**Laravel Queues**

Laravel Queues provide integration of a variety of different queue backends like Beanstalkd, Amazon SQS, Redis, synchronous (sync) and database. You can find the queue configurations in the config/queue.php file. In this file, we will define the connection’s configurations for each queue drivers. You can check the [official documentation](https://laravel.com/docs/5.7/queues) for more details.

**Requirements**

We will need a working Laravel Application to complete this tutorial. You can start a new Laravel project by running below command in terminal.

composer create-project laravel/laravel LaravelQueuesTutorial

In this post’s example, we are going to send emails from our application using queues. As we know, sending emails is a time-consuming task, so I’m going to use **sending email as an example**.

I alreay wrote an article on the [Laravel Mailable](https://www.larashout.com/sending-emails-using-laravel-mailables) which will help you to understand how to send an email in Laravel application.

**Database Setup for Laravel Queues**

We will use the **database** driver to process our queues, but you can use Amazon SQS or Redis if you have that setup already. I will stick to **database**.

Before using the Laravel Queues, we need to make a jobs table in the database to store all queues. Laravel provides the table creation command out of the box, so go to your terminal and run the below command.

php artisan queue:table

This command will create a migration file in the database/migrations folder. The newly created file will contain the schema for the jobs table which we need to process the queues.

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateJobsTable extends Migration

{

*/\*\**

*\* Run the migrations.*

*\**

*\* @return void*

*\*/*

public function up()

{

Schema::create('jobs', function (Blueprint $table) {

$table->bigIncrements('id');

$table->string('queue')->index();

$table->longText('payload');

$table->unsignedTinyInteger('attempts');

$table->unsignedInteger('reserved\_at')->nullable();

$table->unsignedInteger('available\_at');

$table->unsignedInteger('created\_at');

});

}

*/\*\**

*\* Reverse the migrations.*

*\**

*\* @return void*

*\*/*

public function down()

{

Schema::dropIfExists('jobs');

}

}

Now it’s time to run the migrate command which will create the jobs table in database. Run below command to migrate:

php artisan migrate

Once your database is updated with migrate command, you will find the newly created table called jobs in the database.

Now we need to update the current queue drive in our environment file. Open .env file and change the queue drive to the database like below.

QUEUE\_DRIVER=database

**Creating Mail And the View For Email**

Next, we need to create a Laravel Mail which will just return a simple view with a welcome message. Run the following command to create a mail.

php artisan make:mail WelcomeEmail

Once the above command finishes, a new folder with name **Mail** along with a **WelcomeEmail** class file will be generated in the **app** folder.

You can read more hrer about [Laravel Mailable](https://www.larashout.com/sending-emails-using-laravel-mailables).

namespace App\Mail;

use Illuminate\Bus\Queueable;

use Illuminate\Mail\Mailable;

use Illuminate\Queue\SerializesModels;

use Illuminate\Contracts\Queue\ShouldQueue;

class WelcomeEmail extends Mailable

{

use Queueable, SerializesModels;

*/\*\**

*\* Create a new message instance.*

*\**

*\* @return void*

*\*/*

public function \_\_construct()

{

*//*

}

*/\*\**

*\* Build the message.*

*\**

*\* @return $this*

*\*/*

public function build()

{

return $this->view('mail.welcome');

}

}

Next, we will update the build() method so it can return our email view mail.welcome.

**Creating Email Template**

For creating a email template, create a new folder inside **views** folder with name **mail**and then create a new file called welcome.blade.php.

This file will container the following simple template:

<h1>Welcome To Our App</h1>

<p>You are welcome to our platform.</p>

The view of the email is now complete, next we will create a new queue that will send this email.

**Create Queue Job**

Now, run the below command to generate a new queue job.

php artisan make:job SendWelcomeEmail

When you run this command, a new folder with name **Jobs** will be created inside **app**folder along with the **SendWelcomeEmail** job class.

namespace App\Jobs;

use Illuminate\Bus\Queueable;

use Illuminate\Queue\SerializesModels;

use Illuminate\Queue\InteractsWithQueue;

use Illuminate\Contracts\Queue\ShouldQueue;

use Illuminate\Foundation\Bus\Dispatchable;

class SendWelcomeEmail implements ShouldQueue

{

use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

*/\*\**

*\* Create a new job instance.*

*\**

*\* @return void*

*\*/*

public function \_\_construct()

{

*//*

}

*/\*\**

*\* Execute the job.*

*\**

*\* @return void*

*\*/*

public function handle()

{

*//*

}

}

Now we will update this class to send our email. Firstly we need to add the **Mail** and **SendWelcomeEmail** namespaces in it.

use Mail;

use App\Mail\WelcomeEmail;

Next, we need to setup email sending process inside the handle() method.

*/\*\**

*\* Execute the job.*

*\**

*\* @return void*

*\*/*

public function handle()

{

$email = new WelcomeEmail();

Mail::to('info@larashout.com')->send($email);

}

In handle function we have created a new instance of **WelcomeEmail** and used the **Mail**facade to send the email. Simple enough.

Our queue job is ready now to be tested. Normally, in a real-world scenario, we would like to dispatch this job within a controller when a certain action is completed. Like when a new user registers on our website we want to send email using the queue job.

I want to keep this tutorial simple just to prove the concept. So I will create a new route and controller for testing this job. When we will visit that route a new job will be created in the jobs table and will be processed on the time we specify.

**Setup Route**

For testing, you will need to add the below route in your routes/web.php file.

Route::get('test-email', 'JobController@processQueue');

**Create Controller**

Now, we will create a new controller named **JobController** which will have a processQueue method to dispatch the queue. Run below artisan command to generate this controller.

php artisan make:controller JobController

Now update your **JobController** with below code to process the queue.

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Jobs\SendWelcomeEmail;

class JobController extends Controller

{

*/\*\**

*\* Handle Queue Process*

*\*/*

public function processQueue()

{

$emailJob = new SendWelcomeEmail();

dispatch($emailJob);

}

}

Now if you visit /test-email, a new job will insereted in the jobs table.

**Running Queue Worker**

The last thing to process this queue, you have to start the process of jobs by running below command in the terminal and leave it running.

php artisan queue:work

When your job will be processed, you will be able to see the status which will change from processing to processed with job name.

**Note**

If you want to process all your jobs when your application is in the production mode, add the above command to crob jobs.

**Delay Dispatch**

If you want to delay a job, means want to send the email after a certain time, you can use the delay() method by passing the time you want to delay like below:

namespace App\Http\Controllers;

use Carbon\Carbon;

use Illuminate\Http\Request;

use App\Jobs\SendWelcomeEmail;

class JobController extends Controller

{

*/\*\**

*\* Handle Queue Process*

*\*/*

public function processQueue()

{

$emailJob = new SendWelcomeEmail()->delay(Carbon::now()->addMinutes(5));

dispatch($emailJob);

}

}

In the above code example, we are adding **Carbon** package, which is used to work with date and time. Carbon package comes with Laravel out of box. So just use it.

We are delaying the queue for five minutes using delay() method.

Now, execute the queue:work command in terminal and leave the terminal open for a while. After five minutes Laravel will process this queue and this job will be removed from the jobs table in the database and your email will be sent.

**Note**

Before running the queue:work command, you have to setup the main configuration in .env file, for testing purpose you can set the mail dirver to **log** like this MAIL\_DRIVER=log. When Laravel queue will process the job and send the email look into storage/logs/laravel.log file and you will find your email there.

**Conclusion**

In this post, I tried my best to keep things simple so you can have a better understanding of how Laravel Queues work. I hope if you have followed all the above steps you will have a clear understanding. Make sure run the queue:work command on the server. Otherwise, all of the queue jobs stored in the database will not be processed.