Jalpi Product Integration Guide

Nov 17, 2026 · @Konceptwise

Version 2.0 · For engineering teams of **new vendor products** and of **existing Jalpi products** · Replaces the Vendor Integration Guide v1.0 and the `/Developers/ProductIntegration` page.

This guide lists everything a product must do to work with **Jalpi Account** (`account.jalpi.com`):

- how customers sign in, both from Jalpi and from your own app
- how users are registered and linked
- how invoices and payments work
- the security rules every product must follow

Your application stays yours. Any language or framework works, because everything is plain HTTPS and JSON.

0. Read this first: which parts apply to you

Section	New vendor product	Existing Jalpi product
1–2 Concepts, setup	Required	Check your config
3 Sign-in flows (A–D)	A, B, C required, D if you need silent login	Already built; **D changes** (see 12)
4 register-global	Required	Already built; **send ClientSecret**
5 Linking existing users	Required	Already built
6 Invoices and payment	Required	Already built; **send ClientSecret + SourceRequestId**
7 Payment webhook	Required	Already built
8 Support link	Required	Already built
9 Referral (`?ref=`)	Only if Jalpi enables it for you	As today
10 Jalpi-owned extras (shared cookie)	**Not available**	Only for products on `*.jalpi.com`

Section	New vendor product	Existing Jalpi product
11 Security rules	Required from day one	Required: deadlines in section 12
12 What existing products must change	—	**Your action list**
13 Commercial rules	Required	—
14–16 Errors, checklists	Required	Use for re-testing

1. How it works

Jalpi Account is the parent platform. It owns:

- customer accounts, sign-in and OTP
- billing profiles and GST
- invoices and payment collection
- refunds
- support tickets

Your product owns its features, its own user table and its own session.

The usual journey:

1. The customer signs in at account.jalpi.com and opens your product from **My Products**, or clicks **Sign in with Jalpi** inside your app.
2. Jalpi sends the browser to your callback page with a one-time **ticket**.
3. Your **server** exchanges the ticket for the customer's details, then finds or creates the local user and starts your own session.
4. Your server tells Jalpi your user id for this customer (`register-global` or `confirm-link`).
5. The customer picks a plan. Your server asks Jalpi for an invoice, and Jalpi returns a payment link.
6. The customer pays on Jalpi's payment page. Jalpi issues the GST invoice.
7. Jalpi sends a signed, encrypted **webhook** to your server.
8. Your server verifies it and switches on the plan.

Words used in this guide

Term	Meaning
`AuthBase`	`https://account.jalpi.com`
`ClientKey`	Your product's **public** id. It may appear in browser URLs.
`ClientSecret`	Your product's **private** key. It authorises server calls and verifies and decrypts webhooks.
`ExternalUserId`	**Your** user id for a customer, as a string. It is the join key on both sides and must never change.
Jalpi `userId`	Jalpi's own GUID for the customer. Never send it as `ExternalUserId`.
`SourceRequestId`	**Your** order id, used as the idempotency key for invoices.
`ExternalPaymentId`	The payment id that **Jalpi** creates and returns. The webhook refers to it.

2. Setup

What Jalpi gives you

Item	Where to keep it
`AuthBase`	config
`ClientKey`	config
`ClientSecret`	**server only**: environment variable or secret store
Support product id	config
(Vendors) vendor portal login	your team

What you give Jalpi

Item	Example	Used for
Server domain (`Domain`)	`https://app.acme.in`	the webhook is sent to `{Domain}/api/payment-responce`
Public URL (`PublicUrl`)	`https://acme.in`	the customer-facing site; also allowed for `RedirectUri`
Callback path (`CallbackPath`)	`/jalpi/callback`	where Jalpi sends sign-in tickets
Link path (`LinkPath`)	`/jalpi/link`	where Jalpi sends link tickets (section 5)
(Optional) referral path	`/signup`	section 9
(Vendors) PAN, bank, GSTIN if any		payouts

Jalpi registers these in **Admin → Websites**. A `RedirectUri` you send must be on your `Domain` or `PublicUrl` host, or a subdomain of either.

Configuration names we recommend

```
JALPI_AUTH_BASE=https://account.jalpi.com
JALPI_CLIENT_KEY=YOUR_CLIENT_KEY
JALPI_CLIENT_SECRET=********        # server-side secret store only
JALPI_SUPPORT_PRODUCT_ID=...
```

Technical requirements

- **HTTPS everywhere**, with a valid certificate on your callback, link and webhook pages.
- **Vendors must use their own domain.** A vendor app must not run on `*.jalpi.com`. Jalpi's login cookie is shared across `.jalpi.com`, so only Jalpi's own products may live there.
- **All `/api/...` calls in this guide are server-to-server.** Never call them from a browser or a mobile app, and never put the `ClientSecret` in a browser page, a mobile app or a log.
- Prices are in **INR** for vendor products in this version.

3. Sign-in flows

There are four ways a customer reaches your product signed in. **A** and **B** look the same to your code, because both end at your **callback** with a ticket.

Flow	Starts at	Ends at	You call
A. Open from Jalpi Account	account.jalpi.com → My Products → **Open**	your `CallbackPath?ticket=`	`ticket/exchange`
B. "Sign in with Jalpi" in your app	your login page	your `RedirectUri?ticket=`	`ticket/exchange`
C. First connection from Jalpi Account	account.jalpi.com → My Products → **Link**	your `LinkPath?ticket=`	`validate-ticket` → `confirm-link`
D. Passive (silent) login	your server already knows the user	your callback `?ticket=`	`passive-token` → `ticket/exchange`

A. Open from Jalpi Account

When a signed-in customer clicks **Open** on your product card, Jalpi sends the browser to:

```
GET {AuthBase}/Sso/Login?clientKey=YOUR_CLIENT_KEY&redirectUri={Domain}{CallbackPath}
```

The customer is already signed in, so Jalpi **immediately** redirects to your callback with a ticket. The customer sees no login screen. Handle it exactly like flow B, step 3 onwards.

If the customer's access to your product has been **revoked** by Jalpi, no ticket is issued.

B. "Sign in with Jalpi" from your app

Step 1. Redirect the browser to Jalpi.

```
GET {AuthBase}/Sso/Login?ClientKey=YOUR_CLIENT_KEY&RedirectUri=https%3A%2F%2Facme.in%2Fjalpi%2Fcallback
```

- `RedirectUri` must be URL-encoded, absolute `https`, and on your registered `Domain` or `PublicUrl` host. If you leave it out, Jalpi uses `{Domain}{CallbackPath}`.
- Never build `RedirectUri` from user input.

Step 2. The customer signs in at Jalpi with a password or OTP. Jalpi owns this screen.

Step 3. Jalpi redirects to your callback with a ticket.

```
https://acme.in/jalpi/callback?ticket=3f2a9c0e5b7d4e1f8a6b2c9d0e1f2a3bXy7...
```

Step 4. Split the ticket. The first 32 characters are the ticket id, and the rest is the ticket secret.

```
ticketId = ticket.substring(0, 32)
secret   = ticket.substring(32)
```

Step 5. Exchange it from your server.

```
POST {AuthBase}/api/auth/ticket/exchange?ticket={ticketId}&secret={secret}
```

Response `200`:

```
{
  "success": true,
  "accessToken": "...",
  "refreshToken": "...",
  "expiresAt": "2026-11-17T10:30:00Z",
  "userId": "8f6c1e2a-...",
  "email": "customer@example.com",
  "mobileNumber": "919876543210",
  "username": "customer",
  "externalUserId": "acme-user-42",
  "referralCode": null
}
```

`401` means the ticket is invalid, expired or already used. Start sign-in again.

Step 6. Map the user and start your session.

- If `externalUserId` is set, it is your user. Load that user.
- If it is `null`, find or create a local user by `email`, then call **`register-global`** (section 4) with your new id.

- Then start **your own** session as usual. Keeping Jalpi's `accessToken` and `refreshToken` is optional; your session is yours.

Ticket rules

- A ticket works **once** and expires after about **10 minutes**. Exchange it straight away, server-side.
- Do not log the ticket, and do not keep it in the URL after use. Redirect to a clean URL.

C. First connection from Jalpi Account (Link)

When a customer adds your product from **My Products** before Jalpi knows your user id for them, Jalpi sends the ticket to your **`LinkPath`** instead of your callback:

```
{Domain}{LinkPath}?ticket=...
```

Your link page must:

1. **Split** the ticket (as in B4).
2. **Peek** at it to learn who the customer is. This does **not** use the ticket up.

```
POST {AuthBase}/api/auth/validate-ticket
{ "Ticket": "TICKET_ID", "Secret": "TICKET_SECRET" }
→ { "success": true, "profile": { "email": "...", "mobile": "...",
"username": "...", "externalId": null } }
```

3. **Find or create** the local user for that email. If your product requires it, show your own sign-up or consent step first.
4. **Confirm the link.** This **uses the ticket up**.

```
POST {AuthBase}/api/auth/confirm-link
{ "Ticket": "TICKET_ID", "Secret": "TICKET_SECRET", "ExternalUserId": "acme-
user-42" }
→ { "success": true }
```

5. Start your session. The product now shows as **Active** in the customer's Jalpi account.

Rules

- `ExternalUserId` is mandatory in `confirm-link`. Without it the link stays **Pending**.
- **Peek first, confirm last.** Do **not** call `ticket/exchange` on a link ticket, because that also uses it up.

- Errors:
  - `401` : invalid or expired ticket
  - `403` : access revoked
  - `409` : this Jalpi user is already linked to a **different** `ExternalUserId`

The same steps work when **your app** starts linking for a user who already exists on your side. Use flow B's URL with `&Prompt=link` added, and point `RedirectUri` at your link page.

D. Passive (silent) login

Use this when your server already knows who the user is (for example, they signed in with your own login) and you want them signed in to Jalpi too, without a Jalpi login screen.

Secure token flow. This is **required for new vendor products** and **recommended for all products**.

Step 1. Your server asks Jalpi for a one-time token.

```
POST {AuthBase}/api/auth/passive-token
Content-Type: application/json

{
  "ClientKey": "YOUR_CLIENT_KEY",
  "ClientSecret": "<your ClientSecret>",
  "Email": "customer@example.com",
  "Username": "optional",
  "MobileNumber": "optional",
  "RedirectUri": "optional — must be on your Domain or PublicUrl"
}
```

Response `200` :

```
{
  "success": true,
  "token": "…",
  "expiresAt": "2026-11-17T10:32:00Z",
  "loginPath": "/Sso/PassiveAuth?token=…",
  "loginUrl": "https://account.jalpi.com/Sso/PassiveAuth?token=…"
}
```

Errors:

- `400`: Email or RedirectUri missing or invalid, or the return address is not allowed
- `401`: wrong or inactive ClientKey, or a wrong or missing ClientSecret

Step 2. Redirect the browser to `loginUrl` straight away.

Jalpi signs the user in. If the user is new to Jalpi, Jalpi creates them. Jalpi then returns the browser to your callback with `?ticket=...`. Continue with **B, step 4**.

Token rules

- The token works **once** and expires after **2 minutes**. Request a new one for every login.
- Make the call from your **server** only, and do not log the token.
- In this flow Email, Username, Mobile and RedirectUri come from **your server call**. Anything in the URL is ignored.

Legacy link (existing products only, being phased out)

```
GET {AuthBase}/Sso/PassiveAuth?
Email=...&ClientKey=...&Username=...&Mobile=...&RedirectUri=...
```

This link signs a user in from an unsigned email address. Anyone who knows an email and your public ClientKey can use it. So:

- **New vendor products cannot use it.** It is switched off for them.
- Existing products may keep it for now, but must move to the token flow (section 12).
- Once Jalpi switches your product to **Token** in Admin → Websites, the legacy link shows an error for your product.

4. register-global: create or sync a user

Call this **after** your local user exists, to tell Jalpi your user id. It creates the Jalpi user if needed and makes the link **Active**. It is safe to repeat.

```
POST {AuthBase}/api/auth/register-global
Content-Type: application/json

{
  "ClientKey": "YOUR_CLIENT_KEY",
  "ClientSecret": "<your ClientSecret>",
  "Email": "customer@example.com",
  "Username": "customer",
  "MobileNumber": "919876543210",
  "ExternalUserId": "acme-user-42",
  "ReferralCode": "ABC12XYZ"
}
```

Field		Notes
`ClientKey`	Required	
`ClientSecret`	**Required**	See section 11
`Email`	Required	Lower-cased by Jalpi
`ExternalUserId`	Required in practice	Your stable id. Without it the link stays Pending.
`Username`	Optional	Spelled `Username`. Defaults to the part of the email before `@`.
`MobileNumber`	Optional	Spelled `MobileNumber`. `+` and spaces are removed.
`ReferralCode`	Optional	Only if section 9 applies

Response `200`:

```
{ "success": true, "userId": "8f6c1e2a-...", "wasNewUser": false,
"wasNewLink": true,
  "referralCodeApplied": null, "referralOutcome": "None" }
```

Errors:

- `401`: wrong ClientKey or ClientSecret
- `403`: access revoked
- `409`: this Jalpi user is linked to a different `ExternalUserId`

5. Linking existing users: summary

Situation	What to do
A customer opens you from Jalpi for the first time	Flow **C** on your `LinkPath`
A customer signs up in your app	Create the local user, then call `register-global`
A customer who already has an account in your app clicks "Connect Jalpi"	Flow B with `&Prompt=link` → `validate-ticket` → `confirm-link`
A ticket arrives with `externalUserId` already filled in	The customer is already linked. Just sign them in.

6. Invoices and payment

When the customer chooses a plan, **your server** asks Jalpi to create the invoice. Jalpi applies the customer's billing profile and GST, and returns a payment link.

```
POST {AuthBase}/api/billing/external-invoice-request
Content-Type: application/json

{
  "ClientKey": "YOUR_CLIENT_KEY",
  "ClientSecret": "<your ClientSecret>",
  "ExternalUserId": "acme-user-42",
  "SourceRequestId": "ACME-ORDER-100231",
  "CustomerName": "Riya Shah",
  "CustomerEmail": "customer@example.com",
  "Currency": "INR",
  "QuickGenerate": true,
  "Validity": "Monthly",
  "Items": [
    { "ProductName": "Acme Pro — Monthly", "ProductPrice": 999.00,
"Quantity": 1,
      "Description": "Plan for Nov 2026" }
  ]
}
```

Field		Notes
`ClientKey`, `ClientSecret`	Required	
`ExternalUserId`	Required	Must already be linked (sections 3–5), or Jalpi returns `404`
`SourceRequestId`	**Required**	Your unique order id. Sending the same id again returns the first invoice with `"isDuplicate": true`, so use it for safe retries.
`Items[]`	Required	`ProductName`, `ProductPrice` (**per unit, before GST**), `Quantity`, `Description`
`QuickGenerate`	Optional	`true` means the customer pays now (prepaid). `false` means Jalpi emails the invoice and the customer pays later (postpaid).
`Validity`	Optional	`One-Time`, `Daily`, `7 days`, `Weekly`, `Monthly`, `Quarterly`, `Yearly`. Postpaid defaults to `One-Time`.
`Currency`	Optional	Defaults to `INR`
`CustomerName`, `CustomerEmail`, `CustomerPhone`, `Notes`, `Terms`	Optional	Shown on the invoice

Send prices before GST. Jalpi adds GST and calculates the totals. Any total you send is ignored.

Response `200` (shortened):

```
{
  "success": true,
  "requestStatus": "AwaitingPayment",
  "subtotal": 999.00, "totalTax": 179.82, "totalAmount": 1178.82,
  "payment": {
    "clientKey": "YOUR_CLIENT_KEY",
    "externalPaymentId": "generated-by-jalpi",
    "paymentUrl": "/payment/YOUR_CLIENT_KEY/generated-by-jalpi"
  }
}
```

- Save `payment.externalPaymentId` against your order.
- Redirect the customer to `{AuthBase}{payment.paymentUrl}`.
- `requestStatus: "BillingPending"` means the customer has not completed a billing profile yet. Jalpi asks them for it on the payment page.
- **Never switch on access when the customer returns from the payment page.** Wait for the webhook.

7. Payment webhook

Jalpi sends a `POST` to a **fixed path on your registered server domain**:

```
POST {Domain}/api/payment-responce
```

The path is spelled **`payment-responce`**. Use exactly this spelling.

Headers

Header	Value
`X-JALPI-CLIENTKEY`	your ClientKey
`X-JALPI-TIMESTAMP`	Unix seconds, used in the signature
`X-JALPI-SIGNATURE`	lower-case hex HMAC-SHA256 of `"{timestamp}.{raw body}"`, keyed with your ClientSecret
`X-JALPI-EVENT`	the event name

Events

Event	When	What you do
`payment.success`	The customer paid	Switch on the plan for `payment.externalUserId`
`payment.cancelled`	An unpaid invoice was cancelled	Mark the order cancelled. Do not switch anything on.
`recurring.payment.success`	An automatic renewal was charged (only products that use Jalpi recurring billing)	Extend the plan
`recurring.payment.failed`	An automatic renewal failed	Follow your grace or suspend rule

Ignore event names you do not know. Jalpi will add more, such as a refund event.

The body

The body is an encrypted packet:

```
{ "Salt": "...", "Nonce": "...", "Tag": "...", "Data": "..." }
```

Verify, then decrypt

1. Read the **raw** body as a string. Do not parse it and re-serialise it before checking the signature.
2. Compute HMAC-SHA256 of `timestamp + "." + rawBody`, keyed with the UTF-8 bytes of your ClientSecret, as lower-case hex. Compare it with `X-JALPI-SIGNATURE` in **constant time**. If it does not match, return `401`.
3. Reject the request if the timestamp is more than **5 minutes** from your server clock.
4. Derive the key: PBKDF2-SHA256, password = ClientSecret, salt = `Salt`, **100,000** iterations, 32 bytes.
5. Decrypt `Data` with AES-256-GCM using `Nonce` (12 bytes) and `Tag` (16 bytes). The result is the JSON event.
6. Return `200` within **20 seconds**. Do slow work after you respond.

Node.js (Express)

```
const crypto = require('crypto');
app.post('/api/payment-responce', express.text({ type: '*/*' }), (req, res)
=> {
  const secret = process.env.JALPI_CLIENT_SECRET;
  const ts = req.get('X-JALPI-TIMESTAMP') || '';
  const sig = req.get('X-JALPI-SIGNATURE') || '';
  const body = req.body; // raw string
  const expected = crypto.createHmac('sha256',
secret).update(`${ts}.${body}`).digest('hex');
  const valid = sig.length === expected.length &&
    crypto.timingSafeEqual(Buffer.from(sig), Buffer.from(expected));
  if (!valid || Math.abs(Date.now() / 1000 - Number(ts)) > 300) return
res.sendStatus(401);
  const p = JSON.parse(body);
  const key = crypto.pbkdf2Sync(secret, Buffer.from(p.Salt, 'base64'),
100000, 32, 'sha256');
  const d = crypto.createDecipheriv('aes-256-gcm', key, Buffer.from(p.Nonce,
'base64'));
  d.setAuthTag(Buffer.from(p.Tag, 'base64'));
  const json = Buffer.concat([d.update(Buffer.from(p.Data, 'base64')),
d.final()]).toString('utf8');
  const event = JSON.parse(json);
  // process once per event.payment.externalPaymentId
  res.status(200).json({ received: true });
});
```

C# (ASP.NET Core, .NET 8+)

```
app.MapPost("/api/payment-responce", async (HttpRequest req, IConfiguration
cfg) =>
{
    var secret = cfg["Jalpi:ClientSecret"]!;
    var ts  = req.Headers["X-JALPI-TIMESTAMP"].ToString();
    var sig = req.Headers["X-JALPI-SIGNATURE"].ToString();
    using var reader = new StreamReader(req.Body, Encoding.UTF8);
    var body = await reader.ReadToEndAsync(); // raw string

    using var hmac = new HMACSHA256(Encoding.UTF8.GetBytes(secret));
    var expected =
Convert.ToHexString(hmac.ComputeHash(Encoding.UTF8.GetBytes($"{ts}.
{body}"))).ToLowerInvariant();
    if
(!CryptographicOperations.FixedTimeEquals(Encoding.UTF8.GetBytes(expected),
```

```
Encoding.UTF8.GetBytes(sig))
        || !long.TryParse(ts, out var t)
        || Math.Abs(DateTimeOffset.UtcNow.ToUnixTimeSeconds() - t) > 300)
        return Results.Unauthorized();

    var p = JsonSerializer.Deserialize<JsonElement>(body);
    byte[] B(string name) =>
Convert.FromBase64String(p.GetProperty(name).GetString()!);
    var key = Rfc2898DeriveBytes.Pbkdf2(secret, B("Salt"), 100_000,
HashAlgorithmName.SHA256, 32);
    var data = B("Data");
    var plain = new byte[data.Length];
    using (var aes = new AesGcm(key, 16)) aes.Decrypt(B("Nonce"), data,
B("Tag"), plain);
    using var evt = JsonDocument.Parse(plain);
    // process once per payment.externalPaymentId
    return Results.Ok(new { received = true });
});
```

PHP

```
$secret = getenv('JALPI_CLIENT_SECRET');
$ts  = $_SERVER['HTTP_X_JALPI_TIMESTAMP'] ?? '';
$sig = $_SERVER['HTTP_X_JALPI_SIGNATURE'] ?? '';
$body = file_get_contents('php://input'); // raw string
$expected = hash_hmac('sha256', $ts . '.' . $body, $secret);
if (!ctype_digit($ts) || !hash_equals($expected, $sig) || abs(time() -
(int)$ts) > 300) {
    http_response_code(401); exit;
}
$p = json_decode($body, true);
$key = hash_pbkdf2('sha256', $secret, base64_decode($p['Salt']), 100000, 32,
true);
$plain = openssl_decrypt(base64_decode($p['Data']), 'aes-256-gcm', $key,
    OPENSSL_RAW_DATA, base64_decode($p['Nonce']), base64_decode($p['Tag']));
if ($plain === false) { http_response_code(400); exit; }
$event = json_decode($plain, true);
// process once per $event['payment']['externalPaymentId']
http_response_code(200);
echo json_encode(['received' => true]);
```

Python (Flask)

```
import base64, hashlib, hmac, json, os, time
from cryptography.hazmat.primitives.ciphers.aead import AESGCM

@app.post("/api/payment-responce")
def jalpi_webhook():
    secret = os.environ["JALPI_CLIENT_SECRET"]
    ts = request.headers.get("X-JALPI-TIMESTAMP", "")
    sig = request.headers.get("X-JALPI-SIGNATURE", "")
    body = request.get_data(as_text=True)  # raw string
    expected = hmac.new(secret.encode(), f"{ts}.{body}".encode(),
hashlib.sha256).hexdigest()
    if not ts.isdigit() or not hmac.compare_digest(expected, sig) or
abs(time.time() - int(ts)) > 300:
        return "", 401
    p = json.loads(body)
    b = lambda k: base64.b64decode(p[k])
    key = hashlib.pbkdf2_hmac("sha256", secret.encode(), b("Salt"), 100000,
32)
    event = json.loads(AESGCM(key).decrypt(b("Nonce"), b("Data") + b("Tag"),
None))
    # process once per event["payment"]["externalPaymentId"]
    return {"received": True}, 200
```

The decrypted event (shortened)

```
{
  "payment":  { "externalPaymentId": "...", "externalUserId": "acme-user-42",
"clientKey": "...",
                "amount": 999.00, "currency": "INR", "status": "Success",
                "paymentGateway": "Razorpay", "transactionId": "...",
"planName": "...", "quantity": 1 },
  "invoice":  { "externalInvoiceNumber": "INV-83", "invoiceStatus": "Paid" },
  "selections": [ ... ],
  "gatewayResponse": { ... },
  "settlement": null,
  "meta": { "source": "account.jalpi.com", "eventName": "payment.success",
"processedAt": "..." }
}
```

- Use `payment.externalUserId` to find your user, and `payment.externalPaymentId` to find your order.

- **Handle repeats.** The same event can arrive more than once. Save each processed `externalPaymentId` and return `200` without doing anything on a repeat.
- A customer can pay for several products in one checkout. You receive only **your** part.
- Ignore fields you do not recognise.
- If you think you missed a webhook, contact Jalpi and it can be resent.

8. Support link

Add a **Help** or **Support** link in your app:

```
{AuthBase}/SupportTicket?productId={your support product id}
```

The ticket form opens with your product already selected. An unknown id falls back to "General". When a ticket needs your team, Jalpi passes it to you.

9. Referral (`?ref=`), only if Jalpi enables it for your product

Partner links look like `{PublicUrl}{ReferralPath}?ref=PARTNER_CODE`. On that landing page:

1. Read `?ref=`, then validate it: ASCII letters and digits, `_` and `-`, up to about 50 characters.
2. Store it in an HttpOnly cookie named `ref`, with `Path=/`, `Secure`, `SameSite=Lax` and about 7 days' lifetime. Delete the cookie when the value is missing or invalid.
3. Send it as `ReferralCode` in `register-global`.

Jalpi does the attribution and partner crediting. If Jalpi has not given you a referral path, skip this section.

10. Jalpi-owned products only: shared-cookie features

These work **only for Jalpi's own products on `*.jalpi.com`**, because they rely on Jalpi's `.jalpi.com` login cookies (`jalpi_at`, `jalpi_rt`). **Vendor products cannot use them.**

Endpoint	Purpose
`GET/POST {AuthBase}/api/auth/validate?clientKey=` (browser sends the cookie)	Returns `{ valid, userId, email, externalUserId, ... }` for the signed-in user
`POST {AuthBase}/api/auth/refresh` `{ "ClientKey": "..." }`	Renews the access cookie from the refresh cookie
`POST {AuthBase}/api/auth/logout/single` `{ "ClientKey": "..." }`	Signs the user out of one product
`POST {AuthBase}/api/auth/logout/all`	Signs the user out everywhere and clears the cookies

New code should prefer the ticket flows in section 3, which work on any domain.

11. Security rules (mandatory)

#	Rule	Why
R1	Send `ClientSecret` on every server call (`register-global` , `external-invoice-request` , `passive-token`).	The ClientKey is public. The secret proves the call is yours.
R2	Keep `ClientSecret` on the server. Never put it in browser code, a mobile app or logs.	It authorises billing and decrypts webhooks.
R3	Send `RedirectUri` only on your registered `Domain` or `PublicUrl` host, or leave it out.	Tickets must go only to you.
R4	Use the **passive token flow**, not the legacy email link.	The legacy link lets anyone sign in with only an email address.
R5	Vendors: host on your own domain, never `*.jalpi.com` .	The Jalpi login cookie is visible on every `*.jalpi.com` host.
R6	Verify every webhook (signature, time, decrypt) before acting, and process it idempotently.	Forged or replayed payments
R7	Exchange tickets server-side, once. Never log tickets, tokens, secrets or OTPs.	Replay of a login

#	Rule	Why
R8	Send a unique `SourceRequestId` on every invoice request.	Prevents duplicate invoices on retry
R9	Keep `ExternalUserId` stable forever, and never send Jalpi's `userId` in its place.	It is the join key on both sides
R10	Do not read Jalpi's anonymous payment-details endpoint (`GET /api/billing/external-payment-request/{id}`) for decisions; rely on the webhook.	That endpoint will return only status fields (section 12).

How Jalpi enforces these. Each rule starts in **Observe** mode: Jalpi logs calls that break it and lets them through. Jalpi then switches to **Enforce**, product by product, and broken calls are refused. New vendor products are enforced from day one.

12. What existing Jalpi products must change

Nothing breaks today. Every change below is opt-in until Jalpi tells your team a date, and Jalpi watches its logs first, so your team will know if your product is affected.

#	Change	What your code must do	Today	Later
E1	**ClientSecret required** (R1)	Always send `ClientSecret` in `register-global` and `external-invoice-request`	A missing secret is logged and accepted (a **wrong** one is already refused)	A missing secret gets `401 "ClientSecret is required"`
E2	**Passive login token flow** (R4)	Replace the `/Sso/PassiveAuth?Email=...` link with section 3D: server calls `passive-token` , then redirect to `loginUrl`	Legacy link works (product set to **Legacy**)	Jalpi sets your product to **Token**, and the legacy link shows an error

#	Change	What your code must do	Today	Later
E3	**RedirectUri on your own host** (R3)	Make sure every `RedirectUri` you send to `/Sso/Login` or `/Sso/PassiveAuth` is on your registered Domain or PublicUrl host	A foreign host is logged and allowed	A foreign host gets "Invalid redirectUri"
E4	**SourceRequestId on every invoice** (R8)	Send your order id	Optional	Will become required
E5	**Payment-details GET** (R10)	If you call `GET .../external-payment-request/{id}`, use only status and reference fields, or switch to the webhook	Full row returned	Name, email, mobile, userId and the gateway blob are removed
E6	**Ignore unknown webhook events**	Make your `/api/payment-responce` return `200` for event names it does not handle		A refund event is coming

Switching a product to the token flow (E2)

1. Your team deploys the section 3D code.
2. Your team tells Jalpi.
3. Jalpi sets **Passive login = Token** for your product in Admin → Websites.
4. If anything goes wrong, Jalpi sets it back to **Legacy** at once.

13. Commercial rules (vendor products)

Your vendor agreement is the final word. These are the rules your software must support.

Topic	Rule
Who sells	Jalpi sells to the customer and issues the GST invoice in Jalpi's name and GSTIN. Do not issue your own invoice for sales made through Jalpi.
Payment	Customers pay only on Jalpi's payment page. Do not collect money from Jalpi customers directly for these plans.
Prices	You set the price. Send it per unit, **before GST**. Jalpi applies the GST rate set for your product (18% unless agreed otherwise).
Monthly packages	Each period is a new invoice request with a new `SourceRequestId`.
Commission	Jalpi keeps a commission on each payment, calculated on the price before GST, at the rate in your agreement. Jalpi pays the gateway charges.
Refunds	Within 7 days of payment, full refunds only, approved by Jalpi. When Jalpi tells you a refund is approved, switch off access for that payment.
Payouts	First week of each month, for payments whose refund period is over. A late-month payment may move to the next month's payout.
Your invoice to Jalpi	If you are GST-registered, raise an invoice to Jalpi for each payout.
Vendor portal	Your Jalpi admin login shows only your own product's customers, invoices, sales and payouts.

14. Error reference

Status	Meaning	What to do
`400`	A required field is missing or invalid, or the RedirectUri or return address is not allowed	Check the request against the field tables

Status	Meaning	What to do
`401`	Wrong or inactive ClientKey or ClientSecret, a missing ClientSecret once enforced, or an invalid or expired ticket or token	Check credentials. For tickets or tokens, start again.
`403`	The customer's access to your product was revoked	Ask the customer to contact Jalpi Support
`404`	`ExternalUserId` is not linked to a Jalpi user	Link first (sections 3–5), then retry
`409`	The Jalpi user is linked to a different `ExternalUserId`	Use the original id, or contact Jalpi
`5xx`	Temporary problem at Jalpi	Retry with the **same** `SourceRequestId`; no duplicate invoice is created

15. Go-live checklist

- [ ] Domain, PublicUrl, CallbackPath and LinkPath are registered with Jalpi. Vendors are not on `*.jalpi.com`.
- [ ] `ClientSecret` is only on the server and never logged.
- [ ] **Flow A:** Open from My Products lands signed in.
- [ ] **Flow B:** Sign in with Jalpi works. The ticket is split, exchanged once, and mapped to the right user.
- [ ] **Flow C:** Link from My Products calls validate-ticket and then confirm-link, and the product shows Active.
- [ ] **Flow D** (if used): passive-token, then loginUrl, then callback. The legacy link is not used.
- [ ] `register-global` is sent with `ClientSecret` and the real `ExternalUserId`.
- [ ] Every invoice request has `ClientSecret` and a unique `SourceRequestId`. A repeat returns `isDuplicate: true`.
- [ ] Prices are sent before GST, and the totals look right.
- [ ] The webhook is live at `{Domain}/api/payment-responce` (this spelling) over HTTPS.
- [ ] A bad signature gets `401`. A valid request decrypts and returns `200` in under 20 s.
- [ ] The same webhook twice switches access on only once. Unknown events return `200`.

- [ ] Access is switched on only after `payment.success`.
- [ ] (Vendors) your team can switch off access after an approved refund.
- [ ] The support link opens with your product selected.

16. Test plan (run before go-live and after every Jalpi change)

Test	Expected
Open from My Products	Your app opens signed in
Sign in with Jalpi from your login page	Callback gets a ticket and your session starts
Exchange the same ticket twice	The second call returns `401`
RedirectUri on a foreign host	Logged now, error once enforced. Never send one.
Link a new customer from My Products	`LinkPath` gets a ticket, `confirm-link` succeeds, and the product is Active
`confirm-link` without ExternalUserId	Rejected
`passive-token` with a wrong secret	`401`
Passive `loginUrl` opened twice	The second time shows "expired or already used"
`register-global` without ClientSecret	Accepted with a log line now, `401` once enforced
Invoice with a repeated SourceRequestId	Same invoice, `isDuplicate: true`
Webhook with a tampered body	`401`
Same webhook twice	Access switched on once

Change log

Version	Date	Changes
2.0	17 Nov 2026	One guide for vendors and existing products. Added: account-side flows (Open, Link), the secure passive token flow, security rules with Observe/Enforce, the existing-product action list, recurring events, and the Jalpi-owned cookie endpoints.
1.0	14 Nov 2026	Vendor Integration Guide (first version)