Alright Suryansh — here’s a crisp, end-to-end MVP plan you can ship from your current Turborepo. I’ll cover feasibility, architecture, tech stack, a pragmatic scope, data models, smart contracts, AI bits, and step-by-step implementation (wired into your repo layout).

---

1) MVP Scope (what we’ll actually build)

Actors

- **Tourist (public app):** registers, gets a Digital Tourist ID (DTID), sees geo-fence warnings, uses Panic button, can opt-in location sharing.
- **Police/Tourism staff (dashboard):** see live heatmap, alerts feed, tourist detail (ID + last location), trigger/close e-FIR (MVP mock), manage risk zones.
- **Verifier (hotel/airport desk):** verifies DTID QR at check-in; creates trips.

MVP Features

- **Digital ID (DTID)**: off-chain PII, on-chain hash anchor + DID URI; QR code for verification.
- **Geo-fencing**: client-side fence checks + server validation; configurable “Risk Zones.”
- **Panic/SOS**: push + SMS/WhatsApp fallback to ops center + trusted contacts; live location session link.
- **Anomaly rules (v0)**: no-AI first (reliable + explainable) — “long inactivity,” “sudden drop,” “off-route” via simple thresholds.
- **Dashboards**: live map with clusters/heat, alert inbox, DTID lookup, case log (e-FIR mock).
- **Privacy**: opt-in tracking; E2E for panic session; PII encrypted at rest; on-chain only anchors, never PII.

Non-Goals (MVP)

- Aadhaar integration (needs govt rails) → mock KYC fields.
- Full e-FIR integration → create a case record + export PDF.

---

2) High-Level Architecture

```
[ Mobile App (Expo) ]  <->  [ API Gateway (NestJS) ]  <->  [ Postgres+PostGIS (Neon) ]
       |                          |            \
       |                          |             -> [ Redis (BullMQ) for alerts,
digests ]
       |                          |             -> [ Notification Service (FCM + SMS
provider) ]
       |                          |
       |                          ---> [ Anomaly Worker (Node) ]
       |
       ---> [ Websockets (Socket.IO) ]  <->  [ Police Dashboard (Next.js) ]

[ Smart Contracts (Hardhat, packages/smart-contracts) ]
   - TouristIDRegistry (store hash anchors, DID URI)
```

Data flow snapshot

- Create DTID → hash(PHI minimal pack) stored in DB, anchor hash on chain, issue QR with DID URI + signature.
- App streams location (opt-in) → API validates geofences → alerts via WS/FCM → stored in PostGIS.
- Panic → notify nearest police unit + contacts; start secure live location “session”.
- Dashboard subscribes WS → shows heatmap, clusters, alerts; staff can acknowledge/close.

---

3) Tech Stack (pragmatic + fast)

Monorepo (you already have it)

- **apps/**
  - `web/` → Tourist web (marketing + DTID wallet/backup)
  - `police/` → Police/Tourism dashboard (Next.js App Router)
  - `docs/` → Dev/docs (already present)
- **packages/**
  - `api/` → **NestJS** backend (REST + WS, BullMQ workers)
  - `db/` → Prisma schema + migration scripts (Postgres + PostGIS)
  - `smart-contracts/` → your Hardhat package (Polygon Amoy testnet for MVP)
  - `ui/` → your existing shared UI (add map components)
  - `common/` → shared types (zod), constants, DTOs
- **Mobile** (optional path under `apps/mobile/`) → **Expo (React Native)** with Expo Location + FCM

Core choices

- **Postgres + PostGIS** (Neon): geofencing, clustering, nearest-unit queries.
- **Maps**: Mapbox GL JS (web) + Mapbox SDK (mobile) OR react-map-gl.
- **Auth**: NextAuth (passkeys + email OTP) for web; JWT for mobile. Roles: TOURIST, VERIFIER, POLICE, ADMIN.
- **Notifications**: FCM for push; SMS provider (Gupshup/Twilio/etc.) for fallback; WhatsApp template (optional).
- **Realtime**: Socket.IO namespaces: `/tourist`, `/ops`.
- **Crypto**: `tweetnacl` / `libsodium` for signatures; `jose` for JWT, `@didtools` (optional) for DID URIs.
- **AI (later)**: Start with **rules**; keep a feature table to swap in ML (Isolation Forest) later.

---

4) Extend your Turborepo (concrete)

Add folders

```
apps/
  police/
  mobile/          # (optional now, recommended)
packages/
  api/
  db/
  common/
```

Root `package.json` workspaces (snippet)

```
{
  "workspaces": [
    "apps/*",
    "packages/*"
  ]
}
```

Root `turbo.json` (pipelines)

```
{
  "$schema": "https://turbo.build/schema.json",
  "pipeline": {
    "build": { "dependsOn": ["^build"], "outputs": ["dist/**", ".next/**"] },
    "dev": { "cache": false, "persistent": true },
    "lint": {},
    "test": {}
  }
}
```

---

5) Data Model (Prisma, `packages/db`)

Key tables

- `User` (role, auth)
- `TouristProfile` (minimal KYC, encrypted blob, public DID)
- `Trip` (itinerary, validity window)
- `DigitalID` (DTID, on-chain anchor hash, QR payload)
- `EmergencyContact`
- `LocationPing` (PostGIS `geometry(Point,4326)`, speed, accuracy)
- `RiskZone` (Polygon/MultiPolygon + level)
- `Alert` (type: PANIC | GEOFENCE | INACTIVITY | DROP | OFF_ROUTE; status)
- `CaseFile` (e-FIR mock, pdf link)
- `Unit` (police team, current location)
- `AuditLog`

Prisma schema sketch

```
// packages/db/schema.prisma
datasource db { provider = "postgresql"; url = env("DATABASE_URL") }
generator client { provider = "prisma-client-js" }

model User {
  id        String  @id @default(cuid())
  email     String  @unique
  role      Role
  createdAt DateTime @default(now())
  profile   TouristProfile?
}

enum Role { TOURIST VERIFIER POLICE ADMIN }

model TouristProfile {
  id        String @id @default(cuid())
  userId    String @unique
  name      String
  docType   String  // "passport" | "license" | "aadhaar-mock"
  docRef    String  // last4 or masked ID
  didUri    String
  encPII    Bytes   // encrypted JSON blob
  createdAt DateTime @default(now())
  user      User @relation(fields: [userId], references: [id])
  trips     Trip[]
  contacts  EmergencyContact[]
  dtids     DigitalID[]
}

model Trip {
  id         String @id @default(cuid())
  touristId  String
  from       DateTime
  to         DateTime
  origin     String
  itinerary  Json     // waypoints
  createdAt  DateTime @default(now())
  tourist    TouristProfile @relation(fields: [touristId], references: [id])
}

model DigitalID {
  id         String @id @default(cuid())
  touristId  String
  tripId     String?
  anchorHash String  // on-chain hash
  qrPayload  String  // JWS or DID doc bundle
  validFrom  DateTime
  validTo    DateTime
  createdAt  DateTime @default(now())
  tourist    TouristProfile @relation(fields: [touristId], references: [id])
  trip       Trip? @relation(fields: [tripId], references: [id])
}

model EmergencyContact {
  id         String @id @default(cuid())
  touristId  String
  name       String
  phone      String
  relation   String
  tourist    TouristProfile @relation(fields: [touristId], references: [id])
}

model LocationPing {
  id         String @id @default(cuid())
  touristId  String
  at         DateTime @default(now())
  lat        Float
  lng        Float
  speedKmh   Float?
  accuracyM  Float?
  // store Geo as WKT for Prisma; PostGIS column added via migration
  tourist    TouristProfile @relation(fields: [touristId], references: [id])
  @@index([touristId, at])
}

model RiskZone {
  id         String @id @default(cuid())
  name       String
  level      Int      // 1-5 severity
  geojson    Json     // polygon(s)
  createdBy  String
  createdAt  DateTime @default(now())
}

model Alert {
  id         String @id @default(cuid())
  touristId  String?
  type       AlertType
  severity   Int
  status     AlertStatus @default(OPEN)
  meta       Json
  createdAt  DateTime @default(now())
  assignedTo String?
}

enum AlertType { PANIC GEOFENCE INACTIVITY DROP OFF_ROUTE }
enum AlertStatus { OPEN ACK CLOSED }

model CaseFile {
  id         String @id @default(cuid())
  alertId    String
  summary    String
  pdfUrl     String?
  createdBy  String
  createdAt  DateTime @default(now())
}
```

PostGIS migration (raw SQL)

- Add `geometry(Point,4326)` column on `LocationPing` and GIST index.
- Maintain materialized views for **clusters** and **last known location per tourist**.

---

6) Smart Contract (in your `packages/smart-contracts`)

Purpose (MVP): tamper-proof anchor of a DTID bundle hash (no PII), and a DID URI.

- Network: Polygon Amoy (testnet) or any cheap L2.
- Contract: `TouristIDRegistry.sol`
  - `register(bytes32 anchorHash, string didUri)` → emits `Registered(tourist, anchorHash, didUri, block.timestamp)`
  - `update(bytes32 newHash, string didUri)` with owner-only (EOA per tourist or custody by department multisig in MVP)
  - `getLatest(address subject) -> (bytes32, string, uint256)`

Security: minimal; keep it simple. Your DB stores everything else.

Hardhat tasks

- `npx hardhat run scripts/deploy.ts --network amoy`
- `npx hardhat task:register --hash <0x..> --did "did:web:..."`

---

7) Geo-Fencing & Safety Score

Server-side validation

- Risk zones table = polygons (GeoJSON) with `level`.
- On ping ingest, do `ST_Contains(zone.geom, ping.geom)`. If inside, raise GEOFENCE alert; severity = `zone.level`.

Client-side

- Mobile uses OS geofencing APIs (Android GeofencingClient, iOS Region Monitoring) to reduce latency and battery.

Safety Score (0–100)

```
Score = 100
  - 20 * I_inRiskZone(level)             // 1..5 scaled
  - 10 * night_hours_weight               // 22:00–05:00 activity
  - 10 * off_route_ratio                  // distance from itinerary > threshold
  - 10 * long_inactivity_events/week
  - 10 * speed_anomaly_ratio              // sudden high speed/zero speed anomalies
Clamp to [0,100]
```

Show it as “advisory”, not a label on the person.

---

8) Anomaly Detection (v0 Rules → ML-ready)

Rules (MVP)

- **INACTIVITY**: no ping for `> X` minutes during active window.

- **DROP**: altitude change or GPS accuracy spike + velocity pattern (e.g., `speed=0 + accuracy>80m for 10min`).
- **OFF_ROUTE**: `ST_Distance(ping, itinerary buffer 200m) > 300m` for N consecutive pings.

Data contract: Keep a `features_location` table (rolling aggregates) so you can later train Isolation Forest/XGBoost on the same features without plumbing changes.

---

9) API Design (NestJS in `packages/api`)

Modules

- `auth` (NextAuth/JWT integration, RBAC guard)
- `dtid` (create, verify, fetch)
- `geo` (zones CRUD, geofence check)
- `ping` (ingest location stream, last-seen, cluster feed)
- `alert` (create/ack/close, subscribe WS)
- `case` (generate PDF)
- `notify` (FCM/SMS)
- `units` (police unit positions)

Sample endpoints

```
POST /dtid/create            (verifier/admin)
GET  /dtid/:id               (police/admin/self)
POST /ping/batch             (tourist app)
GET  /map/heat               (police dashboard)
POST /alert/panic            (tourist app)
POST /alerts/:id/ack         (police)
POST /case/from-alert/:id    (police/admin)
POST /geo/zones              (admin)
GET  /geo/zones
```

WebSockets

- Namespace `/ops`: `alerts:new`, `alerts:update`, `map:clusters`
- Namespace `/tourist`: `panic:session:<id>` live updates

---

10) Frontends

apps/web (Tourist web)

- **Pages**: Home, “Get Digital ID”, Wallet (QR), Safety tips, Language switcher.
- **Flows**:
  - DTID issuance (verifier side also supports scanning).
  - Show QR, export as PDF.
  - Manage emergency contacts.
- **Tech**: Next.js App Router, NextAuth, i18n (next-intl), Tailwind + your `packages/ui`.

apps/police (Dashboard)

- **Views**:
  - Live map (clusters + heat layer + risk zone overlay).
  - Alerts inbox with triage board (OPEN/ACK/CLOSED).
  - Tourist lookup (DTID), last location, contact info (masked).
  - Zones manager (draw polygon → save).
  - Case file view (PDF export).
- **Tech**: Next.js + react-map-gl, Socket.IO client, shadcn/ui components from `packages/ui`.

apps/mobile (Expo)

- **Screens**: Sign-in (OTP/passkey), My ID (QR), Live Safety Score, Panic button, Settings (opt-in tracking).
- **SDKs**: `expo-location`, `expo-notifications`, `react-native-maps` (Mapbox optional).

---

11) Security & Privacy Defaults

- **PII off-chain only**, AES-GCM encrypted at rest (`encPII`), key derived from server KMS + user secret.
- **On-chain**: hash anchor + DID URI only.
- **Transport**: TLS everywhere. Panic session uses ephemeral keypair; location channel sealed (X25519).
- **Access**: RBAC guards; police can **view only minimal** (name masked; full PII gated on case open).
- **Consent**: explicit toggle for continuous tracking; default OFF.
- **Retention**: pings retention 90 days (MVP suggestion); aggregated stats longer.
- **Audit**: every staff read/write logged.

---

12) Step-by-Step Implementation

A) Boot the backend (`packages/api`)

1. `pnpm -w add -D typescript ts-node nodemon @types/node`

2. `pnpm -w add @nestjs/common @nestjs/core @nestjs/platform-express @nestjs/config class-validator class-transformer`

3. `pnpm -w add @nestjs/websockets @nestjs/platform-socket.io socket.io socket.io-client`

4. `pnpm -w add @nestjs/axios bullmq ioredis`

5. `pnpm -w add @prisma/client zod jsonwebtoken jose bcrypt`

6. Create Nest skeleton: `nest new packages/api` (or manual). Wire to Turborepo scripts.

7. Add `.env` in repo root (turbo can pass through):

   ```
   DATABASE_URL=postgres://...
   REDIS_URL=redis://...
   JWT_SECRET=...
   MAPBOX_TOKEN=...
   FCM_KEY=...
   CHAIN_RPC=https://...
   CONTRACT_ADDR=0x...
   ```

8. In `packages/db`, init Prisma, generate client. Add PostGIS migration SQL.
9. Implement modules in order: `auth` → `dtid` → `geo` → `ping` → `alert` → `notify`.

DTID create flow (verifier/admin)

- Receive `{profile, trip, contacts}`.
- Normalize + encrypt PII → `encPII`.
- Compute `anchorHash = keccak256( canonicalize({did, tripId, contacts_min}) )`.
- Call contract `register(anchorHash, didUri)`.
- Store `DigitalID` with `qrPayload = JWS(did, trip, validity, anchorHash, sig)`.
- Return QR payload.

Ping ingest

- Accept batch `{ts, lat, lng, speed, acc}`; insert; server validates geo-fences; raise alerts; push WS.

Panic

- Create `Alert{type:PANIC, severity:5}`; notify nearest `Unit` via geospatial nearest query; push to contacts.

B) Smart Contracts (you already have Hardhat)

1. Add `contracts/TouristIDRegistry.sol`:

   ```
   // SPDX-License-Identifier: MIT
   pragma solidity ^0.8.20;

   contract TouristIDRegistry {
       event Registered(address indexed subject, bytes32 anchorHash, string
   didUri, uint256 ts);

       mapping(address => bytes32) public latestHash;
       mapping(address => string) public latestDID;

       function register(bytes32 anchorHash, string calldata didUri) external {
           latestHash[msg.sender] = anchorHash;
           latestDID[msg.sender] = didUri;
           emit Registered(msg.sender, anchorHash, didUri, block.timestamp);
       }
   }
   ```

2. Deploy to testnet; save address in `.env`.

3. In `packages/api`, write a thin ethers service to call `register`.

C) apps/web (Tourist)

1. Add routes `/id/new`, `/id/wallet`, `/settings/contacts`.
2. Add QR code component (embed `qrPayload` JWS).
3. Build i18n (10+ languages): start with static JSON; wrap UI with language picker; plan server translations later.
4. Panic page (web fallback) with **deep-link** to mobile if installed.

D) apps/police (Dashboard)

1. Map page:
    - Layers: **clusters** (server endpoint), **heat** (aggregated last 10m), **risk zones** overlay.
    - Alerts drawer (WS live).
2. Tourist lookup: Enter DTID → show last known location + valid trip window; actions (ACK/CLOSE alert).
3. Zones manager: draw polygon → POST `/geo/zones`.
4. Case file: click "Generate Case" → POST → show PDF link.

E) apps/mobile (Expo) — recommended

1. Background location (low-power strategy):
    - Significant-change updates, throttle to every 30–60s when moving, back off when stationary.
2. Panic button:
    - Single tap → `POST /alert/panic`; start foreground service to send location every 5s for 10 minutes.
3. Geo-fence:
    - Download zones bounding box for region; register OS geofences; show banner + vibrate.

---

13) Testing Plan

- **Unit**: Prisma services, geofence checks (`ST_Contains`) with fixtures.
- **Integration**: Ping ingest → alert raised → WS event received in dashboard.
- **E2E**: Cypress (web, dashboard). For mobile, Detox/maestro for flows.
- **Load**: 5k concurrent tourists (simulate pings), cluster query stays < 300ms.
- **Security**: Attempt replay of QR; validate signature + freshness.

---

14) DevOps & Deployment

- **DB**: Neon Postgres with PostGIS enabled.
- **Redis**: Upstash/Valkey for BullMQ queues.
- **Backend**: Render/Fly.io; set `PORT`, `DATABASE_URL`, `REDIS_URL`.

- **Web apps**: Vercel (apps/web, apps/police).
- **Mobile**: Expo EAS build; FCM keys configured.
- **CI**: GitHub Actions — lint, test, prisma migrate, build, deploy.
- **Observability**: pino logs → Logtail; uptime checks; Sentry for web & API.

---

15) Public vs Police Implementation Notes

Public (Tourist)

- Minimal friction sign-in (OTP/passkey).
- Clear consent toggles for tracking + analytics off by default.
- Panic always available offline; queued request when connectivity returns.
- Multilingual: store `accept-language` & persist preference; provide voice-to-text button on Panic (for elderly).

Police/Tourism

- Role-gated dashboards:
  - **Tourism staff**: zone management, trend reports, aggregate stats.
  - **Police**: live alerts, unit dispatch view, case creation.
- Acknowledge path: OPEN → ACK (assign to unit) → CLOSED (with resolution note).
- Exportable case PDF includes: DTID details (non-PII), last locations, alert log, timestamps, on-chain anchor tx hash.

---

16) Example Commands & Scripts

Root scripts

```json
{
  "scripts": {
    "dev": "turbo run dev",
    "build": "turbo run build",
    "db:migrate": "pnpm --filter db prisma migrate deploy",
    "db:studio": "pnpm --filter db prisma studio"
  }
}
```

Spin everything locally

```bash
# 1) start db/redis (docker compose) or point to Neon/Upstash
pnpm i
pnpm run db:migrate
pnpm --filter packages/api dev
pnpm --filter apps/web dev
pnpm --filter apps/police dev
```

---

17) What to build first (sensible order)

1. **DB + APIs**: DTID create/verify → Zones CRUD → Ping ingest → Alerts.
2. **Dashboard**: live map + alerts feed.
3. **Tourist web**: DTID issue, QR wallet, contacts, panic (web).
4. **Mobile (Expo)**: background location, panic, geofencing.
5. **On-chain anchor** (toggle behind env flag) once the flow is stable.
6. **Anomaly rules** (cron worker) → then iterate toward ML.

---

18) Nice-to-have (post-MVP)

- On-device ML (TinyML) to detect “fall” or “sudden stop.”
- Family portal link (read-only live session).
- Offline QR verification (verifier app caches recent CRLs).
- Unit auto-dispatch suggestion based on ETA graph (OSRM).

If you want, I can scaffold the **NestJS API module structure**, a **minimal Prisma schema file**, and a **Solidity contract** directly into your folder layout and give you ready-to-run files.