UCRM — Universal iGaming CRM

Complete User & Operator Documentation

Version: 2026-08-13 · **Status:** Live production (Tenant TIKETABETcom)

Table of Contents

---

1. Executive Summary

UCRM is a **multi-tenant Customer Relationship Management platform purpose-built for iGaming operators** — casinos, sportsbooks, and hybrid platforms. It sits alongside (never inside) the gaming platform of record, ingests player events through a documented adapter contract, and drives outbound player communication across email, SMS, WhatsApp, Telegram, push, and generic webhooks.

What UCRM does for you:

- Keeps a single, deduplicated view of every player across whatever number of platform accounts they hold, without ever mutating the platform's own identity records.
- Runs deterministic customer journeys (welcome, first-deposit, reactivation, sunset) with multi-channel touchpoints, per-language content, and mid-journey deposit/exclusion guards.
- Reports revenue, cost, and player behaviour in the operator's chosen display currency while storing money natively in each transaction's currency of record.
- Enforces regulator- and compliance-friendly guardrails: audit log for every admin mutation, GDPR export/delete, self-exclusion honour list, list-quality suppression, per-player exclusion from statistics.
- Ships with a marketing-ready admin console (Next.js) and a documented HTTP API for headless integrations.

Who uses UCRM:

- **CRM operators** — daily driver: segment lists, launch campaigns, review journey health, respond to list-quality issues.
- **Retention / VIP managers** — VIP tier config, per-player intervention, bonus tuning.
- **Compliance officers** — audit log, GDPR requests, self-exclusion enforcement.
- **Finance** — cost tracking, revenue reports, bonus economics.
- **Engineering** — adapter integration, provider wiring, journey graph edits.

What makes UCRM different:

UCRM was designed on a strict **"platform is the source of truth"** doctrine. It never mutates player wallets, never authorises bets, never issues bonuses on its own. Every write that could affect money flows back into the platform via the adapter contract. This makes UCRM safe to deploy alongside licensed platforms without disturbing regulated ledger boundaries.

The **two-layer identity model** (Person layer + PlatformAccount sub-table) makes multi-account fraud detection a first-class citizen rather than a data cleaning chore: every merged Person keeps the full list of external platform accounts visible, so operators can see "this one deposit came from three accounts sharing the same phone" without having to run offline joins.

The **universal i18n resolver** picks a body language per message per player based on explicit player language, then tenant currency → language map, then tenant default, then English. This lets a single template serve any tenant configuration without operator branching.

---

2. System Architecture

2.1 High-level services

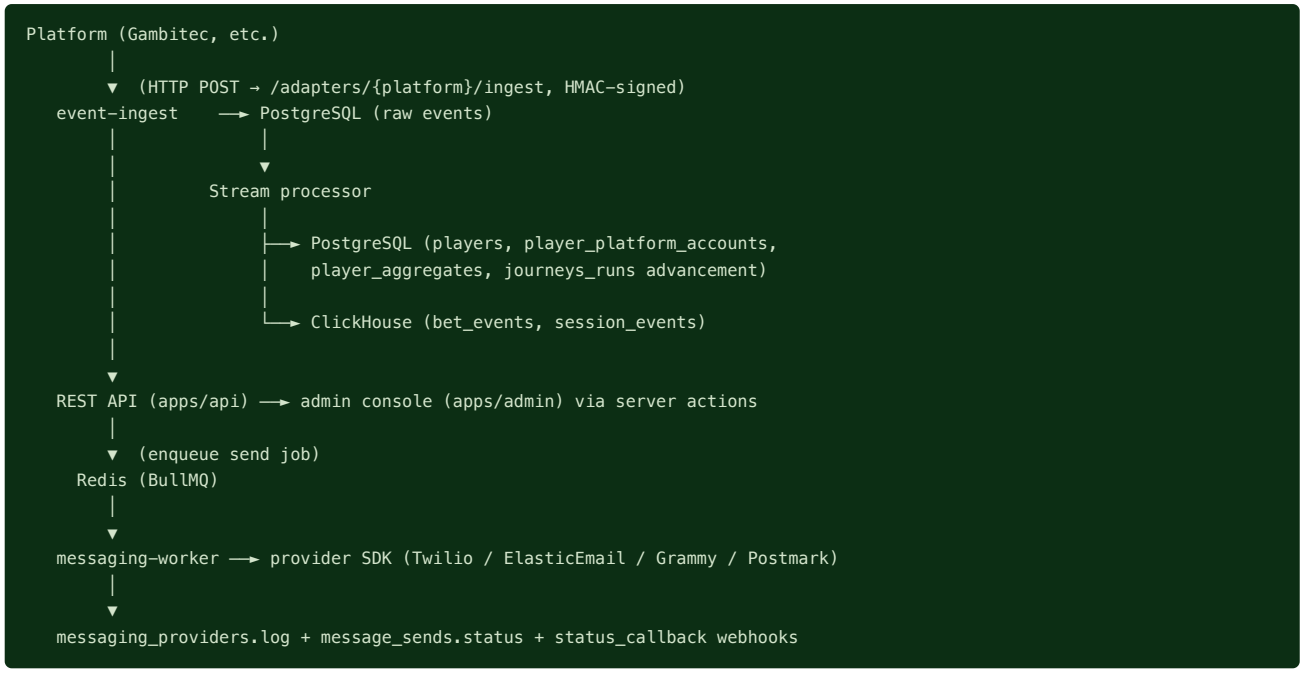
UCRM is a **monorepo** organised as:

```
apps/
  admin/            Next.js 15 admin console (server actions + RSC)
  api/              Fastify REST API + Bull queue producer
services/
  messaging-worker/ BullMQ consumer: renders templates, calls provider SDKs
  journey-worker/   Journey scheduler: advances runs through the graph
  event-ingest/     Adapter-agnostic ingest surface for platform events
packages/
  ui/               shared design system (Cards, Tables, forms)
  ui-client/        client-only widgets (rich text editor, charts)
  sdk/              typed TS client for consumers of the REST API
infra/
  scripts/          staging deploy, backup, provisioning
  compose/          docker-compose files per env
```

2.2 Runtime dependencies

Component	Purpose	Notes
PostgreSQL 16	Primary store	Row-Level-Security per tenant, `app.tenant_id` GUC
Redis 7	BullMQ backing store	Delayed jobs, retries, dead letters
ClickHouse	Analytics events store	Bet events, session events, aggregates for dashboards
Docker Compose	Orchestration	Single-box staging + prod, no Kubernetes required
Nginx	TLS termination & routing	Behind Cloudflare in production

2.3 Data flow



2.4 Tech stack

- **Languages:** TypeScript everywhere (Node 20+), a handful of SQL migrations.
- **HTTP:** Fastify 5 for the API, Next.js 15 (App Router, React Server Components) for admin.
- **DB toolchain:** Drizzle ORM for schema + queries, plain SQL for migrations.
- **Queue:** BullMQ (Redis-backed), one queue per channel (email/sms/whatsapp/telegram/push/webhook).
- **Auth:** Clerk for admin login + organisation membership. API keys (bearer) for programmatic access.
- **Deploy:** Docker images built in CI, pushed to a private registry, pulled on the staging/prod host and started by `infra/scripts/staging-deploy.sh` (which also handles Postgres migrations and `docker-compose up -d --force-recreate` for the changed services).

---

3. Multi-Tenancy Model

3.1 Tenants and projects

Every piece of user-visible data lives inside a **tenant**. Tenants map one-to-one to Clerk organisations. When an operator switches organisation in the top bar, the whole admin console reloads scoped to that tenant.

Each tenant contains one or more **projects**. A project is a scoping boundary for resources that would otherwise collide (API keys, segments, templates, campaigns, journeys, players). Typical setup:

- `TenantX-dev` — sandbox project used by engineers testing changes
- `TenantX-prod` — live project receiving real platform events

You are always working "as of" one active project; the top bar has a project switcher. If you have only one project it is picked automatically; the switcher hides.

3.2 Isolation

All tenant-scoped tables use **Postgres Row-Level-Security** with the pattern:

```
CREATE POLICY tenant_isolation ON some_table
  USING (tenant_id::text = current_setting('app.tenant_id', true))
  WITH CHECK (tenant_id::text = current_setting('app.tenant_id', true));
ALTER TABLE some_table FORCE ROW LEVEL SECURITY;
```

The API enforces `SET LOCAL app.tenant_id = ...` on every request after Clerk auth. Query the DB from the wrong tenant and you see zero rows, not an error — RLS is silent by design.

3.3 What is per-tenant vs shared

Per-tenant: players, campaigns, journeys, templates, providers, API keys, segments, bonus codes, reports, VIP tiers, players/exclusions, i18n config, everything else you configure in admin.

Shared (single-instance): the currency table (ISO 4217 code + scale), the FX rates table (though tenants can add their own overriding rate row), the Clerk user + organisation table.

3.4 Tenant configuration

Tenant-wide settings live in `tenants.config` (JSONB) with a versioned inner schema:

```
{
  "license": "curacao|malta|isle-of-man|other",
  "verticals": ["casino", "sportsbook", "poker"],
  "kyc": { "provider": "sumsub|inhouse", "required_before_withdrawal": true },
  "responsible_gaming": {
    "self_exclusion_min_days": 180,
    "deposit_limit_default_currency": "EUR"
  },
  "i18n": {
    "default_language": "en",
    "default_language_by_currency": { "ARS": "es", "BRL": "pt" }
  }
}
```

Edited through the admin page `/dashboard/settings/tenant` (license, KYC, verticals, RG) and `/dashboard/settings/i18n` (language & currency mapping).

---

4. Authentication & RBAC

4.1 Admin authentication

Admin users authenticate through Clerk. UCRM honours Clerk's **Membership required** mode: every user must belong to at least one organisation. Invitations use Clerk's built-in flow (Team → Invite in the admin, or Clerk's dashboard).

Roles are Clerk-native:

- `admin` — full access
- `basic_member` — read-only + a small list of write actions (mark list-quality items as OK, tag exclusions)

Attempted writes without permission return HTTP `403` with the code `RBAC_FORBIDDEN` and a body pointing at the required role.

4.2 API authentication

For programmatic access (adapters, ETL, integrations) UCRM issues **project-scoped API keys**. A key is minted from `/dashboard/projects/{id}/api-keys`:

- The full key is shown **once** at creation time. Store it immediately — the DB retains only the prefix + a bcrypt hash.
- Keys carry an `environment` label (`live` or `sandbox`) surfaced in headers to help debugging.
- Rotate: create a new key, deploy it, revoke the old one. There is no automatic overlap window.

Requests present the key as `Authorization: Bearer ucrm_sk_...`. Every API call resolves `tenant_id` and `project_id` from the key.

4.3 SDK

Consumers of the API can import `@ucrm/sdk`:

```
import { UCRM } from "@ucrm/sdk";

const ucrm = new UCRM({ apiKey: process.env.UCRM_API_KEY!, baseUrl: "https://api.casinocrm.io" });
const player = await ucrm.players.identify({
  external_id: "gam_1234",
  email: "user@example.com",
  default_currency_code: "EUR",
});
```

The SDK is a thin typed wrapper — everything it can do is available over plain HTTP.

---

5. Identity Architecture (Two-Layer)

UCRM identity is deliberately **two-layer** so that the platform can retain its own account-per-registration model without polluting CRM analytics with duplicates.

5.1 Layer 1 — the `players` table (Person)

One row per **Person**. Populated by identify() calls and by the ingest pipeline whenever a new external account arrives with unseen contact identifiers. Merges happen on:

- exact match of normalised `email`
- exact match of E.164-normalised `phone`
- exact match of `external_id` within the same project

`players` holds Person-level aggregates: total deposited, GGR, last login, first FTD date, KYC status, exclusion flags. When two Persons merge, aggregates are summed.

5.2 Layer 2 — the `player_platform_accounts` sub-table (PlatformAccount)

One row per **platform account** as it exists on the source (Gambitec, etc.). Fields:

- `player_id` — FK to `players` (the Person this account currently rolls up to)
- `external_id` — the source platform's user_id (unique per project)
- `platform_source` — string tag (`gambitec` , `st8` , custom…)
- `first_email` , `first_phone` , `first_name` , `last_name` — captured on first sight
- `first_identified_at` , `last_identified_at`
- `metadata` (JSONB) — anything the adapter wants to persist per-account

If a Person is the merge of 31 platform accounts (real case discovered during Shifatte fraud review), you see all 31 external_ids on the player detail page. The Person aggregates sum across them; the sub-table lets you drill down.

5.3 Fraud detection value

The sub-table exposes what the platform hides: **shared contact identifiers across many accounts**. The Identity Conflicts admin console lists Persons whose sub-table membership crosses a suspicious threshold (default: 5+ accounts sharing a phone number, 3+ accounts sharing an email prefix pattern). One-click "Mark fraud" flag flows to `players.excluded_from_stats = 'fraud'` which removes them from dashboards, reports, and messaging.

5.4 Merge rules

- **Deterministic** — merges only happen on exact identifier match, never on fuzzy heuristics.
- **Reversible** — a merged Person keeps every merged sub-table row; unmerging is a background operation that reassigns sub-table rows to a new Person and recomputes aggregates.
- **Silent for platform** — no writes go back to the platform on merge. The adapter contract only reads.

5.5 Excluded from stats

The `players.excluded_from_stats` enum takes one of:

- `null` — regular player
- `test` — internal test/dev account
- `streamer` — an influencer whose numbers should not skew reports
- `fraud` — abuse detected, excluded from revenue/GGR aggregates but kept in DB for pattern matching

Dashboards, reports, and CSV exports filter these out by default. Messaging is unaffected — an excluded player still receives transactional emails.

---

6. Currency & FX System

6.1 Storage convention

Every monetary field is stored as **subunits (integer)** in the transaction's native currency. `deposit.amount_native = 3000` in EUR means 30.00 EUR (scale 2). This mirrors Stripe's convention and eliminates floating-point drift.

The `currencies` table holds every supported ISO 4217 code and its `scale` :

Code	Scale	Notes
EUR	2	eurocent
USD	2	cent
ARS	2	centavo (though ARS rounds up in real-world use)
BRL	2	centavo
INR	2	paise
COP	2	centavo
TZS	2	senti

6.2 Base currency

Each tenant defines one **base currency** (typically EUR). All aggregates carry both a `native` value in the transaction currency and a `base` value converted via FX. Reports show both by default.

6.3 FX rates

The `fx_rates` table stores `(base, quote, rate, ts)` rows. Rate = "how many `quote` per one `base`". EUR→ARS at `1700` means 1 EUR = 1700 ARS.

The resolver `fxLookup(from, to, at)` picks the freshest rate whose `ts <= at`:

1. Direct rate `from → to`
2. Inverse rate `to → from`, then inverted
3. Triangulated through EUR (or tenant base) if neither exists
4. **Forward fallback** (added 2026-08-05) — for historic dates before the FX table had coverage, pick the earliest rate for the pair going forward. This makes cost entries backdated to before the FX module existed still convert cleanly.

6.4 Display currency

Operators can toggle the admin's display currency independently of the ledger's base currency. The choice is per-user and stored in Clerk metadata. Ledger writes always go in native + base.

---

7. Internationalisation (i18n)

7.1 Language resolution order

For any outbound message the resolver picks a language in this order:

1. **`players.language`** — explicit per-player language (set by operator, or captured via adapter).
2. **`tenants.config.i18n.default_language_by_currency[player.default_currency_code]`** — currency-derived fallback (e.g. ARS → es).
3. **`tenants.config.i18n.default_language`** — tenant-wide default (e.g. `en`).
4. **`"en"`** — final safety net.

Language codes are always lowercased 2- or 3-char ISO 639. Regional tags like `es-AR` are stripped down to `es` at lookup.

7.2 Template body translations

Every `message_templates` row carries two JSONB columns:

```
{
  "body_translations":    { "es": "…", "pt": "…" },
  "subject_translations": { "es": "…", "pt": "…" }
}
```

Empty means "fall back to `body` / `subject`" for that language. Editing happens in the template detail's Translations accordion (`/dashboard/templates/{id}`), one variant per supported language.

7.3 Bonus code map

The `bonus_code_map` table couples a bonus concept (`bonus_type`) with per-currency amounts:

Field	Purpose
`bonus_type`	e.g. `reactivation_7d`, `welcome_d1`, `welcome_d2`, `welcome_d3`, `reactivation_30d`
`currency`	ISO code
`code`	the platform's coupon code (`REACT7D`, `CHIP10`, …) — **same across all currencies for a given `bonus_type`**
`min_deposit_native`	subunits
`max_bonus_native`	subunits
`wager_multiplier`	e.g. 35, 40
`bonus_percentage`	e.g. 100, 150 (null for non-deposit)
`is_non_deposit`	true for free-chip promos
`spin_count`	free spins bundled with the bonus (nullable)
`spin_value_native`	subunits per spin
`bonus_lifetime_hours`	how long the code stays claimable
`external_bonus_id`	the platform's own ID for reconciliation

The resolver takes `(bonus_type, currency)` and returns the row so template renderers can inject `{{bonus.max_bonus}}`, `{{bonus.min_deposit}}`, `{{bonus.wager}}` with the right subunit-to-currency formatting.

7.4 Player language edit

The player detail page has a Language dropdown offering: Auto (fall back to tenant map) / en / es / pt / cs / pl / de / fr / it. Setting to a value overrides the currency-based resolution. Setting to Auto (null) restores the fallback chain.

---

8. Player Model & Aggregates

8.1 Core columns

The `players` table (abbreviated):

```
id                    uuid PK
tenant_id, project_id  scoping
external_id           source platform id (unique per project)
email, phone          deduplication identifiers (normalised on write)
first_name, last_name
default_currency_code  ISO 4217
language              ISO 639 (nullable — see §7.1)
kyc_status            none|pending|verified|rejected
country_code          ISO 3166-1 alpha-2
signed_up_at          ts
first_deposit_at      ts (nullable until FTD)
last_login_at
last_deposit_at
last_bet_at
excluded_from_stats   null|test|streamer|fraud
excluded_at           ts (set with excluded_from_stats, cleared with it)
metadata              jsonb
```

8.2 Aggregates

`player_aggregates` is a materialised summary refreshed by the stream processor whenever a money event arrives:

- `total_deposited_native` / `_base`
- `total_withdrawn_native` / `_base`
- `total_wagered_native` / `_base`
- `total_won_native` / `_base`
- `ngr_native` / `_base` (wagered − won)
- `deposit_count`, `withdrawal_count`, `bet_count`
- `first_ftd_amount_native` / `_base`
- `days_since_last_deposit`, `days_since_signup`
- `session_count_last_30d`

8.3 Lifecycle events

- `player_created` — first identify() call.
- `player_updated` — email/phone/name/language change.
- `player_kyc_updated` — status transitions.
- `player_excluded` — added to exclusion list (fraud, streamer, test).
- `player_self_excluded` — RG self-exclusion active. Immediately suppresses all messaging.
- `player_merged` — two Persons combined into one.

All emit domain events consumed by ClickHouse for analytics and by BullMQ for downstream side effects (e.g. self-exclusion triggers a suppression list write).

---

9. Messaging Subsystem

9.1 Providers

Every physical delivery route is a **provider**: an SMTP endpoint, a Twilio Messaging Service, a Telegram bot, a WhatsApp Business number. The `messaging_providers` table stores:

- `provider` — kind (`elastic-email` , `postmark` , `twilio` , `grammy` , `webhook`)
- `channel` — target channel (`email` , `sms` , `whatsapp` , `telegram` , `push` , `webhook`)
- `config_encrypted` — provider-specific config, encrypted at rest (Twilio SID + auth token, ESP API key, sender identifiers)
- `rate_limit_per_minute` — polite ceiling
- `frequency_cap_per_24h` — per-player anti-fatigue cap
- `is_default` — one default per (channel, project)
- `status` — active/paused/errored

Providers are configured under `/dashboard/providers` . Each provider type has a dedicated editor with the config fields it needs and inline hints (e.g. Twilio's Multi-GEO Messaging Service SID field surfaces a hint about alphanumeric sender registration per country).

9.2 Send flow

1. Producer (campaign/journey/direct call) enqueues a `message_send` row and a Bull job onto `q:{channel}` .
2. `messaging-worker` picks up, loads context (`player` , `template` , `provider` , `tenant.config` , `bonus_code_map` if `{{bonus.*}}` is referenced).
3. Resolver picks language (see §7.1) and body (see §7.2).
4. Template renderer runs Liquid-style variable expansion.
5. Provider SDK sends. Provider message id captured back into `message_sends.provider_message_id` .
6. Status callback webhook (Twilio delivery receipt, ESP bounce webhook) updates `message_sends.status` and drives suppression list writes on bounces.

9.3 Frequency cap

Before enqueue, the API checks per-player per-24h send count across all channels. Configurable per provider (`frequency_cap_per_24h`) and globally per tenant. If exceeded, the send is not enqueued; the intent is logged with status `capped` for audit.

9.4 Suppression lists

The `messaging_suppressions` table holds per-tenant per-address suppression entries:

- `address` (email or phone)
- `channel`
- `reason` — `bounce_hard` , `unsubscribe` , `spam_complaint` , `stop_keyword` , `manual`
- `source_message_send_id`
- `created_at`

Every send does a fast INDEX lookup here first; a hit blocks the send with status `suppressed` . Reasons are surface-relevant, not just for audit — a `bounce_hard` blocks that address forever; a `stop_keyword` blocks only that channel; a `manual` can be lifted by an operator via the list-quality review workflow.

---

10. Templates

10.1 Schema

Templates carry a body per channel, categorisation, per-language variants, and a declared variable list:

```
id                   uuid
tenant_id, project_id
name                 text
channel              email|sms|push|telegram|webhook|whatsapp
subject              text (required for email, null otherwise)
body                 text (HTML for email, plain for other channels)
variables            jsonb array of merge-tag names
status               draft|active|archived
category             welcome|deposit|bonus|retention|reactivation|
                     vip|promo|kyc|transactional|compliance|other
body_translations    jsonb { lang: body }
subject_translations jsonb { lang: subject }
```

10.2 Variables

Templates use Liquid-style variables:

- `{{first_name | default: "player"}}`
- `{{event.amount_formatted}}` — event context passed at enqueue time
- `{{current_tier_code}}` — VIP tier
- `{{bonus.max_bonus}}`, `{{bonus.min_deposit}}`, `{{bonus.wager}}` — resolved from `bonus_code_map` for the player's currency

An unknown variable renders as an empty string, never throws. This keeps rendering total on bad content but shifts responsibility to preview.

10.3 Preview & send-test

The template detail page has a **live preview** (right pane, renders the current body with sample variable values) and a **send-test** panel (send to your Clerk email address with the currently-selected provider). Both use the same renderer that production sends will use.

10.4 Categories

Categories drive:

- Sidebar filters in the campaign / journey step editors ("show me only reactivation templates").
- Suppression policy — `transactional` bypasses the frequency cap (never delay a deposit-confirmation email).
- Default provider — a per-channel per-category default provider mapping.

10.5 Live retention templates

TIKETABETcom currently runs 7 EN retention templates, each with a full Spanish (`es`) variant:

Template	Category	Fires
Welcome — production	welcome	on signup
Winback — welcome bonus (never deposited)	reactivation	day 3 no-FTD
Reactivation 7d — REACT7D (100% match $80)	reactivation	day 7 no-FTD
Reactivation 14d — REACT14D (150% match $500)	reactivation	day 14 no-FTD
Reactivation 30d — CHIP10 (no-dep $10)	reactivation	day 30 no-FTD
Reactivation 90d — Sunset re-opt-in	reactivation	day 90 no-FTD
Winback — cards + bank live (lapsed depositor)	reactivation	segment-triggered

Plus 3 SMS chase templates (`Reactivation 10d/17d/33d SMS`), each with `es` variant, sent 3 days after the corresponding email.

Plus 2 transactional templates without ES variants yet (Deposit successful, Withdrawal confirmed).

---

11. Campaigns

A **campaign** is a one-shot broadcast to a segment. Unlike journeys, campaigns run once and terminate — no scheduling loop, no per-player state.

11.1 Lifecycle

`draft` → `scheduled` → `sending` → `completed` (or `cancelled` / `errored`).

11.2 Preflight

Before sending, the campaign editor runs a **Preflight simulator** (`/dashboard/campaigns/{id}/preflight`):

- Resolves the target segment against current player rows.
- Counts recipients per channel.
- Estimates provider cost (per-segment SMS cost with GSM-7 vs Unicode detection).
- Warns on suppressed / self-excluded / frequency-capped recipients.
- Warns on missing translations for the currency mix ("47 of 193 AR recipients would fall back to EN because the template has no `es` variant").

You cannot send until preflight is Green or you explicitly override with a comment.

11.3 Delivery

At `scheduled_at`, the API expands the segment into individual `message_sends` rows and enqueues a job per recipient. The worker respects rate limits (`rate_limit_per_minute` on the provider) so no thundering herd.

11.4 Analytics

Per-campaign metrics computed from `message_sends` + delivery-status webhooks:

- Sent / Delivered / Bounced / Opened / Clicked / Unsubscribed
- Conversion (bonus code redemption if wired via ingest, deposit within N days of send)

---

12. Journeys (Automation Graphs)

12.1 Model

A **journey** is an entry-triggered, deterministic graph of steps that advance one player at a time. Stored as JSONB in `journeys.graph`:

```
{
  "start_node_id": "wait_3d",
  "nodes": [
    { "id": "wait_3d",   "type": "wait", "duration_ms": 259200000, "next": "check_3d" },
    { "id": "check_3d",  "type": "condition",
      "when": { "op": ">", "field": "total_deposited", "value": 0 },
      "if_true_next": "done", "if_false_next": "send_3d" },
    { "id": "send_3d",   "type": "send_message",
      "provider_id": "...", "template_id": "...", "next": "wait_7d" },
    { "id": "done",      "type": "end" }
  ]
}
```

12.2 Node types

Type	Fields	Behaviour
`wait`	`duration_ms`, `next`	Schedule advancement to `next` after `now + duration_ms`.
`condition`	`when`, `if_true_next`, `if_false_next`	Evaluate `when` against the player's current aggregates. Branch.
`send_message`	`provider_id`, `template_id`, `next`	Enqueue a message send. Advance immediately (delivery is async).
`webhook`	`url`, `body_template`, `next`	POST to an external URL. Retries with backoff.
`end`	—	Terminate the run.

12.3 Triggers

Journeys are triggered by:

- `signup` — every new player that passes tenant filters.
- `first_deposit` — first successful deposit.
- `deposit_confirmed` — every deposit (transactional flow).
- `withdrawal_confirmed` — every withdrawal.
- `segment_entered` — player becomes a member of a segment.
- `manual` — API-triggered, one-off (e.g. Call Manager tool).

Triggers stored as `journeys.trigger` JSONB:

```
{ "type": "signup", "filters": { "excluded_from_stats": null } }
```

12.4 Runs

Each triggered entry creates a `journey_runs` row:

```
id           uuid
journey_id   FK
player_id    FK
current_node_id
next_wake_at  ts (nullable)
status       running|completed|cancelled|errored
context      jsonb (variables captured at entry)
```

The `journey-worker` polls for runs whose `next_wake_at <= now()`, advances the node, updates state, exits. Idempotent: crash mid-advancement resumes cleanly on next poll.

12.5 Live journeys (TIKETABETcom)

Journey	Trigger	Nodes	Channels
Welcome — on signup	signup	3	email
Deposit successful	deposit_confirmed	3	email
Withdrawal confirmed	withdrawal_confirmed	3	email
Reactivation — never deposited (3D/7D/14D/30D/90D)	signup + no-FTD filter	25	email × 5 + sms × 3

The Reactivation journey is the most complex: it interleaves emails at days 3/7/14/30/90 with SMS chases at days 10/17/33, with a `condition` guard before every send that exits the run if the player deposited in the interim.

12.6 Editing

Journey editor is a visual DAG at `/dashboard/journeys/{id}`. Add / delete / rewire nodes with drag-and-drop; the underlying JSON is what gets saved. Manual JSON editing is supported for advanced flows via the "Show raw" toggle.

12.7 Pause / resume

Setting `journeys.status = 'paused'` freezes new entries. Existing runs continue advancing; to freeze runs too, cancel them individually or run a batch cancel from the admin.

---

13. Segments

A **segment** is a saved query over players. Two kinds:

13.1 Rule-based

A serialised set of predicates on player columns and aggregates:

```
{
  "type": "and",
  "conditions": [
    { "field": "default_currency_code", "op": "in", "value": ["ARS"] },
    { "field": "total_deposited_native", "op": "=", "value": 0 },
    { "field": "days_since_signup", "op": ">=", "value": 7 },
    { "field": "excluded_from_stats", "op": "is_null" }
  ]
}
```

Editable in `/dashboard/segments/{id}` with a form. Preview count is live (query runs against current player rows).

13.2 ClickHouse-backed

For behavioural segments needing bet events, session events, or aggregates ClickHouse computes better than Postgres:

```
{
  "type": "clickhouse",
  "query": "SELECT player_id FROM bet_events_1d WHERE bet_count > 50 AND ngr < -100"
}
```

The query is stored as SQL, validated on save (no DDL, no cross-tenant leaks — the API prepends a tenant filter automatically).

13.3 Freshness

Segments are re-evaluated at every reference (campaign send, journey trigger, list export). No caching — always current.

---

14. Bonus Code Map

Covered in §7.3. Not much more to say — this is a small, high-value table.

Data load pattern: an ETL script or a manual admin form syncs rows from the platform's own bonus definitions. For TIKETABETcom, `bonus_code_map` is populated from Gambitec's `/bonuses/{id}` endpoint definitions:

```
Gambitec bonus #87 (REACT14D, 150% match)
  → 7 rows in bonus_code_map (one per supported currency: EUR/USD/INR/COP/TZS/ARS/BRL)
```

Each row carries the currency-specific min-deposit and max-bonus in native subunits, all sharing the same `code` string (`REACT14D`).

Reconciliation: `external_bonus_id` on each row references the platform's own ID. A daily job (planned) will re-sync all rows and flag drift.

---

15. Reports & CSV Exports

15.1 Report types

- **Daily summary** — depositors, deposit count, deposit sum, withdrawals sum, GGR, NGR — per calendar day in tenant TZ.
- **Wagering** — per-player wagering totals over date range.
- **Bonus grants** — per-player bonus grants + redemptions + expirations.
- **Top players** — sorted by ngr / deposit sum / bet count.

15.2 Generation

Reports are generated on demand via `/dashboard/reports/{kind}?from=...&to=...&currency=...`. Long-running reports enqueue a background job and email the operator when the CSV is ready.

15.3 Currency handling

Every CSV amount column is presented in **the tenant's base currency, scaled to human-readable units** (not subunits). Column headers include the currency code — `deposit_amount_eur` , `ngr_eur` . This was rebuilt 2026-08-05 after a bug where raw subunits leaked into headers (`_amount_base` showed `2500000` instead of `25,000.00 EUR`).

15.4 Exclusion filter

`excluded_from_stats` players are filtered out by default. A checkbox on the report form re-includes them (needed for e.g. fraud audit).

15.5 Drill-down

Every dashboard tile is clickable and opens a filtered drill-down table showing the individual rows behind the number. Same currency conventions as CSVs.

---

16. Dashboards

16.1 Main dashboard

Landing page after login. Shows:

- **24h KPIs** — depositors, deposit sum, withdrawal sum, FTD count, NGR, DAU
- **30d trend suite** — sparkline per KPI, hover for exact daily value
- **Manager health metrics** — bounce rate 7d, unsubscribe rate 7d, active journey run count, list quality issues open

All numbers in tenant display currency. All filtered by `excluded_from_stats` unless "Include excluded" is toggled.

16.2 Segmentation views

- `/dashboard/players` — global player table with column-level filters (country, currency, KYC status, exclusion status)
- `/dashboard/segments` — segment library with recipient counts
- `/dashboard/journeys` — journey library with run stats + active/paused status

16.3 Detail pages

- `/dashboard/players/{id}` — full player picture: aggregates, event history, journey memberships, sends history, platform accounts (from sub-table), language + exclusion toggles.
- `/dashboard/templates/{id}` — editor + preview + send-test + translations.
- `/dashboard/journeys/{id}` — graph editor + run stats + recent-runs table.
- `/dashboard/campaigns/{id}` — campaign details + preflight + delivery stats.

---

17. VIP Tiers

Per-tenant ladder with auto-promotion thresholds. Migration seeds Bronze → Silver → Gold → Platinum by default. Editable at `/dashboard/settings/vip-tiers`.

Each tier has:

- `code` — the internal identifier (`BRONZE` , `PLATINUM`)
- `display_name` — human label
- `promotion_threshold` — an object like `{ metric: "total_deposited_base", op: ">=", value: 500000 }` (5000 base currency)
- `perks` — free-text list of perks to render in the tier badge tooltip
- `bonus_multiplier` — how much bonus max is scaled for this tier (default 1.0)

A background job runs nightly and promotes players who cross a threshold. Demotions are manual (never automatic — regulatory reasons).

VIP tier is exposed as `{{current_tier_code}}` in templates.

---

18. List Quality & Insights

`/dashboard/list-quality` (channel-scoped: email or SMS) surfaces:

- **Bounce rate** — hard vs soft, trending over last 30 days
- **Unsubscribe rate**
- **Frequency cap hits** — how often the cap fires (a signal that your journey is overloading recipients)
- **Suspicious address patterns** — emails at disposable domains, phone numbers with wrong-length pattern per country

Each row surfaces an action menu:

- **Suppress** — write a `manual` entry into `messaging_suppressions`
- **Mark OK** — flag the row as reviewed so it doesn't re-surface tomorrow
- **Open player** — deep link to the underlying player

18.1 SMS-specific insights

Reachability by country (share of numbers passing the phone-classifier's country/mobile check), share of Unicode-forced sends (expensive!), share of alphanumeric-sender-eligible destinations.

---

19. Identity Conflicts Console

`/dashboard/identity-conflicts` lists Persons whose `player_platform_accounts` sub-table shows cross-account patterns worth reviewing:

- Multiple external_ids on a shared phone
- Email prefix families sharing a phone (e.g. `user1@…` , `user2@…` , `user3@…` all on the same phone)
- IP or geo overlap across accounts (if the adapter provides IP)

Each cluster shows:

- All member accounts with their external_ids, emails, phones, GGR
- A "Mark cluster as fraud" button — flips every member's `excluded_from_stats = 'fraud'` and adds an audit-log entry with the operator's Clerk id
- A "Un-exclude" button per member — for false positives

The console is the operational face of the two-layer identity model (§5).

---

20. Operational Costs

`/dashboard/costs` — a simple, non-accounting cost ledger for the CRM operator:

- Enter a cost (amount, currency, vendor, category, date, notes)
- Attach a file (invoice PDF, screenshot) — stored on-disk under `/var/www/costs`
- Categorise (SMS, email, hosting, tooling, contractor)

Costs are converted to base currency at the entered date's FX rate (with the forward-fallback resolver — §6.3). The `/dashboard/costs` view aggregates by category / vendor / month.

Purpose: pair spend against revenue on the same currency basis when calculating campaign ROI.

---

21. Compliance (GDPR, Audit, RG)

21.1 Audit log

`audit_log` table records every mutation performed via the admin console or the API. Fields:

- `at`, `user_id` (Clerk id), `tenant_id`, `project_id`
- `resource_type`, `resource_id`, `action` (`create|update|delete|exclude|merge|...`)
- `before` / `after` (JSONB snapshots for update actions)

Append-only. Never soft-deleted, never edited. Retention indefinitely for now.

Browsable at `/dashboard/audit` with filters by user, resource, date range.

21.2 GDPR export

From a player's detail page, "Export data" produces a machine-readable JSON bundle containing:

- The player row + all sub-table rows
- All aggregates
- All message sends (with rendered bodies where available)
- All journey run rows
- All audit-log entries mentioning this player

Delivered as a download or emailed to a designated data-protection officer contact.

21.3 GDPR delete

"Delete player" is a hard delete. Confirms twice. Cascades to sub-table, aggregates, sends (recipients replaced with `[REDACTED]`), journey runs. Audit-log entries mentioning the player are pseudonymised (id kept, personal fields nulled) so audit trail integrity is preserved.

21.4 Self-exclusion

Setting `player_self_excluded_until` (from the player detail page or via the adapter contract) immediately writes suppression entries for every known contact address of that player. Messages already in the queue are cancelled at worker pick-up. Journeys drop the player at the next `condition` node.

21.5 RG features

Tenant `config.responsible_gaming` controls:

- `self_exclusion_min_days` — minimum SE duration
- Deposit limit reminders (planned)
- Reality-check nudges (planned)

---

22. Adapter Integration Pattern

The adapter is the contract between UCRM and the gaming platform of record. Every event ingested and every write pushed back into the platform go through it.

22.1 Contract shape

An adapter implements:

```
interface Adapter {
  readonly name: string;                          // "gambitec", "st8", "yourvendor"

  // Inbound: platform → UCRM
  ingest(payload: unknown): Promise<IngestResult>; // parse platform's event
  verifyIngestSignature(headers, rawBody): boolean;

  // Outbound (read-only): UCRM → platform
  fetchPlayer(externalId: string): Promise<PlatformPlayer>;
  fetchDeposits(externalId: string, from: Date): Promise<Deposit[]>;
  fetchBets(externalId: string, from: Date): Promise<Bet[]>;
  fetchBonuses(externalId: string): Promise<BonusInstance[]>;
}
```

UCRM never calls a write method — no `updatePlayer`, no `awardBonus`, no `debitWallet`. All bonus grants and player mutations go through the platform's own admin, and UCRM merely observes them via `ingest()` afterwards.

22.2 Ingest surface

The API exposes `POST /adapters/{platform}/ingest`. HMAC-signed with the platform's shared secret. Body is the raw platform event JSON, unmodified. The adapter's `ingest()` parses it into UCRM-native shapes:

```
type IngestResult =
  | { kind: "player_created", player: PlayerFields }
  | { kind: "deposit_succeeded", externalId: string, amount_native: number, currency: string, at: string, method: string, dedu
  | { kind: "withdrawal_succeeded", ... }
  | { kind: "bet_placed", ... }
  | { kind: "bet_settled", ... }
  | { kind: "session_started", ... }
  | { kind: "session_ended", ... }
  | { kind: "bonus_granted", ... }
  | { kind: "bonus_redeemed", ... }
  | { kind: "bonus_expired", ... }
  | { kind: "kyc_updated", ... }
  | { kind: "self_excluded", ... };
```

Every event carries a `dedup_key` (typically the platform's own event id). The stream processor dedupes on `(tenant_id, dedup_key)` before persisting. Idempotent by construction.

22.3 Backfill

Adapters expose a bulk-backfill mode:

```
docker compose exec api node scripts/adapter-backfill.js \
  --adapter gambitec --project TIKETABETcom-prod \
  --from 2026-01-01 --to 2026-08-01 \
  --kinds player_created,deposit_succeeded,withdrawal_succeeded
```

Reads from the platform's export endpoints in pages, streams into the same `ingest()` pipeline. Backfills are safe to re-run — dedup handles overlap.

22.4 Writing a new adapter

Skeleton lives at `packages/adapter-zero` — a no-op reference implementation with a full test suite. To ship a new adapter:

1. Copy `adapter-zero` → `adapter-{yourvendor}`.
2. Implement `verifyIngestSignature()` — the vendor's webhook signing method (HMAC-SHA256, HMAC-SHA1, ECDSA-P256 for St8, etc.).
3. Implement `ingest()` — map each vendor event kind onto one of the `IngestResult` variants.
4. Implement the read-only fetch methods against the vendor's REST API.
5. Register the adapter in `apps/api/src/adapters/index.ts`.
6. Add config UI: `/dashboard/settings/adapter/{yourvendor}` wired to the encrypted config JSON.
7. Run the integration test suite: `pnpm test:adapter -- --adapter yourvendor` — includes fixture-based ingest tests and a live smoke test against the vendor's sandbox.

22.5 Two shipped adapters

- **gambitec** — the primary integration for TIKETABETcom. Handles 12 event kinds. Backfilled 2026 Jan–Aug into staging + prod.
- **st8** — aggregator bridge for St8's seamless wallet callbacks. 8 callback kinds. ECDSA-signed. See `services/games` for the wallet endpoints.

---

23. REST API Reference

All endpoints are `https://api.casinocrm.io/v1/...`. Every request must present:

```
Authorization: Bearer ucrm_sk_...
Content-Type: application/json
```

Responses are JSON. Errors follow:

```
{ "error": { "code": "PLAYER_NOT_FOUND", "message": "No player with id ..." } }
```

Status codes: `200` success, `201` created, `204` no content, `400` bad request, `401` unauthorised, `403` forbidden, `404` not found, `409` conflict, `422` validation error, `429` rate limited, `500` server error.

Pagination for list endpoints: cursor-based. Responses carry `meta.next_cursor` (null when done). Pass `cursor=...` on the next request. Default `limit=50`, max `limit=200`.

23.1 Players

```
POST   /v1/players/identify               # upsert by external_id / email / phone
GET    /v1/players?cursor=&limit=&q=
GET    /v1/players/{id}
PATCH  /v1/players/{id}
DELETE /v1/players/{id}                   # GDPR hard delete
POST   /v1/players/{id}/exclude           # body: { reason, note }
DELETE /v1/players/{id}/exclude
POST   /v1/players/{id}/merge             # body: { into_player_id }
GET    /v1/players/{id}/platform-accounts
GET    /v1/players/{id}/aggregates
GET    /v1/players/{id}/sends             # message send history
GET    /v1/players/{id}/journey-runs
POST   /v1/players/{id}/gdpr-export       # returns job id; poll status
```

`POST /v1/players/identify` body:

```
{
  "external_id": "gam_1234",
  "email": "u@example.com",
  "phone": "+5493425112263",
  "first_name": "Ana",
  "last_name": "Diaz",
  "default_currency_code": "ARS",
  "language": "es",
  "country_code": "AR",
  "signed_up_at": "2026-08-01T12:00:00Z",
  "platform_source": "gambitec"
}
```

Returns `{ player_id, is_new, merged_into_existing }`. If identifiers matched an existing Person, `merged_into_existing = true` and the sub-table gets a new row for this `external_id`.

23.2 Templates

```
GET    /v1/templates?cursor=&limit=&channel=&status=&category=&q=
POST   /v1/templates
GET    /v1/templates/{id}
PATCH  /v1/templates/{id}
DELETE /v1/templates/{id}                 # soft delete
POST   /v1/templates/{id}/preview         # body: { player_id | sample_context }
POST   /v1/templates/{id}/send-test       # body: { to, provider_id?, language? }
```

`POST /v1/templates` body:

```
{
  "name": "Reactivation 7d — REACT7D",
  "channel": "email",
  "subject": "TIKETABET · 100% match up to $80",
  "body": "<html>...</html>",
  "body_translations": { "es": "<html lang=\"es\">...</html>" },
  "subject_translations": { "es": "TIKETABET · 100% de match hasta $80" },
  "variables": ["first_name", "bonus.max_bonus"],
  "status": "active",
  "category": "reactivation"
}
```

23.3 Campaigns

```
GET    /v1/campaigns?cursor=&limit=&status=
POST   /v1/campaigns
GET    /v1/campaigns/{id}
PATCH  /v1/campaigns/{id}
POST   /v1/campaigns/{id}/preflight        # returns preflight report
POST   /v1/campaigns/{id}/schedule         # body: { scheduled_at }
POST   /v1/campaigns/{id}/cancel
GET    /v1/campaigns/{id}/deliveries       # sends + delivery status
GET    /v1/campaigns/{id}/analytics
```

23.4 Journeys

```
GET    /v1/journeys?cursor=&limit=&status=&trigger_type=
POST   /v1/journeys
GET    /v1/journeys/{id}
PATCH  /v1/journeys/{id}                   # body may include full graph replacement
POST   /v1/journeys/{id}/activate
POST   /v1/journeys/{id}/pause
GET    /v1/journeys/{id}/runs?cursor=&limit=&status=
POST   /v1/journeys/{id}/runs/{run_id}/cancel
```

23.5 Segments

```
GET    /v1/segments?cursor=&limit=
POST   /v1/segments
GET    /v1/segments/{id}
PATCH  /v1/segments/{id}
DELETE /v1/segments/{id}
POST   /v1/segments/{id}/preview           # returns { count, sample_player_ids }
GET    /v1/segments/{id}/players?cursor=&limit=
```

23.6 Providers

```
GET    /v1/providers?channel=
POST   /v1/providers
GET    /v1/providers/{id}
PATCH  /v1/providers/{id}
DELETE /v1/providers/{id}
POST   /v1/providers/{id}/test             # send a canary message
```

Provider config body is per-provider-kind. Example Twilio:

```
{
  "name": "Twilio SMS Multi-GEO",
  "provider": "twilio",
  "channel": "sms",
  "config": {
    "account_sid": "AC...",
    "auth_token": "...",
    "messaging_service_sid": "MG1a06dcec926d01e6983c483492c31404",
    "status_callback_url": "https://api.casinocrm.io/hooks/twilio/{provider_id}"
  },
  "rate_limit_per_minute": 60,
  "frequency_cap_per_24h": 2
}
```

23.7 Bonus code map

```
GET    /v1/bonus-code-map?bonus_type=&currency=&code=
POST   /v1/bonus-code-map                  # upsert on (bonus_type, currency)
PATCH  /v1/bonus-code-map/{id}
DELETE /v1/bonus-code-map/{id}
```

23.8 Reports

```
GET    /v1/reports/daily-summary?from=&to=&currency=
GET    /v1/reports/wagering?from=&to=&currency=&format=csv|json
GET    /v1/reports/bonus-grants?from=&to=
GET    /v1/reports/top-players?from=&to=&sort=ngr|deposit_sum&limit=
```

CSV responses stream with `Transfer-Encoding: chunked`. Include `Accept: text/csv` to force CSV; `application/json` for JSON.

23.9 Ingest (adapters)

```
POST   /adapters/{adapter}/ingest
```

Body: raw platform event. Headers include the platform's HMAC signature. Response `202 Accepted` when queued for stream processing, `400` when signature invalid or body malformed.

23.10 Webhooks

Outbound webhooks (from UCRM to your systems) are registered per-tenant at `/dashboard/settings/webhooks`. Each webhook subscribes to a set of event kinds (`player_created`, `deposit_succeeded`, `campaign_completed`, `journey_run_completed`, etc.).

Delivery attempts retry with exponential backoff up to 24 h. Failed deliveries visible at `/dashboard/webhook-deliveries` for replay.

23.11 Rate limits

Per-API-key: 1200 requests/minute default. `429` includes `Retry-After` header.

Bulk endpoints (`/players/bulk-identify`, `/campaigns/{id}/deliveries`) have separate higher limits.

---

24. Admin UI Guide

24.1 Sidebar layout

- Dashboard
- Players
- Segments
- Templates
- Campaigns
- Journeys
- Reports
- List Quality (email, sms)
- Identity Conflicts
- Bonus Codes
- Providers

- Costs
- Audit
- Settings
  - Tenant
  - Projects
  - VIP Tiers
  - Language & i18n
  - Webhooks

24.2 Top bar

- Organisation switcher (Clerk) — switch tenant
- Project switcher — switch active project within tenant
- Display currency picker — global for the operator's view
- User menu (Clerk) — profile, sign out

24.3 Keyboard shortcuts

- `⌘K` — command palette (jump to any player, template, journey by name)
- `⌘/` — global search
- `g` then any nav letter — go to that section (`gp` = players, `gt` = templates)

24.4 Bulk operations

Player list and message-send list support multi-select via checkboxes. Bulk actions:

- Add to segment
- Exclude / un-exclude
- Set language
- Merge (players only)
- Cancel (sends only)

24.5 Search

The player search box supports:

- Free-text on name/email/phone
- Prefix `id:` for a player id
- Prefix `ext:` for a platform external_id
- Prefix `email:`, `phone:` for exact-match

---

25. Deployment & Infrastructure

25.1 Environments

Three tiers on a single Hetzner box (physical separation would be next step):

- **dev** — engineers push here first, plaintext secrets acceptable
- **staging** — pre-prod, real-shape data (anonymised copy of prod), TLS
- **prod** — customer traffic, encrypted secrets, restricted SSH

Each tier is its own Docker Compose project on the same host. Ports namespaced.

25.2 Services on the box

```
ucrm-postgres         Postgres 16 with tenant RLS
ucrm-redis            Redis 7 (BullMQ)
ucrm-clickhouse       ClickHouse 24
ucrm-api              Fastify API
ucrm-admin            Next.js admin
ucrm-messaging-worker BullMQ consumer
ucrm-journey-worker   Journey scheduler
ucrm-event-ingest     Adapter ingest surface
ucrm-caddy            TLS + routing (staging), Nginx in prod
```

25.3 Deployment script

`infra/scripts/staging-deploy.sh` does:

1. `git pull` on the deployment branch
2. Build Docker images
3. Run pending Postgres migrations
4. `docker compose up -d --force-recreate` for services with changed images (Caddy always force-recreates due to bind-mount rewrites — see the known-issues note)
5. Smoke-test the API via `/healthz`

Same script structure for prod with a manual approval gate.

25.4 Backups

- Postgres: daily `pg_dump` to on-disk snapshot in `/var/backups/postgres/YYYY-MM-DD/`. Retained 30 days. Off-box object storage deferred until first paying client.
- ClickHouse: relies on replication, no explicit backup yet.
- File attachments (`/var/www/costs`): daily rsync to a second disk on the same box.

25.5 Secrets

Encrypted per-service `.env.encrypted` decrypted at compose-up time by an `age`-keyed helper. Master key lives in a hardware token controlled by the ops lead. Rotating a provider API key means editing the file, re-encrypting, redeploying that service only.

25.6 Observability

- **Logs** — every service emits JSON logs; Docker's `json-file` driver, tailed by an on-box script that ships to a lightweight Loki instance.
- **Metrics** — Prometheus scraping `/metrics` on API + workers; G