

POS Colombia API v1

Base URL: `https://poscolombia.com/api/v1`

REST API pública de **lectura y escritura** para integraciones de terceros. Un desarrollador externo puede leer y también **crear/editar** productos, categorías, clientes e inventario, y **registrar ventas desde su propio sistema** — sin tocar el dashboard. Trae **entorno de pruebas (sandbox)** al estilo Stripe y un servidor **MCP** para agentes IA.

Ningún competidor colombiano (Alegra, Siigo, Loggro, Helisa, ContaPyme, Aliaddo) tiene una API pública de escritura documentada — es un diferenciador defendible para POS Colombia.

Status

- **v1** — Estable. Read + write de productos, categorías, órdenes, inventario y clientes. Los cambios breaking se anuncian con 90 días de aviso.
- Cambios **non-breaking** (endpoints nuevos, campos opcionales adicionales) entran sin aviso previo.

Disponibilidad por plan

⚠ **La API pública está disponible SOLO en planes Profesional, Empresa y Cadena, y solo después de activar el pago.** Los usuarios en prueba (trial) no pueden generar llaves — es una defensa contra abuso (trial → genera llave → descarga o inyecta datos → cancela).

Si tu plan es Gratis / Emprendedor / Negocio, podés leer esta documentación, pero `/settings/api-keys` te muestra un paywall con link a upgrade. Los endpoints `/api/v1/*` responden **402 api_access_denied** a cualquier llave si el org baja de plan o su suscripción deja de estar activa después de emitirla.

Plan	Rate limit	Cupo de escritura/mes	Webhooks/mes
Profesional	120 req/min	10.000 escrituras	10.000 entregas
Empresa	300 req/min	50.000 escrituras	50.000 entregas
Cadena	600 req/min	Ilimitado	Ilimitado

El rate limit es por **llave**; el cupo de escritura es por **org** y por mes calendario (hora Bogotá, UTC-5). Las llaves **sandbox** (`pk_test_...`) **no consumen** cupo de escritura ni topan.

Probar durante la prueba gratis. Si el negocio está en la prueba del plan Profesional (o superior), el dueño o un administrador ya puede crear llaves de **sandbox** (`pk_test_...`) en Configuración → API Keys e integrar de punta a punta contra la org de pruebas. Las llaves de **producción** (`pk_live_...`) se habilitan cuando se activa el plan pagado.

Autenticación

Header obligatorio en toda request:

```
X-API-Key: pk_live_<48 chars base32 lowercase>
```

- **pk_live_...** → producción. Lee y escribe contra los datos **reales** del negocio.
- **pk_test_...** → sandbox. Lee y escribe contra una **org de pruebas aislada** (ver "Sandbox vs Producción").

Generás y revocás llaves desde el dashboard: **Configuración** → **API Keys** (`/settings/api-keys`). Solo owner/admin del organization puede hacerlo. Cada llave:

- Tiene scopes específicos (ver tabla de scopes abajo).
- Puede tener fecha de expiración opcional.
- Es revocable instantáneamente.
- Tiene un `mode` fijo (`live` o `test`).

⚠ **El secreto completo se muestra UNA SOLA VEZ al crear la llave.** Después solo verás el prefijo (`pk_live_abcd1234...`). Si lo perdés, generará una nueva.

Scopes

Scope	Permite
<code>orders:read</code>	Leer órdenes, resúmenes de ventas, top productos
<code>orders:write</code>	Registrar ventas / crear órdenes
<code>products:read</code>	Leer productos
<code>products:write</code>	Crear y editar productos
<code>categories:write</code>	Crear y editar categorías
<code>inventory:write</code>	Crear/editar insumos y ajustar stock
<code>customers:read</code>	Leer clientes
<code>customers:write</code>	Crear y editar clientes

Una llave puede tener cualquier combinación. Pedí solo los scopes que tu integración necesita (mínimo privilegio). En v1, categorías e inventario son **write-only** (aún no hay endpoint de lectura dedicado; el catálogo se lee vía `GET /api/v1/products`).

Ejemplo de request

```
curl https://poscolombia.com/api/v1/orders \
-H "X-API-Key: pk_live_abcd12345xyzbase32secretexample48charsxxxxxxxxxxxxx"
```

```
// Node 18+ (fetch nativo) o navegador
const res = await fetch("https://poscolombia.com/api/v1/orders", {
  headers: { "X-API-Key": process.env.POS_API_KEY },
});
const { data, pagination } = await res.json();
```

Headers de respuesta

Toda respuesta de la API incluye:

```
X-API-Version: v1
Cache-Control: private, no-store
```

Códigos de error

Las respuestas de error tienen forma `{ "error": "<code>", "detail?": "..." }`.

Status	error	Significado
401	<code>missing_api_key</code>	Falta el header <code>X-API-Key</code>
401	<code>invalid_api_key_format</code>	El formato no es <code>pk_live_...</code> / <code>pk_test_...</code> de 48 chars base32
401	<code>invalid_api_key</code>	La llave no existe o no matchea
401	<code>api_key_revoked</code>	La llave fue revocada en el dashboard

Status	error	Significado
401	api_key_expired	La llave pasó su expires_at
402	api_access_denied	El plan no permite API o la suscripción no está activa. Ver reason .
403	insufficient_scope	La llave no tiene el scope requerido (detail dice cuál falta)
400	validation_error	El body no pasó la validación (detail con el campo/motivo)
404	not_found	El recurso no existe en tu org (o la API pública está deshabilitada)
409	conflict	Choque de unicidad (p. ej. SKU repetido)
429	rate_limited	Excediste el rate limit. Header Retry-After en segundos.
429	write_quota_exceeded	Alcanzaste tu cupo mensual de escritura (used , cap). Pasá a Empresa.
500	query_failed / db_error	Error interno (registrado en Sentry)

402 api_access_denied trae reason : plan_too_low · trial_only · subscription_inactive · subscription_expired · no_subscription .

Sandbox vs Producción

Modelo estilo Stripe: la API tiene dos entornos, elegidos por el **prefijo de la llave**.

- **pk_live_...** opera contra la **org real**. Todo lo que escribas es producción (ventas reales, inventario real, reportes, facturación).
- **pk_test_...** opera contra una **org sandbox aislada** de tu cuenta — un organization_id distinto. Nada de lo que escribas con una llave test puede rozar tus datos productivos. El sandbox:
 - No aparece en reportes, cierres de caja, conteo de clientes ni billing.
 - Simula la facturación electrónica (**nunca** emite DIAN real).
 - **No consume** tu cupo mensual de escritura.

Usá sandbox para desarrollar y probar tu integración de punta a punta, y recién pasá a pk_live_... cuando esté lista.

Reset del sandbox

```
POST /api/v1/sandbox/reset
```

Borra la data operativa de tu org sandbox (órdenes, clientes, productos, categorías de prueba) y la deja limpia. **Solo funciona con una llave pk_test_...** . Nunca puede vaciar una org real.

```
curl -X POST https://poscolombia.com/api/v1/sandbox/reset \
-H "X-API-Key: pk_test..."
```

Idempotencia

Al **registrar ventas** (POST /api/v1/orders), el clientReference es **obligatorio**: un UUID que generarás vos, único por intento de venta. Sin él, el server responde **400 client_reference_required** . Si repetís la request con el mismo clientReference (por un timeout, un reintento de red o dos procesos concurrentes del mismo intento), el server devuelve **la MISMA orden** en vez de duplicar la venta. En una carrera concurrente exacta, el request perdedor recibe **409 idempotency_conflict** (reintentá en un momento).

```
const clientReference = crypto.randomUUID(); // uno por intento de venta
async function registrarVenta() {
  return fetch("https://poscolombia.com/api/v1/orders", {
    method: "POST",
    headers: { "X-API-Key": KEY, "Content-Type": "application/json" },
```

```
body: JSON.stringify({ clientReference, orderType: "local", paymentMethod: "cash", items: [/*...*/] }),
});
}
// reintentar registrarVenta() es seguro: nunca cobra dos veces.
```

Para los demás endpoints de escritura (crear producto, categoría, cliente, ajuste de inventario) la seguridad ante reintentos la das con tu propia lógica (p. ej. no reintentar un 201 que ya recibiste); recomendamos usar SKU/NIT como clave natural para detectar duplicados con un GET previo.

Paginación

Todos los list endpoints aceptan:

- `limit` (default 50, max 200)
- `offset` (default 0)

Respuesta:

```
{
  "data": [ /* ... */ ],
  "pagination": { "limit": 50, "offset": 0, "total": 287, "has_more": true }
}
```

Multi-tenant y seguridad

El `organization_id` sale **siempre de la llave**, nunca del body ni del query string. Es imposible que una llave lea o escriba datos de otra org. Cualquier `id` que mandes en el body y que apunte a otro org se ignora o se rechaza.

Endpoints

Órdenes / Ventas

GET /api/v1/orders

Lista de órdenes del org, ordenada por `created_at` DESC. Excluye cotizaciones (`quote`).

Scope: `orders:read`

Query params: `limit`, `offset`, `status` (`open|pending|preparing|ready|delivered|cancelled`), `from` (ISO 8601, `created_at >= from`), `to` (ISO 8601, `created_at < to`, **exclusivo**).

```
{
  "data": [
    {
      "id": "uuid",
      "order_number": "ORD-2026-001",
      "status": "delivered",
      "customer_name": "Juan Perez",
      "customer_phone": "+573001234567",
      "total_amount": 45000,
      "subtotal": 38000,
      "tax_amount": 7000,
      "payment_method": "card",
```

```
    "order_type": "local",
    "table_number": 5,
    "created_at": "2026-05-11T10:00:00Z",
    "updated_at": "2026-05-11T10:30:00Z"
  }
],
"pagination": { "limit": 50, "offset": 0, "total": 287, "has_more": true }
}
```

```
curl "https://poscolombia.com/api/v1/orders?status=delivered&from=2026-04-11T00:00:00Z" \
-H "X-API-Key: pk_live_..."
```

GET /api/v1/orders/summary

Totales de **ventas** del rango + desglose por día. No pagina. Usa la misma definición de venta que el cierre de caja (`status != cancelled` , sin consumo de empleados, sin cotizaciones).

Scope: `orders:read`

Query params: `from` / `to` (ISO 8601, opcionales; default últimos 30 días; rango **inclusivo en ambos extremos**).

```
{
  "range": { "from": "2026-08-15T00:00:00.000Z", "to": "2026-09-14T00:00:00.000Z" },
  "summary": { "total_sales": 4820000, "order_count": 214, "average_ticket": 22523 },
  "by_day": [
    { "date": "2026-08-15", "total": 180000, "count": 8 },
    { "date": "2026-08-16", "total": 240000, "count": 11 }
  ],
  "truncated": false
}
```

`by_day` usa el día calendario de Bogotá (UTC-5). `truncated: true` si el rango superó 20.000 órdenes (acortó el rango).

```
curl "https://poscolombia.com/api/v1/orders/summary?from=2026-09-01T00:00:00Z&to=2026-09-30T23:59:59Z" \
-H "X-API-Key: pk_live_..."
```

POST /api/v1/orders

Registra una venta desde tu sistema. El server calcula totales e impuestos con la config del org, descuenta inventario y (según la config) la manda a cocina — igual que una venta hecha en el POS.

Scope: `orders:write` · **Consume 1** de tu cupo de escritura.

Body (campos principales):

Campo	Tipo	Notas
<code>orderType</code>	string	Requerido. <code>local</code> <code>para_llevar</code> <code>domicilio</code>
<code>items[]</code>	array	Requerido (al menos una línea): { <code>productId</code> , <code>quantity</code> , <code>notes?</code> , <code>modifiers?</code> }
<code>paymentMethod</code>	string	Requerido. Exactamente uno de: <code>cash</code> <code>card</code> <code>transfer</code> <code>nequi</code> <code>daviplata</code>
<code>clientReference</code>	uuid	Requerido. Idempotencia (ver arriba): un uuid por intento de venta.
<code>customerId</code>	string?	Cliente del directorio (opcional)
<code>customerName</code>	string?	Nombre del cliente ocasional (si no mandás <code>customerId</code>)
<code>customerPhone</code>	string?	Teléfono del cliente ocasional. Acepta internacional con indicativo (<code>+1...</code>); se guarda en dígitos.

Campo	Tipo	Notas
notes	string?	Nota general del pedido
autoAccept	boolean?	true = entra directo a cocina (preparing). Sin él nace pending y el POS muestra Aceptar / Rechazar.
paid	boolean?	true = ya la cobraste (pago en línea): nace con payment_status: "paid" .
paymentReference	string?	Referencia de la transacción (≤120). Queda en la orden y en el cierre.
deliveryAddress	string?	Domicilio: dirección
deliveryNeighborhood	string?	Domicilio: barrio
deliveryReferences	string?	Domicilio: indicaciones (torre, apto, portería...)
deliveryLat / deliveryLng	number?	Domicilio: coordenadas; con ellas la tarifa sale de la zona donde cae el punto
deliveryZoneId	uuid?	Domicilio: zona de reparto del negocio (alternativa a las coordenadas)

Líneas: cada ítem acepta notes ("sin cebolla") y modifiers (adiciones y opciones: [{ "modifierId": "uuid" }]). Los ids salen de GET /api/v1/products?include=modifiers . El precio de cada adición lo pone el server desde el catálogo; también valida que la opción sea de ese producto y respeta los mínimos/máximos del grupo.

Tarifa de envío: la calcula el server con las **zonas de reparto** que el negocio tiene configuradas (por coordenadas o deliveryZoneId); nunca se toma del body. Si la dirección no cae en ninguna zona, la orden entra con delivery_fee_pending: true y el negocio la cotiza desde el POS.

Domicilio en ruta: la orden trae delivery_status (pending , assigned con repartidor, picked_up en ruta, delivered , returned); cuando el negocio toca "Va en camino" pasa a picked_up y llega order.updated .

Estado al entrar: sin autoAccept , la orden nace pending (el POS avisa con Aceptar / Rechazar y la cocina la ve en Pendientes). Con autoAccept: true nace preparing , sin paso de aceptación.

Necesitas al menos una línea en items . El **dinero nunca se confía del body**: los precios salen del producto en tu catálogo, no de lo que mandes.

No soportado en v1 (el body que los use recibe 400): cuentas abiertas, consumo interno / ventas de práctica, líneas libres (customItems — mano de obra), fiado (credito), pago mixto (mixed), cupones de descuento, variantes (talla/color) y pizza mitad y mitad. Se rechazan explícito en vez de cobrar de menos en silencio.

Respuesta:

- 201 { "data": { ... } } — venta creada.
- 200 { "data": { ... }, "replayed": true } — reintento con el mismo clientReference : se devuelve la MISMA orden, no se duplica la venta.
- 409 { "error": "idempotency_conflict" } — carrera concurrente: otro request con el mismo clientReference está en curso. Reintentá en un momento.
- 400 — falta clientReference (client_reference_required), falta o no está permitido el paymentMethod (payment_method_required / unsupported_payment_method), o el body usa una función no soportada en v1 (unsupported_in_v1 , ver arriba).

```
{ "data": { "id": "uuid", "order_number": 45, "status": "pending", "payment_status": "paid",
"payment_reference": "txn-123", "delivery_fee": 5000, "delivery_fee_pending": false, "total_amount": 45000,
"created_at": "2026-09-18T14:00:00Z" } }
```

```
curl -X POST https://poscolombia.com/api/v1/orders \
-H "X-API-Key: pk_live..." -H "Content-Type: application/json" \
-d '{
  "clientReference": "8f1c...uuid",
  "orderType": "para_llevar",
  "items": [{ "productId": "prod-uuid", "quantity": 2 }],
```

```
"paymentMethod": "cash"
},'
```

```
await fetch("https://poscolombia.com/api/v1/orders", {
  method: "POST",
  headers: { "X-API-Key": KEY, "Content-Type": "application/json" },
  body: JSON.stringify({
    clientReference: crypto.randomUUID(),
    orderType: "domicilio",
    items: [{ productId: "prod-uuid", quantity: 1 }],
    paymentMethod: "transfer",
    customerId: "cust-uuid",
  }),
});
```

GET /api/v1/orders/{id}

Una venta con sus líneas, estado de cocina (`status`) y de pago (`payment_status`). **Scope:** `orders:read` . `404` si no existe en tu negocio.

POST /api/v1/orders/{id}/pay

Marca como **pagada** una venta **creada por la API** (por ejemplo, cuando tu pasarela confirma el cobro después de crear la orden). No cambia el estado de cocina. **Scope:** `orders:write` · **Consume 1** del cupo.

Campo	Tipo	Notas
<code>paymentMethod</code>	string?	Si cambió: <code>cash</code> <code>card</code> <code>transfer</code> <code>nequi</code> <code>daviplata</code>
<code>paymentReference</code>	string?	Referencia de la transacción (≤120)

- `200 { data }` — quedó pagada. Si ya estaba pagada: `200 { data, replayed: true }` (no consume cupo).
- `404 order_not_found` — no existe o no fue creada por la API (las ventas de la caja no se cobran por API).
- `409 order_cancelled` — la orden está anulada.
- `409 delivery_fee_pending` — la tarifa de envío está por cotizar; cuando el negocio la fije, vuelve a llamar. Lo mismo aplica a `paid: true` al crear: si la tarifa queda pendiente, la orden entra sin marcar pagada.

Para cobros en línea usa `card` o `transfer` como método: así la venta no suma al efectivo esperado del cierre de caja.

POST /api/v1/orders/{id}/cancel

Anula una venta **creada por la API** y **devuelve el inventario** que había descontado. **Scope:** `orders:write` · **Consume 1** del cupo.

Campo	Tipo	Notas
<code>reason</code>	string?	Motivo (≤300). Queda en la orden.
<code>confirmPaidVoid</code>	boolean?	Obligatorio en <code>true</code> si la orden ya está pagada (la devolución del dinero la haces en tu pasarela).

- `200 { data }` — anulada. Si ya estaba anulada: `200 { data, replayed: true }`.
- `404 order_not_found` — no existe o no fue creada por la API.
- `409 order_delivered` — ya se entregó; se anula desde el POS.
- `409 invoice_active` — tiene factura electrónica vigente; primero la nota crédito.
- `409 requires_void_confirm` — está pagada; reenvía con `confirmPaidVoid: true`.

Productos

GET /api/v1/products

Lista de productos, ordenada por `name` ASC.

Scope: `products:read` · **Query:** `limit`, `offset`, `active` (bool, default `true`), `category_id` (uuid), `include=modifiers` (agrega `modifier_groups` con las adiciones/opciones de cada producto y sus ids).

- `is_active: false` = el negocio lo apagó (no se vende). Por defecto la lista trae solo activos.
- `sold_out: true` = controla stock propio y llegó a 0. Los preparados por receta no se marcan: su disponibilidad la maneja el negocio con `is_active`.

```
{
  "data": [
    {
      "id": "uuid", "name": "Bandeja paisa", "description": "...",
      "price": 22000, "sku": "BP-001", "category_id": "uuid",
      "category_name": "Platos fuertes", "image_url": "https://.../bandeja.webp",
      "is_active": true, "stock": null, "sold_out": false, "tax_category": "consumo",
      "created_at": "2025-01-15T08:00:00Z", "updated_at": "2026-04-20T14:00:00Z"
    }
  ],
  "pagination": { "limit": 50, "offset": 0, "total": 120, "has_more": true }
}
```

GET /api/v1/products/top

Top productos por cantidad vendida e ingreso en el rango.

Scope: `orders:read` o `products:read`. **Query:** `from`, `to` (como `orders/summary`), `limit` (1-50, default 10), `sort` (`units` | `revenue`, default `units`).

```
{
  "range": { "from": "2026-08-15T00:00:00.000Z", "to": "2026-09-14T00:00:00.000Z" },
  "sort": "units",
  "products": [
    { "product_id": "uuid-1", "product_name": "Hamburguesa clásica", "units": 320, "revenue": 6400000 }
  ],
  "truncated": false
}
```

POST /api/v1/products

Crea un producto.

Scope: `products:write` · **Consume 1** de cupo.

Body (principales): `name` (req, ≤100), `price` (req, > 0), `description?`, `categoryId?` (uuid), `sku?` (≤64), `barcode?` (≤128), `taxCategory?` (`standard` | `reduced` | `exempt` | `consumo`), `isActive?` (default `true`), `isAvailable?`, `stock?` (`retail`), `costPerUnit?`, `imageUrl?`. Ver el esquema completo en el OpenAPI.

Respuesta 201: `{ "data": { /* producto creado */ } }`

```
curl -X POST https://poscolombia.com/api/v1/products \
  -H "X-API-Key: pk_live..." -H "Content-Type: application/json" \
  -d '{ "name": "Limonada", "price": 6000, "categoryId": "cat-uuid", "taxCategory": "consumo" }'
```

PATCH /api/v1/products/{id}

Edita un producto existente de tu org. Todos los campos son opcionales (solo se actualiza lo que mandes).

Scope: `products:write` · **Consume 1** de cupo. 404 si el `id` no es de tu org.

```
curl -X PATCH https://poscolombia.com/api/v1/products/prod-uuid \
  -H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
  -d '{"price": 6500, "isActive": true}'
```

```
await fetch(`https://poscolombia.com/api/v1/products/${id}`, {
  method: "PATCH",
  headers: { "X-API-Key": KEY, "Content-Type": "application/json" },
  body: JSON.stringify({ price: 6500 }),
});
```

Categorías

GET /api/v1/categories

Categorías en el orden del POS. **Scope:** `products:read` · **Query:** `active` (bool, default `true`). Con `category_id` de cada producto (o `category_name`, que ya viene en `GET /products`) armas la carta por secciones.

```
{ "data": [ { "id": "uuid", "name": "Rolls", "display_order": 1, "is_active": true, "created_at": "2026-09-01T12:00:00Z" } ] }
```

Write-only en v1 (para leer el catálogo usá `GET /api/v1/products`, que trae `category_id`).

POST /api/v1/categories

Scope: `categories:write` · **Consume 1** de cupo. `409` si el nombre ya existe.

Body: `name` (req, ≤50), `description?`, `color?`, `icon?`, `sortOrder?` (int, default 0), `isActive?` (default `true`).

Respuesta 201: `{ "data": { /* categoría creada */ } }`

```
curl -X POST https://poscolombia.com/api/v1/categories \
  -H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
  -d '{"name": "Postres", "sortOrder": 3}'
```

PATCH /api/v1/categories/{id}

Edita una categoría. Campos opcionales. **Scope:** `categories:write` · `404` si no es de tu org.

Inventario (insumos)

Modela los **insumos** del negocio. Los metadatos (nombre, costo, mínimo, proveedor) se crean/editan en `/api/v1/ingredients`; el **stock nunca se sobrescribe** con un valor absoluto por `PATCH` — se mueve con un **ajuste** (delta firmado o conteo físico) en `/api/v1/inventory/adjust`, que conserva la trazabilidad y respeta las bodegas si el org las usa.

POST /api/v1/ingredients

Crea un insumo.

Scope: `inventory:write` · **Consume 1** de cupo.

Body: `name` (req, ≤100), `unit` (req), `category?`, `currentStock?` (default 0), `minStock?` (default 0), `costPerUnit?` (default 0), `supplier?`, `isActive?` (default `true`).

Anti-duplicados: si ya existe un insumo activo con el mismo nombre (normalizado) → 409 . Si existe pero está archivado, se reactiva y actualiza en vez de crear una copia.

Respuesta 201 : { "data": { "id": "uuid", "name": "Queso mozzarella", "unit": "kg", "current_stock": 12, "min_stock": 3, "cost_per_unit": 18000, "is_active": true } }

```
curl -X POST https://poscolombia.com/api/v1/ingredients \
-H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
-d '{ "name": "Queso mozzarella", "unit": "kg", "currentStock": 12, "minStock": 3, "costPerUnit": 18000 }'
```

PATCH /api/v1/ingredients

Edita un insumo (nombre, costo, mínimo, proveedor...). El `id` va en el **body**. El `currentStock` se **ignora** acá — para mover stock usá el ajuste.

Scope: `inventory:write` · **Consume 1** de cupo. 404 si el `id` no es de tu org.

```
curl -X PATCH https://poscolombia.com/api/v1/ingredients \
-H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
-d '{ "id": "ing-uuid", "costPerUnit": 19500, "minStock": 4 }'
```

POST /api/v1/inventory/adjust

Ajusta el stock de un insumo. Elegí **una** de dos semánticas:

- `delta` (number firmado): ajuste relativo (positivo = entra, negativo = sale). Sin piso — vender con stock negativo es flujo normal.
- `countedStock` (number ≥ 0): conteo físico absoluto; el stock se fija a ese valor (piso 0) y se registra la varianza.

Scope: `inventory:write` · **Consume 1** de cupo. Body también acepta `reason?` (string) y `locationId?` (uuid de la bodega, si el org usa bodegas; si se omite, la bodega por defecto). 404 si el insumo no es de tu org.

Respuesta 200 :

```
{
  "data": { "id": "ing-uuid", "name": "Harina", "current_stock": 37.5 },
  "adjustment": { "previous_stock": 40, "new_stock": 37.5, "delta": -2.5, "mode": "delta", "location_id": null }
}
```

```
curl -X POST https://poscolombia.com/api/v1/inventory/adjust \
-H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
-d '{ "ingredientId": "ing-uuid", "delta": -2.5, "notes": "merma" }'
```

```
// Conteo físico absoluto en vez de delta
await fetch("https://poscolombia.com/api/v1/inventory/adjust", {
  method: "POST",
  headers: { "X-API-Key": KEY, "Content-Type": "application/json" },
  body: JSON.stringify({ ingredientId: "ing-uuid", countedStock: 35 }),
});
```

Cientes

GET /api/v1/customers

Lista de clientes, ordenada por `created_at` DESC.

Scope: customers:read · **Query:** limit , offset , search (substring Cl en name o phone).

```
{
  "data": [
    { "id": "uuid", "name": "Juan Perez", "phone": "+573001234567", "email": "juan@example.com",
      "total_orders": 12, "total_spent": 540000, "last_order_at": "2026-05-10T18:00:00Z", "created_at": "2025-08-01T12:00:00Z" }
  ],
  "pagination": { "limit": 50, "offset": 0, "total": 340, "has_more": true }
}
```

POST /api/v1/customers

Scope: customers:write · **Consume 1** de cupo.

Body: name (req, ≤100), idType? (NIT | CC | CE | PP | consumidor_final , default consumidor_final), idNumber? (documento CO, ≤15 dígitos), email?, phone? (celular colombiano a 10 dígitos, fijo, o internacional con indicativo: +1 305 555 0100 ; el internacional se guarda como dígitos con indicativo, 13055550100), address?, city?, notes?, contactName?, contactPhone?. Un phone o email ya registrado en tu org → 409 .

Respuesta 201 : { "data": { /* cliente creado */ } }

```
curl -X POST https://poscolombia.com/api/v1/customers \
-H "X-API-Key: pk_live_..." -H "Content-Type: application/json" \
-d '{ "name": "María Gómez", "idType": "CC", "idNumber": "1023456789", "phone": "3001234567" }'
```

PATCH /api/v1/customers/{id}

Edita un cliente. Campos opcionales. **Scope:** customers:write . 404 si no es de tu org.

Rate limits y cupos

- **Rate limit por llave:** Profesional 120 · Empresa 300 · Cadena 600 req/min. Al superarlo → 429 rate_limited con header Retry-After .
- **Cupo de escritura por org/mes:** Profesional 10.000 · Empresa 50.000 · Cadena ilimitado. Cada escritura exitosa con una llave pk_live_... cuenta 1. **Qué cuenta:** una llamada de escritura = 1, sin importar lo que haga por dentro. Crear una venta es 1 aunque descuente inventario y cree o enlace al cliente; marcarla pagada es otra (1) y anularla otra (1). Crear un cliente con POST /customers es 1. Las lecturas (GET), los reintentos idempotentes (replayed) y todo lo del sandbox **no cuentan**. Al agotarlo → 429 write_quota_exceeded con { used, cap } . Se reinicia el 1° de cada mes (hora Bogotá). Las llaves sandbox no consumen cupo.

Buenas prácticas: respetá Retry-After , hacé backoff exponencial ante 429 , y para cargas grandes espaciá las escrituras. Para límites custom escribí a gerencia@poscolombia.com .

Webhooks (Profesional, Empresa y Cadena)

Los webhooks salientes están disponibles desde el plan **Profesional**, con un tope mensual de entregas: Profesional 10.000 · Empresa 50.000 · Cadena sin tope (mes calendario, hora Bogotá). Al llegar al tope, los eventos del resto del mes no se envían; puedes seguir consultando GET /orders . En vez de hacer polling, POS Colombia hace POST a tu endpoint cuando ocurre un evento.

Eventos que llegan hoy:

- order.created — venta nueva (en la caja o por la API).
- order.updated — la orden cambió de estado (aceptada, en preparación, lista, entregada) en el POS o en la cocina, o cambió el domicilio: repartidor asignado, **en ruta** (delivery_status: "picked_up" , botón "Va en camino") o tarifa de envío cotizada.
- order.paid — quedó pagada (en el POS, con paid: true o con POST /orders/{id}/pay).

- `order.cancelled` — se anuló (en el POS o con `POST /orders/{id}/cancel`).
- `product.created` / `product.updated` — se creó, editó (precio, nombre, foto, categoría, activo) o archivó un producto en la pantalla Productos o por la API. Un producto archivado llega con `deleted: true`. Las ventas no disparan `product.updated` (el stock que baja con cada venta no genera eventos).

El payload de órdenes trae `id`, `order_number`, `status`, `delivery_status`, `payment_status`, `payment_method`, `payment_reference`, `total_amount`, `delivery_fee`, `order_type`, `source`, `client_reference` (tu referencia si la creaste por API) y fechas. El de productos trae los mismos campos de `GET /products` (incluye `category_name`, `image_url` y `sold_out`).

En preparación (puedes suscribirte ya; empiezan a llegar cuando se activen): `customer.created`, `invoice.dian_emitted`.

Filtrar lo que te llega. Al crear la suscripción (`POST /webhooks`) eliges los `events` que quieres. Con `onlyApiOrders: true`, los eventos de órdenes solo llegan para las órdenes que creaste por la API (no las de la caja), así no gastas el tope en ventas de mostrador. Los eventos de productos no se filtran.

Cómo cuenta el tope. Cada evento que se envía a una suscripción cuenta 1 entrega (cada evento es su propio POST, aunque salgan varios en la misma pasada). Si tienes dos suscripciones al mismo evento, cuenta 2. Lo que filtras con `events` u `onlyApiOrders` no cuenta.

Los eventos salen cada 2 minutos como máximo.

Payload:

```
{
  "id": "delivery-uuid",
  "event": "order.created",
  "data": { "id": "order-uuid", "order_number": "ORD-2026-045", "total_amount": 45000 },
  "timestamp": "2026-09-18T14:00:00Z"
}
```

Cabeceras de cada entrega. Todos los webhooks salientes llevan:

- `X-POSColombia-Signature: sha256=<hmac>` — HMAC-SHA256 del **cuerpo crudo** con tu secreto de firma (lo ves al crear la suscripción). **Verificalo SIEMPRE** antes de procesar.
- `X-POSColombia-Event: <tipo_de_evento>` — p. ej. `order.created`.
- `X-POSColombia-Delivery: <id_de_entrega>` — id único de esta entrega.
- `X-POSColombia-Idempotency-Key: <evento>:<entity_id>:<ts>` — para deduplicar entregas repetidas (reintentos) con esta clave.

Verificá la firma con el header `X-POSColombia-Signature`:

```
import crypto from "node:crypto";
function verify(rawBody, header, secret) {
  const expected = "sha256=" + crypto.createHmac("sha256", secret).update(rawBody).digest("hex");
  const a = Buffer.from(expected), b = Buffer.from(header || "");
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}
```

Respondé `2xx` para confirmar la entrega. Reintentamos con backoff ante `5xx` /timeouts. Si llegaste al tope del mes, usá polling: `GET /api/v1/orders?from=<último_visto>` cada 30 a 60 segundos.

MCP (Model Context Protocol)

POS Colombia trae un servidor **MCP** para que agentes IA (Claude Desktop, Claude Code, etc.) consulten y operen tu negocio con lenguaje natural, respetando exactamente los scopes y cupos de tu llave.

No es un endpoint HTTP hospedado. Es un paquete que corre como **proceso local (stdio)** en tu máquina (`packages/pos-mcp`): el host MCP lo arranca por vos y habla con la API v1 usando tu llave. Instalación y build en el README del paquete (`npm install && npm run build` → `dist/index.js`).

- **Auth:** variable de entorno `POS_API_KEY` con tu llave `pk_live_...` (o `pk_test_...` para sandbox).
- **Herramientas:** espejo de los endpoints v1 (leer órdenes/resúmenes/ productos/clientes, crear ventas, crear/editar productos y categorías, ajustar inventario) — cada una gateada por su scope.

Ejemplo de configuración (Claude Desktop / Claude Code — usá la ruta ABSOLUTA a `dist/index.js`):

```
{
  "mcpServers": {
    "pos-colombia": {
      "command": "node",
      "args": ["/ruta/absoluta/a/packages/pos-mcp/dist/index.js"],
      "env": { "POS_API_KEY": "pk_live_..." }
    }
  }
}
```

Probá primero con una llave `pk_test_...` para que el agente opere en sandbox sin tocar tu operación real.

Privacidad y Habeas Data

POS Colombia opera bajo la Ley 1581 de 2012 (Habeas Data Colombia) y las políticas en `/privacy`.

- **El owner consintió al emitir la llave.** Generar una API key implica el consentimiento del responsable del tratamiento (dueño del org) para compartir esa data con los integradores que reciban la llave.
- **Solo Profesional+ post-pago.** Los usuarios en prueba no pueden generar llaves.
- **PII mínima:** `/v1/customers` expone name, phone, email y agregados; **no** expone `id_number` ni dirección salvo scope opt-in con audit adicional (contactá soporte). `/v1/products` no tiene PII.
- **Sos custodian de la data** después de la llamada: mantené tu copia mínima, cifrada en tránsito y at-rest, y borrarla cuando ya no sea necesaria.
- **Revocación inmediata.** El owner revoca cualquier llave desde `/settings/api-keys`; futuras requests devuelven 401 al instante.
- **Auditoría interna.** Cada escritura exitosa queda registrada (org, acción, recurso, `key_id`, origen `api_v1`), y cada emisión/revocación de llave se loggea. Ante abuso podemos reconstruir el origen.

Para incidentes de seguridad o auditoría de uso: `gerencia@poscolombia.com`.

Soporte

- **Bugs / feedback:** `gerencia@poscolombia.com`
- **WhatsApp soporte:** +57 300 617 1431 (L-V 8am-8pm CO, respuesta <30 min)
- **Referencia OpenAPI:** `/docs/api/openapi.yaml` (importable a Postman/Insomnia)

Versionado

Garantizamos backward-compatibility dentro de v1. Los cambios breaking solo entran en un major version bump, con 90 días de aviso. Los cambios non-breaking (endpoints nuevos, campos opcionales) entran sin aviso.