

INNOVACIONES REALIZADAS

GTV



DAVID ALEGRÍA ILLÁN

ALBERTO CAPARRÓS RUIZ

DAVID IBAÑEZ NAVARRO

CARLOS GUTIÉRREZ GARCÍA

JUAN MANUEL DELGADO BARRANCO

Índice

1. Testing de componentes de Livewire	3
2. Uso de colas con RabbitMQ	6
3. Notificaciones por email con Mailtrap.io	15
4. Logging de eventos	21

1. Testing de componentes de Livewire

En el desarrollo de este proyecto, ya que los controladores/componentes han sido creados usando Livewire, se ha optado por realizar el testing usando la propia librería que incluye una suite de testing muy completa además de la ya incorporada por PHPUnit. A continuación, se detalla cómo realizar un test para comprobar la creación de un vídeo.

```
public function testICanCreateAVideo()
{
    $adminUser = $this->createAdmin();
    $place = $this->createPlace();
    $pointOfInterest = $this->createPointOfInterest($place->id);
    $thematicArea = $this->createThematicArea($pointOfInterest->id);

    $this->actingAs($adminUser);

    $this->assertDatabaseCount( table: 'videos', count: 0);
    $this->assertDatabaseCount( table: 'video_items', count: 0);

    $file = UploadedFile::fake()->create( name: 'video.mp4', kilobytes: 1, mimeType: 'video/mp4');


    Livewire::test( name: CreateVideo::class)
        ->set('createForm.file', $file)
        ->set('createForm.pointOfInterest', $pointOfInterest->id)
        ->set('createForm.thematicArea', $thematicArea->id)
        ->set('createForm.description', 'Video description')
        ->call( method: 'save');

    $this->assertDatabaseCount( table: 'videos', count: 1);
    $this->assertDatabaseCount( table: 'video_items', count: 1);

    $this->assertDatabaseHas( table: 'videos', [
        'point_of_interest_id' => $pointOfInterest->id,
        'order' => 1,
        'creator' => $adminUser->id,
        'updater' => null,
        'thematic_area_id' => $thematicArea->id,
        'description' => 'Video description',
    ]);

    $createdVideo = Video::first();

    $this->assertDatabaseHas( table: 'video_items', [
        'video_id' => $createdVideo->id,
    ]);
}
```



En primer lugar, para crear un vídeo necesitamos un usuario, un lugar, un punto de interés y un área temática. Para crear cada uno hemos usado los test helpers que hemos desarrollado, de modo que reducimos líneas de código en cada test.

En segundo lugar, usamos `actingAs()` para simular que estamos logueados como ese usuario y con `assertDatabaseCount()` nos aseguramos de que en la BBDD no existe ningún registro en las tablas `videos` y `video_items`. Usando `UploadedFile::fake()` generamos un archivo de vídeo falso que necesitaremos para crear la entidad vídeo.

En tercer lugar, comenzamos el test de Livewire especificando la clase que queremos probar y con el método `set()` vamos dando a cada variable necesaria para crear un vídeo su valor correspondiente. Después llamamos al método `call()` con el nombre de la función que crea el vídeo en `CreateVideo` (en este caso es `save`) para que se ejecute.

Finalmente, comprobamos que existe un registro en las tablas `videos` y `video_items` con `assertDatabaseCount()` y nos aseguramos de que cada una contiene los datos que hemos dado a la clase con `assertDatabaseHas()`.

En el caso que nos encontramos en un test de validación de un campo cuando creamos un vídeo, podemos darle un valor erróneo al campo que estamos comprobando y usar `assertHasErrors()` para ver que no pasa la validación. A continuación se muestra un test de ejemplo.

```

public function testThematicAreaIsRequiredWhenCreatingAVideo()
{
    $adminUser = $this->createAdmin();
    $place = $this->createPlace();
    $pointOfInterest = $this->createPointOfInterest($place->id);
    $this->createThematicArea($pointOfInterest->id);

    $this->actingAs($adminUser);

    $this->assertDatabaseCount( table: 'videos', count: 0);
    $this->assertDatabaseCount( table: 'video_items', count: 0);

    $file = UploadedFile::fake()->create( name: 'video.mp4', kilobytes: 1, mimeType: 'video/mp4');

    Livewire::test( name: CreateVideo::class)
        ->set('createForm.file', $file)
        ->set('createForm.pointOfInterest', $pointOfInterest->id)
        ->set('createForm.thematicArea', '')
        ->set('createForm.description', 'Video description')
        ->call( method: 'save')
        ->assertHasErrors(['createForm.thematicArea' => 'required']);

    $this->assertDatabaseCount( table: 'videos', count: 0);
    $this->assertDatabaseCount( table: 'video_items', count: 0);
}

```

2. Uso de colas con RabbitMQ

Supongamos que nuestra aplicación fuera usada por cientos de usuarios por minuto. Cada vez que uno realiza una petición, debería esperar a que llegue a su turno en el servidor, se procese y le devuelva una respuesta. En momentos de mucho tráfico un usuario tendría que esperar más de lo normal por lo que tendría una mala experiencia de usuario al usar nuestra aplicación, ya que las peticiones se procesan de forma síncrona.

Para solventar este problema podríamos usar un sistema de colas que se encargaría de publicar, por ejemplo, los datos para crear un usuario, en una cola, por lo que el cliente no tendría que mantenerse en la página esperando a que se procese su petición. El sistema tendría una pila con las tareas e iría resolviendo una a una mientras que el usuario puede seguir realizando tareas en nuestra aplicación. Al mismo tiempo, podríamos tener un servicio escuchando esa cola y consumiendo sus datos para crear usuarios. En esta situación las peticiones se procesan de forma asíncrona.

Tras esta introducción, pasaremos a detallar cómo implementar un sistema de colas usando Laravel, AMQP y RabbitMQ. En este caso publicaremos en las colas con carácter informativo ya que no tenemos ningún servicio consumiendo de ellas. Laravel presenta facilidades para gestionar colas usando Redis o una tabla en base de datos por ejemplo, pero para el caso de RabbitMQ no tiene integración por lo que necesitaremos instalar las siguientes librerías.

```
composer require vladimir-yuldashev/laravel-queue-rabbitmq
```

```
composer require enqueue/amqp-bunny
```

Una vez hecho, añadimos a nuestro .env las variables necesarias para la conexión al sistema de colas.

```
# RabbitMQ
RABBITMQ_HOST=rabbitmq
RABBITMQ_PORT=5672
RABBITMQ_VHOST=/
RABBITMQ_LOGIN=guest
RABBITMQ_PASSWORD=guest
RABBITMQ_QUEUE=default

RABBITMQ_EXCHANGE_NAME=gtv
RABBITMQ_EXCHANGE_DECLARE=
RABBITMQ_EXCHANGE_TYPE=
RABBITMQ_EXCHANGE_PASSIVE=
RABBITMQ_EXCHANGE_DURABLE=
RABBITMQ_EXCHANGE_AUTODELETE=
RABBITMQ_EXCHANGE_ARGUMENTS=default

RABBITMQ_QUEUE_DECLARE=
RABBITMQ_QUEUE_DECLARE_BIND=
RABBITMQ_QUEUE_PASSIVE=
RABBITMQ_QUEUE_DURABLE=
RABBITMQ_QUEUE_EXCLUSIVE=
RABBITMQ_QUEUE_AUTODELETE=
RABBITMQ_QUEUE_ARGUMENTS=default

RABBITMQ_ERROR_SLEEP=5

RABBITMQ_SSL=
RABBITMQ_SSL_CAFILE=
RABBITMQ_SSL_LOCALCERT=
RABBITMQ_SSL_LOCALKEY=
RABBITMQ_SSL_VERIFY_PEER=
RABBITMQ_SSL_PASSPHRASE=
# RabbitMQ
```

Ahora nos dirigimos al archivo config/queue.php y en 'connections' añadimos una para rabbitmq usando las variables definidas.

```
'rabbitmq' => [
    'driver' => 'rabbitmq',
    'worker' => env( key: 'RABBITMQ_WORKER', default: 'default'),
    'dsn' => env( key: 'RABBITMQ_DSN', default: null),
    'factory_class' => \Enqueue\AmqpBunny\AmqpConnectionFactory::class,
    'host' => env( key: 'RABBITMQ_HOST', default: '127.0.0.1'),
    'port' => env( key: 'RABBITMQ_PORT', default: 5672),
    'vhost' => env( key: 'RABBITMQ_VHOST', default: '/'),
    'login' => env( key: 'RABBITMQ_LOGIN', default: 'guest'),
    'password' => env( key: 'RABBITMQ_PASSWORD', default: 'guest'),
    'queue' => env( key: 'RABBITMQ_QUEUE', default: 'default'),
    'options' => [
        'exchange' => [
            'name' => env( key: 'RABBITMQ_EXCHANGE_NAME'),
            'declare' => env( key: 'RABBITMQ_EXCHANGE_DECLARE', default: true),
            'type' => env( key: 'RABBITMQ_EXCHANGE_TYPE', default: \Interop\Amqp\AmqpTopic::TYPE_DIRECT),
            'passive' => env( key: 'RABBITMQ_EXCHANGE_PASSIVE', default: false),
            'durable' => env( key: 'RABBITMQ_EXCHANGE_DURABLE', default: true),
            'auto_delete' => env( key: 'RABBITMQ_EXCHANGE_AUTODELETE', default: false),
            'arguments' => env( key: 'RABBITMQ_EXCHANGE_ARGUMENTS'),
        ],
        'queue' => [
            'declare' => env( key: 'RABBITMQ_QUEUE_DECLARE', default: true),
            'bind' => env( key: 'RABBITMQ_QUEUE_DECLARE_BIND', default: true),
            'passive' => env( key: 'RABBITMQ_QUEUE_PASSIVE', default: false),
            'durable' => env( key: 'RABBITMQ_QUEUE_DURABLE', default: true),
            'exclusive' => env( key: 'RABBITMQ_QUEUE_EXCLUSIVE', default: false),
            'auto_delete' => env( key: 'RABBITMQ_QUEUE_AUTODELETE', default: false),
            'arguments' => env( key: 'RABBITMQ_QUEUE_ARGUMENTS'),
            'options' => [
                'exchange' => env( key: 'RABBITMQ_EXCHANGE_NAME'),
            ],
        ],
    ],
],
```



```

'sleep_on_error' => env( key: 'RABBITMQ_ERROR_SLEEP', default: 5),
'ssl_params' => [
  'ssl_on' => env( key: 'RABBITMQ_SSL', default: false),
  'cafile' => env( key: 'RABBITMQ_SSL_CAFILE', default: null),
  'local_cert' => env( key: 'RABBITMQ_SSL_LOCALCERT', default: null),
  'local_key' => env( key: 'RABBITMQ_SSL_LOCALKEY', default: null),
  'verify_peer' => env( key: 'RABBITMQ_SSL_VERIFY_PEER', default: true),
  'passphrase' => env( key: 'RABBITMQ_SSL_PASSPHRASE', default: null),
],
],

```

Para este ejemplo, vamos a usar una cola para procesar la creación de vídeos. Esta cola pertenece a un exchange (gestor de colas) llamado gtv, por lo que la petición se enviará al exchange y este la dirigirá a la cola llamada default.

Para enviar una tarea a una cola tenemos que generar un job o tarea con artisan del siguiente modo.

```

root@e37e9b01b572:/var/www/gtv# art make:job ProcessVideo

```

Esta clase será la encargada de enviar nuestro job (crear vídeo) a la cola, por lo que recibirá el modelo Video ya formado a través del constructor y pasará por el handler que lo enviará a la cola. Si fuera necesario en el handler podríamos añadir lógica o cualquier tarea que deba realizarse en ese momento.

```
class ProcessVideo implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $tries = 3;
    public $backoff = 3;

    protected $video;

    public function __construct(Video $video)
    {
        $this->video = $video;
    }

    public function handle()
    {
        //
    }
}
```

Para hacer uso de esta clase ProcessVideo, una vez creada la entidad Video hacemos el dispatch como se muestra en la imagen. También lo hacemos para su vídeo item. El comportamiento esperado sería que nuestra entidad se persistiría en base de datos y se comunicaría a la cola su información..

```

public function save()
{
    $this->validate();

    $fileRoute = $this->createForm['file']->store('public/videos');

    $video = Video::create([
        'route' => $fileRoute,
        'point_of_interest_id' => $this->createForm['pointOfInterest'],
        'order' => $this->order,
        'creator' => auth()->user()->id,
        'updater' => null,
        'thematic_area_id' => $this->createForm['thematicArea'],
        'description' => $this->createForm['description'],
    ]);

    ProcessVideo::dispatch($video);

    $videoItem = VideoItem::create([
        'video_id' => $video->id,
    ]);

    ProcessVideoItem::dispatch($videoItem);

    $this->reset( ...properties: 'videoTemporaryUrl');
    $this->reset( ...properties: 'createForm');
    $this->emit( event: 'videoCreated');
    $this->emitTo( name: 'admin.video.list-videos', event: 'render');
}

```

Para comprobar que todo funciona correctamente vamos a lanzar el siguiente comando de artisan, que nos permite ver lo que entraría por las colas sin que llegue a estas.

```

root@e37e9b01b572:/var/www/gtv# art queue:work

```

Si ahora añado un video desde el front, podremos ver que se procesa en el comando.

```
root@e37e9b01b572:/var/www/gtv# art queue:work
[2022-06-16 16:03:35][c7d4bcac-2dd2-48d6-887a-5d03916210a0] Processing: App\Jobs\ProcessVideo
[2022-06-16 16:03:35][c7d4bcac-2dd2-48d6-887a-5d03916210a0] Processed: App\Jobs\ProcessVideo
[2022-06-16 16:03:35][cbb888ca-a980-4536-951d-e45e396842be] Processing: App\Jobs\ProcessVideoItem
[2022-06-16 16:03:35][cbb888ca-a980-4536-951d-e45e396842be] Processed: App\Jobs\ProcessVideoItem
```

Ahora cancelamos la ejecución del comando, abrimos la interfaz de RabbitMQ ubicada en <http://localhost:15672>, nos dirigimos a nuestra cola y creamos el video.

Antes de crearlo:

RabbitMQ™ RabbitMQ 3.10.5 Erlang 24.3.4.1

Overview Connections Channels Exchanges **Queues** Admin

Queues

▼ All queues (1)

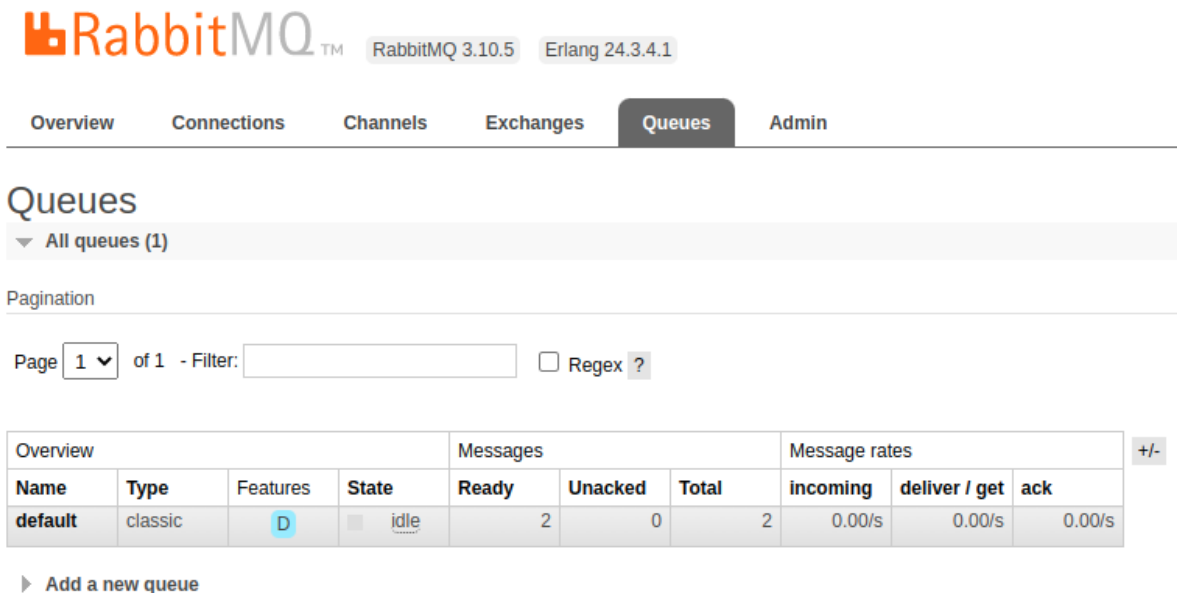
Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
default	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s

► Add a new queue

Después de crearlo:



RabbitMQ 3.10.5 Erlang 24.3.4.1

Overview Connections Channels Exchanges **Queues** Admin

Queues

▼ All queues (1)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	Incoming	deliver / get	ack	
default	classic	D	idle	2	0	2	0.00/s	0.00/s	0.00/s	

► Add a new queue

Como podemos ver, aparecen dos tareas procesadas. En el panel de mensajes le damos a Get Messages y le ponemos la cantidad 2 para verlos.



Message 1

The server reported 1 messages remaining.

Exchange (AMQP default)

Routing Key default

Redelivered 0

Properties

- correlation_id: 442294e4-9aee-4303-8e02-a5267c9c86a
- delivery_mode: 2
- headers: laravel: attempts: 0
- content_type: application/json

Payload 588 bytes Encoding: string

```
{"uuid": "7f76e3c5-c149-41b9-a61f-17c7033b2748", "displayName": "App\\Jobs\\ProcessVideo", "job": "Illuminate\\Queue\\CallQueuedHandler@call"}
```

Message 2

The server reported 0 messages remaining.

Exchange (AMQP default)

Routing Key default

Redelivered 0

Properties

- correlation_id: 71d5a3b0-75e7-4d94-b9ac-01543eba9692
- delivery_mode: 2
- headers: laravel: attempts: 0
- content_type: application/json

Payload 609 bytes Encoding: string

```
{"uuid": "eb2fde25-31f9-4ba2-aa38-4b094a3fff00", "displayName": "App\\Jobs\\ProcessVideoItem", "job": "Illuminate\\Queue\\CallQueuedHandler@call"}
```

Estas pertenecen a nuestro job ProcessVideo y ProcessVideoItem, por lo que se han procesado correctamente. Por defecto, Laravel incluye el identificador de la entidad enviada, en este caso Video y VideoItem. Si queremos que incluya más propiedades de estos debemos especificarlo a la hora de hacer el dispatch de la tarea.

```
,"App\\Models\\Video\\";s:2:"id\\";i:12;s:9:"relations\\";a:0:{}s:10:"connection\\";s:5:"mysql\\";}}},"id":"442294e4-9aee-4303-8e02-a5267cf9c86a"}
```

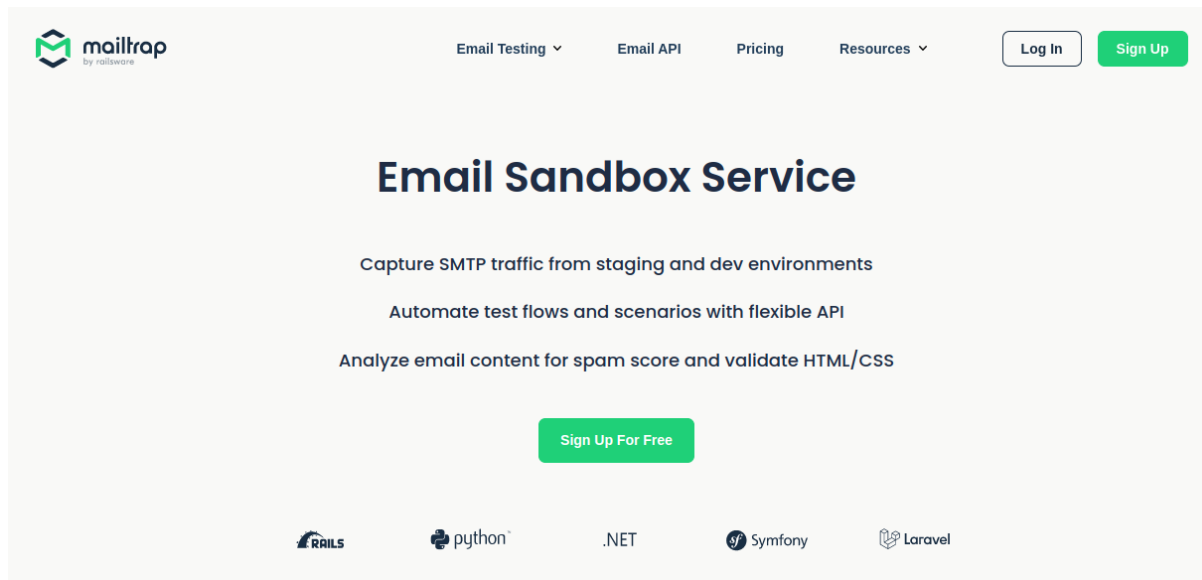
```
,"App\\Models\\VideoItem\\";s:2:"id\\";i:12;s:9:"relations\\";a:0:{}s:10:"connection\\";s:5:"mysql\\";}}},"id":"71d5a3b0-75e7-4d94-b9ac-01543eba9692"}
```

De este modo, podemos concluir que implementar un sistema de colas en nuestra aplicación supone grandes ventajas a la hora de gestionar el tráfico, ya que los usuarios no tienen que permanecer en la misma ruta esperando a que se procese su petición y el servidor les devuelva la respuesta.

3. Notificaciones por email con Mailtrap.io

En esta ocasión vamos a configurar nuestro proyecto para que se envíe un email al administrador cuando un usuario se da de alta. Para ello, vamos a configurar Laravel para poder enviar emails usando Mailtrap.

En primer lugar, creamos una cuenta en Mailtrap.



Una vez hecho nos dirigimos a Testing - Inboxes - My inbox - SMTP settings. En el desplegable de Integrations seleccionamos Laravel 7+ y veremos que nos genera las variables de entorno que necesitamos para usar sus servidores de correo.

mailtrap by railware

Sending **new**

Testing

Inboxes

API

Billing

User Management

Account Settings

Inboxes > My Inbox

Search...

How it works

- 1 Choose your technology
- 2 Copy configuration
- 3 Paste configuration to your project
- 4 Run your email sending code

My Inbox

SMTP Settings Email Address

SMTP / POP3 ? Reset Credentials

Use these settings to send messages directly

⚠ Don't disclose your username or password

Show Credentials

Integrations ?

Laravel 7+

Laravel provides a clean, simple API over the web. With the default Laravel setup you can configure your application to use Mailtrap as the SMTP provider.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=a7d42d516e7064
MAIL_PASSWORD=1209e960c66108
MAIL_ENCRYPTION=tls
```

Colocamos dichas variables en nuestro archivo .env, nos aseguramos que MAIL_DRIVER es smtp y añadimos MAIL_FROM_ADDRESS y MAIL_FROM_NAME.

```
# Mail
MAIL_DRIVER=smtp
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=a7d42d516e7064
MAIL_PASSWORD=1209e960c66108
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=no-reply@gtv.com
MAIL_FROM_NAME="GTV"
# Mail
```


En segundo lugar, abrimos el archivo config/mail.php y nos aseguramos de que las claves driver, host y from tienen los valores que necesitamos, declarados en el .env.

```
'driver' => env( key: 'MAIL_DRIVER', default: 'smtp'),  
  
'host' => env( key: 'MAIL_HOST', default: 'smtp.mailtrap.io'),  
  
'from' => [  
    'address' => env( key: 'MAIL_FROM_ADDRESS', default: 'no-reply@gtv.com'),  
    'name' => env( key: 'MAIL_FROM_NAME', default: 'GTV'),  
],
```

En tercer lugar, creamos una nueva clase de mail que usaremos para notificar cuando un usuario se registra. Para ello accedemos al bash del contenedor y ejecutamos el comando de artisan.

```
root@e37e9b01b572:/var/www/gtv# art make:mail UserCreated  
Mail created successfully.
```

En esta nueva clase, el método build retorna la vista que se verá en el cuerpo del email recibido por el administrador. Como podemos observar, es una vista de blade por lo que podemos inyectarle variables. En este caso queremos mostrar el nombre y el email del usuario creado en el cuerpo del mensaje, por lo que le pasamos el usuario en cuestión.

```

class UserCreated extends Mailable
{
    use Queueable, SerializesModels;

    protected $user;

    public function __construct(User $user)
    {
        $this->user = $user;
    }

    public function build()
    {
        $user = $this->user;

        return $this->view('email.user-created', compact('var_name' => $user));
    }
}

```

En cuarto lugar, una vez tenemos preparada la clase mail, nos queda llamarla al crear un nuevo usuario. Para ello usamos los métodos estáticos de la clase Mail. Con to() especificamos el destinatario que es el administrador. Si tuviéramos más de uno podríamos obtener todos los admins y recorrerlos enviando un mail por cada uno. En este caso solo tenemos uno así que hemos decidido hacerlo así.

```

public function save()
{
    $this->validate();

    if (null !== $this->createForm['avatar']) {
        $avatarRoute = $this->createForm['avatar']->store('public/user-avatars');
    }

    $user = User::create([
        'name' => $this->createForm['name'],
        'email' => $this->createForm['email'],
        'password' => \bcrypt($this->createForm['password']),
        'profile_photo_path' => $avatarRoute ?? null,
    ]);

    $role = Role::findById($this->createForm['role']);
    $user->assignRole($role);

    Mail::to( users: 'admin@mail.com' )->send(new UserCreated($user));

    $this->reset(...properties: 'createForm');
    $this->emit( event: 'userCreated');
    $this->emitTo( name: 'admin.user.list-users', event: 'render');
}

```

La vista que hemos usado como cuerpo del mensaje contiene clases de TailwindCSS, pero en realidad no se procesa como una vista normal y los estilos externos no se cargan. Para que los cargue hemos importado una librería que se encarga de incluirlos de modo 'inline' dentro de la propia vista, así los tiene disponibles.

```
composer require "fedehisas/laravel-mail-css-inliner:dev-master"
```

Finalmente, nos dirigimos a nuestro panel de usuarios, creamos uno y en unos instantes podemos ver en la bandeja de Mailtrap.io el email que se ha enviado al administrador. Podemos observar que el cuerpo del mensaje ha generado nuestra vista.

Usuario registrado

From: GTV <no-reply@gtv.com>
To: <admin@mail.com>

[Show Headers](#)

2022-06-16 12:28, 17 KB

HTML HTML Source Text Raw Spam Analysis HTML Check 12 Tech Info



GTV

Nuevo registro de usuario



Nombre: Juan

Email: juan.garcia@gmail.com

Rol: Alumno

[Ir a usuarios](#)

De este modo, usando Laravel, sus funcionalidades de emailing y Mailtrap.io podemos notificar a cualquier usuario del sistema sobre cualquier evento que ocurra.

4. Logging de eventos

En cualquier aplicación es esencial tener un control de lo que ocurre en cada momento, por lo que es necesario tener un log de las acciones que realiza cada usuario.

En esta ocasión hemos considerado interesante añadir logging a nuestro proyecto para tener registrada la edición y eliminación de usuarios ya que lo consideramos un proceso crítico. Al eliminar un usuario, toda la información relacionada con este también sería eliminada, por lo que en caso de que esto ocurra, nos gustaría tener constancia de quién llevó a cabo la acción y en qué momento.

Para llevar a cabo esta tarea, usaremos el sistema de logging que incorpora Laravel que por defecto los almacena como archivos .log en storage/logs. Además, incorporaremos una librería llamada LogViewer que nos ayudará a analizar los logs de una forma más gráfica y clara.

En primer lugar, en el archivo .env cambiaremos LOG_CHANNEL a daily para que cada archivo de log sea de un día y poder gestionarlos mejor.

```
LOG_CHANNEL=daily
```

En segundo lugar, instalamos LogViewer y publicamos sus archivos a nuestro proyecto con artisan.

```
root@e37e9b01b572:/var/www/gtv# composer require arcanedev/log-viewer
```

```

root@e37e9b01b572:/var/www/gtv# php artisan log-viewer:publish

  --
  //  ___  __ _\ /(\_)  _____  _____  --
  //  / _ \ / ' \ \ / / / _ \ \ \ / / _ \ ' _ \
  / ___/ ( ) / ( / \ / / / ___ \ \ / / ___/
  \___/\___/ \___/ \___/ \___/ \___/ \___/ \___/
          |___/

Version 9.0.0 - Created by ARCANEDEV

Copied File [/vendor/arcanedev/log-viewer/config/log-viewer.php] To [/config/log-viewer.php]
Copied Directory [/vendor/arcanedev/log-viewer/translations] To [/resources/lang/vendor/log-viewer]
Copied Directory [/vendor/arcanedev/log-viewer/views] To [/resources/views/vendor/log-viewer]
Publishing complete.

```

En tercer lugar, usaremos el facade Log de Laravel para crear nuestros logs, a los que nos interesa añadirle un mensaje con la acción llevada a cabo. Al tratarse de la edición y borrado de usuarios, incluiremos el identificador del usuario que lleva a cabo la acción y el usuario afectado.

```

public function update(User $user)
{
    $this->rules['editForm.email'] = 'required|confirmed|string|max:45|unique:users,email,' . $this->userId;

    $this->validate();

    $isUpdated = $user->update([
        'name' => $this->editForm['name'],
        'email' => $this->editForm['email'],
        'password' => \bcrypt($this->editForm['password']),
    ]);

    $role = Role::findById($this->editForm['role']);
    $user->syncRoles($role);

    if ($isUpdated) {
        Log::info( message: 'User with ID ' . auth()->user()->id . ' edited the following user ' . $user);
    }

    $this->editForm['open'] = false;
    $this->reset(['editForm']);
    $this->emitTo( name: 'admin.user.list-users', event: 'render');
    $this->emit( event: 'userUpdated');
}

```

```

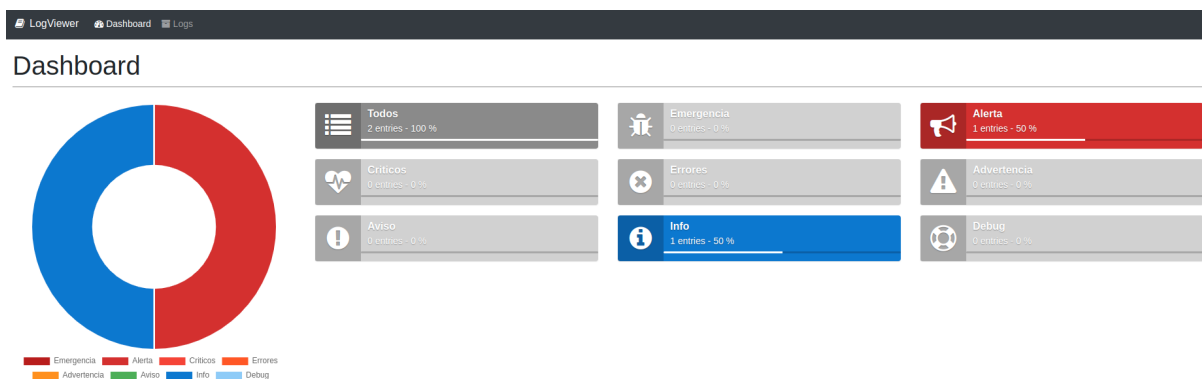
public function delete(User $user)
{
    if(Storage::exists($user->profile_photo_path)) {
        Storage::delete($user->profile_photo_path);
    }

    $isDeleted = $user->delete();

    if ($isDeleted) {
        Log::alert( message: 'User with ID ' . auth()->user()->id . ' removed an user ' . $user);
    }
}

```

Por último, desde nuestro panel editamos y borramos un usuario para ver si los logs se están publicando en LogViewer, a cuyo panel accedemos desde <http://gtv.local/log-viewer>.



Como podemos observar, tenemos un log de tipo info y otro tipo warning. Si nos dirigimos a la pestaña Logs podemos acceder al archivo donde se han registrado estas acciones y ver información más detallada.

The screenshot shows the LogViewer Logs tab. It displays a table of log entries with columns for Fecha, Log Level, and Actions.

Fecha	Log Level	Actions
2022-06-16	Alerta	Search, Download, Delete

Log [2022-06-16]

Levels

Todos

Emergencia

Alerta

Criticos

Errores

Advertencia

Aviso

Info

Debug

Log info

Download

Delete

File path : /var/www/giv/storage/logs/laravel-2022-06-16.log

Log entries : 3 Size : 1.24 KB Created at : 2022-06-16 22:15:40 Updated at : 2022-06-16 22:15:38

Type here to search

ENV	Level	Time	Header	Actions
local	Alerta	22:15:38	User with ID 1 removed an user	Context
<pre>{ "id": 7, "name": "Pope", "email": "pepe.garcia@gmail.com", "two_factor_confirmed_at": null, "email_verified_at": null, "current_team_id": null, "profile_photo_path": "public/user-avatars\\\\NF\\KXZB57ICLQV6V5CHXZYL9XFN14Y2Q5S08.jpg", "deleted_at": null, "created_at": "2022-06-16T22:15:15.000000Z", "updated_at": "2022-06-16T22:15:31.000000Z", "profile_photo_url": "http://localhost/storage/public/user-avatars\\\\NF\\KXZB57ICLQV6V5CHXZYL9XFN14Y2Q5S08.jpg" }</pre>				
local	Info	22:15:31	User with ID 1 edited the following user	Context

A continuación podemos ver el log con la hora a la que ocurrió, de qué tipo es, en qué entorno ocurrió, el header que contiene el mensaje que especificamos en las clases ListUsers y EditUser y el modelo del usuario afectado serializado, es decir, en formato JSON.

ENV

Level

Time

Header

local

Alerta

22:15:38

User with ID 1 removed an user

```
{
  "id": 7,
  "name": "Pepe",
  "email": "pepe.garcia@gmail.com",
  "two_factor_confirmed_at": null,
  "email_verified_at": null,
  "current_team_id": null,
  "profile_photo_path": "public\\user-avatars\\sNFxKbXzBS71CLQYGv5tChXZYL90xFNi4Y2qSso8.jpg",
  "deleted_at": null,
  "created_at": "2022-06-16T22:15:15.000000Z",
  "updated_at": "2022-06-16T22:15:31.000000Z",
  "profile_photo_url": "http://localhost/storage/public/user-avatars/sNFxKbXzBS71CLQYGv5tChXZYL90xFNi4Y2qSso8.jpg"
}
```

local

Info

22:15:31

User with ID 1 edited the following user

```
{
  "id": 7,
  "name": "Pepe",
  "email": "pepe.garcia@gmail.com",
  "two_factor_confirmed_at": null,
  "email_verified_at": null,
  "current_team_id": null,
  "profile_photo_path": "public\\user-avatars\\sNFxKbXzBS71CLQYGv5tChXZYL90xFNi4Y2qSso8.jpg",
  "deleted_at": null,
  "created_at": "2022-06-16T22:15:15.000000Z",
  "updated_at": "2022-06-16T22:15:31.000000Z",
  "profile_photo_url": "http://localhost/storage/public/user-avatars/sNFxKbXzBS71CLQYGv5tChXZYL90xFNi4Y2qSso8.jpg",
  "roles": [
    {
      "id": 3,
      "name": "Alumno",
      "display_name": null,
      "guard_name": "web",
      "created_at": "2022-06-16T21:49:35.000000Z",
      "updated_at": "2022-06-16T21:49:35.000000Z",
      "pivot": {
        "model_id": 7,
        "role_id": 3,
        "model_type": "App\\Models\\User"
      }
    }
  ]
}
```