1v0 Ship v1.0 Faster

How to Integrate Stripe Payments in Laravel

How to Integrate Stripe Payments in Laravel

Stripe is one of the most popular payment processors for web apps. Laravel makes it simple to integrate using the official `stripe/stripe-php` SDK. In this guide, you'll set up Stripe, create checkout sessions, handle webhooks, and build a small UI to accept payments securely.

1 – Install Stripe SDK

Start by requiring the Stripe PHP SDK with Composer.

```
composer require stripe/stripe-phpCode language: Bash (bash)
```

This installs the official Stripe library, giving you access to Checkout Sessions, PaymentIntents, and webhooks.

2 – Configure API Keys

Add your Stripe keys to `.env` so you can access them securely in your code.

STRIPE_KEY=pk_test_1234567890 STRIPE_SECRET=sk_test_1234567890

`STRIPE_KEY` is the publishable key for the frontend, and `STRIPE_SECRET` is the secret key

used on the backend. Never expose the secret key in your views.

3 – Checkout Controller

Create a controller that initializes a Stripe Checkout Session when a user wants to make a purchase.

```
// app/Http/Controllers/CheckoutController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Stripe\Stripe;
use Stripe\Checkout\Session as CheckoutSession;

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

        $session = CheckoutSession::create([
            'payment_method_types' => ['card'],
            'line_items' => [[
                'price_data' => [
                    'currency' => 'usd',
                    'unit_amount' => 1999, // $19.99
                    'product_data' => [
                        'name' => 'Pro Subscription',
                    ],
                ],
                'quantity' => 1,
            ]],
            'mode' => 'payment',
```

```
            'success_url' => url('/payment-
success?session_id={CHECKOUT_SESSION_ID}'),
            'cancel_url' => url('/payment-cancel'),
        ]);

        return response()->json(['id' => $session->id]);
    }
}Code language: PHP (php)
```

This method creates a checkout session for one product (`Pro Subscription`) at $19.99. On success, Stripe redirects to `/payment-success`; if canceled, to `/payment-cancel`.

4 – Webhook Handling

Stripe uses webhooks to notify your app about successful payments. Handle them to mark orders as paid.

```
// app/Http/Controllers/StripeWebhookController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Stripe\Webhook;

class StripeWebhookController extends Controller
{
    public function handle(Request $request)
    {
        $endpointSecret = config('services.stripe.webhook_secret');
        $sigHeader = $request->header('Stripe-Signature');
        $payload = $request->getContent();

        try {
            $event = Webhook::constructEvent($payload, $sigHeader,
```

```
$endpointSecret);
        } catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()],
400);
        }

        if ($event->type === 'checkout.session.completed') {
            $session = $event->data->object;
            // Lookup order, mark as paid
        }

        return response()->json(['status' => 'success']);
    }
}Code language: PHP (php)
```

The webhook verifies the signature with `webhook_secret`. On `checkout.session.completed`, update your DB to reflect a successful payment. Always validate events this way to prevent spoofing.

5 – API Routes

Add routes for creating checkout sessions and handling Stripe webhooks.

```
// routes/api.php
use App\Http\Controllers\CheckoutController;
use App\Http\Controllers\StripeWebhookController;

Route::post('/checkout', [CheckoutController::class, 'create']);
Route::post('/stripe/webhook', [StripeWebhookController::class,
'handle']);Code language: PHP (php)
```

Clients call `/api/checkout` to start a payment, and Stripe calls `/api/stripe/webhook` when payments succeed or fail.

6 – UI: Payment Button with Stripe.js

Use Stripe.js on the frontend to redirect customers to the hosted checkout page.

```
<!-- resources/views/payment.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>Buy Pro Subscription</h1>
  <button id="checkout-button" class="btn btn-theme">Pay
$19.99</button>
</div>

<script src="https://js.stripe.com/v3/"></script>
<script>
const stripe = Stripe("{{ config('services.stripe.key') }}");

document.getElementById("checkout-button").addEventListener("click",
function() {
  fetch("/api/checkout", { method: "POST" })
    .then(res => res.json())
    .then(data => {
      stripe.redirectToCheckout({ sessionId: data.id });
    });
});
</script>
@endsectionCode language: HTML, XML (xml)
```

This Blade file adds a payment button. When clicked, it requests a checkout session from Laravel, then redirects the user to Stripe Checkout securely.

Wrapping Up

You just integrated Stripe with Laravel by installing the SDK, configuring keys, creating checkout sessions, handling webhooks, and building a UI with Stripe.js. This flow ensures secure payments and reliable updates to your database when charges succeed.

What's Next

- PayPal Integration in Laravel (Step by Step)
- How to Build a Secure File Upload API in Laravel
- Automating Laravel Deployments with Deployer

How to Integrate Stripe Payments in Laravel

Stripe is one of the most popular payment processors for web apps. Laravel makes it simple to integrate using the official `stripe/stripe-php` SDK. In this guide, you'll set up Stripe, create checkout sessions, handle webhooks, and build a small UI to accept payments securely.

1 – Install Stripe SDK

Start by requiring the Stripe PHP SDK with Composer.

```
composer require stripe/stripe-phpCode language: Bash (bash)
```

This installs the official Stripe library, giving you access to Checkout Sessions, PaymentIntents, and webhooks.

2 – Configure API Keys

Add your Stripe keys to `.env` so you can access them securely in your code.

STRIPE_KEY=pk_test_1234567890 STRIPE_SECRET=sk_test_1234567890

`STRIPE_KEY` is the publishable key for the frontend, and `STRIPE_SECRET` is the secret key used on the backend. Never expose the secret key in your views.

3 – Checkout Controller

Create a controller that initializes a Stripe Checkout Session when a user wants to make a purchase.

```
// app/Http/Controllers/CheckoutController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
use Stripe\Stripe;
use Stripe\Checkout\Session as CheckoutSession;

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

        $session = CheckoutSession::create([
            'payment_method_types' => ['card'],
            'line_items' => [[
                'price_data' => [
                    'currency' => 'usd',
                    'unit_amount' => 1999, // $19.99
                    'product_data' => [
                        'name' => 'Pro Subscription',
                    ],
                ],
                'quantity' => 1,
            ]],
            'mode' => 'payment',
            'success_url' => url('/payment-
success?session_id={CHECKOUT_SESSION_ID}'),
            'cancel_url' => url('/payment-cancel'),
        ]);

        return response()->json(['id' => $session->id]);
    }
}Code language: PHP (php)
```

This method creates a checkout session for one product (`Pro Subscription`) at $19.99. On success, Stripe redirects to `/payment-success`; if canceled, to `/payment-cancel`.

4 – Webhook Handling

Stripe uses webhooks to notify your app about successful payments. Handle them to mark orders as paid.

```
// app/Http/Controllers/StripeWebhookController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Stripe\Webhook;

class StripeWebhookController extends Controller
{
    public function handle(Request $request)
    {
        $endpointSecret = config('services.stripe.webhook_secret');
        $sigHeader = $request->header('Stripe-Signature');
        $payload = $request->getContent();

        try {
            $event = Webhook::constructEvent($payload, $sigHeader,
$endpointSecret);
        } catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()],
400);
        }

        if ($event->type === 'checkout.session.completed') {
            $session = $event->data->object;
            // Lookup order, mark as paid
        }

        return response()->json(['status' => 'success']);
    }
}Code language: PHP (php)
```

The webhook verifies the signature with `webhook_secret`. On `checkout.session.completed`, update your DB to reflect a successful payment. Always

validate events this way to prevent spoofing.

5 – API Routes

Add routes for creating checkout sessions and handling Stripe webhooks.

```
// routes/api.php
use App\Http\Controllers\CheckoutController;
use App\Http\Controllers\StripeWebhookController;

Route::post('/checkout', [CheckoutController::class, 'create']);
Route::post('/stripe/webhook', [StripeWebhookController::class,
'handle']);Code language: PHP (php)
```

Clients call `/api/checkout` to start a payment, and Stripe calls `/api/stripe/webhook` when payments succeed or fail.

6 – UI: Payment Button with Stripe.js

Use Stripe.js on the frontend to redirect customers to the hosted checkout page.

```
<!-- resources/views/payment.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>Buy Pro Subscription</h1>
```

```
  <button id="checkout-button" class="btn btn-theme">Pay
$19.99</button>
</div>

<script src="https://js.stripe.com/v3/"></script>
<script>
const stripe = Stripe("{{ config('services.stripe.key') }}");

document.getElementById("checkout-button").addEventListener("click",
function() {
  fetch("/api/checkout", { method: "POST" })
    .then(res => res.json())
    .then(data => {
      stripe.redirectToCheckout({ sessionId: data.id });
    });
});
</script>
@endsectionCode language: HTML, XML (xml)
```

This Blade file adds a payment button. When clicked, it requests a checkout session from Laravel, then redirects the user to Stripe Checkout securely.

Wrapping Up

You just integrated Stripe with Laravel by installing the SDK, configuring keys, creating checkout sessions, handling webhooks, and building a UI with Stripe.js. This flow ensures secure payments and reliable updates to your database when charges succeed.

What’s Next

- PayPal Integration in Laravel (Step by Step)
- How to Build a Secure File Upload API in Laravel
- Automating Laravel Deployments with Deployer