LIKE", "%$filterKeyword%")-
>paginate(10);
}
```

Perubahan tersebut sudah cukup agar fungsi filter kita bisa berfungsi. Hasil akhir dari action **index** di **CategoryController** seperti ini:

```
public function index(Request $request){
    $categories = \App\Category::paginate(10);

    $filterKeyword = $request->get('name');

    if($filterKeyword){
        $categories = \App\Category::where("name", "LIKE",
"%$filterKeyword%")->paginate(10);
    }

    return view('categories.index', ['categories' => $categories]);
}
```

Kini kita bisa melakukan filter kategori berdasarkan nama seperti ini:

## Category List

<input type="text" value="sport"/>		<button>Filter</button>	
Name	Slug	Image	Actions
Sports	sports		[TODO: actions]

### Fitur edit category

Selanjutnya kita buat fitur edit category, semua field di category bisa diedit.

Pertama kita buat tombol edit di halaman category list, buka file view `categories/index.blade.php` dan tambahkan kode berikut ini:

```
<a href="{{route('categories.edit', ['id' => $category->id])}}>
    Edit </a>
```

Kode di atas menggantikan kode ini:

```
[TODO: actions]
```

Jika sudah, kita refresh halaman <http://larashop.test/categories> maka kita akan melihat tombol edit. Klik tombol edit di salah satu kategori maka kita akan diarahkan ke halaman baru yang masih blank. Selanjutnya di situ akan kita buat form untuk melakukan edit category.

Tombol edit tadi mengarahkan kita ke url `http://larashop.test/categories/{id}/edit`, url tersebut merupakan route yang dihasilkan oleh Route resource yang kita definisikan di awal pembuatan CRUD Categories. Url tersebut menggunakan `CategoryController@edit` maka kita perlu mengubah kode di action tersebut.

Pertama, di action `edit` kita tangkap route parameter ID, kemudian nilainya kita gunakan untuk mencari model `Category` seperti ini:

```
public function edit($id){
    $category_to_edit = \App\Category::findOrFail($id);

    return view('categories.edit', ['category' => $category_to_edit]);
}
```

Dalam kode di atas kita mencari **Category** yang id nya bernilai sesuai dengan nilai dari **\$id** (route parameter), kemudian kita lempar data tersebut sebagai variabel **\$category** ke view **categories.edit**. View tersebut belum ada, jadi kita buat terlebih dahulu file tersebut di **resources/views/categories/edit.blade.php** lalu kita buat form edit di section content seperti ini:

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

@section('title') Edit Category @endsection

@section('content')
<div class="col-md-8">
<form
    action="{{ route('categories.update', ['id' => $category->id])}}"
    enctype="multipart/form-data"
    method="POST"
    class="bg-white shadow-sm p-3"
>

    @csrf

    <input
        type="hidden"
        value="PUT"
        name="_method">

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

Saya tidak perlu menjelaskan tentang `@extends` dan `@section` saya kira kamu semua sudah memahaminya karena telah berulangkali kita bahas. Begitu pula untuk kerangka form edit, akan tetapi untuk form tidak ada salahnya kita refresh lagi pengetahuan kita.

Untuk melakukan update kita akan mengirimkan ke route **categories.update** atau url `http://larashop.test/categories/{id}` dengan method PUT. Ingat bahwa kita tidak bisa langsung memberikan nilai PUT pada form kita, memang begitulah bagaimana Laravel menghandle request selain GET dan POST.

Agar laravel menerima request kita sebagai method PUT, kita tetap menggunakan form dengan method POST, lalu kita buat hidden input dengan nama **\_method** bernilai PUT. Seperti yang kita lihat pada form di atas.

Kemudian kita perlu menggunakan helper `@csrf` agar request kita valid.

Selanjutnya kita bisa menuliskan kode untuk masing-masing field input, karena ini form edit jangan lupa untuk mengisikan atribut value di masing-masing input berdasarkan data **\$category** yang diedit.

Menambahkan field text **name** dan **slug** caranya seperti ini:

```

<label>Category name</label> <br>
<input
  type="text"
  class="form-control"
  value="{{ $category->name }}"
  name="name">
<br><br>

<label>Category slug</label>
<input
  type="text"
  class="form-control"
  value="{{ $category->slug }}"
  name="slug">
<br><br>

```

Sementara itu untuk file image, selain input bertipe file picker kita juga perlu menampilkan gambar kategori yang saat ini dimiliki oleh kategori yang diedit, dan juga memberikan keterangan "Kosongkan jika tidak ingin mengubah gambar":

```

@if($category->image)
  <span>Current image</span><br>
  
  <br><br>
@endif

<input
  type="file"
  class="form-control"
  name="image">
<small class="text-muted">Kosongkan jika tidak ingin mengubah
gambar</small>
<br><br>

```

File view categories.blade.php kita kini secara lengkap menjadi seperti ini:

```

@extends('layouts.global')

@section('title') Edit Category @endsection

@section('content')
  <div class="col-md-8">
    <form
      action="{{ route('categories.update', ['id' => $category->id]) }}"
      enctype="multipart/form-data"
      method="POST"
      class="bg-white shadow-sm p-3"
    >

```

```

@csrf

<input
    type="hidden"
    value="PUT"
    name="_method">

<label>Category name</label> <br>
<input
    type="text"
    class="form-control"
    value="{{ $category->name }}"
    name="name">
<br><br>

<label>Category slug</label>
<input
    type="text"
    class="form-control"
    value="{{ $category->slug }}"
    name="slug">
<br><br>

<label>Category image</label><br>
@if($category->image)
    <span>Current image</span><br>
    
    <br><br>
@endif
<input
    type="file"
    class="form-control"
    name="image">
Kosongkan jika tidak ingin mengubah gambar
<br><br>

<input type="submit" class="btn btn-primary" value="Update">

</form>
</div>
@endsection

```

#### Menangkap request untuk update

Form yang kita buat di `categories.blade.php` mengirimkan data ke `http://larashop.test/categories/{id}` dengan method PUT alias route `categories.update`. Route tersebut menggunakan `CategoryController@store` maka di action `store()` tersebutlah kita akan menangkap data-data dari form edit untuk melakukan update `Category`

Buka file `app/Http/Controllers/CategoryController.php` lalu pada action `store()` kita tangkap masing-masing field text seperti ini:

```
public function update (Request $request, $id){  
    $name = $request->get('name');  
    $slug = $request->get('slug');  
}
```

Setelah itu kita cari `Category` yang sedang diedit seperti ii:

```
public function update (Request $request, $id){  
    $name = $request->get('name');  
    $slug = $request->get('slug');  
  
    $category = \App\Category::findOrFail($id);  
}
```

Setelah itu kita berikan field-field yang diedit dengan nilai dari request yang kita tangkap seperti ini:

```
public function update (Request $request, $id){  
    $name = $request->get('name');  
    $slug = $request->get('slug');  
  
    $category = \App\Category::findOrFail($id);  
  
    $category->name = $name;  
    $category->slug = $slug;  
}
```

Sebelum kita menyimpan perubahan terhadap model `Category` yang sedang diedit, kita perlu mengecek apakah ada file image yang diupload jika ada kita juga perlu mengupdate field `image`, dan sebelum menyimpan image baru kita juga perlu mengecek apakah kategori yang diedit ini memiliki image sebelumnya di server, jika ada kita hapus file tersebut, baru setelah itu assign image path yang baru seperti ini:

```
public function update (Request $request, $id){  
    $name = $request->get('name');  
    $slug = $request->get('slug');  
  
    $category = \App\Category::findOrFail($id);  
  
    $category->name = $name;  
    $category->slug = $slug;  
  
    if($request->file('image')){  
        if($category->image && file_exists(storage_path('app/public/' .
```

```
$category->image)) {
    \Storage::delete('public/' . $category->name);
}

$new_image = $request->file('image')->store('category_images',
'public');

$category->image = $new_image;
}

$category->save();
}
```

Ingat bahwa di tabel **categories** terdapat field **updated\_by**, field ini harus kita isi terlebih dahulu dengan ID dari user yang sedang login seperti ini:

```
$category->updated_by = \Auth::user()->id;
```

Selain itu kita juga perlu mengupdate slug berdasarkan nama baru category yang disubmit dari form edit, seperti ini:

```
$category->slug = str_slug($name);
```

Jika sudah maka kita bisa melakukan save terhadap model **Category** yang sedang diedit itu:

```
$category->save();
```

Lalu kita redirect kembali ke halaman form edit

```
return view('categories.edit', ['id' => $category->id]);
```

Kita tambahkan pula status bahwa update berhasil:

```
return redirect()->route('categories.edit', ['id' => $id])->with('status',
'Category successfully updated');
```

Sehingga hasil akhir dari action **update()** di controller **CategoryController** terlihat seperti ini:

```
public function update(Request $request, $id){
    $name = $request->get('name');
```

```

$slug = $request->get('slug');

$category = \App\Category::findOrFail($id);

$category->name = $name;
$category->slug = $slug;

if($request->file('image')){
    if($category->image && file_exists(storage_path('app/public/' .
$category->image))){
        \Storage::delete('public/' . $category->name);
    }

    $new_image = $request->file('image')->store('category_images',
'public');

    $category->image = $new_image;
}

$category->updated_by = \Auth::user()->id;

$category->slug = str_slug($name);

$category->save();

return redirect()->route('categories.edit', ['id' => $id]);
}

```

Jangan lupa di file view `categories/edit.blade.php` kita tambahkan kode untuk menampilkan pesan bahwa update berhasil ini di atas kode form edit:

```

@if(session('status'))
<div class="alert alert-success">
{{session('status')}}
</div>
@endif

```

Dengan begitu, maka telah selesai fitur edit category kita. Jika kita melakuka update sebuah category dan berhasil kita akan menyaksikan tampilan berikut ini:

Category successfully updated

Category name

Sports One

Category slug

sports-one

Category image

Current image



No file chosen

Kosongkan jika tidak ingin menambah gambar

### Fitur show detail category

Fitur show category menggunakan route `http://larashop.test/categories/{id}` dengan method GET alias route `categories.show`. Route tersebut menggunakan `CategoryController@show`.

Kita buka controller `CategoryController.php` tambahkan kode ini di action `show`

```
public function show($id){
    $category = \App\Category::findOrFail($id);

    return view('categories.show', ['category' => $category]);
}
```

Kemudian kita buat file view `categories/show.blade.php` di `resources/views` dan kita isikan kode untuk menampilkan detail kategori:

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

@section('title') Detail Category @endsection
```

```

@section('content')
    <div class="col-md-8">
        <div class="card">
            <div class="card-body">
                <label><b>Category name</b></label><br>
                {{$category->name}}
                <br><br>

                <label><b>Category slug</b></label><br>
                {{$category->slug}}
                <br><br>

                <label><b>Category image</b></label><br>
                @if($category->image)
                    
                @endif
            </div>
        </div>
    </div>
@endsection

```

Langkah terakhir di fitur detail category adalah kita buat tombol detail dari halaman list category. Buka file view `categories/index.blade.php` dan tambahkan kode berikut ini:

```

<a href="{{route('categories.show', ['id' => $category->id])}}" class="btn btn-primary"> Show </a>

```

Kode tersebut kamu letakan tepat di bawah kode untuk menampilkan tombol edit.

## Fitur delete category

Untuk kategori kita akan menggunakan konsep soft delete, yaitu ketika dihapus `Category` tidak langsung dihapus dari database, akan tetapi ditandai saja bahwa status dari sebuah kategori tersebut sedang didelete atau dalam tong sampah (trashed).

Status tersebut diindikasikan oleh field `deleted_at` yang memiliki nilai. Jika `deleted_at` bernilai NULL maka kategori sedang aktif (tidak di tong sampah), sebaliknya jika field `deleted_at` memiliki nilai yaitu tanggal kapan data tersebut dihapus maka dia dianggap di tong sampah.

Secara default fitur soft delete itu tidak aktif, untuk itu untuk menggunakan fitur tersebut kita perlu mengaktifkannya di model Category.

### Menyesuaikan model Category

Buka model `app/Category.php` lalu tambahkan kode ini di bagian atas file tersebut:

```
use Illuminate\Database\Eloquent\SoftDeletes;
```

Setelah itu di class **Category** kita tambahkan kode ini:

```
use SoftDeletes();
```

Kini model **Category** kita terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Category extends Model
{
    use SoftDeletes;
}
```

Dengan begini maka model **Category** telah siap untuk soft delete. Satu lagi yang perlu dipastikan adalah tabel **categories** memiliki field **deleted\_at** dan kita telah membuatnya pada migration **create\_categories\_table.php** di awal materi CRUD Categories ini. Jadi semua sudah siap!

#### Delete

Langkah selanjutnya adalah kita buat tombol untuk delete di halaman categories list. Buka file view **categories/index.blade.php** pada kolom actions, tepat di bawah kode untuk tombol detail kita tambahkan kode untuk menampilkan tombol delete seperti ini:

```
<form
    class="d-inline"
    action="{{route('categories.destroy', ['id' => $category->id])}}"
    method="POST"
    onsubmit="return confirm('Move category to trash?')"
    >

    @csrf

    <input
        type="hidden"
        value="DELETE"
        name="_method">
```

```

<input
    type="submit"
    class="btn btn-danger btn-sm"
    value="Trash">

</form>

```

**Menangkap request delete**

Setelah tombol delete berhasil kita buat selanjutnya adalah menangkap request tersebut di `CategoryController@destroy` dan melakukan delete terhadap model `Category` berdasarkan route parameter `id`. Buka file `CategoryController.php` dan pada action `destroy()` kita cari `Category` seperti ini:

```

public function destroy($id){
    $category = \App\Category::findOrFail($id);
}

```

Lalu kita delete `Category` tersebut menggunakan method `delete()` seperti ini:

```

public function destroy($id){
    $category = \App\Category::findOrFail($id);

    $category->delete();
}

```

Kemudian langkah terakhir jika telah berhasil delete kita juga redirect kembali ke halaman categories list dengan pesan status bahwa delete berhasil seperti ini:

```

public function destroy($id){
    $category = \App\Category::findOrFail($id);

    $category->delete();

    return redirect()->route('categories.index')
        ->with('status', 'Category successfully moved to trash');
}

```

Selanjutnya tambahkan kode untuk membaca pesan status di halaman categories list, buka file view `categories/index.blade.php` lalu tambahkan kode berikut:

```

@if(session('status'))
<div class="row">
    <div class="col-md-12">

```

```
<div class="alert alert-warning">
    {{session('status')}}
</div>
</div>
@endif
```

Letakan kode tersebut di atas kode tabel daftar kategori. Kini fitur soft delete **Category** telah selesai.

#### Show soft deleted category

**Category** yang telah di *soft delete* sebelumnya masih ada di database. Untuk itu kita perlu sebuah fitur untuk menampilkan **Category** apa saja yang statusnya *soft deleted*.

Untuk menampilkan daftar **Category** yang sedang *soft delete* kita akan menggunakan route <http://larashop.test/categories/trash>. Oleh karenanya mari kita buat sebuah action di **CategoryController** yaitu **trash()** seperti ini:

file **CategoryController.php**

```
public function trash(){
}
```

Setelah itu kita buat route di **routes/web.php** seperti ini:

```
Route::get('/categories/trash', 'CategoryController@trash')-
>name('categories.trash');
```

Perlu diingat kode route di atas harus diletakkan sebelum route ini:

```
Route::resource('categories', 'CategoryController');
```

Jika tidak maka route yang kita definisikan tidak akan dideteksi oleh Laravel, sehingga jika kita mengakses <http://larashop.test/categories/trash> akan error not found.

Setelah itu kita tambahkan navigasi di halaman categories list, buka file view **categories/index.blade.php** dan ubah kode filter kita yang sebelumnya seperti ini:

```
<div class="row">
    <div class="col-md-6">
        <form action="{{route('categories.index')}}">
            <div class="input-group">
```

```

<input
    type="text"
    class="form-control"
    placeholder="Filter by category name"
    value="{{Request::get('name')}}"
    name="name">

<div class="input-group-append">
    <input
        type="submit"
        value="Filter"
        class="btn btn-primary">
</div>
</div>

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

```

Menjadi seperti ini:

```

<div class="row">
    <div class="col-md-6">
        <form action="{{route('categories.index')}}">

            <div class="input-group">
                <input
                    type="text"
                    class="form-control"
                    placeholder="Filter by category name"
                    value="{{Request::get('name')}}"
                    name="name">

                <div class="input-group-append">
                    <input
                        type="submit"
                        value="Filter"
                        class="btn btn-primary">
                </div>
            </div>

        </form>
    </div>

    <div class="col-md-6">
        <ul class="nav nav-pills card-header-pills">
            <li class="nav-item">
                <a class="nav-link active" href="{{route('categories.index')}}">Published</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="">

```

```

{{route('categories.trash')}}>Trash</a>
    </li>
</ul>
</div>

</div>

```

Dengan kode di atas kini kita akan mendapatkan navigasi untuk ke halaman trash di sebelah kanan input filter seperti ini:

The screenshot shows a 'Category List' page. At the top, there is a search bar with the placeholder 'Filter by category name'. To its right is a blue 'Filter' button. Below the search bar are two blue tabs: 'Published' and 'Trash'. The 'Trash' tab is currently active.

Coba klik link **Trash** di atas maka kamu akan diarahkan ke <http://larashop.test/categories/trash> dan mendapati blank page. Maka kini kita perlu mengubah **CategoryController@trash** untuk mendapatkan data berupa daftar **Category** yang statusnya sedang *soft deleted* dan melemparnya ke view **categories.trash**

file **CategoryController.php** action **trash()**

```

public function trash(){
    $deleted_category = \App\Category::onlyTrashed()->paginate(10);
    return view('categories.trash', ['categories' => $deleted_category]);
}

```

Kode tersebut melakukan query ke database **categories** menggunakan model **Category** dan kita gunakan method **onlyTrashed()** untuk mendapatkan hanya kategori yang status nya *soft delete* yaitu yang field **deleted\_at** nya tidak NULL.

Setelah itu kita lempar sebagai variable **\$categories** ke view **categories.index**. Akan tetapi kita belum memiliki view tersebut untuk itu silahkan buat file terlebih dahulu yaitu **resources/views/categories/trash.blade.php** lalu tambahkan kode untuk menampilkan daftar kategori yang dihapus seperti ini:

```

@extends('layouts.global')

@section('title') Trashed Categories @endsection

@section('content')
<div class="row">
    <div class="col-md-6">

```

```

<form action="{{route('categories.index')}}">

    <div class="input-group">
        <input
            type="text"
            class="form-control"
            placeholder="Filter by category name"
            value="{{Request::get('name')}}"
            name="name">

        <div class="input-group-append">
            <input
                type="submit"
                value="Filter"
                class="btn btn-primary">
        </div>
    </div>

</form>
</div>

<div class="col-md-6">
    <ul class="nav nav-pills card-header-pills">
        <li class="nav-item">
            <a class="nav-link" href="{{route('categories.index')}}">Published</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="{{route('categories.trash')}}">Trash</a>
        </li>
    </ul>
</div>

</div>

<hr class="my-3">

<div class="row">
    <div class="col-md-12">
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Nama</th>
                    <th>Slug</th>
                    <th>Image</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody>
                @foreach($categories as $category)
                    <tr>
                        <td>$category->name</td>
                        <td>$category->slug</td>

```

```

<td>
    @if($category->image)
        
    @endif
</td>
<td>
    [TODO: actions]
</td>
</tr>
@endforeach
</tbody>
<tfoot>
<tr>
    <td colSpan="10">
        {{$categories->appends(Request::all())->links()}}
    </td>
</tr>
</tfoot>
</table>
</div>
</div>

@endsection

```

Silahkan klik link trash maka kini kamu akan mendapat tampilan kurang lebih seperti ini:

## Trashed Categories

Filter by category name		Filter	Published	Trash
Nama	Slug	Image	Action	
Sports One	sports-one		[TODO: actions]	

## Restore

Selanjutnya kita akan membolehkan **Category** yang statusnya dihapus / *soft deleted* untuk direstore agar kembali menjadi **Category** aktif. Untuk melakukan hal tersebut kita menggunakan route <http://larashopt.test/categories/{id}/restore> untuk itu mari kita buat sebuah action terlebih dahulu di **CategoryController** dengan nama **restore()** seperti ini:

```
public function restore($id){
```

}

Kemudian kita buat route ke action controller tersebut di file `routes/web.php` seperti ini:

```
Route::get('/categories/{id}/restore', 'CategoryController@restore')->name('categories.restore');
```

Route di atas kita beri nama 'categories.restore' agar kita bisa menggunakan helper `route('categories.restore')` di view untuk menggenerate linknya.

Dan seperti biasa karena kita menggunakan resource controller, maka definisi kode route di atas harus diletakkan sebelum kode route resource ini:

```
Route::resource('categories', 'CategoryController');
```

Kemudian kita buat tombol restore di halaman trashed, buka file view `categories/trash.blade.php` dan tambahkan kode ini:

```
<a href="{{route('categories.restore', ['id' => $category->id])}}>
    Restore
</a>
```

Kode tersebut menggantikan kode ini:

```
[TODO: actions]
```

Selanjutnya mari kita restore `Category` pada action `restore()` buka kembali `CategoryController.php` lalu ubah kode action restore kita:

```
public function restore($id){
    $category = \App\Category::withTrashed()->findOrFail($id);

    if($category->trashed()){
        $category->restore();
    } else {
        return redirect()->route('categories.index')
            ->with('status', 'Category is not in trash');
    }

    return redirect()->route('categories.index')
        ->with('status', 'Category successfully restored');
}
```

Perhatikan bahwa untuk mencari **Category** kita menggunakan method eloquent **withTrashed()** karena kita ingin mencari semua **Category** baik yang aktif maupun yang ada di trash / *soft deleted*. Jika tidak menggunakan **withTrashed()** akan sangat mungkin kita mendapatkan error not found.

Kini fitur restore delete category kita telah selesai dan bisa mulai kita gunakan.

#### Delete permanent

Selanjutnya dari halaman trash kita ingin memberikan fitur bagi user untuk menghapus **Category** secara permanent, artinya bukan *soft deleted* lagi tetapi langsung di hapus record di databasenya.

Untuk kepentingan delete permanent kita akan menggunakan route `http://larashop.test/categories/{id}/delete-permanent` dengan method `DELETE`. Route tersebut akan mengarah ke action **deletePermanent()** di **CategoryController**. Untuk itu mari kita buat action tersebut seperti ini:

```
public function deletePermanent($id){  
}
```

Setelah itu kita buat route di **routes/web.php** seperti ini:

```
Route::delete('/categories/{id}/delete-permanent',  
'CategoryController@deletePermanent')->name('categories.delete-permanent');
```

Setelah itu kita tambahkan kode untuk menangkap request delete permanent di action controller **deletePermanent** tadi seperti ini:

```
public function deletePermanent($id){  
    $category = \App\Category::withTrashed()->findOrFail($id);  
  
    if(!$category->trashed()){  
        return redirect()->route('categories.index')  
            ->with('status', 'Can not delete permanent active category');  
    } else {  
        $category->forceDelete();  
  
        return redirect()->route('categories.index')  
            ->with('status', 'Category permanently deleted');  
    }  
}
```

Dengan kode di atas kita tetap menggunakan method **withTrashed()** saat melakukan query model **Category** lalu kita cek jika kategori yang akan dihapus permanent statusnya tidak di tong sampah / trashed / *soft delete* maka kita stop operasi hapus permanent ini dan redirect dengan pesan tidak bisa menghapus kategori yang sedang aktif.

Selanjutnya jika kategori tersebut memang sedang dalam tong sampah maka kita hapus secara permanent menggunakan method `forceDelete()` kemudian redirect kembali ke halaman categories list dengan pesan bahwa kategori telah dihapus secara permanent.

Setelah itu kita tambahkan tombol delete permanent di halaman categories trash, buka file view `categories/trash.blade.php` lalu tambahkan kode berikut ini:

```
<form
  class="d-inline"
  action="{{route('categories.delete-permanent', ['id' => $category->id])}}"
  method="POST"
  onsubmit="return confirm('Delete this category permanently?')"
>

@csrf

<input
  type="hidden"
  name="_method"
  value="DELETE"/>

<input
  type="submit"
  class="btn btn-danger btn-sm"
  value="Delete"/>

</form>
```

Kita harus menggunakan form, karena kita akan mengirim request delete permanent dengan method `DELETE`. Sekarang jika kamu mengeklik salah satu tombol delete permanent di halaman categories trash, kamu akan benar-benar menghapus sebuah kategori dari database dan tidak bisa merestore kembali.

Satu hal lagi yang perlu kita lakukan yaitu menambah menu "Manage categories" di sidebar, caranya buka kembali file layout kita `resources/views/layouts/global.blade.php` lalu tambahkan kode berikut ini:

```
<li><a href="{{route('categories.index')}}"><span class="oi oi-tag"></span>
  Manage categories</a></li>
```

Letakkan kode di atas tepat di bawah kode ini:

```
<li><a href="{{route('users.index')}}"><span class="oi oi-people"></span>
  Manage users</a></li>
```

Oia satu lagi kita belum membuat tombol `Create category` di halaman daftar kategori. Kita perlu membuatnya, untuk itu buka kembali file `resources/views/index.blade.php` lalu tambahkan kode

berikut ini:

```
<div class="row">
  <div class="col-md-12 text-right">
    <a href="{{route('categories.create')}}" class="btn btn-primary">Create
category</a>
  </div>
</div>
<br>
```

Sehingga kini file view `resources/views/index.blade.php` menjadi seperti ini:

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

@section('title') Category list @endsection

@section('content')
<div class="row">
  <div class="col-md-6">
    <form action="{{route('categories.index')}}">

      <div class="input-group">
        <input
          type="text"
          class="form-control"
          placeholder="Filter by category name"
          value="{{Request::get('name')}}"
          name="name">

      <div class="input-group-append">
        <input
          type="submit"
          value="Filter"
          class="btn btn-primary">
      </div>
    </div>

    </form>
  </div>

  <div class="col-md-6">
    <ul class="nav nav-pills card-header-pills">
      <li class="nav-item">
        <a class="nav-link active" href=
{{route('categories.index')}}>Published</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href=
{{route('categories.trash')}}>Trash</a>
      </li>
    </ul>
  </div>
</div>
```

```

        </ul>
    </div>

</div>

<hr class="my-3">

@if(session('status'))
    <div class="row">
        <div class="col-md-12">
            <div class="alert alert-warning">
                {{session('status')}}
            </div>
        </div>
    </div>
@endif

<br>

<div class="row">
    <div class="col-md-12 text-right">
        <a href="{{route('categories.create')}}" class="btn btn-primary">Create category</a>
    </div>
</div>
<br>

<div class="row">
    <div class="col-md-12">
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th><b>Name</b></th>
                    <th><b>Slug</b></th>
                    <th><b>Image</b></th>
                    <th><b>Actions</b></th>
                </tr>
            </thead>

            <tbody>
                @foreach ($categories as $category)
                    <tr>
                        <td>{{$category->name}}</td>
                        <td>{{$category->slug}}</td>
                        <td>
                            @if($category->image)
                                
                            @else
                                No image
                            @endif
                        </td>
                        <td>
                            <a
                                href="{{route('categories.edit', ['id' => $category->id])}}>

```

```

        class="btn btn-info btn-sm"> Edit </a>

        <a href="{{route('categories.show', ['id' => $category->id])}}>
            class="btn btn-primary"> Show </a>

        <form
            class="d-inline"
            action="{{route('categories.destroy', ['id' => $category->id])}}"
            method="POST"
            onsubmit="return confirm('Move category to trash?')"
        >

            @csrf

            <input
                type="hidden"
                value="DELETE"
                name="_method">

            <input
                type="submit"
                class="btn btn-danger btn-sm"
                value="Trash">

        </form>
    </td>
</tr>
@endforeach
</tbody>
<tfoot>
    <tr>
        <td colSpan="10">
            {{$categories->appends(Request::all())->links()}}
        </td>
    </tr>
</tfoot>

</table>
</div>
</div>
@endsection

```

Dengan demikian maka telah selesai fitur CRUD Category di aplikasi kita.

## Manage Book

Kita telah sampai pada model **Book**, model central yang menjadi tokoh utama dalam aplikasi kita. Karena toko kita menjual produk buku. Mari kita mulai.

## Membuat migration

### Membuat migration untuk tabel books

Pertama kita akan membuat file migration untuk membuat tabel **book** dan strukturnya. Desain tabel nya adalah seperti ini:

books	
+ id:	Integer {PK}
+ title:	String
+ slug:	String{U}
+ description:	String
+ author:	String
+ publisher:	String
+ cover:	String
+ price:	Float
+ views:	Integer
+ stock:	Integer
+ status:	Enum {PUBLISH   DRAFT}
+ created_at:	timestamps
+ updated_at:	timestamps
+ created_by:	Integer
+ deleted_by:	Integer
+ updated_by:	Integer
+ deleted_at:	Integer

Selanjutnya, buat file migration tersebut dengan perintah ini:

```
php artisan make:migration create_books_table
```

Jika sudah buka file migration yang baru saja dibuat di direktori **database/migrations** dan kita buat struktur tabel menggunakan Schema builder agar sesuai desain database seperti ini:

file **....\_create\_books\_table.php**

```
<?php

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

class CreateBooksTable extends Migration
{
```

```

/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title');
        $table->string('slug');
        $table->text('description');
        $table->string('author');
        $table->string('publisher');
        $table->string('cover');
        $table->float('price');
        $table->integer('views')->default(0)->unsigned();
        $table->integer('stock')->default(0)->unsigned();
        $table->enum('status', ['PUBLISH', 'DRAFT']);
        $table->integer('created_by');
        $table->integer('updated_by')->nullable();
        $table->integer('deleted_by')->nullable();

        $table->timestamps();
        $table->softDeletes();
    });
}

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

```

Setelah itu jalankan perintah migrate

```
php artisan migrate
```

### Membuat migration untuk relationship ke tabel categories

Selanjutnya sebelum kita membuat model **Book**, kita juga perlu merelasikan tabel **books** kita dengan tabel **categories**. Relasi antara **book** dan **categories** adalah **many-to-many** relationship, untuk itu kita perlu juga membuat pivot tabel dengan nama **book\_category**. Adapun tabel tersebut sesuai desain database kita strukturnya adalah seperti ini:

book_category
+ id: Integer{PK}
+ book_id: Integer
+ category_id: Integer
+ created_at: Integer
+ updated_at: Integer

Buat file migrationnya

```
php artisan make:migration create_book_category_table
```

Lalu buka file yang baru dibuat tersebut yang terletak di direktori **database/migrations**

```
<?php

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

class CreateBookCategoryTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('book_category', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('book_id')->unsigned()->nullable();
            $table->integer('category_id')->unsigned()->nullable();
            $table->timestamps();

            $table->foreign('book_id')->references('id')->on('books');
            $table->foreign('category_id')->references('id')-
>on('categories');
        });
    }

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

```

    }
}
```

Pada method `up()` selain mendefinisikan struktur tabel, kita juga mendefinisikan dua foreign key yaitu `book_id` dan `category_id` menggunakan Schema builder method `foreign()`. Sebelum menggunakan method `foreign()` tersebut kita harus memastikan hal-hal ini agar tidak terjadi error:

1. Pastikan tabel yang akan direferensikan sudah ada terlebih dahulu, itulah kenapa kita membuat tabel `categories` dan `books` dulu, baru mendefinisikan relationship di tabel.
2. Pastikan bahwa saat membuat field yang akan menjadi foreign (`book_id` dan `category_id`) menggunakan method `unsigned()`.

Jangan lupa ya, jalankan perintah migrate untuk mengeksekusi file migration yang baru kita buat:

```
php artisan migrate
```

Kini kita siap membuat model `Book` dan selanjutnya membuat relationship many-to-many ke tabel `Category`

## Membuat model Book

Mari kita buat model `Book` dengan perintah berikut:

```
php artisan make:model Book
```

Dengan begini model `Book` sebetulnya telah siap kita gunakan, tapi ingat jika kita ingin menggunakan fitur relationship, maka kita definisikan terlebih dahulu relationshipnya, yang sudah kita setup tabelnya adalah relationship many-to-many dengan model `Category`. Mari kita definisikan relationship tersebut di kedua model tersebut.

## Mendefinisikan relationship

### Mendefinisikan relationship di model Book

Buka file `app/Book.php` lalu tambahkan kode relationship ke model `Category` seperti ini:

```

public function categories(){
    return $this->belongsToMany('App\Category');
}
```

Sehingga model `Book` sekarang terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    public function categories(){
        return $this->belongsToMany('App\Category');
    }
}
```

#### Mendefinisikan relationship di model Category

Begitu pula kita perlu membuka kembali model **Category** yang terletak di `app/Category.php` dan mendefinisikan relationship dengan model **Book** seperti ini:

```
public function books(){
    return $this->belongsToMany('App\Book');
}
```

Sehingga model **Category** kini terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Category extends Model
{
    use SoftDeletes;

    public function books(){
        return $this->belongsToMany('App\Book');
    }
}
```

Baiklah, kini model **Book** telah siap kita gunakan untuk membuat fitur CRUD.

#### CRUD Book

Seperti biasa langkah pertama untuk membuat fitur CRUD adalah membuat controller dan route resource. Buat **BookController** dengan perintah berikut ini:

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

Setelah itu buka `routes/web.php` dan tambahkan kode berikut ini:

```
Route::resource('books', 'BookController');
```

## Fitur create book

Buat view `create.blade.php` pada direktori `resources/views/books` lalu berikan kerangka dasar view aplikasi seperti ini:

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

@section('title') Create book @endsection

@section('content')
    Form create di sini
@endsection
```

Selanjutnya kita ubah action `BookController@create` agar mengembalikan view yang baru saja kita buat, buka file `BookController.php` lalu ubah action `create()` seperti ini:

```
public function create(){
    return view('books.create');
}
```

Setelah itu buka file view `books/create.blade.php` lalu ubah agar menjadi seperti ini untuk menampilkan form create book:

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

@section('title') Create book @endsection

@section('content')
<div class="row">
    <div class="col-md-8">
        <form
            action="{{route('books.store')}}"
            method="POST"
            enctype="multipart/form-data"
            class="shadow-sm p-3 bg-white"
        >
```

```

@csrf

    <label for="title">Title</label> <br>
    <input type="text" class="form-control" name="title"
placeholder="Book title">
    <br>

    <label for="cover">Cover</label>
    <input type="file" class="form-control" name="cover">
    <br>

    <label for="description">Description</label><br>
    <textarea name="description" id="description" class="form-control"
placeholder="Give a description about this book"></textarea>
    <br>

    <label for="stock">Stock</label><br>
    <input type="number" class="form-control" id="stock" name="stock"
min=0 value=0>
    <br>

    <label for="author">Author</label><br>
    <input type="text" class="form-control" name="author" id="author"
placeholder="Book author">
    <br>

    <label for="publisher">Publisher</label> <br>
    <input type="text" class="form-control" id="publisher"
name="publisher" placeholder="Book publisher">
    <br>

    <label for="Price">Price</label> <br>
    <input type="number" class="form-control" name="price" id="price"
placeholder="Book price">
    <br>

    <button
        class="btn btn-primary"
        name="save_action"
        value="PUBLISH">Publish</button>

    <button
        class="btn btn-secondary"
        name="save_action"
        value="DRAFT">Save as draft</button>
    </form>
</div>
</div>
@endsection

```

Kode di atas tidak jauh berbeda dari apa yang telah kita pelajari pada materi CRUD-CRUD sebelumnya. Hanya satu hal yang berbeda yaitu kita menggunakan dua tombol submit di form tersebut. Satu tombol

"Publish" untuk menyimpan dan langsung publish buku yang satu adalah tombol "Save as draft" untuk menyimpan buku tapi statusnya masih draft belum published.

Berikut potongan kode dua tombol tersebut:

```
<button
  class="btn btn-primary"
  name="save_action"
  value="PUBLISH">Publish</button>

<button
  class="btn btn-secondary"
  name="save_action"
  value="DRAFT">Save as draft</button>
```

Perlu diperhatikan bahwa kode di atas memiliki nama yang sama yaitu "save\_action". Hal ini penting untuk membedakan antara keinginan user untuk langsung publish atau save as draft, untuk tombol publish **value** nya adalah "PUBLISH", sementara itu untuk tombol save as draft **value** nya adalah "DRAFT". Kita akan menggunakan nilai dari input "save\_action" di controller action **store**.

#### Menangkap request create user di controller action

Buka file **BookController.php** lalu kita ubah action yang bertanggung jawab untuk pembuatan **book** baru yaitu method **store()**, kita tangkap request pembuatan **book** seperti ini:

```
public function store(Request $request){
    $new_book = new \App\Book;
    $new_book->title = $request->get('title');
    $new_book->description = $request->get('description');
    $new_book->author = $request->get('author');
    $new_book->publisher = $request->get('publisher');
    $new_book->price = $request->get('price');
    $new_book->stock = $request->get('stock');

    $new_book->status = $request->get('save_action');
}
```

Kode di atas menangkap request create book dan menyimpannya ke dalam model **Book** baru yang belum ada di database. Perhatikan bagaimana kita memberikan nilai pada properti **status** dari request bernama 'save\_action' alias tipe button yang diklik oleh user. Ingat bahwa nilainya hanya dua yaitu PUBLISH atau DRAFT sehingga keduanya sudah sesuai dengan tipe field **status** pada tabel **books** kita.

Selanjutnya kita akan menangkap cover image dari request ini:

```
$cover = $request->file('cover');

if($cover){
```

```
$cover_path = $cover->store('book-covers', 'public');

$new_book->cover = $cover_path;
}
```

Selanjutnya kita juga perlu memberikan nilai terhadap properti-properti yang tidak ada di form, yaitu `slug` dan `created_by`. Nilai dari field `slug` akan kita isi dengan menggenerate slug berdasarkan `title` dari book seperti ini:

```
$new_book->slug = str_slug($request->get('title'));
```

Kita sekali lagi menggunakan helper `str_slug()` seperti yang telah kita lakukan saat menggenerate slug untuk model `Category`.

Selanjutnya kita isi field `created_by` dengan ID dari user yang sedang login dan mensubmit data tersebut seperti ini:

```
$new_book = \Auth::user()->id;
```

Selanjutnya kita simpan data model tersebut ke database dengan method `save()` seperti ini:

```
$new_book->save();
```

Dan setelah itu kita redirect ke halaman create books dengan pesan status berhasil berdasarkan save action seperti ini:

```
if($request->get('save_action') == 'PUBLISH'){
    return redirect()->route('books.create')->with('status', 'Book
successfully saved and published');
} else {
}

return redirect()->route('books.create')->with('status', 'Book saved as
draft');
}
```

Sehingga keseluruhan kode `BookController@store` terlihat seperti ini:

```
public function store(Request $request){
    $new_book = new \App\Book;
    $new_book->title = $request->get('title');
    $new_book->description = $request->get('description');
```

```

$new_book->author = $request->get('author');
$new_book->publisher = $request->get('publisher');
$new_book->price = $request->get('price');
$new_book->stock = $request->get('stock');

$new_book->status = $request->get('save_action');

$cover = $request->file('cover');

if($cover){
    $cover_path = $cover->store('book-covers', 'public');

    $new_book->cover = $cover_path;
}

$new_book->slug = str_slug($request->get('title'));

$new_book->created_by = \Auth::user()->id;

$new_book->save();

if($request->get('save_action') == 'PUBLISH'){
    return redirect()
        ->route('books.create')
        ->with('status', 'Book successfully saved and published');
} else {
    return redirect()
        ->route('books.create')
        ->with('status', 'Book saved as draft');
}
}
}

```

Setelah itu kita tambahkan kode untuk menampilkan status pembuatan buku di file view `books/create.blade.php` seperti ini:

```

@if(session('status'))
<div class="alert alert-success">
    {{session('status')}}
</div>
@endif

```

Letakkan kode di atas tepat di atas kode `<form>`

Kini fitur form create book sudah bisa kita gunakan, coba buat buku baru dan isi semua field, kamu akan berhasil membuat data buku.

Selanjutnya kita akan membuat halaman daftar buku, tapi tunggu dulu, sebelum kita melangkah ke fitur tersebut, kita belum memberikan input field untuk mengisi kategori buku, padahal kita sudah membuat kategori dan sudah mendefinisikan relationship antara model `Book` dengan model `Category`. Oleh karenanya mari kita buat input tersebut.

## Fitur pilih kategori buku

Untuk fitur ini kita akan menggunakan bantuan jQuery plugin bernama `select2`. Oleh karenanya ada beberapa hal yang perlu kita lakukan antara lain:

- Include script js dan css dari plugin select2 Pertama kita buka file layout global kita yaitu

`resources/views/layouts/global.blade.php` dan tambahkan kode ini `@yield('footer-scripts')` tepat di atas kode penutup head (`</body>`)

Setelah itu di view create book yaitu `resources/views/books/create.blade.php` kita tambahkan kode ini setelah kode `@extends('layouts.global')`:

```
@section('footer-scripts')
<link href="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/css/select2.min.css" rel="stylesheet" />

<script src="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/js/select2.min.js"></script>
@endsection
```

- Membuat ajax endpoint untuk kategori

Ini karena select2 akan mengambil data via ajax, sehingga kita perlu siapkan route untuk ajax ini. Buka kembali `CategoryController.php` dan kita buat action baru yaitu `ajaxSearch()` dengan tambahkan kode berikut ini:

```
public function ajaxSearch(Request $request){
    $keyword = $request->get('q');

    $categories = \App\Category::where("name", "LIKE", "%$keyword%")-
    >get();

    return $categories;
}
```

Lalu buka file `routes/web.php` dan kita tambahkan jalur untuk mengakses `CategoryController@ajaxSearch` seperti ini:

```
Route::get('/ajax/categories/search',
'CategoryController@ajaxSearch');
```

Dengan setup seperti itu maka kini kita punya sebuah route untuk mencari kategori berdasarkan keyword. Route ini akan digunakan oleh select2 nantinya, adapun route tersebut adalah seperti ini:

`http://larashop.test/ajax/categories/search?q=keyword`

Setelah kita melakukan kedua hal di atas kita buka kembali file view

`resources/views/books/create.blade.php` lalu tambahkan input untuk memilih categori seperti ini:

```
<label for="categories">Categories</label><br>

<select
  name="categories[]"
  multiple
  id="categories"
  class="form-control">
</select>

<br><br/>
```

Letakkan kode di atas di bawah kode untuk menampilkan textarea description. Input untuk memilih categori di atas merupakan `select` karena input tersebut yang akan kita pasangkan dengan plugin `select2` kemudian kita beri attribute `multiple` dan name menggunakan array (`[]`) seperti ini `categories[]` karena kita akan mengizinkan lebih dari satu categori dipilih. Dan kita beri id `categories` sebagai selector untuk `select2`.

Nah, setelah input tersebut dibuat pada section `footer-script` di bagian bawah kode yang ada tambahkan kode ini:

```
<script>
$('#categories').select2({
  ajax: {
    url: 'http://larashop.test/ajax/categories/search',
    processResults: function(data){
      return {
        results: data.map(function(item){return {id: item.id, text:
item.name} })
      }
    }
  });
</script>
```

Kode di atas merupakan kode jquery `select2`, lihat kita menggunakan route ajax untuk mencari categori di situ. Inilah gunanya kita membuat route ajax sebelumnya.

Nah kini form create book kita sudah siap untuk mengizinkan user menambahkan categori buku. Selanjutnya kita perlu memodifikasi `BookController@store` agar dapat menangkap request `categories` dan menyimpannya ke model yang akan dibuat. Caranya adalah tambahkan kode berikut ini:

```
$new_book->categories()->attach($request->get('categories'));
```

Letakkan kode tersebut **SETELAH** kode ini:

```
$new_book->save();
```

## Fitur list book

Setelah kita berhasil membuat buku baru melalui form create book, mari kita buat halaman untuk menampilkan buku-buku tersebut.

Pertama tentu kita ubah terlebih dahulu `BookController@index` agar mengembalikan view `books.index` dengan data buku per halaman 10 item, caranya seperti ini:

File `BookController.php`

```
public function index(){
    $books = \App\Book::with('categories')->paginate(10);

    return view('books.index', ['books'=> $books]);
}
```

Perhatikan bahwa kita hanya mengambil buku yang statusnya PUBLISH saja menggunakan query `where('status', 'PUBLISH')`

Setelah itu kita buat view `books.index` alias file `resources/views/books/index.blade.php` dan kita isikan view untuk menampilkan daftar buku tersebut seperti ini:

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

@section('title') Books list @endsection

@section('content')
<div class="row">
    <div class="col-md-12">
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th><b>Cover</b></th>
                    <th><b>Title</b></th>
                    <th><b>Author</b></th>
                    <th><b>Status</b></th>
                    <th><b>Categories</b></th>
                    <th><b>Stock</b></th>
                    <th><b>Price</b></th>
                    <th><b>Action</b></th>
                </tr>
            </thead>
            <tbody>
                @foreach($books as $book)
                    <tr>
```

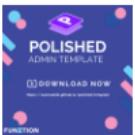
```

        <td>
            @if($book->cover)
                
            @endif
        </td>
        <td>{{$book->title}}</td>
        <td>{{$book->author}}</td>
        <td>
            @if($book->status == "DRAFT")
                <span class="badge bg-dark text-white">{{$book->status}}</span>
            @else
                <span class="badge badge-success">{{$book->status}}</span>
            @endif
        </td>
        <td>
            <ul class="pl-3">
                @foreach($book->categories as $category)
                    <li>{{$category->name}}</li>
                @endforeach
            </ul>
        </td>
        <td>{{$book->stock}}</td>
        <td>{{$book->price}}</td>
        <td>
            [TODO: actions]
        </td>
    </tr>
    @endforeach
</tbody>
<tfoot>
    <tr>
        <td colspan="10">
            {{$books->appends(Request::all())->links()}}
        </td>
    </tr>
</tfoot>
</table>
</div>
</div>
@endsection

```

Jika sudah kita kini bisa membuka halaman <http://larashop.test/books> dan kita akan melihat daftar buku yang sudah kita simpan & publish seperti ini:

## Books list

Cover	Title	Author	Status	Categories	Stock	Price	Action
	The first book	IBI	PUBLISH		100	20000	[TODO: actions]
	Book as a draft	Noone	DRAFT		10	29999	[TODO: actions]
	The second book	Muhammad Azamuddin	PUBLISH		400	450000	[TODO: actions]
	Book with categories	Muhammad Azamuddin	PUBLISH	<ul style="list-style-type: none"> <li>• Fiksi</li> <li>• Ilmiyah</li> </ul>	0	200000	[TODO: actions]

Sebelum berlanjut ke fitur edit book, mari kita buat tombol untuk create book di halaman daftar buku dengan kode ini:

```
<div class="row mb-3">
  <div class="col-md-12 text-right">
    <a
      href="{{route('books.create')}}"
      class="btn btn-primary"
    >Create book</a>
  </div>
</div>
```

Letakkan kode di atas tepat di atas kode `<table ...>`.

Dan kita akan memiliki tombol create book seperti ini:

Books list								Create book
Cover	Title	Author	Status	Categories	Stock	Price	Action	
	The first book	IBI	DRAFT	<ul style="list-style-type: none"> <li>• Ilmiyah</li> </ul>	100	20000	<button>Edit</button>	<button>Trash</button>
	Book with categories	Muhammad Azamuddin	PUBLISH	<ul style="list-style-type: none"> <li>• Fiksi</li> <li>• Ilmiyah</li> </ul>	0	200000	<button>Edit</button>	<button>Trash</button>

## Fitur edit book

Mari kita tambahkan tombol edit di daftar buku, buka kembali file view

`resources/views/books/index.blade.php`, dan tambahkan kode berikut ini:

```
<a  
    href="{{route('books.edit', ['id' => $book->id])}}"  
    class="btn btn-info btn-sm"  
> Edit </a>
```

Kode tersebut mereplace kode ini:

```
[TODO: actions]
```

Selanjutnya kita buka `app/BookController.php` dan kita sesuaikan action `edit` supaya menjadi seperti ini:

```
public function edit($id)  
{  
    $book = \App\Book::findOrFail($id);  
  
    return view('books.edit', ['book' => $book]);  
}
```

Kemudian kita buat view untuk edit buku tersebut di `resources/views/books/edit.blade.php` dan kita isikan kerangka kode form untuk edit buku seperti ini:

```
@extends('layouts.global')  
  
@section('title') Edit book @endsection  
  
@section('content')  
  
<div class="row">  
    <div class="col-md-8">  
  
        <form  
            enctype="multipart/form-data"  
            method="POST"  
            action="{{route('books.update', ['id' => $book->id])}}"  
            class="p-3 shadow-sm bg-white"  
        >  
  
            @csrf  
            <input type="hidden" name="_method" value="PUT">  
  
        </form>  
    </div>
```

&lt;/div&gt;

@endsection

Kode di atas merupakan kode form kosong yang akan mengirimkan data dengan method PUT ke route <http://larashop.test/books/{id}>. Setelah itu mari kita tambahkan input-input untuk edit sehingga view books.edit kini akan terlihat seperti ini:

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

@section('title') Edit book @endsection

@section('content')

<div class="row">
    <div class="col-md-8">

        @if(session('status'))
            <div class="alert alert-success">
                {{session('status')}}
            </div>
        @endif

        <form
            enctype="multipart/form-data"
            method="POST"
            action="{{route('books.update', ['id' => $book->id])}}"
            class="p-3 shadow-sm bg-white"
        >

            @csrf
            <input type="hidden" name="_method" value="PUT">

            <label for="title">Title</label><br>
            <input
                type="text"
                class="form-control"
                value="{{ $book->title }}"
                name="title"
                placeholder="Book title"
            />
            <br>

            <label for="cover">Cover</label><br>
            <small class="text-muted">Current cover</small><br>
            @if($book->cover)
                
            @endif
            <br><br>
            <input
                type="file"
```

```

    class="form-control"
    name="cover"
  >
  <small class="text-muted">Kosongkan jika tidak ingin mengubah
cover</small>
<br><br>

<label for="slug">Slug</label><br>
<input
  type="text"
  class="form-control"
  value="{{ $book->slug }}"
  name="slug"
  placeholder="enter-a-slug"
/>
<br>

<label for="description">Description</label> <br>
<textarea name="description" id="description" class="form-control">
{{ $book->description }}</textarea>
<br>

<label for="categories">Categories</label>
<select multiple class="form-control" name="categories"
id="categories"></select>
<br>
<br>

<label for="stock">Stock</label><br>
<input type="text" class="form-control" placeholder="Stock" id="stock"
name="stock" value="{{ $book->stock }}">
<br>

<label for="author">Author</label>
<input placeholder="Author" value="{{ $book->author }}" type="text"
id="author" name="author" class="form-control">
<br>

<label for="publisher">Publisher</label><br>
<input class="form-control" type="text" placeholder="Publisher"
name="publisher" id="publisher" value="{{ $book->publisher }}">
<br>

<label for="price">Price</label><br>
<input type="text" class="form-control" name="price"
placeholder="Price" id="price" value="{{ $book->price }}">
<br>

<label for="">Status</label>
<select name="status" id="status" class="form-control">
  <option {{ $book->status == 'PUBLISH' ? 'selected' : '' }}>
value="PUBLISH">PUBLISH</option>
  <option {{ $book->status == 'DRAFT' ? 'selected' : '' }}>

```

```

value="DRAFT">DRAFT</option>
</select>
<br>

<button class="btn btn-primary" value="PUBLISH">Update</button>

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

@endsection

@section('footer-scripts')
<link href="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/css/select2.min.css" rel="stylesheet" />

<script src="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/js/select2.min.js"></script>

<script>
$('#categories').select2({
  ajax: {
    url: 'http://larashop.test/ajax/categories/search',
    processResults: function(data){
      return {
        results: data.map(function(item){return {id: item.id, text:
item.name} })
      }
    }
  }
});

var categories = {!! $book->categories !!}

categories.forEach(function(category){
  var option = new Option(category.name, category.id, true, true);
  $('#categories').append(option).trigger('change');
});
</script>
@endsection

```

Selain input-input text perlu diperhatikan bahwa kita juga membuat input select dengan plugin **select2** sama seperti di form create book, yaitu untuk memilih **Category**. Dan perbedaan selanjutnya karena kita akan menampilkan kategori-kategori dari buku yang diedit (jika sebelumnya sudah terdapat kategori), perhatikan kita telah menambahkan kode berikut ini:

```

var categories = {!! $book->categories !!}

categories.forEach(function(category){
  var option = new Option(category.name, category.id, true, true);

```

```
$('#categories').append(option).trigger('change');
});
```

di section 'footer-scripts', kode di atas berfungsi untuk menampilkan kategori yang sudah ada sebelumnya ke input select categories.

Kemudian kita perlu mengupdate `BookController@update` seperti ini:

```
public function update(Request $request, $id)
{
    $book = \App\Book::findOrFail($id);

    $book->title = $request->get('title');
    $book->slug = $request->get('slug');
    $book->description = $request->get('description');
    $book->author = $request->get('author');
    $book->publisher = $request->get('publisher');
    $book->stock = $request->get('stock');
    $book->price = $request->get('price');

    $new_cover = $request->file('cover');

    if($new_cover){
        if($book->cover && file_exists(storage_path('app/public/' . $book->cover))){
            \Storage::delete('public/' . $book->cover);
        }

        $new_cover_path = $new_cover->store('book-covers', 'public');

        $book->cover = $new_cover_path;
    }

    $book->updated_by = \Auth::user()->id;

    $book->status = $request->get('status');

    $book->save();

    $book->categories()->sync($request->get('categories'));

    return redirect()->route('books.edit', ['id'=>$book->id])->with('status', 'Book successfully updated');
}
```

Dan kita juga perlu menambahkan kode ini di view `books.edit`

```
@if(session('status'))
<div class="alert alert-success">
```

```
 {{session('status')}}
```

```
</div>
```

```
@endif
```

## Fitur soft delete book

### Aktifkan soft delete pada model Book

Kita akan menggunakan fitur `softDeletes` pada model `Book`, untuk itu kita buka model ini yang terletak di `app/Book.php`.

Pertama kita tambahkan kode berikut:

```
use Illuminate\Database\Eloquent\SoftDeletes;
```

Lalu kita tambahkan

```
use SoftDeletes;
```

Sehingga model `Book` kita akan terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Book extends Model
{
    use SoftDeletes;

    public function categories(){
        return $this->belongsToMany('App\Category');
    }
}
```

## Trash

Model kita telah siap untuk fitur soft delete (move to trash), selanjutnya kita buat tombol untuk delete buku tertentu di halaman daftar buku. Buka file `resources/views/books/index.blade.php` lalu tambahkan kode ini:

```

<form
  method="POST"
  class="d-inline"
  onsubmit="return confirm('Move book to trash?')"
  action="{{route('books.destroy', ['id' => $book->id ])}}"
>

@csrf
<input
  type="hidden"
  value="DELETE"
  name="_method">

<input
  type="submit"
  value="Trash"
  class="btn btn-danger btn-sm">

</form>

```

Letakkan kode di atas tepat dibawah kode untuk tombol edit yang telah kita buat sebelumnya.

Jika sudah maka kita akan melihat tombol trash di samping tombol edit seperti ini:

## Books list

Cover	Title	Author	Status	Categories	Stock	Price	Action
	The first book	IBI	PUBLISH		100	20000	<button>Edit</button> <button>Trash</button>
	Book as a draft I	Noone	DRAFT		10	29999	<button>Edit</button> <button>Trash</button>
	The second book	Muhammad Azamuddin	PUBLISH		400	450000	<button>Edit</button> <button>Trash</button>
	Book with categories	Muhammad Azamuddin	PUBLISH	• Fiksi • Ilmiah	0	200000	<button>Edit</button> <button>Trash</button>

Selanjutnya kita ubah `BookController@destroy`, buka file `BookController.php` dan ubah action `destroy()` sehingga menjadi seperti ini:

```

$book = \App\Book::findOrFail($id);
$book->delete();

```

```
return redirect()->route('books.index')->with('status', 'Book moved to trash');
```

Dan langkah terakhir kita buka file view `books.index` yaitu file `resources/views/books/index.blade.php` dan di atas kode `<table ...>` kita tambahkan kode berikut untuk menampilkan pesan.

```
@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif
```

Kini fitur move to trash telah selesai dan berfungsi dengan baik.

#### Show soft deleted book / show trash

Selanjutnya kita ingin memberikan fitur di halaman books list untuk bisa menampilkan buku-buku yang berada di tong sampah (soft deleted).

Pertama kita buat action `trash` di `BookController` untuk menampilkan buku-buku yang ada di tong sampah.

#### File `BookController.php`

```
public function trash(){
}
```

Setelah itu kita query buku-buku yang berada di tong sampah / soft deleted dengan method `onlyTrashed()` dan lempar ke view seperti ini:

```
public function trash(){
    $books = \App\Book::onlyTrashed()->paginate(10);

    return view('books.trash', ['books' => $books]);
}
```

Setelah itu kita buat view `books.trash` yaitu di `resources/views/books/trash.blade.php` dan kita isikan kode berikut ini:

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

@section('title') Trashed Books @endsection
```

```

@section('content')


@if(session('status'))


{{session('status')}}


@endif



| <b>Cover</b>                                                                                 | <b>Title</b>      | <b>Author</b>      | <b>Categories</b>                                                                                                                                  | <b>Stock</b>      | <b>Price</b>      | <b>Action</b>   |  |  |  |
|----------------------------------------------------------------------------------------------|-------------------|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|-----------------|--|--|--|
| @if(\$book->cover)  @endif | {{\$book->title}} | {{\$book->author}} | <ul class="pl-3" style="list-style-type: none"> @foreach(\$book-&gt;categories as \$category) <li> {{\$category-&gt;name}} </li> @endforeach </ul> | {{\$book->stock}} | {{\$book->price}} | [TODO: actions] |  |  |  |
| {{\$books->appends(Request::all()->links())}}                                                |                   |                    |                                                                                                                                                    |                   |                   |                 |  |  |  |


```

```
</tfoot>
</table>
</div>
</div>
@endsection
```

Setelah itu kita buat jalur akses ke fitur show trashed books ini dengan mengedit file route di `routes/web.php` dan tambahkn kode ini:

```
Route::get('/books/trash', 'BookController@trash')->name('books.trash');
```

Pastikan bahwa kode tersebut diletakkan di atas kode ini

```
Route::resource('books', 'BookController');
```

Jika sudah maka kita kini bisa melihat daftar buku yang ada di tong sampah di link berikut <http://larashop.test/books/trash>.

#### Show published or draft

Kita juga akan membuat dua halaman baru untuk menampilkan buku-buku yang statusnya PUBLISH saja atau statusnya DRAFT saja.

Kita akan memanfaatkan route dan action yang sudah ada yaitu `books.index`, kita akan memberikan query param / query string di url seperti ini:

- <http://larashop.test/books?status=publish>

Untuk menampilkan buku berstatus PUBLISH

- <http://larashop.test/books?status=draft>

Untuk menampilkan buku berstatus DRAFT

Maka mari kita buka kembali `BookController.php` dan ubah action `index` supaya menerima query string `status` seperti di url di atas dan query model `Book` berdasarkan dari status yang didapat. Ubah method `index` menjadi seperti ini:

```
public function index(Request $request)
{
    $status = $request->get('status');

    if($status){
        $books = \App\Book::with('categories')->where('status',
stroupper($status))->paginate(10);
    } else {
        $books = \App\Book::with('categories')->paginate(10);
    }
}
```

```

    }

    return view('books.index', ['books' => $books]);
}

```

Dengan begini maka kita bisa menampilkan buku-buku berdasarkan statusnya menggunakan url berikut:

- <http://larashop.test/books?status=publish>
- <http://larashop.test/books?status=draft>

**Menampilkan navigasi all, publish, draft dan trash**

Selanjutnya kita juga perlu untuk membuat navigasi di halaman daftar buku dan daftar buku trash. Kode untuk navigasinya adalah seperti ini:

```

<div class="row">
    <div class="col-md-6"></div>
    <div class="col-md-6">
        <ul class="nav nav-pills card-header-pills">
            <li class="nav-item">
                <a class="nav-link" href="{{route('books.index')}}">All</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{route('books.index', ['status' => 'publish'])}}">Publish</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{route('books.index', ['status' => 'draft'])}}">Draft</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{route('books.trash')}}">Trash</a>
            </li>
        </ul>
    </div>
</div>

<hr class="my-3">

```

Letakkan kode tersebut di atas kode ini:

```

<div class="row mb-3">
    <div class="col-md-12 text-right">
        <a
            href="{{route('books.create')}}"
            class="btn btn-primary"
        >Create book</a>
    </div>
</div>

```

Letakan kode di atas di dua file yaitu `resources/views/books/index.blade.php` dan `resources/views/books/trash.blade.php`

Kode di atas sudah berfungsi sebagaimana mestinya.

Kita bisa sedikit memodifikasi kode navigasi tersebut agar menampilkan link mana yang sedang aktif seperti ini:



Ubah kode navigasi tersebut agar membaca query string sehingga menjadi seperti ini:

```
<div class="row">
    <div class="col-md-6"></div>
    <div class="col-md-6">
        <ul class="nav nav-pills card-header-pills">
            <li class="nav-item">
                <a class="nav-link {{Request::get('status') == NULL &&
Request::path() == 'books' ? 'active' : ''}}>{{route('books.index')}}</a>
            </li>
            <li class="nav-item">
                <a class="nav-link {{Request::get('status') == 'publish' ? 'active' : ''}}>{{route('books.index', ['status' => 'publish'])}}</a>
            </li>
            <li class="nav-item">
                <a class="nav-link {{Request::get('status') == 'draft' ? 'active' : ''}}>{{route('books.index', ['status' => 'draft'])}}</a>
            </li>
            <li class="nav-item">
                <a class="nav-link {{Request::path() == 'books/trash' ? 'active' : ''}}>{{route('books.trash')}}</a>
            </li>
        </ul>
    </div>
</div>
```

## Restore

Kita juga akan membuat fitur restore. Yaitu untuk mengembalikan buku-buku yang berstatus soft deleted / trashed menjadi aktif.

Untuk itu mari kita buat `BookController@restore` di file `BookController.php` seperti ini:

```
public function restore($id){
    $book = \App\Book::withTrashed()->findOrFail($id);

    if($book->trashed()){
        $book->restore();
        return redirect()->route('books.trash')->with('status', 'Book
successfully restored');
    } else {
        return redirect()->route('books.trash')->with('status', 'Book is not in
trash');
    }
}
```

Lalu kita buat jalur / route untuk menggunakan action `restore` di atas menggunakan method POST. Buka file `routes/web.php` dan tambahkan kode berikut:

```
Route::post('/books/{id}/restore', 'BookController@restore')-
>name('books.restore');
```

Dan seperti biasa pastikan kode tersebut letaknya DI ATAS kode ini:

```
Route::get('books', 'BookController');
```

Setelah itu buka file view `resources/views/books/trash.blade.php` dan kita tambahkan tombol untuk merestore buku menggunakan form dengan method POST seperti ini:

```
<form
    method="POST"
    action="{{route('books.restore', ['id' => $book->id])}}"
    class="d-inline">
@csrf
    <input type="submit" value="Restore" class="btn btn-success"/>
</form>
```

Kode tersebut menggantikan kode berikut:

```
[TODO: actions]
```

**Delete permanent**

Selanjutnya kita buat fitur delete permanent, langkah-langkahnya hampir sama dengan membuat fitur restore. Pertama kita buat sebuah action yaitu action `deletePermanent` di `BookController.php` dan isikan kode berikut:

```
public function deletePermanent($id){
    $book = \App\Book::withTrashed()->findOrFail($id);

    if(!$book->trashed()){
        return redirect()->route('books.trash')->with('status', 'Book is not in trash!')->with('status_type', 'alert');
    } else {
        $book->categories()->detach();
        $book->forceDelete();

        return redirect()->route('books.trash')->with('status', 'Book permanently deleted!');
    }
}
```

Perhatian: Kenapa kita harus menambahkan kode `$book->categories()->detach();` sebelum menggunakan `forceDelete()`? karena kita ingin menghapus relationship buku yang akan dihapus dengan Category jika ada. Jika tidak kita lakukan kita akan mendapatkan error constraint violation dari mysql.

Setelah itu kita buat jalur akses / route untuk menggunakan action di atas via method DELETE. Buka file `routes/web.php` dan isikan kode ini:

```
Route::delete('/books/{id}/delete-permanent',
    'BookController@deletePermanent')->name('books.delete-permanent');
```

Letakkan kode tersebut di atas kode

```
Route::resource('books', 'BookController');
```

Lalu buka kembali file view `resources/views/books/trashed.blade.php` dan tambahkan kode berikut ini untuk menampilkan tombol delete permanent:

```
<form
    method="POST"
    action="{{route('books.delete-permanent', ['id' => $book->id])}}"
    class="d-inline"
    onsubmit="return confirm('Delete this book permanently?')"
>

@csrf
<input type="hidden" name="_method" value="DELETE">
```

```
<input type="submit" value="Delete" class="btn btn-danger btn-sm">
</form>
```

## Filter book by title

Selanjutnya kita buat filter berdasarkan nama buku, kita tidak perlu membuat action tersendiri tapi kita akan memanfaatkan action `index` di `BookController`. Kita akan membuat action tersebut bisa membaca query string "keyword" dan menjadikannya criteria saat query model `Book` jadi misal seperti ini:

`http://larashop.test/books?keyword=first`

Cari buku yang judulnya mengandung kata "first". Agar hal ini bisa dilakukan ubah `BookController@index` menjadi seperti ini:

```
public function index(Request $request)
{
    $status = $request->get('status');
    $keyword = $request->get('keyword') ? $request->get('keyword') : '';

    if($status){
        $books = \App\Book::with('categories')->where('title', 'LIKE',
"%$keyword%")->where('status', strtoupper($status))->paginate(10);
    } else {
        $books = \App\Book::with('categories')->where("title", "LIKE",
"%$keyword%")->paginate(10);
    }

    return view('books.index', ['books' => $books]);
}
```

Nah, setelah itu kita buat input untuk mengetikkan keyword pada view `books.index` dan `books.trash` kita tambahkan kode ini:

```
<form
    action="{{route('books.index')}}"
>

<div class="input-group">
    <input name="keyword" type="text" value="{{Request::get('keyword')}}">
    <div class="form-control placeholder="Filter by title">
        <div class="input-group-append">
            <input type="submit" value="Filter" class="btn btn-primary">
        </div>
    </div>
</div>

</form>
```

Sehingga kini file view `books/index.blade.php` menjadi seperti ini:

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

@section('title') Books list @endsection

@section('content')
    <div class="row">
        <div class="col-md-12">
            @if(session('status'))
                <div class="alert alert-success">
                    {{session('status')}}
                </div>
            @endif

            <div class="row">
                <div class="col-md-6">
                    <form
                        action="{{route('books.index')}}"
                    >

                        <div class="input-group">
                            <input name="keyword" type="text" value=
{{Request::get('keyword')}}" class="form-control" placeholder="Filter by
title">
                            <div class="input-group-append">
                                <input type="submit" value="Filter" class="btn btn-
primary">
                            </div>
                        </div>

                    </form>
                </div>
                <div class="col-md-6">
                    <ul class="nav nav-pills card-header-pills">
                        <li class="nav-item">
                            <a class="nav-link {{Request::get('status') == NULL &&
Request::path() == 'books' ? 'active' : ''}}" href=
{{route('books.index')}}>All</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link {{Request::get('status') == 'publish' ? 'active' : ''}}" href=
{{route('books.index', ['status' =>
'publish'])}}>Publish</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link {{Request::get('status') == 'draft' ? 'active' : ''}}" href=
{{route('books.index', ['status' =>
'draft'])}}>Draft</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link {{Request::path() == 'books/trash' ? 'active' : ''}}" href=
{{route('books.trash')}}>Trash</a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

        </li>
    </ul>
</div>
</div>

<hr class="my-3">

<div class="row mb-3">
    <div class="col-md-12 text-right">
        <a href="{{route('books.create')}}" class="btn btn-primary">Create book</a>
    </div>
</div>

<table class="table table-bordered table-striped">
    <thead>
        <tr>
            <th><b>Cover</b></th>
            <th><b>Title</b></th>
            <th><b>Author</b></th>
            <th><b>Status</b></th>
            <th><b>Categories</b></th>
            <th><b>Stock</b></th>
            <th><b>Price</b></th>
            <th><b>Action</b></th>
        </tr>
    </thead>
    <tbody>
        @foreach($books as $book)
        <tr>
            <td>
                @if($book->cover)
                    
                @endif
            </td>
            <td>{{$book->title}}</td>
            <td>{{$book->author}}</td>
            <td>
                @if($book->status == "DRAFT")
                    <span class="badge bg-dark text-white">{{$book->status}}</span>
                @else
                    <span class="badge badge-success">{{$book->status}}</span>
                @endif
            </td>
            <td>
                <ul class="pl-3">
                    @foreach($book->categories as $category)
                        <li>{{$category->name}}</li>
                    @endforeach
                </ul>
            </td>
        </tr>
    @endforeach
</tbody>
</table>

```

```

        </ul>
    </td>
    <td>{{$book->stock}}</td>
    <td>{{$book->price}}</td>
    <td>
        <a
            href="{{route('books.edit', ['id' => $book->id])}}"
            class="btn btn-info btn-sm"
        > Edit </a>

        <form
            method="POST"
            class="d-inline"
            onsubmit="return confirm('Move book to trash?')"
            action="{{route('books.destroy', ['id' => $book->id
])}}"
        >

            @csrf
            <input
                type="hidden"
                value="DELETE"
                name="_method">

            <input
                type="submit"
                value="Trash"
                class="btn btn-danger btn-sm">

        </form>

    </td>
</tr>
@endforeach
</tbody>
<tfoot>
    <tr>
        <td colspan="10">
            {{$books->appends(Request::all())->links()}}
        </td>
    </tr>
</tfoot>
</table>
</div>
</div>
@endsection

```

Dan file `books/trash.blade.php` menjadi seperti ini:

```

@extends('layouts.global')

@section('title') Trashed Books @endsection

```

```

@section('content')
<div class="row">
<div class="col-md-12">
@if(session('status'))
<div class="alert alert-success">
{{session('status')}}
</div>
@endif
<div class="row">
<div class="col-md-6">
<form
action="{{route('books.index')}}">
<div class="input-group">
<input name="keyword" type="text" value="{{Request::get('keyword')}}" class="form-control" placeholder="Filter by title">
<div class="input-group-append">
<input type="submit" value="Filter" class="btn btn-primary">
</div>
</div>
</form>
</div>
<div class="col-md-6">
<ul class="nav nav-pills card-header-pills">
<li class="nav-item">
<a class="nav-link {{Request::get('status') == NULL && Request::path() == 'books' ? 'active' : ''}}" href="{{route('books.index')}}">All</a>
</li>
<li class="nav-item">
<a class="nav-link {{Request::get('status') == 'publish' ? 'active' : ''}}" href="{{route('books.index', ['status' => 'publish'])}}>Publish</a>
</li>
<li class="nav-item">
<a class="nav-link {{Request::get('status') == 'draft' ? 'active' : ''}}" href="{{route('books.index', ['status' => 'draft'])}}>Draft</a>
</li>
<li class="nav-item">
<a class="nav-link {{Request::path() == 'books/trash' ? 'active' : ''}}" href="{{route('books.trash')}}>Trash</a>
</li>
</ul>
</div>
</div>

<hr class="my-3">

```

```

<div class="row mb-3">
    <div class="col-md-12 text-right">
        <a href="{{route('books.create')}}" class="btn btn-primary">Create book</a>
    </div>
</div>

<table class="table table-bordered table-striped">
    <thead>
        <tr>
            <th><b>Cover</b></th>
            <th><b>Title</b></th>
            <th><b>Author</b></th>
            <th><b>Categories</b></th>
            <th><b>Stock</b></th>
            <th><b>Price</b></th>
            <th><b>Action</b></th>
        </tr>
    </thead>
    <tbody>
        @foreach($books as $book)
            <tr>
                <td>
                    @if($book->cover)
                        
                    @endif
                </td>
                <td>{{$book->title}}</td>
                <td>{{$book->author}}</td>
                <td>
                    <ul class="pl-3">
                        @foreach($book->categories as $category)
                            <li>{{$category->name}}</li>
                        @endforeach
                    </ul>
                </td>
                <td>{{$book->stock}}</td>
                <td>{{$book->price}}</td>
                <td>
                    <form
                        method="POST"
                        action="{{route('books.restore', ['id' => $book->id])}}"
                        class="d-inline"
                    >
                        @csrf
                        <input type="submit" value="Restore" class="btn btn-success"/>
                    </form>
                </td>
            </tr>
        @endforeach
    </tbody>
</table>

```

```

        </form>

        <form
            method="POST"
            action="{{route('books.delete-permanent', ['id' =>
$book->id])}}"
            class="d-inline"
            onsubmit="return confirm('Delete this book
permanently?')"
        >

            @csrf
            <input type="hidden" name="_method" value="DELETE">

            <input type="submit" value="Delete" class="btn btn-danger
btn-sm">
        </form>
    </td>
</tr>
@endforeach
</tbody>
<tfoot>
<tr>
    <td colspan="10">
        {{$books->appends(Request::all())->links()}}
    </td>
</tr>
</tfoot>
</table>
</div>
</div>
@endsection

```

Dan sebelum kita melanjutkan ke CRUD Manage order, kita buat terlebih dahulu navigasi item di sidebar untuk Manage books. Buka file `resources/views/layouts/global.blade.php` lalu tambahkan kode ini:

```

<li><a href="{{route('books.index')}}"><span class="oi oi-book"></span>
Manage books</a></li>

```

Letakkan kode di atas tepat di bawah kode ini:

```

<li><a href="{{route('categories.index')}}"><span class="oi oi-tag"></span>
Manage categories</a></li>

```

Sehingga kini kita memiliki 1 menu di sidebar menuju manage books seperti ini:

## Larashop

-  Home
-  Manage users
-  Manage categories
-  Manage books

## Manage Order

Selanjutnya kita akan membuat fitur manage order. Kita overview terlebih dahulu bahwa model `Order` memiliki relationship `one-to-one` dengan model `User` dan memiliki `many-to-many` relationship dengan model `Book`.

Apa maksud dari relationship di atas? maksudnya adalah bahwa setiap `Order` selalu memiliki pemilik / pembeli dari model `User`. Dan di dalam `Order` terdapat satu atau lebih `Book`.

Untuk itu nanti kita akan kembali belajar mendefinisikan relationship.

Membuat migration

### **Membuat migration untuk tabel orders**

Mari kita buat tabel `books` menggunakan migration, struktur tabel `books` sesuai desain database kita adalah seperti ini:

orders
+ id: Integer {PK}
+ user_id: Integer
+ total_price: Float
+ invoice_number: String
+ status: Enum
+ created_at: Integer
+ updated_at: Integer

Perlu kita ingat kembali bahwa untuk field `status` yang bertipe `enum` akan memiliki opsi-opsi ini:

- SUBMIT
- PROCESS
- FINISH
- CANCEL

Buat sebuah file migration dengan perintah ini:

```
php artisan make:migration create_orders_table
```

Setelah itu buka file yang baru saja tergenerate di direktori `database/migration` nama file tersebut berakhiran `create_orders_table.php`. Isi dengan kode migration ini:

```
<?php

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

class CreateOrdersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned();
            $table->float('total_price')->unsigned()->defaults(0);
            $table->string('invoice_number');
            $table->enum('status', ['SUBMIT', 'PROCESS', 'FINISH',
'CANCEL']);
            $table->timestamps();
        });
    }

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

```
        $table->foreign('user_id')->references('id')->on('users');
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('orders', function(Blueprint $table){
        $table->dropForeign('orders_user_id_foreign');
    });

    Schema::dropIfExists('orders');
}
}
```

Perhatikan kode migration di atas, selain kita mendefinisikan struktur tabel `orders` kita juga sekaligus mendefinisikan relationship antara table `orders` ke tabel `users` melalui field `user_id`.

Kemudian pada method `down()` untuk migration rollback, sebelum kita melakukan `dropIfExists('orders')` kita menghapus terlebih dahulu index / definisi relationship di field `user_id` menggunakan `$table->dropForeign('orders_user_id_foreign')`.

Setelah itu jalankan perintah berikut:

```
php artisan migrate
```

### Membuat migration untuk relationship ke tabel books

Seperti kita bahas di awal bahwa selain relationship ke table `users`, table `orders` juga memiliki relationship ke table `books`. Relation ini merupakan `many-to-many` relationship sehingga kita memerlukan sebuah pivot table dengan nama `book_order` oleh karenanya mari kita buat migration untuk membuat table `book_order` dan definisikan relationship table `books` dan `orders` di file tersebut.

Jalankan perintah ini:

```
php artisan make:migration create_book_order_table
```

Setelah itu kita lihat lagi seperti apa struktur table `book_order` ini sesuai desain database kita, yaitu seperti ini:

book_order
+ id: Integer(PK)
+ order_id: Integer
+ book_id: Integer
+ quantity: Integer
+ created_at: timestamps
+ updated_at: timestamps

Maka pada file migration tersebut kita isikan kode ini:

```
<?php

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

class CreateBookOrderTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('book_order', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('order_id')->unsigned();
            $table->integer('book_id')->unsigned();
            $table->integer('quantity')->unsigned()->defaults(1);
            $table->timestamps();

            $table->foreign('order_id')->references('id')->on('orders');
            $table->foreign('book_id')->references('id')->on('books');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('book_order', function(Blueprint $table){
            $table->dropForeign(['order_id']);
            $table->dropForeign(['book_id']);
        });
    }
}
```

```
Schema::dropIfExists('book_order');
}
}
```

File migration di atas selain membuat struktur table **order** juga mendefinisikan relationship antara tabel **books** dengan **orders** melalui field **book\_id** dan **order\_id**.

Selanjutnya jalankan perintah ini:

```
php artisan migrate
```

Dengan begini sekarang kita memiliki tabel **book\_order** yang kita gunakan sebagai pivot tabel untuk relationship **many-to-many** antara tabel **books** dan **orders**. Kini kita juga siap untuk mendefinisikan relationship tersebut di model **Book** dan model **Order**.

## Membuat model Order

Untuk membuat fitur CRUD Order mari kita buat terlebih dahulu model **Order** dengan perintah ini:

```
php artisan make:model Order
```

## Mendefinisikan relationship

Setelah kita memiliki model **Order** dan kita sudah mendefinisikan relationship di tabel **orders** ke tabel **users** dan **book** maka kini saatnya kita definisikan relationship tersebut di model.

### Relationship di model Order

Pertama adalah model **User**, kita akan membuat 2 relationship di sini, yaitu **user** merefensikan model **User** dan **books** merepresentasikan banyak model **Book** yang masuk dalam sebuah `Order`.

Untuk membuat relationship dengan model **User** buka file **app/Order.php** lalu tambahkan kode berikut ini:

```
public function user(){
    return $this->belongsTo('App\User');
}
```

Kemudian pada model **User** yaitu file **app/User.php** kita tambahkan relationship **orders** seperti ini:

```
public function orders(){
    return $this->hasMany('App\Order');
}
```

Dengan begitu relationship **one-to-many** antara model **User** dan model **Book** telah terbentuk.

Selanjutnya kita definisikan relationship **many-to-many** relationship antara model **Book** dengan model **Order**.

Pertama kita definisikan relation dengan nama **book** pada model **Order**, buka kembali file **app/Order.php** lalu tambahkan kode ini:

```
public function books(){
    return $this->belongsToMany('App\Book')->withPivot('quantity');;
}
```

Pada definisi relationship di atas kita juga ingin mengambil field yang berada di tabel pivot yaitu field **quantity**, oleh karenanya kita juga menggunakan method **withPivot('quantity')**. Sehingga nanti di view kita bisa mendapatkan data **quantity** menggunakan kode seperti ini:

```
@foreach($order->books as $book)
    Quantity: {{$book->pivot->quantity}}
@endforeach
```

Kemudian kita buka model **Book** yaitu file **app/Book.php** dan kita definisikan relation dengan nama **orders** seperti ini:

```
public function orders(){
    return $this->belongsToMany('App\Order');
}
```

Kini model **Order** kita akan terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    public function user(){
        return $this->belongsTo('App\User');
    }

    public function books(){
        return $this->belongsToMany('App\Book')->withPivot('quantity');;
    }
}
```

Dan model **Book** kita kini terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Book extends Model
{
    use SoftDeletes;

    public function categories(){
        return $this->belongsToMany('App\Category');
    }

    public function orders(){
        return $this->belongsToMany('App\Order');
    }
}
```

Sementara itu model **User** kita akan terlihat seperti ini:

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password'
    ];

    /**

```

```
* The attributes that should be hidden for arrays.  
*  
* @var array  
*/  
protected $hidden = [  
    'password', 'remember_token',  
];  
  
public function orders(){  
    return $this->hasMany('App\Order');  
}  
}
```

Nah kini model kita sudah saling terhubung, dan kita siap untuk membuat fitur CRUD Order.

## CRUD Order

Kita buat controller resource terlebih dahulu untuk fitur CRUD Order, dengan perintah ini :

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

Setelah itu kita buat auth resource untuk **OrderController**. Buka file **routes/web.php** dan tambahkan kode ini:

```
Route::resource('orders', 'OrderController');
```

## Persiapan database

Untuk membuat fitur manage orders kita memerlukan data dummy di database. Karena pada manage orders kita tidak akan membuat fitur create order, kenapa? Fitur tersebut akan dibuat di front store yaitu pada study kasus menggunakan Vue.

Untuk itu kami telah menyediakan file .sql untuk diimport, file tersebut ada di folder **files studi kasus** bernama **larashop.sql**. Buka phpmyadmin, untuk xampp ada di <http://localhost/phpmyadmin> sementara untuk laradock ada di <http://localhost:8080>. Setelah itu pilih database yang dipakai oleh aplikasi Laravel kita, jika kamu tidak mengikuti seperti di bab ini, maka nama database tersebut adalah **larashop**

Pilih database tersebut dari daftar database di sidebar kiri, lalu klik menu **Operations** dan pilih **Drop database**. Yup hapus terlebih dahulu tabel **larashop**. Kamu juga bisa melakukan backup terlebih dahulu dengan menu **Export** sebelum menghapus baru setelah itu drop / hapus. Jika sudah terhapus maka kita buat lagi database baru dengan nama yang sama yaitu **larashop** kemudian pilih kembali database tersebut, lalu klik import dan pilih file **larashop.sql**.

The screenshot shows the phpMyAdmin interface with the 'larashop' database selected. In the left sidebar, there's a tree view of tables: books, book\_category, book\_order, categories, migrations, orders, password\_resets, and users. The 'File to import' section is highlighted, containing fields for choosing a file (with a note about max size 512MB), character set (utf-8), and other import settings like skipping queries and enabling foreign key checks.

Setelah itu database kita akan memiliki data dummy agar kita bisa menampilkan daftar order.

Alasan kita menghapus tabel `larashop` terlebih dahulu adalah jika tidak maka kita akan gagal saat import. Hal tersebut disebabkan database tersebut sudah memiliki tabel-tabel seperti yang akan diimport.

## Fitur list orders

Nah kini siap untuk membuat halaman daftar orders, pertama tentu kita buka `app/Http/Controllers/OrderController.php`. Lalu kita ubah action `index` agar menjadi seperti ini:

```
public function index(){
    $orders = \App\Order::with('user')->with('books')->paginate(10);

    return view('orders.index', ['orders' => $orders]);
}
```

Penjelasan kode: kita melakukan query ke tabel `orders` menggunakan model `\App\Order` selain data dari tabel tersebut kita juga sekaligus mendapatkan data `User` dari order yang bersangutan. Juga data `Book` yang masuk dalam `Order` tersebut menggunakan `with('user')` dan `with('books')`. Method `with()` merupakan cara untuk mendapatkan data relationship berdasarkan nama relation di model tertentu.

Lalu kita buat file view di `resources/views/orders/index.blade.php` dan kita tampilkan berikan kode untuk menampilkan daftar orders seperti ini:

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

@section('title') Orders list @endsection

@section('content')
    <div class="row">
```

```

<div class="col-md-12">
    <table class="table table-striped table-bordered">
        <thead>
            <tr>
                <th>Invoice number</th>
                <th><b>Status</b></th>
                <th><b>Buyer</b></th>
                <th><b>Total quantity</b></th>
                <th><b>Order date</b></th>
                <th><b>Total price</b></th>
                <th><b>Actions</b></th>
            </tr>
        </thead>
        <tbody>
            @foreach($orders as $order)
            <tr>
                <td>{{$order->invoice_number}}</td>
                <td>
                    @if($order->status == "SUBMIT")
                        <span class="badge bg-warning text-light">{{$order->status}}</span>
                    @elseif($order->status == "PROCESS")
                        <span class="badge bg-info text-light">{{$order->status}}</span>
                    @elseif($order->status == "FINISH")
                        <span class="badge bg-success text-light">{{$order->status}}</span>
                    @elseif($order->status == "CANCEL")
                        <span class="badge bg-dark text-light">{{$order->status}}</span>
                </td>
                @endif
                <td>
                    {{$order->user->name}} <br>
                    <small>{{$order->user->email}}</small>
                </td>
                <td>{{$order->totalQuantity}} pc (s)</td>
                <td>{{$order->created_at}}</td>
                <td>{{$order->total_price}}</td>
                <td>
                    [TODO: actions]
                </td>
            </tr>
        @endforeach
    </tbody>
    <tfoot>
        <tr>
            <td colspan="10">
                {{$orders->append(Request::all()->links())}}
            </td>
        </tr>
    </tfoot>
</table>
</div>

```

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

Bila kamu perhatikan kita menampilkan total quantity yaitu penjumlahan dari seluruh data `quantity` di tabel pivot. Kita menampilkannya dengan kode  `{{$order->totalQuantity}}`  padahal di tabel `orders` tidak ada field `totalQuantity` lalu bagaimana agar hal itu bisa terjadi?

Field semacam itu disebut dengan dynamic property, kita harus mendefinisikannya terlebih dahulu di model yang akan menggunakannya. Misalnya tadi `$order->totalQuantity` itu berarti kita buat dynamic property tersebut di model `Order`. Oleh karenanya buka file `app/Order.php` dan tambahkan dynamic property seperti ini:

```
public function getTotalQuantityAttribute(){
    $total_quantity = 0;

    foreach($this->books as $book){
        $total_quantity += $book->pivot->quantity;
    }

    return $total_quantity;
}
```

Kode di atas merupakan definisi dynamic property di model `Order` penamaannya adalah menggunakan format `get` dan diakhiri `Attribute`. Seperti contoh di atas nama dari functionnya adalah `getTotalQuantityAttribute` sehingga kita bisa mengakses dengan cara `$order->totalQuantity` tanpa `get` maupun `Attribute`. Bisa dipahami kan? Dynamic property tersebut kalo dilihat di kodennya akan menjumlahkan quantity yang diambil dari tabel pivot, hasil penjumlahan tersebut yang akan menjadi nilai dari dynamic property.

Model `Order` kita kini terlihar seperti ini:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    public function user(){
        return $this->belongsTo('App\User');
    }

    public function books(){
        return $this->belongsToMany('App\Book')->withPivot('quantity');
    }
}
```

```
public function getTotalQuantityAttribute(){
    $total_quantity = 0;

    foreach($this->books as $book){
        $total_quantity += $book->pivot->quantity;
    }

    return $total_quantity;
}
```

## Fitur edit status order

Kita akan membuat form untuk edit order, akan tetapi perlu diperhatikan kita hanya mengizinkan perubahan untuk field **status** sementara field lainnya tidak kita izinkan untuk diubah.

Kita buka `app/Http/Controllers/OrderController.php` dan pada action `edit` kita jadikan seperti ini:

```
public function edit($id){
    $order = \App\Order::findOrFail($id);

    return view('orders.edit', ['order' => $order]);
}
```

Setelah itu kita buat file view baru di `resources/views/orders/edit.blade.php` lalu kita isikan form untuk edit order seperti ini:

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

@section('title') Edit order @endsection

@section('content')



@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif

<form
    class="shadow-sm bg-white p-3"
    action="{{route('orders.update', ['id' => $order->id])}}"
    method="POST"
>

    @csrf


```

```

<input type="hidden" name="_method" value="PUT">

<label for="invoice_number">Invoice number</label><br>
<input type="text" class="form-control" value="{{ $order->invoice_number }}" disabled>
<br>

<label for="">Buyer</label><br>
<input disabled class="form-control" type="text" value="{{ $order->user->name }}">
<br>

<label for="created_at">Order date</label><br>
<input type="text" class="form-control" value="{{ $order->created_at }}" disabled >
<br>

<label for="">Books ({{ $order->totalQuantity }}) </label><br>
<ul>
@foreach($order->books as $book)
    <li>{{ $book->title }} <b>({{$book->pivot->quantity}})</b></li>
@endforeach
</ul>

<label for="">Total price</label><br>
<input class="form-control" type="text" value="{{ $order->total_price }}" disabled>
<br>

<label for="status">Status</label><br>
<select class="form-control" name="status" id="status">
    <option {{ $order->status == "SUBMIT" ? "selected" : "" }} value="SUBMIT">SUBMIT</option>
    <option {{ $order->status == "PROCESS" ? "selected" : "" }} value="PROCESS">PROCESS</option>
    <option {{ $order->status == "FINISH" ? "selected" : "" }} value="FINISH">FINISH</option>
    <option {{ $order->status == "CANCEL" ? "selected" : "" }} value="CANCEL">CANCEL</option>
</select>
<br>

<input type="submit" class="btn btn-primary" value="Update">

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

@endsection

```

Setelah itu kita buat tombol edit dari halaman daftar order. Buka file

resources/views/orders/index.blade.php dan tambahkan kode berikut ini:

```
<a href="{{route('orders.edit', ['id' => $order->id])}}" class="btn btn-info btn-sm"> Edit</a>
```

Kode tersebut menggantikan kode berikut ini:

```
[TODO: actions]
```

Setelah itu kita buka kembali OrderController dan kita ubah action update() agar menerima request update seperti ini:

```
public function update(Request $request, $id){
    $order = \App\Order::findOrFail($id);

    $order->status = $request->get('status');

    $order->save();

    return redirect()->route('orders.edit', ['id' => $order->id])->with('status', 'Order status successfully updated');
}
```

## Fitur pencarian order

Kita akan membuat fitur untuk mencari order berdasarkan email pembeli (buyer email) dan berdasarkan status order. Pertama kita buat tampilan filter di halaman daftar order. Buka file view

resources/views/orders/index.blade.php lalu kita tambahkan kode ini di section('content')

```
<form action="{{route('orders.index')}}">
    <div class="row">
        <div class="col-md-5">
            <input value="{{Request::get('buyer_email')}}" name="buyer_email" type="text" class="form-control" placeholder="Search by buyer email">
        </div>
        <div class="col-md-2">
            <select name="status" class="form-control" id="status">
                <option value="">ANY</option>
                <option {{Request::get('status') == "SUBMIT" ? "selected" : ""}} value="SUBMIT">SUBMIT</option>
                <option {{Request::get('status') == "PROCESS" ? "selected" : ""}} value="PROCESS">PROCESS</option>
                <option {{Request::get('status') == "FINISH" ? "selected" : ""}} value="FINISH">FINISH</option>
            </select>
        </div>
    </div>
</form>
```

```

        <option {{Request::get('status') == "CANCEL" ? "selected" : ""}}>
    value="CANCEL">CANCEL</option>
    </select>
</div>
<div class="col-md-2">
    <input type="submit" value="Filter" class="btn btn-primary">
</div>
</div>
</form>

<hr class="my-3">

```

Sehingga view `orders.index` kini terlihat seperti ini:

```

@extends('layouts.global')

@section('title') Orders list @endsection

@section('content')

<form action="{{route('orders.index')}}">
    <div class="row">
        <div class="col-md-5">
            <input value="{{Request::get('buyer_email')}}" name="buyer_email"
type="text" class="form-control" placeholder="Search by buyer email">
        </div>
        <div class="col-md-2">
            <select name="status" class="form-control" id="status">
                <option value="">ANY</option>
                <option {{Request::get('status') == "SUBMIT" ? "selected" : ""}}>
    value="SUBMIT">SUBMIT</option>
                <option {{Request::get('status') == "PROCESS" ? "selected" : ""}}>
    value="PROCESS">PROCESS</option>
                <option {{Request::get('status') == "FINISH" ? "selected" : ""}}>
    value="FINISH">FINISH</option>
                <option {{Request::get('status') == "CANCEL" ? "selected" : ""}}>
    value="CANCEL">CANCEL</option>
            </select>
        </div>
        <div class="col-md-2">
            <input type="submit" value="Filter" class="btn btn-primary">
        </div>
    </div>
</form>

<hr class="my-3">

<div class="row">
    <div class="col-md-12">
        <table class="table table-striped table-bordered">
            <thead>
                <tr>
```

```

<th>Invoice number</th>
<th><b>Status</b></th>
<th><b>Buyer</b></th>
<th><b>Total quantity</b></th>
<th><b>Order date</b></th>
<th><b>Total price</b></th>
<th><b>Actions</b></th>
</tr>
</thead>
<tbody>
@foreach($orders as $order)
<tr>
<td>{{$order->invoice_number}}</td>
<td>
@if($order->status == "SUBMIT")
<span class="badge bg-warning text-light">{{$order->status}}</span>
@elseif($order->status == "PROCESS")
<span class="badge bg-info text-light">{{$order->status}}</span>
@elseif($order->status == "FINISH")
<span class="badge bg-success text-light">{{$order->status}}</span>
@elseif($order->status == "CANCEL")
<span class="badge bg-dark text-light">{{$order->status}}</span>
</span>
@endif
</td>
<td>
{{$order->user->name}} <br>
<small>{{$order->user->email}}</small>
</td>
<td>{{$order->totalQuantity}} pc (s)</td>
<td>{{$order->created_at}}</td>
<td>{{$order->total_price}}</td>
<td>
 Edit
</td>
</tr>
@endforeach
</tbody>
<tfoot>
<tr>
<td colspan="10">
{{$orders->appends(Request::all()->links())}}
</td>
</tr>
</tfoot>
</table>
</div>
</div>
@endsection

```

Jika sudah maka kamu akan mendapat tampilan seperti ini:

Orders list						
Search by buyer email			ANY	Filter		
Invoice number	Status	Buyer	Total quantity	Order date	Total price	Actions
201807060001	FINISH	Nadia Nurul Mila nadia@gmail.com	1 pc (s)	2018-07-06 00:00:00	390000	<button>Edit</button>
201807250002	CANCEL	Habib Asagaf habib@gmail.com	2 pc (s)	2018-07-26 00:00:00	780000	<button>Edit</button>

Nah sekarang mari kita ubah action `index` di `OrderController` agar dapat menerima request filter berdasarkan status dan buyer email. Buka file `app/Http/Controllers/OrderController.php` dan ubah action `index` menjadi seperti ini:

```
public function index(Request $request)
{
    $status = $request->get('status');
    $buyer_email = $request->get('buyer_email');

    $orders = \App\Order::with('user')
        ->with('books')
        ->whereHas('user', function($query) use ($buyer_email) {
            $query->where('email', 'LIKE', "%$buyer_email%");
        })
        ->where('status', 'LIKE', "%$status%")
        ->paginate(10);

    return view('orders.index', ['orders' => $orders]);
}
```

Kode di atas menangkap request filter dari form yang kita tambahkan di halaman daftar order. Satu hal yang belum pernah kita gunakan di studi kasus ini adalah method `whereHas`. Sebelumnya kita mempelajarinya di bab relationship, dan kini kita benar-benar mempraktikannya.

Penggunaan method `whereHas` ini untuk mencari order berdasarkan dari email pembeli seperti ini:

```
->whereHas('user', function($query) use ($buyer_email){
    $query->where('email', 'like', "%$buyer_email%");
})
```

Dengan begini maka kita bisa melakukan filtering di halaman daftar order berdasarkan status dan email pembeli.

Langkah terakhir mari kita tambahkan menu "Manage orders" di menu sidebar kiri. Buka file layout `resources/views/layouts/global.blade.php` lalu tambahkan kode ini:

```
<li><a href="{{route('orders.index')}}"><span class="oi oi-inbox"></span>
Manage orders</a></li>
```

Letakkan kode itu setelah kode:

```
<li><a href="{{route('books.index')}}"><span class="oi oi-book"></span>
Manage books</a></li>
```

## Validasi Form

Fitur management resource kita telah selesai, akan tetapi dalam setiap form yang sudah kita buat belum ada validasi. Sehingga misalnya kamu tanpa mengisi data langsung submit form create category akan error.

The screenshot shows a browser's developer tools with a stack trace and the source code for `Illuminate\Database\Connection.php`. The error message is a SQLSTATE [23000] integrity constraint violation: 1048, indicating that the 'name' column cannot be null. The stack trace shows the error occurred in the `runQueryCallback` method of the `Illuminate\Database\Connection` class, which is part of the Laravel framework. The source code for this method is shown, along with the arguments and environment details.

Begitu juga dengan create users atau yang lain. Dan di sub bab ini kita akan membuat validasi dan mengirimkan pesan error berdasarkan kesalahan yang dibuat oleh user seperti ini:

## Create New User

Name

Mu

The name must be at least 5 characters.

Username

aza

The username must be at least 5 characters.

Roles

Administrator  Staff  Customer

The roles field is required.

Phone number

084

The phone must be between 10 and 12 digits.

Address

Test

### Validasi form create user

Untuk membuat validasi di form create user seperti gambar sebelumnya ada dua file yang harus kita edit.

Yaitu file `app/Http/Controllers/UserController.php` dan file view

`resources/views/users/create.blade.php`. Pada dasarnya kita lakukan validasi di controller action kemudian kita tampilkan pesan error di masing-masing input form. Selain itu juga kita berikan nilai dari yang diinput user agar user tidak perlu mengetik dari awal.

Buka file `app/Http/Controllers/UserController.php` dan tambahkan kode ini di bagian paling atas action `store`:

```
\Validator::make($request->all(), [  
    "name" => "required|min:5|max:100",  
    "username" => "required|min:5|max:20",  
    "roles" => "required",  
    "phone" => "required|digits_between:10,12",  
    "address" => "required|min:20|max:200",
```

```
"avatar" => "required",
"email" => "required|email",
"password" => "required",
"password_confirmation" => "required|same:password"
])->validate();
```

Kita membuat validation menggunakan facade `\Validator` dengan method `make()`. Method tersebut menerima 2 parameter, parameter pertama kita isi dengan semua request dari form `$request->all()` bertipe array, parameter kedua juga bertipe array, yaitu rules dari masing-masing field yang divalidasi.

Ada beberapa validation rule yang telah kita gunakan antara lain:

- `required`

Menandakan bahwa field harus diisi

- `min:x`

Menandakan bahwa panjang minimal dari nilai field adalah x character

- `max:x`

Menandakan bahwa panjang maximal dari nilai field adalah x character

- `unique:table_name`

Menandakan bahwa field harus unique berdasarkan data di field yang sama pada table `table_name`

Misalnya

```
"email" => unique:users
```

Artinya data `email` harus unique berdasarkan data yang ada di table `users` field `email`

Contoh lain:

```
"email" => unique:customers
```

Artinya data `email` harus unique berdasarkan data yang ada di tabel `customers` field `email`

- `digits_between:x,y`

Menandakan bahwa nilai dari field harus berupa digits dan panjangnya antara x dan y

- `email`

Menandakan bahwa nilai dari field harus berupa email. Kita gunakan untuk validasi email

- `same:field_lain`

Menandakan bahwa nilai dari sebuah field harus sama dengan nilai dari `field_lain`

Setelah itu kita panggil method `validate()` agar Laravel melakukan validasi terhadap `$request->all()` dan otomatis redirect ke halaman form jika terjadi kegagal validasi beserta dengan pesan error yang bisa diakses di view.

Kode action `UserController@store` kita menjadi seperti ini:

```
public function store(Request $request)
{
    $validation = \Validator::make($request->all(), [
        "name" => "required|min:5|max:100",
        "username" => "required|min:5|max:20|unique:users",
        "roles" => "required",
        "phone" => "required|digits_between:10,12",
        "address" => "required|min:20|max:200",
        "avatar" => "required",
        "email" => "required|email|unique:users",
        "password" => "required",
        "password_confirmation" => "required|same:password"
    ])->validate();

    $new_user = new \App\User;
    $new_user->name = $request->get('name');
    $new_user->username = $request->get('username');
    $new_user->roles = json_encode($request->get('roles'));
    $new_user->address = $request->get('address');
    $new_user->phone = $request->get('phone');
    $new_user->email = $request->get('email');
    $new_user->password = \Hash::make($request->get('password'));

    if($request->file('avatar')){
        $file = $request->file('avatar')->store('avatars', 'public');
        $new_user->avatar = $file;
    }

    $new_user->save();

    return redirect()->route('users.create')->with('status', 'User successfully created');
}
```

## Menampilkan pesan error di form

Setelah kita melakukan validasi di `UserController@store`, selanjutnya kita akan menampilkan error di view. Buka file view `resources/views/users/create.blade.php` dan kita akan mengubah setiap inputan yang ada, misalnya untuk input field `name` yang tadinya seperti ini:

```
<label for="name">Name</label>
<input class="form-control" placeholder="Full Name" type="text" name="name"
id="name"/>
<br>
```

Pertama, kita akan menampilkan data yang diketik oleh user sebelumnya menggunakan method `old('name')` seperti ini:

```
<input value="{{old('name')}}" ...dst >
```

Selanjutnya kita berikan class `is-invalid` jika terdapat error untuk field name. Untuk mengecek apakah terdapat error kita bisa gunakan `$errors->first('name')`. Sehingga input field kita tambahkan kode berikut:

```
<input class="form-control {{$errors->first('name') ? "is-invalid" : ""}} ...
...dst/>
```

Pemberian class "is-invalid" adalah agar input tersebut memiliki border berwarna merah. Ini merupakan styling / class dari template bootstrap yang kita pakai, yaitu polished template.

Selanjutnya kita tampilkan pesan error dari field `name` seperti ini:

```
<div class="invalid-feedback">
  {{$errors->first('name')}}
</div>
```

class "invalid-feedback" juga merupakan styling dari template polished. Kita tidak perlu mengecek dengan `@if` karena class ini defaultnya tidak akan muncul / display none, kecuali jika input di atasnya memiliki class "is-invalid". Apabila kamu tidak menggunakan template polished kamu mungkin perlu mengecek terlebih dahulu dengan `@if($errors->first('name'))`.

Hasil akhir dari field name kini menjadi seperti ini:

```
<label for="name">Name</label>
<input value="{{old('name')}}" class="form-control {{$errors->first('name') ? "is-invalid" : ""}}" placeholder="Full Name" type="text" name="name"
id="name"/>
<div class="invalid-feedback">
  {{$errors->first('name')}}
</div>
<br>
```

Dan kita lakukan juga untuk setiap field di form create user. Sehingga hasil akhir dari file view `resources/views/users/create.blade.php` menjadi seperti ini:

```
@extends("layouts.global")

@section("title") Create New User @endsection

@section("content")

<div class="col-md-8">

@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif

<form enctype="multipart/form-data" class="bg-white shadow-sm p-3"
action="{{route('users.store')}}" method="POST">
    @csrf
    <label for="name">Name</label>
    <input value="{{old('name')}}" class="form-control {{$errors->first('name') ? "is-invalid" : ""}}" placeholder="Full Name" type="text"
name="name" id="name"/>
    <div class="invalid-feedback">
        {{$errors->first('name')}}
    </div>
    <br>

    <label for="username">Username</label>
    <input value="{{old('username')}}" class="form-control {{$errors->first('username') ? "is-invalid" : ""}}" placeholder="username"
type="text" name="username" id="username"/>
    <div class="invalid-feedback">
        {{$errors->first('username')}}
    </div>
    <br>

    <label for="">Roles</label>
    <br>
    <input
        class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
    >
        type="checkbox"
        name="roles[]"
        id="ADMIN"
        value="ADMIN">
        <label for="ADMIN">Administrator</label>

    <input
        class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
    >

```

```

        type="checkbox"
        name="roles[]"
        id="STAFF" v
        alue="STAFF">
        <label for="STAFF">Staff</label>

<input
    class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
}>
        type="checkbox"
        name="roles[]"
        id="CUSTOMER"
        value="CUSTOMER">
        <label for="CUSTOMER">Customer</label>

<div class="invalid-feedback">
    {{$errors->first('roles')}}<br>
</div>
<br>

<br>
        <label for="phone">Phone number</label>
        <br>
        <input value="{{$old('phone')}}" type="text" name="phone" class="form-control {{$errors->first('phone') ? "is-invalid" : ""}}">
        <div class="invalid-feedback">
            {{$errors->first('phone')}}<br>
        </div>

        <br>
        <label for="address">Address</label>
        <textarea name="address" id="address" class="form-control {{$errors->first('address') ? "is-invalid" : ""}}">{{$old('address')}}
        <div class="invalid-feedback">
            {{$errors->first('address')}}<br>
        </div>

        <br>
        <label for="avatar">Avatar image</label>
        <br>
        <input id="avatar" name="avatar" type="file" class="form-control {{$errors->first('avatar') ? "is-invalid" : ""}}">
        <div class="invalid-feedback">
            {{$errors->first('avatar')}}<br>
        </div>

<hr class="my-4">

        <label for="email">Email</label>
        <input value="{{$old('email')}}" class="form-control {{$errors->first('email') ? "is-invalid" : ""}} placeholder="user@mail.com" type="text" name="email" id="email"/>
        <div class="invalid-feedback">

```

```

{{ $errors->first('email')}}
</div>
<br>

<label for="password">Password</label>
<input class="form-control {{ $errors->first('password') ? "is-invalid" : "" }}" placeholder="password" type="password" name="password" id="password"/>
<div class="invalid-feedback">
    {{$errors->first('password')}}

```

@endsection

Dengan begini coba kamu submit di form create user tanpa mengisi data, maka kamu akan mendapati pesan error di masing-masing field.

### Validasi form edit user

Untuk membuat validasi di form edit, kita lakukan validasi di action `UserController@update`. Buka file `app/Http/Controllers/UserController.php` dan tambahkan kode validasi berikut ini di action `update`:

```
\Validator::make($request->all(), [
    "name" => "required|min:5|max:100",
    "roles" => "required",
    "phone" => "required|digits_between:10,12",
    "address" => "required|min:20|max:200",
])->validate();
```

Perhatikan, kita hanya melakukan validasi terhadap field `name`, `roles`, `phone` dan `address`. Karena hanya keempat field tersebut yang diizinkan untuk diedit di form. Sehingga kini action `update` kita terlihat seperti ini:

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

```
\Validator::make($request->all(), [
    "name" => "required|min:5|max:100",
    "roles" => "required",
    "phone" => "required|digits_between:10,12",
    "address" => "required|min:20|max:200",
])->validate();

$user = \App\User::findOrFail($id);

$user->name = $request->get('name');
$user->roles = json_encode($request->get('roles'));
$user->address = $request->get('address');
$user->phone = $request->get('phone');
$user->status = $request->get('status');

if($request->file('avatar')){
    if($user->avatar && file_exists(storage_path('app/public/' . $user->avatar))){
        \Storage::delete('public/' . $user->avatar);
    }
    $file = $request->file('avatar')->store('avatars', 'public');
    $user->avatar = $file;
}

$user->save();

return redirect()->route('users.edit', ['id' => $id])->with('status',
'User succesfully updated');
}
```

Selanjutnya kita ubah kode view di `resources/views/users/edit.blade.php` untuk menangkap dan menampilkan pesan error validasi, seperti ini:

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

@section('title') Edit User @endsection

@section('content')
<div class="col-md-8">
@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif
<form enctype="multipart/form-data" class="bg-white shadow-sm p-3"
action="{{route('users.update', ['id'=>$user->id])}}" method="POST">
    @csrf
    <input type="hidden" value="PUT" name="_method">
    <label for="name">Name</label>
    <input value="{{old('name') ? old('name') : $user->name}}" class="form-control {{$errors->first('name') ? "is-invalid" : ""}} placeholder="Full
```

```

Name" type="text" name="name" id="name"/>
<div class="invalid-feedback">
    {{$errors->first('name')}}
```

```
</div>
<br>
```

```
<label for="username">Username</label>
<input value="{{$user->username}}" disabled class="form-control"
placeholder="username" type="text" name="username" id="username"/>
<br>
```

```
<label for="">Status</label>
<br/>
```

```
<input {{$user->status == "ACTIVE" ? "checked" : ""}} value="ACTIVE"
type="radio" class="form-control" id="active" name="status"> <label
for="active">Active</label>
<input {{$user->status == "INACTIVE" ? "checked" : ""}} value="INACTIVE"
type="radio" class="form-control" id="inactive" name="status"> <label for="inactive">Inactive</label>
<br><br>
```

```
<label for="">Roles</label>
<br/>
```

```
<input
    type="checkbox"
    {{$in_array("ADMIN", json_decode($user->roles)) ? "checked" : ""}}
    name="roles[]"
    class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
    id="ADMIN"
    value="ADMIN">
<label for="ADMIN">Administrator</label>
```

```
<input
    type="checkbox"
    {{$in_array("STAFF", json_decode($user->roles)) ? "checked" : ""}}
    name="roles[]"
    class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
    id="STAFF"
    value="STAFF">
<label for="STAFF">Staff</label>
```

```
<input
    type="checkbox"
    {{$in_array("CUSTOMER", json_decode($user->roles)) ? "checked" : ""}}
    name="roles[]"
    class="form-control {{$errors->first('roles') ? "is-invalid" : ""}}"
    id="CUSTOMER"
    value="CUSTOMER">
<label for="CUSTOMER">Customer</label>
```

```
<div class="invalid-feedback">
    {{$errors->first('roles')}}
```

```

<br>
<label for="phone">Phone number</label>
<br>
<input type="text" name="phone" class="form-control {{$errors->first('phone') ? "is-invalid" : ""}} value="{{old('phone') ? old('phone') : $user->phone}}">
<div class="invalid-feedback">
    {{$errors->first('phone')}}</div>

<br>
<label for="address">Address</label>
<textarea name="address" id="address" class="form-control {{$errors->first('address') ? "is-invalid" : ""}}>{{old('address') ? old('address') : $user->address}}</textarea>
<div class="invalid-feedback">
    {{$errors->first('address')}}</div>
<br>

<label for="avatar">Avatar image</label>
<br>
Current avatar: <br>
@if($user->avatar)
    
    <br>
@else
    No avatar
@endif
<br>
<input id="avatar" name="avatar" type="file" class="form-control">
<small class="text-muted">Kosongkan jika tidak ingin mengubah
avatar</small>

<hr class="my-4">

<label for="email">Email</label>
<input value="{{$user->email}}" disabled class="form-control"
    {{$errors->first('email') ? "is-invalid" : ""}} "
    placeholder="user@mail.com" type="text" name="email" id="email"/>
<div class="invalid-feedback">
    {{$errors->first('email')}}</div>
<br>

<input class="btn btn-primary" type="submit" value="Simpan"/>
</form>
</div>
@endsection

```

Validasi form create category

Buka file `app/Http/Controllers/CategoryController.php` lalu pada action `store` tambahkan kode berikut ini:

```
\Validator::make($request->all(), [
    "name" => "required|min:3|max:20",
    "image" => "required"
])->validate();
```

Sehingga kini action `store()` di `CategoryController` menjadi seperti ini:

```
public function index(Request $request)
{
    $categories = \App\Category::paginate(10);

    $filterKeyword = $request->get('name');

    if($filterKeyword){
        $categories = \App\Category::where("name", "LIKE",
"%$filterKeyword%")->paginate(10);
    }

    return view('categories.index', ['categories' => $categories]);
}
```

Lalu buka file view `resources/views/categories/create.blade.php` dan buat agar view ini menampilkan pesan error validasi seperti ini:

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

@section('title') Create Category @endsection

@section('content')

<div class="col-md-8">

@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif

<form
    enctype="multipart/form-data"
    class="bg-white shadow-sm p-3"
    action="{{route('categories.store')}}"
    method="POST">

    @csrf
```

```
<label>Category name</label><br>
<input
    type="text"
    class="form-control {{$errors->first('name') ? "is-invalid" : ""}}"
    value="{{old('name')}}"
    name="name">
<div class="invalid-feedback">
    {{$errors->first('name')}}
```

## Validasi form edit category

Untuk validasi form edit, buka kembali **CategoryController** dan tambahkan kode berikut ini di action **update**:

```
$category = \App\Category::findOrFail($id);

\ Validator::make($request->all(), [
    "name" => "required|min:3|max:20",
    "image" => "required",
    "slug" => [
        "required",
        Rule::unique("categories")->ignore($category->slug, "slug")
    ]
])->validate();
```

Selain itu tambahkan pula di bagian atas file **use Illuminate\Validation\Rule;** seperti ini:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Validation\Rule;

class CategoryController extends Controller
```

Sehingga **CategoryController@update** menjadi seperti ini:

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

$category = \App\Category::findOrFail($id);

\Validator::make($request->all(), [
    "name" => "required|min:3|max:20",
    "image" => "required",
    "slug" => [
        "required",
        Rule::unique("categories")->ignore($category->slug, "slug")
    ]
])->validate();

$name = $request->get('name');
$slug = $request->get('slug');

$category->name = $name;
$category->slug = $slug;

if($request->file('image')){
    if($category->image && file_exists(storage_path('app/public/' .
$category->image))){
        \Storage::delete('public/' . $category->name);
    }

    $new_image = $request->file('image')->store('category_images',
'public');

    $category->image = $new_image;
}

$category->updated_by = \Auth::user()->id;

$category->slug = str_slug($name);

$category->save();
```

```
return redirect()->route('categories.edit', ['id' => $id])->with('status', 'Category successfully updated');
```

Perhatikan bahwa rule untuk field `slug` yaitu kita menggunakan `Illuminate\Validation\Rule` seperti ini

```
Rule::unique("categories")->ignore($category->slug, "slug")
```

Kenapa kita tidak menggunakan cara sebelumnya yaitu `unique:nama_table`, jawabannya karena kita ingin membuat field `slug` unique tapi kita kecualikan nilai `slug` yang sekarang di database.

Jika hanya menggunakan `unique:categories` maka kita akan selalu error ketika akan mengupdate category jika tidak mengubah `slug` karena slug sudah ada di database, yang sebetulnya adalah slug dari kategori yang diedit tersebut. Untuk itu kita perlu mengabaikan nilai dari slug yang sekarang dimiliki oleh kategori yang diupdate. Caranya menggunakan `Illuminate\Validation\Rule`.

Cara penggunaannya adalah pertama kita cari terlebih dahulu kategori yang akan diupdate menggunakan kode ini:

```
$category = \App\Category::findOrFail($id);
```

Setelah dapat kita gunakan kode berikut untuk mengabaikan nilai dari slug berdasarkan `$category->slug` seperti ini:

```
Rule::unique("categories")->ignore($category->slug, "slug")
```

Perlu diperhatikan pula bahwa dalam definisi rule di field ini kita menggunakan array seperti ini:

```
"slug" => ["required", Rule::unique("dst...")]
```

Bukan seperti ini:

```
"slug" => "required|Rule::unique('dst..')"
```

Setelah itu kita buat file view `resources/views/categories/edit.blade.php` agar dapat menangkap pesan error validasi dan menampilkannya di masing-masing field seperti ini:

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

@section('title') Edit Category @endsection

@section('content')

```

<div class="col-md-8">
@if(session('status'))
<div class="alert alert-success">
{{session('status')}}
</div>
@endif
<form
action="{{route('categories.update', ['id' => $category->id])}}"
enctype="multipart/form-data"
method="POST"
class="bg-white shadow-sm p-3"
>

@csrf

<input
type="hidden"
value="PUT"
name="_method">

<label>Category name</label> <br>
<input
type="text"
class="form-control {{$errors->first('name') ? "is-invalid" : ""}}"
value="{{old('name') ? old('name') : $category->name}}"
name="name">
<div class="invalid-feedback">
{{$errors->first('name')}}

```

```

        class="form-control {{$errors->first('image') ? "is-invalid" : ""}}"
            name="image">
        <small class="text-muted">Kosongkan jika tidak ingin mengubah
gambar</small>
        <div class="invalid-feedback">
            {{$errors->first('image')}}<br><br>

        <input type="submit" class="btn btn-primary" value="Update">
    </form>
</div>
@endsection

```

## Validasi form create book

Selanjutnya kita buat validasi untuk create book, buka file

app/Http/Controllers/BookController.php lalu pada action `store()` tambahkan kode validasi ini:

```

\ Validator::make($request->all(), [
    "title" => "required|min:5|max:200",
    "description" => "required|min:20|max:1000",
    "author" => "required|min:3|max:100",
    "publisher" => "required|min:3|max:200",
    "price" => "required|digits_between:0,10",
    "stock" => "required|digits_between:0,10",
    "cover" => "required"
])->validate();

```

Sehingga action `store()` menjadi seperti ini:

```

public function store(Request $request)
{
    \ Validator::make($request->all(), [
        "title" => "required|min:5|max:200",
        "description" => "required|min:20|max:1000",
        "author" => "required|min:3|max:100",
        "publisher" => "required|min:3|max:200",
        "price" => "required|digits_between:0,10",
        "stock" => "required|digits_between:0,10",
        "cover" => "required"
])->validate();

    $new_book = new \App\Book;
    $new_book->title = $request->get('title');
    $new_book->description = $request->get('description');
    $new_book->author = $request->get('author');
}

```

```

$new_book->publisher = $request->get('publisher');
$new_book->price = $request->get('price');
$new_book->stock = $request->get('stock');

$new_book->status = $request->get('save_action');

$new_book->slug = str_slug($request->get('title'));

$new_book->created_by = \Auth::user()->id;

$cover = $request->file('cover');

if($cover){
    $cover_path = $cover->store('book-covers', 'public');

    $new_book->cover = $cover_path;
}

$new_book->save();

$new_book->categories()->attach($request->get('categories'));

if($request->get('save_action') == 'PUBLISH'){
    return redirect()
        ->route('books.create')
        ->with('status', 'Book successfully saved and published');
} else {
    return redirect()
        ->route('books.create')
        ->with('status', 'Book saved as draft');
}
}

```

Kemudian buka file view `resources/views/users/create.blade.php` agar menangkap pesan error validasi dan menampilkannya di masing-masing input field seperti ini:

```

@extends('layouts.global')

@section('title') Create book @endsection

@section('content')
<div class="row">
<div class="col-md-8">
@if(session('status'))
    <div class="alert alert-success">
        {{session('status')}}
    </div>
@endif

<form
    action="{{route('books.store')}}">

```

```

method="POST"
enctype="multipart/form-data"
class="shadow-sm p-3 bg-white"
>

@csrf

<label for="title">Title</label> <br>
<input value="{{old('title')}}" type="text" class="form-control
{{\$errors->first('title') ? "is-invalid" : ""}} " name="title"
placeholder="Book title">
<div class="invalid-feedback">
  {{\$errors->first('title')}}

```

```

<br>

    <label for="publisher">Publisher</label> <br>
    <input value="{{old('publisher')}}" type="text" class="form-control"
{{ $errors->first('publisher') ? "is-invalid" : ""}} " id="publisher"
name="publisher" placeholder="Book publisher">
    <div class="invalid-feedback">
        {{$errors->first('publisher')}}

```

Validasi berikutnya adalah validasi form edit, buka kembali **BookController** dan tambahkan kode berikut ini pada action **update()**:

```
$book = \App\Book::findOrFail($id);

\Validator::make($request->all(), [
    "title" => "required|min:5|max:200",
    "slug" => [
        "required",
        Rule::unique("books")->ignore($book->slug, "slug")
    ],
    "description" => "required|min:20|max:1000",
    "author" => "required|min:3|max:100",
    "publisher" => "required|min:3|max:200",
    "price" => "required|digits_between:0,10",
    "stock" => "required|digits_between:0,10",
])->validate();
```

Pada field **slug** kembali kita menggunakan **Illuminate\Validation\Rule** untuk mengecek unique dengan mengabaikan data slug dari book yang sedang diedit.

Sehingga action **update()** kita menjadi seperti ini:

```
public function update(Request $request, $id)
{
    $book = \App\Book::findOrFail($id);

    \Validator::make($request->all(), [
        "title" => "required|min:5|max:200",
        "slug" => [
            "required",
            Rule::unique("books")->ignore($book->slug, "slug")
        ],
        "description" => "required|min:20|max:1000",
        "author" => "required|min:3|max:100",
        "publisher" => "required|min:3|max:200",
        "price" => "required|digits_between:0,10",
        "stock" => "required|digits_between:0,10",
    ])->validate();

    $book->title = $request->get('title');
    $book->slug = $request->get('slug');
    $book->description = $request->get('description');
    $book->author = $request->get('author');
    $book->publisher = $request->get('publisher');
    $book->stock = $request->get('stock');
    $book->price = $request->get('price');

    $new_cover = $request->file('cover');
```

```

if($new_cover){
    if($book->cover && file_exists(storage_path('app/public/') . $book->cover)){
        \Storage::delete('public/' . $book->cover);
    }

    $new_cover_path = $new_cover->store('book-covers', 'public');

    $book->cover = $new_cover_path;
}

$book->updated_by = \Auth::user()->id;

$book->status = $request->get('status');

$book->save();

$book->categories()->sync($request->get('categories'));

return redirect()->route('books.edit', ['id'=>$book->id])->with('status', 'Book successfully updated');

}

```

Jangan lupa untuk menambahkan kode berikut di bagian atas file

```
use Illuminate\Validation\Rule;
```

Sehingga bagian atas dari **BookController** menjadi seperti ini:

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class BookController extends Controller

```

## Daftar rule validasi keseluruhan

- **accepted**

Nilai harus **on**, **1**, **yes**, atau **true**. Digunakan untuk validasi "Terms of Service".

- **active\_url**

Harus valid A atau AAAA record berdasarkan **dns\_get\_records**

- **after:tanggal**

Digunakan untuk validasi tanggal, nilai field harus tanggal dan lebih besar dari tanggal yang dispecify. **tanggal** akan dimasukan ke fungsi **strtotime**, contoh:

```
"mulai" => "required|date|after:tomorrow"
```

Selain itu kita juga bisa mengisikan field lain sebagai tanggal acuan. Misalnya:

```
"selesai" => "required|date|after:mulai"
```

- **after\_or\_equal:tanggal**

Mirip seperti **after** bedanya nilai tanggal yang divalidasi boleh sama atau lebih besar dari tanggal yang dimasukan.

- **alpha**

Nilai field harus karakter alfabet.

- **alpha\_dash**

Nilai field bisa alpha-numeric termasuk dash (-) dan underscore (\_).

- **alpha\_num**

Nilai field harus alpha-numeric.

- **array**

Nilai field harus bertipe array.

- **bail**

Berhenti validasi rule berikutnya pada kegagalan validasi pertama.

- **before:tanggal**

Nilai harus **date** dan sebelum **tanggal** yang ditentukan.

- **before\_or\_equal:tanggal**

Nilai harus **date** dan nilainya boleh sama atau sebelum **tanggal** yang ditentukan

- **between:min,max**

Nilai harus di antara **min** dan **max**. Misalnya:

```
"price" => "between:10000|250000"
```

- **boolean**

Field harus bernilai `true`, `false`, 1, 0, "1" atau "0".

- **confirmed**

Field yang divalidasi harus punya field lain yang dibelakangnya bernama `_confirmation`, misalnya jika `confirmed` digunakan pada field `password` maka harus ada field lain dengan nama `password_confirmation`. Misalnya jika digunakan pada `phone` maka harus ada field lain berupa `phone_confirmation` dan nilainya harus sama.

- **date**

Nilai field harus berupa tanggal berdasarkan fungsi `strtotime`.

- **date\_equals:tanggal**

Nilai field harus tanggal dan sama dengan `tanggal` yang ditentukan

- **date\_format:format**

Nilai field harus memiliki format tanggal sesuai dengan `format` yang ditentukan.

- **different:fieldlain**

Nilai field harus berbeda dengan nilai `fieldlain`.

- **digits**

Nilai field harus berupa digits.

- **digits\_between:min,max**

Nilai field harus berupa digits dan di antara `min` dan `max`.

- **dimensions** Ini digunakan untuk dimensi image. Contoh penggunaan:

```
"avatar" => "required|dimensions:min_width=200,max_width=200"
```

Atau bisa juga menggunakan ratio seperti ini:

```
"avatar" => "required|dimensions:ratio=3/2"
```

- **distinct**

Digunakan untuk memastikan array tidak memiliki duplikat item.

- **email**

Nilai dari field harus berupa email yang valid.

- **exists:table,column (database)**

Memastikan bahwa nilai dari field **exists** atau ada di **tabel** dan **column** yang ditentukan.

- **file**

Field harus berupa file yang berhasil diupload.

- **filled** Field tidak boleh kosong.

- **gt:fieldlain**

Nilai field harus lebih besar dari nilai **fieldlain**.

- **gte:fieldlain**

Nilai field harus sama tau lebih besar dari **fieldlain**.

- **image**

Nilai field harus berupa image (jpg,png,bmp,gif,svg).

- **in:a,b,c,dst...**

Nilai dari field harus nilai yang ada di daftar **a**, **b**, **c**, dst..

- **in\_array:fieldlain**

Nilai field harus ada di **fieldlain**.

- **integer**

Nilai field harus berupa integer.

- **ip**

Nilai field harus berupa alamat IP valid.

- **ipv4**

Nilai field harus berupa alamat IP versi 4 valid.

- **ipv6**

Nilai field harus berupa alamat IP versi 6 valid.

- **JSON**

Nilai dari field harus berupa JSON string.

- **lt:fieldlain**

Nilai harus lebih kecil dari nilai **fieldlain**

- **lte**

Nilai harus sama atau lebih kecil dari nilai **fieldlain**

- **max:nilai**

Nilai field tidak boleh melebihi **nilai**

- **mimetypes:text/plain,...**

Nilai dari field harus memiliki mime type yang sama.

- **mimes:foo,bar**

Nilai dari field harus memiliki mimes sesuai di daftar **foo,bar**. Contoh:

```
"photo" => "required|mimes:jpg,gif,png"
```

- **min:nilai**

Nilai minimum dari field adalah **min**.

- **not\_in:foo,bar**

Kebalikan dari **in**.

- **not\_regex:pattern**

Nilai dari field tidak boleh sama dengan regex **pattern** yang diberikan.

- **nullable**

Nilai dari field boleh **null**

- **numeric**

Nilai dari field harus berupa **numeric**

- **present**

Field harus ada tapi nilainya boleh kosong.

- **regex:pattern**

Nilai dari field harus sesuai dengan REGEX **pattern**.

- **required**

Nilai dari field tidak boleh kosong. Yang termasuk kosong adalah:

- **null**
- string kosong ("")
- array kosong
- file yang diupload tapi tidak memiliki path.

- **required\_if:fieldlain,nilaidiharapkan**

Nilai dari field yang divalidasi tidak boleh kosong jika **fieldlain** memiliki nilai yaitu **nilaidiharapkan**

- **required\_unless:fieldlain,nilaidiharapkan**

Kebalikan dari **required\_if**, yaitu nilai dari field tidak boleh kosong, kecuali jika **fieldlain** bernilai **nilaidiharapkan**

- **required\_with:foo,bar**

Jika terdapat field **foo** atau **bar** (salah satu) maka field yang divalidasi tidak boleh kosong.

- **required\_with\_all:foo,bar** Jika semua field ini **foo** dan **bar** ada, maka field yang divalidasi tidak boleh kosong.

- **required\_without:foo,bar**

Jika **foo** atau **bar** tidak ada, maka field yang divalidasi harus ada.

- **required\_without\_all:foo,bar**

Jika **foo** dan **bar** tidak ada, maka field yang divalidasi harus ada.

- **same:fieldlain**

Nilai dari field yang divalidasi harus bernilai sama dengan nilai **fieldlain**

- **size:value**

Nilai dari field harus berukuran sesuai **value**.

- Untuk array berarti count.
- Untuk file berarti ukuran file.
- Untuk string berarti jumlah karakter.
- Untuk integer berarti nilai dari integer tersebut.

- **string**

Nilai harus berupa string.

- **timezone**

Nilai harus berupa timezone valid berdasarkan PHP **timezone\_identifiers\_list**

- **unique:table,column,except,idColumn**

Telah kita pelajari langsung sebelumnya.

- **url**

Nilai dari field harus berupa valid URL.

## Kustomisasi Error Page

Kita telah melakukan validasi terhadap form-form di management toko online. Tapi kita belum membuat melindungi form-form tersebut dari akses yang tidak diinginkan.

Contohnya sekarang, tanpa login pun kita tetap bisa mengakses form create user pada halaman <http://larashop.test/users/create>. Ini tentu tidak kita inginkan dan harus kita batasi siapa saja yang boleh mengakses fitur kelola resource di aplikasi kita.

Nah sebentar lagi kita akan belajar menggunakan **Gate** facade untuk membatasi hak akses ke fitur pengelolaan toko online.

Sebelum itu kita perlu membuat halaman kustom error, supaya kita bisa menggunakan method **abort()** di Laravel.

Contoh abort adalah seperti ini:

```
abort(401, 'Anda belum login');
```

Kode di atas akan menghasilkan error dengan status 401 alias unauthorized. Kode di atas akan mencari view di **resources/views/errors/401.blade.php**, jika tidak ada maka akan menggunakan halaman error bawaan dari browser yang mengakses. Kita akan membuat halaman error kustom untuk kode **401**, **403**, dan **404**. Adapun contoh untuk abort **403** dan **404** seperti ini:

```
abort(403, 'Anda tidak memiliki izin untuk mengakses halaman ini');
```

```
abort(404, 'Halaman yang Anda akses tidak ditemukan');
```

## Membuat halaman 401 / Unauthorized

Buat file view yaitu **resources/views/errors/401.blade.php**.

Lalu isikan kode berikut:

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

@section('content')
<div class="d-flex flex-row justify-content-center">
    <div class="col-md-6 text-center">
        <div class="alert alert-danger">
            <h1>401</h1>
            <h4>{{$exception->getMessage()}}</h4>
        </div>
    </div>
</div>
@endsection
```

`{{$exception->getMessage()}}` adalah cara untuk mengakses pesan yang kita isikan sebagai parameter kedua di method `abort()`.

## Membuat halaman 403 / Forbidden

Buat file view yaitu `resources/views/errors/403.blade.php`

Lalu isikan kode berikut:

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

@section('content')
<div class="d-flex flex-row justify-content-center">
    <div class="col-md-6 text-center">
        <div class="alert alert-danger">
            <h1>403</h1>
            <h4>{{$exception->getMessage()}}</h4>
        </div>
    </div>
</div>
@endsection
```

## Membuat halaman 404 / Notfound

Buat file view yaitu `resources/views/errors/404.blade.php`.

Lalu isikan kode berikut:

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

@section('content')
<div class="d-flex flex-row justify-content-center">
    <div class="col-md-6 text-center">
        <div class="alert alert-danger">
            <h1>404</h1>
            <h4>{{$exception->getMessage()}}</h4>
        </div>
    </div>
</div>
@endsection
```

## Gate Authorization

Selanjutnya kita akan membatasi hanya user dengan role tertentu saja yang bisa mengakses halaman management. Dalam authorization ini kita akan memanfaatkan fitur facade `Illuminate\Support\Facades\Gate` alias `Gate`.

Adapun untuk mappingnya adalah seperti ini:

Resource	Gate	Admin	Staff	Customer
users	manage-users	v	x	x
categories	manage-categories	v	v	x
books	manage-books	v	v	x
orders	manage-orders	v	v	x

Kita akan mendefinisikan 4 gate / gerbang sesuai dengan mapping di atas yaitu 'manage-users', 'manage-categories', 'manage-books' dan 'manage-orders'.

Cara mendefinisikan Gate adalah seperti ini:

```
Gate::define('manage-users', function($user){
    return $user->username = "azamuddin"
});
```

Kita definisikan gate bernama "manage-users", gate ini hanya mengizinkan user dengan username "azamuddin" untuk lewat, selainnya tidak boleh.

Nah tentu logika itu bisa kita sesuaikan dengan kebutuhan aplikasi kita, dan sesuai dengan Gate apa yang kita definisikan. Yang jelas setiap function callback dari Gate (parameter kedua) akan selalu mendapatkan argument `$user` yang mereferensikan user yang sedang login saat ini. Variabel `$user` tersebut sangat penting karena hak akses kaitannya sangat erat dengan mengecek status atau peran user.

Contoh lain:

```
Gate::define('update-post', function($user, $post){
    return $user->id == $post->creator_id;
});
```

Nah contoh di atas adalah Gate bernama "update-post", rule dari "update-post" adalah hanya boleh lewat jika `$user` memiliki ID yang sama dengan `$post` alias bahwa User merupakan pemilik / pembuat postingan tersebut. Nanti cara mengeceknya adalah seperti ini:

```
$post = \App\Post::findOrFail($post_id);

if(Gate::allows("update-post", $post)){
    // jika boleh
} else {
    // ternyata tidak boleh
}
```

Untuk kepentingan aplikasi manajemen toko online kita. Kita tidak memerlukan otorisasi per action. Akan tetapi cukup per resource / controller. Oleh karenanya kita akan mengacu pada tabel mapping otorisasi yang

## Mendefinisikan Gate di AuthServiceProvider

Berpikir **Gate** itu seperti route, jika definisi route kita letakkan di dalam direktori **routes**, **Gate** kita definisikan di **AuthServiceProvider**. Yaitu di **app/Providers/AuthServiceProvider.php** pada method **boot()**. Coba buka file **app/Providers/AuthServiceProvider.php** maka kamu akan mendapatkan file seperti ini:

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        'App\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        //
    }
}
```

Nah kamu perhatikan di kode di atas terdapat method **boot()** di situlah kita definisikan **Gate**. Mari kita definisikan **Gate** berdasarkan tabel mapping seperti ini:

```
public function boot(){
    $this->registerPolicies();

    Gate::define('manage-users', function($user){
        // TODO: logika untuk mengizinkan manage users
    });
}
```

```

Gate::define('manage-categories', function($user){
    // TODO: logika untuk mengizinkan manage categories
});

Gate::define('manage-books', function($user){
    // TODO: logika untuk mengizinkan manage books
});

Gate::define('manage-orders', function($user){
    // TODO: logika untuk mengizinkan manage orders
});
}

```

Pada kode di atas kita telah mendefinisikan 4 **Gate** yaitu **manage-users**, **manage-categories**, **manage-book** dan **manage-orders**. Tapi **Gate** tersebut perlu kita berikan logika yang akan menghasilkan TRUE. TRUE berarti user diizinkan untuk mengakses melalui **Gate** yang bersangkutan.

Misalnya begini, lihat kembali tabel mapping otorisasi. Untuk mengelola users (manage-users) hanya boleh user dengan role ADMIN maka kita harus mengecek apakah **\$user->roles** mengandung "ADMIN". Ingat bahwa **\$user->roles** bertipe JSON string array dan berisi 1 atau lebih dari 3 pilihan ini "ADMIN", "STAFF" dan "CUSTOMER", jadi ada beberapa kemungkinan isinya seperti ini:

- ["ADMIN"]
- ["STAFF"]
- ["CUSTOMER"]
- ["ADMIN", "STAFF"]
- ["STAFF", "CUSTOMER"]
- ["ADMIN", "CUSTOMER"]
- ["ADMIN", "STAFF", "CUSTOMER"]

Itu berarti untuk mengecek apakah user itu admin ada beberapa cara, misalnya menggunakan **in\_array** seperti ini:

```

$user->roles = '["ADMIN"]'; // bukan array, tapi JSON string array

in_array("ADMIN", json_decode($user->roles)); // TRUE

in_array("STAFF", json_decode($user->roles)); // FALSE

in_array("CUSTOMER", json_decode($user->roles)); // FALSE

```

Tapi dengan **in\_array** kita hanya bisa mengecek per role, bagaimana jika kita ingin mengecek apakah **\$user->roles** memiliki "ADMIN" atau "STAFF", kita bisa menggunakan **array\_intersect** seperti ini:

```

$user->roles = '["ADMIN"]'; // bukan array, tapi JSON string array

count(array_intersect(["ADMIN", "STAFF"], json_decode($user->roles))); //

```

TRUE

```
count(array_intersect(["STAFF", "CUSTOMER"], json_decode($user->roles)));
// FALSE

count(array_intersect(["ADMIN", "CUSTOMER"], json_decode($user->roles)));
// TRUE
```

Nah dengan menggunakan `array_intersect` dikombinasikan dengan `count` kita bisa mengecek apakah `$user->roles` memiliki salah satu dari beberapa role yang kita cari.

Dengan cara ini kita akan mengecek apakah user boleh mengelola suatu resource. Okay, mari kita isi `Gate` yang telah kita buat sesuai dengan rule di tabel mapping otorisasi sehingga akan terlihat seperti ini file `app/Providers/AuthServiceProvider.php` kita:

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        'App\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        Gate::define('manage-users', function($user){
            return count(array_intersect(["ADMIN"], json_decode($user->roles)));
        });

        Gate::define('manage-categories', function($user){
            return count(array_intersect(["ADMIN", "STAFF"], json_decode($user->roles)));
        });
    }
}
```

```

    });

    Gate::define('manage-books', function($user){
        return count(array_intersect(["ADMIN", "STAFF"],
json_decode($user->roles)));
    });

    Gate::define('manage-orders', function($user){
        return count(array_intersect(["ADMIN", "STAFF"],
json_decode($user->roles)));
    });
}
}

```

Catatan: Kita menggunakan `json_decode()` pada `$user->roles` karena `$user->roles` bertipe JSON String array, sehingga kita perlu menggunakan `json_decode()` untuk mengubahnya menjadi PHP Array.

Nah `Gate` kita telah siap digunakan untuk otorisasi. Kita bisa menggunakannya di mana saja kita membutuhkan pengecekan terhadap `Gate`. Misalnya di controller action, atau di view.

Untuk aplikasi ini, kita akan mendefinisikannya di function `__construct()` pada masing-masing `Controller` sehingga kita tidak perlu mengeceknya di tiap-tiap `action` controller. Tentu ini bergantung dengan mapping otorisasi dari aplikasi kita, sejauh mana aplikasi kita memiliki tingkat otorisasi. Nah untuk aplikasi kita, kita tidak memerlukan otorisasi yang berbeda-beda untuk masing-masing action, sehingga meletakkannya di `__construct()` lebih menghemat waktu dan tentu lebih rapi, karena kita letakkan di satu tempat, tidak perlu di tiap-tiap action. Ok?

## Otorisasi UserController

Buka `app/Http/UserController.php` lalu buat method `__construct()` di bagian paling atas definisi class sehingga bagian atasnya akan terlihat seperti ini:

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Gate;

class UserController extends Controller
{

    public function __construct(){
        // OTORISASI GATE
    }

    ... KODE SETERUSNYA TETAP
}

```

Setelah itu untuk **UserController** ini kita tambahkan pengecekan **Gate** pada method **\_\_construct()** seperti ini:

```
$this->middleware(function($request, $next){  
  
    if(Gate::allows('manage-users')) return $next($request);  
  
    abort(403, 'Anda tidak memiliki cukup hak akses');  
});
```

Kini resource manage user sudah memiliki otorisasi yaitu hanya boleh diakses oleh user yang memiliki role "ADMIN".

Untuk controller lainnya caranya sama persis tinggal kita ubah nama **Gate** yang digunakan misalnya **Gate::allows("manage-users")** jadi **Gate::allows("manage-categories")** untuk **CategoryController**

Okay mari kita praktikan.

### Otorisasi Category Controller

Buka **app/Http/CategoryController.php** lalu buat method **\_\_construct()** di bagian paling atas definisi class sehingga bagian atasnya akan terlihat seperti ini:

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
use Illuminate\Support\Facades\Gate;  
  
class CategoryController extends Controller  
{  
  
    public function __construct(){  
        // OTORISASI GATE  
    }  
  
    ... KODE SETERUSNYA TETAP
```

Setelah itu untuk **CategoryController** ini kita tambahkan pengecekan **Gate** pada method **\_\_construct()** seperti ini:

```
$this->middleware(function($request, $next){  
  
    if(Gate::allows('manage-categories')) return $next($request);
```

```
    abort(403, 'Anda tidak memiliki cukup hak akses');
});
```

Kini hanya user yang boleh lewat melalui **Gate** manage-categories yang bisa menggunakan **CategoryController** ini

### Otorisasi BookController

Buka **app/Http/BookController.php** lalu buat method **\_\_construct()** di bagian paling atas definisi class sehingga bagian atasnya akan terlihat seperti ini:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Gate;

class BookController extends Controller
{

    public function __construct(){
        // OTORISASI GATE
    }

    ... KODE SETERUSNYA TETAP
```

Setelah itu untuk **BookController** ini kita tambahkan pengecekan **Gate** pada method **\_\_construct()** seperti ini:

```
$this->middleware(function($request, $next){
    if(Gate::allows('manage-books')) return $next($request);

    abort(403, 'Anda tidak memiliki cukup hak akses');
});
```

### Otorisasi OrderController

Buka **app/Http/OrderController.php** lalu buat method **\_\_construct()** di bagian paling atas definisi class sehingga bagian atasnya akan terlihat seperti ini:

```
<?php
```

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Gate;

class OrderController extends Controller
{

    public function __construct(){
        // OTORISASI GATE
    }

    ... KODE SETERUSNYA TETAP
```

Setelah itu untuk **OrderController** ini kita tambahkan pengecekan **Gate** pada method **\_\_construct()** seperti ini:

```
$this->middleware(function($request, $next){
    if(Gate::allows('manage-orders')) return $next($request);
    abort(403, 'Anda tidak memiliki cukup hak akses');
});
```

## Source Code

Source code lengkap dapat kamu jumpai pada tautan berikut:

- <https://github.com/laravel-vue-book/larashop>

# Deployment

## Intro

Pada bab ini kita akan melakukan deploy alias memindahkan aplikasi kita ke server production agar bisa diakses oleh publik melalui internet. Pada bab ini kita akan belajar cara untuk melakukan deployment ke shared hosting dan juga cara deployment ke Virtual Private Server (VPS).

Contoh deployment ke shared hosting yang akan dibahas di sini menggunakan hosting dari domainesia. Sementara itu deployment ke VPS akan menggunakan DigitalOcean (Kami tidak ada afiliasi dengan DigitalOcean)

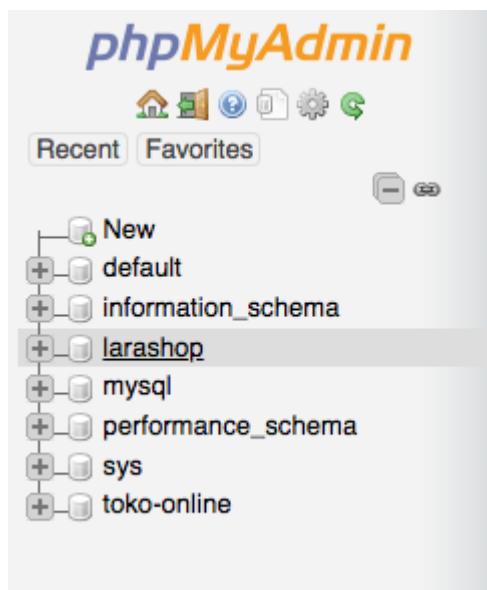
Catatan: Hosting provider yang kita gunakan untuk deploy pada buku ini adalah Domainesia <https://domainesia.com> sekaligus salah satu sponsor utama dari buku ini. Oleh karena itu jika kamu menggunakan provider lain silakan menyesuaikan. Web server yang telah terinstalasi pada hosting adalah Apache sehingga sedikit berbeda dengan web server saat development yaitu Nginx, tapi itu bukan masalah.

## Deployment ke shared hosting

### Persiapan

#### Export database (.sql) dari development

Pertama kita perlu memindahkan data-data saat development ke database yang ada di server / hostingan kita. Buka phpmyadmin lokal di alamat <http://localhost:8080> bagi pengguna docker. Lalu pilih database aplikasi larashop kita.



Setelah kita pilih database yang akan diexport, pilih menu export yang ada di menu atas seperti ini:

The screenshot shows the MySQL Workbench interface with the database 'larashop' selected. The 'Export' tab is highlighted with a red box. Below it, a table lists the database's tables and their properties. The table has columns for Table, Action, Rows, Type, Collation, Size, and Overhead.

Table	Action	Rows	Type	Collation	Size	Overhead
books	Browse  Structure  Search  Insert  Empty  Drop	3	InnoDB	utf8mb4_unicode_ci	16 Kib	-
book_category	Browse  Structure  Search  Insert  Empty  Drop	3	InnoDB	utf8mb4_unicode_ci	48 Kib	-
book_order	Browse  Structure  Search  Insert  Empty  Drop	3	InnoDB	utf8mb4_unicode_ci	48 Kib	-
categories	Browse  Structure  Search  Insert  Empty  Drop	27	InnoDB	utf8mb4_unicode_ci	32 Kib	-
migrations	Browse  Structure  Search  Insert  Empty  Drop	8	InnoDB	utf8mb4_unicode_ci	16 Kib	-
orders	Browse  Structure  Search  Insert  Empty  Drop	2	InnoDB	utf8mb4_unicode_ci	32 Kib	-
password_resets	Browse  Structure  Search  Insert  Empty  Drop	0	InnoDB	utf8mb4_unicode_ci	16 Kib	-
users	Browse  Structure  Search  Insert  Empty  Drop	15	InnoDB	utf8mb4_unicode_ci	48 Kib	-
<b>8 tables</b>	<b>Sum</b>		<b>InnoDB</b>	<b>utf8_general_ci</b>	<b>256 Kib</b>	<b>0 B</b>

Lalu pastikan option "save output to file" dan click Go.

### Output:

Rename exported databases/tables/columns  
 Use `LOCK TABLES` statement  
 Save output to a file

File name template:   use this for future exports

Character set of the file:

Compression:

Export tables as separate files  
 View output as text

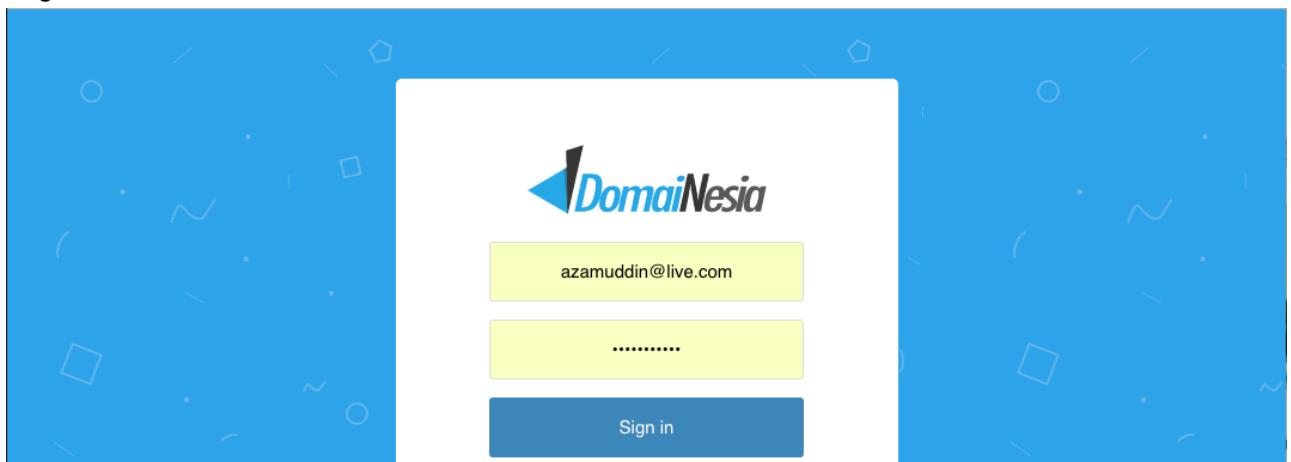
Skip tables larger than  MiB

Nah dengan begitu kita akan mendapatkan file .sql sesuai nama tabel yang kita export, misalnya **larashop.sql**. Simpan file ini karena kita akan mengimportnya di database server hosting kita.

Untuk itu mari buat sebuah tabel pada database di hostinger kita. Caranya adalah sebagai berikut:

1. Buka domainesia

2. Login



3. Pada halaman dahsboard, pilih products

4. Pada halaman dashboard products, pilih hostingnya yaitu pada contoh ini Vueshop

5. Pada halaman dashboard Vueshop, pilih MySQL Databases

6. Buat database baru dengan nama "backend" seperti ini:

7. Assign user ke database yang baru dibuat agar bisa mengakses via phpmyadmin hosting. Masih pada halaman MySQL, scroll terus kebawah. Jika belum membuat user untuk database buat terlebih dahulu:

# MySQL Users

## Add New User

**Username**

vueshopi\_ app

**Password**

.....

**Password (Again)**

.....

**Strength** ⓘ

Very Strong (100/100)

**Create User**

Lalu

scroll ke bawah lagi dan pilih dari daftar user lalu assign ke database yang baru kita buat seperti ini:

## Add User To Database

**User**

vueshopi\_app

**Database**

vueshopi\_backend

**Add**

Tentukan privilege atau hak akses user tersebut pada database. Sebaiknya kamu punya minimal dua jenis user, user admin bisa semua dan user aplikasi yang hanya bisa select, insert, update dan delete saja.

The screenshot shows the 'MySQL® Databases' section of the Domainesia control panel. It displays the user 'vueshopi\_app' and database 'vueshopi\_larashop'. Under 'Manage User Privileges', the 'ALL PRIVILEGES' checkbox is checked. Below it, several specific privilege checkboxes are also checked: ALTER, CREATE, CREATE TEMPORARY TABLES, DELETE, ALTER ROUTINE, CREATE ROUTINE, CREATE VIEW, and DROP.

Scroll ke bawah dan click "Make changes"

8. Setelah itu buka php myadmin. (kembali ke halaman produk / layanan domainesia) pilih phpMyAdmin.

The screenshot shows the 'Overview' tab selected in the Domainesia control panel. On the left, there's a 'Quick Shortcuts' sidebar with icons for Email Accounts, Forwarders, Autoresponders, File Manager, Backup, Subdomains, Addon Domains, Cron Jobs, MySQL Databases, and phpMyAdmin. The 'phpMyAdmin' icon is highlighted with a red box. To the right, there's a 'Usage Statistics' section with disk and bandwidth usage information.

9. Pada halaman phpMyAdmin hostingan kita, pilih database yang baru saja kita buat.

The screenshot shows the phpMyAdmin interface with the title 'phpMyAdmin'. In the left sidebar, under 'Recent', there are two databases listed: 'information\_schema' and 'vueshopi\_backend'. A large red arrow points from the previous screenshot to this 'information\_schema' entry.

10. Lalu click Import pada menu atas.

The screenshot shows the top navigation bar of the phpMyAdmin interface. The 'Import' button is highlighted with a red box. Other buttons in the top menu include Structure, SQL, Search, Query, Export, Operations, Privileges, Routines, and More.

Setelah itu pilih file yang telah kita export sebelumnya yang berekstensi (.sql) dan click Go.

Dengan begini kini database kita di hosting telah siap digunakan.

#### Mengubah konfigurasi di .env

Sebelum kita mengupload file-file source aplikasi Laravel ke domainesia, kita perlu menyesuaikan konfigurasi **.env** kita sesuaikan terutama bagian koneksi ke database dan APP\_URL seperti ini:

```
APP_NAME=Larashop
APP_ENV=production
APP_KEY=base64:lxz3F2S36U/w6CbZzhH1T5yWUzpJ+soV9gRtCYomGW8=
APP_DEBUG=false
APP_URL=http://domain-aplikasikita.com

DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=vueshopi_backend
DB_USERNAME=vueshopi_app
DB_PASSWORD=*****

# konfigurasi lainnya ....
```

The screenshot shows the contents of the .env file with several lines highlighted in red. A legend on the right side of the terminal window provides the following explanations for the highlighted fields:

- APP\_NAME=Larashop**
- APP\_ENV=production**
- APP\_KEY=base64:lxz3F2S36U/w6CbZzhH1T5yWUzpJ+soV9gRtCYomGW8=**
- APP\_DEBUG=false**
- APP\_URL=http://domain-aplikasikita.com**
- DB\_CONNECTION=mysql**
- DB\_HOST=localhost**
- DB\_PORT=3306**
- DB\_DATABASE=vueshopi\_backend**
- DB\_USERNAME=vueshopi\_app**
- DB\_PASSWORD=\*\*\*\*\***

Annotations with arrows point from the legend to the corresponding highlighted lines in the .env file:

- An arrow points from the text "Database yang kita buat di hostingan" to the DB\_DATABASE line.
- An arrow points from the text "User database hosting yang telah kita buat" to the DB\_USERNAME line.
- An arrow points from the text "Password user database" to the DB\_PASSWORD line.

Penjelasan:

Setelah itu kita jalankan perintah ini:

```
php artisan cache:clear
composer install --optimize-autoloader --no-dev
php artisan route:cache // optional
```

Jika kamu mengalami error saat menjalankan perintah **php artisan route:cache** seperti ini:

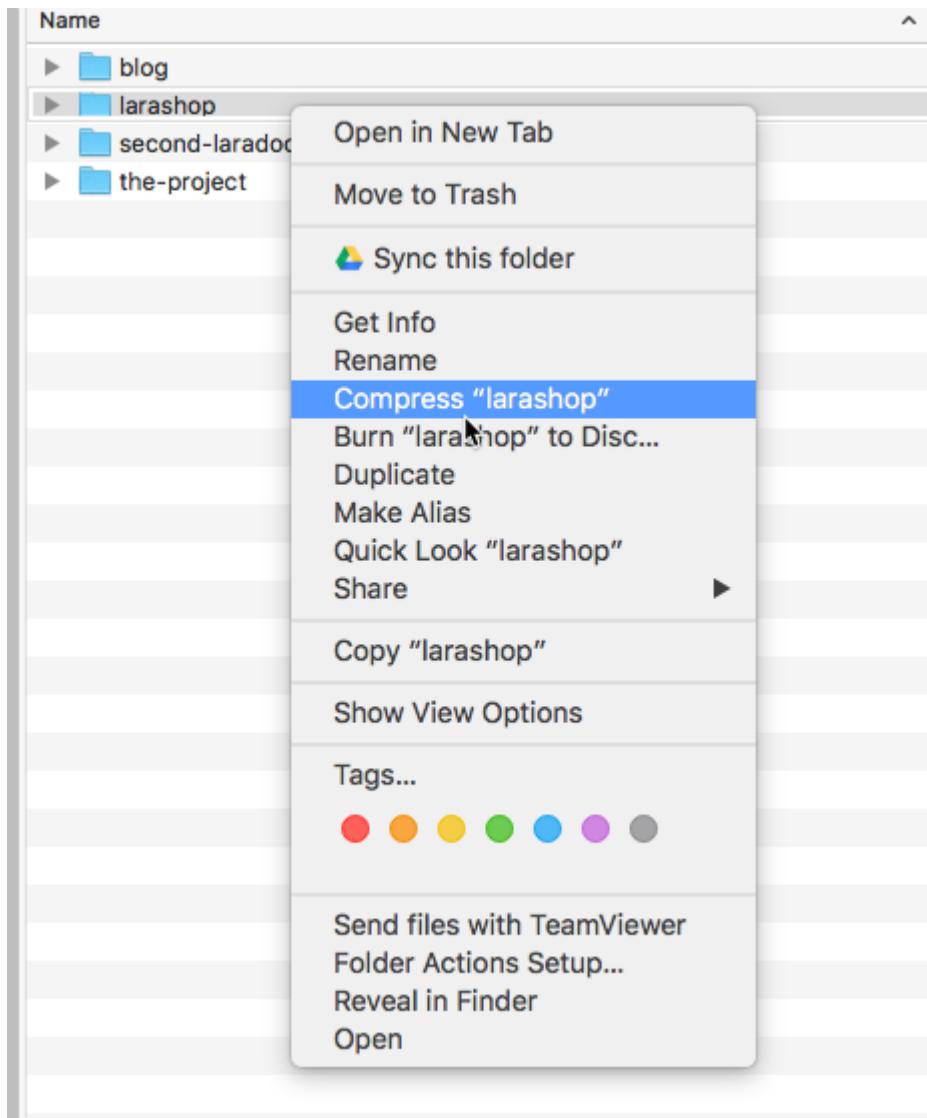
```
Unable to prepare route [/*/*] for serialization. Uses Closure.
```

Maka kamu bisa skip perintah tersebut. Atau jika tetap ingin melakukan caching terhadap route kamu harus menghilangkan penyebab error di atas yaitu adanya definisi route yang tidak menggunakan controller akan tetapi menggunakan Closure langsung di route.

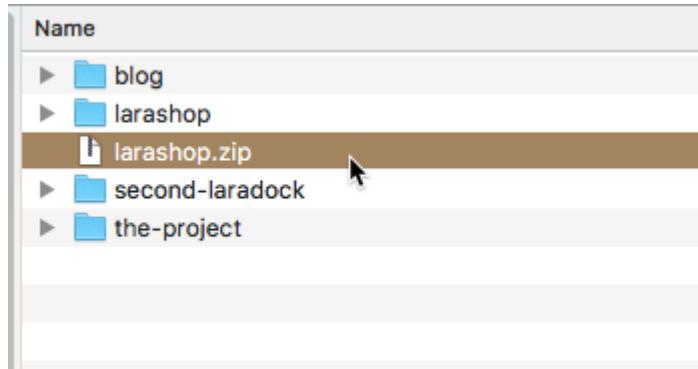
Penggunaan closure di definisi route menyebabkan kita tidak bisa melakukan caching route.

### Upload project zip ke hosting

Jika sudah maka kini saatnya kita upload ke hosting kita. Untuk memudahkan proses upload, kita compress dulu semua file ke dalam format zip.



Jika sudah maka kita akan memiliki file zip seperti ini:



Contoh compress di atas adalah di OS Mac untuk Windows dan Linux harap menyesuaikan ya.

1. Kembali login ke domainesia.com dan pilih produk yang kita gunakan. Lalu pilih "File manager"

A screenshot of the domainesia.com control panel. At the top, there are tabs for 'Overview', 'Access', 'Upgrade', and 'Addons'. Below these are two sections: 'Quick Shortcuts' and 'Usage Statistics'. In the 'Quick Shortcuts' section, there are several icons with labels: 'Email Accounts', 'Forwarders', 'Autoresponders', 'File Manager' (which is highlighted with a red box), 'Backup', 'Subdomains', 'Addon Domains', 'Cron Jobs', 'MySQL Databases', 'phpMyAdmin', and 'Awstats'. To the right, the 'Usage Statistics' section displays 'Disk Usage' (17, 358 M / 2048 M) and 'Bandwidth Usage' (0, 114 M / Unlimited M). A note at the bottom says 'Last Updated 18/09/2018 (01:47)'.

2. Upload file zip projek kita. Click "Upload" di menu atas file manager seperti ini:

A screenshot of the 'File Manager' interface. The top navigation bar includes buttons for '+ File', '+ Folder', 'Upload' (which is highlighted with a red box and has a red arrow pointing to it), 'Download', 'Delete', 'Restore', 'Rename', 'Edit', 'HTML Editor', 'Permissions', 'View', 'Extract', 'Search', 'Go', and 'Settings'. On the left, there's a sidebar with a tree view of the directory structure under '/home/vueshop1'. The main area shows a table of files with columns: Name, Size, Last Modified, Type, and Permissions. The table lists various files and directories like '.cagefs', '.cl.selector', '.cpanel', '.cphorde', '.htpasswd', '.razor', '.softaculous', '.spamassassin', '.trash', 'api.vueshop.id', 'backend.vueshop.id', 'deploy.vueshop.id', 'etc', 'larashop-api', 'logs', 'mail', 'public\_ftp', 'public\_html', 'ssl', and 'tmp'. The 'Permissions' column shows values like 0771, 0755, 0700, etc.

3. Pilih file zip yang tadi kita buat. Dan tunggu sampai selesai.

 File Upload

Select the file you want to upload to "/home/vueshopi".

Maximum file size allowed for upload: 1.65 GB

Overwrite existing files

Drop files here to start uploading

or

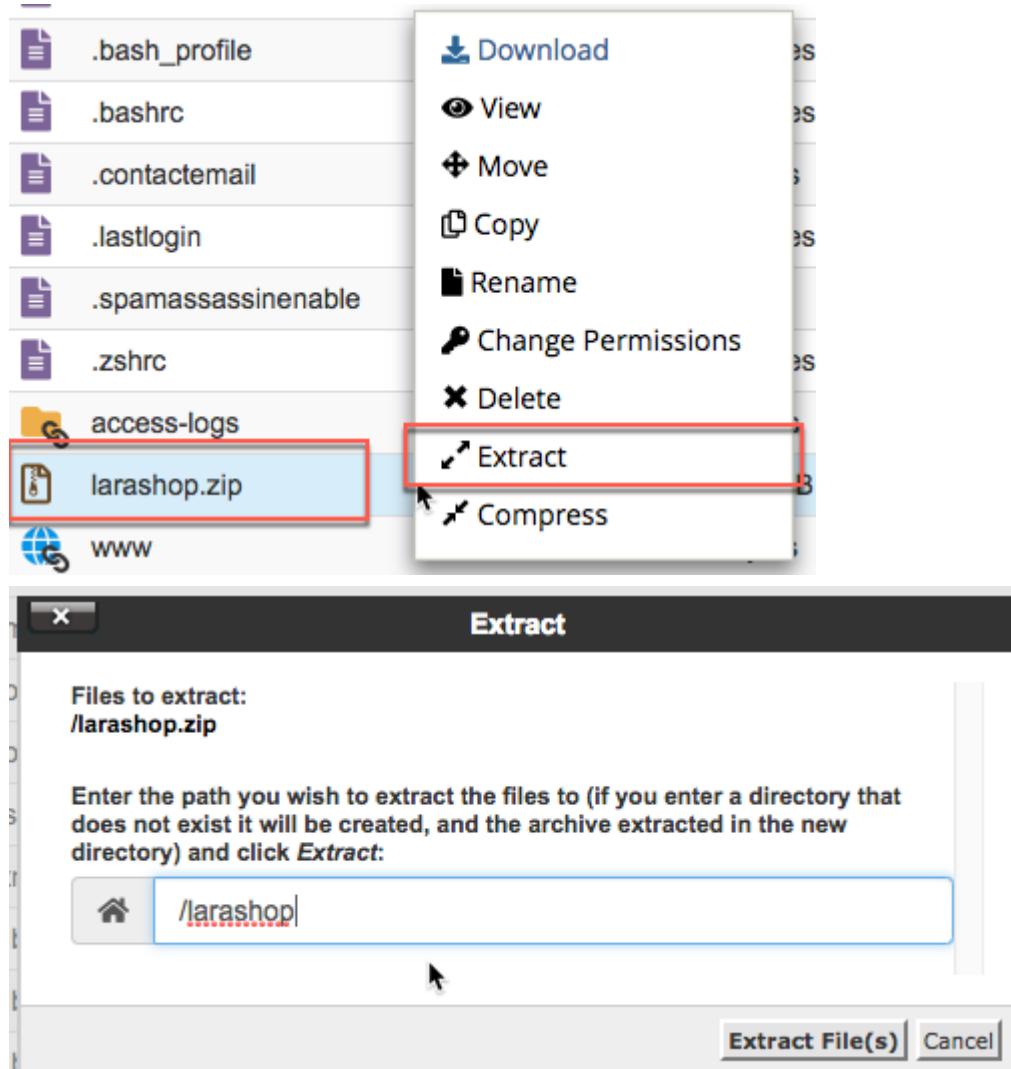
[Select File](#)

[Go Back to "/home/vueshopi"](#)

4. Setelah itu kembali ke filemanager dan kamu akan melihat file yang baru kamu upload.

	<a href="#">Home</a>	<a href="#">Up One Level</a>	<a href="#">Back</a>	<a href="#">Forward</a>	<a href="#">Reload</a>
	<a href="#">Name</a>	<a href="#">Size</a>	<a href="#">Last Modified</a>	<a href="#">Type</a>	<a href="#">Permissions</a>
	logs	4 KB	Yesterday, 6:52 PM	httpd/unix-directory	0700
	mail	4 KB	Yesterday, 5:56 PM	mail	0751
	public_ftp	4 KB	Sep 5, 2018, 4:44 PM	publicftp	0750
	public_html	4 KB	Yesterday, 6:33 AM	publichtml	0750
	ssl	4 KB	Yesterday, 5:52 PM	httpd/unix-directory	0755
	tmp	4 KB	Today, 4:11 PM	httpd/unix-directory	0755
	.bash_logout	18 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0644
	.bash_profile	193 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0644
	.bashrc	231 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0644
	.contactemail	18 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0640
	.lastlogin	416 bytes	Today, 2:08 PM	text/x-generic	0600
	.spamassassinenable	0 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0644
	.zshrc	658 bytes	Sep 5, 2018, 4:44 PM	text/x-generic	0644
	access-logs	34 bytes	Sep 5, 2018, 4:57 PM	httpd/unix-directory	0777
	<b>larashop.zip</b>	24.72 MB	Sep 16, 2018, 6:25 PM	package/x-generic	0644
	www	11 bytes	Sep 5, 2018, 4:44 PM	publichtml	0777

5. Ekstrak file tersebut ke dalam folder "/larashop"



**Extraction Results**

```
Archive: /home/vueshopi/larashop.zip
  creating: larashop/
  inflating: larashop/.DS_Store
  creating: __MACOSX/
  creating: __MACOSX/larashop/
  inflating: __MACOSX/larashop/._.DS_Store
  inflating: larashop/.editorconfig
  inflating: larashop/.env
  creating: larashop/.git/
  inflating: larashop/.git/COMMIT_EDITMSG
  inflating: larashop/.git/config
  inflating: larashop/.git/description
  inflating: larashop/.git/FETCH_HEAD
  inflating: larashop/.git/HEAD
  creating: larashop/.git/hooks/
  inflating: larashop/.git/hooks/applypatch-msg.sample
  inflating: larashop/.git/hooks/commit-msg.sample
  inflating: larashop/.git/hooks/post-update.sample
  inflating: larashop/.git/hooks/pre-applypatch.sample
```

**Close**

---

[Home](#) [Up One Level](#) [Back](#) [Forward](#) [Reload](#)  Select All  Unselect All [View Trash](#) [Empty Trash](#)

Name	Size	Last Modified	Type	Permissions
.cagefs	4 KB	Sep 9, 2018, 3:39 PM	httpd/unix-directory	0771
.cl.selector	4 KB	Today, 12:11 AM	httpd/unix-directory	0755
.cpanel	4 KB	Today, 11:06 AM	httpd/unix-directory	0700
.cphorde	4 KB	Sep 5, 2018, 4:44 PM	httpd/unix-directory	0700
.htpasswd	4 KB	Sep 5, 2018, 4:44 PM	httpd/unix-directory	0750
.razor	4 KB	Sep 17, 2018, 4:03 PM	httpd/unix-directory	0755
.softaculous	4 KB	Sep 9, 2018, 12:06 PM	httpd/unix-directory	0711
.spamassassin	4 KB	Sep 5, 2018, 4:44 PM	httpd/unix-directory	0700
.trash	4 KB	Yesterday, 4:29 PM	httpd/unix-directory	0700
api.vueshop.id	4 KB	Sep 9, 2018, 5:11 PM	httpd/unix-directory	0750
backend.vueshop.id	4 KB	Sep 17, 2018, 11:36 AM	httpd/unix-directory	0750
deploy.vueshop.id	4 KB	Sep 17, 2018, 5:55 PM	httpd/unix-directory	0750
etc	4 KB	Sep 17, 2018, 5:40 AM	httpd/unix-directory	0750
<b>larashop</b>	4 KB	Today, 11:19 AM	httpd/unix-directory	0755
larashop-api	4 KB	Sep 8, 2018, 12:44 PM	httpd/unix-directory	0755

6. Kemudian masuk ke folder tempat kita mengekstrak tadi dan pilih semua file. Kita harus memilih semua file termasuk hidden files, untuk itu kita setting terlebih dahulu filemanager kita agar menampilkan hidden files seperti ini:

File Manager

larashop/larashop Go

Home Up One Level Back Forward Reload Select All Unselect All View Trash Empty Trash

Name	Size	Last Modified	Type	Permissions
.git	4 KB	Yesterday, 3:50 PM	httpd/unix-directory	0755
app	4 KB	Jul 26, 2018, 1:53 PM	httpd/unix-directory	0755
bootstrap	4 KB	Jul 10, 2018, 2:44 PM	httpd/unix-directory	0755
config	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
database	4 KB	Jul 10, 2018, 2:44 PM	httpd/unix-directory	0755
public	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
resources	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
routes	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
storage	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755

Always open this directory in the future by default:

- Home Directory
- Web Root (public\_html or www)
- Public FTP Root (public\_ftp)
- Document Root for:  
vueshop.id

Show Hidden Files (dotfiles)

Disable Character Encoding Verification Dialogs

Save Cancel

Jika sudah,

selanjutnya kita pilih semua file di dalam folder "larashop"

Home Up One Level Back Forward Reload Select All Unselect All View Trash Empty Trash

Name	Size	Last Modified	Type	Permissions
.git	4 KB	Yesterday, 3:50 PM	httpd/unix-directory	0755
app	4 KB	Jul 26, 2018, 1:53 PM	httpd/unix-directory	0755
bootstrap	4 KB	Jul 10, 2018, 2:44 PM	httpd/unix-directory	0755
config	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
database	4 KB	Jul 10, 2018, 2:44 PM	httpd/unix-directory	0755
public	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
resources	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
routes	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
storage	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
tests	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755
vendor	4 KB	Yesterday, 3:08 PM	httpd/unix-directory	0755
.DS_Store	8 KB	Yesterday, 4:03 PM	text/x-generic	0644
.editorconfig	234 bytes	Jul 10, 2018, 2:44 PM	text/x-generic	0644
.env	690 bytes	Sep 16, 2018, 5:51 PM	text/x-generic	0644
.gitattributes	111 bytes	Jul 10, 2018, 2:44 PM	text/x-generic	0644
.gitignore	215 bytes	Jul 15, 2018, 9:32 PM	text/x-generic	0644

## Menjalankan perintah artisan storage:link di server hosting

Selanjutnya agar file-file di folder `storage/app/public` dapat diakses melalui url kita perlu menjalankan perintah `storage:link`. Perintah ini pernah kita jalankan, tapi itu di lokal / komputer kita sehingga kita perlu menjalankannya di server.

Dengan asumsi kita tidak memiliki akses ke SSH hosting kita. Maka kita bisa menggunakan alternatif selain SSH yaitu cronjob. Kita akan menjalankan perintah tersebut, caranya seperti ini:

1. Pada overview produk kita di domainesia click cronjob

The screenshot shows the Domainesia hosting control panel interface. At the top, there are tabs: Overview (which is selected and highlighted in blue), Access, Upgrade, and Addons. Below the tabs, there are two main sections: 'Quick Shortcuts' and 'Usage Statistics'. The 'Quick Shortcuts' section contains icons for Email Accounts, Forwarders, Autoresponders, File Manager, Backup, Subdomains, Addon Domains, and Cron Jobs. The 'Cron Jobs' icon is highlighted with a red box. The 'Usage Statistics' section displays disk usage (17%, 353 M / 2048 M) and bandwidth usage (0%, 116 M / Unlimited M). A note at the bottom right of the stats says 'Last Updated 19/09/2018 (01:45)'.

2. Setelah itu pada halaman tambah cronjob pilih settingan "Once per minute" seperti ini:

**Common Settings**

Once Per Minute(\* \* \* \* \*)

**Minute:**  
\* Once Per Minute(\*)

**Hour:**  
\* Every Hour (\*)

**Day:**  
\* Every Day (\*)

**Month:**  
\* Every Month (\*)

**Weekday:**  
\* Every Day (\*)

**Command:**

```
cd /home/vueshopi/backend.vueshop.id && php -d register_argc_argv=On artisan stc
```

**Add New Cron Job**

Pada input "Command" ketikan perintah ini:

```
cd /home/[FOLDERMU]/public_html && php -d register_argc_argv=On artisan storage:link
```

Ganti [FOLDERMU] dengan folder home, lihat di file manager seperti ini:

The screenshot shows the cPanel File Manager interface. On the left, a tree view lists the contents of the directory `/home/vueshopi`. The `storage` folder is highlighted with a red border and a red arrow points to it from the text above. The right panel shows a list of files and folders with their names.

Name
.cagefs
.cl.selector
.cpanel
.cphorde
.htpasswd
.razor
.softaculous
.spamassassin
.trash
api.vueshop.id
backend.vueshop.id
deploy.vueshop.id
etc
larashop
larashop-api

Sehingga contoh jadi command adalah seperti ini:

```
cd /home/vueshopi/public_html && php -d register_argc_argv=On artisan
storage:link
```

Click "Add new cronjob" dan tunggu 1 - 5 menit, lalu cek pada filemanager di folder `public_html/public` pastikan folder storage sudah muncul seperti ini:

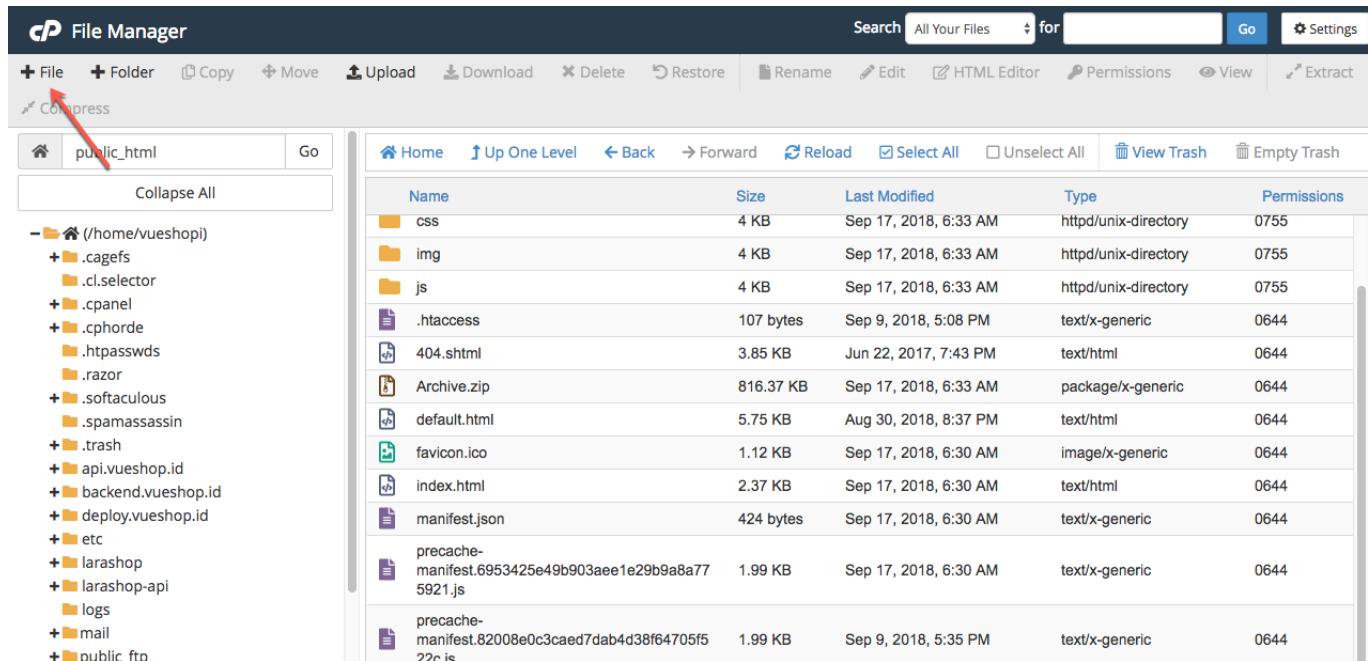
File Manager					
<a href="#">Home</a> <a href="#">Up One Level</a> <a href="#">Back</a> <a href="#">Forward</a> <a href="#">Reload</a> <input checked="" type="checkbox"/> Select All <input type="checkbox"/> Unselect All <a href="#">View Trash</a> <a href="#">Empty Trash</a>					
Name	Size	Last Modified	Type	Permissions	
css	4 KB	Jul 15, 2018, 9:32 PM	httpd/unix-directory	0755	
js	4 KB	Jul 10, 2018, 2:44 PM	httpd/unix-directory	0755	
polished	4 KB	Jun 4, 2018, 2:23 AM	httpd/unix-directory	0755	
.htaccess	594 bytes	Sep 16, 2018, 9:40 PM	text/x-generic	0644	
index.php	1.78 KB	Jul 10, 2018, 2:44 PM	application/x-httpd-php	0644	
robots.txt	24 bytes	Jul 10, 2018, 2:44 PM	text/plain	0644	
storage	52 bytes	Sep 17, 2018, 9:37 AM	httpd/unix-directory	0777	

Jika sudah ada folder storage maka link telah berhasil dan kamu bisa menghapus cronjob yang kita buat sebelumnya karena sudah tidak diperlukan lagi.

## Menghapus "public" dari URL

Misalkan domain kamu adalah "larashop.id" maka kini kamu bisa mengakses aplikasimu di <http://larashop.id/public>. Jika ingin menghapus "public" sehingga kita bisa mengaksesnya hanya dengan <http://larashop.id> lakukan langkah-langkah berikut ini:

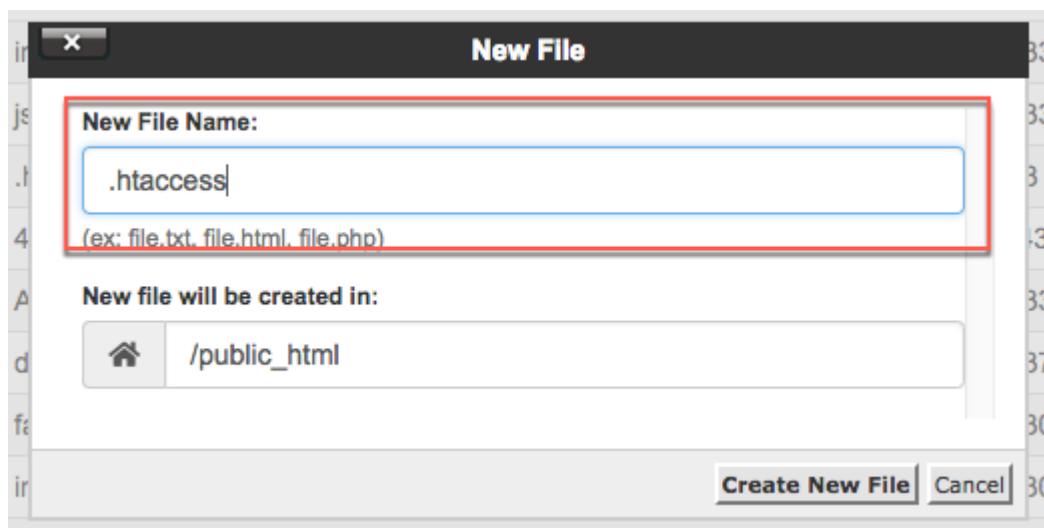
1. Masuk ke filemanager hostingan kita dan click tambah file



The screenshot shows the cPanel File Manager interface. On the left, there's a tree view of the directory structure under '/home/vueshop1'. On the right, a list view shows files and directories with columns for Name, Size, Last Modified, Type, and Permissions. A red arrow points to the '+ File' button in the top left of the interface.

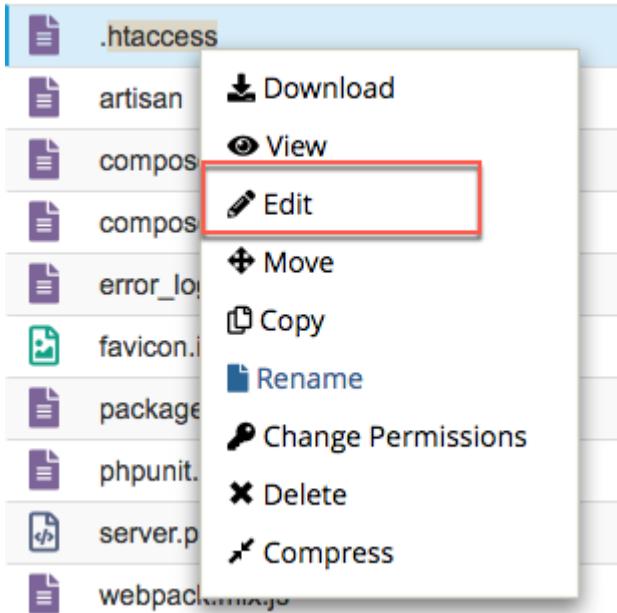
Name	Size	Last Modified	Type	Permissions
css	4 KB	Sep 17, 2018, 6:33 AM	httpd/unix-directory	0755
img	4 KB	Sep 17, 2018, 6:33 AM	httpd/unix-directory	0755
js	4 KB	Sep 17, 2018, 6:33 AM	httpd/unix-directory	0755
.htaccess	107 bytes	Sep 9, 2018, 5:08 PM	text/x-generic	0644
404.shtml	3.85 KB	Jun 22, 2017, 7:43 PM	text/html	0644
Archive.zip	816.37 KB	Sep 17, 2018, 6:33 AM	package/x-generic	0644
default.html	5.75 KB	Aug 30, 2018, 8:37 PM	text/html	0644
favicon.ico	1.12 KB	Sep 17, 2018, 6:30 AM	image/x-generic	0644
index.html	2.37 KB	Sep 17, 2018, 6:30 AM	text/html	0644
manifest.json	424 bytes	Sep 17, 2018, 6:30 AM	text/x-generic	0644
precache-manifest.6953425e49b903aee1e29b9a8a775921.js	1.99 KB	Sep 17, 2018, 6:30 AM	text/x-generic	0644
precache-manifest.82008e0c3caed7dab4d38f64705f522c.js	1.99 KB	Sep 9, 2018, 5:35 PM	text/x-generic	0644

2. Berikan nama file tersebut adalah ".htaccess" jangan salah ketik ya.



The screenshot shows a 'New File' dialog box. The 'New File Name:' input field contains '.htaccess' and has a red border around it. Below the input field is a note '(ex: file.txt, file.html, file.php)'. The 'New file will be created in:' dropdown shows '/public\_html'. At the bottom right are 'Create New File' and 'Cancel' buttons.

3. Klik kanan pada file yang baru saja kita buat tadi (.htaccess)

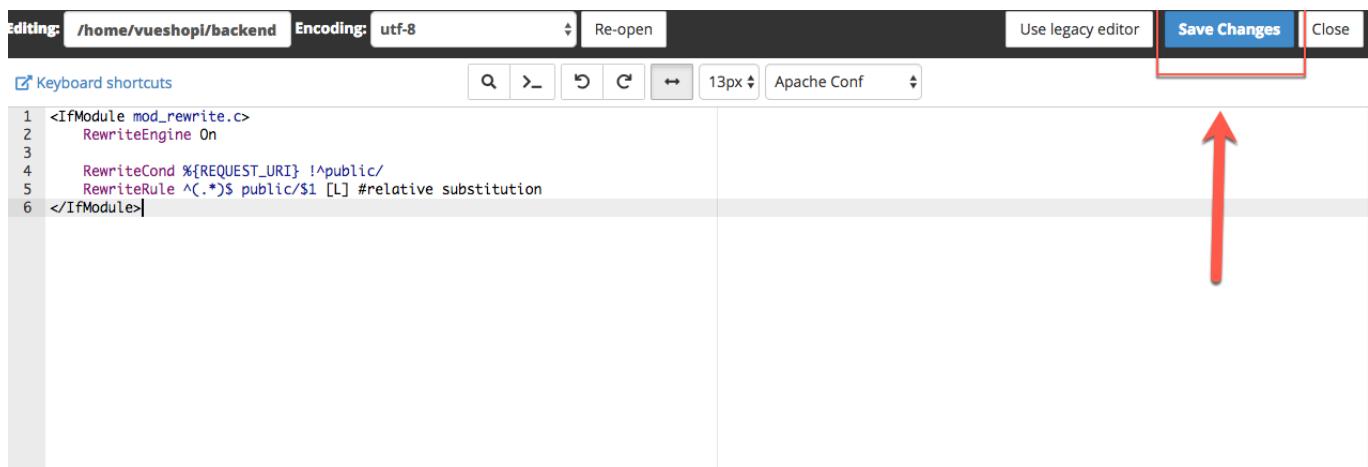


4. Paste kan kode berikut ini:

```
<IfModule mod_rewrite.c>
    RewriteEngine On

    RewriteCond %{REQUEST_URI} !^public/
    RewriteRule ^(.*)$ public/$1 [L] #relative substitution
</IfModule>
```

5. Click save changes



Selesai. Dengan demikian maka selesailah proses deployment kita ke shared hosting.

## Deployment ke VPS

Materi selanjutnya adalah kita akan belajar mendeploy aplikasi Larashop ke VPS di DigitalOcean. Apa saja yang dibutuhkan jika kita ingin mendeploy ke VPS, berikut daftarnya:

1. VPS Tentu saja kita harus memiliki sebuah Virtual Private Server, kamu bisa menggunakan penyedia manapun misalnya DigitalOcean, Vulter atau yang lain. Perlu diperhatikan bahwa akan mendeploy ke

DigitalOcean, jika pembaca menggunakan provider lain silahkan menyesuaikan.

2. Akun gitlab / bitbucket / github Karena kita akan memanfaatkan git selain untuk version control juga untuk mentransfer file dari komputer lokal kita VPS. File-file yang akan ditransfer menggunakan git selain source code project larashop juga file-file laradock di komputer kita. Kenapa file-file laradock juga perlu ditransfer dari lokal ke VPS, kenapa tidak install laradock di server saja?

Jawabannya adalah kita ingin memastikan bahwa environment kita benar-benar sama antara saat development di lokal dengan di VPS sehingga tidak ada lagi keluhan klasik seperti "Di komputerku jalan kok, kenapa di server ga jalan ya?". Jika kita menginstall laradock baru di server ada dua permasalahan, yaitu:

- versi laradock yang kita install di server bisa saja berbeda dengan di lokal (lebih baru) dan menggunakan konfigurasi berbeda.
- jika kita telah melakukan kustomisasi konfigurasi di lokal maka kita harus konfigurasi ulang di VPS.

Untuk itu strategi untuk melakukan deploy aplikasi Laravel kita ke VPS adalah sebagai berikut:

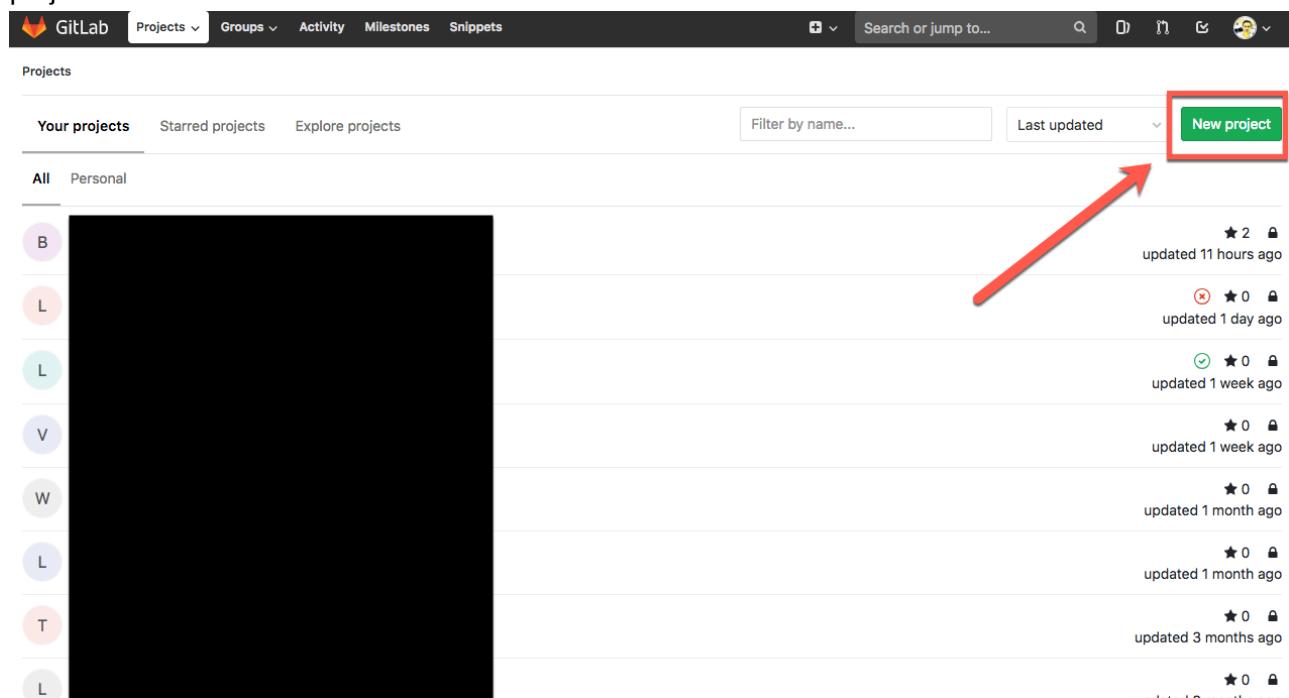
1. Push laradock kita di lokal ke repository di gitlab / bitbucket / github
2. Masuk VPS lalu pull repository laradock pada poin 1
3. Push projek larashop kita ke repository di gitlab / bitbucket / github
4. Pada VPS pull juga repository larashop kita
5. Jalankan docker-compose pada laradock seperti di lokal.

Secara garis besar begitu strategi yang akan kita lakukan. Adapun detail langkah-langkah yang harus kita lakukan akan segera kita bahas.

Tambahkan remote di repository laradock lokal

1. Buat repository git untuk laradock kita

Penulis akan memberikan contoh cara membuat repository git baru di platform gitlab. Jika kamu belum punya akun gitlab, silahkan buat akun terlebih dahulu. Jika sudah punya, login ke gitlab lalu click "New project"



ketikan nama repository baru kita misalnya "laradock" lalu pilih visibility "private" atau yang lainnya. Jika ingin repo ini hanya bisa diakses oleh kita sendiri, pilih private, kabar baiknya gitlab menyediakan unlimited private repository gratis. Jika sudah click create project.

Blank project      Create from template      Import project      CI/CD for external repo

**Project name**  
laradock

**Project URL**  
https://gitlab.com/ mas.azamuddin

**Project slug**  
laradock-development

Want to house several dependent projects under the same namespace? [Create a group](#).

**Project description (optional)**

Description format

**Visibility Level ?**

 Private  
Project access must be granted explicitly to each user.

 Internal  
The project can be accessed by any logged in user.

 Public  
The project can be accessed without any authentication.

**Initialize repository with a README**  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

## 2. Copy repository url dari project yang baru kita buat

Muhammad Azamuddin > laradock > Details

Project 'laradock' was successfully created.

**laradock**  

Project ID: 8576299

0  Star  HTTPS <https://gitlab.com/mas.azamuddin/laradock> 

 SSH

The repository   HTTPS

If you already have files you can push them using the command line instructions below.

Note that the master branch is automatically protected. [Learn more about protected branches](#)

You can automatically build and test your application if you enable Auto DevOps for this project. You can automatically deploy it as well, if you add a Kubernetes cluster.

Otherwise it is recommended you start with one of the options below.

Files (0 Bytes) Commits (0) Branches (0) Tags (0)

New file Add Readme Enable Auto DevOps Add Kubernetes cluster

## 3. Buka terminal lalu pindah ke direktori laradock kita, misalnya:

```
cd user/azamuddin/laravel-projects/laradock
```

user/azamuddin bisa berbeda bergantung di mana kamu meletakkan folder laravel-projects.

4. Pada folder tersebut commit dulu setiap perubahan sudah kita lakukan di laradock

```
git commit -a -m "Persiapan deploy ke VPS"
```

5. Tambahkan remote repository laradock kita

```
git remote add origin  
URL_REPOSITORY_LARADOCK_KITA_DI_GITLAB_BITBUCKET_ATAU_GITHUB
```

Ganti URL\_REPOSITORY\_LARADOCK\_KITA\_DI\_GITLAB\_BITBUCKET\_ATAU\_GITHUB dengan url sungguhan repository laradock yang telah kita buat sebelumnya.

6. Push laradock kita ke repository remote

```
git push origin master
```

Tambahkan remote di repository larashop lokal

1. Ulangi langkah pada poin 1 di bahasan sebelumnya kali ini ganti nama project / repository dengan "larashop"

Blank project	Create from template	Import project	CI/CD for external repo
<b>Project name</b> <input type="text" value="larashop"/>			
Project URL	Project slug		
<input type="text" value="https://gitlab.com/"/> mas.azamuddin	<input type="text" value="laradock-development"/>		
Want to house several dependent projects under the same namespace? <a href="#">Create a group.</a>			
<b>Project description (optional)</b> <input type="text" value="Description format"/>			
<b>Visibility Level</b> <a href="#">?</a> <input checked="" type="radio"/> Private Project access must be granted explicitly to each user. <input type="radio"/> Internal The project can be accessed by any logged in user. <input type="radio"/> Public The project can be accessed without any authentication.			
<input type="checkbox"/> <b>Initialize repository with a README</b> Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.			

## 2. Copy url repository project yang baru kita buat tadi.

Muhammad Azamuddin > laradock > [Details](#)

Project 'laradock' was successfully created.

**larashop** Private [Add license](#)

Project ID: 8576299

0 ⭐ Star [HTTPS](#) <https://gitlab.com/mas.azamuddin/larashop> + Global

**The repository** [HTTPS](#)

If you already have files you can push them using the command line instructions below.

Note that the master branch is automatically protected. [Learn more about protected branches](#)

You can automatically build and test your application if you [enable Auto DevOps](#) for this project. You can automatically deploy it as well, if you [add a Kubernetes cluster](#).

Otherwise it is recommended you start with one of the options below.

Files (0 Bytes) Commits (0) Branches (0) Tags (0)

New file Add Readme Enable Auto DevOps Add Kubernetes cluster

## 3. Buka terminal lalu pindah ke direktori laradock kita, misalnya:

```
cd user/azamuddin/laravel-projects/larashop
```

user/azamuddin bisa berbeda bergantung di mana kamu meletakkan folder laravel-projects.

4. Pada folder tersebut commit dulu setiap perubahan sudah kita lakukan di laradock

```
git commit -a -m "Persiapan deploy ke VPS"
```

5. Tambahkan remote repository laradock kita

```
git remote add origin  
URL_REPOSITORY_LARASHOP_KITA_DI_GITLAB_BITBUCKET_ATAU_GITHUB
```

Ganti URL\_REPOSITORY\_LARASHOP\_KITA\_DI\_GITLAB\_BITBUCKET\_ATAU\_GITHUB dengan url sungguhan repository aplikasi Laravel (larashop) kita.

6. Push laradock kita ke repository remote

```
git push origin master
```

## Persiapkan VPS di Digital Ocean

Buat akun di digitalocean.com, lalu login. Setelah itu create droplet baru. Anggap saja droplet adalah sebutan lain untuk VPS di digitalocean.

The screenshot shows the DigitalOcean control panel. At the top, there's a search bar labeled 'Search by Droplet name or IP (Cmd+B)' and a green 'Create ^' button. To the right of the search bar are two small icons: a gear and a bell with a '11' notification badge. Below the search bar, the main heading is 'Create Droplets'. Underneath, there's a section titled 'Choose an image' with a question mark icon. Below this, there are four distribution options: 'Ubuntu' (selected), 'FreeBSD', 'Fedora', and 'Debian'. Each option has a dropdown menu below it labeled 'Select version'. To the right of these options is a sidebar with several sections: 'Droplets' (highlighted with a red box and a red arrow pointing to it), 'Volumes', 'Domains/DNS', 'Cloud Firewalls', 'Floating IPs', 'Load Balancers', 'Alert Policies', and 'Spaces'. The 'Droplets' section contains sub-links: 'Create cloud servers', 'Add storage to Droplets', 'Route your existing domains', 'Increase Droplet security', 'Reserve IP addresses for Droplets', 'Distribute traffic to Droplets', 'Monitor your Droplets', and 'Store and serve static assets'.

Setelah itu pilih one-click-apps dan pilih droplet yang menggunakan docker langsung. Itu berarti kita tidak perlu secara manual menginstall docker.

Choose an image ?

**Distributions** Container distributions **One-click apps** Custom images

- Discourse 2.0.20180802 on 18.04
- Django 1:1.11.11 on 18.04
- Docker 18.06.1~ce~3 on 18.04**
- Dokku 0.12.12 on 18.04
- Ghost on 18.04
- GitLab 11.2.3-ce.0 on 18.04
- LAMP on 18.04
- LEMP on 18.04
- MongoDB 4.0.2 on 18.04
- MySQL on 18.04
- NodeJS 8.10.0 on 18.04
- PhpMyAdmin on 18.04
- Ruby-on-Rails on 18.04
- WordPress on 18.04

Kemudian pilih size berdasarkan harga yang mau kamu bayar per bulannya.

MEMORY	vCPUs	SSD DISK	TRANSFER	PRICE
<b>1 GB</b>	1 vCPU	25 GB	1 TB	<b>\$5/mo \$0.007/hr</b>
<b>2 GB</b>	1 vCPU	50 GB	2 TB	\$10/mo \$0.015/hr
<b>3 GB</b>	1 vCPU	60 GB	3 TB	\$15/mo \$0.022/hr
<b>2 GB</b>	2 vCPUs	60 GB	3 TB	\$15/mo \$0.022/hr
<b>1 GB</b>	3 vCPUs	60 GB	3 TB	\$15/mo \$0.022/hr
<b>4 GB</b>	2 vCPUs	80 GB	4 TB	\$20/mo \$0.030/hr
<b>8 GB</b>	4 vCPUs	160 GB	5 TB	\$40/mo \$0.060/hr
<b>16 GB</b>	6 vCPUs	320 GB	6 TB	\$80/mo \$0.119/hr
<b>32 GB</b>	8 vCPUs	640 GB	7 TB	\$160/mo \$0.238/hr

MEMORY	DEDICATED vCPUs	SSD DISK	TRANSFER	PRICE
4 GB	<b>2 vCPUs</b>	25 GB	4 TB	\$40/mo \$0.060/hr
8 GB	<b>4 vCPUs</b>	50 GB	5 TB	\$80/mo \$0.119/hr
16 GB	<b>8 vCPUs</b>	100 GB	6 TB	\$160/mo \$0.238/hr
32 GB	<b>16 vCPUs</b>	200 GB	7 TB	\$320/mo \$0.476/hr
64 GB	<b>32 vCPUs</b>	400 GB	9 TB	\$640/mo \$0.952/hr

Scroll kebawah lagi dan pilih di region mana kita mau buat VPS kita, misalnya kita pilih singapura karena pengguna aplikasi kita kebanyakan dari Indonesia. Pilih yang paling dekat.

## Choose a datacenter region

 New York	 San Francisco	 Amsterdam	 Singapore	 London	 Frankfurt
1    2    3	1            2	2    3	1	1	1
 Toronto	 Bangalore				
1	1				

Setelah itu berikan nama yang mudah diingat untuk hostname VPS kita lalu klik "Create"

## Finalize and create

### How many Droplets?

Deploy multiple Droplets with the same configuration .

-	1 Droplet	+
---	-----------	---

### Choose a hostname

Give your Droplets an identifying name you will remember them by. Your Droplet name can only contain alphanumeric characters, dashes, and periods.

laravelvue

### Select project

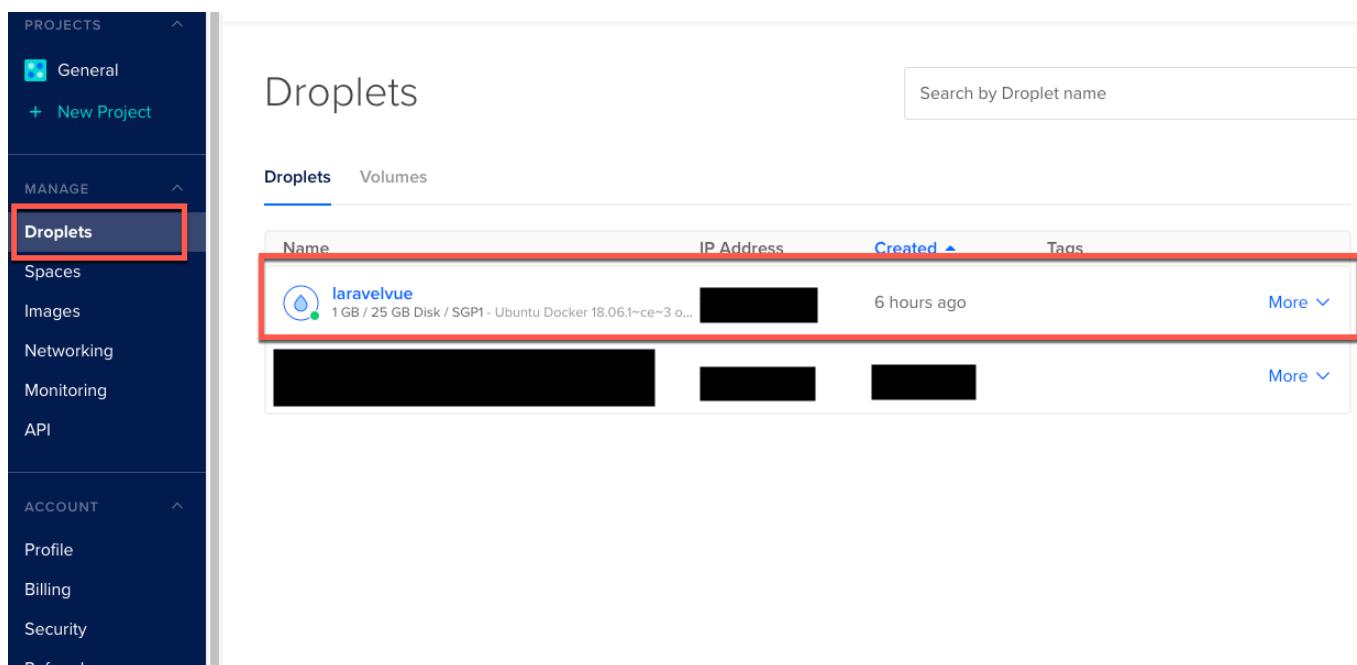
Select an existing project for this Droplet/s to belong to.

Add Tags

General

Create

Jika sudah maka kini kita bisa melihat droplet yang kita miliki di menu "Droplets" seperti ini:



## Siapkan VPS agar bisa diakses via SSH

Karena kita belum bisa mengakses VPS kita melalui terminal di komputer lokal kita, kita buka dulu melalui terminal web based dari dashboard. Pada dashboard digitalocean, pilih menu droplet dan pilih droplet kita lalu klik access console.

## Droplets

Search by Droplet name

Droplets Volumes

Name	IP Address	Created ▾	Tags	More ▾
 laravelvue 1 GB / 25 GB Disk / SGP1 - Ubuntu Docker 18.06.1~ce~3	[REDACTED]	6 hours ago		<a href="#">Add a domain</a> <a href="#">Access console</a> <span style="border: 2px solid red; padding: 2px;">(highlighted)</span> <a href="#">Resize droplet</a> <a href="#">View usage</a> <a href="#">Enable backups</a> <a href="#">Add tags</a> <a href="#">Destroy</a>
	[REDACTED]	[REDACTED]	[REDACTED]	

Setelah itu akan muncul window console, kamu login dengan username **root** dan password adalah akun digitaloceanmu

🔒 DigitalOcean, LLC [US] | [https://cloud.digitalocean.com/droplets/111887076/console?no\\_layout=true](https://cloud.digitalocean.com/droplets/111887076/console?no_layout=true)

Ubuntu 18.04.1 LTS laravelvue tty1

laravelvue login:

PUBLIC IP ADDRESS      GATEWAY:      NETMASK:

Connected (encrypted) to: QEMU (Droplet-111887076)

Setelah itu konfigurasi file sshd\_config untuk ssh agar mengizinkan otentikasi via password. Ketik perintah berikut setelah login:

```
nano /etc/ssh/sshd_config
```

Kemudian replace baris ini

```
# PasswordAuthentication no
```

Menjadi

```
PasswordAuthentication yes
```

Setelah itu reload / restart service ssh dengan perintah ini:

```
sudo service sshd restart
```

Dengan begitu kamu siap untuk login ke vpsmu dari terminal di komputer dengan cara mengetikkan perintah ini:

```
ssh root@IPVPSMU
```

## Mengarahkan custom domain ke VPS

Agar domain yang kita miliki mengarah ke VPS kita ada beberapa langkah yang harus kita lakukan.

1. Set DNS pada domain provider kita agar menggunakan nameserver dari digitalocean. Login ke domain provider yang kamu miliki, dalam hal ini bisa domainesia, godaddy, atau provider lainnya. Lalu pilih domain yang ingin kamu arahkan ke VPSmu dan pilih dns. Pada isian dns masukan 3 name server berikut ini:

```
ns1.digitalocean.com  
ns2.digitalocean.com  
ns3.digitalocean.com
```

Sebagai contoh jika kamu menggunakan domainesia maka langkahnya adalah sebagai berikut:

The screenshot shows the 'My Domains' page of the Domainesia website. On the left, there's a sidebar with 'Services', 'Domains' (which is highlighted with a red box and has a red arrow pointing to it), 'Billing 0', 'Support 0', 'Affiliate', and 'Promotion 3'. The main area shows a table of domains with columns for Domain, Activation, Expiration, and Status. One row is selected, showing 'larashop.id' with activation on 05/09/2018, expiration on 31/08/2019, and status 'Active'. There are also entries for 'vueshop.id'. At the bottom, there are buttons for 'View 10' and 'Action'.

Klik pada sidebar menu "Domains" lalu klik pada nama domain yang akan kita ubah DNS nya.

Setelah itu pilih tab "Nameservers" dan masukan 3 nameserver digitalocean seperti berikut:

Nameservers

Your Current Nameservers:

- Nameserver 1: ns1.digitalocean.com
- Nameserver 2: ns2.digitalocean.com
- Nameserver 3: ns3.digitalocean.com

Change where your domain points to. Please be aware changes can take up to 24 hours to propagate.

Use default nameservers [?](#)

DomaiNesia hosting nameservers [?](#)

Use custom nameservers (enter below) [?](#)

Nameserver 1	ns1.digitalocean.com
Nameserver 2	ns2.digitalocean.com
Nameserver 3	ns3.digitalocean.com
Nameserver 4	

Jangan lupa klik tombol "Change nameservers" di bawah.

## 2. Setting domain pada droplet kita di Digital Ocean.

a. Login ke digitalocean

b. Lalu pilih menu "Droplets"

c. Kemudian pada droplet yang akan kita tambahkan domain baru, pilih "Add domain" seperti ini:

General

+ New Project

MANAGE

Droplets **1**

Spaces

Images

Networking

Monitoring

API

ACCOUNT

Profile

Billing

Security

Droplets

Volumes

Search by Droplet name

Name	IP Address	Created	Tags
laravelvue	[REDACTED]	Created 1 hour ago	
[REDACTED]	[REDACTED]	Created 1 hour ago	

More

Add a domain

Access console

Resize droplet

View usage

Enable backups

Add tags

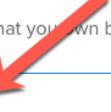
Destroy

d. Ketikan nama domain yang akan ditambahkan di kolom yang disediakan lalu klik "Add domain" seperti di gambar berikut:

Domains Floating IPs Load Balancers Firewalls PTR records

## Add a domain

Enter a domain that you own below and start managing your DNS within your DigitalOcean account.

Enter domain larashop.id  General  Add Domain

Jika sudah maka kamu akan diarahkan ke halaman berikutnya dan terdapat 3 nameserver seperti ini:

NS	larashop.id	directs to ns1.digitalocean.com.	1800	More 
NS	larashop.id	directs to ns2.digitalocean.com.	1800	More 
NS	larashop.id	directs to ns3.digitalocean.com.	1800	More 

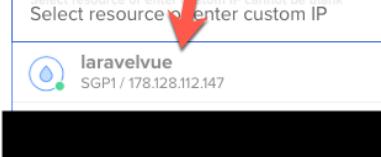
### e. Buat A record baru pada form di bagian atas

Create new record

A AAAA CNAME MX TXT NS SRV CAA    

Use @ to create the record at the root of the domain or enter a hostname to create it elsewhere. A records are for IPv4 addresses only and tell a request where your domain should direct to.

HOSTNAME   WILL DIRECT TO   TTL (SECONDS)   Create Record 

larashop.id 

DNS records

### f. Setelah itu buat juga CNAME record baru

Create new record

A AAAA  MX TXT NS SRV CAA 

CNAME records act as an alias by mapping a hostname to another hostname.

HOSTNAME   IS AN ALIAS OF   TTL (SECONDS)   Create Record 

\*.larashop.id  

Okay, jika sudah dilakukan semua maka kini domain dari domain provider kita sudah mengarah ke VPS kita di DigitalOcean.

DNS records pada droplet kita akan terlihat seperti ini:

#### DNS records

Type	Hostname	Value	TTL (seconds)	
CNAME	*.larashop.id	is an alias of larashop.id.	43200	<a href="#">More ▾</a>
A	larashop.id	directs to [REDACTED]	3600	<a href="#">More ▾</a>
NS	larashop.id	directs to ns1.digitalocean.com.	1800	<a href="#">More ▾</a>
NS	larashop.id	directs to ns2.digitalocean.com.	1800	<a href="#">More ▾</a>
NS	larashop.id	directs to ns3.digitalocean.com.	1800	<a href="#">More ▾</a>

Selanjutnya tinggal atur di webserver kita agar mendengarkan (listening) ke domain yang baru saja kita tambahkan. Kita akan membahasnya pada salah satu langkah deployment ke VPS berikutnya.

#### Siapkan laradock dan larashop di VPS

1. Masuk ke VPS kita menggunakan ssh

```
ssh root@IP_VPS_KITA
```

Lalu masukan password yang diminta.

2. Buat folder baru dengan nama **laravel-projects**, penamanaan ini untuk menyamakan dengan apa yang kita lakukan di lokal. Tentu kamu bisa memberikan nama folder apa saja.

```
mkdir laravel-projects
```

3. Masuk ke folder laravel-project dan clone repository laradock kita.

```
cd laravel-projects
```

```
git clone URL_REPOSITORY_LARADOCK_KITA_DI_GITLAB_BITBUCKET_ATAU_GITHUB
laradock
```

Jika sudah selesai maka kini di folder **laravel-projects** akan terdapat folder **laradock**

```
// FOLDER KITA
+ laravel-projects
```

Sebelum melangkah ke tahap selanjutnya, kita buat dulu file .env di laradock karena file tersebut tidak masuk dalam git repository. Caranya, masuk ke direktori laradock:

```
cd laradock
```

Lalu ketikan perintah ini

```
cp env-example .env
```

Ubah beberapa konfigurasi di dalam file .env tadi, yang pertama adalah konfigurasi MySQL.

```
MYSQL_VERSION=5.7
MYSQL_DATABASE=larashop
MYSQL_USER=larashop_app
MYSQL_PASSWORD=RahasiaAplikasiBanget
MYSQL_PASSWORD_ROOT_PASSWORD=RahasiaRoot
```

#### 4. Jalankan service mysql di background

```
docker-compose up -d mysql
```

Menjalankan webserver nginx

1. Masuk ke folder **laravel-projects/laradock**
2. Buka file **laravel-projects/laradock/nginx/sites/default.conf** Hapus **default\_server** pada baris-baris kode ini:

```
listen 80 default_server;
listen [::]:80 default_server ipv6only=on;
```

Tambahkan domain kita setelah **server\_name** seperti ini:

```
server_name larashop.id www.larashop.id
```

Arahkan root ke folder larashop kita dengan konfigurasi berikut

```
root /var/www/larashop/public;
```

Sehingga file **default.conf** akan memuat kode-kode berikut:

```
listen 80;
listen [::]:80 ipv6only=on;

server_name larashop.id;
root /var/www/larashop/public;
index index.php index.html index.htm;
```

3. Rebuild container nginx Jika nginx sebelumnya pernah dijalankan, stop dulu dengan perintah ini

```
docker-compose down
```

Lalu jalankan perintah berikut:

```
docker-compose build nginx
```

4. Jalankan container nginx di background

```
docker-compose up -d nginx
```

Apabila saat menjalankan perintah **docker-compose up nginx** untuk pertama kali mengalami kendala silahkan lihat subbagian di bagian akhir bab deployment ini.

Clone source code larashop

1. Kemudian clone repository aplikasi larashop kita, masih di folder **laravel-projects**

```
git clone URL_REPOSITORY_LARASHOP_KITA_DI_GITLAB_BITBUCKET_ATAU_GITHUB
larashop
```

Sehingga kini kita memiliki struktur folder seperti ini:

```
// FOLDER KITA
+ laravel-projects
    + laradock
    + larashop
```

2. Selanjutnya mari kita jalankan composer install di larashop kita melalui container workspace

Pindah ke folder laradock

```
cd laradock
```

Lalu masuk ke workspace

```
docker-compose exec workspace bash
```

Pada container itu masuk ke folder **larashop**

```
cd larashop
```

Lalu jalankan composer install

```
composer install
```

3. Setelah itu kita perlu ubah permission untuk folder storage dan isinya di folder larashop

```
chmod -R 777 storage
```

4. Ubah konfigurasi aplikasi laravel (larashop) kita. Masih di dalam container workspace, folder larashop.  
Buka file .env

```
nano .env
```

Lalu sesuaikan beberapa konfigurasi terutama koneksi database mysql kita agar sesuai dengan apa yang telah kita isikan pada poin 3

```
# Ganti dengan domain Anda
APP_ENV=production
APP_URL=http://domainkamu.com

DB_USERNAME=larashop_app
DB_PASSWORD=RahasiaAplikasiBanget
```

Setelah itu simpan dengan menekan tombol CTRL + x, kemudian tekan tombol "y" untuk overwrite dan ENTER dengan nama yang sama ".env"

5. Setelah itu jalankan migration dan seed Masih di workspace kita folder laradock jalankan

```
php artisan migrate
```

Kemudian,

```
php artisan db:seed
```

6. Buka web browser dan ketik alamat IP VPS kita <http://domainkamu>, maka aplikasi larashop sudah berjalan

Tips saat create droplet tidak bisa access console

Saat pertama kali membuat droplet kamu ingin melakukan setting SSH, akan tetapi sayangnya kamu gagal ketika login menggunakan username **root** dan password akun domainesiamu. Sayangnya ternyata kamu mendapat error "invalid credentials".

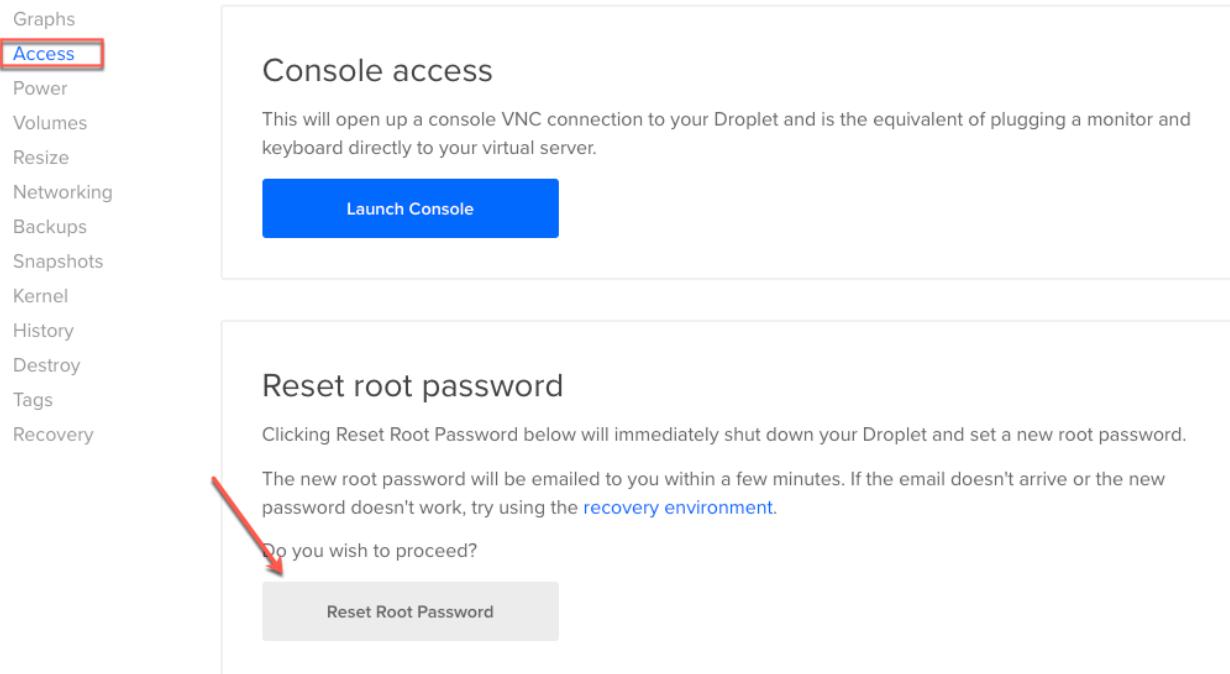
Solusinya adalah dengan melakukan reset password root pada droplet tersebut. Berikut langkah-langkahnya:

1. Pilih / klik pada nama droplet yang ingin kita reset root passwordnya

Droplets

Name	IP Address	Created	Tags
laravelvue 1 GB / 25 GB Disk / SGP1 - Ubuntu Docker 18.06.1~ce~3 o...	[REDACTED]	Yesterday	More ▾
[REDACTED]			More ▾

2. Pilih menu "Access" lalu klik tombol "Reset Root Password"



Jika sudah silahkan cek email yang kamu gunakan untuk login ke digitalocean. Password baru akan dikirimkan ke email tersebut.

Tips jika gagal menjalankan "docker-compose up nginx"

Ketika pertama kali menjalankan perintah "docker-compose up nginx" apabila kita mengalami error saat instalasi Imagick, kita harus sedikit mengubah Dockerfile di salah satu folder laradock, yaitu [laradock/php-fpm/Dockerfile](#). Buka file tersebut di VPS dengan perintah ini:

```
nano `PATHMU/laravel-projects/laradock/php-fpm/Dockerfile`
```

Lalu cari baris ini:

```
apt-get install -y libmagickwand-dev imagemagick && \
```

Dan replace dengan kode ini:

```
apt-get install -y libmagickwand-dev --allow-unauthenticated imagemagick && \  
\
```

Setelah itu jalankan lagi [docker-compose up nginx](#)

**Kesimpulan**

Aplikasi yang kita bangun dengan Laravel dapat kita deploy ke hosting ataupun ke VPS. Kita telah belajar bagaimana mendeploy baik ke hosting maupun ke VPS.

Saat mendeploy ke shared hosting kita cukup mentransfer file-file di lokal ke shared hosting menggunakan file manager. Kemudian mengimport database dari lokal ke database di shared hosting melalui PHPMyadmin. Kemudian kita juga perlu menjalankan perintah `storage:link` dengan cronjob agar file-file bisa diakses melalui internet.

Mendeploy ke VPS agak berbeda dimana kita terlebih dahulu membuat git repository untuk laradock dan projek laravel kita (larashop) kemudian kita pull dari server VPS kita dan kita atur beberapa konfigurasi agar bisa menggunakan domain dengan https. Dan tentunya sebelum melakukan itu kita harus mengarahkan terlebih dahulu domain dari provider ke VPS.

Semuanya telah dibahas dibab ini, selamat dengan demikian aplikasi yang kamu buat bisa diakses melalui internet.

The advertisement features a large, stylized blue server tower in the center, with several smaller cloud icons floating around it. A white blimp with the 'Domainesia' logo is positioned above the server. The background is a light blue gradient.

**// HOSTING //**

Incredibly fast and reliable hosting infrastructure with custom built features to accelerate your website. Ranked as No. 1 web hosting provider in Indonesia\*.

PT Deneva

YAP Square No. C-5,  
Jl C. Simanjuntak No.2 Yogyakarta 55223,  
Daerah Istimewa Yogyakarta, Indonesia  
info@domainesia.com | (0274) 545653  
www.domainesia.com

\*according to rankingshosting.com as of September 2018

Khusus pembaca buku ini, kami menyediakan diskon khusus sebesar 30% dari harga reguler untuk pembelian produk Hosting di <https://domainesia.com>, dengan menggunakan kode kupon **FullstackHosting**

Catatan: masukkan kode kupon di atas setiap transaksi. Kode ini dapat digunakan secara berulang.



## // VPS //

Instantly spin Linux virtual server instance  
and get root access within seconds.  
Powered by enterprise-grade server with  
Native SSD storage to deliver  
blazing fast performance.



PT Deneva

YAP Square No. C-5,  
Jl C. Simanjuntak No.2 Yogyakarta 55223,  
Daerah Istimewa Yogyakarta, Indonesia  
[info@domainesia.com](mailto:info@domainesia.com) | (0274) 545653  
[www.domainesia.com](http://www.domainesia.com)

Khusus pembaca buku ini, kami menyediakan diskon khusus sebesar 30% dari harga reguler untuk pembelian produk VPS di <https://domainesia.com>, dengan menggunakan kode kupon **FullstackVPS**

Catatan: masukkan kode kupon di atas setiap transaksi. Kode ini dapat digunakan secara berulang.