

Flutterwave for Laravel

Take card, bank transfer, USSD and mobile money payments in your Laravel app, then send payouts and react to payment events. This guide takes you from composer require to a verified test payment and a safe launch, in about 30 minutes.

VERSION **2.3.0** LARAVEL 9 TO 13 PHP **8.1+** SETUP TIME ~30 MIN
UPDATED OCT 2026

View on Packagist
ABRAHAM-FLUTTERWAVE/LARAVEL-PAYMENT
<https://packagist.org/packages/abraham-flutterwave/laravel-payment>

LICENCE MIT	WHERE TO INSTALL Packagist, with Composer	LATEST RELEASE 2.3.0 · 15 Oct 2026
LICENCE MIT	REQUIRES PHP 8.1+ · Laravel 9 to 13	SUPPORT Slack, email & GitHub

ON THIS PAGE

- Before you begin
- 1· Install the package
- 2· Connect your account
- 3· Take a test payment
- 4· Go live
- Troubleshooting
- Get help
- Report a bug

What you need in front of you

This guide is for the developer adding Flutterwave to a Laravel app. Have these ready and setup goes straight through. Tick them off as you go.



A Flutterwave for Business account

You can finish this whole guide with test keys before your account is approved for live payments.



Your API keys

Public key, secret key and encryption key, from the [API keys page](#) on your dashboard. The secret key is the password to your money: keep it in `.env`, never in code or Git.



PHP 8.1 or newer, and Laravel 9, 10, 11, 12 or 13

Check with `php -v` and `php artisan --version`.



Composer, and access to the app's `.env`

The installer writes the Flutterwave keys into `.env` for you, so it needs to exist.



A domain or `localhost`, not an IP address

Flutterwave's checkout doesn't load on an address like `127.0.0.1`. Open your local app on `localhost` instead.



For production: HTTPS and a shared cache

Webhooks need a public HTTPS URL, and a cache every server shares (redis or database) so each payment event is handled once.

Install the package

Composer pulls it from Packagist. Pick the tab that matches your app: most apps want the first one.

COMPOSER

For any Laravel app using Blade, an API or Livewire. Updates arrive with `composer update`.

1 Require the package

Laravel finds the service provider and the Flutterwave facade on its own.

```
composer require abraham-flutterwave/laravel-payment
```

2 Run the installer

It publishes `config/flutterwave.php`, offers to add the four Flutterwave keys to `.env`, and prints the webhook URL you'll need in step two.

```
php artisan flutterwave:install
```

3 Add anything else you need

The installer publishes only the config. Add `--billable` to save customers' cards, or publish the example checkout routes to start from working code:

```
php artisan vendor:publish --tag=flutterwave-routes
```

Don't publish with `--provider`

Publishing every tag at once puts all three frontends' components in one folder and adds routes and a migration you may not want. Publish by tag instead.

For Inertia apps with React, Vue 3 or Svelte 5. You get a pay button and a bank transfer component, styled like Flutterwave's own screens.

1 Require the package

```
composer require abraham-flutterwave/laravel-payment
```

2 Install with your frontend

Pick one framework. The components are copied into `resources/js/components/flutterwave`, so they're yours to edit and add no npm dependency.

```
php artisan flutterwave:install --react # or --vue, or --svelte
```

3 Rebuild your assets

Run your usual Vite build so the components, stylesheet and fonts are bundled. Skip this and the imports won't resolve.

```
npm run build
```

STEP TWO

Connect your Flutterwave account

This is where the app learns which account to charge and how to recognise Flutterwave's payment notifications. Keep your editor and the Flutterwave dashboard open side by side.

1 Copy your keys into .env

On the dashboard, open the [API keys page](#). Copy the public, secret and encryption keys into the lines the installer added:

```
FLW_PUBLIC_KEY=FLWPUBK_TEST-xxxxxxxxx-X  
FLW_SECRET_KEY=FLWSECK_TEST-xxxxxxxxx-X  
FLW_ENCRYPTION_KEY=xxxxxxxxxxxxxx
```

Keep both keys from the same mode

Test keys contain `_TEST`. Mixing a test key with a live one is the most common setup fault, and it shows up as an authorisation error at checkout, not as a config error.

2 Choose a secret hash

Invent a long random string and treat it like a password. Put it in `.env`, then type the same value into [Settings → Webhooks → Secret hash](#) on the dashboard. It's how your app proves a payment notification really came from Flutterwave.

```
FLW_SECRET_HASH=choose-a-long-random-string
```

Leave it empty and the package rejects every webhook, on purpose.

3 Add your webhook URL

On the same dashboard screen, paste your app's webhook URL. The package registers this route for you; the installer printed the exact address for your app.

```
https://your-app.com/flutterwave/webhook
```

Replace `your-app.com` with your real domain. If you changed `FLW_WEBHOOK_PATH`, use that path instead.

4 Clear cached config

If your app caches config, Laravel won't see the new `.env` values until you clear it.

```
php artisan config:clear
```

5 Run the doctor

It checks your keys, webhook secret hash and URL, cache, queue and admin routes, then asks Flutterwave to confirm your secret key. Fix anything marked as failing before you move on.

```
php artisan flutterwave:doctor
```

SETTINGS REFERENCE

FLW_PUBLIC_KEY	Identifies your account in the browser. Needed for the inline checkout and pay buttons.
FLW_SECRET_KEY	Authorises every API call from your server. Never commit it or share it.
FLW_ENCRYPTION_KEY	Encrypts the payloads Flutterwave requires encrypted. From the same API keys page.
FLW_SECRET_HASH	Your shared password for webhooks. Must match the dashboard exactly. Empty means every webhook is rejected.
FLW_REDIRECT_URL	Where customers land after paying. Defaults to <code>APP_URL/flutterwave/payment/callback</code> .
FLW_DEFAULT_CURRENCY	Used when a checkout doesn't name a currency. Defaults to NGN ; set <code>FLW_DEFAULT_COUNTRY</code> to match.

FLW_WEBHOOK_DEDUPE_STORE

The cache store that remembers handled webhooks. **Use redis or database on more than one server.**

FLW_WEBHOOK_QUEUE

Queue name to handle webhooks in the background. Empty handles them during the request.

Every option, with its default, is in `config/flutterwave.php` and on the documentation site's config reference.

STEP THREE

Take a test payment

Do this once before launch. It takes a few minutes and catches key mismatches, webhook faults and currency problems in one pass.

1 Confirm you're on test keys

Both keys in `.env` should contain `_TEST`. `flutterwave:doctor` says which mode it sees.

2 Start a checkout

The quickest way is from the terminal. It creates a hosted payment link and opens it in your browser:

```
php artisan flutterwave:pay you@example.com 100 --open
```

Or from your own code, which is what your customers will use:

```
use Flutterwave\Payments\Facades\Flutterwave;

return Flutterwave::redirect([
    'amount' => 5000,
    'email' => auth()->user()->email,
    'tx_ref' => $order->reference, // so you can match it later
]);
```

3 Pay with a test card

Use a card from Flutterwave's [testing documentation](#). It covers success, failure and OTP flows. Real cards are always declined with test keys.

4 Listen for the payment

Fulfil orders from the webhook event, not the redirect: customers close tabs, and bank transfers can finish minutes later. The package has already checked the signature and re-fetched the charge from the API when this runs.

```
use Flutterwave\Payments\Events\ChargeCompleted;
use Illuminate\Support\Facades\Event;

// In AppServiceProvider::boot()
Event::listen(function (ChargeCompleted $event) {
    if (!$event->isSuccessful()) {
        return;
    }

    $order = Order::firstWhere('reference', $event->reference());

    if ($order
        && $event->amount() >= $order->amount
        && $event->currency() === $order->currency) {
        $order->markAsPaid();
    }
});
```

Flutterwave can't reach localhost. To send a signed webhook for your test payment to your local app, use the transaction id from the dashboard:

```
php artisan flutterwave:ping-webhook 4975363
```

5 Check all three places agree

A healthy test shows the same transaction in each:

- `php artisan flutterwave:verify <transaction id>` reports it as successful
- Your order is marked paid, and `storage/logs/flutterwave.log` shows the `charge.completed` webhook arriving
- The Flutterwave dashboard lists the payment under Transactions

All three match?

Your keys, webhook URL and secret hash are all correct. You're ready to go live.

STEP FOUR

Go live

Switching to real money is a handful of changes on your production server. Work through them in order.



Swap in your live API keys

Replace all three test keys in the production `.env` with the live set from the dashboard. Both keys must be live.



Set a fresh secret hash

A new random value in `.env` and in the dashboard's webhook settings. Don't reuse the one from testing.



Point the webhook at production over HTTPS

`https://your-domain/flutterwave/webhook`, with a shared cache store (and a running queue worker, if you set `FLW_WEBHOOK_QUEUE`).



Re-cache config and run the doctor

Run `php artisan config:cache`, then `php artisan flutterwave:doctor` on the production server. Every line should pass.



Make one small real purchase, then refund it

Use your own card, confirm the order is marked paid, then refund it with `Flutterwave::refund($transactionId)` or from the dashboard.



Note the version you're on

`composer show abraham-flutterwave/laravel-payment` tells you. Fixes arrive in releases, and support will ask.

When something's not right

Search by what you're seeing. Each entry explains what's actually happening before telling you what to change. Run `php artisan flutterwave:doctor` first: it catches most of these on its own.

— The payment modal or checkout page doesn't load

WHAT'S HAPPENING

Flutterwave's checkout refuses to load on a bare IP address, and the inline modal can't start without your public key.

TRY IN THIS ORDER

1. Open your app on `localhost` or a domain, not `127.0.0.1`.
2. Check `FLW_PUBLIC_KEY` is set, then run `php artisan config:clear`.
3. Using the Inertia components? Rebuild your assets with `npm run build`.

— "Invalid authorization key" or another authentication error

WHAT'S HAPPENING

The secret key Flutterwave received doesn't belong with the request. Almost always a test key mixed with a live one, or a stale value from cached config.

TRY IN THIS ORDER

1. Run `php artisan flutterwave:doctor`. It flags test and live keys mixed together.
2. Re-copy the keys from the dashboard. A trailing space breaks them silently.
3. Run `php artisan config:clear` (or `config:cache` in production) after editing `.env`.
4. Check you copied the keys from the right account if you have more than one.

- The customer paid, but the order is still unpaid

WHAT ' S HAPPENING

Your app never handled the webhook. It didn't arrive, was rejected, or arrived with nothing listening for `ChargeCompleted`.

TRY IN THIS ORDER

1. Check `storage/logs/flutterwave.log`. No `charge.completed` line means the webhook never got through.
2. Confirm the dashboard's webhook URL matches your app's, including `https://` and the path.
3. Confirm you registered a `ChargeCompleted` listener, and that it matches orders by `tx_ref`.
4. Set `FLW_WEBHOOK_QUEUE`? Make sure a queue worker is running for that queue.
5. Testing locally? Flutterwave can't reach you; use `php artisan flutterwave:ping-webhook <id>`.

In the meantime

Confirm the payment with `php artisan flutterwave:verify <id>` and mark the order paid by hand, so the customer isn't left waiting. Then `flutterwave:resend-webhook <id>` once it's fixed.

- Every webhook gets a 401

WHAT ' S HAPPENING

The signature check failed. The secret hash in your `.env` doesn't match the one on the dashboard, or it's empty, which rejects everything by design.

TRY IN THIS ORDER

1. Retype `FLW_SECRET_HASH` in both places rather than trusting a copy.
2. Clear cached config so the app sees the new value.
3. Check nothing in front of your app (a proxy or firewall) strips the `flutterwave-signature` or `verif-hash` header.

- Webhooks fail with a 500 "Transaction verification failed"

WHAT ' S HAPPENING

Before firing `ChargeCompleted`, the package re-fetches the charge from the API so a forged payload can't mark an order paid. That lookup failed, so it answered 500 and Flutterwave will retry.

TRY IN THIS ORDER

1. Check your keys are from the same mode (test or live) as the payment.
2. Sending your own test payload? Use a real test transaction id, not a made-up one.
3. Look in `storage/logs/flutterwave.log` for the API's error message.

- Orders are fulfilled twice, or events fire more than once

WHAT ' S HAPPENING

Flutterwave retries webhooks until it gets a 2xx. The package remembers handled webhooks in your cache, but a cache that forgets between requests, or isn't shared between servers, can't do that.

TRY IN THIS ORDER

1. Set `FLW_WEBHOOK_DEDUPE_STORE=redis` (or database). The array store forgets everything.
2. Run `php artisan flutterwave:doctor`; it tests that the cache keeps values and can lock.
3. Make your listener idempotent too: skip orders already marked paid.

- Test transfers stay "PENDING" forever

WHAT 'S HAPPENING

With test keys, Flutterwave leaves transfers pending unless the reference says how they should end. Sandbox transfers also need your server's IP whitelisted on the dashboard.

TRY IN THIS ORDER

1. Pass 'test_outcome' => 'success' (or 'failure') when you create the transfer.
2. Whitelist your server's IP address on the dashboard.
3. Use Flutterwave's test bank accounts; real account numbers don't resolve in test mode.

- The admin page or chargeback screen returns 403

WHAT 'S HAPPENING

These screens refund customers and move money, so every action checks the `flutterwave.manage` ability. Until you define it, everyone is refused, even signed-in admins.

TRY IN THIS ORDER

1. Define the ability in `AppServiceProvider::boot()`:

```
Gate::define('flutterwave.manage', fn (User $user) => $user->is_admin);
```
2. Check `Route::flutterwaveAdmin()` sits inside your authenticated route group.

Talk to a human

Pick the channel that matches what you need. Code and setup questions go to the developer channels; questions about money that has already moved go through your Flutterwave dashboard.

FASTEST

Developer Slack

The Flutterwave developer community.
Good for setup questions, error messages
and "is this normal?"

When you're stuck mid-setup

EMAIL

developers@flutterwavego.com

The developer experience team. Include
your package version, Laravel and PHP
versions, and what you've already tried.

For detail that needs attachments

REFERENCE

Flutterwave error reference

What each API error message means. The
package passes Flutterwave's message
through in FlutterwaveException.

When an API call fails

BUG

Report it on GitHub

If the package itself is at fault, an issue gets
it in front of the engineers who maintain it.

For reproducible faults

Tell us what broke

Open an issue on [the package's GitHub issues](#). Search first: if someone has hit it already, add to their thread rather than opening a new one. Questions belong in Slack, not issues.

Never paste your secret key, encryption key or secret hash

Issues are public. Redact keys, customer names, emails and card details from logs before you post. If you've already exposed a key, roll it on the dashboard immediately.

Copy this, fill it in, and paste it into the issue:

Description

What went wrong, in one or two sentences.

Steps to reproduce

- 1.
- 2.
- 3.

****Expected:**** what should have happened

****Actual:**** what happened instead

****Happens:**** every time / sometimes / once

Configuration

- Package version (composer show abraham-flutterwave/laravel-payment):
- Laravel version:
- PHP version:
- Frontend: Blade / Livewire / Inertia (React, Vue, Svelte) / API
- Environment: test keys / live keys
- Output of php artisan flutterwave:doctor --offline:

Evidence

Error message, stack trace, or lines from storage/logs/flutterwave.log, with keys and customer details removed.

A good report answers "what did you expect, what happened, and how do I make it happen on my machine?" A failing test is the best possible version of that.

Flutterwave.

Developer quickstart for Flutterwave for Laravel 2.3.0 · Generated Oct 2026

Full [changelog](#) and [source](#) on GitHub, and every feature on the [documentation site](#).