



Documentación Técnica — API de Sincronización de Inventario



Versión: 3.2 — Liquidación en `financials` y fee del comprador en Cashea

Última actualización: Septiembre 2026

Base URL: Se proporcionará al activar la integración

Autenticación

Formato de la API Key

Permisos

Conceptos Clave

Identificadores de Variante

Resolución Inteligente de Variantes

Endpoints

Sincronizar Stock

Headers

Request Body

Campos del request

Operaciones disponibles

Response exitosa (200)

Estados por ítem

Consultar Stock

Headers

Parámetros de consulta (Query Params)

Ejemplo de solicitud

Response exitosa (200)

Actualizar Precios

Headers

[Request Body](#)

[Campos del request](#)

[Response exitosa \(200\)](#)

[Consultar Órdenes \(Reconciliación\)](#)

[¿Para qué sirve?](#)

[Headers](#)

[Parámetros de consulta \(Query Params\)](#)

[Ejemplo de solicitud](#)

[Response exitosa \(200\)](#)

[Chat / CRM](#)

[Listar mensajes](#)

[Query Params](#)

[Response exitosa \(200\)](#)

[Campos del mensaje](#)

[Historial de un chat](#)

[Responder en un chat](#)

[Request Body](#)

[Los bots solo llegan por REST, nunca por webhook](#)

[Notificaciones Webhook](#)

Evento: `order.items.sold`

[Solicitud que recibirás](#)

[Payload](#)

[Campos del webhook](#)

[Ventas con Cashea \(`payment.cashea` \)](#)

[Liquidación: de `total` a `payout`](#)

[¿Qué debe hacer tu servidor?](#)

[Ejemplo de receptor \(Node.js\)](#)

Evento: `order.disbursement.completed`

[Solicitud que recibirás](#)

[Payload](#)

[Diferencias respecto a `order.items.sold`](#)

Evento: `chat.message.received`

[Solicitud que recibirás](#)

[Payload](#)

[Campos del webhook](#)

[Autenticación de Webhooks \(HMAC-SHA256\)](#)

[Configuración](#)

[Headers de Webhook](#)

[Verificación de Firma](#)

[Comportamiento](#)

[Códigos de Error](#)

[Errores de autenticación](#)

[Errores de stock sync \(por ítem\)](#)

[Errores de price update \(por ítem\)](#)

[Errores de chat / CRM](#)

[Errores de validación \(HTTP 400\)](#)

[Ejemplos Completos](#)

[Ejemplo 1: Sincronización diaria desde ERP](#)

[Ejemplo 2: Venta en tienda física](#)

[Ejemplo 3: Actualizar precio](#)

[Ejemplo 4: Consultar stock por SKU](#)

[Ejemplo 5: Reconciliar órdenes](#)

[Ejemplo 6: Leer chat \(catch-up\) y responder](#)

[Límites](#)

[Buenas Prácticas](#)

Autenticación

Todas las solicitudes a la API deben incluir tu **API Key** en el header `x-api-key`.

```
x-api-key: mk_tst_aBcDeFgHiJkLmNoPqRsTuVwXyZ123456
```



Tu API Key es secreta. Nunca la compartas en código público, repositorios, ni la expongas en aplicaciones del lado del cliente (frontend). Úsala solo desde tu

servidor (backend).

Formato de la API Key

Componente	Valor	Descripción
Prefijo	<code>mk</code>	Identifica que es una key de Yummy Marketplace
Entorno	<code>tst</code> / <code>prd</code>	Indica si es de pruebas o producción
Secreto	32 caracteres hexadecimales	Parte secreta de la key

Ejemplo completo: `mk_tst_aBcDeFgHiJkLmNoPqRsTuVwXyZ123456`

Permisos

Cada API Key tiene permisos específicos asignados por el equipo de Yummy:

Permiso	Permite
<code>stock:write</code>	Sincronizar / actualizar stock (POST /stock/sync)
<code>stock:read</code>	Consultar stock actual (GET /stock)
<code>price:write</code>	Actualizar precios (PATCH /sku)
<code>orders:read</code>	Consultar órdenes para reconciliación (GET /orders)
<code>chat:read</code>	Leer el chat vía REST (GET /chat/messages, GET /chat/:chatId/messages) y recibir el webhook <code>chat.message.received</code>
<code>chat:write</code>	Responder en el chat (POST /chat/:chatId/messages)

Si tu key no tiene el permiso requerido para un endpoint, recibirás un error `403`.



Opt-in de chat (permisos inmutables): Los permisos se fijan al **crear** la API Key y NO cambian después — el `PATCH` de la key solo edita `name`, `webhookUrl` e `isActive`, nunca `permissions`. Las keys existentes **no** reciben `chat:*` automáticamente. Para habilitar chat en un comercio existente, creá una **nueva** key con `chat:read` / `chat:write` (menor privilegio: una key de stock/precio no gana acceso al chat sin acción explícita).

Permisos granulares: Una key puede tener solo `stock:write + stock:read` sin poder modificar precios, o solo `price:write` sin acceso al stock, o `orders:read` para consulta de órdenes. Podés crear hasta 5 keys con combinaciones diferentes para distintos sistemas.

Conceptos Clave

Identificadores de Variante

Identificador	Uso	Único?
<code>externalVariantId</code>	Identificador principal de variante. Se usa en stock sync, price update, y aparece en responses/webhooks. Si no lo configuraste en tu CSV, se generó automáticamente a partir de tu SKU.	Sí, por producto
<code>sku</code>	Código SKU de la variante. Puede usarse como identificador en stock sync y price update siempre que sea único entre tus productos. Si se repite, debes usar <code>externalVariantId</code> .	No necesariamente
<code>externalId</code>	ID externo del producto (post) en tu sistema. Aparece en responses como referencia cruzada.	No necesariamente

Resolución Inteligente de Variantes

Los endpoints de escritura (POST /stock/sync y PATCH /sku) aceptan `externalVariantId` o **SKU** para identificar la variante (al menos uno). La resolución funciona así:

1. **Si envías `externalVariantId`**, se busca **solo** por ese campo. Si no existe entre tus productos → error `SELLER_API_VARIANT_NOT_FOUND`, aunque también hayas enviado un `sku` válido: no se hace una segunda búsqueda por SKU.
2. **Si envías solo `sku`**, se busca por SKU entre los productos del comercio:
 - **1 resultado** → usa esa variante (caso más común para sellers que no configuraron `externalVariantId`)
 - **0 resultados** → error `SELLER_API_VARIANT_NOT_FOUND` (también si la variante pertenece a otro comercio)
 - **2+ resultados** → error `SELLER_API_VARIANT_AMBIGUOUS_SKU` con mensaje indicando que debe usar `externalVariantId`



¿Cómo saber tu `externalVariantId`? Consultá GET /stock — cada variante en la respuesta incluye su `externalVariantId`. Si no configuraste uno manualmente, será igual a tu SKU.

Endpoints

Sincronizar Stock

POST /seller-api/v1/stock/sync

Permiso requerido: `stock:write`

Envía una o más actualizaciones de stock en un solo request. Cada item se procesa de forma independiente: si uno falla, los demás se procesan normalmente.

Headers

Header	Valor	Requerido
<code>x-api-key</code>	Tu API Key	Sí
<code>Content-Type</code>	<code>application/json</code>	Sí

Request Body

```
{
  "items": [
    {
      "externalVariantId": "CAMISA-AZUL-M",
      "branchExternalId": "SUC-001",
      "stock": 25,
      "operation": "set"
    }
  ]
}
```

También podés identificar por SKU:

```
{
  "items": [
    {
      "sku": "CAMISA-AZUL-M",
      "branchExternalId": "SUC-001",
      "stock": 25,
      "operation": "set"
    }
  ]
}
```

```
]
}
```

Campos del request

Campo	Tipo	Requerido	Descripción
<code>items</code>	Array	Sí	Lista de actualizaciones (mín. 1, máx. 100)
<code>items[].externalVariantId</code>	String	No*	Identificador externo de la variante. Máx. 200 caracteres. Tiene prioridad sobre <code>sku</code> .
<code>items[].sku</code>	String	No*	Código SKU de la variante. Máx. 100 caracteres. Si se repite entre productos, devuelve error <code>SELLER_API_VARIANT_AMBIGUOUS_SKU</code> .



***Nota:** Debés enviar al menos uno de `externalVariantId` o `sku`. Si envías ambos, se usa `externalVariantId`.

Campo	Tipo	Requerido	Descripción
<code>items[].branchExternalId</code>	String	No*	ID externo de la sucursal (el que tu sistema usa). Máx. 100 caracteres.
<code>items[].branchId</code>	String	No*	ID interno de Yummy para la sucursal (ObjectId).
<code>items[].stock</code>	Integer	Sí	Cantidad de unidades. Debe ser ≥ 0 .
<code>items[].operation</code>	String	Sí	Tipo de operación: <code>set</code> , <code>increment</code> , o <code>decrement</code> .



***Nota sobre sucursal:** Podés identificar la sucursal usando `branchExternalId` (tu código) o `branchId` (ID de Yummy). Si no enviás ninguno, se usa la sucursal principal del comercio.

Operaciones disponibles

Operación	Comportamiento	Caso de uso
<code>set</code>	Establece el stock en la cantidad exacta indicada. Sobrescribe el valor anterior.	Sincronización diaria completa desde ERP. "Tengo 25 unidades, punto."

Operación	Comportamiento	Caso de uso
increment	Suma la cantidad al stock actual.	Recepción de mercancía, devolución de cliente. "Llegaron 10 unidades más."
decrement	Resta la cantidad del stock actual. Si no hay suficiente, deja en 0 y retorna warning.	Venta en tienda física u otro canal. "Se vendieron 5 en el POS."

 **Atomicidad:** Todas las operaciones son atómicas en base de datos. Si dos sistemas envían actualizaciones simultáneas, no se "pisan" — cada una ve el stock actualizado del anterior.

Response exitosa (200)

```
{
  "batchId": "batch_a1b2c3d4",
  "status": "completed",
  "results": {
    "total": 3,
    "succeeded": 2,
    "failed": 1,
    "warnings": 1,
    "items": [
      {
        "externalVariantId": "CAMISA-AZUL-M",
        "externalId": "PROD-001",
        "variantId": "665a1b2c3d4e5f...",
        "postId": "664b2c3d4e5f6a...",
        "sku": "CAMISA-AZUL-M",
        "status": "ok",
        "previousStock": 10,
        "newStock": 25
      },
      {
        "externalVariantId": "PANTALON-NEGRO-L",
        "sku": "PANTALON-NEGRO-L",
        "status": "warning",
        "previousStock": 5,
        "newStock": 0,
        "warning": "INSUFFICIENT_STOCK: requested -99999 but only 5 a
        vailable. Stock set to 0."
      }
    ]
  }
}
```

```

    },
    {
      "externalVariantId": "ID-INEXISTENTE",
      "status": "error",
      "errorCode": "SELLER_API_VARIANT_NOT_FOUND",
      "errorMessage": "Variant not found"
    }
  ]
}

```

Estados por ítem

Status	Significado	Campos adicionales
ok	Stock actualizado correctamente.	previousStock, newStock, externalVariantId, externalId, variantId, postId, sku
warning	Se procesó pero con advertencia (ej: stock quedó en 0).	previousStock, newStock, warning
error	No se pudo procesar este ítem.	errorCode, errorMessage

Consultar Stock

GET /seller-api/v1/stock

Permiso requerido: stock:read

Consulta el stock actual de tus productos en Yummy. Solo podés ver el stock de tus propios productos.

Headers

Header	Valor	Requerido
x-api-key	Tu API Key	Sí

Parámetros de consulta (Query Params)

Parámetro	Tipo	Requerido	Descripción
externalVariantId	String	No	Filtrar por identificador externo de variante. Máx. 200 caracteres.

Parámetro	Tipo	Requerido	Descripción
<code>sku</code>	String	No	Filtrar por SKU de variante. Máx. 100 caracteres. Puede devolver múltiples resultados si el SKU se repite entre productos.
<code>externalId</code>	String	No	Filtrar por ID externo de producto. Máx. 200 caracteres.
<code>branchExternalId</code>	String	No	Filtrar por ID externo de sucursal.
<code>branchId</code>	String	No	Filtrar por ID interno de sucursal (ObjectId).
<code>postId</code>	String	No	Filtrar por ID de producto/publicación (ObjectId).
<code>page</code>	Integer	No	Número de página (default: 1).
<code>limit</code>	Integer	No	Cantidad de resultados por página (1-100, default: 20).

Ejemplo de solicitud

```
GET /seller-api/v1/stock?sku=CAMISA-AZUL-M&limit=10
```

Response exitosa (200)

```
{
  "items": [
    {
      "externalVariantId": "CAMISA-AZUL-M",
      "externalId": "PROD-001",
      "sku": "CAMISA-AZUL-M",
      "variantId": "665a1b2c3d4e5f...",
      "postId": "664b2c3d4e5f6a...",
      "price": 25.00,
      "branches": [
        {
          "branchId": "663c3d4e5f6a7b...",
          "branchName": "Sucursal Centro",
          "branchExternalId": "SUC-001",
          "stock": 25
        },
        {

```

```

        "branchId": "663d4e5f6a7b8c...",
        "branchName": "Sucursal Este",
        "branchExternalId": "SUC-002",
        "stock": 12
      }
    ]
  },
  "pagination": {
    "currentPage": 1,
    "totalPages": 1,
    "totalCount": 1,
    "limit": 10
  }
}

```

Actualizar Precios

PATCH /seller-api/v1/sku

Permiso requerido: `price:write`

Actualiza el precio de una o más variantes identificadas por `externalVariantId`. Cada ítem se procesa de forma independiente.

Headers

Header	Valor	Requerido
<code>x-api-key</code>	Tu API Key	Sí
<code>Content-Type</code>	<code>application/json</code>	Sí

Request Body

```

{
  "items": [
    {
      "externalVariantId": "CAMISA-AZUL-M",
      "price": 29.99
    }
  ]
}

```

```
]
}
```

También podés identificar por SKU:

```
{
  "items": [
    {
      "sku": "CAMISA-AZUL-M",
      "price": 29.99
    }
  ]
}
```

Campos del request

Campo	Tipo	Requerido	Descripción
<code>items</code>	Array	Sí	Lista de actualizaciones (mín. 1, máx. 100)
<code>items[].externalVariantId</code>	String	No*	Identificador externo de la variante. Máx. 200 caracteres. Tiene prioridad sobre <code>sku</code> .
<code>items[].sku</code>	String	No*	Código SKU de la variante. Máx. 100 caracteres. Error si SKU ambiguo.
<code>items[].price</code>	Number	Sí	Nuevo precio de la variante en USD. Debe ser ≥ 0 .



***Nota:** Debés enviar al menos uno de `externalVariantId` o `sku`. Si envías ambos, se usa `externalVariantId`.

Response exitosa (200)

```
{
  "batchId": "batch_a1b2c3d4",
  "status": "completed",
  "results": {
    "total": 2,
    "succeeded": 1,
  }
}
```

```

    "failed": 1,
    "items": [
      {
        "externalVariantId": "CAMISA-AZUL-M",
        "externalId": "PROD-001",
        "variantId": "665a1b2c3d4e5f...",
        "postId": "664b2c3d4e5f6a...",
        "sku": "CAMISA-AZUL-M",
        "status": "ok",
        "previousPrice": 25.00,
        "newPrice": 29.99
      },
      {
        "externalVariantId": "ID-INEXISTENTE",
        "status": "error",
        "errorCode": "SELLER_API_VARIANT_NOT_FOUND",
        "errorMessage": "Variant not found"
      }
    ]
  }
}

```

Nota de diseño: El endpoint `/sku` es el punto de extensión para futuras actualizaciones de datos de variante (descripción, atributos, etc.). El precio vive en la variante — es único por variante, independiente de las sucursales.

Consultar Órdenes (Reconciliación)

GET `/seller-api/v1/orders`

Permiso requerido: `orders:read`

Consulta las órdenes aprobadas de tu comercio dentro de un rango de fechas. Retorna el **mismo formato** que el webhook `order.items.sold`, permitiendo reutilizar el mismo parser para ambos flujos.

¿Para qué sirve?

Si tu webhook falla (timeout, servidor caído, error de red), podés consultar este endpoint para obtener las órdenes que no recibiste y reconciliar tu sistema.

Headers

Header	Valor	Requerido
x-api-key	Tu API Key	Sí

Parámetros de consulta (Query Params)

Parámetro	Tipo	Requerido	Descripción
startDate	String (ISO 8601)	Sí	Fecha inicio del rango (ej: 2026-07-01).
endDate	String (ISO 8601)	Sí	Fecha fin del rango (ej: 2026-07-08).
page	Integer	No	Número de página (default: 1).
limit	Integer	No	Cantidad de resultados por página (1-100, default: 20).



Restricción: El rango entre `startDate` y `endDate` no puede exceder **7 días**. Esto protege la base de datos de consultas amplias. Si necesitás más de 7 días, hacé múltiples consultas con rangos consecutivos.

Ejemplo de solicitud

```
GET /seller-api/v1/orders?startDate=2026-07-01&endDate=2026-07-08&page=1&limit=20
```

Response exitosa (200)

```
{
  "orders": [
    {
      "orderId": "667a1b2c-3d4e-5f6a-7b8c-9d0e1f2a3b4c",
      "timestamp": "2026-07-08T15:45:00.000Z",
      "buyer": {
        "buyerId": "usr_88776655",
        "fullName": "Juan Pérez",
        "firstName": "Juan",
        "lastName": "Pérez",
        "email": "juan.perez@email.com",
        "phone": "+584121234567",
      }
    }
  ]
}
```

```

    "documentType": "V",
    "documentNumber": "12345678",
    "documentVerified": true,
    "fiscalAddress": null
  },
  "shippingAddress": {
    "recipientName": "Juan Pérez",
    "street": "Av. Francisco de Miranda, Edif. Galipán, Piso 4",
    "city": "Caracas",
    "state": "Distrito Capital",
    "zipCode": ""
  },
  "shipping": {
    "couponUrl": "https://example.com/coupon/abc123",
    "guideNumber": "123456789",
    "shippingMethod": "standard"
  },
  "payment": {
    "paymentMethod": "pago-movil",
    "usedCashea": false,
    "paymentIntent": {
      "bank": "0102",
      "phone": "+584121234567",
      "transactionPhone": "+584121234567",
      "dni": "V12345678",
      "documentType": "V",
      "documentNumber": "12345678"
    }
  },
  "disbursement": {
    "status": "paid",
    "reference": "20260722150502",
    "currency": "VES",
    "amount": 2275.0,
    "transactionId": "9fdfb05d-06b3-53a9-bf96-d3fcc0e0fbc2",
    "paidAt": "2026-07-22T21:05:02.070Z"
  },
  "financials": {
    "baseCurrency": "USD",
    "exchangeRate": 45.5,

```

```

    "subtotal": 50.0,
    "subtotalBs": 2275.0,
    "shippingCost": 5.0,
    "shippingCostBs": 227.5,
    "freeShipping": false,
    "tax": 0,
    "discount": 0,
    "total": 55.0,
    "totalBs": 2502.5,
    "deductions": [
      { "type": "shipping", "amount": 5.0, "amountBs": 227.5 }
    ],
    "payout": 50.0,
    "payoutBs": 2275.0
  },
  "items": [
    {
      "externalVariantId": "ZAPATO-NEGRO-42",
      "externalId": "PROD-001",
      "sku": "ZAPATO-NEGRO-42",
      "variantId": "665a1b2c3d4e5f...",
      "branchId": "663c3d4e5f6a7b...",
      "branchExternalId": "SUC-001",
      "quantity": 2,
      "unitPrice": 25.0,
      "totalPrice": 50.0,
      "totalPriceBs": 2275.0,
      "stockAfter": 12
    }
  ]
},
{
  "pagination": {
    "currentPage": 1,
    "totalPages": 3,
    "totalCount": 52,
    "limit": 20
  }
}
}

```



Campo `disbursement` : Presente en `GET /orders` y en el webhook

`order.disbursement.completed` . **No** viene en el webhook `order.items.sold` , ya que al momento de la venta aún no se ha procesado el desembolso. Contiene la referencia bancaria del pago móvil enviado al comercio. El campo `status` puede ser `pending` , `paid` , `failed` , `processing` o `cancelled` . La `reference` es la referencia bancaria del pago móvil — úsala para conciliar con tu estado de cuenta.

Formato compatible con el webhook: Cada objeto en el array `orders` tiene la misma estructura base que el payload del webhook `order.items.sold` , más el campo adicional `disbursement` con la información del desembolso al comercio. Podés reutilizar el mismo parser/handler para ambos flujos. Las órdenes de este endpoint no llevan el campo `event` (solo lo incluyen los webhooks).

Chat / CRM

Canal híbrido **webhook + REST** para consumir y responder desde tu propio CRM el chat comprador ↔ comercio, sin depender del portal ni del app. Modelo de referencia (Stripe/GitHub/Shopify): el webhook `chat.message.received` es la señal realtime **best-effort**; los endpoints REST con cursor son la **red de seguridad** — si el webhook falla, recuperarás el gap con `?since={cursor}` .



Cursor = `messageId` (ObjectId de Mongo), NO timestamp. Es total-ordered y monótono: nunca perdés ni duplicás mensajes con el mismo timestamp (el ObjectId a segundos lleva un contador incremental por proceso que mantiene el orden estable, a diferencia de `creationDate`). El `timestamp` es informativo; el orden canónico es por `messageId` . El CRM distingue dirección con el campo `direction` (`inbound` = del comprador, `outbound` = del comercio).

Listar mensajes

```
GET /seller-api/v1/chat/messages
```

Permiso requerido: `chat:read`

Lista todos los mensajes del comercio (todos sus chats) en orden de cursor. Para catch-up global tras una caída del webhook.

Query Params

Parámetro	Tipo	Requerido	Descripción
<code>since</code>	String (hex 24)	No	Cursor keyset: devuelve mensajes con <code>messageId > since</code> . Omitir para empezar desde el inicio.
<code>limit</code>	Integer	No	Mensajes por página (1-100, default 50).

Response exitosa (200)

```
{
  "messages": [
    {
      "id": "665a1b2c3d4e5f6a7b8c9d0e",
      "chatId": "664b2c3d4e5f6a7b8c9d0e1f",
      "content": "Hola, sigue disponible el zapato talla 42?",
      "files": [],
      "from": "usr_88776655",
      "to": "cmr_11223344",
      "direction": "inbound",
      "isBot": false,
      "timestamp": "2026-09-01T15:45:00.000Z",
      "cursor": "665a1b2c3d4e5f6a7b8c9d0e"
    }
  ],
  "nextCursor": null
}
```

`nextCursor` es el `messageId` del último mensaje **solo cuando la página viene llena** (`messages.length === limit`, puede haber más); es `null` cuando el backlog se drenó — como en este ejemplo, 1 mensaje con `limit` 50. Paginá repitiendo con `?since={nextCursor}` mientras `nextCursor` no sea `null`.

Campos del mensaje

Campo	Tipo	Descripción
<code>id</code>	String	ID del mensaje (= <code>cursor</code>).
<code>chatId</code>	String	Chat al que pertenece.
<code>content</code>	String	Texto en plano (Yummy lo descifra en memoria). <code>""</code> si el mensaje era solo-archivos o histórico sin texto.

Campo	Tipo	Descripción
<code>files</code>	Array	Adjuntos <code>[{ "uri", "type" }]</code> . <code>[]</code> si no hay.
<code>from</code> / <code>to</code>	String	Autor y destinatario del mensaje.
<code>direction</code>	String	<code>inbound</code> (del comprador) u <code>outbound</code> (del comercio).
<code>isBot</code>	Boolean	<code>true</code> si es mensaje de bot (envío MRW, tracking QR, recordatorios). Ver nota de bots.
<code>timestamp</code>	String (ISO 8601)	Fecha de creación (informativa).
<code>cursor</code>	String	Cursor keyset de este mensaje (= <code>id</code>).

Historial de un chat

```
GET /seller-api/v1/chat/:chatId/messages
```

Permiso requerido: `chat:read`

Igual que el anterior pero acotado a **un solo chat**. Mismo shape de response y misma paginación por cursor. Path param `chatId` (ObjectId, requerido); query `since` y `limit` idénticos.



Si el `chatId` no existe o no pertenece a tu comercio → `404 SELLER_API_CHAT_NOT_FOUND`.

Responder en un chat

```
POST /seller-api/v1/chat/:chatId/messages
```

Permiso requerido: `chat:write`

Responde en un chat. Reusa el mismo flujo interno que el portal/app, así que el comprador recibe push y el dashboard del comercio ve el mensaje en vivo, gratis. Path param `chatId` (ObjectId, requerido).

Request Body

Campo	Tipo	Requerido	Descripción
<code>content</code>	String	No*	Texto de la respuesta. Max 5000 chars.
<code>files</code>	Array	No*	1 a 10 adjuntos <code>{ "uri", "type" }</code> (<code>uri</code> debe ser URL válida).

Campo	Tipo	Requerido	Descripción
<code>replyTo</code>	String (ObjectId)	No	ID del mensaje al que se responde.



***Debés enviar al menos uno de `content` o `files` .**

```
{
  "content": "Si! Disponible. Te lo reservo?",
  "replyTo": "665a1b2c3d4e5f6a7b8c9d0e"
}
```

Response (200): el mensaje creado, en el mismo shape de mensaje de arriba (`direction: "outbound"`).



Atribución: la respuesta queda atribuida al **comercio root** (la API Key ES el actor, no un agente/persona). No hay scope por sucursal: la key ve y escribe todos los chats del comercio.

Códigos de error del POST: `404 SELLER_API_CHAT_NOT_FOUND` (chat inexistente o ajeno); `403 CHAT_INACTIVE` (chat inactivo/bloqueado, ej. pago expirado); `403 CHAT_RECIPIENT_BLOCKED` / `CHAT_RECIPIENT_BLOCKED_YOU` / `CHAT_UNBLOCK_TO_SEND` (relación de bloqueo).

Los bots solo llegan por REST, nunca por webhook



Los mensajes de **bot** (estado de envío MRW, tracking QR, recordatorios de pago) son señal operacional del app, **no** un evento para el CRM. Por diseño **NO se emiten por el webhook** `chat.message.received` (su payload siempre lleva `isBot: false`). Si tu CRM quiere verlos, los obtiene por `GET /chat/messages` , donde llegan marcados `isBot: true` .

Notificaciones Webhook

Si tu comercio tiene configurada una **URL de webhook**, Yummy enviará automáticamente una notificación HTTP a tu servidor cada vez que se venda un producto tuyo en la plataforma.

Evento: `order.items.sold`

Se dispara cuando un comprador completa el pago de una orden que incluye productos de tu comercio.

Solicitud que recibirás

```
POST {tu_webhook_url}
Content-Type: application/json
X-Webhook-Id: {orderId}
X-Webhook-Timestamp: {ISO 8601}
X-Webhook-Signature: {HMAC-SHA256 signature}
```

Payload

```
{
  "event": "order.items.sold",
  "orderId": "667a1b2c3d4e5f6a7b8c9d0e",
  "timestamp": "2026-06-17T15:45:00.000Z",
  "buyer": {
    "buyerId": "usr_88776655",
    "fullName": "Juan Pérez",
    "firstName": "Juan",
    "lastName": "Pérez",
    "email": "juan.perez@email.com",
    "phone": "+584121234567",
    "documentType": "V",
    "documentNumber": "12345678",
    "documentVerified": true,
    "fiscalAddress": null
  },
  "shippingAddress": {
    "recipientName": "Juan Pérez",
    "street": "Av. Francisco de Miranda, Edif. Galipán, Piso 4",
    "city": "Caracas",
    "state": "Distrito Capital",
    "zipCode": ""
  },
  "shipping": {
    "couponUrl": "https://example.com/coupon/abc123",
    "guideNumber": "123456789",
  }
}
```

```

    "shippingMethod": "standard"
  },
  "payment": {
    "paymentMethod": "pago-movil",
    "usedCashea": false,
    "paymentIntent": {
      "bank": "0102",
      "phone": "+584121234567",
      "transactionPhone": "+584121234567",
      "dni": "V12345678",
      "documentType": "V",
      "documentNumber": "12345678"
    }
  },
  "financials": {
    "baseCurrency": "USD",
    "exchangeRate": 45.50,
    "subtotal": 50.00,
    "subtotalBs": 2275.00,
    "shippingCost": 5.00,
    "shippingCostBs": 227.50,
    "freeShipping": false,
    "tax": 0,
    "discount": 0,
    "total": 55.00,
    "totalBs": 2502.50,
    "deductions": [
      { "type": "shipping", "amount": 5.00, "amountBs": 227.50 }
    ],
    "payout": 50.00,
    "payoutBs": 2275.00
  },
  "items": [
    {
      "externalVariantId": "ZAPATO-NEGRO-42",
      "externalId": "PROD-001",
      "sku": "ZAPATO-NEGRO-42",
      "variantId": "665a1b2c3d4e5f...",
      "branchId": "663c3d4e5f6a7b...",
      "branchExternalId": "SUC-001",

```

```

    "quantity": 2,
    "unitPrice": 25.00,
    "totalPrice": 50.00,
    "totalPriceBs": 2275.00,
    "stockAfter": 12
  }
]
}

```

Campos del webhook

Campo	Tipo	Descripción
<code>event</code>	String	Siempre <code>"order.items.sold"</code> para este evento.
<code>orderId</code>	String	ID único de la orden en Yummy.
<code>timestamp</code>	String (ISO 8601)	Fecha y hora de creación de la orden. Es el mismo valor en <code>order.items.sold</code> , <code>order.disbursement.completed</code> y <code>GET /orders</code> : no cambia según cuándo se envía el evento o se consulta.
<code>buyer</code>	Object	Datos del comprador: <code>buyerId</code> , <code>fullName</code> , <code>email</code> , <code>phone</code> , <code>firstName</code> , <code>lastName</code> , <code>documentType</code> (ej. "V"), <code>documentNumber</code> (cédula), <code>documentVerified</code> (booleano), <code>fiscalAddress</code> (objeto o <code>null</code>).
<code>shippingAddress</code>	Object (opcional)	Dirección de envío: <code>recipientName</code> , <code>street</code> , <code>city</code> , <code>state</code> , <code>zipCode</code> . Solo presente si la orden requiere envío.
<code>shipping</code>	Object (opcional)	Datos de envío: <code>couponUrl</code> (URL del cupón de envío), <code>guideNumber</code> (número de guía), <code>shippingMethod</code> (tipo de servicio). Solo presente cuando hay datos de shipment.
<code>payment</code>	Object	<code>paymentMethod</code> (ej: "pago-movil"), <code>usedCashea</code> (booleano), <code>paymentIntent</code> (datos del pago del comprador: <code>bank</code> , <code>phone</code> , <code>transactionPhone</code> , <code>dni</code> , <code>documentType</code> , <code>documentNumber</code> — opcional), <code>cashea</code> (objeto, solo cuando <code>usedCashea</code> es <code>true</code> — ver "Ventas con Cashea").
<code>disbursement</code>	Object (opcional)	Presente en <code>GET /orders</code> y en el webhook <code>order.disbursement.completed</code> . Información del desembolso al comercio: <code>status</code> (pending/paid/failed/processing/cancelled),

Campo	Tipo	Descripción
		<code>reference</code> (referencia bancaria del pago móvil), <code>currency</code> (siempre <code>"VES"</code>), <code>amount</code> (monto en VES enviado al seller; igual a <code>financials.payoutBs</code>), <code>transactionId</code> (ID interno de la transacción), <code>paidAt</code> (fecha ISO 8601 de la última actualización del desembolso; cuando <code>status</code> es <code>paid</code> , es la fecha del pago). No viene en el webhook <code>order.items.sold</code> porque el desembolso se procesa después de la venta (~15 min).
<code>financials</code>	Object	Detalles financieros: <code>baseCurrency</code> ("USD"), <code>exchangeRate</code> , <code>subtotal</code> , <code>subtotalBs</code> , <code>shippingCost</code> (costo real cobrado por el courier — siempre el valor real, incluso con envío gratis activo, nunca se fuerza a 0 internamente), <code>shippingCostBs</code> , <code>freeShipping</code> (booleano — indica si el envío fue gratis para el comprador; cuando es <code>true</code> , <code>total</code> NO incluye <code>shippingCost</code>), <code>casheaBuyerFee</code> / <code>casheaBuyerFeeBs</code> (solo en ventas Cashea que lo cobraron: fee de Cashea que pagó el comprador, IVA incluido), <code>tax</code> , <code>discount</code> , <code>total</code> / <code>totalBs</code> (lo que se le cobró al comprador: <code>subtotal</code> • <code>shippingCost</code> si no es envío gratis + <code>casheaBuyerFee</code>), <code>deductions</code> , <code>payout</code> / <code>payoutBs</code> (tu liquidación — ver "Liquidación: de <code>total</code> a <code>payout</code> ").
<code>items</code>	Array	Productos vendidos que pertenecen a tu comercio.
<code>items[].externalVariantId</code>	String	Identificador externo de la variante.
<code>items[].externalId</code>	String (opcional)	ID externo del producto en tu sistema.
<code>items[].sku</code>	String	SKU de la variante vendida.
<code>items[].variantId</code>	String	ID interno de la variante en Yummy.
<code>items[].branchId</code>	String	ID interno de la sucursal donde se descontó stock.
<code>items[].branchExternalId</code>	String (opcional)	Tu ID externo de la sucursal (si fue configurado).
<code>items[].quantity</code>	Integer	Cantidad de unidades vendidas.
<code>items[].unitPrice</code>	Number	Precio unitario de la variante en USD al momento de la venta.
<code>items[].totalPrice</code>	Number	<code>unitPrice × quantity</code> en USD.
<code>items[].totalPriceBs</code>	Number	<code>totalPrice</code> en bolívares, a la tasa <code>financials.exchangeRate</code> .

Campo	Tipo	Descripción
<code>items[].stockAfter</code>	Integer	Stock restante en Yummy después de la venta.



Envío gratis (`freeShipping`): `shippingCost` siempre refleja el costo real cobrado por el courier — nunca se fuerza a 0 internamente — para que el comercio conozca cuánto está asumiendo por la promoción. Cuando `freeShipping` es `true`, ese costo NO se le cobra al comprador, por lo que `total` = `subtotal` (excluye `shippingCost`). Cuando es `false`, `total` = `subtotal + shippingCost`.



Datos del comprador (`buyer.documentNumber`, `buyer.fiscalAddress`): Se agregaron para que el comercio pueda facturar y verificar identidad al entregar, especialmente en pedidos con retiro en tienda.

- `documentNumber` (cédula) prioriza la cédula **verificada** por Yummy (`documentVerified: true`). Si no está verificada, es una cédula auto-declarada por el comprador o, como último recurso, la del **titular de la cuenta bancaria** que usa para cobrar reembolsos/ganancias — **puede no coincidir con la persona que retira el pedido**. Cuando `documentVerified` es `false`, validá la identidad contra el documento físico antes de entregar.
- `fiscalAddress` es **best-effort y puede llegar null** — Yummy aún no captura la dirección fiscal en el checkout para todos los compradores.

Ventas con Cashea (`payment.cashea`)

Cuando la venta se pagó con Cashea (`payment.usedCashea: true`), `payment` incluye un bloque `cashea` con el detalle de la financiación. El mismo bloque llega en `order.items.sold`, `order.disbursement.completed` y `GET /orders`. En ventas sin Cashea el bloque no viene.

`financials.subtotal` sigue siendo el **precio completo** del producto. El bloque `cashea` lo separa en dos partes: la **inicial** que pagó el comprador (lo que Yummy te deposita junto con la venta) y el monto que **Cashea financia en cuotas**.

```
"payment": {
  "paymentMethod": "pago-movil",
  "usedCashea": true,
  "paymentIntent": { "bank": "0105", "phone": "+584141234567", "trans
actionPhone": "+584141234567", "dni": "V12345678", "documentType":
"V", "documentNumber": "12345678" },
```

```

"cashea": {
  "casheaOrderId": 221028073,
  "status": "confirmed",
  "initialAmount": 5.4,
  "initialAmountBs": 4614.1,
  "financedAmount": 21.59,
  "financedAmountBs": 18447.87,
  "sellerFee": 0,
  "sellerFeeRate": 0,
  "buyerFee": { "rate": 0.021, "amount": 0.57, "ivaRate": 0.16, "iva": 0.09, "total": 0.66 },
  "installments": [
    { "installmentNumber": 1, "amount": 7.2, "dueDate": "2026-10-08T15:40:46.696Z" },
    { "installmentNumber": 2, "amount": 7.2, "dueDate": "2026-10-22T15:40:46.696Z" },
    { "installmentNumber": 3, "amount": 7.2, "dueDate": "2026-11-05T15:40:46.696Z" }
  ]
}

```

Ejemplo con una venta de \$26.99 a tasa 854.4637: inicial \$5.40, financiado \$21.59 en 3 cuotas, acuerdo con Cashea al 0% y fee del comprador de 2.1% + IVA.

Campo	Tipo	Descripción
<code>casheaOrderId</code>	Number (opcional)	ID de la orden en Cashea, para conciliar con ellos. Puede faltar en ventas antiguas.
<code>status</code>	String (opcional)	Estado de la orden en Cashea: <code>pending</code> , <code>confirmed</code> , <code>cancelled</code> o <code>failed</code> .
<code>initialAmount</code> / <code>initialAmountBs</code>	Number	Inicial pagada por el comprador, en USD y en bolívars. Es lo único que Yummy te deposita junto con la venta.
<code>financedAmount</code> / <code>financedAmountBs</code>	Number	<code>subtotal - initialAmount</code> , en USD y en bolívars. Es lo que Cashea financia en cuotas.
<code>sellerFee</code> / <code>sellerFeeRate</code>	Number (opcional)	Comisión de Cashea descontada de tu desembolso: monto en USD y tasa como fracción (<code>0.045</code> = 4.5%). <code>0</code> indica un acuerdo al 0%.
<code>buyerFee</code>	Object (opcional)	Fee de Cashea que pagó el comprador encima del precio, en USD: <code>rate × subtotal = amount</code> , más <code>iva</code> (<code>ivaRate</code> sobre <code>amount</code>); <code>total = amount + iva</code> . Se cobra junto con

Campo	Tipo	Descripción
		la inicial y no es tuyo : se descuenta de tu desembolso. No viene si no se cobró. El IVA de este fee no va en <code>financials.tax</code> , que corresponde al producto.
<code>installments</code>	Array (opcional)	Cuotas financiadas tal como las agendó Cashea, sin la inicial. Cada una trae <code>installmentNumber</code> (1..n), <code>amount</code> (USD, redondeado a centavos) y <code>dueDate</code> (fecha de corte, ISO 8601). Por el redondeo, la suma de las cuotas puede diferir en un centavo de <code>financedAmount</code> .



Cuotas: el bloque muestra el cronograma de cuotas, no su estado de cobro. Las cuotas las gestionás directamente con Cashea.

Ventas antiguas: una venta con Cashea sin inicial registrada trae `usedCashea: true` sin bloque `cashea`, y sin `deductions` / `payout` / `payoutBs` en `financials`.

Liquidación: de `total` a `payout`

`financials` cierra con tu liquidación. Viene en `order.items.sold`, `order.disbursement.completed` y `GET /orders`, así que desde el primer evento sabés cuánto vas a recibir.

- `total`: lo que se le cobró al comprador.
- `deductions`: lo que **no** llega a tu comercio. Cada línea trae `type`, `amount` (USD), `amountBs` y, en las porcentuales, `rate` y `base`. Solo vienen líneas mayores a 0.
- `payout` / `payoutBs`: lo que Yummy te deposita. `payoutBs` es igual a `disbursement.amount`.

$$\text{payout} = \text{total} - \sum \text{deductions.amount}$$

<code>type</code>	Qué es
<code>cashea_financed</code>	Parte del precio que financia Cashea. No es un cobro : la cobrás directamente con Cashea en cuotas y nunca pasa por Yummy.
<code>cashea_seller_fee</code>	Comisión por vender con Cashea: <code>rate</code> × <code>base</code> (el precio completo del producto), descontada de la inicial.
<code>cashea_buyer_fee</code>	Fee de Cashea que pagó el comprador (<code>payment.cashea.buyerFee.total</code>). Está en <code>total</code> pero no es tuyo.
<code>shipping</code>	Costo del courier. Siempre se descuenta: si lo pagó el comprador, está en <code>total</code> y va al courier; con envío gratis no está en <code>total</code> y lo asume tu comercio.

El envío **no** entra en la inicial ni en lo financiado: el comprador lo paga completo junto con la inicial.

Ejemplo: venta Cashea de \$20 a tasa 577.5461, inicial \$6.67, comisión de 3.5% al comercio, fee de 2.1% + IVA al comprador, sin envío.

```
"financials": {
  "baseCurrency": "USD",
  "exchangeRate": 577.5461,
  "subtotal": 20,
  "subtotalBs": 11550.92,
  "shippingCost": 0,
  "shippingCostBs": 0,
  "freeShipping": false,
  "casheaBuyerFee": 0.49,
  "casheaBuyerFeeBs": 283,
  "tax": 0,
  "discount": 0,
  "total": 20.49,
  "totalBs": 11833.92,
  "deductions": [
    { "type": "cashea_financed", "amount": 13.33, "amountBs": 7698.69 },
    { "type": "cashea_seller_fee", "rate": 0.035, "base": 20, "amount": 0.7, "amountBs": 404.28 },
    { "type": "cashea_buyer_fee", "amount": 0.49, "amountBs": 283 }
  ],
  "payout": 5.97,
  "payoutBs": 3447.95
}
```

$20.49 - 13.33 - 0.70 - 0.49 = 5.97$, y `payoutBs` (3447.95 Bs) coincide con `disbursement.amount`.



Redondeo: cada línea en bolívares se redondea por separado, así que la suma en Bs puede diferir en un centavo. En ese caso, el valor que manda es `payoutBs`, que es el monto que efectivamente se depositó.

¿Qué debe hacer tu servidor?

Tu endpoint debe:

1. Verificar la firma HMAC-SHA256 (ver sección de autenticación de webhooks)
2. Responder con un **HTTP 200** lo más rápido posible

3. Procesar la información de forma asíncrona (descontar en tu ERP, actualizar tu POS, etc.)



Importante: Yummy espera respuesta por máximo **5 segundos**. Si tu servidor tarda más o no responde, el webhook se registra como fallido pero **la venta no se cancela**. Tu sistema puede consultar las órdenes perdidas vía GET /seller-api/v1/orders.

Ejemplo de receptor (Node.js)

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

app.post('/webhook/yummy', express.json(), (req, res) => {
  // 1. Verificar firma
  const signature = req.headers['x-webhook-signature'];
  const expected = crypto
    .createHmac('sha256', WEBHOOK_SECRET)
    .update(JSON.stringify(req.body))
    .digest('hex');

  if (!crypto.timingSafeEqual(Buffer.from(signature), Buffer.from(expected))) {
    return res.status(401).send('Invalid signature');
  }

  // 2. Responder inmediatamente
  res.status(200).json({ received: true });

  // 3. Procesar en background
  const { event, orderId, items, payment, financials } = req.body;

  if (event === 'order.items.sold') {
    items.forEach(item => {
      console.log(
        `Venta: ${item.quantity}x ${item.sku} @ ${item.unitPrice} -  

        Stock restante: ${item.stockAfter}`
      );
    });
  }
});
```

```

    });
    console.log(`Total: ${financials.total} (Bs ${financials.totalBs})`);
  }

  if (event === 'order.disbursement.completed') {
    const { reference, amount, paidAt } = req.body.disbursement;
    console.log(
      `Desembolso de la orden ${orderId} pagado: ref ${reference}, Bs ${amount} (${paidAt})`
    );
  }
});

app.listen(3000);

```

Evento: `order.disbursement.completed`

Se dispara **~15 minutos después de la venta**, cuando Yummy confirma que el **pago móvil del desembolso a tu comercio fue exitoso** (el banco reporta `status: "paid"` para la transacción del seller). Es un evento **informativo para conciliación contable**: te entrega la referencia bancaria del pago que recibiste.



Separación de eventos: `order.items.sold` se dispara al momento del pago (descontó stock, preparás el envío). `order.disbursement.completed` se dispara después, cuando tu dinero fue depositado (conciliás la referencia bancaria). Ambos eventos usan la misma autenticación HMAC, los mismos headers y el mismo formato de payload.

Solicitud que recibirás

```

POST {tu_webhook_url}
Content-Type: application/json
X-Webhook-Id: {orderId}
X-Webhook-Timestamp: {ISO 8601}
X-Webhook-Signature: {HMAC-SHA256 signature}

```

Payload

Es el mismo payload completo de la orden que `order.items.sold` (`buyer`, `shippingAddress`, `shipping`, `payment`, `financials`, `items`) con dos diferencias: el campo `event` es

"order.disbursement.completed" y el campo `disbursement` siempre está presente con `status: "paid"`. Así reutilizás el mismo parser y tenés toda la información de la orden sin un GET adicional.

```
{
  "event": "order.disbursement.completed",
  "orderId": "667a1b2c-3d4e-5f6a-7b8c-9d0e1f2a3b4c",
  "timestamp": "2026-07-22T21:05:02.070Z",
  "buyer": {
    "buyerId": "usr_88776655",
    "fullName": "Juan Pérez",
    "firstName": "Juan",
    "lastName": "Pérez",
    "email": "juan.perez@email.com",
    "phone": "+584121234567",
    "documentType": "V",
    "documentNumber": "12345678",
    "documentVerified": true,
    "fiscalAddress": null
  },
  "shippingAddress": {
    "recipientName": "Juan Pérez",
    "street": "Av. Francisco de Miranda, Edif. Galipán, Piso 4",
    "city": "Caracas",
    "state": "Distrito Capital",
    "zipCode": ""
  },
  "payment": {
    "paymentMethod": "pago-movil",
    "usedCashea": false
  },
  "disbursement": {
    "status": "paid",
    "reference": "20260722150502",
    "currency": "VES",
    "amount": 2275.0,
    "transactionId": "9fdfb05d-06b3-53a9-bf96-d3fcc0e0fbc2",
    "paidAt": "2026-07-22T21:05:02.070Z"
  },
  "financials": {
```

```

    "baseCurrency": "USD",
    "exchangeRate": 45.5,
    "subtotal": 50.0,
    "subtotalBs": 2275.0,
    "shippingCost": 5.0,
    "shippingCostBs": 227.5,
    "freeShipping": false,
    "tax": 0,
    "discount": 0,
    "total": 55.0,
    "totalBs": 2502.5,
    "deductions": [
      { "type": "shipping", "amount": 5.0, "amountBs": 227.5 }
    ],
    "payout": 50.0,
    "payoutBs": 2275.0
  },
  "items": [
    {
      "externalVariantId": "ZAPATO-NEGRO-42",
      "externalId": "PROD-001",
      "sku": "ZAPATO-NEGRO-42",
      "variantId": "665a1b2c3d4e5f...",
      "branchId": "663c3d4e5f6a7b...",
      "branchExternalId": "SUC-001",
      "quantity": 2,
      "unitPrice": 25.0,
      "totalPrice": 50.0,
      "totalPriceBs": 2275.0,
      "stockAfter": 12
    }
  ]
}

```

Diferencias respecto a `order.items.sold`

- `event`: vale `"order.disbursement.completed"` en lugar de `"order.items.sold"`.
- `disbursement`: **siempre presente**. `status` = `paid`; `reference` = referencia bancaria del pago móvil (úsala para conciliar con tu estado de cuenta); `currency` = `"VES"`; `amount` = monto en VES depositado a tu comercio (igual a `financials.payoutBs`); `transactionId` = ID interno de la

transacción; `paidAt` = fecha ISO 8601 del pago (última actualización del desembolso, que en este evento es la confirmación `paid`).

- El resto (`buyer` , `shippingAddress` , `shipping` , `payment` , `financials` , `items`) es idéntico al webhook de venta.



Aplican las mismas reglas de entrega que `order.items.sold` : timeout de 5 segundos, sin retries (fire-and-forget), el fallo del webhook **no** afecta el desembolso, y se envía a todas las API keys con `webhookUrl` configurado. Verificá la firma HMAC-SHA256 igual que con el evento de venta.

Evento: `chat.message.received`

Se dispara cuando un **comprador** envía un mensaje a un comercio que tiene una API Key activa con `chat:read` + `webhookUrl` . **Best-effort** — si falla, el gap se recupera por `GET /chat/messages?since={cursor}` (por eso no hay retries pesados).



Requisitos para recibirlo: key activa con permiso `chat:read` y `webhookUrl` configurada. **Anti-eco:** el webhook va solo hacia el **destinatario** cuando es un comercio del canal; la respuesta de tu propio CRM tiene como destinatario al comprador, así que nunca se te reenvía tu propio mensaje.

Solicitud que recibirás

```
POST {tu_webhook_url}
Content-Type: application/json
X-Webhook-Id: {messageId}
X-Webhook-Timestamp: {ISO 8601}
X-Webhook-Signature: {HMAC-SHA256 signature}
```

Headers idénticos a los eventos de orden, salvo que `X-Webhook-Id` = `messageId` (en vez de `orderId`), para idempotencia del consumidor.

Payload

```
{
  "event": "chat.message.received",
  "messageId": "665a1b2c3d4e5f6a7b8c9d0e",
  "chatId": "664b2c3d4e5f6a7b8c9d0e1f",
```

```

"cursor": "665a1b2c3d4e5f6a7b8c9d0e",
"content": "Hola, sigue disponible el zapato talla 42?",
"files": [],
"timestamp": "2026-09-01T15:45:00.000Z",
"from": { "buyerId": "usr_88776655" },
"isBot": false
}

```

Campos del webhook

Campo	Tipo	Descripción
<code>event</code>	String	Siempre <code>"chat.message.received"</code> .
<code>messageId</code>	String	ID del mensaje (= <code>X-Webhook-Id</code>).
<code>chatId</code>	String	Chat al que pertenece.
<code>cursor</code>	String	Igual a <code>messageId</code> . Reanudá el cursor REST desde aquí: <code>GET /chat/messages?since={cursor}</code> .
<code>content</code>	String	Texto en plano (Yummy descifra en memoria; nunca viaja cifrado ni se loguea). <code>""</code> si es solo-archivos.
<code>files</code>	Array	Adjuntos <code>[{ "uri", "type" }]</code> .
<code>timestamp</code>	String (ISO 8601)	Fecha del mensaje (informativa; el orden canónico es por <code>messageId</code>).
<code>from.buyerId</code>	String	Autor del mensaje (el comprador).
<code>isBot</code>	Boolean	Siempre <code>false</code> en el webhook — los bots no se emiten por aquí (ver sección Chat / CRM).



Catch-up: guardá el `cursor` del último webhook procesado. Si tu servidor estuvo caído, `GET /seller-api/v1/chat/messages?since={cursor}` te devuelve exactamente los mensajes que faltaron, sin duplicados (`$gt` estricto sobre `messageId`).

Aplican las **mismas reglas** que los eventos de orden: firma HMAC-SHA256 sobre el raw body, timeout 5s, sin retries (fire-and-forget), el fallo no afecta el mensaje, y se envía a todas las keys del comercio con `chat:read` + `webhookUrl`.

Autenticación de Webhooks (HMAC-SHA256)

Configuración

Al crear un API key con `webhookUrl`, Yummy genera automáticamente un `webhookSecret` (64 caracteres hex). Este secret se muestra **una sola vez** en la respuesta de creación del API key.

Si se actualiza la `webhookUrl` vía PATCH, se genera un nuevo `webhookSecret` y se retorna en la respuesta.

Headers de Webhook

Cada webhook enviado por Yummy incluye los siguientes headers:

Header	Descripción
<code>Content-Type</code>	<code>application/json</code>
<code>X-Webhook-Id</code>	ID único del evento: <code>orderId</code> (eventos de orden) o <code>messageId</code> (evento de chat). Para idempotencia del consumidor.
<code>X-Webhook-Timestamp</code>	Timestamp ISO 8601 del momento de envío
<code>X-Webhook-Signature</code>	Firma HMAC-SHA256 del body usando el <code>webhookSecret</code>

Verificación de Firma

```
const crypto = require("crypto");

function verifyWebhookSignature(rawBody, signature, secret) {
  const expected = crypto
    .createHmac("sha256", secret)
    .update(rawBody)
    .digest("hex");
  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expected)
  );
}
```

Comportamiento

- **Timeout:** 5 segundos — si el servidor no responde, se loguea como fallo
- **Retries:** No hay retries automáticos en la versión actual (fire-and-forget)
- **Fallo NO cancela la venta:** El pago se completa normalmente independientemente del resultado del webhook

- **Múltiples keys:** Si el comercio tiene varias API keys con webhookUrl configurado, se envía a todas

Códigos de Error

Errores de autenticación

HTTP	Código	Descripción	Solución
401	<code>SELLER_API_KEY_MISSING</code>	No se envió el header <code>x-api-key</code> .	Incluí el header <code>x-api-key</code> con tu API Key.
401	<code>SELLER_API_KEY_INVALID</code>	La API Key no es válida o fue revocada.	Verificá que la key es correcta. Si fue rotada, usá la nueva.
403	<code>SELLER_API_KEY_PERMISSION_DENIED</code>	La key no tiene el permiso requerido para este endpoint.	Contactá al equipo de Yummy para que actualice los permisos.

Errores de stock sync (por item)

Estos errores aparecen dentro de `results.items[]` con `status: "error"`:

Código	Descripción	Solución
<code>SELLER_API_VARIANT_NOT_FOUND</code>	No existe una variante con ese <code>externalVariantId</code> o <code>sku</code> en tu catálogo de Yummy. También se devuelve si la variante pertenece a otro comercio.	Verificá que el identificador coincida exactamente con el registrado en Yummy.
<code>SELLER_API_BRANCH_NOT_FOUND</code>	No se encontró la sucursal indicada.	Verificá que el <code>branchExternalId</code> o <code>branchId</code> sea correcto.
<code>SELLER_API_NO_STOCK_IN_BRANCH</code>	Solo con <code>decrement</code> : la variante no tiene registro de stock en esa sucursal. Con <code>set</code> o <code>increment</code> el registro se crea.	Usá la operación <code>set</code> para inicializar el stock en esa sucursal.
<code>SELLER_API_PROCESSING_ERROR</code>	Error interno al procesar el item.	Reintentá el request. Si persiste, contactá soporte.
<code>SELLER_API_VARIANT_AMBIGUOUS_SKU</code>	Se encontraron múltiples variantes con el mismo SKU en distintos productos.	Usá <code>externalVariantId</code> en lugar de SKU para identificar la variante exacta. Consultá

Código	Descripción	Solución
		GET /stock para obtener los identificadores.

Errores de price update (por item)

Código	Descripción	Solución
SELLER_API_VARIANT_NOT_FOUND	No existe una variante con ese <code>externalVariantId</code> o <code>sku</code> . También se devuelve si la variante pertenece a otro comercio.	Verificá que el identificador coincida exactamente.
SELLER_API_PROCESSING_ERROR	Error interno al procesar el item.	Reintentá el request. Si persiste, contactá soporte.
SELLER_API_VARIANT_AMBIGUOUS_SKU	Se encontraron múltiples variantes con el mismo SKU en distintos productos.	Usá <code>externalVariantId</code> en lugar de SKU para identificar la variante exacta.

Errores de chat / CRM

HTTP	Código	Descripción
404	SELLER_API_CHAT_NOT_FOUND	El chat no existe o no pertenece a tu comercio (en <code>GET /chat/:chatId/messages</code> y <code>POST /chat/:chatId/messages</code>).
403	CHAT_INACTIVE	El chat está inactivo/bloqueado (ej. pago expirado).
403	CHAT_RECIPIENT_BLOCKED / CHAT_RECIPIENT_BLOCKED_YOU / CHAT_UNBLOCK_TO_SEND	Relación de bloqueo entre las partes al intentar responder.

Errores de validación (HTTP 400)

Se retornan cuando el body/query del request no cumple el formato esperado:

- Item sin identificador (ni `externalVariantId` ni `sku`) en operaciones de escritura
- Stock negativo
- Operación inválida (debe ser `set`, `increment`, o `decrement`)
- Array `items` vacío o con más de 100 elementos
- Rango de fechas mayor a 7 días en GET /orders

Ejemplos Completos

Ejemplo 1: Sincronización diaria desde ERP

```
curl -X POST {BASE_URL}/seller-api/v1/stock/sync \
-H "x-api-key: mk_tst_tu_api_key_aqui" \
-H "Content-Type: application/json" \
-d '{
  "items": [
    { "externalVariantId": "CAMISA-AZUL-S", "branchExternalId": "SU
C-001", "stock": 15, "operation": "set" },
    { "externalVariantId": "CAMISA-AZUL-M", "branchExternalId": "SU
C-001", "stock": 25, "operation": "set" },
    { "externalVariantId": "CAMISA-AZUL-L", "branchExternalId": "SU
C-001", "stock": 8, "operation": "set" }
  ]
}'
```

Ejemplo 2: Venta en tienda física

```
curl -X POST {BASE_URL}/seller-api/v1/stock/sync \
-H "x-api-key: mk_tst_tu_api_key_aqui" \
-H "Content-Type: application/json" \
-d '{
  "items": [
    { "externalVariantId": "ZAPATO-NEGRO-42", "branchExternalId":
"SUC-001", "stock": 2, "operation": "decrement" }
  ]
}'
```

Ejemplo 3: Actualizar precio

```
curl -X PATCH {BASE_URL}/seller-api/v1/sku \
-H "x-api-key: mk_tst_tu_api_key_aqui" \
-H "Content-Type: application/json" \
-d '{
  "items": [
    { "externalVariantId": "CAMISA-AZUL-M", "price": 29.99 }
  ]
}'
```

```
]
}'
```

Ejemplo 4: Consultar stock por SKU

```
curl -X GET "{BASE_URL}/seller-api/v1/stock?sku=CAMISA-AZUL-M" \
-H "x-api-key: mk_tst_tu_api_key_aqui"
```

Ejemplo 5: Reconciliar órdenes

```
curl -X GET "{BASE_URL}/seller-api/v1/orders?startDate=2026-07-01&endDate=2026-07-08&page=1" \
-H "x-api-key: mk_tst_tu_api_key_aqui"
```

Ejemplo 6: Leer chat (catch-up) y responder

```
# Traer mensajes nuevos desde el ultimo cursor procesado
curl -X GET "{BASE_URL}/seller-api/v1/chat/messages?since=665a1b2c3d4e5f6a7b8c9d0e&limit=50" \
-H "x-api-key: mk_tst_tu_api_key_aqui"

# Responder en un chat
curl -X POST "{BASE_URL}/seller-api/v1/chat/664b2c3d4e5f6a7b8c9d0e1f/messages" \
-H "x-api-key: mk_tst_tu_api_key_aqui" \
-H "Content-Type: application/json" \
-d '{ "content": "Si! Disponible. Te lo reservo?" }'
```

Límites

Concepto	Límite	Nota
Items por request de sync/update	100 máximo	Para inventarios grandes, divíd en múltiples requests.
API Keys por comercio	5 máximo	Suficiente para ERP, POS, app móvil, staging, etc.

Concepto	Límite	Nota
Rango de fechas en /orders	7 días máximo	Dividí en consultas consecutivas para rangos mayores.
Paginación máxima	100 items por página	Default: 20 items por página.
Timeout de webhook	5 segundos	Tu servidor debe responder dentro de este tiempo.
Chat: mensajes por página	100 máximo	Default 50. Paginá por cursor con <code>?since=</code> .
Chat: <code>content</code> por respuesta	5000 chars	En <code>POST /chat/:chatId/messages</code> .
Chat: adjuntos por respuesta	1-10 archivos	Cada uno <code>{ uri, type }</code> .

Buenas Prácticas

1. **Usá `set` para sincronizaciones completas** — Es la operación más segura cuando tu ERP tiene el número exacto de stock.
2. **Usá `increment` / `decrement` para cambios en tiempo real** — Ventas en POS, recepciones, devoluciones.
3. **Guardá el `batchId` de cada response** — Sirve para rastreo y soporte.
4. **Revisá los items con `warning`** — Indican desincronización entre tu sistema y Yummy.
5. **Verificá la firma HMAC** de todos los webhooks — Protege contra payloads falsificados.
6. **Usá el endpoint de órdenes para reconciliación** — Si tu webhook falló, consultá `GET /orders` con el rango de fechas afectado.
7. **Nunca expongas tu API Key** en código frontend, aplicaciones móviles, ni repositorios públicos.
8. **Manejá errores por item** — Si un request de 50 items tiene 2 errores, los otros 48 se procesaron bien. Solo reintentá los que fallaron.
9. **Chat: guardá el `cursor` y usá REST como red de seguridad** — El webhook `chat.message.received` es best-effort. Persistí el `cursor` del último mensaje procesado y, ante cualquier duda o caída, hacé `GET /chat/messages?since={cursor}` para recuperar el gap sin duplicados. Los bots (`isBot: true`) solo aparecen por REST, no por webhook.



¿Dudas técnicas? Contactá al equipo de Yummy y pedí comunicación directa con nuestro equipo de desarrollo para resolver cualquier duda de implementación.