MÉXICO

Neat México · Documentación para comercios

Todo lo que un comercio necesita para aceptar pagos con Neat México, desde el primer request hasta la salida a producción.

Descargar guía de integración (PDF) Brochure comercial (PDF)

Tu servidor crea una sesión de checkout, redirige al pagador a una página hospedada por Neat y recibe el resultado por webhook o consultando el estado de la sesión. No necesitas capturar tarjetas ni manejar datos de pago en tu propio sitio.

Tu primera sesión de checkout

Sigue el quickstart para crear tu primera sesión de checkout de prueba y confirmar su resultado, en cURL y en Node.

Antes de cobrar dinero real

Revisa el checklist de salida a producción: qué probar en sandbox, cómo se valida tu webhook y qué confirma Neat antes de habilitar cobros en vivo.

Cómo está organizada esta guía

Sección	Contenido
Empezar	Quickstart, alta de tu comercio, autenticación y ambientes.

Sección	Contenido
Cobros	Sesiones de checkout, redirección y regreso, estados y glosario, importes y comisiones.
Notificaciones	Eventos y firma de webhooks, reintentos e idempotencia, verificar la firma en Node, Python y PHP.
Conciliación	Formato del archivo CSV de pagos.
Referencia	Catálogo de errores, referencia de cada endpoint, especificación OpenAPI descargable.
Salida a producción	Checklist a completar antes de aceptar cobros reales.

Los identificadores y las fechas de los ejemplos son ilustrativos. Los nombres de campo, los tipos y los códigos de error son exactos, tal como los devuelve la API.

MÉXICO

Quickstart

Al terminar esta página habrás creado una sesión de checkout de prueba, redirigido a un pagador de prueba al checkout hospedado por Neat, y confirmado el resultado del cobro consultando la sesión.

Antes de empezar

Necesitas las credenciales de prueba que Neat te entrega al dar de alta tu comercio (ver Alta de tu comercio): una API key de pruebas y al menos un `return_id` registrado con la URL a la que Neat debe devolver al pagador. Guarda ambos valores como variables de entorno; nunca los escribas directamente en tu código.

```
export NEAT_API_BASE="https://api-beta.neatpagos.mx/v1"
export NEAT_API_KEY="<tu API key de pruebas>"
```

La API key de pruebas empieza con `nmx_test_` y mide 52 caracteres. El `webhook_secret`, que empieza con `whsec_`, no va en esta variable ni en ninguna llamada a la API: sólo sirve para verificar los webhooks. Si la variable llega vacía a tu código, la API responde `401 UNAUTHORIZED`; ver Si recibes un 401.

`NEAT_API_BASE` es el ambiente de pruebas vigente. Neat te confirmará el dominio de producción al habilitar tu comercio para cobros en vivo, ver Autenticación y ambientes.

> ⚠ **Si cobras en nombre de varios condominios**
>
> Si Neat habilitó tu comercio para cobrar por beneficiario, **cada creación de sesión tiene que incluir `beneficiary`**; sin él, la API responde 422 `BENEFICIARY_REQUIRED`. Usa los ejemplos de Si cobras por varios beneficiarios. Si no sabes en qué modo está tu comercio, pregúntale a tu contacto en Neat.

1. Crear una sesión de checkout

Envía el monto en centavos (`amount_minor`), tu propia referencia de la obligación (`external_reference`) y el `return_id` registrado. La cabecera `Idempotency-Key` es obligatoria: identifica esta solicitud de creación, no el cobro en sí, y te permite reintentar sin abrir dos sesiones si la respuesta original se perdió.

Genera una llave nueva para cada intento de cobro y guárdala junto a ese intento antes de llamar. Si la llamada se corta, repítela con la misma llave y el mismo cuerpo; si es un cobro nuevo, usa otra llave. Una llave fija sólo sirve una vez: con otro cuerpo responde `409 IDEMPOTENCY_CONFLICT`, y con el mismo devuelve la sesión que ya creaste, aunque haya vencido.

> ⚠ **El monto va en centavos**
>
> `amount_minor` es un número entero de centavos. Para cobrar MXN 12,850.00 envía `1285000`; si envías `12850`, el cobro queda en MXN 128.50. El mínimo es `2000` (MXN 20.00), salvo que tu comercio tenga otro, y el máximo es el límite acordado para tu comercio. Al crear tu primera sesión de prueba, abre su `checkout_url` y confirma que el monto que ve el pagador es el que quieres cobrar.

cURL | Node | PHP

Crear una sesión de checkout

```
IDEMPOTENCY_KEY="checkout-$(uuidgen)"

curl -X POST "$NEAT_API_BASE/checkout-sessions" \
  -H "Authorization: Bearer $NEAT_API_KEY" \
  -H "Idempotency-Key: $IDEMPOTENCY_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "external_reference": "CUOTA-2026-09-000123",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "amount_minor": 125000,
    "currency": "MXN",
    "return_id": "principal"
  }'
```

En los ejemplos, la llave vive sólo mientras corre el código. En tu integración, guárdala con el intento de cobro antes de llamar, por ejemplo en la fila del intento en tu base de datos, y reutilízala con el mismo cuerpo si la llamada se corta: una llave generada de nuevo abre otra sesión.

⚠ `payer.email` decide quién puede pagar la sesión

Estos ejemplos no mandan `payer`. Si lo agregas, esto es lo que cambia:

- **Si mandas `payer.email`**, el checkout no le pregunta el correo a quien llega: le muestra esa dirección enmascarada y le envía el código de acceso sólo a ella. Nadie más puede pagar la sesión. Otra cuenta, incluida una de Google o Apple con otro correo, recibe `403 PAYER_MISMATCH` y ve «Este pago es para otra cuenta», con el correo con que entró y la dirección del cobro enmascarada. Desde ahí sólo puede entrar con esa dirección.
- **Si no lo mandas**, quien llega escribe su correo en el checkout y recibe el código ahí. La sesión queda para el primero que entre con la liga.
- **En pruebas**, manda el correo de tu equipo o no mandes `payer`. Con un correo de ejemplo, la sesión no la puede pagar nadie de tu equipo.
- **Si la sesión salió con el correo equivocado**, no se corrige: crea otra con el correo correcto, o sin `payer`, y una `Idempotency-Key` nueva. Puedes repetir el mismo `external_reference`, porque el correo no es un término de la obligación. La sesión anterior sigue viva hasta su `expires_at`.

⚠ `external_reference` fija los términos del cobro

`external_reference` identifica tu obligación de cobro, y la obligación queda con el `amount_minor`, la `currency`, la `category`, el `concept` y el `beneficiary` de la primera sesión que la creó:

- **Misma referencia y mismos términos, sin pago:** la API acepta y abre otra sesión sobre la misma obligación. La anterior sigue viva hasta su `expires_at`.
- **Misma referencia con cualquier término distinto: `409 OBLIGATION_CONFLICT`, aunque no se haya pagado.** Un `concept` con otro texto cuenta como distinto. Es a propósito: todas las ligas de una referencia cobran la misma obligación (solo la comisión puede variar con `fee_bearer`), y la anterior puede seguir abierta en manos del pagador; una obligación no se reapunta a otra CLABE; y tus webhooks y tu archivo de conciliación se cruzan por esta referencia.
- **Si la cuota cambió antes de pagarse** (un recargo por mora, una corrección de monto o de concepto), usa una referencia nueva con una versión, por ejemplo `CUOTA-2026-09-000123-R1`. Antes, espera a que pase el `expires_at` de la sesión más reciente de la referencia anterior y confirma con `GET /v1/checkout-sessions/{sessionId}`, sesión por sesión, que ninguna de las que creaste para esa referencia tiene un pago `succeeded` ni `pending`. Una sesión no se puede cerrar, y que el pagador ya no tenga la liga no basta: si se pagaran las dos referencias, serían dos cobros.
- **Si lo que está mal es la CLABE** y todavía no hay pago, escríbenos con el `external_reference`; no la corrijas con una referencia nueva.

La referencia `CUOTA-2026-09-000123` de los ejemplos es ilustrativa: no la reutilices con datos reales. Ver `external_reference`.

Si cobras por varios beneficiarios

Si tu comercio cobra en nombre de varios condominios, agrega `beneficiary`: quién recibe el dinero de este cobro. La `clabe` va siempre como texto, entre

comillas. El detalle de cada campo está en Beneficiario por sesión.

cURL | Node | PHP

Crear una sesión con beneficiario

```
IDEMPOTENCY_KEY="checkout-$(uuidgen)"

curl -X POST "$NEAT_API_BASE/checkout-sessions" \
  -H "Authorization: Bearer $NEAT_API_KEY" \
  -H "Idempotency-Key: $IDEMPOTENCY_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "external_reference": "CONDO-00123-2026-09-402",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "amount_minor": 125000,
    "currency": "MXN",
    "return_id": "principal",
    "beneficiary": {
      "reference": "CONDO-00123",
      "legal_name": "CONDOMINIO EJEMPLO LAS PALMAS AC",
      "rfc": "CEP010101AB1",
      "clabe": "032180000118359719"
    }
  }'
```

Datos de ejemplo

El condominio, el RFC y la CLABE de estos ejemplos son ficticios: reemplázalos por los del condominio de cada cobro. Neat deposita a la CLABE que declara la sesión. Si la CLABE sale de una columna numérica de tu base de datos, conviértela a texto de 18 dígitos antes de armar el cuerpo: como número pierde los ceros iniciales y la API responde 422.

Respuesta (`201 Created`):

201 Created

```
{
  "session_id": "cs_164596add1a3",
  "checkout_url":
"https://pago.neat.example/pagar/cs_164596add1a3.CQGg0fEWT5o…",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "status": "open",
  "fee_bearer": "payer"
}
```

`checkout_url` en este ejemplo usa un dominio de referencia: en el ambiente de pruebas real apunta al dominio de checkout que Neat te confirma al dar de alta tu comercio. La sesión expira 30 minutos después de creada; guarda `session_id` y tu `external_reference` para consultarla después.

2. Redirigir al pagador

Responde a la petición que originó el cobro (por ejemplo, el clic de tu usuario en "Pagar") con una redirección HTTP a `checkout_url`. Neat captura el medio de pago; tu servidor no participa en esa parte del flujo.

3. Confirmar el resultado

Cuando el pagador termina, Neat lo redirige de vuelta a tu `return_url`, ver Redirección y regreso sobre por qué esa redirección **no** confirma el pago por sí sola. La fuente de verdad es el webhook (ver Webhooks: eventos y firma) o consultar la sesión directamente:

cURL | Node

Consultar el estado de la sesión

```
curl "$NEAT_API_BASE/checkout-sessions/cs_164596add1a3" \
-H "Authorization: Bearer $NEAT_API_KEY"
```

200 OK — pago confirmado


```
{
  "session_id": "cs_164596add1a3",
  "status": "completed",
  "external_reference": "CUOTA-2026-09-000123",
  "category": "maintenance_fee",
  "concept": "Mensualidad de mantenimiento, septiembre 2026",
  "amount_minor": 125000,
  "currency": "MXN",
  "fee_bearer": "payer",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "payment": {
    "payment_id": "pay_3ac4efd9f068",
    "version": 2,
    "status": "succeeded",
    "currency": "MXN",
    "principal_amount_minor": 125000,
    "payer_fee_minor": 3750,
    "payer_fee_tax_minor": 600,
    "provider_fee_minor": 0,
    "provider_fee_tax_minor": 0,
    "charged_amount_minor": 129350,
    "provider_net_amount_minor": 125000,
    "refunded_amount_minor": 0,
    "fee_bearer": "payer",
    "cancellation_state": "none",
    "paid_at": "2026-09-14T23:17:17.894Z",
    "category": "maintenance_fee",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "payee_name": "Administradora Ejemplo, S.A. de C.V."
  }
}
```


`payee_name` es el nombre de tu comercio tal como quedó registrado en el alta o, si la sesión trae `beneficiary`, su `legal_name`.

El ejemplo no manda `category` porque el comercio tiene habilitada una sola, la cuota de mantenimiento, y la respuesta la devuelve igual. Si tu comercio

cobra varias (por ejemplo, renta y póliza de renta), mándala en cada sesión; ver Categoría del cobro.

Siguientes pasos

- Sesiones de checkout: todos los campos, límites y errores de creación.
- Importes y comisiones: quién paga la comisión y cómo se calcula.
- Webhooks: eventos y firma: cómo recibir el resultado sin tener que hacer polling.

MÉXICO

Alta de tu comercio

Al terminar el alta tendrás credenciales de prueba, un `return_id` registrado y acceso al ambiente de pruebas para hacer tu primera integración.

El alta no es un formulario en línea: nos escribes, te respondemos con tus credenciales de prueba, y producción es una segunda conversación después de que tu integración pase el checklist. Lo hacemos así a propósito — el `webhook_secret` firma el aviso con el que marcas una deuda como pagada, y eso no se entrega por un formulario.

Qué entregas a Neat

Dato	Para qué se usa
Razón social y RFC	Identifica legalmente a tu comercio.
Uno o más dominios fijos de retorno	Cada uno queda registrado como un `return_id`; Neat redirige al pagador ahí al terminar el checkout.
URL de webhook (HTTPS)	Dirección a la que Neat envía los eventos de pago. Ver Webhooks: eventos y firma.
Contacto técnico	A quién avisa Neat ante un incidente o una rotación de credenciales, y a quién le llegan las credenciales.

Correo para pedir el alta

Cópialo, complétalo y mándalo a tu contacto en Neat o a `info@neatpagos.com`. Si algún dato todavía no lo tienes, déjalo escrito como pendiente en vez de borrarlo: así sabemos qué falta.

```
Asunto: Alta de comercio en Neat México — <razón social>

Razón social:
RFC (12 caracteres si es persona moral, 13 si es persona física):
Nombre comercial (el que verá quien paga):

Dominios de retorno (uno por línea, HTTPS, sin parámetros):
  https://
Para cada uno, en qué momento del flujo lo usas:

URL de webhook (HTTPS):
¿Tu webhook exige IPs en lista blanca? (sí / no)

Contacto técnico (recibe las credenciales)
  Nombre:
  Correo:
  Teléfono para incidentes (con lada internacional, por ejemplo +52 55
1234 5678):

Correo del equipo que va a probar la integración:
  (a esa dirección le llegan los códigos de acceso del ambiente de
pruebas)

Monto típico de un cobro y monto máximo esperado:
Volumen mensual estimado de cobros:
```

Qué significa cada cosa

- **Dominios de retorno.** A dónde mandamos de vuelta a quien paga cuando termina. Cada uno queda registrado como un `return_id` y cada sesión elige uno. Un dominio no registrado hace fallar la creación de la sesión con `422 RETURN_NOT_REGISTERED`.
- **URL de webhook.** La dirección de tu servidor que recibe los avisos de pago. Tiene que ser HTTPS y estar accesible desde internet. Si está detrás de una lista blanca de IPs, dilo: hoy no publicamos un rango fijo y hay que resolverlo antes.
- **Si tu sitio está detrás de Cloudflare u otro WAF.** Exime la ruta exacta del webhook de las reglas que bloquean o desafían tráfico automatizado.

Un bloqueo o un desafío le contesta a Neat en lugar de tu servidor, y **cualquier `3xx` o `4xx` deja la entrega como fallida, sin reintento** (ver Reintentos e idempotencia). En Cloudflare, crea una regla personalizada para esa ruta con la acción *Skip* sobre las protecciones que tengas activas; Bot Fight Mode, el del plan gratuito, no se puede eximir por ruta, así que tendrás que apagarlo o usar Super Bot Fight Mode, que sí admite la regla.

- **Correo del equipo que prueba.** En el ambiente de pruebas, quien paga entra con un código de seis dígitos que enviamos por correo. Usa una dirección de ustedes: así prueban exactamente lo que va a vivir su residente, código incluido.
- **Monto típico y máximo.** Con eso fijamos tu límite por cobro. Un cobro por encima del límite falla con `422 VALIDATION_ERROR`, así que conviene decirlo ahora y no descubrirlo en producción.

Qué entrega Neat

Dato	Para qué se usa
`provider_id`	Identifica a tu comercio en las respuestas de la API.
API key de pruebas y API key de producción	Autentican cada request; ver Autenticación y ambientes.
`webhook_secret`	Firma HMAC de cada notificación; ver Verificar la firma.
Acceso al ambiente de pruebas	Para integrar y probar antes de mover dinero real.

Cómo se entregan las credenciales

La API key y el `webhook_secret` llegan al correo de tu contacto técnico en un mensaje aparte del resto del alta, enviado en modo confidencial: caduca a

los siete días y no se puede reenviar, copiar ni descargar. Si el mensaje caducó antes de que lo guardaras, pídenos otro; no se reenvía el mismo.

Son credenciales independientes y se rotan por separado. Guarda ambas en el secret manager de tu infraestructura apenas las recibas; no las incluyas en HTML, JavaScript de navegador, enlaces, tickets ni control de versiones.

Una API key identifica únicamente a tu servidor: la petición de creación de una sesión no permite elegir otro `provider_id` ni una cuenta bancaria de destino distinta a la acordada con Neat. La excepción son los comercios que cobran en nombre de varios condominios: cada sesión nombra al beneficiario y su CLABE, y Neat deposita a esa cuenta (ver Beneficiario por sesión).

Rotar una credencial

La rotación es bajo demanda: la pides cuando lo necesites, no sigue un calendario fijo. Las dos credenciales no rotan igual:

- **API key.** Neat emite una llave nueva y la anterior sigue funcionando hasta que Neat la revoca, en el momento que acordemos contigo. Así actualizas tu configuración sin una ventana de corte.
- **`webhook_secret`.** Rota de inmediato, sin período de gracia: desde ese momento cada evento se firma sólo con el secreto nuevo. Coordina el momento con tu contacto en Neat y ten lista la actualización de tu lado, ver Rotación del secreto.

`return_id`: una o varias páginas de regreso

Puedes registrar más de un `return_id` si tu comercio necesita distintas páginas de regreso para distintos flujos. Cada sesión de checkout elige uno al crearse. Un `return_id` que no esté registrado y activo hace que la creación de la sesión falle con `422 RETURN_NOT_REGISTERED`; ver Sesiones de checkout.

Ambientes

Pruebas y producción usan credenciales y URLs distintas. No hay un encabezado de ambiente: la API key que uses determina si el cobro es de prueba o real. Las URLs y el detalle están en Autenticación y ambientes.

Cómo pruebas, siendo tú quien paga

Para probar la integración tienes que hacer lo mismo que hará tu residente: abrir el `checkout_url`, entrar con un código, guardar una tarjeta y pagar.

Entras con **el correo de tu equipo**, el que nos diste en el alta. El código de seis dígitos llega ahí, igual que le va a llegar a tu residente. No hay un modo especial ni un código fijo: el ambiente de pruebas se recorre igual que el real, y esa es la idea — lo que pruebas es lo que van a vivir.

Si la sesión de prueba trae `payer.email`, **sólo ese correo puede pagarla**: el código se envía ahí y cualquier otra cuenta ve «Este pago es para otra cuenta». Crea tus sesiones de prueba sin `payer.email` o con el correo de tu equipo, nunca con el de un ejemplo de esta documentación. Si una sesión salió con otro correo, no se corrige: crea otra con una `Idempotency-Key` nueva.

La primera vez que pagas con un correo, antes de guardar la tarjeta el checkout te pide tu nombre y tu RFC; si entraste con Google, sólo el RFC, porque el nombre ya viene en tu cuenta. Si en pruebas no tienes un RFC a mano, usa el RFC genérico del SAT, `XAXX010101000`. Sirve sólo para tus datos como pagador: en `beneficiary.rfc` la API lo rechaza con `422 VALIDATION_ERROR`, porque ahí va el RFC real del condominio.

Tarjetas de prueba

Son las del procesador. Funcionan con **cualquier CVV** y **cualquier fecha de vencimiento futura**. No uses tarjetas reales aquí.

Al guardar una tarjeta, el checkout la verifica con 3D Secure. En pruebas, el procesador abre una ventana de un banco de prueba que pide un código:

escribe 1234.

Tarjeta	Qué pasa
`4456528080389860`	Pide el código 1234, la tarjeta se guarda y el cobro se aprueba. Úsala para pagar.
`5451951574925480`	La tarjeta se rechaza al guardarla: en pruebas no pasa la verificación 3D Secure.
`4574441215190335`	La tarjeta se rechaza al guardarla: sirve para probar tu manejo de error.
`4349001210846432`	La tarjeta se rechaza al guardarla: el procesador responde que no es compatible.

Con las de rechazo puedes comprobar que, tras un rechazo, la sesión sigue disponible para otro intento y tu integración no marca la obligación como pagada.

Cómo pides producción

Cuando tu integración pase el checklist, escríbenos con:

- tu `provider_id`,
- **la fecha en que hiciste las pruebas** (o el rango de días),
- el correo con el que entraste como pagador.

Con eso revisamos tu tráfico del ambiente de pruebas y confirmamos que cada punto del checklist ocurrió de verdad. No te pedimos capturas ni que nos cuentes lo que hiciste: lo vemos en los cobros, en los webhooks que entregamos y en el archivo de conciliación que descargaste.

Si algo falta, te decimos exactamente qué y lo repites. No es un examen — es que producción mueve dinero de personas y preferimos encontrarlo en pruebas.

Si el alta queda detenida

Escribe a tu contacto técnico en Neat con tu `provider_id` (si ya lo recibiste) y la razón social registrada. No incluyas tu API key ni tu `webhook_secret` en ese mensaje.

MÉXICO

Autenticación y ambientes

Al terminar esta página sabrás qué encabezado enviar en cada request, qué significa cada error de autenticación y qué distingue al ambiente de pruebas del de producción.

Encabezado de autorización

Cada request a un endpoint de comercio lleva tu API key como token portador:

```
Authorization: Bearer <tu API key>
```

No hay otro mecanismo de autenticación para el servidor de tu comercio: sin este encabezado, con un formato distinto a `Bearer <token>` o con el token vacío, la API responde `401 UNAUTHORIZED` antes de intentar validar la llave.

Situación	HTTP	Código
Sin encabezado `Authorization`, sin el prefijo `Bearer` o con el token vacío	`401`	`UNAUTHORIZED`
API key inexistente o revocada	`401`	`INVALID_API_KEY`
API key válida pero comercio deshabilitado	`403`	`PROVIDER_DISABLED`

Un `401 INVALID_API_KEY` no distingue entre una llave inexistente y una revocada. Ambos casos piden lo mismo de tu parte, confirmar la credencial vigente con Neat, así que la API no expone esa diferencia.

Cómo reconocer cada credencial

Neat te entrega dos credenciales distintas, y sólo una va en las llamadas a la API:

Credencial	Empieza con	Largo	Dónde va
API key de pruebas	`nmx_test_`	52 caracteres	`Authorization: Bearer <API key>` contra `https://api-beta.neatpagos.mx`.
API key de producción	`nmx_live_`	52 caracteres	`Authorization: Bearer <API key>` contra `https://api.neatpagos.mx`.
`webhook_secret`	`whsec_`	49 caracteres	Sólo en tu servidor, para verificar la firma de cada webhook. **Nunca** en `Authorization`.

Si tu código lee la llave de una variable de entorno o de un secret manager, comprueba que llegue completa antes de llamar: que empiece con `nmx_` y mida 52 caracteres, sin espacios ni saltos de línea alrededor.

Si recibes un 401

Código	Qué revisar
`UNAUTHORIZED`	La API no encontró un token. Lo típico es la variable de la llave vacía o sin definir, que deja el encabezado en `Bearer ` a secas. También pasa si falta el encabezado o si el prefijo no es exactamente `Bearer ` (con `B` mayúscula y un espacio): `bearer` en minúscula no sirve.
`INVALID_API_KEY`	Hay un token, pero no es una API key vigente de este ambiente. Las causas más comunes: mandaste el

Código	Qué revisar
	`webhook_secret` (`whsec_…`) en vez de la API key, la llave se cortó al copiarla, usas la llave de producción contra pruebas o al revés, la llave fue revocada, o en JavaScript la variable no está definida y el encabezado sale como `Bearer undefined`.

Cuando pidas ayuda, comparte el `request_id` de la respuesta, nunca la llave.

Qué identifica la API key

La API key identifica exclusivamente a tu comercio y a su ambiente (pruebas o producción). No hay un campo en el cuerpo de la petición para elegir otro comercio, otra cuenta de destino o otro ambiente: eso siempre sale de la credencial, nunca de la request.

Ambientes

Pruebas y producción son credenciales y despliegues separados, no un valor que se declare en la request. No existe un encabezado ni un parámetro de ambiente.

Ambiente	URL de API
Pruebas	`https://api-beta.neatpagos.mx`
Producción	`https://api.neatpagos.mx`

Las rutas van bajo `/v1`, así que una sesión de checkout de pruebas se crea con `POST https://api-beta.neatpagos.mx/v1/checkout-sessions`.

Fuera de `/v1`, la API responde a una sola ruta: `GET /health`, sin credencial, con `200` y un JSON corto. Sirve para comprobar desde tu servidor que llegas al ambiente correcto. No es parte del contrato de `v1`: no lleva datos de tu comercio, su cuerpo puede cambiar sin aviso y no lo uses para decidir si crear una sesión o no.

Los dos dominios atienden solo llamadas de servidor a servidor. La página que abre quien paga vive en otro dominio, el que viene en `checkout_url` al crear la sesión: no la armes tú a mano, porque ese dominio puede cambiar y el de la API no.

Usa siempre tu API key de pruebas contra el ambiente de pruebas. Una API key de producción contra el ambiente de pruebas, o viceversa, se comporta igual que una llave inválida: `401 INVALID_API_KEY`.

Formato de la request

- `Content-Type: application/json` en toda petición con cuerpo. Sin este encabezado, la API no reconoce el cuerpo como JSON.
- El cuerpo admite hasta 16 KiB.
- Todas las respuestas llevan `Cache-Control: no-store`: no caches una respuesta de la API en un proxy ni en el navegador.

Un cuerpo mal formado responde `400 INVALID_JSON`, uno sin `Content-Type: application/json` responde `415 UNSUPPORTED_MEDIA_TYPE` y uno mayor a 16 KiB responde `413 PAYLOAD_TOO_LARGE`. Los tres son errores de tu lado: corrige el request y no lo reintentes automáticamente. Ver el detalle en Errores.

Límites de tasa

Cada endpoint de comercio tiene su propio límite, por API key:

Endpoint	Límite
`POST /v1/checkout-sessions`	120 por minuto
`GET /v1/checkout-sessions/{sessionId}`	300 por minuto
`GET /v1/reconciliation/payments.csv`	10 por minuto

Al exceder el límite, la API responde `429 RATE_LIMITED` con un encabezado `Retry-After` en segundos. Respeta ese valor antes de reintentar. Exceder el límite no bloquea tu credencial ni afecta a otros comercios.

MÉXICO

Sesiones de checkout

Al terminar esta página sabrás qué campos acepta la creación de una sesión, cómo se identifica cada obligación de cobro, y cómo consultar su estado después de crearla.

Crear una sesión

```
POST /v1/checkout-sessions
```

cURL | Node

Crear una sesión de checkout

```
IDEMPOTENCY_KEY="checkout-$(uuidgen)"

curl -X POST "$NEAT_API_BASE/checkout-sessions" \
  -H "Authorization: Bearer $NEAT_API_KEY" \
  -H "Idempotency-Key: $IDEMPOTENCY_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "external_reference": "CUOTA-2026-09-000123",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "amount_minor": 125000,
    "currency": "MXN",
    "return_id": "principal",
    "payer": { "email": "<correo de tu equipo de pruebas>", "name":
"Ana Torres" },
    "metadata": { "order_id": "ORD-000123" }
  }'
```

`<correo de tu equipo de pruebas>` es un marcador: la API lo rechaza con 422 `VALIDATION_ERROR` si lo mandas tal cual. Pon ahí el correo de quien va a pagar, o quita el campo: con `payer.email`, sólo esa dirección puede pagar la sesión

(ver el aviso debajo de la tabla). Genera una `Idempotency-Key` nueva por cada intento de cobro, ver `Idempotency-Key`.

Cuerpo de la petición

La API rechaza cualquier propiedad fuera de esta lista con `422 VALIDATION_ERROR`.

Campo	Obligatorio	Tipo y validación
`external_reference`	Sí	Texto ASCII de 1 a 100 caracteres: letras, números, `.`, `_`, `-`. Identifica tu obligación de cobro; ver más abajo.
`category`	Según tu comercio	`maintenance_fee`, `rent` o `rent_guarantee_insurance`: qué está pagando el pagador. Opcional si tu comercio tiene habilitada una sola categoría. Ver Categoría del cobro.
`concept`	Sí	Texto de 1 a 140 caracteres. Sin caracteres de control ni </>. Se muestra al pagador.
`amount_minor`	Sí	Entero positivo en centavos: MXN 12,850.00 es `1285000`, y `12850` es MXN 128.50. Entre el mínimo y el máximo de tu comercio (ver nota abajo).
`currency`	Sí	Literal `"MXN"`. Cualquier otro valor devuelve `422 VALIDATION_ERROR`.
`return_id`	Sí	Texto ASCII de 1 a 50 caracteres: letras, números, `_`, `-`. Debe estar registrado y activo (ver Alta de tu comercio).
`payer`	No	Objeto con `email` y/o `name`. No puede ir vacío ni traer otras propiedades.

Campo	Obligatorio	Tipo y validación
`payer.email`	No	Correo válido de hasta 254 caracteres, guardado en minúsculas. **No es cosmético: fija quién puede pagar.** Ver el aviso debajo de la tabla.
`payer.name`	No	Texto de 1 a 100 caracteres, sin caracteres de control ni HTML.
`metadata`	No	Objeto de hasta 10 pares. Claves de 1 a 40 caracteres `[A–Za–z0–9_–]`; valores de texto de hasta 200 caracteres, sin caracteres de control. **Sólo de escritura**: no vuelve en ninguna respuesta, webhook ni en el CSV.
`beneficiary`	Según tu comercio	Solo si cobras por varios beneficiarios: quién recibe el dinero de este cobro. Obligatorio en ese caso y rechazado en cualquier otro. Ver Beneficiario por sesión.
`fee_bearer`	Según tu comercio	`payer` o `provider`: quién paga la comisión de Neat. Opcional si tu comercio tiene habilitada una sola forma. Ver Quién paga la comisión.

> ⚠ **`payer.email` decide quién puede pagar**
>
> Si mandas `payer.email`, el checkout no le pregunta el correo a quien llega: le muestra esa dirección enmascarada, le envía el código de acceso sólo a ella, y **sólo una identidad con ese correo exacto puede abrir y pagar la sesión**. Cualquier otra, incluida una cuenta de Google o Apple con otro correo, recibe `403 PAYER_MISMATCH`, aunque haya probado su propio buzón, y el checkout no le ofrece escribir otro. Si no lo mandas, quien llega escribe su correo en el checkout.
>
> Es deliberado: la liga de checkout está a un reenvío de distancia de cualquiera, y este campo es lo que impide que el cobro dirigido a una persona lo abra otra. Pero también significa que **un correo mal escrito deja al pagador fuera de su propia deuda**, sin forma de entrar. Si no estás seguro de la dirección, no mandes el campo: la sesión queda abierta a quien llegue primero con la liga, y esa persona la reclama.
>
> Una sesión creada con el correo equivocado no se corrige. Crea otra con el correo correcto, o sin `payer`, y una `Idempotency-Key` nueva; puedes repetir el `external_reference`, porque `payer` no es un término de la obligación. En pruebas, usa el correo de tu equipo: con un correo de ejemplo, nadie de tu equipo puede pagar la sesión.

Además, el cuerpo completo no puede superar 16 KiB.

Categoría del cobro

`category` dice qué está pagando el pagador, y con ella cambian el texto del checkout y de los correos que le mandamos:

Valor	El pagador ve
`maintenance_fee`	Cuota de mantenimiento
`rent`	Renta
`rent_guarantee_insurance`	Póliza de renta

Neat habilita las categorías de cada comercio en el alta. Si tu comercio tiene una sola, puedes omitir `category` y se usa esa; un comercio dado de alta antes de que existiera el campo cobra cuotas de mantenimiento y no tiene que cambiar nada. Si tiene varias, `category` es obligatoria y sin ella la API responde `422 VALIDATION_ERROR` con `field: "category"`. Una categoría que tu comercio no tiene habilitada responde `422 CATEGORY_NOT_ALLOWED`.

La categoría es un término de la obligación, como el monto: vuelve en la sesión, en el pago, en el webhook y en el archivo de conciliación, y reusar un `external_reference` con otra categoría responde `409 OBLIGATION_CONFLICT`.

Quién paga la comisión

`fee_bearer` dice quién paga la comisión de Neat en esta sesión:

Valor	El pagador paga	El depósito
`payer`	`amount_minor` más la comisión y su IVA	`amount_minor` completo
`provider`	Exactamente `amount_minor`; el checkout no muestra comisión	`amount_minor` menos la comisión y su IVA

Neat habilita las formas de cada comercio. Si tu comercio tiene una sola, puedes omitir `fee_bearer` y se usa esa. Un comercio dado de alta antes de que existiera el campo cobra la comisión al pagador y no tiene que cambiar nada. Si Neat después cambia esa forma única a que tu comercio la absorba, tus sesiones nuevas sin `fee_bearer` la absorben sin cambios; si mandabas `fee_bearer: "payer"`, empiezan a responder `422 FEE_BEARER_NOT_ALLOWED`. Si tiene las dos, `fee_bearer` es obligatorio y sin él la API responde `422 VALIDATION_ERROR` con `field: "fee_bearer"`. Una forma que tu comercio no tiene habilitada responde `422 FEE_BEARER_NOT_ALLOWED`.

Queda fijo en la sesión al crearla: lo que el pagador ve en el checkout es lo que se le cobra, aunque Neat cambie quién paga la comisión en tu comercio mientras la sesión está abierta. Vuelve en la respuesta de creación, en la

sesión, en el pago, en el webhook y en el archivo de conciliación. Los montos y el cálculo están en Importes y comisiones.

Con `provider`, el monto tiene que dejar algo que depositar con cualquier tarjeta; si no lo deja, la API responde `422 AMOUNT_BELOW_FEE` con `field: "amount_minor"`. Con las tasas de tarjeta esto solo pasa con montos de unos pocos centavos, muy por debajo del mínimo.

Beneficiario por sesión

Si tu sistema cobra en nombre de varios condominios o administradoras, cada sesión dice a quién le pagamos. Neat deposita directo en la cuenta del beneficiario; tu comercio no recibe ni reparte el dinero. Este modo lo habilita Neat para tu comercio: sin él, `beneficiary` responde `422 BENEFICIARY_NOT_ALLOWED`; con él, una sesión sin `beneficiary` responde `422 BENEFICIARY_REQUIRED`.

Datos de ejemplo, ficticios

```
"beneficiary": {
  "reference": "CONDO-00123",
  "legal_name": "CONDOMINIO EJEMPLO LAS PALMAS AC",
  "rfc": "CEP010101AB1",
  "clabe": "032180000118359719"
}
```

Estos datos son ficticios: no los mandes tal cual, ni en pruebas. Usa los del condominio de cada cobro. El Quickstart tiene la llamada completa en cURL, Node y PHP.

Campo	Validación
`reference`	Tu identificador del beneficiario, de 1 a 64 caracteres `[A-Za-z0-9._-]`. Usa siempre el mismo para el mismo condominio.
`legal_name`	Razón social del titular de la cuenta, de 1 a 120 caracteres, sin caracteres de control, invisibles ni HTML. Es el nombre que ve el pagador.

Campo	Validación
`rfc`	RFC del titular: 12 caracteres para persona moral, 13 para persona física. Se guarda en mayúsculas. Los RFC genéricos del SAT (`XAXX010101000`, `XEXX010101000`) responden `422`.
`clabe`	18 dígitos **como texto**, con su dígito verificador válido. Si la lees de una columna numérica, conviértela a texto antes de serializar: en PHP, `json_encode` de un entero pierde los ceros iniciales (002, 012, 014) y la API responde `422`.

Cómo lo trata Neat:

- **La cuenta es un término de la obligación**, igual que el monto: reutilizar un `external_reference` con otro `beneficiary` responde `409 OBLIGATION_CONFLICT`. Por eso el `external_reference` tiene que ser único entre todos tus condominios, por ejemplo `CONDO-00123-2026-09-402`. Si te equivocaste en la CLABE antes del primer pago, escríbenos con el `external_reference`; no la corrijas cambiando la referencia.
- **Neat deposita a la CLABE que declara la sesión**, a nombre del titular que declara. Revisa que la CLABE, el RFC y la razón social correspondan al condominio antes de mandarlos: el depósito sigue esos datos.
- **La CLABE no vuelve en ninguna respuesta**, webhook, correo ni archivo de conciliación. Tampoco la pongas en `metadata`: en este modo, un valor con forma de CLABE ahí responde `422 VALIDATION_ERROR`.
- En la página de pago y en la confirmación, el pagador ve «Le pagas a», seguido de `legal_name`, y «vía», seguido del nombre de tu comercio. Si los dos nombres coinciden, «vía» no aparece.

Cuándo recibe el dinero el condominio

Después de confirmar el cobro, Neat transfiere por SPEI a la CLABE que declaró la sesión, a nombre de su `legal_name`. Lo que conviene saber hoy:

- **`payment.succeeded` confirma el cobro, no el depósito.** La transferencia al condominio es un paso posterior.

- **Quien paga se entera por correo**: recibe uno al confirmarse el cobro y otro cuando la transferencia se hace o si falla.
- **Si tu comercio absorbe la comisión, el depósito llega neto**: el monto es `provider_net_amount_minor`, no `amount_minor`. Avísale al condominio.
- **Tu comercio no recibe aviso de la transferencia.** No hay un evento de webhook para ella, y el archivo de conciliación no trae su estado. Si un condominio pregunta por un depósito, escríbenos con el `external_reference` del cobro.

Esta documentación no fija un plazo de depósito. Si necesitas uno para informar a tus condominios, pregúntale a tu contacto en Neat.

> **Límites de monto**
>
> `amount_minor` tiene un mínimo de 2000 centavos (MXN 20.00) y un máximo que es el límite que acordaste con Neat en el alta. Quedarte fuera de cualquiera de los dos devuelve `422 VALIDATION_ERROR` con `"field": "amount_minor"`, y el `message` dice el límite en centavos y en pesos. Los dos límites se ajustan por comercio: si necesitas otro, pídelo a tu contacto en Neat.

`external_reference`: identifica tu obligación, no tu intento de cobro

Neat usa `external_reference` para reconocer la misma obligación de cobro entre distintas llamadas de tu comercio. Reutilizarlo se comporta así:

Situación	Resultado
Referencia nueva	Crea una obligación y una sesión nuevas.
Misma referencia, mismos `amount_minor`, `currency`, `category`, `concept` y `beneficiary`, sin pago exitoso ni en curso	Abre otra sesión sobre la misma obligación (por ejemplo, para reintentar tras un enlace vencido). La sesión anterior **sigue viva** con su propio enlace; ver debajo.

Situación	Resultado
Misma referencia con `amount_minor`, `currency`, `category`, `concept` o `beneficiary` distintos	`409 OBLIGATION_CONFLICT`, **aunque la obligación no se haya pagado**.
Misma referencia y mismos términos, obligación ya pagada	`409 OBLIGATION_ALREADY_PAID`
Misma referencia y mismos términos, cobro en curso	`409 PAYMENT_IN_PROGRESS`

Si el cliente paga cada mes, usa una referencia que distinga el período: `CUOTA-2026-09-000123` y `CUOTA-2026-10-000123` son obligaciones distintas. `payer`, `metadata` y `fee_bearer` no son términos: cambiarlos no provoca el `409`. `fee_bearer` queda fijo en cada sesión, así que dos ligas vivas de la misma referencia creadas con distinto `fee_bearer` le muestran al pagador totales distintos.

Por qué los términos no cambian

La primera sesión fija los términos de la obligación, y un `concept` con otro texto ya cuenta como distinto. Es a propósito:

- **Todas las ligas de una referencia cobran la misma obligación.** Solo la comisión puede variar entre ligas creadas con distinto `fee_bearer`. La sesión anterior sigue viva y puede estar en manos del pagador; si una llamada nueva pudiera cambiar el monto, esa liga cobraría algo que el pagador no vio.
- **Una obligación no se reapunta a otra cuenta.** Si cobras por beneficiario, la CLABE es un término: una segunda llamada no puede desviar a otra cuenta el dinero de una deuda ya emitida.
- **Tu conciliación se cruza por esta referencia.** Los webhooks y el archivo CSV la traen: una referencia, un monto, un beneficiario.

Si la cuota cambia antes de pagarse

Un recargo por mora o una corrección de monto o de concepto es otra obligación para la API. Usa una referencia nueva con una versión, por ejemplo `CUOTA-2026-09-000123-R1`, pero sólo después de dos comprobaciones sobre la referencia anterior:

1. **Que haya pasado el `expires_at` de su sesión más reciente.** Hasta entonces la liga puede iniciar un pago, y una sesión no se puede cerrar antes. Que el pagador ya no tenga la liga no basta: puede haberla guardado o recuperarla del correo.
2. **Que ninguna de sus sesiones tenga un pago `succeeded` ni `pending`.** Consulta con `GET /v1/checkout-sessions/{sessionId}` cada sesión que creaste para esa referencia, no sólo la última: cada una devuelve su propio pago, y un pago iniciado en una sesión anterior, o justo antes del vencimiento, puede seguir en curso.

Si emites la referencia nueva antes, el pagador puede terminar pagando las dos, y serían dos cobros.

Si lo que está mal es la CLABE y todavía no hay pago, escríbenos con el `external_reference`; no la corrijas con una referencia nueva.

Dos cosas que conviene saber al reutilizar una referencia:

- **La sesión anterior no se cierra.** Su enlace sigue funcionando hasta su propio `expires_at`, así que por un rato existen dos ligas vivas para la misma obligación. Sólo una puede terminar en cobro —la primera que reserve el pago gana y la otra recibe `409 PAYMENT_IN_PROGRESS` o `409 CHECKOUT_NOT_PAYABLE`—, pero si le mandaste las dos al pagador, va a ver una fallar.
- **El `metadata` de la sesión nueva no reemplaza al de la obligación.** El de la primera es el que queda. Como además `metadata` no vuelve en ninguna respuesta, no hay forma de consultarlo: si el dato tiene que viajar, mándalo en `external_reference` o guárdalo de tu lado contra el `session_id`.

Idempotency-Key

Obligatoria en cada creación. Genera una nueva por cada intento de cobro, guárdala junto a ese intento y repítela sólo si reintentas esa misma llamada con el mismo cuerpo. Lleva de 8 a 128 caracteres ASCII: letras, números, `.`, `_`, `:` y –. Una llave fuera de ese largo o con otro carácter responde 422 `VALIDATION_ERROR` con `"field": "Idempotency-Key"`. Ver el detalle completo, incluyendo qué pasa con solicitudes repetidas, en Reintentos e idempotencia.

Respuesta

201 Created

```
{
  "session_id": "cs_164596add1a3",
  "checkout_url":
"https://pago.neat.example/pagar/cs_164596add1a3.CQGg0fEWT5o…",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "status": "open",
  "fee_bearer": "payer"
}
```

Campo	Descripción
`session_id`	Identificador de la sesión. Único en tu comercio.
`checkout_url`	Página hospedada por Neat a la que debes redirigir al pagador. Termina en `/pagar/<token>`; el token es opaco y se puede abrir las veces que haga falta hasta `expires_at` o hasta que la sesión deje de estar `open`.
`expires_at`	Vencimiento del enlace: 30 minutos después de creado, fijo.
`status`	Siempre `"open"` en la respuesta de creación.
`fee_bearer`	Quién paga la comisión en esta sesión: el que mandaste, o el que Neat habilitó para tu comercio si no mandaste ninguno.

Consultar una sesión

```
GET /v1/checkout-sessions/{sessionId}
```

cURL | Node

Consultar una sesión

```
curl "$NEAT_API_BASE/checkout-sessions/cs_164596add1a3" \
-H "Authorization: Bearer $NEAT_API_KEY"
```

200 OK — antes de cualquier intento de pago


```
{
  "session_id": "cs_164596add1a3",
  "status": "open",
  "external_reference": "CUOTA-2026-09-000123",
  "category": "maintenance_fee",
  "concept": "Mensualidad de mantenimiento, septiembre 2026",
  "amount_minor": 125000,
  "currency": "MXN",
  "fee_bearer": "payer",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "payment": null
}
```


`payment` es `null` hasta el primer intento de cobro. Después de un intento, muestra el pago más reciente de la sesión, ver la forma completa de `payment` en Importes y comisiones. Un `sessionId` que no existe, o que pertenece a otro comercio, devuelve `404 RESOURCE_NOT_FOUND` en ambos casos; la API nunca revela si el identificador existe bajo otra credencial.

Errores de esta sección

HTTP	Código	Cuándo
401	`UNAUTHORIZED` / `INVALID_API_KEY`	Ver Autenticación y ambientes.
403	`PROVIDER_DISABLED`	Tu comercio está deshabilitado.
404	`RESOURCE_NOT_FOUND`	La sesión no existe o pertenece a otro comercio.
409	`IDEMPOTENCY_CONFLICT`	Repetiste `Idempotency-Key` con un cuerpo distinto.
409	`OBLIGATION_CONFLICT` / `OBLIGATION_ALREADY_PAID` / `PAYMENT_IN_PROGRESS`	Ver la tabla de `external_reference` arriba.
422	`VALIDATION_ERROR`	Un campo no cumple su formato, o el monto está fuera de tus límites.
422	`RETURN_NOT_REGISTERED`	El `return_id` no está registrado o no está activo.
422	`BENEFICIARY_REQUIRED` / `BENEFICIARY_NOT_ALLOWED`	Ver Beneficiario por sesión.
422	`CATEGORY_NOT_ALLOWED`	Ver Categoría del cobro.
422	`FEE_BEARER_NOT_ALLOWED` / `AMOUNT_BELOW_FEE`	Ver Quién paga la comisión.
429	`RATE_LIMITED`	Excediste el límite por minuto de este endpoint.

Catálogo completo en Errores.

MÉXICO

Redirección y regreso

Al terminar esta página sabrás qué parámetros trae la URL de regreso y por qué nunca debes usarla para marcar una obligación como pagada.

Ir al checkout

Redirige el navegador del pagador a `checkout_url`, tal como la devolvió `POST /v1/checkout-sessions` (ver Sesiones de checkout). Neat aloja toda la experiencia de pago ahí: inicio de sesión del pagador, selección de tarjeta y confirmación. Tu servidor no participa en esa parte del flujo ni recibe datos de tarjeta.

Regreso al comercio

Al terminar, con pago exitoso, con error o porque el pagador salió sin pagar, Neat redirige el navegador a la URL registrada bajo tu `return_id`, con estos parámetros de consulta:

```
https://comercio.example/regreso?
session_id=cs_164596add1a3&status=pagado&payment_status=succeeded&paymen
```

Parámetro	Siempre presente	Valores
`session_id`	Sí	El identificador de la sesión.
`status`	Sí	`abortado` o `pagado`.
`payment_status`	Sí	`not_started`, `pending`, `failed`, `succeeded`.
`payment_id`	Solo si ya existe un intento de cobro	El pago más reciente de la sesión.

`status: "pagado"` significa que, al momento de la redirección, la sesión tenía un pago confirmado (`succeeded`, `partially_refunded` o `refunded`). En cualquier otro caso (el pagador salió antes de pagar, el intento falló o quedó `pending`) el valor es `abortado`; usa `payment_status` para distinguir esos tres casos.

Ejemplos:

Pago confirmado

```
https://comercio.example/regreso?
session_id=cs_164596add1a3&status=pagado&payment_status=succeeded&paymen
```

El pagador salió sin iniciar un cobro

```
https://comercio.example/regreso?
session_id=cs_5b05557c4a90&status=abortado&payment_status=not_started
```

Por qué el regreso no confirma el pago

La redirección es para mostrar una pantalla al pagador; no es prueba de cobro. El navegador puede cerrarse antes de completar la redirección, el parámetro puede perderse en un proxy, y nada impide que alguien arme esa misma URL a mano. Antes de marcar una obligación como pagada, confirma con una de estas dos fuentes:

1. El webhook `payment.succeeded`, verificado por firma, ver Webhooks: eventos y firma y Verificar la firma.
2. `GET /v1/checkout-sessions/{sessionId}`, ver Sesiones de checkout.

Si el pagador cierra la página antes de volver a tu sitio, el cobro sigue su curso del lado de Neat: consulta la sesión o espera el webhook, no le pidas al pagador que repita el pago.

MÉXICO

Estados y glosario

Al terminar esta página sabrás qué significa cada estado de sesión y de pago, y qué transiciones maneja tu integración.

Estados de la sesión (`status`)

Estado	Significado
`open`	El pagador puede continuar. Es el estado inicial y también el estado al que vuelve la sesión después de un intento de cobro fallido.
`processing`	Hay un intento de cobro en curso.
`completed`	El checkout terminó con un pago confirmado (`succeeded`).
`canceled`	El pagador salió antes de completar un cobro, o una devolución posterior canceló la obligación pagada.
`expired`	Reservado. La API no lo asigna hoy; ver la nota de vencimiento más abajo.

Transiciones observadas

De	A	Disparador
Nueva	`open`	`POST /v1/checkout-sessions`
`open`	`processing`	El pagador confirma un medio de pago (`POST /v1/checkout/{token}/pay`).
`processing`	`completed`	El cobro se confirma (`payment.status = succeeded`).

De	A	Disparador
`processing`	`open`	El intento de cobro falla (`payment.status = failed`); el pagador puede reintentar mientras la sesión no haya vencido.
`open`	`canceled`	El pagador sale del checkout sin haber iniciado un cobro.
`completed`	`canceled`	Una solicitud de cancelación del pagador se confirma por completo, ver más abajo.

> ⚠ **Vencimiento: usa `expires_at`, no `status`**
>
> La API no escribe `status: "expired"` hoy, aunque el valor exista en el esquema. Una sesión sin intentos de cobro sigue mostrando `status: "open"` en `GET /v1/checkout-sessions/{sessionId}` después de vencer. Para saber si un enlace sigue vigente, compara `expires_at` con la hora actual.

Estados del pago (`payment.status`)

Estado	Significado
`pending`	Neat todavía no tiene un resultado definitivo del cobro.
`succeeded`	El cobro fue confirmado. No implica por sí solo que el dinero ya se depositó a tu comercio.
`failed`	Este intento fue rechazado de forma definitiva. Un reintento explícito del pagador genera otro `payment_id`.
`partially_refunded`	**Reservado.** Existe en el contrato y v1 no lo emite nunca: hoy toda cancelación devuelve el cargo completo.

Estado	Significado
`refunded`	El cargo fue devuelto por completo y confirmado.

Un pago llega a `refunded` cuando el pagador solicita la cancelación desde su historial en Neat y el adquirente confirma la devolución; tu comercio se entera por el webhook `payment.refunded`. Ver Webhooks: eventos y firma.

No hay devoluciones parciales en v1. Una cancelación devuelve el cargo completo, con la comisión si la pagó el pagador, así que `refunded_amount_minor` siempre termina igual a `charged_amount_minor`. El estado `partially_refunded` está reservado por si eso cambia; si programas contra él hoy, estás manejando un caso que no ocurre.

Estado de cancelación (`cancellation_state`)

Acompaña a `payment.status` cuando el pagador pide cancelar un cobro ya confirmado.

Estado	Significado
`none`	No hay una cancelación en curso.
`requested`	El pagador pidió cancelar; Neat está confirmando con el adquirente.
`completed`	La cancelación se confirmó. `payment.status` pasa a `refunded` y `refunded_amount_minor` iguala al monto cobrado completo, con la comisión de Neat e IVA si los pagó el pagador: al pagador se le devuelve todo lo que se le cobró.
`rejected`	El adquirente no confirmó la cancelación. `payment.status` no cambia.

Si el adquirente no responde de forma definitiva a la solicitud de cancelación, el pago se queda en `requested` y Neat la resuelve con el adquirente antes de moverla. Tu comercio no necesita hacer nada durante esa espera: el

webhook o la consulta de la sesión te avisan cuando se resuelva. Una cancelación que lleve más de un día sin resolverse la atiende una persona en Neat.

Glosario

Término	Significado
Sesión (`checkout_session`)	Un intento de checkout: tiene un enlace, un vencimiento y, eventualmente, un pago.
Obligación	Lo que el pagador debe, identificado por tu `external_reference`. Sobrevive a varias sesiones si la primera vence sin pagarse.
Pago (`payment`)	Un intento de cobro contra una obligación. Cada intento tiene su propio `payment_id`.
`version`	Entero creciente del pago; aumenta con cada cambio de estado confirmado (por ejemplo, al pasar de `pending` a `succeeded`, o al completarse una cancelación).

MÉXICO

Importes y comisiones

Al terminar esta página sabrás en qué unidad vienen los montos, quién paga la comisión de Neat, cómo se calcula y qué monto neto le corresponde a tu comercio.

Formato de montos

Todo campo terminado en `_minor` está en centavos, como entero. `125000` significa MXN 1,250.00. La única moneda soportada en `v1` es `"MXN"`. Las fechas usan RFC 3339 en UTC (`2026-09-14T23:17:17.894Z`).

Quién paga la comisión

Hay dos formas, y cada pago dice cuál se usó en `fee_bearer`:

`fee_bearer`	Qué paga el pagador	Qué se deposita
`"payer"`	El monto de la obligación más la comisión y su IVA	El monto completo de la obligación
`"provider"`	Exactamente el monto de la obligación, sin comisión	El monto de la obligación menos la comisión y su IVA

Si no acordaste otra cosa con Neat, la comisión la paga el pagador (`"payer"`). Que tu comercio la absorba (`"provider"`) es un acuerdo comercial que Neat habilita en tu comercio. Si lo habilita como única forma, tus sesiones nuevas lo usan sin cambios, siempre que no mandes `fee_bearer`. Si Neat habilita las dos formas, `fee_bearer` es obligatorio en cada sesión; sin él la API responde `422 VALIDATION_ERROR` (ver Sesiones de checkout).

Si cobras por varios beneficiarios, con `"provider"` la comisión y su IVA se descuentan del depósito a la CLABE de cada beneficiario: el condominio

recibe `provider_net_amount_minor`, no `amount_minor`. Acuerda con cada condominio quién asume esa diferencia antes de pedirle a Neat que lo habilite.

La forma queda fija en la sesión al crearla y en el pago al reservarlo. Si Neat cambia tu acuerdo después, los pagos que ya existen no cambian: ni lo que se cobró ni lo que te corresponde.

Cómo se calcula la comisión

La comisión es la misma pague quien la pague. Siempre se calcula sobre el monto de la obligación y cada componente se redondea hacia arriba al centavo, de forma independiente:

```
comisión     = ceil(principal_amount_minor × tasa_de_comisión)
IVA          = ceil(comisión × tasa_de_IVA)
```

Lo único que cambia es dónde queda:

```
fee_bearer "payer":
  payer_fee_minor = comisión,  payer_fee_tax_minor = IVA
  provider_fee_minor = 0,      provider_fee_tax_minor = 0
  charged_amount_minor      = principal_amount_minor + comisión + IVA
  provider_net_amount_minor = principal_amount_minor

fee_bearer "provider":
  payer_fee_minor = 0,         payer_fee_tax_minor = 0
  provider_fee_minor = comisión, provider_fee_tax_minor = IVA
  charged_amount_minor      = principal_amount_minor
  provider_net_amount_minor = principal_amount_minor − comisión − IVA
```

En todo pago confirmado se cumple `charged_amount_minor − provider_net_amount_minor = payer_fee_minor + payer_fee_tax_minor + provider_fee_minor + provider_fee_tax_minor`.

Las tasas de referencia son 3 % del principal (3.5 % con American Express) más 16 % de IVA sobre la comisión. Las tarjetas de débito pagan lo mismo que

las de crédito. Las tasas vigentes para tu comercio son las que acordaste con Neat en el alta. Cuando tu comercio absorbe la comisión, el pagador paga lo mismo con cualquier tarjeta; la diferencia de tasa de American Express la absorbe tu comercio.

Si tu comercio absorbe la comisión, el monto tiene que dejar algo que depositar con cualquier tarjeta. Una sesión cuyo monto no lo deja se rechaza con `422 AMOUNT_BELOW_FEE`. Con las tasas de tarjeta esto solo pasa con montos de unos pocos centavos, muy por debajo del mínimo de MXN 20.00.

Ejemplo verificado

Cobro de MXN 1,250.00 (`amount_minor: 125000`):

Campo	`"payer"`, tarjeta	`"provider"`, tarjeta	`"provider"`, American Express
`principal_amount_minor`	125000	125000	125000
`payer_fee_minor`	3750	0	0
`payer_fee_tax_minor`	600	0	0
`provider_fee_minor`	0	3750	4375
`provider_fee_tax_minor`	0	600	700
`charged_amount_minor`	129350	125000	125000
`provider_net_amount_minor`	125000	120650	119925

Qué devuelve una cancelación

Una cancelación confirmada le devuelve al pagador `charged_amount_minor` completo, que es lo que pagó. Neat no se queda con la comisión de un cobro que se anuló, la haya pagado el pagador o tu comercio: si la absorbía tu comercio, no paga comisión por ese cobro. En la contabilidad de tu comercio, una anulación revierte el neto que ibas a recibir y nada más.

Campos de un pago

Estos campos aparecen tanto en `payment` dentro de una sesión (ver Sesiones de checkout) como en `data` dentro de un evento de webhook (ver Webhooks: eventos y firma). Es el mismo objeto en ambos lugares.

Campo	Descripción
`principal_amount_minor`	Monto de la obligación para este intento.
`payer_fee_minor`, `payer_fee_tax_minor`	Comisión e IVA que paga el pagador. `0` si los absorbe tu comercio.
`provider_fee_minor`, `provider_fee_tax_minor`	Comisión e IVA que absorbe tu comercio y se descuentan del depósito. `0` si los paga el pagador.
`charged_amount_minor`	Total cobrado al pagador. `null` mientras el pago está `pending`; `0` si el intento falló definitivamente.
`provider_net_amount_minor`	Monto neto que corresponde a tu comercio por este pago. `null` mientras está `pending`; `0` si falló definitivamente.
`refunded_amount_minor`	Total devuelto y confirmado. `0` hasta la primera devolución confirmada.
`fee_bearer`	Quién pagó la comisión: `"payer"` o `"provider"`. Fijo desde que se reservó el pago.
`paid_at`	Fecha de confirmación del cobro. `null` antes de confirmarse.

Devoluciones y cancelaciones

Cuando una cancelación se confirma por completo (`cancellation_state: "completed"`), Neat devuelve el monto **total** cobrado, no solo el principal:

```
refunded_amount_minor = charged_amount_minor  (al completarse la cancelación)
```

Ejemplo: un cobro de MXN 800.00 con la comisión a cargo del pagador (`principal_amount_minor: 80000`, `charged_amount_minor: 82784`) cancelado y confirmado quedó con `refunded_amount_minor: 82784`, el total cobrado, no los 80000 originales. Con la comisión a cargo de tu comercio, el total cobrado es el principal y eso es lo que se devuelve.

`provider_net_amount_minor` no se ajusta automáticamente al devolverse un pago: el efecto sobre el saldo de tu comercio se refleja por separado en tu conciliación, ver Archivo CSV.

MÉXICO

Webhooks: eventos y firma

Al terminar esta página conocerás los cuatro eventos que Neat puede enviar a tu URL de webhook, la forma exacta de su cuerpo y cómo viene firmado.

Eventos

Evento	Cuándo se envía
`payment.pending`	Un intento de cobro quedó registrado y todavía no tiene resultado definitivo.
`payment.succeeded`	Un cobro fue confirmado.
`payment.failed`	Un intento de cobro fue rechazado de forma definitiva.
`payment.refunded`	Un cobro confirmado fue devuelto por completo.

Cuándo Neat notifica a tu comercio

Neat encola un evento cuando registra el resultado de un cobro —`succeeded`, `failed` o `pending`— y cuando se completa una cancelación (`payment.refunded`). Si tu comercio no tiene URL de webhook y `webhook_secret` registrados, Neat no encola ningún evento ni lo guarda para después. Confirma tu webhook antes de aceptar cobros reales, ver Pruebas y checklist.

> ⚠ **Un `payment.pending` se resuelve a mano y puede tardar**
>
> `payment.pending` significa que el adquirente no dio una respuesta definitiva y que Neat no le vuelve a preguntar por su cuenta. Alguien de Neat revisa el cobro en la consola del adquirente y registra el resultado; en ese momento recibes `payment.succeeded` o `payment.failed` para el mismo `payment_id`, con una `resource.version` mayor. Puede pasar minutos u horas después; si un cobro tuyo lleva horas en `pending`, escríbenos.
>
> Mientras tanto, `GET /v1/checkout-sessions/{sessionId}` devuelve el estado vigente. No inicies otro cobro para la misma obligación: si el primero sí se cobró, el pagador pagaría dos veces.

Formato de la notificación

```
POST <tu URL de webhook>
Content-Type: application/json
Neat-Event-Id: evt_8f21c2a6b3e9
Neat-Signature:
t=1789427837,v1=f1f09c2b66d45b85429a5328428a138dd762b3c530ea08bc022aa4e7
```

payment.succeeded


```
{
  "event_id": "evt_8f21c2a6b3e9",
  "type": "payment.succeeded",
  "api_version": "v1",
  "provider_id": "administradora_ejemplo",
  "occurred_at": "2026-09-14T23:17:17.894Z",
  "resource": {
    "type": "payment",
    "id": "pay_3ac4efd9f068",
    "version": 2,
    "session_id": "cs_6b1d0f4a92c7",
    "external_reference": "depto-301-2026-09"
  },
  "data": {
    "payment_id": "pay_3ac4efd9f068",
    "version": 2,
    "status": "succeeded",
    "currency": "MXN",
    "principal_amount_minor": 125000,
    "payer_fee_minor": 3750,
    "payer_fee_tax_minor": 600,
    "provider_fee_minor": 0,
    "provider_fee_tax_minor": 0,
    "charged_amount_minor": 129350,
    "provider_net_amount_minor": 125000,
    "refunded_amount_minor": 0,
    "fee_bearer": "payer",
    "cancellation_state": "none",
    "paid_at": "2026-09-14T23:17:17.894Z",
    "category": "maintenance_fee",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "payee_name": "Administradora Ejemplo, S.A. de C.V."
  }
}
```


La firma del ejemplo se calculó con un `webhook_secret` de muestra (`whsec_demo_example_secret_do_not_use`) sobre el JSON **exactamente como**

se ve arriba, saltos de línea e indentación incluidos: el HMAC es sobre bytes, así que reformatear el cuerpo cambia la firma. Sirve para probar tu verificador copiando y pegando. El `webhook_secret` real es el que Neat te entrega en el alta, y las entregas de verdad vienen en JSON compacto.

Campo	Descripción
`event_id`	Identifica el hecho. No cambia entre reintentos del mismo evento. Igual al valor de `Neat-Event-Id`.
`type`	Uno de los cuatro eventos de la tabla anterior.
`api_version`	Siempre `"v1"`.
`provider_id`	El `provider_id` de tu comercio.
`occurred_at`	Fecha del hecho. No cambia entre reintentos.
`resource.type`	Siempre `"payment"` en `v1`.
`resource.id`, `resource.version`	El `payment_id` afectado y su `version` en ese momento.
`resource.session_id`	La sesión de checkout del pago, el mismo valor que recibiste al crearla.
`resource.external_reference`	Tu propia referencia de la obligación, tal como la mandaste al crear la sesión.
`data`	El pago completo, en la misma forma que `payment` dentro de una sesión, ver Importes y comisiones.

No hay un campo `environment` en el cuerpo del evento: el ambiente (prueba o producción) está implícito en cuál `webhook_secret` verificó la firma.

Firma

Cada entrega trae dos cabeceras:

- `Neat-Event-Id`: igual a `event_id` en el cuerpo. Úsalo para deduplicar.
- `Neat-Signature`: `t=<segundos unix>,v1=<hexadecimal>`, un HMAC-SHA256 del texto `"<t>.<cuerpo crudo>"`, con tu `webhook_secret` como clave.

El detalle de cómo calcular y comparar la firma, con ejemplos en Node, Python y PHP, está en Verificar la firma. Nunca proceses un evento cuya firma no hayas verificado.

Qué hacer con cada evento

Evento	Acción esperada
`payment.pending`	No inicies otro cobro para la misma obligación mientras esté pendiente.
`payment.succeeded`	Marca la obligación como pagada para tu `external_reference`, que viene en `resource.external_reference`.
`payment.failed`	Registra el rechazo; la obligación sigue abierta para un nuevo intento del pagador.
`payment.refunded`	Registra la devolución y aplica tu propia política sobre la obligación, ver Importes y comisiones.

Un evento puede llegar fuera de orden. Compara `resource.version` contra la versión que ya tengas guardada para ese `payment_id` y no dejes que un evento antiguo sobreescriba un estado más nuevo, ver Reintentos e idempotencia.

MÉXICO

Reintentos e idempotencia

Al terminar esta página sabrás cuántas veces y con qué espaciado Neat reintenta un webhook no confirmado, y cómo usar `Idempotency-Key` para que reintentar una request desde tu servidor nunca abra un cobro duplicado.

Reintentos de webhook

Neat espera hasta 9 segundos por una respuesta a cada entrega. Confirma la entrega con cualquier `2xx`. Se reintenta con espera creciente cuando no respondes, cuando la conexión falla y cuando respondes `408`, `429` o cualquier `5xx`. **Cualquier otra respuesta se toma como un rechazo definitivo: la entrega queda marcada como fallida y no se reintenta**, porque un `400` o un `404` no mejoran solos. Si tu endpoint devuelve `4xx` por una caída temporal, devuelve `503` en su lugar.

> **Una redirección también es un fallo definitivo**
>
> Neat **no sigue redirecciones** al entregar un webhook, y un `3xx` cae en el mismo grupo que un `400`: la entrega se marca fallida al primer intento y no se reintenta.
>
> Esto muerde en el caso más común de todos. Si registras `http://` y tu servidor redirige a `https://`, o si tu URL lleva o le falta la barra final, o si redirige de `www` al dominio desnudo, **no vas a recibir ningún evento** — y no vas a ver ningún error, porque desde tu lado la request llegó.
>
> Registra la URL final, exacta, con `https://` y con la barra tal como la sirves. Pruébala con `curl -i` y confirma que responde `200` sin un `Location` de por medio.

La espera crece así:

Intento	Espera hasta el siguiente
1	2 minutos
2	4 minutos
3	8 minutos
4	16 minutos
5	32 minutos
6	64 minutos
7	128 minutos
8 en adelante	256 minutos

Neat reintenta hasta 12 veces o durante 24 horas desde el primer intento, lo que ocurra primero. Al agotarse, el evento queda marcado como no entregado y Neat no lo reintenta más por su cuenta; el cobro sigue visible en `GET /v1/checkout-sessions/{sessionId}`. Estos tiempos describen el comportamiento actual de la API, no un compromiso contractual.

El timestamp y la firma se renuevan en cada intento; `event_id` y el cuerpo permanecen iguales entre reintentos del mismo evento.

Responde rápido, procesa después

Guarda el evento por su `event_id` y responde `2xx` de inmediato; procesa el efecto (marcar la obligación como pagada, por ejemplo) de forma asíncrona si tu procesamiento puede tardar. Una respuesta lenta cuenta como no confirmada a los 9 segundos y dispara un reintento, aunque tu servidor sí lo haya recibido.

Deduplica por `Neat-Event-Id`

La entrega es al menos una vez: el mismo evento puede llegar más de una vez, incluida una entrega después de que ya lo procesaste. Guarda los

`event_id` que ya procesaste y responde `2xx` sin volver a aplicar el efecto ante uno repetido.

`Idempotency-Key` al crear una sesión

`Idempotency-Key` es obligatoria en `POST /v1/checkout-sessions`. Sin ella, la API responde `422 VALIDATION_ERROR` con `"field": "Idempotency-Key"`. Usa entre 8 y 128 caracteres ASCII (`[A-Za-z0-9._:-]`).

La llave identifica esa solicitud de creación, no un cobro. Si tu llamada termina en timeout o no recibes respuesta, repite exactamente la misma llave con exactamente el mismo cuerpo.

Situación	Resultado
Llave nueva	Crea una sesión nueva (sujeto a las reglas de `external_reference`, ver Sesiones de checkout).
Misma llave, mismo cuerpo	Devuelve la misma respuesta guardada, mismo `session_id`, mismo `checkout_url`, mismo `expires_at`, con `201 Created` otra vez, aunque esa sesión ya haya vencido. No abre una segunda sesión.
Misma llave, cuerpo distinto	`409 IDEMPOTENCY_CONFLICT`.
Sin cabecera `Idempotency-Key`, o con un formato inválido	`422 VALIDATION_ERROR`, `"field": "Idempotency-Key"`.

La comparación de “mismo cuerpo” es sobre el contenido ya validado del request, no sobre el texto literal: cambiar el orden de las propiedades del JSON no cambia el resultado.

Los registros de `Idempotency-Key` no expiran: la misma llave con un cuerpo distinto responde `409 IDEMPOTENCY_CONFLICT` indefinidamente. Genera una

llave nueva por cada intento de cobro, por ejemplo un UUID, y guárdala junto a ese intento: el reintento de esa llamada usa la misma llave, y un intento nuevo, otra. Una llave fija en tu código, o copiada de un ejemplo, sólo sirve para la primera sesión.

La API no expone hoy un código de error para dos solicitudes simultáneas con la misma llave mientras la primera se procesa. No envíes la misma llave en paralelo; reintenta solo después de un timeout o de una respuesta perdida.

MÉXICO

Verificar la firma (Node, Python y PHP)

Al terminar esta página tendrás una función que verifica la firma de un webhook entrante antes de procesarlo, en Node, en Python y en PHP.

Nunca proceses un evento sin verificar su firma

Cualquiera puede enviar un POST con la forma de un evento de Neat a tu URL de webhook. La firma es lo único que confirma que el cuerpo salió de Neat y que no fue alterado en tránsito. Verifica la firma antes de leer el contenido del evento, no solo antes de aplicarlo.

El algoritmo

`Neat-Signature` trae `t=<segundos unix>,v1=<hexadecimal>`. El valor `v1` es un HMAC-SHA256:

```
hex( HMAC_SHA256( key = webhook_secret, message = "<t>" + "." +
<cuerpo crudo> ) )
```

Puntos que suelen romper una implementación de verificación:

- Usa el **cuerpo crudo** (los bytes tal como llegaron), antes de parsear el JSON o cambiar espacios. Si tu framework ya parseó el body antes de que tu middleware de verificación lo vea, tendrás que capturar el buffer crudo aparte.
- El secreto es el texto UTF-8 que Neat te entregó, no lo decodifiques como Base64.

- Compara la firma con una comparación de tiempo constante (`timingSafeEqual` en Node, `hmac.compare_digest` en Python, `hash_equals` en PHP), nunca con === ni ==.
- Rechaza un `t` demasiado viejo o demasiado adelantado; Neat recomienda una tolerancia de 5 minutos. Esto exige que el reloj de tu servidor esté sincronizado.

Node.js

verify-neat-signature.js

```
import { createHmac, timingSafeEqual } from 'node:crypto';

const TOLERANCE_SECONDS = 5 * 60;

export function verifyNeatSignature(rawBody, header, secret,
nowSeconds = Math.floor(Date.now() / 1000)) {
  if (!Buffer.isBuffer(rawBody) || typeof header !== 'string' ||
typeof secret !== 'string' || !secret) {
    return false;
  }

  let timestamp;
  let signature;
  for (const part of header.split(',')) {
    const separator = part.indexOf('=');
    if (separator < 1) return false;
    const key = part.slice(0, separator).trim();
    const value = part.slice(separator + 1).trim();
    if (key === 't') timestamp = value;
    else if (key === 'v1') signature = value;
  }
  if (timestamp === undefined || signature === undefined) return
false;
  if (!/^\d{1,12}$/.test(timestamp) || !/^[a-f0-9]
{64}$/.test(signature)) return false;

  const timestampNumber = Number(timestamp);
  if (Math.abs(nowSeconds - timestampNumber) > TOLERANCE_SECONDS)
return false;

  const expected = createHmac('sha256',
secret).update(`${timestamp}.`, 'utf8').update(rawBody).digest();
  const provided = Buffer.from(signature, 'hex');
  return expected.length === provided.length &&
timingSafeEqual(expected, provided);
}
```

`rawBody` debe ser el `Buffer` sin procesar del cuerpo de la petición. En Express, por ejemplo, monta `express.raw({ type: 'application/json' })` en la ruta del webhook en vez del parser JSON por defecto, y parsea el JSON tú mismo después de verificar.

Python

verify_neat_signature.py

```python
import hashlib
import hmac
import re
import time

TOLERANCE_SECONDS = 5 * 60


def verify_neat_signature(
    raw_body: bytes, header: str, secret: str, now_seconds: int | None
= None
) -> bool:
    if not isinstance(raw_body, bytes) or not header or not secret:
        return False

    timestamp = None
    signature = None
    for part in header.split(","):
        if "=" not in part:
            return False
        key, _, value = part.partition("=")
        key, value = key.strip(), value.strip()
        if key == "t":
            timestamp = value
        elif key == "v1":
            signature = value

    if timestamp is None or signature is None:
        return False
    if not re.fullmatch(r"\d{1,12}", timestamp) or not re.fullmatch(r"
[a-f0-9]{64}", signature):
        return False

    now = now_seconds if now_seconds is not None else int(time.time())
    if abs(now - int(timestamp)) > TOLERANCE_SECONDS:
        return False

    message = f"{timestamp}.".encode("utf-8") + raw_body
```

```
    expected = hmac.new(secret.encode("utf-8"), message,
hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, signature)
```

En Flask, lee `request.get_data()` antes de acceder a `request.json` para conservar el cuerpo crudo; en Django, usa `request.body`.

PHP

verify_neat_signature.php

```php
<?php
const NEAT_SIGNATURE_TOLERANCE_SECONDS = 5 * 60;

function verify_neat_signature(string $rawBody, string $header, string
$secret, ?int $nowSeconds = null): bool
{
    if ($header === '' || $secret === '') {
        return false;
    }

    $timestamp = null;
    $signature = null;
    foreach (explode(',', $header) as $part) {
        $separator = strpos($part, '=');
        if ($separator === false || $separator < 1) {
            return false;
        }
        $key = trim(substr($part, 0, $separator));
        $value = trim(substr($part, $separator + 1));
        if ($key === 't') {
            $timestamp = $value;
        } elseif ($key === 'v1') {
            $signature = $value;
        }
    }

    if ($timestamp === null || $signature === null) {
        return false;
    }
    if (!preg_match('/^\d{1,12}$/D', $timestamp) || !preg_match('/^[a-
f0-9]{64}$/D', $signature)) {
        return false;
    }

    $now = $nowSeconds ?? time();
    if (abs($now - (int) $timestamp) >
NEAT_SIGNATURE_TOLERANCE_SECONDS) {
        return false;
    }
```

```
    $expected = hash_hmac('sha256', $timestamp . '.' . $rawBody,
$secret);
    return hash_equals($expected, $signature);
}
```

En el script que recibe el webhook, lee el cuerpo crudo de `php://input` y verifica antes de decodificar el JSON. La cabecera llega en `$_SERVER['HTTP_NEAT_SIGNATURE']`:

webhook.php

```
<?php
require __DIR__ . '/verify_neat_signature.php';

$rawBody = file_get_contents('php://input');
$header = $_SERVER['HTTP_NEAT_SIGNATURE'] ?? '';
$secret = getenv('NEAT_WEBHOOK_SECRET') ?: '';

if ($rawBody === false || !verify_neat_signature($rawBody, $header,
$secret)) {
    http_response_code(401);
    exit;
}

$event = json_decode($rawBody, true, 512, JSON_THROW_ON_ERROR);
```

No uses `$_POST`: con `Content-Type: application/json` viene vacío. En Laravel o Symfony, `$request->getContent()` devuelve el cuerpo crudo.

Después de verificar la firma

Verificar la firma solo confirma el origen y la integridad del cuerpo. Todavía debes:

1. Validar que `provider_id` sea el tuyo.
2. Confirmar que `Neat-Event-Id` coincide con `event_id` dentro del cuerpo.

3. Deduplicar por `event_id` antes de aplicar el efecto, ver Reintentos e idempotencia.

Rotación del secreto

El `webhook_secret` es uno solo por comercio y no tiene período de gracia: al rotarlo, el anterior deja de firmar en el momento del corte, a diferencia de tu API key. Coordina ese momento con tu contacto técnico en Neat y ten lista la actualización de tu lado para aplicarla en el corte.

Hasta que tu servidor tenga el secreto nuevo, los eventos firmados con él no van a verificar. Si les respondes `401`, como el ejemplo de PHP, o cualquier otro `4xx`, quedan como entregas fallidas y Neat no los reintenta. Por eso conviene aplicar el secreto nuevo apenas Neat confirma la rotación. Un cobro cuyo evento se perdió así sigue visible en `GET /v1/checkout-sessions/{sessionId}`.

MÉXICO

Archivo CSV

Al terminar esta página sabrás qué columnas trae el archivo de conciliación y cómo cruzarlo contra tus propios registros.

Descargar el archivo

```
GET /v1/reconciliation/payments.csv
Authorization: Bearer <tu API key>
```

Límite de tasa: 10 solicitudes por minuto por credencial. Devuelve `text/csv; charset=utf-8` con `Content-Disposition: attachment; filename="payments.csv"`. Incluye todos los pagos de tu comercio, sin filtro de fecha ni paginación, así que descargas el archivo completo cada vez.

Columnas

Una fila por pago, ordenadas por `created_at` ascendente. Separador `,`, fin de línea `\r\n`, codificación UTF-8.

Tres detalles del formato que conviene tener en cuenta al parsearlo:

- Los valores van entre comillas dobles, y unas comillas dentro del valor se duplican (`""`), como manda el RFC 4180.
- Un `null` se escribe como celda **vacía y sin comillas**, para distinguirlo de una cadena vacía.
- Un valor que empiece con =, +, – o @ lleva un apóstrofo delante (`'=A1`). Es la defensa estándar contra la inyección de fórmulas al abrir el archivo en una hoja de cálculo. Si procesas el CSV con código, quítalo antes de comparar.

Columna	Descripción
`payment_id`	Identificador del pago.
`session_id`	Sesión de checkout a la que pertenece.
`external_reference`	Tu referencia de la obligación.
`payment_version`	La misma `version` que ves en la API y en los webhooks.
`currency`	Siempre `MXN`.
`principal_amount_minor`	Monto de la obligación, en centavos.
`payer_fee_minor`, `payer_fee_tax_minor`	Comisión e IVA a cargo del pagador. `0` si los absorbe tu comercio.
`provider_fee_minor`, `provider_fee_tax_minor`	Comisión e IVA que absorbe tu comercio y se descuentan del depósito. `0` si los paga el pagador.
`charged_amount_minor`	Total cobrado. Vacía si el pago sigue `pending`; `0` si falló definitivamente.
`provider_net_amount_minor`	Monto neto para tu comercio. Vacía si sigue `pending`; `0` si falló definitivamente.
`refunded_amount_minor`	Total devuelto y confirmado hasta el momento de la descarga.
`payment_status`	Mismo valor que `payment.status` en la API, ver Estados y glosario.
`created_at`	Cuándo se creó el intento de cobro.
`paid_at`	Cuándo se confirmó. Vacía si nunca se confirmó.
`updated_at`	Última actualización de este pago, se mueve, por ejemplo, cuando se confirma una devolución.

Columna	Descripción
`category`	Qué pagó el pagador: `maintenance_fee`, `rent` o `rent_guarantee_insurance`, ver Categoría del cobro. Va después de `updated_at`, antes de `fee_bearer`, para no correr las columnas anteriores.
`fee_bearer`	Quién pagó la comisión: `payer` o `provider`, ver Importes y comisiones. Va al final, después de `category`.

Una celda vacía en `charged_amount_minor` o `provider_net_amount_minor` significa “todavía no hay un resultado definitivo”, no “cero”. No la conviertas en `0` al procesar el archivo.

Ejemplo

```
payment_id,session_id,external_reference,payment_version,currency,princi
"pay_3ac4efd9f068","cs_164596add1a3","CUOTA-2026-09-000123","2","MXN","1
"pay_4ae7fad5b7da","cs_1e1a4def4f6f","CUOTA-2026-10-000124","3","MXN","8
"pay_9c1d0b7e2a54","cs_2b7e93c0d1f8","CUOTA-2026-10-000125","2","MXN","1
```

La segunda fila es un pago que fue cobrado y luego cancelado por el pagador: `payment_status` pasó a `refunded`, `refunded_amount_minor` quedó igual al total cobrado, y `payment_version` avanzó a `3` (de `1` al reservarse, `2` al confirmarse el cobro, `3` al confirmarse la cancelación). La tercera es un cobro con la comisión absorbida por tu comercio: el pagador pagó `125000` y el neto a depositar es `120650`.

Cómo cruzarlo contra tus propios registros

1. Indexa tus obligaciones por `external_reference`; es el único campo del archivo que coincide con lo que tú enviaste al crear la sesión.

2. Actualiza por `payment_id` y `payment_version`, no sumes cada fila como un movimiento nuevo: si descargas el archivo más de una vez, un mismo pago puede aparecer con distinto `payment_status` en cada descarga.
3. Usa el webhook (ver Webhooks: eventos y firma) como la señal en tiempo real y este archivo para conciliar el corte del día. Ambos describen el mismo pago; si discrepan, `payment_version` y `updated_at` indican cuál es más reciente.

El archivo no tiene corte diario ni metadatos de período; refleja el estado de todos tus pagos en el momento de la descarga. Para conciliar por día, descárgalo a la misma hora cada día y conserva cada copia con su fecha.

⚠ El archivo no muestra una anulación en curso

No hay columna `cancellation_state`. Un pago cuya cancelación fue solicitada pero que el adquirente todavía no resuelve aparece **idéntico a un cobro limpio**: `payment_status` sigue en `succeeded` y `refunded_amount_minor` en `0`, porque el dinero efectivamente no se ha devuelto todavía.

No está mal, pero es incompleto para decidir. Si vas a liberar un servicio contra este archivo, consulta `GET /v1/checkout-sessions/{sessionId}` antes: la API sí trae `cancellation_state`, y ahí un pago con la cancelación en `requested` se distingue de uno sin ninguna. El webhook también lo lleva.

MÉXICO

Errores

Al terminar esta página sabrás la forma de un error, qué código HTTP y de negocio esperar en cada situación, y qué acción tomar en cada caso.

Forma de un error

422 Unprocessable Entity

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid return identifier.",
    "field": "return_id",
    "request_id": "req_4f9c2b7e1a8d4c3b9e6f0a1d2c3b4e5f"
  }
}
```

Campo	Presente	Descripción
`code`	Siempre	Código estable. Decide tu lógica sobre este campo, nunca sobre `message`.
`message`	Siempre	Texto en inglés para depuración humana. No lo muestres al pagador ni lo traduzcas para decidir un flujo.
`field`	Cuando aplica	El campo del cuerpo o la cabecera que causó el error.

Campo	Presente	Descripción
`request_id`	Siempre	Identificador único de la request (`req_` y 32 caracteres hexadecimales). Compártelo con Neat al pedir ayuda: con él encontramos tu request en nuestros registros. Nunca compartas tu API key, tu `webhook_secret` ni datos de tarjeta.

Catálogo, por endpoint

POST /v1/checkout-sessions

HTTP	Código	Cuándo	Qué hacer
401	`UNAUTHORIZED`	Falta el encabezado `Authorization`, no usa el prefijo `Bearer` o el token viene vacío.	Corrige el encabezado, ver Si recibes un 401.
401	`INVALID_API_KEY`	La API key no existe o fue revocada.	Revisa que mandas la API key y no el `webhook_secret`; si sigue, confirma la credencial vigente con Neat.
403	`PROVIDER_DISABLED`	Tu comercio está deshabilitado.	Contacta a Neat.
422	`VALIDATION_ERROR`	Un campo no cumple su formato, incluido un monto fuera de tus límites,	Corrige el campo indicado en `field`.

HTTP	Código	Cuándo	Qué hacer
		o `Idempotency-Key` ausente o inválida.	
422	`RETURN_NOT_REGISTERED`	El `return_id` no está registrado o no está activo.	Usa un `return_id` vigente, ver Alta de tu comercio.
409	`IDEMPOTENCY_CONFLICT`	Repetiste `Idempotency-Key` con un cuerpo distinto.	Usa el cuerpo original, o una llave nueva si de verdad es otra solicitud.
422	`BENEFICIARY_REQUIRED`	Tu comercio cobra por varios beneficiarios y la sesión no trae `beneficiary`.	Manda el beneficiario de ese cobro, ver Sesiones de checkout.
422	`BENEFICIARY_NOT_ALLOWED`	La sesión trae `beneficiary` y tu comercio cobra a la cuenta acordada con Neat.	Quita el campo, o pide a Neat habilitar el cobro por beneficiario.
422	`CATEGORY_NOT_ALLOWED`	La sesión trae una `category` que tu comercio no tiene habilitada.	Usa una de las categorías acordadas con Neat, o pide habilitar la nueva.

HTTP	Código	Cuándo	Qué hacer
422	`FEE_BEARER_NOT_ALLOWED`	La sesión trae un `fee_bearer` que tu comercio no tiene habilitado.	Omite el campo o usa la forma acordada con Neat, ver Quién paga la comisión.
422	`AMOUNT_BELOW_FEE`	Tu comercio absorbe la comisión y el monto no deja nada que depositar con alguna de tus tasas.	Sube el monto; con las tasas de tarjeta solo pasa con montos de unos pocos centavos.
409	`OBLIGATION_CONFLICT`	El `external_reference` ya existe con otro monto, moneda, categoría, concepto o beneficiario, aunque no se haya pagado.	Manda los mismos términos. Si la cuota cambió de verdad, usa una referencia nueva con versión, ver Sesiones de checkout.
409	`OBLIGATION_ALREADY_PAID`	Esa obligación ya tiene un pago confirmado.	Consulta el pago; no inicies otro cobro.
409	`PAYMENT_IN_PROGRESS`	Esa obligación tiene un cobro en curso.	Consulta el estado antes de reintentar.
429	`RATE_LIMITED`	Superaste 120 solicitudes por	Respeta `Retry-After`.

HTTP	Código	Cuándo	Qué hacer
		minuto con esa credencial.	

GET /v1/checkout-sessions/{sessionId}

HTTP	Código	Cuándo	Qué hacer
401	UNAUTHORIZED / INVALID_API_KEY	Igual que arriba.	Igual que arriba.
403	PROVIDER_DISABLED	Igual que arriba.	Igual que arriba.
404	RESOURCE_NOT_FOUND	El sessionId no existe, o pertenece a otro comercio.	Verifica el identificador; la API no distingue ambos casos entre sí.
429	RATE_LIMITED	Superaste 300 solicitudes por minuto con esa credencial.	Respeta Retry-After.

GET /v1/reconciliation/payments.csv

HTTP	Código	Cuándo	Qué hacer
401	UNAUTHORIZED / INVALID_API_KEY	Igual que arriba.	Igual que arriba.
403	PROVIDER_DISABLED	Igual que arriba.	Igual que arriba.
429	RATE_LIMITED	Superaste 10 solicitudes por minuto con esa credencial.	Respeta Retry-After.

Errores genéricos

HTTP	Código	Cuándo
400	INVALID_JSON	El cuerpo viene vacío o no es JSON válido.
400	BAD_REQUEST	El request no se pudo aceptar como vino; el caso típico es un cuerpo que no coincide con su Content-Length.
400	INVALID_PATH	El path no es una URL válida.
414	INVALID_PATH	Un parámetro del path excede el largo permitido.
404	ROUTE_NOT_FOUND	La ruta no existe. Revisa el método y el path.
413	PAYLOAD_TOO_LARGE	El cuerpo supera 16 KiB.
415	UNSUPPORTED_MEDIA_TYPE	Falta Content-Type: application/json o el cuerpo viene con otro tipo.
500	INTERNAL_ERROR	Cualquier falla no clasificada del lado de Neat.

Todos menos el último son errores de tu lado: corrige el request antes de reintentar, sin reintento automático. 429 RATE_LIMITED sí se reintenta, esperando lo que diga Retry-After, y 500 INTERNAL_ERROR también, con espera creciente.

Qué no vas a ver en v1

Estos códigos aparecían en un borrador anterior del contrato de integración y no existen en v1: IDEMPOTENCY_IN_PROGRESS, INSUFFICIENT_SCOPE. Si tu integración ya los maneja por haber seguido ese borrador, puedes dejar el manejo; hoy no los vas a recibir.

MÉXICO

Referencia de API

Referencia compacta de los tres endpoints de comercio de la API de Neat México v1. Para el flujo completo con contexto, ve a Sesiones de checkout, Redirección y regreso y Archivo CSV.

Todos los endpoints:

- Requieren `Authorization: Bearer <tu API key>`.
- Devuelven `Cache-Control: no-store`.
- Devuelven errores con la forma `{ "error": { "code", "message", "field"?, "request_id" } }`, ver Errores.

Método y ruta	Límite	Descripción
`POST /v1/checkout-sessions`	120/min	Crea una sesión de checkout.
`GET /v1/checkout-sessions/{sessionId}`	300/min	Consulta el estado de una sesión y su pago más reciente.
`GET /v1/reconciliation/payments.csv`	10/min	Descarga el CSV con todos los pagos de tu comercio.

`POST /v1/checkout-sessions`

Crea una obligación de cobro y una sesión de checkout para ella.

Cabeceras: `Authorization` (obligatoria), `Idempotency-Key` (obligatoria, 8–128 caracteres `[A-Za-z0-9._:-]`, una nueva por cada intento de cobro), `Content-Type: application/json`.

Cuerpo:

Campo	Tipo	Obligatorio
`external_reference`	string, 1–100 ASCII `[A-Za-z0-9._-]`	Sí
`concept`	string, 1–140 caracteres	Sí
`amount_minor`	integer positivo, centavos	Sí
`currency`	`"MXN"`	Sí
`return_id`	string, 1–50 ASCII `[A-Za-z0-9_-]`	Sí
`payer.email`	string, correo, ≤254	No
`payer.name`	string, 1–100	No
`metadata`	objeto, hasta 10 pares string:string	No
`beneficiary`	objeto `reference`, `legal_name`, `rfc`, `clabe` (string de 18 dígitos)	Según tu comercio
`fee_bearer`	`"payer"` o `"provider"`	Según tu comercio

`payer.email` decide quién puede pagar: si lo mandas, sólo esa dirección recibe el código y puede pagar la sesión. Reutilizar un `external_reference` con otro monto, moneda, concepto o beneficiario responde `409 OBLIGATION_CONFLICT` aunque no se haya pagado. Ver Sesiones de checkout.

cURL | Node

```
IDEMPOTENCY_KEY="checkout-$(uuidgen)"

curl -X POST "$NEAT_API_BASE/checkout-sessions" \
  -H "Authorization: Bearer $NEAT_API_KEY" \
  -H "Idempotency-Key: $IDEMPOTENCY_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "external_reference": "CUOTA-2026-09-000123",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "amount_minor": 125000,
    "currency": "MXN",
    "return_id": "principal"
  }'
```

Respuesta `201 Created`:

```
{
  "session_id": "cs_164596add1a3",
  "checkout_url":
"https://pago.neat.example/pagar/cs_164596add1a3.CQGg0fEWT5o…",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "status": "open",
  "fee_bearer": "payer"
}
```

Detalle completo de campos, reglas de `external_reference`, `beneficiary` y errores en Sesiones de checkout.

GET /v1/checkout-sessions/{sessionId}

Consulta una sesión y su pago más reciente.

cURL | Node

```
curl "$NEAT_API_BASE/checkout-sessions/cs_164596add1a3" \
  -H "Authorization: Bearer $NEAT_API_KEY"
```

Respuesta `200 OK`:

```
{
  "session_id": "cs_164596add1a3",
  "status": "completed",
  "external_reference": "CUOTA-2026-09-000123",
  "category": "maintenance_fee",
  "concept": "Mensualidad de mantenimiento, septiembre 2026",
  "amount_minor": 125000,
  "currency": "MXN",
  "fee_bearer": "payer",
  "expires_at": "2026-09-14T23:45:55.013Z",
  "payment": {
    "payment_id": "pay_3ac4efd9f068",
    "version": 2,
    "status": "succeeded",
    "currency": "MXN",
    "principal_amount_minor": 125000,
    "payer_fee_minor": 3750,
    "payer_fee_tax_minor": 600,
    "provider_fee_minor": 0,
    "provider_fee_tax_minor": 0,
    "charged_amount_minor": 129350,
    "provider_net_amount_minor": 125000,
    "refunded_amount_minor": 0,
    "fee_bearer": "payer",
    "cancellation_state": "none",
    "paid_at": "2026-09-14T23:17:17.894Z",
    "category": "maintenance_fee",
    "concept": "Mensualidad de mantenimiento, septiembre 2026",
    "payee_name": "Administradora Ejemplo, S.A. de C.V."
  }
}
```


`payment` es `null` antes del primer intento. Ver el significado de cada campo en Importes y comisiones y cada estado en Estados y glosario.

GET /v1/reconciliation/payments.csv

Descarga el CSV con todos los pagos de tu comercio, sin filtro de fecha.

cURL | Node

```
curl "$NEAT_API_BASE/reconciliation/payments.csv" \
  -H "Authorization: Bearer $NEAT_API_KEY" \
  -o payments.csv
```

Respuesta `200 OK`: `Content-Type: text/csv; charset=utf-8`. Columnas y ejemplo completo en Archivo CSV.

Especificación descargable

La forma exacta de cada request y respuesta también está disponible como OpenAPI 3.1, ver OpenAPI.

MÉXICO

OpenAPI

La especificación cubre los tres endpoints de comercio (`POST /v1/checkout-sessions`, `GET /v1/checkout-sessions/{sessionId}`, `GET /v1/reconciliation/payments.csv`) y el esquema del webhook saliente, en OpenAPI 3.1.

Descárgala o impórtala a tu cliente HTTP o generador de SDKs desde:

/openapi.yaml

Se actualiza junto con esta guía: un cambio en un endpoint de comercio actualiza la especificación en la misma publicación.

Qué no cubre

Los endpoints internos de Neat y los que usa el checkout hospedado para interactuar con el pagador no están en esta especificación; no son parte de la integración de tu servidor. Ver el alcance completo en Referencia de API.

MÉXICO

Pruebas y checklist

Son cinco pasos, y son el camino feliz completo: crear un cobro, pagarlo, enterarte por webhook, recibir de vuelta a quien pagó y encontrarlo en la conciliación. Si eso funciona, tu integración funciona.

No te pedimos que provoques errores. Los casos raros son responsabilidad nuestra: cada error trae un `code` estable y ese código está explicado en Errores, con qué hacer en cada caso. Si alguno no se entiende, dínoslo — es un problema de nuestra documentación, no tuyo.

Los cinco pasos

Cada paso tiene un identificador entre corchetes. Los usamos al revisar tus pruebas, así que si algo falta te vamos a decir "falta `webhook`" y no un párrafo.

- ☐ **`sesion`** — Crea una sesión con `POST /v1/checkout-sessions`, con tu `Idempotency-Key`, y guarda el `session_id` y el `checkout_url` que te devuelve.
- ☐ **`pago`** — Abre ese `checkout_url` en un navegador, entra con el correo de tu equipo, guarda la tarjeta de prueba aprobada y paga. Ver Alta de tu comercio para las tarjetas.
- ☐ **`webhook`** — Recibe `payment.succeeded` en tu URL de webhook, **verifica la firma** antes de hacer nada, y marca como pagada la obligación de `resource.external_reference`. Ver Verificar la firma.
- ☐ **`retorno`** — Comprueba que quien pagó vuelve a tu página de retorno, y que tu sitio **no** marca la obligación como pagada solo porque llegó ahí: el webhook es la fuente. Ver Redirección y regreso.
- ☐ **`conciliacion`** — Descarga `GET /v1/reconciliation/payments.csv` y encuentra ese pago por su `payment_id`. Ver Archivo CSV.

Dos cosas que revisamos y no dejan rastro

Estas no las podemos ver en tu tráfico, así que te las preguntamos:

- ☐ Tu API key y tu `webhook_secret` viven en el gestor de secretos de tu infraestructura, no en código fuente ni en variables de un CI compartido.
- ☐ Tienes un contacto técnico designado para incidentes y para rotar credenciales.

Cuando termines

Escríbenos con tu `provider_id`, la fecha en que hiciste las pruebas y el correo con el que entraste a pagar. Revisamos tu tráfico del ambiente de pruebas y confirmamos los cinco pasos. Si falta alguno te decimos cuál.

Después de eso te entregamos la API key de producción. Ver Alta de tu comercio.

MÉXICO

Changelog

Cambios que afectan a un comercio integrado con la API de Neat México. La API sigue en v1.

Mientras la API esté en pruebas, v1 puede cambiar y cada cambio se anuncia aquí: si ya estás integrado contra el ambiente de pruebas, esta página es lo que tienes que mirar. Cuando pases a producción te avisamos antes de cualquier cambio incompatible, y a partir de ahí no lo hacemos dentro de la versión que estés usando.

2026-10-30 · la tarjeta de prueba aprobada ahora es `4456528080389860`

Ningún cambio en la API. Desde el 22 de octubre el checkout verifica con 3D Secure cada tarjeta que se guarda, y en pruebas `5451951574925480` ya no pasa esa verificación: se rechaza al guardarla. Para un pago aprobado usa `4456528080389860` y, cuando el banco de prueba pida el código, escribe `1234`. `4349001210846432` tampoco se guarda: el procesador responde que la tarjeta no es compatible, así que ya no sirve para probar un cobro rechazado. Ver Alta de tu comercio.

2026-10-30 · un cobro lento termina con resultado en vez de quedar `pending`

Ningún cambio de forma en la API ni en lo que significa cada evento. Neat espera ahora hasta 20 segundos la respuesta del adquirente a un cobro con tarjeta; antes cortaba a los 9. Un cobro que el adquirente aprueba o rechaza en ese rango, como los de tarjetas con verificación 3D Secure en el ambiente de pruebas, termina en `payment.succeeded` o `payment.failed` en vez de `payment.pending`. Si el adquirente responde con un error o con algo ilegible, el

pago sigue quedando `payment.pending`. Vas a recibir menos `payment.pending`; los que lleguen se siguen resolviendo como dice la entrada del 23 de octubre.

2026-10-23 · un `payment.pending` ahora se resuelve

Ningún cambio de forma en la API. Un pago que quedó `pending` porque el adquirente no respondió sigue así hasta que alguien de Neat revisa el cargo en la consola del adquirente y registra el resultado; entonces recibes `payment.succeeded` o `payment.failed` para el mismo `payment_id`, con una `resource.version` mayor. La entrada del 15 de septiembre decía que un `payment.pending` no tenía continuación: ya no es así. Si tu integración trataba `pending` como final, trátalo como transitorio y deja que el evento con mayor `version` mande, ver Webhooks: eventos y firma. Si un cobro lleva horas en `pending`, escríbenos.

2026-10-22 · tu comercio puede absorber la comisión: `fee_bearer`

Ahora la comisión de Neat la puede pagar tu comercio en vez del pagador. En ese caso el pagador paga exactamente el monto de la obligación, el checkout no le muestra comisión, y la comisión con su IVA se descuenta del depósito. Es un acuerdo comercial que Neat habilita por comercio. **Si ya estás integrado no tienes que cambiar nada:** mientras no lo acuerdes, el pagador sigue pagando la comisión. Si Neat habilita que tu comercio la absorba como única forma, tus sesiones nuevas la absorben sin cambios, siempre que no mandes `fee_bearer`. Si habilita las dos, tienes que mandar `fee_bearer` en cada sesión.

- **Campo nuevo en las respuestas:** `fee_bearer` (`payer` o `provider`) en la respuesta de creación, en la sesión (`GET /v1/checkout-sessions/{sessionId}`), en el pago y en el `data` de cada webhook. Con `provider`, `provider_fee_minor` y `provider_fee_tax_minor` dejan de ser `0`,

`payer_fee_minor` y `payer_fee_tax_minor` quedan en `0`, y `provider_net_amount_minor` es el monto menos la comisión y su IVA. Los pagos anteriores traen `payer` en la API, el webhook y el CSV.

- **Columna nueva en el archivo de conciliación:** `fee_bearer`, al final, después de `category`. Si tu parser exige un número fijo de columnas, actualízalo.
- **Campo nuevo en la creación:** `fee_bearer`. Opcional si tu comercio tiene una sola forma habilitada; obligatorio si tiene las dos, y sin él la respuesta es `422 VALIDATION_ERROR` con `field: "fee_bearer"`.
- **Errores nuevos:** `422 FEE_BEARER_NOT_ALLOWED` si mandas una forma que tu comercio no tiene habilitada, y `422 AMOUNT_BELOW_FEE` si tu comercio absorbe la comisión y el monto no deja nada que depositar.
- **No es un término de la obligación:** queda fijo en cada sesión y reusar un `external_reference` con otro `fee_bearer` no responde `409`.

Detalle en Importes y comisiones y en Quién paga la comisión.

2026-10-22 · la sesión dice qué se está pagando: `category`

`POST /v1/checkout-sessions` acepta `category`, con tres valores: `maintenance_fee` (cuota de mantenimiento), `rent` (renta) y `rent_guarantee_insurance` (póliza de renta). Con ella cambian el texto del checkout y de los correos al pagador. **Si ya estás integrado no tienes que cambiar nada:** tu comercio tiene habilitada solo la cuota de mantenimiento, puedes seguir sin mandar el campo y tus reintentos con la misma `Idempotency-Key` devuelven la misma sesión.

- **Campo nuevo en las respuestas:** `category` en la sesión (`GET /v1/checkout-sessions/{sessionId}`), en el pago y en el `data` de cada webhook. En el pago puede ser `null` solo si su obligación ya no se puede leer.
- **Columna nueva en el archivo de conciliación:** `category`, al final. Si tu parser exige un número fijo de columnas, actualízalo.

- **Error nuevo:** `422 CATEGORY_NOT_ALLOWED` si mandas una categoría que tu comercio no tiene habilitada. Si tu comercio tiene varias y no mandas `category`, la respuesta es `422 VALIDATION_ERROR` con `field: "category"`.
- **Término de la obligación:** reusar un `external_reference` con otra categoría responde `409 OBLIGATION_CONFLICT`, igual que con otro monto.

Detalle en Categoría del cobro.

2026-10-19 · los errores de monto y de `Idempotency-Key` dicen qué corregir

Cambia el texto de cuatro respuestas `422 VALIDATION_ERROR` de `POST /v1/checkout-sessions`. El código, el `field` y el estado HTTP siguen iguales; si tu integración compara el `message` exacto, actualízala.

- **Monto bajo tu mínimo:** `Amount is below the provider minimum of 2000 cents (MXN 20.00).`, con el mínimo de tu comercio.
- **Monto sobre tu máximo:** `Amount exceeds the provider maximum of 5000000 cents (MXN 50,000.00).`, con el máximo de tu comercio.
- **Monto que no es un entero positivo:** `Amount must be a positive integer in cents: MXN 12,850.00 is 1285000.`
- **`Idempotency-Key` ausente o inválida:** `Idempotency-Key must be 8 to 128 characters from A-Z, a-z, 0-9, '.', '_', ':' and '-'.`

Esta documentación ahora muestra la conversión del monto: `amount_minor` es un entero en centavos, así que MXN 12,850.00 se envía como `1285000`, y `12850` son MXN 128.50. Está en el Quickstart, en Sesiones de checkout y en la especificación OpenAPI. Sesiones de checkout también dice ahora el formato de la `Idempotency-Key`, que antes sólo traía la especificación.

2026-10-14 · el checkout dice para qué cuenta es el cobro

Ningún cambio en la API. Quien entra con un correo distinto del `payer.email` de la sesión ya no ve «Liga de pago no disponible» en rojo: ve «Este pago es para otra cuenta», con el correo con que entró, la dirección del cobro enmascarada, a quién le paga y el monto, y, mientras el cobro sigue pendiente, que no se hizo ningún cargo. El botón lleva a pedir el código para la dirección del cobro. La respuesta sigue siendo `403 PAYER_MISMATCH`, y el Quickstart ya lo dice así. Cuando la sesión trae `payer.email`, la pantalla del código tampoco ofrece «Usar otro correo»: antes aparecía y volvía a la misma dirección.

2026-10-14 · cada error trae un `request_id` único

El `request_id` de las respuestas de error ahora es `req_` seguido de 32 caracteres hexadecimales y no se repite. Antes era un contador corto, como `req-7`, que se repetía entre servidores y no alcanzaba para encontrar tu request. Sigue siendo un texto: si lo guardas, no tienes que cambiar nada. Cuando nos escribas por un error, incluye ese valor: con él encontramos tu request en nuestros registros.

2026-10-13 · el checkout pide menos datos al pagador

Ningún cambio en la API. Antes de guardar la primera tarjeta, el checkout ya no pide fecha de nacimiento, y a quien entró con Google le pide sólo el RFC, porque el nombre ya viene en su cuenta. Cómo pruebas ya lo dice así.

2026-10-13 · correcciones a esta documentación

Ningún cambio en la API. Corregimos lo que llevaba a un primer intento a fallar copiando los ejemplos.

- **Si tu comercio cobra por beneficiario, el Quickstart ahora lo dice.** Cada creación tiene que incluir `beneficiary` o responde `422 BENEFICIARY_REQUIRED`. Hay ejemplos completos en cURL, Node y PHP, y la referencia de la API ahora lista el campo.
- **Los ejemplos ya no mandan `payer.email`, y el Quickstart explica qué hace.** Traían un correo de ejemplo, y una sesión creada así sólo la puede pagar ese buzón: el checkout no pregunta el correo, envía el código ahí y responde `403 PAYER_MISMATCH` a cualquier otra cuenta. Una sesión con el correo equivocado no se corrige; se crea otra con una `Idempotency-Key` nueva.
- **`409 OBLIGATION_CONFLICT`, explicado.** Reutilizar un `external_reference` con otro monto, moneda, concepto o beneficiario responde `409` aunque la obligación no se haya pagado. Ahora el Quickstart y Sesiones de checkout dicen por qué y qué hacer si la cuota cambia antes de pagarse.
- **La `Idempotency-Key` se genera por intento** en todos los ejemplos. Una llave fija copiada de esta documentación responde `409 IDEMPOTENCY_CONFLICT` con otro cuerpo, o devuelve la sesión anterior, aunque haya vencido, con el mismo.
- **Cómo reconocer cada credencial y qué revisar ante un `401`**, en Autenticación y ambientes: la API key empieza con `nmx_test_` o `nmx_live_` y mide 52 caracteres; el `webhook_secret` (`whsec_…`) nunca va en `Authorization`.
- **Verificar la firma en PHP**, con `php://input` y `hash_equals`.
- **Cómo pruebas** dice que la primera vez el checkout te pide nombre, RFC y fecha de nacimiento antes de guardar la tarjeta, y que en pruebas puedes usar el RFC genérico del SAT, `XAXX010101000`. En `beneficiary.rfc` los RFC genéricos responden `422`.
- **El `webhook_secret` rota de inmediato, sin período de gracia.** El alta prometía que el secreto anterior y el nuevo convivían un tiempo; eso sólo vale para la API key.
- **Un WAF delante de tu webhook puede dejarte sin eventos.** Un bloqueo o desafío responde `3xx` o `4xx`, y esas entregas no se reintentan. El alta explica cómo eximir la ruta en Cloudflare.

- **Para cambiar una cuota con una referencia nueva, espera a que venza la sesión anterior** y confirma que ninguna sesión de esa referencia tiene un pago `succeeded` ni `pending`. Que el pagador ya no tenga la liga no basta: si paga las dos, son dos cobros.
- **La guía en PDF de `/descargas/` se regeneró** con estas correcciones. La anterior era del 15 de septiembre y todavía traía los ejemplos con `payer.email` y la llave fija.
- **Cuándo recibe el dinero el condominio**, en Sesiones de checkout: qué confirma `payment.succeeded` y qué avisos existen hoy.

2026-09-16

`GET /health` responde en `api-beta.neatpagos.mx` y `api.neatpagos.mx`, sin credencial, con `200` y un JSON corto. Antes ese host respondía una página `404` fuera de `/v1`. Sirve para comprobar que tu servidor llega al ambiente correcto; no es parte del contrato de `v1` y su cuerpo puede cambiar sin aviso. Ningún endpoint de `/v1` cambió.

2026-09-15 · correcciones a esta documentación

Ningún cambio en la API: lo que cambió es lo que esta documentación decía de ella. Si integraste contra alguna de estas páginas, vale la pena releerlas.

- **`payer.email` no es un dato cosmético, es control de acceso.** Si lo mandas, sólo una identidad con ese correo exacto puede abrir y pagar la sesión; cualquier otra recibe `403 PAYER_MISMATCH`. Un correo mal escrito deja al pagador fuera de su propia deuda.
- **La liga de checkout no es de un solo uso**, como decía la tabla de respuesta. Se puede abrir las veces que haga falta hasta `expires_at` o hasta que la sesión deje de estar `open`.
- **Una redirección en tu URL de webhook es un fallo definitivo.** Neat no sigue 3xx, así que un `http://` que redirige a `https://`, o una barra final

de más o de menos, hace que no recibas ningún evento — y sin error visible de tu lado.

- **Un `payment.pending` no tiene continuación automática.** No esperes un segundo evento que lo resuelva; consulta la sesión, y escríbenos si un cobro lleva horas ahí.
- **`partially_refunded` está reservado y `v1` no lo emite:** toda cancelación devuelve el cargo completo.
- **`metadata` es sólo de escritura** —no vuelve en ninguna respuesta, webhook ni en el CSV— y al reutilizar un `external_reference` se conserva el de la primera sesión.
- **Reutilizar un `external_reference` no cierra la sesión anterior**: su enlace sigue vivo hasta su propio vencimiento.
- **El archivo de conciliación no distingue una anulación en curso.** Un pago con la cancelación solicitada se ve idéntico a uno limpio; para eso está `cancellation_state` en la API.
- La especificación OpenAPI ahora dice que el mínimo y el máximo de `amount_minor` los fija tu comercio, y lleva el patrón de las claves de `metadata`.

2026-09-15

El cuerpo de cada webhook trae dos campos nuevos dentro de `resource`: **`session_id`**, la sesión de checkout del pago, y **`external_reference`**, tu propia referencia de la obligación tal como la mandaste al crearla. Es un cambio aditivo: nada cambió de lugar ni de nombre, y si ya estabas procesando eventos no tienes que tocar nada.

Vale la pena saber que existen. Antes, para saber a qué cobro tuyo correspondía un `payment.succeeded`, había que guardar la relación entre tu referencia y el `payment_id`, o bajar el archivo de conciliación. Ahora el evento se basta solo.

La especificación OpenAPI (`openapi.yaml`) servía desde `neat-mx-checkout-beta.web.app`, que es el host del checkout y no el de la API. Ahora declara los

dos hosts correctos, `api-beta.neatpagos.mx` para pruebas y `api.neatpagos.mx` para producción. Si generaste tu cliente a partir del archivo anterior, vuelve a generarlo o corrige la URL base; no cambió ningún endpoint ni ningún campo.

2026-09-14

Publicación inicial de esta documentación para la API v1: alta de comercio, autenticación, creación y consulta de sesiones de checkout, redirección de regreso, estados, importes y comisiones, webhooks (eventos, firma, reintentos), archivo de conciliación, catálogo de errores y especificación OpenAPI.

El objeto `payment`, tanto en la sesión de checkout como en el cuerpo de cada webhook, incluye `concept` y `payee_name`.