

MENYELAMI FRAMEWORK

Laravel^{5.2}

panduan komprehensif dan aplikatif untuk
menguasai framework laravel

RAHMAT AWALUDIN

Menyelami Framework Laravel

Panduan komprehensif dan aplikatif untuk menguasai framework Laravel.

Rahmat Awaludin

This book is for sale at <http://leanpub.com/bukularavel>

This version was published on 2016-04-25



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2014 - 2016 Rahmat Awaludin

Tweet This Book!

Please help Rahmat Awaludin by spreading the word about this book on [Twitter](#)!

The suggested hashtag for this book is [#bukularavel](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

<https://twitter.com/search?q=#bukularavel>

Also By Rahmat Awaludin

[Seminggu Belajar Laravel](#)

Untuk istriku tercinta, Irna Rahayu dan jagoan kecilku, Shidqi Abdullah Mubarak.

Contents

Koreksi	1
Saran dan Masukan	2
Sekilas	3
Metode Penulisan	3
Membaca sample source code	3
Apa buku ini sudah selesai?	5
Kalau ada materi yang kurang gimana?	5
Lisensi	5
Reseller	5
Tawaran Kerjaan	6
Konsep Dasar	7
JSON	7
PHP5 Anonymous Function/Lambda & Closure	9
PHP5 Autoloader	14
PHP5 Abstract dan Interfaces	17
PHP5 Traits	37
PHP5 Namespace	42
PHP5 Magic Method	50
PHP5 Reflection	56
Composer	72
Konfigurasi, untuk Project Skala Besar	85
Kebutuhan Sistem	85
Instalasi	89
Homestead	96
Artisan CLI	115
Environment	118
Arsitektur Laravel	130
Struktur Aplikasi	130
Service Providers	134
Service Container	145

CONTENTS

Facades	160
Contracts	166
Alur Request	179
Routing, Kendalikan Alur Aplikasi	184
Kenapa menggunakan routing?	184
Konfigurasi Routing	185
Memahami HTTP Verb	185
Menggunakan Beberapa Verb untuk Satu Route	190
Memahami Parameter dalam Routing	198
Regex Parameter	200
Optional Parameter	203
Memahami Named Routes	205
Memahami Route Groups	208
Subdomain Routing	210
Whats next?	215
Mengakses Database	216
Konfigurasi	216
Database Migration & Schema Builder	222
Basic CRUD Query	239
Database Seeding	243
Query Builder	250
Transaksi	282
Logging	294
Database Caching	296
Menggunakan Multiple Database	299
Konfigurasi Read/Write	302
Model dari MVC	306
Memahami ORM	306
Eloquent, ORM Powerfull dari Laravel	310
Konfigurasi	311
Mass Assignment	323
CRUD Dasar	327
Output Model ke Array / JSON	334
Query dalam Eloquent	341
Soft Delete	349
Query Scope	353
Global Scope	360
Menggunakan Accessors dan Mutators	385
Date Mutator & Carbon	394
Attribute Casting	406

CONTENTS

Model Event & Observer	412
Route Model Binding	424
Relationship dalam Eloquent	427
Memahami Jenis Relationship	427
One To One	428
One To Many	435
Many To Many	441
Has Many Through	449
Polymorphic Relations	453
Self Referencing Relationship	460
Many To Many Polymorphic Relations	463
Custom Key untuk Relationship	468
Eager Loading	470
Collection dengan Eloquent	475
Class Collection	475
Membuat Collection	475
Method pada Collection	477
View dari MVC	484
Penggunaan View	484
Konfigurasi	485
Passing Variable	486
View Composer	491
Blade, Templating Engine dari Laravel	493
Blade Layout	496
Blade Control Structure	500
Elixir	502
Form	507
Form Model Binding	514
Controller dari MVC	519
Penggunaan Controller	519
Konfigurasi	522
Implicit Controller	523
RESTful Resource Controller	525
Dependency Injection	543
Route Caching	545
Whats Next?	548
Request & Response	549
Memahami Request di Laravel	549
Menerima Input	552

CONTENTS

Old Input	561
Menerima Upload File	563
Bekerja dengan Cookie	565
Mengambil info dari Request	569
Memahami Response di Laravel	570
Membuat Response Redirect	572
Membuat Response JSON	573
Membuat Response Download File	576
Menggunakan Response Macro	577
Middleware	580
Membuat Middleware	581
After Middleware	585
Register	589
Terminable	591
Middleware Parameters	594
Route Middleware	599
Controller Middleware	607
Middleware Group (update Laravel 5.2)	608
Authentikasi & Authorisasi, dari Login hingga Hak Akses	611
Memahami Alur Authentikasi Laravel	611
Proteksi route dengan Auth Middleware	618
Menambah field baru ke table user	619
Menggunakan Username untuk login	623
Kustomisasi Login	624
Password Reminder & Reset	626
Menggunakan Throttles Login	632
Auto Generate Authentikasi (Update Laravel 5.2)	633
Menambahkan fitur Suspend	633
Mengakses User yang telah Login	636
Social Authentication dengan Socialite	637
Membangun Hak Akses User (RBAC + ACL)	652
Menggunakan Ability & Policy	661
JWT (JSON Web Token) Authentication	690
Validasi Data	700
Memahami Validasi di Laravel	700
Menampilkan pesan error	705
Memodifikasi pesan error	710
Validasi Kondisional	713
Validasi Array (update Laravel 5.2)	718
Validasi untuk Ajax / API	720

CONTENTS

Menggunakan berbagai rule validasi di Laravel	723
Membuat Custom Rule	744
Validasi dengan Form Request	746
Case Study: Membangun CMS Toko Online	751
Penjelasan project	751
Menjalankan sample code	752
Persiapan Project	752
Authentikasi & Level User	753
Mengatur Guard dan Middleware	758
Pengaturan Asset	759
Manajemen Produk	762
Katalog	816
Berbagai tipe Cart dan Checkout	842
Cart	844
Checkout	883
Manajemen Order	985
Cek Order	1006
Whats Next?	1011

Koreksi

Setiap manusia pasti tidak pernah luput dari kesalahan, begitupun dengan penulis. Jika Anda menemukan kesalahan dalam penulisan buku ini, silahkan kirim ke rahmat.awaludin@gmail.com

Setiap kesalahan yang ditemukan akan segera diperbaiki pada buku, Anda cukup mendownload ulang buku ini dari Leanpub.

Saran dan Masukan

Saya sangat mengharapkan saran bagi perbaikan penulisan buku ini kedepan. Jika Anda punya saran yang ingin disampaikan, kirim email ke rahmat.awaludin@gmail.com¹ atau via twitter di [@rahmatawaludin](https://twitter.com/rahmatawaludin)².

¹<mailto:rahmat.awaludin@gmail.com>

²<https://twitter.com/rahmatawaludin>

Sekilas

Perkenalkan, saya Rahmat Awaludin. Saya seorang senior web developer dari Bandung, Indonesia. Sudah sejak tahun 2011 saya berada di dunia web development, lebih tepatnya ketika saya masih berstatus sebagai mahasiswa di Universitas Padjadjaran. Bisa cek [profile LinkedIn³](#) saya untuk kenal lebih jauh dimana saya bekerja.

Buku ini saya tulis untuk programmer yang sudah familiar dengan framework Laravel, yang ingin menambah pemahamannya tentang framework ini. Setelah membaca buku ini harapan saya pembaca dapat memahami core Laravel dengan lebih baik sehingga mampu membuat aplikasi Laravel yang lebih solid.

Metode Penulisan

Pembahasan materi dibuku ini akan dibuat secara aplikatif. Jadi, tiap materi akan saya berikan contoh faktual cara mengaplikasikannya. Agar materi di buku ini tidak hanya menjadi pengetahuan, tapi bisa Anda praktikan.

Saya membuat aplikasi dalam buku ini dengan OS X. Sebagian syntax terminal, saya buat dua versi (*nix dan windows). Namun, kebanyakan syntax yang saya tulis adalah syntax pada OS *nix.

Saya yakin tidak akan semua materi Laravel akan mampu saya bahas di buku ini. Karena itu, saya akan membuat buku Laravel ini berseri, misalnya Seri API, Seri Design Pattern, Seri Testing, dll. Silahkan cek Seri lain yang sedang saya tulis di <https://leanpub.com/u/rahmatawaludin⁴>. Jika punya ide seri yang lain, silahkan kirim email atau posting di [fanpage⁵](#)

Membaca sample source code

Dalam buku ini, terdapat beberapa cara penulisan source code. Berikut contohnya:

³www.linkedin.com/in/rahmatawaludin

⁴<https://leanpub.com/u/rahmatawaludin>

⁵<https://www.facebook.com/bukularavel>

app/config.php

```
....  
'aliases' = [  
    ....  
    'App'————=> Illuminate\Support\Facades\App::class,  
,  
'providers' => [  
    ....  
    Collective\Html\HtmlServiceProvider::class,  
,  
....
```

app/config.php menunjukan alamat file pada project.

Isian menandakan ada banyak baris code pada bagian tersebut. Kode tersebut sengaja tidak ditulis karena bukan bagian dari pembahasan.

Code yang ditulis tebal menandakan code yang baru ditambahkan.

Code yang bergaris menandakan bahwa kode tersebut dihapus.

Jika baris dari code tidak disebutkan, artinya Anda harus mencari baris tersebut pada file yang dimaksud. Pada contoh diatas, pasti akan ditemukan baris dengan tulisan aliases dan providers pada file config/app.php.

Jika baris yang bergaris dan tebal di letakkan berdampingan, biasanya Anda diminta untuk mengganti baris tersebut. Misalnya seperti berikut:

resources/views/layouts/app.blade.php

```
....  
<title>Laravel</title>  
<title>Aplikasi Ku</title>  
....
```

Pada code diatas, berarti tag <title> akan diubah menjadi tag <title> dengan isian Aplikasi Ku.

Terkadang, Anda juga akan diminta untuk **menjalankan** perintah. Misalnya seperti ini:

```
php artisan serve
```

Maksudnya adalah mengetikkan perintah tersebut di terminal / CMD. Perintah biasanya dijalankan dari folder project. Pada code diatas, artinya Anda harus menjalankan php artisan serve dari folder project. Namun, jika project belum dibuat, biasanya perintah tersebut dapat dijalankan dimanapun di terminal.

Apa buku ini sudah selesai?

Sudah.

Kalau ada materi yang kurang gimana?

Boleh request ke saya, langsung kontak saja. Kalau memang penting, bisa saya tambahkan sebagai materi bonus

Lisensi

Lisensi buku ini boleh digunakan oleh satu orang pembaca. Artinya, jika satu lisensi digunakan oleh dua pembaca atau lebih, maka saya kategorikan membajak. Bagaimana pun caranya. Jika Anda sudah terlanjur membajak buku ini, silahkan membayar dengan transfer ke BNI 0186355188 an Rahmat Awaludin (harga sesuai di <http://leanpub.com/bukulara>). Kemudian, konfirmasikan ke email rahmat.awaludin@gmail.com.

BTW, saya juga memberikan diskon jika membeli banyak lisensi dari buku ini.

- Untuk 2-5 lisensi, diskon 20rb.
- Untuk 6-10 lisensi, diskon 50rb.
- Untuk lebih dari 10 lisensi, diskon 70rb.

Yap, saya memang tidak tahu siapa saja yang telah membajak buku ini. Tapi, Allah tahu koq. Makanya, jika Anda tidak membayar lisensi buku ini di dunia, saya akan menagih pembayarannya di akhirat. *Deal ya? :)*

Reseller

Jika Anda telah memiliki buku ini, Anda juga berpeluang untuk menjadi Reseller. Silahkan infokan buku ini ke rekan Anda. Ketika mereka telah order ke saya, kontak saya dan sampaikan nama teman Anda. Selanjutnya, 50rb akan saya transfer ke rekening Anda.

Reseller hanya berlaku untuk pembelian dengan harga normal.

Tawaran Kerjaan

Sebagai freelance developer, saya terbuka untuk tawaran konsultasi, project, dan pembicara (seminar/training). Jika Anda tertarik mengundang saya kontak saya via:

- Email : rahmat.awaludin@gmail.com⁶
- FB : www.facebook.com/rahmat.awaludin⁷
- Twitter : <https://twitter.com/rahmatawaludin>⁸
- linkedin : www.linkedin.com/in/rahmatawaludin⁹
- Phone : [+62878 2225 0272](tel:+6287822250272)¹⁰

⁶<mailto:rahmat.awaludin@gmail.com>

⁷www.facebook.com/rahmat.awaludin

⁸<https://twitter.com/rahmatawaludin>

⁹www.linkedin.com/in/rahmatawaludin

¹⁰<tel:+6287822250272>

Konsep Dasar

Oke *bro*, saya yakin udah ngga sabar mau belajar Laravel. Udah tau kan, katanya Laravel itu framework PHP yang lagi **booming¹¹**? Tapi, tenang dulu, saya ingin agar pemahaman kita tentang framework ini lengkap banget. Bab ini, sengaja saya selipkan untuk membahas topik-topik “dasar” (sebenarnya ngga dasar banget sih) yang perlu dipahami sebelum mempelajari framework Laravel. Supaya nanti tidak tenggelam dalam lautan kebingungan ketika pembahasan tentang Laravel sudah sangaaaaat dalam.



Peringatan

Sebagian materi di bab ini bukan untuk yang baru mengenal PHP. Jika mengalami gejala pusing dan mual-mual setelah membaca materi di bab ini, silahkan pelajari kembali dasar-dasar PHP Procedural dan OOP.

JSON

JSON (dilafalkan “Jason”), singkatan dari JavaScript Object Notation (bahasa Indonesia: notasi objek JavaScript), adalah suatu format ringkas pertukaran data komputer. Itu. Ngerti kan?

Oke, oke, becanda. :D

Saya jelaskan lebih detail ya, JSON itu format data yang digunakan untuk bertukar data antar bahasa pemrograman. Awalnya JSON ini hanya digunakan di Javascript. Tapi, seiring waktu, JSON juga digunakan oleh bahasa pemrograman yang lain.

Syntax dasar JSON kayak gini:

¹¹ <http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/>

Contoh JSON

```
{  
    key1 : value1,  
    key2 : {  
        key3 : value3,  
        key4 : value4  
    }  
}
```

JSON mendukung berbagai format data, yaitu :

- Number : berisi bilangan bulat atau desimal.
- String : teks yang diapit tanda petik.
- Boolean : Isian benar (`true`) atau salah (`false`).
- Array : Data terurut yang diapit [] dan dibatasi koma. Larik bisa berisi gabungan tipe data yang lain.
- Object : Data tidak terurut yang diapit { } dan dibatasi koma. Setiap elemen object berisi `key` dan `value` dibatasi :.
- null : nilai kosong, diisi dengan keyword `null`.

Jika digabungkan, semua tipe data diatas jadi seperti ini:

Contoh JSON

```
{  
    "nomor_antrian" : 20,  
    "nama" : "Firman",  
    "menikah" : false,  
    "hobi" : ["memancing", "berenang", "bersepeda"],  
    "alamat" : {  
        "jalan" : "Pakuan No. 12",  
        "kota" : "Bandung",  
        "provinsi" : "Jawa Barat",  
        "kode_pos" : "43655"  
    },  
    "telpon" : {  
        { "jenis" : "rumah", "nomor" : "022-345352" },  
        { "jenis" : "mobile", "nomor" : "0878-23422321" }  
    },  
    "istri" : null  
}
```

JSON akan sangat sering kita temui dalam pengembangan web dengan Laravel. Oleh karena itu, pastikan paham tentang JSON ini. Kalau belum paham, ngga apa-apa juga sih, ntar juga paham kalau sudah praktek. Sip.

PHP5 Anonymous Function/Lambda & Closure

Sebagaimana seorang hacker yang tidak ingin diketahui namanya, begitu pula dengan Anonymous Function/Lambda ini. Fungsi ini dapat kita gunakan tanpa harus menulis nama fungsinya. Jadi, kalau dari segi kesolehan, fungsi ini termasuk tipe tawadhu (rendah hati) karena ia banyak bekerja tanpa ingin diketahui namanya. Hadeuuuhhh.. :v

Perhatikan contoh syntax berikut:

~/Code/lambda-anonymous-closure-sip/index.php

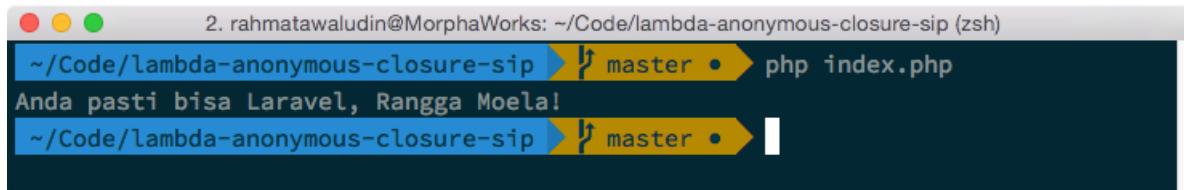
```
<?php
function namaAnda() {
    return "Rangga Moela";
}

function bisa($nama) {
    echo "Anda pasti bisa Laravel, $nama!\n";
}

bisa(namaAnda());
```

Yang syntax ini lakukan adalah membuat dua buat fungsi yaitu `namaAnda()` yang menampilkan nama yang kita inginkan dan `bisa()` yang menampilkan tulisan motivasi mempelajari framework Laravel. Fungsi `bisa()` menerima parameter sebuah string yang kita beri nama `$nama`.

Pada baris terakhir, kita memanggil fungsi `bisa()` dengan parameter output dari fungsi `namaAnda()`. Tentunya ini bisa dilakukan, karena output dari fungsi `namaAnda()` adalah string. Sehingga output nya menjadi:



The screenshot shows a terminal window with the following content:

```
2. rahmatawaludin@MorphaWorks: ~/Code/lambda-anonymous-closure-sip (zsh)
~/Code/lambda-anonymous-closure-sip ➜ master • php index.php
Anda pasti bisa Laravel, Rangga Moela!
~/Code/lambda-anonymous-closure-sip ➜ master •
```

Menggunakan fungsi sebagai paramater

Jika membuat fungsi biasa harus pakai nama misal:

Contoh Fungsi biasa

```
function namaAnda() {  
    return "Rangga Moela";  
}
```

maka, Anonymous Function ditulis dengan menghilangkan nama fungsinya:

Contoh Anonymous Function

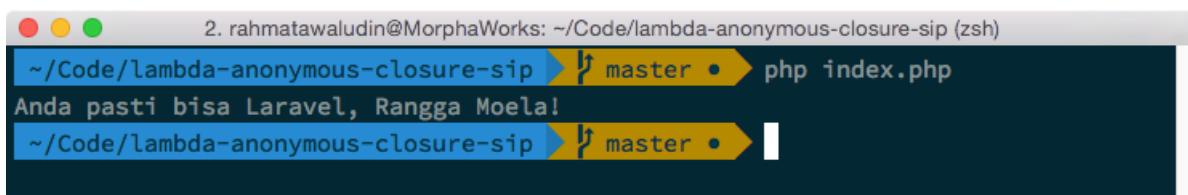
```
function() {  
    return "Rangga Moela";  
}
```

Untuk menggunakan anonymous function ini, kita memassing fungsi tersebut ke sebuah variable. Berikut ini contoh kalau kita mengubah fungsi `namaAnda()` menjadi anonymous function dan menyimpannya ke variable `$nama`:

`~/Code/lambda-anonymous-closure-sip/index.php`

```
<?php  
$nama = function() {  
    return "Rangga Moela";  
};  
  
function bisa($nama) {  
    echo "Anda pasti bisa Laravel, $nama!\n";  
}  
  
bisa($nama());
```

Coba jalankan lagi, hasilnya tetap sama kan?



```
2. rahmatawaludin@MorphaWorks: ~/Code/lambda-anonymous-closure-sip (zsh)  
~/Code/lambda-anonymous-closure-sip ➜ master • ➜ php index.php  
Anda pasti bisa Laravel, Rangga Moela!  
~/Code/lambda-anonymous-closure-sip ➜ master • ➜
```

Anonymous function sedang beraksi

Kalau udah paham, sekarang kita pelajari tentang *Closure*.

Closure mirip-mirip dengan lambda/anonymous function, bedanya closure ini *dapat menerima parameter dan mengakses variable dari luar fungsi itu*. Misal kayak gini:

~/Code/lambda-anonymous-closure-sip/closure.php

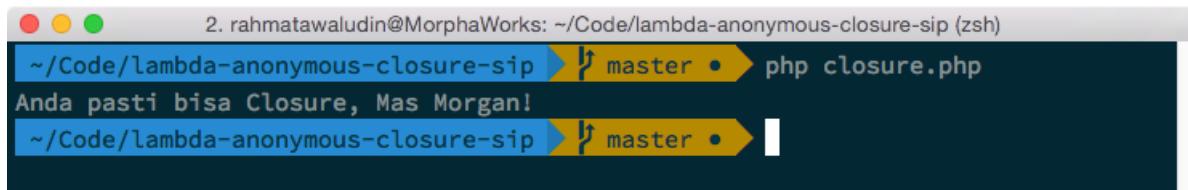
```
<?php
$title = "Mas";
$nama = "Morgan";

function namaAnda($title, $nama) {
    return $title . ' ' . $nama;
}

function bisa($nama) {
    echo "Anda pasti bisa Closure, $nama!\n";
}

bisa(namaAnda($title, $nama));
```

Pada file ini kita membuat fungsi `namaAnda()` yang menerima input `title` dan `nama` kemudian menampilkan output `title` dan `nama`. Kita juga membuat fungsi `bisa()` yang menerima input string dan menampilkan output kalimat motivasi. Di baris terakhir, kita memanggil fungsi `bisa()` dengan parameter output dari fungsi `namaAnda()`. Outputnya seperti ini:



```
2. rahmatawaludin@MorphaWorks: ~/Code/lambda-anonymous-closure-sip (zsh)
~/Code/lambda-anonymous-closure-sip ✘ master • php closure.php
Anda pasti bisa Closure, Mas Morgan!
~/Code/lambda-anonymous-closure-sip ✘ master •
```

Fungsi biasa dengan parameter

Jika kita merubah fungsi `namaAnda()` menjadi closure, maka syntaxnya akan berubah menjadi :

~/Code/lambda-anonymous-closure-sip/closure.php

```
<?php
$title = "Mas";
$nama = "Morgan";

function bisa($nama) {
    echo "Anda pasti bisa Closure, $nama!\n";
}

$nama = function() use ($title, $nama) {
```

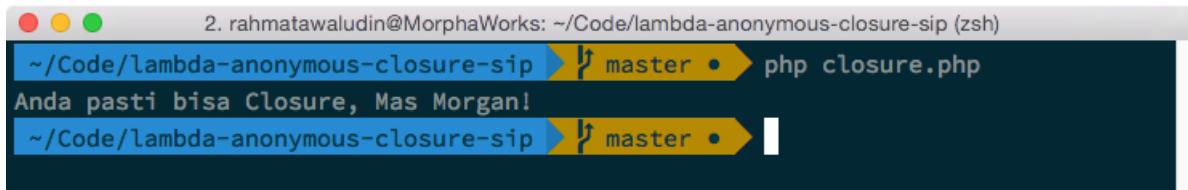
```

    return $title . ' ' . $nama;
};

bisa($nama());

```

Terlihat disini, **untuk menggunakan variable diluar closure** (dalam contoh ini variable \$title dan \$nama), **kita menggunakan keyword use**. Outputnya akan tetap sama:



Menggunakan closure

Closure ini biasanya digunakan pada fungsi yang menggunakan *callback* sebagai parameternya.

Apaan tuh callback?

Callback adalah parameter berupa fungsi.

Contohnya seperti fungsi [array_walk¹²](#) (bawaan PHP). Fungsi ini akan mengiterasi (mirip foreach) elemen dari suatu array. Parameter pertama dari fungsi ini adalah array dan parameter keduanya harus berupa callback. Nah, callback inilah closure. Perhatikan syntax berikut (boleh sambil dicoba):

[~/Code/lambda-anonymous-closure-sip/closure-as-callback.php](#)

```

<?php
$minimal = 75;
$dataNilai = [
    ["nama"=>"Rangga", "nilai"=>90],
    ["nama"=>"Bisma", "nilai"=>80],
    ["nama"=>"Dicky", "nilai"=>40],
    ["nama"=>"Morgan", "nilai"=>75],
];

array_walk($dataNilai, function($siswa) use ($minimal) {
    echo "Siswa : " . $siswa['nama'] . "\n";
    echo "Nilai : " . $siswa['nilai'] . "\n";
}

```

¹²<http://php.net/manual/en/function.array-walk.php>

```
echo "Status : ";
if ($siswa['nilai'] >= $minimal) {
    echo "Lulus\n\n";
} else {
    echo "Tidak Lulus\n\n";
}
});
```

Syntax ini akan menggunakan fungsi `array_walk` untuk melooping semua element dari array `$dataNilai`. Pada parameter kedua di fungsi `array_walk` kita menggunakan closure untuk memproses setiap elemen array agar menampilkan detail nama, nilai dan status kelulusan siswa berdasarkan variabel `$minimal` yang kita buat sebelumnya. Berikut ini output dari syntax berikut:

```
2. rahmatawaludin@MorphaWorks: ~/Code/lambda-anonymous-closure-sip (zsh)
~/Code/lambda-anonymous-closure-sip master • php closure-as-callback.php
Siswa : Rangga
Nilai : 90
Status : Lulus

Siswa : Bisma
Nilai : 80
Status : Lulus

Siswa : Dicky
Nilai : 40
Status : Tidak Lulus

Siswa : Morgan
Nilai : 75
Status : Lulus
```

Closure sebagai callback

Karena, fitur Lambda dan Closure akan sering kita gunakan pada pembahasan di bab-bab selanjutnya. Semoga penjelasan singkat ini dapat memberikan gambaran tentang fitur ini.



Source code dari latihan ini bisa didapat di
<https://github.com/rahmatawaludin/lambda-anonymous-closure-sip>¹³

¹³<https://github.com/rahmatawaludin/lambda-anonymous-closure-sip/commits/master>

PHP5 Autoloader

Sebagai seorang veteran di pemrograman PHP (*ehm*), tentu sudah sangat familiar dengan syntax `include` untuk memasukkan syntax dari file lain ke file yang sedang aktif. Biasanya, syntax ini digunakan jika kita hendak menggunakan class yang berada pada file yang lain. Perhatikan contoh syntax berikut:

~/Code/autoload-oh-autoloader/Printer.php

```
<?php
class Printer {
    public function cetakBuku($buku) {
        echo "Class " . __CLASS__ . " : ";
        echo "Mencetak buku $buku\n";
        return "Buku $buku";
    }
}
```

~/Code/autoload-oh-autoloader/Kurir.php

```
<?php
class Kurir {
    public function kirim($file, $tujuan) {
        echo "Class " . __CLASS__ . " : ";
        echo "Mengirim $file ke $tujuan\n";
    }
}
```

~/Code/autoload-oh-autoloader/index.php

```
<?php
include 'Printer.php';
$printer = new Printer();
$buku = $printer->cetakBuku('Menyelami Framework Laravel');

include 'Kurir.php';
$kurir = new Kurir();
$kurir->kirim($buku, 'Bandung');
```

Pada syntax ini, terlihat kita memiliki dua buah class yaitu `Printer` dan `Kurir`.

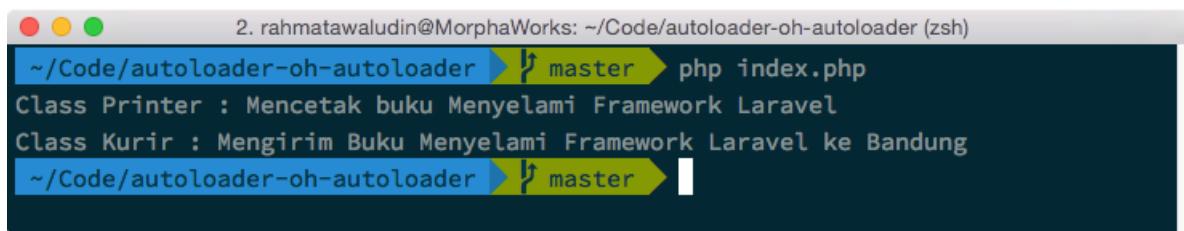
Pada Class Printer, terdapat method `cetakBuku()` yang berfungsi menampilkan tulisan "Mencetak Buku" dengan nama buku yang menjadi parameter. Ketika kita mengambil method ini, kita juga memanggil nama Class yang aktif dengan syntax `echo "Class ". __CLASS__ . " : ".`

Pada Class Kurir, terdapat method `kirim()` yang berfungsi menampilkan tulisan "Mengirim \$file ke \$tujuan" dengan `$file` dan `$tujuan` merupakan parameter dari method ini. Kita juga menampilkan nama Class yang aktif ketika memanggil method ini.

Perhatikan file `index.php`.

Di baris ke-2 dan ke-6 kita menggunakan syntax `include` untuk memasukan file `Printer.php` dan `Kurir.php`.

Ini merupakan cara lama agar kita dapat membuat object dari class `Printer` dan `Kurir` yang berada di kedua file tersebut. Syntax selanjutnya adalah syntax biasa untuk membuat object `Printer` dan `Kurir` kemudian memanggil method pada masing-masing object tersebut. Output dari syntax ini seperti ini:



```
2. rahmatawaludin@MorphaWorks: ~/Code/autoload-oh-autoloader (zsh)
~/Code/autoload-oh-autoloader > master > php index.php
Class Printer : Mencetak buku Menyelami Framework Laravel
Class Kurir : Mengirim Buku Menyelami Framework Laravel ke Bandung
~/Code/autoload-oh-autoloader > master >
```

Tanpa Autoloader

Sip. Seperti yang kita lihat, syntax ini berjalan sebagaimana mestinya. Dan cara inilah yang sering digunakan dari dulu (bahkan sampai sekarang masih dipakai di Wordpress) untuk memanggil Class dari file yang berbeda.



Cara ini, meskipun dapat digunakan, tetapi kurang efektif. *Bayangkan, jika kita memiliki 100 Class di 100 file yang berbeda, apakah mau membuat 100 statement include?*

Tentu tidak. Dan para pendahulu kita yang telah lebih dulu memahami PHP pun memikirkan hal tersebut. Maka, lahirlah fitur autoloader di PHP. Dengan fitur ini, kita tidak perlu menulis `include` untuk setiap file PHP yang akan di masukkan ke file.

Untuk menggunakan autoloader kita akan menggunakan fungsi `spl_autoload_register()`¹⁴. Fungsi ini menerima parameter closure/fungsi yang memiliki sebuah parameter `$class` yang berisi nama class yang akan dipanggil. Di dalam closure ini, kita melakukan `include` ke class yang diinginkan.

¹⁴<http://php.net/manual/en/function.spl-autoload-register.php>

Mari kita praktikan, ubah file `index.php` menjadi :

~/Code/autoload-oh-autoloader/index.php

```
<?php
spl_autoload_register(function ($class) {
    include $class . '.php';
});

$printer = new Printer();
$buku = $printer->cetakBuku('Menyelami Framework Laravel');

$kurir = new Kurir();
$kurir->kirim($buku, 'Bandung');
```

Pada perubahan ini kita menghapus dua statement `include` dan menambahkan syntax:

~/Code/autoload-oh-autoloader/index.php

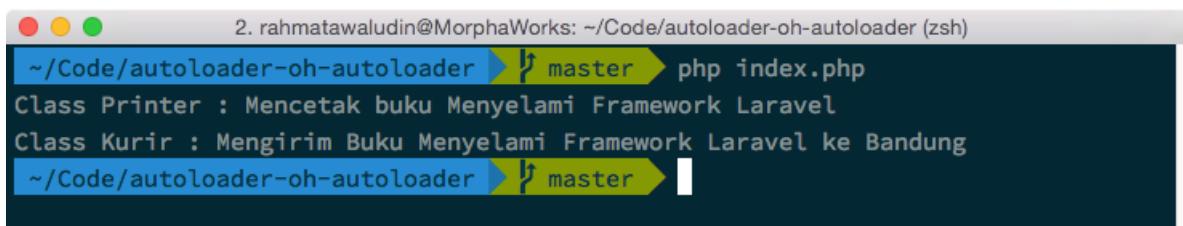
```
<?php
spl_autoload_register(function ($class) {
    include $class . '.php';
});
```

Yang dilakukan syntax ini adalah melakukan `include` setiap kali kita melakukan `new Class()`.

Terlihat di baris ke-3, kita menggunakan `include $class . '.php';`.

Ketika code berjalan (runtime), misalnya kita memanggil `new Printer()`, maka syntax ini akan berubah menjadi `include Printer.php;` yang akan memasukkan konten file tersebut ke code yang sedang aktif. Itulah alasan kita membuat sebuah file untuk sebuah class. Dengan cara ini, kita tidak perlu lagi melakukan `include` manual untuk tiap Class yang dibutuhkan. *Keren kan?*

Kalau dijalankan, outputnya akan tetap sama.



The terminal window shows the command `php index.php` being run in the directory `~/Code/autoload-oh-autoloader`. The output displays two class definitions: `Class Printer : Mencetak buku Menyelami Framework Laravel` and `Class Kurir : Mengirim Buku Menyelami Framework Laravel ke Bandung`. The terminal prompt is `~/Code/autoload-oh-autoloader`.

Setelah Autoloader

Tentunya, autoloader ini dapat disesuaikan dengan kebutuhan kita. Misalnya, semua class berada di folder `class` dan berakhiran `.inc.php`, maka syntax autoloader berubah menjadi :

Merubah autoloader

```
<?php
spl_autoload_register(function ($class) {
    include 'class/' . $class . '.inc.php';
});
```

Laravel sangat aktif menggunakan Autoloader ini. Dengan memahami dasar dari Autoloader ini, mudah-mudahan kita tidak tenggelam dalam kebingungan ketika pembahasan tentang Laravel semakin dalam. Sip.



Source code dari latihan ini bisa didapat di
<https://github.com/rahmatawaludin/autoloader-oh-autoloader>¹⁵

PHP5 Abstract dan Interfaces

Memahami Abstract dan Interfaces sangat penting untuk dapat mendalami framework Laravel. Jika pernah mempelajari mempelajari OOP di PHP, pasti telah memahami Class dan Inheritance. Jika Class dan Inheritance adalah nasi dan sayur asem, maka Abstract dan Interfaces adalah ikan asin dan sambelnya. Nah. :D

Abstract

Oke. Kita mulai dari Abstract. *Apa itu Abstract?*

Abstrak adalah tipe yang tidak dapat dipakai secara langsung. (Wikipedia)

Maksudnya, Abstract itu adalah semacam class di PHP tapi tidak bisa langsung dibuat objectnya. Misalnya, sebuah tombol. Kita semua pasti tahu, bahwa tombol apapun pasti bisa ditekan. Hanya saja, tiap tindakan yang terjadi ketika kita menekan tombol akan berbeda, tergantung jenis tombolnya. Perhatikan contoh ini:

¹⁵<https://github.com/rahmatawaludin/autoloader-oh-autoloader/commits/master>

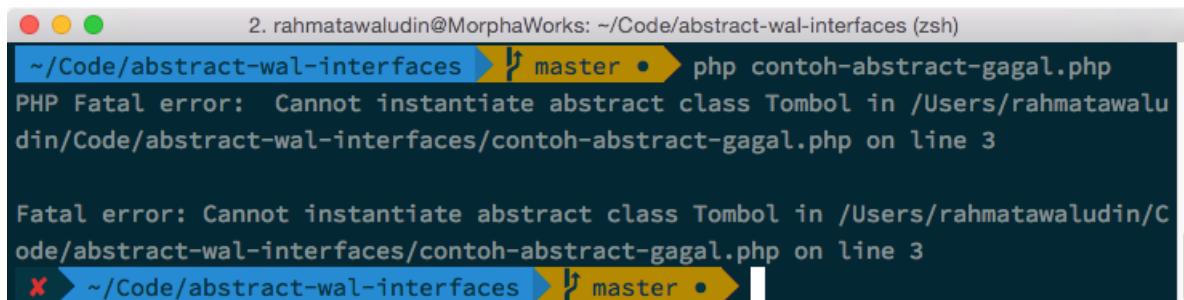
~/Code/abstract-wal-interfaces/Tombol.php

```
<?php  
abstract class Tombol {  
    abstract public function tekan();  
}
```

~/Code/abstract-wal-interfaces/contoh-abstract-gagal.php

```
<?php  
include 'Tombol.php';  
$tombol = new Tombol();  
$tombol->tekan();
```

Terlihat disini, kita langsung mencoba membuat object (*instantiate*) dari abstract class Tombol. Maka, akan ada error seperti ini:



2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➤ ↵ master • ➤ php contoh-abstract-gagal.php
PHP Fatal error: Cannot instantiate abstract class Tombol in /Users/rahmatawaludin/Code/abstract-wal-interfaces/contoh-abstract-gagal.php on line 3

Fatal error: Cannot instantiate abstract class Tombol in /Users/rahmatawaludin/Code/abstract-wal-interfaces/contoh-abstract-gagal.php on line 3
✖ ➤ ~/Code/abstract-wal-interfaces ➤ ↵ master •

Class Abstract tidak bisa langsung dibuat object

Ini menunjukkan bahwa abstract tidak bisa langsung di-*instantiate* menjadi object. Untuk membuat object, kita perlu membuat class baru yang merupakan turunan dari abstract class Tombol ini. Perhatikan syntax ini:

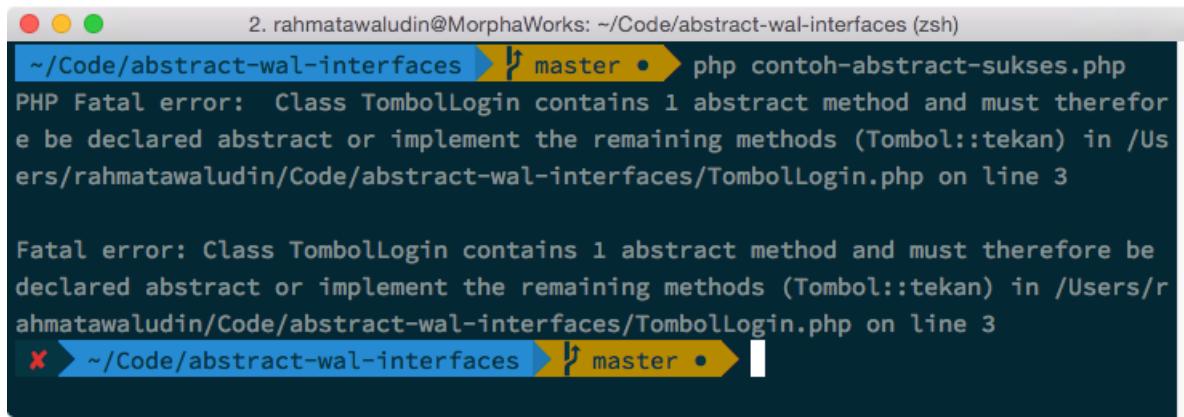
~/Code/abstract-wal-interfaces/TombolLogin.php

```
<?php  
include "Tombol.php";  
class TombolLogin extends Tombol {  
}
```

~/Code/abstract-wal-interfaces/contoh-abstract-sukses.php

```
<?php
include "TombolLogin.php";
$tombol = new TombolLogin();
$tombol->tekan();
```

Disini kita membuat class baru bernama TombolLogin yang meng-*extend* abstract class Tombol. Kalau kita jalankan :



The screenshot shows a terminal window with the following output:

```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➜ master • ➜ php contoh-abstract-sukses.php
PHP Fatal error: Class TombolLogin contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Tombol::tekan) in /Users/rahmatawaludin/Code/abstract-wal-interfaces/TombolLogin.php on line 3

Fatal error: Class TombolLogin contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Tombol::tekan) in /Users/rahmatawaludin/Code/abstract-wal-interfaces/TombolLogin.php on line 3
✖ ➜ ~/Code/abstract-wal-interfaces ➜ master •
```

Gagal meng-extends interface, karena ada method yang belum diimplementasikan

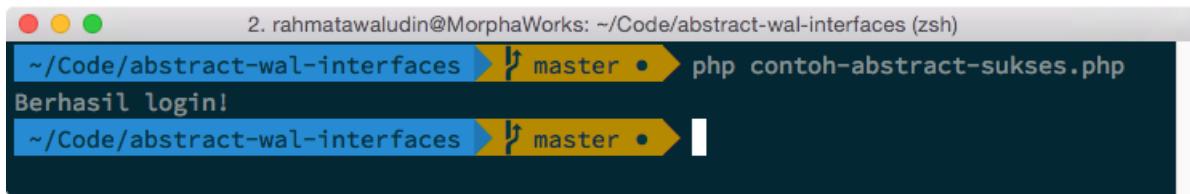
Uuupppss... Dia error lagi.

Ini terjadi karena, kita belum mengimplementasikan method abstract yaitu method tekan() yang ada di abstract class Tombol. Mari kita perbaiki:

~/Code/abstract-wal-interfaces/TombolLogin.php

```
<?php
include "Tombol.php";
class TombolLogin extends Tombol {
    public function tekan() {
        echo "Berhasil login!\n";
    }
}
```

Kalau kita jalankan lagi:



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➜ master • php contoh-abstract-sukses.php
Berhasil login!
~/Code/abstract-wal-interfaces ➜ master •
```

Berhasil menggunakan abstract class Tombol

Tuhh.. dia berhasil kan? :)

Ini penting. Jadi, abstract itu sangat berguna kalau kita ingin memastikan bahwa suatu method selalu tersedia pada class, apapun implementasinya. Dalam contoh ini, kita selalu bisa memanggil method `tekan()` apapun jenis tombolnya. Biar lebih paham, kita tambah lagi contohnya. Kali ini, ceritanya kita mau bikin tombol untuk meluncurkan bom nuklir. Perlu dicatat, ini hanya contoh, tidak ada yang bom yang diluncurkan dalam pembuatan contoh ini. Perhatikan syntax ini:

~/Code/abstract-wal-interfaces/TombolNuklir.php

```
<?php
include "Tombol.php";
class TombolNuklir extends Tombol {
    public function tekan() {
        echo "Bom nuklir telah diluncurkan!\n";
        sleep(3);
        echo "Boooooomm!!\n";
    }
}
```

Untuk menjalankan tombol ini, kita tetap menggunakan method yang sama, yaitu `tekan()`:

~/Code/abstract-wal-interfaces/contoh-abstract-sukses-2.php

```
<?php
include "TombolNuklir.php";
$tombol = new TombolNuklir();
$tombol->tekan();
```

Kalau dijalankan..

```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➜ master • ➜ php contoh-abstract-sukses-2.php
Bom nuklir telah diluncurkan!
Boooooomm!!!
~/Code/abstract-wal-interfaces ➜ master • ➜
```

Berhasil membuat Tombol Nuklir

Boooooomm!!!

Oke, saya rasa sudah cukup penjelasannya. Pertanyaanya, *kapan biasanya teknik abstract class ini digunakan?*

Contohnya banyak, salah satunya saya contohkan dengan dependency injection. Teknik ini memudahkan kita untuk memasukkan (*inject*) class yang kita butuhkan pada class yang sedang digunakan.

Contoh dependency injection, misalnya seorang **Pembeli** harus punya kartu **BNI** untuk melakukan pembayaran. Bisa kita implementasikan dengan meng-*inject* class BNI ke class Pembeli. Seperti ini:

~/Code/abstract-wal-interfaces/BNI.php

```
<?php
```

```
class BNI {

    private $saldo;

    public function __construct($pin) {
        // ceritanya cek PIN ke database
        if ($pin == '12345') {
            echo "Berhasil mengaktifkan Kartu BNI!\n";
        } else {
            $pesan = "PIN yang Anda masukkan salah :(";
            throw new Exception($pesan);
        }
    }

    private function catatTransaksi($jenis, $jumlah) {
        echo "Mencatat transaksi $jenis sejumlah $jumlah ke Buku Tabungan.\n";
    }

    public function kredit($jumlah) {
        $this->catatTransaksi('transfer keluar', $jumlah);
```

```

    $this->saldo -= $jumlah;
}

public function deposit($jumlah) {
    $this->catatTransaksi('deposit dana', $jumlah);
    $this->saldo += $jumlah;
}

public function cekSaldo() {
    return $this->saldo;
}
}

```

Class BNI ini mempunyai:

- attribute `$saldo` yang berfungsi mencatat saldo terakhir.
- method `__construct()` yang berfungsi membangun object BNI, di method ini kita mengharuskan input PIN. Dalam prakteknya, tentu saja PIN ini disimpan di database, tapi disini kita sederhanakan dengan menyimpannya langsung di method ini.
- method `kredit()` yang berfungsi untuk mengurangi jumlah saldo.
- method `deposit()` yang berfungsi untuk menambah jumlah saldo.
- method `cekSaldo()` yang berfungsi mengecek jumlah saldo terkini.

Mari kita buat class Pembeli, Class ini akan membutuhkan class BNI :

~/Code/abstract-wal-interfaces/Pembeli-DI.php

```

<?php
include "BNI.php";
class Pembeli {
    private $nama;
    private $bni;

    public function __construct($nama = "Seseorang", BNI $bni) {
        $this->bni = $bni;
        $this->nama = $nama;
    }

    public function beli($nama = "Barang", $harga = 0) {
        $this->bni->kredit($harga);
        echo "Berhasil melakukan pembelian $nama seharga Rp$harga.\n";
    }
}

```

```
    echo "Terima kasih $this->nama :)\n";
}
}
```

Class Pembeli ini mempunyai :

- atribut \$nama untuk menyimpan nama pembeli
- atribut \$bni untuk menyimpan object kartu BNI
- method beli() untuk melakukan pembelian barang
- method __construct() untuk membangun object Pembeli

Yang paling penting untuk diperhatikan disini adalah method __construct(). Di method ini kita menginject class BNI sebagai parameternya. Kalau didemokan syntaxnya seperti ini:

~/Code/abstract-wal-interfaces/beli-pakai-bni.php

```
<?php
require_once "Pembeli-DI.php";
// Melakukan pembelian dengan BNI
try {
    $bniKu = new BNI('12345');
    $bniKu->deposit(20000000);
    $rudy = new Pembeli("Rudy", $bniKu);
    $rudy->beli("CD Smash - Step Forward", 80000);
    echo "Saldo terakhir Rp".$bniKu->cekSaldo()."\\n";
} catch (Exception $e) {
    echo $e->getMessage()."\\n";
}
```

Terlihat disini, sebelum membuat object Pembeli, kita membuat object BNI dulu. Kemudian meng-*inject* object BNI itu ketika membuat object Pembeli. Jika dijalankan, hasilnya seperti berikut :

```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➜ master • ➜ php beli-pakai-bni.php
Berhasil mengaktifkan Kartu BNI!
Mencatat transaksi deposit dana sejumlah 20000000 ke Buku Tabungan.
Mencatat transaksi transfer keluar sejumlah 8000 ke Buku Tabungan.
Berhasil melakukan pembelian CD Smash - Step Forward seharga Rp80000.
Terima kasih Rudy :)
Saldo terakhir Rp19920000
~/Code/abstract-wal-interfaces ➜ master ➜
```

Berhasil meng-*inject* BNI ke Pembeli

Masalah dari dependency injection ini adalah bagaimana bila kita akan menggunakan metode pembayaran lain? Misalnya, Paypal. Tentunya, cara mengakses paypal ini pun akan berbeda dengan BNI, karena Paypal harus login dengan email dan password. Begitupun cara paypal melakukan kredit dan deposit, karena paypal perlu mengirim email setiap kali terjadi transaksi.

Kalau gitu langsung di extends dari class BNI saja gimana mas?

Memang, sekiranya implementasinya akan sama, kita cukup meng-extends class `Paypal` dari `BNI`. Namun, karena implementasi dari method `kredit()` dan `deposit()` ini bisa berbeda, maka fitur pembayaran ini cocok untuk di-abstraksi. Dengan abstraksi pula, akan lebih memudahkan jika akan ada implementasi jenis pembayaran yang baru, misalnya BitCoin.

Kita bisa membuat abstraksi dengan membuat class abstract yang berisi method apa saja yang harus ada di Class tersebut yang akan digunakan di class `Pembeli`. Tahapannya dimulai dari class `Pembeli`, ubah menjadi :

~/Code/abstract-wal-interfaces/Pembeli.php

```
<?php
class Pembeli {
    private $nama;
    private $payment;

    public function __construct($nama = "Seseorang", PaymentMethod $payment) {
        $this->nama = $nama;
        $this->payment = $payment;
    }

    public function beli($nama = "Barang", $harga = 0) {
        if ($this->payment->cekSaldo() < $harga) {
```

```

        echo "Uang tidak cukup\n";
    } else {
        $this->payment->kredit($harga);
        echo "Terima kasih $this->nama :)\n";
        echo "Berhasil melakukan pembelian $nama seharga Rp".number_format($harga)."\n";
    }
}
}
}

```

Perubahan terbesar dari class Pembeli adalah kita mengubah atribut `$bni` menjadi `$payment` dan mengubah mengabstraksi class `BNI` menjadi `PaymentMethod`.

Terlihat disini, class `PaymentMethod` perlu menambahkan beberapa method:

- `cekSaldo()` untuk mengecek saldo terakhir
- `kredit()` untuk mengambil sejumlah uang
- `deposit()` untuk mengisi sejumlah uang

Untuk memudahkan pengecekan, kita akan menambah method `cekNamaPembayaran()` yang berfungsi menampilkan nama Class yang digunakan untuk melakukan pembayaran. Mari kita buat abstract class `PaymentMethod` :

~/Code/abstract-wal-interfaces/PaymentMethod.php

```

<?php
abstract class PaymentMethod {
    public function cekNamaPembayaran() {
        return "Anda melakukan pembayaran dengan ".get_class($this)."\n";
    }
    abstract public function kredit($jumlah);
    abstract public function deposit($jumlah);
    abstract public function cekSaldo();
}

```

Selanjutnya, ubah class `BNI` agar mengekstends `PaymentMethod`. Untuk memudahkan contoh, kita ubah nama classnya menjadi `DebitBNI`:

~/Code/abstract-wal-interfaces/DebitBNI.php

```
<?php
require_once "PaymentMethod.php";
class DebitBNI extends PaymentMethod {
    private $saldo;

    public function __construct($pin) {
        // ceritanya cek PIN ke database
        if ($pin == '12345') {
            echo "Berhasil mengaktifkan Kartu Debit!\n";
        } else {
            $pesan = "PIN yang Anda masukkan salah :(";
            throw new Exception($pesan);
        }
    }

    private function catatTransaksi($jenis, $jumlah) {
        echo "Mencatat transaksi $jenis sejumlah $jumlah ke Buku Tabungan.\n";
    }

    public function kredit($jumlah) {
        $this->catatTransaksi('transfer keluar', $jumlah);
        $this->saldo -= $jumlah;
    }

    public function deposit($jumlah) {
        $this->catatTransaksi('deposit dana', $jumlah);
        $this->saldo += $jumlah;
    }

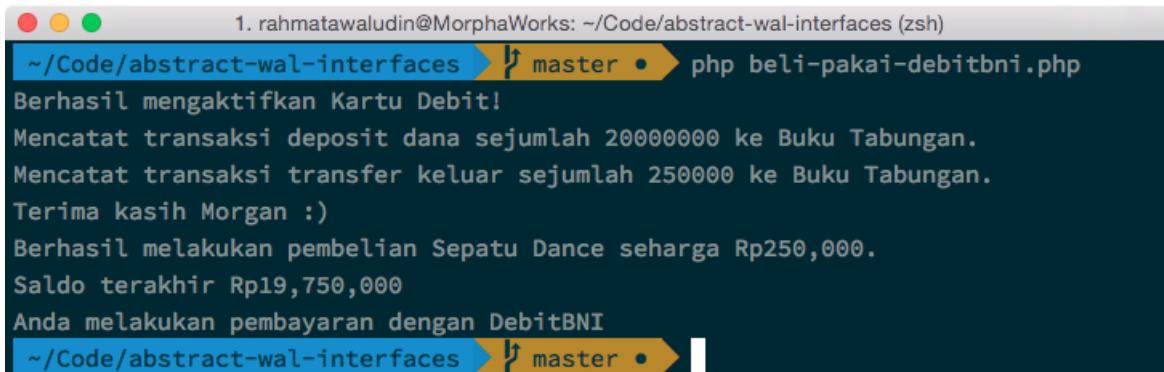
    public function cekSaldo() {
        return $this->saldo;
    }
}
```

Mari kita buat demo untuk metode pembayaran ini:

~/Code/abstract-wal-interfaces/beli-pakai-debitbni.php

```
<?php  
require_once "DebitBNI.php";  
require_once "Pembeli.php";  
  
// Melakukan pembelian dengan DebitBNI  
try {  
    $paymentMethod = new DebitBNI("12345");  
    $paymentMethod->deposit(20000000);  
    $rahmat = new Pembeli("Morgan", $paymentMethod);  
    $rahmat->beli("Sepatu Dance", 250000);  
    echo "Saldo terakhir Rp".number_format($paymentMethod->cekSaldo())."\n";  
    echo $paymentMethod->cekNamaPembayaran();  
} catch (Exception $e) {  
    echo $e->getMessage()."\n";  
}
```

Jika dijalankan, hasilnya akan seperti ini :



```
1. rahmatalawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)  
~/Code/abstract-wal-interfaces ➔ ⚡ master • ➔ php beli-pakai-debitbni.php  
Berhasil mengaktifkan Kartu Debit!  
Mencatat transaksi deposit dana sejumlah 20000000 ke Buku Tabungan.  
Mencatat transaksi transfer keluar sejumlah 250000 ke Buku Tabungan.  
Terima kasih Morgan :)  
Berhasil melakukan pembelian Sepatu Dance seharga Rp250,000.  
Saldo terakhir Rp19,750,000  
Anda melakukan pembayaran dengan DebitBNI  
~/Code/abstract-wal-interfaces ➔ ⚡ master • ➔
```

Membuat DebitBNI dengan abstract PaymentMethod

Di baris terakhir output terlihat kita menggunakan implementasi PaymentMethod dengan class DebitBNI.

Untuk implementasi Paypal, kita buat seperti ini:

~/Code/abstract-wal-interfaces/Paypal.php

```
<?php
require_once 'PaymentMethod.php';
class Paypal extends PaymentMethod {
    private $balance;
    private $email;

    public function __construct($email, $password) {
        // Ceritanya ini akses ke database
        if ($email == "morgan@gmail.com" & $password == "12345") {
            $this->email = $email;
            echo "Berhasil login ke Paypal!\n";
        } else {
            $pesan = "User ada user dengan username/password tersebut :(";
            throw new Exception($pesan);
        }
    }

    private function kirimNotifikasi($pesan = "Informasi penting") {
        echo "Mengirim email notifikasi $pesan ke $this->email \n";
    }

    public function kredit($jumlah) {
        $this->kirimNotifikasi('pengeluaran dana');
        $this->balance -= $jumlah;
    }

    public function deposit($jumlah) {
        $this->kirimNotifikasi('penerimaan dana');
        $this->balance += $jumlah;
    }

    public function cekSaldo() {
        return $this->balance;
    }
}
```

Terlihat disini, class Paypal ini mengimplementasikan semua method dari class abstract PaymentMethod, ini diharuskan. Karena, sebagaimana saya jelaskan di pembahasan sebelumnya, class yang meng-*extends* abstract class harus mengimplementasikan semua abstract methodnya. Jika tidak, aplikasi akan error.

Perbedaan lain di class Paypal adalah :

- Untuk membuat object harus menggunakan email dan password yang kita hard-code di method `__construct()`. Tentunya, di kenyataannya kita akan mengecek ini ke database.
- atribut `$balance` digunakan untuk menyimpan dana.
- Setiap kali ada transaksi uang masuk atau keluar, memanggil method `kirimNotifikasi()` yang disimulasikan akan mengirim email.

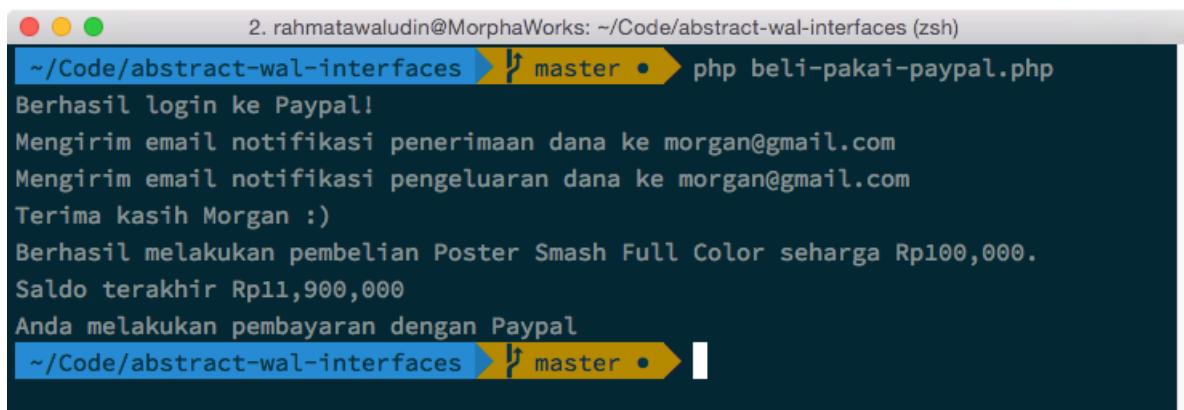
Demo dari metode pembayaran ini :

`~/Code/abstract-wal-interfaces/beli-pakai-paypal.php`

```
<?php
require_once "Paypal.php";
require_once "Pembeli.php";

// Melakukan pembelian dengan paypal
try {
    $paymentMethod = new Paypal("morgan@gmail.com", "12345");
    $paymentMethod->deposit(12000000);
    $pembeli = new Pembeli("Morgan", $paymentMethod);
    $pembeli->beli("Poster Smash Full Color", 100000);
    echo "Saldo terakhir Rp".number_format($paymentMethod->cekSaldo())."\n";
    echo $paymentMethod->cekNamaPembayaran();
} catch (Exception $e) {
    echo $e->getMessage()."\n";
}
```

Outputnya akan menjadi :



```
2. rahmatalawudin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➜ master • ➜ php beli-pakai-paypal.php
Berhasil login ke Paypal!
Mengirim email notifikasi penerimaan dana ke morgan@gmail.com
Mengirim email notifikasi pengeluaran dana ke morgan@gmail.com
Terima kasih Morgan :)
Berhasil melakukan pembelian Poster Smash Full Color seharga Rp100,000.
Saldo terakhir Rp11,900,000
Anda melakukan pembayaran dengan Paypal
~/Code/abstract-wal-interfaces ➜ master •
```

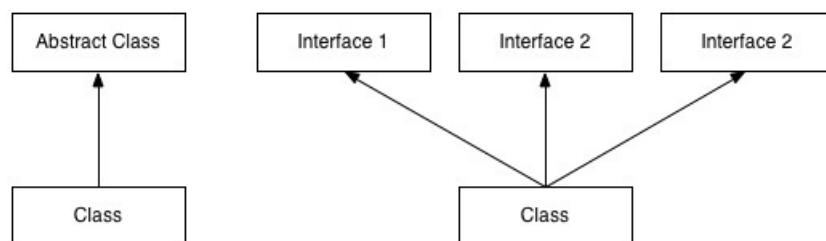
Membuat Paypal dengan abstract PaymentMethod

Jika diperhatikan, method pada class `$paymentMethod` yang kita gunakan disini, sama dengan method yang kita pakai di demo dengan pembayaran DebitBNI. Inilah kekuatan dari abstract class. **Kita bisa melakukan standarisasi nama method, apapun bentuk implementasinya.**

Nah.. udah ah. Segini aja dulu. Silahkan istirahat, ambil snack dan minumannya ya..

Interfaces

Interface hampir mirip dengan abstract class. Dimana kita harus membuat method yang ada di Interface pada class yang mengimplementasikan interfaces tersebut. Perbedaannya adalah sebuah class hanya bisa meng-extends satu abstract class, tapi bisa mengimplementasikan banyak interfaces. Kira-kira seperti ini ilustrasinya:



Ilustrasi Abstract dan Interfaces

Misalnya, ada class Kucing, dia meng-extends abstract class HewanMamalia. Nah, dia juga bisa mengimplementasikan interface sayap, jadi dia bisa punya method terbang(). Atau ditambah lagi dengan implementasi interface Insang, jadi dia bisa punya method menyelam() (kebayang ngga kucing yang bisa terbang dan bisa menyelam? :v). Tapi, kucing ini tidak bisa meng-extends abstract class HewanMelata. Karena, dia hanya boleh meng-extends satu abstract class (dalam contoh ini HewanMamalia). Sip.

Apa manfaat Interface?

Banyak. Salah satunya, Interface biasanya digunakan untuk kita melakukan komunikasi antar object yang tidak saling berhubungan. Contoh, misalnya kita ingin membuat sebuah Class untuk membandingkan statistik berbagai akun di media sosial, kita sebut saja SocialGraph. Class ini punya fungsi `compareLike` yang bisa membandingkan membandingkan jumlah Like di berbagai akun media sosial. Versi pertama dari SocialGraph, hanya bisa membandingkan jumlah like antar akun Facebook.

~/Code/abstract-wal-interfaces/SocialGraph.php

```
<?php
class SocialGraph {
    public static function compareLike(Facebook $fb1, Facebook $fb2) {
        if ($fb1->totalLike() > $fb2->totalLike()) {
            echo "Status " . $fb1->user() . " Lebih populer dari " . $fb2->user() .
        "\n";
        } elseif ($fb1->totalLike() < $fb2->totalLike()) {
            echo "Status " . $fb2->user() . " Lebih populer dari " . $fb1->user() .
        "\n";
        } else {
            echo "Status kedua user sama-sama populer";
        }
    }
}
```

Yang perlu diperhatikan disini adalah kita menggunakan *type hinting* object class Facebook pada method `compareLike`. Ini digunakan agar script menampilkan error jika user tidak memberikan parameter berupa object Facebook.

Kita juga menggunakan method `totalLike()` untuk membandingkan popularitas antar status Facebook. Dengan class Facebook :

~/Code/abstract-wal-interfaces/Facebook.php

```
<?php
class Facebook {
    private $status;
    private $user;
    private $like = 0;
    public function __construct($user, $status) {
        $this->user = $user;
        $this->status = $status;
    }
    public function status() {
        return $this->status;
    }
    public function user() {
        return $this->user;
    }
    public function like() {
        $this->like++;
    }
}
```

```
    }
    public function totalLike() {
        return $this->like;
    }
}
```

Demokan dengan syntax ini :

~/Code/abstract-wal-interfaces/demo-socialgraph.php

```
<?php
include "Facebook.php";
include "SocialGraph.php";

$fbTukul = new Facebook("Tukul Arwana", "Kembali ke laptop!");
$fbTukul->like();
$fbTukul->like();

$fbJokowi = new Facebook("Joko Widodo", "Aku rapopo..");
$fbJokowi->like();
$fbJokowi->like();
$fbJokowi->like();

$socialGraph = new SocialGraph();
$socialGraph->compareLike($fbTukul, $fbJokowi);
```

Terlihat disini, status Tukul mendapat 2 like, sedangkan status Jokowi mendapat 3 like. Jika dijalankan:



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➔ master • ➔ php demo-socialgraph.php
Facebook Joko Widodo Lebih populer dari Facebook Tukul Arwana
~/Code/abstract-wal-interfaces ➔ master •
```

SocialGraph membandingkan popularitas status facebook

Oke, SocialGraph versi 1 sudah berjalan. Aplikasi kita terus berkembang, kini kita akan mensupport akun Twitter. Misalnya dengan class Twitter seperti berikut:

~/Code/abstract-wal-interfaces/Twitter.php

```
<?php
class Twitter {
    private $tweet;
    private $user;
    private $favorite;
    public function __construct($user, $tweet) {
        $this->user = $user;
        $this->tweet = $tweet;
    }
    public function tweet() {
        return $this->tweet;
    }
    public function user() {
        return $this->user;
    }
    public function favorite() {
        $this->favorite++;
    }
    public function totalFavorite() {
        return $this->favorite;
    }
}
```

Disinilah masalahnya muncul, cara menentukan jumlah like di twitter ternyata berbeda. Karena, twitter tidak mengenal istilah *like*, yang dikenal di twitter adalah *favorite*. Jika kita coba bandingkan popularitas akun facebook dengan twitter, pasti tidak akan bisa. Misalnya dengan syntax :

~/Code/abstract-wal-interfaces/demo-socialgraph.php

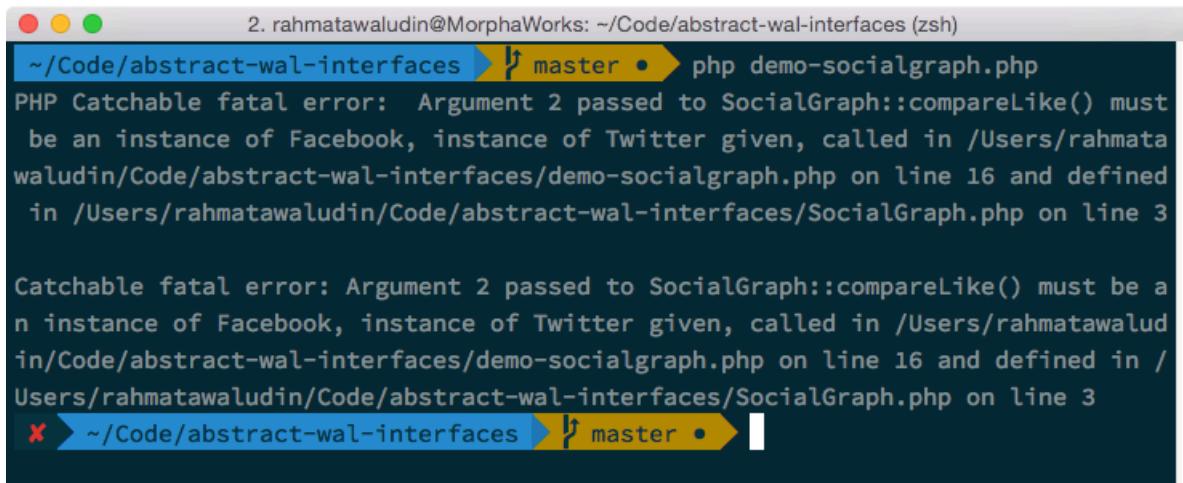
```
<?php
include "Facebook.php";
include "Twitter.php";
include "SocialGraph.php";

$fbTukul = new Facebook("Tukul Arwana", "Kembali ke laptop!");
$fbTukul->like();
$fbTukul->like();

$twJokowi = new Twitter("Joko Widodo", "Aku rapopo..");
$twJokowi->favorite();
```

```
$twJokowi->favorite();  
$twJokowi->favorite();  
  
$socialGraph = new SocialGraph();  
$socialGraph->compareLike($fbTukul, $twJokowi);
```

Akan muncul error



```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)  
~/Code/abstract-wal-interfaces ➜ master • ➔ php demo-socialgraph.php  
PHP Catchable fatal error: Argument 2 passed to SocialGraph::compareLike() must  
be an instance of Facebook, instance of Twitter given, called in /Users/rahmata  
waludin/Code/abstract-wal-interfaces/demo-socialgraph.php on line 16 and defined  
in /Users/rahmatawaludin/Code/abstract-wal-interfaces/SocialGraph.php on line 3  
  
Catchable fatal error: Argument 2 passed to SocialGraph::compareLike() must be a  
n instance of Facebook, instance of Twitter given, called in /Users/rahmatawalud  
in/Code/abstract-wal-interfaces/demo-socialgraph.php on line 16 and defined in /  
Users/rahmatawaludin/Code/abstract-wal-interfaces/SocialGraph.php on line 3  
✖ ➜ ~/Code/abstract-wal-interfaces ➜ master •
```

Tidak bisa membandingkan jumlah like facebook dengan favorite twitter

Ini terjadi karena method `compareLike()` hanya menerima input berupa object Facebook. Ada beberapa cara untuk menyelesaiannya:

1. Gunakan method yang berbeda untuk membandingkan facebook dan twitter misalnya `compareFacebookTwitter(Facebook $fb, Twitter $tw)`. Tapi akan muncul masalah baru, bagaimana jika membandingkan Twitter dengan Twitter apa perlu dibuat method baru `compareTwitterTwitter()`? Dan bagaimana jika membandingkan Twitter ke Facebook apa dibuat `compareTwitterFacebook()`? Solusi ini tidak akan digunakan.
2. Gunakan parent class yang sama, misalnya `Social` untuk class Facebook dan Twitter. Dan ubah parameter method `compareTo` menjadi `social`. Solusi ini akan berhasil sekarang, tapi akan merepotkan jika kita akan mengimplementasikan fitur `compareLike` ini pada object lain, misalnya `Artikel` di web, `Video` di Youtube, dll. Solusi ini tidak akan digunakan.
3. Buat interface `Likeable` untuk menunjukkan object yang bisa di Like. Solusi ini yang akan kita gunakan.

Mari kita buat interfacenya :

~/Code/abstract-wal-interfaces/Likeable.php

```
<?php
interface Likeable {
    public function platform();
    public function user();
    public function totalLike();
    public function like();
}
```

Disini, kita mendefinisikan beberapa method yang harus di buat pada Class yang mengimplementasikan Likeable yaitu platform(), user(), totalLike() dan like().

Mari kita implementasikan Likeable di class Facebook :

~/Code/abstract-wal-interfaces/Facebook.php

```
<?php
include_once "Likeable.php";
class Facebook implements Likeable {

    ...
    public function platform() {
        return "Facebook";
    }
}
```

Dan di class Twitter :

~/Code/abstract-wal-interfaces/Twitter.php

```
<?php
include_once "Likeable.php";
class Twitter implements Likeable {

    ...
    public function like() {
        $this->favorite();
    }
    public function totalLike() {
        return $this->totalFavorite();
    }
    public function platform() {
        return "Twitter";
    }
}
```

Kita ubah juga method `compareLike` di class `SocialGraph` agar parameter nya menjadi interface `Likeable` :

`~/Code/abstract-wal-interfaces/SocialGraph.php`

```
<?php
class SocialGraph {
    public static function compareLike(Likeable $social1, Likeable $social2) {
        if ($social1->totalLike() > $social2->totalLike()) {
            echo $social1->platform() . " ". $social1->user();
            echo " Lebih populer dari " . $social2->platform() . " ";
            echo $social2->user() . "\n";
        } elseif ($social1->totalLike() < $social2->totalLike()) {
            echo $social2->platform() . " ". $social2->user();
            echo " Lebih populer dari " . $social1->platform() . " ";
            echo $social1->user() . "\n";
        } else {
            echo "Kedua user sama-sama populer.\n";
        }
    }
}
```

Sehingga, kalau kita jalankan lagi demonya :

```
2. rahmatawaludin@MorphaWorks: ~/Code/abstract-wal-interfaces (zsh)
~/Code/abstract-wal-interfaces ➔ master • ➔ php demo-socialgraph.php
Twitter Joko Widodo Lebih populer dari Facebook Tukul Arwana
~/Code/abstract-wal-interfaces ➔ master •
```

Berhasil membandingkan favorite di Twitter dan like di Facebook

Berhasil kan?

Dengan cara ini kita berhasil membandingkan (mengkomunikasikan) jumlah like di facebook dan jumlah favorite di Twitter. Jika kedepannya kita akan mendukung media yang lain, misalnya video di Youtube, maka class `VideoYoutube` tersebut tinggal mengimplementasikan interface `Likeable`.

Jika kita ambil kesimpulan, ketika kita menggunakan abstract class kita fokus pada **siapa Class ini** (*instance of*). Sedangkan, ketika kita menggunakan interface kita fokus pada **apa yang bisa dilakukan Class ini** (*capable of*).

Dengan menggunakan interface, kita dapat menyeragamkan API (method `like()`) untuk aplikasi kita, meskipun jenis implementasinya beragam (facebook, twitter, youtube, dll).

Interfaces sangat membantu dalam membangun aplikasi skala besar. Penjelasan saya ini merupakan contoh sederhana penggunaan interface di lapangan, masih banyak contoh lainnya.

Laravel menggunakan banyak interfaces dalam bentuk contract untuk berbagai fiturnya. Sehingga memungkinkan kita untuk merubah implementasinya sesuai kehendak kita. Contoh penggunaan sederhananya seperti ini:

- Mau merubah implementasi view dari Blade ke Twig? Bisa.
- Mau merubah penyimpanan Cache ke MongoDB? Bisa.
- Mau merubah implementasi Queue ke driver lain? Bisa.
- dll.

Dan semua perubahan itu bisa dilakukan hanya dengan merubah implementasi *contract (interfaces)* yang berhubungan. **Serius, Keren.**

Tentunya tidak akan cukup kalau saya jelaskan contract di Laravel pada bab ini. Oleh karena itu, pembahasan lebih lanjut tentang contract akan kita bahas pada bab **Arsitektur Laravel**.



Source code dari latihan ini bisa didapat di
[https://github.com/rahmatawaludin/abstract-wal-interfaces¹⁶](https://github.com/rahmatawaludin/abstract-wal-interfaces)

PHP5 Traits

PHP 5.4 memperkenalkan fitur [Traits¹⁷](#). Fitur ini agak mirip dengan interface yang telah kita pelajari sebelumnya. Bedanya, jika interface memiliki method yang harus diimplementasikan logic nya tiap kali digunakan. Sedangkan traits, ketika digunakan kita langsung dapat menggunakan methodnya tanpa menulis logic untuk method tersebut. Hal ini bisa dilakukan karena dalam trait sebuah method dideklarasikan berikut dengan logicnya.

Coba cermati syntax interface berikut :

¹⁶ <https://github.com/rahmatawaludin/abstract-wal-interfaces>

¹⁷ <http://php.net/manual/en/language.oop5.traits.php>

~/Code/traits-copy-paste/Sayap.php

```
<?php
interface Sayap {
    public function terbang();
}
```

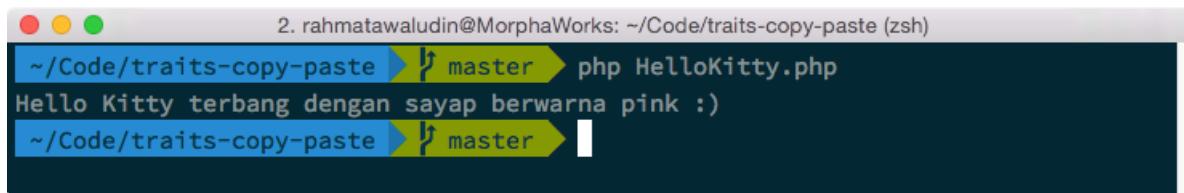
Jika kita hendak mengimplementasikan interface Sayap ini ke sebuah class, misalnya ke class HelloKitty maka kita harus mengimplementasikan method terbang():

~/Code/traits-copy-paste/HelloKitty.php

```
<?php
include "Sayap.php";
class HelloKitty implements Sayap {
    public function terbang() {
        echo "Hello Kitty terbang dengan sayap berwarna pink :)\n";
    }
}

$helloKitty = new HelloKitty();
$helloKitty->terbang();
```

Jika dijalankan :



The screenshot shows a terminal window with the following content:

```
2. rahmatawaludin@MorphaWorks: ~/Code/traits-copy-paste (zsh)
~/Code/traits-copy-paste > master > php HelloKitty.php
Hello Kitty terbang dengan sayap berwarna pink :)
```

Hello Kitty berhasil terbang dengan interface Sayap

Tentunya, jika kita tidak membuat method terbang() di Class HelloKitty, maka script ini pasti akan memunculkan error.

Berbeda dengan interface, traits mengizinkan kita untuk tidak menulis kembali method yang berada di Traits. Kita cukup menggunakan keyword `use <nama_traits>` dan semua method yang berada di dalam traits itu langsung dapat digunakan di class kita.

Dalam membuat traits, logic dari method-methodnya berada di Traits. Bukan di class yang menggunakan traits. Misalnya kita membuat Traits JetPack:

~/Code/traits-copy-paste/JetPack.php

```
<?php
trait JetPack
{
    function terbang()
    {
        echo __CLASS__." terbang dengan JetPack. Wuzzz... :D\n";
    }
}
```

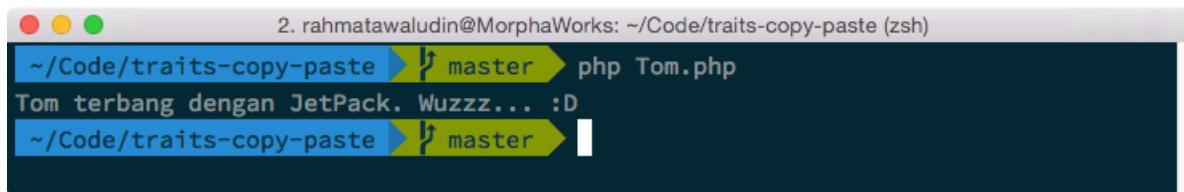
Contoh penggunaan Trait ini di class Tom:

~/Code/traits-copy-paste/Tom.php

```
<?php
include "JetPack.php";
class Tom {
    use JetPack;
}

$tom = new Tom();
$tom->terbang();
```

Terlihat disini, kita tidak perlu mendefinisikan logic dari method terbang() dari trait JetPack di class Tom. Jika dijalankan:



The screenshot shows a terminal window with the following content:

```
2. rahmataludin@MorphaWorks: ~/Code/traits-copy-paste (zsh)
~/Code/traits-copy-paste > master > php Tom.php
Tom terbang dengan JetPack. Wuzzz... :D
~/Code/traits-copy-paste > master >
```

Logic dari method di Trait berada di Trait

Cara kerja trait ini adalah dengan **melakukan copy paste syntax dari trait ke class yang menggunakan pada saat script berjalan (runtime)**.

Mas, apa contoh penggunaan traits di dunia nyata?

Traits, biasanya digunakan untuk menambahkan fungsionalitas pada child class, dimana child class yang lain tidak membutuhkan fungsionalitas tersebut. Misalnya, ada class SocialThing yang merupakan abstract class dari semua object di media sosial.

~/Code/traits-copy-paste/SocialThing.php

```
<?php
abstract class SocialThing {
    private $content;
    public function __construct($content) {
        $this->content = $content;
    }
    public function content() {
        return $this->content;
    }
}
```

Class turunan dari SocialThing misalnya Status, Photo dan Message. Ketiganya memiliki karakteristik yang sama. Bedanya, Status dan Photo bisa di share sedangkan Message tidak bisa (karena message bersifat privat antar user). Masalah seperti ini bisa diselesaikan dengan trait. Syntaxnya akan seperti ini:

~/Code/traits-copy-paste/Shareable.php

```
<?php
trait Shareable {
    public function share() {
        echo "Sharing ".$this->content()."\n";
    }
}
```

~/Code/traits-copy-paste/Status.php

```
<?php
include "SocialThing.php";
include "Shareable.php";
class Status extends SocialThing {
    use Shareable;
}

$status = new Status("Menikah itu mendewasakan. Bukan menunggu dewasa. Berhentilah mencari alasan. Move On!");
$status->share();
```

~/Code/traits-copy-paste/Photo.php

```
<?php
include "SocialThing.php";
include "Shareable.php";
class Photo extends SocialThing {
    use Shareable;
}

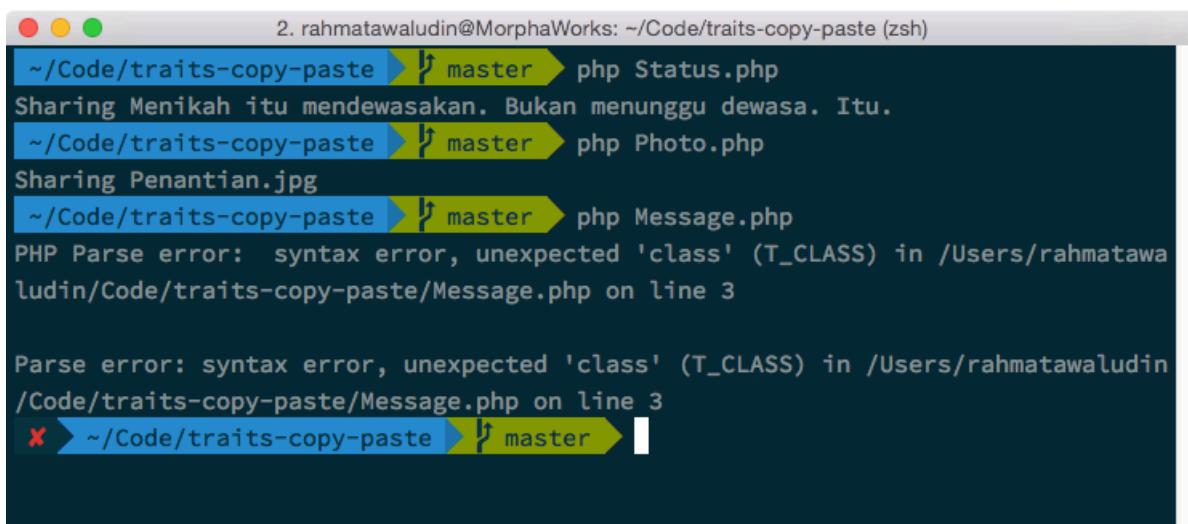
$photo = new Photo('Penantian.jpg');
$photo->share();
```

~/Code/traits-copy-paste/Message.php

```
<?php
include "SocialThing.php";
class Message extends SocialThing { }

$message = new Message("De, mas sudah siap untuk nikah. Besok mau ke rumah ketemu\\
u Bapak. Do'akan ya.. ");
$message->share();
```

Mari kita jalankan :



```
2. rahmatawaludin@MorphaWorks: ~/Code/traits-copy-paste (zsh)
~/Code/traits-copy-paste ➤ master ➤ php Status.php
Sharing Menikah itu mendewasakan. Bukan menunggu dewasa. Itu.
~/Code/traits-copy-paste ➤ master ➤ php Photo.php
Sharing Penantian.jpg
~/Code/traits-copy-paste ➤ master ➤ php Message.php
PHP Parse error:  syntax error, unexpected 'class' (T_CLASS) in /Users/rahmatawala
ludin/Code/traits-copy-paste/Message.php on line 3

Parse error: syntax error, unexpected 'class' (T_CLASS) in /Users/rahmatawaludin
/Code/traits-copy-paste/Message.php on line 3
✖ ➤ ~/Code/traits-copy-paste ➤ master ➤
```

Bisa share status dan photo, tapi tidak bisa share message

Terlihat disini, kita meskipun Status, Photo dan Message merupakan child dari SocialThing, hanya Status dan Photo yang bisa kita share. Sementara, Message memunculkan

error ketika kita akan mencoba melakukan share. Kedepannya, jika Message akan memiliki fitur `share`, kita cukup menggunakan trait `Shareable`.

Laravel, mewajibkan kita menggunakan PHP 5.4 sebagai syarat minimun. Alasannya, Laravel menggunakan traits di dalam corenya. Trait ini salah satunya digunakan di ORM nya, yaitu Eloquent. Pembahasan tentang Laravel dan Trait, akan kita bahas di bab-bab selanjutnya. Sip.

 Source code dari latihan ini bisa didapat di [https://github.com/rahmatawaludin/traits-copy-paste¹⁸](https://github.com/rahmatawaludin/traits-copy-paste)

PHP5 Namespace

Name = Nama

Space = Ruang

Namespace = Ruang Nama (Google translate)

Prinsip pertama ketika belajar coding, jangan percaya pada google translate. Jika pernah translate artikel coding di google translate, dapat dipastikan malah akan bertambah... bingung. :v

Tapi, sebenarnya kita dapat mengambil beberapa kesimpulan dari hasil Google Translate ini. Jika **Name = Nama** dan **Space = Ruang**, maka namespace adalah semacam ruangan yang memiliki nama.

Tentunya ruangan itu pasti digunakan untuk menyimpan "sesuatu". Nah, "sesuatu" itu dalam PHP salah satunya adalah class. Itulah namespace.

Namespace di PHP, merupakan fitur baru yang dapat membantu kita untuk menge lumpukan class dalam sebuah ruangan virtual yang bisa kita beri nama.

Saya paham, pemrograman itu hal yang sangat abstrak. Oleh karena itu, mari kita buat analogi yang lebih membumi. Misalnya ada sebuah kelas di SD. Di kelas ini ada 30 orang siswa. Karena orang tuanya kurang kreatif, nama anaknya banyak yang sama. Di kelas ini ada 3 Agus: Agus putra pak Wahyu, Agus putra pak Beni dan Agus putra pak Dudung. Ada pula 2 Siti: Siti putra pak Wahyu dan Siti putra pak Roni. Nah, untuk memudahkan memanggil anak-anak ini kita bisa menggunakan nama ayahnya sebagai akhiran, misalnya:

¹⁸<https://github.com/rahmatawaludin/traits-copy-paste/commits/master>

- Agus Wahyu
- Agus Beni
- Agus Dudung
- Siti Wahyu
- Roni Siti

Disini nama ayahnya kita tulis dibelakang. Dalam namespace di php, nama namespace (dalam hal ini Ayah), dibuat di depan kemudian dilanjutkan dengan garis miring dan nama class (dalam hal ini Siswa). Jika semua siswa itu dibuat dalam namespace Ayahnya, maka penulisannya seperti ini:

- Wahyu\Agus
- Beni\Agus
- Dudung\Agus
- Wahyu\Siti
- Roni\Siti

Nah, sekarang jadi lebih gampang untuk membedakan anak-anak itu kan? Sekarang, lanjut ke coding.

Mengelompokan Class

Salah satu manfaat dari namespace bisa digunakan untuk mengelompokan class dari vendor yang sama. Pengelompokan ini juga memungkinkan kita membuat nama class yang sama, asalkan namespace (vendor) nya berbeda.

Di bahasa PHP, kita tidak bisa memiliki nama class yang sama dalam satu aplikasi. Misalnya, kita akan menggunakan class `URLShortener` dari buatan kita, dan `URLShortener` buatan Bitly.

`~/Code/namespace-oh-namespace/index.php`

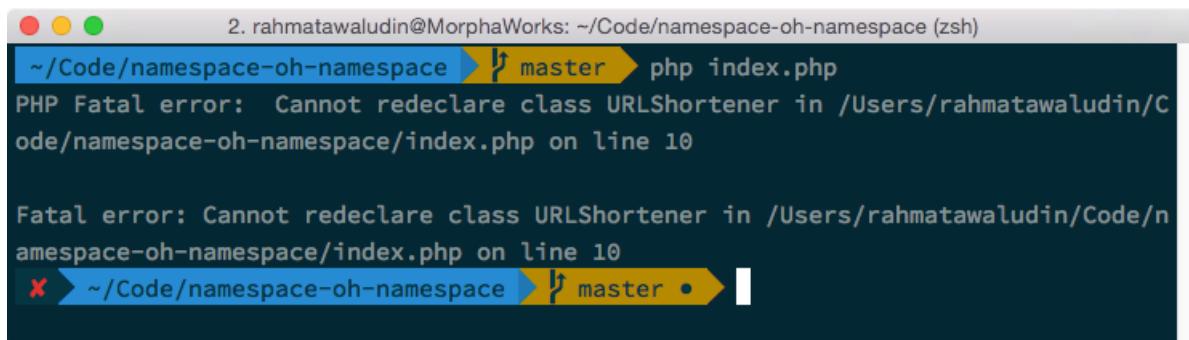
```
<?php
// buatan kita
class URLShortener {
    public function __construct() {
        echo "Membuat shortener...";
    }
}

// dari Bitly
class URLShortener {
```

```
public function __construct() {
    echo "Membuat shortener dengan API Bitly...";
}

$shortener = new URLShortener();
```

Jika dijalankan, akan muncul error :



The screenshot shows a terminal window with the following output:

```
2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)
~/Code/namespace-oh-namespace ➜ master ➜ php index.php
PHP Fatal error:  Cannot redeclare class URLShortener in /Users/rahmatawaludin/C
ode/namespace-oh-namespace/index.php on line 10

Fatal error: Cannot redeclare class URLShortener in /Users/rahmatawaludin/Code/n
amespace-oh-namespace/index.php on line 10
✖ ➜ ~/Code/namespace-oh-namespace ➜ master •
```

Tidak bisa membuat class yang sama

Dulu, cara mengakali masalah seperti ini adalah dengan membuat “virtual namespace” dengan menambahkan prefix di nama Class:

~/Code/namespace-oh-namespace/index.php

```
<?php
// buatan kita
class Gue_URLShortener {
    public function __construct() {
        echo "Membuat shortener...";
    }
}

// dari Bitly
class Bitly_URLShortener {
    public function __construct() {
        echo "Membuat shortener dengan API Bitly...";
    }
}

$shortener = new Gue_URLShortener();
```

Masih pakai yang seperti ini untuk membuat *virtual namespace*? Hebat! Berarti Anda seorang developer PHP *veteran*. Hehehe..

Betul. Solusi ini memang berhasil, tapi membuat code kita terlihat tidak indah dan merepotkan untuk melakukan *maintenance code*.



Jangan salah paham dengan penggunaan underscore sebagai *virtual namespace* dengan underscore sebagai *snake case*.

Virtual namespace digunakan untuk mengelompokkan class sebelum ada fitur namespace di PHP, sedangkan *snake case* digunakan jika kita menamai variable yang terdiri dari beberapa kata misalnya `nama_siswa`, `gelas_merah`, `kelas_a`, dll. Penggunaan underscore sebagai *snake case*, masih sangat umum digunakan saat ini.

Solusi yang lebih elegan, semenjak PHP 5.3, adalah dengan menggunakan namespace. Untuk menambahkan namespace kita cukup menambah keyword `namespace <nama_namespace>`. Sehingga syntax yang tadi akan berubah menjadi :

`~/Code/namespace-oh-namespace/index.php`

```
<?php
// buatan kita
namespace Gue;
class URLShortener {
    public function __construct() {
        echo "Membuat shortener...\n";
    }
}

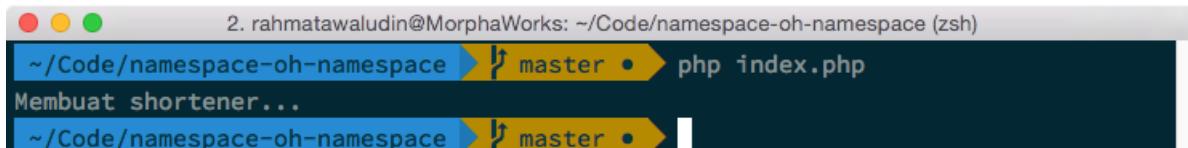
// dari Bitly
namespace Bitly;
class URLShortener {
    public function __construct() {
        echo "Membuat shortener dengan API Bitly...\n";
    }
}
```

Untuk membuat object dari Class yang berada di dalam namespace, kita gunakan \ untuk memisahkan nama namespace dan class. Seperti ini :

~/Code/namespace-oh-namespace/index.php

```
....  
$shortener = new \Gue\URLShortener();
```

Jika dijalankan, syntax ini pasti berhasil:



A terminal window titled "2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)". The command entered is "php index.php". The output shows the process: "Membuat shortener..." followed by a success message: "Berhasil menggunakan namespace".

Jika kita tambahkan shortener Bitly :

~/Code/namespace-oh-namespace/index.php

```
....  
$shortener = new \Gue\URLShortener();  
$bitly = new \Bitly\URLShortener();
```

Maka, syntax ini akan tetap berjalan :

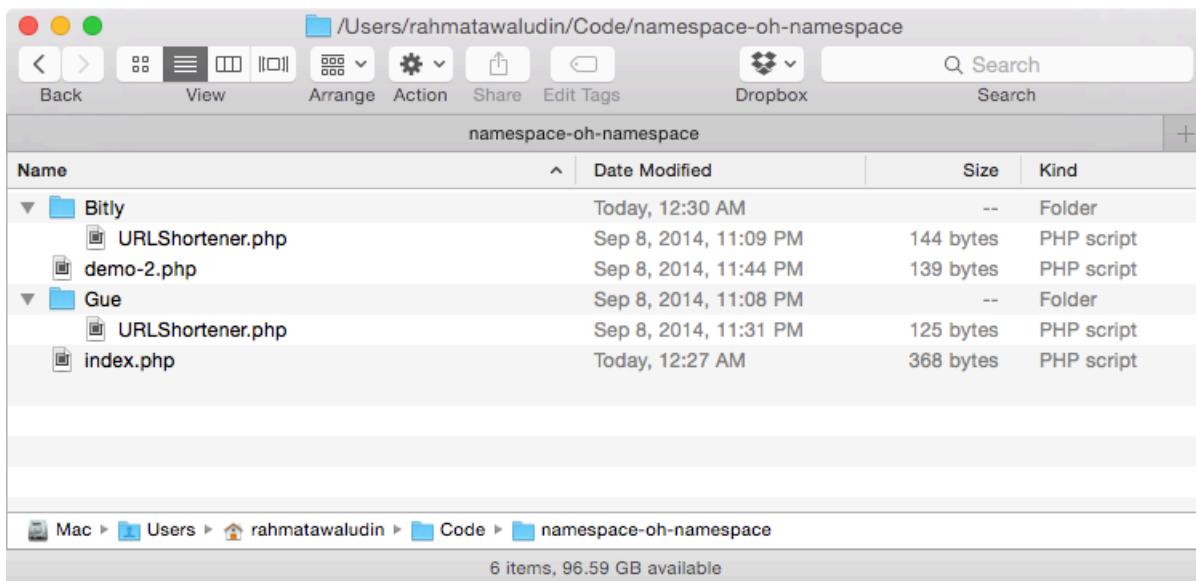


A terminal window titled "2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)". The command entered is "php index.php". The output shows two processes: "Membuat shortener..." and "Membuat shortener dengan API Bitly...". Both processes are completed successfully, indicated by the status bar message "Berhasil membuat dua namespace".

Terlihat disini, menggunakan namespace akan memudahkan kita dalam mengelompokkan Class yang sama dalam satu namespace.

Struktur File dan Autoloader

Manfaat lain dari menggunakan namespace adalah kita dapat membuat struktur folder file di project menjadi lebih rapi. Misalnya kita buat struktur folder seperti ini:



Struktur folder dengan namespace

Dengan konten masing-masing :

~/Code/namespace-oh-namespace/Gue/URLShortener.php

```
<?php
namespace Gue;
class URLShortener {
    public function __construct() {
        echo "Membuat shortener...\\n";
    }
}
```

~/Code/namespace-oh-namespace/Bitly/URLShortener.php

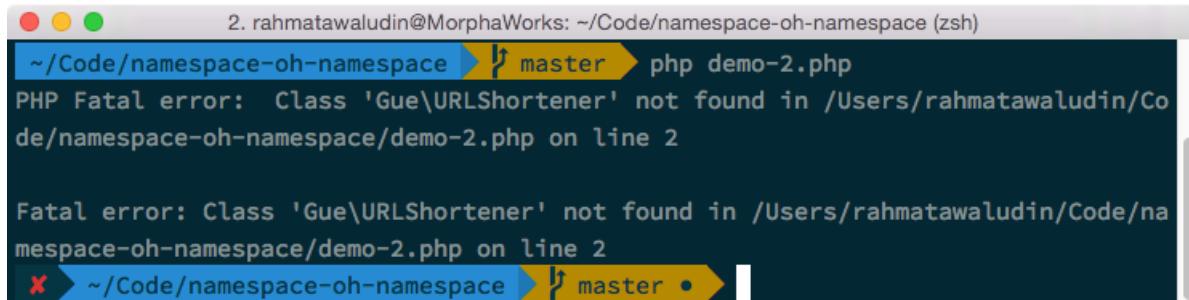
```
<?php
namespace Bitly;
class URLShortener {
    public function __construct() {
        echo "Membuat shortener dengan API Bitly...\\n";
    }
}
```

Jika kita coba jalankan demo dengan syntax berikut :

~/Code/namespace-oh-namespace/demo-2.php

```
<?php  
$shortener = new \Gue\URLShortener();  
$bitly = new \Bitly\URLShortener();
```

Akan muncul error :



```
2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)  
~/Code/namespace-oh-namespace ➜ master ➜ php demo-2.php  
PHP Fatal error: Class 'Gue\URLShortener' not found in /Users/rahmatawaludin/Co  
de/namespace-oh-namespace/demo-2.php on line 2  
  
Fatal error: Class 'Gue\URLShortener' not found in /Users/rahmatawaludin/Code/na  
mespace-oh-namespace/demo-2.php on line 2  
✖ ➜ ~/Code/namespace-oh-namespace ➜ master •
```

Error karena Class belum diinclude

Ini terjadi karena kita belum meng-*include* Class yang bersangkutan. Solusinya, kita bisa meng-*include* Class tersebut satu-persatu atau menggunakan autoloader yang telah kita pelajari sebelumnya.

Kabar baiknya, fungsi `spl_autoload_register()` dapat dijalankan tanpa parameter jika kita hendak meng-*include* class yang lokasinya sesuai dengan namespace. Ubah demo menjadi :

~/Code/namespace-oh-namespace/demo-2.php

```
<?php  
spl_autoload_register();  
$shortener = new \Gue\URLShortener();  
$bitly = new \Bitly\URLShortener();
```

Dan outputnya :



```
2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)  
~/Code/namespace-oh-namespace ➜ master ➜ php demo-2.php  
Membuat shortener...  
Membuat shortener dengan API Bitly...  
~/Code/namespace-oh-namespace ➜ master •
```

Berhasil membuat autoload di namespace tanpa parameter

Ajaib! Inilah kelebihan autoload jika digabungkan dengan namespace. kita tidak perlu menjelaskan bagaimana sebuah Class akan di load, karena autoloader PHP akan secara default mencari Class sesuai dengan namespacenya pada folder yang ada.

Misalnya kita tambahkan Class yang lebih dalam di Bitly\Auth\Login.php :

~/Code/namespace-oh-namespace/Bitly/Auth/Login.php

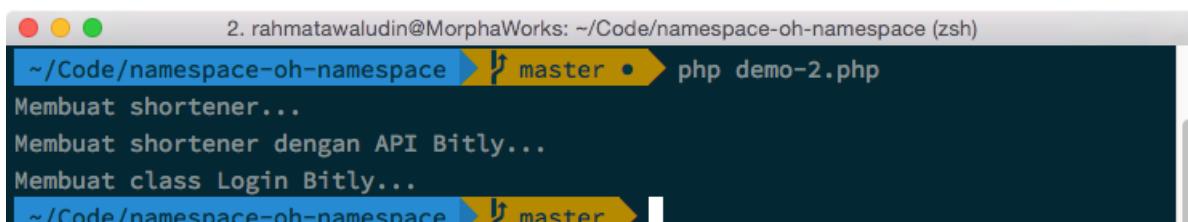
```
<?php
namespace Bitly\Auth;
class Login {
    public function __construct() {
        echo "Membuat class Login Bitly...\n";
    }
}
```

Karena class Auth ini berada di dalam folder Bitly\Auth, maka kita tulis namespace Bitly\Auth (baris ke-2). Jika didemokan dengan:

~/Code/namespace-oh-namespace/demo-2.php

```
<?php
spl_autoload_register();
$shortener = new \Gue\URLShortener();
$bitly = new \Bitly\URLShortener();
$login = new \Bitly\Auth\Login();
```

Maka, demo ini akan tetap berjalan :



```
2. rahmatawaludin@MorphaWorks: ~/Code/namespace-oh-namespace (zsh)
~/Code/namespace-oh-namespace > master • php demo-2.php
Membuat shortener...
Membuat shortener dengan API Bitly...
Membuat class Login Bitly...
~/Code/namespace-oh-namespace > master
```

Autoloader memudahkan loading class di dalam namespace

Framework Laravel, mulai versi 4.3, menggunakan namespace ini secara native. Berbagai fitur dasar di Laravel di implementasikan di dalam scope sebuah namespace. Semoga dengan memahami namespace ini, membuat proses belajar Laravel kita lebih mulus. Sip.



Source code dari latihan ini bisa didapat di
[https://github.com/rahmatawaludin/namespace-oh-namespace¹⁹](https://github.com/rahmatawaludin/namespace-oh-namespace)

¹⁹<https://github.com/rahmatawaludin/namespace-oh-namespace/commits/master>

PHP5 Magic Method

Suatu ketika saya pernah menyaksikan teman sekantor melakukan sulap. Dia membawa segelas air teh, diperlihatkannya kepada kami bahwa air teh itu biasa-biasa saja. Setelah berbasa-basi panjang lebar, teman saya itupun mengocok air itu dengan sebuah sedotan. Seperti anak kecil, kami perhatikan setiap kocekannya di gelas itu. Dan, disinilah keajaiban muncul, air teh itu berubah menjadi bening!

Kami pun serentak memberikan tepukan. Luar biasa, koq bisa ya?!

Karena dia teman saya, akhirnya saya diberi tahu rahasianya. Ternyata, sedotan yang digunakan untuk mengocok air teh itu telah dicelupkan terlebih dahulu ke cairan pemutih pakaian. Jadi, umumnya cairan apapun kalau kena tetesan pemutih pakaian, akan menjadi bening. *Owala.. gitu toh caranya.. -_-*

Begitulah sulap, kalau kita belum tahu triknya, semuanya terasa begitu ajaib. Tapi, kalau sudah tahu triknya, jadi biasa-biasa aja. Begitupun dengan topik Magic Method di PHP, ini merupakan teknik yang sering digunakan berbagai framework untuk membuat beberapa *magic* dalam frameworknya. Dan, framework Laravel, tak luput dari menggunakan fitur ini untuk membuat *magic* dalam core Laravel.

Jadi, Magic Method itu apa mas?

Magic method di PHP, adalah fungsi-fungsi khusus di PHP yang diawali dengan __ (dua underscore). Fungsi-fungsi tersebut yaitu __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(), __invoke(), __set_state(), __clone() and __debugInfo().

Semua fungsi diatas, akan dijalankan secara otomatis ketika sebuah event tertentu berjalan. Misalnya, fungsi __construct() berjalan ketika kita membuat object. Contohnya:

`~/Code/rahasia-magic-method/contoh-construct.php`

```
<?php
class Draw {
    public function __construct() {
        echo "method __construct hadir mas..\n";
    }
}

$draw = new Draw();
```

Jika dijalankan :

```
2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)
~/Code/rahasia-magic-method master php contoh-construct.php
method __construct hadir mas...
~/Code/rahasia-magic-method master
```

Method `__construct` dipanggil ketika membuat object

Terlihat disini, method `__construct()` telah dipanggil ketika kita membuat object dari class `Draw` meskipun kita tidak secara eksplisit memanggil method tersebut.

Daftar lengkap dari semua event untuk tiap magic method dapat dibaca di [php.net²⁰](http://php.net/manual/en/language.oop5.magic.php).

Magic Methods

The function names `__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` and `__debugInfo()` are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

Caution PHP reserves all function names starting with `__` as magical. It is recommended that you do not use function names with `__` in PHP unless you want some documented magic functionality.

`__sleep()` and `__wakeup()`

```
public array __sleep ( void )
```

Halaman dokumentasi magic method

Agar lebih paham, mari kita buat contoh implementasi magic method `__get()` dan `__set()`. Method `__get()` berjalan ketika kita mencoba **mengakses** atribut dari class yang di tidak di set `public`. Sedangkan method `__set()` berjalan ketika kita akan **mengisi** atribut yang tidak di set `public`.

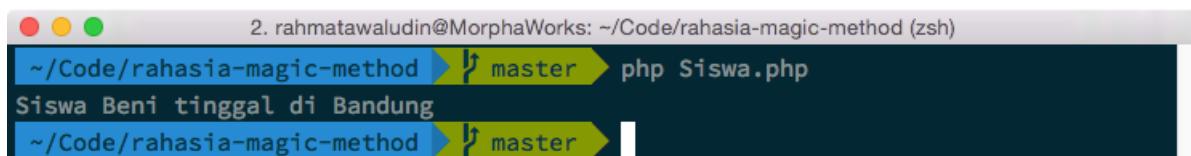
Biasanya, dalam mengakses atribut yang tidak di set `public` pada suatu Class kita menggunakan getter dan setter kan? Misalnya seperti syntax ini:

²⁰ <http://php.net/manual/en/language.oop5.magic.php>

~/Code/rahasia-magic-method/Siswa.php

```
<?php
class Siswa {
    private $nama;
    private $alamat;
    public function setNama($val) {
        $this->nama = $val;
    }
    public function getNama() {
        return $this->nama;
    }
    public function setAlamat($val) {
        $this->alamat = $val;
    }
    public function getAlamat() {
        return $this->alamat;
    }
}
$siswa = new Siswa();
$siswa->setNama("Beni");
$siswa->setAlamat("Bandung");
echo "Siswa " . $siswa->getNama() . " tinggal di " . $siswa->getAlamat() . "\n";
```

Jika dijalankan :



The screenshot shows a terminal window with the following content:

```
2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)
~/Code/rahasia-magic-method > master > php Siswa.php
Siswa Beni tinggal di Bandung
~/Code/rahasia-magic-method > master >
```

Menggunakan getter dan setter

Terlihat di syntax diatas, kita membuat getter dan setter untuk tiap atribut dari class Siswa (\$nama dan \$alamat). Teknik ini, meskipun bekerja, kurang efisien dari sisi coding. Karena, kita mengulang syntax yang hampir sama untuk semua setter dan getter. Dan, ketika suatu saat kita menambah atribut baru, kita harus membuat getter dan setter untuk atribut itu secara manual.

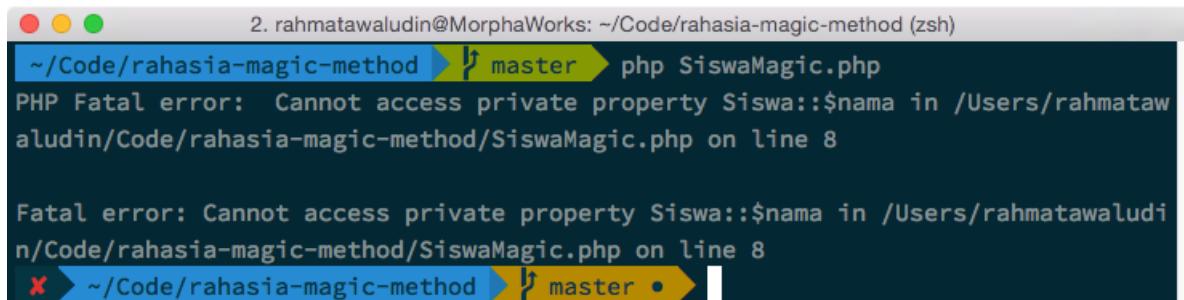
Cara yang lebih elegan, adalah dengan menggunakan magic method `__get()` dan `__set()`. Misalnya jika kita ubah syntax diatas menjadi :

~/Code/rahasia-magic-method/SiswaMagic.php

```
<?php
class Siswa {
    private $nama;
    private $alamat;
}

$siswa = new Siswa();
$siswa->nama = "Beni";
$siswa->alamat = "Bandung";
echo "Siswa " . $siswa->nama . " tinggal di " . $siswa->alamat . "\n";
```

Akan muncul error :



The screenshot shows a terminal window with the following output:

```
2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)
~/Code/rahasia-magic-method > php SiswaMagic.php
PHP Fatal error:  Cannot access private property Siswa::$nama in /Users/rahmataw
aludin/Code/rahasia-magic-method/SiswaMagic.php on line 8

Fatal error: Cannot access private property Siswa::$nama in /Users/rahmatawaludi
n/Code/rahasia-magic-method/SiswaMagic.php on line 8
x ~/Code/rahasia-magic-method >
```

Tidak bisa mengakses private atribut

Ini terjadi, karena atribut \$nama dan \$alamat di set private. Mari kita tambahkan magic method :

~/Code/rahasia-magic-method/SiswaMagic.php

```
<?php
class Siswa {
    private $nama;
    private $alamat;
    public function __get($atribute)
    {
        if (property_exists($this, $atribute)) {
            return $this->$atribute;
        }
    }
    public function __set($atribut, $value)
    {
        if (property_exists($this, $atribut)) {
```

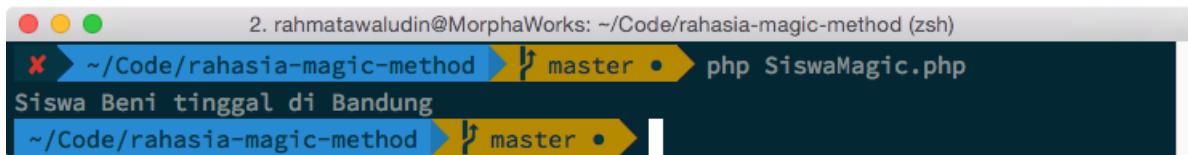
```

        $this->$atribut = $value;
    }
}
}

$siswa = new Siswa();
$siswa->nama = "Beni";
$siswa->alamat = "Bandung";
echo "Siswa " . $siswa->nama . " tinggal di " . $siswa->alamat . "\n";

```

Maka, secara *ajaib* error itu lenyap:



```

2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)
x ~/Code/rahasia-magic-method master • php SiswaMagic.php
Siswa Beni tinggal di Bandung
~/Code/rahasia-magic-method master •

```

Berhasil menggunakan magic method untuk setter dan getter

Keren kan? Ini bisa dilakukan karena pada method `__get()` kita menggunakan fungsi `property_exists()` untuk mengecek atribut yang sedang diakses ke class yang aktif (`Siswa`). Jika ada, ia akan meneruskan untuk mengambil nilai dari atribut tersebut.

Begitupun yang kita lakukan pada method `__set()`, kita menggunakan fungsi `property_exists()` untuk mengecek apakah atribut tersebut ada, jika ada ia akan meneruskan untuk mengeset nilai dari atribut tersebut.

Yang lebih menarik dari penggunaan `__get()` dan `__set()` ini adalah ketika kita menambahkan atribut baru, kita tidak perlu membuat getter dan setter lagi. Contoh :

~/Code/rahasia-magic-method/SiswaMagic.php

```

<?php
class Siswa {
    private $nama;
    private $alamat;
    private $berat;
    private $tinggi;
    ...

}

$siswa = new Siswa();
$siswa->nama = "Beni";
$siswa->alamat = "Bandung";
$siswa->berat = 67;

```

```
$siswa->tinggi = 180;  
echo "Siswa " . $siswa->nama . " tinggal di " . $siswa->alamat . "\n";  
echo "Dengan berat " . $siswa->berat . " Kg dan tinggi " . $siswa->tinggi . " cm\n";\n";
```

Maka syntax ini akan tetap berjalan :



```
2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)  
~/Code/rahasia-magic-method [master] $ php SiswaMagic.php  
Siswa Beni tinggal di Bandung  
Dengan berat 67 Kg dan tinggi 180 cm
```

Tidak perlu membuat getter dan setter untuk atribut baru

Lebih jauh lagi, misalnya kita ingin memberikan method tersendiri untuk setiap getter atau setter dapat dilakukan dengan cara yang mudah. Dalam contoh ini, misalnya setiap kali kita mengambil atribut `tinggi` kita ingin menambahkan teks `cm` dan ketika mengambil atribut `berat` kita ingin menambahkan teks `kg`. Maka, kita cukup merubah method `__get()` dan menambah dua method (dalam contoh ini method `getBerat()` dan `getTinggi()`):

~/Code/rahasia-magic-method/SiswaMagic.php

```
class Siswa {  
    ...  
    public function __get($attribute)  
    {  
        if (property_exists($this, $attribute)) {  
            if (method_exists($this, 'get' . $attribute)) {  
                return call_user_func([$this, 'get' . $attribute]);  
            } else {  
                return $this->$attribute;  
            }  
        }  
        ...  
    }  
    public function getBerat()  
    {  
        return $this->berat . " Kg";  
    }  
    public function getTinggi()  
    {  
        return $this->tinggi . " Cm";  
    }  
}
```

```
echo "Dengan berat " . $siswa->berat . " dan tinggi " . $siswa->tinggi . "\n";
";
}
```

Dan hasilnya akan tetap berhasil :



```
2. rahmatawaludin@MorphaWorks: ~/Code/rahasia-magic-method (zsh)
~/Code/rahasia-magic-method ➜ master • php SiswaMagic.php
Siswa Beni tinggal di Bandung
Dengan berat 67 Kg dan tinggi 180 cm
~/Code/rahasia-magic-method ➜ master •
```

Menggunakan custom get method

Terlihat disini, kita menggunakan syntax `return call_user_func([$this, 'get' . $attribute])` untuk memanggil fungsi `get` untuk variable yang dipanggil jika fungsi tersebut ada.

Teknik diatas yang menggunakan custom getter, kalau dalam framework Laravel, dinamakan *accessor*. Sedangkan kalau kita menggunakan custom setter dinamakan *mutator*.

Salah satu core Laravel yaitu **Facade** menggunakan magic method ini di dalam log-icnya. Akan kita bahas di pembahasan tentang **Arsitektur Laravel**. Tetep semangat, Sip!



Source code dari latihan ini bisa didapat di
[https://github.com/rahmatawaludin/rahasia-magic-method²¹](https://github.com/rahmatawaludin/rahasia-magic-method)

PHP5 Reflection

Reflection dalam bahasa yang sederhana adalah kemampuan sebuah program untuk mengidentifikasi dirinya sendiri. Reflection tuh ibarat cermin.

Dengan cermin kita dapat mengetahui bentuk hidung, mulut, alis, mata, dan telinga. Dengan cermin, kita dapat melihat rambu kita yang belum rapi (tentunya tidak berlaku kalau kita botak). Dengan cermin, kita dapat *merapihkan* wajah kita, biar ganteng maksimal! :D

Balik lagi ke Reflection. Fitur ini seperti cermin yang memungkinkan class melihat method apa saja yang dimilikinya, property apa saja yang dimilikinya, dsb.

Reflection memang akan jarang kita gunakan di coding aplikasi. Namun, memahami fitur ini merupakan gerbang masuk ke tingkat yang lebih tinggi di pemrograman PHP.

²¹ <https://github.com/rahmatawaludin/rahasia-magic-method/commits/master>

Class/Object Functions

Sebelum masuk ke Reflection Class, kita akan mempelajari fitur Class/Object Function di PHP. Fitur ini mirip dengan Reflection, hanya saja kita tidak perlu membuat Reflection Object secara eksplisit.

Beberapa contoh dari Object Function diantaranya:

- `get_class` untuk mengambil nama class dari object
- `get_class_methods` untuk mengambil semua method yang ada pada object
- `get_parent_class` untuk mengambil parent class dari object

Masih banyak yang lain lagi, silahkan cek di [dokumentasi resmi PHP²²](http://php.net/manual/en/ref.classobj.php)

The screenshot shows a web browser displaying the PHP Manual page for 'Classes/Object Functions'. The URL in the address bar is <http://php.net/manual/en/ref.classobj.php>. The page has a dark blue header with the 'php' logo and navigation links for 'Downloads', 'Documentation', 'Get Involved', and 'Help'. A search bar is also present in the header. The main content area features a purple header 'Classes/Object Functions' followed by a 'Table of Contents' section with a list of functions. To the right, there is a sidebar with sections for 'Introduction', 'Installing/Configuring', 'Predefined Constants', 'Examples', and a link to '» Classes/Object Functions'. A note at the bottom of the sidebar states: 'Note: This page is part of the legacy API documentation. It is recommended to use the new reflection API instead.' The footer of the page includes links to 'Examples', '__autoload', and other related topics.

Classes/Object Functions

Table of Contents

- [__autoload](#) – Attempt to load undefined class
- [call_user_method_array](#) – Call a user method given with an array of parameters [deprecated]
- [call_user_method](#) – Call a user method on an specific object [deprecated]
- [class_alias](#) – Creates an alias for a class
- [class_exists](#) – Checks if the class has been defined
- [get_called_class](#) – the "Late Static Binding" class name
- [get_class_methods](#) – Gets the class methods' names
- [get_class_vars](#) – Get the default properties of the class
- [get_class](#) – Returns the name of the class of an object
- [get_declared_classes](#) – Returns an array with the name of the defined classes
- [get_declared_interfaces](#) – Returns an array of all declared interfaces

Halaman dokumentasi Class/Object Functions

Mari kita buat contoh untuk masing-masing fungsi yang saya sebut diatas, kita akan menganalisis class Twitter yang merupakan child dari class Social.

²²<http://php.net/manual/en/ref.classobj.php>

~/Code/reflection-ganteng-maksimal/Social.php

```
<?php
class Social {
    private $username;
    public function __construct($username) {
        $this->username = $username;
    }
    public function username() {
        return $this->username;
    }
}
```

~/Code/reflection-ganteng-maksimal/Twitter.php

```
<?php
include_once "Social.php";
class Twitter extends Social {
    private $tweet;
    public function __construct($username, $tweet) {
        $this->tweet = $tweet;
        parent::__construct($username);
    }
    public function tweet() {
        return $this->tweet;
    }
}
```

Mari kita cek dengan syntax ini :

~/Code/reflection-ganteng-maksimal/cek-twitter.php

```
<?php
include_once "Twitter.php";
$twitter = new Twitter("Mario", "Menikahi orang yang dicintai adalah kemungkinan\
. Tapi, mencintai orang yang dinikahi adalah kewajiban. Itu.");
echo "Akun twitter : " . $twitter->username() . "\n";
echo "Tweet : " . $twitter->tweet() . "\n";
```

Pastikan tidak ada yang error:

```
~/Code/reflection-ganteng-maksimal master • php cek-twitter.php
Akun twitter : Mario
Tweet : Menikahi orang yang dicintai adalah kemungkinan. Tapi, mencintai orang y
ang dinikahi adalah kewajiban. Catet.
```

Class Twitter berjalan

Mari kita coba fungsi `get_class` untuk mengecek class dari object `$twitter`:

`~/Code/reflection-ganteng-maksimal/contoh-object-functions.php`

```
<?php
include_once "Twitter.php";
$twNagita = new Twitter("Nagita", "Pria sejati datang menemui orang tua gadis pu\
jaannya bukan mengajak si gadis ke tempat2 sepi. Itu.");
echo "Class dari \$twNagita adalah : " . get_class($twNagita) . "\n";
```

Jika dijalankan terlihat, fungsi `get_class()` berhasil menampilkan class dari object `$twNagita`:

```
~/Code/reflection-ganteng-maksimal master • php contoh-object-functions.p
hp
Class dari $twNagita adalah : Twitter
```

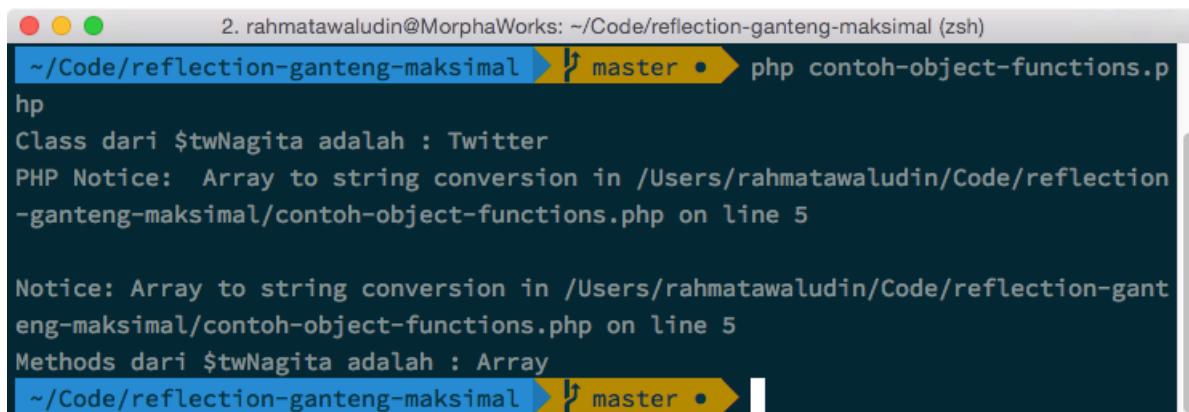
Fungsi `get_class()` berhasil menampilkan nama class

Lanjut ke fungsi `get_class_methods`, tambah baris ini di akhir file tadi:

`~/Code/reflection-ganteng-maksimal/contoh-object-functions.php`

```
.....
echo "Methods dari \$twNagita adalah : " . get_class_methods($twNagita) . "\n";
```

Jika dijalankan:



```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal master • php contoh-object-functions.php
hp
Class dari $twNagita adalah : Twitter
PHP Notice: Array to string conversion in /Users/rahmatawaludin/Code/reflection-ganteng-maksimal/contoh-object-functions.php on line 5

Notice: Array to string conversion in /Users/rahmatawaludin/Code/reflection-ganteng-maksimal/contoh-object-functions.php on line 5
Methods dari $twNagita adalah : Array
~/Code/reflection-ganteng-maksimal master •
```

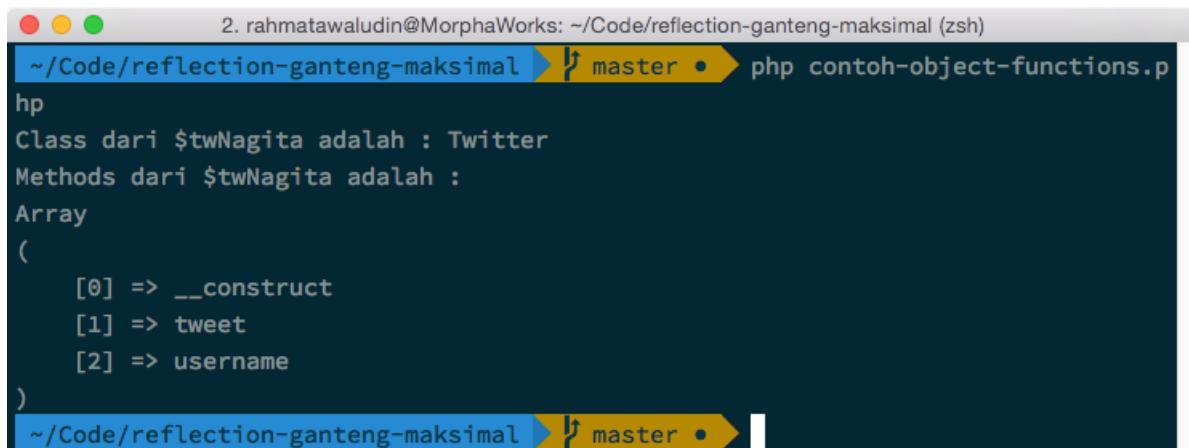
Error karena hasil get_class_methods adalah array

Uuupppss.. ternyata hasil dari fungsi `get_class_methods` adalah array, mari kita ubah baris terakhir menjadi :

~/Code/reflection-ganteng-maksimal/contoh-object-functions.php

```
.....
echo "Methods dari \$twNagita adalah : \n";
print_r(get_class_methods($twNagita));
```

Kini, kita dapat melihat method apa saja yang ada dimiliki method `$twNagita`:



```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal master • php contoh-object-functions.php
hp
Class dari $twNagita adalah : Twitter
Methods dari $twNagita adalah :
Array
(
    [0] => __construct
    [1] => tweet
    [2] => username
)
~/Code/reflection-ganteng-maksimal master •
```

get_class_methods berhasil menampilkan method yang dimiliki object

Terakhir, mari kita cek parent class dari object `$twNagita` dengan `get_parent_class`:

~/Code/reflection-ganteng-maksimal/contoh-object-functions.php

```
....  
echo "Parent class dari \$twNagita` : " . get_parent_class($twNagita) . "\n";
```

Kini, kita bisa mengetahui parent class dari object \$twNagita:

```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)  
~/Code/reflection-ganteng-maksimal master • php contoh-object-functions.p  
hp  
Class dari $twNagita adalah : Twitter  
Methods dari $twNagita adalah :  
Array  
(  
    [0] => __construct  
    [1] => tweet  
    [2] => username  
)  
Parent class dari $twNagita` : Social  
~/Code/reflection-ganteng-maksimal master •
```

Berhasil menampilkan parent class dengan `get_parent_class`

Oke, mungkin sedikit bingung apa manfaat dari semua method ini. Simpan dulu kebingungannya, kita lanjut dulu pembahasan tentang Reflection API. Di pembahasan ini kita akan membuat contoh real penggunaan fitur reflection ini. Lanjut...

Reflection API

Fitur Reflection di PHP 5 akan sangat bermanfaat dalam melakukan *reverse engineering* sebuah class, interface, fungsi dan methods. Jika kita melihat ke dokumentasi resmi, ada banyak sekali method di Reflection Class ini. Saya akan mencontohkan beberapa method kemudian membuat sample implementasi Reflection API ini di lapangan.

Secara garis besar Reflection API ini dikelompokan beberapa class, saya akan jelaskan sekilas beberapa yang akan kita gunakan:

- `ReflectionClass` untuk mengidentifikasi class
- `ReflectionMethod` untuk mengidentifikasi method pada class
- `ReflectionObject` untuk mengidentifikasi object
- `ReflectionParameter` untuk mengidentifikasi parameter pada method
- `ReflectionProperty` untuk mengidentifikasi property pada class

Untuk fitur lain dari Reflection ini, silahkan kunjungi halaman [dokumentasi Reflection API](#)²³.

²³<http://php.net/manual/en/book.reflection.php>

The screenshot shows a web browser displaying the PHP Manual page for the 'Reflection' extension. The URL is <http://php.net/manual/en/book.reflection.php>. The page title is 'Reflection'. On the left, there's a sidebar with navigation links like 'Introduction', 'Installing/Configuring', 'Requirements', 'Installation', 'Runtime Configuration', 'Resource Types', 'Predefined Constants', 'Examples', 'Extending', 'Reflection — The Reflection class', 'Reflection::export — Exports', 'Reflection::getModifierNames — Gets modifier names', and 'ReflectionClass — The ReflectionClass class'. On the right, there's a sidebar titled 'Variable and Type Related Extensions' with links to 'Arrays', 'Classes/Objects', 'Classkit', 'Ctype', 'Filter', 'Function Handling', 'Quickhash', and '» Reflection'. At the top right, there are links for 'Edit' and 'Report a Bug'. The main content area contains the detailed documentation for the Reflection class.

Dokumentasi Reflection API

Untuk mendemokan Reflection API, mari kita buat beberapa class berikut:

~/Code/reflection-ganteng-maksimal/Penulis.php

```
<?php
class Penulis {
    private $nama;
    private $alamat;
    public function __construct($nama="Seseorang", $alamat="Alamat Penulis") {
        $this->nama = $nama;
        $this->alamat = $alamat;
    }
    public function nama() {
        return $this->nama;
    }
    public function alamat() {
        return $this->alamat;
    }
}
```

Syntax diatas membuat class penulis dengan dua properti \$nama dan \$alamat. Kita juga membuat dua getter untuk properti tersebut yaitu method nama() dan alamat(). Di constructor, kita menginject variable \$nama dan \$alamat dan menyediakan nilai default (nama = "Seseorang" dan alamat = "Alamat Penulis") jika tidak disediakan oleh user.

~/Code/reflection-ganteng-maksimal/Buku.php

```
<?php
abstract class Tulisan { }
interface publishable { }
interface printable { }
class Buku extends Tulisan implements publishable, printable {
    private $judul;
    private $penulis;
    public function __construct($judul = "Belum ada judul", Penulis $penulis) {
        $this->judul = $judul;
        $this->penulis = $penulis;
    }
    public function __toString() {
        return "Judul buku : " . $this->judul . "\n"
            . "Penulis : " . $this->penulis->nama() . "\n"
            . "Alamat : " . $this->penulis->alamat() . "\n";
    }
}
```

Syntax untuk class Buku ini cukup panjang, mari kita telaah satu persatu.

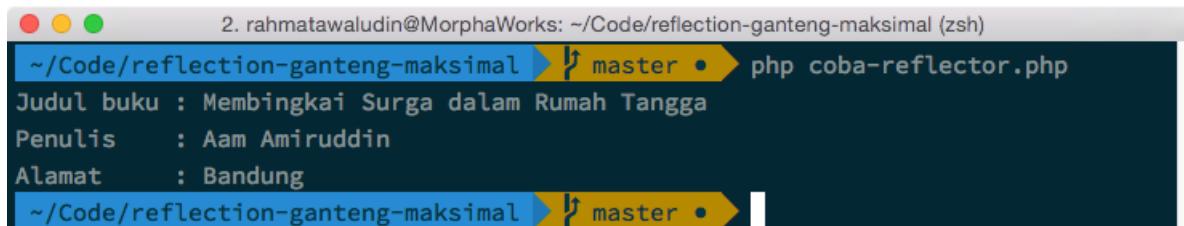
- Class Buku merupakan child dari class Tulisan.
- Class Buku mengimplementasi interface publishable dan printable
- Class buku memiliki property \$judul dan \$penulis yang diinject pada constructor. Disini terlihat class buku memiliki dependensi class Penulis.
- Terdapat magic method __toString() yang akan berjalan ketika kita menggunakan fungsi echo atau print pada object class Buku. Disini kita mencetak judul, penulis dan alamat penulis.

Untuk memastikan semua class ini berfungsi, mari kita demokan dengan syntax berikut:

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
<?php
include_once "Penulis.php";
include_once "Buku.php";
$aan = new Penulis("Aam Amiruddin", "Bandung");
$buku = new Buku("Membingkai Surga dalam Rumah Tangga", $aan);
echo $buku;
```

Kita coba :



```
2. rahmataludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal master ✘ php coba-reflector.php
Judul buku : Membingkai Surga dalam Rumah Tangga
Penulis : Aam Amiruddin
Alamat : Bandung
```

Class Buku dan Penulis berjalan

Sip. Class Buku dan Penulis telah berjalan. Terlihat disini, kita berhasil menggunakan echo pada object \$buku. Ini bisa dilakukan karena kita telah mempersiapkan method __toString().

Untuk mempraktekan Reflection API, silahkan hapus/komen baris echo \$buku. Kemudian tambahkan syntax berikut untuk membuat object ReflectionClass:

~/Code/reflection-ganteng-maksimal/coba-reflector.php

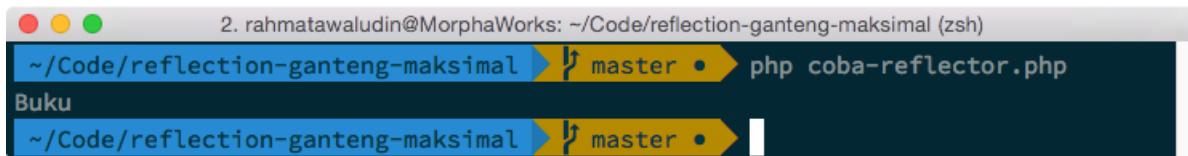
```
.....
$reflector = new ReflectionClass($buku);
```

Selanjutnya, mari kita coba beberapa method yang disediakan oleh Reflection API:

- Mengambil nama class dengan getName()

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
.....
echo $reflector->getName() . "\n";
```



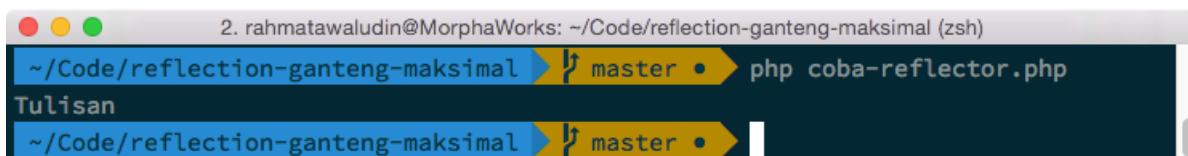
```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal > php coba-reflector.php
Buku
~/Code/reflection-ganteng-maksimal >
```

Mengambil class

- Mengambil nama parent class dengan getParentClass()

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
echo $reflector->getParentClass()->getName() . "\n";
```



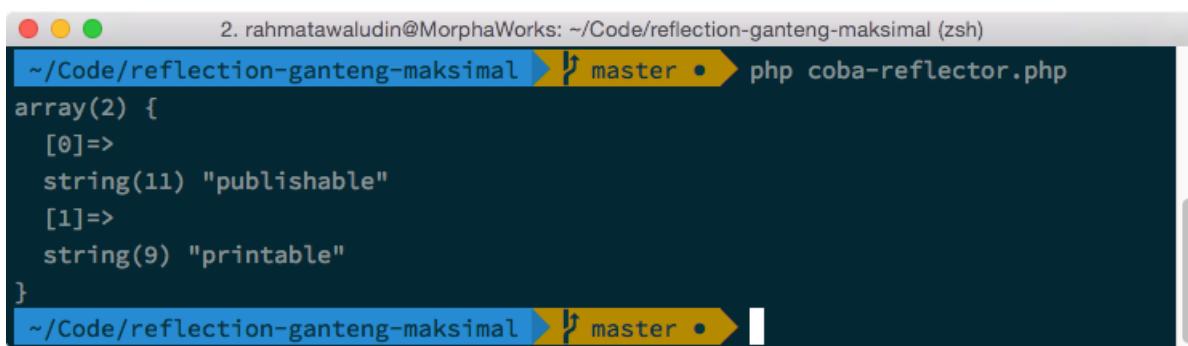
```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal > php coba-reflector.php
Tulisan
~/Code/reflection-ganteng-maksimal >
```

Mengambil parent class

- Mengecek interfaces dengan getInterfaceNames()

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
var_dump($reflector->getInterfaceNames());
```



```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal > php coba-reflector.php
array(2) {
[0]=>
string(11) "publishable"
[1]=>
string(9) "printable"
}
~/Code/reflection-ganteng-maksimal >
```

Mengambil interface

- Mengecek semua method di class dengan getMethods()

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
var_dump($reflector->getMethods());
```

The screenshot shows a terminal window with the following output:

```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)  
~/Code/reflection-ganteng-maksimal master ✘ php coba-reflector.php  
array(2) {  
    [0]=>  
    &object(ReflectionMethod)#4 (2) {  
        ["name"]=>  
        string(11) "__construct"  
        ["class"]=>  
        string(4) "Buku"  
    }  
    [1]=>  
    &object(ReflectionMethod)#5 (2) {  
        ["name"]=>  
        string(10) "__toString"  
        ["class"]=>  
        string(4) "Buku"  
    }  
}
```

Mengambil Method

Terlihat disini, output dari `getMethods` adalah object `ReflectionMethod` yang bisa kita identifikasi lebih lanjut.

- Mengecek constructor dengan `getConstructor()`

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
var_dump($reflector->getConstructor());
```

```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal master • php coba-reflector.php
object(ReflectionMethod)#4 (2) {
    ["name"]=>
    string(11) "__construct"
    ["class"]=>
    string(4) "Buku"
}
~/Code/reflection-ganteng-maksimal master •
```

Mengambil Constructor

Terlihat disini, output dari `getConstructor()` adalah object dari `ReflectionMethod` yang bisa kita identifikasi lebih lanjut. Misalnya, kita ingin mengecek parameternya:

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
.....
$constructor = $reflector->getConstructor();
var_dump($constructor->getParameters());
```

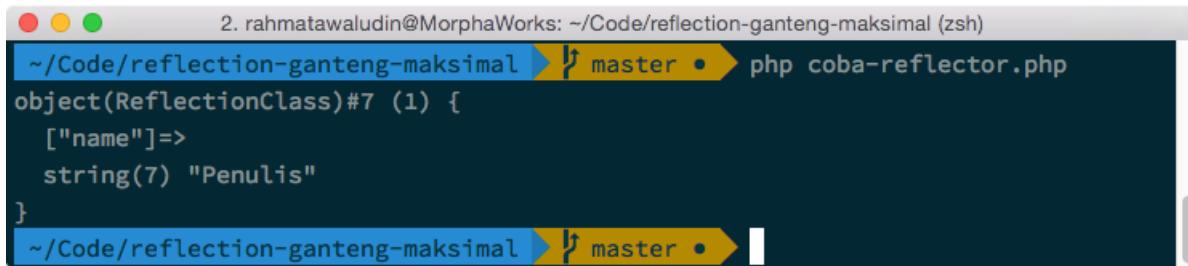
```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal master • php coba-reflector.php
array(2) {
    [0]=>
    &object(ReflectionParameter)#5 (1) {
        ["name"]=>
        string(5) "judul"
    }
    [1]=>
    &object(ReflectionParameter)#6 (1) {
        ["name"]=>
        string(7) "penulis"
    }
}
~/Code/reflection-ganteng-maksimal master •
```

Mengambil Parameter

Terlihat disini, output dari `getParameter()` pada class `ReflectionMethod` menghasilkan object `ReflectionParameter` yang bisa kita identifikasi lebih lanjut. Misalnya kita ingin mengecek class dari parameter:

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
$constructor = $reflector->getConstructor();  
$parameters = $constructor->getParameters();  
var_dump($parameters[1]->getClass());
```



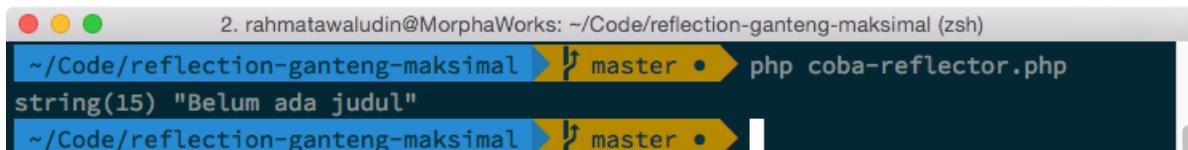
```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)  
~/Code/reflection-ganteng-maksimal ✘ master • php coba-reflector.php  
object(ReflectionClass)#7 (1) {  
    ["name"]=>  
        string(7) "Penulis"  
}  
~/Code/reflection-ganteng-maksimal ✘ master •
```

Mengambil Parameter

Kita juga bisa mengambil nilai default dari parameter misalnya dengan perintah:

~/Code/reflection-ganteng-maksimal/coba-reflector.php

```
....  
$constructor = $reflector->getConstructor();  
$parameters = $constructor->getParameters();  
var_dump($parameters[0]->getDefaultValue());
```



```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)  
~/Code/reflection-ganteng-maksimal ✘ master • php coba-reflector.php  
string(15) "Belum ada judul"  
~/Code/reflection-ganteng-maksimal ✘ master •
```

Mengambil Parameter

Terlihat disini, kita dapat mengidentifikasi *daleman* dari object \$buku ini dengan Reflection API. Fitur ini sangat powerfull untuk melakukan reverse engineering sebuah code. Tools semacam code analysis menggunakan fitur semacam ini untuk bekerja. Sip.

Mas, bisa kasih contoh coding yang lebih real?

Oke. Laravel, memiliki fitur core andalan bernama Service Container (Di Laravel 4 dinamakan IoC Container). Pembahasan yang lengkap tentang Service Container, tentunya akan saya bahas di bab **Arsitektur Laravel**. Salah satu dari fungsi Service

Container di framework Laravel adalah menginject dependensi class secara otomatis, yang biasa disebut *automatic resolution*.

Cara kerja dari automatic resolution adalah dengan menginject parameter yang dibutuhkan di method `__construct()` secara otomatis. Jika dibutuhkan object dari class lain, maka ia akan membuat object itu. Jika dibutuhkan string atau angka sebagai parameter, maka ia akan mencari nilai defaultnya. Ini penjelasan sederhana ya, masih banyak lagi sebenarnya fungsi dari *automatic resolution*.

Hasil akhirnya dia akan merubah syntax ini (perhatikan object `Buku` butuh object `Penulis` dan kita membuat kemudian menginjectnya):

Sebelum automatic resolution

```
$penulis = new Penulis();
$buku = new Buku("Judul", $penulis);
```

Menjadi ini:

Sesudah automatic resolution

```
$buku = App::make("Buku");
```

Yang perlu diperhatikan disini adalah pada contoh diatas, class `Buku` hanya membutuhkan satu dependensi yaitu class `Penulis`. Bisa jadi, di lapangan, sebuah object membutuhkan lebih dari 5 dependensi ke class lain. Nah, *automatic resolution* mampu membuat semua object untuk dependensi itu secara otomatis.

Fitur automatic resolution di Laravel cukup kompleks. Untuk memudahkan pemahaman saya akan buat contoh Automatic Resolution yang sangat sederhana.

Agar lebih paham, mari kita buat contoh sebelum menggunakan automatic resolution:

~/Code/reflection-ganteng-maksimal/sebelum-automatic-resolution.php

```
<?php
include_once "Penulis.php";
include_once "Buku.php";
$penulis = new Penulis();
$buku = new Buku("Belum ada judul",$penulis);
echo $buku;
```

```
~/Code/reflection-ganteng-maksimal [master] > php sebelum-automatic-resolution.php
Judul buku : Belum ada judul
Penulis : Seseorang
Alamat : Alamat Penulis
~/Code/reflection-ganteng-maksimal [master] >
```

Sebelum Automatic Resolution

Terlihat disini, kita menggunakan nilai default untuk membuat object `Penulis`. Object `Penulis`, meskipun menggunakan nilai default dalam pembuatannya, tetap kita buat karena kita menginject secara manual pada pembuatan object `Buku`.

Untuk membuat *automatic resolution* mari buat class baru bernama `App`:

`~/Code/reflection-ganteng-maksimal/App.php`

```

1 <?php
2 class App {
3     public static function make($class) {
4         $reflector = new ReflectionClass($class);
5         $constructor = $reflector->getConstructor();
6
7         if (is_null($constructor))
8         {
9             return new $class;
10        }
11
12        $dependencies = $constructor->getParameters();
13
14        $args = [];
15
16        foreach ($dependencies as $dependency) {
17            if (is_null($dependency->getClass())) {
18                array_push($args, $dependency->getDefaultValue());
19            } else {
20                array_push($args, self::make($dependency->getClass()->name));
21            }
22        }
23
24        return $reflector->newInstanceArgs($args);
25    }
26 }
```

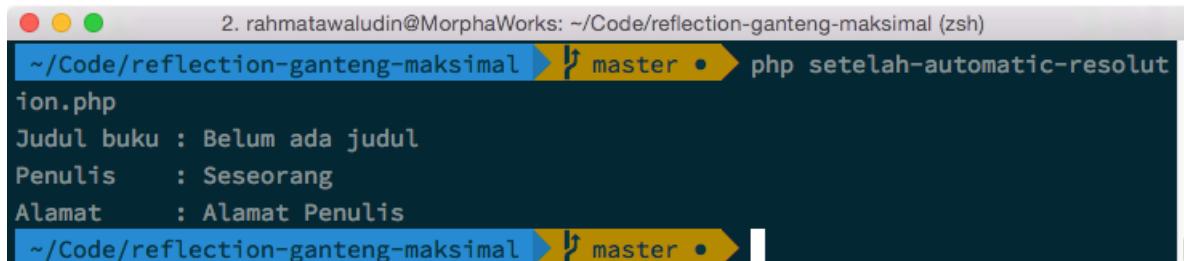
Ini merupakan Service Container minimalis kita, disini hanya terdapat satu method statis yaitu `make()`. Berikut penjelasannya:

- Method ini menerima input berupa nama class (baris 3)
- Nama class yang masuk akan dibuat ReflectionClass nya (baris 4)
- Dari Reflection Class yang didapatkan diambil constructor nya (baris 5)
- Jika constructor tidak didapatkan, artinya class tersebut tidak memiliki dependensi. Makanya langsung dibuat object baru dari class itu dengan `return new $class` (baris 9)
- Dari constructor yang didapatkan, diambil semua parameternya dan disimpan sebagai variable `$dependencies` (baris 12)
- Jika parameter tersebut bukanlah class (baris 17), maka diambil nilai defaultnya dan dimasukkan menjadi argumen di variable `$args` (baris 18)
- Jika parameternya sebuah class, maka ia akan secara rekursif memanggil kembali method `make()` di class App untuk membuat object dari Class tersebut dan memasukkannya sebagai parameter di variable `$args` (baris 20).
- Terakhir, dia membuat object dari ReflectorClass yang dibuat dengan parameter constructor berupa variable `$args` yang telah didapatkan sebelumnya (baris 24)

Mari kita cek dengan syntax:

`~/Code/reflection-ganteng-maksimal/setelah-automatic-resolution.php`

```
<?php
include_once "Penulis.php";
include_once "Buku.php";
include_once "App.php";
$buku = App::make("Buku");
echo $buku;
```



```
2. rahmatawaludin@MorphaWorks: ~/Code/reflection-ganteng-maksimal (zsh)
~/Code/reflection-ganteng-maksimal ✘ master • php setelah-automatic-resolution.php
Judul_buku : Belum ada judul
Penulis     : Seseorang
Alamat      : Alamat Penulis
~/Code/reflection-ganteng-maksimal ✘ master •
```

Berhasil membuat automatic resolution

Terlihat disini, dengan automatic resolution, kita tidak perlu membuat object Penulis yang merupakan dependensi dari object Buku. Jika aplikasi terus berkembang dan

dependensi bertambah banyak, automatic resolution akan tetap mampu melakukan inject dependensi secara otomatis!



Latihan

Cobalah buat contoh class misalnya Mobil yang membutuhkan class Bensin dan Aki. Class Aki pun membutuhkan class Listrik. Cek apakah Service Container minimalis yang telah kita buat berhasil melakukan automatic resolution.



Source code dari latihan ini bisa didapat di [https://github.com/rahmatawaludin/reflection-ganteng-maksimal²⁴](https://github.com/rahmatawaludin/reflection-ganteng-maksimal)

Composer

Jika Anda seorang developer PHP masa kini (*cieee..*), maka wajib kenal dengan Composer. Composer merupakan dependency management untuk PHP.

Apaan tuh *dependency management*? Dependency management merupakan sebuah tools yang memecahkan masalah berikut :

- Kita punya project yang butuh library. Misalnya, library buat eksport data ke Excel yaitu [PHPExcel²⁵](#) dan DomPDF buat eksport data ke PDF.
- Library-library tersebut juga butuh library lainnya. Dalam kasus ini, PHPExcel butuh library ext-xml dan DomPDF butuh phenx/php-font-lib. Kasus seperti ini, namanya dependensi. Jadi, bisa disebut PHPExcel memiliki dependensi ext-xml.
- Kita males download (install) semua library tersebut berikut dependensinya secara manual.

Cara kerja composer tuh seperti ini:

- Kita menulis library apa saja yang dibutuhkan project.
- Composer mencari versi library yang sesuai dan menginstallnya (download) secara otomatis.

Dengan composer, kita tidak usah repot-report download library PHP manual satu-persatu. Begitupun dengan proses update library, tidak usah diupdate satu-persatu, cukup ubah satu file (`composer.json`), jalankan perintah `composer update` untuk mengupdate semua library. *Kalau bisa gampang gini, ngapain dibikin susah? :D*

²⁴ <https://github.com/rahmatawaludin/reflection-ganteng-maksimal/commits/master>

²⁵ <https://github.com/PHPOffice/PHPExcel>

Instalasi

Cara menginstall composer di sistem berbasis Unix (Linux dan OSX) dan Windows cukup berbeda. Namun tenang, di buku ini, saya akan menjelaskan untuk dua OS tersebut.

Pengguna Linux. Buka terminal, jalankan perintah

```
curl -sS https://getcomposer.org/installer | php  
mv composer.phar /usr/local/bin/composer
```

Sebelum menjalankan diatas, pastikan sudah menginstall PHP.

Pengguna OSX. Buka terminal, jalankan perintah

```
brew update  
brew tap homebrew/homebrew-php  
brew tap homebrew/dupes  
brew tap homebrew/versions  
brew install php55-intl  
brew install homebrew/php/composer
```

Sebelum menjalankan diatas, pastikan sudah menginstall homebrew^a

^a<http://brew.sh>

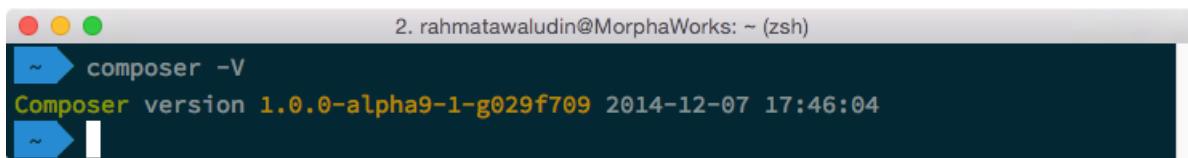
Pengguna Windows. Buka browser (beda sendiri), download installer di <https://getcomposer.org/Composer-Setup.exe>²⁶. Jalankan installer tersebut. Selesai.

Setelah menginstall composer, coba buka terminal/command prompt di Windows (Tekan Window+R ketik cmd). Lalu jalankan perintah berikut:

```
composer -V
```

Pastikan ada output seperti ini:

²⁶<https://getcomposer.org/Composer-Setup.exe>



```
2. rahmatawaludin@MorphaWorks: ~ (zsh)
~ composer -V
Composer version 1.0.0-alpha9-1-g029f709 2014-12-07 17:46:04
~
```

Sukses menginstall composer

Jika tampilan tersebut muncul, berarti kita telah **sukses** menginstall composer. Sip.

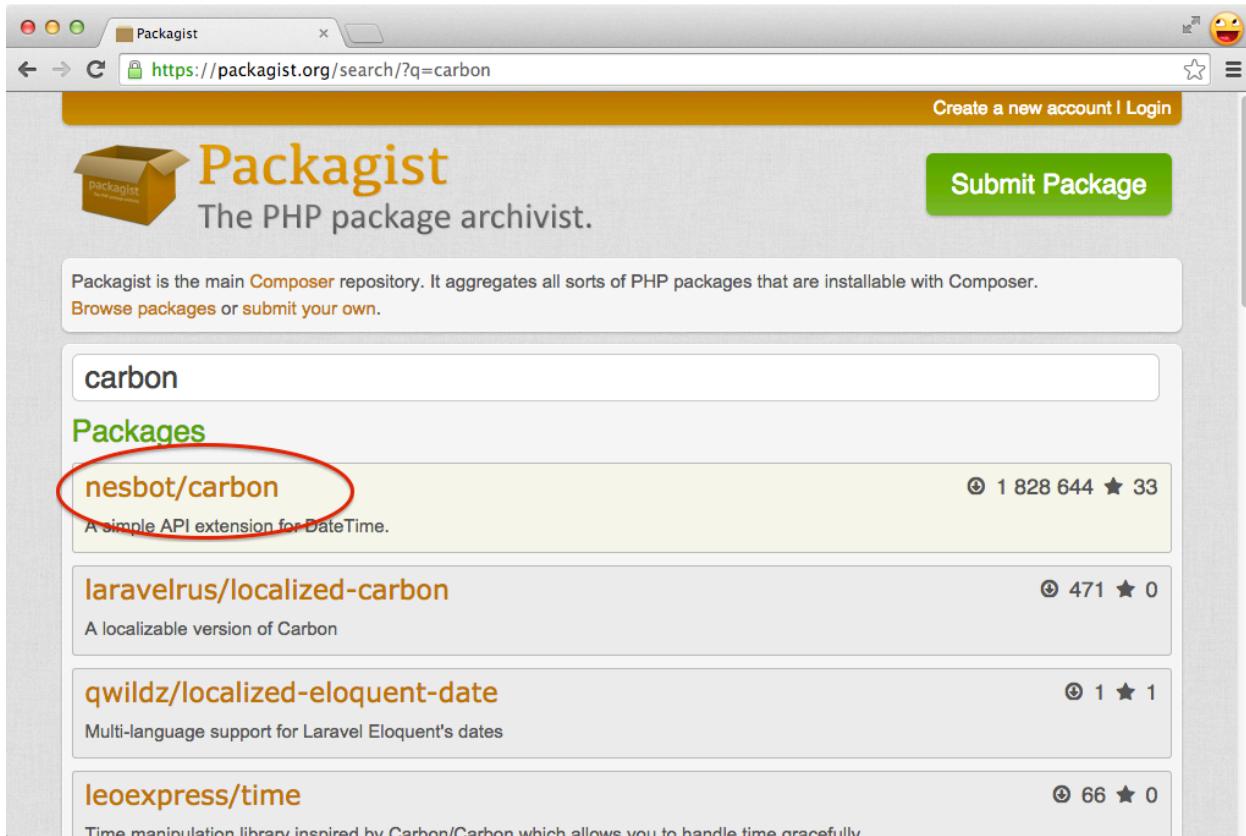
Penggunaan

Yuk kita pelajari bagaimana cara pakai makhluk bernama composer ini. Saya akan buat sample kasus dimana kita membutuhkan library [Carbon²⁷](#). Library ini digunakan untuk memanipulasi tampilan objek DateTime. Pernah melihat tampilan tanggal “2 hour ago”, “2 minutes ago” atau “A month ago” di web? Nah, itu salah satu fungsi dari Carbon.

Untuk mendapatkan package dengan Composer, pertama kita datang ke pusat package, kunjungi di [packagist.org²⁸](#). Setelah terbuka, ketikkan “carbon” di kotak pencarian dan klik pada hasil pertama yang muncul yaitu “nesbot/carbon”.

²⁷<https://github.com/briannesbitt/Carbon>

²⁸<http://packagist.org>



The screenshot shows a web browser window for Packagist.org. The URL in the address bar is <https://packagist.org/search/?q=carbon>. The page title is "Packagist - The PHP package archivist". A green button on the right says "Submit Package". Below the title, there's a message: "Packagist is the main Composer repository. It aggregates all sorts of PHP packages that are installable with Composer. Browse packages or submit your own." A search bar contains the text "carbon". Under the heading "Packages", a list of packages is shown:

- nesbot/carbon** (circled in red) - A simple API extension for DateTime. It has 1,828 releases, 644 dependencies, and 33 stars.
- laravelrus/localized-carbon** - A localizable version of Carbon. It has 471 releases, 0 dependencies, and 0 stars.
- qwildz/localized-eloquent-date** - Multi-language support for Laravel Eloquent's dates. It has 1 release, 0 dependencies, and 1 star.
- leoexpress/time** - Time manipulation library inspired by Carbon/Carbon which allows you to handle time gracefully. It has 66 releases, 0 dependencies, and 0 stars.

Mencari package Carbon di Packagist

Pada halaman selanjutnya, kita dapat melihat detail dari package Carbon ini.

nesbot/carbon time date datetime

A simple API extension for DateTime.

Maintainer: briannesbitt

Homepage: <https://github.com/briannesbitt/Carbon>

Canonical: <https://github.com/briannesbitt/Carbon.git>

Source: <https://github.com/briannesbitt/Carbon/tree/master>

Issues: <https://github.com/briannesbitt/Carbon/issues>

dev-master reference: 2b60366

require: "nesbot/carbon": "dev-master"

Author

- Brian Nesbitt <brian@nesbot.com>

Requires

- php: >=5.3.0

Provides

Total install

Overall:	1 828 670 installs
30 days:	226 181 installs
Today:	5 742 installs

Update terakhir

2014-08-26 03:18 UTC MIT

Pembuat

dependensi

Requires (Dev)

- phpunit/phpunit: ~4.0

Conflicts

Suggests

- None

Replaces

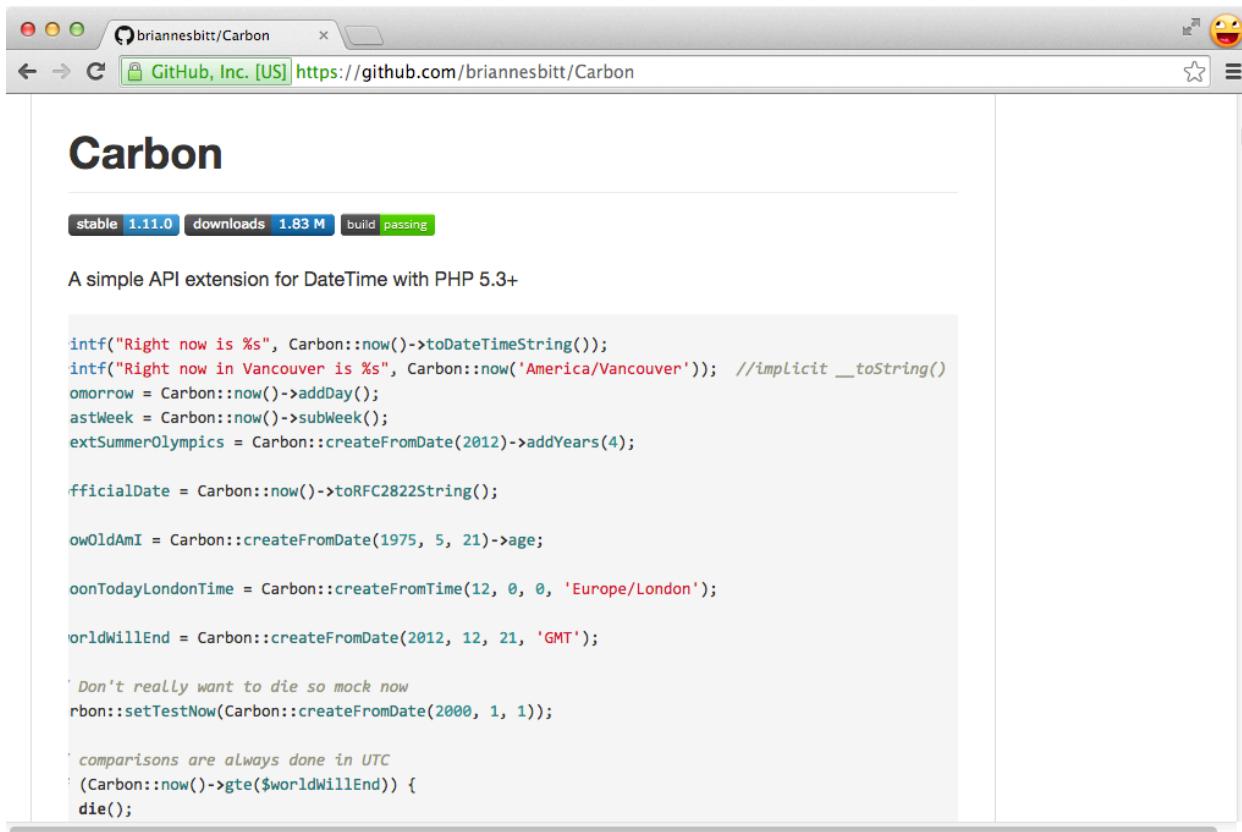
Detail package Carbon di Packagist

Beberapa informasi yang bisa kita dapatkan di halaman ini yaitu:

- Deskripsi package
- Total Install
- Website dokumentasi (Homepage) dan hosting source code
- Data pembuat package
- Update package terakhir
- Dependensi

Jika kita mengunjungi website resmi dari package ini di [https://github.com/briannesbitt/Carbon²⁹](https://github.com/briannesbitt/Carbon), kita akan mendapatkan dokumentasi lengkap cara penggunaan package ini.

²⁹ <https://github.com/briannesbitt/Carbon>



The screenshot shows a Mac OS X desktop with a browser window open to the GitHub page for the Carbon package. The title bar says "briannesbitt/Carbon". The page content includes the package name "Carbon", its stable version "1.11.0", 1.83 M downloads, and a green "build passing" badge. Below this is a brief description: "A simple API extension for DateTime with PHP 5.3+". A large code block follows, showing examples of Carbon usage:

```
intf("Right now is %s", Carbon::now()->toDateTimeString());
intf("Right now in Vancouver is %s", Carbon::now('America/Vancouver')); //implicit __toString()
tomorrow = Carbon::now()->addDay();
lastWeek = Carbon::now()->subWeek();
extSummerOlympics = Carbon::createFromDate(2012)->addYears(4);

officialDate = Carbon::now()->toRFC2822String();

owOldAmI = Carbon::createFromDate(1975, 5, 21)->age;

oonTodayLondonTime = Carbon::createTime(12, 0, 0, 'Europe/London');

orldWillEnd = Carbon::createFromDate(2012, 12, 21, 'GMT');

' Don't really want to die so mock now
arbon::setTestNow(Carbon::createFromDate(2000, 1, 1));

' comparisons are always done in UTC
(Carbon::now()->gte($worldWillEnd)) {
die();
```

Dokumentasi Carbon

Kembali ke halaman packagist, untuk menginstall package Carbon ini, coba kita scroll agak ke bawah. kita akan melihat beberapa versi dari Carbon. Ada dua jenis, yaitu versi dalam pengembangan (dev-master) dan stabil. Saran saya, **selalu gunakan versi yang stabil.**

nesbot/carbon - Packagist

<https://packagist.org/packages/nesbot/carbon>

dev-master reference: 2b60366 Terbaru 2014-08-26 03:18 UTC MIT

require: "nesbot/carbon": "dev-master"

Author
• Brian Nesbitt <brian@nesbot.com>

Requires
• php: >=5.3.0

Provides
None

Requires (Dev)
• phpunit/phpunit: ~4.0

Conflicts
None

Suggests
None

Replaces
None

1.11.0 reference: 2b60366 Stable 2014-08-26 03:18 UTC MIT

1.10.0 reference: 9b42a1a 2014-07-18 03:44 UTC MIT

1.9.0 reference: b94de71 2014-05-13 02:29 UTC MIT

1.8.0 reference: 21c4cb4 2014-01-07 05:10 UTC MIT

1.7.0 reference: 03ede52 2013-12-05 04:13 UTC MIT

Versi Carbon

Pada contoh ini, kita akan menginstall versi 1.11.0 maka kita klik pada versi itu. Pada kotak yang muncul, copy pada tulisan setelah require. Yaitu "nesbot/carbon": "1.11.0".

The screenshot shows the Packagist.org page for the `nesbot/carbon` package. It displays three versions: `dev-master`, `1.11.0`, and `1.10.0`. The `require` field for `1.11.0` is highlighted with a red oval.

Version	Reference	Last Updated	Author	Requires	Provides	Requires (Dev)	Conflicts	Suggests	Replaces
<code>dev-master</code>	<code>2b60366</code>	<code>2014-08-26 03:18 UTC</code>	Brian Nesbitt <brian@nesbot.com>	<code>php: >=5.3.0</code>	None	<code>phpunit/phpunit: ~4.0</code>	None	None	None
<code>1.11.0</code>	<code>reference: 2b60366</code>	<code>2014-08-26 03:18 UTC</code>	Brian Nesbitt <brian@nesbot.com>	<code>php: >=5.3.0</code>	None	<code>phpunit/phpunit: ~4.0</code>	None	None	None
<code>1.10.0</code>	<code>reference: 9b42a1a</code>	<code>2014-07-18 03:44 UTC</code>	Brian Nesbitt <brian@nesbot.com>	<code>php: >=5.3.0</code>	None	<code>phpunit/phpunit: ~4.0</code>	None	None	None

Pilih versi Carbon

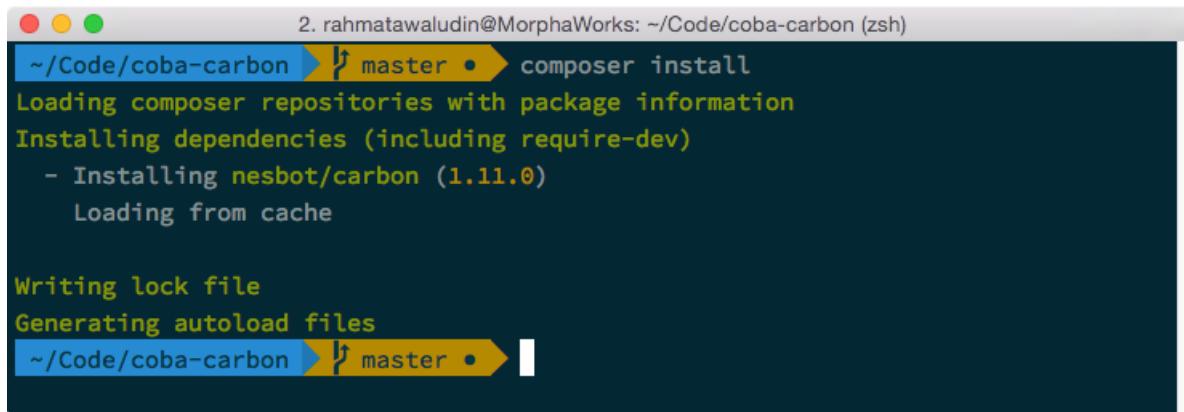
Untuk menginstall package ini, mari buat sebuah folder misalnya di `~/Code/coba-carbon`. Pada folder ini, buatlah file baru bernama `composer.json` dengan isi:

`~/Code/coba-carbon`

```
{
    "require" : {
        "nesbot/carbon": "1.11.0"
    }
}
```

Pada file ini, kita mem-paste teks dari packagist tadi di baris ke 3. Dengan composer, kita cukup menyimpan nama dari package yang dibutuhkan dan versinya pada bagian `require`. Selanjutnya, untuk mendownload package tersebut, jalankan perintah:

`composer install`

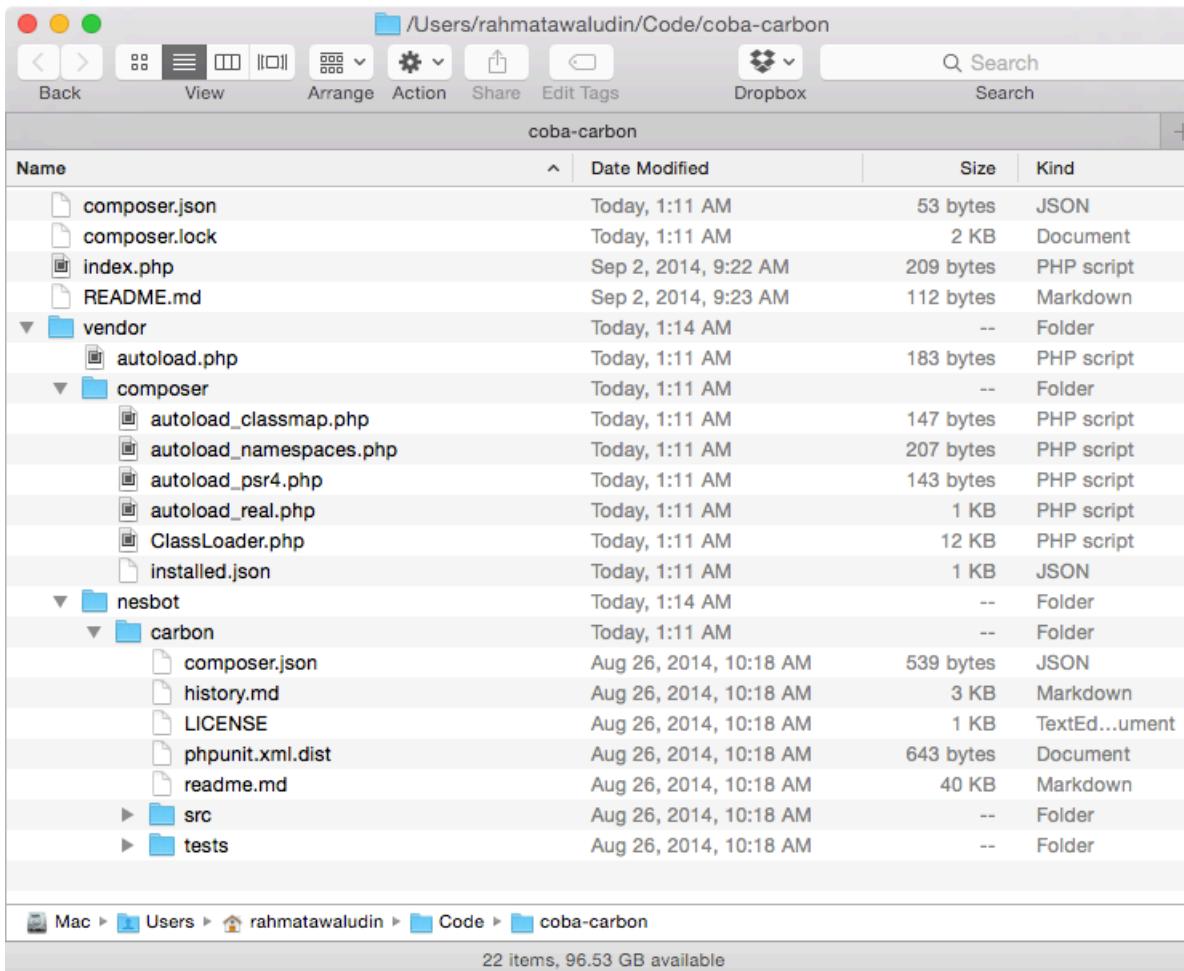


```
2. rahmatawaludin@MorphaWorks: ~/Code/coba-carbon (zsh)
~/Code/coba-carbon ➜ master • composer install
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing nesbot/carbon (1.11.0)
  Loading from cache

Writing lock file
Generating autoload files
~/Code/coba-carbon ➜ master •
```

Menginstall Carbon

Jika berhasil mendownload Carbon, kini akan ada beberapa file dan folder baru di folder coba-carbon:



Struktuf folder composer

Biar lebih paham, berikut penjelasan file dan folder itu:

- **composer.json**, ini file yang kita buat, berisi dependensi library dari project kita.
- **composer.lock**, file ini mencatat versi package yang saat ini terinstal.
- **vendor**, folder ini berisi package yang telah kita install. Setiap package yang kita tulis di bagian `require` akan di download ke folder ini.
- **vendor/autoload.php**, berfungsi memanggil autoloader dari composer.

Salah satu keunggulan dari composer adalah autoloader. Fitur ini berfungsi untuk memanggil class yang sesuai ketika kita membutuhkan class dari suatu library. Jadi, jika banyak library yang kita install kita hanya cukup menulis `require` untuk file `vendor/autoload.php`. Dan, autoload ini cukup cerdas dengan hanya me-load class yang kita butuhkan.

Untuk mempraktekan penggunaan autoload ini, kita akan menggunakan library Carbon untuk menampilkan tanggal dalam bentuk "... ago". Caranya, buatlah file `index.php` di `~/Code/coba-carbon` dengan isi:

`~/Code/coba-carbon/index.php`

```
<?php
require 'vendor/autoload.php';
use Carbon\Carbon;
date_default_timezone_set('Asia/Jakarta');
$date = Carbon::createFromDate(1945, 8, 17);
printf("Kapan Indonesia Merdeka? %s\n", $date->diffForHumans());
```

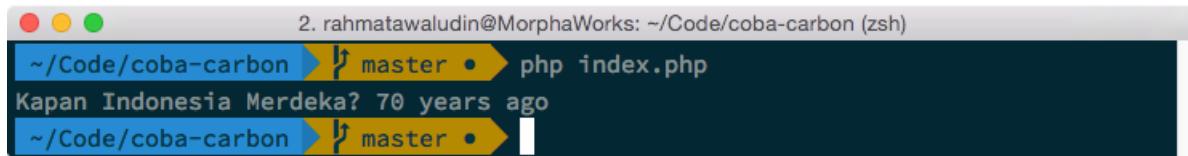
File ini memiliki tugas yang sangat penting, yaitu... memberitahu kita kapan Indonesia merdeka :D

Oke, ini penjelasannya:

- Baris 2: me-`require` autoload dari composer.
- Baris 3: Menggunakan *namespace* (akan dijelaskan kemudian) kita meload library Carbon.
- Baris 4: Mengeset timezone `default_content`
- Baris 5: Membuat tanggal
- Baris 6: Memanggil tanggal dalam bentuk "... ago"

Untuk mengetesnya jalankan perintah ini dari folder `~/Code/coba-carbon`:

```
php index.php
```



```
2. rahmatawaludin@MorphaWorks: ~/Code/coba-carbon (zsh)
~/Code/coba-carbon > master • php index.php
Kapan Indonesia Merdeka? 70 years ago
~/Code/coba-carbon > master •
```

Berhasil mendemokan Carbon dengan composer

Sip. Berhasil!

Kini, ceritanya kita butuh package baru di project ini misalnya kita butuh mengakses dropbox untuk hosting file kita. Mari kita tambahkan package [Flysystem³⁰](#). Package ini berguna untuk menyeragamkan syntax untuk mengakses file di FTP, AWS S3, Rackspace, Dropbox, Sftp, Zip dan WebDAV.

Ubah file `~/Code/coba-carbon/composer.json` menjadi :

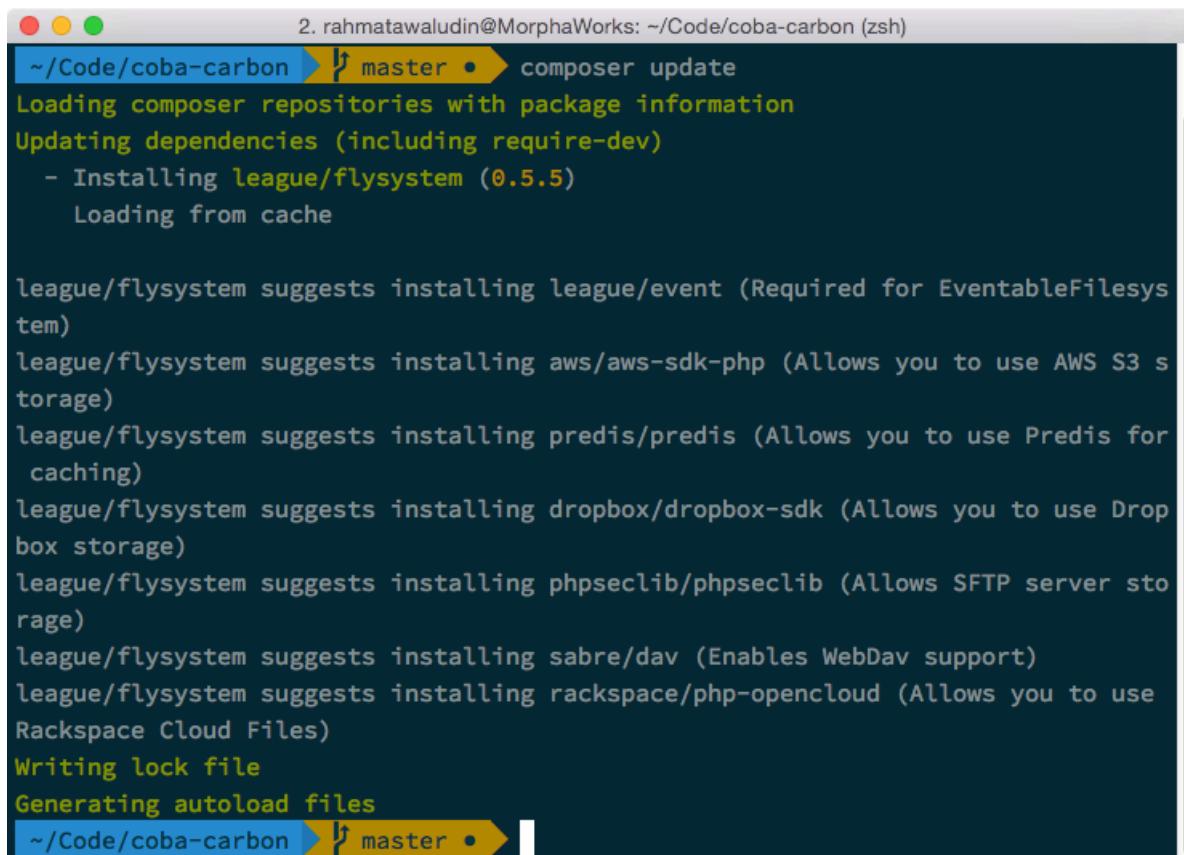
`~/Code/coba-carbon`

```
{  
    "require" : {  
        "nesbot/carbon": "1.11.0",  
        "league/flysystem": "0.5.5"  
    }  
}
```

Untuk menginstall package baru tersebut, jalankan perintah

```
composer update
```

³⁰<http://flysystem.thephpleague.com>



```
2. rahmatawaludin@MorphaWorks: ~/Code/coba-carbon (zsh)
~/Code/coba-carbon ➜ master • composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing league/flysystem (0.5.5)
  Loading from cache

league/flysystem suggests installing league/event (Required for EventableFilesystem)
league/flysystem suggests installing aws/aws-sdk-php (Allows you to use AWS S3 storage)
league/flysystem suggests installing predis/predis (Allows you to use Predis for caching)
league/flysystem suggests installing dropbox/dropbox-sdk (Allows you to use Dropbox storage)
league/flysystem suggests installing phpseclib/phpseclib (Allows SFTP server storage)
league/flysystem suggests installing sabre/dav (Enables WebDav support)
league/flysystem suggests installing rackspace/php-opencloud (Allows you to use Rackspace Cloud Files)
Writing lock file
Generating autoload files
~/Code/coba-carbon ➜ master •
```

Update composer untuk menginstall package baru

Perintah ini, selain berfungsi menginstall package baru yang ditambahkan di bagian require, juga berfungsi mengupdate package yang telah terinstall jika kita merubah versi package yang digunakan.

Jika dibutuhkan, kita juga dapat menginstruksikan composer untuk menginstall package hanya untuk development. Caranya dengan membuat key require-dev di composer.json berisi package yang kita butuhkan hanya pada saat development. Untuk menginstall/mengupdate berikut package yang berada di require-dev ini, jalankan composer dengan opsi --dev.

Saya rasa penjelasan sekilas tentang composer ini sudah cukup. Masih banyak topik lain dari composer misalnya membuat package dan repository privat yang bisa dipelajari di [https://getcomposer.org/doc³¹](https://getcomposer.org/doc).



Source code dari latihan ini bisa didapat di
[https://github.com/rahmatawaludin/coba-carbon³²](https://github.com/rahmatawaludin/coba-carbon)

³¹ <https://getcomposer.org/doc>

³² <https://github.com/rahmatawaludin/coba-carbon/commits/master>



Ringkasan

Pada bab ini kita telah mempelajari topik-topik dasar sebelum memulai mempelajari Framework Laravel. Memahami topik ini sangat penting agar kita tidak tenggelam dalam lautan kebingungan ketika mempelajari Laravel. Banyak fitur Laravel yang akan dijelaskan di bab-bab selanjutnya yang bergantung pada topik di bab ini.

Topik yang kita bahas disini merupakan beberapa teknik coding PHP Modern. Untuk pembahasan Laravel, teknik yang kita bahas di bab ini sudah cukup sebagai fondasi untuk memahami core Laravel. Jika Anda tertarik mempelajari teknik lainnya, bisa membaca buku [Teknik PHP Modern³³](#).

Selanjutnya, kita akan membahas instalasi dan konfigurasi Framework Laravel. Semangat!

³³<https://leanpub.com/teknikphpmodern>

Konfigurasi, untuk Project Skala Besar

Dalam pengembangan aplikasi PHP, banyak sekali cara menyiapkan *development environment*. Sebagian ada yang mudah diinstal, namun repot ketika di deploy. Sebagian multi platform, sebagian tidak. Sebagian terkadang terlalu kompleks untuk di setup.

Dalam pembahasan ini, kita akan mempelajari berbagai cara menyiapkan *development environment* untuk Laravel. Jika sudah memiliki *development environment* yang sudah berjalan, pastikan sudah sesuai dengan kebutuhan sistem yang tertera di bab ini.

Kebutuhan Sistem

Untuk menjalankan framework Laravel dan mempelajari buku ini dibutuhkan :

- **Komputer/Laptop**, kalau berniat mempraktekan ilmu di buku ini, maka perangkat ini hukumnya wajib.
- **Operating System**, selama menulis buku ini saya menggunakan OSX. Tapi, semua syntax di buku ini seharusnya bisa dijalankan di Linux maupun Windows. Jika setelah mengikuti setiap instruksi dengan seksama tapi mendapat error dari perintah perintah yang dijalankan, jangan sungkan untuk menghubungi saya.
- **Web Server**, bisa pakai apache atau ngix.
- **PHP >= 5.4**, karena Laravel adalah framework PHP, maka pasti dibutuhkan PHP. Laravel juga menggunakan beberapa fitur modern dari PHP, makanya dibutuhkan PHP versi 5.4 keatas. Untuk mengetahui versi php yang terinstal di sistem, kita dapat menjalankan perintah `php -v` di terminal.

```
2. rahmataludin@MorphaWorks: ~ (zsh)
~ ➔ php -v
PHP 5.6.2 (cli) (built: Oct 28 2014 00:01:08)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.6.0, Copyright (c) 1998-2014 Zend Technologies
~ ➔ |
```

Mengecek versi PHP

- Beberapa ekstensi PHP yaitu **mcrypt**, **mbstring**, **json** dan **fileinfo**. Untuk mengecek apakah semua extension itu telah aktif, kita dapat menggunakan `phpinfo()`. Misalnya dengan cara seperti ini:
 1. Buat folder baru, misalnya `test-info`.
 2. Didalam folder tersebut buatlah file baru bernama `index.php` dengan isi

~/Code/test-info/index.php

<?php

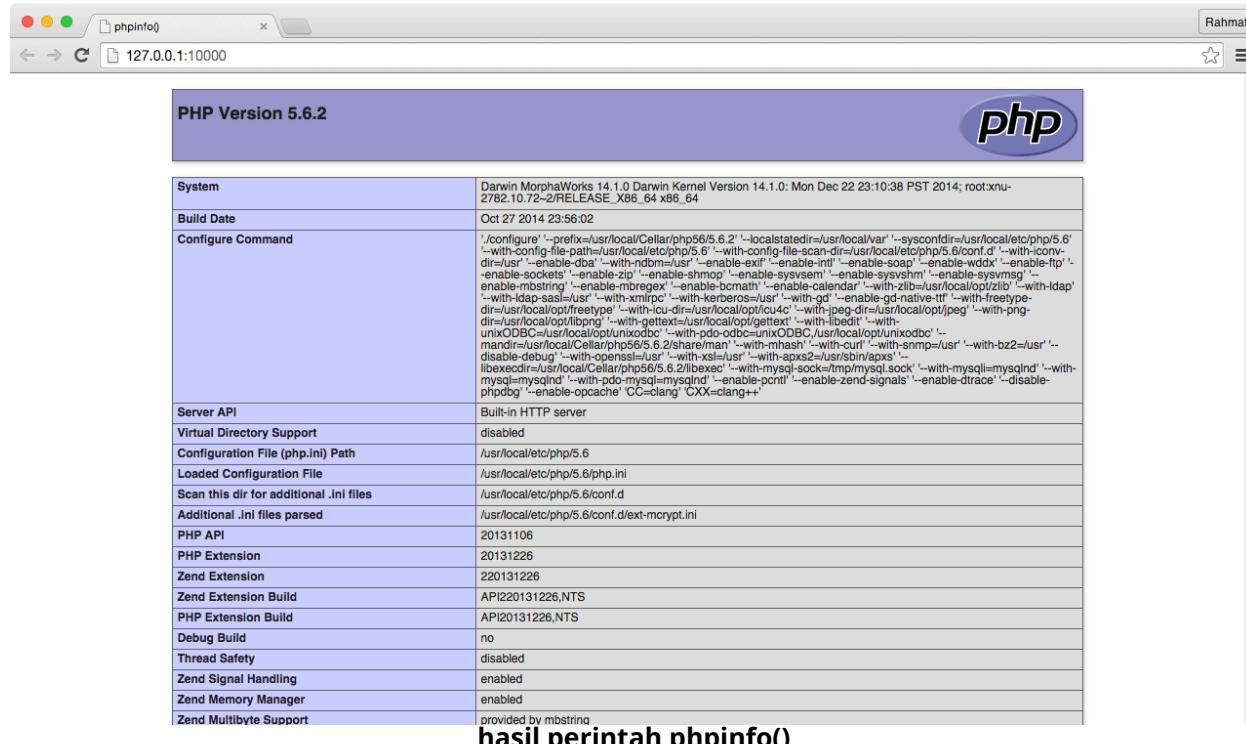
```
phpinfo();
```

3. Kita akan menjalankan built-in web server di PHP³⁴. Jalankan perintah `php -S 127.0.0.1:10000` pada folder `test-info` dari terminal.

```
2. php -S 127.0.0.1:10000 (php)
~/Code/test-info ➜ php -S 127.0.0.1:10000
PHP 5.6.2 Development Server started at Thu Feb 19 10:00:46 2015
Listening on http://127.0.0.1:10000
Document root is /Users/rahmatawaludin/Code/test-info
Press Ctrl-C to quit.
```

Menjalankan built-in web server

4. Buka <http://127.0.0.1:10000> dari browser. Kemudian carilah extension yang dituliskan diatas, pastikan sudah tersedia. Jika belum ada, silahkan diinstall/diaktifkan terlebih dahulu.



hasil perintah phpinfo()

³⁴<http://php.net/manual/en/features.commandline.webserver.php>

mcrypt

| | | |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| mcrypt support | enabled | |
| mcrypt_filter support | enabled | |
| Version | 2.5.8 | |
| Api No | 20021217 | |
| Supported ciphers | cast-128 gost rijndael-128 twofish arcfour cast-256 loki97 rijndael-192 saferplus wake blowfish-compat des rijndael-256 serpent xtea blowfish enigma rc2 tripledes | |
| Supported modes | cbc cfb ctr ecb ncfb nofb ofb stream | |
| Directive | Local Value | Master Value |
| mcrypt.algorithms_dir | <i>no value</i> | <i>no value</i> |
| mcrypt.modes_dir | <i>no value</i> | <i>no value</i> |

mhash

| | |
|--------------------------|-------------------------|
| MHASH support | Enabled |
| MHASH API Version | Emulated Support |

mysql

| | | |
|---------------------------------|----------------------------------------------------------------------------------|---------------------|
| MySQL Support | enabled | |
| Active Persistent Links | 0 | |
| Active Links | 0 | |
| Client API version | mysqld 5.0.11-dev - 20120503 - \$Id: f373ea5dd5538761406a8022a4b8a374418b240e \$ | |
| Directive | Local Value | Master Value |
| mysql.allow_local_infile | On | On |
| mysql.allow_persistent | On | On |

Ekstensi Mcrypt aktif

mbstring

| | |
|------------------------------|----------------|
| libXML Loaded Version | 20900 |
| libXML streams | enabled |

mbstring

| | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|---------------------------------|
| Multibyte Support | enabled | |
| Multibyte string engine | libmbfl | |
| HTTP input encoding translation | disabled | |
| libmbfl version | 1.3.2 | |
| mbstring extension makes use of "streamable kanji code filter and converter", which is distributed under the GNU Lesser General Public License version 2.1. | | |
| Multibyte (japanese) regex support | enabled | |
| Multibyte regex (oniguruma) backtrack check | On | |
| Multibyte regex (oniguruma) version | 5.9.5 | |
| Directive | Local Value | Master Value |
| mbstring.detect_order | <i>no value</i> | <i>no value</i> |
| mbstring.encoding_translation | Off | Off |
| mbstring.func_overload | 0 | 0 |
| mbstring.http_input | <i>no value</i> | <i>no value</i> |
| mbstring.http_output | <i>no value</i> | <i>no value</i> |
| mbstring.http_output_conv_mimetypes | ^(text/* application/xhtml+xml) | ^(text/* application/xhtml+xml) |
| mbstring.internal_encoding | <i>no value</i> | <i>no value</i> |
| mbstring.language | neutral | neutral |
| mbstring.strict_detection | Off | Off |
| mbstring.substitute_character | <i>no value</i> | <i>no value</i> |

mcrypt

| | |
|------------------------------|----------------|
| mcrypt support | enabled |
| mcrypt_filter support | enabled |
| Version | 2.5.8 |
| Api No | 20021217 |

Ekstensi Mbstring aktif

json

| Directive | Local Value | Master Value |
|---------------------|-------------|--------------|
| intl.default_locale | no value | no value |
| Intl.error_level | 0 | 0 |
| intl.use_exceptions | 0 | 0 |

ldap

| Directive | Local Value | Master Value |
|----------------|-------------|--------------|
| ldap.max_links | Unlimited | Unlimited |

libxml

| Directive | Local Value | Master Value |
|-------------------------|-------------|--------------|
| libXML support | active | |
| libXML Compiled Version | 2.9.0 | |
| libXML Loaded Version | 20900 | |
| libXML streams | enabled | |

mbstring

| Directive | Local Value | Master Value |
|-------------------------|-------------|--------------|
| Multibyte Support | enabled | |
| Multibyte string engine | libmbfl | |

Ekstensi Json aktif

fileinfo

| Directive | Local Value | Master Value |
|---------------------|-------------|--------------|
| exif.encode_jis | no value | no value |
| exif.encode_unicode | ISO-8859-15 | ISO-8859-15 |

filter

| Directive | Local Value | Master Value |
|----------------------|-------------|--------------|
| filter.default | unsafe_raw | unsafe_raw |
| filter.default_flags | no value | no value |

ftp

| Directive | Local Value | Master Value |
|-------------|-------------|--------------|
| FTP support | enabled | |

gd

| Directive | Local Value | Master Value |
|--------------------|----------------------------|--------------|
| GD Support | enabled | |
| GD Version | bundled (2.1.0 compatible) | |
| FreeType Support | enabled | |
| FreeType Linkage | with freetype | |
| FreeType Version | 2.5.3 | |
| GIF Read Support | enabled | |
| GIF Create Support | enabled | |
| JPEG Support | enabled | |
| libJPEG Version | 8 | |

Ekstensi Fileinfo aktif

- **Database Server**, bisa pakai MySQL, PostgreSQL, SQLite atau SQLServer.

- **Text Editor.** Untuk teks editor ini kita dapat menggunakan editor apapun yang disukai. Saya sendiri menggunakan [Sublime Text³⁵](#).
- **Terminal**, pada pembahasan buku ini Anda akan diminta untuk menjalankan perintah di Terminal. Pada windows, artinya menjalankan CMD. Sedangkan pada OSX atau Linux artinya menjalankan aplikasi terminal.
- **Virtual host**, akan kita pelajari lebih lanjut.

Instalasi

Laravel berjalan diatas berbagai komponen yang memiliki berbagai dependensi. Untuk menangani ini, Laravel menggunakan composer sebagai tools untuk melakukan instalasi. Jika belum memahami penggunaan composer, silahkan baca kembali di bab **Konsep Dasar**. Terdapat dua cara untuk menginstal laravel, yaitu menggunakan composer create-project dan laravel installer.

Composer create-project

Untuk membuat project baru di folder `webapp`, kita cukup menjalankan:

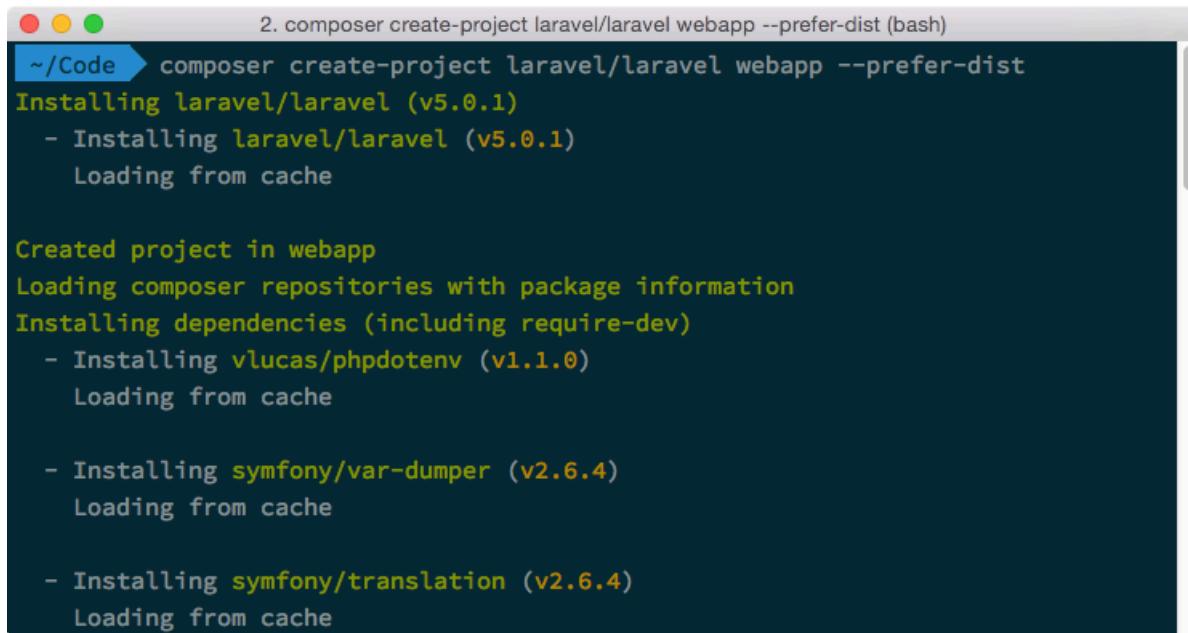
composer create-project

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

Perintah ini akan mendownload dari repository [https://github.com/laravel/laravel³⁶](https://github.com/laravel/laravel). Dan menginstall semua dependensi dari Laravel menggunakan `composer install`.

³⁵ <http://www.sublimetext.com>

³⁶ <https://github.com/laravel/laravel>



```
2. composer create-project laravel/laravel webapp --prefer-dist (bash)
~/Code ➤ composer create-project laravel/laravel webapp --prefer-dist
Installing laravel/laravel (v5.0.1)
- Installing laravel/laravel (v5.0.1)
  Loading from cache

Created project in webapp
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing vlucas/phpdotenv (v1.1.0)
  Loading from cache

- Installing symfony/var-dumper (v2.6.4)
  Loading from cache

- Installing symfony/translation (v2.6.4)
  Loading from cache
```

Menginstall laravel dengan composer create-project

Terlihat disini saya menginstall Laravel versi 5. Secara default perintah ini akan menginstall versi terbaru dari Laravel. Ketika buku ini ditulis, versi terbaru adalah 5.0.1. Jika ingin menginstall versi lain dari laravel, cukup menambah versi setelah laravel/laravel. Misalnya, untuk menginstall versi 4.2.11 jalankan perintah:

composer create-project dengan versi

```
composer create-project laravel/laravel=4.2.11 laravel-4 --prefer-dist
```

```
2. composer create-project laravel/laravel=4.2.11 laravel-4 --prefer-dist (bash)
~/Code ➔ composer create-project laravel/laravel=4.2.11 laravel-4 --prefer-dist

Installing laravel/laravel (v4.2.11)
- Installing laravel/laravel (v4.2.11)
  Loading from cache

Created project in laravel-4
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing symfony/translation (v2.5.10)
  Loading from cache

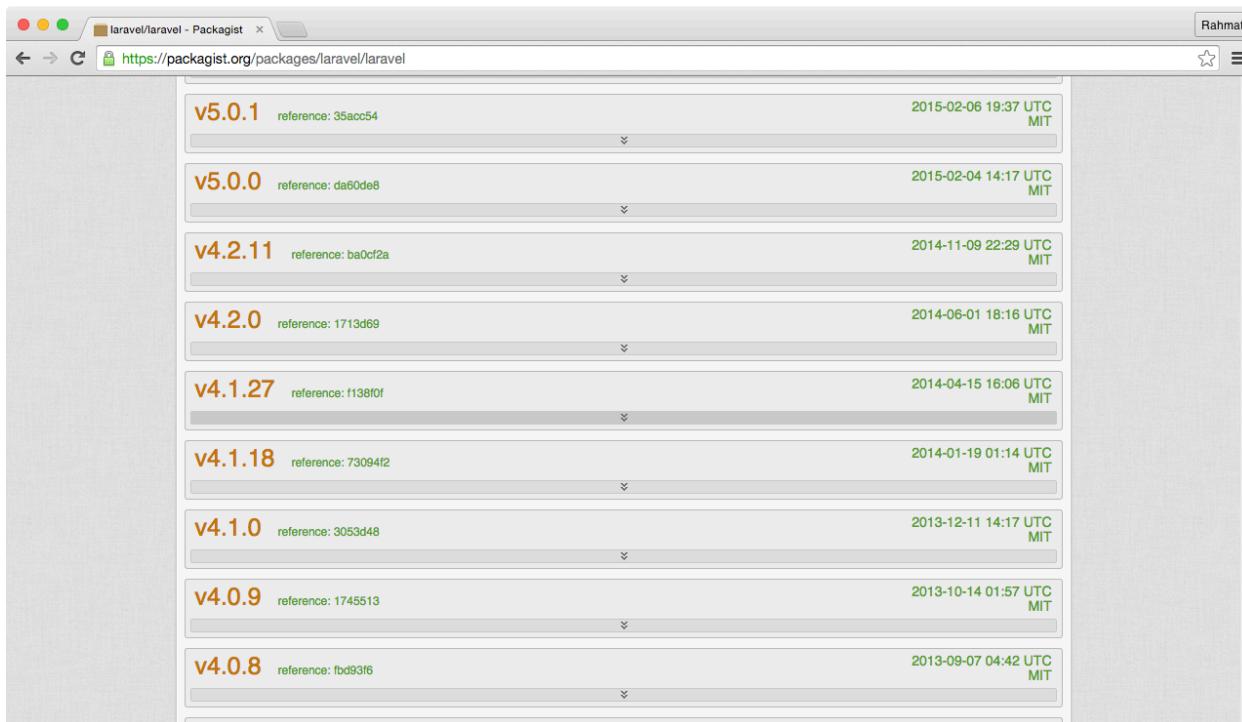
- Installing symfony/security-core (v2.5.10)
  Loading from cache

- Installing symfony/routing (v2.5.10)
  Loading from cache
```

composer create-project untuk versi 4.2.11

Untuk mengecek versi Laravel berapa saja yang tersedia, cek di packagist³⁷

³⁷ <https://packagist.org/packages/laravel/laravel>

**Versi Laravel di Packagist**

Cek instalasi Laravel

Untuk mengecek versi Laravel yang terinstal, kita dapat menggunakan perintah:

```
php artisan -V
```

Cara cepat untuk mengecek instalasi Laravel adalah dengan menggunakan fitur built-in web server di PHP. Caranya, masuk ke folder webapp/public dan jalankan perintah

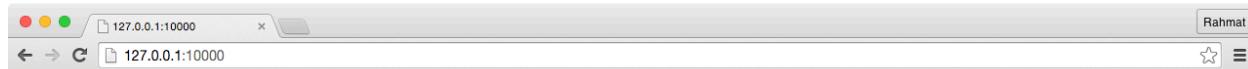
menjalankan built-in web server di port 10000

```
php -S 127.0.0.1:10000
```

```
2. php -S 127.0.0.1:10000 (php)
~/Code/webapp/public ➤ php -S 127.0.0.1:10000
PHP 5.6.2 Development Server started at Sat Feb 21 22:32:58 2015
Listening on http://127.0.0.1:10000
Document root is /Users/rahmatawaludin/Code/webapp/public
Press Ctrl-C to quit.
```

Menjalankan built-in web server

Perintah diatas akan menjalankan built-in web server di port 10000. Akses di `http://127.0.0.1:10000`



Laravel 5

Simplicity is an acquired taste. - Katharine Gerould

Laravel berjalan di port 10000

Untuk menghentikannya, tekan `CTRL+C`.

Kabar baiknya, Laravel sendiri sudah punya fitur untuk menjalankan built-in web server ini. Caranya, masuk ke folder `webapp` dan jalankan perintah:

artisan serve untuk menjalankan built-in web server

```
php artisan serve
```

A screenshot of a terminal window titled "2. php artisan serve (php)". The terminal shows the command `~/Code/webapp/public > cd ..`, then `~/Code/webapp > php artisan serve`, and finally the output `Laravel development server started on http://localhost:8000`.

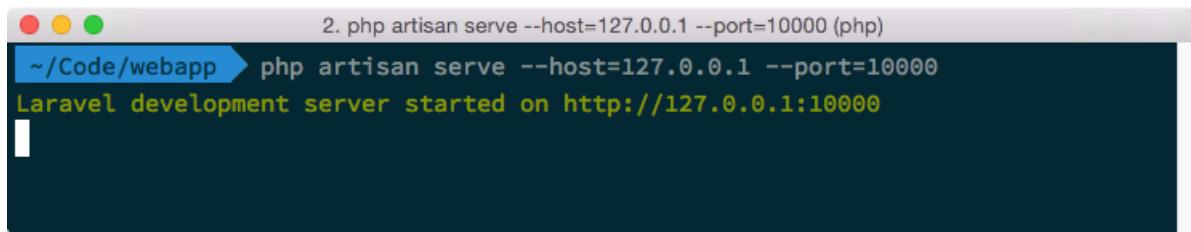
```
2. php artisan serve (php)
~/Code/webapp/public > cd ..
~/Code/webapp > php artisan serve
Laravel development server started on http://localhost:8000
```

Artisan serve

Terlihat disini, artisan serve menjalankan built-in web server di localhost pada port 8000. Kita dapat mengaksesnya di `http://localhost:8000`. Jika kita ingin merubahnya ip dan portnya, misalnya ke ip 127.0.0.1 dan port 10000, ubah perintahnya menjadi:

artisan serve untuk menjalankan built-in web server

```
php artisan serve --host=127.0.0.1 --port=10000
```



```
2. php artisan serve --host=127.0.0.1 --port=10000 (php)
~/Code/webapp ➤ php artisan serve --host=127.0.0.1 --port=10000
Laravel development server started on http://127.0.0.1:10000
```

Artisan serve custom ip dan port

Dan hasilnya akan tetap sama



Laravel 5

Simplicity is an acquired taste. - Katharine Gerould

Artisan serve berhasil**Kita bisa menggunakan artisan serve untuk development**

Untuk pembuatan website sederhana (misalnya CRUD) yang tidak akan membutuhkan banyak fitur Laravel. Penggunaan artisan serve sudah cukup untuk development. Kita bahkan tidak perlu menggunakan homestead. Ketika menggunakan artisan serve, kita juga harus merestart server ketika melakukan perubahan file .env.

Error koneksi database?

Menggunakan artisan serve, terkadang menyebabkan koneksi database error. Tidak selalu terjadi sih. Penyebabnya karena Laravel tidak dapat menentukan letak file socket mysql. Solusinya, kita harus menemukan letak file socket dengan login ke mysql dan jalankan status sehingga muncul output seperti berikut:

```
2. mysql -u root -p (mysql)
Current pager: less
Using outfile: ''
Using delimiter: ;
Server version: 5.5.42 Source distribution
Protocol version: 10
Connection: Localhost via UNIX socket
Server characterset: latin1
Db      characterset: latin1
Client characterset: utf8
Conn.   characterset: utf8
UNIX socket: /Applications/MAMP/tmp/mysql/mysql.sock
Uptime: 19 hours 13 min 17 sec

Threads: 3 Questions: 871 Slow queries: 0 Opens: 57 Flush tables: 1 Open tables: 50 Queries per second avg: 0.012
-----
mysql> _
```

Lokasi file socket

Bila masih tidak ditemukan, carilah file konfigurasi mysql pada sistem dan temukan konfigurasi untuk lokasi file socket.

Selanjutnya, kta tambahkan isian `unix_socket` pada konfigurasi koneksi database dengan isi alamat yang kita temukan diatas. Misalnya, hasil akhirnya akan seperti berikut:

```
....  
'mysql' => [  
    'driver'    => 'mysql',  
    'host'      => env('DB_HOST', 'localhost'),  
    'database'  => env('DB_DATABASE', 'forge'),  
    'username'  => env('DB_USERNAME', 'forge'),  
    'password'  => env('DB_PASSWORD', ''),  
    'charset'   => 'utf8',  
    'collation' => 'utf8_unicode_ci',  
    'prefix'    => '',  
    'strict'    => false,  
    'unix_socket' => '/Applications/MAMP/tmp/mysql/mysql.sock',  
,  
....
```

Homestead

Setelah kita melihat semua kebutuhan diatas, mungkin terpikir untuk mendownload semua kebutuhan tersebut secara manual satu persatu atau paketan seperti xampp dan wampp. Tapi, karena laravel itu **pro developer males**, sudah ada tools yang mempercepat proses persiapan development environment ini, namanya Homestead.

Homestead adalah sebuah virtual machine berisi OS Ubuntu yang berjalan di virtual box yang dikonfigurasi dengan vagrant. Bagian menarik dari homestead adalah virtual machine yang dijalankan sudah terinstall dan terkonfigurasi semua aplikasi server yang kita butuhkan untuk mulai development laravel.

Penggunaan homestead akan memberikan banyak keuntungan:

- Proses pembuatan *development environment* lebih cepat.
- Tidak perlu repot melakukan konfigurasi untuk setiap aplikasi server (web server, database server, dll).
- Jika kita bekerja dalam tim dengan sistem operasi yang berbeda-beda, penggunaan homestead akan menyeragamkan *development environment*. Sehingga tidak ada istilah, aplikasi jalan di Laptop Joni tapi tidak jalan di Mac Udin.
- Proses pembuatan virtual host untuk setiap aplikasi Laravel akan lebih cepat.

Beberapa aplikasi yang sudah terinstal di homestead:

- Ubuntu 14.04
- PHP 5.6
- Composer

- HHVM
- Nginx
- MySQL
- Postgres
- Node (dengan Bower, Grunt, dan Gulp)
- Redis
- Memcached
- Beanstalkd
- Laravel Envoy
- Fabric + HipChat Extension



Homestead tidak wajib

Sebagaimana dijelaskan sebelumnya, penggunaan homestead ini tidak wajib. Bahkan, untuk pemula langkah-langkah dibawah akan cukup membingungkan. Jika Anda siap untuk menghadapinya, lanjutkan. Jika belum siap, saya sarankan untuk langsung ke pembahasan "Artisan CLI" dan menggunakan `php artisan serve` selama development Laravel.

Instalasi

Mari kita install homestead.

1. Install Virtualbox dari <https://www.virtualbox.org/>³⁸

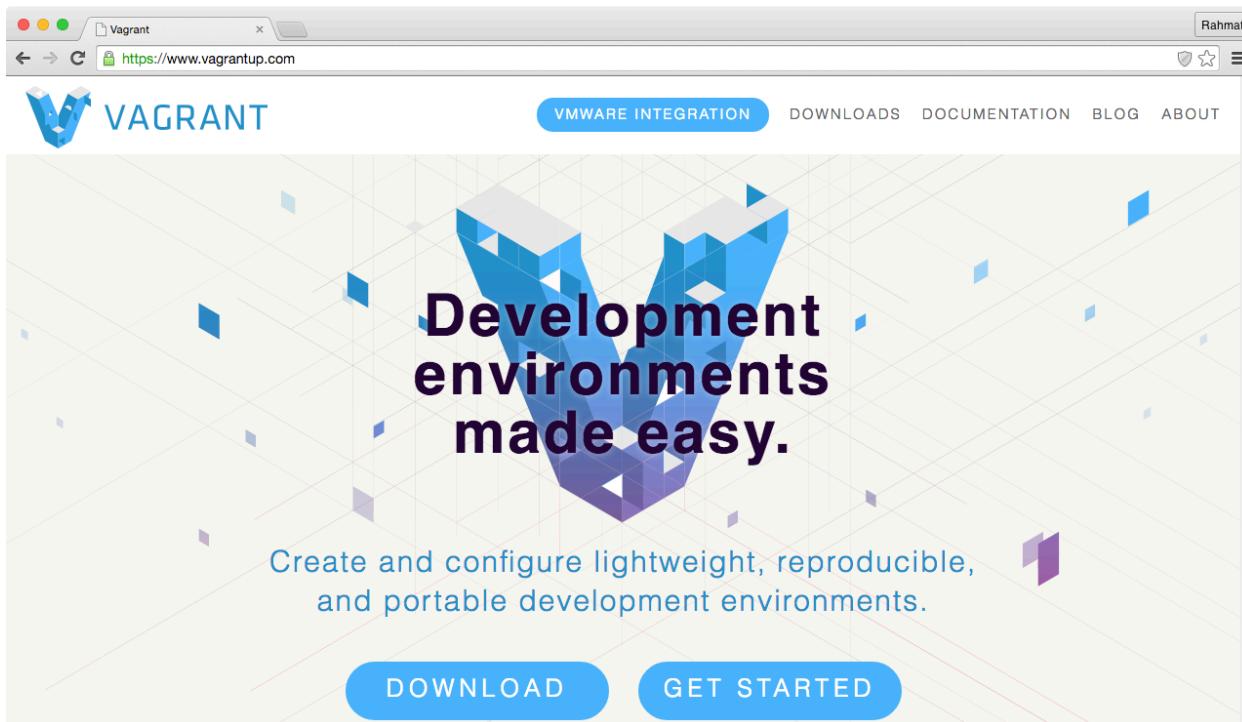
³⁸<https://www.virtualbox.org/>

The screenshot shows the official website for Oracle VM VirtualBox. The page features a large blue header with the word "VirtualBox" in white. Below the header is a banner with the text "Welcome to VirtualBox.org!". To the left, there's a sidebar with links to "About", "Screenshots", "Downloads", "Documentation", "End-user docs", "Technical docs", "Contribute", and "Community". The main content area contains text about the product's features and operating systems supported. A "Hot picks:" section lists several developer tools. On the right, there's a "News Flash" box with three items: "New Feb 12th, 2015 VirtualBox 4.3.22 released!", "Important February, 2015 We're hiring!", and "New Jan 17th, 2015 VirtualBox 4.2.28, 4.1.36, 4.0.28 and 3.2.26 released!". At the bottom, there's an "ORACLE" logo and links to "Contact", "Privacy policy", and "Terms of Use".

Homepage VirtualBox

2. Install Vagrant dari <https://www.vagrantup.com/>³⁹

³⁹<https://www.vagrantup.com/>



Homepage Vagrant

3. Dalam vagrant, setiap virtual machine di dalam virtualbox dinamakan **box**. Install box homestead dengan mengetikkan vagrant box add laravel/homestead di terminal. Perintah ini akan memakan waktu yang cukup lama (tergantung koneksi internet), silahkan bikin kopi dulu.

A screenshot of a Mac OS X terminal window. The title bar says '2. vagrant box add laravel/homestead (curl)'. The terminal output shows the command being run: 'vagrant box add laravel/homestead'. It then displays the progress of the download, including metadata loading, URL, adding the box to the provider (virtualbox), and the start of the download process. The progress bar indicates 0% completion with a rate of 114k/s and an estimated time remaining of 3:16:17.

```
vagrant box add laravel/homestead
==> box: Loading metadata for box 'laravel/homestead'
box: URL: https://vagrantcloud.com/laravel/homestead
==> box: Adding box 'laravel/homestead' (v0.2.2) for provider: virtualbox
box: Downloading: https://vagrantcloud.com/laravel/boxes/homestead/versions/
0.2.2/providers/virtualbox.box
box: Progress: 0% (Rate: 114k/s, Estimated time remaining: 3:16:17)
```

Mendownload box homestead

4. Jika sudah selesai, cek dengan perintah `vagrant box list`. Pastikan terdapat `laravel/homestead` pada outputnya. Jika belum ada, ulangi langkah 3.

```
vagrant box list
base          (virtualbox, 0)
laravel/homestead (virtualbox, 0.2.2)
precise32      (virtualbox, 0)
```

Cek box homestead

5. Setelah berhasil menginstall box homestead, langkah selanjutnya adalah menginstall Homestead CLI dengan composer. Jalankan perintah `composer global require "laravel/homestead=~2.0"`.

```
composer global require "laravel/homestead=~2.0"
Changed current directory to /Users/rahmatawaludin/.composer
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing laravel/homestead (v2.0.9)
  Loading from cache

Writing lock file
Generating autoload files
```

Menginstall Homestead CLI

6. Setelah terinstall, jika belum, kita harus menambahkan folder `~/composer/vendor/bin` atau `C:\Users\<user>\AppData\Roaming\Composer\vendor\bin` ke PATH variable. Silahkan cek langkah ini pada tahap menginstall Laravel Installer pada bagian sebelumnya. Setelah berhasil cek dengan perintah `homestead -V`

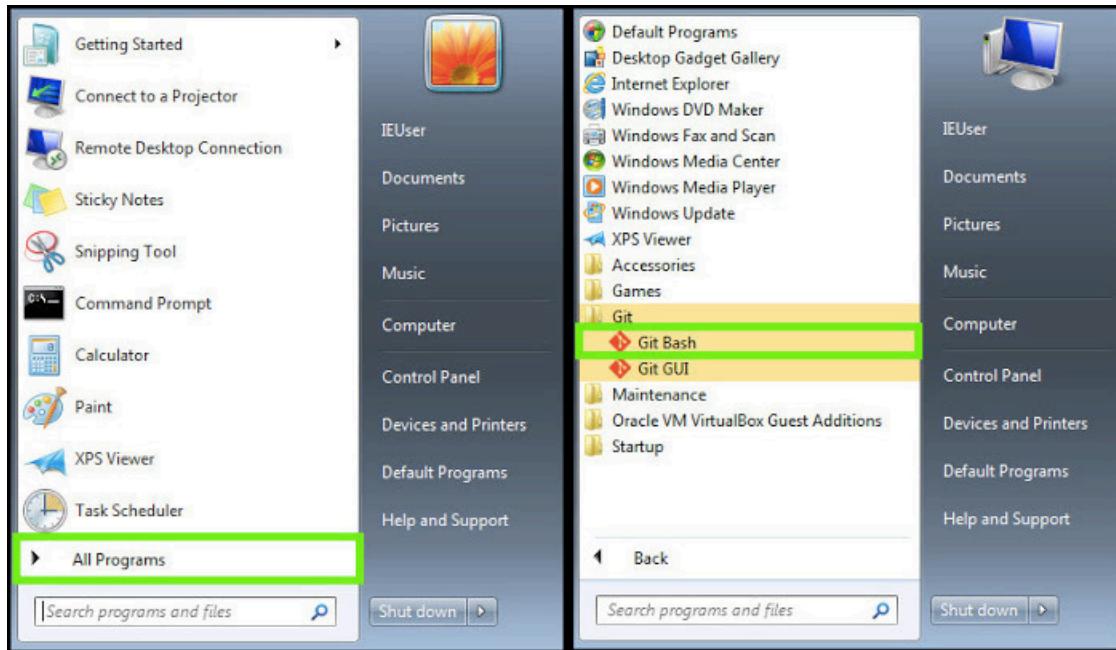
```
homestead -V
Laravel Homestead version 2.0.9
```

Homestead CLI berhasil diinstall

7. Langkah tambahan untuk pengguna Windows. Karena kita akan menggunakan Git dan SSH, silahkan install Git dari <http://msysgit.github.io/>⁴⁰. Setelah terinstall,

⁴⁰<http://msysgit.github.io/>

pastikan dapat menjalankan aplikasi Git Bash.



Membuka Git Bash

Cek juga versi git yang terinstall dengan `git --version`.

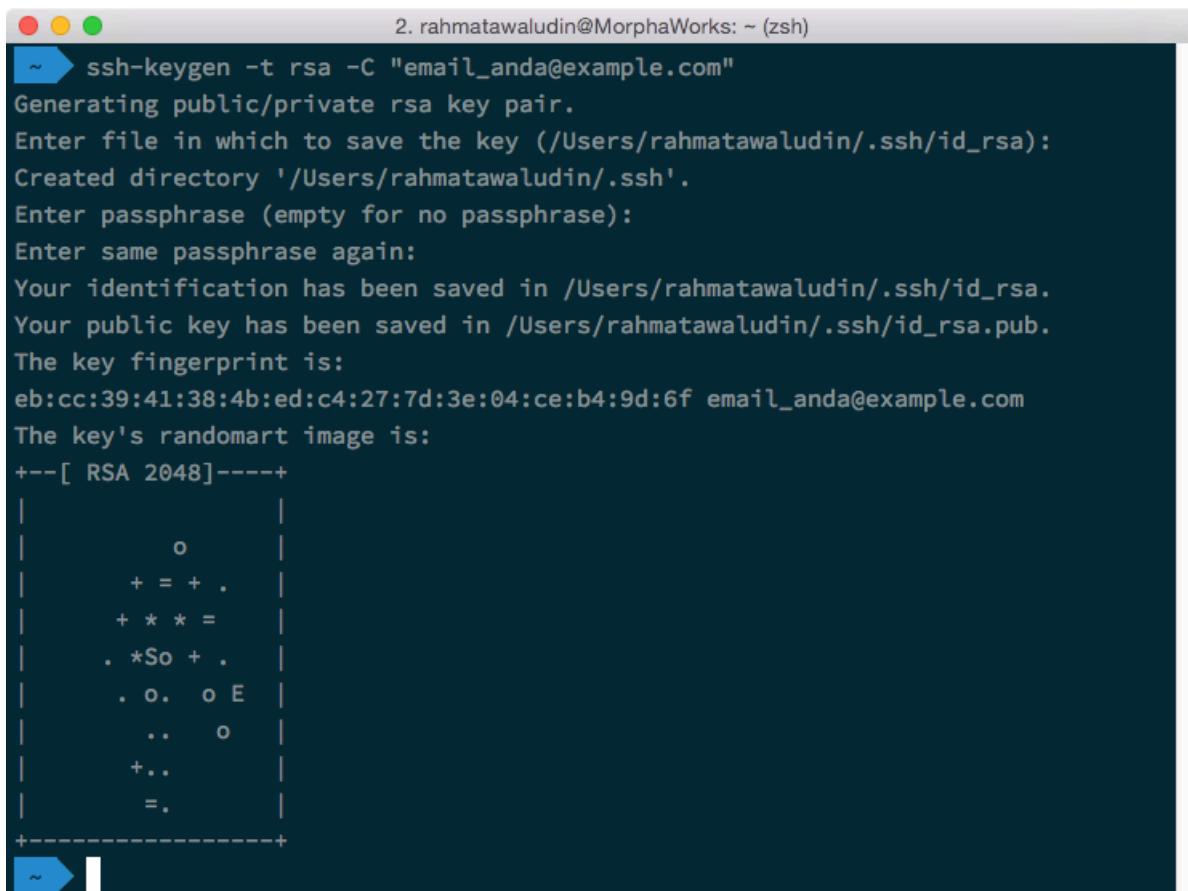
A screenshot of a terminal window titled 'MINGW32:/c/Users/IEUser'. The window shows the command 'git --version' being run and its output: 'git version 1.9.4.msysgit.1'. The terminal has a standard Windows-style window frame with minimize, maximize, and close buttons.

Git Bash telah terinstall

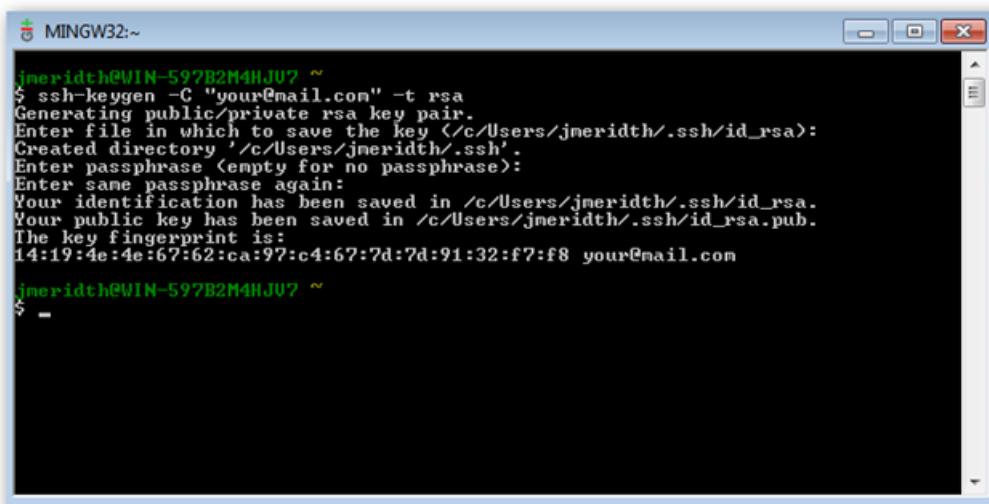
Konfigurasi

Setelah berhasil mendownload dan install tools yang dibutuhkan, kini kita akan melakukan konfigurasi homestead. Silahkan ikuti langkah berikut.

1. Untuk pengguna windows, aktifkan hardware virtualization (VT-x) di BIOS.
2. Cek apakah sudah memiliki SSH key di `~/.ssh` atau `C:\Users\<user>\.ssh`. Jika belum ada, buat SSH key (Windows: Git Bash) dengan perintah `ssh-keygen -t rsa -C "email_anda@example.com"`



```
2. rahmatawaludin@MorphaWorks: ~ (zsh)
$ ssh-keygen -t rsa -C "email_anda@example.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/rahmatawaludin/.ssh/id_rsa):
Created directory '/Users/rahmatawaludin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/rahmatawaludin/.ssh/id_rsa.
Your public key has been saved in /Users/rahmatawaludin/.ssh/id_rsa.pub.
The key fingerprint is:
eb:cc:39:41:38:4b:ed:c4:27:7d:3e:04:ce:b4:9d:6f email_anda@example.com
The key's randomart image is:
+---[ RSA 2048]----+
| |
| o |
| + = + . |
| + * * = |
| . *So + . |
| . o. o E |
| .. o |
| +.. |
| =. |
+-----+
```

Membuat SSH Key


```
MINGW32:~
$ jmeridth@WIN-597B2M4HJU2 ~
$ ssh-keygen -C "your@email.com" -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key </c/Users/jmeridth/.ssh/id_rsa>:
Created directory '/c/Users/jmeridth/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/jmeridth/.ssh/id_rsa.
Your public key has been saved in /c/Users/jmeridth/.ssh/id_rsa.pub.
The key fingerprint is:
14:19:4e:4e:67:62:ca:97:c4:67:7d:7d:91:32:f7:f8 your@email.com
$ -
```

Membuat SSH Key di Windows

3. Selanjutnya silahkan cek folder `~/.ssh` atau `C:\Users\<user>\.ssh` pastikan sudah ada file `id_rsa` dan `id_rsa.pub`.

```
2. rahmatawaludin@MorphaWorks: ~/ssh (zsh)
~ cd .ssh
~/ssh ls
id_rsa id_rsa.pub
~/ssh
```

Mengecek SSH Key

4. Kita perlu membuat konfigurasi default untuk homestead, jalankan perintah homestead init. Perintah ini akan membuat folder baru di `~/homestead` atau `C:\Users\<user>\homestead`.

```
2. rahmatawaludin@MorphaWorks: ~ (zsh)
~ homestead init
Creating Homestead.yaml file... ✓
Homestead.yaml file created at: /Users/rahmatawaludin/.homestead/Homestead.yaml
~
```

Homestead init

5. Selanjutnya akan terbuat 3 file baru,

- `Homestead.yaml` : berisi konfigurasi homestead
- `after.sh` : berisi perintah yang akan dijalankan ketika homestead pertama kali dijalankan (optional)
- `aliases` : berisi alias yang ingin kita set ketika pada homestead (optional)

```
2. rahmatawaludin@MorphaWorks: ~/homestead (zsh)
~ homestead init
Creating Homestead.yaml file... ✓
Homestead.yaml file created at: /Users/rahmatawaludin/.homestead/Homestead.yaml
~ cd .homestead
~/homestead ls
Homestead.yaml after.sh      aliases
~/homestead
```

File homestead

6. Mari kita mulai dengan file `Homestead.yaml`. Isian default nya seperti ini:

Homestead.yaml

```
---
ip: "192.168.10.10"
memory: 2048
cpus: 1

authorize: ~/.ssh/id_rsa.pub

keys:
- ~/.ssh/id_rsa

folders:
- map: ~/Code
  to: /home/vagrant/Code

sites:
- map: homestead.app
  to: /home/vagrant/Code/Laravel/public

databases:
- homestead

variables:
- key: APP_ENV
  value: local
```

Untuk mengubahnya kita dapat langsung membuka file ini atau menjalankan perintah `homestead edit`. Perintah ini akan membuka file `Homestead.yaml` dan menggunakan editor default untuk merubahnya.

Silahkan ubah konfigurasi diatas mengikuti petunjuk berikut:

- **ip** : IP yang akan kita gunakan untuk mengakses homestead (biarkan default).
- **memory** : Besarnya memory yang dialokasikan untuk homestead dalam Mb. Minimal 1024.
- **cpu** : Jumlah core cpu yang diberikan untuk homestead (biarkan default).
- **authorize** : alamat public key ssh (`id_rsa.pub`) kita. Jika langkah 3 sudah selesai biarkan default.
- **keys** : alamat private key ssh (`id_rsa`) kita. Jika langkah 3 sudah selesai biarkan default.
- **folders** : Setiap isian folder disini akan di map pada VM homestead. Di windows, istilahnya map network drive. Jadi, setiap isian **map** (`~/Code`)

akan di petakan/map/mount ke folder di isian **to** di homestead. Kita dapat mengubah dan menambahnya sesuka hati, saya sendiri membiarkannya default, karena semua code saya memang berada di folder `~/Code`. Jika kita menyimpan file project di tempat berbeda, misalnya `C:\xampp\htdocs`, maka ubahlah isian **map**.

- **sites** : Ini adalah alamat URL kita akan mengakses aplikasi Laravel. Pada contoh default isian **map** berisi `homestead.app` yang mengarah ke folder `/home/vagrant/Code/Laravel/public`. Yang artinya, jika kita menyimpan project di `C:\xampp\htdocs` maka URL `homestead.app` akan mengakses project dari folder `C:\xampp\htdocs\Laravel\public`. Kita bebas menambah **sites** sebanyak apapun yang diinginkan. Setelah menambah isian disini, kita juga harus menambah isian di file hosts `/etc/hosts` atau `C:\Windows\System32\Drivers\etc\hosts` (windows). Tambah isian dengan format **ip url**. Misalnya: `192.168.10.10 homestead.app`

Terkadang ketika merubah file hosts di windows akan muncul error "Access is Denied". Solusinya, simpan dulu file hosts ke desktop, copy dan replace file aslinya.

- **databases** : Isian disini akan membuat database di homestead. - **variables** : environment variables yang ingin kita set di homestead.

7. File `after.sh` akan dijalankan jika homestead telah booting. Kita bebas mengisinya. Contohnya, saya pernah menambahkan `max upload` di ngix dengan perintah:

after.sh

```
sudo sed -i 's/post_max_size = 8M/post_max_size = 10M/g' /etc/php5/fpm/php.ini
sudo sed -i 's/upload_max_filesize = 2M/upload_max_filesize = 10M/g' /etc/php5/\fpm/php.ini

# restart service
sudo service nginx restart
sudo service php5-fpm restart
```

8. File `aliases` berisi alias yang ingin kita set ketika homestead dijalankan. Contohnya saya membuat alias `art` untuk perintah `php artisan`.

aliases

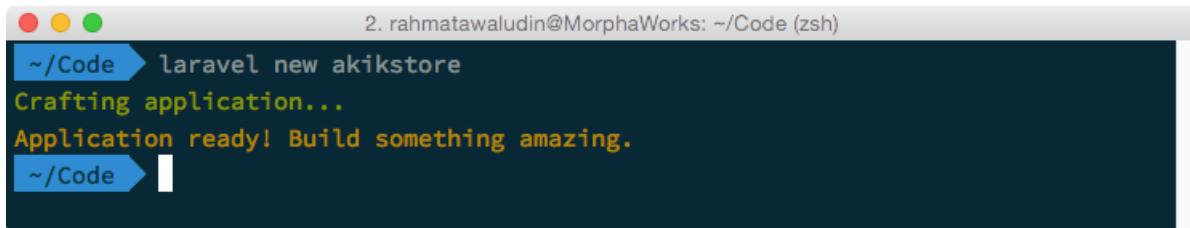
```
.....
alias art='php artisan'
```

Setelah selesai di konfigurasi, mari kita install Laravel di homestead.

Cara Pakai

Mari kita buat aplikasi Laravel baru di homestead. Pada saat tulisan ini dibuat, di Indonesia lagi rame batu akik, mari kita buat web <http://akikstore.dev..> :D

1. Buat project laravel baru pada folder project (~/Code atau C:\xampp\htdocs) misalnya dengan nama akikstore.



```
2. rahmatawaludin@MorphaWorks: ~/Code (zsh)
~/Code ➔ laravel new akikstore
Crafting application...
Application ready! Build something amazing.
~/Code ➔
```

Membuat project baru

2. Tambahkan isian **sites** pada file Homestead.yaml.

Homestead.yaml

```
.....
sites:
  - map: homestead.app
    to: /home/vagrant/Code/Laravel/public
  - map: akikstore.dev
    to: /home/vagrant/Code/akikstore/public
.....
```

3. Tambahkan isian di file hosts (/etc/hosts atau C:\Windows\System32\Drivers\etc\hosts) dengan IP dan URL homestead.

/etc/hosts

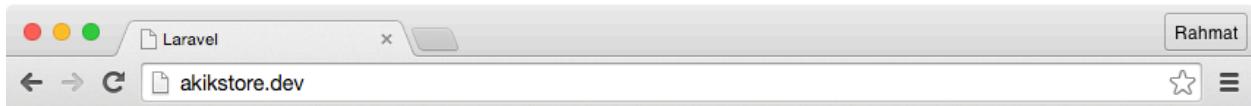
```
192.168.10.10 akikstore.dev
....itunes
```

4. Jalankan homestead dengan perintah homestead up untuk menjalankan homestead.

```
1. homestead up (ruby)
~/Code ➔ homestead up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'laravel/homestead'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'laravel/homestead' is up to date...
==> default: There was a problem while downloading the metadata for your box
==> default: to check for updates. This is not an error, since it is usually due
==> default: to temporary network problems. This is just a warning. The problem
==> default: encountered was:
==> default:
==> default: Couldn't resolve host 'vagrantcloud.com'
==> default:
==> default: If you want to check for box updates, verify your network connectio
n
==> default: is valid and try again.
==> default: Setting the name of the VM: homestead
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 80 => 8000 (adapter 1)
    default: 443 => 44300 (adapter 1)
    default: 3306 => 33060 (adapter 1)
    default: 5432 => 54320 (adapter 1)
```

homestead up

5. Setelah berhasil, kita dapat mengakses web dari <http://akikstore.dev>



Laravel 5

Simplicity is an acquired taste. - Katharine Gerould

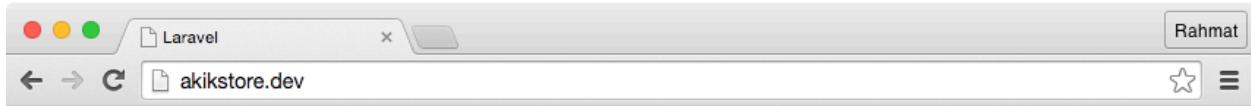
Berhasil setup homestead

Kini setiap perubahan yang kita lakukan di folder project `~/Code/akikstore` akan direfleksikan di homestead. Contohnya saya ubah view untuk welcome

`resources/view/welcome.blade.php`

```
42 ....
43 <div class="title">Akik Store</div>
44 <div class="quote">Temukan keajaiban batu akik disini!</div>
45 ....
```

Refresh halaman, maka akan tampil hasil perubahan yang baru kita buat.



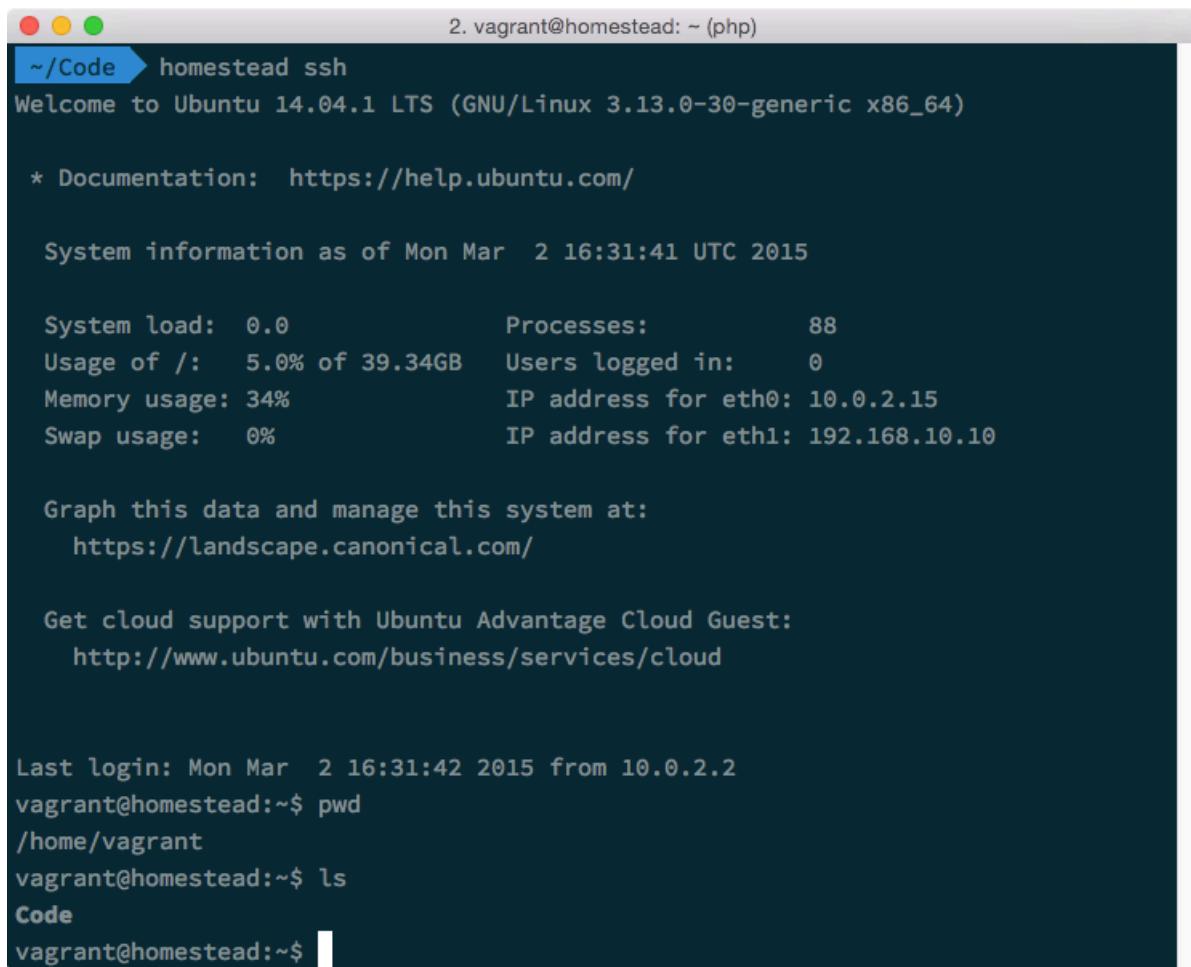
Akik Store

Temukan keajaiban batu akik disini!

Berhasil melakukan perubahan

Akses SSH

Untuk mengakses homestead kita perlu menggunakan SSH. Untuk pengguna windows dapat menggunakan `gitbash` yang sudah diinstall pada langkah sebelumnya. Untuk mengakses SSH, jalankan homestead `ssh`.



```
2. vagrant@homestead: ~ (php)
~/Code ➔ homestead ssh
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-30-generic x86_64)

 * Documentation: https://help.ubuntu.com/

System information as of Mon Mar  2 16:31:41 UTC 2015

System load:  0.0          Processes:           88
Usage of /:   5.0% of 39.34GB  Users logged in:    0
Memory usage: 34%          IP address for eth0: 10.0.2.15
Swap usage:   0%          IP address for eth1: 192.168.10.10

Graph this data and manage this system at:
https://landscape.canonical.com/

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

Last login: Mon Mar  2 16:31:42 2015 from 10.0.2.2
vagrant@homestead:~$ pwd
/home/vagrant
vagrant@homestead:~$ ls
Code
vagrant@homestead:~$
```

Homestead SSH

Terlihat disini homestead menggunakan Ubuntu 14.04. Setelah mengakses SSH, kita dapat mengakses folder yang telah di map, misalnya folder akikstore kita di /home/vagrant/Code/akikstore.

```
1. vagrant@homestead: ~/Code/akikstore (php)

141 packages can be updated.
67 updates are security updates.

Last login: Mon Mar 16 09:52:43 2015 from 10.0.2.2
vagrant@homestead:~$ ls
Code
vagrant@homestead:~$ pwd
/home/vagrant
vagrant@homestead:~$ cd Code/akikstore/
vagrant@homestead:~/Code/akikstore$ ls
app      composer.json  database      phpspec.yml  readme.md   storage
artisan  composer.lock  gulpfile.js  phpunit.xml  resources   tests
bootstrap config        package.json  public       server.php  vendor
vagrant@homestead:~/Code/akikstore$ 
```

Mengecek folder project di Homestead

Dari sini, kita bisa menjalankan berbagai perintah artisan.

```
1. vagrant@homestead: ~/Code/akikstore (php)

vagrant@homestead:~/Code/akikstore$ ls
app      composer.json  database      phpspec.yml  readme.md   storage
artisan  composer.lock  gulpfile.js  phpunit.xml  resources   tests
bootstrap config        package.json  public       server.php  vendor
vagrant@homestead:~/Code/akikstore$ php artisan env
Current application environment: local
vagrant@homestead:~/Code/akikstore$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> config('app.debug')
=> true
>>> config('cache.default')
=> "file"
>>> echo "Anda keren!"
Anda keren! ↵
=> null
>>> 
```

Menjalankan Artisan di Homestead

Port

Berikut beberapa port yang di-forward oleh homestead berikut port aslinya:

- **SSH:** 2222 ↳ Forwards To 22
- **HTTP:** 8000 ↳ Forwards To 80
- **MySQL:** 33060 ↳ Forwards To 3306
- **Postgres:** 54320 ↳ Forwards To 5432
- **username :** homestead
- **password :** secret

Bisa dilihat, misalnya untuk mysql jika kita menggunakan homestead ssh maka menggunakan port 3306. Jika dari host, kita gunakan 33060.

Jika diinginkan kita juga dapat mem-forward port lain ke host dengan menambah isian berikut di Homestead.yaml. Misalnya forward port 10000 (host) ke 1000 (homestead):

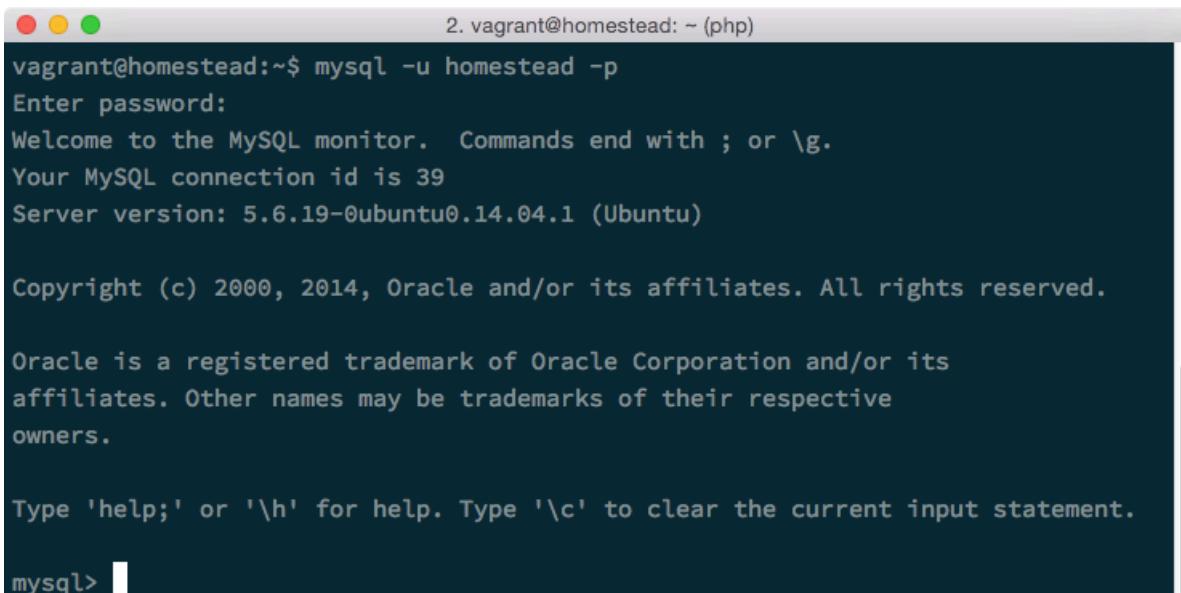
Homestead.yaml

```
....  
ports:  
  - send: 10009  
    to: 1000  
....
```

Akses Database

Untuk mengakses homestead ada beberapa cara. Berikut saya contohkan untuk mengakses database mysql:

1. Menggunakan homestead ssh dan mengakses langsung di dalam homestead.



The screenshot shows a terminal window titled "2. vagrant@homestead: ~ (php)". The user is connected to a MySQL monitor. The session starts with:

```
vagrant@homestead:~$ mysql -u homestead -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 39  
Server version: 5.6.19-0ubuntu0.14.04.1 (Ubuntu)
```

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

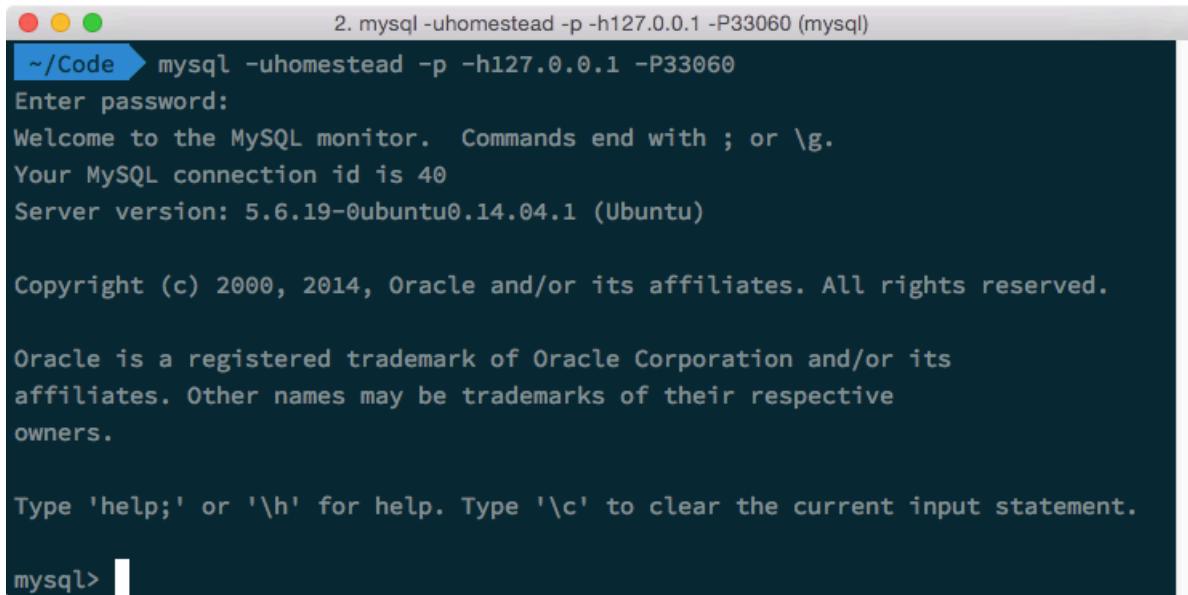
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

homestead mysql cara 1

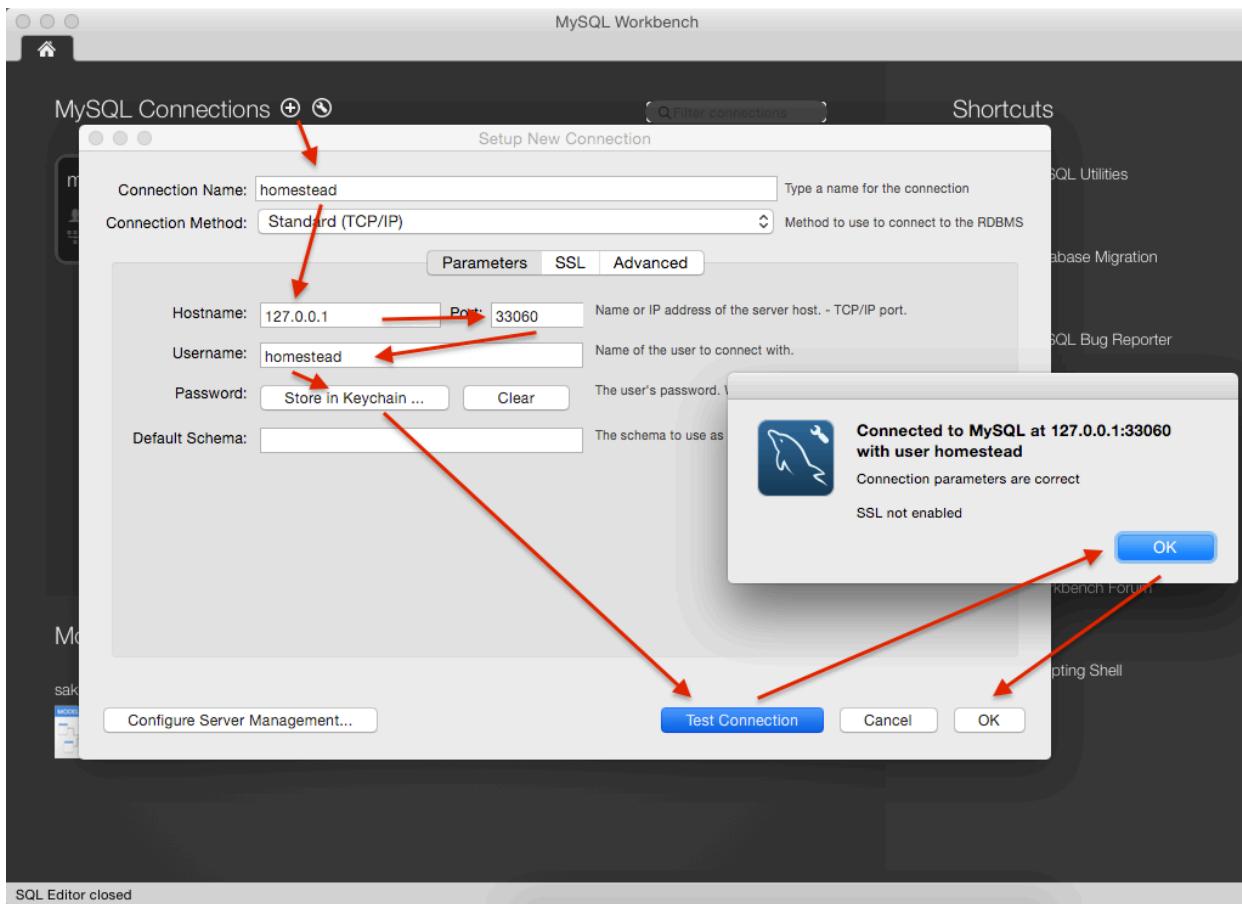
2. Menggunakan mysql dari host. Gunakan port 33060 dan IP 127.0.0.1

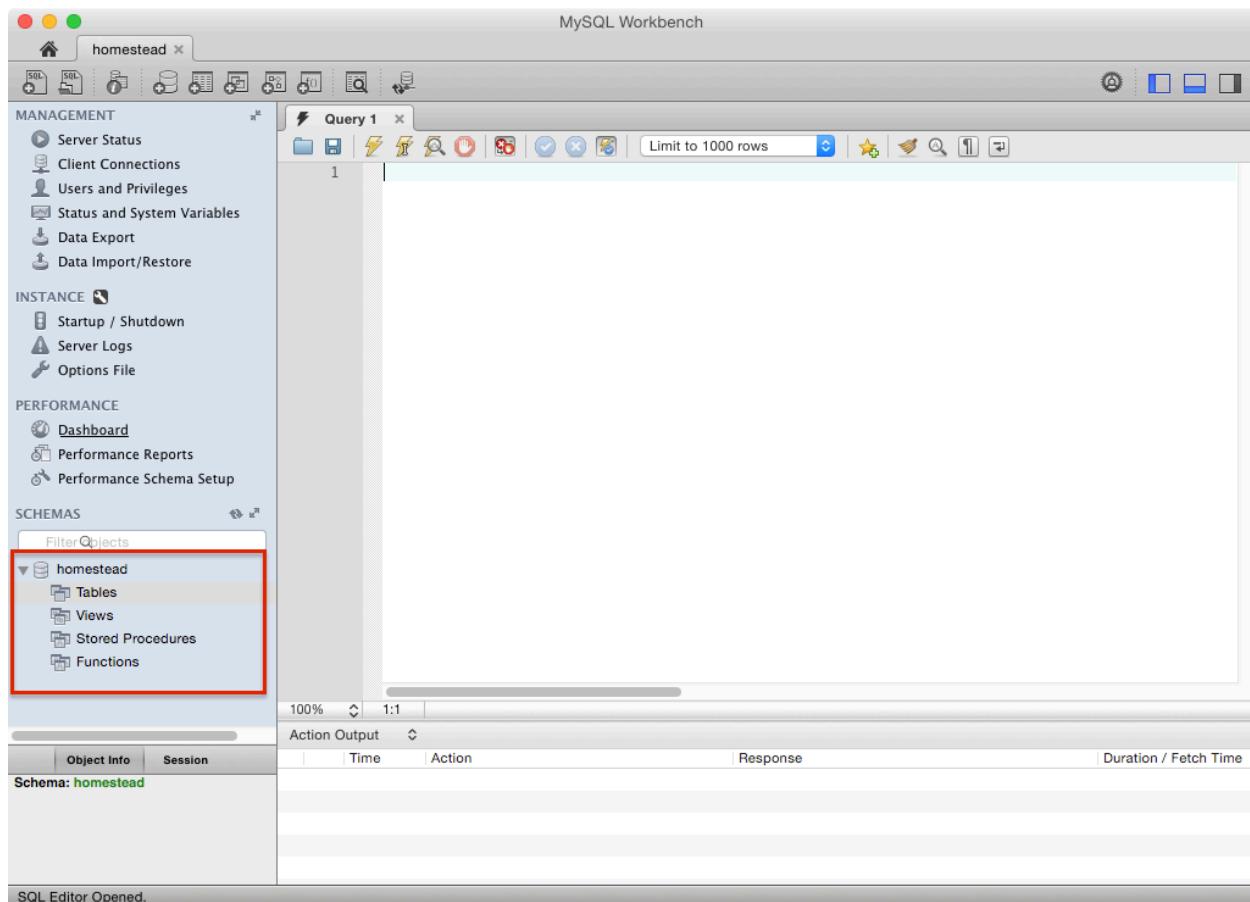


The screenshot shows a terminal window with the title "2. mysql -uhomestead -p -h127.0.0.1 -P33060 (mysql)". The command entered is "mysql -uhomestead -p -h127.0.0.1 -P33060". The MySQL monitor prompt "Enter password:" is displayed. The welcome message includes the connection id (40), server version (5.6.19-0ubuntu0.14.04.1 (Ubuntu)), copyright information (Oracle and/or its affiliates, 2000-2014), and trademark information (Oracle is a registered trademark). It also provides help instructions ("Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.") and ends with the MySQL prompt "mysql>".

homestead mysql cara 2

3. Menggunakan aplikasi GUI dari host, misalnya MySQLWorkbench. Pastikan mengisi port dengan 33060 dan IP 127.0.0.1.

**Homestead cara 3**

**Homestead cara 3**

e

Artisan CLI

Untuk memudahkan selama mengembangkan aplikasi, laravel menyediakan interface command line bernama Artisan. Untuk mengakses fitur ini, kita dapat menggunakan terminal dan mengarahkan ke folder project Laravel kemudian mengetik `php artisan`.

```
1. rahmatawaludin@MorphaWorks: ~/Code/laravel-5 (zsh)
~/Code/laravel-5 ➤ php artisan
Laravel Framework version 5.0.13

Usage:
[options] command [arguments]

Options:
--help (-h)          Display this help message
--quiet (-q)         Do not output any message
--verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal output,
2 for more verbose output and 3 for debug
--version (-V)        Display this application version
--ansi                Force ANSI output
--no-ansi              Disable ANSI output
--no-interaction (-n) Do not ask any interactive question
--env                 The environment the command should run under.

Available commands:
clear-compiled      Remove the compiled class file
down                Put the application into maintenance mode
env                 Display the current framework environment
fresh               Remove the scaffolding included with the framework
help                Displays help for a command
inspire             Display an inspiring quote
list                Lists commands
```

php artisan

Terlihat disini banyak perintah disediakan oleh Artisan. Untuk menampilkan bantuan terhadap suatu perintah, jalankan `php artisan help <nama_perintah`. Misalnya, `php artisan help app:name`

```
1. rahmatawaludin@MorphaWorks: ~/Code/laravel-5 (zsh)
~/Code/laravel-5 ➤ php artisan help app:name
Usage:
  app:name name

Arguments:
  name          The desired namespace.

Options:
  --help (-h)      Display this help message
  --quiet (-q)     Do not output any message
  --verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal output,
  2 for more verbose output and 3 for debug
  --version (-V)   Display this application version
  --ansi           Force ANSI output
  --no-ansi         Disable ANSI output
  --no-interaction (-n) Do not ask any interactive question
  --env             The environment the command should run under.

~/Code/laravel-5 ➤
```

php artisan help app:name

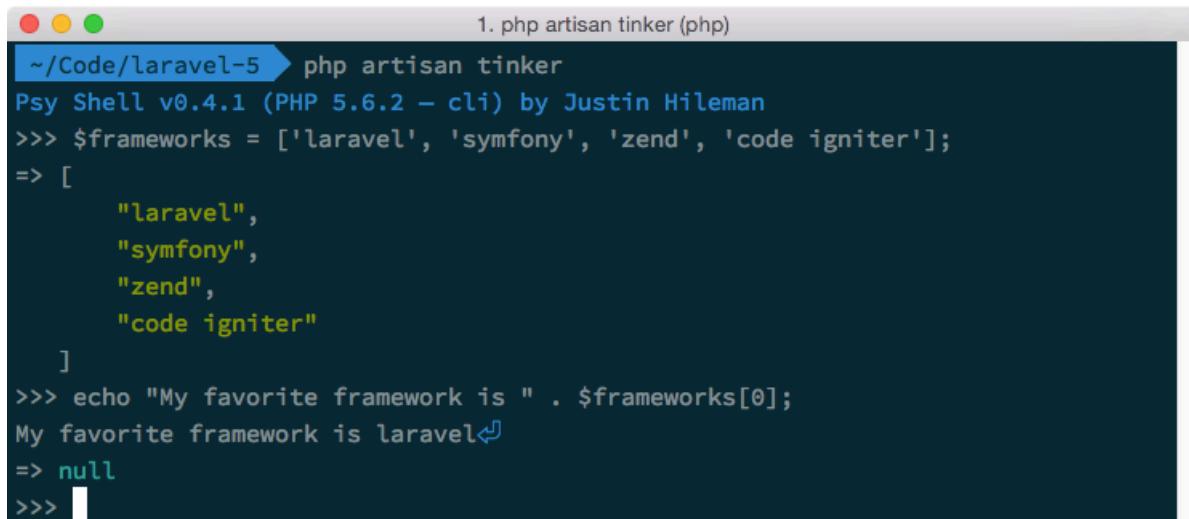
Terlihat disini perintah `app:name` menerima parameter berupa namespace dari aplikasi kita. Akan kita bahas di bab selanjutnya.

Kita juga dapat melihat versi Laravel yang sedang kita pakai dengan perintah `php artisan --version`.

```
1. rahmatawaludin@MorphaWorks: ~/Code/laravel-5 (zsh)
~/Code/laravel-5 ➤ php artisan --version
Laravel Framework version 5.0.13
~/Code/laravel-5 ➤
```

php artisan --version

Kedepannya kita akan sering menggunakan Artisan. Salah satu fitur Artisan yang akan sering kita pakai adalah `tinker`. Tinker memudahkan kita untuk berinteraksi dengan aplikasi yang sedang dibangun, misalnya menjalankan script php langsung dari terminal. Berikut contohnya:



```
1. php artisan tinker (php)
~/Code/laravel-5 ➔ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.2 – cli) by Justin Hileman
>>> $frameworks = ['laravel', 'symfony', 'zend', 'code igniter'];
=> [
    "laravel",
    "symfony",
    "zend",
    "code igniter"
]
>>> echo "My favorite framework is " . $frameworks[0];
My favorite framework is laravel
=> null
>>> 
```

php artisan tinker

Environment

Dalam mengembangkan aplikasi dengan Laravel kita akan sering mendengar istilah **environment**. Maksud dari environment ini adalah mesin/lingkungan tempat Laravel sedang berjalan. Mari kita pelajari lebih lanjut.

Memahami .env

Laravel menggunakan [.env](#)⁴¹ untuk memudahkan konfigurasi di project dengan banyak developer. Secara sederhana, dengan menggunakan .env, kita dapat mengakses nilai dari file **.env** dengan menggunakan syntax `env('KEY', 'default-value')`.

Mari kita ambil contoh sederhana. Ketika project Laravel pertama kali dibuat (menggunakan `composer create-project`) ia akan menyalin file **.env.example** ke file **.env**. Isinya seperti berikut:

⁴¹ <https://github.com/vlucas/phpdotenv>

.env

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=yS0P1XZCLY6yhVyK1Krf00u72cE6KS8

DB_HOST=localhost
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

CACHE_DRIVER=file
SESSION_DRIVER=file
```

Misalnya kita mau mengambil nilai dari DB_DATABASE dengan nilai default 'db_akik', maka cara memanggilnya `env('DB_DATABASE', 'db_akik')`. Gitu.

Nah, Laravel menggunakan file .env ini untuk mengisi nilai konfigurasinya. Contohnya untuk konfigurasi database di file config/database.php, kita akan menemukan baris semacam ini:

config/database

```
54 ....
55 'mysql' => [
56     'driver'    => 'mysql',
57     'host'      => env('DB_HOST', 'localhost'),
58     'database'  => env('DB_DATABASE', 'forge'),
59     'username'  => env('DB_USERNAME', 'forge'),
60     'password'  => env('DB_PASSWORD', ''),
61     'charset'   => 'utf8',
62     'collation' => 'utf8_unicode_ci',
63     'prefix'    => '',
64     'strict'   => false,
65 ],
66 ....
```

Semua nilai konfigurasi database itu didapatkan dari file **.env**. Kalau mau dirubah, ada dua cara:

1. Ubah isian di .env untuk key yang diinginkan.
2. Ubah langsung di sini.

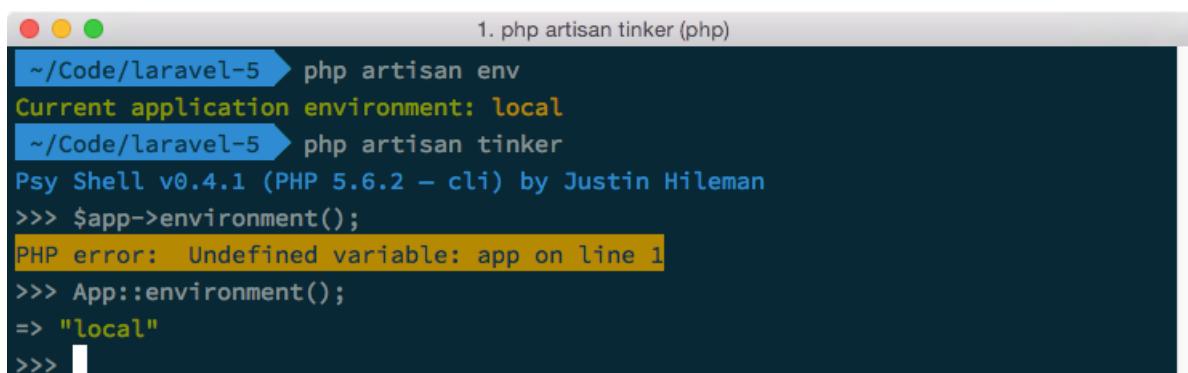
Saya sendiri menyukai cara pertama. Kenapa? Berikut beberapa alasannya:

1. File .env tidak akan di commit ke source code, artinya tiap orang bisa memiliki konfigurasi yang berbeda tanpa mempengaruhi source code.
2. Menyimpan hal yang sensitif seperti password memang sebaiknya tidak di dalam source code yang akan di commit. Bahaya. [Baca ini](#)⁴².
3. Semua isian dari .env akan masuk ke variable \$_ENV. Jadi, bisa saya akses dimanapun di aplikasi.

Memahami Environment

Laravel menggunakan environment default sebagai **production**. Ini bisa kita ubah dengan menambah key **APP_ENV** di file **.env**. Semua logicnya bisa kita lihat [disini](#)⁴³.

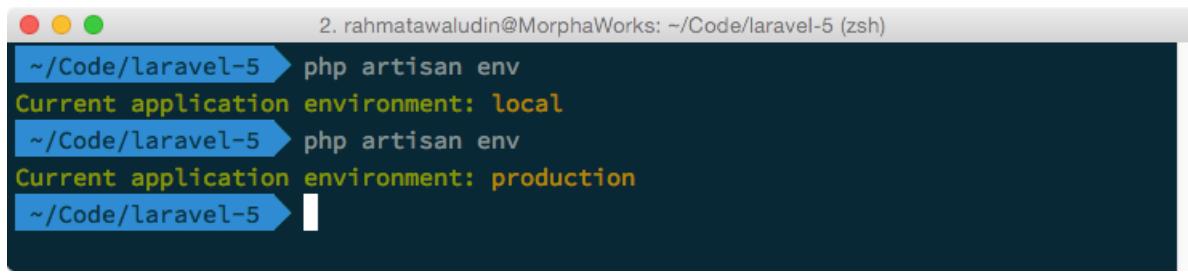
Untuk mengecek environment yang sedang aktif kita dapat menggunakan `php artisan env`. Kita juga dapat mengakses environment yang aktif dari aplikasi dengan perintah `$app->environment()` atau `App::environment()`.



```
1. php artisan tinker (php)
~/Code/laravel-5 ➔ php artisan env
Current application environment: local
~/Code/laravel-5 ➔ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.2 – cli) by Justin Hileman
>>> $app->environment();
PHP error: Undefined variable: app on line 1
>>> App::environment();
=> "local"
>>>
```

app environment

Jika saya menghapus isian APP_ENV di file .env, ini yang akan terjadi:



```
2. rahmatawaludin@MorphaWorks: ~/Code/laravel-5 (zsh)
~/Code/laravel-5 ➔ php artisan env
Current application environment: local
~/Code/laravel-5 ➔ php artisan env
Current application environment: production
~/Code/laravel-5 ➔
```

php artisan env

⁴²<http://www.12factor.net/config>

⁴³<https://github.com/laravel/framework/blob/5.0/src/Illuminate/Foundation/Bootstrap/DetectEnvironment.php>

Jadi, ketika kita publish aplikasi ke hosting, cukup membuat file .env baru dengan semua isiannya disana. Tanpa harus merubah isian di file konfigurasi. Simple kan?

Memahami Konfigurasi

Semua konfigurasi Laravel ada di folder **config**. Ada beberapa file disini:

- **app.php** : berisi konfigurasi tentang aplikasi, dari mulai debug mode, log, service provider, alias, dll.
- **auth.php** : berisi konfigurasi authentikasi default dari Laravel.
- **cache.php** : berisi konfigurasi tentang cache.
- **compile** : berisi tambahan class yang akan di compile (agar load lebih cepat) ketika menjalankan perintah `php artisan optimize`.
- **database.php** : berisi konfigurasi database. Secara default Laravel mendukung sqlite, postgresql, mysql dan sql server.
- **filesystems.php** : berisi konfigurasi filesystem. Secara default Laravel mendukung media penyimpanan lokal, s3 dan rackspace.
- **mail.php** : berisi konfigurasi email. Secara default Laravel menggunakan server smtp, fungsi mail php, mailgun, mandril dan log (untuk menyimpan isi email yang terkirim ke storage/logs).
- **queue.php** : berisi konfigurasi antrian (queue). Secara default Laravel mendukung pengimplementasian queue menggunakan sync, database, beanstalkd, sqs, iron dan redis.
- **services.php** : berisi konfigurasi misalnya token untuk third party service yang kita gunakan.
- **session.php** : berisi konfigurasi session. Laravel mendukung media penyimpanan berupa file, cookie, database, apc, memcached, redis dan array.
- **view.php** : berisi konfigurasi untuk view, yaitu lokasi file view dan lokasi file view hasil compile.

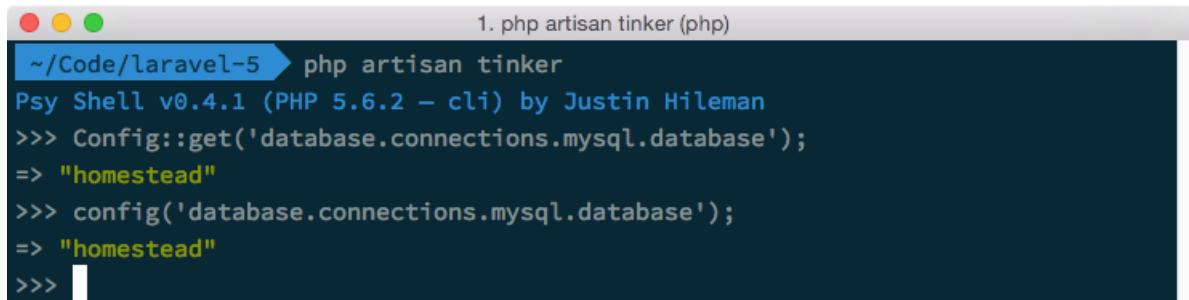


Baca dokumentasi

Setiap file konfigurasi yang saya sebutkan diatas memiliki dokumentasi di source codenya. Silahkan dibaca, dengan membacanya pasti dapat sesuatu yang baru. Percaya deh.

Mengakses nilai konfigurasi

Setiap nilai konfigurasi pada file yang terdapat di folder config dapat kita akses dari aplikasi. Cara mengaksesnya juga mudah, cukup menggunakan `Config::get('nama_file.key')` atau menggunakan helper `config('nama_file.key')`. Misalnya jika kita ingin mengakses nama database mysql yang sedang aktif:



```
1. php artisan tinker (php)
~/Code/laravel-5 ➔ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.2 - cli) by Justin Hileman
>>> Config::get('database.connections.mysql.database');
=> "homestead"
>>> config('database.connections.mysql.database');
=> "homestead"
>>>
```

Mengakses nama database mysql

Membuat konfigurasi Custom

Terkadang ada kondisi dimana kita harus menyimpan konfigurasi yang spesifik untuk aplikasi kita. Untuk itu, kita bisa membuatnya. Misalnya, saya ingin menyimpan konfigurasi tentang theme dengan key berupa nama (*name*) theme dan foldernya (*path*). Berikut cara membuatnya.

1. Buat file baru dengan nama `theme.php` di folder config.
2. Isi `theme.php` dengan isian berikut:

config/theme.php

```
<?php
return [
    /** Theme Name */
    'name' => 'lumut',
    /** Theme Path */
    'path' => base_path('resources/themes/lumut')
];
```

3. Untuk mengakses isian dari konfigurasi theme ini kita dapat mengakses dengan nama file dan key nya:



```
1. php artisan tinker (php)
~/Code/laravel-5 ➔ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.2 - cli) by Justin Hileman
>>> config('theme.name');
=> "lumut"
>>> config('theme.path');
=> "/Users/rahmatawaludin/Code/laravel-5/resources/themes/lumut"
>>>
```

mengakses custom config

Mode Debug

Dalam mengembangkan aplikasi, kemampuan membaca error adalah hal penting. Di awal-awal memang bikin frustasi. Tapi, seiring kita kemampuan coding kita yang terus berkembang, melihat aplikasi error selama pengembangan adalah hal yang biasa. Justru, ketika ada fitur yang tidak jalan tapi tidak ada error yang muncul, malah itulah yang lebih membuat frustasi.

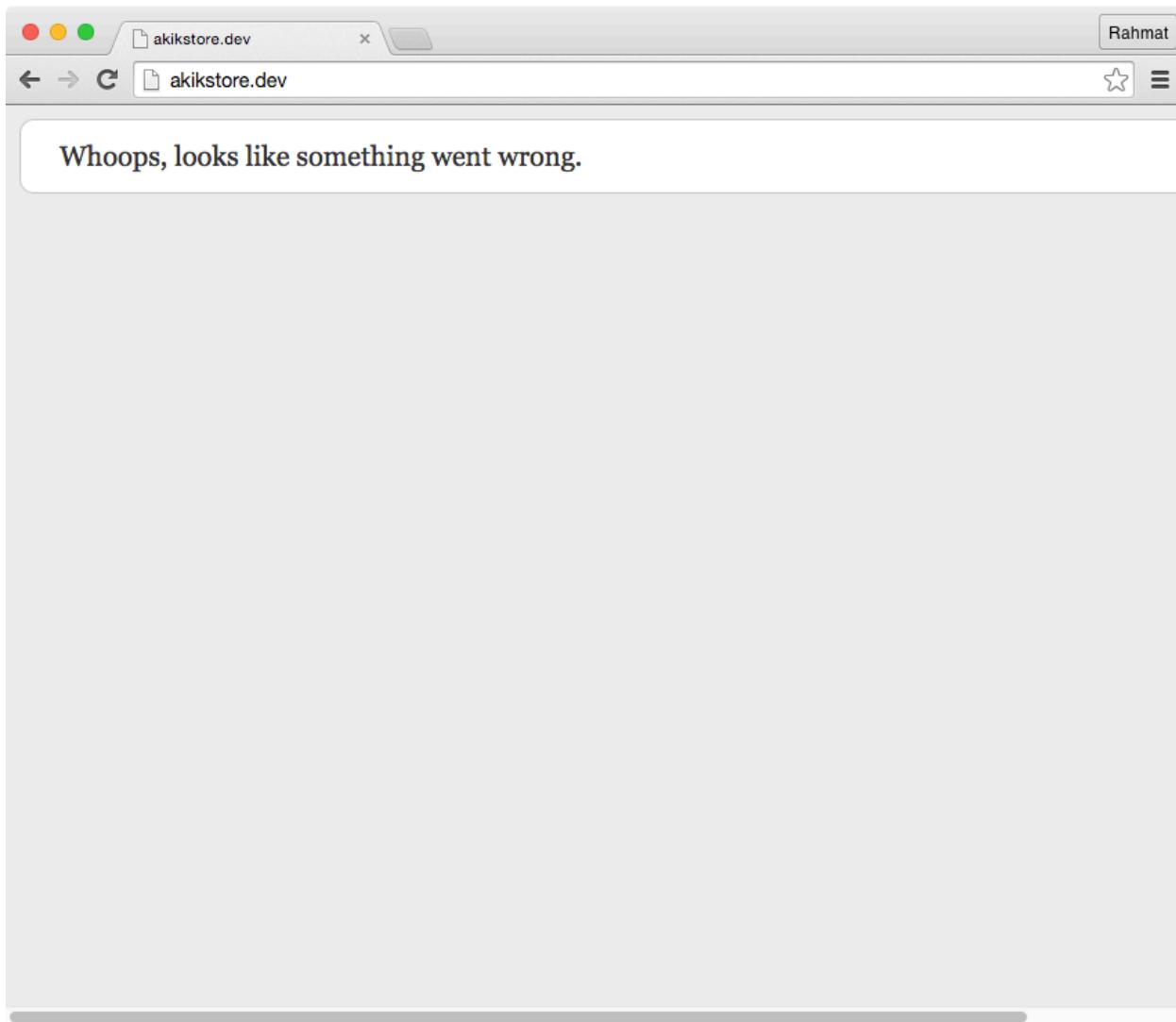
Mari kita coba buat aplikasi kita jadi error dengan mengubah isian

Route::get('/', 'WelcomeController@index');
menjadi
Route::get('/', 'WelcomeController@inde');
di app/Http/routes.php.

app/Http/routes.php

```
13 ....
14 Route::get('/', 'WelcomeController@index');
15 ....
```

Perubahan ini akan membuat URL root (<http://akikstore.dev>) aplikasi kita menjadi error. Mari kita cek..



error tidak muncul

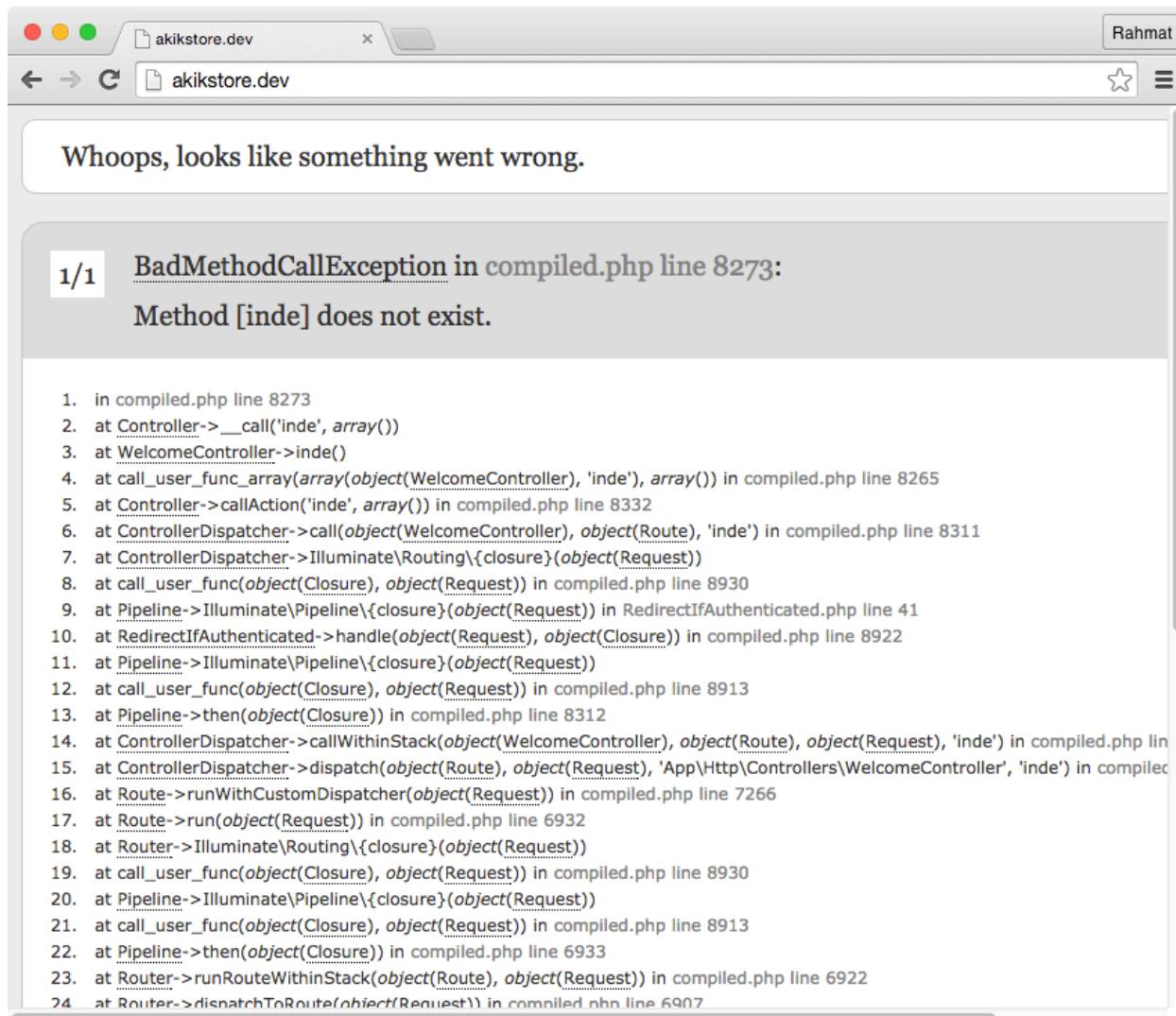
Upss.. errornya tidak muncul. Ini terjadi karena kita telah mengubah environment menjadi `production` dengan menghapus key `APP_ENV` di file `.env`. Ketika kita menghapus key `APP_ENV`, maka Laravel tidak akan menggunakan key yang lainnya di file `.env`.

Maka, secara default mode debug yang aktif adalah false. Mari kita tambahkan kembali..

`.env`

```
APP_ENV=local
APP_DEBUG=true
...
```

Ketika di refresh akan muncul error seperti berikut:



error muncul

Terlihat disini, kita mencoba memanggil method `inde` padahal method tersebut tidak ada.

Mode Maintenance

Terkadang kita harus mematikan sementara aplikasi yang kita bangun untuk melakukan maintenance rutin, misalnya melakukan backup. Untuk itu, Laravel menyediakan fitur maintenance. Fitur ini bisa diaktifkan dengan perintah `php artisan down`.

```
1. rahmatawaludin@MorphaWorks: ~/Code/akikstore (zsh)
~/Code/akikstore ➔ php artisan down
Application is now in maintenance mode.
~/Code/akikstore ➔
```

php artisan down

Ketika kita mencoba mengakses aplikasi, akan muncul:



Be right back.

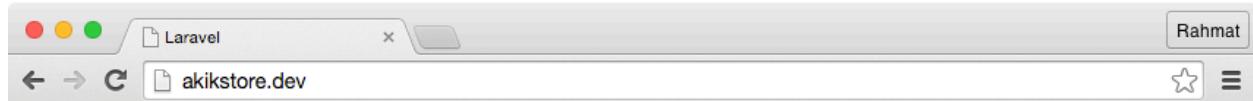
Mode maintenance aktif

Untuk menonaktifkan mode maintenance, gunakan perintah `php artisan up`.

```
1. rahmatawaludin@MorphaWorks: ~/Code/akikstore (zsh)
~/Code/akikstore ➔ php artisan up
Application is now live.
~/Code/akikstore ➔
```

php artisan up

Maka aplikasi dapat kembali diakses:

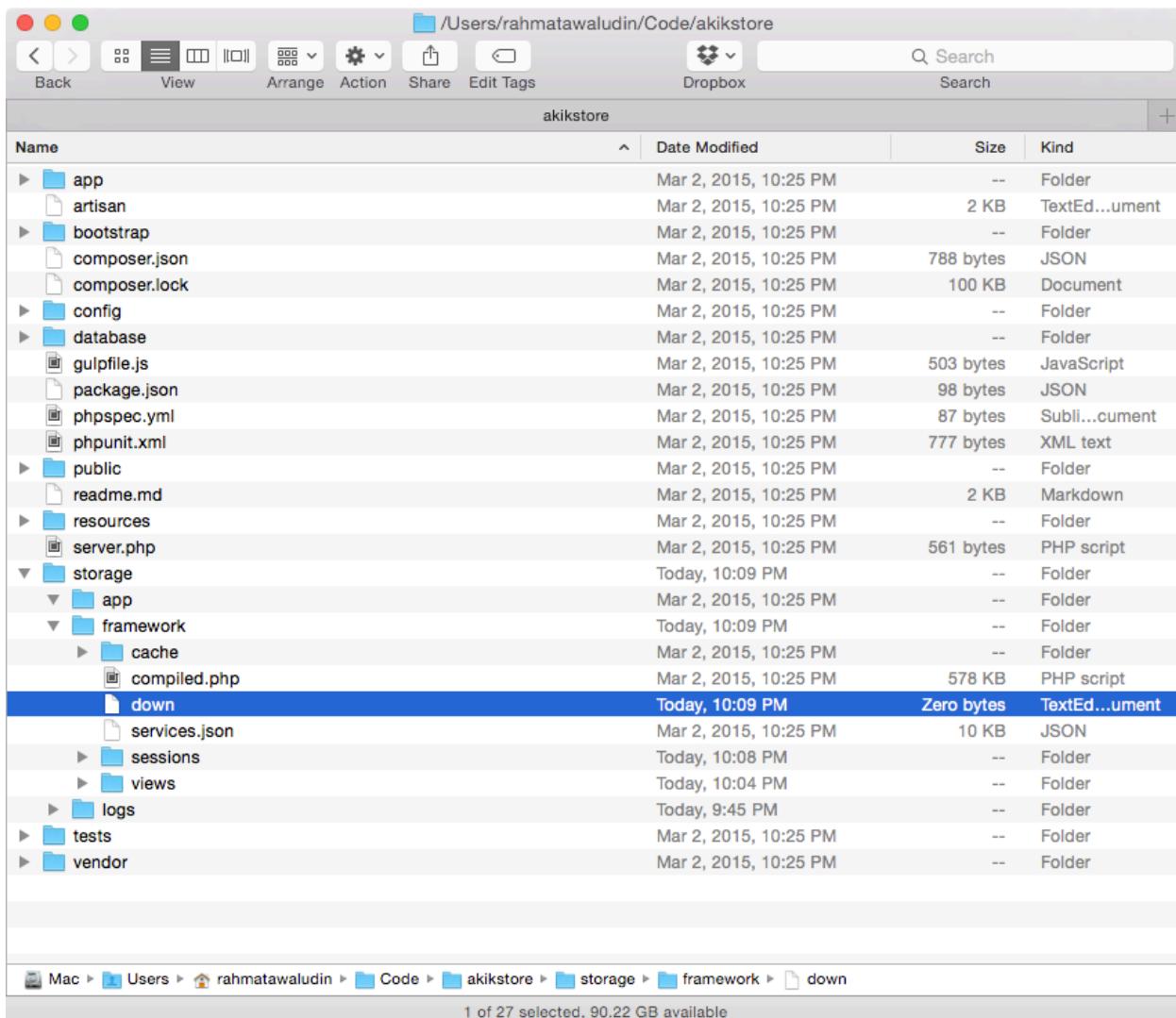


Akik Store

Temukan keajaiban batu akik disini!

Mode Maintenance non-aktif

Logic dari fitur ini sangat sederhana, ketika kita menjalankan perintah `php artisan down` Laravel akan membuat file bernama **down** di `storage/framework`.

**file down**

Ketika perintah `php artisan up` dijalankan, laravel akan menghapus file tersebut. Tentunya, file tersebut bisa kita buat / hapus manual. Ini sangat berguna jika kita menggunakan Laravel di shared hosting, dimana kita tidak memiliki akses ke terminal.

Cara kerjanya juga sederhana. Setiap kali aplikasi di akses, laravel akan mengecek keberadaan file ini. Jika file ditemukan dia akan menampilkan tampilan maintenance default dari view di `resources/view/errors/503.blade.php`. Tentunya tampilan maintenance ini dapat kita ubah dengan sesuai keinginan.



Ringkasan

Pada bab ini kita telah mempelajari tahapan instalasi dan konfigurasi di Laravel. Penggunaan homestead sangat disarankan bagi pengguna Laravel. Jika masih belum pakai, sekarang saat yang tepat untuk menginstallnya. Pada bab-bab selanjutnya, saya akan mengasumsikan Anda menggunakan homestead untuk mengembangkan aplikasi.

Selanjutnya, kita akan membahas Arsitektur dari Laravel. Semangat!

Arsitektur Laravel

PHP adalah bahasa pemrograman yang cukup tua. Banyak yang suka karena mudahnya membuat aplikasi, ada pula yang membencinya. Yap, sebagai seorang developer PHP, saya sudah terlalu sering mendengar banyak orang yang menjelek-jelakan PHP. Biasanya saya acuhkan saja.

Masalah terbesar dari orang yang menjelek-jelakan PHP adalah mereka tidak mengikuti perkembangannya. Mereka tidak perduli (atau tidak mau perduli) bahwa PHP berkembang dengan sangat cepat. Apalagi 5 tahun belakangan ini.

PHP sudah banyak berubah. Komunitas PHP kini sangat aktif. Banyak fitur baru ditambahkan di PHP (diantaranya yang saya jelaskan di bab pertama). Begitupun dengan tools, composer sangat membantu mengatur dependency library dari aplikasi yang kita kembangkan. PHP-Fig (www.php-fig.org), memudahkan standarisasi code (ya, PHP sudah punya standarisasi kode). Ada pula website seperti PHP The Right Way (www.phptherightway.com) yang membantu programmer PHP baru untuk memahami coding PHP modern (wajib baca kalau belum pernah).

Ya, PHP sudah banyak berubah. *BTW, kenapa saya menceritakan semua ini? :D*

Oke, oke, ini ada hubungannya dengan topik kita di bab ini. Laravel, sebagai salah satu framework populer saat ini, menggunakan banyak sekali fitur modern dari PHP. Dengan mempelajari struktur Laravel di bab ini, kita juga akan memahami struktur sebuah aplikasi PHP yang lebih **modern**.

Struktur Aplikasi

Banyaknya folder Laravel terkadang membuat bingung developer yang baru pertama kali mempelajarinya. Tapi, jangan takut. Mari kita pelajari di bab ini.

Folder Dasar

Berikut ini struktur folder awal sebuah aplikasi Laravel:

- **app** : kebanyakan logic dari aplikasi yang kita bangun ada di folder ini. Detailnya akan dijelaskan lebih lanjut.
- **bootstrap** : file di folder ini akan berjalan ketika kita menginisiasi Laravel.
- **config** : sesuai namanya, menyimpan semua konfigurasi aplikasi Laravel.

- **database** : berisi file migration dan seeder untuk aplikasi Laravel. Migration dan seeder akan kita bahas lebih lanjut di bab selanjutnya.
- **public** : folder ini yang diakses oleh pengunjung kita. Jika kita menghosting Laravel, maka root dari aplikasi diarahkan ke folder ini. Semua file yang disimpan di folder ini dapat diakses oleh pengunjung.
- **resources** : berisi semua resource untuk frontend. Folder ini akan kita bahas lebih lanjut ketika pembahasan tentang View dan Laravel Elixir.
- **storage** : berisi compiled file dan logs dari aplikasi. Bayangkan folder ini sebagai folder privat tempat menampung file dari aplikasi yang tidak bisa diakses langsung oleh pengunjung. Misalnya kita mengupload file yang hanya bisa diakses oleh user yang sudah login, maka disini tempat menyimpannya.
- **tests** : sesuai namanya, berisi semua test yang kita buat untuk aplikasi.
- **vendor** : berisi semua library yang telah diinstal dengan composer.

Namespacing

Mari kita pahami lebih lanjut tentang folder `app`. Dalam mengembangkan aplikasi Laravel, kita akan sangat sering bersentuhan dengan folder ini. Sebagai sebuah framework modern, Laravel menggunakan [standar PSR-4⁴⁴](#) untuk melakukan autoloading (kalau belum paham autoload, buka kembali bab pertama). Caranya, secara default Laravel akan membuat *namespace App* untuk folder app.

kita dapat melihat ini pada file `composer.json`:

`composer.json`

```
13 ....
14 "autoload": {
15     "classmap": [
16         "database"
17     ],
18     "psr-4": {
19         "App\\": "app/"
20     }
21 },
22 ....
```

Namespace `App` ini secara default digunakan untuk semua file didalam folder `App`. Mari kita lihat beberapa contohnya:

⁴⁴<http://www.php-fig.org/psr/psr-4>

app/User.php

```
<?php namespace App;  
  
use Illuminate\Auth\Authenticatable;  
....
```

app/Http/Controllers/HomeController.php

```
<?php namespace App\Http\Controllers;  
  
class HomeController extends Controller {  
....
```

app/Services/Registrar.php

```
<?php namespace App\Services;  
  
use App\User;  
....
```

Terlihat pada beberapa contoh file diatas, namespace yang digunakan akan sesuai dengan lokasi file pada file system. Misalnya kita membuat file baru di `App/Entity/-Books.php`, maka namespace filenya harus seperti ini:

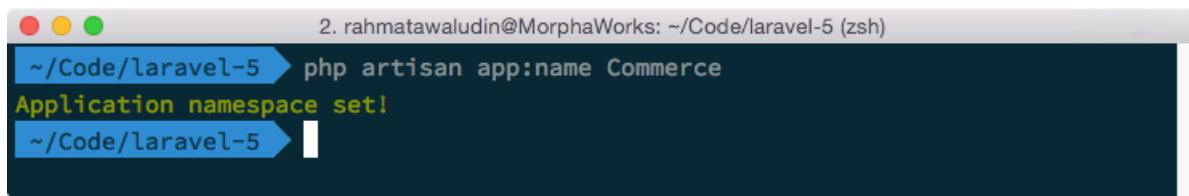
app/Entity/Books.php

```
<?php namespace App\Entity;  
  
class Books {  
....
```

Biar lebih *make sense*, kita juga dapat mengubah namespace ini dengan mudah. Misalnya kita sedang membuat aplikasi e-Commerce dan ingin mengubah namespace menjadi `Commerce`, kita tidak perlu mengubah satu persatu. Cukup jalankan perintah:

Mengubah namespace

```
1 php artisan app:name Commerce
```



```
2. rahmatawaludin@MorphaWorks: ~/Code/laravel-5 (zsh)
~/Code/laravel-5 ➤ php artisan app:name Commerce
Application namespace set!
~/Code/laravel-5 ➤
```

Mengubah namespace

Maka Laravel akan mengubah **semua** namespace App menjadi `Commerce`. Misalnya pada file `app/User.php` akan berubah menjadi :

app/User.php

```
<?php namespace Commerce;

use Illuminate\Auth\Authenticatable;
....
```

Cek juga di `composer.json`:

composer.json

```
13 ....
14 "autoload": {
15     "classmap": [
16         "database"
17     ],
18     "psr-4": {
19         "Commerce\\": "app/"
20     }
21 },
22 ....
```

Pokoknya semua file yang memanggil namespace App akan berubah menjadi memanggil `Commerce`.

Jika ingin mengembalikan kembali ke namespace App, gunakan

Mengembalikan namespace

```
1 php artisan app:name App
```

Keren kan? :)

Folder App

Oke, balik lagi ke struktur folder `app`. Berikut penjelasan singkat beberapa folder yang ada didalamnya berikut fungsinya:

- **Commands** : tempat menyimpan custom Artisan commands yang telah kita buat.
- **Console** : tempat menyimpan semua file console yang digunakan ketika kita berinteraksi dengan aplikasi via Console (misalnya dengan `php artisan tinker`).
- **Events** : tempat menyimpan custom event yang dibuat.
- **Exceptions** : tempat menyimpan custom exceptions yang dibuat.
- **Handlers** : tempat menyimpan handler untuk event dan commands yang kita buat.
- **HTTP** : tempat menyimpan semua file yang kita gunakan ketika berinteraksi dengan aplikasi via HTTP request (misalnya diakses dari browser). Di dalamnya terdapat file controller dan route.
- **Providers** : tempat menyimpan semua Service Providers.
- **Services** : tempat menyimpan custom services yang digunakan oleh aplikasi.

Kedepannya akan kita bahas lebih jauh lagi penggunaan folder-folder tersebut.



Bingung dengan banyaknya folder?

Yang membuat pemula kebingungan ketika mulai mempelajari Laravel salah satunya dia beranggapan harus memahami dan menggunakan semua folder ini. Padahal, **kita tidak wajib menggunakan semua folder ini**. Gunakan folder di Laravel sesuai kebutuhan/kompleksitas aplikasi kita. Terkadang untuk aplikasi sederhana kita paling sering akan menyentuh folder `Http` dan `database`.

Service Providers

Dalam menjalankan sebuah aplikasi, biasanya akan ada fase dimana kita akan menginisialisasi berbagai servis yang akan dibutuhkan. Pada bagian ini, kita akan mempelajari bagaimana menggunakan service provider yang sudah ada dan membuat service provider baru.

Fungsi dari Service Provider?

Fungsi utama dari service provider adalah melakukan bootstrapping aplikasi. Bootstrapping disini maksudnya proses registrasi berbagai komponen yang berada di aplikasi. Beberapa contohnya, melakukan binding ke service container, melakukan binding event listener, routing, dsb.

Konfigurasi Service Provider

Di Laravel, semua service provider yang di load oleh aplikasi ditulis di config/app pada bagian providers.

config/app

```
110 ....
111 'providers' => [
112
113     /*
114      * Laravel Framework Service Providers...
115      */
116     'Illuminate\Foundation\Providers\ArtisanServiceProvider',
117     'Illuminate\Auth\AuthServiceProvider',
118     'Illuminate\Bus\BusServiceProvider',
119     'Illuminate\Cache\CacheServiceProvider',
120     'Illuminate\Foundation\Providers\ConsoleSupportServiceProvider',
121     'Illuminate\Routing\ControllerServiceProvider',
122     'Illuminate\Cookie\CookieServiceProvider',
123     'Illuminate\Database\DatabaseServiceProvider',
124     'Illuminate\Encryption\EncryptionServiceProvider',
125     'Illuminate\Filesystem\FilesystemServiceProvider',
126     'Illuminate\Foundation\Providers\FoundationServiceProvider',
127     'Illuminate\Hashing\HashServiceProvider',
128     'Illuminate\Mail\MailServiceProvider',
129     'Illuminate\Pagination\PaginationServiceProvider',
130     'Illuminate\Pipeline\PipelineServiceProvider',
131     'Illuminate\Queue\QueueServiceProvider',
132     'Illuminate\Redis\RedisServiceProvider',
133     'Illuminate\Auth\Passwords\PasswordResetServiceProvider',
134     'Illuminate\Session\SessionServiceProvider',
135     'Illuminate\Translation\TranslationServiceProvider',
136     'Illuminate\Validation\ValidationServiceProvider',
137     'Illuminate\View\ViewServiceProvider',
138
139     /*
140      * Application Service Providers...
141      */
142     'App\Providers\AppServiceProvider',
143     'App\Providers\BusServiceProvider',
144     'App\Providers\ConfigServiceProvider',
145     'App\Providers\EventServiceProvider',
```

```
146     'App\Providers\RouteServiceProvider',
147
148 ],
149 ....
```

Update: Pada saat buku ini ditulis, fitur `::class` di di PHP 5⁴⁵ sudah digunakan di Laravel. Sehingga, penulisan nama class disini tidak lagi menggunakan string tapi menggunakan `::class`. Misalnya `'App\Providers\AppServiceProvider'` menjadi `App\Providers\AppServiceProvider::class`. Hasilnya akan sama dengan menggunakan string seperti yang kita pelajari sekarang.

Dari file ini, isian provider pada bagian **Laravel Framework Service Providers...** adalah service provider dari framework Laravel yang isinya tidak boleh kita rubah. Semua file untuk service provider ini berada di folder `vendor\laravel\framework\Illuminate`.

Sedangkan provider pada bagian **Application Service Providers...** merupakan service provider default setiap aplikasi dibuat. Service provider pada bagian ini boleh kita modifikasi sesuai kebutuhan.

Pemisahan jenis service provider ini, sangat sesuai dengan salah satu definisi fitur sebuah framework dari Wikipedia⁴⁶:

Non-modifiable Framework Code

The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but should not modify its code.

Mari kita pelajari beberapa service provider default yang bisa kita modifikasi. Jika kita akses folder `app/Providers` akan ditemui beberapa file service provider, berikut penjelasan singkatnya :

- **AppServiceProvider.php** : file ini digunakan untuk melakukan *binding* ke service container.
- **BusServiceProvider.php** : file ini digunakan untuk melakukan *binding* command bus.
- **ConfigServiceProvider.php** : file ini digunakan untuk melakukan *override* terhadap isian dari konfigurasi.
- **EventServiceProvider.php** : file ini digunakan untuk me-register event di aplikasi
- **RouteServiceProvider.php** : file ini digunakan untuk melakukan registrasi file routing.

⁴⁵ <http://php.net/manual/tr/migration55.new-features.php#migration55.new-features.class-name>

⁴⁶ http://en.wikipedia.org/wiki/Software_framework

Penggunaan Service Provider

Mari kita coba pakai service provider yang sudah ada sesuai kebutuhan. Dalam contoh ini, kita akan membuat file baru untuk menyimpan mapping route. Secara default, Laravel akan me-*load* mapping route dari file app/Http/routes.php. Ini tertulis di app/Providers/RouteServiceProvider.php pada method map.

app/Providers/RouteServiceProvider.php

```

35 ...
36 public function map(Router $router)
37 {
38     $router->group(['namespace' => $this->namespace], function($router)
39     {
40         require app_path('Http/routes.php');
41     });
42 }
43 ...

```

kita dapat mengecek route default ini dengan perintah php artisan route:list

| Domain | Method | URI | Name | Action | Middleware |
|--------|--------------------------------|-------------------------------------------------------|------|------------------------------------------------------------|------------|
| | GET HEAD | / | | App\Http\Controllers\WelcomeController@index | guest |
| | GET HEAD | home | | App\Http\Controllers\HomeController@index | auth |
| | GET HEAD | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getRegister | guest |
| | POST | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postRegister | guest |
| | GET HEAD | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getLogin | guest |
| | POST | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postLogin | guest |
| | GET HEAD | auth/logout/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getLogout | guest |
| | GET HEAD POST PUT PATCH DELETE | auth/_missing | | App\Http\Controllers\Auth\AuthController@missingMethod | guest |
| | GET HEAD | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@gmail | guest |
| | POST | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postEmail | guest |
| | GET HEAD | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@getReset | guest |
| | POST | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postReset | guest |
| | GET HEAD POST PUT PATCH DELETE | password/_missing | | App\Http\Controllers\Auth\PasswordController@missingMethod | guest |

Route default

Yang kita lakukan adalah membuat file baru di app/Http/custom-routes.php. Dengan route baru misalnya untuk halaman kontak.

app/Http/custom-routes.php

```
<?php

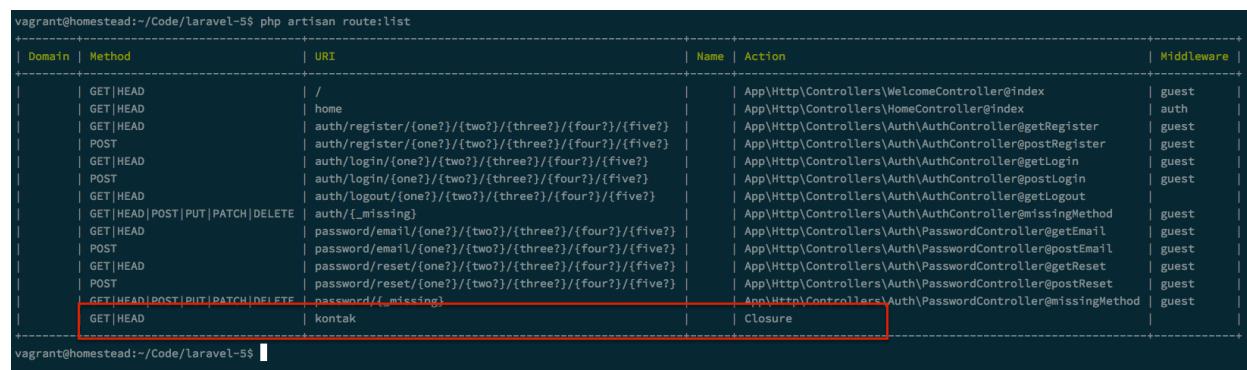
Route::get('/kontak', function() {
    return 'Kontak saya di 087822250272';
});
```

Kemudian, tambahkan alamat file ini di RouteServiceProvider pada method `map`:

app/Providers/RouteServiceProvider.php

```
35 ....
36 public function map(Router $router)
37 {
38     $router->group(['namespace' => $this->namespace], function($router)
39     {
40         require app_path('Http/routes.php');
41         require app_path('Http/custom-routes.php');
42     });
43 }
44 ....
```

Kemudian kita cek kembali route nya:



| Domain | Method | URI | Name | Action | Middleware |
|--------|--------------------------------|-------------------------------------------------------|------|------------------------------------------------------------|------------|
| | GET HEAD | / | | App\Http\Controllers\WelcomeController@index | guest |
| | GET HEAD | home | | App\Http\Controllers\HomeController@index | auth |
| | GET HEAD | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getRegister | guest |
| | POST | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postRegister | guest |
| | GET HEAD | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getLogin | guest |
| | POST | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postLogin | guest |
| | GET HEAD | auth/logout/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getLogout | |
| | GET HEAD POST PUT PATCH DELETE | auth/_missing | | App\Http\Controllers\Auth\AuthController@missingMethod | guest |
| | GET HEAD | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@getEmail | guest |
| | POST | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postEmail | guest |
| | GET HEAD | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@getReset | guest |
| | POST | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postReset | guest |
| | GET HEAD POST PUT PATCH DELETE | password/_missing | | App\Http\Controllers\Auth\PasswordController@missingMethod | guest |
| | GET HEAD | kontak | | Closure | |

Route dari file baru berhasil di load

Terlihat disini, kita berhasil me-load route dari file yang telah kita buat. Penggunaan RouteServiceProvider provider ini sangat bermanfaat untuk merapihkan struktur routing yang kita buat. Misalnya, dalam membangun API, kita dapat menggunakan file route yang berbeda untuk tiap versinya.



Route baru tidak muncul?

Terkadang mapping ke file route baru tidak berefek ke Laravel, untuk itu jalankan perintah `php artisan clear-compiled` untuk me-refresh class yang sudah di compile.

Membuat Service Provider

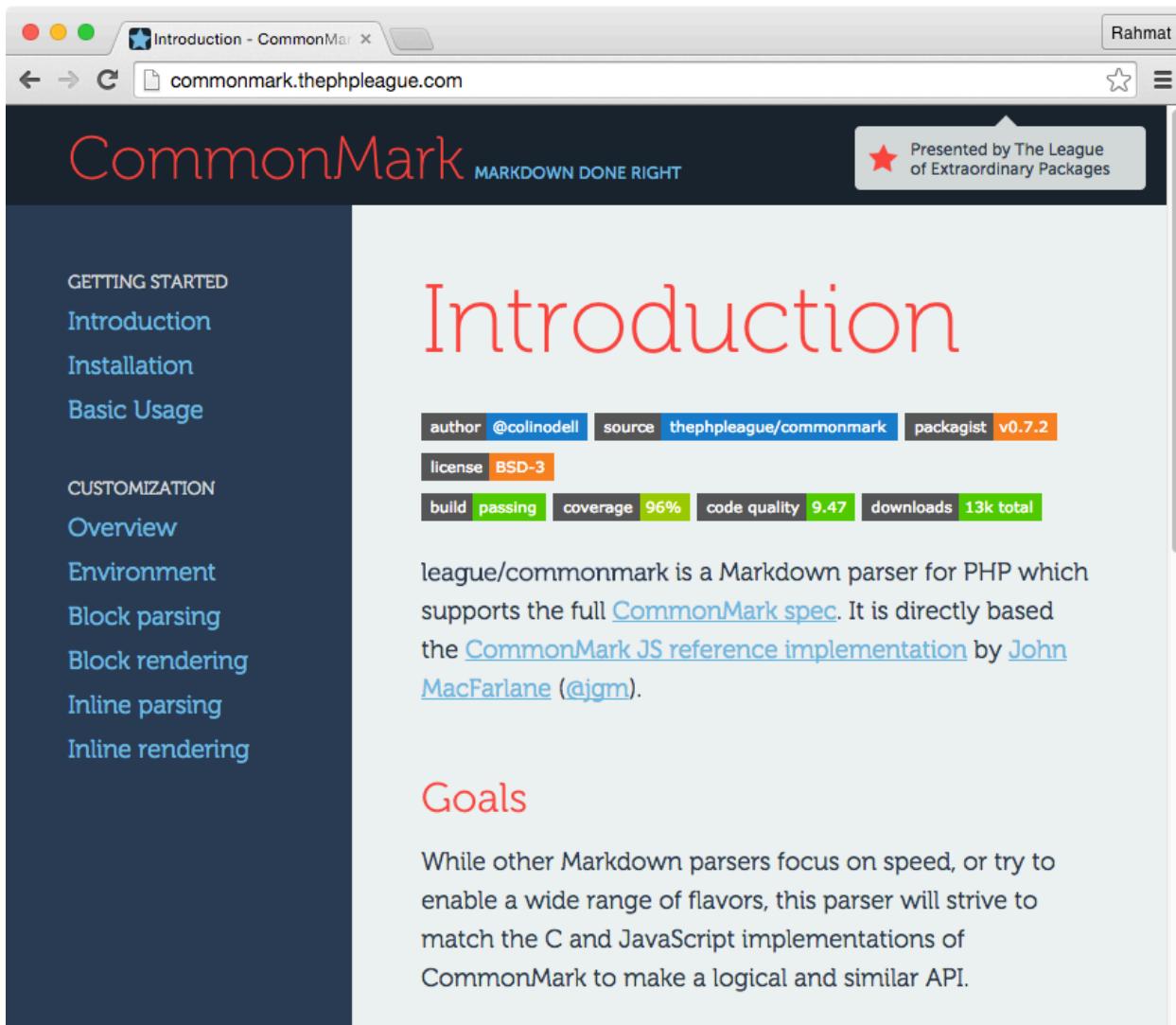
Kita juga dapat membuat service provider sendiri jika dibutuhkan. Misalnya, terkadang kita membutuhkan markdown parser untuk aplikasi kita. Sebenarnya telah banyak package Laravel untuk parsing markdown. Tapi, pada contoh ini, kita akan menggunakan library [CommonMark dari The PHP League⁴⁷](#) dan melakukan binding ke Service Container via service provider. Tujuan akhirnya, kita tidak perlu melakukan inisialisasi CommonMark setiap kali kita membutuhkannya.



Service Container?

Pembahasan Service Container akan kita bahas pada bagian selanjutnya. Untuk sekarang, kita cukup memahami Service Container sebagai tempat kita dapat menyimpan instance dari sebuah object. Ketika sudah tersimpan, object tersebut dapat kita panggil kapanpun dan dimanapun dari aplikasi.

⁴⁷ <http://commonmark.thephpleague.com>



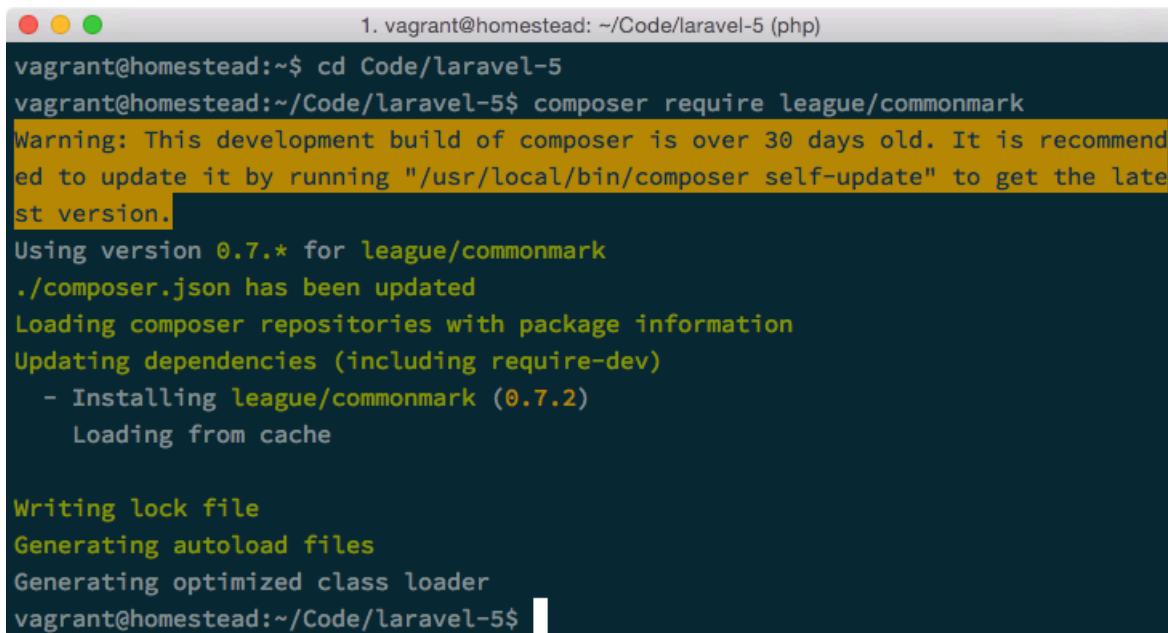
CommonMark dari The PHP League

Pertama, *require* dulu league/commonmark di composer dengan perintah `composer require league/commonmark`

Perintah diatas akan menambah isian `require` pada file `composer.json` dan mendownload package tersebut ke folder `vendor\league\commonmark`.

composer.json

```
6 ....  
7 "require": {  
8     "laravel/framework": "5.0.*",  
9     "league/commonmark": "0.7.*"  
10 },  
11 ....
```



1. vagrant@homestead: ~/Code/laravel-5 (php)

```
vagrant@homestead:~$ cd Code/laravel-5  
vagrant@homestead:~/Code/laravel-5$ composer require league/commonmark  
Warning: This development build of composer is over 30 days old. It is recommended to update it by running "/usr/local/bin/composer self-update" to get the latest version.  
Using version 0.7.* for league/commonmark  
.composer.json has been updated  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
- Installing league/commonmark (0.7.2)  
    Loading from cache  
  
Writing lock file  
Generating autoload files  
Generating optimized class loader  
vagrant@homestead:~/Code/laravel-5$
```

Menambahkan league/commonmark via composer

Berikut ini contoh sederhana penggunaan CommonMark untuk melakukan parsing markdown:



```
vagrant@homestead:~/Code/laravel-5$ cd Code/laravel-5
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> use League\CommonMark\CommonMarkConverter;
=> false
>>> $markdown = new CommonMarkConverter();
=> <League\CommonMark\CommonMarkConverter #000000008473d7500000000850287a> {}
>>> $markdown->convertToHtml('# Laravel keren');
=> "\n<h1>Laravel keren</h1>\n"
>>> $markdown->convertToHtml('## Markdown top');
=> "\n<h2>Markdown top</h2>\n"
>>> $markdown->convertToHtml('**Tulisan tebal**');
=> "\n<p><strong>Tulisan tebal</strong></p>\n"
>>> $markdown->convertToHtml('[Website Laravel](http://laravel.com)');
=> "\n<p><a href=\"http://laravel.com\">Website Laravel</a></p>\n"
>>> 
```

Contoh parsing markdown

Terlihat disini, kita perlu menginisialisasi CommonMark untuk menggunakannya. Di aplikasi yang cukup besar, cukup merepotkan jika harus melakukan inisialisasi setiap kali kita membutuhkannya. Mari kita buat service provider untuk CommonMark ini dengan nama `MarkdownServiceProvider`.

Jalankan `php artisan make:provider MarkdownServiceProvider`.



```
vagrant@homestead:~/Code/laravel-5$ php artisan make:provider MarkdownServiceProvider
Provider created successfully.
vagrant@homestead:~/Code/laravel-5$ 
```

Meng-generate MarkdownServiceProvider

Akan muncul file baru `app/Providers/MarkdownServiceProvider.php` dengan isi:

app/Providers/MarkdownServiceProvider.php

```
<?php namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class MarkdownServiceProvider extends ServiceProvider {
```

```
    /**
     * Bootstrap the application services.
```

```
*  
* @return void  
*/  
public function boot()  
{  
    //  
}  
  
/**  
 * Register the application services.  
 *  
 * @return void  
 */  
public function register()  
{  
    //  
}  
}
```

Yang akan kita lakukan adalah membuat instance dari CommonMark dan menyimpannya di Service Container. Caranya kita membuat object dari CommonMark pada method `boot` dan melakukan binding ke container pada method `register`. Sehingga code `MarkdownServiceProvider.php` berubah menjadi:

app/Providers/MarkdownServiceProvider.php

```
<?php namespace App\Providers;  
  
use Illuminate\Support\ServiceProvider;  
use League\CommonMark\CommonMarkConverter;  
  
class MarkdownServiceProvider extends ServiceProvider {  
  
    protected $markdown;  
  
    /**  
     * Bootstrap the application services.  
     *  
     * @return void  
     */  
    public function boot()  
    {
```

```
        $this->markdown = new CommonMarkConverter;
    }

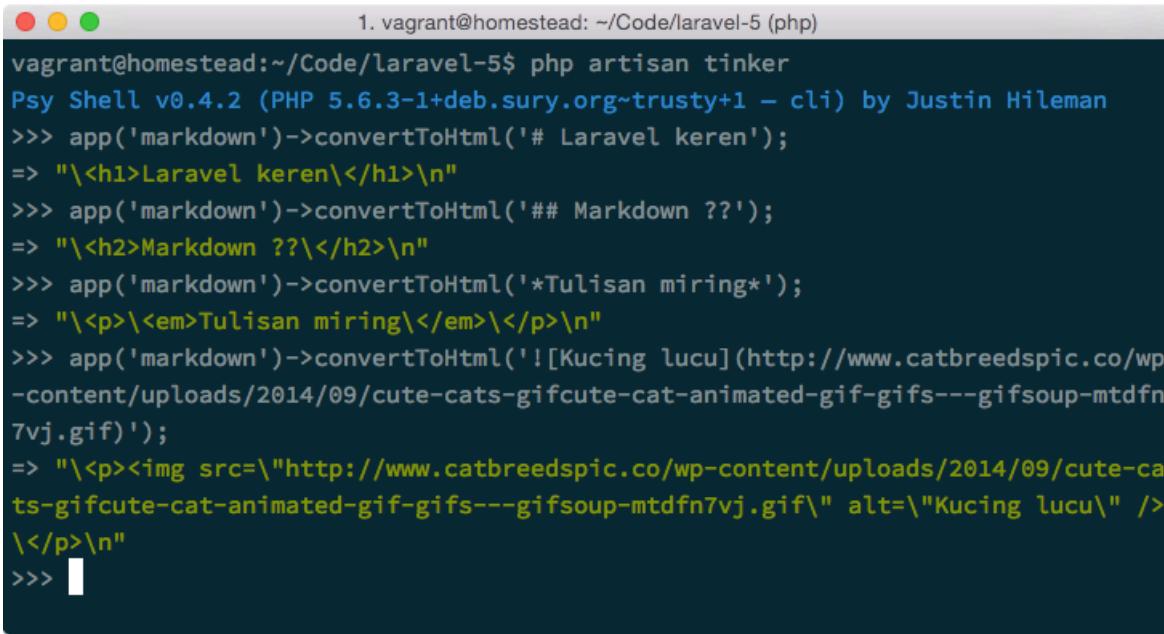
    /**
     * Register the application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton('markdown', function() {
            return $this->markdown;
        });
    }
}
```

Selanjutnya, kita harus menambahkan provider ini di `config/app.php`

config/app.php

```
138 ....
139 /*
140  * Application Service Providers...
141 */
142 'App\Providers\AppServiceProvider',
143 'App\Providers\BusServiceProvider',
144 'App\Providers\ConfigServiceProvider',
145 'App\Providers\EventServiceProvider',
146 'App\Providers\RouteServiceProvider',
147 'App\Providers\MarkdownServiceProvider',
148 ....
```

Kini, untuk mendapatkan instance dari `CommonMark` kita cukup menggunakan `app(['markdown'])`, `app()->make('markdown')` atau `app('markdown')`. Misalnya seperti berikut:

A screenshot of a terminal window titled "1. vagrant@homestead: ~/Code/laravel-5 (php)". The window displays the Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman. The user is testing the MarkdownServiceProvider. The output shows various markdown strings being converted to HTML. For example, "# Laravel keren" is converted to "<h1>Laravel keren</h1>\n", and "## Markdown ???" is converted to "<h2>Markdown ???\n". Other examples include converting "*Tulisan miring*" to "<p>Tulisan miring</p>\n" and a complex URL in a code block to "<p>\n". The session ends with a final '>>> |'.

```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('markdown')->convertToHtml('# Laravel keren');
=> "\u003ch1\u003eLaravel keren\u003c/h1\u003e\n"
>>> app('markdown')->convertToHtml('## Markdown ???');
=> "\u003ch2\u003eMarkdown ???\u003c/h2\u003e\n"
>>> app('markdown')->convertToHtml('*Tulisan miring*');
=> "\u003cp\u003e\u003ctem\u003eTulisan miring\u003c/em\u003e\u003c/p\u003e\n"
>>> app('markdown')->convertToHtml('![Kucing lucu](http://www.catbreedspic.co/wp-content/uploads/2014/09/cute-cats-gifcute-cat-animated-gif-gifs---gifsoup-mtdfn7vj.gif)');
=> "\u003cp\u003cimg src='http://www.catbreedspic.co/wp-content/uploads/2014/09/cute-cats-gifcute-cat-animated-gif-gifs---gifsoup-mtdfn7vj.gif'\u003e alt='Kucing lucu' /\u003c/p\u003e\n"
>>> |
```

Test MarkdownServiceProvider

Sip.

Dari contoh ini, kita dapat melihat manfaat Service Providers untuk menginisialisasi CommonMark. Contoh ini memang terlalu sederhana, karena kita hanya menginisialisasi satu object. Tapi, dalam aplikasi yang besar, terkadang untuk menginisialisasi sebuah object diperlukan beberapa object tambahan yang saling bergantung. Penggunaan Service Provider yang tepat akan membuat proses inisialisasi ini lebih singkat dan rapi.

Service Container

Jika kita ditanya apa salah satu keunggulan Laravel, jawab saja Service Container. Ya, fitur ini merupakan salah satu fitur yang menarik di Laravel. Fungsi utamanya melakukan automatic resolutions untuk class dependency. Apa itu? yuk kita pelajari..

Apa itu class dependency?

Class dependency dalam pemrograman adalah kondisi dimana kita membutuhkan class lain untuk membuat object dari sebuah class. Misalnya perhatikan class Dompet dan Customer berikut:

app/Dompet.php

```
<?php namespace App;

class Dompet
{
    private $saldo;

    public function __construct($saldo = 100)
    {
        $this->saldo = $saldo
    }

    public function cekIsi()
    {
        return $this->saldo;
    }
}
```

app/Customer.php

```
<?php namespace App;

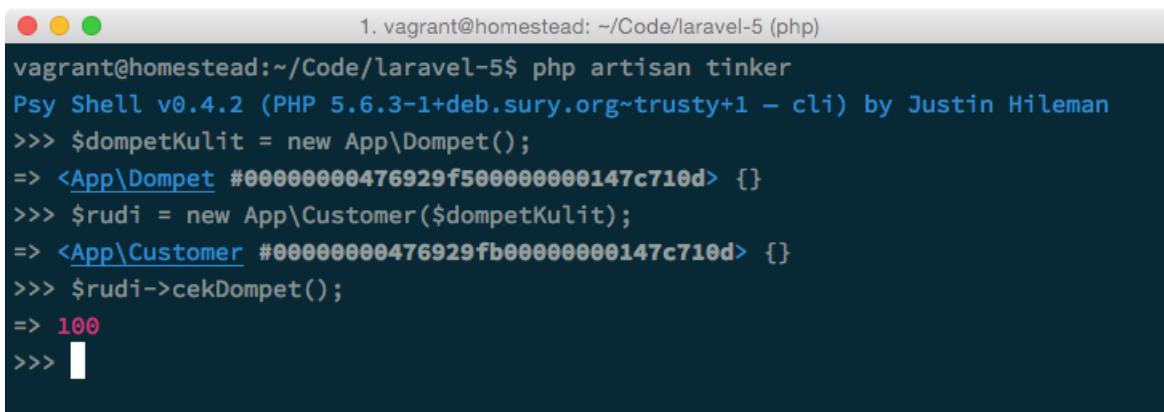
class Customer
{
    private $dompet;

    public function __construct(Dompet $dompet)
    {
        $this->dompet = $dompet;
    }

    public function cekDompet()
    {
        return $this->dompet->cekIsi();
    }
}
```

Terlihat disini, untuk membuat instance dari class `Customer`, kita membutuhkan instance dari class `Dompet`. Untuk membuat objec dari class `Dompet`, kita juga perlu memberikan nilai isi dompetnya, jika tidak diisi maka akan default ke 100.

Tanpa memanfaatkan Service Container, syntax untuk membuat object dari class `Customer` akan seperti berikut:



```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $dompetKulit = new App\Dompet();
=> <App\Dompet #00000000476929f500000000147c710d> []
>>> $rudi = new App\Customer($dompetKulit);
=> <App\Customer #00000000476929fb00000000147c710d> []
>>> $rudi->cekDompet();
=> 100
>>> 
```

Tanpa Service Container

Terlihat disini, untuk membuat object dari class `Customer` kita harus menginisialisasi class yang menjadi dependency-nya (class `Dompet`).

Masalah yang muncul dari dependency ini diantaranya

- Bagaimana jika class `Customer` tidak hanya bergantung pada class `Dompet`? Misalnya, `Customer` juga bergantung pada class `KTP`, `KartuMember`, `Kontak`, dan lainnya?
- Bagaimana jika class `Dompet` juga memiliki dependency class yang lain?
- Bagaimana jika terdapat banyak sekali dependency yang saling bergantung satu sama lain?
- Seberapa banyak syntax yang harus kita tulis untuk membuat object dari class `Customer`?

Tentunya, syntax untuk membuat object dari class `Customer` ini akan sangat panjang.

Memahami Automatic Resolution

Service Container hadir untuk menyelesaikan masalah ini. Fitur automatic resolution, memungkinkan kita untuk memiliki object dari class `Customer`, tanpa harus menginisialisasi dependency-nya. Misalnya seperti ini:

```
1. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $rudi = app('App\Customer');
=> <App\Customer #000000003ac3f1e00000000064917a9d> {}
>>> $rudi->cekDompet();
=> 100
>>> 
```

Dengan Service Container

Wow!

Terlihat disini, kita tidak pernah menginisialisasi dependency dari class `Customer`. Tapi, ketika kita memanggil `app('App\Customer')`, Laravel secara ajaib mengetahui apa saja dependency dari class `Customer` dan membuatkannya untuk kita. Bayangkan jika ada ratusan dependency yang saling bergantung, fitur Service Container ini akan sangat membantu efisiensi koding.

Semua keajaiban fitur ini terjadi berkat penggunaan Reflection API. Untuk memahaminya, silahkan buka kembali pembahasan Reflection API di bab pertama.

Memahami Binding & Resolve

Sesuai namanya, Service Container berfungsi sebagai sebuah tempat dimana kita bisa menyimpan berbagai hal dari string biasa, closure, object, class, dll. Ketika kita menyimpan "sesuatu" ke dalam Service Container dinamakan **binding**. Sementara ketika kita mengambil "sesuatu" dari Service Container dinamakan **resolve**.

Proses binding ini seperti mengisi array asosiatif, jadi kita harus menyediakan **key** dan **value** nya. Ketika melakukan resolve, sama seperti array asosiatif, kita menggunakan key nya untuk mengambil value yang telah disimpan.

Untuk melakukan binding, kita bisa menggunakan `app()->bind('key', 'value');`. Contohnya, jika kita ingin menyimpan string ke dalam Service Container kita dapat menggunakan closure:

```
2. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()->bind('apa itu laravel?', function() { return 'Laravel adalah framework Laravel paling keren dan elegan.'; });
=> null
>>> 
```

Simple binding

Untuk mengambil nilai yang telah disimpan di Service Container, kita dapat menggunakan `app('key')`:

```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()->bind('apa itu laravel?', function() { return 'Laravel adalah framework Laravel paling keren dan elegan.'; });
=> null
>>> app('apa itu laravel?');
=> "Laravel adalah framework Laravel paling keren dan elegan."
>>>
```

Simple resolve

Biasanya proses binding dilakukan di Service Provider pada method `register`. Di dalam Service Provider, untuk mendapatkan instance dari Service Container dapat menggunakan `$this->app`. Misalnya, jika kita hendak melakukan binding diatas pada `MarkdownServiceProvider` yang telah kita buat sebelumnya, syntaxnya akan seperti ini:

app/Providers/MarkdownServiceProvider.php

```
24 ....
25 public function register()
26 {
27     $this->app->singleton('markdown', function() {
28         return $this->markdown;
29     });
30
31     $this->app->bind('apa itu laravel?', function() {
32         return 'Laravel adalah framework Laravel paling keren dan elegan.';
33     });
34 }
35 ....
```

Ketika kita mencoba melakukan resolve, akan tetap berhasil.

```
2. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('apa itu laravel?');
=> "Laravel adalah framework Laravel paling keren dan elegan."
>>>
```

Resolve binding dari Service Provider

Oke, saya rasa ini cukup untuk memberikan gambaran proses binding dan resolve pada Service Container. Selanjutnya, kita akan mempelajari jenis binding lainnya. Tentunya, yang lebih menarik dan menantang.. :)



Resolve dari Service Container

Selain menggunakan `app('key')`, kita juga dapat menggunakan `app()['key']` dan `app()->make('key')` untuk melakukan resolve nilai/object dari Service Container

Memahami Binding Interface

Selain dependency berupa class, dalam coding terkadang kita juga menggunakan interface sebagai dependency. Menggunakan interface akan membuat code yang kita bangun lebih flexibel. Pada contoh class `Customer` sebelumnya, kita dapat membuat customer memiliki berbagai jenis metode pembayaran dengan membuat interface `PaymentMethod`.

app/PaymentMethod.php

```
<?php namespace App;

interface PaymentMethod
{
    public function cekIsi();
}
```

Kemudian kita inject dependency ini ke class `Customer`.

app/Customer.php

```
<?php namespace App;

class Customer
{
    private $payment;

    public function __construct(PaymentMethod $payment)
    {
        $this->payment = $payment;
    }

    public function setPayment(PaymentMethod $payment)
    {
        $this->payment = $payment;
    }

    public function cekPayment()
    {
        return $this->payment->cekIsi();
    }
}
```

Misalnya kita buat implementasi dari PaymentMethod dengan KartuKredit dan Rekening-Ponsel.

app/KartuKredit.php

```
<?php namespace App;

class KartuKredit implements PaymentMethod
{
    private $saldo;
    private $cardNumber;

    public function __construct($saldo = 100, $cardNumber = null)
    {
        $this->saldo = $saldo;
        $this->cardNumber = $cardNumber;
    }

    public function cekIsi()
```

```
{  
    return $this->saldo . ' saldo tersisa.';  
}  
}
```

app/RekeningPonsel.php

```
<?php namespace App;  
  
class RekeningPonsel implements PaymentMethod  
{  
    private $pulsa;  
    private $phoneNo;  
  
    public function __construct($pulsa = 100, $phoneNo = null)  
    {  
        $this->pulsa = $pulsa;  
        $this->phoneNo = $phoneNo;  
    }  
  
    public function cekIsi()  
    {  
        return $this->pulsa . ' pulsa tersisa.';  
    }  
}
```

Jika menggunakan cara manual, berikut contoh syntaxnya:

```
1. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $cc = new App\KartuKredit(10000, '2342523452352');
=> <App\KartuKredit #0000000016efbbd60000000055f063f2> {}
>>> $rudi = new App\Customer($cc);
=> <App\Customer #0000000016efbbc30000000055f063f2> {}
>>> $rudi->cekPayment();
=> "10000 saldo tersisa."
>>> $ponsel = new App\RekeningPonsel(2000, '0878345354');
=> <App\RekeningPonsel #0000000016efbb2c0000000055f063f2> {}
>>> $rudi->setPayment($ponsel);
=> null
>>> $rudi->cekPayment();
=> "2000 pulsa tersisa."
>>> 
```

Membuat object dari interface manual

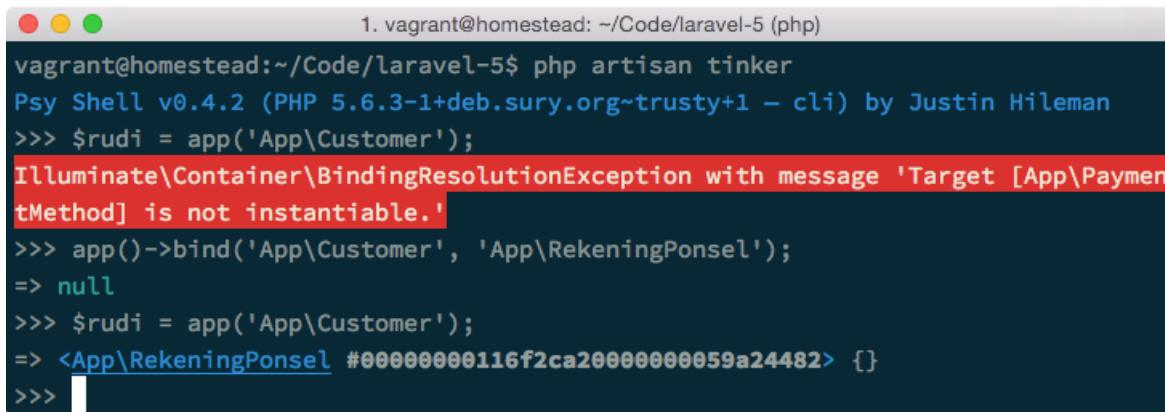
Mari kita coba buat dengan Service Container:

```
1. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $rudi = app('App\Customer');
Illuminate\Container\BindingResolutionException with message 'Target [App\Paymen
tMethod] is not instantiable.'
>>> 
```

Error karena dependency berupa interface

Ooooppss.. Ternyata ada error. Error ini terjadi karena dependency dari class `Customer` adalah sebuah interface. Karena sebuah interface tidak bisa menjadi object, maka kita harus memberitahu Service Container class apa yang akan digunakan ketika kita membutuhkan implementasi dari `PaymentMethod`.

Caranya dengan melakukan binding interface, misalnya kita bind `PaymentMethod` ke `RekeningPonsel` dengan cara berikut:



```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $rudi = app('App\Customer');
Illuminate\Container\BindingResolutionException with message 'Target [App\PaymentMethod] is not instantiable.'
>>> app()->bind('App\Customer', 'App\RekeningPonsel');
=> null
>>> $rudi = app('App\Customer');
=> <App\RekeningPonsel #00000000116f2ca20000000059a24482> {}
>>>
```

Melakukan binding di terminal

Pada syntax ini, kita memberitahu Laravel, jika dia butuh implementasi dari PaymentMethod maka gunakan RekeningPonsel.

Dalam project sungguhan, binding ini biasanya diletakkan di method register di Service Provider. Misalnya kita bind PaymentMethod ke KartuKredit di AppServiceProvider:

app/Providers/AppServiceProvider.php

```
25 ....
26 public function register()
27 {
28     $this->app->bind(
29         'Illuminate\Contracts\Auth\Registrar',
30         'App\Services\Registrar'
31     );
32
33     $this->app->bind(
34         'App\PaymentMethod',
35         'App\KartuKredit'
36     );
37 }
38 ....
```

Jika kita mencoba kembali membuat object Customer, maka akan berhasil.

```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $rudi = app('App\Customer');
=> <App\Customer #0000000027b75994000000004cabaeef2> []
>>> $rudi->cekPayment();
=> "100 saldo tersisa."
>>> 
```

Binding dari service provider

Beberapa manfaat lain dari binding interface adalah membuat code kita mudah di test. Pada saat menjalankan test, menggunakan service provider kita dapat merubah binding interface ke class lain *on-the-fly*.

Memahami Binding Instance

Binding instance ini akan sangat bermanfaat jika kita ingin mengeset nilai default untuk dependency dari class yang dibuat.

Contoh penggunaanya, misalnya untuk implementasi [Factory Pattern⁴⁸](#). Dimana ketika kita hendak membuat instance suatu object, dibutuhkan cukup banyak parameter. Dengan binding instance, proses pembuatan instance ini (berikut semua parameternya) akan diserahkan kepada Service Container.

Pada contoh sebelumnya, kita melakukan binding untuk interface `PaymentMethod` dengan class `KartuKredit`. Pada binding tersebut kita menggunakan nilai default dari untuk menginisialisasi class `KartuKredit`.

```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $cc = app('App\KartuKredit');
=> <App\KartuKredit #000000007464105a0000000035a4457b> []
>>> $cc->cekIsi();
=> "100 saldo tersisa."
>>> 
```

Menggunakan nilai default untuk KartuKredit

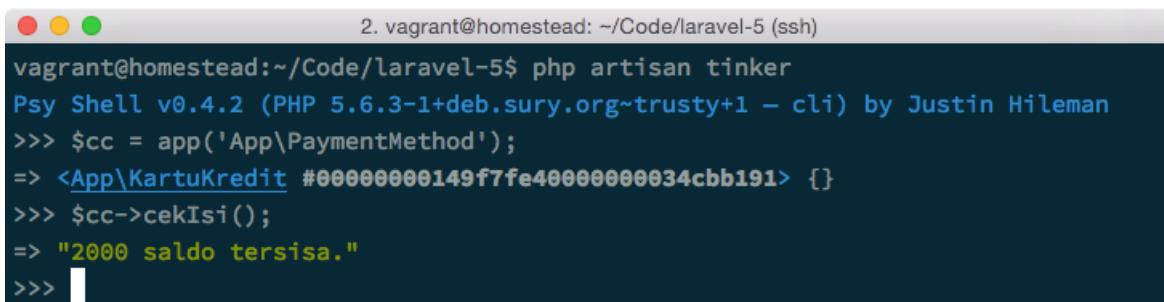
Jika diinginkan, kita juga bisa melakukan binding ke object-instance dari `KartuKredit` menggunakan closure. Dengan cara ini, kita dapat memberikan nilai untuk `KartuKredit` diluar nilai default. Misalnya, dengan mengubah `AppServiceProvider` menjadi:

⁴⁸http://en.wikipedia.org/wiki/Factory_method_pattern

app/Providers/AppServiceProvider.php

```
25 ....  
26 public function register()  
27 {  
28     $this->app->bind(  
29         'Illuminate\Contracts\Auth\Registrar',  
30         'App\Services\Registrar'  
31     );  
32  
33     $cc = new \App\KartuKredit(2000, '42542353545');  
34     $this->app->instance('App\PaymentMethod', $cc);  
35 }  
36 ....
```

Disini kita menginisialisasi object dari class `KartuKredit` secara manual dengan saldo default 2000. Kita bisa mengeceknya dengan:



The screenshot shows a terminal window titled "2. vagrant@homestead: ~/Code/laravel-5 (ssh)". It displays a Psy Shell session. The user runs the command `php artisan tinker`. Inside the session, they create a new instance of `\App\KartuKredit` with a balance of 2000 and store it in the variable `$cc`. Then, they call the `cekIsi()` method on `$cc`, which returns the string "2000 saldo tersisa."

Melakukan binding instance

Memahami Contextual Binding

Terkadang kita ingin agar Service Container menggunakan implementasi yang berbeda untuk interface ketika kita memanggilnya. Ini dinamakan Contextual Binding.

Dalam contoh sebelumnya, misalnya untuk implementasi interface misalnya jika yang dibutuhkan adalah class `Customer` maka kita akan memberikan `RekeningPonsel`. Sedangkan, jika yang dibutuhkan adalah class `Seller` maka kita akan memberikan `KartuKredit`.

Mari kita buat class `Seller` terlebih dahulu.

App/Seller.php

```
<?php namespace App;

class Seller
{
    private $payment;

    public function __construct(PaymentMethod $payment)
    {
        $this->payment = $payment;
    }

    public function setPayment(PaymentMethod $payment)
    {
        $this->payment = $payment;
    }

    public function cekPayment()
    {
        return $this->payment->cekIsi();
    }
}
```

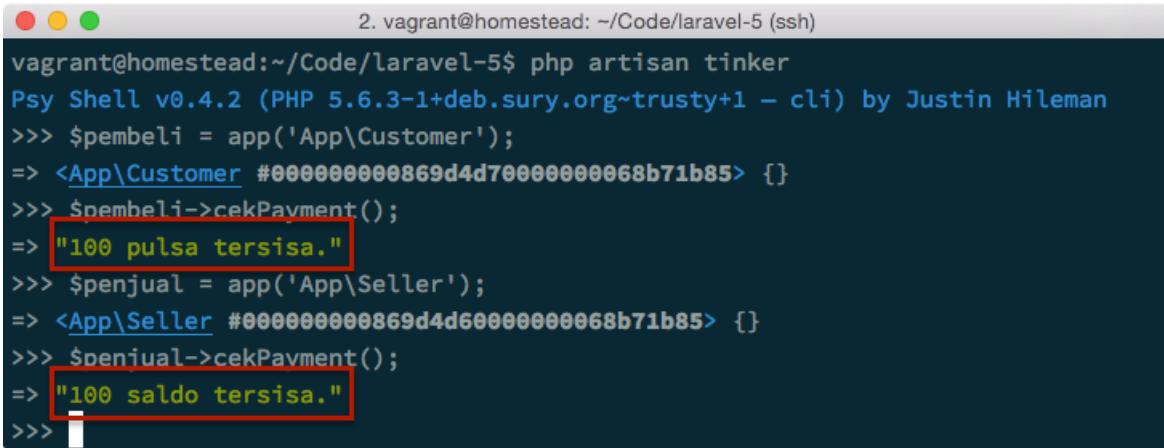
Selanjutnya, kita tambahkan contextual binding di AppServiceProvider dengan mengubah method register menjadi:

app/Providers/AppServiceProvider.php

```
25 ....
26 public function register()
27 {
28     $this->app->bind(
29         'Illuminate\Contracts\Auth\Registrar',
30         'App\Services\Registrar'
31     );
32
33     $this->app->when('App\Customer')
34             ->needs('App\PaymentMethod')
35             ->give('App\RekeningPonsel');
36
37     $this->app->when('App\Seller')
38             ->needs('App\PaymentMethod')
```

```
39         ->give('App\\KartuKredit');
40     }
41     ....
```

Kemudian kita test dengan syntax berikut:



```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $pembeli = app('App\\Customer');
=> <App\Customer #000000000869d4d70000000068b71b85> {}
>>> $pembeli->cekPayment();
=> "100 pulsa tersisa."
>>> $penjual = app('App\\Seller');
=> <App\Seller #000000000869d4d60000000068b71b85> {}
>>> $penjual->cekPavment();
=> "100 saldo tersisa."
>>>
```

Contextual Binding berhasil

Terlihat disini, walaupun class `Customer` dan `Seller` sama-sama membutuhkan implementasi dari `PaymentMethod`, namun implementasi yang diberikan untuk keduanya berbeda. Tergantung proses binding yang kita lakukan pada `AppServiceProvider`.

Memahami Singleton Binding

Dalam teknik Design Pattern, ada yang namanya Singleton Pattern. Saya tidak akan menjelaskan secara detail tentang design pattern ini dan manfaatnya (nanti saja di buku tentang Design Pattern :)). Singkatnya, pattern ini memungkinkan satu class hanya boleh membuat satu instance.

Dalam Laravel, pattern ini sangat mudah diimplementasikan dengan menggunakan Service Container. Sebelum kita mempelajari tekniknya, mari kita buat contoh tanpa singleton. Untuk mengeceknya, mari kita tambahkan `echo` ketika membuat `KartuKredit` baru.

app/KartuKredit

```
7 ....  
8 public function __construct($saldo = 100, $cardNumber = null)  
9 {  
10     $this->saldo = $saldo;  
11     $this->cardNumber = $cardNumber;  
12     echo 'membuat kartu kredit...';  
13 }  
14 ....
```

Bayangkan jika terdapat beberapa penjual yang masih satu keluarga, misalnya Papa Budi, Mama Budi dan Budi. Mereka menggunakan kartu kredit yang sama. Tanpa singleton, ini yang akan kita dapatkan:



The screenshot shows a terminal window titled "2. vagrant@homestead: ~/Code/laravel-5 (ssh)". It displays the following command and its output:

```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker  
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman  
>>> $papaBudi = app('App\Seller');  
membuat kartu kredit...  
=> <App\Seller #00000006e702bdb00000000743690b6> {}  
>>> $mamaBudi = app('App\Seller');  
membuat kartu kredit...  
=> <App\Seller #00000006e702bda00000000743690b6> {}  
>>> $budi = app('App\Seller');  
membuat kartu kredit...  
=> <App\Seller #00000006e702b2400000000743690b6> {}  
>>> 
```

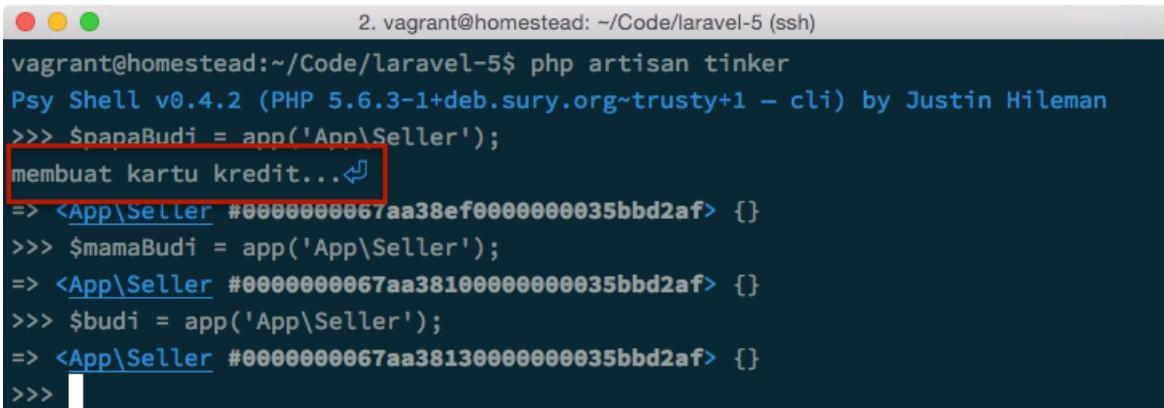
Tanpa Singleton, kartu kredit dibuat beberapa kali

Terlihat disini, setiap kali kita membuat Seller maka KartuKredit baru akan dibuat. Kita dapat menggunakan Singleton agar kartu kredit ini hanya dibuat sekali. Caranya tambahkan syntax berikut pada method register di AppServiceProvider:

app/Providers/AppServiceProvider.php

```
25 ....  
26 public function register()  
27 {  
28     ....  
29     $this->app->singleton('App\KartuKredit', function($app)  
30     {  
31         return new \App\KartuKredit;  
32     });  
33 }  
34 ....
```

Jika kita mencobanya kembali, kini KartuKredit hanya akan dibuat sekali untuk semua instance dari Seller.



```
vagrant@homestead:~/Code/laravel-5 (ssh)  
vagrant@homestead:~/Code/laravel-5$ php artisan tinker  
Psy Shell v0.4.2 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman  
>>> $papaBudi = app('App\Seller');  
membuat kartu kredit... ↵  
=> <App\Seller #0000000067aa38ef0000000035bbd2af> []  
>>> $mamaBudi = app('App\Seller');  
=> <App\Seller #0000000067aa38100000000035bbd2af> []  
>>> $budi = app('App\Seller');  
=> <App\Seller #0000000067aa38130000000035bbd2af> []  
>>> █
```

Menggunakan Singleton, kartu kredit hanya dibuat sekali

Sebenarnya masih banyak contoh penggunaan Service Container. Yang kita bahas disini adalah fitur dasar dari Service Container. Harapannya dengan memahami konsep dasar ini, kita lebih mudah dalam memahami fitur-fitur lainnya dari Laravel. Sip.

Facades

Sebagaimana yang kita pelajari pada bagian sebelumnya, untuk melakukan resolve dari Service Container kita dapat menggunakan `app('key')`. Meskipun beginilah cara me-resolve binding dari Service Container, tapi cara ini tidak ideal untuk semua kondisi dan programmer. Terutama untuk programmer yang baru pertama kali menggunakan Laravel dan baru belajar PHP.

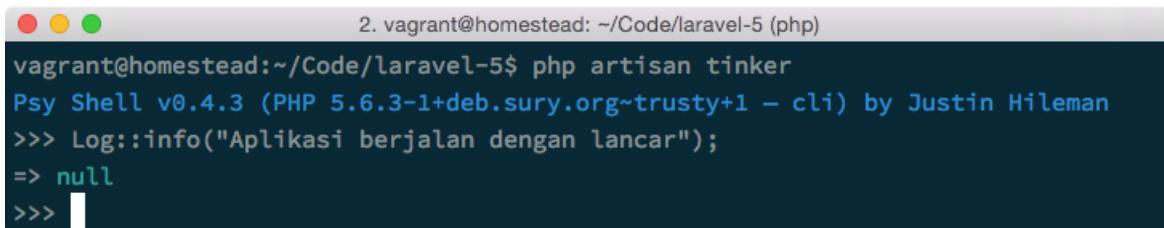
Facade & Service Container

Kebanyakan dari programmer yang baru menggunakan Laravel tidak terlalu paham dengan Service Container. Jika Laravel langsung “memaksakan” penggunaan teknik resolve biasa untuk mengambil binding dari Service Container, akan banyak programmer yang kebingungan.

Untuk menjawab masalah itu, Laravel menggunakan teknik Facade. Dengan teknik ini, untuk melakukan resolve dari Service Container kita dapat memanggilnya dengan lebih elegan.

Penggunaan Facade

Contohnya, di Laravel terdapat Facade untuk logging dengan nama Log. Fitur ini memungkinkan kita menyimpan log dengan mudah. Contoh syntax sederhana untuk menyimpan Log info adalah `Log::info('isi pesan')`. Setelah syntax ini dijalankan, secara default sebuah baris baru akan ditambahkan di `storage/logs/laravel-tahun-bulan-tanggal.log`. Jalankan perintah berikut dari tinker:



```
2. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> Log::info("Aplikasi berjalan dengan lancar");
=> null
>>> 
```

Membuat log info dengan facade

Untuk mengeceknya, buka file `storage/logs/laravel-tahun-bulan-tanggal.log`. Kurang lebih isinya akan seperti berikut:

storage/logs/laravel-xxxx-xx-xx.log

1 [xxxx-xx-xx xx:xx:xx] local.INFO: Aplikasi berjalan dengan lancar

Hmmmm.. apa yang terjadi disini?

Kalau dilihat sekilas, kita terlihat seperti memanggil method statis pada class Log. Mari kita cek lokasi file class Log dengan Reflection API:



```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> $reflector = new ReflectionClass('Log');
=> <ReflectionClass #000000045ba75ba00000005f03c587> {
    name: "Illuminate\\Support\\Facades\\Log"
}
>>> $reflector->getFileName();
=> "/home/vagrant/Code/laravel-5/vendor/laravel/framework/src/Illuminate/Support/Facades/Log.php"
>>>
```

Mengecek alamat class Log

Terlihat disini, class nya berada di vendor/laravel/framework/src/Illuminate/Support/- Facades/Log.php. Berikut ini konten dari file tersebut:

vendor/laravel/framework/src/Illuminate/Support/Facades/Log.php

```
1 <?php namespace Illuminate\Support\Facades;
2
3 /**
4 * @see \Illuminate\Log\Writer
5 */
6 class Log extends Facade {
7
8     /**
9      * Get the registered name of the component.
10     *
11     * @return string
12     */
13     protected static function getFacadeAccessor() { return 'log'; }
14 }
```

Wah.. mana method statis yang kita panggil??

Terlihat disini, tidak ada method statis yang kita panggil. Sebenarnya, yang terjadi di belakang layar adalah kita melakukan resolve dengan key log dari Service Container.

Hmm.. ke class apakah binding ini di resolve? Kita dapat cek di vendor/laravel/framework/src/Illuminate/Foundation/Application.php pada method registerCoreContainerAliases():

vendor/laravel/framework/src/Illuminate/Foundation/Application.php

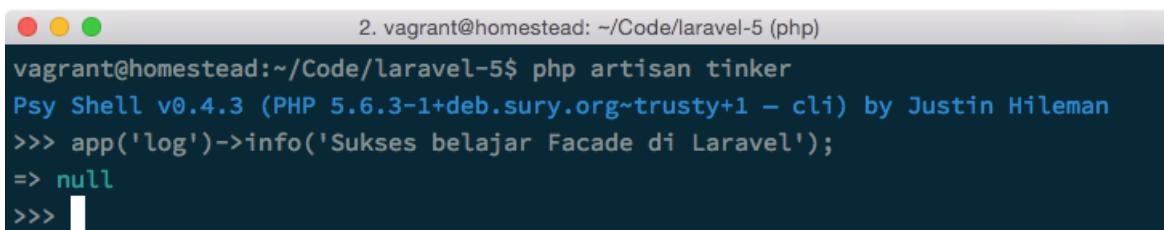
```
.....
public function registerCoreContainerAliases()
{
    $aliases = array(
    .....
    'log'          => ['Illuminate\Log\Writer', 'Illuminate\Contracts\Lo\
gging\Log', 'Psr\Log\LoggerInterface'],
    .....
}
```

Terlihat disini kita melakuan binding key `log` ke class `Illuminate\Log\Writer`. Mari kita cek method `info` pada class ini:

vendor/laravel/framework/src/Illuminate/Log/Writer.php

```
145 .....
146 /**
147 * Log an informational message to the logs.
148 *
149 * @param string $message
150 * @param array $context
151 * @return void
152 */
153 public function info($message, array $context = array())
154 {
155     return $this->writeLog(__FUNCTION__, $message, $context);
156 }
157 .....
```

Terlihat disini, method `info` bukanlah sebuah method statis. Nah, sebenarnya method inilah yang kita panggil ketikan memanggil `Log::info()` atau jika kita ingin melakukan resolve manual syntaxnya akan menjadi `app('log')->info()`:



```
2. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('log')->info('Sukses belajar Facade di Laravel');
=> null
>>> 
```

Resolve facade manual

Jika kita cek kembali file log, maka akan terdapat baris baru:

storage/logs/laravel-xxxx-xx-xx.log

```
1 [xxxx-xx-xx xx:xx:xx] local.INFO: Sukses belajar Facade di Laravel
```

Nah, begitulah cara kerja Facade. Jadi, sebenarnya Facade ini bukanlah sebuah class dengan method statis. Tapi, sebuah shortcut untuk melakukan resolve dari Service Container. Bagi programmer yang baru menggunakan Laravel, adanya Facade ini akan sangat membantu mereka coding di Laravel. Bagi sebagian orang, menggunakan Facade juga membuat syntaxnya lebih bersih dan elegan.

Banyak sekali Facade di Laravel berikut class asli dan bindingnya. Supaya kita tidak perlu mengecek satu-satu semua binding Facade dan class aslinya, silahkan cek referensi di [dokumentasi resmi](#)⁴⁹ atau gunakan referensi berikut:

| Facade | Class | IoC Binding |
|--------------------|----------------------------------------------|----------------|
| App | Illuminate\Foundation\Application | app |
| Artisan | Illuminate\Console\Application | artisan |
| Auth | Illuminate\Auth\AuthManager | auth |
| Auth (Instance) | Illuminate\Auth\Guard | |
| Blade | Illuminate\View\Compilers\BladeCompiler | blade.compiler |
| Bus | Illuminate\Contracts\Bus\Dispatcher | |
| Cache | Illuminate\Cache\Repository | cache |
| Config | Illuminate\Config\Repository | config |
| Cookie | Illuminate\Cookie\CookieJar | cookie |
| Crypt | Illuminate\Encryption\Encrypter | encrypter |
| DB | Illuminate\Database\DatabaseManager | db |
| DB (Instance) | Illuminate\Database\Connection | |
| Event | Illuminate\Events\Dispatcher | events |
| File | Illuminate\Filesystem\Filesystem | files |
| Hash | Illuminate\Contracts\Hashing\Hasher | hash |
| Input | Illuminate\Http\Request | request |
| Lang | Illuminate\Translation\Translator | translator |
| Log | Illuminate\Log\Writer | log |
| Mail | Illuminate\Mail\Mailer | mailer |
| Password | Illuminate\Auth\Passwords\PasswordBroker | auth.password |
| Queue | Illuminate\Queue\QueueManager | queue |
| Queue (Instance) | Illuminate\Queue\QueueInterface | |
| Queue (Base Class) | Illuminate\Queue\Queue | |
| Redirect | Illuminate\Routing\Redirector | redirect |
| Redis | Illuminate\Redis\Database | redis |
| Request | Illuminate\Http\Request | request |
| Response | Illuminate\Contracts\Routing\ResponseFactory | |
| Route | Illuminate\Routing\Router | router |

⁴⁹ <http://laravel.com/docs/5.0/facades#facade-class-reference>

| Facade | Class | IoC Binding |
|----------------------|-----------------------------------------|-------------|
| Schema | Illuminate\Database\Schema\Blueprint | |
| Session | Illuminate\Session\SessionManager | session |
| Session (Instance) | Illuminate\Session\Store | |
| Storage | Illuminate\Contracts\Filesystem\Factory | filesystem |
| URL | Illuminate\Routing\UrlGenerator | url |
| Validator | Illuminate\Validation\Factory | validator |
| Validator (Instance) | Illuminate\Validation\Validator | |
| View | Illuminate\View\Factory | |
| View (Instance) | Illuminate\View\View | view |

Membuat Facade

Terkadang kita juga butuh membuat Facade untuk binding yang sudah kita lakukan. Contohnya, pada contoh Service Provider sebelumnya kita membuat binding `markdown` ke instance dari class `CommonMarkConverter`. Mari kita buat facade Markdown untuk binding tersebut.

Ada 3 langkah untuk membuat Facade:

1. Membuat binding ke Service Container.
2. Membuat Facade class yang meng-extends class `Illuminate\Support\Facades\Facade`.
3. Membuat alias di `config/app`.

Langkah pertama sudah kita lakukan pada penjelasan tentang Service Provider dimana kita melakukan binding dengan key `markdown`.

Langkah kedua, kita dapat membuat file dimanapun, selama masih bisa di load oleh composer. Contohnya, mari kita buat folder `app\Facades` dan membuat file `MarkdownFacade.php` isi dengan konten berikut:

app/Facades/MarkdownFacade.php

```

1 <?php namespace App\Facades;
2 use Illuminate\Support\Facades\Facade;
3
4 class MarkdownFacade extends Facade {
5     protected static function getFacadeAccessor() { return 'markdown'; }
6 }
```

Terlihat disini, kita cukup meng-extends class `Illuminate\Support\Facades\Facade` dan melakukan overide pada method `getFacadeAccessor()`. Pada fungsi tersebut kita kembalikan key dari binding ke Service Container yang telah kita lakukan (`markdown`). Tidak lupa,

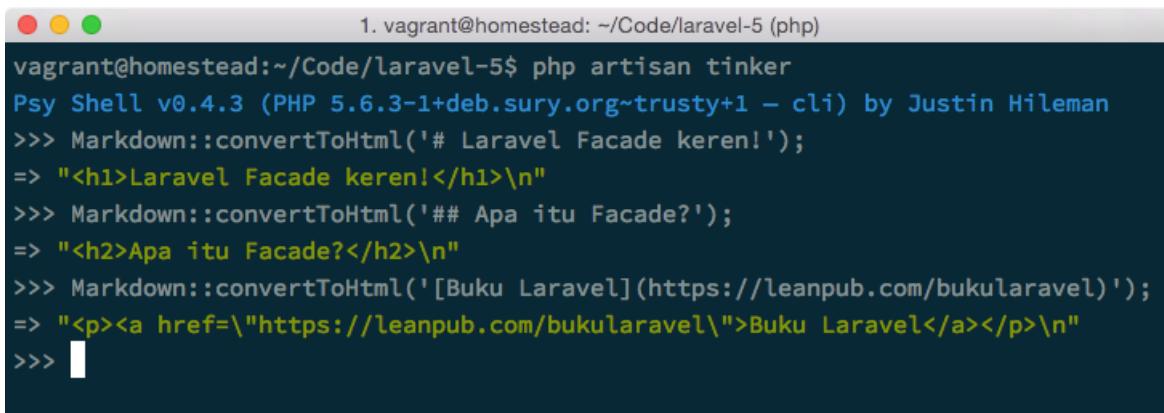
diawal baris kita juga meambahkan namespace App\Facades sesuai dengan alamat foldernya. Tujuannya agar composer dapat melakukan autoload class ini.

Tahap ketiga, tambahkan alias di config/app.php pada array aliases. Untuk ini, kita bebas memberikan nama misalnya MD Markdown, dll. Untuk memudahkan mari kita namakan Markdown:

config/app.php

```
1 ....  
2 'aliases' => [  
3     ....  
4     'Markdown' => 'App\Facades\MarkdownFacade',  
5 ]  
6 ....
```

Mari kita coba



```
vagrant@homestead:~/Code/laravel-5$ php artisan tinker  
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman  
>>> Markdown::convertToHtml('# Laravel Facade keren!');  
=> "<h1>Laravel Facade keren!</h1>\n"  
>>> Markdown::convertToHtml('## Apa itu Facade?');  
=> "<h2>Apa itu Facade?</h2>\n"  
>>> Markdown::convertToHtml('[Buku Laravel](https://leanpub.com/bukularavel)');  
=> "<p><a href=\"https://leanpub.com/bukularavel\">Buku Laravel</a></p>\n"  
>>> 
```

Mencoba Facade Markdown



Autoload tidak berhasil?

Yang harus dicek adalah penempatan file folder dan namespace class. Terkadang kita juga harus me-refresh autoload composer. Untuk melakukannya jalankan composer dump-autoload dari folder project.

Contracts

Salah satu sisi arsitektural yang fundamental dalam penggunaan Laravel adalah contract. Di bagian ini kita akan mempelajari kenapa Laravel menggunakan contract dan bagaimana mengaplikasikannya dalam aplikasi yang kita bangun.

Apa itu Contract?

Bayangkan ketika sebuah perusahaan misalnya PT Madu Sehat membutuhkan seorang web developer untuk membuat websitenya. Sebut saja pak Budi, bagian HRD, punya kenalan pak Ruslan yang seorang web developer. Bisakah pak Budi langsung mengontak pak Ruslan? Tentu bisa. Tapi kurang etis. Meskipun pak Ruslan dapat membuat web, sangat mungkin diluaran sana banyak web developer yang dapat membuat web misalnya mas Rudi, kang Dadang dan pembaca buku ini.. :)

Yang akan dilakukan pak Budi adalah membuat lowongan kerja. Di lowongan tersebut misalnya tertulis:

Lowongan Web Developer (WD)

PT Madu Sehat membutuhkan seorang web developer dengan kriteria:

- Mampu membuat website eCommerce.
- Mampu membangun integrasi dengan pembayaran online.
- Mampu membuat website bersahabat dengan perangkat mobile.

Dengan cara ini, PT Madu Sehat siapapun yang memiliki kemampuan yang tertulis dapat mengisi lowongan web developer ini. Kini, Web Developer tidak terikat pada pak Ruslan.

Nah, lowongan kerja ini dalam pemrograman dinamakan **Contract**. Fungsinya mendefinisikan fitur apa saja yang dibutuhkan. Biasanya contract ini dibuat dalam bentuk interface dengan beberapa method yang harus di override oleh class yang meng-implement-nya. Dengan menggunakan contract, aplikasi kita menjadi lebih flexibel. Dia juga tidak terlalu terikat ke concrete class lain yang menjadi dependency-nya untuk implementasi fitur-fiturnya.

Kenapa Kita Menggunakan Contract?

Mari kita buat sebuah kasus dalam coding. Misalnya, untuk membuat mailing list dalam aplikasi kita bisa menggunakan [MailChimp⁵⁰](https://mailchimp.com) dengan library [disini⁵¹](https://bitbucket.org/mailchimp/mailchimp-api-php) atau [Campaign Monitor⁵²](https://www.campaignmonitor.com) dengan library [disini⁵³](https://github.com/campaignmonitor/createsend-php). Mari kita pelajari cara untuk subscribe untuk keduanya.

⁵⁰<https://mailchimp.com>

⁵¹<https://bitbucket.org/mailchimp/mailchimp-api-php>

⁵²<https://www.campaignmonitor.com>

⁵³<https://github.com/campaignmonitor/createsend-php>

Meskipun proses subscribe yang dilakukan sama, tapi method yang digunakan kedua library ini berbeda. Di MailChimp, untuk melakukan subscribe kita harus menggunakan method `subscribe()` (bisa dicek [disini⁵⁴](#)) sementara di Campaign Monitor kita harus menggunakan method `add()` (bisa dicek [disini⁵⁵](#)).

Saya akan buat contoh sederhana untuk kedua library itu. Misalnya, untuk MailChimp, class nya akan terlihat seperti ini:

App/MailChimp.php

```
1 <?php namespace App;
2 class MailChimp {
3
4     public function subscribe($email)
5     {
6         return 'subscribing ' . $email . ' to mailchimp server.';
7     }
8 }
```

Sedangkan untuk Campaign Monitor, classnya akan terlihat seperti ini:

App/CampaignMonitor.php

```
1 <?php namespace App;
2 class CampaignMonitor {
3
4     public function add($email)
5     {
6         return 'subscribing ' . $email . ' to campaign monitor server.';
7     }
8 }
```

Terlihat disini, untuk melakukan registrasi email ke tiap mail server tersebut, method yang digunakan berbeda.

Ceritanya, kita punya class NewsLetter yang berfungsi untuk mendaftarkan alamat email ke server yang kita inginkan. Di versi pertama, kita baru kenal dengan MailChimp, maka kita buat code seperti ini:

⁵⁴<https://apidocs.mailchimp.com/api/2.0/lists/subscribe.php>

⁵⁵https://github.com/campaignmonitor/createsend-php/blob/master/csrest_subscribers.php

App/NewsLetterV1.php

```
1 <?php namespace App;
2 class NewsLetterV1 {
3     protected $mailchimp;
4
5     public function __construct(MailChimp $mailchimp)
6     {
7         $this->mailchimp = $mailchimp;
8     }
9     public function register($email) {
10        return $this->mailchimp->subscribe($email);
11    }
12 }
```

Terlihat disini, kita meng-*inject* mailchimp ke class ini. Istilahnya **Dependency Injection**. Pada method `register` kita memanggil method `subscribe` dari MailChimp untuk mendaftarkan email yang kita terima.

Mari kita coba syntax ini. Untuk memudahkan saya akan menggunakan *automatic resolution* dari Service Container agar kita tidak perlu membuat dependency dari class NewsLetter secara manual.

NewsLetterV1 menggunakan MailChimp

Sip. Untuk saat ini aplikasi telah berjalan dengan normal.

Semakin lama aplikasi terus berkembang. Setelah riset oleh atasan, ternyata kita juga harus bisa mensupport Campaign Monitor. Sebagaimana kita ketahui, untuk registrasi ke server Campaign Monitor method yang digunakan berbeda, yaitu `add()`. Hmm.. karena belum paham dengan contract, bisa jadi kita akan merubah class NewsLetter menjadi seperti berikut:

App/NewsLetterV2.php

```
1 <?php namespace App;
2 class NewsLetterV2 {
3     protected $campaignMonitor;
4
5     public function __construct(CampaignMonitor $campaignMonitor)
6     {
7         $this->campaignMonitor = $campaignMonitor;
8     }
9     public function register($email) {
10        return $this->campaignMonitor->add($email);
11    }
12 }
```

Terlihat disini, cukup banyak perubahan yang kita lakukan. Perubahan yang paling besar adalah kita merubah inject class yang kita butuhkan menjadi `CampaignMonitor` di method `__construct()`. Kita juga melakukan perubahan di method `register()` dengan mengakses method `add()` dari class `CampaignMonitor`. Memang fitur ini akan berjalan, jika kita tes:



```
1. vagrant@homestead: ~/Code/laravel-5 (ssh)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('App\NewsLetterV2')->register('rahmat.awaludin@gmail.com');
=> "subscribing rahmat.awaludin@gmail.com to campaign monitor server."
>>> 
```

NewsLetterV2 menggunakan Campaign Monitor

Tapi, apa beberapa masalah dari solusi ini:

- Bagaimana jika kita ingin merubah server mail yang digunakan on-the-fly? Misalnya, ketika server Campaign Monitor overload, kita ingin pindah ke server MailChimp atau sebaliknya?
- Bagaimana jika ada server mail baru misalnya KirimSurat yang harus kita support yang menggunakan method ? Apa kita harus merubah kembali class NewsLetter?

Nah, untuk menjawab masalah diatas kita akan menggunakan contract. Mari kita beri nama `MailListContract`. Implementasi contract ini adalah berupa *interface*. Kita akan membuatnya di `app/Contracts/MailListContract.php`. Silahkan buat folder yang dibutuhkan. Berikut isi contract-nya:

app/Contracts/MailListContract.php

```
1 <?php namespace App\Contracts;
2
3 interface MailListContract {
4
5     /**
6      * Mendaftarkan email ke server mailing list
7      * @param String $email
8      */
9     public function register($email);
10 }
```

Pada contract ini, kita abstract method `register()` yang harus di *override* oleh class yang mengimplementasikan `MailListContract` ini.

Nah, kini kita perlu merubah `NewsLetter` agar menggunakan contract ini:

app/NewsLetterV3.php

```
1 <?php namespace App;
2
3 use App\Contracts\MailListContract;
4 class NewsLetterV3 {
5     protected $mailList;
6
7     public function __construct(MailListContract $mailList)
8     {
9         $this->mailList = $mailList;
10    }
11    public function register($email) {
12        return $this->mailList->register($email);
13    }
14 }
```

Terlihat disini, kita menggunakan contract yang telah dibuat. Karena contract ini interface, kita perlu mengimplementasikan class ini ke class `MailChimp` dan `CampaignMonitor` agar `NewsLetter` bisa berjalan.

Ada beberapa cara yang bisa kita lakukan:

- Merubah class `MailChimp/CampaignMonitor` monitor agar meng-*implements* interface ini dan meng-*override* method `register`

- Menggunakan *class wrapper* yang akan menjadi adapter untuk method untuk registrasi di class MailChimp/CampaignMonitor.

Karena ceritanya class MailChimp dan CampaignMonitor datang dari library luar, kita tidak mungkin merubahnya. Solusi yang bisa kita pakai adalah membuat *class wrapper*. Berikut wrapper untuk MailChimp:

app/MailChimpList.php

```
1 <?php namespace App;
2
3 use App\Contracts\MailListContract;
4 class MailChimpList implements MailListContract {
5     protected $mailchimp;
6
7     public function __construct(MailChimp $mailchimp)
8     {
9         $this->mailchimp = $mailchimp;
10    }
11
12    public function register($email)
13    {
14        return $this->mailchimp->subscribe($email);
15    }
16 }
```

Terlihat disini, kita memanggil method `subscribe()` dari class MailChimp pada method `register()` di class ini.

Sedangkan wrapper untuk CampaignMonitor seperti berikut:

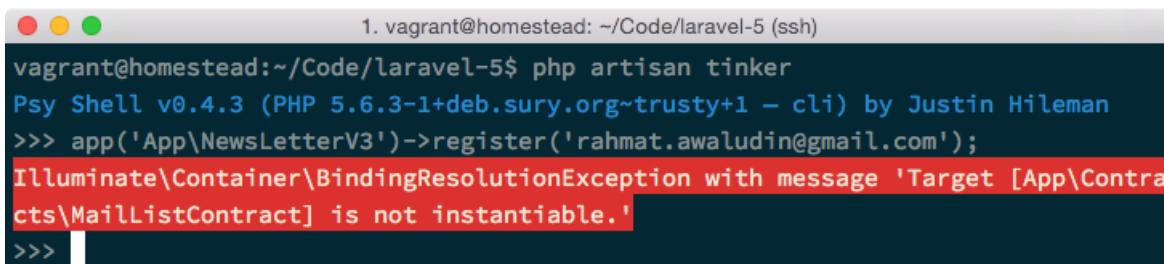
app/CampaignMonitorList.php

```
1 <?php namespace App;
2
3 use App\Contracts\MailListContract;
4 class CampaignMonitorList implements MailListContract {
5     protected $campaignMonitor;
6
7     public function __construct(CampaignMonitor $campaignMonitor)
8     {
9         $this->campaignMonitor = $campaignMonitor;
10    }
11 }
```

```
11
12     public function register($email)
13     {
14         return $this->campaignMonitor->add($email);
15     }
16 }
```

Terlihat disini, kita memanggil method `add()` dari class `CampaignMonitor` pada method `register()` di class ini.

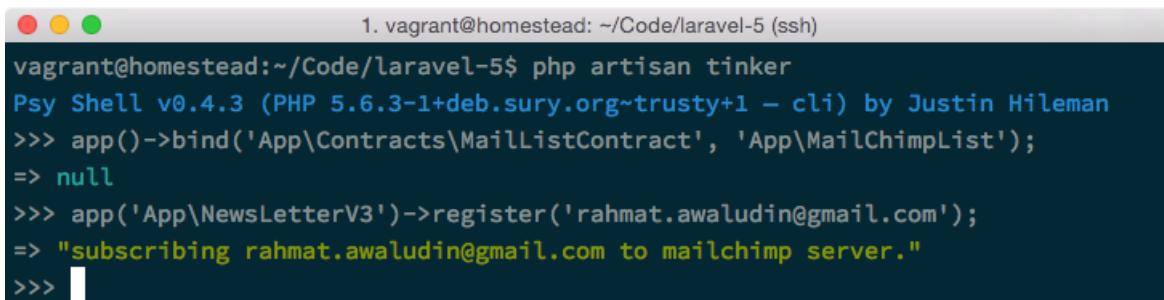
Mari kita coba class `NewsLetter` yang baru ini.



```
1. vagrant@homestead: ~/Code/laravel-5 (ssh)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('App\NewsLetterV3')->register('rahmat.awaludin@gmail.com');
Illuminate\Container\BindingResolutionException with message 'Target [App\Contra
cts\MailListContract] is not instantiable.'
>>>
```

Error.. MailListContract belum dibind ke class

Ooooppss.. terlihat kita mendapat error disini. Ini terjadi, karena `MailListContract` berupa interface dan Service Container tidak menemukan binding ke concrete class. Solusinya, kita harus melakukan binding untuk `MailListContract`. Kita dapat melakukannya di Service Provider atau on-the-fly langsung. Untuk memudahkan, mari kita binding on-the-fly misalnya kita binding ke `MailChimpList`:



```
1. vagrant@homestead: ~/Code/laravel-5 (ssh)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()->bind('App\Contracts\MailListContract', 'App\MailChimpList');
=> null
>>> app('App\NewsLetterV3')->register('rahmat.awaludin@gmail.com');
=> "subscribing rahmat.awaludin@gmail.com to mailchimp server."
>>>
```

Berhasil mengirim newsletter dengan MailChimp

Sip, Berhasil. Jika kita ingin merubah implementasi ke Campaign Monitor, kita cukup merubah binding ke `CampaignMonitorList`.

```
1. vagrant@homestead: ~/Code/laravel-5 (ssh)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()>bind('App\Contracts\MailListContract', 'App\CampaignMonitorList');
=> null
>>> app('App\NewsLetterV3')>register('rahmat.awaludin@gmail.com');
=> "subscribing rahmat.awaludin@gmail.com to campaign monitor server."
>>> 
```

Berhasil mengirim newsletter dengan Campaign Monitor

Tuh kan.. :)

Terlihat disini, kita tidak perlu melakukan perubahan pada class NewsLetter untuk menggunakan server mail yang berbeda. Cukup ubah binding saja. Disinilah kelebihan dari contract.

Ketika kedepannya ada server mail baru buatan kita sendiri, misalnya KirimSurat, kita cukup mengimplementasikan contract MailListContract pada class tersebut. Misalnya seperti berikut:

app/KirimSurat.php

```
1 <?php namespace App;
2
3 use App\Contracts\MailListContract;
4 class KirimSurat implements MailListContract {
5
6     public function register($email)
7     {
8         return 'Mendaftarkan email ' . $email . ' ke server Kirim Surat.';
9     }
10 }
```

Jika kita mau menggunakan class ini, cukup merubah binding:

```
1. vagrant@homestead: ~/Code/laravel-5 (ssh)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()>bind('App\Contracts\MailListContract', 'App\KirimSurat');
=> null
>>> app('App\NewsLetterV3')>register('rahmat.awaludin@gmail.com');
=> "Mendaftarkan email rahmat.awaludin@gmail.com ke server Kirim Surat."
>>> 
```

Berhasil mengirim newsletter dengan Kirim Surat

Keren kan? :)

Dokumentasi

Kelebihan lain dari contract adalah contract dapat berfungsi sebagai dokumentasi. Jika kita melihat ke interface MailListContract, dapat kita temui method apa saja yang ada berikut definisinya dari tiap method tersebut.

Loose Coupling

Contract mengikuti salah satu paradigma dalam membangun aplikasi yang solid:

Program to an interface, not an implementation.

(Gang of Four 1995:18)

Dengan menggunakan contract, aplikasi kita tidak akan terikat pada implementasi logic dari satu library. Ia akan terikat pada interface (contract) yang telah kita definisikan. Sehingga, implementasinya bisa beragam dan flexibel. Kalau dalam pemrograman, arsitektur seperti ini dinamakan **Loose Coupling**. Sip.

Penggunaan Contract di Laravel

Laravel datang dengan banyak sekali contract. Semua contract yang ada di Laravel bisa dilihat di [repository resmi⁵⁶](#) di github.

⁵⁶ [vendor/laravel/framework/src/Illuminate/Foundation/Application.php](https://github.com/laravel/framework/blob/5.8/src/Illuminate/Foundation/Application.php)

[READ ONLY] Subtree split of the Illuminate Contracts component (see laravel/framework)

133 commits 2 branches 1 release 19 contributors

branch: master contracts / +

Merge branch '5.0'

taylorotwell authored 10 days ago latest commit 66c7ad3c21

| Component | Commit Message | Time Ago |
|------------|-------------------------------------------------------------------------------|--------------|
| Auth | Phpdoc fixes | 17 days ago |
| Bus | Fixing a few things. | 25 days ago |
| Cache | Adding 'remember' method to contract 'Illuminate\Contracts\Cache\Repository'. | 2 months ago |
| Config | continuing work on config and assets. | 4 months ago |
| Console | Add ability to easily queue an Artisan command. | 4 months ago |
| Container | Add type hinting on \$parameters | 13 days ago |
| Cookie | Implement contracts for many major components of the framework. | 7 months ago |
| Database | Working on various queue / bus / command things. | 3 months ago |
| Debug | CS fixes | 3 months ago |
| Encryption | Check FQN | 3 months ago |
| Events | Dispatcher::fire() accepts either string or object as first arg | 2 months ago |

Code Pull requests 0

Pulse

Graphs

SSH clone URL
git@github.com:111ur

You can clone with HTTPS, SSH, or Subversion. ↗

Clone in Desktop Download ZIP

Laravel Contract di Github

Semua contract itu secara default sudah di binding ke implementasi komponen yang disediakan oleh Laravel, dapat kita cek di [repository Laravel](#)⁵⁷.

⁵⁷ <https://github.com/laravel/framework/blob/5.0/src/Illuminate/Foundation/Application.php#L891>

```

886     /**
887      * Register the core class aliases in the container.
888      *
889      * @return void
890      */
891     public function registerCoreContainerAliases()
892     {
893         $aliases = array(
894             'app'                => ['Illuminate\Foundation\Application', 'Illuminate\Contracts\Container\Container'],
895             'artisan'            => ['Illuminate\Console\Application', 'Illuminate\Contracts\Console\Application'],
896             'auth'               => 'Illuminate\Auth\AuthManager',
897             'auth.driver'        => ['Illuminate\Auth\Guard', 'Illuminate\Contracts\Auth\Guard'],
898             'auth.password.tokens' => 'Illuminate\Auth\Passwords\TokenRepositoryInterface',
899             'blade.compiler'     => 'Illuminate\View\Compilers\BladeCompiler',
900             'cache'              => ['Illuminate\Cache\CacheManager', 'Illuminate\Contracts\Cache\Factory'],
901             'cache.store'        => ['Illuminate\Cache\Repository', 'Illuminate\Contracts\Cache\Repository'],
902             'config'              => ['Illuminate\Config\Repository', 'Illuminate\Contracts\Config\Repository'],
903             'cookie'              => ['Illuminate\Cookie\CookieJar', 'Illuminate\Contracts\Cookie\Factory', 'Illuminate\Cookie\Queue'],
904             'crypter'             => ['Illuminate\Encryption\Encrypter', 'Illuminate\Contracts\Encryption\Encrypter'],
905             'db'                 => 'Illuminate\Database\DatabaseManager',
906             'events'             => ['Illuminate\Events\Dispatcher', 'Illuminate\Contracts\Events\Dispatcher'],
907             'files'              => 'Illuminate\Filesystem\Filesystem',
908             'filesystem'          => 'Illuminate\Contracts\Filesystem\Factory',
909             'filesystem.disk'    => 'Illuminate\Contracts\Filesystem\Filesystem',
910             'filesystem.cloud'   => 'Illuminate\Contracts\Filesystem\Cloud',
911             'hash'                => 'Illuminate\Contracts\Hashing\Hasher',
912             'translator'          => ['Illuminate\Translation\Translator', 'Symfony\Component\Translation\Translator'],
913             'log'                 => ['Illuminate\Log\Writer', 'Illuminate\Contracts\Logging\Log', 'Psr\Log\LoggerInterface'],
914             'mailer'              => ['Illuminate\Mail\Mailer', 'Illuminate\Contracts\Mail\Mailer', 'Illuminate\Contract'],
915             'paginator'           => 'Illuminate\Pagination\Factory',
916             'auth.password'       => ['Illuminate\Auth\Passwords>PasswordBroker', 'Illuminate\Contracts\Auth\Password'],
917             'queue'               => ['Illuminate\Queue\QueueManager', 'Illuminate\Contracts\Queue\Factory', 'Illuminate\Queue\Sync'],
918             'queue.connection'   => 'Illuminate\Contracts\Queue\Queue',
919             'redirect'            => 'Illuminate\Routing\Redirector',
920             'redis'               => ['Illuminate\Redis\Database', 'Illuminate\Contracts\Redis\Database'],
921             'request'             => 'Illuminate\Http\Request',
922             'router'              => ['Illuminate\Routing\Router', 'Illuminate\Contracts\Routing\Registrar'],
923             'session'             => 'Illuminate\Session\SessionManager',
924             'session.store'       => ['Illuminate\Session\Store', 'Symfony\Component\HttpFoundation\Session\SessionInterface']

```

Binding Contract Default

Terlihat disini, misalnya untuk contract `Illuminate\Contracts\Logging\Log` di binding ke `Illuminate\Log\Writer`, yang di binding kembali menggunakan key `log`.

Masih ingat dengan Facade Log yang kita gunakan sebelumnya? Nah, sebenarnya inilah yang terjadi. Facade tersebut mengambil binding dengan key `log`. Key tersebut akan mengambil contract `Illuminate\Contracts\Logging\Log` yang mana akan berubah menjadi `Illuminate\Log\Writer`. Sehingga, sebenarnya kita bisa melakukan ini untuk membuat Log:

```

2. vagrant@homestead: ~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app('Illuminate\Contracts\Logging\Log')->info("Membuat log dengan contract log");
=> null
>>>

```

Membuat log dengan resolve contract

Dan syntax ini akan berhasil membuat baris log baru di `storage/logs/laravel-xxxx-xx-xx.log`:

storage/logs/laravel-xxxx-xx-xx.log

```
1 [xxxx-xx-xx xx:xx:xx] local.INFO: Membuat log dengan contract log
```

Setelah kita paham bahwa Facade yang kita gunakan ternyata di binding ke contract, kita dapat menggunakan contract sebagai dependency di class yang kita bangun. Ini akan membuat aplikasi kita lebih flexibel. Misalnya, pada class NewsLetter yang kita bangun sebelumnya, setiap kali kita melakukan registrasi user kita ingin menambah log info. Selain menggunakan Facade Log, kita juga bisa menginject contract Illuminate\Contracts\Logging\Log:

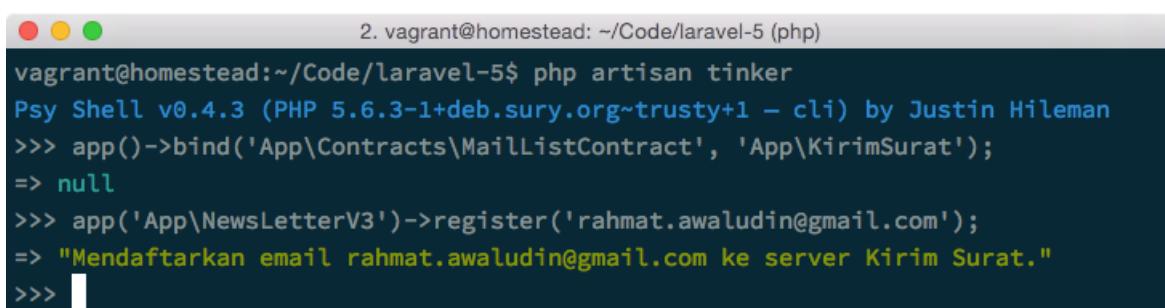
app/NewsLetterV3.php

```
<?php namespace App;

use App\Contracts\MailListContract;
use Illuminate\Contracts\Logging\Log;
class NewsLetterV3 {
    protected $mailList;
    protected $logger;

    public function __construct(MailListContract $mailList, Log $logger)
    {
        $this->mailList = $mailList;
        $this->logger = $logger;
    }
    public function register($email) {
        $this->logger->info('User baru diregistrasi : ' . $email);
        return $this->mailList->register($email);
    }
}
```

Jika dijalankan :



```
vagrant@homestead:~/Code/laravel-5 (php)
vagrant@homestead:~/Code/laravel-5$ php artisan tinker
Psy Shell v0.4.3 (PHP 5.6.3-1+deb.sury.org~trusty+1 - cli) by Justin Hileman
>>> app()->bind('App\Contracts\MailListContract', 'App\KirimSurat');
=> null
>>> app('App\NewsLetterV3')->register('rahmat.awaludin@gmail.com');
=> "Mendaftarkan email rahmat.awaludin@gmail.com ke server Kirim Surat."
>>> 
```

Menjalankan NewsLetterV3 dengan contract log

Sehingga akan terdapat log baru:

storage/logs/laravel-xxxx-xx-xx.log

```
1 [xxxx-xx-xx xx:xx:xx] local.INFO: User baru diregistrasi : rahmat.awaludin@gmail\
2 .com
```

Tetap bisa jalan kan? :)

Yap, memang salah satu tujuan dibuatnya contract ini agar aplikasi yang kita bangun tidak terlalu bergantung ke Facade. Supaya aplikasinya tidak terlalu terikat ke komponen Laravel dan lebih framework agnostic. Keuntungan lain dengan teknik ini, misalnya ketika testing. Kita dapat dengan mudah melakukan binding contract ini ke mocking class dengan mudah.

Yang lebih menarik, ketika kita meng-*inject* contract ke class yang kita bangun, kita tidak perlu membuat class itu secara manual. Karena, semua telah ditangani oleh fitur automatic resolution dari Service Container. Jadi, kita tinggal pakai saja method-method yang ada di contract tersebut.

Semua referensi binding contract ke Facade dapat kita temui di [dokumentasi resmi](#)⁵⁸.

Alur Request

Jika kita menggunakan sebuah alat baru, terkadang semua terasa begitu ajaib jika kita tidak mengetahui cara kerjanya. Begitupun dengan Laravel, sebagai sebuah framework yang sudah dewasa, banyak sekali *moving parts* di dalamnya. Di bagian ini, kita akan mendapatkan *helicopter view* dari alur sebuah request di Laravel.

Sebelum memulai saya ingin sedikit berbagi tentang cara belajar yang sering saya lakukan. Ketika kecil, saya suka sekali bermain dengan mobil 4WD atau biasa disebut Tamiya. Waktu itu, tidak banyak buku panduan atau dokumentasi untuk mempelajari cara kerja mainan tersebut. Karena penasaran dengan cara kerjanya, saya bongkar mainan itu. Dari situ saya memahami cara kerjanya.

Begitupun dengan Laravel, meskipun sudah terdapat dokumentasi resmi di <https://laravel.com/docs>, tapi cara yang lebih efektif (yang sudah saya lakukan) untuk mempelajari cara kerjanya yaitu dengan **membaca source codenya langsung**. Serius, baca source codenya. Menariknya, baris-baris source code Laravel biasanya sudah dilengkapi dengan komen yang deskriptif. Tentunya, dalam bahasa Inggris.

Untuk mempelajari alur request, kita harus mengetahui bahwa Laravel menggunakan teknik [Front Controller Pattern](#)⁶⁰. Saya tidak akan menjelaskan teknik ini secara

⁵⁸<http://laravel.com/docs/5.0/contracts#contract-reference>

⁵⁹<https://laravel.com/docs>

⁶⁰http://en.wikipedia.org/wiki/Front_Controller_pattern

detail, nanti saja di buku selanjutnya. Sederhananya, teknik ini memungkinkan kita hanya menggunakan satu file index.php untuk melayani semua URL yang di-request. Jika tertarik untuk membuat sample untuk Front Controller ini dengan native PHP, silahkan baca [tutorial di Sitepoint](#)⁶¹.

Request di Laravel terbagi menjadi dua, HTTP Request dan Console Request. HTTP Request adalah semua request yang berupa HTTP, misalnya dari browser, curl, dll. Gerbang masuk untuk HTTP Request adalah file **public/index.php**. Sementara Console Request adalah request ketika kita mengakses aplikasi menggunakan perintah `php artisan tinker`. Gerbang masuk untuk Console Request adalah file **artisan**.

Karena alur untuk keduanya hampir sama, kita hanya akan mempelajari alur untuk HTTP Request.

Ketika kita browser melakukan request, maka semua URL yang masuk akan diarahkan ke file `public/index.php`. Untuk memudahkan pemahaman, **silahkan buka file tersebut**.

Secara umum, file ini melakukan 3 hal:

1. Menginisialisasi Autoload.
2. Menginisialisasi Service Container.
3. Melayani request dengan HTTP Kernel.

Tahap pertama, terlihat pada baris ini:

public/index.php

```
9  /*
10  -----
11  / Register The Auto Loader
12  -----
13  /
14  / Composer provides a convenient, automatically generated class loader for
15  / our application. We just need to utilize it! We'll simply require it
16  / into the script here so that we don't have to worry about manual
17  / loading any of our classes later on. It feels nice to relax.
18  /
19  */
20
21 require __DIR__ . '/../../bootstrap/autoload.php';
```

⁶¹ www.sitepoint.com/front-controller-pattern-1/

Disini, Laravel me-load file `bootstrap/autoload.php`. Pada file tersebut kita akan menemukan bahwa Laravel menggunakan `autoload` dari composer. Pada file tersebut kita juga bisa melihat Laravel me-load class yang di compile di folder `vendor/compiled.php` (hasil perintah `php artisan optimize`).

Tahap kedua, terlihat pada baris:

public/index.php

```
23 /*  
24 /-----  
25 / Turn On The Lights  
26 /-----  
27 /  
28 / We need to illuminate PHP development, so let us turn on the lights.  
29 / This bootstraps the framework and gets it ready for use, then it  
30 / will load up this application so that we can run it and send  
31 / the responses back to the browser and delight our users.  
32 /  
33 */  
34  
35 $app = require_once __DIR__ . '/../bootstrap/app.php';
```

Disini, Laravel melakukan inisialisasi Service Container. Ini dilakukan dengan cara me-load file `bootstrap/app.php`. Pada file tersebut, Laravel akan melakukan 3 hal:

1. Membuat instance dari `Illuminate\Foundation\Application` yang merupakan Service Container. Instance Service Container ini yang kedepannya berfungsi mengikat (bind) ke semua komponen Laravel. Di balik layar, pada langkah ini kita juga mengaktifkan Service Provider yang telah kita buat.
2. Binding beberapa contract yang penting (`Illuminate\Contracts\Http\Kernel`, `Illuminate\Contracts\Console\Kernel` dan `Illuminate\Contracts\Debug\ExceptionHandler`).
3. Mengembalikan instance dari `Illuminate\Foundation\Application`.

Setelah Laravel mendapatkan instance `Illuminate\Foundation\Application` dari tahap kedua. Maka, Laravel menyimpannya di dalam variable `$app`.

Tahap ketiga, terlihat pada baris:

public/index.php

```
37 /*  
38 /-----  
39 / Run The Application  
40 /-----  
41 /  
42 / Once we have the application, we can handle the incoming request  
43 / through the kernel, and send the associated response back to  
44 / the client's browser allowing them to enjoy the creative  
45 / and wonderful application we have prepared for them.  
46 /  
47 */  
48  
49 $kernel = $app->make('Illuminate\Contracts\Http\Kernel');  
50  
51 $response = $kernel->handle(  
52     $request = Illuminate\Http\Request::capture()  
53 );  
54  
55 $response->send();  
56  
57 $kernel->terminate($request, $response);
```

Disini, Laravel memberikan respon dari request yang diberikan menggunakan instance dari `Illuminate\Contracts\Http\Kernel`. Caranya, dengan:

1. Melakukan resolve `Illuminate\Contracts\Http\Kernel` dari Service Container dan menyimpannya ke variable `$kernel`.
2. Meng-generate response dengan menggunakan method `$kernel->handle()` dan menyimpannya di variable `$response`.
3. Mengirim response dengan method `$response->send()`.
4. Mematikan semua middleware yang bisa dimatikan

Dapat kita lihat, alurnya cukup sederhana. Jika dianalogikan, bayangkan HTTP Kernel sebagai seorang pelayan di restoran. Kita cukup memesan jenis makanan (*request*), dia akan kembali dengan makanan yang kita pesan (*response*). Di balik layar, banyak hal yang sebenarnya terjadi. Dia memberikan pesanan kita ke koki, koki menyiapkan semua bahan, memasak, menyiapkan tempat, dan lain-lain. Hingga akhirnya, makanan akan ada di meja kita.

Teknik yang Laravel gunakan untuk HTTP Kernel ini dinamakan [Command Design Pattern](#)⁶². Saya berencana untuk menjelaskan lebih detail pada buku tentang Design

⁶²en.wikipedia.org/wiki/Command_pattern

Pattern. Untuk sekarang, konsep dasar dari teknik ini adalah kita menggunakan method yang sama untuk mengirim perintah berikut instruksinya (di Laravel, method `handle()` dan URL yang kita kirim) dan menerima outputnya (di Laravel, method `send()`). Sip.



Source code dari bab ini bisa didapat di <https://github.com/rahmatawaludin/arsitektur-laravel>⁶³



Ringkasan

Pada bab ini kita telah mempelajari sisi fundamental dari Laravel. Dari struktur aplikasi, service provider, service container, facade, contract dan alur request. Semua API Laravel dibangun diatas teknik fundamental ini. Jadi, memahami hal-hal fundamental ini, akan sangat membantu kita dalam mempelajari fitur Laravel yang lain.

Masih banyak yang akan kita bahas. Bab selanjutnya, kita belajar tentang routing. Semangat! :)

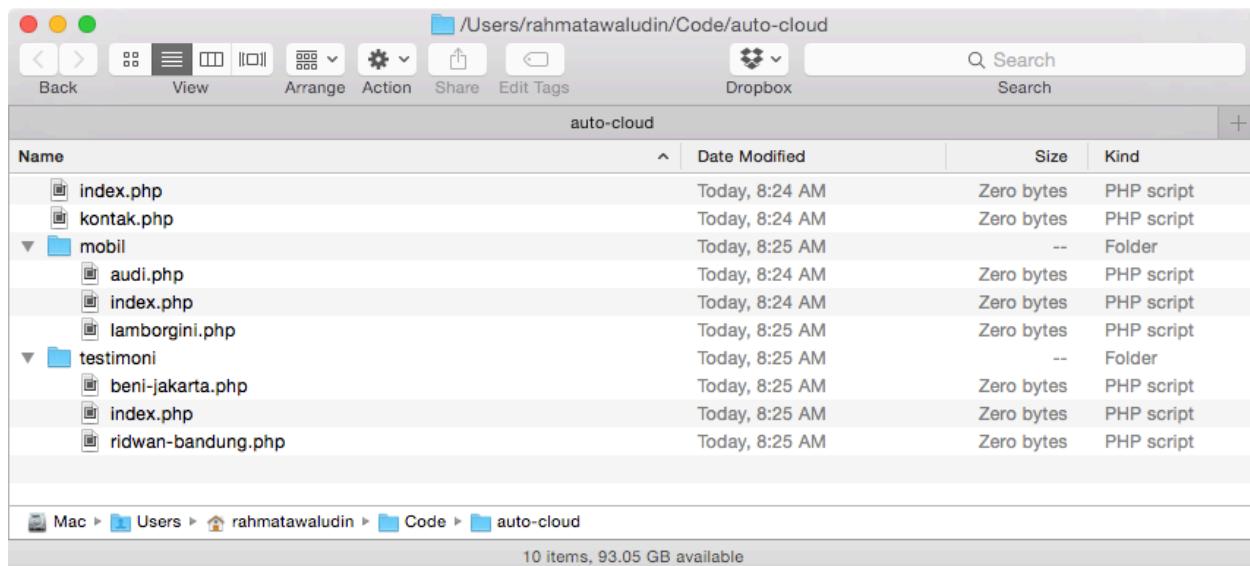
⁶³<https://github.com/rahmatawaludin/arsitektur-laravel>

Routing, Kendalikan Alur Aplikasi

Dalam membangun website, URL yang mudah diingat itu penting. Selain memudahkan user untuk mengingat alamat web, URL yang deskriptif juga sangat berpengaruh dalam indexing google.

Kenapa menggunakan routing?

Laravel menangani perihal URL ini dengan serius. Jika menggunakan PHP native, untuk membangun URL, biasanya dibangun dengan subfolder. Misalnya, kita memiliki website penjualan mobil dengan submenu daftar mobil, kontak dan testimoni. Kurang lebih seperti ini struktur folder yang akan kita bangun:



Sample struktur folder untuk URL dengan PHP Native

Terlihat disini, kita menginginkan halaman home (index.php), kontak (kontak.php), daftar mobil (mobil/index.php), detail tiap mobil (mobil/audi.php, mobil/lamborgini.php), daftar testimoni (testimoni/index.php) dan detail testimoni (testimoni/beni-jakarta.php, testimoni/ridwan-bandung.php).

Berbeda di Laravel, kita tidak perlu membuat struktur folder seperti itu untuk membangun URL yang diinginkan. Di Laravel, semua routing di simpan pada file route. Jadi, semua URL yang ada di aplikasi dapat dilihat dalam file tersebut.

Konfigurasi Routing

Secara default Laravel menyimpan konfigurasi routing di file `app/Providers/RouteServiceProvider.php`. Untuk sekarang, method yang perlu kita pahami adalah method `map()`:

`app/Providers/RouteServiceProvider.php`

```
36 public function map(Router $router)
37 {
38     $router->group(['namespace' => $this->namespace], function($router)
39     {
40         require app_path('Http/routes.php');
41     });
42 }
```

Pada method ini Laravel me-*load* file routing default yaitu di `app/Http/routes.php`. Masih ingat dengan penjelasan Service Provider? Nah, di method ini kita bisa me-*load* file routes yang lain jika diinginkan.

Memahami HTTP Verb

Untuk memahami penggunaan routing Laravel, kita harus memahami HTTP Verb. Verb atau kata kerja dalam HTTP, menentukan jenis aksi yang akan dilakukan oleh request yang dilakukan. Terdapat 5 jenis verb yang didukung Laravel:

1. **GET** : Digunakan untuk meminta resource dari server.
2. **POST** : Digunakan untuk menyimpan resource ke server.
3. **PUT** : Digunakan untuk melakukan update resource di server.
4. **PATCH** : Digunakan untuk melakukan update resource di server.
5. **DELETE** : Digunakan untuk menghapus resource di server.

PUT dan PATCH sering digunakan untuk hal yang sama (*meng-update*). Perbedaan keduanya, secara teoritis PUT bekerja dengan menimpa (*replace*) resource dengan resource yang baru. Sementara PATCH digunakan untuk *meng-update* sebagian field dari resource. Tapi, pada aplikasi di lapangan, saya sendiri hanya menggunakan PUT untuk *meng-update* resource.

Penjelasan diatas merupakan gambaran sederhana dari tiap verb tersebut. Jika butuh yang lebih detail, silahkan baca [disini](#)⁶⁴.

Syntax dasar routing di Laravel menggunakan HTTP Verb dan Closure.

⁶⁴<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

```
Route::get('...', ...);
Route::post('...', ...);
Route::put('...', ...);
Route::patch('...', ...);
Route::delete('...', ...);
```

Misalnya untuk membuat URL ke halaman kontak, kita bisa menambah baris berikut di file `app/Http/routes.php`:

app/Http/routes.php

```
...
Route::get('kontak', function() {
    return '<h1>halaman kontak</h1>';
});
```

Kini kita bisa mengakses /kontak dari web browser.



route kontak

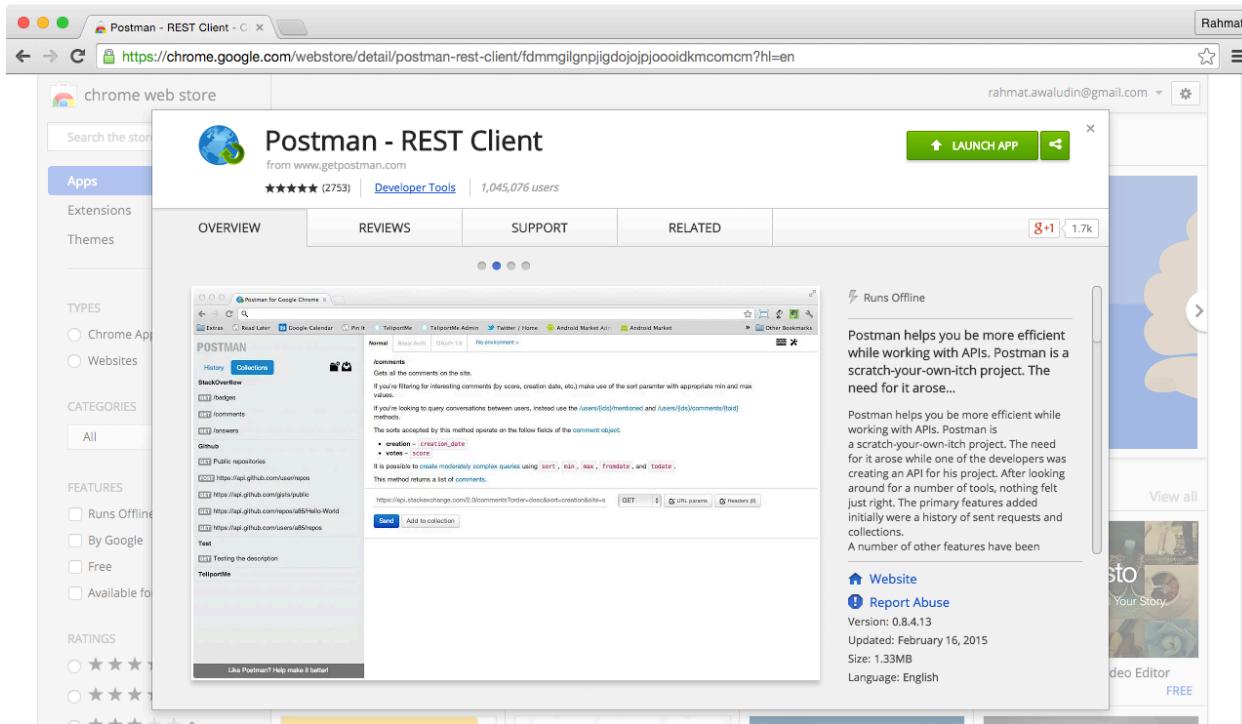
Kita juga bisa membuat Route untuk Post. Misalnya, kita akan menerima pesan dari form dan menampilkan pesannya. Kita buat route seperti ini:

app/Http/routes.php

```
...
Route::post('kontak-post', function() {
    return 'Anda mengirim pesan "' . $_POST['pesan'] . '"';
});
```

Untuk mengeceknya, mari kita pakai ekstensi Postman - REST Client⁶⁵ di Google Chrome.

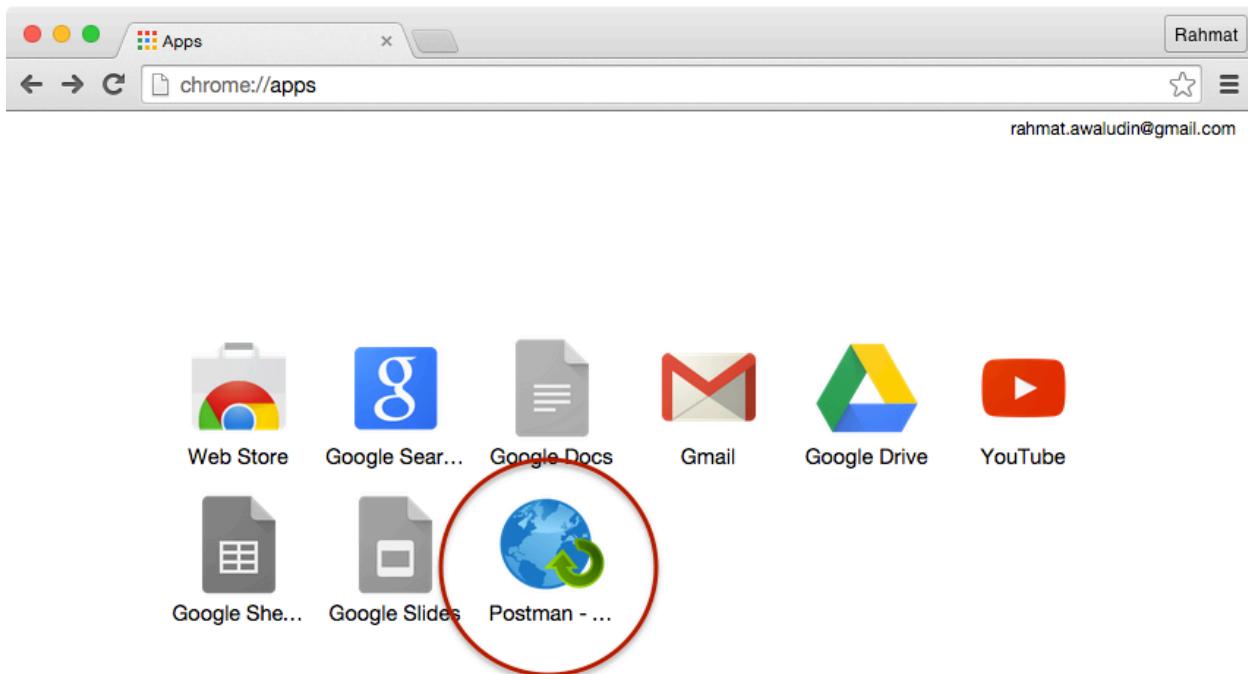
⁶⁵ <https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjgdojopjoooidkmcomcm?hl=en>



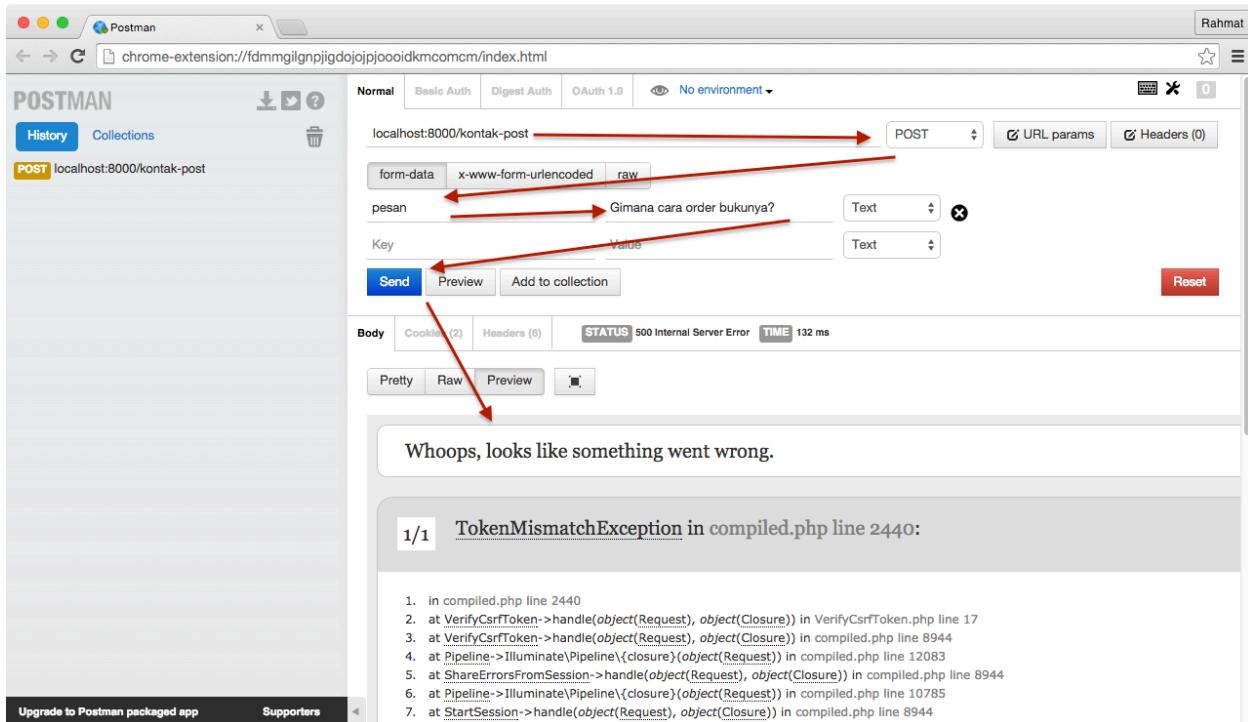
Postman - REST Client

Setelah terinstall, kita dapat mengakses Postman dengan mengunjungi URL <chrome://apps>⁶⁶. Kemudian klik pada icon Postman.

⁶⁶<chrome://apps>



Setelah Postman terbuka, kita dapat mengetes route post tadi dengan memasukan URL dan form data yang diinginkan (`pesan`). Pada contoh ini, untuk memudahkan kita menggunakan local server (`php artisan serve`), sehingga URL nya `http://localhost:8000`. Jika ingin menggunakan virtual host, silahkan sesuaikan URLnya. Isi juga form data dengan **key** `pesan` dan **value** dengan pesan yang kita inginkan. Kemudian klik **Send**.



Postman gagal

Oooopppsss... ternyata ada yang error. Ini terjadi karena secara default fitur security Laravel mengaktifkan middleware app/Http/Middleware/VerifyCsrfToken.php untuk menghindari serangan CSRF⁶⁷ ke website kita. Untuk sementara, mari kita non-aktifkan middleware tersebut dengan mengubah baris:

app/Http/Kernel.php

```
18 'App\Http\Middleware\VerifyCsrfToken',
```

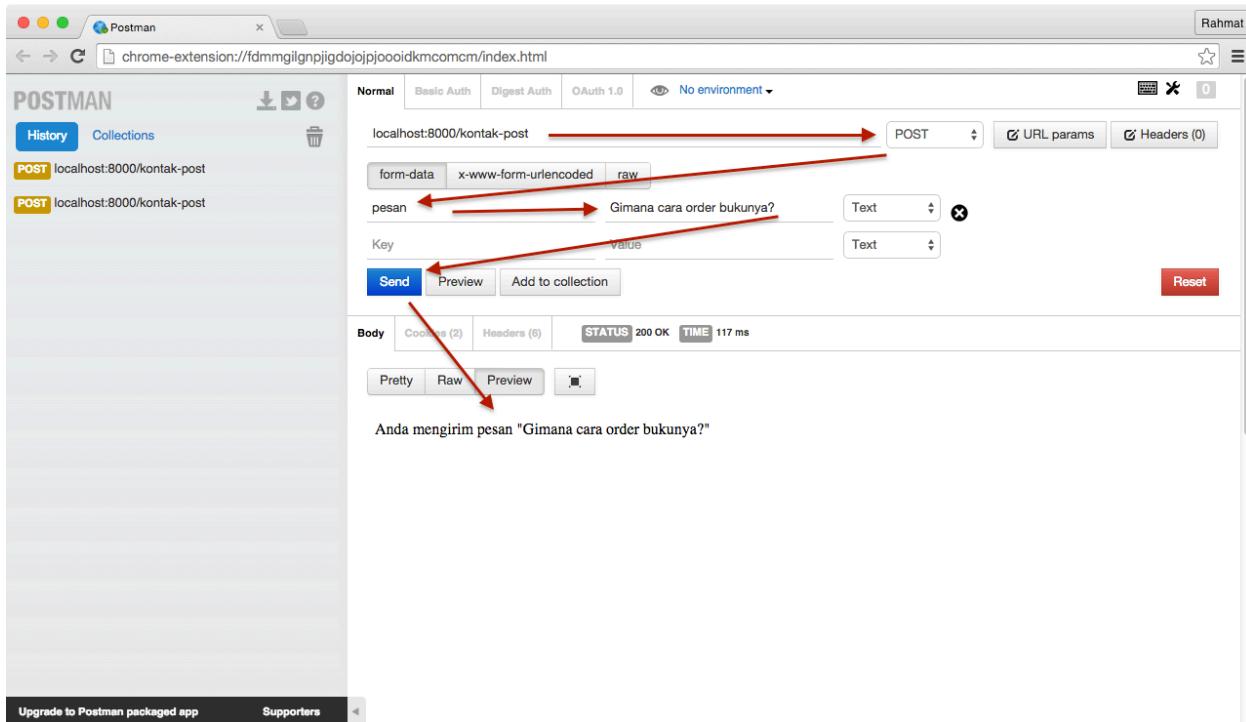
Menjadi:

app/Http/Kernel.php

```
18 // 'App\Http\Middleware\VerifyCsrfToken',
```

Kini kita coba lagi...

⁶⁷ https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet

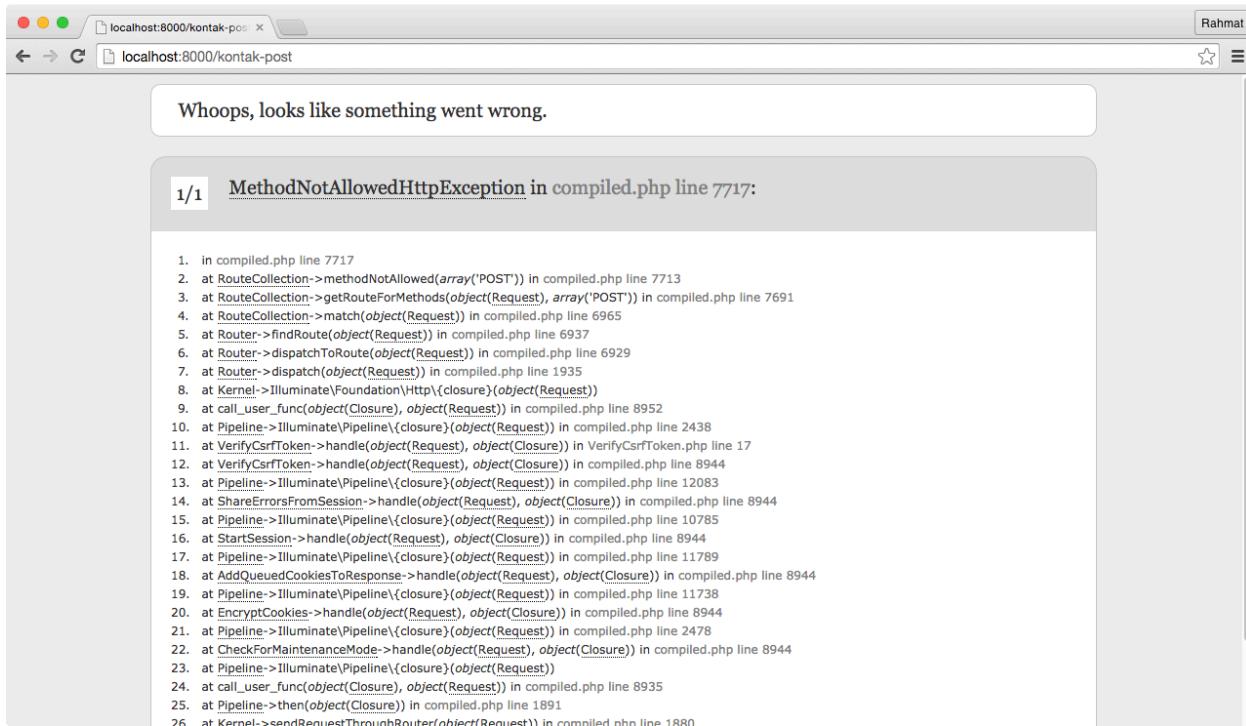


Postman Berhasil

Terlihat disini, kita berhasil membuat route post dan mengirim data menggunakan Postman.

Menggunakan Beberapa Verb untuk Satu Route

Terkadang kita membutuhkan route yang bisa diakses oleh beberapa verb. Jika menggunakan teknik sebelumnya, ketika kita mencoba mengakses route POST dengan method GET maka akan muncul error `MethodNotAllowedHttpException`. Karena, HTTP Verb yang kita gunakan berbeda dengan yang didefinisikan di file routes.



Tidak bisa mengakses route post dengan method get

Begitupun ketika kita mencoba mengakses route GET dengan method POST, akan muncul error

The screenshot shows the Postman application interface. In the top navigation bar, 'POSTMAN' is selected. Below it, a search bar shows 'localhost:8000/kontak'. The main area is set up for a 'POST' request, with the URL 'localhost:8000/kontak' in the address bar. The 'Body' tab is active, showing a 'form-data' section with two fields: 'pesan' (Value: 'Gimana cara order bukunya?') and 'Key' (Value: 'Value'). Below the body, the status bar indicates 'STATUS 405 Method Not Allowed TIME 214 ms'. A message box at the bottom says 'Whoops, looks like something went wrong.' A detailed error message follows:

```

1/1 MethodNotAllowedHttpException in compiled.php line 7717:
1. in compiled.php line 7717
2. at RouteCollection->methodNotAllowed(array('GET', 'HEAD')) in compiled.php line 7713
3. at RouteCollection->getRouteForMethods(object(Request), array('GET', 'HEAD')) in compiled.php line 7691
4. at RouteCollection->match(object(Request)) in compiled.php line 6965
5. at Router->findRoute(object(Request)) in compiled.php line 6937
6. at Router->dispatchToRoute(object(Request)) in compiled.php line 6929
7. at Router->dispatch(object(Request)) in compiled.php line 1935

```

Tidak bisa mengakses route get dengan method post

Beberapa Verb

Jika diinginkan, kita bisa membuat route yang bisa diakses dengan berbagai method menggunakan `Route::match()`. Misalnya, jika kita ingin agar route /kontak dapat diakses dengan GET dan POST, bisa kita ubah seperti ini:

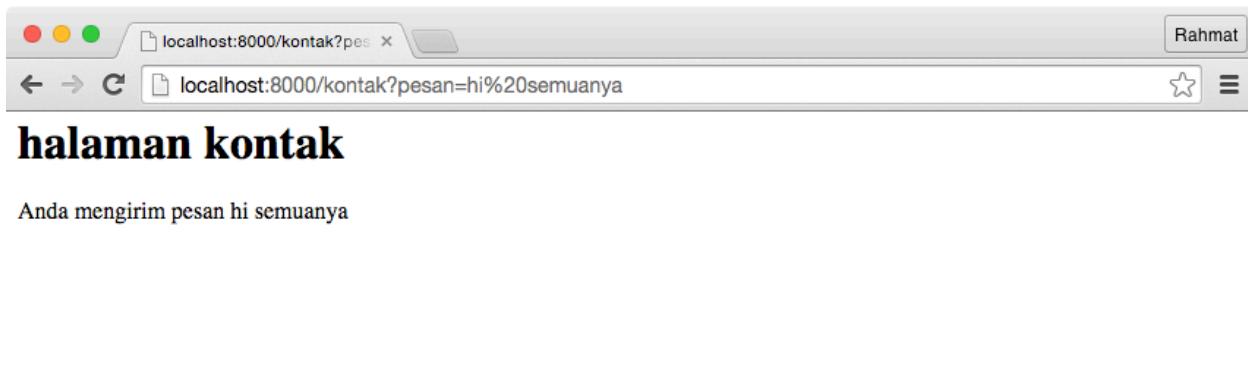
app/Http/routes.php

```

1 Route::match(['get', 'post'], 'kontak', function() {
2     $html = '<h1>halaman kontak</h1>';
3     if (isset($_REQUEST['pesan'])) {
4         $html .= 'Anda mengirim pesan ' . $_REQUEST['pesan'];
5     }
6     return $html;
7 });

```

Disini, kita mengecek variable request (`$_POST` atau `$_GET`) dengan `$_REQUEST['pesan']` dan menampilkan nilainya jika ada. Kita dapat mengaksesnya dengan request GET atau POST:



Mengakses /kontak dengan method GET

A screenshot of the Postman application interface. It shows a POST request to "localhost:8000/kontak". The request body is set to "form-data" and contains a key-value pair: "pesan" with value "Siapa yang bikin Laravel?". The response status is 200 OK with a time of 132 ms. The response body displays the same "halaman kontak" and message as the browser screenshot.

Mengakses /kontak dengan method POST

Semua Verb

Selain menggunakan `Route::match()`, kita juga dapat menggunakan `Route::any()` agar route dapat diakses menggunakan HTTP Verb apapun. Misalnya, route kontak yang tadi dapat kita ubah menjadi:

app/Http/routes.php

```

1 Route::any('kontak', function() {
2     $html = '<h1>halaman kontak</h1>';
3     $html .= 'Anda mengakses dengan method ' . Request::method();
4     return $html;
5 });

```

Disini, kita menampilkan jenis request yang dilakukan oleh user dengan facade Request::method(). Mari kita coba:

The screenshot shows the Postman interface. In the top bar, the URL is set to `http://localhost:8000/kontak`. Below it, the method dropdown is circled in red and is set to `GET`. The response status is `200 OK`. The response body contains the text `halaman kontak` and `Anda mengakses dengan method GET`.

Mengakses dengan method GET

The screenshot shows the Postman interface. In the top bar, the URL is set to `http://localhost:8000/kontak`. Below it, the method dropdown is circled in red and is set to `POST`. The response status is `200 OK`. The response body contains the text `halaman kontak` and `Anda mengakses dengan method POST`. There are also some warning messages at the bottom: `Deprecated: Automatically populating $HTTP_RAW_POST_DATA is deprecated and will be removed in a future version. To avoid this warning set 'always_populate_raw_post_data' to '1' in php.ini and use the php://input stream instead.` and `Warning: Cannot modify header information - headers already sent in Unknown on line 0`.

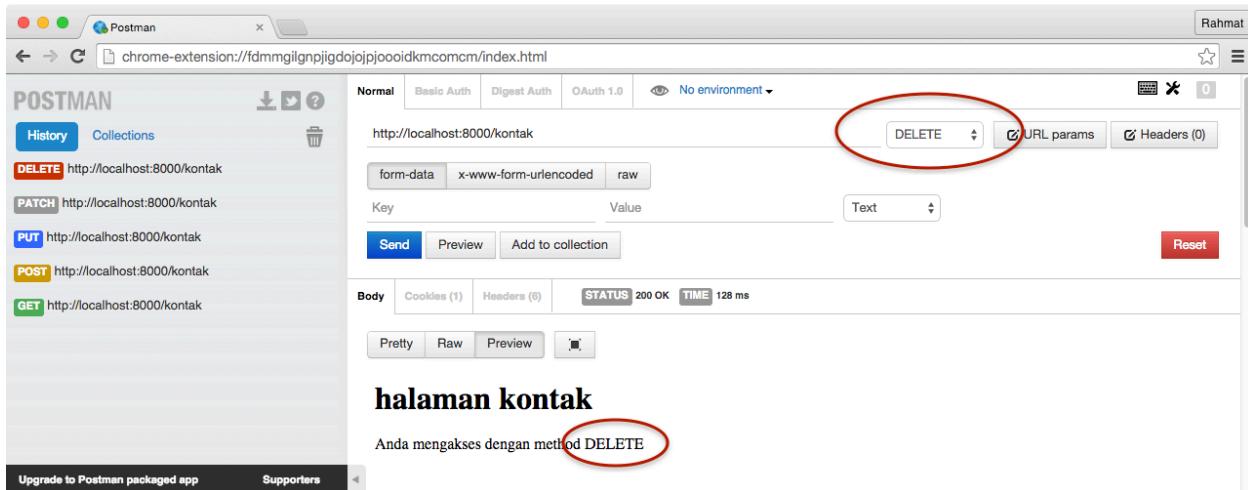
Mengakses dengan method POST

The screenshot shows the Postman application interface. The URL in the header is `http://localhost:8000/kontak`. The method dropdown is set to `PATCH`, which is highlighted with a red circle. The response status is `200 OK`. The main content area displays the text **halaman kontak** and **Anda mengakses dengan method PATCH**.

Mengakses dengan method PATCH

The screenshot shows the Postman application interface. The URL in the header is `http://localhost:8000/kontak`. The method dropdown is set to `PUT`, which is highlighted with a red circle. The response status is `200 OK`. The main content area displays the text **halaman kontak** and **Anda mengakses dengan method PUT**.

Mengakses dengan method PUT



Mengakses dengan method DELETE

Selain menampilkan jenis method request, kita juga dapat mengecek jenis request yang digunakan dengan facade `Request::isRequest::isMethod()`. Misalnya, jika menggunakan method `DELETE` kita akan menampilkan pesan tambahan, maka kita dapat mengubahnya menjadi:

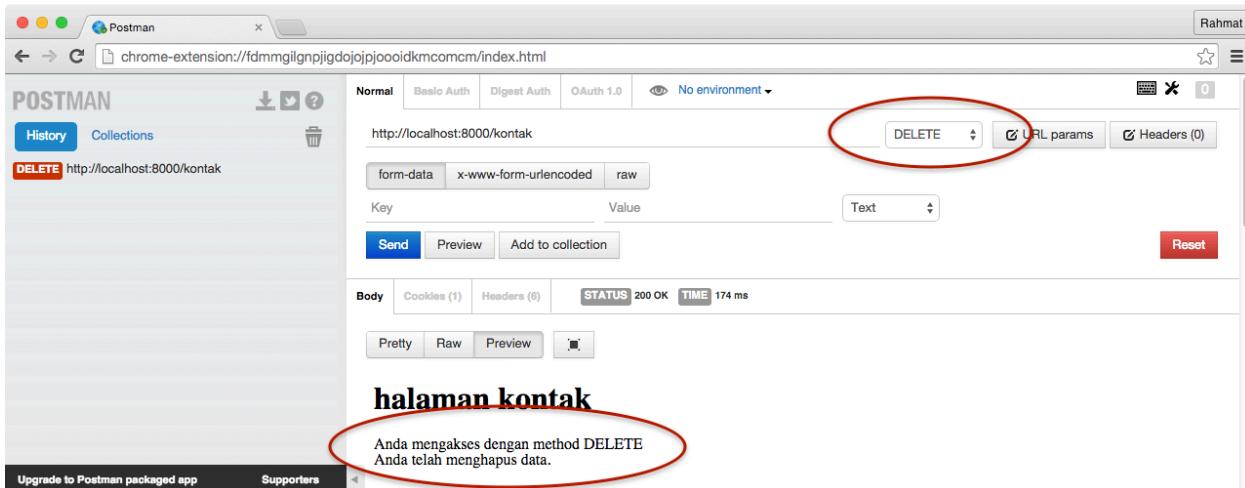
app/Http/routes.php

```

1 Route::any('kontak', function() {
2     $html = '<h1>halaman kontak</h1>';
3     $html .= 'Anda mengakses dengan method ' . Request::method();
4     if ( Request::isMethod('delete') ) {
5         $html .= "<br>Anda telah menghapus data.";
6     }
7     return $html;
8 });

```

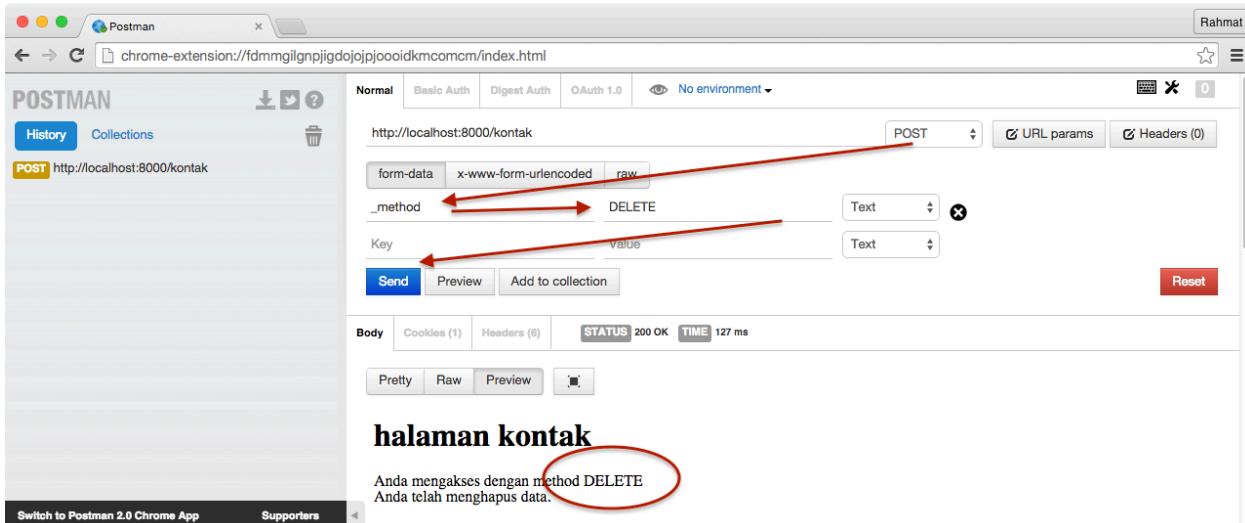
Jika kembali kita check dengan method `DELETE` pada Postman. Maka pesan tersebut akan muncul:



Mengecek method dengan isMethod

Method Spoofing

Karena secara default browser hanya mendukung verb **POST** dan **GET**, maka untuk verb lainnya digunakan method spoofing. Cara kerja method spoofing adalah dengan menggunakan **POST** request dan menambah key baru bernama `_method` yang berisi HTTP Verb yang akan kita gunakan. Misalnya, untuk membuat **DELETE** request, ini yang akan kita buat :



Melakukan Spoofing untuk DELETE Request

Telihat disini, request kita masih terdeteksi sebagai DELETE Request. Begitupun untuk PUT dan PATCH :

POSTMAN

Normal Basic Auth Digest Auth OAuth 1.0 No environment

http://localhost:8000/kontak

POST

Method: PATCH

Key Value

Send Preview Add to collection Reset

Body Cookies (1) Headers (6) STATUS 200 OK TIME 122 ms

Pretty Raw Preview

halaman kontak

Anda mengakses dengan method PATCH

Melakukan Spoofing untuk PATCH Request

POSTMAN

Normal Basic Auth Digest Auth OAuth 1.0 No environment

http://localhost:8000/kontak

POST

Method: PUT

Key Value

Send Preview Add to collection Reset

Body Cookies (1) Headers (6) STATUS 200 OK TIME 132 ms

Pretty Raw Preview

halaman kontak

Anda mengakses dengan method PUT

Melakukan Spoofing untuk PUT Request

Memahami Parameter dalam Routing

Dalam membangun URL biasanya kita ingin menerima input dari user. Jika menggunakan PHP Native, parameter dari URL dapat dibuat dalam bentuk variable. Misalnya, kita membuat URL `/welcome?nama=Joni`. Disini kita menerima paramter `nama` dan berisi nilai `Joni`. Teknik seperti ini, tentunya bisa dipakai di Laravel. Misalnya dengan route seperti berikut:

app/Http/routes.php

```
....  
Route::get('welcome', function() {  
    return "Selamat datang " . $_REQUEST['nama'] . ". Anda luar biasa!";  
});
```

Maka, kita dapat mengaksesnya:

**Route dengan parameter GET**

Selain menggunakan cara diatas, di Laravel kita dapat menggunakan parameter dalam route. Caranya, dengan menambahkan `{key}` pada route yang kita buat dan melakukan passing ke closure. Misalnya, route diatas dapat kita ubah menjadi `welcome/joni` dengan mengubahnya menjadi:

app/Http/routes.php

```
....  
Route::get('welcome/{nama}', function($nama) {  
    return "Selamat datang " . $nama . ". Anda luar biasa!";  
});
```

Terlihat disini, setelah kita melakukan passing parameter route ke closure, kita dapat mengakses variable tersebut (`$nama`) dari dalam closure.



Parameter dalam Route

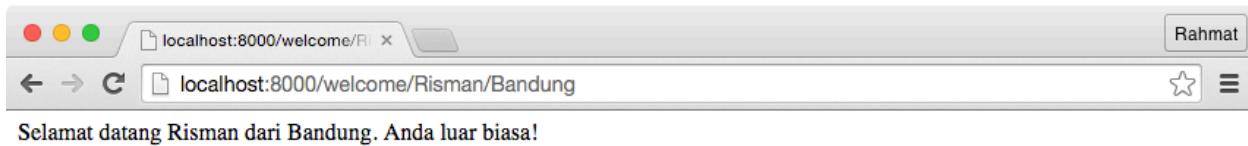
Tentunya, kita juga dapat memberikan lebih dari satu parameter. Misalnya, kita berikan parameter nama dan asal.

app/Http/routes.php

```
...  
Route::get('welcome/{nama}/{asal}', function($nama, $asal) {  
    return "Selamat datang " . $nama . " dari " . $asal . ". Anda luar biasa!";  
});
```

Terlihat disini, agar dapat mengakses variabel yang lain, selain kita menambah parameter pada URL, kita juga perlu menambah pada closure.

Kini kita dapat mengakses memberikan dua parameter:



Dua parameter

Regex Parameter

Kita juga bisa membatasi parameter dalam route dengan regex. Misalnya, variable nama yang tadi kita buat akan dibatasi hanya berupa huruf besar dan kecil (tanpa

angka dan karakter khusus lain). Maka, ini syntax route yang perlu kita buat:

app/Http/routes.php

```
....  
Route::get('welcome/{nama}', ['as'=>'home.welcome', function($nama) {  
    return "Selamat datang " . $nama . ". Anda luar biasa!";  
}])->where('nama', '[A-Za-z]+');
```

Kini jika kita mencoba memberikan parameter selain huruf, misalnya angka akan muncul error:

Sorry, the page you are looking for could not be found.

1/1 **NotFoundHttpException** in compiled.php line 7693:

```
1. in compiled.php line 7693
2. at RouteCollection->match(object(Request)) in compiled.php line 6965
3. at Router->findRoute(object(Request)) in compiled.php line 6937
4. at Router->dispatchToRoute(object(Request)) in compiled.php line 6929
5. at Router->dispatch(object(Request)) in compiled.php line 1935
6. at Kernel->Illuminate\Foundation\Http\{closure}(object(Request))
7. at call_user_func(object(Closure), object(Request)) in compiled.php line 8952
8. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 2438
9. at VerifyCsrfToken->handle(object(Request), object(Closure)) in VerifyCsrfToken.php line 17
10. at VerifyCsrfToken->handle(object(Request), object(Closure)) in compiled.php line 8944
11. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 12083
12. at ShareErrorsFromSession->handle(object(Request), object(Closure)) in compiled.php line 8944
13. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 10785
14. at StartSession->handle(object(Request), object(Closure)) in compiled.php line 8944
15. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 11789
16. at AddQueuedCookiesToResponse->handle(object(Request), object(Closure)) in compiled.php line 8944
17. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 11738
18. at EncryptCookies->handle(object(Request), object(Closure)) in compiled.php line 8944
19. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in compiled.php line 2478
20. at CheckForMaintenanceMode->handle(object(Request), object(Closure)) in compiled.php line 8944
21. at Pipeline->Illuminate\Pipeline\{closure}(object(Request))
22. at call_user_func(object(Closure), object(Request)) in compiled.php line 8935
23. at Pipeline->then(object(Closure)) in compiled.php line 1891
24. at Kernel->sendRequestThroughRouter(object(Request)) in compiled.php line 1880
25. at Kernel->handle(object(Request)) in index.php line 53
26. at require_once('/Users/rahmatawalidin/Code/sample-routing/public/index.php') in server.php line 21
```

Parameter tidak sesuai dengan Regex

Tapi, jika kita coba memberikan parameter berupa huruf, maka route ini bisa diakses:



Parameter sesuai dengan Regex

Global Pattern

Jika kita menggunakan regex pattern yang sama untuk beberapa route, akan lebih efektif jika kita jadikan global pattern. Caranya dengan menambahkan `Route::pattern()` pada file route. Contohnya, mari kita buat pattern nama tadi menjadi global.

Tambahkan isian berikut pada file baris kedua pada file routes:

app/Http/routes.php

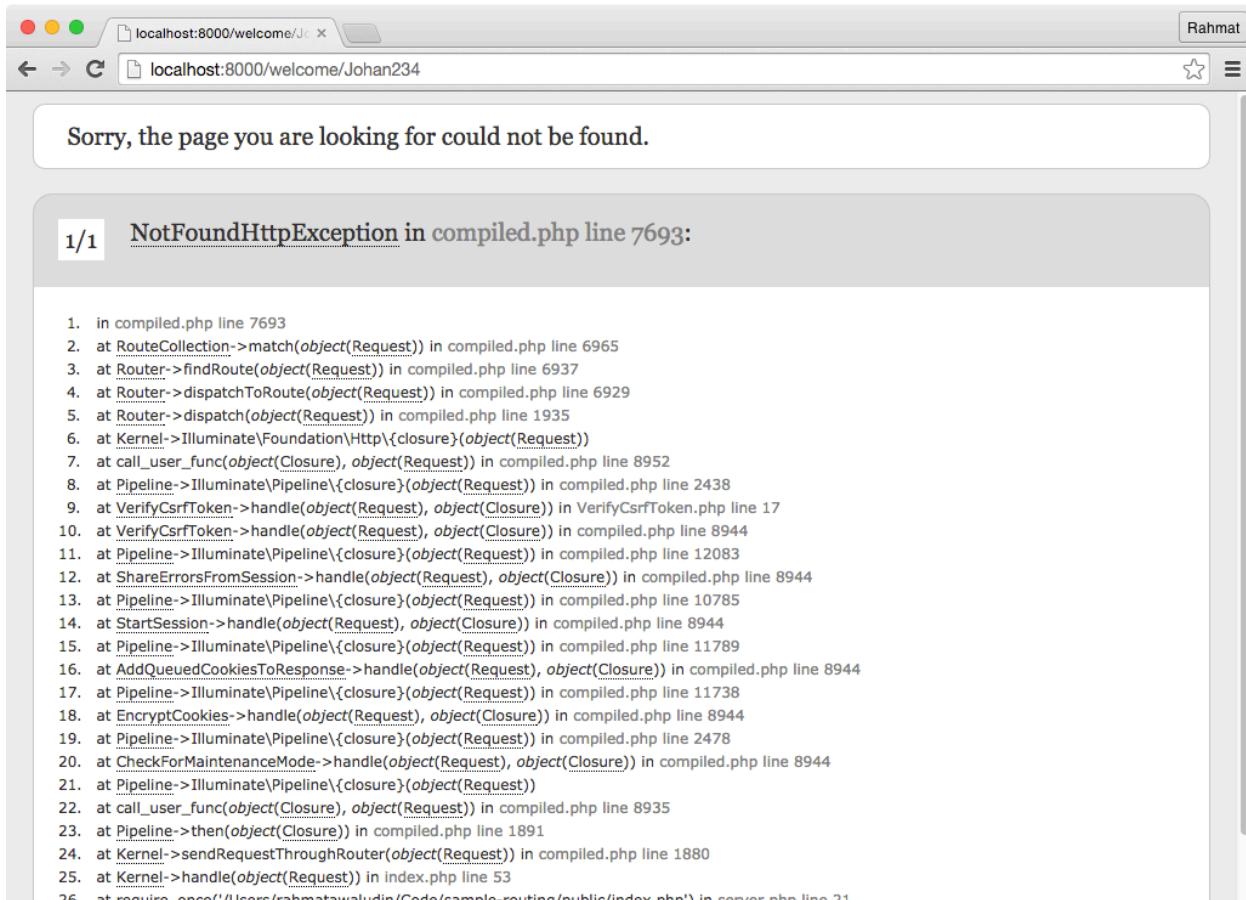
```
1 <?php
2 Route::pattern('nama', '[A-Za-z]+');
3 ...
```

Kini, kita dapat merubah route /welcome menjadi:

app/Http/routes.php

```
...
Route::get('welcome/{nama}', ['as'=>'home.welcome', function($nama) {
    return "Selamat datang " . $nama . ". Anda luar biasa!";
}]);
```

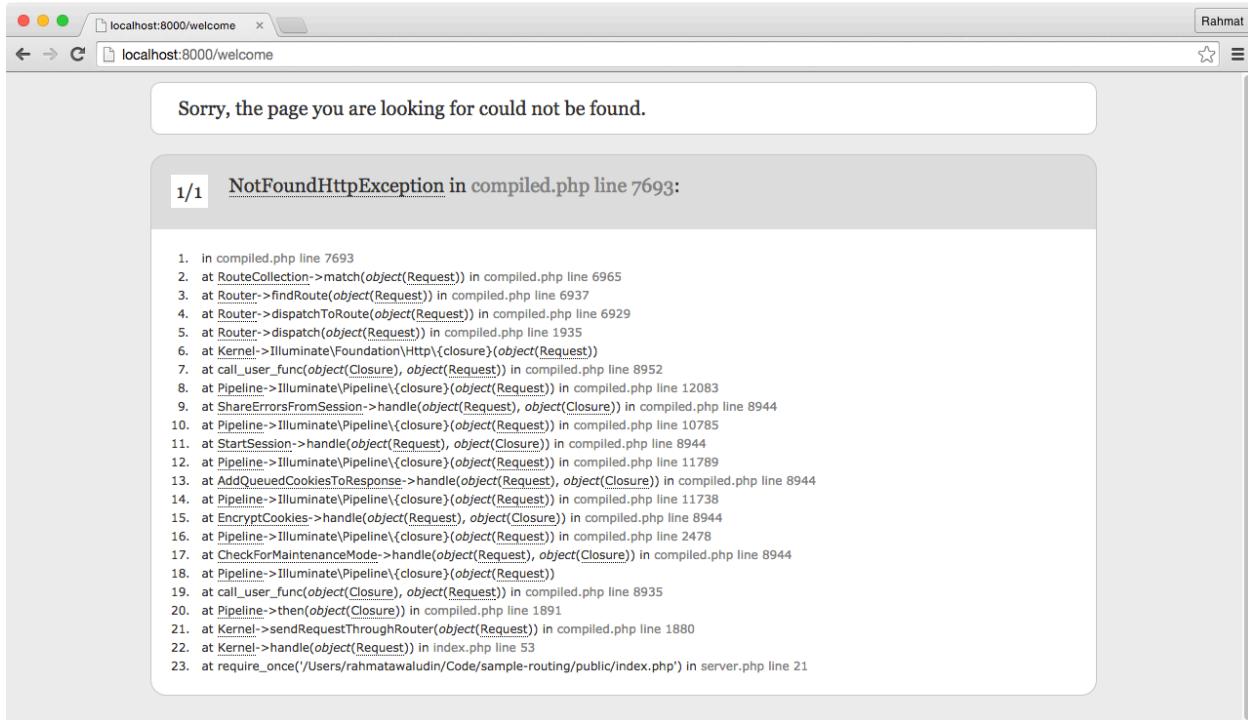
Ketika parameter tidak sesuai dengan pattern yang kita tentukan, dia akan tetap memunculkan error.



Membuat Global Pattern

Optional Parameter

Jika kita mencoba mengakses URL barusan tanpa parameter, maka akan muncul error:



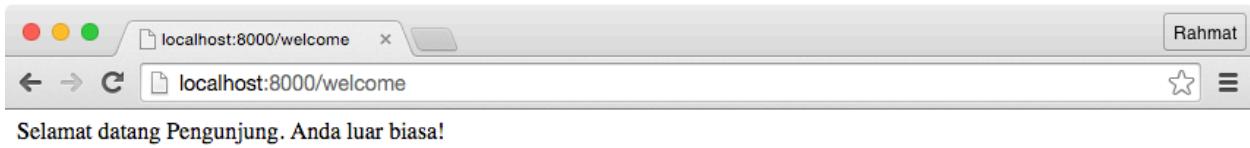
Route error tanpa parameter

Ini terjadi karena parameter kita jadikan mandatory. Jika diinginkan, kita dapat membuat parameter ini menjadi optional. Caranya dengan menambahkan membuat key menjadi `{key?}` dan memberikan nilai default pada closure. Misalnya, kita akan membuat `nama` menjadi optional dan memberi nilai default `Pengunjung`, maka route kita ubah menjadi:

app/Http/routes.php

```
...  
Route::get('welcome/{nama?}', function($nama = 'Pengunjung') {  
    return "Selamat datang " . $nama . ". Anda luar biasa!";  
});
```

Kini, walaupun tanpa parameter, route ini tidak akan error:



Optional Route Parameter

Memahami Named Routes

Laravel mengizinkan kita untuk menamai route yang kita buat menggunakan key `as`. Misalnya, route `welcome` yang sebelumnya mau kita namai `home.welcome` maka syntaxnya kita ubah menjadi:

`app/Http/routes.php`

```
....  
Route::get('welcome/{nama?}', ['as'=>'home.welcome', function($nama = 'Pengunjung') {  
    return "Selamat datang " . $nama . ". Anda luar biasa!";  
}]);
```

Terlihat disini, perbedaan utama adalah parameter kedua kini kita ubah menjadi array. Setelah kita memberikan nama untuk route, jika kita cek dengan `php artisan route:list` akan terlihat nama dari route yang kita buat:

| Domain | Method | URI | Name | Action | Middleware |
|--------|--------------------------------|-------------------------------------------------------|--------------|------------------------------------------------------------|------------|
| | GET HEAD | / | | App\Http\Controllers\WelcomeController@index | guest |
| | GET HEAD | home | home.index | App\Http\Controllers\HomeController@index | auth |
| | GET HEAD | home/create | home.create | App\Http\Controllers\HomeController@create | auth |
| | POST | home | home.store | App\Http\Controllers\HomeController@store | auth |
| | GET HEAD | home/{home} | home.show | App\Http\Controllers\HomeController@show | auth |
| | GET HEAD | home/{home}/edit | home.edit | App\Http\Controllers\HomeController@edit | auth |
| | PUT | home/{home} | home.update | App\Http\Controllers\HomeController@update | auth |
| | PATCH | home/{home} | | App\Http\Controllers\HomeController@update | auth |
| | DELETE | home/{home} | home.destroy | App\Http\Controllers\HomeController@destroy | auth |
| | GET HEAD | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@register | guest |
| | POST | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postRegister | guest |
| | GET HEAD | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@login | guest |
| | POST | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postLogin | guest |
| | GET HEAD | auth/logout/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@logout | guest |
| | GET HEAD POST PUT PATCH DELETE | auth/_{missing} | | App\Http\Controllers\Auth\AuthController@missingMethod | guest |
| | GET HEAD | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth>PasswordController@email | guest |
| | POST | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth>PasswordController@postEmail | guest |
| | GET HEAD | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth>PasswordController@getReset | guest |
| | POST | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth>PasswordController@postReset | guest |
| | GET HEAD POST PUT PATCH DELETE | password/_{missing} | | App\Http\Controllers\Auth>PasswordController@missingMethod | guest |
| | GET HEAD POST PUT PATCH DELETE | kontak | | Closure | |
| | POST | kontak-post | | Closure | |
| | GET HEAD | welcome/{nama?} | home.welcome | Closure | |

Mengecek Named Routes

Penggunaan Named Routes

Dalam menggenerate URL di Laravel, kita dapat menggunakan alamat URL biasa. Misalnya, untuk URL ke `welcome` bisa kita buat menjadi `/welcome` seperti ini:

contoh pembuatan URL manual

```
echo '<a href="/welcome">Kunjungi Halaman Welcome</a>';
```

Atau, kita juga bisa menggunakan nama dari route yang telah kita buat dengan syntax `route('nama-route')`. Misalnya:

contoh pembuatan URL dengan named route

```
echo 'Kunjungi <a href="'.route('home.welcome').'">Halaman Welcome</a>';
```

Manfaat Named Routes

Kelebihan dari named routes adalah jika alamat url berubah, maka kita tidak perlu merubah coding untuk meng-generate url ke route tersebut. Misalnya, kita buat halaman untuk yang berisi link untuk mengunjungi halaman welcome:

app/Http/routes.php

```
....  
Route::get('menu', function() {  
    return 'Kunjungi <a href="'.route('home.welcome').'">Halaman Welcome</a>';  
});
```

Jika kita mengunjungi url /menu, maka akan muncul link ke route welcome:

**Generate Route dengan named routes**

Jika kita ubah URL ke route welcome, misalnya menjadi selamat-datang:

app/Http/routes.php

```
....  
Route::get('selamat-datang/{nama?}', ['as'=>'home.welcome', function($nama = 'Penungjung') {  
    return "Selamat datang " . $nama . ". Anda luar biasa!";  
}]);
```

Menggunakan cara manual, kita harus merubah syntax yang meng-generate link ini. Tapi, dengan named routes kita tidak perlu melakukannya. Mari kita coba:



URL otomatis berubah dengan named routes

Disinilah letak kelebihan penggunaan named routes. Saran saya, selalu gunakan named routes untuk route yang dibuat. Sip.

Memahami Route Groups

Dalam membangun route, Laravel mengizinkan kita untuk mengelompokkannya dengan `group()`. Dengan menggunakan route group, kita dapat melakukan beberapa hal ini dalam satu baris:

- Route Prefix
- Penambahan Middleware
- Penambahan Namespace Route
- Membuat Sub-Domain Routing

Mari kita bahas route prefix, untuk fungsi lainnya akan kita bahas pada bagian selanjutnya.

Sesuai namanya, route prefix akan mengizinkan kita untuk membuat URL dengan awalan. Misalnya, jika kita ingin membuat route `/dashboard/settings`, `/dashboard/profile` dan `/dashboard/inbox`, kita dapat mengelompokkannya dengan `prefix dashboard` seperti berikut:

app/Http/routes.php

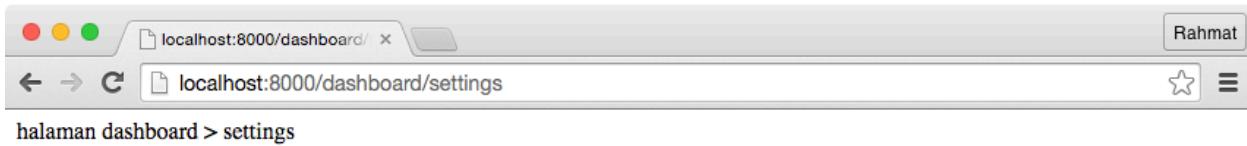
```
.....
Route::group(['prefix'=>'dashboard'], function() {
    Route::get('settings', function() {
        return 'halaman dashboard > settings';
    });
    Route::get('profile', function() {
        return 'halaman dashboard > profile';
    });
    Route::get('inbox', function() {
        return 'halaman dashboard > inbox';
    });
});
```

Jika, kita cek dengan `php artisan route:list` maka akan terlihat kita berhasil membuat URL tersebut:

| Domain Method | URI | Name | Action | Middleware |
|--------------------------------|-------------------------------------------------------|--------------|------------------------------------------------------------|------------|
| GET HEAD | / | | App\Http\Controllers\WelcomeController@index | guest |
| GET HEAD | home | home.index | App\Http\Controllers\HomeController@index | auth |
| GET HEAD | home/create | home.create | App\Http\Controllers\HomeController@create | auth |
| POST | home | home.store | App\Http\Controllers\HomeController@store | auth |
| GET HEAD | home/{home} | home.show | App\Http\Controllers\HomeController@show | auth |
| GET HEAD | home/{home}/edit | home.edit | App\Http\Controllers\HomeController@edit | auth |
| PUT | home/{home} | home.update | App\Http\Controllers\HomeController@update | auth |
| PATCH | home/{home} | home.destroy | App\Http\Controllers\HomeController@destroy | auth |
| DELETE | home/{home} | | App\Http\Controllers\Auth\AuthController@getLogout | guest |
| GET HEAD | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@register | guest |
| POST | auth/register/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postRegister | guest |
| GET HEAD | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@login | guest |
| POST | auth/login/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@postLogin | guest |
| GET HEAD | auth/logout/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\AuthController@getLogout | guest |
| GET HEAD POST PUT PATCH DELETE | auth/missing | | App\Http\Controllers\Auth\AuthController@missingMethod | guest |
| GET HEAD | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@getEmail | guest |
| POST | password/email/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postEmail | guest |
| GET HEAD | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@getReset | guest |
| POST | password/reset/{one?}/{two?}/{three?}/{four?}/{five?} | | App\Http\Controllers\Auth\PasswordController@postReset | guest |
| GET HEAD POST PUT PATCH DELETE | password/_missing | | App\Http\Controllers\Auth\PasswordController@missingMethod | guest |
| GET HEAD POST PUT PATCH DELETE | kontak | | Closure | |
| POST | kontak-post | | Closure | |
| GET HEAD | selamat-datang/{nama?} | home.welcome | Closure | |
| GET HEAD | menu | | Closure | |
| GET HEAD | dashboard/settings | | Closure | |
| GET HEAD | dashboard/profile | | Closure | |
| GET HEAD | dashboard/inbox | | Closure | |

Berhasil membuat route group

Kita pun dapat mengakses URL tersebut:



Halaman Dashboard > Settings



Halaman Dashboard > Profile



Halaman Dashboard > Inbox

Subdomain Routing

Jika kita menggunakan aplikasi misalnya Slack, biasanya setelah signup kita memiliki domain sendiri misalnya `malescast.slack.com`, `bukularavel.slack.com`, dsb. Di Lar-

avel, kita dapat membuat fitur dengan menggunakan Subdomain Routing. Syntax dasarnya seperti berikut:

Subdomain routing

```
Route::group(['domain' => '{account}.myapp.com'], function()
{
    Route::get('user/{id}', function($account, $id)
    {
        //
    });
});
```

Misalnya, kita hendak membuat website `fakebook.dev` dan bisa menerima subdomain dengan username.

- Jika kita mengakses `joni.fakebook.dev` maka akan menampilkan halaman akun Joni
- Jika kita mengakses `joni.fakebook.dev/profile` maka akan menampilkan profile Joni
- Jika kita mengakses `joni.fakebook.dev/status/1` maka akan menampilkan status Joni dengan id 1

Mari kita buat untuk case pertama, menampilkan halaman akun. Untuk membuatnya silahkan persiapkan aplikasi laravel di homestead dengan domain ke `fakebook.dev`. Kemudian, tambahkan route berikut:

t

```
Route::group(['domain' => '{username}.fakebook.dev'], function()
{
    Route::get('/', function($username) {
        return 'Anda mengunjungi akun ' . $username;
    });
});
```

Disini, kita menggunakan **username** sebagai nama dari subdomain. Pada route '/' (root) kita menampilkan teks informasi akun siapa yang sedang dikunjungi.

Karena di host lokal tidak diizinkan membuat wildcard subdomain, kita perlu menambah domain untuk tiap user secara manual. Mari kita buat untuk user `joni` dan `kiki`. Tambahkan baris berikut di file host (sesuaikan IP dengan IP homestead):

/etc/hosts atau C:\Windows\System32\Drivers\etc\hosts

```
1 192.168.10.10 kiki.fakebook.dev
2 192.168.10.10 joni.fakebook.dev
3 ....
```

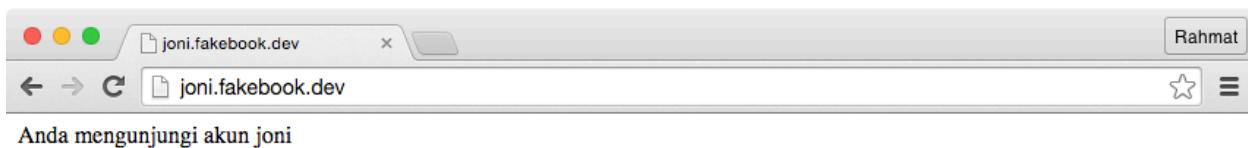


Wildcard Subdomin

Untuk Linux dan OSX, selain menggunakan cara manual, kita juga dapat menggunakan [dnsmasq](#)⁶⁸. Silahkan baca tutorial untuk [OSX](#)⁶⁹ atau [Ubuntu](#)⁷⁰.

Di server produksi, untuk membuat wildcard subdomain ini dengan cara membuat **A Record** baru. Isi **name** dengan *.domainkita.com, **address** dengan IP server dan **TTL** 1 hour. Tunggu 1 jam, wildcard subdomain kitapun akan berjalan.

Setelah siap, mari kita cek:

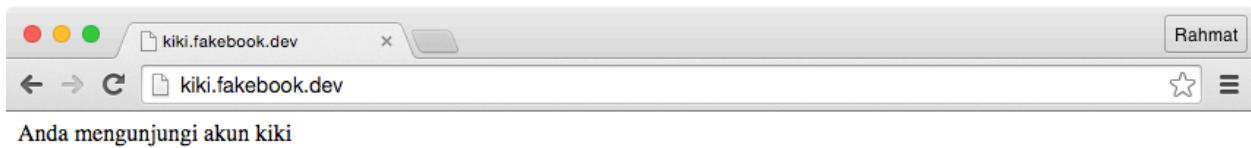


Mengunjungi Akun Joni

⁶⁸<http://www.thekelleys.org.uk/dnsmasq/doc.html>

⁶⁹<https://vinkla.com/posts/setup-wildcard-dns-on-mac-os-x/>

⁷⁰<https://help.ubuntu.com/community/Dnsmasq>



Mengunjungi Akun Kiki

Sip. Mari kita lanjut ke case kedua, menampilkan profile. Untuk memudahkan, mari kita simpan semua data user dalam array. Kemudian, kita passing array ini ke closure di route untuk menampilkan profile. Sehingga file route berubah menjadi:

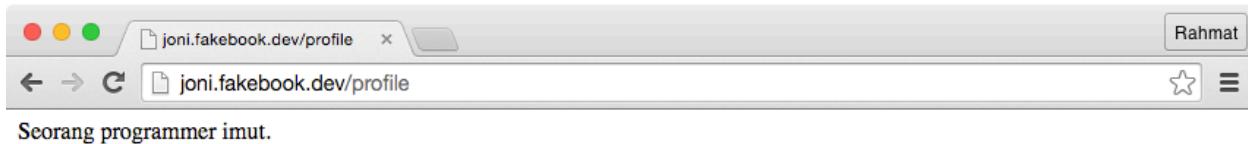
app/Http/routes.php

```

1 ....
2 Route::group(['domain' => '{username}.fakebook.dev'], function()
3 {
4     ....
5     $data_user = [
6         'joni' => [
7             'profile' => 'Seorang programmer imut.',
8             'status' => ['Gue keren!', 'Gue keren bgt!', 'Top dah!']
9         ],
10        'kiki' => [
11            'profile' => 'Seorang programmer cute.',
12            'status' => ['Mantap!', 'Hari ini luar biasa!', 'Cemungut ea..']
13        ]
14    ];
15
16    Route::get('profile', function($username) use ($data_user)
17    {
18        return $data_user[$username]['profile'];
19    });
20 });

```

Mari kita coba kunjungi profile Joni:



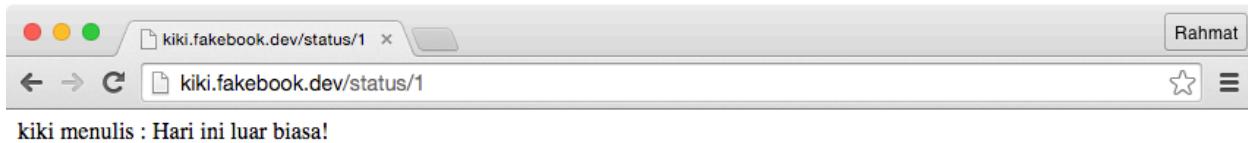
Mengunjungi Profile Joni

Untuk case ketiga, menampilkan status, kita juga akan menggunakan array yang telah dibuat tadi. Tambahkan route berikut:

app/Http/routes.php

```
1 ....
2 Route::group(['domain' => '{username}.fakebook.dev'], function()
3 {
4     ...
5     Route::get('status/{id}', function($username, $id) use ($data_user)
6     {
7         return $username . ' menulis : ' . $data_user[$username]['status'][$id];
8     });
9 });
```

Mari kita coba mengecek status Kiki:



Mengecek Status Kiki



Mengecek Status Kiki

Sip.

Whats next?

Agar lebih berurut, kita akan bahas beberapa materi tentang routing berikut pada bab yang berhubungan:

- Controller Routing dibahas di bab Controller
- RESTful Routing dibahas di bab Controller
- Route Namespace dibahas di bab Controller
- Mengakses route parameter dari luar route dibahas di bab Middleware
- Routing + Middleware dibahas di bab Middleware
- Route Model Binding dibahas di bab Model



Ringkasan

Pada bab ini kita telah mempelajari konsep dasar dan implementasi routing di Laravel. Sebagai framework PHP, penggunaan routing di Laravel pasti akan sangat memudahkan kita dalam membangun aplikasi yang kompleks. File route, selain digunakan sebagai tempat konfigurasi, juga dapat berfungsi sebagai dokumentasi fitur-fitur dari web yang kita bangun.

Pada bab selanjutnya, kita akan belajar teknik penggunaan Database di Laravel. Semangat!

Mengakses Database

Dalam sebuah aplikasi, database memegang peran penting dalam untuk menyimpan data. Konfigurasi database yang tepat dapat meningkatkan performa aplikasi secara signifikan. Meskipun data yang di handle banyak, dengan teknik yang tepat, aplikasi yang kita bangun akan tetap responsif.

Laravel memudahkan kita dalam mengakses database. Secara native, Laravel mendukung penggunaan MySQL, Postgres, SQLite, dan SQL Server. Di Laravel, penggunaan database terbagi menjadi tiga jenis yaitu menggunakan Raw Query, Query Builder dan Eloquent ORM.

Konfigurasi

Untuk mengakses database, kita perlu memberikan parameter konfigurasi di `config/database.php`. Berikut beberapa parameter yang perlu diperhatikan dengan nilai defaultnya.

```
'fetch' => PDO::FETCH_CLASS,
```

Akses database Laravel dibangun diatas PDO. Disini kita menentukan jenis return yang akan didapatkan oleh Laravel. Opsi default membuat setiap result menjadi sebuah object stdClass. Sehingga, jika kita punya table dengan kolom `nama`, `alamat` dan `usia` hasil query dapat diakses menjadi:

```
$user->nama;  
$user->alamat;  
$user->usia;
```

Jika diinginkan, kita bisa merubahnya misalnya menjadi array asosiatif dengan `PDO::FETCH_ASSOC`. Sehingga, untuk mengakses hasil query dengan table yang sama menjadi:

```
$user['nama'];  
$user['alamat'];  
$user['usia'];
```

Opsi default biasanya sudah cukup. Untuk mengetahui opsi lainnya, cek dokumentasi `PDO`⁷¹.

⁷¹ <http://php.net/manual/en/pdo.constants.php>

```
'default' => 'mysql',
```

Variabel ini menentukan database default yang akan kita gunakan. Isian mysql disini, merupakan nama koneksi. Bukan jenis (driver) koneksi. Isian koneksi ini berdasarkan konfigurasi di bawahnya yaitu:

```
'connections' => [  
  
    'sqlite' => [  
        'driver'    => 'sqlite',  
        'database'  => storage_path().'/database.sqlite',  
        'prefix'    => '',  
    ],  
  
    'mysql' => [  
        'driver'    => 'mysql',  
        'host'      => env('DB_HOST', 'localhost'),  
        'database'  => env('DB_DATABASE', 'forge'),  
        'username'  => env('DB_USERNAME', 'forge'),  
        'password'  => env('DB_PASSWORD', ''),  
        'charset'   => 'utf8',  
        'collation' => 'utf8_unicode_ci',  
        'prefix'    => '',  
        'strict'    => false,  
    ],  
  
    'pgsql' => [  
        'driver'    => 'pgsql',  
        'host'      => env('DB_HOST', 'localhost'),  
        'database'  => env('DB_DATABASE', 'forge'),  
        'username'  => env('DB_USERNAME', 'forge'),  
        'password'  => env('DB_PASSWORD', ''),  
        'charset'   => 'utf8',  
        'prefix'    => '',  
        'schema'    => 'public',  
    ],  
  
    'sqlsrv' => [  
        'driver'    => 'sqlsrv',  
        'host'      => env('DB_HOST', 'localhost'),  
        'database'  => env('DB_DATABASE', 'forge'),  
        'username'  => env('DB_USERNAME', 'forge'),
```

```
'password' => env('DB_PASSWORD', ''),
'prefix'   => '',
],
],
```

Terlihat disini, terdapat 4 koneksi berikut driver yang digunakan. Koneksi `mysql` menggunakan `driver` / engine database `mysql`. Tentunya, nama koneksi dapat kita ubah sesuai kebutuhan misalnya `mysql_serverA` dan mengubah isian `default` menjadi:

```
'default' => 'mysql_serverA',
```

Oke, untuk sekarang kita tidak akan merubahnya. Setidaknya, kita paham bahwa nama koneksi dapat dirubah sesuai kebutuhan, sedangkan jenis koneksi (`driver`) harus diisi sesuai dengan jenis koneksi yang didukung oleh Laravel yaitu `sqlite`, `mysql`, `pgsql` (untuk PostgreSQL) dan `sqlsrv` (untuk SQL Server).

Untuk membuat koneksi, Laravel membuat string DSN (Data Source Name) berdasarkan parameter konfigurasi yang kita buat. Berikut ini beberapa parameter yang bisa kita gunakan ketika membuat koneksi:

- `driver` : Menentukan driver database yang digunakan (`sqlite`, `mysql`, `pgsql`, `sqlsrv`).
- `host` : host/IP server lokasi database.
- `database` : Nama/lokasi database (untuk `sqlite`). Khusus untuk `sqlite`, selain diisi dengan lokasi database, kita juga dapat mengisi `:memory:` untuk menyimpan database secara temporary ke ram. Ini sangat berguna ketika kita melakukan testing.
- `username` : Username database.
- `password` : Password database.
- `charset` : Menentukan charset yang digunakan.
- `collation` : Menentukan collation yang digunakan.
- `prefix` : Menentukan prefix table. Ini bermanfaat jika kita akan menggunakan database yang sama untuk banyak aplikasi. Misalnya, jika diisi dengan `office` maka ketika mengakses menggunakan ORM Eloquent, table `user` akan diasumsikan table `office_user`.
- `strict` : Mengaktifkan *strict mode* di MySQL.
- `schema` : Menentukan *schema* di PostgreSQL.



Nilai Konfigurasi Default

Secara default Laravel menggunakan nilai dari file `.env` untuk beberapa parameter konfigurasi. Misalnya, untuk parameter nama database Laravel mengisi `env('DB_DATABASE', 'forge')`, yang artinya dia akan mencari variable `DB_DATABASE` dari file `.env` dan jika tidak ditemukan akan menggunakan `forge` sebagai isiannya.

Selain opsi diatas, ada beberapa opsi yang tidak tertulis yang bisa kita temukan dari beberapa file di [Illuminate/Database/Connectors](#)⁷²:

- `unix_socket` : Digunakan untuk menentukan lokasi file sock MySQL jika menggunakan lokasi yang tidak umum. Saya sendiri di Mac pernah menggunakan opsi ini untuk mengeset lokasi file sock ke `/Applications/MAMP/tmp/mysql/mysql.sock` (karena menggunakan MAMP).
- `port` : Digunakan untuk menentukan port server database.
- `sslmode` : Digunakan untuk menentukan model ssl untuk postgresql.
- `timezone` : Digunakan untuk menentukan timezone yang digunakan.

Karena Database Laravel dibangun diatas PDO, Laravel telah mengatur beberapa konfigurasi PDO-nya. Beberapa opsi PDO default yang diaktifkan Laravel dapat kita cek di [IlluminateDatabaseConnectorsConnector](#)⁷³ :

[Illuminate/Database/Connectors/Connector.php](#)

```
....  
protected $options = array(  
    PDO::ATTR_CASE => PDO::CASE_NATURAL,  
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,  
    PDO::ATTR_ORACLE_NULLS => PDO::NULL_NATURAL,  
    PDO::ATTR_STRINGIFY_FETCHES => false,  
    PDO::ATTR_EMULATE_PREPARES => false,  
);  
....
```

Untuk menambah/mereplace opsi PDO, kita dapat menggunakan parameter `options`. Misalnya, kita ingin mengaktifkan [persistent connections](#)⁷⁴ sebagaimana yang dilakukan oleh [CodeIgniter](#)⁷⁵ (secara default Laravel tidak menggunakananya). Bisa kita buat dengan menambah attribute `PDO::ATTR_PERSISTENT` dengan isi `true`. Maka, kita ubah koneksi MySQL menjadi:

⁷²<https://github.com/illuminate/database/tree/master/Connectors>

⁷³<https://github.com/illuminate/database/blob/master/Connectors/Connector.php#L12>

⁷⁴<http://php.net/manual/en/pdo.connections.php#example-980>

⁷⁵<https://github.com/EllisLab/CodeIgniter/blob/develop/application/config/database.php#L90>

```
'mysql' => [
    'driver'     => 'mysql',
    'host'        => env('DB_HOST', 'localhost'),
    'database'   => env('DB_DATABASE', 'forge'),
    'username'   => env('DB_USERNAME', 'forge'),
    'password'   => env('DB_PASSWORD', ''),
    'charset'    => 'utf8',
    'collation'  => 'utf8_unicode_ci',
    'prefix'     => '',
    'strict'     => false,
    'options'    => array(
        PDO::ATTR_PERSISTENT => true,
    ),
],
],
```

Sip. Silahkan cek beberapa attribute PDO yang bisa pakai di [dokumentasi resmi](#)⁷⁶.



Tidak semua aplikasi sesuai dengan Persistent Connections. Baca artikel ini⁷⁷ untuk mempelajari kondisi terbaik untuk menggunakannya.

Tes Koneksi

Mari kita coba melakukan koneksi database dengan homestead. Secara default isian parameter database di file .env (DB_HOST, DB_DATABASE, DB_USERNAME, DB_PASSWORD) akan sesuai dengan *credential* database MySQL dan PostgreSQL yang telah terinstall di homestead.

Untuk mengetesnya, silahkan jalankan homestead dan ssh ke homestead dengan homestead ssh. Kita akan masuk ke folder project, kemudian menggunakan tinker pada untuk menjalankan beberapa perintah pengecekan database.

```
vagrant@homestead:~$ cd Code/sample-database/
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::connection();
=> <Illuminate\Database\MySqlConnection #0000000028a1884c000000000db0eeee> {}
```

Perintah DB::connection() akan memberikan kita *instance* dari koneksi default jika koneksi berhasil dilakukan. Sebagaimana terlihat, kita menggunakan konfigurasi 'default' => 'mysql' sehingga kita mendapatkan *instance* dari Illuminate\Database\MySqlConnection.

⁷⁶ <http://php.net/manual/en/pdo.setattribute.php>

⁷⁷ <http://stackoverflow.com/questions/3332074/what-are-the-disadvantages-of-using-persistent-connection-in-pdo>

Hal ini juga menunjukkan *credential* database yang kita masukan sesuai. Jika kita salah memasukan *credential*, kita akan mendapatkan PDOException.

Misalnya kita ubah isian DB_DATABASE di .env menjadi

.env

DB_DATABASE=dbku

Maka, ini output yang kita dapatkan

```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::connection();
PDOException with message 'SQLSTATE[HY000] [1049] Unknown database 'dbku''
```

Oke, jika sudah paham, silahkan kembalikan lagi nilai DB_DATABASE ke homestead.

.env

DB_DATABASE=homestead

Selain mengecek koneksi default, kita juga dapat mengecek koneksi lain yang telah kita buat dengan menambahkan parameter nama koneksi pada perintah DB::connection(). Misalnya, mari kita cek koneksi ke koneksi sqlite yang menggunakan driver sqlite dan database di storage_path().'/database.sqlite' atau di storage/database.sqlite.

```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::connection('sqlite');
InvalidArgumentException with message 'Database does not exist.'
```

Oopppsss... Tentu saja kita mendapat error, karena file storage/database.sqlite belum ada. Mari kita buat file tersebut, misalnya dengan perintah touch berikut dari dalam folder project

```
vagrant@homestead:~/Code/sample-database$ touch storage/database.sqlite
```

Kini, jika kita tes kembali koneksi sqlite maka akan mendapatkan *instance* dari Illuminate\Database\SQLiteConnection

```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::connection('sqlite');
=> <Illuminate\Database\SQLiteConnection #0000000489c5d3e0000000034360de7> {}
```

Setelah kita mendapatkan connection, kita dapat mengecek credential database yang kita gunakan dengan method `getConfig()`.

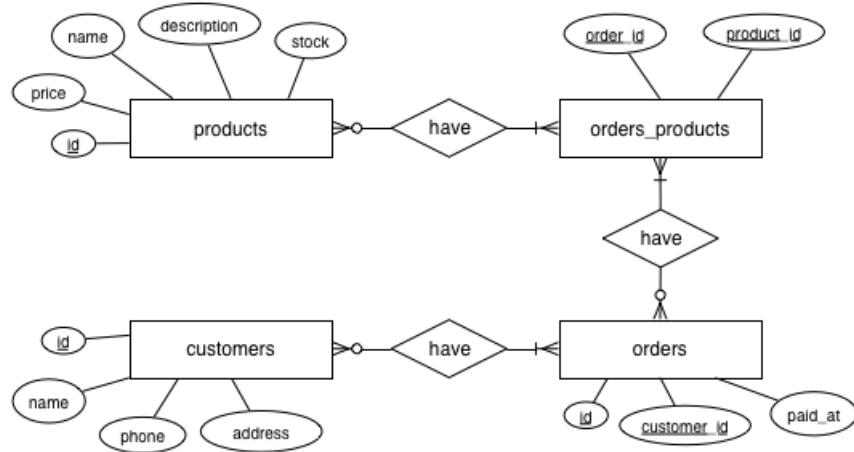
```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::connection();
=> <Illuminate\Database\MySQLConnection #000000005ef8df3d000000000545e3e1> {}
>>> DB::connection()->getConfig('driver');
=> "mysql"
>>> DB::connection()->getConfig('database');
=> "homestead"
>>> DB::connection()->getConfig('username');
=> "homestead"
>>> DB::connection()->getConfig('password');
=> "secret"
```

Database Migration & Schema Builder

Di Laravel, dalam membuat database kita dapat membuatnya manual atau dengan migration. Fitur ini terinspirasi dari migration di Ruby On Rails. Dengan menggunakan migration, proses pembuatan struktur database dapat kita buat dalam script PHP. Ada beberapa keuntungan dengan menggunakan migration:

- Perubahan struktur database dapat tercatat di source control (SVN, GIT, dll).
- Kita tidak perlu berkirim file SQL untuk tiap anggota member setiap kali melakukan perubahan struktur database. Member yang lain cukup mengambil source code terbaru dan menjalankan migration.
- Ketika mengembangkan aplikasi, kita dapat melakukan undo dan redo struktur database dengan mudah.
- Jika digabungkan dengan seeding, kita dapat mereset isi database dengan mudah.

Dalam beberapa contoh ke selanjutnya kita akan membuat beberapa table untuk database barang dan pemesanan di sebuah toko online, dengan hasil akhir sesuai ERD berikut:



ERD Barang dan Pemesanan

ERD ini akan kita bangun secara bertahap, sambil kita mempelajari fitur-fitur yang ada di Laravel untuk berinteraksi dengan database.

Migrate

Untuk membuat table baru dengan migration, kita harus membuat file migration terlebih dahulu dengan perintah `php artisan make:migration nama_migration`. Mari kita buat migration untuk membuat table Products:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration create_products_table
Created Migration: 2015_04_09_010128_create_products_table
```

Akan terbuat file baru di `database/migrations/2015_04_09_010128_create_products_table.php` dengan isi:

database/migrations/2015_04_09_010128_create_products_table.php

```
<?php

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

class CreateProductsTable extends Migration {

    /**
     * Run the migrations.
     *

```

```
* @return void
*/
public function up()
{
    //
}

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

}
```

Penomoran file migration akan dibuat secara acak, nama file yang terbuat setiap kali kita menjalankan perintah ini akan berbeda. Pada tiap file migration akan terdapat 2 method:

- `up`: method ini harus diisi dengan perubahan struktur database yang kita inginkan.
- `down`: method ini harus diisi dengan perintah untuk membatalkan perubahan di method `up`.

Untuk melakukan perubahan struktur database kita dapat menggunakan [Schema Builder](#)⁷⁸.

Pada tahap ini kita akan membuat table `Products` dengan field `id`, `name` dan `description`. Untuk melakukannya kita akan memakai [Schema Builder](#)⁷⁹ method `Schema::create()`. Isi method `up` dengan:

⁷⁸<http://laravel.com/docs/5.0/schema>

⁷⁹<http://laravel.com/docs/5.0/schema>

database/migrations/2015_04_09_010128_create_products_table.php

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

Method `Schema::create()` menerima parameter berupa nama table yang akan kita buat dan parameter kedua berupa instance dari `Blueprint`⁸⁰. Disini, kita menggunakan method :

- `increments()` untuk membuat `auto_increment` id dengan tipe `integer`.
- `string()` untuk membuat `varchar`.
- `nullable()` digunakan untuk mengizinkan field tidak diisi (null)

Ada sangat banyak method untuk membuat field yang didukung oleh Blueprint, cek di [dokumentasi resmi](#)⁸¹.

Karena pada method `up` kita membuat table baru, maka pada method `down` kita harus menghapus table tersebut menggunakan method `Schema::drop()` dengan parameter nama table yang akan kita hapus:

database/migrations/2015_04_09_010128_create_products_table.php

```
....  
public function up()  
{  
    Schema::drop('products');  
}  
....
```

⁸⁰ <http://laravel.com/api/5.0/Illuminate/Database/Schema/Blueprint.html>

⁸¹ <https://laravel.com/docs/5.2/migrations#creating-columns>

Secara default, Laravel memiliki migration untuk table `users` dan `password_resets`. Kedua table ini akan digunakan oleh Laravel untuk membuat fitur authentikasi bawaan. Untuk bab ini kita belum membutuhkan dua table tersebut. Hapuslah file `database/migrations/2014_10_12_000000_create_users_table.php` dan `database/migrations/2014_10_12_100000_create_password_resets_table.php`.

Untuk melakukan migration, jalankan perintah berikut:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate
Migration table created successfully.
Migrated: 2015_04_09_010128_create_products_table
```

Setelah berhasil menjalankan perintah diatas, di database akan terdapat table `products`:

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(255) | NO | | NULL | |
| description | varchar(255) | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

Karena kita pertama kali menjalankan migration, Laravel juga akan membuat table `migrations`. Ini terlihat dari output `Migration table created successfully..`

```
1 mysql> desc migrations;
2 +-----+-----+-----+-----+-----+
3 | Field | Type | Null | Key | Default | Extra |
4 +-----+-----+-----+-----+-----+
5 | migration | varchar(255) | NO | | NULL | |
6 | batch | int(11) | NO | | NULL | |
7 +-----+-----+-----+-----+-----+
```

Table `migrations` digunakan untuk mencatat file migrations apa saja yang telah dijalankan.

Mari kita buat migration baru untuk membuat table `customers`. Kali ini mari kita gunakan opsi `--create` agar Laravel menyediakan template untuk membuat table baru:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration --create="c\\
ustomers" create_customers_table
Created Migration: 2015_04_09_143657_create_customers_table
```

Menggunakan opsi `--create`, berikut template file migration yang dihasilkan:

database/migrations/2015_04_09_143657_create_customers_table.php

```
<?php

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

class CreateCustomersTable extends Migration {

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

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

Terlihat disini, Laravel sudah menyediakan isian pada method `up` dan `down` dengan table yang kita inginkan. Secara default Laravel juga menambahkan `$table->increments('id')` untuk membuat field `id` & `$table->timestamps()` untuk membuat field

created_at dan updated_at yang akan bermanfaat ketika kita sudah belajar tentang Eloquent ORM. Mari ubah agar menjadi field id, name, phone dan address pada method up:

database/migrations/2015_04_09_143657_create_customers_table.php

```
....  
public function up()  
{  
    Schema::create('customers', function(Blueprint $table)  
    {  
        $table->increments('id');  
        $table->string('name');  
        $table->string('phone')->nullable();  
        $table->string('address')->nullable();  
    });  
}  
....
```

Mari kita jalankan kembali perintah `migrate` agar Laravel membuat table ini:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate  
Migrated: 2015_04_09_143657_create_customers_table
```

Kini, di database akan terdapat table `customers`:

```
mysql> desc customers;  
+-----+-----+-----+-----+-----+  
| Field | Type           | Null | Key | Default | Extra          |  
+-----+-----+-----+-----+-----+  
| id   | int(10) unsigned | NO   | PRI | NULL    | auto_increment |  
| name | varchar(255)    | NO   |     | NULL    |                |  
| phone | varchar(255)    | YES  |     | NULL    |                |  
| address | varchar(255)    | YES  |     | NULL    |                |  
+-----+-----+-----+-----+-----+
```

Selain membuat satu table, dalam satu migration kita juga bisa membuat beberapa table. Mari kita buat table `orders` dan `orders_products` dalam satu migration:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration create_orders_tables
Created Migration: 2015_04_09_233100_create_orders_tables
```

Buat dua table tersebut di method `up`:

database/migrations/2015_04_09_233100_create_orders_tables.php

```
....  
public function up()  
{  
    Schema::create('orders', function(Blueprint $table)  
    {  
        $table->increments('id');  
        $table->integer('customer_id');  
        $table->date('paid_at');  
    });  
  
    Schema::create('orders_products', function(Blueprint $table)  
    {  
        $table->integer('order_id');  
        $table->integer('product_id');  
    });  
}  
....
```

Kemudian kita hapus dua table tersebut di method `down`:

database/migrations/2015_04_09_233100_create_orders_tables.php

```
....  
public function down()  
{  
    Schema::drop('orders');  
    Schema::drop('orders_products');  
}  
....
```

Kemudian kita jalankan migration:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate
Migrated: 2015_04_09_233100_create_orders_tables
```

Maka akan terbuat dua table dengan satu migration:

```
mysql> desc orders;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| customer_id | int(11) | NO | | NULL | |
| paid_at | date | NO | | NULL | |
+-----+-----+-----+-----+-----+
mysql> desc orders_products;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| order_id | int(11) | NO | | NULL | |
| product_id | int(11) | NO | | NULL | |
+-----+-----+-----+-----+-----+
```

Sip. Kita juga dapat menggunakan migration untuk merubah struktur table. Untuk itu, kita harus menggunakan `Schema::table()` pada `Schema Builder`. Pada contoh diatas, kita belum menambah relationship di table `orders` dan `orders_products`. Untuk membuat migration ini, selain menggunakan perintah migration biasa, kita juga dapat menggunakan opsi `--table`:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration --table="orders" add_relationships_to_orders_table
Created Migration: 2015_04_09_235458_add_relationships_to_orders_table
```

Dengan perintah ini, akan terbuat file migration dengan template:

database/migrations/2015_04_09_235458_add_relationships_to_orders_table.php

```
<?php

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

class AddRelationshipsToOrdersTable extends Migration {

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('orders', function(Blueprint $table)
        {
            //
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('orders', function(Blueprint $table)
        {
            //
        });
    }
}
```

Mari kita tambahkan foreign key dari table `orders` ke table `customers` pada method `up`:

database/migrations/2015_04_09_235458_add_relationships_to_orders_table.php

```
....  
public function up()  
{  
    Schema::table('orders', function(Blueprint $table)  
    {  
        $table->integer('customer_id')->unsigned()->change();  
        $table->foreign('customer_id')->references('id')->on('customers')  
            ->onUpdate('cascade')->onDelete('cascade');  
    });  
}  
....
```

Untuk membuat field `customer_id` menjadi `foreign key`, kita harus merubahnya menjadi `unsigned`, oleh karena itu kita memakai syntax:

```
$table->integer('customer_id')->unsigned()->change();
```

Agar kita dapat menjalankan method `change()`, kita perlu menambah `doctrine/dbal` dengan `composer`:

```
vagrant@homestead:~/Code/sample-database$ composer require doctrine/dbal  
Using version ~2.5 for doctrine/dbal  
.composer.json has been updated  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
- Installing doctrine/lexer (v1.0.1)  
  Downloading: 100%  
  
- Installing doctrine/annotations (v1.2.4)  
  Downloading: 100%  
  
- Installing doctrine/collections (v1.3.0)  
  Downloading: 100%  
  
- Installing doctrine/cache (v1.4.1)  
  Downloading: 100%  
  
- Installing doctrine/common (v2.5.0)  
  Downloading: 100%
```

```
- Installing doctrine/dbal (v2.5.1)
  Downloading: 100%
```

```
Writing lock file
Generating autoload files
Generating optimized class loader
```

Pada baris selanjutnya, kita menambahkan `foreign key` pada field `customer_id`:

```
$table->foreign('customer_id')->references('id')->on('customers')
  ->onUpdate('cascade')->onDelete('cascade');
```

Pada baris ini ada beberapa method yang kita gunakan:

- `foreign()`: Menentukan field yang akan menjadi `foreign key`. Method ini harus dipanggil pertama kali.
- `references()`: Menentukan field di table relasi yang akan menjadi acuan.
- `on()`: Menentukan table yang akan menjadi relasi.
- `onUpdate()`: Menentukan aksi yang akan dilakukan ketika data di table index di rubah. Kita dapat mengisinya sesuai kebutuhan, misalnya `cascade`, `set null`, `restrict`, dll.
- `onDelete()`: Menentukan aksi yang akan dilakukan ketika data di table index dihapus. Kita dapat mengisinya sesuai kebutuhan, misalnya `cascade`, `set null`, `restrict`, dll.

Pada method `down()`, kita menghapus `foreign key` yang telah dibuat dan merubah kembali field `customer_id`:

```
public function down()
{
    Schema::table('orders', function(Blueprint $table)
    {
        $table->dropForeign('orders_customer_id_foreign');
    });

    Schema::table('orders', function(Blueprint $table)
    {
        $table->dropIndex('orders_customer_id_foreign');
    });
}
```

```
Schema::table('orders', function(Blueprint $table)
{
    $table->integer('customer_id')->change();
});
```

Method `dropForeign()` menerima input berupa nama `foreign key` yang akan kita hapus. Laravel menggunakan format `table_field_foreign` untuk menamai `foreign key`, maka pada field diatas nama field nya akan menjadi `orders_customer_id_foreign`.

Kita juga menggunakan `dropIndex()` untuk menghapus index baru yang telah dibuat otomatis oleh Laravel ketika kita membuat `foreign key`.

Kita harus memisahkan penghapusan `foreign key` dengan `unsigned` pada `customer_id` agar tidak terjadi bentrok.

Mari kita jalankan migrate:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate
Migrated: 2015_04_09_235458_add_relationships_to_orders_table
```

Pastikan `foreign key` yang kita buat telah aktif:

```
mysql> SHOW CREATE TABLE orders;
+-----+-----+
| Table | Create Table |
+-----+-----+
| orders | CREATE TABLE `orders` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `customer_id` int(11) NOT NULL,
    `paid_at` date NOT NULL,
    PRIMARY KEY (`id`),
    KEY `orders_customer_id_foreign` (`customer_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+-----+-----+
```

Mari kita tambah juga relasi untuk table `orders_products`:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration --table="orders_products" add_relationships_to_orders_products
Created Migration: 2015_04_09_236144_add_relationships_to_orders_products
```

Mari kita tambahkan relasi pada method `up`:

database/migrations/2015_04_09_236144_add_relationships_to_orders_products.php

```
public function up()
{
    Schema::table('orders_products', function(Blueprint $table)
    {
        $table->integer('product_id')->unsigned()->change();
        $table->foreign('product_id')->references('id')->on('products')
            ->onUpdate('cascade')->onDelete('restrict');

        $table->integer('order_id')->unsigned()->change();
        $table->foreign('order_id')->references('id')->on('orders')
            ->onUpdate('cascade')->onDelete('restrict');
    });
}
```

Kemudian kita hapus pada method down:

database/migrations/2015_04_09_236144_add_relationships_to_orders_products.php

```
public function down()
{
    Schema::table('orders_products', function(Blueprint $table)
    {
        $table->dropForeign('orders_products_product_id_foreign');
        $table->dropForeign('orders_products_order_id_foreign');
    });

    Schema::table('orders_products', function(Blueprint $table)
    {
        $table->dropIndex('orders_products_product_id_foreign');
        $table->dropIndex('orders_products_order_id_foreign');
    });

    Schema::table('orders_products', function(Blueprint $table)
    {
        $table->integer('product_id')->change();
        $table->integer('order_id')->change();
    });
}
```

Kita jalankan kembali migrationnya:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate
Migrated: 2015_04_09_236144_add_relationships_to_orders_products
```

Pastikan relasi table berhasil dibuat:

```
mysql> SHOW CREATE TABLE orders_products;
+-----+-----+
| Table | Create Table |
+-----+-----+
| orders_products | CREATE TABLE `orders_products` (
  `order_id` int(10) unsigned NOT NULL,
  `product_id` int(10) unsigned NOT NULL,
  KEY `orders_products_product_id_foreign` (`product_id`),
  KEY `orders_products_order_id_foreign` (`order_id`),
  CONSTRAINT `orders_products_order_id_foreign` FOREIGN KEY (`order_id`) REFERENCES `orders` (`id`) ON UPDATE CASCADE,
  CONSTRAINT `orders_products_product_id_foreign` FOREIGN KEY (`product_id`) REFERENCES `products` (`id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+-----+-----+
```

Rollback (undo sekali)

Kelebihan dari penggunaan migration adalah kita dapat melakukan undo pada struktur database. Proses undo ini dilakukan dengan mengecek isian di table migrations:

```
mysql> select * from migrations;
+-----+-----+
| migration | batch |
+-----+-----+
| 2015_04_09_010128_create_products_table | 1 |
| 2015_04_09_143657_create_customers_table | 2 |
| 2015_04_09_233100_create_orders_tables | 3 |
| 2015_04_09_235458_add_relationships_to_orders_table | 4 |
| 2015_04_09_236144_add_relationships_to_orders_products | 5 |
+-----+-----+
```

Proses undo dilakukan dengan perintah `php artisan migrate:rollback`:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:rollback
Rolled back: 2015_04_09_236144_add_relationships_to_orders_products
```

Perintah ini akan menjalankan method `down` pada file migration yang memiliki nilai batch terbesar. Jika terdapat dua file migration yang memiliki nilai batch yang sama, maka keduanya akan dijalankan sesuai nomor urutannya.

Dalam contoh diatas, method `down` pada file `database/migrations/2015_04_09_236144_-add_relationships_to_orders_products.php` yang akan menghapus relationship pada table `orders_products`.

```
mysql> SHOW CREATE TABLE orders_products;
+-----+-----+
| Table | Create Table |
+-----+-----+
| orders_products | CREATE TABLE `orders_products` (
  `order_id` int(11) NOT NULL,
  `product_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+-----+-----+
```

Jika kita menjalankan kembali `php artisan migrate:rollback` Laravel akan terus menjalankan method `down` pada file migration sesuai urutan batchnya.

Reset (undo all)

Selain menggunakan `migrate:rollback`, kita juga dapat menggunakan membatalkan semua migration yang dilakukan dengan `migrate:reset`.

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:reset
Rolled back: 2015_04_09_235458_add_relationships_to_orders_table
Rolled back: 2015_04_09_233100_create_orders_tables
Rolled back: 2015_04_09_143657_create_customers_table
Rolled back: 2015_04_09_010128_create_products_table
Nothing to rollback.
```

Refresh

Jika `migrate:reset` akan membatalkan semua migration, perintah `migrate:refresh` akan membatalkan semua migration (jika sudah ada) dan menjalankan kembali semua migration.

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:refresh
Nothing to rollback.
Migrated: 2015_04_09_010128_create_products_table
Migrated: 2015_04_09_143657_create_customers_table
Migrated: 2015_04_09_233100_create_orders_tables
Migrated: 2015_04_09_235458_add_relationships_to_orders_table
Migrated: 2015_04_09_236144_add_relationships_to_orders_products
```

Perintah ini akan sangat bermanfaat jika kita sering menambah/menghapus field pada table selama proses development. Saya sendiri selama mengembangkan aplikasi dengan Laravel, jika perubahannya hanya sekedar menambah field pada table biasanya saya langsung menambahnya pada file migration yang sudah ada. Tanpa menambah migration baru.

Misalnya, mari kita tambah field `price` dan `stock` langsung di file `database/migrations/2015_04_09_010128_create_products_table.php`:

database/migrations/2015_04_09_010128_create_products_table.php

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

Untuk mendapatkan perubahan ini, kita cukup melakukan `migrate:refresh`:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:refresh
Rolled back: 2015_04_09_236144_add_relationships_to_orders_products
Rolled back: 2015_04_09_235458_add_relationships_to_orders_table
Rolled back: 2015_04_09_233100_create_orders_tables
Rolled back: 2015_04_09_143657_create_customers_table
Rolled back: 2015_04_09_010128_create_products_table
Nothing to rollback.

Migrated: 2015_04_09_010128_create_products_table
Migrated: 2015_04_09_143657_create_customers_table
Migrated: 2015_04_09_233100_create_orders_tables
```

```
Migrated: 2015_04_09_235458_add_relationships_to_orders_table
Migrated: 2015_04_09_236144_add_relationships_to_orders_products
```

Pastikan kedua field tersebut telah muncul:

```
mysql> desc products;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id   | int(10) unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(255) | NO | | NULL | |
| description | varchar(255) | YES | | NULL | |
| price | int(10) unsigned | NO | | NULL | |
| stock | int(10) unsigned | NO | | NULL | |
+-----+-----+-----+-----+-----+
```

Sip.

Basic CRUD Query

Secara umum, proses yang kita lakukan dengan database adalah seputar CRUD (Create, Read, Update & Delete) data. Meskipun proses sesungguhnya dari sebuah aplikasi [tidak hanya sekedar CRUD⁸²](#). Di level dasar, tetap saja kita akan melakukan proses ini.

Karena Laravel dibangun diatas PDO, tentunya teknik [parameter binding⁸³](#) dapat dilakukan dalam ketika melakukan query. Teknik ini akan sangat bermanfaat untuk menghindari serangan [SQL Injection⁸⁴](#).

Ada dua jenis tipe parameter yang didukung, menggunakan placeholder binding (tanda tanya) atau named binding. **Dalam satu query, placeholder dan named binding tidak boleh digunakan secara bersamaan.** Mari kita buat sample crud untuk table products yang telah dibuat dengan tinker.

Insert

Untuk melakukan query `insert` dengan perintah `DB::insert()`.

⁸²<http://verraes.net/2013/04/crud-is-an-anti-pattern>

⁸³<http://php.net/manual/en/pdostatement.bindparam.php>

⁸⁴https://www.owasp.org/index.php/SQL_Injection

```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::insert('insert into products (name, description, price, stock) values (?\n, ?, ?, ?)', ["Brio Satya A", "Tipe Manual", 114900000, 11]);
=> true
```

Contoh diatas menggunakan placeholder binding. Kita juga bisa menggunakan named binding dengan perintah:

```
>>> DB::insert('insert into products (name, description, price, stock) values (:mobil, :penjelasan, :harga, :stok)', ["mobil"=>"Jazz RS", "penjelasan"=>"Tipe CVT", "harga"=>252000000, "stok"=>7]);
=> true
```

Select

Untuk melakukan query select dengan perintah DB::select().

```
>>> DB::select('select * from products');
=> [
    <stdClass #000000002bb439d200000000570861d8> {
        id: 1,
        name: "Brio Satya A",
        description: "Tipe Manual",
        price: 114900000,
        stock: 11
    },
    <stdClass #000000002bb439d300000000570861d8> {
        id: 2,
        name: "Jazz RS",
        description: "Tipe CVT",
        price: 252000000,
        stock: 7
    }
]
```

Sebagaimana terlihat, kita mendapatkan array berisi object stdClass dengan attribute field dari table products. Kita bisa mengaksesnya:

```
>>> $products = DB::select('select * from products');
>>> $products[0]->id;
=> 1
>>> $products[0]->name;
=> "Brio Satya A"
>>> $products[0]->description;
=> "Tipe Manual"
```

Tentunya, kita juga dapat menambahkan parameter lain ketika melakukan query `select` dengan parameter binding. Misalnya, menambahkan `where`:

```
>>> DB::select('select * from products where name like :name', [ 'name'=> '%brio%' \
]);
=> [
    <stdClass #000000002bb439d200000000570861d8> {
        id: 1,
        name: "Brio Satya A",
        description: "Tipe Manual",
        price: 114900000,
        stock: 11
    }
]
>>>
```

Tentunya, jika kita ingin melakukan query relationship juga dapat dilakukan dengan `DB::select()` ini.

Update

Untuk melakukan query `update` dengan perintah `DB::update()`.

```
>>> DB::update('update products set name = :name, price = :price where id = :id' \
, [ 'name'=> 'Brio Sport E', 'price'=>190000000, 'id'=>1]);
=> 1
```

Kita dapat mengecek hasilnya:

```
>>> DB::select('select * from products where id = 1');
=> [
    <stdClass #00000002bb4392d00000000570861d8> {
        id: 1,
        name: "Brio Sport E",
        description: "Tipe Manual",
        price: 19000000,
        stock: 11
    }
]
>>>
```

Delete

Untuk melakukan query `delete` dengan perintah `DB::delete()`.

```
>>> DB::delete('delete from products where id = :id', ['id'=>1]);
=> 1
```

Perintah ini akan mengembalikan total record yang terhapus.

General Statement

Untuk melakukan statement `DDL`⁸⁵, misalnya `create`, `alter`, `drop`, dll. Kita dapat menggunakan `DB::statement()`. Contohnya, kita hendak menghapus kolom `stock` di table `products`:

```
>>> DB::statement('alter table products drop stock');
=> true
```

Kita dapat mengecek kolom yang terbuat dengan method `Schema::getColumnListing()` pada Schema Builder:

⁸⁵en.wikipedia.org/wiki/Data_definition_language

```
>>> Schema::getColumnListing('products');
=> [
    "id",
    "name",
    "description",
    "price"
]
>>>
```

Tentunya, kita juga dapat menggunakan parameter binding di dalam `DB::statement()`.

Raw Query

Jika kita ingin menggunakan raw query dapat menggunakan `DB::raw()`. Misalnya, kita ingin menambahkan tanggal input di `description` (walaupun bukan ide bagus) dengan fungsi `concat()` dan `now()` di MySQL:

```
>>> DB::insert( DB::raw('insert into products (name, description, price) values \
("CR-V", CONCAT("I-VTEC Manual. Diinput tanggal ", NOW()), 395000000)');
=> true
```

Kemudian kita cek:

```
>>> DB::select('select * from products where name = :name', ['name'=>'CR-V']);
=> [
    <stdClass #000000002bb4393b00000000570861d8> {
        id: 4,
        name: "CR-V",
        description: "I-VTEC Manual. Diinput tanggal 2015-04-17 00:36:46",
        price: 395000000
    }
]
```

Tentunya, contoh diatas akan *vulnerable* terhadap *sql injection*. Makanya, kita juga tetap bisa menggunakan parameter binding di dalam `DB::raw()`.

Database Seeding

Selain mampu membuat struktur database dengan migration, Laravel juga mempermudah kita dalam membuat sample data dengan bantuan `Seeder`⁸⁶. Dengan

⁸⁶<http://laravel.com/docs/5.0/migrations#database-seeding>

menggunakan seeder, kita dapat membuat sample data yang teratur maupun acak. Mari kita pelajari.

Semua file Seeder disimpan di folder `database/seeds` dan harus meng-extends `Illuminate\Database\Seeder`. Untuk nama class bebas, tapi disarankan menggunakan format `NamaTableSeeder`. Misalnya, mari kita isi table `products` dengan sample data. Silahkan buat file `database/seeds/ProductsTableSeeder.php` dengan isi:

database/seeds/ProductsTableSeeder.php

```
<?php

use Illuminate\Database\Seeder;

class ProductsTableSeeder extends Seeder {
    public function run()
    {
        DB::insert('insert into products (name, price, stock) values (:name, :price, :stock)', [
            'name' => "Brio Sport E",
            'price' => 180000000,
            'stock' => 14
        ]);

        DB::insert('insert into products (name, price, stock) values (:name, :price, :stock)', [
            'name' => "Brio Satya E",
            'price' => 125900000,
            'stock' => 23
        ]);

        $this->command->info('Berhasil menambah 2 mobil!');
    }
}
```

Pada file migration ini hanya ada satu method yang harus diisi yaitu method `up`. Didalamnya, kita isi dengan berbagai syntax untuk mengisi sample data. Pada contoh diatas, kita membuat 2 product (mobil) dan menampilkan pesan “Berhasil menambah 2 mobil!”.

Sebelum menjalankan migration ini, mari kita reset struktur database dengan `php artisan migrate:refresh`. Untuk menjalankan migration, kita panggil dengan perintah `php artisan db:seed --class "nama class"`:

```
vagrant@homestead:~/Code/sample-database$ php artisan db:seed --class "ProductsTableSeeder"
Berhasil menambah 2 mobil!
```

Kini akan terdapat 2 record baru di table products:

```
mysql> select * from products;
+----+-----+-----+-----+
| id | name      | description | price     | stock   |
+----+-----+-----+-----+
| 1  | Brio Sport E | NULL        | 1800000000 | 14      |
| 2  | Brio Satya E | NULL        | 1259000000 | 23      |
+----+-----+-----+-----+
```

Tentunya, jika kita menjalankan perintah diatas lagi maka akan dibuat 2 record lagi, dan seterusnya.

Tentunya, jika terdapat banyak file seeder yang kita buat, memanggil *class*-nya satu-persatu akan cukup merepotkan. Maka secara default, jika kita tidak memberikan parameter `--class`, Laravel akan memanggil `DatabaseSeeder`. Pada class ini, kita bisa memanggil *class* yang lain dengan perintah `$this->call('nama class')`. Mari kita panggil `ProductsTableSeeder`:

database/seeds/DatabaseSeeder.php

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder {

    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Model::unguard();

        $this->call('ProductsTableSeeder');
    }
}
```

```
}
```

Secara default, Laravel akan memanggil `Model::unguard()` yang berfungsi untuk mengizinkan *mass assignment*. Tenang, ini akan kita pelajari di bab Model. Pada baris selanjutnya, kita memanggil class `ProductsTableSeeder` dengan method `$this->call('ProductsTableSeeder')`. Kini, kita dapat menjalankan seed tanpa memanggil class Seeder:

```
vagrant@homestead:~/Code/sample-database$ php artisan db:seed
Berhasil menambah 2 mobil!
Seeded: ProductsTableSeeder
```

Data Random

Dengan sedikit kreativitas, kita juga dapat membuat data random misalnya dengan seeding seperti berikut:

database/seeds/ProductsTableSeeder.php

```
....  
public function run()  
{  
    $products = ["Accord", "Civic", "City", "CR-V", "Jazz", "Freed", "Mobilio"];  
    $descriptions = ["Tipe manual", "Tipe Otomatis"];  
  
    foreach ($products as $product) {  
        DB::insert('insert into products (name, description, price, stock) value\\  
s (:name, :description, :price, :stock)', [  
            'name' => $product,  
            'description' => $descriptions[rand(0,1)],  
            'price' => rand(100,800) * 1000000,  
            'stock' => rand(10,40)  
        ]);  
    }  
  
    $this->command->info('Berhasil menambah mobil!');  
}  
....
```

Syntax ini akan membuat data product baru dengan deskripsi, harga dan stock yang random:

```
mysql> select * from products;
+----+-----+-----+-----+
| id | name | description | price | stock |
+----+-----+-----+-----+
| 1 | Accord | Tipe manual | 564000000 | 35 |
| 2 | Civic | Tipe Otomatis | 139000000 | 29 |
| 3 | City | Tipe Otomatis | 745000000 | 13 |
| 4 | CR-V | Tipe manual | 723000000 | 32 |
| 5 | Jazz | Tipe Otomatis | 233000000 | 16 |
| 6 | Freed | Tipe manual | 740000000 | 33 |
| 7 | Mobilio | Tipe manual | 465000000 | 20 |
+----+-----+-----+-----+
```

Faker

Selain menggunakan cara diatas, kita juga dapat memanfaatkan [Faker⁸⁷](#) untuk membuat data random. Beberapa hal yang bisa dibuat secara acak oleh Faker diantaranya nama, alamat, nomor telepon, kota, negara, dll. Silahkan cek di [dokumentasi resmi⁸⁸](#) untuk melihat field apa saja yang bisa di generate oleh Faker.

Untuk menggunakan Faker, kita harus mendownloadnya dengan composer:

```
vagrant@homestead:~/Code/sample-database$ composer require fzaninotto/faker
Using version ~1.4 for fzaninotto/faker
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing fzaninotto/faker (v1.4.0)
  Downloading: 100%
Writing lock file
Generating autoload files
Generating optimized class loader
```

Mari kita buat sample 10 data customer dengan Faker. Buatlah file database/seeds/-CustomersTableSeeder.php dengan isi:

⁸⁷<https://github.com/fzaninotto/Faker>

⁸⁸<https://github.com/fzaninotto/Faker>

database/seeds/CustomersTableSeeder.php

```
<?php
use Faker\Factory as Faker;
use Illuminate\Database\Seeder;

class CustomersTableSeeder extends Seeder {

    public function run()
    {
        $faker = Faker::create();
        foreach ( range(1,10) as $index ) {
            DB::insert('insert into customers (name, phone, address) values (:name, :phone, :address)', [
                'name' => $faker->name,
                'phone' => $faker->phoneNumber,
                'address' => $faker->address
            ]);
        }
        $this->command->info('Berhasil menambah customer!');
    }
}
```

Terlihat disini, untuk menggunakan Faker, kita mengimportnya terlebih dahulu dengan `use Faker\Factory as Faker;` kemudian membuat object Faker dengan `$faker = Faker::create();`. Selanjutnya kita membuat data random nama dengan `$faker->name`, nomor telepon dengan `$faker->phoneNumber` dan alamat dengan `$faker->address`.

Mari kita panggil class ini dari DatabaseSeeder:

database/seeds/DatabaseSeeder.php

```
.....
public function run()
{
    .....
    $this->call('CustomersTableSeeder');
}
.....
```

Kemudian kita jalankan migration:

```
vagrant@homestead:~/Code/sample-database$ php artisan db:seed
Berhasil menambah mobil!
Seeded: ProductsTableSeeder
Berhasil menambah customer!
Seeded: CustomersTableSeeder
```

Kini akan terdapat 10 data customer baru:

```
mysql> select * from customers;
+----+-----+-----+-----+
| id | name           | phone          | address        |
+----+-----+-----+-----+
| 1  | Winfield Kshlerin | 708-435-9436x744 | 8065 Trantow Key Apt. 077... |
| 2  | Virgie Klein    | +22(1)2213106977 | 4480 Kurtis Plain Madilyn... |
| 3  | Wendy Witting   | +23(6)1022406372 | 21 Arlie Flat Eichmannbor... |
| 4  | Mr. Morton Yost | 04728088278    | 48 Jean Islands New Scott... |
| 5  | Miss Marilie    | (158)573-6074x1643 | 026 Bogisich Valley Soled... |
| 6  | Kariane Walker   | (944)589-8058x8433 | 72 Alisa InletKeeganhaven... |
| 7  | Carolyne Gislason | 931.577.3749x53335 | 325 Carlotta Prairie Apt.... |
| 8  | Frederik Reilly  | 978.938.3412x9696 | 94 Pouros Prairie Suite 3... |
| 9  | Shanel Brown     | +03(3)9229912558 | 75 Alivia Camp Finnfurt, ... |
| 10 | Harrison Koelpin | 1-796-740-6257   | 83 Destiney Well West Bar... |
+----+-----+-----+-----+
```

Keren kan? :)

Dalam mengerjakan project, saya sendiri sangat sering menggunakan Seeding + Faker ini. Kombinasi dua tools ini sangat bermanfaat untuk membuat sample data pada aplikasi yang masih dikembangkan.



Class Seeder Not Found?

Terkadang file seeder yang telah dibuat tidak ditemukan dengan pesan error **[ReflectionException] Class XXX does not exist**. Solusinya, jalankan composer dump-autoload untuk me-regenerate file autoload composer.

Refresh + Seed

Masih ingat dengan perintah `migrate:refresh`? Seperti yang kita ketahui, perintah itu berfungsi untuk me-refresh struktur database sesuai migration yang kita tulis. Kabar baiknya, kita dapat menggabungkan `refresh` dengan `--seed`. Manfaatnya, selain mengembalikan struktur database, Laravel juga akan meng-generate sample data.

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:refresh --seed
Rolled back: 2015_04_09_236144_add_relationships_to_orders_products
Rolled back: 2015_04_09_235458_add_relationships_to_orders_table
Rolled back: 2015_04_09_233100_create_orders_tables
Rolled back: 2015_04_09_143657_create_customers_table
Rolled back: 2015_04_09_010128_create_products_table
Nothing to rollback.

Migrated: 2015_04_09_010128_create_products_table
Migrated: 2015_04_09_143657_create_customers_table
Migrated: 2015_04_09_233100_create_orders_tables
Migrated: 2015_04_09_235458_add_relationships_to_orders_table
Migrated: 2015_04_09_236144_add_relationships_to_orders_products
Berhasil menambah mobil!
Seeded: ProductsTableSeeder
Berhasil menambah customer!
Seeded: CustomersTableSeeder
```

Query Builder

Selain menggunakan basic query, kita juga dapat menggunakan query builder untuk mengakses database. Dengan menggunakan Query Builder, output yang dihasilkan bukan berupa array object stdClass, tapi berupa object [Illuminate\Database\Query\Builder](#)⁸⁹. Tentunya, banyak method menarik yang bisa kita pakai pada object tersebut, yang akan kita pelajari di bagian ini.

Untuk menginisialisasi Query Builder, kita gunakan facade `DB::table()` dengan parameter berupa nama table yang ada di database. Setelah kita mendapatkan object dari `Illuminate\Database\Query\Builder` kita dapat melakukan banyak query yang kompleks dengan syntax yang cukup sederhana.

Select

Berikut ini beberapa contoh penggunaan query select dengan Query Builder.

Mengambil semua baris

⁸⁹ <http://laravel.com/api/5.0/Illuminate/Database/Query/Builder.html>

```
vagrant@homestead:~/Code/sample-database$ php artisan tinker
Psy Shell v0.4.1 (PHP 5.6.6-1+deb.sury.org~utopic+1 - cli) by Justin Hileman
>>> DB::table('products')->get();
=> [
    <stdClass #000000005acea6e600000000521d741e> {
        id: 1,
        name: "Accord",
        description: "Tipe Otomatis",
        price: 459000000,
        stock: 33
    },
    <stdClass #000000005acea6e400000000521d741e> {
        id: 2,
        name: "Civic",
        description: "Tipe manual",
        price: 413000000,
        stock: 30
    },
    ....
]
```

Mengambil baris pertama dari hasil query

```
>>> DB::table('products')->first();
=> <stdClass #000000005acea6e700000000521d741e> {
    id: 1,
    name: "Accord",
    description: "Tipe Otomatis",
    price: 459000000,
    stock: 33
}
```

Jika hanya hendak mengambil record dari id tertentu, kita dapat menggunakan `find()`:

```
>>> DB::table('products')->find(5);
=> <stdClass #000000002c234963000000001c4bf9d4> {
    id: 5,
    name: "Jazz",
    description: "Tipe Otomatis",
    price: 754000000,
    stock: 23
}
```

Lists, Menampilkan isian 1 kolom tertentu

```
>>> DB::table('products')->lists('name');
=> [
    "Accord",
    "Civic",
    "City",
    "CR-V",
    "Jazz",
    "Freed",
    "Mobilio"
]
```

Perintah diatas, akan memberikan output berupa array 1 dimensi. Kita juga dapat menampilkan outputnya menjadi array asosiatif dengan key berupa field lain. Misalnya, kita jadikan key nya berupa Id dari record:

```
>>> DB::table('products')->lists('name', 'id');
=> [
    1 => "Accord",
    2 => "Civic",
    3 => "City",
    4 => "CR-V",
    5 => "Jazz",
    6 => "Freed",
    7 => "Mobilio"
]
```

Fitur ini akan berguna ketika menggenerate data untuk Dropdown list.

Pluck, Menampilkan hanya 1 kolom dari baris pertama

Jika kita ingin menampilkan isian pertama dari kolom tertentu, tentunya bisa saja menggunakan `lists()` dan menambahkan array pertama seperti berikut:

```
>>> DB::table('products')->lists('name')[0];
=> "Accord"
```

Cara ini memang valid. Tapi, ada cara yang lebih elegan yaitu menggunakan `pluck()`

```
>>> DB::table('products')->pluck('name');
=> "Accord"
```

Menampilkan isian beberapa kolom

Kita dapat menggunakan `select()` untuk memilih kolom yang akan ditampilkan dari record:

```
>>> DB::table('products')->select('name', 'stock')->get();
=> [
    <stdClass #000000005acea6e800000000521d741e> {
        name: "Accord",
        stock: 33
    },
    <stdClass #000000005acea6eb00000000521d741e> {
        name: "Civic",
        stock: 30
    },
    ...
]
```

Adakah kita ingin menambah kolom yang akan ditampilkan berdasarkan kondisi tertentu. Untuk itu, kita dapat menggunakan `addSelect()`. Misalnya, diawal kita hanya menampilkan kolom `name` dan `stock`:

```
>>> $products = DB::table('products')->select('name', 'stock');
=> <Illuminate\Database\Query\Builder #000000005acea6e300000000521d741e> {
    aggregate: null,
    columns: [
        "name",
        "stock"
    ],
    distinct: false,
    from: "products",
    ...
}
```

Jika kita ingin menambah kolom yang akan ditampilkan misalnya kolom `price`, syntaxnya akan seperti berikut:

```
>>> $products->addSelect('price')->get();
=> [
    <stdClass #000000005acea6e200000000521d741e> {
        name: "Accord",
        stock: 33,
        price: 459000000
    },
    <stdClass #000000005acea6ef00000000521d741e> {
        name: "Civic",
        stock: 30,
        price: 413000000
    },
    ....
]
```

Chunk, memproses banyak data dengan lebih efisien memory

Menggunakan chunk sangat bermanfaat ketika kita akan memproses banyak baris dalam database. Penggunaan akan meminimalkan penggunaan memory dalam eksekusi script yang kita buat. Untuk menunjukkan manfaatnya, mari kita buat 10.000 data pada table `products` dengan membuat seeder `SampleChunkSeeder` dengan isi:

database/seeds/SampleChunkSeeder.php

```
use Faker\Factory as Faker;
use Illuminate\Database\Seeder;

class SampleChunkSeeder extends Seeder {

    public function run()
    {
        $faker = Faker::create();
        $products = ["Accord", "Civic", "City", "CR-V", "Jazz", "Freed", "Mobilio"];
        $descriptions = ["Tipe manual", "Tipe Otomatis"];

        for ( $i=0; $i < 10000; $i++ ) {
            DB::insert('insert into products (name, description, price, stock) values (:name, :description, :price, :stock)', [
                'name' => $products[rand(0,6)] . ' ' . $faker->firstNameMale,
                'description' => $descriptions[rand(0,1)],
                'price' => rand(100,800) * 1000000,
                'stock' => rand(2,40)
            ]);
        }
    }
}
```

```
        ]);
    }

    $this->command->info('Berhasil 10.000 menambah mobil!');
}
}
```

Setelah dibuat, mari kita jalankan seeder ini hingga muncul tulisan **Berhasil menambah 10.000 mobil!**.

```
vagrant@homestead:~/Code/sample-database$ php artisan db:seed --class "SampleChu\
nkSeeder"
Berhasil menambah 10.000 mobil!
```

Untuk mengetes penggunaan chunk, kita akan membuat route untuk menampilkan product yang memiliki stock lebih dari 20. Kita akan menggunakan 2 route, pertama tanpa chunk dan kedua dengan chunk, kemudian kita bandingkan penggunaan memory keduanya dengan fungsi `memory_get_usage()` di PHP. Mari kita buat route pertama:

app/Http/routes.php

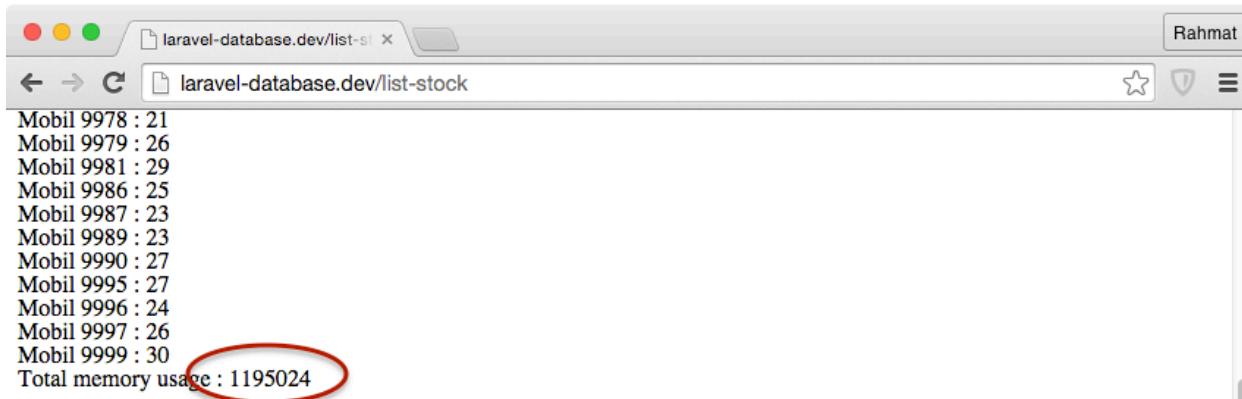
```
.....
Route::get('/list-stock', function() {
    $begin = memory_get_usage();
    foreach (DB::table('products')->get() as $product) {
        if ( $product->stock > 20 ) {
            echo $product->name . ' : ' . $product->stock . '<br>';
        }
    }
    echo 'Total memory usage : ' . (memory_get_usage() - $begin);
});
```

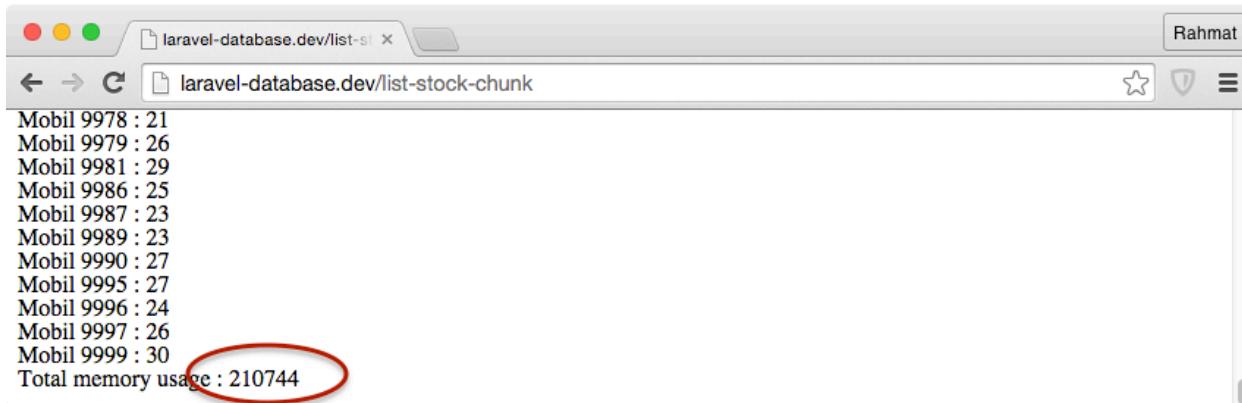
Kita buat juga route kedua dengan chunk:

app/Http/routes.php

```
....  
Route::get('/list-stock-chunk', function() {  
    $begin = memory_get_usage();  
    DB::table('products')->chunk(100, function($products)  
    {  
        foreach ($products as $product)  
        {  
            if ( $product->stock > 20 ) {  
                echo $product->name . ' : ' . $product->stock . '<br>';  
            }  
        }  
    });  
    echo 'Total memory usage : ' . (memory_get_usage() - $begin);  
});
```

Kini, kita dapat membandingkan penggunaan memory keduanya dengan mengunjungi /list-stock dan /list-stock-chunk.

**Tanpa Chunk**



Mobil 9978 : 21
Mobil 9979 : 26
Mobil 9981 : 29
Mobil 9986 : 25
Mobil 9987 : 23
Mobil 9989 : 23
Mobil 9990 : 27
Mobil 9995 : 27
Mobil 9996 : 24
Mobil 9997 : 26
Mobil 9999 : 30
Total memory usage : 210744

Dengan Chunk

Terlihat disini, tanpa menggunakan chunk memory yang digunakan sebanyak 1195024 byte atau hampir 1 Mb. Sementara, menggunakan chunk memory yang digunakan sebanyak 210744 byte atau sekitar 200 Kb. **Wow!**.

Penggunaan chunk ini sangat disarankan ketika kita berinteraksi dengan banyak data. Tentunya, jika kita menggunakan lebih banyak data, akan sangat terlihat penghematan memory yang kita lakukan.

Untuk memudahkan latihan selanjutnya, silahkan jalankan kembali `php artisan migrate:refresh --seed` untuk me-reset kondisi database.

Insert

Untuk menambah record baru ke table, kita gunakan method `insert()` dengan parameter berupa array asosiatif berisi field yang akan kita isi:

```
>>> DB::table('products')->insert(['name'=>'Fortuner', 'price'=>532700000, 'stock'\
=>12]);  
=> true
```

Jika diinginkan kita juga dapat mengambil ID yang di-generate dengan method `insertGetId()`:

```
>>> DB::table('products')->insertGetId(['name'=>'Land Cruiser', 'price'=>19776000\
00, 'stock'=>20]);  
=> 9
```



Agar `insertGetId()` dapat berjalan, primary key harus diberi nama `id` dan dikonfigurasi auto-increment.

Jika ingin menambah beberapa record secara sekaligus, bisa dilakukan dengan memberikan beberapa array sebagai parameter:

```
>>> DB::table('products')->insert([['name'=>'Fortuner', 'price'=>532700000, 'stock'=>12], ['name'=>'Land Cruiser', 'price'=>1977600000, 'stock'=>9]]);  
=> true
```

Update

Untuk mengupdate record, kita dapat menggunakan `update()`:

```
>>> DB::table('products')->where('name', 'Fortuner')->update(['price'=>400000000, 'stock'=>5]);  
=> 2
```

Pada contoh diatas kita menggabungkan dengan `where()` untuk memfilter products dengan nama Fortuner.

Jika kita hanya meningkatkan / mengurangi nilai sebuah field (misalnya pada counter visitor website atau counter stock mobil yang baru/terjual) akan lebih elegant jika kita menggunakan `increment()` atau `decrement()`. Berikut kita buat contoh `increment()` saja, karena `decrement()` syntaxnya juga sama.

Misalnya kita ingin menambah stock semua mobil sebanyak 1:

```
>>> DB::table('products')->increment('stock');  
=> 11
```

Menambah stock semua mobil sebanyak 10:

```
>>> DB::table('products')->increment('stock', 10);  
=> 11
```

Jika kita ingin menambah stock pada product dengan id tertentu:

```
>>> DB::table('products')->where('id', 1)->increment('stock', 5);  
=> 1
```

Pada saat melakukan penambahan, kita juga dapat mengupdate field yang lain. Misalnya, kita ubah kolom `description`:

```
>>> DB::table('products')->where('id',1)->increment('stock', 10, ['description'=\n    'Ditambah 10']);\n=> 1\n>>> DB::table('products')->find(1);\n=> <stdClass #000000002c234971000000001c4bf9d4> {\n    id: 1,\n    name: "Accord",\n    description: "Ditambah 10",\n    price: 672000000,\n    stock: 61\n}
```

Delete

Untuk menghapus record, kita dapat menggunakan `delete()`.

Menghapus record dengan id 5:

```
>>> DB::table('products')->delete(5);\n=> 1
```

Menghapus record berdasarkan hasil query:

```
>>> DB::table('products')->where('stock', '<', 20)->delete();\n=> 2
```

Menghapus isi table:

```
>>> DB::table('products')->delete();\n=> 8
```

Basic Where

Berikut beberapa contoh penggunaan `where` dalam Query Builder.

Where sederhana

```
>>> DB::table('products')->where('stock', '>', 25)->get();
=> [
    <stdClass #000000002c23496d000000001c4bf9d4> {
        id: 7,
        name: "Mobilio",
        description: "Tipe Otomatis",
        price: 249000000,
        stock: 39
    }
    ...
]
```

Contoh lain..

```
>>> DB::table('products')->where('name', 'Freed')->get();
=> [
    <stdClass #000000002c23496b000000001c4bf9d4> {
        id: 6,
        name: "Freed",
        description: "Tipe manual",
        price: 786000000,
        stock: 21
    }
]
```

Contoh where dengan like..

```
>>> DB::table('products')->where('name', 'like', '%cr%')->get();
=> [
    <stdClass #000000002c234978000000001c4bf9d4> {
        id: 4,
        name: "CR-V",
        description: "Tipe manual",
        price: 652000000,
        stock: 20
    }
]
```

Where Between

Gunakan `whereBetween()` untuk memilih data yang memiliki nilai dengan range tertentu. Misalnya, kita ingin memfilter products yang memiliki stock 10 - 20:

```
>>> DB::table('products')->whereBetween('stock', [10,20])->get();  
=> [  
    <stdClass #000000002c234970000000001c4bf9d4> {  
        id: 2,  
        name: "Civic",  
        description: "Tipe manual",  
        price: 768000000,  
        stock: 14  
    },  
    ....  
]
```

Kebalikan dari `whereBetween()`, kita dapat menggunakan `whereNotBetween()`:

```
>>> DB::table('products')->whereNotBetween('stock',[10,20])->get();  
=> [  
    <stdClass #000000002c234972000000001c4bf9d4> {  
        id: 1,  
        name: "Accord",  
        description: "Tipe manual",  
        price: 672000000,  
        stock: 35  
    },  
    ....  
]
```

Where In

`whereIn` dapat kita gunakan untuk memfilter kolom dengan array. Misalnya, kita ingin menampilkan record dengan id 4 dan 7:

```
>>> DB::table('products')->whereIn('id',[4,7])->get();  
=> [  
    <stdClass #000000002c23496c000000001c4bf9d4> {  
        id: 4,  
        name: "CR-V",  
        description: "Tipe manual",  
        price: 652000000,  
        stock: 20  
    },  
    <stdClass #000000002c234977000000001c4bf9d4> {  
        id: 7,
```

```
        name: "Mobilio",
        description: "Tipe Otomatis",
        price: 249000000,
        stock: 39
    }
]
```

Kebalikan dari `whereIn` adalah `whereNotIn`:

```
>>> DB::table('products')->whereNotIn('id',[4,7])->get();
=> [
    <stdClass #000000002c234986000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 672000000,
        stock: 35
    },
    <stdClass #000000002c23497f000000001c4bf9d4> {
        id: 2,
        name: "Civic",
        description: "Tipe manual",
        price: 768000000,
        stock: 14
    },
    ...
]
```

Where Null

Jika kita hendak memfilter record berdasarkan field yang null, dapat menggunakan `whereNull()`. Misalnya, mari kita hapus description pada product:

```
>>> DB::update('update products set description = null where id = 2');
=> 1
```

Untuk mendapatkan record ini, kita dapat menggunakan:

```
>>> DB::table('products')->whereNull('description')->get();
=> [
    <stdClass #000000002c234971000000001c4bf9d4> {
        id: 2,
        name: "Civic",
        description: null,
        price: 76800000,
        stock: 14
    }
]
```

Tentunya, kebalikan dari `whereNull()` adalah `whereNotNull()`:

```
>>> DB::table('products')->whereNotNull('description')->get();
=> [
    <stdClass #000000002c234987000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 67200000,
        stock: 35
    },
    ...
]
```

Or Where

Misalnya, kita menampilkan record yang memiliki nama `Freed` atau memiliki `stock > 30`:

```
>>> DB::table('products')->where('name', 'Freed')->orWhere('stock', '>', 30)->get();
=> [
    <stdClass #000000002c234959000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 67200000,
        stock: 35
    },
    <stdClass #000000002c234967000000001c4bf9d4> {
        id: 6,
        name: "Freed",
```

```
        description: "Tipe manual",
        price: 786000000,
        stock: 21
    },
    ...
]
```

Tentunya variasi dari `orWhere()` ini bisa dikombinasikan dengan `where` jenis lain yang telah kita pelajari. Misalnya, `orWhereBetween()`, `orWhereNull()`, dll.

And Where

Dalam membangun query, kita tidak dibatasi untuk menggunakan satu statement `where`. Untuk menggunakan beberapa `where` dengan query AND kita dapat menambahkan query selanjutnya secara langsung. Misalnya kita cari product dengan `price > 500,000,000` dan `stock > 10`:

```
>>> DB::table('products')->where('price', '>', 500000000)->where('stock', '>', 10)->\get();
=> [
    <stdClass #00000000726257dc000000002b67f510> {
        id: 2,
        name: "Civic",
        description: "Tipe Otomatis",
        price: 619000000,
        stock: 25
    },
    <stdClass #00000000726257db000000002b67f510> {
        id: 3,
        name: "City",
        description: "Tipe manual",
        price: 549000000,
        stock: 22
    },
    <stdClass #00000000726257d0000000002b67f510> {
        id: 6,
        name: "Freed",
        description: "Tipe Otomatis",
        price: 611000000,
        stock: 27
    }
]
```

Jika ingin menggunakan menggabungkan `where` dengan statement `AND` dan `OR` silahkan pelajari pada topik [parameter grouping](#).

Dynamic Where

Ini salah satu fitur yang saya suka. Kita dapat memanggil method `where` baru dengan nama field. Misalnya, untuk memfilter berdasarkan `name` atau `stock`:

```
>>> DB::table('products')->whereNameOrStock('Jazz', 20)->get();
=> [
    <stdClass #000000002c23496c000000001c4bf9d4> {
        id: 4,
        name: "CR-V",
        description: "Tipe manual",
        price: 652000000,
        stock: 20
    },
    <stdClass #000000002c23498e000000001c4bf9d4> {
        id: 5,
        name: "Jazz",
        description: "Tipe Otomatis",
        price: 754000000,
        stock: 12
    }
]
```

Kita juga dapat memfilter menggunakan misalnya `whereNameAndStock()`, `whereNameAndPrice()`, dst.

Order By

Untuk mengurutkan data kita dapat menggunakan `orderBy`:

```
>>> DB::table('products')->orderBy('stock', 'asc')->get();
=> [
    <stdClass #000000002c234983000000001c4bf9d4> {
        id: 5,
        name: "Jazz",
        description: "Tipe Otomatis",
        price: 754000000,
        stock: 12
    },
]
```

```
<stdClass #000000002c234972000000001c4bf9d4> {
    id: 2,
    name: "Civic",
    description: null,
    price: 768000000,
    stock: 14
},
.....
]
>>> DB::table('products')->orderBy('stock', 'desc')->get();
=> [
    <stdClass #000000002c234963000000001c4bf9d4> {
        id: 7,
        name: "Mobilio",
        description: "Tipe Otomatis",
        price: 249000000,
        stock: 39
    },
    <stdClass #000000002c234965000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 672000000,
        stock: 35
    },
    .....
]
```

Group By

Untuk memisahkan data berdasarkan isian yang sama, kita dapat menggunakan `groupBy()`. Misalnya, kita filter berdasarkan isian `description`:

```
>>> DB::table('products')->groupBy('description')->get();
=> [
    <stdClass #000000002c234958000000001c4bf9d4> {
        id: 2,
        name: "Civic",
        description: null,
        price: 768000000,
        stock: 14
},
```

```
<stdClass #000000002c23496b000000001c4bf9d4> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 67200000,
    stock: 35
},
<stdClass #000000002c234970000000001c4bf9d4> {
    id: 3,
    name: "City",
    description: "Tipe Otomatis",
    price: 61800000,
    stock: 22
}
]
```

Terlihat disini, meskipun banyak di table products hanya ditampilkan 3 record berdasarkan isian description yang berbeda.

Having

Selain menggunakan `where()`, untuk memfilter data berdasarkan nilai aggregate, kita juga dapat menggunakan `having`. Misalnya, kita filter data yang memiliki `price` lebih dari 500juta:

```
>>> DB::table('products')->having('price', '>', 50000000)->get();
=> [
    <stdClass #000000002c234986000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 67200000,
        stock: 35
    },
    <stdClass #000000002c23498e000000001c4bf9d4> {
        id: 2,
        name: "Civic",
        description: null,
        price: 76800000,
        stock: 14
    },
    ...
]
```

Offset & Limit

Jika kita hendak mengambil hanya beberapa record dari hasil query, kita dapat menggunakan `take()`. Misalnya, kita hanya ambil 2 record:

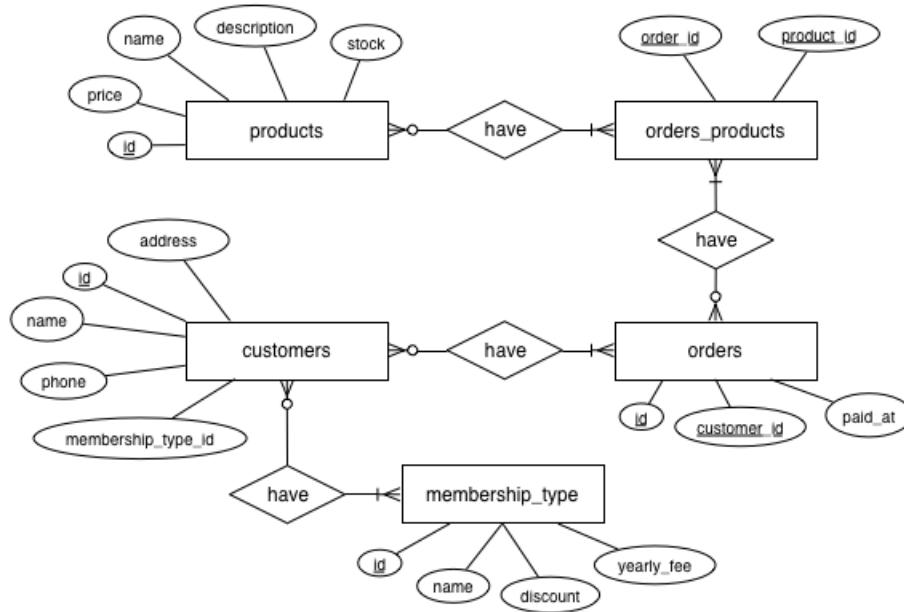
```
>>> DB::table('products')->take(2)->get();
=> [
    <stdClass #000000002c234960000000001c4bf9d4> {
        id: 1,
        name: "Accord",
        description: "Tipe manual",
        price: 672000000,
        stock: 35
    },
    <stdClass #000000002c234963000000001c4bf9d4> {
        id: 2,
        name: "Civic",
        description: null,
        price: 768000000,
        stock: 14
    }
]
```

`take()` dapat digabungkan dengan `skip()` untuk melewati beberapa hasil query. Contohnya, jika kita hendak me-skip 2 record pertama:

```
>>> DB::table('products')->take(2)->skip(2)->get();
=> [
    <stdClass #000000002c234967000000001c4bf9d4> {
        id: 3,
        name: "City",
        description: "Tipe Otomatis",
        price: 618000000,
        stock: 22
    },
    <stdClass #000000002c23497f000000001c4bf9d4> {
        id: 4,
        name: "CR-V",
        description: "Tipe manual",
        price: 652000000,
        stock: 20
    }
]
```

Join

Untuk mempelajari join, mari kita tambah table `membership_type` dan field `membership_type`. Table ini akan mencatat jenis membership yang bisa diperoleh oleh customer. Tentunya, customer hanya bisa memiliki satu jenis membership yang aktif atau tidak memiliki membership sama sekali. ERD nya akan berubah menjadi:



Menambah table `membership_type`

Mari kita buat migration untuk membuat table ini:

```
vagrant@homestead:~/Code/sample-database$ php artisan make:migration create_membership_types_table
Created Migration: 2015_04_09_236252_create_membership_types_table
```

Pada method `up` kita akan membuat table `membership_types` dan menambah relasi field `membership_type_id` di table `customers`. Karena beberapa customer tidak akan memiliki membership, kita tidak akan membuat relasi secara eksplisit di database:

database/migrations/2015_04_09_236252_create_membership_types_table.php

```
....  
public function up()  
{  
    Schema::create('membership_types', function(Blueprint $table)  
    {  
        $table->increments('id');  
        $table->string('type');  
        $table->integer('discount');  
        $table->integer('yearly_fee');  
    });  
  
    Schema::table('customers', function(Blueprint $table)  
    {  
        $table->integer('membership_type_id')->unsigned()->nullable();  
    });  
}  
....
```

Pada method `down`, kita akan menghapus relasi tadi dan menghapus table `membership_types`:

database/migrations/2015_04_09_236252_create_membership_types_table.php

```
....  
public function down()  
{  
    Schema::table('customers', function(Blueprint $table)  
    {  
        $table->dropColumn('membership_type_id');  
    });  
  
    Schema::drop('membership_types');  
}  
....
```

Mari kita buat sample data untuk table `memberships`. Buatlah file `database/seeds/MembershipTypesTableSeeder.php` dengan isi:

database/seeds/MembershipTypesTableSeeder.php

```
<?php  
use Illuminate\Database\Seeder;  
  
class MembershipTypesTableSeeder extends Seeder {  
  
    public function run()  
    {  
        $types = ["Silver", "Gold", "Platinum"];  
        $discounts = [5, 10, 15];  
        $fees = [100000, 300000, 600000];  
  
        foreach (range(0,2) as $index) {  
            DB::table('membership_types')->insert([  
                'type' => $types[$index],  
                'discount' => $discounts[$index],  
                'yearly_fee' => $fees[$index]  
            ]);  
        }  
    }  
}  
?>
```

Kita juga perlu menambah membership ini di seeder untuk table customer, ubahlah isi method `run()` menjadi:

database/seeds/CustomersTableSeeder.php

```
....  
public function run()  
{  
    $faker = Faker::create();  
    $membership_type_id = [null, 1, 2];  
  
    foreach ( range(1,10) as $index ) {  
        DB::table('customers')->insert([  
            'name' => $faker->name,  
            'phone' => $faker->phoneNumber,  
            'address' => $faker->address,  
            'membership_type_id' => $membership_type_id[rand(0,3)]  
        ]);  
    }  
}
```

```
    }  
}  
....
```

Selanjutnya, kita panggil `MembershipTypesTableSeeder` dari `DatabaseSeeder`. Pemanggilan class ini harus sebelum memanggil class `CustomersTableSeeder` agar tidak error karena relasi:

database/seeds/DatabaseSeeder.php

```
....  
public function run()  
{  
    Model::unguard();  
    $this->call('ProductsTableSeeder');  
    $this->call('MembershipTypesTableSeeder');  
    $this->call('CustomersTableSeeder');  
}  
....
```

Sip. Mari kita refresh:

```
vagrant@homestead:~/Code/sample-database$ php artisan migrate:refresh --seed  
Rolled back: 2015_04_09_236144_add_relationships_to_orders_products  
Rolled back: 2015_04_09_235458_add_relationships_to_orders_table  
Rolled back: 2015_04_09_233100_create_orders_tables  
Rolled back: 2015_04_09_143657_create_customers_table  
Rolled back: 2015_04_09_010128_create_products_table  
Nothing to rollback.  
Migrated: 2015_04_09_010128_create_products_table  
Migrated: 2015_04_09_143657_create_customers_table  
Migrated: 2015_04_09_233100_create_orders_tables  
Migrated: 2015_04_09_235458_add_relationships_to_orders_table  
Migrated: 2015_04_09_236144_add_relationships_to_orders_products  
Migrated: 2015_04_09_236252_create_membership_types_table  
Berhasil menambah mobil!  
Seeded: ProductsTableSeeder  
Seeded: MembershipTypesTableSeeder  
Seeded: CustomersTableSeeder
```

Pastikan table `membership_types` sudah memiliki isi:

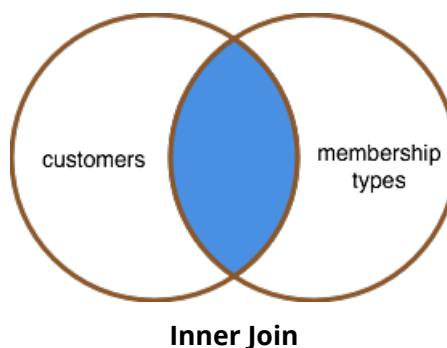
```
mysql> select * from membership_types;
+----+-----+-----+-----+
| id | type | discount | yearly_fee |
+----+-----+-----+-----+
1	Silver	5	100000
2	Gold	10	300000
3	Platinum	15	600000
+----+-----+-----+
```

Dan table customer sudah memiliki field `membership_type_id`:

```
mysql> select id, name, membership_type_id from customers;
+----+-----+-----+
| id | name | membership_type_id |
+----+-----+-----+
1	Rita Bosco	2
2	Jabari Cummerata	NULL
3	Enos Koelpin PhD	NULL
4	Tony Kunde III	NULL
5	Gwen Cummerata IV	1
6	Marcia Bogisich	1
7	Dr. German Ruecker IV	1
8	Luciano Prohaska	NULL
9	Rene Paucek	1
10	Mr. Alex Brown PhD	2
+----+-----+-----+
```

Inner Join (Join)

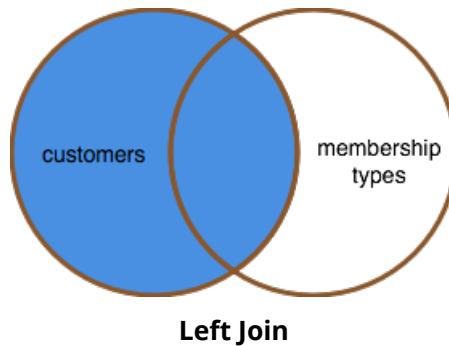
Kita mulai dari join sederhana, dimana kita menampilkan semua customer yang memiliki membership dan semua membership yang memiliki customer.



```
>>> DB::table('customers')
    ->join('membership_types', 'customers.membership_type_id', '=', 'membership_types.id')
    ->select('customers.name', 'membership_types.type')
    ->get();
=> [
    <stdClass #0000000038cd0d37000000001f49429a> {
        name: "Rita Bosco",
        type: "Gold"
    },
    <stdClass #0000000038cd0d230000000001f49429a> {
        name: "Gwen Cummerata IV",
        type: "Silver"
    },
    ...
]
```

Left Join

Untuk menampilkan hanya customer yang memiliki membership (berikut detail membershipnya) maupun tidak, kita bisa pakai left join.



```
>>> DB::table('customers')->leftJoin('membership_types', 'customers.membership_type_id', '=', 'membership_types.id')->get();
=> [
    <stdClass #0000000038cd0dc8000000001f49429a> {
        id: 1,
        name: "Gwen Cummerata IV",
        phone: "646-510-7093",
        address: "9159 Barbara Pike South Brisa, NM 92534-5263",
        membership_type_id: 1,
        type: "Silver",
        discount: 5,
    }
]
```

```

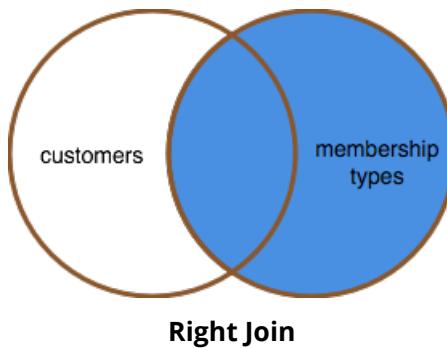
    yearly_fee: 100000
},
<stdClass #0000000038cd0dcb000000001f49429a> {
    id: null,
    name: "Jabari Cummerata",
    phone: "322-218-1176x2203",
    address: "95003 Flatley Ramp Suite 442\nSouth Elsebury, NY 65755",
    membership_type_id: null,
    type: null,
    discount: null,
    yearly_fee: null
},
...
]

```

Terlihat disini, customer yang tidak memiliki `membership_type_id` tetap ditampilkan.

Right Join

Untuk menampilkan semua membership baik yang memiliki member maupun tidak, kita dapat menggunakan left join dari table `membership_types` atau right join dari table `customers`. Mari kita gunakan right join.



```

>>> DB::table('customers')->rightJoin('membership_types', 'customers.membership_t\
ype_id', '=', 'membership_types.id')->get();
=> [
    <stdClass #0000000038cd0d3b000000001f49429a> {
        id: 2,
        name: "Rita Bosco",
        phone: "(929)062-7235x6812",
        address: "20410 Corrine Summit Apt. 887\nSouth Scarlett, ID 25784",
        membership_type_id: 2,
    }
]

```

```
        type: "Gold",
        discount: 10,
        yearly_fee: 300000
    },
    ...
<stdClass #0000000038cd0d17000000001f49429a> {
    id: 3,
    name: null,
    phone: null,
    address: null,
    membership_type_id: null,
    type: "Platinum",
    discount: 15,
    yearly_fee: 600000
}
```

Terlihat disini, meskipun tidak ada customer dengan membership platinum, ia tetap ditampilkan.

Where Lanjutan

Terkadang ingin membuat kondisi yang cukup kompleks dalam query. Perhatikan isi table products berikut:

```
mysql> select * from products;
+----+-----+-----+-----+-----+
| id | name      | description | price     | stock   |
+----+-----+-----+-----+-----+
1	Accord	Tipe Otomatis	137000000	37
2	Civic	Tipe manual	347000000	29
3	City	Tipe Otomatis	594000000	37
4	CR-V	Tipe manual	279000000	23
5	Jazz	Tipe manual	276000000	30
6	Freed	Tipe manual	160000000	39
7	Mobilio	Tipe manual	661000000	33
+----+-----+-----+-----+-----+
```

Jika seorang customer hanya ingin mencari mobil “Civic” maka ini query yang akan dibuat:

```
mysql> select * from products where name = "Civic";
+----+-----+-----+-----+
| id | name | description | price | stock |
+----+-----+-----+-----+
| 2 | Civic | Tipe manual | 347000000 | 29 |
+----+-----+-----+-----+
```

Jika ia ingin mencari mobil dengan nama “Civic” ATAU mobil dengan deskripsi “Tipe manual” DAN harga < 200.000.000 maka ini query yang akan dilakukan:

```
mysql> select * from products where name = "Civic" or (description = "Tipe manua\
l" and price < 200000000);
+----+-----+-----+-----+
| id | name | description | price | stock |
+----+-----+-----+-----+
| 2 | Civic | Tipe manual | 347000000 | 29 |
| 6 | Freed | Tipe manual | 160000000 | 39 |
+----+-----+-----+-----+
```

Terlihat kompleks? Oke, Bagaimana cara membuat query ini di Laravel?

Parameter Grouping

Untuk menyelesaikan query diatas, kita bisa menggunakan parameter grouping di Query Builder. Caranya dengan membuat kondisi where selanjutnya dalam bentuk Closure, seperti berikut:

```
>>> DB::table('products')->where('name', 'Civic')
    ->orWhere(function($query) {
        $query->where('description', 'Tipe manual')
            ->where('price', '<', 200000000);
    })->get();
=> [
    <stdClass #000000002bf265c4000000007c47b32c> {
        id: 2,
        name: "Civic",
        description: "Tipe manual",
        price: 347000000,
        stock: 29
    },
    <stdClass #000000002bf265ca000000007c47b32c> {
        id: 6,
```

```
        name: "Freed",
        description: "Tipe manual",
        price: 160000000,
        stock: 39
    }
]
```

Exists Statement

Kita juga dapat menggunakan statement exist di Query Builder. Statement ini biasanya digunakan sebagai sub-query sebelum kita mengeksekusi query utama. Misalnya, kita akan menampilkan semua product jika ada product dengan stock < 10. Misalnya, kondisi table seperti ini:

```
mysql> select * from products;
+----+-----+-----+-----+
| id | name      | description | price      | stock |
+----+-----+-----+-----+
1	Accord	Tipe Otomatis	137000000	37
2	Civic	Tipe manual	347000000	29
3	City	Tipe Otomatis	594000000	37
4	CR-V	Tipe manual	279000000	23
5	Jazz	Tipe manual	276000000	30
6	Freed	Tipe manual	160000000	39
7	Mobilio	Tipe manual	661000000	33
+----+-----+-----+-----+
```

Untuk membuat exist statement, di SQL berikut syntaxnya:

```
mysql> select * from products where exists (select * from products where stock < \
10);
Empty set (0.01 sec)
```

Menggunakan query builder, syntaxnya akan menjadi:

```
>>> DB::table('products')
    ->whereExists( function($query) {
        $query->select('*')
            ->from('products')
            ->where('stock', '<', 10);
    })->get();
=> []
```

Jika kita merubah stock sebuah product menjadi 10, maka kita akan mendapatkan output:

```
>>> DB::table('products')->where('id', 5)->update(['stock'=>5]);
=> 1
>>> DB::table('products')
    ->whereExists( function($query) {
        $query->select('*')
            ->from('products')
            ->where('stock', '<', 10);
    })->get();
=> [
    <stdClass #000000002bf265e8000000007c47b32c> {
        id: 1,
        name: "Accord",
        description: "Tipe Otomatis",
        price: 137000000,
        stock: 37
    },
    <stdClass #000000002bf265f6000000007c47b32c> {
        id: 2,
        name: "Civic",
        description: "Tipe manual",
        price: 347000000,
        stock: 29
    },
    ....
]
```

Aggregate

Beberapa method aggregate di Query Builder akan sangat bermanfaat untuk membuat report.

Count

Digunakan untuk menghitung jumlah record. Misalnya menghitung jumlah customer yang terdaftar:

```
>>> DB::table('customers')->count();
=> 10
```

Max

Digunakan untuk mencari nilai terbesar dari sebuah field. Misalnya mencari products dengan harga termahal:

```
>>> DB::table('products')->max('price');
=> 661000000
```

Jika kita ingin menampilkan detail productnya, bisa digabungkan dengan `where`:

```
>>> DB::table('products')
    ->where('price',
        DB::table('products')->max('price'))
    )->get();
=> [
    <stdClass #000000002bf265f2000000007c47b32c> {
        id: 7,
        name: "Mobilio",
        description: "Tipe manual",
        price: 661000000,
        stock: 33
    }
]
```

Min

Digunakan untuk mencari nilai terkecil dari sebuah field. Misalnya mencari products dengan stock paling sedikit:

```
>>> DB::table('products')->min('stock');
=> 5
```

Seperti max, kita juga dapat menggabungkannya dengan `where` untuk mendapatkan detail product dengan stock terendah:

```
>>> DB::table('products')
    ->where('stock',
        DB::table('products')->min('stock')
    )->get();
=> [
    <stdClass #000000002bf26515000000007c47b32c> {
        id: 5,
        name: "Jazz",
        description: "Tipe manual",
        price: 27600000,
        stock: 5
    }
]
```

Avg

Digunakan untuk mencari nilai rata-rata dari sebuah field. Misalnya mencari rata-rata stock products:

```
>>> DB::table('products')->avg('stock'); =\
> "29.0000"
```

Tentunya, kita dapat menggabungkan dengan `where`. Misalnya untuk mencari product yang stocknya dibawah rata-rata:

```
>>> DB::table('products')
    ->where('stock', '<',
        DB::table('products')->avg('stock')
    )->get();
=> [
    <stdClass #000000002bf26514000000007c47b32c> {
        id: 4,
        name: "CR-V",
        description: "Tipe manual",
        price: 27900000,
        stock: 23
    },
    <stdClass #000000002bf26510000000007c47b32c> {
        id: 5,
        name: "Jazz",
        description: "Tipe manual",
        price: 27600000,
```

```
        stock: 5
    }
]
```

Sum

Digunakan untuk mencari total dari nilai sebuah field. Misalnya untuk mencari total jumlah product yang dimiliki:

```
>>> DB::table('products')->sum('stock');
=>
> "203"
```

Raw Query

Untuk menggunakan raw query di Query Builder, kita dapat menggunakan `DB::raw()`. Misalnya kita akan menghitung total harga dari semua product (stock x harga). Query SQL nya akan seperti ini:

```
mysql> select sum(price*stock) from products;
+-----+
| sum(price*stock) |
+-----+
|      72960000000 |
+-----+
```

Jika kita buat di Query Builder tanpa `DB::raw()`, akan muncul error:

```
>>> DB::table('products')->sum('price*stock');
Illuminate\Database\QueryException with message 'SQLSTATE[42S22]: Column not found: 1054 Unknown column 'price*stock' in 'field list' (SQL: select sum(`price*stock`) as aggregate from `products`)'
```

Untuk itu, kita harus menggunakan `DB::raw()`:

```
>>> DB::table('products')->sum(DB::raw('price*stock'));
=> "72960000000"
```

Transaksi

Laravel mendukung penggunaan transaksi secara native. Sebelum kita membahas penggunaan transaksi mari kita *refresh* kembali pemahaman tentang transaksi di database.

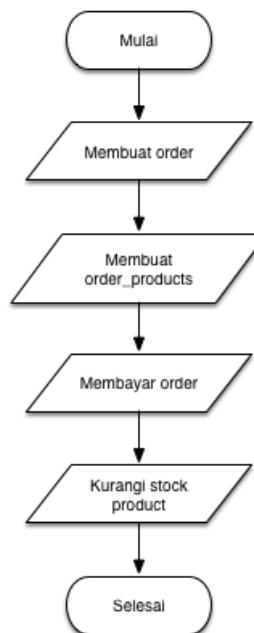
Apa itu Transaksi?

Transaksi adalah suatu teknik di database dimana kita dapat melakukan perintah DML (Insert, Update, Delete) dengan lebih aman. Dengan menggunakan transaksi, proses data menjadi lebih aman karena:

1. Jika terjadi kegagalan dalam salah satu proses di dalam transaksi, perubahan bisa dibatalkan.
2. Sistem akan mengunci record yang sedang dalam transaksi dari operasi DML lain, hingga proses transaksi selesai.

Mari kita buat contoh penggunaan transaksi pada order product. Dalam membuat order, kita akan buat 4 proses di database:

1. Membuat record di table `orders`
2. Menambah record baru di `order_products`
3. Membayar order (mengisi field `paid_at` di table `orders`)
4. Mengurangi stock product



Transaksi Order Product

Salah satu masalah yang akan muncul tanpa transaksi adalah bagaimana jika setelah proses ke-3 server mengalami error? Maka akan ada order yang dibayar tapi stock

product tidak berkurang. Untuk sebuah sistem yang penting, tentunya kesalahan ini tidak bisa ditolerir.

Dalam membuat transaksi, ada 3 tahapan yang biasanya dilakukan:

1. **Memulai transaksi** - Memberi tahu database bahwa beberapa query yang akan dilakukan merupakan bagian dari transaksi.
2. **Rolling back transaksi** - Membatalkan proses query yang dilakukan. Semua query yang telah dibuat tidak akan ditulis ke database dan memberi tahu database bahwa proses transaksi telah selesai.
3. **Commit transaksi** - Mengkonfirmasi proses query yang dilakukan. Semua query yang telah dibuat akan ditulis ke database dan memberi tahu database bahwa proses transaksi telah selesai.

Di Laravel kita dapat membuat transaksi dengan dua cara, otomatis dan manual. Mari kita pelajari keduanya dan pahami kapan kita harus menggunakan otomatis atau manual.

Transaksi Otomatis

Yang dimaksud dengan transaksi otomatis disini adalah kita tidak perlu memberi tahu Laravel secara eksplisit tahapan transaksi apa yang sedang dilakukan. Untuk membuat transaksi otomatis, kita menyimpan proses query database sebagai *closure* pada method `DB::transaction()`.

Untuk mencoba transaksi manual, mari kita buat transaksi order product dengan route `/order-product`:

app/Http/routes.php

```
Route::get('/order-product', function() {
    // memulai transaksi
    DB::transaction(function()
    {
        // Membuat record di table `orders`
        $order_id = DB::table('orders')->insertGetId(['customer_id'=>1]);

        // Menambah record baru di `order_products`
        DB::table('orders_products')->insert(['order_id'=>$order_id, 'product_id'=>5]);

        // Membayar order (mengisi field `paid_at` di table `orders`)
        DB::table('orders')->where('id',$order_id)->update(['paid_at'=>new DateT\
```

```
ime]);  
  
    // Mengurangi stock product  
    DB::table('products')->decrement('stock');  
});  
  
echo "Berhasil menjual " . DB::table('products')->find(5)->name . '. <br>';  
echo "Stock terkini : " . DB::table('products')->find(5)->stock;  
});
```

Misalnya, kita punya data product dengan id 5:

```
mysql> select * from products where id = 5;  
+----+-----+-----+-----+  
| id | name | description | price | stock |  
+----+-----+-----+-----+  
| 5 | Jazz | Tipe manual | 606000000 | 10 |  
+----+-----+-----+-----+
```

Jika kita mengakses route /order-product, ini yang akan kita dapatkan:



Berhasil menjual Jazz.
Stock terkini : 9

Berhasil membuat transaksi

Kitapun akan mendapatkan record baru di table orders dan orders_products:

```
mysql> select * from orders;
+----+-----+-----+
| id | customer_id | paid_at   |
+----+-----+-----+
| 1  |           1 | 2015-04-20 |
+----+-----+-----+

mysql> select * from orders_products;
+-----+-----+
| order_id | product_id |
+-----+-----+
|      1 |         5 |
+-----+-----+
```

Stock product dengan id 5 pun akan berkurang menjadi 9:

```
mysql> select * from products where id = 5;
+----+-----+-----+-----+
| id | name | description | price | stock |
+----+-----+-----+-----+
| 5  | Jazz  | Tipe manual | 606000000 |     9 |
+----+-----+-----+-----+
```

Menggunakan transaksi otomatis, semua query di dalam DB::transaction() akan dibatalkan jika terjadi error. Misalnya, mari kita buat Exception baru sebelum mengurangi stock product:

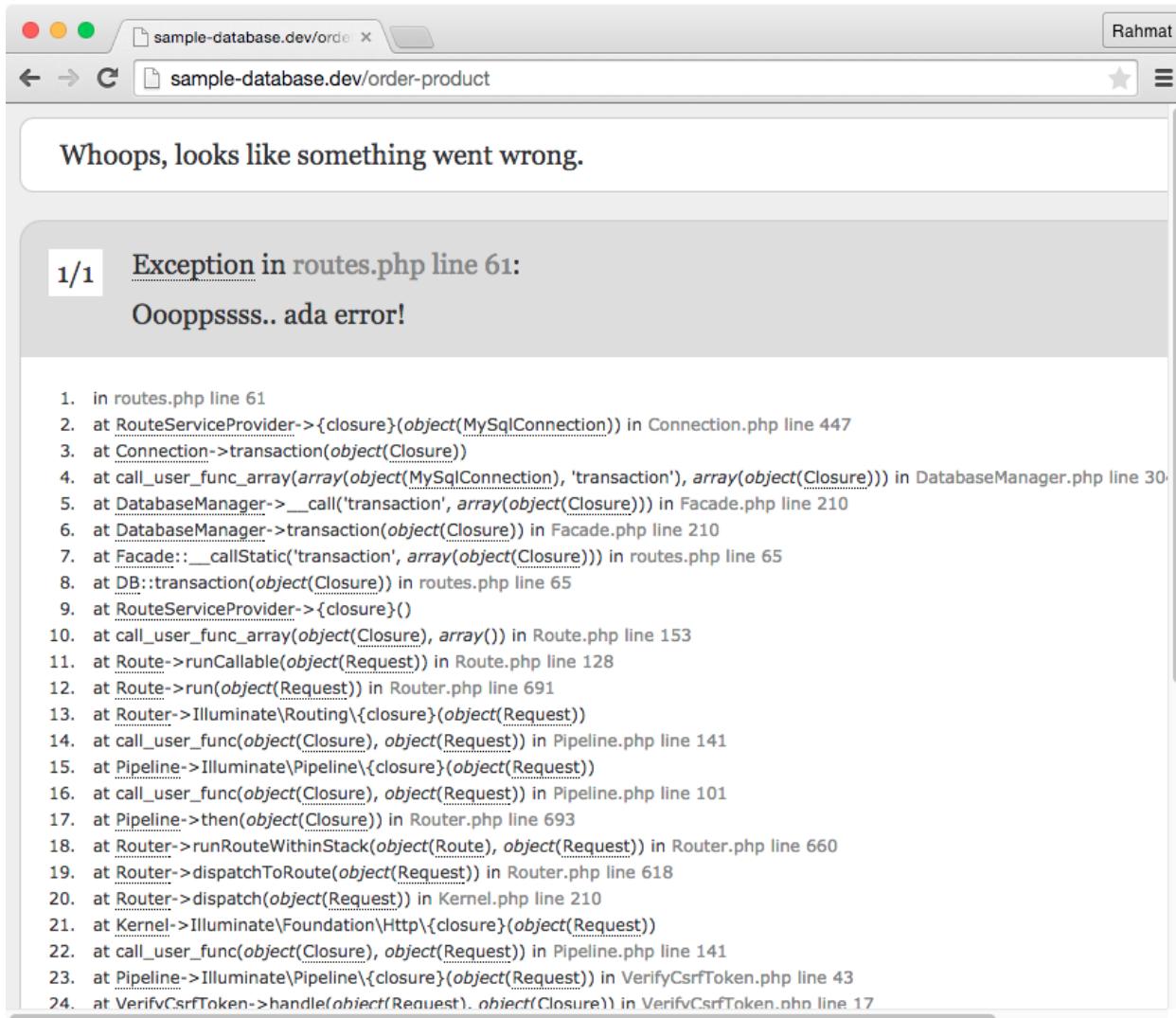
app/Http/routes.php

```
.....
// Membayar order (mengisi field `paid_at` di table `orders`)
DB::table('orders')->where('id',$order_id)->update(['paid_at'=>new DateTime]);

// Terjadi error..
throw new Exception("Ooops.. ada error!");

// Mengurangi stock product
DB::table('products')->decrement('stock');
....
```

Jika kita akses route /order-product, ini yang akan kita dapatkan:



Error pada Order Product

Meskipun kita telah menjalankan query untuk mengisi record baru di table orders dan orders_products, tapi keduanya akan dibatalkan. Sehingga isi database tetap tidak berubah:

```
mysql> select * from orders;
+----+-----+-----+
| id | customer_id | paid_at   |
+----+-----+-----+
| 1  |           1 | 2015-04-20 |
+----+-----+-----+

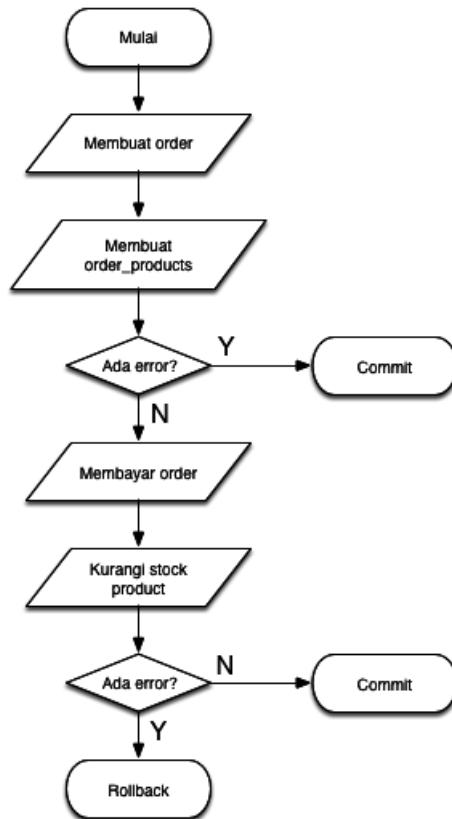
mysql> select * from orders_products;
+-----+-----+
| order_id | product_id |
+-----+-----+
|      1    |        5   |
+-----+-----+
```

Transaksi Manual

Berbeda dengan transaksi otomatis, menggunakan transaksi manual kita dapat menentukan kapan tiap tahapan dalam transaksi akan dilakukan. Ini akan sangat bermanfaat ketika kita ingin melakukan *rolling back* atau *commit* secara kondisional. Untuk melakukan transaksi secara manual, ada 3 method yang akan kita gunakan:

1. DB::beginTransaction() - memulai transaksi.
2. DB::rollback() - membatalkan transaksi.
3. DB::commit() - mengkonfirmasi transaksi.

Mari kita buat contoh pada alur order yang telah kita buat. Jika terjadi error sebelum proses ketiga (bayar order) maka transaksi akan di commit. Jika terjadi error setelah proses ketiga, maka semua proses query akan dibatalkan.



Alur Order Product

Jika kita ubah transaksi yang telah kita buat dengan logic diatas, maka syntaxnya akan menjadi:

app/Http/routes.php

```

Route::get('/order-product', function() {
    // memulai transaksi
    DB::beginTransaction();

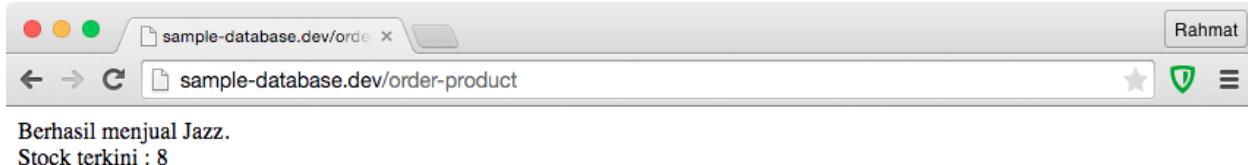
    try {
        // Membuat record di table `orders`
        $order_id = DB::table('orders')->insertGetId(['customer_id'=>1]);
        // Menambah record baru di `order_products`
        DB::table('orders_products')->insert(['order_id'=>$order_id, 'product_id'=>5]);
    } catch (Exception $e) {
        DB::commit();
        return "Error: " . $e->getMessage();
    }
})
  
```

```
try {
    // Membayar order (mengisi field `paid_at` di table `orders`)
    DB::table('orders')->where('id',$order_id)->update(['paid_at'=>$new Date\ime]);
    // Mengurangi stock product
    DB::table('products')->decrement('stock');
} catch (Exception $e) {
    DB::rollback();
    return "Error: " . $e->getMessage();
}

// menyimpan transaksi
DB::commit();

echo "Berhasil menjual " . DB::table('products')->find(5)->name . ". <br>";
echo "Stock terkini : " . DB::table('products')->find(5)->stock;
});
```

Pastikan terlebih dahulu syntax ini berjalan:



Order product berhasil

Untuk mengecek logic pertama, mari kita tambah error setelah proses kedua (menambah product pada order):

app/Http/routes.php

```
....  
try {  
    // Membuat record di table `orders`  
    $order_id = DB::table('orders')->insertGetId(['customer_id'=>1]);  
    // Menambah record baru di `order_products`  
    DB::table('orders_products')->insert(['order_id'=>$order_id, 'product_id'=>5\\  
]);  
    throw new Exception("Koneksi ke database terputus..");  
} catch (Exception $e) {  
    DB::commit();  
    return "Error: " . $e->getMessage();  
}  
....
```

Jika kita jalankan, maka akan muncul error **Koneksi ke database terputus...**

**Error di Transaksi**

Tapi, query ke database untuk table `orders` dan `orders_products` akan tetap tersimpan.

```
mysql> select * from orders;  
+----+-----+-----+  
| id | customer_id | paid_at |  
+----+-----+-----+  
1	1	2015-04-20
3	1	2015-04-20
4	1	0000-00-00
+----+-----+-----+  
  
mysql> select * from orders_products;
```

```
+-----+-----+
| order_id | product_id |
+-----+-----+
1	5
3	5
4	5
+-----+-----+
```

Tentunya, stock product tetap tidak akan berkurang.

```
mysql> select * from products where id = 5;
+-----+-----+-----+-----+
| id | name | description | price     | stock   |
+-----+-----+-----+-----+
|  5 | Jazz  | Tipe manual | 606000000 |      8 |
+-----+-----+-----+-----+
```

Tapi, jika error terjadi setelah proses ketiga (membayar order), maka semua query akan dibatalkan. Misalnya kita ubah menjadi:

app/Http/routes.php

```
try {
    // Membuat record di table `orders`
    $order_id = DB::table('orders')->insertGetId(['customer_id'=>1]);
    // Menambah record baru di `order_products`
    DB::table('orders_products')->insert(['order_id'=>$order_id, 'product_id'=>5\]);
    throw new Exception("Koneksi ke database terputus..");
} catch (Exception $e) {
    DB::commit();
    return "Error: " . $e->getMessage();
}

try {
    // Membayar order (mengisi field `paid_at` di table `orders`)
    DB::table('orders')->where('id',$order_id)->update(['paid_at'=>new DateTime]\);
    throw new Exception("Server Database mati..");
    // Mengurangi stock product
    DB::table('products')->decrement('stock');
} catch (Exception $e) {
    DB::rollback();
```

```
    return "Error: " . $e->getMessage();
}
```

Jika kita akses route /order-product, maka akan muncul:



Error: Server Database mati..

Error di Transaksi

Dan semua query akan dibatalkan, sehingga isi database tetap tidak berubah:

```
mysql> select * from orders;
+----+-----+-----+
| id | customer_id | paid_at   |
+----+-----+-----+
1	1	2015-04-20
3	1	2015-04-20
4	1	0000-00-00
+----+-----+-----+

mysql> select * from orders_products;
+-----+-----+
| order_id | product_id |
+-----+-----+
1	5
3	5
4	5
+-----+-----+

mysql> select * from products where id = 5;
+----+-----+-----+-----+
| id | name | description | price   | stock |
+----+-----+-----+-----+
```

```
| 5 | Jazz | Tipe manual | 606000000 |     8 |
+-----+-----+-----+-----+-----+
```

Dengan alur pembuatan transaksi yang sederhana di Laravel, diharapkan pengolahan data yang sensitif dari error akan lebih mudah. Sip.

Logging

Query database di Laravel memang menyenangkan. Tapi dibalik kemudahan syntax itu terkadang kita ingin mengetahui query apa saja yang sebenarnya terjadi. Disini, kita dapat menggunakan fitur query logging di Laravel. Dengan query logging, kita dapat mengetahui syntax SQL apa saja yang dijalankan oleh Laravel.

Untuk melakukan logging query, ada 3 langkah yang harus kita lakukan:

1. Memulai logging dengan `DB::connection()->enableQueryLog()`.
2. Menjalankan query.
3. Mendapatkan log query dengan `DB::getQueryLog()`.

Mari kita buat contoh untuk melakukan log terhadap beberapa query berikut:

app/Http/routes.php

```
Route::get('/customers', function() {
    DB::connection()->enableQueryLog();

    $products = DB::table('products')->get();
    $products = DB::table('customers')->whereIn('id', [1,4,5])->select(['name', \
'phone'])->get();
    $customers = DB::table('customers')->leftJoin('membership_types', 'customers.' \
membership_type_id', '=', 'membership_types.id')->get();

    var_dump(DB::getQueryLog());
});
```

Jika kita jalankan, kita akan mendapatkan query yang dijalankan oleh 3 query diatas berikut parameter yang dibinding (jika ada):



A screenshot of a web browser window titled "sample-database.dev/customers". The address bar also shows "sample-database.dev/customers". The main content area displays a large array dump representing a query log. The array has three elements (index 0, 1, 2) corresponding to three database queries. Each element contains a 'query' string, 'bindings' array, and a 'time' float value.

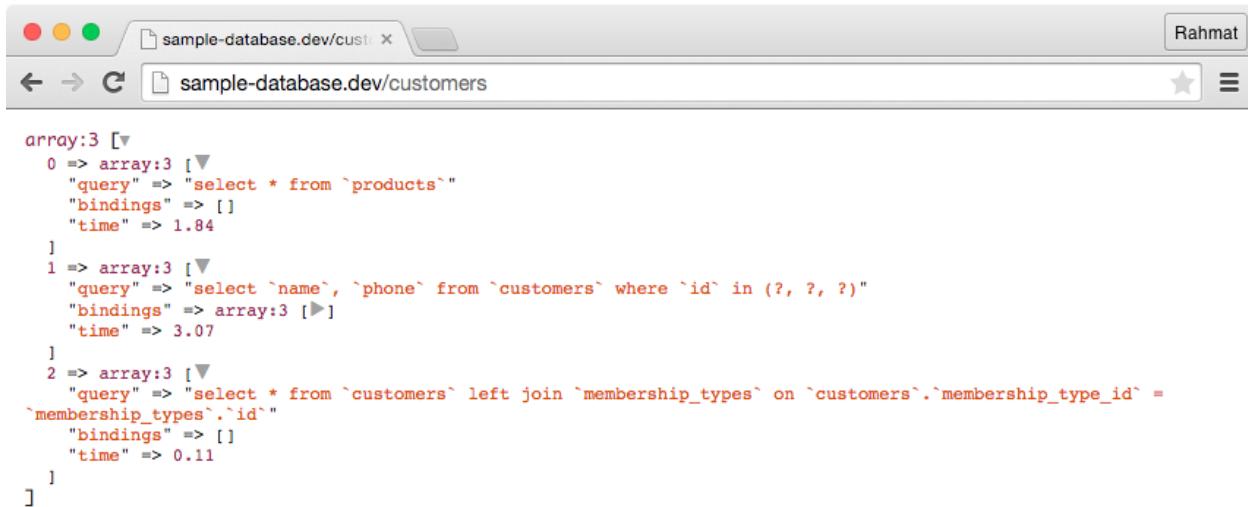
```
array (size=3)
0 =>
    array (size=3)
        'query' => string 'select * from `products`' (length=24)
        'bindings' =>
            array (size=0)
                empty
        'time' => float 2.92
1 =>
    array (size=3)
        'query' => string 'select `name`, `phone` from `customers` where `id` in (?, ?, ?)' (length=63)
        'bindings' =>
            array (size=3)
                0 => int 1
                1 => int 4
                2 => int 5
        'time' => float 0.01
2 =>
    array (size=3)
        'query' => string 'select * from `customers` left join `membership_types` on `customers`.`membership_type` = `membership_types`.`id`' (length=100)
        'bindings' =>
            array (size=0)
                empty
        'time' => float 4.05
```

Query Log

Kita juga dapat membuat tampilannya lebih menarik dengan helper dd():

app/Http/routes.php

```
Route::get('/customers', function() {
    ...
    var_dump(DB::getQueryLog());
    dd(DB::getQueryLog());
});
```



```

array:3 [▼
  0 => array:3 [▼
    "query" => "select * from `products`"
    "bindings" => []
    "time" => 1.84
  ]
  1 => array:3 [▼
    "query" => "select `name`, `phone` from `customers` where `id` in (?, ?, ?)"
    "bindings" => array:3 [▶]
    "time" => 3.07
  ]
  2 => array:3 [▼
    "query" => "select * from `customers` left join `membership_types` on `customers`.`membership_type_id` = `membership_types`.`id`"
    "bindings" => []
    "time" => 0.11
  ]
]

```

Dump variable dengan dd()

Database Caching

Penggunaan database yang tidak efisien akan menjadi salah satu penyebab lambatnya aplikasi. Jika sebuah query sering dijalankan dan data yang diakses jarang berubah, kita dapat memanfaatkan Cache dari Laravel untuk mengurangi jumlah query database yang dilakukan.

Syntax dasar untuk menyimpan hasil query ke database sebagai berikut:

```
$value = Cache::remember('nama.cache', $minutes, function()
{
    // query database
});
```

Hasil query akan disimpan dalam cache dengan nama `nama.cache` selama `$minutes`. Setelah melewati `$minutes`, hasil query akan di cache ulang ketika kita mencoba mengaksesnya. Kita juga dapat menghapus hasil query secara manual dengan:

```
Cache::forget('nama.cache');
```

Mari kita buat contoh cache untuk menyimpan product dengan harga termurah:

app/Http/routes.php

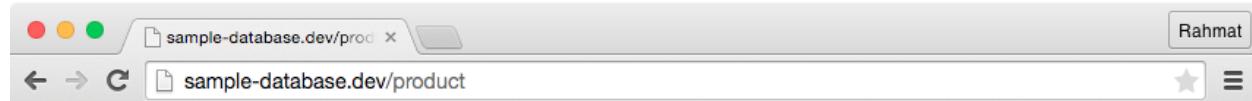
```
Route::get('/product', function() {
    DB::connection()->enableQueryLog();

    $product = Cache::remember('product.lowest-price', 10, function()
    {
        return DB::table('products')
            ->where('price', DB::table('products')->min('price'))
            ->get();
    });

    var_dump($product);

    var_dump(DB::getQueryLog());
});
```

Disini, kita menyimpan hasil query selama 10 menit. Jika kita akses route ini pertama kali, maka akan terlihat semua query yang dilakukan:



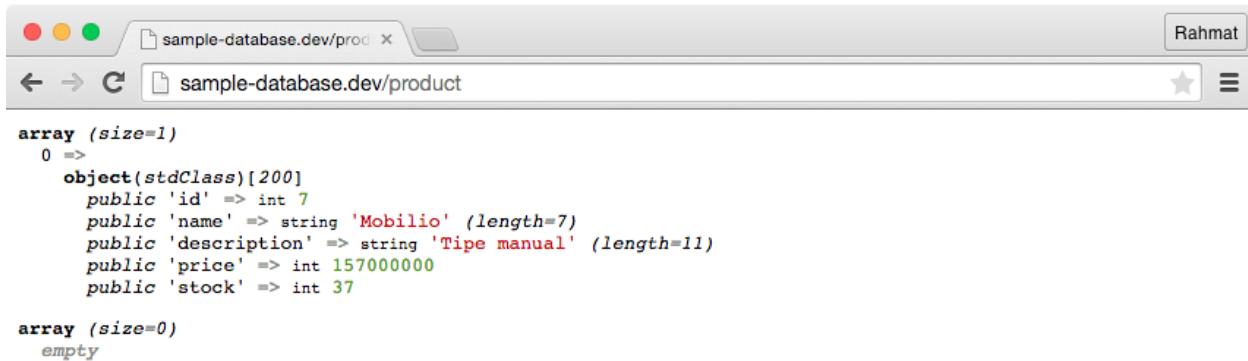
The screenshot shows a browser window with the URL 'sample-database.dev/product'. The page content displays the Laravel query log, which includes two arrays of data. The first array represents the results of the query to find the minimum price, and the second array represents the query itself.

```
array (size=1)
  0 =>
    object(stdClass)[191]
      public 'id' => int 7
      public 'name' => string 'Mobilio' (length=7)
      public 'description' => string 'Tipe manual' (length=11)
      public 'price' => int 157000000
      public 'stock' => int 37

array (size=2)
  0 =>
    array (size=3)
      'query' => string 'select min(`price`) as aggregate from `products`' (length=48)
      'bindings' =>
        array (size=0)
          empty
      'time' => float 4.58
  1 =>
    array (size=3)
      'query' => string 'select * from `products` where `price` = ?' (length=42)
      'bindings' =>
        array (size=1)
          0 => int 157000000
      'time' => float 1.98
```

Query product termurah

Jika kita *refresh* halaman, maka akan terlihat Laravel tidak melakukan query ke database:



```
array (size=1)
  0 ->
    object(stdClass)[200]
      public 'id' => int 7
      public 'name' => string 'Mobilio' (length=7)
      public 'description' => string 'Tipe manual' (length=11)
      public 'price' => int 157000000
      public 'stock' => int 37

array (size=0)
empty
```

Query product termurah dari cache

Misalnya kita hendak mengupdate harga sebuah product sambil menghapus cache yang kita buat, maka syntaxnya akan seperti berikut:

app/Http/routes.php

```
Route::get('/product', function() {
    DB::connection()->enableQueryLog();
    DB::table('products')->where('id', 1)->update(['price'=>110000000]);
    Cache::forget('product.lowest-price');
    ....
```

Jika kita *refresh* halaman, maka Laravel akan kembali melakukan semua query meskipun belum 10 menit:



```

array (size=1)
0 =>
object(stdClass)[202]
public 'id' => int 1
public 'name' => string 'Accord' (length=6)
public 'description' => string 'Tipe Otomatis' (length=13)
public 'price' => int 110000000
public 'stock' => int 20

array (size=3)
0 =>
array (size=3)
  'query' => string 'update `products` set `price` = ? where `id` = ?' (length=48)
  'bindings' =>
    array (size=2)
      0 => int 110000000
      1 => int 1
  'time' => float 4.84
1 =>
array (size=3)
  'query' => string 'select min(`price`) as aggregate from `products`' (length=48)
  'bindings' =>
    array (size=0)
      empty
  'time' => float 0.01
2 =>
array (size=3)
  'query' => string 'select * from `products` where `price` = ?' (length=42)
  'bindings' =>
    array (size=1)
      0 => int 110000000
  'time' => float 2.4

```

Clear cache product termurah

Penggunaan cache sangat disarankan untuk query yang sangat sering dilakukan tapi memiliki data yang jarang berubah. Jika digunakan dengan tepat, akan sangat terasa peningkatan performa Laravel dengan mengaplikasikan cache. Untuk memahami lebih lengkap tentang penggunaan Cache, baca [dokumentasi resmi⁹⁰](#). Sip.

Menggunakan Multiple Database

Dalam membuat koneksi ke database, secara default Laravel akan menggunakan koneksi yang kita isi pada key `default` di file `config/database.php`.

⁹⁰ <http://laravel.com/docs/5.0/cache>

config/database.php

```
<?php  
return [  
    ...  
    'default' => 'mysql',  
    ...  
]
```

Kita juga dapat mengubah koneksi database yang digunakan pada saat *runtime*. Ini akan sangat bermanfaat untuk membagi beban aplikasi. Untuk melakukan hal tersebut, ada 2 langkah yang harus kita lakukan:

1. Membuat konfigurasi koneksi database di config/database.php
2. Mengubah koneksi yang akan digunakan dengan method connection()

Mari kita buat contoh untuk menggunakan dua koneksi database untuk sample struktur database yang telah kita buat. Mari kita buat database baru dengan nama homestead2:

```
mysql> create database homestead2;  
Query OK, 1 row affected (0.00 sec)
```

Untuk membuat koneksi ke database ini, mari kita buat konfigurasi dengan nama mysql2 di config/database.php:

config/database.php

```
...  
'mysql2' => [  
    'driver'      => 'mysql',  
    'host'        => env('DB_HOST', 'localhost'),  
    'database'   => env('DB_DATABASE_2', 'homestead2'),  
    'username'   => env('DB_USERNAME', 'forge'),  
    'password'   => env('DB_PASSWORD', ''),  
    'charset'     => 'utf8',  
    'collation'   => 'utf8_unicode_ci',  
    'prefix'      => '',  
    'strict'      => false,  
,  
...  
]
```

Untuk memudahkan, disini kita menggunakan konfigurasi yang hampir sama dengan konfigurasi default mysql, hanya saja nama database yang kita rubah. Karena kita menggunakan key DB_DATABASE_2, mari kita tambahkan pada file .env:

.env

```
....  
DB_DATABASE_2=homestead2  
....
```

Mari kita cek terlebih dahulu koneksi yang kita buat:

```
>>> DB::connection('mysql2')  
=> <Illuminate\Database\MySqlConnection #0000000016d043150000000031c4124f> {}  
>>>
```

Sip. Selanjutnya, pastikan database sudah berisi sample data. Jika belum, silahkan jalankan migrate / refresh dengan opsi --seed untuk menambah sample data.

Kita juga akan menambah struktur database yang sama berikut sample data pada database homestead2. Untuk melakukannya kita dapat menggunakan opsi --database pada saat menjalankan `migrate` atau `seed`:

```
vagrant@homestead:~/Code/sample-database$ art migrate --seed --database mysql2  
Migration table created successfully.  
Migrated: 2015_04_09_010128_create_products_table  
Migrated: 2015_04_09_143657_create_customers_table  
Migrated: 2015_04_09_233100_create_orders_tables  
Migrated: 2015_04_09_235458_add_relationships_to_orders_table  
Migrated: 2015_04_09_236144_add_relationships_to_orders_products  
Migrated: 2015_04_09_236252_create_membership_types_table  
Berhasil menambah mobil!  
Seeded: ProductsTableSeeder  
Seeded: MembershipTypesTableSeeder  
Seeded: CustomersTableSeeder
```

Sip. Untuk menggunakan koneksi selain koneksi default, kita dapat menggunakan opsi `connection()`. Misalnya, kita ambil produk dengan `id 1`:

```
>>> DB::connection('mysql2')->table('products')->find(1);
=> <stdClass #0000000016d043f90000000031c4124f> {
    id: 1,
    name: "Accord",
    description: "Tipe Otomatis",
    price: 566000000,
    stock: 17
}
```

Sementara jika menggunakan koneksi default:

```
>>> DB::table('products')->find(1);
=> <stdClass #0000000016d043120000000031c4124f> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 656000000,
    stock: 28
}
```

Begitupun, untuk melakukan query lain misalnya update:

```
>>> DB::connection('mysql2')->table('products')->where('id', 1)->update(['stock'=>40]);
=> 1
```

Sip.

Konfigurasi Read/Write

Jika Anda pernah mengkonfigurasi server database di skala besar, tentu pernah membuat konfigurasi beberapa koneksi database untuk satu aplikasi. Contohnya pada konfigurasi *master-slave* di MySQL. Dengan konfigurasi ini, setiap proses WRITE dilakukan pada satu server master. Sedangkan proses READ dibagikan ke beberapa server slave. Tentunya, dibalik layar kita telah mengkonfigurasi server slave agar menyalin data dari server master secara berkala.

Untuk membuat konfigurasi ini, kita dapat menambah key READ dan WRITE pada konfigurasi koneksi database. Misalnya, kita buat proses READ ke database homestead dan proses WRITE ke database homestead2:

config/database.php

```
....  
'mysql' => [  
    'read' => [  
        'database' => env('DB_DATABASE', 'forge'),  
    ],  
    'write' => [  
        'database' => env('DB_DATABASE_2', 'homestead2'),  
    ],  
    'driver' => 'mysql',  
    'host' => env('DB_HOST', 'localhost'),  
    'username' => env('DB_USERNAME', 'forge'),  
    'password' => env('DB_PASSWORD', ''),  
    'charset' => 'utf8',  
    'collation' => 'utf8_unicode_ci',  
    'prefix' => '',  
    'strict' => false,  
],  
....
```

Karena pada konfigurasi yang kita miliki hanya nama database yang berbeda, maka pada isian `read` dan `write` kita hanya menambah key `database`. Konfigurasi lainnya seperti `host`, `username` dan `password` akan menggunakan isian yang sama. Tentunya, jika parameter konfigurasi tersebut berbeda, kita dapat mengisinya pada isian `read` dan `write`.

Sebelum mencoba konfigurasi ini mari kita samakan isian table products dengan perintah berikut:

```
mysql> use homestead2;  
Database changed  
  
mysql> delete from products;  
Query OK, 7 rows affected (0.00 sec)  
  
mysql> insert into products select * from homestead.products;  
Query OK, 7 rows affected (0.02 sec)  
Records: 7  Duplicates: 0  Warnings: 0  
  
mysql> select * from homestead2.products;  
+-----+-----+-----+-----+-----+
```

```
| id | name      | description    | price      | stock |
+---+-----+-----+-----+-----+
| 1 | Accord   | Tipe manual  | 6560000000 | 28 |
| 2 | Civic    | Tipe Otomatis | 6790000000 | 27 |
| 3 | City     | Tipe Otomatis | 6600000000 | 22 |
| 4 | CR-V     | Tipe manual  | 5900000000 | 32 |
| 5 | Jazz     | Tipe manual  | 2600000000 | 23 |
| 6 | Freed    | Tipe Otomatis | 2810000000 | 12 |
| 7 | Mobilio  | Tipe manual  | 5040000000 | 39 |
+---+-----+-----+-----+
7 rows in set (0.01 sec)
```

```
mysql> select * from homestead.products;
+---+-----+-----+-----+
| id | name      | description    | price      | stock |
+---+-----+-----+-----+
| 1 | Accord   | Tipe manual  | 6560000000 | 28 |
| 2 | Civic    | Tipe Otomatis | 6790000000 | 27 |
| 3 | City     | Tipe Otomatis | 6600000000 | 22 |
| 4 | CR-V     | Tipe manual  | 5900000000 | 32 |
| 5 | Jazz     | Tipe manual  | 2600000000 | 23 |
| 6 | Freed    | Tipe Otomatis | 2810000000 | 12 |
| 7 | Mobilio  | Tipe manual  | 5040000000 | 39 |
+---+-----+-----+-----+
7 rows in set (0.00 sec)
```

Mari kita coba ubah stock product dengan id 1:

```
>>> DB::table('products')->where('id',1)->update(['stock'=>100]);
=> 1
```

Jika kita coba mengecek perubahan ini:

```
>>> DB::table('products')->find(1);
=> <stdClass #00000004799a197000000000f056753> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 656000000,
    stock: 28
}
```

Terlihat disini, stock product tidak berubah karena kita mengarahkan proses `write` ke database homestead2:

```
>>> DB::connection('mysql2')->table('products')->find(1);
=> <stdClass #00000004799a173000000000f056753> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 656000000,
    stock: 100
}
```

Tentunya, dibelakang layar mysql harus kita konfigurasi agar menduplikasi setiap perubahan pada database tersebut. Cara konfigurasi mysql tersebut diluar pembahasan buku ini, silahkan baca [tutorial ini⁹¹](#).



Source code dari bab ini bisa didapat di

Ringkasan

Menggunakan database dengan bantuan Query Builder di Laravel memang sangat nyaman. Bandingkan dengan mengetik semua query itu secara manual. Selain menggunakan query buider, kita juga dapat menggunakan ORM dari Laravel bernama Eloquent yang akan kita pelajari pada bab selanjutnya. Semangat!

⁹¹ <http://www.sitepoint.com/mysql-master-slave-replication-setting-up>

⁹² <https://github.com/rahmatawaludin/sample-database>

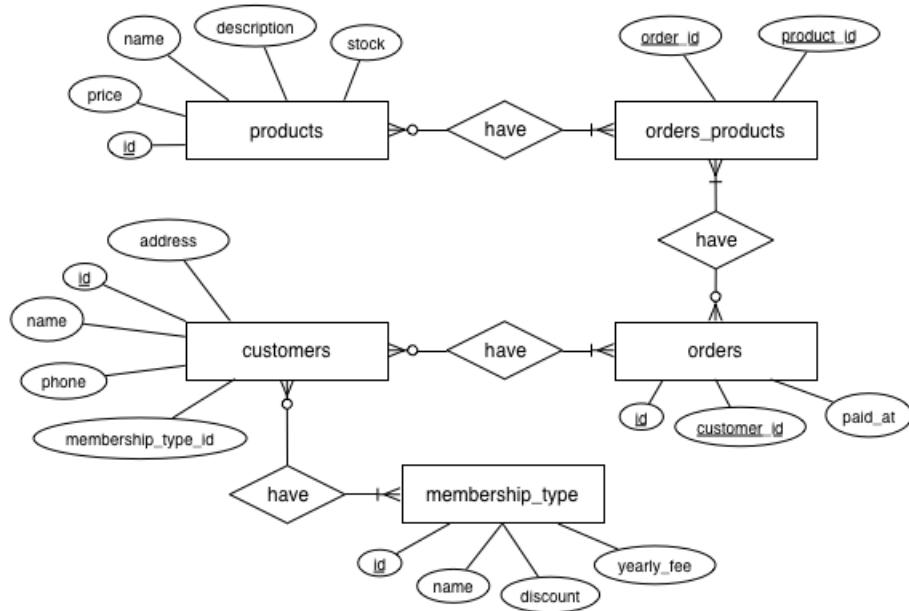
Model dari MVC

Banyak tools yang dikembangkan karena *kemalasan* programmer. Begitupun dengan ORM, meskipun teknik ini dibuat untuk memudahkan operasi query yang berhubungan dengan data relasional, saya lebih setuju jika dikatakan ORM ini tools yang pro programmer *males*. Pada bab ini kita akan mempelajari bagaimana menggunakan Eloquent, ORM dari Laravel, akan membuat kita semakin males membuat query database. *Hidup programmer males! :D*

Note: Oke, disamping bikin males, tentunya bikin coding kita juga lebih cepat.

Memahami ORM

Selama mengembangkan aplikasi yang terikat dengan database relasional misalnya MySQL, PostgreSQL atau SQLServer, pasti akan bertemu dengan query yang mengakses banyak table. Mari kita gunakan contoh struktur database yang kita buat pada bab sebelumnya:



Struktur Database Toko

Untuk mendapatkan semua `customer` yang memiliki `membership gold`, kita perlu melakukan query ke table `membership_types` dan `customers`. Salah satu contoh querinya seperti berikut:

```
mysql> mysql> select * from `customers` inner join `membership_types` on `customers`.`membership_type_id` = `membership_types`.`id` where `mep_types`.`type` = "Gold";
+-----+-----+-----+-----+-----+-----+
| id | name      | phone      | address          | membership_type_id | id | typ\\
e | discount | yearly_fee |
+-----+-----+-----+-----+-----+-----+
| 2 | Kraig..   | 927-3... | 651 Rhiannon Bypa... |                 2 | 2 | Gol\\
d |           10 |       300000 |                               2 | 2 | Gol\\
+-----+-----+-----+-----+-----+-----+
```

Atau menggunakan query builder:

```
>>> DB::table('customers')
    ->join('membership_types', 'customers.membership_type_id', '=', 'membership\\
types.id')
    ->where('membership_types.type', 'Gold')
    ->get();
=> [
    <stdClass #000000004799a14b000000000f056753> {
        id: 2,
        name: "Kraig Spinka",
        phone: "927-317-1561x1288",
        address: "651 Rhiannon Bypass Suite 045\\nNashland, NE 91287",
        membership_type_id: 2,
        type: "Gold",
        discount: 10,
        yearly_fee: 300000
    }
]
```

Begitupun untuk query yang lebih kompleks. Contohnya mendapatkan product yang apa saja yang telah diorder oleh seorang customer, kita harus membuat query yang melibatkan table `customers`, `orders`, `orders_products`, dan `products`. Jika kita memiliki data seperti berikut:

```
mysql> select name from customers where id = 1;
+-----+
| name      |
+-----+
| Leta Volkman |
+-----+

mysql> select * from products;
+----+-----+-----+-----+-----+
| id | name      | description | price      | stock |
+----+-----+-----+-----+-----+
| 1  | Accord    | Tipe manual | 6560000000 | 28   |
| 2  | Civic     | Tipe Otomatis | 6790000000 | 27   |
| 3  | City      | Tipe Otomatis | 6600000000 | 22   |
| 4  | CR-V      | Tipe manual | 5900000000 | 32   |
| 5  | Jazz       | Tipe manual | 2600000000 | 23   |
| 6  | Freed      | Tipe Otomatis | 2810000000 | 12   |
| 7  | Mobilio   | Tipe manual | 5040000000 | 39   |
+----+-----+-----+-----+-----+

mysql> select * from orders;
+----+-----+-----+
| id | customer_id | paid_at      |
+----+-----+-----+
| 1  |           3 | 2015-04-26 |
| 2  |           1 | 2015-04-26 |
| 3  |           3 | 2015-04-26 |
+----+-----+-----+

mysql> select * from orders_products;
+-----+
| order_id | product_id |
+-----+
|      1 |          3 |
|      1 |          2 |
|      2 |          3 |
|      2 |          5 |
|      3 |          2 |
|      3 |          1 |
+-----+
```

Jika kita ingin mengambil semua product yang pernah diorder oleh customer dengan id 1, salah satu query yang bisa dilakukan adalah:

```
mysql> select `customers`.`name`, `products`.`name` from `customers` inner join \
`orders` on `orders`.`customer_id` = `customers`.`id` inner join `orders_product` \
`orders_products` on `orders`.`id` = `orders_products`.`order_id` inner join `products` on `ord` \
`orders_products`.`product_id` = `products`.`id` where `customers`.`id` = 1;
+-----+-----+
| name | name |
+-----+-----+
| Leta Volkman | City |
| Leta Volkman | Jazz |
+-----+-----+
```

Atau menggunakan query buider:

```
>>> DB::table('customers')
    ->join('orders', 'orders.customer_id', '=', 'customers.id')
    ->join('orders_products', 'orders.id', '=', 'orders_products.order_id')
    ->join('products', 'orders_products.product_id', '=', 'products.id')
    ->where('customers.id', 1)
    ->select('customers.name', 'products.name')
    ->get();
=> [
    <stdClass #000000003f545c8f00000000646b438f> {
        name: "City"
    },
    <stdClass #000000003f545ceb00000000646b438f> {
        name: "Jazz"
    }
]
```

Wow!

Bayangkan jika untuk setiap query yang melibatkan relasi kita harus melakukan query sepanjang dan sekompelks itu, *repot kan?*

Nah, dari masalah seperti inilah ORM dikembangkan.

ORM (Object Relational Mapping)

Object-relational mapping (ORM, O/RM, and O/R mapping) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a “virtual object database” that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools. - Wikipedia

Sesuai deskripsi Wikipedia, menggunakan ORM kita akan dapat mengakses setiap baris record di database dengan syntax yang lebih OOP *friendly*. Dengan menggunakan ORM, kita tidak perlu lagi membuat query database manual. Semua field dan relasi antar table pun akan dipetakan menjadi attribut atau method pada object ORM yang kita buat.

Salah satu manfaat yang bisa kita dapatkan (disamping manfaat lainnya) diantaranya query pertama diatas dapat kita dapatkan dengan syntax berikut:

```
$membershipType = MembershipType::where('name', 'Gold')->first();  
$membershipType->customers;
```

Sementara query kedua bisa kita dapatkan dengan syntax berikut:

```
$customer = Customer::find(1);  
$customer->orders->products;
```

Silahkan bandingkan dengan query manual diatas. Saya yakin Anda akan setuju bahwa **mengunakan ORM akan membuat syntax kita lebih elegan dan lebih mudah dipahami maksudnya**.

Eloquent, ORM Powerful dari Laravel

Dari sekian banyak ORM PHP yang bertebaran di internet, Taylor Otwell (pembuat Laravel) memilih untuk membuat ORM sendiri untuk Laravel dengan nama Eloquent. Meskipun kita tidak wajib menggunakan Eloquent, tapi sangat disarankan untuk menggunakan Eloquent ketika kita hendak berinteraksi dengan database relasional.

Sebagai sebuah ORM, tentunya kelebihan Eloquent terletak pada bagaimana dia menangani relasi antar table. Saat tulisan ini dibuat, Eloquent telah mendukung penggunaan relasi berikut:

- One To One
- One To Many
- Many To Many
- Has Many Through
- Polymorphic Relations
- Many To Many Polymorphic Relations

Terdapat pula fitur-fitur lain yang akan memudahkan kita selama berinteraksi dengan database, diantaranya:

- Mass Assignment
- Soft Deleting
- Timestamps
- Query Scopes
- Eager Loading
- Accessors & Mutators
- Attribute Casting
- Model Events
- Model Observers
- Model URL Generation
- Converting To Arrays / JSON

Untuk basic query, Eloquent menggunakan syntax dari Query Builder. Jika belum paham, silahkan buka kembali bab sebelumnya. Mari kita pelajari fitur-fitur Eloquent satu-persatu.

Konfigurasi

Dalam menggunakan Eloquent, setiap class yang kita buat akan mewakili satu table. Class tersebut umumnya disebut Model, karena memodelkan table yang diwakilinya. Jika sebuah class mewakili table `customers`, maka ia akan disebut model Customer. Jika sebuah class mewakili table `orders`, maka ia akan disebut model Order. Dan seterusnya...

Membuat Model

Untuk membuat model baru, ada 2 langkah yang harus kita lakukan:

1. Membuat table dengan nama plural.
2. Membuat model dengan nama singular pada folder `app` atau lokasi lain yang kita kehendaki.

Mari kita buat model untuk table `customers` yang telah kita buat pada bab sebelumnya. Disini, table `customers` sudah memiliki nama yang plural sehingga kita tidak perlu merubah nama table. Jika ingin mengecek nama plural dari sebuah kata di Laravel dapat menggunakan helper `str_plural()`:

```
>>> str_plural('customer');
=> "customers"
>>> str_plural('chair');
=> "chairs"
>>> str_plural('mouse');
=> "mice"
>>> str_plural('horse');
=> "horses"
>>> str_plural('teach');
=> "teaches"
>>> str_plural('data');
=> "data"
>>> str_plural('money');
=> "money"
```

Selanjutnya mari kita buat model `Customer` di folder `app`. Langkah ini dapat dilakukan manual atau menggunakan generator dari Laravel. Jika menggunakan generator, gunakan perintah berikut:

```
vagrant@homestead:~/Code/sample-model$ art make:model Customer
Model created successfully.
Created Migration: 2015_04_10_093425_create_customers_table
```

Seperti terlihat pada output diatas, Laravel akan membuat dua buah file yaitu file model dan file migration. Karena kita sudah memiliki file migration dari bab sebelumnya, silahkan hapus file migration yang baru dibuat ini.

```
vagrant@homestead:~/Code/sample-model$ rm -rf database/migrations/2015_04_10_093\425_create_customers_table.php
```

Kita juga dapat mengintruksikan Laravel agar tidak membuat file migration dengan opsi `--no-migration`:

```
vagrant@homestead:~/Code/sample-model$ rm -rf app/Customer.php
vagrant@homestead:~/Code/sample-model$ art make:model Customer --no-migration
Model created successfully.
```

Isian default dari file model yang di-*generate* oleh Laravel sebagai berikut:

app/Customer

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

Nama model ini harus selalu singular dari nama tablenya. Untuk mengecek singular dari sebuah kata di Laravel, kita dapat menggunakan fungsi `str_singular()`:

```
>>> str_singular('books');  
=> "book"  
>>> str_singular('mice');  
=> "mouse"  
>>> str_singular('products');  
=> "product"
```

Karena model ini berada di folder `app`, maka secara default Laravel akan memberikan namespace `App`:

app/Customer

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

Selesai deh. Kini kita dapat menggunakan model `Customer`. Misalnya menampilkan customer dengan `id 1`:

```
>>> App\Customer::find(1);
=> <App\Customer #0000000043406bde000000001806c540> {
    id: 1,
    name: "Leta Volkman",
    phone: "1-640-989-2661x010",
    address: "467 Kari Circles Suite 247\nGorczanychester, CT 19945",
    membership_type_id: 3
}
```

Nama table dengan underscore

Laravel cukup cerdas untuk mengetahui nama table jika menggunakan underscore dalam namanya. Pada contoh kita sebelumnya, kita menggunakan nama table `membership_types` yang jika kita cek singular nya,

```
>>> str_singular('membership_types');
=> "membership_type"
```

Tapi, kita dapat menggunakan nama model `MembershipType`:

```
vagrant@homestead:~/Code/sample-model$ art make:model MembershipType --no-migration
Model created successfully.
```

Dan modelnya akan tetap berjalan:

```
>>> App\MembershipType::all();
=> <Illuminate\Database\Eloquent\Collection #0000000082dc7a8000000047dbfeac> [
    <App\MembershipType #0000000082dc7a7000000047dbfeac> {
        id: 1,
        type: "Silver",
        discount: 5,
        yearly_fee: 100000
    },
    <App\MembershipType #0000000082dc7a4000000047dbfeac> {
        id: 2,
        type: "Gold",
        discount: 10,
        yearly_fee: 300000
    },
    <App\MembershipType #0000000082dc7a5000000047dbfeac> {
```

```

        id: 3,
        type: "Platinum",
        discount: 15,
        yearly_fee: 600000
    }
]

```

Menggunakan table dengan nama berbeda

Terkadang kita dihadapkan untuk membuat aplikasi dengan database yang sudah terbuat. Biasanya nama tablenya pun kebanyakan tidak mengikuti aturan Laravel seperti `tb_customer`, `tb_produk`, dll. Jika kita lihat pada [source code Eloquent⁹³](#) untuk mendeteksi nama table, akan terlihat logic yang dilakukannya:

illuminate/database/blob/v5.0.28/Eloquent/Model.php#L1919-L1929

```

/**
 * Get the table associated with the model.
 *
 * @return string
 */
public function getTable()
{
    if (isset($this->table)) return $this->table;
    return str_replace('\\', '', snake_case(str_plural(class_basename($this))));
}

```

Terlihat disini, sebelum Eloquent menentukan nama table secara otomatis ia akan mengecek apakah ada attribut `$table` yang di set. Maka, kita dapat membuat model dengan table yang berbeda menggunakan attribut `$table`. Mari kita buat table `Pelanggan` dengan table `customers`:

```
vagrant@homestead:~/Code/sample-model$ php artisan make:model Pelanggan --no-migration
Model created successfully.
```

Kemudian kita tambahkan attribut `$table`:

⁹³<https://github.com/illuminate/database/blob/v5.0.28/Eloquent/Model.php#L1919-L1929>

app/Pelanggan.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Pelanggan extends Model {
    protected $table = 'customers';
    //
}
```

Kini, kita dapat menggunakan model ini:

```
>>> App\Pelanggan::find(4);
=> <App\Pelanggan #000000003578a549000000005c1379cc> {
    id: 4,
    name: "Gabriel Nolan",
    phone: "1-220-069-4253x303",
    address: "6896 Laron Cliffs\nRatkeberg, RI 14707-9159",
    membership_type_id: 3
}
```

Ubah lokasi model

Secara default, Laravel meletakan model di folder `App`. Jika diinginkan kita dapat memindahkan ke lokasi lain, selama masih dapat di-*load* oleh autoload composer (tentunya harus memberikan konfigurasi namespace). Contohnya kita pindahkan model customer ke folder `app/Models`:

```
vagrant@homestead:~/Code/sample-model$ mkdir app/Models
vagrant@homestead:~/Code/sample-model$ mv app/Customer.php app/Models/
```

Selanjutnya kita harus merubah namespace dari model `Customer`:

app/Models/Customer.php

```
<?php namespace App\Models;
...
```

Kini, model siap digunakan:

```
>>> App\Models\Customer::find(2);
=> <App\Models\Customer #000000033122f88000000004032be7e> {
    id: 2,
    name: "Dr. Bennett Pacocha I",
    phone: "344-242-8296x672",
    address: "7119 Christine Junctions Suite 360\nEast Dulce, RI 11014-4598",
    membership_type_id: 1
}
```

Timestamps

Secara default, Eloquent akan mengaktifkan `timestamp` dalam setiap modelnya. Fitur ini akan sangat bermanfaat untuk mencatat kapan record dibuat dan diubah. Dia bekerja dengan mengisi field `created_at` (bertipe timestamp) pada saat membuat record dan mengubah field `updated_at` (bertipe timestamp) pada saat mengubah record. Jika kedua field ini tidak ditemukan, maka Laravel akan memunculkan error. Tentunya, kita dapat menon-aktifkan fitur ini jika tidak diinginkan.

Mari kita aktifkan fitur timestamp pada model `customers`. Untuk melakukannya kita perlu menambah field `created_at` `updated_at` di table `customers`. Kita dapat membuatnya satu-persatu dengan migration atau menggunakan `$table->timestamps()` untuk menambah keduanya secara otomatis. Mari kita gunakan cara kedua, tambahkan baris berikut pada table `customers`:

database/migrations/2015_04_09_143657_create_customers_table.php

```
....  
public function up()  
{  
    Schema::create('customers', function(Blueprint $table)  
    {  
        $table->increments('id');  
        $table->string('name');  
        $table->string('phone')->nullable();  
        $table->string('address')->nullable();  
        $table->timestamps();  
    });  
}  
....
```

Refresh dan seed kembali database dengan `php artisan migrate:refresh --seed`. Setelah dijalankan, pastikan kolom `created_at` dan `updated_at` telah dibuat:

```
mysql> desc customers;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Ext |
+-----+-----+-----+-----+-----+
| id   | int(10) unsigned | NO | PRI | NULL | auto\_increment |
| name | varchar(255) | NO | | NULL | |
| phone | varchar(255) | YES | | NULL | |
| address | varchar(255) | YES | | NULL | |
| created_at | timestamp | NO | | 0000-00-00 00:00:00 | |
| updated_at | timestamp | NO | | 0000-00-00 00:00:00 | |
| membership_type_id | int(10) unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

Mari kita coba tambah satu customer dengan model Customer:

```
>>> $customer = new App\Models\Customer;
=> <App\Models\Customer #0000000049e592f8000000002b94928e> {}
>>> $customer->name = "Beni";
=> "Beni"
>>> $customer->address = "Jakarta";
=> "Jakarta"
>>> $customer->phone = "0212342342"
=> "0212342342"
>>> $customer->save();
=> true
```

Jika kita cek record customer, akan terlihat tanggal record dibuat pada field `created_at`:

```
>>> $customer;
=> <App\Models\Customer #0000000049e592f8000000002b94928e> {
    name: "Didi",
    address: "Jakarta",
    phone: "0212342342",
    updated_at: "2015-04-09 00:43:15",
    created_at: "2015-04-09 00:43:15",
    id: 11
}
```

Jika kita coba update salah satu fieldnya:

```
>>> $customer->name = "Didi";
=> "Didi"
>>> $customer->save();
=> true
```

Akan terlihat field `updated_at` akan berubah:

```
>>> $customer->toArray();
=> [
    "name"      => "Didi",
    "address"   => "Jakarta",
    "phone"     => "0212342342",
    "updated_at" => "2015-04-09 00:45:36",
    "created_at" => "2015-04-09 00:43:15",
    "id"        => 11
]
```

Disini kita menggunakan method `toArray()` untuk menampilkan model dalam bentuk array.

Jika field `created_at` dan `updated_at` tidak ditemukan, maka Eloquent akan memunculkan error. Mari kita coba pada model Product. Kita buat dulu modelnya:

```
vagrant@homestead:~/Code/sample-model$ php artisan make:model Product --no-migration
Model created successfully.
```

Kita buat record baru:

```
>>> $mbPro = new App\Product;
=> <App\Product #00000000714e53d80000000052ee6663> {}
>>> $mbPro->name = "MacBook Pro 2015"
=> "MacBook Pro 2015"
>>> $mbPro->description = "Retina Display";
=> "Retina Display"
>>> $mbPro->price = 24000000;
=> 24000000
>>> $mbPro->stock = 18;
=> 18
>>> $mbPro->save();
Illuminate\Database\QueryException with message 'SQLSTATE[42S22]: Column not found: 1054 Unknown column 'updated_at' in 'field list' (SQL: insert into `products` (`name`, `description`, `price`, `stock`, `updated_at`, `created_at`) values ('MacBook Pro 2015', 'Retina Display', 24000000, 18, 2015-04-27 00:56:35, 2015-04-27 00:56:35)'
```

Terlihat diatas kita mendapat error **IlluminateDatabaseQueryException**. Untuk memperbaikinya kita dapat menambah field `created_at` dan `updated_at` seperti langkah sebelumnya. **Jika kita tidak punya kontrol terhadap struktur database, kita juga dapat menonaktifkan fitur timestamps dengan menambahkan attribut `$timestamps berisi false`.** Mari kita coba:

app/Product.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model {

    public $timestamps = false;
    //
}
```

Jika kita coba lagi menambah Product, maka akan berhasil (restart tinker terlebih dahulu):

```
>>> $mbPro = new App\Product;
=> <App\Product #0000000022b965ae000000000062d55b> {}
>>> $mbPro->name = "MacBook Pro 2015"
=> "MacBook Pro 2015"
>>> $mbPro->description = "Retina Display";
=> "Retina Display"
>>> $mbPro->price = 24000000;
=> 24000000
>>> $mbPro->stock = 18;
=> 18
>>> $mbPro->save();
=> true
```



Timestamp dan UTC

Jika diperhatikan, terkadang isian di `create_at` atau `updated_at` tidak sesuai dengan waktu di komputer. Ini karena Laravel akan menggunakan waktu UTC untuk mengisinya. Kita dapat mengubahnya di file `config/app.php`, cari isian `timezone` dan ubah dengan [kode timezone⁹⁴](#) yang kita inginkan. Misalnya untuk Jakarta menjadi **Asia/Jakarta**.

Mengubah Primary Key

Secara default Eloquent akan mengasumsikan field dengan nama `id` sebagai primary key, jika diinginkan kita dapat merubahnya menjadi field lain dengan mengisi attribut `$primaryKey` dengan nama field yang kita inginkan. Ini akan sangat berguna jika kita membangun aplikasi dengan struktur data yang telah ditentukan sebelumnya dimana kita tidak memiliki kontrol untuk merubah struktur database.

Pada contoh sebelumnya, mari kita coba pada model `product`. Jika kita ubah field `id` menjadi `product_id`:

```
>>> DB::statement('ALTER TABLE products CHANGE COLUMN id product_id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT');
=> true
```

Agar query ke record ini tetap berhasil, kita harus menambah attribut `$primaryKey`:

⁹⁴http://en.wikipedia.org/wiki/List_of_tz_database_time_zones

app/Product.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model {

    public $timestamps = false;
    protected $primaryKey = 'product_id';
    //
}
```

Jika kita coba membuat record baru maka akan tetap berhasil :

```
>>> $iPhone = new App\Product;
=> <App\Product #0000000022b965570000000000062d55b> {}
>>> $iPhone->name = "iPhone 6";
=> "iPhone 6"
>>> $iPhone->description = "Warna Hitam"
=> "Warna Hitam"
>>> $iPhone->price = 13000000;
=> 13000000
>>> $iPhone->stock = 23;
=> 23
>>> $iPhone->save();
=> true
```

Note: Pastikan untuk mengembalikan primary key ke `id` setelah mencoba tahap diatas untuk memudahkan contoh selanjutnya.

Multiple Database

Pada bab sebelumnya, kita telah memahami bahwa di Laravel kita dapat menggunakan beberapa konfigurasi database secara bersamaan. Bisa dengan menggunakan method `connection()` untuk merubah koneksi yang digunakan atau menggunakan key `read` dan `write` untuk merubah koneksi untuk membaca atau menulis.

Di level model, koneksi yang akan digunakan secara default adalah koneksi yang diisikan pada key `default` di `config/database.php`. Untuk merubah koneksi yang digunakan, ada dua cara:

- Menggunakan attribut `$connection` pada model dan mengisinya dengan koneksi yang akan digunakan.
- Menggunakan method `on()` pada model untuk merubah koneksi yang akan digunakan secara live.

Misalnya kita akan menggunakan koneksi `mysql2` yang telah kita buat pada bab sebelumnya. Mari kita contohkan pada model Customer. Untuk cara pertama, kita cukup menambah attribut berikut:

app/Models/Customer.php

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Customer extends Model {
    protected $connection = 'mysql2';
    //
}
```

Setelah diubah seperti diatas, kita dapat menjalankan query seperti biasa dan semua query akan dijalankan pada koneksi `mysql2`.

Sedangkan cara kedua, kita dapat menggunakan method `on()` seperti berikut:

```
>>> App\Models\Customer::on('mysql2')->find(1);
=> <App\Models\Customer #000000063049e4c00000003893b5ad> {
    id: 1,
    name: "Miss Magdalen Legros V",
    phone: "(776)245-5987x342",
    address: "0652 Luisa Heights\nEast Jobury, KY 54222-3569",
    membership_type_id: 2
}
```

Mass Assignment

Query dasar untuk proses insert dan update dengan Eloquent adalah dengan mengisi setiap attribut pada model kemudian mengakhirinya dengan method `save()` seperti yang kita lakukan pada [contoh timestamps](#). Eloquent juga memungkinkan kita melakukan *mass assignment*. Dengan *mass assignment*, kita dapat membuat record baru dengan method `create()` dengan menggunakan parameter berupa array asosiatif berisi nama field berikut nilainya. Seperti berikut:

```
$model = Model::create(["field1"=>"value", "field2"=>"value"]);
```

Begitupun untuk proses update kita dapat menggunakan method `update()` menggunakan *mass assignment*:

```
$model = Model::find(1);
$model->update(["field1"=>"value", "field2"=>"value"]);
```

Mass assignment ini akan sangat bermanfaat ketika kita akan mengisi model yang memiliki attribut sangat banyak. Misalnya, sebuah model memiliki 20 field yang harus diisi, kita cukup membuat form dengan nama input yang sama dengan nama attribut dan kita dapat membuat model dengan syntax:

```
$model = Model::create(Input::all());
```

Untuk menggunakan *mass assignment* kita harus mengkonfigurasi field mana saja yang bisa diisi. Dengan teknik ini, kita dapat mengisi field yang dibutuhkan dengan cepat tanpa mengorbankan sisi keamanan.

Contohnya, dalam sebuah blog model `Post` memiliki field `title`, `content`, dan `published`. Misalnya seorang member dapat menulis artikel tapi tidak dapat mem-publish artikel. Disini, kita bisa mengizinkan *mass assignment* untuk field `title` dan `content`, sambil tetap melindungi field `published` agar tidak di set oleh member.

Ada dua teknik untuk konfigurasi *mass assignment*, **whitelist** dan **blacklist**. Mari kita pelajari keduanya.

Whitelist

Menggunakan teknik ini, kita harus memberitahu Laravel field mana saja yang **boleh diisi**. Teknik ini yang paling sering saya pakai. Untuk menggunakan teknik ini kita harus membuat attribut `$fillable` bertipe array dengan isi field yang kita izinkan untuk diisi. Mari kita coba pada model `Customer`, kita buat agar hanya field `name` dan `phone` yang dapat diisi dengan *mass assignment*. Ubah model `Customer` menjadi:

app/Models/Customer.php

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Customer extends Model {

    protected $fillable = ['name', 'phone'];

}
```

Jika kita mencoba membuat record baru dengan menggunakan *mass assignment* sambil mengisi field address:

```
>>> $beni = App\Models\Customer::create([ 'name'=>'Beni', 'phone'=>'0214532231', 'a\
ddress'=>'Bandung']);
=> <App\Models\Customer #000000004411f73f0000000030409604> {
    name: "Beni",
    phone: "0214532231"
}
```

Secara default Eloquent akan menyembunyikan field yang tidak dapat diisi. Kita dapat mengeceknya dengan:

```
>>> $beni->address;
=> null
```

Terlihat disini, kita field address berisi null meskipun kita mengisinya pada saat menggunakan method `create()`.

Tentunya, meskipun kita tidak dapat mengisi field tersebut dengan menggunakan method `create()`, kita tetap dapat mengisinya per-field seperti berikut:

```
>>> $beni = App\Models\Customer::create(['name'=>'Beni', 'phone'=>'0214532231']);
=> <App\Models\Customer #00000004411f7ce000000030409604> {
    name: "Beni",
    phone: "0214532231"
}
>>> $beni->address = "Bandung";
>>> $beni->save();
=> "Bandung"
>>> $beni->toArray();
=> [
    "name" => "Beni",
    "phone" => "0214532231",
    "address" => "Bandung"
]
```

Blacklist

Kebalikan dari metode *whitelist*. Dengan teknik ini, kita memberitahu Laravel field apa saja yang **tidak boleh diisi** dengan *mass assignment*. Untuk menggunakan *blacklist* kita dapat menambah attribut `$guarded` di model. Sebagai catatan, kita tidak dapat menggunakan `$fillable` dan `$guarded` secara bersamaan. Mari kita gunakan *blacklist* pada model Customer agar kita tidak dapat menggunakan *mass assignment* pada field phone dan address.

app/Models/Customer.php

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Customer extends Model {

    protected $guarded = ['phone', 'address'];
}
```

Kini ketika kita mencoba mengisi dengan *mass assignment*:

```
>>> $joni = App\Models\Customer::create(['name'=>'Joni', 'phone'=>'0227873435', 'a\\
ddress'=>'Bandung']);
=> <App\Models\Customer #000000004039ff4c0000000042ee4d93> {
    name: "Joni",
    updated_at: "2015-04-29 14:22:18",
    created_at: "2015-04-29 14:22:18",
    id: 20
}
```

Terlihat disini kita tidak dapat mengisi field phone dan address. Tentunya, kita dapat mengisi manual:

```
>>> $joni->phone = '0227873435';
=> "0227873435"
>>> $joni->address = 'Bandung';
=> "Bandung"
>>> $joni->save();
=> true
>>> $joni->toArray();
=> [
    "name"      => "Joni",
    "updated_at" => "2015-04-29 14:24:33",
    "created_at" => "2015-04-29 14:22:18",
    "id"        => 20,
    "phone"      => "0227873435",
    "address"    => "Bandung"
]
```

CRUD Dasar

Proses query dasar menggunakan Eloquent cukup mudah. Mari kita pelajari satu-persatu.

Insert

Untuk melakukan query insert, kita harus membuat object baru dari model yang kita inginkan. Untuk mengisi setiap fieldnya, kita isi sebagai attribut pada model tersebut. Kemudian diakhiri dengan method `save()`. Mari kita coba membuat Customer baru:

```
>>> $customer = new App\Models\Customer;
=> <App\Models\Customer #000000003eb100f10000000625f033e> {}
>>> $customer->name = "Dani";
=> "Dani"
>>> $customer->phone = "081376582342";
=> "081376582342"
>>> $customer->address = "Yogyakarta";
=> "Yogyakarta"
>>> $customer->save();
=> true
```

Kita juga dapat menggunakan method `create()` (dengan *mass assingment*):

```
>>> $customer = App\Models\Customer::create([
    'name'=> 'Budi',
    'phone'=> '087872342123',
    'address'=> 'Makassar'
]);
=> <App\Models\Customer #0000000057a4b8930000000056211b75> {
    name: "Budi",
    phone: "087872342123",
    address: "Makassar",
    updated_at: "2015-04-30 01:42:46",
    created_at: "2015-04-30 01:42:46",
    id: 23
}
```

Silahkan pelajari kembali di topik [mass assingment](#)

Select

Untuk mengambil semua record, kita dapat menggunakan method `all()`:

```
>>> App\Models\Customer::all();
=> <Illuminate\Database\Eloquent\Collection #00000003eb100150000000625f033e> [
    <App\Models\Customer #00000003eb100100000000625f033e> {
        id: 1,
        name: "Prof. Danielle Rath IV",
        phone: "05888575811",
        address: "0727 Lilla Expressway Apt. 327\nCharlesburgh, DC 56460",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00",
        membership_type_id: 3
    },
    <App\Models\Customer #00000003eb1001100000000625f033e> {
        id: 2,
        name: "Zelma Armstrong DVM",
        phone: "805.361.7792",
        address: "7216 Agustin Isle Suite 416\nPort Luciustown, SC 62426",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00",
        membership_type_id: 3
    },
    ....
]
```

Untuk mengambil satu record dengan id tertentu kita dapat menggunakan `find()`:

```
>>> App\Models\Customer::find(3);
=> <App\Models\Customer #00000003eb1001d00000000625f033e> {
    id: 3,
    name: "Fred Pacocha",
    phone: "1-129-353-3925x752",
    address: "96351 Hagenes Hollow Suite 789\nJacobimouth, MD 90758",
    created_at: "0000-00-00 00:00:00",
    updated_at: "0000-00-00 00:00:00",
    membership_type_id: null
}
```

Jika pada method `find()` kita memberikan array berisi id, maka kita akan mendapatkan beberapa record:

```
>>> App\Models\Customer::find([1,3,4]);
=> <Illuminate\Database\Eloquent\Collection #00000006f69d4c10000000019560d8f> [
    <App\Models\Customer #000000006f69d4ca0000000019560d8f> {
        id: 1,
        name: "Dr. Anwar",
        phone: "05888575811",
        address: "0727 Lilla Expressway Apt. 327\nCharlesburgh, DC 56460",
        created_at: "0000-00-00 00:00:00",
        updated_at: "2015-04-30 01:39:38",
        membership_type_id: 3
    },
    <App\Models\Customer #000000006f69d43a0000000019560d8f> {
        id: 3,
        name: "Fred Pacocha",
        phone: "1-129-353-3925x752",
        address: "96351 Hagenes Hollow Suite 789\nJacobimouth, MD 90758",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00",
        membership_type_id: null
    },
    <App\Models\Customer #000000006f69d4c60000000019560d8f> {
        id: 4,
        name: "Ed Dooley",
        phone: "(420)280-4073",
        address: "337 Herman Key Apt. 271\nLake Louvenialand, NV 00812-6311",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00",
        membership_type_id: 2
    }
]
```

Kita juga dapat menggunakan method `findOrFail()` untuk mengambil satu record dengan `id` tertentu. Perbedaan dengan method `find()`, method ini akan memberikan `Illuminate\Database\Eloquent\ModelNotFoundException` ketika record tidak ditemukan. Ini akan sangat bermanfaat ketika kita hendak memberikan respon khusus ketika record tidak ditemukan.

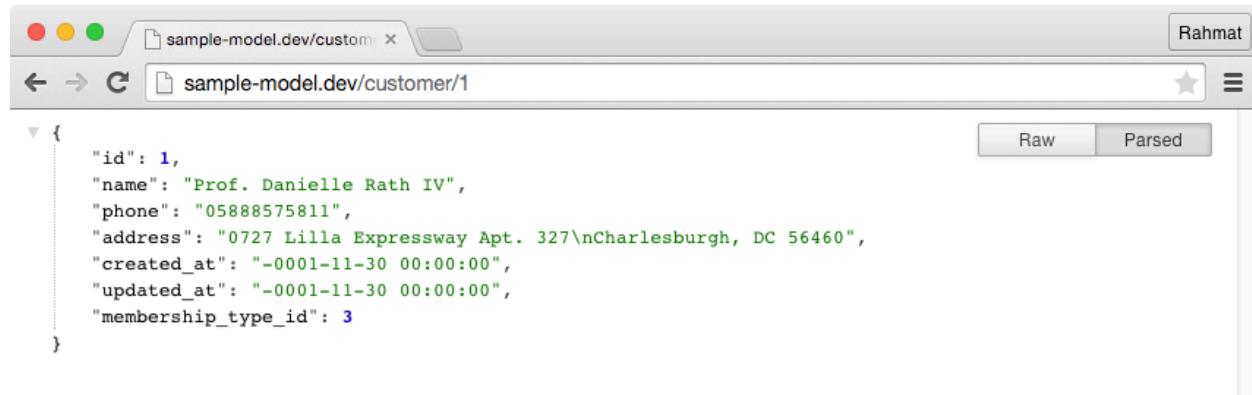
```
>>> App\Models\Customer::findOrFail(390);
Illuminate\Database\Eloquent\ModelNotFoundException with message 'No query results for model [App\Models\Customer].'
```

Mari kita buat route untuk mengetes fitur ini:

app/Http/routes.php

```
.....
Route::get('/customer/{id}', function($id) {
    try {
        $customer = App\Models\Customer::findOrFail($id);
        return $customer;
    } catch (Illuminate\Database\Eloquent\ModelNotFoundException $e) {
        return "Oppsss.. Customer tidak ditemukan";
    }
});
```

Pada route ini kita akan menampilkan isi model Customer berdasarkan parameter Id yang diberikan. Jika tidak ditemukan, maka kita akan menampilkan teks “Oppsss.. Customer tidak ditemukan”.



findOrFail() berhasil menemukan record

Terlihat disini kita berhasil mendapatkan record dari database. Yang menarik adalah output model ini sudah secara otomatis berubah menjadi JSON. Inilah salah satu kelebihan Laravel. Pembahasan lebih lanjut akan kita bahas pada topik [merubah output model ke array / json](#).

Note: Agar tampilan seperti diatas silahkan gunakan [JSON Formatter⁹⁵](#) di Google Chrome

⁹⁵<https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfapjjmafapmmgkkhgoa/related>



findOrFail() gagal menemukan record

Pada contoh ini, kita tidak menemukan record dan berhasil menampilkan teks "Oppsss.. Customer tidak ditemukan" dengan meng-catch exception dari method `findOrFail()`.

Update

Untuk melakukan update prosesnya hampir sama dengan insert. Perbedaannya, di awal tentunya kita harus mencari terlebih dahulu record yang akan kita update.

```
>>> $customer = App\Models\Customer::find(1);
=> <App\Models\Customer #0000000010b95d9f00000000167180a8> {
    id: 1,
    name: "Prof. Danielle Rath IV",
    phone: "05888575811",
    address: "0727 Lilla Expressway Apt. 327\nCharlesburgh, DC 56460",
    created_at: "0000-00-00 00:00:00",
    updated_at: "0000-00-00 00:00:00",
    membership_type_id: 3
}
>>> $customer->name = "Dr. Azhari";
=> "Dr. Azhari"
>>> $customer->save();
=> true
>>> $customer;
=> <App\Models\Customer #0000000010b95d9f00000000167180a8> {
    id: 1,
    name: "Dr. Azhari",
    phone: "05888575811",
    address: "0727 Lilla Expressway Apt. 327\nCharlesburgh, DC 56460",
    created_at: "0000-00-00 00:00:00",
```

```
        updated_at: "2015-04-30 01:35:27",
        membership_type_id: 3
    }
```

Kita juga dapat menggunakan method `update()` (dengan *mass assingment*):

```
>>> $customer = App\Models\Customer::find(1);
=> <App\Models\Customer #000000010b95d9900000000167180a8> {
    id: 1,
    name: "Dr. Azhari",
    phone: "05888575811",
    address: "0727 Lilla Expressway Apt. 327\nCharlesburgh, DC 56460",
    created_at: "0000-00-00 00:00:00",
    updated_at: "2015-04-30 01:35:27",
    membership_type_id: 3
}
>>> $customer->update([
    'name'=> 'Dr. Anwar',
    'phone'=> '022-123123',
    'address'=> 'Bali'
]);
=> true
>>> $customer;
=> <App\Models\Customer #000000010b95d9900000000167180a8> {
    id: 1,
    name: "Dr. Anwar",
    phone: "05888575811",
    address: "Bali",
    created_at: "0000-00-00 00:00:00",
    updated_at: "2015-04-30 01:39:38",
    membership_type_id: 3
}
```

Silahkan pelajari kembali di topik **mass assingment**

Delete

Untuk menghapus model, kita dapat menggunakan method `delete()`:

```
>>> $customer = App\Models\Customer::find(20);
>>> $customer->delete();
=> true
```

Output Model ke Array / JSON

Dengan interface `Jsonable`⁹⁶ Laravel akan secara otomatis merubah output model ketika diakses melalui HTTP Request menjadi JSON. Ini dilakukan dengan menggunakan magic method `__toString()`. Untuk memudahkan pemahaman mari kita pelajari method `__toString()` terlebih dahulu.

Memahami Method `__toString()`

Method `__toString()`⁹⁷ akan dipanggil ketika kita memperlakukan sebuah object sebagai string, misalnya ketika kita mencoba memanggilnya dengan `echo()` atau `return`. Sebagai contoh kita punya class Sekolah seperti berikut:

app/Sekolah.php

```
<?php namespace App;

class Sekolah {
    protected $nama;
    protected $alamat;

    public function __construct($nama, $alamat)
    {
        $this->nama = $nama;
        $this->alamat = $alamat;
    }
}
```

Kita dapat melihat detail object dari class ini dengan `var_dump()`:

⁹⁶ <https://github.com/illuminate/contracts/blob/master/Support/Jsonable.php>

⁹⁷ <http://php.net/manual/en/language.oop5.magic.php#object.tostring>

```
>>> $sekolah = new App\Sekolah("SD Nusa Indah", "Bandung");
=> <App\Sekolah #000000004f711d0c000000001697f6d7> {}
>>> var_dump($sekolah);
class App\Sekolah#633 (2) {
    protected $nama =>
    string(13) "SD Nusa Indah"
    protected $alamat =>
    string(7) "Bandung"
}
=> null
```

Namun, ketika kita mencoba memperlakukannya sebagai string, akan muncul error:

```
>>> $sekolah = new App\Sekolah("SD Nusa Indah", "Bandung");
=> <App\Sekolah #000000004f711d0c000000001697f6d7> {}
>>> echo $sekolah;
PHP error: Object of class App\Sekolah could not be converted to string on line 1
```

Agar kita dapat memperlakukannya sebagai string, kita dapat membuat membuat method `__toString()`:

app/Sekolah.php

```
<?php namespace App;

class Sekolah {
    ...
    public function __toString()
    {
        return "Nama : " . $this->nama . "\n" . "Alamat : " . $this->alamat;
    }
}
```

Pada method diatas, kita menampilkan nama dan alamat sekolah ketika kita memperlakukannya sebagai string:

```
>>> $sekolah = new App\Sekolah("SD Nusa Indah", "Bandung");
=> <App\Sekolah #0000000036e807c10000000024dbd3b6> {}
>>> echo $sekolah;
Nama : SD Nusa Indah
Alamat : Bandung
=> null
```

Jika ingin lebih menarik, kita bahkan dapat membuatnya menjadi JSON setiap kali object Sekolah diperlakukan sebagai string:

app/Sekolah.php

```
<?php namespace App;

class Sekolah {
    ...
    public function __toString()
    {
        return json_encode([ 'nama' => $this->nama, 'alamat' => $this->alamat]);
    }
}
```

Disini kita menggunakan method `json_encode()` untuk membuat JSON dari attribut `nama` dan `alamat`. Sehingga output yang akan kita dapatkan:

```
>>> $sekolah = new App\Sekolah("SD Nusa Indah", "Bandung");
=> <App\Sekolah #00000000421be67f00000000356ca4a0> {}
>>> echo $sekolah;
{"nama": "SD Nusa Indah", "alamat": "Bandung"}
=> null
```

Output ke JSON

Nah, saya rasa cukup penjelasannya. Di Laravel, setiap kali kita mempelakukan sebuah model sebagai string, maka Laravel akan [merubahnya menjadi JSON] (<https://github.com/illuminate/illuminate/blob/3.8.7/src/Illuminate/Foundation/helpers.php#L3387>) dengan memanggil method `toJson()`:

vendor/laravel/framework/src/Illuminate/Database/Eloquent/Model.php

```
.....
public function __toString()
{
    return $this->toJson();
}
```

Kita dapat mengeceknya misalnya dengan menggunakan route berikut:

app/Http/routes.php

```
.....
Route::get('/product/{id}', function($id) {
    return App\Product::find($id);
});
```

Jika kita cek...

**Output model otomatis berupa JSON**

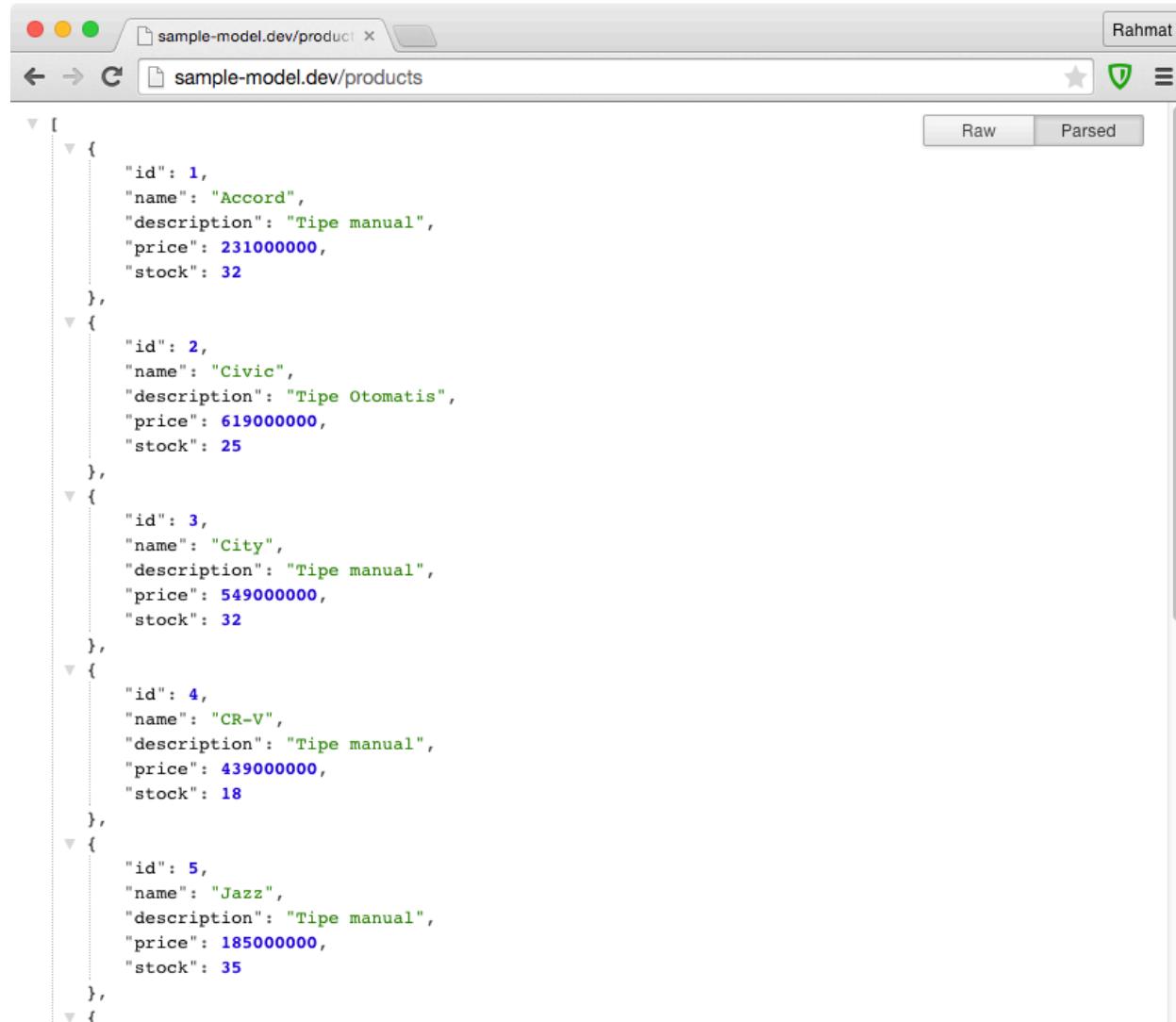
Tentunya, kita juga dapat membuat JSON secara manual dengan memanggil method toJson():

```
>>> $product = App\Product::find(1);
>>> echo $product->toJson();
{"id":1,"name":"Accord","description":"Tipe manual","price":231000000,"stock":32}
=> null
```

Fitur ini akan sangat bermanfaat ketika kita hendak membangun API. Dengan fitur ini, kita tidak perlu membuat json secara manual dengan json_encode(). Kabar baiknya, perubahan menjadi JSON ini juga berlaku untuk koleksi beberapa model, bahkan nilai dari relasinya pun dapat ditampilkan dalam JSON. Misalnya route berikut:

app/Http/routes.php

```
....  
Route::get('/products', function() {  
    return App\Product::all();  
});
```



The screenshot shows a browser window with two tabs: 'sample-model.dev/product' and 'sample-model.dev/products'. The 'sample-model.dev/product' tab is active, displaying a JSON array of product data. The 'Raw' and 'Parsed' buttons are visible at the top right of the JSON viewer. The JSON data is as follows:

```
[  
  {  
    "id": 1,  
    "name": "Accord",  
    "description": "Tipe manual",  
    "price": 231000000,  
    "stock": 32  
  },  
  {  
    "id": 2,  
    "name": "Civic",  
    "description": "Tipe Otomatis",  
    "price": 619000000,  
    "stock": 25  
  },  
  {  
    "id": 3,  
    "name": "City",  
    "description": "Tipe manual",  
    "price": 549000000,  
    "stock": 32  
  },  
  {  
    "id": 4,  
    "name": "CR-V",  
    "description": "Tipe manual",  
    "price": 439000000,  
    "stock": 18  
  },  
  {  
    "id": 5,  
    "name": "Jazz",  
    "description": "Tipe manual",  
    "price": 185000000,  
    "stock": 35  
  }]
```

JSON untuk koleksi model

Penjelasan lebih lanjut, akan kita pelajari di bab **Relationship dalam Eloquent** dan **Collection dengan Eloquent**. Sip.

Output ke Array

Kita juga dapat membuat output model menjadi array asosiatif dengan method `toArray()`:

```
>>> $product = App\Product::find(1);
>>> $product->toArray();
=> [
    "id"          => 1,
    "name"        => "Accord",
    "description" => "Tipe manual",
    "price"       => 231000000,
    "stock"       => 32
]
```

Konfigurasi Lanjutan

Kita dapat menyembunyikan field yang kita inginkan dari method `toJson()` dan `toArray()` ini jika dibutuhkan. Contoh penggunaannya banyak, salah satunya adalah field `password` pada model `User`. Tentunya, kita tidak ingin menampilkan field `password` ini dalam output JSON.

Untuk melakukannya ada dua cara menggunakan attribut `$hidden` atau `$visible`. Yang perlu diperhatikan adalah kita tidak dapat menggunakan keduanya secara bersamaan.

Hidden

Menggunakan teknik ini kita memberitahu Eloquent field apa saja yang **tidak boleh tampil**. Contohnya, kita sembunyikan field `description` dan `stock` pada model **Product**:

app

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model {

    public $timestamps = false;
    protected $hidden = ['description', 'stock'];
}
```

Kita coba lagi pada route yang telah kita buat pada tahap sebelumnya:



Berhasil menyembunyikan field description dan stock

Tentunya, kita masih dapat mengakses kedua field tersebut secara manual:

```
>>> $product = App\Product::find(2);
>>> $product->description;
=> null
>>> $product->stock;
=> 25
```

Visible

Menggunakan teknik ini kita memberitahu Eloquent field apa saja yang ** boleh tampil**. Contohnya, kita hanya menampilkan field `name` dan `price` pada model:

app

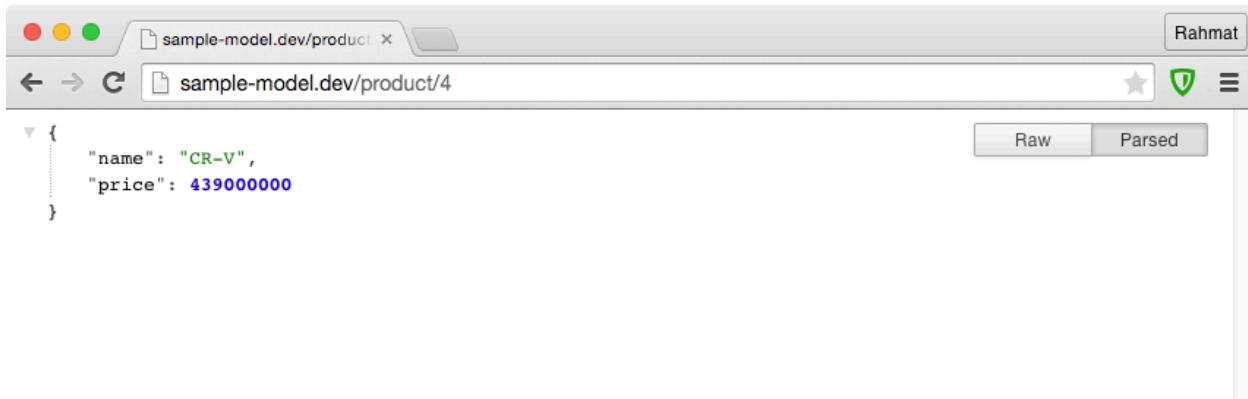
```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model {

    public $timestamps = false;
    protected $visible = ['name', 'price'];
}
```

Jika kita cek kembali:



```
Raw Parsed
{
  "name": "CR-V",
  "price": 43900000
}
```

Hanya menampilkan field `name` dan `price`

Query dalam Eloquent

Dalam membangun query SQL, tentunya kita akan memberikan berbagai kriteria agar hasil yang kita dapatkan sesuai dengan kebutuhan kita. Dalam Eloquent, proses pembuatan query dibagi menjadi 3 bagian:

1. Model
2. Query Constraints
3. Fetch Method

Model adalah model eloquent yang akan kita proses.

Query Constraints adalah kriteria yang akan kita berikan dalam query yang akan kita lakukan. Dengan menambah kriteria ini, hasil query yang kita dapatkan akan sesuai dengan kebutuhan. Contoh constraint yang sederhana adalah `where()`.

Terakhir, Fetch Method digunakan untuk mengeksekusi query dan mendapatkan hasilnya.

Mari kita pelajari struktur query yang paling sederhana yaitu hanya Model dan Fetch Method:

```
Model::fetch();
```

Ini bisa kita lakukan karena Query Constraint itu opsional.

Untuk query yang lebih kompleks, kita dapat menambahkan constraint:

```
Model::constraint()  
    ->fetch();
```

Fetch Method hanya boleh dipanggil satu kali. Sedangkan constraint, dapat kita tambahkan lebih dari satu, tergantung kompleksitas query yang kita bangun:

```
Model::constraint()  
    ->constraint()  
    ->constraint()  
    ->fetch();
```

Karena constraint itu opsional, mari kita pelajari dari Fetch Method.

Fetch Method

Beberapa fetch method sebenarnya telah kita pelajari pada beberapa contoh sebelumnya. Untuk fungsi yang sudah pernah digunakan, tidak akan kita bahas lagi contohnya.

Find

Mencari satu record. Cek contohnya di topik [CRUD Dasar](#).

FindOrFail

Mencari satu record dan mengembalikan Exception jika record tidak ditemukan. Cek contohnya di topik [CRUD Dasar](#).

All

Mengambil semua record. Cek contohnya di topik [CRUD Dasar](#).

First

Mengambil record pertama dari hasil query. Contohnya:

```
>>> App\Product::first();
=> <App\Product #000000006f69d4c80000000019560d8f> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 231000000,
    stock: 32
}
```

FindOrNew

Method ini dapat kita gunakan untuk mencari record berdasarkan id atau membuat record baru jika tidak ditemukan. Untuk memudahkan memahami beberapa contoh kedepan, berikut isi table products:

```
mysql> select * from products;
+----+-----+-----+-----+-----+
| id | name           | description     | price   | stock |
+----+-----+-----+-----+-----+
| 1  | Accord         | Tipe manual    | 231000000 | 32    |
| 2  | Civic          | Tipe Otomatis  | 619000000 | 25    |
| 3  | City           | Tipe manual    | 549000000 | 32    |
| 4  | CR-V           | Tipe manual    | 439000000 | 18    |
| 5  | Jazz            | Tipe manual    | 185000000 | 35    |
| 6  | Freed           | Tipe Otomatis  | 611000000 | 27    |
| 7  | Mobilio         | Tipe manual    | 226000000 | 32    |
| 8  | MacBook Pro 2015 | Retina Display | 240000000 | 18    |
| 9  | iPhone 6        | Warna Hitam   | 130000000 | 23    |
+----+-----+-----+-----+-----+
```

Jika kita menggunakan `findOrNew` untuk id 20, maka Laravel akan membuat record baru:

```
>>> $product = App\Product::findOrNew(20);
=> <App\Product #000000006f69d4c10000000019560d8f> {}
```

Tentunya, record ini belum disimpan, jika ingin menyimpannya gunakan method `save()`. Tentunya, biasanya kita akan mengeset beberapa atributnya:

```
>>> $product->name = "iPad Air";
=> "iPad Air"
>>> $product->save();
=> true
>>> $product->id;
=> 10
```

Terlihat disini, id yang diberikan oleh Laravel adalah 10 (karena itu id selanjutnya setelah record terakhir) meskipun kita mencari id 20.

FirstOrCreate

Method ini akan mencari record dengan kriteria tertentu dan langsung menyimpannya jika tidak ditemukan. Contoh penggunaan di lapangan adalah membuat record preferences / setting untuk seorang user. Ketika seorang user mengunjungi halaman setting, biasanya kita akan mencari record dengan `user_id` user tersebut pada table preferences. Jika tidak ditemukan, kita akan membuat record baru.

Contoh penggunaan method ini pada table Product, misalnya kita akan mencari product `MacBook Air` (yang belum ada) dan langsung menyimpannya ketika tidak ditemukan:

```
>>> $mbAir = App\Product::firstOrCreate([ 'name'=>'MacBook Air']);
Illuminate\Database\Eloquent\MassAssignmentException with message 'name'
```

Oooppss.. kita mendapat error karena belum melakukan setup `mass assignment`. Mari kita setup attribut `$fillable`:

app/Product.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

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

Kita coba lagi (restart tinker):

```
>>> $mbAir = App\Product::firstOrCreate([ 'name'=> 'MacBook Air']);
=> <App\Product #0000000033ebd3da000000001fb02632> {
    name: "MacBook Air"
}
>>> $mbAir->id;
=> 11
```

FirstOrNew

Hampir mirip dengan `firstOrCreate()`, dengan `firstOrNew()` Laravel akan membuat object Eloquent baru. Bedanya, dengan method ini record yang baru dibuat belum akan disimpan sebelum kita memanggil method `save()`.

```
>>> $appleWatch = App\Product::firstOrNew([ 'name'=> 'Apple Watch', 'price'=>500000\0]);
=> <App\Product #00000000386f33c30000000020751fc5> {
    name: "Apple Watch",
    price: 500000
}
>>> $appleWatch->description = 'Sport Blue';
=> "Sport Blue"
>>> $appleWatch->stock = 12;
=> 12
>>> $appleWatch->save();
=> true
```

Get

Ini adalah fetch method yang paling penting. Method ini yang kita gunakan jika telah memberikan query constraint pada Eloquent. **Penggunaannya wajib bersamaan dengan query constraint.** Mari kita gunakan contoh query constraint untuk mengambil product dengan stock lebih dari 30. Untuk memudahkan mengecek, tambahkan field `stock` pada attribut visible di model Product:

App/Product.php

```
....  
protected $visible = [ 'name', 'price', 'stock' ];  
....
```

Kita coba dengan syntax berikut (restart tinker):

```
>>> App\Product::where('stock', '>', 30)->get();
=> <Illuminate\Database\Eloquent\Collection #00000000726257cc00000002b67f510> [
    <App\Product #00000000726257cb000000002b67f510> {
        name: "Accord",
        price: 23100000,
        stock: 32
    },
    <App\Product #00000000726257c8000000002b67f510> {
        name: "City",
        price: 54900000,
        stock: 32
    },
    <App\Product #00000000726257c9000000002b67f510> {
        name: "Jazz",
        price: 18500000,
        stock: 35
    },
    <App\Product #00000000726257d6000000002b67f510> {
        name: "Mobilio",
        price: 22600000,
        stock: 32
    }
]
```

Update

Method `update()` digunakan untuk mengupdate field. **Penggunaannya wajib bersamaan dengan query constraint.** Kita dapat mengupdate beberapa field secara sekali-gus, sebagaimana yang sudah kita pelajari pada topik `mass assigment`, kita harus melakukan konfigurasi attribut `$fillable` di model. Mari kita stock barang dengan id 1,3 dan 4 menjadi 50:

```
>>> App\Product::whereIn('id',[1,3,4])->update(['stock'=>50]);
=> 3
>>> App\Product::whereIn('id',[1,3,4])->get();
=> <Illuminate\Database\Eloquent\Collection #00000000726257e500000002b67f510> [
    <App\Product #00000000726257e3000000002b67f510> {
        name: "Accord",
        price: 23100000,
        stock: 50
    },
    <App\Product #00000000726257d4000000002b67f510> {
```

```
        name: "City",
        price: 549000000,
        stock: 50
    },
    <App\Product #0000000726257e7000000002b67f510> {
        name: "CR-V",
        price: 439000000,
        stock: 50
    }
]
```

Lists

Fiturnya sama dengan penggunaan method lists di query builder, silahkan buka kembali topic [lists di query builder](#).

Pluck

Fiturnya sama dengan penggunaan method pluck di query builder, silahkan buka kembali topic [pluck di query builder](#).

toSql

Jika kita menggunakan method ini, maka Laravel akan menampilkan query SQL yang dieksekusi oleh Eloquent. Sangat bermanfaat ketika melakukan debug.

```
>>> App\Product::whereIn('id',[1,3,4])->where('stock','>',30)->toSql();
=> "select * from `products` where `id` in (?, ?, ?) and `stock` > ?"
```

Tanda tanya pada hasil query diatas adalah parameter yang dibinding oleh Eloquent. Jika ingin menampilkan isi dari parameter tersebut, sebaiknya menggunakan [query logging](#) yang telah kita pelajari di bab sebelumnya.

Delete

Method ini bisa kita gunakan untuk menghapus record sebagaimana telah kita pelajari pada topic [CRUD Dasar](#). **Penggunaannya wajib bersamaan dengan query constraint**. Misalnya kita hapus semua product yang tidak memiliki description:

```
>>> App\Product::whereNull('description')->delete();
=> 2
```

Aggregate

Kita juga dapat menggunakan semua **fungsi aggregate** yang telah kita pelajari pada topik query builder di Eloquent:

```
>>> App\Product::sum('stock');
=> "322"
>>> App\Product::max('stock');
=> 50
>>> App\Product::min('stock');
=> 12
>>> App\Product::avg('stock');
=> "32.2000"
>>> App\Product::count();
=> 10
```

Query Constraint

Dalam mengambil data dari database tentunya kita tidak selalu menginginkan mengambil semua record. Disinilah kita harus menggunakan *query constraint*. Di Eloquent, query constraint menggunakan method `where`. Kabar baiknya, hampir semua method `where` yang kita pelajari di bab query buider dapat kita gunakan di Eloquent. Mari kita buat beberapa contoh.

- Mencari Product dengan stock 20 - 30:

```
>>> App\Product::whereBetween('stock',[20,30])->get();
=> <Illuminate\Database\Eloquent\Collection #000000001167e06e000000003db90f30> [
    <App\Product #000000001167e09f000000003db90f30> {
        name: "Civic",
        price: 61900000,
        stock: 25
    },
    <App\Product #000000001167e09e000000003db90f30> {
        name: "Freed",
        price: 61100000,
        stock: 27
    },
]
```

```
<App\Product #00000001167e08100000003db90f30> {
    name: "iPhone 6",
    price: 13000000,
    stock: 23
}
]
```

- Mencari Product dengan id 3,4 dan 7:

```
>>> App\Product::whereIn('id',[3,4,7])->get();
=> <Illuminate\Database\Eloquent\Collection #00000001167e08700000003db90f30> [
    <App\Product #00000001167e08300000003db90f30> {
        name: "City",
        price: 54900000,
        stock: 50
    },
    <App\Product #00000001167e08200000003db90f30> {
        name: "CR-V",
        price: 43900000,
        stock: 50
    },
    <App\Product #00000001167e08500000003db90f30> {
        name: "Mobilio",
        price: 22600000,
        stock: 32
    }
]
```

Jenis query `where` lainnya dapat dipelajari di topik **Basic Where** pada bab sebelumnya. Query Constraint dapat juga kita gunakan untuk mencari record berdasarkan nilai pada relasinya. Topik ini akan kita pelajari pada bab **Relationship dalam Eloquent**.

Soft Delete

Jika pernah menggunakan CMS seperti WordPress, tentu familiar dengan fitur **Trash**⁹⁸. Fitur ini berfungsi menyimpan semua post yang “dihapus”. Tujuannya, agar kita dapat mengembalikan post tersebut jika dibutuhkan. Tentunya, jika benar-benar ingin dihapus kitapun dapat menghilangkannya dari **Trash**.

⁹⁸ <https://en.support.wordpress.com/trash>

Konfigurasi

Kita dapat mengimplementasikan fitur ini dengan Soft Delete di Laravel. Untuk menggunakannya, ada dua langkah persiapan yang harus kita lakukan:

1. Menambah field `deleted_at` dengan tipe timestamp pada table.
2. Mengaktifkan trait `Illuminate\Database\Eloquent\SoftDeletes` pada model.

Mari kita aktifkan Soft Delete pada model Product. Pertama, kita tambah field `deleted_at`. Ini bisa kita buat manual atau menggunakan `$table->softDeletes()` pada file migration. Mari kita gunakan cara kedua:

/database/migrations/2015_04_09_010128_create_products_table.php

```
....  
public function up()  
{  
    Schema::create('products', function(Blueprint $table)  
    {  
        ....  
        $table->softDeletes();  
    });  
}  
....
```

Langkah kedua, kita trait Soft Delete di model:

app/Product

```
<?php namespace App;  
....  
use Illuminate\Database\Eloquent\SoftDeletes;  
  
class Product extends Model {  
    use SoftDeletes;  
    protected $dates = ['deleted_at'];  
    ....  
}
```

Baris `protected $dates = ['deleted_at']` memberi tahu Eloquent bahwa kita ingin memperlakukan field `deleted_at` sebagai tanggal dan otomatis mengkonversinya ke object Carbon. Fitur ini akan kita pelajari lebih lanjut pada topik [Date Mutator](#)

Terakhir, jalankan `php artisan migrate:refresh --seed` untuk menerapkan perubahan table dan mereset datanya.

Sip.

Penggunaan

Cara kerja soft delete adalah dengan mengisi field `deleted_at` setiap kali kita memanggil method `delete`. Misalnya kita hapus product dengan id 1:

```
>>> $product = App\Product::find(1);
=> <App\Product #000000000817bc0e000000000593e183> {
    name: "Accord",
    price: 330000000,
    stock: 39
}
>>> $product->delete();
```

Kini, jika kita mencoba mencari record tersebut dengan Eloquent, tidak akan ditemukan:

```
>>> $product = App\Product::find(1);
=> null
>>> App\Product::where('name', 'Accord')->get()->toArray();
=> []
```

Soft Delete ini hanya akan berlaku jika kita mengakses database dengan Eloquent. Oleh karena itu, kita tetap dapat mengakses record ini menggunakan Query Buider:

```
>>> DB::table('products')->find(1);
=> <stdClass #000000000817bcf8000000000593e183> {
    id: 1,
    name: "Accord",
    description: "Tipe manual",
    price: 330000000,
    stock: 39,
    deleted_at: "2015-05-05 13:48:26"
}
```

Terlihat disini, field `deleted_at` telah terisi.

Kita juga dapat memaksa Eloquent untuk memasukan record yang telah “dihapus” dengan menggunakan method `withTrashed()`:

```
>>> $product = App\Product::withTrashed()->find(1);
=> <App\Product #000000000817bcf10000000000593e183> {
    name: "Accord",
    price: 330000000,
    stock: 39
}
>>> App\Product::withTrashed()->where('name', 'Accord')->get()->toArray();
=> [
    [
        "name" => "Accord",
        "price" => 330000000,
        "stock" => 39
    ]
]
```

Untuk memastikan bahwa record ini telah “dihapus”, kita dapat menggunakan method `trashed()`. Method ini akan menghasilkan nilai `true` jika record sudah dihapus dan `false` jika belum:

```
>>> $product->trashed();
=> true
```

Untuk mengembalikan record yang telah “dihapus”, kita dapat menggunakan method `restore()`. Tentunya, penggunaan method ini harus bersamaan dengan method `withTrashed()`. Misalnya kita kembalikan Product dengan ID 1:

```
>>> $product = App\Product::withTrashed()->find(1);
=> <App\Product #000000000817bcf60000000000593e183> {
    name: "Accord",
    price: 330000000,
    stock: 39
}
>>> $product->restore();
=> true
```

Setelah kita `restore`, maka field `deleted_at` akan kembali kosong:

```
>>> $product->deleted_at;
=> null
```

Kita juga dapat mengeceknya dengan method `trashed()`:

```
>>> $product->trashed();
=> false
```

Jika kita benar-benar ingin menghapus sebuah record tanpa melewati Trash, kita dapat menggunakan method `forceDelete()`:

```
>>> $product->forceDelete();
Illuminate\Database\QueryException with message 'SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails (`homestead`.`orders_products`, CONSTRAINT `orders_products_product_id_foreign` FOREIGN KEY (`product_id`) REFERENCES `products` (`id`) ON UPDATE CASCADE) (SQL: delete from `products` where `id` = 1)'
```

Ooops... Ternyata masih ada relasi ke table `orders_products` untuk product dengan ID 1. Karena kita belum membuat model / relasi ke table `orders_products`, mari kita hapus dengan Query Builder:

```
>>> DB::table('orders_products')->where('product_id',1)->delete();
=> 1
```

Mari kita coba lagi `forceDelete()`:

```
>>> $product = App\Product::find(1);
>>> $product->forceDelete();
=> null
```

Kini record tersebut benar-benar terhapus dari table. Dapat kita cek dengan mencari record tersebut dengan Query Builder:

```
>>> DB::table('products')->find(1);
=> null
```

Query Scope

Fitur ini sangat bermanfaat untuk membuat query kita lebih elegant. Penggunaan sangat berkaitan erat dengan *business rule* di aplikasi yang kita bangun. Misalnya, pada contoh model Product yang kita bangun misalnya kita memiliki fitur untuk mencari product yang *overstock*. Fitur ini bekerja dengan mencari product yang memiliki stock > 30.

Menggunakan query biasa, ini query yang akan kita lakukan:

```
$products = Product::where('stock', '>', 30)->get();
```

Sedangkan menggunakan query scope, ini query yang akan kita lakukan:

```
$products = Product::overstock()->get();
```

Ada beberapa alasan kenapa kita harus menggunakan query scope:

1. Jika query ini merupakan salah satu *business rule* aplikasi, sangat mungkin query ini akan dipanggil diberbagai tempat di aplikasi. Untuk kriteria sederhana seperti diatas memang cukup mudah untuk menuliskannya. Tapi, jika kriteria / *rule*-nya cukup kompleks (melibatkan berbagai field, bahkan relasi ke table lain), querynya pun akan ikut kompleks. Tentunya, akan sangat merepotkan jika kita harus menulis query lengkap di semua tempat.
2. *Business Rule* dalam sebuah aplikasi rentan dengan perubahan. Terlebih jika kita bekerja di sebuah startup. Untuk aturan *overstock* diatas, bisa saja kriterinya akan berubah suatu saat nanti, mungkin jika stock > 20 sudah dikatakan *overstock*. Tanpa query scope, kita harus merubah semua query untuk rule ini. Dengan menggunakan query scope, kita hanya perlu merubahnya pada model, tanpa perlu merubah query di tempat lain.

Konfigurasi

Untuk membuat query scope, kita perlu menambahkan method baru dengan method baru dengan *prefix* scope dan memberikan parameter \$query. Pada method tersebut kita isi dengan kriteria query yang kita butuhkan. Mari kita buat scope *overstock* pada model Product:

app/Product.php

```
....  
class Product extends Model {  
    ....  
    public function scopeOverstock($query)  
    {  
        return $query->where('stock', '>', 30);  
    }  
}
```

Terlihat disini, kita menggunakan nama method `scopeOverstock`. Penggunaan hurus kapital pada *overstock* disengaja, karena ini merupakan aturan dari Laravel. Pada query scope, kita tidak memanggil method `get()`, karena method ini akan kita panggil ketika membuat query.

Penggunaan

Untuk memanggil scope ini, kita dapat memanggil method `overstock()` pada query di model:

```
>>> App\Product::overstock()->get();
=> <Illuminate\Database\Eloquent\Collection #000000005d41e486000000049ad7777> [
    <App\Product #000000005d41e485000000049ad7777> {
        name: "Civic",
        price: 540000000,
        stock: 32
    },
    <App\Product #000000005d41e4820000000049ad7777> {
        name: "City",
        price: 210000000,
        stock: 37
    }
]
```

Tentunya kita dapat menggabungkan query scope ini dengan query biasa. Misalnya kita cari product yang `overstock` dan memiliki harga diatas 400,000,000:

```
>>> App\Product::overstock()->where('price','>',400000000)->get();
=> <Illuminate\Database\Eloquent\Collection #000000005d41e487000000049ad7777> [
    <App\Product #000000005d41e49d000000049ad7777> {
        name: "Civic",
        price: 540000000,
        stock: 32
    }
]
```

Kita juga dapat memanggil beberapa scope dalam satu query. Misalya kita buat scope `overprice` untuk mencari product dengan harga lebih dari 400,000,000:

App/Product.php

```
....  
class Product extends Model {  
    ....  
    public function scopeOverprice($query)  
    {  
        return $query->where('price', '>', 400000000);  
    }  
}
```

Jika kita gabungkan scope `overstock` dengan scope `overprice` maka ini syntaxnya (restart tinker):

```
>>> App\Product::overstock()->overprice()->get();  
=> <Illuminate\Database\Eloquent\Collection #00000004823ad1300000000112e287d> [  
    <App\Product #00000004823ad1000000000112e287d> {  
        name: "Civic",  
        price: 54000000,  
        stock: 32  
    }  
]
```

Kita juga dapat membuat scope baru dengan menggabungkan beberapa scope. Misalnya kita buat scope `premium` untuk gabungan scope `overstock` dan `overprice`:

App/Product.php

```
....  
class Product extends Model {  
    ....  
    public function scopePremium($query)  
    {  
        return $query->overstock()->overprice();  
    }  
}
```

Mari kita coba (restart tinker):

```
>>> App\Product::premium()->get();
=> <Illuminate\Database\Eloquent\Collection #000000001b9d901d00000000620cc472> [
    <App\Product #000000001b9d901000000000620cc472> {
        name: "Civic",
        price: 54000000,
        stock: 32
    }
]
```

Dynamic Scope

Kita juga dapat membuat scope yang menerima parameter. Misalnya kita buat query `level` product. Dengan kriteria:

- `lux` untuk product dengan harga lebih dari 500,000,000
- `mid` untuk product dengan harga 300,000,000 hingga 500,000,000
- `entry` untuk product dengan harga kurang dari 300,000,000

Untuk membuat scope ini dengan dynamic scope, kita dapat menggunakan syntax berikut pada model Product:

App/Product.php

```
<?php namespace App;
.....
class Product extends Model {
    .....
    public function scopeLevel($query, $parameter)
    {
        switch ($parameter) {
            case 'lux':
                return $query->where('price', '>', 500000000);
                break;
            case 'mid':
                return $query->whereBetween('price', [30000000, 500000000]);
                break;
            case 'entry':
                return $query->where('price', '<', 30000000);
                break;
            default:
                return $query;
                break;
        }
    }
}
```

```

        }
    }
}
```

Terlihat diatas kita menggunakan statement switch case untuk menentukan kriteria untuk tiap level product. Pada statement `default`, kita tidak melakukan perubahan pada query.

Mari kita cek scope ini untuk mencari product dengan level `lux`:

```

>>> App\Product::level('lux')->get();
=> <Illuminate\Database\Eloquent\Collection #00000003e21c36a00000000752224e9> [
    <App\Product #00000003e21c36f00000000752224e9> {
        name: "Civic",
        price: 54000000,
        stock: 32
    },
    <App\Product #00000003e21c36e00000000752224e9> {
        name: "CR-V",
        price: 68000000,
        stock: 28
    },
    <App\Product #00000003e21c37100000000752224e9> {
        name: "Freed",
        price: 60600000,
        stock: 21
    }
]
```

Product dengan level `mid`:

```

>>> App\Product::level('mid')->get();
=> <Illuminate\Database\Eloquent\Collection #00000003e21c36900000000752224e9> [
    <App\Product #00000003e21c37000000000752224e9> {
        name: "Jazz",
        price: 38500000,
        stock: 27
    },
    <App\Product #00000003e21c37300000000752224e9> {
        name: "Mobilio",
        price: 40600000,
        stock: 21
    }
]
```

```
    }
]
```

Dan product dengan level entry:

```
>>> App\Product::level('entry')->get();
=> <Illuminate\Database\Eloquent\Collection #000000003e21c36700000000752224e9> [
    <App\Product #000000003e21c36d00000000752224e9> {
        name: "City",
        price: 21000000,
        stock: 37
    }
]
```

Jika kita memberikan parameter yang salah, Eloquent akan memberikan hasil semua recordnya:

```
>>> App\Product::level('mewah')->get();
=> <Illuminate\Database\Eloquent\Collection #000000005e3c139d000000003c733cd6> [
    <App\Product #000000005e3c1382000000003c733cd6> {
        name: "Civic",
        price: 54000000,
        stock: 32
    },
    <App\Product #000000005e3c1381000000003c733cd6> {
        name: "City",
        price: 21000000,
        stock: 37
    },
    <App\Product #000000005e3c1380000000003c733cd6> {
        name: "CR-V",
        price: 68000000,
        stock: 28
    },
    <App\Product #000000005e3c1387000000003c733cd6> {
        name: "Jazz",
        price: 38500000,
        stock: 27
    },
    <App\Product #000000005e3c1386000000003c733cd6> {
        name: "Freed",
        price: 60600000,
```

```
    stock: 21
},
<App\Product #000000005e3c1385000000003c733cd6> {
    name: "Mobilio",
    price: 40600000,
    stock: 21
}
]
```

Global Scope

Adakalanya kita membutuhkan scope yang akan digunakan pada berbagai model. Misalnya, ketika kita memiliki fitur `published` pada model `Product`. Fitur ini bekerja dengan mengecek kolom `published`, jika kolom ini terisi maka model tersebut dianggap sudah `di-publish`. Jika masih kosong, maka ia akan dianggap masih `draft`. Untuk membuat fitur ini, mari kita tambah kolom `published` pada table `products`:

database/migrations/2015_04_09_010128_create_products_table.php

```
<?php
...
class CreateProductsTable extends Migration {
    ...
    public function up()
    {
        Schema::create('products', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('name');
            $table->string('description')->nullable();
            $table->integer('price')->unsigned();
            $table->integer('stock')->unsigned();
            $table->boolean('published')->nullable();
            $table->softDeletes();
        });
    }
}
```

Ubah juga isi `ProductsTableSeeder` agar mengisi field `published` secara acak:

database/seeds/ProductsTableSeeder.php

```
<?php

use Illuminate\Database\Seeder;

class ProductsTableSeeder extends Seeder {

    public function run()
    {
        $products = ["Accord", "Civic", "City", "CR-V", "Jazz", "Freed", "Mobilio"];
        $descriptions = ["Tipe manual", "Tipe Otomatis"];

        foreach ($products as $product) {
            DB::insert('insert into products (name, description, price, stock,published) values (:name, :description, :price, :stock, :published)', [
                'name' => $product,
                'description' => $descriptions[rand(0,1)],
                'price' => rand(100,800) * 1000000,
                'stock' => rand(10,40),
                'published' => rand(0,1)
            ]);
        }
    }

    $this->command->info('Berhasil menambah mobil!');
}

}
```

Jalankan `php artisan migrate:refresh --seed` untuk menerapkan perubahan ini.

Terdapat dua macam global scope yang bisa kita buat, manual dan otomatis. Mari kita pelajari keduanya.

Manual

Penggunaan scope manual sama seperti scope pada umumnya, dimana kita harus memanggil method scope untuk mengaktifkannya. Untuk membuatnya, kita tidak akan menulis scope langsung pada model Product. Kita akan menggunakan trait untuk menyimpan scope `published`. Misalnya, kita buat di `App\Scope` dengan nama `PublishedTrait`:

App/Scope/PublishedTrait.php

```
<?php namespace App\Scope;

trait PublishedTrait {
    public function scopePublished($query)
    {
        return $query->where('published', 1);
    }
}
```

Untuk mengimplementasikan pada model Product, kita `use` trait ini:

app/Product.php

```
<?php namespace App;
...
class Product extends Model {
    use \App\Scope\PublishedTrait;
}

```

Sebelum menggunakan trait ini, mari kita cek isi table products:

```
mysql> select name,published from products;
+-----+-----+
| name   | published |
+-----+-----+
| Accord |      1 |
| Civic  |      1 |
| City   |      0 |
| CR-V   |      1 |
| Jazz   |      0 |
| Freed  |      0 |
| Mobilio|      1 |
+-----+-----+
```

Mari kita coba mencari product yang sudah di-publish:

```
>>> App\Product::published()->get();
=> <Illuminate\Database\Eloquent\Collection #000000046974dc2000000016cb5d90> [
    <App\Product #000000046974dc50000000016cb5d90> {
        name: "Accord",
        price: 17200000,
        stock: 37
    },
    <App\Product #000000046974dc60000000016cb5d90> {
        name: "Civic",
        price: 26500000,
        stock: 34
    },
    <App\Product #000000046974dc70000000016cb5d90> {
        name: "CR-V",
        price: 23100000,
        stock: 20
    },
    <App\Product #000000046974dd80000000016cb5d90> {
        name: "Mobilio",
        price: 34200000,
        stock: 28
    }
]
```

Sip, berhasil.

Kelebihan dari menggunakan Global Scope adalah kita tidak perlu menulis kembali scope ini jika ingin mengimplementasikan pada model lain. Misalnya jika ingin kita implementasikan pada model Post. Karena, model ini belum ada, mari kita buat:

```
vagrant@homestead:~/Code/sample-model$ php artisan make:model Post
Model created successfully.
Created Migration: 2015_04_10_002726_create_posts_table
```

Dengan struktur migration sebagai berikut:

app/database/migrations/2015_04_10_002726_create_posts_table.php

```
<?php

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

class CreatePostsTable extends Migration {

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('title');
            $table->text('content');
            $table->boolean('published');
            $table->timestamps();
        });
    }

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

Terlihat disini, kita hanya membuat 3 field title, content dan migration.

Pada model Post kita akan mengaktifkan mass assignment untuk tiga field diatas:

app/Post.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model {
    protected $fillable = ['title', 'content', 'published'];
}
```

Mari kita buat sample data untuk table post ini pada file PostsTableSeeder:

database/seeds/PostsTableSeeder.php

```
<?php

use Faker\Factory as Faker;
use Illuminate\Database\Seeder;

class PostsTableSeeder extends Seeder {

    public function run()
    {
        $faker = Faker::create();
        for ($i=0; $i <= 10 ; $i++) {
            App\Post::insert([
                'title' => $faker->sentence,
                'content' => $faker->paragraph,
                'published' => rand(0,1)
            ]);
        }
    }
}
```

Terlihat pada file ini, kita menggunakan Faker untuk memberikan title dan content Post secara acak. Kita juga mengeset isian published secara acak dengan fungsi rand(0,1). Mari kita tambahkan seeder ini pada DatabaseSeeder:

database/seeds/DatabaseSeeder.php

```
....  
public function run()  
{  
    ....  
    $this->call('PostsTableSeeder');  
}  
....
```

Terakhir, jalankan `php artisan migrate:refresh --seed` untuk penambahan table Posts dan sample datanya. Setelah perintah tersebut dijalankan, pastikan table posts sudah memiliki isi:

| id title published |
|--------------------------------------|
| 1 Nihil adipisci ducimus e... 0 |
| 2 Id enim repellat distinc... 1 |
| 3 Inventore vitae nihil mi... 0 |
| 4 Ut esse quos voluptatem ... 0 |
| 5 Qui qui blanditiis nihil... 1 |
| 6 Doloribus quos mollitia ... 0 |
| 7 Et eum veritatis tempor... 1 |
| 8 Et labore omnis dolorum ... 1 |
| 9 Neque dolores fugit unde... 1 |
| 10 Rerum aliquam omnis enim... 0 |
| 11 Iusto voluptates assumen... 0 |

Untuk menambahkan scope `published` pada model ini, kita cukup menambahkan `PublishedTrait`:

app/Product.php

```
<?php namespace App;  
...  
class Post extends Model {  
    use \App\Scope\PublishedTrait;  
    ...  
}
```

Dan scope published-pun akan bisa digunakan pada model Post:

```
>>> App\Post::published()->get();  
=> <Illuminate\Database\Eloquent\Collection #00000002647435300000000233fde26> [  
    <App\Post #00000002647435e00000000233fde26> {  
        id: 2,  
        title: "Id enim repellat distinctio veritatis rerum et.",  
        content: "Nam hic nobis totam necessitatibus odio ...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000002647435f00000000233fde26> {  
        id: 5,  
        title: "Qui qui blanditiis nihil rem quam.",  
        content: "Et tempora ut unde tenetur laudantium ...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000002647435c00000000233fde26> {  
        id: 7,  
        title: "Et eum veritatis temporibus corrupti et.",  
        content: "Illum sint quos rerum dolores consequ...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000002647435d00000000233fde26> {  
        id: 8,  
        title: "Et labore omnis dolorum dolor vel laborum.",  
        content: "Magnam nisi fuga odio expedita vitae dolor...",  
        published: 1,
```

```
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000002647435a00000000233fde26> {
        id: 9,
        title: "Neque dolores fugit unde rerum necessitatibus.",
        content: "Quia nemo quia libero et ipsam sit qui...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]
```

Ketika kita ingin menambah scope yang masih berhubungan, misalnya scope `draft` untuk melihat semua record yang belum di publish, kita cukup menambahkan satu kali di `PublishedTrait`:

app/Scope/PublishedTrait.php

```
.....
trait PublishedTrait {
    .....
    public function scopeDraft($query)
    {
        return $query->where('published', '!=', 1);
    }
}
```

Kini, scope `draft` dapat digunakan pada model `Product` maupun `Post`:

```
>>> App\Product::draft()->get();
=> <Illuminate\Database\Eloquent\Collection #00000000157d21360000000034ce2a75> [
    <App\Product #00000000157d21330000000034ce2a75> {
        name: "Accord",
        price: 479000000,
        stock: 14
    },
    <App\Product #00000000157d21320000000034ce2a75> {
        name: "Civic",
        price: 773000000,
        stock: 16
    },
    .....
]
```

```
>>> App\Post::draft()->get();
=> <Illuminate\Database\Eloquent\Collection #00000000157d21350000000034ce2a75> [
    <App\Post #00000000157d213b0000000034ce2a75> {
        id: 1,
        title: "Nihil adipisci ducimus et quas molestias temporibus sint.",
        content: "Veniam enim quaerat et quidem. Sunt culpa quia ex velit odi\ t ratione quisquam. Nihil omnis eum aut quibusdam voluptatem aliquam. Minima qui\ molestias et hic.",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #00000000157d21380000000034ce2a75> {
        id: 3,
        title: "Inventore vitae nihil minus.",
        content: "Cumque non mollitia officiis enim fugit incidunt. Cum aliqu\ am et aperiam in ab unde maiores quo. Quia temporibus quod et veritatis aut.",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    ...
]
```

Otomatis

Menggunakan global scope otomatis, kita tidak harus selalu memanggil method scope untuk mengaktifkannya. Di Laravel, fitur `soft delete` dibuat dengan teknik ini. Ketika kita melakukan query pada model yang telah mengaktifkan soft delete, secara otomatis scope tersebut terpanggil dan kita mendapatkan semua record yang belum “dihapus”.

Memahami Alur Query Builder pada Eloquent

Untuk menggunakan Global Scope ini, kita wajib memahami apa yang terjadi dibalik layar ketika sebuah query dilakukan di Eloquent. Mari kita praktikan pada model Post. Untuk memudahkan pemahaman, ubah dulu model Post menjadi seperti berikut:

App/Post.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model {
    protected $fillable = ['title', 'content', 'published'];
}
```

Setiap kali kita menambahkan **query constraint** pada model, maka Laravel akan secara otomatis merubah hasilnya menjadi instance dari `Illuminate\Database\Eloquent\Builder` (sebelum kita memanggil **fetch method** pada model ini). Ini dapat kita lihat dari contoh query berikut:

```
>>> $query = App\Post::where('id',1);
=> <Illuminate\Database\Eloquent\Builder #00000006f5b10ba00000000269be990> {}
```

Dari instance query builder ini, kita dapat melakukan beberapa hal menarik. Misalnya, melihat opsi apa saja yang telah diset pada query dengan method `getQuery()`:

```
>>> $query = App\Post::where('id',1);
=> <Illuminate\Database\Eloquent\Builder #00000006f5b10ba00000000269be990> {}
>>> $query->getQuery();
=> <Illuminate\Database\Query\Builder #00000006f5b10b000000000269be990> {
    aggregate: null,
    columns: null,
    distinct: false,
    from: "posts",
    joins: null,
    wheres: [
        [
            "type"      => "Basic",
            "column"    => "id",
            "operator"  => "=",
            "value"     => 1,
            "boolean"   => "and"
        ]
    ],
    groups: null,
    havings: null,
```

```
orders: null,  
limit: null,  
offset: null,  
unions: null,  
unionLimit: null,  
unionOffset: null,  
unionOrders: null,  
lock: null  
}
```

Terlihat disini, banyak opsi yang tersedia. Beberapa telah terisi misalnya opsi `from` yang berisi nama table yang akan kita gunakan. Fokus kita saat ini adalah memperhatikan opsi `wheres`:

```
....  
wheres: [  
  [  
    "type"      => "Basic",  
    "column"     => "id",  
    "operator"   => "=",  
    "value"      => 1,  
    "boolean"   => "and"  
  ]  
,  
....
```

Opsi ini berisi array untuk semua query constraint yang telah kita tambahkan. Pada contoh diatas, hanya terdapat satu elemen yaitu query untuk mencari record dengan id 1.

Pada saat yang sama, Eloquent juga membuat binding untuk query ini. Kita dapat mengeceknya dengan menggunakan method `getRawBindings()` pada instance `Illuminate\Database\Query\Builder` hasil method `getQuery()`:

```
>>> $query->getQuery()->getRawBindings();
=> [
    "select" => [],
    "join"   => [],
    "where"  => [
        1
    ],
    "having" => [],
    "order"  => []
]
```

Terlihat disini, isian `where` berisi `value` pada array `wheres` yang ada pada hasil perintah `getQuery()`. Dalam hal ini, pada array `where` yang pertama kita mencari record dengan ID 1 (cek isian `value`), sehingga binding `where` pertama ini juga berisi 1.

Jika kita jalankan fetch method `get()` pada query ini, maka kita akan mendapatkan record tersebut:

```
>>> $query->get();
=> <Illuminate\Database\Eloquent\Collection #000000006f5b10ab00000000269be990> [
    <App\Post #000000006f5b10b200000000269be990> {
        id: 1,
        title: "Nihil adipisci ducimus et quas molestias temporibus sint.",
        content: "Veniam enim quaerat et quidem...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]
```

Setiap kali kita menambahkan query constraint, maka isian opsi `wheres` ini akan bertambah. Misalnya kita tambah constraint untuk mengecek record yang sudah publish:

```
>>> $query->where('published',1);
=> <Illuminate\Database\Eloquent\Builder #000000006f5b10ba00000000269be990> {}
```

Maka isian `wheres` pun akan bertambah:

```
>>> $query->getQuery()->wheres;
=> [
    [
        "type"      => "Basic",
        "column"    => "id",
        "operator"  => "=",
        "value"     => 1,
        "boolean"   => "and"
    ],
    [
        "type"      => "Basic",
        "column"    => "published",
        "operator"  => "=",
        "value"     => 1,
        "boolean"   => "and"
    ]
]
```

Binding untuk `where` pun akan bertambah:

```
>>> $query->getQuery()->getRawBindings()['where'];
=> [
    1,
    1
]
```

Disini terlihat isian untuk binding kedua tetap 1. Karena, kita mencari field `published` yang berisi nilai 1.

Jika kita coba method `get()`, maka tidak akan ada record yang muncul (karena record dengan ID 1 tidak memiliki isian `published` 1):

```
>>> $query->get();
=> <Illuminate\Database\Eloquent\Collection #000000006f5b10b600000000269be990> []
```

Menariknya, karena isian dari opsi `wheres` ini adalah array biasa, kita dapat menghapusnya dengan menggunakan method `unset()`. Misalnya, kita hapus constraint untuk `published`:

```
>>> unset($query->getQuery()->wheres[1]);
=> null
```

Disini, kita memanggil index 1 pada array `wheres` yang merupakan constraint untuk `publish`. Kita dapat mengecek apakah constraint ini telah terhapus dengan method `getQuery()`:

```
>>> $query->getQuery();
=> <Illuminate\Database\Query\Builder #00000006f5b10b000000000269be990> {
    ....
    wheres: [
        [
            "type"      => "Basic",
            "column"    => "id",
            "operator"  => "=",
            "value"     => 1,
            "boolean"   => "and"
        ]
    ],
    ....
}
```

Terlihat diatas, isian `wheres` untuk constraint `publish` telah terhapus. Kini, ketika kita memanggil method `get()`, kita akan mendapatkan record dengan ID 1 lagi:

```
>>> $query->get();
=> <Illuminate\Database\Eloquent\Collection #00000006f5b10a200000000269be990> []
```

Oopppsss... record nya tidak muncul..

Ini terjadi karena masih terdapat binding untuk field `publish`. Bisa kita cek dengan perintah ini:

```
>>> $query->getQuery()->getRawBindings()['where'];
=> [
    1,
    1
]
```

Terlihat disini, masih terdapat binding kedua untuk field `publish` yang telah kita buat pada query sebelumnya. Untuk menghapusnya, kita harus me-*replace* binding ini dengan method `setBindings()`. Misalnya dengan cara berikut:

```
>>> $bindings = $query->getQuery()->getRawBindings()['where'];
>>> unset($bindings[1]);
=> null
>>> $query->getQuery()->setBindings($bindings);
```

Pada baris pertama kita mengambil semua binding yang sudah ada. Di baris selanjutnya, kita menghapus binding untuk index 1 (ini binding untuk constraint published). Pada baris terakhir, kita me-replace binding dengan method `setBindings()`. Kini query kita akan berhasil lagi:

```
>>> $query->get();
=> <Illuminate\Database\Eloquent\Collection #00000006f5b10ab00000000269be990> [
    <App\Post #00000006f5b10b200000000269be990> {
        id: 1,
        title: "Nihil adipisci ducimus et quas molestias temporibus sint.",
        content: "Veniam enim quaerat et quidem...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]
```

Fiuh.. Penjelasan yang cukup panjang.

Setelah memahami cara kerja query di Eloquent ini, baru kita akan memahami bagaimana menggunakan global scope otomatis.

Contoh Implementasi Global Scope Otomatis

Pada contoh kasus yang telah kita buat pada tahap sebelumnya, kita dapat menggunakan fitur ini agar Laravel secara otomatis mengaktifkan scope `published` sehingga record yang muncul hanya yang sudah di-publish.

Untuk membuatnya ada 3 langkah yang harus kita lakukan:

1. Membuat scope (kita beri nama `PublishedScope`) yang mengimplementasikan `Illuminate\Database\Eloquent\ScopeInterface`.
2. Membuat trait (kita beri nama `Published`) yang akan melakukan “boot” ke scope tadi dan mengimplementasikan method yang kita butuhkan.
3. Menggunakan trait yang kita buat (`Published`) pada model-model yang akan menggunakan global scope ini.

Langkah pertama, kita buat `PublishedScope` yang mengimplementasikan `Illuminate\Database\Eloquent\ScopeInterface`. Mari kita buat pada folder `App\Scope`:

AppScopePublishedScope.php

```
<?php namespace App\Scope;

use Illuminate\Database\Eloquent\ScopeInterface;

class PublishedScope implements ScopeInterface {

}
```

Interface ini akan mengharuskan kita untuk membuat method `apply()` dan `remove()`, keduanya menerima parameter berupa `\Illuminate\Database\Eloquent\Builder` dan `\Illuminate\Database\Eloquent\Model`.

Pada method `apply()`, kita harus membuat syntax untuk menambahkan kriteria scope yang kita inginkan. Pada kasus ini, kita akan memfilter record dengan isian field `published` 1:

AppScopePublishedScope.php

```
<?php namespace App\Scope;
...

use Illuminate\Database\Eloquent\Builder;
use Illuminate\Database\Eloquent\Model;

class PublishedScope implements ScopeInterface {
    public function apply(Builder $builder, Model $model)
    {
        $builder->where('published', 1);
    }
}
```

Pada method `remove()`, kita akan membuat syntax yang akan menghapus kriteria query yang telah kita buat pada method `apply()`:

AppScopePublishedScope.php

```
<?php namespace App\Scope;  
....  
  
class PublishedScope implements ScopeInterface {  
    ....  
    public function remove(Builder $builder, Model $model)  
    {  
        $query = $builder->getQuery();  
  
        foreach ((array) $query->wheres as $key => $where)  
        {  
            // mengecek apakah opsi where ini merupakan constraint untuk published  
            if ($where['type'] == 'Basic' && $where['column'] == 'published' && $where['value'] == 1) {  
                // menghapus query dari opsi where  
                unset($query->wheres[$key]);  
                $query->wheres = array_values($query->wheres);  
                // menghapus binding  
                $bindings = $query->getRawBindings()['where'];  
                unset($bindings[$key]);  
                $query->setBindings($bindings);  
            }  
        }  
        return $query;  
    }  
}
```

Syntax yang kita gunakan disini, sesuai dengan yang kita pelajari pada [topik sebelumnya](#). Disini, kita melakukan *looping* pada setiap opsi where yang telah dibuat. Kemudian mengecek apakah opsi where tersebut sesuai dengan kriteria yang telah kita tetapkan pada method `apply`. Jika sesuai (bertipe `Basic`, kolom `published` dan isian 1), maka kita hapus opsi where tersebut dan menghapus bindingnya.

Jika kebingungan membaca syntax ini, silahkan pelajari kembali [topik sebelumnya](#).

Langkah kedua, membuat trait `Published` yang akan melakukan “boot” pada `PublishedScope`. Kita simpan pada folder `App\Scope`:

AppScopePublished.php

```
<?php namespace App\Scope;

trait Published {

    public static function bootPublished()
    {
        static::addGlobalScope(new PublishedScope);
    }

}
```

Untuk melakukan “boot”, kita harus membuat static method dengan nama `bootNamaTrait`. Terlihat diatas, karena nama trait nya `Published` maka kita memberi nama `bootPublished`. Didalamnya, kita harus menggunakan syntax:

```
static::addGlobalScope(new NamaScope);
```

Karena disini kita akan melakukan boot terhadap `PublishedScope` maka kita tulis:

```
static::addGlobalScope(new PublishedScope);
```

Tentunya, jika scope yang akan kita “boot” memiliki namespace yang berbeda, kita harus menulis lengkap dengan namespace nya.

Langkah ketiga, kita gunakan trait `Published` pada model-model yang akan menggunakannya. Mari kita tambahkan pada model `Post`:

AppPost.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model {
    use \App\Scope\Published;
    protected $fillable = ['title', 'content', 'published'];
}
```

Kini, setiap kali kita melakukan query terhadap model `Post`, scope `published` akan dimasukan ke dalam query. Mari kita coba:

```
>>> App\Post::all();
=> <Illuminate\Database\Eloquent\Collection #000000006db8a4da0000000062f7bb77> [
    <App\Post #000000006db8a4d60000000062f7bb77> {
        id: 2,
        title: "Id enim repellat distinctio veritatis rerum et.",
        content: "Nam hic nobis totam necessitatibus...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4fe0000000062f7bb77> {
        id: 5,
        title: "Qui qui blanditiis nihil rem quam.",
        content: "Et tempora ut unde tenetur...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4ca0000000062f7bb77> {
        id: 7,
        title: "Et eum veritatis temporibus corrupti et.",
        content: "Illum sint quos rerum dolores conse...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4dc0000000062f7bb77> {
        id: 8,
        title: "Et labore omnis dolorum dolor vel laborum.",
        content: "Magnam nisi fuga odio expedita vitae...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4fb0000000062f7bb77> {
        id: 9,
        title: "Neque dolores fugit unde rerum...",
        content: "Quia nemo quia libero et ipsam sit qui. Non labore deleniti s\\
int.",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
```

```
    }
]
```

Terlihat disini, kita hanya menampilkan record yang sesuai dengan kriteria pada method `apply()` di `PublishedScope`.

Kita juga dapat menonaktifkan secara temporary semua global scope pada model menggunakan method `newQueryWithoutScopes()`. Ini akan berguna untuk mengecek method `remove()` yang kita buat pada `PublishedScope` telah berjalan.

```
>>> $p = new App\Post;
=> <App\Post #00000006db8a4dd0000000062f7bb77> {}
>>> $q = $p->newQueryWithoutScopes();
=> <Illuminate\Database\Eloquent\Builder #00000006db8a4dc0000000062f7bb77> {}
>>> $q->get();
=> <Illuminate\Database\Eloquent\Collection #00000006db8a4db0000000062f7bb77> [
    <App\Post #00000006db8a4c9000000062f7bb77> {
        id: 1,
        title: "Nihil adipisci ducimus et quas molestias...",
        content: "Veniam enim quaerat et quidem. Sunt...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #00000006db8a4ff0000000062f7bb77> {
        id: 2,
        title: "Id enim repellat distinctio veritatis rerum et.",
        content: "Nam hic nobis totam necessitatibus odio...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #00000006db8a4d20000000062f7bb77> {
        id: 3,
        title: "Inventore vitae nihil minus.",
        content: "Cumque non mollitia officiis enim fugit...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #00000006db8a4c00000000062f7bb77> {
        id: 4,
        title: "Ut esse quos voluptatem debitisi."
    }
]
```

```
        content: "Dolores quidem necessitatibus impedit...",  
        published: 0,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4cf0000000062f7bb77> {  
        id: 5,  
        title: "Qui qui blanditiis nihil rem quam.",  
        content: "Et tempora ut unde tenetur laudantiu...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4fd0000000062f7bb77> {  
        id: 6,  
        title: "Doloribus quos mollitia vel provident eum.",  
        content: "Et omnis qui magnam quam quia odio...",  
        published: 0,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4df0000000062f7bb77> {  
        id: 7,  
        title: "Et eum veritatis temporibus corrupti et.",  
        content: "Illum sint quos rerum dolores conse...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4cb0000000062f7bb77> {  
        id: 8,  
        title: "Et labore omnis dolorum dolor vel laborum.",  
        content: "Magnam nisi fuga odio expedita vita...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4d00000000062f7bb77> {  
        id: 9,  
        title: "Neque dolores fugit unde rerum...",  
        content: "Quia nemo quia libero et ipsam sit...",  
        published: 1,
```

```

        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4270000000062f7bb77> {
        id: 10,
        title: "Rerum aliquam omnis enim doloribus quaerat sint.",
        content: "Dicta ab ipsa sunt voluptas recusandae...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4de0000000062f7bb77> {
        id: 11,
        title: "Iusto voluptates assumenda esse.",
        content: "Nobis voluptas eveniet doloremque...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]

```

Terlihat disini, kita berhasil menampilkan semua record meskipun tidak memiliki isian published 1. Menandakan method `remove` pada `PublishedScope` telah berfungsi dengan benar.

Sudah sangat umum ketika membuat global scope, dibuat pula method untuk menon-aktifkan global scope tersebut sementara. Pada fitur `softDelete`, Laravel membuat method `withTrashed` untuk memasukan record yang sudah “dihapus” pada table.

Pada contoh kasus yang kita miliki, mari kita buat method `withDrafts` untuk memasukan record yang belum di `publish` ke dalam hasil query:

AppScopePublished.php

```

<?php namespace App\Scope;

trait Published {
    ...
    public static function withDrafts()
    {
        return with(new static)->newQueryWithoutScope(new PublishedScope);
    }
}

```

Untuk menghilangkan global scope ini, kita menggunakan method `newQueryWithoutScope()`. Berbeda dengan method `newQueryWithoutScopes()` yang akan menonaktifkan semua global scope, method ini hanya akan menonaktifkan satu global scope. Dia bekerja dengan cara memanggil method `remove` pada scope yang kita jadikan parameter (`PublishedScope`).

```
>>> App\Post::withDrafts()->get();
=> <Illuminate\Database\Eloquent\Collection #000000006db8a4db0000000062f7bb77> [
    <App\Post #000000006db8a4c90000000062f7bb77> {
        id: 1,
        title: "Nihil adipisci ducimus et quas molestias...",
        content: "Veniam enim quaerat et quidem. Sunt...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4ff0000000062f7bb77> {
        id: 2,
        title: "Id enim repellat distinctio veritatis rerum et.",
        content: "Nam hic nobis totam necessitatibus odio...",
        published: 1,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4d20000000062f7bb77> {
        id: 3,
        title: "Inventore vitae nihil minus.",
        content: "Cumque non mollitia officiis enim fugit...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4c00000000062f7bb77> {
        id: 4,
        title: "Ut esse quo voluptatem debitis.",
        content: "Dolores quidem necessitatibus impedit...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000006db8a4cf0000000062f7bb77> {
        id: 5,
        title: "Qui qui blanditiis nihil rem quam."
    }
]
```

```
        content: "Et tempora ut unde tenetur laudantiu...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4fd0000000062f7bb77> {  
        id: 6,  
        title: "Doloribus quos mollitia vel provident eum.",  
        content: "Et omnis qui magnam quam quia odio...",  
        published: 0,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4df0000000062f7bb77> {  
        id: 7,  
        title: "Et eum veritatis temporibus corrupti et.",  
        content: "Illum sint quos rerum dolores conse...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4cb0000000062f7bb77> {  
        id: 8,  
        title: "Et labore omnis dolorum dolor vel laborum.",  
        content: "Magnam nisi fuga odio expedita vita...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4d00000000062f7bb77> {  
        id: 9,  
        title: "Neque dolores fugit unde rerum...",  
        content: "Quia nemo quia libero et ipsam sit...",  
        published: 1,  
        created_at: "0000-00-00 00:00:00",  
        updated_at: "0000-00-00 00:00:00"  
    },  
    <App\Post #00000006db8a4270000000062f7bb77> {  
        id: 10,  
        title: "Rerum aliquam omnis enim doloribus quaerat sint.",  
        content: "Dicta ab ipsa sunt voluptas recusandae...",  
        published: 0,
```

```
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #00000006db8a4de000000062f7bb77> {
        id: 11,
        title: "Iusto voluptates assumenda esse.",
        content: "Nobis voluptas eveniet doloremque...",
        published: 0,
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]
```

Sip.

Sebagaimana dijelaskan diawal, penggunaan scope akan sangat bermanfaat untuk mengumpulkan logic scope yang sama dalam satu tempat. Fitur ini digunakan pada fitur Soft Delete di Laravel. Contoh penggunaan lainnya misalnya digunakan untuk memfilter product yang memiliki stock > 0, memfilter hanya customer yang memiliki membership, dll.

Menggunakan Accessors dan Mutators

Dalam mengakses database, terkadang kita ingin merubah input sebelum disimpan ke dalam table atau mengubahnya sebelum dioutputkan dari model. Di Eloquent, teknik ini bisa dilakukan dengan mengaktifkan Accessor dan Mutator. Ada beberapa kasus yang akan sangat cocok jika kita menggunakan fitur ini:

- **Hanya Mutator.** Contohnya pada saat pengisian field password. Pada saat kita mengisi password untuk user, kita ingin mengisinya dengan teks biasa. Tapi, ketika disimpan ke database, kita ingin meng-hash-nya secara otomatis.
- **Hanya Accessor.** Contohnya, kita ingin membuat attribut baru yang merupakan isinya dinamis sesuai isi pada field lain. Misalnya, di Database kita hanya ingin menyimpan field `firstname` dan `lastname`. Kita ingin membuat attribut `fullname` yang berisi gabungan `firstname` dan `lastname`, tanpa menambah field tersebut ke table.
- **Accessor dan Mutator.** Contohnya, penggunaan tanggal di form. Misalnya, pada saat kita menginputkan tanggal, kita ingin menggunakan format `d/m/Y`, sedangkan di database kita ingin menyimpannya dengan format `Y-m-d`. Begitupun ketika mengampilkan isinya, meskipun di database disimpan dengan format `Y-m-d`, kita ingin outputnya dengan format `d/m/Y`.

Untuk memudahkan pembahasan ini, mari kita buat model user untuk mempraktekan semua kasus diatas. Karena model User ini sudah ada bawaan dari Laravel, mari kita hapus terlebih dahulu:

```
vagrant@homestead:~/Code/sample-model$ rm -rf app/User.php
```

Kemudian kita generate Model dan Migrationnya:

```
vagrant@homestead:~/Code/sample-model$ php artisan make:model User
Model created successfully.
Created Migration: 2015_05_17_063621_create_users_table.php
```

Pastikan isi model User seperti berikut:

app/User.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model {
    //
}
```

Kita tambahkan field yang dibutuhkan pada method up di file migration:

database/migrations/2015_05_17_063621_create_users_table.php

```
<?php
....
class CreateUsersTable extends Migration {
    ...
    public function up()
    {
        Schema::create('users', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('email');
            $table->string('firstname')->nullable();
            $table->string('lastname')->nullable();
            $table->string('password');
            $table->date('birth_date')->nullable();
        });
    }
}
```

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

Jalankan `php artisan migrate:refresh --seed` untuk mendapatkan struktur database yang baru. Setelah siap, mari kita buat Accessor dan Mutator yang kita butuhkan.

Hanya Mutator

Mutator digunakan pada saat kita hendak menyimpan nilai sebuah attribut ke dalam database. Pada kasus ini, kita hendak melakukan *hash* setiap kali nilai password di simpan di database. Untuk melakukan *hashing*, kita akan menggunakan fitur `Hash::make()` di Laravel. Fitur ini akan melakukan *hashing* dengan bcrypt untuk string yang kita jadikan parameter.

```
>>> Hash::make('rahasia');
=> "$2y$10$ibNXjA5G6ti7jIeoKEVjROefnrB3NSxIuUxKGJdqdyB.Spos5k1J2"
```

Untuk membuat mutator, kita harus membuat method baru dengan nama `setNamafieldAttribute()`. Untuk field password, maka nama methodnya `setPasswordAttribute()`. Method ini akan menerima parameter nilai yang kita set.

app/User.php

```
...
class User extends Model {
  public function setPasswordAttribute($password)
  {
    $this->attributes['password'] = \Hash::make($password);
  }
}
```

Kita menggunakan syntax

```
$this->attributes['password'] = \Hash::make($password);
```

untuk mengubah isian field `password` sebelum disimpan ke dalam database. mari kita coba:

```

>>> $user = new App\User;
>>> $user->email = "lulung@gmail.com";
>>> $user->firstname = "haji";
>>> $user->lastname = "lulung";
>>> $user->password = "jakarta123";
>>> $user->save();
=> true
>>> $user->toArray();
=> [
    "email"      => "lulung@gmail.com",
    "firstname"   => "haji",
    "lastname"    => "lulung",
    "password"    => "$2y$10$ . jLqNhJhgaNs3KKkr . CHd . CqQgVp83E4szYwjCk1sAG996CQ8 \
SyrS",
    "updated_at"  => "2015-05-17 09:19:08",
    "created_at"  => "2015-05-17 09:19:08",
    "id"          => 1
]

```

Sip. Field password telah berhasil melakukan *hashing* ketika menyimpan field password.

Hanya Accessor

Untuk membuat accessor, kita dapat menggunakan cara yang sama dengan pembuatan mutator. Perbedaannya, nama methodnya diawali dengan `get`, menjadi `getNamafieldAttribute()`. Jika field ini sudah ada di dalam table, maka ia akan menerima parameter berupa nilai field tersebut. Jika tidak, maka method ini tidak menerima parameter.

Pada kasus yang kita miliki, kita akan membuat accessor untuk field `fullname` yang merupakan gabungan dari field `firstname` dan `lastname`. Kita buat seperti berikut:

app/User.php

```

.....
class User extends Model {

    .....

    public function getfullnameAttribute()
    {
        return $this->firstname . ' ' . $this->lastname;
    }
}

```

Mari kita coba:

```
>>> $user = new App\User;  
>>> $user->email = "lulung@gmail.com";  
>>> $user->firstname = "haji";  
>>> $user->lastname = "lulung";  
>>> $user->password = "jakarta123";  
>>> $user->fullname;  
>>> $user->save();  
=> true
```

Sip. Berhasil.

Ada sedikit tambahan, jika kita hendak menampilkan model tersebut dalam bentuk Array / JSON, maka field custom buatan kita tidak akan muncul:

```
>>> $user->toArray();  
=> [  
    "email"      => "lulung@gmail.com",  
    "firstname"   => "haji",  
    "lastname"    => "lulung",  
    "password"    => "$2y$10$t5hj8KF%KVhgFhhTT11Z10jVYS0i23G3.DRtQ/t4YwCm2na/f\\  
6ABK",  
    "updated_at"  => "2015-05-17 09:27:37",  
    "created_at"  => "2015-05-17 09:27:37",  
    "id"          => 2  
]
```

Jika kita ingin memunculkan field tersebut, kita harus menambahkannya ke attribut `$appends` di model User seperti berikut:

app/User.php

```
....  
class User extends Model {  
    protected $appends = ['fullname'];  
....  
}
```

Jika kita memiliki custom accessor yang lain, cukup menambahkan pada attribut `$appends` ini. Setelah selesai, mari kita coba:

```
>>> $user = App\User::find(1);
>>> $user->toArray();
=> [
    "id"      => 1,
    "email"   => "lulung@gmail.com",
    "firstname" => "haji",
    "lastname"  => "lulung",
    "password"  => "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCk1sAG996CQ8\
SyrS",
    "birth_date" => "0000-00-00",
    "created_at" => "2015-05-17 09:19:08",
    "updated_at" => "2015-05-17 09:19:08",
    "fullname"   => "haji lulung"
]

```

Sip. Berhasil.

Accessor dan Mutator

Setelah kita memahami bagaimana membuat accessor dan mutator, mari kita gabungkan keduanya pada satu field. Pada kasus kita, field `birth_date` ingin kita simpan dengan format `Y-m-d` tetapi mengaksesnya dengan format `d/m/Y`.

Format `Y-m-d` merupakan format default untuk menyimpan field bertipe `Date` di database. Dapat kita cek dengan syntax berikut:

```
>>> $user = App\User::find(1);
>>> $user->birth_date = "1959-07-24";
=> "1959-07-24"
>>> $user->save();
=> true
>>> $user->toArray();
=> [
    "id"      => 1,
    "email"   => "lulung@gmail.com",
    "firstname" => "haji",
    "lastname"  => "lulung",
    "password"  => "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCk1sAG996CQ8\
SyrS",
    "birth_date" => "1959-07-24",
    "created_at" => "2015-05-17 09:19:08",
    "updated_at" => "2015-05-17 09:43:19",
]
```

```

    "fullname" => "haji lulung"
]

```

Terlihat disini, kita berhasil menyimpan field `birth_date`.

Format `d/m/Y` biasanya digunakan di form. Jika kita menggunakan untuk mengisi field bertipe `Date` di database, maka ia akan berubah menjadi `0000-00-00`. Ini dapat dibuktikan dengan syntax berikut:

```

>>> $user = App\User::find(1);
>>> $user->birth_date = "24/07/1959";
=> "24/07/1959"
>>> $user->save();
=> true
>>> unset($user);
=> null
>>> $user = App\User::find(1);
=> <App\User #000000006eafa5460000000023fa690a> {
    id: 1,
    email: "lulung@gmail.com",
    firstname: "haji",
    lastname: "lulung",
    password: "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCklsAG996CQ8SyrS",
    birth_date: "0000-00-00",
    created_at: "2015-05-17 09:19:08",
    updated_at: "2015-05-17 09:49:24"
}
>>> $user->birth_date;
=> "0000-00-00"

```

Kita menggunakan syntax `unset($user)`; untuk me-refresh model User yang kita dapatkan. Terlihat disini, isian `birth_date` telah berubah menjadi `"0000-00-00"`.

Untuk menggunakan format `d/m/Y` pada saat mengakses maupun mengisi field `birth_date`, kita akan membuat accessor dan mutator menggunakan class Carbon. Untuk merubah format `d/m/Y` menjadi `Y-m-d` ketika menyimpan field ke database kita akan menggunakan syntax berikut:

```

>>> Carbon\Carbon::createFromFormat('d/m/Y', '30/04/2015')->toDateString();
=> "2015-04-30"

```

Sementara, untuk merubah format `Y-m-d` menjadi `d/m/Y` ketika mengakses field, kita gunakan syntax berikut:

```
>>> Carbon\Carbon::createFromFormat('Y-m-d', '2015-03-20')->format('d/m/Y'); => \
"20/03/2015"
```

Untuk mengimplementasikannya ke dalam model User, kita buat syntax seperti berikut:

app/User.php

```
....  
class User extends Model {  
    ....  
    public function setBirthDateAttribute($date)  
    {  
        $this->attributes['birth_date'] = \Carbon\Carbon::createFromFormat('d/m/Y', \  
$date)->toDateString();  
    }  
  
    public function getBirthDateAttribute($date)  
    {  
        return \Carbon\Carbon::createFromFormat('Y-m-d', $date)->format('d/m/Y');  
    }  
}
```

Mari kita cek dengan mengubah birth_date pada record dengan ID 1:

```
>>> $user = App\User::find(1);  
=> <App\User #000000005038fc47000000003405cd08> {  
    id: 1,  
    email: "lulung@gmail.com",  
    firstname: "haji",  
    lastname: "lulung",  
    password: "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCklsAG996CQ8Syrs",  
    birth_date: "0000-00-00",  
    created_at: "2015-05-17 09:19:08",  
    updated_at: "2015-05-17 09:49:24"  
}  
>>> $user->birth_date = "24/07/1959";  
=> "24/07/1959"  
>>> $user->save();  
=> true  
>>> unset($user);  
=> null  
>>> $user = App\User::find(1);
```

Terlihat disini, kita menggunakan format d/m/Y untuk mengisi field birth_date. Sengaja kita melakukan `unset()` pada variable `$user` setelah menyimpannya, untuk me-refresh isi record. Kini, jika kita mengecek isi `birth_date` dia akan menggunakan format d/m/Y:

```
>>> $user->birth_date;
=> "24/07/1959"
```

Pada output Array / JSON pun, ia akan tetap menggunakan format d/m/Y:

```
>>> $user->toArray();
=> [
    "id"      => 1,
    "email"   => "lulung@gmail.com",
    "firstname" => "haji",
    "lastname"  => "lulung",
    "password"  => "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCklsAG996CQ8\SyrS",
    "birth_date" => "24/07/1959",
    "created_at" => "2015-05-17 09:19:08",
    "updated_at" => "2015-05-17 13:03:34",
    "fullname"   => "haji lulung"
]
```

Tapi, pada database tetap akan tersimpan dengan format Y-m-d. Ini bisa kita buktikan dengan mengaksesnya menggunakan Query Builder:

```
>>> DB::table('users')->find(1);
=> <stdClass #000000005038fc43000000003405cd08> {
    id: 1,
    email: "lulung@gmail.com",
    firstname: "haji",
    lastname: "lulung",
    password: "$2y$10$.jLqNhJhgaNs3KKkr.CHd.CqQgVp83E4szYwjCklsAG996CQ8SyrS",
    birth_date: "1959-07-24",
    created_at: "2015-05-17 09:19:08",
    updated_at: "2015-05-17 13:03:34"
}
```

Sip. Saya rasa sudah cukup penjelasannya sampai disini. Di lapangan, banyak sekali contoh penggunaan Accessor dan Mutator ini. Saya sendiri pernah menulis artikel untuk [menggabungkan fitur ini dengan relationship⁹⁹](#) untuk membuat relasi berdasarkan kode unik pada salah satu field.

⁹⁹<https://medium.com/laravel-indonesia/tips-membuat-relasi-berdasarkan-kode-di-id-pada-framework-laravel-f2944d8ca7fb?source=1>

Date Mutator & Carbon

Masih ingat dengan Carbon di pembahasan tentang [Composer](#)? Pada pembahasan sebelumnya, kita hanya membahas penggunaan package ini untuk menampilkan waktu dalam bentuk “ago”. Pada Eloquent, kita dapat mengubah setiap output dari field yang bertipe Date atau DateTime menjadi instance dari Carbon.

Memahami Penggunaan Carbon

Jika kita cek ke [dokumentasi resmi](#)¹⁰⁰, terdapat banyak sekali manfaat dari Carbon. Fitur-fiturnya akan sangat memudahkan jika kita akan berinteraksi dengan tanggal dibandingkan menggunakan class DateTime bawaan PHP. Carbon sendiri meng-*extends* class DateTime, sehingga semua yang bisa kita lakukan dengan class DateTime akan bisa kita lakukan dengan Carbon.

A screenshot of a web browser displaying the official Carbon API documentation at carbon.nesbot.com. The browser window shows the title 'Carbon - A simple PHP API' and the URL 'carbon.nesbot.com'. The main content area features a green header with the text 'A simple PHP API extension for DateTime.' Below this, there is a code block containing PHP code demonstrating various Carbon methods like now(), addDay(), addWeek(), createFromDate(), and isWeekend().

```

printf("Right now is %s", Carbon::now()->toDateTimeString());
printf("Right now in Vancouver is %s", Carbon::now('America/Vancouver')); //implicit __toString()
$tomorrow = Carbon::now()->addDay();
$lastWeek = Carbon::now()->subWeek();
$nextSummerOlympics = Carbon::createFromDate(2012)->addYears(4);

$officialDate = Carbon::now()->toRfc2822String();

$showOldAmI = Carbon::createFromDate(1975, 5, 21)->age;

$noonTodayLondonTime = Carbon::createFromTime(12, 0, 0, 'Europe/London');

$worldWillEnd = Carbon::createFromDate(2012, 12, 21, 'GMT');

// Don't really want to die so mock now
Carbon::setTestNow(Carbon::createFromDate(2000, 1, 1));

// comparisons are always done in UTC
if (Carbon::now()->gte($worldWillEnd)) {
    die();
}

// Phew! Return to normal behaviour
Carbon::setTestNow();

if (Carbon::now()->isWeekend()) {
    echo 'Party!';
}

```

Dokumentasi Carbon

Laravel, secara default, telah memasukan library ini sebagai dependensinya. Mari kita pelajari beberapa fitur nya:

Berbagai cara membuat instance Carbon

¹⁰⁰ <http://carbon.nesbot.com/>

Tanpa memberi parameter (sama dengan Carbon::now())

```
>>> $carbon = new Carbon\Carbon();
=> <Carbon\Carbon #000000004f586434000000016a5c7faa> {
    date: "2015-05-16 13:18:37.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membuat dari tanggal saat ini (tanpa jam)

```
>>> $carbon = Carbon\Carbon::today();
=> <Carbon\Carbon #000000004f586431000000016a5c7faa> {
    date: "2015-05-16 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membuat dari tanggal kemarin

```
>>> $carbon = Carbon\Carbon::yesterday();
=> <Carbon\Carbon #000000004f586438000000016a5c7faa> {
    date: "2015-05-15 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membuat dari tanggal besok

```
>>> $carbon = Carbon\Carbon::tomorrow();
=> <Carbon\Carbon #000000004f5864ce000000016a5c7faa> {
    date: "2015-05-17 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membuat dengan parameter tanggal

```
>>> $carbon = Carbon\Carbon::createFromDate(2014, 5, 30);
=> <Carbon\Carbon #000000004f5864ca000000016a5c7faa> {
    date: "2014-05-30 13:25:59.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membuat dengan parameter tanggal dan waktu

```
>>> $carbon = Carbon\Carbon::create(2015, 4, 30, 10, 45, 30, 'Asia/Jakarta');
=> <Carbon\Carbon #000000004f5864c8000000016a5c7faa> {
    date: "2015-04-30 10:45:30.000000",
    timezone_type: 3,
    timezone: "Asia/Jakarta"
}
```

Memparsing string tanggal

```
>>> $carbon = Carbon\Carbon::parse('2015-5-30');
=> <Carbon\Carbon #000000004f5864ce000000016a5c7faa> {
    date: "2015-05-30 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Memformat Output

Mengambil attribut tanggal

```
>>> $carbon = Carbon\Carbon::now();
=> <Carbon\Carbon #000000004f5864c7000000016a5c7faa> {
    date: "2015-05-16 13:33:42.000000",
    timezone_type: 3,
    timezone: "UTC"
}
>>> $carbon->year;
=> 2015
>>> $carbon->month;
=> 5
```

```
>>> $carbon->day;  
=> 16  
>>> $carbon->hour;  
=> 13  
>>> $carbon->minute;  
=> 33  
>>> $carbon->second;  
=> 42  
>>> $carbon->date;  
=> "2015-05-16 13:33:42.000000"  
>>> $carbon->timezone;  
=> "UTC"
```

Mengambil tanggal yang telah di format

```
>>> $carbon->toDateString();  
=> "2015-05-16"
```

Mengambil waktu yang telah di format

```
>>> $carbon->toTimeString();  
=> "13:33:42"
```

Mengambil tanggal dengan format custom

```
>>> $carbon->format('T\o\d\a\y \i\s 1, jS \\of F Y');  
=> "Today is Saturday, 16th of May 2015"
```

Melakukan operasi terhadap tanggal

Menambah hari

```
>>> $carbon = Carbon\Carbon::parse('2015-01-12');  
>>> $carbon->addDays(3);  
=> <Carbon\Carbon #000000004f586433000000016a5c7faa> {  
    date: "2015-01-15 00:00:00.000000",  
    timezone_type: 3,  
    timezone: "UTC"  
}
```

Mengurangi hari

```
>>> $carbon = Carbon\Carbon::parse('2015-01-12');
>>> $carbon->subDays(3);
=> <Carbon\Carbon #000000004f586434000000016a5c7faa> {
    date: "2015-01-09 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Menambah minggu

```
>>> $carbon = Carbon\Carbon::parse('2015-01-12');
>>> $carbon->addWeeks(2);
=> <Carbon\Carbon #000000004f586432000000016a5c7faa> {
    date: "2015-01-26 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Menambah bulan

```
>>> $carbon = Carbon\Carbon::parse('2015-01-12');
>>> $carbon->addMonths(5);
=> <Carbon\Carbon #000000004f5864c8000000016a5c7faa> {
    date: "2015-06-12 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Membandingkan tanggal

```
>>> $carbonA = Carbon\Carbon::parse('2015-01-12');
>>> $carbonB = Carbon\Carbon::parse('2015-02-12');
```

Sama dengan

```
>>> $carbonA->eq($carbonB);  
=> false
```

Tidak sama dengan

```
>>> $carbonA->ne($carbonB);  
=> true
```

Lebih dari

```
>>> $carbonA->gt($carbonB);  
=> false
```

Lebih dari atau sama dengan

```
>>> $carbonA->lte($carbonB);  
=> true
```

kurang dari

```
>>> $carbonA->lt($carbonB);  
=> true
```

kurang dari atau sama dengan

```
>>> $carbonA->lte($carbonB);  
=> true
```

Untuk lebih lengkapnya, silahkan cek [dokumentasi Carbon](#)¹⁰¹.

Mengaktifkan Date Mutator

Secara default, Laravel sudah mengaktifkan Date Mutator untuk field `created_at`, `updated_at` (harus mengaktifkan [fitur timestamp](#)) dan `deleted_at` (harus mengaktifkan [soft delete](#)). Kita dapat mengeceknya pada model Post. Mari kita cek isinya terlebih dahulu:

¹⁰¹ <http://carbon.nesbot.com>

```
>>> App\Post::select('id','title','created_at','updated_at')->get();
=> <Illuminate\Database\Eloquent\Collection #000000007b55ab1e00000000212256e0> [
    <App\Post #000000007b55ab1800000000212256e0> {
        id: 2,
        title: "Tempore non labore ullam qui.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000007b55ab1100000000212256e0> {
        id: 3,
        title: "Iusto deserunt laborum consequatur beatae sapiente.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000007b55ab0f00000000212256e0> {
        id: 4,
        title: "Accusamus inventore laudantium tempora nesciunt aut voluptati\\
bus ipsum.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000007b55ab0c00000000212256e0> {
        id: 6,
        title: "Alias perferendis nihil temporibus vitae nihil est hic.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000007b55ab0d00000000212256e0> {
        id: 8,
        title: "Quia aut eveniet velit molestias.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    },
    <App\Post #000000007b55ab3200000000212256e0> {
        id: 9,
        title: "Consequatur facere deserunt laboriosam laboriosam autem.",
        created_at: "0000-00-00 00:00:00",
        updated_at: "0000-00-00 00:00:00"
    }
]
```

Hmm.. ada yang aneh disini. Isian created_at dan updated_at tidak terisi..

Ini terjadi karena kita menggunakan method `insert` untuk membuat record Post. Ada dua cara yang bisa kita lakukan:

- Memberikan nilai `created_at` dan `updated_at` secara eksplisit pada method `insert`.
- Menggunakan method `save` untuk membuat record

Mari kita gunakan solusi kedua. Ubah `PostTableSeeder` menjadi:

```
1 <?php
2 use Faker\Factory as Faker;
3 use Illuminate\Database\Seeder;
4
5 class PostsTableSeeder extends Seeder {
6
7     public function run()
8     {
9         $faker = Faker::create();
10        for ($i=0; $i <= 10 ; $i++) {
11            $post = new App\Post;
12            $post->title = $faker->sentence;
13            $post->content = $faker->paragraph;
14            $post->published = rand(0,1);
15            $post->save();
16        }
17    }
18 }
```

Jalankan `php artisan migrate:refresh --seed` untuk me-refresh isi database. Kemudian cek kembali isi table Post:

```
>>> App\Post::select('id','title','created_at','updated_at')->get();
=> <Illuminate\Database\Eloquent\Collection #000000007b55ab1100000000212256e0> [
    <App\Post #000000007b55ab0100000000212256e0> {
        id: 1,
        title: "Voluptatum et saepe maxime minus rerum nam.",
        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    },
    <App\Post #000000007b55ab15000000000212256e0> {
        id: 2,
        title: "Qui est in quisquam et incidunt rerum.",
```

```

        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    },
    <App\Post #000000007b55ab1700000000212256e0> {
        id: 4,
        title: "Vel aut ut eos ipsa qui distinctio perspiciatis.",
        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    },
    <App\Post #000000007b55ab1c00000000212256e0> {
        id: 5,
        title: "Voluptas nemo ipsa est sapiente.",
        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    },
    <App\Post #000000007b55ab1a00000000212256e0> {
        id: 10,
        title: "Ea corporis consequatur dolor.",
        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    },
    <App\Post #000000007b55ab0b00000000212256e0> {
        id: 11,
        title: "Et quae accusamus eius.",
        created_at: "2015-05-16 15:50:06",
        updated_at: "2015-05-16 15:50:06"
    }
]

```

Kita dapat mengecek date mutator pada field `created_at` dan `updated_at` dengan cara berikut:

```

>>> $post = App\Post::find(1);
>>> $post->created_at;
=> <Carbon\Carbon #000000007b55ab050000000021224960> {
    date: "2015-05-16 15:50:06.000000",
    timezone_type: 3,
    timezone: "UTC"
}
>>> $post->updated_at;
=> <Carbon\Carbon #000000007b55ab040000000021224960> {
    date: "2015-05-16 15:50:06.000000",

```

```
    timezone_type: 3,  
    timezone: "UTC"  
}
```

Tuh kan?

Jika diinginkan kita dapat mengaktifkan Date Mutator pada field lain, misalnya kita tambahkan field `viewed_at` pada table post untuk memberikan tanggal terakhir kali post dibaca. Mari kita tambahkan field ini pada file migration:

database/migrations/2015_04_10_002726_create_posts_table.php

```
<?php  
....  
class CreatePostsTable extends Migration {  
    ....  
    public function up()  
    {  
        Schema::create('posts', function(Blueprint $table)  
        {  
            $table->increments('id');  
            $table->string('title');  
            $table->text('content');  
            $table->boolean('published');  
            $table->timestamp('viewed_at');  
            $table->timestamps();  
        });  
    }  
    ....  
}
```

Syarat menggunakan Date Mutator adalah field harus bertipe timestamp atau date. Pada contoh ini, kita menggunakan tipe timestamp.

Selanjutnya, kita perlu mengaktifkan Date Mutator untuk field ini pada model Post. Untuk mengaktifkannya, kita akan meng-override method `getDates()` pada model. Method ini harus berisi array nama field yang akan memiliki fitur Date Mutator:

App/Post.php

```
<?php namespace App;  
....  
class Post extends Model {  
    ....  
    public function getDates()  
    {  
        return ['created_at', 'updated_at', 'viewed_at'];  
    }  
}
```

Karena kita mengaktifkan timestamp, pada method ini kita juga menambahkan `created_at` dan `updated_at`. Kita juga perlu merubah `PostTableSeeder` agar mengisi field `viewed_at` secara acak:

database/seeds/PostsTableSeeder.php

```
<?php  
use Faker\Factory as Faker;  
use Illuminate\Database\Seeder;  
  
class PostsTableSeeder extends Seeder {  
  
    public function run()  
    {  
        $faker = Faker::create();  
        for ($i=0; $i <= 10 ; $i++) {  
            $post = new App\Post;  
            $post->title = $faker->sentence;  
            $post->content = $faker->paragraph;  
            $post->published = rand(0,1);  
            $post->viewed_at = Carbon\Carbon::now()  
                ->addDays(rand(1,5))  
                ->addHours(rand(0,18));  
            $post->save();  
        }  
    }  
}
```

Jalankan kembali `php artisan migrate:refresh --seed` untuk menerapkan perubahan database ini. Setelah selesai, kita dapat mengecek fitur Date Mutator pada field `viewed_at` dengan cara berikut:

```
>>> $post = App\Post::find(1);
>>> $post->viewed_at;
=> <Carbon\Carbon #000000004e4d4ce7000000001ee4e87b> {
    date: "2015-05-18 23:11:47.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Ketika kita sudah mengaktifkan fitur Date Mutator pada sebuah field maka kita dapat mengisinya dengan berbagai cara:

Dengan string format Y-m-d

```
>>> $post->viewed_at = "2015-08-15";
=> "2015-08-15"
>>> $post->viewed_at;
=> <Carbon\Carbon #000000004e4d4ce7000000001ee4e87b> {
    date: "2015-08-15 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Instance dari DateTime

```
>>> $post->viewed_at = new DateTime();
>>> $post->viewed_at;
=> <Carbon\Carbon #000000004e4d4ce8000000001ee4e87b> {
    date: "2015-05-16 16:19:30.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Instance dari Carbon

```
>>> $post->viewed_at = Carbon\Carbon::tomorrow();
>>> $post->viewed_at;
=> <Carbon\Carbon #000000004e4d4c15000000001ee4e87b> {
    date: "2015-05-17 00:00:00.000000",
    timezone_type: 3,
    timezone: "UTC"
}
```

Tentunya, kita juga dapat menggunakan semua fitur Carbon pada field ini.

Attribute Casting

Di dalam PHP, *Casting* adalah merubah type data sebuah variable ke tipe lainnya. Untuk memahaminya perhatikan syntax berikut.

Pada contoh ini, kita akan mengisi sebuah angka yang diapit tanda kutip:

```
>>> $nilai = "100";
=> "100"
```

Jika kita mengecek tipenya, maka ia akan mengembalikan `string`:

```
>>> gettype($nilai);
=> "string"
```

Jika kita mengisinya dengan angka yang tidak diapit tanda kutip, maka ia akan bertipe `integer`:

```
>>> $nilai = 100;
=> 100
>>> gettype($nilai);
=> "integer"
```

Bagaimana jika kita ingin membuat angka tersebut diapit tanda kutip tapi tetap memperlakukannya sebagai `integer`? Kita dapat melakukan *casting*:

```
>>> $nilai = (int) "100";
=> 100
>>> gettype($nilai);
=> "integer"
```

Pada syntax diatas, `$nilai = (int) "100"` artinya merubah tipe data dari "100" menjadi integer.

Memahami Attribut Casting

Attribut casting ibarat Mutator tapi hanya melakukan casting, tanpa menambah logic lain. Misalnya, pada model `Product`, kita ingin merubah tipe data dari `price` menjadi `double` setiap kali diakses. Menggunakan mutator, ini yang akan kita lakukan:

app/Product.php

```
.....
class Product extends Model {
    .....
    public function getPriceAttribute($price)
    {
        return (double) $price;
    }
}
```

Kita dapat mengeceknya dengan tinker:

```
>>> $product = App\Product::find(1);
>>> $product->price;
=> 677000000.0
>>> gettype($product->price);
=> "double"
```

Jika menggunakan attribut casting, ini syntax yang akan kita buat:

app/Product.php

```
....  
class Product extends Model {  
    ....  
    protected $casts = [  
        'price' => 'double',  
    ];  
    ....  
    public function getPriceAttribute($price)  
    {  
        return (double) $price;  
    }  
}
```

Jika kita cek menggunakan tinker, maka dia akan tetap di-cast sebagai double:

```
>>> $product = App\Product::find(1);  
>>> $product->price;  
=> 677000000.0  
>>> gettype($product->price);  
=> "double"
```

Jenis Tipe Casting

Laravel mendukung berbagai tipe attribut casting, diantaranya:

integer (atau int)

Tipe ini akan merubah sebuah field menjadi `integer`.

float (atau real atau double)

PHP memperlakukan Real, Float, dan Double sebagai tipe yang sama. Semuanya menjadi tipe data desimal.

string

Tipe ini akan merubah sebuah field menjadi `string`.

boolean (atau bool)

Tipe ini akan merubah sebuah field menjadi boolean. Untuk menggunakan fitur ini, Laravel mengasumsikan isian field berupa 0 dan 1.

object

Casting ke Object dan Array merupakan fitur yang paling menarik. Fitur ini mengasumsikan kita menyimpan field dalam bentuk JSON. Ketika kita melakukan casting ke Object, maka Laravel akan menggunakan syntax `json_decode($value)` dan mengembalikan object stdClass.

array

Mirip dengan casting Object, tipe ini mengasumsikan kita menyimpan field dalam bentuk JSON. Ketika melakukan casting ke Array, maka Laravel akan menggunakan syntax `json_decode($value, true)` dan mengembalikan Array.

Silahkan cek [source code¹⁰²](#) Laravel untuk melihat implementasi fitur ini.

Contoh Penggunaan Attribut Casting

Banyak manfaat yang bisa kita dapatkan dari penggunaan fitur ini. Misalnya, dalam table User kita dapat menambah field `activated` untuk mengecek apakah seorang user sudah aktif. Tentunya field ini akan kita simpan dengan tipe data TINYINT. Mari kita tambahkan pada file migration:

database/migrations/2015_05_17_063621_create_users_table.php

```
....  
class CreateUsersTable extends Migration {  
    ....  
    public function up()  
    {  
        Schema::create('users', function(Blueprint $table)  
        {  
            $table->increments('id');  
            $table->string('email');  
            $table->string('firstname')->nullable();  
            $table->string('lastname')->nullable();  
            $table->string('password');
```

¹⁰² <https://github.com/illuminate/database/blob/32fee7419ddbaa0bcfcfb829872c1a0bb4b8ead8/Eloquent/Model.php#L2721>

```

    $table->date('birth_date')->nullable();
    $table->tinyInteger('activated')->default(false);
    $table->timestamps();
});

}

....
```

Jalankan `php artisan migrate:refresh --seed` untuk mendapatkan struktur baru.

Mari kita coba buat sample user yang sudah aktif:

```

>>> $user = new App\User;
>>> $user->email = "rhoma.irama@gmail.com";
>>> $user->firstname = "Bang";
>>> $user->lastname = "Haji";
>>> $user->birth_date = "11/12/1946";
>>> $user->password = "terlalu";
>>> $user->activated = 1;
>>> $user->save();
>>> $user->id;
=> 1
```

Jika kita menggunakan PHP, tentunya field `activated` akan bernilai `true` jika kita masukan ke dalam sebuah kondisi (restart tinker):

```

>>> $user = App\User::find(1);
>>> $user->activated == true;
=> true
```

Nah, jika kita buat dalam bentuk JSON (misalnya dalam membangun), lebih baik jika field ini langsung memberikan nilai `true` atau `false`. Tanpa attribut casting, kita hanya akan mendapatkan angka 1 atau 0:

```

>>> $user->toJson();
=> "{\"id\":1,\"email\":\"rhoma.irama@gmail.com\",\"firstname\":\"Bang\",\"lastn\
ame\":\"Haji\",\"password\":\"$2y$10$jzSzZlaYjr9tqgeoDfCjmutTP2w.ZLg0FLp56abrSaQ\
x.P4b3.r76\",\"birth_date\":\"11\\12\\1946\",\"activated\":1,\"created_at\":\"\
2015-05-17 14:14:07\",\"updated_at\":\"2015-05-17 14:21:01\",\"fullname\":\"Bang\
Haji\"}"
```

Hmm.. agak susah dibaca. Mari kita gunakan `toArray()`, agar outputnya lebih mudah dibaca:

```
>>> $user->toArray();
=> [
    "id"      => 1,
    "email"   => "rhoma.irama@gmail.com",
    "firstname" => "Bang",
    "lastname"  => "Haji",
    "password"  => "$2y$10$jzSzZlaYjr9tqgeoDfCjmutTP2w.ZLgOFLp56abrSaQx.P4b3\
.r76",
    "birth_date" => "11/12/1946",
    "activated"  => 1,
    "created_at" => "2015-05-17 14:14:07",
    "updated_at" => "2015-05-17 14:21:01",
    "fullname"   => "Bang Haji"
]
```

Kita dapat membuat field ini berisi nilai `true` atau `false` dengan mengaktifkan Attribut Casting pada model User untuk field `activated`:

app/User.php

```
...
class User extends Model {
    protected $casts = [
        'activated' => 'boolean'
    ];
...
}
```

Mari kita cek (restart tinker):

```
>>> $user = App\User::find(1);
>>> $user->toArray();
=> [
    "id"      => 1,
    "email"   => "rhoma.irama@gmail.com",
    "firstname" => "Bang",
    "lastname"  => "Haji",
    "password"  => "$2y$10$jzSzZlaYjr9tqgeoDfCjmutTP2w.ZLgOFLp56abrSaQx.P4b3\
.r76",
    "birth_date" => "11/12/1946",
    "activated"  => true,
    "created_at" => "2015-05-17 14:14:07",
```

```
    "updated_at" => "2015-05-17 14:21:01",
    "fullname"   => "Bang Haji"
]
```

Terlihat disini, field `activated` sudah langsung berubah menjadi boolean.

Dalam view pun, ketika kita ingin menampilkan status ini, kita dapat membuat syntaxnya lebih sederhana. Seperti berikut:

```
>>> $user->activated ? "User Aktif" : "User Belum diaktivasi";
=> "User Aktif"
```

Model Event & Observer

Dalam setiap proses CRUD yang kita lakukan terhadap model, Eloquent akan menjalankan event yang bisa kita isi (*hook*) dengan berbagai syntax sesuai kebutuhan. Berikut beberapa event berikut waktu dijalankannya:

- `creating` : Dijalankan ketika kita hendak menyimpan model ke database pertama kali (record baru). Event ini akan dijalankan **sebelum** model di simpan ke database.
- `created` : Hampir sama dengan `creating`. Perbedaannya event ini dijalankan **setelah** model disimpan ke database.
- `updating` : Dijalankan ketika ketika kita hendak menyimpan model yang telah kita ubah datanya (update). Event ini akan dijalankan **sebelum** model di simpan ke database.
- `updated` : Hampir sama dengan `updating`. Perbedaannya event ini dijalankan **setelah** model disimpan ke database.
- `saving` : Akan dijalankan ketika kita hendak menyimpan model. Event ini akan selalu dijalankan tanpa mengecek apakah model tersebut merupakan record baru atau bukan. Event ini akan dijalankan **sebelum** model di simpan ke database.
- `saved` : Hampir sama dengan `saving`. Perbedaannya event ini dijalankan **setelah** model disimpan ke database.
- `deleting` : Dijalankan ketika ketika kita hendak menghapus model. Event ini akan dijalankan **sebelum** model dihapus.
- `deleted` : Hampir sama dengan `saved`. Perbedaannya event ini dijalankan **setelah** model dihapus.
- `restoring` : Dijalankan ketika ketika kita hendak mengembalikan model yang sudah dihapus, dimana model tersebut mengaktifkan soft delete. Event ini akan dijalankan **sebelum** model dikembalikan.
- `restored` : Hampir sama dengan `restoring`. Perbedaannya event ini dijalankan **setelah** model dikembalikan.

Konfigurasi

Untuk menggunakan setiap event ini, kita harus melakukan *hook*. Proses ini dapat kita lakukan dengan menambahkan pada method `boot()` di modelnya atau pada Service Provider. Misalnya, kita akan menggunakan event `created` untuk menambah Log baru setiap kali membuat Product baru. Menggunakan cara pertama, ini yang perlu kita tambahkan pada model Product:

app/Product.php

```
....  
class Product extends Model {  
    ....  
    protected static function boot()  
    {  
        parent::boot();  
  
        static::created(function($model) {  
            \Log::info('Berhasil menambah ' . $model->name . '. Stok : ' . $model->sto\  
ck);  
        });  
    }  
    ....  
}
```

Disini, kita menggunakan `static` untuk melakukan *hook* terhadap event di model yang digunakan. Syntax `static::created` artinya kita hendak melakukan *hook* terhadap event `created` pada model yang sedang aktif (Product).

Jika diperhatikan, kita juga menggunakan `\"` ketika memanggil Log. Ini kita lakukan agar class Log yang kita gunakan merupakan class yang berada di global namespace. Tanpa `\"`, method `boot` ini akan menganggap class tersebut berada pada namespace yang aktif (`App\Product`).

Kita coba dengan syntax berikut:

```
>>> $product = new App\Product;  
>>> $product->name = "Kawasaki NinjaZX-10R";  
>>> $product->price = 348000000;  
>>> $product->stock = 23;  
>>> $product->save();  
=> true
```

Jika kita cek pada file log, maka akan terdapat baris berikut:

storage/logs/laravel-XXXX-XX-XX.log

```
[XXXX-XX-XX XX:XX:XX] local.INFO: Berhasil menambah Kawasaki NinjaZX-10R. Stok :\n23
```

Menggunakan cara kedua, kita harus menambahkan hook pada method `boot()` di Service Provider. Kita dapat membuat service provider sendiri atau menggunakan EventServiceProvider yang sudah disediakan Laravel. Mari kita gunakan EventServiceProvider, ubah method `boot()` seperti berikut:

app/Providers/EventServiceProvider.php

```
....\n\nclass EventServiceProvider extends ServiceProvider {\n\n    ....\n\n    public function boot(DispatcherContract $events)\n    {\n        parent::boot($events);\n\n        \App\Product::created(function($model) {\n            \Log::info('Berhasil menambah ' . $model->name . '. Stok : ' . $model->sto\\ck . ' (dari EventServiceProvider));\n        });\n    }\n}
```

Kita cek dengan syntax berikut:

```
>>> $product = new App\Product;\n>>> $product->name = "Kawasaki Ninja-RR mono";\n>>> $product->price = 48900000;\n>>> $product->stock = 16;\n>>> $product->save();\n=> true
```

Pastikan di file log terdapat baris berikut:

storage/logs/laravel-XXXX-XX-XX.log

```
[XXXX-XX-XX XX:XX:XX] local.INFO: Berhasil menambah Kawasaki Ninja-RR mono. Stok\\\n: 16 (dari EventServiceProvider)
```

Sip. Setelah kita memahami bagaimana melakukan konfigurasi, mari kita pelajari beberapa contoh penggunaan model event ini.

Model Observer

Jika dibutuhkan, kita dapat membuat class khusus yang menangani setiap event pada model. Ini akan bermanfaat untuk memisahkan tanggung jawab (*Separation of Concern¹⁰³*). Di Laravel, kita dapat menggunakan Model Observer untuk melakukan hal ini.

Untuk membuat model observer, kita harus melakukan 3 langkah:

- Membuat sebuah class dengan nama method berupa nama event pada model. Nama classnya bebas, untuk memudahkan disarankan dengan nama `NamamodelObserver`. Misalnya observer untuk class User, kita beri nama `UserObserver`. Untuk class Product, kita beri nama `ProductObserver`, dst..
- Menambah method untuk tiap event yang akan di hook pada class yang kita buat.
- Me-*register* observer yang kita buat pada model dengan method `observe`.

Pada contoh sebelumnya, kita dapat hook pada model Product menjadi model Observer dengan membuat class `ProductObserver` seperti berikut:

app/ProductObserver

```
<?php namespace App;
class ProductObserver {
    public function created($model)
    {
        \Log::info('Berhasil menambah ' . $model->name . ' . Stok : ' . $model->stock\.
        . '(dibuat dari ProductObserver)');
    }
}
```

Karena kita hanya melakukan hook untuk event `created`, maka pada class ini kita hanya membuat method `created`. Selanjutnya, kita me-*register* observer ini. Proses ini bisa kita lakukan pada method `boot` di model:

¹⁰³https://en.wikipedia.org/wiki/Separation_of_concerns

app/Product.php

```
....  
class Product extends Model {  
    ....  
    protected static function boot()  
    {  
        parent::boot();  
  
        static::observe(new \App\ProductObserver);  
    }  
    ....  
}
```

Mari kita coba dengan syntax berikut:

```
>>> $product = new App\Product;  
>>> $product->name = "Vellfire";  
>>> $product->stock = 34;  
>>> $product->save();  
=> true
```

Jika kita cek pada file log, pastikan terdapat baris berikut:

storage/logs/laravel-XXXX-XX-XX.log

```
[XXXX-XX-XX XX:XX:XX] local.INFO: Berhasil menambah Vellfire. Stok : 34 (dibuat \  
dari ProductObserver
```

Contoh Penggunaan: Membatalkan proses CRUD

Untuk setiap event yang dijalankan **sebelum** proses CRUD dilakukan (*creating, saving, deleting, updating, restoring*), jika kita mengembalikan nilai `false`, maka proses CRUD akan dibatalkan.

Contoh penggunaannya untuk melakukan validasi data. Pada table Product, kita dapat membatalkan proses perubahan nama product, jika product sudah pernah di order. Ini dapat kita lakukan dengan mengecek table `orders_products`. Mari kita tambahkan hook ini pada model Product. Ubah isian method `boot` menjadi:

app/Product.php

```
....  
protected static function boot()  
{  
    parent::boot();  
    static::updating(function($model) {  
        if ( DB::table('orders_products')->where('product_id', $model->id)->count() \> 0 && $model->isDirty('name')) {  
            return false;  
        }  
    });  
}  
....  
}
```

Pada syntax ini, kita menggunakan query builder untuk mengecek table `orders_products` dengan `product_id` berupa id model yang aktif. Kita menggunakan method `count()` untuk mengecek apakah terdapat record yang memiliki `product_id` tersebut. Cara yang lebih elegan tentunya menggunakan relasi di Eloquent. Tapi, karena kita belum mempelajarinya, menggunakan query builder sudah cukup.

Kita juga menggunakan syntax `$model->isDirty('name')` untuk mengecek apakah atribut `name` telah berubah. Method `isDirty` dapat kita lihat di [source code Eloquent](#)¹⁰⁴.

Misalnya, kita memiliki isian table `orders_products` seperti berikut:

```
mysql> select * from orders_products;  
+-----+-----+  
| order_id | product_id |  
+-----+-----+  
| 1 | 6 |  
| 1 | 6 |  
| 2 | 2 |  
| 2 | 7 |  
| 3 | 6 |  
| 3 | 4 |  
+-----+-----+
```

Jika kita mencoba merubah field `name` untuk product dengan id 6, maka Laravel akan membatalkannya:

¹⁰⁴ <https://github.com/illuminate/database/blob/5.0/Eloquent/Model.php#L3001-L3015>

```
>>> $product = App\Product::find(6);
=> <App\Product #000000003aeeef39d000000000413bc9b> {
    name: "Freed",
    price: 638000000,
    stock: 40
}
>>> $product->name = "Brio";
>>> $product->save();
=> false
```

Terlihat disini, output dari method `save()` adalah `false`. Ini menandakan proses penyimpanan model ke database dibatalkan.

Jika kita merubah field selain `name`, maka proses perubahan akan berhasil disimpan:

```
>>> $product = App\Product::find(6);
=> <App\Product #000000003e83946f00000000051c2a522> {
    name: "Freed",
    price: 638000000,
    stock: 40
}
>>> $product->stock = 100;
>>> $product->price = 400000000;
>>> $product->save();
=> true
```

Begitupun jika kita melakukan perubahan pada Product yang belum di order, maka perubahan tersebut akan berhasil disimpan:

```
>>> $product = App\Product::find(1);
=> <App\Product #000000003e83946c00000000051c2a522> {
    name: "Accord",
    price: 271000000,
    stock: 39
}
>>> $product->name = "Swift Sport"
>>> $product->price = 319000000;
>>> $product->stock = 14;
>>> $product->save();
=> true
```

Contoh Penggunaan: Membuat log perubahan model

Dalam membuat aplikasi, terkadang kita perlu mencatat perubahan apa saja yang sudah terjadi pada suatu model. Banyak teknik yang bisa kita pakai. Untuk sekarang, mari kita gunakan table `product_logs` untuk mencatat perubahan yang terjadi pada model `Product`.

Pertama, kita buat model dan table untuk `product_logs`:

```
vagrant@homestead:~/Code/sample-model$ php artisan make:model ProductLog
Model created successfully.
Created Migration: 2015_05_24_103953_create_product_logs_table
```

Kita isi tambahkan beberapa field pada table `product_logs`:

database/migrations/2015_05_24_103953_create_product_logs_table.php

```
...
class CreateProductLogsTable extends Migration {
    ...
    public function up()
    {
        Schema::create('product_logs', function(Blueprint $table)
        {
            $table->integer('product_id');
            $table->text('changes');
            $table->timestamps();
        });
    }
    ...
}
```

Pada table `product_logs`, kita membuat field `product_id` yang akan mencatat id dari product yang telah berubah. Kita juga membuat field `changes` yang akan berisi json berisi perubahan apa saja yang telah di lakukan. Struktur JSON nya akan seperti berikut:

```
[  
 {  
   "attribute" : "nama_field",  
   "from" : "isian_lama",  
   "to" : "isian_baru"  
 },  
 ...  
 ]
```

Pada table ini kita juga mengaktifkan timestamps untuk mencatat kapan perubahan dilakukan. Jalankan `php artisan migrate` untuk menambahkan table ini ke database.

Aktifkan *mass assignment* pada model `ProductLog` untuk field-field tersebut:

app/ProductLog.php

```
<?php namespace App;  
use Illuminate\Database\Eloquent\Model;  
class ProductLog extends Model {  
  protected $fillable = ['product_id', 'changes'];  
}
```

Untuk memudahkan, kita juga akan menggunakan mutator untuk field `changes` agar kita dapat mengisinya dengan array dan otomatis merubahnya menjadi JSON:

app/ProductLog.php

```
....  
class ProductLog extends Model {  
  ....  
  public function setChangesAttribute($value)  
  {  
    $this->attributes['changes'] = json_encode($value);  
  }  
}
```

Sebaliknya, pada saat kita mendapatkan isian field `changes`, kita akan merubahnya menjadi array. Ada dua cara yang bisa kita lakukan, menggunakan accessor atau attribut casting. Mari kita gunakan accessor:

app/ProductLog.php

```
....  
class ProductLog extends Model {  
    ....  
    public function getChangesAttribute($value)  
    {  
        return json_decode($value);  
    }  
}
```

Setelah model ProductLog siap, mari kita hook event `updating` pada model Product. Hook ini akan membuat record baru dengan perubahan yang telah dilakukan pada model Product. Berikut syntaxnya:

app/Providers/EventServiceProvider.php

```
1 ....  
2 class EventServiceProvider extends ServiceProvider {  
3     ....  
4     public function boot(DispatcherContract $events)  
5     {  
6         parent::boot($events);  
7  
8         \App\Product::updating(function($model) {  
9             $changes = [];  
10            foreach($model->getDirty() as $attribute => $new){  
11                $original = $model->getOriginal($attribute);  
12                if ($original != $new) {  
13                    $change = [  
14                        'attribute' => $attribute,  
15                        'from' => $original ,  
16                        'to' => $new  
17                    ];  
18                    $changes[] = $change;  
19                }  
20            }  
21  
22            if ( count($changes) > 0 ) {  
23                \App\ProductLog::create([  
24                    'product_id' => $model->id,  
25                    'changes' => $changes  
26                ]);  
27            }  
28        }  
29    }  
30}
```

```

27     }
28
29     return true;
30 });
31 }
32 }
```

Weiss.. syntaxnya cukup panjang. Tenang, akan kita bahas perbagian:

- Baris 8: Melakukan hook terhadap event `updating` pada model `App\Product`.
- Baris 9: Menyiapkan array `$changes` yang akan mencatat semua perubahan yang kita lakukan.
- Baris 10: Menggunakan method `getDirty()` kita akan mendapatkan array berisi atribut dan nilainya yang terbaru. Method ini bisa kita temui di [source code Eloquent¹⁰⁵](#).
- Baris 11: Menggunakan method `getOriginal()` kita mengambil nilai lama untuk attribut yang kita inginkan. Method ini bisa kita temui di [source code Eloquent¹⁰⁶](#).
- Baris 12-18: Mengecek apakah nilai attribut sudah berubah. Jika ya, maka kita menambahnya pada array `$changes`.
- Baris 22-26: Mengecek isian array `$changes`. Jika ada isinya, maka kita buat `ProductLog` baru untuk mencatat perubahan yang kita lakukan.
- Baris 29: Kita harus mengembalikan `true` agar proses update disimpan ke database.

Untuk memudahkan mari kita hapus hook yang telah kita buat sebelumnya pada model Product:

app/Product.php

```

.....
class Product extends Model {
    .....
    protected static function boot()
    {
        parent::boot();
        static::updating(function($model) {
            if (\DB::table('orders_products')->where('product_id', $model->id)->count() > 0 && $model->isDirty('name')) {
                return false;
            }
        });
    }
}
```

¹⁰⁵<https://github.com/illuminate/database/blob/5.0/Eloquent/Model.php#L3022-L3040>

¹⁰⁶<https://github.com/illuminate/database/blob/5.0/Eloquent/Model.php#L2965-L2968>

```
    }
    });
}
...
}
```

Untuk mengecek hook ini, mari kita buat beberapa perubahan pada sebuah Product:

```
>>> $product = App\Product::find(1);
=> <App\Product #000000006d687af10000000079518f39> {
    name: "Accord",
    price: 189000000,
    stock: 17
}
>>> $product->name = "Yaris";
>>> $product->save();
=> true
>>> $product->name = "New Camry";
>>> $product->price = 74450000;
>>> $product->save();
=> true
```

Disini, kita membuat dua kali perubahan. Perubahan pertama, kita hanya merubah nama product. Perubahan kedua, kita mengubah nama product berikut harganya. Mari kita cek hasilnya pada model ProductLog:

```
>>> App\ProductLog::where('product_id',1)->get()->toArray();
=> [
    [
        "product_id" => 1,
        "changes"      => [
            <stdClass #000000006d687a1a0000000079518f39> {
                attribute: "name",
                from: "Accord",
                to: "Yaris"
            }
        ],
        "created_at" => "2015-05-24 13:58:45",
        "updated_at" => "2015-05-24 13:58:45"
    ],
    [

```

```
"product_id" => 1,
"changes"      => [
    <stdClass #000000006d687af40000000079518f39> {
        attribute: "name",
        from: "Yaris",
        to: "New Camry"
    },
    <stdClass #000000006d687aee0000000079518f39> {
        attribute: "price",
        from: 189000000,
        to: 74450000
    }
],
"created_at"  => "2015-05-24 14:00:07",
"updated_at"  => "2015-05-24 14:00:07"
]
]
```

Sip. Dapat kita lihat disini, kita berhasil mencatat semua perubahan yang kita lakukan. Masih banyak contoh penggunaan model event dilapangan, misalnya mengirim notifikasi ke user, membatasi penghapusan record berdasarkan relasinya, dll. Batasnya adalah imajinasi kita.. :)

Route Model Binding

Pada contoh di routing, kita telah membuat route seperti ini:

app/Http/routes.php

```
Route::get('/product/{id}', function($id) {
    return App\Product::find($id);
});
```

Syntax ini akan mengambil ID yang dikirim dan mencari record yang sesuai. Ini contoh outputnya:



Contoh route tanpa binding

Route ini berjalan dengan baik. Tentunya, kita dapat menggunakan route seperti ini dalam produksi. Tapi, karena fitur seperti ini sudah sangat umum, Laravel memberikan fitur model binding untuk melakukan proses pencarian model ini secara otomatis.

Untuk melakukannya, ada dua langkah yang perlu kita lakukan:

- Melakukan binding nama parameter ke sebuah model pada RouteServiceProvider.
- Menambahkan parameter pada route

Mari kita ubah route diatas menjadi model binding. Pertama, kita binding nama parameter ke model Product. Untuk memudahkan, mari kita namai `product`. Berikut syntaxnya:

app/Providers/RouteServiceProvider.php

```
....  
class RouteServiceProvider extends ServiceProvider {  
    ....  
    public function boot(Router $router)  
    {  
        parent::boot($router);  
        $router->model('product', 'App\Product');  
    }  
    ....  
}
```

Selanjutnya, kita ubah route sebelumnya agar menggunakan parameter ini:

app/Http/routes.php

```
Route::get('/product/{product}', function($product) {
    return $product;
});
```

Terlihat disini, karena Laravel secara otomatis akan mencari model product dengan Id berupa parameter, di baris kedua kita langsung mengembalikan model yang kita dapatkan. Tanpa mencarinya terlebih dahulu. Jika kita coba:



Route dengan model binding

Keren kan? :)



Source code dari bab ini bisa didapat di <https://bitbucket.org/rahmatawaludin/sample-model>¹⁰⁷



Ringkasan

Di bab ini kita telah mempelajari beberapa fitur dari Eloquent. Setiap fitur yang ada di Eloquent dibuat untuk memudahkan developer dalam membangun aplikasi yang solid dan *Maintainable*. Tapi, kekuatan sesungguhnya dari Eloquent terletak pada pengelolaan relasi database yang akan kita pelajari pada bab selanjutnya. Semangat!

¹⁰⁷ <https://bitbucket.org/rahmatawaludin/sample-model/commits/all>

Relationship dalam Eloquent

Tahapan yang paling penting dalam membangun sebuah aplikasi adalah menentukan struktur aplikasi. Ketika hal ini sudah direncanakan dengan baik, proses membangun aplikasi akan berjalan dengan lebih cepat (disamping mengurangi jumlah revisi).

Salah satu proses yang harus kita lakukan ketika merencanakan sebuah aplikasi adalah struktur database. Tentunya merencanakan sebuah database tidak sekedar membuat ERD berikut semua relasinya. Pada level yang lebih tinggi, pada proses ini kita sedang melakukan [Domain Modeling¹⁰⁸](#). Penjelasan di Wikipedia tentang [Domain Driven Design¹⁰⁹](#) bisa jadi referensi yang bagus untuk mempelajari konsep Domain Modeling.

Sebagaimana kita pelajari di bab sebelumnya, dengan Eloquent kita dapat menentukan berbagai *behaviour* dari suatu model. Kita juga telah mempelajari berbagai fitur yang ada di Eloquent. Tapi, kita belum menyentuh kekuatan asli dari Eloquent, yaitu relationship.

Proses pembangunan relationship antar model dapat kita lakukan pada dua level, database dan aplikasi. Kita dapat membuat relasi antar model berikut menambahkan relasinya di database, maupun tanpa relasi di database sama sekali. Untuk memahami proses pembuatan relasi di database pada migration, silahkan buka kembali topik tentang [Migration](#).

Sip, yuk kita mulai!



Untuk membuat penjelasan lebih terfokus, saya tidak akan membuat migration dan seeder untuk topik-topik di bab ini. Saya yakin, Anda sudah mampu membuatnya sendiri.

Memahami Jenis Relationship

Dalam bekerja dengan model kita akan sangat sering mendengar istilah *relationship*. Jika diterjemahkan, *relatioSHIP* artinya hubungan. Tentunya, hubungan disini adalah hubungan antar model (table). Bukan hubungan jenis lainnya. Halah.. -_-

Relasi ini menentukan bagaimana sebuah model berinteraksi dengan model lain. Misalnya model `Movie` akan memiliki keterangan `Studio` yang membuatnya. Tentunya,

¹⁰⁸ http://en.wikipedia.org/wiki/Domain_model

¹⁰⁹ http://en.wikipedia.org/wiki/Domain-driven_design

Studio tidak hanya akan memiliki satu Movie. Disini, kita akan menggunakan relasi hasMany dari Studio ke Movie. Secara bahasa, hasMany artinya *memiliki banyak*, makanya kita menggunakan relasi ini.

Sebaliknya, kita akan menggunakan relasi belongsTo dari model Movie ke Studio. Kembali kita cek terjemahnya, belongsTo artinya *milik*. Ini tentunya sangat *human friendly*, karena sebuah Movie hanya dimiliki satu Studio (dalam contoh ini satu Movie hanya boleh dimiliki satu Studio).

Terdapat berbagai jenis relasi yang bisa kita gunakan di Eloquent sesuai dengan kebutuhan kita selama membangun aplikasi. Mari kita pelajari satu persatu.

One To One

Relasi jenis ini bisa kita gunakan ketika kita ingin menambah atribut dari suatu model pada model yang berbeda. Contohnya, kita memiliki table users seperti berikut:

Table Users

| id | email | password |
|----|----------------|-----------------------------------------|
| 1 | dini@gmail.com | \$2y\$10\$C/YbVVNsnhv.1ku3b83e/.65gz... |
| 2 | dani@gmail.com | \$2y\$10\$AUZIkqoRAj4.8oC1R7dkxuouZK... |
| 3 | deon@gmail.com | \$2y\$10\$/uBuJfwgbKxdNZJcPZd1GupoaS... |

Model User memiliki tanggung jawab untuk menyimpan data autentikasi user. Misalnya, dalam membuat aplikasi kita ingin menyimpan data *preferences* (settingan) user berupa negara, jenis mata uang yang akan digunakan dan sebagainya.

Kita bisa saja menyimpan semua data ini pada table User, tapi dengan teknik itu akan menambah tanggung jawab model User. Cara yang lebih disarankan adalah menggunakan model terpisah untuk menangani data ini. Misalnya, kita buat table Preferences dengan konten seperti berikut:

Table Preferences

| id | user_id | country | currency | subscribe_mailing_list |
|----|---------|---------|----------|------------------------|
| 1 | 1 | ID | IDR | 1 |
| 2 | 2 | MY | MYR | 0 |
| 3 | 3 | SG | SGD | 1 |

Karena model `User` hanya diizinkan memiliki satu `Preferences`, maka kita menggunakan relasi **One-to-One**.

Konfigurasi

Untuk menggunakan relasi `One-to-One`, kita perlu menambahkan field yang dibutuhkan untuk relasi ini. Secara default Eloquent akan mengasumsikan pada table induk (`User`) memiliki primary key bernama `id`. Sedangkan pada table relasi, terdapat field dengan format `model_id` bertipe `integer` dan `unsigned`. Untuk kasus ini berarti harus terdapat field `user_id` pada table `preferences`.

Teknik penamaan ini merupakan ketentuan di Eloquent. Untuk relasi jenis lain, Eloquent akan menggunakan skema penamaan field yang sama. Tentunya, kita pun dapat mengkonfigurasi nama field yang berbeda pada table relasi. Teknik ini akan kita pelajari pada bagian selanjutnya.

Setelah menambahkan field yang sesuai kita perlu menambah method relasinya pada model `User` dan `Preference`. Berikut syntax pada model `User`:

app/User.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model {

    protected $fillable = ['email', 'password'];

    public function preference()
    {
        return $this->hasOne('App\Preference');
    }
}
```

Pada method `preference()` kita menggunakan method `hasOne('App\Preference')`. Method ini menandakan bahwa kita sedang membuat relasi one-to-one ke model `App\Preference`. Sehingga kita akan mendapatkan model `App\Preference` yang menjadi relasinya. Sedangkan pada model `Preference`, kita menggunakan syntax berikut:

app/Preference.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Preference extends Model {

    protected $fillable = ['user_id', 'country', 'currency', 'subscribe_mailing_list'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }
}
```

Pada method `user()` kita menggunakan method `belongsTo('App\User')`. Method ini akan memberikan kita model `User` yang menjadi relasi dari model `Preference`.



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹⁰

Penggunaan

Kita dapat mengakses relasi dengan memanggil method `preference()`. Output dari method ini adalah instance dari `Illuminate\Database\Eloquent\Relations\HasOne`:

```
>>> App\User::find(1)->preference()
=> <Illuminate\Database\Eloquent\Relations\HasOne #0000000072dd6f8b0000000073aff\3e1> {}
```

Dari sini, kita dapat menggunakan `fetch method` maupun `query constraint` yang kita butuhkan. Misalnya, kita dapat mengecek query yang dilakukan oleh Eloquent dengan method `toSql()`:

¹¹⁰<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/1fbb51009dcdedc4d239861ab9682d62820e67d9>

```
>>> App\User::find(1)->preference()->toSql();
=> "select * from `preferences` where `preferences`.`user_id` = ? and `preferenc\
es`.`user_id` is not null"
```

Untuk mendapatkan model `Preference` yang menjadi relasinya, kita dapat menggunakan method `get()`:

```
>>> App\User::find(1)->preference()->get();
=> <Illuminate\Database\Eloquent\Collection #0000000072dd6f8b0000000073aff3e1> [
    <App\Preference #0000000072dd6f710000000073aff3e1> {
        id: 1,
        user_id: 1,
        country: "ID",
        currency: "IDR",
        subscribe_mailing_list: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    }
]
```

Tentunya, memanggil method `get()` setiap kali kita ingin mendapatkan model `Preference` yang menjadi relasi dari model `User` akan cukup merepotkan. Disinilah “keajaiban” Eloquent muncul.



Dynamic Properties

Untuk setiap method yang menghasilkan relasi, Eloquent akan secara otomatis membuat attribut baru dengan nama method yang kita buat. Attribut ini akan menampilkan output dari method `getResults()` pada class relasi.

Pada contoh ini, pada model `User` akan terdapat attribut `preference`. Attribut ini akan menampilkan hasil dari method `getResults()` pada `Illuminate\Database\Eloquent\Relations\HasOne`:

vendor/laravel/framework/src/Illuminate/Database/Eloquent/Relations/HasManyThrough.php

```
....  
public function getResults()  
{  
    return $this->query->first();  
}  
....
```

Telihat disini, Eloquent akan menampilkan record pertama dari query yang dilakukan pada table preferences. Sehingga kita akan mendapatkan output berikut:

```
>>> $user = App\User::find(1);  
>>> $user->preference;  
=> <App\Preference #0000000064893eff000000005b8c42c0> {  
    id: 1,  
    user_id: 1,  
    country: "ID",  
    currency: "IDR",  
    subscribe_mailing_list: 1,  
    created_at: "2015-05-30 16:07:54",  
    updated_at: "2015-05-30 16:07:54"  
}
```

Kitapun dapat memanggil field pada model Preference dengan mudah:

```
>>> $user->preference->country;  
=> "ID"  
>>> $user->preference->currency;  
=> "IDR"  
>>> $user->preference->subscribe_mailing_list;  
=> 1
```

Jika hendak mengubah nilai pada model Preference, kita dapat melakukannya seperti berikut:

```
>>> $user->preference->country = "US";
>>> $user->preference->currency = "USD";
>>> $user->preference->save();
>>> $user->preference;
=> <App\Preference #0000000064893efc000000005b8c42c0> {
    id: 1,
    user_id: 1,
    country: "US",
    currency: "USD",
    subscribe_mailing_list: 1,
    created_at: "2015-05-30 16:07:54",
    updated_at: "2015-05-31 09:49:38"
}
```

Atau menggunakan *mass assignment* dengan method `update()`:

```
>>> $user->preference->update(['country'=>'JPN', 'currency'=>'JPY']);
>>> $user->preference;
=> <App\Preference #0000000064893efc000000005b8c42c0> {
    id: 1,
    user_id: 1,
    country: "JPN",
    currency: "JPY",
    subscribe_mailing_list: 1,
    created_at: "2015-05-30 16:07:54",
    updated_at: "2015-05-31 09:55:48"
}
```

Jika pada model `Preference` terdapat dua record dengan `user_id` yang sama, Eloquent akan memilih record yang pertama. Misalnya, kita ubah `user_id` dari semua record pada table `preferences` menjadi 1:

```
mysql> update preferences set user_id = 2;
Query OK, 2 rows affected (0.02 sec)
Rows matched: 3  Changed: 2  Warnings: 0

mysql> select * from preferences;
+----+-----+-----+-----+
| id | user_id | country | currency | subscribe_mailing_list |
+----+-----+-----+-----+
| 1 | 2 | JPN | JPY | 1 |
| 2 | 2 | MY | MYR | 0 |
```

```
| 3 |      2 | SGD      | SGD      |  
+---+-----+-----+-----+  
3 rows in set (0.01 sec)
```

Jika kita mencoba mengakses `preferences` dari `User` dengan id 1, maka kita hanya akan mendapatkan record yang pertama:

```
>>> App\User::find(2)->preference;  
=> <App\Preference #0000000064893e04000000005b8c42c0> {  
    id: 1,  
    user_id: 2,  
    country: "JPN",  
    currency: "JPY",  
    subscribe_mailing_list: 1,  
    created_at: "2015-05-30 16:07:54",  
    updated_at: "2015-05-31 09:55:48"  
}
```

Jika kita hendak menambah `preference` pada `User` yang baru dibuat, kita dapat membuat instance dari `Preference` dan menggunakan method `save` ketika memanggil relasi `preference` dari model `User`.

```
>>> $user = App\User::create(['email'=>'eric@gmail.com', 'password'=>Hash::make(\  
'123456'));  
>>> $preference = App\Preference::create(['country'=>'UK', 'currency'=>'GBP']);  
>>> $user->preference()->save($preference);
```

Menggunakan method `save()` diatas merupakan kelebihan penggunaan relasi. Tentunya, kita selalu bisa membuat `Preference` baru dan memasukkan field `user_id` secara manual.

Pada penggunaan di lapangan, biasanya akan ditemui kasus dimana kita akan menyimpan nilai `preference` yang baru atau mengupdate yang sudah ada. Untuk melakukan hal ini, kita dapat menggunakan method `FirstOrCreate()` pada model `Preference`:

```
> $user = App\User::create(['email'=>'evan@gmail.com', 'password'=>Hash::make('6\54321'));  
>>> $preference = App\Preference::FirstOrCreate(['user_id'=>$user->id]);  
>>> $preference->update(['country'=>'CN', 'currency'=>'CNY']);  
=> true  
>>> $user->preference;  
=> <App\Preference #00000007aa43ff10000000001ecbad2> {  
    id: 5,  
    user_id: 6,  
    country: "CN",  
    currency: "CNY",  
    subscribe_mailing_list: 1,  
    created_at: "2015-05-31 12:52:01",  
    updated_at: "2015-05-31 12:53:40"  
}
```

Method `FirstOrCreate` akan mencari record berdasarkan kriteria nilai field yang telah kita tentukan. Jika record tidak ditemukan, maka Laravel akan membuatkannya untuk kita. Pada contoh ini, kita mencari model `Preference` dengan `user_id` sesuai `User` yang baru kita buat. Karena `Preference` tersebut belum ada, maka Eloquent akan membuatkannya.

Setelah kita mendapatkan instance dari `Preference`, kita dapat menggunakan method `update()` untuk mengupdate data dari record yang telah kita dapatkan.

One To Many

Sesuai namanya relasi jenis ini kita gunakan ketika hendak membuat hubungan 1 model ke banyak model. Contohnya hubungan antara `Movie` dan `Studio`, dimana sebuah `Studio` dapat **memiliki banyak** `Movie`. Dan satu `Movie` **hanya dimiliki** satu `Studio` (untuk contoh ini kita menganggap tidak ada `Movie` yang dibuat oleh beberapa `Studio`).

Relasi ini dapat kita buat dengan struktur database berikut:

Table Studio

| id name founded_at |
|---------------------------------------|
| 1 Pixar 1986-02-03 |
| 2 DreamWorks Animation 2004-10-27 |

Table Movies

| id name date_released studio_id |
|---------------------------------------|
| 1 Cars 2 2011-06-24 1 |
| 2 Finding Nemo 2003-05-30 1 |
| 3 Kung Fu Panda 2 2011-05-26 2 |

Terlihat disini Studio dengan nama Pixar memiliki 2 Movie. Dan setiap Movie hanya memiliki satu Studio, dapat dilihat dari field studio_id.

Konfigurasi

Untuk membuat relasi ini, Laravel memiliki struktur penamaan field yang sama dengan relasi one-to-one. Dimana kita harus membuat field dengan nama model_id pada table yang direlasikan. Disini, kita harus membuat field studio_id pada table movies.

Pada model pun hampir sama, kita hanya merubah `hasOne` menjadi `hasMany`.

Berikut syntax lengkap untuk model Studio:

app/Studio.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Studio extends Model {

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

```
public function movies()
{
    return $this->hasMany('App\Movie');
}
```

Disini kita menggunakan method `movies()` untuk membuat relasi `hasMany` ke model `Movie`. Penggunaan plural untuk menamai method ini merupakan teknik yang disarankan ketika membuat relasi `hasMany`. Ini karena output yang dihasilkannya merupakan koleksi dari beberapa model `Movie`.

Syntax lengkap untuk model `Movie` seperti berikut:

app/Movie.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Movie extends Model {

    protected $fillable = ['name', 'date_released', 'studio_id'];

    public function studio()
    {
        return $this->belongsTo('App\Studio');
    }
}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹¹

Penggunaan

Untuk menggunakan relasi ini kita dapat menggunakannya dari model `studio` maupun dari model `Movie`. Dari model `studio`, kita dapat memanggil method `movies()` dan menambahkan `fetch method` maupun `query constraint` yang kita butuhkan. Misalnya, kita ambil semua model `Movie` yang berelasi dengan method `get()`:

¹¹¹ <https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/aef2dd014ca66e39c5a2988f35b4958ae2f63749>

```
>>> App\Studio::find(1)->movies()->get();
=> <Illuminate\Database\Eloquent\Collection #0000000072dd6f770000000073aff3e1> [
    <App\Movie #0000000072dd6f870000000073aff3e1> {
        id: 1,
        name: "Cars 2",
        date_released: "2011-06-24",
        studio_id: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    },
    <App\Movie #0000000072dd6f740000000073aff3e1> {
        id: 2,
        name: "Finding Nemo",
        date_released: "2003-05-30",
        studio_id: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    }
]
```

Atau menghitung jumlah Movie yang menjadi relasinya dengan method count():

```
>>> App\Studio::find(1)->movies()->count();
=> 2
```

Kita juga bisa menambah **query constraint** yang dibutuhkan. Misalnya, pada Movie yang berelasi kita hanya ingin mencari movie yang memiliki nama `nemo`:

```
>>> App\Studio::find(1)->movies()->where('name', 'LIKE', '%nemo%')->get();
=> <Illuminate\Database\Eloquent\Collection #0000000072dd6f760000000073aff3e1> [
    <App\Movie #0000000072dd6f790000000073aff3e1> {
        id: 2,
        name: "Finding Nemo",
        date_released: "2003-05-30",
        studio_id: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    }
]
```

Seperti biasa, Eloquent telah membuatkan kita attribut baru sesuai nama relasi. Dalam contoh ini, kita bisa menggunakan attribut `movies` untuk mengakses semua Movie yang berelasi:

```
>>> App\Studio::find(1)->movies;
=> <Illuminate\Database\Eloquent\Collection #0000000072dd6f830000000073aff3e1> [
    <App\Movie #0000000072dd6f8e0000000073aff3e1> {
        id: 1,
        name: "Cars 2",
        date_released: "2011-06-24",
        studio_id: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    },
    <App\Movie #0000000072dd6f840000000073aff3e1> {
        id: 2,
        name: "Finding Nemo",
        date_released: "2003-05-30",
        studio_id: 1,
        created_at: "2015-06-01 00:23:23",
        updated_at: "2015-06-01 00:23:23"
    }
]
```

Dari model `Movie`, kita dapat mendapatkan `Studio`-nya dengan memanggil attribut `studio`.

```
>>> App\Movie::find(1)->studio;
=> <App\Studio #00000006b60d27d0000000073635e8a> {
    id: 1,
    name: "Pixar",
    founded_at: "1986-02-03",
    created_at: "2015-06-01 00:23:23",
    updated_at: "2015-06-01 00:23:23"
}
>>> App\Movie::find(1)->studio->name;
=> "Pixar"
```

Ketika kita membuat `Movie` baru kita dapat menambah `movie_id` secara manual atau secara otomatis menggunakan method `save`. Misalnya, kita menambah `Movie` Toy Story 3 dan hendak di relasikan ke `Studio Pixar`. Menggunakan cara kedua, ini yang kita lakukan:

```
>>> $movie = App\Movie::create(['name'=>'Toy Story 3', 'date_released'=>'2013-06-21']);
>>> $studio = App\Studio::where('name', 'Pixar')->first();
>>> $studio->movies()->save($movie);
=> <App\Movie #000000006b60d2850000000073635e8a> {
    name: "Toy Story 3",
    date_released: "2013-06-21",
    updated_at: "2015-06-01 13:38:53",
    created_at: "2015-06-01 13:38:17",
    id: 4,
    studio_id: 1
}
```

Ketika kita menggunakan relasi `hasMany()`, kita dapat menggunakan method `saveMany()` dari model `Studio` untuk merelasikan banyak `Movie` ke `Studio` secara sekaligus dengan parameter berupa beberapa model `Movie`:

```
>>> $studio = App\Studio::create(['name'=>'Universal Studio', 'founded_at'=>'1912-04-30']);
>>> $studio->movies()->saveMany([App\Movie::find(1), App\Movie::find(2)]);
=> [
    <App\Movie #0000000051d12c9f00000000669acaf4> {
        id: 1,
        name: "Cars 2",
        date_released: "2011-06-24",
        studio_id: 3,
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 13:20:13"
    },
    <App\Movie #0000000051d12c8f00000000669acaf4> {
        id: 2,
        name: "Finding Nemo",
        date_released: "2003-05-30",
        studio_id: 3,
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 13:20:13"
    }
]
```

Karena relasi dari `Movie` ke `Studio` adalah `belongsTo`, kita bisa menggunakan method `associate` dengan parameter model `Studio` untuk menambahkan relasi dari `Movie` ke `Studio`:

```
>>> App\Movie::find(1)->studio()->associate(App\Studio::find(1));
=> <App\Movie #0000000051d12c8100000000669acaf4> {
    id: 1,
    name: "Cars 2",
    date_released: "2011-06-24",
    studio_id: 1,
    created_at: "2015-06-13 09:58:09",
    updated_at: "2015-06-13 13:20:13",
    studio: <App\Studio #0000000051d12c8d00000000669acaf4> {
        id: 1,
        name: "Pixar",
        founded_at: "1986-02-03",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09"
    }
}
```

Many To Many

Sesuai namanya, *Many to Many* atau banyak ke banyak, relasi jenis ini kita gunakan ketika sebuah model memiliki relasi ke lebih satu record pada model lain atau sebaliknya. Contohnya seorang student (mahasiswa) dapat mengambil berbagai course (mata kuliah). Dan setiap course, sangat mungkin diikuti oleh banyak student.

Untuk membuat relasi jenis ini, kita harus membuat table perantara. Contoh penamaan table perantara untuk table ini adalah `course_student`. Penamaan ini diatur secara alfabetis berdasarkan nama model (tentunya kita dapat merubahnya, untuk sekarang mari kita gunakan penamaan yang disarankan).

Berikut contoh struktur table nya:

Table Courses

| id title units room |
|-----------------------------------------------|
| 1 Kemampuan Menulis Dasar 3 B13 |
| 2 Menggambar Manga Level 1 2 A10 |
| 3 Mewarnai Manga Level 1 2 A01 |

Table Students

| id name date_of_birth gender |
|------------------------------------|
| 1 Joni 1990-08-14 m |
| 2 Dadang 1987-12-04 m |
| 3 Yuni 1992-11-14 f |

Table Course_Student

| course_id student_id |
|------------------------|
| 1 1 |
| 2 1 |
| 3 1 |
| 1 2 |
| 2 2 |
| 3 3 |

Konfigurasi

Untuk membuat relasi ini, kita akan menggunakan method `belongsToMany` pada kedua model seperti berikut:

app/Course.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Course extends Model {

    protected $fillable = ['title', 'units', 'room'];

    public function students()
    {
        return $this->belongsToMany('App\Student');
    }
}
```

app/Student.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Student extends Model {

    protected $fillable = ['name', 'date_of_birth', 'gender'];

    public function courses()
    {
        return $this->belongsToMany('App\Course');
    }
}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹²

Penggunaan

Penggunaan dari relasi ini hampir sama dengan relasi `hasMany`. Dari model `Course` kita dapat mengakses method `students()` dan menambahkan `fetch method` maupun `query constraint` yang kita butuhkan. Atau memanggil attribut `students` untuk mendapatkan semua model.

```
// Menampilkan semua nama student dalam array
>>> App\Course::find(1)->students()->lists('name');
=> [
    "Joni",
    "Dadang"
]
>>> App\Course::find(1)->students;
=> <Illuminate\Database\Eloquent\Collection #00000000440da33c000000007a502488> [
    <App\Student #00000000440da33b000000007a502488> {
        id: 1,
        name: "Joni",
        date_of_birth: "1990-08-14",
    }
]
```

¹¹²<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/461666d179e4f81c64fb2d0d871cc74089717c8f>

```
gender: "m",
created_at: "2015-06-01 15:37:14",
updated_at: "2015-06-01 15:37:14",
pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000440da33a0\00000007a502488> {
    course_id: 1,
    student_id: 1
},
<App\Student #00000000440da3380000000007a502488> {
    id: 2,
    name: "Dadang",
    date_of_birth: "1987-12-04",
    gender: "m",
    created_at: "2015-06-01 15:37:14",
    updated_at: "2015-06-01 15:37:14",
    pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000440da33d0\00000007a502488> {
        course_id: 1,
        student_id: 2
    }
}
]
```

Hal ini berlaku juga dari model Student, kita dapat menggunakan method `courses()` atau attribut `courses`:

```
// Mengambil total unit untuk semua course yang diambil
>>> App\Student::find(1)->courses()->sum('units');
=> "7"
>>> App\Student::find(1)->courses;
=> <Illuminate\Database\Eloquent\Collection #00000000440da331000000007a502488> [
    <App\Course #00000000440da329000000007a502488> {
        id: 1,
        title: "Kemampuan Menulis Dasar",
        units: 3,
        room: "B13",
        created_at: "2015-06-01 15:37:14",
        updated_at: "2015-06-01 15:37:14",
        pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000440da3c00\00000007a502488> {
            student_id: 1,
```

```
        course_id: 1
    }
},
<App\Course #00000000440da3c5000000007a502488> {
    id: 2,
    title: "Menggambar Manga Level 1",
    units: 2,
    room: "A10",
    created_at: "2015-06-01 15:37:14",
    updated_at: "2015-06-01 15:37:14",
    pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000440da3c30\00000007a502488> {
        student_id: 1,
        course_id: 2
    }
},
<App\Course #00000000440da3c7000000007a502488> {
    id: 3,
    title: "Mewarnai Manga Level 1",
    units: 2,
    room: "A01",
    created_at: "2015-06-01 15:37:14",
    updated_at: "2015-06-01 15:37:14",
    pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000440da3390\00000007a502488> {
        student_id: 1,
        course_id: 3
    }
}
]
```

Untuk menambah relasi dari `course` ke `student` (atau sebaliknya), kita dapat menggunakan method `attach()` dengan parameter model `student`:

```
>>> $course = App\Course::create(['title'=>'Laravel Basic', 'units'=>4, 'room'=>'U19']);
>>> $course->students()->attach(App\User::student(1));
>>> $course->students;
=> <Illuminate\Database\Eloquent\Collection #0000000051d12c7600000000669acaf4> [
    <App\Student #0000000051d12c7400000000669acaf4> {
        id: 1,
        name: "Joni",
        date_of_birth: "1990-08-14",
        gender: "m",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09",
        pivot: <Illuminate\Database\Eloquent\Relations\Pivot #0000000051d12c770\000000669acaf4> {
            course_id: 4,
            student_id: 1
        }
    }
]
```

Method `attach` juga menerima parameter berupa array berisi id dari `Student`. Ini sangat berguna jika kita hendak merelasikan banyak `student` ke `Course`:

```
>>> $course->students()->attach([2,3]);
>>> $course->students;
=> <Illuminate\Database\Eloquent\Collection #00000000154888e80000000051b6417b> [
    <App\Student #00000000154888e70000000051b6417b> {
        id: 1,
        name: "Joni",
        date_of_birth: "1990-08-14",
        gender: "m",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09",
        pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000154888e30\000000051b6417b> {
            course_id: 4,
            student_id: 1
        }
    },
    <App\Student #00000000154888e60000000051b6417b> {
        id: 2,
        name: "Dadang",
    }
]
```

```
date_of_birth: "1987-12-04",
gender: "m",
created_at: "2015-06-13 09:58:09",
updated_at: "2015-06-13 09:58:09",
pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000154888e20\000000051b6417b> {
    course_id: 4,
    student_id: 2
},
<App\Student #00000000154888e50000000051b6417b> {
    id: 3,
    name: "Yuni",
    date_of_birth: "1992-11-14",
    gender: "f",
    created_at: "2015-06-13 09:58:09",
    updated_at: "2015-06-13 09:58:09",
    pivot: <Illuminate\Database\Eloquent\Relations\Pivot #00000000154888e10\000000051b6417b> {
        course_id: 4,
        student_id: 3
    }
}
]
```

Untuk menghilangkan relasi, kita dapat menggunakan `detach()`. Sama seperti method `attach`, method ini menerima parameter berupa model atau berupa array berisi dari ID model. Misalnya kita panggil dari model `Course`:

```
>>> $course->students()->detach([1,3]);
>>> $course->students;
=> <Illuminate\Database\Eloquent\Collection #0000000043a853120000000037c9b76a> [
    <App\Student #0000000043a853180000000037c9b76a> {
        id: 2,
        name: "Dadang",
        date_of_birth: "1987-12-04",
        gender: "m",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09",
        pivot: <Illuminate\Database\Eloquent\Relations\Pivot #0000000043a8531\90000000037c9b76a> {
            course_id: 4,
```

```
        student_id: 2
    }
}
]
```

Proses ini (menambah/menghapus relasi) biasanya akan sering kita lakukan. Oleh karena itu, selain menggunakan cara manual menggunakan `attach` atau `detach`, Laravel juga menyediakan method `sync()` untuk menambah/menghapus relasi berdasarkan parameter baru yang diberikan. Misalnya, diawal kita memiliki relasi ke Student dengan ID 2, jika ingin menambah relasi untuk Student dengan ID 3, ini syntax yang akan kita jalankan:

```
>>> App\Course::find(4)->students()->sync([2,3]);
=> [
    "attached" => [
        3
    ],
    "detached" => [],
    "updated"   => []
]
>>> App\Course::find(4)->students;
=> <Illuminate\Database\Eloquent\Collection #0000000043a853e10000000037c9b76a> [
    <App\Student #0000000043a853e60000000037c9b76a> {
        id: 2,
        name: "Dadang",
        date_of_birth: "1987-12-04",
        gender: "m",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09",
        pivot: <Illuminate\Database\Eloquent\Relations\Pivot #0000000043a853\ e90000000037c9b76a> {
            course_id: 4,
            student_id: 2
        }
    },
    <App\Student #0000000043a853e30000000037c9b76a> {
        id: 3,
        name: "Yuni",
        date_of_birth: "1992-11-14",
        gender: "f",
        created_at: "2015-06-13 09:58:09",
        updated_at: "2015-06-13 09:58:09",
    }
]
```

```
pivot: <Illuminate\Database\Eloquent\Relations\Pivot #0000000043a853>
e80000000037c9b76a> {
    course_id: 4,
    student_id: 3
}
]
]
```

Disini, Laravel akan secara otomatis membiarkan relasi ke `Student` dengan ID 2 dan menambah relasi ke `Student` dengan ID 3.

Jika kita menjalankan syntax berikut:

```
>>> App\Course::find(4)->students()->sync([1,2]);
=> [
    "attached" => [
        1
    ],
    "detached" => [
        1 => 3
    ],
    "updated"   => []
]
```

Maka Laravel akan secara otomatis menghapus relasi ke `Student` dengan ID 3 dan menambah relasi ke `Student` dengan ID 1.

Has Many Through

Terkadang ada sebuah kondisi dimana kita merelasikan 3 model. Misalnya, model `Artist` akan memiliki banyak `Album` dan model `Album` akan memiliki banyak `Song`. Disini, bisa dikatakan sebuah `Artist` dapat memiliki banyak (*Has Many*) `Song` melalui (*Through*) `Album`. Kita dapat membuat struktur databasenya seperti berikut:

Table Artists

| id name genre |
|---------------------------------|
| 1 Coldplay Alternative Rock |
| 2 Avicii Electro House |

Table Albums

| id artist_id title released |
|------------------------------------|
| 1 1 Parachutes 2000-07-10 |
| 2 1 Ghost Stories 2014-04-16 |
| 3 2 True 2013-09-13 |

Table Songs

| id album_id title length |
|----------------------------------------|
| 1 1 Yellow 00:04:29 |
| 2 1 Don't Panic 00:02:17 |
| 3 2 Magic 00:04:45 |
| 4 2 A Sky Full of Stars 00:04:28 |
| 5 3 Wake Me Up 00:04:09 |

Konfigurasi

Untuk membuat relasi ini, kita akan menggunakan method `hasManyThrough` pada model `Artist`. Method ini memiliki dua parameter, pertama model yang menjadi perantara (`App\Album`) dan kedua model yang dijadikan target (`App\Song`).

Tentunya, sangat disarankan kita juga membuat relasi `hasMany` dari `Artist` ke `Album` dan dari `Album` ke `Song`. Kita juga disarankan membuat relasi `belongsTo` dari `Song` ke `Album` dan dari `Album` ke `Artist`.

Berikut syntax untuk ketiga model tersebut:

App/Artist.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Artist extends Model {

    protected $fillable = ['name', 'genre'];

    public function albums()
    {
        return $this->hasMany('App\Album');
    }

    public function songs()
    {
        return $this->hasManyThrough('App\Song', 'App\Album');
    }
}
```

app/Album.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Album extends Model {

    protected $fillable = ['artist_id', 'title', 'released'];

    public function artist()
    {
        return $this->belongsTo('App\Artist');
    }

    public function songs()
    {
        return $this->hasMany('App\Song');
    }
}
```

app/Song.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Song extends Model {

    protected $fillable = ['album_id', 'title', 'length'];

    public function album()
    {
        return $this->belongsTo('App\Album');
    }

}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹³

Penggunaan

Kita dapat menggunakan relasi `hasManyThrough` sebagaimana menggunakan relasi `hasMany`. Kita dapat menggunakan method `songs()` atau attribut `songs` dari model `Artist`. Misalnya kita ambil semua `Song` dari `Coldplay` (id 1):

```
>>> App\Artist::find(1)->songs;
=> <Illuminate\Database\Eloquent\Collection #00000000659efc29000000006b092876> [
    <App\Song #00000000659efcd60000000006b092876> {
        id: 1,
        album_id: 1,
        title: "Yellow",
        length: "00:04:29",
        created_at: "2015-06-02 01:10:50",
        updated_at: "2015-06-02 01:10:50",
        artist_id: 1
    },
    <App\Song #00000000659efcd7000000006b092876> {
```

¹¹³<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/505a274b6d3e9a033a395b05ca0d1cd946902350>

```
        id: 2,
        album_id: 1,
        title: "Don't Panic",
        length: "00:02:17",
        created_at: "2015-06-02 01:10:50",
        updated_at: "2015-06-02 01:10:50",
        artist_id: 1
    },
    <App\Song #00000000659efcd40000000006b092876> {
        id: 3,
        album_id: 2,
        title: "Magic",
        length: "00:04:45",
        created_at: "2015-06-02 01:10:50",
        updated_at: "2015-06-02 01:10:50",
        artist_id: 1
    },
    <App\Song #00000000659efcd5000000006b092876> {
        id: 4,
        album_id: 2,
        title: "A Sky Full of Stars",
        length: "00:04:28",
        created_at: "2015-06-02 01:10:50",
        updated_at: "2015-06-02 01:10:50",
        artist_id: 1
    }
]
```

Terlihat disini, meskipun kita tidak membuat field `artist_id` pada table `songs`, kita tetap berhasil mengambil semua `Song` untuk Coldplay.

Polymorphic Relations

Jenis relasi ini sangat bermanfaat ketika hendak membuat beberapa relasi `belongsTo` dari sebuah model ke beberapa model lain dalam satu relasi. Contoh penggunaannya, dalam media sosial seperti Facebook terdapat `Status`, `Photo` dan `Comment`. `Comment` ini dapat diberikan untuk `Status` maupun `Photo`.

Ada beberapa teknik yang bisa kita lakukan, pertama kita dapat membuat table `Comment` terpisah untuk menyimpan komentar pada `Status` dan `Photo`. Sehingga akan terdapat 4 table:

- `statuses`

- photos
- comment_status
- comment_photo

Meskipun teknik ini dapat kita lakukan, tapi akan lebih baik kita menyimpan komentar pada satu table. Meskipun model yang akan direlasikannya berbeda. Menggunakan relasi di Database, cukup rumit untuk melakukan relasi seperti ini. Disinilah kita dapat menggunakan **Polymorphic Relations** di Laravel. Sehingga, kita hanya butuh membuat 3 table:

- statuses
- photos
- comments

Seperti kebanyakan relasi di Laravel pada umumnya, kita harus mengikuti aturan yang telah ditetapkan oleh Laravel untuk membuat field pada relasi ini. Jika relasi ini akan kita namakan `commentable`, maka pada table `comments` harus terdapat :

- field `commentable_type` yang berisi nama model yang berelasi.
- field `commentable_id` yang berisi ID dari model yang berelasi.

Pembuatan kedua field ini dapat kita buat secara manual maupun otomatis dengan Schema Builder menggunakan method `$table->morphs('nama_relasi');`. Pada kasus ini, pada method `up` di migrasi untuk table `comments`, kita harus menambahkan baris:

```
$table->morphs('commentable');
```

Untuk membuat contoh kasus ini lebih nyata, mari kita tambahkan juga model User. Sehingga hasil akhir dari struktur databasenya, akan seperti berikut:

Table Users

| id | name | email | password |
|----|------|----------------|----------------------------------|
| 1 | Dini | dini@gmail.com | \$2y\$10\$LMxDSR62XqshY20fLfo... |
| 2 | Dani | dani@gmail.com | \$2y\$10\$DMmNi3uMU44p0DCKjMk... |
| 3 | Deon | deon@gmail.com | \$2y\$10\$AFJUxYd5jNX5bMvi.a9... |

Table Statuses

| id content user_id |
|-------------------------------------------|
| 1 Hari ini #belajar Laravel 5! 1 |
| 2 Cari framework PHP? Laravel aja.. 2 |

Table Photos

| id title filename user_id |
|--------------------------------------------------|
| 1 Buku Pranikah Recomended BVOMDFeE2.jpg 1 |
| 2 Memulai Hidup Baru VN74aUmLD.jpg 2 |
| 3 Rumah Impian HVpdKBe.jpg 2 |

Table Comments

| id content user_id commentable_id commentable_type |
|---------------------------------------------------------------|
| 1 Jadikan aku muridmu guru.. :o 3 1 App\Stat\us |
| 2 Ini juga baru belajar mas.. 1 1 App\Stat\us |
| 3 Ahh.. merendah untuk meninggi nih.. 3 1 App\Stat\us |
| 4 +1 like this banget dah.. 3 2 App\Stat\us |
| 5 Kalau cari jodoh? Hahaha.. :v 1 2 App\Stat\us |
| 6 Catet ah.. 2 1 App\Phot\o |
| 7 Belinya dimana nih? 3 1 App\Phot\o |

| | | |
|-----------------------------------------|---|---------------|
| 8 Di Gramedia ada koq. | 1 | 1 App\Phot\ |
| o | | |
| 9 Akhirnya nikah juga. Selamat ya! :D | 1 | 2 App\Phot\ |
| o | | |
| 10 Sekarang tinggal dimana? | 3 | 2 App\Phot\ |
| o | | |
| 11 Di Bandung.. :) | 2 | 2 App\Phot\ |
| o | | |
| +-----+-----+-----+-----+-----+\ | | |
| -----+ | | |

Konfigurasi

Untuk membuat relasi ini, pada table yang memiliki relasi morph-to-many (`comments`) kita perlu menambah relasi `morphTo` dengan nama method harus sesuai dengan nama field relasi di table `comments`. Karena kita menggunakan field `commentable_id` dan `commentable_type`, maka nama methodnya harus bernama `commentable`. Syntax akhirnya akan seperti berikut:

App/Comment.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Comment extends Model {

    protected $fillable = ['content', 'user_id', 'commentable_id', 'commentable\_type'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }

    public function commentable()
    {
        return $this->morphTo();
    }

}
```

Sementara pada model yang berelasi pada model `Comment`, kita harus menggunakan relasi `morphMany` dengan parameter berupa nama model (`App\Comment`) dan nama field relasi (`commentable`). Hasil akhir dari model `Status` dan `Photo` akan seperti berikut:

App/Status.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Status extends Model {

    protected $fillable = ['content', 'user_id'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }

    public function comments()
    {
        return $this->morphMany('App\Comment', 'commentable');
    }
}
```

App/Photo.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Photo extends Model {

    protected $fillable = ['title', 'filename', 'user_id'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }

    public function comments()
    {
        return $this->morphMany('App\Comment', 'commentable');
    }
}
```

```
    }  
}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹⁴

Penggunaan

Untuk mengakses relasi ini, sama halnya sebagaimana kita mengakses relasi one-to-many. Kita dapat mendapatkan semua Comment dari sebuah status menggunakan method `comments()` atau attribut `comments`:

```
>>> App>Status::find(1)->comments;  
=> <Illuminate\Database\Eloquent\Collection #00000000101e03b80000000062b3e677> [  
    <App\Comment #00000000101e03b90000000062b3e677> {  
        id: 1,  
        content: "Jadikan aku muridmu guru.. :o",  
        user_id: 3,  
        commentable_id: 1,  
        commentable_type: "App\\Status",  
        created_at: "2015-06-07 14:40:54",  
        updated_at: "2015-06-07 14:40:54"  
    },  
    <App\Comment #00000000101e03ba0000000062b3e677> {  
        id: 2,  
        content: "Ini juga baru belajar mas.. ",  
        user_id: 1,  
        commentable_id: 1,  
        commentable_type: "App\\Status",  
        created_at: "2015-06-07 14:40:54",  
        updated_at: "2015-06-07 14:40:54"  
    },  
    <App\Comment #00000000101e03bb0000000062b3e677> {  
        id: 3,  
        content: "Ahh.. merendah untuk meninggi nih..",  
        user_id: 3,  
        commentable_id: 1,  
        commentable_type: "App\\Status",
```

¹¹⁴<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/0e684232ed05101ed8fd52fda51482d5d6fba426>

```

    created_at: "2015-06-07 14:40:54",
    updated_at: "2015-06-07 14:40:54"
}
]
```

Untuk menambah Comment dari status (atau Photo) kita dapat menggunakan method `save()`:

```

>>> App\Status::find(1)->comments()->save(
    App\Comment::create([
        'content'=>'Ahh.. ngga juga..',
        'user_id'=>1
    ]));
=> <App\Comment #00000000101e034a0000000062b3e677> {
    content: "Ahh.. ngga juga..",
    user_id: 1,
    updated_at: "2015-06-07 15:25:10",
    created_at: "2015-06-07 15:25:10",
    id: 12,
    commentable_type: "App\\Status",
    commentable_id: 1
}
```

Jika ingin menambahkan beberapa Comment secara sekaligus kita dapat menggunakan method `saveMany()`:

```

>>> App\Status::find(2)->comments()->saveMany([
    App\Comment::create(['content'=>'wahh.. asyik ya..', 'user_id'=>3]),
    App\Comment::create(['content'=>'Kapan-kapan saya main ya! :) ', 'user_id'=>1\
])
]);
=> [
    <App\Comment #00000000101e03600000000062b3e677> {
        content: "wahh.. asyik ya..",
        user_id: 3,
        updated_at: "2015-06-07 15:30:51",
        created_at: "2015-06-07 15:30:51",
        id: 13,
        commentable_type: "App\\Status",
        commentable_id: 2
    },
    <App\Comment #00000000101e036c0000000062b3e677> {
```

```

        content: "Kapan-kapan saya main ya! :)",
        user_id: 1,
        updated_at: "2015-06-07 15:30:51",
        created_at: "2015-06-07 15:30:51",
        id: 14,
        commentable_type: "App\\Status",
        commentable_id: 2
    }
]

```

Self Referencing Relationship

Terkadang kita membutuhkan sebuah model berrelasi ke dirinya sendiri. Contohnya relasi antara orang tua dan anak. Keduanya merupakan model Person (orang).

Struktur tablenya akan seperti ini:

Table Peolpe

| id | name | birth_date | parent_id |
|----|--------|------------|-----------|
| 1 | Budi | 1971-03-10 | NULL |
| 2 | Deni | 1992-08-10 | 1 |
| 3 | Dadang | 1993-09-02 | 1 |

Catatan: Disini kita menggunakan table People karena plural (jamak) dari Person adalah People.

Konfigurasi

Untuk membuat relasi seperti ini, kita harus menambah field khusus misalnya `parent_id` yang akan digunakan untuk menentukan orang tua dan anak.

Kita juga harus menggunakan relasi `belongsTo` (untuk menentukan orang tua) dan `hasMany` (untuk menentukan anak). Tentunya, kita juga pada kedua method ini, kita harus menambah field `parent_id` sebagai custom foreign key.

Hasil akhir modelnya akan seperti berikut:

App/Person.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Person extends Model {

    protected $fillable = ['name', 'birth_date', 'parent_id'];

    public function getDates()
    {
        return ['created_at', 'updated_at', 'birth_date'];
    }

    public function parent()
    {
        return $this->belongsTo('App\Person', 'parent_id');
    }

    public function childs()
    {
        return $this->hasMany('App\Person', 'parent_id');
    }
}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹⁵

Penggunaan

Kita dapat mengakses orang tua dari seorang Person dengan method parent() atau attribut parent:

¹¹⁵<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/0954f466d867c304ef959e832f1c71b3ba5ac003>

```
>>> App\Person::find(2)->parent;  
=> <App\Person #00000001bfb357500000005c31ce49> {  
    id: 1,  
    name: "Budi",  
    birth_date: "1971-03-10",  
    parent_id: null,  
    created_at: "2015-06-07 15:58:28",  
    updated_at: "2015-06-07 15:58:28"  
}
```

Ketika field `parent_id` tidak diisi, maka Laravel akan membuat relasinya bernilai `null`:

```
>>> App\Person::find(1)->parent;  
=> null
```

Untuk mencari anak, kita dapat menggunakan `childs`:

```
>>> App\Person::find(1)->childs  
=> <Illuminate\Database\Eloquent\Collection #00000004b632b9d000000002d28f26a> [  
    <App\Person #00000004b632b9a000000002d28f26a> {  
        id: 2,  
        name: "Deni",  
        birth_date: "1992-08-10",  
        parent_id: 1,  
        created_at: "2015-06-07 15:58:28",  
        updated_at: "2015-06-07 15:58:28"  
    },  
    <App\Person #00000004b632b9b000000002d28f26a> {  
        id: 3,  
        name: "Dadang",  
        birth_date: "1993-09-02",  
        parent_id: 1,  
        created_at: "2015-06-07 15:58:28",  
        updated_at: "2015-06-07 15:58:28"  
    }  
]
```

Sebagaimana relasi `hasMany` pada umumnya, untuk menambahkan relasi kita dapat menggunakan `save()` (untuk satu model):

```
>>> App\Person::find(2)->childs()->save(App\Person::create([ 'name'=>'Tina', 'bir\\
th_date'=>'2014-06-10']));
=> <App\Person #000000001fb3583000000005c31ce49> {
    name: "Tina",
    birth_date: "2014-06-10 00:00:00",
    updated_at: "2015-06-07 16:17:50",
    created_at: "2015-06-07 16:17:50",
    id: 4,
    parent_id: 2
}
```

Maupun saveMany() (untuk banyak model):

```
>>> App\Person::find(2)->childs()->saveMany([
    App\Person::create(['name'=>'Rudi', 'birth_date'=>'2014-05-10']),
    App\Person::create(['name'=>'Dodi', 'birth_date'=>'2014-01-11'])
]);
=> [
    <App\Person #000000001fb35aa000000005c31ce49> {
        name: "Rudi",
        birth_date: "2014-05-10 00:00:00",
        updated_at: "2015-06-07 16:20:18",
        created_at: "2015-06-07 16:20:18",
        id: 5,
        parent_id: 2
    },
    <App\Person #000000001fb359e000000005c31ce49> {
        name: "Dodi",
        birth_date: "2014-01-11 00:00:00",
        updated_at: "2015-06-07 16:20:18",
        created_at: "2015-06-07 16:20:18",
        id: 6,
        parent_id: 2
    }
]
```

Many To Many Polymorphic Relations

Relasi ini akan sangat berguna ketika kita ingin menggunakan relasi many-to-many tapi dengan model relasi yang berubah. Contoh klasik penggunaan relasi ini adalah untuk fitur Tag di blog. Pada sebuah blog, selain pada Article, kita juga dapat menambahkan

Tag pada jenis konten lain, misalnya Video. Tentunya, kita harus menggunakan relasi many-to-many untuk membuat fitur ini. Menggunakan skema many-to-many biasa, kita akan membuat 5 table:

- articles
- videos
- tags
- tag_video
- article_tag

Disini, table tag_video dan article_tag, digunakan untuk menambah relasi many-to-many dari Tag ke Article dan Video. Solusinya, kurang efektif ketika spesifikasi sistem belum jelas. Bisa jadi, kedepannya kita akan menambah tipe konten lain yang dapat di tag, misalnya Audio. Dan untuk menambahkan Tag ke konten ini, kita perlu membuat table audio_tag.

Disinilah kita dapat menggunakan relasi **Many to Many Polymorphic**. Dengan relasi ini, kita hanya perlu membuat 4 table:

- articles
- videos
- tags
- taggables

Disini, table taggables yang akan digunakan untuk menyimpan data relasi.

Misalnya kita memiliki table articles, videos dan tags dengan konten seperti berikut:

Table Articles

| id | title | content |
|----|-----------------------------------------|------------------------------|
| 1 | Multi Gateway Payment dengan Omnipay | Laborum aliuid culpa reru... |
| 2 | Brunch, Alternatif Gulp dan Grunt | Omnis assumnda culpa sed ... |
| 3 | 5 Kesalahan Fatal dalam Memulai Startup | Laboriosam ui sint nemo d... |

Table Videos

| id | title | length | filename |
|----|-------------------------------------|----------|--------------|
| 1 | Method setUp dalam PHPUnit | 00:08:25 | 57a8d3f2.mp4 |
| 2 | Interview dengan founder Wunderlist | 00:14:05 | 22dc1f05.mp4 |

Table Tags

| id | name |
|----|---------|
| 1 | Design |
| 2 | Startup |
| 3 | Backend |
| 4 | Testing |
| 5 | HTML |

Maka, table `taggables` akan memiliki konten seperti berikut:

Table Taggables

| tag_id | taggable_id | taggable_type |
|--------|-------------|---------------|
| 3 | 1 | App\Article |
| 1 | 2 | App\Article |
| 5 | 2 | App\Article |
| 2 | 3 | App\Article |
| 3 | 1 | App\Video |
| 4 | 1 | App\Video |
| 2 | 2 | App\Video |

Konfigurasi

Untuk membuat relasi ini, kita perlu menambahkan setiap model yang berrelasi ke model `Tag` dengan nama relasi masing dengan method `morphedByMany()`. Pada contoh ini, kita harus membuat relasi untuk `articles` dan `videos`:

App/Tag.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Tag extends Model
{

    protected $fillable = ['name'];

    public function articles()
    {
        return $this->morphedByMany('App\Article', 'taggable');
    }

    public function videos()
    {
        return $this->morphedByMany('App\Video', 'taggable');
    }

}
```

Sementara pada model Article dan Video, kita harus membuat relasi tags dengan method morphToMany():

App/Article.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Article extends Model
{

    protected $fillable = ['title', 'content'];

    public function tags()
    {
        return $this->morphToMany('App\Tag', 'taggable');
    }

}
```

App/Video.php

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Video extends Model
{

    protected $fillable = ['title', 'length', 'filename'];

    public function tags()
    {
        return $this->morphToMany('App\Tag', 'taggable');
    }

}
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹¹⁶

Penggunaan

Untuk menggunakan relasi ini, sama dengan menggunakan relasi many-to-many yang telah kita pelajari sebelumnya. Kita dapat memanggil semua Article yang berelasi pada Tag dengan perintah berikut

```
>>> App\Tag::find(2)->articles;
=> <Illuminate\Database\Eloquent\Collection #0000000051d12c6300000000669acaf4> [
    <App\Article #0000000051d12c6500000000669acaf4> {
        id: 3,
        title: "5 Kesalahan Fatal dalam Memulai Startup",
        content: "Laboriosam qui sint nemo...",
        created_at: "2015-06-13 09:58:10",
        updated_at: "2015-06-13 09:58:10",
        pivot: <Illuminate\Database\Eloquent\Relations\MorphPivot #0000000051d1 \
2c6200000000669acaf4> {
            tag_id: 2,
```

¹¹⁶<https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/d8ad91b2a23e2cb190ce522b4957a4ffdd7208f8>

```
    taggable_id: 3
  }
]
>>>
```

Ataupun sebaliknya, mencari Tag untuk sebuah Article:

```
>>> App\Article::find(2)->tags;
=> <Illuminate\Database\Eloquent\Collection #000000051d12c670000000669acaf4> [
  <App\Tag #000000051d12c980000000669acaf4> {
    id: 1,
    name: "Design",
    created_at: "2015-06-13 09:58:10",
    updated_at: "2015-06-13 09:58:10",
    pivot: <Illuminate\Database\Eloquent\Relations\MorphPivot #000000051d1 \
2c9900000000669acaf4> {
      taggable_id: 2,
      tag_id: 1
    }
  },
  <App\Tag #000000051d12c9b00000000669acaf4> {
    id: 5,
    name: "HTML",
    created_at: "2015-06-13 09:58:10",
    updated_at: "2015-06-13 09:58:10",
    pivot: <Illuminate\Database\Eloquent\Relations\MorphPivot #000000051d1 \
2c6600000000669acaf4> {
      taggable_id: 2,
      tag_id: 5
    }
  }
]
```

Tentunya, hal yang sama juga berlaku untuk model Video.

Untuk menambah/menghapus relasi kita dapat menggunakan method attach(), detach() atau sync(). Silahkan buka kembali topik tentang relasi many-to-many untuk mempelajarinya.

Custom Key untuk Relationship

Sebagaimana dijelaskan sebelumnya, Eloquent menggunakan aturan khusus untuk penamaan field untuk relationship. Pada contoh relasi one-to-one yang telah kita

pelajari, Eloquent akan menganggap pada model `User` akan menggunakan primary key bernama `id` dan pada model `Preference` akan terdapat field `user_id`.

Jika kita hendak field yang kita gunakan untuk relasi, kita dapat menggunakan format berikut:

```
$this->NamaRelasi('model', 'nama_foreign_key', 'nama_local_key');
```

Misalnya, jika kita menggunakan field `account_id` pada table `preference`, maka pada model `User` kita ubah syntax relasinya menjadi:

App/User.php

```
....  
public function preference()  
{  
    return $this->hasOne('App\Preference', 'account_id');  
}  
....
```

Sedangkan pada model `Preference` kita ubah menjadi:

App/Preference.php

```
....  
public function user()  
{  
    return $this->belongsTo('App\User', 'account_id');  
}  
....
```

Berikut saya sertakan link ke API Laravel untuk menggunakan custom key pada jenis relasi lain:

- `hasOne`¹¹⁷
- `morphOne`¹¹⁸
- `belongsToMany`¹¹⁹
- `morphTo`¹²⁰

¹¹⁷ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_hasOne

¹¹⁸ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_morphOne

¹¹⁹ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_belongsToMany

¹²⁰ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_morphTo

- `hasMany121`
- `hasManyThrough122`
- `morphMany123`
- `belongsToMany124`
- `morphToMany125`
- `morphedByMany126`



Source code dari latihan ini bisa didapat di [Bitbucket¹²⁷](#)

Eager Loading

Fitur eager loading akan sangat bermanfaat ketika kita bekerja dengan relasi di model. Menggunakan fitur ini dengan tepat akan membuat aplikasi yang kita bangun lebih efisien. Untuk memahami konsep ini, mari kita gunakan contoh relasi `one-to-many` yang telah kita pelajari sebelumnya.

Pada contoh tersebut, sebuah `Studio` dapat memiliki banyak `Movie`. Mari kita tampilkan semua nama `Studio` berikut `Movie` yang berelasinya di browser. Tanpa menggunakan Eager Loading, ini syntax yang akan kita buat pada view:

resources/views/studios.blade.php

```
<html>
  <head>
    <title>Studio</title>
  </head>
  <body>
    @foreach (App\Studio::all() as $studio)
      <h1>{{ $studio->name }}</h1>
      <ul>
        @forelse ($studio->movies as $movie)
          <li>{{ $movie->name }}</li>
        @empty
      
```

¹²¹ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_hasMany

¹²² http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_hasManyThrough

¹²³ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_morphMany

¹²⁴ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_belongsToMany

¹²⁵ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_morphToMany

¹²⁶ http://laravel.com/api/5.1/Illuminate/Database/Eloquent/Model.html#method_morphedByMany

¹²⁷ <https://bitbucket.org/rahmatawaludin/sample-model-relationship/commits/03b8738de896821575b45fe7404ae2cf79a6ee1e>

```
Tidak ada film untuk studio ini.  
@endforelse  
</ul>  
@endforeach  
</body>  
</html>
```

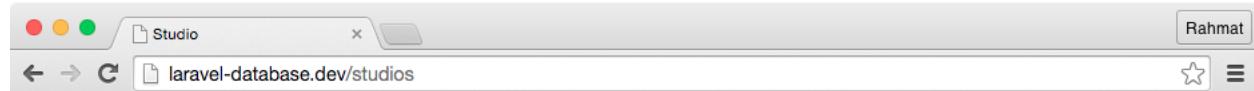
Pada syntax ini kita akan menampilkan semua Studio. Untuk setiap Studio yang ditampilkan, kita akan menampilkan semua Movie yang berelasi menggunakan forelse looping. Jika tidak ada Movie pada Studio tersebut, kita akan menampilkan Tidak ada film untuk studio ini..

Tentunya, kita harus memanggil view ini dari route:

app/Http/routes.php

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

Kini, mari kita coba akses url /studios:



Pixar

- Cars 2
- Finding Nemo

DreamWorks Animation

- Kung Fu Panda 2

Universal Studio

- Despicable Me
- Fast & Furious 7

20th Century Fox

Tidak ada film untuk studio ini.

Menampilkan semua Studio berikut Movie

Query ini terlihat normal dan semua berjalan lancar. Tapi, dibalik layar, cukup banyak query yang dilakukan. Kita dapat mengeceknya dengan menambahkan syntax berikut pada file route:

app/Http/routes.php

```
....  
DB::listen(function ($sql) {  
    var_dump($sql);  
});
```

Jika kita cek kembali url /studios:



Pixar

```
string 'select * from `movies` where `movies`.`studio_id` = ? and `movies`.`studio_id` is not null' (length=90)  
• Cars 2  
• Finding Nemo
```

DreamWorks Animation

```
string 'select * from `movies` where `movies`.`studio_id` = ? and `movies`.`studio_id` is not null' (length=90)  
• Kung Fu Panda 2
```

Universal Studio

```
string 'select * from `movies` where `movies`.`studio_id` = ? and `movies`.`studio_id` is not null' (length=90)  
• Despicable Me  
• Fast & Furious 7
```

20th Century Fox

```
string 'select * from `movies` where `movies`.`studio_id` = ? and `movies`.`studio_id` is not null' (length=90)  
Tidak ada film untuk studio ini.
```

Menampilkan semua query

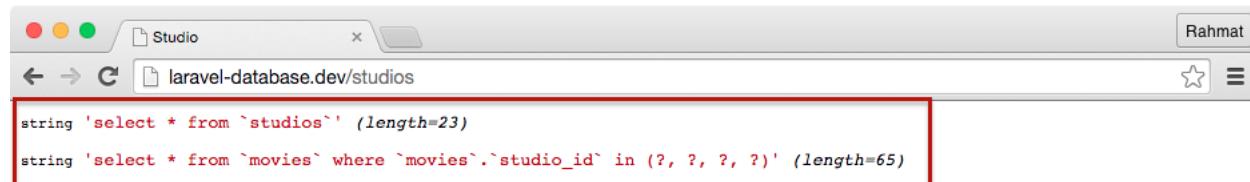
Terlihat disini, ada 5 query database yang kita lakukan. Ini terjadi karena kita melakukan query baru setiap kali mencari Movie yang berelasi untuk setiap Studio yang ditampilkan. Tentunya, jika terdapat lebih banyak Studio, akan semakin banyak query yang kita lakukan.

Eager loading menyelesaikan masalah ini dengan menggunakan [operator IN¹²⁸](#) pada SQL untuk mencari semua model `Movie` yang berelasi ketika kita mencari model `Studio`. Untuk menggunakan fitur ini, kita harus menggunakan method `with()` dengan parameter nama relasi. Pada contoh ini, kita akan *me-load* relasi `movies`, sehingga syntax pada view akan berubah menjadi:

resources/views/studios.blade.php

```
....  
<body>  
 @foreach (App\Studio::with('movies')->get() as $studio)  
 <h1>{{ $studio->name }}</h1>  
....
```

Kini, jika kita mencoba mengakses url `/studios`:



Pixar

- Cars 2
- Finding Nemo

DreamWorks Animation

- Kung Fu Panda 2

Universal Studio

- Despicable Me
- Fast & Furious 7

20th Century Fox

Tidak ada film untuk studio ini.

Menggunakan Eager Loading

Dapat kita lihat, hanya 2 query yang dijalankan. **Wow!**

Dengan teknik ini, kita dapat menghemat banyak query database. Yang pada akhirnya akan membuat aplikasi kita menjadi lebih cepat.

¹²⁸http://www.w3schools.com/sql/sql_in.asp

Oh iya, jika diinginkan, kita juga dapat me-*load* beberapa relasi sekaligus. Misalnya model `Studio` memiliki relasi `City` dan `Staff`, kita dapat me-*load* nya secara sekaligus dengan syntax:

```
App\Studio::with('city', 'staff')->get();
```

Begitupun jika terjadi *nested relationship*, misalnya dari model `Movie` berelasi ke table `Actor`. Kita dapat me-*load* keduanya dengan syntax:

```
App\Studio::with('movies.actors')->get();
```

Selain menggunakan eager loading diawal, kita juga dapat memanggil eager loading setelah model kita dapatkan. Ini akan berguna ketika kita ingin memanggil eager loading ini secara dynamis. Untuk memanggilnya kita dapat menggunakan method `load()`:

```
$studio = App\Studio::find(1);  
$studio->load('movies');
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹²⁹



Ringkasan

Di bab ini kita telah mempelajari penggunaan berbagai jenis relationship di Eloquent. Semua relasi ini ditujukan untuk memudahkan kita selama membangun aplikasi. Tentunya, jika terdapat kasus yang tidak bisa diselesaikan dengan relationship di Eloquent, kita selalu bisa menggunakan Query Builder atau Raw Query.

Dari beberapa jenis relasi `x-to-many`, biasanya hasil yang kita dapatkan berupa Collection. Apa itu? Mari kita pelajari di bab selanjutnya. Semangat!

¹²⁹ <https://bitbucket.org/rahmatawalidin/sample-model-relationship/commits/3d1c9a9c6dbf63825e3eeb0c0ee6e35b01afde94>

Collection dengan Eloquent

Ketika buku ini ditulis, di Indonesia sedang demam batu akik. Banyak orang **mengoleksi** batu ini, dari yang muda hingga yang tua. Kalau sudah punya banyak **koleksi** batu, biasanya menjadi nilai tersendiri bagi pemiliknya. Maklum, harga batu akik sendiri bisa sangat mahal hingga ratusan juta.

Di bab ini, kita akan membahas tentang **koleksi**. Tentunya, bukan koleksi batu akik.. :D

Koleksi yang akan kita bahas adalah koleksi pada Eloquent.

Class Collection

Di Laravel, setiap kali kita melakukan query ke database maka kita akan mendapatkan instance dari `Illuminate\Support\Collection`¹³⁰ atau childnya. Mari kita gunakan sample data dari bab sebelumnya untuk membuktikan hal ini:

```
>>> $collection = App\Studio::all();
>>> get_class($collection);
=> "Illuminate\\Database\\Eloquent\\Collection"
>>> get_parent_class($collection);
=> "Illuminate\\Support\\Collection"
```

Terlihat disini, query `all` pada model `Studio` menghasilkan instance dari `Illuminate\Database\Eloquent\Collection` yang merupakan child dari `Illuminate\Support\Collection`.

`Collection` mengadopsi [Iterator Pattern](#)¹³¹. Dalam pattern ini terdapat istilah Element (object di dalam iterator) dan Container (class yang berfungsi sebagai iterator). Biasanya element dari iterator adalah object yang memiliki tipe berbeda. Tujuan akhir dari pattern ini adalah menyeragamkan method untuk mengelola semua elemen ini, meskipun tipenya berbeda.

Membuat Collection

Selain dari hasil query pada model, kita juga dapat membuat collection baru dengan membuat instance dari `Illuminate\Support\Collection`:

¹³⁰ <http://laravel.com/api/5.1/Illuminate/Support/Collection.html>

¹³¹ https://en.wikipedia.org/wiki/Iterator_pattern

```
>>> $collection = new Illuminate\Support\Collection;  
=> <Illuminate\Support\Collection #00000000151ec920000000000be123c2> []
```

Untuk menambahkan elemen ke dalam Collection, kita dapat menggunakan method `push()`. Misalnya, kita tambahkan beberapa nama bahasa pemrograman:

```
>>> $collection->push('php');  
>>> $collection->push('ruby');  
>>> $collection->push('java');
```

Pada contoh ini, kita hanya menambahkan elemen baru berupa string. Tentunya, kita dapat menambah element baru dengan tipe lainnya misalnya array, object, dll.

```
>>> $collection->push(['nama'=> 'Andi']);  
>>> $collection->push(App\Movie::find(1));
```

Untuk menampilkan semua elemennya, kita dapat menggunakan method `all()`:

```
>>> $collection->all();  
=> [  
    "php",  
    "ruby",  
    "java",  
    [  
        "nama" => "Andi"  
    ],  
    <App\Movie #00000000151ec950000000000be123c2> {  
        id: 1,  
        name: "Cars 2",  
        date_released: "2011-06-24",  
        studio_id: 1,  
        created_at: "2015-06-13 15:21:24",  
        updated_at: "2015-06-13 15:21:24"  
    }  
]
```

Selain menggunakan cara manual seperti diatas, kita juga dapat membuat Collection baru dengan helper `collect()`:

```
>>> $collection = collect(['Bandung', 'Jakarta', 'Surabaya']);  
=> <Illuminate\Support\Collection #00000000151ec6a000000000be123c2> [  
    "Bandung",  
    "Jakarta",  
    "Surabaya"  
]
```

Method pada Collection

Method yang disediakan oleh Collection cukup banyak, kita dapat melihat semuanya di [API Collection¹³²](#) atau [dokumentasi resmi¹³³](#). Pada bagian ini, kita tidak akan mempelajari semuanya. Cukup beberapa method yang sering dipakai dan contoh pengaplikasianya.

Jika ada method Collection yang belum dibahas disini dan Anda kebingungan untuk menggunakannya, silahkan kontak saya. Dengan senang hati saya akan menambahkan materi tersebut ke bab ini.

All

Cukup jelas. Method ini digunakan untuk menampilkan semua element dari Collection. Karena sudah sering kita pakai, saya tidak akan membuat contoh untuk method ini.

Push

Method `push()` digunakan untuk menambah element pada Collection:

```
>>> $balon = collect(['merah', 'kuning', 'kelabu', 'hijau muda']);  
>>> $balon->push('biru');  
>>> $balon->all();  
=> [  
    "merah",  
    "kuning",  
    "kelabu",  
    "hijau muda",  
    "biru"  
]
```

¹³²<http://laravel.com/api/5.1/Illuminate/Support/Collection.html>

¹³³<http://laravel.com/docs/5.1/eloquent-collections>

Put

Method `put()` digunakan untuk menambah element Collection berdasarkan `key` dan `value`:

```
>>> $framework = collect(['php'=>'Laravel']);
>>> $framework->put('ruby', 'Ruby on Rails');
>>> $framework->all();
=> [
    "php" => "Laravel",
    "ruby" => "Ruby on Rails"
]
```

Pop

Method `pop()` digunakan untuk menghapus element Collection berdasarkan `key`:

```
>>> $framework = collect(['php'=>'Laravel', 'ruby'=>'Ruby on Rails']);
>>> $framework->pop('ruby');
>>> $framework->all();
=> [
    "php" => "Laravel"
]
```

Pull

Method `pull()` digunakan untuk menghapus element Collection berdasarkan `key`-nya dan memberikan element tersebut:

```
>>> $balon = collect(['merah', 'kuning', 'kelabu', 'hijau muda', 'biru']);
>>> $meletus = $balon->pull(3);
>>> $meletus;
=> "hijau muda"
>>> $balon->all();
=> [
    0 => "merah",
    1 => "kuning",
    2 => "kelabu",
    4 => "biru"
]
```

ToArray /ToJson

Method ini berguna untuk merubah Collection menjadi Array atau JSON. Sangat bermanfaat ketika membangun API.

```
>>> App\Movie::all()->toArray();
=> [
    [
        "id"          => 1,
        "name"        => "Cars 2",
        "date_released" => "2011-06-24",
        "studio_id"   => 1,
        "created_at"  => "2015-06-13 15:21:24",
        "updated_at"  => "2015-06-13 15:21:24"
    ],
    [
        "id"          => 2,
        "name"        => "Finding Nemo",
        "date_released" => "2003-05-30",
        "studio_id"   => 1,
        "created_at"  => "2015-06-13 15:21:24",
        "updated_at"  => "2015-06-13 15:21:24"
    ],
    ...
]
```

First

Mendapatkan element pertama dari Collection:

```
>>> $movie = App\Movie::all();
>>> $movie->first();
=> <App\Movie #000000000151ec62000000000be123c2> {
    id: 1,
    name: "Cars 2",
    date_released: "2011-06-24",
    studio_id: 1,
    created_at: "2015-06-13 15:21:24",
    updated_at: "2015-06-13 15:21:24"
}
```

Last

Mendapatkan element terakhir dari Collection:

```
>>> $movie = App\Movie::all();
>>> $movie->last();
=> <App\Movie #00000000151ec66000000000be123c2> {
    id: 5,
    name: "Fast & Furious 7",
    date_released: "2015-03-16",
    studio_id: 3,
    created_at: "2015-06-13 15:21:24",
    updated_at: "2015-06-13 15:21:24"
}
```

Each

Mirip dengan fungsi `foreach` di PHP, method ini digunakan untuk melakukan looping terhadap element dari Collection dengan parameter berupa closure. Misalnya kita ingin menampilkan judul teks "Film" sebelum nama film:

```
>>> $movie = App\Movie::all();
>>> $movie->each(function($movie) {
    var_dump('Film ' . $movie->name);
});
string(11) "Film Cars 2"
string(17) "Film Finding Nemo"
string(20) "Film Kung Fu Panda 2"
string(18) "Film Despicable Me"
string(21) "Film Fast & Furious 7"
```

Transform

Method ini hampir sama dengan method `each()`. Perbedaannya, isi Collection akan dirubah berdasarkan nilai yang kita dapatkan dari closure.

```
>>> $movie = App\Movie::all();
>>> $movie->transform(function($movie){
    return 'Film ' . $movie->name;
});
>>> $movie->all();
=> [
    "Film Cars 2",
    "Film Finding Nemo",
    "Film Kung Fu Panda 2",
    "Film Despicable Me",
    "Film Fast & Furious 7"
]
```

Terlihat disini, Collection \$movie telah berubah menjadi array string, bukan model Movie.

Map

Hampir mirip dengan method transform(). Perbedaannya, method ini akan membuat Collection baru, sedangkan Collection asli tidak akan berubah nilainya.

```
>>> $movie = App\Movie::all();
>>> $film = $movie->map(function($movie){ return 'Film ' . $movie->name; });
>>> $film->all();
=> [
    "Film Cars 2",
    "Film Finding Nemo",
    "Film Kung Fu Panda 2",
    "Film Despicable Me",
    "Film Fast & Furious 7"
]
>>> $movie->all();
=> [
    <App\Movie #000000000151ec670000000000be123c2> {
        id: 1,
        name: "Cars 2",
        date_released: "2011-06-24",
        studio_id: 1,
        created_at: "2015-06-13 15:21:24",
        updated_at: "2015-06-13 15:21:24"
    },
    <App\Movie #000000000151ec640000000000be123c2> {
```

```
    id: 2,
    name: "Finding Nemo",
    date_released: "2003-05-30",
    studio_id: 1,
    created_at: "2015-06-13 15:21:24",
    updated_at: "2015-06-13 15:21:24"
},
...
]
```

Terlihat disini variable `$film` telah berisi Collection baru hasil dari method `map` pada Collection `$movie`. Sementara variable `$movie` tidak berubah.

Filter

Sesuai namanya, method ini digunakan untuk menyaring element berdasarkan kriteria tertentu. Method ini akan mengembalikan Collection baru, tanpa mengubah Collection aslinya.

```
>>> $siswa = App\Student::all();
=> <Illuminate\Database\Eloquent\Collection #00000000151ec6900000000be123c2> [
    <App\Student #00000000151ec9500000000be123c2> {
        id: 1,
        name: "Joni",
        date_of_birth: "1990-08-14",
        gender: "m"
    },
    <App\Student #00000000151ec9a000000000be123c2> {
        id: 2,
        name: "Dadang",
        date_of_birth: "1987-12-04",
        gender: "m"
    },
    <App\Student #00000000151ec6e000000000be123c2> {
        id: 3,
        name: "Yuni",
        date_of_birth: "1992-11-14",
        gender: "f"
    }
]
>>> $siswa->filter(function($siswa){
    return $siswa->gender == 'm';
})
```

```
});  
=> <Illuminate\Database\Eloquent\Collection #00000000151ec7a00000000be123c2> [  
    <App\Student #00000000151ec95000000000be123c2> {  
        id: 1,  
        name: "Joni",  
        date_of_birth: "1990-08-14",  
        gender: "m"  
    },  
    <App\Student #00000000151ec9a000000000be123c2> {  
        id: 2,  
        name: "Dadang",  
        date_of_birth: "1987-12-04",  
        gender: "m"  
    }  
]
```

Terlihat disini, kita menggunakan method `filter` untuk memilih `student` dengan gender `m`.



Ringkasan

Di bab ini kita telah mempelajari penggunaan Collection di Laravel. Menggunakan Collection akan memudahkan kita dalam mengelola koleksi dari banyak object. Pada bab selanjutnya, kita akan membahas View di Laravel. Semangat!

View dari MVC

Dalam mengembangkan aplikasi, sering kali terdapat backend developer dan designer yang memiliki tugas yang berbeda (tentunya, beda cerita kalau Anda seorang Single Fighter Developer :v). Backend developer yang mengerjakan logic aplikasi, sementara designer yang mengerjakan bagaimana informasi tampil dalam aplikasi kita. Terkadang, ketika kita hendak mengerjakan sebuah fitur, seorang designer harus menunggu logic dari Backend Developer hingga selesai.

Disinilah manfaat menggunakan View, kita dapat memisahkan tampilan aplikasi dari logika pengelolaan data. Pada View, kita dapat membuat data *dummy* (palsu). Sehingga, backend developer dan designer dapat bekerja secara paralel.

Penggunaan View

Untuk membuat view, kita dapat membuat file baru di dalam folder `resources/views` dengan extensi `.php`. Misalnya kita buat view untuk menampilkan kalimat motivasi dengan nama `view motivasi.php`:

`resources/views/motivasi.php`

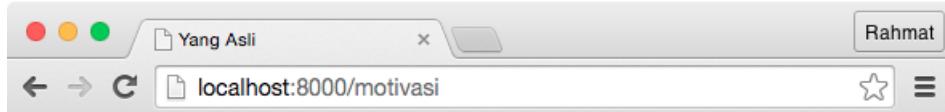
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Yang Asli</title>
</head>
<body>
    <h1>Yang Asli</h1>
    <p>
        Hari ini, kurangilah hal-hal yang palsu dalam kehidupan mu. Agar rezeki \
& cinta yang datang lebih asli. - Mario Teguh
    </p>
</body>
</html>
```

Terlihat disini, kita menggunakan html biasa pada view ini. Untuk menampilkan view ini, kita dapat menggunakan method `return view('nama-view')`. Misalnya, kita buat route `/motivasi` untuk menampilkannya:

app/Http/routes.php

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

Jika kita kunjungi /motivasi maka akan muncul:



Yang Asli

Hari ini, kurangilah hal-hal yang palsu dalam kehidupan mu. Agar rezeki & cinta yang datang lebih asli. - Mario Teguh

View motivasi berhasil dibuat

Konfigurasi

Kita dapat membuat view dalam sub-folder untuk merapihkan strukturnya. Misalnya, kita dapat menyimpan view `motivasi` ke sub-folder `quote`.

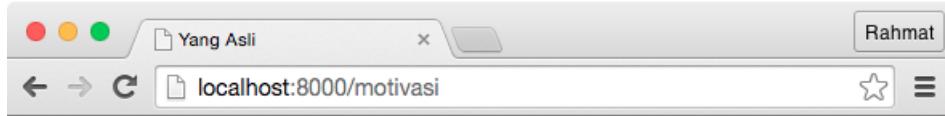
```
$ pwd  
/Users/rahmatawaludin/Code/sample-view/resources/views  
$ mkdir quote  
$ mv motivasi.php quote
```

Untuk memanggil view ini, kita dapat menggunakan *dot notation* untuk mengakses struktur foldernya. Sehingga, pemanggilan view ini akan menjadi:

app/Http/routes.php

```
...  
Route::get('/motivasi', function() {  
    return view('quote.motivasi');  
});
```

Dan view ini akan tetap berjalan:



Yang Asli

Hari ini, kurangilah hal-hal yang palsu dalam kehidupan mu. Agar rezeki & cinta yang datang lebih asli. - Mario Teguh

View motivasi berhasil diakses

Passing Variable

Kita dapat melakukan passing variable ke view dengan menambahkan variable yang hendak kita kirimkan menggunakan array asosiatif sebagai parameter kedua pada saat memanggil view. Misalnya, kita buat nama penulis quote menjadi dynamic:

app/Http/routes.php

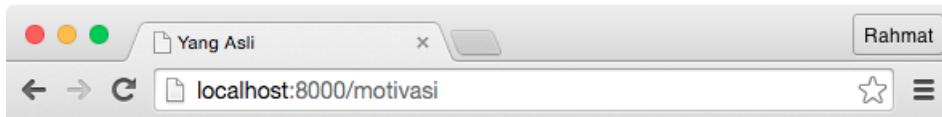
```
...  
Route::get('/motivasi', function() {  
    return view('quote.motivasi', ["penulis" => "Joni"]);  
});
```

Menggunakan syntax ini, kita dapat mengakses variable \$penulis dari view:

resources/views/motivasi.php

```
....  
<p>  
    Hari ini, kurangilah hal-hal yang palsu dalam kehidupan mu. Agar rezeki \& cinta yang datang lebih asli. - <?php echo $penulis; ?>  
</p>  
....
```

Sehingga, kini nama penulis kalimat motivasi akan berubah:



Yang Asli

Hari ini, kurangilah hal-hal yang palsu dalam kehidupan mu. Agar rezeki & cinta yang datang lebih asli. - Joni

Mengirim variable ke view

Tentunya, kita dapat mengirimkan lebih dari satu variable. Misalnya, kita passing juga kalimat motivasinya:

app/Http/routes.php

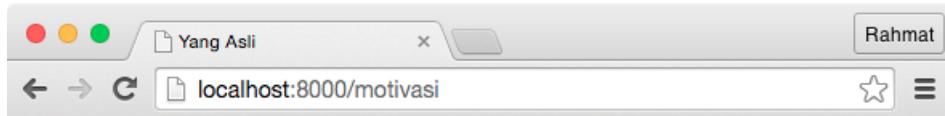
```
....  
Route::get('/motivasi', function() {  
    return view('quote.motivasi', [  
        'penulis' => 'Joni',  
        'kalimat' => 'Jatuh adalah cara maju yang rasanya tidak enak. Bangkit, m\ove on! Jangan menyerah!'  
    ]);  
});
```

Kita ubah juga viewnya:

resources/views/motivasi.php

```
....  
<p>  
    <?php echo $kalimat; ?> - <?php echo $penulis; ?>  
</p>  
....
```

Sehingga kita akan mendapatkan:



Yang Asli

Jatuh adalah cara maju yang rasanya tidak enak. Bangkit, move on! Jangan menyerah! - Joni

Passing lebih dari satu variable

Selain menggunakan array assosiatif, kita juga dapat melakukan passing variable menggunakan method `with()` seperti berikut:

```
return view('quote.motivasi')  
    ->with('penulis', '...')  
    ->with('kalimat', '...');
```

Bisa juga menggunakan magic method seperti berikut: `{lang="php", linenos=off}`
`return view('quote.motivasi') ->withPenulis('...') ->withKalimat('...');`

Passing Variable ke Beberapa View

Selain passing variable dalam sebuah view, kita juga dapat melakukan passing variable ke beberapa view secara sekaligus. Misalnya mari kita passing variable nama aplikasi ke beberapa view. Untuk melakukannya kita perlu menggunakan `View::share()` pada service provider. Kita dapat membuatnya atau menggunakan yang sudah ada. Untuk memudahkan, mari kita gunakan service provider yang sudah ada, misalnya `AppServiceProvider`. Tambahkan baris berikut pada method `boot`:

app/Providers/AppServiceProvider.php

```
<?php  
....  
class AppServiceProvider extends ServiceProvider  
{  
....  
    public function boot()  
    {  
        \View::share([ 'app_name'=> 'Quote App v2.0' ], 'quote.motivasi');  
    }  
....  
}
```

Di baris ini, kita melakukan passing variable \$app_name ke view quote.motivasi. Mari kita panggil variable ini dari view:

resources/views/quote/motivasi.php

```
....  
<p>  
    Made by <?php echo $app_name; ?>  
</p>  
</body>  
</html>
```

Sehingga akan tampil tampilan seperti berikut:



Yang Asli

Jatuh adalah cara maju yang rasanya tidak enak. Bangkit, move on! Jangan menyerah! - Joni

Made by Quote App v2.0

Share variable

Mari kita coba passing variable ini ke view lain, buatlah view quote.inspirasi seperti berikut:

resources/views/quote/inspirasi.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Inspirasi</title>
</head>
<body>
    <h1>Inspirasi</h1>
    <p>
        Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - Bob Sadino
    </p>
    <p>
        Made by <?php echo $app_name; ?>
    </p>
</body>
</html>
```

Kita buat juga route untuk view ini:

app/Http/routes.php

```
Route::get('/inspirasi', function() {
    return view('quote.inspirasi');
});
```

Untuk passing variable \$app_name ke view ini, kita ubah `view::share()` menjadi:

app/Providers/AppServiceProvider.php

```
<?php
...
class AppServiceProvider extends ServiceProvider
{
    ...
    public function boot()
    {
        \View::share(['app_name'=>'Quote App v2.0'], ['quote.motivasi', 'quote.inspirasi']);
    }
    ...
}
```

Mari kita coba akses view ini:



Inspirasi

Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - Bob Sadino

Made by Quote App v2.0

Share variable ke dua view

Sip. Terlihat disini, kita berhasil mengakses variable `$app_name` dari view `quote.inspirasi`.

View Composer

Menggunakan fitur ini, kita dapat menjalankan berbagai logic ketika hendak di render. Misalnya, kita dapat membuat logging, melakukan passing variable untuk beberapa view, dll.

Mari kita coba implementasi sharing variable yang sudah kita buat pada fitur `share()` menggunakan fitur ini. Berbeda dengan fitur `share()` yang telah kita bahas sebelumnya, menggunakan view composer, kita dapat menggunakan class terpisah untuk mengatur logic yang akan kita jalankan sebelum me-render view.

Mari kita buat `AppNameComposer` untuk menyimpan logic nya. Secara default Laravel tidak menyediakan folder untuk menyimpan View Composer, mari kita buat di `app/Http/ViewComposers`:

app/Http/ViewComposers/AppNameComposer.php

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

use Illuminate\Contracts\View\View;

class AppNameComposer {

    /**
     * Bind data to the view.
     *
     * @param  View  $view
     *
```

```
* @return void
*/
public function compose(View $view)
{
    $view->with('app_name', 'Quote App v2.4');
}
}
```

Pada class ini, method `compose()` wajib kita buat. Method ini yang akan dipanggil pada saat Laravel me-*render* view.

Untuk menggunakan view composer, kita harus menggunakan service provider. Mari kita gunakan `AppServiceProvider`, ubah method `boot()` seperti berikut:

app/Providers/AppServiceProvider.php

```
.....
public function boot()
{
    \View::composer(['quote.motivasi', 'quote.inspirasi'], 'App\Http\ViewCompose\
rs\AppNameComposer');
}
.....
```

Method `\View::composer()` ini menerima dua parameter, parameter pertama berisi view dan parameter kedua berisi class dari view composer.

Mari kita coba mengaksesnya:



Inspirasi

Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - Bob Sadino

Made by Quote App v2.4

View composer berhasil

Sip.

Salah satu kelebihan dari view composer (selain struktur folder yang lebih rapi) adalah kita dapat menggunakan wildcard untuk menentukan view. Misalnya, kita dapat menggunakan syntax berikut:

```
\View::composer('quote.*', 'App\Http\ViewComposers\AppNameComposer');
```

Artinya view composer ini akan berlaku untuk semua view di folder quote.

```
\View::composer('*', 'App\Http\ViewComposers\AppNameComposer');
```

Artinya view composer ini akan berlaku untuk semua view.

Untuk fitur lain view composer, silahkan buka [dokumentasi resmi](#)¹³⁴.

Blade, Templating Engine dari Laravel

Menggunakan Blade kita akan lebih mudah dalam mengatur view. Untuk menggunakan Blade, kita cukup mengubah ekstensi file menjadi .blade.php. Misalnya view quote.inspirasi dapat kita ubah namanya menjadi inspirasi.blade.php.

Menampilkan data

Untuk menampilkan data (echo) kita dapat menggunakan syntax {{ }}. Pada contoh view quote.inspirasi, kita dapat merubah syntaxnya menjadi:

resources/views/quote/inspirasi.blade.php

```
....  
<p>  
    Made by {{ $app_name }}  
</p>  
</body>  
</html>
```

Terkadang, kita ingin menampilkan nilai default pada view jika sebuah variable belum diset, untuk itu kita dapat menggunakan syntax berikut:

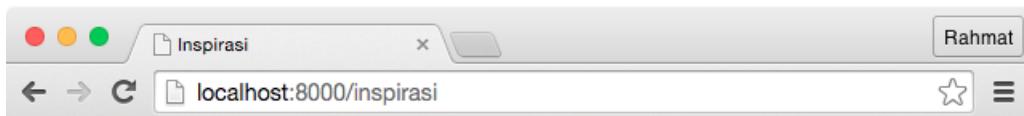
¹³⁴<http://laravel.com/docs/5.0/views#view-composers>

```
{{ $app_name or 'Nama Aplikasi' }}
```

Syntax {{ }} akan meng-escape data markup html, misalnya contoh berikut:

```
....  
<p>  
    Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama\  
tubuh saya kelak. - {{ '<strong>Bob Sadino</strong>' }}  
</p>  
....
```

Maka output akan seperti berikut:



Inspirasi

Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - Bob Sadino

Made by Quote App v2.4

[View escape html](#)

Ini sebenarnya bagus, agar user tidak dapat memasukan script javascript ke text box. Tapi, terkadang kita ingin menghilangkan *behaviour* ini. Untuk itu, kita dapat menggunakan {{! !}}:

```
....  
<p>  
    Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama\  
tubuh saya kelak. - {{! ! '<strong>Bob Sadino</strong>' ! !}}  
</p>  
....
```

Kini, markupnya akan berjalan:



Inspirasi

Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - **Bob Sadino**

Made by Quote App v2.4

unescape variable di view

Lebih lengkapnya, cek [dokumentasi¹³⁵](#).

Komentar

Dalam html kita dapat menggunakan `<!-- --!>` untuk memberikan komentar pada baris. Menggunakan blade, kita dapat menggunakan syntax `{{-- --}}` untuk membuat komentar. Misalnya seperti berikut:

resources/views/master.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    {{-- Title untuk halaman --}}
    <title>@yield('title', 'Quote App')</title>
</head>
<body>
    {{-- Konten untuk halaman --}}
    @yield('body')
</body>
</html>
```

Menggunakan komentar dengan blade sangat disarankan. Berbeda dengan komentar menggunakan `<!-- --!>`, komentar dengan blade tidak akan di cetak di browser.

¹³⁵<http://laravel.com/docs/5.0/templates#other-blade-control-structures>

Blade Layout

Kelebihan utama penggunaan blade adalah kita dapat menggunakan templating untuk merapikan struktur folder aplikasi. Untuk menggunakannya, kita harus membuat file layout yang akan digunakan oleh berbagai view. Misalnya kita buat di resorces/views/master.blade.php:

resorces/views/master.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quote App</title>
</head>
<body>
    @yield('body')
</body>
</html>
```

Pada file ini, kita menggunakan syntax `@yield()` yang menandakan lokasi untuk child view mengisi konten yang diinginkan. View yang kita gunakan sebagai layout biasanya disebut *parent view*. Mari kita gunakan layout ini untuk view quotes.inspirasi :

resources/views/quote/inspirasi.blade.php

```
@extends('master')

@section('body')
<h1>Inspirasi</h1>
<p>
    Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama\'
    tubuh saya kelak. - {!! '<strong>Bob Sadino</strong>' !!}

</p>
<p>
    Made by {{ $app_name }}
</p>
@stop
```

View yang menggunakan layout, kita namakan *child view*. Pada view ini, kita menggunakan syntax `@extends('master')` untuk menandakan bahwa view ini akan menggunakan layout `master.blade.php`.

Selanjutnya, terdapat syntax `@section('body')` dan diakhiri dengan `@stop`. Syntax ini menandakan bahwa child view ini akan mengisi tempat `@yield('body')` pada parent view (`master.blade.php`). Silahkan cek kembali view ini, pastikan bahwa view ini tetap berjalan. Sebagai latihan ubah juga view `quote.motivasi` agar menggunakan blade dan menggunakan layout `master.blade.php`.

Fitur lain dari layouting ini adalah kita dapat menentukan nilai default yang dapat di *override* oleh child view. Misalnya, mari kita buat title dari halaman quote ini menjadi dinamis dengan default berisi tulisan Quote App.

resources/views/master.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>@yield('title', 'Quote App')</title>
</head>
<body>
    @yield('body')
</body>
</html>
```

Kita override pada tampilan halaman motivasi..

resources/views/quote/motivasi.blade.php

```
@extends('master')

@section('title', 'Kalimat Motivasi')

@section('body')
<p>
    <h1>Asli</h1>
    <?php echo $kalimat; ?> - <?php echo $penulis; ?>
</p>
<p>
    Made by <?php echo $app_name; ?>
</p>
@stop
```

Sehingga akan tampil..



Asli

Jatuh adalah cara maju yang rasanya tidak enak. Bangkit, move on! Jangan menyerah! - Joni

Made by Quote App v2.4

Override title pada parent view

Sedangkan pada tampilan halaman /inspirasi akan menggunakan nilai default..



Inspirasi

Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama tubuh saya kelak. - **Bob Sadino**

Made by Quote App v2.4

Menggunakan nilai default dari parent view

Sub View

Syntax @extends() digunakan untuk memberitahu parent view yang akan dia gunakan oleh sebuah view. Namun, adakalanya kita ingin memasukan view lain ke dalam view yang sedang aktif. Contoh penggunaannya untuk form, sidebar, footer, dll. Teknik ini dinamakan "sub view", sementara view yang akan kita masukan ke dalam view aktif biasanya dinamakan "partial view". Untuk menggunakannya kita dapat menggunakan syntax @include().

Umumnya, nama file untuk partial view diawali dengan underscore "_". Misalnya, kita buat footer menjadi partial view dengan file resources/views/partials/_footer.blade.php:

resources/views/partials/_footer.blade.php

```
<p>
    Made by {{ $app_name }}
</p>
```

Kemudian kita ubah view quote.motivasi dan quote.inspirasi:

resources/views/quote/motivasi.blade.php

```
....
@section('body')
<p>
    <h1>Asli</h1>
    <?php echo $kalimat; ?> - <?php echo $penulis; ?>
</p>
@include('partials._footer')
@stop
```

resources/views/quote/inspirasi.blade.php

```
....
@section('body')
<h1>Inspirasi</h1>
<p>
    Saya tidak mau pengalaman dan pengetahuan yang saya miliki, terkubur bersama\
    tubuh saya kelak. - {!! '<strong>Bob Sadino</strong>' !!}
</p>
@include('partials._footer')
@stop
```

Silahkan cek kembali kedua view tersebut dan pastikan footer pada kedua view masih berjalan.

Jika dinginkan, kita juga dapat passing variable ke subview dengan menambah array assosiatif sebagai parameter kedua. Misalnya seperti berikut:

```
@include('partials._footer', ['user'=>Auth::user()])
```

Pada syntax diatas, variable \$user dapat diakses oleh view partials._footer berisi nilai dari Auth::user().

Blade Control Structure

Menggunakan blade, kita dapat menyederhanakan beberapa syntax untuk logic.

If

Untuk syntax `if`, kita dapat menulisnya seperti berikut:

```
@if (...)  
@elseif (...)  
  
@else  
  
@endif
```

Unless

Unless merupakan kebalikan dari if. Contoh syntaxnya seperti berikut:

```
@unless ($user->count() > 0)  
    Tidak ada user  
@endunless
```

Pada syntax diatas, teks "Tidak ada user" akan selalu ditampilkan **kecuali** nilai dari `$user->count()` lebih dari 0.

For

Syntax `for` dapat ditulis seperti berikut:

```
@for (...)  
    // jalankan logic disini  
@endfor
```

Foreach

Syntax `foreach` dapat ditulis seperti berikut:

```
@foreach (...)
    // jalankan logic disini
@endforeach
```

Forelse

Syntax `forelse` merupakan gabungan dari `if` dan `foreach`.

```
@forelse($books as $books)
<li>{{ $book->title }}</li>
@empty
    <p>Tidak ada buku.</p>
@endforelse
```

Pada syntax diatas, jika variable `$books` memiliki konten, maka kita akan mendapatkan daftar judul buku. Sedangkan jika tidak, kita akan mendapat teks "Tidak ada buku."

While

Syntax `while` dapat ditulis seperti berikut:

```
@while (...)
    // jalankan logic disini
@endwhile
```

Asset

Dalam mengelola view, pastinya kita akan sering membutuhkan file css/js. Secara default, Laravel menganggap kita akan menyimpan semua file asset tersebut pada folder `public`. Jika kita menyimpan file css pada `public/css/style.css`, maka pada view, kita dapat menggenerate path ke file ini dengan syntax:

```
<link rel="stylesheet" href="{{ asset('css/style.css') }}">
```

Dengan menggunakan method `asset()`, URL untuk file ini akan selalu mengarah ke root dari URL website kita. Teknik yang sama juga kita gunakan untuk file JS, image, dll.

Elixir

Untuk memahami penggunaan Elixir, sebaiknya Anda mempelajari tentang *frontend tools* bernama [Gulp¹³⁶](#). Dengan Gulp kita dapat menjalankan berbagai *frontend tools* dengan mudah. Umumnya digunakan sebagai *build tools*, misalnya untuk compile file SASS/LESS, minify CSS/JS, Cache busting, dll. Pembahasan gulp dan berbagai frontend tools akan menghabiskan satu buku, oleh karena itu, pada bagian ini kita akan lebih fokus ke penggunaan Elixir.

Konfigurasi

Untuk menggunakan Elixir, ada beberapa tools yang harus diinstall:

1. NodeJS. Ikuti panduan install di [https://nodejs.org¹³⁷](https://nodejs.org). Setelah berhasil install, pastikan dapat menjalankan:

```
$ node -v  
v0.12.7
```

2. Install Gulp secara global, ikuti [panduan ini¹³⁸](#). Singkatnya, jalankan perintah:

```
$ npm install --global gulp
```

Untuk pengguna Mac, pastikan menambah `sudo` sebelum menjalankan perintah diatas. Setelah berhasil, pastikan bisa menjalankan:

```
$ gulp -v  
[20:59:17] CLI version 3.8.11
```

3. Pada instalasi Laravel terbaru akan terdapat file `package.json` yang berisi dependensi *frontend package* yang mesti diinstall. Untuk menginstallnya, jalankan perintah `npm install`. Jika menggunakan windows, tambahkan opsi `--no-bin-links`. Tunggu hingga proses download selesai.
4. Ubah konfigurasi pada file `gulpfile.js`. Secara default, isinya akan seperti berikut:

¹³⁶<http://gulpjs.com>

¹³⁷<https://nodejs.org>

¹³⁸<https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md>

gulpfile.js

```
elixir(function(mix) {  
    mix.sass('app.scss');  
});
```

Untuk sekarang, mari kita gunakan konfigurasi ini.

Penggunaan

Untuk menggunakan elixir, kita dapat menjalankan perintah `gulp` dari terminal:

```
$ gulp  
[08:18:08] Using gulpfile ~/Code/sample-view/gulpfile.js  
[08:18:08] Starting 'default'...  
[08:18:08] Starting 'sass'...  
[08:18:08] Running Sass: resources/assets/sass/app.scss  
[08:18:09] Finished 'default' after 642 ms  
[08:18:09] gulp-notify: [Laravel Elixir] Sass Compiled!  
[08:18:09] Finished 'sass' after 775 ms
```

Dengan konfigurasi default, Elixir akan meng-compile file `resources/assets/sass/app.scss` ke folder `public/css/app.css`. File yang akan digenerate, hasilnya akan seperti berikut:

public/css/app.css

```
/*# sourceMappingURL=app.css.map */
```

Ini terjadi karena file `resources/assets/sass/app.scss` masih kosong. Secara default, Elixir telah memasukan `bootstrap`¹³⁹ ke dalam `*-nya`. Untuk mengaktifkannya, ubahlah isian pada file `app.scss` menjadi:

resources/assets/sass/app.scss

```
@import "node_modules/bootstrap-sass/assets/stylesheets/bootstrap";
```

Jalankan kembali perintah `gulp`, maka isian `app.css` akan berisi css untuk bootstrap:

¹³⁹<http://getbootstrap.com>

public/css/app.css

```
/*
 * Bootstrap v3.3.5 (http://getbootstrap.com)
 * Copyright 2011-2015 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 */
/*! normalize.css v3.0.3 | MIT License | github.com/necolas/normalize.css */
html {
    font-family: sans-serif;
    -ms-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%; }
```

Mode Production

Kita juga dapat menggunakan perintah `gulp --production` untuk mengkompress dan minify file asset:

```
$ gulp --production
[08:25:06] Using gulpfile ~/Code/sample-view/gulpfile.js
[08:25:06] Starting 'default'...
[08:25:06] Starting 'sass'...
[08:25:06] Running Sass: resources/assets/sass/app.scss
[08:25:07] Finished 'default' after 536 ms
[08:25:08] gulp-notify: [Laravel Elixir] Sass Compiled!
[08:25:08] Finished 'sass' after 2.02 s
```

Dengan perintah ini, file app.css yang dihasilkan akan menjadi:

public/css/app.css

```
/*
 * Bootstrap v3.3.5 (http://getbootstrap.com)
 * Copyright 2011-2015 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 *//*! normalize.css v3.0.3 | MIT License | github.com/necolas/normalize.css */\h\
t\ml{font-family:sans-serif;-ms-text-size-adjust:100%;-webkit-text-size-adjust:10\%}\b\ody{margin:0}\a\rticle,.....
```

Menggunakan opsi `--production` sangat disarankan ketika hendak mengirim source code ke server.

Mode Watch

Menjalankan perintah `gulp` setiap kali kita melakukan perubahan file asset cukup merepotkan. Menggunakan opsi `watch` akan membuat `gulp` terus berjalan di-*background* dan secara otomatis menjalankan `gulp` setiap kali ada perubahan file. Mari kita coba:

```
$ gulp watch
[20:30:25] Using gulpfile ~/Code/sample-view/gulpfile.js
[20:30:25] Starting 'watch'...
[20:30:25] Starting 'sass'...
[20:30:25] Running Sass: resources/assets/sass/app.scss
[20:30:25] Finished 'watch' after 541 ms
[20:30:28] gulp-notify: [Laravel Elixir] Sass Compiled!
[20:30:28] Finished 'sass' after 3.58 s
[20:30:28] Starting 'watch-assets'...
[20:30:28] Finished 'watch-assets' after 24 ms
```

Mari kita tambahkan syntax scss berikut pada file `app.scss`:

resources/assets/sass/app.scss

```
@import "node_modules/bootstrap-sass/assets/stylesheets/bootstrap";
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Ketika kita menyimpan file ini, maka pada terminal akan terlihat output seperti berikut:

```
[20:31:53] Starting 'sass'...
[20:31:53] Running Sass: resources/assets/sass/app.scss
[20:31:55] gulp-notify: [Laravel Elixir] Sass Compiled!
[20:31:55] Finished 'sass' after 2.51 s
```

Output ini menandakan bahwa file scss telah di-*compile* ulang.

Menggabungkan Javascript

Kita dapat menggabungkan file javascript dengan menambahkan syntax berikut pada file `gulpfile.js`:

gulpfile.js

```
elixir(function(mix) {
  mix.sass('app.scss');
  mix.scripts([
    'plugin.js',
    'app.js'
  ]);
});
```

Dengan syntax ini, Elixir akan menggabungkan isi dari file `resources/assets/js/plugin.js` dan `resources/assets/js/app.js`, kemudian menyimpan outputnya pada file `public/js/all.js`.

Cache Busting

Teknik ini disebut juga *versioning*. Jika sering bekerja dengan CSS, pastinya pernah mengalami kondisi dimana file css telah kita ubah namun di sisi client belum ada perubahan karena *cache* nya masih tersimpan. Untuk mengatasi hal ini, kita dapat meminta client melakukan *hard refresh* (menekan Shift + tombol refresh) untuk mere-load paksa semua file asset atau menggunakan teknik cache busting.

Dengan teknik ini, kita memberikan nama file baru untuk setiap kali terdapat perubahan pada file css. Menggunakan cara manual, penggunaan teknik ini cukup merepotkan, karena kita harus merubah cara memanggil file css setiap kali melakukan perubahan. Dengan menggunakan Elixir, teknik ini sangat mudah untuk digunakan.

Caranya, tambahkan baris berikut pada file `gulpfile.js`:

gulpfile.js

```
elixir(function(mix) {
  mix.sass('app.scss');
  mix.version('css/app.css');
  mix.scripts([
    'plugin.js',
    'app.js'
  ]);
});
```

Syntax ini, menandakan bahwa kita akan memberikan *cache busting* untuk file `public/css/app.css`. Hasil dari perintah ini akan dibuat folder `public/build` yang berisi:

- *rev-manifest.json* : berisi catatan nama file yang telah dirubah
- *css* : berisi css yang telah berubah namanya

Untuk memanggil file css yang telah di generate ini, kita cukup merubah `asset()` menjadi `elixir()` pada view.

```
<link rel="stylesheet" href="{{ elixir('css/app.css') }}">
```

Secara otomatis URL nya akan berubah menjadi `/build/css/app-nomoraneh.css`. Tentunya, teknik *cache busting* ini dapat pula kita gunakan untuk file js.

Masih banyak opsi lain dari Elixir ini. Untuk lebih lengkapnya, silahkan cek [dokumentasi resmi](#)¹⁴⁰.

Form

Semenjak Laravel versi 5, helper untuk form dari Laravel telah dihilangkan. Tentunya, kita bisa menggunakan cara manual dengan menulis setiap element html dari sebuah form. Tapi, saya sendiri lebih suka menggunakan cara otomatis dengan menambah package [laravelcollective/html](#)¹⁴¹ dengan menambah baris berikut pada file `composer.json`:

`composer.json`

```
...
"require": {
    ...
    "laravelcollective/html": "~5.0"
}
```

Jalankan `composer update` untuk menginstallnya. Tunggu hingga selesai.

Setelah selesai, masukan isian berikut pada bagian `providers` di `config/app.php`:

¹⁴⁰ <http://laravel.com/docs/5.1/elixir>

¹⁴¹ <http://laravelcollective.com/docs/5.0/html>

config/app.php

```
'providers' => [
    // ...
    Collective\Html\HtmlServiceProvider::class,
    // ...
],
```

Dan isian berikut pada isian `aliases`:

config/app.php

```
'aliases' => [
    // ...
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
    // ...
],
```

Membuat Form

Untuk membuat form, kita dapat menggunakan syntax `Form::open()` dan diakhiri dengan `Form::close()`. Method `open()` menerima berbagai parameter berupa array opsi form yang akan kita buat.

Mari kita buat form untuk menambah quote. Untuk memudahkan, kita akan menggunakan Bootstrap yang sudah bawaan dari Elixir, pertama pastikan isian `resources/assets/sass/app.scss` seperti berikut:

resources/assets/sass/app.scss

```
@import "node_modules/bootstrap-sass/assets/stylesheets/bootstrap";
```

Jalankan `gulp` untuk menggenerate css ke folder `public/css/app.css`. Ubah pula isian `master.blade.php` menjadi:

resources/views/master.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    {{-- Title untuk halaman --}}
    <title>@yield('title', 'Quote App')</title>
    <link rel="stylesheet" href="{{ asset('css/app.css') }}">
</head>
<body>
    {{-- Konten untuk halaman --}}
    <div class="container">
        <div class="row">
            @yield('body')
        </div>
    </div>
</body>
</html>
```

Syntax `asset('css/app.css')` menandakan kita akan menggenerate url ke file `css/app.css` di folder public.

Untuk form pembuatan quote, buat view seperti berikut di `/resources/views/new-quote.blade.php`:

resources/views/new-quote.blade.php

```
@extends('master')

@section('body')
{!! Form::open(['url'=>'/quote']) !!}
<legend>New Quote</legend>

<div class="form-group">
    {!! Form::label('author', 'Author') !!}
    {!! Form::text('author', null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
    {!! Form::label('type', 'Type') !!}
    {!! Form::select('type', ['inspiration' => 'Inspirasi', 'motivation' => 'Motivasi'],
```

```
    null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
  {!! Form::label('quote', 'Quote') !!}
  {!! Form::textarea('quote', null, ['class'=>'form-control']) !!}
</div>

{!! Form::submit('Save', ['class'=>'btn btn-primary']) !!}

{!! Form::close() !!}
@stop
```

Untuk melihat hasil akhir dari form ini, buatlah route berikut:

app/Http/routes.php

```
Route::get('/new-quote', function() {
    return view('new-quote');
});
```

Akses di /new-qoute:

New Quote

Author

Tipe

Inspirasi

Quote

Save

Form new quote

Form ini akan memiliki tiga field: author, tipe dan quote.

Terlihat disini, untuk membuat form kita menggunakan syntax `{!! Form::open(['url'=>'/quote']) !!}` yang diakhiri dengan `{!! Form::close() !!}`. Pada syntax ini kita memberitahu Laravel bahwa kita akan mengirim form ke url /new-quote dengan method post.

Cara lain untuk membuat form dapat kita lihat di [dokumentasi](#)¹⁴², diantaranya:

- Menggunakan method selain post (get, put atau delete).

```
{!! Form::open(array('url' => '/quote', 'method' => 'get')) !!}
```

- Menggunakan named route. Pada contoh ini, misalnya kita telah membuat route dengan nama `quote.store`.

```
{!! Form::open(array('route' => 'quote.store')) !!}
```

- Mengizinkan file upload.

¹⁴²<http://laravelcollective.com/docs/5.1/html#opening-a-form>

```
{!! Form::open(array('url' => '/quote', 'files' => true)) !!}
```

- Menambah attribut lain ke form, misalnya class. Format nya adalah 'attribute' => 'value'. Pada contoh ini, kita menambah class `form-inline` dan attribute `novalidate` untuk mematikan fitur validasi form dari HTML5.

```
{!! Form::open(array('url' => '/quote', 'class' => 'form-inline', 'novalidate') !!}
```

Sip, lebih lengkapnya cek dokumentasi ya. Selanjutnya kita menambah field untuk form. Banyak sekali field yang didukung oleh package ini, berikut beberapa jenis field yang kita gunakan pada form ini.

- Label. Kita gunakan untuk menamai setiap field pada form.

```
{!! Form::label('quote', 'Quote') !!}
```

Dari syntax diatas akan didapatkan:

```
<label for="quote">Quote</label>
```

- Text field. Kita gunakan pada field `author`.

```
{!! Form::text('author', null, ['class'=>'form-control']) !!}
```

Pada field ini, parameter pertama adalah nama field, kedua adalah isian field, dan ketiga adalah opsi field. Dari syntax diatas akan digenerate:

```
<input class="form-control" name="author" type="text" id="author">
```

- Dropdown List. Kita gunakan pada field `tipe`.

```
{!! Form::select('type', ['inspiration' => 'Inspirasi', 'motivation' => 'Motivasi'], null, ['class'=>'form-control']) !!}
```

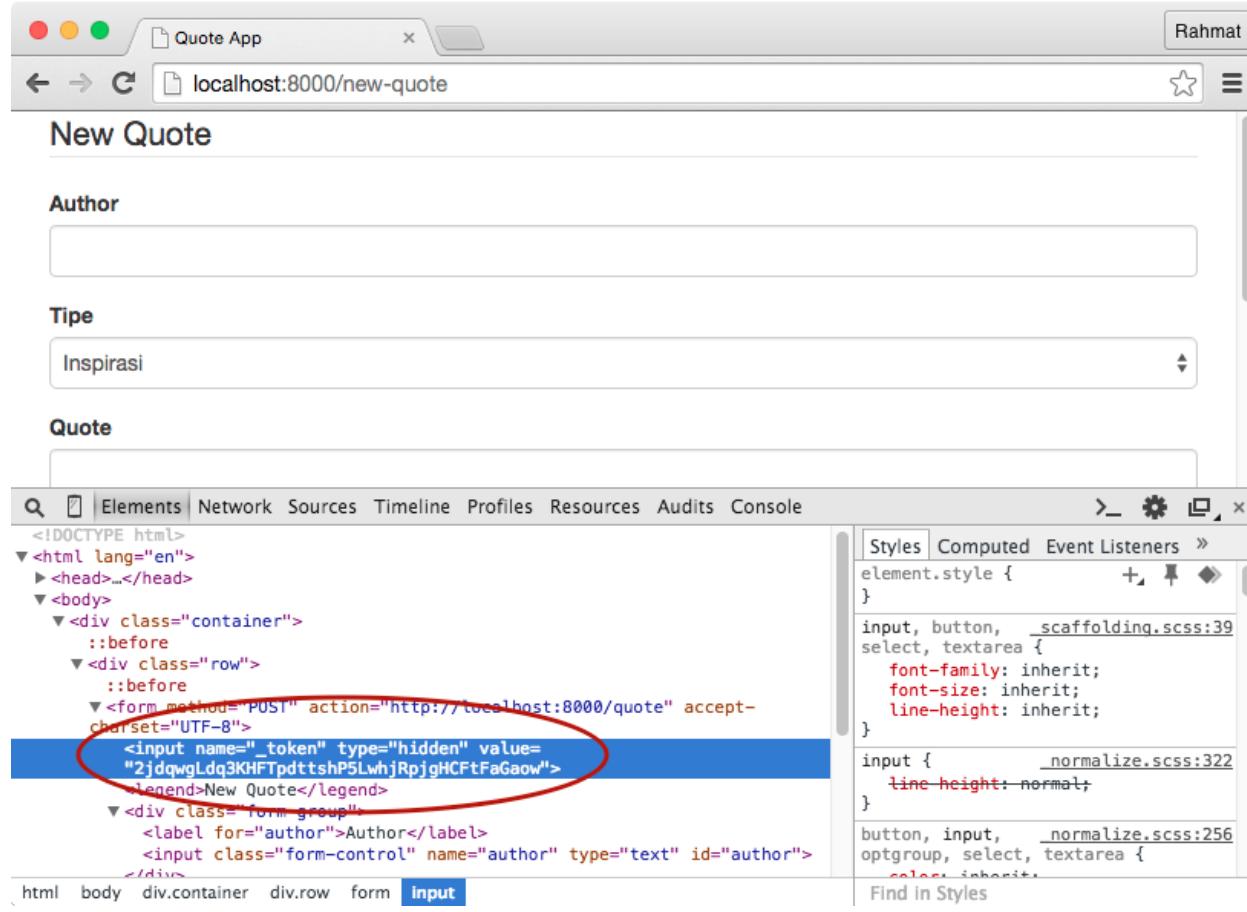
Pada field ini, parameter pertama adalah nama field, kedua adalah array asosiatif berisi element dropdown, ketiga adalah nilai dari field dan terakhir opsi dari field ini. Sehingga akan digenerate:

```
<select class="form-control" id="type" name="type">
    <option value="inspiration">Inspirasi</option>
    <option value="motivation">Motivasi</option>
</select>
```

- Textarea. Kita gunakan untuk membuat field quote. Syntaxnya sama dengan tipe text, jadi tidak akan saya jelaskan kembali.

Masih banyak jenis field lain yang didukung package ini, silahkan cek [dokumentasi](#)¹⁴³. Sebagai catatan, dengan menggunakan `Form::open()`, secara default field `_token` akan digenerate. Field ini akan berisi token `csrf` yang akan digunakan oleh Laravel untuk mengamankan form dari serangan CSRF.

CSRF (Cross Site Request Forgery) secara sederhana adalah teknik yang digunakan hacker untuk menjalankan *action* pada suatu website sebagai user yang sedang login tanpa sepengetahuan user yang bersangkutan.



The screenshot shows a browser window titled "Quote App" with the URL "localhost:8000/new-quote". The page displays a "New Quote" form with fields for "Author" (empty), "Type" (set to "Inspirasi"), and "Quote" (empty). Below the form, the browser's developer tools are open, specifically the Elements and Styles tabs. The Elements tab shows the HTML structure of the page, including a highlighted `<input name="_token" type="hidden" value="2jdqwgLdq3KHFTpdttshP5LwhjRpjgHCFtFaGaow">` field. A red oval highlights this field. The Styles tab shows the CSS rules applied to this element, which includes inheritance from "normalize.scss" and "scaffolding.scss".

Generate token

¹⁴³ <http://laravelcollective.com/docs/5.1/html>

Untuk mengecek apakah form ini berhasil di submit, mari kita buat route POST dengan url /quote dengan isi:

app/Http/routes.php

```
Route::post('/quote', function() {
    dd(Input::all());
});
```

Syntax `dd(Input::all());` akan menampilkan semua inputan yang telah kita isi. Silahkan coba isi form, kemudian submit. Hasilnya lebih kurang akan seperti berikut:

```
array:4 [▼
  "_token" => "LNGmtrPgdnRjFfXaeOdgrHuNSWoWTLSiFBdeVr"
  "author" => "Tung Desem Waringin"
  "type" => "motivation"
  "quote" => "Tidak ada kata GAGAL, yang ada hanya SUKSES atau BELAJAR."
]
```

Menampilkan semua input



Untuk mempelajari teknik menangani input di Laravel, cek Bab 11.

Form Model Binding

Fitur ini merupakan salah satu fitur menarik dari penggunaan package HTML. Bayangkan kita sedang meng-edit sebuah quote, tentunya isian setiap field nya harus diisi dengan data yang telah tersimpan di database. Menggunakan cara manual, kita harus mengisi setiap field tersebut. Misalnya:

```
{!! Form::text('author', $quote->author, ['class'=>'form-control']) !!}
```

Tidak masalah jika fieldnya sedikit. Tapi, jika field nya banyak, tentu hal ini akan sangat merepotkan. Menggunakan *form model binding*, syntax untuk tiap field akan sama seperti saat kita membuat form baru.

```
{!! Form::text('author', null, ['class'=>'form-control']) !!}
```

Tapi telah terisi dengan data yang telah tersimpan di database.

Selain manfaat diatas, penggunaan *form model binding* juga memungkinkan kita untuk menggunakan form yang sama untuk membuat maupun meng-edit sebuah resource.

Mari kita praktikan dengan membuat form untuk mengedit quote. Sebelum mempraktekannya, silahkan buat dulu model App\Quote dengan isian kira-kira seperti berikut:

id	author	type	quote
1	Mario Teguh	inspiration	Jadilah pria baik-baik.
2	Andrie Wongso	inspiration	1000 orang tidak percaya keamampuan kita, tidak masalah.
3	Bob Sadino	inspiration	Bergayalah sesuai isi dompetmu.

Sebagaimana dijelaskan pada bagian sebelumnya, dengan menggunakan form model binding kita dapat menggunakan form yang sama untuk membuat maupun meng-edit resource. Untuk itu, mari kita pisahkan field dari form new-qoute ke file _quote-form.blade.php:

resources/views/_quote-form.blade.php

```
<div class="form-group">
    {!! Form::label('author', 'Author') !!}
    {!! Form::text('author', null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
    {!! Form::label('type', 'Tipe') !!}
    {!! Form::select('type', ['inspiration' => 'Inspirasi', 'motivation' => 'Motivasi'],
        null, ['class'=>'form-control']) !!}

```

```
</div>

<div class="form-group">
    {!! Form::label('quote', 'Quote') !!}
    {!! Form::textarea('quote', null, ['class'=>'form-control']) !!}
</div>
```

Dan isian new-quote.blade.php menjadi:

resources/views/new-quote.blade.php

```
@extends('master')

@section('body')
{!! Form::open(['url'=>'/quote']) !!}
<legend>New Quote</legend>
@include('_quote-form')
{!! Form::submit('Save', ['class'=>'btn btn-primary']) !!}
{!! Form::close() !!}
@stop
```

Untuk mengedit quote, kita akan menggunakan url /edit-quote/{id}. Dimana {id} merupakan id dari quote yang hendak kita edit. Mari kita buat route nya:

app/Http/routes.php

```
Route::get('/edit-quote/{id}', function($id) {
    $quote = App\Quote::findOrFail($id);
    return view('edit-quote', compact('quote'));
});
```

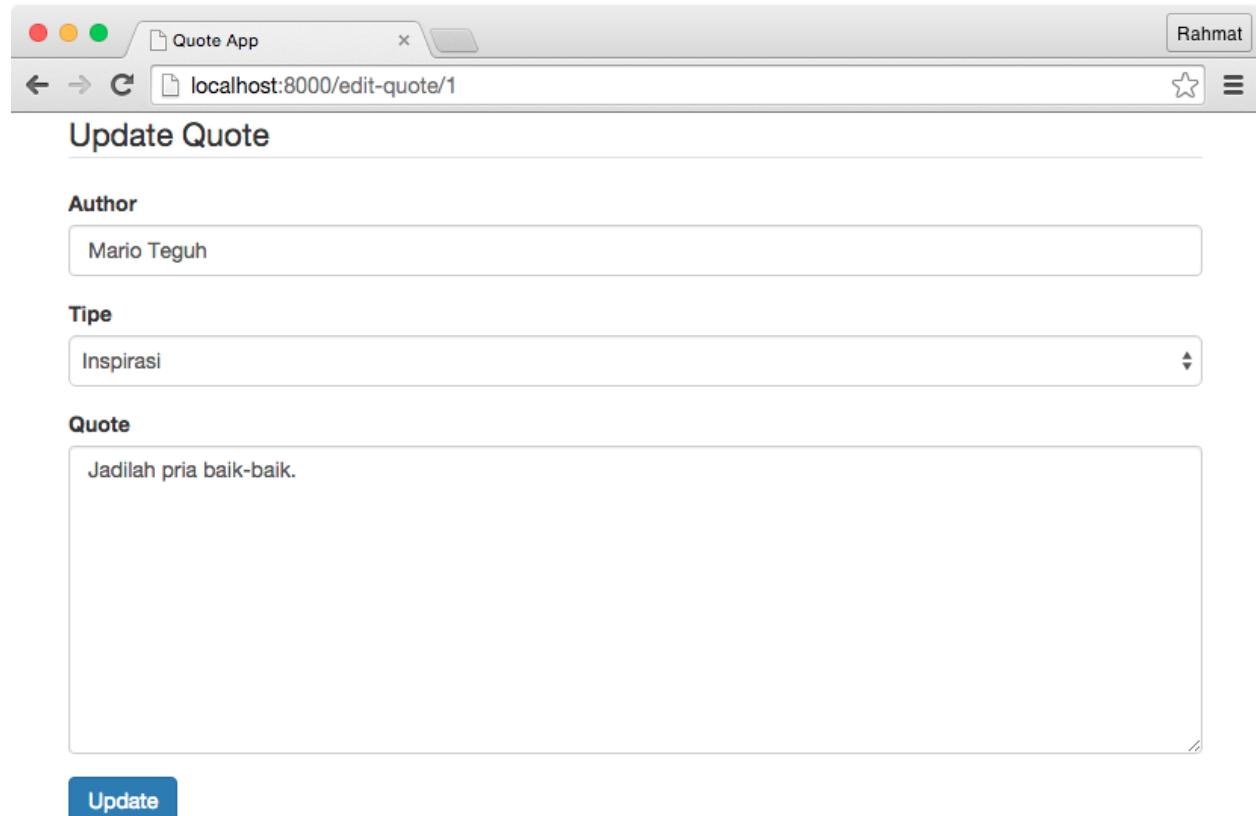
Pada route ini, kita menggunakan method `findOrFail()` pada model `App\Quote` untuk mencari quote dengan id yang kita kirim. Kemudian menampilkan view `edit-quote.blade.php` sambil mengirim variable `$quote`. Isian dari `edit-quote.blade.php` akan seperti berikut:

resources/views/edit-quote.blade.php

```
@extends('master')

@section('body')
{!! Form::model($quote, ['url'=>'/quote/' . $quote->id, 'method'=>'put']) !!}
<legend>Update Quote</legend>
@include('_quote-form')
{!! Form::submit('Update', ['class'=>'btn btn-primary']) !!}
{!! Form::close() !!}
@stop
```

Terlihat disini, kita menggunakan `Form::model()` dengan parameter variable `$quote` yang telah kita kirim. Jika kita mengakses url `/edit-quote/{id}` dengan id yang benar, maka form akan otomatis terisi dengan data yang tersimpan di database:



Berhasil melakukan form binding



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁴⁴



Ringkasan

Di bab ini kita telah mempelajari penggunaan View di Laravel. Menggunakan fitur-fitur Blade di Laravel akan memudahkan kita dalam membuat tampilan UI yang kompleks. Pada bab selanjutnya kita akan menghubungkan komponen model dan view dengan controller. Semangat!

¹⁴⁴<https://bitbucket.org/rahmatawaludin/sample-view/commits>

Controller dari MVC

Dalam mengembangkan aplikasi dengan Design Pattern MVC, controller berperan untuk mengarahkan setiap request ke aplikasi. Controller umum digunakan untuk menjawab request yang dikirim ke route, yang selanjutnya akan diarahkan ke model yang diperlukan.

Dengan pemahaman ini, sebaiknya sebuah controller memang memiliki sedikit kode. Sementara model memiliki banyak kode, karena di model-lah *business logic* dari aplikasi akan berjalan. Teknik sedikit code di controller dan banyak kode di model ini biasa dikenal dengan istilah *thick model, thin controller*. Dan ini merupakan *best practices* yang bisa kita jadikan pegangan selama koding.

Penggunaan Controller

Pada bab-bab sebelumnya, kita selalu menyimpan logic untuk menjawab *request* di file routes. Untuk aplikasi yang sederhana, ini tidak masalah. Tapi, ketika aplikasi sudah besar, cara ini sangat tidak efektif. Disinilah controller akan membantu. Menggunakan controller, kita cukup mengarahkan request pada sebuah route ke method pada controller tertentu.

Untuk membuat controller kosong dengan nama `HomeController`, kita dapat menggunakan perintah:

```
$ php artisan make:controller HomeController --plain  
Controller created successfully.
```

Opsi `--plain` menandakan kita hendak membuat controller tanpa method default.

Update Laravel 5.2

Pada Laravel 5.2, opsi `--plain` sudah ditiadakan. Secara default semua controller yang dibuat tanpa method default. Untuk menambahkan method default, kita harus menambahkan opsi `--resource`.

Perintah diatas akan menghasilkan file `app/Http/Controllers/HomeController.php` dengan isi:

app/Http/Controllers/HomeController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class HomeController extends Controller
{
    //
}
```

Untuk menggunakan controller ini, kita harus menghubungkannya dengan file route. Misalnya, kita memiliki url /about yang akan dijawab dengan method about pada HomeController, maka ini yang perlu kita tulis di file routes:

app/Http/routes.php

```
Route::get('/about', 'HomeController@about');
```

Kita juga dapat memberikan nama pada route ini, misalnya kita beri nama home.about. Maka syntaxnya kita ubah menjadi:

app/Http/routes.php

```
Route::get('/about', ['as' => 'home.about', 'uses' => 'HomeController@about']);
```

Silahkan cek apakah route ini dengan perintah `php artisan route:list`.

Tentunya, kita harus membuat method about tersebut. Mari kita buat agar method ini menampilkan view `about.blade.php`:

app/Http/Controllers/HomeController.php

```
....  
class HomeController extends Controller  
{  
    public function about()  
    {  
        return view('about');  
    }  
}
```

Kita buat pula view about :

resources/views/about.blade.php

```
<h1>About</h1>  
<p>Wiihh.. mau kenalan nih?</p>
```

Kini, jika kita mengakses /about:



About

Wiihh.. mau kenalan nih?

route about berhasil dibuat

Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁴⁵

¹⁴⁵<https://bitbucket.org/rahmatawaludin/sample-controllers/commits/ae323a20f093d1ad33fb5d0ccad26364aa6b7ba7>

Konfigurasi

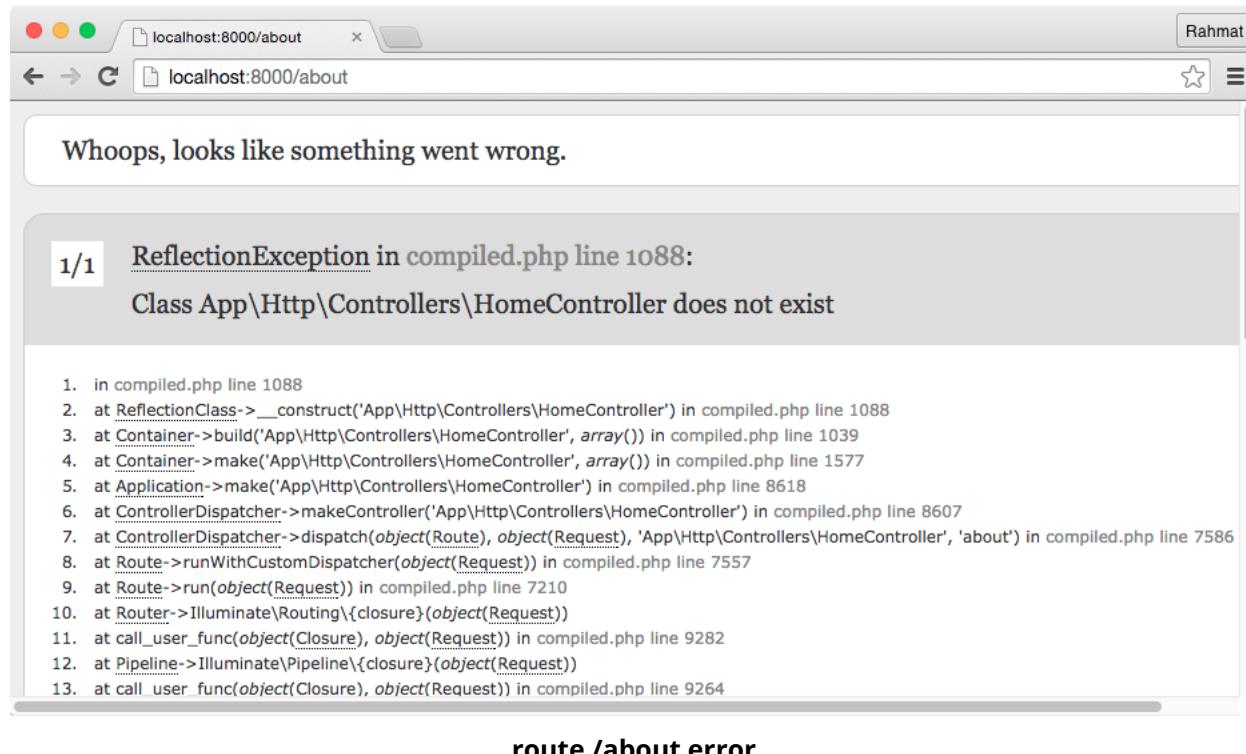
Secara default, Laravel akan menganggap kita menyimpan controller pada folder `app\Http\Controller`. Itulah sebabnya kita tidak perlu menulis namespace lengkap dari controller yang kita buat pada file routes.

Tentunya untuk aplikasi yang besar adakalanya kita harus membuat struktur folder custom untuk controller yang kita buat. Sebagai contoh, mari kita simpan `HomeController` pada folder `app\Http\Api` (buatlah folder `Api` secara manual). Dengan perubahan lokasi file ini, kita perlu merubah isian namespace pada `HomeController` menjadi:

`app\Http\Api\HomeController.php`

```
<?php  
  
namespace App\Http\Api;  
...
```

Karena lokasi `HomeController` di luar folder controller, jika kita mencoba route `/about` akan muncul error:



Untuk mengatasinya, kita perlu menulis nama class berikut namespace lengkapnya di file routes:

app/Http/routes.php

```
Route::get('/about', '\App\Http\Api\HomeController@about');
```

Kini, route /about akan berhasil dijalankan:



route /about dengan lokasi controller custom



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁴⁶

Implicit Controller

Jika malas membuat route untuk setiap method yang ada pada controller yang telah dibuat, kita dapat menggunakan fitur Implicit Controller agar setiap method berubah menjadi route.

Menggunakan fitur ini, kita dapat menggunakan method `Route::controller()`. Mari kita praktekan, silahkan buat `BlogController`:

```
$ art make:controller BlogController
Controller created successfully.
```

Kemudian kita tambahkan pada file routes:

¹⁴⁶ <https://bitbucket.org/rahmatawalidin/sample-controllers/commits/f88fa20cd94c7f72e4ad38f1c68e5a290deecd48>

app/Http/routes.php

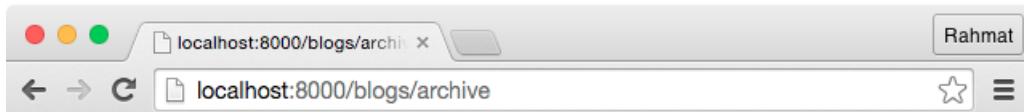
```
Route::controller('blogs', 'BlogController');
```

Misalnya, kita ingin membuat route GET dengan url /blogs/archive maka ini method yang harus kita buat:

app/Http/Controllers/BlogController.php

```
....  
class BlogController extends Controller  
{  
    public function getArchive()  
    {  
        return '<h1>Halaman Archive</h1>';  
    }  
}
```

Kita coba akses:



Halaman Archive

Membuat route /blogs/archive dengan Implicit controller

Gitu. Jadi, nama methodnya harus diawali dengan jenis request yang kita inginkan. Bagian selanjutnya akan jadi URL. Kalau kita ingin membuat route POST dengan url /blogs/update, maka ini method yang harus kita buat:

app/Http/Controllers/BlogController.php

```
....  
class BlogController extends Controller  
{  
    public function postUpdate()  
    {  
        return 'Menyimpan article..';  
    }  
}
```

Jika dinginkan, kita juga dapat memberikan nama untuk tiap route diatas seperti berikut:

app/Http/routes.php

```
Route::controller('blogs', 'BlogController', [  
    'getArchive' => 'blog.archive',  
    'postUpdate' => 'blog.update'  
]);
```

Disini kita memberikan nama (named route) `blog.archive` dan `blog.update` untuk kedua method yang telah kita buat.

Saya sendiri jarang menggunakan implicit controller. Salah satu alasannya, menurut saya menggunakan fitur ini, cukup sulit untuk mengetahui route yang tersedia di aplikasi. Tapi, tentunya tiap orang punya pertimbangan yang berbeda. So, sah-sah aja kalau Anda mau memakainya. Sip.



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁴⁷

RESTful Resource Controller

Fitur Resource Controller akan sangat bermanfaat dalam mempercepat proses pembuatan CRUD aplikasi. Dengan menggunakan teknik ini, Laraveen akan membuatkan method dan url untuk setiap proses CRUD yang akan kita lakukan. Untuk membuatnya, kita dapat menjalankan perintah `php artisan make:controller` tanpa opsi `--plain`.

¹⁴⁷ <https://bitbucket.org/rahmatawaludin/sample-controllers/commits/597925216a58cdba73966ac12188be4d2edba39e>

Untuk mendemokan fitur ini, mari kita gabungkan yang sudah kita pelajari dari bab sebelumnya dengan membuat proses CRUD untuk order pada sebuah website Martabak. Mari kita beri nama Domino Martabak! :D

Oke, pertama buatlah model `Order` berikut seeder untuk model tersebut hingga hasil akhirnya akan terdapat table `orders` dengan isi sebagai berikut:

id	customer	tipe	jumlah	alamat
1	Doni	coklat-kacang	3	Jl. Garuda 20 New York
2	Rini	keju	2	Jl. Juara 12 New York

Saya yakin Anda sudah paham bagaimana mencapai hasil diatas. Kalau belum paham, buka kembali bab-bab sebelumnya.

Selanjutnya, kita akan buat RESTful controller untuk model ini. Caranya, jalankan perintah berikut:

```
$ php artisan make:controller OrdersController
Controller created successfully.
```

Perintah ini akan menghasilkan file `app/Http/Controllers/OrdersController.php` dengan isi:

app/Http/Controllers/OrdersController.php

```
<?php
...
class OrdersController extends Controller
{
    public function index()
    {
        //
    }

    public function create()
    {
        //
    }
}
```

```
public function store(Request $request)
{
    //
}

public function show($id)
{
    //
}

public function edit($id)
{
    //
}

public function update(Request $request, $id)
{
    //
}

public function destroy($id)
{
    //
}
```

Mari kita tambahkan route untuk controller ini:

app/Http/routes.php

```
Route::resource('orders', 'OrdersController');
```

Oke, itu cara buat controller dan route nya. Saya yakin, itu cukup cepat. Lantas, apa yang kita dapatkan? Coba jalankan perintah ini..

```
$ php route:list
+-----+-----+-----+
| Method | URI           | Name        | Action      |
+-----+-----+-----+
| GET|HEAD | orders       | orders.index | OrdersController@index |
| POST     | orders       | orders.store | OrdersController@store |
| GET|HEAD | orders/create | orders.create | OrdersController@create |
| DELETE   | orders/{orders} | orders.destroy | OrdersController@destroy |
| PATCH    | orders/{orders} |               | OrdersController@update |
| GET|HEAD | orders/{orders} | orders.show  | OrdersController@show |
| PUT      | orders/{orders} | orders.update | OrdersController@update |
| GET|HEAD | orders/{orders}/edit | orders.edit | OrdersController@edit |
+-----+-----+-----+
```

Note: sebagian outputnya saya potong untuk memudahkan pemahaman.

Wow! Bisa kita lihat disini, Laravel secara otomatis membuatkan URL berikut HTTP Verb yang sesuai untuk setiap proses CRUD. Tidak hanya itu, Laravel juga memasangkan setiap URL tersebut ke method yang sesuai pada `OrdersController` dan memberikan nama untuk setiap route yang dibuat.

Secara sederhana, REST adalah suatu teknik dalam membuat API dimana *action* yang dilakukan oleh tidak hanya ditentukan oleh URL nya, tapi berikut HTTP Verb yang digunakan pula. Misalnya, url `/orders` akan memanggil method yang berbeda jika dipanggil dengan Verb `GET` (method `index`) atau `POST` (method `store`).

Mari kita pahami apa yang diharapkan oleh Laravel pada setiap method yang telah dibuat pada `OrdersController`:

- `index`: method ini biasanya digunakan untuk menampilkan semua resources. Disini, kita akan menampilkan semua order yang telah dibuat.
- `store`: method ini untuk menerima form pembuat resource baru. Disini, kita akan memproses form order sehingga menjadi order di database.
- `create`: method ini untuk menampilkan form pembuatan resource. Disini, kita akan memanggil view yang berisi form pembuatan order.
- `destroy`: method ini digunakan untuk menghapus resource. Terlihat diatas, Laravel menggunakan verb `DELETE` untuk method ini. Itu artinya, method ini dipanggil dari sebuah form.

- `edit`: method ini digunakan untuk menampilkan form untuk mengupdate resource. Pada URL untuk method ini (dan beberapa method lain), Laravel menggunakan parameter `{orders}` untuk menandai resource yang sedang aktif. Parameter ini bisa berisi `id`, `slug` dsb. Intinya, dengan parameter ini pada method `edit` kita akan mencari resource yang diinginkan.
- `update`: method ini yang akan menerima isian form dari view yang ditampilkan pada method `edit`.
- `show`: method ini digunakan untuk menampilkan detail dari sebuah resource. Dalam contoh di lapangan, tidak hanya isian dari model yang akan ditampilkan, bisa saja kita menampilkan isian relasi dari sebuah resource pada method ini.

Untuk memudahkan pembuatan proses CRUD, saya sendiri biasanya membuat file kosong untuk view pada folder `resources/views`. Untuk CRUD orders, maka ini struktur view yang akan saya buat:

```
.  
└── orders  
    ├── _form.blade.php  
    ├── create.blade.php  
    ├── edit.blade.php  
    └── index.blade.php  
        └── show.blade.php
```

Berikut penjelasan dari file-file tersebut:

- `index.blade.php`: Disini kita akan menampilkan semua order.
- `show.blade.php`: Disini kita akan menampilkan detail dari sebuah order.
- `create.blade.php`: Disini kita menampilkan form untuk membuat order.
- `edit.blade.php`: Disini kita menampilkan form untuk mengubah order.
- `_form`: underscore pada nama file menandakan bahwa view ini merupakan sebuah partial. File ini akan kita gunakan untuk menyimpan form pada view `create` dan `edit`.

Menampilkan Semua Resource

Kita mulai proses CRUD ini dari membuat halaman yang akan menampilkan semua order. Menurut struktur REST, kita harus membuatnya pada url `/orders` dengan method `GET`. URL tersebut mengarah pada method `index`. Isinya akan seperti berikut:

app/Http/Controllers/OrdersController.php

```
...
public function index()
{
    return view('orders.index');
}
...
```

Pada method ini kita menampilkan view `orders.index` yang akan menampilkan semua order. Bentuk sederhana dari file ini seperti berikut:

resources/views/orders/index.blade.php

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

@section('content')
<h1>Domino Martabak - Order</h1>
<table class="table table-hover table-striped">
    <thead>
        <tr>
            <th>Customer</th>
            <th>Tipe</th>
            <th>Jumlah</th>
            <th>Alamat</th>
            <th>Tanggal Order</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (App\Order::all() as $order)
        <tr>
            <td>{{ $order->customer }}</td>
            <td>{{ $order->tipe }}</td>
            <td>{{ $order->jumlah }}</td>
            <td>{{ $order->alamat }}</td>
            <td>{{ $order->created_at->format('M d, Y') }}</td>
            <td>edit | delete</td>
        </tr>
    @endforeach
    </tbody>
</table>
@stop
```

Pada view ini, kita menggunakan `@foreach` untuk *me-looping* semua isi dari table `orders` melalui model `App\Order`. Saya rasa setiap field nya sudah cukup jelas. Field terakhir yang berisi `edit | delete` akan kita isi dengan konten asli pada langkah-langkah selanjutnya.

Untuk mempercantik tampilan, pada view diatas kita meng-*extends* view `layouts/master`. View ini akan menggunakan bootstrap yang telah di sediakan oleh Elixir. Untuk itu, silahkan konfigurasi Elixir sehingga didapatkan CSS Bootstrap pada `public/css/app.css`. Isi dari file layout ini akan seperti berikut:

resources/views/layouts/master.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Domino Martabak</title>
    <link rel="stylesheet" href="{{ asset('css/app.css') }}>
</head>
<body>
    <div class="container">
        @yield('body')
    </div>
</body>
</html>
```

Tampilan akhir dari view ini akan seperti berikut:



Domino Martabak - Order

Customer	Tipe	Jumlah	Alamat	Tanggal Order	
Doni	coklat-kacang	3	Jl. Garuda 20 New York	Sep 02, 2015	edit delete
Rini	keju	2	Jl. Juara 12 New York	Sep 02, 2015	edit delete

Menampilkan semua order

Membuat Resource

Tahapan selanjutnya dari proses CRUD ini adalah membuat halaman untuk membuat order. Untuk membuatnya, kita akan menggunakan package `laravelcollective\html` yang sudah kita jelaskan pada bab view. Silahkan Anda install terlebih dahulu (jika lupa langkahnya, cek bab View).

....

Oke, saya anggap Anda sudah selesai. Dalam REST, untuk mengakses halaman untuk membuat resource baru biasanya diakses pada URL `/orders/create` dengan method GET. Selanjutnya, isian form dari halaman tersebut akan dikirim ke URL `/orders` dengan method POST.

Pada `OrdersController`, method untuk menampilkan form ini adalah `create`:

app/Http/Controllers/OrdersController.php

```
....  
public function create()  
{  
    return view('orders.create');  
}  
....
```

Method ini akan menampilkan view `orders/create.blade.php`. Isinya seperti berikut:

resources/views/orders/create.blade.php

```
@extends('layouts.master')  
  
@section('content')  
<h1>Order Baru</h1>  
{!! Form::open(['route' => 'orders.store']) !!}  
  
@include('orders._form')  
  
{!! Form::close() !!}  
@sto
```

Syntax `{!! Form::open(['route' => 'orders.store']) !!}` menandakan kita akan mengirim isian form ke route `orders.store`. Yang jika kita cek dengan perintah `php artisan route:list` akan mengarah ke method `store` pada `OrdersController`.

Seperti dijelaskan pada bagian sebelumnya, kita akan menggunakan form yang sama untuk membuat dan mengubah sebuah order. Pada `@include('orders._form')` kita memanggil view `orders._form` untuk ditampilkan pada halaman ini. Isian dari `orders/_form.blade.php` akan seperti berikut:

resources/views/orders/_form.blade.php

```
<div class="form-group">
{!! Form::label('customer', 'Customer') !!}
{!! Form::text('customer', null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
{!! Form::label('tipe', 'Tipe') !!}
{!! Form::select('tipe',
['kacang'=>'Kacang', 'coklat-kacang'=>'Coklat Kacang', 'keju'=>'Keju'],
null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
{!! Form::label('jumlah', 'Jumlah') !!}
{!! Form::text('jumlah', null, ['class'=>'form-control']) !!}
</div>

<div class="form-group">
{!! Form::label('alamat', 'Alamat') !!}
{!! Form::textarea('alamat', null, ['class'=>'form-control']) !!}
</div>

{!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
```

Untuk mempermudah, kita menggunakan array biasa untuk menentukan tipe martabak. Hasil akhir dari view ini akan seperti berikut:

The screenshot shows a web browser window titled "Domino Martabak". The address bar displays "localhost:8000/orders/create". The main content is a form titled "Order Baru". The form has four input fields: "Customer" (empty), "Tipe" (dropdown menu showing "Kacang"), "Jumlah" (empty), and "Alamat" (large text area). At the bottom is a blue "Simpan" button.

Membuat order baru

Untuk menerima isian form ini, kita harus mengisi method `store` pada `OrdersController`:

app/Http/Controllers/OrdersController.php

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

Pada method `store`, kita mendapatkan instance dari `Illuminate\Http\Request` sebagai variable `$request`. Dari class ini, kita dapat mengakses berbagai property dari form. Berikut beberapa method yang dapat kita gunakan:

- `all()`: Mengakses semua isian form sebagai array assosiatif
- `input()`: Untuk mendapatkan isian dari sebuah field
- `has()`: Untuk mengecek apakah sebuah field memiliki isi

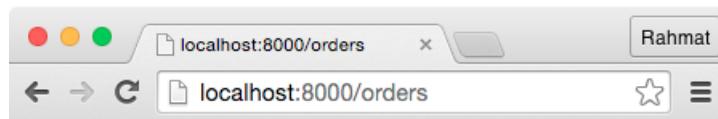
Method lainnya, dapat kita cek pada [dokumentasi untuk request¹⁴⁸](#).

Jika kita ingin mengecek output dari perintah diatas, kita dapat menggunakan method `dd()`. Misalnya seperti berikut:

app/Http/Controllers/OrdersController.php

```
....  
public function store(Request $request)  
{  
    dd($request->all());  
}  
....
```

Cobalah mengisi form pembuatan order dan men-*submit*-nya. Output yang akan kita dapatkan akan seperti ini:



The screenshot shows a Mac OS X desktop with a browser window open to `localhost:8000/orders`. The browser title bar says "localhost:8000/orders". The page content displays the output of the `dd($request->all())` code, which is an array with the following items:

```
array:5 [▼  
  "_token" => "M1DmtRBymU5DqVRcBI8dQ03FulxBQT0mlFtE5XOU"  
  "customer" => "Beni"  
  "tipe" => "kacang"  
  "jumlah" => "4"  
  "alamat" => "Jl. Satria No. 14 Jogja"  
]
```

Mengecek method `all()` pada `$request`

Method `dd()` akan sangat berguna untuk mengecek isi form, terutama jika terjadi error selama proses pembuatan resource.

Oke, kita balik lagi ke proses pembuatan order. Syntax untuk membuat order akan seperti berikut:

¹⁴⁸<http://laravel.com/docs/5.1/requests>

app/Http/Controllers/OrdersController.php

```
public function store(Request $request)
{
    \App\Order::create($request->all());
    return redirect()->route('orders.index');
}
```

Disini kita menggunakan method `create` pada model `\App\Order` untuk membuat order baru. Agar method ini bisa berjalan, kita harus mengatur `mass assignment` pada model:

app/Order

```
...
class Order extends Model
{
    protected $fillable = ['customer', 'tipe', 'jumlah', 'alamat'];
}
```

Pada syntax `return redirect()->route('orders.index');` kita me-*redirect* used ke route `orders.index` yang akan menampilkan semua order.

Cobalah untuk mengisi dan men-*submit* form ini. Pastikan order berhasil dibuat dan kita di-*redirect* ke halaman index.

Mengubah Resource

Secara URL untuk menampilkan halaman untuk mengubah resource adalah `/orders/{orders}/edit` kita harus mengaksesnya dengan method `GET`. URL ini akan mengakses method `edit` pada controller. Syntax `{orders}`, menandakan kita harus memberikan pengenal untuk model yang akan kita gunakan. Biasanya diisi dengan `id` atau `slug` dari model. Pada contoh ini, kita akan menggunakan `id`. Penggunaan `slug` biasanya akan sangat bermanfaat dalam membuat URL yang lebih SEO friendly.

`slug` adalah teknik penandaan resource dengan menggunakan url friendly version dari namanya. Misalnya, kita memiliki resource lagu dengan nama Tuhan Tahu Kita Mampu, maka slugnya akan tuhan-tahu-kita-mampu. Di Laravel, kita dapat menggunakan helper `str_slug()`¹⁴⁹ untuk menggenerate slug.

Hasil dari form yang kita tampilkan harus kita kirim ke URL `orders/{orders}` dengan method `PUT`. URL ini akan mengakses method `update` pada controller.

Mari kita isi dulu method `edit` yang akan menampilkan form untuk mengedit order:

¹⁴⁹ <http://laravel.com/docs/5.1/helpers#method-str-slug>

app/Http/Controllers/OrdersController.php

```
....  
public function edit($id)  
{  
    $order = \App\Order::findOrFail($id);  
    return view('orders.edit', compact('order'));  
}  
....
```

Pada method ini, kita menggunakan method `findOrFail()` untuk mencari model. Method ini kita gunakan agar aplikasi menunjukkan error 500 ketika kita tidak menemukan model. Syntax `compact('order')` merupakan *shorthand* dari syntax array `['order'=>$order]`, ini kita gunakan untuk mengirim variable `$order` ke view.

Mari kita buat viewnya:

resources/views/orders/edit.blade.php

```
@extends('layouts.master')  
  
@section('content')  
<h1>Ubah Order untuk {{ $order->customer }}</h1>  
{!! Form::model($order, ['route' => ['orders.update', $order], 'method'=>'PUT'])\br/>!!}  
  
@include('orders._form')  
  
{!! Form::close() !!}  
@stop
```

Pada form ini, kita menggunakan fitur form model binding yang telah kita pelajari pada bab View agar setiap field terisi dengan data dari database secara otomatis. Syntax

```
{!! Form::model($order, ['route' => ['orders.update', $order], 'method'=>'PUT'])\br/>!!}
```

kita gunakan untuk menampilkan form binding ke model `$order` (yang kita kirim dapatkan dari controller dengan action ke `/orders/{orders}` dengan method `PUT`). Kita juga menggunakan form yang sama dengan form `create`. Berikut tampilan dari halaman ini:

Customer
Dadang

Tipe
Coklat Kacang

Jumlah
3

Alamat
Jl. Garuda 20 New York

Simpan

Halaman edit order

Untuk menerima form ini, mari kita isi method update:

app/Http/Controllers/OrdersController.php

```
....  
public function update(Request $request, $id)  
{  
    $order = \App\Order::findOrFail($id);  
    $order->update($request->all());  
    return redirect()->route('orders.index');  
}  
....
```

Pada syntax ini, kita mencari order baru dengan method `findOrFail()`. Variable `$id`

merupakan isian `{orders}` pada URL yang diakses. Kita mengupdate order dengan method `update()`. Terakhir, kita me-*redirect* user ke halaman `/orders`.

Untuk memudahkan, mari kita isi link `edit` pada halaman `/orders` ke URL yang benar:

resources/views/orders/index.blade.php

```
....  
<td><a href="{{ route('orders.edit', $order) }}">edit</a> | delete</td>  
....
```

Cobalah merubah data sebuah order dan menyimpannya kembali. Pastikan semua prosesnya berhasil.

Menghapus Resource

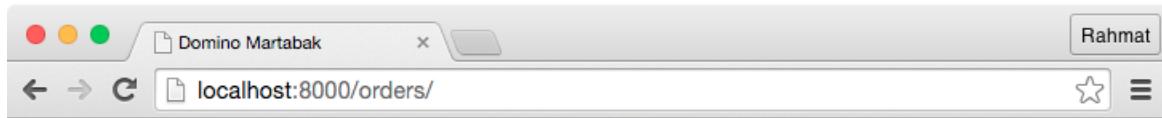
Untuk menghapus resource, kita harus memanggil URL `/orders/{orders}` dengan method `DELETE`. Seperti biasa, isian `{orders}` merupakan pengenal dari model. Pada contoh ini, kita menggunakan `id`.

Dalam menghapus, kita tidak perlu membuat halaman baru. Sehingga, kita dapat menambahkan form ini pada halaman `/orders`:

resources/views/orders/index.blade.php

```
....  
<td>  
{!! Form::model($order, ['route' => ['orders.destroy', $order], 'method'=>'delet\ e', 'class' => 'form-inline'])!!}  
    <a href="{{ route('orders.edit', $order) }}">edit</a> | {!! Form::submit('dele\ te', ['class'=>'btn btn-xs btn-danger']) !!}  
{!! Form::close() !!}  
</td>  
....
```

Pada form ini, kita menggunakan method `delete`. Untuk merapikan tampilan kita menggunakan class `form-inline` dari bootstrap agar form menjadi sebaris dengan tombol edit dan memasukan link `edit` ke dalam form. Tampilannya akan seperti berikut:



Domino Martabak - Order

Customer	Tipe	Jumlah	Alamat	Tanggal Order	
Dadang	coklat-kacang	3	Jl. Garuda 20 New York	Sep 02, 2015	edit delete
Rini	keju	2	Jl. Juara 12 New York	Sep 02, 2015	edit delete
Budi	coklat-kacang	4	Kp. Melayu no. 7 Surabaya	Sep 06, 2015	edit delete

Menambahkan tombol delete order

Untuk menerima form delete ini, mari kita ubah method 'destroy':

app/Http/Controllers/OrdersController.php

```
.....
public function destroy($id)
{
    \App\Order::findOrFail($id)->delete();
    return redirect()->route('orders.index');
}
....
```

Pada method ini, kita menghapus Order berdasarkan ID yang dikirim kemudian me-*redirect* user ke halaman /orders.

Cobalah menghapus order dan pastikan semuanya berjalan lancar.

Menampilkan Resource

Langkah terakhir dalam struktur REST adalah menampilkan resource. Kita mengaksesnya dengan mengunjungi URL /orders/{orders} dengan method GET. Method yang akan digunakan untuk proses ini adalah method show:

app/Http/Controllers/OrdersController.php

```
....  
public function show($id)  
{  
    $order = \App\Order::findOrFail($id);  
    return view('orders.show', compact('order'));  
}  
....
```

Cukup sederhana, disini kita mencari model order dan menampilkan view orders/show.blade.php:

resources/views/orders/show.blade.php

```
@extends('layouts.master')  
  
@section('content')  
<h1>Detail pesanan {{ $order->customer }}</h1>  
<table class="table table-condensed">  
    <tr>  
        <th>Tipe</th>  
        <td>{{ $order->tipe }}</td>  
    </tr>  
    <tr>  
        <th>Jumlah</th>  
        <td>{{ $order->jumlah }}</td>  
    </tr>  
    <tr>  
        <th>Alamat</th>  
        <td>{{ $order->alamat }}</td>  
    </tr>  
    <tr>  
        <th>Tanggal Order</th>  
        <td>{{ $order->created_at->format('M d, Y') }}</td>  
    </tr>  
</table>  
<a href="{{ route('orders.index') }}">Kembali ke semua order</a>  
@stop
```

Untuk memudahkan, mari kita tambah link pada nama customer di halaman utama untuk melihat detail dari order:

resources/views/orders/index.blade.php

```
....  
<td><a href="{{ route('orders.show', $order) }}>{{ $order->customer }}</a></td>
```

Hasil akhir dari kedua view diatas akan seperti berikut:



Domino Martabak - Order

Customer	Tipe	Jumlah	Alamat	Tanggal Order	
Dadang	coklat-kacang	3	Jl. Garuda 20 New York	Sep 02, 2015	edit delete
Rini	keju	2	Jl. Juara 12 New York	Sep 02, 2015	edit delete

localhost:8000/orders/1

Menambah link untuk melihat detail order



Detail pesanan Dadang

Tipe	coklat-kacang
Jumlah	3
Alamat	Jl. Garuda 20 New York
Tanggal Order	Sep 02, 2015

[Kembali ke semua order](#)

localhost:8000/orders

Melihat detail order

Cobalah untuk mengunjungi detail dari setiap order. Pastikan semua dapat berjalan dengan benar.

Sip. Dengan selesainya fitur terakhir dari REST ini, telah lengkaplah fitur CRUD sederhana untuk order ini. Dapat kita lihat, proses pembuatan CRUD sangat dimudahkan dengan adanya RESTful Controller dari Laravel.



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁵⁰

Dependency Injection

Di bab 3, kita pernah membahas mengenai fitur dari Service Container untuk melakukan Automatic Resolution. Pada controller, umumnya jika kita ingin menggunakan fitur automatic resolution, kita harus memasukkannya pada method `__construct`.

Misalnya, kita ingin menambahkan Log dengan class yang mengimplementasikan `Psr\Log\LoggerInterface` setiap kali kita membuat Order pada aplikasi Domino Martabak, pertama kita import interface tersebut:

app/Http/Controllers/OrdersController.php

```
<?php  
namespace App\Http\Controllers;  
....  
use Psr\Log\LoggerInterface as PsrLoggerInterface;  
....
```

Kedua kita tambahkan dependency di `__constructor` dan menyimpannya sebagai atribut `$log`:

app/Http/Controllers/OrdersController.php

```
....  
class OrdersController extends Controller  
{  
    protected $log;  
  
    public function __construct(PsrLoggerInterface $log)  
    {  
        $this->log = $log;  
    }  
....
```

Terakhir, kita gunakan pada method `store`:

¹⁵⁰ <https://bitbucket.org/rahmatawaludin/domino-martabak/commits>

app/Http/Controllers/OrdersController.php

```
....  
public function store(Request $request)  
{  
    $order = \App\Order::create($request->all());  
    $this->log->info('Order baru telah dibuat. ID: ' . $order->id);  
    return redirect()->route('orders.index');  
}  
....
```

Secara default interface `Psr\Log\LoggerInterface` akan di resolve ke class `Illuminate\Log\Writer` yang akan menyimpan log ke `storage/logs/laravel.log`. Cobalah buat order baru, dan pastikan terdapat isian berikut pada file log:

storage/logs/laravel.log

```
[xxxx-xx-xx xx:xx:xx] local.INFO: Order baru telah dibuat. ID: X
```



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁵¹

Menginject dependensi pada `constructor` sudah umum dilakukan. Namun, jika dependensi tersebut hanya digunakan pada sebuah method di controller, kita dapat menggunakan fitur method injection di Laravel.

Misalnya, kita ingin membuat log juga setiap kali order dihapus. Menggunakan method injection, ini yang akan kita lakukan:

app/Http/Controllers/OrdersController.php

```
....  
public function destroy(PsrLoggerInterface $log, $id)  
{  
    \App\Order::findOrFail($id)->delete();  
    $log->info("Order telah dihapus. ID: " . $id);  
    return redirect()->route('orders.index');  
}  
....
```

¹⁵¹ <https://bitbucket.org/rahmatawaludin/domino-martabak/commits/6bcd0eacbc8758ce812cdac589ab2b1de628ba33>

Terlihat disini, dengan method injection kita membuat code lebih singkat. Pertama, kita tidak perlu mempersiapkan attribut khusus untuk meyimpan instance dari `Psr-LoggerInterface`. Kedua, kita tidak perlu mempersiapkan `PsrLoggerInterface` pada `constructor`. Dan terakhir, cara kita mengakses nya pun tidak `$this->log`, tapi cukup `$log`.

Cobalah hapus sebuah order, dan pastikan pesan berikut muncul di file log:

storage/logs/laravel.log

```
[xxxx-xx-xx xx:xx:xx] local.INFO: Order telah dihapus. ID: X
```

Sebenarnya, tanpa kita sadari kita telah menggunakan fitur ini pada saat membuat dan meng-update order. Dapat kita lihat pada method `store` dan `update`, kita meng-*inject* `Illuminate\Http\Request`:

app/Http/Controllers/OrdersController.php

```
....  
public function store(Request $request)  
....  
public function update(Request $request, $id)  
....
```

Laravel cukup cerdas. Ketika kita menginject dependency ke sebuah method, dia tidak akan menganggap dependency itu merupakan parameter dari URL seperti `$id` pada method `update`.

Manfaat lain dari menggunakan method injection tentunya memubuat code kita lebih testable dan maintainable. Jika aplikasi kita membutuhkan banyak dependensi pada controller, method injection ini akan sangat bermanfaat. Sip.



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁵²

Route Caching

Ketika kita menggunakan routing di Laravel, khusunya implicit routing performa nya sebenarnya tidak terlalu bagus. Untuk itu, jika aplikasi kita sudah berada di mode produksi, sangat disarankan menggunakan route caching.

Fitur ini bisa diaktifkan dengan menjalankan perintah `php artisan route:cache`. Ada syarat untuk menggunakan fitur ini, yaitu semua route kita harus dilayani oleh controller, bukan closure. Contohnya, pada aplikasi Domino Martabak, jika kita jalankan perintah ini:

¹⁵²<https://bitbucket.org/rahmatawaludin/domino-martabak/commits/c5bd8cf84c57464fbc0370b07db68b1acddeb4fa>

```
$ php artisan route:cache
Route cache cleared!

[LogicException]
Unable to prepare route [/] for serialization. Uses Closure.
```

Ini terjadi karena kita menggunakan closure pada route /.

app/Http/routes.php

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

Solusinya, kita dapat membuat controller baru untuk menghandle route ini. Misalnya kita buat HomeController dan kita isi logic untuk menampilkan view pada method index:

```
$ php artisan make:controller HomeController --plain
Controller created successfully.
```

app/Http/Controllers/HomeController.php

```
.....
class HomeController extends Controller
{
    public function index()
    {
        return view('welcome');
    }
}
....
```

Dan kita ubah route:

app/Http/routes.php

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

Kini, perintah `route:cache` akan berhasil:

```
$ php artisan route:cache  
Route cache cleared!  
Routes cached successfully!
```

Cara kerja dari route cache adalah dengan membuat file cache di `bootstrap/cache/routes.php`. Silahkan buka file tersebut. Dengan menggunakan file ini, Laravel tidak perlu melakukan parsing ke file `app/Http/routes.php` untuk melihat semua route yang tersedia.

Yang perlu diperhatikan adalah ketika route caching aktif, maka perubahan apapun pada file `app/Http/routes.php` tidak akan berefek. Misalnya, kita hapus route untuk route `/`.

app/Http/routes.php

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

Route ini masih akan tetap terdaftar di aplikasi. Kita dapat mengeceknya dengan perintah:

```
$ php artisan route:list
```

Domain	Method	URI	Name	Action
	GET HEAD	/		App\Http\Controllers\HomeController@index
	

Untuk mengatasinya, kita perlu menjalankan menghapus file cache secara manual dengan perintah `php artisan route:clear` atau dengan meng-generate file cache baru dengan perintah `php artisan route:cache`.

Nah, kadang biasanya suka lupa untuk melakukan hal ini. Bukan meningkatkan performa, fitur ini malah nambah masalah karena perubahan pada route tidak berefek. Makanya, route caching ini disarankan hanya digunakan di aplikasi yang sudah *live* bukan aplikasi dalam status *development*. Salah satu tekniknya adalah dengan menjalankan perintah `route:cache` setiap kali kita deploy ke server. Sip.



Source code dari latihan ini bisa didapat di [Bitbucket](#)¹⁵³

¹⁵³<https://bitbucket.org/rahmatawaludin/domino-martabak/commits/52610cfbe628a8bbe965f422d4a13cc7748dc1db>

Whats Next?

Pada bab ini kita telah mempelajari penggunaan controller untuk mengarahkan request di aplikasi. Dalam aplikasi real, request tidak hanya diarahkan. Lebih dari itu, terkadang kita harus melakukan validasi data atau melakukan pembatasan akses. Untuk itu, ada pembahasan yang akan sangat erat dengan controller yaitu:

- Controller Middleware dibahas di bab Middleware. Middleware akan kita gunakan untuk melakukan action, misalnya pembatasan akses pada controller
- Validasi dibahas di bab Validasi Data. Kita akan belajar mengenai validasi data menggunakan `request` maupun `form request`.



Ringkasan

Pada bab ini kita telah mempelajari bagaimana controller menerima request dan memberikan response. Disini, kita baru belajar memberikan respon berupa view dan redirect. Bagaimana jika kita ingin memberikan response lain, misalnya file download, json, dll?

Jawabannya, pelajari bab selanjutnya! :)

Request & Response

Dalam membangun web, dua hal ini memang tidak bisa dipisahkan. Dan, dalam prinsipnya, membangun web adalah membuat aplikasi kita memberikan `response` yang sesuai berdasarkan `request` yang diminta oleh user.

Pada beberapa bab sebelumnya, kita telah mempelajari bagaimana memberikan `response` berupa `view` di Laravel. Tentunya, masih dengan cara yang biasa. Di Laravel, kita dapat melakukan banyak hal dengan `request`, misalnya mengecek jenis `request`, data yang dikirim dan info tentang `request`.

Begitupun dengan `response`. Selain `response` berupa `view`, kita juga dapat memberikan respon jenis lain misalnya download file, json, dsb. Tentunya, selain jenis konten yang berbeda, di Laravel kita juga dapat memodifikasi header dan status code dari `response` yang kita berikan.



Status Code adalah teknik penomoran dalam memberikan respon HTTP. Dengan menggunakan teknik penomoran ini, kita dapat mengetahui status respon yang kita dapatkan. Banyak sekali jenis status code, cek di [Wikipedia¹⁵⁴](#).

Memahami Request di Laravel

Kita dapat mengakses setiap `request` di Laravel dengan melakukan `resolve` terhadap `Illuminate\Http\Request` dari *Service Container*. Misalnya, kita dapat melakukan hal ini di route:

`app/Http/routes.php`

```
Route::get('/', function () {
    $request = app('Illuminate\Http\Request');
    dd($request);
});
```

Jika kita kunjungi route `/`, ini yang akan kita dapatkan:

¹⁵⁴https://en.wikipedia.org/wiki/List_of_HTTP_status_codes



```
Request {#39 ▾
  #json: null
  #userResolver: Closure {#126 ▶}
  #routeResolver: Closure {#128 ▶}
  +attributes: ParameterBag {#41 ▶}
  +request: ParameterBag {#47 ▶}
  +query: ParameterBag {#47 ▶}
  +server: ServerBag {#44 ▶}
  +files: FileBag {#43 ▶}
  +cookies: ParameterBag {#42 ▶}
  +headers: HeaderBag {#45 ▶}
  #content: null
  #languages: null
  #Charsets: null
  #encodings: null
  #acceptableContentTypes: null
  #pathInfo: "/"
  #requestUri: "/"
  #baseUrl: ""
  #basePath: null
  #method: "GET"
  #format: null
  #session: Store {#107 ▶}
  #locale: null
  #defaultLocale: "en"
}

Mengecek isi request
```

Oke, melihat output ini biasanya kurang begitu jelas apa manfaatnya. Mari kita coba gunakan beberapa method pada object ini. Misalnya:

- Mengakses HTTP Verb yang digunakan:

app/Http/routes.php

```
Route::get('/', function () {
    $request = app('Illuminate\Http\Request');
    return $request->method();
});
```

Output yang akan kita dapatkan:

GET

- Mengambil URL yang sedang diakses

app/Http/routes.php

```
Route::get('/contact/about', function () {
    $request = app('Illuminate\Http\Request');
    return $request->path();
});
```

Disini, kita menggunakan route /contact/about, maka output yang akan kita dapatkan ketika kita memanggil method path() adalah :

```
contact/about
```

Output yang akan kita dapatkan meskipun menambahkan query string akan tetap sama:

**Mengambil path tanpa query string**

Terlihat disini, cukup banyak info yang bisa kita dapatkan dari object ini. Lebih lengkapnya, kita bisa mengecek [dokumentasi API class ini¹⁵⁵](#).

Ketika kita hendak mengakses request dari controller, biasanya kita melakukan method injection. Misalnya, pada contoh aplikasi Domino Martabak kita dapat melihatnya pada method store di OrdersController:

```
public function store(Request $request)
{
    ...
}
```

Yang merupakan instance dari Illuminate\Http\Request. Dapat kita lihat dari statement use nya:

¹⁵⁵<http://laravel.com/api/5.1/Illuminate/Http/Request.html>

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

Menerima Input

Dalam menerima input, Laravel akan menggunakan cara yang sama untuk parameter yang kita kirim melalui request GET maupun POST. Mari kita buktikan dengan membuat controller untuk menerima sebuah request untuk yang mengirim nama dan email kemudian memberikan respon sederhana. Kita namakan controller ini HomeController. Jalankan perintah berikut untuk membuat controller tanpa method:

```
$ php artisan make:controller HomeController --plain  
Controller created successfully.
```

Mari kita buat method `index` yang akan menerima request yang akan kita kirim. Berikut code lengkap untuk method ini:

app/Http/Controllers/HomeController.php

```
<?php  
....  
use Illuminate\Http\Request;  
....  
  
class HomeController extends Controller  
{  
    public function index(Request $request)  
    {  
        $nama = $request->get('nama');  
        $email = $request->get('email');  
        return 'Halo ' . $nama . '. Email Anda : ' . $email;  
    }  
}
```

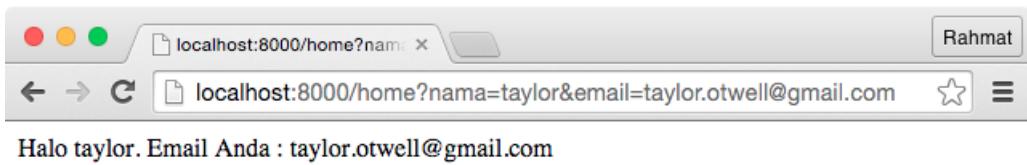
Kita buat route untuk method ini dengan nama `home`:

app/Http/routes.php

```
Route::any('/home', 'HomeController@index');
```

Disini, sengaja kita menggunakan `any` agar route ini bisa diakses dengan method GET maupun POST.

Mari kita coba akses dengan method GET, kunjungi url `/home?nama=taylor&email=taylor.otwell@gmail.com` dari browser:



Test GET request

Sip. Dapat kita lihat, kita berhasil mengambil parameter dari request GET dengan method `get()`. Selanjutnya, mari kita test request POST.

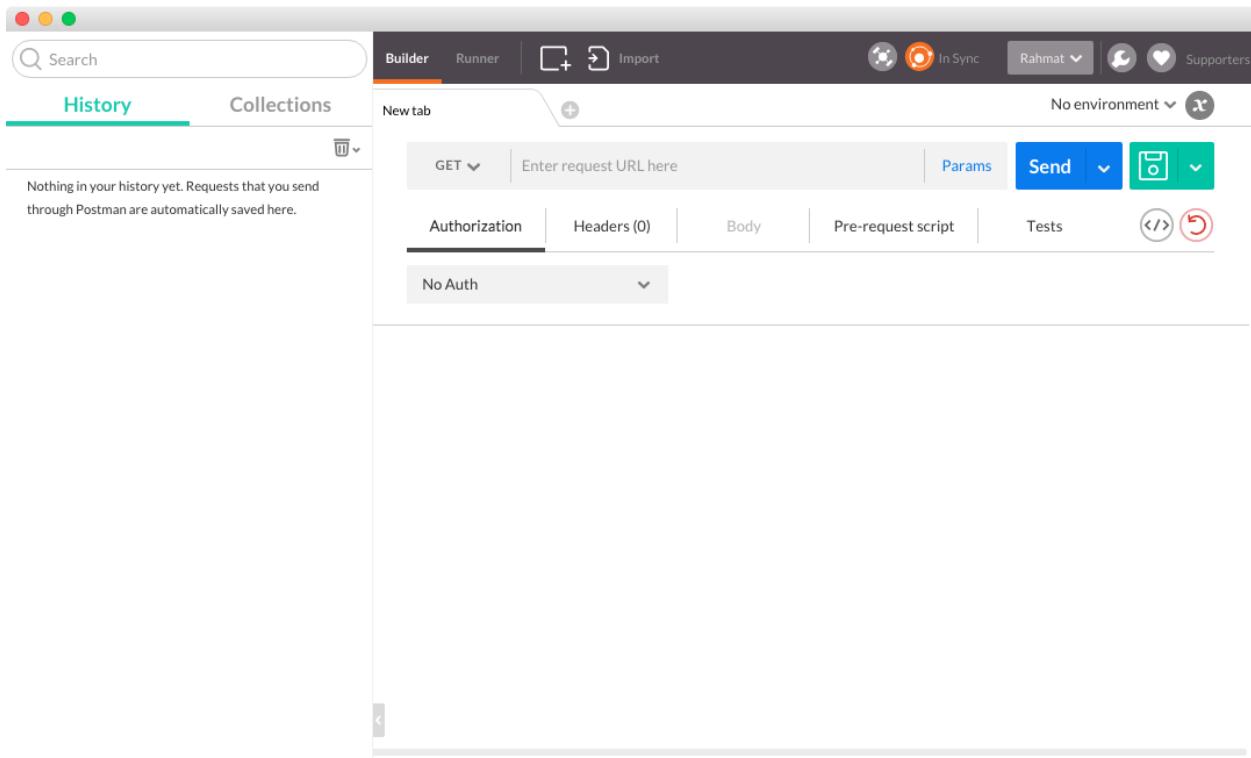
Umumnya, untuk mengetes request POST kita harus membuat form. Kali ini, kita akan menggunakan cara yang berbeda menggunakan tools bernama **Postman**.

Postman merupakan extensi dari Browser Chrome. Tentunya, untuk menggunakan nya kita harus menggunakan chrome. Untuk menginstallnya kunjungi <https://www.getpostman.com> dan ikuti cara installnya. Atau, langsung kunjungi link [berikut¹⁵⁷](#) menggunakan Google Chrome.

Tampilan awal saat buku ini ditulis akan seperti berikut:

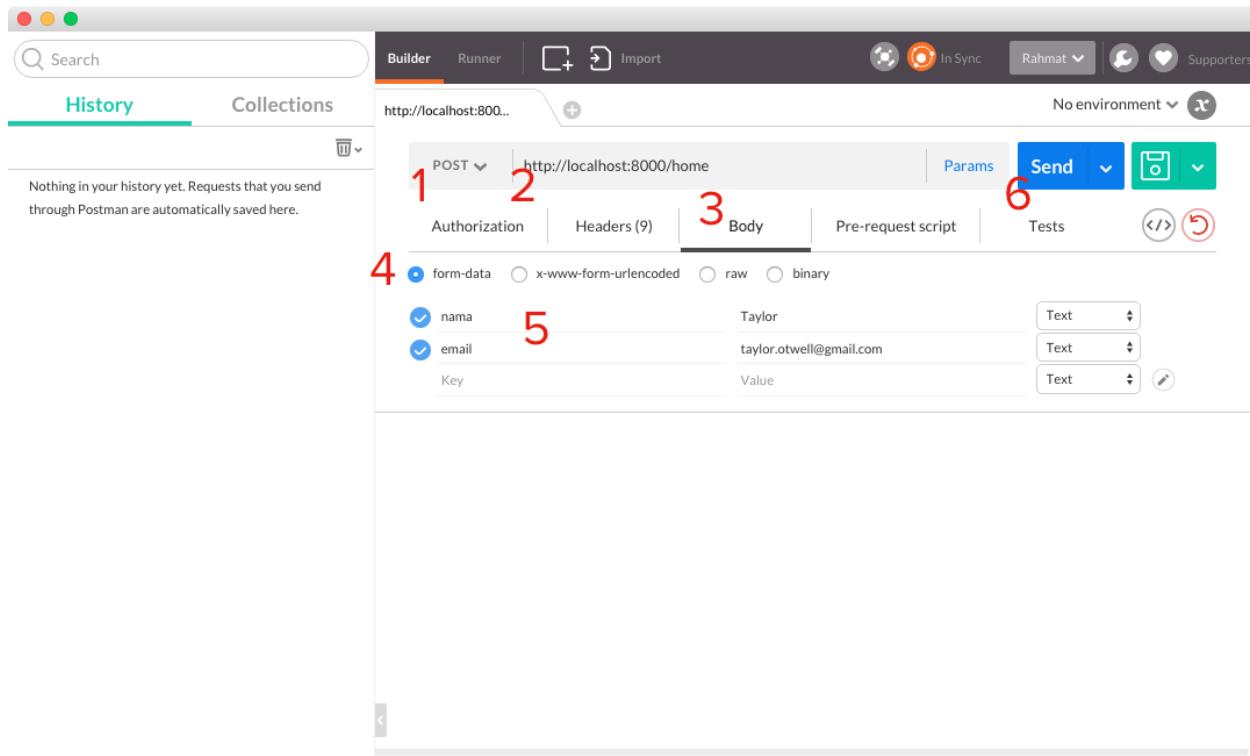
¹⁵⁶<https://www.getpostman.com>

¹⁵⁷<https://chrome.google.com/webstore/detail/postman-rest-client/fhbjgbiflinjbdggehcdcncddomop>



Tampilan Postman

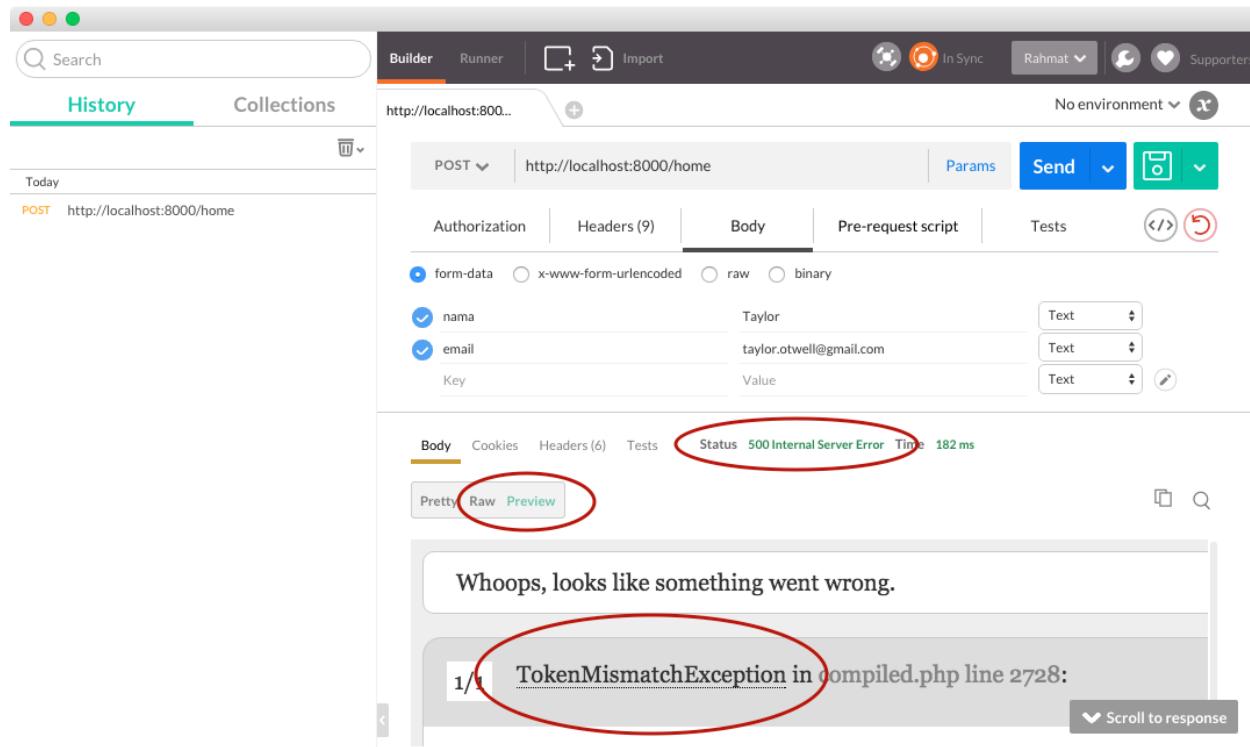
Fitur dari Postman ini cukup banyak. Mari kita fokus pada pembuatan POST request dengan parameter nama dan email. Untuk melakukannya, konfigurasi Postman seperti berikut:



konfigurasi Postman untuk POST request

1. Pilih tipe request POST
2. Masukan url yang akan kita akses
3. Pilih tab *body*
4. Pilih opsi *form-data*
5. Masukan parameter yang kita inginkan
6. Setelah semua terisi, klik *Send*.

Outputnya akan seperti berikut:



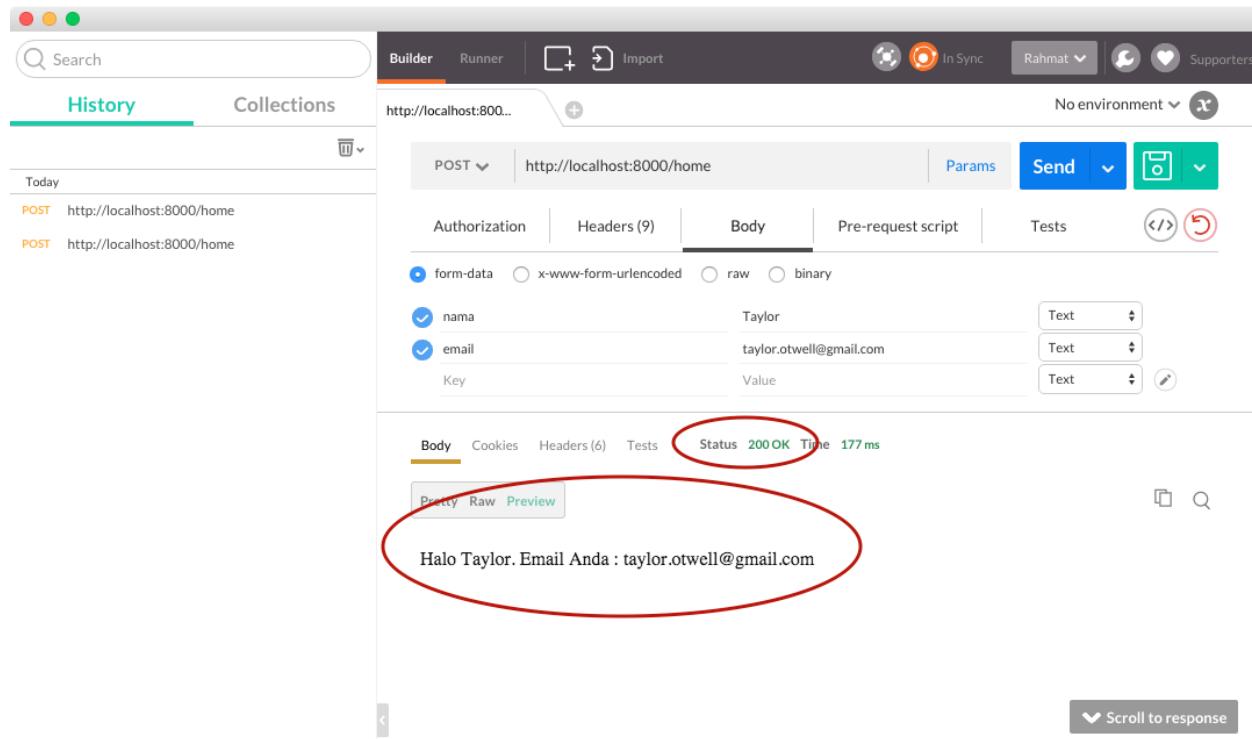
Postman error

Oooppsss... kita mendapat error *TokenMismatchException*. Ini terjadi karena Laravel secara default akan mengaktifkan proteksi CSRF. Untuk mengatasinya, kita dapat men-disable fitur ini sama sekali atau hanya pada route yang kita inginkan. Mari kita gunakan cara kedua, tambahkan isian `home` pada property `$except` di file `app/Http/Middleware/VerifyCsrfToken.php`:

`app/Http/Middleware/VerifyCsrfToken.php`

```
....  
protected $except = [  
    'home'  
];  
....
```

Coba lagi klik *Send* pada Postman, kini outputnya akan seperti berikut:



Post request berhasil

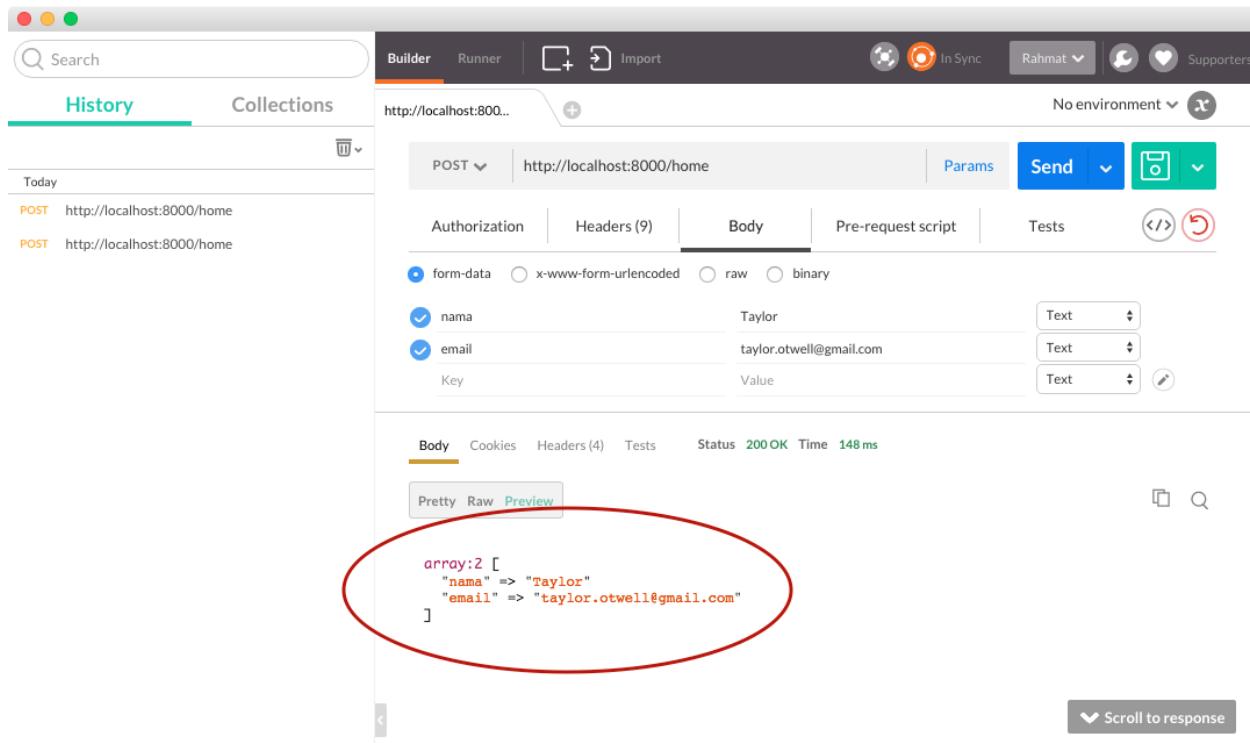
Terlihat disini, kita berhasil menerima input dari request GET maupun POST dengan method yang sama, yaitu `get()`. Terdapat method lain yang bisa kita gunakan:

- Mengambil semua input, gunakan method `all()`.

Output dari method ini adalah sebuah array asosiatif berisi nama field berikut nilainya. Kita dapat mengetesnya dengan syntax berikut:

app/Http/Controllers/HomeController.php

```
...  
public function index(Request $request)  
{  
    dd($request->all());  
}  
...
```

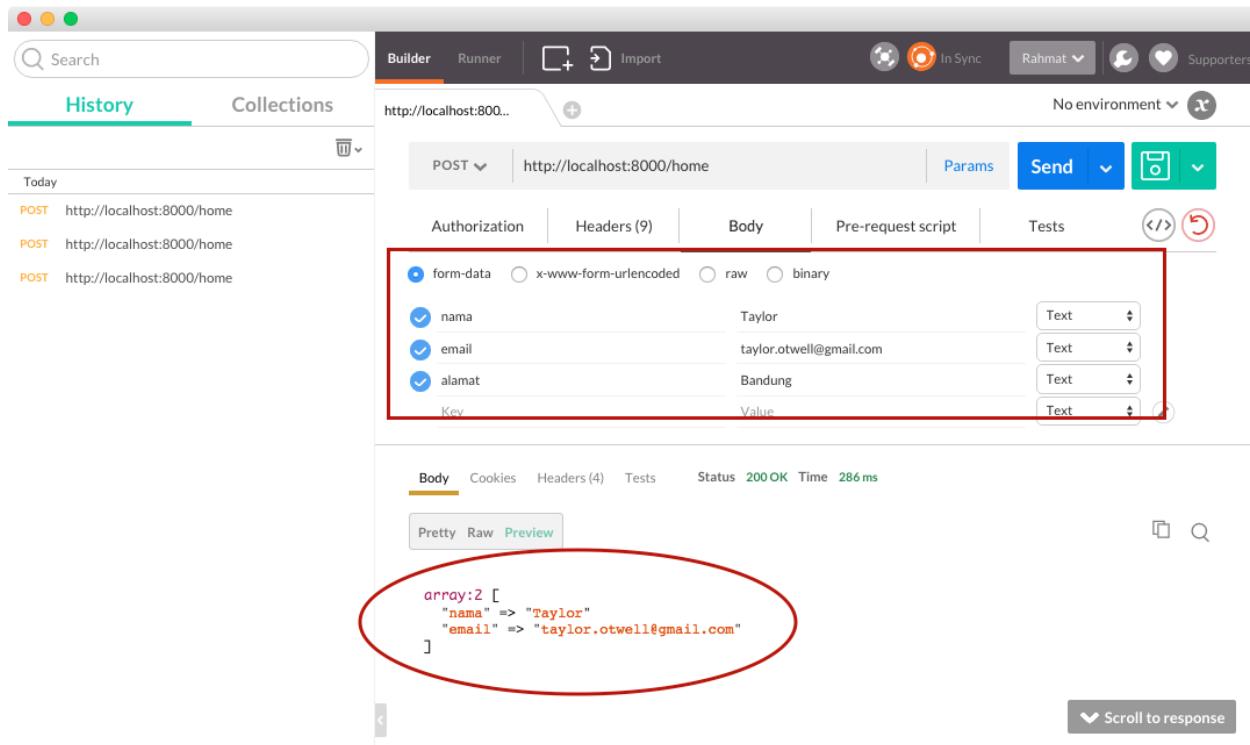


Hasil method all()

- Hanya mengambil parameter tertentu, gunakan method `only()`. Misalnya, kita hanya ingin mengambil parameter nama dan email meskipun user juga mengirim parameter alamat. Ini syntax yang akan kita buat:

app/Http/Controllers/HomeController.php

```
....  
public function index(Request $request)  
{  
    dd($request->only('nama', 'email'));  
}  
....
```

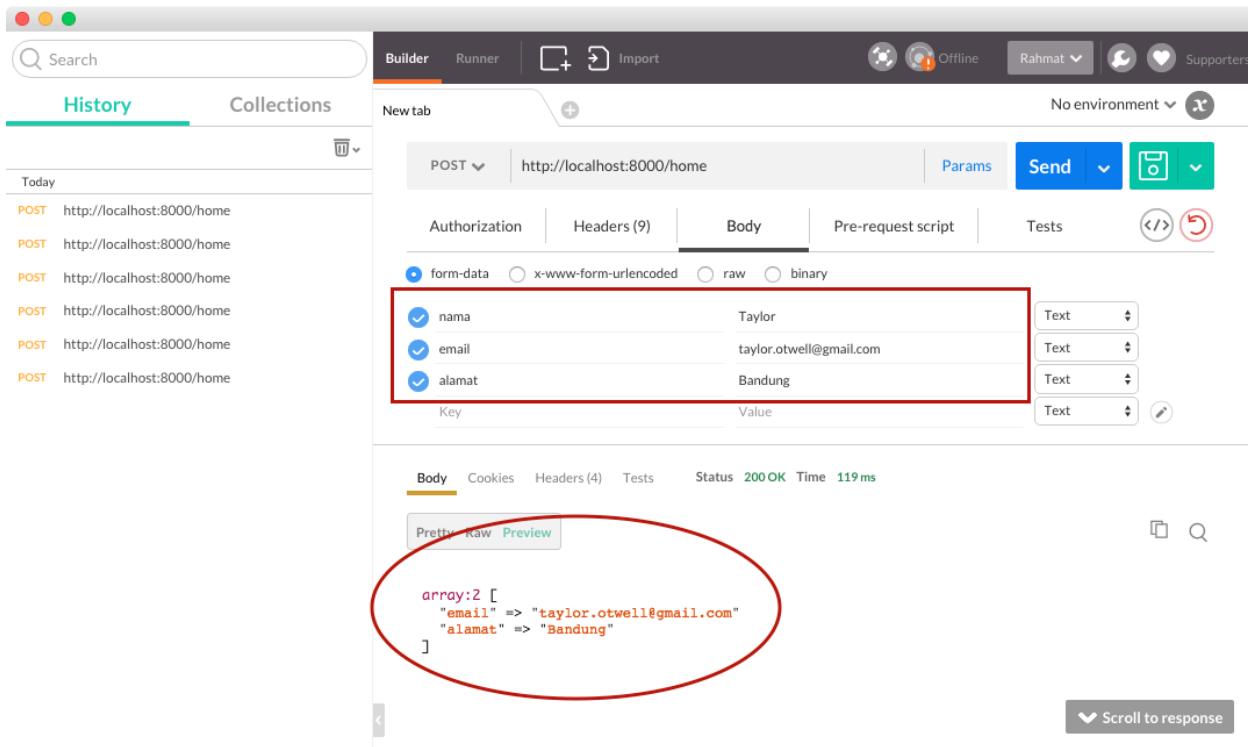


Hasil method only()

- Mengambil semua parameter, kecuali parameter tertentu gunakan `except()`. Misalnya, kita ingin mengambil semua parameter kecuali parameter `nama`:

app/Http/Controllers/HomeController.php

```
...
public function index(Request $request)
{
    dd($request->except('nama'));
}
...
```



Hasil method except()

- Mengecek apakah parameter tertentu telah diset, gunakan method `has()`. Misalnya, kita ingin mengecek apakah parameter alamat diisi, ini syntax yang akan kita buat:

`app/Http/Controllers/HomeController.php`

```
...
public function index(Request $request)
{
    dd($request->has('alamat'));
}
...
```

The screenshot shows the Postman application interface. In the top navigation bar, 'Builder' is selected. Below it, a search bar and a 'History' tab are visible. The main workspace shows a POST request to 'http://localhost:8000/home'. The 'Body' tab is active, showing form-data with three fields: 'nama' (Value: Taylor), 'email' (Value: taylor.otwell@gmail.com), and 'alamat' (Value: Bandung). The response section at the bottom shows the word 'true' highlighted with a red oval.

Hasil method has()

Old Input

Dalam bekerja dengan form, terkadang kita ingin membatalkan input dari user setelah diterima oleh server karena beberapa alasan. Bisa saja ada error disisi server, validasi, dsb. Untuk mengatasi hal ini, di Laravel kita dapat melakukan *flashing*, yaitu membuat input pada request sebelumnya tersedia pada request selanjutnya.

Misalnya, kita hendak melakukan flashing, untuk input `alamat` dan `email`, syntax di sisi backend akan seperti berikut:

```
$request->flashOnly('alamat', 'email');
```

Kita juga dapat melakukan flashing untuk semua input:

```
$request->flash();
```

Atau, semua input kecuali input tertentu, misalnya password:

```
$request->flashExcept('password');
```

Nah, untuk menerima flashing disisi frontend, kita dapat menggunakan syntax `old('nama-field')`.

Mari kita demokan, misalnya kita punya form dengan isian nama, alamat dan nomor ktp. Ceritanya, jika user menggunakan nomor ktp 123, maka form akan error dan mengembalikan user ke form dengan isian nama dan alamat yang masih terisi. Disini, kita belum akan menggunakan fitur validasi dari Laravel, agar kita lebih fokus pada fitur flashing.

Buatlah route berikut:

app/Http/routes.php

```
Route::get('registrasi', function() {
    return View::make('registrasi');
});
Route::post('registrasi', 'HomeController@registrasi');
```

View berikut:

resources/views/registrasi.blade.php

```
<form method="POST" action="/registrasi">
    <input type="hidden" name="_token" value="{{ csrf_token() }}>
    <p>
        <label for="nama">Nama</label>
        <input type="text" name="nama" value="{{ old('nama') }}>
    </p>
    <p>
        <label for="alamat">Alamat</label>
        <input type="text" name="alamat" value="{{ old('alamat') }}>
    </p>
    <p>
        <label for="nomor_ktp">Nomor KTP</label>
        <input type="text" name="nomor_ktp" value="{{ old('nomor_ktp') }}>
    </p>
    <p>
        <input type="submit" value="Daftar">
    </p>
</form>
```

Dan terakhir, method `registrasi` pada `HomeController`:

app/Http/HomeController.php

```
public function registrasi(Request $request)
{
    if ($request->get('nomor_ktp') == 123) {
        $request->flashOnly('nama', 'alamat');
        return redirect()->back();
    }
    return "Berhasil Registrasi!";
}
```

Setelah dibuat, silahkan kunjungi halaman /registrasi dan cobalah mengisi form dengan isian nomor_ktp berupa 123. Pastikan form tidak akan berhasil dikirim namun isian nama dan alamat masih ada disana. Sementara, jika menggunakan nomor_ktp selain 123 pastikan muncul tulisan "Berhasil Registrasi!".

Menerima Upload File

Dalam membangun website, tentunya kita sering bekerja dengan upload file. Mari kita pelajari bagaimana menerima upload file dengan Laravel.

Untuk menerima upload file, kita dapat menggunakan syntax `$request->file('nama_field');`.

Untuk mengecek apakah ada file yang diupload dengan nama field tertentu, kita dapat menggunakan syntax `$request->hasFile('nama_field');`.

Untuk memindahkan file yang diterima ke folder, kita dapat menggunakan syntax `$request->file('nama_field')->move('alamat_folder_tujuan', 'nama_file_baru (optional)');`.

Oh iya, setiap file yang kita terima merupakan instance dari [Symfony\Component\HttpFoundation\File\UploadedFile](#) yang berisi method yang akan memudahkan kita untuk mendapatkan detail dari file yang diterima misalnya:

- `getMimeType()`: Mendapatkan mime type dari file
- `getExtension()`: Mendapatkan ekstensi dari file
- `getClientSize()`: Mendapatkan ukuran file
- dll.

Mari kita demokan fitur upload ini untuk mengupload sebuah file image dan memindahkannya ke folder public/img.

Buatlah route seperti berikut:

¹⁵⁸ <http://api.symfony.com/2.5/Symfony/Component/HttpFoundation/File/UploadedFile.html>

app/Http/routes.php

```
Route::get('upload-form', function() {
    return View::make('upload_form');
});
Route::post('upload-profile-picture', 'HomeController@uploadProfilePicture');
```

Kemudian, formnya:

resources/views/upload_form.blade.php

```
<form method="POST" action="/upload-profile-picture" enctype="multipart/form-data">
<input type="hidden" name="_token" value="{{ csrf_token() }}">
<p>
    <label for="nama">Pilih photo</label>
    <input type="file" name="photo">
</p>
<p>
    <input type="submit" value="Upload">
</p>
</form>
```

Terakhir, kita buat method `uploadProfilePicture` pada `HomeController` untuk menerima upload:

app/Http/HomeController.php

```
public function uploadProfilePicture(Request $request)
{
    if (!$request->hasFile('photo')) {
        return "Tidak ada photo yang diupload";
    }

    $photo = $request->file('photo');
    $filename = str_random(6) . "." . $photo->getClientOriginalExtension();
    $path = public_path() . '/img';
    $photo->move($path, $filename);
    return "Berhasil upload " . $photo->getClientOriginalName() . " ke " .
        $path . " dengan nama file " . $filename;
}
```

Pada method ini kita mengecek apakah photo telah diupload. Jika foto sudah diupload, kita mengeset nama file secara acak dengan method `str_random(6)`. Tentunya masih menggunakan ekstensi file aslinya dengan method `getClientOriginalExtension()`.

Kemudian kita set tempat penyimpanan foto di

```
$path = public_path() . '/img';
```

Kita pindahkan dengan method `move()` kemudian kita tampilkan bahwa user telah berhasil menupload file berikut lokasi upload file diserver.

Jika sudah dibuat, silahkan dicoba fitur upload ini. Pastikan pesan muncul dan file berhasil muncul di folder `public/img`.

Bekerja dengan Cookie

Di Laravel kita dapat menambahkan cookie pada response dengan mudah dengan menggunakan method `withCookie` pada instance `Illuminate\Http\Response`. Untuk mendapatkan instance ini, kita dapat menggunakan beberapa cara. Salah satunya dengan menggunakan helper `response()`.

Misalnya kita akan membuat sebuah halaman yang akan menampilkan "halo" dan mengisi cookie dengan nama `api_key` yang valid selama 10 menit. Ini syntax yang akan kita buat:

app/Http/HomeController.php

```
public function generateCookie(Request $request)
{
    return response('halo')->withCookie(cookie('api_key', 's3cr3t', 10));
}
```

app/Http/routes.php

```
Route::get('generate-cookie', 'HomeController@generateCookie');
```

Untuk mengeceknya, silahkan kunjungi `/generate-cookie`. Pastikan outputnya seperti berikut:

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Sec...	Firs...
XSRF-TOKEN	eyJpdil6InRpQ1...	127.0.0.1	/	2015-11-26T01...	288			
api_key	eyJpdil6lk40Zm...	127.0.0.1	/	2015-11-25T23...	195	✓		
hblid	sLQqbn0lZ9jG...	127.0.0.1	/	2017-06-04T04...	37			
laravel_session	eyJpdil6lisxXC9...	127.0.0.1	/	2015-11-26T01...	293	✓		
olfsk	olfsk30800998...	127.0.0.1	/	2017-06-04T04...	27			

Membuat cookie

Terlihat disini, `value` dari cookie yang kita buat secara otomatis akan di-*encrypt* oleh Laravel.

Jika kita ingin meng-set cookie sambil menampilkan view, kita dapat mengubah syntax diatas menjadi:

app/Http/HomeController.php

```
return response(view('nama-view'))->withCookie(cookie('api_key', 's3cr3t', 10));
```

Kita juga dapat membuat sebuah cookie bertahan “selamanya” (tepatnya 5 tahun) dengan menggunakan syntax `forever` seperti berikut:

app/Http/HomeController.php

```
return response(view('nama-view'))->withCookie(cookie()->forever('api_key', 's3c\r3t'));
```

Untuk mengecek apakah sebuah request memiliki cookie, kita dapat menggunakan method `cookie()` pada instance `Illuminate\Http\Request`. Misalnya seperti berikut:

app/Http/HomeController.php

```
public function testCookie(Request $request)
{
    if ($request->cookie('api_key')) {
        return "Cookie api_key valid";
    } else {
        return "Cookie api_key tidak valid";
    }
}
```

Jika valid, outputnya akan seperti berikut:

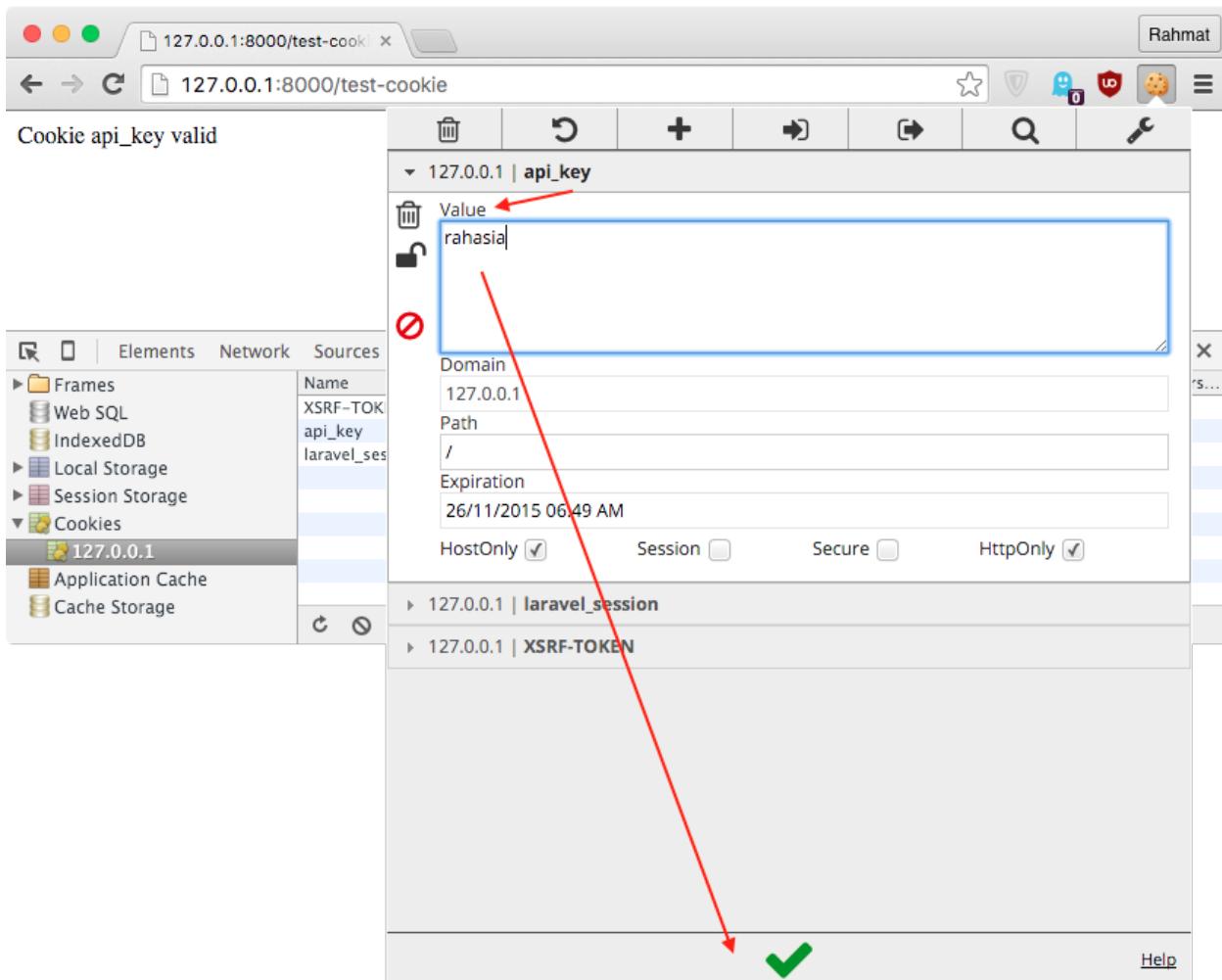


Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Sec...	Firs...
XSRF-TOKEN	eyJpdil6lkpmS0...	127.0.0.1	/	2015-11-26T01...	286	✓		
api_key	eyJpdil6lm50UD...	127.0.0.1	/	2015-11-25T23...	195	✓		
laravel_session	eyJpdil6lkdQV3I...	127.0.0.1	/	2015-11-26T01...	291	✓		

Cookie valid

Jika kita mengubah cookie, misalnya dengan menggunakan extensi [EditThisCookie](#)¹⁵⁹ di Chrome:

¹⁵⁹ <https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnplhplaeedifhccceomclgfbg/related?hl=en>



Mengubah cookie

Maka, cookie ini menjadi tidak valid:

The screenshot shows a browser window with the URL `127.0.0.1:8000/test-cookie`. The page content says "Cookie api_key tidak valid". Below the browser is the Chrome developer tools Resources panel. The left sidebar shows "Frames", "Web SQL", "IndexedDB", "Local Storage", "Session Storage", and "Cookies". Under "Cookies", there is an entry for "api_key" with the value "rahasia". The main table in the Resources panel lists the following cookies:

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Sec...	Firs...
XSRF-TOKEN	eyJpdil6litUQ28...	127.0.0.1	/	2015-11-26T01...	288			
api_key	rahasia	127.0.0.1	/	2015-11-25T23...	14	✓		
laravel_session	eyJpdil6lnBubO...	127.0.0.1	/	2015-11-26T01...	299	✓		

Cookie tidak valid

Hal yang sama juga akan terjadi jika kita menghapus cookie atau cookie tersebut telah expired.

Mengambil info dari Request

Sebagaimana dijelaskan di awal, di Laravel merupakan instance dari `Illuminate\Http\Request`¹⁶⁰ yang meng-*extend `Symfony\Component\HttpFoundation\Request`¹⁶¹ dari kedua class tersebut. Di awal, kita telah jelaskan beberapa manfaat dari class ini, mari kita bahas informasi lain seputar request yang bisa kita dapatkan dari class ini.

Misalnya kita memiliki URL `http://127.0.0.1:8000/post/memahami-request?source=fb` yang sedang diakses dengan method GET, kita dapat mendapatkan beberapa info berikut:

Method	Output
<code>method()</code>	GET
<code>root()</code>	<code>http://127.0.0.1:8000</code>
<code>url()</code>	<code>http://127.0.0.1:8000/post/memahami-laravel</code>
<code>fullUrl()</code>	<code>http://127.0.0.1:8000/post/memahami-laravel?source=fb</code>
<code>path()</code>	<code>post/memahami-laravel</code>
<code>is('post/*')</code>	TRUE

¹⁶⁰ <http://laravel.com/api/5.1/Illuminate/Http/Request.html>

¹⁶¹ <http://api.symfony.com/2.0/Symfony/Component/HttpFoundation/Request.html>

Masih banyak lagi yang lainnya, silahkan kunjungi API doc dari dua class diatas untuk melihatnya.

Oh iya, untuk mengetes semua method diatas, silahkan gunakan syntax berikut:

app/Http/routes.php

```
Route::get('/post/memahami-laravel', function () {
    $request = app('Illuminate\Http\Request');
    // isi disini
    return $request->method();
});
```

Memahami Response di Laravel

Ketika kita membuat response sebenarnya kita sedang membuat instance dari `Illuminate\Http\Response`. Memang, biasanya kita tidak membuat instance dari class ini secara manual, melainkan langsung dengan mengirim response berupa string atau view. Tentunya, hal ini bagus, karena akan mempermudah proses development. Tapi, ada kalanya kita harus membuat nya secara manual. Misalnya ketika kita ingin menambah custom header. Ini yang akan kita lakukan

app/Http/routes.php

```
Route::get('custom-header', function() {
    return (new Illuminate\Http\Response("Berhasil membuat custom header"))->header('Made-By', '@rahmatawaludin');
});
```

Jika kita mengetesnya dengan Postman, maka kita akan mendapatkan custom header `Made-By` yang telah kita buat:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History' selected, showing a single entry: 'GET 127.0.0.1:8000/custom-header'. The main area is the 'Builder' tab, where a GET request is being made to '127.0.0.1:8000/custom-header'. The 'Headers' tab is active, displaying seven custom headers: Cache-Control → no-cache, Connection → close, Content-Type → text/html; charset=UTF-8, Date → Thu, 26 Nov 2015 00:29:49 GMT, Host → 127.0.0.1:8000, Made-By → @rahmatawaludin, and X-Powered-By → PHP/5.6.10. The status bar at the bottom indicates a 200 OK status and a response time of 294 ms.

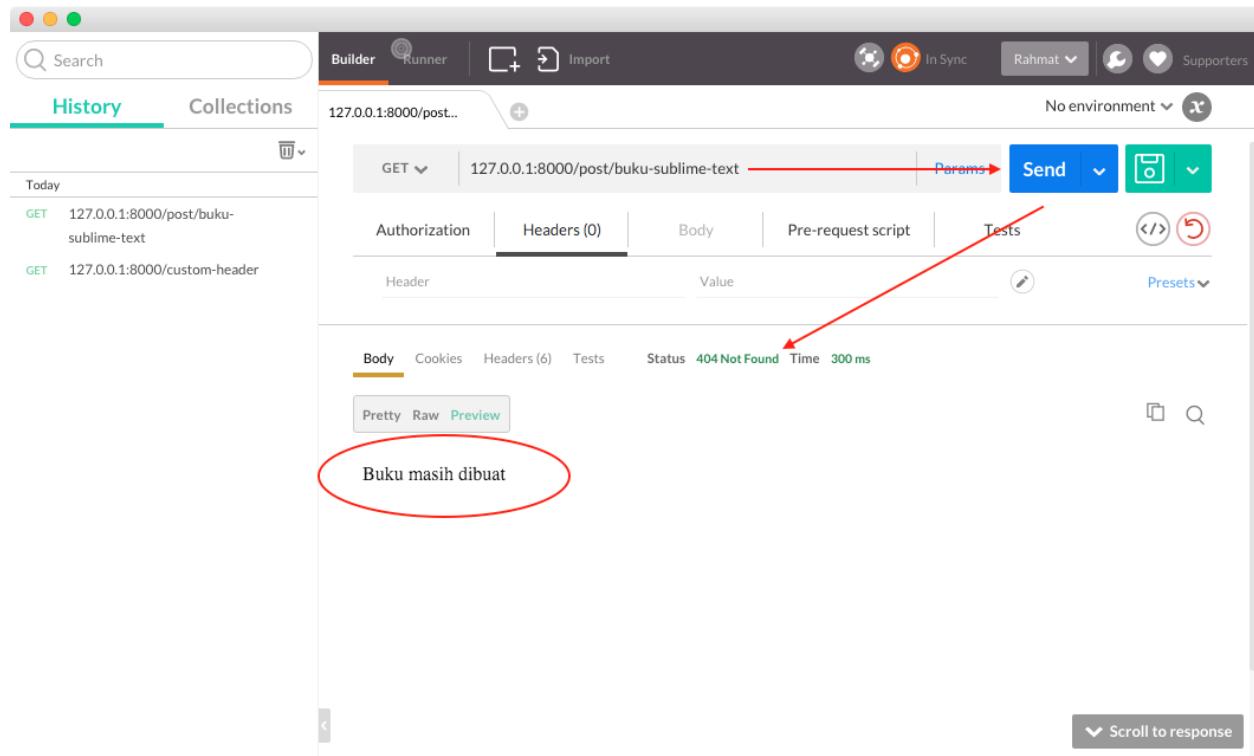
Custom header berhasil dibuat

Selain custom header, kita juga dapat mengubah status code dari response yang kita berikan, misalnya seperti berikut:

app/Http/routes.php

```
Route::get('post/buku-sublime-text', function() {
    return (new Illuminate\Http\Response("Buku masih dibuat", 404));
});
```

Maka output response yang akan kita dapatkan:



Membuat custom status code

Untuk memudahkan membuat custom response, Laravel juga menyediakan helper `response()` yang bisa kita gunakan untuk membuat response. Syntax diatas dapat kita ubah menjadi:

`app/Http/routes.php`

```
Route::get('post/buku-sublime-text', function() {
    return response("Buku masih dibuat", 404);
});
```

Membuat Response Redirect

Redirect halaman biasanya kita butuhkan jika kita sedang melakukan maintenance atau migrasi website. Untuk melakukan redirect di Laravel, kita dapat menggunakan helper `redirect()`. Misalnya, kita ingin mengarahkan url `/buku` ke `http://leanpub.com/u/rahmatawaludin` maka ini yang akan kita lakukan:

app/Http/routes.php

```
Route::get('buku', function() {
    return redirect('http://leanpub.com/u/rahmatawaludin');
});
```

Silahkan kunjungi url /buku, pastikan redirect berhasil dilakukan.

Ada beberapa variasi lain dari helper redirect ini, diantaranya:

- Redirect ke named route

```
redirect()->route('nama-route')[]
```

- Redirect ke route biasa

```
redirect('url-path');
```

- Redirect ke controller action

```
redirect()->action('NamaController@namaMethod');
```

- Redirect sambil passing variable ke session

```
redirect('path')->with('key', 'value');
```

Membuat Response JSON

Dalam membangun API, biasanya kita membutuhkan response berupa json. Di Laravel, kita dapat menggunakan `response()->json()` untuk membuatnya. Method ini akan secara otomatis membuat header Content-Type dengan isian `application/json`. Mari kita coba:

app/Http/routes.php

```
Route::get('daftar-buku', function() {
    return response()->json([
        ['judul' => 'Seminggu Belajar Laravel', 'url' => 'http://leanpub.com/seminggubelajarlaravel'],
        ['judul' => 'Menyelami Framework Laravel', 'url' => 'http://leanpub.com/bukularavel']
    ]);
});
```

Syntax diatas akan menampilkan json yang berisi buku yang telah saya tulis, jika kita coba akses:

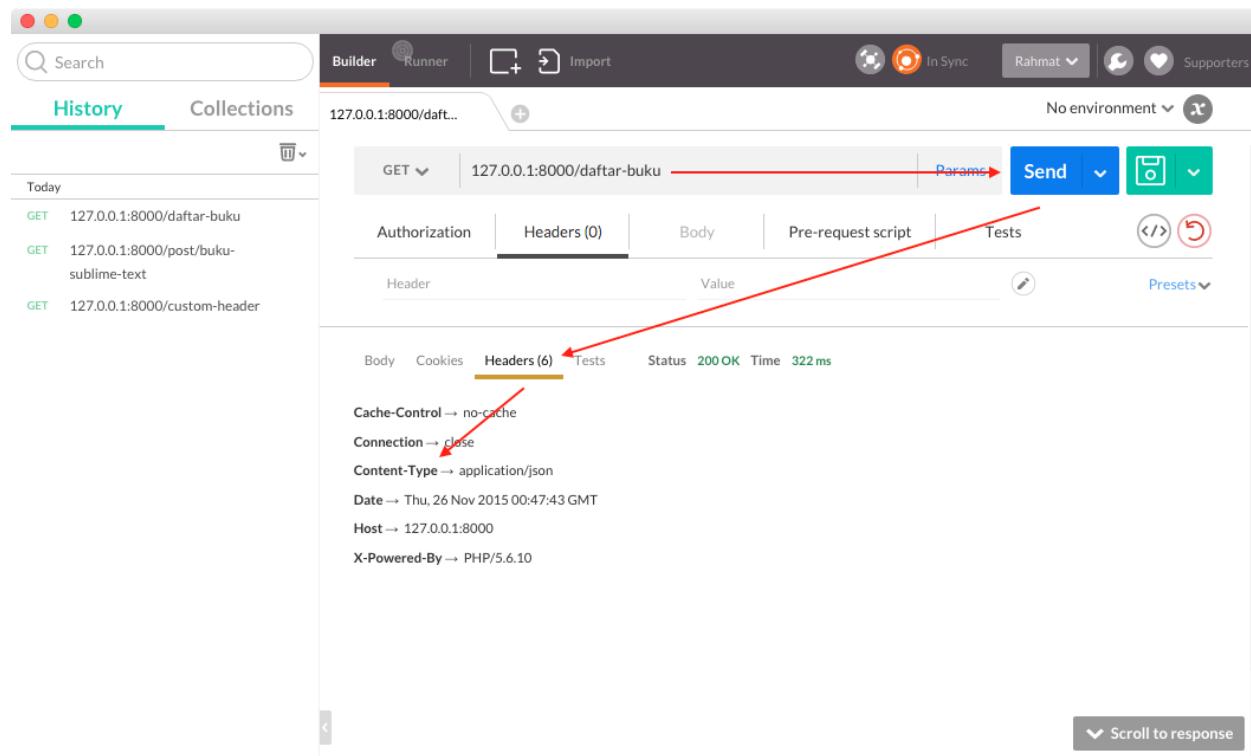
The screenshot shows the Postman application interface. In the top navigation bar, 'Builder' is selected. Below it, the URL '127.0.0.1:8000/daftar-buku' is entered. The 'Send' button is highlighted in blue. The 'Body' tab is selected, showing a JSON response with two items. The first item has a red arrow pointing to its line number (1). The JSON data is as follows:

```
1 [  
2 {  
3   "judul": "Seminggu Belajar Laravel",  
4   "url": "http://leanpub.com/seminggubelajarlaravel"  
5 },  
6 {  
7   "judul": "Menyelami Framework Laravel",  
8   "url": "http://leanpub.com/bukularavel"  
9 }  
10 ]
```

Below the JSON response, there is a 'Pretty' button, followed by 'Raw', 'Preview', and 'JSON' dropdown menus. The status bar at the bottom indicates 'Status 200 OK'.

Berhasil membuat response json

Headernya pun akan otomatis terisi:



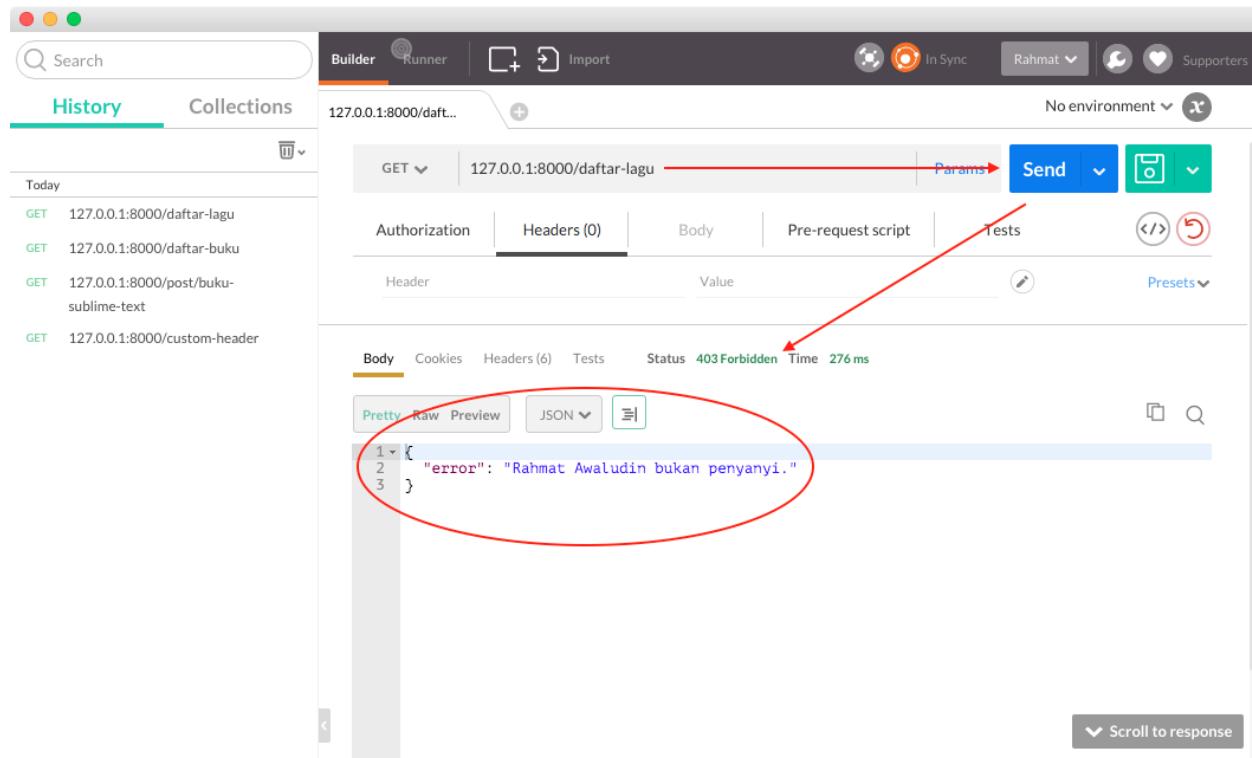
Header berhasil dibuat

Kita juga dapat menambahkan status code pada response JSON:

app/Http/routes.php

```
Route::get('daftar-lagu', function() {
    return response()->json(['error' => 'Rahmat Awaludin bukan penyanyi.'], 403);
});
```

Pada syntax diatas, selain JSON, kita mengirim status code 403.



Membuat custom status code pada response JSON

Tentunya, kita dapat menambahkan custom header dengan menggunakan method `header()` seperti kita bahas pada topik sebelumnya. Sip.

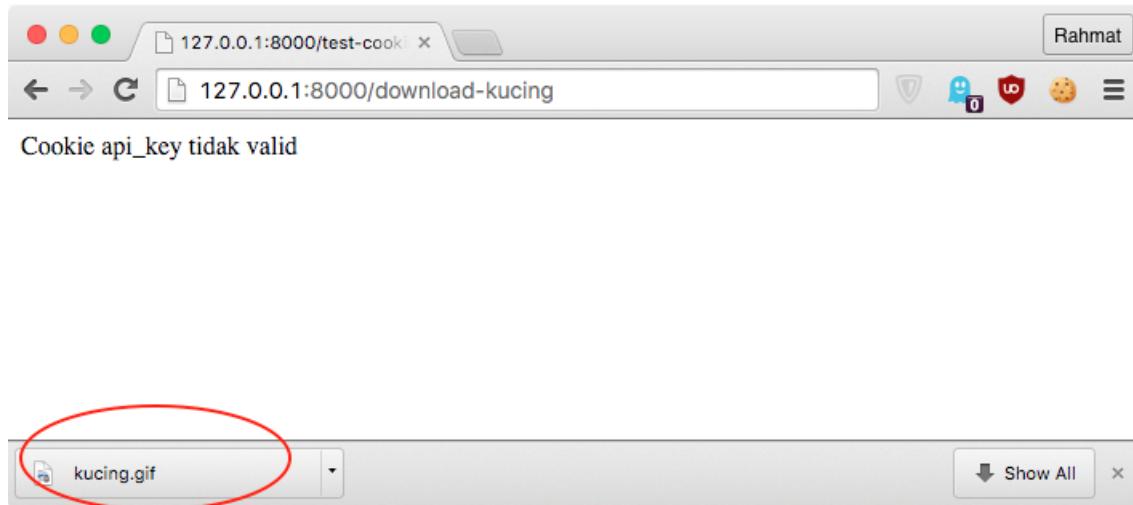
Membuat Response Download File

Kita dapat membuat fitur download file dengan mudah di Laravel. Caranya kita gunakan `syntax response()->download('alamat-file')` untuk memberikan response berupa file download. Misalnya kita akan menyimpan GIF kucing dari giphy.com di folder `public/img/kucing.gif` dan mendownloadnya dengan URL `download-kucing`. Ini syntax yang akan kita buat:

app/Http/routes.php

```
Route::get('download-kucing', function() {
    return response()->download(public_path() . '/img/kucing.gif');
});
```

Silahkan akses `/download-kucing`, pastikan Anda mendapatkan GIF kucing yang lucu.. :D



Berhasil membuat response download

Menggunakan Response Macro

Fitur ini sangat bermanfaat dalam membuat API. Biasanya, dalam membuat API kita perlu menyeragamkan setiap output. Misalnya, kita punya dalam membuat error not found, developer harus membuat response JSON dengan konten:

```
{  
    error_message: "Pesan error"  
}
```

Menggunakan cara manual, kita harus mengetik response yang cukup panjang. Misalnya seperti ini:

app/Http/routes.php

```
Route::get('buku/menguasai-go-lang', function() {  
    return response(json_encode(['error_message' => 'Buku Go belum ditulis.']), \  
404)->header('Content-Type', 'application/json');  
});
```

Menggunakan Response Macro, kita bisa membuat syntax diatas menjadi seperti ini:

```
return response()->jsonNotFound('Buku Go belum ditulis.');
```

Cara membuatnya, kita harus me-*register* macro pada method `boot` di Service Provider misalnya pada `AppServiceProvider`:

app/Providers/AppServiceProvider.php

```
<?php

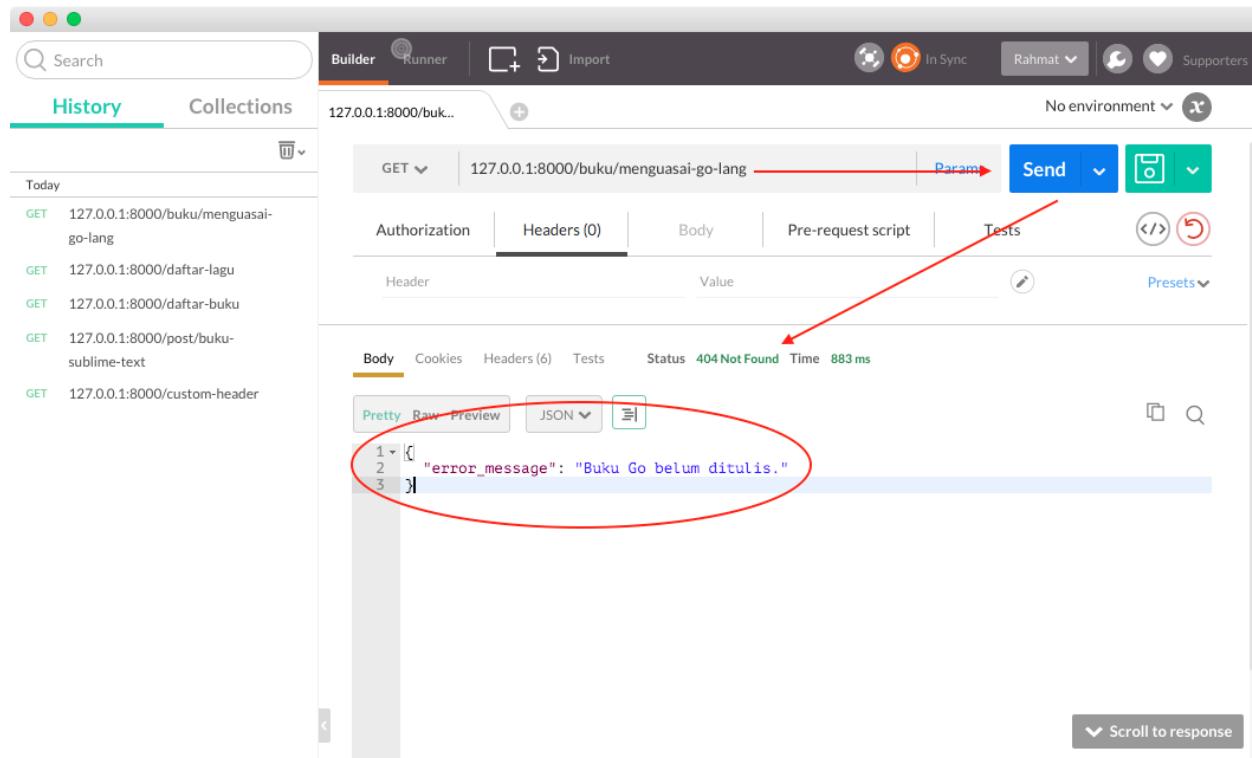
namespace App\Providers;

use Illuminate\Contracts\Routing\ResponseFactory;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot(ResponseFactory $factory)
    {
        $factory->macro('jsonNotFound', function ($value) use ($factory) {
            return $factory->make(json_encode(['error_message' => $value]), 404)
                ->header('Content-Type', 'application/json');
        });
    }
}
```

Syntax ini hampir sama dengan menggunakan cara manual, hanya saja kita awali dengan `$factory->make()`. Variable `$factory` merupakan instance dari `Illuminate\Contracts\Routing\ResponseFactory`. Disini kita menamai macro dengan `jsonNotFound`.

Jika kita coba akses `/buku/menguasai-go-lang`, maka kita akan mendapatkan output:



Berhasil membuat response macro

Sip.



Source code dari latihan di baba ini bisa didapat di [Bitbucket](#)¹⁶²



Ringkasan

Pada bab ini kita telah mempelajari bagaimana bekerja dengan request dan response di Laravel. Menguasai bagaimana membuat response yang tepat tentunya penting, tapi bagaimana kita memberikan pembatasan akses untuk resource?

Jawabannya, dengan middleware yang akan kita pelajari di bab selanjutnya. Semangat! :)

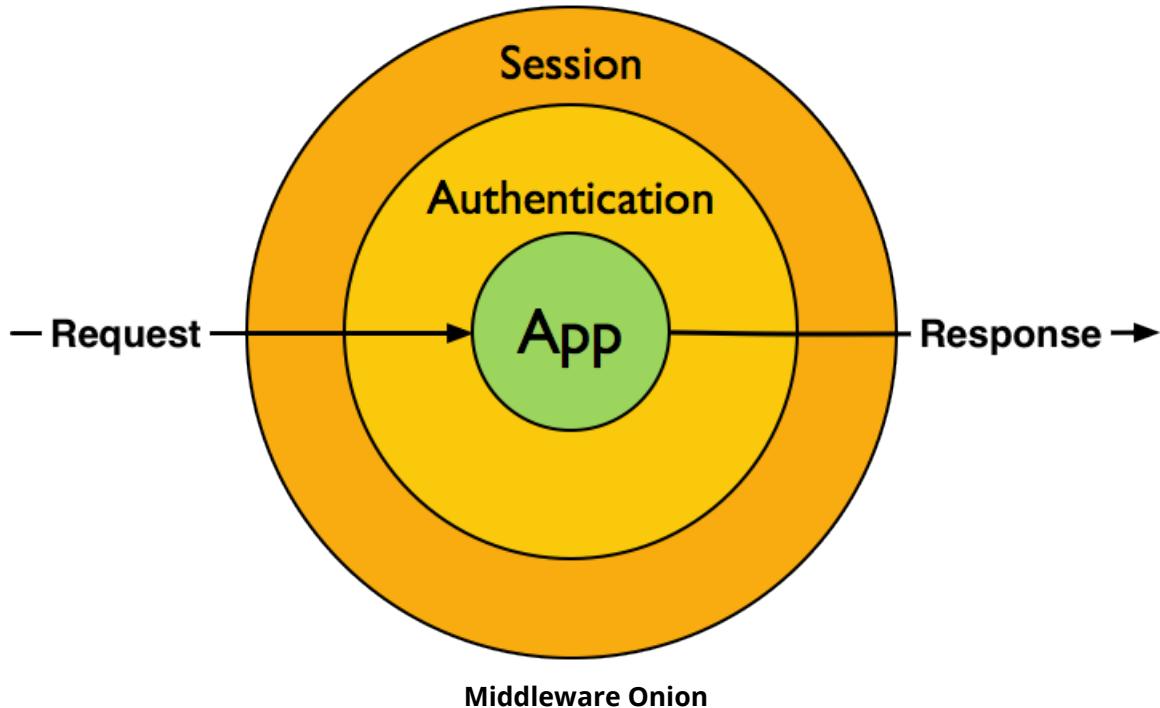
¹⁶²<https://bitbucket.org/rahmatawaludin/sample-request-response/commits/all>

Middleware

Dalam membuat aplikasi sering kali kita butuh untuk melakukan action sebelum request sampai ke aplikasi kita. Contoh paling sederhana adalah tentang authentikasi. Dalam authentikasi, yang akan kita bahas lebih lengkap di bab 13, kita mengecek apakah user memiliki session aktif sebelum mengizinkannya untuk mengakses resource yang dimiliki aplikasi kita.

Itu salah satu contoh penggunaan middleware. Contoh lain adalah ketika kita menginginkan melakukan sebuah *action* ketika response telah kita berikan kepada user. Misalnya dengan melakukan log. Nah, itu juga contoh penggunaan middleware. Yang kedua ini, di Laravel dikenal dengan istilah *after middleware*, karena dia berjalan setelah response diberikan. Kalau yang pertama, biasa dikenal dengan *before middleware*, karena dia berjalan ketika response belum diberikan.

Untuk memahami bagaimana middleware bekerja, coba perhatikan diagram berikut:



Gambar diatas berasal dari [StackPHP¹⁶³](#), sebuah website rujukan untuk penggunaan middleware di PHP. Jadi, middleware itu ibarat lapisan yang berada di sekeliling aplikasi kita. Disini, buletan yang hijau itu *core* aplikasi kita. Nah, tanda panah yang

¹⁶³<http://stackphp.com>

dari kiri itu request dari user. Ketika request masuk, dia bisa melewati beberapa lapis middleware. Kalau disini contohnya Session dan Authentication.

Kalau panah yang dari app ke kanan itu menandakan response yang diberikan app kita ke user. Terlihat disini, ketika keluarpun, kita bisa memberikan middleware.

Kalau mau lebih detail lagi, middleware itu salah satu implementasi dari Decorator Pattern. Sederhananya, dia akan memodifikasi request maupun response dari aplikasi kita. Kalau mau lebih lanjut belajar tentang Decorator Pattern, baca artikel ini¹⁶⁴ (atau mau saya tulis jadi buku juga? Hehe.. :D)

Sip, cukup teorinya, kita coding ya..

Membuat Middleware

Mari kita buat middleware yang akan membatasi akses user ke apliksi kita jika dia tidak mengirimkan query paramter atau post data dengan key `api_token` dengan nilai `indonesia-hebat`. Jika, parameter ini tidak diset atau nilainya salah, maka kita akan memberikan error 403 dengan pesan yang sesuai. Ikuti langkah ini:

1. Jalankan `php artisan make:middleware ApiToken`. Perintah ini akan membuat file `/app/Http/Middleware/ApiToken.php`.
2. Isi file yang digenerate dengan code berikut:

/app/Http/Middleware/ApiToken.php

```
<?php
namespace App\Http\Middleware;
use Closure;
class ApiToken
{
    public function handle($request, Closure $next)
    {
        if ($request->has('api_token')) {
            if ($request->get('api_token') == 'indonesia-hebat') {
                return $next($request);
            }
            return response('API Token salah.', 403);
        }
        return response('API Token tidak diisi.', 403);
    }
}
```

¹⁶⁴<http://code.tutsplus.com/tutorials/design-patterns-the-decorator-pattern--cms-22641>

Disini, kita menggunakan method `handle()` untuk menambahkan logic dari middleware kita. Method ini akan selalu menerima dua parameter, `$request` yang merupakan instance dari `\Illuminate\Http\Request` dan `$next` berupa Closure yang kita gunakan untuk menandakan bahwa request berlanjut ke lapisan selanjutnya (lihat lagi gambar StackPHP).

Kita juga menggunakan helper `response()` untuk memudahkan memberikan response error dengan status code dan message yang sesuai. Di tentunya, kita juga bisa mengarahkannya ke view jika dibutuhkan.

Kita akan pelajari lebih lanjut mengenai apa saja yang bisa kita lakukan pada method `handle()` ini pada pembahasan selanjutnya.

3. Tambahkan baris ini pada `/app/Http/Kernel.php`:

`/app/Http/Kernel.php`

```
<?php  
namespace App\Http;  
use Illuminate\Foundation\Http\Kernel as HttpKernel;  
class Kernel extends HttpKernel  
{  
    ...  
    protected $routeMiddleware = [  
        ...  
        'api-token' => \App\Http\Middleware\ApiToken::class,  
    ];  
}
```

Baris ini, menandakan kita akan menggunakan middleware ini sebagai middleware di route atau controller dengan nama `api-token`.

Setelah selesai dengan 3 langkah diatas, kita telah berhasil membuat middleware. Untuk mengecek middleware ini, mari kita buat route `/api-test` yang akan menggunakan middleware yang telah kita buat ini. Buatlah route seperti berikut:

`app/Http/routes.php`

```
Route::get('api-test', ['middleware' => 'api-token', 'uses' => 'HomeController@apiTest']);
```

Pada route diatas, kita menggunakan method `apiTest` pada `HomeController`. Jika controller tersebut belum ada, silahkan buat dengan perintah:

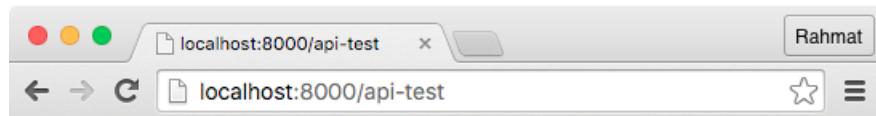
```
$ php artisan make:controller HomeController --plain
```

Dan tambahkan code berikut:

app/Http/Controllers/HomeController.php

```
<?php  
....  
class HomeController extends Controller  
{  
    public function apiTest()  
    {  
        return "Berhasil mengakses API";  
    }  
}
```

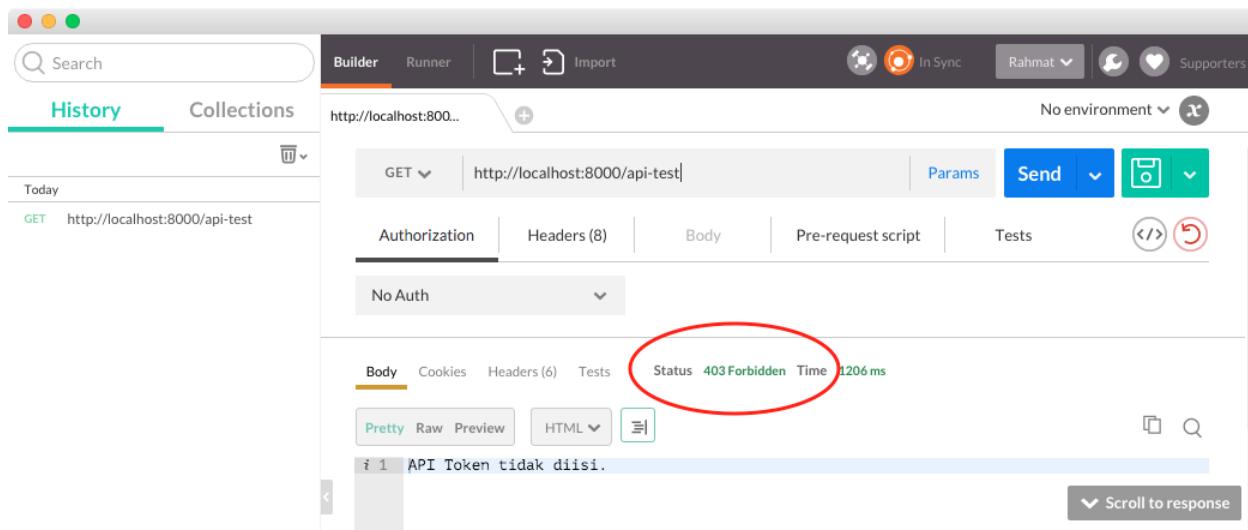
Oke, sekarang coba akses url /api-test, hasilnya akan muncul pesan gagal seperti berikut:



API Token tidak diisi.

Middleware berhasil menggagalkan request

Jika dicek dengan Postman, kitapun berhasil mendapatkan status code 403:



Mengecek status code dengan Postman

Kini, kita akses url tersebut dengan API token, sehingga urlnya akan menjadi /api-test?api_token=indonesia-hebat:



Berhasil mengakses route

Tapi, bila kita mengakses dengan API Token yang salah, maka tetap akan muncul pesan gagal:



API Token salah

Nah, jadi begitu cara bikin middleware. Lebih tepatnya, yang kita buat diatas adalah *route middleware*. Secara singkat langkahnya:

1. Generate middleware.
2. Ubah method handle dengan logic yang kita butuhkan.
3. Register middleware di `app/Http/kernel.php`.
4. Gunakan middleware pada route yang diinginkan.

After Middleware

Di contoh pertama, kita telah berhasil membuat *before middleware* yang berjalan ketika user hendak mengakses apliksi. Jika kita ingin membuat middleware yang berjalan setelah user mendapatkan response dari aplikasi, kita perlu menggunakan *after middleware*.

Nah, di dalam membuat aplikasi biasanya kita membutuhkan fitur logging. Manfaatnya sangat beragama, dari mengetahui jumlah total halaman diakses, ip yang mengakses, dll. Biasanya kita dapat melakukan hal itu dengan frontend side analytics, misalnya Google Analytics. Tapi, user sekarang semakin cerdas, terkadang Google Analytics kita dinonaktifkan misalnya dengan extensi chrome bernama [Ghostery¹⁶⁵](#).

Untuk mengatasinya, sambil belajar tentang *after middleware*, mari kita buat sebuah middleware yang akan mencatat kapan akses ke sebuah route terjadi ke database berikut ip yang mengaksesnya. Ikuti langkah berikut:

1. Buatlah table untuk mencatat log dengan membuat migration berikut:

¹⁶⁵ <https://chrome.google.com/webstore/detail/ghostery/mlomiejdfkolichcfejclcbmpeanij?hl=en>

```
$ php artisan make:migration create_access_logs_table --create=access_logs
```

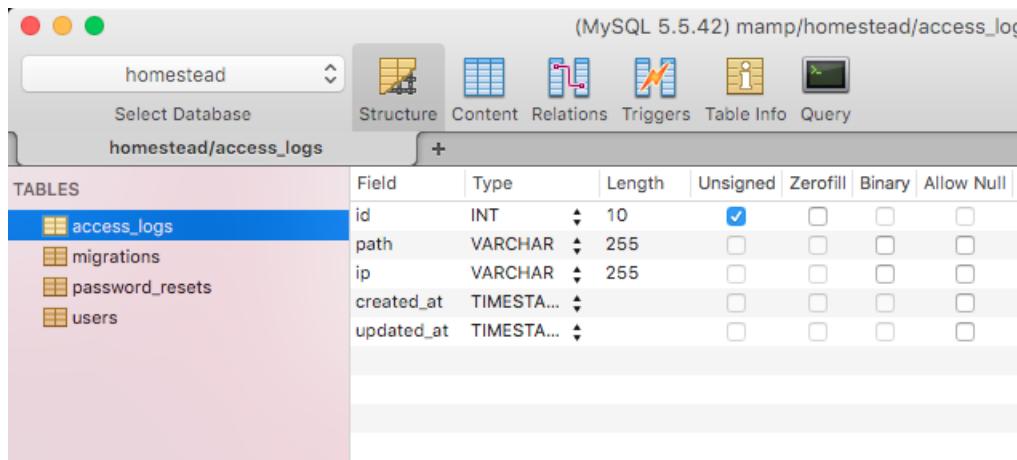
2. Kita akan menggunakan struktur table seperti berikut:

/database/migrations/xxxx_xx_xx_xxxxxxx_create_access_logs_table.php

```
<?php
...
class CreateAccessLogsTable extends Migration
{
    public function up()
    {
        Schema::create('access_logs', function (Blueprint $table) {
            $table->increments('id');
            $table->string('path');
            $table->string('ip');
            $table->timestamps();
        });
    }
}
```

Field `path` akan kita gunakan untuk mencatat url yang diakses. `ip` untuk mencatat ip dari client. Sementara `$table->timestamps()` akan kita gunakan untuk mencatat waktu akses (field `created_at`).

Jalankan `php artisan migrate` untuk menambahkan table `access_logs` ke database. Pastikan di database ada table seperti berikut:



The screenshot shows the MySQL Workbench interface with the database 'homestead' selected. The 'Structure' tab is active, displaying the schema for the 'access_logs' table. The table has the following columns:

Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null
id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
path	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ip	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
created_at	TIMESTAMPE		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
updated_at	TIMESTAMPE		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Table access_logs

3. Mari kita buat middleware `access-logs`. Jalankan `php artisan make:middleware AccessLog`. Setelah di-`generate`, kita isi method `handle` dengan isian berikut:

/app/Http/Middleware/AccessLog.php

```
<?php

namespace App\Http\Middleware;

use Closure;
use DB;
use DateTime;

class AccessLog
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);
        // buat log
        DB::table('access_logs')->insert([
            'path' => $request->path(),
            'ip' => $request->getClientIp(),
            'created_at' => new DateTime,
            'updated_at' => new DateTime
        ]);
        return $response;
    }
}
```

Nah, teknik untuk membuat after middleware ada disini. Di baris awal method `handle()`, kita menggunakan syntax `$response = $next($request)` untuk menyimpan *instance* dari response. Di akhirnya, kita gunakan syntax `return $response` untuk mengirimkan response yang telah kita simpan.

Nah, ditengah kedua syntax itulah kita mengisikan *after middleware* yang kita inginkan. Kalau disini, menggunakan query builder untuk membuat record baru di table `access_logs`. Sebagaimana yang kita pelajari di Bab Request dan Response, kita dapat mengambil banyak informasi dari request. Disini, kita menggunakan `$request->path()` untuk mendapatkan url yang sedang diakses dan `$request->getClientIp()` untuk mendapatkan ip dari user yang sedang mengakses.

4. Setelah dibuat, kita register middleware tersebut:

/app/Http/Kernel.php

```
<?php  
....  
class Kernel extends HttpKernel  
{  
....  
protected $routeMiddleware = [  
....  
    'access-log' => \App\Http\Middleware\AccessLog::class,  
];  
}
```

5. Untuk mengecek middleware ini, mari kita buat route /log-test yang akan menampilkan berapa kali dia telah diakses.

/app/Http/routes.php

```
Route::get('log-test', ['middleware' => 'access-log', 'uses' => 'HomeController@logTest']);
```

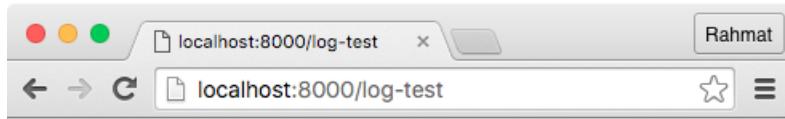
6. Kita buat method controller nya:

/app/Http/Controllers/HomeController.php

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
use App\Http\Requests;  
use App\Http\Controllers\Controller;  
use DB;  
  
class HomeController extends Controller  
{  
....  
public function logTest(Request $request)  
{  
    $total_access = DB::table('access_logs')->where('path', $request->path())->count();  
    return "Halaman ini telah diakses sebanyak " . $total_access . " kali.";  
}  
}
```

Disini, kita menggunakan query builder untuk mengambil total record dengan path yang sesuai dengan url yang sedang kita akses.

Sip, sudah selesai. Kini kita cek dengan mengunjungi url /log-test beberapa kali. Pastikan nilainya selalu bertambah setiap kali kita me-refresh halaman.



Halaman ini telah diakses sebanyak 6 kali.

Middleware acces logs berjalan

Untuk membuktikan bahwa middleware ini hanya berjalan ketika response telah diberikan, kita dapat menambahkan middleware `api-token` ke route ini. Sehingga, ketika kita tidak mengirimkan `api_token` yang benar, seharusnya middleware ini tidak dieksekusi. Caranya, ubah route menjadi seperti berikut:

app/Http/routes.php

```
Route::get('log-test', ['middleware' => ['access-log', 'api-token'], 'uses' => '\
HomeController@logTest']);
```

Silahkan dicek. Kini, jika kita tidak melewati middleware `api-token`, maka middleware `access-log` tidak akan pernah dieksekusi.



Penggunaan Database untuk Logging

Di *real world application*, biasanya kita lebih memilih untuk menggunakan NoSQL database untuk menyimpan data log. Hal ini dilakukan karena NoSQL database misalnya redis dan mongodb dapat diakses lebih cepat.

Register

Ada saatnya kita butuh menggunakan middleware untuk beberapa route, misalnya pada contoh sebelumnya kita ingin menggunakan middleware `access-logs` untuk semua route. Ada beberapa teknik yang bisa kita lakukan diantaranya:

1. Menggunakan Route group

Teknik ini lebih cocok jika middleware dibutuhkan oleh beberapa route tapi tidak semuanya. Menggunakan teknik ini, file route kita akan terlihat seperti berikut:

app/Http/routes.php

```
Route::group(['middleware' => 'access-log'], function() {
    Route::get('/', function () {
        return view('welcome');
    });

    Route::get('api-test', ['middleware' => 'api-token', 'uses' => 'HomeController@apiTest']);
    Route::get('log-test', 'HomeController@logTest');
});
```

2. Menggunakan Global Middleware

Penggunaan Global Middleware pada Laravel 5.2 sudah diganti dengan middleware group. Cek penjelasan pada pembahasan tentang middleware group. Teknik ini sangat cocok jika hendak menggunakan middleware pada SEMUA route. Caranya, dengan menambahkan middleware pada array \$middleware di /app/Http/Kernel.php. Sehingga syntax pada class tersebut akan seperti berikut:

/app/Http/Kernel.php

```
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    protected $middleware = [
        ...
        \App\Http\Middleware\AccessLog::class
    ];

    protected $routeMiddleware = [
```

```
....  
'api-token' => \App\Http\Middleware\ApiToken::class,  
'access-log' => \App\Http\Middleware\AccessLog::class,  
];  
}
```

Laravel, secara default menggunakan teknik ini untuk beberapa fiturnya. Pada saat buku ini ditulis, isian default untuk array `$middleware` adalah seperti berikut:

```
protected $middleware = [  
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,  
    \App\Http\Middleware\EncryptCookies::class,  
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,  
    \Illuminate\Session\Middleware\StartSession::class,  
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,  
    \App\Http\Middleware\VerifyCsrfToken::class,  
];
```

Terminable

Fitur ini sangat bermanfaat untuk menjalankan sebuah *action* ketika response telah diberikan ke user. Sekilas terlihat mirip dengan *after middleware*. Bedanya, menggunakan *after middleware*, action di middleware kita tidak akan dieksekusi jika ada *before middleware* yang gagal. Menggunakan *Terminable*, action pada middleware akan tetap dijalankan ketika response telah diberikan meskipun ada *before middleware* yang gagal.

Contohnya, pada sample middleware *access-log* yang kita buat, kita hanya mencatat log jika route berhasil diakses. Mari kita modifikasi agar dia juga mencatat ketika request gagal. Ikuti langkah berikut:

1. Kita perlu menambah field `status` pada table `access_logs` untuk mencatat log yang sukses dan gagal. Ubahlah method `up` pada file migration nya menjadi seperti berikut.

/database/migrations/xxxx_xx_xx_xxxxxxx_create_access_logs_table.php

```
public function up()
{
    Schema::create('access_logs', function (Blueprint $table) {
        $table->increments('id');
        $table->string('path');
        $table->string('ip');
        $table->string('status');
        $table->timestamps();
    });
}
```

Selanjutnya, jalankan `php artisan migrate:refresh` untuk me-refresh struktur table `access_logs`.

2. Middleware `access-logs` akan kita ubah seperti berikut:

/Users/rahmatawaludin/Code/sample-middleware/app/Http/Middleware/Access-Log.php

```
<?php
...
class AccessLog
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // buat log
        DB::table('access_logs')->insert([
            'path' => $request->path(),
            'ip' => $request->getClientIp(),
            'status' => 'success',
            'created_at' => new DateTime,
            'updated_at' => new DateTime
        ]);

        // tambah custom field
        $request->merge(['isLogged' => 1]);

        return $response;
    }

    public function terminate($request, $response)
}
```

```

    {
        if (!$request->has('isLoggedIn')) {
            // buat log
            DB::table('access_logs')->insert([
                'path' => $request->path(),
                'ip' => $request->getClientIp(),
                'status' => 'failed',
                'created_at' => new DateTime,
                'updated_at' => new DateTime
            ]);
        }
    }
}

```

Oke, syntax ini cukup panjang. Kita bahas satu-persatu ya.

Yang perlu diperhatikan ketika kita menggunakan Terminable adalah *action* yang akan kita jalankan ditulis pada method `terminate()`. Disini, kita membuat record baru untuk table `access_logs` tapi dengan field `status` diisi `failed`.

Ketika semua *before middleware* sukses, tentunya dia akan melewati method `handle`. Itulah sebabnya kita menggunakan syntax `$request->merge(['isLoggedIn' => 1])` untuk menambah field `isLoggedIn` pada instance `$request`.

Field ini yang akan kita gunakan di method `terminate` untuk mengecek apakah request ini sudah di log atau belum. Terlihat pada syntax `if (!$request->has('isLoggedIn'))` pada method `terminate()`. Ini perlu kita lakukan karena method `terminate` akan dieksekusi pada saat semua *before middleware* berhasil maupun ada yang gagal.

Hasil akhirnya, kita hanya akan membuat satu log untuk request yang sukses dan gagal.

- Untuk mendemokan fitur ini, mari kita ubah method `logTest` pada `HomeController`:

/app/Http/Controllers/HomeController.php

```

public function logTest(Request $request)
{
    $total_success = DB::table('access_logs')
        ->where('path', $request->path())
        ->where('status', 'success')
        ->count();
    $total_failed = DB::table('access_logs')
        ->where('path', $request->path())
        ->where('status', 'failed')
        ->count();
    return "Halaman ini telah diakses sebanyak $total_success kali sukses dan $\

```

```
total_failed kali gagal.";  
}
```

Di syntax ini, kita melakukan dua query untuk mengambil total log akses yang berhasil dan gagal, kemudian menampilkannya.

4. Terakhir, agar beberapa request gagal, kita akan menggunakan middleware `api-test` pada route:

app/Http/routes.php

```
Route::get('log-test', ['middleware' => ['api-token', 'access-log'], 'uses' => \  
'HomeController@logTest']);
```

Oh iya, disini saya tidak mengaktifkan global middleware. Jika Anda sudah mengaktifkannya, silahkan di-nonaktifkan lagi.

Oke, mari kita test. Silahkan kunjungi `/log-test` sebanyak 4 kali, dan `/log-test?api_token=indonesia-hebat` sebanyak 5 kali. Pastikan outputnya seperti berikut:



Halaman ini telah diakses sebanyak 4 kali sukses dan 3 kali gagal.

Terminable middleware berhasil dibuat

Mungkin ada sedikit kejanggalan disini, kenapa yang sukses hanya 4. Ini terjadi karena penambahan record untuk sukses dilakukan **setelah** response diberikan. Makanya nilainya masih 4. Sip.

Middleware Parameters

Dalam membuat middleware, seringkali kita membutuhkan logic yang sedikit berubah dalam method handlenya. Contoh sederhana dalam penggunaan role dalam sebuah aplikasi. Kita ingin setiap user memiliki role, tapi jenis role yang dibutuhkan oleh seorang user akan berbeda tergantung *resource* yang coba diakses. Halaman dashboard

mungkin dapat diakses semua role, tapi halaman management user hanya bisa diakses oleh role admin.

Pada bagian ini, kita akan membuat contoh yang agak mirip dengan case diatas. Karena kita belum belajar tentang authentikasi, mari kita kembangkan middleware `api-token` yang telah kita buat agar menerima parameter.

Kita akan kembangkan middleware `api-token` ini agar menggunakan record di database untuk menentukan nilai token yang valid. Selain itu, untuk setiap record token, kita akan menambahkan field `access` yang menentukan jenis akses yang diizinkan. Yap, hampir mirip dengan penggunaan role pada contoh kasus diatas.

Ikuti langkah berikut untuk membuatnya.

1. Kita butuh table `tokens` yang akan menyimpan data token dan jenis akses. Buatlah dengan menjalankan perintah `php artisan make:migration create_tokens_table --create=tokens`. Ubahlah isian method `up()` pada file migration seperti berikut:

/database/migrations/xxxx_xx_xx_xxxxxxx_create_tokens_table.php

```
public function up()
{
    Schema::create('tokens', function (Blueprint $table) {
        $table->increments('id');
        $table->string('key');
        $table->string('access');
    });
}
```

Disini, field `key` akan menyimpan semua token yang valid. Sementara, field `access` akan menyimpan jenis akses yang diizinkan untuk token tersebut.

2. Kita akan membuat sample token yang valid dengan menggunakan seeder, tentunya kalau mau manual juga boleh.

/database/seeds/DatabaseSeeder.php

```
<?php
...
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        Model::unguard();

        DB::table('tokens')->insert([
            [
                'key' => '1234567890',
                'access' => 'read'
            ],
            [
                'key' => '9876543210',
                'access' => 'write'
            ]
        ]);
    }
}
```

```

        'key' => 'indonesia-hebat',
        'access' => 'admin',
    ]);

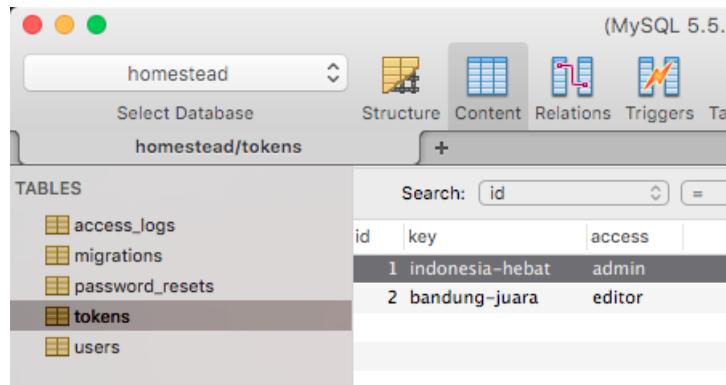
    DB::table('tokens')->insert([
        'key' => 'bandung-juara',
        'access' => 'editor'
    ]);

    Model::reguard();
}
}

```

Ada dua sample token yang kita buat, token `indonesia-hebat` dengan akses `admin` dan token `bandung-juara` dengan akses `editor`.

Jalankan `php artisan migrate:refresh --seed` untuk menambah table `tokens` dan mengisi sample data. Sehingga database akan terlihat seperti berikut:



The screenshot shows the MySQL Workbench interface with the database 'homestead' selected. The 'Content' tab is active, displaying the 'tokens' table. The table has three columns: id, key, and access. There are two rows of data: one with id 1, key 'indonesia-hebat', and access 'admin'; and another with id 2, key 'bandung-juara', and access 'editor'. The table structure is shown on the left.

	id	key	access
1	1	indonesia-hebat	admin
2	2	bandung-juara	editor

Table tokens

- Untuk menggunakan middleware parameter, kita perlu merubah method `handle()` pada middleware `api-token`:

```

public function handle($request, Closure $next, $access)
{
    if ($request->has('api_token')) {
        $valid_token = \DB::table('tokens')
            ->where('key', $request->get('api_token'))
            ->where('access', $access)
            ->get();

        if ($valid_token) {

```

```
        return $next($request);
    }

    return response('API Token salah.', 403);
}
return response('API Token tidak diisi.', 403);
}
```

Perubahan yang kita lakukan disini terlihat dari definisi methodnya. Awalnya hanya `handle($request, Closure $next)` kini menjadi `handle($request, Closure $next, $access)`. Isian `$access` menandakan parameter yang kita berikan.

Pada method ini, kita juga merubah logic setelah mengecek apakah field `api-token` terisi. Awalnya kita hanya mengecek apakah field tersebut sama dengan `indonesia-hebat`. Kini, kita juga mengecek apakah terdapat token dengan key yang dikirim yang memiliki `access` yang sesuai dengan parameter.

Oke, sampai disini perubahan yang kita lakukan. Pada tahapan register, tidak ada yang perlu kita ubah. Sekarang mari kita demokan dengan membuat route yang hanya bisa diakses dengan API dengan access `admin` dan `editor`. Tambahkan route berikut:

app/Http/routes.php

```
Route::get('admin-test', ['middleware' => 'api-token:admin', 'uses' => 'HomeController@adminTest']);
Route::get('editor-test', ['middleware' => 'api-token:editor', 'uses' => 'HomeController@editorTest']);
```

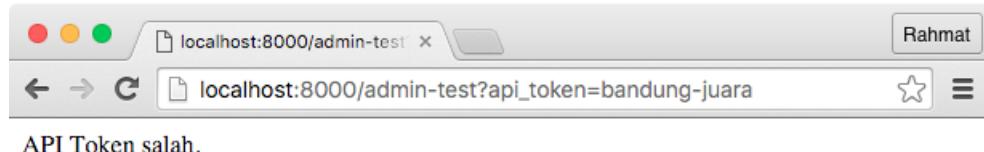
Disini, route `admin-test` hanya bisa diakses token dengan akses `admin`. Cirinya, kita menggunakan `api-token:admin` untuk menulis middleware pada route tersebut. Hal yang sama kita lakukan pada route `editor-test` yang hanya bisa diakses oleh token dengan akses `editor`.

Kita tambahkan kedua method untuk route tersebut pada `HomeController`:

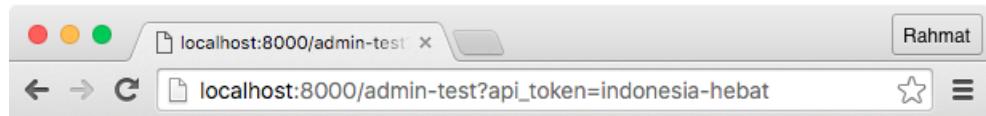
/app/Http/Controllers/HomeController.php

```
<?php  
....  
class HomeController extends Controller  
{  
....  
    public function adminTest()  
    {  
        return "Berhasil mengakses resource admin.";  
    }  
  
    public function editorTest()  
    {  
        return "Berhasil mengakses resource editor.";  
    }  
}
```

Kini, ketika kita mencoba mengakses route /admin-test dengan token bandung-juara, request kita akan gagal:

**API Token tidak valid**

Ini terjadi karena token bandung-juara hanya memiliki akses editor. Sementara, route ini membutuhkan akses admin. Tapi, jika kita menggunakan token indonesia-hebat, maka request akan berhasil:



Berhasil mengakses resource admin.

API Token valid

Silahkan coba test pada url /editor-test. Pastikan semua logicnya berjalan.

Route Middleware

Mari kita bahas lebih dalam tentang penggunaan middleware di route. Pada pembahasan sebelumnya, telah kita pelajari beberapa hal tentang route middleware:

1. Untuk membuat middleware yang bisa digunakan oleh route, kita perlu menambahkan middleware di array `$routeMiddleware` pada `/app/Http/Kernel.php` dengan alias tertentu:

/app/Http/Kernel.php

```
<?php  
....  
class Kernel extends HttpKernel  
{  
    ....  
    protected $routeMiddleware = [  
        ....  
        'alias' => class_middleware,  
    ];  
}
```

2. Untuk menggunakan middleware, kita menggunakan array dengan key `middleware` pada route:

```
Route::get('url', ['middleware'=>'nama-middleware', 'uses'=>'Controller@method'\']);
```

3. Kita juga bisa menggunakan beberapa middleware dalam satu route:

```
Route::get('url', ['middleware'=>['nama-middleware1', 'nama-middleware12', 'dst\'], 'uses'=>'Controller@method']);
```

4. Bila dibutuhkan kita juga bisa menggunakan route group untuk menggunakan middleware pada banyak route:

```
Route::group(['middleware' => 'nama-middleware'], function() {  
    // route kita..  
});
```

5. Ketika kita membutuhkan sebuah middleware yang berjalan pada semua route, kita bisa menggunakan global middleware dengan menambahkan pada array \$middleware di /app/Http/Kernel.php:

/app/Http/Kernel.php

```
<?php  
....  
class Kernel extends HttpKernel  
{  
    protected $middleware = [  
        // class middleware kita..  
    ];  
}
```

Nah, biasanya di route itu ada parameter. Misalnya kita buat route seperti berikut:

```
Route::get('posts/{post_id}', ['middleware'=>'transform-resource', 'uses'=>'Post\Controller@show']);
```

Kita dapat mengakses post_id dari middleware transform-resource.

Pertanyaannya, untuk apa kita melakukan ini? Kalau dalam kasus saya kerja, ketika membuat API biasanya di controller itu kita akan mencari instance dari post dengan menggunakan parameter post_id dan ketika tidak ditemukan maka akan memberikan response json post_not_found dengan status code 404. Kira-kira seperti ini syntax di controllernya:

app/Http/Controllers/HomeController.php

```
<?php  
....  
use Illuminate\Database\Eloquent\ModelNotFoundException;  
class HomeController {  
    public function showPost($post_id) {  
        try {  
            $post = App\Post::findOrFail($post_id);  
            return $post;  
        } catch(ModelNotFoundException $e) {  
            return response()->json(['error' => 'post_not_found'], 404);  
        }  
    }  
}
```

Nah, cara ini tentunya valid dan bisa dilakukan. Tapi, ketika jenis resource sudah banyak, akan terlihat banyak repetisi kode. Ketika kita bisa mengakses `post_id` dari middleware kita dapat memindahkan proses pencarian model dan pemberian error di middleware sehingga kode kita akan lebih rapi.

Sehingga, hasil akhir dari method diatas akan seperti ini:

app/Http/Controllers/HomeController.php

```
<?php  
....  
use Illuminate\Database\Eloquent\ModelNotFoundException;  
class HomeController {  
    public function showPost(Request $request, $post_id) {  
        return $request->get('post');  
    }  
}
```

Terlihat disini, kita hanya menggunakan `$request->get('post')` untuk mendapatkan *instance* dari Post.

Lebih sederhana kan? Dengan teknik ini, kita bisa lebih fokus pada *action* yang akan kita lakukan pada model. Sementara proses pencarian model dan pemberian respon error jika model tidak ditemukan di *handle* oleh middleware.

Oke, setelah paham hasil akhir yang akan kita tuju, mari kita buat codingnya. Pertama kita siapkan dulu modelnya:

1. Jalankan perintah `php artisan make:model Post -m`. Pada file migration yang dibuat, isilah method `up()` dengan syntax berikut:

/database/migrations/xxxx_xx_xx_xxxxxxx_create_posts_table.php

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title');
        $table->text('content');
        $table->timestamps();
    });
}
```

Disini kita hanya akan menggunakan field `title` dan `content` untuk menyimpan data Post. Pastikan juga file `app/Post.php` telah di-`generate`.

2. Selanjutnya, kita buat seeder untuk Post. Tambahkan syntax berikut pada file seeder:

/database/seeds/DatabaseSeeder.php

```
<?php
...
class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        ...
        $faker = Faker\Factory::create();
        foreach (range(1,5) as $index) {
            // sample post
            $post = new \App\Post;
            $post->title = $faker->sentence(3);
            $post->content = $faker->paragraph;
            $post->save();
        }

        Model::reguard();
    }
}
```

Kita menggunakan faker untuk membuat sample data random. Silahkan install dulu dengan perintah `composer require fzaninotto/faker` dari terminal.

Oke, modelnya sudah siap. Mari kita buat middlewarenya dengan perintah `php artisan make:middleware TranfromResource`. Pada method file middleware yang di-generate isi dengan syntax berikut:

/app/Http/Middleware/TransformResource.php

```
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Closure;
6 use Illuminate\Database\Eloquent\ModelNotFoundException;
7
8 class TransformResource
9 {
10     ....
11     public function handle($request, Closure $next, $resource)
12     {
13         $resource_class = '\App\\' . $resource;
14         $resource_id_key = $resource . '_id';
15         $resource_id = $request->route()->parameters()[$resource_id_key];
16
17         try {
18             $model = $resource_class::findOrFail($resource_id);
19             $request->merge([$resource => $model]);
20             return $next($request);
21         } catch (ModelNotFoundException $e) {
22             return response()->json(['error' => $resource . '_not_found'], 404);
23         }
24     }
25 }
```

Oke, syntax ini cukup panjang. Mari kita bahas satu-persatu:

1. Middleware ini menggunakan middleware parameter. Terlihat pada baris 11 `handle($request, Closure $next, $resource)`.
2. Pada baris 13, kita menentukan nama class untuk modelnya. Penggunaan parameter untuk middleware ini untuk model post akan ditulis `transform-resource:post`. Karena, parameternya berupa `post`, maka kita tambahkan namespacenya `\App\` sehingga hasil akhirnya akan kita dapatkan `\App\post`.

3. Selanjutnya, kita tentukan nama parameter di routing dengan menambahkan `_id` pada parameter middleware. Tentunya, menggunakan teknik ini, kita harus menamai parameter dengan nama `post_id`. Sehingga syntax routenya harus seperti:

```
Route::get('/posts/{post_id}', ...);
```

Begitupun untuk model lain misalnya customer, kita harus menggunakan `customer_id` sebagai parameternya.

4. Baris 15, disinilah kita mengambil nilai dari parameter di route. Ketika syntax ini berjalan, baris ini:

```
$resource_id = $request->route()->parameters()[$resource_id_key];
```

Untuk model post akan berubah menjadi:

```
$resource_id = $request->route()->parameters()['post_id'];
```

Method `$request->route()->parameters()` akan menghasilkan semua parameter dari route dalam bentuk array asosiatif. Itulah sebabnya kita menggunakan `$request->route()->parameters()[$resource_id_key]` untuk mengakses nilainya.

5. Selanjutnya, kita menggunakan `try-catch` untuk menentukan apakah harus meneruskan request atau memberikan pesan error ketika model tidak ditemukan.

Di baris 18, kita mencari instance dari model yang sedang kita cari berdasarkan id yang kita dapatkan dari route. Selanjutnya, kita tambahkan model yang kita temukan pada request di baris 19. Itulah sebabnya kita dapat mengakses model di controller dengan syntax `$request->get('post')`. Terakhir kita meneruskan request (ke controller atau middleware selanjutnya).

Di baris 21-22, kita memberikan error ketika model tidak ditemukan. Disini, kita buat pesan error menjadi dinamis dengan syntax `$resource . '_not_found'`. Sehingga, jika model post yang sedang kita cari, pesan errornya akan `post_not_found`. Sedangkan jika model customer yang kita cari, pesan errornya akan menjadi `customer_not_found`, dst.

Oh iya, jangan lupa import `Illuminate\Database\Eloquent\ModelNotFoundException` agar `try-catch` berfungsi (lihat baris 6).

Nah, setelah middleware dibuat, mari kita register:

app/Http/Kernel.php

```
<?php  
....  
class Kernel extends HttpKernel  
{  
....  
protected $routeMiddleware = [  
....  
    'transform-resource' => \App\Http\Middleware\TransformResource::class,  
];  
}
```

Untuk mendemokannya, mari kita buat resource yang akan menampilkan json sebuah Post ketika kita mengakses /posts/{post_id}.

app/Http/routes.php

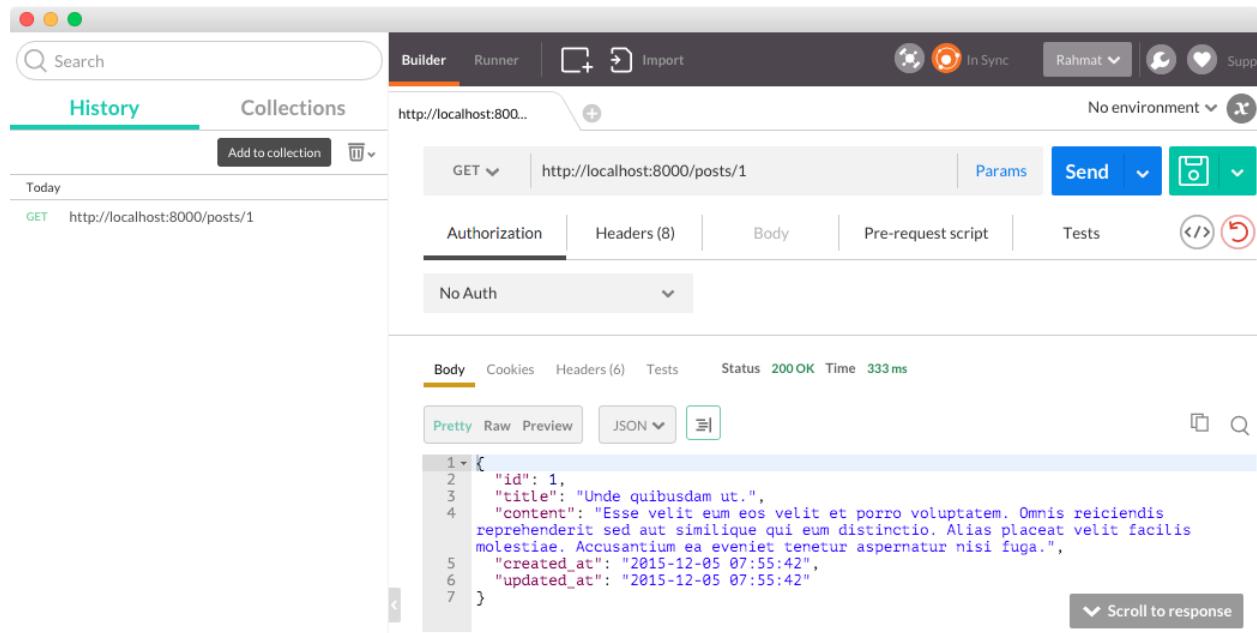
```
Route::get('posts/{post_id}', ['middleware' => 'transform-resource:post', 'uses' \  
=> 'HomeController@showPost']);
```

Kita buat method showPost di HomeController:

/app/Http/Controllers/HomeController.php

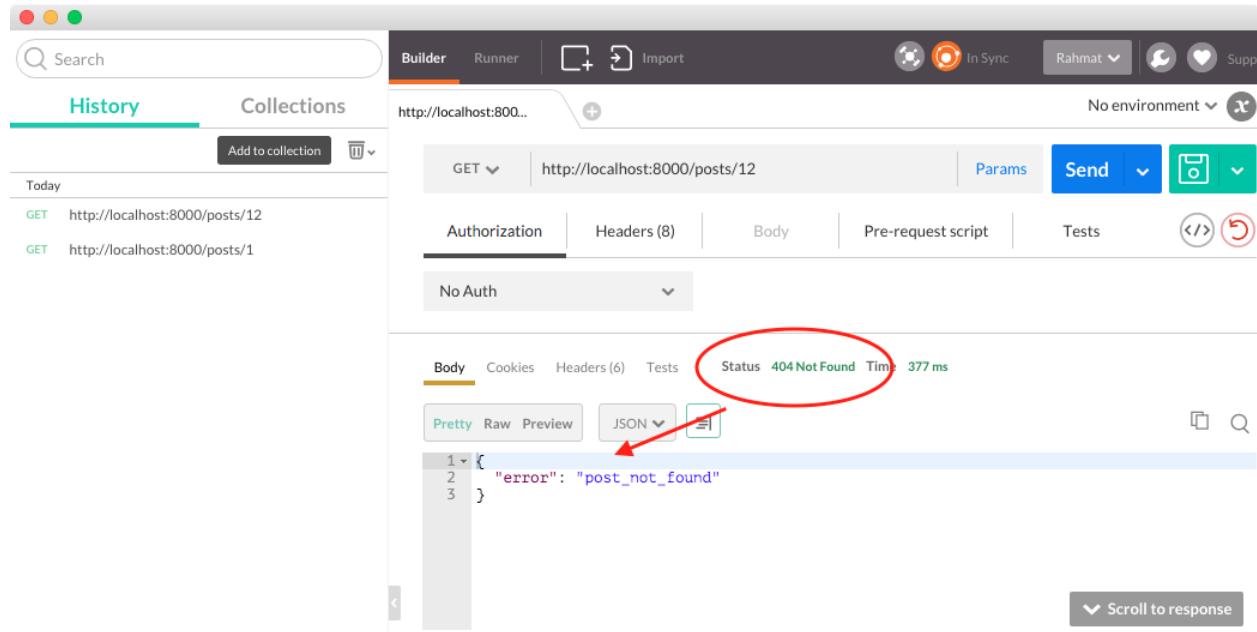
```
public function showPost(Request $request, $post_id)  
{  
    return $request->get('post');  
}
```

Ok, kita test dengan Postman untuk mengakses Post dengan id 1:



Berhasil mengakses Post dalam bentuk JSON

Tapi, jika kita mencoba mengakses post yang tidak ada, misalnya id 12, maka akan muncul error `post_not_found` berikut status code 404:



Error muncul dari middleware

Sip, begitulah cara mengakses parameter dari route pada middleware. Sebagai latihan, cobalah buat model customer berikut sample datanya. Kemudian, gunakan middleware ini untuk mengakses model Customer.



Kalau bingung, cek source code bab ini di link dibawah.

Controller Middleware

Sejauh ini ketika kita hendak menggunakan sebuah route middleware, kita selalu menambahkan di file route. Ini cara yang saya sering gunakan, menurut saya ini lebih rapi karena kita dapat melihat middleware yang sedang aktif pada route yang kita miliki hanya dengan menggunakan file route.

Sebenarnya ada cara lain, yaitu dengan menambahkan di controller. Misalnya, kita punya route seperti berikut:

app/Http/routes.php

```
Route::get('about', 'HomeController@about');
```

Kita dapat menambahkan middleware pada route ini dengan menambahkannya pada controller. Caranya, kita gunakan method `__construct` (buat dulu jika belum ada) di controller untuk menambahkan middleware pada method `about`:

/app/Http/Controllers/HomeController.php

```
<?php  
....  
class HomeController extends Controller  
{  
    public function __construct()  
    {  
        $this->middleware('access-logs', ['only' => ['about']]);
    }  
    public function about()  
    {  
        return "Ini tentang kita, tentang kisah kita.. #ciee..";
    }  
    ....
}
```

Pada method `__construct()`, kita menggunakan syntax:

```
$this->middleware('access-logs', ['only' => ['about']]));
```

Untuk mengaktifkan middleware `access-logs` hanya pada method `about`. Cara penulisan lainnya, jika ingin mengaktifkan middleware untuk semua method:

```
$this->middleware('access-logs');
```

Dan jika ingin mengaktifkan middleware pada semua method, kecuali pada method-method tertentu:

```
$this->middleware('access-logs', ['except' => ['welcome', 'help']]));
```

Nah, cukup sulit kan untuk mengetahui middleware apa yang aktif pada route `/about` kalau hanya dengan melihat file route? Itulah alasan saya jarang menggunakan teknik ini.

Untuk mengatasinya, sebagaimana kita pelajari di bab 4, kita dapat menggunakan perintah `php artisan route:list` untuk mengetahui semua route pada aplikasi kita berikut middleware yang aktif pada route tersebut.

\$ php artisan route:list					
Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/	Closure		
	GET HEAD	about	App\Http\Controllers\HomeController@about		access-logs
	GET HEAD	admin-test	App\Http\Controllers\HomeController@adminTest		api-token:admin
	GET HEAD	api-test	App\Http\Controllers\HomeController@apiTest		api-token
	GET HEAD	customer/{customer_id}	App\Http\Controllers\HomeController@showCustomer		transform-resource:customer
	GET HEAD	editor-test	App\Http\Controllers\HomeController@editorTest		api-token:editor
	GET HEAD	log-test	App\Http\Controllers\HomeController@logTest		api-token,access-log
	GET HEAD	posts/{post_id}	App\Http\Controllers\HomeController@showPost		transform-resource:post

Mengecek middleware pada route

Middleware Group (update Laravel 5.2)

Di Laravel 5.2 ketika kita membuat project baru, isian file route secara default akan seperti ini:

/app/Http/routes.php

```
<?php

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

Route::group(['middleware' => ['web']], function () {
    //
});
```

Isian middleware `web` merupakan contoh penggunaan middleware grup. Jadi, apa itu middleware grup?

Dengan menggunakan middleware grup, kita dapat mengelompokan beberapa middleware dalam satu middleware. Pada sample middleware `web`, di file `/app/Http/Kernel.php` kita akan mendapati isian seperti berikut:

/app/Http/Kernel.php

```
<?php
...
class Kernel extends HttpKernel
{
    ...
    protected $middlewareGroups = [
        'web' => [
            \App\Http\Middleware\EncryptCookies::class,
            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
            \Illuminate\Session\Middleware\StartSession::class,
            \Illuminate\View\Middleware\ShareErrorsFromSession::class,
            \App\Http\Middleware\VerifyCsrfToken::class,
        ],
        ...
    ];
}
```

Disini terlihat, middleware `web` adalah gabungan dari 5 middleware `EncryptCookies`, `AddQueuedCookiesToResponse`, `StartSession`, `ShareErrorsFromSession` dan `VerifyCsrfToken`.

Contoh penggunaan middleware grup adalah ketika kita hendak menggunakan satu aplikasi Laravel untuk dua jenis aplikasi, Web dan API. Salah satu contohnya, pada

penggunaan Web, kita akan membutuhkan CSRF Token dan Session, tapi pada API kita tidak akan membutuhkannya. Menggunakan teknik ini, kita akan membuat request ke API lebih cepat karena kita tidak perlu menjalankan middleware yang tidak dibutuhkan.



Source code dari latihan di baba ini bisa didapat di [Bitbucket](#)¹⁶⁶



Ringkasan

Memahami penggunaan middleware merupakan dasar yang harus kita pahami untuk menggunakan fitur-fitur Laravel yang lainnya. Pada bab selanjutnya, kita akan mempelajari tentang authentikasi dan hak akses, tentunya akan lebih mudah karena kita sudah paham tentang middleware.

Siap belajar lagi? :)

¹⁶⁶<https://bitbucket.org/rahmatawaludin/sample-middleware/commits/all>

Authentikasi & Authorisasi, dari Login hingga Hak Akses

Ada dua istilah yang harus kita pahami dalam mempelajari bab ini, Authentikasi dan Authorisasi. Authentikasi adalah proses dimana kita mengidentifikasi seorang user berdasarkan credential yang dia kirim (username, email, password) dengan data di database. Proses ini kita lakukan sebagai tahapan pertama melindungi sistem. Setelah user ter-authentikasi, proses selanjutnya yang kita lakukan adalah authorisasi. Proses ini kita lakukan untuk mengecek apakah seorang user memiliki akses untuk melakukan sebuah tindakan pada resource di aplikasi kita.

Saat buku ini ditulis, proses authentikasi sudah sangat berkembang. Dari mulai authentikasi biasa dengan session, oauth (social login), token login, dll. Seringkali, beberapa teknik authentikasi digabungkan dengan teknik authentikasi yang lain dalam satu aplikasi.

Begitupun dengan authorisasi, saat buku ini ditulis, ada banyak teknik authorisasi dan yang cukup populer adalah RBAC, ACL dan Policy Based. Setiap teknik authorisasi ini dapat kita gunakan di Laravel dengan *effort* yang ringan.

Dari sekian banyak cara melakukan authentikasi dan authorisasi ini, mana yang harus kita gunakan di aplikasi kita? Nah, itulah yang akan kita pelajari di bab ini.

Memahami Alur Authentikasi Laravel

Untuk mempelajari topik ini, saya asumsikan Anda sudah pernah membuat authentikasi sederhana di aplikasi web, khususnya dengan PHP. Jadi, dalam proses authentikasi itu, ada beberapa tahapan yang kita lakukan:

1. Kita siapkan table di database, minimalnya ada isian username / email dan password (field-field ini selanjutnya kita sebut *credential*). Isian password biasanya sudah di-*encrypt*.
2. Authentikasi dilakukan dengan mengirimkan *credential* ke server. Jika **tidak sesuai** dengan yang tercatat di database, kita batalkan proses authentikasi, biasanya sambil mengirim pesan error. Jika **sesuai**, kita akan set *Session* di server. Secara default, kita juga akan mengeset *cookie* pada browser client yang menyimpan data login ini.

3. Setelah berhasil login, setiap kali client melakukan request ke server, dia juga akan mengirim *cookie*. Cookie ini akan kita cek dengan data session di server. Selama session masih valid, akses akan diizinkan.
4. Untuk menutup akses, biasanya ada dua cara. Pertama, client melakukan logout manual yang akan menghapus data session di server. Sehingga meskipun *cookie* di browser client ada yang meng-*hijack* (dicopy), yang meng-*hijack* tetap tidak dapat mengakses aplikasi.

Cara kedua dengan menggunakan *expired*. Jadi, kita set waktu *expired* di session. Meskipun user tidak *logout* dari aplikasi, jika session sudah *expired*, *cookie* yang dia kirim tetap tidak akan valid. Sehingga dia dipaksa untuk login kembali.

Tentunya, membuat fitur sesuai urutan logika diatas, terkadang cukup merepotkan. Terlebih jika menggunakan PHP native (tanpa framework). Belum lagi, jika kita ingin menambahkan fitur seperti logging, remember user, suspend, throttle, dll.

Beruntunglah kita yang sudah mengenal framework Laravel. Di Laravel, untuk membuat fitur ini kita cukup meng-*copy - paste* beberapa syntax dan memodifikasinya sesuai kebutuhan. Mari kita mulai dengan mengaktifkan fitur login dan logout.

Sebelum memulai coba cek dulu konfigurasi authentikasi di `config/auth.php`:

config/auth.php

```
<?php  
return [  
    /*  
     * Default Authentication Driver  
     */  
    'driver' => 'eloquent',  
  
    /*  
     * Authentication Model  
     */  
    'model' => App\User::class,  
  
    /*  
     * Authentication Table  
     */  
    'table' => 'users',  
  
    /*  
     * Password Reset Settings  
     */  
    'password' => [
```

```

'email' => 'emails.password',
'table' => 'password_resets',
'expire' => 60,
],
];

```

Mari kita pelajari apa yang terdapat pada file konfigurasi ini:

- `driver` : Isian ini akan digunakan bagaimana laravel mengakses database untuk mendapatkan data credential user. Ada dua isi eloquent dan database. Jika kita isi eloquent, kita harus mengisi isian `model`. Sementara, jika kita isi database maka kita perlu mengisi isian `table`.
- `model & table` : Diisi dengan nama model atau table yang sesuai yang berisi data credential user.
- `password` : Isian ini digunakan untuk fitur reset password. Lebih lengkapnya akan kita bahas di topik **Password Reminder & Reset**.

Seperti yang saya sampaikan, untuk membuat fitur authentikasi ini, kita cukup *copy-paste* syntax. Pertama, kita copy syntax untuk routing dari [dokumentasi¹⁶⁷](#). Sehingga file routing kita akan seperti berikut:

app/Http/routes.php

```

<?php
...
// Authentication routes...
Route::get('auth/login', 'Auth\AuthController@getLogin');
Route::post('auth/login', 'Auth\AuthController@postLogin');
Route::get('auth/logout', 'Auth\AuthController@getLogout');

// Registration routes...
Route::get('auth/register', 'Auth\AuthController@getRegister');
Route::post('auth/register', 'Auth\AuthController@postRegister');

```

Pada routing ini, kita membuat url `/auth/login` untuk mengakses halaman login, `/auth/logout` untuk logout dari website dan `/auth/register` untuk mendaftar ke web.

Kedua, kita syntax untuk menampilkan form login dan register dari [dokumentasi¹⁶⁸](#). Secara default, kedua file ini tidak ada, kita harus membuatnya di `resources/views/auth/login.blade.php` dan `resources/views/auth/register.blade.php`:

¹⁶⁷ <http://laravel.com/docs/5.1/authentication#included-routing>

¹⁶⁸ <http://laravel.com/docs/5.1/authentication#included-views>

resources/views/auth/login.blade.php

```
<!-- resources/views/auth/login.blade.php -->

<form method="POST" action="/auth/login">
  {!! csrf_field() !!}

  <div>
    Email
    <input type="email" name="email" value="{{ old('email') }}">
  </div>

  <div>
    Password
    <input type="password" name="password" id="password">
  </div>

  <div>
    <input type="checkbox" name="remember"> Remember Me
  </div>

  <div>
    <button type="submit">Login</button>
  </div>
</form>
```

resources/views/auth/register.blade.php

```
<!-- resources/views/auth/register.blade.php -->

<form method="POST" action="/auth/register">
  {!! csrf_field() !!}

  <div>
    Name
    <input type="text" name="name" value="{{ old('name') }}">
  </div>

  <div>
    Email
    <input type="email" name="email" value="{{ old('email') }}">
  </div>
```

```
<div>
    Password
    <input type="password" name="password">
</div>

<div>
    Confirm Password
    <input type="password" name="password_confirmation">
</div>

<div>
    <button type="submit">Register</button>
</div>
</form>
```

Tentunya, untuk styling kedua form ini diserahkan kepada kita sebagai developer.

Ada tahapan yang tidak ditulis didokumentasi, yaitu setelah login Laravel akan otomatis mengarahkan kita ke URL /home. Untuk itu, mari kita buat route untuk url ini yang akan menampilkan bahwa user telah login:

app/Http/routes.php

```
Route::get('/home', function() {
    return "Anda berhasil login";
});
```

Setelah semua konfigurasi diatas dilakukan, kita membutuhkan table yang akan menyimpan data users. Secara default, Laravel telah memiliki migrationnya di /database/migrations/2014_10_12_000000_create_users_table.php. Kita cukup menjalankan php artisan migrate. Pastikan di database akan seperti ini:

The screenshot shows the MySQL Workbench interface with the database 'homestead' selected. The 'Structure' tab is active, displaying the schema for the 'users' table. The table has the following columns:

Field	Type	Length	Unsigned	Zerofill
id	INT	10	<input checked="" type="checkbox"/>	
name	VARCHAR	255	<input type="checkbox"/>	
email	VARCHAR	255	<input type="checkbox"/>	
password	VARCHAR	60	<input type="checkbox"/>	
remember_token	VARCHAR	100	<input type="checkbox"/>	
created_at	TIMESTAMP		<input type="checkbox"/>	
updated_at	TIMESTAMP		<input type="checkbox"/>	

Default migration untuk authentikasi

Kini, kita dapat mengakses halaman register di /auth/register:

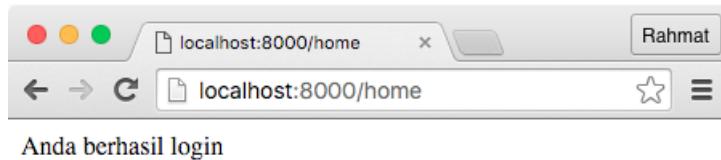
The screenshot shows a web browser window with two tabs. The active tab is 'localhost:8000/auth/register' and the other tab is 'localhost:8000/auth/register'. The page displays a registration form with the following fields:

- Name: Joni
- Email: joni@laravel.com
- Password: (redacted)
- Confirm Password: (redacted)

A 'Register' button is at the bottom of the form.

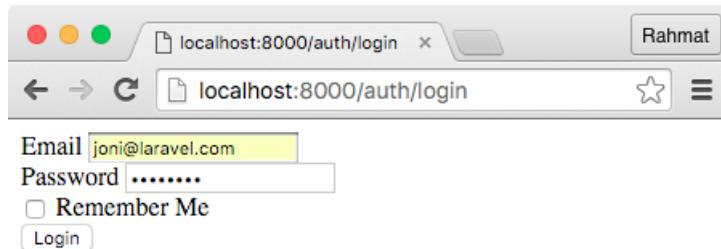
halaman register

Cobalah masukan nama, email dan password yang diinginkan. Setelah klik register, kita akan otomatis login dan diarahkan ke /home.



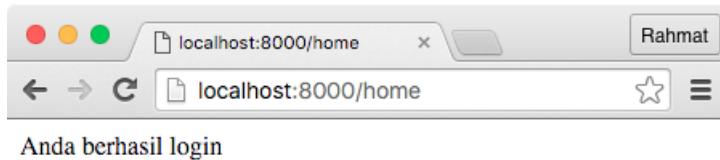
berhasil daftar

Jika sudah berhasil login, kunjungi `/auth/logout` untuk logout dari aplikasi. Secara default, kita akan kembali diarahkan ke `/`. Kini, cobalah login dengan mengakses `/auth/login` dan menggunakan email dan password yang tadi di daftarkan.



Halaman login

Pastikan kita berhasil login dan logout kembali.



Berhasil login

Proteksi route dengan Auth Middleware

Authentikasi, biasanya kita gunakan untuk memproteksi sebuah halaman agar tidak diakses tanpa login. Dalam contoh diatas, kita belum melakukan hal ini pada route `/home`. Ini bisa kita buktikan dengan logout dari aplikasi dengan mengakses `/auth/logout` dan mengakses halaman `/home`. Masih bisa kan?

Untuk melakukan proteksi ini, kita dapat menggunakan middleware `auth` yang sudah disediakan oleh Laravel. Misalnya, pada route `/home`, kita ubah menjadi seperti berikut:

app/Http/routes.php

```
Route::get('/home', ['middleware' => 'auth', function () {
    return "Anda berhasil login";
}]);
```

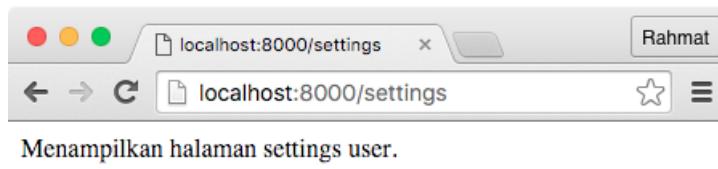
Kini, cobalah akses url `/home` tanpa login, kita pasti diarahkan ke halaman login terlebih dahulu.

Kelebihan lain dari penggunaan authentikasi bawaan Laravel adalah kita dapat melakukan redirect ke url yang dikehendaki user setelah login (melewati url `/home`). Misalnya, kita punya route `/settings` seperti berikut:

app/Http/routes.php

```
Route::get('/settings', ['middleware' => 'auth', function () {
    return "Menampilkan halaman settings user.";
}]);
```

Kini, coba logout dari aplikasi dan akses /settings. Kita akan diarahkan ke halaman login. Setelah login, kita tidak akan diarahkan ke /home, tapi langsung ke /settings. Sip, simple tapi sangat bermanfaat.. :)

**Redirect otomatis setelah login**

Menambah field baru ke table user

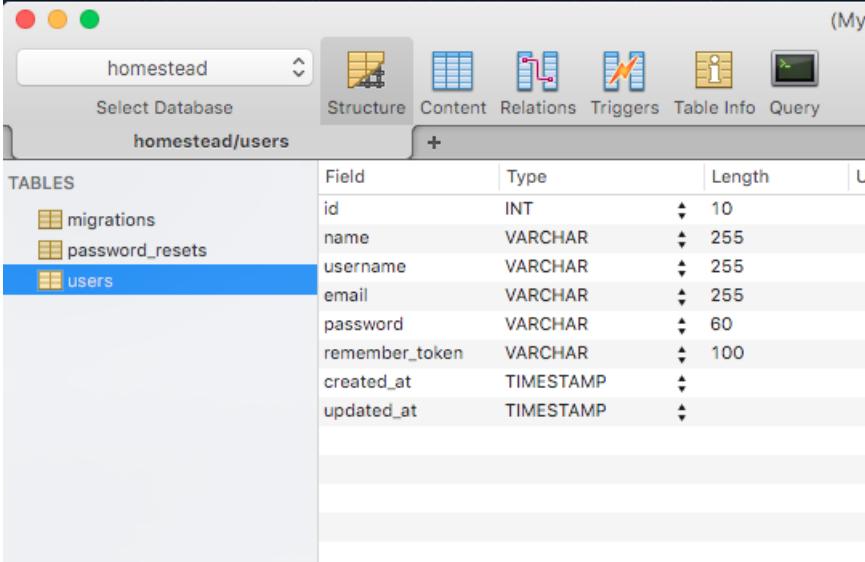
Sangat umum jika kita hendak menambahkan field lain di table user misalnya nomor telepon, alamat, username, dll. Mari kita praktekan bagaimana melakukannya dengan menambahkan field username ketika register dan menyimpannya di table users.

1. Untuk memudahkan kita akan modifikasi file migration untuk table users. Cara lainnya, kita dapat membuat migration baru:

/database/migrations/2014_10_12_000000_create_users_table.php

```
....  
public function up()  
{  
    Schema::create('users', function (Blueprint $table) {  
        $table->increments('id');  
        $table->string('name');  
        $table->string('username')->unique();  
        $table->string('email')->unique();  
        $table->string('password', 60);  
        $table->rememberToken();  
        $table->timestamps();  
    });  
}  
....
```

Refresh struktur database dengan `php artisan migrate:refresh`. Pastikan kini strukturnya akan seperti berikut:



The screenshot shows the MySQL Workbench interface with the 'Structure' tab selected for the 'homestead' database. The 'homestead/users' table is selected. The table structure is as follows:

Field	Type	Length	U
<code>id</code>	INT	10	
<code>name</code>	VARCHAR	255	
<code>username</code>	VARCHAR	255	
<code>email</code>	VARCHAR	255	
<code>password</code>	VARCHAR	60	
<code>remember_token</code>	VARCHAR	100	
<code>created_at</code>	TIMESTAMP		
<code>updated_at</code>	TIMESTAMP		

Menambah field username

2. Tambahkan field `username` pada form register.

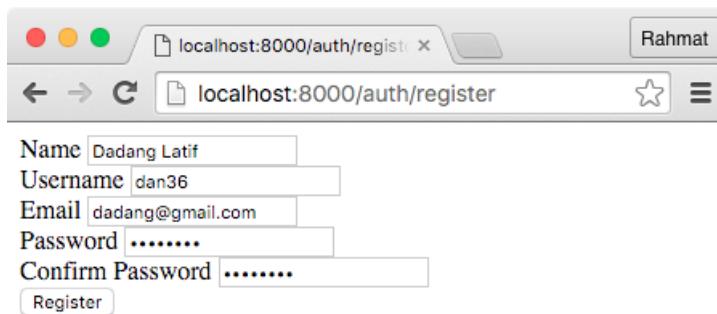
/resources/views/auth/register.blade.php

```
<!-- resources/views/auth/register.blade.php -->

<form method="POST" action="/auth/register">
    ...
    <div>
        Username
        <input type="text" name="username" value="{{ old('username') }}"/>
    </div>
    ...

```

Pastikan tampilannya akan seperti berikut:

**Username di form login**

3. Disisi controller, kita perlu mengubah /app/Http/Controllers/Auth/AuthController.php agar menerima field username:

/app/Http/Controllers/Auth/AuthController.php

```
<?php

...
class AuthController extends Controller
{
    ...

    protected function validator(array $data)
    {
        return Validator::make($data, [
            ...
            'username' => 'required|max:255|unique:users',
        ]);
    }
}
```

```

    ...
protected function create(array $data)
{
    return User::create([
        ...
        'username' => $data['username'],
    ]);
}

```

Method `validator` kita modifikasi agar field `username` harus diisi dan tidak boleh ada yang sama di table `users` (lebih lengkap tentang validasi data akan kita pelajari di bab “Validasi Data”). Method `create` kita modifikasi agar menyimpan field `username` ke table `users`.

4. Kitapun harus melakukan perubahan pada model `User` agar mengizinkan melakukan *mass assignment* untuk field `username`.

/app/User.php

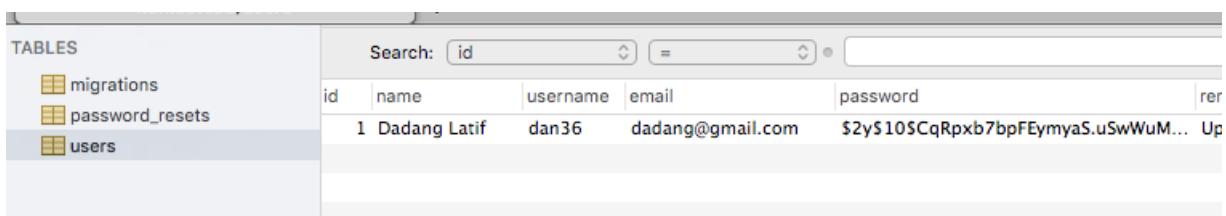
```

<?php

...
class User extends Model implements AuthenticatableContract,
    AuthorizableContract,
    CanResetPasswordContract
{
    ...
    protected $fillable = ['name', 'username', 'email', 'password'];
    ...
}

```

Cobalah register user, kini kita akan diminta mengisi username dari user yang kita buat. Setelah berhasil register di database pun akan tersimpan nama user yang kita buat.



id	name	username	email	password
1	Dadang Latif	dan36	dadang@gmail.com	\$2y\$10\$CqRpzb7bpFEymyaS.uSwWuM...

Username berhasil disimpan ke database

Menggunakan Username untuk login

Selain menggunakan `email` untuk login, kita juga dapat menggunakan field lain. Contoh kasusnya misal menggunakan ID khusus untuk user yang kita generate di sistem atau menggunakan username. Tahapan yang harus kita lakukan adalah menambahkan field tersebut pada table user dan mengubah property `username` pada `/app/Http/Controllers/Auth/AuthController.php` dengan nama field yang kita inginkan.

Karena pada topik sebelumnya kita telah menambahkan field `username`, mari kita buat sample agar kita bisa login dengan menggunakan `username`. Jika belum, ikuti tutorial di topik sebelumnya untuk menambahkan field `username` ke form dan database.

Selanjutnya, kita tambahkan isian berikut pada `/app/Http/Controllers/Auth/AuthController.php`:

```
<?php  
....  
class AuthController extends Controller  
{  
  
    protected $username = 'username';  
    ....  
}
```

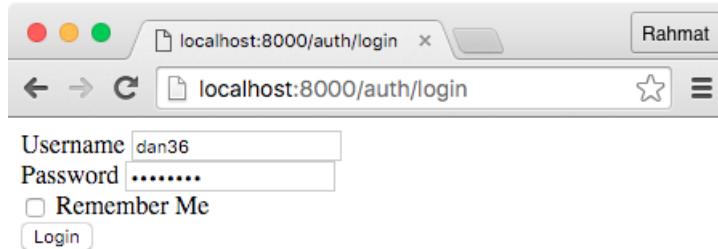
Isian `$username` ini secara default berisi `email`. Selanjutnya, kita ubah form login:

/resources/views/auth/login.blade.php

```
<!-- resources/views/auth/login.blade.php -->  
  
<form method="POST" action="/auth/login">  
    {!! csrf_field() !!}  
  
    <div>  
        Email  
        <input type="email" name="email" value="{{ old('email') }}"/>  
    </div>  
  
    <div>  
        Username  
        <input type="text" name="username" value="{{ old('username') }}"/>  
    </div>
```

```
....  
</form>
```

Sekarang, cobalah akses halaman login dan login menggunakan username dan password yang telah kita daftarkan.



Berhasil menggunakan field username untuk login

Pastikan kita berhasil login.

Kustomisasi Login

Selain mengikuti konfigurasi default, kita juga dapat melakukan beberapa perubahan jika diinginkan. Berikut beberapa kustomisasi yang dapat dengan mudah kita lakukan:

Merubah url setelah logout

Tambahkan properti `$redirectToLogout` pada `/app/Http/Controllers/Auth/AuthController.php` dengan url yang kita kehendaki.

Merubah url untuk login

Tambahkan properti `$loginPath` pada `/app/Http/Controllers/Auth/AuthController.php` dengan url yang kita kehendaki. Ubah juga isian pada file `/app/Http/routes.php`.

Mengubah url setelah berhasil login

Secara default kita akan diarahakan ke url `/home` setelah berhasil login. Untuk merubahnya ke url lain, tambahkan properti `$redirectTo` (atau `$redirectToPath`) pada `/app/Http/Controllers/Auth/AuthController.php` dengan url yang kita kehendaki.

Pesan Error

Ketika kita tidak berhasil login atau register, sebenarnya Laravel mengirimkan instance dari MessageBag yang bisa kita gunakan untuk menampilkan pesan error. Mari kita tambahkan pada tampilan login dan register agar menampilkan pesan error ini.

Tambahkan baris ini pada file `/resources/views/auth/login.blade.php`:

/resources/views/auth/login.blade.php

```
<form method="POST" action="/auth/login">

@if (count($errors) > 0)
<ul>
    @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
@endif
....
```

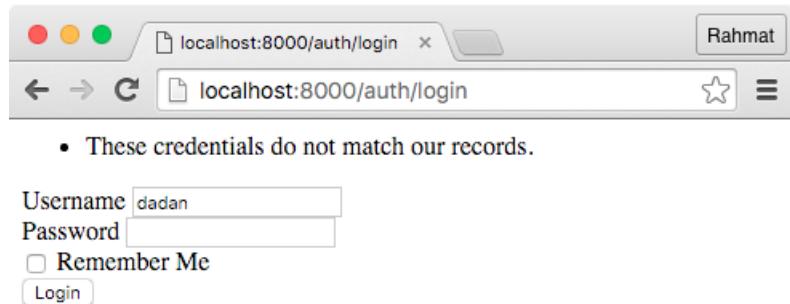
Dan pada file `/resources/views/auth/register.blade.php`:

/resources/views/auth/register.blade.php

```
<form method="POST" action="/auth/register">

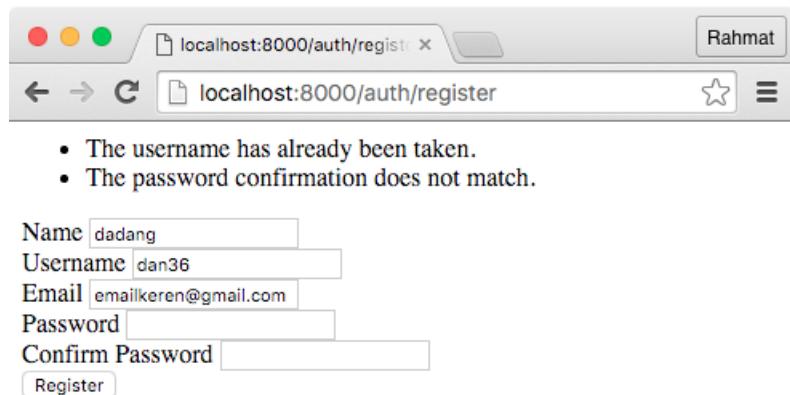
@if (count($errors) > 0)
<ul>
    @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
@endif
....
```

Kini, jika kita salah memasukan isian pada form login atau register akan muncul error:



A screenshot of a web browser window titled "localhost:8000/auth/login". The address bar also shows "localhost:8000/auth/login". The page contains a form with fields for "Username" (containing "dadan") and "Password". Below the form is a checkbox labeled "Remember Me" and a "Login" button. An error message at the top of the page states: "• These credentials do not match our records."

Pesan error pada halaman login



A screenshot of a web browser window titled "localhost:8000/auth/register". The address bar shows "localhost:8000/auth/register". The page contains a form with fields for "Name" (containing "dadang"), "Username" (containing "dan36"), "Email" (containing "emailkeren@gmail.com"), "Password", and "Confirm Password". Below the form is a "Register" button. Two error messages are displayed above the form: "• The username has already been taken." and "• The password confirmation does not match."

Pesan error pada halaman register

Tentunya, ini jika dibutuhkan, kita dapat menambahkan styling pada pesan error ini sesuai kebutuhan kita.

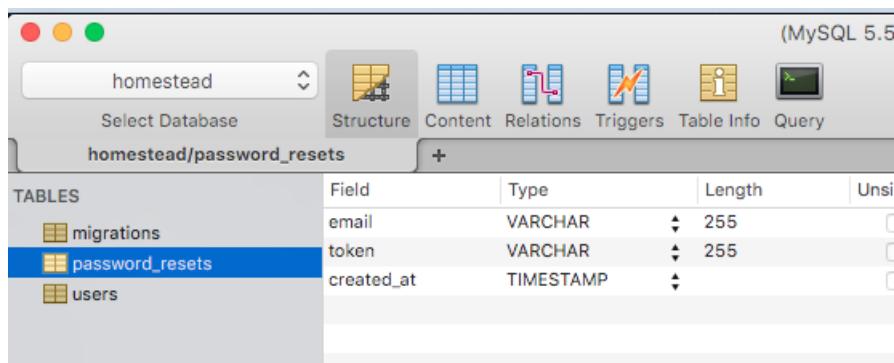
Password Reminder & Reset

Fitur reset password sangat umum dibutuhkan di sebuah webapp. Cara kerja dari fitur ini yaitu seperti berikut:

1. User mengunjungi halaman reset password
2. Memasukan email akunnya
3. Mendapat email dari sistem berisi link untuk mereset password
4. Mengunjungi halaman untuk mengganti password dengan meng-klik link di email
5. Mengganti password

Jika dibuat secara manual, semua tahapan ini cukup merepotkan. Di Laravel, kita dapat membuat fitur ini dengan copy paste beberapa code.

1. Kita harus memiliki table untuk menyimpan token reset password yang berisi field `email` dan `token`. Di Laravel secara default akan ada migration untuk table `password_resets` dapat dilihat di `/database/migrations/2014_10_12_100000_create_password_resets_table.php`. Setelah di migrate, di database seharusnya akan seperti berikut



The screenshot shows the MySQL Workbench interface with the database 'homestead' selected. The 'Structure' tab is active, displaying the columns of the 'password_resets' table. The table has three columns: 'email' (VARCHAR, 255), 'token' (VARCHAR, 255), and 'created_at' (TIMESTAMP). Other tables like 'migrations' and 'users' are also listed in the left pane.

Field	Type	Length	Unsi
email	VARCHAR	255	
token	VARCHAR	255	
created_at	TIMESTAMP		

Table `password_resets`

2. Kita harus mengatur agar Laravel menggunakan table `password_resets` untuk menyimpan token reset password. Ini sudah dilakukan secara default. Dapat dilihat di `/config/auth.php` pada isian **Password Reset Settings**.
3. Model user harus meng-implement interface `Illuminate\Contracts\Auth\CanResetPassword` dan menggunakan trait `Illuminate\Auth\Passwords\CanResetPassword`. Ini sudah dilakukan secara default, bisa dilihat di `/app/User.php`.
4. Menambahkan routing untuk menampilkan halaman reset password dan ubah password:

/app/Http/routes.php

```

...
// Password reset link request routes...
Route::get('password/email', 'Auth\PasswordController@getEmail');
Route::post('password/email', 'Auth\PasswordController@postEmail');

// Password reset routes...
Route::get('password/reset/{token}', 'Auth\PasswordController@getReset');
Route::post('password/reset', 'Auth\PasswordController@postReset');

```

Syntax ini dapat di copy dari dokumentasi¹⁶⁹

5. Tambahkan dua view berikut:

¹⁶⁹<https://laravel.com/docs/5.1/authentication#resetting-routing>

/resources/views/auth/password.blade.php

```
<form method="POST" action="/password/email">
    {!! csrf_field() !!}

    @if (count($errors) > 0)
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    @endif

    <div>
        Email
        <input type="email" name="email" value="{{ old('email') }}">
    </div>

    <div>
        <button type="submit">
            Send Password Reset Link
        </button>
    </div>
</form>
```

View diatas digunakan untuk menampilkan halaman reset password.

/resources/views/auth/reset.blade.php

```
<!-- resources/views/auth/reset.blade.php -->

<form method="POST" action="/password/reset">
    {!! csrf_field() !!}
    <input type="hidden" name="token" value="{{ $token }}>

    @if (count($errors) > 0)
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    @endif

    <div>
```

```
Email
<input type="email" name="email" value="{{ old('email') }}">
</div>

<div>
    Password
    <input type="password" name="password">
</div>

<div>
    Confirm Password
    <input type="password" name="password_confirmation">
</div>

<div>
    <button type="submit">
        Reset Password
    </button>
</div>
</form>
```

View diatas merupakan view yang akan diakses oleh user ketika mengklik link dari email yang diterimanya untuk merubah password.

Kedua view diatas dapat di copy dari [dokumentasi¹⁷⁰](#)

6. Tambahkan view untuk email yang akan dikirim ke user. Email ini akan berisi link token yang akan digunakan user untuk merubah password:

/resources/views/emails/password.blade.php

```
<!-- resources/views/emails/password.blade.php -->

Click here to reset your password: {{ url('password/reset/' . $token) }}
```

7. Untuk memudahkan, mari kita tambah link forgot password dari halaman login:

¹⁷⁰ <https://laravel.com/docs/5.1/authentication#resetting-views>

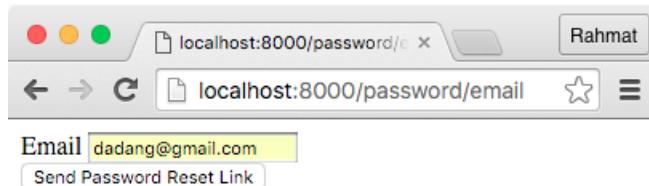
/resources/views/auth/login.blade.php

```
....  
<div>  
    <button type="submit">Login</button>  
    <a href="/password/email">Forgot password?</a>  
</div>  
</form>
```

Untuk memudahkan development, mari kita buat agar email yang dikirim oleh Laravel diarahkan ke file log. Caranya, ubah isian MAIL_DRIVER menjadi log pada file .env.

Jika menggunakan `php artisan serve` untuk menjalankan project Laravel, pastikan untuk merestart kembali perintah tersebut setiap kali melakukan perubahan pada file `.env`.

Mari kita test fitur ini. Silahkan kunjungi halaman `/password/email`. Akan muncul tampilan seperti ini:

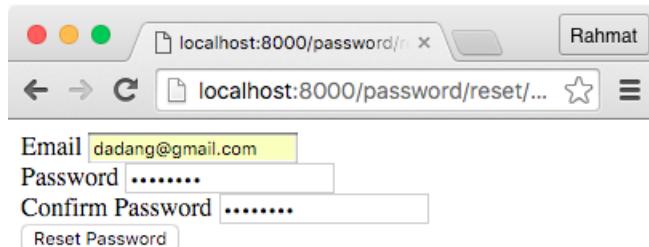
**Request reset password link**

Setelah memasukan email dan klik pada tombol “Send Reset Password Link”, pada file `/storage/logs/laravel.log` akan muncul baris berikut:

/storage/logs/laravel.log

```
....  
[2015-12-27 09:45:44] local.DEBUG: Message-ID: <ee28f34c469aac13e40dcad3916d0c57\  
@localhost>  
Date: Sun, 27 Dec 2015 09:45:44 +0000  
Subject: Your Password Reset Link  
From:  
To: dadang@gmail.com  
MIME-Version: 1.0  
Content-Type: text/html; charset=utf-8  
Content-Transfer-Encoding: quoted-printable  
  
<!-- resources/views/emails/password.blade.php -->  
  
Click here to reset your password: http://localhost:8000/password/reset/721ac47a\  
5004774f49ff3c3f9824f72a5a9104be2f7ffa7a52874903fe7367bb
```

Link ini yang akan digunakan oleh user untuk me-reset password. Silahkan copy paste link tersebut ke browser. Selanjutnya, masukan email beserta password baru bagi user tersebut.

**Ubah password**

Setelah berhasil, kita akan diarahkan ke halaman /home. Cobalah logout dan login kembali. Pastikan password baru berhasil kita gunakan.

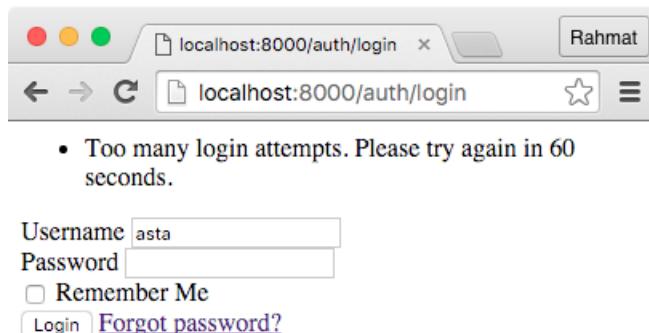
Secara default, link reset password yang kita kirim ke user akan *expire* setelah 60 menit. Untuk merubahnya, ganti isian password.expire pada /config/auth.php.

Menggunakan Throttles Login

Fitur throttle sangat berguna untuk meningkatkan keamanan webapp kita. Cara kerja fitur ini adalah dengan mencatat berapa kali sebuah username + ip gagal login ke webapp kita (catatan ip dan jumlah login gagal disimpan pada file cache). Ketika sudah gagal login 5x berturut-turut, maka Laravel akan menutup akses login untuk IP tersebut selama 60 detik.

Untuk mengaktifkan fitur ini, kita cukup menggunakan trait `Illuminate\Foundation\Auth\ThrottlesLogins` pada `/app/Http/Controllers/Auth/AuthController.php`. Yang seharusnya sudah aktif secara default.

Cobalah login dengan password yang salah selama 5x berturut-turut. Akan muncul tampilan seperti berikut:



Throttles Login

Merubah maximum jumlah login gagal

Secara default, setelah user gagal 5 kali melakukan login, maka dia tidak bisa login kembali selama 60 detik. Untuk merubah jumlah maksimum gagal login, tambahkan properti `$maxLoginAttempts` di `/app/Http/Controllers/Auth/AuthController.php` dengan jumlah yang kita kehendaki.

Merubah waktu timeout

Secara default, waktu timeout setelah mencapai maksimum gagal login adalah 60 detik. Untuk merubahnya, tambahkan properti `$lockoutTime` di `/app/Http/Controllers/Auth/AuthController.php` dengan durasi yang kita kehendaki dalam menit.

Auto Generate Authentikasi (Update Laravel 5.2)

Selain menggunakan cara manual diatas, di Laravel 5.2 kita juga dapat menggunakan generator. Caranya, kita jalankan perintah:

```
php artisan make:auth
```

Ketika perintah ini dijalankan Laravel akan membuat routing, controller dan view yang dibutuhkan untuk membuat fitur authentikasi. Lebih lengkap tentang fitur ini dapat dibaca di bab terakhir.

Menambahkan fitur Suspend

Fitur ini umum digunakan di berbagai webapp, biasanya bisa kita peroleh dengan menginstall package. Karena fitur seperti cukup mudah untuk dibuat, mari kita coba membuatnya dari nol.

Logic dari fitur ini adalah kita akan menambah field suspended di table users yang akan menyimpan nilai 1 / 0 yang akan menentukan apakah user sedang di suspend atau tidak. Untuk melakukan pengecekan ini, kita dapat menggunakan berbagai cara. Salah satu cara yang bisa kita ambil adalah dengan memodifikasi middleware Authenticate yang menangani authentikasi. Silahkan ikuti langkah berikut:

1. Kita akan menambah field suspended ke table users. Bisa dengan cara membuat migration baru atau merubah migration yang sudah ada. Mari kita ubah yang sudah ada:

/database/migrations/2014_10_12_000000_create_users_table.php

```
....  
public function up()  
{  
    Schema::create('users', function (Blueprint $table) {  
        ....  
        $table->boolean('suspended')->default(0);  
        ....  
    });  
}  
....
```

2. Buat agar field suspended selalu menjadi boolean pada model User:

/app/User.php

```
1 <?php
2 ...
3
4 class User extends Model implements AuthenticatableContract,
5   AuthorizableContract,
6   CanResetPasswordContract
7 {
8 ...
9   protected $casts = [
10     'suspended' => 'boolean',
11   ];
12 }
```

3. Kita modifikasi middleware authenticate agar mengecek field suspended:

/app/Http/Middleware/Authenticate.php

```
1 <?php
2 ...
3 class Authenticate
4 {
5 ...
6   public function handle($request, Closure $next)
7   {
8     if ($this->auth->guest()) {
9       if ($request->ajax()) {
10         return response('Unauthorized.', 401);
11       } else {
12         return redirect()->guest('auth/login');
13       }
14     }
15
16     $next_request = $next($request);
17
18     if ($request->user()->suspended) {
19       // logout user
20       $this->auth->logout();
21
22       // redirect to login page
23       return redirect()->guest('auth/login')
24         ->withErrors([
25           'email' => 'This account is suspended',
```

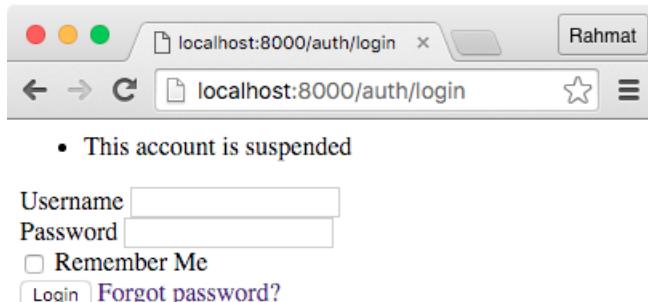
```
26          ]);  
27      }  
28  
29      return $next_request;  
30  }  
31 }
```

Berikut penjelasan untuk perubahan pada middleware ini:

- Baris 16: kita menyimpan instance dari `$request` selanjutnya setelah user terbukti berhasil login
- Baris 18-27: kita mengecek apakah field `suspended` bernilai true, jika ya kita logout user dan arahkan ke halaman login.
- Baris 29: kita melanjutkan request seperti biasa.

Setelah semua selesai, mari kita cek fitur ini. Jalankan `php artisan migrate:refresh` untuk mendapatkan field `suspended` di table `users`. Karena database sudah kita reset, silahkan buat user baru dari halaman register. Pastikan kita berhasil login. Jika berhasil, langsung logout.

Untuk mengetes fitur suspend, kita harus mengubah isi field `suspended` menjadi 1. Pada aplikasi di lapangan, biasanya kita membuat tampilan khusus untuk mengubah field ini. Pada tutorial ini, kita akan mengubah isinya secara manual di database. Jika sudah, cobalah login dengan user yang telah kita buat. Kita akan gagal dan mendapat tampilan seperti berikut:



Fitur suspended berhasil

Ada banyak cara untuk mendapatkan fitur ini. Silahkan Anda explore lebih lanjut.. :)

Mengakses User yang telah Login

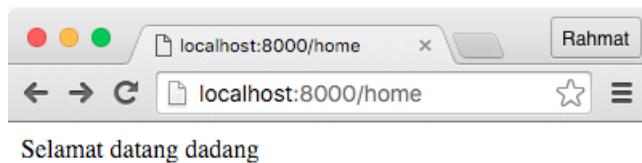
Kita dapat mengecek apakah user telah login dengan method `Auth::check()`. Jika user telah login, method ini akan menghasilkan nilai `true`. Untuk mendapatkan instance dari user yang sedang login, kita dapat menggunakan `Auth::user()` atau menggunakan method `user()` pada `Illuminate\Http\Request`.

Misalnya, mari kita coba tampilkan nama user ketika user telah login. Menggunakan cara pertama, kita tambahkan syntax tersebut pada route `/home` seperti berikut:

app/Http/routes.php

```
Route::get('/home', ['middleware' => 'auth', function () {
    return "Selamat datang " . Auth::user()->name;
}]);
```

Setelah login, ini tampilan yang akan kita dapatkan:



Menampilkan nama user

Menggunakan cara kedua, kita dapat mengarahkan route `/home` ke controller tertentu misalnya `HomeController`. Kita buat dulu controlernya dengan perintah `php artisan make:controller HomeController --plain`. Pada controller tersebut, kita isi syntax berikut:

app/Http/Controllers/HomeController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class HomeController extends Controller
{
    public function index(Request $request)
    {
        return "Selamat datang " . $request->user()->name;
    }
}
```

Pada route, kita ubah menjadi seperti ini:

app/Http/routes.php

```
Route::get('/home', ['middleware' => 'auth', 'uses' => 'HomeController@index']);
```

Silahkan dicek, pastikan output yang dihasilkan tetap sama.

Social Authentication dengan Socialite

Menggunakan social login misalnya dengan facebook atau twitter dapat kita lakukan dengan mudah menggunakan Socialite. Bahkan, secara native Socialite mendukung penggunaan OAuth untuk Facebook, Twitter, Github, Linkedin dan Bitbucket. Sebelum kita mulai coding, mari kita pelajari apa yang terjadi ketika kita melakukan authentikasi dengan menjadi client OAuth provider.

1. Kita membuat key, secret dan callback url di website provider. Berikut ini link untuk tiap provider yang didukung secara default oleh Laravel:
 - Facebook: <https://developers.facebook.com/apps>
 - Twitter: <https://apps.twitter.com>
 - Github: <https://github.com/settings/applications/new>
 - Linkedin: <https://www.linkedin.com/secure/developer>

- Bitbucket: <https://bitbucket.org/account/user/<username>/oauth-consumers/new>
2. Ketika user hendak login dengan provider yang sudah kita setup, dia akan diarahkan ke halaman provider tersebut dan menyetujui untuk memberikan akses.
 3. Ketika akses sudah diizinkan user akan diarahkan ke `callback url` yang sudah kita tentukan sambil membawa `access token` dari provider.
 4. Menggunakan `access token` yang didapatkan, webapp kita dapat mengakses data user yang berada pada provider tersebut.
 5. Data user pada provider tersebut dapat kita simpan sebagai data tambahan untuk user atau kita gunakan sebagai data utama untuk login pada webapp.

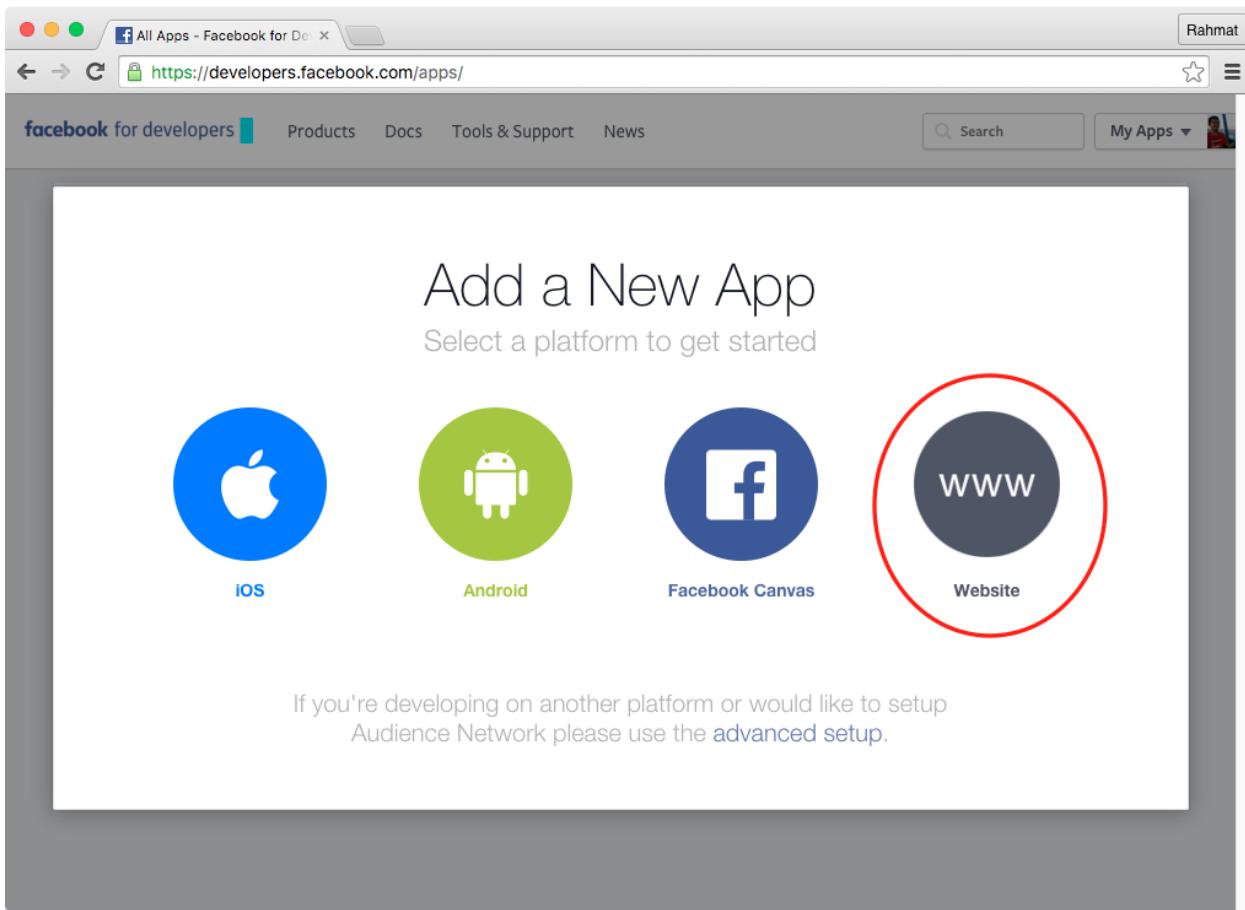
Pada sample ini, kita akan praktekan dengan membuat login menggunakan Facebook. Setelah kita dapatkan data user, kita akan menyimpannya pada table user. Silahkan ikuti langkah-langkah berikut:

1. Buat app baru di Facebook dengan mengunjungi <https://developers.facebook.com/apps>¹⁷¹:

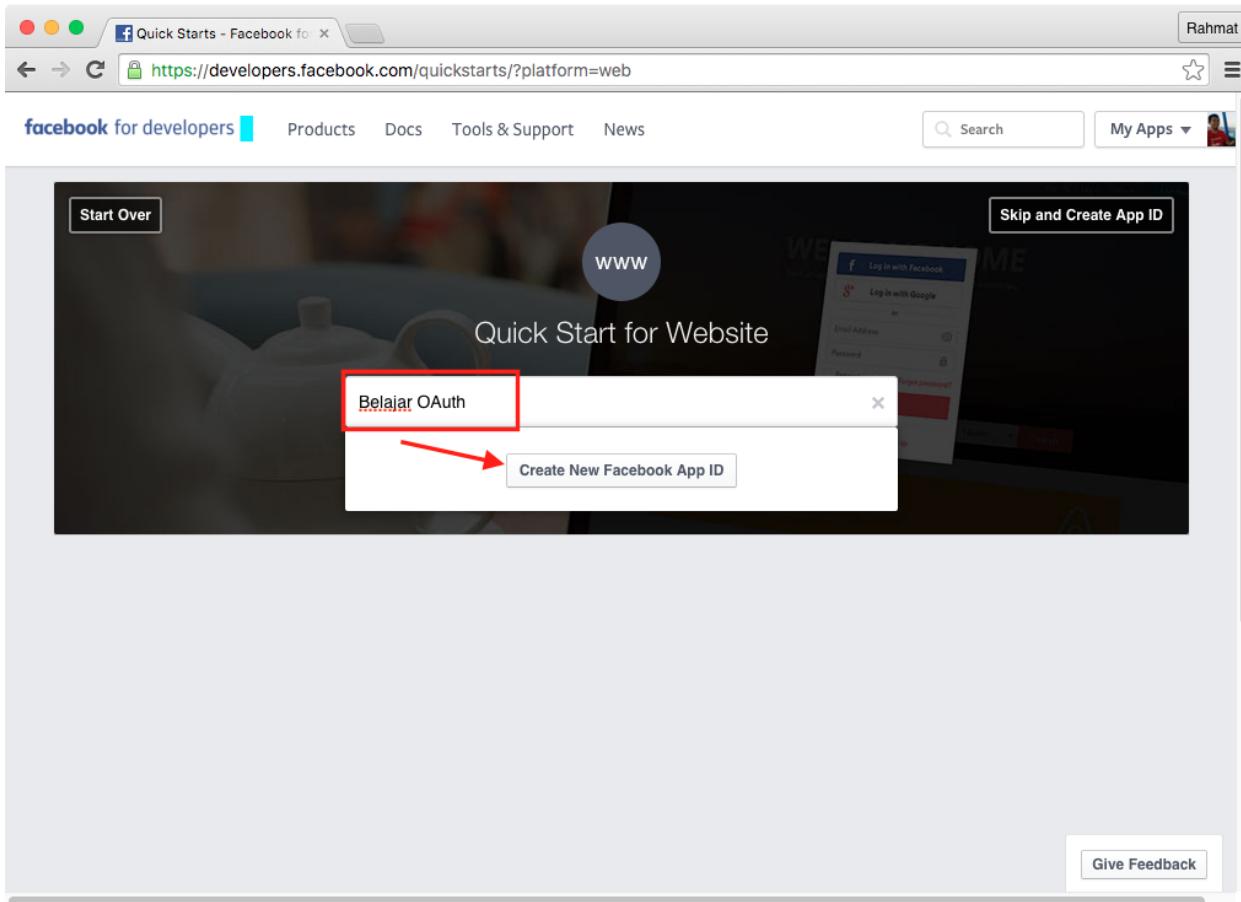
¹⁷¹ <https://developers.facebook.com/apps>

The screenshot shows a web browser window titled "All Apps - Facebook for Developers". The URL in the address bar is <https://developers.facebook.com/apps/>. The page content includes a search bar labeled "Search apps by title" and a green button labeled "+ Add a New App" which is highlighted with a red rectangular box. Below the search bar, there's a section titled "Developers" with links to various developer tools and resources. At the bottom of the page, there are links for "About", "Create Ad", "Careers", "Platform Policy", "Privacy Policy", "Cookies", and "Terms". On the right side, there's a copyright notice: "Facebook © 2015 English (US)".

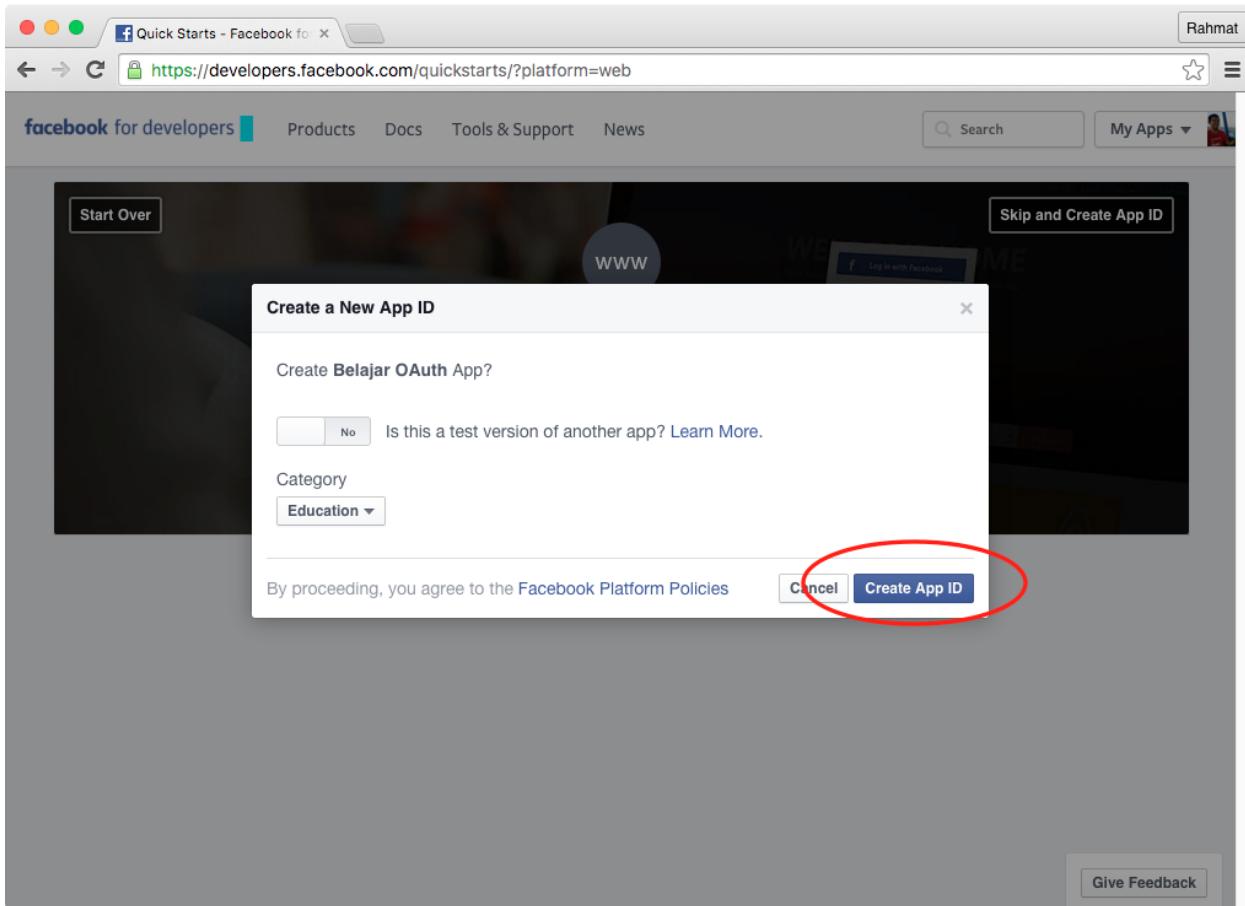
Pilih Add a New App



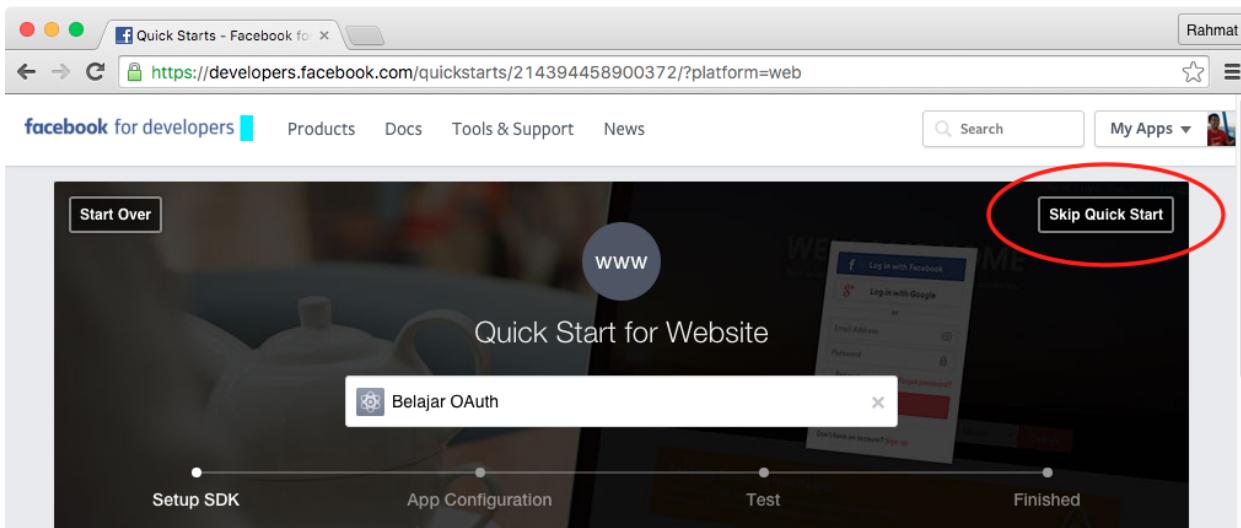
Pilih website



Masukan nama dan klik Create New Facebook App ID



Klik Create App ID



The screenshot shows the 'Quick Start for Website' interface. At the top right, there is a red circle highlighting the 'Skip Quick Start' button. Below it, a progress bar shows four steps: 'Setup SDK', 'App Configuration', 'Test', and 'Finished'. A tooltip 'Belajar OAuth' is visible over the 'App Configuration' step. The URL in the browser address bar is <https://developers.facebook.com/quickstarts/214394458900372/?platform=web>.

Setup the Facebook SDK for JavaScript

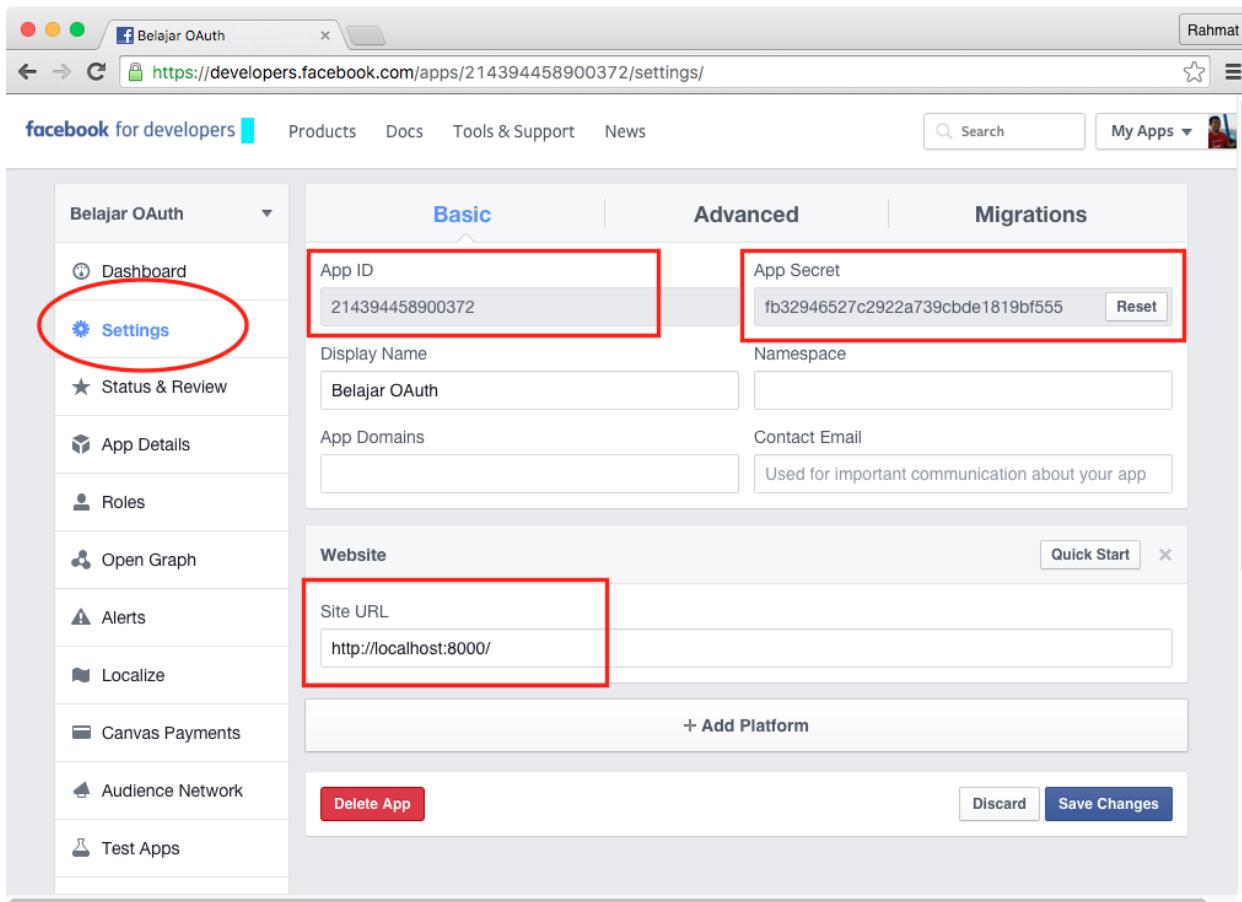
The following snippet of code will give the basic version of the SDK where the options are set to their most common defaults. You should insert it directly after the opening `<body>` tag on each page you want to load it:

```
<script>
  window.fbAsyncInit = function() {
    FB.init({
      appId      : '214394458900372',
      xfbml      : true,
      version   : 'v2.5'
    });
  };

```

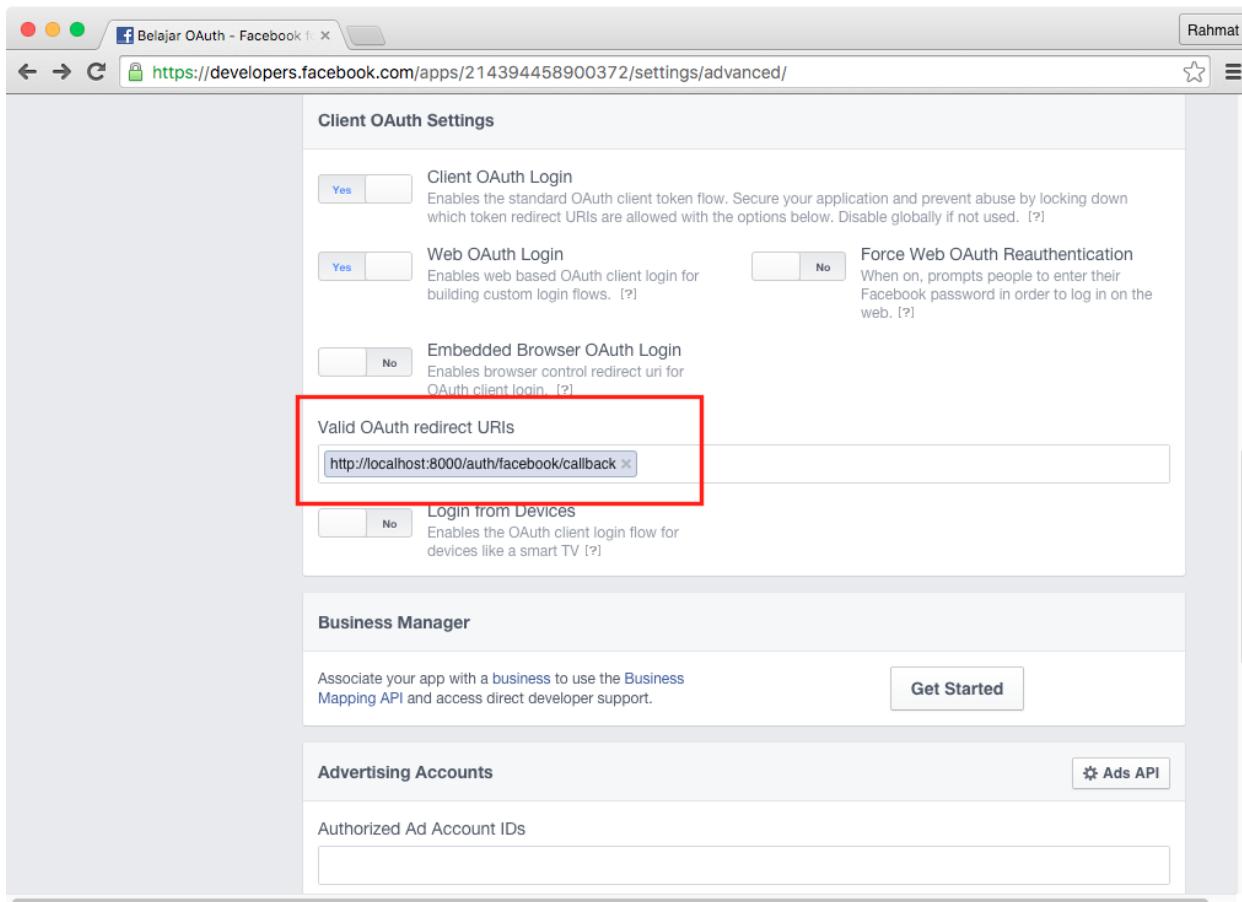
[Give Feedback](#)

Klik Skip Quick Start



Klik Opsi Settings, catat App ID, App Secret dan isi Site URL. Save.

Sesuaikan Site URL dengan URL yang digunakan selama development.



Buka tab Advanced, isi Valid OAuth redirect URIs. Save.

Isian ini kita isi dengan <alamat_webapp>/auth/facebook/callback. Akan kita bahas detailnya pada langkah-langkah berikutnya.

2. Selanjutnya, kita install Socialite. Download dengan perintah berikut di terminal:

```
composer require laravel/socialite
```

Setelah selesai download, tambahkan isian berikut pada config/app.php:

config/app.php

```
<?php

return [
    ...
    'providers' => [
        ...
        Laravel\Socialite\SocialiteServiceProvider::class,
    ],
]
```

```
'aliases' => [
    ...
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
],
];
```

3. Agar kita dapat menggunakan provider facebook untuk login, kita perlu menambah key dan secret yang telah kita dapatkan di langkah pertama ke konfigurasi di config/services.php. Format untuk konfigurasi ini yaitu:

```
'<provider>' => [
    'client_id' => 'xxx'
    'client_secret' => 'xxx'
    'redirect' => 'xxx'
]
```

Agar tetap mengikuti *best practices*, key dan secret dari facebook akan kita simpan pada file .env:

.env

```
FB_KEY=xxx
FB_SECRET=xxx
```

Sehingga, isian konfigurasi untuk provider facebook akan seperti berikut:

/config/services.php

```
<?php

return [
    ...
    'facebook' => [
        'client_id' => env('FB_KEY'),
        'client_secret' => env('FB_SECRET'),
        'redirect' => 'http://localhost:8000/auth/facebook/callback',
    ],
];
```

4. Untuk mengarahkan user ke facebook dan menerima redirect, kita akan membuatnya di AuthController:

/app/Http/Controllers/Auth/AuthController.php

```
<?php  
....  
class AuthController extends Controller  
{  
....  
public function redirectToProvider($provider)  
{  
    try {  
        return \Socialite::driver($provider)->redirect();  
    } catch (InvalidArgumentException $e) {  
        return abort(404, 'Driver tidak dikenal.');//  
    }  
}  
  
public function providerCallback($provider)  
{  
    try {  
        $account = \Socialite::driver($provider)->user();  
        $user = User::firstOrCreate(['provider' => $provider, 'provider_id' \  
=> $account->id]);  
        // update details  
        $user->name = $account->name;  
        $user->email = $account->email;  
        $user->avatar = $account->avatar;  
        $user->save();  
  
        \Auth::login($user, true);  
        return redirect('home');  
    } catch (InvalidArgumentException $e) {  
        return abort(404, 'Driver tidak dikenal.');//  
    } catch (\GuzzleHttp\Exception\ClientException $e) {  
        return redirect('auth/login');//  
    }  
}  
}
```

Pada file ini, kita membuat dua method. Method `redirectToProvider` akan mengarahkan kita ke halaman login di provider yang di support oleh Laravel. Didalamnya, kita menggunakan `try-catch` untuk menghindari error ketika user mengakses provider yang tidak dikenal.

Syntax `return \Socialite::driver($provider)->redirect();` yang akan mengarahkan kita ke halaman login. Terlihat disini, kita menggunakan variable `$provider` yang bisa kita set dari route. Ini salah satu cara yang bisa dipakai untuk mengarahkan user ke halaman login. Kelebihannya, kita tidak perlu membuat method baru untuk provider yang berbeda. Kekurangannya, jika konfigurasi untuk provider yang didukung tidak ditemui dia akan menampilkan error. Sebagian developer terkadang lebih suka membuatnya secara eksplisit dengan membuat method untuk setiap jenis provider, misalnya methodnya akan seperti ini:

```
public function redirectToFacebook()
{
    return \Socialite::driver('facebook')->redirect();
}
```

Pada method `providerCallback` kita akan menerima account user di provider tersebut. Method ini kita buat pula secara dinamis agar kita tidak perlu membuat method berbeda untuk provider yang lain. Pada method ini, kita akan menyimpan data user yang kita dapatkan. Dan melakukan login secara manual ke webapp. Menggunakan teknik ini, jika user mengubah datanya pada provider, maka data pada webapp kita akan otomatis berubah ketika dia login kembali.

Pada method ini juga kita menambahkan catch untuk `\GuzzleHttp\Exception\ClientException` yang kita gunakan ketika user klik `cancel` atau `deny` pada halaman login di provider.

Kita tambahkan route untuk kedua method ini:

/app/Http/routes.php

```
Route::get('auth/{provider}', 'Auth\AuthController@redirectToProvider');
Route::get('auth/{provider}/callback', 'Auth\AuthController@providerCallback');
```

5. Terlihat pada method `providerCallback` kita menggunakan field tambahan ketika menyimpan data user. `provider` untuk menentukan nama provider, `provider_id` ID user pada provider tersebut dan `avatar` URL ke profile photo user pada provider tersebut. Mari kita tambahkan field tersebut pada file migration:

/database/migrations/2014_10_12_000000_create_users_table.php

```
<?php
...
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('username');

            $table->string('name');
            $table->string('email');
            $table->string('password', 60);

            $table->string('provider');
            $table->string('provider_id')->unique()->nullable();
            $table->string('avatar');

            $table->boolean('suspended')->default(0);
            $table->rememberToken();
            $table->timestamps();
        });
    }
    ...
}
```

Setelah selesai, mari kita jalankan refresh `php artisan migrate:refresh`.

Kita juga perlu menambahkan field `provider` dan `provider_id` pada mass assignment di model User.

/app/User.php

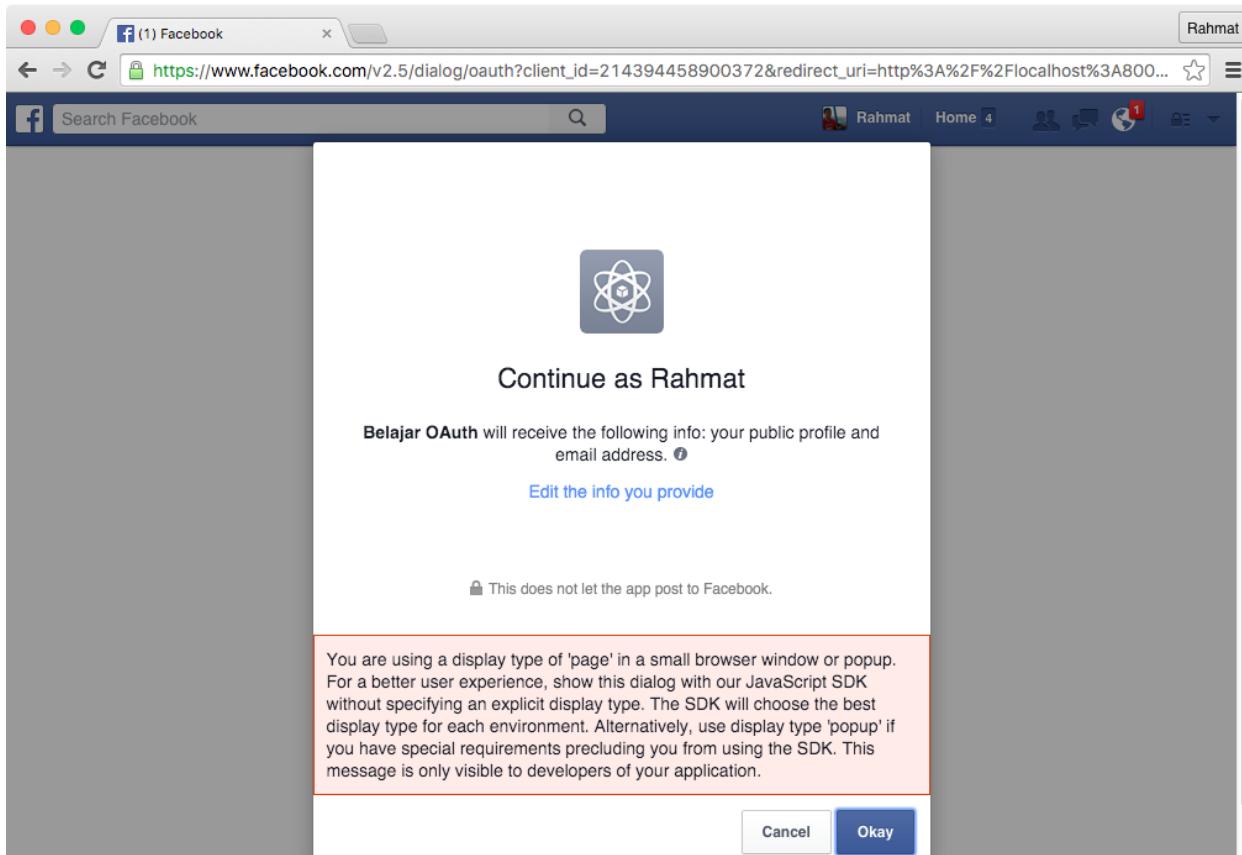
```
...
protected $fillable = ['name', 'username', 'email', 'password', 'provider', 'pr\
ovider_id'];
...
```

6. Kita tambahkan link untuk login dengan facebook di halaman login:

/resources/views/auth/login.blade.php

```
....  
<div>  
    <button type="submit">Login</button>  
    <a href="/password/email">Forgot password?</a>  
</div>  
<div>  
    <a href="/auth/facebook">Login dengan Facebook</a>  
</div>  
</form>
```

Sip, setelah semua konfigurasi selesai, kita dapat mencoba fitur login dengan facebook ini. Mari kita akses halaman /auth/login dan klik pada Login dengan facebook. Kita akan diarahkan ke facebook seperti berikut:

**Halaman authentikasi facebook**

Setelah kita klik Okay, kita akan otomatis login dan diarahkan ke halaman /home:



Berhasil login dengan facebook

Di database, data kita juga telah tersimpan:

A screenshot of MySQL Workbench showing the "homestead" database and the "users" table. The table has columns: id, username, name, email, password, provider, provider_id, avatar, suspended, and remember_token. There is one row with id 1, username "Rahmat Awaludin", name "Rahmat Awaludin", email "rahmat.awaludin@gmail.com", provider "facebook", provider_id "1", avatar "https://graph.facebook.com/v2.5...", suspended "0", and remember_token "aKXmyjPsuGi3qMJ1lqQegCdP6iZa2".

id	username	name	email	password	provider	provider_id	avatar	suspended	remember_token
1	Rahmat Awaludin	Rahmat Awaludin	rahmat.awaludin@gmail.com		facebook	1	https://graph.facebook.com/v2.5...	0	aKXmyjPsuGi3qMJ1lqQegCdP6iZa2

Data user dari facebook



Latihan

Cobalah buat login dengan twitter dan github. Langkah yang perlu dilakukan adalah menambah konfigurasi dan link login.

Membangun Hak Akses User (RBAC + ACL)

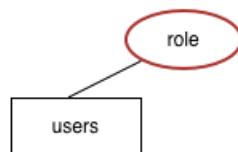
Dalam membangun webapp dengan banyak user, biasanya kita membutuhkan pengaturan hak akses, biasa dikenal dengan proses *authorization*. Contoh sederhana adalah website yang memiliki user admin dan user biasa. Tentunya, user biasa tidak memiliki akses sebanyak akses admin, misalnya menghapus user lain.

Contoh yang lebih kompleks adalah webapp yang memiliki banyak tipe user. Dimana tiap tipe memiliki hak akses yang beririsan dengan tipe yang lain.

Pada topik ini, kita akan coba membahas bagaimana menyelesaikan kedua masalah diatas. Tentunya, dengan teknik yang berbeda.

Memahami RBAC & ACL

RBAC merupakan singkatan dari [Role Based Access Control](#)¹⁷². Saya akan coba jelaskan secara sederhana. Menggunakan teknik ini, kita akan mengelompokkan user ke berbagai tipe user. Yang paling sederhana, tiap user memiliki satu jenis tipe (biasa disebut *role* atau *group*). Ketika user hendak melakukan akses aplikasi akan mengecek apakah user memiliki role yang dibutuhkan. Teknik ini cukup sederhana, kita cukup menambahkan kolom *role* (atau sejenisnya) ke table *Users* dan melakukan pengecekan terhadap isinya.



RBAC Tipe 1

Varian lain dari RBAC adalah user dapat memiliki banyak role, dimana table untuk menyimpan *role*, *user_role* dan *user* dipisahkan. Pengecekan yang dilakukan tetap sama, kita mengecek apakah user memiliki role yang dibutuhkan untuk melakukan suatu tindakan.

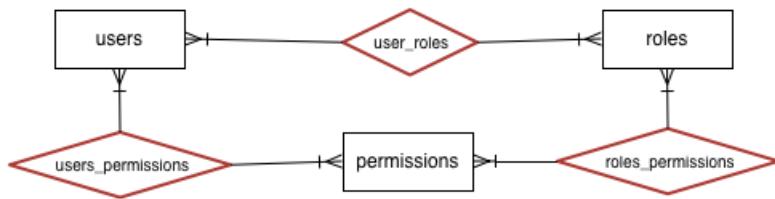


RBAC Tipe 2

Varian yang lebih kompleks adalah menambahkan *permission*. Cara kerjanya yaitu dengan membuat table *permissions* (atau sejenisnya) yang menyimpan data tindakan / resource. Dari table ini dapat di relasikan ke table *role* (*role_permission*) atau langsung

¹⁷²https://en.wikipedia.org/wiki/Role-based_access_control

ke `user` (`user_permission`). Ketika user hendak melakukan akses, pengecekan dapat dilakukan ke `role` atau `permission`.



RBAC Tipe 3

Berbeda dengan ACL. ACL (Access Control List¹⁷³) dibuat untuk menyederhanakan penggunaan RBAC. Di ACL, kita hanya memiliki `users`, `users_permission` dan `permission`. Tidak ada `role`. Ini dimaksudkan agar kita lebih fokus pada resource yang akan diakses, bukan level usernya.



ACL

Pada beberapa implementasi ACL, terkadang kita tidak membuat table khusus untuk menyimpan permission. Misalnya pada fitur Policy di Laravel yang akan kita bahas di pembahasan selanjutnya.

Dapat kita lihat disini, kedua istilah itu agak mirip dalam struktur database. Makanya, terkadang ada juga yang menamainya role based permission. Misalnya pada package [ZizacoEntrust](#)¹⁷⁴.

Memilih jenis implementasi hak akses

Dari sekian banyak jenis hak akses diatas, mana yang harus kita implementasi? Tentunya, ini kembali ke jenis aplikasi dan kebutuhan aplikasi itu sendiri. Berikut beberapa hal yang bisa dipertimbangkan.

Pilih RBAC Tipe 1

Jika:

- Kita hanya membutuhkan membedakan user & hak akses hanya dengan mengecek role user.
- Tiap user tidak akan memiliki lebih dari satu role.

¹⁷³https://en.wikipedia.org/wiki/Access_control_list

¹⁷⁴<https://github.com/Zizaco/entrust>

Pilih RBAC Tipe 2

Jika:

- User dapat memiliki banyak role
- Pengecekan hak akses hanya terjadi pada role

Pilih RBAC Tipe 3

Jika:

- User dapat memiliki banyak role
- Ada hak akses yang beririsan antar role

Pilih ACL

Jika:

- Kita tidak memerlukan pengelompokan user berdasarkan role
- Hak akses yang dibutuhkan belum jelas (requirement sistem belum final)

Membangun Hak Akses

Dalam contoh ini, kita akan membangun RBAC Tipe 1 dari nol. Untuk membuatnya kita akan menambahkan field `role` ke table `users` dan membuat middleware untuk membatasi akses berdasarkan role. Studi kasus disini, kita akan membuat role `user` untuk website semacam Event dimana user dapat mendaftar sebagai `organizer` (panitia) atau `participant` (peserta). Tentunya, halaman pembuatan event hanya dapat diakses oleh `organizer` dan halaman profile history event hanya dapat diakses oleh `participant`. Ikuti langkah berikut.

1. Tambahkan field `role` pada table `users`:

/database/migrations/2014_10_12_000000_create_users_table.php

```
<?php
...
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('username');
            $table->string('role');

            ...
        });
    }
}
```

Jalankan `php artisan migrate:refresh` untuk menggenerate struktur table yang baru.

Tentunya, kita harus menambahkan field ini pada model User agar dapat melakukan mass assignment:

/app/User.php

```
...
protected $fillable = [ 'name', 'username', 'email', 'password', 'provider', 'pr\
ovider_id', 'role'];
...
```

2. Ketika user register, kita akan izinkan dia memilih jenis user yang akan dia gunakan:

/resources/views/auth/register.blade.php

```
...
<div>
    Daftar sebagai
    <select name="role">
        <option value="" selected></option>
        <option value="organizer">Panitia</option>
        <option value="participant">Peserta</option>
    </select>
</div>
```

```
<div>
    <button type="submit">Register</button>
</div>
</form>
```

3. Di AuthController kita juga perlu melakukan sedikit perubahan agar dapat menerima field `role` yang kita kirim dari form.

/app/Http/Controllers/Auth/AuthController.php

```
<?php
...
class AuthController extends Controller
{
    protected function validator(array $data)
    {
        return Validator::make($data, [
            ...
            'role' => 'in:organizer,participant'
        ]);
    }

    protected function create(array $data)
    {
        return User::create([
            ...
            'role' => isset($data['role']) ? $data['role'] : 'participant'
        ]);
    }
}
```

Pada method `validator`, kita memvalidasi jika field `role` telah diisi harus berisi `organizer` atau `participant`.

Pada method `create`, kita menambahkan field `role` saat membuat user. Kita juga mengeset nilai default sebagai `participant` jika user tidak memilih `role` ketika dia mendaftar.

4. Untuk membatasi akses berdasarkan role, kita akan membuat middleware `role`. Jalankan perintah `php artisan make:middleware HasRole` untuk membuat template middleware. Ubahlah isinya seperti berikut:

/app/Http/Middleware/HasRole.php

```
<?php  
....  
class HasRole  
{  
    public function handle($request, Closure $next, $role)  
    {  
        if ($request->user()->role == $role) {  
            return $next($request);  
        }  
        return redirect('home')  
            ->with('message', 'Anda tidak memiliki akses untuk halaman tersebut\  
            .');  
    }  
}
```

Kita tambahkan pada /app/Http/Kernel.php:

/app/Http/Kernel.php

```
<?php  
....  
class Kernel extends HttpKernel  
{  
    ....  
    protected $routeMiddleware = [  
        ....  
        'role' => \App\Http\Middleware\HasRole::class,  
    ];  
}
```

Kita menggunakan middleware parameter pada middleware ini. Penggunaan middleware ini misalnya `role:organizer` jika hanya user yang memiliki role organizer yang diizinkan mengakses halaman yang dimaksud.

Ketika user tidak memiliki role yang diharapkan, kita me-*redirect* user ke halaman `home` dan membuat session variable dengan key `message` dan pesan `Anda tidak memiliki akses untuk halaman tersebut..`

5. Karena kita akan me-*redirect* user ke halaman `home`, mari kita ubah method `index` pada `HomeController` agar menampilkan pesan error tersebut. Untuk memudahkan, kita juga akan mengubah response ke view `/resources/views/home.blade.php`. Silahkan dibuat terlebih dahulu. Isi dengan konten seperti berikut:

/resources/views/home.blade.php

```
@if (session('message'))
    <p>{{ session('message') }}</p>
@endif

<p>Selamat datang {{ Auth::user()->name }}</p>
```

Pada view ini, kita menggunakan `session('message')` untuk mengecek isi session dengan key `message` dan menampilkan nilainya.

Pada `HomeController`, ini yang kita lakukan:

/app/Http/Controllers/HomeController.php

```
public function index(Request $request)
{
    return view('home');
```

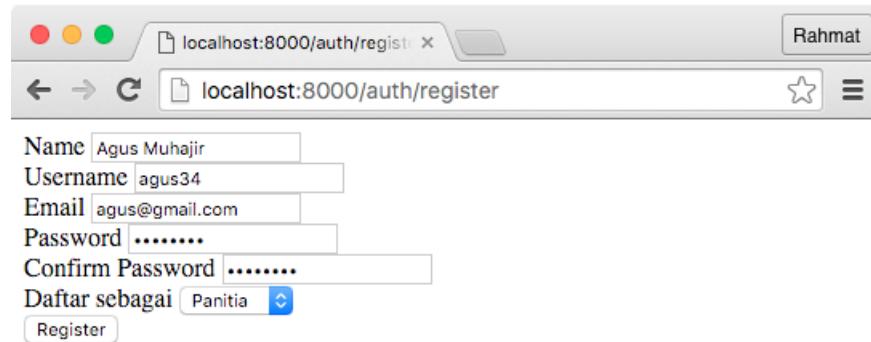
6. Untuk mengetes middleware, kita akan membuat dua route. event yang hanya bisa diakses oleh `organizer` dan `event-history` yang hanya bisa diakses oleh `participant`.

/app/Http/routes.php

```
Route::get('event', ['middleware' => ['auth', 'role:organizer'], function() {
    return "Berhasil mengakses halaman event";
}]);

Route::get('event-history', ['middleware' => ['auth', 'role:participant'], function() {
    return "Berhasil mengakses history event.";
}]);
```

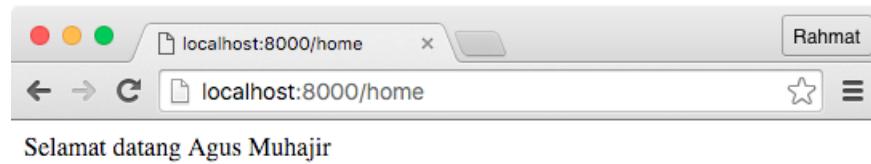
Sip. Setelah semua konfigurasi selesai kita dapat mengetes fitur ini. Cobalah buat user baru dengan role `organizer`:



Name Agus Muhajir
Username agus34
Email agus@gmail.com
Password
Confirm Password
Daftar sebagai Panitia
Register

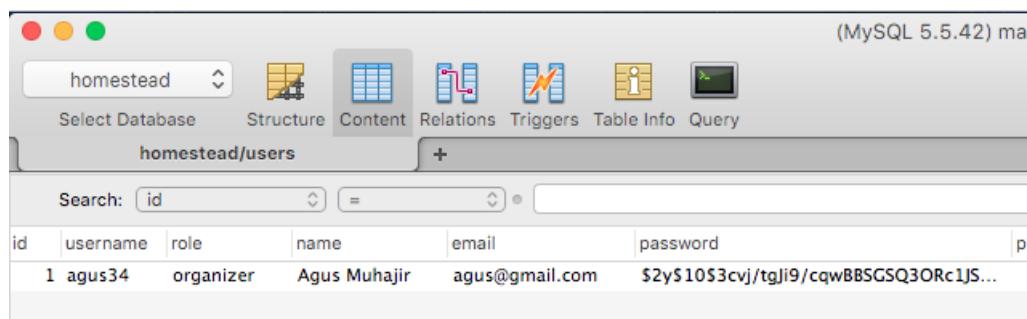
Membuat user dengan role organizer

Pastikan kita bisa login.



Berhasil login

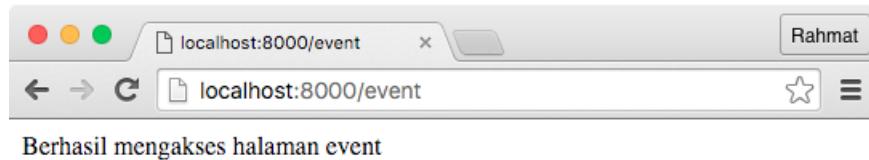
Pada database, pastikan role telah tersimpan.



id	username	role	name	email	password
1	agus34	organizer	Agus Muhajir	agus@gmail.com	\$2y\$10\$3cvj/tgJi9/cqwBBSGSQ3ORc1JS...

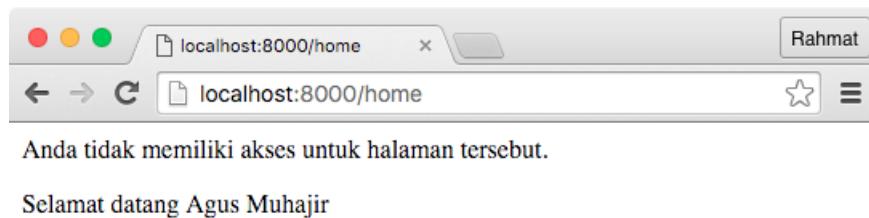
Data role berhasil disimpan

Cobalah mengakses halaman event, maka kita akan berhasil.



Berhasil mengakses halaman event

Sementara, jika kita mencoba mengakses event-history, kita akan gagal karena tidak memiliki role participant.



Gagal mengakses event-history

Sip. Sampai disini kita telah berhasil membuat sendiri RBAC. Pada beberapa aplikasi yang saya buat, penggunaan role seperti ini sudah sangat mencukupi. Tentunya, hal ini akan berbeda tergantung kebutuhan sistem.



Latihan

Setelah memahami proses pembuatan RBAC tipe 1, Cobalah buat RBAC tipe 2 & 3. Tambahkan studi kasus dimana seorang organizer dapat pula menjadi participant.

Menggunakan package untuk hak akses

Selain membangun dari nol, kita juga dapat menggunakan package untuk mengimplementasikan hak akses ini. Menggunakan package biasanya saya lakukan jika saya membutuhkan RBAC tipe 1 dan 2 atau ACL. Beberapa package yang saya rekomendasikan:

- Entrust¹⁷⁵
- Sentinel¹⁷⁶
- Verify¹⁷⁷

Saya sendiri agak jarang menggunakan package. Salah satu alasannya yaitu terkadang kita mendapatkan error ketika package tersebut tidak kompatible dengan Laravel. Dan, ketika ketika membuat *issue* di repositorynya membutuhkan waktu yang lama sampai kode tersebut di perbaiki.

Kebanyakan pemula ketika membutuhkan fitur seperti ini cenderung langsung mencari package, padahal fitur seperti ini cukup mudah untuk dibuat manual. Kelebihannya, kita jadi lebih paham alur hak akses di aplikasi kita. Sekiranya ada error, akan lebih mudah untuk diperbaiki. Saran saya, sebelum menggunakan package, coba develop sendiri fitur untuk hak akses ini. Jika hendak menggunakan ACL, saran saya sebaiknya menggunakan fitur Policy di Laravel.

Menggunakan Ability & Policy

Fitur Policy merupakan fitur baru di Laravel 5.1.11, jika versi yang Anda gunakan dibawah itu, ikuti panduan upgrade di <https://laravel.com/docs/5.1/upgrade#upgrade-5.1.11>. Untuk mengecek versi Laravel yang digunakan jalankan perintah `php artisan` (tanpa parameter).

Fitur authorisasi bawaan Laravel ini merupakan salah satu implementasi ACL yang *customizable* banget. Untuk menggunakan fitur ini, ada beberapa istilah yang harus kita pahami:

Memahami Ability

Ini adalah logic yang menentukan akses. Berbeda dengan menggunakan table `permission` dan `role` sederhana yang hanya memiliki nama, pada saat kita membuat `ability`, kita juga membuat logic untuk `ability` tersebut.

¹⁷⁵<https://github.com/Zizaco/entrust>

¹⁷⁶<https://cartalyst.com/manual/sentinel>

¹⁷⁷<https://github.com/Toddish/Verify>

Ini dimungkinkan karena setiap kali kita mendefinisikan *ability*, kita akan menerima instance User. Tentunya setelah kita mendapatkan instance dari User ini, kita dapat melakukan pengecekan ke attribute user atau relasi ke table lain untuk menentukan apakah user memiliki akses.

Selain instance dari User, kita juga dapat melakukan *passing instance* dari model lain. Misalnya, jika kita ingin membuat *ability* untuk mengecek akses user ke model *Event*, kita dapat menjadikan model *Event* sebagai paramater pada saat kita mendefinisikan *ability*.

Memahami Policy

Untuk aplikasi yang sederhana, biasanya *ability* ditulis pada `AuthServiceProvider` sebagai closure. Pada aplikasi yang besar, dimana banyak model yang terlibat biasanya digunakan *policy*. Secara sederhana, *policy* merupakan koleksi dari berbagai *ability* untuk sebuah resource (model).

Misalnya, Policy untuk model Post kita namakan `PostPolicy`, untuk model Event kita namakan `EventPolicy`, dst.

Memahami Gate

Gate adalah facade yang kita gunakan untuk melakukan pengecekan seorang user terhadap *ability*. Gate ini dapat diakses menggunakan facade `Gate`, model `User`, `controll`, `Form Request` dan `View`.

Case Study: Menggunakan Gate di Middleware

Mari kita lihat bagaimana kita akan mengimplementasi penggunaan `role` pada topik sebelumnya menggunakan Ability di Laravel. Untuk menggunakannya, logic yang akan kita gunakan tetap sama, yaitu mengecek isi field `role`. Perbedaannya, pengecekan untuk `role organizer` dan `participant` akan kita lakukan pada sebuah ability bernama `be-organizer` dan `be-participant`.

Untuk membuatnya, ikuti langkah berikut:

1. Buatlah ability `be-organizer` dan `be-participant` di `AuthServiceProvider`:

/app/Providers/AuthServiceProvider.php

```
<?php  
....  
class AuthServiceProvider extends ServiceProvider  
{  
    ....  
    public function boot(GateContract $gate)  
    {  
        parent::registerPolicies($gate);  
  
        $gate->define('be-organizer', function ($user) {  
            return $user->role == 'organizer';  
        });  
  
        $gate->define('be-participant', function ($user) {  
            return $user->role == 'participant';  
        });  
    }  
}
```

Cara yang kita lakukan disini adalah menggunakan closure untuk membuat ability baru pada method `boot`. Syntax dasarnya seperti berikut:

```
$gate->define('nama-ability', function ($user) {  
    // logic ability  
});
```

Setiap ability harus menghasilkan output berupa boolean.

Yang kita lakukan disini adalah memindahkan logic dari middleware `HasRole` ke ability.

2. Kini, pada middleware `HasRole`, kita dapat menggunakan `Gate` untuk mengecek ability ini:

/app/Http/Middleware/HasRole.php

```
<?php  
....  
class HasRole  
{  
....  
public function handle($request, Closure $next, $role)  
{  
    if ($request->user()->can('be-' . $role)) {  
        return $next($request);  
    }  
  
    return redirect('home')  
        ->with('message', 'Anda tidak memiliki akses untuk halaman tersebut\\  
.');
```

Teknik yang kita gunakan untuk mengakses Gate adalah dengan method `can` pada model User. Ini bisa kita lakukan karena secara default model User menggunakan trait `Illuminate\Contracts\Auth\Access\Authorizable`. Dapat di cek di `/app/User.php`.

Method `can` akan menghasilkan nilai `true` jika logic pada ability tervalidasi pada model User yang aktif. Kebalikan dari method `can` adalah method `cannot` yang akan menghasilkan nilai `true` jika logic pada ability tidak tervalidasi pada model User yang aktif.

Untuk mengecek ability ini, silahkan login sebagai seorang `organizer` kemudia coba akses `event-history`. Maka, kita akan gagal.



Berhasil mengimplementasikan role dengan ability

Case Study: Menggunakan Gate di View

Oke, pada contoh pertama, penggunaan ability terlihat membuat code kita kompleks. Tentunya, ada keuntungan yang semestinya kita dapatkan dari menggunakan ability ini. Yap, menggunakan middleware, rule untuk menentukan `role` user hanya bisa peroleh menggunakan middleware `role: ...` pada route (atau controller). Nah, menggunakan ability, logic tersebut bisa kita pakai di banyak tempat.

Kita ambil contoh, di halaman home ingin menampilkan link ke halaman `event` jika user adalah seorang `organizer` dan link ke halaman `event-history` jika ia seorang `participant`. Tanpa menggunakan ability, kita akan mengulang logic pada middleware `HasRole` misalnya seperti berikut:

`/resources/views/home.blade.php`

```
....  
@if (Auth::user()->role == 'organizer')  
    <p><a href="/event">Event</a></p>  
@endif  
  
@if (Auth::user()->role == 'participant')  
    <p><a href="/event-history">Event History</a></p>  
@endif
```

Tapi, coba kita renungkan sejenak beberapa masalah berikut:

- Bagaimana jika role `participant` ternyata tidak hanya dilihat dari user dengan field `role` berisi `participant` tetapi juga `sponsor`? Tanpa menggunakan ability, kita harus menduplikasi proses cek ini ke banyak tempat.

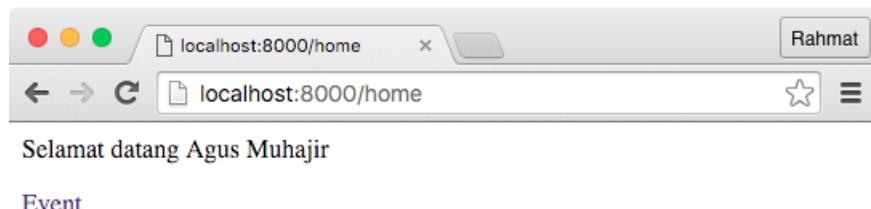
- Bagaimana jika aturan untuk menentukan role berubah? Misalnya, untuk mengeceknya kita harus menggunakan relasi table user ke table role (RBAC Tipe 2). Tanpa menggunakan ability, kita harus merubah semua logic tersebut di banyak tempat.

Menggunakan ability, kita dapat menggunakan helper @can di view seperti berikut:

/resources/views/home.blade.php

```
....  
@can('be-organizer')  
    <p><a href="/event">Event</a></p>  
@endcan  
  
@can('be-participant')  
    <p><a href="/event-history">Event History</a></p>  
@endcan
```

Kini, jika ada perubahan logic untuk menentukan role, hanya akan kita lakukan pada ability. Keren kan? :)



Menampilkan link secara dinamis

Case Study: Menggunakan Gate di Controller

Selain dari model dan view, kita juga dapat mengecek ability dari controller. Misalnya, kita ingin menampilkan halaman yang berbeda untuk URL /settings jika diakses oleh organizer atau participant. Tentunya, kita tidak bisa menggunakan middleware role:... untuk membatasi akses. Atau menggunakan @can di view untuk menampilkan view yang berbeda karena tidak efektif.

Cara pertama yang kita lakukan adalah menggunakan facade Gate untuk mengecek ability be-organizer atau be-participant, kemudian menampilkan view yang berbeda. Silahkan ikuti langkah berikut:

1. Buat route /settings yang mengarah ke HomeController@settings:

/app/Http/routes.php

```
Route::get('settings', ['middleware' => 'auth', 'uses' => 'HomeController@settings']);
```

2. Buatlah method settings pada HomeController dengan isi sebagai berikut:

/app/Http/Controllers/HomeController.php

```
<?php  
....  
use Gate;  
class HomeController extends Controller  
{  
    ....  
    public function settings()  
    {  
        if (Gate::allows('be-organizer')) {  
            return view('settings.organizer');  
        }  
  
        if (Gate::allows('be-participant')) {  
            return view('settings.participant');  
        }  
    }  
}
```

Pada method ini, kita akan menggunakan `Gate::allows()` untuk mengecek apakah user memiliki ability yang kita butuhkan. Sama dengan method `can` pada model User, method `allows()` akan menghasilkan `true` jika user yang sedang aktif dapat memvalidasi ability yang dibutuhkan. Kebalikan dari method `allows` adalah `denies` yang akan menghasilkan `true` jika user yang aktif tidak dapat memvalidasi ability yang dibutuhkan.

Kita bisa juga menggunakan facade Gate pada user lain, caranya kita gunakan method `forUser()`. Misalnya seperti berikut:

```
if (Gate::forUser($user)->allows('be-organizer')) {  
    //  
}
```

3. Kita buat view untuk kedua halaman settings ini:

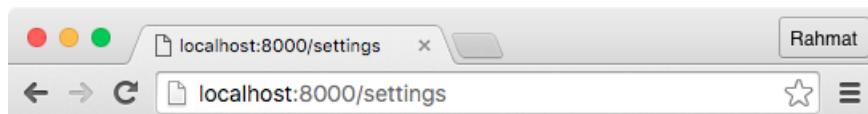
/resources/views/settings/organizer.blade.php

Halaman setting untuk organizer...

/resources/views/settings/participant.blade.php

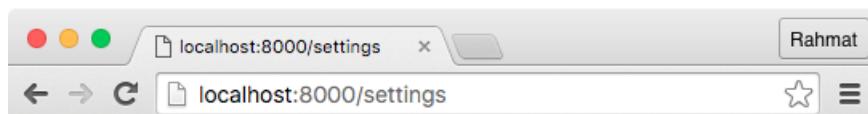
Halaman konfigurasi untuk participant...

Setelah semua konfigurasi selesai, kita dapat mencoba mengunjungi URL /settings. Konten dari halaman ini akan berbeda sesuai role user yang mengaksesnya:



Halaman setting untuk organizer...

Halaman settings untuk organizer



Halaman konfigurasi untuk participant...

Halaman settings untuk participant

Yang kedua adalah menggunakan `$this->authorize()`. Method `authorize()` bisa kita gunakan karena secara default App\Http\Controllers\Controller akan menggunakan trait Illuminate\Foundation\Auth\Access\AuthorizesRequests.

Method ini secara default akan menghasilkan `HTTPException` dengan status code 403 Not Authorized jika user yang sedang login tidak dapat memvalidasi ability yang dibutuhkan.

Mari kita buat fitur membership, dimana akan terdapat tiga jenis tipe membership: silver, gold dan platinum. Kemudian, kita membuat halaman premium yang hanya dapat diakses oleh user dengan membership gold / platinum.

Untuk mengisi membership, disini tentunya kita tidak akan menggunakan logic yang terlalu kompleks. Kita cukup mengizinkan User untuk memilih tipe membership dari opsi yang disediakan. Silahkan ikuti langkah berikut:

1. Kita perlu menambahkan field `membership` ke table User, tambahkan pada migration User (atau buat baru):

/database/migrations/2014_10_12_000000_create_users_table.php

```
<?php  
....  
class CreateUsersTable extends Migration  
{  
    ....  
    public function up()  
    {  
        Schema::create('users', function (Blueprint $table) {  
            ....  
            $table->boolean('suspended')->default(0);  
            $table->string('membership');  
            $table->rememberToken();  
            $table->timestamps();  
        });  
    }  
    ....  
}
```

Jalankan `php artisan migrate:refresh` untuk mendapatkan struktur table yang baru.

2. Kita perlu memodifikasi model User agar dapat melakukan mass assignment untuk field `membership`:

/app/User.php

```
protected $fillable = [ 'name', 'username', 'email', 'password',
    'provider', 'provider_id', 'role', 'membership'];
```

3. Pada form register, kita tambahkan field pilihan membership:

/resources/views/auth/register.blade.php

```
.....
<div>
    Tipe membership
    <select name="membership">
        <option value="" selected></option>
        <option value="silver">Silver</option>
        <option value="gold">Gold</option>
        <option value="platinum">Platinum</option>
    </select>
</div>

<div>
    <button type="submit">Register</button>
</div>
</form>
```

4. AuthController juga perlu kita modifikasi agar menerima isian membership:

/app/Http/Controllers/Auth/AuthController.php

```
<?php

.....
class AuthController extends Controller
{
    ...

    protected function validator(array $data)
    {
        return Validator::make($data, [
            ...
            'membership' => 'in:silver,gold,platinum'
        ]);
    }

    protected function create(array $data)
    {
        return User::create([
            ...
        ]);
    }
}
```

```

    ....
    'membership' => isset($data['membership']) ? $data['membership'] : \
'silver'
]);
}
.....
}

```

Secara default, kita akan menggunakan membership silver jika user tidak memilih tipe membership ketika mendaftar.

5. Kita buat ability `premium-access` untuk mengizinkan akses premium jika membership user gold / platinum:

/app/Providers/AuthServiceProvider.php

```

<?php
....
class AuthServiceProvider extends ServiceProvider
{
    ...
    public function boot(GateContract $gate)
    {
        ...
        $gate->define('premium-access', function ($user) {
            return $user->membership == 'gold' || $user->membership == 'platinu\
m';
        });
    }
}

```

6. Kita buat routenya...

/app/Http/routes.php

```

Route::get('premium', ['middleware' => ['auth'], 'uses' => 'HomeController@prem\
ium']);

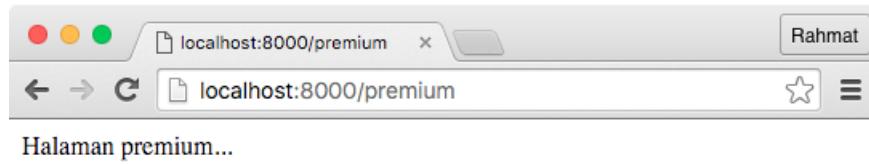
```

7. Dan, controlernya...

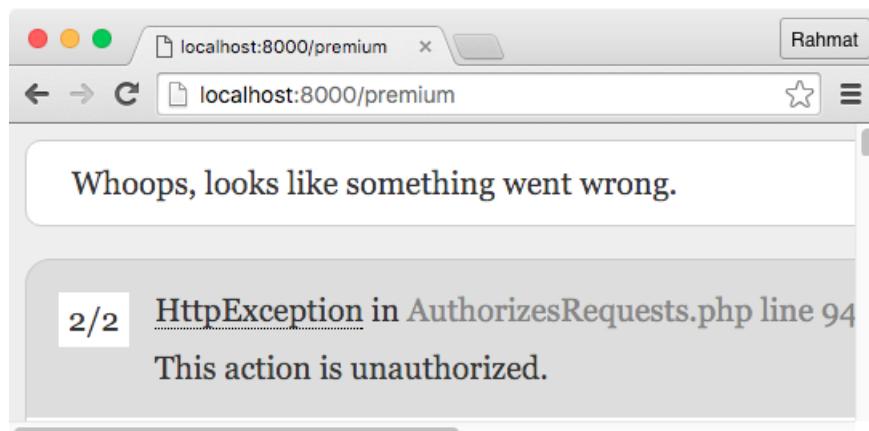
/app/Http/Controllers/HomeController.php

```
public function premium()
{
    $this->authorize('premium-access');
    return 'Halaman premium...';
}
```

Untuk mengeceknya, silahkan daftar sebagai user dengan membership gold / platinum. Cobalah akses halaman /premium, maka kita akan berhasil.

**Berhasil mengakses halaman premium dengan membership gold/platinum**

Jika kita mendaftar dengan membership silver, maka kita akan mendapatkan error seperti ini.

**Gagal mengakses halaman premium dengan membership silver**

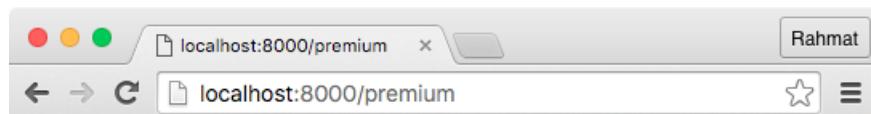
Error diatas merupakan error bawaan Laravel. Untuk mengkostumisasinya, kita dapat membuat view baru sesuai status code errornya di `/resources/views/errors/<code>.blade.php`.

Karena error diatas memiliki status code 403, maka kita buat view di `/resources/views/errors/403.blade.php`. Misalnya dengan isi sebagai berikut:

/resources/views/errors/403.blade.php

Anda tidak memiliki akses untuk halaman ini.

Bila di refresh, ini yang akan kita dapatkan dengan user yang memiliki membership silver:



Anda tidak memiliki akses untuk halaman ini.

Menggunakan custom view untuk menampilkan error 403

Case Study: Passing model lain ke ability

Salah satu kelebihan penggunaan ability adalah kita dapat mengecek hak akses berdasarkan kondisi tertentu dari sebuah model terhadap satu atau banyak model lainnya. Contohnya ketika melakukan proses editing sebuah resource, misalnya event. Kita beri nama ability `edit-event`. Ability ini bekerja dengan melakukan pengecekan apakah `organizer_id` pada model event apakah sama dengan `id` user yang sedang mengakses.

Mari kita demokan fitur ini dengan membuat codenya. Ikuti langkah berikut:

1. Kita belum memiliki model dan migration untuk Event. Mari kita buat dengan menjalankan perintah `php artisan make:model Event -m`. Setelah dijalankan, ubah isian file migration yang dihasilkan menjadi:

/database/migrations/XXXX_XX_XX_XXXXXX_create_events_table.php

```
<?php
...
class CreateEventsTable extends Migration
{
    ...
    public function up()
    {
        Schema::create('events', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('organizer_id');
            $table->string('name');
            $table->text('description');
            $table->string('location');
            $table->date('begin_date');
            $table->date('finish_date');
            $table->boolean('published')->default(0);
            $table->timestamps();
        });
    }
    ...
}
```

Table event ini kana memiliki beberapa field:

- id: ID dari event.
- organizer_id: ID dari user yang memiliki event ini.
- description: penjelasan singkat event.
- location: lokasi event diselenggarakan.
- begin_date & finish_date: Tanggal event dimulai dan berakhir.
- published: apakah event sudah dipublish.

Pada model yang dihasilkan, kita buat perubahan seperti berikut:

/app/Event.php

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Event extends Model
{
    protected $fillable = ['organizer_id', 'name', 'description', 'location', '\
```

```
begin_date', 'finish_date', 'published'];
```

```
protected $casts = [
    'published' => 'boolean'
];
}
```

Kita set mass assignment untuk semua field dan membuat field `published` otomatis dirubah menjadi boolean.

- Untuk memudahkan testing, kita tidak akan membuat form untuk menambah event. Tapi, kita akan membuat event berikut sample organizer dan participant dengan seeder. Silahkan ubah `DatabaseSeeder` menjadi seperti berikut:

/database/seeds/DatabaseSeeder.php

```
<?php
...
class DatabaseSeeder extends Seeder
{
    ...
    public function run()
    {
        Model::unguard();

        // sample organizer
        $jajang = App\User::create([
            'name' => 'Jajang Sopandi',
            'username' => 'jajang',
            'email' => 'jajang@gmail.com',
            'password' => bcrypt('rahasia'),
            'role' => 'organizer',
            'membership' => 'gold'
        ]);

        $ucok = App\User::create([
            'name' => 'Ucok Prayogo',
            'username' => 'ucok',
            'email' => 'ucok@gmail.com',
            'password' => bcrypt('rahasia'),
            'role' => 'organizer',
            'membership' => 'platinum'
        ]);
    }
}
```

```

    // sample participant
    $beni = App\User::create([
        'name' => 'Beni Wijaya',
        'username' => 'beni',
        'email' => 'beni@gmail.com',
        'password' => bcrypt('rahasia'),
        'role' => 'participant',
        'membership' => 'gold'
    ]);

    // sample event
    $meetupJS = App\Event::create([
        'organizer_id' => $jajang->id,
        'name' => 'Meetup JS Jakarta',
        'description' => 'Kumpul bareng developer JS',
        'location' => 'Balai Kartini',
        'begin_date' => '2016-03-10',
        'finish_date' => '2016-03-11',
        'published' => 1
    ]);

    $meetupLaravel = App\Event::create([
        'organizer_id' => $ucok->id,
        'name' => 'Meetup Laravel Bandung',
        'description' => 'Kumpul bareng developer Laravel',
        'location' => 'Sabuga',
        'begin_date' => '2016-04-02',
        'finish_date' => '2016-04-05',
        'published' => 0
    ]);

    Model::reguard();
}

}

```

Cukup panjang, berikut penjelasan yang kita buat disini.

- 2 orang organizer, Jajang dan Ucok.
- 1 orang participant, Beni.
- Event Meetup JS dengan organizer Jajang dan sudah dipublish.
- Event Meetup Laravel dengan organizer Ucok tapi belum dipublish.

3. Mari kita buat ability edit-event:

/app/Providers/AuthServiceProvider.php

```
<?php
...
class AuthServiceProvider extends ServiceProvider
{
    ...
    public function boot(GateContract $gate)
    {
        ...
        $gate->define('edit-event', function($user, $event) {
            return $user->id == $event->organizer_id;
        });
    }
}
```

Terlihat pada syntax diatas, kita melakukan passing \$event dan mengecek apakah `id` dari user yang aktif sesuai dengan `organizer_id` pada event.

- Untuk mengecek ability ini, kita akan menggunakan url `/edit-event/{id}` yang kita arahkan ke method `editEvent` di `HomeController`:

/app/Http/routes.php

```
Route::get('edit-event/{id}', 'HomeController@editEvent');
```

/app/Http/Controllers/HomeController.php

```
...
public function editEvent($id)
{
    $event = \App\Event::findOrFail($id);
    $this->authorize('edit-event', $event);
    return "Anda sedang mengakses halaman edit event " . $event->name;
}
...
```

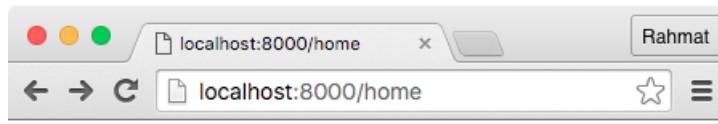
Terlihat disini, ketika kita melakukan pengecekan kita harus mengirim instance dari Event. Ketika sudah berhasil, kita menampilkan tulisan "Anda sedang mengakses halaman edit event xxx".

- Untuk memudahkan mengunjungi halaman ini, mari kita buat list event di halaman `/home` dan menampilkan link ke halaman edit.

/resources/views/home.blade.php

```
....  
<h3>Semua Event</h3>  
 @foreach (App\Event::get() as $event)  
   <p><strong>Event: {{ $event->name }}</strong></p>  
   <p>{{ $event->description }}</p>  
   <p>Tempat/Waktu: {{ $event->location }}, {{ $event->begin_date }} - {{ $event->finish_date }}</p>  
   @can ('be-organizer')  
     <a href="/edit-event/{{ $event->id }}">Edit Event</a>  
   @endcan  
 @endforeach
```

Sip. Mari kita coba fitur ini, pertama login lah sebagai Jajang. Maka akan muncul daftar semua event pada halaman /home:



The screenshot shows a web browser window with the address bar displaying "localhost:8000/home". The title bar has a button labeled "Rahmat". The main content area displays a welcome message "Selamat datang Jajang Sopyandi" followed by a link "Event". Below this, there is a section titled "Semua Event" containing two entries. Each entry includes the event name, description, location and dates, and an "Edit Event" link. The first entry is for "Meetup JS Jakarta" and the second for "Meetup Laravel Bandung".

Selamat datang Jajang Sopyandi

Event

Semua Event

Event: Meetup JS Jakarta

Kumpul bareng developer JS

Tempat/Waktu: Balai Kartini, 2016-03-10 - 2016-03-11

Edit Event

Event: Meetup Laravel Bandung

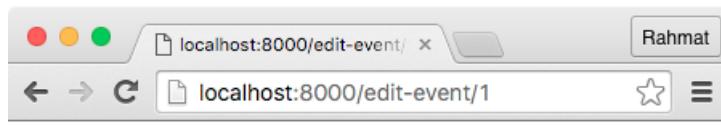
Kumpul bareng developer Laravel

Tempat/Waktu: Sabuga, 2016-04-02 - 2016-04-05

Edit Event

Organizer melihat semua event

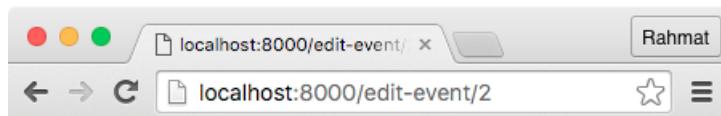
Cobalah klik pada link "Edit event" pada event Meetup JS Jakarta, maka kita akan berhasil.



Anda sedang mengakses halaman edit event Meetup JS Jakarta

Berhasil mengunjungi halaman edit event

Tapi, ketika kita mencoba mengunjungi halaman edit untuk event Meetup Larvel Bandung, maka kita akan gagal. Ini terjadi karena `organizer_id` untuk event ini adalah Ucok, bukan Jajang.



Anda tidak memiliki akses untuk halaman ini.

Gagal mengunjungi halaman edit

Sip. Pada contoh barusan kita menggunakan relasi dari user ke event. Bisa juga ability ini kita cek tanpa relasi sama sekali. Contohnya, kita memiliki ability `join-event`, ability ini berkerja dengan mengecek apakah seorang user seorang `participant` dan sebuah event memiliki status `published`.

Mari kita coba buat. Ikuti langkah berikut.

1. Buat ability `join-event`:

/app/Providers/AuthServiceProvider.php

```
<?php
...
class AuthServiceProvider extends ServiceProvider
{
    ...
    public function boot(GateContract $gate)
    {
        ...
        $gate->define('join-event', function($user, $event) {
            return $user->role == 'participant' && $event->published;
        });
    }
}
```

2. Kita cek dengan membuat url /join-event/{id} dan kita arahkan ke method joinEvent pada HomeController:

/app/Http/routes.php

```
Route::get('join-event/{id}', 'HomeController@joinEvent');
```

/app/Http/Controllers/HomeController.php

```
public function joinEvent($id)
{
    $event = \App\Event::findOrFail($id);
    $this->authorize('join-event', $event);
    return "Anda sedang mengakses halaman join event " . $event->name;
}
```

Sama seperti contoh sebelumnya, pada contoh ini kita juga melakukan passing event ketika mengecek sebuah ability. Ketika sudah berhasil, maka akan tampil tulisan “Anda sedang mengakses halaman join event xxx”.

3. Kita tambahkan link untuk join event pada halaman /home:

/resources/views/home.blade.php

```
....  
<h3>Semua Event</h3>  
@foreach (App\Event::get() as $event)  
....  
@can ('be-participant')  
    <a href="/join-event/{{ $event->id }}">Join Event</a>  
@endcan  
@endforeach
```

Sip. Mari kita coba dengan login sebagai Beni. Ini yang akan muncul pada halaman /home kita.

The screenshot shows a web browser window with the URL `localhost:8000/home`. The title bar says "Rahmat". The page content includes:

Selamat datang Beni Wijaya

[Event History](#)

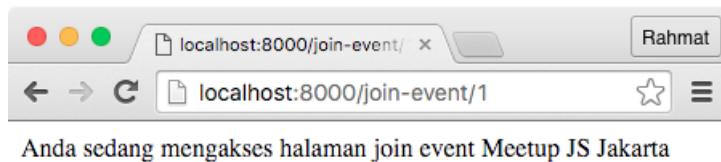
Semua Event

Event: Meetup JS Jakarta
Kumpul bareng developer JS
Tempat/Waktu: Balai Kartini, 2016-03-10 - 2016-03-11
[Join Event](#)

Event: Meetup Laravel Bandung
Kumpul bareng developer Laravel
Tempat/Waktu: Sabuga, 2016-04-02 - 2016-04-05
[Join Event](#)

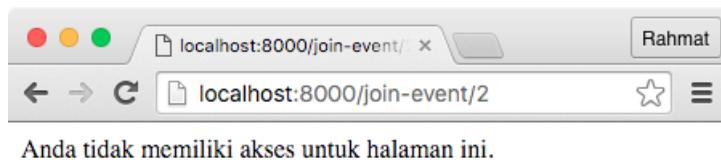
Participant melihat semua event

Ketika kita coba join ke event Meetup JS Jakarta, maka kita akan berhasil.



Berhasil mengunjungi halaman join event

Tapi, ketika mencoba join ke event Meetup Laravel Bandung, kita akan gagal. Ini terjadi karena event tersebut belum dipublish (field publish berisi 0).



Gagal mengakses halaman join

Case Study: Menggunakan Policy untuk mengelompokan Ability

Setelah kita paham memahami penggunaan ability, biasanya jumlah ability yang kita buat akan banyak. Terlebih ketika kita memahami bahwa kita dapat melakukan passing instance dari model lain ke ability, biasanya banyak hak akses terhadap model kita buat dalam bentuk ability. Makin lama, isi dari AuthServiceProvider akan semakin besar jika kita mendefinisikan semua ability pada file tersebut.

Disinilah kita harus mulai menggunakan Policy. Menggunakan teknik policy, ability yang berhubungan dengan sebuah model tertentu akan kita simpan pada sebuah class khusus. Pada contoh sebelumnya, kita memiliki 2 proses yang bisa dilakukan oleh seorang user terhadap Event, edit dan join. Pada aplikasi di lapangan proses ini biasanya banyak, tergantung aturan bisnis yang dimiliki oleh aplikasi tersebut.

Mari kita kelompokan 2 ability pada model Event pada policy bernama EventPolicy. Ikuti langkah berikut:

1. Buatlah EventPolicy dengan menjalankan perintah `php artisan make:policy EventPolicy`. Setelah dijalankan akan muncul file `/app/Policies/EventPolicy.php`.
2. Ubah isian file tadi menjadi:

/app/Policies/EventPolicy.php

```
<?php

namespace App\Policies;

use Illuminate\Auth\Access\HandlesAuthorization;
use App\Event;
use App\User;

class EventPolicy
{
    use HandlesAuthorization;

    public function update(User $user, Event $event)
    {
        return $user->id == $event->organizer_id;
    }

    public function join(User $user, Event $event)
    {
        return $user->role == 'participant' && $event->published;
    }
}
```

Yang kita lakukan disini adalah memindahkan logic untuk membuat ability dari AuthServiceProvider ke file ini. Berbeda dengan AuthServiceProvider, pada file ini kita mendefinisikan ability baru dengan nama method.

Disini juga kita menggunakan nama method `update`, bukan `edit-event` sebagaimana yang kita gunakan pada teknik sebelumnya. Ini saya lakukan agar kita lebih mudah dalam memahami proses CRUD (U=Update). Dan sudah sangat umum rule untuk proses CRUD pada sebuah model didefinisikan pada file polisinya.

3. Kita hapus ability pada AuthServiceProvider:

/app/Providers/AuthServiceProvider.php

```

<?php
...
class AuthServiceProvider extends ServiceProvider
{
    ...
    public function boot(GateContract $gate)
    {
        ...
        $gate->define('edit-event', function($user, $event) {
            return $user->id == $event->organizer_id;
        });

        $gate->define('join-event', function($user, $event) {
            return $user->role == 'participant' && $event->published;
        });
    }
}

```

4. Ubah juga nama ability yang kita gunakan pada HomeController:

/app/Http/Controllers/HomeController.php

```

...
public function editEvent($id)
{
    $event = \App\Event::findOrFail($id);
    $this->authorize('update', $event);

    ...
}

public function joinEvent($id)
{
    $event = \App\Event::findOrFail($id);
    $this->authorize('join', $event);

    ...
}

```

5. Ketika kita membuat Policy, kita perlu mendaftarkan policy tersebut ke AuthServiceProvider pada variable \$policies, formatnya adalah path_class => path_policy:

/app/Providers/AuthServiceProvider.php

```
<?php
...
class AuthServiceProvider extends ServiceProvider
{
    ...
protected $policies = [
    \App\Event::class => \App\Policies\EventPolicy::class
];
...
}
```

Sip. Setelah semua konfigurasi selesai, silahkan dicoba untuk mengakses logika yang kita gunakan pada tahap sebelumnya. Pastikan semuanya berhasil.

Case Study: Menggunakan nama ability yang sama dengan policy

Ketika kita memiliki banyak model, tanpa menggunakan policy, mungkin kita akan menggunakan nama ability edit-event, edit-post, edit-organization, dll untuk setiap ability untuk model tersebut. Menggunakan policy, kita tidak perlu melakukan hal seperti itu. Meskipun kita menggunakan nama ability yang sama (misalnya update), Laravel akan mengetahui method update pada policy yang mana yang harus dia pakai, tergantung model yang kita passing.

Mari kita demokan dengan membuat model Organization dan memuat ability update, sama dengan yang kita gunakan pada ability untuk model Event. Ikuti langkah berikut:

1. Buatlah model dan migration untuk Organization dengan menjalankan perintah `php artisan make:model Organization -m`. Pada file migration yang dihasilkan, isi dengan syntax berikut:

/database/migrations/XXXX_XX_XX_XXXXXX_create_organizations_table.php

```
<?php
...
class CreateOrganizationsTable extends Migration
{
    ...
public function up()
{
    Schema::create('organizations', function (Blueprint $table) {
```

```

        $table->increments('id');
        $table->string('name');
        $table->integer('admin_id');
        $table->timestamps();
    });
}

...
}

```

Pada model Organization, kita buat agar bisa mass assignment untuk fileld `name` dan `admin_id`.

/app/Organization.php

```

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Organization extends Model
{
    protected $fillable = [ 'name', 'organization_id'];

    public function admin()
    {
        return $this->belongsTo( 'App\User', 'admin_id' );
    }
}

```

Untuk memudahkan, kita juga menambah method `admin` untuk mengakses user yang menjadi admin dari organisasi ini.

2. Kita buat sample organization dengan seeder:

/database/seeds/DatabaseSeeder.php

```

<?php

...

class DatabaseSeeder extends Seeder
{
    ...

    public function run()
    {
        ...
    }
}

```

```

    // sample organization
    $artisanBdg = App\Organization::create([
        'name' => 'Artisan Bandung',
        'admin_id' => $ucok->id
    ]);

    Model::reguard();
}
}

```

Terlihat disini, kita membuat sample organisasi dengan nama “Artisan Bandung” dan Ucok sebagai adminnya. Jalankan `php artisan migrate:refresh --seed` untuk mendapatkan table organizations dan sample datanya.

3. Mari kita buat policy untuk Organization, jalankan `php artisan make:policy OrganizationPolicy`. Pada file yang digenerate, isi dengan syntax berikut:

/app/Policies/OrganizationPolicy.php

```

<?php

namespace App\Policies;

use Illuminate\Auth\Access\HandlesAuthorization;
use App\User;
use App\Organization;

class OrganizationPolicy
{
    use HandlesAuthorization;

    public function update(User $user, Organization $organization)
    {
        return $user->id == $organization->admin_id;
    }
}

```

Seperti EventPolicy, pada OrganizationPolicy kita membuat ability `update`. Bedanya, disini kita akan membandingkan `admin_id` dengan ID user.

Kita daftarkan pada AuthServiceProvider:

/app/Providers/AuthServiceProvider.php

```
<?php
...
class AuthServiceProvider extends ServiceProvider
{
    ...
protected $policies = [
    ...
\App\Organization::class => \App\Policies\OrganizationPolicy::class,
];
...
}
```

- Untuk mengecek Policy ini, kita akan menggunakan url /edit-organization/{id} untuk menampilkan halaman bahwa kita sedang mengedit organisasi. URL ini akan kita arahkan ke method editOrganization pada HomeController:

/app/Http/routes.php

```
Route::get('edit-organization/{id}', 'HomeController@editOrganization');
```

/app/Http/Controllers/HomeController.php

```
public function editOrganization($id)
{
    $organization = \App\Organization::findOrFail($id);
    $this->authorize('update', $organization);
    return "Anda sedang mengakses halaman edit organisasi " . $organization->na\
me;
```

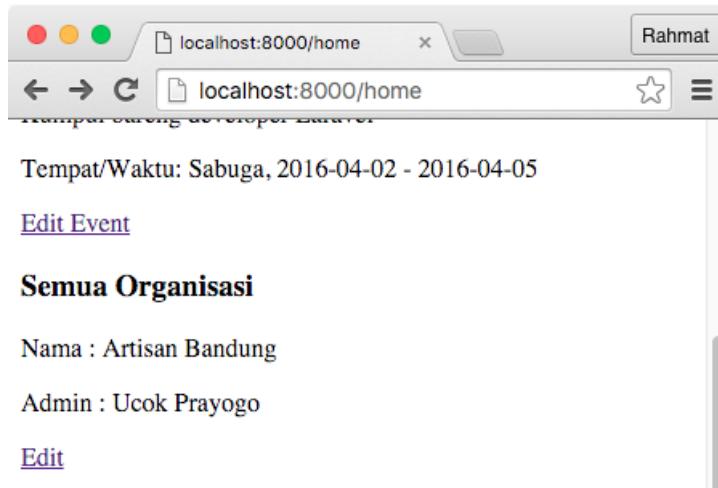
Terlihat disini, kita tetap menggunakan ability update sebagaimana kita menggunakan ability tersebut pada Event. Perbedaannya, yang kita passing disini adalah instance dari Organization, bukan Event.

- Untuk memudahkan testing, mari kita tampilkan daftar organisasi berikut adminnya pada halaman /home:

/resources/views/home.blade.php

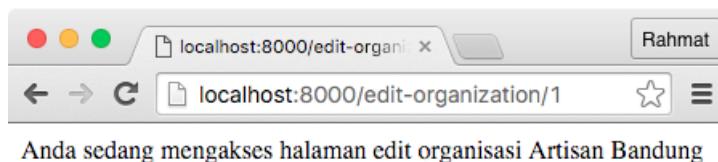
```
....  
<h3>Semua Organisasi</h3>  
 @foreach (App\Organization::get() as $organization)  
     <p>Nama : {{ $organization->name }}</p>  
     <p>Admin : {{ $organization->admin->name }}</p>  
     <p><a href="/edit-organization/{{ $organization->id }}">Edit</a></p>  
 @endforeach
```

Sip. Setelah semua siap, mari kita test. Cobalah login sebagai Ucok, maka akan tampil daftar organisasi pada halaman /home:



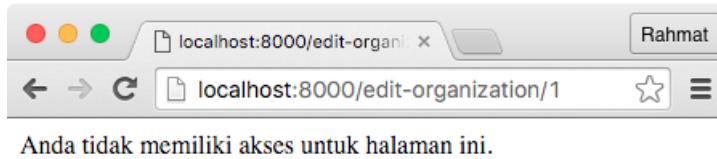
Organisasi tampil di /home

Jika klik "Edit" maka akan muncul tampilan seperti berikut:



Berhasil menampilkan halaman edit organisasi

Tapi, bila kita login sebagai Jajang, maka dia tidak dapat mengakses halaman organisasi:



Gagal mengakses halaman edit organisasi

Sebagaimana pepatah mengatakan, banyak jalan menuju romawi. Begitupun dengan authorisasi, banyak cara yang bisa kita lakukan untuk mencapainya. Prinsip saya ketika membangun aplikasi adalah menghindari penggunaan *third party library* se-bisa mungkin jika ada fitur framework yang mengakomodasi fungsionalitas yang diinginkan. Menggunakan fitur bawaan framework, memastikan kode kita selalu kompatibel dengan framework dan akan memudahkan untuk memperbaiki bug.

JWT (JSON Web Token) Authentication

Dalam membangun API, sangat mungkin salah satu requirementnya adalah login dengan menggunakan token. Salah satu teknik yang populer untuk login dengan token ini adalah JWT atau JSON Web Token. Secara singkat, cara kerja JWT adalah seperti berikut:

1. Client merequest token dengan mengirimkan *credential* (username dan password).
2. Server memvalidasi credential yang dikirim dan, jika valid, mengirimkan token yang digenerate dengan algoritma tertentu. Token ini yang akan digunakan oleh client untuk request selanjutnya. Token ini, biasanya memiliki waktu expire. Jadi, ketika token sudah expire, client harus meminta token yang baru.
3. Client menyimpan token yang diterima, dan untuk setiap request selanjutnya menambahkan token tersebut pada headernya.

Salah satu hal yang mesti kita pertimbangkan adalah ketika kita menggunakan JWT, server tidak dapat melakukan expire kapanpun sebagaimana kalau kita menggunakan Session. Token yang sudah dikirim, hanya akan expire jika waktu expire pada token tersebut sudah tercapai.

Untuk memahami bagaimana sebuah token JWT digenerate, silahkan baca artikel berikut¹⁷⁸.

Di Laravel, sampai saat buku ini ditulis, tidak ada implementasi native login dengan token. Untuk menggunakan JWT Auth kita dapat menggunakan package [tymon/jwt-auth](#)¹⁷⁹. Mari kita awali dengan menginstall package ini.

1. Install `tymon/jwt-auth` dengan menjalankan perintah `composer require tymon/jwt-auth`. Pada saat saya menulis buku ini, versi yang terinstall adalah versi 0.5.6. Seharusnya, selama masih 0.5.* maka cara penggunaannya tidak akan berbeda.
2. Kita perlu menambahkan beberapa konfigurasi pada `config/app.php`:

/config/app.php

```
<?php

return [
    ...
    'providers' => [
        ...
        Tymon\JWTAuth\Providers\JWTAuthServiceProvider::class,
    ],
    'aliases' => [
        ...
        'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
    ],
];
```

Disini kita menambahkan provider `JWTAuthServiceProvider` untuk menangani proses generate token dan facade `JWTAuth` yang akan kita gunakan untuk melakukan authentikasi.

3. Kita juga perlu mempublish file konfigurasi JWT, jalankan perintah `php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\JWTAuthServiceProvider"`. Setelah selesai, akan terdapat file baru di `/config/jwt.php`:

¹⁷⁸<https://scotch.io/tutorials/the-anatomy-of-a-json-web-token>

¹⁷⁹<https://github.com/tymondesigns/jwt-auth>

/config/jwt.php

```
<?php

return [
    'secret' => env('JWT_SECRET', 'changeme'),
    'ttl' => 60,
    'refresh_ttl' => 20160,
    'algo' => 'HS256',
    'user' => 'App\User',
    'identifier' => 'id',
    'required_claims' => ['iss', 'iat', 'exp', 'nbf', 'sub', 'jti'],
    'blacklist_enabled' => env('JWT_BLACKLIST_ENABLED', true),
    'providers' => [
        'user' => 'Tymon\JWTAuth\Providers\User\EloquentUserAdapter',
        'jwt' => 'Tymon\JWTAuth\Providers\JWT\NamshiAdapter',
        'auth' => function ($app) {
            return new Tymon\JWTAuth\Providers\Auth\IlluminateAuthAdapter($app['auth']);
        },
        'storage' => function ($app) {
            return new Tymon\JWTAuth\Providers\Storage\IlluminateCacheAdapter($app['cache']);
        }
    ]
];
```

Sebenarnya banyak komen yang berada di file konfigurasi ini. Disini, saya sederhanakan untuk membacanya lebih mudah. Untuk mendapatkan penjelasan lebih detail tentang file ini, baca [dokumentasinya](#)¹⁸⁰.

Salah satu konfigurasi yang penting kita isi adalah `secret`. Field ini harus kita isi dengan key yang kita gunakan untuk menggenerate token. Kita dapat mengisinya manual atau dengan menggunakan perintah `php artisan jwt:generate`, sehingga isinya akan seperti berikut:

```
'secret' => env('JWT_SECRET', 'IHN3Ck4JxeavzLtIa3RvfHsa2nIURzSQ'),
```

Isian lain yang penting adalah `ttl`. Isian ini merupakan lama waktu token sebelum expire dalam menit. Secara default adalah 60 menit. Kita dapat merubahnya sesuai kebutuhan.

¹⁸⁰<https://github.com/tymondesigns/jwt-auth/wiki/Configuration>

- Untuk memproteksi sebuah route dengan JWT Auth, kita dapat menggunakan middleware yang sudah disediakan oleh package ini. Kita cukup menambahkan baris berikut pada `/app/Http/Kernel.php`:

/app/Http/Kernel.php

```
<?php  
...  
class Kernel extends HttpKernel  
{  
    ...  
    protected $routeMiddleware = [  
        ...  
        'jwt.auth' => 'Tymon\JWTAuth\Middleware\GetUserFromToken',  
    ];  
}
```

Disini, kita menggunakan nama `jwt.auth` untuk middleware JWT Auth. Ketika token tidak valid, ada beberapa jenis response yang akan diberikan oleh middleware ini:

- `token_expired`, jika token sudah expired.
- `token_not_provided`, jika client tidak menyertakan token dalam requestnya.
- `token_invalid`, jika token tidak valid.

Dalam topik ini, kita akan mendemokan penggunaan JWT Auth untuk membuat API sederhana untuk mengakses daftar user. Kita akan menggunakan route yang diawali dengan `api`. Maka, di file route akan kita buat seperti ini:

/app/Http/routes.php

```
Route::group(['prefix' => 'api'], function() {  
    // route lain...  
});
```

Kita awali dengan membuat route untuk menggenerate token. Untuk mengakses route ini kita kirim variable `email` dan `password`. Kemudian, controller akan merespon dengan membuat token baru. Agar seragam, kita akan menggunakan `AuthController` dan membuat method `getToken`:

/app/Http/Controllers/Auth/AuthController.php

```
<?php  
....  
use Illuminate\Http\Request;  
use JWTAuth;  
use JWTException;  
  
class AuthController extends Controller  
{  
    ....  
    public function getToken(Request $request)  
    {  
        // mengambil credential dari client  
        $credentials = $request->only('email', 'password');  
  
        try {  
            // memvalidasi credential yang dikirim client  
            if (! $token = JWTAuth::attempt($credentials)) {  
                return response()->json(['error' => 'invalid_credentials'], 401);  
            }  
        } catch (JWTException $e) {  
            // ada error  
            return response()->json(['error' => 'could_not_create_token'], 500);  
        }  
  
        // mengirim token ke client  
        return response()->json(compact('token'));  
    }  
}
```

Pada controller ini kita akan menggenerate token dengan method `JWTAuth::attempt()`. Jika terjadi error kita akan mengirim response dengan status code 500 dan isian `json { error: could_not_create_token }`. Pada method ini, kita menggunakan email dan password. Jika diinginkan, kita juga bisa menggunakan username dan password.

Mari kita buat route untuk method ini dengan URL `/api/authenticate`:

/app/Http/routes.php

```
Route::group(['prefix' => 'api'], function() {
    Route::post('authenticate', 'Auth\AuthController@getToken');
});
```

Kita gunakan Postman untuk mengetes route ini. Isi method dengan POST, URL dengan <nama_domain>/api/authenticate dan isi body dengan email dan password yang telah kita buat di seeder. Jika kita coba kirimkan request, maka hasilnya akan seperti berikut.

The screenshot shows the Postman interface with a red circle highlighting the 'POST' method and the URL 'http://localhost:8000/api/authenticate'. Red arrows point from these highlights to the 'Body' tab where form-data fields 'email' (jajang@gmail.com) and 'password' (rahasia) are listed. Another red arrow points from the 'Send' button to the response window. The response window displays an error message: 'Whoops, looks like something went wrong.' followed by a detailed error stack trace: '1/1 TokenMismatchException in VerifyCsrfToken.php line 53:'. The stack trace shows three entries: 1. in VerifyCsrfToken.php line 53, 2. at VerifyCsrfToken->handle(object(Request), object(Closure)), and 3. at call user func anonymous(VerifyCsrfToken, handle, arrayToObject, object(Closure)) in Di.

Gagal membuat token karena CSRF middleware

Ooppsss... ada error. Error ini terjadi karena secara default Laravel mengaktifkan proteksi CSRF untuk route dengan http verb POST. Salah satu cara yang bisa kita lakukan adalah dengan mendisable middleware ini untuk route yang diawali dengan api. Tambahkan isian `api/*` pada variable `$except` di file `/app/Http/Middleware/VerifyCsrfToken.php`:

/app/Http/Middleware/VerifyCsrfToken.php

```
<?php
...
class VerifyCsrfToken extends BaseVerifier
{
    ...
    protected $except = [
        'api/*'
    ];
}
```

Kini, jika kita coba jalankan kembali request, kita akan berhasil mendapatkan token.

The screenshot shows the Postman interface. In the 'History' tab, there are two entries: a POST request to <http://localhost:8000/api/authenticate> and another POST request to the same endpoint. The second request is selected. The 'Body' tab is active, showing form-data fields: 'email' with value 'jajang@gmail.com' and 'password' with value 'rahasia'. The response tab shows a 200 OK status with a response time of 1216 ms. The response body is displayed in a JSONpretty-printed format:

```
1 {
2     "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiEsImlzcyI6Imh0dHA6XC9cL2xvY2FsaG9zdDo4MDAwXC9hcGlL2f1dGhlbnRpY2F0ZSi8ImhdC16MTQ1MjA1MDU3MywvIzXhwIjoxNDUyMDU0MTczLCJuYmYioJE0NTIwNTA1NzMsImp0aS16ImhNDE3ZTNjMzU0NTQ0YT15NzI1ODI4ZmQzOTg1ZjAwIn0.gzSo_oLD4D-HGFY6-EdpaR3dqWkiMeFuvUiXT3jL7xg"
3 }
```

Berhasil mendapatkan token

Sip. Kita sudah berhasil membuat token.

Selanjutnya, mari kita coba membuat route yang diproteksi dengan JWT Auth dan menampilkan JSON berisi semua user pada sistem:

/app/Http/routes.php

```
Route::group(['prefix' => 'api'], function() {
    Route::post('authenticate', 'Auth\AuthController@getToken');
    Route::group(['middleware' => 'jwt.auth'], function() {
        Route::get('users', function() {
            return App\User::all();
        });
    });
});
```

Terlihat disini, kita menggunakan middleware `jwt.auth` untuk melindungi route `/api/users` dengan JWT Auth. Penggunaan route group disini saya maksudkan untuk mempermudah pembuatan route yang diproteksi dengan `jwt.auth`, silahkan disesuaikan syntaxnya sesuai kebutuhan aplikasi yang sedang dibangun.

Untuk mencoba apakah middleware ini berfungsi cobalah lakukan request ke `/api/users` tanpa menggunakan token.

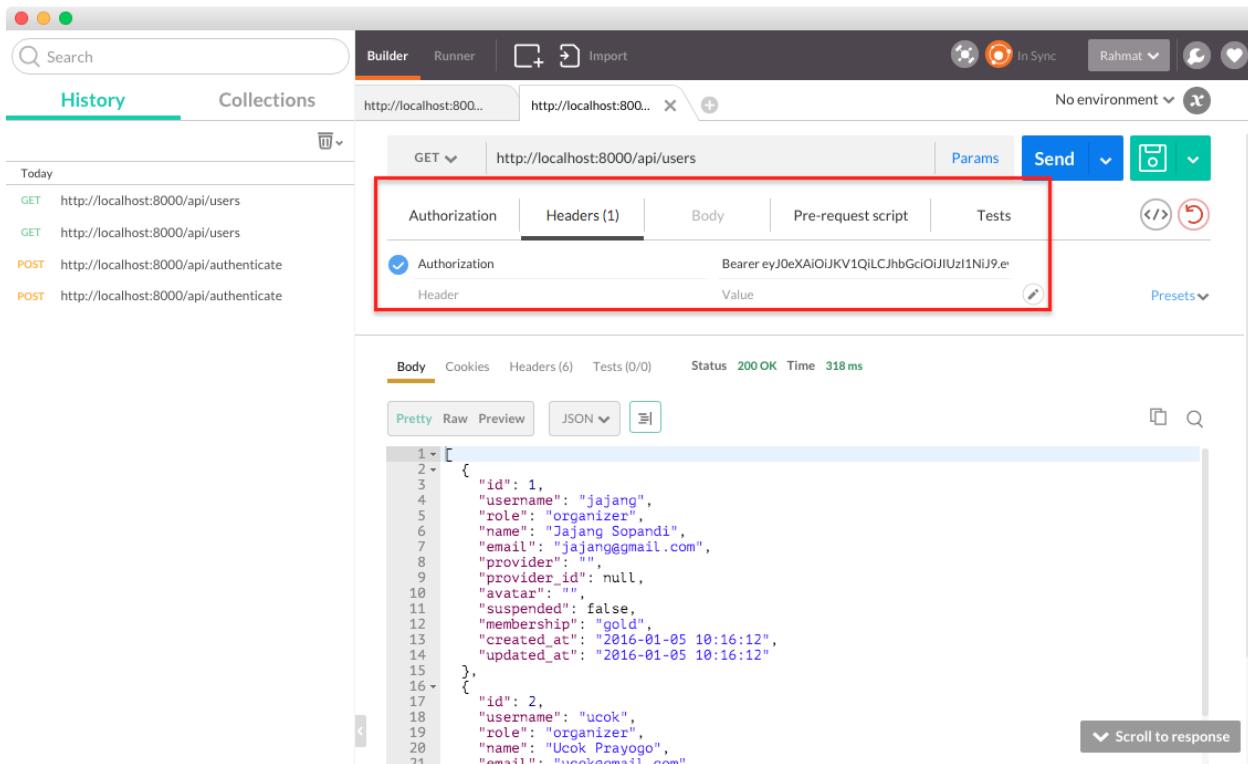
The screenshot shows the Postman application interface. On the left, there's a sidebar titled 'History' with a list of recent requests. The main area shows a single request configuration for a 'GET' request to 'http://localhost:8000/api/users'. The 'Authorization' tab is selected, showing 'No Auth'. Below the request details, the response body is displayed in JSON format, showing an error message:


```
1 { "error": "token_not_provided" }
```

 A red oval highlights the error message in the response body. At the bottom right of the main window, there's a button labeled 'Scroll to response'.

Request gagal tanpa token

Sip. Untuk menggunakan token pada request, ada dua cara yang bisa kita lakukan. Pertama, menggunakan header dengan *key* `Authorization` dan *value* `Bearer <token>` seperti berikut:

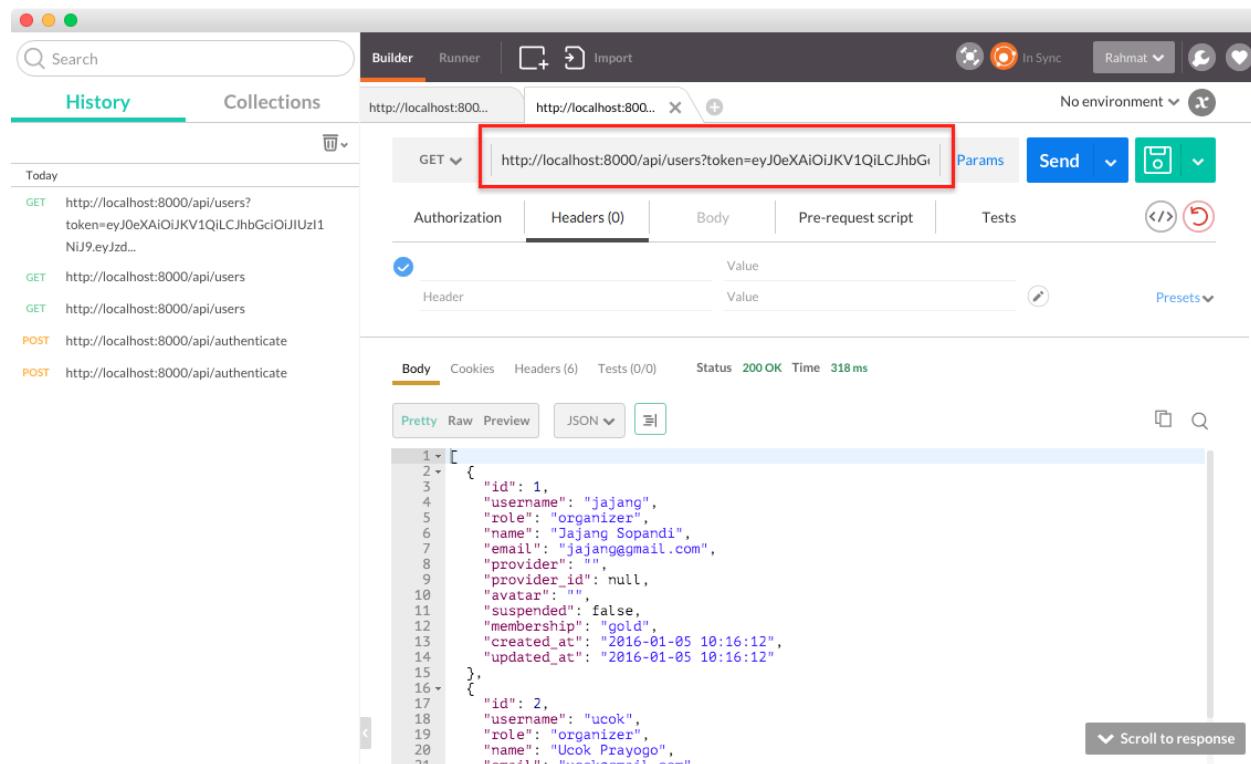


The screenshot shows the Postman application interface. At the top, there are tabs for 'Builder' and 'Runner', and a status indicator 'In Sync'. Below the tabs, there are two tabs: 'History' (selected) and 'Collections'. In the 'History' tab, there are several entries: 'GET http://localhost:8000/api/users', 'GET http://localhost:8000/api/users', 'POST http://localhost:8000/api/authenticate', and 'POST http://localhost:8000/api/authenticate'. The main area shows a 'GET' request to 'http://localhost:8000/api/users'. The 'Headers' tab is selected, showing an 'Authorization' header with the value 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e'. The 'Body' tab shows the JSON response:

```
1 [  
2   {  
3     "id": 1,  
4     "username": "jajang",  
5     "role": "organizer",  
6     "name": "Jajang Sopandi",  
7     "email": "jajang@gmail.com",  
8     "provider": "",  
9     "provider_id": null,  
10    "avatar": "",  
11    "suspended": false,  
12    "membership": "gold",  
13    "created_at": "2016-01-05 10:16:12",  
14    "updated_at": "2016-01-05 10:16:12"  
15  },  
16  {  
17    "id": 2,  
18    "username": "ucok",  
19    "role": "organizer",  
20    "name": "Ucok Prayogo",  
21    "email": "ucok@mail.com"  
]
```

Berhasil mengakses resource dengan menambah token di header

Yang kedua, dengan menambah *query string* pada url dengan key token seperti berikut:



Berhasil mengakses resource dengan menambah token di URL

Sip. Penjelasan tentang JWT Auth ini sebenarnya masih cukup panjang. Yang kita bahas disini akan cukup sebagai panduan awal untuk menggunakan JWT dalam pembuatan API. Jika tertarik belajar lebih lanjut saya sarankan membaca artikel di [toptal¹⁸¹](#) dan [scotch.io¹⁸²](#).



Source code dari latihan di bab ini bisa didapat di [Bitbucket¹⁸³](#)



Ringkasan

Authentikasi adalah topik yang selalu berkembang. Memahami penggunaan authorisasi di Laravel akan sangat bermanfaat dalam membangun aplikasi dengan *business rule* yang kompleks. Ketika kita sudah meng-authentikasi dan meng-authorisasi user untuk mengakses sistem, pastinya kita ingin agar data yang dia kirim telah valid. Itulah yang akan kita pelajari di bab selanjutnya tentang Validasi Data.

Semangat! :)

¹⁸¹ <http://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs>

¹⁸² <https://scotch.io/tutorials/the-anatomy-of-a-json-web-token>

¹⁸³ <https://bitbucket.org/rahmatalawudin/sample-authentikasi/commits/all>

Validasi Data

Jangan pernah percaya pada input dari user. Selama saya berada di industri web development, nasehat ini yang selalu saya pegang. Bila aplikasi kita memiliki celah untuk di bobol melalui input user, sekecil apapun, pasti akan ditemukan oleh user. Sudah kewajiban kita untuk memastikan agar setiap input user merupakan input yang valid.

Validasi data, selama saya membuat web, merupakan tahapan yang cukup memakan waktu. Salah satu masalah yang biasanya muncul dari bagaimana validasi harus dilakukan. Masalah ini, seharusnya sudah selesai dengan fitur validasi di Laravel yang, menurut saya, cukup komplit. Bila tidak cukup, kita bahkan bisa membuat *rule* validasi sesuai alur bisnis aplikasi yang sedang kita bangun.

Tanpa banyak basa-basi lagi, yuk kita mulai!

Memahami Validasi di Laravel

Untuk menggunakan fitur validasi di Laravel, kita akan membuat form sederhana untuk mengisi table `songs` yang berisi nama lagu, album, penyanyi dan durasinya. Silahkan buat dulu project Laravel baru dengan perintah `composer create-project laravel/laravel --prefer-dist sample-validasi` dan konfigurasi databasenya.

Saat saya menulis buku ini, saya menggunakan Laravel 5.2.

Mari kita buat sample validasi data sederhana dimana kita akan membuat sebuah form untuk menambah lagu dengan isian `track`, `title`, `album` dan `artist`. Form ini akan kita validasi dari controller.

Buatlah view `/resources/views/songs/create.blade.php` dengan isi:

/resources/views/songs/create.blade.php

```
<form method="POST" action="/songs">

    {!! csrf_field() !!}

    <div>
        Track
        <input type="text" name="track_no" value="{{ old('track_no') }}">
    </div>

    <div>
        Judul
        <input type="text" name="title" value="{{ old('title') }}">
    </div>

    <div>
        Album
        <input type="text" name="album" value="{{ old('album') }}">
    </div>

    <div>
        Penyanyi
        <input type="text" name="artist" value="{{ old('artist') }}">
    </div>

    <div>
        <button type="submit">Tambah</button>
    </div>
</form>
```

Pada form ini kita menggunakan 4 field yang sudah kita sebutkan diatas dan mengirimnya ke URL songs.

Untuk menampilkan form yang kita buat, kita akan membuat URL /songs/new pada file route dan mengarahkannya ke method `create` di SongsController:

/app/Http/routes.php

```
Route::group(['middleware' => ['web']], function () {
    Route::get('songs/new', 'SongsController@create');
});
```

Route ini kita masukan ke middleware group `web`. Kita harus membuat `SongsController` jalankan perintah `php artisan make:controller SongsController`. Pada file yang dihasilkan, tambahkan method `create` dengan isian berikut:

/app/Http/Controllers/SongsController.php

```
public function create()
{
    return view('songs.create');
}
```

Disini, kita menampilkan form yang telah kita buat di langkah pertama.

Untuk menerima form, kita akan menggunakan URL `/songs` dengan method `POST`:

/app/Http/routes.php

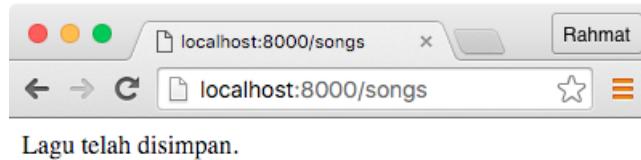
```
Route::group(['middleware' => ['web']], function () {
    ...
    Route::post('songs', 'SongsController@store');
});
```

Terlihat disini, kita mengarahkan ke method `store` pada `SongsController`. Pada method `store`-lah kita akan melakukan validasi. Untuk sementara, mari kita biarkan method ini tanpa validasi.

/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    // Validasi akan dilakukan disini
    // Ceritanya data disimpan ke database
    return 'Lagu telah disimpan.';
}
```

Mari kita coba form submit form ini tanpa mengisi semua field nya. Maka akan tampil:



Submit form berhasil meskipun form tidak diisi

Tuh, dia berhasil? Seharusnya, kita tidak mengizinkan hal seperti ini terjadi. Jika kita benar-benar menyimpan data ke database, tentunya akan terjadi error karena kita mencoba membuat record baru tanpa isian.

Mari kita coba buat validasi sederhana untuk memastikan isian title diisi:

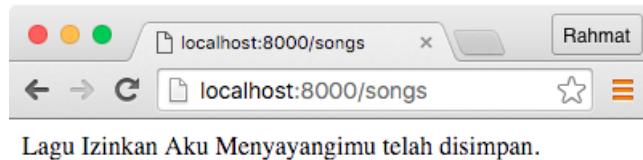
/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required'
    ]);

    return 'Lagu ' . $request->title . ' telah disimpan.';
}
```

Method validate bisa kita jalankan karena App\Http\Controllers\Controller secara default menggunakan trait Illuminate\Foundation\Validation\ValidatesRequests.

Kini, jika kita coba submit tanpa mengisi judul lagu, maka kita akan dikembalikan ke form. Cobalah isi judul lagunya, maka kita akan mendapatkan tampilan berikut:



Berhasil melakukan validasi judul lagu

Oke, saya fans Iwan Fals, lagu yang diatas merupakan salah lagu favorit saya.

Menggunakan beberapa rule validasi pada satu field

Balik lagi ke validasi. Kita sudah berhasil memvalidasi bahwa field `title` sudah diisi. Tidak hanya sampai disini, kita dapat menggunakan beberapa rule validasi. Misalnya kita hendak memvalidasi bahwa `artist` harus diisi dan isiannya harus Iwan Fals atau Afgan. Tenang, kalau Anda ngga suka kedua penyanyi, silahkan diganti sesuai selera.

/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required',
        'artist' => 'required|in:Iwan Fals,Afgan'
    ]);

    return 'Lagu ' . $request->title . ' oleh ' . $request->artist . ' telah disimpan.';
}
```

Untuk menggunakan beberapa rule, kita menggunakan karakter `|` sebagai pemisahnya. Rule kedua yang kita pakai untuk memastikan artis yang diinput user adalah artis yang saya suka, eh, artis yang sesuai dengan rule validasi. Rule `in` menerima parameter. Untuk menambah parameter, kita menggunakan karakter `:`. Terlihat disini, kita menambahkan dua parameter. Silahkan tambahkan artis lain yang Anda suka disini.

Cobalah menambah lagu baru dengan tanpa mengisi penyanyi, pasti kita akan dikembalikan ke form. Begitupun jika kita menambah lagu dengan penyanyi selain Iwan Fals atau Afgan, maka kita akan gagal. Kini cobalah masukan "Pencari Jalan Mu" dan isi penyanyi Afgan, pasti berhasil.. :)



Lagu Pencari Jalan Mu oleh Afgan telah disimpan.

Berhasil memvalidasi judul dan penyanyi

Menampilkan pesan error

Tidak menarik jika kita memiliki rule validasi tapi user tidak mampu melihat apa yang salah dari input nya. Mari kita tampilkan pesan error dari validasi ini. Setiap kali kita menggunakan method `validate()` pada controller dan data tidak valid, maka Laravel akan mengisi variable flash session (session yang hanya aktif pada route selanjutnya) bernama `$errors` yang berisi instance dari `Illuminate\Support\MessageBag`. Ketika kita kembali ke view, Laravel akan selalu mengecek apakah di session terdapat key `errors`, jika ya, maka Laravel akan mengirimkannya ke view dengan key `errors`.

Singkatnya, kita bisa akses variable `$errors` yang berisi semua error pada validasi dari view.

Ada beberapa method yang disediakan `MessageBag`, mari kita pelajari satu-persatu.

Menampilkan semua error

Untuk menampilkan semua error, kita dapat menggunakan method `all()`. Misalnya, kita buat seperti ini:

/resources/views/songs/create.blade.php

```
<form method="POST" action="/songs">

@if (count($errors) > 0)
<ul>
    @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
@endif
....
```

Ketika terjadi error, ini yang akan tampil di view:

**Menampilkan semua error****Menampilkan semua error untuk sebuah field**

Kita dapat menggunakan method `get()` dengan parameter nama field yang kita cari. Untuk memudahkan mendemokan fitur ini, mari kita tambah `rule` agar isian artist minimal 3 karakter:

/app/Http/Controllers/SongsController.php

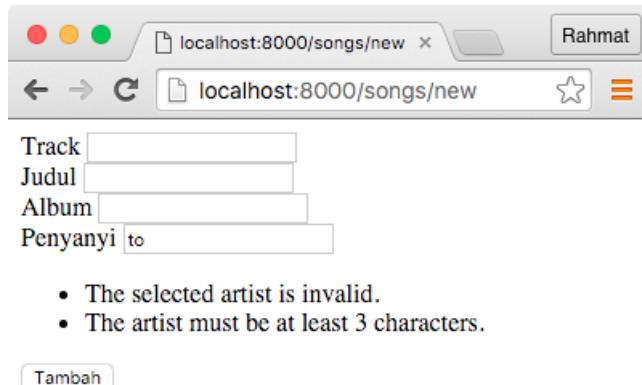
```
public function store(Request $request)
{
    $this->validate($request, [
        ...
        'artist' => 'required|in:Iwan Fals,Afgan|min:3'
    ]);
    ...
}
```

Kita tambahkan method get() dibawah field artist:

/resources/views/songs/create.blade.php

```
<div>
    Penyanyi
    <input type="text" name="artist" value="{{ old('artist') }}>
    <ul>
        @foreach ($errors->get('artist') as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
```

Jika kita mengisi field artist dengan 2 karakter, ini yang akan tampil:



Menampilkan semua error untuk sebuah field

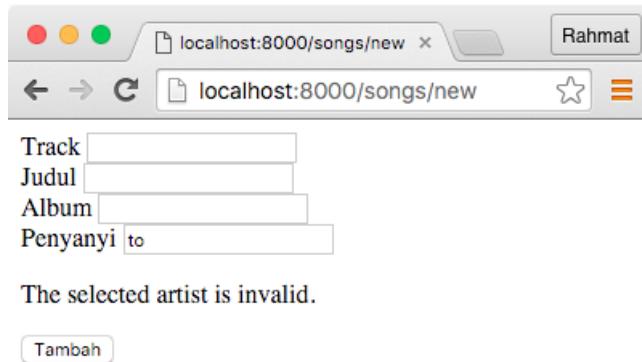
Menampilkan error pertama untuk sebuah field

Tentunya, dalam penggunaan di lapangan, terkadang kita hanya ingin menampilkan error pertama pada sebuah field. Untuk itu, kita dapat menggunakan method `first()`.

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

```
<div>
    Penyanyi
    <input type="text" name="artist" value="{{ old('artist') }}>
    <p>{{ $errors->first('artist') }}</p>
</div>
```

Dengan error yang sama seperti contoh sebelumnya, ini yang akan tampil:



Hanya menampilkan satu error untuk sebuah field

Menampilkan error dalam format khusus

Pada contoh sebelumnya, kita mengapit error dengan tag `<p>`. Ini sangat umum dalam menampilkan error, mungkin kita mengapitnya dalam sebuah `<div>` dengan class tertentu. Untuk itu, kita dapat membuat template untuk error misalnya seperti berikut:

/resources/views/songs/create.blade.php

```
<div>
    Penyanyi
    <input type="text" name="artist" value="{{ old('artist') }}>
    {!! $errors->first('artist', '<div class="error">:message</div>') !!}
</div>
```

Sengaja kita menggunakan {!! !!} karena kita hendak menampilkan tag html. Maka yang akan tampil:

The screenshot shows a browser window with a form for creating a song. The fields are labeled 'Track', 'Judul', 'Album', and 'Penyanyi'. The 'Penyanyi' field has the value 'to'. Below the form, an error message is displayed: 'The selected artist is invalid.' A 'Tambah' button is also visible.

The second part of the screenshot shows the browser's developer tools Elements tab. It highlights the generated HTML code for the 'Penyanyi' field and its associated error message. The error message is enclosed in a red box. The code snippet is as follows:

```
<input type="text" name="artist" value="to">
<div class="error">The selected artist is invalid.</div>
```

The 'div.error' tab in the developer tools is selected. Below the tabs, there are buttons for Styles, Event Listeners, DOM Breakpoints, and Properties.

Menampilkan error dengan format

Mengecek error untuk sebuah field

Kita dapat mengecek error dengan method `has()`. Misalnya, kita cek apakah ada error pada field `title` dan menambahkan class `has-error` pada inputnya. Ini yang kita tulis:

/resources/views/songs/create.blade.php

```
<div>
    Judul
    <input type="text" name="title" class="{{ $errors->has('title') ? 'has-error' : '' }}" value="{{ old('title') }}>
</div>
```

Disini kita menggunakan [ternary operator](#)¹⁸⁴ untuk memudahkan mengecek error dan menampilkan string has-error. Output yang akan kita dapatkan ketika terdapat error pada field title:

The screenshot shows a browser window with a form for creating a song. The 'Judul' field is empty and has a red border, indicating it is required. Below the form, a developer tools window shows the HTML code for the 'Judul' field, specifically the line: `<input type="text" name="title" class="has-error" value>`, which is highlighted with a red box.

Mengecek error

Memodifikasi pesan error

Menggunakan pesan error bawaan Laravel biasanya dilakukan oleh developer mainstream. Ehm, karena kita anti mainstream, mari kita coba modifikasi pesan error validasi ini biar lebih kekinian.

Mengganti pesan error sebuah rule untuk semua field

Kita bisa mengganti pesan error untuk sebuah rule validasi dengan merubahnya pada file /resources/lang/en/validation.php. Teknik ini pernah saya lakukan ketika

¹⁸⁴<http://php.net/manual/en/language.operators.comparison.php>

membangun API dan membutuhkan pesan error yang *machine readable*. Formatnya `key => format_pesan_error`. Misalnya pesan error untuk rule `required` kita ubah menjadi `is_required`:

/resources/lang/en/validation.php

```
'required' => 'is_required',
```

Jika kita coba refresh dan tidak mengisi isian `artist`, maka kita akan mendapatkan output seperti berikut:

The screenshot shows a browser window with the URL `localhost:8000/songs/new`. The page displays a form with four input fields: 'Track' (filled with 'Lagu'), 'Judul' (empty), 'Album' (empty), and 'Penyanyi' (empty). Below the inputs is a label 'is_required' and a 'Tambah' button. The browser's status bar at the bottom shows the message 'Required field is empty'.

Mengubah pesan validasi secara global

Untuk memudahkan pembahasan selanjutnya, silahkan kembalikan isian pesan validasi untuk rule `required` ke nilai default.

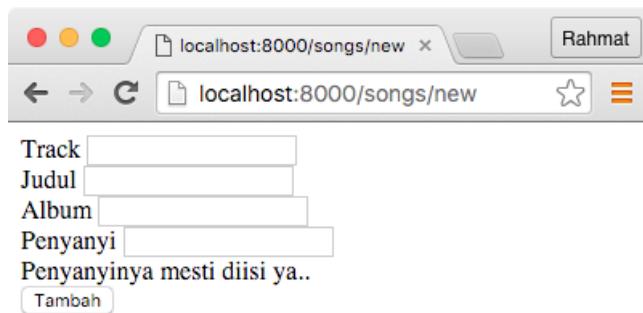
Mengganti pesan error untuk sebuah field dan rule tertentu

Terkadang kita butuh mengganti pesan error sebuah rule untuk field tertentu. Misalnya, setiap rule `required` untuk field `artist` akan kita tampilkan pesan error `Penyanyinya mesti diisi ya...`. Maka kita dapat menambahkan pada key `custom` di `/resources/lang/en/validation.php`. Misalnya seperti berikut:

/resources/lang/en/validation.php

```
'custom' => [
    ...
    'artist' => [
        'required' => 'Penyanyinya diisi ya...'
    ]
],
```

Jika kita tidak mengisi penyanyi, maka pesan ini yang akan muncul.

**Merubah pesan error untuk sebuah rule pada field tertentu****Mengganti pesan error secara inline**

Menggunakan file validation.php efektif digunakan jika nama field yang kita gunakan tidak dimiliki oleh table lain. Dalam contoh diatas, pengaturan pesan error untuk field artist pada file validation.php sangat cocok karena kita asumsikan field artist tidak dimiliki oleh table lain.

Untuk field yang standar, misalnya title, akan lebih baik jika kita menggunakan inline error message untuk mengubah pesan errornya. Caranya kita menambah array kedua pada saat menggunakan method validate dengan key field.rule dan value berupa pesan errornya.

Misalnya, kita ubah seperti berikut:

/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required',
        'artist' => 'required|in:Iwan Fals,Afgan|min:3'
    ], [
        'title.required' => 'Isi dengan judul lagu yang populer bro.'
    ]);
    ...
}
```

Jika kita coba submit form tanpa isian `title`, ini yang akan kita dapatkan.

**Inline error message**

Sebenarnya, ada satu teknik lagi untuk merubah pesan error ini yaitu menggunakan fitur Localization di Laravel. Ini akan berguna jika aplikasi kita mendukung penggunaan multibahasa. Misalnya, jika kita ingin menambah pesan error dalam bahasa Indonesia, kita perlu mengcopy file `/resources/lang/en/validation.php` ke `/resources/lang/id/validation.php` dengan format yang sama.

Selanjutnya, kita dapat menggunakan `App::setLocale('id')` di controller untuk merubah bahasanya. Bisa juga merubah bahasa default di `/config/app.php` dan mengubah key `locale` menjadi `id`. Lebih lengkapnya, cek dokumentasi¹⁸⁵.

Validasi Kondisional

Adakalanya kita memiliki sebuah table dengan beberapa field dimana sebagian field pada form hanya kita tampilkan dalam kondisi tertentu. Di contoh kita, misalnya

¹⁸⁵ <https://laravel.com/docs/5.2/localization>

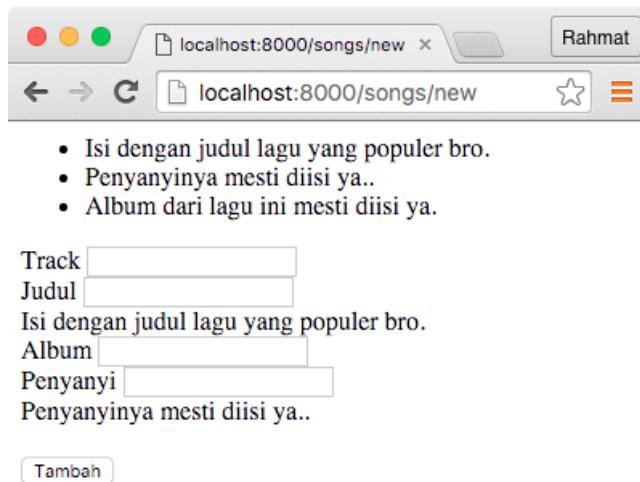
dengan logic tertentu (misalnya dengan JS) pada HTML kita akan menampilkan isian album. Tentunya, ketika isian album tidak terdapat di form, kita tidak perlu melakukan validasi. Namun, jika ada di form, kita akan melakukan validasi.

Untuk melakukan validasi seperti ini, kita dapat menggunakan key `sometimes` pada rule validasi. Misalnya seperti berikut:

/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        ...
        'album' => 'sometimes|required|alpha_num'
    ], [
        ...
        'album.required' => 'Album dari lagu ini mesti diisi ya.'
    ]);
}
```

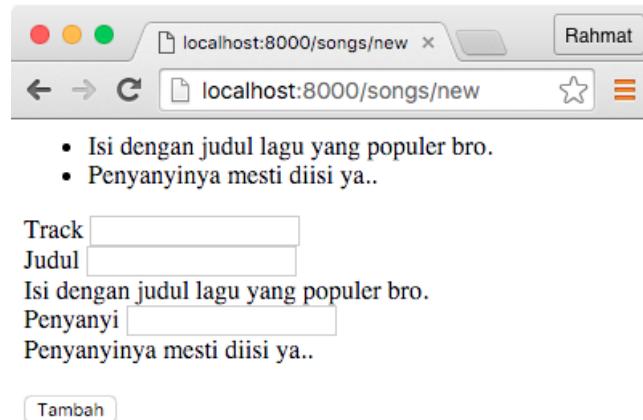
Jika terdapat field album di form (seperti yang terjadi sekarang), maka Laravel akan melakukan validasi untuk field album:



Validasi untuk field album dilakukan

Disini saya menampilkan kembali semua error diatas form. Silahkan ubah view jika diinginkan.

Jika kita menghilangkan field album dari form, maka field tersebut tidak akan divalidasi (meskipun memiliki rule `required`):



Field album tidak divalidasi

Sip, itu contoh pertama.

Selanjutnya, terkadang kita juga ingin melakukan validasi untuk sebuah field jika field lain terisi. Pada contoh kita, ketika user mengisi isian `track_no`, maka kita akan mewajibkan dia mengisi isian Album. Kita dapat menggunakan berbagai cara.

Pertama cara manual, jika kita ingin menggunakan teknik ini di method `validate` di Controller, kita dapat membuat rule wajib pada sebuah variable dan menambah rule lainnya sesuai kondisi yang kita inginkan.

Misalnya seperti berikut:

`/app/Http/Controllers/SongsController.php`

```
public function store(Request $request)
{
    $rules = [
        'title' => 'required',
        'artist' => 'required|in:Iwan Fals,Afgan|min:3',
        'album' => 'sometimes|required|alpha_num'
    ];
    if ($request->has('album')) $rules['track_no'] = 'required|integer|min:0';

    $this->validate($request, $rules, [
        'title.required' => 'Isi dengan judul lagu yang populer bro.',
        'album.required' => 'Album dari lagu ini mesti diisi ya.',
        'track_no.required' => 'Nomor tracknya diisi ya.'
    ]);
    ...
}
```

Jika kita submit form tanpa mengisi album, maka `track_no` tidak akan divalidasi:

The screenshot shows a browser window with the URL `localhost:8000/songs/new`. The page contains a form with fields for Track, Judul, Album, and Penyanyi. Below the form, three validation messages are listed:

- Isi dengan judul lagu yang populer bro.
- Penyanyinya mesti diisi ya..
- Album dari lagu ini mesti diisi ya.

The 'Album' field is empty, and its corresponding validation message is highlighted with a red box.

Tidak melakukan validasi pada `track_no`

Tapi, jika kita mengisi album, maka validasi pada `track_no` akan berjalan:

The screenshot shows a browser window with the URL `localhost:8000/songs/new`. The page contains a form with fields for Track, Judul, Album, and Penyanyi. The 'Album' field now contains the value 'test album'. Below the form, four validation messages are listed, with the fourth one highlighted by a red box:

- Isi dengan judul lagu yang populer bro.
- Penyanyinya mesti diisi ya..
- The album may only contain letters and numbers.
- Nomor tracknya diisi ya.

The 'Album' field and its validation message are highlighted with a red box.

Melakukan validasi pada `track_no`

Kedua menggunakan method `sometimes` pada instance dari Validator. Ini yang tertulis di dokumentasi Laravel. Teknik ini tidak bisa kita lakukan pada method `validate` karena, saat tulisan ini dibuat, kita tidak bisa memodifikasi instance Validator pada method `validate`. Maka, kita harus membuat validasi manual, misalnya seperti berikut:

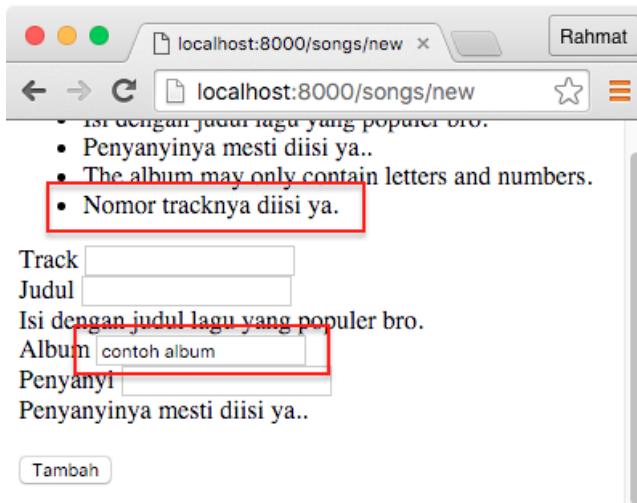
/app/Http/Controllers/SongsController.php

```
1 public function store(Request $request)
2 {
3     $validator = \Validator::make($request->all(), [
4         'title' => 'required',
5         'artist' => 'required|in:Iwan Fals,Afgan|min:3',
6         'album' => 'sometimes|required|alpha_num'
7     ], [
8         'title.required' => 'Isi dengan judul lagu yang populer bro.',
9         'album.required' => 'Album dari lagu ini mesti diisi ya.',
10        'track_no.required' => 'Nomor tracknya diisi ya.'
11    ]);
12
13    $validator->sometimes('track_no', 'required|integer|min:0', function($input) {
14    {
15        return $input->album !== '';
16    });
17
18    if ($validator->fails()) {
19        return redirect('/songs/new')
20            ->withInput()
21            ->withErrors($validator);
22    }
23
24    App::setLocale('id');
25
26    return 'Lagu ' . $request->title . ' oleh ' . $request->artist . ' telah dis\
27 impan.';
28 }
```

Oke, syntaxnya cukup banyak, mari kita bahas lebih detail.

- Baris 3-11: kita membuat instance validator dengan rule wajib dan pesan error custom.
- Baris 13-15: kita menambah rule dynamic untuk field `track_no` jika isian `album` tidak kosong.
- Baris 17-21: kita mengecek apakah validasi gagal. Ketika terjadi gagal, kita redirect user ke halaman `/songs/new` berikut input yang telah dia kirim (`withInput()`) dan mengeset variable `error` dari validator (`withErrors($validator)`).
- Baris 23-25: asih sama dengan logika sebelumnya, kita merubah locale ke Indonesia dan memberikan response ke user.

Silahkan dicek. Pastikan rule untuk `track_no` hanya berjalan jika kita mengisi isian album.



Kondisional validation dengan method `sometimes()`

Untuk memudahkan pembahasan selanjutnya, ubah kembali validasi menggunakan method `validate()`.

Validasi Array (update Laravel 5.2)

Ketika membuat form dengan isian berupa array. Pada contoh kita, misalnya kita kita mengizinkan nama artist sebagai Band dan dapat mengisi nama personilnya maksimal 5 orang. Kita dapat menambah field isian personil seperti berikut:

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

```
<div>
    Anggota
    @foreach (range(1,5) as $index)
        <p>Personil {{ $index }} : <input type="text" name="personil[{{ $index }}][name]" value="{{ old("personil.$index.name") }}"></p>
        {!! $errors->first("personil.$index.name", '<p>:message</p>') !!}
    @endforeach
</div>

<div>
    <button type="submit">Tambah</button>
</div>
</form>
```

Disini kita menggunakan fungsi `range()` untuk menggenerate array dari 1 - 5. Kemudian me-looping array tersebut dan membuat input dengan nama `personil.<index>.name`. Dan menampilkan pesan error jika gagal melakukan validasi.

Pada controller, kita akan menggunakan rule `alpha_num` supaya hanya angka dan huruf yang bisa diisi pada nama personil:

/app/Http/Controllers/SongsController.php

```
public function store(Request $request)
{
    $rules = [
        ...
        'personil.*.name' => 'alpha_num'
    ];
    ...
    $this->validate($request, $rules, [
        ...
        'personil.*.name.alpha_num' => 'Nama personil mesti valid.'
    ]);
    ...
}
```

Untuk mengakses array ketika validasi, kita menggunakan karakter `*`. Disini, `personil.*.name` artinya kita mengakses semua elemen `name` pada elemen dari array `personil`. Pesan error untuk semua field tersebut pun kita samakan “Nama personil mesti valid.”.

Untuk mencoba validasi ini, bukalah form dan coba isi personil dengan karakter selain angka dan huruf.

localhost:8000/songs/new

Rahmat

Track

Judul

Isi dengan judul lagu yang populer bro.

Album

Penyanyi

Penyanyinya mesti diisi ya..

Anggota

Personil 1 :

Nama personil mesti valid.

Personil 2 :

Personil 3 :

Personil 4 :

Personil 5 :

Tambah

Berhasil melakukan validasi array

Validasi untuk Ajax / API

Dalam membangun aplikasi, seringkali kita menerima request ajax terutama bagi yang mengembangkan API. Tentunya ketika kita menerima request ajax, response untuk validasipun berbeda. Kabar baiknya, fitur validasi Laravel akan secara otomatis merubah response menjadi sebuah JSON berisi detail error berikut Status Code 422.

Mari kita coba demokan fitur ini dengan membuat url /ajax-validation yang akan menerima request ajax. Tambahkan baris berikut diluar middleware web pada file route:

/app/Http/routes.php

```
Route::group(['middleware' => ['web']], function () {
    ...
});
Route::post('/ajax-validation', 'HomeController@testAjax');
```

Kita akan menggunakan method `testAjax` pada `HomeController`. Mari kita buat controllernya dengan menjalankan `php artisan make:controller HomeController`. Pada file yang dihasilkan buatlah method `testAjax()` dengan isi sebagai berikut:

/app/Http/Controllers/HomeController.php

```
public function testAjax(Request $request)
{
    $this->validate($request, [
        'name' => 'required|alpha_num|min:5'
    ]);

    return 'All data valid.';
}
```

Disini kita menggunakan 3 rule untuk memastikan field `name` diisi, hanya berisi huruf dan angka dan minimal 5 karakter.

Mari kita simulasi Ajax request dengan Postman. Caranya, kita menambah header `X-Requested-With` dengan isi `XMLHttpRequest`. Buatlah request POST tanpa data ke `/ajax-validation`. Ini yang akan kita dapatkan:

The screenshot shows the Postman application interface. In the top navigation bar, 'Builder' is selected. Below it, a request is listed: 'localhost:8000/ajax-validation'. The 'POST' method is chosen. The 'Headers' tab shows an 'X-Requested-With' header with the value 'XMLHttpRequest'. The 'Body' tab is selected, showing a JSON response: { "name": ["The name field is required."] }. A red arrow points from the 'Send' button to the 'Body' tab. Another red arrow points from the 'Pretty' button to the JSON response. A red oval highlights the JSON response body.

Validasi ajax berhasil

Terlihat disini, kita berhasil melakukan validasi untuk request Ajax yang dilakukan. Laravel juga akan mengeset header Content-Type menjadi application/json pada response yang dibuat. Keren!

Response error yang dibuat oleh Laravel memiliki format seperti berikut:

```
{  
    field1: [  
        pesan_error_1,  
        pesan_error_2,  
        dst  
    ],  
    field2: [  
        pesan_error_1,  
        pesan_error_2,  
        dst  
    ],  
}
```

Mari kita coba mengisi name dengan isian yang tidak valid:

The screenshot shows the Postman interface. In the 'Body' tab, there is a 'form-data' section with a single entry: 'name' with a value of '#'. A red box highlights this entry. Below it, another red box highlights the JSON response in the 'Pretty' tab. The response is:

```
{  
  "name": [  
    "The name may only contain letters and numbers.",  
    "The name must be at least 5 characters."  
  ]  
}
```

Validasi untuk field name via ajax

Terlihat disini, validasi untuk rule `alpha_num` dan `min` berhasil dilakukan.

Menggunakan berbagai rule validasi di Laravel

Topik ini akan cukup panjang karena kita akan mendemokan semua fitur validasi di Laravel. Cukup banyak, ketika buku ini ditulis terdapat 42 rule. Untuk memudahkan kita mencoba berbagai rule ini, kita akan menggunakan Postman dan method `testAjax()` yang kita gunakan pada pembahasan sebelumnya.

Accepted

Rule ini biasanya digunakan untuk checkbox, misalnya checkbox bahwa user telah menyetujui aturan penggunaan situs. Misalnya, kita buat rule seperti ini:

```
'term_of_service' => 'accepted'
```

Ketika kita tidak mengisi isian ini, maka akan muncul error:

```
{  
    "term_of_service": [  
        "The term of service must be accepted."  
    ]  
}
```

Active URL

Rule ini akan mengecek apakah input yang dimasukan user merupakan URL yang valid dengan mengetesnya menggunakan fungsi `checkdnsrr` di PHP¹⁸⁶. Misalnya, kita menulis seperti ini:

```
'your_website' => 'active_url'
```

Ketika kita memasukan domain yang tidak valid, maka kita akan mendapat error:

```
{  
    "your_website": [  
        "The your website is not a valid URL."  
    ]  
}
```

After (Date)

Rule ini bisa kita gunakan untuk memastikan input tanggal yang dimasukkan oleh user lebih dari tanggal yang kita tulis pada parameter rule. Parameter tanggal yang diterima fungsi ini akan dipassing ke fungsi `strtotime`¹⁸⁷. Biasanya penggunaan rule ini digunakan bersamaan dengan rule `date` agar data yang diinput merupakan tanggal yang valid.

Misalnya kita tulis:

```
'release_date' => 'date|after:10 January 2000'
```

Ketika kita mengisi `release_date` dengan isian `1998-03-10` maka kita akan mendapat error:

¹⁸⁶ <http://php.net/manual/en/function.checkdnsrr.php>

¹⁸⁷ <http://php.net/manual/en/function.strptime.php>

```
{  
    "release_date": [  
        "The release date must be a date after 10 January 2000."  
    ]  
}
```

Contoh parameter lain yang bisa digunakan untuk rule ini:

- “now”
- “10 September 2000”
- “+1 day”
- “+1 week”
- “+1 week 2 days 4 hours 2 seconds”
- “next Thursday”
- “last Monday”
- “tomorrow”
- dll.

Alpha

Rule ini digunakan ketika kita menginginkan sebuah field hanya berisi huruf. Misalnya kita buat rule:

```
'name' => 'alpha'
```

Ketika kita mencoba mengisi `name` dengan karakter selain huruf, ini error yang akan kita dapatkan:

```
{  
    "name": [  
        "The name may only contain letters."  
    ]  
}
```

Alpha Dash

Sama dengan rule `alpha` dengan tambahan mengizinkan penggunaan - dan _.

Alpha Numeric

Sama dengan rule `alpha` dengan tambahan mengizinkan penggunaan angka.

Array

Bisa kita gunakan untuk mengecek apakah input merupakan array. Pada contoh kita sebelumnya, ini bisa kita gunakan untuk memvalidasi input `personil`. Kita dapat menggunakannya seperti ini:

```
'personil' => 'array'
```

Jika input yang diterima bukan array, kita akan mendapat error:

```
{
    "personil": [
        "The personil must be an array."
    ]
}
```

Before (Date)

Kebalikan dari rule `after`. Penggunaannya sama dengan rule `after`.

Between

Bisa kita pakai untuk memvalidasi sebuah input berada dalam batas tertentu. Bisa digunakan untuk input string, integer dan file. Teknik yang digunakan untuk menentukannya ukuran field sama dengan rule `size`.

Misalnya kita buat rule seperti ini:

```
'jumlah' => 'integer|between:10,20'
```

Jika kita mengisi isian dibawah 10 atau diatas 20, maka ini error yang akan kita dapatkan:

```
{
    "jumlah": [
        "The jumlah must be between 10 and 20."
    ]
}
```

Boolean

Ketika kita hendak merubah isian sebuah field menjadi boolean, ini rule yang kita gunakan. Kita dapat mengisinya dengan `true`, `false`, `1`, `0`, `"1"`, dan `"0"`.

Misalnya kita punya rule seperti ini:

```
'married' => 'boolean'
```

Jika kita tidak mengisi dengan salah satu isian yang diperbolehkan diatas, ini error yang akan kita dapatkan:

```
{
  "married": [
    "The married field must be true or false."
  ]
}
```

Confirmed

Sesuai namanya, rule ini akan memaksa user untuk mengkonfirmasi inputnya. Menggunakan rule ini, kita dapat memaksa user untuk mengetikan input sebanyak dua kali. Contoh sederhana adalah input untuk password ketika user register. Untuk menggunakannya, kita harus membuat input kedua dengan nama `field_confirmation`. Misalnya, jika kita hendak memaksa user mengkonfirmasi isian `password`, maka kita harus membuat `password_confirmation` pada form.

Rule yang kita buat akan seperti ini:

```
'password' => 'confirmed'
```

Jika isian pada `password_confirmation` tidak sesuai dengan isian pada `password` atau kita tidak mengisi `password_confirmation`, ini error yang akan kita dapatkan:

```
{
  "password": [
    "The password confirmation does not match."
  ]
}
```

Date

Jika kita hendak memastikan bahwa input yang user masukan format tanggal yang benar, ini rule yang kita gunakan. Rule ini akan mengecek input dari user dan menggunakan fungsi `strtotime`¹⁸⁸.

Misalnya kita tulis rule seperti ini:

¹⁸⁸<http://php.net/manual/en/function.strptime.php>

```
'birth_date' => 'date'
```

Jika kita tidak memasukkan format yang bisa diproses oleh fungsi `strtotime`, kita akan mendapat error:

```
{
    "birth_date": [
        "The birth date is not a valid date."
    ]
}
```

Date Format

Rule ini bisa kita gunakan ketika kita hendak memaksa user untuk mengirim input tanggal dengan format tertentu. Rule ini menggunakan fungsi `date_parse_from_format`¹⁸⁹.

Misalnya, kita ingin agar input `date_birth` dalam format `tanggal-bulan-tahun`, maka kita tulis:

```
'birth_date' => 'date_format:j-n-Y'
```

Contoh input yang valid untuk rule ini `1-12-2001` untuk tanggal 1 Januari 2001. Jika kita menulis input yang tidak valid, ini error yang akan kita dapatkan:

```
{
    "birth_date": [
        "The birth date does not match the format j-n-Y."
    ]
}
```

Untuk melihat code format untuk tanggal ini, cek dokumentasi untuk `DateTime::createFromFormat`

Different

Rule ini bisa kita gunakan untuk memastikan isian pada sebuah field berbeda dengan isian pada field lainnya pada sebuah form. Misalnya, kita ingin memastikan bahwa isian `father` dan `mother` berbeda, kita dapat membuat rule seperti berikut:

¹⁸⁹ <http://php.net/manual/en/function.date-parse-from-format.php>

¹⁹⁰ <http://php.net/manual/en/datetime.createfromformat.php>

```
'father' => 'required',
'mother' => 'required|different:father'
```

Ketika isian `father` dan `mother` sama, ini error yang akan kita dapatkan:

```
{
  "mother": [
    "The mother and father must be different."
  ]
}
```

Digits

Rule ini bisa kita gunakan untuk memastikan sebuah input angka memiliki jumlah angka yang sesuai. Misalnya, kita ingin agar isian `price` selalu 4 digit (ribuan), maka kita tulis rule seperti ini:

```
'price' => 'digits:4'
```

Ketika kita mengisi dengan angka yang memiliki jumlah digit yang tidak 4, misalnya 300, kita akan mendapat error:

```
{
  "price": [
    "The price must be 4 digits."
  ]
}
```

Digits Between

Rule ini merupakan gabungan dari rule `digits` dan `between`. Kita gunakan jika menginginkan isian angka berada dalam batasan jumlah digit tertentu. Misalnya, isian `price` antar 4 dan 8 digit, kita buat rule:

```
'price' => 'digits_between:4,8'
```

Ketika kita mengisi dengan angka yang memiliki jumlah digit yang tidak sesuai, error ini yang akan kita dapatkan:

```
{
  "price": [
    "The price must be between 4 and 8 digits."
  ]
}
```

E-Mail

Rule ini bisa kita gunakan untuk memastikan isian dari user merupakan string yang memiliki format email yang benar. Jadi, dia cuman mengecek formatnya aja. Bukan memastikan email itu benar-benar valid.

Bisa kita gunakan seperti ini:

```
'client_email' => 'email'
```

Jika isian `client_email` tidak memiliki format email yang valid, error ini yang akan kita dapatkan:

```
{
  "client_email": [
    "The client email must be a valid email address."
  ]
}
```

Exists (Database)

Rule ini sangat bermanfaat untuk memastikan isian sebuah field berada di database. Ada beberapa cara menggunakan rule ini yang akan kita bahas dibawah.

Ketika nama field di form sama dengan di table

Ini merupakan teknik pertama, misalnya kita membuat isian `genre` dan harus sama dengan isian pada field `genre` pada table `musics`. Untuk memudahkan buatlah table dengan struktur dan data seperti berikut:

id	genre	status	description
1	dangdut	under-review	musik rakyat
2	nasyid	under-review	musik positif
3	pop	published	musik remaja
4	rock	published	NULL

Table `musics`

Kita dapat menulis rule seperti berikut:

```
'genre' => 'exists:musics'
```

Jika kita mengisi selain dangdut, nasyid, pop dan rock, kita akan mendapat error:

```
{
  "genre": [
    "The selected genre is invalid."
  ]
}
```

Ketika nama field di form berbeda dengan di table

Misalnya kita menggunakan field `jenis_musik` di form. Untuk mengeceknya ke field `genre` di table `musics`, ini yang kita tulis:

```
'jenis_musik' => 'exists:musics,genre'
```

Mengecek isi field lain

Kita juga dapat mengecek isi field lain, misalnya kita hanya akan menggunakan genre yang field `description` nya terisi:

```
'jenis_musik' => 'exists:musics,genre,NOT_NULL'
```

Kini, yang valid sebagai isian `jenis_music` hanya dangdut, nasyid dan pop. Tentunya, kita juga dapat menggunakan `NULL` sebagai kebalikan dari `NOT_NULL` untuk menggunakan genre yang field `description` nya tidak diisi.

Menggunakan query tambahan ke field lain

Kita juga dapat menggunakan `where` sederhana. Misalnya, hanya ingin menggunakan genre yang isian `status` nya `published`. Ini yang kita tulis:

```
'jenis_musik' => 'exists:musics,genre,status,published'
```

Kini, isian yang valid hanya pop dan rock. Tentunya, kita dapat menggunakan `where not` menggunakan tanda `!`. Misalnya seperti berikut:

```
'jenis_musik' => 'exists:musics,genre,status,!published'
```

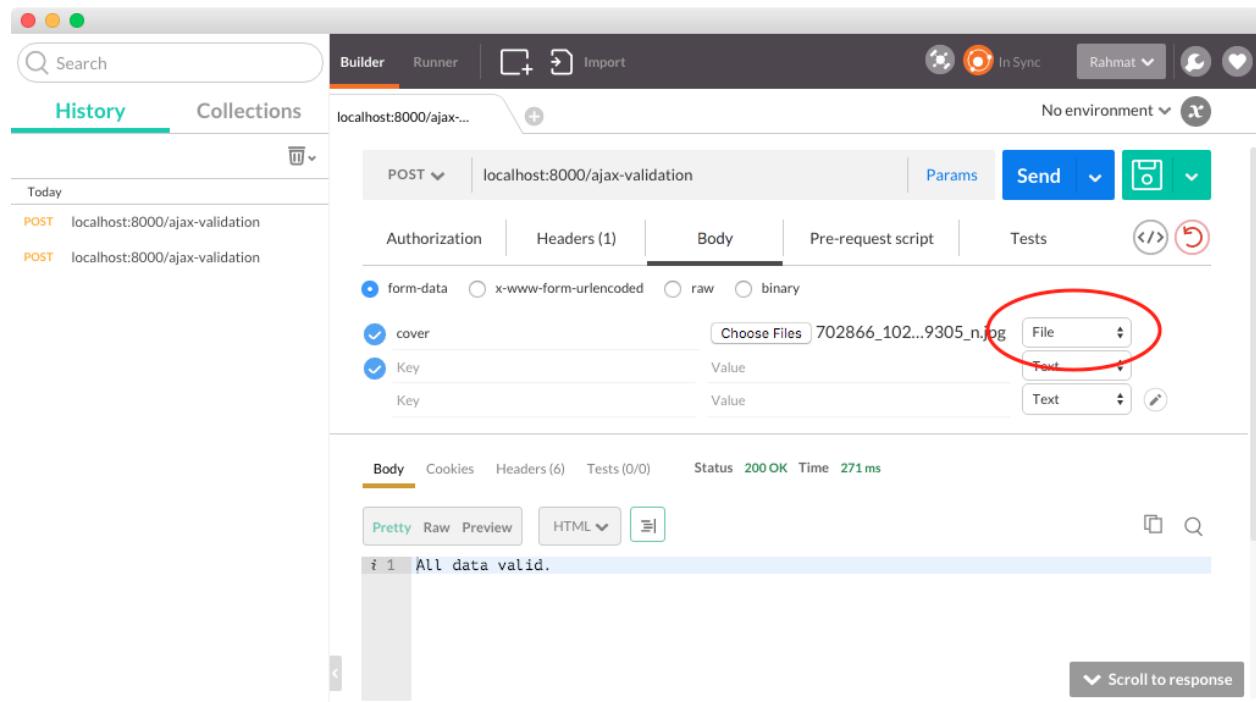
Kini, isian yang valid hanya dangdut dan nasyid.

Image (File)

Rule ini bisa kita gunakan untuk memastikan field yang diinput oleh user merupakan sebuah file. Ekstensi file yang valid untuk rule ini adalah jpeg, png, bmp, gif, dan svg. Biasanya rule ini digunakan bersamaan dengan rule `max` untuk membatasi ukuran image yang diupload. Misalnya, kita dapat membuat sebuah field `cover` yang harus berupa image dengan ukuran maksimum 2MB, rule yang kita buat seperti ini:

```
'cover' => 'image|max:2048'
```

Untuk mengetes upload image di Postman, ketika dapat memilih input dengan tipe `file`:



The screenshot shows the Postman application interface. In the center, there is a request configuration window for a POST request to 'localhost:8000/ajax-validation'. The 'Body' tab is selected, showing a 'form-data' section. A file named '702866_102...9305_n.jpg' is uploaded under the key 'cover'. Below the file input, there are two empty text inputs labeled 'Value' and 'Text'. A red circle highlights the 'File' dropdown menu next to the file input. At the bottom of the request window, the status is shown as '200 OK' with a response time of '271 ms'. The response body is displayed as 'All data valid.'.

Upload file dengan Postman

Jika kita tidak mengupload image, ini error yang kita dapatkan:

```
{
  "cover": [
    "The cover must be an image.",
    "The cover may not be greater than 2048 kilobytes."
  ]
}
```

Error "The cover may not be greater than 2048 kilobytes." muncul karena disini saya mencoba mengupload file mp3 yang ukurannya 4MB.

In

Rule ini pernah kita gunakan pada contoh sebelumnya untuk membatasi nama artis yang bisa kita input (Iwan Fals dan Afgan). Ini rule yang pernah kita buat:

```
'artist' => 'in:Iwan Fals,Afgan'
```

Ketika kita mencoba mengisi artis yang tidak didalam list tersebut, ini error yang akan kita dapatkan:

```
{
  "artist": [
    "The selected artist is invalid."
  ]
}
```

Integer

Rule ini bisa kita gunakan untuk memastikan input dari user adalah sebuah integer (bilangan bulat). Misalnya kita tulis seperti ini:

```
'age' => 'integer'
```

Jika kita mencoba mengisi `age` dengan non-integer, misalnya bilangan desimal atau huruf, ini error yang akan kita dapatkan:

```
{
  "age": [
    "The age must be an integer."
  ]
}
```

Untuk mengetahui ukuran minimum dan maksimum integer di PHP, cek jawaban dari [pertanyaan di stackoverflow¹⁹¹](#) ini.

IP Address

Rule ini kita gunakan untuk memastikan field yang dikirimkan user merupakan ip address yang valid. Misalnya kita buat seperti ini:

¹⁹¹ <http://stackoverflow.com/questions/670662/whats-the-maximum-size-for-an-int-in-php>

```
'client_address' => 'ip'
```

Jika kita tidak mengisi `client_address` dengan format ip yang valid, error yang akan kita dapatkan akan seperti berikut:

```
{
  "client_address": [
    "The client address must be a valid IP address."
  ]
}
```

JSON

Rule ini untuk memastikan input yang kita terima merupakan string JSON yang valid. Misalnya kita buat rule seperti berikut:

```
'user_property' => 'json'
```

Jika kita mengisi `user_property` dengan data JSON yang salah, ini error yang akan kita dapatkan:

```
{
  "user_property": [
    "The user property must be a valid JSON string."
  ]
}
```



Rule ini akan menganggap angka dan huruf yang diapit tanda kutip sebagai JSON yang valid.

Max

Rule ini bisa digunakan untuk membatasi ukuran field. Bisa digunakan untuk field string, integer dan file. Teknik yang digunakan untuk menentukannya ukuran field sama dengan rule `size`.

Misalnya kita buat rule seperti ini:

```
'name' => 'alpha_num|max:40'
```

Jika kita mencoba mengisi `name` dengan jumlah karakter lebih dari 40, maka ini error yang akan kita dapatkan:

```
{
  "name": [
    "The name may not be greater than 40 characters."
  ]
}
```

MIME Types (File)

Rule ini bisa kita gunakan untuk memastikan isian sebuah field merupakan file dengan tipe tertentu. Untuk mengetahui berbagai tipe file yang didukung oleh rule ini, kunjungi link berikut <http://svn.apache.org/repos/asf/httpd/httpd/trunk/docs/conf/mime.types>¹⁹².

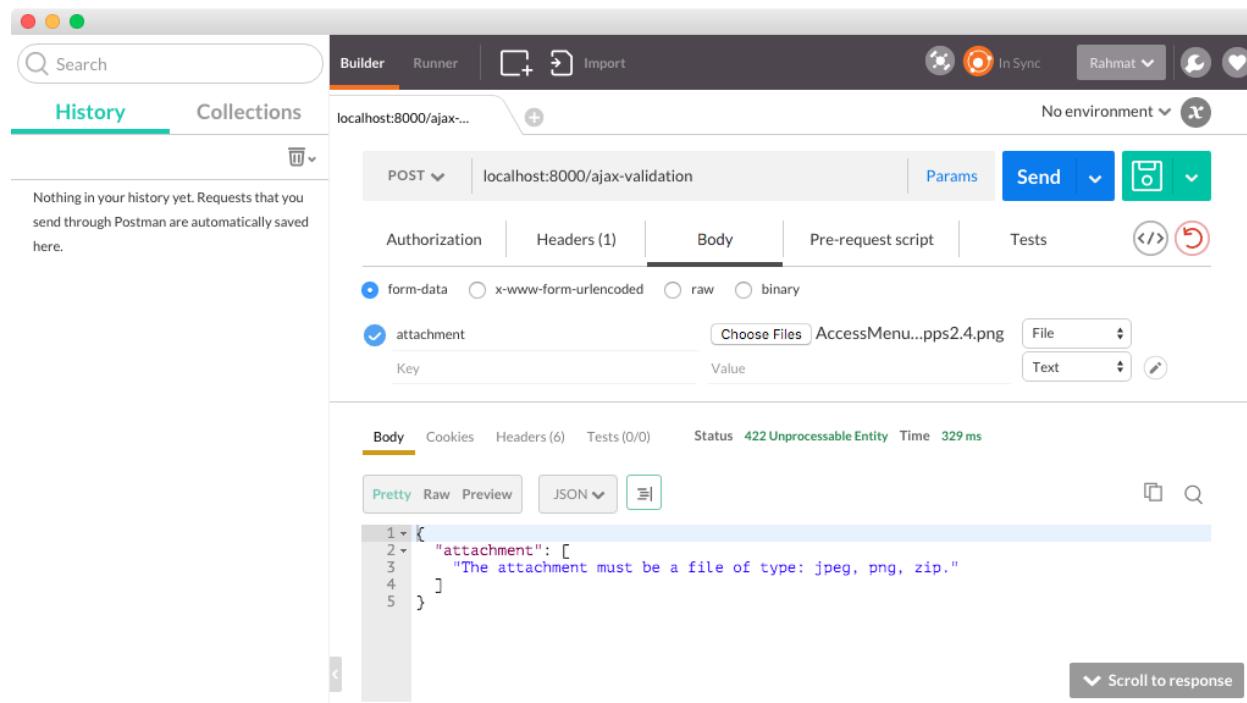
Misalnya, kita ingin membuat field `attachment` dengan tipe jpeg, png atau zip dengan ukuran maksimum 10MB. Ini yang akan kita tulis:

```
'attachment' => 'mimes:jpeg,png,zip|max:10240'
```

Ini error yang akan kita dapatkan ketika mencoba mengirim file selain tipe yang diizinkan. {lang="php", linenos=off} { "attachment": ["The attachment must be a file of type: jpeg, png, zip."] }

Rule ini menentukan tipe file dengan membaca konten dari file tersebut, **bukan hanya dengan membaca ekstensi file**. Disini saya dengan me-rename ekstensi dmg menjadi png, maka kita akan tetap mendapat error:

¹⁹²<http://svn.apache.org/repos/asf/httpd/httpd/trunk/docs/conf/mime.types>



Mencoba merubah ekstensi file

Min

Rule ini kebalikan dari rule `max`. Penggunaan sama dengan rule `max`.

Not In

Rule ini kebalikan dari rule `in`. Penggunaannya sama dengan rule `in`, hanya saja menggunakan keyword `not_in`.

Numeric

Mirip dengan rule `integer`, bedanya rule ini mengizinkan angka desimal. Penggunaannya sama dengan rule `integer`.

Regular Expression

Bila Anda tipe orang yang suka dengan regex, maka ini sahabat sejati Anda. Rule ini memungkinkan kita untuk memvalidasi sebuah field berdasarkan syntax regex tertentu.

Misalnya, kita akan menggunakan regex `/d[aeio]ni/` untuk membatasi input `child` yang mengandung kata `dani`, `deni`, `dini`, dan `doni`. Ini rule yang akan kita tulis:

```
'child' => 'regex:/d[aeiou]ni/'
```

Jika kita tidak memasukan input yang sesuai dengan regex tersebut, ini error yang akan kita dapatkan:

```
{
  "child": [
    "The child format is invalid."
  ]
}
```

Required

Rule ini sudah sering kita pakai untuk memastikan bahwa sebuah input diisi. Saya rasa tidak perlu dibuat contoh lagi.

Required If

Masih ingat dengan penggunaan sometimes? Nah, menggunakan required_if kita akan memiliki fitur yang hampir sama dengan sometimes. Misalnya kita ingin agar field reason diisi jika isian genre adalah rock. Ini rule yang kita tulis:

```
'genre' => 'in:pop,rock,nasyid',
'reason' => 'required_if:genre,rock'
```

Kita juga bisa menggunakan beberapa value misalnya:

```
'genre' => 'in:pop,rock,nasyid',
'reason' => 'required_if:genre,pop,rock'
```

Akan mewajibkan field reason diisi jika isian genre adalah pop atau rock.

Jika kita mengisi pop pada genre, tapi tidak mengisi field reason, maka ini error yang akan kita dapatkan:

```
{  
  "reason": [  
    "The reason field is required when genre is pop."  
  ]  
}
```

Required Unless

Ini kebalikan dari rule `required_if`. Misalnya, kita tidak ingin isian `reason` diisi jika isian `genre` adalah nasyid. Maka ini rule yang akan kita tulis:

Seperti penggunaan `required_if`, kita juga dapat memberikan banyak opsi pada isian value.

Ini error yang akan kita dapatkan ketika mengisi `genre` selain selain `nasyid` dan tidak mengisi field `reason`:

```
{  
  "reason": [  
    "The reason field is required unless genre is in nasyid."  
  ]  
}
```

Required With

Rule ini kita gunakan jika ingin sebuah field diisi ketika field lain diisi. Mirip dengan `required_if` tapi dengan value `NOT_NULL`. Misalnya, kita ingin membuat agar field `artist` diisi jika field `song` diisi.

```
'song' => 'alpha_num',  
'artist' => 'required_with:song'
```

Jika kita mengisi field `song` tapi `artist` tidak kita isi, ini error yang kita dapatkan:

```
{  
  "artist": [  
    "The artist field is required when song is present."  
  ]  
}
```

Kita juga dapat menggunakan beberapa field untuk rule ini. Misalnya field `artist` harus diisi jika field `song` **ATAU** `album` diisi:

```
'song' => 'alpha_num',  
'album' => 'alpha_num',  
'artist' => 'required_with:song,album'
```

Ketika kita mengisi field `song` atau `album` dan tidak mengisi field `artist`, ini error yang akan kita dapatkan:

```
{  
  "artist": [  
    "The artist field is required when song / album is present."  
  ]  
}
```

Required With All

Pada contoh `required_with`, kita menggunakan beberapa field untuk membuat sebuah field `required` jika **salah satu** field yang menjadi parameter telah terisi. Rule ini, mirip dengan rule tersebut, bedanya pada rule ini **semua** field yang menjadi parameter harus terisi.

Misalnya, kita ingin agar isian `track_no` diisi jika field `song` **DAN** `album` terisi. Ini yang kita tulis:

```
'song' => 'alpha_num',  
'album' => 'alpha_num',  
'track_no' => 'required_with_all:song,album'
```

Ketika kita mengisi field `song` dan `album` tapi tidak mengisi field `track_no`, ini error yang akan kita dapatkan:

```
{  
    "track_no": [  
        "The track no field is required when song / album is present."  
    ]  
}
```

Required Without

Rule ini kebalikan dari rule `required_with`, keyword yang digunakan adalah `required_without`. Penggunaannya sama dengan `required_with`.

Required Without All

Rule ini kebalikan dari rule `required_with_all`, keyword yang digunakan adalah `required_without_all`. Penggunaannya sama dengan `required_with_all`.

Same

Rule ini merupakan kebalikan dari rule `different`, untuk memastikan isian sebuah field sama dengan field lain. Keyword yang digunakan adalah `same`. Penggunaannya sama dengan penggunaan rule `different`.

Size

Rule ini bisa kita gunakan untuk memastikan ukuran sebuah field sesuai dengan ukuran yang kita inginkan. Penggunaan rule ini biasanya digabungkan dengan rule lain:

- `alpha` atau `alpha_num`: akan menghitung jumlah karakter.
- `integer` atau `numeric`: akan menghitung total angka (bukan jumlah digit)
- `image` atau `mime`: akan menghitung ukuran file dalam KB.

Misal kita ingin menerima input `token` yang harus 8 karakter, ini rule yang kita tulis:

```
'token' => 'alpha_num|size:8'
```

Jika kita mengisi isian `token` selain 8 karakter, ini error yang akan kita dapat:

```
{  
    "token": [  
        "The token must be 8 characters."  
    ]  
}
```

String

Rule ini biasanya digunakan input yang kita gunakan adalah string, bukan array. Misalnya kita buat seperti ini:

```
'address' => 'string'
```

Jika kita mencoba mengisi `address` dengan array, misalnya dengan memberi nama field `address[0]`, kita akan mendapat error:

```
{  
    "address": [  
        "The address must be a string."  
    ]  
}
```

Timezone

Rule ini biasanya digunakan ketika ingin agar user memasukan timezone dimana dia tinggal. Rule akan mengecek validitas input dengan menggunakan fungsi `timezone_identifiers_list`¹⁹³ di PHP. Contoh isian yang valid untuk field ini misalnya Asia/Jakarta, Asia/Tokyo, America/New_York, dll. Untuk melihat semua isian yang valid, cek di <http://php.net/manual/en/timezones.php>¹⁹⁴.

Misalnya kita dapat membuat rule seperti ini:

```
'your_timezone' => 'timezone'
```

Jika kita mencoba mengisi Asia/Bandung, yang jelas tidak valid, ini error yang akan kita dapatkan:

¹⁹³ <http://php.net/manual/en/function.timezone-identifiers-list.php>

¹⁹⁴ <http://php.net/manual/en/timezones.php>

```
{
  "your_timezone": [
    "The your timezone must be a valid zone."
  ]
}
```

Unique (Database)

Rule ini merupakan kebalikan dari rule `exists`. Penggunaannya pun sama. Mari kita gunakan kembali table `musics` yang kita buat pada contoh rule `exists`.

TABLES		Search: id =		
		id	genre	status
	migrations	1	dangdut	under-review
	musics	2	nasyid	under-review
	password_resets	3	pop	published
	users	4	rock	published
				NULL

Table `musics`

Misalnya kita hanya mengizinkan isian `genre` yang belum ada di database:

```
'genre' => 'unique:musics'
```

Ketika kita mengisi `genre` dengan nilai yang sudah ada, misalnya `pop`, ini error yang akan kita dapatkan:

```
{
  "genre": [
    "The genre has already been taken."
  ]
}
```

Tentunya, kita juga dapat menggunakan nama field yang berbeda:

```
'jenis_musik' => 'unique:musics,genre'
```

Kita juga bisa mendisable cek untuk field tertentu. Misalnya, ketika kita hendak mengupdate `genre` `pop` dengan mass assignment, tentunya kita tidak ingin memasukan record dengan id yang sama. Kita bisa menulis seperti berikut:

```
'jenis_musik' => 'unique:musics,genre,id,' . $music->id
```

Syntax `$music->id` disini kita simulasikan dari record `music` yang sedang di update. Menggunakan rule diatas, Laravel mengasumsikan `$music->id` nilainya akan sama dengan field `id` di table. Jika kita menggunakan primary key selain id, misalnya `id_musik`, rule diatas akan kita tulis seperti ini:

```
'jenis_musik' => 'unique:musics,genre,id,' . $music->id . ',id_musik'
```

Kita juga dapat membuat menambah query `where` pada field lain. Misalnya, kita hanya ingin mengecek pada record yang isian `status` nya `published`. Ini yang kita tulis

```
'jenis_musik' => 'unique:musics,genre,id,NULL,id,status,published'
```

URL

Rule ini bisa kita gunakan untuk mengecek apakah input yang diberikan oleh user merupakan sebuah url dengan format yang valid. Validitas URL akan ditentukan oleh fungsi `filter_var`¹⁹⁵ di PHP. Lebih detailnya, fungsi `filter_var` akan menggunakan filter `FILTER_VALIDATE_URL`¹⁹⁶ yang menggunakan definisi di <http://www.faqs.org/rfcs/rfc2396.html>

Misalnya kita buat rule seperti ini:

```
'article_link' => 'url'
```

Bila kita mengisi field `article_link` dengan url yang tidak valid, ini error yang akan kita dapatkan:

```
{
    "article_link": [
        "The article link format is invalid."
    ]
}
```

¹⁹⁵ <http://php.net/manual/en/function.filter-var.php>

¹⁹⁶ <http://php.net/manual/en/filter.filters.validate.php>

¹⁹⁷ <http://www.faqs.org/rfcs/rfc2396.html>

Membuat Custom Rule

Jika rule di Laravel ini tidak cukup untuk kebutuhan kita, kita bahkan dapat membuat rule sendiri. Ini pernah saya lakukan ketika dikerjaan butuh memvalidasi input password user dengan password ter-*hash* yang tersimpan di database. Dalam webapp, biasanya kita melakukan ini ketika user hendak melakukan tindakan yang sangat penting, misalnya menambah payment.

Mari kita buat rule `passcheck` untuk mendemokannya. Hasil akhir dari rule ini adalah kita dapat menggunakannya seperti berikut:

```
'password' => 'passcheck:<password-dari-db>'
```

Untuk membuatnya kita akan menggunakan method `Validator::extend()`. Method ini harus kita tambahkan pada sebuah service provider di method `boot`. Mari kita tambahkan di `/app/Providers/AppServiceProvider.php`:

/app/Providers/AppServiceProvider.php

```
<?php  
....  
class AppServiceProvider extends ServiceProvider  
{  
    ....  
    public function boot()  
    {  
        \Validator::extend('passcheck', function ($attribute, $value, $parameters)  
s) {  
            return \Hash::check($value, $parameters[0]);  
        });  
    }  
    ....  
}
```

Method `Validator::extend()` ini menerima 2 parameter, nama rule dan closure yang berisi logika dari rule tersebut. Pada closure, kita akan menerima 3 parameter:

- `$attribute`: nama field yang sedang divalidasi
- `$value`: nilai dari field yang sedang divalidasi
- `$parameters`: parameter dari rule

Closure ini hanya boleh mengembalikan nilai boolean.

Disini, kita menggunakan `\Hash::check($value, $parameters[0])` untuk mengecek nilai hash dari field yang divalidasi dengan parameter pertama (password user di database).

Mari kita tambahkan pesan error default untuk rule ini:

/resources/lang/en/validation.php

```
<?php
return [
    ...
    'passcheck' => 'Isian :attribute tidak sesuai dengan password yang tersimpan\
di database.',
    ...
]
```

Untuk mendemokannya, mari kita buat user dengan seeder. Tambahkan syntax berikut pada `DatabaseSeeder`:

/database/seeds/DatabaseSeeder.php

```
<?php
...
class DatabaseSeeder extends Seeder
{
    ...
    public function run()
    {
        App\User::create([
            'name' => 'doni',
            'email' => 'doni@gmail.com',
            'password' => bcrypt('rahasia')
        ]);
        ...
    }
}
```

Jalankan `php artisan migrate:refresh --seed` dan pastikan pada table users terdapat isian seperti berikut:

TABLES		Search: id = Search					
		id	name	email	password	remember_token	created_at
		1	doni	doni@gmail.com	\$2y\$10\$SC2Yoy4Fb2eP86aL9Ycd3...	NULL	2023-01-01 12:00:00

Sample user

Untuk memudahkan demo, kita akan menggunakan password untuk user Doni yang telah kita buat. Pada aplikasi nyata, biasanya akan menggunakan password dari user yang sedang login menggunakan syntax `Auth::user()->password`.

Masukan syntax ini pada bagian validasi di method `testAjax` di `HomeController`:

/app/Http/Controllers/HomeController.php

```
public function testAjax(Request $request)
{
    $this->validate($request, [
        'password' => 'passcheck:' . \App\User::where('email', 'doni@gmail.com')\ 
->first()->password
    ]);
    return 'All data valid.';
}
```

Jika kita mencoba mengakses menggunakan postman dan mengisi password selain rahasia, maka ini error yang akan kita dapatkan:

```
{
    "password": [
        "Isian password tidak sesuai dengan password yang tersimpan di database."
    ]
}
```

Validasi dengan Form Request

Jika aplikasi kita sudah besar, akan sangat baik jika kita memisahkan bagian untuk validasi diluar controller. Inilah yang akan kita lakukan dengan fitur Form Request. Menggunakan teknik ini, validasi akan dilakukan pada class terpisah.

Mari kita coba demokan dengan memindahkan logic untuk validasi ketika kita menambah di method `store` pada `SongsController`. Pertama, kita buat dulu form request dengan perintah:

```
php artisan make:request StoreSongRequest
```

Disini, kita menggunakan nama `StoreSongRequest` file yang digenerate akan disimpan di `/app/Http/Requests/StoreSongRequest.php`. Secara default akan ada dua method pada file tersebut `authorize` dan `rules`.

Method `authorize` digunakan untuk melakukan authorisasi akses. Masih ingat dengan penggunaan `Ability` di bab sebelumnya? Nah, itulah yang dilakukan method ini. Karena kita tidak akan melakukan authorisasi, kita cukup memberikan `true` sebagai hasil dari method ini:

/app/Http/Requests/StoreSongRequest.php

```
public function authorize()
{
    return true;
}
```

Method `rules` kita gunakan untuk mendefinisikan rule validasi untuk request ini. Kita dapat copy paste rule nya dari `SongsController` dengan sedikit modifikasi sehingga akan seperti ini:

/app/Http/Requests/StoreSongRequest.php

```
public function rules()
{
    $rules = [
        'title' => 'required',
        'artist' => 'required|in:Iwan Fals,Afgan|min:3',
        'album' => 'sometimes|required|alpha_num',
        'personil.*.name' => 'alpha_num'
    ];
    if ($this->has('album')) $rules['track_no'] = 'required|integer|min:0';
    return $rules;
}
```

Rule ini sama dengan yang kita buat di `SongsController`. Perbedaannya adalah kita menggunakan `$this->has()` untuk mengecek apakah field `album` memiliki isi (di Controller kita menggunakan `$request->has()`).

Pada validasi yang kita buat, kita juga menggunakan pesan error custom. Di form request, kita dapat mendefinisikannya di method `messages()`. Secara default method ini belum ada pada file `StoreSongRequest` yang digenerate.

/app/Http/Requests/StoreSongRequest.php

```
public function messages()
{
    return [
        'title.required' => 'Isi dengan judul lagu yang populer bro.',
        'album.required' => 'Album dari lagu ini mesti diisi ya.',
        'track_no.required' => 'Nomor tracknya diisi ya.',
        'personil.*.name.alpha_num' => 'Nama personil mesti valid.'
    ];
}
```

Sudah deh. Sekarang kita tinggal “membersihkan” method `store` di `SongsController` agar menggunakan form request ini:

/app/Http/Controllers/SongsController.php

```
<?php
...
use App\Http\Requests\StoreSongRequest;
class SongsController extends Controller
{

    public function store(StoreSongRequest $request)
    {
        \App::setLocale('id');
        return 'Lagu ' . $request->title . ' oleh ' . $request->artist . ' telah\
disimpan.';
    }
}
```

Cobalah isi form pembuatan di `/songs/new`, pastikan semua validasinya tetap berfungsi.

localhost:8000/songs/new

Rahmat

Track

Judul

Isi dengan judul lagu yang populer bro.

Album

Penyanyi

Penyanyinya mesti diisi ya..

Anggota

Personil 1 :

Nama personil mesti valid.

Personil 2 :

Personil 3 :

Personil 4 :

Personil 5 :

Menggunakan form request untuk validasi

Keren kan? :)

Menggunakan form request akan membuat syntax di controller lebih ringkas. Dengan teknik ini, logic di controller akan lebih terarah, yaitu untuk mengarahkan request.



Source code dari latihan di bab ini bisa didapat di [Bitbucket](#)¹⁹⁸

¹⁹⁸<https://bitbucket.org/rahmatawalidin/sample-validasi/commits/all>



Ringkasan

Membangun aplikasi yang bebas dari celah keamanan memang berat. Dengan fitur validasi di Laravel ini, seharusnya dapat mengurangi jumlah pekerjaan kita untuk mengamankan aplikasi. Semua bab yang telah kita pelajari, belum afdhol jika belum kita buat sample projectnya. Mari kita buat sample project untuk menggunakan skill yang kita pelajari dari 14 bab ini di bab terakhir.

Semangat! :)

Case Study: Membangun CMS Toko Online

Membangun sebuah web dari 0 sampai jadi akan memberikan kita gambaran menyeluruh bagaimana menggunakan Laravel. Pada bab ini, kita akan menggunakan teknik-teknik yang telah kita pelajari pada bab sebelumnya untuk membuat sebuah web e-commerce. Dalam membangun web ini, selain menggunakan teknik yang telah kita pelajari pada bab sebelumnya, kita juga akan mempelajari beberapa fitur laravel diantaranya:

- Session
- Pagination
- Cookie

Untuk namanya, saya akan menamai aplikasi ini “Reselia”. Kenapa Reselia? Sebenarnya tidak ada yang spesial. Cuman, ini salah satu nama yang dulu pernah diberikan oleh temen saya untuk sebuah startup di bidang reseller online shop. Sip.

Penjelasan project

Sebelum kita memulai membangun project ini, mari kita bahas dulu skema sistem yang akan kita buat. Aplikasi e-commerce yang akan kita buat disini, tentunya tidak akan sekompelks lazada, tokopedia atau situs e-commerce besar lainnya. Setidaknya beberapa fitur dasar dari sebuah e-commerce akan kita buat disini:

- Manajemen produk dan kategori oleh admin
- Pendaftaran customer
- Katalog produk untuk customer
- Cart
- Checkout
- Penentuan harga ongkos kirim menggunakan API pihak ketiga
- Cek status pengiriman barang

Sengaja fiturnya saya batasi, agar pembuatan aplikasi ini bisa lebih cepat. Adapun untuk fitur-fitur lainnya yang lebih kompleks, saya serahkan kepada pembaca untuk mengembangkannya. Tidak lupa, dibagian akhir bab ini terdapat list ide fitur yang *nice to have*.

Dalam mengembangkan project ini, kita tidak akan menjelaskan struktur aplikasi lengkap diawal. Pengembangan aplikasi ini akan kita buat per fitur. Setiap kali kita akan membuat fitur, akan kita bahas dulu akan seperti apa hasil akhir fiturnya, struktur table yang dibutuhkan (jika ada), dll. Jadi, *planning* untuk tiap fitur akan kita lakukan pada pembahasan masing-masing.

Menjalankan sample code

Untuk mendapatkan gambaran hasil akhir aplikasi, saya sarankan untuk mendownload source code dari bab ini. Silahkan ikuti langkah berikut:

1. Download source code dari <https://bitbucket.org/rahmatawaludin/sample-reselia>¹⁹⁹
2. Jalankan `composer install`
3. Jalankan `npm install`
4. Konfigurasi database (copy file `.env` untuk menyimpan nama db, user dan password)
5. (optional) buat akun starter (gratis) di rajaongkir.com. Dapatkan API key di <http://rajaongkir.com/akun/panel> dan isikan di file `.env` dengan key `RAJAONGKIR_APIKEY`
6. Jalankan `php artisan key:generate`
7. Jalankan `php artisan migrate:refresh --seed` (pastikan ada koneksi internet)
8. Jalankan `gulp`
9. Setup domain di homestead atau gunakan `php artisan serve`.

Persiapan Project

Mari kita mulai dengan membuat project Laravel baru dengan perintah:

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

Di bab ini, versi Laravel yang akan saya gunakan adalah 5.2.

Konfigurasi juga database sesuai environment yang digunakan.

Setelah berhasil terinstal, kita download dependency untuk elixir dengan perintah:

¹⁹⁹ <https://bitbucket.org/rahmatawaludin/sample-reselia>

```
npm install
```

Kita akan menggunakan bootstrap yang sudah terinstall oleh Elixir, untuk itu hilangkan komen pada file resources/assets/sass/app.scss:

resources/assets/sass/app.scss

```
@import "node_modules/bootstrap-sass/assets/stylesheets/bootstrap";
```

Selanjutnya kita compile file scss tersebut dengan:

```
gulp
```

Authentikasi & Level User

Di Reselia kita akan menggunakan RBAC sederhana untuk membedakan admin dan customer. Caranya, kita menggunakan authentikasi bawaan Laravel dan menambah field role. Pada saat user mendaftar ke aplikasi kita akan set role tersebut dengan isian `customer`. Sementara untuk role `admin` kita akan buat user default menggunakan seeder dan menggunakan email `admin@gmail.com`.

Mari kita tambahkan field `role` ke migration untuk table users:

database/migrations/2014_10_12_000000_create_users_table.php

```
<?php
...
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            ...
            $table->string('role');
        });
    }
    ...
}
```

Untuk memudahkan dalam pengembangan, mari kita buat sample user dengan menggunakan seeder. Jalankan perintah berikut untuk membuat file seeder:

```
php artisan make:seeder UsersSeeder
```

Pada file yang dihasilkan isi method `run()` dengan:

database/seeds/UsersSeeder.php

```
....  
public function run() {  
    // sample admin  
    App\User::create([  
        'name' => 'Admin',  
        'email' => 'admin@gmail.com',  
        'password' => bcrypt('secret'),  
        'role' => 'admin'  
    ]);  
  
    // sample customer  
    App\User::create([  
        'name' => 'customer',  
        'email' => 'customer@gmail.com',  
        'password' => bcrypt('secret'),  
        'role' => 'customer'  
    ]);  
}  
....
```

Ubah isian method `run()` pada `database/seeds/DatabaseSeeder.php` menjadi:

database/seeds/DatabaseSeeder.php

```
public function run()  
{  
    $this->call(UsersSeeder::class);  
}
```

Setelah sample user siap, kita migrate dan seed:

```
php artisan migrate:refresh --seed
```

Pastikan sample user sudah muncul di database dengan role yang sesuai:

Sample user

Kita akan menggunakan authentikasi bawaan Laravel, mari kita gunakan generator untuk mendapatkan template aplikasi berikut authentikasinya:

```
php artisan make:auth
```

Setelah perintah ini dijalankan, akan terdapat beberapa file baru:

- app/Http/Controllers/HomeController.php
- resources/views/auth/emails/password.blade.php
- resources/views/auth/login.blade.php
- resources/views/auth/passwords/email.blade.php
- resources/views/auth/passwords/reset.blade.php
- resources/views/auth/register.blade.php
- resources/views/home.blade.php
- resources/views/layouts/app.blade.php

Dan beberapa file yang dimodifikasi:

- app/Http/routes.php
- resources/views/welcome.blade.php

Pada file route akan ada baris berikut:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    Route::auth();
    Route::get('/home', 'HomeController@index');
});
```

Syntax diatas digunakan untuk membuat route untuk fitur authentikasi dasar berikut:

- GET|HEAD home
- GET|HEAD login
- POST login
- GET|HEAD logout
- POST password/email
- POST password/reset
- GET|HEAD password/reset/{token?}
- GET|HEAD register
- POST register

Ketika User mendaftar ke Reselia, kita akan set role-nya menjadi customer. Caranya, kita ubah method create pada `app/Http/Controllers/Auth/AuthController.php`:

app/Http/Controllers/Auth/AuthController.php

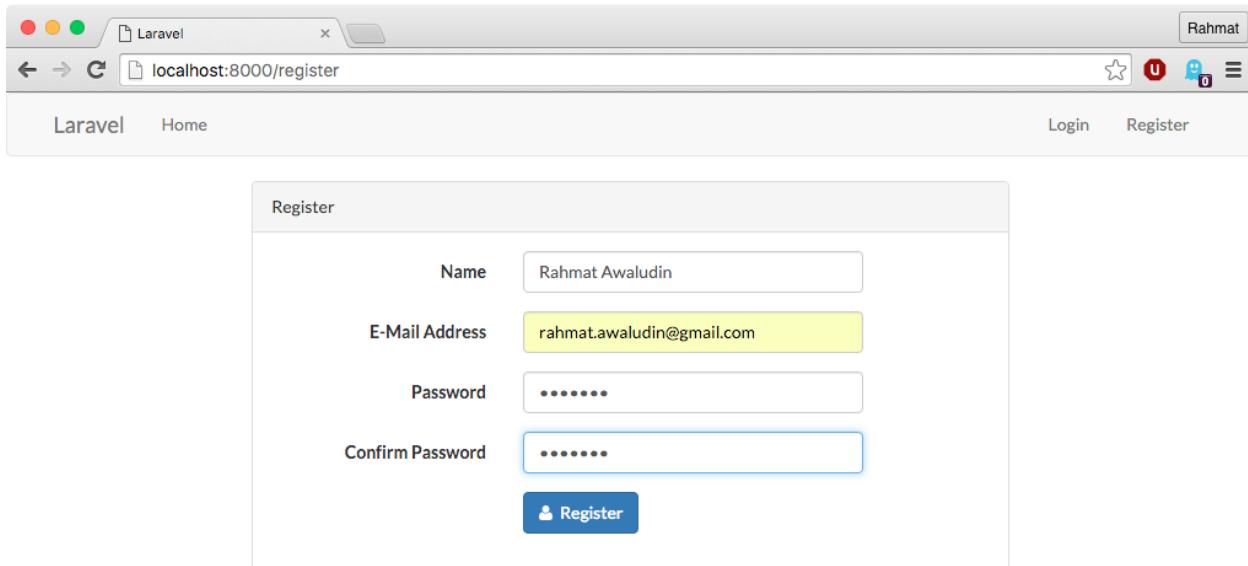
```
....  
protected function create(array $data)  
{  
    $user = User::create([  
        'name' => $data['name'],  
        'email' => $data['email'],  
        'password' => bcrypt($data['password'])  
    ]);  
    $user->role = 'customer';  
    $user->save();  
    return $user;  
}  
....
```

Karena kita sudah memiliki route `/home` mari kita ubah default route setelah login ke route tersebut:

app/Http/Controllers/Auth/AuthController.php

```
....  
protected $redirectTo = '/home';  
....
```

Kini, cobalah mendaftar sebagai customer dengan mengisi data user di halaman `/register`. Setelah berhasil, cobalah untuk login.



Register

Name: Rahmat Awaludin

E-Mail Address: rahmat.awaludin@gmail.com

Password: *****

Confirm Password: *****

Register

Mendaftar sebagai customer

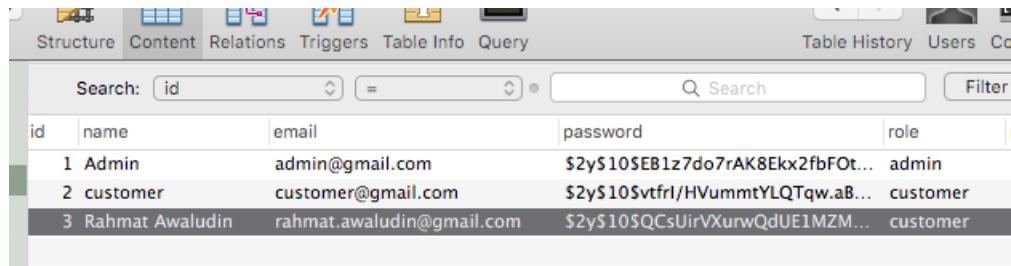


Welcome

Your Application's Landing Page.

Berhasil login

Pastikan di database field `role` terisi dengan `customer`:



id	name	email	password	role
1	Admin	admin@gmail.com	\$2y\$10\$EB1z7do7rAK8Ekx2fbFOt...	admin
2	customer	customer@gmail.com	\$2y\$10\$vtfrl/HVummtYLQTqw.aB...	customer
3	Rahmat Awaludin	rahmat.awaludin@gmail.com	\$2y\$10\$QCSCsUirVXurwQdUE1MZM...	customer

Role customer berhasil diisi

Disini saya tidak melakukan banyak perubahan pada tampilan authentikasi. Jika diinginkan, silahkan ubah sesuai kebutuhan.

Mengatur Guard dan Middleware

Untuk membatasi akses kita akan membuat ability `admin-access` dan `customer-access` yang akan menentukan akses berdasarkan role yang sesuai. Tambahkan baris berikut pada method `boot` di `app/Providers/AuthServiceProvider.php`:

app/Providers/AuthServiceProvider.php

```
public function boot(GateContract $gate)
{
    $this->registerPolicies($gate);

    $gate->define('admin-access', function($user) {
        return $user->role == 'admin';
    });

    $gate->define('customer-access', function($user) {
        return $user->role == 'customer';
    });
}
```

Mari kita buat middleware yang akan menggunakan ability yang telah kita buat. Jalankan:

```
php artisan make:middleware Role
```

Pada file yang dihasilkan kita isi method `handle` dengan:

app/Http/Middleware/Role.php

```
public function handle($request, Closure $next, $role)
{
    if (\Auth::user()->can($role . '-access')) {
        return $next($request);
    }
    return response('Unauthorized.', 401);
}
```

Kita menggunakan middleware parameter dengan parameter berupa role user yang diizinkan untuk mengakses route. Disini, kita menggabungkan parameter yang dikirim dengan string `-access` untuk mengecek ke ability `admin-access` atau `custome-access`. Ketika user tidak memiliki role yang dibutuhkan, kita memberikan response 401 dengan pesan "Unauthrized". Tentunya, kita dapat mengkonfigurasinya ke response yang lain misalnya menampilkan view khusus atau redirect.

Kita register middleware ini di `app/Http/Kernel.php`:

app/Http/Kernel.php

```
.....
protected $routeMiddleware = [
    ...
    'role' => \App\Http\Middleware\Role::class,
];
.....
```

Pengaturan Asset

Untuk merapikan tampilan, ubah isian title pada `resources/views/layouts/app.blade.php` menjadi:

resources/views/layouts/app.blade.php

```
<title>Laravel</title>
```

Dan isian untuk brandnya menjadi:

resources/views/layouts/app.blade.php

```
....  
<a class="navbar-brand" href="{{ url('/') }}>  
    Reselia  
</a>  
....
```

Untuk mempercepat loading asset (css dan js), kita akan menggunakan offline asset yang dicompile oleh Elixir. Tambahkan baris berikut pada `package.json`:

package.json

```
{  
    ....  
    "dependencies": {  
        ....  
        "font-awesome": "^4.5.0",  
        "jquery": "^2.2.0"  
    }  
}
```

Install dengan:

```
npm install
```

Kita ubah file `gulpfile.js` menjadi:

gulpfile.js

```
var elixir = require('laravel-elixir')  
  
elixir(function (mix) {  
    mix.sass('app.scss')  
    mix.styles([  
        '../.../public/css/app.css',  
        '../.../node_modules/font-awesome/css/font-awesome.css',  
    ], 'public/css/app.css')  
  
    mix.scripts([  
        '../.../node_modules/jquery/dist/jquery.js',  
        '../.../node_modules/bootstrap-sass/assets/javascripts/bootstrap.min.js'
```

```
])
mix.version(['css/app.css', 'js/all.js'])
mix.copy('node_modules/font-awesome/fonts', 'public/fonts')
mix.copy('node_modules/font-awesome/fonts', 'public/build/fonts')
})
```

Syntax ini akan mengcompile file css dan js yang kita butuhkan menjadi satu file. Kita juga menggunakan `mix.version` untuk melakukan versioning dari kedua file tersebut. Dan 2 baris terakhir akan menyalin file font untuk font-awesome ke folder `public/fonts` dan `public/build/fonts`.

Kita compile dengan:

`gulp`

Untuk menggunakan css dan js ini, kita ubah isian `head` pada `resources/views/layouts/app.blade.php`:

resources/views/layouts/app.blade.php

```
<head>
    ...
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.4.0/css/font-awesome.min.css" rel='stylesheet' type='text/css'>

    <link href="https://fonts.googleapis.com/css?family=Lato:100,300,400,700" rel='stylesheet' type='text/css'>

    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">

    <link href="{{ elixir('css/app.css') }}" rel="stylesheet">
    ...
</head>
```

Pada baris sebelum tag `</body>` kita ubah menjadi:

resources/views/layouts/app.blade.php

```

.....
<!-- JavaScripts -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>

<script src="{{ elixir('js/all.js') }}"></script>
</body>
.....

```

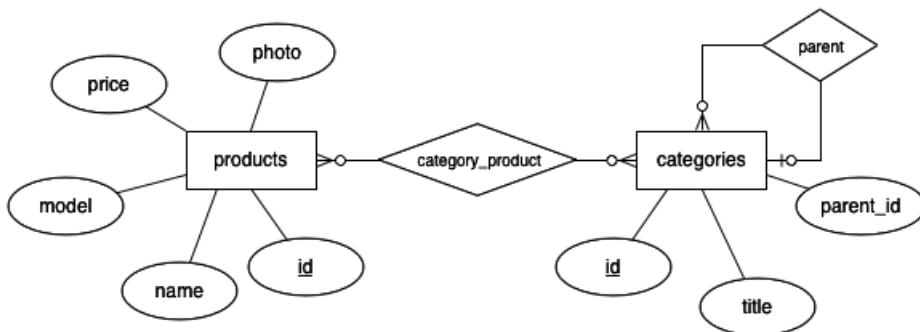
Coba refresh lagi halaman Reselia, pastikan tidak ada yang error.

Manajemen Produk

Untuk mengelompokan produk di Reselia kita akan menggunakan aturan seperti berikut:

- Setiap produk dapat memiliki lebih dari satu kategori
- Setiap kategori dapat memiliki satu parent kategori

Skema database yang akan kita buat akan seperti:



Skema table products dan categories

Konfigurasi model

Mari kita buat migration dan model yang dibutuhkan untuk produk. Jalankan perintah:

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

Pada file migration yang dihasilkan untuk model Product, pada method `up()` isi dengan syntax berikut:

database/migrations/2016_01_10_053803_create_products_table.php

```
public function up()  
{  
    Schema::create('products', function (Blueprint $table) {  
        $table->increments('id');  
        $table->string('name');  
        $table->string('photo');  
        $table->string('model');  
        $table->decimal('price', 10, 2); // max XX,XXX,XXX.XX  
        $table->timestamps();  
    });  
}
```

Pada migration untuk model Category, kita isi method `up()` dengan:

database/migrations/2016_01_10_054050_create_categories_table.php

```
public function up()  
{  
    Schema::create('categories', function (Blueprint $table) {  
        $table->increments('id');  
        $table->string('title');  
        $table->integer('parent_id');  
        $table->timestamps();  
    });  
  
    Schema::create('category_product', function (Blueprint $table) {  
        $table->increments('id');  
        $table->integer('product_id')->unsigned();  
        $table->integer('category_id')->unsigned();  
        $table->foreign('product_id')->references('id')->on('products');  
        $table->foreign('category_id')->references('id')->on('categories');  
        $table->timestamps();  
    });  
}
```

Untuk mempercepat index, pada table category_product kita juga menambah index pada field product_id dan category_id.

Karena disini kita membuat dua table, kita juga harus mengubah method down() menjadi:

database/migrations/2016_01_10_054050_create_categories_table.php

```
public function down()
{
    Schema::drop('category_product');
    Schema::drop('categories');
}
```

Mari kita konfigurasi model Product:

app/Product.php

```
<?php
...
class Product extends Model
{
    protected $fillable = ['name', 'photo', 'model', 'price'];

    public function categories()
    {
        return $this->belongsToMany('App\Category');
    }
}
```

Pada syntax diatas kita mengaktifkan mass assignment dan menambahkan relasi many-to-many ke model Category dengan nama categories.

Kita konfigurasi juga model Category:

app/Category.php

```
....  
class Category extends Model  
{  
    protected $fillable = ['title', 'parent_id'];  
  
    public function childs()  
    {  
        return $this->hasMany('App\Category', 'parent_id');  
    }  
  
    public function parent()  
    {  
        return $this->belongsTo('App\Category', 'parent_id');  
    }  
  
    public function products()  
    {  
        return $this->belongsToMany('App\Product');  
    }  
}
```

Pada model Category kita juga memiliki relasi many-to-many ke model Product. Seperti dijelaskan sebelumnya, pada model Category kita akan memiliki self reference table:

- parent: digunakan untuk mencari parent dari sebuah category, relasinya many-to-one.
- childs: digunakan untuk mencari sub kategori, relasinya one-to-many.

Sample data

Dalam mengembangkan sebuah fitur, sebelum membuat UI, saya lebih suka membuat sample data dengan seeder terlebih dahulu. Ini akan membuat kita lebih mudah untuk berinteraksi dengan aplikasi ketika UI sudah jadi. Untuk itu, mari kita buat sample data untuk Product dan Category.

Jalankan perintah berikut untuk membuat template seeder:

```
php artisan make:seeder ProductsSeeder
```

Pada file seeder yang dihasilkan, isi method `run()` dengan syntax berikut:

database/seeds/ProductsSeeder.php

```
<?php  
....  
use App\Category;  
use App\Product;  
  
class ProductsSeeder extends Seeder  
{  
    public function run()  
    {  
        // sample category  
        $sepatu = Category::create(['title' => 'Sepatu']);  
        $sepatu->childs()->saveMany([  
            new Category(['title' => 'Lifestyle']),  
            new Category(['title' => 'Berlari']),  
            new Category(['title' => 'Basket']),  
            new Category(['title' => 'Sepakbola'])  
        ]);  
  
        $pakaian = Category::create(['title' => 'Pakaian']);  
        $pakaian->childs()->saveMany([  
            new Category(['title' => 'Jaket']),  
            new Category(['title' => 'Hoodie']),  
            new Category(['title' => 'Rompi'])  
        ]);  
  
        // sample product  
        $running = Category::where('title', 'Berlari')->first();  
        $lifestyle = Category::where('title', 'Lifestyle')->first();  
        $sepatu1 = Product::create([  
            'name' => 'Nike Air Force',  
            'model' => 'Sepatu Pria',  
            'photo'=>'stub-shoe.jpg',  
            'price' => 340000]);  
        $sepatu2 = Product::create([  
            'name' => 'Nike Air Max',  
            'model' => 'Sepatu Wanita',  
            'photo'=>'stub-shoe.jpg',  
            'price' => 420000]);  
        $sepatu3 = Product::create([  
            'name' => 'Nike Air Zoom',  
            'model' => 'Sepatu Wanita',
```

```

    'photo'=>'stub-shoe.jpg',
    'price' => 360000]);
$running->products()->saveMany([$sepatu1, $sepatu2, $sepatu3]);
$lifestyle->products()->saveMany([$sepatu1, $sepatu2]);

$jacket = Category::where('title', 'Jaket')->first();
$vest = Category::where('title', 'Rompi')->first();
$jacket1 = Product::create([
    'name' => 'Nike Aeroloft Bomber',
    'model' => 'Jaket Wanita',
    'photo'=>'stub-jacket.jpg',
    'price' => 720000]);
$jacket2 = Product::create([
    'name' => 'Nike Guild 550',
    'model' => 'Jaket Pria',
    'photo'=>'stub-jacket.jpg',
    'price' => 380000]);
$jacket3 = Product::create([
    'name' => 'Nike SB Steele',
    'model' => 'Jaket Pria',
    'photo'=>'stub-jacket.jpg',
    'price' => 1200000]);
$jacket->products()->saveMany([$jacket1, $jacket3]);
$vest->products()->saveMany([$jacket2, $jacket3]);
}
}

```

Pada sample data ini, kita membuat beberapa kategori berikut sub kategorinya. Selanjutnya kita juga membuat beberapa produk dengan kategori yang sudah kita tentukan. Untuk memudahkan pengembangan, kita akan mengupload foto produk ke folder `public/img`. Sementara di table `products` kita hanya menyimpan nama filenya. Agar aplikasi tidak error ketika dijalankan buatlah folder `public/img` dan salinlah file `stub-shoe.jpg` dan `stub-shoe.jpg` dari sample code (atau buat sendiri).

Tambahkan baris berikut pada method `run` di `database/seeds/DatabaseSeeder.php`:

database/seeds/DatabaseSeeder.php

```
$this->call(ProductsSeeder::class);
```

Selanjutnya, kita refresh dan seed dengan:

```
php artisan migrate:refresh --seed
```

Setelah dijalankan, pastikan terdapat isian berikut di database.

id	title	parent_id	created_at	updated_at
1	Sepatu	0	2016-02-24 04:56:43	2016-02-24 04:56:43
2	Lifestyle	1	2016-02-24 04:56:43	2016-02-24 04:56:43
3	Berlari	1	2016-02-24 04:56:43	2016-02-24 04:56:43
4	Basket	1	2016-02-24 04:56:43	2016-02-24 04:56:43
5	Sepakbola	1	2016-02-24 04:56:43	2016-02-24 04:56:43
6	Pakaian	0	2016-02-24 04:56:43	2016-02-24 04:56:43
7	Jaket	6	2016-02-24 04:56:43	2016-02-24 04:56:43
8	Hoodie	6	2016-02-24 04:56:43	2016-02-24 04:56:43
9	Rompi	6	2016-02-24 04:56:43	2016-02-24 04:56:43

Sample Kategori

id	name	photo	model	price	category_id
1	Nike Air Force	stub-shoe.jpg	Sepatu Pria	340000.00	2
2	Nike Air Max	stub-shoe.jpg	Sepatu Wanita	420000.00	2
3	Nike Air Zoom	stub-shoe.jpg	Sepatu Wanita	360000.00	2
4	Nike Aeroloft Bomber	stub-jacket.jpg	Jaket Wanita	720000.00	2
5	Nike Guild 550	stub-jacket.jpg	Jaket Pria	380000.00	2
6	Nike SB Steele	stub-jacket.jpg	Jaket Pria	1200000.00	2

Sample Product

CRUD Kategori

Proses pembuatan CRUD untuk Kategori akan kita jadikan acuan untuk proses CRUD yang lainnya. Mari kita mulai dengan membuat controller untuk Categories:

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

Kita tambahkan di route pada group web:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::resource('categories', 'CategoriesController');
});
```

Untuk middleware yang akan membatasi akses hanya untuk user dengan role admin, akan kita tambahkan pada controller:

app/Http/Controllers/CategoriesController.php

```
...
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('role:admin');
}
...
```

Disini kita menggunakan middleware `auth` dan `role` (dengan menggunakan parameter `admin`) untuk membatasi akses hanya user yang sudah login dan memiliki role `admin`.

List Kategori

Pertama kita akan membuat listing kategori, isi syntax berikut pada method `index`:

app/Http/Controllers/CategoriesController.php

```
<?php
...
use App\Category;

class CategoriesController extends Controller
{
    ...
    public function index()
    {
        $categories = Category::get();
        return view('categories.index', compact('categories'));
    }
}
```

Buat view yang dibutuhkan di `resources/views/categories/index.blade.php` dengan isi:

resources/views/categories/index.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h3>Category</h3>
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <td>Title</td>
                            <td>Parent</td>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach($categories as $category)
                            <tr>
                                <td>{{ $category->title }}</td>
                                <td>{{ $category->parent ? $category->parent->title : '' }}</td>
                            </tr>
                        @endforeach
                    </tbody>
                </table>
            </div>
        </div>
    </div>
@endsection
```

Pada view ini, kita menampilkan nama dan parent dari tiap kategori. Cobalah login sebagai admin dan kunjungi `/categories` ini yang akan kita dapatkan:

Title	Parent
Sepatu	
Lifestyle	Sepatu
Berlari	Sepatu
Basket	Sepatu
Sepakbola	Sepatu
Pakaian	
Jaket	Pakaian
Hoodie	Pakaian
Rompi	Pakaian

Halaman list category

Untuk memudahkan kita mengakses halaman ini, mari kita buat menu di navbar. Ubah baris berikut di `resources/views/layouts/app.blade.php`:

resources/views/layouts/app.blade.php

```
<!-- Left Side Of Navbar -->
<ul class="nav navbar-nav">
    <li><a href="{{ url('/home') }}">Home</a></li>
</ul>
```

menjadi:

resources/views/layouts/app.blade.php

```
<!-- Left Side Of Navbar -->
<ul class="nav navbar-nav">
    @if(Auth::check())
        <li><a href="{{ url('/home') }}>Home</a></li>
        @can('admin-access')
            <li class="dropdown">
                <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-expanded="false">
                    Manage <span class="caret"></span>
                </a>
                <ul class="dropdown-menu" role="menu">
                    <li><a href="{{ route('categories.index') }}><i class="fa fa-btn fa-tags"></i>Categories</a></li>
                </ul>
            </li>
        @endcan
    @endif
</ul>
```

Syntax ini akan menghilangkan link `home` dan menambah link untuk `category` ketika user login sebagai admin. Sehingga, tampilannya akan seperti berikut:

Title	Parent
Sepatu	
Lifestyle	Sepatu
Berlari	Sepatu
Basket	Sepatu
Sepakbola	Sepatu
Pakaian	
Jaket	Pakaian
Hoodie	Pakaian
Rompi	Pakaian

Menu Kategori

Pagination

Fitur pagination sudah umum dimiliki oleh website yang memiliki banyak data. Secara sederhana fitur ini membatasi jumlah data yang ditampilkan dalam satu halaman dan menampilkan link untuk melihat data sebelumnya atau selanjutnya. Di Laravel, kita dapat membuat fitur pagination dengan mudah. Mari kita demokan dengan membuat pagination pada list kategori.

Pada method index, ubah menjadi seperti ini:

app/Http/Controllers/CategoriesController.php

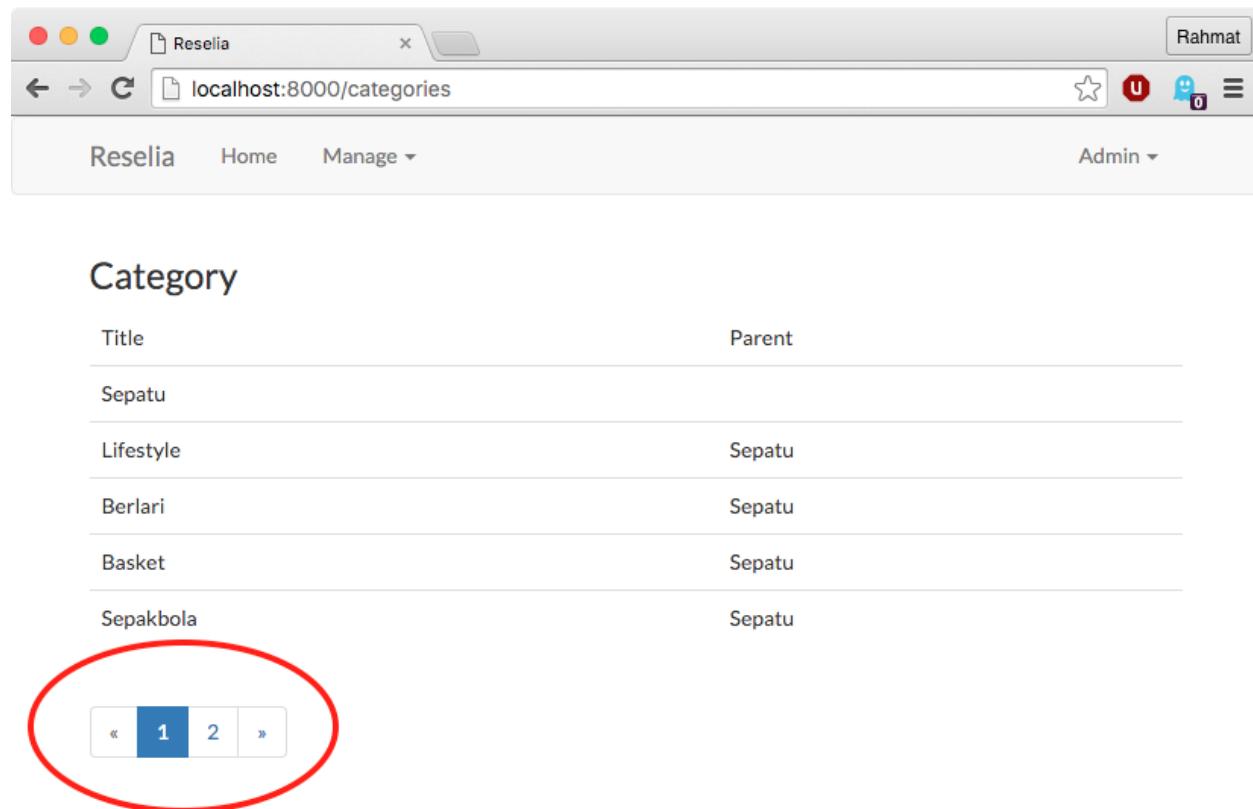
```
public function index()
{
    $categories = Category::paginate(5);
    return view('categories.index', compact('categories'));
}
```

Method paginate(5) akan membuat pagination dengan jumlah item 5 per halaman. Pada view, kita tambahkan syntax berikut setelah </table>:

resources/views/categories/index.blade.php

```
.....
</table>
{{ $categories->links() }}
```

Sehingga akan muncul tampilan berikut:



The screenshot shows a web browser window titled 'Reselia'. The address bar displays 'localhost:8000/categories'. The page content is titled 'Category' and lists five categories: Sepatu, Lifestyle, Berlari, Basket, and Sepakbola. Each category has 'Sepatu' listed under the 'Parent' column. At the bottom of the list, there is a pagination control with links for '«', '1', '2', and '»'. A red oval highlights the '1', '2', and '»' links.

Title	Parent
Sepatu	Sepatu
Lifestyle	Sepatu
Berlari	Sepatu
Basket	Sepatu
Sepakbola	Sepatu

Pagination berhasil

Secara default, pagination yang dihasilkan oleh Laravel akan compatible dengan tampilan bootstrap.

Pencarian pada pagination

Akan lebih menarik jika pagination yang kita buat memiliki fitur search. Mari kita buat.

Untuk fitur search ini, kita akan menggunakan url parameter dengan key `q`. Pada controller kita akan mengambil key ini dan menggunakannya untuk mencari kategori yang memiliki `title` dengan keyword tersebut.

Untuk memudahkan dalam membuat form, kita akan memakai package [laravelcollective/html](#)²⁰⁰. Kita install dengan perintah:

```
composer require laravelcollective/html
```

Tambahkan baris berikut pada isian `providers` di `config/app.php`:

config/app.php

```
'providers' => [
    ...
    Collective\Html\HtmlServiceProvider::class,
]
```

Dan pada isian `aliases`:

config/app.php

```
'aliases' => [
    ...
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
],
```

Sip. Mari kita buat form di `resources/views/categories/index.blade.php` sebelum tag `<table>`:

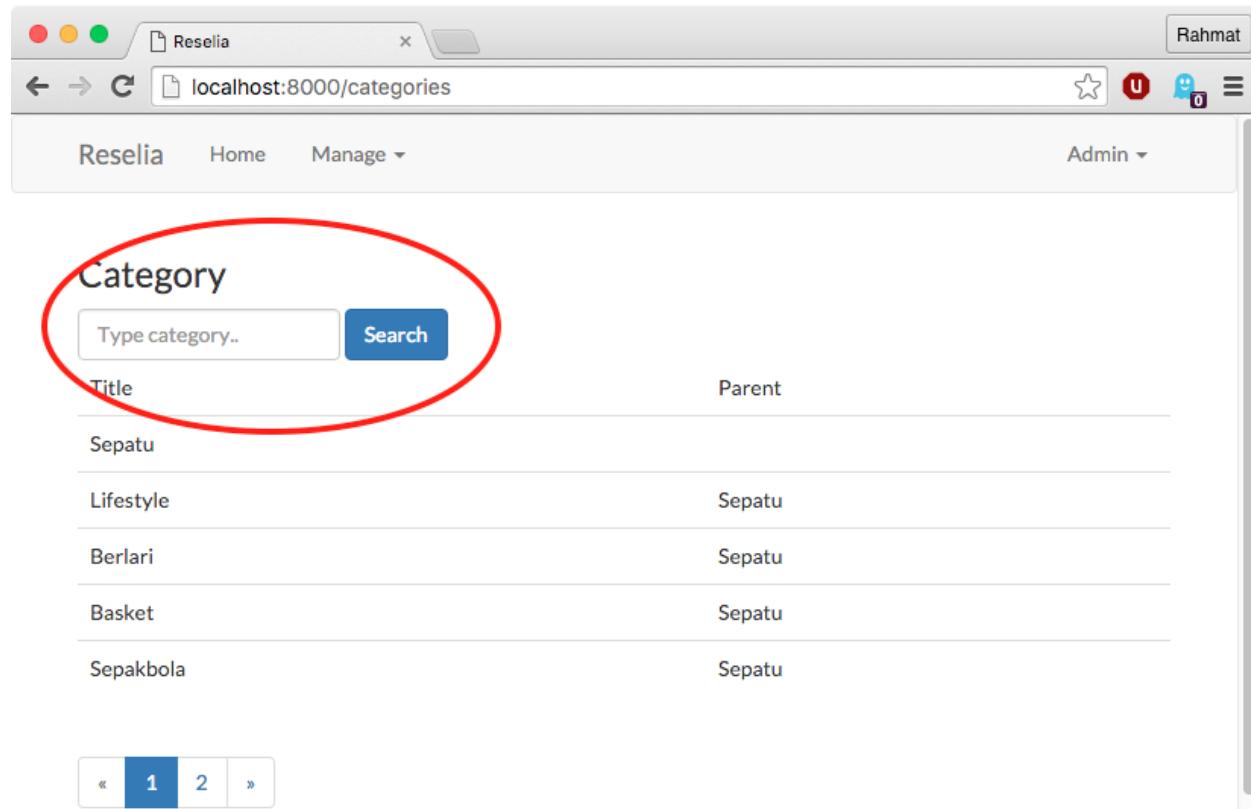
²⁰⁰ [composerrequirelaravelcollective/html](#)

resources/views/categories/index.blade.php

```
.....
{!! Form::open(['url' => 'categories', 'method'=>'get', 'class'=>'form-inline']) !!}
  <div class="form-group {!! $errors->has('q') ? 'has-error' : '' !!}">
    {!! Form::text('q', isset($q) ? $q : null, ['class'=>'form-control', 'placeholder' => 'Type category..']) !!}
    {!! $errors->first('q', '<p class="help-block">:message</p>') !!}
  </div>

  {!! Form::submit('Search', ['class'=>'btn btn-primary']) !!}
  {!! Form::close() !!}
<table class="table table-hover">
  ....
```

Sehingga tampilannya akan menjadi:



The screenshot shows a web application interface. At the top, there's a header with a logo, a navigation bar with 'Reselia', 'Home', 'Manage', and 'Admin' dropdown, and a user profile icon. Below the header, the main content area has a title 'Category'. A search form is displayed, consisting of a text input field with placeholder 'Type category..' and a blue 'Search' button. This search form is circled in red. Below the search form is a table with two columns: 'Title' and 'Parent'. The table contains the following data:

Title	Parent
Sepatu	Sepatu
Lifestyle	Sepatu
Berlari	Sepatu
Basket	Sepatu
Sepakbola	Sepatu

At the bottom of the table, there are navigation links: '<', '1', '2', and '>'. The number '1' is highlighted in blue.

Form pencarian kategori

Sip. Disini, kita mengarahkan formnya ke method `index` di `CategoriesController`. Mari kita modifikasi agar dapat menerima form ini dan melakukan pencarian:

app/Http/Controllers/CategoriesController.php

```
public function index(Request $request)
{
    $categories = Category::where('title', 'LIKE', '%' . $request->get('q') . '%')->\paginate(5);
    return view('categories.index', compact('categories'));
}
```

Mari kita coba mencari kategori yang memiliki huruf e:

The screenshot shows a web browser window titled 'Reselia'. The address bar contains 'localhost:8000/categories?q=e'. The page itself is titled 'Category' and features a search bar with the letter 'e' typed in, and a 'Search' button. Below the search bar is a table with two columns: 'Title' and 'Parent'. The table contains five rows of data: 'Sepatu', 'Lifestyle', 'Berlari', 'Basket', and 'Sepakbola', all of which are listed under the 'Sepatu' parent category. At the bottom of the page is a pagination control with links for '«', '1', '2', and '»'.

Title	Parent
Sepatu	Sepatu
Lifestyle	Sepatu
Berlari	Sepatu
Basket	Sepatu
Sepakbola	Sepatu

Mencari kategori

Sip, berhasil. Tapi, perhatikan apa yang terjadi ketika kita klik link pagination untuk page 2:

Title	Parent
Pakaian	
Jaket	Pakaian
Hoodie	Pakaian
Rompi	Pakaian

Pencarian error

Ini terjadi, karena pada link pagination kita tidak menambah url param q yang berisi keyword yang sedang kita cari:

Memiliki url param q

Tidak ada url param q di page 2

Untuk memperbaikinya, kita harus mengirim parameter q yang diterima oleh con-

troller ke view dan menambahkan ke pagination links.

Ubah controller menjadi seperti ini:

app/Http/Controllers/CategoriesController.php

```
public function index(Request $request)
{
    $q = $request->get('q');
    $categories = Category::where('title', 'LIKE', '%' . $q . '%')->orderBy('title')\ 
->paginate(10);
    return view('categories.index', compact('categories', 'q'));
}
```

Pada view, ubah pagination links menjadi seperti ini:

resources/views/categories/index.blade.php

```
{{ $categories->appends(compact('q'))->links() }}
```

Method `appends()` menerima parameter berupa array asosiatif yang akan kita tambahkan sebagai parameter tambahan di pagination links. Disini, kita menggunakan method `compact()` untuk membuat array tersebut dan mengeset variable `q` dengan isian `$q` dari controller.

Kini, jika kita mencari dengan keyword `e`, pada halaman kedua parameter `q` akan tetap ada:



Url param q tetap muncul di page 2

Menambah Kategori

Mari kita buat fitur untuk menambah kategori baru, kita tambahkan dulu tombol untuk membuat kategori dari halaman index dengan mengubah tag `<h3>` menjadi:

resources/views/categories/index.blade.php

```
<h3>Category <small><a href="{{ route('categories.create') }}" class="btn btn-warning btn-sm">New Category</a></small></h3>
```

Tombol ini akan mengarah ke /categories/create. Tampilannya akan seperti ini:

The screenshot shows a web application interface titled 'Reselia'. In the top navigation bar, there are links for 'Reselia', 'Home', and 'Manage'. On the right, there is a user profile icon labeled 'Admin'. The main content area is titled 'Category' and features a prominent orange button labeled 'New Category'. Below this, there is a search bar with the placeholder 'Type category...' and a blue 'Search' button. The entire interface has a clean, modern design with a light color palette.

Tombol tambah kategori

Kita tambah logic di method `create`:

app/Http/Controllers/CategoriesController.php

```
public function create()
{
    return view('categories.create');
}
```

Kita buat viewnya:

resources/views/categories/create.blade.php

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

@section('content')


### New Category


{!! Form::open(['route' => 'categories.store']) !!}
@include('categories._form')


```

```

{!! Form::close() !!}
</div>
</div>
</div>
@endsection

```

Disini kita menggunakan partial view `categories._form` untuk menyimpan field dari form. Ini kita lakukan karena kita akan menggunakan form yang sama untuk halaman `create` dan `edit`. Mari kita buat partial view ini:

resources/views/categories/_form.blade.php

```

<div class="form-group {!! $errors->has('title') ? 'has-error' : '' !!}">
    {!! Form::label('title', 'Title') !!}
    {!! Form::text('title', null, ['class'=>'form-control']) !!}
    {!! $errors->first('title', '<p class="help-block">:message</p>') !!}
</div>

<div class="form-group {!! $errors->has('parent_id') ? 'has-error' : '' !!}">
    {!! Form::label('parent_id', 'Parent') !!}
    {!! Form::select('parent_id', [''=>'']+App\Category::lists('title','id')->all(), null, ['class'=>'form-control']) !!}
    {!! $errors->first('parent_id', '<p class="help-block">:message</p>') !!}
</div>

{!! Form::submit(isset($model) ? 'Update' : 'Save', ['class'=>'btn btn-primary']) !!}

```

Format form seperti ini akan kita gunakan di berbagai form, berikut beberapa penjelasannya:

- Tiap field akan diapit oleh `div` dengan class `form-group`. Pada div tersebut kita menggunakan menambahkan class `has-error` jika terdapat error pada field tersebut.
- Kita juga menampilkan pesan error validasi yang pertama untuk tiap field dengan format `<p class="help-block">pesan-error</p>` dibawah field tersebut (jika ada error).
- Tombol submit kita beri nama `Save` untuk form `create` dan `update` untuk form `edit`.

Pada form, ini kita menggunakan `[''=>'']+App\Category::lists('title','id')->all()` untuk membuat array asosiatif sebagai pilihan `parent category`.

Tampilan form akan seperti ini:

New Category

Title

Parent

Save

Form untuk membuat kategori

Untuk menerima form ini, kita akan mengubah method `store` menjadi:

app/Http/Controllers/CategoriesController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|string|max:255|unique:categories',
        'parent_id' => 'exists:categories,id'
    ]);

    Category::create($request->all());
    return redirect()->route('categories.index');
}
```

Disini kita melakukan validasi untuk form, membuat kategori dan mengarahkan user ke halaman `/categories`. Cobalah membuat kategori baru, pastikan validasi berjalan.



New Category

Title

The title field is required.

Parent

Save

Validasi kategori

Kini, cobalah membuat kategori baru, pastikan berhasil.



Membuat kategori baru

Title	Parent
Pakaian	
Jaket	Pakaian
Hoodie	Pakaian
Rompi	Pakaian
Kaos	Pakaian

« 1 2 »

Berhasil membuat kategori baru

Konfigurasi Feedback

Salah satu *best practices* dalam membuat UI adalah dengan memberikan feedback ke user atas aksi yang telah dilakukan. Pada kasus ini, kita akan menampilkan pesan bahwa user berhasil membuat kategori. Laravel memiliki fitur [session flash data](#)²⁰¹ yang dapat kita gunakan untuk membuat fitur ini. Caranya kerja flash data adalah dengan membuat sebuah session variable yang hanya dapat diakses pada 1 request selanjutnya. Untuk membuat flash data kita dapat menggunakan syntax:

```
Session::flash('key', 'value');
// atau
session()->flash('key', 'value');
// atau
$request->session()->flash('key', 'value');
```

Pada view, kita dapat mengecek apakah session tersebut ada dengan menggunakan helper:

²⁰¹ <https://laravel.com/docs/5.2/session#flash-data>

```
Session::has('key');
// atau
session()->has('key');
```

Untuk menampilkan nilai dari session kita dapat menggunakan:

```
Session::get('key');
// atau
session()->get('key');
```

Pada kasus ini, kita tidak akan membuat fitur ini secara manual. Untuk mempercepat pembuatan fitur ini, kita akan menggunakan package [laracasts/flash](#)²⁰². Install dengan perintah:

```
composer require laracasts/flash
```

Setelah terdownload, tambahkan isian berikut pada isian providers di config/app.php:

config/app.php

```
'providers' => [
    ...
    Laracasts\Flash\FlashServiceProvider::class,
]
```

Dan pada isian aliases:

config/app.php

```
'aliases' => [
    ...
    'Flash' => 'Laracasts\Flash\Flash',
]
```

Kita tambahkan syntax berikut sebelum isian @yield('content') pada resources/views/layouts/app.blade.php:

²⁰²<https://github.com/laracasts/flash>

resources/views/layouts/app.blade.php

```
.....
@if (session()->has('flash_notification.message'))
    <div class="container">
        <div class="alert alert-{{ session()->get('flash_notification.level') }}\"
    "gt;
        <button type="button" class="close" data-dismiss="alert" aria-hidden\
="true">&times;</button>
        {{ session()->get('flash_notification.message') }}
    </div>
</div>
@endif

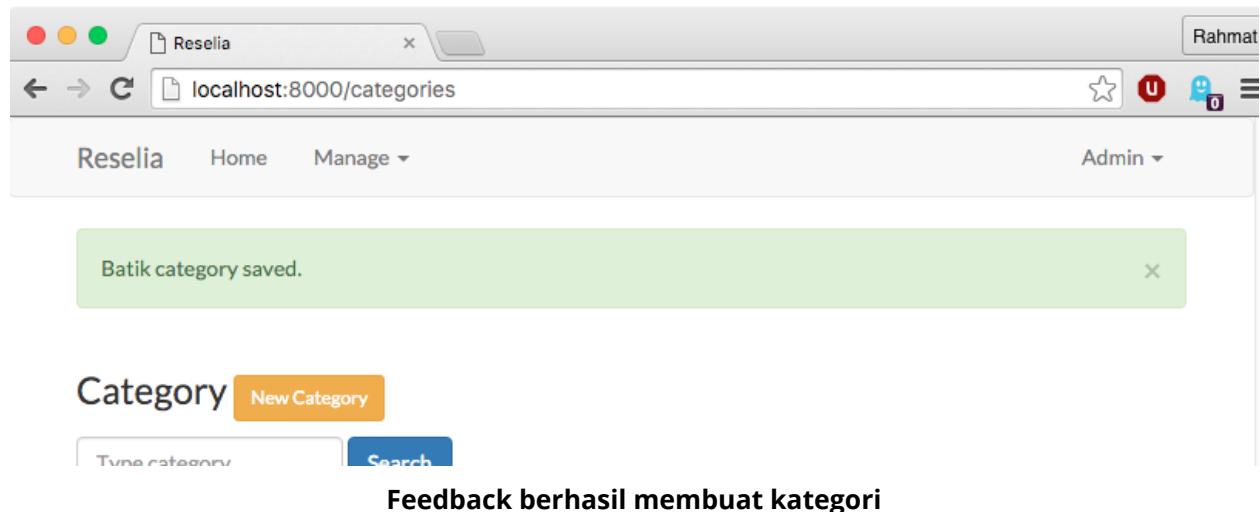
@yield('content')
....
```

Pada method `store`, kita tambah syntax berikut sebelum melakukan redirect:

app/Http/Controllers/CategoriesController.php

```
public function store(Request $request)
{
    .....
    \Flash::success($request->get('title') . ' category saved.');
    return redirect()->route('categories.index');
}
```

Kini, ketika kita berhasil menyimpan sebuah kategori, akan muncul tampilan:



The screenshot shows a web browser window titled 'Reselia'. The address bar displays 'localhost:8000/categories'. The page content includes a navigation bar with 'Reselia', 'Home', 'Manage', and 'Admin' dropdown. Below the navigation is a green success message box containing 'Batik category saved.' with a close button. At the bottom of the page, there is a search form with a placeholder 'Type category' and a blue 'Search' button. A bold black text 'Feedback berhasil membuat kategori' is centered at the bottom of the page.

Selain pesan berupa success, package ini juga mendukung jenis pesan lain:

- Flash::error()
- Flash::info()
- Flash::warning()
- Flash::overlay() (untuk modal)

Silahkan kunjungi dokumentasi dari package ini untuk informasi lebih lanjut.

Mengubah Kategori

Pembuatan fitur ini cukup sederhana, ikuti langkah berikut.

Kita tambahkan link menuju halaman edit dari halaman list kategori:

resources/views/categories/index.blade.php

```
....  
<table class="table table-hover">  
  <thead>  
    <tr>  
      ....  
      <td></td>  
    </tr>  
  </thead>  
  <tbody>  
    @foreach($categories as $category)  
    <tr>  
      ....  
      <td>  
        <a href="{{ route('categories.edit', $category->id)}}>Edit</a>  
      </td>  
    </tr>  
    @endforeach  
  </tbody>  
</table>  
....
```

Pada method `edit`, kita akan menampilkan view untuk edit:

app/Http/Controllers/CategoriesController.php

```
public function edit($id)
{
    $category = Category::findOrFail($id);
    return view('categories.edit', compact('category'));
}
```

Kita buat viewnya:

resources/views/categories/edit.blade.php

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

@section('content')


### Edit {{ $category->title }}


{!! Form::model($category, ['route' => ['categories.update', $category], \
'method' => 'patch']) !!}
@include('categories._form', ['model' => $category])
{!! Form::close() !!}


@endsection
```

Seperti yang sudah dijelaskan dipembahasan sebelumnya, kita akan menggunakan form yang sama untuk halaman `create` dan `edit`. Perbedaan utama halaman `edit` adalah kita menggunakan fitur `form model binding`²⁰³.

```
{!! Form::model($category, ['route' => ['categories.update', $category], 'method\
' => 'patch']) !!}
....
```

Dengan fitur ini, kita akan mengisi tiap field di form dengan data yang sudah ada di database.

Untuk menerima data yang dikirim oleh form ini, method `update` akan kita ubah menjadi:

²⁰³<https://laravelcollective.com/docs/5.2/html#form-model-binding>

app/Http/Controllers/CategoriesController.php

```
public function update(Request $request, $id)
{
    $category = Category::findOrFail($id);
    $this->validate($request, [
        'title' => 'required|string|max:255|unique:categories,title,' . $category->id,
        'parent_id' => 'exists:categories,id'
    ]);

    $category->update($request->all());
    \Flash::success($request->get('title') . ' category updated.');
    return redirect()->route('categories.index');
}
```

Ada lima tahapan yang kita lakukan pada method ini:

1. Mencari model yang sesuai dengan `$id` yang dikirim
2. Melakukan validasi
3. Melakukan proses update
4. Mengeset flash untuk pesan berhasil di update
5. Mengarahkan user ke halaman list category

Cobalah melakukan proses update, pastikan fitur ini berjalan dengan baik.

Menghapus Kategori

Proses update dapat kita lakukan dengan mengirimkan form kosong ke `/categories/{id}` dengan method `DELETE`. Untuk memudahkan proses delete, kita akan menambahkan tombol delete ini dari halaman listing kategori.

Ubahlah baris berikut pada halaman index:

resources/views/categories/index.blade.php

```
<td>
    <a href="{{ route('categories.edit', $category->id) }}">Edit</a>
</td>
```

Menjadi:

resources/views/categories/index.blade.php

```
<td>
  {!! Form::model($category, ['route' => ['categories.destroy', $category], 'method' => 'delete', 'class' => 'form-inline']) !!}
    <a href="{{ route('categories.edit', $category->id)}}>Edit</a> |
    {!! Form::submit('Delete', ['class'=>'btn btn-xs btn-danger']) !!}
  {!! Form::close() !!}
</td>
```

Sehingga tampilannya akan menjadi seperti ini:

Title	Parent	
Sepatu		Edit delete
Lifestyle	Sepatu	Edit delete
Berlari	Sepatu	Edit delete
Basket	Sepatu	Edit delete
Sepakbola	Sepatu	Edit delete

Type category..

« 1 2 3 »

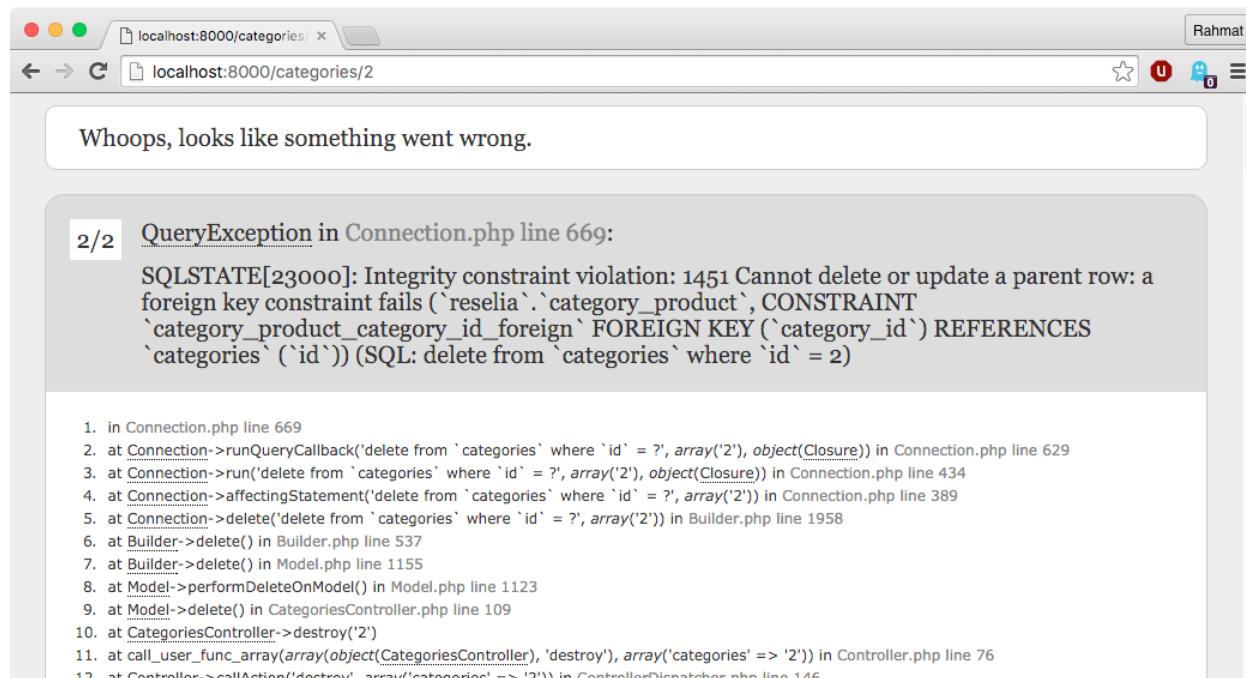
Tombol delete kategori

Pada controller, kita ubah method `destroy` menjadi:

app/Http/Controllers/CategoriesController.php

```
public function destroy($id)
{
    Category::find($id)->delete();
    \Flash::success('Category deleted.');
    return redirect()->route('categories.index');
}
```

Cobalah hapus sebuah kategori..

**Terjadi error ketika menghapus kategori**

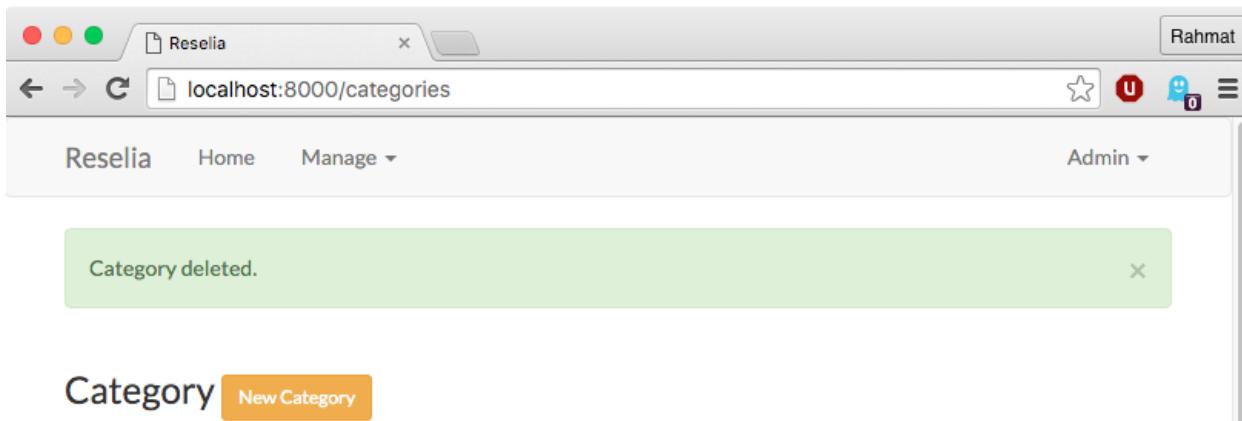
Uppss.. ada error. Ini terjadi karena kita menambahkan foreign key pada field category_id di table category_product ke field id di table categories. Ini tidak akan terjadi jika kategori yang kita hapus tidak memiliki produk.

Solusinya, kita akan menghapus record untuk kategori yang sedang kita hapus pada table category_product dengan menggunakan event deleting pada model Category. Tambahkan method berikut pada model Category:

app/Category.php

```
....  
public static function boot()  
{  
    parent::boot();  
  
    static::deleting(function($model) {  
        // remove relations to products  
        $model->products()->detach();  
    });  
}  
....
```

Cobalah hapus kembali sebuah category, pastikan berhasil.



The screenshot shows a web browser window titled 'Reselia'. The address bar contains 'localhost:8000/categories'. The page header includes 'Reselia', 'Home', 'Manage', and 'Admin'. A green success message box at the top states 'Category deleted.'. Below this, the main content area has a heading 'Category' and a sub-heading 'Berhasil menghapus kategori'.

Ada satu relasi lagi yang perlu kita hapus pada event ini, yaitu relasi ke child. Jadi, setiap kali kita menghapus kategori, semua kategori yang memiliki `parent_id` dengan id kategori yang sedang kita hapus akan kita hapus isian `parent_id`-nya. Caranya, tambahkan baris berikut pada event `deleting`:

app/Category.php

```
....  
static::deleting(function($model) {  
    ....  
    // remove parent from this category's child  
    foreach ($model->childs as $child) {  
        $child->parent_id = '';  
        $child->save();  
    }  
});  
....
```

Sip.

Konfirmasi Delete

Proses penghapusan data merupakan salah satu proses yang cukup “berbahaya” dalam sebuah aplikasi. Ini karena, data yang sudah dihapus tidak bisa dikembalikan lagi (kecuali kita mengaktifkan fitur soft delete di Laravel). Untuk menghindari salah klik, mari kita tambahkan konfirmasi dengan javascript sebelum melakukan penghapusan.

Kita akan menggunakan library Sweet Alert²⁰⁴ agar tampilan konfirmasi akan lebih menarik.. :)

Kita install dengan perintah:

```
npm install --save sweetalert
```

Setelah selesai pastikan ada folder `node_modules/sweetalert` pada project kita dan pada file `package.json` ada baris berikut:

²⁰⁴<http://t4t5.github.io/sweetalert>

package.json

```
{  
  ...  
  "dependencies": {  
    ...  
    "sweetalert": "^1.1.3"  
  }  
}
```

Jika sudah, kita konfigurasi Elixir untuk menambahkan library ini pada saat proses compile:

gulpfile.js

```
elixir(function (mix) {  
  ...  
  mix.styles([  
    ...  
    '.../.../.../node_modules/sweetalert/dist/sweetalert.css',  
    ], 'public/css/app.css')  
  
  mix.scripts([  
    ...  
    '.../.../.../node_modules/sweetalert/dist/sweetalert.min.js',  
    'app.js'  
  ])  
  ...  
})
```

Terlihat disini, selain file js untuk sweetalert, kita juga menambah file baru app.js. Buatlah file ini di resources/asset/js/app.js dengan isi:

resources/asset/js/app.js

```
$(document).ready(function () {
    $(document.body).on('click', '.js-submit-confirm', function (event) {
        event.preventDefault()
        var $form = $(this).closest('form')
        var $el = $(this)
        var text = $el.data('confirm-message') ? $el.data('confirm-message') : 'Kamu\
tidak akan bisa membatalkan proses ini!'

        swal({
            title: 'Kamu yakin?',
            text: text,
            type: 'warning',
            showCancelButton: true,
            confirmButtonColor: '#DD6B55',
            confirmButtonText: 'Yap, lanjutkan!',
            cancelButtonText: 'Batal',
            closeOnConfirm: true
        },
        function () {
            $form.submit()
        })
    })
})
```

Saya tidak akan menjelaskan dengan sangat detail untuk isi file `app.js` ini karena ini bukan buku javascript. Saran saya coba pelajari tentang javascript dan jQuery. Secara sederhana, yang dilakukan script ini sebagai berikut:

- Untuk menggunakan konfirmasi dengan sweet alert, kita menambahkan class '`js-submit-confirm`' pada tombol submit.
- Kita melakukan binding terhadap event `click` pada tombol tersebut dan menampilkan konfirmasi sebelum menjalankan action default pada form tersebut. Pada contoh kasus ini, kita membatalkan proses delete sebelum user mengklik `ok` pada konfirmasi.
- Isian untuk konfirmasi ini secara default adalah "Kamu tidak akan bisa membatalkan proses ini!". Jika ingin merubah pesan pesan ini, kita harus mengisi attribute `data-confirm-message` pada tombol submit.

Ubah tombol `submit` untuk menghapus categori sehingga memiliki class `js-submit-confirm`:

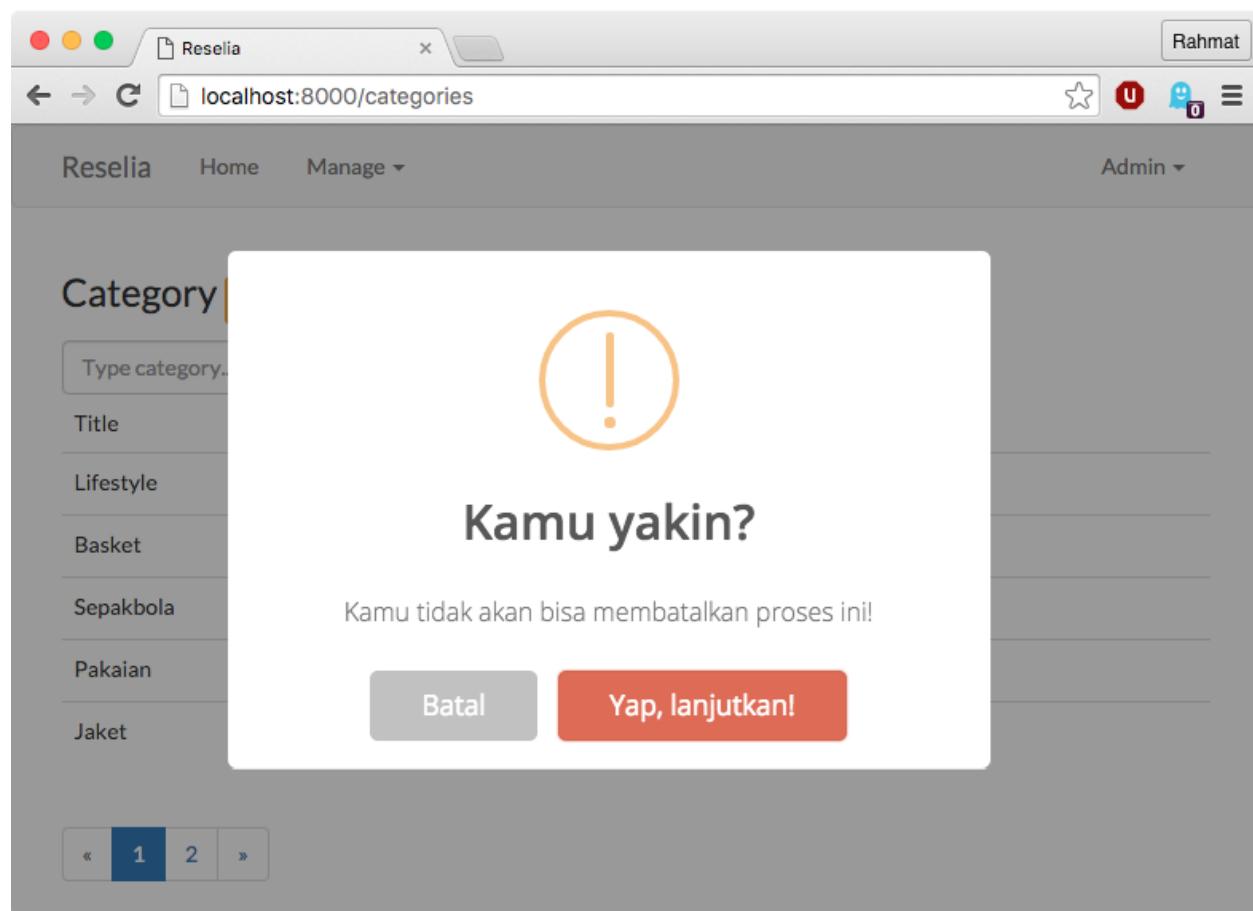
resources/views/categories/index.blade.php

```
....  
{!! Form::submit('delete', ['class'=>'btn btn-xs btn-danger js-submit-confirm'])}\  
!!}  
....
```

Kita compile dulu file CSS dan JS:

gulp

Refresh halaman, dan coba klik pada tombol `delete`, akan muncul konfirmasi berikut:



Konfirmasi sebelum menghapus kategori

Pastikan penghapusan kategori dibatalkan jika kita klik tombol "Batal" dan dilakukan jika kita klik pada tombol "Yap, Lanjutkan!".

CRUD Produk

Dalam mengembangkan fitur CRUD Produk kita akan banyak meniru syntax dari CRUD untuk Kategori. Bahkan, dalam kenyataannya saya ketika menulis buku ini, syntax dari CRUD Kategori saya salin dan beberapa bagianya disesuaikan dengan kebutuhan untuk CRUD Produk. Namun, untuk proses belajar, saya tidak menyarankan melakukan hal itu. Akan lebih baik jika setiap fitur ini dibangun bertahap.

Listing Produk

Sebelum kita membuat fitur ini, mari kita buat controller untuk Produk:

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

Dan kita buat route nya:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::resource('products', 'ProductsController');
});
```

Di ProductsController kita tambahkan buat method `__construct` untuk menyimpan middleware:

app/Http/Controllers/ProductsController.php

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('role:admin');
}
```

Pada method index di file ProductsController akan kita isi seperti berikut:

app/Http/Controllers/ProductsController.php

```
<?php
...
use App\Product;

class ProductsController extends Controller
{
    ...
    public function index(Request $request)
    {
        $q = $request->get('q');
        $products = Product::where('name', 'LIKE', '%' . $q . '%')
            ->orWhere('model', 'LIKE', '%' . $q . '%')
            ->orderBy('name')->paginate(10);
        return view('products.index', compact('products', 'q'));
    }
    ...
}
```

Pada method ini, kita akan menerima parameter `q` untuk melakukan pencarian berdasarkan field `name` atau `title`. Disini kita juga melakukan sorting berdasarkan field `name`. Terakhir, kita buat agar pagination 10 produk per halaman.

Kita buat view index dengan isi:

resources/views/products/index.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h3>Product <small><a href="{{ route('products.create') }}" class="btn btn-warning btn-sm">New Product</a></small></h3>
                {!! Form::open(['url' => 'products', 'method' => 'get', 'class' => 'form-inline']) !!}
                    <div class="form-group>{!! $errors->has('q') ? 'has-error' : '' !!}>
                        {!! Form::text('q', isset($q) ? $q : null, ['class' => 'form-control', 'placeholder' => 'Type name / model...']) !!}
                        {!! $errors->first('q', '<p class="help-block">:message</p>') !!}
                    </div>
                {!! Form::close() !!}
            </div>
        </div>
    </div>
```

```
{!! Form::submit('Search', ['class'=>'btn btn-primary']) !!}
{!! Form::close() !!}


|                       |                        |                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------|------------------------|---------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name                  | Model                  | Category                                                                                                |                                                                                                                                                                                                                                                                                                                                                                   |
| {{ \$product->name }} | {{ \$product->model }} | <span class="label label-primary"> <i class="fa fa-btn fa-tags"></i> {{ \$category-&gt;title }} </span> | {!! Form::model(\$product, ['route' => ['products.destroy', \$product], 'method' => 'delete', 'class' => 'form-inline']) !!} <a !!}="" &gt;edit&lt;="" <="" ['class'="&gt;'btn" a&gt;="" btn-danger="" btn-xs="" form::close()="" form::submit('delete',="" href="{{ route('products.edit', \$product-&gt;id) }}" js-submit-confirm'])="" td="" {!!=""  =""> </a> |


{!! $products->appends(compact('q'))->links() !!}

```

Upss.. banyak ya? :D

Tenang, meskipun banyak ini mirip-mirip koq dengan halaman index untuk kategori. Hanya beberapa baris kita sesuaikan:

- Diisiin table kita menampilkan 4 kolom: nama, model, kategori dan aksi
- Pada kolom kategori, kita melakukan looping untuk menampilkan semua kategori produk dalam bentuk tag.

Selebihnya, sama saja dengan tampilan index untuk kategori.

Tampilan halaman ini akan seperti berikut:

Name	Model	Category	
Nike AeroLoft Bomber	Jaket Wanita	Jaket	Edit delete
Nike Air Force	Sepatu Pria		Edit delete
Nike Air Max	Sepatu Wanita		Edit delete
Nike Air Zoom	Sepatu Wanita		Edit delete
Nike Guild 550	Jaket Pria	Rompi	Edit delete
Nike SB Steele	Jaket Pria	Jaket Rompi	Edit delete

Halaman listing produk

Kita juga perlu menambahkan link untuk halaman ini ke navbar:

resources/views/layouts/app.blade.php

```
....  
<!-- Left Side Of Navbar -->  
<ul class="nav navbar-nav">  
    @if(Auth::check())  
        ....  
        @can('admin-access')  
            <li class="dropdown">  
                ....  
                <ul class="dropdown-menu" role="menu">  
                    ....  
                    <li><a href="{{ route('products.index') }}"><i class="fa fa-\  
btn fa-gift"></i>Products</a></li>  
                </ul>  
            </li>  
        @endcan  
    @endif  
</ul>
```

Menambah Produk

Produk yang kita buat di Reselia dapat memiliki banyak kategori. Untuk memudahkan user dalam memilih banyak kategori, kita akan menggunakan library [selectize.js²⁰⁵](#). Menggunakan library ini, kita akan merubah input `<select>` agar terlihat seperti memilih tag.

Kita install dengan:

```
npm install --save selectize
```

Setelah terinstal, kita konfigurasi agar di compile Elixir:

²⁰⁵ <http://selectize.github.io/selectize.js>

gulpfile.js

```
....  
elixir(function (mix) {  
  mix.sass('app.scss')  
  mix.styles([  
    ....  
    '/node_modules/selectize/dist/css/selectize.bootstrap3.css',  
    ], 'public/css/app.css')  
  
  mix.scripts([  
    ....  
    '/node_modules/selectize/dist/js/standalone/selectize.min.js',  
    'app.js'  
  ])  
})
```

Dan kita tambahkan pada resources/asset/js/app.js:

resources/asset/js/app.js

```
$(document).ready(function () {  
  ....  
  $('.js-selectize').selectize({  
    sortField: 'text'  
  })  
})
```

Kita compile dengan:

```
gulp
```

Disini kita melakukan konfigurasi agar setiap element <select> yang memiliki class js-selectize menggunakan library selectize.

Mari kita buat dulu logic di controllernya:

app/Http/Controllers/ProductsController.php

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

Selanjutnya, kita buat viewnya:

resources/views/products/create.blade.php

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

@section('content')


### New Product


{!! Form::open(['route' => 'products.store', 'files' => true]) !!}
@include('products._form')
{!! Form::close() !!}


@endsection
```

Syntax disini sama dengan syntax untuk halaman create pada kategori. Kita hanya melakukan sedikit penyesuaian. Disini, kita menambahkan 'files' => true agar kita bisa mengupload file pada form ini. Kita juga menggunakan partial view untuk field form:

resources/views/products/_form.blade.php

```
<div class="form-group {!! $errors->has('name') ? 'has-error' : '' !!}>
    {!! Form::label('name', 'Name') !!}
    {!! Form::text('name', null, ['class'=>'form-control']) !!}
    {!! $errors->first('name', '<p class="help-block">:message</p>') !!}
</div>

<div class="form-group {!! $errors->has('model') ? 'has-error' : '' !!}>
    {!! Form::label('model', 'Model') !!}
    {!! Form::text('model', null, ['class'=>'form-control']) !!}
    {!! $errors->first('model', '<p class="help-block">:message</p>') !!}
</div>
```

```

</div>

<div class="form-group {!! $errors->has('category_lists') ? 'has-error' : '' !!}\">
  {!! Form::label('category_lists', 'Categories') !!}
  {{-- Simplify things, no need to implement ajax for now --}}
  {!! Form::select('category_lists[]', [''=> ''+App\Category::lists('title', 'id')\n)->all(), null, ['class'=>'form-control js-selectize', 'multiple']) !!}
  {!! $errors->first('category_lists', '<p class="help-block">:message</p>') !!}
</div>

<div class="form-group {!! $errors->has('photo') ? 'has-error' : '' !!}">
  {!! Form::label('photo', 'Product photo (jpeg, png)') !!}
  {!! Form::file('photo') !!}
  {!! $errors->first('photo', '<p class="help-block">:message</p>') !!}
  @if (isset($model) && $model->photo != '')
    <div class="row">
      <div class="col-md-6">
        <p>Current photo:</p>
        <div class="thumbnail">
          
        </div>
      </div>
    </div>
  @endif
</div>
{!! Form::submit(isset($model) ? 'Update' : 'Save', ['class'=>'btn btn-primary']) !!}

```

Beberapa hal yang perlu di perhatikan di view ini:

- Pada input `Form::select()`, kita menambah `class js-selectize` agar menggunakan library `selectize`. Kita juga mengaktifkan `multiple` agar user dapat memilih lebih dari satu kategori.
- Setelah input foto, kita melakukan pengecekan apakah form ini merupakan halaman edit dan memiliki foto produk yang sudah di upload dengan `if (isset($model) && $model->photo)`. Jika ya, maka kita tampilkan foto tersebut.
- Kita menggunakan syntax `{{ url('/img/' . $model->photo) }}` untuk menampilkan url ke image. Ini kita lakukan karena kita akan menyimpan semua foto produk

di folder public/img/.

Tampilan halaman create akan seperti berikut:

New Product

Name

Model

Harga

Categories

Product photo (jpeg, png)

No file chosen

Halaman tambah produk

Untuk menerima form ini, mari kita ubah method store menjadi:

app/Http/Controllers/ProductsController.php

```
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required|unique:products',
        'model' => 'required',
        'photo' => 'mimes:jpeg,png|max:10240',
        'price' => 'required|numeric|min:1000'
    ]);
}
```

```

$data = $request->only('name', 'model', 'price');

if ($request->hasFile('photo')) {
    $data['photo'] = $this->savePhoto($request->file('photo'));
}

$product = Product::create($data);
$product->categories()->sync($request->get('category_lists'));

\Flash::success($product->name . ' saved.');
return redirect()->route('products.index');
}

```

Pada method ini, kita melakukan beberapa hal:

- Validasi isian form produk. Untuk foto produk, kita batasi hanya jpeg dan png dengan ukuran maksimum 10mb.
- Kita mengambil field `name`, `model` dan `price` langsung dari `request`.
- Jika user mengupload photo, kita akan menyimpan nya dengan nama unik. Proses penyimpanan akan di handle oleh method `savePhoto` yang akan kita jelaskan di penjelasan selanjutnya.
- Setelah semua data siap, kita menyimpan data user dan menambahkan relasi ke Kategori dengan menggunakan method `sync()`.
- Terakhir, kita mengatur pesan feedback dan mengarahkan user ke halaman listing produk.

Mari kita buat method `savePhoto` untuk menyimpan foto produk:

app/Http/Controllers/ProductsController.php

```

.....
use Symfony\Component\HttpFoundation\File\UploadedFile;

class ProductsController extends Controller
{
    .....
    protected function savePhoto(UploadedFile $photo)
    {
        $fileName = str_random(40) . '.' . $photo->guessClientExtension();
        $destinationPath = public_path() . DIRECTORY_SEPARATOR . 'img';
        $photo->move($destinationPath, $fileName);
        return $fileName;
    }
}

```

```

    }
    ...
}

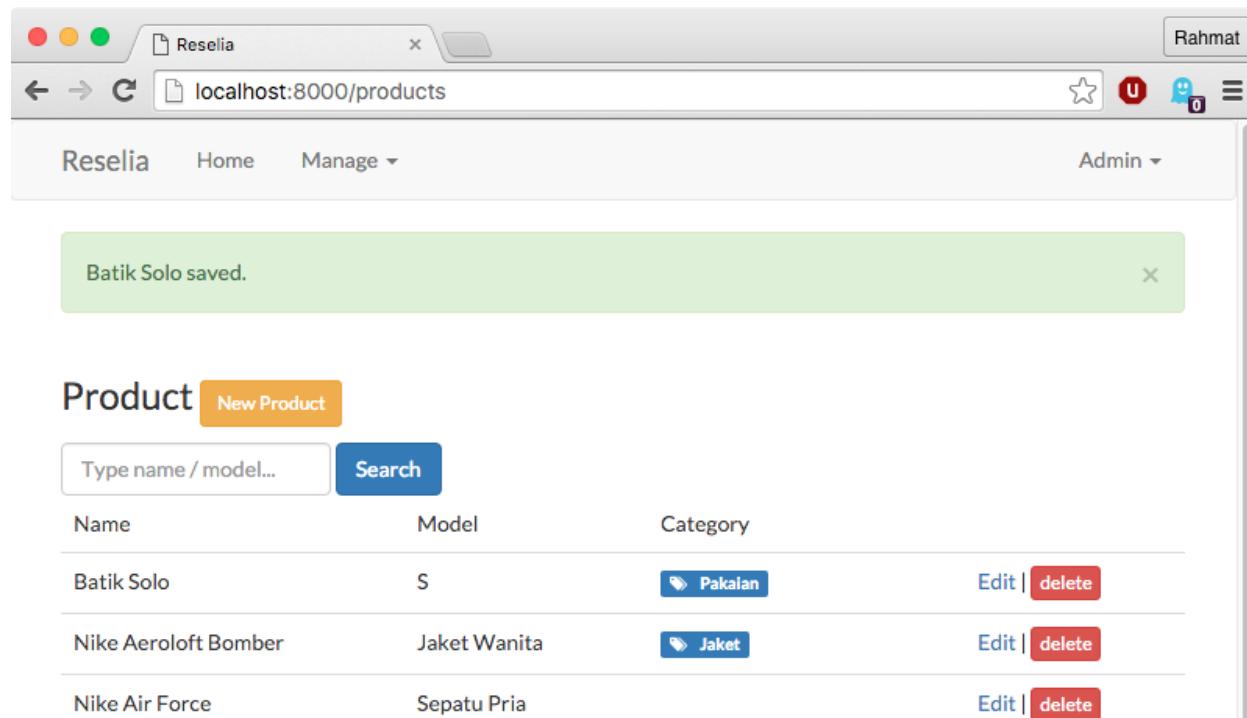
```

Cara kerja method ini dengan:

1. Membuat nama file dengan menentukan sebuah string acak sebanyak 40 karakter dan ditambah dengan ekstensi file.
2. Menentukan tujuan disimpan file ke `public/img`.
3. Memindahkan file ke folder yang dituju dengan nama file yang sudah ditentukan.
4. Mengembalikan nama file yang sudah tersimpan.

Jika diperhatikan pada saat kita hendak menentukan alamat directory, kita menggunakan `DIRECTORY_SEPARATOR`. Ini kita lakukan untuk menghindari error karena unix menggunakan `/` (forward slash) untuk pemisah folder sedangkan windows menggunakan `\` (back slash).

Cobalah fitur ini, pastikan berhasil menambah produk baru.



The screenshot shows a web application interface for managing products. At the top, there's a header with the title 'Reselia' and a user 'Rahmat'. Below the header, the URL 'localhost:8000/products' is visible. The main navigation bar includes 'Reselia', 'Home', 'Manage', and 'Admin'. A green success message box displays 'Batik Solo saved.' with a close button. The main content area is titled 'Product' and features a 'New Product' button. Below this, there's a search bar with a placeholder 'Type name / model...' and a 'Search' button. A table lists four products:

Name	Model	Category	
Batik Solo	S	Pakalan	Edit delete
Nike Aeroloft Bomber	Jaket Wanita	Jaket	Edit delete
Nike Air Force	Sepatu Pria		Edit delete

Berhasil menambah produk baru

Mengubah Produk

Mari kita tambah fitur untuk mengubah produk, tambahkan baris berikut pada method edit:

app/Http/Controllers/ProductsController.php

```
public function edit($id)
{
    $product = Product::findOrFail($id);
    return view('products.edit', compact('product'));
}
```

Kita buat viewnya:

resources/views/products/edit.blade.php

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

@section('content')


### Edit {{ $product->title }}


{!! Form::model($product, ['route' => ['products.update', $product], 'method' => 'patch', 'files' => true]) !!}
@include('products._form', ['model' => $product])
{!! Form::close() !!}


@endsection
```

Kita menggunakan form model binding untuk membuat halaman edit. Sebagai catatan, dipenjelasan sebelumnya kita telah membuat agar image yang dari product yang sedang diedit tampil. Yang sedikit membutuhkan trik adalah menampilkan semua kategori yang dimiliki produk ini. Di form, kita menggunakan nama field `category_lists`. Field ini, tentunya tidak ada. Kita akan membuatnya menggunakan custom accessor agar menampilkan array berisi semua id kategori yang berhubungan dengan produk ini.

app/Product.php

```
public function getCategoryListsAttribute()
{
    if ($this->categories()->count() < 1) {
        return null;
    }
    return $this->categories->lists('id')->all();
}
```

Jika produk memiliki image, berikut tampilan halaman editnya:

The screenshot shows a web application interface for editing a product. At the top, there's a header with the title 'Reselia' and a user 'Rahmat'. Below the header, the URL 'localhost:8000/products/6/edit' is visible. The main content area is titled 'Edit' and contains the following fields:

- Name:** Nike SB Steele
- Model:** Jaket Pria
- Harga:** 1200000.00
- Categories:** Pakaian, Jaket, Rompi
- Product photo (jpeg, png):** A file input field showing 'No file chosen'. Below it, a placeholder text 'Current photo:' followed by an image of a camouflage jacket.

Halaman edit

Mari kita tambahkan logic untuk menerima form ini di method update:

app/Http/Controllers/ProductsController.php

```
public function update(Request $request, $id)
{
    $product = Product::findOrFail($id);
    $this->validate($request, [
        'name' => 'required|unique:products,name,' . $product->id,
        'model' => 'required',
        'photo' => 'mimes:jpeg,png|max:10240',
        'price' => 'required|numeric|min:1000'
    ]);
}
```

```

]);
$data = $request->only('name', 'model', 'price');

if ($request->hasFile('photo')) {
    $data['photo'] = $this->savePhoto($request->file('photo'));
    if ($product->photo !== '') $this->deletePhoto($product->photo);
}

$product->update($data);
if (count($request->get('category_lists')) > 0) {
    $product->categories()->sync($request->get('category_lists'));
} else {
    // no category set, detach all
    $product->categories()->detach();
}

\Flash::success($product->name . ' updated.');
return redirect()->route('products.index');
}

```

Pada method ini, yang kita lakukan sebagai berikut:

1. Kita mencari instance dari produk yang sedang dicari berdasarkan `$id` yang dikirim.
2. Melakukan validasi.
3. Mengambil data `name`, `model` dan `price` dari request.
4. Jika user mengupload foto, itu artinya dia hendak mengganti foto produk. Kita akan menggunakan method `savePhoto` untuk menyimpan foto. Jika saat ini produk memiliki foto, kita menghapusnya dengan method `deletePhoto` yang akan kita jelaskan di penjelasan selanjutnya.
5. Melakukan proses update produk.
6. Jika field kategori diisi, kita sesuaikan relasi ke kategori dengan method `sync`. Jika tidak diisi, kita hapus semua relasi ke kategori dengan method `detach()`.
7. Terakhir kita mengatur pesan feedback dan mengarahkan user ke listing produk.

Proses penghapusan foto akan kita gunakan pada method `update` dan `delete`. Itulah sebabnya kita pindahkan method `deletePhoto`:

app/Http/Controllers/ProductsController.php

```
<?php
....
use File;

class ProductsController extends Controller
{
    ...
    public function deletePhoto($filename)
    {
        $path = public_path() . DIRECTORY_SEPARATOR . 'img'
            . DIRECTORY_SEPARATOR . $filename;
        return File::delete($path);
    }
}
```

Pada method ini kita menggunakan facade `File` untuk memudahkan dalam proses penghapusan file.

Di penjelasan tentang seeder, kita menyimpan file sample image untuk produk di folder `public/img`. Karena sekarang proses update akan menghapus file tersebut, kita harus merubah logic untuk seeder. Caranya, kita akan menyimpan sample image pada folder `database/seeds/img` dan menyalin sample image yang akan kita gunakan ke folder `public/img` pada saat proses seeding.

Buatlah folder `database/seeds/img`. Salin file `stub-jacket.jpg` dan `stub-jacket.jpg` dari folder `public/img` ke folder tersebut. Pada file seeder, kita tambahkan baris berikut:

database/seeds/ProductsSeeder.php

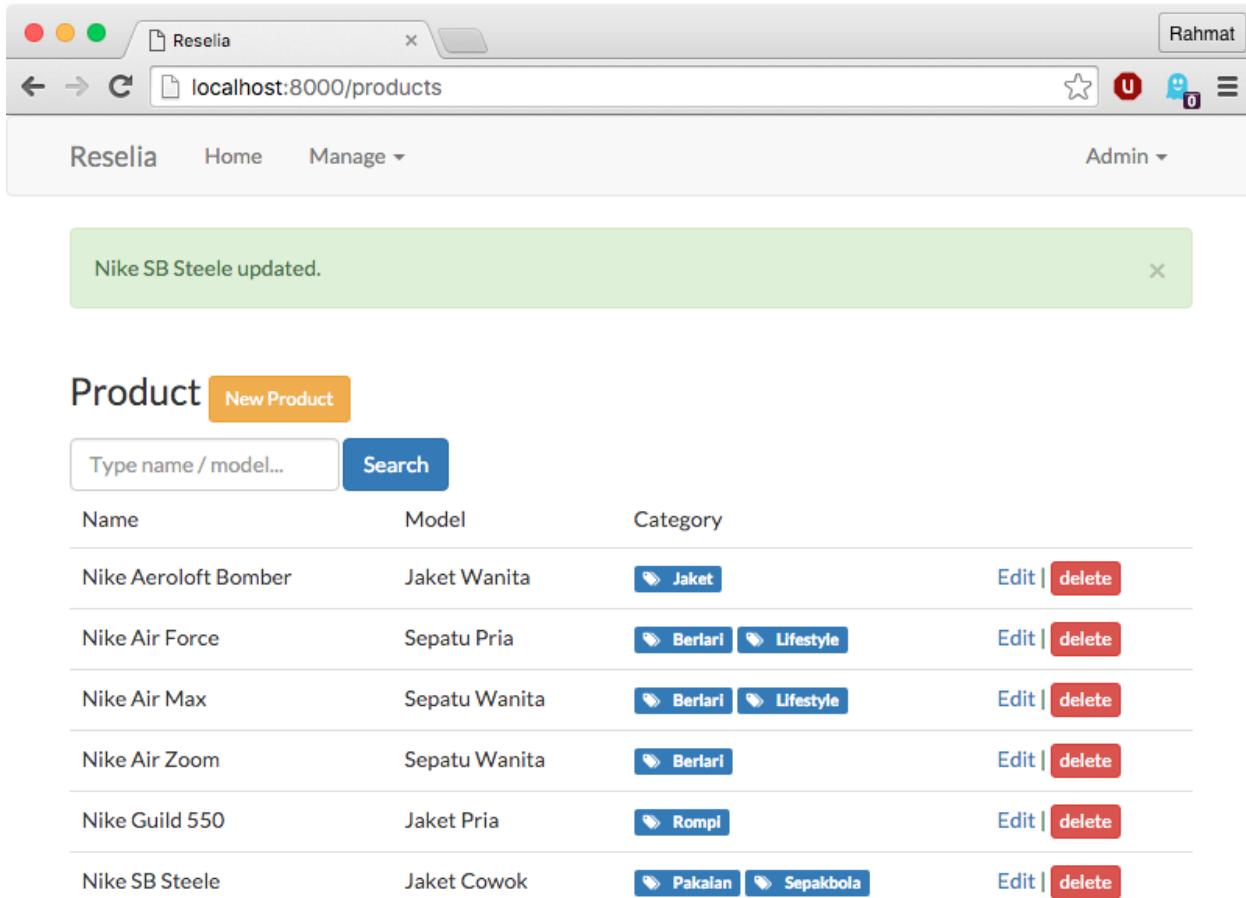
```
public function run()
{
    ...
    // copy image sample to public folder
    $from = database_path() . DIRECTORY_SEPARATOR . 'seeds' .
        DIRECTORY_SEPARATOR . 'img' . DIRECTORY_SEPARATOR;
    $to = public_path() . DIRECTORY_SEPARATOR . 'img' . DIRECTORY_SEPARATOR;
    File::copy($from . 'stub-jacket.jpg', $to . 'stub-jacket.jpg');
    File::copy($from . 'stub-shoe.jpg', $to . 'stub-shoe.jpg');
}
```

Untuk memastikan seeder ini berjalan, hapuslah file pada folder `public/img` dan jalankan:

```
php artisan migrate:refresh --seed
```

Pastikan kedua sample image itu tetap muncul.

Oke, setelah semua logic ditulis, silahkan tes fitur update ini.



The screenshot shows a web application interface for managing products. At the top, there's a header with the brand name 'Reselia', a 'Home' link, a 'Manage' dropdown, and a user account for 'Admin'. Below the header, a green success message box displays the text 'Nike SB Steele updated.' A close button (X) is visible in the top right corner of the message box. The main content area is titled 'Product' and includes a 'New Product' button. There's a search bar with placeholder text 'Type name / model...' and a 'Search' button. A table lists six products:

Name	Model	Category	Action
Nike Aeroloft Bomber	Jaket Wanita	Baket	Edit delete
Nike Air Force	Sepatu Pria	Berlari Lifestyle	Edit delete
Nike Air Max	Sepatu Wanita	Berlari Lifestyle	Edit delete
Nike Air Zoom	Sepatu Wanita	Berlari	Edit delete
Nike Guild 550	Jaket Pria	Rompi	Edit delete
Nike SB Steele	Jaket Cowok	Pakalan Sepakbola	Edit delete

Berhasil mengupdate produk

Menghapus Produk

Mari kita buat fitur untuk menghapus produk. Untuk view, telah kita siapkan pada saat membuat halaman index. Di tahap ini, kita cukup mengubah method `destroy`:

app/Http/Controllers/ProductsController.php

```
public function destroy($id)
{
    $product = Product::find($id);
    if ($product->photo !== '') $this->deletePhoto($product->photo);
    $product->delete();
    \Flash::success('Product deleted.');
    return redirect()->route('products.index');
}
```

Yang kita lakukan di method ini:

1. Mencari instance dari produk berdasarkan `$id` yang dikirim.
2. Jika produk memiliki foto, kita hapus fotonya.
3. Menghapus record produk dari database.
4. Terakhir kita mengatur pesan feedback dan mengarahkan user ke listing produk.

Sebagaimana kategori, pada saat proses penghapusan produk kita juga harus menghapus relasi ke kategori. Mari kita buat pada event `deleting` di model Produk:

app/Product.php

```
public static function boot()
{
    parent::boot();

    static::deleting(function($model) {
        // remove relations to category
        $model->categories()->detach();
    });
}
```

Sip, jika semua logic diatas sudah ditulis, cobalah menghapus produk.

Product deleted.

Name	Model	Category
Nike Air Force	Sepatu Pria	Berlari Lifestyle

Berhasil menghapus produk

Katalog

Setelah kita berhasil menambah produk dan kategori, langkah selanjutnya adalah membuat halaman katalog. Dari halaman katalog ini, customer dapat:

- Melihat semua produk yang ada di Reselia
- Mencari produk dengan keyword tertentu
- Melihat produk per kategori
- Mengurutkan produk berdasarkan harga
- Menambah produk ke cart

Untuk fitur terakhir, akan kita bahas pada pembahasan tentang Cart.

Daftar Produk

Untuk membuat halaman katalog, kita akan membuat `CatalogsController`:

```
php artisan make:controller CatalogsController
```

Kita akan menggunakan method `index` untuk menampilkan halaman katalog. Halaman ini akan kita jadikan halaman default /. Kita ubah route menjadi:

app/Http/routes.php

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

Route::group(['middleware' => 'web'], function () {
    Route::get('/', 'CatalogsController@index');

    ...
});


```

Di controller kita akan menambahkan syntax berikut pada method index:

app/Http/Controllers/CatalogsController.php

```
<?php

...

use App\Product;

class CatalogsController extends Controller
{
    public function index()
    {
        $products = Product::paginate(4);
        return view('catalogs.index', compact('products'));
    }
}
```

Disini kita akan menampilkan 4 produk pada setiap halaman katalog. Kita buat viewnya:

resources/views/catalogs/index.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-3">
                Akan diisi sidebar
            </div>
            <div class="col-md-9">
```

```
<div class="row">
    <div class="col-md-12">
        <ol class="breadcrumb">
            <li>Kategori: Semua Produk</li>
        </ol>
    </div>
    @foreach ($products as $product)
        <div class="col-md-6">
            @include('catalogs._product-thumbnail', ['product' => $product])
        </div>
    @endforeach

    <div class="pull-right">
        {!! $products->links() !!}
    </div>
    </div>
</div>
@endsection
```

Pada view ini, kita akan menampilkan daftar produk dengan melakukan looping pada variable \$products yang dikirim dari controller.

```
@foreach ($products as $product)
<div class="col-md-6">
    @include('catalogs._product-thumbnail', ['product' => $product])
</div>
@endforeach
```

Kita juga menggunakan partial view catalogs._product-thumbnail yang digunakan untuk menampilkan detail dari tiap produk. Kita buat dengan isi sebagai berikut:

resources/views/catalogs/_product-thumbnail.blade.php

```

<h3>{{ $product->name }}</h3>
<div class="thumbnail">
    
    <p>Model: {{ $product->model }}</p>
    <p>Harga: <strong>Rp{{ number_format($product->price, 2) }}</strong></p>
    <p>Category:<br>
        @foreach ($product->categories as $category)
            <span class="label label-primary">
                <i class="fa fa-btn fa-tags"></i>
                {{ $category->title }}</span>
        @endforeach
    </p>
</div>
```

Pada partial view ini, kita menampilkan:

- \$product->name: untuk menampilkan nama produk
- \$product->photo_path: ini merupakan custom accessor yang kita gunakan untuk menggenerate path ke foto untuk produk. Akan kita jelaskan lebih lanjut dibawah.
- number_format(\$product->price, 2): ini kita gunakan untuk menampilkan harga dengan yang sudah terformat dengan dua angka di belakang koma.
- @foreach (\$product->categories as \$category): ini kita gunakan untuk menampilkan semua kategori yang dimiliki oleh produk dalam bentuk label.

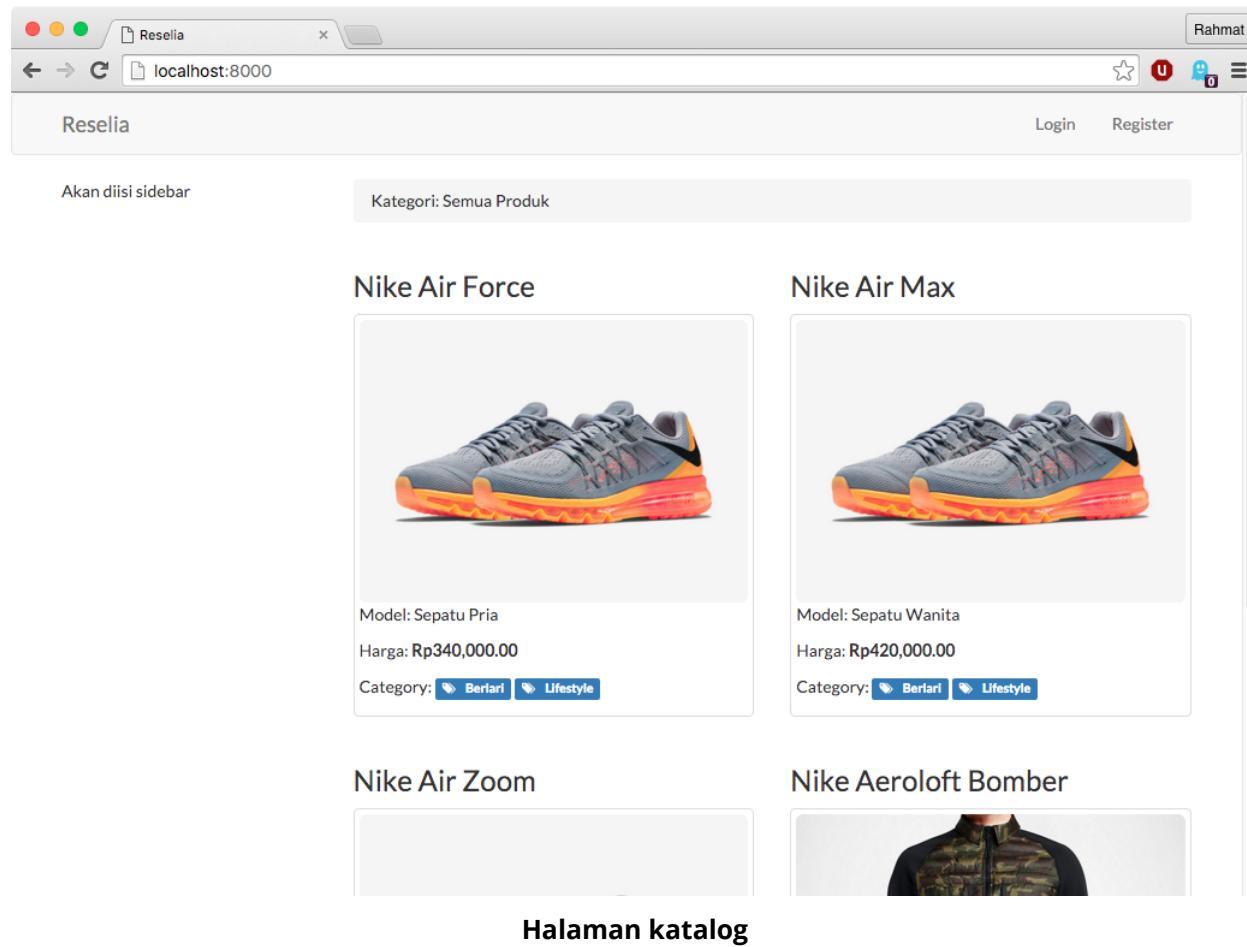
Seperti disebutkan diatas, kita menggunakan custom accessor photo_path untuk menggenerate url ke foto produk. Mari kita buat pada model Product:

app/Product.php

```

public function getPhotoPathAttribute()
{
    if ($this->photo !== '') {
        return url('/img/' . $this->photo);
    } else {
        return 'http://placehold.it/850x618';
    }
}
```

Pada accessor ini, ketika sebuah produk memiliki foto, maka kita akan menampilkan url ke path /img/ + nama file. Sementara jika sebuah produk tidak memiliki foto, kita akan menampilkan placeholder dengan menggunakan service dari http://placehold.it. Tampilan dari halaman katalog akan seperti ini:



Navigasi Kategori

Pada sidebar di katalog, mari kita tambahkan fitur navigasi per kategori. Beberapa hal yang perlu diperhatikan dalam membangun fitur ini:

- Untuk mengakomodasi kategori bisa memiliki sub kategori, kita akan menampilkan sub kategori ketika user mengunjungi halaman kategori.
- Kita akan menampilkan jumlah produk pada tiap kategori dari menu navigasi.

Mari kita siapkan dulu controller agar bisa menerima parameter kategori ketika menampilkan katalog. Ubahlah method index menjadi:

app/Http/Controllers/CatalogsController.php

```
<?php
...
use App\Category;

class CatalogsController extends Controller
{
    public function index(Request $request)
    {
        if ($request->has('cat')) {
            $cat = $request->get('cat');
            $category = Category::findOrFail($cat);
            // we use this to get product from current category and its child
            $products = Product::whereIn('id', $category->related_products_id)
                ->paginate(4);
        } else {
            $products = Product::paginate(4);
        }

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

Ini yang dilakukan syntax diatas:

- Untuk melakukan filter berdasarkan kategori, kita akan menggunakan parameter `cat` dari request.
- Ketika request memiliki parameter `cat`, kita mencari kategori berdasarkan id yang dikirim. Setelah kita mendapatkan kategori, kita menggunakan `whereIn` untuk mencari produk berdasarkan array id produk dari `$category->related_products_id`. Field `related_products_id` merupakan custom accessor yang akan kita jelaskan di penjelasan selanjutnya. Terakhir, kita melakukan pagination pada hasilnya.
- Jika request tidak memiliki parameter `cat`, kita melakukan pagination seperti biasa.
- Ketika mengirim view, kita juga melakukan passing terhadap variable `cat`. Ini akan kita gunakan untuk menambahkan `cat` pada link pagination.

Sebagaimana dijelaskan diatas, kita menggunakan custom accessor `related_products_id` untuk mengambil id produk yang berhubungan dengan produk. Ini kita lakukan agar ketika menampilkan produk untuk sebuah kategori, kita juga menampilkan produk dari sub kategorinya. Berikut syntax untuk accessor ini:

app/Category.php

```
/*
 * Get all product id from active category and its child
 */
public function getRelatedProductsIdAttribute()
{
    $result = $this->products->lists('id')->toArray();
    foreach ($this->childs as $child) {
        $result = array_merge($result, $child->related_products_id);
    }
    return $result;
}
```

Accessor ini merupakan sebuah *recursive function*. Cara kerjanya sebagai berikut:

- Catat semua id dari produk yang bereaksi ke kategori ini pada variable \$variable.
- Lakukan looping pada semua sub-kategori dan memanggil method ini pada child tersebut. Hasil dari tiap child tersebut akan ditambahkan ke variable \$result.
- Berikan array variable \$result.

Mari kita tambahkan sidebar untuk memilih kategori produk pada halaman katalog:

resources/views/catalogs/index.blade.php

```
.....
<div class="col-md-3">
    Akan diisi sidebar
    @include('catalogs._category-panel')
</div>
....
```

Disini kita akan menggunakan partial view catalogs._category-panel untuk menampilkan sidebar kategori:

resources/views/catalogs/_category-panel.blade.php

```
<div class="panel panel-default">
    <div class="panel-heading">
        <h3 class="panel-title">Lihat per kategori</h3>
    </div>
    <div class="list-group">
        <a href="/catalogs" class="list-group-item">Semua produk <span class="badge"\>
        >{{ App\Product::count() }}</span></a>
        @foreach(App\Category::noParent()->get() as $category)
            <a href="{{ url('/catalogs?cat=' . $category->id) }}" class="list-group-item">
                {{ $category->title }}
                {!! $category->total_products > 0 ? '<span class="badge">' . $category->to\
tal_products . '</span>' : '' !!}
            </a>
        @endforeach
    </div>
</div>
```

Disini, kita menggunakan custom scope `noParent` agar hanya kategori yang tidak memiliki parent (kategori utama) yang muncul pada list ini. Untuk sub-kategori akan kita buat pada panel yang lain. Logic untuk custom scope `noParent` akan seperti ini:

app/Category.php

```
public function scopeNoParent($query)
{
    return $this->where('parent_id', '');
}
```

Pada sidebar kategori, kita juga menambahkan total produk dengan menjumlahkan total produk pada kategori tersebut dan total produk pada sub kategori nya. Ini kita lakukan dengan menggunakan custom accessor `$category->total_products`. Jika tidak ada produk, maka kita tidak menampilkannya. Logic untuk custom accessor ini seperti berikut:

app/Category.php

```
public function getTotalProductsAttribute()
{
    return Product::whereIn('id', $this->related_products_id)->count();
}
```

Pada accessor ini, kita juga menggunakan custom accessor `related_products_id` untuk mencari produk yang berhubungan dengan kategori dan subkategorinya. Setelah semua produknya didapatkan, kita hitung dengan method `count()`.

Pada tiap link kategori yang kita tampilkan, kita menggunakan link `/catalogs?cat={category_id}`:

```
url('/catalogs?cat=' . $category->id)
```

Link ini akan kita arahkan ke method `index` di `CatalogsController`. Tentunya, kita harus menyiapkan routenya:

app/Http/routes.php

```
Route::get('/catalogs', 'CatalogsController@index');
```

Setelah semua syntax diatas selesai, kita akan melihat tampilan berikut:

The screenshot shows a web browser window titled 'Reselia' with the URL 'localhost:8000'. The page has a header with 'Reselia' on the left and 'Login' and 'Register' on the right. A sidebar on the left lists categories: 'Lihat per kategori', 'Semua produk' (6), 'Sepatu' (3), and 'Pakaian' (3). The main content area displays four product cards:

- Nike Air Force**: Shows two grey and orange Nike Air Force shoes. Details: Model: Sepatu Pria, Harga: Rp340,000.00, Category: Berlari, Lifestyle.
- Nike Air Max**: Shows two grey and orange Nike Air Max shoes. Details: Model: Sepatu Wanita, Harga: Rp420,000.00, Category: Berlari, Lifestyle.
- Nike Air Zoom**: Shows a placeholder image for the Nike Air Zoom model.
- Nike Aeroloft Bomber**: Shows a placeholder image for the Nike Aeroloft Bomber model.

Halaman katalog

Jika kita coba klik pada kategori pakaian:

Nike Aeroloft Bomber

Model: Jaket Wanita
Harga: Rp720,000.00
Category: Jaket

Nike Guild 550

Model: Jaket Pria
Harga: Rp380,000.00
Category: Rompi

Nike SB Steele

Filter dengan kategori

Terlihat disini, produk yang ditampilkan hanya produk yang memiliki kategori pakaian. Tapi, breadcrumb nya masih berisi tulisan "Kategori: Semua Produk". Mari kita perbaiki ubah agar menampilkan nama kategori yang sedang diakses:

resources/views/catalogs/index.blade.php

```
....  
<ol class="breadcrumb">  
--<li>Kategori: Semua Produk</li>  
</ol>  
  
@include('catalogs._breadcrumb', [  
    'current_category' => isset($category) ? $category : null  
)  
....
```

Disini kita menggunakan partial view `catalogs._breadcrumb` yang akan menerima passing variable `current_category`. Variable tersebut akan berisi `$category` yang kita passing

dari controller. Mari kita ubah pada controller:

app/Http/Controllers/CatalogsController.php

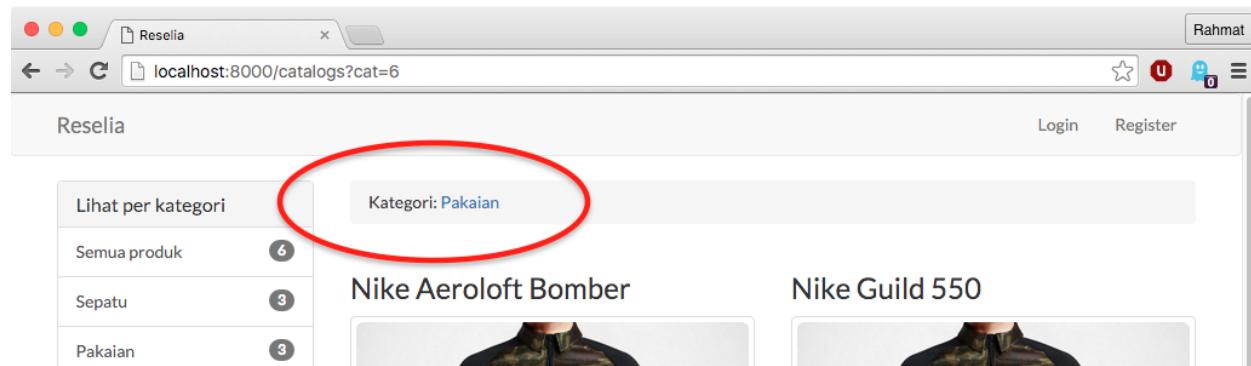
```
public function index(Request $request)
{
    ...
    return view('catalogs.index', compact('products', 'cat', 'category'));
}
```

Untuk partial viewnya akan kita buat seperti ini:

resources/views/catalogs/_breadcrumb.blade.php

```
<ol class="breadcrumb">
    @if(!is_null($current_category))
        <li>Kategori: <a href="{{ url('/catalogs?cat=' . $current_category->id) }}>{{ $current_category->title }}</a></li>
    @else
        <li>Kategori: Semua Produk</a></li>
    @endif
</ol>
```

Disini, jika kategori diisi, maka kita akan menampilkan nama kategori tersebut.



Nama kategori muncul di breadcrumb

Agar pagination tetap berjalan, kita juga perlu menambahkan variable \$cat ke pagination:

resources/views/catalogs/index.blade.php

```
<div class="pull-right">
—{!! $products->links() !!}
{!! $products->appends(compact('cat'))->links() !!}
</div>
```

Agar lebih menarik, mari kita juga tampilkan sub kategori dari kategori yang sedang dipilih pada sidebar:

resources/views/catalogs/index.blade.php

```
.....
<div class="col-md-3">
    @include('catalogs._category-panel')

    @if (isset($category) && $category->hasChild())
        @include('catalogs._sub-category-panel', ['current_category' => $category])
    @endif

    @if (isset($category) && $category->hasParent())
        @include('catalogs._sub-category-panel', [
            'current_category' => $category->parent
        ])
    @endif
</div>
```

Kita akan menggunakan partial view untuk menampilkan sub kategori:

resources/views/catalogs/_sub-category-panel.blade.php

```
<div class="panel panel-default">
    <div class="panel-heading">
        <h3 class="panel-title">Subkategori untuk {{ $current_category->title }}</h3>
    </div>
    <div class="list-group">
        @foreach($current_category->childs as $category)
            <a href="{{ url('/catalogs?cat=' . $category->id) }}" class="list-group-item">{{ $category->title }}
                {!! $category->total_products > 0 ? '<span class="badge">' . $category->total_products . '</span>' : '' !!}
            </a>
        @endforeach
    </div>
</div>
```

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

Tampilan dari navigasi untuk sub kategori ini hampir sama dengan tampilan untuk navigasi kategori. Kita menambahkan nama kategori yang aktif pada panel ini.

Di view ini, kita juga memanggil partial view untuk sub kategori sebanyak 2 kali menggunakan method `$category->hasChild()` dan `$category->hasParent()`. Ini kita lakukan agar panel ini tetap tampil ketika sebuah kategori tidak memiliki child. Mari kita buat kedua method itu:

app/Category.php

```
public function hasParent()
{
    return $this->parent_id > 0;
}

public function hasChild()
{
    return $this->child_id->count() > 0;
}
```

Kini, tampilan pada halaman kategori pakaian akan seperti ini:

Lihat per kategori

Semua produk 6

Sepatu 3

Pakaian 3

Kategori: Pakaian

Nike Aeroloft Bomber

Model: Jaket Wanita

Harga: Rp720,000.00

Category: Jaket

Nike Guild 550

Model: Jaket Pria

Harga: Rp380,000.00

Category: Rompi

Nike SB Steele

Menampilkan sub kategori

Pembahasan untuk navigasi ini bisa sangaaaat panjang, apalagi kalau kita menggunakan library javascript untuk menampilkannya. Di buku ini, saya sengaja tidak membuat navigasi untuk kategori yang terlalu rumit agar kita tetap fokus membahas Laravel.

Pencarian Produk

Fitur pencarian produk akan sangat bermanfaat untuk customer yang tidak memiliki banyak waktu browsing produk. Pada bagian ini, kita akan membuat fitur pencarian berdasarkan nama produk. Untuk mengembangkan fitur ini, kita akan menggunakan query `LIKE` terhadap field `name` dari table `products`. Mari kita buat implementasinya.

Kita siapkan dulu controllernya agar bisa menerima parameter `q`:

app/Http/Controllers/CatalogsController.php

```

public function index(Request $request)
{
    $q = $request->get('q');
    if ($request->has('cat')) {
        $cat = $request->get('cat');
        $category = Category::findOrFail($cat);
        // we use this to get product from current category and its child
        $products = Product::whereIn('id', $category->related_products_id)
            ->where('name', 'LIKE', '%' . $q . '%');
    } else {
        $products = Product::where('name', 'LIKE', '%' . $q . '%');
    }
    $products = $products->paginate(4);

    return view('catalogs.index', compact('products', 'cat', 'category', 'q'));
}

```

Disini, kita menggunakan query

```
where('name', 'LIKE', '%' . $q . '%')
```

Untuk mencari produk yang mengandung nama dengan keyword yang diterima. Proses pagination kita pisahkan dari proses query:

```
$products = $products->paginate(4);
```

Terakhir, kita juga men-passing variable q:

```
return view('catalogs.index', compact('products', 'cat', 'category', 'q'));
```

Pada view, kita akan menambahkan panel untuk search diatas navigasi kategori:

resources/views/catalogs/index.blade.php

```
....  
<div class="col-md-3">  
    @include('catalogs._search-panel', [  
        'q' => isset($q) ? $q : null,  
        'cat' => isset($cat) ? $cat : ''  
    ])  
  
    @include('catalogs._category-panel')  
....
```

Disini kita akan menggunakan partial view yang kita passing variable `q` dan `cat` (jika keduanya terisi). Syntaxnya akan seperti berikut:

resources/views/catalogs/_search-panel.blade.php

```
<div class="panel panel-default">  
    <div class="panel-heading">  
        <h3 class="panel-title">Cari produk</h3>  
    </div>  
    <div class="panel-body">  
        {!! Form::open(['url' => 'catalogs', 'method'=>'get']) !!}  
        <div class="form-group {!! $errors->has('q') ? 'has-error' : '' !!}">  
            {!! Form::label('q', 'Apa yang kamu cari?') !!}  
            {!! Form::text('q', $q, ['class'=>'form-control']) !!}  
            {!! $errors->first('q', '<p class="help-block">:message</p>') !!}  
        </div>  
  
        {!! Form::hidden('cat', $cat) !!}  
  
        {!! Form::submit('Cari', ['class'=>'btn btn-primary']) !!}  
        {!! Form::close() !!}  
    </div>  
</div>
```

Terlihat disini, form ini hanya memiliki dua field:

- `q`: untuk menyimpan keyword yang dipakai untuk pencarian
- `cat`: untuk menyimpan kategori yang aktif. Ini kita lakukan agar pencarian hanya pada kategori tersebut.

Tentunya akan ada saat dimana proses pencarian produk tidak dapat menemukan produk yang sesuai. Untuk itu, kita akan menampilkan link untuk melakukan pencarian pada semua kategori (jika pencarian dilakukan pada sebuah kategori) dan link untuk melihat semua produk:

resources/views/catalogs/index.blade.php

```
....  
@foreach ($products as $product)  
    <div class="col-md-6">  
        @include('catalogs._product-thumbnail', ['product' => $product])  
    </div>  
@endforeach  
  
@forelse ($products as $product)  
    <div class="col-md-6">  
        @include('catalogs._product-thumbnail', ['product' => $product])  
    </div>  
@empty  
    <div class="col-md-12 text-center">  
        @if (isset($q))  
            <h1>:(</h1>  
            <p>Produk yang kamu cari tidak ditemukan.</p>  
            @if (isset($category))  
                <p><a href="{{ url('/catalogs?q=' . $q) }}>Cari di semua kategori <i class="fa fa-arrow-right"></i></a></p>  
            @endif  
        @else  
            <h1>:|</h1>  
            <p>Belum ada produk untuk kategori ini.</p>  
        @endif  
        <p><a href="{{ url('/catalogs') }}>Lihat semua produk <i class="fa fa-arrow-right"></i></a></p>  
    </div>  
@endforelse
```

Disini kita menggunakan `forelse` untuk menampilkan produk yang ditemukan atau menampilkan link ketika produk tidak ditemukan.

Terakhir, kita juga perlu mengubah link pagination agar menambah variable `q` ketika sudah terisi:

resources/views/catalogs/index.blade.php

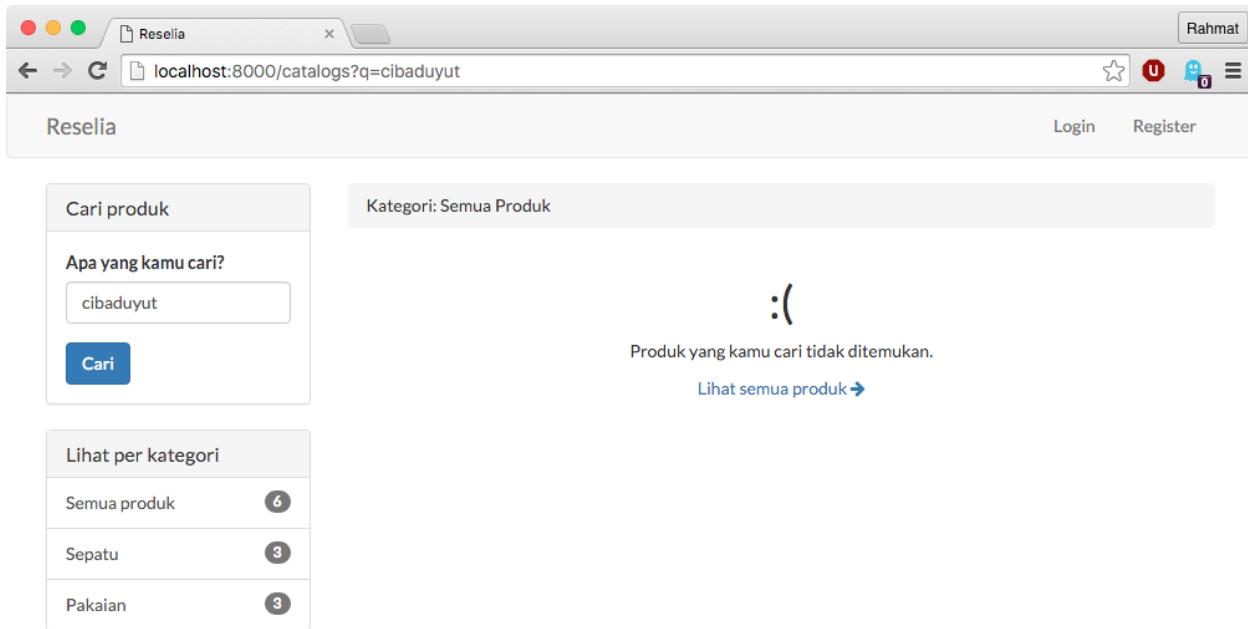
```
<div class="pull-right">
{!! $products->appends(compact('cat'))->links() !!}
{!! $products->appends(compact('cat', 'q'))->links() !!}
</div>
```

Sip, ini tampilan ketika kita tidak berhasil mencari produk pada kategori sepatu:

The screenshot shows a web browser window titled 'Reselia'. The address bar displays 'localhost:8000/catalogs?q=cibaduyut&cat=1'. The main content area has a heading 'Kategori: Sepatu'. On the left, there's a search form with 'Cari produk' placeholder, an input field containing 'cibaduyut', and a blue 'Cari' button. Below it is a sidebar titled 'Lihat per kategori' with three items: 'Semua produk' (6), 'Sepatu' (3), and 'Pakaian' (3). A message at the bottom left says 'Subkategori untuk Sepatu'. In the center, there's a large ':('. To its right, a message says 'Produk yang kamu cari tidak ditemukan.' with links 'Cari di semua kategori' and 'Lihat semua produk'.

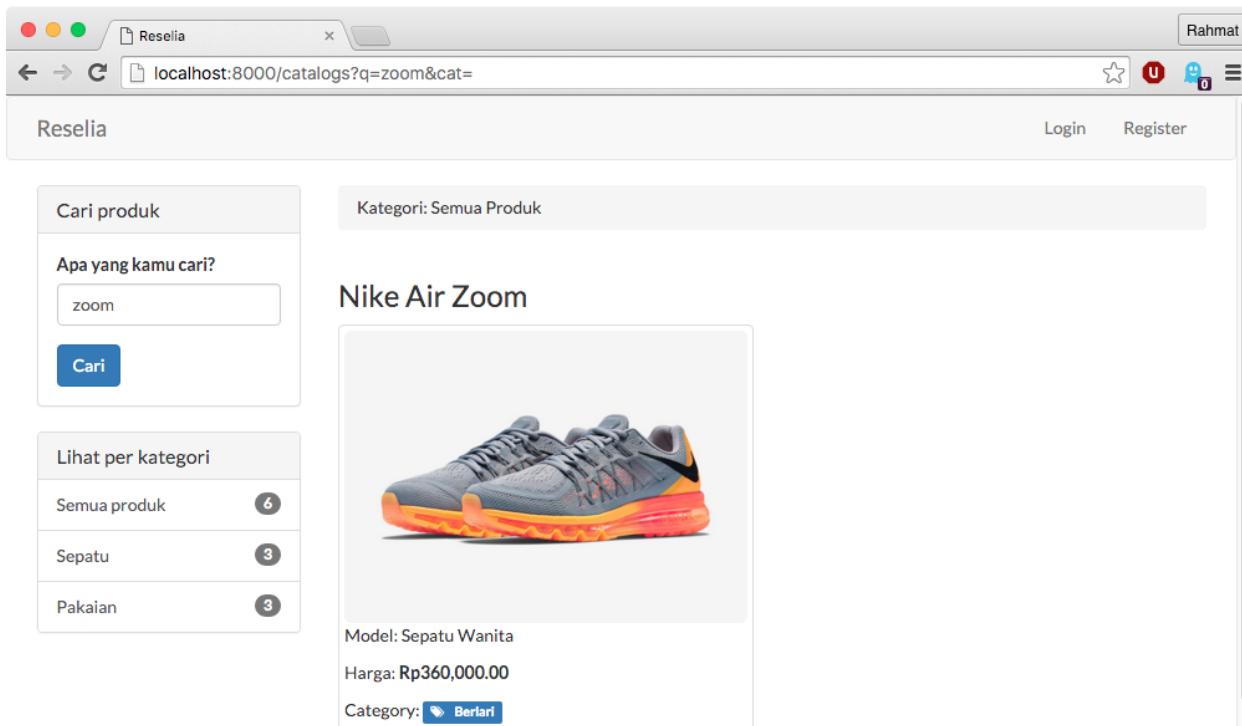
Tidak ada produk pada kategori

Ketika tidak ada produk pada semua kategori:



Tidak ada produk pada semua kategori

Dan ketika berhasil menemukan produk:



Berhasil mencari produk

Sebagai latihan, cobalah tampilkan keyword yang digunakan user dan kategori yang sedang aktif pada saat dia melakukan pencarian.

Mengurutkan Produk berdasarkan Harga

Fitur sorting merupakan fitur yang umum dimiliki oleh sebuah website. Pada fitur katalog yang sedang kita bangun, akan sangat bermanfaat jika kita membangun sorting berdasarkan harga termahal atau termurah. Untuk membuat fitur ini, kita memanfaatkan fitur `orderBy` di Eloquent.

Syntax pada method `index` akan kita ubah seperti ini:

app/Http/Controllers/CatalogsController.php

```
public function index(Request $request)
{
    ...
    if ($request->has('sort')) {
        $sort = $request->get('sort');
        $order = $request->has('order') ? $request->get('order') : 'asc';
        $field = in_array($sort, ['price', 'name']) ? $request->get('sort') : 'p\
rice';
    }
}
```

```

    $products = $products->orderBy($field, $order);
}

$products = $products->paginate(4);

return view('catalogs.index', compact('products', 'q', 'cat', 'selected_category',
    'sort', 'order'));
}

```

Disini, kita menggunakan field `sort` untuk menentukan nama field yang akan diurutkan. Dalam kasus ini, kita akan menggunakan field `price`. Tapi, sebenarnya kita juga mengizinkan mengurutkan dengan field `name`. Hanya saja default kita set ke field `price`:

```
$field = in_array($sort, ['price', 'name']) ? $request->get('sort') : 'price';
```

Untuk metode pengurutan kita atur default di `asc`:

```
$order = $request->has('order') ? $request->get('order') : 'asc';
```

Dan kita urutkan dengan method `orderBy`:

```
$products = $products->orderBy($field, $order);
```

Tidak lupa, kita juga passing variable `sort` dan `order` ke view:

```
return view('catalogs.index', compact('products', 'q', 'cat', 'selected_category',
    'sort', 'order'));
```

Pada view, kita akan menambahkan pengurutan ini sebagai sebuah tombol di sebelah kanan breadcrumb:

resources/views/catalogs/_breadcrumb.blade.php

```
<ol class="breadcrumb">
    ...
    <span class="pull-right">Urutkan Harga:
        <a href="{{ appendQueryString(['sort'=>'price', 'order'=>'asc']) }}"
            class="btn btn-default btn-xs
            {{ isQueryStringEqual(['sort'=>'price', 'order'=>'asc']) ? 'active' : ''}}>Termurah</a> |
        <a href="{{ appendQueryString(['sort'=>'price', 'order'=>'desc']) }}"
            class="btn btn-default btn-xs
            {{ isQueryStringEqual(['sort'=>'price', 'order'=>'desc']) ? 'active' : ''}}>Termahal</a>
    </ol>
```

Pada view ini, kita menggunakan dua method custom yang tidak ada di Laravel:

- `appendQueryString()`: untuk merubah url aktif dan menambahkan array asosiatif ke query string.
- `isQueryStringEqual`: untuk mengecek apakah url yang aktif memiliki query string dengan key dan value dengan array asosiatif yang dijadikan parameter.

Ada beberapa cara untuk menyimpan kedua fungsi ini di Laravel. Disini kita akan menggunakan fitur autoloader di composer untuk meload fungsi tersebut pada file yang kita buat di `app/Support/Helpers.php`:

app/Support/Helpers.php

```
<?php
function appendQueryString($params) {
    $query = Request::all();
    foreach ($params as $key => $value) {
        $query[$key] = $value;
    }
    return Request::url(). '?' . http_build_query($query);
}

function isQueryStringEqual($params) {
    return !array_diff($params, Request::all());
}
```

Cara kerja fungsi `appendQueryString` seperti berikut:

- Kita mengambil array asosiatif dari semua parameter request dengan menggunakan method `all()`. Hasil dari method ini kita simpan pada variable `$query`.
- Parameter dari fungsi ini adalah sebuah array asosiatif. Menggunakan `foreach`, kita melakukan *merging* parameter tersebut dengan variable `$query`.
- Terakhir, kita kembalikan url aktif dengan parameter yang sudah dibuat ulang menggunakan fungsi `http_build_query`²⁰⁶.

Sedangkan cara kerja fungsi `isQueryString` cukup sederhana. Kita hanya menggunakan fungsi `array_diff`²⁰⁷ dengan parameter pertama berisi array asosiatif key dan value yang ingin kita cek dan parameter kedua berisi array asosiatif key dan value yang dimiliki url aktif. Fungsi array `array_diff` akan menampilkan nilai `[]` (array kosong) jika array parameter pertama ada di array parameter kedua. Inilah alasan kita menggunakan negasi (`!`). Dengan menggunakan negasi, kita akan mendapatkan nilai `true` ketika hasil dari fungsi `array_diff` berupa `[]`.

Fungsi `isQueryString` kita gunakan untuk menambah class `active` pada link sorting:

```
isQueryStringEqual(['sort'=>'price', 'order'=>'asc']) ? 'active' : ''
```

Untuk meload file helper ini, kita perlu menambah baris berikut pada bagian `autoload` di file `composer.json`. Sehingga isian bagian tersebut akan seperti berikut:

composer.json

```
....  
"autoload": {  
    "classmap": [  
        "database"  
    ],  
    "psr-4": {  
        "App\\": "app/"  
    },  
    "files": [  
        "app/Support/Helpers.php"  
    ]  
},  
....
```

Setiap kali kita merubah isian `autoload` di file `composer.json`, kita harus menjalankan perintah berikut untuk meng-*generate* ulang file autoload:

²⁰⁶ <http://php.net/manual/en/function.http-build-query.php>

²⁰⁷ <http://php.net/manual/en/function.array-diff.php>

```
composer dump-autoload
```

Agar pagination tetap berjalan dengan benar, kita juga harus menambahkan variable sort dan order ke pagination:

resources/views/catalogs/index.blade.php

```
....  

{!! $products->appends(compact('cat', 'q', 'sort', 'order'))->links() !!}  

{!! $products->appends(compact('cat', 'q'))->links() !!}  

....
```

Ini hasil yang akan kita dapatkan ketika mencoba mengurutkan semua produk berdasarkan harga termahal:

The screenshot shows a web application interface for a catalog. At the top, there's a header with the title 'Reselia' and a user 'Rahmat'. Below the header, the URL 'localhost:8000/catalogs?sort=price&order=desc' is visible. The main content area has three search/filter sections: 'Cari produk' (Search product) with a search input and 'Cari' button, 'Kategori: Semua Produk' (Category: All Products), and 'Urutkan Harga: Termurah | Termahal' (Sort by Price: Lowest | Highest). Below these are four product cards:

- Nike SB Steele**: Model: Jaket Pria, Harga: Rp1,200,000.00, Category: Jaket, Rompi.
- Nike Aeroloft Bomber**: Model: Jaket Wanita, Harga: Rp720,000.00, Category: Jaket.
- Nike Air Max**: Partially visible card.
- Nike Guild 550**: Partially visible card.

Katalog berdasarkan harga termahal

Kita juga bisa mengurutkan dalam satu kategori. Misalnya, kita urutkan berdasarkan harga termurah pada kategori pakaian:

The screenshot shows a web application interface for a shopping site named "Reselia". The top navigation bar includes a user profile "Rahmat", "Login", and "Register" buttons. On the left, there's a search bar with placeholder "Apa yang kamu cari?", a "Cari" button, and a sidebar for "Lihat per kategori" with links for "Semua produk" (6 items), "Sepatu" (3 items), "Pakaian" (3 items), "Subkategori untuk Pakaian" (with "Jaket" (2 items), "Hoodie", and "Rompis"), and a sorting section "Urutkan Harga: Termurah | Termahal". The main content area shows three product cards:

- Nike Guild 550**: Model: Jaket Pria, Harga: Rp380,000.00, Category: Rompi.
- Nike Aeroloft Bomber**: Model: Jaket Wanita, Harga: Rp720,000.00, Category: Jaket.
- Nike SB Steele**: (Thumbnail shown but no detailed description or price).

Mengurutkan pada kategori

Terakhir, kita juga bisa mengurutkan pada hasil pencarian.

Cari produk

Apa yang kamu cari?

Reselia

Kategori: Semua Produk

Urutkan Harga: Termurah | Termahal

Reselia

Login Register

Nike Air Max

Nike Air Zoom

Nike Air Force

Mengurutkan hasil pencarian

Berbagai tipe Cart dan Checkout

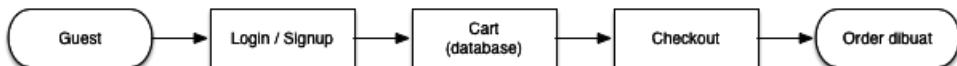
Dalam sebuah website E-commerce, cart dan checkout memiliki peran yang sangat penting dalam proses bisnis. Ada beberapa teknik dalam mengelola dua fitur ini, mari kita bahas dua teknik yang sejauh ini pernah saya temui dari yang paling mudah untuk dibuat hingga yang cukup kompleks.

Teknik pertama adalah membatasi proses penambahan cart dan checkout hanya pada user yang sudah login. Teknik ini tergolong mudah untuk diimplementasikan. Kelebihan dari teknik ini adalah kita dapat memastikan setiap barang yang user masukan ke dalam cartnya tersimpan di *persistent storage* (misalnya database). Sehingga, kapanpun user membuka website, asalkan dia telah login, produk yang telah ditambahkan ke dalam cart tetap tersimpan. Pada saat buku ini ditulis, [Tokopedia](https://www.tokopedia.com)²⁰⁸ menggunakan teknik ini.

Salah satu kekurangan dari teknik pertama adalah kita membuat proses penambahan cart & checkout menjadi lebih lama. Terkadang, ada user yang hanya ingin

²⁰⁸ <https://www.tokopedia.com>

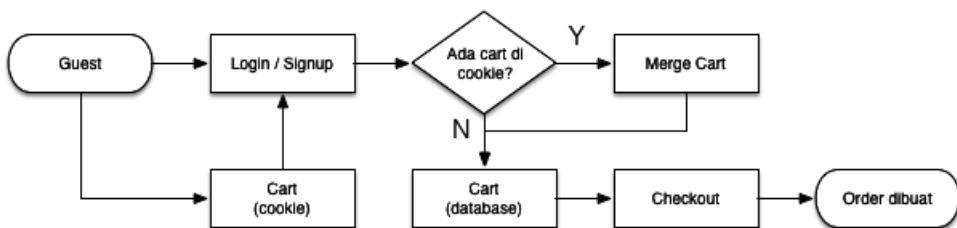
melihat-lihat produk. Pada beberapa kasus, ketika user menambahkan produk ke cart dia hanya sekedar ingin melakukan “bookmark” terhadap produk yang ingin dia beli.



Teknik Cart & Checkout 1

Teknik kedua merupakan peningkatan dari tipe pertama. Teknik ini menggunakan cookie untuk menyimpan data cart sehingga user tidak perlu login untuk menambah produk ke cart. Sedangkan ketika user telah login, data cart di cookie akan dihapus dan digabungkan dengan data cart di *persistent storage*.

Dengan mengizinkan penambahan cart sebelum user login, diharapkan akan memudahkan untuk user baru untuk menambah produk. Untuk proses checkout, masih harus dilalui dengan login atau membuat akun terlebih dahulu. Pada saat buku ini ditulis, [Jaknot²⁰⁹](#) menggunakan teknik ini.



Teknik Cart dan Checkout 2

Teknik ketiga merupakan yang paling kompleks untuk dikembangkan. Teknik ini meningkatkan teknik kedua dengan mengizinkan user untuk menambah cart & checkout tanpa harus login atau membuat akun terlebih dahulu. Ini bisa dilakukan karena pada saat proses checkout user cukup memasukan email. Dibalik layar, kita akan mengecek email tersebut:

1. Jika email belum ada, kita buat akun tanpa password setelah proses checkout selesai.
2. Jika email sudah ada dengan password, kita minta user untuk memasukan password.
3. Jika email sudah ada tanpa password, kita tampilkan halaman untuk mengirim link untuk merubah password.

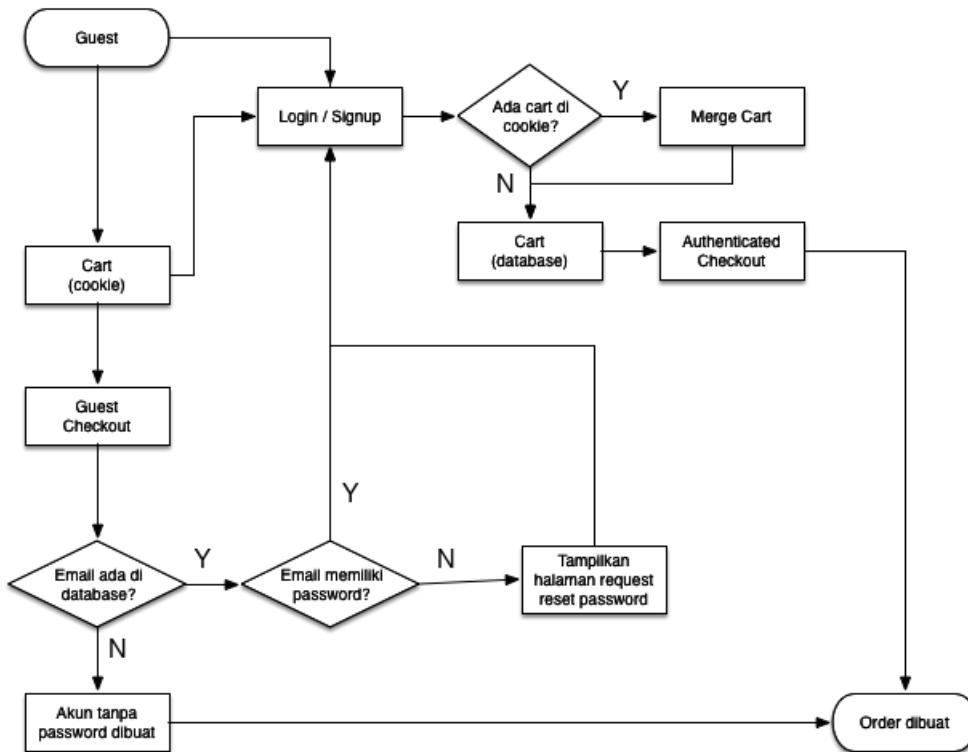
Teknik ini terbukti dapat meningkatkan jumlah penjualan di sebuah website e-commerce. Pada saat buku ini ditulis, [Lazada²¹⁰](#) dan [BukaLapak²¹¹](#) menggunakan

²⁰⁹<https://www.jakartanotebook.com>

²¹⁰<https://www.lazada.co.id>

²¹¹<https://www.bukalapak.com>

teknik ini.



Teknik Cart dan Checkout 3

Pada buku ini, kita akan menggunakan teknik yang ke-3. Kenapa? Karena buku ini keren..:D

Cart

Seperti yang dijelaskan sebelumnya kita akan menggunakan dua metode untuk menyimpan cart.

- Untuk guest, kita akan menggunakan cookie.
- Untuk user yang sudah login (authenticated), kita akan menggunakan table carts.

Ketika user login dan dia memiliki data cart di cookie, kita akan menggabungkan data cart tersebut dengan data di table carts. Pembahasan untuk pembuatan fitur ini akan kita pisahkan antara cart untuk guest dan authenticated.

Mari kita mulai dengan menyiapkan controller dan view untuk menambah produk ke cart. Buatlah controller dengan nama CartController:

```
php artisan make:controller CartController
```

Untuk view, akan kita tambahkan sebagai sebuah form dimana user bisa mengisi jumlah produk yang ingin dia order. Form ini akan kita simpan pada tiap thumbnail produk:

resources/views/catalogs/_product-thumbnail.blade.php

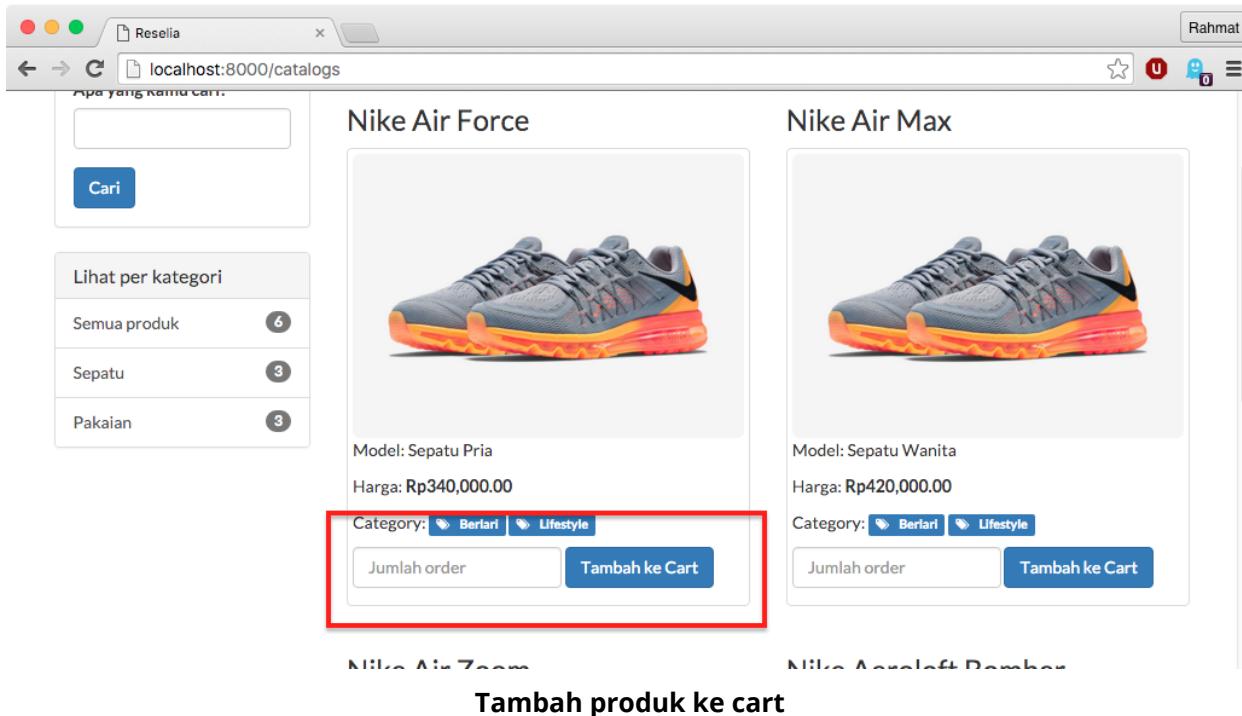
```
....  
<div class="thumbnail">  
    ....  
    @can('customer-access')  
        @include('catalogs._add-product-form', compact('product'))  
    @else  
        @if (auth()->guest())  
            @include('catalogs._add-product-form', compact('product'))  
        @endif  
    @endcan  
</div>
```

Tampilan diatas akan membuat form ini muncul pada saat user belum login atau sudah login dan memiliki role customer. Kita menggunakan partial view catalogs._add-product-form dan mengirimkan variable \$product.

resources/views/catalogs/_add-product-form.blade.php

```
<p>  
    {!! Form::open(['url' => 'cart', 'method'=>'post', 'class'=>'form-inline']) !!}  
    {!! Form::hidden('product_id', $product->id) !!}  
  
    <div class="form-group">  
        {!! Form::number('quantity', null, ['class'=>'form-control', 'min' => 1,\  
        'placeholder' => 'Jumlah order']) !!}  
    </div>  
    {!! Form::submit('Tambah ke Cart', ['class'=>'btn btn-primary']) !!}  
    {!! Form::close() !!}  
</p>
```

Di form ini, kita menggunakan validasi html5 untuk memastikan isian berupa angka dan minimal 1. Kita juga mengirim field product_id berisi id produk yang aktif. Untuk menangani pesan error, akan kita bahas dipembahasan selanjutnya. Tampilan dari form ini akan seperti berikut:



Tambah produk ke cart

Kita juga akan menampilkan link untuk menampilkan data cart pada navbar. Sebagaimana form cart, link ini juga hanya bisa diakses oleh user yang belum login atau sudah login dengan role customer. Kita bisa saja menggunakan logic yang sama seperti menampilkan form, tapi mari kita gunakan teknik yang saya kembangkan bernama "Mediator View". Sebagaimana namanya, teknik ini menjadi perantara untuk menampilkan sebuah partial view.

Caranya, kita buat partial view yang akan menyimpan logic untuk menampilkan view, misalnya di `layouts._customer-feature.blade.php`:

`resources/views/layouts/_customer-feature.blade.php`

```
@can('customer-access')
    @include($partial_view, isset($data) ? $data : [])
@else
    @if (auth()->guest())
        @include($partial_view, isset($data) ? $data : [])
    @endif
@endcan
```

Disini, view `layouts._customer-feature` kita gunakan untuk menyimpan logic untuk menampilkan view yang lain. Cara menggunakannya seperti berikut:

resources/views/catalogs/_product-thumbnail.blade.php

```
....  
<div class="thumbnail">  
    ....  
    @can('customer-access')  
        @include('catalogs._add-product-form', compact('product'))  
    @else  
        @if (auth()>guest())  
            @include('catalogs._add-product-form', compact('product'))  
        @endif  
    @endcan  
  
    @include('layouts._customer-feature', ['partial_view'=>'catalogs._add-product-form', 'data' => compact('product')])  
</div>
```

Terlihat disini, kita hanya menggunakan satu `@include` pada thumbnail produk untuk menampilkan form. Pada saat memanggil view `layouts._customer-feature`, kita passing `partial_view` berisi view yang ingin ditampilkan dan `data` yang akan berisi data yang ingin kita passing pada view tersebut (jika ada).

Kini, kita juga bisa menggunakan mediator view ini untuk menampilkan link navbar untuk mengakses halaman cart:

resources/views/layouts/app.blade.php

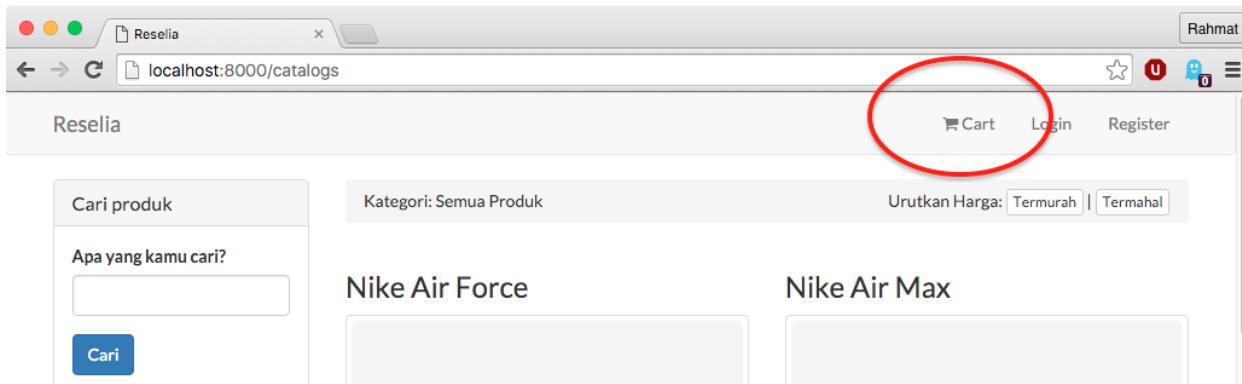
```
....  
<!-- Right Side Of Navbar -->  
<ul class="nav navbar-nav navbar-right">  
  
    @include('layouts._customer-feature', ['partial_view'=>'layouts._cart-menu-bar'])  
    ....
```

Disini, kita passing partial view `layouts._cart-menu-bar`. Isinya akan seperti berikut:

resources/views/layouts/_cart-menu-bar.blade.php

```
<li><a href="{{ url('/cart') }}><i class="fa fa-shopping-cart"></i> Cart</a></li>
```

Tampilan yang kita dapatkan akan seperti berikut:



Link ke cart

Tentunya, link untuk cart ini belum berfungsi karena kita belum membuat fungsionalitasnya.

Cart untuk Guest

Mari kita mulai pembuatan alur checkout untuk guest.

Menambah Produk

Pada tahap persiapan, kita telah menyiapkan form dengan action ke /cart dengan method POST. URL tersebut akan kita arahkan pada method addProduct di CartController. Berikut route nya:

app/Http/routes.php

```
....  
Route::group(['middleware' => 'web'], function () {  
    ....  
    Route::post('cart', 'CartController@addProduct');  
});
```

Pada controller, ini method yang akan kita buat:

app/Http/Controllers/CartController.php

```
<?php  
....  
use App\Product;  
  
class CartController extends Controller  
{  
    public function addProduct(Request $request)  
    {  
        $this->validate($request, [  
            'product_id' => 'required|exists:products,id',  
            'quantity' => 'required|integer|min:1'  
        ]);  
  
        $product = Product::find($request->get('product_id'));  
        $quantity = $request->get('quantity');  
  
        $cart = $request->cookie('cart', []);  
        if (array_key_exists($product->id, $cart)) {  
            $quantity += $cart[$product->id];  
        }  
        $cart[$product->id] = $quantity;  
        return redirect('catalogs')  
            ->withCookie(cookie()->forever('cart', $cart));  
    }  
}
```

Berikut penjelasan syntax diatas:

- Kita melakukan validasi untuk memastikan field `product_id` dan `quantity` diisi. Tentunya, kita memastikan `product_id` memang ada di database dengan validasi `exists` dan field `quantity` tidak kurang dari 1.
- Kita cari instance product dengan method `Product::find()` dan menyimpannya ke variable `$product`.
- Quantity yang dikirim oleh user juga kita simpan pada variable `$quantity`.
- Jika produk sudah ada di dalam cart, kita menambah quantitynya sesuai jumlah yang dikirim oleh user.
- Menggunakan method `$request->cookie()` kita mengambil nilai cookie untuk key `cart` dan membuat isian defaultnya array kosong `[]`. Format dari cookie untuk menyimpan `cart` adalah sebagai berikut:

```
[ 'product_id'=>'quantity', 'product_id'=>'quantity' ]
```

- Menggunakan fungsi `array_key_exists`²¹² kita mengecek apakah product dengan id yang dikirim oleh user sudah ada di cart. Jika sudah, kita tambah `$quantity` yang dikirim user dengan quantity yang dimiliki cart.
- Selanjutnya, kita ubah quantity untuk produk yang dikirim.
- Terakhir, kita arahkan user kembali ke halaman katalog dengan cookie `cart` berisi data cart yang baru. Menggunakan method `cookie()->forever()` kita akan mengeset cookie untuk disimpan browser selama 5 tahun. Tentunya, cookie ini akan hilang jika kita menghapus manual atau user membersihkan cookie dari browsernya.

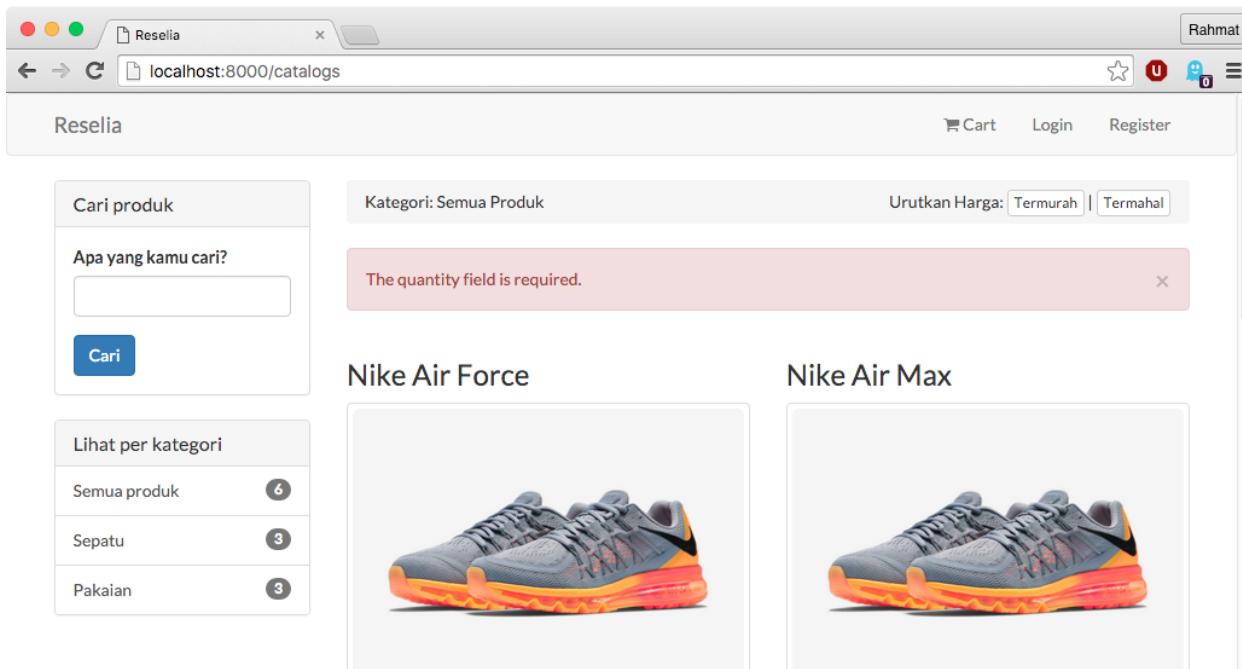
Karena kita menggunakan nama field yang sama untuk tiap form di produk, kita tidak bisa menambahkan pesan error langsung di dalam form. Dengan cara itu, ketika terjadi error, maka semua produk akan menampilkan pesan error. Yang akan kita lakukan adalah dengan menampilkan pesan error dibawah breadcrumb:

resources/views/catalogs/index.blade.php

```
....  
<div class="col-md-12">  
    @include('catalogs._breadcrumb', [  
        'current_category' => isset($selected_category) ? $selected_category : null  
    ])  
    @if ($errors->has('quantity'))  
        <div class="alert alert-danger">  
            <button type="button" class="close" data-dismiss="alert" aria-hidden="true" style="font-size: 1em; font-weight: bold;">&times;            {{ $errors->first('quantity') }}  
        </div>  
    @endif  
....
```

Ini yang akan tampil ketika kita tidak mengisi quantity:

²¹²<http://php.net/manual/en/function.array-key-exists.php>



Validasi quantity

Silahkan ubah pesan validasi sesuai dengan kebutuhan.

Ketika kita berhasil menambah produk ke cart, maka akan muncul cookie `cart` di browser:

The screenshot shows a browser window for the 'Reselia' application at 'localhost:8000/catalogs'. The top navigation bar includes 'Cart', 'Login', and 'Register' buttons. The main content area has a search bar ('Cari produk') and a filter section ('Kategori: Semua Produk'). Below this is a table for sorting products by price ('Urutkan Harga: Termurah | Termahal'). On the left, a sidebar titled 'Elements' lists storage types: 'Frames', 'Web SQL', 'IndexedDB', 'Local Storage', 'Session Storage', and 'Cookies'. A red arrow points from the 'Cart' button in the top bar to the 'Resources' tab in the sidebar. The 'Cookies' section is expanded, showing three entries: 'XSRF-TOKEN', 'cart', and 'laravel_session'. The 'cart' cookie is highlighted.

Berhasil menambah produk ke cart

Tentunya isian cookie ini akan terenkripsi agar user tidak dapat mengubahnya dengan mudah.

Menambah Feedback

Sama dengan fitur lainnya, akan lebih baik jika kita memberikan feedback ke user ketika dia berhasil menambah produk ke cart. Disini kita akan menggunakan sweet alert yang sudah kita instal di tahap sebelumnya untuk menampilkan pesan sukses.

Caranya, kita akan menambahkan nama produk yang berhasil ditambahkan pada session flash dengan nama `flash_product_name`. Pada view, kita akan mengecek data tersebut dan menampilkan pesan sukses yang sesuai. Mari kita tambahkan syntax di controller:

app/Http/Controllers/CartController.php

```
use Session;

class CartController extends Controller
{

    public function addProduct(Request $request)
    {
        $this->validate($request, [
            ...
        ]);
    }
}
```

```

'product_id' => 'required|exists:products,id',
'quantity' => 'required|integer|min:1'
]);

$product = Product::find($request->get('product_id'));
$quantity = $request->get('quantity');
Session::flash('flash_product_name', $product->name);
...

```

Kita akan menyimpan view untuk menampilkan pesan sukses pada file layout:

resources/views/layouts/app.blade.php

```

.....
<!-- JavaScripts -->
<script src="{{ elixir('js/all.js') }}"></script>
@if (Session::has('flash_product_name'))
    @include('catalogs._product-added', ['product_name' => Session::get('flash_p\
roduct_name')])
@endif
.....

```

Menggunakan method `Session::has()` kita mengecek apakah `flash_product_name` ada di session. Ketika variable tersebut ditemukan, kita memanggil partial view `catalogs._product-added`. Isinya sebagai berikut:

resources/views/catalogs/_product-added.blade.php

```

<script>
$(document).ready(function() {
    swal({
        title: "Sukses",
        text: "Berhasil menambahkan <strong>{{ $product_name }}</strong> ke cart!",
        type: "success",
        showCancelButton: true,
        confirmButtonColor: "#63BC81",
        confirmButtonText: "Konfirmasi pesanan",
        cancelButtonText: "Lanjutkan belanja",
        html: true
    }, function(isConfirm) {
        if (isConfirm) {
            window.location = '/cart';
        }
    })
});

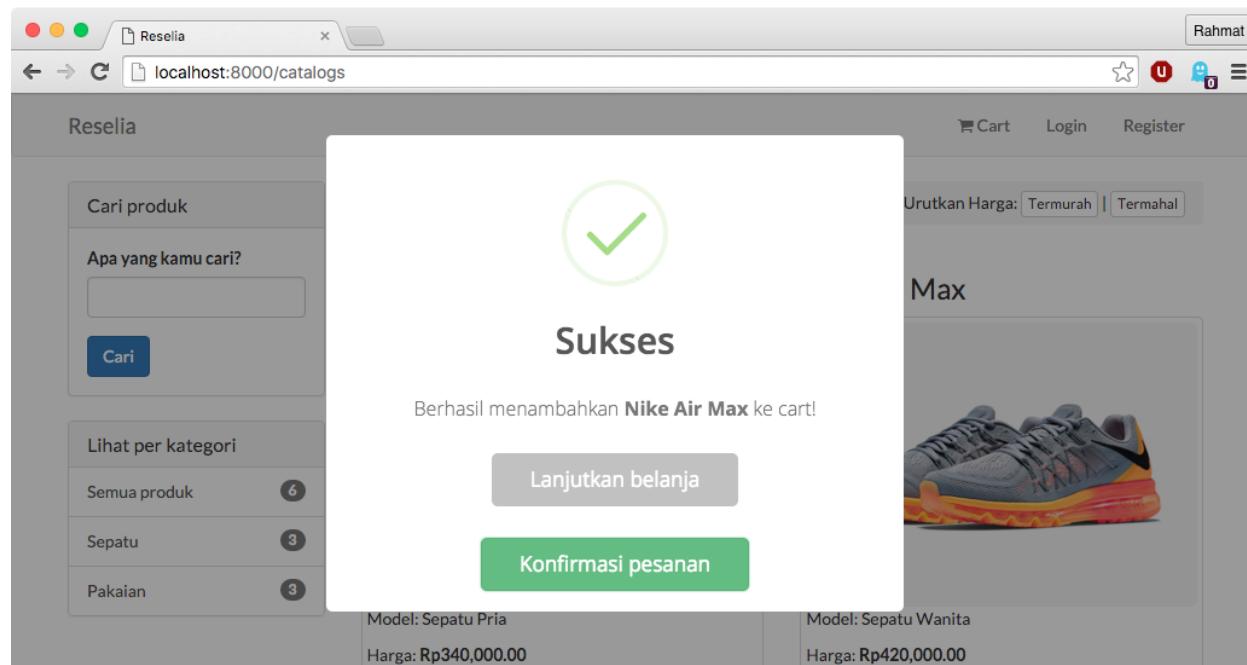
```

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

Cara kerja syntax ini adalah dengan menampilkan pesan bahwa user berhasil menambah produk yang dimaksud ke dalam cart. Akan muncul dua tombol:

- Konfirmasi pesanan: akan mengarahkan user ke halaman cart (belum kita buat).
- Lanjutkan belanja: menutup pesan sukses.

Tampilan dari pesan sukses ketika user menambah produk ke cart sebagai berikut:



Sukses menambah produk

Melihat Cart

Setelah kita berhasil menambah produk ke cart, tentunya kita ingin melihat produk apa saja yang sudah kita tambahkan. Tentunya ada beberapa operasi yang akan kita lakukan pada cart tersebut:

- Menghitung total harga dari tiap produk
- Menghitung total produk di cart
- Menampilkan detail produk
- dsb

Membangun Cart Service

Kita bisa saja langsung mengakses cookie cart dari berbagai tempat di aplikasi. Tapi, mengingat kita akan melakukan banyak operasi pada cart, tentunya cara manual akan merepotkan. Yang akan kita lakukan adalah menambahkan abstraksi pada cart ini. Misalnya, kita buat class `CartService` yang akan memiliki method untuk semua operasi diatas. Mari kita buat class ini di `app/Support/CartService.php`:

app/Support/CartService.php

```
<?php
```

```
namespace App\Support;
```

```
class CartService { }
```

Method pertama yang akan kita tambahkan pada class ini adalah `lists` yang berfungsi untuk menampilkan cart yang kita miliki dalam format:

```
[ 'product_id'=>'quantity', 'product_id'=>'quantity' ]
```

Berikut syntax untuk method ini:

app/Http/CartService.php

```
<?php
```

```
....
```

```
use App\Product;
```

```
use Illuminate\Http\Request;
```

```
class CartService {
```

```
    protected $request;
```

```
    public function __construct(Request $request)
```

```
    {
```

```
        $this->request = $request;
```

```
    }
```

```
    public function lists()
```

```
    {
```

```
        return $this->request->cookie('cart');
```

```
    }
```

```
}
```

Di method ini kita menggunakan instance dari `Illuminate\Http\Request` untuk mengambil cookie cart.

Untuk menghitung total jenis produk yang diorder kita akan membuat method dengan nama `totalProduct`. Syntaxnya akan seperti berikut:

app/Http/CartService.php

```
public function totalProduct()
{
    return count($this->lists());
}
```

Untuk memudahkan apakah cart memiliki produk, kita akan membuat method `isEmpty`:

app/Http/CartService.php

```
public function isEmpty()
{
    return $this->totalProduct() < 1;
}
```

Method ini menggunakan hasil dari method `totalProduct`. Jika hasil dari method tersebut kurang dari 1, pasti tidak ada produk di cart.

Untuk menghitung jumlah total semua produk, kita buat sebuah method dengan nama `totalQuantity`:

app/Http/CartService.php

```
public function totalQuantity()
{
    $total = 0;
    if ($this->totalProduct() > 0) {
        foreach ($this->lists() as $id => $quantity) {
            $product = Product::find($id);
            $total += $quantity;
        }
    }
    return $total;
}
```

Cara kerja method ini adalah dengan menjumlahkan quantity dari semua produk. Jika tidak ada produk di cart, kita set default ke 0.

Pada cart, kita hanya menyimpan id produk dan quantity yang diorder. Untuk itu kita akan membuat method `details` yang akan menampilkan array berisi detail dari order tersebut. Hasil akhir yang akan kita dapatkan akan seperti berikut:

```
[  
  [  
    "id" => ...,  
    "detail" => [  
      "id" => "...",  
      "name" => "...",  
      "photo" => "...",  
      "model" => "...",  
      "price" => "...",  
      "created_at" => "...",  
      "updated_at" => "...",  
      "photo_path" => "...",  
    ],  
    "quantity" => ...,  
    "subtotal" => ....  
  ],  
  [  
    "id" => ...,  
    "detail" => [  
      "id" => "...",  

```

Syntax untuk method ini akan seperti berikut:

app/Http/CartService.php

```

public function details()
{
    $result = [];
    if ($this->totalProduct() > 0) {
        foreach ($this->lists() as $id => $quantity) {
            $product = Product::find($id);
            array_push($result, [
                'id' => $id,
                'detail' => $product->toArray(),
                'quantity' => $quantity,
                'subtotal' => $product->price * $quantity
            ]);
        }
    }

    return $result;
}

```

Terlihat disini, kita menggunakan method `lists` yang telah kita buat pada tahap sebelumnya untuk mengambil semua produk di cart. Menggunakan method `foreach`, kita melakukan looping pada isian cart. Untuk mendapatkan details dari tiap produk, kita menggunakan method `$product->toArray()`. Terakhir, kita kirimkan variable `$result` yang berisi array dengan format yang kita jelaskan diatas.

Sedikit tambahan, pada array diatas terlihat kita juga mendapatkan `photo_path` pada detail produk. Karena `photo_path` merupakan custom accessor, kita harus menambahkan pada attribut `$appends` di model Produk:

app/Product.php

```

.....
protected $appends = ['photo_path'];
.....

```

Untuk menghitung total harga yang harus dibayar oleh customer kita akan membuat method bernama `totalPrice`:

app/Http/CartService.php

```
public function totalPrice()
{
    $result = 0;
    foreach ($this->details() as $order) {
        $result += $order['subtotal'];
    }
    return $result;
}
```

Di method ini, kita melakukan looping atas hasil dari method `details` dan menjumlahkan semua isian `subtotal` dari tiap detail order.

Menggunakan ViewComposer untuk mengakses CartService

Di Reselia, CartService akan kita akses dari berbagai tempat:

- Pada halaman cart
- Pada navigasi cart untuk menampilkan total produk di cart
- Pada halaman checkout

Tentunya, menggunakan cara manual kita harus menambah instance dari CartService pada setiap halaman tersebut. Dan untuk cart di navbar, nampaknya akan muncul di semua halaman. View Composer merupakan fitur yang tepat untuk memudahkan kita mengakses CartService dari berbagai view.

Mari kita buat ViewComposer yang memberikan akses ke CartService dengan nama variable `$cart` untuk semua view. Kita tambahkan baris berikut pada `app/Providers/AppServiceProvider.php` di method `boot`:

app/Providers/AppServiceProvider.php

```
public function boot()
{
    \View::composer('*', 'App\Http\ViewComposers\CartComposer');
}
```

Code untuk cart composer seperti berikut:

app/Http/ViewComposers/CartComposer.php

```
<?php

namespace App\Http\ViewComposers;

use Illuminate\View\View;
use App\Support\CartService;

class CartComposer {

    public function __construct(CartService $cart)
    {
        $this->cart = $cart;
    }

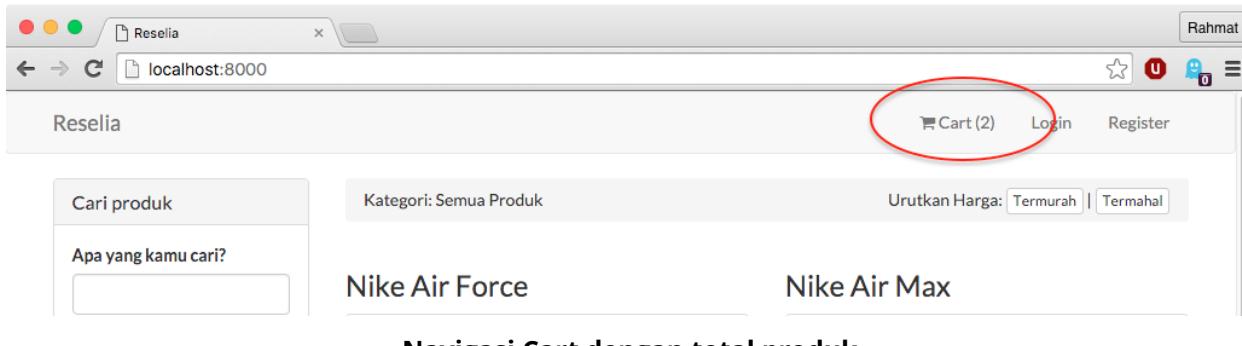
    public function compose(View $view)
    {
        $view->with('cart', $this->cart);
    }
}
```

Kita dapat mengecek apakah View Composer ini berjalan dengan menambahkan total produk pada navigasi untuk cart:

resources/views/layouts/_cart-menu-bar.blade.php

```
<li><a href="{{ url('/cart') }}"><i class="fa fa-shopping-cart"></i> Cart</a></li>
<li><a href="{{ url('/cart') }}"><i class="fa fa-shopping-cart"></i> Cart {{ $ca\rt->totalProduct() > 0 ? '(' . $cart->totalProduct() . ')' : ''}}</a></li>
```

Kini, ketika cart sudah berisi produk, tampilan navigasi untuk cart akan seperti berikut:



Membuat Halaman Cart

Halaman cart akan kita akses dari url `/cart`. Mari kita buat routenya terlebih dahulu:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::get('cart', 'CartController@show');
});
```

Di controller, kita cukup mengarahkan dengan menampilkan view:

app/Http/Controllers/CartController.php

```
public function show()
{
    return view('carts.index');
```

Mari kita buat viewnya:

resources/views/carts/index.blade.php

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

@section('content')


@if ($cart->isEmpty())
    <div class="text-center">
        <h1>: | </h1>
        <p>Cart kamu masih kosong.</p>
    </div>
@endif


```

```
<p><a href="{{ url('/catalogs') }}">Lihat semua produk <i class="fa fa-arrow-right"></i></a></p>
</div>
@else
<table class="cart table table-hover table-condensed">
<thead>
<tr>
<th style="width:50%">Produk</th>
<th style="width:10%">Harga</th>
<th style="width:8%">Jumlah</th>
<th style="width:22%" class="text-center">Subtotal</th>
<th style="width:10%"></th>
</tr>
</thead>
<tbody>
@foreach($cart->details() as $order)
<tr>
<td data-th="Produk">
<div class="row">
<div class="col-sm-2 hidden-xs"></div>
<div class="col-sm-10">
<h4 class="nomargin">{{ $order['detail']['name'] }}</h4>
</div>
</div>
</td>
<td data-th="Harga">Rp{{ number_format($order['detail']['price']) }}</td>
<td data-th="Jumlah">{{ $order['quantity'] }}</td>
<td data-th="Subtotal" class="text-center">Rp{{ number_format($order['subtotal']) }}</td>
<td>Untuk action</td>
</tr>
@endforeach
</tbody>
<tfoot>
<tr class="visible-xs">
<td class="text-center"><strong>Total Rp{{ number_format($cart->totalPrice()) }}</strong></td>
</tr>
<tr>
<td><a href="{{ url('/catalogs') }}" class="btn btn-warning"><i class="fa fa-arrow-right"></i> Lihat semua produk</a></td>
</tr>

```

```

"fa fa-angle-left"></i> Belanja lagi</a></td>
    <td colspan="2" class="hidden-xs"></td>
    <td class="hidden-xs text-center"><strong>Total Rp{{ number_format($cart->totalPrice()) }}</strong></td>
        <td><a href="{{ url('/checkout/login') }}" class="btn btn-success btn-block">Pembayaran <i class="fa fa-angle-right"></i></a></td>
    </tr>
</tfoot>
</table>
@endif

</div>
@endsection

```

Pada tampilan ini, kita menggunakan method pada CartService untuk menampilkan berbagai data:

- `isEmpty`: digunakan untuk menampilkan tampilan bahwa cart masih kosong dan link ke halaman katalog.
- `details()`: digunakan untuk menampilkan detail dari produk. Beberapa detail yang kita tampilkan disini yaitu foto produk, harga, jumlah order dan subtotal.
- `totalPrice`: digunakan untuk menampilkan total harga untuk semua produk user pada footer.
- Pada kolom terakhir dari tampilan table, kita siapkan untuk action mengubah quantity dan menghapus produk dari cart yang akan kita bahas di penjelasan selanjutnya.

Pada footer, kita juga menambahkan link untuk kembali belanja dan menuju halaman checkout (belum kita buat). Agar view ini dapat tampil dengan benar, kita perlu menambah baris CSS berikut pada `resources/assets/sass/app.scss`:

resources/assets/sass/app.scss

```

@import "node_modules/bootstrap-sass/assets/stylesheets/bootstrap";

.table>tbody>tr>td, .table>tfoot>tr>td{
    vertical-align: middle;
}

@media screen and (max-width: 600px) {
    table.cart tbody td .form-control{
        width: 20%;
        display: inline !important;
    }
}

```

```
}

.actions .btn{
    width:36%;
    margin:1.5em 0;
}

.actions .btn-info{
    float:left;
}
.actions .btn-danger{
    float:right;
}

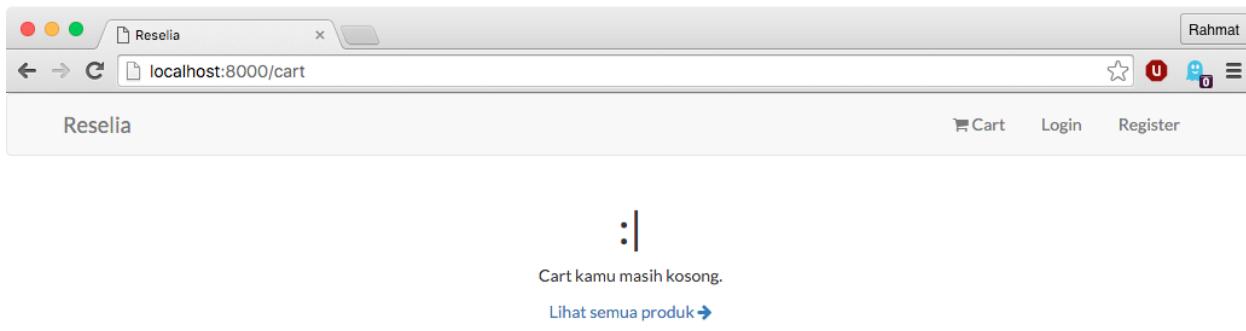
table.cart thead { display: none; }
table.cart tbody td { display: block; padding: .6rem; min-width:320px; }
table.cart tbody tr td:first-child { background: #333; color: #fff; }
table.cart tbody td:before {
    content: attr(data-th); font-weight: bold;
    display: inline-block; width: 8rem;
}
table.cart tfoot td{display:block; }
table.cart tfoot td .btn{display:block; }

.form-inline {
    display: inline;
}
```

Kita compile kembali file scss dengan

gulp

Berikut tampilan yang akan kita dapatkan ketika kita belum menambah produk ke cart (untuk mencobanya, hapus cookie `cart`):



Cart kosong

Ketika kita sudah menambahkan produk, ini tampilan halaman cart:

Produk	Harga	Jumlah	Subtotal	
Nike Air Force	Rp340,000	7	Rp2,380,000	Untuk action
Nike Air Max	Rp420,000	5	Rp2,100,000	Untuk action
Nike Aeroloft Bomber	Rp720,000	6	Rp4,320,000	Untuk action

< Belanja lagi Total Rp8,800,000 Pembayaran >

Cart terisi

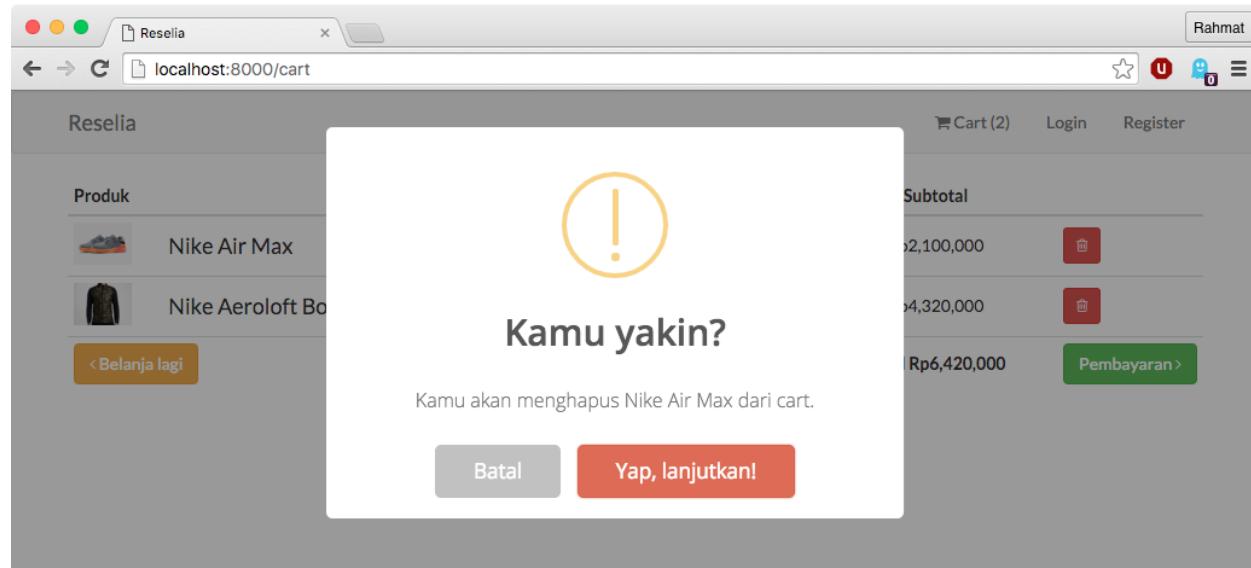
Menghapus Produk

Mari kita tambah fitur untuk menghapus produk dari cart. Pertama, kita buat dulu formnya:

resources/views/carts/index.blade.php

```
....  
<td>Untuk action</td>  
  
<td class="actions" data-th="">  
  {!! Form::open(['url' => ['cart', $order['id']], 'method'=>'delete', 'class' =\n> 'form-inline']) !!}  
  {!! Form::button('<i class="fa fa-trash-o"></i>', array('type' => 'submit', 'c\lass' => 'btn btn-danger btn-sm js-submit-confirm', 'data-confirm-message' => 'K\amu akan menghapus ' . $order['detail']['name'] . ' dari cart.')) !!}  
  {!! Form::close() !!}  
</td>  
....
```

Kita menambahkan form ini pada kolom terakhir dari table cart. Form ini kita kirim ke URL /cart/{product_id} dengan method DELETE. Pada form ini, kita juga menggunakan konfirmasi. Disini kita menggunakan custom message dengan isi "Kamu akan menghapus <nama_produk> dari cart.". Berikut tampilan dari form ini:



Konfirmasi menghapus produk dari cart

Mari kita siapkan routenya:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::delete('cart/{product_id}', 'CartController@removeProduct');
});
```

Kita mengarahkan route ini ke method `removeProduct` di `CartController`. Berikut isi method tersebut:

app/Http/Controllers/CartController.php

```
<?php
...
use App\Support\CartService;
use Flash;

class CartController extends Controller
{
    protected $cart;

    public function __construct(CartService $cart)
    {
        $this->cart = $cart;
    }

    ...

    public function removeProduct(Request $request, $product_id)
    {
        $cart = $this->cart->find($product_id);
        if (!$cart) return redirect('cart');

        Flash::success($cart['detail']['name'] . ' berhasil dihapus dari cart.');

        $cart = $request->cookie('cart', []);
        unset($cart[$product_id]);
        return redirect('cart')
            ->withCookie(cookie()->forever('cart', $cart));
    }
}
```

Berikut penjelasan untuk method diatas:

- Kita membuat instance dari CartService pada constructor dan menyimpannya pada attribut \$cart.
- Pada method `removeProduct` kita mencari apakah produk ada di cart dengan menggunakan method `find` pada CartService. Hasil dari method ini adalah `null` jika tidak ada produk dengan id tersebut di cart atau mengembalikan array detail produk di cart dengan format:

```
[  
    "id" => ...,  
    "detail" => [  
        "id" => "...",  
        "name" => "...",  
        "photo" => "...",  
        "model" => "...",  
        "price" => "...",  
        "created_at" => "...",  
        "updated_at" => "...",  
        "photo_path" => "...",  
    ],  
    "quantity" => ...,  
    "subtotal" => ...  
]
```

Detail dari method `find` akan kita jelaskan di penjelasan selanjutnya.

- Ketika kita tidak menemukan produk dengan id yang dimaksud pada cart, kita arahkan user ke halaman cart.
- Ketika data tersebut ditemukan, kita set flash message untuk feedback. Setelah itu, kita ambil nilai cart terakhir dari cookie dan menghapus isian cart untuk id produk tersebut dengan method `unset`.
- Terakhir, kita arahkan user ke halaman cart sambil mengirimkan cookie cart dengan data yang baru.

Untuk method `find` pada CartService, syntaxnya seperti berikut:

app/Support/CartService.php

```
public function find($product_id)
{
    foreach ($this->details() as $order) {
        if ($order['id'] == $product_id) return $order;
    }
    return null;
}
```

Yang kita lakukan disini adalah melakukan looping pada hasil dari method `details`. Ketika ditemukan order dengan id produk yang sesuai, kita kembalikan order tersebut. Jika tidak ditemukan, kita berikan `null`.

Ini tampilan ketika kita berhasil menghapus produk dari cart:



The screenshot shows a web browser window for a site named 'Reselia'. The address bar displays 'localhost:8000/cart'. The page content includes a success message: 'Nike Air Max berhasil dihapus dari cart.' Below this, there is a table showing the remaining items in the cart:

Produk	Harga	Jumlah	Subtotal
Nike Aeroloft Bomber	Rp720,000	6	Rp4,320,000

At the bottom, there are buttons for 'Belanja lagi' (Buy again) and 'Pembayaran' (Payment).

Berhasil menghapus produk dari cart

Mengubah Jumlah Order

Fitur terakhir yang akan kita buat untuk cart guest ini adalah mengubah jumlah order untuk tiap produk. Untuk membuat fitur ini, kita akan membuat field `jumlah` pada halaman cart menjadi sebuah form dan menambah tombol update di samping tombol delete:

resources/views/carts/index.blade.php

```
....  
<td data-th="Subtotal" class="text-center">Rp{{ number_format($order['subtotal']) }}</td>  
  
<td data-th="Jumlah">  
    {!! Form::open(['url' => ['cart', $order['id']], 'method'=>'put', 'class' => 'form-inline']) !!}  
    {!! Form::number('quantity', $order['quantity'], ['class'=>'form-control text-center']) !!}  
    </td>  
  
<td data-th="Subtotal" class="text-center">Rp{{ number_format($order['subtotal']) }}</td>  
<td class="actions" data-th="">  
  
    {!! Form::button('<i class="fa fa-refresh"></i>', array('type' => 'submit', 'class' => 'btn btn-info btn-sm')) !!}  
    {!! Form::close() !!}  
  
    {!! Form::open(['url' => ['cart', $order['id']], 'method'=>'delete', 'class' => 'form-inline']) !!}  
    {!! Form::button('<i class="fa fa-trash-o"></i>', array('type' => 'submit', 'class' => 'btn btn-danger btn-sm js-submit-confirm', 'data-confirm-message' => 'Kamu akan menghapus ' . $order['detail'][name] . ' dari cart.')) !!}  
    {!! Form::close() !!}  
</td>  
....
```

Tampilan dari halaman cart akan menjadi seperti ini:

Produk	Harga	Jumlah	Subtotal
Nike Aeroloft Bomber	Rp720,000	6	Rp4,320,000
Nike Air Zoom	Rp360,000	12	Rp4,320,000
			Total Rp8,640,000
			Pembayaran >

Form untuk ubah quantity

Pada teknik ini, setiap kali customer hendak merubah quantity untuk suatu produk, dia harus klik pada tombol update. Form yang kita gunakan akan mengarah ke '/cart/{product_id}' dengan method PUT. Pada form ini, kita hanya mengirimkan field quantity.

Mari kita siapkan routenya:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::put('cart/{product_id}', 'CartController@changeQuantity');
});
```

Pada route ini, kita arahkan url untuk merubah quantity tersebut ke method change-Quantity pada CartController:

app/Http/Controllers/CartController.php

```
public function changeQuantity(Request $request, $product_id)
{
    $this->validate($request, ['quantity' => 'required|integer|min:1']);
    $quantity = $request->get('quantity');
    $cart = $this->cart->find($product_id);
    if (!$cart) return redirect('cart');

    \Flash::success('Jumlah order untuk ' . $cart['detail']['name'] . ' berhasil\
dirubah.');

    $cart = $request->cookie('cart', []);
    $cart[$product_id] = $quantity;
    return redirect('cart')
        ->withCookie(cookie()->forever('cart', $cart));
}
```

Berikut penjelasan untuk syntax diatas:

- Kita melakukan validasi untuk quantity yang dikirm agar isiannya berupa angka dan minimal 1.
- Kita simpan quantity pada variable \$quantity.
- Kita cari produk pada cart dengan menggunakan method `find` pada CartService.
- Jika produk tidak ditemukan, kita arahkan user ke halaman cart.
- Kita set pesan sukses untuk feedback.
- Kita ambil nilai cart terbaru dari cookie dan mengubah quantitynya.
- Terakhir, kita arahkan user ke halaman cart dengan nilai cookie cart yang baru.

Berikut tampilan ketika kita berhasil merubah quantity:

Produk	Harga	Jumlah	Subtotal	
Nike Aeroloft Bomber	Rp720,000	20	Rp14,400,000	
Nike Air Zoom	Rp360,000	12	Rp4,320,000	
			Total Rp18,720,000	Pembayaran >

Berhasil merubah quantity produk

Sip. Selesai sudah pembahasan cart untuk guest. Ada satu fitur lagi, yaitu menggabungkan cart cookie ini dengan cart di database ketika user login. Ini akan kita bahas di pembahasan selanjutnya. Silahkan bikin kopi dulu..

Cart untuk Authenticated

Oke, fitur cart untuk authenticated user akan memiliki alur yang sama disisi frontend. Artinya, kita tidak akan mengubah tampilan Reselia. Yang kita lakukan adalah melakukan penyesuaian disisi backend agar melakukan tindakan yang berbeda ketika user belum dan sudah login.

Salah satu kelebihan dari fitur cart ketika user telah login adalah cart ini akan kita simpan ke database. Dengan teknik ini, dimanapun user mengakses Reselia, selama dia telah login, dia dapat mengakses cart miliknya.

Mari kita mulai dengan menyiapkan model dan migration untuk Cart:

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

Pada migration yang dihasilkan isi method `up` dengan syntax berikut:

database/migrations/xxxx_xx_xx_xxxxxxx_create_carts_table.php

```
public function up()
{
    Schema::create('carts', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('user_id')->unsigned();
        $table->integer('product_id')->unsigned();
        $table->integer('quantity');

        $table->foreign('product_id')->references('id')->on('products');
        $table->foreign('user_id')->references('id')->on('users');
        $table->timestamps();
    });
}
```

Pada table carts, selain field id kita menambahkan field:

- user_id: untuk mencatat user yang memiliki record cart.
- product_id: menyimpan id produk
- quantity: jumlah order untuk produk yang dimaksud

Kita juga menambahkan index foreign key dari field product_id ke field id di table products dan user_id ke field id di table users.

Berdasarkan struktur diatas, setiap produk yang berada di cart user akan memiliki 1 record di table. Mari kita isi syntax untuk modelnya:

app/Cart.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Cart extends Model
{
    protected $fillable = ['user_id', 'product_id', 'quantity'];

    public function user()
    {
```

```
    return $this->belongsTo('App\User');  
}  
  
public function product()  
{  
    return $this->belongsTo('App\Product');  
}  
}
```

Pada model ini, kita mengaktifkan mass assignment dan menambahkan relasi ke model user dan produk.

Karena kita sekarang memiliki foreign key dari cart ke product, kita harus membuat perubahan pada event penghapusan product agar menghapus cart yang memiliki produk tersebut. Pertama, kita buat dulu relasi ke cart:

app/Product.php

```
public function carts()  
{  
    return $this->hasMany('App\Cart');  
}
```

Kemudian kita tambahkan baris berikut pada handle untuk event `deleting`:

app/Product.php

```
public static function boot()  
{  
    parent::boot();  
  
    static::deleting(function($model) {  
        // remove relations to category  
        $model->categories()->detach();  
        // remove relation to cart  
        $model->carts()->detach();  
    });  
}
```

Terakhir, jalankan migration dengan:

```
php artisan migrate
```

Pastikan di database muncul table carts dengan struktur yang tepat:

The screenshot shows the MySQL Workbench interface with the 'reselia' database selected. The 'Structure' tab is active, displaying the 'carts' table. The table has the following columns:

Field	Type	Len...	Unsigned	Zerofill	Binary	Allow Null	Key	Defa...	Extra	Encod...	Collation	Co...
id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI	auto.i...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
user_id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	M...	None	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
product_id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	M...	None	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
quantity	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		None	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
created_at	TIMEST...		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	None	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updated_at	TIMEST...		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	None	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Below the table, the 'INDEXES' section shows three indexes:

Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Comment
0	PRIMARY	1	id	A	0	NUL	NUL	
1	carts_pro...	1	product_id	A	0	NUL	NUL	
1	carts_use...	1	user_id	A	0	NUL	NUL	

Table carts

Sip.

Menambah Produk

Untuk menambah produk, kita cukup mengubah method `addProduct` pada `CartController`. Yang kita lakukan adalah dengan melakukan pengecekan apakah user sedang login menggunakan `Auth::check()` dan memberikan logic yang sesuai. Berikut syntaxnya:

app/Http/Controllers/CartController.php

```
....  
use Auth;  
use App\Cart;  
  
class CartController extends Controller  
{  
    ....  
    public function addProduct(Request $request)  
    {  
        ....
```

```

$cart = $request->cookie('cart', []);
if (array_key_exists($product->id, $cart)) {
    $quantity += $cart[$product->id];
}
$cart[$product->id] = $quantity;
return redirect('catalogs')
    ->withCookie(cookie()->forever('cart', $cart));

if (Auth::check()) {
    $cart = Cart::firstOrCreate([
        'product_id' => $product->id,
        'user_id' => $request->user()->id
    ]);
    $cart->quantity += $quantity;
    $cart->save();
    return redirect('catalogs');
} else {
    $cart = $request->cookie('cart', []);
    if (array_key_exists($product->id, $cart)) {
        $quantity += $cart[$product->id];
    }
    $cart[$product->id] = $quantity;
    return redirect('catalogs')
        ->withCookie(cookie()->forever('cart', $cart));
}
}

```

Terlihat disini, didalam kondisi `Auth::check()` kita membuat syntax untuk proses pembuatan cart.

- Kita menggunakan method `firstOrCreate` sebagai pengaman jika sudah ada record cart dengan `product_id` dan `user_id` yang kita kirimkan. Jika belum ada, maka record tersebut akan dibuat.
- Selanjutnya, kita tingkatkan `quantity` dari record tersebut sesuai `quantity` yang dikirim user.
- Kita simpan recordnya.
- Dan terakhir kita arahkan user ke halaman katalog.

Logic cart untuk guest kita pindahkan ke blok `else`. Isinya tetap sama.

Untuk mencoba fitur ini, pastikan sudah login dengan user yang memiliki role `customer`. Cobalah menambah produk ke cart, setelah berhasil menambah produk ke cart pastikan ada record baru di table `carts`:

	id	user_id	product_id	quantity	created_at	updated_at
carts	1	2	1	10	2016-03-01 14:38:41	2016-03-01 14:38:41

Berhasil menyimpan cart ke database

Melihat Cart

Untuk dapat mengakses cart yang berada di table `carts` pada saat user telah login, kita cukup merubah satu method di `CartService` yaitu method `lists`:

app/Support/CartService.php

```

use Auth;
use App\Cart;

class CartService {

    ...

    public function lists()
    {
        if (Auth::check()) {
            return Cart::where('user_id', Auth::user()->id)
                ->lists('quantity', 'product_id');
        } else {
            return $this->request->cookie('cart');
        }
    }

}

```

Menggunakan `Auth::check()` kita mengecek apakah user sudah login. Jika sudah login kita ambil semua record cart untuk user tersebut. Menggunakan method `lists('quantity', 'product_id')`, kita akan mendapatkan array seperti berikut:

```
[ 'product_id'=>'quantity', 'product_id'=>'quantity' ]
```

Yap, formatnya akan sama persis dengan array yang kita dapatkan dari cookie. Dengan teknik ini, kita tidak perlu merubah method yang lain di CartService.

Untuk mengetesnya, cobalah menambah produk ke cart dan mengunjungi halaman cart. Pastikan produk yang berada di halaman tersebut sesuai dengan data yang kita miliki di table carts untuk user yang sedang login.

Menghapus Produk

Fitur hapus produk di cart authenticated harus menambahkan perubahan berikut pada method removeProduct di CartController:

app/Http/Controllers/CartController.php

```
public function removeProduct(Request $request, $product_id)
{
    ...
    $cart = $request->cookie('cart', []);
    unset($cart[$product_id]);
    return redirect('cart')
        ->withCookie(cookie()->forever('cart', $cart));

    if (Auth::check()) {
        $cart = Cart::firstOrCreate([
            'product_id' => $product_id,
            'user_id' => $request->user()->id
        ]);
        $cart->delete();
        return redirect('cart');
    } else {
        $cart = $request->cookie('cart', []);
        unset($cart[$product_id]);
        return redirect('cart')
            ->withCookie(cookie()->forever('cart', $cart));
    }
}
```

Seperti pada saat menambah produk ke cart, disini kita juga mencari atau membuat record di table cart. Setelah ditemukan, kita menghapusnya dengan \$cart->delete(). Terakhir, kita arahkan kembali user ke halaman cart.

Cobalah fitur ini, pastikan bisa menghapus produk pada cart user yang telah login.

Mengubah Jumlah Order

Untuk mengubah jumlah order, kita akan ubah method `changeQuantity` di `CartController`:

app/Http/Controllers/CartController.php

```
public function changeQuantity(Request $request, $product_id)
{
    ....
    $cart = $request->cookie('cart', []);
    $cart[$product_id] = $quantity;
    return redirect('cart')
        ->withCookie(cookie()->forever('cart', $cart));

    if (Auth::check()) {
        $cart = Cart::firstOrCreate(['user_id'=>$request->user()->id, 'product_i\
d'=>$product_id]);
        $cart->quantity = $quantity;
        $cart->save();
        return redirect('cart');
    } else {
        $cart = $request->cookie('cart', []);
        $cart[$product_id] = $quantity;
        return redirect('cart')
            ->withCookie(cookie()->forever('cart', $cart));
    }
}
```

Saya rasa cukup jelas, pada cart yang ditemukan kita mengubah quantity, save dan mengarahkan user ke halaman cart. Silahkan dicoba, pastikan quantity nya berubah.

Menggabungkan Cart dari Cookie ketika Login

Ada kalanya user sudah memiliki akun pada Reselia, namun dia membuka Reselia tanpa login terlebih dahulu. Tentunya, ketika dia menambahkan produk ke cart, yang aktif adalah cart di cookie. Yang akan kita lakukan disini adalah menggabungkan data cart yang berada di cookie dengan data cart di database. Ada beberapa hal yang harus kita lakukan:

- Jika produk pada cart cookie belum ada di cart database, kita harus membuat yang baru.

- Jika produk pada cart cookie sudah ada di database, kita akan menggunakan cart di database. Artinya, quantity yang kita gunakan adalah quantity yang tersimpan di database.
- Setelah proses penggabungan selesai, kita harus menghapus cart cookie dari browser user.

Untuk memudahkan proses penggabungan cart ini, kita akan melakukan abstraksi di CartService. Caranya, kita akan membuat method `merge` yang akan menjalankan semua logic diatas dan menghasilkan cookie baru menggunakan `Cookie::forget()` untuk menghapus cart cookie. Berikut syntaxnya:

app/Support/CartService.php

```
use Cookie;

class CartService {
    ...
    public function merge()
    {
        $cart_cookie = $this->request->cookie('cart', []);
        foreach ($cart_cookie as $product_id => $quantity) {
            $cart = Cart::firstOrCreate([
                'user_id' => $this->request->user()->id,
                'product_id' => $product_id]);
            $cart->quantity = $cart->quantity > 0 ? $cart->quantity : $quantity;
            $cart->save();
        }

        return Cookie::forget('cart');
    }
    ...
}
```

Agar method diatas dapat berjalan setiap kali user login, kita akan menambahkan pada method `handle` di `app/Http/Middleware/Authenticate.php`:

app/Http/Middleware/Authenticate.php

```

.....
use App\Support\CartService;

class Authenticate
{
    protected $cart;

    public function __construct(CartService $cart)
    {
        $this->cart = $cart;
    }
    .....
    public function handle($request, Closure $next, $guard = null)
    {
        if (Auth::guard($guard)->guest()) {
            if ($request->ajax() || $request->wantsJson()) {
                return response('Unauthorized.', 401);
            } else {
                return redirect()->guest('login');
            }
        }

        if ($request->user()->can('customer-access')) {
            // merge cart from cookie to db
            // send response while remove cart from cookie
            $cookie = $this->cart->merge();
            return $next($request)->withCookie($cookie);
        }
    }
    return $next($request);
}

```

Pada middleware ini, kita meng-*inject* CartService pada method constructor. Disini kita mengecek apakah user yang login seorang customer dengan menggunakan gate:

```
$request->user()->can('customer-access')
```

Jika dia customer, kita gabungkan cart dengan method `merge` yang telah kita buat. Terakhir, kita hapus cookie dengan memanggil `withCookie` dengan hasil dari method

merge (yaitu `Cookie::forget()`). Jika dia bukan customer, kita handle request seperti biasa.

Cobalah fitur ini dengan menambahkan produk ke cart sebelum login. Pastikan ketika login sebagai customer, semua produk tersebut ada pada cart. Kemudian cobalah logout, pastikan cart menjadi kosong.

Coba juga membuat cart sebagai guest dan login sebagai admin. Kemudian coba logout kembali. Pastikan cart cookie masih ada.

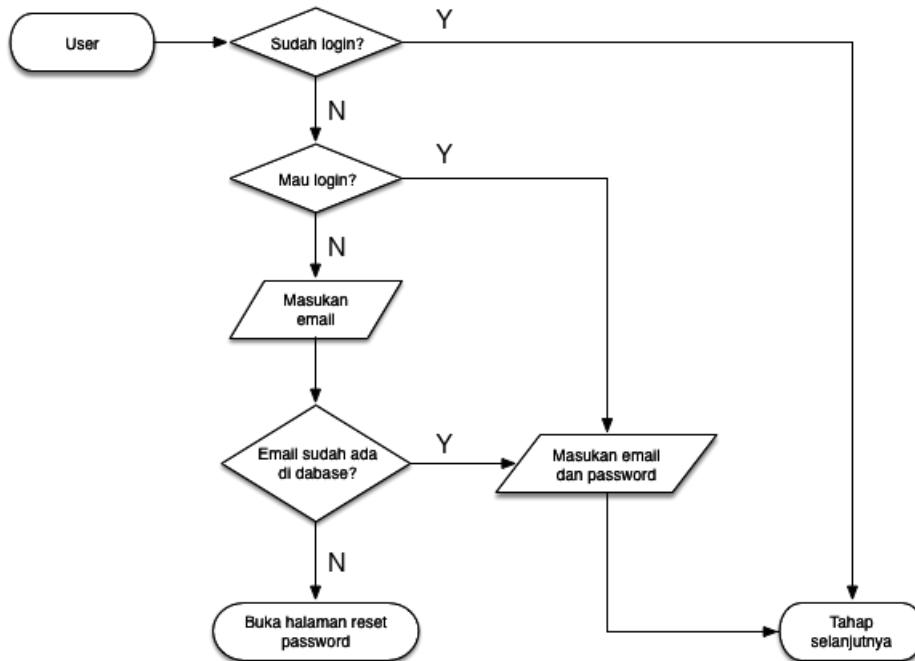
Checkout

Di Reselia, kita akan menggunakan alur checkout yang biasa digunakan situs-situs e-commerce di Indonesia. Proses checkout akan kita bagi menjadi 4 tahap besar:

1. Menentukan identitas user

Pada tahap ini, kita mengidentifikasi user menggunakan alamat email. Jika user sudah login, tentunya tahap ini akan otomatis kita lewati. Jika user belum login dan belum terdaftar, dia hanya perlu memasukan email tanpa perlu membuat akun. Jika email user sudah ada di database dan memiliki password, kita akan mengembalikan user ke halaman tersebut untuk login. Jika tidak memiliki password, kita arahkan ke halaman reset password.

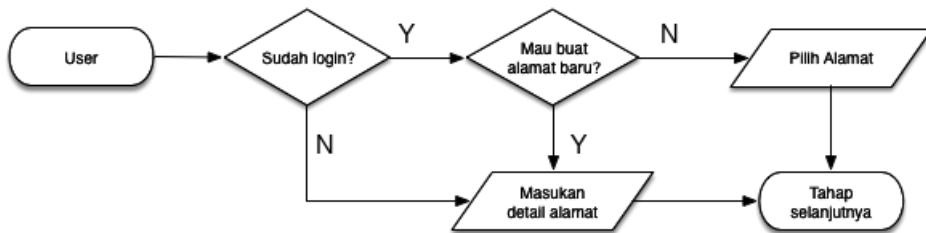
Diagram untuk tahap ini akan seperti berikut:

**Checkout Step 1**

- Menentukan alamat** Sebagaimana situs e-commerce lain di Indonesia, kita akan menggunakan fitur alamat ini untuk menentukan ongkos kirim. Untuk menghitungnya kita akan menggunakan tarif JNE yang kita dapatkan dari [rajaongkir.com](#). Tentunya, daerah yang bisa kita tampilkan pada alamat ini, hanya daerah yang bisa dijangkau oleh JNE.

Ketika user belum login, tahap ini hanya menampilkan form untuk menambah alamat baru. Sementara ketika user telah login, tahap ini akan menampilkan alamat yang sudah tersimpan di database dan form untuk membuat form alamat baru.

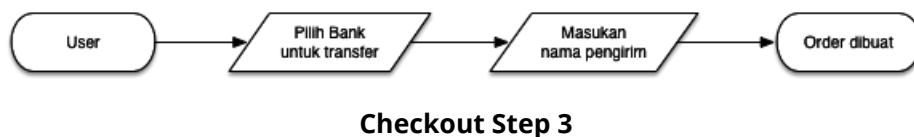
Diagram untuk tahap ini akan seperti berikut:

**Checkout Step 2**

- Menentukan metode pembayaran** Tahap ini akan sama untuk user yang belum maupun sudah login. Pada tahap ini kita akan meminta user untuk

memilih Bank untuk melakukan transfer dan nama pengirim yang akan digunakan.

Diagram untuk tahap ini akan seperti berikut:



1. **Order dibuat** Tahap terakhir ini, kita menampilkan bahwa user berhasil membuat order. Kita juga menampilkan detail lain seperti nomor order, nama bank dan nomor rekening untuk transfer dan link untuk kembali melihat katalog.

Pada tahap 1 sampai 3, kita akan menampilkan sidebar berisi detail cart yang dimiliki oleh user. Khusus untuk tahap 3, kita juga akan menambahkan ongkos kirim pada total harga yang harus user transfer.

Semua data yang kita terima pada setiap tahap akan kita simpan pada session. Ketika kita sampai di tahap terakhir, barulah kita membuat semua record yang dibutuhkan untuk membuat order.

Untuk routing, kita akan menggunakan route yang sama untuk user yang belum dan sudah login. Mari kita siapkan controller untuk semua tahapan checkout:

```
php artisan make:controller CheckoutController
```

Checkout untuk Guest

Seperti proses pengembangan cart, mari kita mulai pembuatan fitur checkout untuk user yang belum login (guest).

Tahap 1: Login

Pada tahap ini, kita akan memberikan user pilihan untuk checkout sebagai customer baru atau customer yang sudah pernah login. Tentunya, untuk sekarang yang akan kita buat adalah logic ketika user memilih checkout sebagai user baru. Ketika user memilih opsi ini, ada tiga hal yang mungkin terjadi:

- Email tidak ditemukan di database. Kita simpan email tersebut pada session dan mengarahkan user ke tahap selanjutnya.

- Email ditemukan dan memiliki password. Kita kembalikan user ke halaman tahap login, memberikan pesan error dan memindahkan opsi sebagai user yang sudah pernah login.
- Email ditemukan tapi tidak memiliki password.

Mari kita buat route `checkout/login` untuk mengakses tahap ini:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::get('checkout/login', 'CheckoutController@login');
    Route::post('checkout/login', 'CheckoutController@postLogin');
});
```

Disini kita membuat dua method untuk menampilkan halaman dan menerima form dari halaman tersebut. Kita buat methodnya:

app/Http/Controllers/CheckoutController.php

```
public function login()
{
    return view('checkout.login');
}

public function postLogin() { }
```

Pada method ini `login` kita menampilkan view `checkout.login`. Untuk method `postLogin` akan kita isi syntaxnya pada pembahasan selanjutnya.

Dalam membangun halaman checkout, kita akan banyak menggunakan elemen yang akan muncul pada tiap halaman. Untuk itu, kita akan menggunakan beberapa partial view. Berikut penjelasan partial view yang akan kita gunakan:

- `checkout._step`: digunakan untuk menampilkan tahapan checkout yang sedang dilakukan oleh user. Tampilan pada tahapan ini, akan berubah sesuai tahapan checkout yang sedang aktif.
- `checkout._login-form`: digunakan untuk menampilkan form untuk memilih metode checkout (customer baru atau login).
- `checkout._cart-panel`: digunakan untuk menampilkan data cart user.

Mari kita tambahkan setiap partial view diatas satu-persatu. Kita mulai dari `checkout._step`. Syntax untuk halaman login akan seperti ini:

resources/views/checkout/login.blade.php

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

@section('content')


@include('checkout._step')


Login atau Checkout tanpa mendaftar


@endsection
```

Ini tampilan partial viewnya:

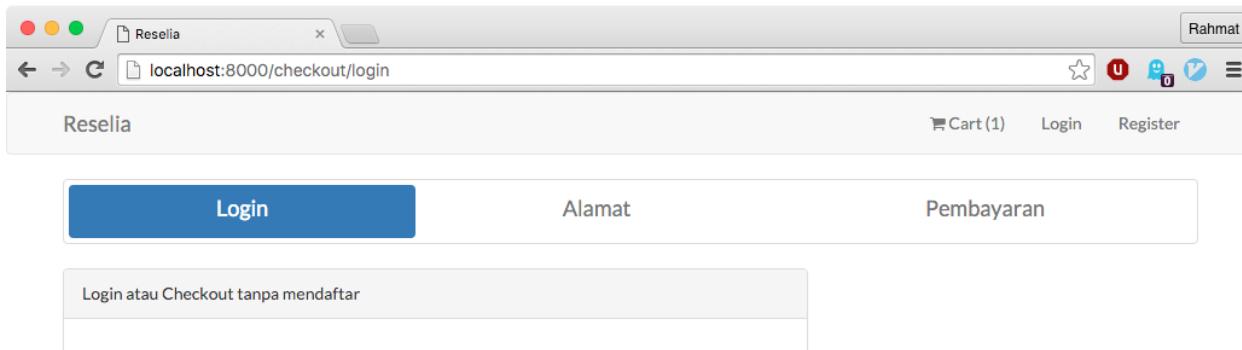
resources/views/checkout/_step.blade.php

```
<div class="row">
<div class="col-xs-12">
    <ul class="nav nav-pills nav-justified thumbnail setup-panel">
        <li class="{{ Request::is('checkout/login') ? 'active' : 'disabled' }}><a>
            <h4 class="list-group-item-heading">Login</h4>
        </a></li>
        <li class="{{ Request::is('checkout/address') ? 'active' : 'disabled' }}><\a>
            <h4 class="list-group-item-heading">Alamat</h4>
        </a></li>
        <li class="{{ Request::is('checkout/payment') ? 'active' : 'disabled' }}><\a>
            <h4 class="list-group-item-heading">Pembayaran</h4>
        </a></li>
    </ul>
</div>
</div>
```

Pada partial view ini, kita menggunakan `ul` dengan class `nav` untuk menampilkan tahapan checkout. Kita juga menggunakan `Request::is()` untuk mengecek url yang sedang aktif dan menambahkan class `active` jika sesuai. URL yang kita gunakan sebagai berikut:

- `checkout/login`: tahap login
- `checkout/address`: tahap pembuatan / pemilihan alamat
- `checkout/payment`: tahap pembayaran

Cobalah kunjungi `/checkout/login`, ini yang akan kita dapatkan:



Tahap login

Untuk saat ini, kita belum melakukan proteksi terhadap setiap tahapan checkout. Untuk menghindari error selama development, pastikan cart memiliki isi. Proteksi untuk tahapan checkout akan kita lakukan dengan menggunakan middleware pada pembahasan selanjutnya.

Selanjutnya kita akan menambahkan `checkout._login-form` seperti berikut:

resources/views/checkout/login.blade.php

```
....  
<div class="panel-heading">Login atau Checkout tanpa mendaftar</div>  
<div class="panel-body">  
    @include('checkout._login-form')  
</div>  
....
```

Syntax untuk form ini sebagai berikut:

resources/views/checkout/_login-form.blade.php

```
{!! Form::open(['url' => '/checkout/login', 'method'=>'post', 'class' => 'form-h\orizonta!']) !!}  
  
<div class="form-group {!! $errors->has('email') ? 'has-error' : '' !!}">  
    {!! Form::label('email', 'Email', ['class' => 'col-md-4 control-label']) !!}\br/>!}<br>  
    <div class="col-md-6">  
        {!! Form::email('email', null, ['class'=>'form-control']) !!}  
        {!! $errors->first('email', '<p class="help-block">:message</p>') !!}  
    </div>  
</div>  
  
<div class="form-group {!! $errors->has('is_guest') ? 'has-error' : '' !!}">  
    <div class="col-md-6 col-md-offset-4 radio">  
        <p><label>{{ Form::radio('is_guest', 1, true) }} Saya adalah pelanggan baru</label></p>  
        <p><label>{{ Form::radio('is_guest', 0) }} Saya adalah pelanggan tetap</label></p>  
        {!! $errors->first('is_guest', '<p class="help-block">:message</p>') !!}  
    </div>  
</div>  
  
<div class="form-group {!! $errors->has('checkout_password') ? 'has-error' : '' !!}">  
    {!! Form::label('checkout_password', 'Password', ['class' => 'col-md-4 control-label']) !!}<br>  
    <div class="col-md-6">  
        {!! Form::password('checkout_password', ['class'=>'form-control']) !!}  
        {!! $errors->first('checkout_password', '<p class="help-block">:message</p>') !!}
```

```

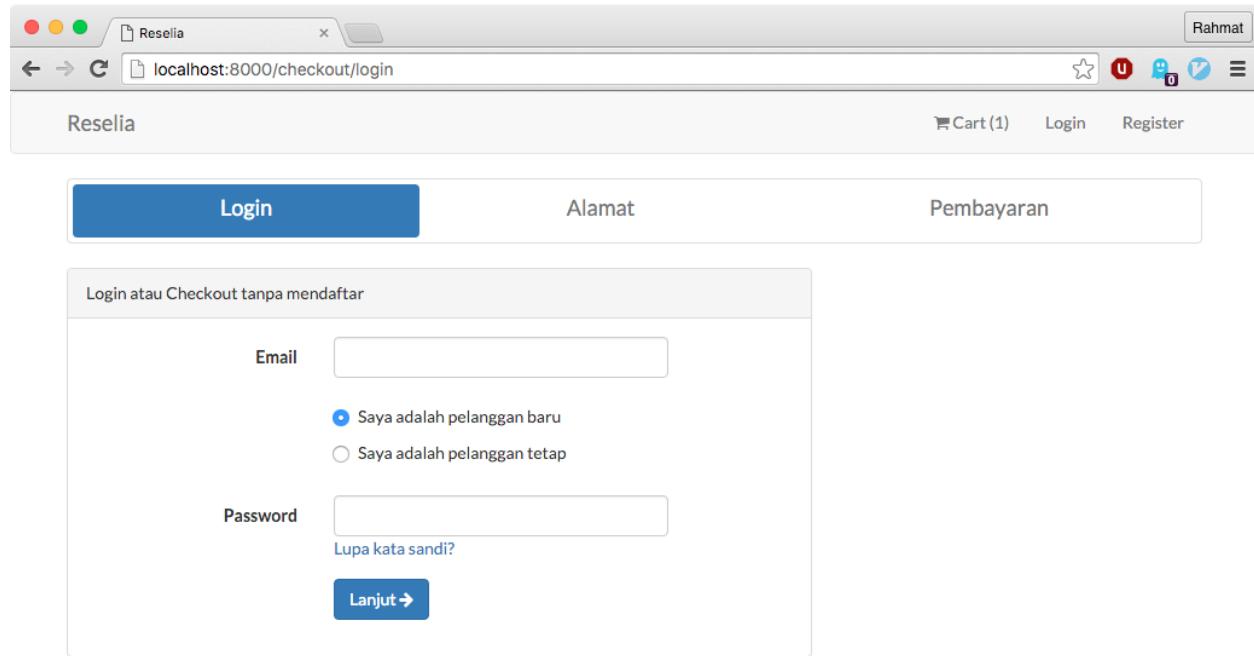
<a href="{{ url('/password/reset') }}">Lupa kata sandi?</a>
</div>
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::button('Lanjut <i class="fa fa-arrow-right"></i>', array('
type' => 'submit', 'class' => 'btn btn-primary')) !!}
    </div>
</div>
{!! Form::close() !!}

```

Pada form ini kita menambahkan field email, pilihan untuk pelanggan baru atau tetap dan isian untuk password pelanggan tetap. Untuk membuat pemilihan pelanggan baru dan tetap kita menggunakan radion button dengan nama `is_guest`. Kita juga menambahkan link ke halaman reset password jika user lupa password.

Sekarang, tampilan halaman ini akan seperti berikut:



Form pada tahap login

Akan lebih baik jika isian password hanya dapat diisi ketika user memilih sebagai pelanggan tetap. Untuk melakukan itu, mari kita tambah sedikit javascript:

resources/assets/js/app.js

```
$(document).ready(function () {
    ...
    // checkout login form
    if ($('input[name=checkout_password]').length > 0 && $('input[name=is_guest]')\n.length > 0 && $('input[name=is_guest]:checked').val() > 0) {
        $('input[name=checkout_password]').prop('disabled', true)
    }
})
```

Syntax ini akan mengecek apakah pada form terdapat input dengan isian `name` berupa `checkout_password`, input dengan isian `name` berupa `is_guest` dengan isian bukan 0. Ketika semua kondisi itu terpenuhi, kita disable input password.

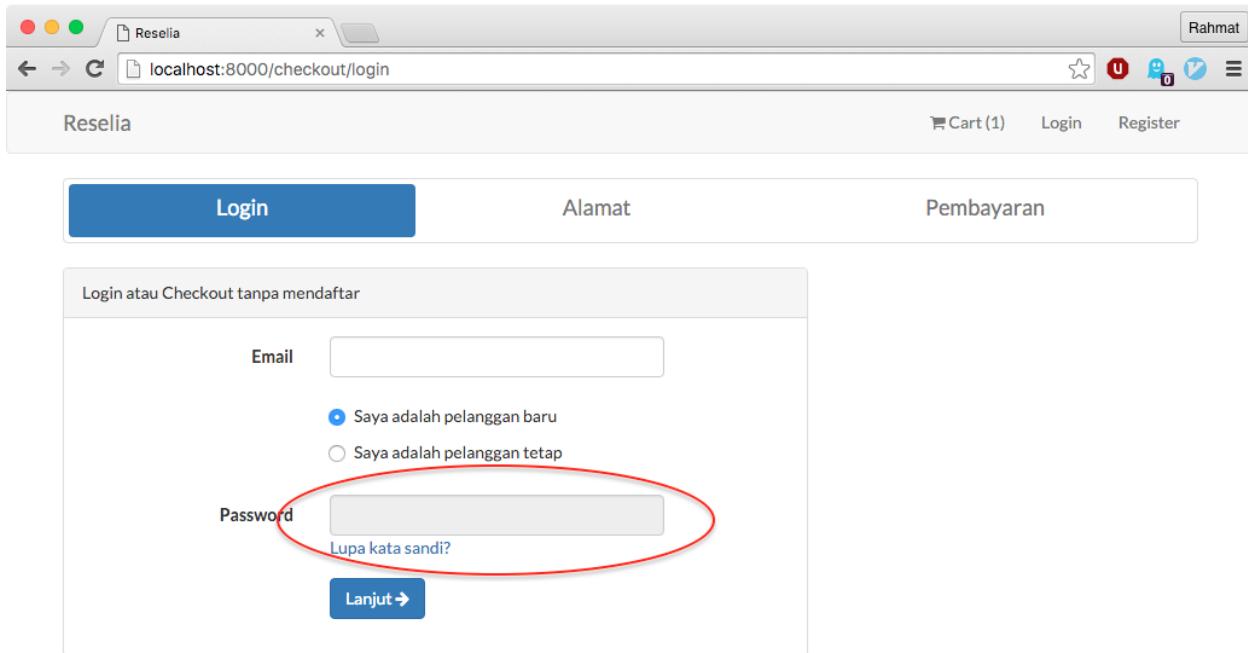
Setiap kali isian `is_guest` di klik, kita cek kembali nilainya. Jika isiannya bukan 0, maka kita disable. Jika 0, maka kita enable isian password.

Pastikan untuk menjalankan

gulp

setelah melakukan perubahan ini.

Hasil yang akan kita dapatkan ketika kita aktifkan `saya adalah pelanggan baru` adalah



Disable isian password

Jika kita klik pada "Saya adalah pelanggan tetap", tentunya isian password bisa diisi kembali.

Terakhir, kita tambahkan panel untuk melihat cart:

resources/views/checkout/login.blade.php

```
....  
<div class="col-xs-4">  
    @include('checkout._cart-panel')  
</div>  
....
```

Syntax untuk partial view untuk panel cart ini sebagai berikut:

resources/views/checkout/_cart-panel.blade.php

```
<div class="panel panel-default">
    <div class="panel-heading">Cart</div>
    <div class="panel-body">
        <table class="table table-condensed">
            <thead>
                <tr>
                    <th style="width:50%">Produk</th>
                    <th style="width:20%">Jumlah</th>
                    <th style="width:30%">Harga</th>
                </tr>
            </thead>
            <tbody>
                @foreach($cart->details() as $order)
                <tr>
                    <td data-th="Produk">{{ $order['detail']['name'] }}</td>
                    <td data-th="Jumlah" class="text-center">{{ $order['quantity'] }}</td>
                    <td data-th="Harga" class="text-right">{{ number_format($order['detail']\['price']) }}</td>
                </tr>
                <tr>
                    <td data-th="Subtotal">Subtotal</td>
                    <td data-th="Subtotal" class="text-right" colspan="2">Rp{{ number_form\at($order['subtotal']) }}</td>
                </tr>
                @endforeach
            </tbody>
            <tfoot>
                <tr>
                    <td><strong>Total</strong></td>
                    <td class="text-right" colspan="2"><strong>Rp{{ number_format($cart->t\otalPrice()) }}</strong></td>
                </tr>
            </tfoot>
        </table>
    </div>
</div>
```

Yang kita lakukan di panel ini adalah menampilkan kolom produk, jumlah order dan harga satuan produk tersebut. Di setiap produk, kita juga menampilkan subtotal. Pada footer, kita tampilkan total pembayaran. Untuk mendapatkan semua data

ini, kita menggunakan variable `$cart` yang telah kita atur sebelumnya dengan View Composer.

Tampilan yang kita dapatkan akan seperti ini:

Produk	Jumlah	Harga
Nike Air Zoom	20	360,000
Subtotal		Rp7,200,000
Nike Aeroloft Bomber	30	720,000
Subtotal		Rp21,600,000
Total		Rp28,800,000

Panel cart

Selanjutnya, mari kita buat logic untuk menghandle form yang dikirimkan. Proses validasi untuk form ini akan kita handle menggunakan form request mari kita buat dengan nama `CheckoutLoginRequest`:

```
php artisan make:request CheckoutLoginRequest
```

Ini syntax yang akan kita buat pada form request tersebut:

app/Http/Requests/CheckoutLoginRequest.php

```
<?php

namespace App\Http\Requests;

use App\Http\Requests\Request;

class CheckoutLoginRequest extends Request
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'email'          => 'required|email',
            'is_guest'       => 'required|in:0,1',
            'checkout_password' => 'required_if:is_guest,0',
        ];
    }

    public function messages()
    {
        return [
            'email.required'           => 'Email harus diisi',
            'checkout_password.required_if' => 'Password harus diisi jika Anda adalah pelanggan tetap',
        ];
    }
}
```

Mari kita bahas rule yang kita lakukan pada method-method diatas.

Pada method `authorize`, kita langsung memberikan nilai `true`. Ini kita lakukan karena kita akan menggunakan middleware untuk melindungi semua tahapan ini.

Pada method `rules`, kita menggunakan beberapa aturan validasi berikut:

- field `email` wajib diisi.
- field `is_guest` wajib diisi dengan isian 1 atau 0.

- field `checkout_password` wajib diisi jika isian `is_guest` adalah 0.

Pada method `messages`, saya rasa cukup jelas. Disini kita memberikan pesan validasi custom untuk field `email` dan `checkout_password`.

Kita lanjut ke controller, method `postLogin` yang sudah kita siapkan sebelumnya. Berikut codenya:

app/Http/Controllers/CheckoutController.php

```
....  
use App\Http\Requests\CheckoutLoginRequest;  
  
class CheckoutController extends Controller  
{  
    ....  
    public function postLogin(CheckoutLoginRequest $request)  
    {  
        $email = $request->get('email');  
        $password = $request->get('checkout_password');  
        $is_guest = $request->get('is_guest') > 0;  
  
        if ($is_guest) {  
            return $this->guestCheckout($email);  
        }  
  
        return $this->authenticatedCheckout($email, $password);  
    }  
}
```

Terlihat disini, kita membuat variable `$is_guest` berisi nilai boolean. Field ini akan berisi `true` jika user memilih "Saya adalah pelanggan baru". Berdasarkan nilai variable ini, kita akan mengarahkan logic ke method `guestCheckout` dengan parameter `email` untuk memprosesnya. Terakhir, kita arahkan ke method `authenticatedCheckout` dengan parameter `email` dan `password`.

Mari kita bahas kedua method yang kita panggil pada method `postLogin`.

Method `authenticatedCheckout` untuk saat ini akan kita isi dengan text pengingat bahwa logicnya belum kita buat:

app/Http/Controllers/CheckoutController.php

```
protected function authenticatedCheckout($email, $password)
{
    return 'logic untuk authenticated checkout belum dibuat';
}
```

Method `guestCheckout` adalah method yang akan kita kerjakan pada bagian ini. Untuk memudahkan pembahasan, mari kita bahas method ini secara bertahap. Silahkan buat dulu syntax seperti ini:

app/Http/Controllers/CheckoutController.php

```
...
use Illuminate\Support\MessageBag;
use App\User;

class CheckoutController extends Controller
{
    ...
    protected function guestCheckout($email)
    {
        // check if user exist, if so, ask login
        if ($user = User::where('email', $email)->first()) {
            if ($user->hasPassword()) {
                // (A) Logic ketika email ada di DB dengan password
            }
            // (B) Logic ketika email di DB tanpa password
        }
        // (C) Logic ketika email tidak ada di DB
    }
    ...
}
```

Dapat kita lihat kita melakukan pengecekan ke database terhadap parameter `$email`.

```
if ($user = User::where('email', $email)->first()) {
```

Selanjutnya, kita cek apakah email tersebut memiliki password. Dengan memanggil method `hasPassword` pada model User.

```
if ($user->hasPassword()) {
```

Kita harus membuat method ini pada model User:

app/User.php

```
public function hasPassword()
{
    return $this->password != '';
}
```

Terlihat disini, ada 3 hal yang mungkin terjadi ketika kita memanggil method `guestCheckout`. Kita mulai dari logic untuk kondisi **(A)**:

app/Http/Controllers/CheckoutController.php

```
protected function guestCheckout($email)
{
    // check if user exist, if so, ask login
    if ($user = User::where('email', $email)->first()) {
        if ($user->hasPassword()) {
            // (A) Logic ketika email ada di DB dengan password
            $errors = new MessageBag();
            $errors->add('checkout_password', 'Alamat email "' . $email . '" sudah\
terdaftar, silahkan masukan password.');
            // redirect and change is_guest value
            return redirect('checkout/login')->withErrors($errors)
                ->withInput(compact('email') + ['is_guest' => 0]);
        }
        // (B) Logic ketika email di DB tanpa password
    }
    // (C) Logic ketika email tidak ada di DB
}
```

Response yang akan kita berikan pada user ketika dia menggunakan email yang telah memiliki password dan memilih "Saya adalah pelanggan baru" adalah meminta dia untuk memasukan password pada tahap pertama.

Ini kita lakukan dengan teknik berikut:

- Kita siapkan pesan error untuk field password. Disini kita membuat instance dari `Illuminate\Support\MessageBag` secara manual. Menggunakan method `add()` kita menambahkan pesan error untuk field `checkout_password`.

- Kita arahkan user kembali ke halaman `checkout/login`. Menggunakan method `withErrors()` kita tambahkan `MessageBag` yang telah kita buat. Kita juga menggunakan method `withInput` agar isian email tetap terisi dan isian `is_guest` menjadi 0. Sehingga yang terpilih berubah menjadi "Saya adalah pelanggan tetap".

Untuk mengetes logic ini, cobalah mengunjungi halaman `checkout/login` sebagai guest. Pada isian email isi dengan akun customer yang sudah ada (misalnya `customer@gmail.com`). Pilih "Saya adalah pelanggan baru". Maka, kita akan mendapatkan hasil seperti ini:

Produk	Jumlah	Harga
Nike Air Max	20	420,000
Subtotal		Rp8,400,000
Nike Aeroloft Bomber	30	720,000
Subtotal		Rp21,600,000
Total		Rp30,000,000

Error ketika user sudah terdaftar dan memiliki password

Selanjutnya, kita akan bahas bagian **(B)**. Berikut codenya:

app/Http/Controllers/CheckoutController.php

```
protected function guestCheckout($email)
{
    // check if user exist, if so, ask login
    if ($user = User::where('email', $email)->first()) {
        if ($user->hasPassword()) {
            ...
        }
    }
    // (B) Logic ketika email di DB tanpa password
```

```

    // show view to request new password
    session()->flash('email', $email);
    return view('checkout.reset-password');
}
// (C) Logic ketika email tidak ada di DB
}

```

Pada bagian ini, kita akan menangani ketika email ditemukan tanpa password. Hal ini bisa terjadi karena pada saat proses pembuatan order untuk customer baru, kita akan membuat akun user tanpa password. Ketika kita menemukan record seperti ini, berarti ada yang pernah melakukan order dengan email tersebut.

Yang akan kita lakukan adalah mengarahkan user ke halaman untuk meminta link reset password. Disini, kita tidak akan menggunakan halaman reset password yang telah disediakan laravel. Kita akan membuat tampilan khusus agar sesuai dengan tahapan checkout.

Untuk menjalankan logic diatas ini yang kita lakukan:

- Kita set flash data `email` dengan email yang dikirim oleh user. Variable ini akan kita gunakan agar isian email langsung terisi di halaman reest reset password.
- Kita mengarahkan user ke custom view `checkout.reset-password`.

Isian untuk view `checkout.reset-password` sebagai berikut:

resources/views/checkout/reset-password.blade.php

```

@extends('layouts.app')

@section('content')


@include('checkout._step')


Permintaan Password



@include('checkout._request-password-form')


```

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

Pada view ini, kita juga menampilkan partial view `checkout._step`. Untuk form reset password, kita gunakan juga partial view `checkout._request-password-form`. Isinya akan seperti berikut:

resources/views/checkout/_request-password-form.blade.php

```
{!! Form::open(['url' => '/password/email', 'method'=>'post', 'class' => 'form-h\orizonta\l']) !!}
<div class="form-group {!! $errors->has('email') ? 'has-error' : '' !!}">
    {!! Form::label('email', 'Alamat Email', ['class' => 'col-md-4 control-lab\el']) !!}
    <div class="col-md-6">
        {!! Form::email('email', session()->has('email') ? session('email') : nu\ll, ['class'=>'form-control']) !!}
        <p class="help-block">Nampaknya Anda pernah berbelanja di Reselia. Klik \"Kirim petunjuk\" untuk meminta password baru.</p>
        {!! $errors->first('email', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::button('Kirim Petunjuk <i class="fa fa-arrow-right"></i>', \array('type' => 'submit', 'class' => 'btn btn-primary')) !!}
    </div>
</div>
{!! Form::close() !!}
```

Disini, kita menggunakan url yang sama dengan url untuk reset password yang telah disediakan Laravel yaitu `/password/email`. Sebagaimana reset password bawaan, kita juga hanya mengirim field `email` pada form ini. Secara default, menggunakan response dari URL bawaan untuk reset password adalah mengarahkan user ke halaman sebelumnya dan mengirimkan flash data bernama `status`. Ini bisa kita lihat di file `resources/views/auth/passwords/email.blade.php`.

Mari kita tampilkan isian pesan dari variable tersebut pada halaman `checkout.login`:

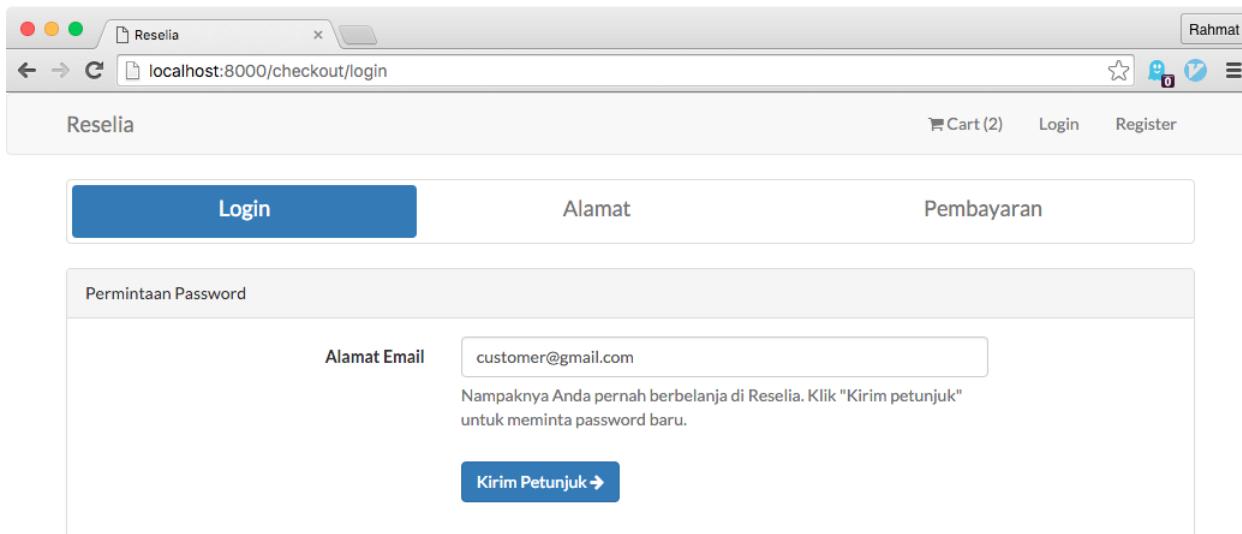
resources/views/checkout/login.blade.php

```
....  
    <div class="panel panel-default">  
        <div class="panel-heading">Login atau Checkout tanpa mendaftar</div>  
        <div class="panel-body">  
            @if (session('status'))  
                <div class="alert alert-success">  
                    {{ session('status') }}  
                </div>  
            @endif  
            @include('checkout._login-form')  
        </div>  
    </div>  
....
```

Untuk mengetes logic ini, kita harus menghapus isian pada field password untuk salah satu customer. Misalnya, kita hapus isian password pada sample user dengan email `customer@gmail.com`.

```
php artisan tinker  
>>> $c = App\User::where('email', 'customer@gmail.com')->first();  
>>> $c->password = null;  
>>> $c->save();
```

Cobalah mengisi halaman `checkout/login` dengan email tersebut, pilih “Saya adalah pelanggan baru”. Dan klik lanjut. Ini yang akan kita dapatkan:



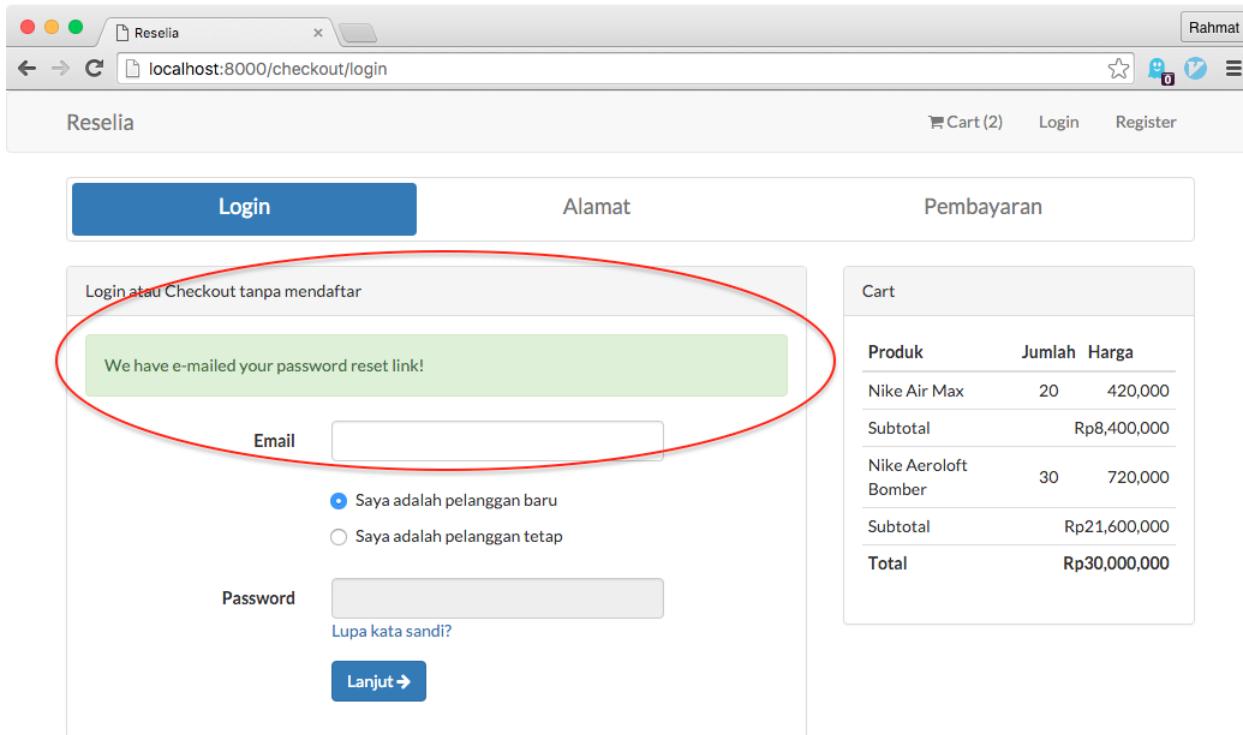
User diarahkan ke halaman reset password

Sebelum menekan tombol "Kirim Petunjuk", untuk memudahkan development, mari kita set agar email dikirim ke log. Ubah isian MAIL_DRIVER menjadi `log` di file `.env`:

`.env`

```
MAIL_DRIVER=log
```

Sekarang, jika kita tekan pada tombol "Kirim Petunjuk", kita akan diarahkan kembali ke halaman pertama dengan pesan default ketika reset password.



Email reset password dikirim

Bukalah file `storage/logs/laravel.log` pada baris paling bawah di file ini, akan kita temui text seperti ini:

storage/logs/laravel.log

[2016-03-04 07:20:09] local.DEBUG: Message-ID: <be074ffb132b0f72b02d039478159b7f\@localhost>

Date: Fri, 04 Mar 2016 07:20:09 +0000

Subject: Your Password Reset Link

From:

To: customer@gmail.com

MIME-Version: 1.0

Content-Type: text/html; charset=utf-8

Content-Transfer-Encoding: quoted-printable

Click here to **reset** your password: http://localhost:8000/password/**reset**/4312045f7cabcb3abea9ad289211\665e25e5b647e26cf7d21f97c0b938983706?email=customer%40gmail.com

Jika kita paste link diatas ke browser, maka kita akan melakukan proses reset password seperti biasa.

Bagian terakhir dari method `guestCheckout` adalah **(C)**. Berikut codenya:

app/Http/Controllers/CheckoutController.php

```
protected function guestCheckout($email)
{
    // check if user exist, if so, ask login
    if ($user = User::where('email', $email)->first()) {
        ....
    }
    // (C) Logic ketika email tidak ada di DB

    // save user data to session
    session(['checkout.email' => $email]);
    return redirect('checkout/address');
}
```

Method terakhir ini kita lakukan ketika email tidak ada di database. Pada saat itu kita menyimpan email yang didapat ke session. Untuk menyimpan data dari setiap tahapan checkout, kita akan menamai setiap variable session dengan prefix `session`. Disini, kita menyimpan email dengan nama variable `checkout.email`.

```
session(['checkout.email' => $email]);
```

Menggunakan prefix `checkout` akan memudahkan kita untuk menghapus semua data session yang berhubungan dengan proses checkout. Ini akan kita temui pada pembahasan untuk tahap 3.

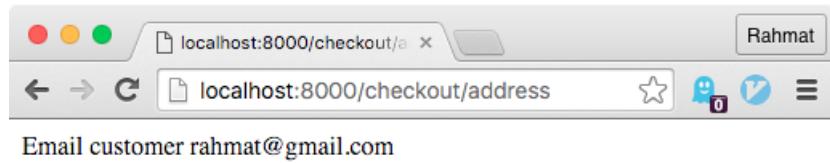
Mari kita tes syntax ini dengan menampilkan email yang dikirim pada url `/checkout/address`:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::get('checkout/address', function() {
        return "Email customer " . session()->get('checkout.email');
    });
});
```

Disini kita menggunakan `session()->get()` untuk mengambil data session.

Cobalah mengisi email yang belum ada di Reselia, pilih “Saya adalah pelanggan baru” dan klik “Lanjut”. Maka ini yang akan muncul:



Data session berhasil disimpan

Tahap 2: Alamat

Sebagaimana dijelaskan sebelumnya, kita akan menggunakan data alamat yang didukung oleh JNE yang akan bisa kita dapatkan dari situs rajaongkir.com.

The screenshot shows the RajaOngkir website's main page. At the top, there is a navigation bar with links for Beranda, Integrasi, Download, Bantuan, and Akun. The main content area features a large heading 'Aplikasi Android' with a subtext explaining that they have created an Android app for tracking shipping costs. A purple button labeled 'Unduh gratis' (Download for free) is visible. To the right of the text is an illustration of a smartphone displaying a map or tracking interface. Below this section, there is a call-to-action button with the text 'Periksa ongkos kirim dan lacak paket dengan cepat di sini!' (Check shipping rates and track packages quickly here!). At the bottom, there are input fields for 'Kota asal pengiriman' (Shipping origin city), 'Kota tujuan pengiriman' (Delivery destination city), 'Berat kiriman' (Shipment weight), and a 'Periksa Ongkir' (Check shipping rates) button. The overall design is clean and modern.

Rajaongkir sebenarnya memiliki API untuk banyak provider kurir. Kita sengaja hanya menggunakan JNE untuk mempersingkat tahapan ini. Selain hanya menggunakan JNE, kita juga hanya akan menggunakan tipe REG (meskipun tipe lain didukung oleh rajaongkir). Tentunya, pembaca dipersilahkan untuk mengeksplorasi lebih lanjut API rajaongkir jika ingin mendukung semua provider kurir di Reselia.

Untuk memulai, silahkan daftar ke rajaongkir dengan mengunjungi <http://rajaongkir.com/akun/daftar>. Setelah mendaftar, klik link pada email yang dikirim rajaongkir untuk melakukan aktivasi akun. Setelah berhasil login, kunjungi <http://rajaongkir.com/akun/panel>²¹⁴ untuk mendapatkan API Key.

²¹³<http://rajaongkir.com/akun/daftar>

²¹⁴<http://rajaongkir.com/akun/panel>

The screenshot shows a web browser window titled "Panel kendali API Raja Ongkir". The URL in the address bar is "rajaongkir.com/akun/panel". The page header includes the RajaOngkir logo and the text "Rajanya Ongkos Kirim Terpadu". On the left, there is a sidebar with links: Profil, API Key (which is selected and highlighted in blue), Upgrade, Ubah perujuk, Ubah password, Ubah email, Ubah API key, and Hapus akun. The main content area is titled "API Key" and contains a message: "Gunakan API Key ini untuk menggunakan API RajaOngkir. Untuk informasi lebih lanjut mengenai cara menggunakan API RajaOngkir, silakan baca [dokumentasi](#)". Below this is a large red box containing the API key value: "dfa4ea26d17b50e2698851676e3a9190". A warning message in orange text reads: "Peringatan: API key Anda berfungsi layaknya username dan password. Jaga baik-baik API key Anda!". There is also a "Tips:" section with the text: "Jika Anda merasa API Key Anda sudah diketahui orang lain, silakan ubah API Key dengan tombol di bawah. Anda juga bisa menggunakan perujuk untuk membatasi penggunaan API Key pada aplikasi/server tertentu." At the bottom of this section are two buttons: "Ubah API Key" and "Ubah Perujuk". At the very bottom of the page is a purple footer bar with the text "Ikuti kami di" followed by icons for Facebook, Twitter, and Google Plus.

Halaman API Key rajaongkir

Di Rajaongkir, kita akan menggunakan akun gratis (starter). Pada jenis akun ini, kita hanya dapat mengakses wilayah sampai level kabupaten/kota. Dokumentasi API untuk akun ini bisa dilihat di <http://rajaongkir.com/dokumentasi/starter>²¹⁵.

²¹⁵<http://rajaongkir.com/dokumentasi/starter>

Dokumentasi ini menjelaskan cara mengakses layanan API RajaOngkir untuk akun **Starter**. Akun **Starter** merupakan akun gratis dengan fitur pengecekan ongkos kirim (ongkir) untuk kurir JNE, POS Indonesia, dan TIKI. Jika Anda membutuhkan fitur lain seperti lacak paket JNE, ongkos kirim internasional, dan ongkos kirim sampai level kecamatan, silakan [upgrade akun Anda](#).

Province

Ringkasan

Method "province" digunakan untuk mendapatkan daftar propinsi yang ada di Indonesia.

Request

URL	Parameter	Contoh request
Method	URL	

Halaman dokumentasi API

Kita bisa saja menggunakan curl atau [guzzle²¹⁶](#), tapi kita tidak akan melakukan itu. Untuk mempercepat development, kita akan menggunakan library yang akan memudahkan kita membuat API call ke rajaongkir yaitu [rizalafani\rajaongkirlaravel²¹⁷](#). Kita install dengan perintah berikut:

```
composer require rizalafani/rajaongkirlaravel dev-master
```

Kita sengaja menggunakan branch `master` ketika menginstall package ini. Teknik menginstall package seperti ini adalah *bad practices*. Jika versi terbaru dari package ini sudah memiliki semua commit terbaru, hilangkan `dev-master` dari syntax diatas. Ini kita lakukan karena ada beberapa commit yang belum masuk ke versi terakhir.

Setelah terinstal, tambahkan isian berikut pada array `providers` di `config/app.php`:

²¹⁶<https://github.com/guzzle/guzzle>

²¹⁷<https://github.com/rizalafani/rajaongkirlaravel>

config/app.php

```
.....
'providers' = [
    ...
    rizalafani\rajaongkirlaravel\RajaOngkirServiceProvider::class,
]
.....
```

Dan isian berikut pada array `aliases`:

config/app.php

```
.....
'aliases' => [
    ...
    'RajaOngkir' => rizalafani\rajaongkirlaravel\RajaOngkirFacade::class,
]
```

Untuk menggunakan package ini, kita harus membuat file konfigurasi. Kita generate templatanya dengan perintah berikut:

```
php artisan vendor:publish
Copied Directory [/vendor/laracasts/flash/src/views] To [/resources/views/vendor\
/flash]
Copied File [/vendor/rizalafani/rajaongkirlaravel/src/config/rajaongkir.php] To \
[/config/rajaongkir.php]
Publishing complete for tag []!
```

File konfigurasinya akan muncul di `config/rajaonkir.php`:

config/rajaongkir.php

```
<?php

return [
    'end_point_api' => env('RAJAONGKIR_ENDPOINTAPI', 'http://rajaongkir.com/api/\\
starter'),
    'api_key' => env('RAJAONGKIR_APIKEY', 'SomeRandomString'),
];
```

Terlihat disini, kita harus mengeset konfigurasi untuk API key rajaongkir. Kita dapat langsung mengisinya disini atau membuat key baru di file `.env` seperti berikut:

.env

RAJAONGKIR_APIKEY=key-yang-kita-dapatkan-dari-raja-ongkir

Sip. Kita dapat mengetes package ini dengan menggunakan tinker:

```
php artisan tinker
>>> // Mengambil data provinsi
>>> Rajaongkir::Provinsi()->all();
>>> // Mengambil dari kabupaten
>>> Rajaongkir::Kota()->all();
>>> // Mencari kota yang memiliki nama "Bandung"
>>> RajaOngkir::Kota()->search('city_name', 'Bandung')->get();
>>> // Mencari kota yang memiliki nama "Bekasi"
>>> RajaOngkir::Kota()->search('city_name', 'bekasi')->get();
>>> // Mencari ongkos kirim dari "Kota Bandung" (23)
>>> // ke "Kota Bekasi" (55)
>>> // Dengan berat 1kg
>>> // Dan courier "JNE"
>>> RajaOngkir::cost([
    'origin' => 23,
    'destination' => 55,
    'weight' => 1000,
    'courier' => 'jne'
])->get();
```

Untuk mengetahui method lain yang tersedia, silahkan kunjungi dokumentasi resmi di [https://github.com/rizalafani/rajaongkirlaravel²¹⁸](https://github.com/rizalafani/rajaongkirlaravel).

Teknik penyimpanan alamat user

Di Reselia, user dapat memiliki lebih dari satu alamat. Alamat ini dibuat ketika user melakukan checkout pertama kali. Setelah user memiliki akun, dia dapat membuat alamat baru ketika melakukan checkout yang lain. Untuk memudahkan development kita sengaja tidak memberikan user akses untuk melakukan CRUD terhadap alamat. Dengan teknik ini, kita juga menghindari penggunaan fitur ini dari penyalah gunaan fitur (memenuhi database dengan data address).

Mari kita buat model dan migration untuk menyimpan alamat:

²¹⁸<https://github.com/rizalafani/rajaongkirlaravel>

```
php artisan make:model Address -m  
php artisan make:model Province  
php artisan make:model Regency
```

Pada perintah diatas, kita membuat 3 model:

- Address: untuk menyimpan detail alamat user
- Province: untuk menyimpan data provinsi dari rajaongkir
- Regency: untuk menyimpan data kabupaten dari rajaongkir

Untuk memudahkan migration, semua table untuk model diatas kita buat pada satu migration. Pada migration yang dihasilkan isilah method `up` dengan syntax berikut:

database/migrations/xxxx_xx_xx_xxxxxxx_create_addresses_table.php

```
public function up()  
{  
    Schema::create('provinces', function (Blueprint $table) {  
        $table->char('id', 2)->primary();  
        $table->string('name');  
        $table->timestamps();  
    });  
  
    Schema::create('regencies', function (Blueprint $table) {  
        $table->char('id', 4)->primary();  
        $table->char('province_id', 2);  
        $table->string('name');  
  
        $table->foreign('province_id')->references('id')->on('provinces');  
        $table->timestamps();  
    });  
  
    Schema::create('addresses', function (Blueprint $table) {  
        $table->increments('id');  
        $table->integer('user_id')->unsigned();  
        $table->string('name');  
        $table->string('detail');  
        $table->char('regency_id', 4);  
        $table->string('phone');  
  
        $table->foreign('regency_id')->references('id')->on('regencies');  
        $table->foreign('user_id')->references('id')->on('users');  
    });  
}
```

```

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

```

Terlihat disini, kita membuat 3 table sekaligus pada method ini. Tidak ada yang spesial dari struktur ketiga table ini. Yang perlu diperhatikan adalah kita hanya menyimpan data `regency_id` pada table `addresses`. Ini kita lakukan karena akun starter rajaongkir hanya mengizinkan pencarian ongkos kirim sampai level kabupaten.

Pada method `down` kita hapus ketiga table tersebut:

database/migrations/xxxx_xx_xx_xxxxxxx_create_addresses_table.php

```

public function down()
{
    Schema::drop('addresses');
    Schema::drop('regencies');
    Schema::drop('provinces');
}

```

Mari kita buat sesuaikan syntax untuk ketiga model diata. Kita mulai dari model `Province`, ubah menjadi seperti berikut:

app/Province.php

```

<?php

namespace App;

use RajaOngkir;
use Illuminate\Database\Eloquent\Model;

class Province extends Model
{
    protected $fillable = ['id', 'name'];

    public static function populate() {
        foreach (RajaOngkir::Provinsi()->all() as $province) {
            $model = static::firstOrNew(['id' => $province['province_id']]);
            $model->name = $province['province'];
            $model->save();
        }
    }
}

```

Disini, kita mengaktifkan *mass assignment* dan membuat method `populate` untuk mengisi record pada table `provinces`. Ini kita lakukan dengan menggunakan method `RajaOngkir::Provinsi()->all()` untuk mengambil semua provinsi yang didukung oleh RajaOngkir.

Pada model `Regency`, kita juga buat method `populate`:

app/Regency.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use RajaOngkir;

class Regency extends Model
{
    protected $fillable = ['id', 'province_id', 'name'];

    public static function populate() {
        foreach (RajaOngkir::Kota()->all() as $kota) {
            $model = static::firstOrNew(['id' => $kota['city_id']]);
            $model->province_id = $kota['province_id'];
            $model->name = $kota['type'] . ' ' . $kota['city_name'];
            $model->save();
        }
    }

    public function province()
    {
        return $this->belongsTo('App\Province');
    }
}
```

Berbeda dengan model `Province`, disini kita menggunakan method `RajaOngkir::Kota()->all()` untuk mengambil data kabupaten untuk mengisi table `regencies`.

Untuk model `Address`, akan kita buat seperti ini:

app/Address.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Address extends Model
{
    protected $fillable = ['user_id', 'name', 'detail', 'regency_id', 'phone'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }

    public function regency()
    {
        return $this->belongsTo('App\Regency');
    }
}
```

Mari kita buat seeder nya:

```
php artisan make:seeder AddressesSeeder
```

Pada file yang dihasilkan kita isi sebagai berikut:

database/seeds/AddressesSeeder.php

```
<?php

use Illuminate\Database\Seeder;
use App\User;
use App\Address;
use App\Province;
use App\Regency;

class AddressesSeeder extends Seeder
{
    /**

```

```
* Run the database seeds.  
*  
* @return void  
*/  
public function run()  
{  
    Province::populate();  
    Regency::populate();  
  
    // sample address for customer  
    $customer = User::where('email', 'customer@gmail.com')->first();  
    $address1 = Address::create([  
        'user_id' => $customer->id,  
        'name' => 'Budi',  
        'detail' => 'Kp Cipadung RT 4 RW 9 Ds Cipadung',  
        // Kota Cimahi, Jawa Barat,  
        'regency_id' => 107,  
        'phone' => '87823451238',  
    ]);  
  
    $address2 = Address::create([  
        'user_id' => $customer->id,  
        'name' => 'Susi',  
        'detail' => 'Kp Karang Jati RT 19 RW 23 Ds Sukamahi',  
        // Kota Bekasi, Jawa Barat,  
        'regency_id' => 55,  
        'phone' => '87823451238',  
    ]);  
}  
}
```

Pada file seeder ini, kita memanggil method `populate` pada model `Province` dan `Regency` untuk mengisi table `provinces` dan `regencies`. Kita juga membuat dua sample alamat untuk sample customer.

Kita panggil seeder ini dari `DatabaseSeeder`:

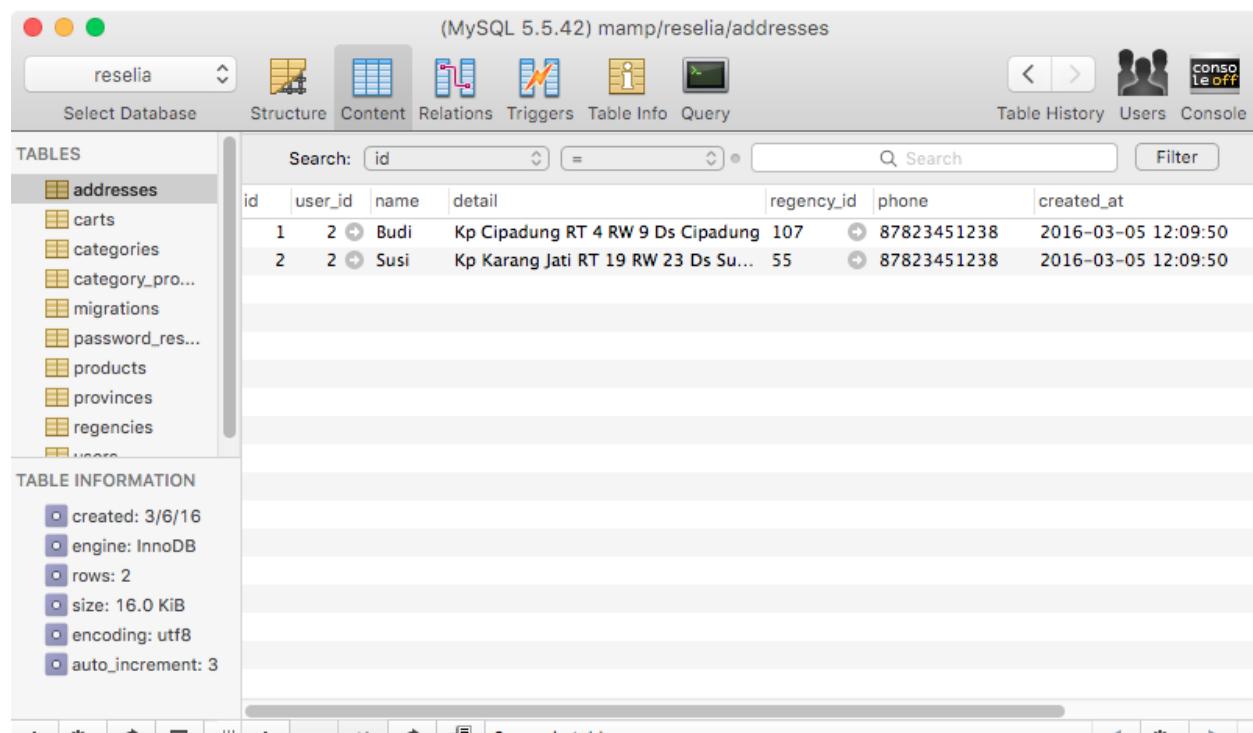
database/seeds/DatabaseSeeder.php

```
public function run()
{
    $this->call(UsersSeeder::class);
    $this->call(ProductsSeeder::class);
    $this->call(AddressesSeeder::class);
}
```

Refresh dan seed lagi Reselia:

```
php artisan migrate:refresh --seed
```

Pastikan di database table provinces, regencies dan addresses sudah memiliki isi.



(MySQL 5.5.42) mamp/reselia/addresses							
Select Database		Structure	Content	Relations	Triggers	Table Info	Query
TABLES							
addresses							
carts							
categories							
category_pro...							
migrations							
password_res...							
products							
provinces							
regencies							
users							
TABLE INFORMATION							
<input type="checkbox"/> created: 3/6/16							
<input type="checkbox"/> engine: InnoDB							
<input type="checkbox"/> rows: 2							
<input type="checkbox"/> size: 16.0 KiB							
<input type="checkbox"/> encoding: utf8							
<input type="checkbox"/> auto_increment: 3							
Sample data address							
id	user_id	name	detail	regency_id	phone	created_at	
1	2	Budi	Kp Cipadung RT 4 RW 9 Ds Cipadung 107	107	87823451238	2016-03-05 12:09:50	
2	2	Susi	Kp Karang Jati RT 19 RW 23 Ds Su... 55	55	87823451238	2016-03-05 12:09:50	

Karena kini kita menggunakan data dari rajaongkir, pastikan terhubung ke internet setiap kali melakukan seeding.

Teknik menentukan ongkos kirim

Di pembahasan sebelumnya, kita mempelajari bahwa kita dapat dengan mudah mengambil data ongkos kirim dari rajaongkir. Berikut contohnya:

```
RajaOngkir::cost([
    'origin' => 23,
    'destination' => 55,
    'weight' => 1000,
    'courier' => 'jne'
])->get();
```

Hasilnya akan seperti berikut:

```
[
  [
    [
      "code" => "jne",
      "name" => "Jalur Nugraha Ekakurir (JNE)",
      "costs" => [
        [
          "service" => "OKE",
          "description" => "Ongkos Kirim Ekonomis",
          "cost" => [
            [
              "value" => 10000,
              "etd" => "2-3",
              "note" => "",
            ],
          ],
        ],
        [
          "service" => "REG",
          "description" => "Layanan Reguler",
          "cost" => [
            [
              "value" => 11000,
              "etd" => "1-2",
              "note" => "",
            ],
          ],
        ],
      ],
    ],
    ...
  ]
] // Tipe pengiriman yang lainnya
```

Mari kita diskusikan apa saja yang bisa kita lakukan agar request ke rajaongkir ini lebih efisien.

Ketika kita melakukan request ke rajaongkir, kita bisa memberikan opsi berikut:

- Kota asal
- Kota tujuan
- Berat
- Kurir

Dari array result yang dihasilkan, berikut beberapa key yang perlu diperhatikan.

```
[  
  [  
    [  
      // detail jne  
      'costs' => [  
        [  
          // detail untuk tipe service  
          'service' => 'kode-service'  
          'cost' => [  
            'value' => 'harga-untuk-berat'  
            // detail lainnya  
          ]  
        ],  
        // detail untuk tipe service lainnya  
      ]  
    ]  
  ]
```

Tujuan akhir kita adalah mendapatkan isian value untuk tipe service yang sesuai. **Teknik yang paling efektif untuk memaksimalkan request ke RajaOngkir adalah dengan tidak membuat request sama sekali.** Nah, bagaimana cara melakukannya? Kita akan melakukan caching terhadap hasil yang kita dapatkan.

Berikut beberapa hal yang akan kita lakukan.

- Kita akan menyimpan data request ke rajaongkir pada table fees dengan detail parameter yang digunakan untuk request.
- Pada table yang disimpan, kita juga menambahkan jenis service yang akan digunakan.
- Ketika mencari lagi ongkos kirim dengan parameter yang sama, kita akan mencari dulu record di database. Jika ditemukan, ada dua hal yang akan kita lakukan. Pertama, jika isian updated_at lebih dari 7 hari yang lalu, kita lakukan request ke rajaongkir dan update record di database. Jika tidak kurang dari 7 hari, kita langsung ambil isian dari database.

Mari kita mulai dengan membuat model dan migration nya:

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

Pada file migration yang dihasilkan, isi method `up` dengan syntax berikut:

database/migrations/xxxx_xx_xx_xxxxxxx_create_fees_table.php

```
public function up()
{
    Schema::create('fees', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('origin');
        $table->integer('destination');
        $table->string('courier');
        $table->string('service');
        $table->integer('weight');
        $table->integer('cost');
        $table->timestamps();
    });
}
```

Berikut isian untuk modelnya:

app/Fee.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Fee extends Model
{
    public static function getCost($origin_id, $destination_id, $weight, $courier, $service)
    {
        $fee = static::firstOrCreate([
            'origin'      => $origin_id,
            'destination' => $destination_id,
            'weight'      => $weight,
            'courier'     => $courier,
            'service'     => $service
        ]);
    }
}
```

```
    if ($fee->haveCost() > 0 && !$fee->isNeedUpdate()) return $fee->cost;
    return $fee->populateCost();
}
}
```

Ini baru sebagian, method yang kita buat bernama `getCost`. Method ini berfungsi mengambil nilai `cost` untuk parameter yang kita kirimkan. Berikut penjelasan method ini:

- Menggunakan method `firstOrCreate` kita membuat atau mencari record di table `fees` dengan kriteria yang kita inginkan.
- Setelah record diambil, kita cek apakah isian `cost`nya memiliki isi (lebih dari 0) dengan memanggil method `$fee->haveCost()` dan update terakhir tidak lebih dari 7 hari dengan memanggil `!$fee->isNeedUpdate()`. Jika ya, kita berikan isian `cost` tersebut.
- Jika pengecekan diatas gagal, kita panggil method `populateCost()` untuk mengisi field tersebut dengan melakukan request ke RajaOngkir dan mengembalikan nilainya.

Berikut syntax unutk method `haveCost` dan `isNeedUpdate`:

app/Fee.php

```
<?php
...
use Carbon\Carbon;

class Fee extends Model
{
    ...
    protected function haveCost()
    {
        return $this->cost > 0;
    }

    protected function isNeedUpdate()
    {
        return $this->updated_at->diffInDays(Carbon::today()) > 7;
    }
}
```

Pada method `haveCost` kita mengecek apakah isian field `cost` lebih dari 0. Sedangkan pada method `isNeedUpdate` kita menggunakan `Carbon` untuk memudahkan mengecek apakah update terakhir lebih dari 7 hari. Kita sengaja menggunakan `protected` karena kedua method ini hanya akan kita gunakan di dalam class ini.

Berikut syntax untuk method `populateCost`:

app/Fee.php

```
<?php
...
use RajaOngkir;
use App\Product;

class Fee extends Model
{
    protected $fillable = ['origin', 'destination', 'courier', 'service',
        'weight', 'cost', 'updated_at'];

    ...
    ...

    public function populateCost()
    {
        $params = $this->toArray();
        $costs = RajaOngkir::Cost($params)->get();

        $cost = 0;
        foreach ($costs[0]['costs'] as $result) {
            if ($result['service'] == $this->service) {
                $cost = $result['cost'][0]['value'];
                break;
            }
        }

        $cost = $cost > 0 ? $cost : config('rajaongkir.fallback_fee');
        // update cost & force update `updated_at` field
        $this->update(['cost' => $cost, 'updated_at'=>Carbon::today()]);
        return $cost;
    }
}
```

Berikut penjelasan untuk method ini:

- Untuk membuat parameter rajaongkir, kita menggunakan method `toArray()`. Method ini akan merubah semua field pada model Fee menjadi array. Hasilnya kita simpan pada variable `$params`.
- Kita cari ongkos kirim dengan `RajaOngkir::Cost($params)->get()`.
- Kita set nilai default untuk `$cost` berupa 0.
- Kita loop hasil dari rajaongkir dan mencari cost untuk tipe service yang sesuai. Jika ditemukan, kita set nilai `$cost` dengan hasil tersebut.
- Selesai looping, kita cek kembali isian `$cost`. Menggunakan ternary operator, kita mengecek apakah isiannya lebih dari 0, jika ya, kita tidak merubahnya. Jika tidak, kita isi dengan nilai default (kita bahas di pembahasan selanjutnya).
- Menggunakan method `update`, kita update field `cost` dan `updated_at`. Disini, kita melakukan update untuk field `updated_at` secara manual karena Laravel tidak akan merubah isian field ini jika tidak ada field yang berubah. Ini kita butuhkan jika ternyata setelah satu minggu harganya tetap tidak berubah (sangat mungkin). Agar syntax ini berfungsi, kita tambahkan field `updated_at` pada setting untuk mass assignment (attribut `$fillable`).
- Terakhir, kita kembalikan nilai `$cost` yang didapatkan.

Seperti penjelasan diatas, kita akan menggunakan harga default jika ongkos kirim tidak ditemukan. Syntax `config('rajaongkir.fallback_fee')` artinya kita membuat field `fallback_fee` pada file `config/rajaongkir.php`. Berikut syntaxnya:

config/rajaongkir.php

```
return [
    ...
    'fallback_fee' => env('RAJAONGKIR_FALLBACK_FEE', 40000)
]
```

Disini, kita set nilai default ke 40000 jika kita tidak menemukan isian `RAJAONGKIR_FALLBACK_FEE` pada file `.env`.

Untuk mengecek method yang kita buat ini, kita menggunakan tinker:

```
php artisan tinker
>>> App\Fee::getCost(23, 55, 1000, 'jne', 'REG');
=> 11000
```

Setelah menjalankan method diatas, akan muncul record baru di table fees. Jika kita jalankan kembali method diatas, responsenya akan lebih cepat karena kita tidak melakukan request ke RajaOngkir.

Cobalah ubah isian `updated_at` ke lebih dari 7 hari yang lalu dan isian `cost`. Ketika kita memanggil kembali method diatas, maka Reselia akan melakukan request ke rajaongkir dan mengubah isian field `cost` dan `updated_at` ke tanggal hari ini.

	id	origin	destination	courier	service	weight	cost	created_at	updated_at
	1	23	55	jne	REG	1000	11000	2016-03-06 09:06:09	2016-03-06 00:00:00

Record fee sebagai cache ongkos kirim

Untuk memudahkan dalam menentukan ongkos kirim suatu produk, kita akan membuat satu method `getCostTo` di model Product. Method ini akan kita gunakan untuk menentukan ongkos kirim produk tersebut ke lokasi yang dituju. Berikut syntaxnya:

app/Product.php

```
<?php
...
use App\Fee;

class Product extends Model
{
    ...
    public function getCostTo($destination_id)
    {
        return Fee::getCost(
            config('rajaongkir.origin'),
            $destination_id,
            $this->weight,
            config('rajaongkir.courier'),
            config('rajaongkir.service')
        );
    }
}
```

Pada method ini kita menggunakan method `getCost` yang telah kita buat sebelumnya pada model Fee untuk menentukan ongkos kirim. Untuk isian `origin_id`, `courier` dan `service`, kita akan menggunakan isian yang kita buat pada file konfigurasi. Berikut isianya:

config/rajaongkir.php

```
return [
    ...
    'origin' => env('RAJAONGKIR_REGENCY_ORIGIN', 23), // Kota Bandung, Jawa Barat
    'courier' => env('RAJAONGKIR_COURIER', 'jne'),
    'service' => env('RAJAONGKIR_SERVICE', 'REG'),
]
```

Disini, kita set default lokasi gudang di Kota Bandung (id 23). Jika mengubahnya, silahkan cek id regency di table `regencies`. Untuk konfigurasi lain, saya rasa cukup jelas.

Pada method ini, terlihat kita juga menggunakan `$this->weight` untuk mengakses berat dari sebuah produk. Field ini belum kita tambahkan pada model Product. Mari kita lakukan dengan cepat.

Pada file migration untuk table `products`, tambah baris ini pada method `up`:

database/migrations/xxxx_xx_xx_xxxxxx_create_products_table.php

```
...
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        ...
        $table->decimal('weight', 8, 2);
    });
}
```

Pada model Product, tambahkan `weight` pada mass assignment:

app/Product.php

```
...
protected $fillable = ['name', 'photo', 'model', 'price', 'weight'];
...
```

Di ProductController, tambahkan validasi untuk `weight` di method `store` dan `update`:

app/Http/Controllers/ProductsController.php

```

.....
public function store(Request $request)
{
    $this->validate($request, [
        ...
        'weight' => 'required|numeric|min:1'
    ]);
    $data = $request->only('name', 'model', 'price', 'weight');
    ...
}

.....
public function update(Request $request, $id)
{
    $product = Product::findOrFail($id);
    $this->validate($request, [
        ...
        'weight' => 'required|numeric|min:1'
    ]);
    $data = $request->only('name', 'model', 'price', 'weight');
    ...
}
.....

```

Pada `resources/views/products/_form.blade.php` tambahkan field untuk menambah berat produk:

resources/views/products/_form.blade.php

```

.....
<div class="form-group {!! $errors->has('weight') ? 'has-error' : '' !!}">
    {!! Form::label('weight', 'Berat (gram)') !!}
    {!! Form::number('weight', null, ['class'=>'form-control']) !!}
    {!! $errors->first('weight', '<p class="help-block">:message</p>') !!}
</div>
.....

```

Pada `database/seeds/ProductsSeeder.php`, isi field `weight` untuk tiap produk. Misalnya, kita bisa gunakan cara syntax ini untuk mengisi `weight` dengan nilai 1000, 2000 atau 3000:

```
'weight' => rand(1,3) * 1000,
```

Sip. Kini, refresh dan seed kembali database:

```
php artisan migrate:refresh --seed
```

Pastikan isian database sudah memiliki weight:

	id	name	photo	model	price	weight	created_at
1	Nike Air Force	stub-shoe.jpg	Sepatu Pria	340000.00	2000	2016-03-06 14:37:19	
2	Nike Air Max	stub-shoe.jpg	Sepatu Wanita	420000.00	3000	2016-03-06 14:37:19	
3	Nike Air Zoom	stub-shoe.jpg	Sepatu Wanita	360000.00	2000	2016-03-06 14:37:19	
4	Nike Aeroloft Bomber	stub-jacket.jpg	Jaket Wanita	720000.00	3000	2016-03-06 14:37:19	
5	Nike Guild 550	stub-jacket.jpg	Jaket Pria	380000.00	3000	2016-03-06 14:37:19	
6	Nike SB Steele	stub-jacket.jpg	Jaket Pria	1200000.00	3000	2016-03-06 14:37:19	

field weight di table products

Dan pada form untuk menambah/mengubah produk, ada field berat barang:

The screenshot shows a web browser window titled 'Reselia'. The address bar displays 'localhost:8000/products/create'. The page header includes 'Reselia', 'Home', 'Manage', and 'Admin'. The main content is a 'New Product' form with fields for Name, Model, Berat (gram), Harga, Categories, and Product photo (jpeg, png). A 'Save' button is at the bottom. The 'Berat (gram)' field is highlighted.

New Product

Name

Model

Berat (gram)

Harga

Categories

Product photo (jpeg, png)
Choose File No file chosen

Save

Field berat di form produk

Mari kita test method `getCostTo` dengan tinker:

```
php artisan tinker
>>> App\Product::find(1)->getCostTo(55);
```

Untuk menggunakan method ini, pastikan laptop terhubung ke internet.

Membuat form alamat

Mari kita buat form untuk menampilkan pembuatan penentuan alamat ini. Kita mulai dari route:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    Route::get('checkout/address', function() {
        return "Email customer ". session()->get('checkout.email');
    });
    Route::get('checkout/address', 'CheckoutController@address');
    Route::post('checkout/address', 'CheckoutController@postAddress');
});
```

Kita buat method address pada CheckoutController dengan isi sebagai berikut:

app/Http/Controllers/CheckoutController.php

```
public function address()
{
    return view('checkout.address');
}
```

Kita buat viewnya seperti berikut:

resources/views/checkout/address.blade.php

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

@section('content')


@include('checkout._step')


Alamat Pengiriman



@include('checkout._address-new-form')



@include('checkout._cart-panel')


```

Pada view ini kita tetap menggunakan partial view untuk menampilkan tahapan checkout dan sidebar untuk menampilkan kondisi cart. Untuk menampilkan form, kita menggunakan partial view `checkout._address-new-form`. Berikut isinya:

resources/views/checkout/_address-new-form.blade.php

```
{!! Form::open(['url' => '/checkout/address', 'method'=>'post', 'class' => 'form\horizonal']) !!}

@include('checkout._address-field')

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
       >{!! Form::button('Lanjut <i class="fa fa-arrow-right"></i>', array('type' => 'submit', 'class' => 'btn btn-primary')) !!}
    </div>
</div>

{!! Form::close() !!}
```

Form ini kita arahkan ke url `/checkout/address` yang akan kita bahas di pembahasan selanjutnya. Sekali lagi, kita menggunakan partial view untuk menampilkan field untuk form ini di `checkout._address-field`. Berikut isinya:

resources/views/checkout/_address-field.blade.php

```
<div class="form-group {!! $errors->has('name') ? 'has-error' : '' !!}">
    {!! Form::label('name', 'Nama', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::text('name', null, ['class'=>'form-control']) !!}
        {!! $errors->first('name', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group {!! $errors->has('detail') ? 'has-error' : '' !!}">
    {!! Form::label('detail', 'Alamat', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::textarea('detail', null, ['class'=>'form-control', 'rows' => 3]) !!}
        {!! $errors->first('detail', '<p class="help-block">:message</p>') !!}
    </div>
</div>
```

```

<div class="form-group {!! $errors->has('province_id') ? 'has-error' : '' !!}">
    {!! Form::label('province_id', 'Provinsi', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::select('province_id', [''=>'']+DB::table('provinces')->lists('name', 'id'), null, ['class'=>'form-control']) !!}
        {!! $errors->first('province_id', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group {!! $errors->has('regency_id') ? 'has-error' : '' !!}">
    {!! Form::label('regency_id', 'Kabupaten / Kota', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::select('regency_id', [], old('regency_id'), ['class'=>'form-control']) !!}
        {!! $errors->first('regency_id', '<p class="help-block">:message</p>') !!}
    </div>
</div>
n
<div class="form-group {!! $errors->has('phone') ? 'has-error' : '' !!}">
    {!! Form::label('phone', 'Telepon', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        <div class="input-group">
            <div class="input-group-addon">+62</div>
            {!! Form::text('phone', null, ['class'=>'form-control']) !!}
        </div>
        {!! $errors->first('phone', '<p class="help-block">:message</p>') !!}
    </div>
</div>

```

Pada view ini, kita menampilkan field berikut:

- Nama
- Detail alamat
- Provinsi berupa dropdown dengan isian dari table provinces
- Kabupaten / Kota yang masih kosong. Untuk mengisi pilihan di form ini, kita akan menggunakan dependent dropdown yang akan kita bahas di pembahasan selanjutnya.

Berikut tampilan dari form ini:

Produk	Jumlah	Harga
Nike Air Max	20	420,000
Subtotal		Rp8,400,000
Nike Aeroloft Bomber	30	720,000
Subtotal		Rp21,600,000
Total		Rp30,000,000

Halaman pembuatan alamat

Membuat dependent dropdown untuk memilih kota

Pada saat kita menentukan alamat, kita akan melakukan pemilihan provinsi dan kabupaten. Sengaja kita batasi sampai level kabupaten karena akun RajaOngkir starter hanya mendukung pengecekan ongkos kirim sampai level kabupaten. Pilihan yang muncul pada Kabupaten akan tergantung (*dependent*) pada Provinsi yang kita pilih.

Untuk mengembangkan fitur ini, kita akan menggunakan library `selectize` yang sudah kita install di tahap sebelumnya. Pada halaman dokumentasi library ini, sudah ada contoh untuk membuat dependent dropdown di <http://selectize.github.io/selectize.js/#demo-cities>²¹⁹.

²¹⁹<http://selectize.github.io/selectize.js/#demo-cities>

The screenshot shows a web browser window titled "Selectize.js" with the URL "selectize.github.io/selectize.js/#demo-cities". The page title is "City / State Selection". It features two dropdown menus: one for "State" (labeled "Pick a state...") and one for "City" (labeled "Pick a city..."). Below each dropdown is a "Current Value: """. A note on the page states: "A demonstration showing how to use the API to cascade controls for a classic state / city selector. Note: The API for fetching cities is a little spotty, so if it fails to list cities, that's what's going on (try another state)". At the bottom left, there is a "Hide Code" button with a blue dashed border, and below it is a block of JavaScript code:

```

var xhr;
var select_state, $select_state;
var select_city, $select_city;

$select_state = $('#select-cities-state').selectize({
    onChange: function(value) {
        if (!value.length) return;
        select_city.disable();
        select_city.clearOptions();
        select_city.load(function(callback) {
            xhr && xhr.abort();
            xhr = $.ajax({
                url: 'https://reselia.id/api/v1/provinces/' + value + '/regencies',
                type: 'GET',
                success: callback
            });
        });
    }
});

```

Themes: **default** legacy bootstrap2 bootstrap3

Dokumentasi untuk dependent dropdown

Yang akan kita lakukan adalah memodifikasi syntax ini agar sesuai dengan Reselia. Untuk membuat fitur ini, kita akan membutuhkan URL untuk mendapatkan semua Kabupaten dalam bentuk json berdasarkan `province_id` yang kita kirim. Mari kita buat controller baru:

```
php artisan make:controller AddressController
```

Pada controller yang dihasilkan kita buat method `regencies` seperti berikut:

app/Http/Controllers/AddressController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;
use App\Regency;

class AddressController extends Controller
{
    public function regencies(Request $request)
    {
        $this->validate($request, [
            'province_id' => 'required|exists:provinces,id'
        ]);

        return Regency::where('province_id', $request->get('province_id'))
            ->get();
    }
}
```

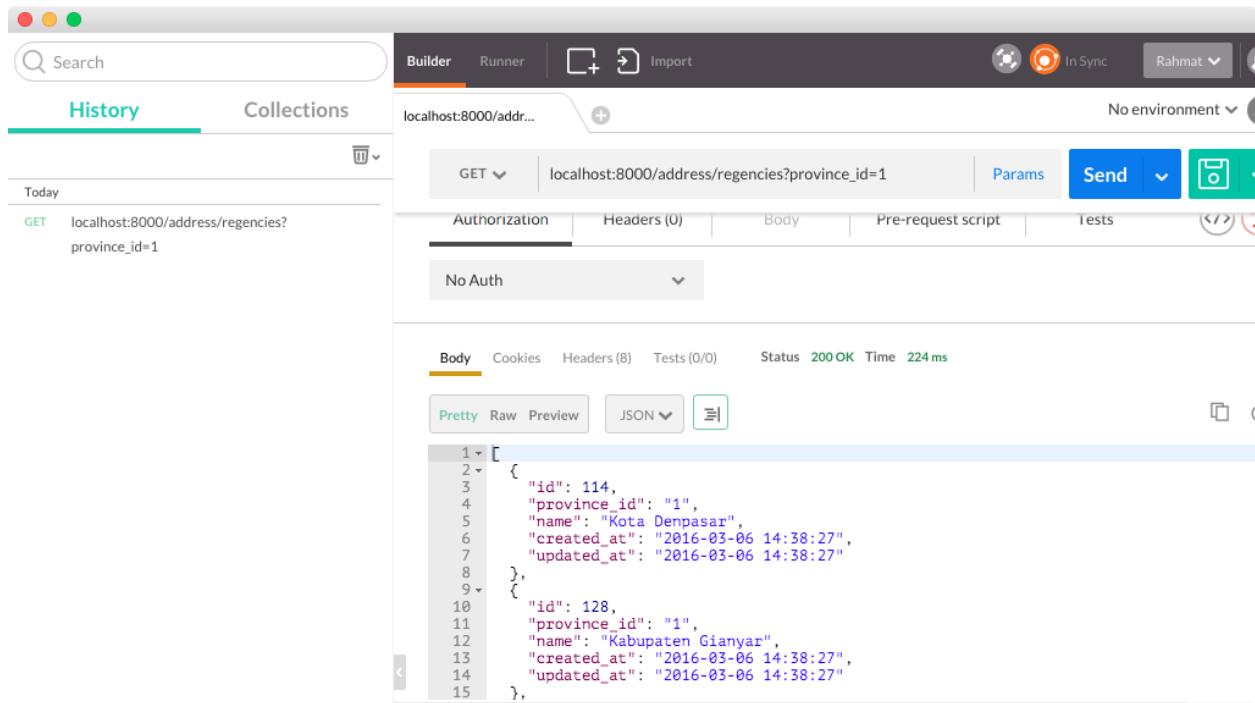
Pada method ini kita melakukan validasi untuk mengecek bahwa field `province_id` dikirim oleh user dan ada pada table `provinces`. Setelah data valid, kita cari semua Kabupaten yang memiliki `province_id` tersebut dan mengirimkan response json.

Kita buat routenya:

app/Http/routes.php

```
Route::group(['middleware' => 'api'], function () {
    Route::get('address/regencies', 'AddressController@regencies');
});
```

Route untuk method ini menggunakan middleware group `api`, bukan `middleware`. Dengan menggunakan middleware group `api`, request kita akan lebih cepat karena banyak middleware yang tidak diaktifkan, misalnya pengecekan session. Kita bisa mengetesnya dengan menggunakan Postman dengan mengakses url `/address/regencies?province_id={id}`.



Mengetes URL untuk mendapatkan data kabupaten

Mari kita tambahkan syntax javascript pada `resources/assets/js/app.js` untuk membuat fitur ini:

`resources/assets/js/app.js`

```

$(document).ready(function () {
  ...
  if ($('#province_selector').length > 0) {
    var xhr
    var province_selector, $province_selector
    var regency_selector, $regency_selector

    $province_selector = $('#province_selector').selectize({
      sortField: 'text',
      onChange: function (value) {
        if (!value.length) {
          regency_selector.disable()
          regency_selector.clearOptions()
          return
        }
        regency_selector.clearOptions()
        regency_selector.load(function (callback) {
          xhr && xhr.abort()
        })
      }
    })
  }
})

```

```

xhr = $.ajax({
    url: '/address/regencies?province_id=' + value,
    success: function (results) {
        regency_selector.enable()
        callback(results)
    },
    error: function () {
        callback()
    }
})
})

$regency_selector = $('#regency_selector').selectize({
    sortField: 'name',
    valueField: 'id',
    labelField: 'name',
    searchField: ['name']
})

province_selector = $province_selector[0].selectize
regency_selector = $regency_selector[0].selectize
})
})

```

Berikut penjelasan sederhana script diatas:

- Pertama kita tentukan apakah pada halaman yang sedang tampil ada element dengan id `province_selector`. Id ini akan kita tambahkan pada dropdown untuk provinsi
- Setelah ditemukan, kita modifikasi event `selectize` pada element tersebut.
- Ketika isiannya berubah dan tidak ada provinsi yang dipilih, kita hapus semua pilihan pada dropdown untuk kabupaten dan disable element tersebut.
- Ketika memilih provinsi yang lain, kita melakukan request ke `/address/regencies?province_id=` dengan parameter id provinsi yang terpilih.
- Setelah data didapatkan, kita aktifkan kembali dropdown untuk memilih dan mengisi dengan hasil yang kita dapatkan dari server.

Setelah semua syntax diatas dijalankan, compile kembali dengan

```
gulp
```

Agar fitur ini berfungsi, kita perlu menambahkan id yang sesuai ke dropdown Provinsi dan Kabupaten

resources/views/checkout/_address-field.blade.php

```
....  
{!! Form::select('province_id', [''=>'']+DB::table('provinces')->lists('name','id'), null, ['class'=>'form-control']) !!}  
{!! Form::select('province_id', [''=>'']+DB::table('provinces')->lists('name','id'), null, ['class'=>'form-control', 'id' => 'province_selector']) !!}  
....  
{!! Form::select('regency_id', [], old('regency_id'), ['class'=>'form-control']) !!}  
{!! Form::select('regency_id',  
    old('province_id') !== null ? DB::table('regencies')->where('province_id',\br/>    old('province_id'))->lists('name', 'id') : [],  
    old('regency_id'), ['class'=>'form-control', 'id' => 'regency_selector']) \br/>!!}  
....
```

Sedikit tambahan, pada isian untuk Kabupaten diatas kita buat agar pilihan yang muncul di set ke Kabupaten jika ditemukan `old('province_id')`. Ini kita butuhkan ketika controller gagal melakukan validasi. Sehingga isian untuk Kabupaten tidak kosong.

Cobalah fitur ini dengan mengubah isian provinsi dan menghapus isian provinsi. Pastikan kondisi isian Kabupaten berubah sesuai penjelasan diatas.

The screenshot shows a web application interface for a checkout process. At the top, there are tabs for 'Login', 'Alamat' (which is highlighted in blue), and 'Pembayaran'. On the left, a form titled 'Alamat Pengiriman' contains fields for 'Nama' (Name), 'Alamat' (Address), 'Provinsi' (Province) set to 'Bali', and 'Kabupaten / Kota' (District/City). A dropdown menu is open under 'Kabupaten / Kota', showing options like 'Kota Denpasar', 'Kabupaten Buleleng', 'Kabupaten Gianyar', etc. On the right, a 'Cart' section displays a table with items and their details:

Produk	Jumlah	Harga
Nike Air Max	20	420,000
Subtotal		Rp8,400,000
Nike Aeroloft Bomber	30	720,000
Subtotal		Rp21,600,000
Total		Rp30,000,000

Dropdown alamat

Memproses Form

Untuk memudahkan kita dalam memproses form alamat ini, kita akan membuat form request dengan nama `CheckoutAddressRequest`:

```
php artisan make:request CheckoutAddressRequest
```

Pada file yang dihasilkan kita isi dengan syntax berikut:

app/Http/Requests/CheckoutAddressRequest.php

```
<?php

namespace App\Http\Requests;

use App\Http\Requests\Request;
use Auth;

class CheckoutAddressRequest extends Request
{
```

```

public function authorize()
{
    return true;
}

public function rules()
{
    $new_address_rules = [
        'name' => 'required',
        'detail' => 'required',
        'province_id' => 'required|exists:provinces,id',
        'regency_id' => 'required|exists:regencies,id',
        'phone' => 'required|digits_between:9,15'
    ];

    return $new_address_rules;
}

```

Tidak ada yang spesial di rule validasi ini. Kita hanya memastikan semua field terisi, isian provinsi dan kabupaten valid dan isian nomor telepon valid.

Kita lanjut ke controller, buat method seperti ini:

app/Http/Controllers/CheckoutController.php

```

...
use App\Http\Requests\CheckoutAddressRequest;
use Auth;

class CheckoutController extends Controller
{
    ...

    public function postAddress(CheckoutAddressRequest $request)
    {
        if (Auth::check()) return $this->authenticatedAddress($request);
        return $this->guestAddress($request);
    }
}

```

Pada method ini, kita memanggil method yang berbeda ketika user belum atau sudah login. Untuk method ketika user sudah login, sementara kita buat dulu seperti ini:

app/Http/Controllers/CheckoutController.php

```
protected function authenticatedAddress(CheckoutAddressRequest $request)
{
    return "Akan diisi untuk logic authenticated address";
}
```

Nah, untuk method ketika user belum login, kita isi seperti ini:

app/Http/Controllers/CheckoutController.php

```
protected function guestAddress(CheckoutAddressRequest $request)
{
    $this->saveAddressSession($request);
    return redirect('checkout/payment');
}
```

Oke, diatas terlihat kita memanggil method `saveAddressSession` untuk menyimpan data session alamat. Ini syntaxnya:

app/Http/Controllers/CheckoutController.php

```
protected function saveAddressSession(CheckoutAddressRequest $request)
{
    session([
        'checkout.address.name' => $request->get('name'),
        'checkout.address.detail' => $request->get('detail'),
        'checkout.address.province_id' => $request->get('province_id'),
        'checkout.address.regency_id' => $request->get('regency_id'),
        'checkout.address.district_id' => $request->get('district_id'),
        'checkout.address.phone' => $request->get('phone')
    ]);
}
```

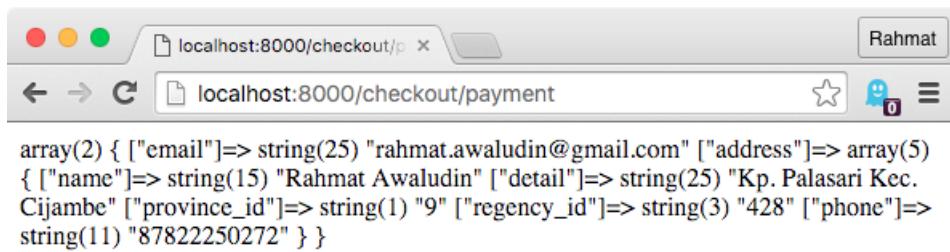
Pada method ini, kita simpan semua data user yang dikirim dengan nama session `checkout.address.*`.

Terlihat pada methdo `guestAddress`, kita mengarahkan user ke halaman `checkout/payment`. Mari kita buat route ini dan tampilkan data alamat yang telah dikirim user:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {  
    ....  
    Route::get('checkout/payment', function() {  
        return var_dump(session()->get('checkout'));  
    });  
});
```

Ini tampilan yang akan kita dapatkan:

**Data session untuk alamat tersimpan****Tahap 3: Pembayaran**

Pada tahap pembayaran, kita akan meminta user untuk memilih bank yang akan dia gunakan untuk melakukan transfer. Kita akan file konfigurasi untuk menyimpan data bank yang bisa digunakan untuk melakukan transfer.

Menyiapkan konfigurasi Bank

File konfigurasi akan kita buat di config/bank-accounts.php. Berikut contohnya:

config/bank-accounts.php

```
<?php
return [
    'bca' => [
        'title' => 'BCA',
        'bank' => 'BCA',
        'name' => 'PT Reselia',
        'number' => '45454545'
    ],
    'bni' => [
        'title' => 'BNI',
        'bank' => 'BNI',
        'name' => 'PT Reselia',
        'number' => '787687678'
    ],
    'mandiri' => [
        'title' => 'Mandiri',
        'bank' => 'Mandiri',
        'name' => 'PT Reselia',
        'number' => '23223232'
    ],
    'atm-bersama' => [
        'title' => 'ATM Bersama',
        'bank' => 'Mandiri',
        'name' => 'PT Reselia',
        'number' => '23223232'
    ],
];
];
```

Format tiap bank pada file konfigurasi ini sebagai berikut:

```
'kode-bank' => [
    'title' => 'Text yang muncul di dropdown',
    'bank' => 'Nama Bank',
    'name' => 'Nama rekening',
    'number' => 'Nomor rekening'
]
```

Silahkan sesuaikan isian dari file konfigurasi ini dengan data yang diinginkan. Kita dapat mengecek konfigurasi ini dengan tinker:

```
php artisan tinker
>>> // Mengambil detail untuk BCA
>>> config('bank-accounts.bca')
=> [
    "title" => "BCA",
    "bank" => "BCA",
    "name" => "PT Reselia",
    "number" => "45454545",
]
```

Untuk memudahkan dalam membuat dropdown pemilihan bank, mari kita buat helper di app/Support/Helpers.php:

app/Support/Helpers.php

```
.....
/** Bank list for <select> element */
function bankList() {
    $result = [];
    foreach (config('bank-accounts') as $account => $detail) {
        $result[$account] = $detail['title'];
    }
    return $result;
}
```

Kita tes method ini dengan tinker:

```
php artisan tinker
>>> bankList()
=> [
    "bca" => "BCA",
    "bni" => "BNI",
    "mandiri" => "Mandiri",
    "atm-bersama" => "ATM Bersama",
]
```

Membuat form

Mari kita buat dulu route untuk menampilkan dan menerima form pembayaran ini:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::get('checkout/payment', function() {
        return var_dump(session()->get('checkout'));
    });
    Route::get('checkout/payment', 'CheckoutController@payment');
    Route::post('checkout/payment', 'CheckoutController@postPayment');
});
```

Kita buat method di controller:

app/Http/Controllers/CheckoutController.php

```
public function payment()
{
    return view('checkout.payment');
```

Kemudian viewnya:

resources/views/checkout/payment.blade.php

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

@section('content')


@include('checkout._step')


Pembayaran



@include('checkout._payment-form')



@include('checkout._cart-panel')


```

Pada view ini, kita menggunakan partial view `checkout._payment-form`. Berikut isinya:

`resources/views/checkout/_payment-form.blade.php`

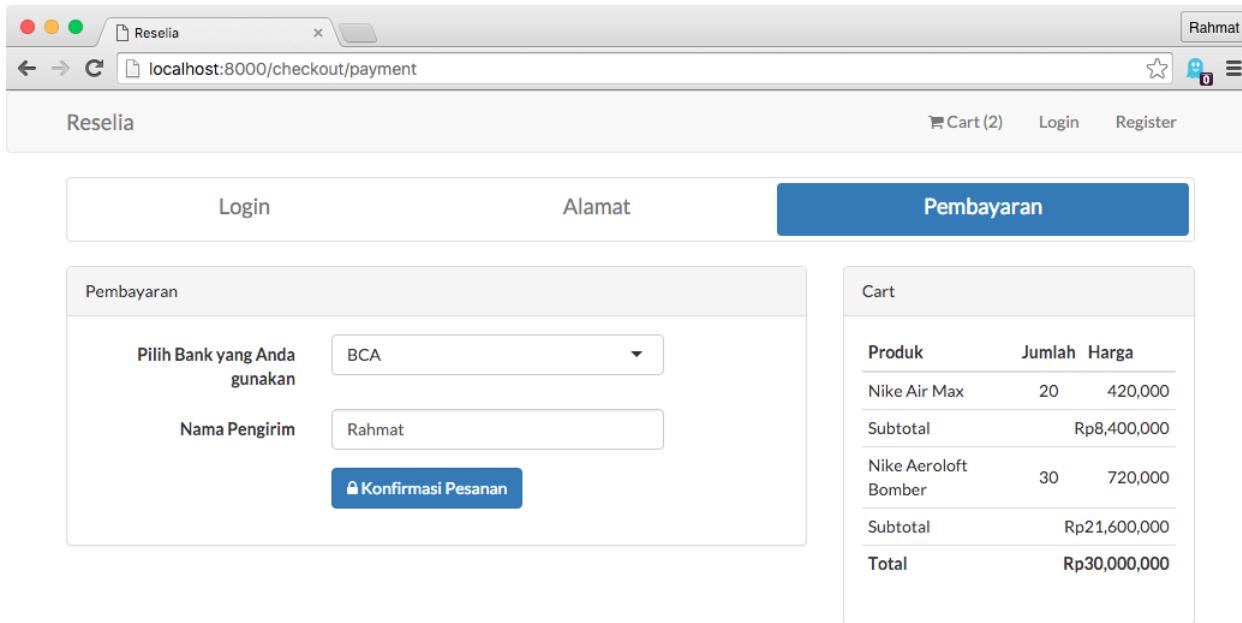
```
{!! Form::open(['url' => '/checkout/payment', 'method'=>'post', 'class' => 'form\horizonal']) !!}

<div class="form-group {!! $errors->has('bank_name') ? 'has-error' : '' !!}">
    {!! Form::label('bank_name', 'Pilih Bank yang Anda gunakan', ['class' => '\
col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::select('bank_name', bankList(), null, ['class'=>'form-control \
js-selectize']) !!}
        {!! $errors->first('bank_name', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group {!! $errors->has('sender') ? 'has-error' : '' !!}">
    {!! Form::label('sender', 'Nama Pengirim', ['class' => 'col-md-4 control-1\\
abel']) !!}
    <div class="col-md-6">
        {!! Form::text('sender', null, ['class'=>'form-control']) !!}
        {!! $errors->first('sender', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::button('<i class="fa fa-lock"></i> Konfirmasi Pesanan', ar\
ray('type' => 'submit', 'class' => 'btn btn-primary')) !!}
    </div>
</div>
{!! Form::close() !!}
```

Disini kita meminta user untuk memilih bank apa yang akan dia gunakan. Kita menggunakan helper `bankList()` yang telah kita buat sebelumnya untuk menampilkan daftar bank. Tampilan form ini akan seperti berikut:



Halaman Payment

Menampilkan ongkos kirim

Pada halaman pembayaran, kita telah mendapatkan alamat user. Disini, kita bisa menggunakan alamat tersebut untuk menambahkan ongkos kirim pada sidebar detail produk. Untuk membuat fitur ini, kita perlu menambah method baru di CartService untuk menghitung total ongkos kirim. Mari kita namai `shippingFee`:

```
/Users/rahmatawaludin/Code/reselia/app/Support/CartService.php
protected function getDestinationId()
{
    return session('checkout.address.regency_id');
}

public function shippingFee()
{
    $totalFee = 0;
    foreach ($this->lists() as $id => $quantity) {
        $fee = Product::find($id)->getCostTo($this->getDestinationId()) * $quantity;
        $totalFee += $fee;
    }
    return $totalFee;
}
```

Berikut penjelasan syntax diatas:

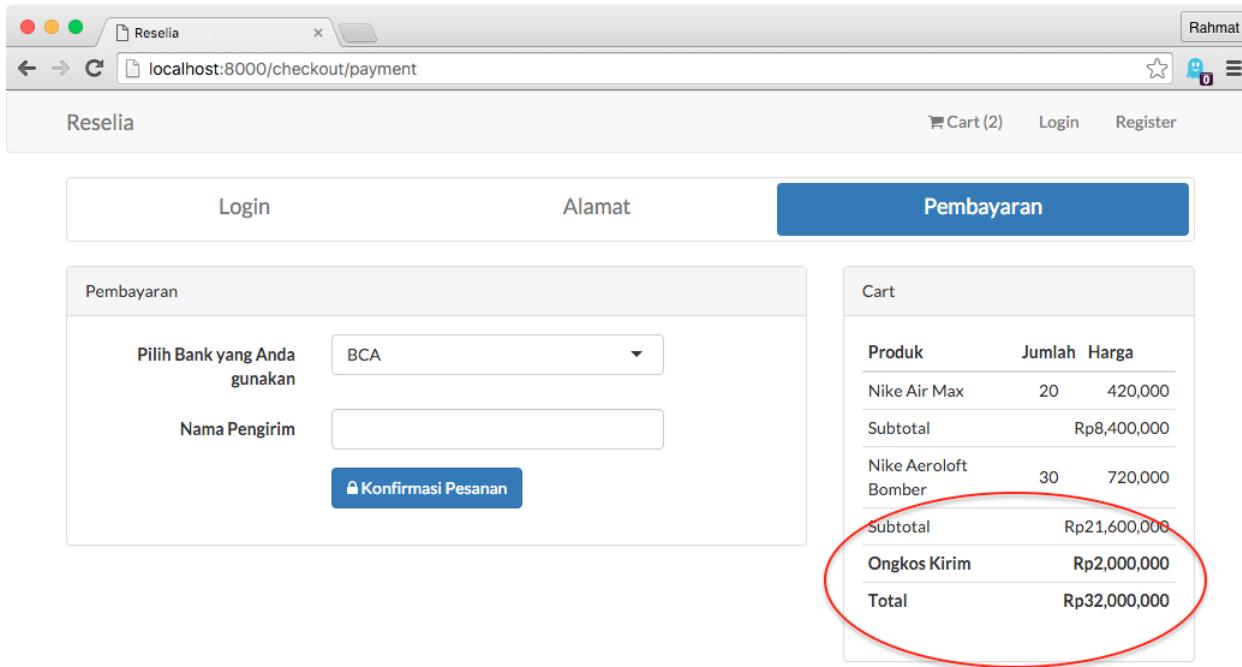
- Kita siapkan variable \$fee sebagai nilai default ongkos kirim yaitu 0.
- Pada method `shippingFee`, kita melakukan looping terhadap semua produk yang berada di cart.
- Untuk setiap produk yang ditemukan kita tentukan ongkos kirimnya dengan method `getCostTo()`. Method ini akan kita beri parameter dari `regency_id` yang diinput oleh user. Untuk memudahkan, kita buat method `getDestinationId` untuk mengambil `regency_id`.
- Hasil dari method `getCostTo` kita kalikan dengan quantity barang tersebut.
- Kita tambahkan ongkos kirim tersebut ke variable \$fee.
- Terakhir, kita kembalikan nilai dari variable \$fee.

Untuk menggunakan total ongkos kirim ini pada halaman payment, kita ubah footer dari `resources/views/checkout/_cart-panel.blade.php` menjadi:

`resources/views/checkout/_cart-panel.blade.php`

```
....  
<tr>  
  <td><strong>Total</strong></td>  
  <td class="text-right" colspan="2"><strong>Rp{{ number_format($cart->totalPrice\(  
e()) )}}</strong></td>  
</tr>  
  
@if(request()->is('checkout/payment'))  
  <tr>  
    <td data-th="Subtotal"><strong>Ongkos Kirim</strong></td>  
    <td data-th="Subtotal" class="text-right" colspan="2"><strong>Rp{{ number fo\brmat($cart->shippingFee()) }}</strong></td>  
  </tr>  
  <tr>  
    <td><strong>Total</strong></td>  
    <td class="text-right" colspan="2"><strong>Rp{{ number_format($cart->totalPr\ice() + $cart->shippingFee()) }}</strong></td>  
  </tr>  
@else  
  <tr>  
    <td><strong>Total</strong></td>  
    <td class="text-right" colspan="2"><strong>Rp{{ number_format($cart->totalPric\ee()) }}</strong></td>  
  </tr>  
@endif  
....
```

Kini, tampilan halaman pembayaran akan seperti berikut:



Menampilkan ongkos kirim

Sebagai latihan, cobalah tampilkan alamat yang digunakan user untuk pengiriman pada halaman ini.

Memproses form

Mari kita buat logic untuk memproses form yang telah kita buat. Sesuai isian pada routing, kita akan membuat method `postPayment` pada `CheckoutController`:

app/Http/Controllers/CheckoutController.php

```
public function postPayment(Request $request)
{
    $this->validate($request, [
        'bank_name' => 'required|in:' . implode(',', array_keys(config('bank-accounts'))),
        'sender' => 'required'
    ]);

    session([
        'checkout.payment.bank' => $request->get('bank_name'),
        'checkout.payment.sender' => $request->get('sender')
    ]);
}
```

```
if (Auth::check()) return $this->authenticatedPayment($request);
return $this->guestPayment($request);
}
```

Hal pertama yang dilakukan syntax diatas adalah validasi terhadap data yang dikirim user. Yang perlu diperhatikan dari validansi diatas adalah cara kita melakukan validasi untuk isian `bank_name`. Kita menggunakan rule `required` dan `in`. Rule `in`, harus diisi dengan string berisi nilai-nilai yang diizinkan. String tersebut harus dibatasi dengan koma. Itulah sebabnya kita menggunakan syntax:

```
implode(' ', array_keys(config('bank-accounts')))
```

Dengan menggunakan konfigurasi yang kita miliki sekarang, kita akan mendapatkan hasil seperti ini (test dengan tinker):

```
"bca,bni,mandiri,atm-bersama"
```

Tahapan selanjutnya, kita menyimpan data pembayaran ke session. Terakhir, kita arahkan request ke method yang sesuai untuk user yang sudah maupun belum login. Untuk method ketika user sudah login, sementara kita buat dulu seperti berikut:

app/Http/Controllers/CheckoutController.php

```
protected function authenticatedPayment(Request $request)
{
    return "akan diisi dengan logic authenticated payment";
}
```

Nah, untuk method ketika user belum login, akan seperti ini:

app/Http/Controllers/CheckoutController.php

```
...
use App\Address;
use App\Product;
use App\Support\CartService;
use App\Order;
use App\OrderDetail;

class CheckoutController extends Controller
{
```

```
protected $cart;

public function __construct(CartService $cart)
{
    $this->cart = $cart;
}

.....

protected function guestPayment(Request $request)
{
    // create user account
    $user = $this->setupCustomer(session('checkout.email'), session('checkout.address.name'));

    // create address
    $bank = session('checkout.payment.bank');
    $sender = session('checkout.payment.sender');
    $address = $this->setupAddress($user, session('checkout.address'));

    // create record
    $order = $this->makeOrder($user->id, $bank, $sender, $address, $this->cart->details());

    // delete session data
    session()->forget('checkout');
    $deleteCartCookie = $this->cart->clearCartCookie();
    return redirect('checkout/success')->with(compact('order'))
        ->withCookie($deleteCartCookie);
}
```

Ada empat tahap yang kita lakukan lakukan pada method ini:

1. Membuat akun untuk user tanpa password
2. Membuat alamat untuk user
3. Membuat record untuk order
4. Pembersihan (menghapus data session & cookie) dan mengarahkan user ke halaman sukses.

Untuk proses pembuatan akun, kita akan memanggil method `setupCustomer` dengan parameter email dan nama user. Berikut syntaxnya:

app/Http/Controllers/CheckoutController.php

```
protected function setupCustomer($email, $name)
{
    $user = User::create(compact('email', 'name'));
    $user->role = 'customer';
    $user->save();
    return $user;
}
```

Pada method ini kita membuat record baru di table user, tapi hanya mengisi field email, name dan role (customer). Pengisian field role sengaja kita pisahkan karena field ini tidak kita aktifkan pada fitur mass assignment.

Untuk membuat order, kita juga harus membuat alamat. Kita menggunakan method setupAddress dengan parameter instance dari User dan data alamat dari session. Berikut syntaxnya:

app/Http/Controllers/CheckoutController.php

```
.....
use App\Address;

class CheckoutController extends Controller
{
    .....
    protected function setupAddress(User $customer, $addressSession)
    {
        return Address::create([
            'user_id' => $customer->id,
            'name' => $addressSession['name'],
            'detail' => $addressSession['detail'],
            'regency_id' => $addressSession['regency_id'],
            'phone' => $addressSession['phone']
        ]);
    }
}
```

Selanjutnya, kita buat record order untuk menyimpan data pesanan user. Untuk menyimpan order, kita akan menggunakan dua table:

- orders: Untuk menyimpan data user, status, bank yang digunakan, nama pengirim untuk pembayaran, total jumlah pembayaran dan tanggal pesanan. Total

pembayaran sengaja kita tulis lagi disini (mesti kita bisa gunakan query). Ini dimaksudkan untuk mengurangi kalkulasi terhadap berbagai field di database. Sehingga aplikasi kita lebih cepat. Untuk isian status akan kita jelaskan lebih lanjut di pembahasan selanjutnya.

- `order_details`: Untuk menyimpan detail dari produk pada pesanan tersebut. Table ini akan menyimpan id pesanan, id produk, jumlah order, harga ketika melakukan pemesanan, ongkos kirim dan total harga. Harga produk sengaja kita catat lagi disini agar kita tetap menggunakan harga produk ketika dijual meskipun harga di table `products` telah berubah. Total harga disini kita gunakan untuk mencatat (`harga + ongkos kirim`) x jumlah. Seperti total pembayaran di table `orders`, kita sengaja menyimpan total harga mengurangi jumlah kalkulasi.

Mari kita buat model dan migration yang dibutuhkan:

```
php artisan make:model Order -m
php artisan make:model OrderDetail
```

Pada file migration yang digenerate isi method `up` dengan syntax berikut:

database/migrations/2016_03_08_033605_create_orders_table.php

```
public function up()
{
    Schema::create('orders', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('user_id')->unsigned();
        $table->integer('address_id')->unsigned();
        $table->string('status')->default('waiting-payment');
        $table->string('bank');
        $table->string('sender');
        $table->decimal('total_payment', 18, 2);
        $table->timestamps();

        $table->foreign('user_id')->references('id')->on('users');
        $table->foreign('address_id')->references('id')->on('addresses');
    });

    Schema::create('order_details', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('order_id')->unsigned();
        $table->integer('product_id')->unsigned();
        $table->integer('quantity')->unsigned();
    });
}
```

```




---



```

Pada method `down` isi dengan syntax berikut:

database/migrations/2016_03_08_033605_create_orders_table.php

```

public function down()
{
    Schema::drop('order_details');
    Schema::drop('orders');
}

```

Selanjutnya, kita terapkan kedua migration tersebut ke database:

`php artisan migrate`

Untuk model Order, kita isi seperti berikut:

app/Order.php

```

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    protected $fillable = ['id', 'user_id', 'address_id', 'bank', 'sender',
        'status', 'total_payment'];

    public function user()
    {
        return $this->belongsTo('App\User');
    }
}

```

```
}

public function details()
{
    return $this->hasMany('App\OrderDetail');
}

public function address()
{
    return $this->belongsTo('App\Address');
}

}
```

Disini, kita mengaktifkan mass assignment dan menambahkan relasi ke user, details (OrderDetail) dan address.

Untuk model OrderDetail, kita buat seperti ini:

app/OrderDetail.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class OrderDetail extends Model
{
    protected $table = 'order_details';

    protected $fillable = ['id', 'order_id', 'product_id', 'quantity', 'price',
        'fee', 'total_price'];

    public function product()
    {
        return $this->belongsTo('App\Product');
    }

    public function order()
    {
        return $this->belongsTo('App\Order');
    }
}
```

Di model ini, kita mengaktifkan mass assignment dan menambah relasi ke Order dan Product.

Ketika model OrderDetail telah disimpan, kita akan otomatis menghitung field `total_price`. Untuk melakukannya, kita akan membuat method `refreshTotalPrice` dan menggunakan event `saved` untuk memanggilnya. Syntaxnya akan seperti ini:

app/OrderDetail.php

```
....  
public static function boot() {  
    parent::boot();  
  
    static::saved(function($model) {  
        if ($model->total_price < 1) {  
            $model->refreshTotalPrice();  
        }  
    });  
}  
  
public function refreshTotalPrice()  
{  
    $total_price = ($this->price + $this->fee) * $this->quantity;  
    $this->total_price = $total_price;  
    $this->save();  
}  
....
```

Disini kita menggunakan pengecekan apakah isian `total_price` belum terisi, jika belum baru kita jalankan `refreshTotalPrice`. Jika kita tidak melakukan pengecekan ini, akan terjadi *infinite looping* di event `saved` yang akan menyebabkan aplikasi kita *crash*.

Kita juga akan otomatis mengupdate isian `total_payment` pada model Order ketika OrderDetail berhasil disimpan. Untuk melakukannya, kita akan membuat method `refreshTotalPayment` di model Order dan memanggilnya dari event `saved` pada model OrderDetail. Kita buat dulu methodnya:

app/Order.php

```
public function refreshTotalPayment()
{
    $total_payment = 0;
    foreach($this->details as $detail) {
        $total_payment += $detail->total_price;
    }
    $this->total_payment = $total_payment;
    $this->save();
}
```

Kita panggil dari event `saved` pada model OrderDetail:

app/OrderDetail.php

```
.....
public static function boot() {
    parent::boot();

    static::saved(function($model) {
        if ($model->total_price < 1) {
            $model->refreshTotalPrice();
        }
        $model->order->refreshTotalPayment();
    });
}
....
```

Oke, sekarang kita buat method `makeOrder` di `CheckoutController`:

app/Http/Controllers/CheckoutController.php

```
.....
use App\Product;
use App\Order;
use App\OrderDetail;

class CheckoutController extends Controller
{
    .....
    protected function makeOrder($user_id, $bank, $sender, Address $address, $cart)
}
```

```

{
    $status = 'waiting-payment';
    $address_id = $address->id;
    $order = Order::create(compact('user_id', 'address_id', 'bank', 'sender' \
, 'status'));
    foreach ($cart as $product) {
        OrderDetail::create([
            'order_id' => $order->id,
            'address_id' => $address->id,
            'product_id' => $product['id'],
            'quantity' => $product['quantity'],
            'price' => $product['detail']['price'],
            'fee' => Product::find($product['id'])->getCostTo($address->rege\
ncy_id)
        ]);
    }

    return Order::find($order->id);
}
}

```

Berikut penjelasan syntax ini:

- Kita membuat status order ini sebagai `waiting-payment`. Status ini menandakan bahwa order ini masih menunggu pembayaran dari customer. Penjelasan lengkap untuk status lainnya akan kita bahas di pembahasan selanjutnya.
- Kita membuat record order dengan memberikan parameter yang dibutuhkan.
- Untuk setiap data yang ada di cart, kita buatkan record pada table `order_details`. Sebagaimana telah kita siapkan sebelumnya, penambahan ini akan otomatis mengupdate nilai `total_payment` pada model Order yang telah kita buat.
- Terakhir, pada saat mengembalikan order kita melakukan query kembali terhadap order tersebut. Ini kita lakukan untuk *me-refresh* data order tersebut karena telah dirubah ketika kita membuat Order Detail.

Tahap terakhir dari method `guestPayment` adalah membersihkan data session dan cookie. Kemudian mengarahkan user ke halaman terakhir.

Untuk menghapus data session `checkout`, ini syntax yang kita gunakan:

```
session()->forget('checkout');
```

Untuk menghapus cookie, kita menggunakan syntax ini:

```
$deleteCartCookie = $this->cart->clearCartCookie();
```

Disini kita menggunakan method `clearCartCookie` pada `CartService`. Syntax method ini akan seperti berikut:

app/Support/CartService.php

```
public function clearCartCookie()
{
    return Cookie::forget('cart');
}
```

Agar cookie ini terhapus, kita perlu menambahkan hasil method ini ke request selanjutnya:

```
return redirect('checkout/success')->with(compact('order'))
    ->withCookie($deleteCartCookie);
```

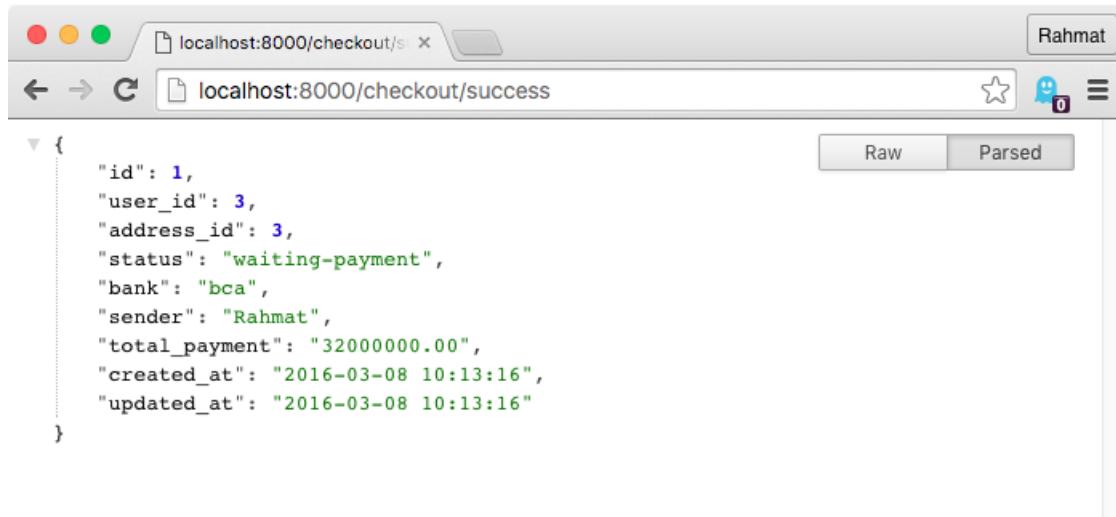
Disini, kita mengirimkan penghapusan cookie dengan menggunakan method `withCookie`. Pada saat kita mengirimkan user ke halaman `checkout/success`, kita juga membuat flash data dengan nama `order` yang akan kita gunakan untuk menampilkan data order yang berhasil dibuat.

Untuk mengetes fitur ini, kita dapat membuat route seperti berikut:

app/Http/routes.php

```
Route::get('checkout/success', function() {
    return session()->get('order');
});
```

Oke, mari kita test. Ikuti semua langkah checkout dan ketika terakhir klik pada "Konfirmasi Pembayaran" akan muncul tampilan seperti berikut:



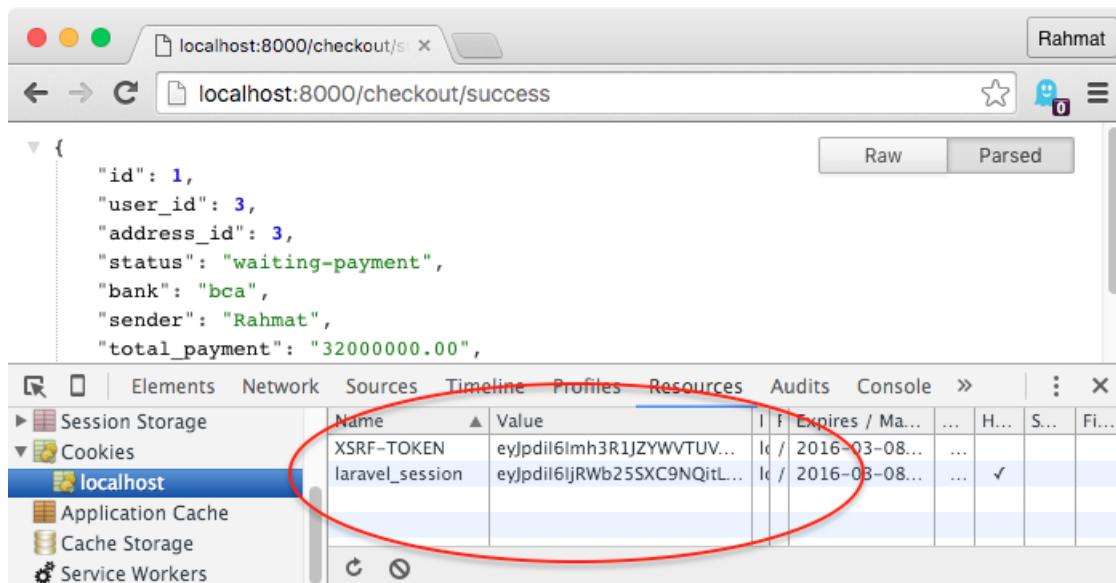
A screenshot of a web browser window titled "localhost:8000/checkout/success". The page content is a JSON object representing an order:

```
{
  "id": 1,
  "user_id": 3,
  "address_id": 3,
  "status": "waiting-payment",
  "bank": "bca",
  "sender": "Rahmat",
  "total_payment": "32000000.00",
  "created_at": "2016-03-08 10:13:16",
  "updated_at": "2016-03-08 10:13:16"
}
```

The "Parsed" tab is selected at the top right.

Order berhasil dibuat

Pastikan data cookie untuk cart juga sudah dihapus:

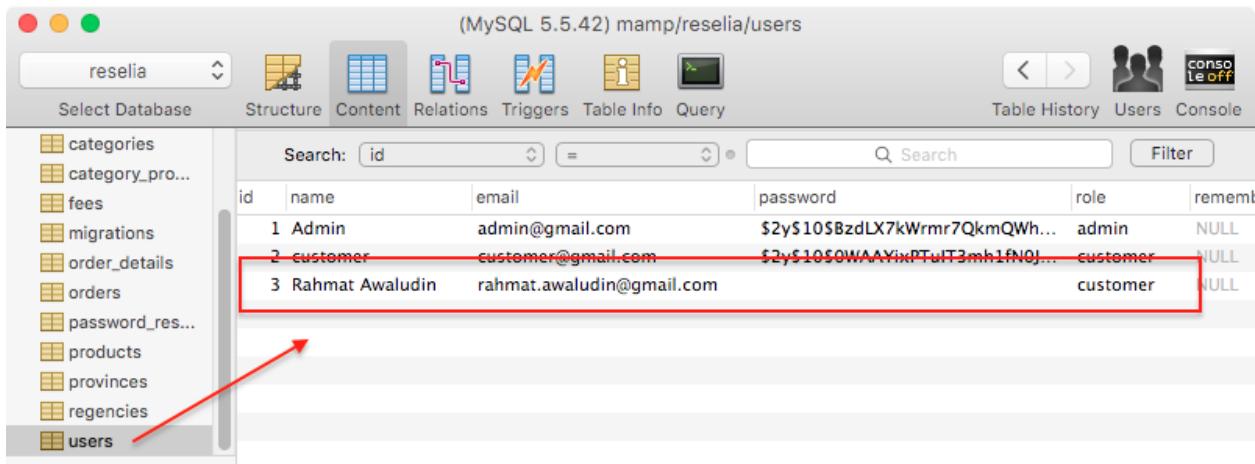


A screenshot of the developer tools resources panel, specifically the "Cookies" section for the "localhost" domain. A red oval highlights the "laravel_session" cookie entry:

Name	Value	Expires / Ma...	H...	S...	Fi...
XSRF-TOKEN	eyJpdi...	2016-03-08...			
laravel_session	eyJpdil...	2016-03-08...	✓		

Data cookie untuk cart dihapus

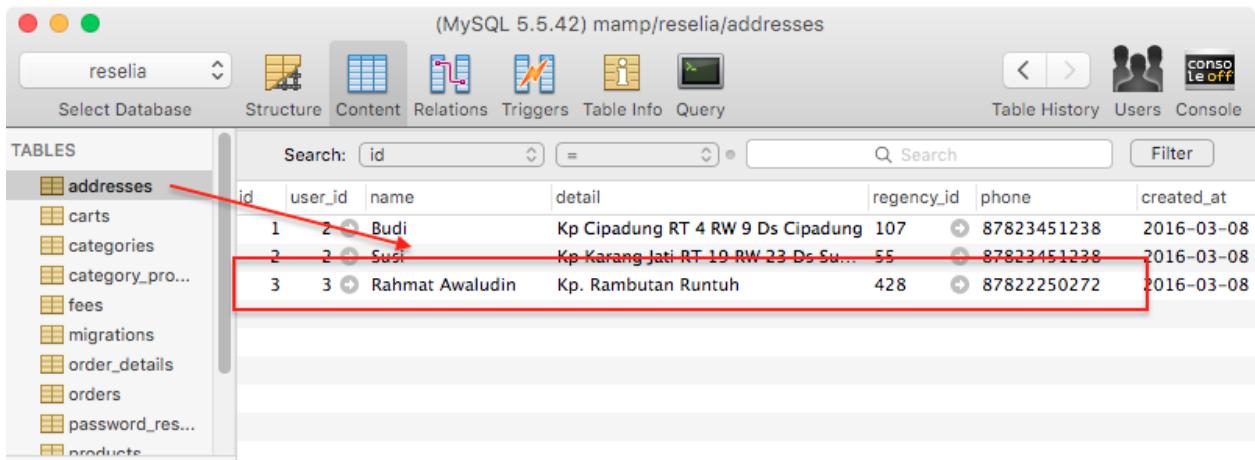
Di database, akan muncul akun customer baru tanpa password dengan email yang kita gunakan:



	id	name	email	password	role	remember_token
1	Admin		admin@gmail.com	\$2y\$10\$BzdLX7kWrmr7QkmQWh...	admin	NULL
2	customer		customer@gmail.com	\$2y\$10\$50WAAYixPTuit3mh1fN0j...	customer	NULL
3	Rahmat Awaludin		rahmat.awaludin@gmail.com		customer	NULL

Akun user dibuat

Alamat untuk akun tersebut juga akan dibuat:



	id	user_id	name	detail	regency_id	phone	created_at
1	2	2	Budi	Kp Cipadung RT 4 RW 9 Ds Cipadung	107	87823451238	2016-03-08
2	2	2	Suci	Kp Karang Jati RT 10 RW 23 Ds Su...	55	87823451238	2016-03-08
3	3	3	Rahmat Awaludin	Kp. Rambutan Runtuh	428	87822250272	2016-03-08

Alamat user

Order juga akan dibuat:

Data Order

	id	user_id	address_id	status	bank	sender	total_payment	created_at	updated_at
	1	3	3	waiting-payment	bca	Rahmat	32000000.00	2016-03-08 10:13:16	2016

Berikut detail dari order tersebut:

Detail Order

	id	order_id	product_id	quantity	price	fee	total_price	created_at	updated_at
	1	1	2	20	420000.00	40000.00	9200000.00	2016-03-08 10:13:16	2016
	2	1	4	30	720000.00	40000.00	22800000.00	2016-03-08 10:13:16	2016

Huft.. Topik ini panjang banget. Saya aja yang nulis kelelahan. Santai aja, istirahat dulu. Kalau udah siap, lanjut ke bagian selanjutnya.

Tahap 4: Sukses

Bagian terakhir dari langkah setup ini adalah menampilkan halaman sukses. Seperti dijelaskan sebelumnya, kita mendapatkan flash data bernama `order` pada halaman terakhir ini. Kita dapat menggunakan data ini untuk menampilkan detail dari order yang telah dibuat oleh user. Mari kita mulai dengan membuat routenya:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::get('checkout/success', function() {
        return session()->get('order');
    });
    Route::get('checkout/success', 'CheckoutController@success');
});
```

Kita siapkan method di controller:

app/Http/Controllers/CheckoutController.php

```
public function success()
{
    return view('checkout.success');
```

Dan, viewnya:

resources/views/checkout/success.blade.php

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

@section('content')


### Berhasil!



Hi {{ session('order')->user->name }},



Terima kasih telah berbelanja di Reselia.



Untuk melakukan pembayaran dengan {{ config('bank-accounts')[session('order')->bank]['title'] }}:



- Silahkan transfer ke rekening {{ config('bank-accounts')[session('order')->bank]['bank'] }} {{ config('bank-accounts')[session('order')]}}


```

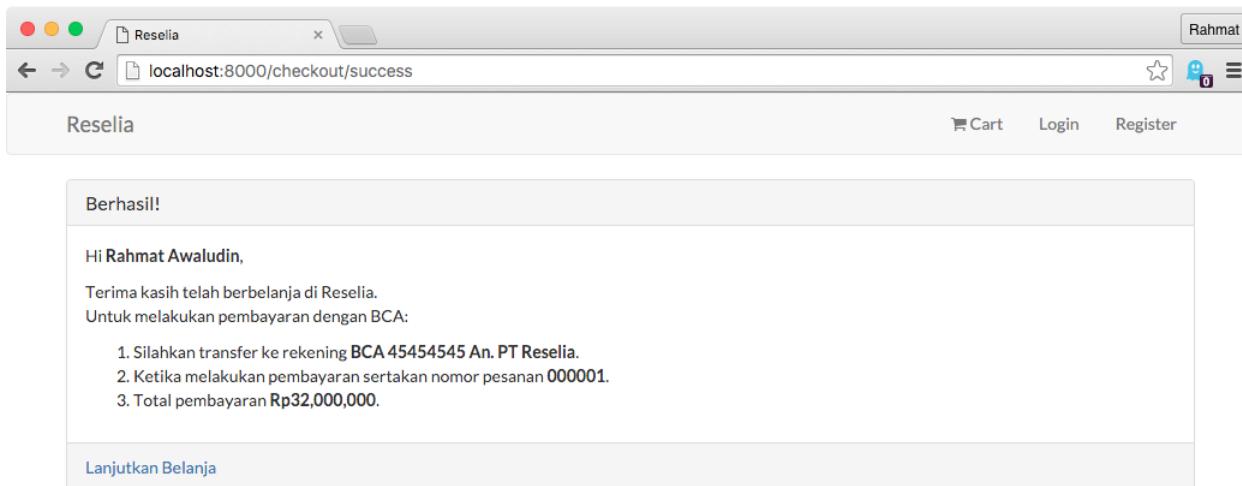
```
)->bank]['number']]}} An. {{ config('bank-accounts')[session('order')->bank]['na\\me']]}}</strong>.</li>
    <li>Ketika melakukan pembayaran sertakan nomor pesanan <strong>{{ \\\nsession('order')->padded_id }}</strong>.</li>
        <li>Total pembayaran <strong>Rp{{ number_format(session('order')->\\ntotal_payment)}}</strong>.</li>
    </ol>
</div>
<div class="panel-footer"><a href="/">Lanjutkan Belanja</a></div>
</div>
</div>
</div>
</div>
</div>
@endsection
```

Pada view ini, kita menampilkan berbagai data dari order yang telah dibuat oleh user. Menggunakan `session('order')`, kita akan mendapatkan instance dari order yang telah dibuat oleh User. Untuk mendapatkan detail dari metode pembayaran kita menggunakan `config['bank-accounts'][session('order')->bank]` dan mengambil field yang dibutuhkan. Terakhir, agar tampilan dari nomor order lebih menarik, kita menambahkan *zero fill*. Ini dapat dilihat dari custom accessor yang kita panggil `session('order')->padded_id`. Mari kita buat custom accessor ini:

app/Order.php

```
public function getPaddedIdAttribute()
{
    return str_pad($this->id, 6, 0, STR_PAD_LEFT);
}
```

Tampilan dari halaman ini akan seperti berikut:



Halaman sukses

Sebagai latihan, cobalah tampilkan semua produk yang telah dipesan oleh customer berikut alamat yang dia gunakan pada halaman ini. Coba juga mengirim email berisi detail dari order nya.

Checkout untuk Authenticated

Proses pembuatan checkout untuk user yang telah login akan lebih mudah karena kita telah banyak mengerjakan fitur-fitur dasar untuk pembuatan order pada pembahasan sebelumnya. Mari kita mulai dengan membuat logic untuk halaman login.

Tahap 1: Login

Ada dua hal yang akan kita lakukan pada tahap ini:

1. Jika user sudah login, maka kita akan lewati halaman ini dan lanjut ke halaman selanjutnya.
2. Jika user melakukan login dari halaman ini, kita akan *merge* data cart yang dimilikinya dengan data di database.

Untuk logic pertama, kita ubah method `login` pada `CheckoutController` menjadi:

app/Http/Controllers/CheckoutController.php

```
public function login()
{
    if (Auth::check()) {
        return redirect('/checkout/address');
    } else {
        return view('checkout.login');
    }
}
```

Untuk logic kedua, kita ubah method authenticatedCheckout menjadi:

app/Http/Controllers/CheckoutController.php

```
protected function authenticatedCheckout($email, $password)
{
    //return 'logic untuk authenticated checkout belum dibuat';

    protected function authenticatedCheckout($email, $password)
    {
        // login
        if (!Auth::attempt(['email' => $email, 'password' => $password])) {
            // Authentication failed..
            $errors = new MessageBag();
            $errors->add('email', 'Data user yang dimasukan salah');
            return redirect('checkout/login')
                ->withInput(compact('email', 'password') + ['is_guest' => 0])
                ->withErrors($errors);
        }

        // logged in, merge cart (destroy cart cookie)
        $deleteCartCookie = $this->cart->merge();
        return redirect('checkout/address')->withCookie($deleteCartCookie);
    }
}
```

Berikut penjelasan untuk syntax diatas:

- Pertama kita mengecek apakah kita bisa login dengan email dan password yang dikirim user.

- Jika gagal login, kita arahkan kembali user ke form tahapan login. Dan menambahkan pesan error custom pada field `email`. Ini kita lakukan dengan membuat instance dari `MessageBag` secara manual. Tentunya, kita juga mengirim kembali isian email, password dan `is_guest` dengan isian 0.
- Jika berhasil login, kita gabungkan data cart user di cookie dengan data di database menggunakan method `merge` pada `CartService`. Method ini telah kita bahas pada pembahasan sebelumnya.
- Terakhir, kita arahkan user ke halaman `checkout/address`.

Untuk mencoba fitur ini, kita dapat melakukan dua hal berikut:

- Login sebagai customer, tambah barang ke cart, kunjungi halaman `checkout`. Maka kita akan langsung ke tahapan membuat alamat.
- Sebagai guest, tambah barang ke cart. Di tahapan login, pilih "Saya adalah pelanggan tetap". Masukan email dan password. Pastikan data cart di cookie dan database telah tergabung pada tahapan selanjutnya.

Tahap 2: Alamat

Untuk user yang telah login, kita akan membuat sedikit perubahan pada tahap halaman. Disini kita akan mengizinkan user untuk memilih alamat yang telah dia gunakan sebelumnya. Tentunya, kita juga tetap mengizinkan user untuk membuat alamat baru.

Membuat form

Mari kita buat penyesuaian di viewnya:

`resources/views/checkout/address.blade.php`

```
....  
<div class="panel-body">  
    @include('checkout._address-new-form')  
  
    @if (auth()->check())  
        @include('checkout._address-choose-form')  
    @else  
        @include('checkout._address-new-form')  
    @endif  
</div>  
....
```

Disini, untuk user yang sudah login kita menggunakan partial view `checkout._address-new-form`. Berikut isinya:

resources/views/checkout/_address-choose-form.blade.php

```
{!! Form::open(['url' => '/checkout/address', 'method'=>'post', 'class' => 'form\horizonal']) !!}

<div class="form-group {!! $errors->has('address_id') ? 'has-error' : '' !!}\">
">
    {!! Form::label('address_id', 'Pilih Alamat', ['class' => 'col-md-4 control-label']) !!}
    <div class="col-md-6">
        @foreach (auth()->user()->addresses as $address)
            <div class="row">
                <div class="col-md-1">
                    <label>
                        {!! Form::radio('address_id', $address->id, null) !!}
                    </label>
                </div>
                <div class="col-md-11">
                    <address>
                        <strong>{{ $address->name }}</strong> <br>
                        {{ $address->detail }} <br>
                        {{ $address->regency->name }}, {{ $address->regency->province->name \}} <br>
                        <abbr title="Phone">P:</abbr> +62 {{ $address->phone }}</address>
                </div>
            </div>
        @endforeach
        <div class="row">
            <div class="col-md-1">
                <label>
                    {!! Form::radio('address_id', 'new-address', null) !!}
                </label>
            </div>
            <div class="col-md-11">
                <strong>Alamat Baru</strong>
            </div>
        </div>
        <div class="row">
            {!! $errors->first('address_id', '<p class="help-block">:message</p>') \ !!}
        </div>
    </div>
</div>
```

```
</div>

<div id="js-new-address">
    @include('checkout._address-field')
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::button('Lanjut <i class="fa fa-arrow-right"></i>', array('
type' => 'submit', 'class' => 'btn btn-primary')) !!}
    </div>
</div>

{!! Form::close() !!}
```

Berikut yang kita lakukan pada view ini:

- Kita melakukan looping untuk semua alamat user.
- Pada setiap alamat yang ditemukan kita membuat radio button dengan nama `address_id` dan isi id dari address tersebut.
- Di samping radio button tersebut, kita menampilkan detail dari alamat.
- Jika user hendak menambah alamat baru, kita berikan radio button `address_id` dengan nilai `new-address`.
- Dibawah radion button tersebut kita menampilkan detail field untuk membuat alamat baru dengan memanggil partial view `checkout._address-field`. Kita juga melakukan wrapping terhadap field tersebut dengan sebuah div yang memiliki id `js-new-address`. Ini kita lakukan untuk menyembunyikan field tersebut dengan javascript jika user tidak memilih untuk membuat alamat baru.

Agar looping untuk alamat diatas berfungsi, kita perlu menambah relasi dari User ke Address pada model User:

app/User.php

```
public function addresses()
{
    return $this->hasMany('App\Address');
```

Syntax javascript untuk menyembunyikan field alamat baru ketika user memilih alamat yang sudah ada sebagai berikut:

resources/assets/js/app.js

```
$(document).ready(function () {  
    ...  
    if ($('#input[name="address_id"]').length > 0) {  
        var selected = $('#input[name="address_id"]:checked').val()  
        if (selected === 'undefined' || selected !== 'new-address') {  
            $('#js-new-address').hide()  
        }  
  
        $('#input[name="address_id"]').change(function () {  
            var selected = $('#input[name="address_id"]:checked').val()  
            if (selected === 'new-address') {  
                $('#js-new-address').show()  
            } else {  
                $('#js-new-address').hide()  
            }  
        })  
    }  
})
```

Setelah menambahkan syntax diatas, compile kembali file javascript:

gulp

Tampilan dari form ini akan seperti berikut:

Reselia

localhost:8000/checkout/address

Rahmat

Reselia Home Cart (2) customer ▾

Login Alamat Pembayaran

Alamat Pengiriman

Pilih Alamat

Budi
Kp Cipadung RT 4 RW 9 Ds Cipadung
Kota Cimahi, Jawa Barat
P: +62 87823451238

Susi
Kp Karang Jati RT 19 RW 23 Ds
Sukamahi
Kota Bekasi, Jawa Barat
P: +62 87823451238

Alamat Baru

Lanjut ➔

Cart

Produk	Jumlah	Harga
Nike Air Force	10	340,000
Subtotal		Rp3,400,000
Nike Air Max	15	420,000
Subtotal		Rp6,300,000
Total		Rp9,700,000

Isian alamat baru tidak aktif

The screenshot shows a web application interface for a shopping cart. On the right, there is a 'Cart' section displaying a summary of items:

Produk	Jumlah	Harga
Nike Air Force	10	340,000
Subtotal		Rp3,400,000
Nike Air Max	15	420,000
Subtotal		Rp6,300,000
Total		Rp9,700,000

On the left, there is a 'Alamat Pengiriman' (Delivery Address) form. It includes a 'Pilih Alamat' section with two existing addresses (Budi and Susi) and a 'Alamat Baru' (New Address) section which is currently selected, indicated by a red arrow pointing to the radio button.

Isian alamat ditampilkan

Memproses form

Untuk memproses form kita perlu merubah validasi untuk melakukan validasi yang berbeda ketika user telah login. Berikut perubahannya:

app/Http/Requests/CheckoutAddressRequest.php

```
use Auth;

class CheckoutAddressRequest extends Request
{
    ...
    public function rules()
    {
        $new_address_rules = [
            'name' => 'required',
            'detail' => 'required',
            'province_id' => 'required|exists:provinces,id',
        ];
    }
}
```

```

'regency_id' => 'required|exists:regencies,id',
'phone' => 'required|digits_between:9,15'
];

if (Auth::check()) {
    $address_limit = implode(',', Auth::user()->addresses->lists('id')->\all()) . ',new-address';
    $rules = ['address_id' => 'required|in:' . $address_limit];
    if ($this->get('address_id') == 'new-address') {
        return $rules += $new_address_rules;
    }
    return $rules;
}

return $new_address_rules;
}

```

Pada perubahan ini kita mengecek apakah user sudah login, jika belum tentunya kita menggunakan validasi yang biasa. Jika user sudah login, berikut yang kita lakukan:

- Kita siapkan daftar address_id yang diizinkan. Kita menggunakan address yang dimiliki user ditambah string “new_address”. Untuk user customer@gmail.com, isian \$address_limit akan menjadi “1,2,new-address”.
- Kita siapkan rules untuk address_id yaitu required dan in (dengan parameter \$address_limit yang telah kita siapkan).
- Jika user memilih opsi new-address pada form, kita gabungkan rule untuk membuat alamat baru dengan rule yang baru dibuat.
- Terakhir kita berikan rules yang baru kita buat.

Selanjutnya, kita buat perubahan pada method authenticatedAddress:

app/Http/Controllers/CheckoutController.php

```

protected function authenticatedAddress(CheckoutAddressRequest $request)
{
    return "Akan diisi untuk logic authenticated address";
    $address_id = $request->get('address_id');
    // clear old
    session()->forget('checkout.address');
    if ($address_id == 'new-address') {
        $this->saveAddressSession($request);
    } else {

```

```

        session(['checkout.address.address_id' => $address_id]);
    }
    return redirect('checkout/payment');
}

```

Yang kita lakukan pada method ini sebagai berikut:

- Kita mengambil isian `address_id` yang dikirim.
- Kita hapus data session `checkout.address` yang lama. Ini kita lakukan disaat user klik back ke halaman alamat setelah mencapai halaman pembayaran. Tujuannya untuk menghindari menyimpan data alamat baru atau alamat lama jika pilihan alamat berubah antara alamat yang sudah ada atau alamat baru.
- Jika isian `address_id` adalah `new-address`, kita simpan data alamat ke session seperti biasa.
- Jika bukan, kita hanya menyimpan `address_id` tersebut ke session dengan nama `checkout.address.address_id`.
- Terakhir, kita arahkan user ke halaman `checkout/payment`.

Agar penghitungan ongkos kirim di tahap 3 tetap berjalan, kita harus melakukan penyesuaian di method `getDestinationId` pada `CartService`:

app/Support/CartService.php

```

.....
use App\Address;

class CartService {
    .....
    protected function getDestinationId()
    {
        if (Auth::check() && session()->has('checkout.address.address_id')) {
            $address = Address::find(session('checkout.address.address_id'));
            return $address->regency_id;
        }

        return session('checkout.address.regency_id');
    }
}

```

Dengan perubahan ini, ketika user sudah login dan memilih alamat yang sudah ada, kita akan mencari instance Address dari id yang dipilih tersebut. Dan mengembalikan `regency_id` nya.

Tahap 3: Pembayaran

Untuk tahap pembayaran tidak ada yang kita ubah pada form. Kita hanya cukup merubah isi dari method authenticatedPayment menjadi:

app/Http/Controllers/CheckoutController.php

```
protected function authenticatedPayment(Request $request)
{
    return "akan diisi dengan logic authenticated payment";

    $user = Auth::user();
    $bank = session('checkout.payment.bank');
    $sender = session('checkout.payment.sender');
    $address = $this->setupAddress($user, session('checkout.address'));
    $order = $this->makeOrder($user->id, $bank, $sender, $address, $this->cart->
details());
    // delete session data
    session()->forget('checkout');
    $this->cart->clearCartRecord();
    return redirect('checkout/success')->with(compact('order'));
}
```

Di method ini, kita masih menggunakan method setupAddress untuk mendapatkan instance dari Address user. Kita harus membuat sedikit penyesuaian pada method ini:

app/Http/Controllers/CheckoutController.php

```
protected function setupAddress(User $customer, $addressSession)
{
    if (Auth::check() && isset($addressSession['address_id'])) {
        return Address::find($addressSession['address_id']);
    }

    return Address::create([
        'user_id' => $customer->id,
        'name' => $addressSession['name'],
        'detail' => $addressSession['detail'],
        'regency_id' => $addressSession['regency_id'],
        'phone' => $addressSession['phone']
    ]);
}
```

Dengan perubahan ini, ketika user telah login dan pada session terdapat `session.address.address_id`, kita akan mencari instance dari Address berdasarkan id tersebut.

Yang kita lakukan disini hampir sama dengan method `guestPayment`. Hanya saja disini, proses pencarian instance dari User kita lakukan dengan mengambil instance user yang sedang login, bukan membuat baru.

Selanjutnya, kita membuat instance dari Address. Ini bisa dengan membuat alamat baru atau mengambil dari alamat yang ada.

Proses pembuatan order kita lakukan seperti biasa.

Terakhir, setelah menghapus data session cart, kita tidak melakukan penghapusan data cookie disini. Tapi menghapus data cart di database. Syntax ini:

```
$this->cart->clearCartRecord();
```

yang akan menghapusnya. Method `clearCartRecord` kita buat di CartService:

app/Support/CartService.php

```
public function clearCartRecord()
{
    return Cart::where('user_id', Auth::user()->id)->delete();
}
```

Yang dilakukan method ini adalah mencari record di table `carts` yang memiliki isian `user_id` sesuai dengan id dari user yang sedang login. Setelah ditemukan, semua record tersebut dihapus.

Terakhir, kita mengarahkan user ke halaman sukses dan menyertakan flash data `order` berisi order yang telah dibuat oleh user:

```
return redirect('checkout/success')->with(compact('order'));
```

Tahap 4: Sukses

Percaya atau tidak, tahap terakhir ini tidak perlu kita ubah sama sekali. Silahkan langsung dicoba.. :)

Melindungi Step Checkout dengan Middleware

Kita perlu membuat proteksi terhadap setiap tahapan checkout yang telah kita buat. Ini perlu kita lakukan untuk menghindari error di sistem. Sebagai contoh, saat ini meskipun kita tidak memiliki data di cart kita tetap dapat mengunjungi tahapan login di checkout:

The screenshot shows a web browser window titled "Reselia". The address bar displays "localhost:8000/checkout/login". The page content is a checkout form. At the top, there are three tabs: "Login" (which is highlighted in blue), "Alamat", and "Pembayaran". Below the tabs, there is a section for logging in or checking out without registering. It includes fields for "Email" and "Password", and two radio buttons for customer status: "Saya adalah pelanggan baru" (selected) and "Saya adalah pelanggan tetap". A link "Lupa kata sandi?" is also present. To the right of the login form is a "Cart" summary table with columns for "Produk", "Jumlah", and "Harga". The table shows one item: "Total" with "Rp0".

Mengakses checkout login tanpa data cart

Begitupun untuk tahapan alamat:

The screenshot shows a web browser window for the Reselia website. The URL in the address bar is `localhost:8000/checkout/address`. The page has a header with the Reselia logo, a user profile for 'Rahmat', and navigation links for 'Cart', 'Login', and 'Register'. Below the header, there are three tabs: 'Login' (disabled), 'Alamat' (selected and highlighted in blue), and 'Pembayaran'. The main content area is divided into two sections: 'Alamat Pengiriman' on the left and 'Cart' on the right. The 'Alamat Pengiriman' section contains fields for 'Nama', 'Alamat', 'Provinsi', 'Kabupaten / Kota', and 'Telepon', along with a 'Lanjut →' button. The 'Cart' section shows a table with columns 'Produk', 'Jumlah', and 'Harga'. A single row is present with 'Total' and 'Rp0'. The entire interface is styled with a clean, modern look.

Mengakses checkout address secara langsung

Disini, kita langsung mengunjungi alamat `checkout/address` tanpa melewati halaman login.

Untuk tahapan payment, ketika kita mencoba langsung mengakses `checkout/payment`, kita akan mendapat error seperti berikut:

The screenshot shows a web browser window with the URL `localhost:8000/checkout/payment`. The page displays an error message: "Whoops, looks like something went wrong." Below this, a detailed error stack trace is shown:

3/3 ErrorException in CartService.php line 114:

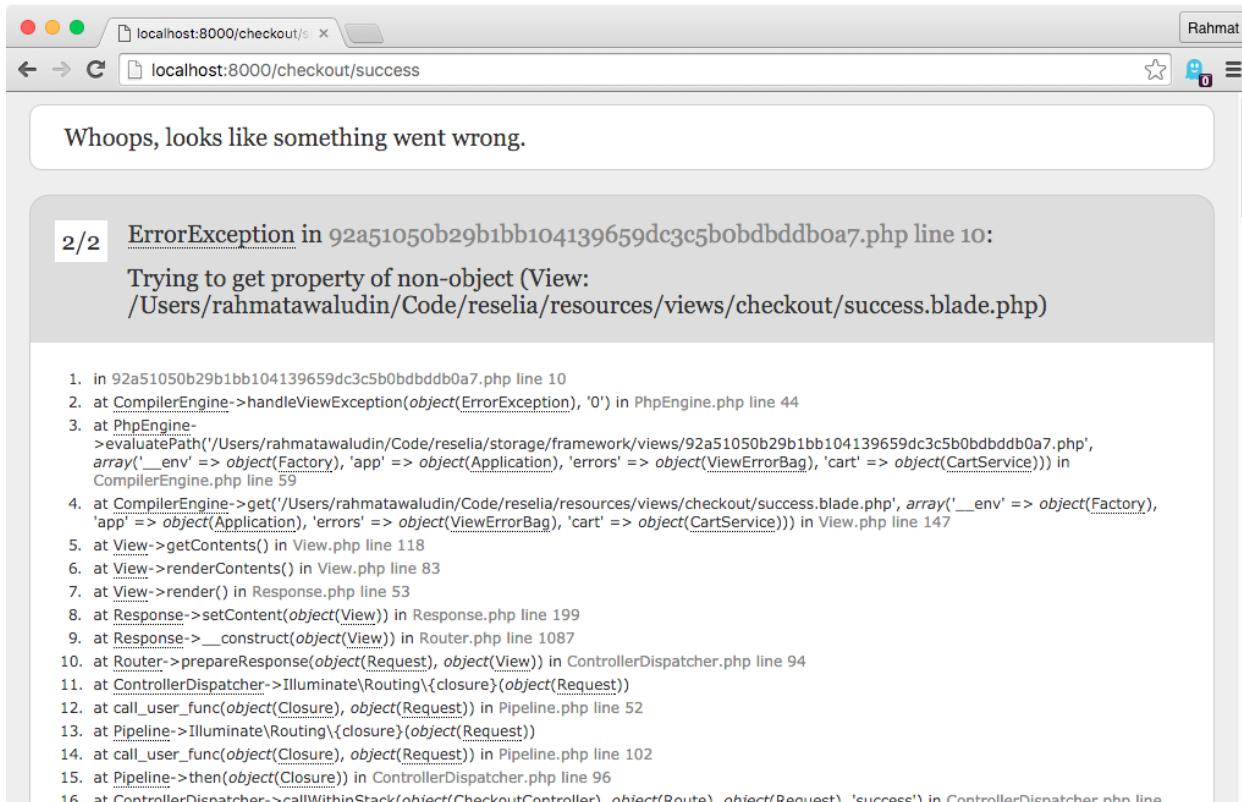
Invalid argument supplied for foreach() (View:
/Users/rahmatawaludin/Code/reselia/resources/views/checkout/_cart-panel.blade.php) (View:
/Users/rahmatawaludin/Code/reselia/resources/views/checkout/_cart-panel.blade.php)

1. in CartService.php line 114
2. at CompilerEngine->handleViewException(object(ErrorException), '0') in PhpEngine.php line 44
3. at PhpEngine->evaluatePath('/Users/rahmatawaludin/Code/reselia/storage/framework/views/3d2d8134932fec6898dda42ab71ef70594739533.php', array('__env' => object(Factory), 'app' => object(Application), 'errors' => object(ViewErrorBag), 'cart' => object(CartService))) in CompilerEngine.php line 59
4. at CompilerEngine->get('/Users/rahmatawaludin/Code/reselia/resources/views/checkout/payment.blade.php', array('__env' => object(Factory), 'app' => object(Application), 'errors' => object(ViewErrorBag), 'cart' => object(CartService))) in View.php line 147
5. at View->getContents() in View.php line 118
6. at View->renderContents() in View.php line 83
7. at View->render() in Response.php line 53
8. at Response->setContent(object(View)) in Response.php line 199
9. at Response->__construct(object(View)) in Router.php line 1087
10. at Router->prepareResponse(object(Request), object(View)) in ControllerDispatcher.php line 94
11. at ControllerDispatcher->Illuminate\Routing\{closure}(object(Request))
12. at call_user_func(object(Closure), object(Request)) in Pipeline.php line 52
13. at Pipeline->Illuminate\Routing\{closure}(object(Request))
14. at call_user_func(object(Closure), object(Request)) in Pipeline.php line 102
15. at Pipeline->then(object(Closure)) in ControllerDispatcher.php line 94

Mengakses checkout payment langsung

Ini terjadi karena pada tahap ini seharusnya kita memiliki data session `checkout.address`.

Untuk tahap sukses pun kita akan mendapat error jika mengakses langsung:



Mengakses checkout success langsung

Error ini terjadi karena kita tidak memiliki session `order` yang digunakan untuk menampilkan detail dari order yang telah dibuat oleh user.

Solusi yang akan kita buat adalah dengan membatasi akses ke setiap halaman tersebut berdasarkan kondisi berikut:

1. Halaman checkout login hanya dapat diakses jika cart memiliki data. Jika belum, kita arahkan user ke halaman cart.
2. Halaman checkout address hanya dapat diakses oleh user jika memiliki data session `checkout.email` (untuk user yang belum login) dan telah melewati kondisi no 1. Jika belum kita arahkan ke halaman checkout login.
3. Halaman checkout payment hanya dapat diakses oleh user jika memiliki data session `checkout.address` dan telah melewati kondisi no 1 dan 2. Jika belum, kita arahkan ke halaman checkout address.
4. Halaman checkout success hanya dapat diakses jika terdapat data session `order`.

Kita akan melakukan implementasi untuk semua kondisi diatas menggunakan middleware.

Mari kita mulai dari kondisi pertama. Kita buat middleware baru dengan nama `CheckoutHaveCart`:

```
php artisan make:middleware CheckoutHaveCart
```

Pada middleware yang digenerate kita isi dengan syntax berikut:

app/Http/Middleware/CheckoutHaveCart.php

```
<?php  
....  
use App\Support\CartService;  
  
class CheckoutHaveCart  
{  
    protected $cart;  
  
    public function __construct(CartService $cart)  
    {  
        $this->cart = $cart;  
    }  
  
    public function handle($request, Closure $next)  
    {  
        if ($this->cart->isEmpty()) return redirect('cart');  
        return $next($request);  
    }  
}
```

Pada middleware ini, kita menggunakan `CartService` untuk menentukan apakah cart memiliki isi. Jika tidak, kita arahkan user ke halaman `cart`. Untuk menggunakan middleware ini, mari kita register dengan nama `checkout.have-cart`:

app/Http/Kernel.php

```
....  
protected $routeMiddleware = [  
    ....  
    'checkout.have-cart' => \App\Http\Middleware\CheckoutHaveCart::class,  
];  
....
```

Kita gunakan middleware ini pada `CheckoutController` di method `__construct`:

app/Http/Controllers/CheckoutController.php

```
....  
public function __construct(CartService $cart)  
{  
    $this->cart = $cart;  
  
    $this->middleware('checkout.have-cart', [  
        'only' => ['login', 'postLogin', 'address', 'postAddress',  
                 'payment', 'postPayment']  
    ]);  
}
```

Pada syntax diatas, kita menambahkan middleware ini pada tahapan login, address dan payment. Cobalah mengakses salah satu URL untuk tiap method diatas dengan cara manual tanpa memiliki data cart, maka kita akan diarahkan ke halaman `cart`.

Kondisi kedua, kita buat dengan nama middleware `CheckoutLoginStepDone`:

```
php artisan make:middleware CheckoutLoginStepDone
```

Kita isi logic di method `handle` nya seperti berikut:

app/Http/Middleware/CheckoutLoginStepDone.php

```
....  
public function handle($request, Closure $next)  
{  
    if (auth()->guest() && !session()->has('checkout.email')) {  
        return redirect('checkout/login');  
    }  
  
    return $next($request);  
}  
....
```

Disini kita mengecek apakah user belum login dan tidak memiliki data session `checkout.email`. Jika ya, kita arahkan user ke halaman `checkout/login`.

Mari kita register dengan nama `checkout.login-step-done`:

app/Http/Kernel.php

```
....  
protected $routeMiddleware = [  
    ....  
    'checkout.login-step-done' => \App\Http\Middleware\CheckoutLoginStepDone::cl\  
ass,  
];  
....
```

Kita gunakan pada CheckoutController:

app/Http/Controllers/CheckoutController.php

```
....  
public function __construct(CartService $cart)  
{  
    ....  
    $this->middleware('checkout.login-step-done', [  
        'only' => ['address', 'postPayment',  
                  'payment', 'postPayment']  
    ]);  
}  
....
```

Pada syntax diatas, kita menggunakan middleware `checkout.login-step-done` pada tahapan checkout address dan checkout payment. Untuk mencoba fitur ini, isilah cart dan langsung mencoba mengunjungi dua tahapan diatas sebelum menyelesaikan checkout login. Kita akan langsung diarahkan ke tahapan checkout login. Sip.

Kondisi ke 3, kita buat dengan midldeware bernama `CheckoutAddressStepDone`:

```
php artisan make:middleware CheckoutAddressStepDone
```

Kita isi method `handle` pada middleware tersebut dengan:

app/Http/Middleware/CheckoutAddressStepDone.php

```
....  
public function handle($request, Closure $next)  
{  
    if (!session()->has('checkout.address')) {  
        return redirect('checkout/address');  
    }  
  
    return $next($request);  
}  
....
```

Pada middleware ini, kita mengecek apakah user tidak memiliki data session checkout.address. Jika ya, kita arahkan ke halaman checkout/address.

Kita daftarkan dengan nama checkout.address-step-done:

app/Http/Kernel.php

```
....  
protected $routeMiddleware = [  
    ....  
    'checkout.address-step-done' => \App\Http\Middleware\CheckoutAddressStepDone\  
    ::class,  
];  
....
```

Kita gunakan pada CheckoutController:

app/Http/Controllers/CheckoutController.php

```
....  
public function __construct(CartService $cart)  
{  
    ....  
    $this->middleware('checkout.address-step-done', [  
        'only' => ['payment', 'postPayment']  
    ]);  
}
```

Untuk mengecek middleware ini, cobalah mengakses halaman checkout payment tanpa menyelesaikan tahapan checkout address. Maka kita akan diarahkan ke halaman checkout/address. Sip.

Middleware terakhir yang akan kita buat digunakan untuk melindungi halaman sukses. Kita beri nama CheckoutPaymentStepDone:

```
php artisan make:middleware CheckoutPaymentStepDone
```

Pada method handle kita isi dengan:

app/Http/Middleware/CheckoutPaymentStepDone.php

```
....  
public function handle($request, Closure $next)  
{  
    if (!session()->has('order')) {  
        return redirect('checkout/payment');  
    }  
  
    return $next($request);  
}  
....
```

Pada method ini kita mengecek apakah terdapat data session dengan nama session. Adanya data session ini menandakan bahwa user telah berhasil melewati tahapan checkout payment.

Kita daftarkan middleware ini dengan nama checkout.payment-step-done:

app/Http/Kernel.php

```
....  
protected $routeMiddleware = [  
    ....  
    'checkout.payment-step-done' => \App\Http\Middleware\CheckoutPaymentStepDone\  
    ::class,  
];  
....
```

Kita gunakan pada controller:

app/Http/Controllers/CheckoutController.php

```
....  
public function __construct(CartService $cart)  
{  
    ....  
    $this->middleware('checkout.payment-step-done', [  
        'only' => ['success']  
    ]);  
}  
....
```

Kini, jika kita sudah menyelesaikan setiap tahapan kecuali tahapan checkout payment, dan mencoba langsung mengakses `checkout/success`, kita akan dikembalikan ke tahapan checkout payment. Sip.

Manajemen Order

Kini kita telah berhasil menerima order dari customer. Fitur selanjutnya yang akan kita buat adalah halaman administrasi order dimana admin dapat melihat semua order, mencari order dan mengubah statusnya. Sebelum kita memulai membuat fitur ini, mari kita bahas dulu bagaimana kita akan mengatur status order di Reselia.

Akan terdapat 4 status order di Reselia:

- “Menunggu Pembayaran”: Status ini terjadi ketika pertama kali order kita isi. Kita akan menggunakan kode `waiting-payment` untuk status ini.
- “Order disiapkan”: Status ini menandakan bahwa barang sedang disiapkan. Kita akan menggunakan kode `packaging` untuk status ini.
- “Paket dikirim”: Status ini menandakan bahwa barang sudah dikirim ke customer. Kita akan menggunakan kode `sent` untuk status ini.
- “Paket diterima”: Status ini menandakan bahwa barang telah diterima oleh customer. Kita akan menggunakan kode `finished` untuk status ini.

Tentunya, jumlah status ini masih bisa dikembangkan sesuai kebutuhan. Untuk saat ini, kita sederhanakan menjadi 4 status.

Untuk memudahkan kita dalam mengakses ke-4 status ini, kita akan membuat static method `statusList` di model `Order` yang akan berisi array kode status berikut nama lengkapnya:

app/Order.php

```
public static function statusList()
{
    return [
        'waiting-payment' => 'Menunggu Pembayaran',
        'packaging' => 'Order disiapkan',
        'sent' => 'Paket dikirim',
        'finished' => 'Paket diterima'
    ];
}
```

Untuk menggunakan nama status di view (bukan kodennya), kita juga akan membuat accessor baru dengan nama `human_status`:

app/Order.php

```
public function getHumanStatusAttribute()
{
    return static::statusList()[$this->status];
}
```

Terakhir, untuk memudahkan validasi ketika merubah status, kita buat method `allowedStatus` yang akan berisi semua kode status dibatasi koma:

app/Order.php

```
public static function allowedStatus()
{
    return array_keys(static::statusList());
}
```

Proses perubahan status kita lakukan secara manual. Mari kita buat controllernya dengan nama `OrdersController`:

```
php artisan make:controller OrdersController
```

Pada controller ini kita hanya membutuhkan method `index`, `edit` dan `update`. Untuk sementara, buat isinya seperti berikut:

app/Http/Controllers/OrdersController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class OrdersController extends Controller
{
    public function index(Request $request)
    {

    }

    public function edit($id)
    {

    }

    public function update(Request $request, $id)
    {

    }
}
```

Untuk routenya, akan kita buat resource routing yang hanya akan menggunakan tiga method diatas. Ini kita lakukan dengan menambah parameter `only`:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {
    ...
    Route::resource('orders', 'OrdersController', ['only' => [
        'index', 'edit', 'update'
    ]]);
});
```

Semua route ini hanya dapat dikunjungi oleh admin. Untuk itu, mari kita buat method `__construct` dan menambahkan middleware yang sesuai:

app/Http/Controllers/OrdersController.php

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('role:admin');
}
```

Menampilkan daftar order

Untuk membuat fitur ini, isi method `index` seperti berikut:

app/Http/Controllers/OrdersController.php

```
....  
use App\Order;  
  
class OrdersController extends Controller  
{  
    ....  
    public function index(Request $request)  
    {  
        $orders = Order::paginate(10);  
        return view('orders.index', compact('orders'));  
    }  
    ....  
}
```

Disini kita melakukan pagination terhadap semua record di table `orders`. Kita buat viewnya:

resources/views/orders/index.blade.php

```
@extends('layouts.app')  
  
@section('content')  
    <div class="container">  
        <div class="row">  
            <div class="col-md-12">  
                <table class="table table-hover">  
                    <thead>  
                        <tr>
```

```

<td>Order #</td>
<td>Customer</td>
<td>Status</td>
<td>Pembayaran</td>
<td>Update terakhir</td>
</tr>
</thead>
<tbody>
@forelse($orders as $order)
<tr>
<td>{{ $order->padded_id }}</td>
<td>{{ $order->user->name }}</td>
<td>{{ $order->human_status }}</td>
<td>
    Total: <strong>{{ number_format($order->total_payment) }} </strong>
<br>
    Transfer ke : {{ config('bank-accounts')[ $order->bank ]['bank'] }} \
} <br>
    Dari : {{ $order->sender }}</td>
<td>{{ $order->updated_at }}</td>
</tr>
@empty
<tr>
<td colspan="4">Tidak ada order yang ditemukan</td>
</tr>
@endforelse
</tbody>
</table>
{!! $orders->links() !!}
</div>
</div>
</div>
@endsection

```

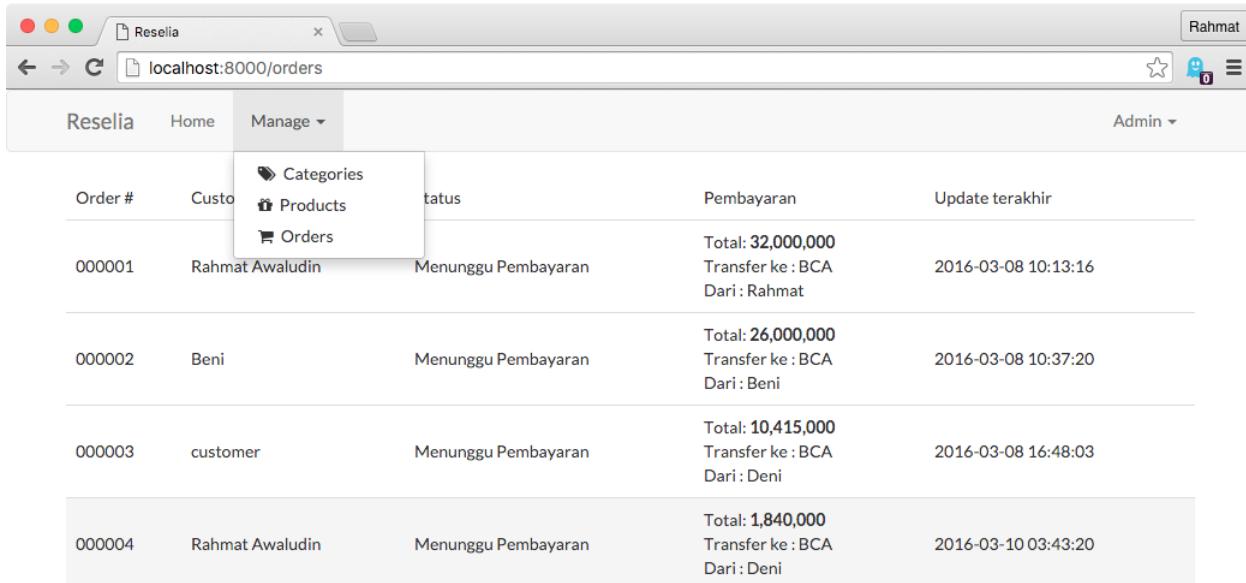
Pada view ini, kita menggunakan `forelse` untuk menampilkan semua order. Jika order tidak ditemukan kita tampilkan "Tidak ada order yang ditemukan". Field yang kita tampilkan pada view ini adalah nomor order, nama customer, status, detail pembayaran dan update terakhir. Kita juga menyertakan link untuk pagination di bawah table.

Kita tambahkan link ke halaman ini pada file layout:

resources/views/layouts/app.blade.php

```
.....
<ul class="nav navbar-nav">
    @if(Auth::check())
        .....
        @can('admin-access')
            <li class="dropdown">
                .....
                <ul class="dropdown-menu" role="menu">
                    .....
                    <li><a href="{{ route('orders.index') }}"><i class="fa fa-bt\
n fa-shopping-cart"></i>Orders</a></li>
                </ul>
            </li>
        @endcan
    @endif
</ul>
.....
```

Tampilan dari halaman ini akan seperti berikut:



Order #	Customer	Status	Pembayaran	Update terakhir
000001	Rahmat Awaludin	Menunggu Pembayaran	Total: 32,000,000 Transfer ke : BCA Dari : Rahmat	2016-03-08 10:13:16
000002	Beni	Menunggu Pembayaran	Total: 26,000,000 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20
000003	customer	Menunggu Pembayaran	Total: 10,415,000 Transfer ke : BCA Dari : Deni	2016-03-08 16:48:03
000004	Rahmat Awaludin	Menunggu Pembayaran	Total: 1,840,000 Transfer ke : BCA Dari : Deni	2016-03-10 03:43:20

Menampilkan semua order**Mencari order**

Kita akan membuat fitur pencarian ini memiliki 3 fitur:

1. Membatasi order untuk status tertentu
2. Pencarian berdasarkan ID
3. Pencarian berdasarkan nama customer

Mari kita bahas satu-persatu.

Membatasi order untuk status tertentu

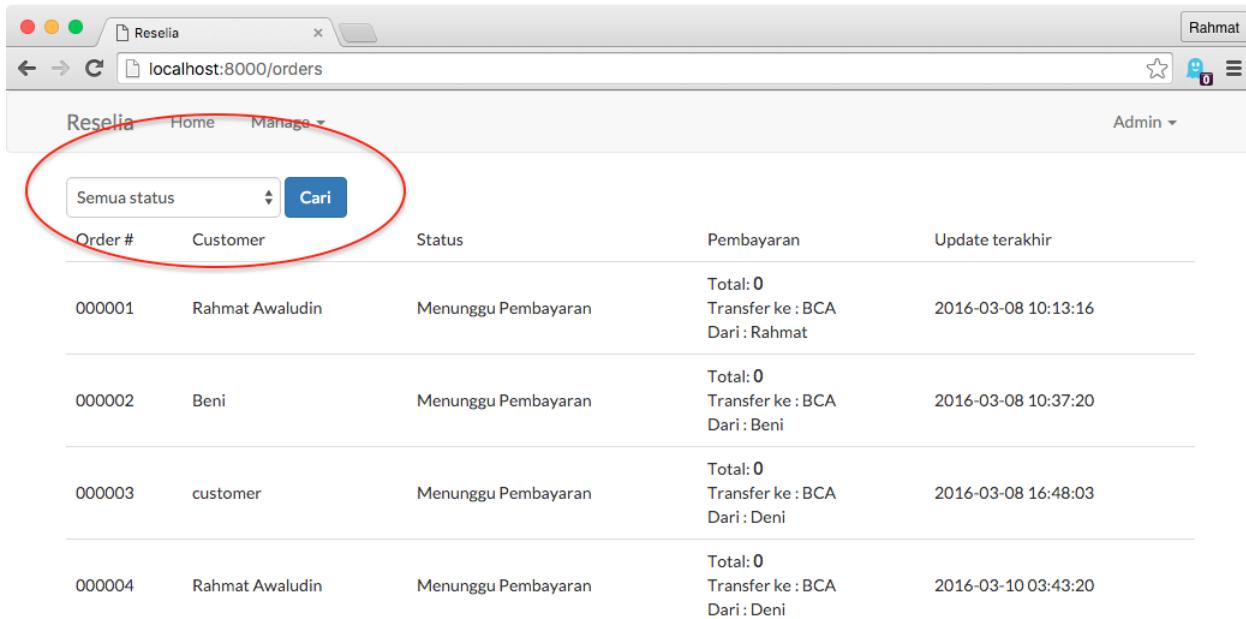
Mari kita buat form filter ini diatas table untuk menampilkan daftar order:

resources/views/orders/index.blade.php

```
....  
<div class="col-md-12">  
    {!! Form::open(['url' => 'orders', 'method'=>'get', 'class'=>'form-inline']) !!}  
    !}  
  
    <div class="form-group {!! $errors->has('status') ? 'has-error' : '' !!}">  
        {!! Form::select('status', [''=>'Semua status']+App\Order::statusList(),  
        isset($status) ? $status : null, ['class'=>'form-control']) !!}  
        {!! $errors->first('status', '<p class="help-block">:message</p>') !!}  
    </div>  
  
    {!! Form::submit('Cari', ['class'=>'btn btn-primary']) !!}  
    {!! Form::close() !!}  
    <table class="table table-hover">  
    ....
```

Disini kita menggunakan isian kosong untuk menampilkan semua status dan menggabungkannya dengan method `statusList` yang telah kita buat sebelumnya.

Tampilan dari form ini akan seperti berikut:



Order #	Customer	Status	Pembayaran	Update terakhir
000001	Rahmat Awaludin	Menunggu Pembayaran	Total: 0 Transfer ke : BCA Dari : Rahmat	2016-03-08 10:13:16
000002	Beni	Menunggu Pembayaran	Total: 0 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20
000003	customer	Menunggu Pembayaran	Total: 0 Transfer ke : BCA Dari : Deni	2016-03-08 16:48:03
000004	Rahmat Awaludin	Menunggu Pembayaran	Total: 0 Transfer ke : BCA Dari : Deni	2016-03-10 03:43:20

Form filter per status

Pada controller, kita ubah method `index` menjadi:

app/Http/Controllers/OrdersController.php

```
public function index(Request $request)
{
    $orders = Order::paginate(10);
    return view('orders.index', compact('orders'));
    $status = $request->get('status');
    $orders = Order::where('status', 'LIKE', '%' . $status . '%')
        ->paginate(10);

    return view('orders.index', compact('orders', 'status'));
}
```

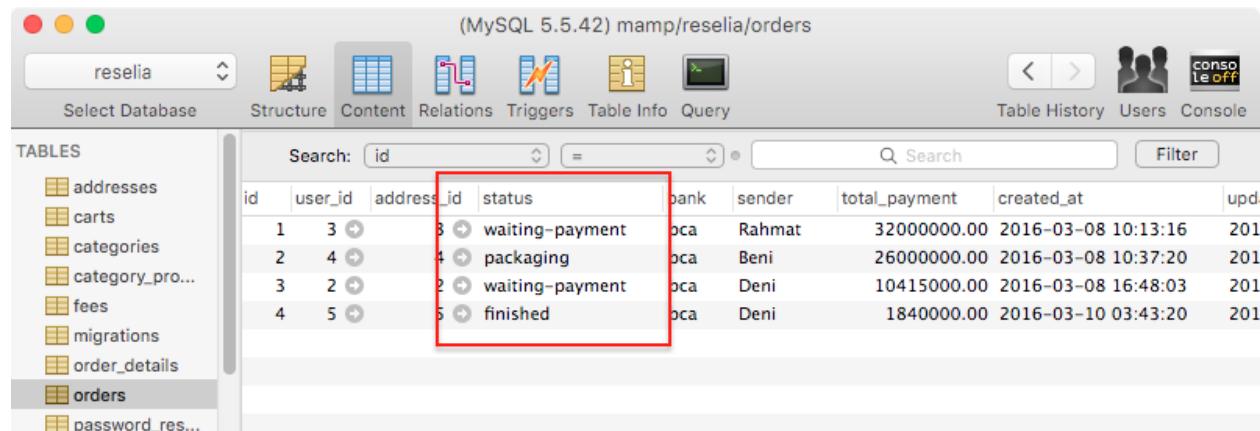
Yang kita lakukan disini adalah menggunakan query `where` untuk mencari order dengan status yang sesuai dengan field `status` yang dikirim. Jika field tersebut tidak dikirim, Laravel akan otomatis menampilkan semua record.

Pada view, kita perlu menambah variable `$status` ke link untuk pagination:

resources/views/orders/index.blade.php

```
{!! $orders->links() !!}
{!! $orders->appends(compact('status'))->links() !!}
```

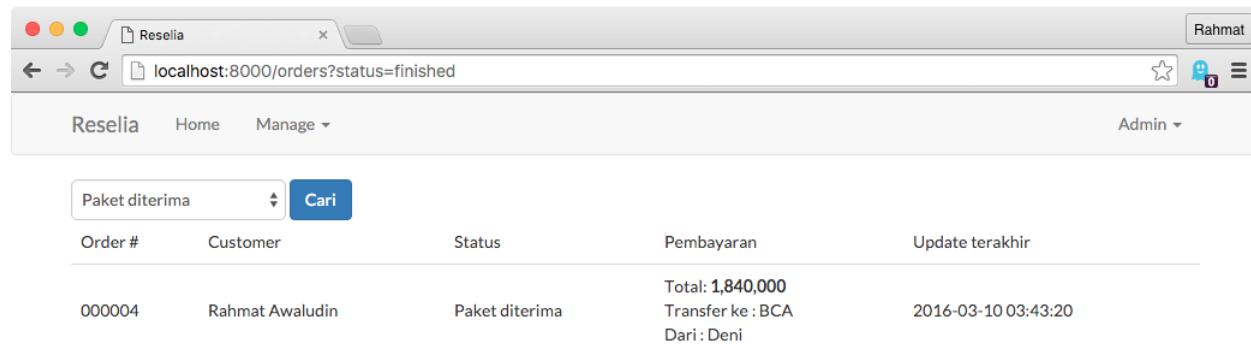
Untuk mencoba filter ini, kita dapat mengubah status secara manual di database.



id	user_id	address_id	status	bank	sender	total_payment	created_at	updated_at
1	3	3	waiting-payment	BCA	Rahmat	32000000.00	2016-03-08 10:13:16	2016-03-08 10:13:16
2	4	4	packaging	BCA	Beni	26000000.00	2016-03-08 10:37:20	2016-03-08 10:37:20
3	2	2	waiting-payment	BCA	Deni	10415000.00	2016-03-08 16:48:03	2016-03-08 16:48:03
4	5	5	finished	BCA	Deni	1840000.00	2016-03-10 03:43:20	2016-03-10 03:43:20

Mengubah status manual

Dan mencoba melakukan filtering:



Order #	Customer	Status	Pembayaran	Update terakhir
000004	Rahmat Awaludin	Paket diterima	Total: 1,840,000 Transfer ke : BCA Dari : Deni	2016-03-10 03:43:20

Filter order berdasarkan status**Pencarian berdasarkan ID**

Untuk melakukan pencarian berdasarkan ID, kita tambahkan isian berikut pada form:

resources/views/orders/index.blade.php

```
....  

{!! Form::open(['url' => 'orders', 'method'=>'get', 'class'=>'form-inline']) !!}  
  

<div class="form-group {!! $errors->has('q') ? 'has-error' : '' !!}">  

  {!! Form::text('q', isset($q) ? $q : null, ['class'=>'form-control',  

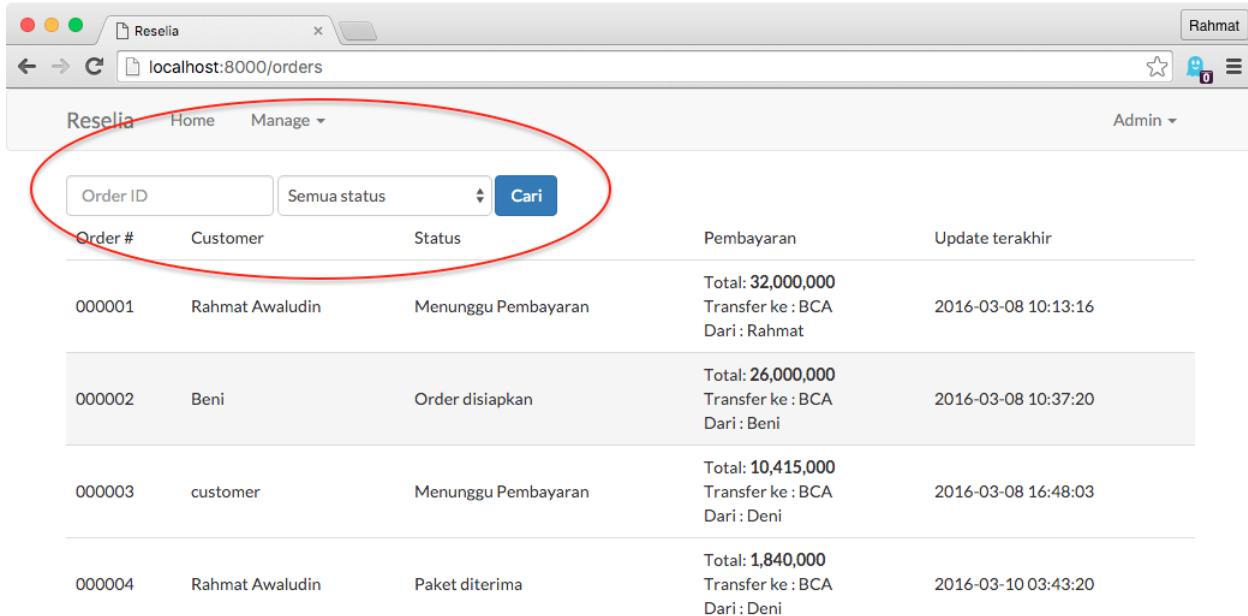
    'placeholder' => 'Order ID']) !!}  

  {!! $errors->first('q', '<p class="help-block">:message</p>') !!}  

</div>  

....
```

Tampilan form akan seperti ini:



The screenshot shows a web application interface for managing orders. At the top, there's a header with the title 'Reselia' and a user profile 'Rahmat'. Below the header, a search bar is highlighted with a red oval, containing fields for 'Order ID' (input), 'Semua status' (dropdown), and a 'Cari' (Search) button. The main content area displays a table of order details with columns: Order #, Customer, Status, Pembayaran (Payment), and Update terakhir (Last Update). The table contains four rows of data.

Order #	Customer	Status	Pembayaran	Update terakhir
000001	Rahmat Awaludin	Menunggu Pembayaran	Total: 32,000,000 Transfer ke : BCA Dari : Rahmat	2016-03-08 10:13:16
000002	Beni	Order disiapkan	Total: 26,000,000 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20
000003	customer	Menunggu Pembayaran	Total: 10,415,000 Transfer ke : BCA Dari : Deni	2016-03-08 16:48:03
000004	Rahmat Awaludin	Paket diterima	Total: 1,840,000 Transfer ke : BCA Dari : Deni	2016-03-10 03:43:20

Form pencarian berdasarkan ID

Kita ubah method index menjadi:

app/Http/Controllers/OrdersController.php

```
public function index(Request $request)
{
    $status = $request->get('status');
    $orders = Order::where('status', 'LIKE', '%' . $status . '%')
        ->paginate(10);

    return view('orders.index', compact('orders', 'status'));
}

$status = $request->get('status');
$orders = Order::where('status', 'LIKE', '%' . $status . '%');
if ($request->has('q')) {
    $q = $request->get('q');
    $orders = $orders->where('orders.id', $q);
}

$orders = $orders->paginate(10);

return view('orders.index', compact('orders', 'status', 'q'));
}
```

Disini kita melakukan sedikit perubahan pada proses query. Kita melakukan pencarian pada field `id` hanya jika field `q` ada di request. Proses pembuatan pagination juga kita pisahkan ke bagian akhir. Terakhir, kita juga menambahkan `q` ke response yang kita kirim.

Pada link pagination di view, kita perlu menambahkan variable `q` agar berjalan dengan benar:

resources/views/orders/index.blade.php

```
{!! $orders->appends(compact('status'))->links() !!}
{!! $orders->appends(compact('status', 'q'))->links() !!}
```

Yang perlu diperhatikan disini adalah jika kita melakukan pencarian berdasarkan ID, field yang akan kita gunakan adalah `id` dan `status`. Jadi, jika status yang pilih adalah “Order disiapkan”, maka pencarian untuk ID **hanya** dilakukan untuk order dengan status tersebut.

Order #	Customer	Status	Pembayaran	Update terakhir
000002	Beni	Order disiapkan	Total: 26,000,000 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20

Berhasil melakukan pencarian dengan ID order

Pencarian untuk nama customer tertentu

Proses pencarian dengan nama customer akan sedikit kompleks. Ini karena kita akan mencari dengan field yang berada pada table relasi (`users`). Untuk melakukan ini, kita dapat menggunakan advanced where dan `whereHas` (atau tepat `orWhereHas`) di Laravel.

Berikut hasil akhir yang akan kita buat pada method `index`:

app/Http/Controllers/OrdersController.php

```
public function index(Request $request)
{
    $status = $request->get('status');

    $orders = Order::where('status', 'LIKE', '%' . $status . '%');
    if ($request->has('q')) {
        $q = $request->get('q');
        $orders = $orders->where('orders.id', $q);
    }

    if ($request->has('q')) {
        $q = $request->get('q');
        $orders = $orders->where(function($query) use ($q) {
            $query->where('id', $q)
                ->orWhereHas('user', function($user) use ($q) {
                    $user->where('name', 'LIKE', '%' . $q . '%');
                });
        });
    }

    $orders = $orders->paginate(10);
```

```
    return view('orders.index', compact('orders', 'status', 'q'));
}
```

Pada syntax diatas, kita menggunakan closure pada query where:

```
$orders = $orders->where(function($query) use ($q) {
```

Menggunakan query ini kita dapat menggabungkan beberapa query dalam query where tersebut. Menggunakan `use` kita juga mengirim variable `q` ke dalam closure tersebut.

Selanjutnya, kita menggunakan `orWhereHas` untuk melakukan query terhadap relasi di table `users`:

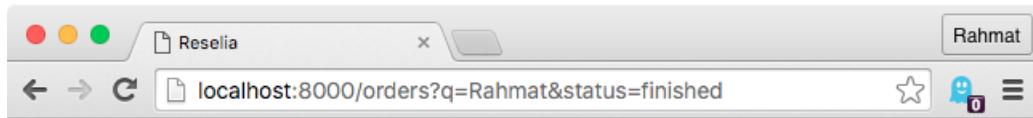
```
->orWhereHas('user', function($user) use ($q) {
    $user->where('name', 'LIKE', '%' . $q . '%');
});
```

Query `orWhereHas` (atau `whereHas`) menerima dua paramter. Pertama nama relasi yang akan kita gunakan. Kedua closure yang akan menerima parameter relasi yang siap di query.

Disini kita menggunakan relasi `user` untuk melakukan query terhadap table `users`. Pada closure, kita *passing* kembali variable `q` dan mencari pada field `name` dengan query `LIKE`. Untuk mengecek query yang dihasilkan, kita dapat melakukan `dump sql` dengan menggunakan method `toSql()`. Misalnya kita buat syntax seperti ini sebelum memanggil methog `paginate()`:

```
dd($orders->toSql());
```

Jika kita mencoba mencari order dengan nama customer dan status tertentu, kita akan mendapatkan hasil query seperti ini:



Query yang dihasilkan

Sip.

Pastikan untuk menghapus kembali syntax dump sql diatas sebelum melanjutkan.

Agar lebih intuitif, mari kita ubah placeholder untuk pencarian:

resources/views/orders/index.blade.php

```
....  
{!! Form::text('q', isset($q) ? $q : null, ['class'=>'form-control',  
'placeholder' => 'Order ID']) !!}  
{!! Form::text('q', isset($q) ? $q : null, ['class'=>'form-control',  
'placeholder' => 'Order ID/Customer...']) !!}  
....
```

Kita dapat mengetes dengan melakukan pencarian berdasarkan ID, customer atau status.

Order #	Customer	Status	Pembayaran	Update terakhir
000001	Rahmat Awaludin	Menunggu Pembayaran	Total: 32,000,000 Transfer ke : BCA Dari: Rahmat	2016-03-08 10:13:16
000004	Rahmat Awaludin	Paket diterima	Total: 1,840,000 Transfer ke : BCA Dari: Deni	2016-03-10 03:43:20

Mencari dengan nama customer

Order #	Customer	Status	Pembayaran	Update terakhir
000003	customer	Menunggu Pembayaran	Total: 10,415,000 Transfer ke : BCA Dari: Deni	2016-03-08 16:48:03

Mencari dengan ID

Order #	Customer	Status	Pembayaran	Update terakhir
000002	Beni	Order disiapkan	Total: 26,000,000 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20

Mencari status tertentu

Mengubah status

Untuk membuat fitur ini, kita akan membuat ID order menjadi link ke halaman untuk mengubah statusnya. Kita ubah menjadi seperti berikut:

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

```
....  
<td>{{ $order->padded_id }}</td>  
<td><a href="{{ route('orders.edit', $order->id)}}">{{ $order->padded_id }}</a><\br/>
```

Kita buat method edit seperti berikut:

`app/Http/Controllers/OrdersController.php`

```
public function edit($id)  
{  
    $order = Order::find($id);  
    return view('orders.edit')->with(compact('order'));  
}
```

Dan viewnya:

resources/views/orders/edit.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h3>Edit {{ $order->title }}</h3>
                {!! Form::model($order, ['route' => ['orders.update', $order], 'method'=>'patch', 'class'=>'form-horizontal']) !!}
                @include('orders._form', ['model' => $order])
                {!! Form::close() !!}
            </div>
        </div>
    </div>
@endsection
```

Pada view ini, kita menggunakan partial view `orders._form`. Berikut syntaxnya:

resources/views/orders/_form.blade.php

```
<div class="form-group">
    <div class="row">
        <div class="col-md-4 text-right">
            <label>Order #</label>
        </div>
        <div class="col-md-6">
            {{ $order->padded_id }}
        </div>
    </div>
</div>

<div class="form-group">
    <div class="row">
        <div class="col-md-4 text-right">
            <label>Customer</label>
        </div>
        <div class="col-md-6">
            {{ $order->user->name }}
        </div>
    </div>
</div>
```

```
<div class="form-group">
    <div class="row">
        <div class="col-md-4 text-right">
            <label>Alamat Pengiriman</label>
        </div>
        <div class="col-md-6">
            <address>
                <strong>{{ $order->address->name }}</strong> <br>
                {{ $order->address->detail }} <br>
                {{ $order->address->regency->name }}, {{ $order->address->regency->provin\
ce->name }} <br>
                <abbr title="Phone">P:</abbr> +62 {{ $order->address->phone }}<br>
            </address>
        </div>
    </div>
</div>

<div class="form-group">
    <div class="row">
        <div class="col-md-4 text-right">
            <label>Detail</label>
        </div>
        <div class="col-md-6">
            @include('orders._details', compact('order'))
        </div>
    </div>
</div>

<div class="form-group{{ $errors->has('status') ? ' has-error' : '' }}>
    <div class="row">
        <div class="col-md-4 text-right">
            <label class="control-label">Status</label>
        </div>
        <div class="col-md-6">
            {!! Form::select('status', App\Order::statusList(), null, ['class'=>'f\
orm-control']) !!}
            {!! $errors->first('status', '<p class="help-block">:message</p>') !!}
        </div>
    </div>
</div>
```

```
{!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
```

Yang kita lakukan pada form ini adalah menampilkan detail dari order dan pilihan status yang sedang aktif. Kita menggunakan partial view `orders._details` untuk menampilkan detail dari order. Isinya akan seperti berikut:

`resources/views/orders/_details.blade.php`

```
<table class="table table-condensed">
  <thead>
    <tr>
      <th style="width:50%">Produk</th>
      <th style="width:20%">Jumlah</th>
      <th style="width:30%">Harga</th>
    </tr>
  </thead>
  <tbody>
    @foreach($order->details as $detail)
    <tr>
      <td data-th="Produk">{{ $detail->product->name }}</td>
      <td data-th="Jumlah" class="text-center">{{ $detail->quantity }}</td>
      <td data-th="Harga" class="text-right">{{ number_format($detail->price) }}\n
    </td>
    </tr>
    <tr>
      <td data-th="Subtotal">Subtotal</td>
      <td data-th="Subtotal" class="text-right" colspan="2">Rp{{ number_format($\n
      detail->total_price) }}</td>
    </tr>

    @endforeach
  </tbody>
  <tfoot>
    <tr>
      <td data-th="Subtotal"><strong>Ongkos Kirim</strong></td>
      <td data-th="Subtotal" class="text-right" colspan="2"><strong>Rp{{ number_\n
      format($order->total_fee) }}</strong></td>
    </tr>
    <tr>
      <td><strong>Total</strong></td>
      <td class="text-right" colspan="2"><strong>Rp{{ number_format($order->to\n
      tal_payment) }}</strong></td>
    </tr>
```

```
</tfoot>
</table>
```

Hasil akhir dari halaman edit order akan seperti berikut:

The screenshot shows a web application interface for editing an order. At the top, there's a header bar with the title 'Reselia' and a user 'Rahmat'. Below it is a navigation bar with links 'Reselia', 'Home', 'Manage', and 'Admin'. The main content area has a title 'Edit' and displays the following information:

Detail	Produk	Jumlah	Harga
	Nike Air Max	20	420,000
	Subtotal		Rp9,200,000
	Nike Aeroloft Bomber	30	720,000
	Subtotal		Rp22,800,000
	Ongkos Kirim		Rp0
	Total		Rp32,000,000

Below the table, there's a 'Status' dropdown menu set to 'Menunggu Pembayaran'. At the bottom left is a blue 'Simpan' button.

Halaman ubah status

Untuk menerima form ini, kita sesuaikan method update menjadi:

app/Http/Controllers/OrdersController.php

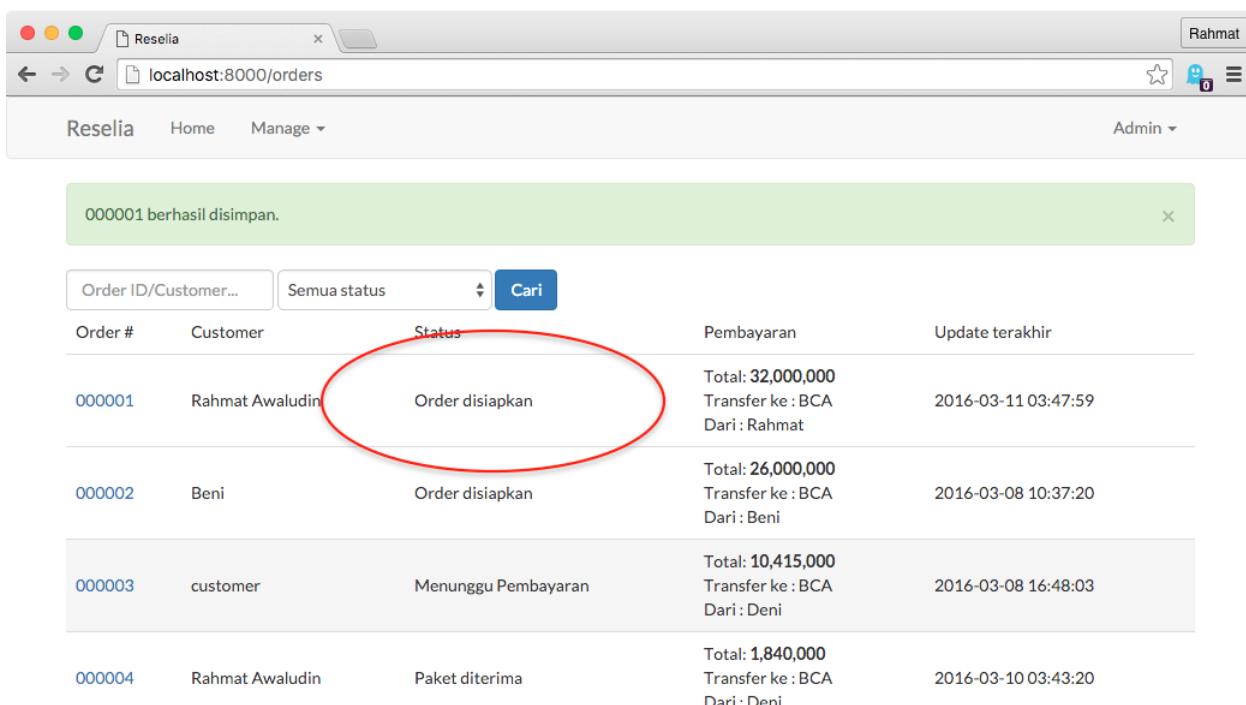
```
public function update(Request $request, $id)
{
    $order = Order::find($id);
    $this->validate($request, [
        'status' => 'required|in:' . implode(',', Order::allowedStatus())
    ]);
    $order->update($request->only('status'));
    \Flash::success($order->padded_id . ' berhasil disimpan.');
}
```

```
return redirect()->route('orders.index');
}
```

Yang kita lakukan pada method ini sebagai berikut:

- Kita melakukan validasi untuk field status. Selain rule `required`, kita juga menggunakan rule `in` dengan parameter hasil dari method `allowedstatus` yang telah kita buat sebelumnya. Menggunakan `implode`, kita merubah response dari method tersebut menjadi string dengan pemisah koma.
- Kita update field `status` pada order.
- Kita siapkan pesan sukses
- Kita arahkan user kembali ke daftar order

Berikut tampilan ketika kita berhasil merubah status order:



The screenshot shows a web application interface for managing orders. At the top, there's a header with the title 'Reselia' and a user 'Rahmat'. Below the header, a navigation bar includes 'Home' and 'Manage'. A success message '000001 berhasil disimpan.' is displayed in a green box. The main content is a table listing four orders. The first order, '000001', has its 'Status' field highlighted with a red oval. The table columns are 'Order #', 'Customer', 'Status', 'Pembayaran', and 'Update terakhir'. The data for each row is as follows:

Order #	Customer	Status	Pembayaran	Update terakhir
000001	Rahmat Awaludin	Order disiapkan	Total: 32,000,000 Transfer ke : BCA Dari : Rahmat	2016-03-11 03:47:59
000002	Beni	Order disiapkan	Total: 26,000,000 Transfer ke : BCA Dari : Beni	2016-03-08 10:37:20
000003	customer	Menunggu Pembayaran	Total: 10,415,000 Transfer ke : BCA Dari : Deni	2016-03-08 16:48:03
000004	Rahmat Awaludin	Paket diterima	Total: 1,840,000 Transfer ke : BCA Dari : Deni	2016-03-10 03:43:20

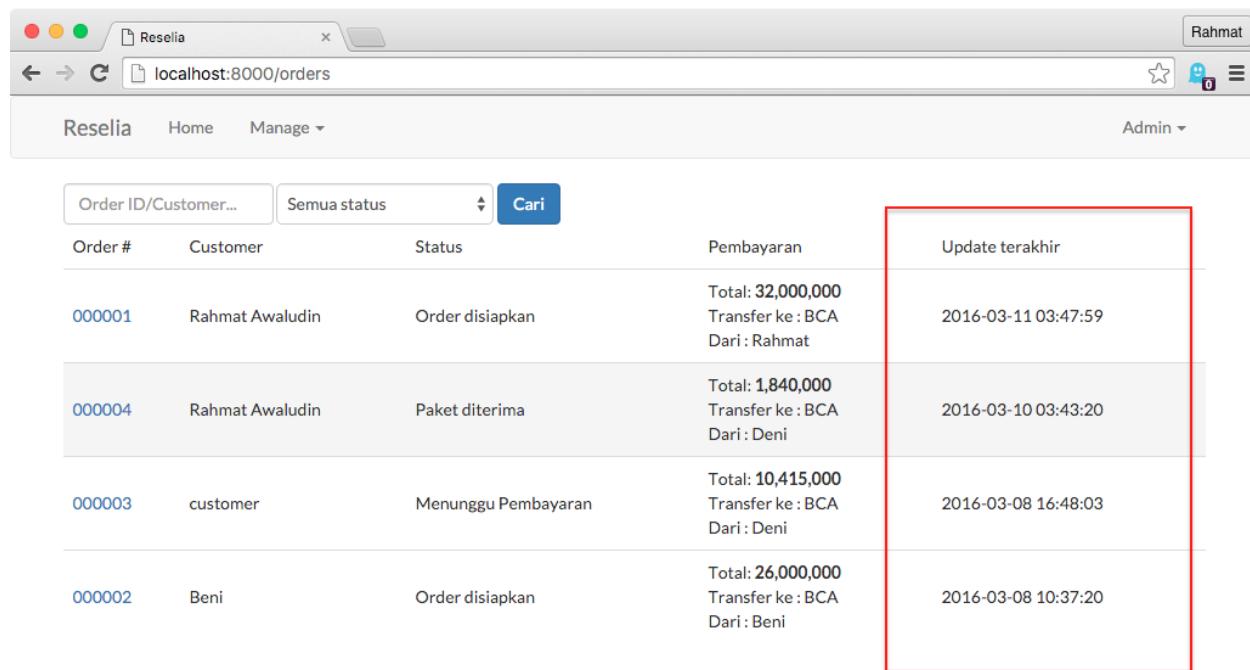
Berhasil merubah status order

Kita juga dapat mengatur agar tampilan diurutkan berdasarkan update terakhir dengan menambah baris berikut pada method `index`:

app/Http/Controllers/OrdersController.php

```
public function index(Request $request)
{
    ....
    $orders = $orders->paginate(10);
    $orders = $orders->orderBy('updated_at', 'desc')
        ->paginate(10);

    return view('orders.index', compact('orders', 'status', 'q'));
}
```



Pengurutan berdasarkan update terakhir

Cek Order

Kita akan membuat dashboard sederhana untuk customer. Pada dashboard ini, customer dapat melihat semua status yang dimilikinya dan mencari berdasarkan ID dan status. Kebanyakan syntax disini akan sama dengan cara kita membuat fitur daftar order di halaman admin.

Mari kita buat dulu link navigasi yang akan diakses oleh customer. Tambahkan baris berikut setelah link untuk cart pada file layout:

resources/views/layouts/app.blade.php

```
....  

<!-- Right Side Of Navbar -->  

<ul class="nav navbar-nav navbar-right">  

    @include('layouts._customer-feature', ['partial_view'=>'layouts._cart-menu-bar', 'data' => compact('cart')])  

    @include('layouts._customer-feature', ['partial_view'=>'layouts._check-order-menu-bar', 'data'=>[]])  

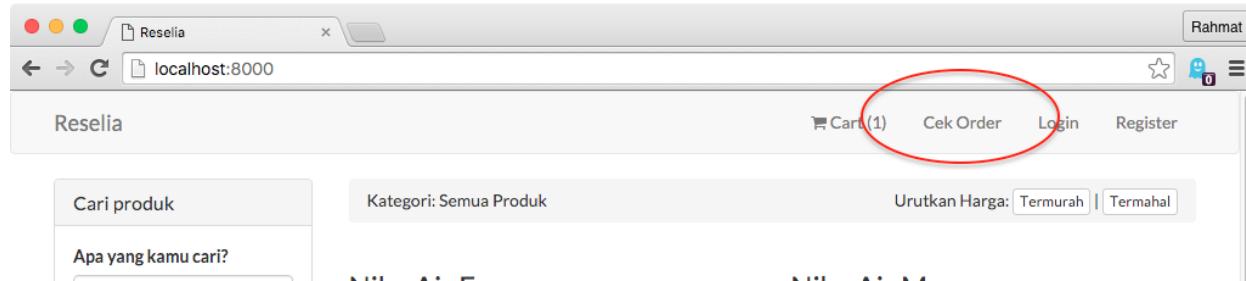
....
```

Disini kita menggunakan mediator view `layouts._customer-feature` karena link untuk mengecek order ini akan kita tampilkan untuk user yang belum login dan yang sudah login dengan role customer. Berikut tampilan `layouts._check-order-menu-bar`:

resources/views/layouts/_check-order-menu-bar.blade.php

```
<li><a href="{{ url('/home/orders') }}>Cek Order</a></li>
```

Tampilan dari menu ini akan seperti berikut:



Link untuk cek order

Mari kita buat routenya:

app/Http/routes.php

```
Route::group(['middleware' => 'web'], function () {  

    ....  

    Route::get('/home/orders', 'HomeController@viewOrders');  

});
```

Terlihat disini, kita akan menggunakan URL `/home/orders` dan mengarahkan ke method `viewOrders` di `HomeController`. Karena method ini hanya bisa diakses oleh customer, mari kita tambahkan dulu middleware nya:

app/Http/Controllers/HomeController.php

```
public function __construct()
{
    $this->middleware('auth', ['only'=>['index', 'viewOrders']]);
    $this->middleware('role:customer', ['only' => 'viewOrders']);
}
```

Kita buat method `viewOrders` seperti berikut:

app/Http/Controllers/HomeController.php

```
public function viewOrders(Request $request)
{
    $q = $request->get('q');
    $status = $request->get('status');
    $orders = auth()->user()->orders()
        ->where('id', 'LIKE', '%' . $q . '%')
        ->where('status', 'LIKE', '%' . $status . '%')
        ->orderBy('updated_at')
        ->paginate(5);
    return view('customer.view-orders')->with(compact('orders', 'q', 'status'));
}
```

Syntax diatas hampir mirip dengan syntax yang kita lakukan pada method `index` di `OrdersController`. Yang kita lakukan disini adalah mencari order yang dimiliki user tersebut:

`auth()->user()->orders()`

Agar query ini berjalan, kita harus menambahkan relasi ke `orders` pada model User:

app/User.php

```
public function orders()
{
    return $this->hasMany('App\Order');
```

Pada field `id` dan `status`. Kita juga mengurutkan berdasarkan field `updated_at` dan melakukan paginasi sebanyak 5 order perhalaman.

View yang akan kita gunakan untuk menampilkan data order ini akan seperti berikut:

resources/views/customer/view-orders.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                {!! Form::open(['url' => 'home/orders', 'method'=>'get', 'class'=>'form-\ninline']) !!}
                    <div class="form-group {!! $errors->has('q') ? 'has-error' : '' !!}">
                        {!! Form::text('q', isset($q) ? $q : null, ['class'=>'form-control', 'placeholder' => 'Order ID']) !!}
                        {!! $errors->first('q', '<p class="help-block">:message</p>') !!}
                    </div>

                    <div class="form-group {!! $errors->has('status') ? 'has-error' : '' !!}">
                        {!! Form::select('status', [''=>'Semua status']+App\Order::statusList(), isset($status) ? $status : null, ['class'=>'form-control']) !!}
                        {!! $errors->first('status', '<p class="help-block">:message</p>') !!}
                    </div>

                    {!! Form::submit('Cari', ['class'=>'btn btn-primary']) !!}
                {!! Form::close() !!}
            <table class="table">
                <thead>
                    <tr>
                        <td>Order #</td>
                        <td>Tanggal Order</td>
                        <td>Status</td>
                        <td>Pembayaran</td>
                        <td>Detail</td>
                    </tr>
                </thead>
                <tbody>
                    @forelse($orders as $order)
                    <tr>
                        <td>{{ $order->padded_id }}</td>
                        <td>{{ $order->created_at }}</td>
                        <td>{{ $order->human_status }}</td>
                        <td>
```

```
Total: <strong>{{ number_format($order->total_payment) }} </stro\
ng><br>
Transfer ke : {{ config('bank-accounts')[ $order->bank ]['bank'] }}\
} a.n {{ config('bank-accounts')[ $order->bank ]['name'] }} {{ config('bank-accoun\
ts')[ $order->bank ]['number'] }}<br>
Dari : {{ $order->sender }}</td>
<td>
    @include('orders._details', compact('order'))</td>
</tr>
@empty
<tr>
    <td colspan="4">Tidak ada order yang ditemukan</td>
</tr>
@endforelse
</tbody>
</table>
{!! $orders->appends(compact('q', 'status))->links() !!}
</div>
</div>
</div>
@endsection
```

Yang kita lakukan disini adalah menampilkan semua order yang ditemukan dalam bentuk table dan melakukan paginasi. Disini, kita juga menggunakan partial view `orders._details` yang telah dibuat sebelumnya untuk menampilkan detail dari order. Tampilan halaman ini akan seperti berikut:

Produk	Jumlah	Harga
Nike Air Force	10	340,000
Subtotal		Rp3,620,000
Nike Air Max	15	420,000
Subtotal		Rp6,795,000
Ongkos Kirim		Rp0
Total		Rp10,415,000

Halaman cek order

Cobalah melakukan pencarian order pada halaman ini. Pastikan semua fitur berfungsi dengan benar.

Whats Next?

Bab ini merupakan bab terpanjang yang pernah saya tulis. Nampaknya cocok menjadi buku terpisah.. :)

Meskipun begitu, aplikasi e-commerce yang kita bangun masih memiliki banyak fitur bisa dikembangkan. Sebagai bahan ide, bisa dicoba kembangkan beberapa fitur berikut:

- Sign up dengan facebook
- Fitur wishlist
- Fitur stock
- Otomatis pembatalan order jika tidak ada konfirmasi setelah 24 jam
- Custom order number
- Halaman crud alamat untuk customer (dari dashboard customer)
- Buat alamat default ketika checkout
- Fitur dropship

- Form untuk langsung buat akun dengan menulis password untuk guest check-out
- Notifikasi (email) admin setiap kali ada order baru
- Notifikasi (email) customer setiap status order berubah
- Konfirmasi transfer + optional upload bukti transfer
- Mekanisme perubah status yang lebih kompleks
- Laporan penjualan terkini
- Laporan statistik user terkini
- Resize foto produk ketika upload
- Dll



Selamat!

Kamu telah berhasil menyelesaikan buku ini. Di luar sana, masih banyak ilmu Laravel yang bisa dipelajari. Sebagaimana ilmu lain, yang lama melekat adalah yang dipraktekan. Ketika kita banyak praktek, pasti akan menemui berbagai *case* baru dalam menggunakan Laravel. Gunakan kesempatan itu untuk belajar.

Harapan saya, semoga buku ini senantiasa menjadi rujukan ketika Anda menemui masalah di Laravel. Setiap saran maupun koreksi untuk buku ini akan saya terima silahkan email atau mention saya [@rahmatawaludin²²⁰](https://twitter.com/rahmatawaludin). Terima kasih.. :) %% Bahan update %% update data provinsi + regency per minggu

²²⁰<https://twitter.com/rahmatawaludin>