Full detailed plan — ESP32 + RFID attendance system for a multi-company incubation centre

Nice. Below is a thorough, practical blueprint you can follow (or I can implement for you). It covers hardware, firmware, backend, database, API, dashboard, security, deployment, testing, operations, costs, and rollout. Think of this as the project's map — annotated, pragmatic, and deliberately unromantic about bugs.

Summary (one-line)

Build networked ESP32 RFID readers that POST authenticated scan events to a central backend (Postgres/Supabase or self-hosted) and a web dashboard where the incubation head and company admins manage companies, employees, tags and reports.

Big picture architecture

- **Edge**: ESP32 + RFID (MFRC522 or PN532) per entry point or mobile kiosk. Optional OLED, buzzer, LEDs. Devices hold a device token and can buffer events when offline.
- **Network**: Wi-Fi (NTP optional). HTTPS to backend.
- **Backend**: REST/GraphQL API + Auth, business logic, device registry, ingestion pipeline, web sockets for live feed.
- **DB**: Postgres (or Supabase managed Postgres).
- **Frontend**: React (Vite/Next) dashboard for incubation head & company admins.
- **Auth**: JWT for users; device tokens for ESP32. RBAC enforcement in backend.
- **Infra**: Dockerized services, run on cloud (DigitalOcean/Render/AWS/GCP) or managed (Supabase + Vercel).
- **Observability**: Heartbeats, device status, logs, backups, audit.

Milestones & timeline (realistic plan you can execute yourself)

Assume moderate experience; parallel work possible. Adjust to team size.

1. **M0 — Planning & procurement (1 week)**
   - Finalize requirements (num devices, always-online vs intermittent).
   - Buy hardware (ESP32 boards, RFID readers, power supplies).
   - Choose backend hosting (Supabase vs self-hosted).
2. **M1 — Minimal working prototype (1–2 weeks)**
   - ESP32 reads RFID and posts to a simple test endpoint.
   - Simple backend endpoint that stores logs.
   - Quick dashboard showing live feed.
3. **M2 — Core features & robustness (2–3 weeks)**
   - Full DB schema, device registration flow, JWT users.
   - Debounce, offline buffer, retry logic on device.
   - Company/employee CRUD in dashboard.
4. **M3 — Production hardening (2–3 weeks)**
   - TLS, token rotation, password policies, backups.
   - Heartbeats, device last_seen, monitoring.
   - CSV exports, basic reports.
5. **M4 — Advanced features & ops (2–4 weeks)**
   - OTA for devices, firmware versioning.
   - Payroll-friendly reporting (IN/OUT inference).
   - Role delegation, device provisioning via QR.

6. **M5 — Rollout & training (1–2 weeks)**
   - Onboard companies, bulk import employees, admin training.
   - Stress testing number of devices and events.

Total: **~8–12 weeks** for a polished system if one full-time dev; faster if you parallelize.

Hardware & cost estimate (per reader)

- **ESP32 dev board**: ₹300–₹800 / $3–$8
- **MFRC522 RFID module**: ₹80–₹300 / $1–$4 (cheap, works for cards/tags)
- **Optional PN532** (NFC, better range/compat): ₹600–₹1500 / $8–$20
- **Buzzer / LEDs / small OLED**: ₹100–₹400 / $1–$5
- **Power supply (5V 2A)**: ₹300–₹800 / $3–$8
- **Enclosure, wiring**: ₹200–₹700 / $2–$9

Estimate per deployed reader: **₹1,000–₹3,000** depending on extras.

Database schema (detailed)

Use Postgres. Below are essential tables; add indices on frequent query columns.

companies

```
CREATE TABLE companies (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  code TEXT UNIQUE, -- optional short code
  is_active BOOLEAN DEFAULT TRUE,
  created_at timestamptz DEFAULT now()
);
```

employees

```
CREATE TABLE employees (
  id SERIAL PRIMARY KEY,
  company_id INT REFERENCES companies(id) ON DELETE CASCADE,
  full_name TEXT NOT NULL,
  email TEXT,
  phone TEXT,
  employee_code TEXT,  -- optional employee id
  is_active BOOLEAN DEFAULT TRUE,
  created_at timestamptz DEFAULT now()
);
```

rfid_tags

```
CREATE TABLE rfid_tags (
  id SERIAL PRIMARY KEY,
  uid TEXT NOT NULL UNIQUE,
  employee_id INT REFERENCES employees(id) ON DELETE SET NULL,
  note TEXT,
  assigned_at timestamptz DEFAULT now()
);
```

devices

```
CREATE TABLE devices (
  id SERIAL PRIMARY KEY,
  device_uuid TEXT UNIQUE NOT NULL,
  device_name TEXT,
  company_id INT REFERENCES companies(id),
  location TEXT,
  firmware_version TEXT,
  last_seen timestamptz,
  is_active BOOLEAN DEFAULT TRUE,
  created_at timestamptz DEFAULT now()
);
```

attendance_logs

```
CREATE TABLE attendance_logs (
  id BIGSERIAL PRIMARY KEY,
  device_id INT REFERENCES devices(id),
  company_id INT REFERENCES companies(id),
  employee_id INT REFERENCES employees(id),
  tag_uid TEXT,
  event_type TEXT, -- 'SCAN' / 'IN' / 'OUT' or freeform
  recorded_at timestamptz DEFAULT now(),
  raw_payload JSONB
);
CREATE INDEX ON attendance_logs (company_id, recorded_at);
CREATE INDEX ON attendance_logs (employee_id, recorded_at);
```

users

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username TEXT UNIQUE,
  password_hash TEXT,
  full_name TEXT,
  role TEXT, -- 'incubation_head','company_admin','technician'
  company_id INT, -- nullable for global roles
  created_at timestamptz DEFAULT now()
);
```

device_tokens

```
CREATE TABLE device_tokens (
  id SERIAL PRIMARY KEY,
  device_id INT REFERENCES devices(id) ON DELETE CASCADE,
  token TEXT UNIQUE,
  created_at timestamptz DEFAULT now(),
  last_used timestamptz
);
```

API design (core endpoints & behavior)

Use HTTPS. Devices authenticate with Bearer token from `device_tokens`. Admin users use JWT from `/auth/login`.

Device endpoints

- `POST /api/v1/devices/register` — register new device (one-time provisioning). Body: `{device_uuid, secret}` → returns device_token.

- `POST /api/v1/device/event` — ingest event. Headers: `Authorization: Bearer <device_token>`. Payload:

```
{
  "device_uuid":"esp-01",
  "tag_uid":"AB:CD:12:34",
  "rssi": -37,
  "antenna": 1,
  "timestamp":"2025-10-29T09:00:00Z",
  "raw": {"extra":"value"}
}
```

- `POST /api/v1/device/heartbeat` — periodic heartbeat updating `last_seen`.

Admin endpoints

- `POST /api/v1/auth/login` → returns user JWT.

- `GET /api/v1/companies` (RBAC enforced).

- `POST /api/v1/companies`

- `POST /api/v1/employees` / `PUT /api/v1/employees/:id`

- `POST /api/v1/tags/assign` — assign scanned tag to employee via a short flow: scan tag in UI to capture UID then link to employee.

- `GET /api/v1/attendance?company_id=&from=&to=&employee_id=&limit=` — with pagination.

Realtime

- `GET /api/v1/live` via web sockets or SSE to stream last N events.

Responses
Return structured errors with codes. Rate-limit device endpoints to prevent floods. Validate timestamps and prefer server time for canonical logs.

Device firmware design (ESP32)

High-level features:

- Wi-Fi connect (support multiple SSIDs).
- Read UID from RFID module.
- Debounce same UID within configurable interval (e.g., 3–5 sec).
- Attach meta: device_uuid, optional local timestamp.
- Send to backend with `Authorization: Bearer <device_token>`.
- On network failure: append event to circular queue in LittleFS/SPIFFS. Retry periodically. Persist across reboots.
- Heartbeat every configurable interval (e.g., 5 minutes) with firmware_version and free space.
- OTA update support (HTTP or MQTT-based).

Key modules in firmware:

- WiFi manager (auto-reconnect).
- RFID driver (MFRC522 or PN532).
- HTTP client with TLS (validate certificate or pin).
- Storage manager (LittleFS).
- Retry/queue manager.
- LED/buzzer feedback manager.

Minimal pseudocode flow:

1. Connect Wi-Fi.

2. Start heartbeat timer.
3. Poll RFID; on tag read → if not duplicate → create event → push to send queue.
4. Worker thread attempts to flush queue: HTTP POST; on success remove item; on failure keep and exponential backoff.
5. Periodic heartbeat POST.

I can provide a tested Arduino/ESP-IDF sketch with offline buffering and OTA on request.

Security & provisioning

- **Device provisioning**: create device record in backend; generate long random token printed on sticker/QR; admin enters in device during setup (or device enters secret once to exchange for token).
- **Transport**: HTTPS (TLS). Use certificate pinning if possible.
- **Auth**: Devices use tokens; admins/users use JWTs from secure login.
- **Secrets**: store device tokens hashed (e.g., HMAC) or at least encrypted in DB.
- **RBAC**: endpoints that modify company data check `company_id` matches user's scope.
- **Rate-limiting & validation**: prevent event floods; validate UID format.
- **Audit logs**: store who changed tags/employees and when.
- **Rotation**: allow revocation/rotation of device tokens.
- **Passwords**: bcrypt or Argon2; enforce strong password policy.

UI/UX and features (dashboard)

Pages & components:

- **Login** (Incubation head, company admins)

- **Overview**: live feed, device status, recent alerts
- **Companies**: create / archive / bulk upload
- **Employees**: CRUD, bulk import (CSV), assign tag button (scan in UI)
- **Tags**: list of unassigned tags, transfer tags
- **Devices**: register, view last seen, firmware, location; QR provisioning flow
- **Reports**: custom date ranges, export CSV/XLSX, IN/OUT inference
- **Audit**: change history
- **Settings**: debounce time, timezone, retention policy
- **Notifications**: email/SMS alerts (optional) for missing check-ins

UX patterns:

- To assign a tag: admin clicks “Assign tag”, UI opens small modal showing “Waiting for scan” and the admin taps the physical tag on a test reader connected to the admin machine or uses QR from device.
- Live feed should show employee photo (optional), tag UID, device location, and company.

Business rules / attendance semantics

Decide whether system records raw scans or interprets IN/OUT:

- **Raw logs**: simplest — every scan is a record.
- **IN/OUT inference**: infer by time of day or by pairing consecutive scans per employee. Implementation note: ambiguous (e.g., multiple entries in/out). Provide UI for manual correction.
- Recommendation: store raw logs and implement inference/labels in reporting layer. Add rules like “first scan after 06:00 counts as IN”, “last before 23:59 counts as OUT” — customizable per company.

Edge cases & operational notes

- **Shared tags**: allow marking tags as shared; add comment system.
- **Lost tags**: provide revoke & reissue functionality.
- **Time sync**: rely on server time for canonical timestamps. Devices send local timestamp only as hint.
- **Offline devices**: show `last_seen` and pending buffered count (if device reports it).
- **Duplicate UID collisions**: warn if same UID assigned to multiple employees.
- **Tag cloning**: physical cards can be cloned; for high security use stronger card tech (MIFARE DESFire) and change credentials periodically.

Reliability, scaling & monitoring

- **Throughput**: each device might produce 1–3 events per minute at most. Postgres + small Node/FastAPI server handles thousands/day easily. Use connection pooling.
- **Horizontally scale**: stateless API, load balancer, multiple worker instances and separate background worker (for reports, exports).
- **Monitoring**: Prometheus + Grafana or simple log-based metrics. Track device heartbeats and alert for missing heartbeats > threshold.
- **Backups**: daily DB backups, retention 30 days (or per policy).
- **SLA considerations**: design device buffering for intermittent networks.

Deployment options (recommendations)

- **Fastest / lowest maintenance**: **Supabase** (DB + Auth) + Vercel for frontend. You still need a small serverless function or edge function to validate device tokens and process events (or use Supabase Edge Functions).
- **More control**: Dockerized FastAPI/Express + Postgres on DigitalOcean/Render; Vercel/Netlify for frontend.

- **Enterprise / on-prem**: host Postgres on your VM or company cloud region; expose API via VPN / secure firewall.

CI/CD & repository layout

- Monorepo with: `firmware/`, `backend/`, `frontend/`, `infra/`.
- Backend: Dockerfile, GitHub Actions for tests & build, deploy to Render.
- Frontend: Vite + React + Tailwind, GH Actions → Vercel.
- Firmware: versioned releases, tags mapping to device firmware_version.

Testing plan

- **Unit tests** for backend endpoints, DB migrations.
- **Integration tests**: simulate device events to ingestion endpoint, assert DB rows.
- **E2E user tests**: login, create company, create employee, assign tag, scan emulated tag.
- **Load test**: simulate N devices sending M events/min to verify throughput.
- **Device acceptance tests**: power cycle, Wi-Fi dropout, buffer flush.

Example queries & reports

- **Daily attendance for a company**

```
SELECT e.employee_code, e.full_name,
  MIN(a.recorded_at) FILTER (WHERE (a.recorded_at::time) BETWEEN
'06:00' AND '12:00') AS first_scan,
  MAX(a.recorded_at) FILTER (WHERE (a.recorded_at::time) BETWEEN
'12:00' AND '23:59') AS last_scan
```

```
FROM attendance_logs a
JOIN employees e ON e.id = a.employee_id
WHERE a.company_id = $1
  AND a.recorded_at::date = $2
GROUP BY e.id, e.employee_code, e.full_name
ORDER BY e.full_name;
```

- **Find unassigned tags**

```
SELECT uid, created_at FROM rfid_tags WHERE employee_id IS NULL
ORDER BY created_at DESC;
```

Security checklist (must-do before production)

- Enforce HTTPS (TLS) everywhere.
- Use strong device token generation and rotation.
- Store passwords with bcrypt/argon2.
- Use RBAC checks in server code (never trust client).
- Rate-limit device endpoints and admin endpoints.
- Audit log critical changes (tag reassign, employee deletion).
- Limit sensitive data retention and document retention policy.

Optional advanced features (roadmap)

- **Face/photo matching** on top of RFID (for anti-fraud).
- **Mobile app** for supervisors to approve corrections.
- **Geofencing** for mobile readers using GPS.
- **BI & analytics** for trends, churn, presence heatmaps.

- **Multi-factor device provisioning** using QR + one-time PIN.
- **SMS/Email alerts** for no-show or anomalies.

Deliverables I can produce immediately (pick by replying with exact name)

1. **ESP32 Firmware** — Production-ready Arduino sketch with MFRC522, LittleFS buffering, heartbeat, retry, and example provisioning flow.
2. **Backend Template** — Node/Express (or FastAPI) app with device auth, endpoints, DB migrations (SQL), and tests.
3. **React Dashboard Scaffold** — Vite + React + Tailwind app with login, live feed, company/employee CRUD, assign tag modal, and CSV export.
4. **Supabase End-to-End Demo** — Supabase project SQL, Edge Function for device ingestion, and minimal frontend mocking device events.
5. **Full Pack** — Firmware + Backend + Frontend + Deployment scripts + docs (largest; will be delivered as a complete repo).

Reply with the exact option name above and I'll generate that artifact next (code, migrations, or full repo scaffolding).

Quick checklist to start buying & prepping hardware right now

- For each entry point: ESP32 dev board + MFRC522 + 5V power supply + enclosure.
- One test bench: ESP32, RFID, USB serial for debugging.
- A spare phone or laptop to act as temporary reader and admin UI.
- A small USB-to-TTL and jumper wires.

This plan gives you the full roadmap and actionable pieces — production thinking included: security, backups, offline, and how to interpret scans. When you say which deliverable you want first (ESP32 firmware / Backend Template / React Dashboard / Supabase Demo / Full Pack), I'll produce it straight away with code, SQL, and instructions.

Ready to turn roll-call into a reliable logbook.