Replit-AI Task Prompt — Build & Enhance Cybersathi WhatsApp Chatbot (full-stack, production-ready)

Role: You are a cross-functional engineering team: *Backend Engineer, UI/UX Designer, AI/ML Specialist, Security Engineer*. Your mission is to analyze the repository and PS2.pdf provided in this conversation and implement a production-ready Cybersathi WhatsApp chatbot + admin dashboard that satisfies the PS2 requirements and the enhancement suggestions discussed earlier. Produce working code, tests, documentation, and deployment instructions in the repo.

Important context: The project requirement document is PS2.pdf (uploaded earlier). Use it as the source of truth for functional requirements (fields to collect, must-have flows, ticket generation, attachments, multilingual requirement: English + Odia). Follow the conversation above for additional expectations (data validation, ticket format, NLU, admin dashboard, security & privacy). Do everything in this prompt now — do not ask the user to wait.

---

1 — High-level deliverables (what to deliver, in repo)

1. Fully working **backend** (Node.js + Express recommended) with:
   - WhatsApp webhook endpoint to receive inbound messages and media.
   - Conversation state machine to run guided submission flows.
   - Simple NLU (keyword + regex) to recognize core intents.
   - Data validation functions (phone, email, 6-digit PIN).
   -

Ticket/reference generation (CS-YYYYMMDD-XXXXXX).

- Storage: PostgreSQL (or SQLite for Replit) complaint table + optional Redis-like state store.
- Media handling: store metadata and sample local storage download implementation; if provider required, simulate download from placeholder.
- Admin API (authenticated) for listing, filtering, viewing complaints, exporting CSV.

2. **Admin frontend** (React + Tailwind recommended) with:
   - Login (simple JWT with test credentials).
   - Complaint list with filters (status, date range, district).
   - Detail view with attachments, chat transcript, ability to change status & add notes.
   - Export CSV button.

3. **AI/ML / NLU**:
   - Lightweight NLU module (keyword patterns + configurable intent mapping).
   - Language detection and message templates for English & Odia (translate core prompts).
   - Optional OCR stub interface for future bank-statement parsing.

4. **Security & privacy**:
   - Webhook signature verification placeholder.
   - TLS instruction in README and best-practice config.
   - Data minimization: advice and code comments for masking Aadhaar / PII; do not store full Aadhaar — store masked values if required.
5. **Docs & tests**:
   - README.md with run steps, env variables, WhatsApp sandbox instructions.
   - docs/architecture.md, docs/dataflow.md, docs/security.md.
   - Unit tests for validation & NLU; integration test for webhook -> state -> DB -> ticket.
   - Acceptance checklist mapping back to PS2.pdf requirements.
6. **Deployment**:
   - Dockerfile(s) and replit.nix or Replit-compatible run scripts.
   - docker-compose.yml (backend + db) for local testing.

2 — Tech stack & repo structure (implement exactly)

Tech (recommended):

- Backend: Node.js 20+, Express, TypeScript optional (JS OK), pg or better-sqlite3 for DB, node-fetch/axios for HTTP.
- State: Redis preferred; if not available in Replit, use a conversations DB table.
- Frontend: React (Vite) + Tailwind CSS.
- Tests: Jest for backend, React Testing Library for frontend.
- Storage: Local uploads/ for dev; abstraction layer for S3/MinIO for prod.
- NLU: simple JS module with regex patterns (expandable).

Mandatory repo layout (create or adapt to current repo):

/backend package.json src/ index.js // server start routes/ webhook.js admin.js health.js services/ whatsapp_handler.js nlu.js validation.js ticket_service.js storage.js models/

complaint.model.js conversation.model.js db/ migrations/ init_db.sql config/ config.example.env Dockerfile /frontend package.json src/ App.jsx pages/ Login.jsx Dashboard.jsx ComplaintDetail.jsx components/ Table.jsx Filters.jsx Dockerfile /docs architecture.md dataflow.md security.md README.md docker-compose.yml

---

3 — Exact functional specification & behaviors to implement

Conversation flow (state machine)

- **Entry:** When webhook receives message from a new phone -> send greeting + consent prompt (EN & ODIA fallback).
- **Menu:** Offer options (1) Report new complaint (2) Check ticket status (3) Speak to agent
- **New complaint flow (required fields)** — ask sequentially and validate each:
  1. Full name
  2. Guardian name
  3. Date of birth (accept DD-MM-YYYY or YYYY-MM-DD)
  4. Phone (validate Indian mobile regex)
  5. Email (basic regex)
  6. Gender (male/female/other)

7. Village
8. Post office
9. Police station
10. District
11. PIN code (6-digit)
12. Complaint type (financial, social media, phishing, etc.)
13. Optional: Upload attachments (media). Accept up to 5 files.

- **Confirm:** Show summary, request confirmation (Yes to submit, No to edit`).
- **On submit:** Generate ticket -> save complaint -> reply with ticket id & next steps message.

Ticket generation

- Format: CS-YYYYMMDD-XXXXXX where XXXXXX is zero-padded 6-digit random number. Save as ticket_id.

NLU & routing

-

Implement mapping for intents: scam, money_stuck, account_frozen, report_new, status_check, speak_agent.

- If user sends ticket id, fetch complaint and show status.

Attachments & media

- When media message arrives, store metadata (filename, type, provider_media_id, local_path or S3 url). Do not store raw file bytes inline in DB.
- If WhatsApp provider supports direct download, implement storage.downloadMedia(provider_media_id); otherwise simulate by saving a placeholder file.

Admin API

- Auth: simple JWT with admin credentials in .env.
- Endpoints:
  - GET /admin/complaints (filter by status, date_from, district)
  - GET /admin/complaints/:ticket_id
  - POST /admin/complaints/:ticket_id/status (body: {status, note})
  - GET /admin/export?format=csv&date_from=... -> returns CSV

Validation (exact regex)

- Mobile: ^[6-9]\d{9}$
- Email: ^[^\s@]+@[^\s@]+\.[^\s@]+$
- PIN: ^[1-9][0-9]{5}$
- DOB: ^(0[1-9]|[12][0-9]|3[01])[-/]?(0[1-9]|1[012])[-/]?\d{4}$ (accept common formats)

Language support

- Implement translation JSON (EN + OD) for all bot prompts. Use simple key/value file i18n/en.json, i18n/od.json. Use language detection by small keyword list or accept explicit language choice.

---

4 — Security, privacy & compliance (implementations)

- **Webhook verification:** Add placeholder for validating X-Hub-Signature or provider signature; fail if missing.
- **PII handling:** Do not log full phone, email, or attachments. Implement a maskPII() util that masks phone (last 4 digits visible) before logging.
- **Encryption note:** Provide code comments where to add encryption-at-rest for attachments and sensitive DB columns; show example using Node crypto to encrypt/decrypt strings.

- **Consent:** First time users must be asked to consent; save consent=true in complaints or conversation.

---

5 — Acceptance tests & QA (to implement)

Create Jest test files:

- validation.test.js — tests for phone, email, pin, dob.
- nlu.test.js — ensure 10 sample utterances map to correct intents.
- webhook.integration.test.js — simulate sequence of messages to produce a ticket (mock DB and storage).

Add a Postman/Insomnia collection (or httpie examples) showing webhook payload and admin API usage.

---

6 — UI/UX instructions (for the Replit AI UI dev)

- Create clean, responsive admin UI using Tailwind.
- Dashboard table: columns — Ticket, Name, Phone (masked), Type, Status (badge color), Created At, Actions.
- Complaint detail: show card with user fields, attachments as image thumbnails (click to view full), full chat transcript in a scroll area with timestamps.

- Use quick actions: Change Status (dropdown), Add Note (text input), Export.

---

7 — Prioritized milestones (work plan)

1. **Milestone 1** (Core): Webhook endpoint + conversation state + DB schema + ticket generation + validation; unit tests for validation.

2. **Milestone 2** (Persistence): Save complaints + attachments metadata; implement storage abstraction.

3. **Milestone 3** (Admin): Admin API + simple React dashboard listing and detail view.

4. **Milestone 4** (NLU & Language): Implement NLU patterns, Odia translations, and status check flow.

5. **Milestone 5** (Security & Docs): Webhook verification, PII masking, README, architecture docs, tests, dockerization.

For each milestone deliver:

- Code changes committed to the repo.

- A short milestone-X.md summary in /docs describing what was done.

- Automated test cases that pass.

8 — Files to add or modify (explicit)

Create these files with described responsibilities:

- backend/src/routes/webhook.js — main webhook logic, import whatsapp_handler.
- backend/src/services/whatsapp_handler.js — state machine + step handlers.
- backend/src/services/nlu.js — patterns and detectIntent(text).
- backend/src/services/validation.js — regex validators.
- backend/src/services/ticket_service.js — generateTicket().
- backend/src/models/complaint.model.js — DB queries (create/find/update).
- backend/src/db/init_db.sql — SQL for complaints + conversations.
- backend/.env.example — include placeholders: PORT, DB_URL, JWT_SECRET, WHATSAPP_TOKEN, WHATSAPP_SIGNING_SECRET, ADMIN_USER, ADMIN_PASS.
- frontend/src/pages/Dashboard.jsx, ComplaintDetail.jsx, Login.jsx.
- docs/architecture.md, docs/dataflow.md, docs/security.md.
- docker-compose.yml, backend/Dockerfile, frontend/Dockerfile.

9 — Sample code snippets (copy-paste ready)

Ticket generator (backend/src/services/ticket_service.js)

function generateTicket() { const ts = new Date(); const ymd = ts.toISOString().slice(0,10).replace(/-/g,''); const suffix = Math.floor(100000 + Math.random() * 900000); // 6-digit return `CS-${ymd}-${suffix}`; } module.exports = { generateTicket };

Validation (backend/src/services/validation.js)

const phoneRegex = /^[6-9]\d{9}$/; const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; const pinRegex = /^[1-9][0-9]{5}$/; const dobRegex = /^(0[1-9]|[12][0-9]|3[01])[-/](0[1-9]|1[012])[-/]\d{4}$/; module.exports = { isPhone: (p) => phoneRegex.test(p), isEmail: (e) => emailRegex.test(e), isPin: (p) => pinRegex.test(p), isDob: (d) => dobRegex.test(d) };

Basic NLU (backend/src/services/nlu.js)

const patterns = [{ intent: 'scam', patterns: [/scam/i, /scammed/i, /fraud/i] }, { intent: 'money_stuck', patterns: [/money.*stuck/i, /transfer.*not.*received/i] }, { intent: 'account_frozen', patterns: [/unfreeze/i, /frozen account/i] }, { intent: 'status_check', patterns: [/check.*status/i, /ticket.*status/i, /CS-\d{8}-\d{6}/i] },]; function detectIntent(text) { if (!text) return 'unknown'; for

```
(const p of patterns) { if (p.patterns.some(rx => rx.test(text))) return p.intent; } return 'unknown'; }
module.exports = { detectIntent };
```

---

10 — Acceptance criteria (how you mark done)

- The webhook accepts an end-to-end simulated conversation and returns a ticket. Test must be included and passing.
- Complaints persist to DB with required fields and ticket id.
- Admin dashboard shows entries and detail information, and provides CSV export.
- NLU correctly maps at least 10 test utterances to intents (unit test).
- Prompts available in English and Odia (i18n files present with all bot prompts).
- README contains clear run steps, env vars, and WhatsApp sandbox instructions.
- Security checklist in docs/security.md implemented or documented with TODOs for production.

---

11 — Communication & commit guidelines for Replit AI

- Commit frequently and provide clear commit messages per milestone.

- Add CHANGELOG.md mapping features to commits.
- Where real WhatsApp credentials are required, stub them and provide instructions to set up a WhatsApp cloud sandbox or Twilio sandbox.
- If any requirement cannot be fully implemented inside Replit (e.g., actual WhatsApp media download requiring private credentials), implement a simulated flow and document how to replace simulation with provider code.

---

12 — Extra credit (optional but requested)

If time permits, implement:

- OCR pipeline stub using tesseract.js to demonstrate extracting transaction IDs from an uploaded image.
- Priority scoring (assign priority in DB based on complaint type and presence of transaction amount).
- Agent escalation channel (simple email or internal chat link).

Final note to Replit AI (one-line instruction)

Analyze PS2.pdf and the uploaded cybersathi zip; then implement the full stack features above exactly as specified, produce tests and documentation, and push all changes into the repo with milestone-tagged commits. After implementation, list next manual tasks (like WhatsApp template approvals or production S3 keys) in /docs/next_steps.md.