

Documentation d'intégration

API marchand v1

Gwongzho Pay

Encaisser (*pay-in*) et décaisser (*pay-out*) par mobile money
avec une seule API, des requêtes signées et des webhooks signés.

Version 2.0 – Octobre 2026

Rédigée par **NOUBISSIE KAMGA WILFRIED**

À propos de ce document

Ce document s'adresse aux **développeurs** qui intègrent l'API Gwongzho Pay dans un site marchand, une application mobile ou un système de gestion. Il va à l'essentiel : de la première requête en sandbox à la mise en production.

Ce que vous trouverez. Le fonctionnement général (chapitre 1), un premier paiement en quelques minutes (chapitre 2), la signature des requêtes (chapitre 3), les opérations de paiement et la consultation du compte (chapitre 4), les webhooks (chapitre 5), les erreurs (chapitre 6) et la mise en production (chapitre 7).

Scripts d'exemple. Des clients prêts à l'emploi en **Python, JavaScript (Node.js), Go, Java, PHP et curl** (shell) sont téléchargeables avec ce document, dans l'onglet *Documentation* de la page *Developers* de votre tableau de bord. Ils font tous la même chose : lister vos méthodes, consulter vos soldes, encaisser, verser, suivre une transaction et vérifier un webhook.

Conventions. Le texte `a_comme_ceci` désigne un nom de champ, d'en-tête ou une valeur littérale. Les encadrés bleus donnent des conseils, les jaunes signalent un piège fréquent, les rouges un point de sécurité.

Table des matières

À propos de ce document **1**

1 Vue d'ensemble **4**
- 1.1 Principe . . . 4
- 1.2 Environnements . . . 4
- 1.3 Vocabulaire . . . 5

2 Premier paiement en cinq minutes **6**
- 2.1 Ce qu'il vous faut . . . 6
- 2.2 Les scripts d'exemple . . . 6
- 2.3 Pas à pas . . . 7
- 2.4 Et ensuite ? . . . 8

3 Authentification et signature **9**
- 3.1 Les trois en-têtes . . . 9
- 3.2 La chaîne à signer . . . 9
- 3.3 Exemple . . . 10
- 3.4 Vecteurs de test . . . 10
- 3.5 Essayer avec Postman ou Insomnia . . . 10
- 3.6 Une signature ne sert qu'une fois . . . 11
- 3.7 Clé inconnue, révoquée ou d'un autre environnement . . . 11
- 3.8 Horloge et adresses IP . . . 11
- 3.9 Dépannage . . . 11

4 Paiements et compte **13**
- 4.1 Conventions . . . 13
- 4.2 Vos méthodes : `GET /v1/methods` . . . 14
- 4.3 Encaisser : `POST /v1/payins` . . . 15
- 4.4 Verser : `POST /v1/payouts` . . . 17
- 4.5 Statuts et cycle de vie . . . 17
- 4.6 Consulter des transactions . . . 18
- 4.7 Votre solde : `GET /v1/balance` . . . 18
- 4.8 Les frais . . . 19

5 Webhooks **20**
- 5.1 Quand et vers où . . . 20
- 5.2 Ce que vous recevez . . . 20
- 5.3 Vérifier la signature . . . 22
- 5.4 Fiabilité de la livraison . . . 23

6 Erreurs **24**
6.1 Codes 24
6.2 Que réessayer ? 25
6.3 Erreur d'API ou échec de transaction ? 26

7 Sandbox et mise en production **27**
7.1 Tester en sandbox 27
7.2 Liste de contrôle avant la production 28
7.3 Passer en production 28
7.4 Demander de l'aide 28

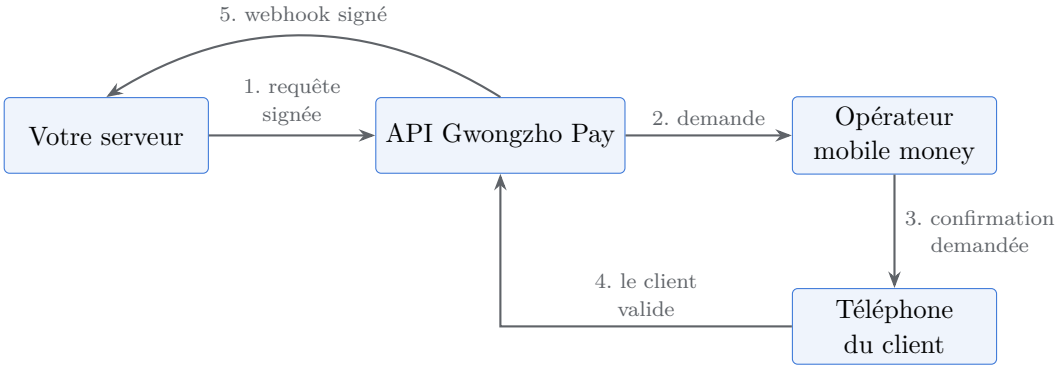
Chapitre 1

Vue d'ensemble

1.1 Principe

En une phrase

Vous envoyez une requête HTTPS signée, vous obtenez tout de suite une transaction `INITIATED`, puis vous recevez un webhook quand elle passe à `SUCCESS`, `FAILED` ou `EXPIRED`.



1.2 Environnements

Environnement	Adresse de base
Sandbox (essais)	`https://api-sandbox-gateway.gwongzhopay.com`
Production	`https://api-gateway.gwongzhopay.com`

	Sandbox	Production
Clé (`X-Api-Key`)	`gk_test_...`	`gk_live_...`
Argent réel	Non	Oui
Résultat des paiements	Piloté par le dernier chiffre du téléphone (section 7.1)	Décidé par le client et l'opérateur

Chaque environnement a ses propres clés, ses propres données et son propre webhook. Une clé de sandbox n'est **jamais** acceptée en production, et inversement : la requête est alors refusée comme une signature invalide.

Vos clés et votre URL de webhook se gèrent dans le tableau de bord, page *Developers*.

1.3 Vocabulaire

Terme	Signification
Pay-in	Encaissement : le client paie et valide sur son téléphone. Le montant, moins les frais, est crédité sur votre solde.
Pay-out	Décaissement : vous versez de l'argent sur le numéro mobile money d'un client. Le montant *plus les frais* est débité de votre solde.
Sens (`direction`)	L'opération que permet une méthode : soit un pay-in (`PAYIN`), soit un pay-out (`PAYOUT`). Une méthode est ouverte pour vous *par sens*, avec ses propres frais et limites.
Transaction	Un pay-in ou un pay-out. Identifiée par sa `transaction_ref` (chez nous) et par votre `order_id` (chez vous).
`order_id`	Votre référence de commande. Elle sert d'**identifiant unique** : renvoyer la même requête ne crée jamais de doublon.
Clé API	Un identifiant public (`X-Api-Key`) associé à un secret qui ne circule jamais.
Méthode	Un moyen de paiement ouvert pour vous, défini par un pays (`country`), un type (`payment_type`), un nom (`method`) et une devise (`currency`). On la découvre avec `GET /v1/methods`.
Solde disponible	Ce que vous pouvez dépenser en pay-outs.
Solde réservé	Ce qui est retenu par des pay-outs en cours.
Webhook	Un appel HTTPS que nous faisons vers votre serveur quand une transaction change d'état.

Chapitre 2

Premier paiement en cinq minutes

Ce chapitre vous fait faire un pay-in, puis un pay-out, en sandbox, avec les scripts d'exemple. Aucun argent réel n'est en jeu.

2.1 Ce qu'il vous faut

1. Une **clé API de sandbox** et son secret. Dans le tableau de bord, page *Developers*, onglet des clés : créez une clé. Le **secret n'est montré qu'une seule fois** : copiez-le tout de suite.
2. Un des scripts d'exemple (onglet *Documentation* de la même page).
3. Les trois variables d'environnement suivantes :

```
export GWONGZHO_API_KEY="gk_test_..."       # l'identifiant de votre clé
export GWONGZHO_API_SECRET="gsk_test_..."   # son secret
# GWONGZHO_API_URL vaut par défaut l'adresse de la sandbox
```

Exemple 2.1 – Configuration (jamais dans le code source)

2.2 Les scripts d'exemple

Les six scripts ont les mêmes commandes et les mêmes options. Ils ne demandent aucune bibliothèque externe.

Langage	Fichier	Lancer
Python 3.8+	`gwongzho.py`	`python3 gwongzho.py methods`
JavaScript (Node.js 18+)	`gwongzho.mjs`	`node gwongzho.mjs methods`
Go 1.20+	`gwongzho.go`	`go run gwongzho.go methods`
Java 17+	`Gwongzho.java`	`java Gwongzho.java methods`
PHP 8+ (curl, json)	`gwongzho.php`	`php gwongzho.php methods`
Shell (curl, openssl)	`gwongzho.sh`	`./gwongzho.sh methods`

Commande	Effet
`methods`	Vos méthodes ouvertes, avec frais et limites par sens.

Commande	Effet
`balance`	Vos soldes, un par devise (disponible et réservé).
`payin --amount M --phone N`	Encaisse le client dont le numéro est `N`.
`payout --amount M --phone N`	Verse `M` au numéro `N`.
`status REF` ou `status --order-id ID`	L'état d'une transaction.
`transactions --status S --limit N`	Vos transactions, avec filtres.
`verify-webhook`	Vérifie la signature d'un webhook reçu (chapitre 5).

Pour `payin` et `payout`, l'option `--wait` suit la transaction jusqu'à son statut final. Sans `--country`, `--payment-type`, `--method` ni `--currency`, le script prend la première méthode ouverte pour vous dans ce sens. Un `--order-id` fixe votre référence ; sans lui, le script en génère une.

2.3 Pas à pas

1. Voir vos méthodes. Lancez :

```
python3 gwongzho.py methods
```

La réponse liste chaque méthode ouverte pour vous, avec ses frais et ses limites pour le pay-in et le pay-out. Si elle est vide, aucune méthode n'est encore ouverte sur votre compte : écrivez à support@gwongzho.com.

2. Encaisser un client. En sandbox, le **dernier chiffre** du numéro décide du résultat (section 7.1) : `0` donne un succès.

```
python3 gwongzho.py payin --amount 1000 --phone 23597326770 --wait
```

```
HTTP 202
{ "transaction_ref": "AD11CF83F0D31E2482518BF2E336E791",
  "order_id": "TEST-1790986876248-e9da", "type": "PAYIN",
  "amount": 1000, "fee": 15, "net_amount": 985,
  "status": "INITIATED", ... }
  status: INITIATED
  status: PENDING
  status: SUCCESS
```

Exemple 2.2 – Réponse immédiate (HTTP 202) puis suivi

Le montant crédité à votre solde est `net_amount` : 1 000 moins 15 de frais.

3. Vérifier le solde. Un solde par devise, avec le disponible et le réservé :

```
python3 gwongzho.py balance
```

4. Verser de l'argent. Un pay-out est refusé si votre solde disponible ne couvre pas le montant *plus* les frais. Après le pay-in ci-dessus, vous avez de quoi en faire un :

```
python3 gwongzho.py payout --amount 500 --phone 23597326770 --wait
```

5. Retrouver une transaction. Par sa référence, par votre `order_id`, ou en listant avec des filtres :

```
python3 gwongzho.py status AD11CF83F0D31E2482518BF2E336E791
python3 gwongzho.py status --order-id TEST-1790986876248-e9da
python3 gwongzho.py transactions --status SUCCESS --limit 10
```

Pay-out avec une clé de production

Avec une clé `gk_live_...`, un pay-out déplace de l'**argent réel**. Les scripts exigent alors l'option `--yes`.

2.4 Et ensuite ?

Les scripts sont un point de départ : lisez-les, ils sont courts, puis reprenez leur logique dans votre application. Le plus délicat est la **signature** des requêtes (chapitre 3) ; ensuite viennent la gestion des statuts (chapitre 4) et la réception des webhooks (chapitre 5).

Chapitre 3

Authentification et signature

Chaque requête vers `/v1` est **signée** avec le secret de votre clé API (HMAC-SHA256). Le secret ne circule jamais : nous le connaissons de notre côté, nous recalculons la signature et nous la comparons à la vôtre. Ainsi, une requête ne peut être ni modifiée en chemin, ni rejouée, ni forgée sans le secret.

3.1 Les trois en-têtes

En-tête	Format	Rôle
`X-Api-Key`	`gk_test_...` ou `gk_live_...`	Identifie votre clé (public).
`X-Timestamp`	Secondes Unix (UTC), 10 chiffres	Heure de la requête, tolérance de **5 minutes**.
`X-Signature`	64 caractères hexadécimaux minuscules	Signature de la requête.

Pour les requêtes avec un corps (`POST`), ajoutez `Content-Type: application/json`.

3.2 La chaîne à signer

Quatre lignes séparées par un saut de ligne (`\n`), **sans saut de ligne final** :

```
<X-Timestamp>
<METHODE HTTP en majuscules>
<chemin>?<requete>            (sans "?" s'il n'y a pas de parametres)
<SHA-256 hexadecimal du corps>
```

Exemple 3.1 – Chaîne à signer

$$\texttt{X-Signature} = \text{hex}(\text{HMAC-SHA256}(\texttt{secret}, \texttt{chaîne à signer}))$$

où `secret` est le texte de votre secret (`gsk_test_...`), utilisé tel quel comme clé, et `hex` produit des minuscules.

3.2.1 Les quatre règles qui évitent presque toutes les erreurs

1. **Le chemin et la requête sont signés « tels qu'envoyés »**, dans le même ordre et avec le même encodage que sur la ligne de requête HTTP. Pour une requête `GET` avec `?status=SUCCESS&limit=5`, la troisième ligne est exactement `/v1/transactions?status=SUCCESS&limit=5`. Ne signez pas l'adresse complète (pas de nom d'hôte).

2. **Le corps est signé octet pour octet.** Calculez le hachage sur la chaîne exacte que vous envoyez. Si votre bibliothèque re-sérialise le JSON, signez *le résultat* de la sérialisation.
3. **Un corps vide se hache aussi.** Pour un `GET`, la quatrième ligne est le SHA-256 de la chaîne vide : `e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`.
4. **Pas de saut de ligne à la fin**, et des `\n` (pas de `\r\n`).

3.3 Exemple

```python
import hashlib, hmac, json, os, time, urllib.request

BASE = os.environ.get("GWONGZHO_API_URL", "https://api-sandbox-gateway.gwongzhopay.com")
KEY, SECRET = os.environ["GWONGZHO_API_KEY"], os.environ["GWONGZHO_API_SECRET"]

def call(method, uri, payload=None):
    body = json.dumps(payload, separators=(",", ":")).encode() if payload else b""  # UNE fois
    ts = str(int(time.time()))
    text = "\n".join([ts, method, uri, hashlib.sha256(body).hexdigest()])
    signature = hmac.new(SECRET.encode(), text.encode(), hashlib.sha256).hexdigest()
    req = urllib.request.Request(BASE + uri, data=body or None, method=method, headers={
        "X-Api-Key": KEY, "X-Timestamp": ts, "X-Signature": signature,
        "Content-Type": "application/json"})
    return urllib.request.urlopen(req, timeout=30)
```

Exemple 3.2 – Signer et envoyer une requête (Python)

La même logique, dans les six langages, est dans les scripts d'exemple (fonction de signature en tête de chaque fichier).

3.4 Vecteurs de test

Vérifiez votre implémentation avec ces valeurs avant de l'essayer sur l'API. Secret : `gsk_test_secret`, horodatage : `1700000000`.

Requête `POST /v1/payins`, corps `{"country":"TD","order_id":"CMD-1"}` (sans espace) :

```
85a7c563e44d8f00088d39258bed39cb28df67529657a52a13f1e45c67e05334
```

Requête `GET /v1/transactions?limit=5&status=SUCCESS`, corps vide :

```
142af07d4113a9905ead359c96b5698e70a376ece15e85468b1b8e3edb1dfe54
```

3.5 Essayer avec Postman ou Insomnia

Une signature est liée à *une requête précise* et ne sert qu'une fois : on ne peut donc pas enregistrer une requête dans un outil graphique et la renvoyer telle quelle, il faut la signer à chaque envoi. Deux collections prêtes à importer le font pour vous avec un script exécuté avant chaque envoi, et couvrent toute l'API : méthodes, soldes, encaisser, verser, consulter, lister, les cas de test de la sandbox et quelques erreurs. Elles sont dans l'archive des exemples (fichiers `Gwongzho.postman\_collection.json`, `Gwongzho.postman\_environment.json` et `Gwongzho.insomnia.json`

importez-les, renseignez `api\_key` et `api\_secret` dans l'environnement, puis envoyez. Le script de signature reproduit les vecteurs de test de la section précédente.

3.6 Une signature ne sert qu'une fois

Chaque signature valide n'est acceptée **qu'une seule fois**. Deux conséquences pratiques :

— **Pour réessayer une requête, signez-la à nouveau** avec un nouvel horodatage. Un client HTTP qui rejoue automatiquement les mêmes en-têtes déclenche `REPLAYED_REQUEST`.

— L'horodatage est en **secondes** : deux requêtes strictement identiques (même méthode, même chemin, même corps) signées dans la même seconde ont la *même* signature, et la seconde est refusée avec `REPLAYED_REQUEST`. Attendez la seconde suivante et signez de nouveau ; les scripts d'exemple le font automatiquement. Cela ne concerne pas vos créations de paiement : un réessai de création garde son `order_id` et ne produit jamais de doublon (section 4.1.2).

3.7 Clé inconnue, révoquée ou d'un autre environnement

Pour ne pas aider un attaquant, une clé inconnue, révoquée ou d'un autre environnement donne *la même réponse* qu'une mauvaise signature : `401 INVALID_SIGNATURE`. Si vous êtes sûr de votre calcul, vérifiez que la clé est bien celle de l'environnement visé et qu'elle n'a pas été révoquée.

3.8 Horloge et adresses IP

— **Horloge.** Un serveur en avance ou en retard de plus de 5 minutes voit toutes ses requêtes refusées (`INVALID_SIGNATURE`). Synchronisez vos serveurs par NTP et utilisez l'heure UTC : un horodatage Unix est toujours en UTC.

— **Adresses IP.** Sur demande, une clé peut être limitée à une liste d'adresses ou de plages (notation CIDR). Une requête correctement signée venant d'une autre adresse reçoit `403 FORBIDDEN`. C'est fortement recommandé pour les clés de production qui font des pay-outs. Déclarez l'adresse *publique de sortie* de votre serveur.

3.9 Dépannage

Symptôme	Cause probable
`INVALID_SIGNATURE` sur toutes les requêtes	Horloge décalée ; mauvaise clé ou clé d'un autre environnement ; secret copié avec un espace ou un retour à la ligne.
Seuls les `POST` échouent	Le corps signé n'est pas exactement le corps envoyé (re-sérialisation JSON, espaces, encodage).
Seuls les `GET` avec paramètres échouent	Le chemin signé n'inclut pas la requête (`?...`), ou la bibliothèque HTTP a réordonné ou ré-encodé les paramètres.
Les `GET` sans paramètres échouent	La quatrième ligne n'est pas le SHA-256 de la chaîne vide.
`REPLAYED_REQUEST`	Même signature envoyée deux fois : un *retry* automatique, ou deux requêtes identiques dans la même seconde.

Symptôme	Cause probable
`UNAUTHORIZED`	Un des trois en-têtes manque, ou l'horodatage n'est pas un nombre.
`403` avec le texte `error code: 1010` (pas du JSON)	Notre protection a refusé l'identifiant par défaut de votre client HTTP : envoyez un `User-Agent` explicite.
`FORBIDDEN`	La clé est restreinte à des adresses IP et votre serveur n'en fait pas partie.

Protégez votre secret

Traitez le secret comme un mot de passe de base de données : variable d'environnement ou gestionnaire de secrets, jamais dans le code source, jamais dans un journal, jamais côté navigateur ou application mobile. En cas de doute (fuite, départ d'un collaborateur), **révoquez la clé** dans le tableau de bord et créez-en une nouvelle : la révocation est immédiate.

Chapitre 4

Paiements et compte

Six routes, toutes sous `/v1` et toutes signées (chapitre 3) :

Méthode	Route	Rôle
GET	`/v1/methods`	Vos méthodes ouvertes, avec frais et limites.
GET	`/v1/balance`	Vos soldes, par devise.
POST	`/v1/payins`	Encaisser un client.
POST	`/v1/payouts`	Verser de l'argent à un client.
GET	`/v1/transactions/{transaction_ref}`	Une transaction, à jour.
GET	`/v1/transactions`	Vos transactions, filtrées et paginées.

4.1 Conventions

— **JSON** en UTF-8, avec `Content-Type: application/json`. Le décodage est **strict** : un champ inconnu, un JSON mal formé ou du contenu après le JSON donnent `400 INVALID_REQUEST`. Taille maximale d'un corps : 1 Mio.

— **Dates** au format ISO 8601 en UTC (`2026-10-02T10:00:00Z`).

— **Identifiants opaques** : ne cherchez pas à les interpréter. Une `transaction_ref` est une chaîne de 32 caractères hexadécimaux majuscules.

— **Téléphone** : au format international, avec l'indicatif du pays, sans espace ni tiret (`23597326770`). Un `+` initial est toléré. 8 à 15 chiffres. Le numéro n'est **jamais** renvoyé dans les réponses ni dans les webhooks : gardez-le de votre côté.

— **User-Agent** : envoyez un `User-Agent` explicite qui nomme votre application (par exemple `MaBoutique/1.0`). L'identifiant par défaut de certaines bibliothèques HTTP est refusé par notre protection (réponse `403` dont le corps est le texte `error code: 1010`, et non du JSON).

— **Réponses** : chacune porte un en-tête `X-Request-Id`. Communiquez-le au support en cas de problème.

— **Débit** : 100 requêtes par seconde et par clé par défaut (adaptable à votre volume). Au-delà : `429 RATE_LIMITED` avec l'en-tête `Retry-After`.

— **Délai d'attente** recommandé sur vos appels : 30 secondes.

4.1.1 Montants

Les montants sont des **nombres décimaux** dans la précision de la devise : XAF et XOF n'ont pas de décimales (`10000` est valable, `10000.5` est refusé avec `INVALID_AMOUNT`). En requête, le montant peut être un nombre ou une chaîne (`"10000"`) ; en réponse, c'est toujours un nombre.

Pas de flottants pour l'argent

Calculez les montants en entiers (devises sans décimales) ou avec une bibliothèque décimale (`decimal.Decimal` en Python, `BigDecimal` en Java…). N'utilisez pas des flottants binaires : en JavaScript, `0.1 + 0.2` ne fait pas `0.3`.

4.1.2 Idempotence : réessayer sans risque

Sur un réseau, une requête ou sa réponse peut se perdre. L'`order_id` (**votre** référence de commande, 1 à 128 caractères parmi `A-Z a-z 0-9 . _ : -`) vous permet de réessayer sans risque :

- Il est unique **par type d'opération** : la même valeur peut exister une fois en pay-in et une fois en pay-out.
- Renvoyer **exactement la même requête** avec le même `order_id` ne crée rien de nouveau : vous recevez la transaction d'origine, avec le code `200` (au lieu de `202`) et l'en-tête `Idempotent-Replayed: true`.
- Le réutiliser avec un contenu **différent** (montant, numéro…) est refusé : `409 DUPLICATE_ORDER`.
- Après un échec définitif (`FAILED` ou `EXPIRED`), utilisez un **nouvel** `order_id` pour une nouvelle tentative.

Après un délai dépassé

Si votre `POST` expire sans réponse, **renvoyez la même requête avec le même `order_id`** (et une nouvelle signature) : vous obtiendrez la transaction créée la première fois, ou une création normale, jamais deux paiements. Générez et enregistrez l'`order_id` *avant* l'appel.

4.2 Vos méthodes : `GET /v1/methods`

Avant de payer, découvrez ce qui est ouvert pour vous. Ne codez pas en dur le pays, le type ou le nom de la méthode : lisez-les ici.

```
{ "data": [ {
    "country": "TD", "payment_type": "TD_MOBILEMONEY",
    "method": "MOOV_MONEY", "currency": "XAF",
    "directions": [
      { "direction": "PAYIN",
        "fee": { "rate_percent": 1.5, "fixed": 0, "min": null, "max": null },
        "min_amount": null, "max_amount": null },
      { "direction": "PAYOUT",
        "fee": { "rate_percent": 1.5, "fixed": 0, "min": null, "max": null },
        "min_amount": null, "max_amount": null } ] } ] }
```

Exemple 4.1 – Une méthode ouverte (valeurs d'exemple)

Une méthode est ouverte *par sens* : si `PAYOUT` manque dans `directions`, vous pouvez encaisser avec cette méthode mais pas verser. Pour chaque sens :

— `fee` : le pourcentage du montant (`rate_percent`, tout compris), la part fixe (`fixed`), et les bornes éventuelles (`min`, `max`) ;

— `min_amount`, `max_amount` : les limites d'un montant (`null` : pas de limite). Hors limites : `400 AMOUNT_OUT_OF_RANGE`.

4.3 Encaisser : `POST /v1/payins`

Demande au client de confirmer le paiement sur son téléphone. La réponse est immédiate, avec le statut `INITIATED` ; le résultat arrive par webhook ou par consultation.

Champ	Type	Requis	Description
`country`	chaîne	oui	Pays, code ISO 3166-1 alpha-2 (`TD`).
`payment_type`	chaîne	oui	Type de paiement, tel que listé par `GET /v1/methods`.
`method`	chaîne	oui	Méthode, telle que listée par `GET /v1/methods`.
`currency`	chaîne	oui	Devise ISO 4217 (`XAF`). Doit être celle de la méthode.
`order_id`	chaîne	oui	Votre référence, unique par type d'opération (section 4.1.2).
`amount`	nombre ou chaîne	oui	Montant demandé au client (section 4.1.1).
`phone`	chaîne	oui	Numéro du client, format international.
`return_url`	chaîne (URL)	non	Pour les méthodes avec redirection : adresse de retour du client. HTTPS obligatoire, sur un domaine enregistré chez nous.
`customer`	objet	selon la méthode	Le payeur : `email`, `ip`, `first_name`, `last_name`. Voir ci-dessous.
`payment_details`	objet	selon la méthode	Champs propres à la méthode (code banque, numéro de compte…), tous des chaînes. Voir ci-dessous.

Méthodes qui demandent plus qu'un numéro de téléphone. Certaines méthodes (une page de paiement hébergée, un virement bancaire) demandent des informations sur le payeur. Chaque méthode de `GET /v1/methods` liste ses champs dans `fields`, par nom (`customer.email`, `customer.ip`, `payment_details.payer.account_number`…) et dit s'ils sont obligatoires. Un champ obligatoire manquant, ou un champ que la méthode ne déclare pas, est refusé avec `INVALID_REQUEST` : rien n'est créé. Une méthode qui ne déclare rien (les méthodes mobile money actuelles) se contente du numéro de téléphone.

`customer.ip` est l'adresse IP du **client final** (celle de son navigateur ou de son téléphone), pas celle de votre serveur : transmettez-la telle que vous la recevez. La réponse contient alors `next_action` de type `redirect\_url` : redirigez le client vers cette adresse pour qu'il finalise le paiement.

```
{ "country": "TD", "payment_type": "TD_MOBILEMONEY", "method": "MOOV_MONEY",
  "currency": "XAF", "order_id": "CMD-1782539795100",
  "amount": 10000, "phone": "23597326770" }
```

```
{ "transaction_ref": "AD11CF83F0D31E2482518BF2E336E791",
  "order_id": "CMD-1782539795100", "type": "PAYIN",
  "country": "TD", "payment_type": "TD_MOBILEMONEY", "method": "MOOV_MONEY",
  "currency": "XAF", "amount": 10000, "fee": 150, "net_amount": 9850,
  "status": "INITIATED", "next_action": null, "operator_reference": null,
  "error_code": null, "created_at": "2026-10-02T10:00:00Z",
  "updated_at": "2026-10-02T10:00:00Z", "completed_at": null }
```


Exemple 4.2 – Requête (corps) et réponse HTTP 202

Exemple : le Nigeria (virement bancaire). Pour une méthode qui demande le payeur, la requête ressemble à ceci. Le numéro de téléphone est au format international, **sans +** (2348123456789). L'e-mail, l'adresse IP du client final, le prénom et le nom sont obligatoires ; les champs `payment_details.payer.*` sont facultatifs. La réponse contient un `next_action` de type `redirect\_url` vers la page de paiement.


```
{ "country": "NG", "payment_type": "NG_BANK", "method": "BANK_TRANSFER",
  "currency": "NGN", "order_id": "CMD-1782539795101", "amount": "1000.50",
  "phone": "2348123456789", "return_url": "https://shop.example/merci",
  "customer": { "email": "john@shop.example", "ip": "101.121.34.143",
                "first_name": "John", "last_name": "Doe" },
  "payment_details": { "payer.account_number": "3660222", "payer.account_bank": "055" } }
```


Exemple 4.3 – Pay-in avec redirection (requête)

L'objet transaction.

Champ	Description
`transaction_ref`	Référence de la transaction chez nous.
`order_id`	Votre référence.
`type`	`PAYIN`, `PAYOUT` ou `TOPUP` (un top-up est un pay-in qui recharge votre solde, fait depuis le tableau de bord).
`amount`	Montant demandé.
`fee`	Frais totaux, **figés** à la création : un changement de tarif ne modifie pas les transactions existantes.
`net_amount`	Pay-in : montant crédité à votre solde (`amount - fee`). Pay-out : montant total débité (`amount + fee`).
`status`	Voir la section 4.5.
`next_action`	Ce que le client doit faire : `{"type":"ussd_push"}` (valider sur son téléphone) ou `{"type":"redirect_url","url":"..."}` (redirigez-le vers cette adresse).
`operator_reference`	Identifiant de l'opération chez l'opérateur, quand il existe.
`error_code`	`PAYMENT_FAILED` si le paiement a échoué, sinon `null`.
`created_at`, `updated_at`, `completed_at`	Dates ISO 8601 UTC ; `completed_at` est `null` tant que la transaction n'est pas finale, et il le reste pour `EXPIRED`.

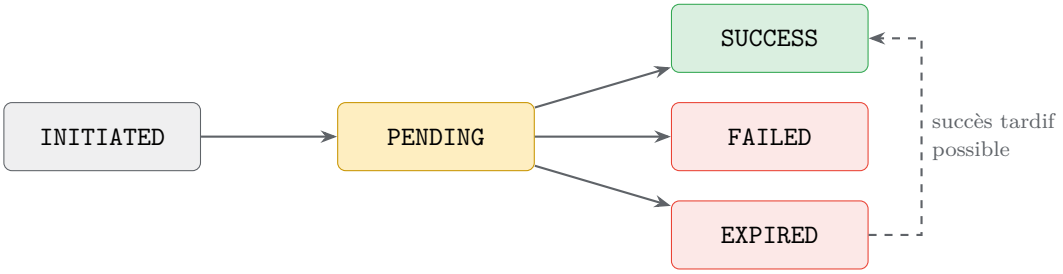
4.4 Verser : `POST /v1/payouts`

Mêmes champs que le pay-in. Votre solde est débité du montant **plus les frais**, *réservés dès l'acceptation* ; le montant est versé au numéro indiqué. Si votre solde disponible ne couvre pas montant + frais, la réponse est `422 INSUFFICIENT_BALANCE` et rien n'est créé. Si le pay-out échoue, les fonds réservés sont libérés.

Un pay-out ne se rejoue jamais « pour voir »

Quand l'opérateur ne répond pas clairement, Gwongzho Pay **ne suppose jamais un échec** : l'argent a peut-être été versé. La transaction reste `PENDING` et les fonds restent réservés le temps de la vérification, parfois plusieurs heures. Ne recréez **pas** un pay-out avec un nouvel `order_id` parce que le premier tarde : consultez son état, attendez son statut final, et en cas d'attente anormale contactez le support avec la `transaction_ref`. Un second pay-out paierait deux fois.

4.5 Statuts et cycle de vie



Statut	Final ?	Signification et action
`INITIATED`	non	Demande acceptée et enregistrée. C'est le statut de la réponse à `POST`. Affichez « en attente » ; n'expédiez pas encore.
`PENDING`	non	Transmise à l'opérateur. Pour un pay-in, le client n'a pas encore confirmé ; pour un pay-out, le résultat n'est pas connu. Même action.
`SUCCESS`	**oui**	L'argent a bougé. Un pay-in a crédité votre solde de `net_amount` ; un pay-out l'a débité définitivement. Marquez la commande payée, **une seule fois**.
`FAILED`	**oui**	Le paiement n'a pas eu lieu (`error_code` vaut `PAYMENT_FAILED`). Aucun argent n'a bougé ; pour un pay-out, les fonds réservés sont libérés. Proposez un nouvel essai avec un **nouvel** `order_id`.
`EXPIRED`	presque	Le client n'a pas confirmé à temps. Traitez-le comme un échec, mais voyez l'encadré ci-dessous.

Un statut `SUCCESS` ou `FAILED` **ne change plus jamais**. L'opérateur ne fournit pas de cause précise d'échec : ne promettez pas à votre client une explication détaillée, dites que le paiement n'a pas abouti et proposez de réessayer.

Le succès tardif

Un pay-in `EXPIRED` peut être suivi d'un `SUCCESS` si le client valide finalement (par exemple parce que la demande était encore affichée sur son téléphone). C'est le seul cas où un statut «

négatif » évolue : vous recevez alors un webhook `SUCCESS`. Si vous avez annulé la commande à l'expiration, prévoyez de traiter ce cas (rétablir la commande ou rembourser).

4.6 Consulter des transactions

Une transaction : `/v1/transactions/{transaction_ref}`. Renvoie la transaction à jour. Une référence inexistante, ou appartenant à un autre compte, donne la même réponse `404 NOT_FOUND`.

Une liste : `GET /v1/transactions`. Les plus récentes d'abord. Elle contient aussi les opérations faites depuis le tableau de bord : les recharges de solde (`type TOPUP`, `order_id` commençant par `TOPUP-`) et les retraits (`PAYOUT`, `order_id` commençant par `WD-`). Filtres facultatifs :

Paramètre	Description
`order_id`	Votre référence : retrouver une commande.
`type`	`PAYIN`, `PAYOUT` ou `TOPUP`.
`status`	`INITIATED`, `PENDING`, `SUCCESS`, `FAILED` ou `EXPIRED`.
`method`, `country`	Filtrer par méthode ou par pays.
`from`, `to`	Période : date-heure ISO 8601 ; `from` inclus, `to` exclu.
`limit`	1 à 200 (50 par défaut).
`cursor`	La valeur `next_cursor` de la page précédente.

La réponse est `{"data": [...], "next_cursor": "..."}`. Tant que `next_cursor` n'est pas `null`, passez-le dans `cursor` pour la page suivante. Le curseur est opaque : ne le fabriquez pas. **Signez chaque page** : le curseur fait partie de la requête.

Un filet de sécurité utile

Combinez webhooks et consultation : une tâche périodique relit avec `GET /v1/transactions?status=PENDING` (et `INITIATED`) les transactions restées en attente depuis plus de quelques minutes. Elle rattrape un webhook définitivement manqué, par exemple si votre serveur est resté en panne plus longtemps que les réessais.

4.7 Votre solde : `GET /v1/balance`

Un solde par devise :

```
{ "data": [ { "currency": "XAF", "available": 98500, "reserved": 10150 } ] }
```

Exemple 4.4 – Réponse

— `available` : ce que vous pouvez dépenser. Un pay-out est refusé si `available` ne couvre pas montant + frais.

— `reserved` : ce qui est retenu par des pay-outs en cours (montant et frais). Il retourne dans `available` si le pay-out échoue, ou est définitivement débité s'il réussit.

Votre solde augmente avec vos pay-ins réussis (du `net_amount`) et avec les recharges faites depuis le tableau de bord (page *Top-up*) ; il diminue avec vos pay-outs et vos retraits. Pour pouvoir verser, il faut donc avoir encaissé ou rechargé avant.

Le solde total d'une devise est `available + reserved`. L'historique détaillé des mouvements (relevé, exports) est dans le tableau de bord.

4.8 Les frais

Les frais d'une opération se calculent ainsi, avec les valeurs de `GET /v1/methods` :

$$\texttt{frais} \;=\; \left\lceil \texttt{montant} \times \frac{\texttt{rate_percent}}{100} \right\rceil \;+\; \texttt{fixed},$$

puis ramenés dans les bornes `min` et `max` si elles existent. L'arrondi se fait à l'unité supérieure. Pour un pay-in, les frais doivent rester inférieurs au montant.

Exemple. Avec 1,5 % et aucune part fixe : un pay-in de 10 000 XAF coûte 150 et crédite 9 850 ; un pay-out de 10 000 XAF débite 10 150 (le bénéficiaire reçoit 10 000).

Chapitre 5

Webhooks

Un *webhook* est un appel HTTPS que Gwongzho Pay fait vers votre serveur chaque fois qu'une transaction change de statut. Il vous évite d'interroger l'API en boucle pour savoir si un paiement est arrivé.

5.1 Quand et vers où

Quand. À **chaque changement de statut** d'un pay-in ou d'un pay-out, *après* la création (la création est déjà la réponse de votre appel). Un pay-in réussi produit par exemple deux webhooks : `INITIATED` → `PENDING`, puis `PENDING` → `SUCCESS`.

Où. Vers l'URL que vous enregistrez dans le tableau de bord, page *Developers*, onglet du webhook. Elle est propre à chaque environnement et reçoit un **secret de webhook** distinct du secret de votre clé API, montré une seule fois (et à chaque rotation). Règles sur l'URL :

- **HTTPS obligatoire**, avec un certificat valide ; pas d'identifiants dans l'URL.
- Pas d'adresse interne : l'hôte ne doit résoudre qu'en adresses **publiques** (pas de `localhost`, d'adresses privées ou de lien local), à l'enregistrement et à chaque envoi.
- Les **redirections** ne sont pas suivies.

Tester en local

Exposez votre poste avec un tunnel HTTPS (par exemple `cloudflared` ou `ngrok`) et enregistrez l'adresse publique du tunnel. Le tableau de bord permet d'envoyer un **événement de test**, de consulter le **journal des livraisons** et de **rejouer** un événement.

5.2 Ce que vous recevez

Une requête `POST` avec un corps JSON et ces en-têtes :

En-tête	Valeur
`X-Timestamp`	Heure d'envoi, en secondes Unix (UTC).
`X-Signature`	Signature hexadécimale minuscule (section suivante).
`X-Event-Id`	Identifiant de l'événement (identique à `event_id` du corps).

```
{
  "event_id": "evt_0123456789abcdef0123456789abcdef",
  "type": "transaction.status_changed",
```

```
  "version": 3,
  "created_at": "2026-10-02T10:00:30Z",
  "data": {
    "previous_status": "PENDING",
    "transaction": {
      "transaction_ref": "AD11CF83F0D31E2482518BF2E336E791",
      "order_id": "CMD-1782539795100",
      "type": "PAYIN",
      "country": "TD",
      "payment_type": "TD_MOBILEMONEY",
      "method": "MOOV_MONEY",
      "currency": "XAF",
      "amount": 10000,
      "fee": 150,
      "net_amount": 9850,
      "status": "SUCCESS",
      "next_action": { "type": "ussd_push" },
      "operator_reference": "SBX-1",
      "error_code": null,
      "created_at": "2026-10-02T10:00:00Z",
      "updated_at": "2026-10-02T10:00:30Z",
      "completed_at": "2026-10-02T10:00:30Z"
    }
  }
}
```


Exemple 5.1 – Corps complet d'un webhook (pay-in passé à SUCCESS)

Pour un paiement échoué, `status` vaut `FAILED`, `error_code` vaut `PAYMENT_FAILED` et `completed_at` est renseigné ; pour une expiration, `status` vaut `EXPIRED` et `completed_at` reste `null`.

— `event_id` : unique ; un même événement renvoyé (réessai, rejeu) porte **toujours le même**.

— `type` : actuellement `transaction.status_changed`. Ignorez les types inconnus (répondez `2xx`).

— `version` : compteur **croissant par transaction**, pour ordonner les événements d'une même transaction.

— `data.previous_status` : le statut avant le changement.

— `data.transaction` : la transaction dans son état après le changement, avec exactement les champs renvoyés par l'API (section 4.5).

Votre réponse. Répondez par un code `2xx` (par exemple `200` ou `204`) dans les 10 secondes ; le contenu de la réponse est ignoré. Tout autre code, ou l'absence de réponse, déclenche un réessai (section 5.4).

Le corps est **figé** à la création de l'événement : un réessai envoie les mêmes octets, avec une nouvelle signature et un nouvel horodatage.

5.3 Vérifier la signature

Toujours vérifier

Votre URL de webhook est publique : n'importe qui peut l'appeler. Sans vérification de la signature, un tiers pourrait déclarer « payée » n'importe quelle commande. **Ne traitez aucun webhook dont la signature n'est pas valide.**

La signature se calcule avec le **secret du webhook** :

`X-Signature` = hex(HMAC-SHA256(`secret_webhook`, `X-Timestamp` ‖ `"\n"` ‖ sha256hex(corps brut)))

c'est-à-dire une chaîne de deux lignes : l'horodatage, puis le SHA-256 hexadécimal du corps *brut*.

Procédure :

1. Lisez le corps **brut**, avant tout décodage JSON (les octets exacts reçus).
2. Refusez si `X-Timestamp` s'écarte de plus de 5 minutes de votre heure.
3. Recalculez la signature et comparez **en temps constant** (`hmac.compare_digest`, `crypto.timingSafeEqua` `hash_equals`, `hmac.Equal`), jamais avec `==`.
4. Dédupliquez sur `event_id` et ordonnez avec `version`.
5. Répondez `2xx` **rapidement** (moins de 10 secondes), avant tout traitement long.

Vecteur de test. Secret `whsec_secret`, horodatage `1700000000`, corps `{"country":"TD","order_id":"CMD-` (sans espace) : la signature attendue est

```
385ce9332513855982671b8857fddb23e5549f380e10a2297c94eba3bb82ff5b
```

```python
import hashlib, hmac, os, time
from flask import Flask, request, abort

app = Flask(__name__)
WEBHOOK_SECRET = os.environ["GWONGZHO_WEBHOOK_SECRET"].encode()

def valid(ts, raw, signature):
    if not ts.isdigit() or abs(time.time() - int(ts)) > 300:
        return False
    text = f"{ts}\n{hashlib.sha256(raw).hexdigest()}"
    expected = hmac.new(WEBHOOK_SECRET, text.encode(), hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, signature)

@app.post("/webhooks/payments")
def payments():
    raw = request.get_data()                        # octets bruts, AVANT tout json.loads
    if not valid(request.headers.get("X-Timestamp", ""), raw, request.headers.get("X-Signature"
    ↪ , "")):
        abort(401)
    event = request.get_json()
    # 1. dédupliquer sur event["event_id"]   2. ignorer si event["version"] est plus ancienne
    # 3. mettre à jour la commande (idempotent), puis répondre vite
    return "", 204
```

Exemple 5.2 – Récepteur de webhook (Flask)

Pour vérifier à la main un webhook reçu, utilisez la commande `verify-webhook` des scripts d'exemple, avec le secret dans `GWONGZHO_WEBHOOK_SECRET`, l'en-tête `X-Timestamp`, l'en-tête `X-Signature`

et un fichier contenant le corps brut.

5.4 Fiabilité de la livraison

— **Au moins une fois.** Un même événement peut arriver plusieurs fois : rendez votre traitement **idempotent** (dédoublonnage sur `event_id`).

— **Pas d'ordre garanti.** Utilisez `version` : ignorez un événement plus ancien que celui déjà traité pour cette transaction.

— **Réessais.** Sans réponse `2xx` dans les 10 secondes, nous réessayons après 10 s, 30 s, 2 min, 10 min, 30 min, 1 h, 3 h, 6 h puis 12 h, et nous abandonnons ensuite. Un événement abandonné peut être rejoué depuis le tableau de bord.

— **Ne vous fiez pas qu'aux webhooks.** `/v1/transactions/{transaction_ref}` donne toujours l'état à jour ; une tâche de rapprochement (chapitre 4) rattrape ce qui aurait été manqué.

— **Le navigateur ne fait pas foi.** Ne marquez jamais une commande payée parce que le client est revenu sur votre page de retour : seuls le webhook signé ou la consultation par l'API font foi.

Chapitre 6

Erreurs

Une erreur a toujours la même forme. Testez le `code`, qui est stable ; le `message` s'adresse à un humain, est en anglais et peut changer.

```
{ "error": { "code": "INSUFFICIENT_BALANCE",
             "message": "insufficient available balance",
             "request_id": "req_84578a47abf909d9",
             "details": {} } }
```

Exemple 6.1 – Format d'une erreur

6.1 Codes

HTTP	Code	Cause et action
400	INVALID_REQUEST	JSON mal formé, champ manquant, inconnu ou hors format. Corrigez la requête.
400	INVALID_PHONE	Numéro invalide pour le pays.
400	INVALID_AMOUNT	Montant nul ou négatif, trop de décimales pour la devise, ou trop petit pour couvrir les frais.
400	AMOUNT_OUT_OF_RANGE	Montant hors des limites de la méthode (`GET /v1/methods`).
400	CURRENCY_MISMATCH	La devise n'est pas celle de la méthode.
400	INVALID_RETURN_URL	`return_url` non HTTPS ou d'un domaine non enregistré.
401	UNAUTHORIZED	En-tête d'authentification manquant ou horodatage non numérique.
401	INVALID_SIGNATURE	Signature fausse, clé inconnue, révoquée ou d'un autre environnement, ou horodatage hors de la fenêtre de 5 minutes (chapitre 3).
401	REPLAYED_REQUEST	Cette requête signée a déjà été utilisée. Signez à nouveau avec un nouvel horodatage.
403	FORBIDDEN	Droits insuffisants, ou adresse IP non autorisée pour cette clé.
403	METHOD_NOT_ENABLED	Méthode inconnue ou non ouverte pour ce sens sur votre compte. Voyez `GET /v1/methods`, ou contactez-nous.
403	MERCHANT_SUSPENDED	Votre compte n'est pas actif. Contactez le support.

HTTP	Code	Cause et action
404	NOT_FOUND	Ressource introuvable (identique pour une ressource inexistante et pour celle d'un autre compte).
409	DUPLICATE_ORDER	`order_id` déjà utilisé avec un contenu différent. Utilisez un nouvel identifiant, ou renvoyez exactement la requête d'origine.
409	INVALID_STATE	Action impossible dans l'état actuel de la ressource.
413	REQUEST_TOO_LARGE	Corps supérieur à 1 Mio.
415	UNSUPPORTED_MEDIA_TYPE	Le contenu n'est pas du JSON.
422	INSUFFICIENT_BALANCE	Solde disponible inférieur à montant + frais (pay-out). Créditez votre compte ou réduisez le montant.
429	RATE_LIMITED	Trop de requêtes. Attendez `Retry-After` secondes.
500	INTERNAL_ERROR	Erreur de notre côté. Réessayez ; si elle persiste, transmettez le `request_id` au support.
503	METHOD_UNAVAILABLE	Méthode temporairement indisponible (maintenance de l'opérateur). Réessayez plus tard.

6.2 Que réessayer ?

Situation	Réessayer ?
400, 403, 404, 409, 413, 415, 422	**Non** : la requête telle quelle échouera de nouveau. Corrigez d'abord.
401 REPLAYED_REQUEST, 401 INVALID_SIGNATURE (horloge)	Oui, **avec une nouvelle signature**, une fois la cause corrigée.
429	Oui, après `Retry-After`, avec un délai croissant.
500, 503, délai dépassé, erreur réseau	Oui, avec un délai croissant. Pour les créations, gardez le **même** `order_id` : c'est ce qui rend le réessai sans risque.

```python
import random, time

RETRYABLE = {429, 500, 502, 503, 504}

def create_with_retry(uri, payload, attempts=5):
    for n in range(attempts):
        try:
            status, data, headers = call("POST", uri, payload)   # re-signe a chaque tentative
        except OSError:
            status, headers = None, {}                           # reseau / delai depasse
        if status is not None and status not in RETRYABLE:
            return status, data
        wait = int(headers.get("Retry-After", 0))
        time.sleep(max(wait, min(30, 2 ** n)) + random.random())
    raise RuntimeError("le service ne répond pas ; gardez le même order_id pour la suite")
```

Exemple 6.2 – Réessai avec délai croissant, en gardant l'order_id et en re-signant

6.3 Erreur d'API ou échec de transaction ?

Ne confondez pas une *erreur d'API* (la requête est refusée, aucune transaction n'est créée) et un *échec de transaction* (la requête est acceptée, puis le paiement échoue : `status = FAILED`, `error_code = PAYMENT_FAILED`). Le second se traite comme un statut, pas comme une exception.

Chapitre 7

Sandbox et mise en production

7.1 Tester en sandbox

La sandbox se comporte comme la production, sans argent réel et sans opérateur : le résultat d'un paiement dépend du **dernier chiffre du numéro de téléphone**.

Dernier chiffre	Pay-in	Pay-out
`0` à `4`, `9`	Le client confirme après quelques secondes : `SUCCESS`.	`SUCCESS` immédiat.
`5`	Le client refuse : `FAILED`.	Refusé : `FAILED`.
`6`	Le client ne répond pas : `EXPIRED`.	Traité comme `0`.
`7`	Réponse perdue, puis `SUCCESS` (cas « incertain »).	Réponse perdue, puis `SUCCESS`.
`8`	Échec net dès l'initiation : `FAILED`.	Échec net : `FAILED`.

— Pour avoir un **solde** avant de tester un pay-out, faites d'abord un pay-in réussi (chiffre `0`) : il crédite votre solde de `net_amount`.

— Testez **aussi les cas d'échec** : `5` (refus), `6` (expiration) et `7` (réponse incertaine). C'est là que les intégrations se trompent.

— Testez vos **webhooks** : enregistrez l'URL de sandbox, faites un paiement et vérifiez la signature, la déduplication et l'ordre (chapitre 5).

7.2 Liste de contrôle avant la production

	Vérification
☐	Les signatures sont calculées sur le corps et le chemin *exacts*, et re-calculées à chaque tentative.
☐	L'horloge de vos serveurs est synchronisée (NTP).
☐	Les secrets sont hors du code et hors des journaux ; une clé par application et par environnement.
☐	Les clés de production sont restreintes à vos adresses IP de sortie (surtout pour les pay-outs).
☐	Chaque commande a un `order_id` unique, enregistré *avant* l'appel, et gardé pour tout réessai.
☐	Les erreurs réseau, `429`, `500` et `503` sont réessayées avec un délai croissant.
☐	Le récepteur de webhook vérifie la signature sur le corps brut, en temps constant, répond `2xx` vite, déduplique sur `event_id` et ignore une `version` plus ancienne.
☐	Une tâche de rapprochement relit les transactions restées `INITIATED` ou `PENDING`.
☐	Le succès tardif d'un pay-in `EXPIRED` est géré.
☐	Aucun pay-out n'est recréé avec un autre `order_id` avant que le premier soit final.
☐	Les montants sont des entiers ou des décimaux exacts, jamais des flottants.
☐	Les cas d'échec (`FAILED`, `EXPIRED`, incertain) ont été testés en sandbox.
☐	La page de retour du navigateur ne marque jamais une commande payée : seuls le webhook signé et l'API font foi.
☐	Vous avez fait un premier paiement réel de faible montant et contrôlé le webhook, le solde et `operator_reference`.

7.3 Passer en production

1. Créez une **clé de production** dans le tableau de bord (la clé de sandbox ne fonctionnera pas), et enregistrez l'**URL de webhook de production**.
2. Remplacez l'adresse de base par `https://api-gateway.gwongzhopay.com` et les clés par celles de production.
3. Vérifiez que vos méthodes de production sont ouvertes (`GET /v1/methods`) et que votre solde permet vos pay-outs.
4. Faites un premier paiement de faible montant avant d'ouvrir à vos clients.

7.4 Demander de l'aide

Écrivez à support@gwongzho.com en indiquant :

— l'environnement (sandbox ou production) ;

— le `X-Request-Id` de la réponse concernée, et/ou la `transaction_ref` ;

— l'heure approximative, avec le fuseau horaire ;

— ce que vous attendiez et ce que vous avez obtenu.

Ne nous envoyez jamais vos secrets

N'envoyez jamais un secret de clé, un secret de webhook ni une signature complète par e-mail ou par messagerie. Nous n'en avons jamais besoin : le `X-Request-Id` suffit à retrouver votre requête.