Topix

Topix Reseller API Documentation

Everything you need to sell game top-ups through Topix: catalog, validation, orders, tracking and signed webhooks.

BASE URL	FORMAT	UPDATED
https://api.topix.one	JSON over HTTPS POST	29 October 2026

Contents

1 Overview

2 Endpoints at a Glance

3 Typical Integration Workflow

4 Quick Start

5 Authentication

6 Account: Balance

7 Games

8 Product Catalog

9 User Validation

10 Submit an Order

11 Order Tracking

12 Order Status Handling

13 Webhooks

14 Errors

15 Rate Limiting

16 Best Practices

17 Appendix: Client Libraries

1. Overview

The Topix Reseller API lets authorised business partners sell game credit, passes and digital products from their own platforms, with real-time processing and full order tracking.

USD
Balances, prices and charges

3× daily
Price refresh (08:00 · 14:00 · 20:00 GMT+8)

HMAC-SHA256
Signed status webhooks

Base URL

All endpoints live under `https://api.topix.one`. Every request is a **POST** with a JSON body, sent from your server.

2. Endpoints at a Glance

ENDPOINT	PURPOSE	RATE LIMIT
`/topix_api_balance`	Your balance and username	20/min
`/topix_games_data`	Games enabled on your account	20/min
`/topix_game_fields`	Input fields a game needs	20/min
`/topix_servers_list`	Servers for games that need one	20/min
`/topix_api_denominations`	Products and your prices (USD)	20/min
`/checkid`	Validate a player before ordering	Unlimited with api_key
`/topix_reseller_topup_api`	Submit an order	Unlimited
`/track_order`	Order status	20/min

The rate limit is shared: one budget of **20 requests per minute per API key** covers every endpoint marked `20/min` together.

3. Typical Integration Workflow

A complete order lifecycle from catalog discovery to fulfilment confirmation.

1. **Setup & discovery** *(cache, refresh 3× daily)*: list games (`topix_games_data`), read each game's fields (`topix_game_fields`), servers (`topix_servers_list`) and denominations (`topix_api_denominations`).
2. **Pre-order validation** *(per transaction)*: validate the player (`checkid`) and check your balance (`topix_api_balance`). Set your webhook URL in the Config tab once.

3. **Submit the order** (`topix_reseller_topup_api`) with game, userid, serverid (if required), denom and a unique reference. The API answers immediately with an `order_id` ; the order then processes asynchronously.
4. **Fulfilment & confirmation**: we POST the final `order_status` to your webhook URL (recommended). If no webhook arrives after ~3 minutes, poll `track_order` .

Critical: unique references
Always use unique business prefixes for order references (e.g. `"yourstore-1703123456-001"`). A reused reference is rejected with `409` . This protects you from double orders on retries.

4. Quick Start

1. Test your API key

cURL

```
curl -X POST https://api.topix.one/topix_api_balance \
  -H "Content-Type: application/json" \
  -H "Origin: https://your-domain.com" \
  -d '{
    "api_key": "YOUR_API_KEY"
  }'
```

2. Get available games

cURL

```
curl -X POST https://api.topix.one/topix_games_data \
  -H "Content-Type: application/json" \
  -H "Origin: https://your-domain.com" \
  -d '{
    "api_key": "YOUR_API_KEY"
  }'
```

3. Submit your first order

cURL

```
curl -X POST https://api.topix.one/topix_reseller_topup_api \
  -H "Content-Type: application/json" \
  -H "Origin: https://your-domain.com" \
  -d '{
    "api_key": "YOUR_API_KEY",
    "game": "mlbb",
    "userid": "123456789",
    "serverid": "2001",
    "denom": "86",
    "reference": "mystore-1703123456-001"
  }'
```

5. Authentication

Every request carries your API key in the JSON body (`api_key`). Find and rotate it in the Config tab of the console.

Required headers

```
Content-Type: application/json
Origin: https://your-domain.com
```

The `Origin` header must contain a domain registered in your Config tab, and the request must come from a **whitelisted server IP**. Requests made from a web browser are refused: your key must never be exposed in frontend code.

- Never expose API keys in frontend JavaScript
- Store API keys as environment variables
- Use HTTPS for all API requests
- Rotate API keys periodically (Config → Regenerate)

6. Account: Balance

`POST` `https://api.topix.one/topix_api_balance`

Test your API key and read your balance in real time. Balances are held and charged in USD.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key

Response

```
{
  "code": "200",
  "balance": 173.14,
  "currency": "USD",
  "username": "yourstore"
}
```

7. Games

Get available games

`POST` `https://api.topix.one/topix_games_data`

Every game code enabled on your account, with display name and artwork (`"Name|image_url"`).

Response

```
{
  "code": "200",
  "info": {
    "mlbb": "Mobile Legends|https://…/ml.png",
    "pubgm": "PUBG Mobile|https://…/pubg.png"
  }
}
```

Get game fields

`POST` `https://api.topix.one/topix_game_fields`

The fields a game needs. Field names are under `info.fields`; possible values: `userid`, `serverid`, `charname`. `info.inputs` adds a label, type, whether it is required and the allowed values for dropdown fields.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key
`game`	Yes	Game code from the games list

Request

```
{
  "api_key": "your-api-key-here",
  "game": "mlbb"
}
```

Response

```
{
  "code": "200",
  "info": {
    "fields": [
      "userid",
      "serverid"
    ],
    "notes": "Server ID is required for Mobile Legends"
  }
}
```

Get server list

`POST` `https://api.topix.one/topix_servers_list`

Available servers for games that require server selection.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key

PARAMETER	REQUIRED	DESCRIPTION
`game`	Yes	Game code

Request

```
{
  "api_key": "your-api-key-here",
  "game": "genshin"
}
```

8. Product Catalog

`POST` `https://api.topix.one/topix_api_denominations`

All products and your prices for a game, as `{ denom: price }`. Use the key exactly as returned for `denom` when ordering.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key
`game`	Yes	Game code
`currency`	No	Only `"USD"` is accepted

Request

```
{
  "api_key": "your-api-key-here",
  "game": "mlbb"
}
```

Response (USD)

```
{
  "86": 1.11,
  "172": 2.18,
  "257": 3.12,
  "344": 4.35,
  "706": 8.54,
  "Weekly Elite Pack": 0.71
}
```

Pricelist refresh schedule

Prices refresh **3 times daily** at `08:00`, `14:00` and `20:00` (GMT+8). Sync your local cache after each window.

Pro tip

Use the smallest denomination for test purchases during integration.

9. User Validation

`POST` `https://api.topix.one/checkid`

Verify a player exists before submitting an order. **Recommended to prevent failed orders.** The API key is optional for this endpoint.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	No	With it: unlimited. Without it: 20 requests per minute per IP
`game`	Yes	Game code
`userid`	Yes	Player ID
`serverid`	No	For games that need a server

Request

```
{
  "game": "mlbb",
  "userid": "123456789",
  "serverid": "2001"
}
```

Success response

```
{
  "code": "200",
  "username": "PlayerName",
  "message": "User ID is valid"
}
```

Error response

```
{
  "code": "404",
  "message": "User ID not found"
}
```

- Always validate before order submission to prevent failures
- Some games require a server ID, others don't. Check the game fields first

10. Submit an Order

`POST` `https://api.topix.one/topix_reseller_topup_api`

No rate limit, and it doesn't count toward your 20 requests per minute. The price is deducted from your balance on submission and refunded automatically if the order ends in `error` or `cancelled`.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key
`game`	Yes	Game code
`userid`	Yes	Player ID (as required by the game fields)
`serverid`	No	For games that need a server
`charname`	No	For games that ask for a character name
`denom`	Yes	A key from the denominations endpoint
`reference`	Yes	Your unique order reference (max 100 characters)

Request

```
{
  "api_key": "your-api-key-here",
  "game": "mlbb",
  "userid": "123456789",
  "serverid": "2001",
  "denom": "86",
  "reference": "mystore-1703123456-001"
}
```

Success response

```
{
  "status": "success",
  "order_id": "TPXM2K9F1A3B4C5",
  "message": "Order submitted successfully",
  "reference": "mystore-1703123456-001"
}
```

Error response

```
{
  "status": "error",
  "code": "402",
  "message": "Insufficient balance",
  "required": 1.11,
  "currency": "USD"
}
```

Unique references

```
// Good: unique business prefix
"yourstore-1703123456-001"
"yourstore-1703123456-002"

// Bad: generic references
"order123"
"test001"
```

11. Order Tracking

`POST` `https://api.topix.one/track_order`

Check the status of a submitted order. **Counts toward your 20 requests per minute.** Gift-card orders also return the delivered `voucher` code (and `sn` when there is one) once successful.

PARAMETER	REQUIRED	DESCRIPTION
`api_key`	Yes	Your API key
`order_id`	Yes	The `order_id` returned when you submitted the order

Request

```
{
  "api_key": "your-api-key-here",
  "order_id": "TPXM2K9F1A3B4C5"
}
```

Successful

```
{
  "order_status": "successful",
  "order_id": "TPXM2K9F1A3B4C5",
  "reference": "mystore-1703123456-001"
}
```

Pending

```
{
  "order_status": "pending",
  "order_id": "TPXM2K9F1A3B4C5",
  "reference": "mystore-1703123456-001"
}
```

Error

```
{
  "order_status": "error",
  "order_id": "TPXM2K9F1A3B4C5",
  "reference": "mystore-1703123456-001",
  "message": "Invalid user ID"
}
```

Cancelled

```
{
  "order_status": "cancelled",
  "order_id": "TPXM2K9F1A3B4C5",
  "reference": "mystore-1703123456-001",
  "message": "Order timed out"
}
```

12. Order Status Handling

Every order result is given by the `order_status` field. Handle each status accordingly:

STATUS	MEANING	ACTION
`pending`	We have received the order and are processing it. No action needed - wait for the status to update.	Still processing, wait
`successful`	The order has been executed and delivered to the player.	Deliver to customer

STATUS	MEANING	ACTION
`error`	There was an issue with the order. Check the "message" field for details. Your balance is refunded.	Notify & investigate
`cancelled`	The order timed out and was cancelled - usually a temporary issue with the game. Your balance is not charged.	Not charged, retry later

Automated retry games

Some games are retried automatically because retrying often resolves transient errors. If you receive `error` for these, **do not refund your customer immediately**; check again after a short while. If such an order later succeeds, it is charged again and you'll receive a `successful` webhook. Currently: `pubgm`.

Suggested integration flow

1. Submit the order and store the returned `order_id`.
2. **Wait for the webhook** (typically within 3 minutes). Configure the URL in the Config tab.
3. If no webhook arrives after **3 minutes**, use `track_order`.
4. Handle the final status (`successful`, `error` or `cancelled`).

IP whitelisting

All IPs that make API requests must be whitelisted. Add your server IPs in the Config tab before making API calls.

13. Webhooks

Every webhook is signed with your **webhook secret** (`whsec_…`, in Config → Webhook), not your API key, so the key that spends money never has to live in your webhook code. Rotate either one independently. Always verify before processing.

Sent as a `POST` to your webhook URL whenever an order's status changes. Failed deliveries are retried after 10 s and 60 s.

POST https://your-server.com/webhook/topix

```
{
  "order_reference": "mystore-1703123456-001",
  "order_status": "successful",
  "message": "Order has been delivered",
  "timestamp": 1703123456,
  "signature": "LntKiBVK/xeEEmI399csISFXzy7jeB6JR0vT+c5QGB8="
}
```

Header

```
X-Topix-Signature: t=1703123456,v1=5f2c8a…e91b
```

Two signatures: use the header

1. **Header (recommended).** `v1` = hex HMAC-SHA256 of `"{t}.{raw request body}"` with your webhook secret. It covers every field and the send time; reject anything older than ~5 minutes to stop replays. Verify the raw bytes, before parsing JSON.
2. **Body field (simple).** `signature` = Base64 HMAC-SHA256 of `{order_reference}|{order_status}|{message}` with your webhook secret.

Python

```
import hmac, hashlib, time

# Recommended - header: covers the whole body and the send time
def verify_header(raw_body: bytes, header: str, secret: str, tolerance=300) -> bool:
    parts = dict(p.split("=", 1) for p in header.split(","))
    ts = int(parts["t"])
    if abs(time.time() - ts) > tolerance:
        return False
    expected = hmac.new(secret.encode(), f"{ts}.".encode() + raw_body, hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, parts["v1"])

# Simple - body field
import base64
def verify_body(payload: dict, secret: str) -> bool:
    canonical = f"{payload['order_reference']}|{payload['order_status']}|{payload['message']}"
    expected = base64.b64encode(hmac.new(secret.encode(), canonical.encode(),
hashlib.sha256).digest()).decode()
    return hmac.compare_digest(expected, payload.get("signature", ""))
```

Node.js

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

// Recommended - header: covers the whole body and the send time
function verifyHeader(rawBody, header, secret, tolerance = 300) {
  const parts = Object.fromEntries(String(header).split(",").map((p) => p.split("=")));
  const ts = Number(parts.t);
  if (!ts || Math.abs(Date.now() / 1000 - ts) > tolerance) return false;
  const expected = crypto.createHmac("sha256", secret).update(`${ts}.${rawBody}`).digest("hex");
  const a = Buffer.from(expected), b = Buffer.from(parts.v1 || "");
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}

// Simple - body field
function verifyBody(payload, secret) {
  const canonical = `${payload.order_reference}|${payload.order_status}|${payload.message}`;
  const expected = crypto.createHmac("sha256", secret).update(canonical).digest("base64");
  const a = Buffer.from(expected), b = Buffer.from(payload.signature || "");
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}
```

PHP

```php
// Recommended - header: covers the whole body and the send time
function verifyHeader(string $rawBody, string $header, string $secret, int $tolerance = 300): bool {
    parse_str(str_replace(',', '&', $header), $p);
    $ts = (int)($p['t'] ?? 0);
    if (!$ts || abs(time() - $ts) > $tolerance) return false;
    $expected = hash_hmac('sha256', $ts . '.' . $rawBody, $secret);
    return hash_equals($expected, $p['v1'] ?? '');
}
// usage: verifyHeader(file_get_contents('php://input'), $_SERVER['HTTP_X_TOPIX_SIGNATURE'] ?? '',
$secret)

// Simple - body field
function verifyBody(array $payload, string $secret): bool {
    $canonical = $payload['order_reference'] . '|' . $payload['order_status'] . '|' .
$payload['message'];
    $expected = base64_encode(hash_hmac('sha256', $canonical, $secret, true));
    return hash_equals($expected, $payload['signature'] ?? '');
}
```

- Always use a timing-safe comparison
- Reject webhooks whose `t` is more than ~5 minutes from your clock (replay protection)
- Reject any webhook with a missing signature
- Rotating the webhook secret takes effect immediately; regenerating your API key does not affect webhooks

14. Errors

Errors return a non-2xx HTTP status and a JSON body with `code` and `message` (order endpoints also include `"status": "error"`).

HTTP	MEANING
`400`	Invalid or missing parameters (unknown denom, missing serverid, ...)
`401`	Missing or invalid api_key
`402`	Insufficient balance
`403`	Origin not registered, IP not whitelisted, browser request, or account not active
`404`	Game, order or player not found
`409`	Duplicate reference (includes the existing order_id)
`429`	Rate limit exceeded (see the Retry-After header)
`502`	Supplier temporarily unavailable; safe to retry

15. Rate Limiting

One limit covers every endpoint together: **20 requests per minute per API key**. Balance, games, fields, servers, prices and tracking all count toward it. **Submitting orders and validating user IDs are unlimited** and don't count.

REQUESTS	RATE LIMIT	WINDOW
Order submission	No limit	-
User validation, with api_key	No limit	-
All other endpoints, with api_key	20 requests	per minute per API key
User validation without api_key	20 requests	per minute per IP

Respect rate limits and use exponential backoff on `429` responses.

16. Best Practices

> **Unique reference generation.** Always use unique business prefixes for order references to prevent duplicate orders.

> **Prefer webhooks.** Webhooks are instant and do not count toward your rate limits. Poll only as a fallback.

> **Cache catalog data.** Games, fields, servers and denominations change only at the 08:00, 14:00 and 20:00 GMT+8 refresh.

> **Security.** Never expose API keys in frontend code. Keep them in environment variables or a secret store.

Production-first testing

Topix runs a **production-only environment**. Test with real transactions on the smallest denominations. The "Try it" panels on the API Docs page in the console call production with your account, exactly like your integration will.

1. Get approved and copy your API key from the Config tab
2. Register your domain and whitelist your server IP
3. Test the balance endpoint
4. Test user validation
5. Submit small-denomination orders
6. Implement webhooks and tracking, then go live

17. Appendix: Client Libraries

A complete client covering every endpoint. Each one has the same method names and payloads, so you can switch language without re-reading the reference.

Python

```
import os
import uuid
import requests

API_KEY = os.getenv("TOPIX_API_KEY")
DOMAIN = "yourdomain.com"          # A domain registered in the Config tab
PREFIX = "YOURSTORE"               # Your unique business prefix

BASE = "https://api.topix.one"


def generate_reference(prefix: str) -> str:
    """Generate a unique order reference with your business prefix."""
    return f"{prefix}-{uuid.uuid4().hex[:12]}"


class TopixAPI:
    def __init__(self, userid=None, serverid=None, charname=None,
                 denom=None, reference=None, game=None):
        self.userid = userid
        self.serverid = serverid
        self.charname = charname
        self.game = game
        self.denom = denom
        self.reference = reference or generate_reference(PREFIX)
        self.session = requests.Session()
        self.session.headers.update({"Origin": DOMAIN})

    def _post(self, path, payload, with_key=True):
        body = {k: v for k, v in payload.items() if v is not None}
        if with_key:
            body["api_key"] = API_KEY
        return self.session.post(f"{BASE}{path}", json=body, timeout=30).json()

    def check_id(self):
        return self._post("/checkid", {
            "game": self.game, "userid": self.userid,
            "serverid": self.serverid, "charname": self.charname,
        })

    def get_pricelist(self):
        return self._post("/topix_api_denominations", {"game": self.game})

    def submit_order(self):
        return self._post("/topix_reseller_topup_api", {
            "game": self.game, "userid": self.userid,
            "serverid": self.serverid, "charname": self.charname,
            "denom": self.denom, "reference": self.reference,
        })

    def track_order(self, order_id):
```

```
        # 4 order statuses: pending, successful, error, cancelled
        return self._post("/track_order", {"order_id": order_id})

    def get_balance(self):
        return self._post("/topix_api_balance", {})

    def get_game_data(self):
        return self._post("/topix_games_data", {})

    def get_server_data(self):
        return self._post("/topix_servers_list", {"game": self.game})

    def get_game_fields(self):
        return self._post("/topix_game_fields", {"game": self.game})


# --- Usage Example ---
api = TopixAPI(userid="123456789", serverid="2001", game="mlbb", denom="86")

print(api.check_id())                # 1. Validate the player
order = api.submit_order()           # 2. Submit the order
print(order)
print(api.track_order(order["order_id"]))  # 3. Track the order
```

Node.js

```
// Node 18+ (built-in fetch)
const API_KEY = process.env.TOPIX_API_KEY;
const DOMAIN = "yourdomain.com";
const BASE = "https://api.topix.one";
const PREFIX = "YOURSTORE";

const reference = () => `${PREFIX}-${Date.now()}-${Math.random().toString(16).slice(2, 8)}`;

async function post(path, payload, withKey = true) {
  const body = Object.fromEntries(Object.entries(payload).filter(([, v]) => v != null));
  if (withKey) body.api_key = API_KEY;
  const res = await fetch(BASE + path, {
    method: "POST",
    headers: { "Content-Type": "application/json", Origin: DOMAIN },
    body: JSON.stringify(body),
  });
  return res.json();
}

export const topix = {
  checkId: ({ game, userid, serverid, charname }) => post("/checkid", { game, userid, serverid,
charname }),
  getPricelist: (game) => post("/topix_api_denominations", { game }),
  submitOrder: ({ game, userid, serverid, charname, denom, ref = reference() }) =>
    post("/topix_reseller_topup_api", { game, userid, serverid, charname, denom, reference: ref }),
  trackOrder: (order_id) => post("/track_order", { order_id }),
  getBalance: () => post("/topix_api_balance", {}),
  getGameData: () => post("/topix_games_data", {}),
  getServerData: (game) => post("/topix_servers_list", { game }),
  getGameFields: (game) => post("/topix_game_fields", { game }),
};

// --- Usage Example ---
const api = topix;
```

```
const player = { game: "mlbb", userid: "123456789", serverid: "2001" };
console.log(await api.checkId(player));
const order = await api.submitOrder({ ...player, denom: "86" });
console.log(await api.trackOrder(order.order_id));
```

PHP

```
<?php
const API_KEY = getenv('TOPIX_API_KEY');
const DOMAIN  = 'yourdomain.com';
const BASE    = 'https://api.topix.one';

function topix_post(string $path, array $payload, bool $withKey = true): array {
    $payload = array_filter($payload, fn($v) => $v !== null);
    if ($withKey) $payload['api_key'] = API_KEY;
    $ch = curl_init(BASE . $path);
    curl_setopt_array($ch, [
        CURLOPT_POST => true,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_TIMEOUT => 30,
        CURLOPT_HTTPHEADER => ['Content-Type: application/json', 'Origin: ' . DOMAIN],
        CURLOPT_POSTFIELDS => json_encode($payload),
    ]);
    $res = curl_exec($ch);
    curl_close($ch);
    return json_decode($res, true) ?? [];
}

// --- Usage Example ---
$check = topix_post('/checkid', ['game' => 'mlbb', 'userid' => '123456789', 'serverid' => '2001']);
$order = topix_post('/topix_reseller_topup_api', [
    'game' => 'mlbb', 'userid' => '123456789', 'serverid' => '2001',
    'denom' => '86', 'reference' => 'yourstore-' . time() . '-001',
]);
$status = topix_post('/track_order', ['order_id' => $order['order_id']]);
```

Webhook receiver

A production-ready endpoint that receives and verifies our webhook callbacks.

FastAPI

```
import hmac, hashlib, json, os, time
from fastapi import FastAPI, HTTPException, Request

app = FastAPI()
WEBHOOK_SECRET = os.getenv("TOPIX_WEBHOOK_SECRET")   # whsec_… from Config → Webhook
TOLERANCE = 300                                       # reject webhooks older than 5 minutes


def verify(raw_body: bytes, header: str) -> bool:
    """X-Topix-Signature: t=<unix>,v1=<hex HMAC-SHA256 of "t.raw_body">"""
    try:
        parts = dict(p.split("=", 1) for p in header.split(","))
        ts, sig = int(parts["t"]), parts["v1"]
    except (KeyError, ValueError):
        return False
    if abs(time.time() - ts) > TOLERANCE:
```

```
        return False
    expected = hmac.new(WEBHOOK_SECRET.encode(), f"{ts}.".encode() + raw_body,
hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, sig)


@app.post("/webhook/topix")
async def receive_webhook(request: Request):
    raw = await request.body()                      # verify the exact bytes received
    if not verify(raw, request.headers.get("X-Topix-Signature", "")):
        raise HTTPException(status_code=401, detail="Invalid signature")
    payload = json.loads(raw)

    if payload["order_status"] == "successful":
        print(f"Order {payload['order_reference']} delivered!")       # mark delivered
    elif payload["order_status"] == "error":
        print(f"Order {payload['order_reference']} error: {payload['message']}")
    elif payload["order_status"] == "cancelled":
        print(f"Order {payload['order_reference']} cancelled")         # never charged
    return {"status": "ok"}

# Run: uvicorn main:app --host 0.0.0.0 --port 8000
```

Express

```
import express from "express";
import crypto from "node:crypto";

const app = express();
const WEBHOOK_SECRET = process.env.TOPIX_WEBHOOK_SECRET; // whsec_… from Config → Webhook

function verify(rawBody, header = "") {
  const parts = Object.fromEntries(header.split(",").map((p) => p.split("=")));
  const ts = Number(parts.t);
  if (!ts || !parts.v1 || Math.abs(Date.now() / 1000 - ts) > 300) return false; // 5-min window
  const expected = crypto.createHmac("sha256",
WEBHOOK_SECRET).update(`${ts}.${rawBody}`).digest("hex");
  const a = Buffer.from(expected), b = Buffer.from(parts.v1);
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}

// express.raw keeps the exact bytes - re-serialising JSON would break the signature
app.post("/webhook/topix", express.raw({ type: "application/json" }), (req, res) => {
  const raw = req.body.toString("utf8");
  if (!verify(raw, req.get("X-Topix-Signature"))) return res.status(401).json({ detail: "Invalid
signature" });
  const { order_reference, order_status, message } = JSON.parse(raw);
  // update your order by order_reference …
  res.json({ status: "ok" });
});

app.listen(8000);
```