What the app looks like — user-facing (visual & UX)

Primary user channel: WhatsApp chat (no new app to learn)

- Clean, conversational flow inside WhatsApp.
- Quick-reply buttons after greeting: **Report Fraud** • **Track Case** • **Awareness Tips** • **Speak to Agent**.
- Multilingual exchange: automatic **English** ⇄ **Odia** detection with readable Odia script (and romanized fallback).
- Guided, form-like chat for reporting: the bot asks for incident type → date → amount → platform → attachment(s) → confirms and returns a **Reference ID**.
- Helpful templated messages: confirmation, status updates, safety tips, escalation acknowledgement.
- Optional rich content: images (screenshots), PDFs, mini-forms (via message templates).

Admin UI (web dashboard) — React + Vite:

- Login-protected admin console.
- Top-level: KPI cards (complaints/day, avg resolution time, language split).
- Complaint table: filter/sort by district, scam type, status, language.

- Case detail panel: full transcript, attachments, audit logs, assign/transfer buttons.
- Analytics view: trend charts, heatmap (by district), top scam patterns — exportable CSV/PDF.
- Campaigns area: schedule awareness broadcasts, quick quiz builder.

Final outcome (what the system delivers)

- **End-to-end reporting pipeline**: User messages WhatsApp bot complaint recorded optionally forwarded to cybercrime.gov.in reference ID returned.
- **Case tracking**: Citizens query reference ID and receive updated status via WhatsApp (automatic pushes on status change).
- **Escalation**: Critical cases routed to 1930 backend / call centre (ticket + callback).
- **Awareness program**: Targeted tips, alerts and interactive quizzes to reduce recurrence.
- **Operational insights**: Authorities receive dashboards and data exports to identify hotspots and repeat campaigns.
- **Privacy-first operations**: PII minimization, field-level encryption, RBAC for officials, audit logs, configurable retention.

User journeys (short)

1. **Report Fraud (user)**

   - User: “I paid money but never received product.”
   - Bot: asks incident type → date → amount → asks for screenshot → confirms consent → registers complaint → replies: Reference ID: CS-3f1a2b.

2. **Track Case (user)**

   - User: “Track CS-3f1a2b”
   - Bot: validates, fetches local + portal status, replies: Forwarded to district cyber cell — Investigator: Ajay (xxxxxxx).

3. **Escalate (user)**

   - User: “This is urgent”
   - Bot: creates escalate ticket to 1930, schedules callback, notifies user when callback is done.

4. **Admin flow**

   - Officer sees new complaint in queue → assigns to district cell → updates status → system pushes update to user automatically.

Backend — what it does (components & responsibilities)

Stack used in repo: FastAPI (service layer), Rasa (NLU & dialog), SQLAlchemy/Postgres (DB), adapters for WhatsApp & CyberPortal, job workers & infra (Docker, Helm, Terraform).

Core functions:

- **Message ingress & routing**
  - whatsapp_adapter: accepts webhook messages from Meta/Twilio, formats and forwards text to nlp_service (or to Rasa).
- **NLP & Dialog**
  - rasa handles intent classification, slot filling, forms (report flow) and triggers custom actions.
  - nlp_service offers fallback parsing (and uses Rasa HTTP API).
- **Complaint lifecycle**
  - db_service persists complaints (reference IDs, attachments, audit trail).
  - cyberportal_adapter submits to cybercrime.gov.in (mock adapter if API unavailable) and fetches status.

- escalation routes urgent tickets to 1930 backend.

- **Security & privacy**
  - security module: JWT auth, token creation/validation, (stubs for) field-level encryption and RBAC enforcement.
  - Audit logs for every read/write.

- **Notifications & webhooks**
  - On status change, backend sends templated WhatsApp messages via whatsapp_adapter.

- **Integration & infra**
  - Queueing (design): message queue (Kafka/RabbitMQ) for decoupling and retries.
  - Observability: Prometheus metrics + Grafana dashboards; Sentry-style error tracking.
  - CI/CD (GitHub Actions), containerized images, Helm charts for K8s, Terraform stubs for cloud infra.

Frontend — what it does (components & responsibilities)

React + Vite + Tailwind (or simple CSS in repo)

Key UI modules:

- **Dashboard** — overview metrics, recent trends.
- **ComplaintTable** — paginated list with search and filters.
- **ComplaintDetail** — full transcript (chat + attachments), case status, portal ID, escalation buttons, notes.
- **Analytics** — charts for scam types, language distribution, district heatmap.
- **Campaigns** — schedule tips/alerts, configure target area (district/language).
- **Auth & RBAC** — login for admins; roles determine access (viewer / investigator / admin).
- **API client** — Axios wrapper calling backend endpoints (complaints, tracking, escalate, analytics).

UX priorities:

- Minimal clicks to assign/acknowledge cases
- Audit trail visible on each case

- Bulk actions (export, assign, schedule alerts)

Data & APIs (important endpoints)

- POST /api/v1/complaints — register complaint
- GET /api/v1/complaints/list — admin list
- GET /api/v1/complaints/{ref} — fetch complaint
- GET /api/v1/tracking/{ref} — user case tracking
- POST /api/v1/escalation/to-1930 — escalate
- Webhook: POST /webhook/whatsapp — provider -> app
- Worker endpoints/webhooks for portal status updates

Non-functional design decisions

- **Scalability:** stateless FastAPI workers behind LB, horizontal autoscaling, workers consume queues for slow tasks (attachments, portal sync).

- **Resilience:** retries with backoff, DLQ for failed submissions.
- **Privacy:** store minimal PII, configurable retention/purge, encryption keys in Vault/KMS.
- **Localization:** training datasets include Odia script + transliterations; allow human-in-loop for low-confidence messages.
- **Monitoring & SLOs:** message latency < 2s (bot response), NLP confidence thresholds with human fallback.

Deployment & final package

- **Local dev:** Docker Compose starts Postgres, backend, rasa, frontend; scripts to migrate DB and demo a complaint.
- **Prod:** Docker images push to registry Helm deploy to Kubernetes (EKS/GKE/AKS) with Terraform-managed infra; secrets in Vault/KMS.
- **Deliverable:** CyberSathi-v1.zip containing repos, infra manifests, trained models (or training pipeline), CI, docs, and runbook.