Android-first Option A MVP — Architecture & Sequence Flows

Contents: high-level architecture diagram, sequence flows (registration, class start, attendance detection), API definitions, DB schema sketch, security & anti-fraud design, permission & privacy checklist, testing/deployment notes, and recommended next steps.

---

1. High-level architecture

Components: - Teacher Android App (advertiser) - Student Android App (scanner) - Backend Server (Auth, Session, Token Issuance, Attendance API) - Database (Postgres) - Optional: Admin Web Portal (create classes/schedule, view logs) - Optional: Dedicated BLE Beacons (fallback for iOS or hard-to-support devices)

Flow (summary): 1. Teacher starts a class session from their app (or admin portal) → server issues a session token for that classroom and time window. 2. Teacher app advertises BLE payload containing classId + short-lived sessionToken + signature. 3. Student app scans in background; on seeing valid payload, it collects local evidence (biometric confirmation, device registration, optional location snapshot), then POSTs attendance to server. 4. Server validates token, checks device binding and biometric proof, records attendance and returns confirmation.

---

2. Sequence flows (detailed)

A. One-time onboarding (student)

1. Student installs Student App.
2. Student opens app → fills: name, college ID, phone number.
3. App requests OTP to phone → server verifies OTP → marks phone number registered.
4. App runs device registration: generate deviceId (UUID) stored in secure storage; send device public key (if using asymmetric auth) to server.
5. Student registers biometric locally (no biometric data sent to server) — server stores user -> device mapping.
6. Optionally request initial permissions: Location (fine), Bluetooth, Background location (Android), Notifications.

Server stores: userId, phoneNumber, deviceId, devicePublicKey, registrationTimestamp, allowedDevices[], profilePhoto(optional).

B. Teacher starts class session

1. Teacher app requests new session from server: `POST /sessions` {teacherId, classId, startTime, duration}.
2. Server creates session record, generates `sessionToken` (random nonce + expiry) and signs it with server private key or returns token to teacher app for on-device broadcasting.

3. Teacher app begins BLE advertising payload: `{classId, sessionToken, signature}`. Payload rotates/refreshes every 20s.

Server session record: sessionId, classId, teacherId, sessionTokenHash, issuedAt, expiresAt, status.

C. Student detection & attendance marking

1. Student app's background scanner discovers BLE advertisement matching expected classId or session pattern.
2. App reads payload, validates signature quickly if signature key is known, then:
3. Prompt biometric auth (BiometricPrompt) or use fast local auth flow.
4. Capture a small location snapshot (lat, lon, accuracy) if permission granted.
5. Build attendance payload: `{userId, deviceId, sessionToken, sessionTimestamp, location, beaconRssi, deviceFingerprintHash, optional selfieHash}`.
6. POST `/attendance` to server with JWT auth or signed request (using device private key). Include the sessionToken as presented.
7. Server verifies:
8. sessionToken is active and matches sessionId and teacherId, not expired, not replayed.
9. deviceId is registered to userId.
10. deviceFingerprint and SIM-phone match registration.
11. optional: location check within campus boundary (loose) and beaconRSSI consistent with being near the teacher.
12. check for replay/reuse: token nonce not previously consumed for this device/user.
13. On success, server records attendance row: `{attendanceId, sessionId, userId, deviceId, timestamp, location, evidenceRef}`; returns 200 OK.
14. Student app shows success UI (green tick) and caches proof locally for offline sync.

---

3. API Design (sketch)

Auth - `POST /auth/register` — register user + device + phone OTP - `POST /auth/login` — issue JWT for app session (short lived)

Sessions - `POST /sessions` — teacher requests session token (auth: teacher) - `GET /sessions/{id}` — get session metadata

Attendance - `POST /attendance` — {userId, deviceId, sessionToken, location, biometricProof, rssi, selfieHash} - `GET /attendance?sessionId=` — list attendance (admin)

Device management - `POST /devices/verify-sim` — optional telecom API integration - `POST /devices/revoke` — admin revoke device

---

4. Database schema (simplified)

users: id, name, collegeId, phone, email, profilePhoto, createdAt

devices: id (deviceId), userId, devicePublicKey, registeredAt, lastSeenAt, phoneNumberHash

sessions: id, classId, teacherId, sessionTokenHash, issuedAt, expiresAt, status

attendance: id, sessionId, userId, deviceId, timestamp, locationLat, locationLon, rssi, evidenceJson, synced

audit_logs: id, eventType, payload, createdAt

---

5. Security & anti-fraud measures

- **Short-lived signed tokens**: sessionToken has expiry (20–60s) and server signature to prevent forging.
- **Nonce consumption / replay protection**: server marks token usage per device/user to prevent replay.
- **Device binding**: deviceId registered during onboarding; pairing uses OTP + device key.
- **SIM verification**: use SMS Retriever, or telecom API for stronger binding.
- **Biometric**: local biometric confirmation; server gets signed assertion or just rely on user action + JWT.
- **Cross-checks**: RSSI thresholds, occasional selfie + liveness check for suspicious cases, IP/geolocation cross-validation.
- **Rate limiting & anomaly detection**: repeated scans from same token from many devices flagged.

---

6. Permissions & privacy checklist

- Explain why each permission is needed (Bluetooth, Location, Background Location, Camera for selfie, SMS receive for OTP).
- Obtain explicit consent and show privacy policy before onboarding.
- Store minimal PII; do not store raw biometric data; only store device public keys and tokens.
- Allow users to view their attendance logs; provide admin access audits.

---

7. iOS support & fallbacks

- iOS background BLE is limited. Options:
- Use dedicated BLE beacons in classrooms (hardware) instead of teacher device advertising.
- Ask iOS students to keep the app in foreground during class or use QR fallback.
- Provide rotating QR as fallback for any platform.

---

8. Testing considerations

- Test with many devices concurrently near a single advertiser (stress RSSI handling).
- Test token expiry and replay attack scenarios.
- Battery/performance test: tune scan windows and advertise intervals to balance reliability & battery.
- Test offline scenarios: student caches attendance proof and syncs later; server validates token lifetime and accepts offline evidence if signature present.

9. Deployment & ops notes

- Use HTTPS for all server endpoints; host private key securely (HSM if possible).
- Monitor attendance events, token issuance rates, and anomalies.
- Provide OTA updates for apps; internal distribution for early pilot.

---

10. Recommended MVP roadmap

1. **Phase 0 (Design)**: finalize session token format, DB schema, and security model.
2. **Phase 1 (Android MVP)**: teacher app (advertise), student app (scan + biometric + POST attendance), backend endpoints for sessions & attendance. Use simple signed tokens issued by server.
3. **Phase 2 (Admin & Audit)**: build admin portal and logs, implement SIM verification, improve UX.
4. **Phase 3 (Scale & iOS support)**: add iOS support or classroom hardware beacons, anomaly detection, polish.

---

11. Appendix — Example payloads

Session token issuance response

```
{
  "sessionId": "sess_abc123",
  "sessionToken": "eyJhbGciOi...",
  "expiresAt": "2025-09-21T10:30:00Z"
}
```

BLE payload (compact)

```
{ classId: "CSE101", token: "ey...", sig: "base64sig" }
```

Attendance POST

```
{
  "userId": "u_123",
  "deviceId": "d_456",
  "sessionToken": "ey...",
  "timestamp": "2025-09-21T10:05:12Z",
  "location": { "lat": 25.68, "lon": 85.21, "accuracy": 15 },
  "rssi": -56,
  "deviceFingerprint": "sha256(...)",
  "selfieHash": "sha256(...)"
}
```

If you want, I can now generate: - a **sequence diagram image** (PNG) of the flows, or - **sample Android code** for BLE advertising (teacher) and BLE scanning (student), or - the **full API spec (OpenAPI/ Swagger)** for the backend.

Pick one and I'll produce it next.