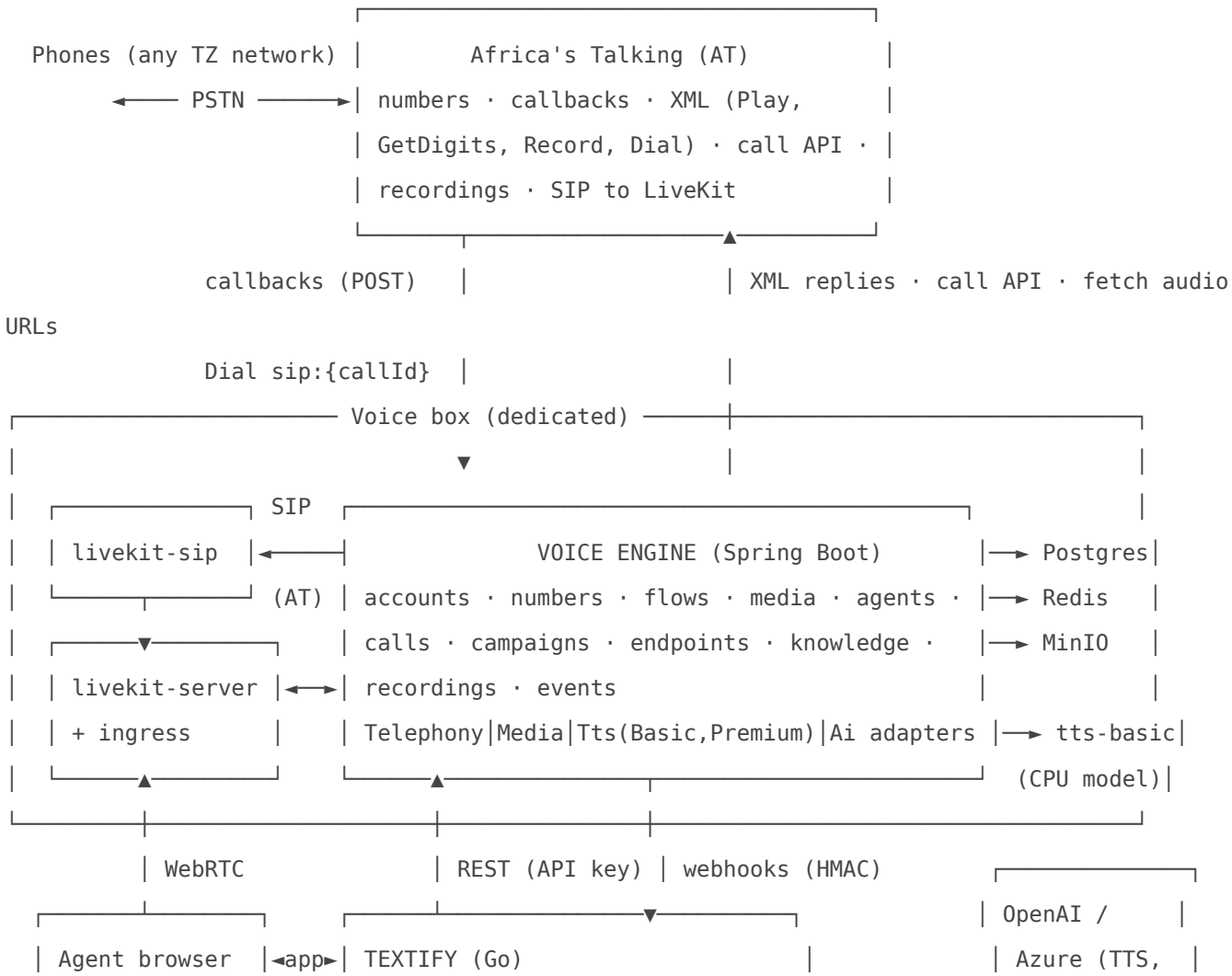
Ringmaster Voice: Architecture

> Companion to `SPEC.md` (what/why) and `FLOWS.md` (flow & action reference). This file is **how**: components, adapters, call paths, media & voice, knowledge base, data model, APIs, auth, communication, infra, tools. Stack: **Textify (Go) → Voice Engine (Java 21, Spring Boot) → Africa's Talking + LiveKit (self-hosted) + TTS (Basic/Premium) + OpenAI + MinIO**. No Python, no message broker. **[VERIFY]** = check against provider docs or Africa's Talking before coding against it.

---

1. Overview



```
   │ (Textify UI +  │     │ users · tenants · wallet · prices│            │ transcribe,  │
   │ LiveKit JS)    │     │ UI · contacts · tickets · SMS/WA │            │ summary, AI) │
   └────────────────┘     └──────────────────────────────────┘            └──────────────┘

                                          customer systems ◄── webhook actions (signed) ──
engine
```

Rules that hold the design together

1. **AT is the only road to the phone network**, both directions.
2. **Listening =** `Play`**, keys =** `GetDigits`**, voicemail =** `Record`**, live =** `Dial` **→ LiveKit.** `Say` is not used.
3. **Waiting for an agent happens on AT** (hold loop); the call enters LiveKit only once an agent accepted.
4. **No TTS during live calls.** Text becomes audio files at save time / before dialing.
5. **The engine is blind**: `accountId`, `agentRef`, numbers, calls. Never users, roles, prices, tickets or contacts.

1.1 Responsibilities

Component	Owns	Never owns
Textify (Go)	Users/login/roles, tenants, partners, wallet, prices, all UI (answer-mode templates, campaign screens, audio library UI, KB editor), contacts, **tickets**, screen-pop data, SMS/WhatsApp sending	Telephony / LiveKit / TTS / AI logic
Voice Engine	Client apps, accounts (opaque), numbers, flows + versions, media + generated voice, agents (opaque refs), departments, presence, waiting orders, calls, campaigns, opt-outs, endpoints, knowledge base, recordings, transcripts, usage events	People, money, tickets
AT	PSTN, numbers, XML execution, outbound dialing, recording	Anything else
LiveKit	Rooms, WebRTC, SIP from AT, agent-hold audio, Ph 3 AI participant	Business logic
TTS providers	Turning text into audio files	State
OpenAI	Ph 2 transcribe/summarize; Ph 3 live AI	State, routing

---

2. Inside the engine

Stack: Java 21, Spring Boot 3.x, PostgreSQL + Flyway, Redis, `@Scheduled` workers, virtual threads, ffmpeg in the image.

2.1 Adapters (plain interfaces; no plugin framework)

```
interface TelephonyAdapter {                // impl: AfricasTalkingAdapter
  String render(List<XmlAction> actions);                  // → AT XML (Play, GetDigits,
Record, Dial, Reject)
  List<OutboundResult> placeCalls(String from, List<String> to);
  InputStream downloadRecording(String url);
}

interface MediaAdapter {                    // impl: LiveKitAdapter
  String sipUriFor(UUID callId);                           // sip:{callId}@sip.voice…
  String roomToken(String room, String identity, boolean publish);
  void sendData(String room, Object msg);                  // ring / ring_cancel /
connecting
  void startHoldAudio(String room, String url); void stopHoldAudio(String room);   // agent
hold
  void removeParticipant(String room, String identity); void deleteRoom(String room);
  // Ph 3: dialAi(room, callId)
}

interface TtsAdapter {                      // impls: BasicTtsAdapter (self-hosted),
PremiumTtsAdapter (OpenAI or Azure)
  byte[] synthesize(String text, String lang, String voice);
  List<VoiceOption> voices();
}

interface AiAdapter {                       // impl: OpenAiAdapter
  Transcript transcribe(InputStream audio, String langHint); // Ph 2
  Summary summarize(Transcript t, String accountContext);    // Ph 2
  // Ph 3: acceptLiveCall(openAiCallId, SessionConfig), control WebSocket, hangup
}
```

A `TtsRouter` picks the adapter by `tier`. Swapping a provider = new class + config.

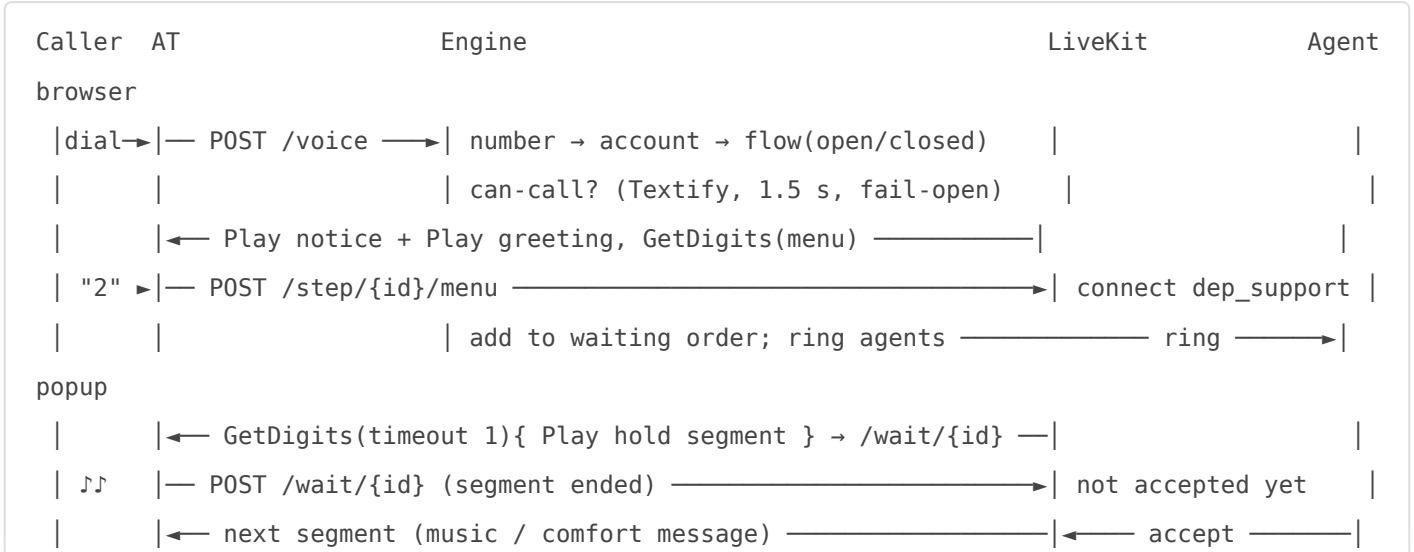
2.2 Packages

```
com.qbitspark.voice
├── api          // client REST /v1/** (API-key auth)
├── hooks        // /hooks/at/{secret}/**, /hooks/livekit, /hooks/openai
├── telephony    // AT adapter, XML builder
├── media        // LiveKit adapter, tokens, lobby/ring, ingress (agent hold)
├── voice        // TtsRouter, Basic/Premium adapters, Swahili text normalizer, generation
cache
├── library      // media assets: upload, validate, ffmpeg transcode, signed play URLs
├── flows        // model, validator, versions, NodeExecutor per node type (FLOWS.md)
├── waiting      // hold loop, waiting order per department, segment builder
├── routing      // departments, presence, ring strategies, claim (first accept wins)
├── calls        // lifecycle, state machine, context, trail
├── campaigns    // campaigns, recipients, pre-generation, dispatcher, retries, opt-outs
├── endpoints    // customer webhook endpoints, signer, SSRF guard, sync/async calls
├── knowledge    // KB storage, prompt builder (Ph 3), search tool (later)
├── ai           // OpenAI adapter
├── recording    // download → MinIO, retention
├── jobs         // Postgres job table + workers
├── events       // outbox + signed dispatcher to client apps
├── accounts     // client apps, accounts, numbers, agents
└── common
```

---

3. Call paths

3.1 Inbound ? menu ? waiting ? browser agent



```
(via Textify)
 |      |                   | claim (Redis SET NX) → room token ──────────────────────►|
joins room
 |      |── POST /wait/{id} ►| accepted → Dial sip:{callId}@… record |                  |
"Connecting…"
 |      |══ SIP INVITE ═══════════════════════════════════════► room call-{id} ◄══|
 |◄═══════╪═══════════════════ live audio ═══════════════════════╪═══════════════════|
 |hangup|── POST /voice (isActive=0: duration, amount, recordingUrl) ►| close → events, jobs |
```

- Agents accept while the caller is mid-segment; connection happens at the next callback (segments 10–15 s).
- `maxWaitSec` reached / key pressed → outcome (voicemail etc.) on AT. Unanswered → `call.missed`.
- Agent hold/transfer inside LiveKit: engine starts Ingress hold audio into the room.
- **[VERIFY]** wait-loop behaviour (FLOWS §6); fallback = Dial immediately, hold music via Ingress.

3.2 Outbound click-to-call

```
Agent → Textify → POST /v1/calls/outbound {accountId, fromNumberId, to, agentRef}
Engine: can-call → create call + room call-{callId} → agent token (agent joins, hears ringback
in page)
        → AT call API (from, to) → store AT sessionId
Customer answers → AT callback (Outbound, sessionId) → XML Dial sip:{callId}@… → customer
joins → talk
End → final callback → call.ended
```

3.3 Campaign

```
Textify → POST /v1/campaigns (+ recipients in chunks) → POST /start
Engine:
  1. validate flow (pinned version), opt-outs, variables vs recipient columns
  2. PRE-GENERATE voice: render each recipient's text, dedupe by hash, TtsRouter → files
(jobs)
     cost estimate (premium chars) available before start; failure → campaign PAUSED
(TTS_FAILED)
  3. Dispatcher (@Scheduled 1 s): window? not paused? active < maxConcurrent?
       SELECT … campaign_recipients WHERE status='PENDING' AND next_attempt_at<=now()
       ORDER BY id LIMIT :cps FOR UPDATE SKIP LOCKED → AT call API → DIALING
```

```
  4. Answer → AT callback → run campaign flow (FLOWS §5.2) with the recipient's variables
  5. Final callback → ANSWERED / NO_ANSWER / BUSY / FAILED; retries per policy
Events: campaign.progress (30 s) · campaign.recipient_results (batches) · campaign.paused · campaign.completed
Textify: SMS fallback to NO_ANSWER; pause on low wallet
```

3.4 After the call (jobs)

```
final callback recordingUrl → DOWNLOAD → MinIO voice-recordings → recording.ready
  → (Ph 2) TRANSCRIBE → SUMMARIZE → summary.ready
voicemail → voicemail.received (+ recording)
daily RETENTION → delete expired recordings
```

Recording comes from **AT** (`Dial record="true"`, `Record`). No LiveKit egress needed. **[VERIFY]** AT records Dial-to-SIP; fallback egress.

3.5 Phase 3 AI

`ai` node → `Dial` into LiveKit → engine dials OpenAI Realtime into the room over SIP → accepts with instructions built from the **knowledge base** + node instructions + tools → controls via WebSocket from Java → `handoff` rings a department (agent joins room, AI leg hung up) → summary job. Appendix A.

3.6 Call state machine

```
INBOUND:  RINGING → IN_FLOW → WAITING_AGENT → CONNECTING → ANSWERED → COMPLETED
OUTBOUND: DIALING → IN_FLOW (campaign) / CONNECTING (click-to-call) → ANSWERED → COMPLETED
any → MISSED | NO_ANSWER | BUSY | FAILED | REJECTED_NO_BALANCE
handled_by: FLOW_ONLY | VOICEMAIL | AGENT | AI | AI_THEN_AGENT
```

---

4. Browser agents

- **No browser extension or app install needed.** The agent UI is a normal page in Textify's web app using the LiveKit JS SDK; browsers have WebRTC built in (Chrome, Edge, Firefox, Safari).
  - Requires: HTTPS, microphone permission, the Textify tab open (can be in the background).

    - Browsers block sound until the user interacts with the page, so agents click **"Start shift / Go available"** once; that click unlocks the ringtone.
    - Ring alerts when the tab is in the background: Web Notifications API + ringtone; optional **PWA** (installable web app) for a desktop-app feel. A Chrome extension or native/mobile app is **not** required; consider only later (e.g. ringing with the browser closed).
- **Lobby room** per agent `lobby-{accountId}-{agentRef}` (mic off) while available:
    - presence = lobby join/leave webhooks + explicit status;
    - ringing = `sendData(lobby, {type:"ring", callId, from, numberLabel, department})`.
- Accept → browser → **Textify backend** → `POST /v1/calls/{id}/accept {agentRef}` → claim → call-room token → join (shows "Connecting caller…" until the caller arrives). Losers: `409` + `ring_cancel`.
- **Screen pop**: the browser uses `from` to ask Textify for contact + open tickets. The engine never sees them.
- **Contacts**: Textify's **existing contact module** (the one SMS/WhatsApp already use) is reused for the call center: campaign recipients, caller names, customer timeline. The engine stores no contacts; campaigns receive `{e164, vars}` only.
- In-call: mute (client-side), hold (engine starts Ingress music, mutes agent), transfer (engine re-rings target; caller stays in room with hold music), hang up.

---

5. Media library & voice generation

5.1 Uploaded audio

```
Textify (file or browser recording) → POST /v1/accounts/{id}/media (multipart)
Engine: validate (≤ 10 MB, ≤ 5 min, magic bytes) → MinIO original/ → job TRANSCODE (ffmpeg:
        mono, 16 kHz, MP3, loudness-normalized, trimmed) → MinIO play/ → READY → media.ready
```

5.2 Text ? voice (generated audio)

- Messages with `source: text` are generated **at flow save** (inbound) or **before dialing** (campaigns, per-recipient text).
- Pipeline: render variables → **Swahili normalizer** (numbers, currency, dates, times → words) → cache lookup `hash(text+lang+voice+tier)` → `TtsRouter.tier → adapter` → ffmpeg normalize → MinIO `generated/` → `media_assets` row (kind `GENERATED`).
- **Basic**: self-hosted light model in a small sidecar (`tts-basic`, CPU), e.g. Piper with a Swahili voice, **only if** quality + commercial licence pass the listening test. **[VERIFY licence]**
- **Premium**: OpenAI TTS or Azure Speech (Swahili neural voices), picked by listening test.

- **Costs predictable**: premium character count computed before generation → shown in Textify → reported (`tts.usage`).
- **No runtime fallback to AT `Say`.** Failure → flow save error (previous version stays live) / campaign paused. Call-time safety net: system file.
- On-demand generation exists only for **sync webhook responses** (Ph 2): `Play url=/media/tts/{hash}.mp3` generated on first fetch, then cached. **[VERIFY]** AT `Play` fetch timeout.

5.3 Playing to AT

- XML uses `<Play url="https://voice-api…/media/{id}.mp3?t={hmac,24h}">`; the engine streams from MinIO (cache headers). Buckets stay private.
- Same URLs feed LiveKit Ingress (agent hold).
- Deleting media used by an active flow / running campaign → 409. Per-account storage quota.

5.4 Buckets

```
voice-media/system/{lang}/{name}.mp3                # notices, hold default, sorry, invalid
option, numbers 1–20
voice-media/{accountId}/original/{mediaId}.{ext}
voice-media/{accountId}/play/{mediaId}.mp3
voice-media/generated/{hash}.mp3                     # shared cache for generated voice
voice-recordings/{accountId}/{yyyy}/{mm}/{callId}.mp3 # retention applies
```

---

6. Knowledge base (Phase 3)

- Stored **in the engine per account** (business facts for call handling; not user data). Edited in Textify, pushed with `PUT /v1/accounts/{id}/knowledge`.
- Structure:

```
{ "profile": { "name": "ABC Ltd", "about": "...", "location": "...", "hours": "..."
},
  "services": [ { "name": "Kifurushi cha mwezi", "price": "TZS 25,000", "notes":
"..." } ],
  "faqs": [ { "q": "...", "a": "..." } ],
  "departments": [ { "id": "dep_sales", "description": "New orders and prices" } ],
  "rules": { "never": ["Promise refunds"], "handoverWhen": ["Caller asks for a
person", "Complaint"] },
```

```
   "documents": [ "kb_doc_1" ] }
```

(`Kifurushi cha mwezi` = "Monthly package".)

- v1 of AI: **prompt builder** puts the whole KB into the AI session instructions (small businesses fit easily).
- Large KB: chunk documents into Postgres (`kb_chunks`, full-text or pgvector) and give the AI a `search_knowledge` tool. Still just Postgres.
- Live data via the account's **endpoints** exposed as AI tools.
- Improvement loop: the summary job extracts unanswered questions → `kb.unanswered` event → Textify suggests FAQ additions.

---

7. Data model (PostgreSQL)

```
client_apps (id uuid pk, name, status,                    -- ACTIVE | SUSPENDED | REMOVED
             webhook_url, webhook_secret_hash, can_call_url, rate_limit_rps, features jsonb,
             suspended_reason null, created_at, removed_at null)
client_app_keys (id uuid pk, client_app_id fk, key_hash, prefix, created_at, last_used_at,
                 last_used_ip, revoked_at null)                -- ≥1 active; 2 during
rotation
audit_log (id bigserial pk, actor, action, target, data jsonb, ip, at)

accounts (id uuid pk, client_app_id fk, external_ref, status, default_language default 'sw',
          retention_days default 90, features jsonb, created_at, unique(client_app_id,
external_ref))

phone_numbers (id uuid pk, e164 unique, provider default 'AT', type, status,
               client_app_id fk null,                         -- pool owner (null = global
pool)
               account_id fk null,
               label, hours jsonb, current_flow_version_id fk null, updated_at)

flows (id uuid pk, account_id fk, kind,                   -- NUMBER | CAMPAIGN
       owner_id uuid, created_at)
flow_versions (id uuid pk, flow_id fk, version int, definition jsonb, created_at,
               premium_chars int default 0, unique(flow_id, version))

media_assets (id uuid pk, account_id fk null,             -- null = system / shared
generated
              kind,                                       -- UPLOAD | GENERATED | SYSTEM
```

```
              name, original_key null, play_key, text_hash null, tier null, lang, voice,
              mime, size_bytes, duration_sec, status, error null, created_at)

agents (id uuid pk, account_id fk, external_ref, display_name, status, last_seen_at,
        unique(account_id, external_ref))
departments (id uuid pk, account_id fk, name, strategy, wrap_up_sec default 0)
department_members (department_id fk, agent_id fk, position, pk(department_id, agent_id))

endpoints (id uuid pk, account_id fk, name, url, secret, active bool, created_at)

calls (id uuid pk, account_id fk, number_id fk, campaign_id fk null, recipient_id fk null,
       flow_version_id fk, direction, from_e164, to_e164, at_session_id, livekit_room,
       status, handled_by, agent_id null, department_id null, context jsonb,
       started_at, answered_at, ended_at, end_reason,
       pstn_seconds, trunk_seconds, ai_seconds, wait_seconds, at_amount numeric, at_currency,
       ai_usage jsonb null)
call_events (id bigserial pk, call_id fk, type, node_id null, at, data jsonb)   -- trail

campaigns (id uuid pk, account_id fk, from_number_id fk, flow_version_id fk, status,
           window jsonb, cps int, max_concurrent int, max_retries int, retry_delay_sec int,
           starts_at, premium_chars_estimate int, created_at, completed_at, pause_reason null)
campaign_recipients (id bigserial pk, campaign_id fk, e164, vars jsonb, media_id fk null,
                     status, attempts, next_attempt_at, last_call_id fk null,
answered_seconds,
                     keys text, index(campaign_id, status, next_attempt_at))
opt_outs (account_id fk, e164, source, created_at, pk(account_id, e164))

knowledge_bases (account_id pk fk, data jsonb, updated_at)                      -- Ph 3
kb_chunks (id bigserial pk, account_id fk, doc_id, text, tsv tsvector, embedding vector
null)  -- later

recordings (id uuid pk, call_id fk unique, kind,                                -- CALL |
VOICEMAIL
            source_url, storage_key, duration_sec, size_bytes, status, delete_after)
transcripts (call_id pk fk, status, language, text, segments jsonb, summary jsonb, model,
created_at)

jobs (id bigserial pk, type, ref_id, status, attempts, run_after, last_error)
      -- TRANSCODE | TTS | DOWNLOAD | TRANSCRIBE | SUMMARIZE | RETENTION
outbox_events (id uuid pk, client_app_id fk, type, payload jsonb, attempts, next_attempt_at,
```

```
                delivered_at null, last_error)
endpoint_deliveries (id uuid pk, endpoint_id fk, call_id fk, node_id, payload jsonb, mode,
                     attempts, next_attempt_at, status, response_code null, last_error)
```

Redis: `presence:{accountId}:{agentRef}`, `call:{id}:agent` (claim, SET NX), `wait:{departmentId}` (sorted set: callId by wait start), `rr:{departmentId}`, `campaign:{id}:active`, flow/number config cache invalidation channel.

---

8. Security (service authentication)

Traffic	Method
Textify → engine	API key per client app (hashed; 2 active keys for rotation; scoped to its accounts; rate-limited)
Engine → Textify	HMAC-SHA256 signed webhooks + timestamp (reject > 5 min)
Engine → customer endpoints	HMAC with the endpoint's secret, idempotency key, HTTPS only, **SSRF guard** (block private/loopback/link-local/metadata IPs after DNS resolution), 3 s timeout
AT → engine	Unguessable secret in path + AT IP allowlist + sanity checks (known number / call) **[VERIFY AT IPs, any signature]**
LiveKit → engine	Signed JWT (`WebhookReceiver`)
OpenAI → engine (Ph 3)	Webhook signature + dedupe
AT → LiveKit SIP	Firewall + inbound trunk `allowed_addresses` = AT IPs; unknown callId → reject
Browser → LiveKit	Short-lived room tokens minted by engine, requested **through Textify** (human auth happens there)
AT → audio URLs	HMAC-signed, expiring media URLs

No OAuth/Keycloak/mTLS between our services. Secrets in env only. Recordings in private buckets with signed URLs, retention job, recording notice. Keep CrowdSec/fail2ban.

8.1 Client apps: how a service gets onto the engine, uses it, and is stopped or removed

Any service that consumes the engine (Textify first; later JikoXpress, Mauzodukani, NexGate…) is a **client app**. The engine treats them all the same and keeps them fully isolated from each other.

Who registers a service: only the **engine owner (QBIT SPARK ops)**. There's no self-signup, and client apps can never create, see or manage other client apps. Textify is simply the first client app registered.

How: CLI-first (`voice-admin`), no public ops API

- The admin tool is a mode of the engine jar (`java -jar voice-engine.jar admin …`, wrapped as `voice-admin`), run **on the Voice box over SSH**. It talks directly to the engine database, so **no ops endpoint exists on the internet**.
- SSH: key-only login, root login disabled, port 22 open only to the owner's IPs (ufw), CrowdSec on.
- Optional later (web ops panel): the `/ops/*` API binds **only to localhost or a WireGuard VPN interface**, never the public domain, and requires an **ops token** stored only on the box (env), plus the admin IP allowlist.

```
$ voice-admin client-app create --name textify \
    --webhook-url https://portal.textify.africa/voice/events \
    --can-call-url https://portal.textify.africa/voice/can-call \
    --features inbound,outbound,campaigns --rate-limit 50
→ clientAppId:    ca_7f3…
→ API key:        vk_live_3f9a…   (shown ONCE)
→ Webhook secret: whsec_81c2…     (shown ONCE)

$ voice-admin numbers allocate +2557XXXXXXXX --client ca_7f3…
$ voice-admin client-app test-webhook ca_7f3…
$ voice-admin key rotate ca_7f3…          # new key; old one stays valid until revoked
$ voice-admin key revoke ca_7f3… key_12…
$ voice-admin client-app suspend ca_7f3… --reason "…" [--hard]
$ voice-admin client-app resume ca_7f3…
$ voice-admin client-app export ca_7f3…  # zip of calls, recordings index, flows
$ voice-admin client-app remove ca_7f3…  # grace period, then delete
$ voice-admin audit --client ca_7f3… --since 7d
```

Security throughout

Stage	Protection
Who can register	Only someone with SSH access to the Voice box (owner's key + IP)
Creating credentials	Random 256-bit secrets generated by the engine; **only hashes stored**; shown **once** in the terminal
Hand-over	Owner pastes them **directly into the consuming service's env / secret store** on its own server. Never email, chat, tickets or git

Stage	Protection
Identifiable keys	Prefix `vk_live_` / `vk_test_` + short key id, so a leaked key is easy to spot and revoke; full key never logged
Using the engine	HTTPS only; key → exactly one client app; all data filtered by `client_app_id`; per-client rate limit + feature flags
Webhooks back	Signed with that client's webhook secret; client verifies signature + timestamp
Leak / staff change	`key rotate` (both valid briefly) → `key revoke`: effective immediately
Stop / remove	`suspend` (reversible) / `remove` (export → release numbers → revoke keys → delete after grace)
Traceability	Every admin command (who, when, from which IP) and each key's first/last use + IP in `audit_log`

No approval steps: protection comes from **who can reach the CLI** plus **show-once, hashed, scoped, rotatable keys**.

Lifecycle

Step	What happens	How
1. Register	Create the client app: name, `webhook_url`, `can_call_url`, rate limit, allowed features (inbound, campaigns, AI)	`voice-admin client-app create …` → returns `clientAppId`, **API key** and **webhook secret**, shown **once** (only hashes stored)
2. Give it numbers	Assign AT numbers from the global pool to the client app's pool	`voice-admin numbers allocate <e164> --client <id>`
3. Client stores credentials	The client keeps the API key + webhook secret in its own secrets/env	Client side
4. Verify	Client calls `GET /v1/ping` (auth check); engine sends a signed `test.ping` webhook to check the client's receiver	`voice-admin client-app test-webhook <id>`
5. Consume	Client creates its accounts (`POST /v1/accounts {externalRef}`), assigns numbers from **its** pool to accounts, saves flows, adds agents, runs campaigns…	`/v1/**` with `Authorization: Bearer <key>`
6. Rotate key	Issue a second key; both work; client switches; old key revoked	`voice-admin key rotate <id>` → `voice-admin key revoke <id> <keyId>`

Step	What happens	How
7. Suspend (stop, reversible)	API returns `403 CLIENT_SUSPENDED`; running campaigns **paused**; no new outbound calls; webhooks held in the outbox (delivered after resume). **Inbound calls keep working by default**, so callers aren't hurt. Option `hard: true` also answers inbound with a system "service unavailable" message	`voice-admin client-app suspend <id> --reason … [--hard]` · `… resume <id>`
8. Remove (permanent)	Suspend → cancel campaigns → release numbers back to the global pool (AT callback config untouched; numbers reusable) → revoke keys → stop webhooks → optional **export** (calls, recordings, flows as a zip) → data deleted after a grace period (default 30 days)	`voice-admin client-app export <id>` · `… remove <id>`

Isolation rules

- Every `/v1` request resolves to exactly one client app from the API key; every query is filtered by `client_app_id`. A key can never see another client app's accounts, numbers, calls or media.
- `external_ref` is unique **per client app** (two apps can both have "tenant_1").
- Numbers belong to one client app's pool at a time; inbound calls are routed by number → client app → account.
- Each client app has its own webhook URL + secret, rate limit (e.g. 50 req/s) and feature flags.
- An account inside a client app can also be suspended (`PATCH /v1/accounts/{id} {status: SUSPENDED}`): same effect as above, but for one tenant. Textify uses this when a customer stops paying.

Audit: every admin command and key use (first/last seen, IP) is logged in `audit_log`; `voice-admin audit` reads it.

---

9. Communication

Need	Mechanism
Textify ⇄ engine commands/queries	REST
Engine → Textify facts	Webhooks via Postgres **outbox** + `@Scheduled` dispatcher
Engine → customer systems	`webhook` node: async via `endpoint_deliveries` (retries), sync direct (Ph 2)
AT call control	Callbacks + XML; AT REST call API

Need	Mechanism
Ring agents	LiveKit data messages (lobby rooms)
Presence, claims, waiting orders, concurrency, cache invalidation	Redis
Background work	Postgres `jobs` + workers (`FOR UPDATE SKIP LOCKED`)
Broadcast queue	`campaign_recipients` table + dispatcher

Not used: MQTT, Kafka, RabbitMQ. Add **Redis Streams** first only if several engine instances + high event volume demand it.

---

10. APIs

10.1 Client API (Textify ? engine) · `Authorization: Bearer <api key>` · base `https://voice-api.textify.africa/v1` **[VERIFY domain]**

Area	Endpoints
Accounts	`POST /accounts` · `PATCH /accounts/{id}`
Numbers	`GET /numbers?status=UNASSIGNED` · `POST /numbers/{id}/assign` · `POST /numbers/{id}/release` · `PUT /numbers/{id}/hours` · `PUT /numbers/{id}/flow` (validate + new version) · `GET /numbers/{id}/flow/versions` · `POST /numbers/{id}/flow/rollback {version}` · `POST /numbers/{id}/test-call {toE164}`
Flow tools	`POST /flows/validate` (dry run, returns errors + premium char estimate)
Media	`POST /accounts/{id}/media` (multipart) · `POST /accounts/{id}/media/generate {text, lang, voice, tier}` (preview) · `GET /accounts/{id}/media` · `GET /media/{id}` · `DELETE /media/{id}` · `GET /voices`
Agents	`PUT /accounts/{id}/agents/{ref}` · `DELETE …` · `PUT …/status` · `POST …/lobby-token`
Departments	`POST/PUT/DELETE /accounts/{id}/departments[/{did}]`
Calls	`POST /calls/outbound` · `POST /calls/{id}/accept` · `POST /calls/{id}/hold

Area	Endpoints
Campaigns	`POST /campaigns` · `POST /campaigns/{id}/recipients` (chunks ≤ 10k) · `POST /campaigns/{id}/estimate` · `POST /campaigns/{id}/start
Opt-outs	`GET/POST/DELETE /accounts/{id}/opt-outs`
Endpoints	`POST/PUT/DELETE /accounts/{id}/endpoints[/{eid}]` · `POST /endpoints/{eid}/test`
Knowledge (Ph 3)	`PUT /accounts/{id}/knowledge` · `POST /accounts/{id}/knowledge/documents` · `GET /accounts/{id}/knowledge/unanswered`
Stats	`GET /numbers/{id}/stats` (per node) · `GET /campaigns/{id}/stats`
Health	`GET /ping` (auth check)
Ops: **CLI** `voice-admin` **on the box** (primary). Optional `/ops` HTTP API bound to localhost/VPN only, ops token + admin IPs	`POST /ops/client-apps` · `POST /ops/client-apps/{id}/keys` · `DELETE /ops/client-apps/{id}/keys/{keyId}` · `POST /ops/client-apps/{id}/test-webhook` · `POST /ops/client-apps/{id}/suspend

10.2 Engine ? Textify

- Balance: `GET {can_call_url}?accountRef&direction&estimatedCalls?` → `{allowed, maxMinutes?}` (1.5 s, fail-open).
- Webhooks (outbox, signed, retried 1m/5m/30m/2h… ≤ 24 h, idempotent by event id):

Event	Data
`call.started` · `call.answered`	callId, accountRef, direction, from, to, number, handledBy, agentRef, department
`call.ended`	status, usage {pstnSeconds, trunkSeconds, waitSeconds, aiSeconds, atAmount}, recorded, endReason, flowVersion
`call.missed`	callId, from, to, numberLabel, department
`call.action_triggered`	callId, node, endpoint, status
`message.requested`	callId/campaignId, to, channel, template, params → Textify sends SMS/WhatsApp
`voicemail.received`	callId, from, mediaId/recording
`recording.ready` · `summary.ready`	callId (+ summary JSON)
`media.ready` · `media.failed`	mediaId, duration / error
`tts.usage`	accountRef, chars, tier, ref (flow version / campaign)
`campaign.progress` · `campaign.recipient_results` · `campaign.paused` · `campaign.completed`	counts / batches / reason / totals
`kb.unanswered` (Ph 3)	accountRef, questions[]

Event	Data
`test.ping`	sent on request to verify the client's webhook receiver

10.3 Hooks into the engine

Endpoint	From
`POST /hooks/at/{secret}/voice` · `/step/{callId}/{node}` · `/wait/{callId}`	AT
`POST /hooks/livekit`	LiveKit
`POST /hooks/openai` (Ph 3)	OpenAI
`GET /media/{id}.mp3?t=…` · `GET /media/tts/{hash}.mp3?t=…`	AT / LiveKit Ingress

---

11. Infrastructure

11.1 Dedicated Voice box (Docker Compose; host networking for media)

Service	Notes
`caddy` / nginx	TLS for `voice-api.…` (API, hooks, media URLs) and `wss://voice-rtc.…` (LiveKit)
`livekit`	`livekit/livekit-server`, 7880 on localhost
`sip`	`livekit/sip`, 5060 + RTP 10000–20000, AT only
`ingress`	`livekit/ingress`, agent-hold audio **[VERIFY URL input]**
`redis`	shared by LiveKit + engine (separate DB index)
`postgres`	engine DB (pgvector extension later for KB)
`minio`	private buckets `voice-media`, `voice-recordings`; console on localhost
`tts-basic`	self-hosted light TTS model (CPU), HTTP on localhost **(if Basic passes the test)**
`voice-engine`	Spring Boot jar + ffmpeg
(fallback) `egress`	only if AT recording isn't sufficient

Start 4 vCPU / 8 GB, measure. Region with lowest RTT to AT SIP servers.

11.2 Ports

Port	Open to	Purpose
22	owner IPs only (key-only login)	SSH, `voice-admin` CLI
80/443	world	API, hooks, media, WSS
7881/tcp, 50000–60000/udp	world	WebRTC (browser agents)
5060 udp/tcp, 10000–20000/udp	AT IPs only	SIP + RTP
7880, 6379, 5432, 9000/9001, tts-basic, 8080	localhost	internal
(Ph 3) out 5061 + RTP to OpenAI CIDRs	OpenAI	AI over SIP

11.3 Env

```
AT_USERNAME / AT_API_KEY / AT_CALLBACK_SECRET / AT_IPS
LIVEKIT_URL / LIVEKIT_PUBLIC_WS_URL / LIVEKIT_API_KEY / LIVEKIT_API_SECRET / LIVEKIT_SIP_HOST
DB_URL / DB_USER / DB_PASSWORD · REDIS_URL / REDIS_PASSWORD
MINIO_ENDPOINT / MINIO_ACCESS_KEY / MINIO_SECRET_KEY
MEDIA_URL_SECRET
TTS_BASIC_URL · TTS_PREMIUM_PROVIDER (openai|azure) / keys
OPENAI_API_KEY (Ph 2) · OPENAI_WEBHOOK_SECRET / OPENAI_SIP_URI (Ph 3)
```

11.4 LiveKit bootstrap (once, idempotent)

1. Inbound trunk `at`: `allowed_addresses` = AT IPs, no number filter.
2. Dispatch rule `dispatch_rule_callee`, `room_prefix: "call-"` → `sip:{callId}@…` ⇒ room `call-{callId}` **[VERIFY]**.
3. (Ph 3) Outbound trunk `openai` (TLS + SRTP).

---

12. Observability & testing

- JSON logs with `callId`, `accountId`, `atSessionId`, `campaignId`, `node`.
- Metrics: active calls, waiting callers + wait time, ring-to-accept, answer/missed rate, callback latency (p95 < 500 ms), campaign dials/sec + answer rate, TTS jobs + premium chars, endpoint delivery failures, outbox backlog.
- Alerts: SIP down, callback errors, zero inbound in business hours, outbox backlog, campaign stalled, TTS failures, system-file fallback used.

- PoC = SPEC §10 Phase 0.
- Unit: flow validator, every NodeExecutor (node → XML), wait loop, claim race, ring strategies, campaign dispatcher (window/cps/retries/opt-out), Swahili normalizer, signers, SSRF guard.
- Integration: Testcontainers (Postgres, Redis, MinIO), WireMock (AT, Textify, endpoints, TTS, OpenAI), LiveKit dev + browser test page, real AT number.

13. Tech stack & tools

Layer	Tool	Used for
Engine	Java 21, Spring Boot 3.x (Web, Validation, JDBC/JPA, Scheduling, Actuator)	All engine logic
DB	PostgreSQL 16 + Flyway (pgvector later)	State, flows, outbox, jobs, campaign queue, KB
Cache/locks	Redis 7	Presence, claims, waiting orders, concurrency
Telephony	Africa's Talking Voice (callbacks + XML: Play, GetDigits, Record, Dial; call API); optional `africastalking-java` for REST	PSTN both directions
Media	LiveKit server, SIP, Ingress; `io.livekit:livekit-server` JVM SDK; LiveKit JS SDK in Textify UI	Rooms, browser agents
Voice (TTS)	Basic: self-hosted light model (e.g. Piper, CPU) **[VERIFY licence/quality]**; Premium: OpenAI TTS or Azure Speech	Text → audio files
AI	OpenAI REST (transcribe, summarize); Ph 3 Realtime over SIP + `java.net.http.WebSocket`	Summaries, live AI
Storage	MinIO via AWS SDK v2	Media library, generated voice, recordings
Audio processing	ffmpeg	Transcode, normalize, trim
HTTP	Spring `RestClient`	AT, Textify, endpoints, TTS, OpenAI
Security	Hashed API keys, HMAC-SHA256, LiveKit `WebhookReceiver`, SSRF guard, ufw, CrowdSec	Service auth
Edge	Caddy (or nginx)	TLS, reverse proxy
Deploy	Docker Compose, dedicated VPS	Everything

Layer	Tool	Used for
Observability	Logback JSON, Actuator + Micrometer (Prometheus)	Logs, metrics
Testing	JUnit 5, Testcontainers, WireMock, browser test page	Unit + integration
Not used	AT `Say`, MQTT, Kafka, RabbitMQ, OAuth/Keycloak, Python, browser extension	Not needed

14. Repository layout

```
textify-voice/
├── docs/          SPEC.md, FLOWS.md, ARCHITECTURE.md
├── voice-engine/  Spring Boot (Java 21)
├── tts-basic/     self-hosted TTS sidecar (if Basic passes)
├── deploy/        docker-compose.yaml, livekit.yaml, sip, ingress, Caddyfile, firewall.sh,
.env.example
├── audio/         system prompts (sw/en): notices, hold default, sorry, invalid option,
numbers
└── agent-test/    minimal HTML page with LiveKit JS for the PoC
```

15. Open items to verify

1. AT: Dial to `sip:{callId}@host`; IPs; codecs/DTMF; billing; Dial-to-SIP recording; CPS/concurrency; per-second vs per-minute; multi-number call API; callback signature (SPEC §11).
2. AT: GetDigits wrapping Play (key interrupt, timeouts, callbacks per call) for the wait loop; Play fetch timeout/streaming; XML continuation after Dial.
3. LiveKit: callee dispatch with arbitrary user part; trunk without number filter; Ingress URL input; JVM SDK method names.
4. TTS: Basic candidate quality + commercial licence; Premium choice; Swahili normalizer coverage.
5. Ph 3: LiveKit → OpenAI SIP interop; Realtime vs Live API; Swahili AI quality + cost.

Appendix A: Phase 3 AI (Java-only)

- No Java SDK joins LiveKit rooms as an audio participant; Spring AI / LangChain4j are text-only. So **OpenAI joins the room as a SIP participant**.

- Engine: `CreateSIPParticipant(outbound trunk "openai", to = OpenAI project SIP URI, header X-Voice-Call-Id)` → OpenAI webhook `realtime.call.incoming` (or Live API `live.transport.incoming`) → match header to a call in `AI_DIALING` → `POST /v1/realtime/calls/{id}/accept` with instructions (KB prompt + node instructions; Swahili first; short answers; never invent prices), voice, tools (`transfer_to_human`, `take_message`, `end_call`, endpoint tools, `send_sms`) → WebSocket `wss://api.openai.com/v1/realtime?call_id=…` for tool calls/transcripts/usage → handoff: ring department, agent joins, `POST …/hangup` AI leg.
- Firewall: out TCP 5061 to `sip.api.openai.com`; RTP with OpenAI media CIDRs `13.79.45.80/28, 23.98.140.64/28, 40.67.149.176/28, 40.83.204.240/28` (OpenAI SIP guide, Oct 2026; re-check).
- Trade-off: OpenAI voices only. If Swahili isn't good enough → fallback: small Python LiveKit Agents worker with another TTS, calling engine internal endpoints; nothing else changes.

Revision #5
Created 21 October 2026 13:04:28 by Admin Qbit
Updated 13 November 2026 19:06:38 by Admin Qbit