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 Otwell[1]

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`:

routes/web.php

```
<?php
....
```

[1] http://taylorotwell.tumblr.com/post/21038245018/why-laravel

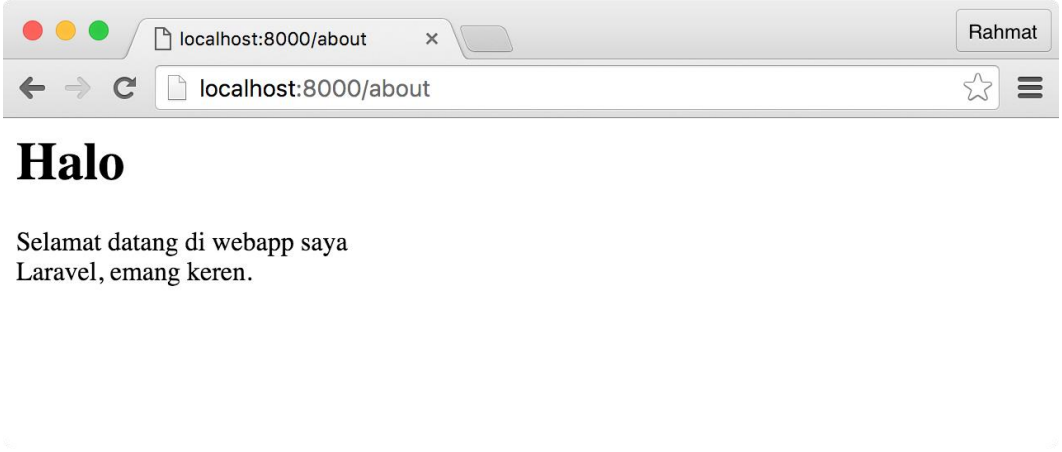
```
Route::get('/', function () {
    return view('welcome');
});
```

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`.



Berhasil membuat route about

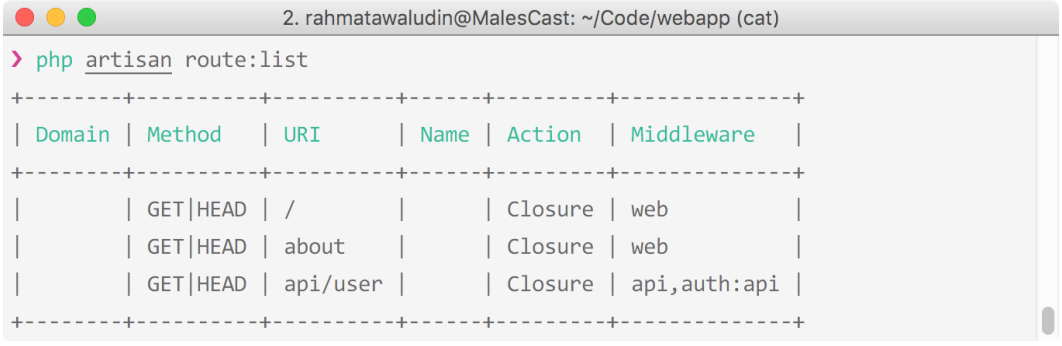
Mari kita perhatikan syntax `Route::get('/about', function() { ... })`.

- 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
```



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[1].

2.2 MVC

Laravel mendukung penuh pembuatan aplikasi dengan arsitektur Model View Controller (MVC)[2] untuk memisahkan logic manipulasi data, antarmuka pengguna dan kontrol aplikasi. Mari kita bahas bagaimana Laravel mengimplementasikannya.

[1] https://laravel.com/docs/5.3/routing
[2] 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 database[4] . Laravel memiliki fitur menarik untuk manajemen database yaitu migrations dan seeding[5] .

2.3.1 Migrations

Laravel memudahkan kita untuk membuat struktur database yang dapat disimpan dalam VCS[6] semisal Git[7] . 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/xxxx_xx_xx_xxxxxx_create_posts_table.php`. Ubahlah isian file tersebut menjadi:

database/migrations/xxxx_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 dokumentasi[8] . 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 Pro[9] di Mac untuk melihat struktur database. Untuk pengguna Windows atau Linux, dapat menggunakan phpMyAdmin[10] (yang sudah bawaan Xampp) atau MySQL Workbench[11].

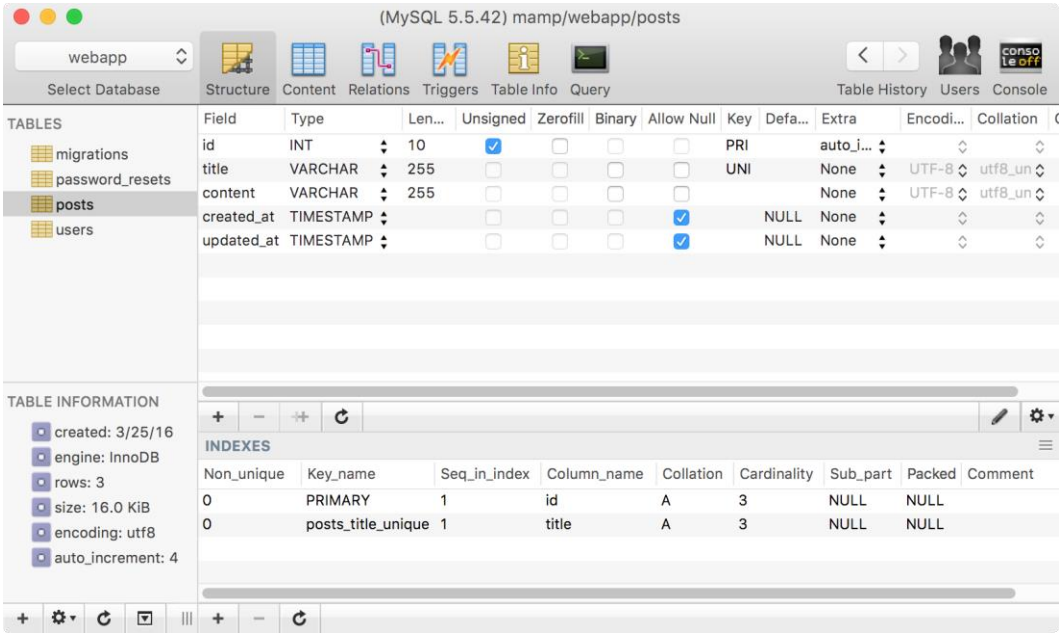
---

[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



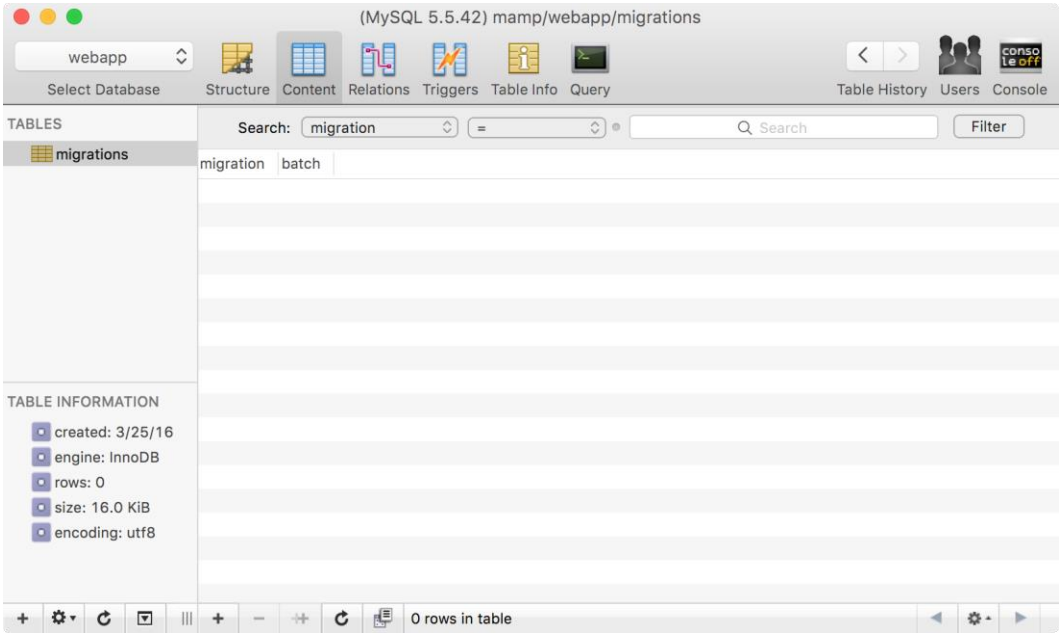
Migrasi berhasil

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.

4. Untuk mendemonstasikan kegunaan migration, mari kita lakukan `rollback` untuk meng-undo migrasi yang 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
```

5. Cek kembali database, maka table `posts` akan terhapus.



Rollback berhasil

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 migrasi[12].

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);
}
....
```

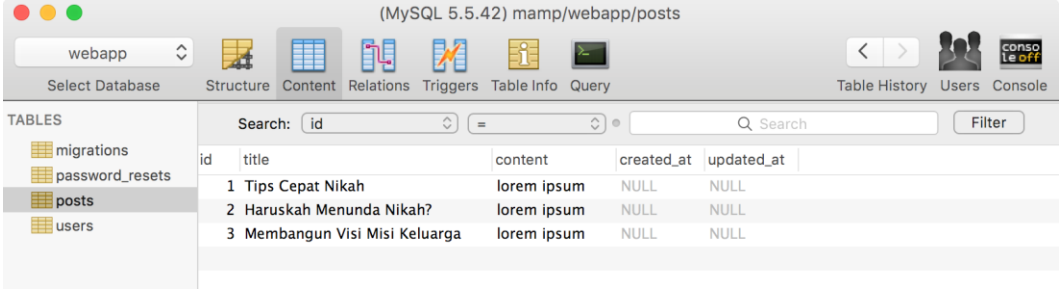
4. Untuk melakukan seeding, jalankan perintah ini:

```
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.

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



Database Seeding berhasil

2.3.3 Membuat Model

Model dalam Laravel dibuat dengan melakukan extends class `Illuminate\Database\Eloquent\M`

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 builder[15] dan eloquent[16] .

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:

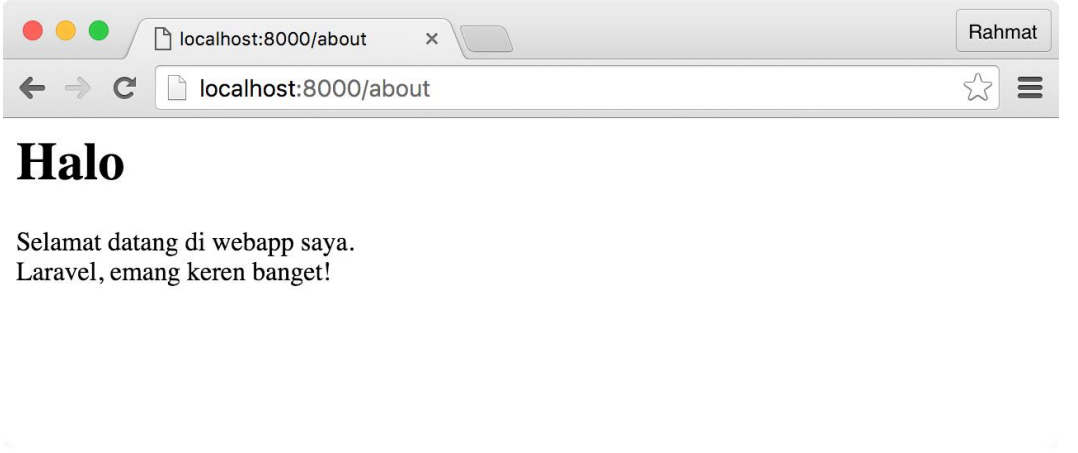
[15] https://laravel.com/docs/5.3/queries

[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.



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 Blade[17] .

[17] https://laravel.com/docs/5.3/blade

Untuk menggunakan view dengan blade template, kita cukup merubah ektensi file view menjadi `.blade.php`. Pada contoh file `about.php`, maka kita ubah menjadi `about.blade.php` untuk menggunakan template 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 placholder 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 form[20] .

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 request[1].

2.5 Controller

Pada contoh-contoh sebelumnya, 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 praktekan 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

[1] https://laravel.com/docs/5.3/requests

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

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

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 controller[1].

[1] https://laravel.com/docs/5.3/controllers

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:

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

Akan muncul beberapa file baru:

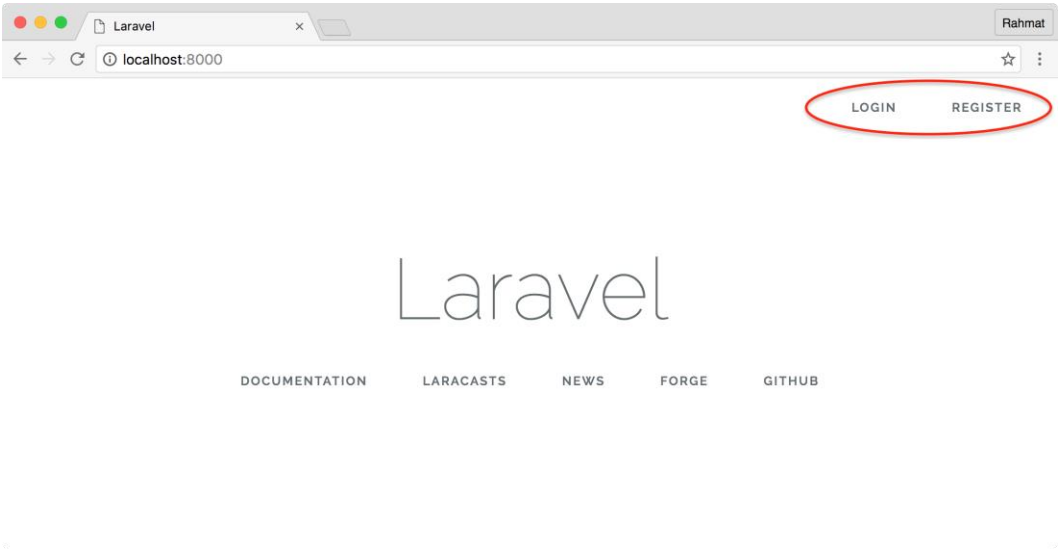
- `app/Http/Controllers/HomeController.php`
- `resources/views/home.blade.php`
- `resources/views/auth/login.blade.php`
- `resources/views/auth/register.blade.php`
- `resources/views/auth/passwords/email.blade.php`
- `resources/views/auth/passwords/reset.blade.php`
- `resources/views/layouts/app.blade.php`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

akan muncul baris berikut:

routes/web.php

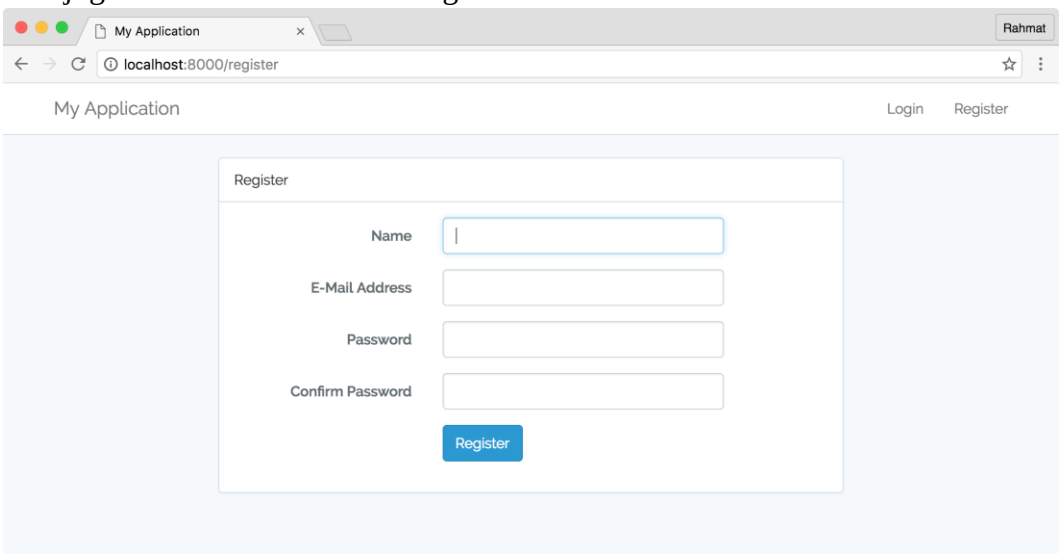
```
Auth::routes();
Route::get('/home', 'HomeController@index');
```

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

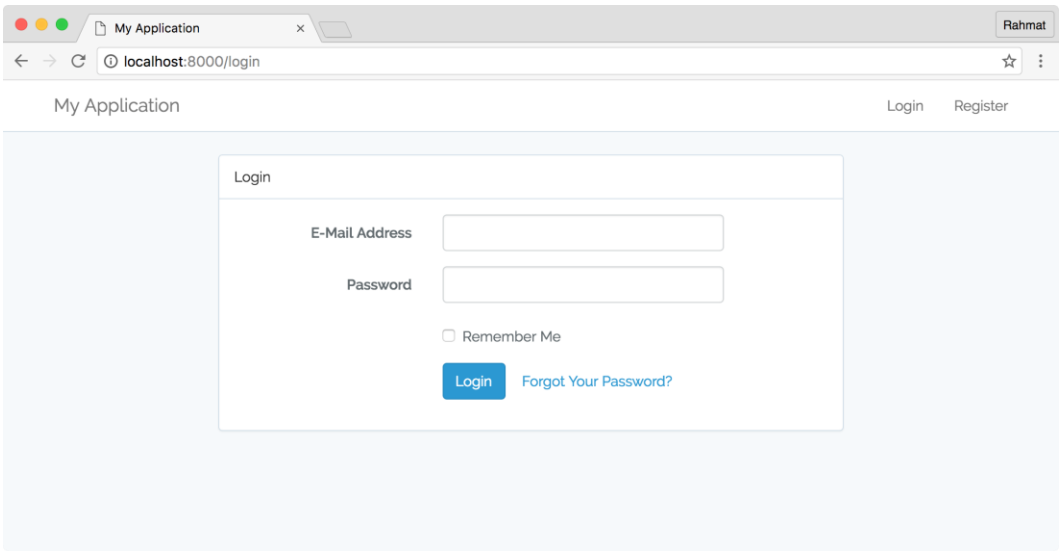


Halaman utama default

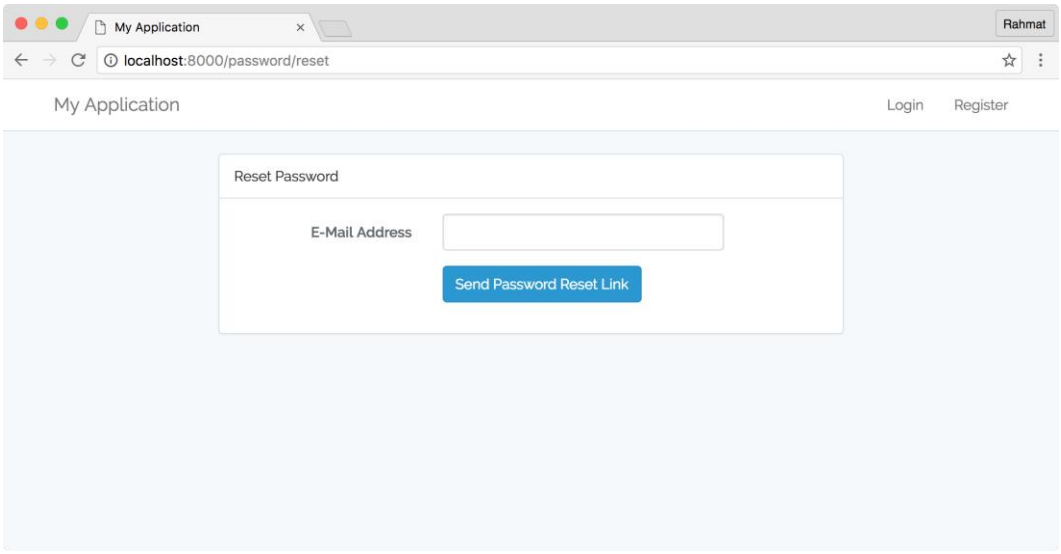
Kita juga sudah disediakan fitur register:



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 memulai membuat perencanaan untuk project yang akan dibangun dengan framework Laravel. Semangat! :)