1v0 Ship v1.0 Faster

PayPal Integration in Laravel (Step by Step)

PayPal Integration in Laravel (Step by Step)

PayPal remains one of the most widely used payment gateways globally. Integrating it into your Laravel app allows you to accept payments from millions of users. In this guide, we'll set up PayPal SDK, configure API credentials, create routes for checkout, handle webhooks for payment confirmation, and build a simple Blade UI to process PayPal transactions.

1 – Install PayPal SDK

Use Composer to install the official PayPal REST SDK.

```
composer require paypal/rest-api-sdk-phpCode language: Bash (bash)
```

This package provides the classes you need to create payment requests, handle approvals, and confirm completed payments via PayPal APIs.

2 – Configure PayPal Credentials

Add your sandbox or live API credentials to `.env`:

PAYPAL_CLIENT_ID=your-sandbox-client-id PAYPAL_SECRET=your-sandbox-secret PAYPAL_MODE=sandbox # or live

Keep two sets of credentials: `sandbox` for testing and `live` for production. Never commit real secrets to your repository.

3 – Create PayPal Service

Encapsulate PayPal logic in a service class for cleaner controllers.

```
// app/Services/PayPalService.php
namespace App\Services;

use PayPal\Rest\ApiContext;
use PayPal\Auth\OAuthTokenCredential;
use PayPal\Api\Amount;
use PayPal\Api\Payer;
use PayPal\Api\Payment;
use PayPal\Api\PaymentExecution;
use PayPal\Api\RedirectUrls;
use PayPal\Api\Transaction;

class PayPalService
{
    protected $apiContext;

    public function __construct()
    {
        $this->apiContext = new ApiContext(
            new OAuthTokenCredential(
                config('services.paypal.client_id'),
                config('services.paypal.secret')
            )
        );
        $this->apiContext->setConfig([
            'mode' => config('services.paypal.mode', 'sandbox'),
```

```
        ]);
    }

    public function createPayment($amount, $currency = 'USD')
    {
        $payer = new Payer();
        $payer->setPaymentMethod('paypal');

        $amountObj = new Amount();
        $amountObj->setTotal($amount)->setCurrency($currency);

        $transaction = new Transaction();
        $transaction->setAmount($amountObj)->setDescription("Order
Payment");

        $redirectUrls = new RedirectUrls();
        $redirectUrls->setReturnUrl(url('/paypal/success'))
                     ->setCancelUrl(url('/paypal/cancel'));

        $payment = new Payment();
        $payment->setIntent('sale')
                ->setPayer($payer)
                ->setTransactions([$transaction])
                ->setRedirectUrls($redirectUrls);

        return $payment->create($this->apiContext);
    }

    public function executePayment($paymentId, $payerId)
    {
        $payment = Payment::get($paymentId, $this->apiContext);
        $execution = new PaymentExecution();
        $execution->setPayerId($payerId);

        return $payment->execute($execution, $this->apiContext);
    }
}Code language: PHP (php)
```

This service handles both creating a payment (with redirect URLs) and executing it after PayPal approves the transaction. Wrapping logic in a service keeps controllers lean.

4 – Checkout Controller

Create a controller to start the payment and handle PayPal redirects.

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

use Illuminate\Http\Request;
use App\Services\PayPalService;

class PayPalController extends Controller
{
    protected $paypal;

    public function __construct(PayPalService $paypal)
    {
        $this->paypal = $paypal;
    }

    public function checkout()
    {
        $payment = $this->paypal->createPayment(19.99);
        return redirect($payment->getApprovalLink());
    }

    public function success(Request $request)
    {
        $paymentId = $request->query('paymentId');
        $payerId = $request->query('PayerID');
```

```
        $result = $this->paypal->executePayment($paymentId, $payerId);

        // Update DB: mark order as paid

        return view('paypal.success', ['result' => $result]);
    }

    public function cancel()
    {
        return view('paypal.cancel');
    }
}Code language: PHP (php)
```

The `checkout` method redirects users to PayPal. After approval, PayPal returns to `success`, where we execute the payment and mark the order as paid. The `cancel` method simply handles user cancellations.

5 – Routes

Add routes to connect the flow.

```
// routes/web.php
use App\Http\Controllers\PayPalController;

Route::get('/paypal/checkout', [PayPalController::class, 'checkout']);
Route::get('/paypal/success', [PayPalController::class, 'success']);
Route::get('/paypal/cancel', [PayPalController::class, 'cancel']);Code language: PHP (php)
```

These routes cover the full PayPal checkout cycle: start, success, and cancel. In production, secure them with auth checks to link payments to logged-in users.

6 – UI: PayPal Button

You can use PayPal's JS SDK to render a checkout button instead of a simple link.

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

@section('content')
<div class="container">
  <h1>Pay with PayPal</h1>
  <div id="paypal-button-container"></div>
</div>

<script src="https://www.paypal.com/sdk/js?client-id={{
config('services.paypal.client_id') }}&currency=USD"></script>
<script>
paypal.Buttons({
  createOrder: function() {
    return fetch('/paypal/checkout')
      .then(res => res.text());
  },
  onApprove: function(data, actions) {
    window.location.href =
'/paypal/success?paymentId='+data.paymentID+'&PayerID='+data.payerID;
  }
}).render('#paypal-button-container');
</script>
@endsectionCode language: HTML, XML (xml)
```

This example renders PayPal's official button on your page. Clicking it starts the checkout, and PayPal handles authentication and redirects. Always test in sandbox before going live.

Wrapping Up

You've successfully integrated PayPal into Laravel. You installed the SDK, configured credentials, built a service for clean code, handled checkout, and tested payments with a UI button. With this flow, you can accept PayPal transactions securely and expand your app's payment options.

What's Next

- How to Build a Multi-Auth API with Laravel & Sanctum
- Using Laravel Passport for Advanced API Authentication
- Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)