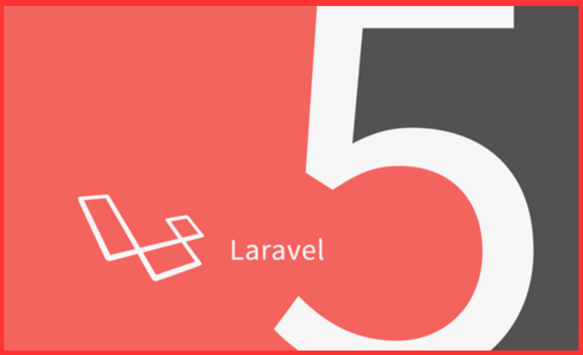
Week 9 Lab: Laravel 5.4 Install and Setup: ITEC Social APP



Step 1: Download and Install Composer: https://getcomposer.org/download/

Step 2: Download and install Artisan (Laravel's CLI):

Step 3: We are now ready to install Laravel. There are a number of different ways but lets use Composer to automate the process.

-In Windows go to the root of your server and click on the address bar
-Type in 'CMD' to open the Command Line Interface (CLI) at that location
-Type 'Composer' in the CLI to ensure it is working and check out the various options
-Lets use the 'create-project' command to download and install Laravel automatically:

```
COMPOSER CREATE-PROJECT LARAVEL/LARAVEL ITECKY “5.4.*”
```

(PACKAGE MANAGER) (MANAGER COMMAND) (PACKAGE NAME) (PROJECT NAME) (PACKAGE VERSION)

```
Or Keep it simple: laravel new itecky
```

Step 4: Once Laravel installs go ahead and open it up in Atom. Check out the file structure and pay attention to the most important parts of the MVC framework. In *APPS/HTTP* directory you will see the routes and controllers. Once created our Models will also reside in the APPS folder. The RESOURCES/VIEWS folder contains our view files.

Step 5: Lets set our application name space. In the CLI add the following command:

```
PHP ARTISAN APP:NAME ITECKY
```

Step 6: Now we are ready to get started. Open the project in your browser, you should see 'LARAVEL 5" in a light weight font. [NOTE: BY DEFAULT LARAVEL IS SERVED FROM THE PUBLIC DIRECTORY, THERE ARE WAYS TO CHANGE THIS BUT FOR NOW DON'T WORRY ABOUT IT] How is this being served? Well remember when you make a HTTP request to Laravel it is first handled by APP/HTTP/Routes.php. The Router may send the request directly to a View or to the controller and model for further processing before going to the View. Open the route.php file, you should see a default route for the welcome page set up:

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

Step 7: Go to the RESOURCES/VIEWS/WELCOME.BLADE.PHP file. This is the view being served by the route. Just to be sure go ahead and add some text to the view, save, and refresh the page. Pretty easy right? Setting up routes is very easy especially for static pages such as this. Go back to routes and create simple "hello world" route and then load it in your browser:

```
Route::get('hello-world', function () {
    return "<h1>Hello World!</h1>;
});
```

Step 8: This is just serving static text. Lets go ahead and create a view and serve it via the route. First in the Views folder create a file HELLOWORLD.BLADE.PHP and give it some HTML text to serve up. Next go back to the route.php file and point it towards the view, notice that you don't need to use the whole file name:

```
Route::get('hello-world', function () {
    return view('helloworld;);
});
```

Step 9: Enough playing around, lets get our app started! First lets create a HomeController. We could do this manually but lets us Artisan CLI because its much faster. In the CLI type:

```
PHP ARTISAN MAKE:CONTROLLER HOMECONTROLLER --PLAIN
```

The "--plain" flag just keeps the controller boilerplate code simple. Open the controller in Atom. Notice the OOP nature of our controllers "class HomeController extends Controller". Lets create a METHOD [remember methods are basically functions inside a class] to call a view when the route.php sends us a request:

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

If you now go to the home page nothing changes right?! Why? Becuase our route is still pointing at the welcome.blade.php view and we haven't created a home.blade.php view. So lets fix that.

Step 10: Go back to route.php and change the root route to point to the HomeController. There are two ways we can specify this:

```
-Route::get('/', 'HomeController@index');
---or---
-Route::get('/', [
  'uses' => 'HomeController@index',
  'as' => 'home'
]);
```

Both do the same thing basically. Don't worry about namespacing the HomeContoller location, we already took care of that in the HomeController. Notice how we are calling the index method in the HomeController with "'HomeController@index". You can have multiple methods within one controller and call them this way. If you want to confirm it is workings simply comment out the return view and type in: return "Everythings Working"; . Save and refresh, works right! Go ahead and delete that and uncomment “return view('home');”.

Passing Data to the View

We can pass data to the view in a number of different ways. Laravel can output our data as JSON data easily which makes it useful for RESTful APIs. Create an assocciative array with your name, age, and nationality. Then call the drop and die function:

```
  $ppl = [
    "name" => "Yasuko",
    "age" => 28,
    "nationality" => "Japan"
  ];
  dd($ppl);
```

You should see the output as JSON data. There are two other common ways to pass data to the view:
-Call the ->with() method,
-use the compact() function and include it in the view() as an argument
Create an about me page and pass it two arrays, one with info about you (name, age, nationality etc) and another with people or things you think are awesome. Make the first an associative array and the second a simple index array. Create the proper view and route to it using ABOUTME controller.

```
PHP ARTISAN MAKE:CONTROLLER AboutMe --PLAIN
```

In the about me View create two loops. The short hand in Blade templating engine it to use:

```
@foreach($ppl as $person)
 <h1>  {{ $person }} </h1>
@endforeach
```

The {{ }} escapes and echos the data.

Step 11: Ok lets create our master template and then feed the Home view to it. In views create master.blade.php:

```
<!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">
  <title>ITECKY</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.4.0/css/font-awesome.min.css">
 </head>
 <body>
   <div class="container">
    @yield('content')
   </div>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
 </body>
</html>
```

Notice the @yield('content') under the div.container? @yield tells Laravel that we want to *yield*, or include, content here. Lets make some content to yield in our home.blade.php file:

```
@extends('master')
@section('content')
  <h1>Welcome to iTECKY</h1>
@endsection
```

If you save and refresh your page you should now see the welcome message. If you have an error troubleshoot from route.php to the controller to the view. Always follow that troubleshooting request route and you will find your problem more quickly. Update the About Me page to use the new Master template file!

Step 12: Time to add some navigation. To keep things cleaner and more organized lets create a new folder in the views: "VIEWS\PARTIALS". We can put our footer and nav etc here. Inside PARTIALS create a new file "nav.blade.php".
`

Step 13: Migrate: Now lets create navbar with some sign in/up links etc. Paste in the nav below.

```
<nav class="navbar navbar-default" role="navigation">
      <div class="container">
            <div class="navbar-header">
                  <a href="" class="navbar-brand">iTECKY</a>
            </div>
            <div class="collapse navbar-collapse">
                  <!-- @if (Auth::check()) -->
                  <ul class="nav navbar-nav">
                        <li><a href="#">Timeline</a></li>
```

```
                <li><a href="#">Friends</a></li>
            </ul>
            <form action="" role="search" class="navbar-form navbar-left">
                <div class="form-group">
                    <input type="text" name="query" class="form-control"
                    placeholder="Find people"/>
                </div>
                <button type="submit" class="btn btn-default">Search</button>
            </form>
            <!-- @endif -->
            <ul class="nav navbar-nav navbar-right">
                <!--@if(Auth::check()) -->
                <li><a href="#">USER NAME<!-- {{Auth::user()->getNameOrUsername() }}
                --></a></li>
                <li><a href="#">Update profile</a></li>
                <li><a href="#">Sign out</a></li>
                <!-- @else -->
                <li><a href="#">Sign up</a></li>
                <li><a href="#">Sign in</a></li>
                <!-- @endif -->
            </ul>
        </div>
    </div>
</nav>
```

Now go back to the master.blade.php and include it using @include('partials.nav') just after the opening <body> tag. Refresh and confirm it is included. Update the Nav to link to the pages we've created thusfar.

Step 14: We are now ready to set up user accounts so we can add and authenticate users. To do this we need to create a users table in the database. If you haven't already create an empty database and put the connection information into the .ENV file in the root of the site. Now that we have a database we can create our users table. Instead of doing this manually though lets use Laravel's migration system to create and manage the table. If you go to the DATABASE/MIGRATIONS folder you will see there are already two default migrations. Let's delete those and create our own. After deleting them open up the CLI and use Artisan to create two new migrations:

```
PHP ARTISAN MAKE:MIGRATION CREATE_USERS_TABLE --CREATE="USER"

PHP ARTISAN MAKE:MIGRATION CREATE_USERS_TABLE --CREATE="ARTICLES"
```

If you now look in the migrations folder you should see our new migration. If you look in the DB there will be no tables because we haven't run the migration yet.

Step 16: When you look at the migration file you will a function up() and down() which just add and drop tables. Because we used Artisan our migration already has some default columns prepared, an auto-incrmenting ID and a timestamp:

```
Schema::create('user', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
```

We can add the fields we need for creating a user now. Add the following:

```
 Schema::create('user', function (Blueprint $table) {
            $table->increments('id');
            $table->string('email');
            $table->string('username');
            $table->string('password');
            $table->string('first_name')->nullable();
            $table->string('location')->nullable();
            $table->string('remember_token')->nullable();
            $table->timestamps();
        });
```

Lets do the same for our Articles table:

```
        Schema::create('article', function (Blueprint $table) {
            $table→increments('id');
           $table→string('title');
            $table->text('body');
            $table->timestamps();
        });
```

Step 17: We are ready to run our migration. In the CLI enter:

```
PHP ARTISAN MIGRATE
```

Check your database and ensure the table has been created.

Manually add a few articles to the articles table. Now lets set up a route to an Articles view and create the View. Using Laravel's query builder we can get the articles and sort them.

```
Route::get('/articles', function () {
  $articles = DB::table('articles')->oldest()->get();
  return view('articles.index', compact('articles'));
});
```

Now in the the Article view call the master template and loop trough and output the title of the articles.

```
    @foreach ($articles as $task)
     <h2>{{ $articles->title }}</h2>
     <p>{{ $articles->body }}</p>
    @endforeach
```

OK thats good but how can we actually navigate to the articles themselves? We need some kind of slug to identify the article. Lets wrap the H2 in an anchor tag and use the id as a slug:

```
<a href="articles/{{ $articles->id }}"> ..... </a>
```

If we follow the links however we get an error, since we dont have a routing system for our articles. Lets create a new route to the individual articles. We will pass the 'id' slug as the identifier and then use that in the Query Builder 'find()' method.

```
Route::get('/articles/{id}', function($id) {
  $article = DB::table('articles')->find($id);
  return view('articles .article', compact('article'));
});
```

Finally we just need to build the single article view and output the data. Try to do that on your own, should be easy!

Now our Routes are getting a little messy with logic that should be in the controllers. So lets clean our routes up and move the DB queries to an articles controller. First create the controller:

```
PHP ARTISAN MAKE:CONTROLLER ARTICLECONTROLLER -r
(dash r creates a “resourceful” controlled with many helper methods)
```

Next we can

Step 18: We are ready to set up user authentication and allow them to create accounts. Lets create controller to handle this process. In Artisan create a new controller:

```
PHP ARTISAN MAKE:CONTROLLER AUTHCONTROLLER --PLAIN
```

Lets now set up a get and post method within our new controller as well as point our router to it. First lets start with the GET request. Create a method the returns a view:

```
public function getSignup() {
    return view('auth.signup');
}
```

In route.php add the route:

```
Route::get('signup', [
  'uses' => '\Chatty2\Http\Controllers\AuthController@getSignup',
  'as' => 'auth.signup',
]);
```

Our route and controller are now set up but if you try to access 'signup' it triggers an error, thats because we need to create a view. In the views folder create a subfolder and file: [views/auth/signup.blade.php]. Add the boilerplate code below:

```
@extends('master')
@section('content')
 <div class="row">
  <div class="col-md-6">
    <form class="form-vertical" role="form" method="post" action="{{ route('auth.signup') }}">
      <div class="form-group {{ $errors->has('email') ?
      ' has-error' : '' }}">
        <label for="email" class="control-label">Your email address</label>
        <input type="text" name="email" class="form-control" id="email" value="{{ Request::old('email') ?: '' }}">
        @if ($errors->has('email'))
         <span class="help-block">{{ $errors->first('email') }}</span>
```

```
            @endif
          </div>
          <div class="form-group {{ $errors->has('username') ?
          ' has-error' : '' }}">
            <label for="username" class="control-label">Choose a username</label>
            <input type="text" name="username" class="form-control" id="username" value="">
            @if ($errors->has('username'))
              <span class="help-block">{{ $errors->first('username') }}</span>
            @endif
          </div>
          <div class="form-group {{ $errors->has('password') ?
          ' has-error' : '' }}">
            <label for="password" class="control-label">Choose a password</label>
            <input type="password" name="password" class="form-control" id="password" value="">
            @if ($errors->has('password'))
              <span class="help-block">{{ $errors->first('password') }}</span>
            @endif
          </div>
          <div class="form-group">
            <button type="submit" class="btn btn-default">Sign up</button>
          </div>
  <input type="hidden" name="_token" value="{{ Session::token() }}">
        </form>
    </div>
</div>
@endsection
```

Step 19: We can now visit the signup page but we need to handle the POST request from the form. Lets go ahead and create that route:

```
Route::post('signup', [
  'uses' => '\Chatty2\Http\Controllers\AuthController@postSignup'
]);
```

Now set up the controller to handle the post in AuthController.php. First lets validate the input:

```
  public function postSignup(Request $request) {
      $this->validate($request, [
        'email' => 'required|unique:user|email|max:55',
        'username' => 'required|unique:user|alpha_dash|max:20',
        'password' => 'required|min:5',
      ]);
```

Next lets actually hand off our input to the database and then redirect to the homepage with a success message. Inside the postSignup method, below the validation add: [1]

```
  User::create([
        'email' => $request->input('email'),
```

1**bcrypt** is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher, and presented at USENIX in 1999. https://en.wikipedia.org/wiki/Bcrypt

```
        'username' => $request->input('username'),
        'password' => bcrypt($request->input('password')),
      ]);

      return redirect()->route('home')->with('info', 'Your account has been created. Welcome to
iTECKY!!!');
```

We need to add the alert to our home template so lets create an alerts.blade.php in our view/partials folder. Add the code for the view:

```
@if (Session::has('info'))
.<div class="alert alert-info">
    {{Session::get('info')}}
  </div>
@endif
```

Then add the include to the master template where you want the notification to appear:

```
        @include('templates/partials/alerts')
```

Finally lets set up our model. Fortunately Laravel provides us a User.php model to work with out of the box. Open the file and check out the structure. Update the “fillable” property/variable so we can enter our login data using the form:

```
protected $fillable = ['username', 'email', 'password'];
```

In our AuthController.php we need to tell the controller to use this model:

```
use Chatty\Models\User;
```

Step 20: We now have our signup page but for existing users we need to allow them to sign in so lets create that page: signin.blade.php

```
@extends('master')
@section('content')
<div class="row">
      <div class="col-lg-6">
```

```
        <form class="form-vertical" role="form" method="post" action="#">
            <div class="form-group">
                <label for="email" class="control-label">Email</label>
                <input type="text" name="email" class="form-control" id="email">
            </div>
            <div class="form-group">
                <label for="password" class="control-label">Password</label>
                <input type="password" name="password" class="form-control" id="password">
            </div>
            <div class="checkbox">
                <label>
                    <input type="checkbox" name="remember"> Remember me
                </label>
            </div>
            <div class="form-gorup">
                <button type="submit" class="btn btn-default">Sign in</button>
            </div>
        </form>
    </div>
</div>
@endsection
```

Go ahead and add the Routes to the Signin controller:

```
Route::get('signin', [
  'uses' => '\Chatty2\Http\Controllers\AuthController@getSignin',
  'as' => 'auth.signin',
]);

Route::post('signin', [
  'uses' => '\Chatty2\Http\Controllers\AuthController@postSignin'
]);
```

Update the Signin nav link in the nav.blade.php. Also add a session token input to the singin.blade.php view and also check the form action and update it to the proper route:

- `<input type="hidden" name="_token" value="{{ Session::token() }}">`
- `{{ route('auth.signin') }}`

Step 20: Now set up the controller to handle the post request information for authentication:

```
   public function getSignin() {
        return view('auth.signin');
   }

  public function postSignin(Request $request) {
    $this->validate($request, [
      'email' => 'required',
      'password' => 'required',
    ]);

    if (!Auth::attempt($request->only(['email', 'password']), $request->has('remember'))) {
      return redirect()->back()->with('info', 'Please check your credentials and try again!');
    }

    return redirect()->route()->with('info', 'You are now signed in!');
  }
```