Z

ZNotice

WHATSAPP MESSAGING GATEWAY

Complete Technical Documentation

Version v0.1.0 (Phase 15 + follow-ups) · 18 September 2026

Prepared for: ZNotice Ltd.
Prepared by: ConsultBofa

https://znotice.consultbofa.com

Confidential — client & operator use. Contains no credentials.

Table of Contents

ZNotice — Complete Technical Documentation 8
- Document control 8
- How to read this set 8
- Support and contacts 9
- Confidentiality note 9

ZNotice — Documentation 10
- Quick facts 10

ZNotice — Business Model & Payments 12
- Onboarding & access policy (Phase 13) 12
- 1. Pricing 12
- 2. Payment workflow (credit top-up) 13
  - IntouchPay instant payment (credit top-up) 14
  - KitovuPay instant payment (credit top-up) 15
- 3. Dedicated sender name add-on ($10.00 / 30 days) 16
- 4. Integration service orders 16
- 5. Editing payment instructions and pricing (admin) 17
- 6. Refunds, cancellations and suspension — policy notes 18
  - Automatic refunds (technical behavior) 18
  - Manual refunds / adjustments 18
  - Cancellations 18
  - Suspension 18
  - Billing contacts 19

ZNotice — Customer API v1 Reference 20
- 1. Authentication 20
- 2. Endpoints 20
  - `GET /api/v1/status` 20
  - `GET /api/v1/messages` 21
  - `GET /api/v1/usage` 21
  - `POST /api/v1/send` 22
- 3. Phone number normalization 26
- 4. Limits 26
- 5. Error codes 27

6. Delivery webhooks (outbound status) ... 28
6.1 Customer outbound webhooks ... 29
6.2 Verifying the signature (Node.js) ... 30
7. Node.js integration example ... 30
Retry guidance ... 31

ZNotice — System Architecture ... 32
1. Overview ... 32
2. Component diagram ... 32
3. Ports and processes ... 34
systemd unit (`/etc/systemd/system/znotice.service`) ... 34
NGINX site (`/etc/nginx/sites-available/znotice.consultbofa.com`) ... 34
4. Database schema ... 35
`users` ... 35
`api_keys` ... 35
`wa_sessions` ... 35
`messages` ... 36
`credit_ledger` — authoritative balance ... 36
`transactions` ... 37
`payment_proofs` (Phase 12, migration `012_payment_proofs.sql`) ... 37
`integration_orders` ... 38
`send_batches` ... 38
`scheduled_messages` ... 39
`message_templates` (Phase 11, migration `011_message_templates.sql`) ... 40
`settings` ... 40
`schema_migrations` ... 40
`sessions` (managed by `express-mysql-session`) ... 41
`password_resets` (Phase 8, migration `008_password_resets.sql`) ... 41
5. Session naming and WhatsApp engine integration ... 41
6. Payment adapter design ... 41
Intouch adapter (Phase 14) — `server/services/payments/intouch.js` ... 42
Intouch webhook — callback / cross-check / ACK ... 43
Intouch sweep — `server/services/intouchSweep.js` ... 43
Kitovu adapter (Phase 15) — `server/services/payments/kitovu.js` ... 43
Kitovu webhook — callback / cross-check / ACK ... 44
Kitovu sweep — `server/services/kitovuSweep.js` ... 44
7. Capacity guard ... 44

8. Core flows ... 45
(a) Customer signup → payment → credit top-up ... 45
(b) QR connect ... 45
(c) API send pipeline with atomic credit charge + refund ... 46
(d) Delivery webhook status updates ... 46
(e) Scheduled & bulk enqueue → dispatch → completion ... 46
(f) Password reset (Phase 8) ... 47
(g) Sender name + dedicated sender-name add-on (Phases 9–11) ... 48
(h) Message templates & dashboard prefill (Phase 11) ... 49
(i) Payment proof uploads (Phase 12) ... 49
(j) Billing lock for unpaid users (Phase 13) ... 51
(k) IntouchPay instant credit top-up (Phase 14) ... 52
(l) KitovuPay instant credit top-up (Phase 15) ... 53
9. Usage analytics (Phase 7) ... 54
Components ... 54
Metric semantics ... 54
Charts are CSP-safe ... 54
Indexes ... 55
Input hardening ... 55
10. Test suite layout ... 55

ZNotice — Security Posture ... 57
1. Authentication and sessions ... 57
1.1 Password reset controls (Phase 8) ... 57
2. CSRF, headers and rate limiting ... 58
2.1 Session and cache hardening (Phase 8) ... 58
3. Admin controls and capacity caps ... 59
3.1 Scheduled & bulk sending controls ... 59
3.2 Usage analytics controls (Phase 7) ... 59
3.3 Dedicated sender-name add-on controls (Phases 9–11) ... 60
3.4 Message template controls (Phase 11) ... 61
3.5 Payment proof upload controls (Phase 12) ... 61
3.6 Billing lock for unpaid accounts (Phase 13) ... 62
3.7 IntouchPay webhook and settlement controls (Phase 14) ... 62
3.8 KitovuPay webhook and settlement controls (Phase 15) ... 63
4. Host and data handling ... 64
4.1 Egress guard (SSRF protection) ... 64
4.2 Host hardening (2026-09-11) ... 65

5. Known risks (explicitly accepted) ... 65
5.1 Unofficial WhatsApp protocol — ban risk ... 65
5.2 Webhook secret is embedded in the URL path ... 65
5.3 In-memory rate limiter resets on restart ... 65
5.4 Stale pending sessions occupy capacity ... 65
5.5 Engine `close-session` 500 bug ... 66
5.6 RAM headroom constraints ... 66
5.7 Other notes ... 66
5.8 No credit reservation at enqueue ... 66
5.9 At-most-once dispatch can drop a message ... 66
5.10 Scheduler fairness under large backlogs ... 66
5.11 Anonymous-session growth — mitigated (Phase 8) ... 66
5.12 L2 — Authenticated responses now send `Cache-Control: no-store` — closed ... 67
5.13 Register user enumeration — open (pre-existing) ... 67
5.14 Forgot-password response timing — open (Low) ... 67
5.15 SMTP credentials — configured (closed 2026-09-11) ... 67
5.16 Inline PDF proofs — polyglot risk (mitigated) ... 67
5.17 Open registration + per-account proof quota (open) ... 67
5.18 IPv6 rate-limit granularity (open) ... 68
5.19 Payment-proof files excluded from backups (open) ... 68
5.20 IntouchPay webhook is unsigned (open, accepted) ... 68
5.21 KitovuPay webhook is unsigned (open, accepted) ... 68
6. Incident response checklist ... 68

ZNotice — Operations Runbook ... 70
1. Service control ... 70
Deploy / restart the app after a code change ... 70
2. Logs ... 71
3. Database access ... 72
4. Migrations workflow ... 72
5. Backup and restore ... 73
Automated backup ... 73
Manual database dump ... 74
Restore — database ... 74
Restore — application files ... 74
6. Health check behavior ... 74
7. TLS certificate renewal ... 75
8. Adding a new payment provider adapter ... 75

9. Changing settings via admin ... 76
10. Common failures and fixes ... 77
Engine offline (`503 engine_offline` / sends fail with `whatsapp_engine_unreachable`) ... 77
QR not appearing ... 77
At capacity (`409 at_capacity`) ... 77
502 after deploy ... 77
Webhook secret rotation ... 78
Engine close-session 500 quirk ... 78
Stale sessions not freeing capacity (reaper) ... 78
Outbound webhooks not arriving ... 79
Scheduled / bulk messages not sending ... 80
`whatsapp_not_connected` on scheduled items ... 80
`insufficient_credits` at dispatch ... 80
`interrupted_by_restart` ... 80
`queue_full` ... 81
At-most-once & scheduler fairness (caveats) ... 81
Sender-name add-on: approval, verification, comps and expiry ... 81
Admin sender-name override (support tooling) ... 82
Message templates: cap reached, duplicate name and prefill ... 83
11. Usage analytics ... 83
Sanity-checking the numbers with SQL ... 84
Indexes ... 84
12. Email delivery (SMTP) and password reset ... 84
12.1 SMTP configuration ... 84
12.2 Password-reset runbook ... 85
13. Test suite ... 86
14. Host hardening (2026-09-11) ... 87
14.1 UFW — removed stale allows ... 87
14.2 Sudoers hygiene ... 87
14.3 NGINX — reset-token access-log suppression ... 88
15. Payment proof storage (Phase 12) ... 88
15.1 Layout and permissions ... 88
15.2 Caps and rate limits ... 89
15.3 Disk-growth monitoring ... 89
15.4 Backup gap — open operational recommendation ... 89
16. IntouchPay operations (Phase 14) ... 90
16.1 Enabling and mode switching ... 90

16.2 Callback URL whitelisting (client action) ... 90
16.3 Verifying the account (`getbalance`) ... 91
16.4 The recovery sweep ... 91
16.5 Manually checking a stuck payment ... 91
16.6 RWF rate maintenance (admin settings) ... 91
16.7 Common failures ... 92
17. KitovuPay operations (Phase 15) ... 92
17.1 Enabling and mode switching ... 92
17.2 Callback URL whitelisting (client action) ... 93
17.3 The recovery sweep ... 93
17.4 Manually checking a stuck payment ... 93
17.5 RWF rate maintenance ... 93
17.6 Common failures ... 93

ZNotice — Changelog ... 94
Phase 14 — IntouchPay mobile money payments (MTN MoMo / Airtel Money) — 2026-09-16 ... 94
Phase 15 — KitovuPay mobile money (MTN MoMo) — 2026-09-18 ... 95
Client-facing PDF documentation — 2026-09-16 ... 96
Register QR share link — 2026-09-14 ... 97
Security hardening — Phase 12/13 follow-up — 2026-09-13 ... 98
Phase 13 — Billing lock for unpaid users + role policy — 2026-09-13 ... 99
Phase 12 — Payment proof uploads — 2026-09-13 ... 100
Phase 11 — Admin sender-name override & message templates — 2026-09-12 ... 101
Phase 10 — Dedicated sender name add-on ($10 / 30 days) — 2026-09-11 ... 102
Phase 9 — WhatsApp sender name — 2026-09-11 ... 103
Phase 8 — Account recovery, hardening & test suite — 2026-09-11 ... 104
Phase 7 — Usage analytics — 2026-09-11 ... 106
Phase 6 — Scheduled & bulk sending — 2026-09-11 ... 107
Phase 5 — Account settings, session reaper & outbound webhooks — 2026-09-11 ... 108
Phase 4 — Operations hardening & documentation — 2026-09-10 ... 109
Phase 3 — Manual payments, integration orders, admin panel, capacity guard — 2026-09-10 ... 109
Phase 2 — QR connect, send pipeline, webhooks, terms & admin basics — 2026-09-09 ... 110
Phase 1 — Scaffold & infrastructure — 2026-09-08 ... 110

ZNotice — Complete Technical Documentation

This set is the complete client-facing documentation for ZNotice, a multi-tenant WhatsApp messaging gateway (SaaS) operated at `https://znotice.consultbofa.com`. It covers the product overview, business and payment model, the customer REST API, system architecture, security posture, operations runbook, and the phase-by-phase build history.

Document control

Field	Value
Title	ZNotice — Complete Technical Documentation
Product	ZNotice (multi-tenant WhatsApp messaging gateway, SaaS)
Version	v0.1.0 (Phase 15, incl. follow-ups through 2026-09-18)
Documentation set date	2026-09-18
Status	Final
Prepared for	ZNotice Ltd.
Prepared by	ConsultBofa
Production domain	https://znotice.consultbofa.com
Confidentiality	Client & operator use; contains no credentials or secrets

How to read this set

The chapters below are presented in the assembly order of this document. The first three are written for a mixed or client-facing audience; the remaining chapters are technical and are intended for developers and operators. Each chapter is self-contained but cross-references the others where useful.

#	Chapter	Source	Audience	What it answers
1	Overview	`README.md`	Client-facing (everyone)	What ZNotice is, its key facts and limits, and how the documentation is organised.
2	Business Model & Payments	`BUSINESS.md`	Client-facing (client/ billing)	How pricing, message credits, add-ons and the manual payment workflow work.
3	Customer API v1	`API.md`	Client-facing (client developers)	Which REST endpoints exist, how to authenticate, limits, webhooks and an integration example.
4	System Architecture	`ARCHITECTURE.md`	Operator-facing (technical)	How the components, database schema, ports and core request/data flows fit together.
5	Security Posture	`SECURITY.md`	Operator-facing (technical/operator)	Which security controls are in place, which risks are accepted, and how to respond to incidents.

#	Chapter	Source	Audience	What it answers
6	Operations Runbook	`OPERATIONS.md`	Operator-facing (internal ops)	How to control services, run backups and migrations, and troubleshoot common failures.
7	Changelog	`CHANGELOG.md`	Operator-facing (technical)	How the system was built, phase by phase, with dates and key changes.

Support and contacts

Support and terms enquiries: **support@consultbofa.com**. Billing and payment enquiries: **billing@consultbofa.com** (editable by an administrator in the application settings).

Confidentiality note

This documentation set was prepared by ConsultBofa for ZNotice Ltd. and its staff. It deliberately contains no passwords, API keys, session secrets, database credentials or other secrets; all such values live only in the application `.env` file or the database and are referred to as "see `.env`" or "admin settings". Please do not redistribute this document outside ZNotice Ltd. without the prior consent of ConsultBofa.

ZNotice — Documentation

ZNotice is a multi-tenant WhatsApp messaging gateway (SaaS) that turns a paired WhatsApp session into a simple REST API. Customers sign up, connect their WhatsApp number by scanning a QR code, buy prepaid message credits, send messages over `POST /api/v1/send` — immediately or scheduled, plus bulk sends via `POST /api/v1/bulk` — and review delivery/credit usage analytics from the dashboard or `GET /api/v1/usage`.

This directory is the canonical technical and operational documentation set. A client-facing downloadable PDF set (a combined volume plus per-chapter PDFs) is also available under `./pdf/` and served at `/docs/pdf/`.

Document	Description
ARCHITECTURE.md	Components, ports, processes, database schema (incl. `password_resets`, `message_templates` and the sender-name columns), session naming, payment adapter design (incl. the sender-name entitlement extension), capacity guard, the scheduler/queue design, the password-reset flow, the sender-name + add-on flow (and the admin override), message templates + `?template=` prefill, the read-only usage-analytics layer (CSP-safe SVG charts), the test-suite layout, and the core request/data flows.
OPERATIONS.md	Runbook: service control, logs, database access, migrations (incl. `011_message_templates.sql`), backup/restore, health checks, TLS renewal, adding payment adapters, admin settings, scheduled-queue operations, sender-name add-on approval/verification/comps and the admin override procedure, message-template troubleshooting, analytics sanity checks, SMTP/email + password-reset procedures, the test suite, host hardening, and common failure fixes.
API.md	Customer REST API v1 reference: authentication, endpoints (immediate, scheduled, bulk, usage analytics, queue/batch management), request/response examples, phone normalization, limits, error codes, webhooks, and a Node.js integration snippet. Message templates are dashboard-only — API v1 is unchanged.
BUSINESS.md	Pricing model (message credits, the dedicated sender-name add-on and the integration service), manual payment workflow, integration service orders, admin-editable payment instructions, and refund/suspension policy notes.
SECURITY.md	Security posture, controls (incl. password-reset, session/cache hardening, the sender-name add-on entitlement gate, the admin sender-name override and the message-template controls), clearly stated known risks, and the incident response checklist.
CHANGELOG.md	Phase-by-phase build history (Phase 1–15 + follow-ups through 2026-09-18) with dates and key files.
PDF set	./pdf/ — client-facing PDF exports (combined + per-chapter) of this documentation set, built from the markdown sources by ./pdf-src/build.sh (pandoc + weasyprint). Served at /docs/pdf/.

Quick facts

Item	Value
Production domain	`https://znotice.consultbofa.com`
Application root	`/var/www/znotice.consultbofa.com`

Item	Value
Runtime	Node.js >= 22, Express 4, EJS, MariaDB-backed sessions
App bind address	`127.0.0.1:5071` (behind NGINX)
WhatsApp engine	WPPConnect Server v2.10.10, container `wppconnect`, `127.0.0.1:21465`
systemd unit	`znotice.service` (user `www-data`)
Database	MariaDB, database `znotice`
Messaging	Immediate, scheduled (up to 90 days ahead) and bulk (up to 1,000 recipients/job)
Pricing	0.01*permessagecredit*(10 = 1,000); $10.00 / 30-day dedicated sender-name add-on (manual renewal); $20 one-time integration service; credit top-ups payable instantly via MTN MoMo / Airtel Money (IntouchPay, RWF at operator rate)
IntouchPay	Live mobile-money gateway (Rwanda MTN MoMo / Airtel Money) for instant credit top-ups. RWF converted at the admin-editable `intouch_rwf_rate` (default 1477.29). Auto-confirmed by cross-check: unsigned webhook `POST /payments/intouch/webhook` + 5-minute sweep. Keys in `.env` (`INTOUCH_*`)
KitovuPay	Live mobile-money gateway (Rwanda MTN MoMo) for instant credit top-ups. RWF converted at the admin-editable `kitovu_rwf_rate` (default 1477.29). Auto-confirmed by cross-check: unsigned webhook `POST /payments/kitovu/webhook` + 5-minute sweep. HMAC-signed collection requests (`x-api-signature`). Keys in `.env` (`KITOVUPAY_*`)
Sender name	Customers: Dashboard → WhatsApp, gated by the paid add-on (`users.sender_name_until`). Admins: override any tenant from **Admin → Sessions** (support tooling; requires a connected session but **not** the add-on). Dashboard-only — **no API v1 changes**. Attachments/media are not gated
Message templates	Dashboard → Templates (create/edit/delete); prefill WhatsApp & scheduled forms via `?template=<id>`. Per-user cap `templates_max_per_user` (default 200, admin-editable). Dashboard-only — **no API v1 changes**
Account recovery	Self-service password reset via email; 30-minute single-use tokens (`password_resets`)
Email (SMTP)	nodemailer; configured via `SMTP_*` in `.env`. **Configured** in production (Hostinger `smtp.hostinger.com:465`, implicit TLS) — password-reset and password-changed emails are delivered (see OPERATIONS.md §12).
Register share link	Scan-to-register QR: short link `/join` (302 → `/register`), share page `/join/qr`, images `/join/qr.png` + `/join/qr.svg`; public, same-origin
PDF docs	/docs/pdf/*.pdf — client download set (combined + per-chapter), rebuilt from docs/ sources via docs/pdf-src/build.sh (see OPERATIONS-style note below)
Tests	`npm test` (164 unit), `npm run test:integration` (174 DB-backed), `npm run test:all`; Node's built-in `node:test`
Operator config	`/var/www/znotice.consultbofa.com/.env` (see `.env`; never committed)

> **Note on secrets:** this documentation deliberately contains **no** credentials. Database passwords, `SESSION_SECRET`, `WEBHOOK_SECRET`, `WPPCONNECT_SECRET_KEY`, API keys and admin passwords live only in the application `.env` or the database, and are referred to as "see `.env`" / "admin settings".

ZNotice — Business Model & Payments

Onboarding & access policy (Phase 13)

- **Existing accounts are admins.** Accounts created before Phase 13 keep the `admin` role and are never billing-locked.
- **New signups are regular users.** Registration hardcodes `role = 'user'` (a `role` field in the signup POST body is ignored), and every new account is **locked to the Billing and Account pages** until an admin confirms the first payment.
- **Locked means Billing + Account (plus the integration request form).** Every other `/dashboard/*` page redirects to the pending payment's pay page (`/dashboard/billing/pay/<id>`), or to `/dashboard/billing` when there is no pending payment, with an error flash explaining the account is not active yet. `/integration` is allowed because the integration request form (shown on the Billing page) creates a pending $20 payment, so it is part of the payment flow.
- **Unlock condition:** at least one `transactions.status = 'paid'` **or** a positive credit balance (`SUM(credit_ledger.delta) > 0`). Pending/failed transactions alone do **not** unlock the account.
- Admins are always unlocked and are exempt from the lock. `/api/v1/*` is unchanged (sends already fail with `402 insufficient_credits` at a zero balance).

1. Pricing

ZNotice is pay-as-you-go. There is no subscription or monthly fee; the optional sender-name add-on is purchased for a fixed period and renewed manually.

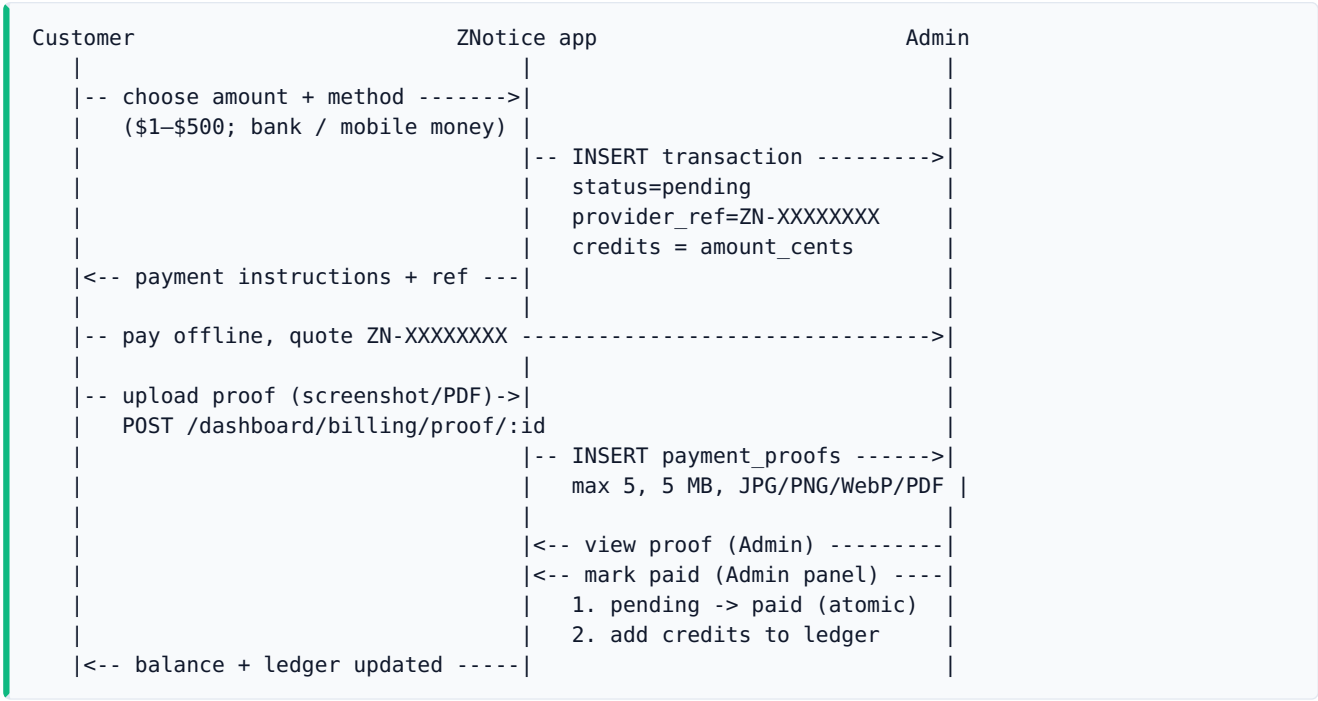
Item	Price	Unit	Notes
Message credits	**$0.01** each	per message	`pricePerCreditCents = 1`
Credit pack	**$10.00 = 1,000 credits**	pack	`pricePer1000CreditsCents = 1000`
Custom top-up	**$1.00 – $500.00**	per checkout	Credits always equal the amount in cents (100–50,000)
Dedicated sender name	**$10.00 / 30 days**	add-on	`sender_name_price_cents = 1000` , `sender_name_days = 30` ; manual renewal
Integration service	**$20.00**	one-time	`integrationFeeCents = 2000`

- Credits are deducted **1 per successfully sent message**. Failed sends are automatically refunded (see §6).
- Credits do not expire and are non-transferable.
- Every movement is recorded in the `credit_ledger` (auditable, running balance). A balance is always `SUM(delta)` for the user.
- The **dedicated sender name** add-on is separate from message credits: it grants the ability to set the WhatsApp profile/push name recipients see. Messages are still billed at `$0.01` each; the add-on does not include credits. Attachments/media are **not** gated by the add-on.

- The integration fee, the per-1000 price and the add-on price/period are stored in the `settings` table and can be changed by an admin (see §5).

2. Payment workflow (credit top-up)

Credit top-ups are paid one of three ways: **manually** — the customer transfers funds offline and an admin approves (the default at launch, described below) — or **instantly** via IntouchPay mobile money (MTN MoMo / Airtel Money) or via KitovuPay mobile money (MTN MoMo), both of which are auto-confirmed with no admin approval and no proof upload (see the IntouchPay and KitovuPay subsections at the end of this section). Every path grants credits exactly once.



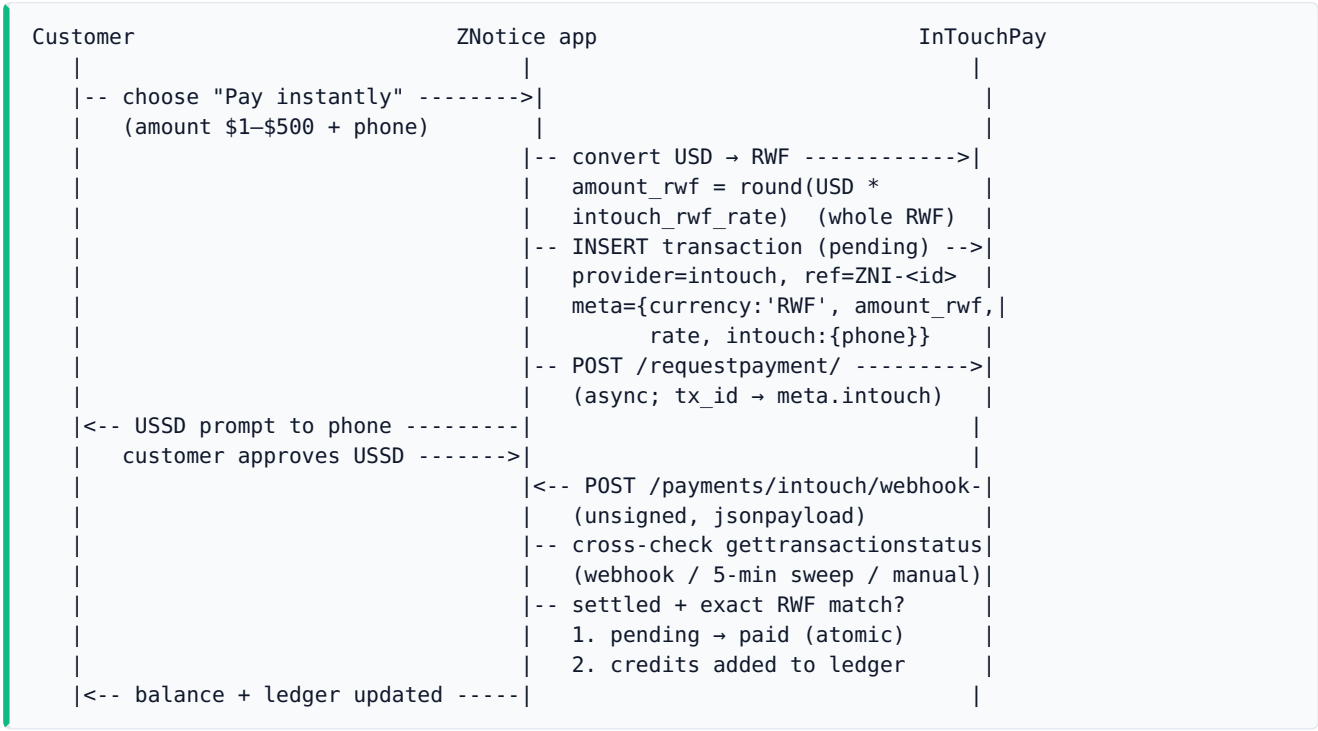
Steps in detail:

1. Customer opens **Dashboard → Billing**, enters an amount (or picks a preset: $10 / $25 / $50 / $100) and a method (bank transfer or mobile money), then continues to payment.
2. The app creates a `pending` transaction with a unique reference `ZN-XXXXXXXX` (8 unambiguous characters; no `0/O/1/I/L`) and shows the admin-configured payment instructions.
3. The customer transfers the exact amount offline, quoting the reference.
4. **The customer uploads proof of payment** on the same pay page (`POST /dashboard/billing/proof/:id`): a screenshot or receipt as **JPG, PNG, WebP or PDF, up to 5 MB**, with up to **5 proofs** per transaction. The file type is verified by magic bytes (the client MIME type and filename are never trusted) and proofs can be added, viewed or deleted only while the transaction is `pending`. Proofs stay readable by the owner and admins after settlement for audit.
5. An admin reviews the uploaded proof(s) from **Admin → Transactions** (the "Proofs (N)" column links each file) and marks the transaction **paid**.
   - The `pending → paid` transition is atomic and idempotent: a double-click cannot double-credit.
   - On success, credits equal to `amount_cents` are added to the ledger (`reason = topup`).
   - If crediting fails, the transaction is rolled back to `pending` for safe retry.
6. The customer's balance and ledger update; they can send immediately.

Customers can cancel a `pending` request themselves from the billing page (`POST /dashboard/billing/cancel/:id`). Pending requests are also visible in the admin overview.

IntouchPay instant payment (credit top-up)

Credit top-ups can instead be paid **instantly** with MTN MoMo or Airtel Money via **IntouchPay** (Rwanda). There is no offline transfer, proof upload or admin approval — the customer approves a USSD prompt on their phone and the app confirms the payment itself by cross-checking the gateway.



Steps in detail:

1. The customer opens **Dashboard → Billing**, picks the "Pay instantly with MTN MoMo / Airtel Money" card, and enters the amount (1–500) and their mobile number.
2. **USD → RWF conversion.** The app converts the USD amount to **whole RWF** at the admin-editable operator rate `intouch_rwf_rate` (float, default `1477.29` RWF per USD; see §5) and shows the RWF amount as a hint. The rate and RWF amount are written into the transaction `meta`; the transaction itself stays in USD (`amount_cents`), so the ledger and analytics remain USD-only.
3. The app registers the payment with IntouchPay (`POST /requestpayment/`, async) and stores the gateway `tx_id`. IntouchPay sends a **USSD prompt** to the customer's phone, which the customer approves to authorize the payment.
4. **Auto-confirmation (no admin, no proof).** The payment is confirmed by **cross-checking the gateway** — never by trusting the callback alone:
   - the **webhook** `POST /payments/intouch/webhook` (unsigned) triggers a `gettransactionstatus` cross-check;
   - a **5-minute sweep** (`server/services/intouchSweep.js`) cross-checks pending Intouch transactions older than 2 minutes and confirms settled ones; and
   - the customer (or an admin) can run a **manual status check** (`POST /dashboard/billing/intouch/:id/status`). Settlement requires the gateway status to be settled **and** the returned RWF amount + currency to match the transaction exactly (whole RWF, off-by-one blocked). Credits

equal to `amount_cents` are then added to the ledger exactly once, using the same atomic `pending → paid` gate as the manual flow.

5. The balance updates immediately; the customer can send right away.

Notes:

- **Credit top-ups only.** The dedicated sender-name add-on (§3) and the integration service (§4) stay on the **manual** workflow — there is no IntouchPay checkout for them.
- **No processor refunds.** IntouchPay exposes no reversal endpoint, so refunds are **local-ledger only** (an admin credit adjustment), exactly like manual payments (see §6).
- A still-pending payment can be re-requested (`POST /dashboard/billing/intouch/:id/start`); the same `POST /dashboard/billing/cancel/:id` route cancels it.

KitovuPay instant payment (credit top-up)

Credit top-ups can also be paid **instantly** with MTN MoMo via **KitovuPay** (Rwanda). There is no offline transfer, proof upload or admin approval — the customer approves the prompt on their phone and the app confirms the payment itself by cross-checking the gateway.

Steps in detail:

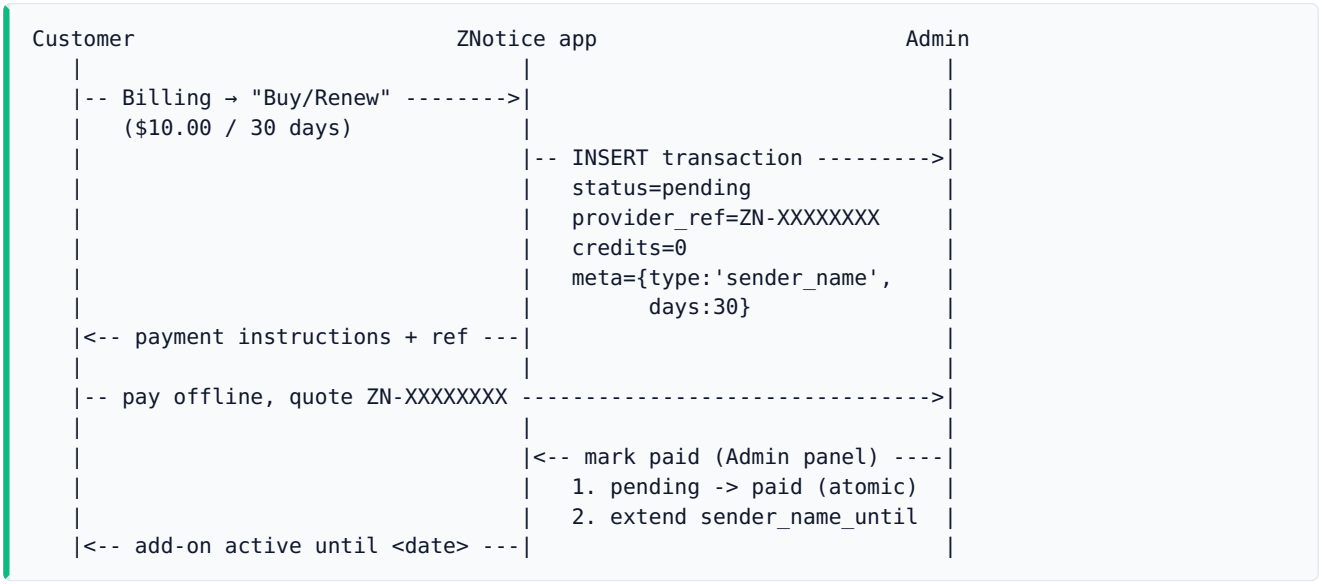
1. The customer opens **Dashboard → Billing**, picks the "Pay instantly with MTN MoMo (KitovuPay)" card, and enters the amount (1–500), their MTN MoMo number, and (optionally) the account name.
2. **USD → RWF conversion.** The app converts the USD amount to **whole RWF** at the admin-editable operator rate `kitovu_rwf_rate` (float, default `1477.29` RWF per USD; see §5) and shows the RWF amount as a hint. The rate and RWF amount are written into the transaction `meta`; the transaction itself stays in USD (`amount_cents`).
3. The app sends a signed collection request (`POST /merchant-transaction/merchant/collection/mobile_money`, HMAC-SHA256 signature, `payment_rail:'mtn'`) with a unique **6-digit `externalId`**, and stores the gateway transaction id. KitovuPay prompts the customer on their phone, which the customer approves to authorize the payment.
4. **Auto-confirmation (no admin, no proof).** The payment is confirmed by **cross-checking the gateway** — never by trusting the callback alone:
   - the **webhook** `POST /payments/kitovu/webhook` (unsigned) triggers a status cross-check;
   - a **5-minute sweep** (`server/services/kitovuSweep.js`) cross-checks pending Kitovu transactions older than 2 minutes and confirms settled ones; and
   - the customer (or an admin) can run a **manual status check** (`POST /dashboard/billing/kitovu/:id/status`). Settlement requires the gateway status to be `APPROVED` (and any webhook amount to match exactly). Credits equal to `amount_cents` are then added to the ledger exactly once, using the same atomic `pending → paid` gate as the manual flow.
5. The balance updates immediately; the customer can send right away.

Notes:

- **Credit top-ups only.** The dedicated sender-name add-on (§3) and the integration service (§4) stay on the **manual** workflow — there is no KitovuPay checkout for them.
- **No processor refunds.** Refunds are **local-ledger only** (an admin credit adjustment), exactly like manual payments (see §6).
- A still-pending payment can be re-requested (`POST /dashboard/billing/kitovu/:id/start`); the same `POST /dashboard/billing/cancel/:id` route cancels it.

3. Dedicated sender name add-on ($10.00 / 30 days)

The optional **dedicated sender name** add-on lets a customer set the WhatsApp profile/push name their recipients see (the Phase 9 feature). It is sold for a fixed period and uses the **same manual payment + admin approval** flow as a credit top-up.



Key behavior:

- **Price / period are server-side settings.** The customer cannot influence them: the dashboard route reads `sender_name_price_cents` (default `1000`) and `sender_name_days` (default `30`) and writes them into the transaction `meta`; `meta.days` is clamped to **1..365** at settlement.
- **Entitlement.** The paid period is stored as `users.sender_name_until` (UTC). The add-on is active while that instant is in the future. The WhatsApp sender-name form is gated on an active entitlement; while inactive or expired the card shows a Billing call-to-action and the form is disabled.
- **Renewal extends, never resets.** Settlement sets `sender_name_until = GREATEST(existing expiry, NOW()) + days`, so buying a renewal before expiry **adds** the new period to the remaining paid time and no paid time is lost.
- **One pending purchase at a time.** The checkout route locks the user row and re-checks for an existing pending `sender_name` transaction before inserting, so concurrent clicks cannot create multiple pending purchases.
- **No credits are involved.** The transaction carries `credits = 0`; approving it extends the entitlement instead of adding credits.
- **Dashboard-only.** There is no API v1 endpoint for the add-on and no change to message pricing ($10 = 1,000 credits).

If an admin approves a payment and the entitlement update fails, the transaction is rolled back to `pending` so it can be retried without losing the payment (see §2 for the same compensation on credit top-ups).

4. Integration service orders

The optional **$20 one-time integration service** is a done-for-you setup: endpoint wiring, webhooks/ delivery handling, test messages and handover.

Flow:

```
Customer submits requirements + contact channel + optional preferred stack
        |
        v
integration_orders row: status = requested
        |
        +--> pending $20 transaction (provider_ref ZN-XXXXXXXX, credits = 0)
        |
        v
Admin approves the $20 payment  --> order status = paid
        |
        v
Admin works the order            --> status = in_progress
        |
        v
Admin marks complete             --> status = delivered
```

Order status	Meaning
`requested`	Customer submitted the request; $20 payment pending.
`paid`	Payment approved; work has not started.
`in_progress`	Admin/team is building the integration.
`delivered`	Work completed and handed over.
`cancelled`	Order cancelled (set by an admin).

- The `$20` service fee is **not** a credit purchase: its transaction carries `credits = 0` and settling it advances the linked order instead of adding credits.
- Order status and `admin_notes` are managed at **Admin → Orders**.
- Customers can track order progress and the linked reference on the Billing page.

5. Editing payment instructions and pricing (admin)

Sign in as an admin and go to **Admin → Settings**. Relevant fields:

Setting	Purpose
`manual_payment_instructions`	Text shown to customers on the payment page. Supports `{{reference}}`, `{{amount}}` and `{{credits}}` placeholders.
`billing_email`	Shown on the payment page for questions.
`integration_fee_cents`	One-time integration fee in cents (default `2000` = $20.00).
`sender_name_price_cents`	Dedicated sender-name add-on price in cents per period (default `1000` = $10.00).
`sender_name_days`	Dedicated sender-name add-on period in days (default `30`; clamped 1–365 at settlement).
`intouch_rwf_rate`	RWF per USD used to convert credit top-ups to IntouchPay mobile money at checkout (float, default `1477.29`).

Setting	Purpose
`kitovu_rwf_rate`	RWF per USD used to convert credit top-ups to KitovuPay mobile money at checkout (float, default `1477.29`).
`default_country_code`	Country code used to normalize `0`-prefixed phone numbers.
`send_rate_per_min`	Per-key send rate limit.
`max_concurrent_sessions`	Gateway capacity guard.

Example `manual_payment_instructions`:

```
Bank transfer: Example Bank, Acct 1234567890, Name ZNotice Ltd.
Mobile money: +250 7XX XXX XXX (ZNotice).
Amount: ${{amount}} for {{credits}} credits. Reference: {{reference}}
Send proof to billing@consultbofa.com quoting {{reference}}.
```

Changes are validated and applied immediately (the settings cache refreshes).

6. Refunds, cancellations and suspension — policy notes

Automatic refunds (technical behavior)

- If the WhatsApp engine rejects a send, the app stores a `failed` message and writes a `+1` refund entry to the ledger (`reason = refund`, linked to the failed message). The customer is not charged for failed sends.

Manual refunds / adjustments

- Admins can adjust any balance at **Admin → Users → Credits** (positive to grant, negative to deduct). A deduction that would make the balance negative is rejected. These are recorded as `admin_adjust`.
- The `transactions.status` enum includes `refunded`, but there is currently **no UI or route that sets it** — manual monetary refunds are handled out-of-band and recorded via an admin credit adjustment. (See the discrepancy note in the project report.)

Cancellations

- A customer may cancel their own `pending` top-up; it becomes `failed`. The same cancel route applies to a pending sender-name purchase.
- Integration orders can be set to `cancelled` by an admin.
- Cancelling a pending sender-name purchase does not remove an already-active entitlement; it only discards the unpaid renewal.

Suspension

- Admins can suspend/activate users at **Admin → Users**. Suspended users cannot send (`403 account_suspended`) and API keys are rejected.
- Per the Terms of Service, ZNotice may rate-limit, suspend or terminate accounts that violate the Acceptable Use Policy, evade limits, or pose a risk to the platform. Credits are non-transferable.

- Because ZNotice uses an **unofficial WhatsApp automation client**, a number may be rate-limited, restricted or banned by WhatsApp. This risk is disclosed in `/terms` and is borne by the customer.

Billing contacts

- Default billing email: `billing@consultbofa.com` (editable in settings).
- Support / terms contact: `support@consultbofa.com`.

ZNotice — Customer API v1 Reference

Base URL: `https://znotice.consultbofa.com`

The v1 API is a JSON REST interface for sending WhatsApp messages and reading account/message state. All endpoints are under `/api/v1` and are authenticated with an API key. The API is CSRF-exempt (it is not a browser surface).

> **Status:** v1 preview. Live endpoints: `GET /status`, `GET /messages`, `GET /usage`, `POST /send` (immediate or scheduled), `POST /bulk`, and the scheduled-queue and batch management endpoints (`GET`/`DELETE /scheduled`, `GET`/`DELETE /batches`).

> **Message templates are dashboard-only.** Saved message templates (**Dashboard → Templates**, per-user cap `templates_max_per_user`) only prefill the dashboard WhatsApp/scheduled forms. There are **no** template endpoints in API v1, and API v1 behaviour is unchanged by templates. `{{name}}` / `{{phone}}` placeholders in a prefilled message are resolved at send time by the same scheduler/bulk pipeline documented below.

1. Authentication

Create an API key in **Dashboard → API Keys**. The full key is shown exactly once at creation and is stored only as a SHA-256 hash. Send it on every request using **either** header:

```
Authorization: Bearer zn_live_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
X-API-Key: zn_live_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Key format: `zn_live_` followed by 32 lowercase hex characters. Keys can be revoked at any time from the dashboard; revoked keys immediately return `401 revoked_api_key`.

2. Endpoints

`GET /api/v1/status`

Returns account, credit balance, WhatsApp connection state and the authenticated key.

```
curl https://znotice.consultbofa.com/api/v1/status \
  -H "Authorization: Bearer $ZNOTICE_API_KEY"
```

```
{
  "ok": true,
  "api_version": "v1",
  "user": { "id": 1, "name": "Jane", "email": "jane@example.com" },
  "credits": 1000,
```

```
  "whatsapp": { "session": "zn_1", "status": "disconnected", "phone_number": null },
  "api_keys_active": 1,
  "key": { "label": "Default", "prefix": "zn_live_abcd" }
}
```

`whatsapp.status` is one of `disconnected`, `pending`, `connected`, `error`. `whatsapp` is always present; `session` / `phone_number` are `null` before a session row exists.

`GET /api/v1/messages`

Returns recent messages for the authenticated user, newest first.

Query param	Type	Default	Notes
`limit`	int	`50`	Clamped to `1` – `200`.

```
curl "https://znotice.consultbofa.com/api/v1/messages?limit=20" \
  -H "X-API-Key: $ZNOTICE_API_KEY"
```

```
{
  "ok": true,
  "count": 2,
  "data": [
    {
      "id": 42,
      "to_number": "250788123456",
      "body": "Hello from ZNotice",
      "media_url": null,
      "status": "sent",
      "wpp_message_id": "true_250788123456@c.us_3EB0...",
      "credits_charged": 1,
      "created_at": "2026-09-10T12:00:00.000Z"
    }
  ]
}
```

`GET /api/v1/usage`

Returns per-day message activity and delivery totals for the authenticated user, plus a roll-up, over the last `days` UTC calendar days. **Read-only** — the endpoint never writes to the database.

Query param	Type	Default	Notes
`days`	int	`30`	Window length. Clamped to `1` – `90`; a missing, non-numeric or out-of-range value is clamped, never an error (`0` → 1 day, `500` → 90 days).

```
curl "https://znotice.consultbofa.com/api/v1/usage?days=3" \
  -H "Authorization: Bearer $ZNOTICE_API_KEY"
```

```
{
  "ok": true,
  "days": 3,
  "data": [
```

```
    { "date": "2026-09-09", "sent": 10, "delivered": 5, "read": 3, "failed": 1, "total": 19,
"credits_spent": 18 },
    { "date": "2026-09-10", "sent": 0, "delivered": 0, "read": 0, "failed": 0, "total": 0,
"credits_spent": 0 },
    { "date": "2026-09-11", "sent": 4, "delivered": 2, "read": 1, "failed": 0, "total": 7,
"credits_spent": 7 }
  ],
  "totals": {
    "sent": 14,
    "delivered": 7,
    "read": 4,
    "failed": 1,
    "total": 26,
    "credits_spent": 25,
    "success_rate": 0.9615
  }
}
```

Semantics:

- **UTC day buckets.** `data` always contains exactly `days` points, oldest first, ending on today. The window starts at `UTC_DATE() - INTERVAL (days-1) DAY` **inclusive**, so it includes today. Days with no activity are zero-filled.
- **Completed statuses only.** `total` is `sent + delivered + read + failed`. `queued` messages are excluded from the counts. `failed` counts messages currently in status `failed`.
- **`success_rate`** is `(sent + delivered + read) / total`, a `0`–`1` ratio (`0` when `total` is `0`).
- **`credits_spent`** is `SUM(credits_charged)` over completed statuses only, so a `queued` message never contributes.
- `data[].date` is a `YYYY-MM-DD` UTC calendar date.

`days` is clamped at both the route and inside the analytics service. The global 120 requests/minute/IP limit applies as for every `/api/v1` endpoint.

`POST /api/v1/send`

Sends a WhatsApp text or media message. Costs **1 credit** per message and requires the account's WhatsApp session to be `connected`.

Request body (JSON):

Field	Type	Required	Notes
`phone`	string	yes	Recipient number. See §3 for normalization.
`message`	string	yes*	1–4096 characters. May be omitted if `media_url` is set.
`media_url`	string	no	Public `http(s)` URL (ports 80/443 only). ZNotice fetches the file server-side and `message` becomes the caption. See §2.1 for limits and the egress guard. `mediaUrl` is also accepted.
`scheduled_at`	string	no	Defer delivery to the scheduler queue. ISO-8601 with `Z` /offset, or naive `YYYY-MM-DDTHH:MM` / `YYYY-MM-DD HH:MM` interpreted as **UTC**. Must be ≥ now+60s and ≤ now+ `schedule_max_days_ahead` (default 90) days. Omitted/blank = immediate send (see §2.2).

* At least one of `message` or `media_url` is required.

2.1 `media_url` fetching & limits

`media_url` is fetched by ZNotice (never by the WhatsApp engine) behind an egress guard, then handed to the engine as base64:

- **Schemes:** `http` and `https`. **Ports:** default only (`80` / `443`); URLs with credentials (`http://user:pass@…`) are rejected.
- **Private-address blocking:** the hostname is resolved and **every** returned address must be public. Loopback (`127.0.0.0/8`, `::1`), private (`10/8`, `172.16/12`, `192.168/16`), link-local (`169.254/16`, `fe80::/10`), CGNAT (`100.64/10`), ULA (`fc00::/7`), multicast, reserved and IPv6-mapped forms of all of these are rejected with `400 invalid_media_url`. The validated IP is pinned for the connection so DNS rebinding cannot bypass the check.
- **Redirects:** followed manually, at most **3** hops, and every hop is re-validated with the same rules.
- **Limits: 30s** total timeout and **16MB** maximum size.
- **Content types:** `image/*`, `video/*`, `audio/*` and `application/pdf`. **SVG is excluded:** `image/svg+xml`, `image/svg` and any `+xml` content type (e.g. `application/xhtml+xml`) are rejected because they can carry active content. Anything else is also rejected.

```
curl -X POST https://znotice.consultbofa.com/api/v1/send \
  -H "Authorization: Bearer $ZNOTICE_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"phone":"250788123456","message":"Hello from ZNotice"}'
```

Success (`200`):

```
{
  "ok": true,
  "id": 42,
  "message_id": "true_250788123456@c.us_3EB0...",
  "credits_remaining": 999
}
```

- `id` is the local `messages.id`.
- `message_id` is the engine/WhatsApp message id, used to correlate delivery webhooks.
- `credits_remaining` is the balance after the atomic charge.

Media example:

```
curl -X POST https://znotice.consultbofa.com/api/v1/send \
  -H "Authorization: Bearer $ZNOTICE_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"phone":"0788123456","media_url":"https://example.com/receipt.pdf","message":"Your
receipt"}'
```

If the engine rejects the send, the credit is **automatically refunded** and a `failed` message row is stored; the response is `502 send_failed` with a short sanitized `detail` (no stack traces).

2.2 Scheduled sends

When `scheduled_at` is present and non-empty, `POST /api/v1/send` **enqueues** the message instead of sending it immediately and returns `202`:

```
{
  "ok": true,
  "scheduled_id": 17,
  "scheduled_at": "2026-09-12T09:30:00.000Z",
  "credits_balance": 999
}
```

- **No credit is charged at enqueue.** The balance is only pre-checked (balance ≥ 1) as a fast-fail guard; the scheduler charges **1 credit per message at actual send time** and auto-refunds on engine failure, so a job that never sends costs nothing.
- **Enqueue prechecks:** account `active` (`403 account_suspended`), WhatsApp session `connected` (`409 whatsapp_not_connected`), balance ≥ 1 (`402 insufficient_credits`), and the per-user pending cap not exceeded (`429 queue_full`).
- `scheduled_at` accepts an ISO-8601 string with a trailing `Z` or `±HH:MM` offset, **or** a naive `YYYY-MM-DDTHH:MM` / `YYYY-MM-DD HH:MM` (seconds optional) which is interpreted as UTC. An unparseable value, a time earlier than now+60s, or a time beyond the horizon returns `400 invalid_scheduled_at`.
- An omitted/blank `scheduled_at` keeps the existing immediate `200` behavior.
- Track and cancel enqueued messages with the queue endpoints in §2.4.

2.3 Bulk sends — `POST /api/v1/bulk`

Enqueues one message per recipient as a single **batch**. Every recipient is normalized (see §3) and duplicates are removed. A bulk job is created whether or not `scheduled_at` is set; omitted/blank sends on the next worker tick.

Request body (JSON):

Field	Type	Required	Notes
`recipients`	array	yes	1–`bulk_max_recipients` (default 1,000) entries. Each entry is either `{ "to": "...", "name": "..." }` or a bare phone string.
`message`	string	yes*	Template, 1–4,096 characters **after** placeholder resolution. May be omitted if `media_url` is set.
`media_url`	string	no	Same rules as §2.1; the resolved `message` becomes the caption.
`scheduled_at`	string	no	Same semantics as §2.2 (defaults to immediate).
`label`	string	no	≤120-character batch label shown in the dashboard/API.

* At least one of `message` or `media_url` is required.

Template placeholders are resolved **at enqueue time** per recipient: `{{name}}` → the recipient's `name` (empty if omitted) and `{{phone}}` → the normalized phone. Any other `{{token}}` is left untouched.

```
curl -X POST https://znotice.consultbofa.com/api/v1/bulk \
  -H "Authorization: Bearer $ZNOTICE_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "label": "September promo",
    "message": "Hi {{name}}, your order is ready.",
    "scheduled_at": "2026-09-12T09:30",
    "recipients": [
      { "to": "0788123456", "name": "Alice" },
      { "to": "+250788654321" }
```

```
    ]
  }'
```

Success (`202`):

```
{
  "ok": true,
  "batch_id": 8,
  "total": 2,
  "duplicates_removed": 0,
  "scheduled_at": "2026-09-12T09:30:00.000Z",
  "credits_balance": 1000
}
```

- `total` is the number of items actually enqueued (after dedupe).
- `duplicates_removed` counts recipients dropped because their normalized phone already appeared earlier in the list.
- If **any** recipient is invalid the whole request is rejected with `400 invalid_recipients` and up to 10 `{ "line", "value" }` entries (`line` is the 1-based position in `recipients`). No partial batch is created.
- More than `bulk_max_recipients` returns `400 too_many_recipients`.
- The enqueue prechecks match §2.2, except the balance check is balance ≥ `total`.

2.4 Queue management

All queue endpoints are scoped to the authenticated user; another user's ids return `404 not_found`.

Method & path	Purpose
`GET /api/v1/scheduled`	List scheduled items. Defaults to the open queue (`pending` + `sending`); `?status=` filters to one of `pending`, `sending`, `sent`, `failed`, `cancelled` (anything else → `400 invalid_status`). `?limit=` 1–200 (default 50). Newest first.
`GET /api/v1/scheduled/:id`	Fetch one item, including `body`, `media_url`, `next_attempt_at`, `attempts` and `last_error`.
`DELETE /api/v1/scheduled/:id`	Cancel one **pending** item. Returns `200 {ok, id, status:"cancelled"}`; if it is already `sending`/terminal → `409 not_pending`.
`GET /api/v1/batches`	List batches, newest first. `?limit=` 1–100 (default 20).
`GET /api/v1/batches/:id`	Fetch a batch plus a page of its items (ordered by id). `?limit=` 1–500 (default 200).
`DELETE /api/v1/batches/:id`	Cancel a `scheduled`/`processing` batch and all its remaining `pending` items. Returns `200 {ok, batch_id, cancelled}`; terminal batches → `409 not_cancellable`.

```
# Open queue
curl "https://znotice.consultbofa.com/api/v1/scheduled?status=pending&limit=100" \
  -H "X-API-Key: $ZNOTICE_API_KEY"

# Cancel a batch's remaining items
curl -X DELETE https://znotice.consultbofa.com/api/v1/batches/8 \
  -H "X-API-Key: $ZNOTICE_API_KEY"
```

An item returned as `sending` is already claimed by a worker and can no longer be cancelled — this is what makes dispatch **at-most-once**. A `pending` item is moved to `sending` immediately before the engine call; a crash mid-send leaves it `sending` until the reaper marks it `failed` with `last_error = interrupted_by_restart` (see ARCHITECTURE.md §8e and OPERATIONS.md §10).

3. Phone number normalization

`phone` is normalized server-side using `default_country_code` (admin setting, default `250`):

1. Spaces, dashes, parentheses and a leading `+` are stripped.
2. A leading international `00` prefix is removed.
3. If the number still starts with a national trunk `0`, the `0` is replaced by the configured country code. Example for `250`: `0788123456` → `250788123456`.
4. The result must be **8–15 digits**, otherwise `400 invalid_phone`.

Input	Normalized (cc=250)
`0788123456`	`250788123456`
`+250 788 123 456`	`250788123456`
`00250788123456`	`250788123456`
`250788123456`	`250788123456`
`123`	rejected → `invalid_phone`

4. Limits

Scope	Limit	Enforcement
Usage window (`?days=`)	1–90 days (default 30)	Clamped silently (never `400`)
Message length	4,096 characters	`400 message_too_long`
Media URL length	500 characters	`400 invalid_media_url`
Media size	16 MB	`400 invalid_media_url`
Media fetch timeout	30 s (all redirect hops)	`400 invalid_media_url`
Media redirects	3 hops, each re-validated	`400 invalid_media_url`
Bulk recipients	`bulk_max_recipients` (default 1,000 / job)	`400 too_many_recipients`
Schedule horizon	now+60 s … `schedule_max_days_ahead` (default 90 days)	`400 invalid_scheduled_at`
Pending scheduled queue	`scheduled_max_pending` (default 5,000 / user)	`429 queue_full`
Sends per API key	`send_rate_per_min` (default 60 / minute)	`429 rate_limited` + `retry_after`
All API requests	120 / minute / IP	`429 rate_limited` (global, no `retry_after`)

Scope	Limit	Enforcement
Concurrent WhatsApp sessions	`max_concurrent_sessions` (default 15)	Dashboard connect only → `409 at_capacity`

The per-key send limiter is an **in-memory fixed window** and resets when the application restarts. The global IP limiter is `express-rate-limit`.

5. Error codes

All API errors return JSON with `ok: false` and a machine-readable `error`. The global 404 handler returns `{ "ok": false, "error": "not_found" }` for unknown `/api/*` paths.

HTTP	`error`	Shape / extra fields	Meaning
400	`phone_required`	`{ ok:false, error }`	`phone` was empty.
400	`invalid_phone`	`{ ok:false, error }`	Not a valid 8–15 digit number after normalization.
400	`message_required`	`{ ok:false, error, message }`	Neither `message` nor `media_url` provided.
400	`message_too_long`	`{ ok:false, error, message }`	Message exceeds 4,096 characters.
400	`invalid_media_url`	`{ ok:false, error, message }`	`media_url` failed the egress guard: not `http(s)`, non-default port, credentials, non-public target, too many redirects, timeout, >16MB, a disallowed content type, or longer than 500 characters.
400	`invalid_scheduled_at`	`{ ok:false, error, message }`	`scheduled_at` is unparseable, earlier than now+60s, or beyond the schedule horizon.
400	`recipients_required`	`{ ok:false, error, message }`	Bulk request has no (or an empty) `recipients` array.
400	`too_many_recipients`	`{ ok:false, error, message }`	More than `bulk_max_recipients` in one bulk job.
400	`invalid_recipients`	`{ ok:false, error, invalid: [{line,value}] }`	One or more recipients failed normalization; up to 10 entries are returned.
400	`invalid_status`	`{ ok:false, error, message }`	`?status=` on `GET /scheduled` is not an allowed value.
400	`invalid_json`	`{ ok:false, error, message }`	Request body is not valid JSON.
413	`payload_too_large`	`{ ok:false, error, message }`	Request body exceeds the 1MB limit.
401	`missing_api_key`	`{ ok:false, error, message }`	No key in `Authorization` / `X-API-Key`.
401	`invalid_api_key`	`{ ok:false, error, message }`	Key hash not found.

HTTP	`error`	Shape / extra fields	Meaning
401	`revoked_api_key`	`{ ok:false, error, message }`	Key exists but was revoked.
402	`insufficient_credits`	`{ ok:false, error, balance }`	Immediate: balance < 1. Bulk enqueue: balance < recipient count. A scheduled item can also fail with this at dispatch if the balance was spent before it ran (no credits are lost).
403	`account_suspended`	`{ ok:false, error }`	User status is not `active`.
404	`user_not_found`	`{ ok:false, error }`	User record no longer exists.
404	`not_found`	`{ ok:false, error, message }`	Unknown `/api/*` endpoint, or a scheduled item/batch that does not exist or belongs to another user.
409	`whatsapp_not_connected`	`{ ok:false, error }`	No `connected` WhatsApp session.
409	`not_pending`	`{ ok:false, error, message }`	Cancel target is not `pending` (it may already be sending).
409	`not_cancellable`	`{ ok:false, error }`	Batch is already terminal (`completed` / `cancelled`).
429	`queue_full`	`{ ok:false, error, message }`	Per-user pending scheduled queue cap (`scheduled_max_pending`) reached.
429	`rate_limited` (send)	`{ ok:false, error, message, retry_after }`	Per-key send limit exceeded.
429	`rate_limited` (global)	`{ ok:false, error, message }`	120 req/min/IP exceeded.
500	`server_error`	`{ ok:false, error, message }`	Unexpected server error.
502	`send_failed`	`{ ok:false, error, detail }`	Engine rejected the send; credit refunded.

6. Delivery webhooks (outbound status)

ZNotice receives delivery updates from the WhatsApp engine and records them against your message. The status lifecycle is:

Status	Trigger (engine ack)	Meaning
`sent`	ack `1`	Accepted by WhatsApp servers.
`delivered`	ack `2`	Delivered to the recipient's device.
`read`	ack `3` or `4`	Read / played by the recipient.
`failed`	negative ack (`-1` … `-7`)	Send failed.

Statuses are **monotonic** — a later webhook never downgrades a status (for example, `read` is not overwritten by `delivered`). `queued` is the initial database default; successful API sends are stored as `sent` immediately.

To track delivery you can either poll `GET /api/v1/messages` and match on `wpp_message_id`, or subscribe to customer outbound webhooks (§6.1).

6.1 Customer outbound webhooks

Configure an endpoint from **Dashboard → Webhooks**. ZNotice POSTs a JSON body to your URL whenever one of your messages changes state. Delivery is best-effort with retries and is **disabled by default** until you save a URL.

Events

Event	Fired when	Payload `data` fields
`message.sent`	The engine accepted the send.	`message_id`, `to_number`, `status`, `wpp_message_id`, `credits_charged`
`message.failed`	The engine rejected the send (credit refunded).	`message_id`, `to_number`, `status`, `error`, `credits_charged`
`message.delivered`	The recipient's device received it (ack `2`).	`message_id`, `to_number`, `status`, `ack`
`message.read`	The recipient read/played it (ack `3` / `4`).	`message_id`, `to_number`, `status`, `ack`
`batch.completed`	A scheduler-driven bulk batch finished with **zero** cancellations.	`batch_id`, `total`, `sent`, `failed`, `cancelled`
`test.ping`	You click **Send test event** in the dashboard.	`message`, `user_id`

Events are emitted only on a real status transition, so a repeated engine ack never produces a duplicate delivery. `batch.completed` is emitted **at most once**, only by the scheduler when the last open item finishes and `cancelled_count` is `0`. A fully-cancelled batch becomes `cancelled` silently and a partially-cancelled batch becomes `completed` silently — neither emits the event. Batch status/progress is also available from `GET /api/v1/batches/:id`.

Request

```
POST /your/endpoint HTTP/1.1
Content-Type: application/json
User-Agent: ZNotice-Webhooks/1.0
X-ZNotice-Event: message.delivered
X-ZNotice-Signature: sha256=<hex hmac_sha256(secret, rawBody)>
```

```
{
  "event": "message.delivered",
  "created_at": "2026-09-11T05:30:00.000Z",
  "data": { "message_id": 42, "to_number": "250788123456", "status": "delivered", "ack": 2 }
}
```

The `X-ZNotice-Signature` value is `sha256=` followed by the hex digest of `HMAC-SHA256(signing_secret, raw_request_body)`. The signing secret is shown on **Dashboard → Webhooks** and can be regenerated at any time (regeneration invalidates the previous secret).

Endpoint requirements (SSRF guard): the URL must be a public `http(s)` URL on port `80` / `443`. Loopback (`127.0.0.0/8` , `::1`), private (`10/8` , `172.16/12` , `192.168/16`), link-local (`169.254/16` , `fe80::/10`), CGNAT (`100.64/10`), ULA (`fc00::/7`) and reserved addresses are rejected at save time and re-validated (with DNS pinning) before every delivery.

Retry policy: a request has a **10s timeout**; any **2xx** response is success. Any non-2xx response or network error increments the attempt counter and schedules a retry **+1 minute**, then **+5 minutes**. After **3 failed attempts** the delivery is marked `failed` and no further attempts are made. The last 25 attempts (status, attempts, response code, error) are visible in the dashboard.

6.2 Verifying the signature (Node.js)

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

// Keep the raw body bytes — do not JSON.parse before verifying.
function verify(rawBody, header, secret) {
  const expected = 'sha256=' + crypto.createHmac('sha256', secret).update(rawBody).digest('hex');
  const a = Buffer.from(String(header || ''));
  const b = Buffer.from(expected);
  return a.length === b.length && crypto.timingSafeEqual(a, b);
}

// Express example (raw body available):
// const ok = verify(req.rawBody, req.get('X-ZNotice-Signature'),
process.env.ZNOTICE_WEBHOOK_SECRET);
// if (!ok) return res.sendStatus(401);
```

Always compare with a timing-safe function and reject requests whose signature does not match before acting on the payload.

7. Node.js integration example

```
// send-whatsapp.mjs — Node 18+ (global fetch)
const BASE = 'https://znotice.consultbofa.com';
const API_KEY = process.env.ZNOTICE_API_KEY; // zn_live_...

async function sendWhatsApp(phone, message, mediaUrl, scheduledAt) {
  const res = await fetch(`${BASE}/api/v1/send`, {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${API_KEY}`,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      phone,                 // e.g. "0788123456" (default cc 250) or "250788123456"
      message,
      ...(mediaUrl ? { media_url: mediaUrl } : {}),
      // ISO-8601, or "YYYY-MM-DDTHH:MM" interpreted as UTC. Omit to send now.
      ...(scheduledAt ? { scheduled_at: scheduledAt } : {}),
    }),
  });

  const data = await res.json();
```

```
  if (!res.ok || !data.ok) {
    // data.error is a stable code; data.detail is present for send_failed
    throw new Error(`ZNotice send failed (${res.status}): ${data.error} ${data.detail || ''}`);
  }
  // Immediate 200: { ok, id, message_id, credits_remaining }
  // Scheduled 202: { ok, scheduled_id, scheduled_at, credits_balance }
  return data;
}

async function scheduleBulk(recipients, message, { scheduledAt, label } = {}) {
  const res = await fetch(`${BASE}/api/v1/bulk`, {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${API_KEY}`,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      recipients,            // [{ to: "0788123456", name: "Alice" }, ...]
      message,               // supports {{name}} and {{phone}}
      ...(scheduledAt ? { scheduled_at: scheduledAt } : {}),
      ...(label ? { label } : {}),
    }),
  });

  const data = await res.json();
  if (!res.ok || !data.ok) {
    throw new Error(`ZNotice bulk failed (${res.status}): ${data.error}`);
  }
  return data; // { ok, batch_id, total, duplicates_removed, scheduled_at, credits_balance }
}

// Usage
sendWhatsApp('0788123456', 'Hello from my app')
  .then((r) => console.log('sent, credits left:', r.credits_remaining))
  .catch((e) => console.error(e.message));

scheduleBulk(
  [{ to: '0788123456', name: 'Alice' }, { to: '+250788654321' }],
  'Hi {{name}}, your order is ready.',
  { scheduledAt: '2026-09-12T09:30', label: 'September promo' }
).then((r) => console.log('batch', r.batch_id, 'queued:', r.total));
```

Retry guidance

- `429 rate_limited`: wait `retry_after` seconds before retrying.
- `502 send_failed`: the credit was refunded; a retry is safe (a new credit will be charged).
- `402 insufficient_credits`: top up via the dashboard, then retry.
- `202` from `/send` or `/bulk`: the message(s) are **queued**, not delivered. Do not retry — poll `GET /api/v1/scheduled` / `GET /api/v1/batches/:id` to follow progress.
- `429 queue_full`: the per-user pending cap is reached; wait for the queue to drain before enqueuing more.
- Do **not** blindly retry `4xx` validation errors.

ZNotice — System Architecture

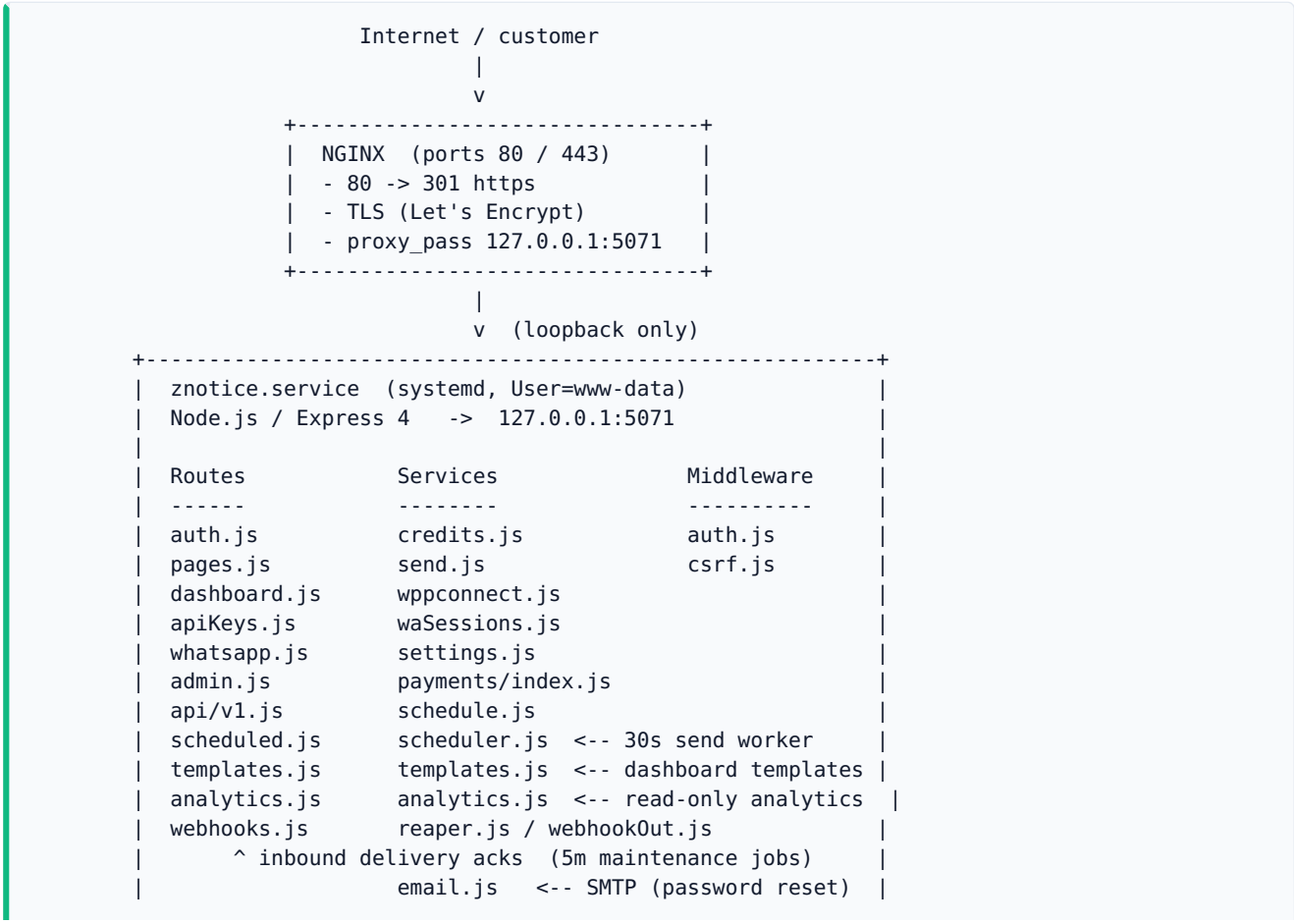
1. Overview

ZNotice is a single-node, multi-tenant SaaS built as a classic three-tier web application. NGINX terminates TLS and reverse-proxies to a Node.js/Express application. The application stores all state in MariaDB and delegates the actual WhatsApp protocol work to a locally-hosted WPPConnect Server process running in Docker.

Key properties:

- **Single-node deployment.** The app is bound to `127.0.0.1:5071` and only reachable through NGINX. There is no horizontal scaling; in-memory state (rate-limit buckets, settings cache, WPPConnect bearer-token cache) is per-process.
- **Tenant isolation by user id.** Every customer has exactly one WhatsApp session row, one credit ledger, and many API keys.
- **Prepaid credits are the source of truth for billing.** The `credit_ledger` table is authoritative; a balance is `SUM(delta)`.
- **Payments via pluggable adapters.** A payment adapter interface supports a manual offline flow (`manual`) and two live mobile-money gateways (`intouch`, Rwanda MTN MoMo / Airtel Money; `kitovu`, Rwanda MTN MoMo), added without touching routes.

2. Component diagram



```
        |                    senderName.js <-- add-on + admin name |
        |                    proofs.js              uploadProof.js |
        |                    billingAccess.js       paymentLock.js |
        |                    payments/intouch.js  intouchSweep.js  |
        |       ^ live MoMo/Airtel gateway  (5m cross-check sweep) |
        |                    payments/kitovu.js   kitovuSweep.js   |
        |       ^ live MTN MoMo gateway  (5m cross-check sweep)    |
        +--------------------------+-----------------------------+
                     |                                ^
                     | SQL (pool)                     | HTTP webhook POST
                     v                                | /webhooks/wppconnect/<secret>
        +-------------------------+              |
        |  MariaDB                |              |
        |  database: znotice      |              |
        |  tables: users, api_keys,|             |
        |  wa_sessions, messages,  |             |
        |  credit_ledger,          |             |
        |  transactions,           |             |
        |  payment_proofs,         |             |
        |  integration_orders,     |             |
        |  send_batches,           |             |
        |  scheduled_messages,     |             |
        |  message_templates,      |             |
        |  password_resets,        |             |
        |  settings, sessions,     |             |
        |  schema_migrations       |             |
        +-------------------------+              |
                                                 |
        +----------------------------------------+---------------+
        |  Docker container: wppconnect                          |
        |  wppconnect/wppconnect-server  v2.10.10                |
        |  127.0.0.1:21465 (loopback only)                       |
        |  One Chromium/WhatsApp Web session per tenant: zn_<userId> |
        +--------------------------------------------------------+
```

Supporting host services (not on the request path):

- **certbot.timer** — renews the Let's Encrypt certificate.
- **/etc/cron.d/znotice-backup** — daily DB + app/secret archive (02:30).
- **/etc/cron.d/znotice-health** — health probe every 5 minutes.
- **logrotate** — weekly rotation of the two ops log files.

In-process background jobs (started from `server/index.js`, no extra ports):

- **Scheduler** (`server/services/scheduler.js`) — every **30s** plus an immediate kick after a successful enqueue; claims due `scheduled_messages` rows and dispatches them through `services/send.js`.
- **Reaper** (`server/services/reaper.js`) — every **5 minutes** plus one pass ~30s after boot; reconciles stale WhatsApp sessions, resets interrupted scheduler claims, prunes old scheduled rows/batches, and deletes expired plus stale anonymous sessions.
- **Webhook worker** (`server/services/webhookOut.js`) — every **30s** plus a kick; delivers queued customer webhooks.
- **Intouch sweep** (`server/services/intouchSweep.js`) — every **5 minutes**; cross-checks pending `intouch` transactions older than **2 minutes** against the gateway's `gettransactionstatus` and confirms any that have settled (heals lost/early callbacks).
- **Kitovu sweep** (`server/services/kitovuSweep.js`) — every **5 minutes**; cross-checks pending `kitovu` transactions older than **2 minutes** against the gateway's `GET /merchant-transaction/payment/:externalId` and confirms any that have settled (heals lost/early callbacks).

Outbound email (`server/services/email.js`) is **not** a background job: it is called inline (fire-and-forget) by the password-reset routes and is inert until `SMTP_HOST` is configured.

Usage analytics (Phase 7) is **not** a background job: it is computed synchronously per request from read-only aggregate queries over the existing tables and adds no worker, queue or cache.

3. Ports and processes

Port	Binding	Process	Purpose
80	`0.0.0.0`	NGINX	ACME challenge + HTTPS redirect
443	`0.0.0.0`	NGINX	Public HTTPS entry point
5071	`127.0.0.1`	`node server/index.js` (systemd `znotice`)	Application HTTP server
21465	`127.0.0.1`	Docker container `wppconnect`	WPPConnect WhatsApp engine
3306	localhost	MariaDB	Application database

Process	Manager	Unit / name	User
ZNotice app	systemd	`znotice.service`	`www-data`
WPPConnect	Docker	container `wppconnect`	container
MariaDB	systemd	`mariadb.service`	`mysql`
NGINX	systemd	`nginx.service`	`www-data`

systemd unit (`/etc/systemd/system/znotice.service`)

- `After=network.target mariadb.service`
- `Type=simple`, `User=www-data`, `Group=www-data`
- `WorkingDirectory=/var/www/znotice.consultbofa.com`
- `EnvironmentFile=/var/www/znotice.consultbofa.com/.env`
- `ExecStart=/usr/bin/node server/index.js`
- `Restart=always`, `RestartSec=5`
- Hardening: `NoNewPrivileges=true`, `PrivateTmp=true`

NGINX site (`/etc/nginx/sites-available/znotice.consultbofa.com`)

- Port 80: serves `/.well-known/acme-challenge/` from `/var/www/html`, then redirects everything else to HTTPS.
- Port 443: TLS 1.2/1.3, `client_max_body_size 50m`, security headers (`X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy`), and a proxy to `http://127.0.0.1:5071` with WebSocket upgrade headers and a 3600s read/send timeout.
- The TLS certificate is issued with the webroot method and NGINX is configured manually (no certbot NGINX installer).

4. Database schema

Charset `utf8mb4` / collation `utf8mb4_unicode_ci`, engine InnoDB. Migrations live in `server/migrations/` and are applied exactly once (tracked in `schema_migrations`). The `sessions` table is created automatically by `express-mysql-session`.

`users`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`name`	VARCHAR(120)	
`email`	VARCHAR(190) UNIQUE	lowercased on registration
`password_hash`	VARCHAR(255)	bcrypt, cost 12
`role`	ENUM('user','admin')	default `user`
`status`	ENUM('active','suspended')	default `active`
`accepted_terms_at`	TIMESTAMP NULL	set on registration (Phase 2)
`sender_name_until`	DATETIME NULL	dedicated sender-name add-on expiry, UTC (Phase 10); `NULL` = never purchased. Active while `> NOW()`
`created_at`, `updated_at`	TIMESTAMP	

`api_keys`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`label`	VARCHAR(120)	default `Default`
`key_prefix`	VARCHAR(16)	first 12 chars of the key, for display
`key_hash`	CHAR(64) UNIQUE	SHA-256 hex of the full key
`last_used_at`	TIMESTAMP NULL	refreshed on each authenticated request
`revoked_at`	TIMESTAMP NULL	soft delete
`created_at`	TIMESTAMP	

`wa_sessions`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED UNIQUE FK -> users	one session per user
`session_name`	VARCHAR(64) UNIQUE	`zn_<userId>`

Column	Type	Notes
`status`	ENUM('disconnected','pending','connected','error')	default `disconnected`
`phone_number`	VARCHAR(32) NULL	connected WID digits
`profile_name`	VARCHAR(60) NULL	last WhatsApp profile/push name pushed through ZNotice (Phase 9); `NULL` = never set here
`connected_at`	TIMESTAMP NULL	first connection time
`last_qr_at`	TIMESTAMP NULL	last QR emission
`created_at`, `updated_at`	TIMESTAMP	

`messages`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`api_key_id`	BIGINT UNSIGNED NULL FK -> api_keys	`ON DELETE SET NULL`; null for dashboard sends
`to_number`	VARCHAR(32)	normalized digits
`body`	TEXT NULL	message text / media caption
`media_url`	VARCHAR(500) NULL	
`status`	ENUM('queued','sent','delivered','read','failed')	default `queued`
`wpp_message_id`	VARCHAR(190) NULL	engine message id (indexed)
`error`	TEXT NULL	sanitized failure detail
`credits_charged`	INT UNSIGNED	1 on success, 0 on failure
`created_at`, `updated_at`	TIMESTAMP	

Indexes: `(user_id, created_at)`, `wpp_message_id`, `api_key_id`, `(user_id, status)` from Phase 2, and `(created_at)` from Phase 7 (migration `007_analytics_indexes.sql`, for platform-wide date-range scans).

`credit_ledger` — authoritative balance

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`delta`	INT	signed (+ topup/refund, − charge)
`balance_after`	INT	running balance at write time

Column	Type	Notes
`reason`	ENUM('topup','purchase','message','refund','admin_adjust','bonus')	
`ref_type`	VARCHAR(40) NULL	e.g. `transaction`, `message`, `api_key`, `dashboard`, `admin`
`ref_id`	BIGINT UNSIGNED NULL	
`note`	VARCHAR(255) NULL	
`created_at`	TIMESTAMP	

Balance = `COALESCE(SUM(delta), 0)` for a user.

`transactions`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`provider`	VARCHAR(40)	default `manual`
`provider_ref`	VARCHAR(190) NULL UNIQUE	`ZN-XXXXXXXX` (Phase 3 unique key)
`amount_cents`	INT UNSIGNED	
`currency`	CHAR(3)	default `USD`
`credits`	INT UNSIGNED	credits to grant (0 for integration fee)
`status`	ENUM('pending','paid','failed','refunded')	default `pending`
`meta`	JSON NULL	`{ type: 'credits'
`created_at`	TIMESTAMP	
`paid_at`	TIMESTAMP NULL	

Indexes: `(user_id, created_at)` from Phase 1 and `(status, paid_at)` from Phase 7 (migration `007_analytics_indexes.sql`, for paid-revenue scans).

`payment_proofs` (Phase 12, migration `012_payment_proofs.sql`)

Metadata for customer-uploaded proof-of-payment files (screenshots/receipts) attached to a `pending` manual transaction. The bytes live on disk (see §8(i)); this table stores only metadata. `original_name` is display-only and is never used as a path; `stored_name` is the generated on-disk filename; `mime` is **server-sniffed** (the client value is never trusted).

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`transaction_id`	BIGINT UNSIGNED FK -> transactions	`ON DELETE CASCADE`

Column	Type	Notes
`original_name`	VARCHAR(255)	sanitized client name (control chars/path stripped), display only
`stored_name`	VARCHAR(128)	generated `<16 hex>.<ext>`, ASCII only
`mime`	VARCHAR(100)	server-sniffed MIME (`image/jpeg` , `image/png` , `image/webp` , `application/pdf`)
`size_bytes`	INT UNSIGNED	byte length of the stored file
`created_at`	TIMESTAMP	

Index: `idx_proof_tx (transaction_id, created_at)` , plus the FK `fk_payment_proofs_tx` -> `transactions(id) ON DELETE CASCADE` . Rows cascade-delete with their transaction; files live under `<UPLOADS_DIR>/payment-proofs/<transaction_id>/<16hex>.<ext>` (see §8(i)).

`integration_orders`

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`transaction_id`	BIGINT UNSIGNED NULL FK -> transactions	`ON DELETE SET NULL` (Phase 3)
`status`	ENUM('requested','paid','in_progress','delivered','cancelled')	default `requested`
`amount_cents`	INT UNSIGNED	default 2000
`contact_channel`	VARCHAR(120) NULL	`"<channel>: <value>"`
`requirements`	TEXT NULL	includes optional preferred stack
`admin_notes`	TEXT NULL	internal notes
`created_at` , `updated_at`	TIMESTAMP	

`send_batches`

One row per bulk job (and per dashboard-created schedule, which is always a batch). Counters are advanced atomically by the scheduler as items are delivered; `message` / `media_url` store the **original** template (per-recipient bodies live on `scheduled_messages`).

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`api_key_id`	BIGINT UNSIGNED NULL FK -> api_keys	`ON DELETE SET NULL` ; null for dashboard jobs
`label`	VARCHAR(120) NULL	optional batch label

Column	Type	Notes
`message`	TEXT NULL	original template, placeholders unresolved
`media_url`	VARCHAR(500) NULL	
`scheduled_at`	DATETIME	UTC due time
`status`	ENUM('scheduled','processing','completed','cancelled')	default `scheduled`
`total_count` / `sent_count` / `failed_count` / `cancelled_count`	INT UNSIGNED	counters, updated by the scheduler
`created_at`, `updated_at`	TIMESTAMP	
`completed_at`	DATETIME NULL	set when terminal

Indexes: `(user_id, created_at)`, `(status, scheduled_at)`.

`scheduled_messages`

The durable per-recipient send queue. `body` is **already placeholder-resolved at enqueue time**; `next_attempt_at` drives retry backoff. A row may belong to a batch (bulk / dashboard) or stand alone (a single API schedule with `scheduled_at`, `batch_id IS NULL`).

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`api_key_id`	BIGINT UNSIGNED NULL FK -> api_keys	`ON DELETE SET NULL`
`batch_id`	BIGINT UNSIGNED NULL FK -> send_batches	`ON DELETE CASCADE`
`recipient_name`	VARCHAR(120) NULL	substituted into `{{name}}`
`to_number`	VARCHAR(32)	normalized digits
`body`	TEXT NULL	resolved message text
`media_url`	VARCHAR(500) NULL	
`scheduled_at`	DATETIME	UTC due time
`next_attempt_at`	DATETIME NULL	retry backoff gate
`status`	ENUM('pending','sending','sent','failed','cancelled')	default `pending`; `sending` is the claim state (migration 006)
`attempts`	TINYINT UNSIGNED	default 0; terminal after 3
`last_error`	VARCHAR(500) NULL	sanitized; e.g. `interrupted_by_restart`
`message_id`	BIGINT UNSIGNED NULL FK -> messages	set on success
`created_at`, `updated_at`	TIMESTAMP	

Indexes: `(status, next_attempt_at, scheduled_at)`, `(user_id, created_at)`, `(batch_id)`.

`message_templates` (Phase 11, migration `011_message_templates.sql`)

Per-user saved message snippets used to prefill the dashboard WhatsApp and scheduled forms (`?template=<id>`). Private to the owner; never exposed through API v1.

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`name`	VARCHAR(80)	trimmed, internal whitespace collapsed; unique per user
`body`	TEXT	message text/caption; required, ≤4096 chars
`media_url`	VARCHAR(500) NULL	optional; `http(s)` only, control chars rejected
`created_at`, `updated_at`	TIMESTAMP	`updated_at` bumps on edit (drives list order)

Indexes: `UNIQUE(user_id, name)` (`uq_message_templates_user_name`), plus the `user_id` foreign key. All service queries filter on `user_id` in the `WHERE` clause, so a foreign id affects 0 rows and `resolvePrefill` returns `null`.

`settings`

Column	Type	Notes
`key`	VARCHAR(80) PK	
`value`	TEXT NULL	
`updated_at`	TIMESTAMP	

Seeded keys: `price_per_1000_credits_cents` (1000), `integration_fee_cents` (2000), `currency` (USD), `default_country_code` (250), `send_rate_per_min` (60), `max_concurrent_sessions` (15), `manual_payment_instructions`, `billing_email`, the Phase 6 queue caps `bulk_max_recipients` (1000), `schedule_max_days_ahead` (90) and `scheduled_max_pending` (5000), the Phase 10 sender-name add-on keys `sender_name_price_cents` (1000) and `sender_name_days` (30), the Phase 11 template cap `templates_max_per_user` (200), the Phase 14 IntouchPay exchange rate `intouch_rwf_rate` (float, default `1477.29`, RWF per USD at checkout), and the Phase 15 KitovuPay exchange rate `kitovu_rwf_rate` (float, default `1477.29`, RWF per USD at checkout).

`schema_migrations`

Column	Type	Notes
`filename`	VARCHAR(190) PK	applied migration file name
`applied_at`	TIMESTAMP	

`sessions` (managed by `express-mysql-session`)

Stores Express session data (`znotice.sid` cookie), created automatically on first use. Contains session state including `userId` and `csrfToken`. `sessions.expires` holds UNIX **seconds** (not milliseconds); the reaper relies on that when pruning expired/anonymous rows.

`password_resets` (Phase 8, migration `008_password_resets.sql`)

Single-use, 30-minute password-reset tokens. Only the SHA-256 hash is stored; the raw token exists solely in the emailed link.

Column	Type	Notes
`id`	BIGINT UNSIGNED PK AI	
`user_id`	BIGINT UNSIGNED FK -> users	`ON DELETE CASCADE`
`token_hash`	CHAR(64) UNIQUE	SHA-256 hex of the raw token
`expires_at`	DATETIME	UTC; 30 minutes after issue
`used_at`	TIMESTAMP NULL	set on consumption (single-use)
`created_at`	TIMESTAMP	

Index: unique `token_hash`, plus `user_id`. The token is claimed with a conditional `UPDATE ... WHERE used_at IS NULL AND expires_at > NOW()` inside the password-change transaction (see §8(f)).

5. Session naming and WhatsApp engine integration

- **Session name:** `zn_<userId>` (e.g. user 42 -> `zn_42`). Enforced unique in `wa_sessions.session_name` and one row per user.
- **Engine auth:** WPPConnect does not accept the raw `SECRET_KEY`. The app computes a per-session bearer token as `bcrypt(sessionName + SECRET_KEY)` (cost 10), URL-safe encoded (`/` → `_`, `+` → `-`), and caches it in memory (`services/wppconnect.js`). The engine validates with `bcrypt.compare(session + SECRET_KEY, presented)`.
- **Webhooks are registered per session** via the `webhook` field in the `start-session` body — there is no `set-webhook` endpoint. The app passes `https://znotice.consultbofa.com/webhooks/wppconnect/<WEBHOOK_SECRET>`.
- **Engine endpoints used** (v2.10.10): `start-session`, `qrcode-session`, `status-session`, `close-session`, `send-message`, `send-file`, `get-phone-number`, `check-number-status/{phone}` and `change-username` (Phase 9 — `POST /api/{session}/change-username` `{ name }`, which the engine maps to `client.setProfileName(name)`; called only while the session is `connected`).
- **Status mapping:** engine `CONNECTED` → `connected`; `QRCODE`, `INITIALIZING`, `PHONECODE` → `pending`; `CLOSED`, `DISCONNECTED`, empty → `disconnected`; anything else → `error`.

6. Payment adapter design

`server/services/payments/index.js` is a provider registry. Each adapter must implement a stable interface so routes never change when a gateway is added:

Method	Contract
`createCheckout({ user, amountCents, credits, meta, conn? })`	Inserts a `pending` transaction, returns `{ transactionId, reference, instructions, provider }`. The optional `conn` joins a caller transaction (defaults to the shared pool).
`markPaid(transactionId, adminUserId)`	Idempotently settles a payment; grants credits for credit top-ups, advances a linked integration order to `paid`, or extends the sender-name entitlement. The shared `settlePaid()` helper accepts an optional **credit-ledger note override** so adapters (e.g. `intouch`) can tag the ledger entry distinctly.
`markFailed(transactionId, adminUserId)`	Idempotently marks a pending transaction failed.
`handleWebhook(req)`	Reserved for async gateways; `manual` returns `{ ok: false }`. `intouch` unwraps the `jsonpayload`, cross-checks `gettransactionstatus`, and returns a pure ACK. `kitovu` cross-checks `GET /merchant-transaction/payment/:externalId` and returns a pure ACK.

Design details:

- **Registry:** `providers = { manual, intouch, kitovu }`; `getProvider(name)` resolves an adapter and throws `UNKNOWN_PROVIDER` otherwise.
- **References:** `manual` uses `ZN-` + 8 characters from an unambiguous alphabet (no `0/O/1/I/L`), collision-retried up to 5 times against the unique `provider_ref` index. `intouch` uses `ZNI-<id>-<8hex>` (a distinct prefix, since the merchant account is shared with another project whose refs use a different prefix). `kitovu` uses a **6-digit numeric** `externalId` (the gateway's lookup key), checked for uniqueness in `provider_ref` before sending and retried on a `DUPLICATE_EXTERNAL_ID` rejection.
- **Idempotency:** settlement uses a single conditional `UPDATE ... WHERE id = ? AND status = 'pending'` as the concurrency gate. Exactly one caller wins `pending → paid`. If the subsequent credit add fails, the transition is rolled back to `pending` so an admin can retry.
- **Credit top-ups:** `credits == amount_cents` ($0.01/credit).
- **Integration service fee:** `credits = 0`; settling it flips the linked `integration_orders` row from `requested` to `paid`.
- **Sender-name add-on (Phase 10):** `credits = 0`, `meta = { type:'sender_name', days }`. Settlement extends `users.sender_name_until` from `GREATEST(COALESCE(sender_name_until, NOW()), NOW())` by `days`, where `days` is clamped to **1..365** (explicit finite check; falls back to the `sender_name_days` setting when unset). The atomic `pending → paid` gate means concurrent approvals cannot double-extend.
- **Settlement compensation:** if any post-transition work (credit add, integration-order update, or sender-name entitlement extension) throws, the transaction is reverted to `pending` (`paid_at = NULL`) and the error rethrown, so an admin can retry without losing the payment.
- **Instructions templating:** the admin-editable `manual_payment_instructions` string supports `{{reference}}`, `{{amount}}`, and `{{credits}}` placeholders.

Intouch adapter (Phase 14) — `server/services/payments/intouch.js`

The first live gateway. It uses the global `fetch` (no new dependencies) and a **30s timeout** on every call.

- **Auth.** A SHA256 hex digest over `username + account_number + partner_password + timestamp` (raw concatenation, UTC `YYYYMMDDHHMMSS`). Only the digest is sent; the account number and partner

password are never transmitted in cleartext. Credentials come from `.env` (`INTOUCH_USERNAME`, `INTOUCH_ACCOUNT_NUMBER`, `INTOUCH_PARTNER_PASSWORD`).

- **Endpoints.** `POST /requestpayment/` (async, returns `Pending` `1000`), `POST /gettransactionstatus/` (no amount), and `POST /getbalance/` (ops) against the live `https://www.intouchpay.co.rw/api` or sandbox `https://developer.intouchpay.co.rw/api/v1/sandbox` base (selected by `INTOUCH_MODE`). Every request carries an `X-Trace-Id` header.
- **Phone normalization.** Rwandan numbers are normalized to `2507XXXXXXXX` before `requestpayment`.
- **Currency & amount.** Checkout converts the USD `amount_cents` to **whole RWF** at `intouch_rwf_rate` (admin-editable float, default `1477.29`, `settings.getFloat()`); the RWF amount and rate are written into the transaction `meta` alongside `{ currency:'RWF', amount_rwf, rate, intouch:{ phone, tx_id, provider, statusdesc } }`. The `transactions` row itself stays in USD (`amount_cents`), so the ledger and analytics remain USD-only.
- **Confirmation.** Settlement only proceeds when the cross-checked status is settled **and** the returned RWF amount and currency match the transaction exactly (whole-RWF, off-by-one blocked).

Intouch webhook — callback / cross-check / ACK

The provider calls `POST /payments/intouch/webhook`, mounted **before** the session/CSRF middleware and requiring **no auth** — the callback is **unsigned**, so it is never trusted on its own. The request body carries the payload wrapped in a `jsonpayload` field (an object or a JSON string).

1. The handler unwraps `jsonpayload` and looks up the local transaction.
2. It always **cross-checks ground truth** by calling `gettransactionstatus` (3 bounded retries) before settling.
3. It responds with a **pure ACK** regardless of outcome: `200 {"message":"success","success":true,"request_id":"<uuid>"}`. Malformed, unknown or mismatched payloads are logged and still ACKed `200` — no validation detail is leaked and there is no retry storm.
4. A terminal failure (amount/currency mismatch, gateway error) is recorded in `meta.failure_reason` and the transaction stays `pending` (or is marked `failed`), where the sweep or an admin status check can recover it.

Intouch sweep — `server/services/intouchSweep.js`

Every **5 minutes** the sweep selects pending `intouch` transactions older than **2 minutes**, cross-checks each via `gettransactionstatus`, and confirms the ones that have settled. This heals callbacks that were lost or arrived before the transaction was fully written. It shares the same settle path and strict amount/currency matching as the webhook, so a late callback and the sweep cannot double-credit.

Kitovu adapter (Phase 15) — `server/services/payments/kitovu.js`

The second live gateway. It uses the global `fetch` (no new dependencies) and a **30s timeout** on every call.

- **Auth.** Every POST carries `x-api-key` (merchant API key), `x-api-timestamp` (Unix **milliseconds as a string**, `Date.now().toString()`) and `x-api-signature` = HMAC-SHA256 (lowercase hex) over the compact JSON envelope `{"method":"POST","path":"<path>","body":{...}}` keyed with the API secret. The HTTP body is the same body object serialized with `JSON.stringify` (compact, no escaped slashes/unicode); envelope key order is method, path, body. GET (status) needs only `x-api-key`. Credentials come from `.env` (`KITOVUPAY_API_KEY`, `KITOVUPAY_API_SECRET`).

- **Endpoints.** `POST /merchant-transaction/merchant/collection/mobile_money` (async, body `{ amount, payment_rail:'mtn', sender_account_issuer:'mtn', sender_account_name, sender_account_number, description, callbackUrl, externalId }`) against the UAT `https://gateway.uat.kitovupay.com` or live `https://gateway.kitovupay.com` base (selected by `KITOVUPAY_MODE`). Status is `GET /merchant-transaction/payment/:externalId` → `{ reference_id, status, reason }`. 429 responses are retried with bounded exponential backoff (3 attempts, 250ms/500ms).
- **External ID.** `externalId` is a **6-digit numeric string**, unique per merchant account. It is checked for uniqueness in `transactions.provider_ref` before sending; a `DUPLICATE_EXTERNAL_ID` rejection allocates a fresh ref and retries (bounded). `provider_ref` stays `NULL` until a collection is sent.
- **Phone normalization.** Rwandan numbers are normalized to `2507XXXXXXXX`.
- **Description.** Sanitized to plain text (letters/numbers/spaces) — symbols like `#` break the USSD push.
- **Currency & amount.** Checkout converts the USD `amount_cents` to **whole RWF** at `kitovu_rwf_rate` (float, default `1477.29`, `settings.getFloat()`); the RWF amount and rate are written into the transaction `meta` alongside `{ currency:'RWF', amount_rwf, rate, kitovu:{ phone, tx_id, status, requested_at } }`. The `transactions` row stays in USD.
- **Confirmation.** Settlement only proceeds when the cross-checked status is `APPROVED`; when a webhook supplies an amount it must match the transaction exactly (whole-RWF, off-by-one blocked).

Kitovu webhook — callback / cross-check / ACK

The provider calls `POST /payments/kitovu/webhook`, mounted **before** the session/CSRF middleware and requiring **no auth** — the callback is **unsigned**, so it is never trusted on its own. The body is `{ transactionId, externalId, status, amount, message }`.

1. The handler looks up the local transaction by `externalId`.
2. It always **cross-checks ground truth** by calling `GET /merchant-transaction/payment/:externalId` (3 bounded retries) before settling.
3. It responds with a **pure ACK** regardless of outcome: `200 {"message":"success","success":true,"request_id":"<uuid>"}`. Malformed, unknown or mismatched payloads are logged and still ACKed `200`.
4. A terminal failure (amount mismatch, gateway error) is recorded in `meta.failure_reason` and the transaction stays `pending`, where the sweep or an admin status check can recover it.

Kitovu sweep — `server/services/kitovuSweep.js`

Every **5 minutes** the sweep selects pending `kitovu` transactions older than **2 minutes** (with `meta.kitovu.requested_at` set), cross-checks each via the status endpoint, and confirms the ones that settled. This heals callbacks that were lost or arrived early. It shares the same settle path and strict amount matching as the webhook, so a late callback and the sweep cannot double-credit.

7. Capacity guard

WhatsApp sessions are RAM-heavy (roughly 300–500 MB per active Chromium session). The operator sets `max_concurrent_sessions` (default **15**) in admin settings. The guard is implemented in `server/routes/whatsapp.js`:

- It counts rows in `wa_sessions` with status `pending` or `connected`.

- If the count is below the cap, the request is allowed.
- If the cap is reached, a user is still allowed through **only if their own session is already counted** (so a user mid-pairing can keep polling or reconnecting to their own QR).
- Otherwise the connect/poll route returns `409 at_capacity` with a friendly message. The public API `/api/v1/send` is not gated by this guard — it only requires the user's own session to be `connected`.

8. Core flows

(a) Customer signup → payment → credit top-up

```
Browser ──POST /register (name,email,password,accept_terms)──▶ auth.js
   auth.js: validate, bcrypt hash, INSERT users(accepted_terms_at=NOW())
   establishSession() → session.regenerate() → session.userId
        │
        ▼
Browser ──POST /dashboard/billing/checkout {amount,method}──▶ dashboard.js
   validate $1.00–$500.00 → amountCents
   payments.getProvider('manual').createCheckout({amountCents, credits=amountCents})
        │  INSERT transactions(status=pending, provider_ref='ZN-XXXXXXXX', credits=N)
        ▼
   redirect → GET /dashboard/billing/pay/:id
        │  render manual_payment_instructions with {{reference}} etc.
        ▼
Customer pays offline (bank / mobile money) quoting ZN-XXXXXXXX
        │
        ▼
Admin ──POST /admin/transactions/:id/paid──▶ admin.js → payments.markPaid(id)
   1. UPDATE transactions SET status='paid' WHERE id=? AND status='pending'  (gate)
   2. if credits > 0: credits.addCredits(reason='topup', ref=transaction)
        └─ ledger row (+credits, balance_after) inside a user-row-locked tx
   3. if meta.type='integration': integration_orders requested → paid
   On credit-add failure: roll back to pending for safe retry.
```

(b) QR connect

```
Dashboard ──POST /dashboard/whatsapp/connect──▶ whatsapp.js
   waSessions.ensure(userId)            → row created status=disconnected
   checkCapacity(userId, wa)            → 409 at_capacity if full and not owner
   if wa.status='connected': probe engine; if still connected, short-circuit
   else wpp.startSession('zn_<id>', { webhook: <secret URL>, waitQrCode:true })
        │  engine returns status + qrcode (data URL)
        ▼
   normalizeStatus() → persist pending/connected/error + last_qr_at
   respond { ok, status, qr, phone_number }
        │
        ▼
UI polls GET /dashboard/whatsapp/qr every 3s
   wpp.getStatus() → status-session
   if pending and no QR: wpp.getQrCode() → qrcode-session (PNG/JSON → data URL)
   if connected: fetch get-phone-number, persist phone + connected_at
        │
        ▼
User scans QR in WhatsApp → engine fires status-find webhook
   /webhooks/wppconnect/<secret> → waSessions.updateBySessionName(status=connected)
```

(c) API send pipeline with atomic credit charge + refund

```
Client ──POST /api/v1/send {phone,message,media_url?}──▶ api/v1.js
   requireApiKey: hash(SHA-256) lookup → active user? → req.apiUser/req.apiKey
        │
        ▼  services/send.js performSend()
   1. settings.ensure()  → default_country_code (250), send_rate_per_min (60)
   2. checkRate(rateKey='api:<keyId>')  → 429 rate_limited + retry_after
   3. validateInput()   → phone normalize + message/media checks → 400
   4. user must exist (404) and be active (403)
   5. wa_sessions.status must be 'connected' → 409 whatsapp_not_connected
   6. chargeCredits(userId, amount=1)   ← atomic:
         BEGIN; SELECT id FROM users WHERE id=? FOR UPDATE;
         balance = SUM(delta); if balance < 1 throw INSUFFICIENT_CREDITS;
         INSERT credit_ledger(delta=-1, balance_after); COMMIT
         → 402 insufficient_credits { balance }
   7. engine call: wpp.sendMessage() or wpp.sendMedia() (send-file)
        ├─ success → INSERT messages(status='sent', wpp_message_id, credits_charged=1)
        │            respond { ok, id, message_id, credits_remaining }
        └─ failure → INSERT messages(status='failed', error=<sanitized>, credits_charged=0)
                     addCredits(+1, reason='refund', ref=message)
                     respond 502 { ok:false, error:'send_failed', detail }
```

(d) Delivery webhook status updates

```
Engine ──POST /webhooks/wppconnect/<secret>  (mounted before session + CSRF)
   secretOk(): timing-safe compare of path token or x-znotice-webhook-secret
        │  401 invalid_webhook_secret on mismatch
        ▼
   event routing (routes/webhooks.js):
     onack        → look up messages by wpp_message_id, advance status
     qrcode       → wa_sessions.status='pending', last_qr_at=NOW()
     status-find  → map engine status → connected / disconnected
     closesession / logoutsession → status='disconnected'
     onmessage    → log only, never charge
        │
        ▼
   Ack → status mapping (monotonic; never downgrades):
     1 sent · 2 delivered · 3 read · 4 read · negative = failed
     STATUS_RANK = { queued:0, sent:1, delivered:2, read:3, failed:4 }
   The handler always returns HTTP 200 so webhook processing can never
   fail the engine's delivery.
```

(e) Scheduled & bulk enqueue → dispatch → completion

```
Client ──POST /api/v1/bulk {recipients[], message, scheduled_at?}──▶ api/v1.js
   requireApiKey → schedule.createBulk()
     settings.ensure(): bulk_max_recipients, schedule_max_days_ahead,
                        scheduled_max_pending
     parseScheduledAt(scheduled_at)   → 400 invalid_scheduled_at
                                        (bad / < now+60s / > horizon)
     normalizeRecipients()            → 400 invalid_recipients (any bad entry)
                                        dedupe → duplicates_removed
     resolveTemplate() per recipient  ({{name}}, {{phone}}; unknown tokens kept)
     account active? session connected? balance ≥ total? → 403 / 409 / 402
     BEGIN;
       SELECT id FROM users WHERE id=? FOR UPDATE   ← serializes the cap check
       pending = COUNT(scheduled_messages WHERE status='pending')
       if pending + total > scheduled_max_pending → ROLLBACK 429 queue_full
       INSERT send_batches(status='scheduled', total_count=N)
       INSERT scheduled_messages × N (status='pending')
```

```
    COMMIT
  scheduler.kick() → 202 { batch_id, total, duplicates_removed,
                           scheduled_at, credits_balance }
       │
       ▼  every 30s (services/scheduler.js runOnce; in-flight guard)
  SELECT due pending rows (ROW_NUMBER per user, ≤25/user, global LIMIT 50)
    for each row:
      rate-limit check (user:<id>, send_rate_per_min) → defer +1 min if over
      CLAIM: UPDATE ... SET status='sending' WHERE id=? AND status='pending'
             (0 rows → skip; a cancel can only ever target 'pending')
      executeSend(): account/session checks, media fetch, charge 1 credit,
                     engine call
        ├─ 200  → item 'sent', message_id; batch sent_count++  (atomic)
        ├─ retryable (409 not connected, 502 engine unreachable / HTTP 5xx):
        │        attempts++; release back to 'pending' (+1 min / +5 min);
        │        failed at 3 attempts
        └─ other → item 'failed' (the pipeline already refunded the credit)
      bumpBatch() → status='processing'
      maybeCompleteBatch(emit=true): single conditional UPDATE
        (status IN ('scheduled','processing') AND no open item)
        → 'completed' (or 'cancelled'); the winner emits `batch.completed`
        once, only for a scheduler-driven batch with 0 cancellations
```

Crash recovery: the reaper resets `sending` rows older than **10 minutes** to `failed` / `last_error='interrupted_by_restart'`, because a claim that outlived the process may already have reached WhatsApp. This keeps dispatch **at-most-once** — a crash can drop a message but never duplicate it.

(f) Password reset (Phase 8)

```
Visitor ──POST /forgot-password {email}──► auth.js  (forgotLimiter 5/15m/IP)
  validate address (≤190 chars, email regex); look up active user
  if found:
    DELETE password_resets WHERE user_id=? AND used_at IS NULL   -- invalidate prior
    INSERT password_resets(user_id, token_hash=sha256(raw), expires_at=NOW()+30m)
    email.send(passwordResetEmail(resetUrl))   -- fire-and-forget, never blocks
  always redirect → /forgot-password?sent=1    -- identical for all addresses
       │
       ▼
User clicks  GET /reset-password?token=<raw>  ──► auth.js
  SELECT ... WHERE token_hash=sha256(raw) AND used_at IS NULL AND expires_at>NOW()
  found → render form (valid); else → 400 invalid-link page
       │
       ▼
User submits POST /reset-password {token,password,confirm}  (resetLimiter 10/15m/IP)
  validate password (≥8 chars, ≤72 bytes, confirm match)
  bcrypt.hash(password, 12)
  BEGIN;
    UPDATE password_resets SET used_at=NOW()
      WHERE id=? AND used_at IS NULL AND expires_at>NOW()   -- ATOMIC CLAIM
    affectedRows !== 1 → ROLLBACK + 400 invalid-link (race/expiry loser)
    UPDATE users SET password_hash=?
    DELETE password_resets WHERE user_id=? AND id<>? AND used_at IS NULL
  COMMIT;
  destroyAllSessions(userId)          -- sign out everywhere (exact $.userId match)
  regenerate current session if it belongs to the reset user
  email.send(passwordChangedEmail())  -- fire-and-forget
  flash success → redirect /login
```

`server/services/email.js` is the only outbound-email component. It reads `SMTP_*` from the environment per call, reuses a transport keyed by host/port/ secure/user plus a short non-reversible password fingerprint (so a rotated `SMTP_PASS` rebuilds the transport), never throws, and logs only a sanitized error code. With no `SMTP_HOST` it returns `email_not_configured` and the request still succeeds.

(g) Sender name + dedicated sender-name add-on (Phases 9–11)

```
Dashboard → WhatsApp ──POST /dashboard/billing/sender-name──▶ dashboard.js
   settings.ensure(): sender_name_price_cents (1000), sender_name_days (30)
   BEGIN;
     SELECT id FROM users WHERE id=? FOR UPDATE          -- serialize per user
     existing pending sender_name tx? → ROLLBACK + redirect to it
     INSERT transactions(status='pending', credits=0,
                         provider_ref='ZN-XXXXXXXX',
                         meta={type:'sender_name', days})   -- same tx
   COMMIT → redirect /dashboard/billing/pay/:id
        │
        ▼
Customer pays offline quoting ZN-XXXXXXXX
        │
        ▼
Admin ──POST /admin/transactions/:id/paid──▶ payments.markPaid(id)
   1. UPDATE transactions SET status='paid' WHERE id=? AND status='pending' (gate)
   2. meta.type='sender_name' →
        UPDATE users
           SET sender_name_until =
             DATE_ADD(GREATEST(COALESCE(sender_name_until, NOW()), NOW()),
                      INTERVAL ? DAY)
         WHERE id=?                                     -- days clamped 1..365
      failure → revert tx to 'pending' (paid_at=NULL), rethrow
        │
        ▼
Dashboard → WhatsApp ──POST /dashboard/whatsapp/sender-name──▶ whatsapp.js
   wa_sessions.ensure(userId)
   wa.status !== 'connected'                → flash + redirect (no engine call)
   senderName.isSenderNameActive(userId)    → active? (sender_name_until > NOW())
       inactive → flash + redirect to Billing  -- entitlement gate BEFORE engine
   validateSenderName(req.body.sender_name)  → control chars/line breaks rejected;
                                               whitespace collapsed; 1..25 chars
   wpp.setProfileName('zn_<id>', name)      → POST /api/{session}/change-username
   waSessions.update(userId, { profileName: name })
   flash success → redirect /dashboard/whatsapp
```

The entitlement check (`services/senderName.js`) is shared by the route and the WhatsApp view, so the form and the server enforce the same rule. On the **customer** route the engine is never called when the session is disconnected or the add-on is inactive. Engine failure detail is logged server-side only; the customer sees a generic message. `wa_sessions.profile_name` is display state — the authoritative profile name lives on the WhatsApp account itself.

Admin sender-name override (Phase 11) — deliberate entitlement bypass

```
Admin → Sessions ──POST /admin/sessions/:id/sender-name──▶ admin.js
   parse id (safe integer > 0; else "Invalid session")
   SELECT wa_sessions JOIN users WHERE w.id=?  → 404-style flash if missing
   validateSenderName(req.body.sender_name)    → same rules as the customer route
   session.status must be 'connected'          → flash if not (NO engine call)
   wpp.setProfileName(session.session_name, name)
       engine offline → generic "engine offline" flash
```

```
       other failure  → log detail server-side, generic flash (never raw error)
   waSessions.update(session.user_id, { profileName: name })
```

Unlike `POST /dashboard/whatsapp/sender-name`, the admin route does **not** call `isSenderNameActive` — it is support tooling, so a platform admin can correct/override a tenant's profile name without the tenant owning the paid add-on. This is the **only** sender-name entitlement bypass; the customer route and its paywall are unchanged. The route is behind `requireAdmin` and the global CSRF middleware, and it never surfaces raw engine errors.

(h) Message templates & dashboard prefill (Phase 11)

Templates are a **dashboard-only** convenience: a user saves named message bodies (plus an optional media URL) and prefills the send forms from them. There is no API v1 surface, no background worker and no change to the send pipeline — `{{name}}` / `{{phone}}` placeholders are still resolved at enqueue/dispatch time by the existing scheduler/bulk code.

```
Dashboard → Templates ──POST /dashboard/templates──▶ routes/templates.js
   (mounted BEFORE the /dashboard catch-all; requireAuth + global CSRF)
   validateTemplate(name ≤80, body ≤4096, media http(s) ≤500, no control chars)
   createForUser(userId, tpl):
     BEGIN; SELECT id FROM users WHERE id=? FOR UPDATE;      -- serialize per user
            SELECT COUNT(*) FROM message_templates
              WHERE user_id=? FOR UPDATE;                    -- lock the scanned rows
            count >= templates_max_per_user → ROLLBACK limit_reached
            INSERT message_templates(...)  (ER_DUP_ENTRY → duplicate_name)
     COMMIT
         │
         ▼
Prefill (server-side, no JS)
   GET /dashboard/whatsapp?template=<id>  ─▶ resolvePrefill(userId, id)
   GET /dashboard/scheduled?template=<id> ─▶ resolvePrefill(userId, id)
     parseTemplateId: digits only, safe integer > 0 (else null)
     SELECT body, media_url WHERE id=? AND user_id=? LIMIT 1
       owned → { body, mediaUrl } rendered into the form
       foreign / invalid / missing → null (no information leak; blank form)
```

`routes/templates.js` also handles `POST /:id` (update) and `POST /:id/delete` (delete); both enforce ownership in the SQL `WHERE` clause, so a foreign id affects 0 rows and flashes "Template not found". The list is bounded by `listForUser(userId, 500)`. The per-user cap is read from the `templates_max_per_user` setting (default 200) inside the create transaction, so concurrent creates for one user cannot overrun it.

(i) Payment proof uploads (Phase 12)

A manual-payment proof is a screenshot/receipt a customer attaches to a `pending` transaction from the pay page. Metadata is written to `payment_proofs`; the bytes are written to disk under `<UPLOADS_DIR>/payment-proofs/`. There is no API v1 surface and no background worker. Types are JPG/PNG/WebP/PDF, max **5 MB** each, max **5 per transaction** and max **25 files / 50 MB per user**.

Middleware order in `server/index.js` (this is load-bearing — see below):

```
app.use('/dashboard/billing/proof', proofUploadLimiter, multerSingle)
        │  proofUploadLimiter: 30 POSTs / 15 min, keyed by `user:<id>` else
        │  `ip:<ip>`; non-POST requests skip. Over limit → error flash + 302
        │  to /dashboard/billing/pay/:id.
```

```
        │  multerSingle = uploadProof.single('proof') (server/middleware/
        │  uploadProof.js): memoryStorage, limits { fileSize: 5 MB, files: 1,
        │  fields: 1, fieldSize: 1024, parts: 3 }. Multipart only; other
        │  content types pass straight through to the routers.
        │  Parses the hidden `_csrf` field into req.body — nothing hits disk.
        ▼
app.use(verifyCsrf)                    ← global CSRF check now sees req.body._csrf
        ▼
app.use('/dashboard', requirePayment)  ← locked users may still upload (billing path)
        ▼
routes/dashboard.js  POST /billing/proof/:id
   validateProofBuffer(req.file.buffer) → magic-byte sniff + ≤5 MB
   safeOriginalName(req.file.originalname) / makeStoredName(mime) → <16hex>.<ext>
   BEGIN;
     SELECT ... FROM transactions WHERE id=? FOR UPDATE      -- row lock
       not owned / not pending → 404 or flash
     SELECT COUNT(*) FROM payment_proofs WHERE transaction_id=? FOR UPDATE
       assertProofLimit(count)                               -- ≤5 per transaction
     getUserProofUsage(userId, conn) → assertUserProofQuota  -- ≤25 / ≤50 MB
     INSERT payment_proofs(...)
   COMMIT; release
   writeProof()                          ← file write OUTSIDE the lock (H4)
     failure → best-effort delete the committed row + unlink partial file
   flash success → redirect /dashboard/billing/pay/:id
```

Why multer precedes CSRF. The global `verifyCsrf` reads `req.body._csrf`, but `express.urlencoded` cannot parse a `multipart/form-data` body. Parsing the upload first (into memory) populates `req.body` so the hidden `_csrf` field is visible and the normal synchronizer-token check still applies. The rate limiter runs before multer so an unauthenticated/abusive client cannot make the server buffer files, and because the storage is `memoryStorage`, nothing touches disk until the route has validated CSRF, ownership, transaction status, size and magic bytes. The DB row is committed before the file write (H4) so the transaction row lock is never held across disk I/O; serving returns 404 during the brief row-exists-file-missing window.

Proof routes:

Route	Auth	Behaviour
`POST /dashboard/billing/proof/:id`	owner (`requireAuth`)	Upload one proof while the transaction is `pending`.
`GET /dashboard/billing/proof/:id/:proofId`	owner	Stream the file; foreign/missing → `404`.
`POST /dashboard/billing/proof/:id/:proofId/delete`	owner	Delete while `pending`; ownership + status re-asserted in the `DELETE`.
`GET /admin/transactions/:id/proof/:proofId`	`requireAdmin`	Admin stream of any proof; missing → `404`.

Serving headers (both owner and admin routes; MIME comes from the DB row, never the client):

```
Content-Type: <sniffed mime>
Content-Disposition: inline; filename="proof-<proofId>.<ext>"
Cache-Control: private, no-store
Content-Security-Policy: default-src 'none'; sandbox
```

Proofs remain readable by the owner and admins after `paid` / `failed` for audit; upload/delete are restricted to `pending`.

Storage layout and permissions (`server/services/proofs.js`):

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

- `UPLOADS_DIR` (env) overrides `config.paths.uploads`; the default is `<app root>/uploads`. `config.paths.uploads` is read at call time, so an override (e.g. in tests) is honoured.
- Directories are created `0o750`, files written `0o640`.
- `uploads/` is **outside** `public/`, so proof files are only reachable through the authenticated Node routes above.
- Stored filenames are always generated (`crypto.randomBytes(8)` → `<16 hex>.<ext>`) and re-validated against `^[0-9a-f]{16}\.(jpg|png|webp|pdf)$` before any path join, so a user-controlled name can never traverse or select a path.

(j) Billing lock for unpaid users (Phase 13)

A non-admin account with no `paid` transaction and no positive credit balance is confined to the payment flow. The gate is mounted at `/dashboard` immediately after `verifyCsrf` and before every dashboard router:

```
app.use(verifyCsrf);
app.use('/dashboard', requirePayment);          // middleware/paymentLock.js
app.use('/dashboard/account', accountRouter);   // allowed for locked users
app.use('/dashboard/webhooks', ...);            // blocked for locked users
... every other /dashboard/* router ...
app.use('/dashboard', dashboardRouter);         // /billing/* lives here
app.use('/admin', adminRouter);                 // unaffected
```

`requirePayment` (`server/middleware/paymentLock.js`) delegates the decision to `server/services/billingAccess.js`:

- **Unlock rule.** An account is unlocked when it is an `admin`, **or** has `EXISTS(SELECT 1 FROM transactions WHERE user_id=? AND status='paid')`, **or** `COALESCE(SUM(credit_ledger.delta), 0) > 0`. Pending/failed transactions alone do not unlock. `getBillingAccess(userId)` resolves this in one query and looks up the most recent `pending` transaction only when locked.
- **Anonymous** requests pass through; the downstream `requireAuth` owns the login redirect (a single auth gate).
- **Admins** always pass (`res.locals.billingLocked = false`).
- **Locked + allowed path** passes with `res.locals.billingLocked = true`.
- **Locked + any other path** sets an error flash and `302`s to `/dashboard/billing/pay/<pendingTransactionId>` (most recent pending transaction) or `/dashboard/billing` when there is none.
- **Allowed prefixes** (relative to the `/dashboard` mount): `/billing`, `/account`, `/integration`. `/integration` is included because the integration request form (`POST /dashboard/integration/request`) creates a pending $20 payment and is shown on the Billing page (hardening H3). `isAllowedBillingPath` is pure and defensive: non-string/empty input is never allowed, so malformed input can only lock down.

`res.locals.billingLocked` is consumed by `partials/dashnav.ejs` (renders only the Billing + Account tabs plus a "Locked — payment required" chip) and by `dashboard/billing.ejs` (amber "your account is locked" banner). `/api/v1/*` is **intentionally not gated**: it is API-key authenticated and a zero balance already fails with `402 insufficient_credits`. `/admin/*` is unaffected.

Admin proof visibility. The admin overview pending-payments query (`routes/admin.js`) returns `proof_count` via a correlated subquery and `views/admin/index.ejs` renders an "N proof(s)" badge (or "No proof"), so an admin can spot transactions with uploaded proof before approving. `GET /admin/transactions` attaches each transaction's proofs in one extra query (no N+1) and the table renders a "Proofs (N)" column linking each file.

(k) IntouchPay instant credit top-up (Phase 14)

Credit top-ups only — the sender-name add-on and integration service remain manual. The customer pays by approving a USSD prompt on their phone; the transaction is auto-confirmed, with no admin approval or proof upload.

```
Browser ──POST /dashboard/billing/intouch/checkout {amount,phone}──▶ dashboard.js
   validate $1.00–$500.00 → amountCents
   convert amountCents → whole RWF at intouch_rwf_rate (settings.getFloat())
   normalize phone → 2507XXXXXXXX
   payments.getProvider('intouch').createCheckout({...})
        │  INSERT transactions(status=pending, provider='intouch',
        │    provider_ref='ZNI-<id>-<8hex>', amount_cents=USD,
        │    meta={currency:'RWF', amount_rwf, rate, intouch:{phone}})
        ▼
   POST /requestpayment/  →  tx_id (async, Pending 1000)
        │  store meta.intouch.tx_id
        ▼
   redirect → GET /dashboard/billing/pay/:id  (Intouch branch, no proof UI)
        │
        ▼
Customer approves the USSD prompt on their MTN MoMo / Airtel Money phone
        │
        ├─(path 1) InTouch POSTs /payments/intouch/webhook (unsigned, jsonpayload)
        │     webhook: unwrap → cross-check gettransactionstatus (3 retries)
        │              → settled + exact RWF/currency match? settlePaid()
        │              → always ACK 200 {"message":"success",...}
        │
        └─(path 2) intouchSweep (every 5m) cross-checks pending rows > 2m old
             → confirm settled ones (heals lost/early callbacks)

   settlePaid(): pending → paid (atomic gate) → credits.addCredits(note override)
        │  credit_ledger (+amount_cents, reason=topup)
        ▼
Browser ──POST /dashboard/billing/intouch/:id/status──▶ manual cross-check
   (optional) re-checks status; shows "auto-confirmed by InTouch" once paid
```

Key properties:

- **Auto-confirmation only.** An Intouch transaction never goes through the manual Approve/Reject buttons — the admin table shows a provider chip and a muted "auto-confirmed by InTouch" hint. The only trust anchor is the `gettransactionstatus` cross-check, never the unsigned callback.
- **Strict matching.** Settlement requires the gateway's returned status to be settled and the RWF amount + currency to equal the transaction's `meta` exactly (whole RWF, off-by-one blocked).
- **Re-request.** `POST /dashboard/billing/intouch/:id/start` re-sends the USSD prompt for a still-pending transaction.

- **No refunds via the processor.** Intouch has no reversal endpoint, so refunds stay local-ledger only (an admin credit adjustment), exactly like `manual`.
- **Idempotency.** The same conditional `pending → paid` gate as `manual` applies, so webhook + sweep racing a single settlement cannot double-credit.

(l) KitovuPay instant credit top-up (Phase 15)

Credit top-ups only — the sender-name add-on and integration service remain manual. The customer pays by approving an MTN MoMo prompt on their phone; the transaction is auto-confirmed, with no admin approval or proof upload.

```
Browser ──POST /dashboard/billing/kitovu/checkout {amount,phone,name?}──▶ dashboard.js
   validate $1.00–$500.00 → amountCents
   convert amountCents → whole RWF at kitovu_rwf_rate (settings.getFloat())
   normalize phone → 2507XXXXXXXX
   payments.getProvider('kitovu').createCheckout({...})
        │  INSERT transactions(status=pending, provider='kitovu',
        │    provider_ref=NULL, amount_cents=USD,
        │    meta={currency:'RWF', amount_rwf, rate, kitovu:{phone:null,...}})
        ▼
   requestForTransaction: allocate unique 6-digit externalId
        │  (checked in transactions.provider_ref; retry on DUPLICATE_EXTERNAL_ID)
   POST /merchant-transaction/merchant/collection/mobile_money (signed)
        │  store provider_ref=externalId, meta.kitovu.{phone,tx_id,status,requested_at}
        ▼
   redirect → GET /dashboard/billing/pay/:id  (Kitovu branch, no proof UI)
        │
        ▼
Customer approves the MTN MoMo prompt on their phone
        │
        ├─(path 1) KitovuPay POSTs /payments/kitovu/webhook (unsigned)
        │     webhook: cross-check GET /merchant-transaction/payment/:externalId (3 retries)
        │               → APPROVED (+ exact RWF amount when present)? settlePaid()
        │               → always ACK 200 {"message":"success",...}
        │
        └─(path 2) kitovuSweep (every 5m) cross-checks pending rows > 2m old
              → confirm settled ones (heals lost/early callbacks)

   settlePaid(): pending → paid (atomic gate) → credits.addCredits(note override)
        │  credit_ledger (+amount_cents, reason=topup, note='kitovu top-up verified')
        ▼
Browser ──POST /dashboard/billing/kitovu/:id/status──▶ manual cross-check
   (optional) re-checks status; shows "auto-confirmed by KitovuPay" once paid
```

Key properties:

- **Auto-confirmation only.** A Kitovu transaction never goes through the manual Approve/Reject buttons — the admin table shows a provider chip and a muted "auto-confirmed by KitovuPay" hint. The only trust anchor is the status cross-check, never the unsigned callback.
- **Strict matching.** Settlement requires the gateway's status to be `APPROVED`; a webhook-supplied amount must equal the transaction's `meta` exactly (whole RWF, off-by-one blocked).
- **Re-request.** `POST /dashboard/billing/kitovu/:id/start` re-sends the prompt for a still-pending transaction.
- **No refunds via the processor.** KitovuPay has no reversal endpoint in scope, so refunds stay local-ledger only (an admin credit adjustment), exactly like `manual`.
- **Idempotency.** The same conditional `pending → paid` gate as `manual` applies, so webhook + sweep racing a single settlement cannot double-credit.

9. Usage analytics (Phase 7)

Analytics is a **read-only aggregation layer** over the existing `messages`, `credit_ledger`, `transactions` and `users` tables. It performs no writes, keeps no state, and has no background worker. All queries are parameterized and all day buckets are UTC.

Components

Piece	Location	Notes
Service	`server/services/analytics.js`	`customerDaily`, `customerSummary`, `adminDaily`, `adminSummary`, `topUsers`. Coerces MySQL string aggregates to numbers and zero-fills each window to exactly `days` points.
Customer route	`server/routes/analytics.js`	Mounted at `/dashboard/analytics` (before the `/dashboard` catch-all), `requireAuth`, fixed 30-day window.
Admin route	`server/routes/admin.js` (`GET /admin/analytics`)	Behind `requireAdmin`, fixed 30-day window.
API route	`server/routes/api/v1.js` (`GET /api/v1/usage`)	API-key auth; `?days=` default 30, clamped 1–90.
Views	`server/views/dashboard/analytics.ejs`, `server/views/admin/analytics.ejs`	Summary cards, charts, funnel, tables and empty states.
Chart partial	`server/views/partials/svg-bars.ejs`	Reusable server-rendered inline SVG bar chart (grouped bars when a second series is present).

Metric semantics

- **Window:** a `days`-long window starts at `UTC_DATE() - INTERVAL (days-1) DAY` (inclusive), so it ends on today and always yields exactly `days` points.
- **Completed statuses:** `sent`, `delivered`, `read`, `failed`. `queued` messages are excluded from every customer count and from every `credits_spent` sum (`credits_spent = SUM(credits_charged)` over completed statuses only).
- **Admin `messages` / `active_users`:** count **all** message rows (including `queued`); `active_users` is `COUNT(DISTINCT user_id)`.
- **Revenue:** `revenue_cents` and `credits_sold` come from `transactions WHERE status='paid'`, grouped by `DATE(paid_at)`.
- **Point-in-time figures:** `credits_outstanding = SUM(delta)` over the whole ledger and `pending_payments = COUNT(*)` of pending transactions are **not** windowed.
- **Failure reasons:** the customer page lists up to 5 error strings for `failed` messages in the last 30 days; blank/NULL errors are normalized to `unknown`.

Charts are CSP-safe

The charts are plain inline SVG generated server-side by `partials/svg-bars.ejs` using **presentation attributes only** — no inline `<script>`, no `style` attributes and no third-party chart library. This is what keeps the strict Content-Security-Policy (`script-src 'self'`, `style-src-attr 'none'`) intact while still rendering interactive tooltips via SVG `<title>` elements.

Indexes

Migration `server/migrations/007_analytics_indexes.sql` adds two indexes (both idempotent and guarded via `information_schema`, so the file is safe to re-run):

- `messages(created_at)` (`idx_messages_created`) — platform-wide date-range scans (admin analytics).
- `transactions(status, paid_at)` (`idx_tx_status_paid`) — paid-revenue scans.

Customer-scoped queries use the pre-existing `messages(user_id, created_at)` index. Because the migration is index-only, it changes no data.

Input hardening

`days` is clamped to `1..90` both at the HTTP routes and again inside the service (`clampDays`), and `topUsers` clamps `limit` to `1..50`. This is defense in depth: a direct service call can never produce an `INTERVAL NaN DAY` SQL error or allocate an unbounded day-key array.

10. Test suite layout

The suite uses Node's built-in `node:test` runner — there are **no** new test dependencies.

```
test/
  unit/                       # no database; pure-function and in-memory coverage
    send.test.js              # normalizePhone + validateInput
    schedule.test.js          # parseScheduledAt / parseRecipientLines / resolveTemplate
    webhook.test.js           # webhookOut.sign (HMAC vectors) + worker surface
    netguard.test.js          # netGuard URL/address classification
    analytics.test.js         # clampDays + read-only service surface
    safeNext.test.js          # open-redirect defenses
    senderName.test.js        # validateSenderName (trim/collapse/control/1-25)
    templates.test.js         # validateTemplate + parseTemplateId + service surface
    proofs.test.js            # sniffMime + proof path/limit helpers (Phase 12)
    billingAccess.test.js     # isAllowedBillingPath + billingRedirectTarget (Phase 13)
    intouch.test.js           # auth digest, RWF math, phone normalize, webhook ACK (Phase 14)
    kitovu.test.js            # signature vector, RWF math, phone normalize, webhook parse (Phase
15)
  integration/                # DB-backed; run against an isolated _test database
    helpers/db.js             # MUST be required first: safety guard + seeds + lock
    credits.test.js           # ledger atomicity / row-lock serialization
    schedule.test.js          # createScheduled / createBulk / pending cap
    scheduler.test.js         # dispatch, retry/backoff, completion
    passwordReset.test.js     # reset lifecycle, single-use race, session destroy
    reaper.test.js            # retention + session pruning selectivity
    senderName.test.js        # engine call + profile_name persistence/failure
    senderNameAddon.test.js   # settlement/renewal, days clamp, rollback, gate, checkout race
    templates.test.js         # CRUD scoping, per-user cap race, duplicate name, prefill
    adminSenderName.test.js   # admin override: bypass, connected gate, persistence
    paymentProofs.test.js     # proof CRUD, caps/quota, admin proof route, multipart CSRF
    billingAccess.test.js     # unlock rule (paid tx / positive balance) over the DB
    paymentLock.test.js       # lock redirect, allowed paths, admin/anonymous bypass
    intouch.test.js           # checkout → webhook → cross-check → settle, sweep healing (Phase
14)
    kitovu.test.js            # checkout → signed request → webhook → settle, sweep (Phase 15)
```

Commands (from the app root):

Command	Scope
`npm test`	164 unit tests, no DB
`npm run test:integration`	174 DB-backed tests
`npm run test:all`	both

`test/integration/helpers/db.js` refuses to run against production: it requires `NODE_ENV !== 'production'`, a `TEST_DB_NAME` ending in `_test`, and a test name different from the `.env` `DB_NAME`. It forces `NODE_ENV=test` and `DB_NAME=$TEST_DB_NAME` before `server/config.js` is loaded, applies the migrations, truncates business tables between tests, and holds a MariaDB advisory lock (`GET_LOCK('znotice_test_suite_lock')`) for each file's lifetime so concurrent `node --test` workers cannot truncate each other's data. See OPERATIONS.md §13 for the one-time test-DB setup.

ZNotice — Security Posture

Defensive summary of the security controls in place, the risks that are known and accepted, and the incident response checklist.

1. Authentication and sessions

Area	Control
Customer API	API key via `Authorization: Bearer zn_live_...` or `X-API-Key`. Keys are random (`zn_live_` + 32 hex chars) and stored **only as a SHA-256 hash** (`api_keys.key_hash`, unique). The full key is displayed once at creation. Revocation is a soft delete (`revoked_at`) and takes effect immediately.
Dashboard accounts	Email + password, bcrypt (cost 12). Registration requires accepting the Terms/AUP; `accepted_terms_at` is recorded.
Sessions	Express sessions stored in MariaDB. Cookie `znotice.sid`: `httpOnly`, `sameSite=lax`, `secure` in production, 30-day rolling max age. Session is regenerated on login/registration to prevent session fixation.
Session invalidation	On each request the user is reloaded; deleted or suspended users have their session destroyed. A dashboard password change destroys every **other** session (current one kept); a password **reset** destroys **all** sessions.
Account recovery	Self-service password reset over email (Phase 8). See §1.1.
Proxy trust	`trust proxy = 1` behind NGINX so secure cookies and real client IPs work.

1.1 Password reset controls (Phase 8)

- **Hashed, single-use tokens.** A reset link contains 32 random bytes (`crypto.randomBytes(32)`, 64 hex chars); only `sha256(token)` is stored in `password_resets`. A used token can never be replayed.
- **Atomic claim.** Consumption runs a conditional `UPDATE ... SET used_at=NOW() WHERE id=? AND used_at IS NULL AND expires_at>NOW()` **inside** the password-change transaction; `affectedRows` must be exactly `1`. Two concurrent submissions of the same token therefore produce exactly one winner (regression test in `test/integration/passwordReset.test.js`).
- **Short expiry.** Tokens expire 30 minutes after issue. Requesting a new link deletes any prior unused token for that user.
- **No enumeration.** `POST /forgot-password` always redirects to the same `?sent=1` page and never reveals whether the address exists. `GET/POST /reset-password` return the same `400` invalid-link page for unknown, used or expired tokens.
- **Rate limits.** Forgot: **5 / 15 min / IP**. Reset: **10 / 15 min / IP**.
- **Session invalidation.** A successful reset calls `destroyAllSessions(userId)` (exact `$.userId` JSON match), signing the user out everywhere, and sends a password-changed notification (fire-and-forget).
- **Password policy.** Minimum 8 characters; maximum **72 bytes** (the bcrypt truncation point) on registration, reset and account change.
- **CSRF.** Both forms are protected by the synchronizer token, and the reset path never logs its query string at the NGINX edge (see §4.2).

2. CSRF, headers and rate limiting

- **CSRF:** synchronizer-token pattern. `_csrf` (form body) or `x-csrf-token` header is required on all state-changing requests. Exempt: `/api/v1/*` (API-key auth) and `/webhooks/*` (machine-to-machine, mounted before the session/CSRF middleware), plus the IntouchPay callback `POST /payments/intouch/webhook` (machine endpoint, mounted before session/CSRF — see §3.7). Comparison is timing-safe. Token minting is **lazy** (Phase 8): a session token is created only for authenticated requests or the auth form pages (`/login`, `/register`, `/forgot-password`, `/reset-password`), so public pages no longer create session rows (see §2.1).
- **Security headers (helmet):** CSP locked to `'self'` plus the Tailwind CDN and Google Fonts; `X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy` also set at the NGINX layer. `x-powered-by` disabled.
- **Rate limits:**
  - Login/register: 10 attempts / 15 min / IP.
  - Forgot password (`POST /forgot-password`): 5 requests / 15 min / IP.
  - Reset password (`POST /reset-password`): 10 submissions / 15 min / IP.
  - Password change (`POST /dashboard/account/password`): 5 attempts / 15 min per IP + user id (slows current-password brute force).
  - All `/api/v1` requests: 120 / min / IP.
  - Sends per API key (and per dashboard user): `send_rate_per_min` (default 60/min), in-memory fixed window.
  - Payment proof uploads (`POST /dashboard/billing/proof/*`): 30 requests / 15 min, keyed by `user:<id>` else `ip:<ip>` (the limiter skips non-POST).
- **Body limits:** JSON and URL-encoded parsers capped at 1 MB; NGINX `client_max_body_size 50m`; multipart proof uploads capped at **5 MB per file** by multer (and re-checked in the service) before CSRF/auth.
- **Inbound webhook auth:** shared secret compared with `crypto.timingSafeEqual`, accepted as a path token or `x-znotice-webhook-secret` header.
- **Outbound webhook auth:** every customer webhook POST carries `X-ZNotice-Signature: sha256=<HMAC-SHA256(secret, rawBody)>` so the receiver can authenticate ZNotice. Secrets are 32 random bytes (64 hex chars), stored per user, and rotatable from the dashboard.

2.1 Session and cache hardening (Phase 8)

- **Lazy CSRF token.** `attachCsrf` exposes `res.locals.csrfToken` as a getter that only mints a token when `needsCsrf(req)` is true (authenticated user, or one of the auth form pages). Before this, Express's `res.locals` copy read every enumerable getter on each render, which created a session row for every anonymous page view. Public pages are now session-free under `saveUninitialized: false`.
- **Anonymous-session pruning.** The reaper deletes expired sessions and anonymous sessions older than 7 days (`sessionsDeleted` in its summary). A one-time cleanup removed ~1,300 rows created before the lazy-token fix.
- **`Cache-Control: no-store`.** Authenticated HTML responses get `no-store` (`server/index.js`), and **all** `/api/v1` responses get it before auth so 401/403 bodies are also uncacheable (`server/routes/api/v1.js`). Static assets are served before the session layer and are unaffected.
- **Error-log redaction.** The global error handler redacts the `token`, `secret`, `key` and `password` query parameters (case-insensitive) before logging, so a reset token or secret cannot reach the journal.

- **NGINX log suppression.** The vhost sets `access_log off` for the exact `location = /reset-password`, so reset tokens in the query string never reach `access.log`.

3. Admin controls and capacity caps

- Admin routes require an authenticated user with `role = admin` (`requireAdmin`); anonymous users are redirected to login, others get 403.
- Admins can suspend/activate users, adjust credits (negative adjustments that would create a negative balance are rejected), approve/fail payments, manage integration orders, force-disconnect sessions, and edit operational settings.
- The **capacity guard** (`max_concurrent_sessions`, default 15) limits pending+connected WhatsApp sessions to protect host RAM. Admins can raise it only with awareness of memory limits.
- Phase 6 adds three queue caps to the settings whitelist: `bulk_max_recipients` (default 1000), `schedule_max_days_ahead` (default 90) and `scheduled_max_pending` (default 5000).
- Phase 10 adds `sender_name_price_cents` (default 1000) and `sender_name_days` (default 30, clamped 1–365 at settlement) to the settings whitelist; both are admin-only.
- Phase 11 adds `templates_max_per_user` (default 200, range 1–10000) to the settings whitelist; admin-only. It bounds per-user template creation.

3.1 Scheduled & bulk sending controls

- **Atomic per-user pending cap.** Enqueue takes a user-row lock (`SELECT ... FOR UPDATE`) around the pending-count check and the batch/item inserts, so concurrent bulk requests for the same user cannot jointly overrun `scheduled_max_pending` (`429 queue_full`).
- **At-most-once claim.** The scheduler moves an item `pending → sending` with a conditional UPDATE *before* calling the engine. Cancels only target `pending`, so a cancel can never race an in-flight send. A crash mid-send is recovered by the reaper as `failed`/`interrupted_by_restart` rather than retried (see §5.9).
- **No credit reservation.** Enqueue only pre-checks the balance; the actual 1-credit charge happens per message at dispatch and is auto-refunded on engine failure. A job can therefore fail item-by-item later without losing credits (see §5.8).
- **Input validation.** `media_url` is capped at 500 characters (shared with immediate send) so an over-long URL can never surface as a DB "data too long" 500; recipients are normalized and deduped; invalid entries reject the whole request before any row is written.
- **Bounded, tenant-scoped listing.** Queue/batch endpoints clamp `limit` and are scoped to the authenticated user (foreign ids return `404`).

3.2 Usage analytics controls (Phase 7)

- **Tenant-scoped queries.** Every customer aggregate filters on `user_id = ?` (the signed-in user or the API-key owner), so one tenant can never read another's analytics. The customer page is behind `requireAuth` and the API endpoint behind API-key auth.
- **Admin-only platform view.** `/admin/analytics` is behind `requireAdmin` (anonymous → login redirect, non-admin → `403`).
- **Parameterized SQL.** All analytics queries use `pool.query(sql, params)` placeholders; there is no string interpolation of user input.

- **Clamped inputs.** `days` is clamped to `1..90` at the route and again in the service; `topUsers` clamps `limit` to `1..50`. Out-of-range values are clamped, not echoed into SQL, which prevents both `INTERVAL NaN DAY` errors and unbounded result/day-array allocation.
- **Read-only.** The analytics service performs only `SELECT`s — it never writes, and there is no background worker. The migration it relies on (`007_analytics_indexes.sql`) is index-only.

3.3 Dedicated sender-name add-on controls (Phases 9–11)

- **Entitlement gate before the engine call.** `POST /dashboard/whatsapp/sender-name` checks `senderName.isSenderNameActive(userId)` (i.e. `users.sender_name_until > NOW()`) **after** the connected-session check and **before** `wpp.setProfileName()` is ever invoked. An inactive/expired add-on redirects to Billing and never touches the WhatsApp engine. The same helper drives the view, so the disabled form and the server enforce the same rule.
- **Server-side price and period.** The purchase route reads `sender_name_price_cents` and `sender_name_days` from the settings cache; the client cannot supply or influence the amount or the period. The amount is written into the transaction at creation.
- **Admin-only settlement.** Entitlements extend only through `payments.markPaid()`, reached from the `requireAdmin`-protected `POST /admin/transactions/:id/paid`. There is no customer-facing route that grants entitlement.
- **Atomic, idempotent purchase.** Checkout locks the user row (`SELECT ... FOR UPDATE`), re-checks for an existing pending `sender_name` transaction, and inserts in the same transaction — concurrent clicks cannot create multiple pending purchases (TOCTOU closed). Settlement uses the single conditional `pending → paid` UPDATE as the concurrency gate, so concurrent admin approvals cannot double-extend the entitlement.
- **Clamped `days`.** `meta.days` is clamped to `1..365` with an explicit finite check; an unset/unparseable value falls back to the configured `sender_name_days`. This prevents both a zero/negative extension and an unbounded one.
- **Rollback safety.** If the entitlement UPDATE throws after the `pending → paid` transition, the transaction is reverted to `pending` (`paid_at = NULL`) and the error rethrown, so an admin can retry and the payment is not lost.
- **Tenant isolation.** `senderName.isSenderNameActive` is called only with the signed-in `req.user.id`; the billing and pay routes scope transactions to `user_id = ?`.
- **Admin override — deliberate, bounded entitlement bypass (Phase 11).** `POST /admin/sessions/:id/sender-name` is behind `requireAdmin` and the global CSRF middleware. It parses the session id as a safe integer, loads the target session with a parameterized `JOIN`, validates the name with the **same** shared `validateSenderName` helper as the customer route, and requires the session to be `connected` before any engine call. It intentionally does **not** call `isSenderNameActive` — support tooling must be able to correct a tenant's profile name without the tenant owning the add-on. This is the **only** sender-name bypass; the customer paywall is unchanged. Engine failures are logged server-side and surfaced to the admin as a generic message (engine-offline gets a distinct friendly message); no raw engine error, stack or secret reaches the UI.

Residual notes.

- The entitlement check is point-in-time at request time. An admin comp or a manual SQL change to `sender_name_until` takes effect on the next request (no cache), but an in-flight change is not re-checked mid-request.

- Changing the sender name mutates the customer's **WhatsApp account profile** and is subject to WhatsApp's own change-rate limits; ZNotice can only surface a generic error when the engine rejects it.
- The public `/pricing` page reads `sender_name_price_cents` / `sender_name_days` at render time (`server/routes/pages.js`), so it reflects admin changes. The `settings` table remains authoritative for what a customer is actually charged (settlement uses the values stored on the transaction).

3.4 Message template controls (Phase 11)

- **Tenant isolation.** Every template query filters on `user_id = ?` in the SQL `WHERE` clause (list, update, delete, prefill), so a user can never read or modify another tenant's template. A foreign/invalid id simply affects 0 rows and `resolvePrefill` returns `null` — no existence oracle.
- **Strict id parsing.** `parseTemplateId` accepts digits only, requires a safe integer `> 0`, and returns `null` otherwise; both the dashboard router and the service use it, so there is no duplicated/looser parsing path.
- **Atomic per-user cap.** Creation runs in one transaction that first locks the user row (`SELECT ... FOR UPDATE`) and then counts the user's templates with a locking read, so concurrent creates cannot jointly exceed `templates_max_per_user` (default 200). Over the cap throws a typed `limit_reached` error and the insert is rolled back.
- **Bounded listing.** `listForUser` caps the result set at **500** rows (`LIMIT ?`, clamped), so a large template set cannot produce an unbounded render.
- **Input validation.** `validateTemplate` rejects control characters and line breaks in the name and media URL, trims/collapses the name, requires a non-empty body ≤4096 chars and an `http(s)` media URL ≤500 chars, mirroring the send pipeline's limits. Names are unique per user (`UNIQUE(user_id, name)`; duplicate → typed `duplicate_name`).
- **Admin-only cap.** `templates_max_per_user` is part of the admin settings whitelist (range 1–10000) and is never client-supplied.

Phase 11 review findings — all resolved before deploy. No Critical or High issues were identified. **P11-01** (Medium) unbounded template creation → fixed by the atomic per-user cap; **P11-02** (Low) unbounded list → fixed by `LIMIT 500`; **P11-03** strict route id parsing; **P11-04** admin sender-name route safe-integer id; **P11-05** `mediaUrl` control-character rejection.

3.5 Payment proof upload controls (Phase 12)

- **Type decided by magic bytes only.** `sniffMime` (`server/services/proofs.js`) accepts JPEG (`FF D8 FF`), PNG (`89 50 4E 47 0D 0A 1A 0A`), WebP (`RIFF....WEBP`) and PDF (`%PDF-`); the client MIME type and filename are **never** trusted for the type decision.
- **Generated names + regex path guard.** `makeStoredName` produces `<16 hex>.<ext>` from `crypto.randomBytes(8)`; `proofPath` re-validates the name against `^[0-9a-f]{16}\.(jpg|png|webp|pdf)$` before joining it to the uploads root. A user-controlled name can never traverse or select a path.
- **Ownership + admin authorization.** The owner routes join `payment_proofs` → `transactions` and filter `t.user_id = req.user.id`; the admin stream route is behind `requireAdmin`. A foreign or missing proof is an indistinguishable `404` (no existence oracle).
- **CSRF ordering rationale.** Multer runs **before** `verifyCsrf` because the global CSRF check reads `req.body._csrf` and `express.urlencoded` cannot parse a multipart body. Parsing the upload into `memoryStorage` first makes the hidden token visible, so the normal synchronizer-token check still applies; nothing is written to disk before CSRF, ownership, status, size and bytes are validated.

- **Multer limits.** `{ fileSize: 5 MB, files: 1, fields: 1, fieldSize: 1024, parts: 3 }` — bounded so busboy cannot buffer unbounded text fields/parts in memory before CSRF/auth (hardening **H1**).
- **Rate limiting.** 30 POSTs / 15 min under the `/dashboard/billing/proof` mount (the delete POST shares the bucket; GET requests are skipped), keyed by authenticated user id else IP. The limiter runs before multer, so an abusive client cannot make the server buffer files.
- **Per-user quota.** 25 files / 50 MB total across all of a user's transactions, enforced by `getUserProofUsage` on the locked connection before insert (hardening **H2**); plus 5 proofs per transaction and 5 MB each.
- **Transaction row lock + disk I/O.** Ownership/status, the caps and the `INSERT` run under `SELECT ... FOR UPDATE`; the file is written **after** commit, so the row lock is never held across disk I/O (hardening **H4**). A write failure best-effort deletes the committed row and any partial file; no code path unlinks a file after a successful commit.
- **Serving headers.** `Content-Type` comes from the byte-sniffed DB value; `Content-Disposition: inline; filename="proof-<id>.<ext>"` (generated); `Cache-Control: private, no-store`; and `Content-Security-Policy: default-src 'none'; sandbox`. With NGINX's `X-Content-Type-Options: nosniff`, this neutralizes an HTML polyglot served as an inline proof (see §5.16).
- **Upload/delete only while pending.** Both re-assert ownership and `t.status = 'pending'` in SQL; proofs stay readable by owner + admin after settlement for audit.
- **Locked users may still upload.** `/billing` is an allowed prefix under the Phase 13 lock, so an unpaid user can attach proof to the pending payment that unlocks them (hardening **H3**).

3.6 Billing lock for unpaid accounts (Phase 13)

- **Isolation of unpaid accounts.** `requirePayment` (`server/middleware/paymentLock.js`) is mounted at `/dashboard` immediately after `verifyCsrf` and before every dashboard router. A non-admin with no `paid` transaction and no positive credit balance is confined to the payment flow: `/billing`, `/account` and `/integration` (paths relative to the `/dashboard` mount). Every other `/dashboard/*` path gets an error flash and a `302` to the pending transaction's pay page or `/dashboard/billing`.
- **Admin bypass.** `role = 'admin'` passes with no DB read and is never locked. The unlock decision for non-admins is `EXISTS(paid transaction) OR SUM(credit_ledger.delta) > 0` (`server/services/billingAccess.js`), resolved in one query.
- **Anonymous pass-through.** Anonymous requests are not handled here; the downstream `requireAuth` owns the login redirect, so there is a single auth gate and no redirect loop.
- **`/api/v1` intentionally not gated (designed).** The lock is dashboard-only. API-key sends already fail with `402 insufficient_credits` at a zero balance, so gating them would only change the error code while breaking key-based integrations. `/admin/*` is behind `requireAdmin` and is unaffected.
- **Defensive path check.** `isAllowedBillingPath` accepts only strings and returns `false` for anything else, so malformed input can only lock the request down, never unlock it.

3.7 IntouchPay webhook and settlement controls (Phase 14)

The IntouchPay gateway is the first live (non-manual) payment provider. Its callback is fundamentally different from the signed WPPConnect webhook, so the controls differ too.

- **Unsigned callback — no signature verification possible.** IntouchPay does not sign its callbacks, so the app cannot authenticate them. **Mitigation: ground-truth cross-checking.** The handler never

trusts the callback on its own; it always calls `gettransactionstatus` (3 bounded retries) to confirm the gateway actually settled the transaction, and only then settles locally.

- **Pure-ACK response contract.** The webhook always responds `200 {"message":"success","success":true,"request_id":"<uuid>"}` — for a valid, malformed, unknown or mismatched payload alike. This prevents both **validation leakage** (an attacker probing the endpoint learns nothing) and **retry storms** (the provider always sees success, so it never retries a callback that we deliberately did not act on).
- **Secrets stay inside the SHA256 hash.** The `username + account_number + partner_password + timestamp` digest means the account number and partner password are **never transmitted** in any request — only the hash is sent. They live only in `.env` (`INTOUCH_*`).
- **No secrets/tokens in logs.** Webhook payloads and gateway responses are logged without credentials; the `X-Trace-Id` correlation id carries no secret. The global error-log redaction (§2.1) still applies to any `token` / `secret` / `key` / `password` query parameter.
- **Strict amount matching.** Settlement requires the cross-checked status to be settled **and** the returned RWF amount and currency to equal the transaction `meta` exactly (whole RWF, off-by-one blocked), so a manipulated or rounded amount can never settle for the wrong value.
- **Mounted pre-CSR as a machine endpoint.** `POST /payments/intouch/webhook` is mounted **before** the session/CSRF middleware, like `/webhooks/*`, because the provider cannot hold a CSRF token. It requires **no auth** (the callback is unsigned), which is safe precisely because the cross-check is the real trust anchor, not the request itself.
- **Sweep as a recovery control.** `server/services/intouchSweep.js` runs every 5 minutes and cross-checks pending Intouch transactions older than 2 minutes, confirming settled ones. This bounds the window in which a lost/early callback would otherwise leave a settled payment unconfirmed, and it shares the same strict matching + idempotent `pending → paid` gate.
- **No processor refund API — local-ledger reversals only.** IntouchPay exposes no reversal endpoint, so the app never initiates a refund against the gateway. Any refund/adjustment is a local credit-ledger entry (admin adjustment), exactly like `manual` — the same surface is also covered by the existing controls in §3.3/§3.6 and BUSINESS.md §6.
- **Scope.** IntouchPay is credit top-ups only; the sender-name add-on and integration service remain on the manual (admin-approved) flow, so no unsigned-callback settlement path touches entitlements or orders.

3.8 KitovuPay webhook and settlement controls (Phase 15)

The KitovuPay gateway is the second live (non-manual) payment provider. Like IntouchPay, its callback is unsigned; unlike IntouchPay, its outbound requests are HMAC-signed.

- **Unsigned callback — no signature verification possible.** KitovuPay does not sign its callbacks, so the app cannot authenticate them. **Mitigation: ground-truth cross-checking.** The handler never trusts the callback on its own; it always calls `GET /merchant-transaction/payment/:externalId` (3 bounded retries) to confirm the gateway actually settled the transaction, and only then settles locally.
- **Pure-ACK response contract.** The webhook always responds `200 {"message":"success","success":true,"request_id":"<uuid>"}` — for a valid, malformed, unknown or mismatched payload alike. This prevents both **validation leakage** and **retry storms**.
- **HMAC-signed outbound requests.** Every collection POST carries an `x-api-signature` = HMAC-SHA256 over the compact `{method,path,body}` envelope keyed with the API secret; `x-api-timestamp` is Unix milliseconds as a string. The secret only feeds the signature and is **never transmitted**; it lives only in `.env` (`KITOVUPAY_API_SECRET`).

- **No secrets/tokens in logs.** Webhook payloads and gateway responses are logged without credentials (only ids/references).
- **Strict amount matching.** Settlement requires the cross-checked status to be `APPROVED`; a webhook-supplied amount must equal the transaction `meta` exactly (whole RWF, off-by-one blocked), so a manipulated amount can never settle for the wrong value.
- **Unique 6-digit externalId.** The gateway's lookup key is checked for uniqueness in `transactions.provider_ref` before sending, and a `DUPLICATE_EXTERNAL_ID` rejection allocates a fresh ref and retries (bounded) — this also keeps the shared UAT merchant account (with zeron-xs) collision-free.
- **Mounted pre-CSR as a machine endpoint.** `POST /payments/kitovu/webhook` is mounted **before** the session/CSRF middleware and requires **no auth**; the cross-check is the real trust anchor, not the request itself.
- **Sweep as a recovery control.** `server/services/kitovuSweep.js` runs every 5 minutes and cross-checks pending Kitovu transactions older than 2 minutes, confirming settled ones, bounded by the same idempotent `pending → paid` gate.
- **Scope.** KitovuPay is credit top-ups only; the sender-name add-on and integration service remain on the manual flow, so no unsigned-callback settlement path touches entitlements or orders.

4. Host and data handling

- The app runs as `www-data` with systemd hardening (`NoNewPrivileges`, `PrivateTmp`).
- `.env` (DB credentials, session/webhook/engine secrets) is mode `0600` `www-data:www-data` and is never committed.
- Backups include `.env`, so they are stored `0640 root:www-data` with `0750` directories — not world-readable. The backup script never prints credentials and passes DB credentials to `mysqldump` via a temporary `0600` defaults file (never in `argv`).
- SSH is key-based only; UFW allows 22/80/443; Fail2Ban protects SSH.
- MariaDB, the Node app and WPPConnect all bind to loopback; only NGINX is publicly exposed.

4.1 Egress guard (SSRF protection)

Server-side fetches of user-supplied URLs are protected by a shared egress guard (`server/services/netGuard.js`). It:

- allows only `http` / `https` on the default ports (`80` / `443`), rejects URLs with embedded credentials, and rejects `localhost` / `.local` / `.internal` hostnames;
- resolves the hostname and rejects the URL if **any** resolved address is loopback, private, link-local, CGNAT, ULA, multicast, reserved or an IPv6-mapped form of those;
- **pins** the validated IP via a custom socket `lookup` so a DNS-rebinding answer cannot be used between the check and the connect.

The guard is used by **media fetching** (`server/services/mediaFetch.js`, which adds its own content-type/size/redirect limits) and by **outbound customer webhooks** (`server/services/webhookOut.js`, which re-validates the endpoint before every delivery). `media_url` and webhook endpoints therefore share the same public-IP-only rule.

4.2 Host hardening (2026-09-11)

- **UFW:** stale allows for **3001/tcp, 3005/tcp and 4000/tcp** (v4 + v6) were removed, so those app backends are no longer directly reachable from the internet. The GameBet `next-server` on 3001 still serves through NGINX on loopback. Rollback: `ufw allow <port>/tcp`.
- **Sudoers:** a duplicate `kali ALL=(ALL) NOPASSWD:ALL` grant and the empty-group `kali-grant-root` file were removed; `kali` passwordless sudo is retained and `visudo -c` is clean.
- **NGINX:** `access_log off` on the exact `location = /reset-password` keeps live reset tokens out of `access.log` (the pre-existing `/webhooks/` location does the same for the embedded webhook secret).

Full evidence and rollback: OPERATIONS.md §14 and `/root/backups/host-hardening-2026-09-11/CHANGES-AND-ROLLBACK.md`.

5. Known risks (explicitly accepted)

5.1 Unofficial WhatsApp protocol — ban risk

ZNotice uses WPPConnect, an **unofficial** WhatsApp Web automation client. This may violate WhatsApp's Terms of Service and can result in a number being rate-limited, restricted, or permanently banned. This risk is **disclosed to customers in `/terms`** (§3). ZNotice is not affiliated with WhatsApp/Meta. Customers are responsible for consent/opt-in and for complying with the WhatsApp Business Messaging Policy.

5.2 Webhook secret is embedded in the URL path

The engine cannot send custom headers, so the shared secret is part of the webhook URL (`/webhooks/wppconnect/<secret>`). Risks: the secret can appear in engine logs or proxy logs. **Rotation caveat:** rotating `WEBHOOK_SECRET` in `.env` requires an app restart, and existing engine sessions keep calling the **old URL** until they re-run `start-session` (disconnect + reconnect). See OPERATIONS.md §10 "Webhook secret rotation".

5.3 In-memory rate limiter resets on restart

The per-key/per-user send limiter (`services/send.js`) and the WPPConnect bearer-token cache are in-process. A restart clears the send counters, so a burst can briefly exceed the intended per-minute limit. This is acceptable for abuse control on a single node; a distributed limiter (Redis) would be needed for strict enforcement across restarts/replicas.

5.4 Stale pending sessions occupy capacity

A `pending` `wa_sessions` row (user opened the connect flow but never scanned) counts against `max_concurrent_sessions`, potentially blocking new connections. **Mitigation:** the stale-session reaper (`server/services/reaper.js`) runs every 5 minutes and marks `pending` rows untouched for more than 10 minutes as `disconnected` (unless the engine reports `CONNECTED`), and marks `connected` rows `disconnected` when the engine reports `CLOSED`. Admins can still force-disconnect a specific session at **Admin → Sessions**.

5.5 Engine `close-session` 500 bug

`POST /close-session` can return HTTP 500 (`req.client.close is not a function`) **after** the engine has already closed the session. The app mitigates this by probing `status-session` and treating `disconnected` as success. Operators should verify with the status probe before retrying a disconnect; do not assume the session is still live.

5.6 RAM headroom constraints

Each active WhatsApp session (a Chromium/WhatsApp Web context) consumes roughly **300–500 MB**. With ~7.8 GB total RAM and other services on the host, the default cap of 15 sessions is deliberately conservative. Symptoms of pressure: health-check `WARN low available RAM` (<400 MB), app restarts, and engine instability. **Mitigation:** keep the session cap within measured RAM, monitor `free -m`, and force-disconnect idle sessions. Never restart the engine to relieve memory pressure without accepting that all tenants must re-scan.

5.7 Other notes

- `transactions.status` includes `refunded` but no route sets it; refunds are recorded as admin ledger adjustments (see BUSINESS.md §6).
- If `SESSION_SECRET` is not set, the config falls back to a known insecure development value. **Verify `SESSION_SECRET` is set in `.env`** in production.

5.8 No credit reservation at enqueue

Scheduled/bulk jobs do not reserve credits. If the balance is spent before an item dispatches, the item fails with `insufficient_credits` (no credit is lost, since none was charged). This avoids holding funds against jobs that may never run, at the cost of possible partial delivery. **Mitigation:** the enqueue pre-check (`balance ≥ total`) catches the common case; monitor failed items with `last_error='insufficient_credits'`.

5.9 At-most-once dispatch can drop a message

A process crash after the engine accepted a send but before the DB transition leaves the item in `sending`; the reaper marks it `failed` (`interrupted_by_restart`) instead of retrying. This is a deliberate **at-most-once** choice: a rare dropped message is preferred over a duplicate. There is no automatic re-send; operators must confirm with the recipient before re-enqueuing (see OPERATIONS.md §10).

5.10 Scheduler fairness under large backlogs

Each tick selects at most 25 items per user and a global `LIMIT 50`, so with 3+ tenants each holding a large due backlog the third tenant may wait for a later tick. This bounds starvation but does not eliminate latency under sustained overload. **Mitigation:** raise host capacity / `send_rate_per_min`, or spread large bulk jobs over time.

5.11 Anonymous-session growth — mitigated (Phase 8)

Resolved. Previously the CSRF token was attached to every render, so `express-mysql-session` persisted a `sessions` row for anonymous/bot traffic even with `saveUninitialized: false`. Phase 8 makes token minting lazy (§2.1), so public pages no longer create rows, and the reaper now deletes

expired sessions and anonymous sessions older than 7 days. A one-time cleanup removed ~1,300 accumulated rows. Residual risk is minimal; monitor the table size if bot traffic is unusually high.

5.12 L2 — Authenticated responses now send `Cache-Control: no-store` — closed

Closed (Phase 8). Authenticated HTML responses (`server/index.js`) and all `/api/v1` responses including 401/403 (`server/routes/api/v1.js`) now send `Cache-Control: no-store`, so a browser cache / bfcache cannot retain a signed-in page or account-scoped JSON after logout. Static assets are served before the session layer and keep their normal caching.

5.13 Register user enumeration — open (pre-existing)

`POST /register` returns `409` with "An account with that email already exists" for an address that is already registered. This lets an attacker confirm whether an email has a ZNotice account. **Status:** open, pre-existing (not introduced in Phase 8). The forgot-password flow is deliberately non-enumerating; the register response is the remaining asymmetry. A future fix would return the same generic success/verification page regardless of existence (email verification). Until then, treat registered emails as potentially discoverable.

5.14 Forgot-password response timing — open (Low)

`POST /forgot-password` returns an identical status and body whether or not the account exists, but the existing-account path does extra work (lookup + token insert + email) and responds about **15 ms** slower. This is a low-bandwidth timing side channel for user enumeration. **Status:** open; the visual/HTTP response is identical, so it is only observable with careful timing measurement.

5.15 SMTP credentials — configured (closed 2026-09-11)

Production email reuses the existing Hostinger mailbox for `info@consultbofa.com` (`smtp.hostinger.com:465`, implicit TLS); credentials live only in `.env`. Password-reset and password-changed emails are delivered (verified with a live test send). If the variables are ever removed, `send()` returns `email_not_configured` and the reset UI still shows its generic message — see OPERATIONS.md §12.

5.16 Inline PDF proofs — polyglot risk (mitigated)

PDF proofs are served **inline**. A malicious PDF could embed active content, or be a polyglot that is also valid HTML/JS. **Mitigations:** the served `Content-Type` comes from the server-side magic-byte sniff, the response carries `X-Content-Type-Options: nosniff` (NGINX) and `Content-Security-Policy: default-src 'none'; sandbox`, and the `Content-Disposition` filename is generated. A PDF rendered in the browser's sandboxed viewer cannot run page script in the ZNotice origin. **Status:** mitigated; keep the CSP/nosniff headers in place if the serving routes change.

5.17 Open registration + per-account proof quota (open)

Registration is open and there is **no email verification** (see §5.13). The proof quota (25 files / 50 MB per user) is therefore per *account*, not per verified identity, so an attacker can create many accounts to obtain many quotas. **Mitigations in place:** the 30/15-min upload rate limit, magic-byte typing, the storage caps, and the fact that files are only reachable through authenticated routes. **Status:** open, pre-existing; a real fix is email verification / CAPTCHA at registration.

5.18 IPv6 rate-limit granularity (open)

The proof-upload limiter keys anonymous requests on the raw `req.ip`. For IPv6 a client controls a large `/64`, so rotating source addresses within that allocation can evade a per-address bucket. Authenticated uploads are keyed on `user:<id>` and are unaffected. **Status:** open, Low (the route still requires an authenticated session to succeed; the limiter is a secondary abuse control). A stricter key would normalize IPv6 to a `/56` / `/64` prefix.

5.19 Payment-proof files excluded from backups (open)

The automated app archive excludes `uploads/`, so the `payment_proofs` DB rows are backed up but the file bytes are not; a DB-only restore leaves rows pointing at missing files. **Status:** open operational gap — see OPERATIONS.md §15.4 for the retention/backup recommendation. Proof files are the only business data not covered by the backup.

5.20 IntouchPay webhook is unsigned (open, accepted)

The IntouchPay callback carries no signature, so the app cannot authenticate it. This is a limitation of the provider, not a ZNotice omission. **Mitigation:** settlement is gated on a `gettransactionstatus` cross-check (never on the callback alone), and the endpoint returns a pure ACK for every payload, so a forged or garbage callback can at most trigger a status lookup — it can never settle a transaction by itself. **Residual:** the endpoint is publicly reachable and unauthenticated, so it can be probed or spammed; the bounded-retry cross-check and the ACK contract keep the per-request cost low, but an attacker can still cause extra `gettransactionstatus` calls. **Status:** open, accepted — see §3.7 for the full control list.

5.21 KitovuPay webhook is unsigned (open, accepted)

The KitovuPay callback carries no signature, so the app cannot authenticate it. This is a limitation of the provider, not a ZNotice omission. **Mitigation:** settlement is gated on a `GET /merchant-transaction/payment/:externalId` cross-check (never on the callback alone), and the endpoint returns a pure ACK for every payload, so a forged or garbage callback can at most trigger a status lookup — it can never settle a transaction by itself. **Residual:** the endpoint is publicly reachable and unauthenticated, so it can be probed or spammed; the bounded-retry cross-check and the ACK contract keep the per-request cost low, but an attacker can still cause extra status calls. **Status:** open, accepted — see §3.8 for the full control list.

6. Incident response checklist

Use in order; stop when the incident is contained.

1. **Confirm and scope**
   - `journalctl -u znotice --since "1 hour ago"` and `docker logs --tail=200 wppconnect`.
   - Identify affected user(s), API key(s) and session(s) in the database.
2. **Suspend the abusive account**
   - **Admin → Users → Suspend** (or `UPDATE users SET status='suspended' WHERE id=?`).
   - Suspension immediately blocks sends (`403 account_suspended`) and API key authentication.
3. **Rotate credentials / keys**
   - Revoke the affected API keys from **Admin → Users** / user dashboard (soft-delete via `revoked_at`).

- If a secret is suspected leaked, rotate `WEBHOOK_SECRET` (and `WPPCONNECT_SECRET_KEY` if applicable) in `.env`, restart the app, and force reconnect of sessions so the engine picks up the new webhook URL.
- If a dashboard password is compromised, reset it and destroy sessions (restart clears in-memory state; delete the user's rows from the `sessions` table to force logout).

4. **Force-disconnect sessions**
   - **Admin → Sessions → Disconnect** to free capacity slots and stop a session that may be spamming. Verify with the engine status probe.
5. **Preserve evidence**
   - Copy relevant journal lines, engine logs and DB rows before cleanup.
6. **Restore if data integrity is affected**
   - Use the daily backup: restore the database and/or app archive (OPERATIONS.md §5). Prefer a clean restore over patching live data.
7. **Follow up**
   - Re-enable the account only after remediation.
   - Record the incident, root cause, and any setting changes (for example a lower `send_rate_per_min` or `max_concurrent_sessions`).

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).

ZNotice — Changelog

Build history of the ZNotice SaaS. Dates are phase completion dates; the full stack was deployed to production by **2026-09-10**. Version follows the `package.json` `0.1.0` line.

Phase 14 — IntouchPay mobile money payments (MTN MoMo / Airtel Money) — 2026-09-16

Adds a live **IntouchPay** mobile-money gateway (Rwanda: MTN MoMo and Airtel Money), the first real (non-manual) payment provider. Customers pay instantly from the Billing page and the transaction is auto-confirmed by cross-checking the gateway — no admin approval and no proof upload required for credit top-ups. Live mode, with keys supplied by the operator.

Added

- **Provider** `intouch` registered in `server/services/payments/index.js` alongside `manual`. The shared `settlePaid()` helper gained an optional credit-ledger **note override** so Intouch settlements can tag the ledger entry distinctly.
- **New gateway service** `server/services/payments/intouch.js` (uses the global `fetch`, no new dependencies). Auth is a **SHA256 hex hash** over `username + account_number + partner_password + timestamp` (raw concatenation, UTC `YYYYMMDDHHMMSS`); the account number and partner password never leave the hash. Requests hit `POST /requestpayment/` (async, `Pending` `1000`) and `POST /gettransactionstatus/` (no amount) against the live `https://www.intouchpay.co.rw/api` or sandbox `https://developer.intouchpay.co.rw/api/v1/sandbox`, plus `POST /getbalance/` for ops. Rwandan numbers are normalized to `2507XXXXXXXX`; provider refs are `ZNI-<id>-<8hex>`; every request carries an `X-Trace-Id` header; 30s timeout.
- **Dashboard Intouch checkout.** A "Pay instantly with MTN MoMo / Airtel Money" card on the Billing page (amount + phone, with an RWF hint at the operator rate). Routes `POST /dashboard/billing/intouch/checkout`, `POST /dashboard/billing/intouch/:id/start` (re-request a USSD prompt) and `POST /dashboard/billing/intouch/:id/status` (manual status check). The pay page gained an Intouch branch with no manual instructions / proof UI.
- **Currency conversion.** USD prices are converted to whole RWF at checkout using the admin-editable `intouch_rwf_rate` (float, default `1477.29`, migration `013_intouch.sql`; `settings.getFloat()` added). Transaction `meta` carries `{ currency:'RWF', amount_rwf, rate, intouch:{ phone, tx_id, provider, statusdesc } }`; the `transactions` row stays in USD (`amount_cents`).
- **Webhook** `POST /payments/intouch/webhook` (mounted **before** session/CSRF, no auth — the callback is **unsigned**). The payload is wrapped in a `jsonpayload` field (object or JSON string). See Security below for the cross-check/ACK contract.
- **Recovery sweep** `server/services/intouchSweep.js`: every **5 minutes** cross-checks pending `intouch` transactions older than **2 minutes** and confirms settled ones (heals lost/early callbacks).
- **Admin UI.** The pending-payments table shows a provider chip; `intouch` rows show a muted "auto-confirmed by InTouch" hint instead of Approve/Reject.

Changed

- `server/services/payments/index.js` now resolves `providers = { manual, intouch }`.

- Credit top-ups via Intouch settle through the same `settlePaid()` path as manual payments, but with the RWF amount and provider metadata attached.
- The sender-name add-on and the integration service remain **manual only** (no Intouch checkout), so their flows are unchanged. **No API v1 changes.**

Security

- The webhook is unsigned, so every callback is **cross-checked** against `gettransactionstatus` (3 bounded retries) before any settlement; the response is always a pure ACK `200 {"message":"success","success":true, "request_id":"<uuid>"}`, and malformed/unknown/mismatched payloads are logged and ACKed `200` rather than leaking validation detail or retrying. Terminal failures are recorded in `meta.failure_reason`.
- Only the account number + partner password are hashed into the auth digest — neither is ever transmitted in cleartext.
- Whole-RWF amounts are strictly matched (currency + amount) at confirmation, so an off-by-one or mismatched amount can never settle.
- See SECURITY.md §3.7 for the full control list.

Tests: `npm test` → **136 unit tests**; `npm run test:integration` → **149 DB-backed tests**. New coverage: `test/unit/intouch.test.js` (auth digest, RWF amount/currency math, phone normalization, webhook payload unwrapping/ACK) and `test/integration/intouch.test.js` (checkout → webhook → cross-check → settlement, sweep healing, `failure_reason` recording).

Phase 15 — KitovuPay mobile money (MTN MoMo) — 2026-09-18

Adds a second live mobile-money gateway, **KitovuPay** (Rwanda, MTN MoMo), alongside IntouchPay. Customers pay instantly from the Billing page and the transaction is auto-confirmed by cross-checking the gateway — no admin approval and no proof upload for credit top-ups. The wire protocol is ported from the reference Laravel integration (zeron-xs `KitovuService`); the UAT merchant account is shared with zeron-xs, so the 6-digit external IDs must be unique.

Added

- **Provider `kitovu`** registered in `server/services/payments/index.js` alongside `manual` and `intouch`. `providers = { manual, intouch, kitovu }`.
- **New gateway service `server/services/payments/kitovu.js`** (global `fetch`, no new dependencies). Auth is an **HMAC-SHA256 signature** (lowercase hex) over the compact JSON envelope `{"method":"POST","path":"<path>","body":{...}}` keyed with the API secret; `x-api-timestamp` is Unix **milliseconds as a string** (`Date.now().toString()`). GET status uses only `x-api-key`. Collection hits `POST /merchant-transaction/merchant/collection/mobile_money` (`amount`, `payment_rail:'mtn'`, `sender_account_issuer:'mtn'`, `sender_account_name`, `sender_account_number`, `description`, `callbackUrl`, `externalId`); status is `GET /merchant-transaction/payment/:externalId`. 429 responses are retried with bounded exponential backoff (3 attempts, 250ms/500ms); 30s timeout. The `description` is sanitized to plain text (symbols break the USSD push).
- **6-digit `externalId` allocation.** The reference is a unique 6-digit string (the gateway's lookup key). It is checked for uniqueness in `transactions.provider_ref` before sending, and a gateway rejection whose message mentions `DUPLICATE_EXTERNAL_ID` allocates a fresh ref and retries (bounded). `provider_ref` stays `NULL` until a collection request is sent.

- **Dashboard Kitovu checkout.** A "Pay instantly with MTN MoMo (KitovuPay)" card on the Billing page (amount + phone + optional account name, RWF hint). Routes `POST /dashboard/billing/kitovu/checkout`, `POST /dashboard/billing/kitovu/:id/start` (re-request) and `POST /dashboard/billing/kitovu/:id/status` (manual status check). The pay page gained a Kitovu branch with no manual instructions / proof UI.
- **Currency conversion.** USD prices convert to whole RWF at the operator rate `kitovu_rwf_rate` (float, default `1477.29`, migration `014_kitovu.sql`; `settings.getFloat()`). `meta` carries `{ currency:'RWF', amount_rwf, rate, kitovu:{ phone, tx_id, status, requested_at } }`; the `transactions` row stays in USD.
- **Webhook** `POST /payments/kitovu/webhook` (mounted **before** session/CSRF, no auth — the callback is **unsigned**). See Security below.
- **Recovery sweep** `server/services/kitovuSweep.js`: every **5 minutes** cross-checks pending `kitovu` transactions older than **2 minutes** and confirms settled ones.
- **Admin UI.** The pending-payments table shows a provider chip; `kitovu` rows show a muted "auto-confirmed by KitovuPay" hint instead of Approve/Reject.

Changed

- `server/services/payments/index.js` now resolves `providers = { manual, intouch, kitovu }`.
- Credit top-ups via Kitovu settle through the same `settlePaid()` path as manual/intouch payments, with the ledger note `kitovu top-up verified`.
- The sender-name add-on and the integration service remain **manual only**. **No API v1 changes.**

Security

- The webhook is unsigned, so every callback is **cross-checked** against `GET /merchant-transaction/payment/:externalId` (3 bounded retries) before any settlement; the response is always a pure ACK `200 {"message":"success","success":true,"request_id":"<uuid>"}`. Terminal failures and amount mismatches are recorded in `meta.failure_reason`.
- The API secret only feeds the HMAC signature and is never transmitted; only ids/references are logged.
- Whole-RWF amounts are strictly matched at confirmation, so an off-by-one amount can never settle. See SECURITY.md §3.7/§5.20.

Tests: `npm test` → **164 unit tests**; `npm run test:integration` → **174 DB-backed tests**. New coverage: `test/unit/kitovu.test.js` (signature vector cross-validated against the PHP `hash_hmac` reference, compact JSON, timestamp-ms string, phone normalization, 6-digit externalId, webhook payload parse, status mappings, RWF math, description sanitization) and `test/integration/kitovu.test.js` (checkout → signed request → webhook → cross-check → settle, amount mismatch, sweep healing, duplicate-externalId retry, ACK contract).

Client-facing PDF documentation — 2026-09-16

Client-facing PDF exports of this documentation set: a combined 94-page volume (cover + table of contents + 7 chapters) plus seven individual chapter PDFs, built from the markdown sources by an idempotent pandoc → weasyprint pipeline and served as static downloads at `/docs/pdf/`.

Added

- **Build pipeline** `docs/pdf-src/`. `build.sh` (idempotent — wipes and regenerates `pdf-src/build/` and all of `docs/pdf/`), `print.css` (print stylesheet, copied next to the generated HTML so the

pandoc `--css` link resolves inside `build/`), and `front-matter.md` (cover, document-control table, chapter map, and confidentiality note). Pipeline: pandoc `gfm → html5` (`--standalone`, `--syntax-highlighting=none`), weasyprint for PDF rendering, and poppler `pdfinfo` for the page-count summary.

- **Deliverables in `docs/pdf/`.** Combined `ZNotice-Complete-Documentation-<date>.pdf` (94 pages: cover + TOC + 7 chapters) plus 7 individual chapter PDFs: Overview 2p, Business Model 6p, API Reference 12p, Architecture 21p, Security 11p, Operations Runbook 20p, Changelog 14p (page counts from the 2026-09-14 build).
- **Static serving of `GET /docs/pdf/*.pdf`.** `express.static` route rooted at `config.paths.docsPdf` (overridable via `DOCS_PDF_DIR`), index disabled, dotfiles denied, and `Cache-Control: public, max-age=1d`. A "Documentation downloads (PDF)" section on the `/docs` page lists the files; names are derived by a strict filename whitelist in `server/services/pdfDocs.js`. Directory traversal is blocked by `express.static` / `send`. Plain links and downloads only — no CSP or security-policy changes required.
- **Test coverage.** New `test/unit/pdfDocs.test.js` (node:test) and `test/integration/docsPdf.test.js` (real HTTP over an ephemeral port using a `DOCS_PDF_DIR` fixture; asserts `200` + `application/pdf` for whitelisted files and `404` for non-matching names, dotfiles and encoded traversal).

Changed

- **`docs/pdf-src/build.sh` hardening.** `DATE` is now overridable via `$1` (defaults to today); the cover version line now interpolates `VERSION_LABEL` / `VERSION_DATE` instead of a hardcoded string.
- **`docs/README.md`.** The quick-facts table gained a PDF-docs row, and the intro now mentions the downloadable PDF set.
- **`docs/pdf-src/front-matter.md`.** Document-control dates moved to 2026-09-16 (`v0.1.0 (Phase 13, incl. follow-ups through 2026-09-16)`).

Tests: `npm test` → **113 unit tests**; `npm run test:integration` → **127 DB-backed tests**.

Register QR share link — 2026-09-14

Adds a "scan to register" QR code that encodes a short link redirecting to the registration page, so the owner can print or share it anywhere ("scan to sign up"). Pure JS, no native dependencies and no NGINX/systemd changes.

Added

- **Dependency `qrcode`** (`^1.5.4`) as a runtime dependency. Installed as `www-data` so `node_modules/` / `package.json` / `package-lock.json` ownership stays consistent.
- **New service `server/services/qr.js`.** `renderPng(url)` returns a `Promise<Buffer>` (1024px PNG, margin 2, error-correction level M, black on white) and `renderSvg(url)` returns a `Promise<string>` (SVG, margin 2, level M). Both share the same quiet-zone/EC options.
- **Public routes in `server/routes/pages.js`** (same-origin, no auth):
  - `GET /join` → `302` redirect to `/register`.
  - `GET /join/qr` → renders the new `server/views/join-qr.ejs` share page (dark theme, `partials/head` + `partials/foot`, no inline scripts/styles).
  - `GET /join/qr.png` → `200 image/png` of `${APP_URL}/join` with `Cache-Control: public, max-age=86400`.

  - `GET /join/qr.svg` → `200 image/svg+xml; charset=utf-8` with the same cache header. Both image routes funnel render failures to `next(err)`.

- **New view `server/views/join-qr.ejs`.** 256px white/rounded QR image (`/join/qr.png`), the absolute short link (`joinUrl` from the route, derived from `config.appUrl` with trailing slashes stripped), an "Open registration" button linking to `/join`, and a print/share note. CSP-safe (no inline scripts or styles).
- **Landing CTA QR card** in `server/views/landing.ejs`: a compact, responsive card with a 128px QR image (`/join/qr.png`), the caption "Scan to create an account" and the link text `znotice.consultbofa.com/join`, using the existing Tailwind style language. `npm run build:css` recompiled `public/css/tailwind.css` so the new utility classes are present.

Tests

- `test/unit/qr.test.js` (node:test): `renderPng` resolves to a Buffer with the PNG magic bytes (`89 50 4E 47`); `renderSvg` resolves to a string containing `<svg`.
- `test/integration/joinQr.test.js`: real-HTTP over an ephemeral loopback port (`fetch`), following the `paymentProofs` pattern (`./helpers/db` first). Covers `/join` (302 + `Location: /register`), `/join/qr.png` (200, `image/png`, non-trivial body), `/join/qr.svg` (200, contains `<svg`), `/join/qr` (200 HTML referencing `/join/qr.png`) and `/` (200 landing HTML referencing `/join/qr.png`). No DB rows are created.

Tests: `npm test` → **106 unit tests**; `npm run test:integration` → **122 DB-backed tests** (both fully passing).

Security hardening — Phase 12/13 follow-up — 2026-09-13

Targeted fixes from the Phase 12 (proof upload) / Phase 13 (billing lock) security review. No auth/CSRF/traversal flaws were found; these close resource and UX gaps.

- **H1 — strict multer limits.** `server/middleware/uploadProof.js` now sets `{ fileSize: MAX_PROOF_BYTES, files: 1, fields: 1, fieldSize: 1024, parts: 3 }` instead of only `fileSize`/`files`, so busboy can no longer buffer unbounded text fields/parts in memory before CSRF/auth. The upload form sends exactly one `_csrf` field + one file. The friendly multer error mapper moved next to the instance (`uploadProof.proofUploadErrorMessage`) and now covers `LIMIT_PART_COUNT`, `LIMIT_FIELD_COUNT`, `LIMIT_FIELD_KEY` and `LIMIT_FIELD_VALUE`; `server/index.js` still flashes + redirects for limit errors.
- **H2 — per-user proof storage quota.** `server/services/proofs.js` adds `MAX_PROOFS_PER_USER = 25` and `MAX_USER_PROOF_BYTES = 50 MB`, plus `getUserProofUsage(userId, executor?)` (single query joining `payment_proofs` to `transactions` on `user_id`) and `assertUserProofQuota(usage, incomingBytes)` (typed `user_limit_reached`, user-safe message). Enforced in `POST /dashboard/billing/proof/:id` inside the locked section before the file write; the per-transaction cap stays at 5.
- **H3 — locked users can order integration.** `/integration` was added to `ALLOWED_PREFIXES` in `server/services/billingAccess.js`; the integration request form is part of the payment flow (it creates a pending $20 payment) and is shown on the Billing page, so locked users may reach it.
- **H4 — row lock no longer held across disk I/O.** The proof route now does begin → `SELECT ... FOR UPDATE` → per-tx cap → per-user quota → `INSERT` → commit → release, then writes the file outside the transaction. A write failure best-effort deletes the committed row and any partial file. No code path deletes a file after a successful commit (the old catch-all compensation could fire on a release error). A

brief window where the row exists before the file is written is acceptable — serving returns 404 until the write completes.

Tests: `npm test` → **104 unit tests**; `npm run test:integration` → **117 DB-backed tests**. New coverage: strict multer limits + error mapping, per-user quota (count + bytes), write-failure compensation, the admin proof route auth guard over real HTTP (non-admin → 403, anonymous → login redirect), and a real-multipart CSRF composition test (session stub → multer → real `verifyCsrf`; valid token passes, missing/wrong → 403, extra fields → friendly rejection).

Phase 13 — Billing lock for unpaid users + role policy — 2026-09-13

Added

- **Billing lock for unpaid users.** A non-admin account is confined to the **Billing** and **Account** pages until an admin confirms its first payment. New middleware `server/middleware/paymentLock.js` (`requirePayment`) is mounted at `/dashboard` in `server/index.js` immediately after `verifyCsrf` and before every dashboard router. Anonymous requests pass through (the downstream `requireAuth` owns the login redirect); admins always pass. Locked users hitting any other `/dashboard/*` page get an error flash and a `302` to the pending transaction's pay page (`/dashboard/billing/pay/<id>`) or, when there is no pending transaction, to `/dashboard/billing`.
- **New service `server/services/billingAccess.js`.** `getBillingAccess(userId)` resolves `{ unlocked, pendingTransactionId }` with one query (`EXISTS(paid transaction)` + `COALESCE(SUM(credit_ledger.delta), 0)`) and only looks up the most recent `pending` transaction when locked. Pure helpers `isAllowedBillingPath` (Billing/Account prefixes; non-strings rejected) and `billingRedirectTarget` are exported for unit tests.
- **Locked-state UX.** `server/views/partials/dashnav.ejs` renders only the Billing and Account tabs (plus a "Locked — payment required" indicator) when `res.locals.billingLocked` is true; `server/views/dashboard/billing.ejs` shows an info banner explaining that the account unlocks after an admin confirms the first payment.
- **Admin proof visibility.** The Admin overview pending-payments query (`server/routes/admin.js`) now returns `(SELECT COUNT(*) FROM payment_proofs pp WHERE pp.transaction_id = t.id) AS proof_count` and `server/views/admin/index.ejs` renders an "N proof(s)" badge (or "No proof") so admins can spot transactions with uploaded proof before confirming.

Changed

- `server/index.js` mounts `app.use('/dashboard', requirePayment)` immediately after `app.use(verifyCsrf)` and before the first dashboard router. The Phase 12 proof-upload multer mount stays before CSRF, so multipart uploads still parse `_csrf`; the lock allows every `/billing/*` path (including proof upload/view/delete, checkout, cancel and sender-name) plus `/account/*` and `/integration/*` (the integration request form starts a pending payment).
- Registration is unchanged: `POST /register` hardcodes `role = 'user'` and a `role` field in the request body is ignored. A regression test now pins this.

Security

- The lock is enforced **before** every dashboard router, so a locked user cannot reach Overview, WhatsApp, API keys, Messages, Scheduled, Templates, Analytics or Webhooks — even by direct URL. Billing/Account paths always pass through, so there is no redirect loop.
- `isAllowedBillingPath` only ever returns `true` for string paths starting with `/billing` or `/account`; non-string input can only lock down.

- The lock is dashboard-only: `/api/v1/*` is unchanged (API sends already fail with `402 insufficient_credits` at a zero balance) and `/admin/*` is unaffected.
- Existing accounts are promoted to `admin` by the deploy agent; production promotion is a one-time SQL step outside this change set.

Tests

- `npm test` → **97 unit tests** (`test/unit/billingAccess.test.js` added).
- `npm run test:integration` → **110 DB-backed tests** (`test/integration/billingAccess.test.js`, `test/integration/paymentLock.test.js` added; the admin overview proof-count query is covered in `test/integration/paymentProofs.test.js`).
- `paymentLock.test.js` drives the middleware with fake req/res (locked → `302` + flash; Billing/Account → `next()` + `billingLocked`; paid/positive balance/admin/anonymous → `next()`; DB error → `next(err)`) and verifies a client-supplied `role: 'admin'` at registration is ignored.

Phase 12 — Payment proof uploads — 2026-09-13

Added

- **Payment proof uploads.** After choosing an amount/method and confirming a manual payment, the customer now sees a **Payment proof** card on the pay page (`server/views/dashboard/pay.ejs`) with instructions and an upload form (`POST /dashboard/billing/proof/:id`, multipart). Files are stored under `<uploads>/payment-proofs/<transaction_id>/<random16hex>.<ext>`; a row is written to the new `payment_proofs` table (migration `server/migrations/012_payment_proofs.sql`). Up to **5 proofs per transaction**, **5 MB** each, types **JPG/PNG/WebP/PDF**.
- **New service `server/services/proofs.js`:** `sniffMime` (magic bytes), `safeOriginalName` (control chars stripped, path dropped, 255-char cap, `proof` fallback), `makeStoredName` (`<16 hex>.<ext>`), `validateProofBuffer`, `assertProofLimit`, and filesystem helpers (`writeProof`, `proofPath`, `proofExists`, `deleteProofFile`) rooted at `config.paths.uploads` (read at call time). Typed errors (`invalid_type`, `too_large`, `limit_reached`) carry user-safe messages.
- **New middleware `server/middleware/uploadProof.js`:** a `multer@^2` instance with `memoryStorage()` and `limits { fileSize: 5 MB, files: 1, fields: 1, fieldSize: 1024, parts: 3 }` (dependency added: `multer`; the stricter field/part limits were added in the Phase 12/13 security hardening above).
- **Authenticated proof serving.** Owners stream their own proofs at `GET /dashboard/billing/proof/:id/:proofId`; admins stream any proof at `GET /admin/transactions/:id/proof/:proofId`. Both send `Content-Disposition: inline; filename="proof-<id>.<ext>"`, `Cache-Control: private, no-store` and `Content-Security-Policy: default-src 'none'; sandbox`; the MIME type comes from the DB row (byte-sniffed). A missing or foreign proof is always a 404.
- **Owner delete while pending** (`POST /dashboard/billing/proof/:id/:proofId/delete`) removes the row and best-effort unlinks the file. Proofs remain readable by owner + admin after `paid` / `failed` for audit.
- **Admin visibility.** `GET /admin/transactions` now attaches each transaction's proofs (single `WHERE transaction_id IN (...)` query) and the table renders a "Proofs (N)" column linking each proof in a new tab.
- **Config:** `UPLOADS_DIR` env override for `config.paths.uploads`.

Security

- File type is decided by **magic bytes only** — the client MIME/filename is never trusted. JPEG (`FF D8 FF`), PNG (`89 50 4E 47 0D 0A 1A 0A`), WebP (`RIFF....WEBP`) and PDF (`%PDF-`) are accepted; everything else is rejected.
- Stored filenames are generated (`crypto.randomBytes(8)` hex) and validated against a strict pattern before any path join — no user-controlled paths, no traversal. Directories are created `0o750`, files written `0o640`.
- Uploads are `memoryStorage` only: nothing hits disk until CSRF, ownership, status, size and bytes have been validated. The DB row is committed first and the file is written afterwards (outside the row lock, H4); if the write fails the committed row (and any partial file) is best-effort removed.
- **CSRF preserved for multipart:** `server/index.js` mounts the multer upload (and a rate limiter) for `/dashboard/billing/proof` **immediately before** `app.use(verifyCsrf)`, so the hidden `_csrf` field is parsed and the global CSRF check still applies. Non-multipart requests pass straight through.
- Upload POSTs are rate-limited (~30 per 15 minutes, keyed by `req.user.id` else IP). Size (5 MB), per-transaction count (5) and per-user storage (25 proofs / 50 MB) caps are enforced in the service inside a transaction that locks the transaction row (`FOR UPDATE`).
- Proofs are served only through authenticated Node routes (`uploads/` is outside `public/`); foreign/missing proofs 404 without leaking existence.

Tests

- `npm test` → **91 unit tests** (`test/unit/proofs.test.js` added).
- `npm run test:integration` → **89 DB-backed tests** (`test/integration/paymentProofs.test.js` added; `test/integration/helpers/db.js` now points `UPLOADS_DIR` at a wiped temp dir and truncates `payment_proofs`).

Phase 11 — Admin sender-name override & message templates — 2026-09-12

Added

- **Admin sender-name override.** Platform admins can view and set/override a tenant's WhatsApp sender (profile) name from **Admin → Sessions** — a new "Sender name" column shows `wa_sessions.profile_name` plus an inline form. `POST /admin/sessions/:id/sender-name` (`server/routes/admin.js`, `server/views/admin/sessions.ejs`) is admin-only, CSRF-protected, validates with the shared `validateSenderName` (1–25 chars, control chars/line breaks rejected), requires the session to be `connected`, calls the engine `setProfileName`, and persists `wa_sessions.profile_name`. Raw engine error detail is logged server-side only and never surfaced to the admin UI.
- **Deliberate entitlement bypass (support tooling).** The customer route (`server/routes/whatsapp.js`) requires both a connected session **and** the paid add-on (`users.sender_name_until`); the admin route intentionally does **not** require the add-on. This is the **only** bypass.
- **Message templates.** New table `message_templates` (migration `server/migrations/011_message_templates.sql`): `id`, `user_id` FK `ON DELETE CASCADE`, `name` VARCHAR(80), `body` TEXT, `media_url` VARCHAR(500), timestamps, `UNIQUE(user_id, name)`.
- New service `server/services/templates.js`: `validateTemplate` (name ≤80, body ≤4096, `http(s)` media URL ≤500, control characters rejected), user-scoped CRUD, strict `parseTemplateId`, `resolvePrefill` (ownership-checked; returns `null` for foreign/invalid/missing ids), `listForUser`

(`LIMIT 500`), and an **atomic per-user cap** (`templates_max_per_user`, default 200) using a user-row lock; typed `duplicate_name` / `limit_reached` errors.
- New dashboard page `/dashboard/templates` (`server/routes/templates.js`, `server/views/dashboard/templates.ejs`): create, inline `<details>` edit (no JS) and delete; a "Templates" tab was added to the dashboard nav (`server/views/partials/dashnav.ejs`).
- Server-side prefill (no JS): `GET /dashboard/whatsapp?template=<id>` and `GET /dashboard/scheduled?template=<id>` pre-fill the message body + media URL from an owned template (`server/routes/dashboard.js`, `server/routes/scheduled.js`).
- Admin setting `templates_max_per_user` (int 1–10000, default 200) in **Admin → Settings** (`server/routes/admin.js`).

Changed

- `validateSenderName` moved to `server/services/senderName.js` (alongside the add-on entitlement helper) and re-exported from `server/routes/whatsapp.js`; the existing unit test is unchanged.
- Templates are dashboard-only — **no API v1 changes**. `{{name}}` / `{{phone}}` placeholders in a prefilled message are still resolved at send time by the existing scheduler/bulk pipeline (unchanged).

Fixed

- Security-review findings fixed before deploy: **P11-01** unbounded template creation → atomic per-user cap (Medium); **P11-02** unbounded template list → `LIMIT 500` (Low); **P11-03** strict route id parsing; **P11-04** admin route safe-integer id; **P11-05** `mediaUrl` control-character rejection.

Tests

- `npm test` → **76 unit tests** (`test/unit/templates.test.js` added).
- `npm run test:integration` → **75 DB-backed tests** (`test/integration/templates.test.js`, `test/integration/adminSenderName.test.js` added).

Phase 10 — Dedicated sender name add-on ($10 / 30 days) — 2026-09-11

Added

- Paid add-on gating the Phase 9 sender-name feature: **$10.00 / 30 days** by default, bought through the existing **manual payment + admin approval** flow. Entitlement is stored on `users.sender_name_until` (UTC) and resolved by `server/services/senderName.js` (`isSenderNameActive`).
- Dashboard Billing card and `POST /dashboard/billing/sender-name` (`server/routes/dashboard.js`, `server/views/dashboard/billing.ejs`): creates a `pending` transaction with `credits = 0` and `meta = { type: 'sender_name', days }`, then redirects to the standard payment page. `server/views/dashboard/pay.ejs` labels the transaction and period.
- Settlement extension (`server/services/payments/index.js`): approving a `meta.type='sender_name'` transaction sets `sender_name_until = DATE_ADD(GREATEST(COALESCE(sender_name_until, NOW()), NOW()), INTERVAL ? DAY)`, so a **renewal extends from the current expiry** and paid time is never lost. `meta.days` is clamped to **1..365**; an unset/unparseable value falls back to the configured `sender_name_days`.
- Migration `server/migrations/010_sender_name_addon.sql`: `users.sender_name_until DATETIME NULL` plus settings seeds `sender_name_price_cents = 1000` and `sender_name_days = 30`.
- Admin settings (`server/routes/admin.js`): `sender_name_price_cents` (int, 1–100000000) and `sender_name_days` (int, 1–365) are editable.

- Tests: `test/integration/senderNameAddon.test.js` — settlement/renewal, `days` clamping, entitlement-update rollback + retry, the connected/entitled gate, and concurrent checkout (exactly one pending transaction).

Changed

- `payments.manual.createCheckout` accepts an optional `conn` so the dashboard sender-name route can insert the pending transaction inside its own transaction. Existing callers are unchanged (defaults to the shared pool).
- The WhatsApp sender-name card and the Billing card show the add-on state and a Billing CTA when inactive; the WhatsApp form is disabled until the add-on is active and the session is `connected`.
- `/pricing` advertises the add-on next to message credits (published default list price **$10.00 / 30 days**).
- **Message pricing is unchanged: 10 = 1,000*credits*(0.01/message).** Attachments/media remain available to all users and are **not** gated by the add-on.

Fixed

- **Settlement rollback safety:** if the entitlement `UPDATE` throws after the `pending → paid` transition, the transaction is reverted to `pending` (`paid_at = NULL`) and the error is rethrown, so an admin can retry and no payment is lost (mirrors the existing credit-add compensation).
- **Atomic duplicate-checkout guard:** `POST /dashboard/billing/sender-name` now locks the user row (`SELECT ... FOR UPDATE`), re-checks for an existing pending `sender_name` transaction, and inserts in the **same** transaction. The previous non-atomic SELECT-then-INSERT allowed N concurrent requests to each create a pending transaction (TOCTOU).
- **`days` clamp correctness:** the entitlement extension previously used `parseInt(meta.days) || fallback`, which treated a legitimate `days: 0` as unset. The code now uses an explicit finite check and clamps to `1..365`.
- Concurrent admin approvals cannot double-extend: the atomic `pending → paid` gate means exactly one winner extends the entitlement.

Phase 9 — WhatsApp sender name — 2026-09-11

Added

- Dashboard sender-name card on `/dashboard/whatsapp` and `POST /dashboard/whatsapp/sender-name` (`server/routes/whatsapp.js`, `server/views/dashboard/whatsapp.ejs`): customers can set the WhatsApp profile/push name recipients see. The form is enabled only while the session is `connected` (and, since Phase 10, while the add-on is active).
- Engine call `wpp.setProfileName(sessionName, name)` (`server/services/wppconnect.js`) → `POST /api/{session}/change-username` with `{ name }`, which the engine maps to `client.setProfileName(name)`.
- Pure `validateSenderName()` helper (exported for tests): control characters and line breaks rejected; whitespace runs collapsed; leading/trailing whitespace trimmed; result must be **1–25 characters**.
- Migration `server/migrations/009_sender_name.sql`: `wa_sessions.profile_name VARCHAR(60) NULL` — the last name pushed through ZNotice (`NULL` = never set here). `waSessions.update()` persists `profileName`.
- Tests: `test/unit/senderName.test.js` (validation) and `test/integration/senderName.test.js` (engine call + persistence and every failure path leaving `profile_name` untouched).

Changed

- The sender-name card states the WhatsApp profile-name caveat: changing the name **also changes it on the customer's phone**, and WhatsApp limits how often a profile name can be changed.

Notes

- Dashboard-only; **no API v1 changes**.
- The engine is called only after the session-connected check (and the Phase 10 entitlement gate), and engine error detail is logged server-side, never shown to the customer.

Phase 8 — Account recovery, hardening & test suite — 2026-09-11

Added

- Self-service password reset (web UI, dashboard only — **no API changes**): `GET/POST /forgot-password` and `GET/POST /reset-password` (`server/routes/auth.js`, `server/views/auth/forgot-password.ejs`, `server/views/auth/reset-password.ejs`). The forgot response is always the same (`302 /forgot-password?sent=1`) whether or not the account exists, so it cannot be used to enumerate users. Both endpoints are CSRF-protected and rate-limited per IP: **5 / 15 min** (forgot) and **10 / 15 min** (reset).
- Reset-token lifecycle: 32 random bytes (`crypto.randomBytes(32).toString('hex')`, 64 hex chars) with **only `sha256(token)` stored**; a 30-minute expiry; and single-use enforced by an **atomic conditional claim** inside the password-change transaction (`UPDATE password_resets SET used_at=NOW() WHERE id=? AND used_at IS NULL AND expires_at>NOW()`; `affectedRows!==1` → invalid link). Requesting a new link deletes any prior unused token. A successful reset changes the password, deletes every session for the user, and sends a password-changed notification (fire-and-forget).
- Outbound email service (`server/services/email.js`, nodemailer **10.0.8**): env-driven SMTP config (`SMTP_HOST`, `SMTP_PORT` default `587`, `SMTP_SECURE`, `SMTP_USER`, `SMTP_PASS`, `SMTP_FROM`), reused transport, `passwordResetEmail()` / `passwordChangedEmail()` templates, and a `send()` that **never throws** and never logs credentials, tokens or message bodies. With no `SMTP_HOST` it returns `{ok:false, error:'email_not_configured'}` and the UI still shows the generic "check your email" message.
- Migration `server/migrations/008_password_resets.sql`: `password_resets` (`token_hash CHAR(64)` unique, `expires_at`, `used_at`, FK `user_id` → `users` `ON DELETE CASCADE`). Defensive `CREATE TABLE IF NOT EXISTS`.
- Automated test suite (Node's built-in `node:test`, **no new dependencies**):
  - `npm test` → **57 unit tests** (`test/unit/`: `send`, `schedule`, `webhook`, `netguard`, `analytics`, `safeNext`), no database required.
  - `npm run test:integration` → **36 DB-backed tests** (`test/integration/`: `credits`, `schedule`, `scheduler`, `passwordReset`, `reaper`).
  - `npm run test:all` runs both.
  - `test/integration/helpers/db.js` refuses to run against production (`NODE_ENV!==production`, `TEST_DB_NAME` must end in `_test` and differ from the `.env` `DB_NAME`), forces the isolated test DB before the app loads, and serializes integration files with a MariaDB advisory lock (`GET_LOCK`).

Changed

- **Lazy CSRF token** (`server/middleware/csrf.js`): `attachCsrf` now materializes `res.locals.csrfToken` only when a view actually reads it, and `needsCsrf(req)` mints a token only

for authenticated requests or the auth form pages (`/login`, `/register`, `/forgot-password`, `/reset-password`). Public pages no longer create `sessions` rows (previously an enumerable `res.locals` getter forced `express-mysql-session` to persist one on every render).

- **Session reaper pruning** (`server/services/reaper.js`): a new pass deletes expired `sessions` rows and anonymous sessions older than **7 days** (`sessionsDeleted` in the summary). `sessions.expires` is stored by `express-mysql-session` in UNIX **seconds**, not milliseconds.
- **No-store on private responses**: `Cache-Control: no-store` is set on authenticated HTML responses (`server/index.js`) and on **all** `/api/v1` responses including 401/403 (`server/routes/api/v1.js`).
- **Log redaction** (`server/index.js`): the global error log now redacts the `token`, `secret`, `key` and `password` query parameters (case-insensitive), so a reset token or secret can never reach the journal.
- **SMTP transport rotation** (`server/services/email.js`): the transport cache key includes a short non-reversible password fingerprint, so a rotated `SMTP_PASS` rebuilds the transport instead of silently reusing old credentials.
- **Host hardening (item E)** applied by `@devops`; full record and rollback in `docs/OPERATIONS.md` §14 and `/root/backups/host-hardening-2026-09-11/CHANGES-AND-ROLLBACK.md`:
  - UFW stale allows for **3001/tcp, 3005/tcp and 4000/tcp** (v4 + v6) removed; the GameBet `next-server` on 3001 stays reachable through NGINX on loopback, direct external access is denied. Rollback: `ufw allow <port>/tcp`.
  - Sudoers hygiene: duplicate `kali ALL=(ALL) NOPASSWD:ALL` and the empty-group `kali-grant-root` file removed; `kali` passwordless sudo retained; `visudo -c` clean.
  - NGINX `access_log off` for the exact `location = /reset-password` so reset tokens in the query string never reach `access.log`.

Fixed

- **F1 — reset-token race (single-use):** the token is claimed with a conditional `UPDATE` *inside* the password-change transaction, so two concurrent submissions of the same token yield exactly one winner and one `400` invalid-link response. Regression test: `test/integration/passwordReset.test.js` ("two parallel consumptions of one token yield exactly one winner").
- **F2 — `safeNext` open redirect:** the post-login/registration redirect target now blocks protocol-relative (`//host`), dot-segment (`/..//host`, `/%2e%2e//host`), raw/encoded backslash (`\`, `%5c`) and encoded-separator (`%2f` / `%5c`, including multi-encoded) variants, and only ever returns a same-origin path/search/hash. `safeNext` is exported for unit tests (`test/unit/safeNext.test.js`).
- **F3 — register 500 on long email:** an email longer than `VARCHAR(190)` is now rejected with a clean `400` (was a `500` "Data too long").
- **F5 — error-log redaction** (above).
- **F6 — session pruning** (above).
- **F7 — SMTP transport rebuild on password rotation** (above).
- **F8 — maximum password length:** passwords over **72 bytes** (the bcrypt truncation point) are rejected on registration, reset and account password change, including multi-byte input.
- One-time cleanup removed ~1,300 anonymous `sessions` rows created by the pre-Phase-8 CSRF behaviour.

Open (documented, not fixed)

- `/register` still returns `409` for an existing email — a pre-existing user enumeration vector (see `docs/SECURITY.md` §5.13).
- `POST /forgot-password` response time differs by ~15 ms for existing vs non-existing accounts; status and body are identical (see §5.14).

Post-review configuration (2026-09-11): production SMTP is configured using the existing Hostinger mailbox (`info@consultbofa.com`, `smtp.hostinger.com:465`, implicit TLS) and verified with a live test send; see `docs/OPERATIONS.md` §12 and `docs/SECURITY.md` §5.15.

Phase 7 — Usage analytics — 2026-09-11

Added

- Read-only analytics service (`server/services/analytics.js`): `customerDaily`, `customerSummary`, `adminDaily`, `adminSummary` and `topUsers`. Pure `SELECT` aggregations over `messages`, `credit_ledger`, `transactions` and `users`; no writes, no cache, no background worker.
- Customer analytics page `/dashboard/analytics` (`server/routes/analytics.js`, `server/views/dashboard/analytics.ejs`, fixed 30-day window): summary cards (messages today / last 7d / last 30d, 30-day success rate, 30-day credits spent), messages-per-day and credits-spent-per-day charts, a sent/delivered/read delivery funnel, and the top 5 failure reasons (last 30d, blank errors normalized to `unknown`), with a friendly empty state. New **Analytics** tab in the dashboard nav after Overview.
- Admin analytics page `/admin/analytics` (admin only; `server/routes/admin.js`, `server/views/admin/analytics.ejs`, fixed 30-day window): summary cards (revenue from paid transactions, credits sold, messages, active users, credits outstanding, pending payments), revenue-per-day and messages-per-day charts, and the top 10 users by message volume (name / email / messages / credits spent / last activity). New **Analytics** tab in the admin nav after Overview.
- Customer API `GET /api/v1/usage?days=N` (`server/routes/api/v1.js`): API-key auth; `days` default 30, clamped 1–90. Returns `{ok, days, data[], totals}` with exactly `days` zero-filled UTC day points ending today and a `success_rate` `0`–`1` ratio.
- Reusable server-rendered inline SVG bar chart (`server/views/partials/svg-bars.ejs`): single or grouped bars, accessible (`role="img"`, `<title>` tooltips), no chart library, presentation attributes only — CSP-safe (no inline scripts or style attributes).
- Migration `server/migrations/007_analytics_indexes.sql`: index-only (`messages(created_at)`, `transactions(status, paid_at)`), idempotent and `information_schema`-guarded so it is safe to re-run.
- Docs: analytics pages, the `/api/v1/usage` endpoint, metric semantics, the new indexes and a runbook section added to `docs/API.md`, `docs/ARCHITECTURE.md`, `docs/OPERATIONS.md`, `docs/SECURITY.md` and the customer `/docs` page.

Changed

- Dashboard and admin navigation each add an **Analytics** tab immediately after Overview.
- `server/index.js` mounts the customer analytics router at `/dashboard/analytics` (before the `/dashboard` catch-all).

Fixed

- **Queued-credit consistency:** every `credits_spent` aggregate sums `credits_charged` only for completed statuses (`sent` / `delivered` / `read` / `failed`). `queued` rows are excluded from customer counts and from all credit sums, so analytics never report credits that were not actually charged.
- **Defense-in-depth input clamp:** `days` is clamped to `1..90` both in the HTTP routes and inside the service (`clampDays`), and `topUsers.limit` is clamped to `1..50` in the service, so a direct service call can never produce an `INTERVAL NaN DAY` SQL error or allocate an unbounded day-key array.

- Security review outcome: **GO**. One low-severity follow-up recorded as **L2** — authenticated HTML/API responses lack `Cache-Control: no-store` (browser-local cache/bfcache only); see `docs/SECURITY.md` §5.12.

Phase 6 — Scheduled & bulk sending — 2026-09-11

Added

- Scheduled send: `POST /api/v1/send` accepts an optional `scheduled_at` (ISO-8601 with `Z`/offset, or naive `YYYY-MM-DDTHH:MM` / `YYYY-MM-DD HH:MM` interpreted as UTC). Omitted/blank keeps the immediate path; a valid future value enqueues and returns `202 {ok, scheduled_id, scheduled_at, credits_balance}`. Window: now+60s … now+`schedule_max_days_ahead` (default 90) days. Enqueue prechecks: active account (`403 account_suspended`), connected WhatsApp session (`409 whatsapp_not_connected`), balance ≥ item count (`402 insufficient_credits`) and the per-user pending cap (`429 queue_full`).
- Bulk send: `POST /api/v1/bulk` (`{message, media_url?, scheduled_at?, label?, recipients:[{to,name?}]}`), 1–`bulk_max_recipients` (default 1000) recipients, normalized + deduped (response `duplicates_removed`); invalid entries reject the request with `400 invalid_recipients` (up to 10 `{line,value}`). `{{name}}` / `{{phone}}` are resolved per recipient at enqueue; unknown tokens are left untouched. Returns `202 {ok, batch_id, total, duplicates_removed, scheduled_at, credits_balance}`.
- Queue management API: `GET /scheduled` (default pending+sending, whitelisted `?status=`, `?limit=` 1–200), `GET /scheduled/:id`, `DELETE /scheduled/:id` (pending only; `409 not_pending`), `GET /batches` (`?limit=` 1–100), `GET /batches/:id` (batch + items, `?limit=` 1–500) and `DELETE /batches/:id` (cancels remaining pending items; `409 not_cancellable` when terminal).
- Scheduler worker (`server/services/scheduler.js`): every 30s plus a kick on enqueue. Claims due `pending → sending` atomically (at-most-once), enforces a per-user rate limit shared with `send_rate_per_min`, and retries only retryable failures (409 not connected, engine unreachable, engine HTTP 5xx) with `attempts++` and +1 min/+5 min backoff, terminal after 3 attempts. Credits are charged per message at actual send time and auto-refunded on engine failure; there is no reservation at enqueue.
- Scheduled/bulk service (`server/services/schedule.js`): `parseScheduledAt`, `resolveTemplate`, `parseRecipientLines`, `normalizeRecipients`, create/list/ cancel helpers, and atomic pending-cap enforcement via a user-row lock.
- Batch completion webhook: `batch.completed` is emitted **once**, only by the scheduler, when a batch finishes with zero cancellations. Fully-cancelled batches become `cancelled` silently; partially-cancelled batches become `completed` silently.
- Dashboard scheduled queue (`server/routes/scheduled.js`, `server/views/dashboard/scheduled.ejs`, `server/views/dashboard/scheduled-batch.ejs`): create form (message, media URL, datetime-local UTC, recipients textarea `phone` or `phone,name`), open queue (pending + sending), batch list, paginated batch detail (100/page), and cancel buttons (pending items only). CSRF-protected; no inline JS.
- Migrations `server/migrations/005_scheduled_bulk.sql` (`send_batches`, `scheduled_messages`, Phase 6 settings seeds) and `server/migrations/006_scheduler_claim.sql` (adds the `sending` claim state).
- Admin settings: `bulk_max_recipients`, `schedule_max_days_ahead` and `scheduled_max_pending` are editable in the settings editor.

- Docs: scheduled/bulk endpoints, the scheduler/queue flow, runbook entries and security controls added to `docs/API.md`, `docs/ARCHITECTURE.md`, `docs/OPERATIONS.md`, `docs/SECURITY.md` and the customer `/docs` page.

Changed

- `server/services/reaper.js` now resets scheduler claims stuck in `sending` for >10 min to `failed` / `interrupted_by_restart` (bumping the batch `failed_count` and reconciling silently), and prunes terminal `scheduled_messages` and `send_batches` older than 90 days. The summary includes `sendingInterrupted`.
- `server/services/send.js` validates `media_url` at a maximum of 500 characters (shared by immediate, scheduled and bulk sends) and exports `MAX_MEDIA_URL_CHARS`.
- `server/index.js` starts the scheduler alongside the reaper and webhook worker; `server/routes/api/v1.js` routes a non-empty `scheduled_at` through the queue service.

Fixed

- Concurrency hardening (security-review follow-ups):
  - **Atomic batch completion:** `maybeCompleteBatch()` is a single conditional UPDATE guarded by `status IN ('scheduled','processing')` and `NOT EXISTS(open item)`, so concurrent callers race on the row and exactly one wins; `batch.completed` cannot be emitted twice.
  - **Atomic pending cap:** enqueue serializes the cap check + insert on the user row (`SELECT ... FOR UPDATE`), closing a TOCTOU that could overrun `scheduled_max_pending`.
  - **Claim pattern:** `pending → sending` before the engine call, so a cancel cannot race an in-flight send and a crash cannot duplicate a message.
  - **`media_url` length validation** prevents an over-long URL from surfacing as an unhandled "Data too long for column" 500.
  - **Scheduler fairness:** `ROW_NUMBER() ... PARTITION BY user_id` caps each user at 25 items per tick (global LIMIT 50), so one tenant can no longer consume a whole tick.
  - **Suspended-account 500:** account status is checked before the WhatsApp session lookup in the send/enqueue paths, so a suspended user receives a clean `403 account_suspended` instead of a 500.

Phase 5 — Account settings, session reaper & outbound webhooks — 2026-09-11

Added

- Account settings page (`server/routes/account.js`, `server/views/dashboard/account.ejs`): editable display name, read-only email, and a password change flow (bcrypt cost 12) that invalidates all other sessions while keeping the current one signed in. Password changes are rate-limited (5 / 15 min / IP+user) and logged without any password values.
- Stale-session TTL reaper (`server/services/reaper.js`, started from `server/index.js`): frees abandoned `pending` sessions (>10 min) and closed `connected` sessions, and prunes `webhook_deliveries` older than 30 days. Exposes `runOnce()` for scripts; no HTTP trigger.
- Outbound customer webhooks:
  - `server/services/netGuard.js`: SSRF egress guard extracted from `mediaFetch.js` (URL/IP validation + DNS pinning), reused by webhooks.

    - `server/services/webhookOut.js`: durable delivery queue with HMAC-SHA256 signatures (`X-ZNotice-Signature`), 10s timeout, +1 min/+5 min backoff, and terminal failure after 3 attempts.
    - `server/routes/webhookConfig.js` + `server/views/dashboard/webhooks.ejs`: configure/validate the endpoint, enable/disable, rotate the secret, send a test event, and view the last 25 deliveries.
    - Events emitted: `message.sent`, `message.failed` (send pipeline), `message.delivered`, `message.read` (engine ack transitions, no duplicates), and `test.ping`.
- Migration `server/migrations/004_outbound_webhooks.sql`: `user_webhooks` (one endpoint per user) and `webhook_deliveries` (queue + log).
- Docs: outbound webhook contract + signature verification in `docs/API.md` and `/docs`; reaper/webhook troubleshooting in `docs/OPERATIONS.md`; SSRF guard note in `docs/SECURITY.md`.

Changed

- `server/services/mediaFetch.js` now delegates URL/IP validation and DNS pinning to `netGuard.js` (behaviour and error messages unchanged).

Phase 4 — Operations hardening & documentation — 2026-09-10

Backups, health monitoring, log rotation, and this documentation set.

Added

- `znotice-backup.sh` (`/usr/local/bin/znotice-backup.sh`): daily MariaDB dump (`--single-transaction --routines --triggers`, gzip-verified) plus an app + secrets archive. Retains 14 DB dumps / 7 app archives; credentials never printed; backups stored `0640 root:www-data`.
- `znotice-health.sh` (`/usr/local/bin/znotice-health.sh`): 5-minute probe of systemd state, public HTTPS (pinned to loopback), engine reachability, and available RAM. Auto-restarts the app only; never the engine.
- Cron schedules: `/etc/cron.d/znotice-backup` (02:30 daily), `/etc/cron.d/znotice-health` (every 5 min).
- `/etc/logrotate.d/znotice`: weekly rotation, 8 copies, compressed.
- Documentation set in `docs/`: `README.md`, `ARCHITECTURE.md`, `OPERATIONS.md`, `API.md`, `BUSINESS.md`, `SECURITY.md`, `CHANGELOG.md`.

Infrastructure

- NGINX vhost with Let's Encrypt TLS, ACME webroot challenge, security headers, and loopback proxy to `127.0.0.1:5071`.
- `znotice.service` systemd unit (user `www-data`, `Restart=always`, `EnvironmentFile=.env`).
- WPPConnect engine container `wppconnect` (v2.10.10) bound to `127.0.0.1:21465`.

Phase 3 — Manual payments, integration orders, admin panel, capacity guard — 2026-09-10

Added

- Payment adapter registry: `server/services/payments/index.js` with the `manual` provider, `ZN-XXXXXXXX` reference generation, idempotent `settlePaid()` / `settleFailed()`, and instruction templating.

- Manual credit top-up: `POST /dashboard/billing/checkout`, `GET /dashboard/billing/pay/:id`, `POST /dashboard/billing/cancel/:id`.
- Integration service orders ($20): `POST /dashboard/integration/request`, order lifecycle `requested → paid → in_progress → delivered` (+ `cancelled`).
- Full admin panel (`server/routes/admin.js`): overview stats, user suspend/activate, credit adjustments, payment approval queue, integration order management, session management with force-disconnect, and editable settings.
- Capacity guard in `server/routes/whatsapp.js` using `max_concurrent_sessions` (default 15), counting `pending` + `connected` sessions while allowing owners mid-pairing.
- Migration `server/migrations/003_phase3.sql`: unique `transactions.provider_ref`, `integration_orders.status` gains `paid`, `integration_orders.transaction_id` FK, and new settings seeds.
- Admin views: `server/views/admin/*.ejs`.

Phase 2 — QR connect, send pipeline, webhooks, terms & admin basics — 2026-09-09

Added

- WhatsApp connection flow (`server/routes/whatsapp.js`): `POST /connect`, `GET /qr`, `POST /disconnect`; WPPConnect client wrapper (`server/services/wppconnect.js`) with per-session bcrypt bearer tokens.
- Outbound send pipeline (`server/services/send.js`): validation, phone normalization, per-key rate limiting, atomic credit charge, engine send, failure persistence + automatic refund.
- Credit accounting (`server/services/credits.js`): ledger-based balance with row-locked atomic `chargeCredits` / `addCredits`.
- Inbound delivery webhooks (`server/routes/webhooks.js`): `onack` status progression (monotonic), `qrcode`, `status-find`, `closesession`, `logoutsession`; secret path token / header auth.
- Dashboard pages: overview, WhatsApp, API keys (create/revoke), messages (paginated), send test, billing.
- API-key management (`server/routes/apiKeys.js`) and API v1 (`server/routes/api/v1.js`): `GET /status`, `GET /messages`, `POST /send`.
- Terms & Acceptable Use page (`server/views/terms.ejs`) with the WhatsApp ban-risk disclosure; public pricing and docs pages.
- Admin basics and `settings` service with 60s cache.
- Migration `server/migrations/002_phase2.sql`: `users.accepted_terms_at`, `messages(user_id,status)` index, `default_country_code` and `send_rate_per_min` settings.

Phase 1 — Scaffold & infrastructure — 2026-09-08

Added

- Node.js/Express 4 application skeleton: `server/index.js`, `server/config.js`, `server/db.js` (connection pool + idempotent migration runner), `server/migrate.js`.
- MariaDB-backed sessions, helmet CSP, rate limiting, CSRF middleware, authentication middleware.
- Web auth (`server/routes/auth.js`): register, login, logout with bcrypt and session regeneration.

- Public pages (`server/routes/pages.js`): landing, pricing, docs, terms.
- EJS layout and views (`server/views/**`).
- Base schema `server/migrations/001_init.sql`: `users`, `api_keys`, `wa_sessions`, `messages`, `credit_ledger`, `transactions`, `integration_orders`, `settings`.
- `server/create-admin.js` CLI to create/reset an admin and print a generated password.
- systemd unit, NGINX vhost and TLS provisioning for `znotice.consultbofa.com`.