Integrating Stripe Payments with Webhooks in Laravel

Here's a comprehensive guide to integrate Stripe payments for subscriptions and one-time charges with webhook support in your Laravel application.

1. Install Required Packages

```
composer require stripe/stripe-php laravel/cashier
```

2. Set Up Configuration

Add to your `.env`:

```
STRIPE_KEY=your_stripe_publishable_key
STRIPE_SECRET=your_stripe_secret_key
STRIPE_WEBHOOK_SECRET=your_stripe_webhook_secret
CASHIER_CURRENCY=usd
CASHIER_CURRENCY_LOCALE=en
```

Publish Cashier migrations:

```
php artisan vendor:publish --tag="cashier-migrations"
php artisan migrate
```

PROF

3. Prepare Your User Model

```
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}
```

4. Set Up Routes

```
// routes/web.php
Route::post('/stripe/webhook',
```

```
'\Laravel\Cashier\Http\Controllers\WebhookController@handleWebhook');

// One-time charge
Route::post('/charge', [PaymentController::class, 'charge'])->
>name('charge');

// Subscription
Route::post('/subscribe', [SubscriptionController::class, 'subscribe'])->
>name('subscribe');
Route::post('/cancel-subscription', [SubscriptionController::class,
'cancel'])->name('cancel-subscription');
```

5. Create Controllers

For one-time charges:

```
// app/Http/Controllers/PaymentController.php

use Illuminate\Http\Request;
use Stripe\Stripe;
use Stripe\PaymentIntent;

class PaymentController extends Controller
{
    public function charge(Request $request)
    {
        Stripe::setApiKey(config('services.stripe.secret'));

        try {
            $paymentIntent = PaymentIntent::create([
                'amount' => $request->amount * 100, // in cents
                'currency' => config('cashier.currency'),
                'payment_method' => $request->payment_method,
                'confirm' => true,
                'description' => $request->description,
                'metadata' => [
                    'user_id' => auth()->id(),
                    'product_id' => $request->product_id
                ],
            ]);

            // Save the payment to your database
            auth()->user()->payments()->create([
                'stripe_id' => $paymentIntent->id,
                'amount' => $paymentIntent->amount / 100,
                'status' => $paymentIntent->status,
            ]);

            return response()->json(['success' => true]);
```

```
        } catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()], 500);
        }
    }
}
```

For subscriptions:

```
// app/Http/Controllers/SubscriptionController.php

use Illuminate\Http\Request;

class SubscriptionController extends Controller
{
    public function subscribe(Request $request)
    {
        $user = $request->user();
        $paymentMethod = $request->payment_method;
        $planId = $request->plan_id;

        try {
            $user->createOrGetStripeCustomer();
            $user->updateDefaultPaymentMethod($paymentMethod);

            $subscription = $user->newSubscription('default', $planId)
                ->create($paymentMethod, [
                    'email' => $user->email,
                ]);

            return response()->json(['success' => true]);

        } catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()], 500);
        }
    }

    public function cancel(Request $request)
    {
        $request->user()->subscription('default')->cancel();

        return response()->json(['success' => true]);
    }
}
```

PROF

6. Set Up Webhooks

Create a webhook handler:

```php
// app/Http/Controllers/WebhookController.php

namespace App\Http\Controllers;

use Laravel\Cashier\Http\Controllers\WebhookController as
CashierController;

class WebhookController extends CashierController
{
    public function handleInvoicePaymentSucceeded($payload)
    {
        // Handle successful invoice payment (for subscriptions)
        $user = $this->getUserByStripeId($payload['data']['object']
['customer']);

        // Your logic here (e.g., send email, update database)

        return response('Webhook Handled', 200);
    }

    public function handleChargeSucceeded($payload)
    {
        // Handle successful one-time charge
        $charge = $payload['data']['object'];

        // Your logic here

        return response('Webhook Handled', 200);
    }

    public function handleCustomerSubscriptionDeleted($payload)
    {
        // Handle subscription cancellation
        $user = $this->getUserByStripeId($payload['data']['object']
['customer']);

        // Your logic here

        return response('Webhook Handled', 200);
    }

    // Add more webhook handlers as needed
}
```

PROF

7. Frontend Implementation (JavaScript)

```javascript
// resources/js/stripe.js

const stripe = Stripe(process.env.MIX_STRIPE_KEY);
```

```
// For one-time payment
async function handlePayment(amount, description) {
    const { error, paymentIntent } = await
stripe.confirmCardPayment(clientSecret, {
        payment_method: {
            card: elements.getElement('card'),
            billing_details: {
                name: document.getElementById('card-holder-name').value,
            },
        }
    });

    if (error) {
        console.error(error);
    } else {
        await fetch('/charge', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-
token"]').content
            },
            body: JSON.stringify({
                payment_method: paymentIntent.payment_method,
                amount: amount,
                description: description
            })
        });
    }
}

// For subscription
async function handleSubscribe(planId) {
    const { error, paymentMethod } = await stripe.createPaymentMethod(
        'card', elements.getElement('card'), {
            billing_details: {
                name: document.getElementById('card-holder-name').value,
            },
        }
    );

    if (error) {
        console.error(error);
    } else {
        await fetch('/subscribe', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-
token"]').content
            },
            body: JSON.stringify({
```

```
                payment_method: paymentMethod.id,
                plan_id: planId
            })
        });
    }
}
```

8. Testing Webhooks Locally

Use the Stripe CLI to test webhooks locally:

```
stripe listen --forward-to localhost:8000/stripe/webhook
```

9. Important Webhook Events to Handle

Here are key Stripe events you should handle:

- `invoice.payment_succeeded` - Subscription payment succeeded
- `invoice.payment_failed` - Subscription payment failed
- `charge.succeeded` - One-time payment succeeded
- `charge.failed` - One-time payment failed
- `customer.subscription.deleted` - Subscription cancelled/ended
- `customer.subscription.updated` - Subscription changed

10. Security Considerations

1. Always verify webhook signatures:

```php
$payload = @file_get_contents('php://input');
$sig_header = $_SERVER['HTTP_STRIPE_SIGNATURE'];
$event = null;

try {
    $event = \Stripe\Webhook::constructEvent(
        $payload, $sig_header, config('services.stripe.webhook_secret')
    );
} catch(\UnexpectedValueException $e) {
    // Invalid payload
    http_response_code(400);
    exit();
} catch(\Stripe\Exception\SignatureVerificationException $e) {
    // Invalid signature
    http_response_code(400);
    exit();
}
```

2. Use idempotency keys for critical operations to prevent duplicate processing.
3. Implement proper error handling and logging for all webhook events.

This implementation provides a solid foundation for both subscription and one-time payments with proper webhook handling in your Laravel application.