Laravel 5.3 Social and Email Multi-Authentication

> Checkout **Live Demo** here and see how awesome it is

In this tutorial I will create **Laravel** application with email authentication, but also I will use Laravel Socialite for Facebook, Twitter, Google+ and GitHub logins. Once when I configure everything, app will also be able to use many other social platforms for sign in process. Entire list of social providers is here. Some of them are **Paypal**, **Reddit**, **Linkedin**, **Tumblr**, **Youtube** and **Google**. Application will have user roles, email activation feature and basic user profile options.

> You can find entire code used in this tutorial on Github.

I will start from new Laravel installation, for detailed installation steps check here. After installation I will copy *.env.example* file to *.env* and insert database credentials there.

While I am developing web projects I like to configure fake .dev domains. My local development machine is running Ubuntu 14.04 and Apache2 web-server, so I configured virtual hosts and modified /etc/hosts file to point social-laravel.dev to localhost. You can do this a number of ways, also you don't need to use fake domains. I pointed out this, cause throughout tutorial you will be seeing that domain.

Development schedule

It is always good idea to create some kind of schedule or plan, before you actually start coding. So I like to write down simple steps and then complete them one by one, until entire app is completed.

For now this app will have home page, login/register pages with forms for email authentication and social buttons for social login. Usually every app needs to have at least 2 user roles, for administrator and for ordinary users, so I will code basic user-role logic. Users will be able to reset their passwords, so app will send some emails. Also site owner will be able to turn on/off email activation feature, if it is on and user didn't activated his email system will display alert above navigation bar. Users will also be able to resend activation email, or change email via profile page.

I will code system in such a way that adding new social providers like YouTube is going to be trivial and short process (under 20 seconds).

- Creating views
    - Layout
    - Login
    - Password Reset
    - Password Reset Form
    - Register
    - Home Page
    - User Panel
    - Admin Panel

- Create migrations and models related to users and roles
- User seeder with some dummy users
- Middleware for administrator and user roles
- Routes and Auth Controller
- Create table and model for Social logins
- Create Social Logic

This schedule is not strict it is more like a guide. I always like to complete frontend first so I can enjoy building logic and backend later.

Creating views

When it comes to HTML, CSS and other frontend stuff I like to use Bootstrap framework. Recently I noticed Material Design for Bootstrap theme and I like how it looks so I'll use it. You can preview MDB here and download it for FREE. We will use latest Bootstrap 4 in this tutorial.

Layout

I will first code main layout file, which will act as shell for every page on our site.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>Laravel Social and Email Authentication</title>

    <meta name="description" content="Laravel 5.3 bootstrap app with Multi
Auth, Social and Email Authentication. Google re-Captcha, Facebook,
Twitter, G+ and much more...">
    <meta name="author" content="Ivan Radunovic">
    <link rel="shortcut icon"
href="https://tuts.codingo.me/assets/img/box.png">

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-
alpha.3/css/bootstrap.min.css">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.6.3/css/font-
awesome.min.css">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.1.1/css/mdb.min.css
```

```
    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and
media queries -->
    <!--[if lt IE 9]>
    <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
</script>
    <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js">
</script>
    <![endif]-->

    @yield('head')

</head>

<body>

<!--Navigation-->
<header>

@include('partials.above-navbar-alert')

<!--Navbar-->
    <nav class="navbar navbar-dark scrolling-navbar mdb-gradient">

        <!-- Collapse button-->
        <button class="navbar-toggler hidden-sm-up" type="button" data-
toggle="collapse" data-target="#collapseEx">
            <i class="fa fa-bars"></i></button>

        <div class="container">

            <!--Collapse content-->
            <div class="collapse navbar-toggleable-xs" id="collapseEx">

                <!--Links-->
                <ul class="nav navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link"
href="https://github.com/codingo-me/laravel-social-email-authentication"
target="_blank"><i class="fa fa-download"></i>  Download</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ route('public.home')
}}"> Home</a>
                    </li>
                    @if(!Auth::check())
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/login') }}">
Login</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/register') }}">
Register</a>
                    </li>
                    @else
                    <li class="nav-item">
                        <a class="nav-link" href="#"> {{ Auth::user()-
>first_name }}</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/logout') }}">
Logout</a>
                    </li>
                    @endif
                </ul>

                <!--Navbar icons-->
                <ul class="nav navbar-nav nav-flex-icons">
                    <li class="nav-item">
                        <a href="https://www.facebook.com/codingo.me/"
class="nav-link"><i class="fa fa-facebook"></i></a>
                    </li>
                    <li class="nav-item">
                        <a href="https://twitter.com/codingo_me"
class="nav-link"><i class="fa fa-twitter"></i></a>
                    </li>
                    <li class="nav-item">
                        <a
href="https://plus.google.com/u/2/b/109783202683475265470/collection/wwmLx"
 class="nav-link"><i class="fa fa-google-plus"></i></a>
                    </li>
```

```html
                    <li class="nav-item">
                        <a href="https://github.com/codingo-me" class="nav-
link"><i class="fa fa-github"></i></a>
                    </li>
                </ul>

            </div>
            <!--/.Collapse content-->

        </div>

    </nav>
    <!--/.Navbar-->


</header>
<!--/Navigation-->

<main>
<div class="container">

    <div style="height: 90px;"></div>
    @yield('content')

</div> <!-- /container -->
</main>

<!-- Bootstrap core JavaScript
================================================== -->
<!-- Placed at the end of the document so the pages load faster -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js"></scr

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/4.0.0-alpha.3/js/bootstrap.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.1.1/js/mdb.min.js">
</script>
<!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
<script src="/assets/js/ie10-viewport-bug-workaround.js"></script>
```

```
@yield('footer')

</body>
</html>
```

I will copy this code into `views/layouts/main.blade.php` this will be layout file.

From layout file you'll notice couple of sections like head, content and footer, I think they are self-explanatory. Above navigation bar I included one bootstrap alert, which will be used later during email activation process. It is stored in `views/partials/above-navbar-alert.blade.php`.

```
@if(session()->has('above-navbar-message') && auth()->check())
    <div class="alert alert-info" role="alert" style="margin-
bottom:0px;background-color:#000;border-color:#000;color:#fff;">
        <button type="button" class="close" data-dismiss="alert"
style="color:#fff;">×</button>
        {!! session()->get('above-navbar-message') !!}
    </div>
@endif
```

In earlier versions of Laravel, framework was shipped with built-in HTML helpers, but not any more, so I need to import them if I plan to use them. In composer.json file I will add html helper require statement and Socialite, Recaptcha and Predis packages cause I will use them later:

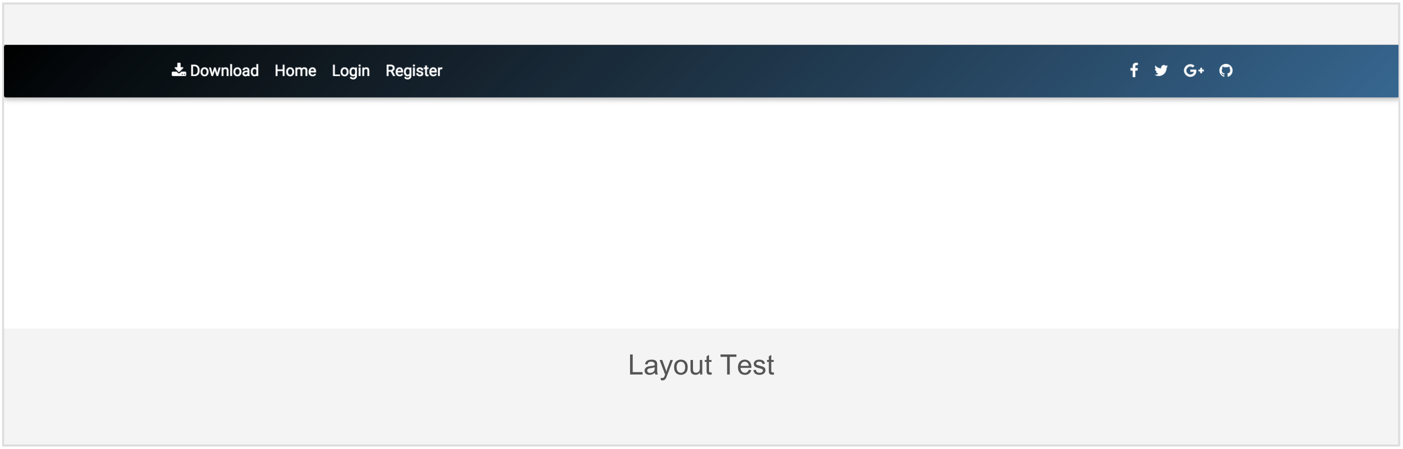
```
    "require": {
        "php": ">=5.6.4",
        "laravel/framework": "5.3.*",
        "laravelcollective/html": "^5.2.0",
        "laravel/socialite": "^2.0",
        "google/recaptcha": "~1.1",
        "predis/predis": "^1.1"
    }
```

After adding new packages to composer.json I need to update the system with: `composer update`

> You can also add these packages without manually modifying composer.json file or updating all dependencies with `composer update` For Socialite package I could execute `composer require laravel/socialite` you the the idea. This option is much faster than updating all the dependencies.

Now I will test main layot file, you can accomplish that by creating route closure which will return this view in `web.php` routes file. Something like this:

```
Route::get('test', function ()
{
   return view('layouts.main');
});
```



Layout Test

Login Page

Login view will be located in *views/auth/login.blade.php* and it will have following content:

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

@section('head')

    <link rel="stylesheet" href="/assets/css/signin.css">

@stop

@section('content')


        {!! Form::open(['url' => url('#'), 'class' => 'form-signin'] ) !!}


        @include('includes.status')

        <h2 class="form-signin-heading">Please sign in</h2>

        <label for="inputEmail" class="sr-only">Email address</label>
        {!! Form::email('email', null, [
            'class'                         => 'form-control',
            'placeholder'                   => 'Email address',
            'required',
            'id'                            => 'inputEmail'
        ]) !!}

        <label for="inputPassword" class="sr-only">Password</label>
        {!! Form::password('password', [
            'class'                         => 'form-control',
            'placeholder'                   => 'Password',
            'required',
            'id'                            => 'inputPassword'
        ]) !!}

        <div style="height:15px;"></div>
        <div class="row">
            <div class="col-md-12">
```

```
                <fieldset class="form-group">
                    {!! Form::checkbox('remember', 1, null, ['id' =>
'remember-me']) !!}
                    <label for="remember-me">Remember me</label>
                </fieldset>
            </div>
        </div>

        <button class="btn btn-lg btn-primary btn-block login-btn"
type="submit">Sign in</button>
        <p><a href="{{ url('#') }}">Forgot password?</a></p>

        <p class="or-social">Or Use Social Login</p>

        @include('partials.socials')

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

@stop
```

From above code you could see that I am including status view from `views/includes/status.blade.php` This status view is in charge of displaying alerts to user, usually success messages.

```
@if(Session::has('message'))
    <div class="alert alert-{{ Session::get('status') }} status-box">
        <button type="button" class="close" data-dismiss="alert"><span
aria-hidden="true">&times;</span><span class="sr-
only">Close</span></button>
        {{ Session::get('message') }}
    </div>
@endif
```

You can see that I am using Bootstrap alert, but you can use any kind of HTML. You can even display Sweet Alert dialog.

In the bottom I am including `partials/socials.blade.php` file, which contains social login buttons. I extracted these buttons to separate file, cause I'll use same buttons on register page also.

```
<div class="row margin-bottom-10">
    <div class="col-md-6 col-sm-6 col-xs-6">
        <a href="#" class="btn btn-lg waves-effect waves-light  btn-block facebook">Facebook</a>
    </div>
    <div class="col-md-6 col-sm-6 col-xs-6">
        <a href="#" class="btn btn-lg  waves-effect waves-light btn-block twitter">Twitter</a>
    </div>
</div>

<div class="row">
    <div class="col-md-6 col-sm-6 col-xs-6">
        <a href="#" class="btn btn-lg waves-effect waves-light btn-block google">Google+</a>
    </div>
    <div class="col-md-6 col-sm-6 col-xs-6">
        <a href="#" class="btn btn-lg waves-effect waves-light btn-block github">GitHub</a>
    </div>
</div>
```

I am using same CSS from example page with some minor modifications. As you can see styling files are located in `/assets/css/`.

```
.form-signin {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
}
.form-signin .form-signin-heading,
```

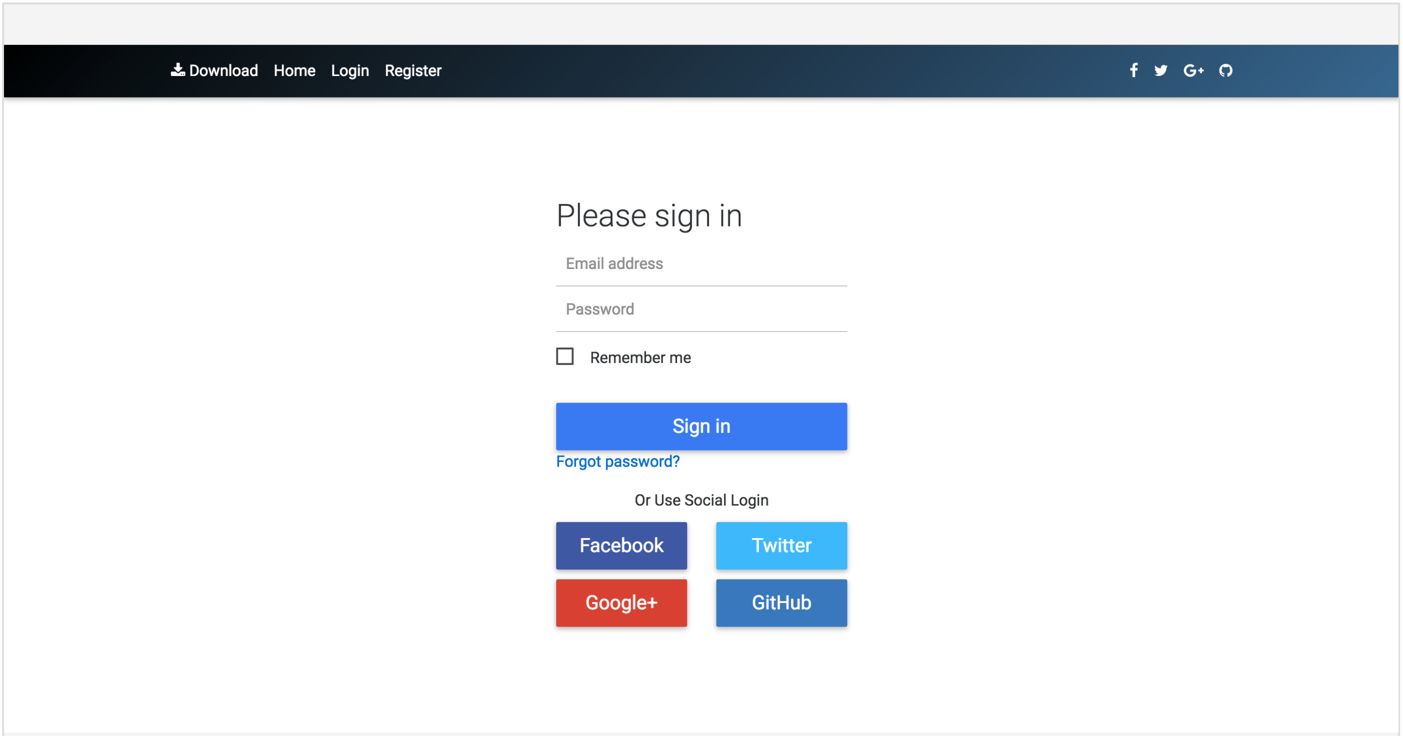
```css
.form-signin .checkbox {
    margin-bottom: 10px;
}
.form-signin .checkbox {
    font-weight: normal;
}
.form-signin .form-control {
    position: relative;
    height: auto;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    padding: 10px;
    font-size: 16px;
}
.form-signin .form-control:focus {
    z-index: 2;
}
.form-signin input[type="email"] {
    margin-bottom: -1px;
    border-bottom-right-radius: 0;
    border-bottom-left-radius: 0;
}
.form-signin input[type="password"] {
    margin-bottom: -1px;
    border-top-left-radius: 0;
    border-top-right-radius: 0;
}

.login-btn{
    margin-top:10px;
}
.or-social{
    text-align:center;
    margin: 10px 0 10px 0;
}
.facebook{
    background-color: #4863ae;
    border-color: #4863ae;
}
```

```
.facebook:hover{
    background-color: #2871aa;
    border-color: #2871aa;
}
.twitter{
    background-color: #46c0fb;
    border-color: #46c0fb;
}
.twitter:hover{
    background-color: #00c7fb;
    border-color: #00c7fb;
}
.google{
    background-color: #DD4B39;
    border-color: #DD4B39;
}
.google:hover{
    background-color: #e15f4f;
    border-color:#e15f4f;
}
.github{
    background-color: #4183C4;
    border-color: #4183C4;
}
.github:hover{
    background-color: #5490ca;
    border-color:#5490ca;
}
.margin-bottom-10{
    margin-bottom:10px;
}
[type=checkbox]:checked, [type=checkbox]:not(:checked) {
    position: absolute;
    left: -9999px;
    visibility: hidden;
}
[type=checkbox], [type=radio] {
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
    padding: 0;
```

```
}
[type=checkbox]+label {
    position: relative;
    height: 25px;
}
[type=checkbox]+label:before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 18px;
    height: 18px;
    z-index: 0;
    border: 2px solid #5a5a5a;
    border-radius: 1px;
    margin-top: 2px;
    -webkit-transition: .2s;
    -moz-transition: .2s;
    -o-transition: .2s;
    -ms-transition: .2s;
    transition: .2s;
}
[type=radio]:checked+label, [type=radio]:not(:checked)+label,
[type=checkbox]+label {
    -khtml-user-select: none;
    -webkit-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    cursor: pointer;
    font-size: 1rem;
    padding-left: 35px;
    display: inline-block;
    line-height: 25px;
}
[type=checkbox]:checked+label:before {
    top: -4px;
    left: -3px;
    width: 12px;
    height: 22px;
    border-top: 2px solid transparent;
    border-left: 2px solid transparent;
    border-right: 2px solid #4285F4;
    border-bottom: 2px solid #4285F4;
    -webkit-transform: rotate(40deg);
    -moz-transform: rotate(40deg);
    -ms-transform: rotate(40deg);
    -o-transform: rotate(40deg);
    transform: rotate(40deg);
    -webkit-backface-visibility: hidden;
    -webkit-transform-origin: 100% 100%;
    -moz-transform-origin: 100% 100%;
    -ms-transform-origin: 100% 100%;
    -o-transform-origin: 100% 100%;
    transform-origin: 100% 100%;
}
[type=checkbox]+label:before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 18px;
    height: 18px;
    z-index: 0;
    border: 2px solid #5a5a5a;
    border-radius: 1px;
    margin-top: 2px;
    -webkit-transition: .2s;
    -moz-transition: .2s;
    -o-transition: .2s;
    -ms-transition: .2s;
    transition: .2s;
}
.error-text{
    color: #F44336;
    transition: .2s opacity ease-out,.2s color ease-out;
}
```

I will test it in a same way as layout page, to cut time needed you can just replace `return view('layouts.main');` with `return view('auth.login');` in routes file. Here is the login page:



Please sign in

Email address

Password

Remember me

Sign in

Forgot password?

Or Use Social Login

Facebook

Twitter

Google+

GitHub

Login Page

Password Reset Page

I will not attach images of password reset pages here cause they are trivial and very similar to Login page.

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

@section('head')
    {!! HTML::style('/assets/css/reset.css') !!}
@stop

@section('content')

        {!! Form::open(['url' => url('/password/email'), 'class' => 'form-
signin' ] ) !!}
```

```
        @include('includes.status')

        <h2 class="form-signin-heading">Password Reset</h2>
        <label for="inputEmail" class="sr-only">Email address</label>
        {!! Form::email('email', null, ['class' => 'form-control',
'placeholder' => 'Email address', 'required', 'autofocus', 'id' =>
'inputEmail' ]) !!}

        <br />
        <button class="btn btn-lg btn-primary btn-block" type="submit">Send
 me a reset link</button>

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

@stop
```

All views related to login, register and passwords are in `/views/auth/` folder.

```
.form-signin {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
}
.form-signin .form-control {
    position: relative;
    height: auto;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    padding: 10px;
    font-size: 16px;
}
.form-signin .form-control:focus {
    z-index: 2;
}
```

Password Reset Form Page

This page is displayed when user clicks on reset link. They will have 2 fields for new password and password confirmation.

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

@section('head')
    {!! HTML::style('/assets/css/reset-form.css') !!}
@stop

@section('content')

        {!! Form::open(['url' => url('/password/reset/'), 'class' => 'form-
signin', 'method' => 'post' ] ) !!}

        @include('includes.errors')

        {{ csrf_field() }}

        <input type="hidden" name="token" value="{{ $token }}">

        <h2 class="form-signin-heading">Set New Password</h2>

        <label for="inputEmail" class="sr-only">Email address</label>
        {!! Form::email('email', null, [
            'class'                         => 'form-control',
            'placeholder'                   => 'Email address',
            'required',
            'id'                            => 'inputEmail'
            'autofocus'
        ]) !!}

        <label for="inputPassword" class="sr-only">Password</label>
        {!! Form::password('password', ['class' => 'form-control',
'placeholder' => 'Password', 'required',  'id' => 'inputPassword' ]) !!}
```

```
        <label for="inputPasswordConfirmation" class="sr-only">Password
Confirmation</label>
        {!! Form::password('password_confirmation', ['class' => 'form-
control', 'placeholder' => 'Password confirmation', 'required',  'id' =>
'inputPasswordConfirmation' ]) !!}


        <button class="btn btn-lg btn-primary btn-block"
type="submit">Change</button>

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

@stop
```

Reset form css looks like this:

```
.form-signin {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
}
.form-signin .form-control {
    position: relative;
    height: auto;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    padding: 10px;
    font-size: 16px;
}
.form-signin .form-control:focus {
    z-index: 2;
}
.form-signin input {
    margin-bottom: -1px;
    border-radius:0px;
}
.form-signin #inputPassword {
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
}
.form-signin #inputPasswordConfirmation {
    margin-bottom: 10px;
    border-bottom-left-radius: 4px;
    border-bottom-right-radius: 4px;
}
```

In code above I include `/views/includes/errors.blade.php` with `@include('includes.errors')` . As you guess, this file is in charge of displaying error messages, usually from validators.

```
@if(session()->has('errors'))
    <div class="alert alert-danger fade in">
        <button type="button" class="close" data-dismiss="alert" aria-hidden="true">×</button>
        <h4>Following errors occurred:</h4>
        <ul>
            @foreach($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Register Page

For register form I will use same form as for login but I will add a few more fields and add custom css. It is good idea to add same social button on this page so users don't need to click more times than needed to sign-in, that's why I moved social buttons to separate file.

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

@section('head')
    {!! HTML::style('/assets/css/register.css') !!}
@stop

@section('content')

        {!! Form::open(['url' => url('/register'), 'class' => 'form-
signin'] ) !!}

        @include('includes.errors')

        <h2 class="form-signin-heading">Please register</h2>

        <label for="inputEmail" class="sr-only">Email address</label>
        {!! Form::email('email', null, [
            'class'                         => 'form-control',
            'placeholder'                   => 'Email address',
            'required',
            'id'                            => 'inputEmail'
        ]) !!}

        <label for="inputFirstName" class="sr-only">First name</label>
        {!! Form::text('first_name', null, [
            'class'                         => 'form-control',
            'placeholder'                   => 'First name',
            'required',
            'id'                            => 'inputFirstName'
        ]) !!}

        <label for="inputLastName" class="sr-only">Last name</label>
        {!! Form::text('last_name', null, [
            'class'                         => 'form-control',
            'placeholder'                   => 'Last name',
            'required',
            'id'                            => 'inputLastName'
        ]) !!}
        <label for="inputPassword" class="sr-only">Password</label>
        {!! Form::password('password', [
            'class'                         => 'form-control',
            'placeholder'                   => 'Password',
            'required',
            'id'                            => 'inputPassword'
        ]) !!}


        <label for="inputPasswordConfirm" class="sr-only has-
warning">Confirm Password</label>
        {!! Form::password('password_confirmation', [
            'class'                         => 'form-control',
            'placeholder'                   => 'Password confirmation',
            'required',
            'id'                            => 'inputPasswordConfirm'
        ]) !!}

        <button class="btn btn-lg btn-primary btn-block register-btn"
type="submit">Register</button>

        <p class="or-social">Or Use Social Login</p>

        @include('partials.socials')

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


@stop
```

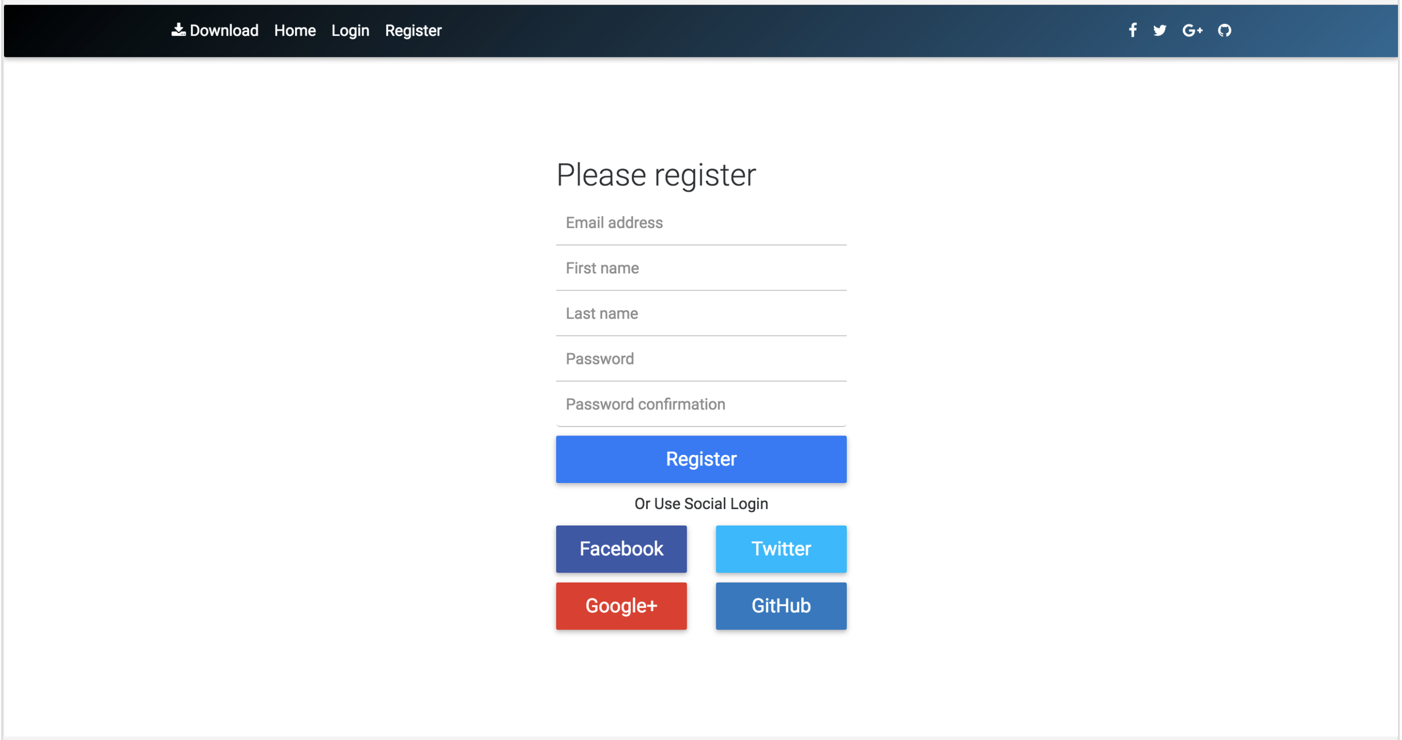
CSS code for register page:

```
.form-signin {
    max-width: 330px;
    padding: 15px;
    margin: 0 auto;
}
.form-signin .form-signin-heading,
.form-signin .checkbox {
    margin-bottom: 10px;
}
.form-signin .checkbox {
    font-weight: normal;
}
.form-signin .form-control {
    position: relative;
    height: auto;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    padding: 10px;
    font-size: 16px;
}
.form-signin .form-control:focus {
    z-index: 2;
}
.form-signin input[type="email"] {
    margin-bottom: -1px;
    border-bottom-right-radius: 0;
    border-bottom-left-radius: 0;
}
.form-signin input:not([type="email"]) {
    margin-bottom: -1px;
    border-radius:0px;
}
.form-signin #inputPasswordConfirm {
    border-bottom-left-radius: 4px;
    border-bottom-right-radius: 4px;
}
.register-btn{
    margin-top:10px;
}
.or-social{
    text-align:center;
    margin: 10px 0 10px 0;
}
.facebook{
    background-color: #4863ae;
```

```css
    border-color: #4863ae;
}
.facebook:hover{
    background-color: #2871aa;
    border-color: #2871aa;
}
.twitter{
    background-color: #46c0fb;
    border-color: #46c0fb;
}
.twitter:hover{
    background-color: #00c7fb;
    border-color: #00c7fb;
}
.g-recaptcha{
    margin-top:10px;
}
.google{
    background-color: #DD4B39;
    border-color: #DD4B39;
}
.google:hover{
    background-color: #e15f4f;
    border-color:#e15f4f;
}
.github{
    background-color: #4183C4;
    border-color: #4183C4;
}
.github:hover{
    background-color: #5490ca;
    border-color:#5490ca;
}
.error-text{
    color: #F44336;
    transition: .2s opacity ease-out,.2s color ease-out;
}
.margin-bottom-10{
    margin-bottom:10px;
}
```

You may notice that I am copying many same parts of css code and few pieces of UI over and over again. I am doing that intentionally, cause I am planning to refactor that code in next tutorial.

Now I will test how this register page looks in browser.



Register Page

With this register page all authentication pages are now completed. Home page, admin panel and user panel pages are just simple placeholder pages that are extending main layout view.

Create migrations and models related to users and roles

Earlier I mentioned that this app will have 2 user types: ordinary user and administrator. I could create more roles but this is basic thing and you could easily extend it. Laravel comes with basic user table migration, it is quite good but I'll add few columns to it, you will see later why. Also I will remove name column and instead of that add first_name and last_name columns.

```php
<?php

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_name');
            $table->string('last_name');
            $table->string('email')->unique()->nullable();
            $table->string('password', 60)->nullable();
            $table->rememberToken();
            $table->boolean('activated')->default(false);
            $table->string('token');
            $table->timestamps();
        });
    }

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

Now I will create tables related to roles, one will be named *roles* and other *role_user* commands are

`php artisan make:migration create-roles --create=roles`

`php artisan make:migration create-role-user --create=role_user`

```
<?php

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

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

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

```
<?php

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

class CreateRoleUser extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('role_user', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned()->index();

$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade')

            $table->integer('role_id')->unsigned()->index();

$table->foreign('role_id')->references('id')->on('roles')->onDelete('no
action');
            $table->timestamps();
        });
    }

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

As you can see role_user table is relationship table where I keep track of which role certain user has.

After this I am ready to run migrations `php artisan migrate`

Now I will create models for users and roles. Laravel does not posses dedicated folder for all models so I like to create it in *app\Models.* Laravel comes with default User model, and I will need to move it inside Models folder and update namespace to **App\Models**. I need basic relationship method, some function to check does user posses certain role, method to assign certain role to new user and method to remove role. At the end of this model I will add roles related logic.

Cause I have modified User model namespace, I need to update authentication configuration file so Laravel can locate new model. Configuration is located in config/auth.php look for key 'model' and change its value to `'App\Models\User'`.

```
    public function roles()
    {
        return $this->belongsToMany('App\Models\Role')->withTimestamps();
    }

    public function hasRole($name)
    {
        foreach($this->roles as $role)
        {
            if($role->name == $name) return true;
        }

        return false;
    }

    public function assignRole($role)
    {
        return $this->roles()->attach($role);
    }

    public function removeRole($role)
    {
        return $this->roles()->detach($role);
    }
```

Database seeders

First I will need to add basic 2 user roles to the database and after that I will create 2 users. I will assign different roles to them so it is important to first run Role seeder and only after that User seeder. All seeders are located inside `database/seeds/` directory.

```
<?php

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

class RoleSeeder extends Seeder{

    public function run(){
        DB::table('roles')->delete();

        Role::create([
            'name'    => 'user'
        ]);
        Role::create([
            'name'    => 'administrator'
        ]);

    }
}
```

To be able to create roles in this way using mass-assignment I will need to add *name* column to fillable array of Role model like this protected $fillable = ['name'];

```
<?php

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

class UserSeeder extends Seeder{

    public function run(){
        DB::table('users')->delete();

        $adminRole = Role::whereName('administrator')->first();
        $userRole = Role::whereName('user')->first();

        $user = User::create(array(
            'first_name'    => 'John',
            'last_name'     => 'Doe',
            'email'         => 'j.doe@codingo.me',
            'password'      => Hash::make('password'),
            'token'         => str_random(64),
            'activated'     => true
        ));
        $user->assignRole($adminRole);

        $user = User::create(array(
            'first_name'    => 'Jane',
            'last_name'     => 'Doe',
            'email'         => 'jane.doe@codingo.me',
            'password'      => Hash::make('janesPassword'),
            'token'         => str_random(64),
            'activated'     => true
        ));
        $user->assignRole($userRole);
    }
}
```

Now to include these seeder files into database seeder I need to call them from `DatabaseSeeder.php` file. You will find there commented example, just modify it and add RoleSeeder as first and UserSeeder as second call. After this I am ready to seed the database with `php artisan db:seed`

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Model::unguard();

         $this->call('RoleSeeder');
         $this->call('UserSeeder');

        Model::reguard();
    }
}
```

Middleware for administrator and user roles

When it comes to security I like to create simple solutions from default filters or use well-known packages like Sentry. For this I will modify existing Laravel's middleware located at `app\Http\Middleware\Authenticate.php`.

```
/**
 * Handle an incoming request.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Closure  $next
 * @param $role
 * @return mixed
 */
public function handle($request, Closure $next, $role)
{
    if(!$this->auth->check())
    {
        return redirect()->to('/login')
            ->with('status', 'success')
            ->with('message', 'Please login.');
    }

    if($role == 'all')
    {
        return $next($request);
    }
    if( $this->auth->guest() || !$this->auth->user()->hasRole($role))
    {
        abort(403);
    }
    return $next($request);
}
```

In this first if statement of handle method I am checking is user logged in at all, if not, user is redirected to login page with appropriate message. I added next if statement cause I have certain routes that are same for both types of users, so I want to handle them in one place in routes file. And last if statement is checking does user posses certain role that is needed if not, application aborts.

Routes and Auth Controller

After I have created middleware it is good time to implement routes for our app and use that middleware. I will use default `RegisterController` with slight modifications for handling user registrations via email.

```php
<?php

namespace App\Http\Controllers\Auth;

use Illuminate\Foundation\Auth\RegistersUsers;
use Illuminate\Support\Facades\Validator;
use App\Http\Controllers\Controller;
use App\Models\User;
use App\Models\Role;

class RegisterController extends Controller
{
    /*
    |----------------------------------------------------------------------
----
    | Register Controller
    |----------------------------------------------------------------------
----
    |
    | This controller handles the registration of new users as well as
their
    | validation and creation. By default this controller uses a trait to
    | provide this functionality without requiring any additional code.
    |
    */
```

```
use RegistersUsers;

/**
 * Where to redirect users after login / registration.
 *
 * @var string
 */
protected $redirectTo = '/';

/**
 * Create a new controller instance.
 *
 * @return void
 */
public function __construct()
{

    $this->middleware('guest');

}

/**
 * Get a validator for an incoming registration request.
 *
 * @param  array  $data
 * @return \Illuminate\Contracts\Validation\Validator
 */
protected function validator(array $data)
{

    return Validator::make($data,
        [
            'first_name'            => 'required',
            'last_name'             => 'required',
            'email'                 => 'required|email|unique:users',
            'password'              => 'required|min:6|max:20',
            'password_confirmation' => 'required|same:password'
        ],
        [
            'first_name.required'   => 'First Name is required',
                'last_name.required'     => 'Last Name is required',
                'email.required'         => 'Email is required',
                'email.email'            => 'Email is invalid',
                'password.required'      => 'Password is required',
                'password.min'           => 'Password needs to have at least
 6 characters',
                'password.max'           => 'Password maximum length is 20
characters'
            ]
            );

    }

    /**
     * Create a new user instance after a valid registration.
     *
     * @param  array  $data
     * @return User
     */
    protected function create(array $data)
    {

        $user =  User::create([
            'first_name' => $data['first_name'],
            'last_name' => $data['last_name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password']),
            'token' => str_random(64),
            'activated' => true
        ]);

        $role = Role::whereName('user')->first();
        $user->assignRole($role);

        return $user;

    }

}
```

Routes file `web.php` now looks like this:

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| This file is where you may define all of the routes that are handled
| by your application. Just tell Laravel the URIs it should respond
| to using a Closure or controller method. Build something great!
|
*/

Route::get('test', function () {
    return view('layouts.main');
});

$s = 'public.';
Route::get('/',          ['as' => $s . 'home',   'uses' =>
'PagesController@getHome']);

Route::group(['prefix' => 'admin', 'middleware' => 'auth:administrator'],
function()
{
    $a = 'admin.';
    Route::get('/', ['as' => $a . 'home', 'uses' =>
'AdminController@getHome']);

});

Route::group(['prefix' => 'user', 'middleware' => 'auth:user'], function()
{
    $a = 'user.';
    Route::get('/', ['as' => $a . 'home', 'uses' =>
'UserController@getHome']);
});

Route::group(['middleware' => 'auth:all'], function()
{
    $a = 'authenticated.';
    Route::get('/logout', ['as' => $a . 'logout', 'uses' =>
'Auth\LoginController@logout']);
});

Auth::routes(['login' => 'auth.login']);
```

As I mentioned before admin and user panel pages are basic placeholder pages and dedicated controllers `AdminController` and `UserController` are having only one method `getHome()` which returns respective view files. Same stands for home page, but I like to keep all public pages in separate controller.

At the end I create common filter for both user types, actually I don't have *all* user type as you saw previously in Authenticate.php middleware, I have hard-coded that value. Last line in routes file is registering Laravel default authentication routes, you can see all routes by executing:

```
php artisan route:list
```

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

class AdminController extends Controller {

    public function getHome()
    {
        return view('panels.admin.home');
    }
}
```

One more thing is needed for this to work, I need to update login form action url and top navigation links to proper routes. After this login and logout should work.

From RegisterControlller's validator method you cann see all validation rules, feel free to modify them according to your needs.

It is pretty obvious now that I am using only server side validation, which is only 50% of job. In next part of this tutorial I will work with one of the best JS validation libraries and validate input on client side.

In create method of same controller I am creating new user model and assiging default user role to it.

Password Reset

For sending password reset emails I am using AWS SES, everything that needs to be done is generating new set of API keys and inserting those values in *.env* file. Set of keys from *.env.example* file won't work cause I disabled them.

> Contacting 3rd party service like SES here, could cause delays. These delays are very annoying to end users, even if they take only 2-3 seconds. Checkout my other tutorial where I explain how you can code email queue Sending emails over Queue with AWS SES using this same codebase.

Create table and model for Social logins

I am planning to use users table for name and email data and put all social login related data into social_logins table. Socialite will return user unique social id and I will use that and social provider to determine is user new or existing one.

```
php artisan make:migration create-social-logins --create=social_logins
```

```php
<?php

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

class CreateSocialLogins extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('social_logins', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned()->index();

$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade'

            $table->string('provider', 32);
            $table->text('social_id');
            $table->timestamps();
        });
    }

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

After this I can run migration `php artisan migrate`

Provider will be string like: facebook, twitter, github etc. And socialite uses configurations stored in *config/services.php* file. Everything that is needed is creating new array element for each new social provider. This is a part of my *services.php* file:

```
    'facebook' => [
        'client_id'     => env('FB_ID'),
        'client_secret' => env('FB_SECRET'),
        'redirect'      => env('FB_REDIRECT')
    ],

    'twitter' => [
        'client_id'     => env('TW_ID'),
        'client_secret' => env('TW_SECRET'),
        'redirect'      => env('TW_REDIRECT')
    ],

    'google' => [
        'client_id'     => env('GOOGLE_ID'),
        'client_secret' => env('GOOGLE_SECRET'),
        'redirect'      => env('GOOGLE_REDIRECT')
    ],

    'github' => [
        'client_id'     => env('GITHUB_ID'),
        'client_secret' => env('GITHUB_SECRET'),
        'redirect'      => env('GITHUB_REDIRECT')
    ]
```

I am storing social app related data in *.env* file.

> Checkout Create your first Facebook application to see how I created new FB application and acquired these client_id and client_secret tokens.

Model for social logins is pretty simple, it has only one relationship for user.

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Social extends Model {

    protected $table = 'social_logins';

    public function user()
    {
        return $this->belongsTo('App\Models\User');
    }
}
```

Create Social Logic

When we want to authenticate user with one of Socialite providers, first we redirect that user to social site and after that social site redirects user back to our server with certain tokens. In the back Socialite contacts social site once more with those tokens and accepts user object if everything is OK. Here I will handle only social redirects when user allows our app to read their social data, in next tutorial I will cover cancelations.

I want this social logic to be very extensible so I can add new providers in matter of seconds. Because of that I will not hardcode any values in routes. These are my social routes:

```
$s = 'social.';
Route::get('/social/redirect/{provider}',   ['as' => $s . 'redirect',
'uses' => 'Auth\AuthController@getSocialRedirect']);
Route::get('/social/handle/{provider}',     ['as' => $s . 'handle',
'uses' => 'Auth\AuthController@getSocialHandle']);
```

So social buttons will use following links *[/social/redirect/facebook]* or *[/social/redirect/twitter]*. When user visits them, system will trigger Socialite redirection and it will get user object in return. You will notice redirect key for each social provider in *services.php,* that url social site will use as callback url. And in my system that route is *[/social/handle/facebook]* or *[/social/handle/twitter]*

Now you can see that adding new provider is matter of inserting new element in *services.php* and creating dedicated social button in views.

Here you can see entire code of `App\Http\Controllers\Auth\SocialController.php`:

```
<?php

namespace App\Http\Controllers\Auth;

use Laravel\Socialite\Facades\Socialite;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\Input;
use App\Http\Controllers\Controller;
use App\Models\Social;
use App\Models\User;
use App\Models\Role;

class SocialController extends Controller
{

    public function getSocialRedirect( $provider )
    {

        $providerKey = Config::get('services.' . $provider);
        if (empty($providerKey)) {

            return view('pages.status')
                ->with('error','No such provider');

        }

        return Socialite::driver( $provider )->redirect();

    }

    public function getSocialHandle( $provider )
    {

        if (Input::get('denied') != '') {

            return redirect()->to('/login')
                ->with('status', 'danger')
                ->with('message', 'You did not share your profile data with
our social app.');

        }

        $user = Socialite::driver( $provider )->user();

        $socialUser = null;

        //Check is this email present
        $userCheck = User::where('email', '=', $user->email)->first();

        $email = $user->email;

        if (!$user->email) {
            $email = 'missing' . str_random(10);
        }

        if (!empty($userCheck)) {

            $socialUser = $userCheck;
        }
        else {

            $sameSocialId = Social::where('social_id', '=', $user->id)
                ->where('provider', '=', $provider )
                ->first();

            if (empty($sameSocialId)) {

                //There is no combination of this social id and provider,
so create new one
                $newSocialUser = new User;
                $newSocialUser->email              = $email;
                $name = explode(' ', $user->name);

                if (count($name) >= 1) {
                    $newSocialUser->first_name = $name[0];
                }

                if (count($name) >= 2) {
                    $newSocialUser->last_name = $name[1];
                }

                $newSocialUser->password = bcrypt(str_random(16));
                $newSocialUser->token = str_random(64);
                $newSocialUser->save();

                $socialData = new Social;
                $socialData->social_id = $user->id;
                $socialData->provider= $provider;
                $newSocialUser->social()->save($socialData);

                // Add role
                $role = Role::whereName('user')->first();
                $newSocialUser->assignRole($role);

                $socialUser = $newSocialUser;

            }
            else {
                //Load this existing social user
                $socialUser = $sameSocialId->user;

            }

        }

        auth()->login($socialUser, true);

        if ( auth()->user()->hasRole('user')) {

            return redirect()->route('user.home');

        }

        if ( auth()->user()->hasRole('administrator')) {

            return redirect()->route('admin.home');

        }

        return abort(500, 'User has no Role assigned, role is obligatory!
You did not seed the database with the roles.');

    }
}
```

In `getSocialRedirect` method I am passing provider string as parameter and I am checking is that provider present in services. If it is present then system redirects user to social site.

In `getSocialHandle` I am catching data which social site sends to the server.

- First I am checking did user allowed our social app, or he denied access. If access is denied then user is redirected to login page with propper error message.
- After that I am grabbing user object from Socialite and checking is email present. If email is not present I am creating random string which starts with word missing.
- Later I am checking is user with same social id and provider present in social_logins table

    - If user is present I just login that user
    - Else I create that new user and associate respective social data to that account. I am also attaching user roles here.

This approach posses one downside, for example user can share partial data from his social profile (without email). So that's why I am adding random string to email field. I will add one simple trick in next tutorial for fixing this.

> I think this is very detailed for first tutorial. I would like to hear your feedback in comments below. Is this solving some of your application needs? Would you code something in different way?

Checkout second part of this project, where I talk about Laravel 5 client side validation with Parsley.js