SaberAlert MVP

Digital Saber | Pre-Seed | “Ship hardware → provision → learn → alert”

North Star Outcome

A beta user can receive a **push notification within ~5 seconds** when an **untrusted / new presence** appears at their property—after the system learns what “normal” looks like.

If we can’t do that end-to-end, we don’t have a product.

---

1) What SaberAlert Is (Plain English)

SaberAlert is a **camera-free presence alert system**. Our hardware passively listens for Wi-Fi + Bluetooth signals that devices naturally emit (phones, cars, wearables, IoT). The cloud system learns a property’s “pattern of life,” then alerts the homeowner when something **new/unfamiliar** shows up.

---

2) MVP Definition

MVP = Full Lifecycle Works

1. User downloads mobile app and creates account
2. User provisions hardware to Wi-Fi and links it to their account/property
3. Device streams telemetry reliably to cloud
4. System learns baseline (“pattern of life”)
5. System triggers “new presence detected” event
6. User receives push notification reliably
7. User can take action: **Trust / Ignore** and that action affects future alerts
8. Gateway OTA updates are possible (so we can iterate after shipping boards)

Not MVP: persona modeling, stalker scoring, advanced dashboards, perfect UI polish, deep analytics views, multi-property complexity, seeing raw MACs/RSSI.

---

3) Platform Components (Tech Stack + What Each Does)

Hardware / Firmware

- **ESP32 Sniffers (2x)**: capture nearby Wi-Fi/BLE broadcasts (telemetry source)
- **ESP32 Gateway (1x)**: forwards telemetry upstream; supports **OTA updates** (MVP required)

Device Management Plane

- **AWS IoT Core (Things / Certs / Shadows / OTA)**
  - Secure device identity & provisioning metadata
  - Online/offline shadow state + config
  - Gateway OTA workflow (MVP required)

Telemetry Data Plane

- **EMQX (MQTT broker)**
  - Receives device telemetry at high volume
  - Routes to consumers (Watcher, rules)

Real-Time Brain

- **MQTT Watcher (Python)**
  - Subscribes to EMQX topics
  - Normalizes/filters data
  - Performs near real-time analysis to decide when to alert
  - Writes durable outputs (events/aggregates) to Postgres
- **Redis (in-memory)**
  - Fast buffer + state store to enable clustering/fingerprinting without polling
  - Enables “decide fast” (sub-second/seconds)

Long-Lived Storage

- **Neon Postgres**
  - System of record: users/properties/devices/alerts
  - Stores aggregates and summaries (not raw packet firehose)

App/API Layer

- **Backend API (Python)**
  - Authenticated access for mobile/web
  - Device registration & linking
  - Baseline status + device lists + alert history

    - Accepts user actions (Trust/Ignore)
    - Triggers push notifications (or queues them)

Mobile App (Required for MVP)

- **Mobile App (framework TBD)**
    - Sign up / login
    - Add property
    - **Provision device** (Wi-Fi + link to account)
    - Push notifications
    - Minimal views: “Device Online”, “Alert”, “Trust/Ignore”

Observability (Strongly Recommended)

- Logging + metrics + error tracking (Watcher/API/EMQX)
- Required to survive beta without guessing

---

4) MVP Gates (Phases) — With Owners + Exit Criteria

GATE 0 — Lab Validation

Product state: engineering system (internal only)

Goal: prove end-to-end pipeline + alert event in controlled environment.

Owners: Firmware + Backend + DevOps + Analytics/Signal
Exit criteria:

- Telemetry flows Sniffer→Gateway→EMQX→Watcher→Redis→Postgres reliably
- A simulated/controlled “new device” produces an **alert event** consistently
- Watcher restarts safely; no Redis memory runaway in normal test
- Latency measured end-to-end (target ~5s alert generation)

GATE 1 — Mobile Provisioning Complete

Product state: installable system (real user can set it up)

Goal: user can get hardware online and linked to their account/property.

Owners: Mobile (primary) + Firmware (pairing) + Backend (device linking) + DevOps (infra)
Required user flow:

1. Create account → 2) Add Property → 3) Add Device → 4) Provision Wi-Fi → 5) "Device Online"

Exit criteria:

- A non-technical beta tester can complete setup in **<10 minutes** without help
- Device ends up **securely associated** to the correct user/property
- Online/offline visible in app (or at least "last seen")

GATE 2 — Baseline Learning ("Pattern of Life")

Product state: learning product (quietly building normal)

Goal: establish baseline to reduce false positives.

Owners: Analytics/Signal (primary) + Backend + DevOps
Behavior:

- System enters "Learning Mode" for a defined window (recommend 3–7 days)
- Builds list of "normal/trusted candidates" and filters drive-bys

Exit criteria:

- System can identify high-frequency home devices vs noise
- Drive-by traffic suppressed to a tolerable level
- Baseline completion logic is explicit and visible ("Learning Mode complete")

GATE 3 — Real-Time Alert MVP

Product state: true beta product

Goal: phone buzzes for unfamiliar presence.

Owners: Analytics + Backend + Mobile + DevOps (all hands)
Required behaviors:

- New/untrusted presence → alert event → push notification
- User can Trust/Ignore and it changes future alerting behavior

Exit criteria (beta reality check):

- In 10–20 beta homes, alerts are mostly accurate and understandable
- Push reliability is high (you can't "sometimes" deliver alerts)

- Latency target remains near real-time (goal ~5 seconds)

GATE 4 — Post-MVP Intelligence Layer (Not required to fundraise MVP)

Product state: differentiated product

Adds: recurring unknowns, personas, friend/neighbor tagging, “seen 3 times this week”, later-night recurrence patterns, etc.

---

5) MVP Deliverables (What the Team Must Produce)

A) Mobile App (Minimum Screens)

- Auth (signup/login/forgot password)
- Add property
- Add device
- **Provisioning** (BLE or SoftAP)
- Device online indicator
- Alert screen + Trust/Ignore
- Push notifications integration

B) Firmware

- Provisioning handshake compatible with app
- Stable sniffer capture
- Gateway forwarding + reconnect logic
- **Gateway OTA** (MVP required)
- Basic local debug + remote logging hooks

C) Backend/API

- User/property management
- Device registration + secure linking
- Online/offline state
- Alert creation endpoint
- Trust/Ignore writeback (affects model)
- Baseline status endpoint (“learning mode”)

D) Watcher + Analytics

- Real-time filtering + clustering pipeline
- Fingerprint logic for randomized identifiers
- Baseline learning logic
- “New/untrusted presence” detection policy
- Alert triggering rules

E) DevOps/Platform

- Hosted EMQX + topics/ACLs
- Hosted Watcher + autos/restart policies
- Hosted API
- Redis sizing + persistence strategy if needed
- Neon Postgres
- Secrets management
- Monitoring + logs + latency metrics
- OTA hosting pipeline for gateway

F) Beta Ops (Business Track)

- 20 beta homes recruited
- Simple install guide + troubleshooting page
- Feedback collection loop (survey + tagged logs)
- Demo videos from real installs