EasySale – Stripe Connect Setup Guide (OAuth)

Version: 1.0 • Date: 2026-01-30

What this enables

EasySale (your POS platform) acts as the **Stripe Connect platform**. Each tenant/merchant connects their own Stripe account via OAuth. In the app, users click **Connect Stripe**, complete Stripe authorization, and EasySale stores a tenant-mapped connected account ID + tokens. Phase 2 may add Hosted Checkout Sessions and webhooks; Phase 1 focuses on connect/test/status/summary/logs.

Important constraints (matches your build philosophy)

- No hardcoded credentials: all platform credentials come from environment variables / secret store.
- Real flows only: no 'Coming Soon' placeholders; the Stripe card is shown only when the backend endpoints exist.
- Tenant isolation: every DB read/write includes tenant_id. Disconnect must remove tenant mapping and revoke tokens.
- No terminal hardware capture in this guide (Hosted Checkout is the payment flow for now).

A. Platform owner setup (you do once in your Stripe dashboard)

Complete these steps while logged into the Stripe account that represents **EasySale** (the platform).

1. **1) Create/finish the EasySale Stripe account**: complete business profile prompts (can skip some, but you will eventually need full activation for live).
2. **2) Enable Stripe Connect**: in Stripe Dashboard, open **Connect** and enable it for the account.
3. **3) Choose account type**: start with **Standard** (fastest, merchants manage their own Stripe dashboard). If you want more white-label later, move to Express/Custom.
4. **4) Create or confirm your Connect application settings**: find your **Client ID** (looks like **ca_...**).
5. **5) Configure OAuth redirect URL**: add the redirect URI that EasySale backend serves, e.g. **https://YOURDOMAIN/api/integrations/stripe/callback**. For local dev, allow **http://localhost:PORT/api/integrations/stripe/callback** in test mode only.
6. **6) Create API keys for the platform**: get the platform **Secret key** (**sk_...**) used by the backend. Store it securely (never in git).
7. **7) Webhook endpoint (Phase 2)**: create a webhook endpoint pointing to **https://YOURDOMAIN/api/payments/webhooks/stripe**. Select events: *checkout.session.completed*, *checkout.session.expired* (and optionally *payment_intent.succeeded*). Copy the webhook signing secret **whsec_...**.

8 **8) Test-mode sanity check**: use the test keypair first. You should be able to OAuth-connect a test Stripe merchant account and see a summary in EasySale.

Outputs to capture (paste into EasySale config)

What	Where you get it	Used for
STRIPE_CLIENT_ID (ca_...)	Dashboard → Connect settings	Build OAuth authorization URL + token exchange
STRIPE_SECRET_KEY (sk_...)	Dashboard → Developers → API keys	Token exchange + API calls (platform key)
STRIPE_REDIRECT_URI	You decide (must match dashboard)	OAuth callback route in backend
STRIPE_WEBHOOK_SECRET (whsec_...)	Webhook endpoint details (Phase 2)	Verify webhook signatures

B. Merchant/tenant connect flow (what your users do in EasySale)

Once the platform is configured, each tenant can connect their own Stripe account from within EasySale. This is the real, production flow—no API key pasting by end users.

- User clicks **Connect Stripe** in Settings → Integrations.
- Backend returns **auth_url**; frontend opens it (popup or redirect).
- User logs in to Stripe (or creates an account) and approves access.
- Stripe redirects to EasySale callback with **code** + **state**.
- Backend validates state, exchanges code for tokens, stores tenant mapping (**acct_...**), logs event, and redirects back to Integrations page.
- Integrations page shows: Connected + summary (business name, country, currency, masked account ID).

C. Implementation contract (what the developer must ensure)

- **Endpoints exist (Phase 1):** POST /api/integrations/stripe/auth-url, GET /api/integrations/stripe/callback, GET /api/integrations/stripe/status, GET /api/integrations/stripe/summary, POST /api/integrations/stripe/test, DELETE /api/integrations/stripe/disconnect, GET /api/integrations/stripe/logs.
- **Disconnect:** revoke tokens using Stripe deauthorization, then remove tenant mapping.
- **Security:** never return access tokens; mask account IDs; redact secrets from logs.
- **State/CSRF:** persist oauth state with expiry and single-use semantics.
- **Summary:** call Stripe API and return real account info (not mocked).

D. Troubleshooting (quick)

- **OAuth redirect mismatch:** Ensure STRIPE_REDIRECT_URI exactly matches the value registered in Stripe dashboard (including scheme/port/path).
- **State invalid/expired:** Confirm callback state exists in DB and hasn't expired; ensure single-use delete-on-success.
- **Connected but summary fails:** Confirm platform secret key is correct and API call uses Stripe-Account header OR /v1/accounts/{acct}.
- **Live vs Test confusion:** Client ID and keys differ between test mode and live mode; keep them consistent per environment.