Software Design Document (SDD)

# Project Title:

EMIR: Cross-Platform Wearable Data Gathering & Command System

# 1. Purpose

This Software Design Document (SDD) describes the architecture, components, data flow, and implementation details for a cross-platform system that collects location and health data from wearable devices (Apple Watch, Wear OS), iOS and Android phones, and provides a web-based dashboard for team monitoring and commands. Commands and messages are delivered using APNs for Apple and FCM for Android.

# 2. System Overview

The system consists of:  
- Native watch apps for watchOS and Wear OS  
- Native phone apps for iOS and Android  
- A React-based web dashboard  
- A secure backend server handling APIs, data storage, and push notifications  
The apps gather location, health data (heart rate), and activity data (steps/distance) and send them in batches to the server. Admins can send commands or messages which are delivered via push notifications.

# 3. System Architecture

The system architecture includes:  
Clients:  
• watchOS app: Uses CoreLocation, HealthKit, BackgroundTasks.  
• Wear OS app: Uses FusedLocationProvider, Health Services API.  
• iOS app: Companion app for permissions and fallback.  
• Android app: Same as iOS.  
Server:  
• API Gateway for REST endpoints.  
• Data Processor for validation and storage.  
• Push Service for APNs and FCM integration.  
• PostgreSQL database with PostGIS.  
Web Dashboard:  
• React frontend with Mapbox/Leaflet for location display.

# 4. Data Flow

Data Upload:  
- Watches/phones collect data and batch it by time or events.  
- Data is sent to the server via HTTPS POST.  
  
Receiving Commands:  
- Admin uses the web dashboard to send a command.  
- The server pushes the command using APNs (Apple) or FCM (Android).  
- The client app wakes up, retrieves any needed files, and displays to the user.

# 5. REST API Endpoints

POST /api/v1/data - Upload batched location, health, and activity data.  
POST /api/v1/command - Admin sends a text/audio command.  
GET /api/v1/map - Retrieve user location tracks.  
GET /api/v1/health - Retrieve health logs.  
GET /api/v1/command/{id} - Download command content.

# 6. Database Design

Users: user\_id, device\_id, push\_token, last\_seen  
Location Logs: user\_id, timestamp, lat, lon, activity\_type  
Health Logs: user\_id, timestamp, heart\_rate, anomaly\_flag  
Commands: command\_id, user\_id, text/audio\_url, status, timestamp

• All data transfer over HTTPS.  
• Health/location data stored encrypted.  
• Push payloads contain minimal data.  
• GDPR compliant with opt-in, data deletion, and audit controls.

# 8. Battery Saving Strategy

• Location updates based on motion/distance thresholds.  
• Heart rate uploads only for anomalies or intervals.  
• Batch uploads to reduce network use.  
• Use APNs/FCM only, no persistent sockets.  
• Respect iOS/Android background task policies.

# 9. Technology Stack

• Database: PostgreSQL with TimescaleDB (time-series) and PostGIS (geo)

# 10. MVP Scope

• Basic data gathering from watches.  
• Periodic batch upload.  
• Secure backend with API.  
• Basic web dashboard with map.  
• Admin command send & push delivery.

# 11. Time-Series and Geo Data Strategy

For this project, TimescaleDB will handle high-volume time-series storage (location, heart rate) and PostGIS will provide advanced geospatial indexing and queries (e.g., radius search, bounding box, geo joins). This ensures scalable writes, fast reads for map queries, and future-proof local/cloud deployment.

# 12. Example Geo Query

Example: Find all user locations within 500 meters of a point in the last 24 hours:  
SELECT \* FROM location\_logs  
WHERE user\_id = 'abc123'  
AND timestamp > now() - interval '24 hours'  
AND ST\_DWithin(geom::geography, ST\_MakePoint(target\_lon, target\_lat)::geography, 500);

# 13. Push Notification System Design

The system uses Apple Push Notification Service (APNs) and Firebase Cloud Messaging (FCM) to deliver commands and messages to watches and phones with minimal battery impact and secure delivery.  
Device tokens are registered at install time and stored securely. Commands are triggered by admin actions in the React dashboard and delivered via APNs/FCM over TLS. Push payloads contain only minimal IDs; full command content is fetched over HTTPS after wake-up.  
All push integrations follow APNs HTTP/2 API and FCM HTTP v1 API best practices.

# 14. Data Security & Encryption

Data in Transit:  
- All communication between devices and the server uses HTTPS (TLS 1.2+ or TLS 1.3).  
- HTTP Strict Transport Security (HSTS) is enforced.  
- Certificate pinning can be added for extra protection.  
  
Data at Rest:  
- The database uses disk-level encryption (e.g., LUKS, cloud disk encryption) for physical data security.  
- No field-level or column-level encryption is used to keep time-series writes fast.  
- Backups are stored encrypted.  
  
Push Notifications:  
- Commands/messages are delivered using APNs (for Apple) and FCM (for Android).  
- Push payloads contain only minimal IDs; no raw health/location data is sent in push content.  
  
Authentication:  
- Devices authenticate with unique API keys or JWTs.  
- Secrets are stored securely in device OS keychains.  
  
Privacy:  
- GDPR-ready consent and opt-out flows.  
- Only operationally required telemetry is retained.