EMIR Android App — Software Design Document (SDD)

# 1. Purpose The app supports three usage scenarios: 1) Standalone GSM-capable watches send data directly to the server without needing this phone app. 2) Watches without GSM use the phone app as a relay for data upload and push. 3) If no watch is connected, the phone app continues to gather minimal location/activity data independently. The EMIR Android app is fully supported on Android phones and Android tablets (pads). Tablets can operate as fallback devices when no watch is connected, with full location and activity tracking, push notifications, and secure data uploads.

The EMIR Android app is part of the EMIR wearable tracking ecosystem. It collects location, activity, and health data from the phone or connected watch, uploads it securely to the EMIR backend, and receives commands via push notifications.

# 2. Platform & Tech Stack

• Language: Kotlin  
• Location: FusedLocationProvider  
• Activity: ActivityRecognition API  
• Health: Health Services API (optional) or companion BLE sensors  
• Background Tasks: WorkManager  
• Push: Firebase Cloud Messaging (FCM)  
• Secure Storage: Android Keystore  
• Minimum SDK: API Level 24 (Android 7.0 and up)  
• Target SDK: Latest available (API Level 34 or above)  
• IDE: Android Studio + Gradle  
  
• Compatible with large-screen tablets (screen size large/xlarge).

# 3. Key Features

• Smart location tracking based on motion & distance thresholds.  
• Activity detection (walking, running, driving).  
• Optional heart rate monitoring.  
• Batch upload with WorkManager respecting Doze mode.  
• Receive commands via FCM push.  
• Secure API authentication and local token storage.  
• Local queue with Room DB for offline mode.  
  
• Supports fallback mode when no watch is connected.  
• Supports direct watch-to-server data if watch has GSM/LTE (phone app not required).  
• Supports tablets as fallback tracking devices with same features as phones.

# 4. Main Components

• LocationService: Uses FusedLocationProvider for efficient GPS.  
• ActivityService: Uses ActivityRecognition.  
• HealthMonitor: Reads heart rate if supported.  
• DataUploader: Batches and uploads via HTTPS.  
• PushReceiver: Handles FCM push events.  
• Local Storage: Room DB for offline telemetry.  
• Security: Uses Android Keystore for tokens.  
  
• WatchConnectivityHandler: Monitors if a Wear OS watch is connected; switches to fallback mode if not.

# 5. Push Notifications

• Uses Firebase Cloud Messaging (FCM).  
• Registers device token, uploads to backend.  
• Backend sends minimal push payload (command\_id).  
• App wakes, downloads full command via secure API.

# 6. Security

• HTTPS (TLS 1.2+) for all API calls.  
• HSTS enforced.  
• Optional certificate pinning.  
• API keys/JWT stored in Android Keystore.  
• Push payloads minimal — no raw health or location data.

# 7. Battery Saving Strategy

• Location updates triggered by motion/distance thresholds.  
• Uses WorkManager for periodic batch uploads.  
• No persistent sockets — push notifications only.  
• Compliant with Android Doze & background execution limits.  
  
• Fallback mode respects Doze/low-power if no watch is connected.  
• Fallback mode works the same on tablets as on phones.

# 8. Data Fields

• Location: timestamp, lat, lon, accuracy, activity  
• Heart Rate: timestamp, bpm, anomaly flag  
• Activity: type (walk/run/drive)  
• Commands: command ID, status

# 9. Build & Deployment

• Android Studio + Gradle.  
• Signing config with Keystore.  
• FCM configuration with google-services.json.  
• API base URLs configurable per environment.

# 10. Future Extensions

• BLE heart rate sensor pairing.  
• Local map view for user tracking.  
• Sync with Wear OS watch for fallback data.