Centro de Documentación Pago46

Documentación técnica de la API Core · v2026.6.0

Contenido

Bienvenido a Pago46

Pago46 es la infraestructura de pagos que digitaliza el efectivo en Latinoamérica. Nuestra plataforma cierra la brecha entre el mundo online y el dinero físico, permitiendo que cualquier empresa procese **depósitos (Pay-In)** y **retiros (Pay-Out)** en tiempo real a través de una única integración.

Al conectarte a nuestra **API Core**, accedes instantáneamente a una red híbrida que combina:

1. **Cadenas comerciales y bancarias:** Partners consolidados como *CajaVecina*, *RapiPago*, *ProvinciaNet*, entre otros.
2. **Red de Socios46:** Nuestra fuerza de agentes móviles ("Uber del efectivo") que recolectan o entregan dinero a domicilio o en puntos geolocalizados.

Ecosistema y Actores

La API está diseñada para orquestar transacciones entre tres actores principales. Identifica cuál es tu rol para navegar esta documentación:

Actor	Rol en la API	Descripción
Merchant (Comercio)	`Initiator`	Empresas online, E-commerce, Wallets o Fintechs que necesitan **cobrar** a sus usuarios o **dispersar** fondos (préstamos, remesas).
Payment Provider (Proveedor)	`Fulfiller`	Redes físicas (Farmacias, Bancos, Kioscos) que utilizan nuestra API para **procesar** las órdenes creadas por los Merchants, ganando comisiones por tráfico.
Consumidor	`End-User`	El usuario final que entrega o recibe el efectivo. Interactúa con el mapa de Pago46 o con el punto de venta del Proveedor.

Servicios Principales

Nuestra nueva API Core unifica los flujos operativos en endpoints polimórficos, permitiendo manejar operaciones en **moneda local** o **extranjera** con la misma semántica.

1. Pay-In (Pagos)

Permite a los usuarios pagar compras online con efectivo.

- **Para Merchants:** Generas una orden y obtienes un QR/Código. Recibes el dinero conciliado digitalmente.
- **Para Providers:** Validas el código del usuario, recibes el efectivo y notificas a Pago46 para liberar el servicio.

2. Pay-Out (Retiros)

Permite a los usuarios retirar saldo de una billetera digital o recibir un préstamo en efectivo.

- **Para Merchants:** Creas una orden de retiro para tu usuario.
- **Para Providers:** Verificas la orden, bloqueas la transacción (Start Payment) y entregas el efectivo de tu caja.

Flujo de Integración

La arquitectura de Pago46 es **Server-to-Server**. Para garantizar la seguridad, utilizamos un esquema de firma digital robusto.

1. **Autenticación:** Implementa el estándar **HMAC SHA-256** para firmar todas tus peticiones.
2. **Creación de Orden:** El Merchant inicia la intención de pago o retiro (`/orders`).
3. **Interacción Física:** El usuario acude a un punto (Provider o Socio46) o solicita un servicio a domicilio.
4. **Confirmación:** El Provider notifica a la API que el dinero cambió de manos.
5. **Webhooks:** Pago46 notifica al Merchant en tiempo real que la transacción fue exitosa.

> **¿ERES NUEVO AQUÍ?**
>
> Recomendamos comenzar leyendo la sección de **Autenticación** antes de intentar cualquier llamada a la API. Es el requisito fundamental para obtener respuesta del servidor.

> **MONITOREO OPERATIVO**
>
> Consulta la **página de Estado del Servicio** para validar incidentes, mantenimientos y suscribirte a alertas por correo o Slack.

¿Por dónde empiezo?

Selecciona la guía que mejor se adapte a tu necesidad de integración:

🛍️ Soy un Comercio

Quiero procesar pagos o enviar dinero a mis usuarios.

- Integrar cobros (Pay-In)
- Integrar retiros (Pay-Out)
- Integrar retiros en moneda extranjera

🏪 Soy un Proveedor

Tengo puntos físicos y quiero procesar transacciones de Pago46.

- Procesar pagos
- Procesar retiros

¿Listo para comenzar? Revisa la sección de **Países** para conocer las disponibilidades operativas.

Países

Cobertura regional

Con **una sola integración** te conectas a la red de pagos en efectivo más extensa de Latinoamérica. Pago46 opera de forma nativa o a través de *Redes de Pago* aliadas, garantizando una liquidación unificada y SLAs homogéneos en todos los mercados.

Países Activos y Monedas Locales (Fiat)

Actualmente, procesamos operaciones en moneda local (Fiat) en los siguientes territorios. Asegúrate de utilizar el `currency_code` correcto en tus solicitudes de API para cada jurisdicción.

País	Moneda Aceptada	Código ISO (Currency)
🇦🇷 **Argentina**	Peso Argentino	`ARS`
🇨🇱 **Chile**	Peso Chileno	`CLP`
🇪🇨 **Ecuador**	Dólar Estadounidense	`USD`
🇲🇽 **México**	Peso Mexicano	`MXN`
🇵🇪 **Perú**	Sol	`PEN`
🇬🇹 **Guatemala**	Quetzal	`GTQ`

> **Configuración por País**
>
> Se puede realizar una configuración específica de comportamiento y funcionalidad para cada país, permitiendo definir distintos flujos de pago, tarifas y reglas de negocio adaptadas a las normativas locales.

Moneda Extranjera (Foreign Currency)

Ofrecemos la funcionalidad de Moneda Extranjera para procesar transacciones con conversión de divisas. Esto permite a tus usuarios pagar o recibir fondos en una moneda diferente a la local, facilitando la expansión internacional sin necesidad de múltiples integraciones.

Divisa de Origen → Divisa Destino	Descripción	Países Disponibles
USDC → MXN	Permite a los usuarios enviar **USDC** vía **Solana**, **Monad** o **Stellar** y **recibir pesos mexicanos en puntos físicos inmediatamente**.	🇲🇽 México

---

Próximos pasos

Selecciona la guía que mejor se adapte a tu necesidad de integración:

🛍️ Soy un Comercio

Quiero procesar pagos o enviar dinero a mis usuarios.

- Integrar cobros (Pay-In)
- Integrar retiros (Pay-Out)

🏪 Soy un Proveedor

Tengo puntos físicos y quiero procesar transacciones de Pago46.

- Procesar pagos
- Procesar retiros

¿Dudas sobre la expansión a tu país? Contacta a mcontreras@pago46.com

Ambientes

Pago46 opera en dos ambientes distintos: **Sandbox** para pruebas y **Producción** para transacciones reales. Cada ambiente tiene sus propias características y requisitos de configuración.

Flujo tradicional de integración

1. Identifica si eres un **Comercio (Merchant)** o un **Proveedor de Pago (Payment Provider)**.
2. Identifica el flujo que deseas integrar: **Cobros (Pay-In)** o **Retiros (Pay-Out)**.
3. Envía un correo a `contacto@pago46.com` indicando tu rol (Comercio o Proveedor de Pago), país de operación y flujo a integrar.
4. Recibirás credenciales para el ambiente **Sandbox** y acceso a la documentación.
5. Realiza las pruebas necesarias en **Sandbox**.
6. Cuando estés listo para producción, solicita las credenciales de producción a tu ejecutivo comercial.

Pruebas (Sandbox)

Servicio Pago46	URL Base	Notas
API Principal	`https://api.dev.pago46.io/api/v1`	Punto de integración API REST HTTPS con firma HMAC.
Aplicación Web de pago	`https://checkout.dev.pago46.io`	Página alojada para que consumidores completen flujos de pago.
Aplicación Móvil	-	No contamos con ambiente sandbox para la aplicación móvil Android ni iOS

*Los montos procesados en sandbox **no** generan movimientos reales de dinero.*

Producción

Servicio Pago46	URL Base	Notas
API Principal	`https://api.prd.pago46.io/api/v1`	Punto de integración API REST HTTPS con firma HMAC.
Aplicación Web de pago	`https://checkout.prd.pago46.io`	Página alojada para que consumidores completen flujos de pago.
Aplicación Móvil	Link a Apple AppStore ó Link a Google Play Store	Para utilizar la aplicación es necesario estar registrado como Socio46

Monitoreo de disponibilidad

Para visibilidad operativa de producción, revisa la página de Estado del Servicio, donde encontrarás incidentes activos, mantenimientos programados y opciones de suscripción por correo y Slack.

> **CREDENCIALES DISTINTAS**
>
> Las llaves sandbox y producción **no son intercambiables**. Solicítalas a tu ejecutivo comercial.

Autenticación

La nueva API Core de Pago46 utiliza un esquema de seguridad basado en **HMAC SHA-256** para garantizar la integridad y autenticidad de cada transacción.

Este mecanismo asegura que quien envía la petición posee las credenciales correctas y que el mensaje no ha sido alterado en el camino ni es una repetición de una petición antigua.

Cabeceras Obligatorias

Cada petición HTTP que realices a nuestra API debe incluir las siguientes cabeceras (headers):

Cabecera	Tipo	Descripción
`Provider-Key`	String	Tu identificador único de proveedor (API Key).
`Message-Date`	Timestamp	Timestamp Unix (segundos o milisegundos flotantes).
`Message-Hash`	String	El resultado hexadecimal de la firma HMAC calculada.

> **IMPORTANTE**
>
> El servidor validará que la diferencia entre el `Message-Date` enviado y la hora actual del servidor no sea mayor a **24 horas**. Si el tiempo excede este rango, la petición será rechazada para prevenir ataques de repetición (*replay attacks*).

Algoritmo de Firma

Para generar el `Message-Hash` válido, debes seguir estrictamente el siguiente algoritmo de construcción de la cadena de firma.

1. Construcción de la Cadena (String to Sign)

A diferencia de versiones anteriores, este flujo **no requiere** ordenar parámetros alfabéticamente. La cadena se construye concatenando los siguientes valores separados por dos puntos (`:`):

1. **Provider Key**
2. **Message Date** (El mismo valor enviado en el header)
3. **Método HTTP** (Ej: `POST`, `GET`)
4. **Path** (La ruta del recurso, ej: `/api/v1/payments`)
5. **Body** (El cuerpo crudo de la petición en UTF-8)

Formato:

```
PROVIDER_KEY:MESSAGE_DATE:METHOD:PATH:BODY
```

> **NOTA SOBRE EL BODY**

Si la petición es `GET` y no tiene cuerpo, el valor de `BODY` debe ser una cadena vacía. Si es un `POST` con JSON, asegúrate de usar el string JSON exacto que enviarás en el request, sin espacios adicionales ni modificaciones.

2. Generación del Hash

Una vez construida la cadena, debes firmarla utilizando:

- **Algoritmo:** HMAC SHA-256
- **Secret:** Tu `Provider Secret` (proporcionado por Pago46)
- **Mensaje:** La cadena construida en el paso 1.
- **Output:** Hexdigest (cadena hexadecimal).

Ejemplos de Implementación

A continuación, presentamos cómo generar esta firma en diferentes lenguajes.

Python | **Node.js** | **Go**

```python
import hmac
import hashlib
import time
import requests
import json

def send_authenticated_request(provider_key, provider_secret, method, path, body_dict=None):
    # 1. Preparar datos
    host = "https://api.dev.pago46.io"
    timestamp = str(time.time()) # Timestamp actual (float en string)

    # Si hay body, lo convertimos a string JSON, si no, es vacío
    body_str = json.dumps(body_dict) if body_dict else ""

    # 2. Construir la cadena de firma (IMPORTANTE: Separador es ":")
    # Formato: KEY:DATE:METHOD:PATH:BODY
    string_to_sign = f"{provider_key}:{timestamp}:{method}:{path}:{body_str}"

    # 3. Calcular HMAC SHA-256
    calculated_hmac = hmac.new(
        provider_secret.encode("utf-8"),
        string_to_sign.encode("utf-8"),
        hashlib.sha256
    ).hexdigest()

    # 4. Enviar Petición
    headers = {
        "Provider-Key": provider_key,
        "Message-Date": timestamp,
        "Message-Hash": calculated_hmac,
        "Content-Type": "application/json"
    }

    response = requests.request(
        method=method,
        url=f"{host}{path}",
        headers=headers,
        data=body_str
    )

    return response

# Uso
response = send_authenticated_request(
    provider_key="PK_12345",
    provider_secret="SECRET_XYZ",
    method="POST",
    path="/api/v1/payments/",
```

```
    body_dict={"amount": 100, "currency": "CLP"}
)
print(response.status_code)
```

Errores Comunes

Código HTTP	Mensaje	Causa Probable
`403 Forbidden`	`Invalid authentication credentials`	El header `Provider-Key` no existe o no se encuentra en base de datos.
`403 Forbidden`	`Possible replay attack`	El `Message-Date` tiene más de 24 horas de diferencia con el servidor.
`403 Forbidden`	`Hash mismatch`	La firma no coincide. Verifica que el separador sea `:`, que el body sea exacto y que el path sea correcto.

OpenAPI

El contrato OpenAPI de Pago46 es la referencia oficial para integrar nuestros servicios REST. Define endpoints, métodos, parámetros, payloads, respuestas y esquemas de error, ayudándote a implementar una integración más rápida y con menos retrabajo.

Recursos disponibles

- **Contrato OpenAPI (YAML)**: Descargar especificación
- **Swagger UI**: Explorar y probar endpoints
- **Redoc UI**: Navegar la documentación

Uso recomendado para integradores

1. **Empieza por el contrato** para entender recursos, operaciones y modelos.
2. **Valida autenticación y headers** antes de consumir endpoints de negocio.
3. **Usa Swagger UI o Redoc** para revisar ejemplos y comportamientos esperados.
4. **Genera artefactos técnicos** (clientes, mocks o pruebas) desde el YAML cuando aplique a tu stack.
5. **Prueba en Sandbox** y promueve a producción solo después de validar flujos críticos y manejo de errores.

Buenas prácticas

- Toma OpenAPI como fuente de verdad técnica durante todo el ciclo de integración.
- Implementa validaciones de esquema para prevenir errores de formato y tipos.
- Estandariza el manejo de errores con base en los códigos y respuestas documentados.
- Revisa periódicamente la documentación para identificar ajustes y nuevas capacidades.

Manejo de errores comunes

Formato de Respuesta de Error

El formato de respuesta de error es el siguiente:

```
{
  "type": "validation_error",
  "errors": [
    {
      "code": "required",
      "detail": "This field is required.",
      "attr": "name"
    }
  ]
}
```

- `type`: puede ser `validation_error`, `client_error` o `server_error`
- `code`: cadena corta que describe el error. Puede ser utilizada por los consumidores de la API para personalizar su comportamiento.
- `detail`: texto descriptivo y comprensible para el usuario sobre el error. Usualmente en inglés, pero puede ser traducido.
- `attr`: puede ser `null` cuando el error no está asociado a un campo específico; cuando aplica, contiene el nombre del campo, por ejemplo en errores de tipo `validation_error`

Errores Comunes

Algunos errores pueden aparecer en la mayoría de nuestros endpoints. En las siguientes secciones describimos los más comunes. Para más detalles, por favor consulta los recursos enlazados en OpenAPI.

401 No Autorizado

Estos errores se retornan con el código de estado 401 siempre que la autenticación falla o se realiza una solicitud a un endpoint sin proporcionar información de autenticación. Estos son los 2 posibles errores que pueden retornar:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "authentication_failed",
            "detail": "Incorrect authentication credentials.",
            "attr": null
        }
    ]
}
```

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "not_authenticated",
            "detail": "Authentication credentials were not provided.",
            "attr": null
        }
    ]
```

```
}
```

405 Método No Permitido

Este error se retorna cuando se llama a un endpoint usando un método HTTP inesperado. Por ejemplo, si para actualizar un usuario se requiere POST y en su lugar se usa PATCH, se retorna este error. Se ve así:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "method_not_allowed",
            "detail": "Method \"PATCH\" not allowed.",
            "attr": null
        }
    ]
}
```

406 No Aceptable

Este error se retorna si se envía el header `Accept` y contiene un valor distinto a `application/json`. La respuesta sería:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "not_acceptable",
            "detail": "Could not satisfy the request Accept header.",
            "attr": null
        }
    ]
}
```

415 Tipo de Medio No Soportado

Este error se retorna cuando el tipo de contenido de la solicitud no es JSON. La respuesta sería:

```
{
    "type": "client_error",
    "errors": [
        {
            "code": "unsupported_media_type",
            "detail": "Unsupported media type \"application/xml\" in request.",
            "attr": null
        }
    ]
}
```

500 Error Interno del Servidor

Este error se retorna cuando el servidor de la API encuentra un error inesperado. La respuesta sería:

```
{
    "type": "server_error",
    "errors": [
        {
            "code": "error",
            "detail": "A server error occurred.",
            "attr": null
        }
```

```
    ]
}
```

Integración para Comercios

Como comercio en Pago46, accedes a una red híbrida de puntos físicos donde tus usuarios pueden pagar o recibir efectivo. Tú gestionas todo digitalmente mediante nuestra API.

Servicios Disponibles

Pay-In (Cobros)

Permite que tus usuarios paguen con efectivo en cualquier punto de la red.

Casos de uso: Recargas de billeteras, suscripciones, e-commerce, servicios.

→ **Ver documentación de Pay-In**

Pay-Out (Dispersiones)

Permite que tus usuarios retiren efectivo o reciban pagos en puntos físicos.

Casos de uso: Retiros de saldo, pagos a trabajadores, remesas, reembolsos.

→ **Ver documentación de Pay-Out**

Moneda Extranjera

Operaciones con conversión de divisas.

→ **Ver documentación de Moneda Extranjera**

Checkout

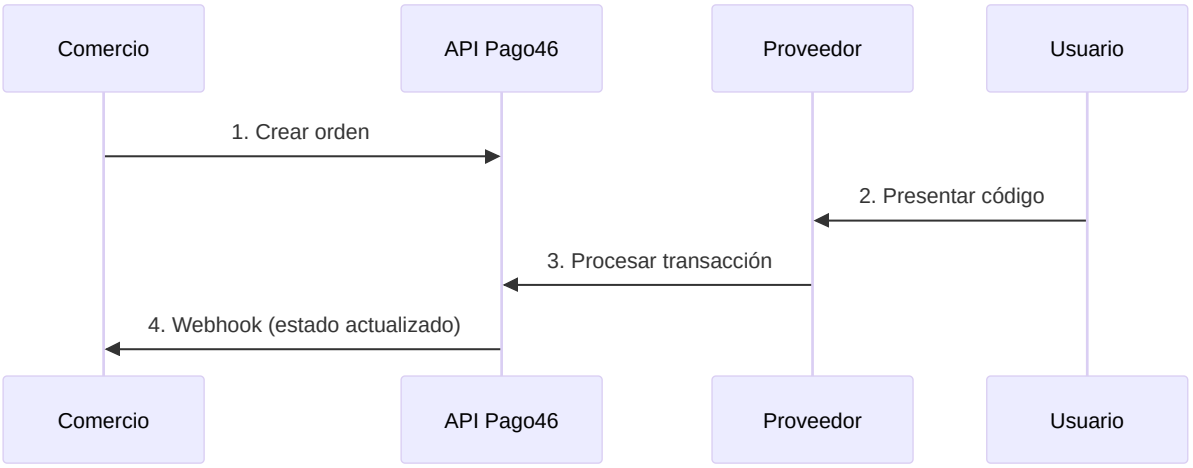
Integra la aplicación web de Checkout en iframe para flujos Pay-In y Pay-Out.

→ **Ver documentación de Checkout**

> **CHECKOUT DE PAGO46**
>
> Las órdenes contienen el campo `redirect_url`, que corresponde al link de Checkout para que tu usuario continúe el flujo.
>
> - Sandbox: `https://checkout.dev.pago46.io`
> - Producción: `https://checkout.prd.pago46.io`
> - Formato por orden: `https://checkout.{dev|prd}.pago46.io/{UUID}`
>
> También puedes entrar al home de Checkout (sin UUID) para capturar la orden manualmente durante pruebas e integración de iframe.

Flujo de Integración

La arquitectura es Server-to-Server con autenticación HMAC SHA-256.



Pasos de Integración

1. **Obtén tus credenciales** - Recibe tu `Merchant-Key` y `Merchant-Secret`
2. **Implementa autenticación HMAC** - Ver guía
3. **Crea órdenes** - Usa `POST /merchants/orders/pay-in` o `/pay-out`
4. **Recibe webhooks** - Configura tu endpoint HTTPS para notificaciones

REQUISITO FUNDAMENTAL

Todas las peticiones requieren autenticación HMAC. Revisa la documentación de autenticación antes de comenzar.

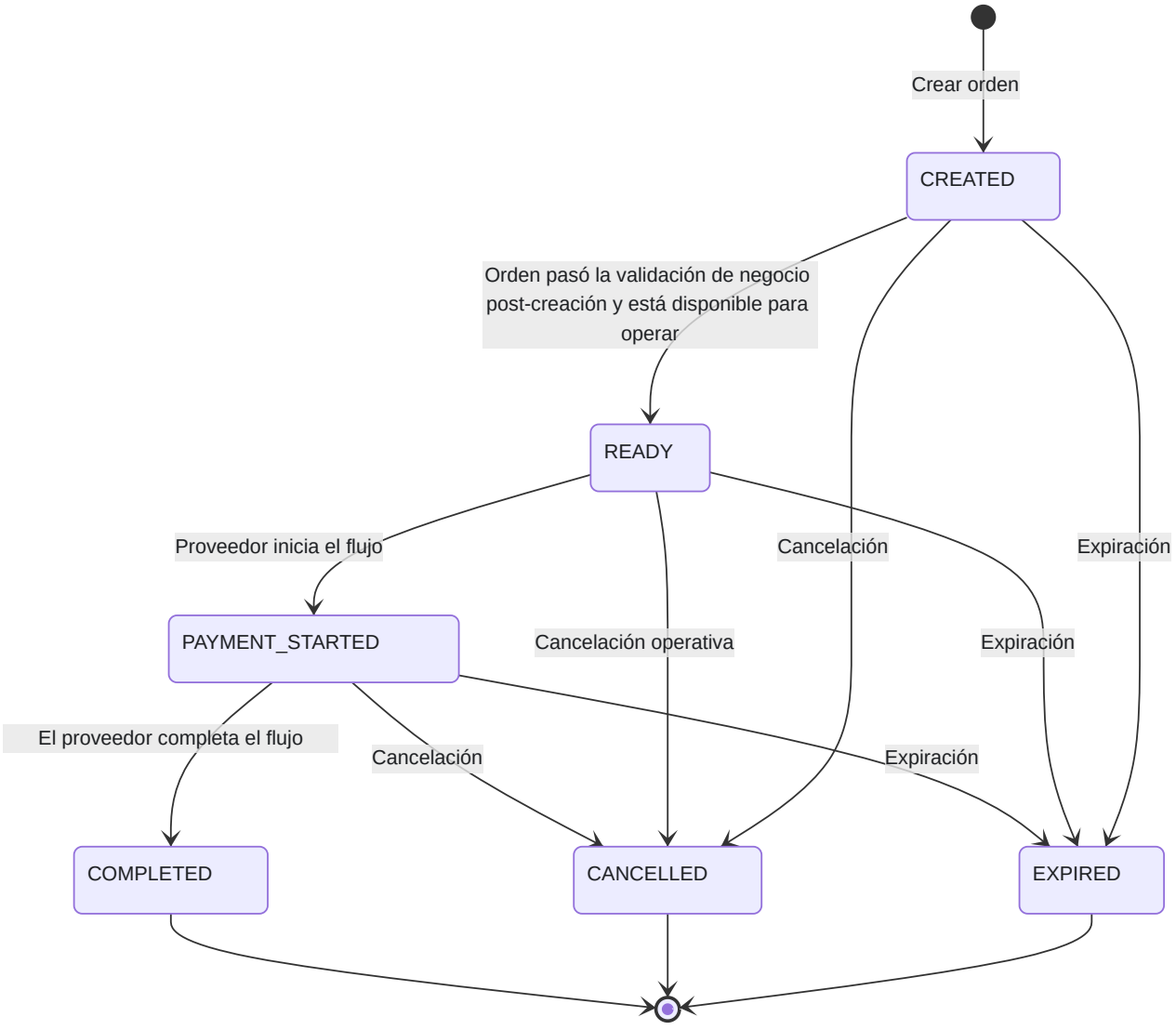
Ambientes

Ambiente	URL	Uso
Sandbox	`api.dev.pago46.io`	Desarrollo y pruebas
Producción	`api.prd.pago46.io`	Operaciones reales

→ Ver detalles de ambientes

Ciclo de Vida de Órdenes

Las órdenes transicionan por estados que reportamos vía webhooks:



Recibirás una notificación HTTP POST en tu `notify_url` cada vez que la orden se actualice (por ejemplo, cuando cambie de estado o se agregue información adicional), por lo que puedes recibir múltiples notificaciones con el mismo `status`.

Cobertura

Pago46 opera en múltiples países de Latinoamérica. Ver lista completa.

Pagos (Pay In)

Este módulo permite a los Comercios (Merchants) crear órdenes de pago (pay-in) para recibir fondos de sus usuarios o clientes. Las órdenes creadas quedan disponibles para ser procesadas por los Proveedores de Pago integrados en la red de Pago46.

ENDPOINT BASE

Todas las rutas descritas a continuación son relativas a la URL base de la API: `/api/v1`

CHECKOUT DE PAGO46

Las órdenes contienen el campo `redirect_url`, que corresponde al link de Checkout para continuar el flujo del usuario.

- Sandbox: `https://checkout.dev.pago46.io`
- Producción: `https://checkout.prd.pago46.io`
- Formato por orden: `https://checkout.{dev|prd}.pago46.io/{UUID}`

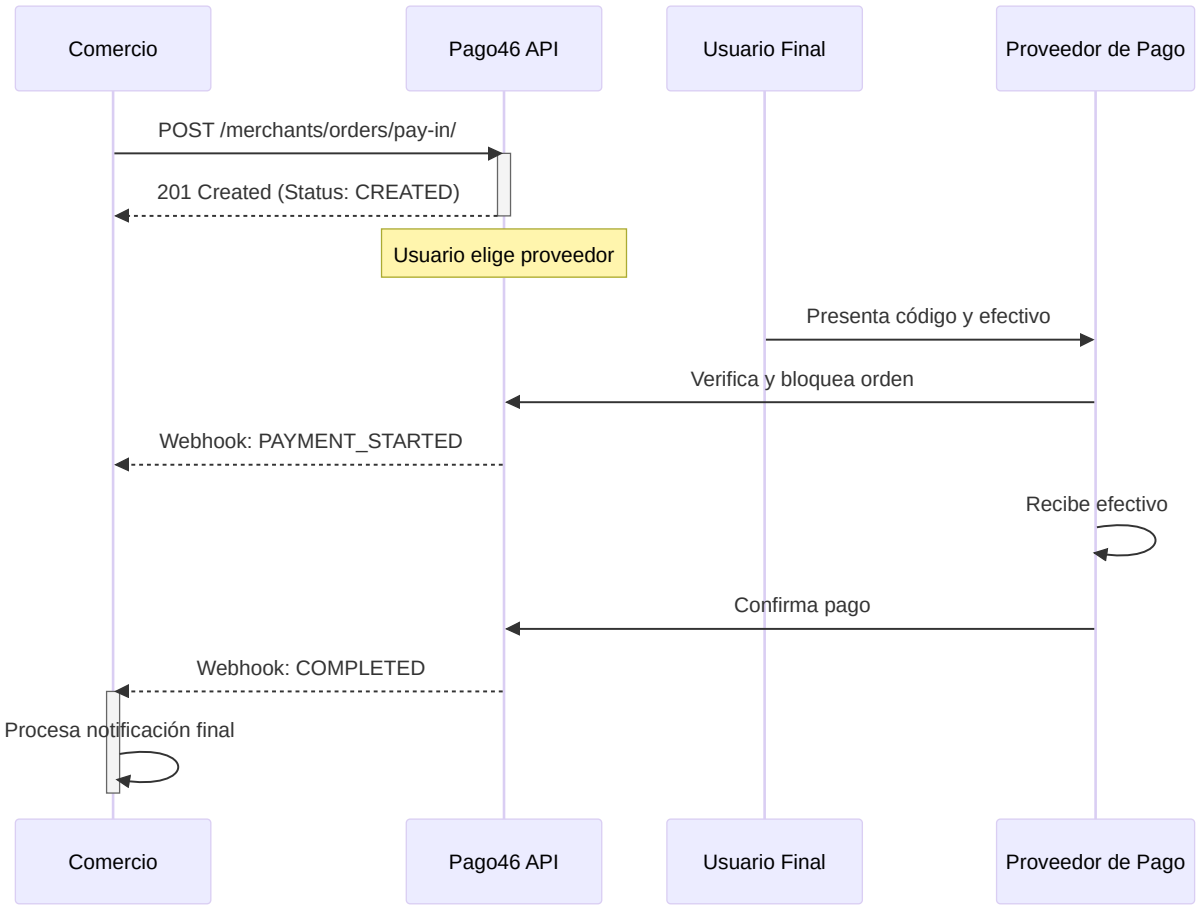
También puedes abrir la URL base sin UUID para capturar una orden manualmente durante pruebas de integración en iframe.

Más detalles en Checkout.

Flujo General

El proceso de pago para comercios se divide en tres etapas principales:

1. **Creación de Orden:** El comercio crea una orden de pago especificando el monto, país, datos del pagador y URLs de notificación.
2. **Procesamiento:** La orden es procesada por un Proveedor de Pago de la red, quien recibe el efectivo del usuario final.
3. **Notificaciones (Webhooks):** El comercio recibe actualizaciones del estado de la orden a través de webhooks configurados.



Requisitos Previos

Antes de comenzar, asegúrate de tener:

- **Credenciales de API:** Tu `Merchant-Key` y `Merchant-Secret` proporcionados por Pago46.
- **Autenticación HMAC:** Familiarízate con el esquema de autenticación descrito en la sección de Autenticación.
- **Endpoint de Webhook:** Una URL pública HTTPS donde recibirás las notificaciones de cambios de estado.

> **SEGURIDAD**
>
> Todas las peticiones deben incluir los headers de autenticación HMAC: `Merchant-Key`, `Message-Date` y `Message-Hash`.

---

1. Crear Orden de Pago

Para iniciar un cobro, debes crear una orden proporcionando la información del monto, país, pagador y configuración de notificaciones.

Endpoint: `POST /merchants/orders/pay-in/`

Parámetros de Request

Campos Obligatorios

Campo	Tipo	Descripción
`order_type`	String	Tipo de orden: `LocalCurrencyOrder`
`country`	String	Código ISO del país (ej: `MX`, `CL`, `CO`)
`price`	Decimal	Monto del pago (formato: `"1500.00"`)
`price_currency`	String	Código ISO de la moneda (ej: `MXN`, `CLP`, `COP`)
`description`	String	Descripción de la transacción
`merchant_order_id`	String	ID único de tu sistema (max 127 caracteres)
`notify_url`	String (URL)	URL para recibir webhooks de cambios de estado
`return_url`	String (URL)	URL de retorno para el usuario
`expiry`	DateTime	Fecha y hora de expiración (formato ISO 8601)

Campos Opcionales

Campo	Tipo	Descripción
`consumer_email`	String	Email del pagador
`consumer_phone_number`	String	Teléfono del pagador (max 128 caracteres)

> ⚠️ **DATOS DE CONTACTO**
>
> En pay-in, si envías datos de contacto, debes enviar ambos campos, solo `consumer_email` o ninguno. No se acepta `consumer_phone_number` sin email. El soporte para teléfono sin email estará disponible próximamente.

Ejemplo de Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-in/" \
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "order_type": "LocalCurrencyOrder",
    "country": "MX",
    "price": "1500.00",
    "price_currency": "MXN",
    "description": "Pago de suscripción - Usuario ABC123",
    "merchant_order_id": "ORDER-2024-001234",
    "notify_url": "https://tu-comercio.com/webhooks/pago46",
    "return_url": "https://tu-comercio.com/pago/volver",
    "consumer_email": "usuario@ejemplo.com",
    "consumer_phone_number": "+525512345678",
    "expiry": "2024-12-31T23:59:59Z"
  }'
```

Respuesta Exitosa (201 Created)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Pago de suscripción - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/pago/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
}
```


GUARDANDO EL ID

Guarda el `id` de la orden devuelto en la respuesta. Lo necesitarás para consultar el estado de la orden posteriormente.

---

2. Consultar Orden

Puedes consultar el estado actual de una orden en cualquier momento usando su ID.

Endpoint: `GET /merchants/orders/pay-in/{id}/`

Parámetros

Parámetro	Ubicación	Descripción
`id`	Path	UUID de la orden

Ejemplo de Request

```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-in/123e4567-e89b-12d3-a456-42661417400
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Respuesta (200 OK)


```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Pago de suscripción - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/pago/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
```

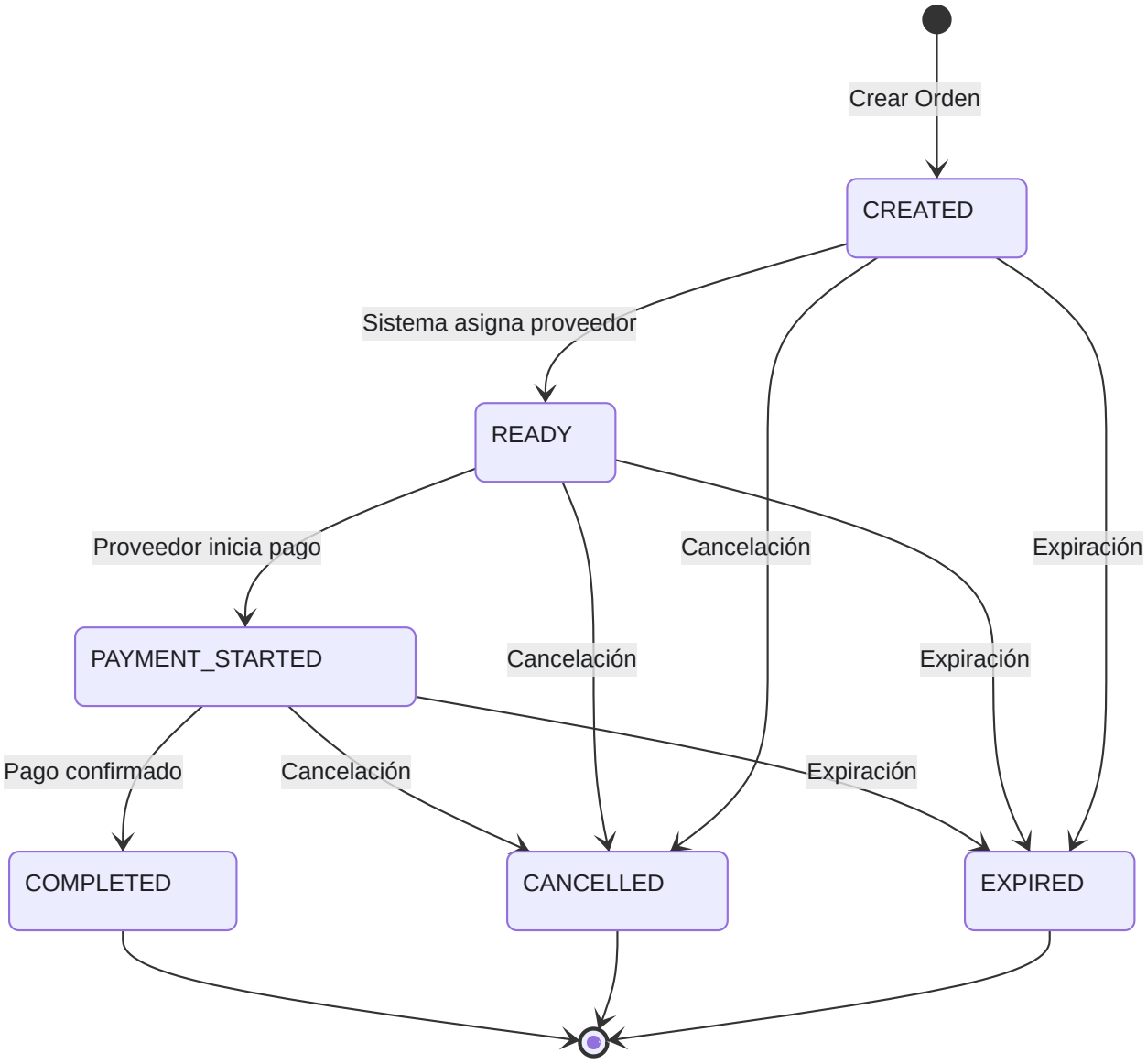
```
}
```

Estados de la Orden

Cada orden pasa por diferentes estados durante su ciclo de vida. Es importante que tu sistema maneje correctamente cada uno de estos estados.

Estado	Descripción	¿Qué significa para el comercio?
`CREATED`	Orden creada exitosamente	La orden se ha registrado y está pendiente de asignación a un proveedor
`READY`	Lista para procesar	Un proveedor puede comenzar a procesar el pago
`PAYMENT_STARTED`	Pago en proceso	Un proveedor ha bloqueado la orden y está procesando el pago
`COMPLETED`	Completada	El pago se completó exitosamente. El proveedor recibió el efectivo del usuario
`CANCELLED`	Cancelada	La orden fue cancelada y no se procesará
`EXPIRED`	Expirada	La orden expiró y no se procesará

Diagrama de Transición de Estados



Webhooks (Notificaciones)

Cada vez que el estado de una orden cambia, Pago46 enviará una notificación HTTP POST a la URL especificada en el campo `notify_url` de tu orden.

Estructura del Webhook

El webhook será enviado con autenticación HMAC. Debes verificar la firma para asegurar que la notificación proviene de Pago46.

Headers del Webhook

```
POST /webhooks/pago46 HTTP/1.1
Host: tu-comercio.com
Content-Type: application/json
Merchant-Key: PAGO46_SYSTEM
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Payload del Webhook

Estado: PAYMENT_STARTED | **Estado: COMPLETED** | **Estado: CANCELLED**


```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Pago de suscripción - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "PAYMENT_STARTED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/pago/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
}
```


Verificación de Webhooks

Es **crítico** que verifiques la autenticidad de cada webhook recibido para evitar procesamiento de notificaciones fraudulentas.

Ejemplo de Verificación en Python

```python
import hmac
import hashlib
import json
from flask import Flask, request, jsonify

app = Flask(__name__)

# Tu Merchant Secret (obtenido de Pago46)
MERCHANT_SECRET = "tu_merchant_secret_aqui"

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    # 1. Extraer headers
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')

    # 2. Obtener el body raw
    body_str = request.get_data(as_text=True)

    # 3. Construir string to sign
    # Formato: MERCHANT_KEY:MESSAGE_DATE:METHOD:PATH:BODY
    method = request.method  # "POST"
    path = request.path      # "/webhooks/pago46"
    string_to_sign = f"{merchant_key}:{message_date}:{method}:{path}:{body_str}"

    # 4. Calcular HMAC
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    # 5. Verificar
    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # 6. Procesar la notificación
    order_data = json.loads(body_str)
    order_id = order_data.get('id')
    order_status = order_data.get('status')
    merchant_order_id = order_data.get('merchant_order_id')
```

```
    print(f"Orden {merchant_order_id} ({order_id}) cambió a estado: {order_status}")

    # Actualizar tu base de datos
    if order_status == 'COMPLETED':
        # Marcar como completada
        paid_at = order_data.get('paid')
        print(f"Pago completado el: {paid_at}")
        # Activar servicio, entregar producto, etc.
    elif order_status == 'CANCELLED':
        # Marcar como cancelada
        print("Pago cancelado")

    # 7. Responder con 200 OK
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    app.run(port=5000)
```

Ejemplo de Verificación en Node.js

```
const express = require('express');
const crypto = require('crypto');
const app = express();

const MERCHANT_SECRET = 'tu_merchant_secret_aqui';

app.post('/webhooks/pago46', express.text({ type: '*/*' }), (req, res) => {
  // 1. Extraer headers
  const merchantKey = req.headers['merchant-key'];
  const messageDate = req.headers['message-date'];
  const receivedHash = req.headers['message-hash'];

  // 2. Body como string
  const bodyStr = req.body;

  // 3. Construir string to sign
  const method = req.method;
  const path = req.path;
  const stringToSign = `${merchantKey}:${messageDate}:${method}:${path}:${bodyStr}`;

  // 4. Calcular HMAC
  const calculatedHash = crypto
    .createHmac('sha256', MERCHANT_SECRET)
    .update(stringToSign)
    .digest('hex');

  // 5. Verificar
  if (calculatedHash !== receivedHash) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // 6. Procesar notificación
  const orderData = JSON.parse(bodyStr);
  const { id, status, merchant_order_id, paid } = orderData;

  console.log(`Orden ${merchant_order_id} (${id}) cambió a estado: ${status}`);

  if (status === 'COMPLETED') {
    console.log(`Pago completado el: ${paid}`);
    // Activar servicio, entregar producto, etc.
  } else if (status === 'CANCELLED') {
    console.log('Pago cancelado');
  }

  // 7. Responder
  res.status(200).json({ status: 'received' });
});

app.listen(5000, () => {
  console.log('Webhook server listening on port 5000');
});
```

⚠ RESPUESTA AL WEBHOOK

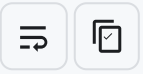
Debes responder con un código HTTP `200` o `201` para confirmar la recepción. Si no respondes exitosamente, Pago46 reintentará enviar la notificación.

Errores Comunes

Consulta la sección Manejo de errores comunes para ejemplos y el formato estándar de respuestas de error para validaciones de límites, campos obligatorios y otros errores de negocio.

Validación de Campos

Si envías datos inválidos o incompletos, recibirás un error `400 Bad Request` con detalles específicos:



```
{
    "type": "validation_error",
    "errors": [
        {
            "code": "invalid",
            "detail": "No matching configuration found for merchant, country, and currency.",
            "attr": "merchant_country_order_setting"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "country"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "price"
        },
        {
            "code": "required",
            "detail": "This field is required.",
            "attr": "price_currency"
        }
    ]
}
```

Tabla de Errores HTTP

Código HTTP	Descripción	Solución
`400 Bad Request`	Datos inválidos o campos faltantes	Verifica que todos los campos obligatorios estén presentes y con el formato correcto
`403 Forbidden`	Autenticación fallida	Verifica tus credenciales y la firma HMAC
`404 Not Found`	Orden no encontrada	Verifica que el ID de la orden sea correcto
`422 Unprocessable Entity`	Error de lógica de negocio	Revisa el mensaje de error específico

---

Mejores Prácticas

1. Idempotencia

Utiliza el campo `merchant_order_id` para identificar órdenes únicas en tu sistema. Si necesitas reintentar una creación de orden, usa el mismo `merchant_order_id` para evitar duplicados.

2. Manejo de Webhooks

- **Procesa webhooks de forma asíncrona:** No bloquees la respuesta HTTP mientras procesas la lógica de negocio.
- **Implementa reintentos:** Si tu servidor webhook está caído, Pago46 reintentará el envío.
- **Valida siempre la firma HMAC:** Nunca confíes en webhooks sin verificar su autenticidad.

3. Expiración de Órdenes

Establece un tiempo de expiración razonable (campo `expiry`). Órdenes típicamente expiran entre 24-72 horas después de su creación.

4. Monitoreo de Estados

- Monitorea órdenes en estado `PAYMENT_STARTED` que no transicionan a `COMPLETED` en un tiempo razonable.
- Implementa alertas para órdenes que permanezcan en estados intermedios por mucho tiempo.
- Revisa la página de Estado del Servicio ante errores atípicos para confirmar si hay incidentes o mantenimientos activos.

5. URLs de Notificación

- Usa URLs HTTPS para `notify_url`.
- Asegúrate de que el endpoint esté siempre disponible.
- Implementa logging para debugging.

Ejemplo Completo de Integración

A continuación, un ejemplo completo en Python que muestra cómo crear una orden y manejar webhooks:

```
import hmac
import hashlib
import time
import requests
import json
from flask import Flask, request, jsonify

# Configuración
API_BASE_URL = "https://api.dev.pago46.io"
MERCHANT_KEY = "tu_merchant_key"
MERCHANT_SECRET = "tu_merchant_secret"

app = Flask(__name__)

def generate_hmac(method, path, body_dict=None):
    """Genera la firma HMAC para autenticación"""
    timestamp = str(time.time())
    body_str = json.dumps(body_dict) if body_dict else ""

    string_to_sign = f"{MERCHANT_KEY}:{timestamp}:{method}:{path}:{body_str}"

    signature = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
```

```
    ).hexdigest()

    return {
        "Merchant-Key": MERCHANT_KEY,
        "Message-Date": timestamp,
        "Message-Hash": signature,
        "Content-Type": "application/json"
    }

def create_payin_order(amount, user_email, user_phone, merchant_order_id):
    """Crea una orden de pago"""
    path = "/api/v1/merchants/orders/pay-in/"

    order_data = {
        "order_type": "LocalCurrencyOrder",
        "country": "MX",
        "price": str(amount),
        "price_currency": "MXN",
        "description": f"Pago de usuario {user_email}",
        "merchant_order_id": merchant_order_id,
        "notify_url": "https://tu-comercio.com/webhooks/pago46",
        "return_url": "https://tu-comercio.com/pago/volver",
        "consumer_email": user_email,
        "consumer_phone_number": user_phone,
        "expiry": "2024-12-31T23:59:59Z"
    }

    headers = generate_hmac("POST", path, order_data)

    response = requests.post(
        f"{API_BASE_URL}{path}",
        headers=headers,
        json=order_data
    )

    if response.status_code == 201:
        order = response.json()
        print(f"✅ Orden creada exitosamente: {order['id']}")
        return order
    else:
        print(f"❌ Error al crear orden: {response.status_code}")
        print(response.text)
        return None

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    """Maneja webhooks de Pago46"""
    # Verificar firma HMAC
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')
    body_str = request.get_data(as_text=True)

    string_to_sign = f"{merchant_key}:{message_date}:{request.method}:{request.path}:{body_str}"
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # Procesar webhook
    order = json.loads(body_str)

    print(f"📩 Webhook recibido para orden: {order['merchant_order_id']}")
    print(f"   Estado: {order['status']}")

    # Actualizar base de datos según el estado
    if order['status'] == 'CREATED':
        print("   ⏳ Orden creada y esperando que el usuario elija un proveedor")
    elif order['status'] == 'PAYMENT_STARTED':
        print("   🔄 Un proveedor está procesando el pago")
    elif order['status'] == 'COMPLETED':
        print(f"   ✅ Pago completado el {order['paid']}")
        # Activar servicio, entregar producto, etc.
    elif order['status'] == 'CANCELLED':
        print("   ❌ Pago cancelado")

    return jsonify({"status": "received"}), 200
```

```
if __name__ == '__main__':
    # Ejemplo: Crear una orden de pago
    order = create_payin_order(
        amount=1500.00,
        user_email="usuario@ejemplo.com",
        user_phone="+525512345678",
        merchant_order_id="ORDER-2024-" + str(int(time.time()))
    )

    # Iniciar servidor de webhooks
    print("\n🚀 Iniciando servidor de webhooks...")
    app.run(port=5000)
```

Próximos Pasos

- **Autenticación:** Revisa la guía de autenticación HMAC para entender el mecanismo de seguridad en detalle.
- **Ambientes:** Familiarízate con los ambientes de desarrollo y producción.

> **SOPORTE**
>
> Si tienes preguntas o necesitas ayuda con tu integración, contacta al equipo de soporte de Pago46.

Retiros (Pay Out)

Este módulo permite a los Comercios (Merchants) crear órdenes de retiro (pay-out) para dispersar fondos a sus usuarios o clientes. Las órdenes creadas quedan disponibles para ser procesadas por los Proveedores de Pago integrados en la red de Pago46.

ENDPOINT BASE

Todas las rutas descritas a continuación son relativas a la URL base de la API: `/api/v1`

CHECKOUT DE PAGO46

Las órdenes contienen el campo `redirect_url`, que corresponde al link de Checkout para continuar el flujo del usuario.

- Sandbox: `https://checkout.dev.pago46.io`
- Producción: `https://checkout.prd.pago46.io`
- Formato por orden: `https://checkout.{dev|prd}.pago46.io/{UUID}`

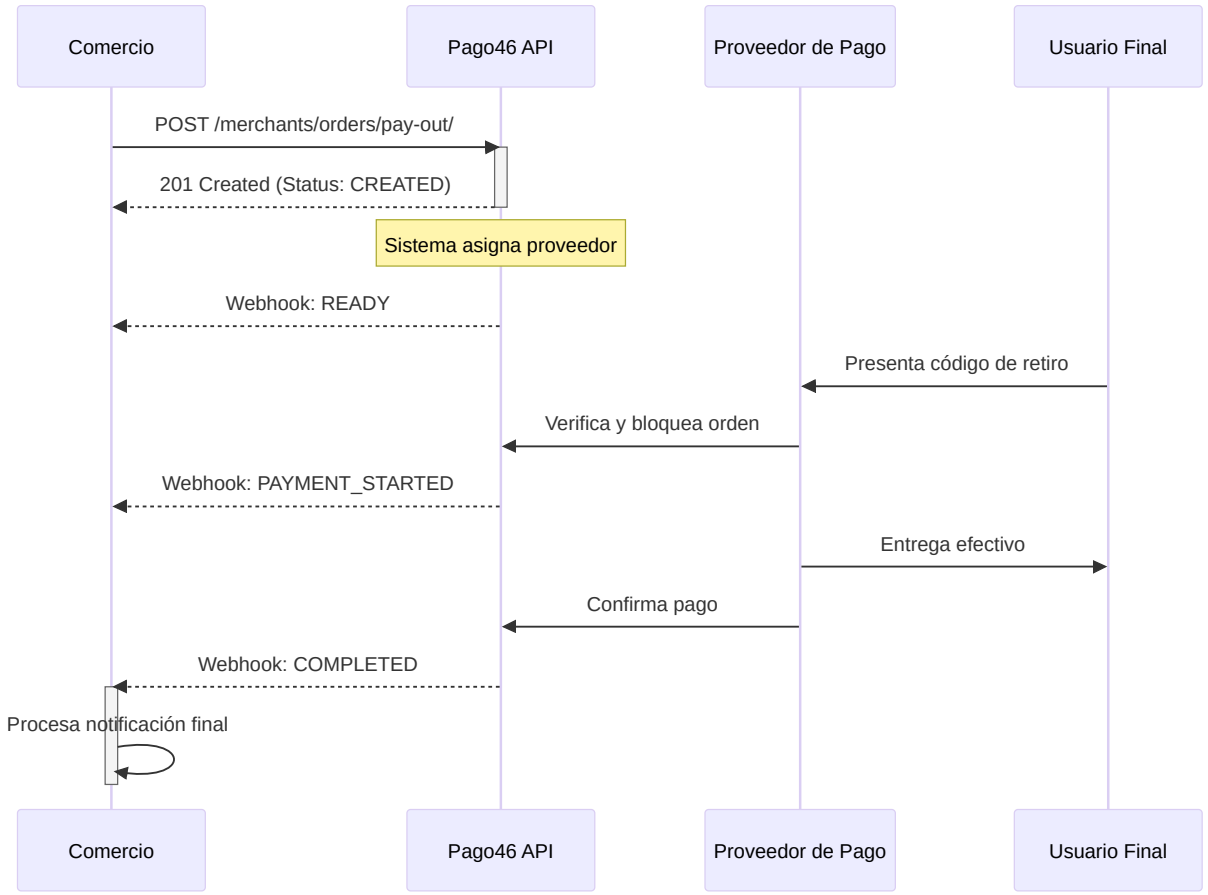
También puedes abrir la URL base sin UUID para capturar una orden manualmente durante pruebas de integración en iframe.

Más detalles en Checkout.

Flujo General

El proceso de retiro para comercios se divide en tres etapas principales:

1. **Creación de Orden:** El comercio crea una orden de retiro especificando el monto, país, datos del beneficiario y URLs de notificación.
2. **Procesamiento:** La orden es procesada por un Proveedor de Pago de la red, quien entrega el efectivo al usuario final.
3. **Notificaciones (Webhooks):** El comercio recibe actualizaciones del estado de la orden a través de webhooks configurados.



Requisitos Previos

Antes de comenzar, asegúrate de tener:

- **Credenciales de API:** Tu `Merchant-Key` y `Merchant-Secret` proporcionados por Pago46.
- **Autenticación HMAC:** Familiarízate con el esquema de autenticación descrito en la sección de Autenticación.
- **Endpoint de Webhook:** Una URL pública HTTPS donde recibirás las notificaciones de cambios de estado.

> **SEGURIDAD**
>
> Todas las peticiones deben incluir los headers de autenticación HMAC: `Merchant-Key`, `Message-Date` y `Message-Hash`.

---

1. Crear Orden de Retiro

Para iniciar un retiro, debes crear una orden proporcionando la información del monto, país, beneficiario y configuración de notificaciones.

Endpoint: `POST /merchants/orders/pay-out/`

Parámetros de Request

Campos Obligatorios

Campo	Tipo	Descripción
`order_type`	String	Tipo de orden: `LocalCurrencyOrder`
`country`	String	Código ISO del país (ej: `MX`, `CL`, `CO`)
`price`	Decimal	Monto del retiro (formato: `"1500.00"`)
`price_currency`	String	Código ISO de la moneda (ej: `MXN`, `CLP`, `COP`)
`description`	String	Descripción de la transacción
`merchant_order_id`	String	ID único de tu sistema (max 127 caracteres)
`notify_url`	String (URL)	URL para recibir webhooks de cambios de estado
`return_url`	String (URL)	URL de retorno para el usuario
`expiry`	DateTime	Fecha y hora de expiración (formato ISO 8601)

Campos Opcionales

Campo	Tipo	Descripción
`consumer_email`	String	Email del beneficiario
`consumer_phone_number`	String	Teléfono del beneficiario (max 128 caracteres)

> **RETIROS CON MONEDA EXTRANJERA**
>
> Para retiros internacionales o con conversión de divisas, consulta la documentación de **Comercios con Moneda Extranjera** en el menú lateral.

Ejemplo de Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/" \
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "order_type": "LocalCurrencyOrder",
    "country": "MX",
    "price": "1500.00",
    "price_currency": "MXN",
    "description": "Retiro de saldo - Usuario ABC123",
    "merchant_order_id": "ORDER-2024-001234",
    "notify_url": "https://tu-comercio.com/webhooks/pago46",
    "return_url": "https://tu-comercio.com/retiro/volver",
    "consumer_email": "usuario@ejemplo.com",
    "consumer_phone_number": "+525512345678",
    "expiry": "2024-12-31T23:59:59Z"
  }'
```

Respuesta Exitosa (201 Created)

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Retiro de saldo - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "CREATED",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/retiro/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
}
```


GUARDANDO EL ID

Guarda el `id` de la orden devuelto en la respuesta. Lo necesitarás para consultar el estado de la orden posteriormente.

---

2. Consultar Orden

Puedes consultar el estado actual de una orden en cualquier momento usando su ID.

Endpoint: `GET /merchants/orders/pay-out/{id}/`

Parámetros

Parámetro	Ubicación	Descripción
`id`	Path	UUID de la orden

Ejemplo de Request

```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/123e4567-e89b-12d3-a456-426614174(
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Respuesta (200 OK)


```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Retiro de saldo - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "READY",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/retiro/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
```

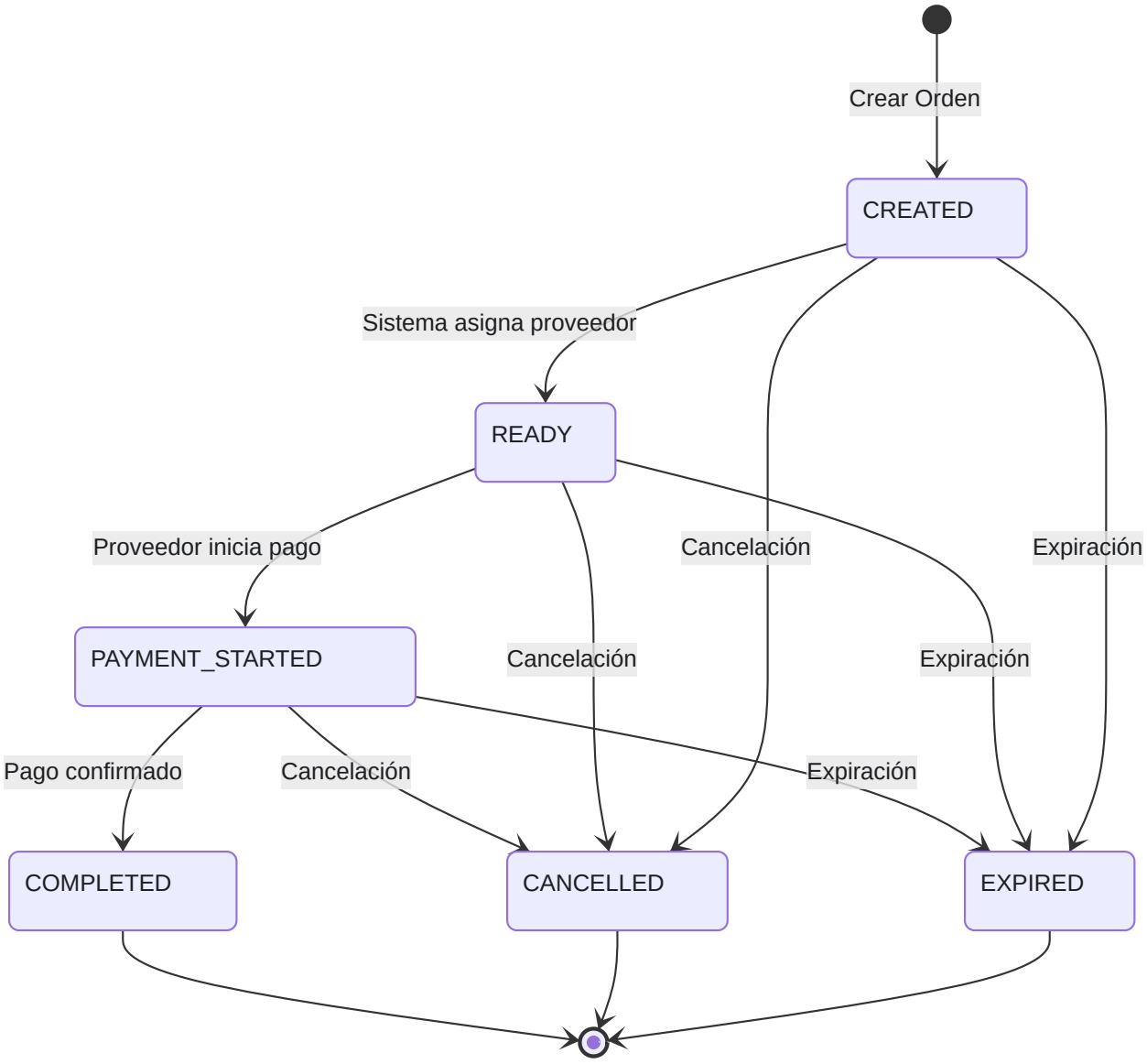
```
}
```

Estados de la Orden

Cada orden de retiro pasa por diferentes estados durante su ciclo de vida. Es importante que tu sistema maneje correctamente cada uno de estos estados.

Estado	Descripción	¿Qué significa para el comercio?
`CREATED`	Orden creada exitosamente	La orden se ha registrado y está pendiente de asignación a un proveedor
`READY`	Lista para procesar	Un proveedor puede comenzar a procesar el retiro
`PAYMENT_STARTED`	Pago en proceso	Un proveedor ha bloqueado la orden y está procesando el retiro
`COMPLETED`	Completada	El retiro se completó exitosamente. El beneficiario recibió el efectivo
`CANCELLED`	Cancelada	La orden fue cancelada y no se procesará
`EXPIRED`	Expirada	La orden expiró y no se procesará

Diagrama de Transición de Estados



Webhooks (Notificaciones)

Cada vez que el estado de una orden cambia, Pago46 enviará una notificación HTTP POST a la URL especificada en el campo `notify_url` de tu orden.

Estructura del Webhook

El webhook será enviado con autenticación HMAC. Debes verificar la firma para asegurar que la notificación proviene de Pago46.

Headers del Webhook

```
POST /webhooks/pago46 HTTP/1.1
Host: tu-comercio.com
Content-Type: application/json
Merchant-Key: PAGO46_SYSTEM
Message-Date: 1704463200.123
Message-Hash: a1b2c3d4e5f6...
```

Payload del Webhook

Estado: READY | **Estado: PAYMENT_STARTED** | **Estado: COMPLETED** | **Estado: CANCELLED**

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "order_type": "LocalCurrencyOrder",
  "country": "MX",
  "price": "1500.00",
  "price_currency": "MXN",
  "description": "Retiro de saldo - Usuario ABC123",
  "merchant_order_id": "ORDER-2024-001234",
  "status": "READY",
  "redirect_url": "https://checkout.dev.pago46.io/123e4567-e89b-12d3-a456-426614174000",
  "return_url": "https://tu-comercio.com/retiro/volver",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "+525512345678",
  "expiry": "2024-12-31T23:59:59Z",
  "paid": null
}
```

Verificación de Webhooks

Es **crítico** que verifiques la autenticidad de cada webhook recibido para evitar procesamiento de notificaciones fraudulentas.

Ejemplo de Verificación en Python

```python
import hmac
import hashlib
import json
from flask import Flask, request, jsonify

app = Flask(__name__)

# Tu Merchant Secret (obtenido de Pago46)
MERCHANT_SECRET = "tu_merchant_secret_aqui"

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    # 1. Extraer headers
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')

    # 2. Obtener el body raw
    body_str = request.get_data(as_text=True)

    # 3. Construir string to sign
    # Formato: MERCHANT_KEY:MESSAGE_DATE:METHOD:PATH:BODY
    method = request.method  # "POST"
    path = request.path      # "/webhooks/pago46"
    string_to_sign = f"{merchant_key}:{message_date}:{method}:{path}:{body_str}"

    # 4. Calcular HMAC
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    # 5. Verificar
    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # 6. Procesar la notificación
    order_data = json.loads(body_str)
    order_id = order_data.get('id')
    order_status = order_data.get('status')
    merchant_order_id = order_data.get('merchant_order_id')
```

```
    print(f"Orden {merchant_order_id} ({order_id}) cambió a estado: {order_status}")

    # Actualizar tu base de datos
    if order_status == 'COMPLETED':
        # Marcar como completada
        paid_at = order_data.get('paid')
        print(f"Retiro completado el: {paid_at}")
    elif order_status == 'CANCELLED':
        # Marcar como cancelada
        print("Retiro cancelado")

    # 7. Responder con 200 OK
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    app.run(port=5000)
```

Ejemplo de Verificación en Node.js

```
const express = require('express');
const crypto = require('crypto');
const app = express();

const MERCHANT_SECRET = 'tu_merchant_secret_aqui';

app.post('/webhooks/pago46', express.text({ type: '*/*' }), (req, res) => {
  // 1. Extraer headers
  const merchantKey = req.headers['merchant-key'];
  const messageDate = req.headers['message-date'];
  const receivedHash = req.headers['message-hash'];

  // 2. Body como string
  const bodyStr = req.body;

  // 3. Construir string to sign
  const method = req.method;
  const path = req.path;
  const stringToSign = `${merchantKey}:${messageDate}:${method}:${path}:${bodyStr}`;

  // 4. Calcular HMAC
  const calculatedHash = crypto
    .createHmac('sha256', MERCHANT_SECRET)
    .update(stringToSign)
    .digest('hex');

  // 5. Verificar
  if (calculatedHash !== receivedHash) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // 6. Procesar notificación
  const orderData = JSON.parse(bodyStr);
  const { id, status, merchant_order_id, paid } = orderData;

  console.log(`Orden ${merchant_order_id} (${id}) cambió a estado: ${status}`);

  if (status === 'COMPLETED') {
    console.log(`Retiro completado el: ${paid}`);
    // Actualizar base de datos
  } else if (status === 'CANCELLED') {
    console.log('Retiro cancelado');
    // Actualizar base de datos
  }

  // 7. Responder
  res.status(200).json({ status: 'received' });
});

app.listen(5000, () => {
  console.log('Webhook server listening on port 5000');
});
```

⚠ RESPUESTA AL WEBHOOK

Debes responder con un código HTTP `200` o `201` para confirmar la recepción. Si no respondes exitosamente, Pago46 reintentará enviar la notificación.

Errores Comunes

Consulta la sección Manejo de errores comunes para ejemplos y el formato estándar de respuestas de error para validaciones de límites, campos obligatorios y otros errores de negocio.

Validación de Campos

Si envías datos inválidos o incompletos, recibirás un error `400 Bad Request` estructurado como se muestra en la sección "Manejo de errores comunes". Allí encontrarás ejemplos detallados de respuestas típicas y explicaciones de cada campo del error.

Ejemplo de un posible error por campos requeridos:

```
{
  "type": "validation_error",
  "errors": [
    {
      "attr": "country",
      "code": "required",
      "detail": "This field is required."
    },
    {
      "attr": "price",
      "code": "invalid",
      "detail": "A valid number is required."
    },
    {
      "attr": "expiry",
      "code": "invalid",
      "detail": "Datetime has wrong format. Use one of these formats instead: YYYY-MM-DDThh:mm[:ss[.uuuuuu]]
[+HH:MM|-HH:MM|Z]."
    }
  ]
}
```

Tabla de Errores HTTP

Código HTTP	Descripción	Solución
`400 Bad Request`	Datos inválidos o campos faltantes	Verifica que todos los campos obligatorios estén presentes y con el formato correcto
`403 Forbidden`	Autenticación fallida	Verifica tus credenciales y la firma HMAC
`404 Not Found`	Orden no encontrada	Verifica que el ID de la orden sea correcto
`422 Unprocessable Entity`	Error de lógica de negocio	Revisa el mensaje de error específico

---

Mejores Prácticas

1. Idempotencia

Utiliza el campo `merchant_order_id` para identificar órdenes únicas en tu sistema. Si necesitas reintentar una creación de orden, usa el mismo `merchant_order_id` para evitar duplicados.

2. Manejo de Webhooks

- **Procesa webhooks de forma asíncrona:** No bloquees la respuesta HTTP mientras procesas la lógica de negocio.
- **Implementa reintentos:** Si tu servidor webhook está caído, Pago46 reintentará el envío.
- **Valida siempre la firma HMAC:** Nunca confíes en webhooks sin verificar su autenticidad.

3. Expiración de Órdenes

Establece un tiempo de expiración razonable (campo `expiry`). Órdenes típicamente expiran entre 24-72 horas después de su creación.

4. Monitoreo de Estados

- Monitorea órdenes en estado `PAYMENT_STARTED` que no transicionan a `COMPLETED` en un tiempo razonable.
- Implementa alertas para órdenes que permanezcan en estados intermedios por mucho tiempo.
- Revisa la página de Estado del Servicio ante errores atípicos para confirmar si hay incidentes o mantenimientos activos.

5. URLs de Notificación

- Usa URLs HTTPS para `notify_url`.
- Asegúrate de que el endpoint esté siempre disponible.
- Implementa logging para debugging.

Ejemplo Completo de Integración

A continuación, un ejemplo completo en Python que muestra cómo crear una orden y manejar webhooks:

```
import hmac
import hashlib
import time
import requests
import json
from flask import Flask, request, jsonify

# Configuración
API_BASE_URL = "https://api.dev.pago46.io"
MERCHANT_KEY = "tu_merchant_key"
MERCHANT_SECRET = "tu_merchant_secret"

app = Flask(__name__)

def generate_hmac(method, path, body_dict=None):
    """Genera la firma HMAC para autenticación"""
    timestamp = str(time.time())
    body_str = json.dumps(body_dict) if body_dict else ""

    string_to_sign = f"{MERCHANT_KEY}:{timestamp}:{method}:{path}:{body_str}"

    signature = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
```

```
    ).hexdigest()

    return {
        "Merchant-Key": MERCHANT_KEY,
        "Message-Date": timestamp,
        "Message-Hash": signature,
        "Content-Type": "application/json"
    }

def create_payout_order(amount, user_email, user_phone, merchant_order_id):
    """Crea una orden de retiro"""
    path = "/api/v1/merchants/orders/pay-out/"

    order_data = {
        "order_type": "LocalCurrencyOrder",
        "country": "MX",
        "price": str(amount),
        "price_currency": "MXN",
        "description": f"Retiro de usuario {user_email}",
        "merchant_order_id": merchant_order_id,
        "notify_url": "https://tu-comercio.com/webhooks/pago46",
        "return_url": "https://tu-comercio.com/retiro/volver",
        "consumer_email": user_email,
        "consumer_phone_number": user_phone,
        "expiry": "2024-12-31T23:59:59Z"
    }

    headers = generate_hmac("POST", path, order_data)

    response = requests.post(
        f"{API_BASE_URL}{path}",
        headers=headers,
        json=order_data
    )

    if response.status_code == 201:
        order = response.json()
        print(f"✅ Orden creada exitosamente: {order['id']}")
        return order
    else:
        print(f"❌ Error al crear orden: {response.status_code}")
        print(response.text)
        return None

@app.route('/webhooks/pago46', methods=['POST'])
def webhook_handler():
    """Maneja webhooks de Pago46"""
    # Verificar firma HMAC
    merchant_key = request.headers.get('Merchant-Key')
    message_date = request.headers.get('Message-Date')
    received_hash = request.headers.get('Message-Hash')
    body_str = request.get_data(as_text=True)

    string_to_sign = f"{merchant_key}:{message_date}:{request.method}:{request.path}:{body_str}"
    calculated_hash = hmac.new(
        MERCHANT_SECRET.encode('utf-8'),
        string_to_sign.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(calculated_hash, received_hash):
        return jsonify({"error": "Invalid signature"}), 403

    # Procesar webhook
    order = json.loads(body_str)

    print(f"📩 Webhook recibido para orden: {order['merchant_order_id']}")
    print(f"   Estado: {order['status']}")

    # Actualizar base de datos según el estado
    if order['status'] == 'READY':
        print("   ⏳ Orden lista para ser procesada por un proveedor")
    elif order['status'] == 'PAYMENT_STARTED':
        print("   🔄 Un proveedor está procesando el retiro")
    elif order['status'] == 'COMPLETED':
        print(f"   ✅ Retiro completado el {order['paid']}")
        # Actualizar saldo del usuario, enviar notificación, etc.
    elif order['status'] == 'CANCELLED':
        print("   ❌ Retiro cancelado")
        # Reembolsar saldo, notificar usuario, etc.
```

```
    return jsonify({"status": "received"}), 200

if __name__ == '__main__':
    # Ejemplo: Crear una orden de retiro
    order = create_payout_order(
        amount=1500.00,
        user_email="usuario@ejemplo.com",
        user_phone="+525512345678",
        merchant_order_id="ORDER-2024-" + str(int(time.time()))
    )

    # Iniciar servidor de webhooks
    print("\n🚀 Iniciando servidor de webhooks...")
    app.run(port=5000)
```

Próximos Pasos

- **Retiros con Moneda Extranjera:** Consulta la documentación de **Comercios con Moneda Extranjera** en el menú lateral para aprender a crear órdenes internacionales.
- **Autenticación:** Revisa la guía de autenticación HMAC para entender el mecanismo de seguridad en detalle.
- **Ambientes:** Familiarízate con los ambientes de desarrollo y producción.

SOPORTE

Si tienes preguntas o necesitas ayuda con tu integración, contacta al equipo de soporte de Pago46.

Retiros en Moneda Extranjera (Pay Out)

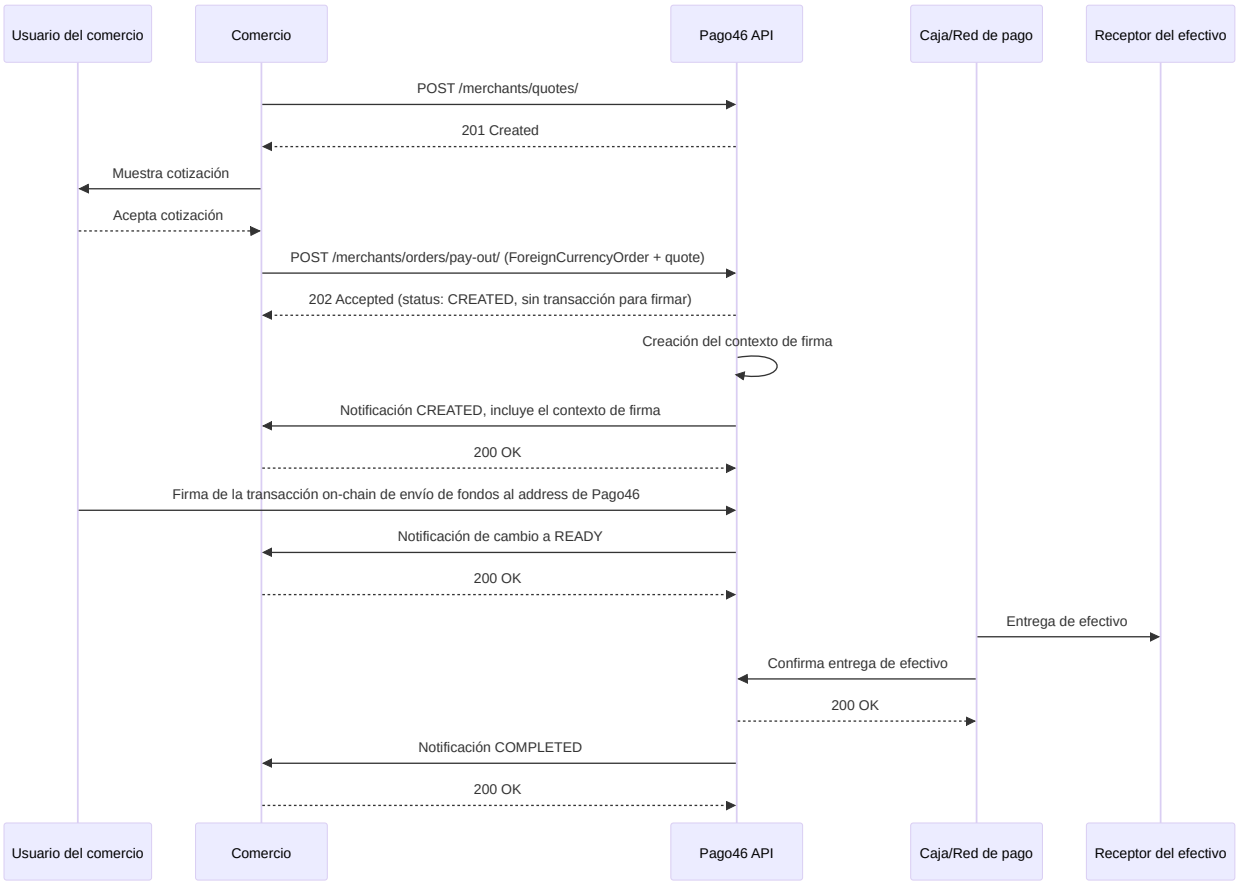
Este módulo permite crear retiros (pay-out) en moneda local usando fondos de origen en moneda extranjera (por ejemplo, USDC → MXN).

Como Comercio, tu integración habla exclusivamente con Pago46. Pago46 gestiona internamente la cotización, el seguimiento de la recepción de fondos y la habilitación del retiro en efectivo.

> **ENDPOINT BASE**
>
> Todas las rutas descritas a continuación son relativas a la URL base de la API: `/api/v1`

Flujo General

1. Creas una cotización para un par de activos (`POST /merchants/quotes/`).
2. Tu usuario acepta la cotización en tu aplicación.
3. Creas una orden de retiro foreign asociada a esa cotización.
4. Tu usuario envía los fondos al address indicado por Pago46.
5. Si el fondeo llega dentro de la ventana válida, la orden avanza a `READY` y queda lista para continuar el flujo normal de retiro.



Nota: la generación y entrega de la transacción blockchain ocurre en segundo plano, nunca en la respuesta inmediata del POST. Siempre espera el webhook antes de continuar con la firma.

Requisitos Previos

- Credenciales de Comercio (`Merchant-Key` y `Merchant-Secret`).
- Firma HMAC en cada request (`Message-Date`, `Message-Hash`).
- `notify_url` pública por HTTPS para cambios de estado.
- Al menos uno entre `consumer_email` o `consumer_phone_number` al crear la orden.

Revisa la sección de Autenticación para la firma.

---

1) Crear Cotización

Solicita una cotización para el par de activos deseado.

Endpoint: `POST /merchants/quotes/`

Campos de Request

Campo	Tipo	Requerido	Descripción
`blockchain`	String	Sí	Red de fondeo (`SOLANA`, `STELLAR`, `MONAD`).
`send`	Decimal	Sí	Monto que enviará el usuario.
`send_currency`	String	Sí	Moneda de origen (ej. `USDC`).
`target_country`	String	Sí	País de destino (`MX`, `CL`, etc.).
`target_currency`	String	Sí	Moneda local objetivo (ej. `MXN`).

Ejemplo de Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/quotes/" \
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "blockchain": "SOLANA",
    "send": "150.00",
    "send_currency": "USDC",
    "target_country": "MX",
    "target_currency": "MXN"
  }'
```

Respuesta (201 Created)

```
{
  "id": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "blockchain": "SOLANA",
  "send": "150.00",
  "send_currency": "USDC",
  "target_country": "MX",
  "target_currency": "MXN",
  "calculated_order_price": "2895.00",
  "calculated_order_price_currency": "MXN",
  "expires": "2030-01-01T11:05:00Z"
}
```

> **TIP**
>
> Guarda el `id` de la cotización. Lo usarás en la creación de la orden foreign.

---

2) Crear Orden de Retiro Foreign

Cuando tu usuario acepte la cotización, crea la orden de pay-out foreign usando el `quote`.

Endpoint: `POST /merchants/orders/pay-out/`

Campos de Request

Campo	Tipo	Requerido	Descripción
`order_type`	String	Sí	Debe ser `ForeignCurrencyOrder`.
`quote`	UUID	Sí	ID de la cotización creada previamente.
`description`	String	Sí	Descripción de la transacción.
`merchant_order_id`	String	Sí	ID único de tu sistema por Comercio.
`notify_url`	URL	Sí	URL para notificaciones de estado.
`return_url`	URL	Sí	URL de retorno.
`expiry`	DateTime	Sí	Expiración de la orden (ISO 8601).
`consumer_email`	String	Condicional	Requerido si no envías teléfono.
`consumer_phone_number`	String	Condicional	Requerido si no envías email.
`wallet`	String	Sí	Wallet que firmará la transacción on-chain.

Ejemplo de Request

```
curl -X POST "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/" \
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
    "description": "Retiro cash desde balance USDC",
    "merchant_order_id": "FX-PO-2026-0001",
    "notify_url": "https://tu-comercio.com/webhooks/pago46",
    "return_url": "https://tu-comercio.com/retiro/volver",
    "consumer_email": "usuario@ejemplo.com",
    "expiry": "2030-01-01T12:05:00Z",
    "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV",
    "order_type": "ForeignCurrencyOrder"
  }'
```

> **IMPORTANTE**
>
> La respuesta de este POST retorna **202 Accepted**:
>
> - La orden ha sido creada, pero **NO INCLUYE** la transacción blockchain por firmar.

- Pago46 genera la transacción de forma asíncrona.
- Cuando la transacción on-chain esté lista, recibirás un webhook de actualización de la orden: el `status` seguirá siendo `CREATED`, pero ahora el payload incluirá el campo `transaction` con los datos para firmar.
- Es decir, puedes recibir múltiples webhooks con el mismo `status` (`CREATED`) pero con distinto contenido; considera este webhook como el evento de **"transaction ready"**.
- **¡No intentes firmar ni mostrar datos de blockchain al usuario hasta recibir este webhook!**

Después de crear la orden y recibir el 202 Accepted, tu integración debe esperar la notificación asincrónica por webhook. Solo cuando llegue el webhook con la transacción, el usuario podrá firmar on-chain.

Respuesta inmediata del POST (sin transacción):


```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "direction": "PAY_OUT",
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Retiro cash desde balance USDC",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "CREATED",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://tu-comercio.com/retiro/volver",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-01T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV",
  "order_type": "ForeignCurrencyOrder"
}
```


Ejemplo de webhook: aquí la orden sí incluye el campo `transaction`:


```
{
  "id": "01952d91-a0ff-7f57-8f4e-68d95be01122",
  "order_type": "ForeignCurrencyOrder",
  "country": "MX",
  "price": "2895.00",
  "price_currency": "MXN",
  "description": "Retiro cash desde balance USDC",
  "merchant_order_id": "FX-PO-2026-0001",
  "status": "CREATED",
  "notify_url": "https://tu-comercio.com/webhooks/pago46",
  "redirect_url": "https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122",
  "return_url": "https://tu-comercio.com/retiro/volver",
  "consumer_email": "usuario@ejemplo.com",
  "consumer_phone_number": "",
  "expiry": "2030-01-01T12:05:00Z",
  "paid": null,
  "quote": "01952d8f-8da1-7f4b-bb36-cfba8a3be829",
  "transaction":
"AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADyTBSt8EsV++3pX0uVJVEfDjDp8mnxoq
  "wallet": "7EcDhSYGxXyscszYEp35KHN8vvw3svAuLKTzXwCFLtV"
}
```


---

3) Consultar Orden

Puedes consultar la orden de retiro cuando necesites verificar su estado.

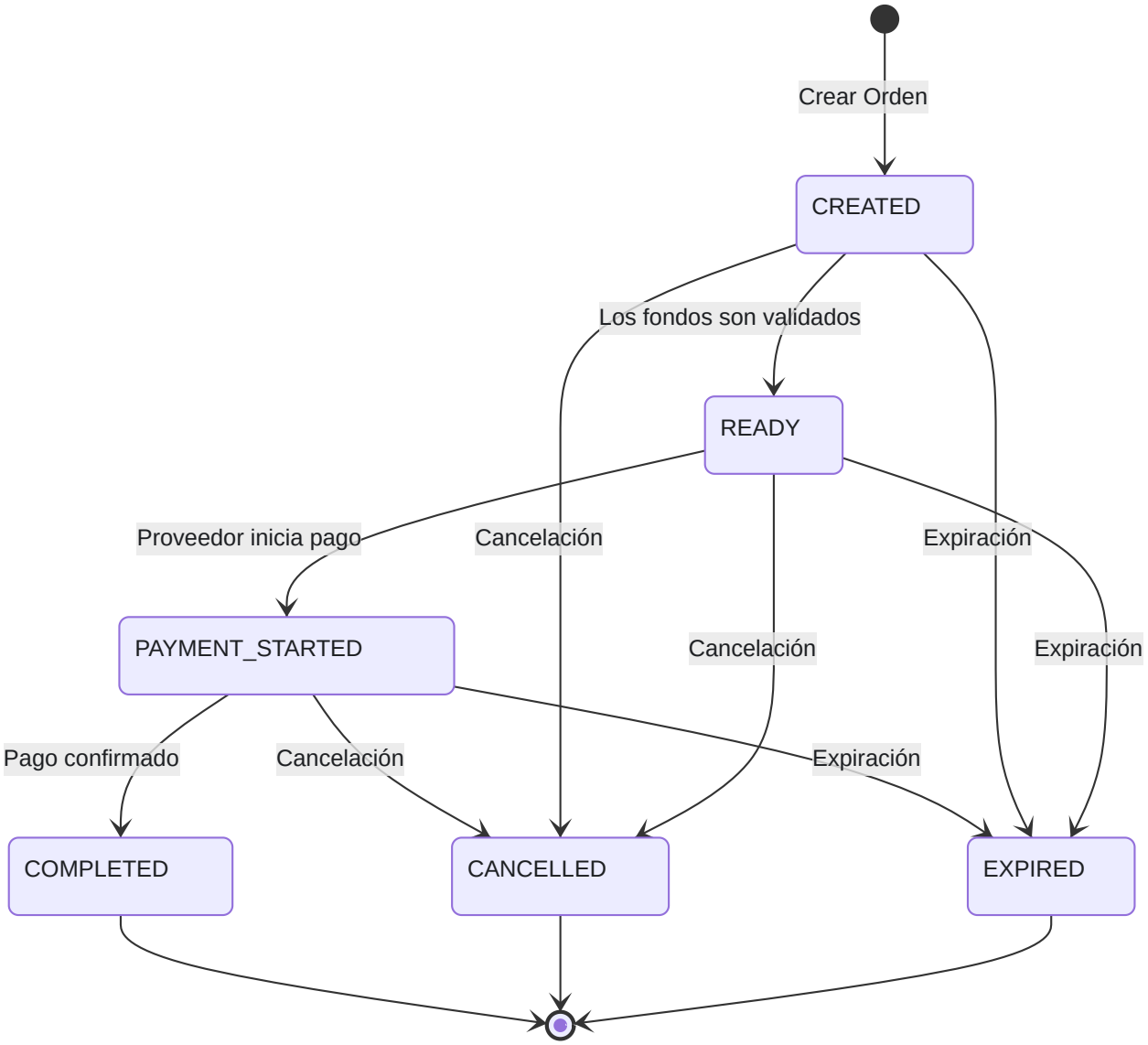
Endpoint: `GET /merchants/orders/pay-out/{id}/`

```
curl -X GET "https://api.dev.pago46.io/api/v1/merchants/orders/pay-out/01952d91-a0ff-7f57-8f4e-68d95be011
  -H "Merchant-Key: <TU_MERCHANT_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

Estados de la Orden (Foreign Pay-Out)

Estado	Descripción	¿Qué significa para el comercio?
`CREATED`	Orden creada exitosamente	La orden se ha registrado y está pendiente de asignación a un proveedor
`READY`	Fondos validados; orden lista para ser procesada	La recepción de fondos fue validada; la orden puede ser tomada por un proveedor para el retiro
`PAYMENT_STARTED`	Pago en proceso	Un proveedor ha bloqueado la orden y está procesando el retiro
`COMPLETED`	Completada	El retiro se completó exitosamente. El beneficiario recibió el efectivo
`CANCELLED`	Cancelada	La orden fue cancelada y no se procesará
`EXPIRED`	Expirada	La orden expiró y no se procesará

Diagrama de Transición de Estados



Buenas Prácticas de Integración

- Usa `merchant_order_id` idempotente por operación de negocio.
- Maneja cambios de estado de forma tolerante a reintentos.
- No asumas `COMPLETED` inmediatamente después de crear la orden.
- Presenta al usuario una cuenta regresiva según la ventana de cotización.
- **Después del POST, mantén un estado "Esperando transacción" hasta recibir el webhook con el campo `transaction`.**
- Almacena el `id` de la orden devuelta para correlacionar el POST inicial y el webhook subsiguiente.
- Ante errores de validación, solicita una nueva cotización antes de reintentar. Consulta la sección Manejo de errores comunes para más detalles sobre el formato de los mensajes de error devueltos por la API.

Si necesitas soporte para nuevos pares (por ejemplo, activos de origen o países adicionales), contacta a tu representante de Pago46.

Checkout

Checkout es la aplicación web de Pago46 diseñada para integrarse como iframe en tu experiencia de usuario. Desde ahí, tus usuarios pueden completar flujos de pay-in y pay-out con una interfaz optimizada para móvil y desktop.

Por qué es importante Checkout

Checkout estandariza la experiencia final del usuario en un solo punto, independientemente del tipo de operación o canal de acceso. Esto te ayuda a:

- reducir tiempos de implementación para nuevos flujos,
- mantener una UX consistente en web y móvil,
- acelerar pruebas de punta a punta,
- simplificar soporte operativo con una ruta de ejecución común.

Además, Checkout incorpora capacidades internas para monitoreo técnico, seguimiento de errores y rendimiento, así como soporte operativo. También puede integrar servicios de terceros para enriquecer la experiencia del usuario, incluyendo componentes geoespaciales con conexiones en tiempo real.

Ambientes

Ambiente	URL base
Sandbox	`https://checkout.dev.pago46.io`
Producción	`https://checkout.prd.pago46.io`

Formato por orden:

- Sandbox: `https://checkout.dev.pago46.io/{UUID}`
- Producción: `https://checkout.prd.pago46.io/{UUID}`

También puedes abrir la URL base (sin UUID) e ingresar el UUID de forma manual.

`redirect_url` llega en la respuesta de creación/consulta de la orden y apunta directamente a la URL de Checkout para esa orden.

Integración en iframe

Flujo recomendado:

1. Crea la orden en Pago46.
2. Toma `redirect_url` de la respuesta de la orden.
3. Carga ese `redirect_url` en un iframe dentro de tu aplicación.
4. Para QA, valida también el ingreso manual del UUID desde la página principal de Checkout.

Ejemplo base de iframe:

```
<iframe
```

```
  src="https://checkout.dev.pago46.io/01952d91-a0ff-7f57-8f4e-68d95be01122"
  width="100%"
  height="720"
  style="border: 0"
  allow="clipboard-write"
></iframe>
```

Recomendaciones puntuales para iframe web:

- Usa ancho completo y altura suficiente para evitar scroll interno excesivo.
- No bloquees cookies, storage o navegación dentro del dominio de Checkout.
- Permite `clipboard-write` y capacidades estándar del navegador cuando aplique.
- Maneja expiración de sesión y recargas sin romper el estado de tu pantalla.
- Si haces pruebas manuales, abre también la URL base e ingresa el UUID sin usar una URL prearmada.

Integración en apps móviles

Checkout se integra como webview, sin SDK adicional.

App móvil con WebView

- Carga `redirect_url` directamente en el WebView.
- Evita interceptar navegación de Checkout salvo casos estrictamente necesarios.
- Mantén un User-Agent estable para evitar diferencias de render inesperadas.

App nativa (iOS/Android)

- Integra Checkout como webview embebido en el flujo de retiro/cobro.
- Controla bien el botón de back para no cerrar el flujo accidentalmente.
- Define una política clara para abrir enlaces externos fuera del WebView.

App React Native

- Integra Checkout con WebView (sin SDK propietario).
- Reutiliza el mismo contrato: recibir `redirect_url` y cargarlo en la vista.
- Prueba ciclo completo en foreground/background para validar continuidad.

Permisos que podrían requerirse en usuario final

Dependiendo del flujo y del contexto operativo, Checkout puede requerir:

- ubicación aproximada o precisa (servicios geoespaciales y puntos cercanos),
- acceso a red/internet estable para actualizaciones en tiempo real,
- permisos del navegador o WebView asociados a geolocalización.

Recomendación: informa al usuario por qué se solicita cada permiso antes de que aparezca el prompt del sistema.

Flujo de pruebas recomendado

1. Crea una orden en sandbox.
2. Toma `redirect_url` y cárgalo en iframe o WebView.
3. Ejecuta el flujo completo con la orden pre-cargada.

4. Repite entrando a la página principal de Checkout e ingresando el UUID manualmente.
5. Valida comportamiento en desktop, Android e iOS.

Demos

Android

iOS

Desktop

Integración para Proveedores de Pago

Como proveedor de pago, integras tu red de puntos físicos (tiendas, kioscos, agentes) a Pago46 para procesar cobros y dispersiones de efectivo generados por comercios.

Servicios Disponibles

Pay-In (Cobros)

Procesa pagos en efectivo de usuarios finales en tus puntos físicos.

Flujo: Usuario presenta código → Validas y bloqueas orden → Recibes efectivo → Confirmas transacción.

→ **Ver documentación de Pay-In**

Pay-Out (Dispersiones)

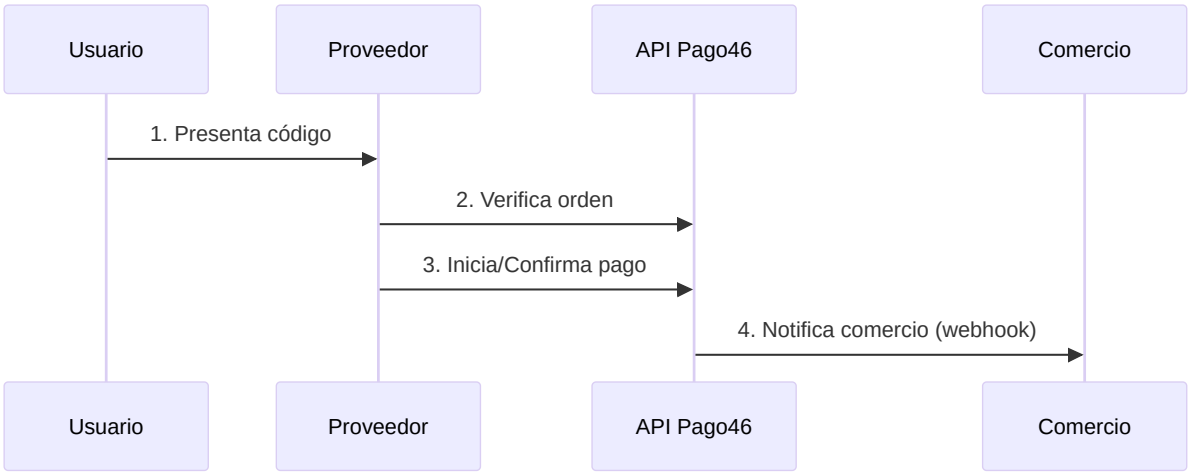
Dispersa efectivo a usuarios finales en tus puntos físicos.

Flujo: Usuario presenta código → Validas y bloqueas orden → Entregas efectivo → Confirmas transacción.

→ **Ver documentación de Pay-Out**

Flujo de Integración

La arquitectura es Server-to-Server con autenticación HMAC SHA-256.



Pasos de Integración

1. **Obtén tus credenciales** - Recibe tu `Provider-Key` y `Provider-Secret`
2. **Registra tus redes** - Envíanos tu lista de redes/subredes para asignar `network_id`
3. **Implementa autenticación HMAC** - Ver guía
4. **Consulta órdenes** - Usa `GET /providers/orders/{type}/{code}/`

5. **Procesa transacciones** - Usa `start-payment` y `confirm-payment`

> **MECANISMO DE BLOQUEO**
>
> El bloqueo aplica tanto en Pay-In como en Pay-Out. Cuando llamas `start-payment`, la orden queda reservada para tu red hasta confirmar o cancelar.

> **REDES Y SUBREDES**
>
> Para usar los endpoints, necesitas un `network_id` válido. Envíanos tu lista de redes/subredes para que podamos registrarlas y asignar los IDs.

Ambientes

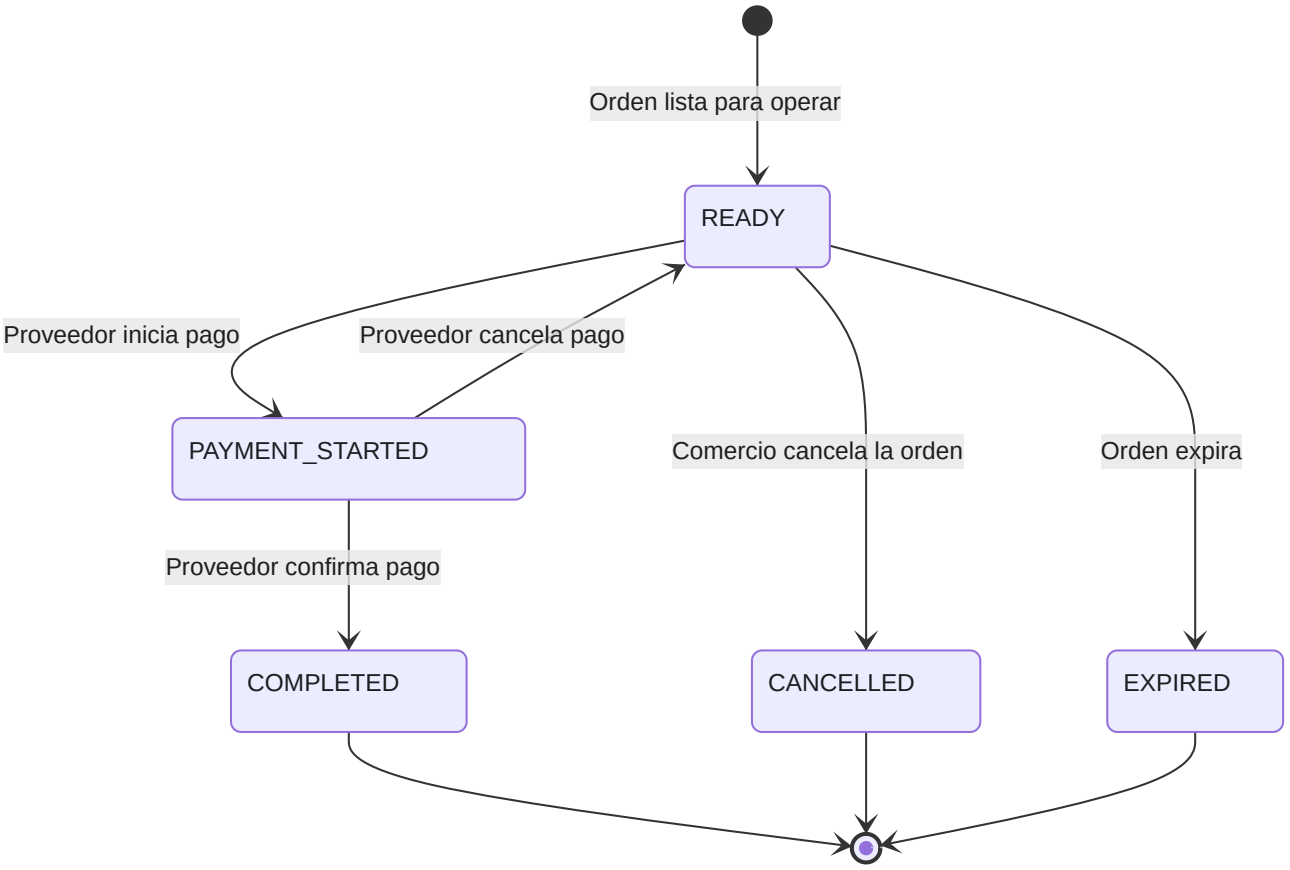
Ambiente	URL	Uso
Sandbox	`api.dev.pago46.io`	Desarrollo y pruebas
Producción	`api.prd.pago46.io`	Operaciones reales

→ Ver detalles de ambientes

> **MONITOREO OPERATIVO**
>
> Para operación diaria en caja/soporte, consulta la página de Estado del Servicio y suscríbete a alertas por correo o Slack para enterarte de incidentes y mantenimientos.

Ciclo de Vida de Órdenes

El diagrama de estados es el mismo para Pay-In y Pay-Out.



Cobertura

Pago46 opera en múltiples países de Latinoamérica. Ver lista completa.

Pagos (Pay In)

Este módulo permite a los Proveedores de Pago (Payment Providers) procesar solicitudes de **depósito o pago** iniciadas por usuarios de Pago46. Al igual que en los retiros, este flujo utiliza un mecanismo de bloqueo (*locking*) para evitar que una misma orden sea cobrada dos veces por distintos proveedores simultáneamente.

> **ENDPOINT BASE**
>
> Todas las rutas descritas a continuación son relativas a la URL base de la API: `/api/v1`

Registro de Redes de Pago

Antes de procesar transacciones, los proveedores deben registrar las redes de pago que utilizan. Cada red debe tener un identificador único (`network_id`) que se utilizará en todas las transiciones de estado.

Ejemplo de IDs de Red:

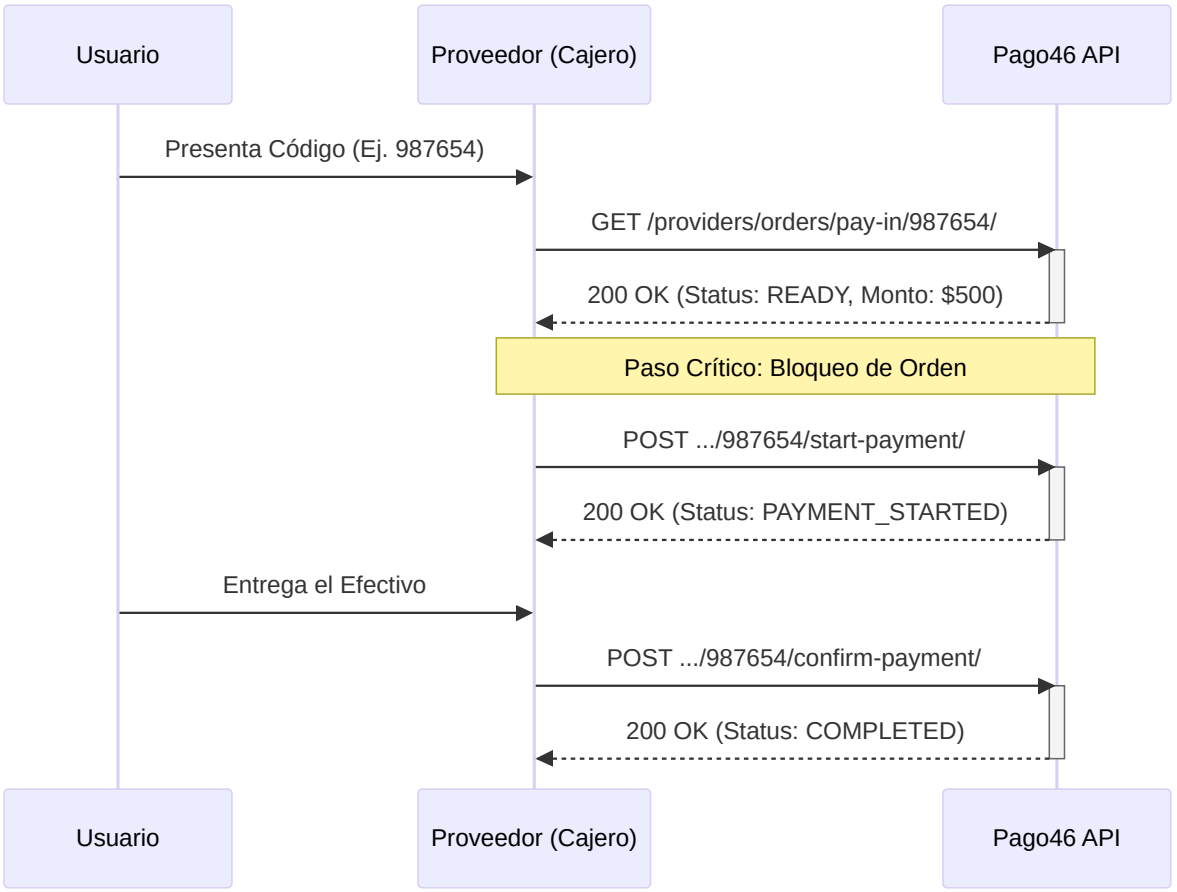
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACTO**
>
> Para registrar tus redes de pago, contacta al equipo de integración de Pago46. Los IDs que proporciones serán los que deberás usar en las solicitudes de cobro.

Ciclo de Vida de la Orden

El proceso asegura que el dinero sea recolectado y conciliado correctamente antes de liberar el servicio al usuario final.

1. **Consulta (Check):** El proveedor escanea o ingresa el código del usuario para ver el monto a cobrar y validar que el estado sea `READY`.
2. **Bloqueo (Start Payment):** El proveedor "toma" la orden. El estado cambia a `PAYMENT_STARTED`. **Esto asegura la intención de cobro.**
3. **Confirmación (Confirm Payment):** Una vez que el usuario entrega el efectivo y este ingresa a la caja, el proveedor confirma la transacción (estado `COMPLETED`).



---

1. Verificar Orden

Cuando el cliente se acerca a la caja para pagar, debes consultar el código para informar el monto exacto a cobrar.

Endpoint: `GET /providers/orders/pay-in/{code}/`

Parámetro	Ubicación	Descripción
`code`	Path	El código de pago generado por el usuario (numérico o QR).

Request | **Response (200 OK) - Order Ready**

```
curl -X GET "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

⚠ VALIDACIÓN DE ESTADO

Verifica siempre que el `status` sea **`READY`**. Si la orden está `EXPIRED` o `COMPLETED`, no debes recibir dinero del usuario.

---

2. Iniciar Cobro (Bloqueo)

Antes de recibir el dinero, debes informar a Pago46 que estás atendiendo esta orden. Esto previene que el usuario intente pagar el mismo código en otro proveedor simultáneamente.

Endpoint: `POST /providers/orders/pay-in/{code}/start-payment/`

Request | **Response (200 OK)** | **Error (403 Forbidden) - Order Already In Progress**

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/start-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Campo	Tipo	Descripción
`network_id`	string	ID de la red de pago utilizada por el proveedor. **Requerido**.
`price`	string	Monto exacto que se está cobrando. Debe coincidir con el `price` de la orden. **Requerido**.
`price_currency`	string	Código de moneda (ej. `MXN`, `USD`). Debe coincidir con el `price_currency` de la orden. **Requerido**.

SUBREDES PARA PROVEEDORES

Como proveedor, debes suministrar el `network_id` que identifica la red específica utilizada para procesar esta transacción. Esto permite a Pago46 mantener un registro detallado de qué red procesó cada pago y proporcionar esta información al usuario final.

MOMENTO DEL COBRO

Una vez recibas el estado `PAYMENT_STARTED`, puedes proceder a solicitar y recibir el dinero del cliente con seguridad.

---

3. Confirmar Cobro (Finalización)

Una vez que el dinero está seguro en tu caja, confirma la transacción. En este momento, Pago46 notificará al comercio original (Merchant) para que libere el producto o servicio al usuario.

Endpoint: `POST /providers/orders/pay-in/{code}/confirm-payment/`

Request | **Response (200 OK)**

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/confirm-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Cancelación

Si el usuario decide no pagar en el último momento o hay un problema con el efectivo, debes liberar la orden para que vuelva a estar disponible (o se cancele definitivamente, según las reglas de expiración de la orden).

Endpoint: `POST /providers/orders/pay-in/{code}/cancel-payment/`

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-in/9876543210/cancel-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "500.00",
    "price_currency": "MXN"
  }'
```

Resumen de Estados

Estado	Descripción	Acción Requerida del Proveedor
`READY`	La orden está pendiente de pago.	Verificar monto y llamar a `start-payment`.
`PAYMENT_STARTED`	La orden está en proceso de cobro.	Recibir dinero y llamar a `confirm-payment`.
`COMPLETED`	El dinero fue recaudado exitosamente.	Entregar comprobante al usuario.
`CANCELLED`	La orden fue cancelada.	No recibir dinero.

Retiros (Pay Out)

Este módulo permite a los Proveedores de Pago (Payment Providers) procesar solicitudes de retiro de efectivo iniciadas por usuarios de Pago46. El flujo está diseñado para garantizar la atomicidad de la transacción y evitar duplicidad en la entrega de fondos mediante un mecanismo de bloqueo (*locking*).

> **ENDPOINT BASE**
>
> Todas las rutas descritas a continuación son relativas a la URL base de la API: `/api/v1`

Registro de Redes de Pago

Antes de procesar transacciones, los proveedores deben registrar las redes de pago que utilizan. Cada red debe tener un identificador único (`network_id`) que se utilizará en todas las transiciones de estado.

Ejemplo de IDs de Red:

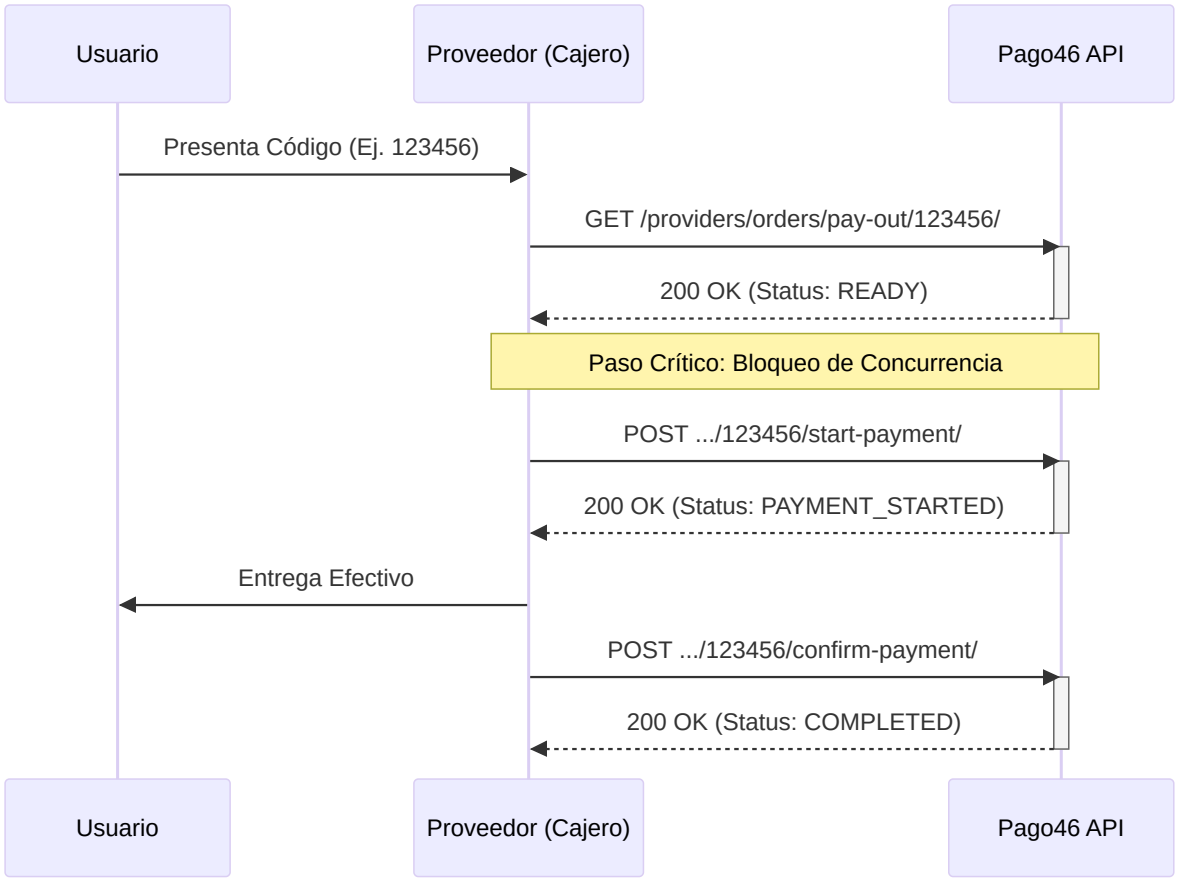
- `01`
- `oxxo_network`
- `seven_eleven_mx`
- `farmacia_abc`
- `network_123`

> **CONTACTO**
>
> Para registrar tus redes de pago, contacta al equipo de integración de Pago46. Los IDs que proporciones serán los que deberás usar en las solicitudes de cobro.

Ciclo de Vida de la Orden

El proceso se rige por un cambio estricto de estados para asegurar que el dinero solo se entregue una vez.

1. **Consulta (Check):** El proveedor verifica si el código entregado por el usuario es válido y la orden está en estado `READY`.
2. **Bloqueo (Start Payment):** El proveedor "toma" la orden. Esto cambia el estado a `PAYMENT_STARTED`. **En este punto, ningún otro proveedor puede procesar esta orden.**
3. **Confirmación (Confirm Payment):** Una vez entregado el dinero, el proveedor confirma la transacción, pasando la orden a `COMPLETED`.



---

1. Verificar Orden

El primer paso ocurre cuando el usuario presenta su código de retiro en el punto físico. Debes consultar los detalles de la orden para validar el monto y su disponibilidad.

Endpoint: `GET /providers/orders/pay-out/{code}/`

Parámetro	Ubicación	Descripción
`code`	Path	El código numérico o alfanumérico proporcionado por el usuario final.

Request | **Response (200 OK) - Order Ready** | **Response (200 OK) - Order Already In Progress**

```
curl -X GET "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>"
```

⚠ VALIDACIÓN DE ESTADO

Solo debes proceder al siguiente paso si el campo `status` es **`READY`**. Si recibes `CANCELLED`, `COMPLETED` o `EXPIRED`, debes informar al usuario que la orden no puede ser procesada.

2. Iniciar Pago (Bloqueo)

Este es el paso más crítico. Al ejecutar este endpoint, estás indicando que tienes la intención de pagar la orden. El sistema **bloqueará** la orden para tu proveedor y cambiará su estado a `PAYMENT_STARTED`.

Si otro proveedor intenta iniciar el pago de la misma orden simultáneamente, recibirá un error indicando que la orden ya está siendo procesada.

Endpoint: `POST /providers/orders/pay-out/{code}/start-payment/`

Request | **Response (200 OK)** | **Error (409 Conflict)**

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/start-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Campo	Tipo	Descripción
`network_id`	string	ID de la red de pago utilizada por el proveedor. **Requerido**.
`price`	string	Monto exacto que se va a entregar. Debe coincidir con el `price` de la orden. **Requerido**.
`price_currency`	string	Código de moneda (ej. `MXN`, `USD`). Debe coincidir con el `price_currency` de la orden. **Requerido**.

SUBREDES PARA PROVEEDORES

Como proveedor, debes suministrar el `network_id` que identifica la red específica utilizada para procesar esta transacción. Esto permite a Pago46 mantener un registro detallado de qué red procesó cada pago y proporcionar esta información al usuario final.

OPERACIÓN SEGURA

Una vez recibes el `200 OK` con status `PAYMENT_STARTED`, es seguro proceder a la entrega física del dinero o la gestión interna de fondos. La orden está reservada para ti.

3. Confirmar Pago (Finalización)

Una vez que el dinero ha sido entregado exitosamente al usuario (o la transacción interna ha finalizado), debes confirmar la operación para cerrar la orden definitivamente.

Endpoint: `POST /providers/orders/pay-out/{code}/confirm-payment/`

Request | Response (200 OK)

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/confirm-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Al recibir esta respuesta, notificaremos al usuario final que su transacción ha concluido exitosamente.

Cancelación (Opcional)

Si por alguna razón (insuficiencia de fondos en caja, error operativo), no puedes completar el pago después de haber ejecutado el `start-payment`, **debes** liberar la orden para no dejarla bloqueada indefinidamente.

Endpoint: `POST /providers/orders/pay-out/{code}/cancel-payment/`

Esto devolverá la orden a un estado donde (dependiendo de la lógica de negocio) podría ser cancelada definitivamente o reintentada.

```
curl -X POST "https://api.dev.pago46.io/api/v1/providers/orders/pay-out/1234567890/cancel-payment/" \
  -H "Provider-Key: <TU_PROVIDER_KEY>" \
  -H "Message-Date: <TIMESTAMP>" \
  -H "Message-Hash: <HMAC_SIGNATURE>" \
  -H "Content-Type: application/json" \
  -d '{
    "network_id": "network_123",
    "price": "1500.00",
    "price_currency": "MXN"
  }'
```

Resumen de Estados

Estado	Descripción	Acción Requerida del Proveedor
`READY`	La orden está lista para ser pagada.	Puede llamar a `start-payment`.
`PAYMENT_STARTED`	La orden está bloqueada por un proveedor.	Debe entregar el dinero y llamar a `confirm-payment`.
`COMPLETED`	El flujo terminó exitosamente.	No requiere acción. Historico.
`CANCELLED`	La orden fue anulada.	No entregar dinero.

Estado del Servicio

La **Status Page de Pago46** centraliza el estado operativo de nuestros servicios para que puedas validar disponibilidad, seguimiento de incidentes y mantenimientos programados en tiempo real.

URL oficial: `https://pago46.statuspage.io`

¿Qué información encontrarás?

- **Estado actual por componente:** Vista rápida de salud de servicios clave.
- **Incidentes activos e históricos:** Actualizaciones sobre degradaciones, interrupciones y resolución.
- **Mantenimientos programados:** Ventanas planificadas y su impacto esperado.

Suscripciones y alertas

Si operas en producción, te recomendamos suscribirte para recibir notificaciones proactivas.

Correo electrónico

1. Entra a `https://pago46.statuspage.io`.
2. Haz clic en **Subscribe to updates**.
3. Selecciona **Email**.
4. Ingresa tu correo y confirma la suscripción.

Slack

1. Entra a `https://pago46.statuspage.io`.
2. Haz clic en **Subscribe to updates**.
3. Selecciona **Slack**.
4. Autoriza la integración y elige el canal donde quieras recibir alertas.

> **RECOMENDACIÓN OPERATIVA**
>
> Suscribe al menos un correo de tu equipo de soporte/on-call y un canal de Slack compartido para visibilidad interna ante incidentes o mantenimientos.

Widget de estado en esta documentación

Además de la Status Page, este sitio de documentación ya tiene integrado el widget oficial de Statuspage.

Cuando exista un **downtime**, degradación o **mantenimiento**, podrás ver señales de estado también desde aquí para tener contexto operativo sin salir de la documentación.

Cuándo consultarla

- Si detectas errores inusuales en llamadas API o caídas de webhook.
- Antes de escalar un incidente, para validar si ya existe una comunicación oficial.

- Al planificar despliegues críticos en producción.

Si necesitas ayuda adicional, puedes escribirnos a contacto@pago46.com.

Guía de Diseño

Esta guía proporciona los recursos visuales y técnicos necesarios para integrar Pago46 en tu plataforma de manera profesional y consistente.

Identidad Visual

Mantener la coherencia visual ayuda a generar confianza en el usuario final durante el proceso de pago.

Colores Institucionales

Hemos preparado los valores exactos en diferentes formatos para que puedas copiarlos y pegarlos directamente en tu hoja de estilos o software de diseño.

Color	Uso Principal	HEX	RGB	HSL	CMYK
Verde	Botones de acción	`#28C339`	`rgb(40, 195, 57)`	`hsl(127, 66%, 46%)`	`79, 0, 71, 24`
Azul Oscuro	Textos y fondos	`#07214F`	`rgb(7, 33, 79)`	`hsl(218, 84%, 17%)`	`91, 58, 0, 69`
Blanco	Fondos claros	`#FFFFFF`	`rgb(255, 255, 255)`	`hsl(0, 0%, 100%)`	`0, 0, 0, 0`

Tipografía

Nuestra fuente oficial es **Lato**. Facilita la legibilidad en interfaces de pago y dispositivos móviles.

- **Uso en Botones:** Negrita (Bold), **16px**, centrado, letter-spacing: 0px.
- **Fallback:** En caso de no poder cargar **Lato**, utiliza sans-serif.

> **OBTENER TIPOGRAFÍA**
>
> Si no cuentas con la tipografía en tu proyecto, puedes descargarla o importarla gratuitamente a través de Google Fonts o encontrarla en Adobe Fonts.

Logotipos Base

Activos oficiales con fondo transparente. Utiliza la versión **SVG** siempre que sea posible para garantizar la máxima nitidez.

Versión	Color	SVG	PNG
Full	**Verde**	**Descargar:** svg	**Descargar:** sm · md · lg
	Azul	**Descargar:** svg	**Descargar:** sm · md · lg
	Blanco	**Descargar:** svg	**Descargar:** sm · md · lg

Versión	Color	SVG	PNG
Isotipo	Verde	**Descargar:** svg	**Descargar:** sm · md · lg
	Azul	**Descargar:** svg	**Descargar:** sm · md · lg
	Blanco	**Descargar:** svg	**Descargar:** sm · md · lg

Catálogo de Botones

Ofrecemos botones listos para usar en múltiples idiomas, tamaños y esquemas de color. Selecciona tu preferencia a continuación para obtener los links de descarga directos.

Español **English**

Variantes con los textos **"Efectivo"** y **"Pagar con"**.

Verde / Azul **Verde / Blanco** **Azul / Verde** **Azul / Blanco** **Blanco / Azul** **Blanco / Verde**

Tamaño	Full (Efectivo)	Full (Pagar con)	Iso (Efectivo)	Iso (Pagar con)
xs	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
sm	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
md	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG
lg	SVG / PNG	SVG / PNG	SVG / PNG	SVG / PNG

DESCARGA MASIVA

Puedes descargar el paquete completo con todas las variantes e idiomas en el siguiente enlace.

📦 Descargar Todos los Activos (.zip)

Ejemplos de Implementación

Copia y pega estos fragmentos de código para integrar el botón rápidamente en tu frontend. Estos ejemplos utilizan la variante **Efectivo + Logo Full (Fondo Verde / Texto y Logo Azul)** en su versión SVG.

HTML & CSS | **React (Tailwind)**

```
<button class="p46-btn-pay">
  <!-- Recomendamos usar el SVG directamente para mejor calidad -->
  <img
    src="/img/buttons/green-and-dark-blue-contrast/logo-full/md/button-cash-ES.svg"
    alt="Pagar con Pago46"
    class="p46-button-img"
  />
</button>

<style>
  .p46-btn-pay {
    background: none;
    border: none;
    padding: 0;
    cursor: pointer;
    transition: transform 0.2s ease, filter 0.2s ease;
  }

  .p46-btn-pay:hover {
    filter: brightness(90%);
  }

  .p46-btn-pay:active {
    transform: scale(0.98);
  }

  .p46-btn-pay:disabled {
    opacity: 0.5;
    cursor: not-allowed;
  }

  .p46-button-img {
    height: 48px; /* Ajusta según tu diseño */
    display: block;
  }
</style>
```

Restricciones y Malas Prácticas

Para evitar el rechazo de la integración durante el proceso de certificación, **no realices** las siguientes acciones:

- **Modificar Proporciones:** No estires ni comprimas el logo o el isotipo.
- **Alterar Colores:** No cambies el verde institucional por otros tonos de verde o colores ajenos a la marca.
- **Tipografía Incorrecta:** No utilices fuentes con serifa (serif) como Times New Roman en el botón.
- **Contraste Deficiente:** No utilices el logo verde sobre fondos de colores similares que dificulten la lectura.

CERTIFICACIÓN

Antes de pasar a producción, nuestro equipo revisará que la implementación del botón cumpla con estos lineamientos para asegurar la mejor tasa de conversión para tu comercio.