**Stripe App**

**Formación en centros de trabajo (FCT)**

Contenido

[Introducción 2](#_Toc197710251)

[Despliegue y repositorios del proyecto 2](#_Toc197710252)

[Instrucciones de uso 2](#_Toc197710253)

[Detalles Importantes 3](#_Toc197710254)

# Introducción

Proyecto que permite la integración de la plataforma de pagos [Stripe](https://stripe.com/es) en una aplicación web hecha con Laravel como framework del backend y React en el frontend.

La aplicación permite al usuario realizar consultas del saldo, recargas, pedir los reembolsos de los pagos, gestionar el historial de las transacciones y visualizar gráficos con la información almacenada en la base de datos.

# Despliegue y repositorios del proyecto

El proyecto se encuentra desplegado en las url:

Fontend: <https://stripe.andrescoder.dev/> (desactivado)

Backend: <https://stripeapi.andrescoder.dev/> (desactivado)

El código fuente del proyecto está disponible en Github a través del repositorio de código:

<https://github.com/andresito87/StripeApp>

En el directorio donde se encuentra esta documentación he agregado el archivo de configuración de Postman para poder probar el api, es el archivo llamado **STRIPE.postman\_collection.json**

En el directorio BD se encuentra el script de MySQL para crear las tablas y poblarla con algunos datos. Laravel mediante migraciones es capaz de crear toda la estructura de la base de datos, pero no introduce datos de prueba.

# Instrucciones de uso

El procedimiento para arrancar el proyecto en local será:

1.- Descargar el código fuente del proyecto.

2.- Instalar las depencias del backend ejecutando desde esta carpeta el comando **composer install**(para ejecutarlo en Windows hubo que agregar los dll de la librería Imagick <https://mlocati.github.io/articles/php-windows-imagick.html> además de modificar los parámetros necesarios en el php.ini) y las dependencias del frontal ejecutando desde esta carpeta el comando **npm install**.

3.- Configuraremos el archivo .env del backend para agregar las credenciales de la base de datos y ejecutaremos las migraciones con el comando **php artisan migrate**.

4.- Para lanzar el api ejecutaremos el comando **php artisan serve**, seguidamente necesitamos ejecutar la cola en la que se encolan los eventos de Stripe con el comando **php artisan queue:work** y, por último, indicar a stripe el endpoint que debe usar para los eventos que hemos configurado en su dasboard con el comando **stripe listen --forward-to** [**http://localhost:8000/api/stripe/webhook**](http://localhost:8000/api/stripe/webhook)al ejecutar este comando la terminal nos devolverá una llave que será la que usará Stripe para firmar todas sus peticiones a nuestra api para validar que son peticiones confiables, debemos copiar esta clave en el parámetro STRIPE\_WEBHOOK\_SECRET de nuestro archivo .env

5.- Para lanzar nuestro frontal en React usaremos el comando **npm run dev** y la aplicación estará disponible en la dirección <http://localhost:5173/>

# Detalles Importantes

* [**Radar**](https://stripe.com/es/radar/fraud-teams) es una opción de Stripe del dashboard de Stripe para personalizar el manejo de transacciones fraudulentas, en esta demostración se usa la configuración por defecto que recomienda Stripe.
* Para el uso de **MFA** en la aplicación se usa la librería <https://github.com/antonioribeiro/google2fa-laravel>
* Para el uso de autenticación con **3DSecure** se usa una configuración específica del payment intent(<https://docs.stripe.com/payments/payment-intents/upgrade-to-handle-actions>)

                $paymentIntent = PaymentIntent::create([

…

                    'confirmation\_method' => 'manual',

                    'confirm' => true,

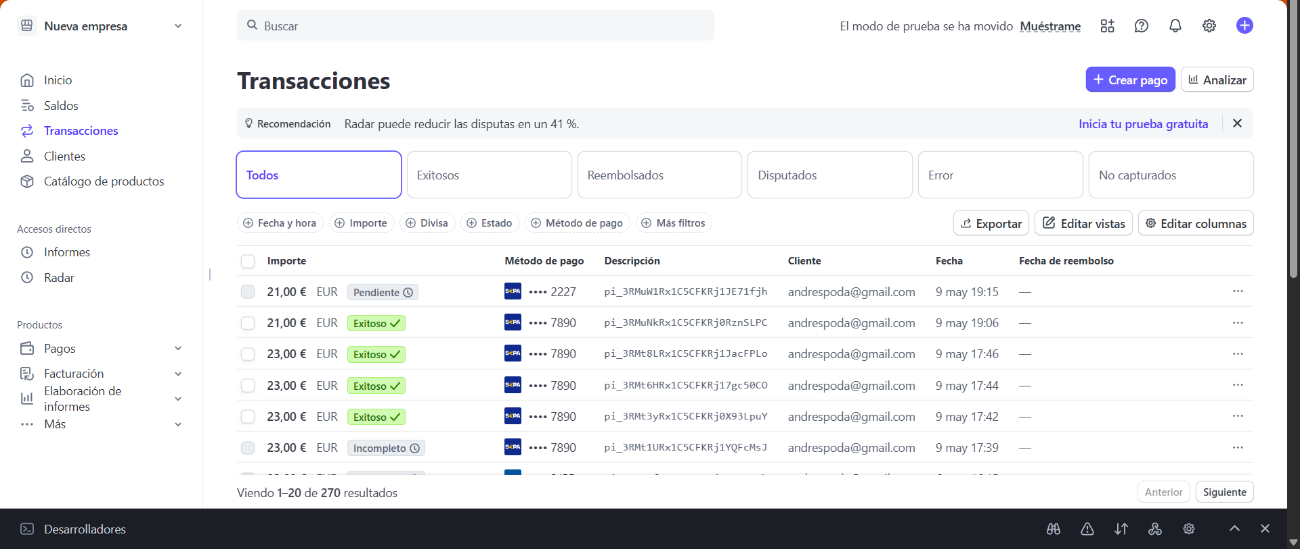
                    'use\_stripe\_sdk' => true,

                    'payment\_method\_types' => ['card'],

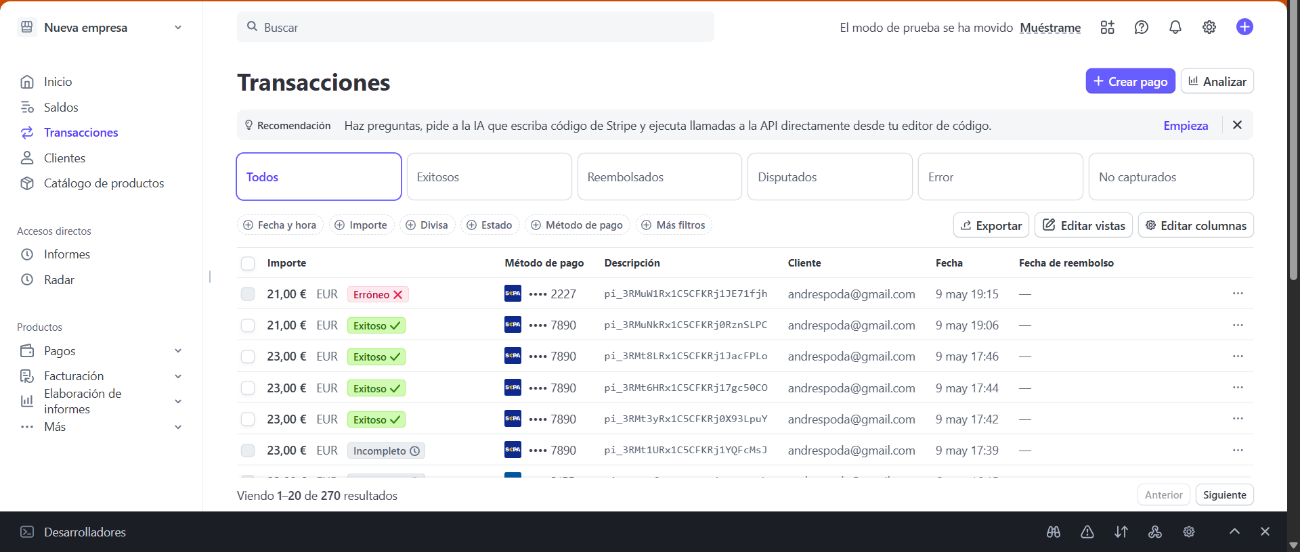
                ]);

* Agregar biblioteca [**NIVO**](https://nivo.rocks/) para los gráficos con las mismas versiones para todos los tipos de gráficos sino puede dar errores. En mi caso uso "^0.88.0" para todos.
* En la parte de reembolso global se considera que las solicitudes de reembolso usarán números de cuenta bancaria tipo SEPA y estas son peticiones que Stripe resuelve asíncronamente, el api de Stripe devuelve como status=processing y unos segunos o minutos después Stripe actualiza su dashboard con el estado concreto. Es por esto por lo que nuestro frontal considera una transferencia exitosa al status processing.

Primero muestra el estado pendiente:



Y más tarde resuelve dependiendo si es exitoso o erróneo:



* Corregido el error que provocaba que en el historial de transacciones no se mostrara el motivo de la disputa correctamente. El webhook actualiza el campo correspondiente en la BD cuando se crea la disputa.