
1️⃣ Infant ankle tag — firmware architecture

Core goals: ultra-low power, secure ID, tamper detection, periodic beacons.

Main modules:

- **Bootloader module**
  - **Tasks:** Secure firmware update (OTA via BLE or wired), image integrity check, rollback.
  - **Features:** Signed firmware validation, versioning, minimal flash footprint.
- **Hardware abstraction layer (HAL)**
  - **Peripherals:** GPIO, timers, RTC, ADC (battery sense), RF/BLE, tamper loop input, LED.
  - Wraps vendor SDK (e.g., Nordic nRF5 SDK / nRF Connect SDK).
- **Power management module**
  - **Responsibilities:** Sleep/wake policy, radio duty cycling, sensor polling intervals.
  - **Logic:**
    - Deep sleep most of the time.
    - Wake on:
      - RTC tick (e.g., every 1–3 s)
      - Tamper GPIO interrupt
      - External command (if connected)
- **Tag identity & provisioning module**
  - Stores:
    - Unique tag ID
    - Cryptographic keys
    - Manufacturing data
  - Uses secure flash region or key storage.
- **Tamper detection module**
  - Monitors tamper loop GPIO.

    - On loop open:
        - Raise tamper event
        - Immediately send high-priority alarm packet
        - Optionally flash LED pattern
- **RF/BLE communication module**
    - **Beacon mode:**
        - Periodic broadcast with:
            - Tag ID
            - Battery status
            - State flags (normal/tamper)
    - **Connected mode (optional):**
        - For configuration, diagnostics, firmware update.
    - **Security:**
        - Encrypted payloads (AES-CCM or similar)
        - Nonce/counter to prevent replay
- **Battery monitoring module**
    - Periodic ADC read.
    - Thresholds:
        - Normal
        - Low battery (send warning flag)
        - Critical (change beacon interval to conserve power)
- **Event manager**
    - Central dispatcher for:
        - Tamper events
        - Low battery events
        - Config commands
    - Simple state machine: IDLE → NORMAL → TAMPER_ALARM → CONFIG_MODE.

2️⃣ Mother wrist tag — firmware architecture (active version)

Simpler than infant tag.

Modules:

- **Bootloader (optional)**
- **HAL**
- **Power management**
  - Very long beacon interval (e.g., 2–5 s or more).
- **Tag identity module**
  - Stores mother tag ID.
- **RF/BLE communication**
  - Periodic broadcast of ID.
  - Optional pairing handshake with infant tag or gateway.
- **Optional user feedback**
  - LED blink on activity or pairing.

No tamper module unless you decide to add it.

3️⃣ RTLS/RFID ceiling reader node — firmware architecture

Core goals: continuous scanning, zone mapping, reliable uplink.

Main modules:

- **Bootloader**
  - Secure update over Ethernet (TFTP/HTTPS/MQTT-based).
- **HAL / BSP**
  - Ethernet, SPI/UART to RFID module, GPIO, timers, watchdog.
- **RFID/RTLS driver module**
  - Controls RFID reader IC/module:
    - Configure power, frequency, inventory cycles.
    - Read tag EPC/ID, RSSI, antenna port.

    - Tag filtering (e.g., only infant/mother tags).
- **Tag tracking & zone logic**
    - Maps reader/antenna to logical zone.
    - Maintains in-memory table:
        - Tag ID → last seen time, RSSI, zone.
    - Generates events:
        - Tag entered zone
        - Tag left zone
        - Tag not seen for X seconds
- **Network communication module**
    - Protocol: MQTT, WebSocket, or REST over HTTPS.
    - Sends:
        - Tag sightings
        - Health status (uptime, temperature, errors)
    - Receives:
        - Config updates
        - Firmware update triggers
- **Configuration module**
    - Stores:
        - Reader ID
        - Zone mapping
        - Server endpoints
    - Local config via:
        - Web UI (if Linux)
        - Serial CLI (if MCU)
- **Diagnostics & watchdog**
    - Hardware watchdog reset.
    - Periodic self-test of RFID module.

- Log critical errors.

4️⃣ Gate movement recorder terminal — firmware/software architecture

Assuming an embedded Linux SBC (e.g., Raspberry Pi-class).

System layers:

- **OS layer**
  - Linux, systemd services, network manager.
  - Local logging (journald + app logs).
- **Device drivers / low-level interfaces**
  - RFID reader (UART/SPI/I^2C).
  - Barcode/QR scanner (USB/UART).
  - Touchscreen + input.
  - Network (Ethernet/Wi-Fi).
- **Core services (daemons)**
  - **RFID service**
    - Reads infant/mother tags.
    - Normalizes tag IDs.
    - Exposes local API (e.g., Unix socket/REST).
  - **Scanner service**
    - Reads staff IDs from barcode/QR.
    - Validates format.
  - **Gate logic service**
    - Orchestrates:
      - Infant tag + mother tag + staff ID sequence.
    - State machine:
      - IDLE → SCAN_INFANT → SCAN_MOTHER → SCAN_STAFF → VALIDATE → RESULT.
    - Calls backend API:

        - /gate/authorizeMovement
    - Handles responses:
        - Authorized → show green, log event.
        - Denied → show red, sound buzzer.
- **UI service**
    - Touchscreen app (e.g., in Qt/Flutter/Web).
    - Shows:
        - Instructions
        - Scan status
        - Result (Authorized/Denied)
        - Error messages
- **Sync & offline buffer**
    - If backend unreachable:
        - Queue events locally.
        - Retry when online.
    - Local SQLite cache.

- **Security**
    - Mutual TLS to backend.
    - Device certificates.
    - Signed updates.

5️⃣ Alarm controller node — firmware architecture

Core goals: simple, deterministic, highly reliable.

Modules:

- **Bootloader**
    - Simple UART/Ethernet update (optional).
- **HAL**

    - GPIO (relays, LEDs, buttons), timers, communication (UART/Ethernet/Wi-Fi).

- **Communication module**
    - Listens for alarm commands from backend or local gateway:
        - RAISE_ALARM, SILENCE, TEST, RESET.
    - Protocol: MQTT, TCP, or serial.
- **Alarm state machine**
    - States:
        - IDLE
        - ALARM_ACTIVE
        - SILENCED
        - TEST_MODE
    - Transitions triggered by:
        - Remote commands
        - Local key switch/button
- **Output control module**
    - Drives:
        - Siren relay
        - Strobe relay
    - Patterns:
        - Continuous
        - Pulsed (configurable)
- **Safety & watchdog**
    - Hardware watchdog.
    - Fail-safe mode:
        - If comms lost but alarm active → keep siren until explicit reset or timeout.

6️⃣ Footprint scanner — firmware/software architecture

Assuming an embedded Linux SOM.

Layers:

- **OS & drivers**
  - Camera driver (MIPI CSI/USB).
  - Display/touch (if local UI).
  - Network stack.
- **Capture service**
  - Controls camera:
    - Exposure, gain, resolution.
  - Captures raw image on command.
  - Saves image to local storage.
- **Preprocessing & biometric engine**
  - Image normalization (crop, contrast, noise reduction).
  - Feature extraction → biometric template.
  - Template format standardized (for backend matching or local pre-match).
- **Enrollment & verification logic**
  - Enrollment:
    - Capture → generate template → send to backend /biometric/enrollInfant.
  - Verification:
    - Capture → template → send to /biometric/verifyInfant or local match.
- **UI application**
  - Guides nurse:
    - Place foot
    - Hold still
    - Capture success/failure

    - Shows result (enrolled/duplicate/error).
- **Sync & security**
    - TLS to backend.
    - Local cache if offline.
    - Signed software updates.
