EcomLabs

GUÍA PASO A PASO

Cobros dobles: Idempotency-Key en Node

El ejemplo completo en Node, cliente y servidor, para que un reintento, un doble clic o un timeout cobren una sola vez. Con una demo con curl que muestra cada caso y lo que falta para llevarlo a producción.

PARA	Checkouts y APIs de pago en Node.js
PROBADO CON	Node 22.22.3 · curl 8.19 · Windows 11
ACTUALIZADA	30 de octubre de 2026
DOCUMENTACIÓN	docs.stripe.com/api/idempotent_requests
ARTÍCULO	ecomlabs.dev/g/pagos
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	El servidor
Paso 02	El cliente
Paso 03	Prueba la demo con curl
Paso 04	Antes de llevarlo a producción
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Un servidor de pagos en Node que guarda cada `Idempotency-Key` con la huella del cuerpo y la respuesta, y un cliente que crea una clave por intento de pago y la repite en cada reintento. Con eso, un doble clic, un reintento o un timeout cobran una sola vez. Todo usa solo módulos de Node: no hay nada que instalar.

Lo probamos el 30 de octubre de 2026 con Node 22.22.3 y curl 8.19 en Windows 11. El cliente necesita `fetch`, `AbortSignal.timeout` y `crypto.randomUUID()` globales: están en cualquier navegador actual (en contexto seguro, HTTPS) y en Node 22.

PASO 01

El servidor

Copia este archivo como `server.mjs`. Recibe la función que cobra (`charge`), así puedes conectarlo a tu proveedor real o a uno falso para probar:

```
import { createServer } from "node:http";
import { createHash } from "node:crypto";

export function createPaymentServer(charge) {
  const keys = new Map();

  const send = (res, status, body, headers = {}) => {
    res.writeHead(status, { "content-type": "application/json", ...headers });
    res.end(JSON.stringify(body));
  };

  return createServer(async (req, res) => {
    if (req.method !== "POST" || req.url !== "/payments") return send(res, 404, { error: "not found" });

    const key = req.headers["idempotency-key"];
    if (!key || key.length > 255) return send(res, 400, { error: "Idempotency-Key is missing or too long" });

    let raw = "";
    for await (const chunk of req) raw += chunk;
    let payment;
    try {
      payment = JSON.parse(raw);
    } catch {
      return send(res, 400, { error: "body is not JSON" });
    }
    if (!Number.isInteger(payment?.amount) || payment.amount <= 0 || !/^[a-z]{3}$/.test(payment.currency)) {
      return send(res, 400, { error: "amount (integer, cents) and currency (e.g. mxn) are required" });
    }
    const fingerprint = createHash("sha256").update(raw).digest("hex");
```

```
    const saved = keys.get(key);
    if (saved) {
      if (saved.fingerprint !== fingerprint) return send(res, 422, { error: "Idempotency-Key already used with another body" });
      if (!saved.done) return send(res, 409, { error: "a request with this Idempotency-Key is in progress" });
      return send(res, saved.status, saved.body, { "idempotent-replayed": "true" });
    }

    keys.set(key, { fingerprint, done: false });

    let status, body;
    try {
      body = await charge(payment);
      status = 201;
    } catch {
      status = 500;
      body = { error: "payment outcome unknown; check the order before trying again" };
    }
    keys.set(key, { fingerprint, done: true, status, body });
    send(res, status, body);
  });
}
```

El orden importa: validar, calcular la huella, buscar la clave y reservarla antes de cobrar. Entre `keys.get` y `keys.set` no hay ningún `await`, así que dos peticiones con la misma clave no pueden pasar las dos.

PASO 02

El cliente

Copia este archivo como `client.mjs`. `postPayment` reintenta con la misma clave; `onPay` es lo que conectas al botón:

```
const sleep = (ms) => new Promise((resolve) => setTimeout(resolve, ms));

export async function postPayment(url, payment, key, { retries = 3, timeoutMs = 10000 } = {}) {
  for (let attempt = 0; ; attempt++) {
    try {
      const res = await fetch(url, {
        method: "POST",
        headers: { "content-type": "application/json", "idempotency-key": key },
        body: JSON.stringify(payment),
        signal: AbortSignal.timeout(timeoutMs),
      });
      if ([409, 502, 503, 504].includes(res.status) && attempt < retries) {
        await sleep(200 * 2 ** attempt);
        continue;
      }
      return { status: res.status, replayed: res.headers.get("idempotent-replayed") === "true", body: await res.json() };
```

```
    } catch (err) {
      if (attempt >= retries) throw err;
      await sleep(200 * 2 ** attempt);
    }
  }
}

let key = null;

export async function onPay(payment, url = "/payments") {
  key ??= crypto.randomUUID();
  const result = await postPayment(url, payment, key);
  if (result.status < 500 && result.status !== 409) key = null;
  return result;
}
```

La regla: la clave se crea al primer clic del intento y se descarta solo tras una respuesta definitiva (un `201` o un `4xx` que no sea `409`). Tras un error de red, un `409` o un `5xx` se conserva.

PASO 03

Prueba la demo con curl

Guarda `demo.mjs` y arráncalo con `node demo.mjs`:

```
import { createPaymentServer } from "./server.mjs";

let charges = 0;
const charge = async ({ amount, currency }) => {
  await new Promise((resolve) => setTimeout(resolve, 300));
  return { id: `ch_${++charges}`, amount, currency, status: "succeeded" };
};

createPaymentServer(charge).listen(8787, () => console.log("POST http://localhost:8787/payments"));
```

En otra terminal con bash (en Windows, Git Bash) guarda y corre `bash curl.sh`:

```
KEY=8e03978e-40d5-43e8-bc93-6894a57f9324
BODY='{"orderId":"A-1001","amount":129900,"currency":"mxn"}'
pay() { curl -s http://localhost:8787/payments -H "content-type: application/json" -H "Idempotency-Key: $1" -d
"$2" -w '  -> %{http_code} %header{idempotent-replayed}\n'; }

pay $KEY "$BODY"
pay $KEY "$BODY"
pay $KEY '{"orderId":"A-1001","amount":1,"currency":"mxn"}'
pay other-key "$BODY" & pay other-key "$BODY"; wait
```

Esta es la salida que obtuvimos. Las dos últimas líneas pueden salir en el otro orden:

```
{"id":"ch_1","amount":129900,"currency":"mxn","status":"succeeded"}  -> 201
{"id":"ch_1","amount":129900,"currency":"mxn","status":"succeeded"}  -> 201 true
{"error":"Idempotency-Key already used with another body"}  -> 422
{"error":"a request with this Idempotency-Key is in progress"}  -> 409
{"id":"ch_2","amount":129900,"currency":"mxn","status":"succeeded"}  -> 201
```

PASO 04

Antes de llevarlo a producción

- **Un almacén compartido.** El `Map` vive en un proceso. Con varias instancias, usa una tabla con la clave como `PRIMARY KEY` y resérvala con `INSERT ... ON CONFLICT (key) DO NOTHING RETURNING key`: solo la primera petición recibe la fila. Es nuestra recomendación; ese SQL no está en las pruebas de esta guía.
- **Caducidad.** Stripe guarda sus claves al menos 24 horas. Define tu plazo, publícalo y borra las claves viejas.
- **Una clave hacia el proveedor.** Stripe acepta `Idempotency-Key` en todos sus `POST`. Manda una clave estable por intento, para que tu propio reintento hacia Stripe tampoco cobre dos veces.
- **Claves por cuenta.** Guarda cada clave junto a la cuenta que la envió, para que nadie reciba la respuesta guardada de otro.

> **CUIDADO**
>
> No borres la clave cuando el cobro falla con un error desconocido. Si el proveedor sí cobró, el reintento cobra otra vez. Guarda el `500` y concilia el pedido (webhooks, consulta al proveedor) antes de abrir un intento con otra clave.

AL TERMINAR

Checklist

- [ ] El cliente crea la clave al empezar el intento de pago, con `crypto.randomUUID()`.
- [ ] Los reintentos y los clics repetidos mandan la misma clave y el mismo cuerpo.
- [ ] El servidor reserva la clave antes de cobrar.
- [ ] Una clave con otro cuerpo recibe `422`; una en curso, `409`.
- [ ] Las repeticiones devuelven la respuesta guardada, también los `500`.
- [ ] Las claves viven en un almacén compartido y caducan según un plazo escrito.
- [ ] Tu servidor manda también una clave al proveedor de pagos.
- [ ] `curl.sh` da `201`, la misma respuesta repetida, `422` y un `409` para la petición simultánea.