# Migration Guide

Laravel has release cycle ever six months. Probably in a year there would be 2 versions release, one around June and another at the end of the year. When I was recording this course I was using Laravel version 5.6, and now the current version is 5.8.

It's hard to make updates the course every time a new version is released. That's why I made this guide so that new students could follow along all course materials using the latest version of Laravel without any problems. I'll do my best to update this guide once new version of Laravel release.

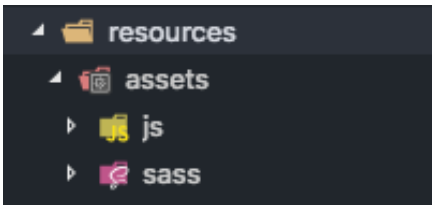
You can **download the complete guide** in the *course resource* and save it as a reference. You don't need to read the guide entirely, just read quickly. And if you find issue or find something different when following a lesson you can go back to the guide and hopefully you'll find the solution here.

But, if you can't find any solution, then you can go to **Q/A** section in Udemy and find any discussion that similar with problem you face. And just in case you can't find anything, then you can post your question there and I'll do my best to help you.



## Asset Directory Flatterned

Before Laravel 5.7 you will find `assets` sub-directory inside `resources` directory that house the script and style files like this:

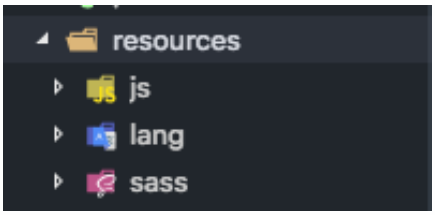



If you open the `webpack.mix.js` file you'll see these lines:

```
mix.js('resources/assets/js/app.js', 'public/js')
   .sass('resources/assets/sass/app.scss', 'public/css');
```

These lines basically tell Laravel Mix to compile the script & sass files and produce normal javascript and css file in `js` and `css` respectively.

But since Laravel 5.7 that directory has been flatterned into `resources` directory like this:




And If you open the `webpack.mix.js` file you'll find a bit changes in javascript and sass paths.

```
mix.js('resources/js/app.js', 'public/js')
   .sass('resources/sass/app.scss', 'public/css');
```

# Email Verification (Optional)

Laravel 5.7 has introduced Email verification and Account activation out of the box. These features force registered users to activate their accounts by hitting the verification code that sent to their emails. As developer we can restrict that only verified users may access a given route.

Now when you run this command `php artisan make:auth` you'll find new view file inside `resources/views/auth` called **verify.blade.php**. If you open this file it will look like this:

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

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">{{ __('Verify Your Email Address')
}}</div>

                <div class="card-body">
```

```
                    @if (session('resent'))
                        <div class="alert alert-success" role="alert">
                            {{ __('A fresh verification link has been sent to
your email address.') }}
                        </div>
                    @endif

                    {{ __('Before proceeding, please check your email for a
verification link.') }}
                    {{ __('If you did not receive the email') }}, <a href="{{
route('verification.resend') }}">{{ __('click here to request another') }}
</a>.
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

If you want to use these feature you can follow these steps

## *Step 1 - Implement `mustVerifyEmail` interface in the User model*

Open your `User` model and implement the `mustVerifyEmail` interface like this:

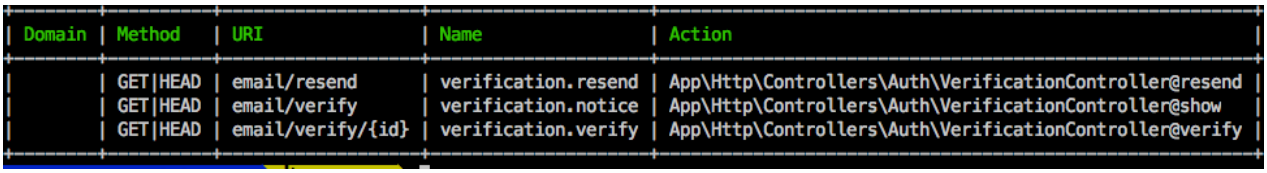
```
class User extends Authenticatable implements MustVerifyEmail
{
    use Notifiable;
    ...
}
```

## *Step 2 - Enable Verification Route*

Open your `routes/web.php` and specify `verify` option in `Auth::routes` as follow:

```
Auth::routes(['verify' => true]);
```

Now if you go to your terminal and type `php artisan route:list --name=verifycation` you'll see new routes added.



| Domain | Method | URI | Name | Action |
|---|---|---|---|---|
| | GET\|HEAD | email/resend | verification.resend | App\Http\Controllers\Auth\VerificationController@resend |
| | GET\|HEAD | email/verify | verification.notice | App\Http\Controllers\Auth\VerificationController@show |
| | GET\|HEAD | email/verify/{id} | verification.verify | App\Http\Controllers\Auth\VerificationController@verify |

## *Step 3 - Protect your route*

Now you can protect particular route(s) by using `verified` middleware whether in `web.php` or in your controller like this:

```
// web.php
Route::middleware('verified')->group(function () {
  // Put your routes here
});

// In controller you can call the verified middleware in the constructor
public function __construct()
{
    $this->middleware(['auth', 'verified']);
}
```

## *Step 4 - Setup Email Credentials*

Before you go try the email verification functionality you need to setup your email credential. If you missed this step you will got error when you try registering new user.

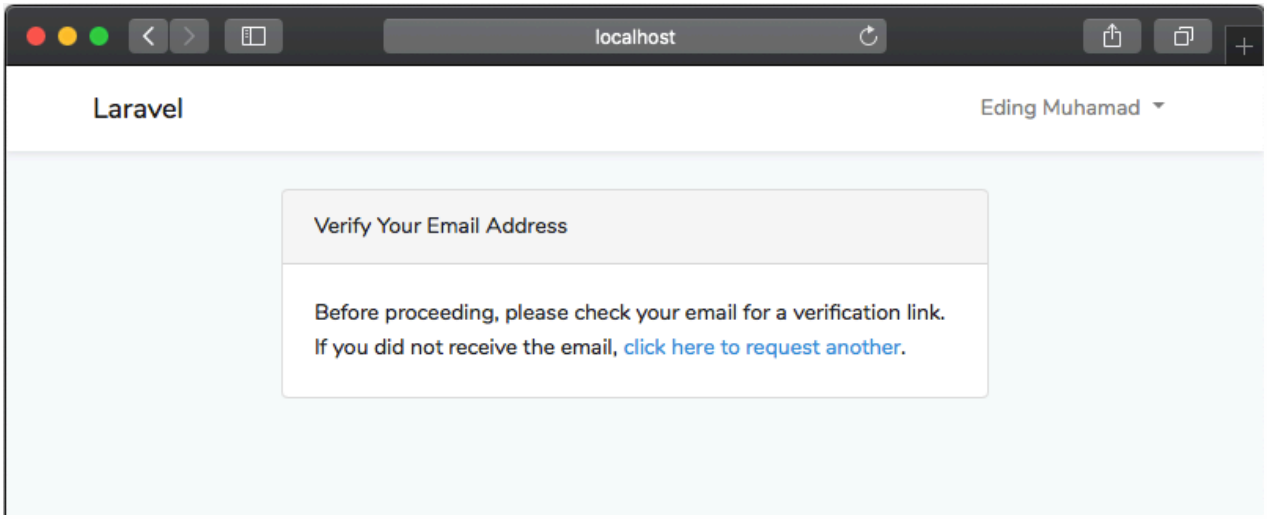
To do this You can simply use **mailtrap.io** to capture your email during development. Just register an account and once you've done that you can go to your inbox, copy the credentials and paste to your `.env` file.

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=your-user-email
MAIL_PASSWORD=your-password-email
MAIL_ENCRYPTION=null
```

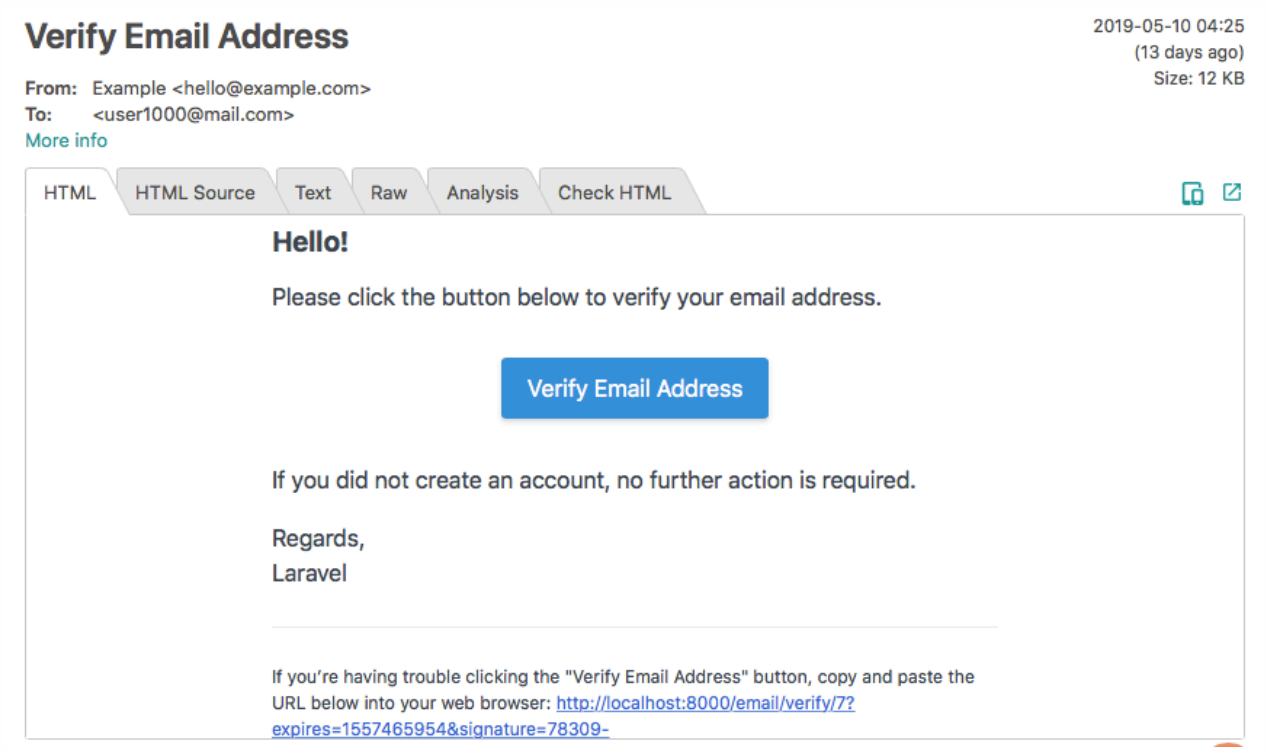
## *Step 5 - Test Email Verification functionality*

You can test the Email Verification functionality by hitting the `register` button or directly enter `http://localhost:8000/register`.

If you protected your `home` route for example (see step 3), once the account registered you'll see the *Verify Yor Email Address* message like this:




If you open your inbox in mailtrap, you'll find verification email comes in. You can then hit the **Verify Email Address** button to verify your account.




# Migrations & bigIncrements

In prior version of Laravel (< 5.8) if you're using `php artisan make:migration create_x_table` command or `php artisan make:model TheModel -m` You'll find in your migration file two automatically added inside table definition like this:

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

What I want to highlight here is `increments` method. This method will generates `id` column as `UNSIGNED INTEGER PRIMARY KEY AUTO_INCREMENT` (in mysql database).

Now in Laravel version 5.8 there's a bit change in id column definition. It's now `bigIncrements` which will generate `id` column as `UNSIGNED BIG INTEGER PRIMARY KEY AUTO_INCREMENT`.

Probabbly it's not a big deal when you're not working with **foreign key** columns. But when you're working with them, this will be a big problem because both columns (primary column and foreign column) should has the same type.

Suppose you have `create_categories_table` migration like this:

```
class CreateCategoriesTable extends Migration
{
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('title');
            $table->timestamps();
        });
    }

    ...
}
```

And then you also have `create_posts_table` migration which look like this:

```
class CreatePostsTable extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('title');
            $table->text('body');
            $table->unsignedInteger('category_id'); // <-- the issue
            $table->foreign('category_id')->references('id')-
>on('categories');
            $table->timestamps();
        });
    }
```

```
    ...
}
```

If you run `php artisan migrate` then the `create_posts_table` migration file will be failed because the foreign key (`category_id` column) is not the same type.

So there are two possible ways that you can tackle:

1. Make both columns as `UNSIGNED INTEGER`
2. Make both columns as `UNSIGNED BIG INTEGER`

## *1. Make both columns in `UNSIGNED INTEGER`*

You can make both columns as `UNSIGNED INTEGER` as follow:

```php
// CreateCategoriesTable
Schema::create('categories', function (Blueprint $table) {
  $table->increments('id'); // <--
  $table->string('title');
  $table->timestamps();
});
// CreatePostsTable
Schema::create('posts', function (Blueprint $table) {
  $table->bigIncrements('id');
  $table->string('title');
  $table->text('body');
  $table->unsignedInteger('category_id');  // <--
  $table->foreign('category_id')->references('id')->on('categories');
  $table->timestamps();
});
```

## *2. Make both column in UNSIGNED BIG INTEGER*

You can make both columns as `UNSIGNED BIG INTEGER` as follow:

```php
// CreateCategoriesTable
Schema::create('categories', function (Blueprint $table) {
  $table->bigIncrements('id'); // <--
  $table->string('title');
  $table->timestamps();
});
// CreatePostsTable
```

```
Schema::create('posts', function (Blueprint $table) {
  $table->bigIncrements('id');
  $table->string('title');
  $table->text('body');
  $table->unsignedBigInteger('category_id'); // <--
  $table->foreign('category_id')->references('id')->on('categories');
  $table->timestamps();
});
```

## String & Array Classes

Since Laravel 5.8 as noted in Laravel official documentation that all `array_*` and `str_*` global helpers have been deprecated. You should use the `Illuminate\Support\Arr` and `Illuminate\Support\Str` methods directly.

For example, in prior persion you used `str_slug` to generate sluggable string like this:

```
str_slug("Lorem ipsum dolor"); // lorem-ipsum-dolor
```

Now in Laravel 5.8 you could use `Str::slug` like this:

```
use Illuminate\Support\Str;
Str::slug("Lorem ipsum dolor"); // lorem-ipsum-dolor
```

If you still prefer using those helpers in your Laravel newer version you need to install laravel/helpers package.

```
composer require laravel/helpers
```

## Default Password Length

In Laravel version 5.8 the minimum password length was change to *eight* characters.

```
protected function validator(array $data)
{
  return Validator::make($data, [
    ...
    'password' => ['required', 'string', 'min:8', 'confirmed'],
  ]);
}
```

This change also effect the default password in `database/factories/UserFactory.php` from `secret` to `password`

```
$factory->define(User::class, function (Faker $faker) {
    return [
        'name' => $faker->name,
        'email' => $faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'password' =>
'$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
        'remember_token' => Str::random(10),
    ];
});
```

# Working with Vue.js Component

There are two common ways to load components in Vue.js.

- Using *ES6-style* `import` statement
- Using *CommonJS-style* `require` statement.

Since *vue-loader* version 13.0.0 there's a bit changes when using `require` statement. You need to chain in the `.default` on the end of the function call.

```
// before
const Foo = require('./Foo.vue')

// after
const Foo = require('./Foo.vue').default
```

# Fontawesome 5 issue

When recording this course I was using fontawesome 5.1.0. And now the current version is 5.8.2. So there's a bit changes on how to install and use this font as follow

## *Step 1 - Install font awesome package*

Open up your terminal and type this command:

```
// Install the base package
npm install @fortawesome/fontawesome-svg-core -D

// install solid icons
npm i -S @fortawesome/free-solid-svg-icons -D

// install regular icons (optional)
npm i -S @fortawesome/free-regular-svg-icons -D

// install brand icons (optional)
npm i -S @fortawesome/free-brands-svg-icons -D
```

## *Step 2 - Load the font types*

In your javascript file you can load some fonts that you need like like this:

```
import { config, library, dom } from '@fortawesome/fontawesome-svg-core';
config.autoReplaceSvg = 'nest';

import { faCaretUp,
         faCaretDown,
         faStar,
         faCheck } from '@fortawesome/free-solid-svg-icons';

library.add(faCaretUp,
            faCaretDown,
            faStar,
            faCheck);

// Kicks off the process of finding <i> tags and replacing with <svg>
dom.watch();
```

## *Step 3 - Using the font*

You can use the font that you have defined in your view by using `<i class="fa font-type"></i>`. Please take a look this code:

```
<i class="fa fa-caret-up"></i>
```