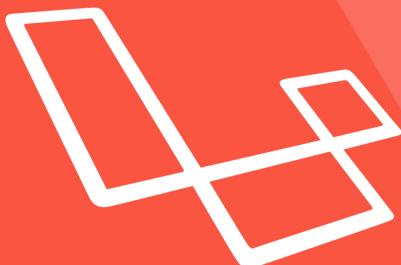


SERI SEMINGGU BELAJAR



seminggu belajar

Laravel^{5.3}

RAHMAT AWALUDIN

Seminggu Belajar Laravel

Laravel itu framework PHP yang bikin hidup programmer lebih menyenangkan. Jadi, belajarnya juga mesti menyenangkan.

Rahmat Awaludin

Buku ini dijual di <http://leanpub.com/seminggubelajarlaravel>

Versi ini diterbitkan pada 2016-09-16



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 Buku Ini!

Bantulah Rahmat Awaludin dengan mewartakan buku ini via [Twitter](#)!

Tweet yang disarankan untuk buku ini adalah:

[Hei, gue baru aja download buku Seminggu Belajar Laravel. Keren nih, buat belajar framework Laravel!](#)

Tagar yang disarankan untuk buku ini adalah [#seminggubelajarlaravel](#).

Temukan kata orang tentang buku ini dengan mengklik tautan ini untuk menampilkan tagar ini di Twitter:

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

Juga Oleh Rahmat Awaludin

Menyelami Framework Laravel

Untuk istriku tercinta Irna Rahyu dan jagoan kecilku Shidqi Abdulllah Mubarak.

Contents

Koreksi	i
Saran dan Masukan	ii
Sekilas	iii
Metode Penulisan	iii
Membaca sample source code	iii
Kalau ada materi yang kurang gimana?	v
Lisensi	v
Reseller	vi
Tawaran Kerjaan	vi
1. Hari 1 : Instalasi dan Konfigurasi Laravel	1
1.1 Text Editor	1
1.2 Kebutuhan Sistem	2
1.3 Composer	2
1.3.1 Install Composer	3
1.3.2 Penggunaan Composer	4
1.4 Instalasi Laravel	7
1.5 Konfigurasi	9
1.6 PHP builtin web server	10
1.7 Konfigurasi Database	11
1.7.1 Error koneksi database?	12
1.8 Virtual Host	14
1.8.1 MAMP	14
1.8.2 XAMPP	15
1.9 Ringkasan	18

CONTENTS

2. Hari 2 : Routing, MVC dan Authentikasi	19
2.1 Routing	19
2.2 MVC	21
2.3 Model	22
2.3.1 Migrations	22
2.3.2 Database Seeder	26
2.3.3 Membuat Model	29
2.3.4 Mengakses model	30
2.4 View	31
2.4.1 Template dengan Blade	32
2.4.2 Form	33
2.4.3 Request	34
2.5 Controller	35
2.6 Authentikasi	36
2.7 Ringkasan	40
3. Hari 3 : Persiapan Project	41
3.1 Design tampilan	41
3.1.1 Tampilan Halaman Depan untuk Guest	42
3.1.2 Halaman Admin	43
3.1.3 Halaman User	45
3.2 Database	48
3.3 Code sample	49
3.4 Instalasi Laravel	51
3.5 Konfigurasi Authentikasi	52
3.6 Memahami Layouting	61
3.6.1 Parent View	61
3.6.2 Child View	62
3.6.3 Partial View	63
3.6.4 Contoh penggunaan layouting	63
3.7 Konfigurasi Asset	66
3.8 Memahami Validasi Data	73
3.9 Konfigurasi Role	75
3.9.1 Membuat sample user	78
3.9.2 Konfigurasi Register	81
3.10 Memahami Middleware	83

CONTENTS

3.11	Ringkasan	86
4.	Hari 4 : Develop Fitur Admin	87
4.1	Penggunaan Route Group	87
4.2	Penggunaan RESTful Resource Controller	88
4.3	Persiapan Model dan Migration	89
4.4	Penggunaan Mass Assignment	91
4.5	Persiapan Relasi One-to-Many di Eloquent	93
4.6	Menyiapkan Sample Buku dan Penulis	94
4.7	Menampilkan Data dengan DataTable	95
4.8	Error di DataTable?	104
4.9	Error NotFoundHttpException di Datatable?	106
4.10	Membatasi Akses dengan Role	107
4.11	Penggunaan Halaman Error Custom	110
4.12	Fitur Tambah Penulis	112
4.13	Penggunaan Flash Messages	118
4.14	Penggunaan Form Model Binding	120
4.15	Penghapusan Data Penulis	125
4.16	Penggunaan Model Event	128
4.17	Konfirmasi ketika Menghapus Data	130
4.18	CRUD Buku	133
4.18.1	Menampilkan daftar buku	133
4.18.2	Menambah Buku	137
4.18.3	Mengubah Buku	144
4.18.4	Menghapus Buku	148
4.19	Penggunaan Form Request	150
4.20	Penggunaan Selectize	155
4.21	Ringkasan	158
5.	Hari 5 : Develop Fitur Non-Admin	159
5.1	Persiapan Database dan Model untuk Peminjaman	159
5.2	Membuat Sample Peminjaman	161
5.3	Pembuatan Fitur Peminjaman	162
5.4	Membatasi Jumlah Buku yang Dipinjam	168
5.5	Penggunaan Attribute Casting pada Eloquent	171
5.6	Penggunaan Artisan Tinker	172

CONTENTS

5.7	Penggunaan Query Scope pada Eloquent	173
5.8	Membedakan Tampilan Dashboard untuk Admin dan Member	174
5.9	Menampilkan Buku yang Sedang Dipinjam	177
5.10	Pengembalian Buku	179
5.11	Penggunaan Custom Accessor	181
5.12	Menampilkan Stok Buku	183
5.13	Membatasi Peminjaman Berdasarkan Stok Buku	185
5.14	Validasi Ketika Mengubah Jumlah Buku	186
5.15	Membatasi Penghapusan Ketika Buku Masih Dipinjam	190
5.16	Ringkasan	192
6.	Hari 6 : User Management	193
6.1	Penggunaan No CAPTCHA reCAPTCHA	193
6.2	Konfigurasi Email dengan Mailgun	199
6.3	Fitur Verifikasi/Aktivasi User	207
6.3.1	Persiapan Database	208
6.3.2	Membatasi Akses yang Belum Terverifikasi dengan Middleware	210
6.3.3	Persiapan Routing	212
6.3.4	Kirim Email Verifikasi ketika Mendaftar	213
6.3.5	Melakukan Verifikasi	216
6.3.6	Mengirim Ulang Link Verifikasi	218
6.4	Pembuatan Halaman Profil	221
6.4.1	Mengubah Data Profil	224
6.4.2	Mengubah Password setelah Login	228
6.4.3	Membuat Rule Validasi Custom	233
6.5	Management Member	236
6.5.1	Menampilkan Daftar Member	237
6.5.2	Menambah Member oleh Admin	239
6.5.3	Mengubah Data Member	246
6.5.4	Melihat Detail Peminjaman	248
6.5.5	Menghapus Member	253
6.6	Menampilkan Daftar Peminjaman	254
6.7	Filter Buku Berdasarkan Status Peminjaman	258
6.8	Ringkasan	262
7.	Hari 7 : Deploy Aplikasi	263

CONTENTS

7.1	Mengupload File	263
7.2	Konfigurasi public_path	264
7.3	Konfigurasi Database	265
7.4	Konfigurasi Captcha	265
7.5	Verifikasi Mailgun	267
7.6	Nonaktifkan Debug	279
7.7	Ringkasan	279
8.	Bonus	280
8.1	Membuat Chart	280
8.2	Export Data ke Excel	290
8.3	Export Data ke PDF	300
8.4	Import Data dari Excel	308
8.5	Penggunaan Ajax	317
8.5.1	Cek Ajax	318
8.5.2	CSRF	318
8.5.3	JSON	320
8.6	Menampilkan Login Terakhir	325
8.7	Penggunaan Blade Macro	330
Penutup	333	
Belajar lagi!	333	
Terima kasih	334	

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.com atau via twitter di [@rahmatawaludin](https://twitter.com/rahmatawaludin).

Sekilas

Perkenalkan, saya Rahmat Awaludin. Saya seorang fullstack web developer dari Subang, Jawa Barat, Indonesia. Sudah sejak tahun 2011 saya berada di dunia web development, lebih tepatnya ketika saya masih berstatus sebagai mahasiswa di Universitas Padjadjaran.

Perkenalan saya dengan Laravel dimulai ketika saya mengerjakan sebuah project untuk LSM Save The Children. Selama mengerjakan project ini saya belajar banyak tentang Laravel dan modern PHP development, termasuk di dalamnya berbagai konsep dan tools seperti SOLID design, TDD, Composer, dll.

Buku ini saya tulis untuk programmer pemula di framework Laravel, namun telah ada sedikit pengalaman menggunakan framework lain. Harapannya jika pembaca telah terbiasa menggunakan framework PHP lain, membaca buku ini sekaligus akan menjelaskan letak kemudahan Laravel untuk pengembangan aplikasi PHP.

Metode Penulisan

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.

Untuk memudahkan pembaca, source code dari aplikasi di buku ini dapat di download dari [bitbucket¹](https://bitbucket.org/rahmatawaludin/larapus).

Membaca sample source code

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

¹<https://bitbucket.org/rahmatawaludin/larapus>

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. Code tersebut sengaja tidak ditulis karena bukan bagian dari pembahasan.

Code yang ditulis tebal menandakan code yang baru ditambahkan.

Code yang bergaris menandakan bahwa code 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 dimana pun di terminal.

Ketika menjalankan perintah dari terminal, terkadang outputnya disertakan pada baris selanjutnya. Ciri dari output adalah menggunakan // diawali baris. Misalnya seperti ini:

```
php artisan -V  
// Laravel Framework version 5.3.4
```

Pada syntax diatas, output dari perintah `php artisan -V` adalah `Laravel Framework version 5.3.4`.

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. Bagaimanapun caranya. Jika Anda sudah terlanjur membajak buku ini, silahkan membayar dengan transfer ke BNI 0186355188 an Rahmat Awaludin (harga sesuai di [web²](#)). 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.

²<http://leanpub.com/bukularavel>

- 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.com
- FB : www.facebook.com/rahmat.awaludin
- Twitter : <https://twitter.com/rahmatawaludin>
- linkedin : www.linkedin.com/in/rahmatawaludin
- Phone : +62878 2225 0272

1. Hari 1 : Instalasi dan Konfigurasi Laravel

Laravel sangat mudah dikonfigurasi untuk mengembangkan web app. Pada bagian ini kita akan bahas apa saja yang harus dipersiapkan untuk memulai menggunakan framework Laravel. Untuk memudahkan pembahasan, kita akan menggunakan penjelasan yang sesederhana mungkin untuk memahami setiap teknik atau istilah yang sering digunakan di Laravel.

1.1 Text Editor

Dalam membuat aplikasi Laravel, saya sarankan menggunakan teks editor [Sublime Text 3](#)¹ dengan plugin [Laravel Blade Highlighter](#)² untuk menampilkan syntax highlighting blade dan [Emmet](#)³ untuk mempercepat mengetik HTML. Jika Anda lebih menyukai IDE, saya sarankan menggunakan [PHPStorm](#)⁴ atau [AksiIDE](#)⁵ karya om [Luri Darmawan](#)⁶, seorang anak bangsa yang merupakan member yang disegani di grup [PHP Indonesia](#)⁷. Saya sendiri, menggunakan [Vim](#)⁸ (lebih tepatnya [neovim](#)⁹) selama menulis buku ini.

Jika akan menggunakan Sublime Text, pastikan untuk menginstal package control agar dapat menginstal package. Panduan cara menginstal package control dapat dibaca di <https://sublime.wbond.net/installation>¹⁰.

¹www.sublimetext.com

²<http://github.com/Medalink/laravel-blade>

³<http://emmet.io>

⁴<http://www.jetbrains.com/phpstorm>

⁵<http://aksiide.com>

⁶<https://www.facebook.com/luridarmawan>

⁷<https://www.facebook.com/groups/35688476100>

⁸<http://www.vim.org>

⁹<https://neovim.io/>

¹⁰<https://sublime.wbond.net/installation>

Untuk menginstal package di Sublime Text dengan package control dapat dibaca di dokumentasi¹¹.



Tahapan instalasi plugin Sublime Text ini opsional. Jika bingung, tidak perlu pusing memikirkan langkah ini. Meskipun tidak dilakukan, proses belajar Laravel tidak akan terganggu. Gunakan text editor yang paling dikuasai saja.

1.2 Kebutuhan Sistem

Laravel mendukung penggunaan web server apache dan nginx. Pada buku ini, saya menggunakan web server Apache. Pastikan PHP yang Anda gunakan sudah versi 5.6.4 keatas dengan ekstensi OpenSSL, PDO, Mbstring dan Tokenizer. Saya sendiri menggunakan [MAMP¹²](#) untuk OSX, jika Anda pengguna windows bisa menggunakan [XAMPP¹³](#) (pastikan selalu menggunakan versi terbaru).

Untuk database Laravel dapat menggunakan database MySQL, PostgreSQL, SQLServer atau SQLite.

1.3 Composer

Untuk menginstall laravel kita akan menggunakan composer. Composer adalah aplikasi yang digunakan untuk mengatur package-package dalam mengembangkan sebuah web dengan PHP. Jika dulu, mungkin Anda mengenal yang namanya [PEAR¹⁴](#), composer tuh mirip-mirip PEAR lah.

Anggaplah kita belum kenal dengan PEAR/Composer. Jika kita akan mengembangkan sebuah aplikasi web dan membutuhkan library untuk user management misalnya ‘UserAuth’ maka kita akan download dari webnya, letakkan di folder tertentu (misalnya library), kemudian me-load dengan require atau include pada class yang kita butuhkan.

¹¹<https://sublime.wbond.net/docs/usage>

¹²www.mamp.info

¹³www.apachefriends.org

¹⁴pear.php.net

Setidaknya ada beberapa masalah dari solusi ini:

- Bagaimana jika web kita membutuhkan lebih dari satu library, misalnya 40 library? Mau download satu-persatu?
- Bagaimana jika library UserAuth bergantung dengan library lain? misalnya SessionManager dan SessionManager juga bergantung kepada library Session. Dan seterusnya, dan seterusnya..

Untuk memahami composer lebih lanjut, kita dapat mengunjungi [dokumentasi resmi composer¹⁵](#).

1.3.1 Install Composer

Instalasi composer agak berbeda untuk OS unix (Linux, OSX, dll) dan Windows, saya jelaskan masing-masing:

1.3.1.1 Windows

Cukup download [composer-setup.exe¹⁶](#) dan jalankan file instalasi.

1.3.1.2 Unix

Jalankan terminal dan masukkan perintah berikut:

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

Pada windows maupun unix kita dapat mengetes apakah composer sudah terinstall dengan perintah:

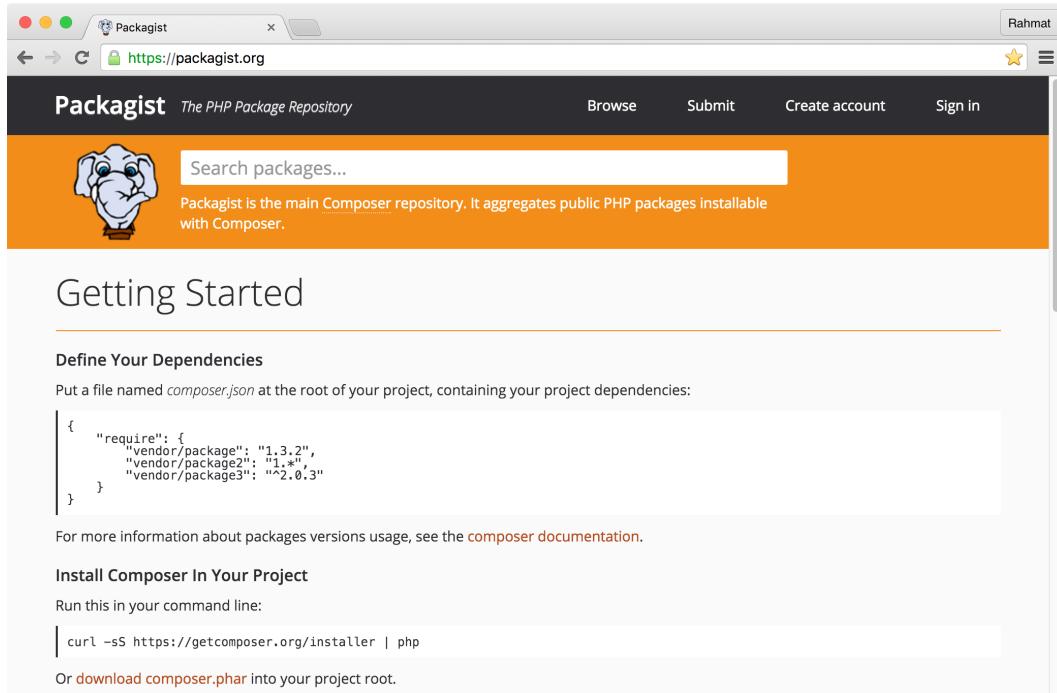
```
composer -V
```

¹⁵<https://getcomposer.org/doc/00-intro.md>

¹⁶<https://getcomposer.org/Composer-Setup.exe>

1.3.2 Penggunaan Composer

Secara default, composer akan menggunakan package yang teregister di packagist.org¹⁷. Selain packagist, kita juga dapat menyiapkan repositori package private menggunakan [satis](#)¹⁸.



The screenshot shows the Packagist website (<https://packagist.org>) with a dark header bar containing the site name, a search bar, and navigation links for Browse, Submit, Create account, and Sign in. A user profile 'Rahmat' is visible in the top right. Below the header is an orange banner featuring a cartoon elephant icon and the text: 'Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.' The main content area is titled 'Getting Started' and includes sections for defining dependencies (with a code snippet example) and installing Composer.

Getting Started

Define Your Dependencies

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": "^2.0.3"
  }
}
```

For more information about packages versions usage, see the [composer documentation](#).

Install Composer In Your Project

Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Or [download composer.phar](#) into your project root.

Packagist.org

Composer menggunakan file dengan format [JSON](#)¹⁹. JSON merupakan format standar untuk menyimpan data `name => value` yang umum digunakan untuk transfer data. Contoh syntax JSON seperti ini:

¹⁷<http://packagist.org>

¹⁸<https://github.com/composer/satis>

¹⁹<http://json.org>

contoh struktur JSON

```
1  {
2      name1 : {
3          subname1 : value,
4          subname2 : value
5      },
6      name2 : value
7 }
```

Composer menggunakan format json ini pada file bernama composer.json (file ini harus kita buat). Berikut contoh isi composer.json :

composer.json

```
1  {
2      "require": {
3          "monolog/monolog": "1.0.*"
4      }
5 }
```

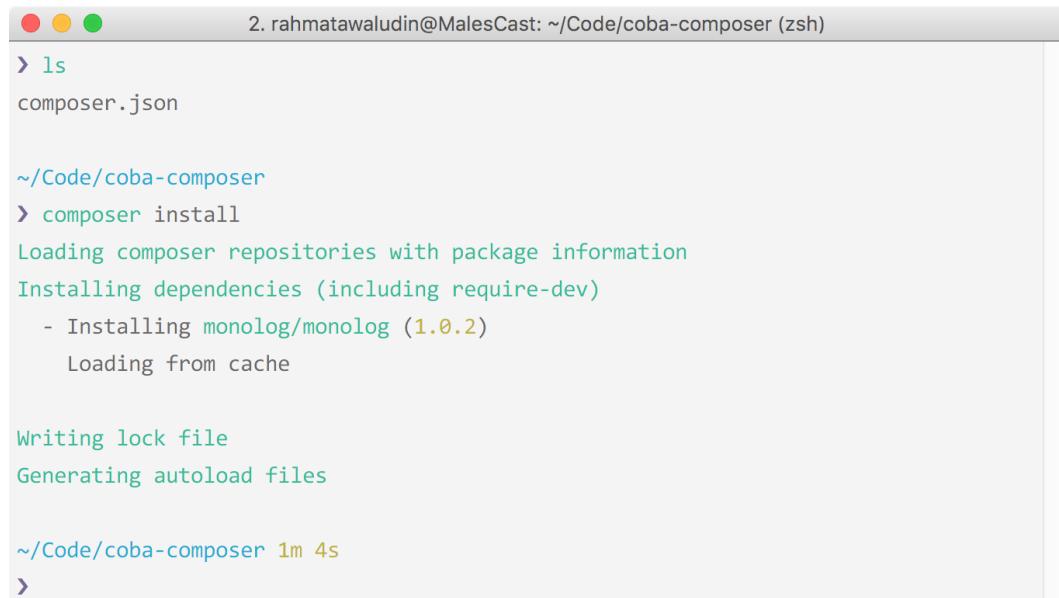
File composer.json berfungsi untuk menentukan library apa saja yang akan kita install pada project PHP yang kita buat. Pada syntax diatas, pada bagian require kita masukkan nama package yang kita butuhkan (monolog/monolog) dan versi yang diinginkan (1.0.*).

Monolog adalah sebuah library yang biasa digunakan untuk bekerja dengan file log. Disini, instalasi Monolog hanya sebagai contoh penggunaan composer untuk menginstal library PHP, tidak ada hubungannya dengan Laravel.

1.3.2.1 Install Package

Untuk menginstall package dengan composer, pindahkan file composer.json diatas ke sebuah folder. Disini saya akan menggunakan folder di ~/Code/coba-composer. Untuk pengguna Windows, misalnya dapat menggunakan C:\coba-composer. Lalu jalankan perintah berikut di dalam folder tersebut:

```
composer install
```



```
2. rahmatawaludin@MalesCast: ~/Code/coba-composer (zsh)
> ls
composer.json

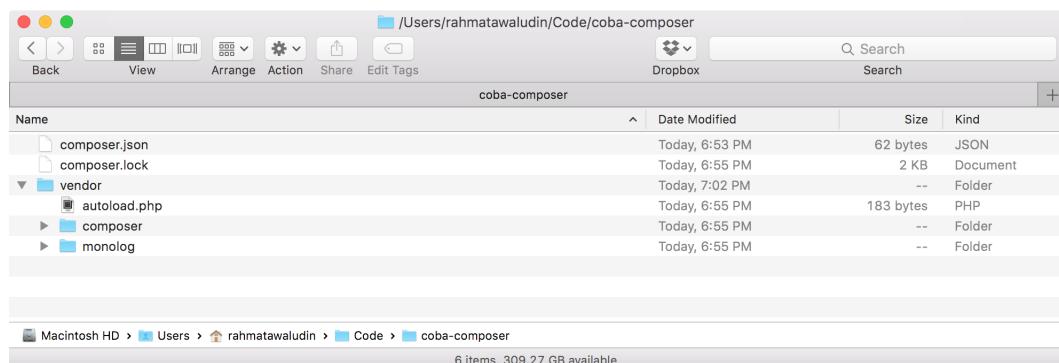
~/Code/coba-composer
> composer install
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing monolog/monolog (1.0.2)
  Loading from cache

Writing lock file
Generating autoload files

~/Code/coba-composer 1m 4s
> _
```

composer install

Perintah diatas akan melakukan instalasi package aplikasi yang kita tulis di bagian require. Cobalah buka folder yang kita gunakan, akan terlihat isian seperti berikut:



Isi folder setelah composer install

- Folder vendor menyimpan package yang dibutuhkan, sebagaimana yang dit-

ulus di bagian require.

- File vendor/autoload.php dapat digunakan untuk mendapatkan fitur autoloading.
- File composer.lock berfungsi mencatat versi package yang saat ini sedang kita gunakan.

1.3.2.2 Update package

Jika package baru telah ditambah pada bagian require atau versi package yang digunakan dirubah, gunakan perintah ini untuk memperbarui package:

```
composer update
```

Untuk menginstall package baru, kita juga dapat menjalankan perintah:

```
composer require <nama-package>
```

Perintah composer lainnya dapat dilihat dengan perintah :

```
composer --help
```

Atau cek di [manual composer²⁰](#).

Untuk memahami lebih jauh tentang composer, bisa juga menonton video yang telah saya buat di [Malescast²¹](#).

1.4 Instalasi Laravel

Sebagaimana disampaikan di bagian sebelumnya, Laravel diinstall menggunakan composer. Gunakan perintah ini untuk membuat project laravel di folder webapp dengan versi laravel 5.3:

²⁰<https://getcomposer.org/doc>

²¹<https://malescast.com/Teknik-PHP-Modern/dependency-management-dengan-composer>

```
1 composer create-project laravel/laravel=5.3 --prefer-dist webapp
```



Jika kita ingin menginstall versi terbaru dari Laravel, gunakan perintah:

```
1 $ composer create-project laravel/laravel --prefer-dist webapp
```

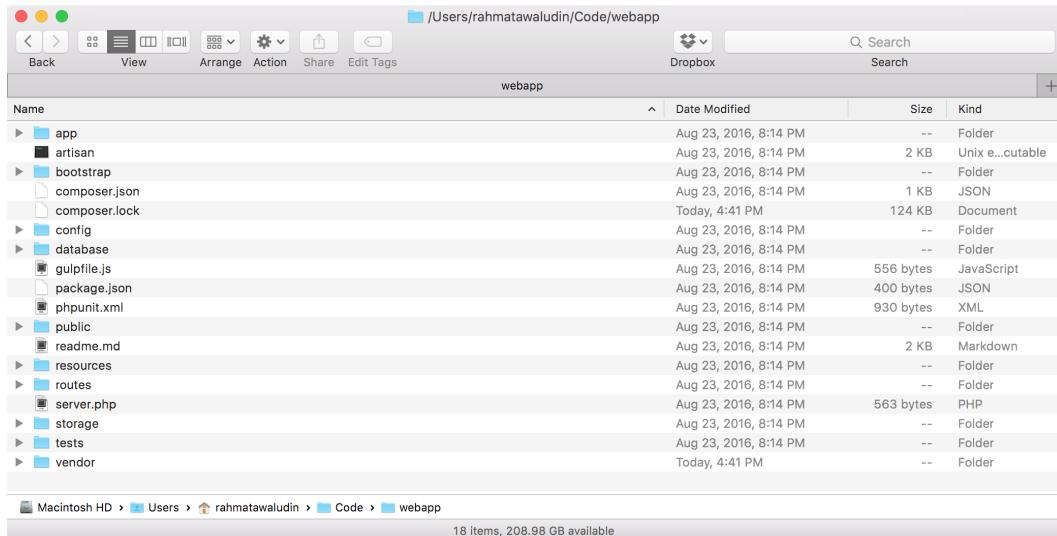
Akan muncul output seperti ini:

```
Installing laravel/laravel (v5.3.0)
- Installing laravel/laravel (v5.3.0)
  Downloading: 100%

Created project in webapp
> php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)

...
Writing lock file
Generating autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
Generating optimized class loader
> php artisan key:generate
Application key [base64:+HH0g9mCiJvBEWYkPB9NwZMAIDBNm7dZGHmDPqAJKrk=] set successfully.
```

Jika Anda telah mendapatkan output seperti ini, itu tandanya Laravel telah berhasil didownload. Ini struktur folder yang akan kita dapatkan:



Struktur folder Laravel

Kita juga dapat mengecek versi Laravel yang terinstall dengan perintah:

```
php artisan -V  
// Laravel Framework version 5.3.4
```

Terlihat pada output diatas, kita mendapatkan versi 5.3.4.

1.5 Konfigurasi

Setelah Laravel terinstall pastikan folder `storage` dan `bootstrap/cache` dapat diakses oleh web server. Seharusnya, ini sudah diset dengan benar. Jika menggunakan unix, kita dapat menjalankan perintah berikut untuk mengeset ulang hak akses kedua folder tersebut

```
sudo chmod -R 777 storage  
sudo chmod -R 777 bootstrap/cache
```

Semua konfigurasi Laravel disimpan di folder config. Misalnya, konfigurasi database akan disimpan di file config/database.php.

Dalam mengisi file konfigurasi Laravel, kita dapat langsung mengisinya atau menggunakan file .env. Misalnya, untuk mengisi konfigurasi database pada koneksi dengan nama mysql, kita akan menemui isian seperti ini:

config/database.php

```
'database' => env('DB_DATABASE', 'forge'),  
'username' => env('DB_USERNAME', 'forge'),  
'password' => env('DB_PASSWORD', ''),
```

Untuk mengisi ketiga field tersebut, kita dapat langsung mengubah isinya atau menggunakan file .env. Caranya, kita buat isian seperti ini (sesuaikan dengan konfigurasi database):

.env

```
DB_DATABASE=Larapus  
DB_USERNAME=rahmat  
DB_PASSWORD=r4ha51a
```

Dalam development yang menggunakan version control misalnya Git, file ini tidak akan di *commit*. Sehingga, isian file .env akan berbeda tergantung *environment* mesin tiap developer.



Git?

Git merupakan salah satu tools yang bisa kita gunakan untuk menyimpan source code dengan lebih terstruktur. Jika belum paham Git, sebaiknya segera belajar setelah menyelesaikan buku ini. Setiap developer saat ini, wajib menguasai Git.

1.6 PHP builtin web server

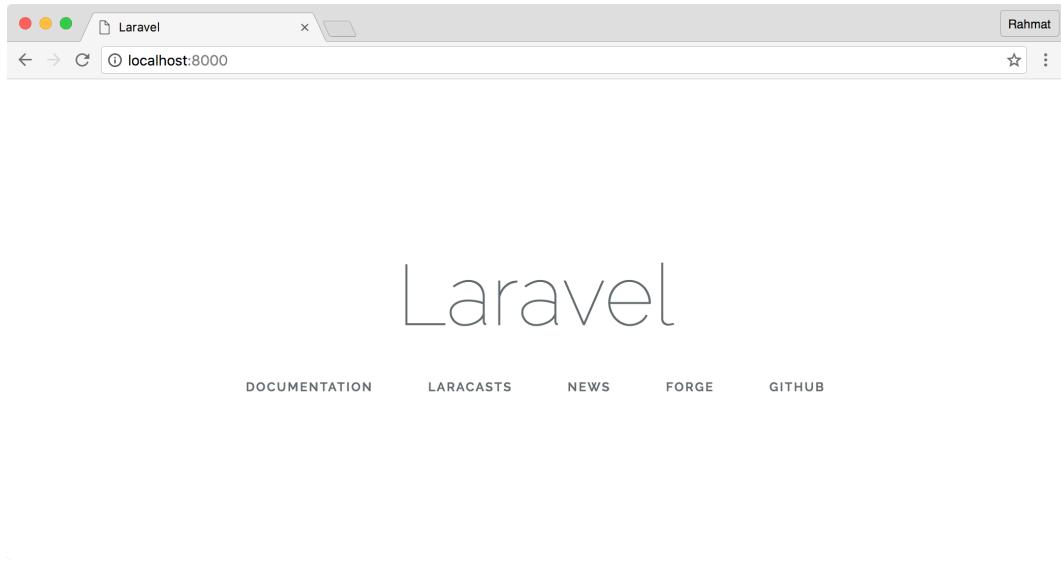
Selama proses pengembangan, kita dapat menggunakan builtin web server untuk mengakses Laravel. Dengan teknik ini, kita tidak perlu menggunakan web server seperti Apache atau Nginx.

Jalankan perintah berikut di folder webapp:

```
php artisan serve  
// Laravel development server started on http://localhost:8000/
```

Untuk mengaksesnya, kita dapat mengunjungi `http://localhost:8000`.

Akan muncul tampilan seperti berikut:



Berhasil menjalankan simple web server

Untuk mematikan server, dapat menggunakan `ctrl+c` di terminal.

Salah satu kekurangannya adalah kita harus *me-restart* server ketika merubah file konfigurasi (`.env`).

1.7 Konfigurasi Database

Pada buku ini, kita akan menggunakan database MySQL. Cara yang paling mudah untuk pengguna OSX adalah dengan menggunakan [Mamp²²](#). Untuk pengguna Linux atau Windows bisa menggunakan [Xampp²³](#).

²²<https://www.mamp.info>

²³<https://www.apachefriends.org>

Setelah server MySQL berjalan, buatlah database baru (bisa menggunakan [phpMyAdmin²⁴](#) atau [Sequel Pro²⁵](#)). Catat nama database, username dan password yang akan digunakan. Kemudian kita isi field berikut pada file .env (sesuaikan dengan data Anda):

.env

```
DB_DATABASE=Larapus
DB_USERNAME=rahmat
DB_PASSWORD=r4ha51a
```

Setelah melakukan perubahan ini, pastikan untuk me-restart server.

1.7.1 Error koneksi database?

Menggunakan teknik ini, terkadang menyebabkan koneksi database error dengan pesan “*Can't connect to local MySQL server through socket ...*”. 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:

²⁴<https://www.phpmyadmin.net>

²⁵<http://www.sequelpro.com>

```
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, kita tambahkan isian `unix_socket` pada konfigurasi koneksi database dengan alamat yang kita temukan. 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',
```

],
....



Sudah cukup.

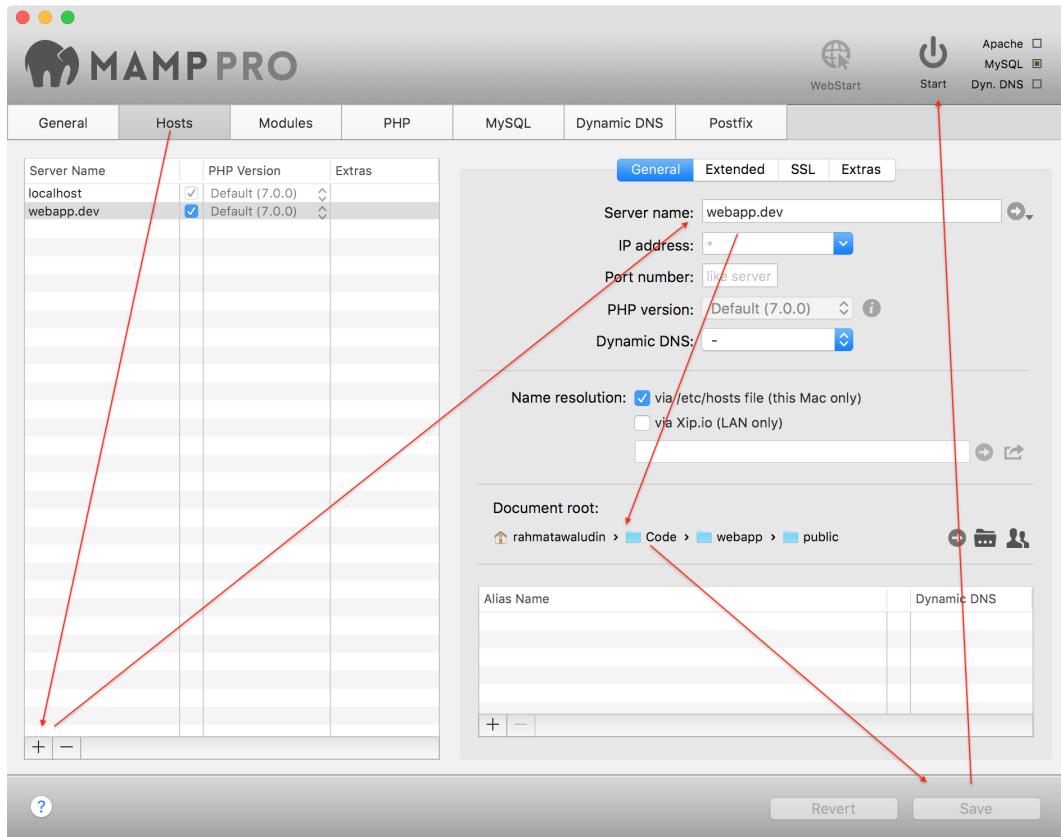
Menggunakan teknik builtin web server, sudah cukup untuk mempelajari isi buku ini. Topik selanjutnya merupakan tambahan bagi yang ingin mendalami bagaimana menjalankan virtual host. Saran saya, sekarang langsung lanjut ke bab selanjutnya dan kunjungi lagi dua topik dibawah setelah buku selesai.

1.8 Virtual Host

Menggunakan virtual host, aplikasi dapat diakses dengan url seperti `http://webapp.dev`, `http://www.webapp.com`, dan sebagainya walupun masih berada di lokal. Berikut cara membuat virtual host:

1.8.1 MAMP

1. Menggunakan MAMP Pro, buka menu **Hosts**
2. Klik tombol **[+]**
3. Isi bagian **Server name** dengan url yang kita inginkan
4. Isi **Document root** dengan alamat folder public di webapp
5. Klik **Save**
6. Klik **Start** untuk merestart server apache.



Konfigurasi virtual host Mamp

1.8.2 XAMPP

1. Buka file hosts yang ada di alamat C:\WINDOWS\system32\drivers\etc\hosts
2. Di bagian paling bawah tambahkan alamat IP Address localhost 127.0.0.1 dan nama domain yang dibuat misalnya webapp.dev

C:\WINDOWS\system32\drivers\etc\hosts

```
1  ....  
2  127.0.0.1    webapp.dev  
3  ....
```



Catatan untuk Windows Terkadang file hosts ini tidak bisa langsung disimpan di tempatnya. Solusinya, simpan terlebih dahulu file ini di desktop, lantas *replace* file aslinya.



3. Buka file httpd.conf yang ada di alamat C:\xampp\apache\conf\httpd.conf
4. Cari bagian Directory, jika aplikasi kita berada di C:/xampp/htdocs/webapp isi seperti ini

C:\xampp\apache\conf\httpd.conf

```
1 <Directory "C:/xampp/htdocs/webapp/public">  
2   Options Indexes FollowSymLinks Includes ExecCGI  
3   AllowOverride All  
4   Order allow,deny  
5   Allow from all  
6   Require all granted  
7 </Directory>
```

5. Buka file httpd-vhosts.conf yang ada di alamat C:\xampp\apache\conf\extra
6. Tambahkan setingan di bawah ini untuk membedakan website yang dipanggil dengan localhost dan website yang dipanggil dengan virtual host

C:\xampp\apache\conf\extra\httpd-vhosts.conf

```
1 NameVirtualHost *:80
2
3 #VirtualHost untuk webapp.dev
4
5 <VirtualHost *:80>
6     DocumentRoot C:/xampp/htdocs/webapp/public
7     ServerName webapp.dev
8 </VirtualHost>
9
10 #Untuk localhost yang biasa
11
12 <VirtualHost *:80>
13     DocumentRoot C:/xampp/htdocs
14     ServerName localhost
15 </VirtualHost>
```

7. Restart Apache pada XAMPP Control Panel dengan klik tombol stop kemudian klik tombol start.

Setelah berhasil, Anda dapat mengakses aplikasi di <http://webapp.dev>.



DOCUMENTATION LARACASTS NEWS FORGE GITHUB

Berhasil setup virtualhost



Homestead?

Ada satu teknik lagi untuk menjalankan Laravel yaitu menggunakan homestead. Teknik ini sangat bermanfaat ketika penggerjaan Laravel dalam sebuah tim. Tujuannya agar konfigurasi tiap anggota tim selalu sama. Pembahasan homestead tidak dibahas dibuku ini, untuk memahaminya dapat mengunjungi [dokumentasi²⁶](https://laravel.com/docs/5.3/homestead) atau baca di buku [Menyelami Framework Laravel²⁷](#).

1.9 Ringkasan

Di Hari 1 ini, saya harap Anda telah memahami bagaimana melakukan setup sebuah project Laravel, poin-poin yang telah kita bahas yaitu:

- Text Editor yang digunakan
- Penggunaan composer untuk development php modern
- Instalasi laravel
- Konfigurasi virtualhost

Pada hari 2 kita akan mempelajari konsep Routing dan MVC di Laravel. Semangat!
:)

²⁶<https://laravel.com/docs/5.3/homestead>

²⁷<https://leanpub.com/bukularavel>

2. Hari 2 : Routing, MVC dan Authentikasi

Really, the web is pretty simple stuff. It's just request - response. I told myself this over and over again when building Laravel. I just want a simple way to catch requests and send out responses. That's it.

Why Laravel? - Taylor Otwell¹

Pada hari 2, kita akan belajar struktur dasar dari sebuah aplikasi yang dibangun dengan framework Laravel. Untuk memudahkan, kita akan menggunakan aplikasi webapp yang dibuat di hari 1.

2.1 Routing

Jika diibaratkan sebuah Mall, routing itu ibarat Bagian Informasi. Jika kita bertanya lokasi toko Sepatu Wiwi, maka Bagian Informasi akan mengarahkan kita ke toko tersebut.

Di Laravel routing diatur pada 3 file:

- `routes/console.php`: routing command yang berjalan di terminal.
- `routes/api.php`: routing untuk pembuatan API.
- `routes/web.php`: routing untuk web biasa.

Pada aplikasi yang lebih kompleks, kita dapat membuat routing pada file lain selain 3 file diatas.

Untuk pembahasan di buku ini, kita hanya menggunakan `routes/web.php`.

Berikut isian default dari `routes/web.php`:

¹<http://taylorotwell.tumblr.com/post/21038245018/why-laravel>

routes/web.php

```
<?php  
...  
Route::get('/', function () {  
    return view('welcome');  
});
```

Misalnya, jika kita ingin membuat halaman statis yang bisa diakses di /about, tambahkan isian seperti ini pada routes/web.php:

routes/web.php

```
...  
Route::get('/about', function() {  
    return '<h1>Halo</h1>'  
        . 'Selamat datang di webapp saya<br>'  
        . 'Laravel, emang keren.';  
});
```

Maka, kita dapat akses /about.



Halo

Selamat datang di webapp saya
Laravel, emang keren.

Berhasil membuat route about

Mari kita perhatikan syntax Route::get('/about', function() { ... }).

- Parameter `get` menjelaskan jenis HTTP Verb yang diizinkan pada route. Selain `get`, kita juga bisa menggunakan `post`, `patch`, `put`, `delete` dan `options`.
- `/about` merupakan URL untuk diakses.
- `function() { ... }` merupakan closure (anonymous function) yang akan memberikan jawaban atas request. Selain menggunakan closure, kita juga dapat mengarahkan request ke method pada sebuah controller.

Untuk mengecek route apa saja yang telah kita buat, dapat menggunakan perintah berikut di terminal (dari folder project):

```
php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	about		Closure	web
	GET HEAD	api/user		Closure	api,auth:api

Mengecek route yang telah dibuat

Fitur routing ini sangat banyak, dan kita tidak akan membahas semua fiturnya. Fitur lainnya akan saya jelaskan selama kita membuat sample aplikasi. Jika tertarik mempelajari lebih jauh tentang routing, silahkan baca di [dokumentasi routing](#)².

2.2 MVC

Laravel mendukung penuh pembuatan aplikasi dengan arsitektur **Model View Controller (MVC)**³ untuk memisahkan logic *manipulasi data*, *antarmuka pengguna* dan *kontrol aplikasi*. Mari kita bahas bagaimana Laravel mengimplementasikannya.

²<https://laravel.com/docs/5.3/routing>

³<http://id.wikipedia.org/wiki/MVC>

2.3 Model

Model mewakili struktur data yang kita gunakan dalam aplikasi. Model dibuat berdasarkan objek dalam aplikasi kita. Misalnya, jika kita membuat aplikasi blog, maka artikel dan komentar dapat menjadi model. Ketika kita membahas model, pasti akan membahas tentang [database⁴](#). Laravel memiliki fitur menarik untuk manajemen database yaitu [migrations](#) dan [seeding⁵](#).

2.3.1 Migrations

Laravel memudahkan kita untuk membuat struktur database yang dapat disimpan dalam [VCS⁶](#) semisal [Git⁷](#). Dengan menggunakan migrations, perubahan struktur database selama pengembangan aplikasi dapat tercatat dan terdistribusikan ke semua anggota tim.

Kita **tidak wajib** menggunakan migration selama mengembangkan web dengan Laravel. Namun, menggunakan migration akan mempermudah kita dalam pengembangan untuk jangka panjang.

Mari kita buat migrations untuk membuat table Post dengan struktur:

Struktur table Post

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_- increment
title	varchar(255)	NO	UNI	NULL	
content	varchar(255)	NO	UNI	NULL	
created_at	timestamp	NO		0000-00-00 00:00:00	
updated_at	timestamp	NO		0000-00-00 00:00:00	

⁴<https://laravel.com/docs/5.3/database>

⁵<https://laravel.com/docs/5.3/migrations>

⁶<https://en.wikipedia.org/wiki/VCS>

⁷<https://git-scm.com>

1. Buka terminal, masuk ke folder webapp, jalankan perintah berikut:

```
php artisan make:migration create_posts_table --create=posts
// Created Migration: xxxx_xx_xx_xxxxxx_create_posts_table
```

Opsi --create=posts akan membuat file migration dengan template untuk membuat table dengan nama posts. Ada juga opsi --table yang akan membuat template untuk melakukan modifikasi table. Tapi, kita tidak bisa menggunakan dua opsi itu secara bersamaan. Jika kita tidak menggunakan opsi sama sekali, Laravel akan membuat file migration tanpa template untuk membuat / menggunakan table.

2. Perintah diatas akan menghasilkan sebuah file, misalnya dengan nama database/migrations/yyyy_xx_xx_xxxxxx_create_posts_table.php. Ubahlah isian file tersebut menjadi:

database/migrations/yyyy_xx_xx_xxxxxx_create_posts_table.php

```
<?php

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

class CreatePostsTable extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title')->unique();
            $table->string('content');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('posts');
    }
}
```

1. Pada method up diatas Laravel akan membuat table posts dengan struktur yang telah kita tentukan. Untuk menambah field dengan tipe lainnya, kita dapat cek di [dokumentasi⁸](#). Sedangkan, pada method down, laravel akan menghapus table posts.
2. Jalankan perintah ini untuk melakukan migrasi :

```
php artisan migrate
// Migration table created successfully.
// Migrated: 2014_10_12_000000_create_users_table
// Migrated: 2014_10_12_100000_create_password_resets_table
// Migrated: xxxx_xx_xx_xxxxxx_create_posts_table
```

Pastikan database sudah dikonfigurasi sesuai [petunjuk di bab sebelumnya](#) sebelum menjalankan perintah diatas.

3. Cek pada database, akan terdapat table migrations dan posts. Table migrations berfungsi untuk mencatat migrasi database yang telah kita lakukan. Table posts adalah table yang didefinisikan di file migrasi yang telah kita buat. Disini saya menggunakan aplikasi [Sequel Pro⁹](#) di Mac untuk melihat struktur database. Untuk pengguna Windows atau Linux, dapat menggunakan [phpMyAdmin¹⁰](#) (yang sudah bawaan Xampp) atau [MySQL Workbench¹¹](#).

⁸<https://laravel.com/docs/5.3/migrations#creating-columns>

⁹<http://www.sequelpro.com>

¹⁰<https://www.phpmyadmin.net>

¹¹<https://www.mysql.com/products/workbench>

The screenshot shows the MySQL Workbench interface for the 'posts' table in the 'webapp' database. The table has five columns: id, title, content, created_at, and updated_at. The 'id' column is defined as INT(10) unsigned with a primary key constraint (PRI). The 'title' and 'content' columns are VARCHAR(255). The 'created_at' and 'updated_at' columns are TIMESTAMP. There are two indexes: a primary key index ('PRIMARY') on the 'id' column and a unique index ('posts_title_unique') on the 'title' column.

Field	Type	Len...	Unsigned	Zerofill	Binary	Allow Null	Key	Defa...	Extra	Encodi...	Collation
id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_in...	<input type="checkbox"/>	<input type="checkbox"/>
title	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	UNI		None	<input type="checkbox"/>	UTF-8 <input type="checkbox"/> utf8_uni <input type="checkbox"/>
content	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			None	<input type="checkbox"/>	UTF-8 <input type="checkbox"/> utf8_uni <input type="checkbox"/>
created_at	TIMESTAMP		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	<input type="checkbox"/>
updated_at	TIMESTAMP		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	<input type="checkbox"/>

TABLE INFORMATION

- created: 3/25/16
- engine: InnoDB
- rows: 3
- size: 16.0 KIB
- encoding: utf8
- auto_increment: 4

INDEXES

Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Comment
0	PRIMARY	1	id	A	3	NULL	NULL	
0	posts_title_unique	1	title	A	3	NULL	NULL	

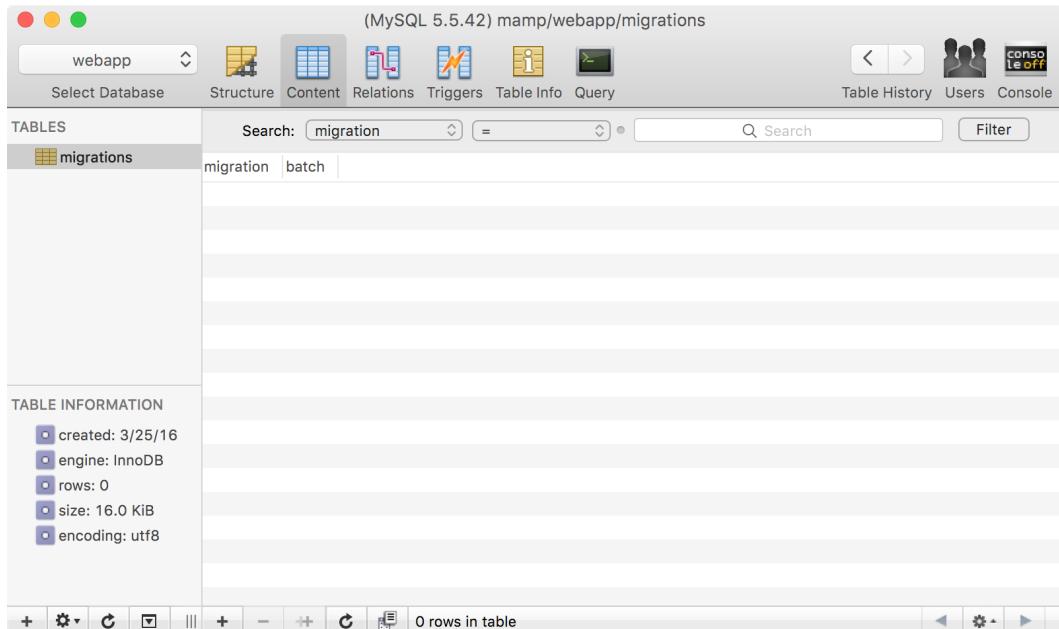
Migrasi berhasil

Pada migration default, akan muncul juga table users dan password_resets yang akan digunakan untuk authentikasi. Dua table ini akan kita bahas di pembahasan selanjutnya.

- Untuk mendemonstrasikan kegunaan migration, mari kita lakukan rollback untuk meng-undo migrasi yang telah kita lakukan. Jalankan perintah ini :

```
php artisan migrate:rollback
// Rolled back: xxxx_xx_xx_xxxxxx_create_posts_table
// Rolled back: 2014_10_12_100000_create_password_resets_table
// Rolled back: 2014_10_12_000000_create_users_table
```

- Cek kembali database, maka table posts akan terhapus.



Sekarang berhenti sejenak, renungkan apa yang telah kita lakukan. Dengan menggunakan migrasi, struktur database dapat lebih mudah dipahami dan di-maintenance. Bandingkan jika menggunakan import/export file .sql. Tentunya cukup merepotkan jika struktur database sering berubah ketika develop aplikasi. Saran saya, gunakan migrasi untuk manajemen database selama pengembangan aplikasi dan export/import file sql ketika produksi.

Yang lebih powerfull, migrasi tidak hanya bisa menambah/menghapus table. Migrasi juga memungkinkan kita merubah struktur suatu table, misalnya menambah/hapus/ubah suatu kolom. Jika ingin belajar lebih lanjut tentang migrasi, kunjungi [dokumentasi migrasi¹²](#).

2.3.2 Database Seeder

Terkadang ketika kita mengembangkan aplikasi dibutuhkan sample data. Bisa saja sample data tersebut kita inject langsung ke database. Namun, cukup merepotkan

¹²<http://laravel.com/docs/migrations>

jika kita ingin meng-*inject* banyak data. Terlebih jika kita ingin me-*reset* database ke kondisi sesuai sample data ini. Database seeder merupakan jawaban untuk masalah ini.

Mari kita buat database seeder untuk table posts:

1. Generate file migration baru dengan perintah:

```
php artisan make:seeder PostsTableSeeder  
// Seeder created successfully.
```

2. Perintah diatas akan membuat file baru di database/seeds/PostsTableSeeder.php. Ubahlah isinya menjadi:

database/seeds/PostsTableSeeder.php

```
<?php  
  
use Illuminate\Database\Seeder;  
  
class PostsTableSeeder extends Seeder  
{  
    public function run()  
    {  
        $posts = [  
            ['title'=>'Tips Cepat Nikah', 'content'=>'lorem ipsum'],  
            ['title'=>'Haruskah Menunda Nikah?', 'content'=>'lorem ipsum'],  
            ['title'=>'Membangun Visi Misi Keluarga', 'content'=>'lorem ipsum']  
        ];  
  
        // masukkan data ke database  
        DB::table('posts')->insert($posts);  
    }  
}
```

3. Agar file migration diatas bisa dijalankan, kita harus melakukan perubahan pada file database/seeds/DatabaseSeeder.php. Caranya, kita tambahkan baris berikut pada method run:

database/seeds/DatabaseSeeder.php

```
....  
public function run()  
{  
    ....  
    $this->call(PostsTableSeeder::class);  
}  
....
```

- Untuk melakukan *seeding*, jalankan perintah ini:

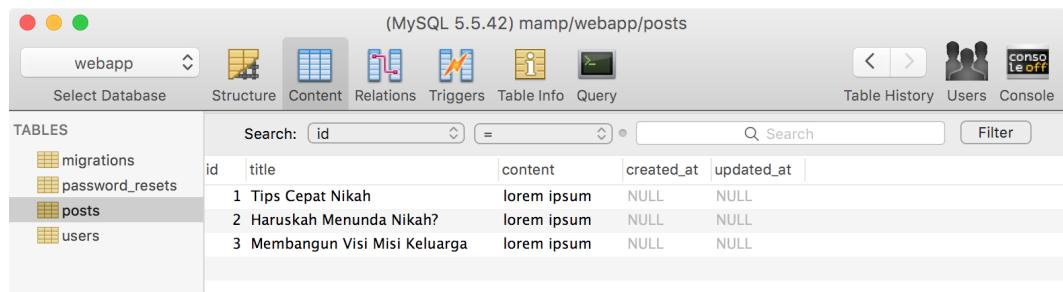
```
php artisan migrate  
php artisan db:seed  
// Seeded: PostsTableSeeder
```

Perintah diatas akan melakukan migrasi struktur database dan melakukan proses *insert* untuk sample data. Kedua kombinasi perintah ini akan sangat sering kita lakukan, Laravel bahkan menyediakan perintah untuk menjalankan dua perintah diatas sekaligus:

```
php artisan migrate:refresh --seed
```

Dengan perintah ini, kita akan melakukan rollback, migrate dan seeding database.

- Cek kembali database, maka sample data telah masuk ke dalam database.



The screenshot shows the MySQL Workbench interface for the 'posts' table. The table has columns: id, title, content, created_at, and updated_at. The data is as follows:

	id	title	content	created_at	updated_at
1	1	Tips Cepat Nikah	lorem ipsum	NULL	NULL
2	2	Haruskah Menunda Nikah?	lorem ipsum	NULL	NULL
3	3	Membangun Visi Misi Keluarga	lorem ipsum	NULL	NULL

Database Seeding berhasil

2.3.3 Membuat Model

Model dalam Laravel dibuat dengan melakukan *extends* class `Illuminate\Database\Eloquent\Model`. Untuk table posts, kita akan membuat model dengan nama Post. Penamaan model dan table ini merupakan aturan di Laravel dimana nama table harus plural dan model harus singular.

Untuk membuat model Post, jalankan perintah berikut:

```
php artisan make:model Post
// Model created successfully.
```

Akan muncul file baru di `app/Post.php`.

`app/Post.php`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    //
}
```



Nama table tidak mengikuti aturan?

Kasus ini bisa diselesaikan dengan menambahkan `protected attribute table` dengan isi nama table yang kita inginkan. Detailnya bisa dilihat di [dokumentasi](#)¹⁴.

¹³<https://laravel.com/api/5.3/Illuminate/Database/Eloquent/Model.html>

¹⁴<https://laravel.com/docs/5.3/eloquent#eloquent-model-conventions>

2.3.4 Mengakses model

Model dapat diakses dengan langsung memanggil class model tersebut dimanapun kita butuhkan. Eloquent merupakan ORM (Object Relational Mapper) yang *powerfull* untuk memanipulasi data. Berikut ini beberapa contoh penggunaan Eloquent:

1. Buat route /testmodel pada file routes/web.php dengan isi sebagai berikut:

routes/web.php

```
Route::get('/testmodel', function() {
    $query = /* isi sample query */;
    return $query;
});
```

2. Pada beberapa contoh dibawah, silahkan ubah /* isi sample query */ dengan contoh yang ingin dicoba. Untuk mengecek hasilnya, akses /testmodel dengan browser.

- Mencari semua model:

```
App\Post::all();
```

- Mencari model berdasarkan id:

```
App\Post::find(1);
```

- Mencari model berdasarkan title:

```
App\Post::where('title', 'like', '%cepat nikah%')->get();
```

- Mengubah record, (hapus semua isi function) :

```
$post = App\Post::find(1);
$post->title = "Ciri Keluarga Sakinah";
$post->save();
return $post;
```

- Menghapus record, (hapus semua isi function) :

```
$post = App\Post::find(1);
$post->delete();
// check data di database
```

- Menambah record, (hapus semua isi function) :

```
$post = new App\Post;
$post->title = "7 Amalan Pembuka Jodoh";
$post->content = "shalat malam, sedekah, puasa sunah, silaturahmi, senyum, doa, tobat";
$post->save();
return $post;
// check record baru di database
```



Penjelasan lengkap untuk beberapa syntax query berikut dapat ditemukan di dokumentasi query builder¹⁵ dan eloquent¹⁶.

2.4 View

View atau istilah lainnya *presentation logic* berfungsi untuk menampilkan data yang telah kita olah di *bussiness logic*. Laravel memudahkan kita untuk membuat view. Mari kita ubah route about yang sudah dibuat menjadi view:

1. Ubah route about menjadi:

`routes/web.php`

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

2. Buat file `resources/views/about.php` dengan isi:

¹⁵<https://laravel.com/docs/5.3/queries>

¹⁶<http://laravel.com/docs/eloquent>

resources/views/about.php

```
<html>
  <body>
    <h1>Halo</h1>
    Selamat datang di webapp saya.<br>
    Laravel, emang keren banget!
  </body>
</html>
```

3. Cek kembali route /about dan hasilnya akan berasal dari isi view.



Halo

Selamat datang di webapp saya.
Laravel, emang keren banget!

Berhasil menggunakan view

2.4.1 Template dengan Blade

Selain dengan memisahkan peletakan view pada file berbeda, Laravel juga lebih menekankan penggunaan view ini dengan templating. Dengan templating, developer akan “terpaksa” hanya menggunakan syntax untuk tampilan dan logic sederhana pada *view* nya. Templating pada Laravel menggunakan [Blade¹⁷](#).

Untuk menggunakan view dengan blade template, kita cukup merubah ekstensi file view menjadi .blade.php. Pada contoh file about.php, maka kita ubah menjadi about.blade.php untuk menggunakan template blade.

¹⁷<https://laravel.com/docs/5.3/blade>

2.4.1.1 Blade Syntax

Syntax yang paling sederhana dalam blade adalah {{ }} (double curly braces). Syntax ini dapat menggantikan fungsi <?php echo ;?> pada file view. Jadi, syntax {{ \$variabel }} akan berubah menjadi syntax <?php echo \$variable; ?>. Menggunakan teknik ini, setiap variable akan di *escape*, sehingga akan lebih aman. Jika hendak melakukan echo tag html, gunakan {!! !!}.

Selain echo sederhana, blade juga mendukung control structures semisal @if, @for, @foreach, @while, @unless, dll untuk templating. Silahkan baca di [dokumentasi resmi](#)¹⁸ untuk penjelasan lebih lengkapnya.

2.4.2 Form

Untuk membuat form di Laravel kita dapat menggunakan html biasa atau menggunakan package [laravelcollective/html](#)¹⁹. Menggunakan package, proses pembuatan form ini akan lebih mudah. Untuk menginstall package ini, ikuti langkah berikut:

1. Jalankan perintah berikut:

```
composer require laravelcollective/html=~5.3
```

2. Tambahkan isian berikut pada array providers di config/app.php:

config/app.php

```
....  
'providers' => [  
    ...  
    Collective\Html\HtmlServiceProvider::class,  
],  
....
```

3. Dan isian berikut pada array aliases:

¹⁸<https://laravel.com/docs/5.3/blade#control-structures>

¹⁹<https://laravelcollective.com/docs/5.3/html>

config/app.php

```
....  
'aliases' => [  
    ....  
    'Form' => Collective\Html\FormFacade::class,  
    'Html' => Collective\Html\HtmlFacade::class,  
],  
....
```

Syntax dasar untuk membuat form seperti ini:

```
{!! Form::open(['url' => 'post/save']) !!}  
//  
{!! Form::close() !!}
```

Berikut contoh untuk menampilkan label dengan *placeholder* E-Mail Address dan *class awesome*:

```
{!! Form::label('email', 'E-Mail Address', ['class' => 'awesome']) !!}
```

Untuk membuat input:

```
{!! Form::text('username') !!}
```

Penjelasan lebih lengkapnya, dapat diakses di [dokumentasi form²⁰](#).

2.4.3 Request

Data request yang dikirim ke aplikasi, dapat diambil menggunakan instance dari class Illuminate\Http\Request. Untuk membuat instance dari class tersebut, kita dapat menggunakan cara manual atau menggunakan helper `request()`. Contoh untuk mengambil `$_GET / $_POST` data dengan key `username`:

²⁰<https://laravelcollective.com/docs/5.3/html>

```
1 $username = request()->get('username');
```

Penjelasan lebih lengkapnya, dapat diakses di [dokumentasi request²¹](#).

2.5 Controller

Pada contoh-contohnya, kita selalu meletakan logic di routes/web.php. Teknik ini tidak efektif jika aplikasinya sudah besar. Cara yang lebih baik adalah router mengarahkan request ke method di controller. Nah, method itulah yang akan melakukan logic untuk request dan memberikan response.

Mari kita praktikan dengan mengubah route about ke method showAbout di MyController.php. Ikuti langkah berikut:

1. Buatlah MyController dengan perintah:

```
php artisan make:controller MyController  
// Controller created successfully.
```

2. Ubah route '/about' menjadi :

routes/web.php

```
Route::get('/about', 'MyController@showAbout');
```

3. Tambah method showAbout pada class MyController dengan isi sebagai berikut:

²¹<https://laravel.com/docs/5.3/requests>

app/Http/Controllers/MyController.php

```
<?php
...
class MyController extends Controller
{
    public function showAbout()
    {
        return view('about');
    }
}
```

4. Akses kembali /about, maka hasilnya akan tetap sama. Yang berubah adalah logic untuk memberikan response sekarang berada di controller, sedangkan router hanya mengarahkan request.

Fitur dari controller ini sangat banyak, diantaranya:

- Validasi request yang datang.
- RESTfull untuk memetakan setiap method pada controller menjadi routes.
- Resource Controller untuk membuat restfull controller pada sebuah model.
- dll.

Penjelasan lebih lengkap, dapat diakses di [dokumentasi controller²²](#).

2.6 Authentikasi

Secara default Laravel telah menyediakan struktur database untuk authentikasi. Kita bahkan telah disediakan generator untuk membuat routing dan viewnya. Cobalah jalankan perintah ini:

```
php artisan migrate
```

Perintah ini akan membuat struktur untuk authentikasi sesuai migration default. Pada database akan muncul table `users` (untuk menyimpan data user) dan table `password_resets` (untuk menyimpan) data reset password.

Selanjutnya jalankan:

²²<https://laravel.com/docs/5.3/controllers>

```
php artisan make:auth  
// Authentication scaffolding generated successfully.
```

Akan muncul beberapa file baru:

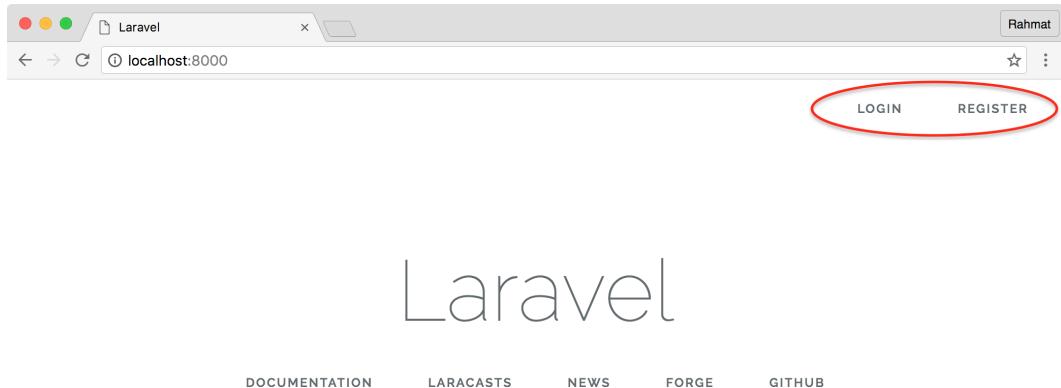
- app/Http/Controllers/HomeController.php
- resources/views/home.blade.php
- resources/views/auth/login.blade.php
- resources/views/auth/register.blade.php
- resources/views/auth/passwords/email.blade.php
- resources/views/auth/passwords/reset.blade.php
- resources/views/layouts/app.blade.php

Pada routes/web.php akan muncul baris berikut:

routes/web.php

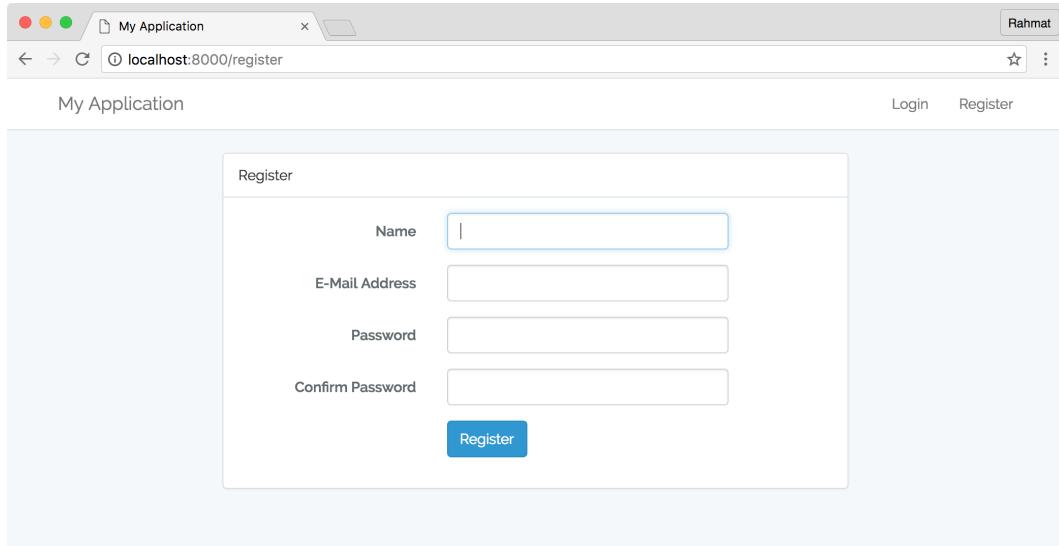
```
Auth::routes();  
Route::get('/home', 'HomeController@index');
```

Kini, ketika kita mengunjungi halaman utama akan muncul tampilan berikut:



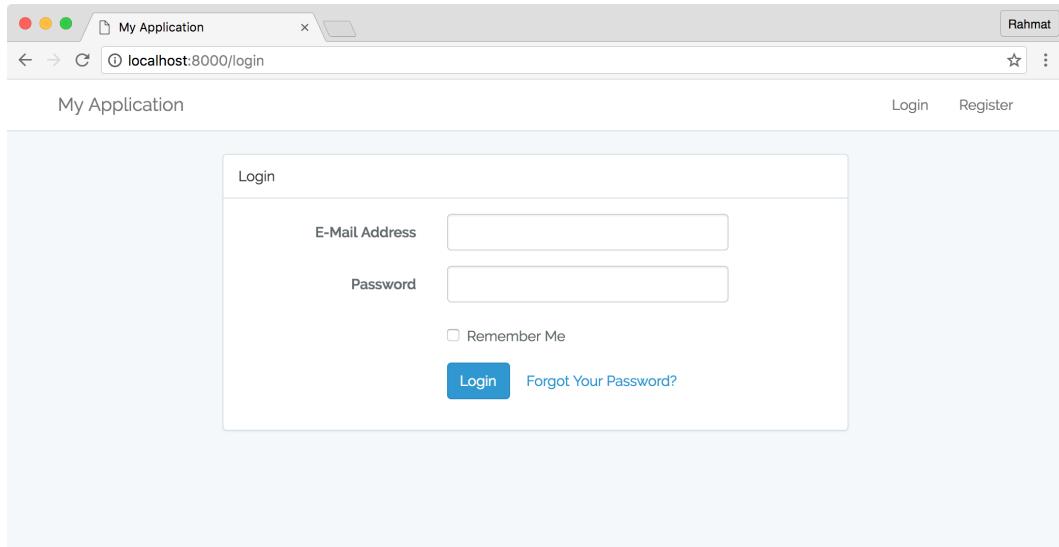
Halaman utama default

Kita juga sudah disediakan fitur register:



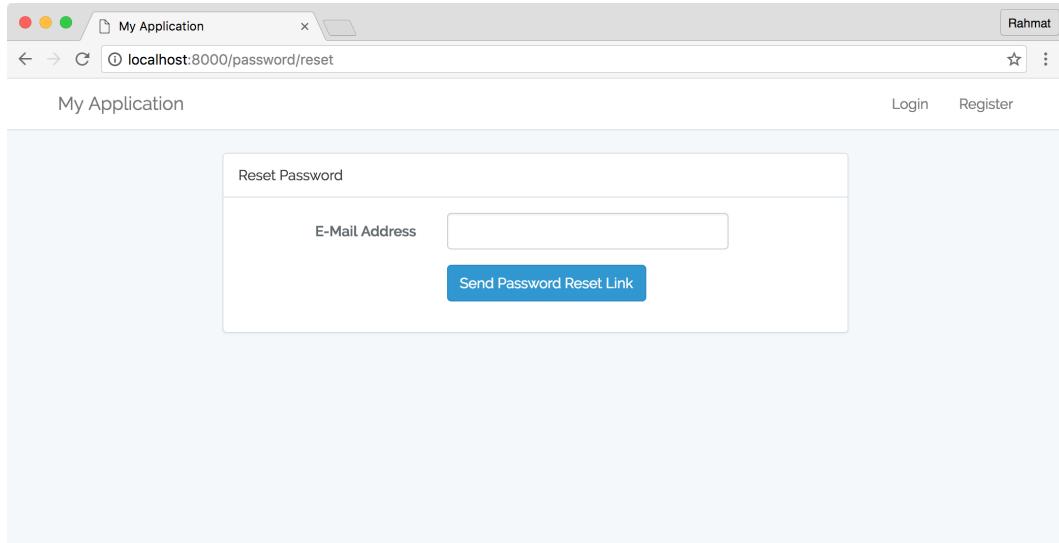
Halaman register

Fitur login:



Halaman login

Dan fitur lupa password:



Halaman lupa password

Kita dapat juga mengecek route baru yang telah dibuat oleh Laravel dengan perintah:

```
php artisan route:list
```



Menerima email ketika development

Ketika menggunakan fitur lupa password Laravel akan mengirim email. Ketika development, kita dapat setting agar email di kirim ke file log di `storage/logs/laravel.log` dengan mengubah isian `MAIL_DRIVER` menjadi `log` pada file `.env`.

Error lain yang sering dialamai adalah pesan error *“Cannot send message without a sender address”*. Error ini bisa diselesaikan dengan mengisi array `from` dengan nama dan alamat email default yang akan digunakan untuk mengirim email pada `config/mail.php`.

2.7 Ringkasan

Pada hari 2 ini, kita telah belajar MVC dan model sebuah aplikasi dalam framework Laravel, poin-poin yang telah kita bahas yaitu:

- Konsep routing
- Konsep MVC
- Migrasi
- Database Seeder
- Authentikasi

Pada hari 3, kita akan mulai membuat perencanaan untuk project yang akan dibangun dengan framework Laravel. Semangat! :)

3. Hari 3 : Persiapan Project

Kita akan membangun aplikasi perpustakaan dengan framework Laravel dengan nama Larapus (singkatan dari Laravel Perpustakaan). Sengaja saya memilih ide aplikasi perpustakaan agar alur bisnisnya lebih mudah dipahami. Fitur-fitur yang ada di aplikasi ini diantaranya:

- Administrasi buku untuk Admin
- Administrasi user untuk Admin
- Administrasi untuk Registered User
- Peminjaman buku untuk Registered User
- Browsing buku untuk non-member
- Dashboard dengan chart untuk Admin (bonus)
- Import dan Export Excel (bonus)
- Export PDF (bonus)

3.1 Design tampilan

Tampilan akhir dari Larapus akan seperti berikut:

3.1.1 Tampilan Halaman Depan untuk Guest

The screenshot shows a web application interface titled "Larapus". At the top, there are navigation links for "Dashboard", "Login", and "Daftar". On the right, there is a user profile placeholder "Rahmat". The main content area is titled "Daftar Buku" and displays a table of books. The columns are "Judul", "Stok", and "Penulis". Each row contains a book title, its stock level, and the author's name, followed by a blue "Pinjam" button. Below the table, it says "Showing 1 to 4 of 4 entries". At the bottom right, there are navigation buttons for "PREVIOUS", "1", and "NEXT".

Judul	Stok	Penulis	
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	Pinjam
Jalan Cinta Para Pejuang	1	Salim A. Fillah	Pinjam
Kupinang Engkau dengan Hamdalah	2	Mohammad Fauzil Adhim	Pinjam
Membingkai Surga dalam Rumah Tangga	4	Aam Amiruddin	Pinjam

Halaman Depan untuk Guest

3.1.2 Halaman Admin

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/books'. The page is titled 'Buku' and displays a list of books. At the top, there are 'Tambah' and 'Export' buttons. Below that, there are filters for 'Show' (set to 10) and a 'Search' input field. The table has columns: Judul, Jumlah, Stok, Penulis, and actions (Ubah, HAPUS). The data is as follows:

Judul	Jumlah	Stok	Penulis	Aksi
Cinta & Seks Rumah Tangga Muslim	3	3	Aam Amiruddin	Ubah HAPUS
Jalan Cinta Para Pejuang	2	1	Salim A. Fillah	Ubah HAPUS
Kupinang Engkau dengan Hamdalih	3	2	Mohammad Fauzil Adhim	Ubah HAPUS
Membingkai Surga dalam Rumah Tangga	4	4	Aam Amiruddin	Ubah HAPUS

At the bottom, it says 'Showing 1 to 4 of 4 entries' and has navigation buttons for 'PREVIOUS', 'NEXT', and a page number '1'.

Halaman Data Buku

Lapusus Dashboard > Penulis

Penulis

TAMBAH

Show 10 entries Search:

Nama	
Aam Amiruddin	Ubah HAPUS
Mohammad Fauzil Adhim	Ubah HAPUS
Salim A. Fillah	Ubah HAPUS

Showing 1 to 3 of 3 entries

PREVIOUS **1** NEXT

Halaman Data Penulis

Lapusus Dashboard > Data Peminjaman

Data Peminjaman

Show 10 entries Status Semua Search:

Judul	Peminjam	Tanggal Pinjam	Tanggal Kembali
Jalan Cinta Para Pejuang	Sample Member	2016-03-25 04:24:54	KEMBALIKAN
Kupinang Engkau dengan Hamdalah	Sample Member	2016-03-25 04:24:54	KEMBALIKAN

Showing 1 to 2 of 2 entries

PREVIOUS **1** NEXT

Halaman Data Peminjaman Buku

The screenshot shows a web browser window titled "Larapus" with the URL "localhost:8000/admin/members". The top navigation bar includes links for "Larapus", "Dashboard", "Buku", "Penulis", "Member", "Peminjaman", "Profil", and a dropdown for "Admin Larapus". Below the navigation is a breadcrumb trail: "Dashboard > Member". A search bar and a "Show 10 entries" button are present. The main content area displays a table with one entry:

Nama	Email	Login terakhir	
Sample Member	member@gmail.com	2016-03-25 11:24:54	Lihat data peminjaman HAPUS

At the bottom, it says "Showing 1 to 1 of 1 entries" and has "PREVIOUS", "NEXT", and a page number "1".

Halaman Data User

3.1.3 Halaman User

The screenshot shows a web browser window titled "Larapus" with the URL "localhost:8000/register". The top navigation bar includes links for "Larapus", "Dashboard", "Login", and "Daftar". The main content area is titled "Daftar" and contains four input fields: "Nama", "Alamat Email", "Password", and "Konfirmasi Password". Below these is a reCAPTCHA field with the text "I'm not a robot" and a checkbox. At the bottom is a blue "DAFTAR" button.

Halaman Registrasi User

The screenshot shows a web browser window titled "Larapus" with the URL "localhost:8000/login". The page has a header with "Larapus" and "Dashboard" on the left, and "Login" and "Daftar" on the right. A "Rahmat" button is in the top right corner. The main content is a "Login" form with fields for "Alamat Email" and "Password", a "Ingat saya" checkbox, and buttons for "LOGIN" and "LUPA PASSWORD".

Halaman Login

The screenshot shows a web browser window titled "Larapus" with the URL "localhost:8000/password/reset". The page has a header with "Larapus" and "Dashboard" on the left, and "Login" and "Daftar" on the right. A "Rahmat" button is in the top right corner. The main content is a "Reset Password" form with a field for "Alamat Email" and a large blue button labeled "KIRIM LINK RESET PASSWORD" with an envelope icon.

Halaman Lupa Password

Larapus

localhost:8000/home

Rahmat

Larapus Dashboard Profil Sample Member

Dashboard

Selamat datang di Larapus.

Login Terakhir 2016-03-25 04:31:54

Buku dipinjam • Jalan Cinta Para Pejuang KEMBALIKAN

Halaman Dashboard dan data peminjaman buku

Larapus

localhost:8000/settings/profile

Rahmat

Larapus Dashboard Profil Sample Member

Dashboard > Profil

Profil

Nama Sample Member

Email member@gmail.com

Login terakhir 2016-03-25 04:31:54

UBAH

Halaman Profil User

Larapus

localhost:8000/settings/password

Rahmat

Sample Member

Dashboard Profil

Dashboard > Ubah Password

Ubah Password

Password lama

Password baru

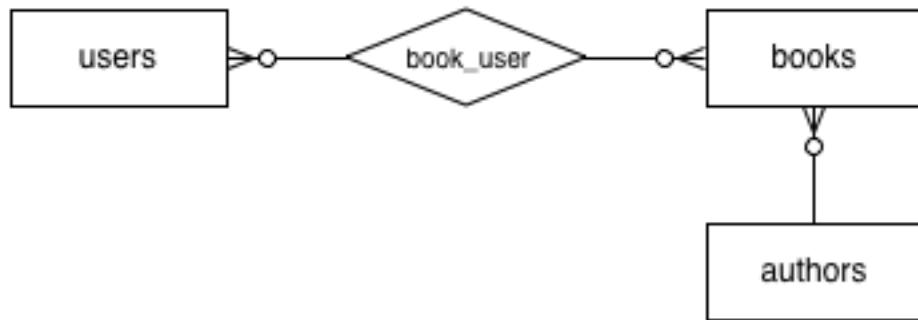
Konfirmasi password baru

SIMPAN

Halaman Ubah Password

3.2 Database

Berikut struktur database yang akan kita gunakan untuk fungsi dasar aplikasi:

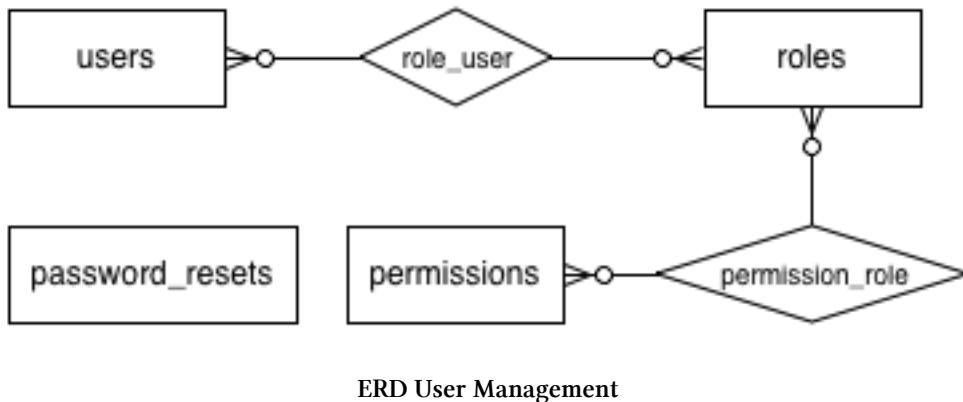


ERD core aplikasi

Sementara untuk manajemen user kita akan menggunakan struktur table default Laravel dan package [santigarcia/laratrust](https://github.com/santigarcia/laratrust)¹. Package ini akan kita gunakan untuk fitur

¹<https://github.com/santigarcia/laratrust>

Role (admin dan member biasa) di Larapus. Struktur akhirnya akan seperti ini:



Penjelasan untuk tiap entity:

- authors: menyimpan data penulis buku
- books: menyimpan data buku
- users: menyimpan data users
- password_resets: menyimpan data “forgot password” tiap user
- roles: menyimpan data Role (User atau Admin)
- role_user: menyimpan data role yang dimiliki oleh user
- permissions: menyimpan data permission
- permission_role: menyimpan data permission yang dimiliki oleh tiap role

Struktur database ini akan dibuat menggunakan migrations. Jadi, Anda tidak perlu membuat struktur database ini sekarang.

3.3 Code sample

Untuk memudahkan, semua syntax yang saya tulis selama pengembangan aplikasi dapat diakses di [bitbucket](https://bitbucket.org/rahmatawalidin/larapus-53)². Silahkan download source code ini untuk memudahkan dalam mengecek syntax selama mengikuti tahapan di buku ini.

²<https://bitbucket.org/rahmatawalidin/larapus-53>

Yang saya sarankan adalah mengikuti setiap langkah di pembahasan buku ini satu-persatu. Tapi, jika ingin melihat hasil akhir dari Larapus, dapat langsung menjalankan code sample dengan cara:

1. Download semua source code dari bitbucket
2. Ekstrak ke folder
3. Jalankan `composer install` untuk menginstall dependency Laravel (untuk melihat proses instalasi lebih detail, jalankan `composer install -vvv`)
4. Salin file `.env.example` ke `.env`.
5. Generate app key untuk aplikasi dengan perintah `php artisan key:generate`.
6. Daftar ke <https://www.google.com/recaptcha>. Pada file `.env`, isi `NOCAPTCHA_SECRET` dan `NOCAPTCHA_SITEKEY` dengan key yang didapatkan dari <https://www.google.com/recaptcha/admin>. Jika bingung, baca penjelasan tentang [Captcha](#) di hari 6.
7. Daftar ke <http://www.mailgun.com>⁴. Pada file `.env`, isi `MAILGUN_DOMAIN` dengan data dari <https://mailgun.com/app/domains>⁵ dan isi `MAILGUN_SECRET` dengan data *private api key* dari <https://mailgun.com/app/account/settings>⁶. Jika bingung, baca penjelasan tentang [Mailgun](#) di hari 6.
8. Sesuaikan konfigurasi database.
9. Migrasi dan seed database dengan `php artisan migrate:refresh --seed`. Jika mendapat error, pastikan sudah mengecek [solusi error database](#) di hari pertama dan [solusi error cache driver](#) di bawah.
10. Jalankan simple web server dengan `php artisan serve`.
11. Akses aplikasi di `http://localhost:8000` (username dan password ada di file `database/seeds/UsersSeeder.php`). Jika mendapat error “cURL error 60: SSL certificate problem”, kemungkinan ada kesalahan saat konfigurasi Mailgun. Silahkan ikuti petunjuk di [solusi error business verification](#) dan [solusi error Guzzle](#) di hari 6.



Email tidak diterima/error?

Pada sample aplikasi ini, kita menggunakan mailgun sebagai gateway untuk mengirim email. Karena kita tidak melakukan verifikasi domain, biasanya email akan masuk folder *spam*.

³<https://www.google.com/recaptcha/admin>

⁴<http://www.mailgun.com>

⁵<https://mailgun.com/app/domains>

⁶<https://mailgun.com/app/account/settings>

3.4 Instalasi Laravel

Untuk memulai project ini, kita install Laravel dengan perintah berikut:

```
composer create-project laravel/laravel=5.3.6 larapus
```

Disini sengaja kita menginstall versi 5.3.6 agar sama dengan yang digunakan di buku ini.

Setelah proses download selesai, konfigurasi database sesuai dengan penjelasan di hari sebelumnya.

Pada saat development, kita akan set agar email dikirim ke file storage/logs/laravel.log. Caranya, ubah isian MAIL_DRIVER menjadi log pada file .env. Untuk menghindari error, isi juga array from di config/mail.php misalnya seperti ini:

config/mail.php

```
....  
'from' => [  
    'address' => 'admin@larapus.com',  
    'name' => 'Admin Larapus',  
,  
....
```

Untuk memudahkan membuat form, pada Larapus kita juga akan menggunakan [laravelcollective/html](#)⁷. Silahkan konfigurasi dulu package ini sesuai penjelasan di hari sebelumnya.

Di Laravel, kita juga dapat set nama aplikasi yang akan kita buat. Nama ini akan digunakan jika Laravel membutuhkan ketika menggenerate fitur bawaan. Caranya, kita set isian name di config/app.php:

⁷<https://laravelcollective.com/docs/5.3/html>

config/app.php

```
'name' => 'Larapus',
```

Terakhir, kita akan menggunakan simple server. Jalankan perintah berikut:

```
php artisan serve
```

Pastikan dapat mengakses Larapus dari `http://localhost:8000`.

3.5 Konfigurasi Authentikasi

Mari kita tambahkan fitur authentikasi sebagaimana kita lakukan pada hari sebelumnya.

```
php artisan make:auth
```

Kemudian kita buat struktur database untuk authentikasi:

```
php artisan migrate
```

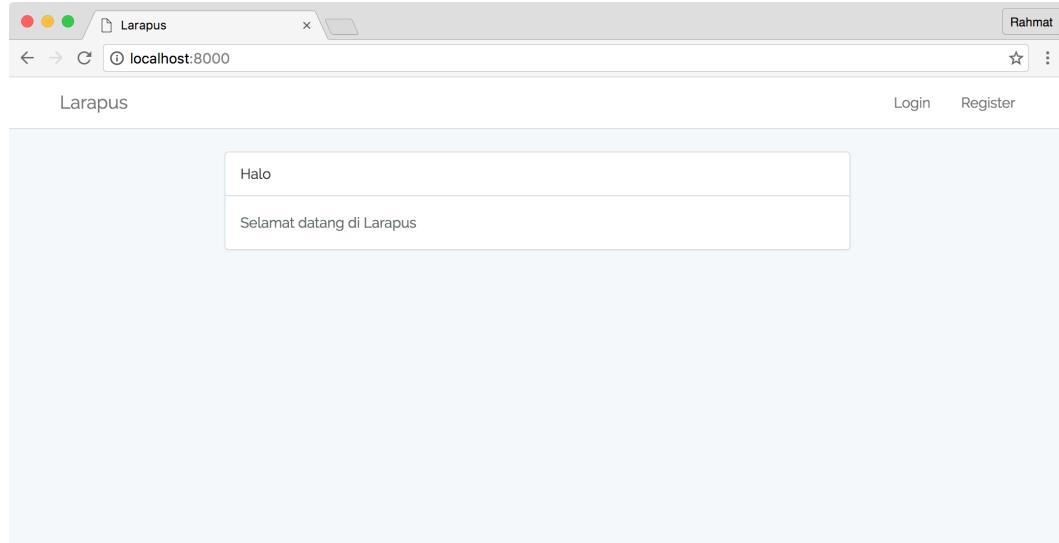
Kita akan melakukan beberapa perubahan pada tampilan default dari Laravel agar sesuai dengan Larapus. Pertama, kita ubah tampilan halaman `welcome`. Ubahlah isi file ini menjadi:

resources/views/welcome.blade.php

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-8 col-md-offset-2">
            <div class="panel panel-default">
                <div class="panel-heading">Halo</div>
                <div class="panel-body">
                    Selamat datang di Larapus
                </div>
            </div>
        </div>
    </div>
@endsection
```

Sehingga tampilan halaman `welcome` akan menjadi:

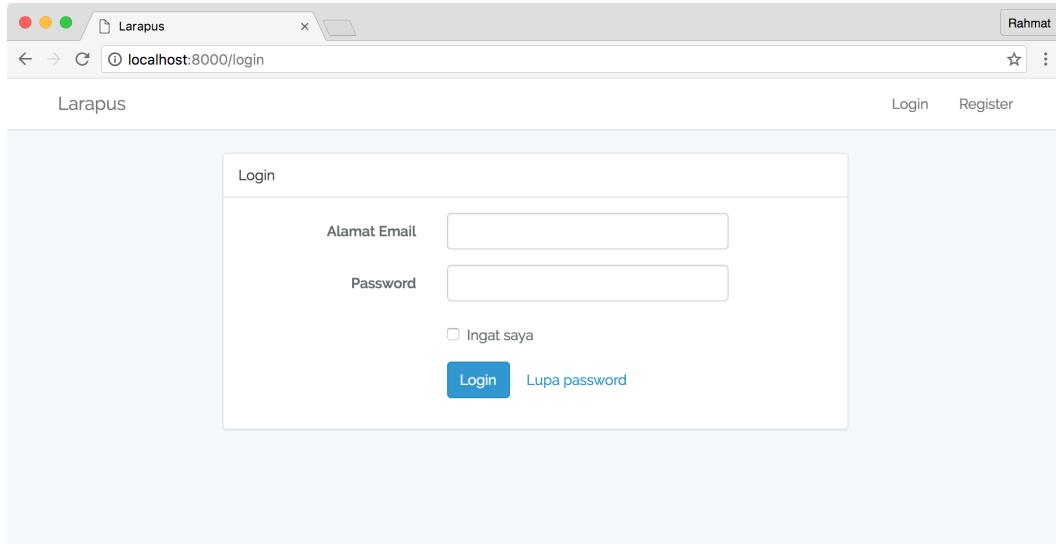
**Halaman welcome**

Kita ubah juga halaman `login` agar menggunakan `laravelcollective/html` untuk formnya:

resources/views/auth/login.blade.php

```
....  
<form class="form-horizontal" role="form" method="POST" action="{{ url('/login') }}>  
....  
</form>  
{!! Form::open(['url'=>'login', 'class'=>'form-horizontal']) !!}  
  
<div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}>  
    {!! Form::label('email', 'Alamat Email', ['class'=>'col-md-4 control-label']) !!}  
    <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('password') ? ' has-error' : '' }}>  
    {!! Form::label('password', 'Password', ['class'=>'col-md-4 control-label']) !!}  
    <div class="col-md-6">  
        {!! Form::password('password', ['class'=>'form-control']) !!}  
        {!! $errors->first('password', '<p class="help-block">:message</p>') !!}  
    </div>  
</div>  
  
<div class="form-group">  
    <div class="col-md-6 col-md-offset-4">  
        <div class="checkbox">  
            <label>  
                {!! Form::checkbox('remember')!!} Ingat saya  
            </label>  
        </div>  
    </div>  
</div>  
  
<div class="form-group">  
    <div class="col-md-6 col-md-offset-4">  
        <button type="submit" class="btn btn-primary">  
            <i class="fa fa-btn fa-sign-in"></i> Login  
        </button>  
  
        <a class="btn btn-link" href="{{ url('/password/reset') }}>Lupa password</a>  
    </div>  
</div>  
{!! Form::close() !!}  
....
```

Yang kita lakukan disini adalah merubah cara membuat setiap elemen form dan cara menampilkan pesan validasi. Untuk validasi, akan kita bahas lebih lanjut di pembahasan selanjutnya. Hasil akhir halaman login akan seperti berikut:



Halaman login

Kita ubah juga halaman register:

resources/views/auth/register.blade.php

```
....  
<form class="form-horizontal" role="form" method="POST" action="{{ url('/register') }}">  
....  
</form>  
{!! Form::open(['url'=>'register', 'class'=>'form-horizontal']) !!}  
  
<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('email') ? ' has-error' : '' }}>  
    {!! Form::label('email', 'Alamat Email', ['class'=>'col-md-4 control-label']) !!}
```

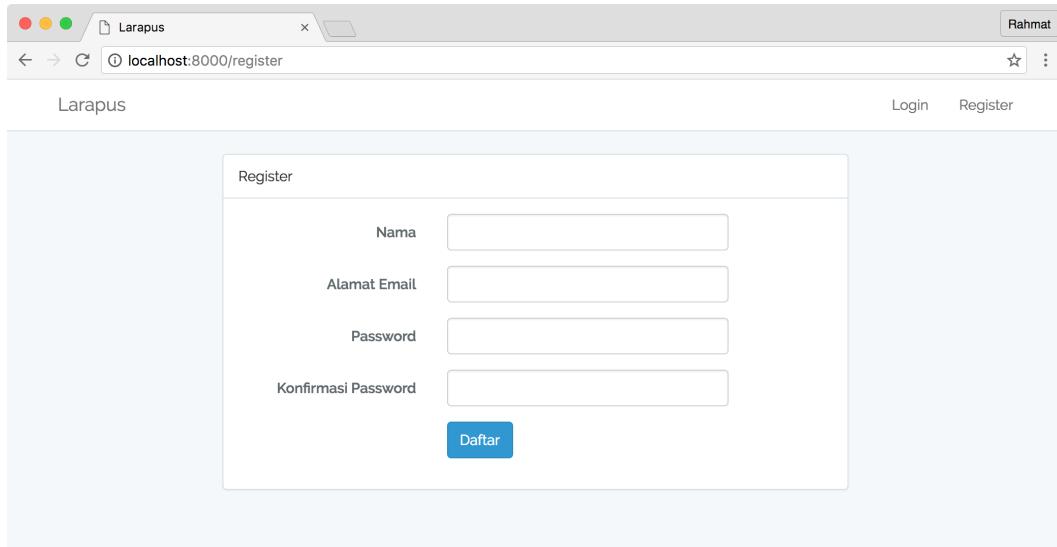
```
<div class="col-md-6">
  {!! Form::email('email', null, ['class'=>'form-control']) !!}
  {!! $errors->first('email', '<p class="help-block">:message</p>') !!}
</div>
</div>

<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">
  {!! Form::label('password', 'Password', ['class'=>'col-md-4 control-label']) !!}
  <div class="col-md-6">
    {!! Form::password('password', ['class'=>'form-control']) !!}
    {!! $errors->first('password', '<p class="help-block">:message</p>') !!}
  </div>
</div>

<div class="form-group{{ $errors->has('password_confirmation') ? ' has-error' : '' }}">
  {!! Form::label('password_confirmation', 'Konfirmasi Password', ['class'=>'col-md-4 control-label']) !!}
  <div class="col-md-6">
    {!! Form::password('password_confirmation', ['class'=>'form-control']) !!}
    {!! $errors->first('password_confirmation', '<p class="help-block">:message</p>') !!}
  </div>
</div>

<div class="form-group">
  <div class="col-md-6 col-md-offset-4">
    <button type="submit" class="btn btn-primary">
      <i class="fa fa-btn fa-user"></i> Daftar
    </button>
  </div>
</div>
{!! Form::close() !!}
....
```

Halaman register akan menjadi seperti ini:



Halaman register

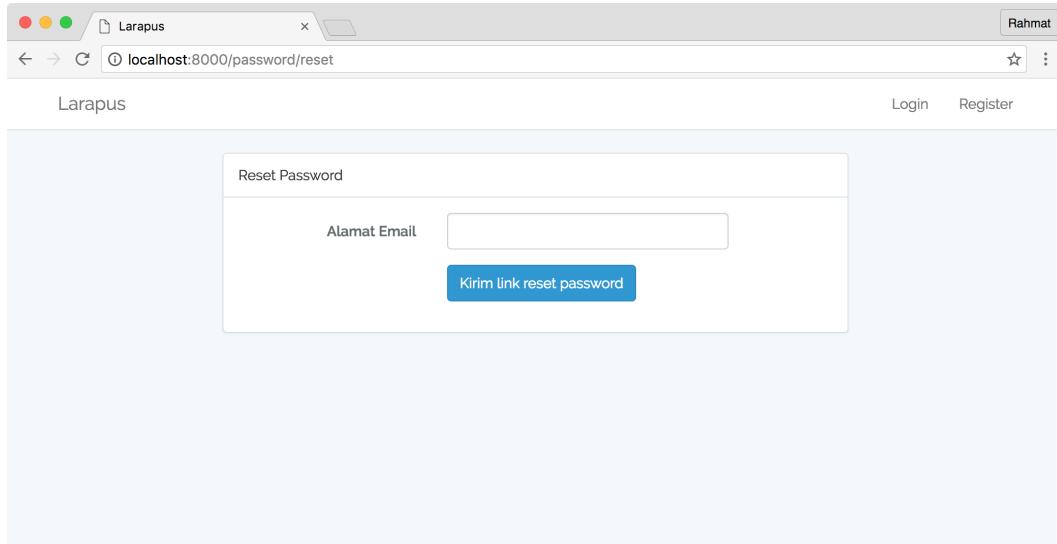
Halaman lupa password juga kita ubah:

resources/views/auth/passwords/email.blade.php

```
....  
<form class="form-horizontal" role="form" method="POST" action="{{ url('/password/email') }}>  
....  
</form>  
{!! Form::open(['url'=>'password/email', 'class'=>'form-horizontal']) !!}  
  
<div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}>  
    {!! Form::label('email', 'Alamat Email', ['class'=>'col-md-4 control-label']) !!}  
    <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">  
    <div class="col-md-6 col-md-offset-4">  
        <button type="submit" class="btn btn-primary">  
            <i class="fa fa-btn fa-envelope"></i> Kirim link reset password  
        </button>  
    </div>  
</div>
```

```
{!! Form::close() !!}
....
```

Sehingga tampilannya akan seperti berikut:



Halaman reset password

Kita ubah juga halaman ketika user klik link untuk reset password:

resources/views/auth/passwords/reset.blade.php

```
....
<form class="form-horizontal" role="form" method="POST" action="{{ url('/password/reset') }}">
.....
</form>
{!! Form::open(['url'=>'password/reset', 'class'=>'form-horizontal']) !!}
.....
<input type="hidden" name="token" value="{{ $token }}>

<div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}>
  {!! Form::label('email', 'Alamat Email', ['class'=>'col-md-4 control-label']) !!}
  <div class="col-md-6">
    {!! Form::email('email', isset($email) ? $email : null, ['class'=>'form-control']) !!}
    {{ $errors->first('email', '<p class="help-block">:message</p>') }}
  </div>
</div>

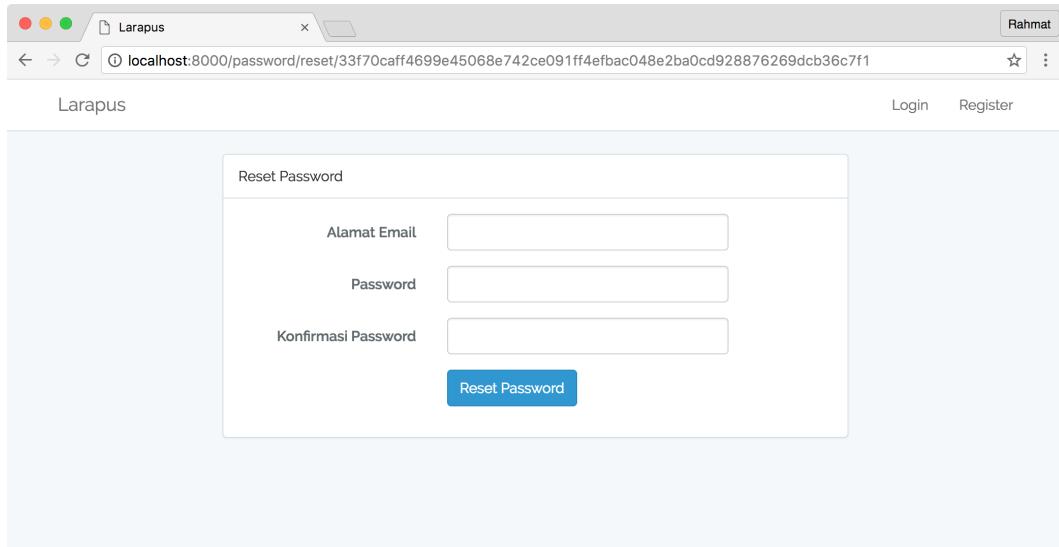
```

```
<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}>
    {!! Form::label('password', 'Password', ['class'=>'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::password('password', ['class'=>'form-control']) !!}
        {!! $errors->first('password', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group{{ $errors->has('password_confirmation') ? ' has-error' : '' }}>
    {!! Form::label('password_confirmation', 'Konfirmasi Password', ['class'=>'col-md-4 contr\ol-label']) !!}
    <div class="col-md-6">
        {!! Form::password('password_confirmation', ['class'=>'form-control']) !!}
        {!! $errors->first('password_confirmation', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        <button type="submit" class="btn btn-primary">
            <i class="fa fa-btn fa-refresh"></i> Reset Password
        </button>
    </div>
</div>
{!! Form::close() !!}
....
```

Untuk mencoba halaman ini, kita harus membuat user baru dari halaman register dan mencoba fitur “Lupa Password”. Pada email yang muncul di file storage/logs/laravel.log, klik link yang muncul. Pastikan halaman yang muncul seperti ini:



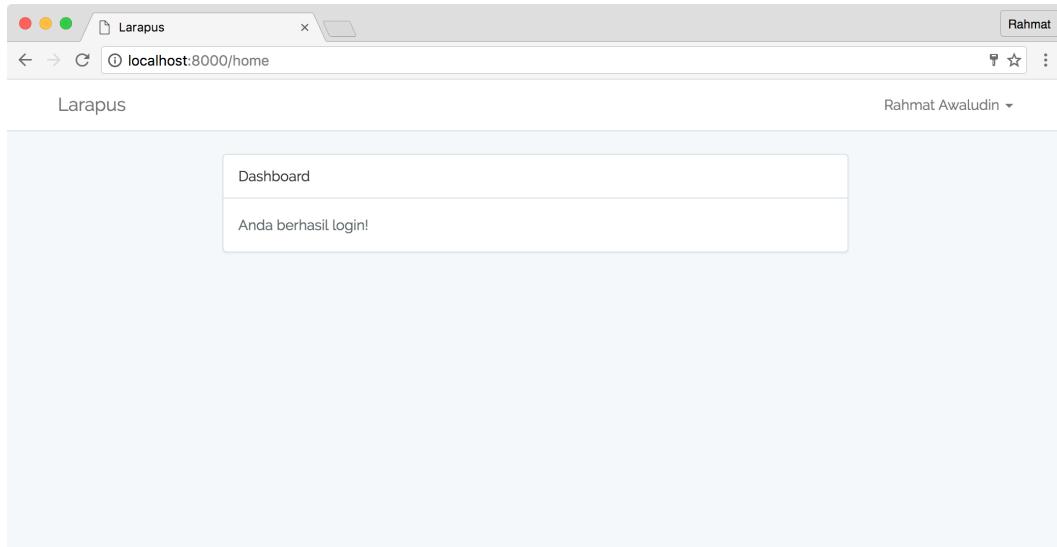
Halaman reset password dari email

Setelah berhasil login, kita ubah juga tampilan untuk halaman dashboard:

resources/views/home.blade.php

```
....  
<div class="panel-body">  
  You are logged in!  
  Anda berhasil login!  
</div>  
....
```

Sehingga tampilan dashboard akan menjadi:



Halaman dashboard

3.6 Memahami Layouting

Dalam mengembangkan website dengan Laravel kita akan menggunakan fitur layouting pada Blade. Ada beberapa istilah yang akan kita pelajari yaitu *parent*, *child* dan *partial view*. Mari kita pelajari satu-persatu.

3.6.1 Parent View

Sesuai namanya view ini yang akan menjadi view induk untuk view lain. Di Laravel, salah satu ciri sebuah view adalah parent view yaitu adanya syntax `@yield`. Contoh parent view adalah file `resources/views/layouts/app.blade.php`. Pada view ini, kita akan menemui baris berikut di dalam tag `<body>`:

resources/views/layouts/app.blade.php

```
....  
<body>  
....  
@yield('content')  
....  
</body>  
....
```

Baris ini akan berubah sesuai data pada child view yang menggunakannya. Jumlah baris @yield() ini dapat lebih dari satu, asalkan namanya berbeda. Pada contoh diatas, kita hanya membuat 1 yield dengan nama content.

3.6.2 Child View

Setelah kita membuat parent view, kita dapat menggunakan view tersebut pada child view. Syntaxnya seperti ini:

```
@extends('alamat-parent')  
  
@section('nama-yield')  
 // Syntax yang mau diisi pada yield `nama-yield`.  
@endsection
```

Jumlah parent dapat di extends hanya satu. Sementara, jumlah yield yang dapat diisi tergantung jumlah yield pada parent nya.

Contoh penggunaan child view dapat kita lihat di resources/views/welcome.blade.php:

```
resources/views/welcome.blade.php
```

```
@extends('layouts.app')  
  
@section('content')  
....  
@endsection
```

Isian untuk parent view selalu dibatasi dengan titik. Pada contoh diatas, `layouts.app` artinya parent view akan berada di `resources/views/layouts/app.blade.php`. Disini, kita juga mengisi isian untuk `yield` dengan nama `content`.

3.6.3 Partial View

Berbeda dengan parent, kita dapat memiliki lebih dari satu partial view. Teknik ini biasanya digunakan untuk memisahkan element yang sering digunakan pada beberapa view, misalnya untuk form. Dalam development, biasanya kita akan menggunakan form yang sama untuk halaman `create` dan `edit`. Disini, kita dapat menjadikan form tersebut menjadi partial view dan memanggilnya dari view `create` dan `edit`.

Untuk menggunakan partial view, kita buat view biasa tanpa parent. Pada view yang akan menggunakan partial view, kita buat syntax:

```
@include('nama-partial-view')
```

Penggunaan partial view akan kita praktekan pada pembahasan selanjutnya.

3.6.4 Contoh penggunaan layouting

Mari kita praktekan penggunaan layouting ini untuk menampilkan link ke dashboard dan merubah tulis “Register” menjadi “Daftar” pada navigasi.

Perubahan ini harus kita lakukan di parent view, yaitu `resources/views/layouts/app.blade.php`. Pertama, kita tambah link ke “Dashboard” atau ke url `/home`. Tambahkan baris berikut:

resources/views/layouts/app.blade.php

```
....  
<!-- Left Side Of Navbar -->  
<ul class="nav navbar-nav">  
    @if (Auth::check())  
        <li><a href="{{ url('/home') }}>Dashboard</a></li>  
    @endif  
</ul>  
....
```

Link ke Dashboard hanya akan kita tampilkan jika user sudah login. Disini kita menggunakan `@if` pada hasil dari `Auth::check()`. Method ini akan menghasilkan nilai true jika user sudah login.

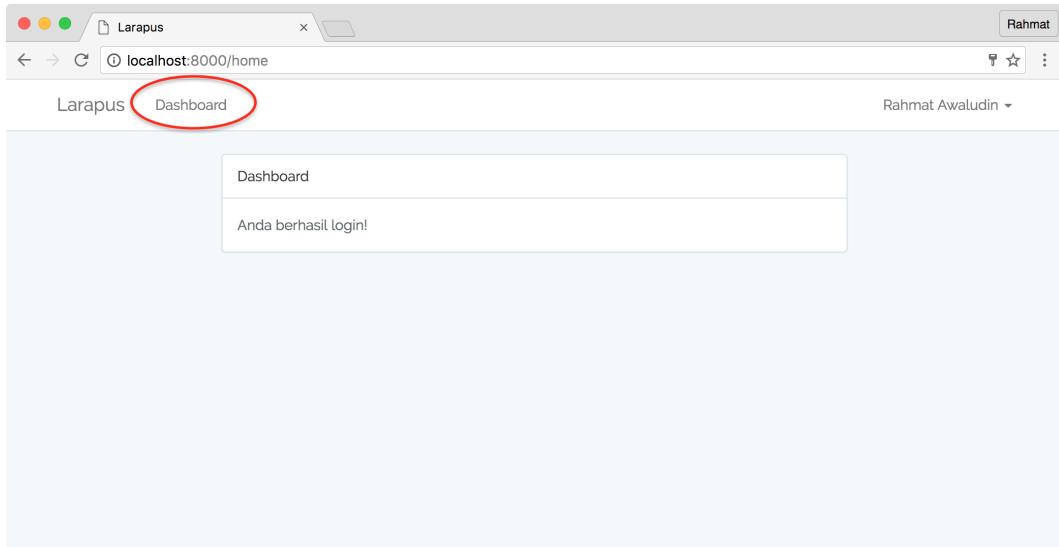
Selanjutnya, kita ubah “Register” menjadi “Daftar”:

resources/views/layouts/app.blade.php

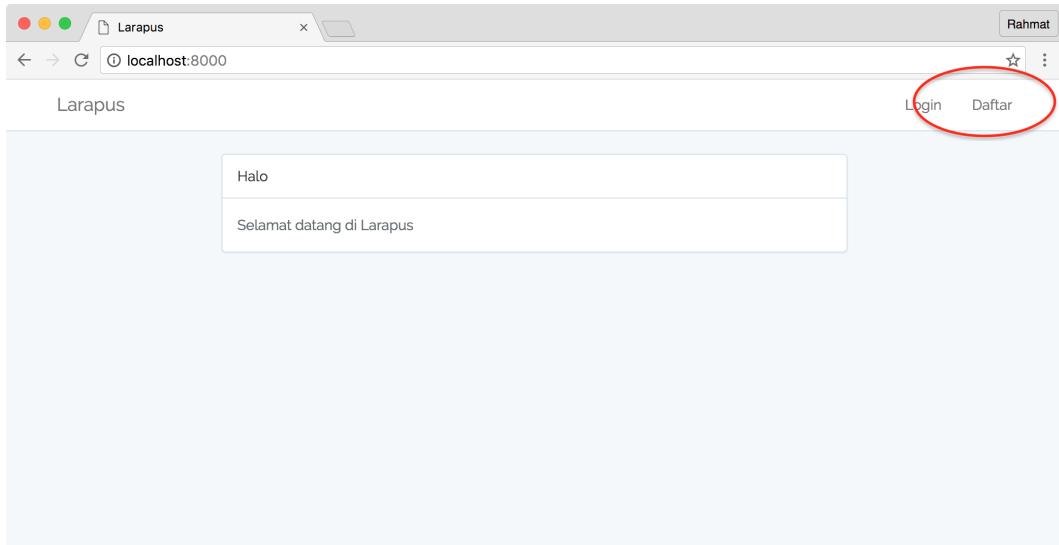
```
....  
<!-- Authentication Links -->  
@if (Auth::guest())  
    <li><a href="{{ url('/login') }}>Login</a></li>  
    <li><a href="{{ url('/register') }}>Register</a></li>  
    <li><a href="{{ url('/register') }}>Daftar</a></li>  
@else  
....
```

Disini kita hanya merubah syntax yang sudah ada. Kebalikan dari method sebelumnya, link “Register” hanya muncul ketika user belum login. Untuk itu, Laravel melakukan pengecekan pada hasil dari `Auth::guest()`. Method ini akan menghasilkan true jika user belum login.

Tampilan akhir header dari tiap halaman akan seperti berikut:



Menambah link Dashboard



Mengubah Register menjadi Daftar

3.7 Konfigurasi Asset

Setiap asset misalnya css dan javascript di Laravel harus disimpan di folder public agar dapat diakses. Untuk menampilkan link ke asset tersebut dari view, kita dapat menggunakan helper `asset()`. Misalnya, ada asset css di `public/css/app.css`, maka kita tulis syntax berikut pada view untuk mendapatkan css ini:

```
<link href="{{ asset('css/app.css') }}" rel='stylesheet' type='text/css'>
```

Kita juga tetap dapat menggunakan cara biasa seperti ini:

```
<link href="/css/app.css" rel="stylesheet">
```

Secara default, Laravel telah memiliki *precompiled* css berisi bootstrap dan style laravel di `public/css/app.css`.

Pada pembahasan ini, kita akan merubah css dengan bootstrap dan menggunakan custom theme dari bootswatch.



Pada tahap ini kita akan mendownload file. Anda juga dapat langsung menyalin file css, js maupun fonts dari sample source code.

Kita mulai dari bootstrap, download bootstrap dari getbootstrap.com⁸. Pilih yang “Compiled and minified” (disini menggunakan versi 3.3.7).

⁸<http://getbootstrap.com/getting-started/#download>

The screenshot shows a web browser window titled "Getting started - Bootstrap". The URL in the address bar is <https://getbootstrap.com/getting-started/#download>. The page content is titled "Download". It contains three main sections: "Bootstrap", "Source code", and "Sass". The "Bootstrap" section has a note about compiled CSS, JS, and fonts, and includes a "Download Bootstrap" button which is circled in red. The "Source code" section includes a note about Less and Sass, and a "Download source" button. The "Sass" section includes a note about porting from Less to Sass, and a "Download Sass" button. On the right side, there is a sidebar with a "Download" heading and a list of links including "What's included", "Compiling CSS and JavaScript", "Basic template", "Examples", "Tools", "Community", "Disabling responsiveness", "Migrating from 2.x to 3.0", "Browser and device support", "Third party support", "Accessibility", "License FAQs", and "Translations". At the bottom of the sidebar is a "Back to top" link.

Bootstrap CDN

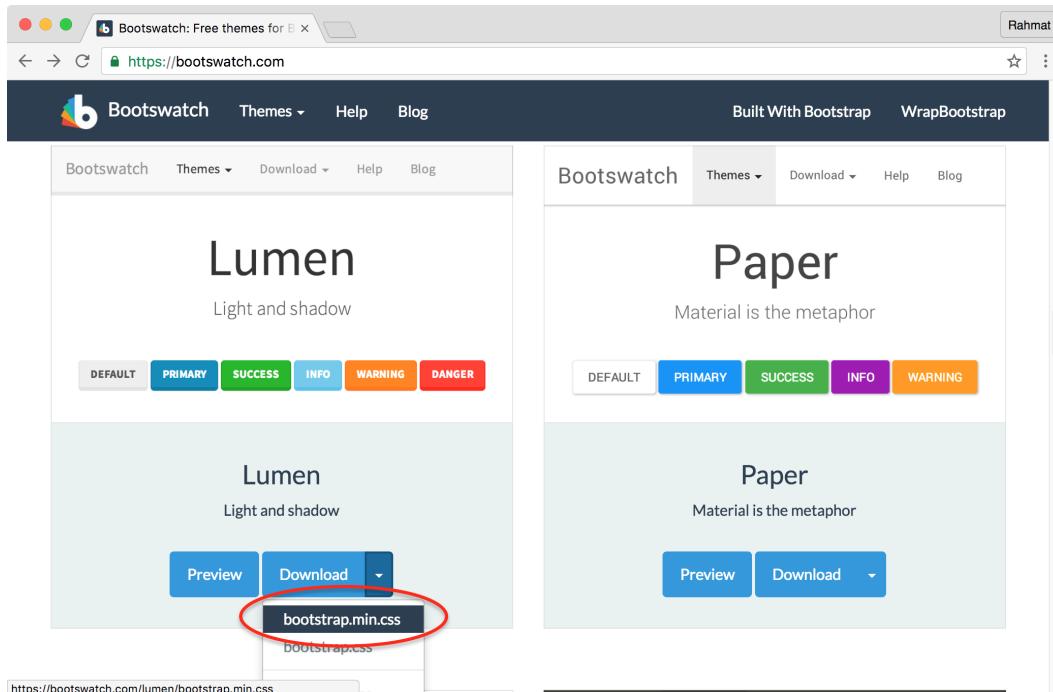
[Download bootstrap](#)

Ekstrak zip yang didapatkan. Buatlah folder fonts pada folder public di Larapus. Selanjutnya, salin file berikut dari zip hasil extract bootstrap:

- Isi folder fonts ke public/fonts.
- File js/bootstrap.min.js ke public/js

Agar tampilan bootstrapnya tidak terlalu monoton, kita akan menggunakan css dari [bootswatch⁹](http://bootswatch.com/). Saya akan memilih theme Lumen. Klik pada tombol panah kecil di samping tombol “Download” dan pilih bootstrap.min.css.

⁹<http://bootswatch.com/>



Download Theme Lumen

Simpan file `bootstrap.min.css` yang didownload ke folder `public/css`.

Kita akan menggunakan file css yang khusus untuk Larapus. Ubahlah isi file `public/css/app.css` dengan syntax berikut:

`public/css/app.css`

```
.tab-pane {
    padding: 15px 0px 15px 0px;
}
```

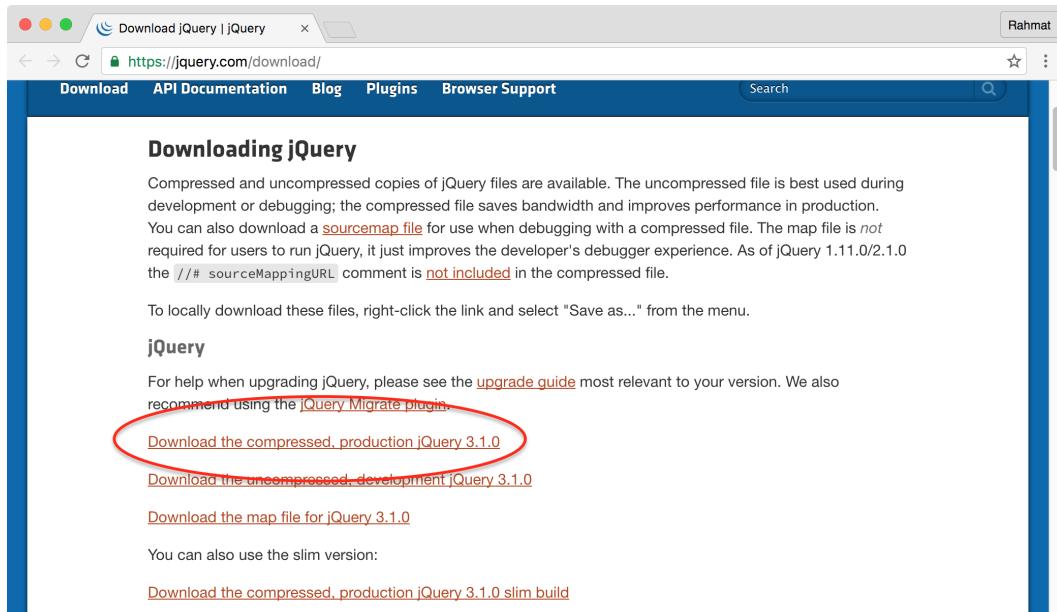
Baris css diatas kita gunakan untuk membuat tampilan lebih rapi.

Kita juga akan membutuhkan FontAwesome untuk icon, download dari [halaman resmi¹⁰](#) (disini menggunakan versi 4.6.3) dan ekstrak zip yang didapatkan. Kemudian, salin file berikut:

¹⁰<https://fontawesome.github.io/Font-Awesome>

- Isi folder fonts ke public/fonts.
- File css/font-awesome.min.css ke public/css

jQuery dibutuhkan untuk bootstrap, download dari [halaman resminya¹¹](#) (disini menggunakan versi 3.1.0).



The screenshot shows a web browser window with the address bar containing <https://jquery.com/download/>. The page header includes links for Download, API Documentation, Blog, Plugins, and Browser Support, along with a search bar and user profile information. The main content area is titled "Downloading jQuery". It discusses the availability of compressed and uncompressed files, noting that the compressed file is best for development or debugging due to bandwidth savings and improved performance. It also mentions the use of a sourcemap file for debugging compressed files. Below this, instructions advise right-clicking on the links to save them. A section titled "jQuery" provides upgrade guidance and links to the [upgrade guide](#) and the [jQuery Migrate plugin](#). Two download links are present: "Download the compressed, production jQuery 3.1.0" (which is circled in red) and "Download the uncompressed, development jQuery 3.1.0". Further down, links for "Download the map file for jQuery 3.1.0" and "Download the compressed, production jQuery 3.1.0 slim build" are shown. At the bottom, it says you can also use the slim version with its own download link.

Download jQuery

Simpan hasil download jQuery ke folder public/js.

Struktur dari folder public setelah kita menambahkan semua asset akan seperti berikut:

¹¹<https://jquery.com/download/>

```
.  
|   └── css  
|       ├── app.css  
|       ├── bootstrap.min.css  
|       └── font-awesome.min.css  
|   └── favicon.ico  
└── fonts  
    ├── FontAwesome.otf  
    ├── fontawesome-webfont.eot  
    ├── fontawesome-webfont.svg  
    ├── fontawesome-webfont.ttf  
    ├── fontawesome-webfont.woff  
    ├── fontawesome-webfont.woff2  
    ├── glyphicons-halflings-regular.eot  
    ├── glyphicons-halflings-regular.svg  
    ├── glyphicons-halflings-regular.ttf  
    ├── glyphicons-halflings-regular.woff  
    └── glyphicons-halflings-regular.woff2  
└── index.php  
└── js  
    ├── app.js  
    ├── bootstrap.min.js  
    └── jquery-3.1.0.min.js  
└── robots.txt  
└── web.config  
  
3 directories, 21 files
```

Mari kita ubah file resources/views/layouts/app.blade.php agar menggunakan semua asset ini:

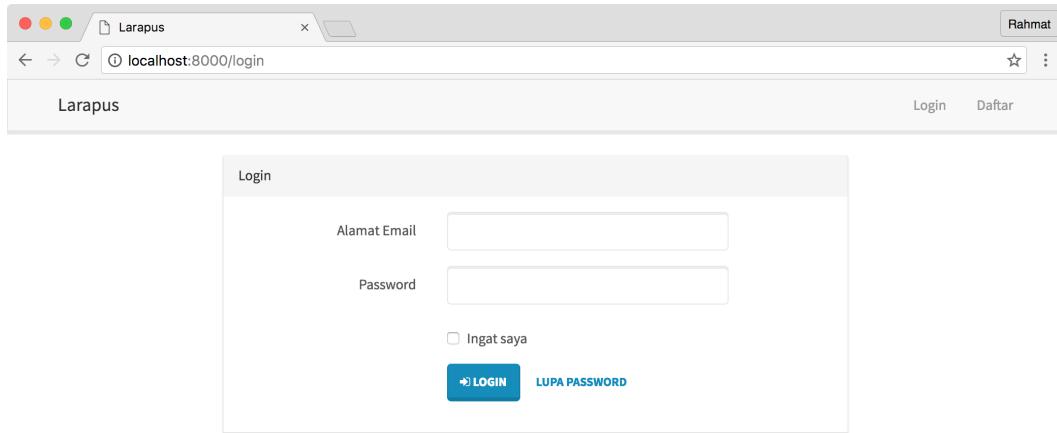
resources/views/layouts/app.blade.php

```
....  
<head>  
    ....  
    <!-- Styles -->  
    <link href="/css/font-awesome.min.css" rel='stylesheet' type='text/css'>  
    <link href="/css/bootstrap.min.css" rel="stylesheet">  
    <link href="/css/app.css" rel="stylesheet">  
    ....  
</head>  
<body>  
    ....
```

```
<!-- Scripts -->
<script src="/js/app.js"></script>
<script src="/js/jquery-3.1.0.min.js"></script>
<script src="/js/bootstrap.min.js"></script>
@yield('scripts')
</body>
....
```

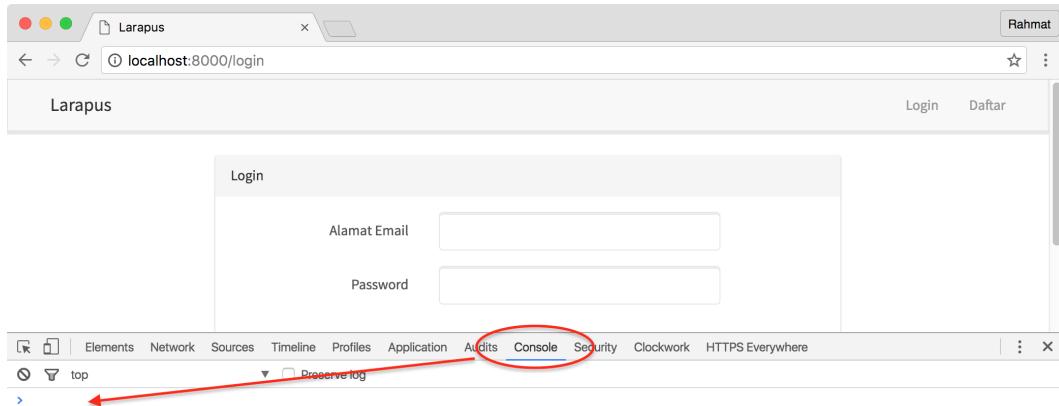
Pada syntax diatas, kita menambahkan semua asset yang sudah kita download. Kita juga menambahkan `yield` baru dengan nama `scripts`. Ini akan kita gunakan jika hendak membuat embedded script pada view.

Setelah semua perubahan diatas, tampilan Larapus akan seperti berikut:



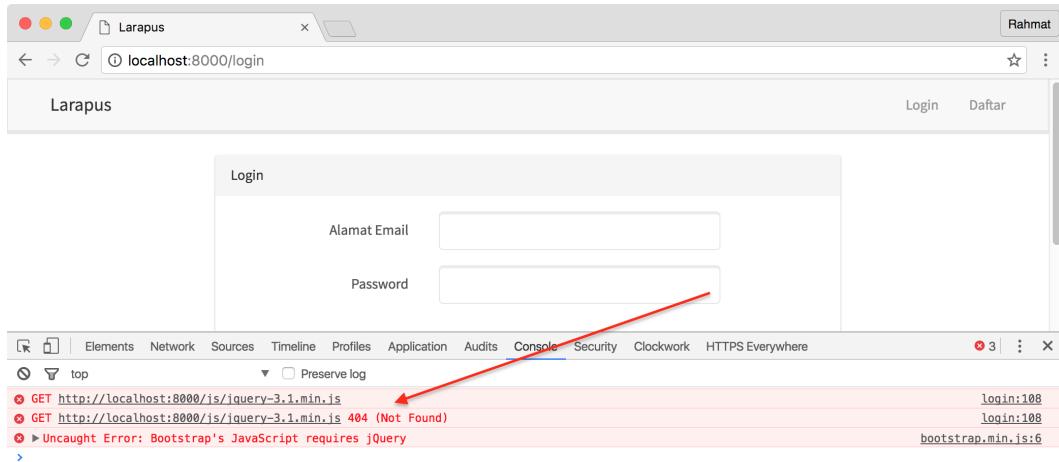
Larapus dengan theme Lumen

Pastikan semua file asset telah di load, dengan cara mengecek console pada browser. Pada chrome, kita dapat menggunakan **Inspect Element > Console**:



Semua file telah di load

Jika mendapatkan error misalnya seperti ini, maka ada link ke file asset yang salah. Silahkan diperbaiki.



File asset tidak diload

Pada contoh ini, terjadi salah ketik nama file untuk asset jQuery. Sip.



Penggunaan Elixir

Laravel juga menyediakan tools bernama elixir untuk memudahkan kita dalam memanage asset. Untuk mempersingkat pembahasan, Elixir tidak akan dibahas di buku ini. Untuk memahaminya Anda dapat mengunjungi [dokumentasi¹²](#) atau di buku [Menyelami Framework Laravel¹³](#).

3.8 Memahami Validasi Data

Laravel sangat memudahkan kita untuk melakukan validasi input dari user. Ada berbagai teknik untuk melakukan validasi, disini kita akan membahas validasi yang dilakukan pada controller.

Dengan teknik validasi di controller, kita perlu membuat instance `Illuminate\Http\Request` pada method yang akan melakukan validasi. Di Laravel, kita tidak perlu membuat instance tersebut secara manual. Kita dapat menggunakan fitur *method injection* untuk membuat instance tersebut secara otomatis.

Misalnya, kita punya code seperti ini (tidak perlu dibuat):

app/Http/MyController.php

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

Pada method `store` diatas, kita akan otomatis mendapatkan instance dari `Illuminate\Http\Request`. Berikut contoh cara kita melakukan validasi:

¹²<https://laravel.com/docs/5.3/elixir>

¹³<https://leanpub.com/bukularavel>

```
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required',
        'age' => 'required|numeric|min:17'
    ]);
    // syntax lain
}
```

Pada syntax diatas, kita melakukan validasi untuk field name dan age.

- Field name harus diisi.
- Field age harus diisi, berupa angka dan minimal 17.

Ada banyak rule validasi yang disediakan oleh Laravel, lengkapnya dapat dilihat di [dokumentasi¹⁴](#).

Ketika rule validasi tidak dapat dipenuhi oleh input dari user, maka Laravel akan mengarahkan user kembali ke halaman sebelumnya. Saat itu, akan terdapat *flash session data* dengan nama \$errors.

Kita dapat menampilkan pesan error untuk satu field dengan menggunakan \$errors->first('nama-field').

Kita juga dapat menambahkan parameter kedua berisi template untuk menampilkan error tersebut. Itulah sebabnya pada beberapa view, misalnya `resources/views/auth/login.blade.php` kita akan menemui syntax ini:

`resources/views/auth/login.blade.php`

```
{!! $errors->first('password', '<p class="help-block">:message</p>') !!}
```

Kita juga dapat mengecek apakah ada error untuk field tertentu dengan menggunakan \$errors->has('nama-field'). Ini akan bermanfaat misalnya untuk menampilkan class tertentu ketika error.

Pada beberapa view, misalnya `resources/views/auth/login.blade.php` kita akan menemui syntax seperti ini:

¹⁴<https://laravel.com/docs/5.3/validation#available-validation-rules>

resources/views/auth/login.blade.php

```
<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}>
```

Syntax diatas akan menambahkan class `has-error` ketika terdapat error pada validasi untuk field `password`.

Masih banyak topik tentang validasi yang belum kita bahas. Kita akan mempelajarinya bertahap sambil membangun Larapus. Setidaknya, ini cukup untuk menjadi bekal memahami syntax-syntax selanjutnya. Jika ingin belajar lebih detail, silahkan cek di [dokumentasi¹⁵](#) atau di buku Menyelami Framework Laravel.

3.9 Konfigurasi Role

Untuk membuat fitur Role, kita akan menggunakan package [santigarcor/laratrust¹⁶](#). Mari kita install dengan perintah:

```
composer require "santigarcor/laratrust=3.0.*"
```

Setelah selesai download, tambahkan isian berikut pada file `config/app.php` di array `providers` dan `aliases`:

config/app.php

```
....  
  
'providers' => [  
    ....  
    Laratrust\LaratrustServiceProvider::class,  
],  
....  
'aliases' => [  
    ....  
    'Laratrust' => Laratrust\LaratrustFacade::class,  
],  
....
```

Tambahkan juga baris berikut pada array `$routeMiddleware` di `app/Http/Kernel.php`:

¹⁵<https://laravel.com/docs/5.3/validation>

¹⁶<https://github.com/santigarcor/laratrust>

app/Http/Kernel.php

```
....  
protected $routeMiddleware = [  
    ....  
    'role' => \Laratrust\Middleware\LaratrustRole::class,  
    'permission' => \Laratrust\Middleware\LaratrustPermission::class,  
    'ability' => \Laratrust\Middleware\LaratrustAbility::class,  
];  
....
```

Baris diatas kita tambahkan agar kita dapat menggunakan middleware yang telah disediakan package ini. Pembahasan tentang middleware akan kita bahas di pembahasan selanjutnya.

Package ini menyediakan file konfigurasi. Untuk saat ini, kita tidak akan menggunakan file konfigurasi ini. Tapi, tidak ada salahnya kita generate file konfigurasi. Jalankan perintah berikut:

```
php artisan vendor:publish  
// ....  
// Copied File [/vendor/santigarcia/laratrust/src/config/config.php] To [/config/laratrust.php]  
// Copied File [/vendor/santigarcia/laratrust/src/config/laratrust_seeder.php] To [/config/lar  
atrustSeeder.php]
```

Package ini harus menambahkan migration dan perubahan pada model user agar berfungsi. Langkah ini bisa kita lakukan dengan menjalankan perintah berikut (jika muncul pertanyaan, ketik “yes”):

```
php artisan laratrust:setup

// Creating migration

// Tables: roles, role_user, permissions, permission_role
// A migration that creates 'roles', 'role_user', 'permissions', 'permission_role' tables will
1 be created in database/migrations directory

// Proceed with the migration creation? (yes/no) [yes]:
// > yes

// Creating migration...
// Migration successfully created!

// Creating Role model
// Role model created successfully.

// Creating Permission model
// Permission model created successfully.

// Adding LaratrustUserTrait to User model
// LaratrustUserTrait added successfully
```

Setelah perintah diatas dijalankan akan muncul file baru:

- database/migrations/xxxx_xx_xx_xxxxxxx_entrust_setup_tables.php: berisi migration untuk membuat table roles, role_user, permission dan permission_role.
- app/Role.php
- app/Permission.php

Pada file app/User.php juga akan terdapat perubahan untuk menambahkan Laratrust\Traits\LaratrustUserTrait.

Selanjutnya, kita dapat menjalankan:

```
php artisan migrate
// Migrated: xxxx_xx_xx_xxxxxxx_laratrust_setup_tables
```

untuk mendapatkan struktur database.

Struktur table untuk role dan permission

Kita hanya akan menggunakan sebagian fitur dari Laratrust. Penjelasan lebih lengkap tentang fitur Laratrust bisa dibaca di [dokumentasi¹⁷](#).

3.9.1 Membuat sample user

Pada Larapus, kita akan membuat role (level user) untuk “Member” dan “Admin”. Pembuatan role akan kita lakukan menggunakan file seeder. Kita juga akan membuat sample user untuk dua role tersebut. Mari kita buat file seeder dengan perintah:

```
php artisan make:seeder UsersSeeder
// Seeder created successfully.
```

Pada file yang dihasilkan di `database/seeds/UsersSeeder.php`, isi dengan syntax berikut:

¹⁷<https://www.gitbook.com/book/santigarcor/laratrust-docs>

database/seeds/UsersSeeder.php

```
<?php

use Illuminate\Database\Seeder;
use App\Role;
use App\User;

class UsersSeeder extends Seeder
{
    public function run()
    {
        // Membuat role admin
        $adminRole = new Role();
        $adminRole->name = "admin";
        $adminRole->display_name = "Admin";
        $adminRole->save();

        // Membuat role member
        $memberRole = new Role();
        $memberRole->name = "member";
        $memberRole->display_name = "Member";
        $memberRole->save();

        // Membuat sample admin
        $admin = new User();
        $admin->name = 'Admin Larapus';
        $admin->email = 'admin@gmail.com';
        $admin->password = bcrypt('rahasia');
        $admin->save();
        $admin->attachRole($adminRole);

        // Membuat sample member
        $member = new User();
        $member->name = "Sample Member";
        $member->email = 'member@gmail.com';
        $member->password = bcrypt('rahasia');
        $member->save();
        $member->attachRole($memberRole);
    }
}
```

Oke, syntax diatas cukup panjang. Ada empat hal yang kita lakukan:

1. Membuat role admin. Kita membuat instance dari `App\Role`, mengisi field `name` dan `display_name`. Terakhir, kita simpan dengan memanggil method `save()`.

2. Membuat role member. Caranya sama dengan tahap pertama.
3. Membuat sample admin. Disini kita membuat instance dari App\User , mengisi field name, email dan password. Isian password, kita enkripsi dengan method bcrypt(). Kita panggil method save() untuk menyimpan data user. Terakhir, kita tambahkan role admin dengan menggunakan method attachRole().
4. Membuat sample member. Caranya sama dengan tahap ketiga.



Enkripsi di Laravel

Helper bcrypt akan membuat string terenkripsi. Laravel menggunakan enkripsi AES-256-CBC, dapat dilihat di file config/app.php pada isian cipher. Salt yang digunakan untuk enkripsi adalah isian APP_KEY di file .env. Setiap membuat isian untuk field password, kita harus selalu menggunakan fungsi bcrypt().

Kita perlu menambahkan seeder ini ke database/seeds/DatabaseSeeder.php. Tambahkan baris berikut pada method run:

database/seeds/DatabaseSeeder.php

```
public function run()
{
    $this->call(UsersSeeder::class);
}
```

Kini, coba jalankan:

```
php artisan migrate:refresh --seed
```

Pastikan table users, roles dan role_user sudah terisi.

The screenshot shows the MySQL Workbench interface with the 'larapus' database selected. The 'users' table is currently open, displaying the following data:

	id	name	email	password	remember_token	created_at
1	Admin Larapus	admin@gmail.com	\$2y\$10\$IHkm4MKPhzEb2KGULdji...	NULL	2016-03-29	
2	Sample Member	member@gmail.com	\$2y\$10\$fXmEfZ/KDaZo3Sbi9CKyf...	NULL	2016-03-29	

Below the table, there is a 'TABLE INFORMATION' panel showing the following details:

- created: 3/30/16
- engine: InnoDB
- rows: 2
- size: 16.0 KIB
- encoding: utf8
- auto_increment: 3

Sample user dan role

Setelah migrate berhasil, pastikan bisa login ke Larapus sesuai data user yang kita buat di file seeder.

3.9.2 Konfigurasi Register

Kita akan membuat setiap user yang mendaftar langsung memiliki role member. Untuk itu, ubah method create pada `app/Http/Controllers/Auth/RegisterController.php` menjadi:

`app/Http/Controllers/Auth/RegisterController.php`

```
<?php
...
use App\Role;

class RegisterController extends Controller
{
    ...
    protected function create(array $data)
    {
        return User::create([
            ...
        ]);
    }
}
```

```
$user = User::create([
    'name' => $data['name'],
    'email' => $data['email'],
    'password' => bcrypt($data['password']),
]);
$memberRole = Role::where('name', 'member')->first();
$user->attachRole($memberRole);
return $user;
}
}
```

Method `create` ini digunakan Laravel untuk memproses pembuatan User dari halaman register. Logic default hanya membuat user. Pada perubahan yang kita lakukan, kita membuat user dan menambahkan role `member` pada user tersebut.

Disini, kita menggunakan:

```
Role::where('name', 'member')->first()
```

Untuk melakukan query `where` terhadap model `App\Role` (table `roles`). Query ini akan mencari record dengan field `name` yang memiliki isian `member`. Method `first()` kita panggil untuk mengambil record pertama dari hasil query tersebut.

Selanjutnya, kita menggunakan method `attachRole` untuk menambahkan role yang kita temukan pada user yang baru dibuat. Terakhir, kita kembalikan user tersebut untuk diproses oleh logic selanjutnya di Laravel.

Poinnya adalah ketika kita hendak menambahkan logic ketika membuat user, biasanya kita merubah method `create`. Ada juga method `validator` yang digunakan oleh Laravel untuk melakukan validasi data dari form register. Jika kita menambah isian baru pada form register (misalnya menambah tanggal lahir), method `validator`-lah yang kita ubah untuk melakukan validasi terhadap data tersebut.

Untuk mengecek fitur ini, silahkan mencoba mendaftar. Pastikan dibuat record baru pada table `role_user` dengan `user_id` berisi id user yang baru dibuat dan `role_id` berisi id dari role `member`.

3.10 Memahami Middleware

Di Laravel, middleware digunakan untuk melakukan logic sebelum *request* user sampai ke aplikasi atau sebelum *response* dari aplikasi sampai ke user.

Contohnya adalah middleware app/Http/Middleware/RedirectIfAuthenticated.php. Middleware ini berfungsi untuk membatasi sebuah route agar hanya bisa diakses oleh user yang belum login. Jika user sudah login, Laravel akan mengarahkan user ke halaman /home.

Kita dapat melihatnya pada method handle:

app/Http/Middleware/Authenticate.php

```
public function handle($request, Closure $next, $guard = null)
{
    if (Auth::guard($guard)->check()) {
        return redirect('/home');
    }

    return $next($request);
}
```

Agar middleware dapat digunakan, kita harus mendaftarkan middleware ke file app/Http/Kernel.php dan mengisi pada array \$routeMiddleware dengan alias tertentu:

app/Http/Kernel.php

```
protected $routeMiddleware = [
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    ...
];
```

Pada contoh ini, middleware app/Http/Middleware/Authenticate.php di daftarkan dengan alias guest.

Untuk menggunakan middleware tersebut, kita dapat menambahkan pada file route atau pada controller. Pada file route, syntax untuk menambahkan middleware akan seperti ini:

```
Route::get('nama-url', ['middleware'=>'guest', 'uses'=>'MyController@myMethod']);
```

Selain middleware biasa, Laravel juga mendukung middleware group. Fitur ini akan menggabungkan beberapa middleware dan menggunakan satu alias untuk memanggil middleware tersebut. Ini dapat kita lihat di array `$middlewareGroups` pada file `app/Http/Kernel.php`:

app/Http/Kernel.php

```
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,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
    'api' => [
        'throttle:60,1',
        'bindings',
    ],
];
```

Pada syntax diatas, alias `web` digunakan untuk memanggil 6 middleware sekaligus. Cara penggunaan middleware group sama dengan penggunaan middleware biasa.

Untuk menambahkan middleware dari controller, kita harus membuat method `__construct` pada controller. Kemudian, kita gunakan salah satu syntax berikut:

```

public function __construct()
{
    // middleware untuk semua method
    $this->middleware('nama-middleware');

    // middleware untuk method tertentu
    $this->middleware('nama-middleware', ['only'=>['methodA', 'methodB']]);

    // middleware untuk semua method, kecuali method tertentu
    $this->middleware('nama-middleware', ['except'=>['methodA', 'methodB']]);
}

```

Pada app/Http/Controllers/HomeController.php kita akan menemui syntax berikut:

app/Http/Controllers/HomeController.php

```

public function __construct()
{
    $this->middleware('auth');
}

```

Ini menandakan semua method pada HomeController harus melalui middleware auth.

Kita dapat mengecek middleware yang aktif pada tiap route dengan perintah:

```
php artisan route:list
```

2. rahmatawaludin@MalesCast: ~/Code/larapups (cat)					
Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	home		App\Http\Controllers\HomeController@index	web,auth
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
	POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest
	POST	logout		App\Http\Controllers\Auth\LoginController@logout	web
	POST	password/email		App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web,guest
	GET HEAD	password/reset		App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web,guest
	POST	password/reset		App\Http\Controllers\Auth\ResetPasswordController@reset	web,guest
	GET HEAD	password/reset/{token}		App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web,guest
	GET HEAD	register		App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest
	POST	register		App\Http\Controllers\Auth\RegisterController@register	web,guest

Mengecek middleware dari route list

Dari hasil perintah diatas, route /home atau dashboard menggunakan middleware group web dan middleware auth. Artinya, jika kita mencoba mengunjungi URL tersebut tanpa login, kita akan diarahkan ke halaman login. Kita dapat mengeceknya dengan mencoba mengunjungi halaman tersebut sebagai user yang belum login, pastikan diarahkan ke halaman login. Sip.



Lebih lanjut tentang middleware

Masih banyak fitur middleware yang belum kita bahas. Pada buku ini, kita akan menggunakan middleware yang sudah disediakan oleh Laravel dan package yang kita install. Kita juga akan membuat middleware sederhana di hari 6. Jika ingin mempelajari lebih lanjut tentang middleware, silahkan baca di [dokumentasi¹⁸](#) atau baca di buku Menyelami Framework Laravel.

3.11 Ringkasan

Di Hari 3 ini, saya harap Anda telah memahami bagaimana aplikasi yang telah dibuat, poin-poin yang telah kita bahas yaitu:

- Mempersiapkan tampilan aplikasi
- Mempersiapkan struktur database
- Memahami layouting di Laravel
- Memahami teknik penggunaan asset di Laravel
- Memahami teknik validasi input di controller
- Melakukan konfigurasi authentikasi dan role
- Memahami penggunaan middleware di Laravel

Pada hari 4, kita akan memulai implementasi dari fitur-fitur admin. Spirit! :)

¹⁸<https://laravel.com/docs/5.3/middleware>

4. Hari 4 : Develop Fitur Admin

Keamanan adalah fitur yang sangat penting dalam membuat aplikasi. Dari sekian banyak cara untuk mengamankan aplikasi, dua topik yang saling berkaitan adalah authentikasi dan authorisasi.

Authentikasi dalam konteks aplikasi merupakan proses validasi user ketika memasuki sistem dengan memastikan *credential* (username dan password) yang diberikan oleh user sesuai dengan data di database. *Authorisasi* merupakan proses verifikasi hak akses user terhadap resources tertentu di sistem. Ada banyak cara untuk mengatur authorisasi, salah satunya dengan menggunakan RBAC (Role Based Access Control). Dalam RBAC, seorang User diberi Role tertentu misalnya grup Admin. Kemudian, hak akses user ini akan dicek berdasarkan group yang dia miliki.

Pada hari sebelumnya, kita telah melakukan konfigurasi untuk authentikasi. Kita juga telah melakukan konfigurasi untuk sebagian dari proses authorisasi yaitu dengan menambahkan role. Pada pembahasan hari ini, kita akan membahas lebih lanjut bagaimana menggunakan role untuk membatasi akses sambil membuat CRUD untuk Penulis dan Buku.

4.1 Penggunaan Route Group

Fitur route group di Laravel berguna untuk menggunakan middleware pada banyak route dan menambahkan prefix pada URL. Kita akan membuat agar semua URL untuk fitur yang diakses oleh Admin memiliki awalan /admin pada URL nya.

Buatlah syntax berikut pada file route:

routes/web.php

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth']], function () {
    // Route diisi disini...
});
```

Kita menggunakan opsi `prefix` untuk menambahkan awalan `/admin` pada setiap route di dalam group tersebut. Route group ini akan kita gunakan untuk menyimpan semua routing untuk fitur yang dapat diakses admin.

Kita juga menggunakan parameter `middleware` dengan isi `auth`. Middleware `auth` digunakan untuk membatasi agar semua route di dalam grup ini hanya bisa diakses oleh user yang telah login.

4.2 Penggunaan RESTful Resource Controller

Untuk memudahkan proses CRUD, kita dapat menggunakan skema RESTful. Skema ini populer digunakan dalam pembuatan API. Dalam REST, aksi yang kita lakukan pada sebuah URL ditentukan oleh HTTP Verb (GET, PUT, POST, DELETE, dll) yang dikirim. Penamaan URL, HTTP Verb dan jenis aksi yang dilakukan sudah disepakati oleh para programmer.

Menggunakan teknik RESTful dalam membangun fitur CRUD merupakan salah satu *best practices* dalam web development. Untuk memahami lebih lanjut tentang REST, silahkan baca di [wikipedia¹](#).

Kabar baiknya, Laravel dapat meng-generate controller untuk RESTful dengan menambahkan opsi `--resource` ketika membuat controller. Mari kita praktikan dengan membuat controller untuk CRUD Penulis:

```
php artisan make:controller AuthorsController --resource
// Controller created successfully.
```

Akan muncul file baru di `app/Http/Controllers/AuthorsController.php` dengan method `index`, `create`, `store`, `show`, `edit`, `update` dan `destroy`. Semua method

¹https://en.wikipedia.org/wiki/Representational_state_transfer

tersebut merupakan bagian dari skema RESTful. Pembahasan untuk tiap method, akan kita bahas sambil membangun tiap fitur Larapus.

Selanjutnya, kita perlu membuat routing untuk controller yang sudah dibuat. Kabar baiknya, kita tidak perlu membuat route untuk tiap method secara manual. Kita dapat menggunakan `Route::resource()` untuk membuat routing secara otomatis.

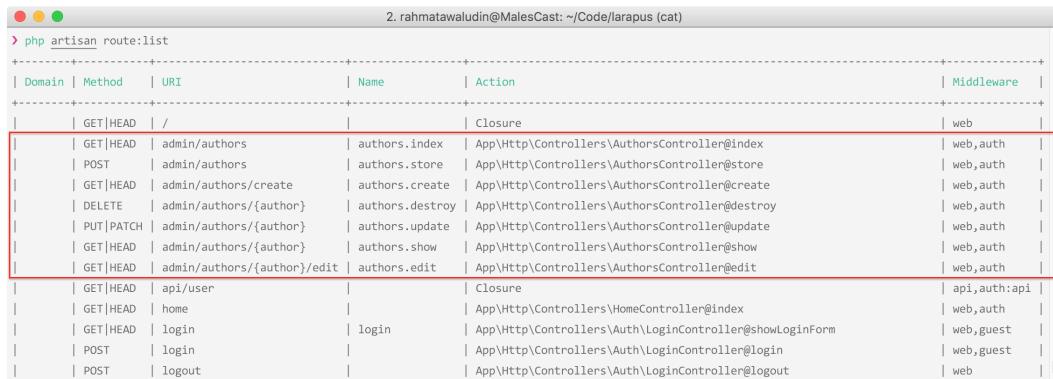
Tambahkan baris berikut pada route group yang telah kita buat:

`routes/web.php`

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth']], function () {
    Route::resource('authors', 'AuthorsController');
});
```

Kita dapat mengecek route yang dibuat dengan perintah:

```
php artisan route:list
```



2. rahmatawaludin@MalesCast: ~/Code/larapus (cat)						
Domain	Method	URI	Name	Action	Middleware	
	GET HEAD	/		Closure		web
	GET HEAD	admin/authors	authors.index	App\Http\Controllers\AuthorsController@index		web,auth
	POST	admin/authors	authors.store	App\Http\Controllers\AuthorsController@store		web,auth
	GET HEAD	admin/authors/create	authors.create	App\Http\Controllers\AuthorsController@create		web,auth
	DELETE	admin/authors/{author}	authors.destroy	App\Http\Controllers\AuthorsController@destroy		web,auth
	PUT PATCH	admin/authors/{author}	authors.update	App\Http\Controllers\AuthorsController@update		web,auth
	GET HEAD	admin/authors/{author}	authors.show	App\Http\Controllers\AuthorsController@show		web,auth
	GET HEAD	admin/authors/{author}/edit	authors.edit	App\Http\Controllers\AuthorsController@edit		web,auth
	GET HEAD	api/user		Closure		api,auth:api
	GET HEAD	home		App\Http\Controllers\HomeController@index		web,auth
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm		web,guest
	POST	login		App\Http\Controllers\Auth\LoginController@login		web,guest
	POST	logout		App\Http\Controllers\Auth\LoginController@logout		web

Hasil RESTful Routing

Pada output diatas, Laravel telah membuat beberapa route. Prefix `admin` juga telah ditambahkan pada tiap route yang dihasilkan. Pembahasan untuk tiap route akan kita bahas pada pembahasan selanjutnya.

4.3 Persiapan Model dan Migration

Mari kita buat model berikut migration untuk Penulis dan Buku:

```
php artisan make:model Author -m
// Model created successfully.
// Created Migration: xxxx_xx_xx_xxxxxx_create_authors_table

php artisan make:model Book -m
// Model created successfully.
// Created Migration: xxxx_xx_xx_xxxxxx_create_books_table
```

Sengaja kita menggunakan Author dan Book untuk modelnya, biar lebih mudah di Laravelnya. Selanjutnya, kita akan menggunakan istilah Author dan Penulis maupun Book dan Buku secara bergantian.

Pada migration untuk Author, Kita hanya perlu menambahkan field `name` pada table `authors` untuk menyimpan nama penulis:

database/migrations/yyyy_xx_xx_xxxxxx_create_authors_table.php

```
public function up()
{
    Schema::create('authors', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->timestamps();
    });
}
```

Selanjutnya, kita ubah method `up` pada migration untuk table `books`:

database/migrations/yyyy_xx_xx_xxxxxx_create_books_table.php

```
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title');
        $table->integer('author_id')->unsigned();
        $table->integer('amount')->unsigned();
        $table->string('cover')->nullable();
        $table->timestamps();

        $table->foreign('author_id')->references('id')->on('authors')
            ->onDelete('cascade')->onUpdate('cascade');
    });
}
```

Pada table books kita menambahkan field title, author_id, amount (total jumlah buku) dan cover. Isian author_id kita gunakan untuk mencatat relasi ke table author itulah sebabnya kita menambahkan method unsigned dan memanggil method foreign().

Setelah dua migration ini selesai di konfigurasi, jalankan perintah berikut untuk mendapatkan table authors dan books di database:

```
php artisan migrate
// Migrated: xxxx_xx_xx_xxxxxx_create_authors_table
// Migrated: xxxx_xx_xx_xxxxxx_create_books_table
```

4.4 Penggunaan Mass Assignment

Kita telah belajar untuk membuat record dengan mengubah fieldnya satu-persatu pada Eloquent:

```
$author = new App\Author;
$author->name = "Aam Amiruddin";
$author->save();
```

Meskipun bisa kita lakukan, namun cara diatas cukup merepotkan karena kita harus memanggil new dan save.

Jika ingin lebih cepat, kita dapat menggunakan fitur *mass assignment*. Dengan fitur ini, kita menentukan field apa saja yang boleh diisi langsung dengan melakukan passing array.

Kita dapat membuatnya seperti ini:

app/Author.php

```
....  
class Author extends Model  
{  
    protected $fillable = ['name'];  
}
```

Setelah menambah baris diatas, kita dapat membuat Author dengan syntax berikut:

```
App\Author::create(['name'=>'Aam Amiruddin']);
```

Untuk proses update pun bisa kita lakukan lebih cepat, seperti ini:

```
$author = App\Author::find(1);  
$author->update(['name'=>'Salim A Fillah']);
```

Mari kita buat juga mass assignment untuk model Book:

app/Book.php

```
....  
class Book extends Model  
{  
    protected $fillable = ['title', 'author_id', 'amount'];  
}
```

Field cover sengaja tidak kita aktifkan untuk mass assignment karena kita akan menggunakan logic terpisah untuk mengisinya.



Security dan Mass Assignment

Penggunaan mass assignment, meskipun memudahkan, harus diperhatikan field apa yang diizinkan untuk melakukan mass assignment. Pastikan hanya field-field yang diizinkan diisi oleh User yang diisi pada array `$fillable`.

Misalnya, kita memiliki field `premium` pada table `users` yang menandakan bahwa user merupakan seorang user dengan level premium setelah dia membayar. Kita sebaiknya tidak menambahkan field `premium` ke array `$fillable` untuk menghindari user mengubah status premium nya secara manual. Dengan cara ini, untuk mengubah isi field `premium`, kita menggunakan cara manual.

4.5 Persiapan Relasi One-to-Many di Eloquent

Relasi yang kita bangun antara Penulis dan Buku adalah one-to-many dimana seorang penulis dapat memiliki banyak buku. Sebagai [ORM](#)², Eloquent memiliki fitur untuk memudahkan kita melakukan query ke relasi. Sehingga, kita tidak perlu menggunakan *join* manual pada database.

Untuk memberitahu Laravel relasi one-to-many antara Penulis dan Buku, tambahkan baris berikut pada model Author dan Book:

app/Author.php

```
....  
class Author extends Model  
{  
    ....  
    public function books()  
    {  
        return $this->hasMany('App\Book');  
    }  
}  
....
```

app/Book.php

```
....  
class Book extends Model  
{  
    public function author()  
    {  
        return $this->belongsTo('App\Author');  
    }  
}
```

Method `books` pada model Author akan menghasilkan semua model Book yang memiliki `author_id` dari Author tersebut. Sedangkan method `author` pada model Book akan mengembalikan model Author yang berelasi pada Buku tersebut.

²https://en.wikipedia.org/wiki/Object-relational_mapping

Di Laravel, untuk membuat relasi ini, kita harus membuat field `author_id` pada table `books` dan field `id` pada table `authors`. Dua field ini, telah kita buat pada tahapan sebelumnya.

Pada buku ini, kita akan selalu mengikuti aturan Laravel untuk membuat relasi. Jika ingin menggunakan nama field yang berbeda, silahkan cek [dokumentasi](#)³ atau buku *Menyelami Framework Laravel*.

4.6 Menyiapkan Sample Buku dan Penulis

Ketika development dengan Laravel, sangat disarankan menggunakan sample data sebelum fitur CRUD di buat. Mari kita lakukan hal yang sama dengan membuat sample untuk Penulis dan Buku menggunakan seeder.

Kita buat seeder dengan menjalankan:

```
php artisan make:seeder BooksSeeder
```

Pada file seeder yang dihasilkan isi method `run` dengan syntax berikut:

`database/seeds/BooksSeeder.php`

```
<?php
...
use App\Author;
use App\Book;

class BooksSeeder extends Seeder
{
    public function run()
    {
        // Sample penulis
        $author1 = Author::create(['name'=>'Mohammad Fauzil Adhim']);
        $author2 = Author::create(['name'=>'Salim A. Fillah']);
        $author3 = Author::create(['name'=>'Aam Amiruddin']);

        // Sample buku
        $book1 = Book::create(['title'=>'Kupinang Engkau dengan Hamdalah',
            'amount'=>3, 'author_id'=>$author1->id]);
    }
}
```

³<https://laravel.com/docs/5.3/eloquent-relationships#one-to-many>

```

$book2 = Book::create(['title'=>'Jalan Cinta Para Pejuang',
    'amount'=>2, 'author_id'=>$author2->id]);
$book3 = Book::create(['title'=>'Membingkai Surga dalam Rumah Tangga',
    'amount'=>4, 'author_id'=>$author3->id]);
$book4 = Book::create(['title'=>'Cinta & Seks Rumah Tangga Muslim',
    'amount'=>3, 'author_id'=>$author3->id]);
}
}

```

Pada sample ini, kita membuat 3 Penulis dan 4 buku. Tambahkan class ini ke database/seeds/DatabaseSeeder.php dengan menambah syntax berikut:

database/seeds/DatabaseSeeder.php

```

public function run()
{
    $this->call(UsersSeeder::class);
    $this->call(BooksSeeder::class);
}

```

Untuk mendapatkan sample buku ini, mari kita refresh database:

```
php artisan migrate:refresh --seed
```

4.7 Menampilkan Data dengan DataTable

Di Larapus, kita akan menggunakan [DataTable⁴](#) untuk menampilkan data. DataTable merupakan library javascript yang populer digunakan untuk menampilkan data dalam bentuk table yang terintegrasi dengan fitur sorting, searching dan pagination.

Menggunakan DataTable, setiap proses untuk mengakses data akan dilakukan dengan ajax, sehingga aplikasi kita terasa lebih cepat dibandingkan menggunakan fitur pagination bawaan Laravel.

Untuk menggunakan DataTable, perlu mendapatkan file javascript, css dan image yang dibutuhkan. Semuanya bisa didapatkan dari web DataTable. Untuk memudahkan, kita juga bisa menyalin file-file ini dari sample source code.

Berikut file dan folder yang harus di salin:

⁴<https://datatables.net>

- public/css/dataTables.bootstrap.css
- public/css/jquery.dataTables.css
- public/js/dataTables.bootstrap.min.js
- public/js/jquery.dataTables.min.js
- Folder public/images

Kemudian kita load asset tersebut dari resources/views/layouts/app.blade.php:

resources/views/layouts/app.blade.php

```
....  
<link href="/css/app.css" rel="stylesheet">  
<link href="/css/jquery.dataTables.css" rel="stylesheet">  
<link href="/css/dataTables.bootstrap.css" rel="stylesheet">  
....  
<script src="/js/jquery.dataTables.min.js"></script>  
<script src="/js/dataTables.bootstrap.min.js"></script>  
@yield('scripts')  
....
```

Untuk menggunakan DataTable, kita harus membuat response yang sesuai dari server. Kita akan menggunakan package [yajra/laravel-datables-oracle](#)⁵ untuk memudahkan proses pembuatan response ini.

Install dengan perintah:

```
composer require yajra/laravel-datables-oracle=~6.0
```

Setelah proses download selesai, tambahkan baris berikut pada isian providers di config/app.php:

⁵<https://github.com/yajra/laravel-datables>

config/app.php

```
....  

'providers' => [  

    ....  

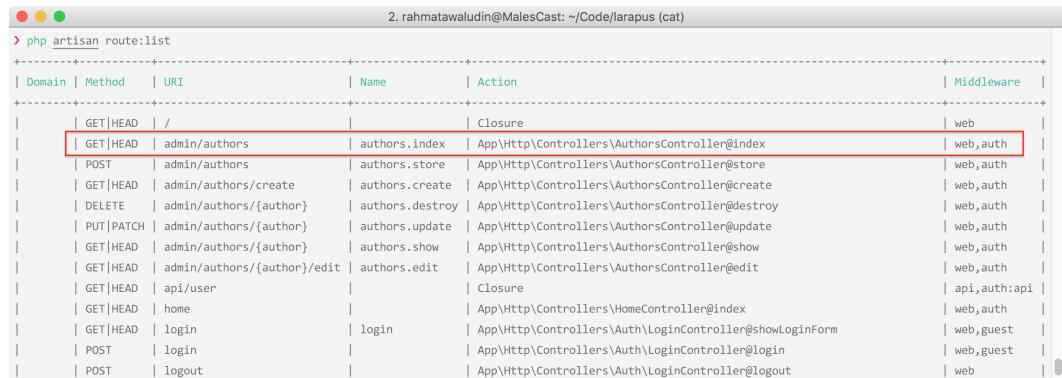
    Yajra\Datatables\DatatablesServiceProvider::class,  

]  

....
```

Mari kita gunakan DataTable untuk menampilkan data Penulis yang telah ada di Larapus. Dalam skema REST, untuk menampilkan daftar data kita harus memanggil dengan url /resources dengan HTTP VERB GET. Resources yang kita miliki adalah authors dengan *base url* di /admin/authors.

Sesuai route yang digenerate oleh Laravel, url /admin/authors dengan Http Verb GET akan di *handle* oleh method index di AuthorsController.



2. rahmatauludin@MalesCast: ~/Code/larapus (cat)						
Domain	Method	URI	Name	Action	Middleware	
	GET HEAD	/		Closure		
	GET HEAD	admin/authors	authors.index	App\Http\Controllers\AuthorsController@index	web,auth	
	POST	admin/authors	authors.store	App\Http\Controllers\AuthorsController@store	web,auth	
	GET HEAD	admin/authors/create	authors.create	App\Http\Controllers\AuthorsController@create	web,auth	
	DELETE	admin/authors/{author}	authors.destroy	App\Http\Controllers\AuthorsController@destroy	web,auth	
	PUT PATCH	admin/authors/{author}	authors.update	App\Http\Controllers\AuthorsController@update	web,auth	
	GET HEAD	admin/authors/{author}	authors.show	App\Http\Controllers\AuthorsController@show	web,auth	
	GET HEAD	admin/authors/{author}/edit	authors.edit	App\Http\Controllers\AuthorsController@edit	web,auth	
	GET HEAD	api/user		Closure	api,auth:api	
	GET HEAD	home		App\Http\Controllers\HomeController@index	web,auth	
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest	
	POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest	
	POST	logout		App\Http\Controllers\Auth\LoginController@logout	web	

Handle untuk /admin/authors

Mari kita tambahkan logic pada method index:

app/Http/Controllers/AuthorsController.php

```
....  
public function index()  
{  
    return view('authors.index');  
}  
....
```

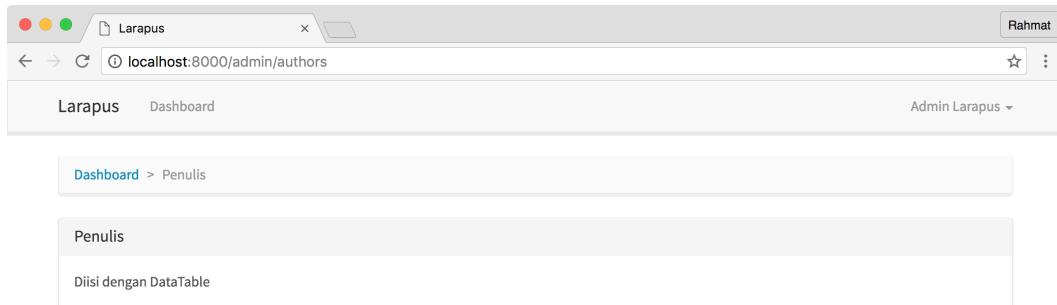
Di Laravel, biasanya view untuk sebuah resource dikelompokkan dalam sebuah folder. Begitupun di Larapus, semua view untuk CRUD authors akan kita simpan di folder resources/views/authors. Biasanya, nama view akan disesuaikan dengan nama method. Itulah sebabnya pada method index kita menggunakan view authors.index atau resources/views/authors/index.blade.php.

Silahkan buat folder dan viewnya. Kemudian isi dengan syntax berikut:

resources/views/authors/index.blade.php

```
@extends('layouts.app')  
  
@section('content')  
    <div class="container">  
        <div class="row">  
            <div class="col-md-12">  
                <ul class="breadcrumb">  
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>  
                    <li class="active">Penulis</li>  
                </ul>  
                <div class="panel panel-default">  
                    <div class="panel-heading">  
                        <h2 class="panel-title">Penulis</h2>  
                    </div>  
  
                    <div class="panel-body">  
                        Diisi dengan DataTable  
                    </div>  
                </div>  
            </div>  
        </div>  
    </div>  
@endsection
```

Kita dapat melihat view ini dengan mengunjungi /admin/authors.



Tampilan awal daftar penulis

Untuk memudahkan navigasi ke halaman author, mari kita buat link navigasi. Tambahkan baris berikut pada `resources/views/layouts/app.blade.php`:

`resources/views/layouts/app.blade.php`

```
....  

<!--Left Side Of Navbar -->  

<ul class="nav navbar-nav">  

@if (Auth::check())  

    <li><a href="{{ url('/home') }}>Dashboard</a></li>  

    <li><a href="{{ route('authors.index') }}>Penulis</a></li>  

@endif  

</ul>  

....
```

Untuk mendapatkan link ke halaman Penulis, kita menggunakan method `route()` untuk mendapatkan link dari [*named routes*⁶](#) dengan nama `authors.index` yang dihasilkan dari routing. Untuk mengetahui named route lain yang dihasilkan oleh Laravel, kita dapat menggunakan:

```
php artisan route:list
```

⁶<https://laravel.com/docs/5.3/routing#named-routes>

2. rahmatawaludin@MalesCast: ~/Code/larapus (cat)					
> php artisan route:list		Name	Action	Middleware	
Domain	Method	URI			
	GET HEAD	/	Closure	web	
	GET HEAD	admin/authors	authors.index	web,auth	
	POST	admin/authors	authors.store	web,auth	
	GET HEAD	admin/authors/create	authors.create	web,auth	
	DELETE	admin/authors/{author}	authors.destroy	web,auth	
	PUT PATCH	admin/authors/{author}	authors.update	web,auth	
	GET HEAD	admin/authors/{author}	authors.show	web,auth	
	GET HEAD	admin/authors/{author}/edit	authors.edit	web,auth	
	GET HEAD	api/user	Closure	api,auth:api	
	GET HEAD	home	App\Http\Controllers\HomeController@index	web,auth	
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
	POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest
	POST	logout		App\Http\Controllers\Auth\LoginController@logout	web

Named Route

Ini tampilan yang akan kita dapatkan:

Link navigasi

Sekarang, mari kita tambahkan logic di controller:

app/Http/Controllers/AuthorsController.php

```
<?php

...
use App\Author;
use Yajra\Datatables\Html\Builder;
use Yajra\Datatables;

class AuthorsController extends Controller
{
    public function index()
}
```

```
    —— return view('authors.index');
}

public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $authors = Author::select(['id', 'name']);
        return Datatables::of($authors)->make(true);
    }

    $html = $htmlBuilder
        ->addColumn(['data' => 'name', 'name'=>'name', 'title'=>'Nama']);

    return view('authors.index')->with(compact('html'));
}
....
```

Ada dua hal yang kita lakukan pada controller ini:

- Syntax `$request->ajax()` kita gunakan untuk mengecek apakah request berupa ajax. Ini kita lakukan untuk menghandle request ajax dari DataTable di view. Pada logic ini, kita menggunakan instance `Yajra\Datatables\Datatables` untuk membuat response dengan struktur data yang sesuai dengan kebutuhan DataTable.
- Sytnax kedua, kita membuat instance dari `Yajra\Datatables\Html\Builder` yang akan digunakan untuk meng-*generate* html dan javascript DataTable di view. Disini, kita hanya akan menampilkan nama dari penulis. Pada saat memanggil view, kita juga melakukan passing variable `$html` yang berisi instance dari `Yajra\Datatables\Html\Builder`.

Untuk dokumentasi lebih lengkap dari penggunaan DataTable di Laravel, silahkan kunjungi dokumentasi dari [Yajra\Datatables](#)⁷.

Pada view, kita tambahkan baris berikut:

⁷<http://datatables.yajrabox.com>

resources/views/authors/index.blade.php

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

@section('content')
    ...
    <div class="panel-body">
        Diisi dengan DataTable
        {!! $html->table(['class'=>'table-striped']) !!}
    </div>
    ...
@endsection

@section('scripts')
    {!! $html->scripts() !!}
@endsection
```

Pada syntax diatas, kita menggunakan `$html->table()` untuk meng-*generate* html dari DataTable. Kita juga menambahkan array berisi `class table-striped` yang akan ditambahkan pada table yang digenerate.

Kita menggunakan `@section('scripts')` untuk mengisi `@yield('scripts')` pada `resources/views/layouts/app.blade.php`. Disini kita menggunakan method `$html->scripts()` untuk meng-*generate* javascript dari DataTable.

Tampilan akhir dari halaman ini akan seperti berikut:

A screenshot of a web browser window titled "Larapus". The URL is "localhost:8000/admin/authors". The page shows a table with three entries: Salim A. Fillah, Mohammad Fauzil Adhim, and Aam Amiruddin. The table includes a search bar at the top right and navigation buttons (PREVIOUS, 1, NEXT) at the bottom right.

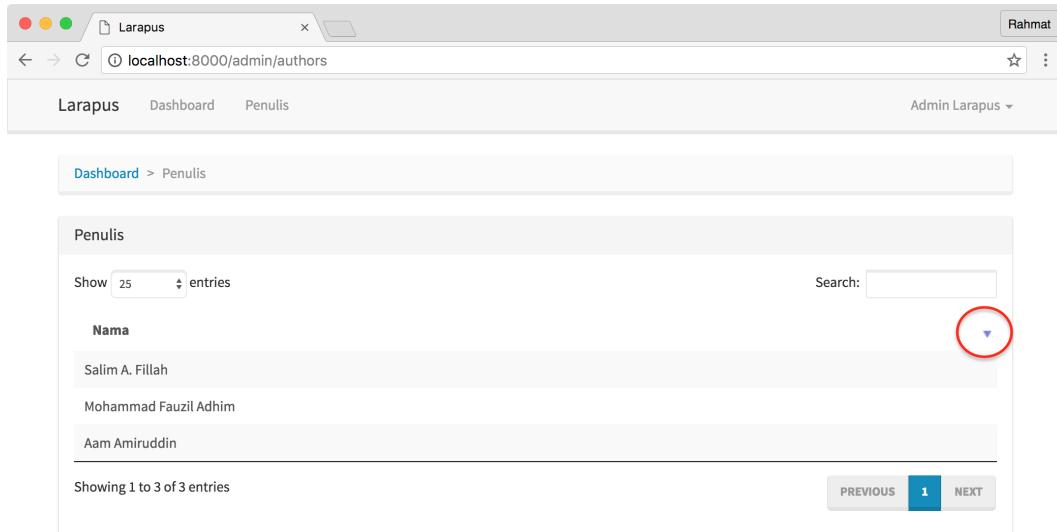
Menampilkan daftar Penulis dengan DataTable

Pastikan fitur search dari DataTable berfungsi:

A screenshot of a web browser window titled "Larapus". The URL is "localhost:8000/admin/authors". The page shows a table with one entry: Salim A. Fillah. A red oval highlights the search input field, which contains the value "salim". The table includes a search bar at the top right and navigation buttons (PREVIOUS, 1, NEXT) at the bottom right.

Pencarian dengan DataTable

Dan fitur sorting:

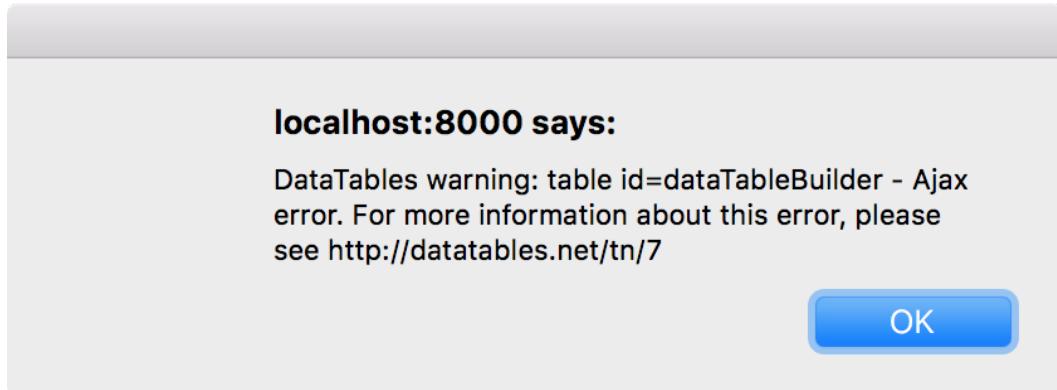


The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/authors'. The page has a header with 'Larapus', 'Dashboard', 'Penulis', and 'Admin Larapus'. Below the header is a breadcrumb navigation 'Dashboard > Penulis'. The main content area is titled 'Penulis' and contains a table with three entries: 'Salim A. Fillah', 'Mohammad Fauzil Adhim', and 'Aam Amiruddin'. The table includes columns for 'Nama' and a sorting icon (down arrow) which is circled in red. At the bottom of the table, it says 'Showing 1 to 3 of 3 entries' and has 'PREVIOUS', 'NEXT', and a page number '1'.

Sorting dengan DataTable

4.8 Error di DataTable?

Dalam menggunakan DataTable di Laravel, terkadang akan ditemui error seperti ini:



The screenshot shows a modal dialog box with the title 'localhost:8000 says:'. The message inside the box reads: 'DataTables warning: table id=dataTableBuilder - Ajax error. For more information about this error, please see <http://datatables.net/tn/7>'. In the bottom right corner of the dialog, there is a blue 'OK' button.

Error di DataTable

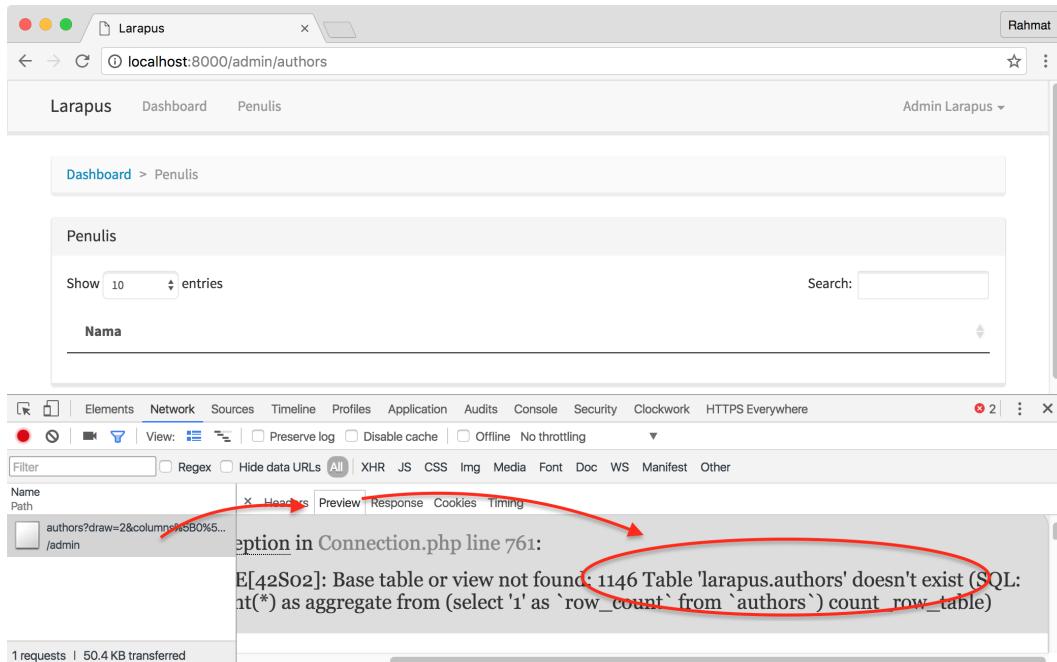
Kita dapat mendapatkan petunjuk apa yang salah dengan melihat *response* dari request ajax yang dilakukan. Untuk melihat request ajax, kita dapat menggunakan

Chrome Dev Tools. Buka tab “Network” kemudian isi isian spasi pada kotak search dan cek response dari request yang dilakukan.

The screenshot shows the Chrome Dev Tools Network tab. At the top, there is a search bar labeled "Search:" with a magnifying glass icon. Below the search bar, there is a table with the following columns: Name, Path, Method, Status Text, Type, Initiator, Size Content, Time, Latency, and Tim. One row in the table is highlighted with red arrows pointing to the "Path" and "Status Text" columns. The "Path" column shows the URL "authors?draw=2&columns%5B0%5D%5Bdata%5D=name&columns%5B0%5D%5B... /admin". The "Status Text" column shows "500 Internal S...". The "Initiator" column shows "jquery-3.1.0.min.j... Script". The "Size Content" column shows "50.4 KB" and "49.4 KB". The "Time" and "Latency" columns show "154 ms" and "149 ms" respectively. At the bottom of the Network tab, it says "1 requests | 50.4 KB transferred".

Mengecek request ajax

Klik pada request tersebut, pilih tab “Preview” maka akan tampil detail dari error yang terjadi:



Detail dari request

Pada error yang terjadi disini, terlihat table authors belum ada di database. Yang artinya kita harus melakukan migrate. Pesan error disini tentunya akan berbeda tergantung kesalahan yang kita buat ketika development. Sip

4.9 Error NotFoundHttpException di Datatable?

Error lain yang sering ditemui ketika bekerja dengan DataTable adalah `NotFoundHttpException`. Error ini biasanya terjadi karena ada salah dalam konfigurasi field yang ditampilkan di DataTable. Cobalah salah satu cara dibawah ini:

- Pastikan semua field yang tidak terdapat di database memiliki opsi `searchable` dan `orderable` diisi dengan `false`.
- Temukan konfigurasi field yang salah. Caranya, cobalah menampilkan hanya 1 field. Cek apakah errornya masih ada. Jika tidak ada error, terus tambah field

satu-persatu hingga error muncul. Jika sudah ditemukan error, perbaikilah konfigurasi untuk field tersebut.

- Setiap kali melakukan perubahan pada field yang ditampilkan, cobalah untuk logout dan login kembali.
- Cobalah menggunakan Query Builder biasa sebagai datasource. Cek dokumentasi [berikut⁸](#) (cek bagian source code).
- Cobalah menggunakan apache dan virtual host untuk menjalankan Laravel (jangan menggunakan `php artisan serve`).

4.10 Membatasi Akses dengan Role

Kita akan mempraktekan pembatasan akses pada fitur untuk CRUD Penulis. Untuk membuatnya, kita akan menggunakan middleware `role` yang sudah disediakan oleh package `santigarcia/laratrust`.

Kita akan menambahkan middleware ini pada file route:

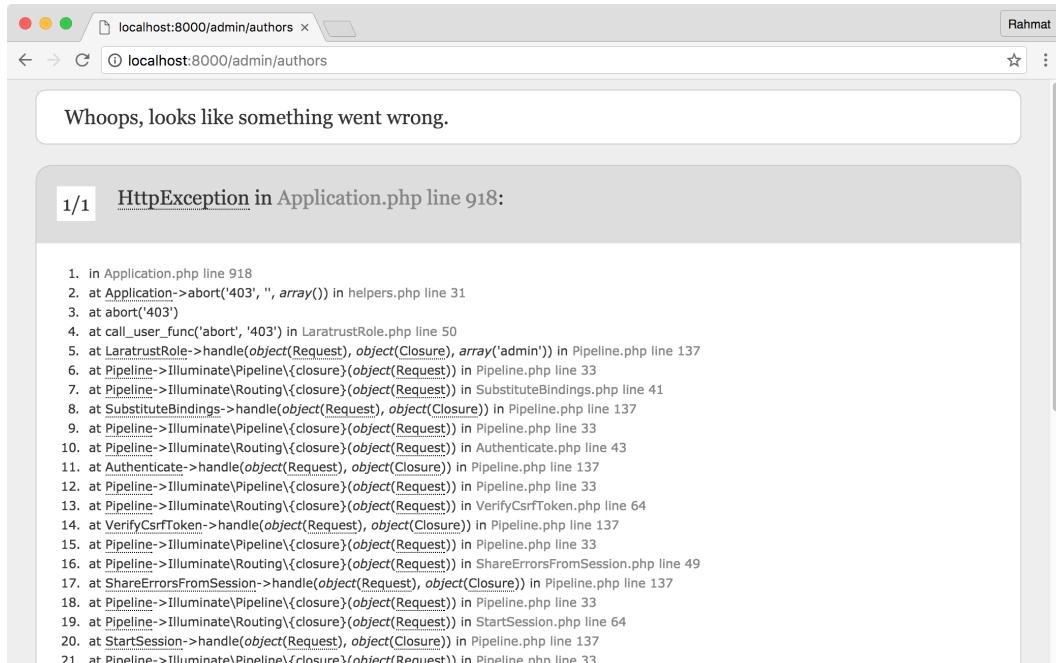
`routes/web.php`

```
....  
Route::group(['middleware' => 'web'], function () {  
    ...  
    Route::group(['prefix'=>'admin', 'middleware'=>['auth']], function () {  
        Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {  
            Route::resource('authors', 'AuthorsController');  
        });  
    });  
});
```

Middleware `role`, menerima parameter berupa role yang dicari berdasarkan field `name` di table `roles`. Jika user tidak memiliki role yang dimaksud, maka ia akan mendapatkan error 403.

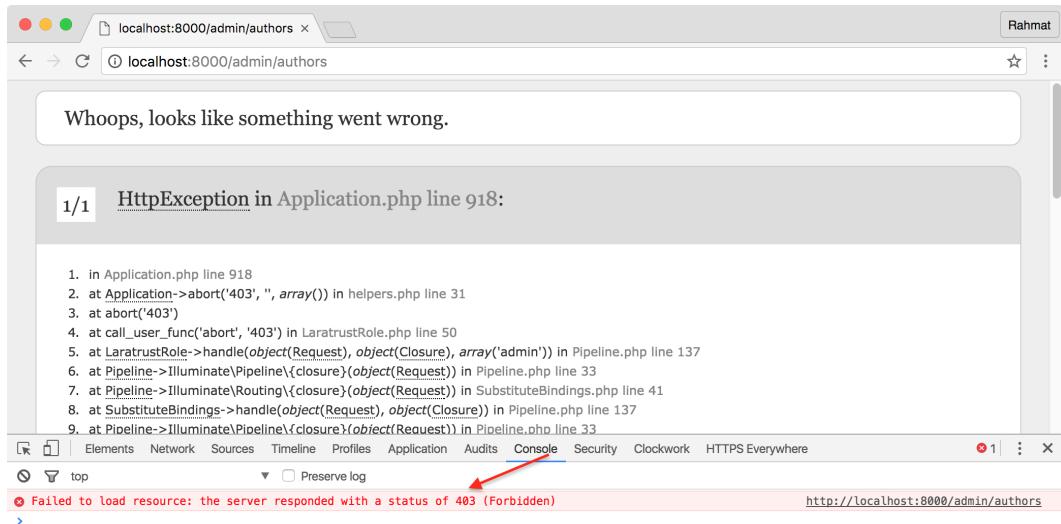
Setelah melakukan perubahan diatas, kini cobalah login sebagai member dan coba akses `/admin/authors`. Kita akan mendapatkan halaman berikut:

⁸<http://datatables.yajrabox.com/fluent/basic>



Pembatasan akses berhasil

Jika kita cek pada Chrome Dev Tools, maka akan terlihat kita mendapatkan error 403:



Error 403

Kita juga dapat membatasi agar link navigasi ke Penulis tidak muncul ketika user login sebagai member. Caranya, kita gunakan directive @role pada view. Seperti ini:

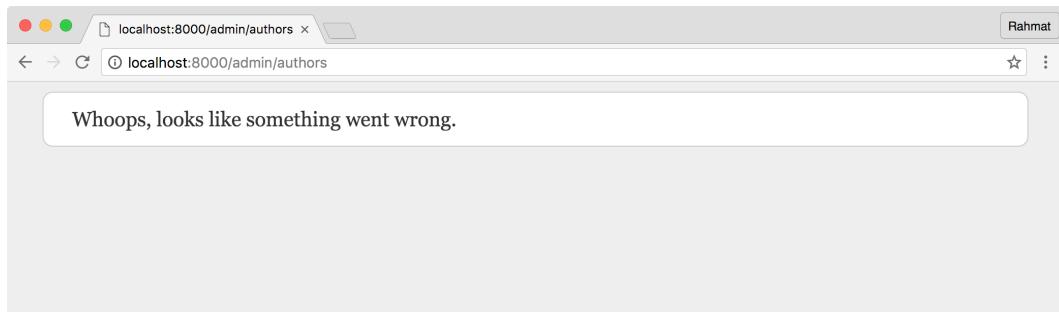
`resources/views/layouts/app.blade.php`

```
<!--Left Side Of Navbar -->
<ul class="nav navbar-nav">
    @if (Auth::check())
        <li><a href="{{ url('/home') }}>Dashboard</a></li>
        <li><a href="{{ route('authors.index') }}>Penulis</a></li>
    @endif
    @role('admin')
        <li><a href="{{ route('authors.index') }}>Penulis</a></li>
    @endrole
</ul>
```

Directive @role menerima parameter berupa role yang harus dimiliki oleh user. Jika user memiliki role ini, maka bagian pada view tersebut akan muncul. Ketika login sebagai member, pastikan link ke Penulis sudah hilang.

4.11 Penggunaan Halaman Error Custom

Menggunakan halaman error default dari sebuah framework biasanya tidak terlalu bermanfaat untuk *end user*. Pesan error yang kita lihat diatas, diberikan oleh Laravel ketika konfigurasi untuk *debug* kita set ke *true*. Jika kita ubah isian APP_DEBUG menjadi *false*, kita akan mendapatkan halaman berikut:



Error 403 ketika debug false

Pada tampilan ini, detail dari error tidak ditampilkan. Dengan teknik ini user tidak akan mengetahui detail dari error, yang akan bermanfaat untuk keamanan aplikasi. Tapi, masih kurang bermanfaat untuk user karena belum jelas apa kesalahan yang telah dilakukannya.

Di Laravel, kita dapat membuat custom error page dengan membuat view dengan nama file berupa kode error pada folder resources/views/errors. Untuk error 403, kita harus membuat file baru di resources/views/errors/403.blade.php.

Mari kita buat dengan isian berikut:

resources/views/errors/403.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>No access.</title>

    <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300" rel="stylesheet"\ type="text/css">

    <style>
        html, body {
            height: 100%;
        }

        body {
            margin: 0;
            padding: 0;
            width: 100%;
            color: #B0BEC5;
            display: table;
            font-weight: 300;
            font-family: 'Source Sans Pro';
        }

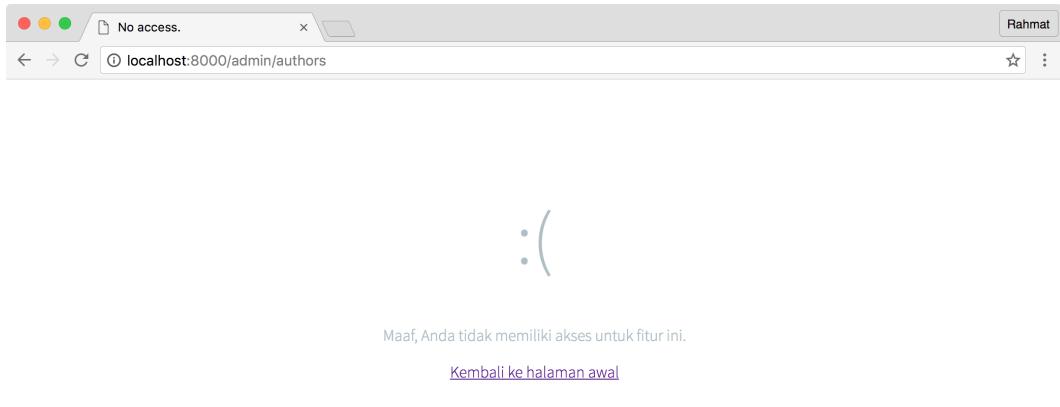
        .container {
            text-align: center;
            display: table-cell;
            vertical-align: middle;
        }

        .content {
            text-align: center;
            display: inline-block;
        }

        .title {
            font-size: 72px;
            margin-bottom: 40px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="content">
            <div class="title">:(</div>
            <p>Maaf, Anda tidak memiliki akses untuk fitur ini.</p>
        </div>
    </div>
</body>
```

```
<p><a href="{{ url('/') }}">Kembali ke halaman awal</a></p>
</div>
</div>
</body>
</html>
```

Kini, tampilan halaman error 403 akan menjadi menjadi:



Berhasil membuat custom error page 403



Kapan Debug harus diisi false?

Ketika kita telah mengupload aplikasi ke production (server), kita HARUS set debug menjadi `false`. Pada saat itu, kita dapat melihat detail error dari aplikasi pada file `storage/logs/laravel.log`.

4.12 Fitur Tambah Penulis

Mari kita lanjutkan fitur CRUD penulis dengan membuat fitur untuk menambah Penulis. Berdasarkan skema RESTful, kita akan menggunakan 2 URL untuk membuat resources:

- GET /resources/create untuk menampilkan form.
- POST /resources untuk menyimpan form.

Untuk kasus CRUD Penulis, berarti kita menggunakan GET /admin/authors/create dan POST /admin/authors. Kedua route ini, telah kita buat di langkah sebelumnya.

Pertama, kita tambahkan link untuk menambah penulis dari halaman index:

resources/views/authors/index.blade.php

```
....  
<div class="panel-body">  
  <p> <a class="btn btn-primary" href="{{ route('authors.create') }}">Tambah</a> </p>  
  {!! $html->table(['class'=>'table-striped']) !!}  
</div>  
....
```

Syntax route('authors.create') akan membuat link ke /admin/authors/create. Sesuai routing yang digenerate, URL tersebut akan di *handle* oleh method create pada AuthorsController.

Mari kita buat syntaxnya:

app/Http/Controllers/AuthorsController.php

```
public function create()  
{  
    return view('authors.create');  
}
```

Pada syntax diatas, kita menampilkan view authors.create. Berikut isi view tersebut:

resources/views/authors/create.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li><a href="{{ url('/admin/authors') }}>Penulis</a></li>
                    <li class="active">Tambah Penulis</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Tambah Penulis</h2>
                    </div>

                    <div class="panel-body">
                        {!! Form::open(['url' => route('authors.store'),
                            'method' => 'post', 'class'=>'form-horizontal']) !!}
                        @include('authors._form')
                        {!! Form::close() !!}
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Disini kita menggunakan partial view `authors._form`. Sesuai penjelasan di hari sebelumnya, penggunaan partial view akan bermanfaat ketika kita hendak menggunakan form dalam beberapa tempat.

Disini, form untuk Author akan kita gunakan untuk halaman `create` dan `edit`. Pada kasus CRUD yang lain, kita juga akan menggunakan teknik yang sama.

Berikut syntax untuk form tersebut:

resources/views/authors/_form.blade.php

```
<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}>
    {!! Form::label('name', 'Nama', ['class'=>'col-md-2 control-label']) !!}
    <div class="col-md-4">
        {!! Form::text('name', null, ['class'=>'form-control']) !!}
        {!! $errors->first('name', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group">
    <div class="col-md-4 col-md-offset-2">
        {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
    </div>
</div>
```

Pada syntax di view `authors.create`, kita juga melihat form diarahkan ke route(`'authors.store'`) yang akan menjadi URL `/admin/authors` dengan method POST. Sesuai routing yang digenerate, route tersebut akan di handle oleh method `store` di `AuthorsController`.

Berikut syntaxnya:

app/Http/Controllers/AuthorsController.php

```
public function store(Request $request)
{
    $this->validate($request, ['name' => 'required|unique:authors']);
    $author = Author::create($request->all());
    return redirect()->route('authors.index');
}
```

Ada 3 hal yang kita lakukan pada method ini:

- Melakukan validasi form untuk memastikan isian `name` diisi dan tidak ada isian `name` yang sama di table `authors`.
- Membuat Author baru dengan method `create`. Field yang diisi oleh Laravel akan disesuaikan dengan field yang diizinkan pada konfigurasi untuk mass assignment di model `App\Author`, yaitu field `name` saja.
- Mengarahkan user kembali ke halaman index author.

Cobalah membuat Penulis baru, pastikan berhasil.

The screenshot shows the Larapus admin interface at localhost:8000/admin/authors. The top navigation bar includes 'Larapus', 'Dashboard', 'Penulis', and 'Admin Larapus'. Below the header, a breadcrumb trail shows 'Dashboard > Penulis'. The main content area is titled 'Penulis' and contains a table with three entries: 'Aam Amiruddin', 'Mohammad Fauzil Adhim', and 'Salim A. Fillah'. A blue button labeled 'TAMBAH' is highlighted with a red oval. Below the table, there are buttons for 'Show 10 entries' and 'Search'. At the bottom, it says 'Showing 1 to 3 of 3 entries' and has 'PREVIOUS', 'NEXT', and a page number '1'.

Tombol tambah penulis

The screenshot shows the Larapus admin interface at localhost:8000/admin/authors/create. The top navigation bar includes 'Larapus', 'Dashboard', 'Penulis', and 'Admin Larapus'. Below the header, a breadcrumb trail shows 'Dashboard > Penulis > Tambah Penulis'. The main content area is titled 'Tambah Penulis' and contains a form with a 'Nama' input field and a blue 'SIMPAN' button.

Form tambah penulis

The screenshot shows the Larapus admin panel at localhost:8000/admin/authors. The title bar says "Larapus" and "Rahmat". The navigation bar includes "Larapus", "Dashboard", "Penulis", and "Admin Larapus". A breadcrumb trail shows "Dashboard > Penulis". The main content area is titled "Penulis" and contains a "TAMBAH" button. A table lists four entries: "Aam Amiruddin", "Mario Teguh" (which is circled in red), "Mohammad Fauzil Adhim", and "Salim A. Fillah". Below the table, it says "Showing 1 to 4 of 4 entries" and has "PREVIOUS" and "NEXT" buttons.

Berhasil menambah penulis

The screenshot shows the Larapus admin panel at localhost:8000/admin/authors/create. The title bar says "Larapus" and "Rahmat". The navigation bar includes "Larapus", "Dashboard", "Penulis", and "Admin Larapus". A breadcrumb trail shows "Dashboard > Penulis > Tambah Penulis". The main content area is titled "Tambah Penulis" and has a "Nama" input field containing "Aam Amiruddin". A validation message "The name has already been taken." is displayed below the input field. A red oval highlights both the input field and the validation message. A "SIMPAN" button is at the bottom.

Berhasil melakukan validasi nama penulis

4.13 Penggunaan Flash Messages

Salah satu tips UX yang pernah saya pelajari adalah memberikan feedback ke user ketika user telah melakukan aksi. Entah aksi yang dilakukannya sukses atau gagal. Ini berguna agar user yakin tindakan yang dilakukannya telah dieksekusi sistem.

Dalam kasus Larapus, kita akan menambahkan fitur untuk menampilkan pesan sukses ketika user telah berhasil menambah penulis.

Untuk membuat fitur ini, kita akan menggunakan fitur [Flash Session Data⁹](#) di Laravel dan alert di bootstrap. Fitur Flash Session di Laravel bekerja dengan membuat sebuah variable di Session yang hanya muncul pada request selanjutnya.

Mari kita tambahkan syntax berikut pada method store di AuthorsController:

app/Http/Controllers/AuthorsController.php

```
....  
use Session;  
  
class AuthorsController extends Controller  
{  
    public function store(Request $request)  
    {  
        $this->validate($request, ['name' => 'required|unique:authors']);  
        $author = Author::create($request->only('name'));  
        Session::flash("flash_notification", [  
            "level"=>"success",  
            "message"=>"Berhasil menyimpan $author->name"  
        ]);  
        return redirect()->route('authors.index');  
    }  
    ....  
}
```

Pada syntax diatas kita membuat flash data dengan nama `flash_notification` dan mengisinya dengan array yang memiliki key `level` dan `message`.

Key `level` akan kita gunakan untuk menampilkan jenis alert yang akan muncul di bootstrap. Ada 4 jenis alert yang bisa dibuat di bootstrap yaitu `success`, `info`, `warning` dan `danger`.

⁹<https://laravel.com/docs/5.3/session#flash-data>

Key message akan kita gunakan untuk mengisi pesan pada alert yang digenerate.

Untuk menampilkan flash data ini, kita akan menggunakan partial view di resources/views/layouts/_flash.blade.php:

resources/views/layouts/_flash.blade.php

```
@if (session()->has('flash_notification.message'))
    <div class="container">
        <div class="alert alert-{{ session()->get('flash_notification.level') }}">
            <button type="button" class="close" data-dismiss="alert" aria-hidden="true">&times;</button>
            {!! session()->get('flash_notification.message') !!}
        </div>
    </div>
@endif
```

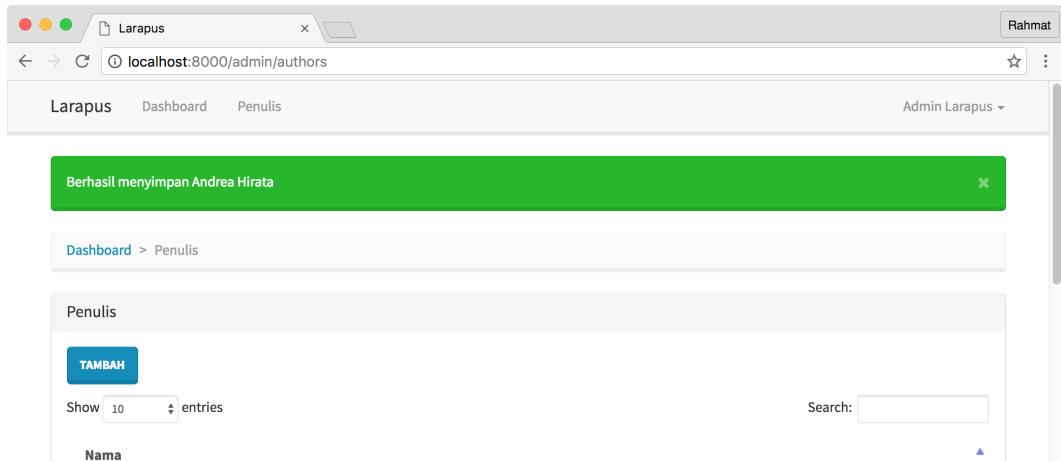
Pada view diatas, kita menggunakan session()->has() untuk mengecek apakah ada data session dengan key yang kita inginkan. Disini, kita mencari data session dengan key flash_notification.message. Setelah kita dapatkan, kita tampilkan alert dan pesannya.

Agar flash message ini bisa digunakan di semua view, kita harus memanggil partial view diatas dari resources/views/layouts/app.blade.php:

resources/views/layouts/app.blade.php

```
.....
@include('layouts._flash')
@yield('content')
....
```

Kini, ketika kita berhasil menyimpan penulis, akan muncul feedback berikut:



Berhasil menambah feedback

4.14 Penggunaan Form Model Binding

Tahap selanjutnya dari CRUD adalah membuat halaman untuk mengubah resource. Pada skema RESTful, untuk mengubah sebuah resources kita menggunakan 2 URL:

- GET `/resources/{id}/edit` untuk menampilkan form.
- PUT `/resources/{id}` untuk menyimpan form.

Berdasarkan routing yang telah digenerate, kita akan menggunakan `/admin/authors/{id}/edit` dan `/admin/authors/{id}` dengan method `PUT`. Untuk menampilkan halaman edit ini, kita akan menambahkan link pada DataTable dari halaman index author.

Ubahlah method index menjadi:

app/Http/Controllers/AuthorsController.php

```

public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $authors = Author::select(['id', 'name']);
        return Datatables::of($authors)->make(true);
    }
    return Datatables::of($authors)
        ->addColumn('action', function($author){
            return view('datatable._action', [
                'edit_url' => route('authors.edit', $author->id),
            ]);
        })
        ->make(true);
}

$html = $htmlBuilder
->addColumn(['data' => 'name', 'name'=>'name', 'title'=>'Nama']);
->addColumn(['data' => 'name', 'name'=>'name', 'title'=>'Nama'])
->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, '\
searchable'=>false]);

return view('authors.index')->with(compact('html'));
}

```

Pada syntax diatas, kita menambah field `action` pada DataTable. Untuk membuat field tersebut, kita menggunakan method `addColumn` dan mengembalikan partial view `datatable._action`. Sengaja kita menggunakan partial view karena kita akan menggunakan action seperti ini untuk DataTable yang lain.

Buatlah view `resources/views/datatable/_action.blade.php` dengan isian berikut:

resources/views/datatable/_action.blade.php

```
<a href="{{ $edit_url }}>Ubah</a>
```

Terlihat disini, kita menggunakan variable `edit_url` yang kita passing dari controller untuk membuat link ke halaman edit.

Agar field `action` ini muncul, kita menambahkan baris berikut pada instance `Yajra\Datatables\Html\Builder`:

```
->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, 'searchable'=>false]);
```

Pada syntax ini, kita akan menggunakan konten dari field action tanpa mengisi header untuk field tersebut pada table yang digenerate. Kita juga set searchable dan orderable menjad **false** agar DataTable tidak mengaktifkan pencarian dan sorting pada field tersebut.

Tampilan yang akan kita dapatkan seperti berikut:

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/authors'. The page has a header with 'Larapus', 'Dashboard', 'Penulis', and 'Admin Larapus'. Below the header, there's a breadcrumb navigation 'Dashboard > Penulis'. The main content area is titled 'Penulis' and contains a table with a single column labeled 'Nama'. The table rows list 'Aam Amiruddin', 'Aam Amirudin', 'Andrea Hirata', and 'Mario Teguh'. To the right of each row, there is a blue 'Ubah' link. A red circle highlights the 'Ubah' link for the fourth row ('Mario Teguh').

Link untuk edit

Sesuai routing yang digenerate, halaman edit akan di handle oleh method `edit` pada `AuthorsController`.

Berikut syntaxnya:

app/Http/Controllers/AuthorsController.php

```
public function edit($id)
{
    $author = Author::find($id);
    return view('authors.edit')->with(compact('author'));
}
```

Pada method ini, kita akan menampilkan view `authors.edit` sambil melakukan passing Author yang kita temukan. Kita perlu melakukan passing ini karena kita hendak menggunakan fitur **Form Model Binding**¹⁰.

Berikut syntax untuk halaman edit:

resources/views/authors/edit.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}">Dashboard</a></li>
                    <li><a href="{{ url('/admin/authors') }}">Penulis</a></li>
                    <li class="active">Ubah Penulis</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Ubah Penulis</h2>
                    </div>

                    <div class="panel-body">
                        {!! Form::model($author, ['url' => route('authors.update', $author->id),
                            'method'=>'put', 'class'=>'form-horizontal']) !!}
                        @include('authors._form')
                        {!! Form::close() !!}
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

¹⁰<https://laravelcollective.com/docs/5.3/html#form-model-binding>

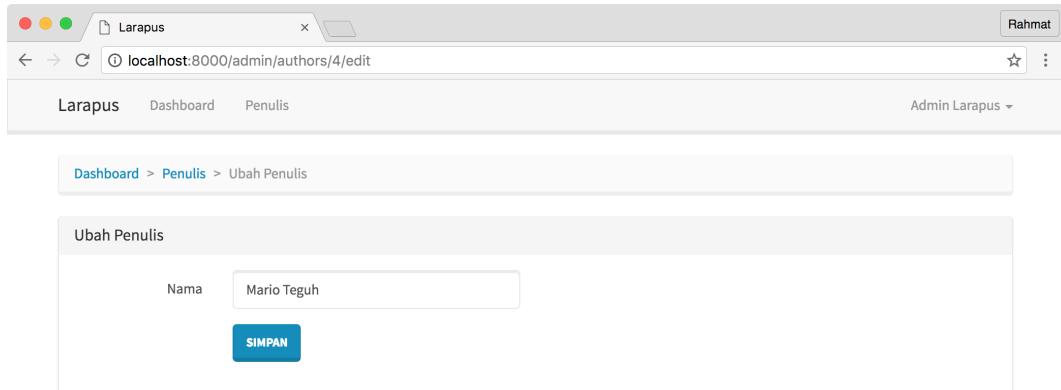
Kita menggunakan partial view yang sama yaitu `authors._form` untuk menampilkan form. Yang berbeda adalah cara kita membuat form, kita menggunakan syntax berikut:

```
Form::model($author, ['url' => route('authors.update', $author->id), 'method'=>'put', 'class' \n=> 'form-horizontal'])
```

Dengan syntax ini, kita menggunakan Form Model Binding dengan model `$author` yang telah kita passing dari controller. URL untuk form ini kita set ke `route('authors.update', $author->id)` (yang akan menjadi `/admin/authors/{id}`) dengan method `PUT`.

Kelebihan dari penggunaan Form Model Binding adalah semua field di form yang memiliki nama field yang sama dengan data di database akan langsung terisi. Dalam kasus kita, meskipun tidak kita set manual, isian field `name` pada form akan memiliki isi.

Berikut tampilan yang akan kita dapatkan ketika mencoba mengubah penulis:



Form Model Binding

Untuk memproses form ini, kita harus menambahkan logic pada method `update` di `AuthorsController`:

app/Http/Controllers/AuthorsController.php

```
public function update(Request $request, $id)
{
    $this->validate($request, ['name' => 'required|unique:authors,name,'. $id]);
    $author = Author::find($id);
    $author->update($request->only('name'));
    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=>"Berhasil menyimpan $author->name"
    ]);

    return redirect()->route('authors.index');
}
```

Pada method ini kita melakukan 4 hal:

- Melakukan validasi terhadap field yang dikirim oleh user. Disini kita memastikan isian name tidak sama dengan yang tercatat pada record di table authors. Kecuali untuk title pada record dengan id yang digunakan pada request.
- Mengupdate record berdasarkan id yang digunakan dengan data di request.
- Set feedback.
- Mengarahkan user kembali ke halaman index.

Cobalah fitur pengubahan data penulis ini. Pastikan berhasil mengubah nama penulis.

4.15 Penghapusan Data Penulis

Bagian terakhir dari proses CRUD untuk Penulis adalah melakukan penghapusan penulis. Pada skema RESTful, untuk menghapus buku kita harus melakukan request ke URL /resources/{id} dengan verb DELETE.

Pada routing yang digenerate oleh Laravel, URL /admin/authors/{id} dengan method DELETE di handle oleh method destroy pada AuthorsController.

Berikut syntax yang akan kita buat:

app/Http/Controllers/AuthorsController.php

```
public function destroy($id)
{
    Author::destroy($id);

    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=>"Penulis berhasil dihapus"
    ]);

    return redirect()->route('authors.index');
}
```

Pada method ini kita menghapus record Author dengan id yang dikirim, set feedback dan mengarahkan user ke halaman index.

Mari kita tambahkan tombol hapus di DataTable. Lakukan perubahan berikut pada method `index` di `AuthorsController`:

app/Http/Controllers/AuthorsController.php

```
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $authors = Author::select(['id', 'name']);
        return Datatables::of($authors)
            ->addColumn('action', function($author){
                return view('datatable._action', [
                    'model'          => $author,
                    'form_url'       => route('authors.destroy', $author->id),
                    'edit_url'       => route('authors.edit', $author->id),
                ]);
            })->make(true);
    }
    ...
}
```

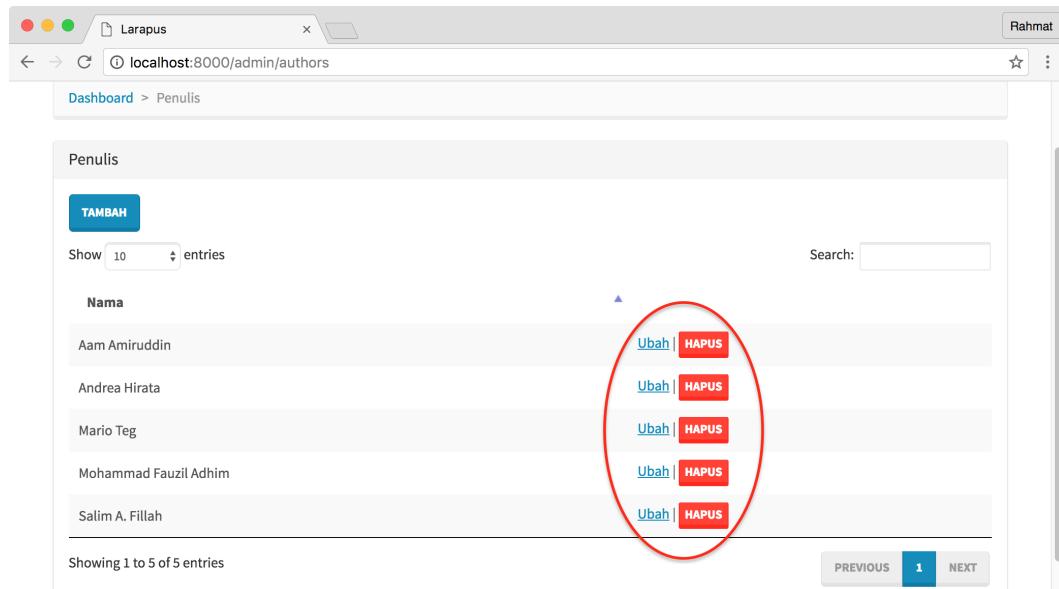
Variable `model` kita passing karena kita akan menggunakan Form Model Binding untuk membuat tombol hapus. Variable `form_url` kita gunakan untuk menentukan URL untuk form yang akan kita buat.

View `datatable._action` kita ubah menjadi:

resources/views/datatables/_action.blade.php

```
<a href="{{ $edit_url }}">Ubah</a>
{!! Form::model($model, ['url' => $form_url, 'method' => 'delete', 'class' => 'form-inline']) !!}
) !!}
<a href="{{ $edit_url }}>Ubah</a> |
{!! Form::submit('Hapus', ['class'=>'btn btn-xs btn-danger']) !!}>
{!! Form::close()!!}
```

Tampilan yang akan kita dapatkan seperti berikut:



The screenshot shows a web browser window titled 'Larapust' with the URL 'localhost:8000/admin/authors'. The page displays a table titled 'Penulis' with five entries: Aam Amiruddin, Andrea Hirata, Mario Teg, Mohammad Fauzil Adhim, and Salim A. Fillah. Each entry has two buttons: 'Ubah' (blue) and 'HAPUS' (red). A red oval highlights the 'HAPUS' buttons for all five entries. At the bottom of the table, there is a message 'Showing 1 to 5 of 5 entries' and navigation buttons for 'PREVIOUS', 'NEXT', and a page number '1'.

Tombol hapus

Kini, cobalah hapus penulis. Akan muncul pesan berikut:

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/authors'. The page header includes 'Rahmat' (logged in user), 'Admin Larapus', and navigation links for 'Larapus', 'Dashboard', and 'Penulis'. A green success message at the top states 'Penulis berhasil dihapus'. Below it, a breadcrumb navigation shows 'Dashboard > Penulis'. The main content area is titled 'Penulis' and contains a table with three rows of data. The table has columns for 'Nama' (Name) and actions. The names listed are 'Aam Amiruddin', 'Andrea Hirata', and 'Mohammad Fauzil Adhim'. Each row has two buttons: 'Ubah' (Edit) and 'HAPUS' (Delete). A search bar is also present above the table. A success message 'Berhasil menghapus penulis' is displayed below the table.

4.16 Penggunaan Model Event

Secara default, ketika kita menghapus seorang penulis, semua data buku untuk penulis tersebut akan dihapus. Kita akan mengubah logic ini, agar Larapus menghentikan proses penghapusan penulis jika masih terdapat buku oleh penulis tersebut.

Untuk membuat fitur ini, kita akan menggunakan fitur Model Event di Laravel. Dengan Model Event, kita dapat menjalankan logic pada berbagai proses CRUD yang dilakukan pada model.

Pada kasus ini, kita akan menggunakan model event dengan nama `deleting`. Event ini akan dieksekusi ketika kita melakukan proses delete pada sebuah model. Jika kita memberikan nilai `false` pada event ini, maka proses penghapusan model akan dibatalkan.

Event lainnya dapat dilihat di [dokumentasi¹¹](#).

Kita buat syntax berikut pada model Author:

¹¹<https://laravel.com/docs/5.3/eloquent#events>

App/Author.php

```
...
use Illuminate\Support\Facades\Session;

class Author extends Model
{
    ...
    public static function boot()
    {
        parent::boot();

        self::deleting(function($author) {
            // mengecek apakah penulis masih punya buku
            if ($author->books->count() > 0) {
                // menyiapkan pesan error
                $html = 'Penulis tidak bisa dihapus karena masih memiliki buku : ';
                $html .= '<ul>';
                foreach ($author->books as $book) {
                    $html .= "<li>$book->title</li>";
                }
                $html .= '</ul>';

                Session::flash("flash_notification", [
                    "level"=>"danger",
                    "message"=>$html
                ]);
            }

            // membatalkan proses penghapusan
            return false;
        });
    }
    ...
}
```

Pada syntax diatas, kita menggunakan method boot untuk melakukan hook ke event deleting. Pada event tersebut, kita mengecek apakah Penulis yang sedang dihapus masih memiliki buku. Jika ya, kita set flash session dengan tipe danger dan pesan berisi detail dari buku yang masih dimiliki oleh penulis tersebut. Terakhir, kita return false agar proses penghapusan dibatalkan.

Kita harus melakukan perubahan pada method destroy di AuthorsController seperti berikut:

app/Http/Controllers/AuthorsController.php

```
public function destroy($id)
{
    Author::destroy($id);
    if(!Author::destroy($id)) return redirect()->back();

    ...
}
```

Pada perubahan ini, ketika method `destroy()` untuk menghapus penulis menghasilkan nilai `false`, kita akan arahkan user ke halaman sebelumnya. Tentunya, akan muncul feedback berdasarkan flash data yang kita set pada event `destroy`.

Berikut tampilan yang akan muncul ketika kita menghapus penulis yang memiliki buku:

The screenshot shows a web browser window titled "Larapus" with the URL "localhost:8000/admin/authors". The page header includes "Rahmat", "Admin Larapus", and navigation links for "Dashboard" and "Penulis". A red alert box displays the message: "Penulis tidak bisa dihapus karena masih memiliki buku :" followed by a bulleted list: "• Membingkai Surga dalam Rumah Tangga" and "• Cinta & Seks Rumah Tangga Muslim". Below the alert box, the breadcrumb navigation shows "Dashboard > Penulis". The main content area is titled "Penulis" and contains a "TAMBAH" button. It also features a "Show 10" dropdown and a "Search:" input field. A table header row is visible with the column name "Nama".

Model Event berhasil

4.17 Konfirmasi ketika Menghapus Data

Dalam sebuah webapp, biasanya ada konfirmasi sebelum kita hendak menghapus sebuah resource. Mari kita coba implementasikan fitur ini untuk melakukan konfirmasi sebelum menghapus Penulis.

Fitur ini akan kita buat dengan javascript. Agar fitur ini dapat digunakan pada proses penghapusan yang lain, kita akan menyimpan scriptnya di file terpisah dan memanggilnya dari resources/views/layouts/app.blade.php.

Mari kita buat script ini di public/js/custom.js

public/js/custom.js

```
$(document).ready(function () {
    // confirm delete
    $(document.body).on('submit', '.js-confirm', function () {
        var $el = $(this)
        var text = $el.data('confirm') ? $el.data('confirm') : 'Anda yakin melakukan tindakan ini\
?'
        var c = confirm(text);
        return c;
    });
});
```

Syntax diatas akan menampilkan dialog confirm untuk setiap form yang memiliki class js-confirm ketika event submit berjalan pada form tersebut. Saat konfirmasi, kita menggunakan pesan konfirmasi “Anda yakin melakukan tindakan ini?” atau menggunakan isian dari attribut data-confirm pada form. Jika user klik “Ok”, maka penghapus dilanjutkan. Jika “Cancel”, maka dibatalkan.

Kita load script ini pada layout utama:

resources/views/layouts/app.blade.php

```
.....
<!-- Scripts -->
.....
<script src="/js/custom.js"></script>
@yield('scripts')
....
```

Untuk menggunakan fitur ini, tambahkan baris berikut pada method index di AuthorsController:

app/Http/Controllers/AuthorsController.php

```

.....
use App\Author;
use Yajra\Datatables\Html\Builder;
use Yajra\Datatables\Datatables;

class AuthorsController extends Controller
{
    public function index(Request $request, Builder $htmlBuilder)
    {
        if ($request->ajax()) {
            $authors = Author::select(['id', 'name']);
            return Datatables::of($authors)
                ->addColumn('action', function($author){
                    return view('datatable._action', [
                        'model'      => $author,
                        'form_url'   => route('authors.destroy', $author->id),
                        'edit_url'   => route('authors.edit', $author->id),
                        'confirm_message' => 'Yakin mau menghapus ' . $author->name . '?'
                    ]);
                })->make(true);
        }
        ....
    }
}
.....

```

Dan ubah view datatable._action menjadi:

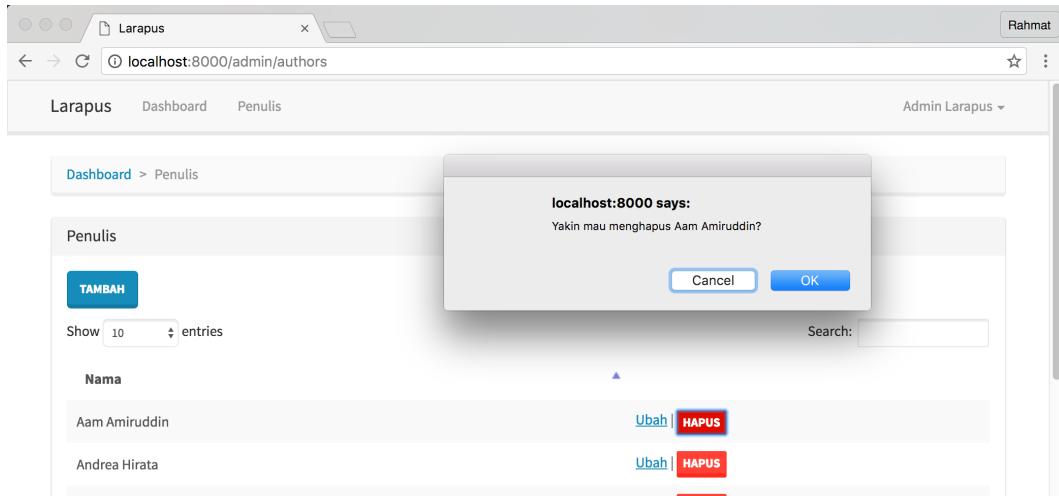
resources/views/datatable/_action.blade.php

```

{!! Form::model($model, ['url' => $form_url, 'method' => 'delete', 'class' => 'form inline']) !!}
{!! Form::model($model, ['url' => $form_url, 'method' => 'delete', 'class' => 'form inline js\-
-confirm', 'data-confirm' => $confirm_message] ) !!}
<a href="{{ $edit_url }}>Ubah</a> |
{!! Form::submit('Hapus', ['class'=>'btn btn-xs btn-danger']) !!}
{!! Form::close() !!}

```

Kini, ketika kita klik pada tombol hapus, akan muncul pesan konfirmasi berikut:



Konfirmasi ketika menghapus penulis

4.18 CRUD Buku

Untuk membuat CRUD buku, kita akan menggunakan banyak syntax yang hampir sama dengan CRUD untuk Penulis.

Pertama, kita siapkan controller:

```
php artisan make:controller BooksController --resource
// Controller created successfully.
```

Dan routing:

routes/web.php

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {
    Route::resource('authors', 'AuthorsController');
    Route::resource('books', 'BooksController');
});
```

4.18.1 Menampilkan daftar buku

Buat link navigasi di `resources/views/layouts/app.blade.php`:

resources/views/layouts/app.blade.php

```
<!--Left Side Of Navbar -->
<ul class="nav navbar-nav">
@if (Auth::check())
    <li><a href="{{ url('/home') }}>Dashboard</a></li>
@endif
@role('admin')
    <li><a href="{{ route('authors.index') }}>Penulis</a></li>
    <li><a href="{{ route('books.index') }}>Buku</a></li>
@endrole
</ul>
```

Kita akan menggunakan DataTable untuk menampilkan field yang ada di table books. Ubah method index pada BooksController:

app/Http/Controllers/BooksController.php

```
.....
use Yajra\Datatables\Html\Builder;
use Yajra\Datatables\Datatables;
use App\Book;

class BooksController extends Controller
{
    public function index(Request $request, Builder $htmlBuilder)
    {
        if ($request->ajax()) {
            $books = Book::with('author');
            return Datatables::of($books)
                ->addColumn('action', function($book){
                    return view('datatable._action', [
                        'model'          => $book,
                        'form_url'       => route('books.destroy', $book->id),
                        'edit_url'       => route('books.edit', $book->id),
                        'confirm_message' => 'Yakin mau menghapus ' . $book->title . '?'
                    ]);
                })
                ->make(true);
        }

        $html = $htmlBuilder
            ->addColumn(['data' => 'title', 'name'=>'title', 'title'=>'Judul'])
            ->addColumn(['data' => 'amount', 'name'=>'amount', 'title'=>'Jumlah'])
            ->addColumn(['data' => 'author.name', 'name'=>'author.name', 'title'=>'Penulis'])
```

```

        ->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, 'se\archable'=>false]);

        return view('books.index')->with(compact('html'));
    }
    ...
}

```

Pada syntax diatas, ada syntax baru yang kita pelajari:

```
$books = Book::with('author');
```

Penggunaan method `with()` akan meload relasi dari Book ke Author dengan teknik eager loading. Dengan teknik ini, jumlah query akan dikurangi sehingga aplikasi kita akan lebih responsif. Lebih detail tentang fitur ini dapat dibaca di [dokumentasi](#)¹².

Menggunakan eager loading, kita dapat mendapatkan nama dari penulis dengan menggunakan key `author.name` pada DataTable:

```
->addColumn(['data' => 'author.name', 'name'=>'author.name', 'title'=>'Penulis'])
```

Selebihnya, syntax pada method ini hampir sama dengan method `index` pada `AuthorsController`.

Kita buat viewnya:

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

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li class="active">Buku</li>
                </ul>
                <div class="panel panel-default">
```

¹²<https://laravel.com/docs/5.3/eloquent-relationships#eager-loading>

```
<div class="panel-heading">
    <h2 class="panel-title">Buku</h2>
</div>

<div class="panel-body">
    <p> <a class="btn btn-primary" href="{{ url('/admin/books/create') }}>Tambah</a> \
</p>
    {!! $html->table(['class'=>'table-striped']) !!}
</div>
</div>
</div>
</div>
</div>
@endsection

@section('scripts')
{!! $html->scripts() !!}
@endsection
```

Tampilan dari halaman ini akan seperti berikut:

Judul	Jumlah	Penulis	
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	Ubah HAPUS
Jalan Cinta Para Pejuang	2	Salim A. Fillah	Ubah HAPUS
Kupinang Engkau dengan Hamdalah	3	Mohammad Fauzil Adhim	Ubah HAPUS
Membingkai Surga dalam Rumah Tangga	4	Aam Amiruddin	Ubah HAPUS

Daftar buku

4.18.2 Menambah Buku

Mari kita lanjut ke fitur untuk menambah buku. Ubah method `create` menjadi:

app/Http/Controllers/BooksController.php

```
public function create()
{
    return view('books.create');
}
```

Dan viewnya:

resources/views/books/create.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li><a href="{{ url('/admin/books') }}>Buku</a></li>
                    <li class="active">Tambah Buku</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Tambah Buku</h2>
                    </div>

                    <div class="panel-body">
                        {!! Form::open(['url' => route('books.store'),
                            'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}
                        @include('books._form')
                        {!! Form::close() !!}
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Pada form untuk buku, kita menambahkan key `files` berisi `true`. Opsi ini akan membuat form yang kita buat dapat mengupload file. Ini kita butuhkan agar dapat mengupload cover buku.

Seperti biasa, field untuk form kita simpan pada partial view:

resources/views/books/_form.blade.php

```
<div class="form-group{{ $errors->has('title') ? ' has-error' : '' }}>
    {!! Form::label('title', 'Judul', ['class'=>'col-md-2 control-label']) !!}
    <div class="col-md-4">
        {!! Form::text('title', null, ['class'=>'form-control']) !!}
        {!! $errors->first('title', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group {{ !$errors->has('author_id') ? 'has-error' : '' !!}}>
    {!! Form::label('author_id', 'Penulis', ['class'=>'col-md-2 control-label']) !!}
    <div class="col-md-4">
        {!! Form::select('author_id', ['=>''+App\Author::pluck('name','id')->all(), null) !!}
        {{$errors->first('author_id', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group{{ $errors->has('amount') ? ' has-error' : '' }}>
    {!! Form::label('amount', 'Jumlah', ['class'=>'col-md-2 control-label']) !!}
    <div class="col-md-4">
        {!! Form::number('amount', null, ['class'=>'form-control', 'min'=>1]) !!}
        {{$errors->first('amount', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group{{ $errors->has('cover') ? ' has-error' : '' }}>
    {!! Form::label('cover', 'Jumlah', ['class'=>'col-md-2 control-label']) !!}
    <div class="col-md-4">
        {!! Form::file('cover') !!}
        {{$errors->first('cover', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group">
    <div class="col-md-4 col-md-offset-2">
        {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
    </div>
</div>
```

Untuk membuat dropdown pemilihan penulis, kita menggunakan syntax berikut:

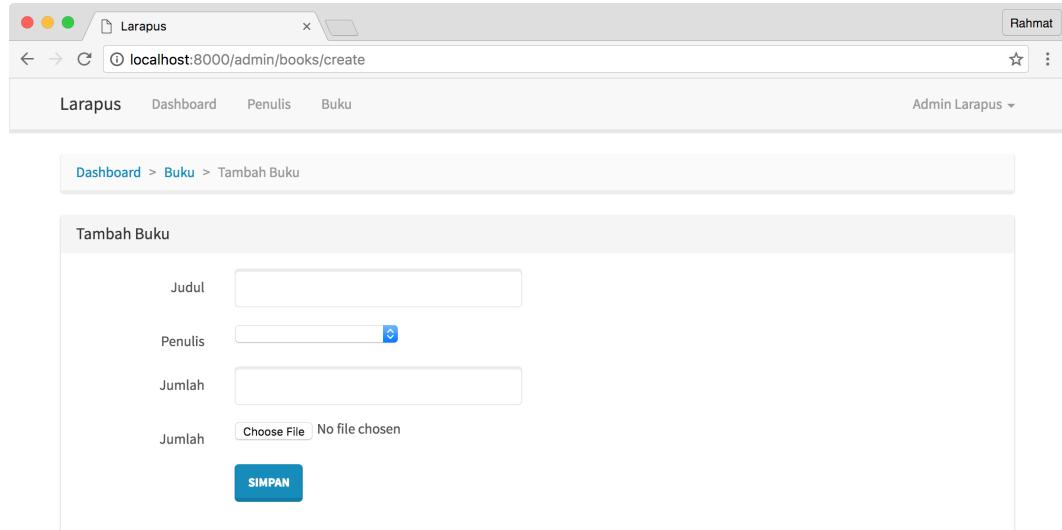
```
{!! Form::select('author_id', ['=>''+App\Author::pluck('name','id')->all(), null) !!}
```

Method `pluck()` pada Eloquent akan menghasilkan array asosiatif dengan parameter pertama sebagai `value` dan parameter kedua sebagai `key`. Pada syntax diatas, kita akan membuat array asosiatif dengan key berisi id dari penulis dan value berisi nama penulis.

Untuk membuat tombol upload, kita menggunakan syntax berikut:

```
{!! Form::file('cover') !!}
```

Tampilan dari halaman ini akan seperti berikut:



Form tambah buku

Untuk menerima form ini, kita harus mengubah isi method `store` di `BooksController`:

app/Http/Controllers/BooksController.php

```
....  
use Illuminate\Support\Facades\Session;  
  
class BooksController extends Controller  
{  
    ....  
    public function store(Request $request)  
    {  
        $this->validate($request, [  
            'title'      => 'required|unique:books,title',  
            'author_id'  => 'required|exists:authors,id',  
            'amount'     => 'required|numeric',  
            'cover'      => 'image|max:2048'  
        ]);  
  
        $book = Book::create($request->except('cover'));  
  
        // isi field cover jika ada cover yang diupload  
        if ($request->hasFile('cover')) {  
            // Mengambil file yang diupload  
            $uploaded_cover = $request->file('cover');  
  
            // mengambil extension file  
            $extension = $uploaded_cover->getClientOriginalExtension();  
  
            // membuat nama file random berikut extension  
            $filename = md5(time()) . '.' . $extension;  
  
            // menyimpan cover ke folder public/img  
            $destinationPath = public_path() . DIRECTORY_SEPARATOR . 'img';  
            $uploaded_cover->move($destinationPath, $filename);  
  
            // mengisi field cover di book dengan filename yang baru dibuat  
            $book->cover = $filename;  
            $book->save();  
        }  
  
        Session::flash("flash_notification", [  
            "level"=>"success",  
            "message"=>"Berhasil menyimpan $book->title"  
        ]);  
  
        return redirect()->route('books.index');  
    }  
}
```

```
    ....  
}
```

Pada syntax diatas, kita melakukan validasi untuk field pada request:

- Field `title` harus diisi dan harus berbeda dengan isian field `title` pada table `books`.
- Field `author_id` harus diisi dan harus terdapat pada table `id` di table `authors`.
- Field `amount` harus diisi dan berupa angka.
- Field `cover`, jika diisi harus berupa image dengan ukuran maksimal 2Mb.

Setelah input dari user kita validasi, kita membuat buku baru:

```
$book = Book::create($request->except('cover'));
```

Disini, kita menggunakan method `except()` untuk mengambil semua input dari user kecuali untuk field `cover`. Ini kita lakukan karena kita akan menggunakan logic terpisah untuk memproses cover buku yang diupload.

Pada syntax selanjutnya, kita akan memproses cover dari buku. Pertama, kita cek apakah ada cover yang diupload:

```
$request->hasFile('cover')
```

Kita menyimpan cover buku pada folder `public/img` dengan nama acak. Logicnya dapat terlihat pada komentar dari setiap source code diatas.

Seperti biasa, kita juga menyiapkan feedback setelah berhasil menambah buku.

Terakhir, kita arahkan user ke halaman index buku.

Berikut tampilan ketika kita berhasil menambah buku:

Berhasil menyimpan Nikmatnya Pacaran setelah Pernikahan

Dashboard > Buku

Judul	Jumlah	Penulis	
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	Ubah HAPUS
Jalan Cinta Para Pejuang	2	Salim A. Fillah	Ubah HAPUS

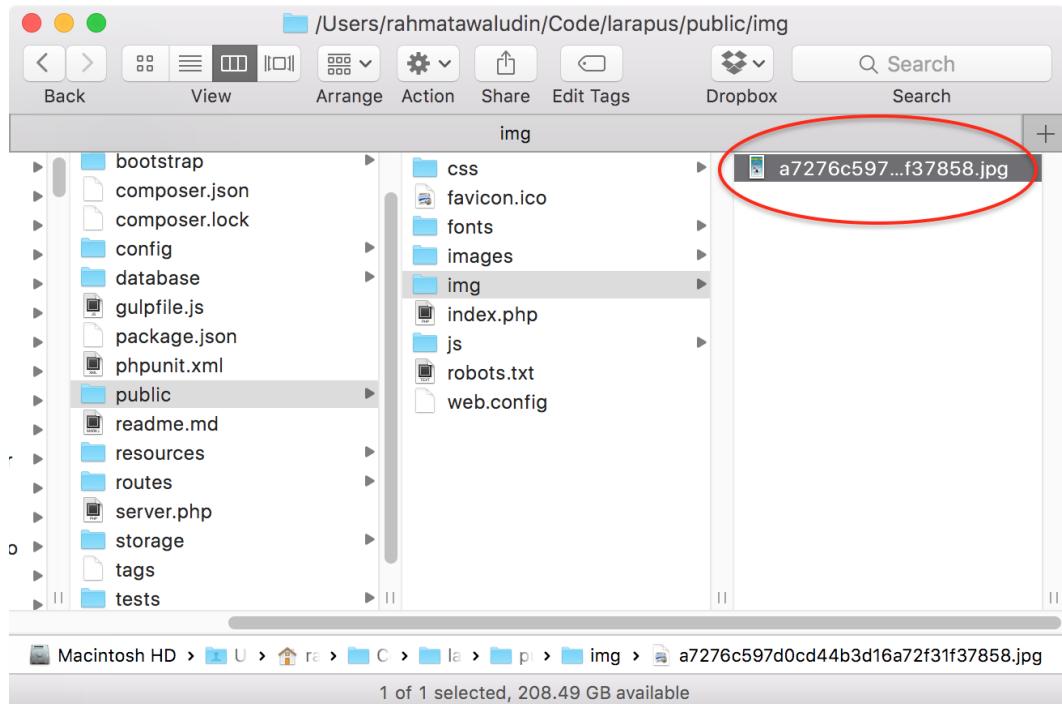
Berhasil menambah buku

Jika kita mengupload cover, isian cover di database akan terisi:

id	title	author_id	amount	cover	created_at	updated_at
1	Kupinang Engkau dengan Hamdalah	1	3	NULL	2016-09-06 02:36:10	2016-09-06 02:36:10
2	Jalan Cinta Para Pejuang	2	2	NULL	2016-09-06 02:36:10	2016-09-06 02:36:10
3	Membingkai Surga dalam Rumah Tangga	3	4	NULL	2016-09-06 02:36:10	2016-09-06 02:36:10
4	Cinta & Seks Rumah Tangga Muslim	3	3	NULL	2016-09-06 02:36:10	2016-09-06 02:36:10
5	Nikmatnya Pacaran setelah Pernikahan	2	4	a7276c597d0cd44b3d16a72f31f37858.jpg	2016-09-06 04:25:59	2016-09-06 04:25:59

Isian cover terisi

Pada folder `public/img` juga akan muncul file baru sesuai nama file di database:



Cover berhasil diupload

4.18.3 Mengubah Buku

Selanjutnya, kita akan melakukan proses untuk mengubah buku. Link untuk mengubah buku sudah kita buat pada tahapan sebelumnya.

Mari kita mulai dengan mengubah method `edit`:

`app/Http/Controllers/BooksController.php`

```
public function edit($id)
{
    $book = Book::find($id);
    return view('books.edit')->with(compact('book'));
}
```

Kita akan menggunakan Form Model Binding sebagaimana yang kita lakukan untuk mengubah Penulis. Itulah alasan kita melakukan passing instance dari model Book yang ditemukan.

Kita buat view nya:

resources/views/books/edit.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}">Dashboard</a></li>
                    <li><a href="{{ url('/admin/books') }}">Buku</a></li>
                    <li class="active">Ubah Buku</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Ubah Buku</h2>
                    </div>

                    <div class="panel-body">
                        {!! Form::model($book, ['url' => route('books.update', $book->id),
                            'method' => 'put', 'files'=>'true', 'class'=>'form-horizontal']) !!}
                        @include('books._form')
                        {!! Form::close() !!}
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Pada field di form buku, kita akan melakukan sedikit perubahan untuk menampilkan cover yang telah diupload (jika ada). Tambahkan syntax berikut pada field cover di resources/views/books/_form.blade.php:

resources/views/books/_form.blade.php

```
....  

<div class="form-group{{ $errors->has('cover') ? ' has-error' : '' }}>  

  {!! Form::label('cover', 'Jumlah', ['class'=>'col-md-2 control-label']) !!}  

  <div class="col-md-4">  

    {!! Form::file('cover') !!}  

    @if ($book) && $book->cover)  

      <p>  

        {!! Html::image(asset('img/'.$book->cover), null, ['class'=>'img-rounded img-responsive']) !!}  

    @endif  

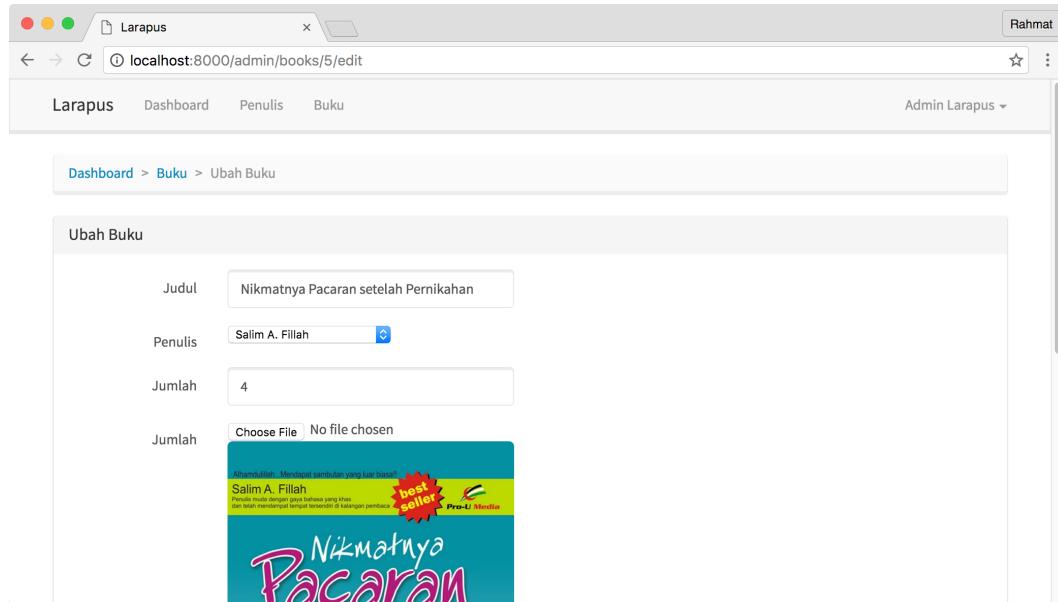
    {!! $errors->first('cover', '<p class="help-block">:message</p>') !!}  

  </div>  

</div>  

....
```

Tampilan halaman edit untuk buku yang telah memiliki cover akan seperti ini:



Edit Buku

Untuk menerima form ini, kita ubah method update menjadi:

app/Http/Controllers/BooksController.php

```
....  
use Illuminate\Support\Facades\File;  
  
class BooksController extends Controller  
{  
    ....  
    public function update(Request $request, $id)  
    {  
        $this->validate($request, [  
            'title'      => 'required|unique:books,title,' . $id,  
            'author_id'  => 'required|exists:authors,id',  
            'amount'     => 'required|numeric',  
            'cover'      => 'image|max:2048'  
        ]);  
  
        $book = Book::find($id);  
        $book->update($request->all());  
  
        if ($request->hasFile('cover')) {  
            // menambil cover yang diupload berikut ekstensinya  
            $filename = null;  
            $uploaded_cover = $request->file('cover');  
            $extension = $uploaded_cover->getClientOriginalExtension();  
  
            // membuat nama file random dengan extension  
            $filename = md5(time()) . '.' . $extension;  
            $destinationPath = public_path() . DIRECTORY_SEPARATOR . 'img';  
  
            // memindahkan file ke folder public/img  
            $uploaded_cover->move($destinationPath, $filename);  
  
            // hapus cover lama, jika ada  
            if ($book->cover) {  
                $old_cover = $book->cover;  
                $filepath = public_path() . DIRECTORY_SEPARATOR . 'img'  
                    . DIRECTORY_SEPARATOR . $book->cover;  
  
                try {  
                    File::delete($filepath);  
                } catch (FileNotFoundException $e) {  
                    // File sudah dihapus/tidak ada  
                }  
            }  
  
            // ganti field cover dengan cover yang baru
```

```
$book->cover = $filename;
$book->save();
}

Session::flash("flash_notification", [
    "level"=>"success",
    "message"=>"Berhasil menyimpan $book->title"
]);

return redirect()->route('books.index');
}
...
}
```

Pada method ini, syntax untuk melakukan validasinya hampir sama. Kita melakukan sedikit perubahan pada validasi untuk field `title` agar mengizinkan menggunakan `title` yang sama dengan yang sekarang digunakan.

Untuk menghandle upload cover, kita melakukan proses upload seperti biasa. Jika sebelumnya field `cover` telah terisi, kita hapus `cover` yang lama. Detail dari logic untuk melakukan ini dapat dilihat pada komentar di source code diatas.

Terakhir kita set feedback dan mengarahkan user ke halaman index buku.

Cobalah untuk melakukan proses update, pastikan semua fiturnya berjalan.

4.18.4 Menghapus Buku

Proses penghapusan akan cukup mudah kita lakukan. Kita telah membuat tombol `hapus` pada tahapan sebelumnya. Yang perlu kita lakukan hanya menambahkan logic di method `destroy` di `BooksController`:

app/Http/Controllers/BooksController.php

```
public function destroy($id)
{
    $book = Book::find($id);

    // hapus cover lama, jika ada
    if ($book->cover) {
        $old_cover = $book->cover;
        $filepath = public_path() . DIRECTORY_SEPARATOR . 'img'
            . DIRECTORY_SEPARATOR . $book->cover;

        try {
            File::delete($filepath);
        } catch (FileNotFoundException $e) {
            // File sudah dihapus/tidak ada
        }
    }

    $book->delete();

    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=> "Buku berhasil dihapus"
    ]);

    return redirect()->route('books.index');
}
```

Pada syntax diatas, kita mencari buku, menghapus file cover (jika ada), set feedback dan mengarahkan user ke halaman index buku. Silahkan cek fitur hapus buku, pastikan semua logic diatas berjalan dengan benar.

The screenshot shows a browser window titled 'Larapus' at the URL 'localhost:8000/admin/books'. The page is titled 'Buku' and displays a list of books with columns for Judul, Jumlah, and Penulis. Each row has 'Ubah' and 'HAPUS' buttons. A green success message at the top says 'Buku berhasil dihapus'. Below the table, a message says 'Berhasil menghapus buku'.

Judul	Jumlah	Penulis
Jalan Cinta Para Pejuang	2	Salim A. Fillah
Kupinang Engkau dengan Hamdalah	3	Mohammad Fauzil Adhim
Membingkai Surga dalam Rumah Tangga	4	Aam Amiruddin



Latihan

Pada syntax diatas, terlihat kita menggunakan syntax yang sama untuk menghapus cover buku pada method update dan destroy. Cobalah refactor syntax diatas pada method terpisah agar tidak terjadi pengulangan code. Kita dapat menyimpan logicnya pada BooksController atau di model App\Book.

4.19 Penggunaan Form Request

Pada dua CRUD diatas kita melakukan validasi yang hampir sama di method store dan update. Untuk jumlah validasi yang sedikit, penggunaan validasi di controller tidak terlalu bermasalah. Tapi, untuk jumlah field yang banyak, misalnya untuk CRUD Buku, akan lebih baik jika kita pisahkan proses validasi ini pada tempat yang terpisah.

Di Laravel, untuk memisahkan validasi dari controller kita dapat menggunakan fitur **Form Request**¹³.

Mari kita praktikan dengan menggunakan Form Request untuk melakukan validasi pada BooksController. Kita buat dulu Form Request untuk method store.

Jalankan perintah berikut:

```
php artisan make:request StoreBookRequest  
// Request created successfully.
```

Akan muncul file baru di app/Http/Requests/StoreBookRequest.php. Pada file tersebut akan terdapat dua method:

- `authorize` digunakan untuk menentukan akses user
- `rules` digunakan untuk menentukan aturan validasi

Isi method `authorize` dengan:

app/Http/Requests/StoreBookRequest.php

```
....  
use Illuminate\Support\Facades\Auth;  
  
class StoreBookRequest extends Request  
{  
    public function authorize()  
    {  
        return Auth::check();  
    }  
    ....  
}
```

Syntax diatas akan memastikan hanya user yang telah login yang dapat mengakses Form Request ini. Tentunya, pada aplikasi yang sudah besar akan terdapat banyak logic disini. Selain sudah login, seharusnya kita juga mengecek apakah user memiliki akses sebagai admin, tapi hal itu sudah kita lakukan pada routing. Sehingga, tidak perlu kita buat lagi disini.

¹³<https://laravel.com/docs/5.3/validation#form-request-validation>



Sebenarnya, kita isi dengan `return true;` juga bisa. Sengaja saya isi dengan syntax ini untuk menunjukkan bagaimana method ini harus diisi.

Pada method rules, kita isi dengan aturan validasi yang kini ada di method store:

app/Http/Requests/StoreBookRequest.php

```
public function rules()
{
    return [
        'title'      => 'required|unique:books,title',
        'author_id'  => 'required|exists:authors,id',
        'amount'     => 'numeric',
        'cover'      => 'image|max:2048'
    ];
}
```

Untuk menggunakan Form Request ini di method store, lakukan perubahan berikut:

app/Http/Controllers/BooksController.php

```
...
use App\Http\Requests\StoreBookRequest;

class BooksController extends Controller
{
    ...
    public function store(Request $request)
    public function store(StoreBookRequest $request)
    {
        $this->validate($request, [
            'title'      => 'required|unique:books,title',
            'author_id'  => 'required|exists:authors,id',
            'amount'     => 'required|numeric',
            'cover'      => 'image|max:2048'
        ]);
        ...
    }
}
```

Cobalah menambah buku, pastikan validasi masih berjalan.

Selanjutnya, mari kita ubah juga validasi untuk method update. Rules untuk validasi untuk method ini hampir sama dengan method store. Kita hanya perlu menyesuaikan rule untuk field title. Untuk itu, kita akan meng-*extends* StoreBookRequest dan melakukan perubahan pada method rules.

Buatlah file app/Http/Requests/UpdateBookRequest.php, isi dengan syntax berikut:

app/Http/Requests/UpdateBookRequest.php

```
<?php

namespace App\Http\Requests;

class UpdateBookRequest extends StoreBookRequest
{
    public function rules()
    {
        $rules = parent::rules();
        $rules['title'] = 'required|unique:books,title,' . $this->route('book');
        return $rules;
    }
}
```

Pada method ini, kita mengambil semua rules yang dimiliki StoreBookRequest dan melakukan perubahan pada rules untuk field title.

Pada form request, untuk mendapatkan id yang dikirim kita harus menggunakan nama parameter yang kita gunakan pada routing. Jika menggunakan resource routing, secara default nama parameter ini akan book. Kita dapat melihatnya pada output php artisan route:list:

2. rahmatawaludin@MalesCast: ~/Code/larapust (cat)						
Domain	Method	URI	Name	Action	Middleware	
	GET HEAD	/		Closure	web	
	GET HEAD	admin/authors	authors.index	App\Http\Controllers\AuthorsController@index	web,auth,role:admin	
	POST	admin/authors	authors.store	App\Http\Controllers\AuthorsController@store	web,auth,role:admin	
	GET HEAD	admin/authors/create	authors.create	App\Http\Controllers\AuthorsController@create	web,auth,role:admin	
	GET HEAD	admin/authors/{author}	authors.show	App\Http\Controllers\AuthorsController@show	web,auth,role:admin	
	DELETE	admin/authors/{author}	authors.destroy	App\Http\Controllers\AuthorsController@destroy	web,auth,role:admin	
	PUT PATCH	admin/authors/{author}	authors.update	App\Http\Controllers\AuthorsController@update	web,auth,role:admin	
	GET HEAD	admin/authors/{author}/edit	authors.edit	App\Http\Controllers\AuthorsController@edit	web,auth,role:admin	
	POST	admin/books	books.store	App\Http\Controllers\BooksController@store	web,auth,role:admin	
	GET HEAD	admin/books	books.index	App\Http\Controllers\BooksController@index	web,auth,role:admin	
	GET HEAD	admin/books/create	books.create	App\Http\Controllers\BooksController@create	web,auth,role:admin	
	DELETE	admin/books/{book}	books.destroy	App\Http\Controllers\BooksController@destroy	web,auth,role:admin	
	PUT PATCH	admin/book/{book}	books.update	App\Http\Controllers\BooksController@update	web,auth,role:admin	
	GET HEAD	admin/books/{book}	books.show	App\Http\Controllers\BooksController@show	web,auth,role:admin	
	GET HEAD	admin/books/{book}/edit	books.edit	App\Http\Controllers\BooksController@edit	web,auth,role:admin	
	GET HEAD	api/user		Closure	api,auth:api	
	GET HEAD	home		App\Http\Controllers\HomeController@index	web,auth	

Nama parameter

Itulah sebabnya kita menggunakan `$this->route('book')`, tidak menggunakan `$id` sebagaimana method update.

Untuk menggunakan form request ini, lakukan perubahan berikut:

app/Http/Controllers/BooksController.php

```

use App\Http\Requests\UpdateBookRequest;

class BooksController extends Controller
{
    ...
    public function update(Request $request, $id)
    public function update(UpdateBookRequest $request, $id)
    {
        $this->validate($request, [
            'title'      => 'required|unique:books,title,' . $id,
            'author_id'  => 'required|exists:authors,id',
            'amount'     => 'required|numeric',
            'cover'      => 'image|max:2048'
        ]);
        ...
    }
}

```

Cobalah melakukan perubahan data buku, pastikan semua validasi berfungsi.



Latihan

Menggunakan form request akan bermanfaat untuk aplikasi yang sudah besar. Cobalah ubah validasi untuk AuthorsController agar menggunakan form request.

4.20 Penggunaan Selectize

Ini tampilan yang kita miliki untuk memilih penulis ketika menambah/mengubah buku:

Larapus Dashboard Penulis Buku

Dashboard > Buku > Ubah Buku

Ubah Buku

Judul	<input type="text"/>
Penulis	<input type="text"/> (dropdown menu showing: Mohammad Fauzil Adhim, ✓ Salim A. Fillah, Aam Amiruddin, Andrea Hirata)
Jumlah	<input type="text" value="2"/>
Jumlah	<input type="file"/> Choose File No file chosen

SIMPAN

Pilih penulis

Tampilan seperti ini, meskipun berfungsi, akan menyulitkan user untuk memilih penulis ketika jumlah penulis sudah banyak.

Kita akan menggunakan library javascript bernama [Selectize¹⁴](#) untuk memudahkan kita dalam memilih penulis. Salinlah file berikut dari sample source code:

¹⁴<http://selectize.github.io/selectize.js>

- public/js/selectize.min.js
- public/css/selectize.css
- public/css/selectize.bootstrap3.css

Load asset diatas pada file resources/views/layouts/app.blade.php:

resources/views/layouts/app.blade.php

```
....  
<link href="/css/dataTables.bootstrap.css" rel="stylesheet">  
<link href="/css/selectize.css" rel="stylesheet">  
<link href="/css/selectize.bootstrap3.css" rel="stylesheet">  
....  
<script src="/js/selectize.min.js"></script>  
<script src="/js/custom.js"></script>  
@yield('scripts')  
<script src="{{ asset('js/app.js') }}></script>  
....
```



Jika Anda mendownload manual selectize dari github, pastikan memilih versi “standalone”.

Untuk menggunakan library ini, kita akan menggunakan teknik yang sama dengan javascript untuk konfirmasi yaitu mengaktifkan library dengan menambahkan class pada element yang hendak diaktifkan. Tambahkan baris berikut pada public/js/custom.js:

public/js/custom.js

```
$(document).ready(function () {  
    ....  
    // add selectize to select element  
    $('.js-selectize').selectize({  
        sortField: 'text'  
    });  
});
```

Untuk mengaktifkan Selectize, kita cukup menambahkan class js-selectize pada element <select> yang kita inginkan.

Mari kita lakukan pada <select> untuk memilih penulis:

resources/views/books/_form.blade.php

```
....  
{!! Form::select('author_id', [ ''=> '' ]+App\Author::pluck('name','id')->all(), null) !!}  
{!! Form::select('author_id', [ ''=> '' ]+App\Author::pluck('name','id')->all(), null, [  
    'class'=>'js-selectize',  
    'placeholder' => 'Pilih Penulis']) !!}  
....
```

Selain menambahkan class `js-selectize`, pada syntax diatas kita juga menambahkan placeholder “Pilih Penulis” yang akan kita tampilkan ketika belum ada penulis yang dipilih.

Tampilan form buku kini akan seperti berikut:

Larapus Dashboard Penulis Buku

Dashboard > Buku > Ubah Buku

Ubah Buku

Judul	Jalan Cinta Para Pejuang
Penulis	Salim A. Fillah
Jumlah	Aam Amiruddin Andrea Hirata Mohammad Fauzil Adhim Salim A. Fillah
Jumlah	

SIMPAN

Berhasil menambahkan selectize

Salah satu kelebihan Selectize adalah kita dapat mencari penulis lebih mudah, seperti ini:

Ubah Buku

Judul: Jalan Cinta Para Pejuang

Penulis: am

Aam Amiruddin
Mohammad Fauzil Adhim

Jumlah:

Choose File No file chosen

SIMPAN

Mencari penulis

4.21 Ringkasan

Di hari 4, kita telah mempelajari bagaimana melakukan pembatasan akses berdasarkan role. Kita juga telah membuat CRUD untuk penulis dan buku dari halaman Admin. Poin-poin yang telah kita bahas yaitu:

- Konfigurasi dan authorisasi aplikasi
- Penggunaan halaman error
- Penggunaan datatable untuk menampilkan data
- Penggunaan Flash Message
- Mass assignment pada model
- Penggunaan macro dalam blade
- Penggunaan relasi pada Eloquent
- Penggunaan model event pada Eloquent

Pada hari 5, kita akan memulai implementasi beberapa fitur non-admin dan fitur stok buku. Spirit! :)

5. Hari 5 : Develop Fitur Non-Admin

Dalam membangun aplikasi yang memiliki berbagai hak akses, sebenarnya cukup sulit untuk memisahkan pengembangan untuk fitur seorang admin maupun non-admin, karena keduanya pasti saling berkaitan. Contohnya, dalam aplikasi ini, jika kita sedang mengembangkan fitur peminjaman akan berkaitan dengan penghapusan buku oleh Admin. Jika buku sedang dipinjam, tentunya Admin tidak bisa menghapus buku tersebut.

Fitur lain yang saling berkaitan adalah fitur peminjaman dan fitur update jumlah total buku (`amount`). Fitur stok ini akan mempengaruhi alur ketika admin merubah jumlah total buku (`amount`), jika buku yang sedang dipinjam ada 3, maka admin tidak boleh mengubah jumlah buku (`amount`) kurang dari 3.

Untuk beberapa fitur yang saling berkaitan seperti itu, semoga Anda tidak akan protes kalau saya bahas disini.. :)

5.1 Persiapan Database dan Model untuk Peminjaman

Di Larapus, alur untuk melakukan peminjaman buku diawali dengan user login, pilih menu buku, kemudian memilih buku yang akan dipinjam, dan User langsung tercatat meminjam buku tersebut. Dalam melakukan peminjaman, kita akan mencatat pada table `borrow_logs`.

Mari kita buat model dan migrationnya:

```
php artisan make:model BorrowLog -m
// Model created successfully.
// Created Migration: xxxx_xx_xx_xxxxxx_create_borrow_logs_table
```

Pada file migration yang dihasilkan, isi method up dengan:

database/migrations/yyyy_xx_xx_xxxxxx_create_borrow_logs_table.php

```
public function up()
{
    Schema::create('borrow_logs', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('book_id')->unsigned()->index();
        $table->foreign('book_id')->references('id')->on('books')
            ->onDelete('cascade')
            ->onUpdate('cascade');
        $table->integer('user_id')->unsigned()->index();
        $table->foreign('user_id')->references('id')->on('users')
            ->onDelete('cascade')
            ->onUpdate('cascade');
        $table->boolean('is_returned')->default(false);
        $table->timestamps();
    });
}
```

Pada migration ini, kita menambahkan beberapa field:

- **id**: kita membutuhkan field ini agar dapat menggunakan record yang berbeda ketika user meminjam buku yang sama setelah dikembalikan.
- **book_id**: untuk mencatat id dari buku yang dipinjam.
- **user_id**: untuk mencatat id dari user yang meminjam buku.
- **is_returned**: field ini digunakan untuk mencatat apakah buku sudah dikembalikan atau belum.
- **timestamps**: field timestamp bawaan Laravel ini akan kita gunakan untuk mencatat tanggal peminjaman dan pengembalian buku. Field `created_at` akan kita gunakan sebagai tanggal peminjaman dan field `updated_at` akan kita gunakan sebagai tanggal pengembalian.

Kita juga menambah foreign key untuk field `book_id` dan `user_id`.

Pada model `BorrowLog`, kita aktifkan mass assignment:

app/BorrowLog.php

```
protected $fillable = ['book_id', 'user_id', 'is_returned'];
```

Kita tambahkan juga relasi ke model Book dan User:

app/BorrowLog.php

```
public function book()
{
    return $this->belongsTo('App\Book');
}

public function user()
{
    return $this->belongsTo('App\User');
}
```

5.2 Membuat Sample Peminjaman

Untuk memudahkan mengembangkan fitur, kita akan membuat sample data peminjaman. Tambahkan baris berikut pada BooksSeeder:

database/seeds/BooksSeeder.php

```
...
use App\BorrowLog;
use App\User;

class BooksSeeder extends Seeder
{
    public function run()
    {
        ...
        // Sample peminjaman buku
        $member = User::where('email', 'member@gmail.com')->first();
        BorrowLog::create(['user_id' => $member->id, 'book_id'=>$book1->id, 'is_returned' => 0]);
        BorrowLog::create(['user_id' => $member->id, 'book_id'=>$book2->id, 'is_returned' => 0]);
        BorrowLog::create(['user_id' => $member->id, 'book_id'=>$book3->id, 'is_returned' => 1]);
    }
}
```

Pada syntax diatas, method `first()` berguna untuk mengambil record pertama dari query yang dilakukan. Disini, kita ingin mencari seorang member dengan query ke field `email` berisi `member@gmail.com`.

Kita refresh database untuk mendapatkan struktur table dan sample data ini:

```
php artisan migrate:refresh --seed
```

5.3 Pembuatan Fitur Peminjaman

Disini, kita akan menampilkan tombol “pinjam”. Tentunya, fitur peminjaman ini hanya diizinkan untuk user yang memiliki akses member. Mari kita buat daftar buku ini dengan menggunakan DataTable.

Pertama, kita buat agar halaman utama di handle oleh controller tersendiri bernama `GuestController`. Kita buat dulu controlernya:

```
php artisan make:controller GuestController
```

Pada file routing, kita lakukan perubahan berikut:

`routes/web.php`

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

Pada `GuestController`, kita buat syntax berikut:

app/Http/Controllers/GuestController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use Yajra\Datatables\Html\Builder;
use Yajra\Datatables\Datatables;
use App\Book;
use Laratrust\LaratrustFacade as Laratrust;

class GuestController extends Controller
{
    public function index(Request $request, Builder $htmlBuilder)
    {
        if ($request->ajax()) {
            $books = Book::with('author');
            return Datatables::of($books)
                ->addColumn('action', function($book){
                    if (Laratrust::hasRole('admin')) return '';
                    return '<a class="btn btn-xs btn-primary" href="#">Pinjam</a>';
                })->make(true);
        }

        $html = $htmlBuilder
            ->addColumn(['data' => 'title', 'name'=>'title', 'title'=>'Judul'])
            ->addColumn(['data' => 'author.name', 'name'=>'author.name', 'title'=>'Penulis'])
            ->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, 'se\archable'=>false]);

        return view('guest.index')->with(compact('html'));
    }
}
```

Pada controller ini, kita menyiapkan DataTable yang akan menampilkan field judul, penulis dan action yang berisi tombol untuk meminjam buku. Tombol untuk meminjam buku ini hanya akan tampil jika user bukan seorang admin. Terlihat pada syntax berikut:

```
if (Laratrust::hasRole('admin')) return '';
```

Link untuk meminjam buku akan kita bahas pada pembahasan selanjutnya.

Mari kita buat viewnya di guest.index:

resources/views/guest/index.blade.php

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

@section('content')


## Daftar Buku



{!! $html->table(['class'=>'table-striped']) !!}


```

Kini, tampilan halaman utama akan seperti berikut:

Judul	Penulis	
Cinta & Seks Rumah Tangga Muslim	Aam Amiruddin	PINJAM
Jalan Cinta Para Pejuang	Salim A. Fillah	PINJAM
Kupinang Engkau dengan Hamdalah	Mohammad Fauzil Adhim	PINJAM
Membingkai Surga dalam Rumah Tangga	Aam Amiruddin	PINJAM

Showing 1 to 4 of 4 entries

Halaman Utama Larapus

Untuk melakukan peminjaman, kita akan membuat GET request ke `/books/{id}/borrow`. URL ini, akan di handle oleh method `borrow` pada `BooksController`. Sebagai catatan, pada skema REST sebaiknya kita tidak menggunakan method GET untuk membuat record baru di database. Pada kasus ini, kita telah melanggar aturan tersebut. Ini kita lakukan agar kita dapat melakukan peminjaman sebagai user yang belum login (karena Laravel tidak dapat melakukan redirect untuk method POST).

Kita buat dulu routingnya:

routes/web.php

```
Route::get('books/{book}/borrow', [
    'middleware' => ['auth', 'role:member'],
    'as'         => 'guest.books.borrow',
    'uses'       => 'BooksController@borrow'
]);
```

Teknik membuat routing yang kita lakukan diatas dinamakan *named routing*. Dengan teknik ini, kita menggunakan parameter `as` untuk memberikan nama pada routing yang kita buat sebagai `guest.books.borrow`. Isian `uses` menentukan method dan controller apa yang akan kita gunakan pada route tersebut.

Kita juga menambahkan middleware auth dan role:member agar route ini hanya bisa diakses oleh User yang sudah login dan memiliki akses member.

Kita buat method borrow pada BooksController:

app/Http/Controllers/BooksController.php

```
....  
use Illuminate\Database\Eloquent\ModelNotFoundException;  
use App\BorrowLog;  
use Illuminate\Support\Facades\Auth;  
  
class BooksController extends Controller  
{  
    ....  
    public function borrow($id)  
    {  
        try {  
            $book = Book::findOrFail($id);  
            BorrowLog::create([  
                'user_id' => Auth::user()->id,  
                'book_id' => $id  
            ]);  
            Session::flash("flash_notification", [  
                "level"=>"success",  
                "message"=>"Berhasil meminjam $book->title"  
            ]);  
        } catch (ModelNotFoundException $e) {  
            Session::flash("flash_notification", [  
                "level"=>"danger",  
                "message"=>"Buku tidak ditemukan."  
            ]);  
        }  
  
        return redirect('/');  
    }  
}
```

Oke, syntax ini cukup panjang. Mari kita bahas:

- Kita menggunakan `findOrFail` untuk memastikan id buku yang digunakan valid. Method ini akan “melemparkan” (*throw*) exception bernama `ModelNotFoundException` ketika model tidak ditemukan. Itulah sebabnya kita menggunakan statement `try-catch`.

- Kita membuat peminjaman baru dengan method BorrowLog::create(). Kita menggunakan Auth::user()->id untuk mendapatkan id dari user yang sedang login. Untuk isian is_returned, akan otomatis terisi menjadi 0.
- Kita membuat feedback sukses setelah berhasil membuat record.
- Kita membuat feedback error ketika ModelNotFoundException kita dapatkan.
- Terakhir, kita arahkan user kembali ke halaman utama.

Untuk menggunakan URL ini, kita ubah link pada tombol untuk melakukan peminjaman:

app/Http/Controllers/GuestController.php

```
....  
return '<a class="btn btn-xs btn-primary" href="#">Pinjam</a>';  
return '<a class="btn btn-xs btn-primary" href="'.route('guest.books.borrow', $book->id).'">P\\injam</a>';  
....
```

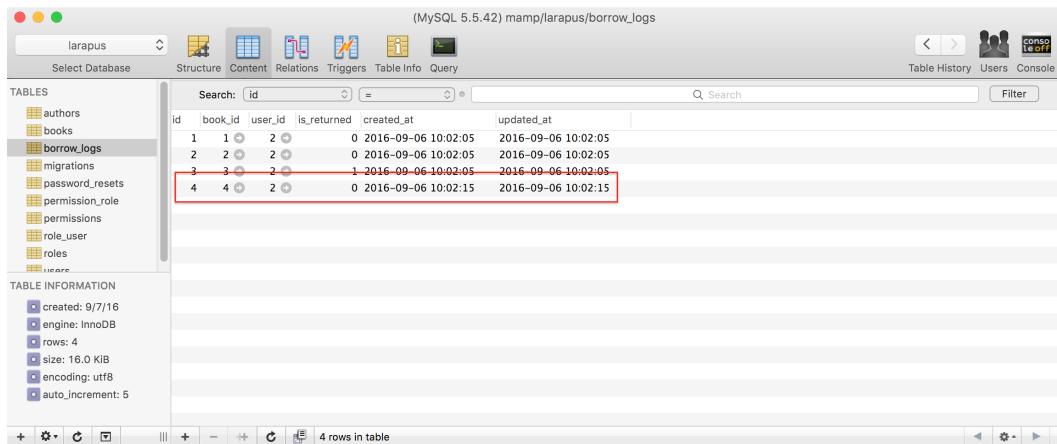
Kini, cobalah login sebagai member dan pinjam sebuah buku, ini tampilan yang akan kita dapatkan:

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000'. The page displays a success message 'Berhasil meminjam Cinta & Seks Rumah Tangga Muslim' in a green bar. Below it is a table titled 'Daftar Buku' showing two books: 'Cinta & Seks Rumah Tangga Muslim' by 'Aam Amiruddin' and 'Jalan Cinta Para Pejuang' by 'Salim A. Fillah'. Each book row has a blue 'PINJAM' button. The browser interface includes a top navigation bar with 'Rahmat', 'Sample Member', and a dropdown menu, and a bottom status bar.

Judul	Penulis	
Cinta & Seks Rumah Tangga Muslim	Aam Amiruddin	PINJAM
Jalan Cinta Para Pejuang	Salim A. Fillah	PINJAM

Berhasil meminjam buku

Pada table borrow_logs akan muncul record baru:



The screenshot shows the MySQL Workbench interface for the 'borrow_logs' database. The 'borrow_logs' table is selected, displaying the following data:

id	book_id	user_id	is_returned	created_at	updated_at
1	1	2	0	2016-09-06 10:02:05	2016-09-06 10:02:05
2	2	2	0	2016-09-06 10:02:05	2016-09-06 10:02:05
3	3	2	1	2016-09-06 10:02:05	2016-09-06 10:02:05
4	4	2	0	2016-09-06 10:02:15	2016-09-06 10:02:15

Record borrow_logs baru dibuat

Jika kita meminjam sebagai user yang belum login, kita akan otomatis diarahkan ke halaman login dan, jika login sebagai member, akan berhasil meminjam buku.

Kita juga akan mendapatkan halaman “No access” ketika mencoba meminjam buku sebagai seorang Admin.

5.4 Membatasi Jumlah Buku yang Dipinjam

Pada Larapus, kita akan membatasi agar member hanya dapat meminjam 1 buku dalam satu waktu untuk judul yang sama. Untuk membuat fitur ini, kita akan *throw* exception baru bernama `BookException`. Sehingga, kita dapat *catch* exception ini dari controller dan set pesan error.

Kita buat dulu file baru di `app/Exceptions/BookException.php` dengan isi:

app/Exceptions/BookException.php

```
<?php

namespace App\Exceptions;

use Exception;

class BookException extends Exception {}
```

Untuk melakukan peminjaman buku, kita akan membuat method borrow pada model User seperti berikut:

app/User.php

```
.....
use App\Book;
use App\BorrowLog;
use App\Exceptions\BookException;

class User extends Authenticatable
{
    .....
    public function borrow(Book $book)
    {
        // cek apakah buku ini sedang dipinjam oleh user
        if($this->borrowLogs()->where('book_id',$book->id)->where('is_returned', 0)->count() > 0 \ 
    ) {
        throw new BookException("Buku $book->title sedang Anda pinjam.");
    }

    $borrowLog = BorrowLog::create(['user_id'=>$this->id, 'book_id'=>$book->id]);
    return $borrowLog;
}

public function borrowLogs()
{
    return $this->hasMany('App\BorrowLog');
}
```

Pada syntax diatas, selain membuat method borrow, kita juga membuat method borrowLogs yang akan kita gunakan untuk mengambil semua data peminjaman oleh user tersebut.

Method borrow menerima parameter berupa instance dari App\Book. Pada method ini, kita mengecek apakah ada record BorrowLog untuk user dan buku tersebut yang isian `is_returned`-nya 0. Jika ya, kita *throw* BookException dengan pesan yang sesuai.

Terakhir, ketika pengecekan diatas sudah dilewati, kita membuat record BorrowLog dan mengembalikan model BorrowLog yang baru dibuat.

Pada method borrow di `app/Http/Controllers/BooksController.php` kita ubah menjadi:

app/Http/Controllers/BooksController.php

```
....  
use App\Exceptions\BookException;  
  
class BooksController extends Controller  
{  
    ....  
    public function borrow($id)  
    {  
        try {  
            $book = Book::findOrFail($id);  
            BorrowLog::create([  
                'user_id' => Auth::user()->id,  
                'book_id' => $id  
            ]);  
            Auth::user()->borrow($book);  
            Session::flash("flash_notification", [  
                "level"=>"success",  
                "message"=>"Berhasil meminjam $book->title"  
            ]);  
        } catch (BookException $e) {  
            Session::flash("flash_notification", [  
                "level"    => "danger",  
                "message" => $e->getMessage()  
            ]);  
        } catch (ModelNotFoundException $e) {  
            Session::flash("flash_notification", [  
                "level"    => "danger",  
                "message" => "Buku tidak ditemukan."  
            ]);  
        }  
  
        return redirect('/');
```

```
}
```

Pada syntax diatas, kita merubah syntax untuk merubah buku dan menambahkan statement catch untuk BookException. Ketika kita mencoba meminjam buku yang telah kita pinjam, ini error yang akan kita dapatkan:

Judul	Penulis	
Cinta & Seks Rumah Tangga Muslim	Aam Amiruddin	<button>PINJAM</button>
Jalan Cinta Para Pejuang	Salim A. Fillah	<button>PINJAM</button>

Berhasil membatasi jumlah peminjaman

5.5 Penggunaan Attribute Casting pada Eloquent

Di Eloquent, terdapat fitur attribute casting yang akan otomatis merubah nilai sebuah field ketika diambil atau disimpan. Pada Larapus, kita akan menggunakan fitur ini untuk merubah isi dari field `is_returned` menjadi nilai `true` atau `false`.

Caranya, tambahkan baris berikut pada model BorrowLog:

app/BorrowLog.php

```
protected $casts = [
    'is_returned' => 'boolean',
];
```

Selain boolean, attribute casting juga dapat merubah nilai sebuah field menjadi integer, real, float, double, string, boolean, object, array, collection, date, datetime, dan timestamp. Lebih detail tentang attribute casting dapat dibaca di [dokumentasi](#)¹.

5.6 Penggunaan Artisan Tinker

Tinker merupakan REPL yang bisa kita gunakan untuk berinteraksi dengan website yang kita bangun tanpa harus membuka browser. Untuk menjalankan tinker, kita jalankan:

```
php artisan tinker
// Psy Shell v0.7.2 (PHP 7.0.7 - cli) by Justin Hileman
>>>
```

Mari kita gunakan tinker untuk mencoba apakah attribute casting pada topik sebelumnya berjalan pada field `is_returned`:

```
>>> $log = App\BorrowLog::first();
=> App\BorrowLog {#757
    id: 1,
    book_id: 1,
    user_id: 2,
    is_returned: 0,
    created_at: "2016-09-06 10:02:05",
    updated_at: "2016-09-06 10:02:05",
}
>>> $log->is_returned
=> false
```

Dari output ini, nilai dari `is_returned` telah berhasil diubah menjadi `false` ketika isinya 0. Begitupun sebaliknya, nilainya akan menjadi `true` ketika isinya 1.

Kita juga dapat mencoba method `pluck()` yang kita pelajari di hari sebelumnya:

¹<https://laravel.com/docs/5.3/eloquent-mutators#attribute-casting>

```
>>> App\Author::pluck('name', 'id')->all();
=> [
    1 => "Mohammad Fauzil Adhim",
    2 => "Salim A. Fillah",
    3 => "Aam Amiruddin",
]
```

Untuk keluar dari tinker, ketik exit.

5.7 Penggunaan Query Scope pada Eloquent

Query Scope sangat bermanfaat untuk menyederhanakan query pada Eloquent. Dengan query scope, kita dapat membuat method baru pada model sebagai alias dari query yang telah kita set.

Pada Larapust, kita akan membuat query scope `returned()` untuk mendapatkan data peminjaman yang sudah dikembalikan dan `borrowed()` untuk mendapatkan data peminjaman yang belum dikembalikan. Tentunya, kedua query scope ini akan kita buat pada model `BorrowLog`.

Tambahkan baris berikut:

app/BorrowLog.php

```
public function scopeReturned($query)
{
    return $query->where('is_returned', 1);
}

public function scopeBorrowed($query)
{
    return $query->where('is_returned', 0);
}
```

Untuk membuat query scope, kita membuat method baru dengan nama `scope{Nama-Scope}` dengan parameter `$query` yang kita gunakan untuk menambahkan query. Pada kedua method diatas, kita menambahkan query `where` untuk mengecek isian `is_returned`.

Kita dapat mengecek query scope ini dengan tinker, misalnya untuk mendapatkan semua data peminjaman yang sudah dikembalikan:

```
>>> App\BorrowLog::returned()->get();
=> Illuminate\Database\Eloquent\Collection {#754
    all: [
        App\BorrowLog {#756
            id: 3,
            book_id: 3,
            user_id: 2,
            is_returned: 1,
            created_at: "2016-09-06 10:02:05",
            updated_at: "2016-09-06 10:02:05",
        },
    ],
}
```

Cobalah gunakan query scope borrowed dan lihat hasilnya.

Lebih lengkap tentang penggunaan Query Scope, dapat dilihat di [dokumentasi](#)².

5.8 Membedakan Tampilan Dashboard untuk Admin dan Member

Untuk memudahkan pengembangan fitur selanjutnya, kita akan set agar view yang ditampilkan ketika user login sebagai Member berbeda dengan Admin.

Lakukan perubahan berikut pada HomeController:

app/Http/Controllers/HomeController.php

```
....  
use Laratrust\LaratrustFacade as Laratrust;  
  
class HomeController extends Controller  
{  
    public function index()  
    {  
        if (Laratrust::hasRole('admin')) return $this->adminDashboard();  
        if (Laratrust::hasRole('member')) return $this->memberDashboard();  
        return view('home');  
    }  
}
```

²<https://laravel.com/docs/5.3/eloquent#query-scopes>

```
protected function adminDashboard()
{
    return view('dashboard.admin');
}

protected function memberDashboard()
{
    return view('dashboard.member');
}
```

Pada perubahan ini, kita menggunakan `Laratrust::hasRole()` untuk mengecek role dari User dan memanggil method yang akan menghandle dashboard untuk role tersebut. Pada role admin, kita menggunakan method `adminDashboard` dan menampilkan view `dashboard.admin`. Pada role member, kita menggunakan method `memberDashboard` dan menampilkan view `dashboard.member`.

Dengan menggunakan teknik ini, URL untuk halaman dashboard tidak akan berubah ketika user login sebagai Admin maupun Member.

Kita buat view untuk dashboard admin:

resources/views/dashboard/admin.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Dashboard</h2>
                    </div>

                    <div class="panel-body">
                        Selamat datang di Menu Administrasi Larapus. Silahkan pilih menu administrasi yang diinginkan.
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Dan view untuk dashboard member:

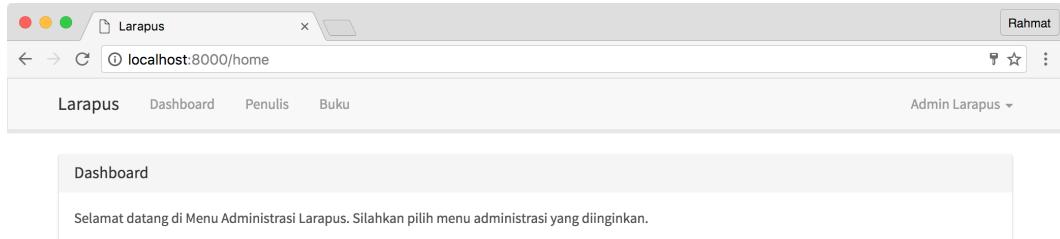
resources/views/dashboard/member.blade.php

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

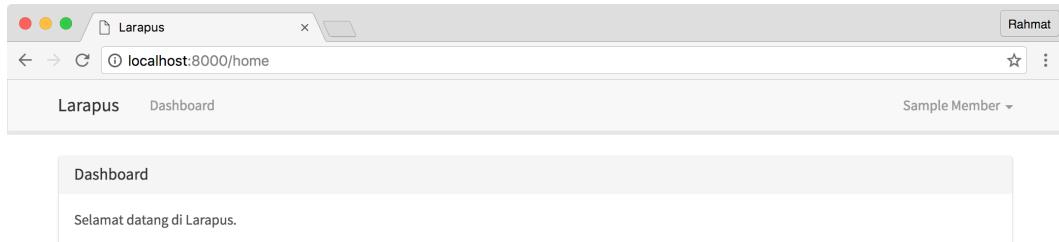
@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Dashboard</h2>
                    </div>

                    <div class="panel-body">
                        Selamat datang di Larapus.
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Kini, jika login sebagai admin, tampilan halaman dashboard akan menjadi:



Sedangkan ketika login sebagai member:



Dashboard Member

5.9 Menampilkan Buku yang Sedang Dipinjam

Pada dashboard member, kita akan menampilkan semua buku yang sedang dipinjam oleh User. Kita akan menggunakan query scope `borrowed` yang telah telah kita buat sebelumnya untuk mendapatkan data peminjaman yang belum dikembalikan. Tambahkan syntax berikut pada method `memberDashboard` di `HomeController`:

app/Http/Controllers/HomeController.php

```
....  
use Illuminate\Support\Facades\Auth;  
  
class HomeController extends Controller  
{  
....  
protected function memberDashboard()  
{  
    return view('dashboard.member');  
    $borrowLogs = Auth::user()->borrowLogs()->borrowed()->get();  
    return view('dashboard.member', compact('borrowLogs'));  
}  
}
```

Pada perubahan ini, kita melakukan passing `borrowLogs` yang berisi data peminjaman yang belum dikembalikan.

Pada view, kita tambahkan baris berikut:

resources/views/dashboard/member.blade.php

```
....  
<div class="panel-body">  
Selamat datang di Larapus.  
<table class="table">  
  <tbody>  
    <tr>  
      <td class="text-muted">Buku dipinjam</td>  
      <td>  
        @if ($borrowLogs->count() == 0)  
          Tidak ada buku dipinjam  
        @endif  
        <ul>  
          @foreach ($borrowLogs as $borrowLog)  
            <li>{{ $borrowLog->book->title }}</li>  
          @endforeach  
        </ul>  
      </td>  
    </tr>  
  </tbody>  
</table>  
</div>  
....
```

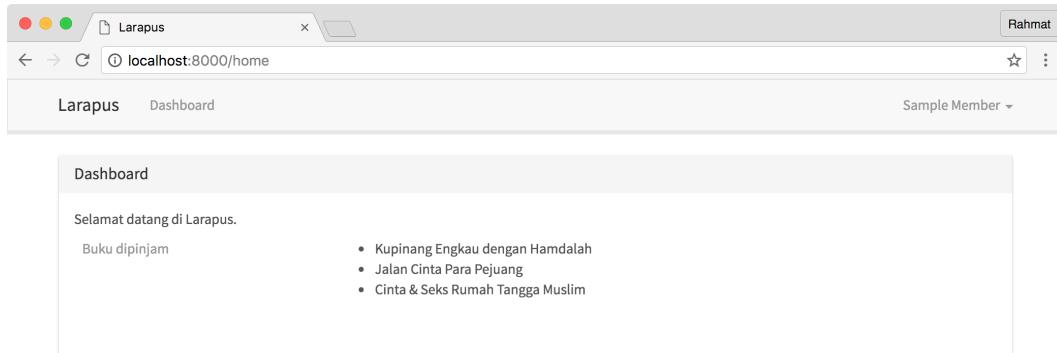
Pada view ini, kita akan menampilkan “Tidak ada buku dipinjam” jika hasil dari method `count()` pada `borrowLogs` yang kita passing dari controller adalah 0. Method `count()` dapat kita gunakan untuk menghitung total model yang kita dapatkan dari hasil query.

Jika model ditemukan, kita menampilkan semua nama bukunya. Karena kita telah menyiapkan relasi dari `BorrowLog` ke `Book`, kita dapat menggunakan:

```
$borrowLog->book->title
```

Untuk mengambil nama buku yang sedang dipinjam.

Ini tampilan halaman dashboard ketika ada buku yang sedang dipinjam:



Menampilkan buku yang sedang dipinjam

5.10 Pengembalian Buku

Untuk mengembalikan buku, kita akan membuat tombol disamping daftar buku yang sedang dipinjam pada dashboard member. Proses ini akan kita arahkan ke `books/{id}/return`. Karena proses pengembalian merupakan proses update record, kita akan menggunakan method `PUT`. Request ini akan kita handle dengan method `returnBack` pada `BooksController`.

Kita buat dulu route nya:

`routes/web.php`

```
Route::put('books/{book}/return', [
    'middleware' => ['auth', 'role:member'],
    'as'          => 'member.books.return',
    'uses'        => 'BooksController@returnBack'
]);
```

Disini, kita juga menggunakan middleware `auth` dan `role:member` untuk memastikan user telah login dan memiliki akses `member`.

Pada `BooksController`, kita tambahkan method `returnBack`:

app/Http/Controllers/BooksController.php

```
public function returnBack($book_id)
{
    $borrowLog = BorrowLog::where('user_id', Auth::user()->id)
        ->where('book_id', $book_id)
        ->where('is_returned', 0)
        ->first();

    if ($borrowLog) {
        $borrowLog->is_returned = true;
        $borrowLog->save();

        Session::flash("flash_notification", [
            "level"  => "success",
            "message" => "Berhasil mengembalikan " . $borrowLog->book->title
        ]);
    }

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

Pada syntax diatas, kita melakukan beberapa hal:

- Mencari data peminjaman untuk user yang sedang login dan id buku yang dikirim. Kita juga menambahkan query where untuk field `is_returned` untuk mendapatkan data peminjaman yang belum dikembalikan. Kita menggunakan method `first()` untuk mendapatkan model pertama dari hasil query ini.
- Jika data peminjaman ditemukan, kita set nilai `is_returned` untuk record tersebut menjadi `true`. Meskipun kita isi `true`, di database isian ini akan berubah menjadi `1` karena kita telah siapkan attribute casting di tahap sebelumnya.
- Set feedback.
- Arahkan user kembali ke halaman dashboard.

Pada view dashboard untuk member, kita lakukan perubahan berikut:

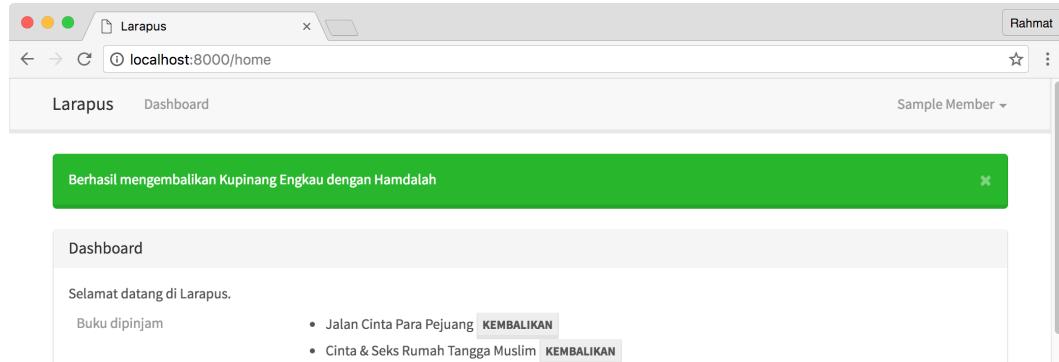
resources/views/dashboard/member.blade.php

```
.....
<li>{{ $borrowLog->book->title }}</li>
<li>
    {!! Form::open(['url' => route('member.books.return', $borrowLog->book_id),
    'method'      => 'put',
    'class'       => 'form-inline js-confirm',
    'data-confirm' => "Anda yakin hendak mengembalikan " . $borrowLog->book->title . "?" ] ) !\n
!}

    {{ $borrowLog->book->title }}
    {!! Form::submit('Kembalikan', ['class'=>'btn btn-xs btn-default']) !!}\n

    {!! Form::close() !!}
</li>
....
```

Tampilan yang akan kita dapatkan ketika berhasil mengembalikan buku akan seperti berikut:



Berhasil mengembalikan buku

5.11 Penggunaan Custom Accessor

Fitur custom accessor pada Eloquent akan memungkinkan kita menambah field baru pada model meskipun field tersebut tidak terdapat di database.

Di Larapus, kita akan membuat field baru bernama `stock` pada model Book. Field ini akan menghitung jumlah field `amount` dikurangi jumlah peminjaman untuk buku tersebut yang belum dikembalikan.

Tambahkan method berikut pada model Book:

app/Book.php

```
public function borrowLogs()
{
    return $this->hasMany('App\BorrowLog');
}

public function getStockAttribute()
{
    $borrowed = $this->borrowLogs()->borrowed()->count();
    $stock = $this->amount - $borrowed;
    return $stock;
}
```

Sebelum membuat accessor, kita membuat method `borrowLogs` yang akan kita gunakan untuk mengakses relasi dari model Book ke BorrowLog.

Custom accessor dibuat dengan menambah method dengan penamaan `get{Nama-Field}Attribute`, disini kita membuat `getStockAttribute`.

Untuk mendapatkan total buku yang sedang dipinjam, kita menggunakan method `borrowLogs()` untuk semua record yang berelasi di model BorrowLog. Selanjutnya, kita menggunakan query scope `borrowLog` untuk mendapatkan hanya data peminjaman yang belum dikembalikan. Terakhir, kita menggunakan method `count()` untuk menghitung total model yang kita dapatkan.

Syntax selanjutnya, kita menghitung field `stock` dengan mengurangi nilai field `amount` dengan hasil yang kita dapatkan sebelumnya. Terakhir, kita kembalikan nilai `stock`.

Misalnya, kita memiliki data seperti berikut:

	<code>id</code>	<code>book_id</code>	<code>user_id</code>	<code>is_returned</code>	<code>created_at</code>	<code>updated_at</code>
1	1	2	2	0	2016-09-06 10:02:05	2016-09-07 03:02:45
2	2	2	2	0	2016-09-06 10:02:05	2016-09-06 10:02:05
3	3	2	2	1	2016-09-06 10:02:05	2016-09-06 10:02:05

TABLE INFORMATION

- created: 9/7/16
- engine: InnoDB
- rows: 3
- size: 16.0 KiB
- encoding: utf8
- auto_increment: 6

Sample data peminjaman

Nilai field stock untuk record dengan ID 2 akan seperti berikut (disini menggunakan tinker):

```
=> App\Book {#765
    id: 2,
    title: "Jalan Cinta Para Pejuang",
    author_id: 2,
    amount: 2,
    cover: null,
    created_at: "2016-09-06 10:02:05",
    updated_at: "2016-09-06 10:02:05",
}
>>> $book->stock
=> 1
```

Masih banyak fitur dari Custom Accessor yang tidak kita gunakan. Lebih lengkapnya dapat membaca di [dokumentasi](#)³.

5.12 Menampilkan Stok Buku

Mari kita tampilkan stok buku pada halaman utama Larapus. Lakukan perubahan berikut pada method index di GuestController:

³<https://laravel.com/docs/5.3/eloquent-mutators#accessors-and-mutators>

app/Http/Controllers/GuestController.php

```
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $books = Book::with('author');
        return Datatables::of($books)
            ->addColumn('stock', function($book){
                return $book->stock;
            })
            ->addColumn('action', function($book){
                if (Laratrust::hasRole('admin')) return '';
                return '<a class="btn btn-xs btn-primary" href="'.route('guest.books.borrow', $book->\id).'">Pinjam</a>';
            })->make(true);
    }

    $html = $htmlBuilder
        ->addColumn(['data' => 'title', 'name'=>'title', 'title'=>'Judul'])
        ->addColumn(['data' => 'stock', 'name'=>'stock', 'title'=>'Stok', 'orderable'=>false, 'se\archable'=>false])
        ->addColumn(['data' => 'author.name', 'name'=>'author.name', 'title'=>'Penulis'])
        ->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, 'sear\chable'=>false]);

    return view('guest.index')->with(compact('html'));
}
```

Pada syntax diatas, kita menggunakan method `addColumn` untuk menambah kolom baru pada DataTable bernama `stock`. Ini kita lakukan karena field `stock` tidak terdapat di table `books`. Pada `$htmlBuilder`, method `addColumn` untuk field `stock` juga kita set agar tidak bisa di urutkan dan di search untuk menghindari error.

Tampilan halaman utama Larapus akan seperti berikut:

Judul	Stok	Penulis	
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	PINJAM
Jalan Cinta Para Pejuang	1	Salim A. Fillah	PINJAM
Kupinang Engkau dengan Hamdalah	3	Mohammad Fauzil Adhim	PINJAM
Membingkai Surga dalam Rumah Tangga	4	Aam Amiruddin	PINJAM

Menampilkan field stock

5.13 Membatasi Peminjaman Berdasarkan Stok Buku

Jika stock buku sudah 0, seharusnya seorang member tidak dapat meminjam buku lagi. Mari kita buat rule ini pada method borrow() di model User:

app/User.php

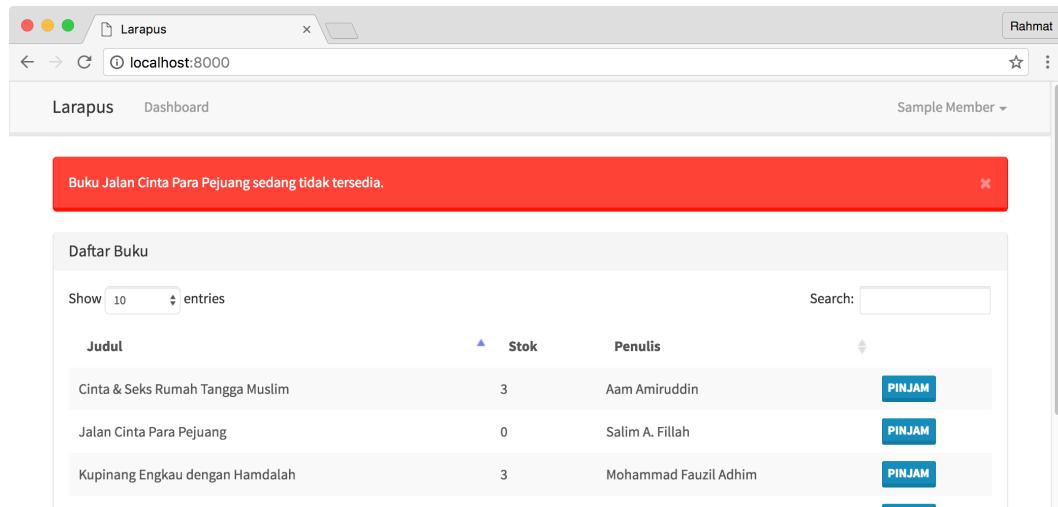
```
public function borrow(Book $book)
{
    // cek apakah masih ada stok buku
    if ($book->stock < 1) {
        throw new BookException("Buku $book->title sedang tidak tersedia.");
    }

    // cek apakah buku ini sedang dipinjam oleh user
    if($this->borrowLogs()->where('book_id',$book->id)->where('is_returned', 0)->count() > 0 ) {
        throw new BookException("Buku $book->title sedang Anda pinjam.");
    }

    $borrowLog = BorrowLog::create(['user_id'=>$this->id, 'book_id'=>$book->id]);
}
```

```
    return $borrowLog;
}
```

Untuk mencoba fitur ini, buatlah beberapa user member dan pinjamlah buku yang sama. Ketika kita mencoba meminjam sebuah buku yang stock nya 0, kita akan mendapat error berikut:



The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000'. The page has a header with 'Rahmat' and 'Sample Member'. Below the header, there's a navigation bar with 'Larapus' and 'Dashboard'. A red error message box contains the text 'Buku Jalan Cinta Para Pejuang sedang tidak tersedia.' (Book 'Jalan Cinta Para Pejuang' is currently unavailable). The main content area is titled 'Daftar Buku' and shows a table of books with columns: Judul, Stok, Penulis, and PINJAM. The table data is as follows:

Judul	Stok	Penulis	
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	<button>PINJAM</button>
Jalan Cinta Para Pejuang	0	Salim A. Fillah	<button>PINJAM</button>
Kupinang Engkau dengan Hamdalih	3	Mohammad Fauzil Adhim	<button>PINJAM</button>

Tidak dapat meminjam buku ketika stok habis

5.14 Validasi Ketika Mengubah Jumlah Buku

Bayangkan sebuah buku sedang dipinjam oleh 2 user, ketika admin mengubah amount dari buku tersebut, tentunya nilainya tidak boleh kurang dari 2 agar sistem tidak error.

Untuk membuat fitur ini kita akan menggunakan model event untuk event updating pada model Book. Event ini akan berjalan pada saat kita mencoba melakukan perubahan pada sebuah model. Seperti event deleting yang kita gunakan sebelumnya, jika kita mengembalikan nilai false pada event ini, maka proses update akan dibatalkan.

Tambahkan syntax berikut pada model Book:

app/Book.php

```
....  
use Illuminate\Support\Facades\Session;  
  
class Book extends Model  
{  
    public static function boot()  
    {  
        parent::boot();  
  
        self::updating(function($book)  
        {  
            if ($book->amount < $book->borrowed) {  
                Session::flash("flash_notification", [  
                    "level"=>"danger",  
                    "message"=>"Jumlah buku $book->title harus >= " . $book->borrowed  
                ]);  
                return false;  
            }  
        });  
    }  
  
    public function getBorrowedAttribute()  
    {  
        return $this->borrowLogs()->borrowed()->count();  
    }  
....  
}
```

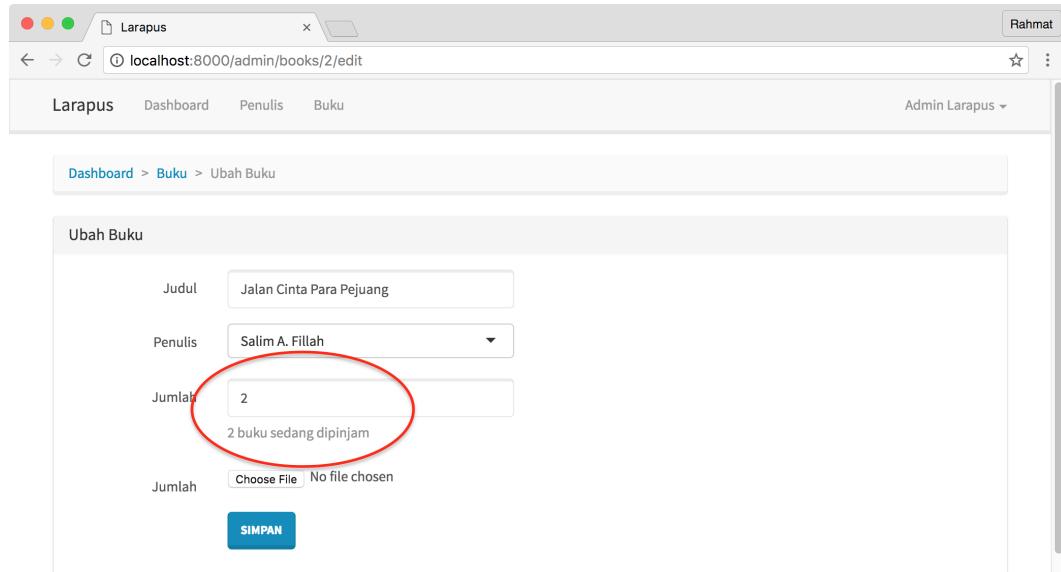
Pada syntax diatas, pada event `updating` kita mengecek isian field `amount` dengan field `borrowed`. Field `borrowed` adalah accessor baru yang kita buat. Field ini berisi total buku yang sedang dipinjam. Jika nilai `amount` kurang dari `borrowed`, kita set flash data untuk pesan error dan mengembalikan nilai `false` untuk membatalkan proses update.

Pada form buku, kita akan menambahkan helper untuk menampilkan jumlah buku yang sedang dipinjam:

resources/views/books/_form.blade.php

```
<div class="form-group{{ $errors->has('amount') ? ' has-error' : '' }}>
  {!! Form::label('amount', 'Jumlah', ['class'=>'col-md-2 control-label']) !!}
  <div class="col-md-4">
    {!! Form::number('amount', null, ['class'=>'form-control', 'min'=>1]) !!}
    {!! $errors->first('amount', '<p class="help-block">:message</p>') !!}
    @if (isset($book))
      <p class="help-block">{{ $book->borrowed }} buku sedang dipinjam</p>
    @endif
  </div>
</div>
```

Kini, tampilan halaman edit buku akan seperti berikut:



Menampilkan jumlah buku yang dipinjam

Pada method update di BooksController, kita ubah menjadi:

app/Http/Controllers/BooksController.php

```
....  
public function update(UpdateBookRequest $request, $id)  
{  
    $book = Book::find($id);  
$book->update($request->all());  
    if (!$book->update($request->all())) return redirect()->back();  
    ....  
}
```

Pada saat kita mencoba mengubah `amount` dengan nilai yang kurang dari total buku yang dipinjam, akan muncul error berikut:

The screenshot shows a browser window titled 'Larapus' with the URL 'localhost:8000/admin/books/2/edit'. The page has a navigation bar with 'Larapus', 'Dashboard', 'Penulis', and 'Buku'. A user 'Admin Larapus' is logged in. A red error message at the top says 'Jumlah buku Jalan Cinta Para Pejuang harus >= 2'. Below it, the 'Ubah Buku' form is displayed. The form fields are: 'Judul' (Jalan Cinta Para Pejuang), 'Penulis' (Salim A. Fillah), 'Jumlah' (2). A note below the 'Jumlah' field says '2 buku sedang dipinjam'. At the bottom, there's a file upload field labeled 'Choose File' with 'No file chosen'. The entire screenshot is framed by a light gray border.

Tidak dapat mengubah buku



Gunakan Form Request

Teknik lain untuk melakukan validasi ini adalah dengan menambahkan rule `min:` pada validasi untuk field `amount`. Pada pembahasan sebelumnya, kita telah menggunakan form request untuk melakukan validasi ketika proses update buku. Ubahlah validasi yang kita lakukan pada pembahasan ini agar menggunakan rule `amount` di `UpdateBookRequest`.

5.15 Membatasi Penghapusan Ketika Buku Masih Dipinjam

Ketika buku sudah pernah atau sedang dipinjam, tentunya kita tidak ingin admin dapat menghapus buku tersebut. Mari kita tambah fitur ini dengan menggunakan event deleting di model Book.

Tambahkan baris berikut pada method boot di model Book:

app/Book.php

```
public static function boot()
{
    ...
    self::deleting(function($book)
    {
        if ($book->borrowLogs()->count() > 0) {
            Session::flash("flash_notification", [
                "level"=>"danger",
                "message"=>"Buku $book->title sudah pernah dipinjam."
            ]);
            return false;
        }
    });
}
```

Pada statement if diatas, kita mengecek apakah ada record di BorrowLog untuk buku ini. Jika ya, kita buat data session untuk pesan error dan kita batalkan proses penghapusan buku.

Pada method destroy di BooksController, kita harus membuat perubahan berikut:

app/Http/Controllers/BooksController.php

```
public function destroy($id)
{
    $book = Book::find($id);
    $cover = $book->cover;
    if (!$book->delete()) return redirect()->back();
    // hapus cover lama, jika ada
    if ($book->cover) {
        if ($cover) {
            $old_cover = $book->cover;
            $filepath = public_path() . DIRECTORY_SEPARATOR . 'img'
                . DIRECTORY_SEPARATOR . $book->cover;

            try {
                File::delete($filepath);
            } catch (FileNotFoundException $e) {
                // File sudah dihapus/tidak ada
            }
        }
    }
    $book->delete();

    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=>"Buku berhasil dihapus"
    ]);

    return redirect()->route('admin.books.index');
}
```

Jika kita mencoba menghapus buku yang sedang dipinjam, akan muncul error berikut:

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/books'. The page header includes 'Larapus', 'Dashboard', 'Penulis', 'Buku', and 'Admin Larapus'. A red success message box at the top states 'Buku Jalan Cinta Para Pejuang sudah pernah dipinjam.' Below this is a breadcrumb navigation 'Dashboard > Buku'. The main content area is titled 'Buku' and contains a table with two rows of book data. The columns are 'Judul', 'Jumlah', and 'Penulis'. The first row has 'Cinta & Seks Rumah Tangga Muslim' in 'Judul', '3' in 'Jumlah', and 'Aam Amiruddin' in 'Penulis'. The second row has 'Jalan Cinta Para Pejuang' in 'Judul', '2' in 'Jumlah', and 'Salim A. Fillah' in 'Penulis'. Each row has 'Ubah' and 'HAPUS' buttons. A modal dialog at the bottom center says 'Batal menghapus buku'.

Judul	Jumlah	Penulis
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin
Jalan Cinta Para Pejuang	2	Salim A. Fillah

5.16 Ringkasan

Di Hari 5 ini, banyak sekali yang kita pelajari diantaranya :

- Penggunaan Attribute Casting
- Penggunaan Artisan Tinker
- Penggunaan Query Scope
- Penggunaan Custom Accessor
- Dsb.

Pada hari 6, kita akan membuat fitur-fitur untuk manajemen user. Semangat! :)

6. Hari 6 : User Management

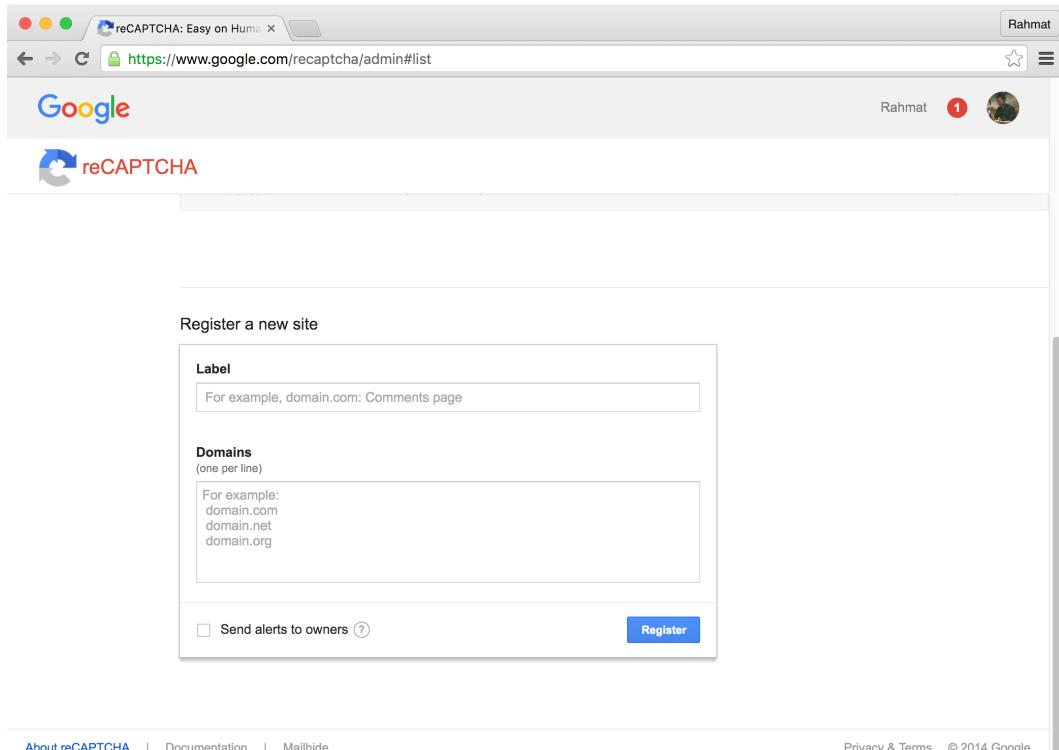
Aplikasi Larapus yang kita bangun hampir lengkap. Pada hari ini kita akan membuat beberapa fitur yang berhubungan dengan manajemen user seperti penggunaan captcha untuk meningkatkan keamanan, verifikasi user, halaman profil, data pem-injaman dan data member untuk admin.

6.1 Penggunaan No CAPTCHA reCAPTCHA

Jika belum mengerti Captcha, teknik ini adalah berguna untuk mengurangi spam di form yang muncul di web. Di Larapu, kita akan menggunakan Captcha untuk menambah proteksi pada halaman register. Captcha, selain digunakan untuk halaman register, biasanya digunakan juga di halaman komentar untuk mengurangi spam. Ada berbagai implementasi untuk Captcha, tapi dalam buku ini kita akan menggunakan No Captcha ReCaptcha dari Google.

Untuk menggunakan ReCaptcha, kita harus mendapatkan *Site Key* dan *Secret Key* dari Google. Untuk itu, silahkan buat dulu akun reCaptcha di <http://www.google.com/recaptcha>¹. Setelah masuk ke halaman Admin, isi form “Register a new site”. Pada saat buku ini ditulis, tampilannya seperti berikut:

¹<http://www.google.com/recaptcha>



Pembuatan Key Captcha Baru

Isian *label* digunakan sebagai catatan untuk key yang kita buat, kita isi dengan Larapus. Isian *Domains* digunakan untuk menandai domain apa saja yang diizinkan untuk menggunakan key ini. Jika sudah punya domain yang akan digunakan untuk mengupload Larapus, silahkan isi disini. Minimal kita harus mengisinya dengan localhost agar key yang dibuat dapat kita gunakan di Larapus versi lokal.

reCAPTCHA: Easy on Humans

https://www.google.com/recaptcha/admin#list

Rahmat

Google

reCAPTCHA

Register a new site

Label

larpus

Domains
(one per line)

localhost

Send alerts to owners ?

Register

About reCAPTCHA | Documentation | Mailhide

Privacy & Terms © 2014 Google

Membuat key baru

Klik “Register”.

Setelah berhasil, pada halaman selanjutnya kita akan mendapatkan *Site Key* dan *Secret Key*. Catatlah kedua key tersebut.

① Adding reCAPTCHA to your site

Keys

Site key
Use this in the HTML code your site serves to users.
6Lf_Ex0TAaaaaO17TnnduG27yH-LMwgR3nDoYq-U

Secret key
Use this for communication between your site and Google. Be sure to keep it a secret.
6Lf_Ex0TAaaaaOgbWYTsnP-yLC5x9Fgx_ocpg9CY

Step 1: client-side integration

Paste this snippet before the closing </head> tag on your HTML template:

```
<script src='https://www.google.com/recaptcha/api.js'></script>
```

Paste this snippet at the end of the <form> where you want the reCAPTCHA widget to appear:

```
<div class="g-recaptcha" data-sitekey="6Lf_Ex0TAaaaaO17TnnduG27yH-LMwgR3nDoYq-U"></div>
```

The reCAPTCHA documentation site describes more details and advanced configurations.

Step 2: Server side integration

Berhasil membuat key untuk captcha

Untuk menggunakan NoCaptcha ReCaptcha kita membutuhkan [logic di sisi server](#)². Agar kita tidak melakukannya secara manual, kita akan menggunakan library [anhskohbo/no-captcha](#)³. Kita install dengan perintah berikut:

```
composer require anhskohbo/no-captcha=~2.0
```

Setelah proses download selesai, tambahkan baris berikut pada isian providers di config/app.php:

²<https://developers.google.com/recaptcha/docs/verify>

³<https://github.com/anhskohbo/no-captcha>

config/app.php

```
....  
'providers' => [  
    ....  
    Anhskohbo\NoCaptcha\NoCaptchaServiceProvider::class,  
,  
....
```

Pada file .env, kita harus membuat variable NOCAPTCHA_SECRET dan NOCAPTCHA_SITEKEY berisi key yang sudah kita dapatkan dari Google.

.env

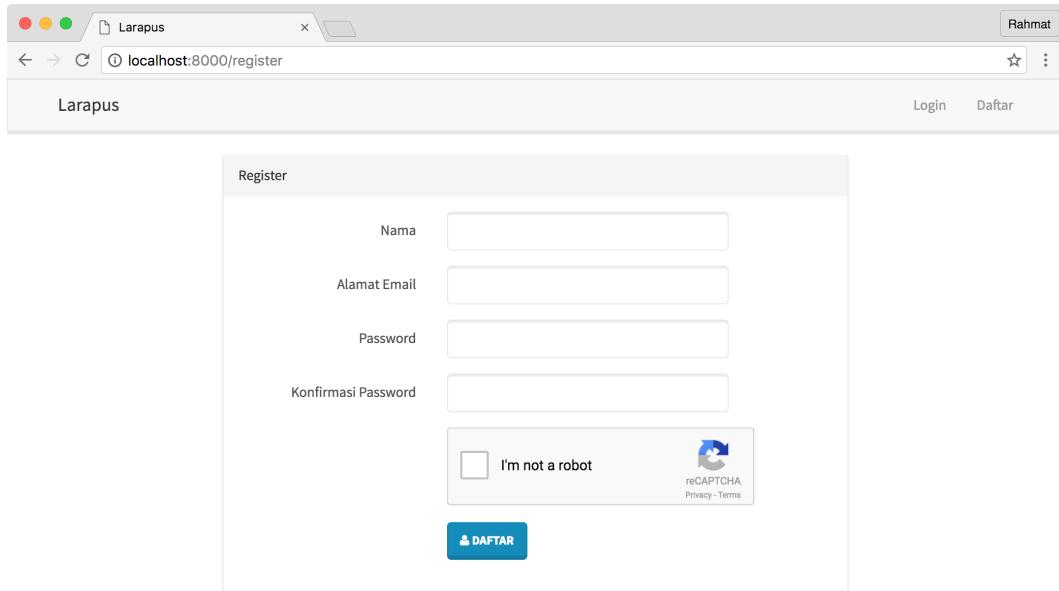
```
....  
NOCAPTCHA_SECRET=secret-key-dari-google  
NOCAPTCHA_SITEKEY=site-key-dari-google
```

Untuk menambahkan Captcha pada halaman daftar user, tambahkan baris berikut pada resources/views/auth/register.blade.php setelah isian untuk password_confirmation:

resources/views/auth/register.blade.php

```
....  
<div class="form-group{{ $errors->has('g-recaptcha-response') ? ' has-error' : '' }}>  
    <div class="col-md-offset-4 col-md-6">  
        {!! app('captcha')->display() !!}  
        {!! $errors->first('g-recaptcha-response', '<p class="help-block">:message</p>') !!}  
    </div>  
</div>  
....
```

Sehingga tampilan halaman daftar akan menjadi:



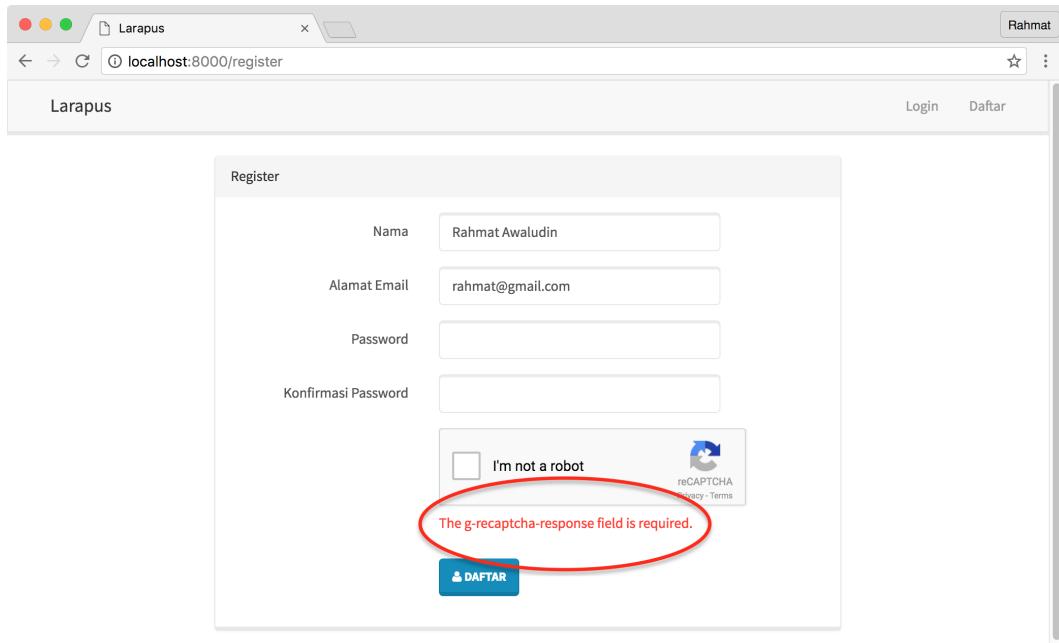
Halaman daftar user dengan Captcha

Kita juga harus melakukan perubahan pada method `validator` di `RegisterController` agar melakukan validasi untuk captcha:

`app/Http/Controllers/Auth/RegisterController.php`

```
....  
protected function validator(array $data)  
{  
    return Validator::make($data, [  
        'name' => 'required|max:255',  
        'email' => 'required|email|max:255|unique:users',  
        'password' => 'required|confirmed|min:6',  
        'g-recaptcha-response' => 'required|captcha',  
    ]);  
}  
....
```

Kini, cobalah mendaftar di Larapus. Jika kita tidak mengisi captcha, kita akan mendapat error berikut:



Captcha harus diisi

6.2 Konfigurasi Email dengan Mailgun

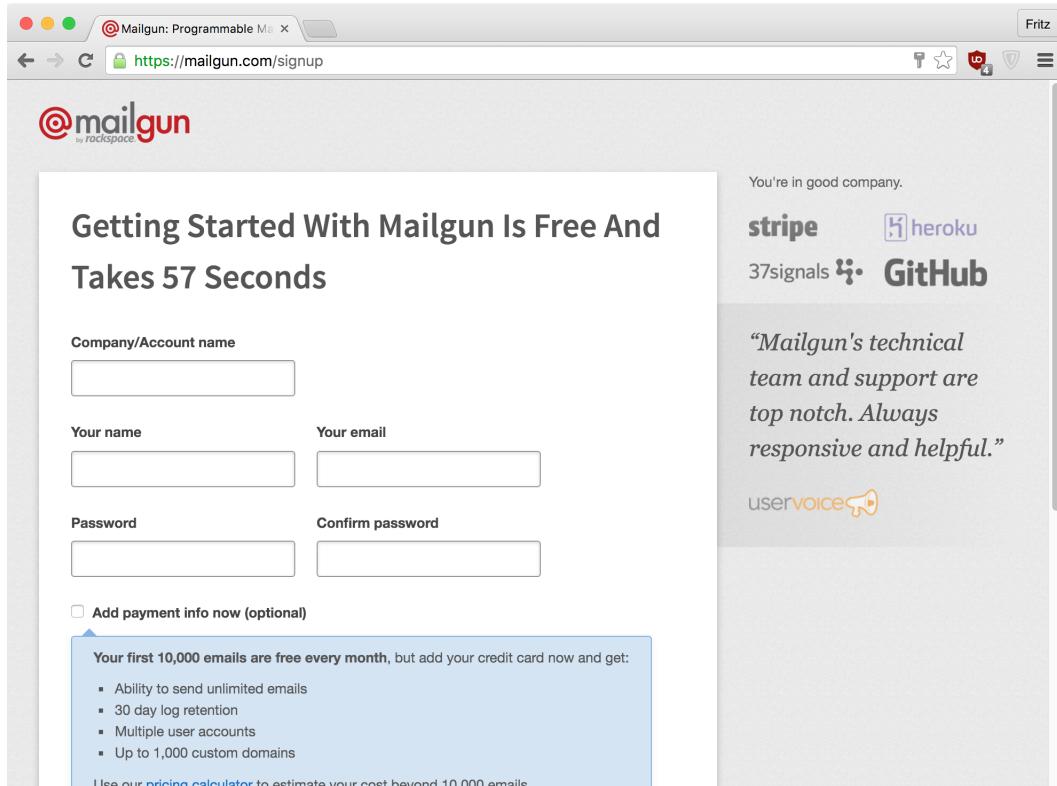
Laravel memungkinkan kita menggunakan berbagai provider untuk mengirimkan email. Beberapa provider untuk mengirim email yang didukung secara native oleh Laravel diantaranya SMTP, Mailgun, Mandrill, Amazon SES, fungsi Mail di PHP, dan sendmail. Menggunakan provider pihak ketiga seperti Mailgun, Mandrill dan Amazon SES akan mempercepat proses pengiriman email karena yang dilakukan aplikasi kita hanya melakukan API request ke provider tersebut.

Pada Larapus, kita akan menggunakan Mailgun untuk mengirim email. Menggunakan akun gratis Mailgun, kita dapat mengirim 10,000 email secara gratis tiap bulan. Untuk ukuran aplikasi yang sedang kita bangun, jumlah itu pasti sudah cukup.

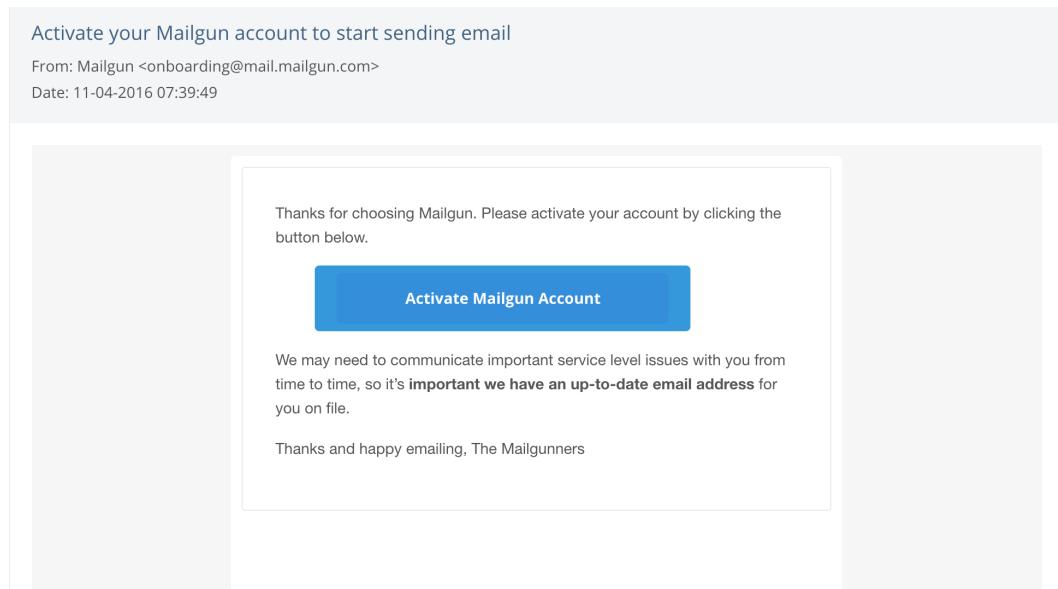
Dalam mengirim email dengan mailgun, kita dapat menggunakan domain yang kita miliki atau menggunakan sandbox domain yang disediakan oleh Mailgun. Dalam proses development, biasanya kita cukup dengan menggunakan sandbox domain.

Kekurangannya, biasanya email akan masuk ke folder spam dan terbatas hanya 300 email per hari.

Mari kita dapatkan sandbox domain dengan mendaftar di Mailgun.

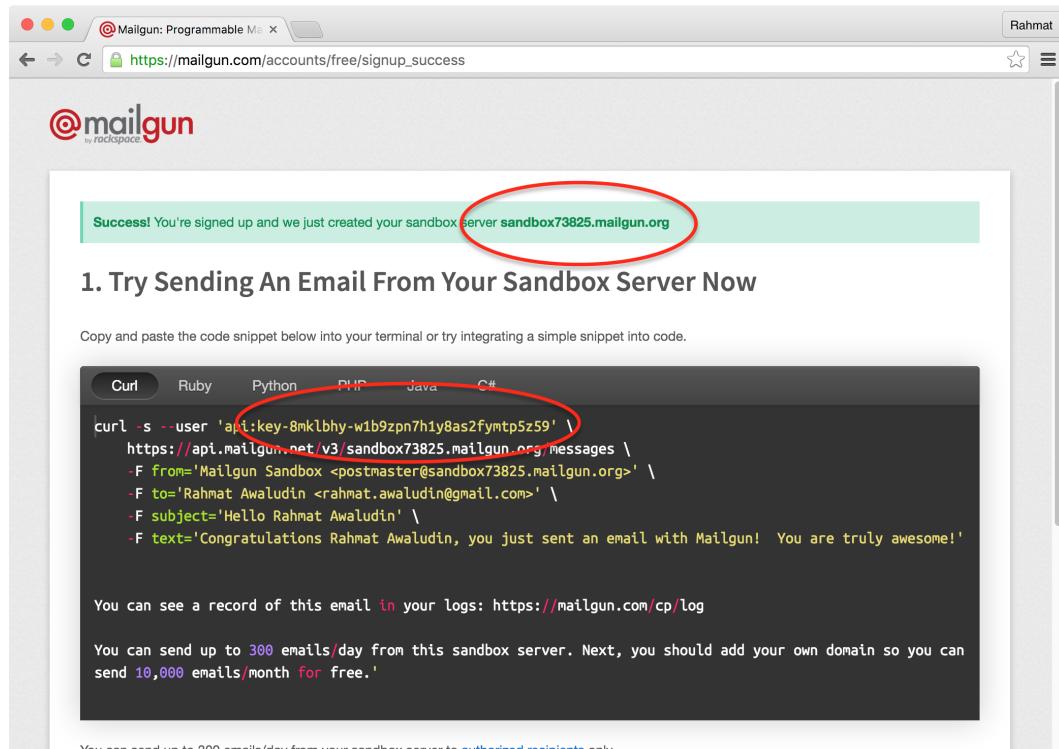


Setelah mendaftar, pastikan untuk klik link aktivasi akun yang didapatkan dari Mailgun agar sandbox kita berfungsi.



Aktivasi Akun

Pada halaman pertama setelah berhasil mendaftar, kita akan mendapatkan alamat sandbox domain:



Success! You're signed up and we just created your sandbox server **sandbox73825.mailgun.org**

1. Try Sending An Email From Your Sandbox Server Now

Copy and paste the code snippet below into your terminal or try integrating a simple snippet into code.

Curl Ruby Python PHP Java C#

```
curl -s --user 'api:key-8mklbhy-w1b9zpn7h1y8as2fymtp5z59' \
  https://api.mailgun.net/v3/sandbox73825.mailgun.org/messages \
  -F from='Mailgun Sandbox <postmaster@sandbox73825.mailgun.org>' \
  -F to='Rahmat Awaludin <rahmat.awaludin@gmail.com>' \
  -F subject='Hello Rahmat Awaludin' \
  -F text='Congratulations Rahmat Awaludin, you just sent an email with Mailgun! You are truly awesome!'
```

You can see a record of this email in your logs: <https://mailgun.com/cp/log>

You can send up to 300 emails/day from this sandbox server. Next, you should add your own domain so you can send 10,000 emails/month for free.'

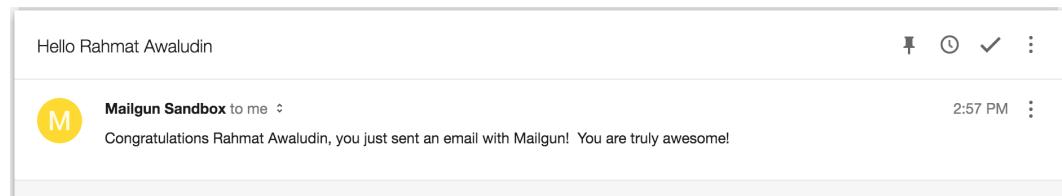
You can send up to 300 emails/day from your sandbox server to [authorized recipients](#) only.

Sandbox domain

Pada contoh ini, kita mendapatkan domain **sandbox73825.mailgun.org**. Terlihat pada halaman diatas, kita juga mendapatkan API Key untuk sandbox domain tersebut yaitu **key-8mklbhy-w1b9zpn7h1y8as2fymtp5z59**.

Kita dapat mencoba contoh syntax [Curl⁴](#) diatas pada terminal (untuk pengguna windows, Curl mesti diinstall dulu). Ini email yang akan kita dapatkan:

⁴<https://curl.haxx.se/download.html>



Test email



Error Business Verification

Jika mendapat error dengan pesan “Business Verification” setelah mendaftar, berarti email yang digunakan terdeteksi sebagai email spam. Kadang deteksi ini bisa salah. Jika itu terjadi, ada dua cara yang bisa dilakukan.

1. Pastikan mendaftar mailgun dengan email yang valid. Saran saya gunakan akun gmail.
2. Kontak customer support dan jelaskan detail sesuai petunjuk di halaman ini⁵. Jika cara ini memungkinkan, gunakan cara kedua,
3. Gunakan Key dan sandbox yang berada di yang tertera di URL <https://mailgun.com> (homepage) bukan yang didalam dashboard Mailgun. Lebih jelasnya, lihat URL pada screenshot dibawah. Untuk mengakses halaman ini, kita harus logout terlebih dahulu dari Mailgun.

The screenshot shows a web browser window with the URL www.mailgun.com in the address bar. A red circle highlights the address bar. To the right, a terminal window displays a command-line interface for sending an email using the Mailgun API. The command is:

```
# Try our API. Copy & run this in your terminal.
curl -s -u user 'api:key-3ax6xnjp29jd6fds4gc373sgvjxteol0' \
  https://api.mailgun.net/v3/samples.mailgun.org/messages \
  -F from='Excited User <excited@smples.mailgun.org>' \
  -F to='devs@mailgun.net' \
  -F subject='Hello' \
  -F text='Testing some Mailgun awesomeness!'
```

Sample key dan sandbox

Disini, key yang kita dapatkan adalah `key-3ax6xnjp29jd6fds4gc373sgvjxteol0` dengan domain `sandbox samples.mailgun.org`.

Mari kita konfigurasi Laravel agar mengirim email dengan mailgun dengan mengubah isian MAIL_DRIVER di file .env:

.env

```
MAIL_DRIVER=mailgun
```

Jika kita cek pada file config/services.php, akan kita temui baris seperti ini:

config/services.php

```
'mailgun' => [
    'domain' => env('MAILGUN_DOMAIN'),
    'secret' => env('MAILGUN_SECRET'),
],
```

Terlihat disini, isian ini menggunakan nilai dari file .env untuk variable MAILGUN_DOMAIN dan MAILGUN_SECRET. Ubahlah kedua isian tersebut pada file .env dengan key dan sandbox domain yang kita dapatkan:

.env

```
MAILGUN_DOMAIN=alamat-domain-sandbox
MAILGUN_SECRET=api-key-sandbox
```

Sebelum mengirim email, pastikan sudah mengisi key from pada file config/mail.php. Misalnya seperti berikut:

config/mail.php

```
'from' => [
    'address' => 'admin@larapus.com',
    'name' => 'Admin Larapus',
],
```



Karena kita melakukan perubahan pada file .env, restartlah server artisan.

Laravel mengharuskan kita untuk menginstall [Guzzle](#)⁶ jika hendak menggunakan Mailgun. Kita install dengan:

⁶<https://github.com/guzzle/guzzle>

```
composer require guzzlehttp/guzzle=~6.0
```



Error Guzzle

Terkadang saat menggunakan Guzzle dengan Mailgun kita mendapatkan error *cURL error 60: SSL certificate problem: unable to get local issuer certificate*. Untuk mengatasinya, ubah versi Guzzle di `composer.json` menjadi ~ 4.0 dan jalankan `composer update`.

Kita dapat mencoba fitur ini dengan mencoba mendaftar ke Larapus dan mencoba fitur lupa password. Ini email yang akan kita dapatkan pada inbox:

The screenshot shows an email titled "Reset Password" from "Admin Larapus" at "rahmat.awaludin@gmail.com". The email body starts with "Hello!", followed by a message about receiving a password reset request. It includes a blue "Reset Password" button. Below the button, it says if you did not request a password reset, no further action is required. It ends with "Regards, Larapus". At the bottom, there is a URL "http://localhost:8000/password/reset/4e0a5ce7d3f2661378ee3224c8822e5f6dfd7e90e96daca9068c2156195641ad" and social sharing icons for LinkedIn, Facebook, and Twitter.

Reset Password

If you did not request a password reset, no further action is required.

Regards,
Larapus

If you're having trouble clicking the "Reset Password" button, copy and paste the URL below into your web browser:

<http://localhost:8000/password/reset/4e0a5ce7d3f2661378ee3224c8822e5f6dfd7e90e96daca9068c2156195641ad>

Email reset password

Akan ada sedikit delay sebelum email sampai di inbox. Cek juga folder spam jika email masih belum sampai.

6.3 Fitur Verifikasi/Aktivasi User

Ketika membuat akun di sebuah website, biasanya kita akan menerima email untuk mengaktifkan user. Laravel secara default tidak memiliki fitur seperti ini. Fitur ini bisa kita buat dengan menggunakan [package](#)⁷ atau secara manual. Di Larapus, kita akan membuat fitur ini secara manual sambil kita belajar berbagai fitur di Laravel.

⁷<https://github.com/jrean/laravel-user-verification>

Proses verifikasi akan kita lakukan sebagaimana proses “Lupa Password”. User akan klik link pada email yang kita kirim. Jika email dan tokennya sesuai dengan data di database, kita akan set user tersebut sebagai user yang sudah terverifikasi.

6.3.1 Persiapan Database

Untuk menyimpan data verifikasi, kita akan menambah field `verification_token` dan `is_verified` ke table `users`. Field tersebut akan kita gunakan untuk menyimpan token verifikasi dan status verifikasi user.

Kita akan membuat migration baru untuk menambah field ini, jalankan perintah berikut:

```
php artisan make:migration add_verification_to_users --table=users
// Created Migration: xxxx_xx_xx_xxxxxx_add_verification_to_users
```

Opsi `--table=users` akan membuat migration yang kita generate memiliki template untuk melakukan perubahan pada table `users`.

Pada method `up` dari file migration yang dibuat, isi dengan:

`database/migrations/xxxx_xx_xx_xxxxxx_add_verification_to_users.php`

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('verification_token')->nullable();
        $table->boolean('is_verified')->default(0);
    });
}
```

Pada method `down`, kita isi dengan:

database/migrations/xxxx_xx_xx_xxxxxx_add_verification_to_users.php

```
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('verification_token');
        $table->dropColumn('is_verified');
    });
}
```

Pada syntax diatas, kita akan membuat kedua field yang kita butuhkan saat kita melakukan migrate dan menghapus kedua field tersebut pada saat kita lakukan rollback.

Untuk memudahkan pengembangan fitur selanjutnya, kita akan set attribute casting untuk field `is_verified` pada model User. Tambahkan syntax berikut:

app/User.php

```
protected $casts = [
    'is_verified' => 'boolean',
];
```

Kita akan set sample user untuk Admin dan Member sebagai user yang sudah terverifikasi. Tambahkan baris berikut pada `UsersSeeder`:

database/seeds/UsersSeeder.php

```
...
// Membuat sample admin
...
$admin->is_verified = 1;
$admin->save();
$admin->attachRole($adminRole);

// Membuat sample member
...
$member->is_verified = 1;
$member->save();
$member->attachRole($memberRole);
...
```

Untuk mendapatkan struktur database dan sample data yang baru, kita refresh database:

```
php artisan migrate:refresh --seed
```

6.3.2 Membatasi Akses yang Belum Terverifikasi dengan Middleware

Kita akan menggunakan middleware untuk melakukan pengecekan apakah user yang login sudah terverifikasi. Jika belum, kita akan logout, set feedback dan arahkan user ke halaman login.

Kita akan menamai middleware ini dengan `UserShouldVerified`. Kita buat dengan perintah:

```
php artisan make:middleware UserShouldVerified
// Middleware created successfully.
```

Pada file yang dihasilkan, isi dengan syntax berikut:

app/Http/Middleware/UserShouldVerified.php

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;

class UserShouldVerified
{
    public function handle($request, Closure $next)
    {
        return $next($request);
        $response = $next($request);
        if (Auth::check() && !Auth::user()->is_verified) {
            Auth::logout();

            Session::flash("flash_notification", [
                "level"    => "warning",
                "message" => "Akun Anda belum aktif. Silahkan klik pada link aktivasi yang telah kami kirim."
            ]);
        }
    }
}
```

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

    return $response;
}
}
```

Pada middleware ini, kita mengecek apakah user sudah login dan akunnya belum terverifikasi dengan mengecek field `is_verified`. Jika ya, maka kita logout dan tampilkan pesan error “Akun Anda belum aktif. Silahkan klik pada link aktivasi yang telah kami kirim.”.

Kita daftarkan middleware ini dengan menambahkan pada array `routeMiddleware` di `app/Http/Kernel.php`:

app/Http/Kernel.php

```
protected $routeMiddleware = [
    ...
    'user-should-verified' => \App\Http\Middleware\UserShouldVerified::class,
];
```

Disini, kita menggunakan alias `user-should-verified` untuk menggunakan middleware yang baru kita buat.

Agar middleware ini aktif setelah user login, kita harus menambahkannya di `LoginController`:

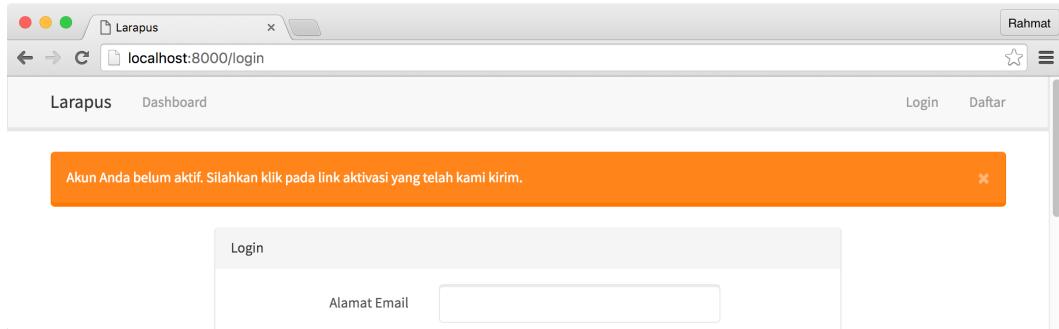
app/Http/Controllers/Auth/LoginController.php

```
public function __construct()
{
    $this->middleware('guest', ['except' => 'logout']);
    $this->middleware('user-should-verified');
}
```

Untuk mencoba middleware ini, kita dapat mencoba mengubah isian `is_verified` untuk user Member dan mencoba login dengan user tersebut.

```
php artisan tinker
Psy Shell v0.7.2 (PHP 7.0.7 - cli) by Justin Hileman
>>> $u = App\User::where('email','member@gmail.com')->first();
>>> $u->is_verified = 0;
>>> $u->save();
```

Kini, jika kita mencoba login sebagai member, kita akan mendapat pesan berikut:



Middleware untuk mengecek verifikasi user berfungsi

6.3.3 Persiapan Routing

Selanjutnya, kita persiapkan routing yang akan diklik oleh user untuk verifikasi. Untuk sekarang, kita buat skema routingsnya dulu.

Tambahkan baris berikut pada file route:

routes/web.php

```
Route::get('auth/verify/{token}', 'Auth\RegisterController@verify');
```

Pada route ini, kita akan menggunakan method `verify` di `RegisterController`. Untuk sekarang, kita akan buat method kosong terlebih dahulu seperti ini:

app/Http/Controllers/Auth/RegisterController.php

```
public function verify(Request $request, $token) { }
```

6.3.4 Kirim Email Verifikasi ketika Mendaftar

Ketika user mendaftar, kita akan kirim email berisi link verifikasi. Tambahkan baris berikut pada method `create` di `RegisterController`:

app/Http/Controllers/Auth/RegisterController.php

```
protected function create(array $data)
{
    $user = User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
    ]);
    $memberRole = Role::where('name', 'member')->first();
    $user->attachRole($memberRole);
    $user->sendVerification();
    return $user;
}
```

Pada syntax diatas, kita memanggil method `sendVerification` pada model User untuk memproses pengiriman email.

Mari kita buat method ini:

app/User.php

```
...
use Illuminate\Support\Facades\Mail;

class User extends Authenticatable
{
    ...
    public function sendVerification()
    {
        $user = $this;
        $token = str_random(40);
        $user->verification_token = $token;
```

```
$user->save();
Mail::send('auth.emails.verification', compact('user', 'token'), function ($m) use ($user)
) {
    $m->to($user->email, $user->name)->subject('Verifikasi Akun Larapus');
});
}
}
```

Pada syntax diatas, kita membuat token secara acak dengan menggunakan:

```
str_random(40)
```

Fungsi ini merupakan fitur helper di Laravel. Opsi 40 artinya kita akan membuat string acak sepanjang 40 karakter.

Selanjutnya, kita set field `verification_token` dan mengirim email:

```
Mail::send('auth.emails.verification', compact('user', 'token'), function ($m) use ($user) {
    $m->to($user->email, $user->name)->subject('Verifikasi Akun Larapus');
});
```

Di Laravel, untuk mengirim email kita menggunakan fungsi `Mail::send()`. Opsi ini menerima 3 parameter:

1. View yang digunakan sebagai template email
2. Array asosiatif berisi variable yang akan di-*passing* ke template email
3. Sebuah fungsi (closure) yang digunakan untuk menentukan alamat email penerima dan subject email

Pada syntax diatas, kita menggunakan view `auth.emails.verification` dan melakukan passing variable `user` dan `token` dengan fungsi `compact()`. Kita juga mengirim email ke alamat email yang tercatat di database. Terakhir, kita set subject email dengan “Verifikasi Akun Larapus”.

Mari kita buat template emailnya:

resources/views/auth/emails/verification.blade.php

Klik link berikut untuk melakukan aktivasi akun Larapus:

```
<a href="{{ $link = url('auth/verify', $token).'?email='.urlencode($user->email) }}> {{ $lin\\k }} </a>
```

Pada view ini, kita membuat link untuk melakukan verifikasi ke URL /auth/verify/{token}?email={user}.

Sebelum mencoba fitur ini, kita harus mengaktifkan middleware user-should-verified di RegisterController agar dia tidak langsung login ketika berhasil mendaftar.

app/Http/Controllers/Auth/RegisterController.php

```
public function __construct()
{
    $this->middleware('guest');
    $this->middleware('user-should-verified');
}
```

Sip. Cobalah fitur ini dengan mendaftar ke Larapus. Kini, ketika mendaftar kita tidak akan langsung login. Dan pada inbox muncul email berikut:



Admin Larapus
Rahmat Awaludin

4:48 PM

Klik link berikut untuk melakukan aktivasi akun Larapus:

<http://localhost:8000/auth/verify/zWFt0K1NiocwCRvXuEY7ufVkdNWJMSZ1hPXI7KRy?email=rahmat.awaludin%40gmail.com>

Email verifikasi diterima



Cara mengirim email di Laravel

Pada contoh ini kita telah mengirim email dengan cara yang sederhana. Jika jumlah email yang dikirim akan banyak dan logisnya cukup kompleks, saya sarankan Anda mempelajari Mailable di Laravel. Silahkan cek di dokumentasi⁸.

6.3.5 Melakukan Verifikasi

Oke, setelah kita berhasil mengirim email verifikasi, mari kita buat logic pada method verify di RegisterController:

app/Http/Controllers/Auth/RegisterController.php

```
....  
use Illuminate\Support\Facades\Auth;  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Session;  
  
class RegisterController extends Controller  
{  
    ....  
    public function verify(Request $request, $token)  
    {  
        $email = $request->get('email');  
        $user = User::where('verification_token', $token)->where('email', $email)->first();  
        if ($user) {  
            $user->verify();  
            Session::flash("flash_notification", [  
                "level" => "success",  
                "message" => "Berhasil melakukan verifikasi."  
            ]);  
            Auth::login($user);  
        }  
        return redirect('/');  
    }  
}
```

Pada method ini, kita mencari user berdasarkan data token dan email yang digunakan oleh User. Jika ditemukan, kita panggil method verify pada model User, set feedback dan melakukan proses login user tersebut dengan syntax:

⁸<https://laravel.com/docs/5.3/mail#generating-mailables>

```
Auth::login($user);
```

Jika user tidak ditemukan, kita arahkan ke halaman utama.

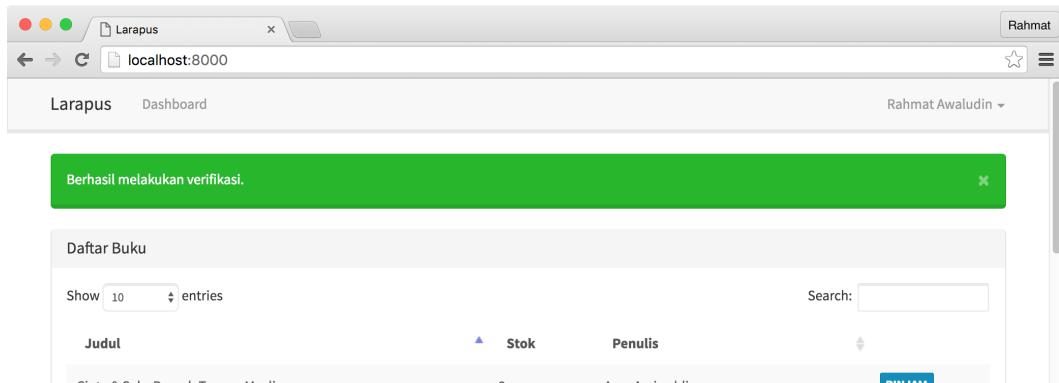
Mari kita buat method verify di model User:

app/User.php

```
public function verify()
{
    $this->is_verified = 1;
    $this->verification_token = null;
    $this->save();
}
```

Untuk melakukan verifikasi kita hanya mengubah isian field `is_verified` menjad 1 dan mengosongkan isi field `verification_token`.

Cobalah klik pada link verifikasi yang telah diterima, kita akan diarahkan ke halaman utama dan mendapat pesan sukses.



Berhasil melakukan verifikasi

Setelah berhasil melakukan verifikasi, cobalah logout dan pastikan berhasil login kembali.

6.3.6 Mengirim Ulang Link Verifikasi

Terkadang link verifikasi yang telah kita kirim terhapus oleh user. Mari kita buat fitur agar user dapat meminta agar Larapust mengirim lagi link untuk verifikasi. Cara kerjanya adalah ketika user gagal untuk login karena belum melakukan aktivasi, kita akan berikan link yang dapat user gunakan untuk meminta link aktivasi lagi.

Mari kita buat dulu routenya:

routes/web.php

```
Route::get('auth/send-verification', 'Auth\RegisterController@sendVerification');
```

Pada route ini kita akan menggunakan menggunakan method `sendVerification` pada `RegisterController`. Mari kita buat method ini:

app/Http/Controllers/Auth/RegisterController.php

```
public function sendVerification(Request $request)
{
    $user = User::where('email', $request->get('email'))->first();
    if ($user && !$user->is_verified) {
        $user->sendVerification();
        Session::flash("flash_notification", [
            "level"=>"success",
            "message"=>"Silahkan klik pada link aktivasi yang telah kami kirim."
        ]);
    }
    return redirect('/login');
}
```

Pada method ini, kita akan menerima input `email` yang akan kita set pada link. Kemudian kita mencari user berdasarkan email tersebut. Jika user ditemukan dan belum diverifikasi, kita panggil kembali method `sendVerification` pada model `User` dan set feedback sukses. Terakhir, kita arahkan user ke halaman login.

Pada logic sebelumnya, pada method `sendVerification` kita selalu membuat token baru. Kita akan ubah logicnya, jika sudah ada token pada field `verification_token` kita akan menggunakan token tersebut. Tambahkan baris berikut pada model `User`:

app/User.php

```

public function generateVerificationToken()
{
    $token = $this->verification_token;
    if (!$token) {
        $token = str_random(40);
        $this->verification_token = $token;
        $this->save();
    }
    return $token;
}

public function sendVerification()
{
    $token = $this->generateVerificationToken();
    $user = $this;
    $token = str_random(40);
    $user->verification_token = $token;
    $user->save();

    Mail::send('auth.emails.verification', compact('user', 'token'), function ($m) use ($user) {
        $m->to($user->email, $user->name)->subject('Verifikasi Akun Larapus');
    });
}

```

Agar link untuk melakukan request verifikasi tampil pada saat user gagal login, kita lakukan perubahan pada method handle di middleware UserShouldVerified:

app/Http/Middleware/UserShouldVerified.php

```

public function handle($request, Closure $next)
{
    $response = $next($request);
    if (Auth::check() && !Auth::user()->is_verified) {
        $link = url('auth/send-verification').'?email='.urlencode(Auth::user()->email);
        Auth::logout();

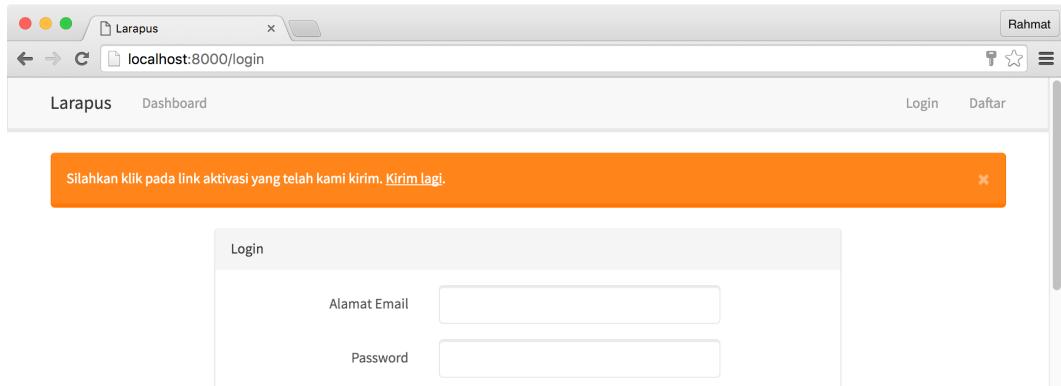
        Session::flash("flash_notification", [
            "level"    => "warning",
            "message" => "Akun Anda belum aktif. Silahkan klik pada link aktivasi yang telah kami kirim."
        ]);
        "message" => "Silahkan klik pada link aktivasi yang telah kami kirim.
        <a class='alert-link' href='$link'>Kirim lagi</a>."
    }
}

```

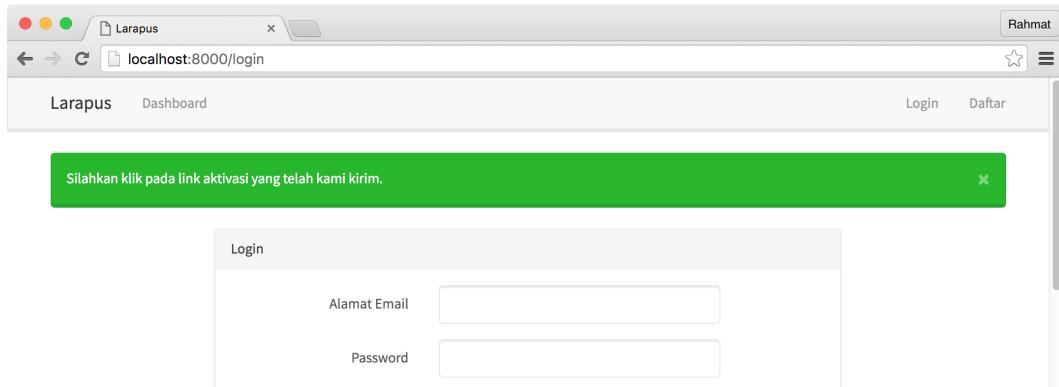
```
]);
return redirect('/login');
}

return $response;
}
```

Untuk mencoba fitur ini, cobalah mendaftar di Larapus. Kemudian lakukan login meskipun belum melakukan verifikasi, akan muncul pesan berikut:



Klik pada link “Kirim lagi”, akan muncul email baru di inbox. Di Larapus akan tampil pesan berikut:



6.4 Pembuatan Halaman Profil

Mari kita buat halaman dimana user dapat melihat profil nya dan melakukan perubahan. Kita buat dulu routenya:

`routes/web.php`

```
Route::get('settings/profile', 'SettingsController@profile');
```

Disini, kita menggunakan method `profile` pada `SettingsController` untuk menampilkan halaman profile. Mari kita buat controller ini:

```
php artisan make:controller SettingsController
```

Pada file yang dihasilkan tambahkan method `profile` seperti berikut:

app/Http/Controllers/SettingsController.php

```
public function profile()
{
    return view('settings.profile');
}
```

Karena URL ini hanya boleh diakses oleh User yang sudah login, kita proteksi SettingsController dengan auth middleware. Tambahkan method __construct pada SettingsController dengan isian berikut:

app/Http/Controllers/SettingsController.php

```
public function __construct()
{
    $this->middleware('auth');
}
```

Kita buat view settings.profile yang akan digunakan method profile:

resources/views/settings/profile.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li class="active">Profil</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Profil</h2>
                    </div>

                    <div class="panel-body">
                        <table class="table">
                            <tbody>
                                <tr>
                                    <td class="text-muted">Nama</td>
```

```

        <td>{{ auth()->user()->name }}</td>
    </tr>
    <tr>
        <td class="text-muted">Email</td>
        <td>{{ auth()->user()->email }}</td>
    </tr>
    </tbody>
</table>
<a class="btn btn-primary" href="#">Ubah</a>
</div>
</div>
</div>
</div>
</div>
@endsection

```

Pada view ini, kita membuat table untuk menampilkan nama dan email user. Kita juga membuat link untuk mengubah profil. URL untuk mengubah profil akan kita isi pada tahap selanjutnya.

Untuk mengakses halaman profil, mari kita tambahkan link pada navigasi:

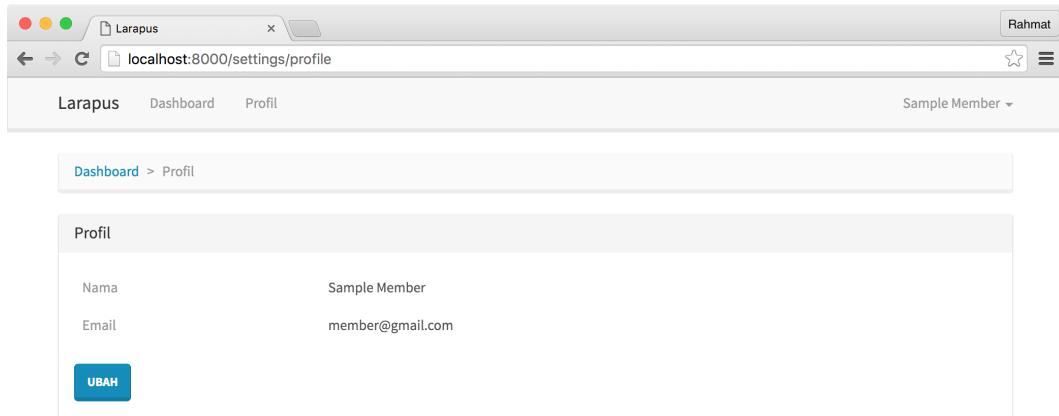
`resources/views/layouts/app.blade.php`

```

.....
<ul class="nav navbar-nav">
@if (Auth::check())
    <li><a href="{{ url('/home') }}>Dashboard</a></li>
@endif
@role('admin')
    <li><a href="{{ route('authors.index') }}>Penulis</a></li>
    <li><a href="{{ route('books.index') }}>Buku</a></li>
@endrole
@if (auth()->check())
    <li><a href="{{ url('/settings/profile') }}>Profil</a></li>
@endif
</ul>
.....

```

Pada syntax ini kita menggunakan `auth()->check()` untuk memastikan agar link navigasi ini hanya muncul ketika user sudah login. Ini tampilan yang akan kita dapatkan:



Halaman Profil

6.4.1 Mengubah Data Profil

Setelah kita berhasil menampilkan profil User, mari kita buat fitur untuk mengubah profil. Buatlah route berikut:

`routes/web.php`

```
Route::get('settings/profile/edit', 'SettingsController@editProfile');
Route::post('settings/profile', 'SettingsController@updateProfile');
```

Pada syntax diatas, kita membuat route `/settings/profile/edit` untuk menampilkan form perubahan profil dan `/settings/profile` dengan method POST untuk menerima perubahan profil.

Kita buat method `editProfile` pada `SettingsController`:

app/Http/Controllers/SettingsController.php

```
public function editProfile()
{
    return view('settings.edit-profile');
}
```

Kita buat view untuk method ini:

resources/views/settings/edit-profile.blade.php

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

@section('content')


- <a href="{{ url('/home') }}>Dashboard</a></li>
- Ubah Profil</li>



## Ubah Profil



{!! Form::model(auth()->user(), ['url' => url('/settings/profile'), 'method' => 'post', 'class'=>'form-horizontal']) !!}



{!! Form::label('name', 'Nama', ['class'=>'col-md-4 control-label']) !!}


{!! Form::text('name', null, ['class'=>'form-control']) !!}
{!! $errors->first('name', '<p class="help-block">:message</p>') !!}



{!! Form::label('email', 'Email', ['class'=>'col-md-4 control-label']) !!}


{!! Form::email('email', null, ['class'=>'form-control']) !!}
{!! $errors->first('email', '<p class="help-block">:message</p>') !!}


```

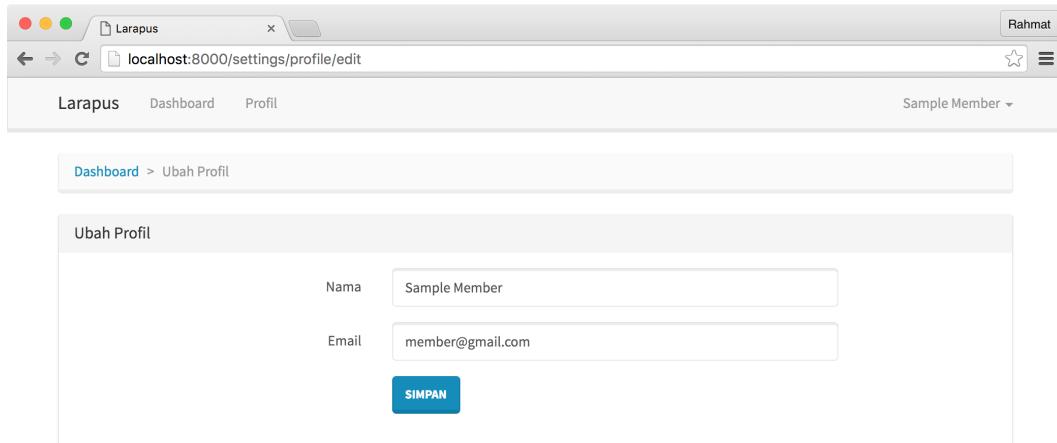
```
<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
    </div>
</div>
{!! Form::close() !!}
</div>
</div>
</div>
</div>
@endsection
```

Pada view ini, kita membuat form untuk mengubah nama dan email. Kita menggunakan Form Model Binding dengan model user yang sedang login agar field name dan email terisi dengan data di database. Form ini kita arahkan ke route /settings/profile dengan method POST yang telah kita buat sebelumnya. Mari kita ubah link “Ubah” di halaman profil agar menuju ke halaman ini:

resources/views/settings/profile.blade.php

```
....  
<a class="btn btn-primary" href="#">Ubah</a>  
<a class="btn btn-primary" href="{{ url('/settings/profile/edit') }}">Ubah</a>  
....
```

Tampilan halaman ini akan seperti berikut:



Halaman ubah profil

Mari kita buat method `updateProfile` pada `SettingsController` untuk menerima form ini:

app/Http/Controllers/SettingsController.php

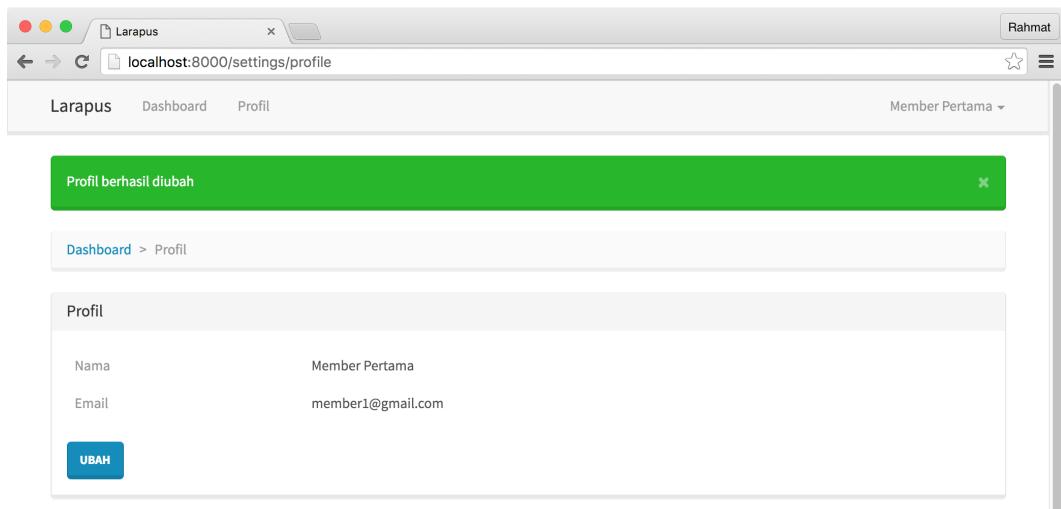
```
....  
use Illuminate\Support\Facades\Auth;  
use Illuminate\Support\Facades\Session;  
  
class SettingsController extends Controller  
{  
    ....  
    public function updateProfile(Request $request)  
    {  
        $user = Auth::user();  
        $this->validate($request, [  
            'name' => 'required',  
            'email' => 'required|unique:users,email,' . $user->id  
        ]);  
  
        $user->name = $request->get('name');  
        $user->email = $request->get('email');  
        $user->save();  
  
        Session::flash("flash_notification", [  
            "level"=>"success",  
            "message"=>"Profil berhasil diubah"  
        ]);  
    }  
}
```

```
        return redirect('settings/profile');
    }
}
```

Pada method ini kita mencari user yang sedang login dan menyimpannya pada variable `$user`. Selanjutnya kita melakukan validasi untuk memastikan field `name` dan `email` diisi oleh user. Kita juga menggunakan rule `unique` pada field `email` untuk memastikan email yang digunakan tidak sama dengan email lain yang telah tercatat di table `users`, kecuali untuk email dengan id yang dimiliki oleh user tersebut.

Setelah data berhasil di validasi, kita update data user, set feedback dan mengarahkan user kembali ke halaman profil.

Cobalah fitur ini, pastikan semua validasi berjalan dan kita berhasil melakukan perubahan data user.



Berhasil mengubah data user

6.4.2 Mengubah Password setelah Login

Fitur lain yang umum dimiliki sebuah website adalah mengubah password setelah user login. Fitur ini, tentunya berbeda dengan fitur lupa password yang telah dimiliki oleh Laravel.

Mari kita buat fitur ini dengan membuat routenya terlebih dahulu:

routes/web.php

```
Route::get('settings/password', 'SettingsController@editPassword');
Route::post('settings/password', 'SettingsController@updatePassword');
```

Pada route ini, kita akan menggunakan URL yang sama untuk menampilkan form untuk mengubah password dan menerima form tersebut.

Kita buat dulu method editPassword untuk menampilkan form ubah password:

app/Http/Controllers/SettingsController.php

```
public function editPassword()
{
    return view('settings.edit-password');
}
```

Dan viewnya:

resources/views/settings/edit-password.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li class="active">Ubah Password</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Ubah Password</h2>
                    </div>

                    <div class="panel-body">
                        {!! Form::open(['url' => url('/settings/password'),
                        'method' => 'post', 'class'=>'form-horizontal']) !!}
                        <div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}>
                            {!! Form::label('password', 'Password lama', ['class'=>'col-md-4 control-label']) <br/>
                            {!! Form::text('password', null, ['class'=>'form-control']) <br/>
                            !!}
                        
```

```

]) !!}
<div class="col-md-6">
    {!! Form::password('password', ['class'=>'form-control']) !!}
    {!! $errors->first('password', '<p class="help-block">:message</p>') !!}
</div>
</div>

<div class="form-group{{ $errors->has('new_password') ? ' has-error' : '' }}">
    {!! Form::label('new_password', 'Password baru', ['class'=>'col-md-4 control-la\
bel']) !!}
    <div class="col-md-6">
        {!! Form::password('new_password', ['class'=>'form-control']) !!}
        {!! $errors->first('new_password', '<p class="help-block">:message</p>') !!}
    </div>
</div>

<div class="form-group{{ $errors->has('new_password_confirmation') ? ' has-error' \
: '' }}">
    {!! Form::label('new_password_confirmation', 'Konfirmasi password baru', ['clas\
s'=>'col-md-4 control-label']) !!}
    <div class="col-md-6">
        {!! Form::password('new_password_confirmation', ['class'=>'form-control']) !!}
        {!! $errors->first('new_password_confirmation', '<p class="help-block">:messag\
e</p>') !!}
    </div>
</div>
<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
    </div>
</div>
{!! Form::close() !!}
</div>
</div>
</div>
</div>
@endsection

```

Pada form ini, kita membuat 3 field; password, new_password dan new_password_confirmation.

Kita membutuhkan field password untuk meminta user mengisi password yang saat ini digunakan. Kita akan meminta user untuk mengetikan password baru nya dua

kali (konfirmasi), itulah sebabnya kita membutuhkan field `new_password_confirmation`.

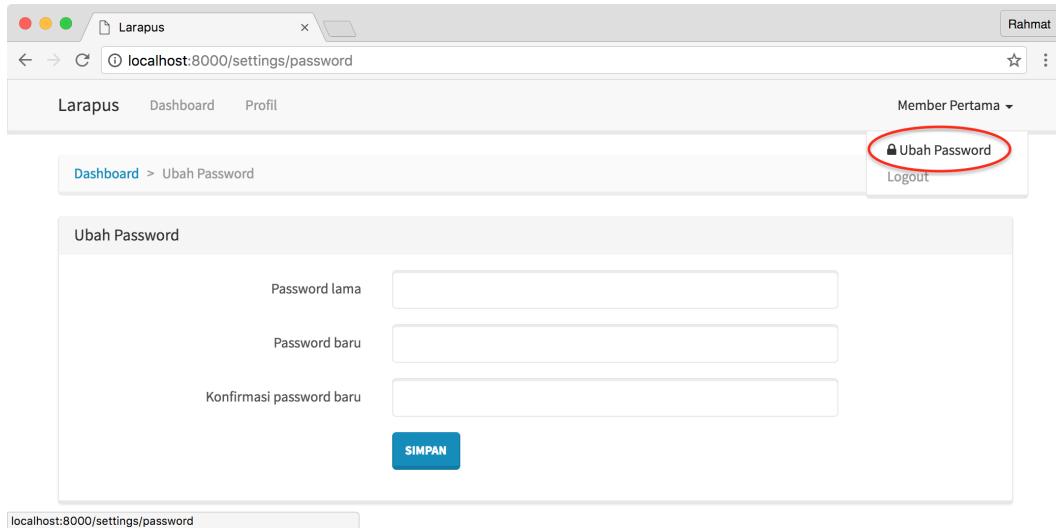
Penambahan `_confirmation` pada nama field merupakan aturan Laravel jika kita hendak menggunakan rule `confirmed`⁹.

Mari kita tambahkan link navigasi untuk mengakses halaman ini:

```
....  
<!-- Right Side Of Navbar -->  
<ul class="nav navbar-nav navbar-right">  
    <!-- Authentication Links -->  
    @if (Auth::guest())  
        ....  
    @else  
        <li class="dropdown">  
            ....  
            <ul class="dropdown-menu" role="menu">  
                <li><a href="{{ url('/settings/password') }}><i class="fa fa-btn fa-lock"></i> Ubah \  
Password</a></li>  
                <li>  
                    ....  
                </li>  
            </ul>  
        </li>  
    @endif  
</ul>  
....
```

Tampilan halaman ini akan seperti berikut:

⁹<https://laravel.com/docs/5.2/validation#rule-confirmed-Paste>



Halaman ubah password

Mari kita buat method updatePassword untuk menerima form ini:

app/Http/Controllers/SettingsController.php

```
public function updatePassword(Request $request)
{
    $user = Auth::user();
    $this->validate($request, [
        'password' => 'required|passcheck:' . $user->password,
        'new_password' => 'required|confirmed|min:6',
    ], [
        'password.passcheck' => 'Password lama tidak sesuai'
    ]);

    $user->password = bcrypt($request->get('new_password'));
    $user->save();

    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=>"Password berhasil diubah"
    ]);

    return redirect('settings/password');
}
```

Pada method ini, kita mencari user yang sedang login dan menyimpannya pada

variable \$user. Selanjutnya kita melakukan validasi untuk field password dan new_password. Pada kedua field tersebut, kita menggunakan rule required agar kedua field tersebut diisi.

Untuk field password, kita juga menggunakan rule passcheck dengan parameter password user saat ini. Rule passcheck tidak dimiliki Laravel, kita akan membuat rule ini untuk memastikan input password user sama dengan password yang tercatat di database pada pembahasan selanjutnya.

Untuk field new_password, kita menggunakan rule confirmed agar isi dari field new_password_confirmation sama dengan field new_password. Kita juga menggunakan rule min:6 agar isi field new_password minimal 6 karakter.

Syntax selanjutnya kita ubah password user, set feedback dan arahkan user ke halaman untuk mengubah password. Ketika mengisi field password, kita harus menggunakan fungsi bcrypt() agar isian password terenkripsi.

Ikuti dulu pembahasan selanjutnya sebelum mencoba fitur ini.

6.4.3 Membuat Rule Validasi Custom

Karena isian field password telah terenkripsi, kita tidak bisa langsung membandingkan input user dengan isi field password. Laravel menyediakan method Hash::check() untuk melakukan perbandingan string biasa dengan string yang telah terenkripsi.

Sebagaimana dijelaskan sebelumnya, kita akan membuat rule passcheck untuk melakukan validasi password lama user. Caranya, tambahkan baris berikut pada method boot di AppServiceProvider:

app/Providers/AppServiceProvider.php

```
....  
use Validator;  
use Hash;  
  
class AppServiceProvider extends ServiceProvider  
{  
    public function boot()  
    {  
        Validator::extend('passcheck', function ($attribute, $value, $parameters) {  
            return Hash::check($value, $parameters[0]);  
        });  
    }  
}
```

```
    });
}
....  
}
```

Untuk membuat rule custom, kita menggunakan method `Validator::extend()`. Method ini menerima 2 parameter; nama rule dan fungsi (atau closure) yang akan menerima 3 parameter (nama field, nilai, parameter rule). Fungsi yang kita passing ke custom rule harus mengembalikan nilai `true` atau `false`.

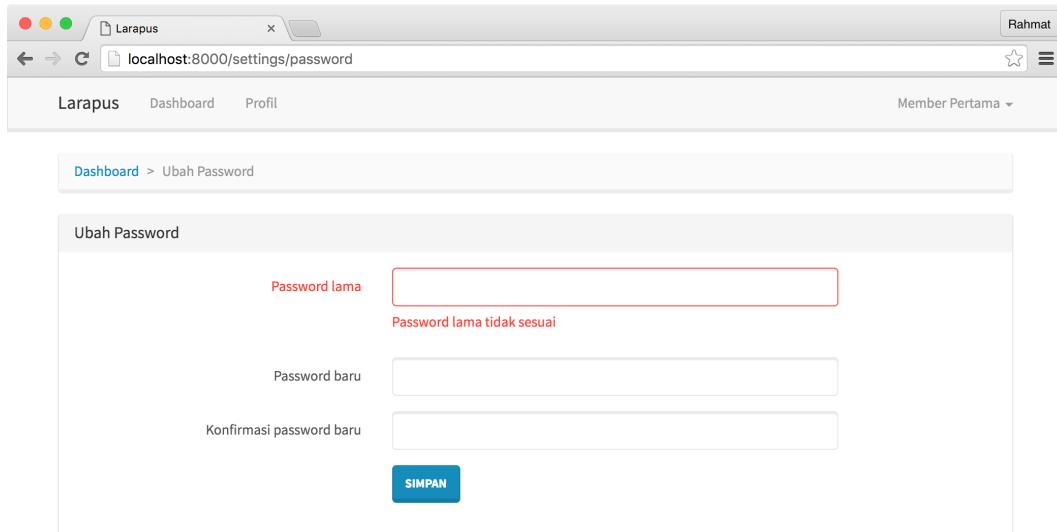
Pada syntax ini, kita menggunakan `Hash::check()` yang sudah kita jelaskan sebelumnya untuk melakukan pengecekan password.

Mari kita lihat lagi bagaimana kita menggunakan rule ini:

```
$this->validate($request, [
    'password' => 'required|passcheck:' . $user->password,
    'new_password' => 'required|confirmed|min:6',
], [
    'password.passcheck' => 'Password lama tidak sesuai'
]);
```

Pada syntax ini, kita passing password user yang ada di database ketika menggunakan rule `passcheck`. Kita juga set pesan error custom untuk rule `passcheck` yaitu “Password lama tidak sesuai”.

Ini tampilan ketika rule ini tidak berhasil di validasi:



Lapus

localhost:8000/settings/password

Rahmat

Lapus Dashboard Profil Member Pertama

Dashboard > Ubah Password

Ubah Password

Password lama

>Password lama tidak sesuai

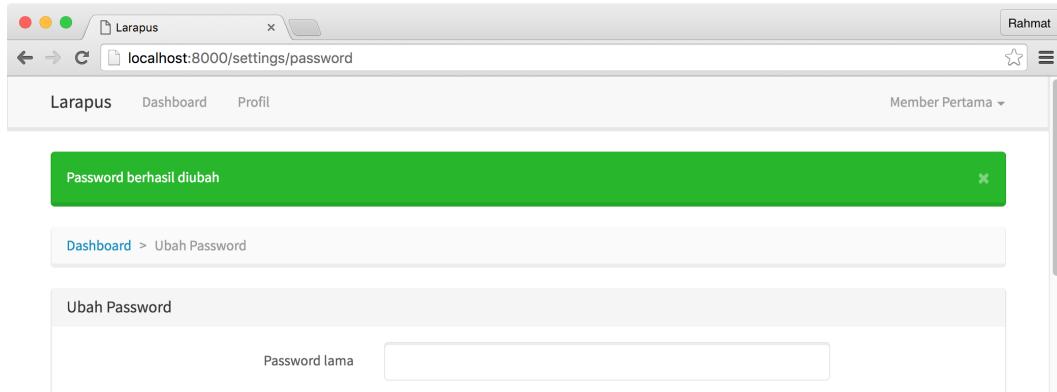
Password baru

Konfirmasi password baru

SIMPAN

Rule passcheck berjalan

Sip, cobalah ubah password user. Pastikan semua validasi berjalan dan berhasil mengubah password user.



Lapus

localhost:8000/settings/password

Rahmat

Lapus Dashboard Profil Member Pertama

Dashboard > Ubah Password

Ubah Password

Password lama

×

Password berhasil diubah

Berhasil mengubah password

6.5 Management Member

Kita akan membuat fitur agar Admin dapat mengatur member. Fitur yang akan kita bangun yaitu:

- Menampilkan semua data member
- Menambah member
- Mengubah data member
- Melihat detail member berikut peminjaman
- Menghapus member

Mari kita buat dulu `MembersController` untuk menyimpan semua logicnya:

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

Siapkan route:

`routes/web.php`

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {
    ...
    Route::resource('members', 'MembersController');
});
```

Kita siapkan juga link navigasi untuk mengakses fitur ini:

resources/views/layouts/app.blade.php

```
....  
@role('admin')  
    <li><a href="{{ route('authors.index') }}>Penulis</a></li>  
    <li><a href="{{ route('books.index') }}>Buku</a></li>  
    <li><a href="{{ route('members.index') }}>Member</a></li>  
@endrole  
....
```

6.5.1 Menampilkan Daftar Member

Mari kita mulai dengan membuat tampilan daftar member. Untuk fitur ini, kita akan menggunakan method `index`. Berikut syntaxnya:

app/Http/Controllers/MembersController.php

```
....  
use App\Role;  
use App\User;  
use Yajra\Datatables\Html\Builder;  
use Yajra\Datatables\Facades\Datatables;  
  
class MembersController extends Controller  
{  
    public function index(Request $request, Builder $htmlBuilder)  
    {  
        if ($request->ajax()) {  
            $members = Role::where('name', 'member')->first()->users;  
            return Datatables::of($members)  
                ->addColumn('action', function($member){  
                    return view('datatable._action', [  
                        'model'          => $member,  
                        'form_url'       => route("members.destroy", $member->id),  
                        'edit_url'       => route('members.edit', $member->id),  
                        'confirm_message' => 'Yakin mau menghapus ' . $member->name . '?'  
                    ]);  
                })->make(true);  
        }  
  
        $html = $htmlBuilder  
            ->addColumn(['data' => 'name', 'name'=>'name', 'title'=>'Nama'])  
            ->addColumn(['data' => 'email', 'name'=>'email', 'title'=>'Email'])  
            ->addColumn(['data' => 'action', 'name'=>'action', 'title'=>'', 'orderable'=>false, 'se\
```

```
archable'=>false]);  
  
    return view('members.index', compact('html'));  
}  
...  
}
```

Pada syntax ini, kita menggunakan DataTables untuk menampilkan daftar member. Untuk mendapatkan semua user dengan akses member, kita menggunakan syntax berikut:

```
$members = Role::where('name', 'member')->first()->users;
```

Arti dari syntax ini adalah kita mencari semua record di table `roles` dengan isian `name` berupa `member`. Kemudian, kita menggunakan `first()` untuk mengambil model pertama dari record yang ditemukan. Dari model tersebut, kita ambil semua user yang berelasi ke role tersebut. Relasi ke `user` dari `Role` telah dibuatkan oleh package Laratrust.

Syntax untuk membuat DataTable kita lakukan seperti biasanya. Disini, kita menampilkan nama, alamat email member yang ditemukan dan kolom action. Terakhir, kita tampilkan view `members.index`. Mari kita buat view ini:

resources/views/members/index.blade.php

```
@extends('layouts.app')  
  
@section('content')  
    <div class="container">  
        <div class="row">  
            <div class="col-md-12">  
                <ul class="breadcrumb">  
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>  
                    <li class="active">Member</li>  
                </ul>  
                <div class="panel panel-default">  
                    <div class="panel-heading">  
                        <h2 class="panel-title">Member</h2>  
                    </div>  
  
                    <div class="panel-body">
```

```

<p> <a class="btn btn-primary" href="{{ url('/admin/members/create') }}">Tambah</\>
a> </p>
    {!! $html->table(['class'=>'table-striped']) !!}
</div>
</div>
</div>
</div>
</div>
</div>
</div>
@endsection

@section('scripts')
{!! $html->scripts() !!}
@endsection

```

Pada view ini, kita menampilkan DataTable yang sudah kita siapkan sebagaimana kita lakukan pada fitur lain. Tampilan dari halaman ini akan seperti berikut:

Nama	Email	
Member Pertama	member@gmail.com	Ubah HAPUS
Rahmat Awaludin	rahmat.awaludin@gmail.com	Ubah HAPUS

Daftar Member

6.5.2 Menambah Member oleh Admin

Untuk membuat fitur ini, berikut logic yang akan kita gunakan:

- Admin hanya mengisi nama dan email
- Password akan dibuat secara acak oleh sistem sebanyak 6 karakter.
- Ketika berhasil membuat user, password akan ditampilkan pada feedback untuk Admin.
- User yang didaftarkan akan menerima email berikut password untuk akunnya.

Mari kita mulai dengan merubah method create untuk menampilkan form pembuatan member:

app/Http/Controllers/MembersController.php

```
public function create()
{
    return view('members.create');
}
```

Struktur untuk view, akan sama dengan proses CRUD yang pernah kita lakukan. Yaitu, menggunakan partial view untuk field formnya.

Buatlah view di resources/views/members/create.blade.php dengan isian:

resources/views/members/create.blade.php

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

@section('content')


- <a href="{{ url('/home') }}>Dashboard</a></li>
- <a href="{{ url('/admin/members') }}>Member</a></li>
- Tambah Member</li>



## Tambah Member



{!! Form::open(['url' => route('members.store'),
'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}


```

```
    @include('members._form')
    {!! Form::close() !!}

```

Dan partial view untuk formnya:

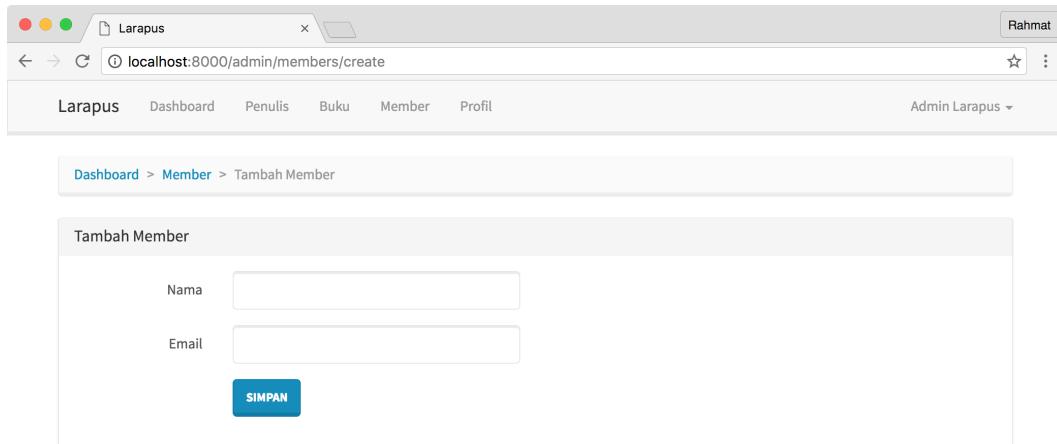
resources/views/members/_form.blade.php

```
<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}>
{!! Form::label('name', 'Nama', ['class'=>'col-md-2 control-label']) !!}
<div class="col-md-4">
{!! Form::text('name', null, ['class'=>'form-control']) !!}
{!! $errors->first('name', '<p class="help-block">:message</p>') !!}
</div>
</div>

<div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}>
{!! Form::label('email', 'Email', ['class'=>'col-md-2 control-label']) !!}
<div class="col-md-4">
{!! Form::email('email', null, ['class'=>'form-control']) !!}
{!! $errors->first('email', '<p class="help-block">:message</p>') !!}
</div>
</div>

<div class="form-group">
<div class="col-md-4 col-md-offset-2">
{!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
</div>
</div>
```

Tampilan formnya akan seperti ini:



Form tambah member

Untuk validasi, kita akan menggunakan form request. Mari kita buat dengan nama `StoreMemberRequest`:

```
php artisan make:request StoreMemberRequest
```

Pada file yang dihasilkan, kita isi dengan syntax berikut:

app/Http/Requests/StoreMemberRequest.php

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreMemberRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'name'  => 'required|max:255',
        ];
    }
}
```

```
    'email' => 'required|email|max:255|unique:users',
];
}
}
```

Pada form request ini, kita langsung memberikan nilai true pada method authorize karena kita sudah menggunakan middleware pada routing. Untuk method rules kita menggunakan rule yang sama di RegisterController untuk field name dan email.

Selanjutnya, kita siapkan method store yang akan menerima form pembuatan member:

MembersController.php

```
...
use App\Http\Requests\StoreMemberRequest;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Facades\Mail;

class MembersController extends Controller
{
    ...
    public function store(StoreMemberRequest $request)
    {
        $password = str_random(6);
        $data = $request->all();
        $data['password'] = bcrypt($password);
        // bypass verifikasi
        $data['is_verified'] = 1;

        $member = User::create($data);

        // set role
        $memberRole = Role::where('name', 'member')->first();
        $member->attachRole($memberRole);

        // kirim email
        Mail::send('auth.emails.invite', compact('member', 'password'), function ($m) use ($member) {
            $m->to($member->email, $member->name)->subject('Anda telah didaftarkan di Larapus!');
        });

        Session::flash("flash_notification", [
            "level"    => "success",
        ]);
    }
}
```

```
    "message" => "Berhasil menyimpan member dengan email " .  
    "<strong>" . $data['email'] . "</strong>" .  
    " dan password <strong>" . $password . "</strong>."  
];  
  
return redirect()->route('members.index');  
}  
}
```

Syntaxnya cukup panjang, mari kita bahas satu-persatu.

Pertama, kita siapkan array data yang berisi detail member.

```
$password = str_random(6);  
$data = $request->all();  
$data['password'] = bcrypt($password);  
// bypass verifikasi  
$data['is_verified'] = 1;
```

Disini, kita set password secara acak dengan fungsi `str_random`. Field `is_verified` langsung kita isi 1 untuk melakukan bypass verifikasi.

Setelah data siap, kita buat member dan set rolenya:

```
$member = User::create($data);  
// set role  
$memberRole = Role::where('name', 'member')->first();  
$member->attachRole($memberRole);
```

Kemudian kita kirim email ke member yang yang baru dibuat. Ketika mengirim email, kita passing member dan passwordnya.

```
Mail::send('auth.emails.invite', compact('member', 'password'), function ($m) use ($member) {  
    $m->to($member->email, $member->name)->subject('Anda telah didaftarkan di Larupus!');  
});
```

Untuk mengirim email, kita harus membuat view di `resources/views/auth/emails/invite.blade.php`. Berikut isinya:

resources/views/auth/emails/invite.blade.php

```
<p>
    Halo {{ $member->name }}.

</p>
<p>
    Admin kami telah mendaftarkan email Anda ({{ $member->email }}) ke Larapus. Untuk login, \
    silahkan kunjungi <a href="{{ $login = url('login') }}>{{ $login }}</a>. Login dengan email \
    Anda dan password <strong>{{ $password }}</strong>.

</p>
>

<p>
    Jika Anda ingin mengubah password, silahkan kunjungi <a href="{{ $reset = url('password/r\ \
    eset') }}>{{ $reset }}</a> dan masukan email Anda.
</p>
```

Terakhir, kita set feedback dan arahkan admin ke halaman index member:

```
Session::flash("flash_notification", [
    "level"    => "success",
    "message"  => "Berhasil menyimpan member dengan email " .
    "<strong>" . $data['email'] . "</strong>" .
    " dan password <strong>" . $password . "</strong>."
]);

return redirect()->route('members.index');
```

Cobalah membuat member. Ketika berhasil, akan muncul tampilan seperti ini:

The screenshot shows a browser window titled 'Larapus' at the URL 'localhost:8000/admin/members'. The top navigation bar includes 'Larapus', 'Dashboard', 'Penulis', 'Buku', 'Member', 'Profil', and a dropdown for 'Admin Larapus'. A green success message box at the top states: 'Berhasil menyimpan member dengan email rahmat.awaludin@gmail.com dan password gmazw0.' Below this, the 'Member' section has a 'TAMBAH' button. The table lists one member: 'Member Pertama' with email 'member@gmail.com'. There are 'Ubah' and 'HAPUS' buttons for this entry. The page also includes a 'Show 10 entries' dropdown and a search input field.

Berhasil membuat member

Kita juga akan mendapatkan email seperti ini:

Anda telah didaftarkan di Larapus!

Admin Larapus, Rahmat Awaludin



Halo Rahmat Awaludin.

Admin kami telah mendaftarkan email Anda (rahmat.awaludin@gmail.com) ke Larapus. Untuk login, silahkan kunjungi <http://localhost:8000/login>. Login dengan email Anda dan password **gmazw0**.

Jika Anda ingin mengubah password, silahkan kunjungi <http://localhost:8000/password/reset> dan masukan email Anda.

Email untuk member

6.5.3 Mengubah Data Member

Untuk mengubah data member, alurnya hampir mirip dengan mengubah data buku. Disini, kita akan menggunakan `UpdateMemberRequest` untuk melakukan validasi. Buatlah file ini dengan isian berikut:

app/Http/Requests/UpdateMemberRequest.php

```
<?php

namespace App\Http\Requests;

class UpdateMemberRequest extends StoreMemberRequest
{
    public function rules()
    {
        $rules = parent::rules();
        $rules['email'] = 'required|unique:users,email,' . $this->route('member');
        return $rules;
    }
}
```

Yang kita lakukan disini adalah meng-*extends* class `StoreMemberRequest` dan mengganti rule untuk field `email` agar bisa menggunakan email yang sekarang digunakan. Selanjutnya, method `update` kita isi dengan syntax berikut:

app/Http/Controllers/MembersController.php

```
.....
use App\Http\Requests\UpdateMemberRequest;

class MembersController extends Controller
{
    ...
    public function update(UpdateMemberRequest $request, $id)
    {
        $member = User::find($id);
        $member->update($request->only('name','email'));
        Session::flash("flash_notification", [
            "level"=>"success",
            "message"=>"Berhasil menyimpan $member->name"
        ]);

        return redirect()->route('members.index');
    }
}
```

Saya rasa syntaxnya cukup jelas. Disini kita mencari member, update field `name` dan `email`, set feedback, dan mengarahkan admin ke halaman index member.

Cobalah melakukan update member, pastikan berhasil dan validasi juga berjalan.

6.5.4 Melihat Detail Peminjaman

Mari kita buat fitur agar admin dapat melihat detail peminjaman yang dilakukan oleh User. Untuk mengakses fitur ini, admin dapat klik nama member dari halaman index member.

Untuk membuat link ini, kita harus meng-*override* kolom name di DataTable seperti ini:

app/Http/Controllers/MembersController.php

```
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $members = Role::where('name', 'member')->first()->users;
        return Datatables::of($members)
            ->addColumn('name', function($member) {
                return '<a href="'.route('members.show', $member->id).'">'.$member->name.'</a>';
            })
            ->addColumn('action', function($member){
                return view('datatable._action', [
                    'model'          => $member,
                    'form_url'       => route('members.destroy', $member->id),
                    'edit_url'       => route('members.edit', $member->id),
                    'confirm_message' => 'Yakin mau menghapus ' . $member->name . '?'
                ]);
            })->make(true);
    }
    ....
}
```

Kini, kolom name akan menjadi link ke method show:

The screenshot shows a Laravel application interface. At the top, there's a header with the title 'Larapus' and a user 'Rahmat'. Below the header, a navigation bar includes 'Dashboard', 'Penulis', 'Buku', 'Member', and 'Profil'. On the right, it says 'Admin Larapus'. The main content area is titled 'Member' and features a table with columns 'Nama' and 'Email'. Two rows are listed: 'Member Pertama' with email 'member@gmail.com' and 'Rahmat Awaludin' with email 'rahmat@awaludin.com'. Each row has 'Ubah' and 'HAPUS' buttons. The first row ('Member Pertama') is circled in red.

Nama	Email		
Member Pertama	member@gmail.com	Ubah	HAPUS
Rahmat Awaludin	rahmat@awaludin.com	Ubah	HAPUS

Mengubah name menjadi link

Kita siapakan method `show` untuk menampilkan data peminjaman:

app/Http/Controllers/MembersController.php

```
public function show($id)
{
    $member = User::find($id);
    return view('members.show', compact('member'));
}
```

Pada method ini kita mencari member berdasarkan ID yang dikirim dan menampilkan view `members.show`. Mari kita buat view nya:

resources/views/members/show.blade.php

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li><a href="{{ url('/admin/members') }}>Member</a></li>
                    <li class="active">Detail {{ $member->name }}</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Detail {{ $member->name }}</h2>
                    </div>

                    <div class="panel-body">
                        <p>Buku yang sedang dipinjam:</p>
                        <table class="table table-condensed table-striped">
                            <thead>
                                <tr>
                                    <td>Judul</td>
                                    <td>Tanggal Peminjaman</td>
                                </tr>
                            </thead>
                            <tbody>
                                @forelse ($member->borrowLogs()->borrowed()->get() as $log)
                                    <tr>
                                        <td>{{ $log->book->title }}</td>
                                        <td>{{ $log->created_at }}</td>
                                    </tr>
                                @empty
                                    <tr>
                                        <td colspan="2">Tidak ada data</td>
                                    </tr>
                                @endforelse
                            </tbody>
                        </table>
                        <p>Buku yang telah dikembalikan:</p>
                        <table class="table table-condensed table-striped">
                            <thead>
                                <tr>
                                    <td>Judul</td>
                                    <td>Tanggal Kembali</td>
                                </tr>
```

```
</thead>
<tbody>
    @forelse ($member->borrowLogs()->returned()->get() as $log)
        <tr>
            <td>{{ $log->book->title }}</td>
            <td>{{ $log->updated_at }}</td>
        </tr>
    @empty
        <tr>
            <td colspan="2">Tidak ada data</td>
        </tr>
    @endforelse
</tbody>
</table>
</div>
</div>
</div>
</div>
</div>
</div>
@endsection
```

Oke, syntaxnya cukup panjang. Yang kita lakukan disini adalah menggunakan relasi borrowLogs pada user untuk mendapatkan data peminjaman. Untuk buku yang masih dipinjam kita menggunakan query scope borrowed(), sedangkan untuk buku yang sudah dikembalikan kita menggunakan query scope returned(). Disini kita juga menggunakan `forelse` untuk memudahkan menampilkan view ketika tidak ada data yang ditemukan untuk di looping.

Tampilan halaman ini untuk data seeder default yang telah kita buat untuk user “member@gmail.com” akan seperti berikut:

The screenshot shows a web application interface for managing members. At the top, there's a header with the title 'Larapus' and a user profile for 'Rahmat'. Below the header, a navigation bar includes links for 'Dashboard', 'Penulis', 'Buku', 'Member', and 'Profil'. A dropdown menu for 'Admin Larapus' is also present. The main content area displays the details of a member named 'Detail Member Pertama'. It shows two sections: 'Buku yang sedang dipinjam:' and 'Buku yang telah dikembalikan:'. Under the first section, there are two entries: 'Kupinang Engkau dengan Hamdalah' (Judul) and '2016-09-08 06:30:03' (Tanggal Peminjaman). Under the second section, there is one entry: 'Jalan Cinta Para Pejuang' (Judul) and '2016-09-08 06:30:03' (Tanggal Kembali).

Halaman Detail Peminjaman

Ketika belum ada buku yang dipinjam, ini tampilan yang akan muncul:

This screenshot shows the same web application interface for managing members. It displays the details for a member named 'Detail Rahmat Awaludin'. In the 'Buku yang sedang dipinjam:' section, it says 'Tidak ada data'. In the 'Buku yang telah dikembalikan:' section, it also says 'Tidak ada data'.

Tidak ada peminjaman



Penggunaan DataTable

Pada tampilan detail dari data peminjaman ini kita hanya menggunakan table sederhana. Ini tidak akan menjadi masalah jika data peminjaman tiap user masih sedikit. Ini salah satu teknik dalam membangun sistem dimana kita tidak terburu-buru membangun fitur yang belum tentu dibutuhkan. Jika kelak ternyata data peminjaman tiap user akan banyak (>50), tentunya kita dapat dengan mudah mengubah tampilan detail peminjaman ini menggunakan DataTable.

6.5.5 Menghapus Member

Kita telah menambahkan tombol untuk menghapus pada langkah sebelumnya. Kini, kita cukup menambahkan logic di method destroy:

app/Http/Controllers/MembersController.php

```
public function destroy($id)
{
    $member = User::find($id);

    if ($member->hasRole('member')) {
        $member->delete();
        Session::flash("flash_notification", [
            "level"=>"success",
            "message"=>"Member berhasil dihapus"
        ]);
    }

    return redirect()->route('members.index');
}
```

Pada method ini kita menghapus User berdasarkan Id yang dikirim. Jika user tersebut adalah member (`hasRole('member')`), maka kita hapus user tersebut dan set feedback. Terakhir, kita arahkan admin kembali ke halaman daftar member.

Cobalah fitur ini, pastikan konfirmasi berjalan dan member berhasil dihapus.

Member berhasil dihapus

Nama	Email	
Member Pertama	member@gmail.com	Ubah HAPUS

Berhasil menghapus member



Menghapus Member ketika Masih Meminjam Buku

Pada syntax kita buat diatas, kita dapat menghapus member meskipun ada buku yang masih dipinjamnya. Cobalah modifikasi proses penghapusan ini agar menghentikan proses penghapusan jika member masih meminjam buku.

6.6 Menampilkan Daftar Peminjaman

Jika tadi kita telah menampilkan data peminjaman tiap user, kini kita akan membuat fitur untuk melihat semua data peminjaman yang ada di sistem. Mari kita buat dulu routenya:

routes/web.php

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {
    ...
    Route::get('statistics', [
        'as'=>'statistics.index',
        'uses'=>'StatisticsController@index'
    ]);
});
```

Untuk mengakses data peminjaman, kita akan menggunakan URL /admin/statistics dan menggunakan method index pada StatisticsController. Kita juga memberikan nama untuk route ini sebagai statistics.index. Mari kita buat controlleranya:

```
php artisan make:controller StatisticsController
```

Pada controller yang dihasilkan, tambahkan method index seperti ini:

app/Http/Controllers/StatisticsController.php

```
...
use Yajra\Datatables\Html\Builder;
use Yajra\Datatables\Facades\Datatables;
use App\BorrowLog;

class StatisticsController extends Controller
{
    public function index(Request $request, Builder $htmlBuilder)
    {
        if ($request->ajax()) {
            $stats = BorrowLog::with('book','user');

            return Datatables::of($stats)
                ->addColumn('returned_at', function($stat){
                    if ($stat->is_returned) {
                        return $stat->updated_at;
                    }
                    return "Masih dipinjam";
                })->make(true);
        }
    }
}
```

```
$html = $htmlBuilder
    ->addColumn(['data' => 'book.title', 'name'=>'book.title', 'title'=>'Judul'])
    ->addColumn(['data' => 'user.name', 'name'=>'user.name', 'title'=>'Peminjam'])
    ->addColumn(['data' => 'created_at', 'name'=>'created_at', 'title'=>'Tanggal Pinjam', '\
searchable'=>false])
    ->addColumn(['data' => 'returned_at', 'name'=>'returned_at', 'title'=>'Tanggal Kembali',
        'orderable'=>false, 'searchable'=>false]);

    return view('statistics.index')->with(compact('html'));
}
}
```

Pada syntax diatas, kita membuat DataTable dengan data dari model BorrowLog. Kita sengaja melakukan eager load untuk relasi ke book dan user:

```
$stats = BorrowLog::with('book', 'user');
```

karena kita akan menampilkan judul buku dan nama member.

Field yang kita tampilkan adalah judul, peminjam, tanggal pinjam dan tanggal kembali. Khusus untuk field Tanggal Kembali kita membuat custom field returned_at karena kita hendak menampilkan tulisan “Masih dipinjam” jika buku belum dikembalikan.

Kita buat viewnya:

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

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

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <ul class="breadcrumb">
                    <li><a href="{{ url('/home') }}>Dashboard</a></li>
                    <li class="active">Data Peminjaman</li>
                </ul>
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h2 class="panel-title">Data Peminjaman</h2>
                    </div>
```

```
<div class="panel-body">
    {!! $html->table(['class'=>'table-striped']) !!}
</div>
</div>
</div>
</div>
</div>
@endsection

@section('scripts')
    {!! $html->scripts() !!}
@endsection
```

Terakhir, tambahkan link navigasi untuk fitur ini:

resources/views/layouts/app.blade.php

```
.....
@role('admin')
    <li><a href="{{ route('authors.index') }}">Penulis</a></li>
    <li><a href="{{ route('books.index') }}">Buku</a></li>
    <li><a href="{{ route('members.index') }}">Member</a></li>
    <li><a href="{{ route('statistics.index') }}">Peminjaman</a></li>
@endrole
....
```

Tampilan yang akan kita dapatkan untuk sample data yang telah kita buat akan seperti berikut:

Judul	Peminjam	Tanggal Pinjam	Tanggal Kembali
Jalan Cinta Para Pejuang	Member Pertama	2016-09-08 06:30:03	Masih dipinjam
Kupinang Engkau dengan Hamdalah	Member Pertama	2016-09-08 06:30:03	Masih dipinjam
Membingkai Surga dalam Rumah Tangga	Member Pertama	2016-09-08 06:30:03	2016-09-08 06:30:03

Data Peminjaman

6.7 Filter Buku Berdasarkan Status Peminjaman

Akan lebih bermanfaat bagi Admin jika pada fitur data peminjaman kita dapat melakukan filter berdasarkan status peminjaman. Mari kita buat fitur ini dengan menambahkan dropdown untuk melakukan filter berdasarkan status sudah dikembalikan atau masih dipinjam.

Untuk membuat fitur ini, kita akan menggunakan event `preXhr.dt10` di DataTable. Menggunakan event ini, kita bisa menambahkan parameter baru saat DataTable melakukan request ke server.

Tambahkan baris berikut pada view `statistics.index`:

¹⁰<https://datatables.net/reference/event/preXhr>

resources/views/statistics/index.blade.php

```
@section('scripts')
{!! $html->scripts() !!}
<script>
$(function() {
  $('\
    <div id="filter_status" class="dataTables_length" style="display: inline-block; margin-left:10px;">\
      <label>Status \
        <select size="1" name="filter_status" aria-controls="filter_status" \
          class="form-control input-sm" style="width: 140px;">\
          <option value="all" selected="selected">Semua</option>\
          <option value="returned">Sudah Dikembalikan</option>\
          <option value="notReturned">Belum Dikembalikan</option>\
        </select>\
      </label>\
    </div>\
  ).insertAfter('.dataTables_length');

  $("#dataTableBuilder").on('preXhr.dt', function(e, settings, data) {
    data.status = $('select[name="filter_status"]').val();
  });

  $('select[name="filter_status"]').change(function() {
    window.LaravelDataTables["dataTableBuilder"].ajax.reload();
  });
});
</script>
@endsection
```

Ada tiga bagian dari syntax diatas. Pertama, kita menambah dropdown untuk memilih status peminjaman di samping “Show x entries”:

```
$(``  
    <div id="filter_status" class="dataTables_length" style="display: inline-block; margin-left\  
:10px;">\n        <label>Status \  
        <select size="1" name="filter_status" aria-controls="filter_status" \  
            class="form-control input-sm" style="width: 140px;">\n            <option value="all" selected="selected">Semua</option>\n            <option value="returned">Sudah Dikembalikan</option>\n            <option value="notReturned">Belum Dikembalikan</option>\n        </select>\n    </label>\n</div>\n`).insertAfter('.dataTables_length');
```

Bagian kedua, menggunakan event `preXhr.dt` kita menambahkan parameter baru dengan nama `status` saat `DataTable` melakukan request. Parameter `status` akan berisi nilai yang kita pilih pada dropdown untuk memilih status yaitu `all`, `returned` atau `notReturned`.

```
$("#dataTableBuilder").on('preXhr.dt', function(e, settings, data) {  
    data.status = $('select[name="filter_status"]').val();  
});
```

Bagian terakhir, kita akan membuat `DataTable` otomatis melakukan reload ketika kita mengubah isian status:

```
($('select[name="filter_status"]').change(function() {  
    window.LaravelDataTables["dataTableBuilder"].ajax.reload();  
});
```

Di sisi server, kita lakukan perubahan berikut:

app/Http/Controllers/StatisticsController.php

```
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $stats = BorrowLog::with('book', 'user');
        if ($request->get('status') == 'returned') $stats->returned();
        if ($request->get('status') == 'notReturned') $stats->borrowed();

        return Datatables::of($stats)
            ->addColumn('returned_at', function($stat){
                if ($stat->isReturned) {
                    return $stat->updated_at;
                }
                return "Masih dipinjam";
            })->make(true);
    }

    ....
}
```

Pada perubahan diatas, kita menambahkan query scope `returned()` ketika parameter `status` yang dikirim berupa `returned` dan scope `borrowed()` ketika `status` yang dikirim `notReturned`.

Cobalah fitur ini, pastikan dapat melakukan filter pencarian data peminjaman berdasarkan status peminjaman.

The screenshot shows a web browser window titled 'Larapus' with the URL 'localhost:8000/admin/statistics'. The top navigation bar includes links for 'Larapus', 'Dashboard', 'Penulis', 'Buku', 'Member', 'Peminjaman', and 'Profil', along with a dropdown for 'Admin Larapus'. Below the navigation is a breadcrumb trail: 'Dashboard > Data Peminjaman'. The main content area is titled 'Data Peminjaman' and displays a table of loan records. The table has columns: 'Judul', 'Peminjam', 'Tanggal Pinjam', and 'Tanggal Kembali'. The data shows three entries: 'Jalan Cinta Para Pejuang' borrowed by 'Member Pertama' on '2016-09-08 06:30:03' and still 'Masih dipinjam'; 'Kupinang Engkau dengan Hamdalah' borrowed by 'Member Pertama' on '2016-09-08 06:30:03' and still 'Masih dipinjam'; and 'Membingkai Surga dalam Rumah Tangga' borrowed by 'Member Pertama' on '2016-09-08 06:30:03' and returned on '2016-09-08 06:30:03'. A search bar and a pagination control with '1' are visible at the bottom of the table.

Judul	Peminjam	Tanggal Pinjam	Tanggal Kembali
Jalan Cinta Para Pejuang	Member Pertama	2016-09-08 06:30:03	Masih dipinjam
Kupinang Engkau dengan Hamdalah	Member Pertama	2016-09-08 06:30:03	Masih dipinjam
Membingkai Surga dalam Rumah Tangga	Member Pertama	2016-09-08 06:30:03	2016-09-08 06:30:03

Filter Data Peminjaman berdasarkan Status

6.8 Ringkasan

Pada hari 6 ini kita telah melengkapi fitur Admin dari Larapus. Beberapa hal yang telah kita pelajari yaitu :

- Penggunaan reCaptcha
- Pembuatan custom rule untuk validasi
- Konfigurasi dan pengiriman Email dengan Mailgun
- Custom Filter pada Datatable

Pada hari 7, kita akan belajar mengupload Larapus ke shared hosting. Semangat! :)

7. Hari 7 : Deploy Aplikasi

Di hari terakhir ini, kita akan mengupload Larapus yang telah kita buat ke web. Ada banyak cara untuk mengupload aplikasi yang dibangun dengan Laravel ke internet. Diantaranya ke shared hosting, fortrabbit, VPS, dll. Bahkan untuk memudahkan deployment Laravel menyediakan fitur berbayar bernama [forge](#)¹ yang akan memudahkan deployment aplikasi di Linode, DigitalOcean, AWS, & Rackspace.

Pada pembahasan di buku ini, saya akan menunjukkan bagaimana mengupload Laravel ke shared hosting. Jika ingin mengikuti pembahasan di hari ini, silahkan membeli hosting dari berbagai provider hosting. Saya tidak akan merekomendasikan hosting yang akan dibeli. Sebagai gambaran, saat buku ini ditulis, saya mendapatkan domain `larapus.web.id` berikut shared hosting untuk satu tahun seharga 88rb di sebuah hosting di Indonesia.

Ketika membeli paket hosting, pastikan server pada hosting tersebut mendukung requirement untuk Laravel yaitu PHP versi 5.6.4 keatas dengan ekstensi OpenSSL, PDO, Mbstring dan Tokenizer. Agar fungsi file upload berjalan, pastikan juga server sudah mengaktifkan ekstensi `php_fileinfo`.

Pada pembahasan ini, topik dasar misalnya cara setup software ftp untuk mengupload file ke server tidak akan kita bahas. Kita akan lebih fokus pada pembahasan khusus untuk Laravel.

7.1 Mengupload File

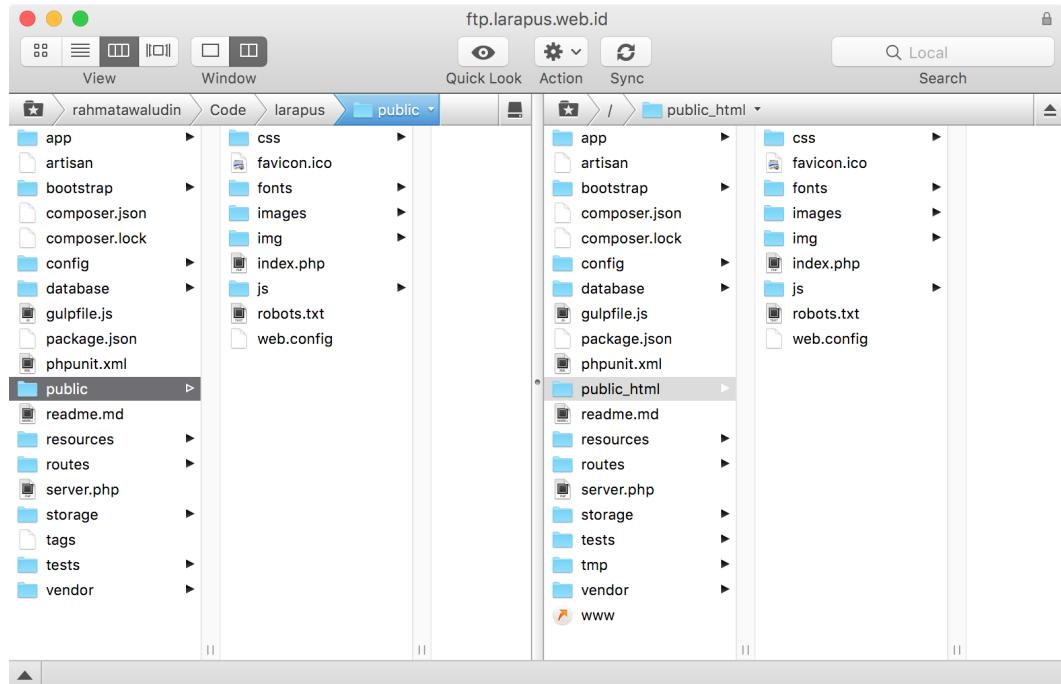
Pada sebuah shared hosting, biasanya *document root* dari domain akan diarahkan ke folder `public_html`. Yang perlu kita lakukan adalah:

1. Upload isi folder `public` ke folder `public_html`
2. Upload folder dan file selain folder `public` ke folder utama.

¹<https://forge.laravel.com/>

3. Pastikan file .env dan public/.htaccess juga diupload (biasanya terlewat karena kedua file ini di *hidden*).

Hasil akhir dari proses upload ini akan seperti berikut:



Hasil upload

Ukuran project Laravel ini akan cukup besar, jika kita upload menggunakan ftp biasanya akan cukup lama. Salah satu teknik yang dilakukan adalah dengan mengkompress project laravel misalnya ke format .zip, upload dan ekstrak di server.

7.2 Konfigurasi public_path

Karena kini lokasi folder public kita rubah, kita harus memberitahu Laravel agar menggunakan folder yang kita gunakan. Jika kita tidak melakukan ini, fungsi `public_path()` akan menghasilkan folder public bukan `public_html`. Caranya, tambahkan baris berikut pada file `public_html/index.php`:

public_html/index.php

```
....  
$app = require_once __DIR__.'/..../bootstrap/app.php';  
  
$app->bind('path.public', function() {  
    return __DIR__;  
});  
....
```

7.3 Konfigurasi Database

Pada kebanyakan shared hosting, kita tidak dapat mengakses ssh untuk menjalankan berbagai perintah artisan. Dengan keterbatasan ini, kita harus mengeksport dan import database manual menggunakan phpmyadmin. Berikut langkah-langkah yang harus kita lakukan:

1. Eksport database di server local
2. Buat user dan database dari halaman cPanel
3. Import database
4. Ubah isian DB_DATABASE, DB_USERNAME dan DB_PASSWORD di file .env.

7.4 Konfigurasi Captcha

Jika pada saat mendaftar captcha sebelumnya belum mendaftarkan captcha, buatlah sekarang. Tambahkan domain yang sekarang digunakan pada isian Domains:

Key Settings

Label
larapus

Domains
(one per line)
larapus.web.id

Owners
(one per line)
rahmat.awaludin@gmail.com

▶ Advanced Settings

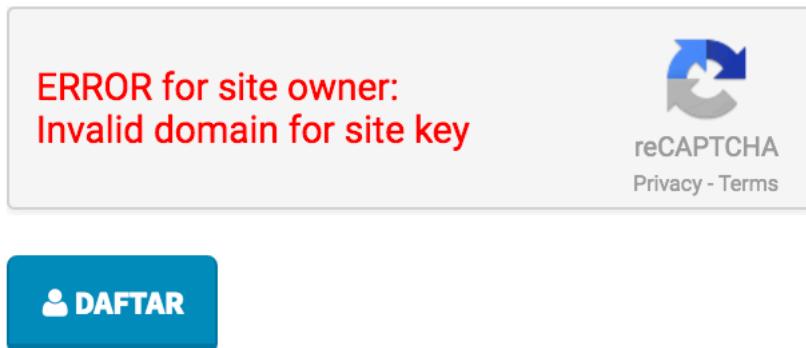
Send alerts to owners [?](#)

Delete Key

Discard changes **Save changes**

Menambah domain ke captcha

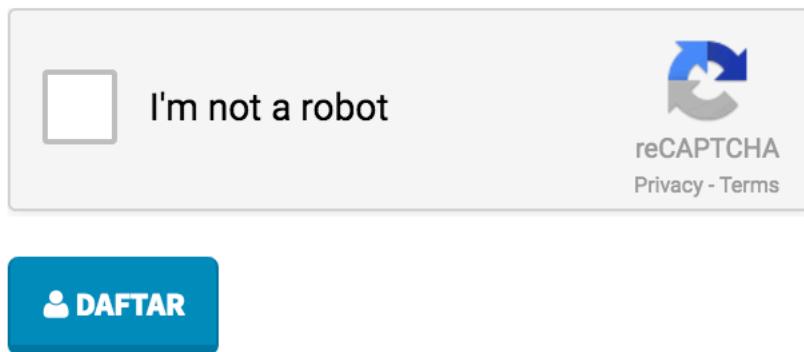
Biasanya perubahan ini akan berefek setelah 30 menit. Jika menemukan tampilan ini:



Delay captcha

berarti perubah captcha masih diproses oleh Google.

Jika tampilannya seperti ini:



Berhasil menambah domain ke captcha

berarti telah berhasil menambah domain.

7.5 Verifikasi Mailgun

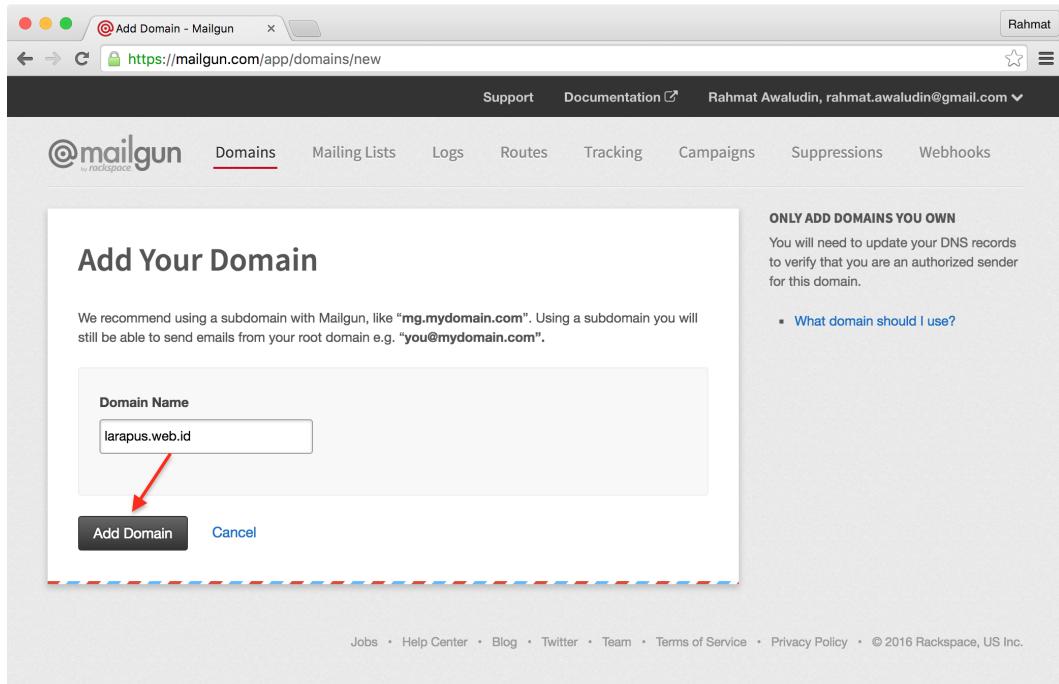
Pada pembahasan di hari sebelumnya, kita masih menggunakan fitur sandbox di Mailgun. Kini, mari kita gunakan domain yang sudah kita beli.

Pada halaman dashboard Mailgun, pilih menu “Domains” dan klik pada “Add New Domain”:

The screenshot shows the Mailgun Domains dashboard. At the top, there's a navigation bar with links for Support, Documentation, and a user account (Rahmat, rahmat.awaludin@gmail.com). Below the navigation is a main menu with options: Domains (which is underlined and has a red arrow pointing to it), Mailing Lists, Logs, Routes, Tracking, Campaigns, Suppressions, and Webhooks. The main content area is titled "Domains". It features a search bar at the top right labeled "Search 1 domains". On the left, there are filter options: Active (1), Unverified (0), and Disabled (0). The main table lists one domain: "sandbox73825.mailgun.org" which is "Active". The table includes columns for State, Domain Name, Outgoing (30d), Bounced, and Complaints. The "Outgoing" column shows 10 messages sent, 0.0% bounced, and 0.0% complaints. The "Bounced" and "Complaints" columns show 0. The table has a dashed red border at the bottom. A large black button labeled "Add New Domain" is centered above the table.

Menambah domain baru

Pada tampilan selanjutnya, isi “Domain name” dengan nama domain yang telah kita beli. Kemudian klik pada “Add Domain”:



The screenshot shows a web browser window for Mailgun's domain management interface. The URL is <https://mailgun.com/app/domains/new>. The page title is "Add Domain - Mailgun". The main content area has a heading "Add Your Domain". Below it, a note says: "We recommend using a subdomain with Mailgun, like "mg.mydomain.com". Using a subdomain you will still be able to send emails from your root domain e.g. "you@mydomain.com".". There is a "Domain Name" input field containing "larapus.web.id", which is highlighted with a red arrow pointing to the "Add Domain" button below it. Other buttons include "Cancel". To the right, there is a sidebar with the heading "ONLY ADD DOMAINS YOU OWN" and a note about updating DNS records. A link "What domain should I use?" is also present.

Menambah domain baru

Pada tampilan selanjutnya, akan tampil instruksi untuk merubah dan menambah DNS Records yang dibutuhkan. Isian disini akan berbeda dengan yang tampil pada akun saya.

The screenshot shows a web browser window with the URL <https://mailgun.com/app/domains/larapus.web.id/verify>. The page displays a success message: "Success! Your domain larapus.web.id was created." Below this, a section titled "Now Follow These Steps To Verify Your Domain" lists three steps:

- 1. Go To Your DNS Provider**

Go to the DNS provider that you use to manage larapus.web.id and add the following DNS records.

Common providers include [GoDaddy](#), [NameCheap](#), [Network Solutions](#), [Rackspace Email & Apps](#), [Rackspace Cloud DNS](#) and [Amazon Route 53](#).
- 2. Add DNS Records For Sending**

TXT records (known as SPF & DKIM) are required to send email through Mailgun.

Type	Hostname	Enter This Value
TXT	larapus.web.id	v=spf1 include:mailgun.org ~all
TXT	pic._domainkey.larapus.web.id	keras; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCaxU+hasifrUEFSaYNayjPKtL UWpfBrGsu7wCbutPF3krbZ4k0iX383/sYhNbPgENkswQH7P0d9dhoH2NnUVbF8TbZ fOduGHH0wUeoUj9X1h2mIXDe3DI+ITS3jz5skGxsU8e5DElmEzt2ON4Xa/HhNrxFZNE CVEW9FdBSwiDAQAB
- 3. Add DNS Records For Tracking**

Petunjuk merubah DNS

4. Add DNS Records For Receiving (Optional)

MX records are **required to receive email**. Unless you already have MX records for `@larapus.web.id` pointing to another email provider (e.g. Gmail), you should update the following records. [More info on MX records ↗](#)

Type	Priority	Enter This Value
MX	10	<code>mx.a.mailgun.org</code>
MX	10	<code>mx.b.mailgun.org</code>

5. Wait For Your Domain To Verify

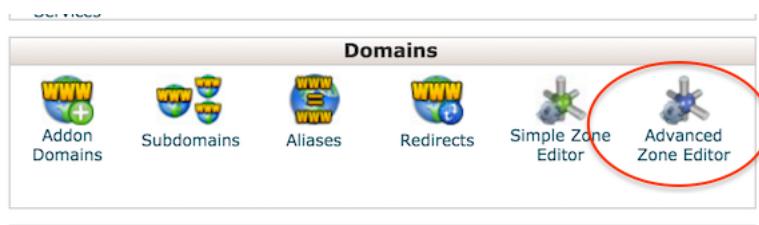
Once you make the above DNS changes it **can take 24-48hrs** for those changes to propagate. We will email you to let you know once your domain is verified.

[Continue To Domain Overview](#)

Jobs • Help Center • Blog • Twitter • Team • Terms of Service • Privacy Policy • © 2016 Rackspace, US Inc.

Petunjuk merubah DNS

Untuk melakukan perubahan ini, bukalah menu “Advanced Zone Editor” di cPanel:



Advanced Zone Editor

Kemudian, tambahkan DNS records yang dibutuhkan. Ini perubahan yang saya lakukan untuk domain larapus.web.id:

larapus.web.id.	14400	IN	TXT	v=spf1 include:mailgun.org ~all	 Edit	 Delete

Perubahan DNS

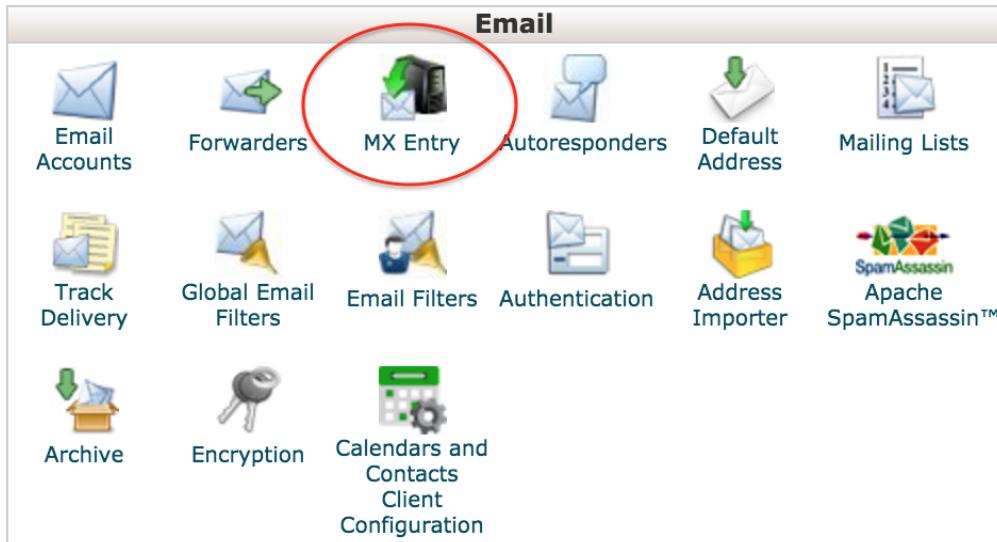
pic._domainkey.larapus.web.id.	14400	IN	TXT	k=rsa; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADC BiQKBgQCaxU+hasifrUEFSaYNayjPKtLUWpfBrGs u7wCbutPF3krbZ4k0lX383/sYihNbPgENkswQHf7 P0d9idhloH2NnUVbF8TbzfOduGHH0wUeoUj9X1h2 mIXDe3DI+ITSjJz5skGxsU8e5DEImEzt2ON4Xa /HhNrxFNECVIEW9FdBSwIDAQAB	 Edit	 Delete

Perubahan DNS

email.larapus.web.id.	14400	IN	CNAME	mailgun.org	 Edit	 Delete

Perubahan DNS

Pada petunjuk yang saya dapatkan, ada juga instruksi untuk menambah MX Records. Untuk melakukan perubahan ini, bukalah menu MX Entry di cPanel:



MX Entry

Pada domain yang saya gunakan ini perubahan yang dibutuhkan:

Add New Record

Priority:

Destination:

[Add New Record](#)

MX Records

Priority	Destination	Actions
10	mxa.mailgun.org	 Edit Remove
10	mxb.mailgun.org	 Edit Remove

Perubahan MX Entry

Setelah semua perubahan diatas dilakukan, klik pada tombol “Continue to Domain Overview” di Mailgun. Sesuai penjelasan di halaman ini, perubahan ini biasanya akan berefek setelah 24-48 jam.

5. Wait For Your Domain To Verify

Once you make the above DNS changes it can take 24-48hrs for those changes to propagate. We will email you to let you know once your domain is verified.



Continue to Domain Overview

Pada halaman selanjutnya, kita dapat juga meminta Mailgun untuk mengecek perubahan DNS secara manual dengan klik pada tombol “Check DNS Records Now” pada menu “Domain Verification and DNS”.

Domain Verification & DNS

Check DNS Records Now

Add DNS Records For Sending

TXT records (known as SPF & DKIM) are required to send email through Mailgun.

Type	Hostname	Enter This Value	Current Value
<input checked="" type="checkbox"/> TXT	larapus.web.id	v=spf1 include:mailgun.org ~all	v=spf1 include:mailgun.org ~all
<input checked="" type="checkbox"/> TXT	pic._domainkey.larapus.web.id	k=rsa; p=MIGfMA0GCSqGSIb3DQEB AQUAA4GNADCBiQKBgQCax U+hasifrUEFSaYNayjPKtLUW pfBrGsu7wCbutPF3krbZ4k0i	k=rsa; p=MIGfMA0GCSqGSIb3DQEB AQUAA4GNADCBiQKBgQCax U+hasifrUEFSaYNayjPKtLUW pfBrGsu7wCbutPF3krbZ4k0i

Cek DNS Manual

Jika domain ini sudah aktif, kita akan melihat status “active” pada menu “Domain Information”.

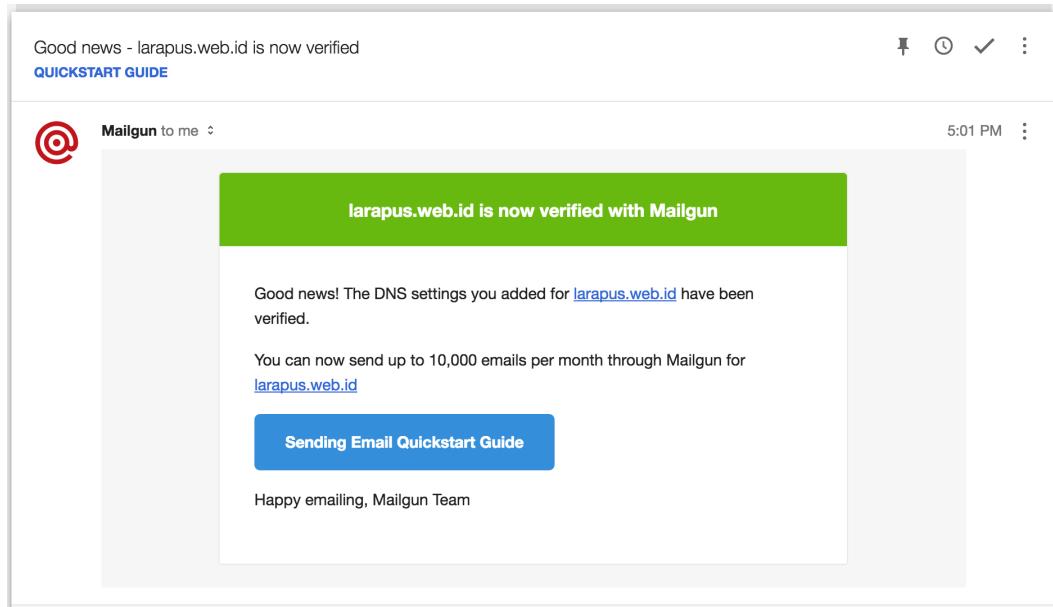
The screenshot shows the Mailgun 'Domain Information' page for the domain `larapus.web.id`. The page header indicates the domain is active. Below the header, there are several configuration settings:

- State:** Active (highlighted with a red circle)
- IP Address:** 184.173.153.204
- SMTP Hostname:** smtp.mailgun.org
- Default SMTP Login:** postmaster@larapus.web.id
- Default Password:** d9df7c279f5cce424376d10be5d24167 · [Manage SMTP credentials](#)
- API Base URL:** https://api.mailgun.net/v3/larapus.web.id (highlighted with a red box)
- API Key:** key-8mklbhy-w1b9zpn7h1y8as2fymtp5z59

Domain sudah aktif

Pada tampilan ini, kita juga bisa mendapatkan API Key yang akan digunakan.

Ketika domain sudah aktif, kita juga akan mendapat email berikut:



Pemberitahuan domain aktif

Kita juga dapat menambah akun email baru dari Mailgun. Pada menu “Domain Information”, klik pada “Manage SMTP Credentials”. Pada tampilan yang muncul, buatlah alamat email baru yang akan kita gunakan untuk mengirim email dari Larapus:

Domains /larapus.web.id /SMTP Credentials

SMTP Credentials For larapus.web.id

New SMTP Credential Delete Selected

		Date Created
<input type="checkbox"/>	Login	
<input type="checkbox"/>	admin@larapus.web.id	04/15/16 06:04 AM
<input type="checkbox"/>	postmaster@larapus.web.id	04/15/16 05:48 AM

Menambah alamat email

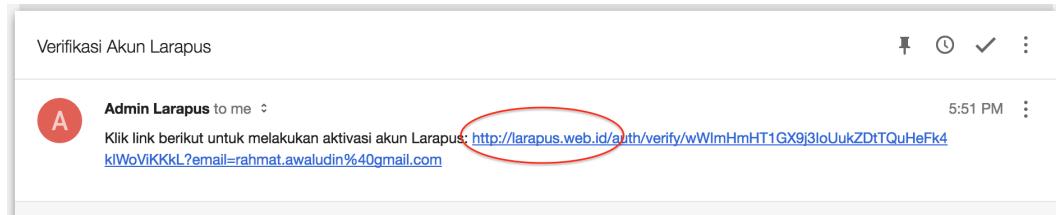
Oke, perubahan di Mailgun sudah cukup. Di Larapus, kita harus merubah isian `from` pada `config/mail.php` sesuai data alamat email diatas. Misalnya seperti berikut:

config/mail.php

```
'from' => ['address' => 'admin@larapus.web.id', 'name' => 'Admin Larapus'],
```

Pada file `.env`, ubah isian `MAILGUN_DOMAIN` dengan nama domain yang kita daftarkan dan `MAILGUN_SECRET` dengan API Key yang kita dapatkan.

Untuk mengetesnya, kita dapat mencoba mendaftar ke Larapus yang sudah berada di shared hosting. Pastikan mendapat email dari Larapus dan email tersebut tidak masuk ke folder spam.



Berhasil verifikasi mailgun

7.6 Nonaktifkan Debug

Langkah terakhir adalah mengubah isian APP_ENV menjadi production dan APP_DEBUG menjadi false pada file .env. Ini perlu kita lakukan agar Laravel tidak menampilkan detail error ketika ada error di aplikasi. Jika ingin melihat detail error, dapat dilihat di file storage/logs/laravel.log.

7.7 Ringkasan

Teknik deployment yang kita lakukan ini adalah yang paling sederhana dan murah. Jika hendak mengembangkan aplikasi dengan skala besar, saya sarankan untuk mempelajari penggunaan VPS. Jika malas melakukan setup server VPS, saya sarankan untuk mempelajari beberapa penyedia layanan Saas diantaranya fortrabbit, box, heroku, dll. Tentunya, untuk menggunakan beberapa penyedia layanan Saas, pemahaman tentang GIT sangat diperlukan.

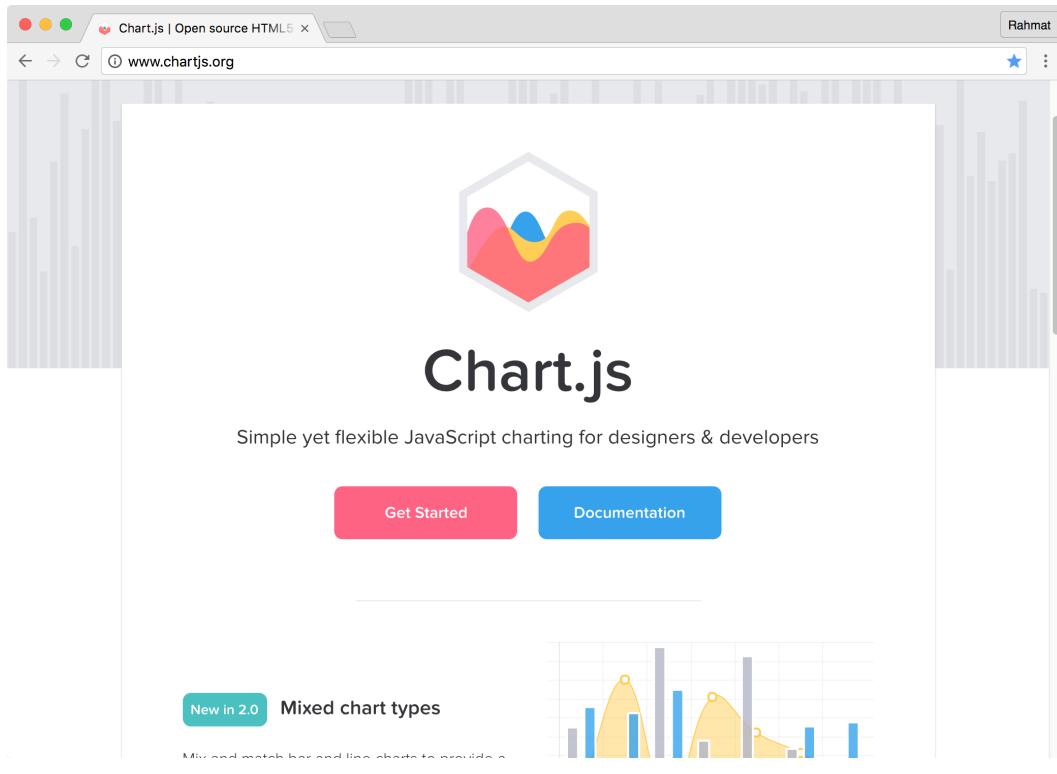
8. Bonus

Setiap orang suka sama yang namanya bonus, pada bab ini, saya menambahkan materi-materi yang direquest oleh pembaca. Anda punya request materi ? Tinggal email saya di rahmat.awaludin@gmail.com :)

8.1 Membuat Chart

Dalam membangun dashboard aplikasi biasanya dibutuhkan chart untuk menampilkan data terkini dari aplikasi secara menyeluruh. Untuk membuat chart ini kita akan menggunakan library chartjs yang bisa didapat dari <http://chartjs.org>¹.

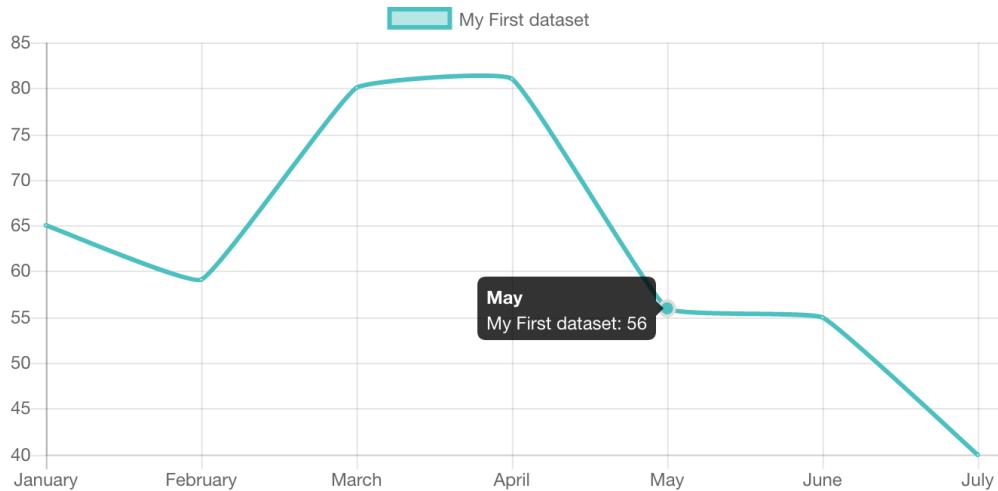
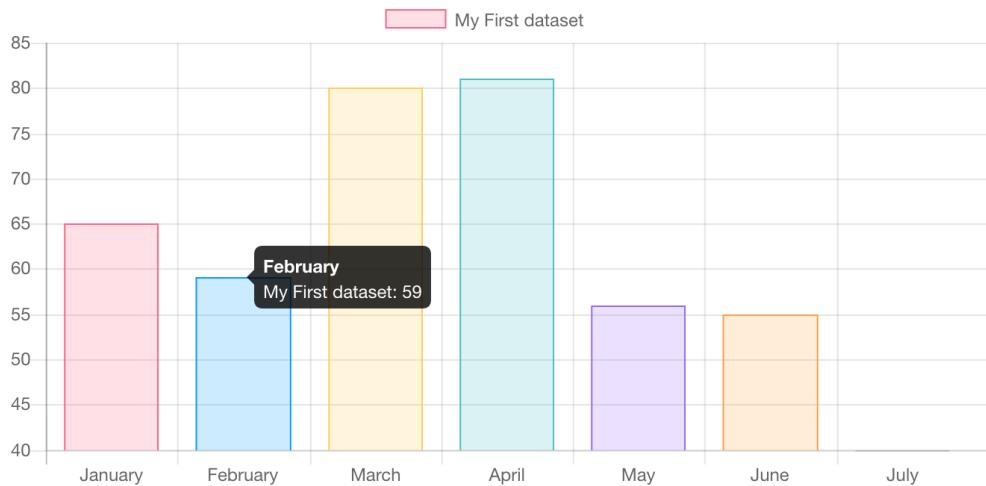
¹<http://chartjs.org>

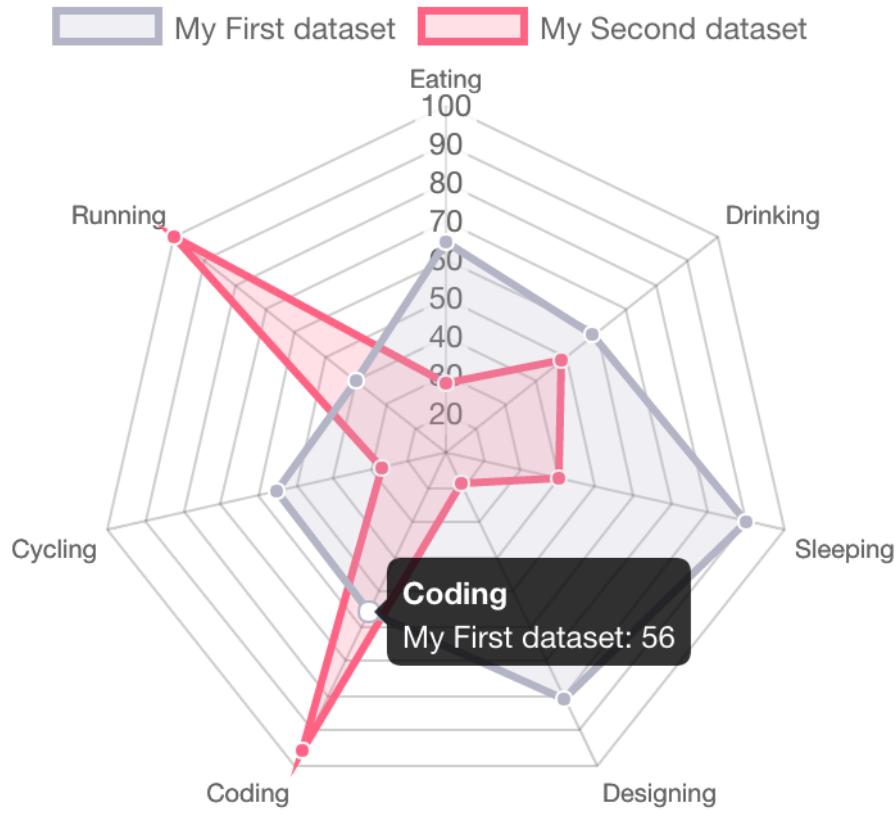


ChartJS mendukung berbagai jenis chart diantaranya:

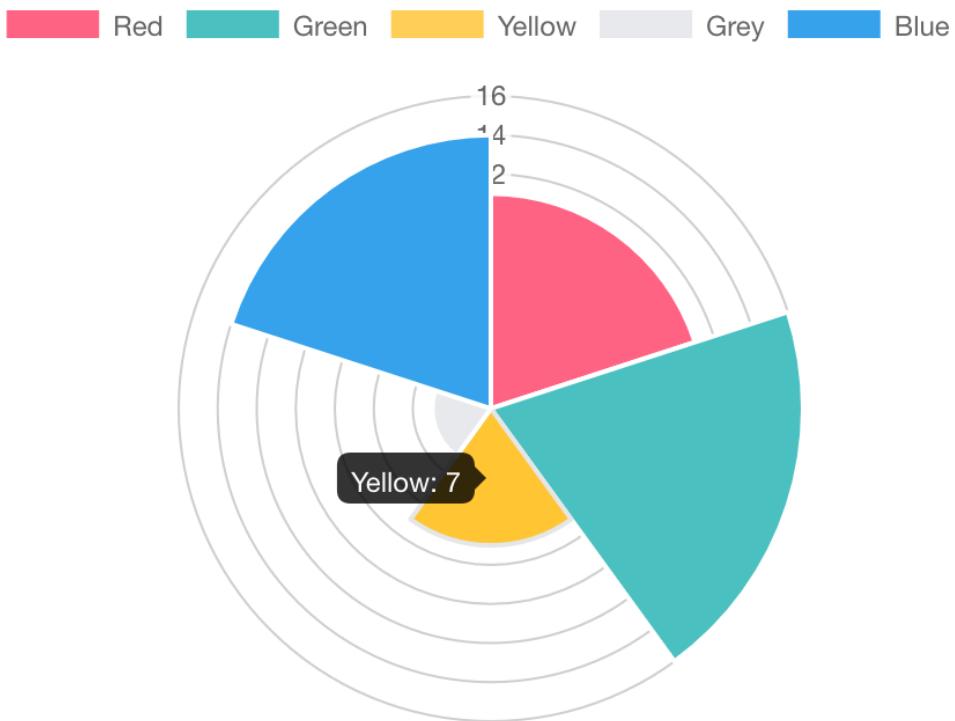
- Line
- Bar
- Radar
- Polar Area
- Pie & Doughnut
- Bubble
- Scales

Berikut contoh tampilan masing-masing jenis chart yang bisa dibuat :

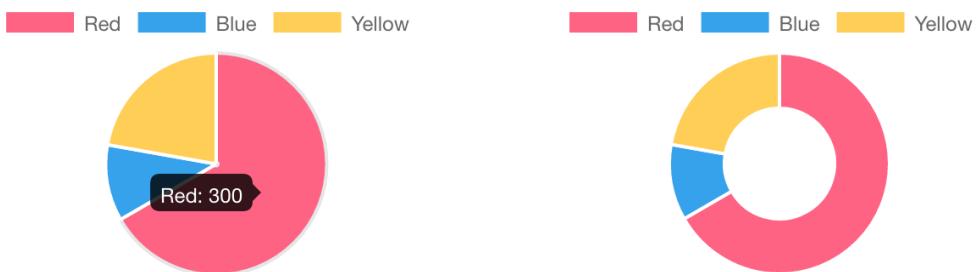
**Line Chart****Bar Chart**



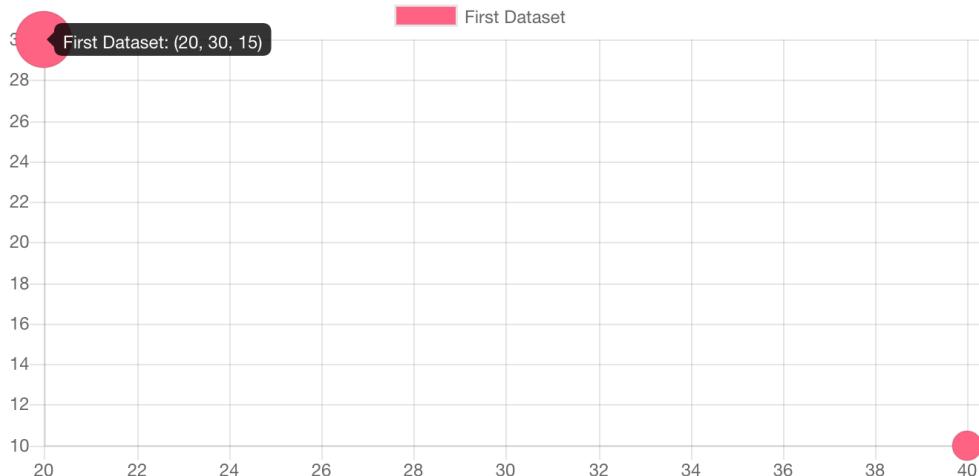
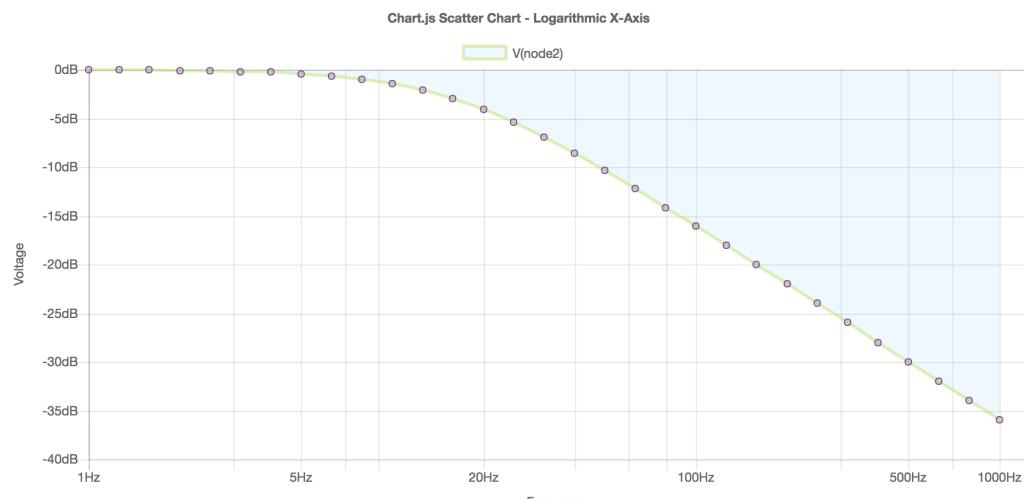
Radar Chart



Polar Area Chart



Pie & Doughnut Chart

**Bubble Chart****Scales Chart**

Kita akan menggunakan bar chart untuk menampilkan grafik data jumlah buku tiap penulis. Chart ini akan berisi nama penulis dan jumlah buku yang ada di Larapus. Silahkan ikuti langkah berikut.

Download dulu chartjs di [github²](#) simpan di public/js/Chart.min.js (di buku ini menggunakan versi 2.2.2) atau bisa juga copy dari sample source code.

Untuk membuat chart, kita membutuhkan dua buah array. Array ini akan digunakan sebagai sumbu X dan sumbu Y pada chart. Pada sumbu X kita akan menggunakan data penulis. Sedangkan pada sumbu Y kita akan menampilkan jumlah buku dari penulis tersebut. Chart ini akan kita tampilkan hanya pada halaman admin. Ubahlah method adminDashboard pada HomeController menjadi:

app/Http/Controllers/HomeController.php

```
1 ....
2 use App\Author;
3
4 class HomeController extends Controller
5 {
6 ....
7 protected function adminDashboard()
8 {
9     $authors = [];
10    $books = [];
11    foreach (Author::all() as $author) {
12        array_push($authors, $author->name);
13        array_push($books, $author->books->count());
14    }
15
16    return view('dashboard.admin', compact('authors', 'books'));
17    //return view('dashboard.admin');
18 }
19 }
```

Pada method ini, kita membuat array bernama authors berisi nama penulis dan books berisi total buku oleh penulis tersebut. Kedua array ini kita passing ke view untuk mengatur sumbu x dan sumbu y.

Pada view untuk dashboard admin, kita tambahkan syntax berikut:

²<https://github.com/chartjs/Chart.js/releases/download/v2.2.2/Chart.min.js>

resources/views/dashboard/admin.blade.php

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

@section('content')
.....
<div class="panel-body">
Selamat datang di Menu Administrasi Larapus. Silahkan pilih menu administrasi yang diinginkan.
<hr>
<h4>Statistik Penulis</h4>
<canvas id="chartPenulis" width="400" height="150"></canvas>
</div>
.....
@endsection

@section('scripts')
<script src="/js/Chart.min.js"></script>
<script>
var data = {
    labels: {!! json_encode($authors) !!},
    datasets: [{{
        label: 'Jumlah buku',
        data: {!! json_encode($books) !!},
        backgroundColor: "rgba(151,187,205,0.5)",
        borderColor: "rgba(151,187,205,0.8)",
    }}]
};

var options = {
    scales: {
        yAxes: [{{
            ticks: {
                beginAtZero:true,
                stepSize: 1
            }
        }}]
    }
};

var ctx = document.getElementById("chartPenulis").getContext("2d");

var authorChart = new Chart(ctx, {
    type: 'bar',
    data: data,
    options: options
});
});
```

```
</script>
@endsection
```

Mari kita bahas apa yang terjadi di syntax ini. Pertama, kita harus meload library chartjs di halaman ini. Untuk melakukan ini, kita menggunakan section scripts yang telah kita buat sebelumnya.

```
<script src="/js/Chart.min.js"></script>
```

Selanjutnya, kita siapkan canvas berukuran 400x150 untuk menyimpan chart yang akan kita buat. Canvas ini memiliki id chartPenulis.

```
<canvas id="chartPenulis" width="400" height="150"></canvas>
```

Untuk melakukan konfigurasi chartjs, kita membuat json bernama data.

```
var data = {
  labels: {!! json_encode($authors) !!},
  datasets: [{
    label: 'Jumlah buku',
    data:>{!! json_encode($books) !!},
    backgroundColor: "rgba(151,187,205,0.5)",
    borderColor: "rgba(151,187,205,0.8)",
  }]
};
```

Pada json data ini, kita membuat key `labels` berisi nama penulis. Key `labels` akan digunakan untuk sumbu x pada chart. Key `datasets` kita gunakan untuk melakukan konfigurasi tampilan chart. Pada key `datasets` juga terdapat key `data` yang akan kita gunakan untuk mengatur sumbu y pada chart. Di sini juga kita membuat konfigurasi untuk pewarnaan chart, lebih lengkap mengenai konfigurasi yang bisa digunakan cek dokumentasi chartjs³.

Kita sengaja menggunakan fungsi `json_encode()` pada array `authors` dan `books` agar array tersebut berubah menjadi json. Selain data, kita juga menyiapkan konfigurasi lain untuk chart pada variable `options`:

³<http://www.chartjs.org/docs/#getting-started>

```
var options = {
  scales: {
    yAxes: [
      ticks: {
        beginAtZero:true,
        stepSize: 1
      }
    ]
  }
};
```

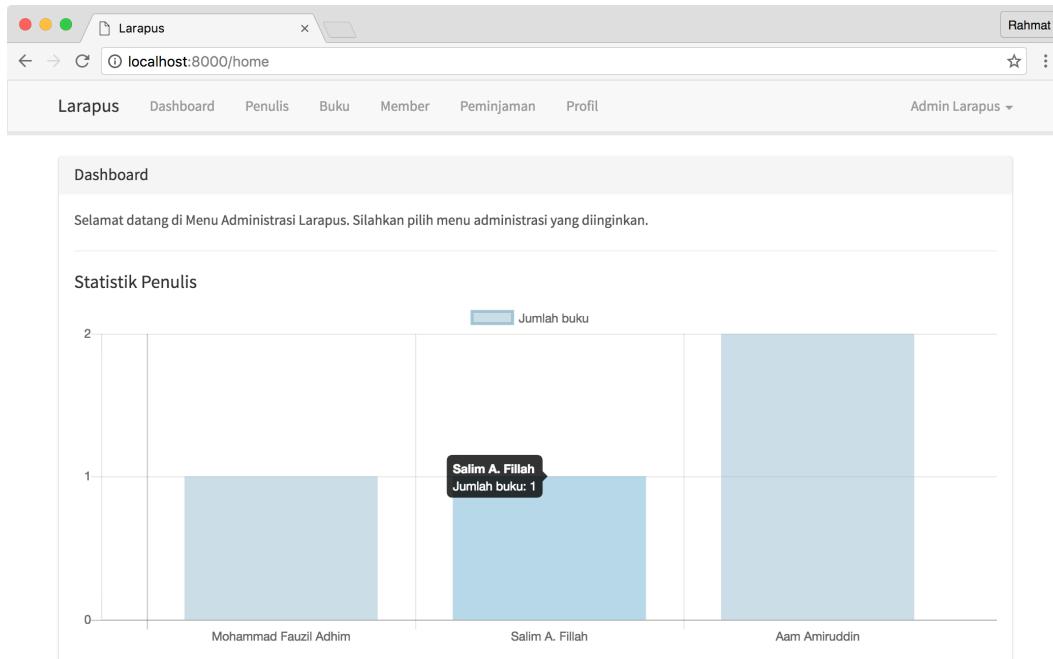
Selanjutnya, kita menentukan canvas yang akan digunakan untuk membuat chart. Tentunya, kita akan menggunakan id `chartPenulis` yang telah kita buat sebelumnya.

```
var ctx = document.getElementById("chartPenulis").getContext("2d");
```

Terakhir, kita inisialisasi chart.

```
var authorChart = new Chart(ctx, {
  type: 'bar',
  data: data,
  options: options
});
```

Setelah semua syntax diatas ditambahkan, loginlah sebagai Admin. Chart berikut akan muncul di dashboard.



Dashboard Admin dengan Chart

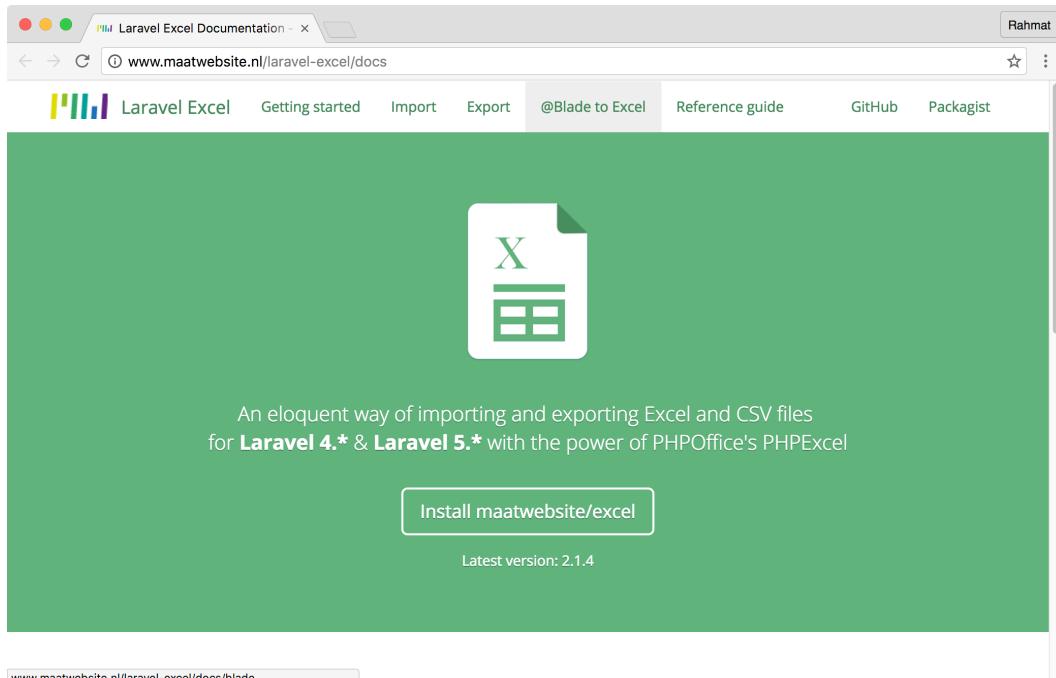
Gampang kan? Sebagai tantangan buatlah chart jenis lain misalnya pie chart, radar chart, dll dengan data yang telah ada.

8.2 Export Data ke Excel

Export data sering menjadi kebutuhan dalam aplikasi. Biasanya fitur ini dibutuhkan untuk backup data. Di bagian ini kita akan belajar bagaimana melakukan export data dari Laravel ke Excel.

Data yang diexport pada contoh ini adalah data buku. Pada saat export, kita juga akan membuat filter data. Filtering data hanya dilakukan berdasarkan penulis buku. Tentunya, kedepannya sangat memungkinkan untuk menambah filtering yang lain misalnya, buku yang sedang dipinjam saja, buku yang tidak sedang dipinjam, buku yang stoknya > 2, dll. Untuk melakukan export data ke excel, kita akan menggunakan library [Laravel Excel⁴](#). Library ini merupakan implementasi PHPExcel untuk Laravel.

⁴<http://www.maatwebsite.nl/laravel-excel/docs>



www.maatwebsite.nl/laravel-excel/docs/blade

Laravel Excel

Mari kita install dulu package ini dengan perintah:

```
composer require maatwebsite/excel=~2.1
```

Setelah proses download selesai, tambahkan syntax berikut pada file config/app.php:

config/app.php

```
....  
'providers' => [  
....  
Maatwebsite\Excel\ExcelServiceProvider::class,  
],  
....  
'aliases' => [  
....  
'Excel' => Maatwebsite\Excel\Facades\Excel::class,  
],
```

Kita siapkan route untuk menampilkan halaman export dan memproses export.

routes/web.php

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {  
    ...  
    Route::get('export/books', [  
        'as' => 'export.books',  
        'uses' => 'BooksController@export'  
    ]);  
    Route::post('export/books', [  
        'as' => 'export.books.post',  
        'uses' => 'BooksController@exportPost'  
    ]);  
});
```

Pada route ini kita menggunakan url /admin/export/books untuk menampilkan halaman export dan memprosesnya. Kita menggunakan method export dan exportPost pada BooksController untuk menghandle kedua route tersebut.

Mari kita buat method kosong untuk kedua route ini.

app/Http/Controllers/BooksController.php

```
public function export() {}  
public function exportPost(Request $request) {}
```

Pada method export, kita tampilkan viewnya:

app/Http/Controllers/BooksController.php

```
public function export()  
{  
    return view('books.export');  
}
```

Berikut isian untuk view books.export:

resources/views/books/export.blade.php

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

@section('content')


- <a href="{{ url('/home') }}>Dashboard</a></li>
- <a href="{{ url('/admin/books') }}>Buku</a></li>
- Export Buku</li>



## Export Buku



{!! Form::open(['url' => route('export.books.post'), 'method' => 'post', 'class'=>'form-horizontal']) !!}


{!! Form::label('author_id', 'Penulis', ['class'=>'col-md-2 control-label']) !!}


{!! Form::select('author_id[]', [''=>'']+App\Author::pluck('name','id')->all(), \null, [
'class'=>'js-selectize',
'multiple',
'placeholder' => 'Pilih Penulis']) !!}
{!! $errors->first('author_id', '<p class="help-block">:message</p>') !!}



{!! Form::submit('Download', ['class'=>'btn btn-primary']) !!}


{!! Form::close() !!}



---


```

Pada view ini, kita menampilkan element `<select>` untuk memilih penulis yang datanya ingin di export ke Excel. Kita juga menggunakan class `js-selectize` agar menggunakan library `selectize` untuk menampilkan field ini.

Agar kita dapat memilih beberapa penulis, kita menggunakan nama `author[]` dan menambahkan opsi `multiple`. Kita juga menggunakan `[' '=> ' ']+App\Author::pluck('name', 'id')`, untuk menampilkan array assosiatif berisi id dan nama penulis. Array `[' '=> ' ']` perlu kita tambahkan agar opsi `placeholder` berjalan. Sehingga akan muncul tulisan “Pilih Penulis” saat belum ada penulis yang dipilih.

Mari kita tambahkan link ke halaman export dari halaman index buku.

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

```
....  
<div class="panel-body">  
<p>  
  <a class="btn btn-primary" href="{{ url('/admin/books/create') }}>Tambah</a>  
  <a class="btn btn-primary" href="{{ url('/admin/export/books') }}>Export</a>  
</p>  
{!! $html->table(['class'=>'table-striped']) !!}  
</div>  
....
```

Pada halaman index buku, kini akan terdapat link berikut:

Bonus

Lapusus Dashboard Penulis Buku Member Peminjaman Profil Admin Larapus

Dashboard > Buku

Buku

TAMBAH **EXPORT**

Show 10 entries Search:

Judul	Jumlah	Penulis	Action
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	Ubah HAPUS
Jalan Cinta Para Pejuang	2	Salim A. Fillah	Ubah HAPUS
Kupinang Engkau dengan Hamdalah	3	Mohammad Fauzil Adhim	Ubah HAPUS
Membingkai Surga dalam Rumah Tangga	4	Aam Amiruddin	Ubah HAPUS

Showing 1 to 4 of 4 entries PREVIOUS 1 NEXT

Link ke export buku

Dan tampilan halaman export akan seperti berikut:

Bonus

Lapusus Dashboard Penulis Buku Member Peminjaman Profil Admin Larapus

Dashboard > Buku > Export Buku

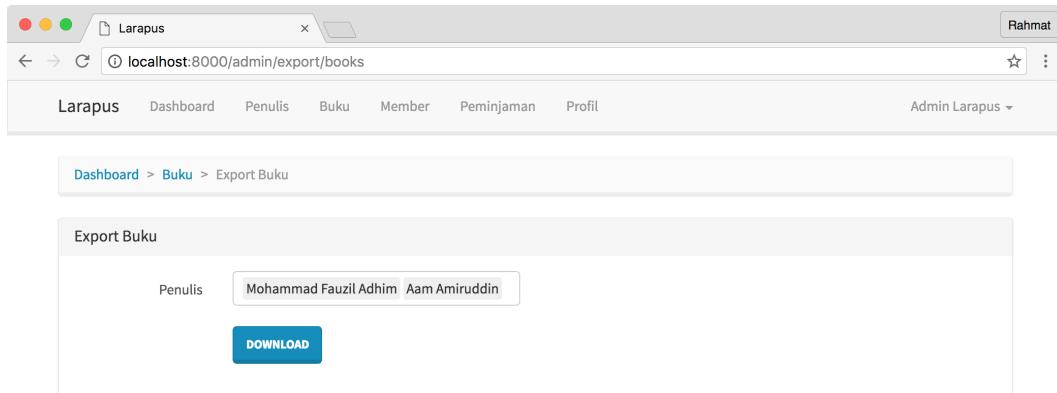
Export Buku

Penulis

DOWNLOAD

Halaman export

Pastikan bisa memilih beberapa penulis seperti berikut:



Memilih beberapa penulis

Mari kita isi syntax di method `exportPost` untuk memproses form ini.

app/Http/Controllers/BooksController.php

```
....  
use Excel;  
  
class BooksController extends Controller  
{  
    ....  
    public function exportPost(Request $request)  
    {  
        // validasi  
        $this->validate($request, [  
            'author_id'=>'required',  
        ], [  
            'author_id.required'=>'Anda belum memilih penulis. Pilih minimal 1 penulis.'  
        ]);  
  
        $books = Book::whereIn('id', $request->get('author_id'))->get();  
  
        Excel::create('Data Buku Larapus', function($excel) use ($books) {  
            // Set property  
            $excel->setTitle('Data Buku Larapus')  
                ->setCreator(Auth::user()->name);  
  
            $excel->sheet('Data Buku', function($sheet) use ($books) {  
                $row = 1;  
                $sheet->row($row, [  
                    'Judul',
```

```
        'Jumlah',
        'Stok',
        'Penulis'
    ]);
foreach ($books as $book) {
    $sheet->row(++$row, [
        $book->title,
        $book->amount,
        $book->stock,
        $book->author->name
    ]);
}
});
})->export('xls');
}
}
```

Syntax ini cukup panjang, mari kita bahas perbagian. Pertama, kita melakukan validasi untuk memastikan user memilih penulis untuk diexport.

```
$this->validate($request, [
    'author_id'=>'required',
], [
    'author_id.required'=>'Anda belum memilih penulis. Pilih minimal 1 penulis.'
]);
```

Disini, kita juga membuat pesan error custom ketika user tidak memilih penulis.

Selanjutnya, kita mencari semua buku berdasarkan author_id yang diterima:

```
$books = Book::whereIn('id', $request->get('author_id'))->get();
```

Query `whereIn` menerima dua parameter; nama field dan array berisi nilai yang diizinkan. Disini kita hanya mencari buku yang memiliki `author_id` yang sesuai dengan id dari semua penulis yang dipilih.

Selanjutnya, kita menggunakan facade `Excel` untuk membuat file excel dan memberikan response berupa download file `xls`.

```
Excel::create('Data Buku Larapus', function($excel) use ($books) {  
    ...  
})->export('xls');
```

Disini kita menggunakan method `create` untuk membuat file excel dan method `export` untuk memberikan response berupa download file. Pada method `create`, kita menggunakan nama file “Data Buku Larapus”. Kita juga melakukan passing data buku yang telah kita dapatkan pada query sebelumnya.

Mari kita bahas apa yang terjadi pada method `create` ketika kita membuat file excel. Pertama, kita membuat property dari file excel yang akan dibuat:

```
$excel->setTitle('Data Buku Larapus')  
->setCreator(Auth::user()->name);
```

Disini kita membuat judul sama dengan nama file yaitu “Data Buku Larapus” dan nama author dari file excel tersebut berdasarkan nama user yang sedang login.

Selanjutnya, kita membuat *sheet* baru pada file excel yang dibuat.

```
$excel->sheet('Data Buku', function($sheet) use ($books) {  
    ...  
});
```

Terlihat disini, kita membuat sheet dengan nama “Data Buku”. Kita juga melakukan passing data buku yang telah kita temukan pada query sebelumnya.

Mari kita bahas apa yang terjadi pada sheet ini. Pertama, kita menambahkan nama field pada baris pertama.

```
$row = 1;
$sheet->row($row, [
    'Judul',
    'Jumlah',
    'Stok',
    'Penulis'
]);

```

Method `$sheet->row()` berfungsi untuk menambah data pada baris tertentu. Opsi pertama adalah nomor baris. Opsi kedua adalah array berisi data untuk baris tersebut dimulai kolom pertama.

Disini, kita menambah field `Judul`, `Jumlah`, `Stok` dan `Penulis`. Sengaja kita menyimpan nomor baris pada variable `$row` agar kita dapat mengubah nilainya pada saat mengisi data buku.

Syntax terakhir, kita melakukan looping pada data buku yang kita temukan dan mengisi tiap baris pada sheet.

```
foreach ($books as $book) {
    $sheet->row(++$row, [
        $book->title,
        $book->amount,
        $book->stock,
        $book->author->name
    ]);
}
```

Pada syntax ini, kita menggunakan `++$row` untuk mengubah baris pada setiap looping.

Oke, kini cobalah login sebagai Admin dan export data buku.

The screenshot shows the Larapus admin panel at localhost:8000/admin/export/books. The top navigation bar includes links for Larapus, Dashboard, Penulis, Buku, Member, Peminjaman, Profil, and Admin Larapus. The main content area is titled "Export Buku". A modal window titled "Export Buku" displays a table of book data. The table has columns for Judul, Jumlah, Stok, and Penulis. The data shown is:

Judul	Jumlah	Stok	Penulis
Kupinang Engkau dengan Hamdalah	3	2	Mohammad Fauzil Adhim
Jalan Cinta Para Pejuang	2	1	Salim A. Fillah
Membingkai Surga dalam Rumah Tangga	4	4	Aam Amiruddin

Below the table, there is a toolbar with various icons for View, Zoom, Formula, Table, Chart, Text, Shape, Media, Comment, Format, Sort & Filter, and a "DOWNLOAD" button.

Export data ke excel

Banyak sekali fitur dari library ini, silahkan kunjungi [dokumentasi lengkap⁵](#) untuk mempelajarinya.

8.3 Export Data ke PDF

Setelah kita berhasil melakukan export data ke Excel, kini kita akan membuat export data ke PDF. Untuk export ini kita akan menggunakan library [Dompdf⁶](#). Dengan library ini, kita dapat mengeksport data html menjadi pdf. Di Laravel, ada beberapa package yang memudahkan kita menggunakan dompdf. Pada buku ini, saya akan menggunakan [barryvdh/laravel-dompdf⁷](#).

⁵<http://www.maatwebsite.nl/laravel-excel/docs>

⁶<https://github.com/dompdf/dompdf>

⁷<http://github.com/barryvdh/laravel-dompdf>

Untuk membuat fitur export data ke PDF ini, kita akan menggunakan form yang sama. Untuk itu kita akan membuat radio button untuk memilih type output (excel/pdf).

Mari kita mulai dengan menginstall barryvdh/laravel-dompdf:

```
composer require barryvdh/laravel-dompdf=~0.7
```

Lakukan perubahan berikut pada config/app.php:

config/app.php

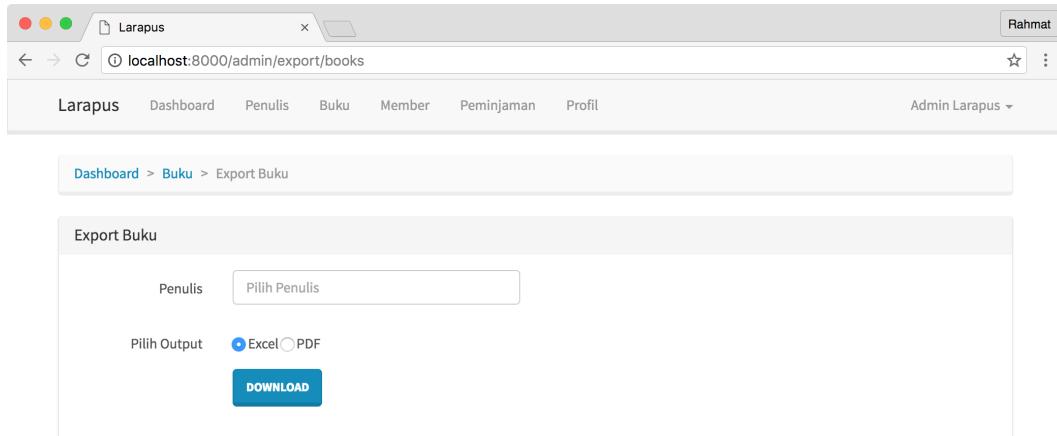
```
....  
'providers' => [  
    ....  
    Barryvdh\DomPDF\ServiceProvider::class,  
],  
....  
'aliases' => [  
    ....  
    'PDF'      => Barryvdh\DomPDF\Facade::class,  
],
```

Kita mulai dengan menambahkan opsi jenis export pada view books.export. Tambahkan syntax berikut sebelum syntax untuk tombol “Download”:

resources/views/books/export.blade.php

```
<div class="form-group {!! $errors->has('type') ? 'has-error' : '' !!}>  
    {!! Form::label('type', 'Pilih Output', ['class'=>'col-md-2 control-label']) !!}  
    <div class="col-md-4 checkbox">  
        {{ Form::radio('type', 'xls', true) }} Excel  
        {{ Form::radio('type', 'pdf') }} PDF  
        {!! $errors->first('type', '<p class="help-block">:message</p>') !!}  
    </div>  
</div>
```

Sehingga tampilan halaman export akan seperti berikut:



Pilih jenis export

Untuk menghandle jenis output yang berbeda, kita akan *refactor* method `exportPost` pada `BooksController`:

`app/Http/Controllers/BooksController.php`

```
public function exportPost(Request $request)
{
    // validasi
    $this->validate($request, [
        'author_id'=>'required',
        'type'=>'required|in:pdf,xls'
    ], [
        'author_id.required'=>'Anda belum memilih penulis. Pilih minimal 1 penulis.'
    ]);

    $books = Book::whereIn('id', $request->get('author_id'))->get();

    $excel::create('Data Buku Larapus', function($excel) use ($books) {
        // Set property
        $excel->setTitle('Data Buku Larapus')
            ->setCreator(Auth::user()->name);

    $excel->sheet('Data Buku', function($sheet) use ($books) {
        $row = 1;
        $sheet->row($row, [
            'Judul',
            'Jumlah',
```

```
        'Stok',
        'Penulis'
    ])/>
    foreach ($books as $book) {
        $sheet->row(++$row, [
            $book->title,
            $book->amount,
            $book->stock,
            $book->author->name
        ]);
    }
});
})->export('xls');
$handler = 'export' . ucfirst($request->get('type'));
return $this->$handler($books);
}
```

Pada syntax ini, kita menambah validasi untuk field type agar user mengisinya dan isiannya hanya boleh pdf atau xls.

```
'type'=>'required|in:pdf,xls'
```

Kita juga menghapus semua logic untuk melakukan export excel dan menggantinya dengan syntax baru.

```
$handler = 'export' . ucfirst($request->get('type'));
return $this->$handler($books);
```

Yang dilakukan syntax ini adalah memanggil method yang sesuai dengan type yang dipilih user. Jika user memilih output pdf, maka dia akan memanggil method exportPdf. Jika memilih xls, maka dia akan memanggil method exportXls.

Teknik memanggil method seperti ini umum digunakan ketika kita ingin membuat sistem yang lebih flexibel. Berbeda dengan if, menggunakan teknik seperti ini akan memudahkan kita menambah jenis output baru. Misalnya, jika kita ingin menambah output xml yang kita lakukan hanya menambah validasi dan menambah method exportXml.

Mari kita buat method untuk export excel. Syntaxnya akan sama dengan syntax untuk export excel yang kita hapus:

app/Http/Controllers/BooksController.php

```
private function exportXls($books)
{
    Excel::create('Data Buku Larapus', function($excel) use ($books) {
        // Set the properties
        $excel->setTitle('Data Buku Larapus')
            ->setCreator('Rahmat Awaludin');

        $excel->sheet('Data Buku', function($sheet) use ($books) {
            $row = 1;
            $sheet->row($row, [
                'Judul',
                'Jumlah',
                'Stok',
                'Penulis'
            ]);
            foreach ($books as $book) {
                $sheet->row(++$row, [
                    $book->title,
                    $book->amount,
                    $book->stock,
                    $book->author->name
                ]);
            }
        });
    })->export('xls');
}
```

Selanjutnya, kita buat method `exportPdf` untuk melakukan export Pdf:

app/Http/Controllers/BooksController.php

```
...
use PDF;

class BooksController extends Controller
{
    ...
    private function exportPdf($books)
    {
        $pdf = PDF::loadview('pdf.books', compact('books'));
        return $pdf->download('books.pdf');
    }
}
```

Pada method ini kita menggunakan PDF::loadview() untuk membuat file pdf pada view dan data yang kita set. Disini, kita menggunakan view pdf.books dan data berupa data buku yang kita passing dari method export. Kemudian kita menggunakan method download() untuk mendownload file pdf yang telah kita buat. Disini kita menggunakan nama file books.pdf.

Isi dari view pdf.books akan seperti berikut:

resources/views/pdf/books.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Data Buku Larapus</title>
    <style>

    /* ----- */

    Hartija Css Print Framework
    * Version: 1.0

    ----- */

body {
    width:100% !important;
    margin:0 !important;
    padding:0 !important;
    line-height: 1.45;
    font-family: Garamond, "Times New Roman", serif;
    color: #000;
    background: none;
    font-size: 14pt; }

/* Headings */
h1,h2,h3,h4,h5,h6 { page-break-after:avoid; }
h1{font-size:19pt;}
h2{font-size:17pt;}
h3{font-size:15pt;}
h4,h5,h6{font-size:14pt; }

p, h2, h3 { orphans: 3; widows: 3; }

code { font: 12pt Courier, monospace; }
blockquote { margin: 1.2em; padding: 1em; font-size: 12pt; }
hr { background-color: #ccc; }
```

```
/* Images */
img { float: left; margin: 1em 1.5em 1.5em 0; max-width: 100% !important; }
a img { border: none; }

/* Links */
a:link, a:visited { background: transparent; font-weight: 700; text-decoration: underline; color: #333; }
a:link[href^="http://"]::after, a[href^="http://"]:visited::after { content: " (" attr(href) ")"; font-size: 90%; }

abbr[title]:after { content: " (" attr(title) ")"; }

/* Don't show linked images */
a[href^="http://"] {color: #000; }
a[href$=".jpg"]::after, a[href$=".jpeg"]::after, a[href$=".gif"]::after, a[href$=".png"]::after {\content: " (" attr(href) ")"; display:none; }

/* Don't show links that are fragment identifiers, or use the `javascript:` pseudo protocol . . taken from html5boilerplate */
a[href^="#"]::after, a[href^="javascript:"]::after {content: "";}

/* Table */
table { margin: 1px; text-align:left; }
th { border-bottom: 1px solid #333; font-weight: bold; }
td { border-bottom: 1px solid #333; }
th,td { padding: 4px 10px 4px 0; }
tfoot { font-style: italic; }
caption { background: #fff; margin-bottom: 2em; text-align: left; }
thead {display: table-header-group; }
img, tr {page-break-inside: avoid; }

/* Hide various parts from the site
   #header, #footer, #navigation, #rightSideBar, #leftSideBar
   {display:none; }
*/
</style>
</head>
<body>
  <h1>Data Buku Larapus</h1>
  <hr>
  <table>
    <thead>
      <tr>
        <td>Judul</td>
        <td>Jumlah</td>
```

```
<td>Stok</td>
<td>Penulis</td>
</tr>
</thead>
<tbody>
@foreach ($books as $book)
<tr>
<td>{{ $book->title }}</td>
<td>{{ $book->amount }}</td>
<td>{{ $book->stock }}</td>
<td>{{ $book->author->name }}</td>
</tr>
@endforeach
</tbody>
</table>
</body>
</html>
```

Yang kita lakukan disini adalah membuat table dengan format yang sama dengan export excel. Pada view ini, kita menggunakan embedded css agar tampilan dari Pdf yang diexport lebih rapi. CSS yang digunakan disini bisa didapatkan dari link berikut⁸.

Cobalah lakukan export pdf dan excel, pastikan berhasil.

⁸<https://github.com/vladocar/Hartija---CSS-Print-Framework>

The screenshot shows a Laravel application interface. At the top, there's a header with the title 'Larapus' and a sub-header 'localhost:8000/admin/export/books'. Below the header is a navigation bar with links: 'Larapus', 'Dashboard', 'Penulis', 'Buku', 'Member', 'Peminjaman', 'Profil', and 'Admin Larapus'. The main content area has a breadcrumb navigation: 'Dashboard > Buku > Export Buku'. A sub-section titled 'Export Buku' displays a list of authors: 'Aam Amiruddin', 'Mohammad Fauzil Adhim', and 'Salim A. Fillah'. Below this, there's a section for selecting the output format, with 'Excel' selected and 'PDF' as an option. A blue 'DOWNLOAD' button is present. To the right, a PDF viewer window titled 'books.pdf (1 page)' is open, showing a table titled 'Data Buku Larapus' with three rows of book data. The table has columns: 'Judul', 'Jumlah', 'Stok', and 'Penulis'. The data is as follows:

Judul	Jumlah	Stok	Penulis
Kupinang Engkau dengan Hamdalah	3	2	Mohammad Fauzil Adhim
Jalan Cinta Para Pejuang	2	1	Salim A. Fillah
Membingkai Surga dalam Rumah Tangga	4	4	Aam Amiruddin

Export PDF

8.4 Import Data dari Excel

Dalam menambahkan data ke aplikasi, terkadang client meminta fitur untuk mengimport data dari Excel. Menggunakan library Laravel Excel yang telah kita gunakan, akan memudahkan kita membuat fitur ini.

Sebelum memulai mengimport excel, silahkan terlebih dahulu baca dokumentasi lengkap tentang import file excel menggunakan library ini di [dokumentasi](#)⁹.

Untuk mengimport data excel kita perlu membuat form. Form ini akan kita letakkan pada halaman penambahan buku, jadi ada dua cara penambahan buku yaitu dengan form dan upload excel. Untuk import excel ini, kita perlu men-standardkan format

⁹<http://www.maatwebsite.nl/laravel-excel/docs/import>

heading agar bisa dibaca (*parsing*) oleh aplikasi kita. Oleh karena itu, kita akan menambahkan tombol untuk mendownload template ini di form upload excel.

Format excel yang kita buat akan memiliki kolom Judul, Penulis dan Jumlah. Data penulis yang kita import, akan langsung didaftarkan jika belum terdaftar. Tentunya, pada saat import file excel kita tidak bisa langsung menambah cover buku.

Setelah data berhasil diimport, kita akan mengarahkan user kembali ke halaman index buku dan memberikan pesan berapa jumlah buku yang berhasil diimport.

Oke, itu logika sederhananya. Mari kita mulai dengan membuat routenya:

routes/web.php

```
Route::group(['prefix'=>'admin', 'middleware'=>['auth', 'role:admin']], function () {
    ...
    Route::get('template/books', [
        'as'    => 'template.books',
        'uses'  => 'BooksController@generateExcelTemplate'
    ]);
    Route::post('import/books', [
        'as'    => 'import.books',
        'uses'  => 'BooksController@importExcel'
    ]);
});
```

Pada route ini kita membuat route untuk mendownload template excel dan menerima form untuk import excel. Routing tersebut menggunakan method generateExcelTemplate dan importExcel pada BooksController. Mari kita buat dulu template untuk kedua method tersebut:

app/Http/Controllers/BooksController.php

```
public function generateExcelTemplate() { }
public function importExcel(Request $request) { }
```

Sebelum kita ke logic di kedua method tersebut, mari kita ubah dulu tampilan halaman pembuatan buku agar menampilkan tab. Pada tab tersebut kita bisa memilih untuk membuat buku dengan mengisi form atau import excel. Buat perubahan berikut:

resources/views/books/create.blade.php

```
....  
<div class="panel-body">  
{!! Form::open(['url' => route('books.store'),  
'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}  
@include('books._form')  
{!! Form::close() !!}  
  
<ul class="nav nav-tabs" role="tablist">  
    <li role="presentation" class="active">  
        <a href="#form" aria-controls="form" role="tab" data-toggle="tab">  
            <i class="fa fa-pencil-square-o"></i> Isi Form  
        </a>  
    </li>  
    <li role="presentation">  
        <a href="#upload" aria-controls="upload" role="tab" data-toggle="tab">  
            <i class="fa fa-cloud-upload"></i> Upload Excel  
        </a>  
    </li>  
</ul>  
<div class="tab-content">  
    <div role="tabpanel" class="tab-pane active" id="form">  
        {!! Form::open(['url' => route('books.store'),  
        'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}  
        @include('books._form')  
        {!! Form::close() !!}  
    </div>  
    <div role="tabpanel" class="tab-pane" id="upload">  
        {!! Form::open(['url' => route('import.books'),  
        'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}  
        @include('books._import')  
        {!! Form::close() !!}  
    </div>  
</div>  
</div>  
....
```

Pada perubahan ini, kita menggunakan fitur tab di bootstrap. Untuk menggunakan fitur ini, kita membutuhkan dua elemen:

```
<ul class="nav nav-tabs" role="tablist">
...
</ul>
```

Digunakan untuk menampilkan tab.

Dan elemen:

```
<div class="tab-content">
...
</div>
```

yang digunakan untuk menampilkan konten dari tab yang dibuat.

Untuk form pembuatan buku, kita masih menggunakan form yang sama. Sementara untuk form import excel, kita menggunakan syntax berikut:

```
{!! Form::open(['url' => route('import.books'),
'method' => 'post', 'files'=>'true', 'class'=>'form-horizontal']) !!}
@include('books._import')
{!! Form::close() !!}}
```

Pada form yang dibuat kita membuat opsi `files` menjadi `true` agar kita dapat mengupload file. Kita juga menggunakan partial view `books._import` untuk menampilkan field di form. Berikut isi dari partial view tersebut:

`resources/views/books/_import.blade.php`

```
<div class="form-group{{ $errors->has('title') ? ' has-error' : '' }}>
  {!! Form::label('template', 'Gunakan template terbaru', ['class'=>'col-md-2 control-label']) !!}
<div class="col-md-4">
  <a class="btn btn-success btn-xs" href="{{ route('template.books') }}"><i class="fa fa-cloud-download"></i> Download</a>
</div>
</div>

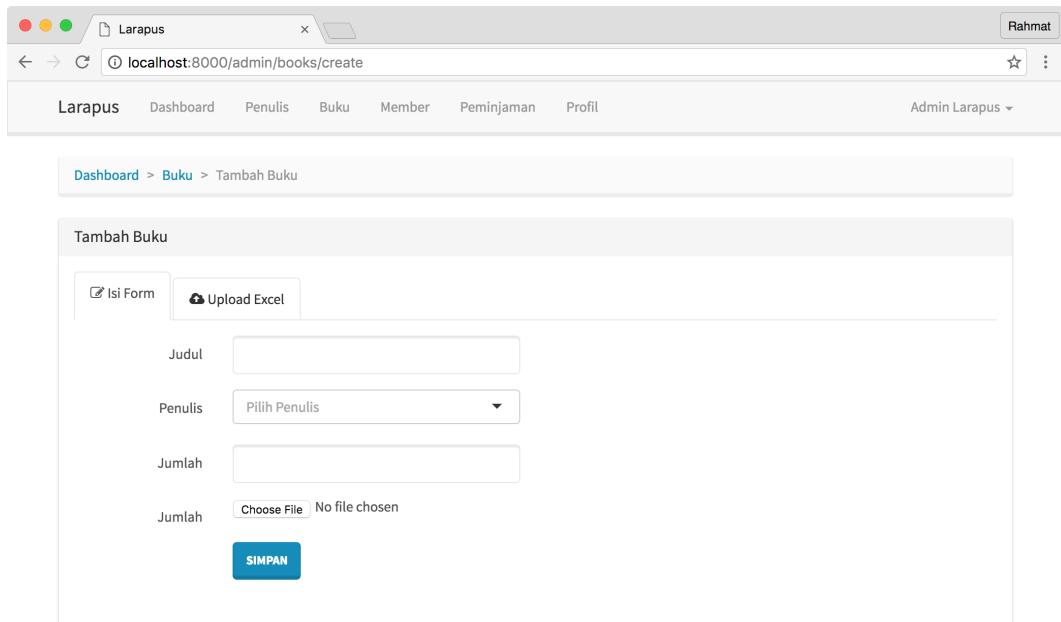
<div class="form-group{{ $errors->has('excel') ? ' has-error' : '' }}>
  {!! Form::label('excel', 'Pilih file', ['class'=>'col-md-2 control-label']) !!}
<div class="col-md-4">
  {!! Form::file('excel') !!}
  {!! $errors->first('excel', '<p class="help-block">:message</p>') !!}
</div>
</div>
```

```
</div>
</div>

<div class="form-group">
  <div class="col-md-4 col-md-offset-2">
    {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
  </div>
</div>
```

Pada partial view ini kita membuat tombol untuk mendownload template excel, field untuk memilih file excel yang hendak diimport dan tombol simpan.

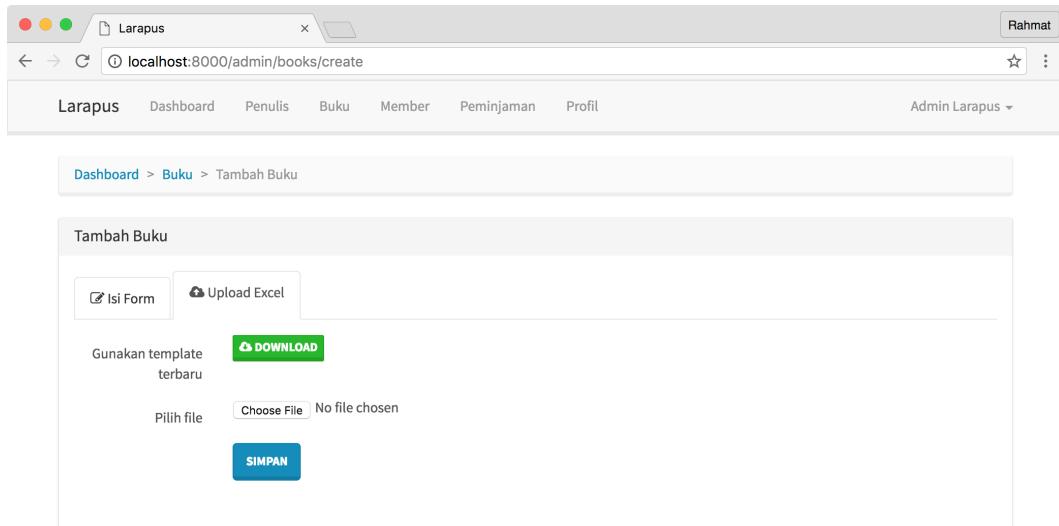
Setelah perubahan ini, tampilan halaman tambah buku akan seperti berikut.



The screenshot shows a web browser window for a Laravel application named 'Larapus'. The URL is 'localhost:8000/admin/books/create'. The page title is 'Tambah Buku'. At the top, there are two tabs: 'Isi Form' (selected) and 'Upload Excel'. Below the tabs are four input fields: 'Judul' (Title), 'Penulis' (Author) with a dropdown menu, 'Jumlah' (Quantity), and a file input field labeled 'Choose File' with the message 'No file chosen'. At the bottom is a blue 'SIMPAN' button.

Halaman tambah buku

Bila kita klik pada tombol “Upload Excel”, kita akan mendapat tampilan berikut.



Upload Excel

Mari kita buat logic untuk menggenerate file template excel. Ubahlah method generateExcelTemplate menjadi:

app/Http/Controllers/BooksController.php

```
public function generateExcelTemplate()
{
    Excel::create('Template Import Buku', function($excel) {
        // Set the properties
        $excel->setTitle('Template Import Buku')
            ->setCreator('Larapus')
            ->setCompany('Larapus')
            ->setDescription('Template import buku untuk Larapus');

        $excel->sheet('Data Buku', function($sheet) {
            $row = 1;
            $sheet->row($row, [
                'judul',
                'penulis',
                'jumlah'
            ]);
        });
    })->export('xlsx');
}
```

Cara kita membuat file excel disini hampir sama dengan cara kita melakukan export file excel pada pembahasan sebelumnya. Disini kita membuat file excel dengan nama “Template Import Buku”, kita set property dari file, membuat sheet “Data Buku” dan menambahkan kolom judul, penulis dan jumlah.

Cobalah klik pada tombol “Download” pada halaman “Upload Excel”, akan muncul file seperti berikut:

The screenshot shows a Microsoft Excel application window titled "Template Import Buku". The ribbon bar at the top includes "View", "Zoom" (set to 125%), "Formula", "Table" (selected), "Chart", "Text", "Format", and "Sort & Filter". Below the ribbon is a toolbar with icons for "View", "Zoom", "Formula", "Table" (highlighted in blue), "Chart", "Text", "Format", and "Sort & Filter". A "Data Buku" tab is selected. The main area contains a table with three columns: "judul", "penulis", and "jumlah". There are 10 rows in the table, each consisting of a single row of empty cells. The table has a light gray border and is centered on the page.

judul	penulis	jumlah		

Template file excel

Selanjutnya, kita ubah method importExcel menjadi:

app/Http/Controllers/BooksController.php

```
....  
use Validator;  
use App\Author;  
  
class BooksController extends Controller  
{  
    ....  
    public function importExcel(Request $request)  
    {  
        // validasi untuk memastikan file yang diupload adalah excel  
        $this->validate($request, [ 'excel' => 'required|mimes:xls,xlsx' ]);  
        // ambil file yang baru diupload  
        $excel = $request->file('excel');  
        // baca sheet pertama  
        $excells = Excel::selectSheetsByIndex(0)->load($excel, function($reader) {  
            // options, jika ada  
        })->get();  
  
        // rule untuk validasi setiap row pada file excel  
        $rowRules = [  
            'judul' => 'required',  
            'penulis' => 'required',  
            'jumlah' => 'required'  
        ];  
  
        // Catat semua id buku baru  
        // ID ini kita butuhkan untuk menghitung total buku yang berhasil diimport  
        $books_id = [];  
  
        // looping setiap baris, mulai dari baris ke 2 (karena baris ke 1 adalah nama kolom)  
        foreach ($excells as $row) {  
            // Membuat validasi untuk row di excel  
            // Disini kita ubah baris yang sedang di proses menjadi array  
            $validator = Validator::make($row->toArray(), $rowRules);  
  
            // Skip baris ini jika tidak valid, langsung ke baris selanjutnya  
            if ($validator->fails()) continue;  
  
            // Syntax dibawah dieksekusi jika baris excel ini valid  
  
            // Cek apakah Penulis sudah terdaftar di database  
            $author = Author::where('name', $row['penulis'])->first();  
  
            // buat penulis jika belum ada  
            if (!$author) {
```

```
$author = Author::create(['name'=>$row['penulis']]);
}

// buat buku baru
$book = Book::create([
    'title'      => $row['judul'],
    'author_id'  => $author->id,
    'amount'     => $row['jumlah']
]);

// catat id dari buku yang baru dibuat
array_push($books_id, $book->id);
}

// Ambil semua buku yang baru dibuat
$books = Book::whereIn('id', $books_id)->get();

// redirect ke form jika tidak ada buku yang berhasil diimport
if ($books->count() == 0) {
    Session::flash("flash_notification", [
        "level"      => "danger",
        "message"   => "Tidak ada buku yang berhasil diimport."
    ]);
    return redirect()->back();
}

// set feedback
Session::flash("flash_notification", [
    "level"      => "success",
    "message"   => "Berhasil mengimport " . $books->count() . " buku."
]);

// Tampilkan index buku
return redirect()->route('books.index');
}
}
```

Pada syntax diatas, sengaja saya tambahkan banyak komentar. Silahkan baca setiap komentar diatas untuk memahami alurnya. Sederhananya, yang kita lakukan adalah:

1. Melakukan validasi file yang diupload
2. Melakukan looping pada setiap baris di sheet pertama (sesuai template)
3. Melakukan validasi pada tiap baris yang di looping

4. Jika validasi sukses, kita membuat penulis (jika belum ada) dan buku nya
5. Jika tidak ada buku yang berhasil diimport, kita arahkan user ke halaman sebelumnya dengan pesan error
6. Jika ada buku yang berhasil diimport, kita arahkan user ke halaman index buku dan menampilkan pesan sukses

Berikut tampilan ketika kita berhasil mengimport buku:

The screenshot shows a web browser window for a Laravel application named 'Larapus'. The URL is 'localhost:8000/admin/books'. The page has a navigation bar with links: Larapus, Dashboard, Penulis, Buku, Member, Peminjaman, Profil, and Admin Larapus. A green success message at the top says 'Berhasil mengimport 3 buku.' Below it, the 'Buku' section displays a table with four rows of book data. Each row includes columns for Judul, Jumlah, Penulis, and two buttons: 'Ubah' and 'HAPUS'. The table data is as follows:

Judul	Jumlah	Penulis	Actions
Buku 1	20	Deni	Ubah HAPUS
Buku 2	30	Deni	Ubah HAPUS
Buku 3	3	Dadang	Ubah HAPUS
Cinta & Seks Rumah Tangga Muslim	3	Aam Amiruddin	Ubah HAPUS

Below the table, a message 'Berhasil import buku' is displayed.

8.5 Penggunaan Ajax

Ajax atau asynchronous Javascript and XML adalah metode yang digunakan berbagai website modern untuk menambah/mengubah/menghapus konten dari halaman web tanpa melakukan refresh halaman. Contohnya di facebook, jika kita membuat status baru maka status itu langsung muncul dan tersimpan di server facebook tanpa perlu melakukan refresh halaman.

Dalam penggunaan di lapangan, biasanya pengaplikasian ajax ini dengan bantuan library javascript seperti jQuery. Meskipun bisa dilakukan tanpa jQuery, saya lebih suka menggunakan jQuery untuk mengaplikasikan ajax dalam web yang dibuat. Sebelum memulai membuat fitur dengan ajax, sebaiknya kita memahami beberapa hal berikut.

8.5.1 Cek Ajax

Untuk mengecek apakah sebuah request adalah ajax, di Laravel dapat menggunakan perintah `Request::ajax()` atau `request()->ajax()`.

8.5.2 CSRF

Salah satu sisi yang penting untuk diperhatikan dalam pengaplikasian ajax di web adalah sisi keamanan. Metode yang sering digunakan oleh cracker untuk merusak web adalah CSRF (Cross Site Request Forgery). Teknik CSRF bekerja dengan memanggil alamat url dari suatu web dari web yang lain. Misalnya, jika kita sedang membuka web sebuah bank dan disitu ada fitur transfer misalnya dengan URL

```
1 http://bank.mandiri.co.id/withdraw?account=Rahmat&amount=1000000&for=Deni
```

maka akun Rahmat akan mentransfer sejumlah Rp1.000.0000 ke akun Deni. Cracker tinggal membuat tag html seperti ini di web nya:

```
1 
```

Ketika user meload halaman dengan tag ini, maka ditransferlah sejumlah uang dari akun Rahmat ke Deni.

Tentunya, contoh diatas terlalu sederhana, karena fitur seperti ini tentunya tidak akan dilakukan dengan GET request. Tapi, walaupun dengan POST request, cracker masih tetap bisa mengirim data ke alamat itu dengan teknik CSRF.

Laravel memudahkan kita untuk memproteksi web dari CSRF dengan teknik token. Jadi, setiap kali sebuah form dibuat di Laravel dengan `Form::open()` atau `Form::model()` maka laravel akan otomatis membuat input baru dengan `name` berisi `_token` yang disembunyikan dan berisi token yang disimpan di sisi server untuk request tersebut. Silahkan cek pada setiap form yang dibuat di Laravel. Pasti ada tulisan seperti ini:

```
1 <input name="_token" type="hidden" value="kar4kterun1q">
```

Tentunya, penggunaan `Form::open()` dan `Form::model()` hanya dimungkinkan setelah kita menginstall package [laravelcollective/html](#)¹⁰. Jika kita tidak menggunakan package tersebut, kita dapat membuat field `_token` ini secara manual dengan syntax:

```
{{ csrf_field() }}
```

Untuk melindungi method/route dari CSRF kita cukup menggunakan middleware `app/Http/Middleware/VerifyCsrfToken.php`. Middleware ini sudah default digunakan pada middleware group `web`. Kita dapat mengeceknya pada file `app/Http/Kernel.php`:

app/Http/Kernel.php

```
protected $middlewareGroups = [
    'web' => [
        ...
        \App\Http\Middleware\VerifyCsrfToken::class,
    ],
    ...
]
```

Jika kita hendak menggunakan form di Laravel dan menggunakan ajax untuk mengirim data form tersebut, kita harus mengirim token ini agar tidak terjadi exception `TokenMismatchException`. Ada dua cara yang bisa kita lakukan:

- Menggunakan field `_token`.
- Menggunakan header `X-CSRF-TOKEN`.

Lebih lanjut tentang CSRF di Laravel dapat dibaca di [dokumentasi](#)¹¹.

¹⁰<https://laravelcollective.com/docs/5.2/html>

¹¹<https://laravel.com/docs/5.3/csrf>

8.5.3 JSON

Response dari server yang biasanya dibutuhkan ajax biasanya berupa JSON. Untuk membuat response JSON di Laravel, gunakan `Response::json()` atau helper `response()->json()`.

Sip, cukup teorinya. Jadi, fitur yang akan kita bangun adalah review buku yang telah diimport. Pada fitur sebelumnya, ketika user telah mengimport excel maka ia langsung diarahkan ke halaman index. Kini, ketika user sudah mengimport buku, ia akan dihadapkan ke halaman review. Pada halaman ini, dia bisa menghapus buku yang baru diimport (jika diinginkan). Fitur penghapusan buku ini akan kita lakukan dengan ajax.

Kita mulai dengan menambahkan halaman review ketika buku berhasil diimport. Kita ubah method `importExcel` menjadi:

app/Http/Controllers/BooksController.php

```
public function importExcel(Request $request)
{
    ...
    // Tampilkan index buku
    return redirect()->route('books.index');

    // Tampilkan halaman review buku
    return view('books.import-review')->with(compact('books'));
}
```

Terlihat disini, setelah berhasil melakukan import, kita tampilkan view `books.import-review` sambil melakukan passing semua buku yang telah diimport. Berikut tampilan untuk view tersebut:

resources/views/books/import-review.blade.php

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

@section('content')


- <a href="{{ url('/home') }}>Dashboard</a></li>
- <a href="{{ url('/admin/books') }}>Buku</a></li>
- <a href="{{ url('/admin/books/create') }}>Tambah Buku</a></li>
- Review Buku</li>



## Review Buku



<a class="btn btn-success" href="{{ url('/admin/books') }}>Selesai</a>



| Judul               | Penulis                    | Jumlah               |                                                                                                                                                                                                                                                                                                                        |
|---------------------|----------------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {{ \$book->title }} | {{ \$book->author->name }} | {{ \$book->amount }} | {!! Form::open(['url' => route('books.destroy', \$book->id), 'id' => 'form-' . \$book->id, 'method' => 'delete', 'data-confirm' => 'Yakin menghapus ' . \$book->title . '?', 'class' => 'form-inline js-review-delete']) !!} {!! Form::submit('Hapus', ['class' => 'btn btn-xs btn-danger']) !!} {!! Form::close() !!} |


```

```
        </table>
    <p> <a class="btn btn-success" href="{{ url('/admin/books') }}">Selesai</a> </p>
</div>
</div>
</div>
</div>
</div>
</div>
@endsection
```

Terlihat disini, kita menampilkan semua buku yang berhasil diimport dalam sebuah table. Pada kolom ke-4, kita membuat tombol “Hapus”:

```
{!! Form::open(['url' => route('books.destroy', $book->id),
'id'          => 'form-' . $book->id, 'method'=>'delete',
'data-confirm' => 'Yakin menghapus ' . $book->title . '?',
'class'        => 'form-inline js-review-delete']) !!}
{!! Form::submit('Hapus', ['class'=>'btn btn-xs btn-danger']) !!}
{!! Form::close() !!}
```

Tombol ini merupakan sebuah form yang kita arahkan ke URL untuk menghapus buku. Pada form ini, kita menambahkan class js-review-delete untuk mengeksekusi javascript yang kita gunakan untuk menghapus buku via ajax. Kita juga menambahkan opsi data-confirm yang akan kita gunakan untuk menampilkan pesan konfirmasi ketika hendak menghapus buku.

Mari kita buat syntax javascriptnya di public/js/custom.js:

public/js/custom.js

```
$(document).ready(function () {
    ...
    // delete review book
    $(document.body).on('submit', '.js-review-delete', function () {
        var $el = $(this);
        var text = $el.data('confirm') ? $el.data('confirm') : 'Anda yakin melakukan tindakan ini\\';
        var c = confirm(text);
        // cancel delete
        if (c === false) return c;

        // delete via ajax
        // disable behaviour default dari tombol submit
```

```
event.preventDefault();
// hapus data buku dengan ajax
$.ajax({
    type      : 'POST',
    url       : $(this).attr('action'),
    dataType : 'json',
    data      : {
        _method : 'DELETE',
        // menambah csrf token dari Laravel
        _token  : $( this ).children( 'input[name=_token]' ).val()
    }
}).done(function(data) {
    // cari baris yang dihapus
    baris = $('#form-'+data.id).closest('tr');
    // hilangkan baris (fadeout kemudian remove)
    baris.fadeOut(300, function() {$(this).remove()});
});
});
```

Syntax ini cukup panjang, silahkan baca komentar yang disertakan untuk memahami tiap baris syntaxnya. Sederhananya, ini yang dilakukan syntax diatas:

1. Kita melakukan konfirmasi sebagaimana kita lakukan pada pembahasan di hari-hari sebelumnya.
2. Kita mengirim request POST ke URL untuk menghapus buku. Disini kita menggunakan jQuery, dokumentasi lengkap untuk syntax ini ada di <http://api.jquery.com/>
3. Pada saat melakukan request POST, kita juga kirim parameter `_method` berisi `DELETE` untuk memberitahu Laravel bahwa kita melakukan proses `destroy`.
4. Kita juga mengirim parameter `_token` berisi token yang ada pada form tersebut.
5. Ketika proses penghapusan berhasil, kita hapus baris tersebut dari tampilan.

Mari kita sesuaikan proses penghapusan buku. Disisi routing, kita tidak perlu melakukan perubahan karena middleware group `web` sudah kita gunakan sebelumnya. Tapi, kita perlu melakukan perubahan pada method `destroy` agar memberikan response yang berbeda ketika kita melakukan penghapusan dengan ajax.

¹²<http://api.jquery.com/jquery.ajax>

app/Http/Controllers/BooksController.php

```
public function destroy($id)
public function destroy(Request $request, $id)
{
    $book = Book::find($id);
    $cover = $book->cover;
    if(!$book->delete()) return redirect()->back();

    // handle hapus buku via ajax
    if ($request->ajax()) return response()->json(['id' => $id]);

    ...
}
```

Pada perubahan ini, ketika user menghapus dengan ajax kita memberikan response berupa json.

Cobalah fitur ini, pastikan proses import berjalan dan kita dapat menghapus setiap buku yang diimport via ajax.

The screenshot shows a browser window titled 'Larapus' with the URL 'localhost:8000/admin/import/books'. A green notification bar at the top says 'Berhasil mengimport 3 buku.' Below it, a breadcrumb navigation shows 'Dashboard > Buku > Tambah Buku > Review Buku'. The main content is a table titled 'Review Buku' with two rows:

Judul	Penulis	Jumlah	
Buku 1	Deni	20	HAPUS
Buku 2	Deni	30	HAPUS

At the bottom left is a green 'SELESAI' button. At the bottom right is a red 'HAPUS' button. Below the table is a developer tools Network tab showing a request to '/admin/books' with a response body containing 'id: "10"'. The status bar at the bottom of the browser indicates '1 requests | 933 B transferred'.

Review dan hapus buku dengan ajax

8.6 Menampilkan Login Terakhir

Ketika Laravel melakukan proses authentikasi, dia akan mengeksekusi beberapa event:

- Illuminate\Auth\Events\Attempting: ketika user gagal login
- Illuminate\Auth\Events\Login: ketika user berhasil login
- Illuminate\Auth\Events\Logout: ketika user berhasil logout
- Illuminate\Auth\Events\Lockout: ketika user beberapa kali gagal login dan diberikan timeout

Menggunakan event yang ada ini, kita dapat melakukan beberapa hal menarik dengan me-listen event yang kita inginkan. Misalnya, kita dapat melakukan pencatatan login terakhir user dengan me-listen event Illuminate\Auth\Events\Login.

Untuk membuat fitur ini, mari kita buat dulu field `last_login` pada table `users` dengan membuat migration baru:

```
php artisan make:migration add_last_login_to_users_table --table=users
// Created Migration: xxxx_xx_xx_xxxxxx_add_last_login_to_users_table
```

Pada migration yang dihasilkan, kita isi method up dan down dengan:

database/migrations/yyyy_xx_xx_xxxxxx_add_last_login_to_users_table.php

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->timestamp('last_login')->after('remember_token')->nullable();
    });
}

public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('last_login');
    });
}
```

Pada migration ini, kita menambah field `last_login` berupa timestamp. Field ini kita tambahkan setelah field `remember_token` dan isiannya boleh null. Kita migrate dulu database untuk mendapatkan field ini:

```
php artisan migrate
// Migrated: xxxx_xx_xx_xxxxxx_add_last_login_to_users_table
```

Selanjutnya, untuk me-*listen* terhadap event yang kita inginkan, kita perlu mendefinisikan listener. Untuk melakukan pendaftaran, kita mengisi array `listen` di `app/Providers/EventServiceProvider.php`:

app/Providers/EventServiceProvider.php

```
protected $listen = [
    'App\Events\SomeEvent' => [
        'App\Listeners\EventListener',
    ],
    'Illuminate\Auth\Events\Login' => [
        'App\Listeners\LogLastLogin',
    ],
];
```

Pada array ini, kita menggunakan format event => listener untuk mendaftarkan setiap listener. Secara default, Laravel akan menggunakan folder app\Listeners untuk menyimpan semua listener. Terlihat disini, kita menggunakan App\Listeners\LogLastLogin untuk melakukan logic yang kita butuhkan.

Nah, disini bagian yang menariknya. Kita dapat generate file untuk listener dengan perintah:

```
php artisan event:generate
// Events and listeners generated successfully!
```

Setelah perintah diatas dijalankan, akan muncul file baru di app/Listeners/LogLastLogin.php dengan isi:

app/Listeners/LogLastLogin.php

```
<?php

namespace App\Listeners;

use Illuminate\Auth\Events\Login;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Contracts\Queue\ShouldQueue;

class LogLastLogin
{
    /**
     * Create the event listener.
     *
     * @return void
     */
}
```

```
public function __construct()
{
    //
}

/**
 * Handle the event.
 *
 * @param Login $event
 * @return void
 */
public function handle(Login $event)
{
    //
}
```

Pada listener ini, kita ubah method `handle` untuk menjalankan logic yang kita inginkan. Syntaxnya akan seperti berikut:

app/Listeners/LogLastLogin.php

```
public function handle(Login $event)
{
    $user = $event->user;
    $user->last_login = new \DateTime;
    $user->save();
}
```

Sip, cukup segitu. Kini kita dapat mencoba login dan pastikan field `last_login` akan berubah setiap kali user berhasil login.

name	email	password	remember_token	last_login	created_at	updated_at
Admin Larapus	admin@gmail.com	\$2y\$10\$2ZnsritozDxhsMextWBAG...GdUeGkaf53xcdudEO5mMCKIEM...		2016-09-10 11:36:44	2016-09-10 11:36:44	2016-09-10 11:36:44
Member Pertama	member@gmail.com	\$2y\$10\$F8KathgGfsZ7YQwm.RIE...7bfqYPEXCPOIP9j6TOThb9IZGgnM...		NULL	2016-09-10 11:36:44	2016-09-10 11:36:44

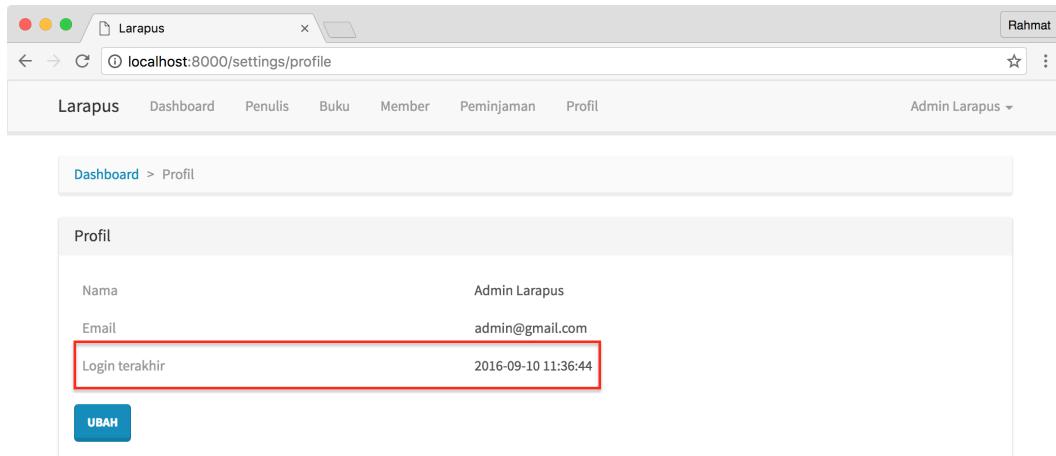
Berhasil mencatat login terakhir

Kita dapat menampilkan field ini pada halaman profil user, tambahkan syntax berikut pada `resources/views/settings/profile.blade.php`:

`resources/views/settings/profile.blade.php`

```
....  
<table class="table">  
  <tbody>  
    <tr>  
      <td class="text-muted">Nama</td>  
      <td>{{ auth()>user()>name }}</td>  
    </tr>  
    <tr>  
      <td class="text-muted">Email</td>  
      <td>{{ auth()>user()>email }}</td>  
    </tr>  
    <tr>  
      <td class="text-muted">Login terakhir</td>  
      <td>{{ auth()>user()>last_login }}</td>  
    </tr>  
  </tbody>  
</table>  
....
```

Kini, halaman profil user akan terlihat seperti berikut.



Menampilkan login terakhir

8.7 Penggunaan Blade Macro

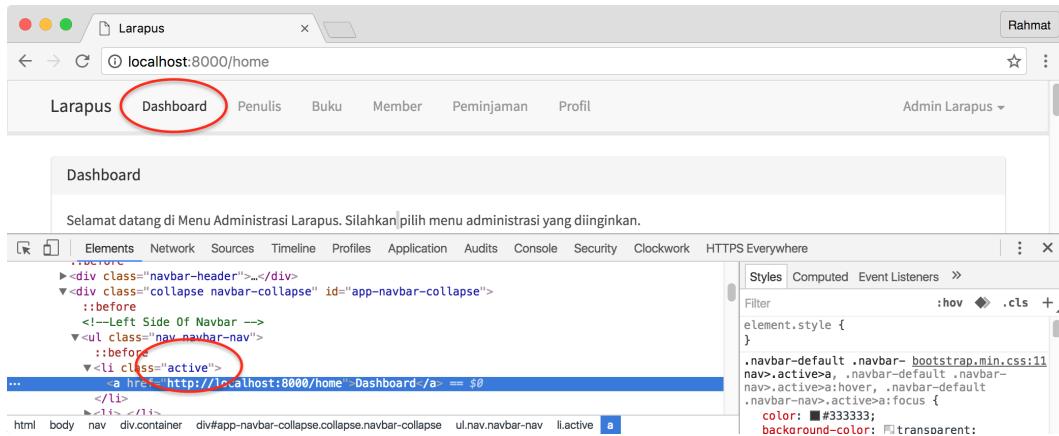
Fitur templating di Blade, memungkinkan kita untuk membuat method baru pada Facade HTML. Teknik ini biasanya digunakan jika kita membutuhkan logic yang sama pada beberapa elemen view. Misalnya kita hendak membangun fitur untuk menambah class “active” pada link navigasi ketika URL nya sama dengan URL yang sedang dikunjungi. Tanpa menggunakan macro, ini syntax yang harus kita tulis pada link navigasi:

resources/views/layouts/app.blade.php

```
....  
<li><a href="{{ url('/home') }}">Dashboard</a></li>  
<li class="{{ url('/home') == request()->url() ? 'active' : '' }}><a href="{{ url('/home') }}>Dashboard</a></li>  
....
```

Disini, kita menggunakan `request()->url()` untuk mengetes URL yang sedang aktif. Jika sama dengan URL pada element `<a>`, maka kita tambahkan class `active` pada element ``.

Ini tampilan yang akan muncul ketika kita mengunjungi halaman dashboard.



Menambah class active

Yap, meskipun berjalan, kita akan melakukan banyak yang sama pada banyak link navigasi. Hasil akhir jika kita menggunakan Macro, syntaxnya akan seperti ini:

```
{!! Html::smartNav(url('/home'), 'Dashboard') !!}
```

Lebih singkat kan?

Oke, mari kita buat macro ini. Untuk membuat macro, kita harus menyimpan syntaxnya pada file yang akan di load oleh Laravel. Mari kita buat file app/Helpers/frontend.php dan meload file ini pada app/Providers/AppServiceProvider.php. Kita buat dulu filenya:

app/Helpers/frontend.php

```
<?php

\Html::macro('smartNav', function($url, $title) {
    $class = $url == request()->url() ? 'active' : '';
    return "<li class=\"$class\"><a href=\"$url\">$title</a></li>";
});
```

Terlihat disini, kita menggunakan `Html::macro()` untuk membuat macro baru. Kita membuat macro dengan nama `smartNav` yang akan menerima parameter berupa

URL dan nama dari link navigasinya. Kemudian kita melakukan logic yang sama yang kita lakukan pada pembahasan sebelumnya untuk menampilkan class active.

Kita load file ini dengan menambah syntax berikut pada method boot di app/Providers/AppServiceProvider.php:

app/Providers/AppServiceProvider.php

```
public function boot()
{
    require base_path() . '/app/Helpers/frontend.php';
    ...
}
```

Untuk menggunakan macro ini, kita ubah link navigasi menjadi:

resources/views/layouts/app.blade.php

```
<ul class="nav navbar-nav">
    @if (Auth::check())
        <li><a href="{{ url('/home') }}">Dashboard</a></li>
        {!! Html::smartNav(url('/home'), 'Dashboard') !!}
    @endif
    @role('admin')
        <li><a href="{{ route('authors.index') }}">Penulis</a></li>
        <li><a href="{{ route('books.index') }}">Buku</a></li>
        <li><a href="{{ route('members.index') }}">Member</a></li>
        <li><a href="{{ route('statistics.index') }}">Peminjaman</a></li>
        {!! Html::smartNav(route('authors.index'), 'Penulis') !!}
        {!! Html::smartNav(route('books.index'), 'Buku') !!}
        {!! Html::smartNav(route('members.index'), 'Member') !!}
        {!! Html::smartNav(route('statistics.index'), 'Peminjaman') !!}
    @endrole
    @if (auth()->check())
        <li><a href="{{ url('/settings/profile') }}">Profil</a></li>
        {!! Html::smartNav(url('/settings/profile'), 'Profil') !!}
    @endif
</ul>
```

Kini, class active akan muncul pada link navigasi setiap kali kita mengunjungi halaman untuk link tersebut. Sip.

Penutup

Akhirnya selesai juga kita mempelajari buku ini. Tidak terasa 7 hari telah dilalui. Memang masih banyak fitur Laravel yang belum kita bahas. Namun, saya harap pengalaman membangun aplikasi Larapus ini dapat menjadi bekal untuk mengembangkan aplikasi yang lebih besar.

Beberapa hal yang belum kita pelajari :

- Pembuatan Package
- Service Provider
- Facade
- Localization/fitur multi bahasa
- Cache
- Queue
- Session
- Testing
- Pagination
- dan masih banyak lagi

Belajar lagi!

Saya harap Anda tidak puas dan terus belajar tentang framework ini. Beberapa sumber belajar Laravel yang bisa dimanfaatkan :

- [<https://leanpub.com/bukularavel>] : Buku Menyelami Framework Laravel, lanjutan dari buku ini.
- [<https://malescast.com>] : Berisi screencast yang sudah saya buat seputar Laravel dan web development.

- [http://laravel.com/docs¹³](http://laravel.com/docs) : Dokumentasi resmi framework Laravel
- [http://laravel.com/api¹⁴](http://laravel.com/api) : Dokumentasi API resmi framework Laravel
- [http://laravel.io/forum¹⁵](http://laravel.io/forum) : Forum resmi framework Laravel
- [http://laravel.io/chat¹⁶](http://laravel.io/chat) : Livechat resmi framework Laravel
- [http://laracasts.com¹⁷](http://laracasts.com) : Koleksi video tutorial Laravel dari Jeffrey Way
- [http://www.laravel-tricks.com¹⁸](http://www.laravel-tricks.com) : Koleksi tips seputar penggunaan Laravel
- [https://www.facebook.com/groups/laravel¹⁹](https://www.facebook.com/groups/laravel) : Grup unofficial Laravel Indonesia
- [https://medium.com/laravel-indonesia²⁰](https://medium.com/laravel-indonesia) : Koleksi artikel Laravel berbahasa Indonesia. Anda dapat submit artikel tentang Laravel di web ini. Saya juga suka menulis disini.

Tips lain belajar, baca source code Laravel. Serius. Saya sendiri, kalau kurang paham beberapa fitur di Laravel, sering buka source codenya langsung.. :D

Terima kasih

Teriring do'a, saya ucapan terima kasih bagi Anda yang telah membeli buku ini. Dengan membeli buku ini, Anda telah membantu saya untuk tetap konsisten berbagi ilmu di bidang web development. Anda juga telah membantu memberi nafkah untuk keluarga saya dengan nafkah yang halal. Saya yakin, rezeki dari Tuhan tidak akan pernah salah sasaran. Setiap orang pasti telah dikaruniai rezeki sesuai dengan kadar kebijaksanaan dalam dirinya. Semoga ilmu yang diperoleh berkah.. :)

NB : Bagi yang masih belum beli buku ini atau yang masih copy pdf dari teman, saya do'akan agar segera dicukupkan rezekinya supaya bisa membeli (lisensi) buku ini. Supaya ilmunya berkah.. :)

¹³<http://laravel.com/docs>

¹⁴<http://laravel.com/api>

¹⁵<http://laravel.io/forum>

¹⁶<http://laravel.io/chat>

¹⁷<http://laracasts.com>

¹⁸<http://www.laravel-tricks.com>

¹⁹[https://www.facebook.com/groups/laravel/](https://www.facebook.com/groups/laravel)

²⁰<https://medium.com/laravel-indonesia>

Sekian,

Salam hangat dari saya, istri dan putra saya untuk Anda.