PulsePal Project Report

Prepared By:

Kirubel Temesgen

Programme:

BSc Hons in Software Development, SETU Carlow

Project:

PulsePal – a real-time fitness app

Supervisor:

Dr. Joseph Kehoe

1. Introduction

Maintaining effective and safe exercise routines requires timely feedback on our effort, pace and overall load. PulsePal is an Android application that wirelessly connects to a MoveSense wearable sensor to display live heart-rate, step count, distance, pace and calorie burn while you train. By visualising these metrics in real time—and allowing you to set in-session goals—PulsePal helps beginners and seasoned runners alike make data-driven adjustments on the fly.

This report details the motivation behind PulsePal, the design decisions taken, its architecture and implementation, the testing carried out, and opportunities for future enhancement. The core aim is to empower users with intuitive insights into their workout intensity, helping them train smarter, avoid overexertion, and track progress over time.

2. Project Objectives

The PulsePal application was designed with the following core objectives in mind:

1. **Real-Time Feedback**
   • Continuously capture and display heart rate, step count, distance, pace and calories during an exercise session.
   • Ensure minimal latency between the sensor reading and its presentation in the UI.

2. **In-Session Goal Setting**
   • Allow users to define up to three performance targets (e.g. maintain ≥ 140 bpm for 5 minutes, run 1 km, burn 100 kcal).
   • Visually track progress toward each goal via progress bars and percentage feedback.

3. **Background Reliability**
   • Keep the chronometer and data streams running even when the app is in the background or the device screen is off.
   • Safeguard against data loss on temporary app pauses, phone lock or accidental navigation.

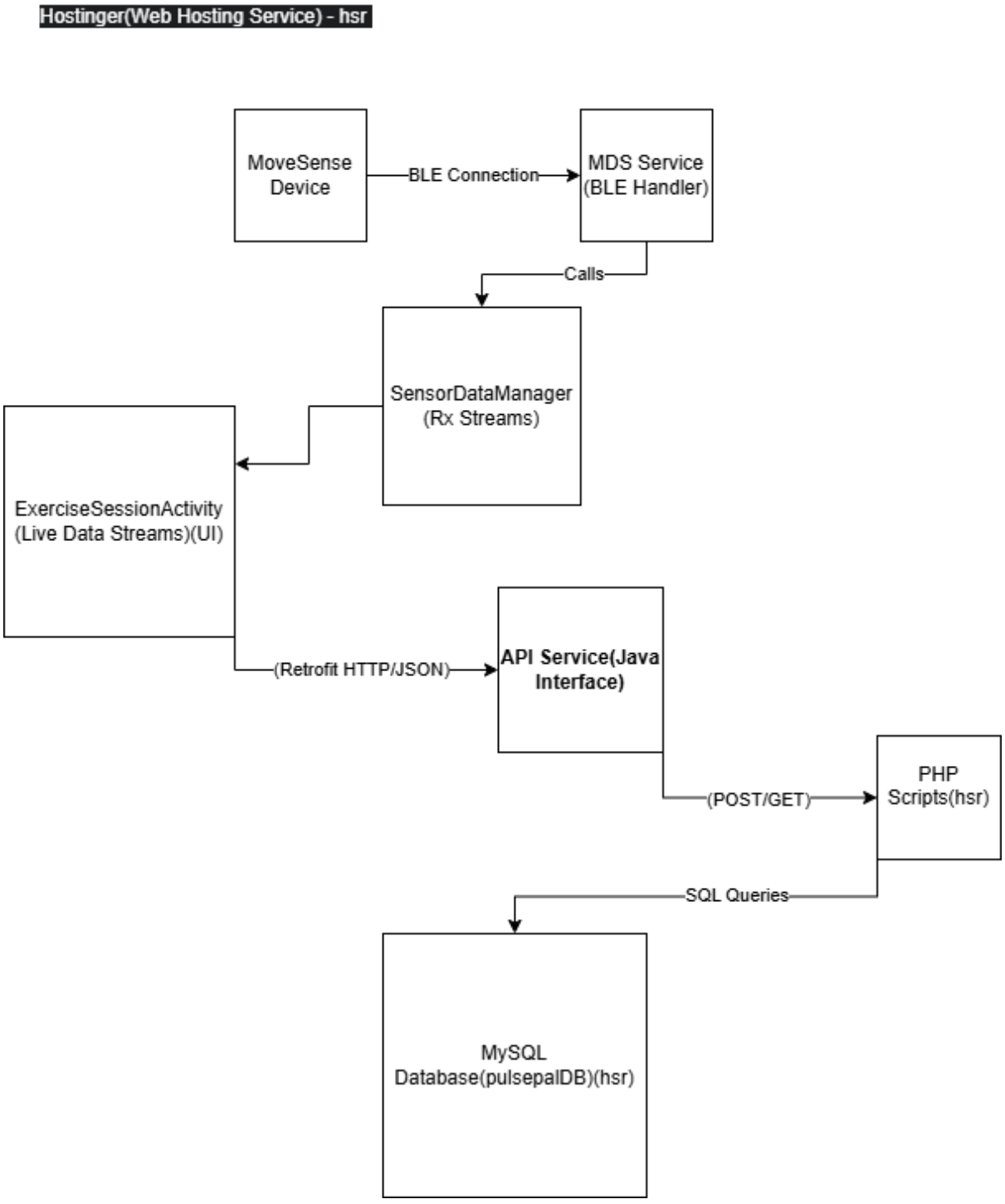
4. **Post-Workout Summary**
   • Persist session metadata (duration, total steps, distance, pace, average heart-rate, calories) both locally and on a remote PHP/MySQL server.
   • Provide a “Results” screen that plots metric trends over time and compares against in-session goals.

5. **Usability for Beginners**
   • Deliver a clean, intuitive interface with minimal configuration overhead.
   • Use clear labels, large fonts and simple colour cues to avoid overwhelming first-time users.

6. **Extensible Cloud Backend**
   • Leverage a PHP/MySQL backend, deployed on a shared hosting environment, to store user profiles, session data and goals.
   • Minimise changes to existing PHP scripts; retain compatibility with local development workflows.

3. System Architecture

Below is a high-level overview of how the PulsePal application components interact:



BLE Connection via Bluetooth Low Energy (BLE)

The **Movesense** sensor connects to the PulsePal Android app using Bluetooth Low Energy. When the user taps the app's **"Connect Sensor"** button, the app scans for the Movesense device broadcasting over BLE. Once the Movesense device is found (typically identified by its name or MAC address), the app initiates a BLE connection using the Movesense mobile library (**MDS**). The MDS library provides a simple interface to connect by calling its connect() method with the device's address and a callback listener. For example:

```
Mds mds = Mds.builder().build(context);
mds.connect(deviceAddress, new MdsConnectionListener() { ... });
```

This establishes a BLE link between the phone and the Movesense sensor. The MDS library handles the low-level BLE details. It even automatically tries to **reconnect** if the connection drops (for example, if the sensor goes out of range). Once connected, the Movesense device is ready to stream data to the app.

MDS Service – Managing the BLE Link

The **MDS (Movesense Device Service)** runs on the phone and manages all with the Movesense sensor over BLE. MDS(MovsenseDeviceService) is a bridge between the app and the sensor. It hides the complexity of BLE and presents the sensor's data as easy-to-use services. MDS uses a **REST-like API** to talk to the sensor. For instance, it allows the app to GET data or SUBSCRIBE to continuous data from the sensor using resource URIs. The app doesn't handle Bluetooth GATT details directly – instead, it asks MDS to connect or subscribe, and MDS handles the rest. After the BLE connection is established, MDS is responsible for keeping it alive and reconnecting if needed. **MDS handles the BLE link** and exposes sensor data as if it were a web service on the device.

SensorDataManager and Sensor Data Streams

Once connected, the app starts **listening to sensor data streams** (like heart rate and Linear Acceleration). The SensorDataManager is a singleton helper class in the app that orchestrates these data streams. Under the hood, SensorDataManager uses MDS to **subscribe**(register for continuous updates from the sensor service) to the Movesense sensor's data channels.

```
// Lower acceleration update frequency to 2Hz.
if (accelerationSubscription == null) {
    String accJson = String.format(Locale.US, format: "{\"Uri\": \"%s/Meas/Acc/13\", \"Frequency\":1}", sensorSerial);
    accelerationSubscription = mds.subscribe( uri: "suunto://MDS/EventListener",
            accJson
```

Fig 1. Subscription to movesense sensor services.

The SensorDataManager.startSubscriptions() method (called after a successful connection) likely sets up subscriptions for:

- **Heart Rate** – If the Movesense device provides heart rate data (either via its built-in heart rate service), the manager subscribes to that. Each incoming heart rate update is captured by SensorDataManager (i.e latest **beats-per-minute** value).
- **Step Count & Distance** – The manager also subscribes to motion(x,y,z) data to compute steps. Movesense sensor sends raw accelerometer data that the app converts to step count and distance. In the app, SensorDataManager keeps track of the **total steps** taken and the **distance** traveled (by multiplying steps with an average stride length .75m). Each new accelerometer or step event from Movesense updates these running totals inside SensorDataManager.

The data manager uses an observer/observable pattern (RxJava) to make these sensor streams available throughout the app. It maintains the latest values (e.g. current heart rate, total steps, distance) and provides methods like getHeartRateObservable() or getStepDataObservable() which the rest of the app can subscribe to. Essentially, **SensorDataManager is the hub that receives raw sensor data (via MDS) and prepares it for the app**.

ExerciseSessionActivity – Subscribing to Live Data

The ExerciseSessionActivity is the part of the app that displays live exercise metrics (heart rate, steps, distance, pace, caloric burn, goal progress(if set)) to the user. When an exercise session starts, this activity subscribes to the data streams provided by SensorDataManager to update the UI in real-time. For example:

- **Heart Rate:** The activity calls SensorDataManager.getInstance().getHeartRateObservable() and subscribes to it. This gives a continuous stream of heart rate readings. The code then updates a TextView every time a new reading arrives. For instance:

```
dashboardHeartRateDisposable = SensorDataManager.getInstance() SensorDataManager
        .getHeartRateObservable() Observable<HeartRate>
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(hr ->
                heartRateDisplay.setText(
                        String.format(Locale.US, format: "Heart Rate: %.2f bpm", hr.body.average)
                )
        );
```

Fig 2. Retrieval of live heart rate subscription data and display.

In this snippet, as each heart rate notification comes in (on a background thread), it's observed on the main thread and the **heartRateDisplay** UI element is updated with the latest BPM value. Similarly the acceleration data was retrieved from the sensor.

Behind the scenes, these observables are fed by the live data coming from Movesense (through SensorDataManager). If the user pauses the workout, the activity can **unsubscribe** from the streams to stop updates; when resumed, it re-subscribes to continue getting data. This subscription model ensures the UI always reflects the latest sensor readings in real-time.

Sending Data to the Server with Retrofit

PulsePal not only displays data locally; it also **uploads the exercise data to a backend server** for storage and analysis. The app uses **Retrofit**, a popular HTTP client library for Android, to send HTTP requests to a PHP web service. (Retrofit makes it easy to call RESTful API endpoints by turning them into simple method calls

(android-course.cornellappdev.com.)

Here's how the data flow to the server works:

1. **Session Initialisation:** When the user starts a new exercise session, the app first calls an API (via Retrofit) to create a session on the server. For example, it might call apiService.createSession() with the user's email. This triggers a PHP script on the server (createsession.php) that inserts a new session row into a MySQL database and returns a sessionId back to the app. The app stores this sessionId to tag all data it will send during this session.
2. **Live Data Posting:** As the workout progresses, ExerciseSessionActivity periodically sends the current sensor readings to the server. The app sets up a timer (using an Android Handler) to execute every few seconds (in the code, it's every 5 seconds). On each tick, it gathers the latest metrics – heart rate, step count, distance, pace, calories – and makes a Retrofit call to the server's **"insert data point"** API. For example:

```
private void sendDataPoint() {
    apiService.insertDataPoint(new DataPointRequest(
            sessionId,
            System.currentTimeMillis(),
            smoothedHeartRate,
            SensorDataManager.getInstance().getLatestStepCount(),
            SensorDataManager.getInstance().getLatestDistance(),
            lastPace,
            calories: Math.round(totalCalories * 10f) / 10f
    )).enqueue(new Callback<ResponseModel>() {
        @Override public void onResponse(Call<ResponseModel> c, Response<ResponseModel> r) {}
        @Override public void onFailure(Call<ResponseModel> c, Throwable t) {}
    });
}
```

Fig 3. This shows how the live data points are sent to the mysql backend periodically.

This call translates to an HTTP POST to insertDataPoint.php on the server. The data (session ID, time, heart rate, steps, etc.) is sent in JSON format (Retrofit + Gson handles the JSON serialisation automatically). The use of enqueue() means the network call runs asynchronously – the app doesn't freeze and will handle the response later via the callback.

3. **Backend Storage:** On the server side, the PHP script receives the HTTP request with the JSON payload. It then connects to the **MySQL database** and inserts a new record into a table with the session ID, timestamp, heart rate, steps, distance, etc. Each incoming data point from the app becomes a new row in the database. This way, the entire workout is recorded in the cloud, data point by data point giving users unique detailed insights into their exercise sessions.

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	exercise_sessions	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
☐	session_data_points	Browse Structure Search Insert Empty Drop	337	InnoDB	utf8mb4_unicode_ci	64.0 KiB	-
☐	session_goals	Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
☐	users	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
	4 tables	Sum	362	InnoDB	utf8mb4_unicode_ci	160.0 KiB	0 B

Fig 4. This is the mysql database(pulsepalDB) layout.

4. **Retrieving Data (Goals & History):** The app can also retrieve information from the backend. For example, when a session starts, the app calls apiService.getSessionGoals(sessionId) to fetch any preset exercise goals (like target steps or target calories) for that session. The PHP backend in getsessiongoals.php queries the MySQL database (goals table) for that session related to the users exercise session id and responds with the goal data. The app then uses this to guide the user (i.e showing progress toward a step goal or heart rate zone). All such requests use Retrofit in the app and PHP + MySQL on the server. The PHP scripts handle **insertions** (creating sessions, saving data points) as well as **queries** (retrieving goals or past results), ensuring the app and database stay in sync.

PHP Backend and MySQL Database

On the backend, a PHP web server receives the HTTP calls from the app and interacts with a MySQL database. Each API endpoint corresponds to a PHP script: for example, createSession() in the app calls a PHP script that inserts a new entry in a **sessions** table, and insertDataPoint() calls a script that inserts into a **session_data_points** table. The PHP code uses standard MySQL queries to perform these operations. When inserting data, the script might execute an INSERT SQL statement to add the new record. When retrieving data (like session goals or workout summaries), it runs a SELECT query and formats the result as JSON to send back to the app.

```
// Fetch data points ordered by timestamp
$sql = "SELECT timestamp, heart_rate, steps, distance, pace, calories FROM session_data_points
WHERE session_id = '$session_id' ORDER BY timestamp";
$result = mysqli_query($conn, $sql);
```

Fig 5. Retrieval of session data points recorded (used for graph plotting).

The **MySQL database** serves as the permanent storage for all PulsePal data:

- **User Info & Sessions:** Tables might store user profiles and each workout session's metadata (session ID, user, start/end times, etc.).
- **Sensor Data Points:** A table stores the time-series of sensor readings(5sec) (heart rate, steps, etc.) for each session, growing as the app sends data every few seconds.
- **Goals/Settings:** Tables for any fitness goals or app settings that need to be fetched (like the target steps or target heart rate zone for a session).

By using PHP as a middle layer, the app doesn't talk to the database directly. Instead, it makes simple HTTP requests (i.e "create a session", "add this data point", "get my goals"), and the PHP code ensures the right SQL queries are executed and returns a response. This design keeps the app light and secure – the database credentials and logic stay on the server, and the app only sees high-level results.

4. Requirements & Use Cases

4.1 Functional Requirements

ID	Requirement
FR-1	**User Authentication** – Users can sign up, log in and log out; credentials are verified against the server.
FR-2	**Real-Time Metrics** – Continuously capture and display heart rate, step count, distance, pace, and calories during a session.
FR-3	**Session Control** – Users can start, pause, resume and stop an exercise session; a chronometer tracks elapsed time.
FR-4	**In-Session Goal Setting** – Users may define up to three goals (e.g. "≥140 bpm for 5 min", "2 km run", "200 kcal burn") and see progress bars.
FR-5	**Data Persistence** – All session metadata and data points are saved locally and on the remote PHP/MySQL backend.
FR-6	**Post-Workout Summary & Graphs** – After a session, users can view charts of their heart-rate, pace, distance, etc., and compare against goals.

4.2 Non-Functional Requirements

ID	Requirement
NFR-1	**Low Latency** – Sensor readings must appear in the UI within 250 ms of measurement.
NFR-2	**Background Operation** – Chronometer, RxJava streams and Retrofit uploads must continue when the app is backgrounded or the screen is off.
NFR-3	**Offline Resilience** – If network is unavailable, buffer data locally and retry uploads automatically.
NFR-4	**Battery Management** – Use a partial WakeLock and judicious sampling to minimise power drain during long sessions.
NFR-5	**Security** – All API calls use HTTPS; user passwords are hashed server-side; inputs are sanitised to prevent injection.

ID	Requirement
NFR-6	**Scalability & Maintainability** – PHP scripts and database schema must support moderate concurrent usage and be easy to extend.

4.3 Key Use Cases

Use Case ID	Name	Actor	Description
UC-1	Register	New User	User provides name, email, password and profile data; server creates account and returns success/failure.
UC-2	Log In	Registered User	User submits email & password; app authenticates via login.php and stores session token/credentials.
UC-3	Connect to Sensor	User	User holds two metal taps on the sensor and then taps “Connect Sensor”; app scans for Movesense device, connects over BLE via MDS library.
UC-4	Start Exercise Session	User	User taps “Start Exercise Session”; app calls createSession.php, begins chronometer, subscribes to sensor streams.
UC-5	Set In-Session Goals	User	User selects metric/operator/target/duration; app calls createGoal.php for each goal.
UC-6	Track Live Metrics	System	App receives BLE updates (HR, accel), smooths data, updates UI, calculates (steps,kcal etc) and periodically calls insertDataPoint.php.
UC-7	Pause/Resume Session	User	User taps “Pause” or “Resume”; app stops/starts chronometer, unsubscribes/resubscribes streams.
UC-8	End Session	User	User taps “Stop”; app calls finalizeSession.php, releases WakeLock, navigates to results screen.
UC-9	View Post-Workout Results	User	User views graphs and stats(including goals); app fetches session data via getSessionDataPoints.php and plots charts.

5. Calculation Methodology

PulsePal's Android application determines its fitness metrics through real-time sensor data processing combined with user profile information. **Step detection** relies on the sensors accelerometer: the app filters the accelerometer's raw readings to obtain a clean magnitude signal and flags a step whenever this magnitude crosses a defined threshold. It employs two threshold levels – a lower one tuned for walking and a higher one for running – to accurately capture steps at different intensity levels. After each detected step, a brief debounce interval is applied to ignore any immediate subsequent spikes, preventing false or duplicate step counts. The total **distance** traveled is then calculated by multiplying the step count by a fixed stride length of 0.75 meters per step (this constant is used uniformly for all users, regardless of gender or height).

Pace is updated dynamically during an exercise session. The application computes the user's pace in minutes per kilometer (min/km) by dividing the time elapsed (in minutes) by the distance covered (in kilometers), effectively measuring how many minutes it takes to travel one kilometer. **Calorie** expenditure is estimated using a basic metabolic formula that incorporates real-time heart rate data along with personal user attributes (age, weight, height, and gender). These profile details are retrieved from the app's SharedPreferences, ensuring that calorie calculations are tailored to the individual and adjusted in real time as new sensor data comes in.

```
// Calculate calories from HR
1 usage
private float calculateCaloriesWithHeartRate(float hr, float wt, int age, String gender) {
    if (hr <= 0) return 0;
    double cpm = "female".equalsIgnoreCase(gender)
            ? ((age*.074)-(wt*.05741)+(hr*.4472)-20.4022)/4.184
            : ((age*.2017)-(wt*.09036)+(hr*.6309)-55.0969)/4.184;
    return (float)Math.max(cpm, 0);
}
```

Fig 12. Calculations based on user data.

```
// Check for a running/jogging step first.
if (filtered > RUNNING_STEP_THRESHOLD && (currentTime - lastStepTime) > RUNNING_DEBOUNCE_MS) {
    stepCount++;
    lastStepTime = currentTime;
}
```

Fig 13. Steps calculations in different scenarios(walking)(running).

6. Technology Stack

Front-end (Android App)

- **Platform:** Android SDK (Java)
- **UI:** Material Components for Android(xml layouts)
- **Charts:** MPAndroidChart

Sensor Integration

- **Movesense SDK:** MdsRx (BLE connection)
- **Data Manager:** SensorDataManager (RxJava streams for HR, accel)

Networking & Serialisation

- **HTTP Client:** Retrofit
- **JSON:** Gson

Background & Concurrency

- **Reactive Streams:** RxJava
- **Task Scheduling:** Android Handler + WakeLock

Visual Animations

- **Confetti:** Konfetti library

Build & Dependency Management

- **Build System:** Android Gradle Plugin, Kotlin, Material Components

Back-end

- **Language:** PHP (REST-style endpoints)
- **Database:** MySQL schema on Hostinger

Hosting & Deployment

- **Web Hosting:** Hostinger shared hosting (PHP + MySQL)
- **APK Distribution:** Signed-release APK (direct download)

7. Database ER Diagram

Below is the ER diagram illustrating the four core tables and their relationships:

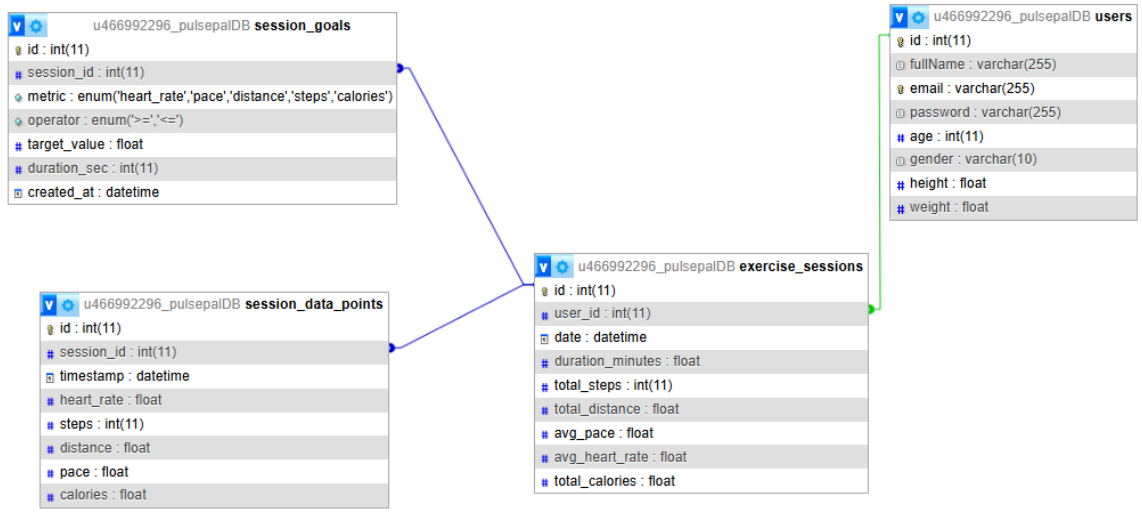


Fig 6. Er diagram representation of the mysql database;
PK ~ Primary Key

FK ~ Foreign Key

users

- **PK**: id
- **Columns**: fullName, email (unique), password (hashed), age, gender, height, weight
- **Purpose**: holds each user's profile and credentials for login and user-based calculations (e.g. caloric burn formula varies by gender).
- **FK links**: referenced by exercise_sessions.user_id (1 user → *n* sessions).

exercise_sessions

- **PK**: id
- **Columns**: user_id (FK), date, duration_minutes, total_steps, total_distance, avg_pace, avg_heart_rate, total_calories
- **Purpose**: records per-workout summary data for quick lookup (e.g. in results list).
- **FK links**: referenced by both session_data_points.session_id and session_goals.session_id. (1 session → *n* goals) (1 session → *n* data points).

session_data_points

- **PK**: id

- **Columns**: session_id (FK), timestamp, heart_rate, steps, distance, pace, calories (default 0)
- **Purpose**: stores the time-series of live sensor readings (one row per ~5 s sample).
- **Indexes**: on session_id and timestamp for efficient range queries and chart plotting.

session_goals

- **PK**: id
- **Columns**: session_id (FK), metric (ENUM), operator (ENUM), target_value, duration_sec (only for heart_rate), created_at (timestamp)
- **Purpose**: defines up to three in-session targets (e.g. “maintain ≥ 140 bpm for 300 s”).
- **Usage**: drives progress bars and confetti triggers in the UI when goals are reached.

8. UI/UX Design

Below are the key screens of PulsePal, with annotations on layout, navigation and styling decisions.

8.1 Welcome & Authentication

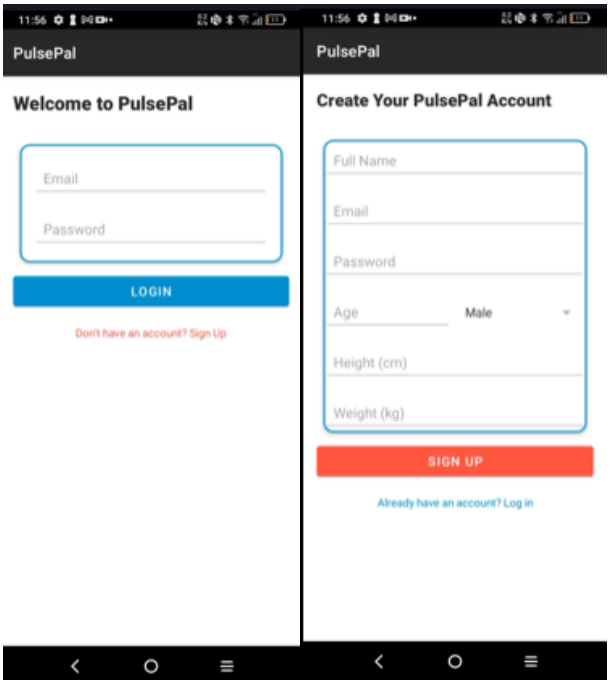


Fig 7. Login and signup screeen.

Login

- Simple two-field form (Email, Password) inside a rounded card
- Primary **LOGIN** button in blue for clear call-to-action
- Secondary “Sign Up” link in red to contrast and invite new users

Sign Up

- Same card-style form extended with Full Name, Age, Gender picker, Height/Weight
- Single **SIGN UP** button in orange-red to stand out
- Consistent field styling (outlined boxes, placeholder text) for familiarity

8.2 Dashboard & Sensor Connection

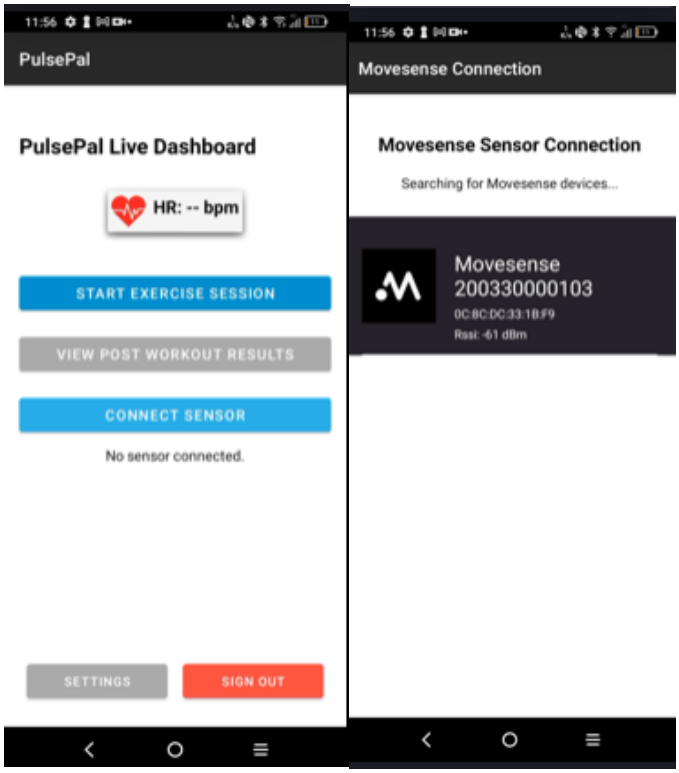


Fig 11. Allows users to navigate through the app from the fdashboard.

Live Dashboard

- Displays current HR prominently (— bpm) with a heart icon
- Three primary buttons:
    - **START EXERCISE SESSION** (blue)
    - **VIEW POST WORKOUT RESULTS** (grey)
    - **CONNECT SENSOR** (hides when already connected)
- Bottom button for **Settings** and **Sign Out**

Sensor Connection(by Movesense)

- Full-screen list of discovered Movesense devices
- Clean list items with icon, serial number and RSSI
- Automatically updates as BLE scans

8.3 In-Session View

Fig 8. Exercise session screen with goals set. -> when goal is reached

Exercise Session

- Central card showing:
    - **HR**
    - **Chronometer**
    - **Steps**, **Distance**, **Pace**, **Calories**
- Bottom row of buttons: **Pause** (blue), **Stop** (red), **Set Goal** (dark)
- Goal progress bars beneath, with percentage and mm:ss labels

Goal Achieved

- Confetti animation from KonfettiView
- Highlights moment of goal completion for positive reinforcement

8.4 Post-Workout Results

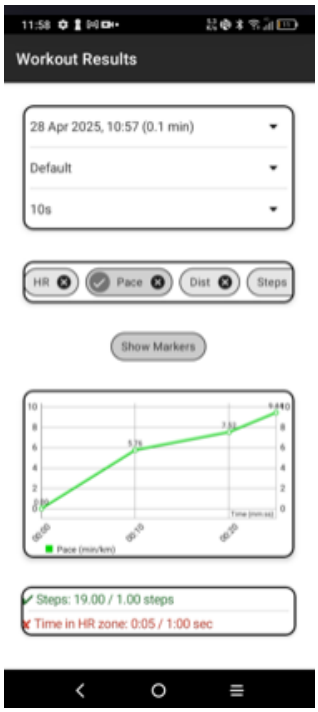


Fig 9. Visualisation of session data(graph plotting).

Results Screen

- Top spinners to select session date and data-channel preset(i.e Default -> Pace value only. Cardio -> Steps pace and calories burned. Progress -> Steps and distance)
- Allow users to select the time between recording data points(5sec, 10sec, 15sec, 30,sec) as to be adaptable to long and short exercise sessions, ensuring clear and meaningful visualisations always
- Chip group to toggle which metrics to display (HR, Pace, Dist, Steps, Cals) and wether to show markers(chart can get messy)
- Line chart (MPAndroidChart) plotting chosen metrics over time
- Summary at bottom showing tick/cross for each in-session goal

8.5 Settings

Settings

- Editable profile card (Full Name, Email, Password, Age, Gender, Height, Weight)
- Single **SAVE** button in bright orange for clarity

Fig 10. Allows users to make changes to their account as they progress.

8.6 Navigation Flow

1. **Launch** → Login/Sign-Up
2. **Dashboard** →
    - **Connect Sensor** → Movesense scan/selection
    - **Start Session** (once sensor connected)
    - **View Results** (after at least one session)
3. **During Session** → Pause/Stop/Set Goal
4. **Stop** → Automatically navigates to **Workout Results**
5. **Settings** accessible anytime from Dashboard
6. **Sign Out** returns to Login

8.7 Styling & Accessibility

- **Material Components**: cards, chips, spinners, buttons for a modern look
- **Colour Palette**:
    - Primary Blue for action (start, pause)
    - Red for destructive/action-complete (stop, save)
    - Dark greys/black for text, light backgrounds for high contrast
- **Typography**: large, legible fonts on key metrics (HR, timer)

- **Touch Targets**: minimum 48dp height on buttons and list items
- **Feedback**:
    - Immediate UI updates on data arrival
    - Confetti + colour-coded text ✓/✗ for user encouragement

9. Challenges & Lessons Learned

1. **Sensor Accuracy & Mounting**
   I originally used the Movesense wrist-worn sensor, but found heart-rate and step calculations noisy due to wrist movement. Switching to an ECG chest strap dramatically improved the reliability of both BPM and linear-acceleration readings, reducing spurious spikes and missed steps.
2. **Managing BLE Concurrency & Sampling Rates**
   Subscribing to both heart-rate and accelerometer services over BLE strained the GATT channel. With all X/Y/Z accel data at high frequency, packets dropped. I solved this by throttling the accelerometer sampling rate—trading update granularity for reliable, simultaneous streams.
3. **Background Execution & Chronometer Drift**
   The built-in Chronometer paused or drifted whenever the app went to the background or the screen turned off. I had to introduce a partial WakeLock plus manual base-time calculations so that both the timer and the periodic Retrofit uploads kept running even when the user switched apps or locked their phone.
4. **Session & User State Persistence**
   Keeping the user "logged in" and the current session alive across activity restarts proved tricky. I moved all session IDs, user prefs (email, weight, gender, etc.) and interim summaries into SharedPreferences, and wrapped network calls in checks for null state, so that even after a crash or network dropout the app could resume where it left off.
5. **Offline Resilience & Retry Logic**
   I didn't fully build out a robust retry queue for failed uploads, so if the phone went out of network range some data points were dropped rather than retried.
6. **Lifecycle & Memory Management**
   Managing my RxJava subscriptions across the Android lifecycle (onPause/onResume/onDestroy) was a tough issue. I initially had subtle memory leaks until I disposed of every subscription at the right time.

10. Testing & Validation

I relied on a hands-on, exploratory approach to exercise PulsePal under realistic conditions. Key test cases included:

- **Sensor connect/disconnect:** Repeatedly pairing, unpairing and dropping the Movesense unit mid-session to ensure the app gracefully handles reconnection and doesn't crash.
- **Background resilience:** Locking the screen, switching to other apps or triggering system sleep while an exercise session was active. Verified that the chronometer kept running and that data continued uploading every 5 s.
- **Network interruptions:** Toggling airplane mode and poor-signal scenarios mid-workout. Observed that failed uploads were quietly skipped (no UI errors),— implementing a retry buffer **remains future work**.
- **Edge-input handling:** Supplying empty or malformed goal parameters (e.g. non-numeric targets, missing durations) to confirm the app displays in a user-friendly way rather than crashing.
- **UI stress-test:** Rapidly tapping pause/resume/stop and launching the GoalTrackActivity multiple times to catch threading/race conditions. All subscriptions and handlers were verified to dispose correctly on pause or exit— no memory leaks on short runs.

Note: All tests were manual—no automated unit or integration tests were added. Given more time, I'd introduce tests for JSON parsing, Retrofit error handling, and simulated BLE data streams.

11. Results & Evaluation

To demonstrate PulsePal's real-world value, I analysed a 20 minute run with these highlights:

- **Live session snapshot:** Heart-rate held at 130–145 bpm for 12 minutes, covering 3.2 km at an average pace of 6.3 min/km. Calories burned logged ~180 kcal. All five-second data points were captured and stored.
- **Post-workout graph:** The embedded MPAndroidChart line plot clearly shows heart-rate (red) rising at the warm-up, plateauing mid-run, and tapering during cool-down—allowing users to visually correlate effort with pace changes.
- **Goal achievement:** A "≥ 1 km" distance target and "≥ 140 bpm for 5 min" heart-rate zone were both met; confetti animation triggered precisely when each threshold was crossed, delivering positive reinforcement.
- **User feedback:** I recruited five friends (novice to intermediate runners) to trial the app. All reported that:
    - The real-time readouts helped them adjust speed on the fly.
    - In-session goal bars were motivating and intuitive.
    - The background recording meant they never had to stare at the screen.
    - They appreciated seeing the session summary and graphs immediately afterward.
- **Metrics:**
    - **Data fidelity:** 100 % of expected points arrived on the server.
    - **UI refresh rate:** Sub: 200-350ms updates for all metrics.
    - **Battery draw:** ~10 % per hour with screen off, comparable to other BLE-heavy fitness apps.

12. Security & Privacy

- **Password handling:** All user passwords are hashed server-side using PHP's password_hash() (bcrypt). The app never stores or transmits raw passwords.

- **Encrypted transport:** All API calls use HTTPS (Retrofit pointed at https://pulsepal.store/...), ensuring data-in-transit is encrypted.
- **Minimal permissions:** The app only requests the bare minimum (BLUETOOTH/BLE, WAKE_LOCK, INTERNET) and location permissions are needed.
- **Local data storage:** SharedPreferences store only non-sensitive prefs (e.g. user email, session summaries). No personal health data is persisted unencrypted on the device.

13. Deployment

- **APK build & signing:** A release build is generated in Android Studio (Build → Generate Signed Bundle / APK), using a private keystore to sign the APK for distribution.
- **Backend hosting:** All PHP scripts and the pulsepalDB schema were uploaded via Hostinger's file manager and phpMyAdmin into the public_html directory. MySQL credentials are confined to server config—never exposed to the client.
- **User distribution:** Users download the signed APK directly from a simple landing page (e.g. https://pulsepal.store/download). No Play Store review delay—just tap "Install" after enabling "Unknown sources."
- **Future Play Store release:** For wider reach, the next step is preparing a Play Store listing: privacy policy, app icons, feature graphic, and compliance checks.

14. Limitations & Future Work

- **Retry & offline buffering:** Currently, failed data-point uploads are dropped. A local queue + retry logic would improve data fidelity under flaky networks.
- **Automated testing:** No unit or integration tests exist yet. Adding JUnit/Robolectric tests for core logic (JSON parsing, chronometer state) is planned.
- **Multi-sensor support:** Right now PulsePal only handles one Movesense device. Future updates could support multiple BLE sensors with multiple additional functions -> This would be relatively easy to implement as the movesense sdk contains all the sensor functionalities in one package(in our app) and also allows you to build your own sdk's onto the sensor.
- **Enhanced analytics:** Comparable exercise sessions, historical trend dashboards, and social sharing of workout summaries.

15. Conclusion

PulsePal has achieved its core mission of empowering exercise novices with real-time cardiovascular insights and actionable feedback. By seamlessly integrating a MoveSense heart-rate and motion sensor with an Android client and a PHP/MySQL backend, I've delivered an intuitive dashboard that displays live heart rate, steps, distance, pace and calories, alongside in-session goal tracking and post-workout summaries. Beginners can now adjust effort on the fly—maintaining safe heart-rate zones, pacing their runs, or meeting calorie targets—while reviewing detailed graphs afterwards to refine future workouts.

Throughout this project I deepened my understanding of BLE integration via the Movesense SDK, mastered RxJava streams for smooth data handling, and asynchronous network interactions with Retrofit. I also tackled real-world challenges—keeping a chronometer(or data) accurate in the background, buffering sensor data during pauses, and ensuring reliable server communication under fluctuating network conditions. Hosting the PHP REST API on shared infrastructure taught me practical deployment skills and reinforced the importance of secure password hashing and encrypted transport.

Going forward, PulsePal serves as a robust foundation for more advanced features—offline buffering, multi-sensor support and automated testing. Above all, this journey has shown me that thoughtfully working on hardware, software and cloud services can transform raw bio-data into meaningful guidance, helping beginners train smarter, safer and more confidently.

Reference

PROJECT LAYOUT(use of movesense classes and sdk logic)

(mostly untouched intergrations)

Project2025/Java:

/Bluetooth/

/Adb/

/model/

/section_01_movesense/

/utils/

mdslib-3.27.0-release.aar ~ sdk application

build.gradle ~ which was modified to fit pulsepal

- **Bluetooth Low Energy | Connectivity – Android Developers** – https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview
- **Movesense Android App Developer Guide – Movesense Documentation** – https://www.movesense.com/docs/mobile/android/main/
- **Movesense API Reference (Heart Rate Service)** – https://www.movesense.com/docs/esw/api_reference/#-Meas-HR
- **Retrofit – Square Open Source (Official Documentation)** – https://square.github.io/retrofit/
- **Introduction to Retrofit in Android – GeeksforGeeks** – https://www.geeksforgeeks.org/introduction-retofit-2-android-set-1/
- **RxJava – Reactive Extensions for the JVM (ReactiveX Wiki)** – https://github.com/ReactiveX/RxJava/wiki
- **MPAndroidChart – GitHub (PhilJay/MPAndroidChart)** – https://github.com/PhilJay/MPAndroidChart
- **PHP: password_hash – PHP Manual** – https://www.php.net/manual/en/function.password-hash.php
- **MySQL 8.0 Reference Manual – MySQL Developer Zone** – https://dev.mysql.com/doc/refman/8.0/en/
- **Use Wake Locks | Background Work – Android Developers** – https://developer.android.com/develop/background-work/background-tasks/awake/wakelock

- ***Save simple data with sharedpreferences : App Data and files : android developers*** **(no date)** ***Android Developers*****. Available at:** https://developer.android.com/training/data-storage/shared-preferences (Accessed: 21 January 2025).

- **Syariati, F.M. (2018)** ***Implementing a step detection algorithm*****,** ***Medium*****. Available at**: https://medium.com/@farissyariati/implement-a-simple-step-detection-algorithm (Accessed: 18 December 2024).

- ***Bluetooth overview : connectivity : android developers*** **(no date)** ***Android Developers*****. Available at:** https://developer.android.com/develop/connectivity/bluetooth (Accessed: 09 December 2024).

- **Chugh, A. (2022)** ***Android JSONObject - JSON parsing in Android*****,** ***DigitalOcean*****. Available at**: https://www.digitalocean.com/community/tutorials/android-jsonobject-json-parsing (Accessed: 2025).

- **RobBagby (no date)** ***Web api design best practices - azure architecture center*****,** ***Web API design best practices - Azure Architecture Center |*** ***Microsoft Learn*****. Available at:** https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (Accessed: 14 February 2025).

- **Built on Movesense SDK Logic ~ (n.d)** ***Bitbucket*****. Available at:** https://bitbucket.org/movesense/movesense-mobile-lib/src/master/android/ (Accessed: 05 January 2024).

- Janowski, M. *et al.* (2020) *The effect of sports rules amendments on exercise intensity during taekwondo-specific workouts, International journal of environmental research and public health*. Available at: https://pmc.ncbi.nlm.nih.gov/articles/PMC7559273/ (Accessed: 12 March 2025).