Biometric Attendance Module – Requirements (In-App Camera + Geofence, Expo SDK 53 + Laravel)

1. Scope

This module ensures that hospital staff attendance is recorded securely and accurately using:

1. **Geofencing** – user must be **physically inside a 100 m radius** of the hospital/office.
2. **Custom Face Recognition** – user's **face must match** the biometric template stored in the hospital database, using an **in-app camera** (no device Face ID / OS biometrics).

Attendance is marked only when **both** conditions pass, preventing proxy attendance and ensuring physical presence.

Platform:

- **Frontend**: Expo SDK 53, React Native, React Navigation, `expo-camera`, `expo-location`.
- **Backend**: Laravel (API only), MySQL (or compatible RDBMS) for data storage.

---

2. System Overview

2.1 High-Level Workflow

1. User Login
2. User taps **"Mark Attendance"**
3. App sends GPS location → backend validates **100 m geofence**
4. If inside geofence → app prompts **face capture**
5. Backend compares live face with stored biometric template
6. If match → **attendance marked present**
7. Else → **attendance rejected** and logged

---

3. Functional Requirements

3.1 User Roles

- **Employee/Staff**
    - Login to the app.
    - Capture face during enrollment (one-time).
    - Mark **check-in** and **check-out** using geofence + face recognition.
    - View own attendance history (future extension).
- **Admin**
    - Manage users and assign geofence locations.
    - View attendance logs.
    - View failed attempts (outside geofence / face mismatch).

- Trigger or reset face enrollment when needed.
- Configure geofence entries per hospital/branch.

---

4. Mobile App Flows

4.1 Authentication Flow (Mobile)

1. User opens the app and logs in with:
   - Mobile Number, and
   - Password.
2. On successful login:
   - Backend issues a **JWT / Sanctum token**.
   - App stores token securely (SecureStore).
3. All subsequent API calls include:
   - `Authorization: Bearer <token>` header.

(Auth module is shared with existing HIMS; this document focuses on biometric/geofence flow.)

4.2 Biometric Enrollment (One-Time Setup)

4.2.1 Enrollment Process

1. After first login (or via a profile/biometric setup screen), user is prompted to **enroll face**.
   For some flows, an admin may enroll staff from an admin app/web panel.
2. App:
   - Requests **camera permission**.
   - Opens **front camera** using `expo-camera`.
   - Shows a guide overlay (e.g., oval frame, "Center your face, good lighting, no mask").
3. User taps **"Capture"**.
4. App:
   - Captures an image.
   - Optionally shows a preview with "Retake" / "Use".
   - Converts image to a suitable format (e.g., JPEG base64 or compressed JPEG).
5. App sends image to backend endpoint:
   - `POST /api/face/enroll`
   - Body includes:
     - `image` (base64 or multipart file).
6. Backend:
   - Validates that a **single face** is present.
   - Extracts a **facial embedding** (vector representation).
   - Encrypts and stores the embedding linked to the user (`face_template_reference`).
   - Stores audit info (who enrolled, timestamp).
   - Returns success/failure status.
7. App:
   - If success → mark user as **Face Enrolled** in local state / profile.
   - If fail → show error and allow re-enrollment.

4.2.2 Biometric Storage

Table: `user_biometrics`

Field	Description
`id`	Primary key
`user_id`	Reference to `users` table
`face_embedding`	Encrypted JSON/vector embedding
`created_at`	Enrollment timestamp
`enrolled_by`	Admin ID or user ID who enrolled

Important:

- Raw images should **not** be permanently stored except when strictly needed (e.g., for audit with explicit policy).
- Only **face embeddings** should be stored for matching.
- Embeddings must be **encrypted at rest**.

4.3 Check-In Flow (Geofence + In-App Camera)

When user taps **"Check-In"** in the app:

4.3.1 Location Permission & Retrieval

1. App requests **foreground location** permission.
2. If denied:
    - Show user-friendly message and stop flow.
3. If granted:
    - Use `expo-location` to get current GPS coordinates (`latitude`, `longitude`) with high accuracy.

4.3.2 Geofence Pre-Check (Backend)

1. App sends request:
    - `POST /api/attendance/pre-check`
    - Body: `{ lat: <number>, lng: <number> }`
    - Header: `Authorization: Bearer <token>`
2. Backend:
    - Resolves authenticated user from token.
    - Fetches assigned geofence for that hospital/branch.

Table: `geofences`

Field	Description
`id`	Primary key

Field	Description
`name`	Hospital/Branch name
`center_lat`	Latitude
`center_lng`	Longitude
`radius`	Default 100 meters
`status`	Active/Inactive
`created_at`	Created timestamp
`updated_at`	Updated timestamp

3. Backend calculates distance using the **Haversine formula** between:
    - Geofence center `(center_lat, center_lng)` and
    - User coordinates `(lat, lng)`.
4. Decision:
    - If `distance > radius (100 m)`:
        - Returns: `{ status: "geo_failed", message: "You are outside the allowed area" }`.
        - Logs event as `geo_failed` in `audit_logs`.
    - If `distance <= radius`:
        - Returns: `{ status: "geo_ok", message: "Proceed to face verification" }`.

4.3.3 Face Capture (In-App Camera)

Only if `status = "geo_ok"`:

1. App:
    - Opens **front camera** using `expo-camera`.
    - Displays framing guides and text instructions.
2. User taps **"Capture Face"**.
3. App:
    - Captures the frame.
    - Optionally compresses image to reduce upload size.
    - Sends POST request:
        - `POST /api/attendance/check-in`
        - Headers: `Authorization: Bearer <token>`
        - Body:
            - `image` (captured face image),
            - `lat`, `lng` (optional redundancy),
            - Optional metadata (device model, OS version, timestamp).

4.3.4 Face Verification & Attendance Creation (Backend)

Backend performs:

1. Identifies user via token.

2. Detects face in incoming image.
3. Extracts **live face embedding**.
4. Retrieves stored embedding from `user_biometrics` for that user.
5. Compares embeddings:
    - Computes a **similarity score** (e.g., cosine similarity or distance metric).
6. Decision rule:
    - If similarity ≥ configured threshold:
        - Face match **success**.
    - If similarity < threshold:
        - Face match **failed**.
        - Logs event as `face_failed` in `audit_logs`.
        - Returns error: `{ status: "face_failed", message: "Face did not match enrolled profile" }`.
7. If **both** geofence and face verification are successful:
    - Backend checks if user already has a check-in record for the day.
    - If not, creates an `attendance` record.

Table: `attendances`

Field	Description
`id`	Primary key
`user_id`	User reference
`date`	Attendance date
`checkin_time`	Check-in timestamp
`checkout_time`	Check-out timestamp (nullable)
`checkin_lat`	GPS latitude
`checkin_lng`	GPS longitude
`checkout_lat`	GPS latitude (optional)
`checkout_lng`	GPS longitude (optional)
`method`	e.g., `face+geo`
`status`	`present` / `late` / etc.
`remarks`	Optional notes
`created_at`	Created timestamp
`updated_at`	Updated timestamp

8. Response to app:
    - Success: `{ status: "success", time: "...", date: "...", site: "...", ... }`.
    - App shows: “Checked-in successfully at HH:MM”.

4.3.5 App Response

- On success:
    - Show confirmation and possibly update attendance UI.
- On failure (`geo_failed` or `face_failed`):
    - Display message and allow retry where appropriate.

4.4 Check-Out Flow

1. User taps **“Check-Out”**.
2. App again performs **location retrieval** (same as check-in).
3. Optional:
    - Repeat **face capture** for stricter security, or
    - Use **token + location** only for a simpler UX.
4. Backend:
    - Finds today’s attendance record for the user.
    - Validates rules (e.g., cannot check out before checking in).
    - Updates record with:
        - `checkout_time`,
        - optional `checkout_lat`, `checkout_lng`.
5. App:
    - Shows “Checked-out at HH:MM”.

---

5. Backend Requirements (Laravel)

5.1 API Endpoints

- `POST /api/login`
    - Input: credentials.
    - Output: token.
- `POST /api/face/enroll`
    - Authenticated.
    - Input: face image.
    - Action: generate and store encrypted `face_embedding` for the user.
- `POST /api/attendance/pre-check`
    - Authenticated.
    - Input: `lat`, `lng`.
    - Action: geofence distance check; returns `geo_ok` or `geo_failed`.

- `POST /api/attendance/check-in`
    - Authenticated.
    - Input:
        - `image` (captured face),
        - optionally `lat`, `lng`.
    - Action:
        - Validate face against stored embedding.
        - Confirm user is within geofence.
        - If success → create `attendance` check-in.
- `POST /api/attendance/check-out`
    - Authenticated.
    - Input:
        - `lat`, `lng` (if geofence also required for checkout),
        - optional image if repeating face verification.
    - Action:
        - Update existing attendance row with `checkout_time`.
- (Future) `GET /api/attendance/history`
    - Authenticated.
    - Input: date range.
    - Output: list of attendance records for the logged-in user.
- (Admin) APIs:
    - Manage geofences (`/api/geofences` CRUD).
    - View logs (`/api/audit-logs`).
    - View and manage biometric enrollment (`/api/user-biometrics`).

5.2 Additional Tables

`audit_logs`

Field	Description
`id`	Primary key
`user_id`	User reference (nullable if unauthenticated)
`event_type`	`geo_failed`, `face_failed`, `enroll_failed`, etc
`details`	JSON/text: lat/lng, reason, device info, etc.
`created_at`	Timestamp

5.3 Geofence Logic

- Use the **Haversine formula** to compute distance between:

    - Geofence center `(center_lat, center_lng)` and current `(lat, lng)`.
- If `distance <= radius (100 m)`:
    - Allow face verification to proceed.
- If `distance > radius`:
    - Reject attendance attempt.
    - Log `geo_failed` in `audit_logs`.

5.4 Face Recognition Logic

- **Enrollment**:
    - Validate presence of exactly one face.
    - Extract **embedding**.
    - Encrypt and store embedding in `user_biometrics`.
    - Do not persist original image unless explicitly required with policy.
- **Verification**:
    - Detect face in incoming image.
    - Generate new embedding.
    - Compare with stored embedding using a similarity metric.
    - Use configurable threshold for match / mismatch decision.
    - Log all failures as `face_failed` with details.
- Processing can be:
    - Implemented directly in Laravel using a library, **or**
    - Delegated to a **Python microservice / external AI service**, with Laravel calling it and handling responses.
      Laravel remains the core API controller and orchestrator.

---

6. Frontend Requirements (Expo SDK 53)

6.1 Libraries

- `expo-location` – GPS and permissions.
- `expo-camera` – front camera capture.
- `expo-secure-store` – secure token storage.
- `axios` – HTTP client.
- `@react-navigation/native`, `@react-navigation/native-stack` – navigation.

6.2 Screens

- **LoginScreen**
    - Inputs: mobile number, password.
    - On success: store token, navigate to Home/Attendance screen.
- **FaceEnrollScreen**

- Shows front camera preview.
- Capture, retake, and upload to `/api/face/enroll`.
- Shows status of enrollment (success/fail).

- **AttendanceScreen**
  - Button: "Check-In".
  - Flow:
    - Get location → call `/api/attendance/pre-check`.
    - If `geo_ok` → open camera for face capture.
    - Capture face → send to `/api/attendance/check-in`.
    - Show success or error based on response.
  - Button: "Check-Out".
    - Similar flow, calling `/api/attendance/check-out`.

- (Future) **HistoryScreen**
  - Lists past attendance records for the logged-in user.

---

7. Non-Functional Requirements

- **Security**
  - All API communication uses **HTTPS**.
  - Face embeddings stored **encrypted at rest**.
  - Raw images are not stored permanently (or stored under strict policy).
  - Authentication via token (JWT/Sanctum) for all attendance endpoints.
  - Rate limiting on attendance attempts to prevent abuse.
  - Prevention of multiple check-ins per day unless business rules allow.

- **Performance**
  - Target **< 5 seconds** total check-in time under normal network conditions.
  - Compress images before upload (e.g., reasonable resolution and quality).
  - Efficient face matching to handle concurrent staff usage.

- **Reliability**
  - Graceful handling when:
    - Location is temporarily unavailable.
    - Camera permission is denied.
    - Backend times out or is unreachable.
  - Clear error messages and retry options for users.

- **Auditability**
  - Every failed attempt (geofence or face) recorded in `audit_logs`.
  - Admin can filter and review:
    - `geo_failed` attempts (outside location).
    - `face_failed` attempts (face mismatch).

- Logs include timestamp, coordinates, and device metadata where possible.

---

8. Summary of Flow (Text)

1. **Login** → user authenticates and receives token.
2. If first time or not enrolled → **Face Enroll** via in-app camera.
3. Tap **"Check-In"**:
   - App gets **location** → calls `/attendance/pre-check` → receives `geo_ok` or `geo_failed`.
   - If `geo_ok` → app opens camera, captures face → sends to `/attendance/check-in`.
   - Backend verifies **geofence + face** → creates/updates attendance row as present.
4. Tap **"Check-Out"**:
   - App sends location (and optionally face) → backend updates attendance checkout time.