# ZNotice — Operations Runbook

Operational procedures for the ZNotice host. All commands assume root (or `sudo`). Never paste secret values into tickets, logs, or shell history.

- App root: `/var/www/znotice.consultbofa.com`
- systemd unit: `znotice.service`
- Engine container: `wppconnect`
- Database: MariaDB database `znotice`

## 1. Service control

```
# Application
systemctl status znotice
systemctl start znotice
systemctl stop znotice
systemctl restart znotice
systemctl is-active znotice

# WhatsApp engine (Docker)
docker ps --filter name=wppconnect
docker start wppconnect
docker stop wppconnect
docker restart wppconnect
docker logs --tail=200 wppconnect

# Supporting services
systemctl status nginx mariadb
nginx -t          # validate config before reload
systemctl reload nginx
```

> **Policy:** do **not** restart the `wppconnect` container to fix an application problem. Restarting the engine drops live WhatsApp sessions and forces every tenant to re-scan a QR code. The health check follows the same policy and never auto-restarts the engine.

### Deploy / restart the app after a code change

```
cd /var/www/znotice.consultbofa.com
npm ci --omit=dev          # only if dependencies changed
npm run migrate            # apply any pending migrations
systemctl restart znotice
systemctl status znotice --no-pager
curl -sk --resolve znotice.consultbofa.com:443:127.0.0.1 https://znotice.consultbofa.com/ -o /dev/
null -w '%{http_code}\n'
```

> **Order matters.** Migrations must be applied **before** the restart whenever a release adds tables/columns. The Phase 12 release (`012_payment_proofs.sql`) is a hard example: the new code queries `payment_proofs`, so restarting first makes the pay page and proof routes 500 until the table exists. There is no runtime fallback — `npm run migrate` then `systemctl restart znotice`.

## 2. Logs

The application writes to **stdout/stderr**, captured by journald. There is no application file log (the `logs/` directory is currently unused).

```
# Application (structured console output: startup, send failures, webhooks)
journalctl -u znotice -n 200 --no-pager
journalctl -u znotice -f
journalctl -u znotice --since "1 hour ago"

# WhatsApp engine
docker logs --tail=200 wppconnect
docker logs -f wppconnect

# Ops scripts (file logs, mode 0640 root:root)
tail -n 100 /var/log/znotice-backup.log
tail -n 100 /var/log/znotice-health.log
```

Log rotation is configured in `/etc/logrotate.d/znotice`: weekly, 8 rotations, compressed, for `znotice-backup.log` and `znotice-health.log`.

Useful log strings:

| String | Meaning |
|---|---|
| `[znotice] listening on ...` | app booted |
| `[error] <METHOD> <path>:` | unhandled request error |
| `[send] engine failure user=...` | engine rejected a send (credit refunded) |
| `[scheduler] processed=... sent=... failed=...` | a scheduler tick dispatched at least one item |
| `[reaper] ... sending_interrupted=...` | a reaper pass changed something (includes interrupted scheduler claims and session pruning) |
| `[reaper] ... sessions_deleted=...` | the reaper pruned expired/stale anonymous sessions |
| `[auth] password reset requested user=... ip=...` | a reset link was generated (no email content or token is logged) |
| `[auth] password reset completed user=... ip=... sessions_removed=...` | a reset was consumed and the user's sessions were destroyed |
| `[email] SMTP_HOST is not configured; outgoing email is disabled` | email is inert — only when `SMTP_*` is unset (logged once per process) |
| `[email] send failed: <code> response=<n>` | an SMTP delivery failed (sanitized: no recipient/body/credentials) |
| `[webhook] handler error:` | webhook processing error |
| `[whatsapp] closeSession ... failed` | engine close-session quirk (see §10) |
| `[whatsapp] setProfileName ... failed` | engine rejected a sender-name change (detail is logged server-side only; the customer sees a generic error) |

## 3. Database access

Credentials are in `/var/www/znotice.consultbofa.com/.env` (see `.env`).

```
# Interactive shell (supply credentials from .env; do not echo them)
mysql -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER" -p znotice

# One-off query example
mysql -h "$DB_HOST" -u "$DB_USER" -p znotice \
  -e "SELECT id,email,role,status FROM users ORDER BY id LIMIT 20;"
```

Common inspection queries:

```
-- balance for a user (authoritative)
SELECT COALESCE(SUM(delta),0) AS balance FROM credit_ledger WHERE user_id = ?;

-- recent failed sends
SELECT id,user_id,to_number,status,error,created_at
  FROM messages WHERE status='failed' ORDER BY created_at DESC LIMIT 20;

-- pending manual payments awaiting approval
SELECT id,user_id,provider_ref,amount_cents,credits,created_at
  FROM transactions WHERE status='pending' ORDER BY created_at;

-- sessions occupying gateway capacity
SELECT session_name,status,updated_at FROM wa_sessions
 WHERE status IN ('pending','connected');

-- open scheduled queue for a user (pending + sending)
SELECT id,batch_id,to_number,status,scheduled_at,next_attempt_at,attempts,last_error
  FROM scheduled_messages WHERE user_id = ? AND status IN ('pending','sending')
 ORDER BY scheduled_at ASC, id ASC LIMIT 100;

-- stuck scheduler claims (crash recovery candidates; reaper handles >10 min)
SELECT id,user_id,batch_id,to_number,status,updated_at
  FROM scheduled_messages WHERE status='sending'
   AND updated_at < DATE_SUB(NOW(), INTERVAL 10 MINUTE);

-- recent batches for a user
SELECT id,label,status,total_count,sent_count,failed_count,cancelled_count,scheduled_at
  FROM send_batches WHERE user_id = ? ORDER BY created_at DESC LIMIT 20;

-- sender-name add-on entitlement (NULL = never purchased; active while > NOW())
SELECT id,email,sender_name_until,
       (sender_name_until IS NOT NULL AND sender_name_until > NOW()) AS active
  FROM users WHERE id = ?;

-- pending sender-name purchases awaiting approval
SELECT id,user_id,provider_ref,amount_cents,status,meta,created_at
  FROM transactions
 WHERE status='pending'
   AND JSON_UNQUOTE(JSON_EXTRACT(meta,'$.type'))='sender_name'
 ORDER BY created_at;
```

## 4. Migrations workflow

Migrations are plain SQL files in `server/migrations/`, applied in lexical order exactly once and tracked in `schema_migrations`.

```
cd /var/www/znotice.consultbofa.com
npm run migrate
```

- `node server/db.js` `runMigrations()` runs each file on a dedicated connection with `multipleStatements` enabled.
- Already-applied files are skipped and listed as such.
- Migration SQL is written defensively (`IF NOT EXISTS`, guarded `information_schema` checks) so a partially-applied file can be retried.
- **Current head:** `012_payment_proofs.sql` (Phase 12) creates the `payment_proofs` table and its `transactions` FK/index. Migrations `008` – `011` are the password-reset, sender-name and template additions; `012` is the payment-proof table.
- **Deploy order matters (Phase 12).** Run `npm run migrate` **before** restarting `znotice`. The proof code queries `payment_proofs` with no runtime fallback, so restarting first makes the pay page (`GET /dashboard/billing/pay/:id`) and the proof routes return `500` until the table exists. Migrate, then restart.
- **To add a migration:** create `server/migrations/00N_description.sql`, keep it idempotent where practical, run `npm run migrate`, then restart the app only if route/service code changed.

## 5. Backup and restore

### Automated backup

- Script: `/usr/local/bin/znotice-backup.sh`
- Schedule: `/etc/cron.d/znotice-backup` — daily at **02:30**
- Output:
  - DB dumps: `/var/www/znotice.consultbofa.com/backups/db/znotice-<stamp>.sql.gz` (keep 14)
  - App + secrets: `/var/www/znotice.consultbofa.com/backups/app/znotice-app-<stamp>.tar.gz` (keep 7)
- The app archive contains `server/ public/ package.json package-lock.json .env` and excludes `node_modules/ logs/ uploads/ backups/`.

> **Note:** `uploads/` holds the Phase 12 payment-proof files, which are therefore **not** in the archive. The DB dump keeps the `payment_proofs` metadata rows but not the file bytes, so a DB-only restore leaves rows pointing at missing files. See §15 for the open retention/backup recommendation.

- Permissions: directories `0750`, files `0640`, owner `root:www-data` (the archive contains `.env`, so it is never world-readable).
- Log: `/var/log/znotice-backup.log`.

```
# Run a backup on demand
/usr/local/bin/znotice-backup.sh; tail -n 5 /var/log/znotice-backup.log
ls -lt /var/www/znotice.consultbofa.com/backups/db | head
ls -lt /var/www/znotice.consultbofa.com/backups/app | head
```

### Manual database dump

```
# Credentials from .env; --single-transaction keeps it safe while live.
mysqldump --single-transaction --routines --triggers \
  -h "$DB_HOST" -u "$DB_USER" -p znotice | gzip -c > /tmp/znotice-manual-$(date +%F).sql.gz
gzip -t /tmp/znotice-manual-$(date +%F).sql.gz && echo OK
```

### Restore — database

> Restore into an empty/maintenance window. The app should be stopped to avoid writes mid-restore.

```
systemctl stop znotice
gunzip -c /var/www/znotice.consultbofa.com/backups/db/znotice-<stamp>.sql.gz \
  | mysql -h "$DB_HOST" -u "$DB_USER" -p znotice
systemctl start znotice
```

### Restore — application files

```
systemctl stop znotice
cd /var/www/znotice.consultbofa.com
tar xzf backups/app/znotice-app-<stamp>.tar.gz     # restores server/ public/ .env
chown -R www-data:www-data server public
systemctl start znotice
```

> The app archive includes `.env`. After restoring, verify the environment matches the current host (DB host, `APP_URL`, `WEBHOOK_SECRET`, `WPPCONNECT_SECRET_KEY`) before starting.

## 6. Health check behavior

- Script: `/usr/local/bin/znotice-health.sh`
- Schedule: `/etc/cron.d/znotice-health` — every **5 minutes**
- Log: `/var/log/znotice-health.log`

The script checks four things:

1. `systemctl is-active znotice`
2. Public HTTPS via NGINX, pinned to loopback (`curl --resolve znotice.consultbofa.com:443:127.0.0.1`)
3. WhatsApp engine reachability (`http://127.0.0.1:21465/api-docs/`)
4. Available RAM from `/proc/meminfo`

Policy:

- If the **app** is not `active` **or** HTTPS is not `200`, the script **restarts** `znotice`, waits 5s, and logs `RECOVERED` or `STILL-FAILING`.
- If the **engine** is unhealthy it logs `WARN` and **never auto-restarts** (live WhatsApp sessions must not be disrupted).
- If available RAM is below **400 MB**, it logs `WARN low available RAM`.

## 7. TLS certificate renewal

Certificates are managed by certbot with the webroot challenge and the `certbot.timer` systemd timer.

```
systemctl list-timers | grep certbot      # next scheduled run
certbot certificates                      # list certs + expiry
certbot renew --dry-run                   # test renewal
certbot renew                             # renew if due
nginx -t && systemctl reload nginx
```

The NGINX port-80 server serves `/.well-known/acme-challenge/` from `/var/www/html` for the challenge. If a certificate is near expiry, check that port 80 is reachable and the webroot path exists.

## 8. Adding a new payment provider adapter

`server/services/payments/index.js` is a provider registry. To add a gateway:

1. Create an adapter object implementing the interface:
   - `createCheckout({ user, amountCents, credits, meta })` → insert a `pending` row in `transactions` (provider name, amount, credits, meta), return `{ transactionId, reference, instructions, provider }`.
   - `markPaid(transactionId, adminUserId)` → call the shared `settlePaid()` helper (or reuse the same idempotent `UPDATE ... WHERE status='pending'` gate) so credits/orders are granted exactly once.
   - `markFailed(transactionId, adminUserId)` → use `settleFailed()`.
   - `handleWebhook(req)` → if the gateway pushes async notifications, verify the gateway signature and call `settlePaid()`; return `{ ok, error? }`.
2. Register it: `const providers = { manual, yourprovider };`.
3. Store any gateway API keys in `.env` (never in code), and read them in `server/config.js`.
4. If the gateway pushes webhooks, mount a route under `/webhooks` **before** the session/CSRF middleware, and verify the provider signature.
5. Route code needs no changes: `payments.getProvider(name)` and the top-level `markPaid()` / `markFailed()` dispatch by the transaction's `provider` column.

> **IntouchPay (`intouch`) is the first live gateway** following this pattern (Phase 14). It is registered in the same `providers` registry, implements `createCheckout` / `handleWebhook`, and relies on the shared `settlePaid()` helper (with an optional credit-ledger note override) — see `server/services/payments/intouch.js` and the runbook in §16. Unlike `manual`, its webhook is **unsigned**, so `handleWebhook` never trusts the callback and always cross-checks `gettransactionstatus` before settling.
>
> **KitovuPay (`kitovu`) is the second live gateway** (Phase 15). It follows the same pattern (`server/services/payments/kitovu.js`, runbook in §17), but signs its collection requests with an HMAC-SHA256 signature and uses a unique **6-digit externalId**. Its webhook is also **unsigned**, so `handleWebhook` always cross-checks `GET /merchant-transaction/payment/:externalId` before settling.

## 9. Changing settings via admin

Sign in as an admin, then **Admin → Settings**. The editable whitelist is:

| Setting | Type | Default | Effect |
|---|---|---|---|
| `default_country_code` | int (1–9999) | `250` | Country code applied to phone numbers with a leading `0`. |
| `send_rate_per_min` | int (1–100000) | `60` | Per-API-key / per-user send rate limit (also gates the scheduler). |
| `bulk_max_recipients` | int (1–10000) | `1000` | Maximum recipients per bulk job. |
| `schedule_max_days_ahead` | int (1–365) | `90` | Maximum schedule horizon (days) for `scheduled_at`. |
| `scheduled_max_pending` | int (1–100000) | `5000` | Maximum pending scheduled messages per user (enforced atomically). |
| `max_concurrent_sessions` | int (1–100000) | `15` | Capacity guard for pending+connected WhatsApp sessions. |
| `manual_payment_instructions` | text (≤5000) | placeholder | Payment instructions shown to customers; supports `{{reference}}`, `{{amount}}`, `{{credits}}`. |
| `billing_email` | email | `billing@consultbofa.com` | Shown on the payment page. |
| `integration_fee_cents` | int (1–100000000) | `2000` | Integration service fee. |
| `sender_name_price_cents` | int (1–100000000) | `1000` | Dedicated sender-name add-on price in cents per period ($10.00). Written into new purchase transactions; does not change existing ones. |
| `sender_name_days` | int (1–365) | `30` | Dedicated sender-name add-on period in days. Clamped 1–365 again at settlement. |
| `templates_max_per_user` | int (1–10000) | `200` | Maximum saved message templates per user (enforced atomically with a user-row lock). |
| `intouch_rwf_rate` | float (1–100000) | `1477.29` | RWF per USD used at IntouchPay checkout; USD prices are converted to whole RWF at this operator rate. |

| Setting | Type | Default | Effect |
|---|---|---|---|
| `kitovu_rwf_rate` | float (1–100000) | `1477.29` | RWF per USD used at KitovuPay checkout; USD prices are converted to whole RWF at this operator rate. |

Changes are validated, written to the `settings` table, and the in-process cache is refreshed immediately. The cache is also refreshed every 60 seconds at runtime.

## 10. Common failures and fixes

### Engine offline (`503 engine_offline` / sends fail with `whatsapp_engine_unreachable`)

1. `docker ps --filter name=wppconnect` — is the container up?
2. `curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:21465/api-docs/` — expect `200`.
3. Check container logs: `docker logs --tail=100 wppconnect`.
4. If the container is stopped: `docker start wppconnect`. **Do not** restart a running engine unless absolutely necessary (it disconnects all sessions).
5. Confirm the app's `WPPCONNECT_URL` / `WPPCONNECT_SECRET_KEY` in `.env` match the container config.

### QR not appearing

1. Confirm the engine is reachable (above).
2. `GET /dashboard/whatsapp/qr` polls `status-session`; if the status is `pending` and no QR is returned, the app falls back to `qrcode-session`.
3. Check `wa_sessions.status` and `last_qr_at` for the user.
4. Capacity guard: if `409 at_capacity`, see below.
5. If the engine has a stuck session, disconnect and reconnect: `POST /dashboard/whatsapp/disconnect` (or admin force-disconnect).
6. Reconnect attempts call `start-session` with `waitQrCode: true` (45s timeout); transient timeouts fall through to a status probe.

### At capacity (`409 at_capacity`)

1. Count occupying sessions: `SELECT COUNT(*) FROM wa_sessions WHERE status IN ('pending','connected');`
2. Inspect: **Admin** → **Sessions** shows the cap and active count.
3. Stale `pending` rows (users who opened the connect flow but never scanned) hold slots. The **stale-session reaper** (`server/services/reaper.js`) automatically frees abandoned `pending` rows older than 10 minutes (see §10 "Stale sessions not freeing capacity"). You can also force-disconnect them from **Admin** → **Sessions**.
4. If more capacity is genuinely needed, raise `max_concurrent_sessions` in **Admin** → **Settings** — but verify host RAM first (≈300–500 MB per active session; check `free -m`).

### 502 after deploy

1. `systemctl status znotice --no-pager` and `journalctl -u znotice -n 100`.

2. Common causes: syntax error in changed JS, missing dependency (`npm ci --omit=dev`), unapplied migration, or a crash loop.
3. Verify the app listens: `ss -ltnp | grep 5071`.
4. Verify NGINX: `nginx -t`, then `systemctl reload nginx`.
5. Verify locally, bypassing NGINX: `curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:5071/`.

### Webhook secret rotation

`WEBHOOK_SECRET` lives in `.env` and is embedded in the per-session webhook URL path that the engine calls. Rotating it is **not** a single-file change.

1. Generate a new secret and update `WEBHOOK_SECRET` in `.env`.
2. `systemctl restart znotice` so the app accepts the new token.
3. **Existing engine sessions still call the old URL** — the webhook URL is fixed when `start-session` runs. To apply the new secret to a session, force a fresh `start-session` (disconnect + reconnect the tenant), which re-registers the webhook URL.
4. Until a session reconnects, its delivery acks are rejected (`401`); the engine treats a failed webhook as non-fatal, but delivery status will not update for that session.
5. Verify after rotation: trigger a test send and confirm the message status advances (check `messages.status` / engine logs for webhook posts).

> The webhook endpoint also accepts an `x-znotice-webhook-secret` header when present, but the engine cannot send custom headers, so the path token is the operative mechanism.

### Engine close-session 500 quirk

`POST /close-session` can return HTTP 500 (`req.client.close is not a function`) **after** the engine has already marked the session closed. The app handles this by probing `status-session`; if the engine reports `disconnected`, the close is treated as successful. If a disconnect reports failure, verify with the status probe before retrying — do not assume the session is still live.

### Stale sessions not freeing capacity (reaper)

`server/services/reaper.js` runs every **5 minutes** (plus once ~30s after boot) and is started from `server/index.js`. It is internally guarded so it can never crash the app and only logs when it changes something. It:

- marks abandoned `pending` `wa_sessions` rows (untouched >10 min) as `disconnected` unless the engine reports `CONNECTED`, freeing capacity slots;
- marks `connected` rows `disconnected` when the engine reports `CLOSED` (engine restart / phone unlinked). `phone_number` is kept, `connected_at` is cleared. A transient engine outage does **not** drop a `connected` row;
- resets `scheduled_messages` stuck in the scheduler's `sending` claim for more than **10 minutes** (a crash mid-send) to `failed` with `last_error='interrupted_by_restart'`, bumps the parent batch's `failed_count`, and reconciles the batch silently (`sendingInterrupted` in the log summary). It deliberately does **not** retry, so a message that may have reached WhatsApp is never sent twice;
- deletes terminal `scheduled_messages` (`sent` / `failed` / `cancelled`) and terminal `send_batches` (`completed` / `cancelled`) older than **90 days** (by `updated_at`; the batch FK cascade removes any remaining items);

- deletes `webhook_deliveries` older than 30 days;
- deletes expired `sessions` rows and **anonymous** sessions older than **7 days** (`sessionsDeleted` in the log summary). `sessions.expires` is stored by `express-mysql-session` in UNIX **seconds** (not milliseconds), so the prune compares against `UNIX_TIMESTAMP()`. Anonymous means no `$.userId` in the JSON blob (missing/invalid JSON counts as anonymous); authenticated sessions are kept for their full 30-day cookie TTL.

Troubleshooting:

1. Confirm the app started the reaper: `journalctl -u znotice | grep '\[reaper\]'` — a summary line appears only when a pass changed something.
2. Check for stale rows manually:

```
SELECT id,user_id,session_name,status,updated_at FROM wa_sessions
 WHERE status='pending' AND updated_at < DATE_SUB(NOW(), INTERVAL 10 MINUTE);
```

3. If stale rows persist, the engine probe is failing in an unexpected way. Check engine reachability (`curl -s -o /dev/null -w '%{http_code}\n' http://127.0.0.1:21465/api-docs/`) and the app log for `[reaper] ... failed`.
4. Run a pass on demand from a script (there is intentionally **no HTTP trigger**):

```
cd /var/www/znotice.consultbofa.com
node -e "require('./server/services/
reaper').runOnce().then(s=>{console.log(s);process.exit(0)})"
```

5. Force-disconnect a specific row from **Admin** → **Sessions** if needed.

### Outbound webhooks not arriving

1. **Dashboard** → **Webhooks**: is an endpoint saved and **Enabled**? A disabled webhook never enqueues a delivery.
2. Inspect the delivery log table for the tenant:

```
SELECT id,`event`,status,attempts,response_code,last_error,next_attempt_at,created_at
  FROM webhook_deliveries WHERE user_id = ? ORDER BY created_at DESC LIMIT 25;
```

   - `pending` with a future `next_attempt_at`: retry backoff in progress (+1 min, then +5 min).
   - `failed`: 3 attempts exhausted. Check `response_code` / `last_error`.
   - `last_error` mentioning a non-public address: the endpoint URL was rejected by the egress guard (private/loopback/link-local, or a DNS record that now resolves privately). Save a public URL.
3. Verify the receiver returns **2xx** within **10s** and validates the `X-ZNotice-Signature` header. A 4xx/5xx counts as a failed attempt.
4. Confirm the app's webhook worker is running: `journalctl -u znotice | grep '\[webhooks\]'`. Worker errors are logged as `[webhooks] worker error:`.
5. Run the worker once on demand:

```
cd /var/www/znotice.consultbofa.com
node -e "require('./server/services/
webhookOut').runOnce().then(n=>{console.log('processed',n);process.exit(0)})"
```

6. Rotating the signing secret (Dashboard → Webhooks → Regenerate) changes it immediately; update the receiver or its signature checks will fail.
7. `test.ping` from the dashboard is the fastest end-to-end check.

### Scheduled / bulk messages not sending

The scheduler (`server/services/scheduler.js`) runs every **30s** and is kicked immediately after a successful enqueue. Diagnose in this order:

1. Confirm the app is running and the worker started: `journalctl -u znotice | grep '\[scheduler\]'`.
2. Inspect the tenant's open queue (§3 query). Items should be `pending` with a `scheduled_at` in the past and `next_attempt_at` NULL/past.
3. If items sit `pending` with a future `next_attempt_at`, a retry backoff is in progress (+1 min / +5 min) — read `last_error`.
4. Run a tick on demand (safe; it shares the in-flight guard):

```
cd /var/www/znotice.consultbofa.com
node -e "require('./server/services/
scheduler').runOnce().then(n=>{console.log('processed',n);process.exit(0)})"
```

5. A `sending` row older than 10 minutes is reset by the reaper (see below). If it persists, the reaper is not running — check `journalctl -u znotice | grep '\[reaper\]'`.

### `whatsapp_not_connected` on scheduled items

A scheduled item that runs while the tenant's WhatsApp session is not `connected` is **retryable**: `attempts` is incremented and the item is released back to `pending` after +1 min, then +5 min, and marked `failed` after 3 attempts. Reconnect the tenant (§10 "QR not appearing") and the next attempt succeeds; otherwise the item fails with `last_error = whatsapp_not_connected`. No credit is charged for a failed attempt.

### `insufficient_credits` at dispatch

Credits are **not reserved at enqueue**. If the balance is spent before a scheduled/bulk item runs, `executeSend()` returns `402 insufficient_credits`, which is **not** retryable: the item is marked `failed` and the batch `failed_count` increments. No credits are lost (none were charged). Top up the account and re-enqueue the affected recipients.

### `interrupted_by_restart`

`last_error='interrupted_by_restart'` means an item was left in the `sending` claim when the app stopped/crashed and the reaper reset it to `failed`. This is the deliberate **at-most-once** trade-off: the worker cannot know whether the engine accepted the send, so it marks the item failed rather than risk a duplicate. Treat it as "unknown — do not automatically re-send"; re-enqueue manually only after confirming with the recipient.

`queue_full`

`429 queue_full` means the per-user pending cap (`scheduled_max_pending`, default 5,000) was reached at enqueue. Wait for the queue to drain, or raise the cap in **Admin → Settings** (verify the worker can keep up first). The cap is enforced atomically per user, so concurrent bulk requests cannot overrun it.

## At-most-once & scheduler fairness (caveats)

- **At-most-once dispatch:** a crash after the engine accepted a send but before the DB update marks the item `failed` / `interrupted_by_restart` — a rare dropped message, never a duplicate. This is intentional.
- **No reservation:** a scheduled/bulk job can fail item-by-item with `insufficient_credits` if the balance is spent before dispatch (see above).
- **Fairness:** each tick selects at most **25 items per user** and a global `LIMIT 50`. With 3+ tenants each holding a large due backlog, the third tenant may wait for a later tick. This is still an improvement over the previous behaviour (one tenant could consume a whole tick); throughput is bounded by `send_rate_per_min` and host capacity, not by the selection caps.

## Sender-name add-on: approval, verification, comps and expiry

The dedicated sender-name add-on ($10.00 / 30 days by default) gates the WhatsApp sender-name form. Entitlement is `users.sender_name_until` (UTC); the add-on is active while that instant is in the future.

### Approving a purchase / renewal

1. The customer buys from **Dashboard → Billing → Dedicated sender name** and pays offline quoting the `ZN-XXXXXXXX` reference.
2. Approve it at **Admin → Transactions → Mark paid** (the same queue as credit top-ups and the integration fee). Settlement atomically flips the transaction `pending → paid` and extends the entitlement.
3. Verify:

   ```
   SELECT id,email,sender_name_until,
          (sender_name_until IS NOT NULL AND sender_name_until > NOW()) AS active
     FROM users WHERE id = ?;
   ```

   A renewal extends from the previous expiry (`GREATEST(existing, NOW()) + days`), so the new date is the **old date plus the period**, not `NOW()` plus the period. `days` is clamped to 1..365.
4. If settlement fails after the `pending → paid` transition (e.g. a DB error), the transaction is reverted to `pending` (`paid_at = NULL`). It stays in the approval queue and can be marked paid again — no payment is lost and no partial extension remains.

### Comp or extend an entitlement manually (SQL)

Admins cannot edit `sender_name_until` from the UI. To grant or extend it out-of-band (support comp / goodwill), set the expiry directly:

```
-- Grant 30 days from now (or from the current expiry if still active)
UPDATE users
   SET sender_name_until = DATE_ADD(GREATEST(COALESCE(sender_name_until, NOW()), NOW()), INTERVAL
30 DAY)
 WHERE id = ?;
```

```
-- Expire the add-on immediately
UPDATE users SET sender_name_until = NULL WHERE id = ?;
```

`NULL` means "never purchased / not active". Manual changes are not recorded in `transactions` or `credit_ledger`; note the reason in the support ticket.

### Expired add-on behavior

- When `sender_name_until <= NOW()`, the WhatsApp sender-name form is disabled and the card shows a Billing call-to-action. The previously set name **remains on the WhatsApp account** — ZNotice does not revert it.
- Message sending and the `/api/v1/*` API are unaffected: message pricing is unchanged ($10 = 1,000 credits) and attachments/media are **not** gated.
- A `pending` sender-name purchase does not grant access; only admin approval extends the entitlement.

### WhatsApp profile-name caveats

- Setting the sender name changes the **profile/push name on the customer's WhatsApp account** (it also changes on their phone), not just on ZNotice.
- WhatsApp limits how often a profile name can be changed. Repeated changes can fail; the app surfaces a generic error and logs the engine detail as `[whatsapp] setProfileName ... failed`.
- The engine call is made only when the session is `connected` **and** the add-on is active; if the engine is offline the customer is told to try again.

## Admin sender-name override (support tooling)

Admins can view and set/override a tenant's WhatsApp sender (profile) name from **Admin → Sessions** without the tenant owning the paid add-on. This is a deliberate entitlement bypass for support (e.g. correcting a bad name, honouring a comp) — the customer-facing paywall on **Dashboard → WhatsApp** is unchanged.

### Procedure

1. Confirm the tenant's session is **connected** (the Sender name column shows the current `wa_sessions.profile_name`, or `–` if never set here).
2. Enter the new name in the inline form (max 25 chars) and submit. It posts to `POST /admin/sessions/:id/sender-name`.
3. The app validates with the shared `validateSenderName` (1–25 chars; control characters and line breaks rejected; whitespace collapsed), calls the engine `setProfileName`, then persists `wa_sessions.profile_name`.
4. Verify by re-reading the column, or check the engine: `docker logs --tail=50 wppconnect | grep -i username`.

### Caveats

- **Connected session required.** The engine call is made only while the session is `connected`; otherwise the admin gets "Session is not connected — connect it first" and nothing is written.
- **No entitlement needed.** Unlike the customer route, the admin route does **not** require `users.sender_name_until` to be active. This is the only sender-name bypass.
- **Changes the customer's global WhatsApp profile name.** The name is changed on the customer's WhatsApp account (it also changes on their phone), not just on ZNotice; WhatsApp limits how often a profile name can be changed. Repeat changes can fail.

- **No raw engine errors.** Engine detail is logged server-side (`[admin] setProfileName ... failed`) and the admin UI shows a generic message; if the engine is offline the admin is told to try again shortly.
- The action is not recorded in `transactions` or `credit_ledger`; note the reason in the support ticket.

### Message templates: cap reached, duplicate name and prefill

Templates are per-user saved snippets managed at **Dashboard → Templates**; they prefill the WhatsApp and scheduled forms via `?template=<id>`. They are dashboard-only and do not touch API v1 or the send pipeline.

- **"Template limit reached (max N per account)."** The user is at `templates_max_per_user` (default **200**, editable at **Admin → Settings**). Ask them to delete unused templates, or raise the setting if justified. The cap is enforced **atomically** inside a create transaction (user-row lock), so concurrent creates cannot overrun it.
- **"You already have a template with that name."** `message_templates` has `UNIQUE(user_id, name)`. The user must rename or delete the existing template. Names are trimmed and internal whitespace collapsed before the uniqueness check.
- **"Template not found."** The id did not resolve for that user (deleted, invalid, or owned by someone else). Ownership is enforced in the SQL `WHERE user_id = ?`, so foreign ids simply affect 0 rows — there is no cross-tenant read.
- **Prefill shows a blank form.** `resolvePrefill` returns `null` for an invalid/foreign/missing `?template=<id>` (no error, no information leak). `{{name}}` / `{{phone}}` in the prefilled body are resolved at send time by the normal scheduler/bulk pipeline.
- **Bounded list.** The Templates page lists at most **500** templates per user (`listForUser`).

Verify a user's templates directly (optional):

```
SELECT id, name, LEFT(body, 60) AS body_preview, media_url, updated_at
  FROM message_templates
 WHERE user_id = ?
 ORDER BY updated_at DESC, id DESC
 LIMIT 500;

SELECT COUNT(*) AS templates FROM message_templates WHERE user_id = ?;
```

## 11. Usage analytics

Analytics are **read-only** aggregations computed on demand by `server/services/analytics.js` over `messages`, `credit_ledger`, `transactions` and `users`. Nothing is written and there is no background job, so the feature adds no operational moving parts. Surfaces:

- **Customer:** `/dashboard/analytics` — fixed last-30-day window for the signed-in user.
- **Admin:** `/admin/analytics` — fixed last-30-day platform roll-up.
- **API:** `GET /api/v1/usage?days=N` — `days` default `30`, clamped to `1..90`. A missing, non-numeric or out-of-range value is **clamped, not rejected** (clamped again inside the service). Response always has exactly `days` zero-filled UTC day points ending today.

### Sanity-checking the numbers with SQL

All buckets are UTC calendar days and the window starts at `UTC_DATE() - INTERVAL (days-1) DAY` (inclusive). For a 30-day window (`INTERVAL 29 DAY`):

```
-- Messages by current status for one user (customer window)
SELECT status, COUNT(*) AS n
  FROM messages
 WHERE user_id = ? AND created_at >= UTC_DATE() - INTERVAL 29 DAY
 GROUP BY status;

-- Customer totals: total = sent+delivered+read+failed, credits spent over
-- completed statuses only (queued is excluded) — matches /api/v1/usage totals
SELECT COUNT(*) AS total,
       SUM(status IN ('sent','delivered','read')) AS successful,
       COALESCE(SUM(CASE WHEN status IN ('sent','delivered','read','failed')
                         THEN credits_charged ELSE 0 END), 0) AS credits_spent
  FROM messages
 WHERE user_id = ? AND created_at >= UTC_DATE() - INTERVAL 29 DAY;

-- Admin paid revenue / credits sold in the window (from transactions)
SELECT COALESCE(SUM(amount_cents), 0) AS revenue_cents,
       COALESCE(SUM(credits), 0)      AS credits_sold
  FROM transactions
 WHERE status = 'paid' AND paid_at >= UTC_DATE() - INTERVAL 29 DAY;

-- Point-in-time figures (not windowed)
SELECT COALESCE(SUM(delta), 0) AS credits_outstanding FROM credit_ledger;
SELECT COUNT(*) AS pending_payments FROM transactions WHERE status = 'pending';
```

`success_rate` on the API is `(sent + delivered + read) / total`, a `0`–`1` ratio. If a number looks wrong, check the day boundary first: the window uses UTC (`UTC_DATE()`), while `NOW()` on the host may be in a different local timezone.

### Indexes

Migration `server/migrations/007_analytics_indexes.sql` adds `messages(created_at)` (`idx_messages_created`) and `transactions(status, paid_at)` (`idx_tx_status_paid`). It is index-only (no data change) and idempotent (`information_schema`-guarded), so it is safe to re-run by hand. Customer queries additionally use the pre-existing `messages(user_id, created_at)` index. Apply it with `npm run migrate` (§4).

## 12. Email delivery (SMTP) and password reset

### 12.1 SMTP configuration

Outbound email is handled by `server/services/email.js` (nodemailer). The configuration is read from the process environment **at call time**, so a rotated value is picked up on the next send. Set these in `/var/www/znotice.consultbofa.com/.env`:

| Variable | Required | Default | Notes |
|---|---|---|---|
| `SMTP_HOST` | yes (enables email) | — | When unset, `send()` returns `email_not_configured` and logs a one-time warning. |

| Variable | Required | Default | Notes |
|---|---|---|---|
| `SMTP_PORT` | no | `587` | Submission port. |
| `SMTP_SECURE` | no | off | `true` / `1` → implicit TLS (usually port 465). |
| `SMTP_USER` | no | — | Auth user; omit for unauthenticated relays. |
| `SMTP_PASS` | no | — | Auth password. Never commit or paste. |
| `SMTP_FROM` | no | `ZNotice <no-reply@znotice.consultbofa.com>` | Envelope/From header. |

**Status (2026-09-11): configured.** Production reuses the existing Hostinger mailbox for `info@consultbofa.com` (`smtp.hostinger.com:465`, implicit TLS); verified with a live test send. Password-reset and password-changed emails are delivered. If the `SMTP_*` variables are ever removed, `send()` returns `email_not_configured`, the reset UI still shows its generic success page, and the link simply never arrives — restore the variables to re-enable delivery.

Restart note (`.env` is loaded at process start):

```
systemctl restart znotice
systemctl is-active znotice
journalctl -u znotice -n 20 --no-pager | grep -E '\[email\]|listening'
```

The app prints `[znotice] listening on ...` when ready. If `SMTP_HOST` is unset you will see `[email] SMTP_HOST is not configured; outgoing email is disabled` on the first send attempt.

How to verify delivery:

1. Request a reset for a test account from `/forgot-password`.
2. Watch the journal: success produces no `[email]` error line; a failure logs `[email] send failed: <code> response=<n>` (sanitized). There is deliberately no "sent" log line.
3. Check the recipient inbox (and spam). Subject: "Reset your ZNotice password".
4. If nothing arrives, verify DNS/SPF/DKIM for the `SMTP_FROM` domain with your mail provider and that host/port/credentials are correct. Never paste credentials into a ticket.

### 12.2 Password-reset runbook

Flow:

- `GET /forgot-password` renders the request form; `POST /forgot-password` is rate-limited **5 / 15 min / IP** and always redirects to `/forgot-password?sent=1` (existing and non-existing accounts are indistinguishable).
- For an existing active account the app stores `sha256(token)` in `password_resets` with `expires_at = NOW() + 30 minutes` and emails `https://<APP_URL>/reset-password?token=<raw>`.
- `GET /reset-password?token=...` validates the hash, the unused flag and the expiry, then renders the form (invalid/expired → `400`).
- `POST /reset-password` is rate-limited **10 / 15 min / IP**, claims the token atomically, updates the password hash, deletes **every** session for the user, and sends the password-changed email.

Token table:

```
-- Inspect a user's reset tokens (only hashes are stored)
SELECT id, user_id, LEFT(token_hash, 12) AS hash_prefix,
       expires_at, used_at, created_at
  FROM password_resets WHERE user_id = ? ORDER BY created_at DESC;
```

| Field | Meaning |
|---|---|
| `token_hash` | SHA-256 hex of the raw token; the raw value exists only in the emailed link. |
| `expires_at` | UTC expiry, 30 minutes after issue. |
| `used_at` | Set when consumed; a used token can never be replayed. |

**Invalidate a user's tokens** (e.g. suspected mailbox compromise) — deletes every unused token while keeping used rows for audit:

```
DELETE FROM password_resets WHERE user_id = ? AND used_at IS NULL;
```

Requesting a new link does the same automatically. To force a signed-in user out immediately, delete their session rows as well:

```
DELETE FROM sessions
 WHERE data IS NOT NULL AND JSON_VALID(data)
   AND JSON_UNQUOTE(JSON_EXTRACT(data, '$.userId')) = ?;
```

(A successful reset already calls `destroyAllSessions(userId)`; a normal password change keeps the current session and removes the others.)

Cleanup (optional) — expired/used rows are harmless but can be pruned:

```
DELETE FROM password_resets WHERE expires_at < NOW() - INTERVAL 7 DAY;
```

## 13. Test suite

The suite uses Node's built-in `node:test` runner (no extra dependencies).

```
cd /var/www/znotice.consultbofa.com
npm test                 # 164 unit tests, no database
npm run test:integration # 174 DB-backed tests (needs the test DB below)
npm run test:all         # both
```

- **Unit** (`test/unit/`): `send`, `schedule`, `webhook`, `netguard`, `analytics`, `safeNext`, `senderName`, `templates`, `proofs`, `billingAccess`, `intouch`, `kitovu`. These never touch MariaDB.
- **Integration** (`test/integration/`): `credits`, `schedule`, `scheduler`, `passwordReset`, `reaper`, `senderName`, `senderNameAddon`, `templates`, `adminSenderName`, `paymentProofs`, `billingAccess`, `paymentLock`, `intouch`, `kitovu`. They run against an isolated `_test` database and truncate business tables between tests. `test/integration/helpers/db.js` points `UPLOADS_DIR` at a wiped temp directory and truncates `payment_proofs`, so proof tests never touch the production uploads tree.

**One-time test-DB setup** (run as a MariaDB root/admin; substitute the app DB user from `.env`):

```
CREATE DATABASE znotice_test CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
GRANT ALL PRIVILEGES ON znotice_test.* TO 'znotice_user'@'localhost';
FLUSH PRIVILEGES;
```

`test/integration/helpers/db.js` (which must be the first require in every integration file) then refuses to run unless **all** of these hold:

- `NODE_ENV` is not `production`,
- `TEST_DB_NAME` (default `znotice_test`) ends in `_test`,
- `TEST_DB_NAME` differs from the `.env` `DB_NAME`.

It forces `NODE_ENV=test` and `DB_NAME=$TEST_DB_NAME` before `server/config.js` loads, runs the migrations, and holds a MariaDB advisory lock (`GET_LOCK('znotice_test_suite_lock')`) for the lifetime of each file, so `node --test` cannot run two DB-truncating files concurrently.

The tests are mutation-checked: breaking `normalizePhone` (unit) or the per-user pending cap (integration) makes the relevant tests fail.

## 14. Host hardening (2026-09-11)

Applied by `@devops`. The full before/after evidence and rollback commands live in `/root/backups/host-hardening-2026-09-11/CHANGES-AND-ROLLBACK.md` (with `ufw-status-before/after.txt`, `ufw-show-added-after.txt` and the `sudoers.d/` backups).

### 14.1 UFW — removed stale allows

Removed the stale allow rules for **3001/tcp**, **3005/tcp** and **4000/tcp** (v4 + v6). Ports 3005/4000 had no listeners; the GameBet `next-server` on 3001 remains reachable **through NGINX on loopback**, and direct external access is now denied.

```
# Verify
ufw status verbose | grep -E '3001|3005|4000' || echo 'no direct allows (expected)'
curl -sk --resolve gamebet.consultbofa.com:443:127.0.0.1 https://gamebet.consultbofa.com/ -o /dev/
null -w '%{http_code}\n'  # 200 via NGINX
curl -s -o /dev/null -w '%{http_code}\n' http://
127.0.0.1:3001/                                                    # 200 loopback

# Rollback (re-adds v4 + v6 rules)
ufw allow 3001/tcp
ufw allow 3005/tcp
ufw allow 4000/tcp
```

MiroTalk's `40000:40100` rules were not touched.

### 14.2 Sudoers hygiene

Removed the duplicate `kali ALL=(ALL) NOPASSWD:ALL` line from `/etc/sudoers.d/90-cloud-init-users` and deleted the empty-group `/etc/sudoers.d/kali-grant-root`. `kali` retains passwordless sudo and `visudo -c` is clean.

```
visudo -c
runuser -u kali -- sudo -n true && echo 'passwordless sudo intact'
```

```
# Rollback (from the backup dir)
cp -a /root/backups/host-hardening-2026-09-11/sudoers.d/90-cloud-init-users /etc/sudoers.d/90-cloud-init-users
cp -a /root/backups/host-hardening-2026-09-11/sudoers.d/kali-grant-root  /etc/sudoers.d/kali-grant-root
chmod 0440 /etc/sudoers.d/90-cloud-init-users /etc/sudoers.d/kali-grant-root
chown root:root /etc/sudoers.d/90-cloud-init-users /etc/sudoers.d/kali-grant-root
visudo -c
```

### 14.3 NGINX — reset-token access-log suppression

`GET /reset-password?token=<raw>` carries a live token in the query string, so the ZNotice vhost has an exact-match location that disables the access log for that path only (`/forgot-password` carries no token and stays logged):

```
location = /reset-password {
    access_log off;
    proxy_pass http://127.0.0.1:5071;
    # ...same proxy headers/timeouts as "location /"
}
```

```
nginx -t
grep -n 'location = /reset-password' /etc/nginx/sites-available/znotice.consultbofa.com
systemctl reload nginx
```

The pre-existing `location ^~ /webhooks/` also uses `access_log off` because the inbound webhook secret is embedded in that URL path.

## 15. Payment proof storage (Phase 12)

Customer payment proofs are rows in `payment_proofs` (migration `012_payment_proofs.sql`) plus files on disk. They are served only through authenticated Node routes (`uploads/` is outside `public/` and is not exposed by NGINX). Uploads are described architecturally in ARCHITECTURE.md §8(i).

### 15.1 Layout and permissions

```
<UPLOADS_DIR>/payment-proofs/<transaction_id>/<16hex>.<ext>
```

- `UPLOADS_DIR` (env) overrides the default `<app root>/uploads`; it is read by `server/config.js` at boot, so changing it requires a restart.
- Directories are created `0o750`, files written `0o640`, owner `www-data`.
- Stored names are always generated (`<16 hex>.<ext>`); the client filename is display-only and never used as a path.
- If the tree was created by another user, fix ownership:

  ```
  chown -R www-data:www-data /var/www/znotice.consultbofa.com/uploads/payment-proofs
  ```

## 15.2 Caps and rate limits

| Limit | Value | Enforced by |
|---|---|---|
| Single file size | 5 MB | multer `fileSize` + `validateProofBuffer` (magic-byte/size) |
| Proofs per transaction | 5 | `assertProofLimit` under the transaction row lock |
| Proofs per user | 25 | `assertUserProofQuota` |
| Total bytes per user | 50 MB | `assertUserProofQuota` |
| Upload rate | 30 POSTs / 15 min | `proofUploadLimiter`, keyed `user:<id>` else `ip:<ip>` |

Accepted types are JPG/PNG/WebP/PDF, decided by magic bytes only.

## 15.3 Disk-growth monitoring

Only the per-user caps bound growth, so watch the tree and the filesystem:

```
du -sh /var/www/znotice.consultbofa.com/uploads/payment-proofs
du -sh /var/www/znotice.consultbofa.com/uploads/payment-proofs/* | sort -h | tail
find /var/www/znotice.consultbofa.com/uploads/payment-proofs -type f | wc -l
df -h /
```

Largest users and non-pending (audit) proofs:

```
-- per-user proof usage
SELECT t.user_id, COUNT(*) AS proofs, COALESCE(SUM(p.size_bytes), 0) AS bytes
  FROM payment_proofs p JOIN transactions t ON t.id = p.transaction_id
 GROUP BY t.user_id ORDER BY bytes DESC LIMIT 20;

-- proofs whose transaction is no longer pending (kept for audit)
SELECT p.id, p.transaction_id, t.status, p.size_bytes, p.created_at
  FROM payment_proofs p JOIN transactions t ON t.id = p.transaction_id
 WHERE t.status <> 'pending'
 ORDER BY p.created_at;
```

## 15.4 Backup gap — open operational recommendation

> **`uploads/` is NOT included in the automated backup.** The app archive built by `/usr/local/bin/znotice-backup.sh` contains `server/ public/ package.json package-lock.json .env` and explicitly excludes `node_modules/ logs/ uploads/ backups/` (see §5). The daily DB dump contains the `payment_proofs` metadata rows but **not the file bytes**, so a DB-only restore leaves rows that point at missing files and proof serving returns `404`.
>
> **Recommendation (open):** decide the retention/backup policy for proof files. Options: (a) extend the archive to include `uploads/payment-proofs` (with age/size limits, since the archive is pruned to 7 copies); (b) back the directory up separately (e.g. `rsync` or object storage); or (c) treat proofs as short-lived and prune them after settlement plus a retention window. Until one is chosen, proof files are the only business data not covered by the backup.

Related caveats:

- Deleting a transaction cascades its `payment_proofs` **rows** but does not unlink the files. The reaper does not prune proof files. Only the owner delete route ( `POST /dashboard/billing/proof/:id/:proofId/delete` ) unlinks a file. Orphans can accumulate; reconcile the directory against the table before reclaiming space.
- Health checks (§6) and the service restart procedure (§1) are unchanged by this feature.

## 16. IntouchPay operations (Phase 14)

IntouchPay is the live mobile-money gateway (Rwanda MTN MoMo / Airtel Money). Credit top-ups only; the sender-name add-on and the integration service remain manual. Keys are supplied by the operator and live **only in** `.env` — never paste them into tickets, logs, or shell history.

### 16.1 Enabling and mode switching

All gateway configuration is read from `.env`:

| Variable | Purpose |
|---|---|
| `INTOUCH_ENABLED` | Master switch ( `true` / `false` ). When `false` , the Billing card and checkout routes are disabled. |
| `INTOUCH_MODE` | `live` or `sandbox` . Selects the base URL ( `LIVE_BASE_URL` vs `SANDBOX_BASE_URL` ). |
| `INTOUCH_USERNAME` | Operator-issued username (hashed into the auth digest). |
| `INTOUCH_ACCOUNT_NUMBER` | Operator-issued merchant account number (hashed, never sent in cleartext). |
| `INTOUCH_PARTNER_PASSWORD` | Operator-issued partner password (hashed, never sent in cleartext). |
| `INTOUCH_SANDBOX_BASE_URL` | Sandbox base ( `https://developer.intouchpay.co.rw/api/v1/sandbox` ). |
| `INTOUCH_LIVE_BASE_URL` | Live base ( `https://www.intouchpay.co.rw/api` ). |
| `INTOUCH_CALLBACK_URL` | The callback URL IntouchPay posts to ( `https://znotice.consultbofa.com/payments/intouch/webhook` ). |

Changing any of these requires `systemctl restart znotice` (config is read at boot). Before going live, switch `INTOUCH_MODE=sandbox` , exercise a checkout end to end, then set `INTOUCH_MODE=live` and restart.

### 16.2 Callback URL whitelisting (client action)

The provider must whitelist the callback URL `https://znotice.consultbofa.com/payments/intouch/webhook` on the merchant account **before** live callbacks are delivered. This is a client/operator action outside the app — confirm with the operator that the URL is registered, and that the path matches `INTOUCH_CALLBACK_URL` exactly (scheme, host, path).

> The callback is **unsigned** (see SECURITY.md §3.7). The app never trusts the callback on its own — it always cross-checks `gettransactionstatus` before settling, so a callback from an unexpected source cannot trigger settlement.

### 16.3 Verifying the account (`getbalance`)

Confirm the gateway credentials and base URL are correct without touching the dashboard, using a small node one-liner against the service (reads `.env` via config, no secrets on the command line):

```
cd /var/www/znotice.consultbofa.com
node -e "const i=require('./server/services/payments/intouch.js');
i.getBalance().then(b=>console.log(JSON.stringify(b))).catch(e=>{console.error('getbalance
failed:', e.message); process.exit(1);})"
```

A successful call returns the merchant balance; an auth or host error (e.g. `0007 No such user`) means the base URL or credentials are wrong — see §16.7.

### 16.4 The recovery sweep

`server/services/intouchSweep.js` runs every **5 minutes** (started from `server/index.js`, alongside the reaper) and cross-checks pending `intouch` transactions older than **2 minutes** against `gettransactionstatus`. Any that have settled are confirmed automatically. This heals:

- callbacks lost in transit to `/payments/intouch/webhook`, and
- callbacks that arrived **before** the transaction row was fully written.

The sweep shares the same settle path and strict amount/currency matching as the webhook, so a late callback and the sweep cannot double-credit. If a sweep pass errors, it logs and returns — the next 5-minute pass retries; there is no manual trigger.

### 16.5 Manually checking a stuck payment

A pending Intouch transaction that has not confirmed can be checked without waiting:

- **Customer page.** An admin can open the customer's pay page (`/dashboard/billing/pay/<id>`) and use the status action (`POST /dashboard/billing/intouch/:id/status`), which performs a manual `gettransactionstatus` cross-check. A settled result confirms the transaction; the admin table then shows the muted "auto-confirmed by InTouch" hint.
- **Re-request the prompt.** `POST /dashboard/billing/intouch/:id/start` re-sends the USSD prompt if the customer missed it.
- **Wait for the sweep.** If nothing is urgent, the 5-minute sweep (§16.4) picks it up once the transaction is older than 2 minutes.

Never approve an Intouch transaction from the manual Approve/Reject UI — those controls are replaced by the cross-check path for `intouch` rows.

### 16.6 RWF rate maintenance (admin settings)

USD prices are converted to whole RWF at checkout using the admin-editable `intouch_rwf_rate` (see §9). When the operator rate moves:

1. **Admin** → **Settings** → `intouch_rwf_rate`, enter the new RWF-per-USD value.
2. The change is validated (float, 1–100000) and applied immediately.

The rate is written into each transaction's `meta` at checkout, so a later rate change does **not** affect in-flight transactions — only new checkouts.

### 16.7 Common failures

- `0007 No such user` — the request reached the gateway but the credentials are unknown. Check `INTOUCH_MODE` against the intended host (`LIVE_BASE_URL` vs `SANDBOX_BASE_URL`) and confirm `INTOUCH_USERNAME` / `INTOUCH_ACCOUNT_NUMBER` / `INTOUCH_PARTNER_PASSWORD` match what the operator issued. Re-verify with §16.3.
- **Gateway unreachable / timeout** — the transaction stays `pending` and is retriable: the sweep (§16.4) and the manual status check (§16.5) re-issue `gettransactionstatus` on later passes. Fix connectivity (or the base URL) and the next pass confirms the payment.
- **Amount mismatch / currency mismatch** — the cross-check found a settled result whose RWF amount or currency differs from the transaction's `meta`. The transaction is left `pending` / `failed` and the reason is recorded in `meta.failure_reason`. Do not force-approve: contact support with the transaction id, `provider_ref`, and the gateway's returned amount.
- **No refund via the processor** — IntouchPay has no reversal endpoint, so refunds are local-ledger only (an admin credit adjustment), exactly like manual payments (BUSINESS.md §6).

## 17. KitovuPay operations (Phase 15)

KitovuPay is a live mobile-money gateway (Rwanda MTN MoMo). Credit top-ups only; the sender-name add-on and the integration service remain manual. Keys are supplied by the operator and live **only in `.env`** — never paste them into tickets, logs, or shell history. The UAT merchant account is **shared with zeron-xs**, so 6-digit external IDs must be unique across both projects.

### 17.1 Enabling and mode switching

All gateway configuration is read from `.env`:

| Variable | Purpose |
|---|---|
| `KITOVUPAY_ENABLED` | Master switch (`true` / `false`). When `false`, the Billing card and checkout routes are disabled. |
| `KITOVUPAY_MODE` | `live` or `uat`. Selects the base URL (`LIVE_BASE_URL` vs `UAT_BASE_URL`). |
| `KITOVUPAY_API_KEY` | Merchant API key (sent as `x-api-key`). |
| `KITOVUPAY_API_SECRET` | API secret (HMAC key; never sent on the wire). |
| `KITOVUPAY_UAT_BASE_URL` | UAT base (`https://gateway.uat.kitovupay.com`). |
| `KITOVUPAY_LIVE_BASE_URL` | Live base (`https://gateway.kitovupay.com`). |
| `KITOVUPAY_CALLBACK_URL` | The callback URL KitovuPay posts to (`https://znotice.consultbofa.com/payments/kitovu/webhook`). |

Changing any of these requires `systemctl restart znotice` (config is read at boot). Before going live, keep `KITOVUPAY_MODE=uat`, exercise a checkout end to end, then set `KITOVUPAY_MODE=live` and restart. The UAT gateway must allow-list the server IP.

### 17.2 Callback URL whitelisting (client action)

The provider must whitelist the callback URL `https://znotice.consultbofa.com/payments/kitovu/webhook` on the merchant account **before** live callbacks are delivered. Confirm with the operator that the URL is registered and matches `KITOVUPAY_CALLBACK_URL` exactly.

> The callback is **unsigned** (see SECURITY.md §3.7). The app never trusts the callback on its own — it always cross-checks the status endpoint before settling.

### 17.3 The recovery sweep

`server/services/kitovuSweep.js` runs every **5 minutes** (started from `server/index.js`) and cross-checks pending `kitovu` transactions older than **2 minutes** (with `meta.kitovu.requested_at` set) against `GET /merchant-transaction/payment/:externalId`. Any that report `APPROVED` are confirmed automatically. This heals callbacks lost in transit or that arrived before the transaction row was written. It shares the same settle path and strict amount matching as the webhook, so a late callback and the sweep cannot double-credit.

### 17.4 Manually checking a stuck payment

- **Customer page.** Open the customer's pay page (`/dashboard/billing/pay/<id>`) and use `POST /dashboard/billing/kitovu/:id/status` to run a manual status check. The admin table then shows the muted "auto-confirmed by KitovuPay" hint once paid.
- **Re-request the prompt.** `POST /dashboard/billing/kitovu/:id/start` re-sends the collection request (allocating a fresh 6-digit externalId if the previous one is no longer valid).
- **Wait for the sweep.** The 5-minute sweep picks it up once the transaction is older than 2 minutes.

Never approve a Kitovu transaction from the manual Approve/Reject UI — those controls are replaced by the cross-check path for `kitovu` rows.

### 17.5 RWF rate maintenance

USD prices are converted to whole RWF at checkout using the operator rate `kitovu_rwf_rate` (see §9). The rate is written into each transaction's `meta` at checkout, so a later rate change does **not** affect in-flight transactions — only new checkouts.

### 17.6 Common failures

- **`DUPLICATE_EXTERNAL_ID`** — the 6-digit externalId is already in use on the shared merchant account. The app automatically retries with a fresh ref (bounded). Persistent failures mean the 6-digit space is exhausted or a bug; inspect `transactions.provider_ref` and contact support.
- **Gateway unreachable / timeout** — the transaction stays `pending` and is retriable: the sweep (§17.3) and the manual status check (§17.4) re-issue the status lookup on later passes.
- **Amount mismatch** — the webhook reported an amount that differs from the transaction's `meta`. The transaction is left `pending` and the reason is recorded in `meta.failure_reason`. Do not force-approve: contact support with the transaction id, `provider_ref`, and the gateway's returned amount.
- **No refund via the processor** — refunds are local-ledger only (an admin credit adjustment), exactly like manual payments (BUSINESS.md §6).