SolintPay

Pasarela de pagos para Venezuela — Documentación para desarrolladores

Guía de integración con SolintPay

Esta es la guía que un desarrollador de un comercio (PadelBox, una clínica, un e-commerce, etc.) sigue para integrar **SolintPay** en su sistema y empezar a cobrar por Pago Móvil.

> **Tiempo estimado:** 2-4 horas para una integración completa con webhooks.
>
> **Cómo usar esta guía:** está escrita pensando en **producción**. Intégrala usando los valores de ejemplo (los `placeholders` tipo `sk_test_TU_KEY`), y cuando SolintPay te entregue tus credenciales reales, **solo las reemplazas** — tu código no cambia.

---

1. Lo que SolintPay te entrega

Antes de escribir una línea de código, SolintPay te da estas cosas:

Dato	Formato	Para qué
API key de prueba	`sk_test_…`	Integrar y probar en el sandbox sin mover dinero real
API key de producción	`sk_live_…`	Cobros reales — te la damos al aprobar tu registro (KYC)
URL base del API	La que te confirme SolintPay (hoy: `https://solintpay.vercel.app`)	A dónde apuntan todas tus llamadas
Webhook secret	`whsec_…`	Verificar que las notificaciones vienen de verdad de SolintPay — te lo damos cuando registras tu webhook

¿Cómo obtienes tu API key? (autoservicio, 2 minutos)

1. Entra a `https://solintpay.vercel.app/panel/registro`, crea tu cuenta con el RIF de tu empresa y acepta los términos. **Recibes tu `sk_test_…` al instante** (en pantalla y por correo) — no tienes que hablar con nadie.
2. Integras y pruebas con ella en el sandbox (no mueve dinero real).
3. Tu panel te muestra el **"Camino a producción"**: qué documentos subir (KYC: RIF + cédula del dueño, en **Verificación**) y qué dato pedirle a tu banco (el ChildClientID del BNC). Cuando esté todo, SolintPay activa tu **modo producción** y tu key empieza a cobrar de verdad.

> Si eres el programador del comercio: **pídele al dueño la API key** (la ve en su panel → API Keys, donde también puede generar otra). No se genera desde el código ni desde esta guía.

> 💡 **¿Necesitas cobrar antes de integrar?** El dueño del comercio ya tiene su **panel** (`/panel`): ahí puede cobrar directo desde **Cobrar (POS)** sin escribir código, ver todas sus transacciones, confirmar pagos Zelle, gestionar API keys y webhooks, y seguir el **"Camino a producción"** — un checklist que le dice exactamente qué dato le falta para operar con dinero real. Esta guía es para conectar **tu sistema o tu app** a ese mismo motor.
>
> La clave se muestra **una sola vez** y se guarda **cifrada** (hash); SolintPay no puede recuperarla después. Si se pierde, se genera una nueva (rotación).

> 🔒 **Tu API key y tu webhook secret son SECRETOS.** Guárdalos en variables de entorno (`SOLINTPAY_API_KEY` , `SOLINTPAY_WEBHOOK_SECRET`), **nunca** en el código fuente, en el repositorio ni en el frontend. Ver la sección 🔒 Seguridad.

Para que los ejemplos de abajo se copien y peguen fácil, define estas dos variables en tu terminal:

```
export BASE="https://solintpay.vercel.app"   # URL base que te dio SolintPay
export KEY="sk_test_TU_KEY"                  # tu API key (de prueba primero)
```

---

2. Probar conectividad (10 min)

Primero confirma que tu API key funciona:

```
curl -H "Authorization: Bearer $KEY" "$BASE/v1/ping"
```

Esperado:

```
{ "ok": true, "cliente": "Tu Comercio", "env": "test", "timestamp": "..." }
```

El campo `env` te dice si estás en `test` (sandbox) o `live` (producción), según el tipo de API key que usaste.

---

3. Hacer un cobro (30 min)

Un cobro C2P (Cliente-a-Persona) funciona así: el cliente final te **dicta** su cédula, su teléfono, su banco y un **Token** que él generó en la app de su banco. Con esos datos, SolintPay le debita al instante.

> ⚠️ **Esos 4 datos del cliente son entrada dinámica** — los recibes de tu cliente en el momento de pagar (desde tu formulario, tu POS, etc.). **No los hardcodees.** Para *probar* en sandbox hay datos de ejemplo en el 💉 Apéndice: Modo de prueba.

```
curl -X POST "$BASE/v1/cobros" \
  -H "Authorization: Bearer $KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "metodo": "c2p_bnc",
    "monto": 25.50,
    "moneda": "VES",
    "cliente": {
      "cedula": "V12345678",
      "telefono": "584141234567",
      "banco": "0102",
      "token": "1234567"
    },
    "referencia_externa": "RESERVA-001",
    "concepto": "Cancha 1 - 18:00"
  }'
```

> El header `Idempotency-Key` (un UUID único por intento) evita cobros duplicados si tu red falla y reintentas. Es opcional pero **muy recomendado** en POSTs.

Respuesta exitosa

```
{
  "id": "cobr_KFt7ZiohxTjaBSTS",
  "estado": "exitoso",
  "metodo": "c2p_bnc",
  "monto": 25.50,
  "moneda": "VES",
  "proveedor": {
    "nombre": "bnc",
    "id_transaccion": 442350,
    "referencia": "036626"
  },
  "creado_en": "2026-06-03T15:58:13.103Z"
}
```

Respuesta rechazada

```
{
  "id": "cobr_xxx",
  "estado": "rechazado",
  "error": "fondos_insuficientes",
  "codigo": "G51",
  "codigo_completo": "ECBG51",
  "mensaje": "El cliente no tiene fondos suficientes"
}
```

Códigos de error frecuentes en un cobro:

- `token_incorrecto` — Token mal o expirado. El token del cliente expira a las 11:59 PM del día que lo generó.
- `fondos_insuficientes` — El cliente no tiene saldo.
- `no_afiliado` — El cliente no está afiliado a Pago Móvil.
- `cedula_invalida` — Cédula con formato o dígito errado.
- `limite_excedido` — El cliente excedió su monto diario.

3b. Cobrar con tarjeta internacional — `stripe_card` (15 min)

Para clientes con tarjeta internacional (Visa/Mastercard, monedas `USD, EUR, GBP, BRL, MXN, COP`). El flujo es distinto al C2P: **no es débito inmediato**, sino un checkout alojado — creas el cobro, rediriges a tu cliente a `checkout_url`, y el resultado te llega por webhook.

```
curl -X POST "$BASE/v1/cobros" \
  -H "Authorization: Bearer $KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "metodo": "stripe_card",
    "monto": 25.00,
    "moneda": "USD",
    "cliente": { "email": "cliente@ejemplo.com" },
    "referencia_externa": "RESERVA-002",
    "concepto": "Cancha 2 - 19:00"
  }'
```

Respuesta (`201`):

```
{
  "id": "cobr_xxx",
  "estado": "pendiente_pago",
  "metodo": "stripe_card",
  "monto": 25.00,
  "moneda": "USD",
  "checkout_url": "https://checkout.stripe.com/c/pay/cs_test_...",
  "proveedor": { "nombre": "stripe", "session_id": "cs_test_...", "expira_en": 1780000000 },
  "creado_en": "..."
}
```

Qué haces tú: redirige a tu cliente a `checkout_url` (o ábrelo en una ventana). Cuando pague, Stripe confirma y el estado del cobro pasa a `exitoso` — te enteras por webhook o consultando `GET /v1/cobros/{id}` (el GET también devuelve el `checkout_url` mientras el cobro sigue `pendiente_pago`, por si necesitas reenviárselo al cliente).

¿A dónde vuelve el cliente después de pagar? Por defecto, a la página de resultado de SolintPay (`/pay/{id}?status=success` o `?status=cancel`). Si prefieres que vuelva a TU app (lo normal en una app de reservas), pasa tus propias URLs `https` al crear el cobro:

```
{
  "metodo": "stripe_card",
  "monto": 25.00,
  "moneda": "USD",
  "success_url": "https://tu-app.com/reserva/123/pagada",
  "cancel_url":  "https://tu-app.com/reserva/123/cancelada"
}
```

> - `cliente` es opcional aquí (solo `email`, para el recibo de Stripe).
> - El cobro usa **la cuenta Stripe del comercio** (configuras tu Secret Key en el panel). Si no está configurada: `412 stripe_no_configurado`.
> - Stripe no liquida a Venezuela: este método es para comercios con cuenta Stripe propia (entidad fuera de VE) que venden a clientes internacionales.

3c. Cobrar por Zelle o con confirmación manual (15 min)

Para pagos que **un humano del comercio verifica y confirma** (Zelle, o comprobante por WhatsApp/transferencia). El cobro nace en `estado: "pendiente_confirmacion"` y pasa a `exitoso` cuando el comercio lo confirma desde su panel.

```
curl -X POST "$BASE/v1/cobros" \
  -H "Authorization: Bearer $KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "metodo": "zelle",
    "monto": 30.00,
    "moneda": "USD",
    "instrucciones": {
      "zelle_email": "pagos@tucomercio.com",
      "titular": "Tu Comercio LLC",
      "nota": "Indica la referencia en el memo"
    },
    "referencia_externa": "RESERVA-003"
  }'
```

Respuesta (`201`): `estado: "pendiente_confirmacion"` + `checkout_url` — una página alojada que muestra a tu cliente **a dónde enviar el Zelle** y la referencia. La página se actualiza sola cuando el comercio confirma.

El ciclo completo:

1. Creas el cobro → muestras/rediriges al `checkout_url` (o muestras las instrucciones tú mismo).
2. Tu cliente envía el Zelle y avisa.
3. El dueño del comercio verifica en su banco → panel → **Transacciones** → botón **Confirmar** (o Rechazar).
4. El cobro pasa a `exitoso` y te llega el webhook `cobro.confirmado`.

> - `metodo: "manual"` funciona igual pero sin campos de Zelle: pasa la instrucción libre en `instrucciones.nota` (ej. "envía el comprobante por WhatsApp al 0414...").
> - Para `zelle` , `instrucciones.zelle_email` es **requerido** (400 si falta).

Binance Pay — próximamente

`metodo: "binance_pay"` ya existe en el API pero está deshabilitado hasta activar las credenciales: hoy responde `503 metodo_no_disponible` . Tu código puede manejarlo desde ya con ese error.

---

4. Mostrar la tasa BCV en tu UI (5 min)

Si vendes en USD pero cobras en bolívares, muestra el equivalente al BCV:

```
curl -H "Authorization: Bearer $KEY" "$BASE/v1/tasa-bcv"
```

```
{ "tasa": 36.6642, "moneda_base": "USD", "moneda_destino": "VES", "fecha": "07/08/2024" }
```

> En sandbox la tasa siempre es un valor fijo de prueba. En producción se actualiza con el BCV real cada día.

¿Cotizas en dólares y también usas tasa paralelo? `GET /v1/tasas` devuelve ambas (`bcv` en vivo + `paralelo` si SolintPay la tiene configurada). Y mejor aún: puedes **fijar el precio en Bs al momento de cobrar** sin calcular nada — en vez de `monto`, envía `monto_usd` + `tasa_tipo`:

```
{
  "metodo": "c2p_bnc",
  "monto_usd": 10,
  "tasa_tipo": "bcv",
  "cliente": { "cedula": "...", "telefono": "...", "banco": "...", "token": "..." }
}
```

La respuesta trae el `monto` en Bs calculado, más `monto_usd`, `tasa_tipo` y `tasa_aplicada` para tu contabilidad. Si la tasa paralelo no está configurada: `400 tasa_no_disponible`.

---

5. Registrar tu webhook (15 min)

Aquí está la magia: cada vez que entre un pago a tu cuenta, SolintPay te avisa con una notificación HTTP (webhook) firmada.

5.1 Levanta tu endpoint

En tu servidor, crea un endpoint `POST` que va a recibir los webhooks. Para probar al principio puedes usar webhook.site y ver lo que llega.

5.2 Registra tu URL en SolintPay

> ⚠️ **Requisitos de la URL:** debe ser `https://` y apuntar a un **dominio público**. Se rechazan `http://`, `localhost`, IPs privadas y hosts internos (medida anti-SSRF). Para probar en local usa un túnel https (ngrok, Cloudflare Tunnel) o webhook.site.

La forma fácil (recomendada): el **dueño del comercio** entra a su panel → sección **Webhooks**, registra la URL y copia el `whsec_…` que se muestra una sola vez. Si eres el programador contratado, pídele al dueño ese `whsec_…` — es lo único que necesitas de él para esta sección.

Por API con tu misma API key (lo más directo si estás programando):

```
curl -X POST "$BASE/v1/webhooks" \
  -H "Authorization: Bearer $KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://tu-servidor.com/webhooks/solintpay",
    "eventos": ["pago.recibido"]
  }'
```

Y cuando tengas tu endpoint listo, pruébalo con un evento firmado de verdad:

```
curl -X POST "$BASE/v1/webhooks/test" -H "Authorization: Bearer $KEY"
# → { "url": "...", "respuesta_status": 200, "respuesta_ok": true, "latencia_ms": 184 }
```

Alternativa desde el panel por API (mismo resultado), con las **credenciales humanas** del panel (correo y contraseña del dueño):

```
# 1. Login con tus credenciales humanas
LOGIN=$(curl -s -X POST "$BASE/admin/login" \
  -H "Content-Type: application/json" \
  -d '{"email":"TU_EMAIL","password":"TU_PASSWORD"}')

# 2. Saca el access_token
ACCESS=$(echo $LOGIN | jq -r .tokens.access_token)

# 3. Registra el webhook
curl -X PUT "$BASE/admin/webhook" \
  -H "Authorization: Bearer $ACCESS" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://tu-servidor.com/webhooks/solintpay",
    "eventos": ["pago.recibido"]
  }'
```

Respuesta:

```
{
  "id": "...",
  "url": "https://tu-servidor.com/webhooks/solintpay",
  "secret": "whsec_xxxxxxxxxxx",
  "creado_en": "..."
}
```

> ⚠️ **Guarda ese `secret` (`whsec_…`) en una variable de entorno (`SOLINTPAY_WEBHOOK_SECRET`). No se muestra de nuevo.**

5.3 Verifica la firma HMAC del lado tuyo

Cuando recibes un webhook, **DEBES verificar la firma** para confirmar que viene de SolintPay y no de un impostor. Ejemplo en Node.js:

```
import crypto from 'crypto';
import express from 'express';

const app = express();

// IMPORTANTE: necesitamos el body crudo para verificar la firma.
app.use('/webhooks/solintpay', express.raw({ type: 'application/json' }));

app.post('/webhooks/solintpay', async (req, res) => {
  const secret = process.env.SOLINTPAY_WEBHOOK_SECRET;
  const signatureHeader = req.headers['x-solintpay-signature']; // "sha256=<hex>"
  const rawBody = req.body; // Buffer

  const expected = 'sha256=' + crypto
    .createHmac('sha256', secret)
    .update(rawBody)
    .digest('hex');

  // Comparación segura contra ataques de tiempo
  if (signatureHeader.length !== expected.length ||
      !crypto.timingSafeEqual(Buffer.from(signatureHeader), Buffer.from(expected))) {
    return res.status(401).json({ error: 'firma_invalida' });
  }

  // ✓ Webhook legítimo
  const event = JSON.parse(rawBody.toString());
  const deliveryId = req.headers['x-solintpay-delivery-id'];

  // IDEMPOTENCIA: guarda el deliveryId; si ya lo procesaste, no lo proceses de nuevo
  if (await yaLoProcesamos(deliveryId)) {
    return res.json({ ok: true, replay: true });
  }

  // Procesa el evento
  if (event.evento === 'pago.recibido') {
    const { monto, moneda, referencia_bnc, cliente, fecha } = event.datos;
    // Marca la reserva como pagada, envía confirmación al cliente, etc.
  }

  await guardarComoProcesado(deliveryId);

  res.json({ ok: true });
});
```

5.4 Headers que trae cada webhook

Header	Para qué
`X-SolintPay-Event`	Nombre del evento (`pago.recibido` , etc.)
`X-SolintPay-Signature`	Firma HMAC, formato `sha256=<hex>`
`X-SolintPay-Delivery-Id`	UUID del envío — úsalo para idempotencia
`X-SolintPay-Attempt`	Número de intento (1 a 7)

5.5 Payload del webhook

```
{
  "id": "delivery_uuid",
  "evento": "pago.recibido",
  "creado_en": "2026-06-03T15:58:16Z",
  "intento": 1,
  "datos": {
    "tipo": "P2P",
    "monto": 1500.00,
    "moneda": "0928",
    "referencia_bnc": "DBR-xxx",
    "referencia_banco_origen": "OBR-xxx",
    "banco_origen": "0102",
    "fecha": "20260603",
    "hora": "1430",
    "cliente": {
      "cedula": "V12345678",
      "telefono": "00584241234567",
      "concepto": "Reserva cancha"
    }
  }
}
```

> Nota: en el payload del webhook, `moneda` llega como el **código ISO 4217 numérico** que usa el banco (`0928` = bolívar VES), no como el string `"VES"` .

5.6 Reintentos

Si no respondes `2xx` en 10 segundos, SolintPay reintenta con backoff: **30s, 2 min, 10 min, 1h, 6h, 24h** — hasta 7 intentos. Después abandona.

Por eso es crítico responder rápido y procesar **después** (en background).

---

6. Listar y consultar tus cobros (10 min)

```
# Listar (filtros opcionales)
curl -H "Authorization: Bearer $KEY" \
  "$BASE/v1/cobros?limit=20&estado=exitoso"

# Detalle de uno
curl -H "Authorization: Bearer $KEY" \
  "$BASE/v1/cobros/cobr_XYZ"
```

6b. Comprobante electrónico del cobro

Todo cobro **exitoso** tiene un comprobante con los datos que exige la normativa bancaria (afiliado, fecha/hora, autorización del banco, monto, referencia, teléfonos y cédula **enmascarados**). Puedes mostrarlo al cliente o imprimirlo en tu propio recibo:

```
# Página lista para mostrar/imprimir (pública, sin API key: solo quien tiene el id)
https://solintpay.vercel.app/pay/cobr_XYZ/comprobante

# Los mismos datos en JSON, para armar tu recibo
curl "$BASE/pay/cobr_XYZ/comprobante?formato=json"
```

Responde `409 sin_comprobante` si el cobro no está en estado `exitoso`.

---

7. Reversar un cobro (10 min)

El mismo endpoint reversa según el método: **reverso C2P** en el banco para `c2p_bnc`, o **refund total en Stripe** para `stripe_card`. Si el cliente reclama o hubo un error:

```
curl -X POST "$BASE/v1/cobros/cobr_XYZ/reversar" \
  -H "Authorization: Bearer $KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{ "motivo": "Cliente reclamó" }'
```

8. Referencia completa de endpoints

Todas las llamadas requieren el header `Authorization: Bearer <api_key>`. Base URL: la que te confirme SolintPay. Todos los `POST` aceptan el header opcional `Idempotency-Key` (UUID v4 o string único de 8-64 caracteres).

Catálogo

`GET /v1/ping`

Verifica conectividad y validez de tu API key. Respuesta: `{ ok, cliente, env, timestamp }`.

`GET /v1/bancos`

Lista de bancos venezolanos con sus códigos oficiales del BCV (`0102` = Banco de Venezuela, `0191` = BNC, etc.). Respuesta: `{ data: [ { codigo, nombre, servicios } ] }`.

`GET /v1/tasa-bcv`

Tasa BCV oficial del día. Respuesta: `{ tasa, moneda_base, moneda_destino, fecha }`.

`GET /v1/tasas`

BCV (en vivo) + paralelo (si SolintPay la configuró). Respuesta: `{ bcv: { tasa, fecha }, paralelo: { tasa, actualizada_en } | null }`.

Cobros

`POST /v1/cobros`

Primitiva única de cobro. `metodo` decide el rail: `c2p_bnc` (Pago Móvil, débito inmediato — requiere `cliente` completo; acepta `monto_usd` + `tasa_tipo` para fijar el precio en Bs), `vpos_bnc` (tarjeta nacional de débito/crédito por punto de venta — requiere objeto `tarjeta`), `stripe_card` (tarjeta internacional — devuelve `checkout_url` , sección 3b), `zelle` / `manual` (confirmación del comercio — sección 3c) o `binance_pay` (próximamente, 503). El ejemplo siguiente es C2P: el cliente te dicta sus datos y el token de su app.

Body:

```
{
  "metodo": "c2p_bnc",
  "monto": 25.50,
  "moneda": "VES",
  "cliente": {
    "cedula": "V12345678",
    "telefono": "584141234567",
    "banco": "0102",
    "token": "1234567"
  },
  "referencia_externa": "RESERVA-001",
  "concepto": "Cancha 1 - 18:00"
}
```

Campos requeridos: `metodo`, `monto`, `cliente` (con `cedula`, `telefono`, `banco`, `token`). Respuesta `201`: objeto `Cobro`. Rechazo del banco: `422` con `{ error, codigo, codigo_completo, mensaje }`.

`GET /v1/cobros`

Lista tus cobros. Query: `limit` (default 20, máx 100), `estado` (`iniciado` | `exitoso` | `rechazado` | `error`).

`GET /v1/cobros/{id}`

Detalle de un cobro. `404` si no existe o no es tuyo.

`POST /v1/cobros/{id}/reversar`

Reversa un cobro **exitoso**. Body opcional: `{ "motivo": "..." }`. `400` si el cobro no es reversable.

Pagos

`POST /v1/pagos`

Tu comercio paga a un beneficiario (P2P). Útil para devoluciones, cashback o payouts.

Body:

```
{
  "metodo": "p2p_bnc",
  "monto": 50.00,
  "moneda": "VES",
  "beneficiario": {
    "cedula": "V12345678",
    "telefono": "584141234567",
    "banco": "0102",
    "nombre": "Juan Pérez",
    "email": "juan@ejemplo.com"
  },
  "concepto": "Devolución reserva",
  "referencia_externa": "DEV-001"
}
```

Requeridos: `metodo`, `monto`, `beneficiario` (con `cedula`, `telefono`, `banco`, `nombre`).

Cuenta

`GET /v1/cuenta/saldo`

Saldo de tus cuentas bancarias. Respuesta: `{ data: [ { numero, moneda, saldo } ] }`.

`GET /v1/cuenta/movimientos`

Movimientos de una cuenta. Query: `cuenta` (requerido), `desde` y `hasta` (fechas opcionales, formato `YYYY-MM-DD`).

Verificación

`POST /v1/pagos/verificar`

Verifica si un pago realmente entró (cuando el cliente dice "ya te pagué pero no me llega notificación").

Body:

```
{
  "tipo": "p2p",
  "cuenta_propia": "0191...",
  "monto": 25.50,
  "referencia": "036626",
  "fecha": "2026-06-03",
  "telefono_pagador": "584141234567",
  "banco_pagador": "0102"
}
```

Requeridos: `tipo` (`p2p` | `referencia`), `cuenta_propia` , `monto` , `referencia` , `fecha` . Respuesta: `{ encontrado: boolean, detalle: object | null }` .

📱 Integración desde una app móvil (iOS / Android / React Native / Flutter)

La regla de oro: **la app NUNCA habla directo con SolintPay.** Tu API key (`sk_test_` / `sk_live_`) solo puede vivir en **tu backend** — cualquier string embebido en un APK/IPA se puede extraer descompilando la app.

```
App móvil  ──▶  TU backend  ──▶  SolintPay  ──▶  Banco
   ▲                │
   └── respuesta ◀──┘   (y los webhooks de SolintPay llegan a TU backend)
```

El flujo correcto para un cobro C2P desde la app:

1. La app muestra tu formulario de pago: cédula, teléfono, banco (usa `GET /v1/bancos` **vía tu backend** para el selector) y el **token** que el cliente genera en la app de su banco.
2. La app manda esos 4 datos a **tu backend** (tu propio endpoint, con tu propia autenticación de usuarios).
3. Tu backend llama `POST /v1/cobros` con tu API key y un `Idempotency-Key` .
4. Tu backend devuelve el resultado a la app (`exitoso` / `rechazado` + mensaje amigable del campo `mensaje`).
5. Para métodos con checkout (`stripe_card` , `zelle`): tu backend recibe el `checkout_url` y la app lo abre en un **navegador in-app** (SFSafariViewController / Chrome Custom Tabs / `Linking.openURL`). La confirmación te llega por **webhook a tu backend**, no a la app — la app se entera consultando tu backend (polling o push propio).

Qué NO hacer en la app:

- ❌ Incluir la API key en el código, en un `.env` bundleado o en remote config.
- ❌ Llamar `https://solintpay.vercel.app/v1/...` directo desde la app.
- ❌ Confiar en el resultado que reporta la app: el estado real del cobro es el que ve tu backend (webhook o `GET /v1/cobros/{id}`).

Tip UX móvil: el token C2P lo genera el cliente en la app de su banco y expira el mismo día — pon el campo token de último en el formulario y permite reintentar con `token_incorrecto` sin perder los demás datos.

🔒 Seguridad

Qué es secreto y qué no, para que no expongas nada sin querer:

Dato	¿Secreto?	Regla
API key (`sk_test_` / `sk_live_`)	**SÍ**	Solo en el backend, en variables de entorno. Nunca en el frontend, el repo ni logs.
Webhook secret (`whsec_…`)	**SÍ**	Solo en el backend. Es lo que te permite confiar en los webhooks.
Contraseña del panel admin	**SÍ**	Nunca la pongas en código.
URL base del API	No	Es pública, la necesita tu servidor.
IDs de cobro (`cobr_…`), referencias	No	Son identificadores, no credenciales.

Buenas prácticas:

- **Verifica SIEMPRE la firma HMAC** de cada webhook (sección 5.3). Sin eso, cualquiera podría enviarte pagos falsos.
- **Sé idempotente** con el `X-SolintPay-Delivery-Id` para no procesar dos veces el mismo pago.
- Si crees que tu API key se filtró, avísanos para **rotarla** de inmediato.

🧪 Apéndice: Modo de prueba (sandbox)

Mientras integras, usa una API key `sk_test_…`. En sandbox **ninguna operación mueve dinero real** — todo es una simulación contra el ambiente de calidad del banco.

Para hacer un **cobro de prueba** necesitas datos de un cliente. El banco tiene una identidad de prueba registrada en su sandbox; usa estos valores:

```
{
  "cedula":   "V23000760",
  "telefono": "584242207524",
  "banco":    "0191",
  "token":    "9982783"
}
```

> ⚠️ **Estos son datos de prueba del sandbox del banco — NO son de un cliente real y solo funcionan en el ambiente de prueba.** En producción, estos cuatro campos los recibes de tu cliente real en el momento de cobrar (el token lo genera él en la app de su banco).
>
> **Cómo trata el sandbox tus cobros:** el comercio en **modo sandbox** hace que el banco **valide** cada operación (formato, cripto, datos) pero **no la ejecute**. Por eso el token fixture de arriba funciona y el cobro sale `exitoso` sin mover dinero. Si en sandbox te devuelve `token_incorrecto` con los fixtures, casi siempre es porque tu comercio

> quedó en **modo producción** (ahí el banco sí ejecuta y exige un token real del cliente) — pídele a SolintPay que lo devuelva a sandbox (consola de operador → el comercio → *Volver a sandbox*).
>
> En sandbox el banco a veces **no emite** `proveedor.referencia` (puede venir `null`) ni `id_transaccion` reversable, aunque el cobro salga `exitoso`. En producción sí vienen siempre — no dependas de ellos para tus pruebas de sandbox.

🚀 Paso a producción

Concepto clave: el ambiente lo decide el **modo del comercio**, no tu código.

- **Sandbox** → el banco **valida** los cobros pero **no los ejecuta**. Usas los fixtures, nada mueve dinero. Es donde integras y pruebas.
- **Producción** → los cobros se **ejecutan** de verdad: el cliente pone su token real (o su tarjeta) y el dinero cae en la cuenta del comercio.

Pasar de uno a otro es eficiente y no requiere reescribir la integración. Hay dos lados:

Del lado de SolintPay (consola de operador → ficha del comercio):

1. Aprobar el **KYC** del comercio.
2. En **Configuración de cobro**, cargar el **ChildClientID** del BNC del comercio (así el dinero cae en SU cuenta, no en la de SolintPay). Sin él, no deja activar producción.
3. **Activar producción**. Se puede **Volver a sandbox** en cualquier momento para seguir probando — el mismo comercio, sin perder su configuración ni sus keys.

Del lado del integrador (tú) — tu código NO cambia:

1. Cambia la API key `sk_test_…` → `sk_live_…` en `SOLINTPAY_API_KEY`. **La key decide el ambiente**; el resto del código es idéntico.
2. Usa **datos reales del cliente** (el token lo genera él en la app de su banco), en vez de los fixtures del sandbox.
3. **Registra tu webhook de producción** apuntando a tu URL real y guarda el nuevo `whsec_…`.
4. A partir de ahí, los cobros **mueven dinero real**.

📦 SDKs oficiales

Si prefieres no escribir el HTTP a mano, hay SDKs con la misma superficie (crear/consultar/listar/reversar + verificación HMAC de webhooks incluida):

Lenguaje	Dónde	Nota
Node.js	`sdk/` del repo (npm `solintpay`)	Con tipos TypeScript, sin dependencias (Node ≥18)
Python	`sdk/python/solintpay.py`	Un solo archivo, stdlib pura (≥3.8)
PHP	`sdk/php/SolintPay.php`	Un solo archivo, cURL (≥7.4)

```
const { SolintPay } = require('solintpay');
const sp = new SolintPay({ apiKey: process.env.SOLINTPAY_API_KEY });
const cobro = await sp.crearCobro({ metodo: 'c2p_bnc', monto: 25.5, moneda: 'VES', cliente: { ... } });
```

🐞 Troubleshooting

Problema	Causa más común
`401 unauthorized`	Olvidaste el header `Authorization`
`401 invalid_api_key`	API key inválida o revocada
`403 insufficient_scope`	Tu API key no tiene el scope `cobros:create`
`422 token_incorrecto`	Token del cliente errado o expirado (renovar)
`422 fondos_insuficientes`	El cliente no tiene saldo
`409 idempotency_conflict`	Reusaste un `Idempotency-Key` con un body distinto
`5xx`	Error de nuestro lado — avísanos

📞 Soporte

- Email: soporte@solintpay.com
- Status: https://status.solintpay.com (próximamente)