FREE CHECKLIST · PAYMENTS ENGINEERING

Never charge twice again.

5 payment fixes Stripe uses, and how to copy them in your own checkout, with code.

Wi-Fi drops → app retries → customer charged twice → angry DM → dispute → $15 fee you keep paying even if you win.

STRIPE US STANDARD PRICING: $15 PER DISPUTE RECEIVED, +$15 TO COUNTER IT (REFUNDED ONLY IF YOU WIN) · VIA CHARGEFLOW, 2026

How to use this: each fix has the **pain** it prevents, the **fix**, copy-paste code (Node / Postgres) and its honest **limit**. Tick a box when it's true for your app, then run the 10-minute audit on the last page.



Afeez Badmos · @afeez_sdbytes

SOFTWAREDEVBYTES × GLOBAL MCLIEN SOFTWARE SOLUTIONS

01 Idempotency keys on every payment

PAIN Wi-Fi drops, the customer taps Pay again, your app retries on a timeout. Without a key, each request is a brand-new charge. That's the double charge in the video.

FIX Create one key per checkout attempt, store it with the order, and send it on every create call. Stripe saves the first response for that key and returns it to any retry instead of charging again. Disable the Pay button while a request is pending.

```
// Node · create the key ONCE per checkout attempt, persist it with the order
const key = order.idempotencyKey ?? (order.idempotencyKey = crypto.randomUUID());
await saveOrder(order);
const pi = await stripe.paymentIntents.create(
  { amount: 2000, currency: "usd", customer: order.customerId },
  { idempotencyKey: key }   // retries replay the first response
);
```

LIMIT A new key per click defeats the point. Stripe can prune keys once they're at least 24 hours old, and a reused key with different parameters returns an error, so keep the key tied to one order.

02 Never touch the card number

PAIN If raw card numbers pass through your server or logs, a single breach exposes every customer, and your PCI DSS scope covers your whole backend.

FIX Use a hosted card field (Stripe Payment Element or Checkout). The card goes straight from the customer's browser to Stripe inside an iframe; your server only ever receives a token or PaymentIntent ID.

```
// Browser · the card field is an iframe served by Stripe
const elements = stripe.elements({ clientSecret });
elements.create("payment").mount("#payment");
await stripe.confirmPayment({ elements, confirmParams: { return_url: RETURN_URL } });
// Server · store pi.id / pm_… IDs only. Never log req.body from a card form.
```

LIMIT Hosted fields shrink your PCI scope a lot, but they don't remove it: you still need to protect the page that loads them (scripts, CSP, dependencies).

03 A double-entry ledger

PAIN A single `balance` column you UPDATE in place can't tell you where money went. A refund bug or a race condition quietly loses cents, and nobody notices until reconciliation.

FIX Record every money movement as at least two append-only entries that sum to zero, written in one database transaction. Check the invariant automatically and alert on anything that doesn't balance.

```
CREATE TABLE ledger_entries (
  id           bigserial PRIMARY KEY,
  txn_id       uuid   NOT NULL,
  account      text   NOT NULL,
  amount_cents bigint NOT NULL,           -- integer cents · negative = out
  created_at   timestamptz NOT NULL DEFAULT now()
);
-- alert on any transaction that doesn't balance
SELECT txn_id, SUM(amount_cents) FROM ledger_entries
GROUP BY txn_id HAVING SUM(amount_cents) <> 0;
```

LIMIT Never edit or delete entries; post a reversing entry instead. Use integer cents (or a decimal type), never floats.

04 Verify (and dedupe) webhooks

PAIN An unsigned webhook endpoint believes anyone. A fake `payment_intent.succeeded` POST can mark an unpaid order as paid. And because Stripe retries, a non-idempotent handler can ship the same order twice.

FIX Verify the `Stripe-Signature` header against the raw request body with your endpoint secret. Record each `event.id` you've processed and skip repeats. Return 2xx quickly and do the slow work in a queue. In live mode Stripe keeps retrying failed deliveries for up to 3 days.

```
app.post("/webhooks", express.raw({ type: "application/json" }), async (req, res) => {
  let event;
  try {
    event = stripe.webhooks.constructEvent(req.body, req.headers["stripe-signature"],
process.env.STRIPE_WEBHOOK_SECRET);
  } catch { return res.sendStatus(400); }      // bad or missing signature
  if (await alreadyProcessed(event.id)) return res.sendStatus(200);  // retries happen
  await enqueue(event); res.sendStatus(200);
});
```

LIMIT Signature checks need the raw body: JSON-parsing middleware that runs first breaks verification. Use a separate secret per endpoint and rotate it if it leaks.

05 Protect checkout under load

PAIN On your biggest sales day, report exports and order-history pages compete with checkout for the same servers and database. Everything slows down together, and paying customers time out.

FIX Rate-limit the non-critical routes first so they return `429 Retry-After` under pressure, and give checkout its own capacity (worker pool, DB connection pool or service) that the slow lane can't eat.

```
import rateLimit from "express-rate-limit";
const slowLane = rateLimit({ windowMs: 60_000, limit: 60 });   // per client, per minute
app.use(["/orders/history", "/reports"], slowLane);       // shed these first
app.post("/checkout", checkoutHandler);                   // separate pool, never behind the slow lane
```

LIMIT Rate limits protect capacity; they don't add it. Load-test checkout at your expected peak before the sale, not during it.

The 10-minute audit YES / NO

QUESTION	HOW TO CHECK	ANSWER
Does every payment create call send an idempotency key that survives a retry?	Search your code for `paymentIntents.create` / `charges.create`; confirm the key is stored with the order, not generated per click.	☐ Y ☐ N
Can a card number ever reach your server or logs?	Submit a test card, then grep request logs and error trackers for `4242`.	☐ Y ☐ N
Does every transaction in your ledger sum to zero?	Run the `HAVING SUM(...) <> 0` query above. Zero rows = pass.	☐ Y ☐ N
Does your webhook endpoint reject an unsigned POST?	`curl -X POST yoursite.com/webhooks -d '{}'` should return 400. Then replay a real event twice; the second must be a no-op.	☐ Y ☐ N
Will checkout stay fast if reports get hammered?	Load-test `/reports` and `/checkout` together; checkout latency should hold steady.	☐ Y ☐ N

Scored a "no"? Start here

Fix 01

First. Double charges turn straight into disputes and refunds.

Fix 04

Second. Fake or repeated "paid" events ship orders you never got paid for.

02 · 03 · 05

Then the rest: card scope, a balanced ledger and peak-day capacity.

Want it done for you?

Global McLien Software Solutions audits and builds payment flows: idempotent checkouts, hosted card fields, ledgers, verified webhooks and load-ready infrastructure.

globalmcliensoftwaresolutions.com →

Sources
Stripe API reference, "Idempotent requests" (keys replay the first response; may be pruned after 24 hours): docs.stripe.com/api/idempotent_requests
Stripe docs, "Manage event destinations" (live mode retries for up to 3 days with exponential back-off): docs.stripe.com/workbench/webhooks
Stripe 2025 annual letter ($1.9 trillion total volume, up 34% from 2024): stripe.com/newsroom/news/stripe-2025-update
Chargeflow, "Stripe dispute fees" ($15 per dispute received, +$15 counter fee refunded only on a win, US standard pricing): chargeflow.io/blog/stripe-dispute-fees

General engineering guidance, not legal, financial or PCI compliance advice. Fees and Stripe behaviour can change and vary by country and contract, so check Stripe's current docs and your own agreement. Stripe is a trademark of Stripe, Inc.; this guide isn't affiliated with or endorsed by Stripe. © 2026 Afeez Badmos / SoftwareDevBytes · Global McLien Software Solutions.