PG18

PAYMENT GATEWAY

Merchant Integration Guide

PG18 Payment Gateway — onboarding, authentication, payment processing, and webhook integration for brand & tenant engineering teams.

Prepared for: Brand / tenant engineering teams
Document type: Integration guide (companion to the full API reference)

PG18 Payment Gateway — Merchant Integration Guide

Audience: Brand/tenant engineering teams integrating their platform with the PG18 payment gateway.

Scope: This guide is the onboarding and narrative layer that sits on top of the full API reference. It explains how the gateway is organized, how to get credentials, how authentication and multi-tenancy work, how to process a payment end-to-end, how to consume webhooks reliably, and how to test in sandbox before going live. It intentionally does not repeat the full per-provider field reference — for exact request/response schemas per payment method, use the live API reference linked in Reference.

Orientation — which API is "the API"
Environments
Onboarding — getting access
Authentication
  Required headers on every request
Core concepts
Quickstart — your first payment
Core payment operations reference
  Endpoint pattern
  Common request fields
  Customer fields
  Transaction statuses
Webhooks / notifications
  Signature verification
  Retry behavior
  Reliability recommendations
Other gateway services (KYC, Cashier)
  KYC (identity verification)
  Cashier
Testing with the Simulator
Error handling
Go-live checklist
Reference — full API specification

Orientation — which API is "the API"

PG18 ships several internal services with "payment" in the name. Only one of them is the integration surface for merchants:

Service	Who it's for	Use it?
Payment gateway (core)	Merchants / brands — this is the API described in this guide	**Yes — this is your integration target**
Payment front-api	Internal only — feeds PG18's own backoffice admin UI	No — not a merchant-facing API despite the name
Payment gateway auth	Human login (JWT + 2FA) for PG18 backoffice staff	No — unrelated to API client authentication

If you're evaluating client libraries, sample code, or Postman collections and something in the name says "front-api" or references 2FA/session login, it's not what you want. Your integration talks to the **payment gateway core API** using HTTP Basic Auth (see Authentication).

Environments

Environment	Base URL
Production	`https://api.pgeighteen.com`
Sandbox / test	`https://api-sta.pgeighteen.com`

Sandbox uses the same API surface as production. Point your integration at sandbox during development and QA, and use the built-in Simulator to exercise success/failure/timeout paths without a live PSP connection.

Onboarding — getting access

Each brand/tenant is provisioned as an **Application** in PG18. Onboarding is currently a platform-team-assisted process rather than self-service signup — reach out to your PG18 integration contact to kick it off. At a high level:

1. **Application creation** — the platform team creates your Application (tenant) record. This provisions your tenant's default configuration (available providers, KYC settings, cashier payment methods, locale mapping) as a starting point, which is then tuned to your brand's needs (payment methods, PSPs, currencies, KYC requirements).

2. **API user / credential issuance** — a separate step from Application creation. You'll be issued an **API user** (username + password for Basic Auth) scoped to your Application, plus the `Application` and `Instance` header values you'll send on every request. Treat the password and the webhook signing secret (see Webhooks) as secrets — store them in your own secret manager, not in source control.
3. **Sandbox access first** — you'll typically receive sandbox credentials before production credentials, so you can integrate and test end-to-end (including webhook delivery) before a go-live conversation.
4. **Provider/method enablement** — which payment providers and methods (credit card, bank transfer, e-wallets, etc.) are available to you depends on what's been configured for your Application. Confirm your enabled provider list with your integration contact — the full provider catalog is large (60+ PSPs) but any given Application typically only has a subset enabled.

> **Known gap:** there isn't yet a fully self-service onboarding runbook publishing the exact internal steps/SLA for provisioning. If your integration contact can't answer a provisioning question, escalate through your normal PG18 account channel rather than assuming a specific process — this guide describes the mechanism (Application + API user), not a fixed internal timeline.

Authentication

All API requests use **HTTP Basic Authentication** — base64-encoded `username:password` in the `Authorization` header. There is no OAuth/JWT flow for API clients; JWT + 2FA is used only for human backoffice login and does not apply here.

```
Authorization: Basic dGVzdHVzZXI6cGFzc3dvcmQ=
```

(`testuser:password` → `dGVzdHVzZXI6cGFzc3dvcmQ=`)

Required headers on every request

Header	Required	Description
`Authorization`	Yes	`Basic <base64(username:password)>` — your API user credential
`Application`	Yes	Identifies your tenant. Decides which Application's configuration and data the request operates against. Delivered to you during onboarding.
`Instance`	Yes	Instance identifier, delivered to you during onboarding.
`Content-Type`	Yes	`application/json` (only supported value)
`Accept`	Yes	`application/json` (only supported value)

A request with valid credentials but a missing/incorrect `Application` header will not resolve to your tenant's data — always send both.

Core concepts

- **Application** — your tenant. One Application = one brand's configuration, credentials scope, and data isolation boundary.
- **Customer** — the end player. Identified to PG18 by *your* `merchantReference` (your own internal player ID), not a PG18-generated ID. Send full customer details (name, DOB, address, etc.) on first use; subsequent requests can reference the customer by `merchantReference` alone, and you can update stored customer data by sending a new customer object.
- **Transaction** — a single payment attempt (deposit or withdrawal) against a specific provider and method. Identified by PG18's `id` (UUID) and your own `merchantReference` (informational only on PG18's side — uniqueness is your responsibility, used for your own reconciliation).
- **Provider** — the underlying PSP (e.g. `neteller`, `trustly`, `hexopay`, `skrill`, ...). Each provider has its own `/payments/{provider}/transaction/process` endpoint, since providers differ in the extra fields they require (e.g. `card` for credit-card methods).

Quickstart — your first payment

This walks through a deposit against a representative provider. Field names are consistent across providers; provider-specific additions (like `card` for credit-card methods) are called out.

1. Send the payment request.

```
POST {baseUrl}/payments/{provider}/transaction/process
Authorization: Basic <credentials>
Application: <your-application>
Instance: <your-instance>
Content-Type: application/json
```

```
{
    "currency": "EUR",
    "ip": "127.0.0.1",
    "locale": "pl_PL",
    "method": "credit-card",
    "amount": "5.11",
    "type": "deposit",
    "deviceType": "mobile",
    "notifyUrl": "https://yourdomain.com/webhooks/pg18",
    "merchantReference": "your-txn-reference-123",
    "redirectUrl": "https://yourdomain.com/payment/return",
    "customer": {
        "merchantReference": "your-player-id-456",
        "firstName": "Joe",
        "lastName": "Doe",
        "gender": "M",
        "email": "joe.doe@email.com",
        "address": "Some address",
        "city": "City",
        "state": "State",
        "zip": "Zip code",
        "countryCode": "PL",
        "birthday": "1974-10-10",
        "phone": "+48 555 555 555"
    },
    "card": {
        "number": "4012001037141112",
        "cardHolder": "Card holder",
        "expiryMonth": "01",
        "expiryYear": "2022",
        "cvv": "123",
        "secure3d": true
    }
}
```

2. Read the response — it has two parts: `transaction` and an action.

```
{
    "action": "redirect",
    "url": "https://provider-hosted-page/to/redirect/customer",
    "transaction": {
        "provider": "neteller",
        "method": "neteller",
        "status": "pending",
        "type": "deposit",
        "id": "33163f5e-412f-460e-b6c5-97738573f7f1",
        "merchantReference": "your-txn-reference-123",
        "startTime": "2020-10-05 14:17:44",
        "customer": { "...": "..." }
    }
}
```

3. Act on the returned action:

`action` value	What you do
(empty)	Nothing — no further action needed on your side.
`redirect`	Redirect the customer's browser to `url`.
`form`	Render a form (fields in `parameters`) and POST it to `url`, respecting `autoSubmit` and, if present, `type: iframe` for embedded rendering.

4. Watch for the final status. If the initial response already carries a final status (`completed`, `failed`, etc.) you can act on it immediately, but a webhook notification will still follow shortly after. If the response is non-final (e.g. `pending`), wait for the webhook notification rather than polling.

Core payment operations reference

Endpoint pattern

Method	URL	Description
`POST`	`/payments/{provider}/transaction/process`	Process a new payment (deposit or withdrawal)
`PATCH`	`/payments/{provider}/transactions/{externalId}/{withdrawalDecision}`	Approve, decline, or cancel a pending withdrawal

`withdrawalDecision` is one of: `approve` (by agent), `decline` (by agent), `cancel` (by customer).

Every enabled provider has its own `process`/withdrawal-decision pair under `/payments/{provider}/...` (e.g. `/payments/trustly/transaction/process`, `/payments/skrill/transaction/process`). The request/response shape is consistent across providers; some providers require an extra method-specific object (credit card requires `card`; card+3DS flows also accept an optional `browserInfo` object — see the full reference for the complete per-provider field list and the current enabled-provider catalog).

Common request fields

Field	Required	Notes
`amount`	Yes	Numeric, precision 16 / scale 8 (max `99999999.99999999`), validated per ISO 4217 for the given currency.
`currency`	Yes	ISO 4217 currency code.
`type`	Yes	`deposit` or `withdrawal`.
`method`	Yes	Must match a method enabled for the provider — an incorrect/empty method rejects the transaction.
`deviceType`	Yes	`mobile`, `tablet`, or `desktop`.
`ip`	Yes	IPv4 or IPv6.
`locale`	Yes	ICU locale ID (e.g. `fr_FR`). Not the transaction's country — that comes from `customer.countryCode`.
`notifyUrl`	Optional	HTTPS webhook URL for this transaction's notifications (RFC 1738).
`merchantReference`	Yes	Your own reference. Not validated for uniqueness on PG18's side — keeping it unique is your responsibility; used for your reconciliation. Max 255 chars.
`redirectUrl`	Yes	Where the customer is sent back to after the payment flow.
`customer`	Yes	Nested customer object — see below.

Customer fields

Full customer detail (name, address, DOB, etc.) is required the first time you reference a given `customer.merchantReference`; afterward you can pass just the `merchantReference` to reuse the stored customer, or send an updated object to overwrite stored data.

Field	Required	Notes
`merchantReference`	Yes	Your player ID. Unique per Application — this is the key PG18 uses to update/reuse customer records.
`firstName` , `lastName`	New customer only	
`gender`	New customer only	`M` , `F` , or `N` (non-binary)
`email`	New customer only	
`address` , `city` , `state` , `zip`	New customer only	
`countryCode`	New customer only	ISO 3166-1 alpha-2. This is the transaction's country of record — not `locale` .
`birthday`	New customer only	Must not be a future date.
`phone`	Optional	Max 128 chars.
`additionalAttributes`	Optional	Custom properties, max 5.

Transaction statuses

Status	Meaning
`started`	Initial status of every transaction.
`pending`	Processing.
`completed`	Finished successfully (deposit or withdrawal).
`failed`	Finished with an error — see `errorMessage` in the response.
`amount_mismatch`	The amount PG18 received differs from the amount confirmed on the provider side.
`pending_approval`	Withdrawal awaiting an approve/decline/cancel decision.
`process_approve` / `process_decline` / `process_cancel`	Withdrawal being processed after that decision.
`declined`	Withdrawal declined (typically by the merchant/agent).
`canceled`	Withdrawal canceled (typically by the customer).
`chargeback`	Transaction charged back.
`reversal`	Transaction reversed.

Webhooks / notifications

PG18 notifies you asynchronously whenever a transaction's status changes, by POSTing the transaction payload to the `notifyUrl` you supplied on the original request.

Signature verification

Every notification carries a `Signature` header — verify it before trusting the payload:

```
signature = base64_encode( HMAC-SHA3-512(raw request body, your API user's secret key) )
```

```
$signature = base64_encode(
    hash_hmac('sha3-512', $requestPayload, $apiUserSecretKey)
);
```

Compute the HMAC over the **raw** request body (before any JSON re-serialization on your side — re-encoding can reorder keys or change whitespace and produce a different signature) using your API user's secret key, and compare it to the `Signature` header.

Retry behavior

Respond with HTTP **200** to acknowledge a notification. Any other status is treated as a failure, and the notification is re-queued for retry.

> **Correction vs. the published API reference:** the live API documentation currently states retries follow "a fibonacci sequence up to a maximum of 20 attempts." That description is out of date. The actual delivery worker uses **exponential backoff** (base delay, doubling multiplier, configurable max-attempt cap) — not a fibonacci schedule. If you're building retry assumptions into your own alerting/reconciliation (e.g. "how long before I should assume a notification is truly lost"), don't rely on the fibonacci description; confirm the current base delay and max-attempt values with your integration contact, since they're environment-configurable rather than fixed constants.

Reliability recommendations

- **Respond fast, process async.** Verify the signature, persist/queue the payload, and return `200` immediately — don't do slow downstream work synchronously in the webhook handler, or PG18 may treat a slow response as a failure and retry.
- **Be idempotent.** You may receive the same notification more than once (including intentionally, when testing the Simulator's `duplicate-notification` scenario — see Testing). Dedupe on the transaction `id`. `merchantReference` alone is not a reliable idempotency key for this purpose since it's not required to be unique.
- **Don't rely solely on webhooks for final state.** If a transaction's initial synchronous response already carries a final status, treat that as authoritative; the webhook is confirmation/async-completion, not the only source of truth.
- **Use HTTPS only.** `notifyUrl` must be HTTPS.

Other gateway services (KYC, Cashier)

If your Application uses PG18's identity-verification or hosted-cashier capabilities, they're separate sub-APIs on the same gateway, using the same Basic Auth + `Application` / `Instance` header model.

KYC (identity verification)

1. You send a document request (customer details + requested document types).
2. PG18 returns a document entity and an `uploadUrl`.
3. The customer uploads the document at that URL.
4. PG18 sends a notification when the document's status changes.

Document statuses: `requested` → `pending` (uploaded, awaiting review) → `approved` / `declined`; `cancelled` is available while still in `requested` state.

Cashier

The hosted Cashier widget sends server-to-server callbacks to a URL you configure in the Cashier application settings, signed the same way as payment webhooks (`Signature` header, HMAC-SHA3-512 of the payload with your API user's secret — note the Cashier variant HMACs to a hex string before base64-encoding: `base64_encode(HMAC_HEX('sha3-512', payload, secret))`). See the full reference for the callback payload shape and the Cashier-specific error code table (`GENERAL_ERROR` , `CALLBACK_INVALID_SIGNATURE` , `INVALID_DATA` , etc.).

Testing with the Simulator

Sandbox includes a built-in Simulator that lets you exercise notification/webhook handling without a live PSP connection. Add a `scenario` query parameter to your `notifyUrl` :

```
https://yourdomain.com/webhooks/pg18?scenario=failed
```

Scenario	Effect
`completed`	Simulates a completed transaction.
`failed`	Simulates a failed transaction.
`amount-mismatch`	Simulates the provider confirming a different amount than requested.
`duplicate-notification`	Sends the same notification twice in a row — use this to verify your idempotency handling.
`timeout`	Simulates a request timeout from PG18's side. Accepts an additional `timeout` query param (seconds, default 60).

Use this during development to exercise every branch of your webhook handler (including your signature verification and retry/idempotency logic) before connecting to a real PSP in sandbox.

Error handling

- A failed transaction carries `status: "failed"` plus an `errorMessage` field describing the failure.
- Cashier callbacks use a dedicated numeric error-code table (`4xxx` for request/validation errors, `5xxx` for processing/system errors) — see the full reference for the complete list.
- Validate your own request payloads client-side against the field tables above before sending — most `4xxx` -class errors are preventable input validation issues (missing required fields, wrong method for the provider, invalid country/locale codes).

Go-live checklist

- [ ] Sandbox credentials received and stored in your secret manager (not source control).
- [ ] `notifyUrl` implemented: verifies `Signature` (HMAC-SHA3-512, raw body), returns `200` promptly, is idempotent on transaction `id`.
- [ ] All target providers/methods tested against the Simulator's `completed`, `failed`, `amount-mismatch`, `duplicate-notification`, and `timeout` scenarios.
- [ ] Redirect/form/iframe handling implemented for every `action` type your enabled methods can return.
- [ ] Withdrawal decision flow implemented, if you support withdrawals (`approve` / `decline` / `cancel`).
- [ ] Reconciliation process in place using `merchantReference` and PG18's transaction `id`.
- [ ] Production credentials requested from your integration contact, with a confirmed cutover/testing plan for the live PSP connections.

Reference — full API specification

This guide covers the mechanics and workflow; the authoritative, field-by-field API specification (every provider's request/response schema, full error code tables, current enabled-provider catalog) is maintained separately and kept current as providers are added.

The staging API reference is hosted at `docs.pgeighteen.com`. Confirm the current production reference URL with your integration contact before relying on it in client code or documentation links — it isn't reproduced in full here since it changes independently of this guide.