Yummy Marketplace — API de Sincronización para comercios

Versión 3.4 · Octubre 2026 · La URL base se entrega al activar la integración · yummy-marketplace.com/portal-comercios/api

Primeros pasos

La API de Yummy Marketplace conecta tu sistema (ERP, POS, tienda propia o CRM) con tu tienda en Yummy: sincronizas stock y precios, editas tus productos, recibes tus ventas en tiempo real, marcas tus pedidos como listos, concilias tus pagos y atiendes el chat desde tus propias herramientas.

1. **Pide la activación.** Escríbele a tu contacto en Yummy o usa el botón **Soporte** del Portal de Comercios y pide la integración por API. Al activarla recibes la **URL base** de la API.
2. **Recibe tu API Key de pruebas.** El equipo de Yummy crea la key con los permisos que necesitas. Las keys de pruebas empiezan con `mk_tst_`.
3. **Haz tu primera llamada.** Consulta tu stock para ver los identificadores de tus variantes:

```
curl -X GET "{BASE_URL}/seller-api/v1/stock?limit=10" \
  -H "x-api-key: mk_tst_tu_api_key_aqui"
```

4. **Configura tu webhook** (opcional). Si le das a Yummy la URL de tu servidor, recibes cada venta al momento. El secreto para verificar las firmas se muestra **una sola vez**, al crear la key.
5. **Pasa a producción** con una key `mk_prd_`.

> **Tu API Key es secreta. Nunca la compartas en código público ni en repositorios, y no la uses en aplicaciones del lado del cliente (frontend o apps móviles). Úsala solo desde tu servidor.**

Autenticación

Todas las solicitudes deben incluir tu API Key en el header `x-api-key`:

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

Formato de la API Key

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

Permisos

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

Permiso	Permite
`stock:write`	Sincronizar y actualizar stock (`POST /stock/sync`).
`stock:read`	Consultar el stock actual (`GET /stock`).
`price:write`	Actualizar precios (`PATCH /sku`). Solo precios.
`product:write`	Editar el título y la descripción de un producto y corregir su `externalId` (`PATCH /products`); en `PATCH /sku` , actualizar precios y corregir el `externalVariantId` . **Incluye** `price:write` : una key con `product:write` no necesita también `price:write` .
`orders:read`	Consultar órdenes para reconciliación (`GET /orders`).
`chat:read`	Leer el chat por REST (`GET /chat/messages` , `GET /chat/:chatId/messages`) y recibir el webhook `chat.message.received` .
`chat:write`	Responder en el chat (`POST /chat/:chatId/messages`).
`dispatch:write`	Marcar un pedido como listo y solicitar el Yummer (`POST /orders/:orderId/ready-for-dispatch`).

Si tu key no tiene el permiso que pide un endpoint, recibes un error `403` .

Permisos granulares: una key puede tener solo `stock:write` y `stock:read` sin poder tocar precios, solo `price:write` sin acceso al stock, o solo `orders:read` . Puedes tener hasta **5 keys** con combinaciones distintas para distintos sistemas.

> **Los permisos no cambian después de crear la key. Al editar una key solo se pueden cambiar su nombre, su `webhookUrl` y si está activa. Por eso las keys existentes no reciben los permisos nuevos automáticamente: para usar el chat, editar productos o marcar pedidos como listos en un comercio que ya tiene una key, hay que crear una nueva con `chat:read` / `chat:write` , `product:write` o `dispatch:write` . Así una key de stock o precios nunca gana más acceso sin una acción explícita.**

Conceptos clave

Identificadores de variante

Identificador	Uso	¿Único?
`externalVariantId`	Identificador principal de la variante. Se usa para sincronizar stock y actualizar precios, y aparece en las respuestas y en los webhooks. Si no lo configuraste en tu CSV, se generó a partir de tu SKU. Se puede corregir con `PATCH /sku` (ver `variantId`).	Sí, por producto. Evita repetirlo entre productos: un `externalVariantId` repetido da error de ambigüedad.
`sku`	Código SKU de la variante. Sirve para sincronizar stock y actualizar precios siempre que sea único entre tus productos. Si se repite, debes usar `externalVariantId` .	No necesariamente

Identificador	Uso	¿Único?
`externalId`	ID del producto en tu sistema. Aparece en las respuestas como referencia cruzada e identifica el producto en `PATCH /products`. Se puede corregir con `PATCH /products` (ver `postId`).	Sí, por comercio
`variantId`	ID de la variante en Yummy. Lo devuelve `GET /stock`. **Nunca cambia**: es el que permite corregir un `externalVariantId` mal cargado.	Sí
`postId`	ID del producto en Yummy. Lo devuelve `GET /stock`. **Nunca cambia**: es el que permite corregir un `externalId` mal cargado.	Sí

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). `PATCH /sku` acepta además `variantId` solo, sin otro identificador, que se busca por ese ID entre tus productos. La resolución funciona así:

1. Si envías `externalVariantId`, se busca **solo** por ese campo. Si no existe entre tus productos, recibes `SELLER_API_VARIANT_NOT_FOUND`, aunque también hayas enviado un `sku` válido: no se hace una segunda búsqueda por SKU. Si **dos** de tus variantes (en productos distintos) comparten ese `externalVariantId`, recibes `SELLER_API_VARIANT_AMBIGUOUS_EXTERNAL_ID` y no se modifica ninguna: corrige el duplicado con `variantId` + `newExternalVariantId`.
2. Si envías solo `sku`, se busca por SKU entre los productos de tu comercio:
   - **1 resultado:** se usa esa variante (el caso más común si no configuraste `externalVariantId`).
   - **0 resultados:** `SELLER_API_VARIANT_NOT_FOUND` (también si la variante es de otro comercio).
   - **2 o más resultados:** `SELLER_API_VARIANT_AMBIGUOUS_SKU`, con un mensaje que te pide usar `externalVariantId`.

> **¿Cómo sé mi `externalVariantId`? Consulta `GET /stock`: cada variante de la respuesta trae su `externalVariantId`. Si no configuraste uno a mano, es igual a tu SKU.**

Sincronizar stock

POST `/seller-api/v1/stock/sync`

Permiso: `stock:write`

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

Headers

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

Header	Valor	Requerido
`Content-Type`	`application/json`	Sí

Request

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

También puedes identificar la variante por SKU:

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

Campo	Tipo	Requerido	Descripción
`items`	Array	Sí	Lista de actualizaciones (mín. 1, máx. 100).
`items[].externalVariantId`	String	No*	Identificador externo de la variante. Máx. 200 caracteres. Tiene prioridad sobre `sku`.
`items[].sku`	String	No*	SKU de la variante. Máx. 100 caracteres. Si se repite entre productos, devuelve `SELLER_API_VARIANT_AMBIGUOUS_SKU`.
`items[].branchExternalId`	String	No**	ID de la sucursal en tu sistema. Máx. 100 caracteres.
`items[].branchId`	String	No**	ID interno de Yummy para la sucursal.
`items[].stock`	Integer	Sí	Cantidad de unidades. Debe ser ≥ 0.
`items[].operation`	String	Sí	`set`, `increment` o `decrement`.

* Debes enviar al menos uno de `externalVariantId` o `sku`. Si envías los dos, se usa `externalVariantId`.

** Puedes identificar la sucursal con `branchExternalId` (tu código) o `branchId` (ID de Yummy). Si no envías ninguno, se usa la sucursal principal del comercio.

Operaciones

Operación	Qué hace	Cuándo usarla
`set`	Establece el stock en la cantidad exacta. Sobrescribe el valor anterior.	Sincronización diaria completa desde tu ERP: "tengo 25 unidades, punto".
`increment`	Suma la cantidad al stock actual.	Recepción de mercancía o devolución: "llegaron 10 unidades más".
`decrement`	Resta la cantidad del stock actual. Si no alcanza, lo deja en 0 y devuelve un warning.	Venta en tienda física u otro canal: "se vendieron 5 en el POS".

> **Atomicidad: todas las operaciones son atómicas. Si dos sistemas envían actualizaciones al mismo tiempo, no se pisan: cada una ve el stock que dejó la anterior.**

Respuesta (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 available. Stock set to 0."
      },
      {
        "externalVariantId": "ID-INEXISTENTE",
        "status": "error",
        "errorCode": "SELLER_API_VARIANT_NOT_FOUND",
        "errorMessage": "Variant not found"
      }
    ]
  }
}
```

Estados por item

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

Consultar stock

GET `/seller-api/v1/stock`

Permiso: `stock:read`

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

Parámetros de consulta

Parámetro	Tipo	Requerido	Descripción
`externalVariantId`	String	No	Filtra por identificador externo de variante. Máx. 200 caracteres.
`sku`	String	No	Filtra por SKU. Máx. 100 caracteres. Puede devolver varios resultados si el SKU se repite.
`externalId`	String	No	Filtra por ID externo de producto. Máx. 200 caracteres.
`branchExternalId`	String	No	Filtra por ID externo de sucursal.
`branchId`	String	No	Filtra por ID interno de sucursal.
`postId`	String	No	Filtra por ID de producto/publicación.
`page`	Integer	No	Número de página (por defecto 1).
`limit`	Integer	No	Resultados por página, de 1 a 100 (por defecto 20).

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

Respuesta (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: `price:write` (o `product:write`, que lo incluye). Para corregir el `externalVariantId` hace falta `product:write`.

Actualiza el precio de una o más variantes y permite corregir su `externalVariantId`. Cada item se procesa por separado. Los headers son los mismos que en Sincronizar stock.

Request

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

También puedes identificar la variante por SKU:

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

Para corregir un `externalVariantId` mal cargado, identifica la variante por su `variantId` (lo devuelve `GET /stock`):

```
{
  "items": [
    {
      "variantId": "665a1b2c3d4e5f6a7b8c9d0e",
      "newExternalVariantId": "CAMISA-AZUL-M"
    }
  ]
}
```

Campo	Tipo	Requerido	Descripción
`items`	Array	Sí	Lista de actualizaciones (mín. 1, máx. 100).
`items[].externalVariantId`	String	No*	Identificador externo de la variante. Máx. 200 caracteres. Tiene prioridad sobre `sku`.
`items[].sku`	String	No*	SKU de la variante. Máx. 100 caracteres. Error si el SKU es ambiguo.
`items[].variantId`	String	No*	ID de la variante en Yummy (`GET /stock`). No se combina con `externalVariantId` ni con `sku`.
`items[].price`	Number	No**	Nuevo precio de la variante en USD. Debe ser ≥ 0.
`items[].newExternalVariantId`	String	No**	Nuevo `externalVariantId`. Máx. 200 caracteres. **Solo junto con `variantId`** y con una key que tenga `product:write`; si no, todo el request responde `403`. No puede estar en uso por otra de tus variantes.

* Debes enviar `variantId`, o al menos uno de `externalVariantId` o `sku`. Si envías `externalVariantId` y `sku`, se usa `externalVariantId`.

** Debes enviar al menos uno de `price` o `newExternalVariantId`.

Respuesta (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",
        "updatedFields": ["price"],
        "previousPrice": 25.00,
        "newPrice": 29.99
      },
      {
        "externalVariantId": "ID-INEXISTENTE",
        "status": "error",
        "errorCode": "SELLER_API_VARIANT_NOT_FOUND",
        "errorMessage": "Variant not found"
      }
    ]
  }
}
```

`updatedFields` lista los campos que envió el item. `previousPrice` y `newPrice` solo vienen si el item traía `price`; `previousExternalVariantId`, solo si traía `newExternalVariantId` (y entonces `externalVariantId` es el valor nuevo). Si un item enviado por `variantId` falla porque la variante no existe, el resultado trae `variantId` y no `externalVariantId`.

> **Nota de diseño: el precio vive en la variante: es único por variante, sin importar la sucursal. Las imágenes de las variantes todavía no se editan por API.**

Editar productos

PATCH `/seller-api/v1/products`

Permiso: `product:write`

Actualiza el título o la descripción de uno o más productos, y permite corregir su `externalId`. Cada item se procesa por separado. Los cambios se reflejan en la búsqueda. Los headers son los mismos que en Sincronizar stock.

Request

```
{
  "items": [
    { "externalId": "PROD-001", "title": "Camisa azul manga larga", "description": "Algodón 100%" },
    { "postId": "664b2c3d4e5f6a7b8c9d0e1f", "newExternalId": "PROD-002" }
  ]
}
```

Campo	Tipo	Requerido	Descripción
`items`	Array	Sí	Lista de actualizaciones (mín. 1, máx. 100). Cada producto puede aparecer una sola vez.
`items[].externalId`	String	No*	ID externo del producto. Máx. 200 caracteres.
`items[].postId`	String	No*	ID del producto en Yummy (`GET /stock`).
`items[].title`	String	No**	Nuevo título. De 1 a 200 caracteres.
`items[].description`	String	No**	Nueva descripción. Máx. 2000 caracteres. `""` la borra.
`items[].newExternalId`	String	No**	Nuevo `externalId` . Máx. 200 caracteres. **Solo junto con** `postId` . No puede estar en uso por otro de tus productos.

* Debes enviar exactamente uno de `externalId` o `postId` .

** Debes enviar al menos uno de `title` , `description` o `newExternalId` .

Respuesta (200)

```
{
  "batchId": "batch_a1b2c3d4",
  "status": "completed",
  "results": {
    "total": 2,
    "succeeded": 2,
    "failed": 0,
    "items": [
      {
        "externalId": "PROD-001",
        "postId": "664b2c3d4e5f6a...",
        "status": "ok",
        "updatedFields": ["title", "description"]
      },
      {
        "externalId": "PROD-002",
        "previousExternalId": "PROD-02",
        "postId": "664b2c3d4e5f6a...",
        "status": "ok",
        "updatedFields": ["externalId"]
      }
    ]
  }
}
```

`externalId` es el valor después del cambio; `previousExternalId` solo viene si el item traía `newExternalId`.

> **Después de corregir un ID, cámbialo también en tu sistema. Si más adelante subes una carga masiva (CSV) con el ID viejo, ya no encontrará el producto o la variante y creará uno nuevo.**

> **Nota de diseño: las imágenes no se editan aquí: son por variante y todavía no se editan por API.**

Consultar órdenes

GET `/seller-api/v1/orders`

Permiso: `orders:read`

Consulta las órdenes aprobadas de tu comercio en un rango de fechas. Devuelve el mismo formato que el webhook `order.items.sold`, así puedes usar el mismo parser para los dos flujos.

¿Para qué sirve? Si tu webhook falla (timeout, servidor caído, error de red), consulta este endpoint para recuperar las órdenes que no recibiste y reconciliar tu sistema.

Parámetros de consulta

Parámetro	Tipo	Requerido	Descripción
`startDate`	String (ISO 8601)	Sí	Inicio del rango (ej.: `2026-07-01`).
`endDate`	String (ISO 8601)	Sí	Fin del rango (ej.: `2026-07-08`).
`page`	Integer	No	Número de página (por defecto 1).
`limit`	Integer	No	Resultados por página, de 1 a 100 (por defecto 20).

> **Máximo 7 días por consulta. El rango entre `startDate` y `endDate` no puede pasar de 7 días. Si necesitas más, haz varias consultas con rangos consecutivos.**

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

Respuesta (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`: viene en `GET /orders` y en el webhook `order.disbursement.completed`, pero no en `order.items.sold`, porque al momento de la venta el desembolso todavía no se ha procesado. Trae la referencia bancaria del pago móvil enviado a tu comercio: úsala para conciliar con tu estado de cuenta. Su `status` puede ser `pending`, `paid`, `failed`, `processing` o `cancelled`.**

Cada objeto de `orders` tiene la misma estructura que el payload del webhook `order.items.sold`, más el campo `disbursement`. La única otra diferencia es que las órdenes de este endpoint no traen el campo `event` (solo lo llevan los webhooks).

Marcar pedido listo

POST `/seller-api/v1/orders/:orderId/ready-for-dispatch`

Permiso: `dispatch:write`

Es el botón **Pedido listo** del Portal de Comercios, desde tu sistema: tu ERP, POS o CRM marca el pedido como listo y Yummy solicita el Yummer (repartidor) para retirarlo. El resultado es idéntico al del botón del Portal.

> **Necesitas una key nueva. Los permisos de una API Key no cambian después de crearla, así que las keys existentes no reciben `dispatch:write`. Para usar este endpoint, pide una key nueva con este permiso.**

¿Cuándo aplica? Solo a pedidos con envío **Yummy** (Yummer) ya pagados. Funciona con los dos modos de despacho de la sucursal:

Modo de la sucursal	Qué hace el endpoint
Botón manual ("Pedido listo")	Solicita el Yummer de inmediato.
Tiempo estimado de preparación	**Adelanta** la solicitud del Yummer. La solicitud programada ya no se dispara: nunca se piden dos Yummers para el mismo pedido.

Parámetros de ruta

Parámetro	Tipo	Descripción
`orderId`	String (UUID v4)	El `orderId` que recibes en el webhook `order.items.sold` y en `GET /orders`.

El request **no lleva body** ni sucursal: cada orden tiene un solo envío y una sola sucursal. Solo hace falta el header `x-api-key`.

```
POST /seller-api/v1/orders/667a1b2c-3d4e-5f6a-7b8c-9d0e1f2a3b4c/ready-for-dispatch
x-api-key: mk_tst_tu_api_key_aqui
```

Respuesta (200)

La respuesta es igual si el Yummer se solicitó en este llamado o si ya estaba solicitado.

```
{
  "orderId": "667a1b2c-3d4e-5f6a-7b8c-9d0e1f2a3b4c",
  "status": "yummer_requested",
  "requestedAt": "2026-10-19T15:45:00.000Z",
  "source": "api",
  "alreadyRequested": false
}
```

Campo	Descripción
`orderId`	El mismo `orderId` del request.
`status`	Siempre `yummer_requested`.
`requestedAt`	Fecha (ISO 8601) en que se solicitó el Yummer.
`source`	Quién lo solicitó: `api` (esta API), `portal` (el botón del Portal de Comercios), `admin` (el equipo de Yummy) o `auto` (el tiempo estimado de la sucursal, o la solicitud inmediata al momento del pago).
`alreadyRequested`	`true` si el Yummer ya estaba solicitado antes de este llamado; en ese caso **no** se genera otra solicitud.

> **Puedes reintentar sin riesgo. Si el Yummer ya estaba solicitado (por la API, el Portal, el tiempo estimado o al momento del pago), la respuesta es `200` con `alreadyRequested: true`, no un error. Así, si tu primer llamado se cortó por timeout pero sí se procesó, el reintento no se ve como un fallo.**

Si la solicitud del Yummer falla del lado de Yummy, recibes un error y el pedido queda en espera: puedes reintentar el llamado. Los códigos están en Errores de pedido listo.

Chat y CRM

Atiende el chat entre compradores y tu comercio desde tu propio CRM, sin depender del Portal ni de la app. Funciona con dos piezas, como en Stripe, GitHub o Shopify:

- El webhook `chat.message.received` es la señal en tiempo real, con el mejor esfuerzo posible.
- Los endpoints REST con cursor son la red de seguridad: si el webhook falla, recuperas lo que faltó con `?since={cursor}`.

> **El cursor es el `messageId`, no la fecha. Está ordenado y siempre crece, así que nunca pierdes ni duplicas mensajes aunque dos tengan la misma hora. El `timestamp` es solo informativo. Para saber quién escribió, usa `direction`: `inbound` es del comprador y `outbound` es de tu comercio.**

Listar mensajes

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

Permiso: `chat:read`

Lista todos los mensajes de tu comercio (todos sus chats) en orden de cursor. Sirve para ponerte al día después de una caída del webhook.

Parámetro	Tipo	Requerido	Descripción
`since`	String (24 caracteres hex)	No	Devuelve los mensajes con `messageId` mayor que `since`. Omítelo para empezar desde el inicio.
`limit`	Integer	No	Mensajes por página, de 1 a 100 (por defecto 50).

```
{
  "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 (puede haber más); es `null` cuando ya no quedan mensajes. Pagina repitiendo con `?since={nextCursor}` mientras no sea `null`.

Campo	Tipo	Descripción
`id`	String	ID del mensaje (igual a `cursor`).
`chatId`	String	Chat al que pertenece.
`content`	String	Texto plano. `""` si el mensaje era solo de archivos o es un mensaje histórico sin texto.
`files`	Array	Adjuntos `[{ "uri", "type" }]`. `[]` si no hay.
`from` / `to`	String	Autor y destinatario del mensaje.
`direction`	String	`inbound` (del comprador) u `outbound` (de tu comercio).
`isBot`	Boolean	`true` si es un mensaje automático (envío MRW, tracking QR, recordatorios).
`timestamp`	String (ISO 8601)	Fecha de creación (informativa).
`cursor`	String	Cursor de este mensaje (igual a `id`).

Historial de un chat

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

Permiso: `chat:read`

Igual que el anterior, pero de un solo chat. Misma respuesta y misma paginación por cursor; `chatId` va en la ruta y `since` / `limit` funcionan igual.

> Si el `chatId` no existe o no es de tu comercio, recibes `404 SELLER_API_CHAT_NOT_FOUND`.

Responder en un chat

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

Permiso: `chat:write`

Responde en un chat. Usa el mismo flujo que el Portal y la app, así que el comprador recibe la notificación push y tu equipo ve el mensaje en vivo en el Portal.

Campo	Tipo	Requerido	Descripción
`content`	String	No*	Texto de la respuesta. Máx. 5000 caracteres.
`files`	Array	No*	De 1 a 10 adjuntos `{ "uri", "type" }` (`uri` debe ser una URL válida).
`replyTo`	String	No	ID del mensaje al que respondes.

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

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

Respuesta (200): el mensaje creado, con el mismo formato de arriba y `direction: "outbound"`.

> **La respuesta queda a nombre del comercio. La API Key es quien responde, no una persona de tu equipo. No hay separación por sucursal: la key ve y escribe en todos los chats del comercio.**

Errores posibles: `404 SELLER_API_CHAT_NOT_FOUND` (chat inexistente o ajeno), `403 CHAT_INACTIVE` (chat inactivo o bloqueado, por ejemplo por un pago vencido) y `403 CHAT_RECIPIENT_BLOCKED` / `CHAT_RECIPIENT_BLOCKED_YOU` / `CHAT_UNBLOCK_TO_SEND` (hay un bloqueo entre las partes).

Mensajes automáticos (bots)

Los mensajes automáticos (estado de envío MRW, tracking QR, recordatorios de pago) son avisos operativos de la app, no conversaciones para tu CRM. Por eso **nunca llegan por el webhook** `chat.message.received`, que siempre trae `isBot: false`. Si tu CRM quiere verlos, los encuentra en `GET /chat/messages`, marcados con `isBot: true`.

Webhooks

Si tu comercio tiene configurada una URL de webhook, Yummy le envía una notificación HTTP a tu servidor en cada uno de estos eventos:

Evento	Cuándo llega
`order.items.sold`	Cuando un comprador paga una orden con productos de tu comercio.

Evento	Cuándo llega
`order.disbursement.completed`	Unos 15 minutos después, cuando el pago a tu comercio se confirma.
`chat.message.received`	Cuando un comprador te escribe (requiere `chat:read`).

Evento order.items.sold

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

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

```
{
  "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
    }
  ]
}
```

Campo	Tipo	Descripción
`event`	String	Siempre `"order.items.sold"` en este evento.
`orderId`	String	ID único de la orden en Yummy.
`timestamp`	String (ISO 8601)	Fecha y hora de creación de la orden. Es el mismo valor en `order.items.sold`, `order.disbursement.completed` y `GET /orders`: no cambia según cuándo se envía o se consulta.
`buyer`	Object	Datos del comprador: `buyerId`, `fullName`, `firstName`, `lastName`, `email`, `phone`, `documentType` (ej. `"V"`), `documentNumber` (cédula), `documentVerified` y `fiscalAddress` (objeto o `null`).
`shippingAddress`	Object (opcional)	Dirección de envío: `recipientName`, `street`, `city`, `state`, `zipCode`. Solo viene si la orden requiere envío.
`shipping`	Object (opcional)	`couponUrl` (URL del cupón de envío), `guideNumber` (número de guía) y `shippingMethod` (tipo de servicio). Solo viene cuando hay datos del envío.
`payment`	Object	`paymentMethod` (ej. `"pago-movil"`), `usedCashea`, `paymentIntent` (datos del pago del comprador, opcional) y `cashea` (solo si `usedCashea` es `true`; ver Ventas con Cashea).
`disbursement`	Object (opcional)	Solo en `GET /orders` y en `order.disbursement.completed`. Ver Consultar órdenes.
`financials`	Object	Montos de la orden y tu liquidación. Ver Liquidación.
`items`	Array	Productos vendidos de tu comercio.
`items[].externalVariantId`	String	Identificador externo de la variante.
`items[].externalId`	String (opcional)	ID externo del producto en tu sistema.
`items[].sku`	String	SKU de la variante vendida.
`items[].variantId`	String	ID interno de la variante en Yummy.
`items[].branchId`	String	ID interno de la sucursal donde se descontó el stock.
`items[].branchExternalId`	String (opcional)	Tu ID de la sucursal, si lo configuraste.

Campo	Tipo	Descripción
`items[].quantity`	Integer	Unidades vendidas.
`items[].unitPrice`	Number	Precio unitario en USD al momento de la venta.
`items[].totalPrice`	Number	`unitPrice × quantity`, en USD.
`items[].totalPriceBs`	Number	`totalPrice` en bolívares, a la tasa `financials.exchangeRate`.
`items[].stockAfter`	Integer	Stock que queda en Yummy después de la venta.

> **Envío gratis (`freeShipping`): `shippingCost` siempre muestra el costo real del courier, aunque el envío sea gratis, para que sepas cuánto estás asumiendo con la promoción. Si `freeShipping` es `true`, ese costo no se le cobra al comprador y `total = subtotal`. Si es `false`, `total = subtotal + shippingCost`.**

> **Datos del comprador para facturar y entregar: `documentNumber` prioriza la cédula verificada por Yummy (`documentVerified: true`). Si no está verificada, es la que declaró el comprador o, como último recurso, la del titular de la cuenta bancaria que usa para cobrar reembolsos, que puede no coincidir con quien retira el pedido. Si `documentVerified` es `false`, valida la identidad con el documento físico antes de entregar. `fiscalAddress` puede llegar `null`: Yummy todavía no pide la dirección fiscal a todos los compradores.**

Ventas con Cashea

Cuando la venta se pagó con Cashea (`payment.usedCashea: true`), `payment` trae un bloque `cashea` con el detalle de la financiación. Llega igual 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 con la venta) y el monto que **Cashea financia** en cuotas.

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

```
"payment": {
  "paymentMethod": "pago-movil",
  "usedCashea": true,
  "paymentIntent": {
    "bank": "0105",
    "phone": "+584141234567",
    "transactionPhone": "+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" }
    ]
  }
}
```

Campo	Tipo	Descripción
`casheaOrderId`	Number (opcional)	ID de la orden en Cashea, para conciliar con ellos. Puede faltar en ventas antiguas.
`status`	String (opcional)	Estado de la orden en Cashea: `pending`, `confirmed`, `cancelled` o `failed`.
`initialAmount` / `initialAmountBs`	Number	Inicial pagada por el comprador, en USD y en bolívares. Es lo único que Yummy te deposita con la venta.
`financedAmount` / `financedAmountBs`	Number	`subtotal − initialAmount`, en USD y en bolívares. Es lo que Cashea financia en cuotas.
`sellerFee` / `sellerFeeRate`	Number (opcional)	Comisión de Cashea que se descuenta de tu desembolso: monto en USD y tasa como fracción (`0.045` = 4,5%). `0` significa un acuerdo al 0%.
`buyerFee`	Object (opcional)	Fee de Cashea que pagó el comprador encima del precio, en USD: `rate × subtotal = amount`, más `iva` (`ivaRate` sobre `amount`); `total = amount + iva`. Se cobra con la inicial y no es tuyo: se descuenta de tu desembolso. El IVA de este fee no va en `financials.tax`, que corresponde al producto. No viene si no se cobró.
`installments`	Array (opcional)	Cuotas tal como las agendó Cashea, sin la inicial: `installmentNumber` (1..n), `amount` (USD, redondeado a centavos) y `dueDate` (ISO 8601). Por el redondeo, la suma puede diferir en un centavo de `financedAmount`.

> **Cuotas: el bloque muestra el cronograma, no si las cuotas ya se cobraron. Las cuotas las gestionas directamente con Cashea. Las ventas antiguas con Cashea sin inicial registrada traen `usedCashea: true` sin bloque `cashea`, y sin `deductions`, `payout` ni `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 sabes 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 las líneas mayores a 0.
- **`payout`** / **`payoutBs`**: lo que Yummy te deposita. `payoutBs` es igual a `disbursement.amount`.

```
payout = total − Σ deductions.amount
```

`type`	Qué es
`cashea_financed`	Parte del precio que financia Cashea. No es un cobro: la recibes directamente de Cashea en cuotas y nunca pasa por Yummy.
`cashea_seller_fee`	Comisión por vender con Cashea: `rate × base` (el precio completo del producto), descontada de la inicial.
`cashea_buyer_fee`	Fee de Cashea que pagó el comprador (`payment.cashea.buyerFee.total`). Está en `total`, pero no es tuyo.
`shipping`	Costo del courier. Siempre se descuenta: si lo pagó el comprador, está en `total` y va al courier; con envío gratis no está en `total` 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 con Cashea de $20 a tasa 577.5461, inicial de $6.67, comisión de 3,5% al comercio y 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` (3.447,95 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 manda `payoutBs`, que es el monto que efectivamente se depositó.**

Qué debe hacer tu servidor

1. Verificar la firma HMAC-SHA256 (ver Firma de los webhooks).
2. Responder con HTTP `200` lo más rápido posible.
3. Procesar la información en segundo plano (descontar en tu ERP, actualizar tu POS, etc.).

> **Yummy espera la respuesta un máximo de 5 segundos. Si tu servidor tarda más o no responde, el webhook queda registrado como fallido, pero la venta no se cancela. Puedes recuperar las órdenes perdidas con `GET /seller-api/v1/orders`.**

Ejemplo de receptor en Node.js:

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

app.post("/webhook/yummy", express.json(), (req, res) => {
  // 1. Verificar la 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 de inmediato
  res.status(200).json({ received: true });

  // 3. Procesar en segundo plano
  const { event, orderId, items, 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 unos **15 minutos después de la venta**, cuando Yummy confirma que el pago móvil a tu comercio fue exitoso. Es un evento informativo para tu contabilidad: te entrega la referencia bancaria del pago que recibiste.

> **Dos momentos distintos: `order.items.sold` llega al momento del pago (descuentas stock y preparas el envío). `order.disbursement.completed` llega después, cuando tu dinero ya fue depositado (concilias la referencia bancaria). Los dos usan la misma firma, los mismos headers y el mismo formato.**

Llega con los mismos headers que `order.items.sold` y el mismo payload completo de la orden (`buyer` , `shippingAddress` , `shipping` , `payment` , `financials` , `items`), con dos diferencias:

- `event` vale `"order.disbursement.completed"` .
- `disbursement` siempre viene, con `status: "paid"` :

```
"disbursement": {
  "status": "paid",
  "reference": "20260722150502",
  "currency": "VES",
  "amount": 2275.0,
  "transactionId": "9fdfb05d-06b3-53a9-bf96-d3fcc0e0fbc2",
  "paidAt": "2026-07-22T21:05:02.070Z"
}
```

`reference` es la referencia bancaria del pago móvil (úsala para conciliar con tu estado de cuenta), `currency` siempre es `"VES"`, `amount` es el monto depositado (igual a `financials.payoutBs`), `transactionId` es el ID interno de la transacción y `paidAt` es la fecha del pago.

> **Aplican las mismas reglas de entrega que `order.items.sold`: 5 segundos de timeout, sin reintentos, el fallo del webhook no afecta el desembolso, y se envía a todas las API Keys que tengan `webhookUrl`.**

Evento chat.message.received

Se dispara cuando un comprador le escribe a tu comercio, si tienes una API Key activa con `chat:read` y `webhookUrl`. Es de mejor esfuerzo: si falla, recuperas lo que faltó con `GET /chat/messages?since={cursor}` (por eso no hay reintentos).

Nunca recibes tus propias respuestas: el webhook solo va al comercio cuando el mensaje es para él, y lo que responde tu CRM va dirigido al comprador.

Los headers son los mismos que en los eventos de orden, salvo que `X-Webhook-Id` es el `messageId` (en vez del `orderId`), para que puedas evitar procesar dos veces el mismo mensaje.

```
{
  "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
}
```

Campo	Tipo	Descripción
`event`	String	Siempre `"chat.message.received"`.
`messageId`	String	ID del mensaje (igual a `X-Webhook-Id`).
`chatId`	String	Chat al que pertenece.
`cursor`	String	Igual a `messageId`. Retoma el cursor REST desde aquí: `GET /chat/messages?since={cursor}`.
`content`	String	Texto plano (nunca viaja cifrado ni queda en logs). `""` si es solo de archivos.
`files`	Array	Adjuntos `[{ "uri", "type" }]`.

Campo	Tipo	Descripción
`timestamp`	String (ISO 8601)	Fecha del mensaje (informativa; el orden real es por `messageId`).
`from.buyerId`	String	Autor del mensaje (el comprador).
`isBot`	Boolean	Siempre `false` en el webhook: los mensajes automáticos no llegan por aquí.

> **Ponte al día sin duplicados: guarda el cursor del último webhook que procesaste. Si tu servidor estuvo caído, `GET /seller-api/v1/chat/messages?since={cursor}` te devuelve exactamente los mensajes que faltaron.**

Aplican las mismas reglas que en los eventos de orden: firma HMAC-SHA256 sobre el body tal como llega, 5 segundos de timeout, sin reintentos, el fallo no afecta el mensaje, y se envía a todas las keys del comercio con `chat:read` y `webhookUrl`.

Firma de los webhooks (HMAC-SHA256)

Al crear una API Key con `webhookUrl`, Yummy genera un `webhookSecret` de 64 caracteres. **Se muestra una sola vez**, en la respuesta de creación de la key. Si luego cambia la `webhookUrl`, se genera un secreto nuevo.

Cada webhook trae estos headers:

Header	Descripción
`Content-Type`	`application/json`
`X-Webhook-Id`	ID único del evento: `orderId` (eventos de orden) o `messageId` (chat). Úsalo para no procesar dos veces el mismo evento.
`X-Webhook-Timestamp`	Fecha y hora ISO 8601 del envío.
`X-Webhook-Signature`	Firma HMAC-SHA256 del body con tu `webhookSecret`.

Verificación de la 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 de los envíos:

- **Timeout de 5 segundos:** si tu servidor no responde, el envío queda como fallido.
- **Sin reintentos automáticos** en la versión actual.
- **Un fallo no cancela la venta:** el pago se completa igual, sin importar lo que pase con el webhook.

- **Varias keys:** si tu comercio tiene varias API Keys con `webhookUrl`, el evento se envía a todas.

Códigos de error

Errores de autenticación

HTTP	Código	Qué pasó	Solución
401	`SELLER_API_KEY_MISSING`	No se envió el header `x-api-key`.	Incluye el header con tu API Key.
401	`SELLER_API_KEY_INVALID`	La key no es válida o fue revocada.	Verifica la key. Si fue rotada, usa la nueva.
403	`SELLER_API_KEY_PERMISSION_DENIED`	La key no tiene el permiso que pide el endpoint.	Pide al equipo de Yummy una key con ese permiso.

Errores de stock (por item)

Aparecen dentro de `results.items[]` con `status: "error"`:

Código	Qué pasó	Solución
`SELLER_API_VARIANT_NOT_FOUND`	No existe una variante con ese `externalVariantId` o `sku` en tu catálogo (o es de otro comercio).	Verifica que el identificador coincida exactamente con el registrado en Yummy.
`SELLER_API_BRANCH_NOT_FOUND`	No se encontró la sucursal indicada.	Verifica el `branchExternalId` o el `branchId`.
`SELLER_API_NO_STOCK_IN_BRANCH`	Solo con `decrement`: la variante no tiene stock registrado en esa sucursal. Con `set` o `increment` el registro se crea.	Usa `set` para inicializar el stock en esa sucursal.
`SELLER_API_VARIANT_AMBIGUOUS_SKU`	Hay varias variantes con el mismo SKU en distintos productos.	Usa `externalVariantId`. Consulta `GET /stock` para obtener los identificadores.
`SELLER_API_PROCESSING_ERROR`	Error interno al procesar el item.	Reintenta. Si persiste, contacta a soporte.

Errores de precios (por item)

Código	Qué pasó	Solución
`SELLER_API_VARIANT_NOT_FOUND`	No existe una variante con ese `externalVariantId` o `sku` (o es de otro comercio).	Verifica que el identificador coincida exactamente.
`SELLER_API_VARIANT_AMBIGUOUS_SKU`	Hay varias variantes con el mismo SKU en distintos productos.	Usa `externalVariantId`.

Código	Qué pasó	Solución
`SELLER_API_VARIANT_AMBIGUOUS_EXTERNAL_ID`	Dos de tus variantes (en productos distintos) comparten el `externalVariantId`. No se modificó ninguna.	Corrige el duplicado con `variantId` + `newExternalVariantId`.
`VARIANT_EXTERNAL_ID_TAKEN`	El `newExternalVariantId` ya lo usa otra de tus variantes.	Elige otro valor o corrige primero la otra variante.
`SELLER_API_PROCESSING_ERROR`	Error interno al procesar el item.	Reintenta. Si persiste, contacta a soporte.

Errores de productos (por item)

Código	Qué pasó	Solución
`POST_NOT_FOUND_OR_FORBIDDEN`	No existe un producto con ese `externalId` o `postId` entre tus productos (o es de otro comercio).	Verifica el identificador con `GET /stock`.
`POST_EXTERNAL_ID_TAKEN`	El `newExternalId` ya lo usa otro de tus productos.	Elige otro valor o corrige primero el otro producto.
`SELLER_API_DUPLICATE_ITEM_TARGET`	Otro item del mismo request apunta al mismo producto. No se aplicó ninguno de los dos.	Envía un solo item por producto, con todos los campos juntos.
`SELLER_API_PROCESSING_ERROR`	Error interno al procesar el item.	Reintenta. Si persiste, contacta a soporte.

Errores de chat

HTTP	Código	Qué pasó
404	`SELLER_API_CHAT_NOT_FOUND`	El chat no existe o no es de tu comercio (en `GET` y `POST /chat/:chatId/messages`).
403	`CHAT_INACTIVE`	El chat está inactivo o bloqueado (por ejemplo, por un pago vencido).
403	`CHAT_RECIPIENT_BLOCKED` / `CHAT_RECIPIENT_BLOCKED_YOU` / `CHAT_UNBLOCK_TO_SEND`	Hay un bloqueo entre las partes al intentar responder.

Errores de pedido listo

En `POST /orders/:orderId/ready-for-dispatch`:

HTTP	Código	Qué pasó
404	`SELLER_API_ORDER_NOT_FOUND`	La orden no existe o no es de tu comercio (no se confirma si existen órdenes ajenas).
409	`ORDER_NOT_YUMMER_DELIVERY`	El envío de la orden no es Yummy (por ejemplo, MRW o retiro en tienda).

HTTP	Código	Qué pasó
409	`ORDER_CANCELLED`	El pago está fallido, cancelado o vencido, o el despacho fue cancelado.
409	`ORDER_NOT_PAID`	El pago de la orden todavía no está aprobado.
403	`SELLER_API_KEY_PERMISSION_DENIED`	Tu key no tiene el permiso `dispatch:write`.

Errores de validación (HTTP 400)

Se devuelven cuando el body o los parámetros no tienen el formato esperado:

- Item sin identificador en operaciones de escritura: ni `externalVariantId` ni `sku` (en `PATCH /sku`, tampoco `variantId`); en `PATCH /products`, ni `externalId` ni `postId`.
- `newExternalVariantId` sin `variantId`, o `newExternalId` sin `postId`.
- `variantId` junto con `externalVariantId` o `sku`; `externalId` junto con `postId`.
- Item sin nada que cambiar: en `PATCH /sku`, ni `price` ni `newExternalVariantId`; en `PATCH /products`, ni `title`, ni `description`, ni `newExternalId`.
- Stock negativo.
- Operación inválida (debe ser `set`, `increment` o `decrement`).
- Arreglo `items` vacío o con más de 100 elementos.
- Rango de fechas mayor a 7 días en `GET /orders`.
- `orderId` que no es un UUID v4 en `POST /orders/:orderId/ready-for-dispatch`.

Ejemplos completos

Sincronización diaria desde tu 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": "SUC-001", "stock": 15, "operation": "set" },
      { "externalVariantId": "CAMISA-AZUL-M", "branchExternalId": "SUC-001", "stock": 25, "operation": "set" },
      { "externalVariantId": "CAMISA-AZUL-L", "branchExternalId": "SUC-001", "stock": 8, "operation": "set" }
    ]
  }'
```

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": "decremen
    ]
  }'
```

Actualizar un 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 }
    ]
  }'
```

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"
```

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"
```

Leer el chat y responder

```
# Traer los mensajes nuevos desde el último 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": "¡Sí! Disponible. ¿Te lo reservo?" }'
```

Marcar un pedido listo y pedir el Yummer

```
curl -X POST "{BASE_URL}/seller-api/v1/orders/667a1b2c-3d4e-5f6a-7b8c-9d0e1f2a3b4c/ready-for-dispatch" \
  -H "x-api-key: mk_tst_tu_api_key_aqui"
```

Editar un producto

```
curl -X PATCH {BASE_URL}/seller-api/v1/products \
  -H "x-api-key: mk_tst_tu_api_key_aqui" \
  -H "Content-Type: application/json" \
  -d '{
    "items": [
      { "externalId": "PROD-001", "title": "Camisa azul manga larga" }
    ]
  }'
```

Corregir un ID mal cargado

```
# 1. Obtener el variantId y el postId
curl -X GET "{BASE_URL}/seller-api/v1/stock?externalVariantId=CAMISA-AZUL-MM" \
  -H "x-api-key: mk_tst_tu_api_key_aqui"

# 2a. Corregir el externalVariantId de la variante
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": [ { "variantId": "665a1b2c3d4e5f6a7b8c9d0e", "newExternalVariantId": "CAMISA-AZUL-M" } ] }'

# 2b. Corregir el externalId del producto
curl -X PATCH {BASE_URL}/seller-api/v1/products \
  -H "x-api-key: mk_tst_tu_api_key_aqui" \
  -H "Content-Type: application/json" \
  -d '{ "items": [ { "postId": "664b2c3d4e5f6a7b8c9d0e1f", "newExternalId": "PROD-001" } ] }'
```

Límites

Concepto	Límite	Nota
Items por request de stock, precios o productos	100	Para inventarios grandes, divide en varios requests.
API Keys por comercio	5	Suficiente para ERP, POS, app móvil, pruebas, etc.
Rango de fechas en `/orders`	7 días	Divide en consultas consecutivas para rangos mayores.
Resultados por página	100	Por defecto, 20.
Timeout de webhooks	5 segundos	Tu servidor debe responder dentro de este tiempo.
Chat: mensajes por página	100	Por defecto, 50. Pagina por cursor con `?since=`.
Chat: texto por respuesta	5000 caracteres	En `POST /chat/:chatId/messages`.
Chat: adjuntos por respuesta	1 a 10	Cada uno `{ uri, type }`.

Buenas prácticas

1. **Usa `set` para sincronizaciones completas.** Es la operación más segura cuando tu ERP tiene el número exacto de stock.
2. **Usa `increment` / `decrement` para cambios en tiempo real:** ventas en tu POS, recepciones, devoluciones.
3. **Guarda el `batchId` de cada respuesta.** Sirve para rastrear y para pedir soporte.
4. **Revisa los items con `warning`.** Indican que tu sistema y Yummy no están sincronizados.
5. **Verifica la firma HMAC de todos los webhooks.** Te protege de payloads falsos.
6. **Usa el endpoint de órdenes para reconciliar.** Si un webhook falló, consulta `GET /orders` con el rango de fechas afectado.
7. **Nunca expongas tu API Key** en código frontend, apps móviles ni repositorios públicos.
8. **Maneja los errores por item.** Si un request de 50 items tiene 2 errores, los otros 48 se procesaron bien: reintenta solo los que fallaron.
9. **Chat: guarda el cursor y usa REST como red de seguridad.** Persiste el cursor del último mensaje procesado y, ante cualquier caída, usa `GET /chat/messages?since={cursor}` para recuperar lo que faltó sin duplicados. Los mensajes automáticos (`isBot: true`) solo aparecen por REST.

¿Dudas técnicas? Escríbele a tu contacto en Yummy y pide comunicación directa con nuestro equipo de desarrollo.

Historial de versiones

Versión	Fecha	Cambios
3.4	Octubre 2026	Editar título y descripción de productos (`PATCH /products`), corregir `externalId` y `externalVariantId` mal cargados con `postId` / `variantId`, permiso `product:write`, y marcar pedidos como listos para pedir el Yummer (`POST /orders/:orderId/ready-for-dispatch`, permiso `dispatch:write`).
3.2	Septiembre 2026	Liquidación en `financials` (`deductions`, `payout`, `payoutBs`) y fee del comprador en ventas con Cashea.