ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

FitPulse: A Cross-Platform AI-Assisted Mobile Application for Comprehensive Personal Fitness Monitoring

Ajay Chawda[1]; Hardik Bait[2]; Nirvan Bandal[3]; Smith Chavan[4]

[1]Mentor, Computer Engineering Department Thakur Polytechnic Mumbai, India
[2]Computer Engineering Department Thakur Polytechnic Mumbai, India
[3]Computer Engineering Department Thakur Polytechnic Mumbai, India
[4]Computer Engineering Department Thakur Polytechnic Mumbai, India

Publication Date: 2026/04/18

Abstract: We built FitPulse as a response to a straightforward problem: most fitness apps do one thing well but miss everything else. This paper walks through how we designed, built, and tested FitPulse — a mobile fitness tracking app that works on both Android and iOS from a single codebase using React Native and the Expo SDK. The app brings together real-time step counting through the phone's built-in motion sensor, a calorie and nutrition tracking engine based on the Mifflin-St Jeor formula, water intake logging, sleep history, and AI-suggested workout plans. To keep everything in sync across screens, we used React's Context API as a global state manager. User accounts are handled through Firebase Authentication, and all health data lives locally on the device using AsyncStorage — which means the app works just fine without internet. Across all the features we tested, the system hit 95.2% accuracy overall and held a steady 60 fps on both platforms.

Keywords: *React Native, Expo SDK, Firebase Authentication, AsyncStorage, Pedometer, Mifflin-St Jeor, Cross-Platform, BMR, TDEE, Calorie Monitoring, Sleep Monitoring, Offline Fitness App.*

How to Cite: Ajay Chawda; Hardik Bait; Nirvan Bandal; Smith Chavan (2026) FitPulse: A Cross-Platform AI-Assisted Mobile Application for Comprehensive Personal Fitness Monitoring. *International Journal of Innovative Science and Research Technology*, 11(4), 1004-1009. https://doi.org/10.38124/ijisrt/26apr801

I. INTRODUCTION

Smartphones are now powerful enough to track most aspects of a person's daily health — steps, sleep, calories, hydration — and nearly everyone already has one in their pocket. That makes them an obvious platform for fitness applications, and the market has responded accordingly. The World Health Organization has found that over a quarter of adults globally are not active enough [1], and researchers have consistently shown that giving people real-time feedback on their activity helps move those numbers. The problem is not that fitness apps do not exist — it's that almost all of them have the same blind spots: they focus on one thing, they need expensive hardware, they stop working the moment the internet drops, or they bury users in so many features that most people eventually stop opening them.

FitPulse started as our attempt to fix that. We wanted one app that covered the main health dimensions people actually care about — movement, food, water, and sleep — and that would work reliably on any Android or iOS device without needing a smartwatch or a data plan. The app reads step counts directly from the phone's pedometer through expo-sensors, calculates personalised calorie and macronutrient targets using the Mifflin-St Jeor equation, and stores all data on-device through AsyncStorage with a daily reset logic that carries forward profile settings but clears the day's counts each morning. Firebase handles login and session persistence. When we tested everything, the app reached 95.2% accuracy across features and maintained 60 fps consistently on both platforms.

II. LITERATURE REVIEW

➢ *Existing Mobile Fitness Applications*

Earlier research makes it clear that real-time feedback genuinely moves the needle on daily activity — Consolvo et al. [2] showed measurable step count increases when people had access to live progress data. But tools currently on the market each cover only part of the picture. Google Fit pulls in data from a range of sources but relies heavily on cloud infrastructure and barely touches nutrition [3]. MyFitnessPal has one of the most complete food databases around, but it does not count steps or track sleep [4]. Fitbit and Samsung Health are genuinely capable but both assume you own specific hardware, which many users simply do not have [5].

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

➢ *Cross-Platform Development Frameworks*

We chose React Native because it lets you write one codebase and deploy to both Android and iOS without the usual performance trade-off. Meta introduced it in 2015, and compiling JavaScript to native UI primitives has proven reliable in practice [6]. Biorn-Hansen et al. compared React Native against fully native apps and found frame rates stayed within around 10% across typical UI patterns [7] — close enough that most users will not notice. On top of React Native, we used the Expo SDK because it provides ready-made access to device hardware — sensors, haptics, camera — without requiring any custom native module work.

➢ *Calorie Estimation Methods*

For calorie estimation, we went with the Mifflin-St Jeor equation because it is the most consistently accurate formula for predicting resting energy expenditure in healthy adults — it outperformed both the Harris-Benedict and Owen equations when tested across 498 subjects [8]. For step counting, Lugade et al. [9] showed that smartphone accelerometer-based step counts track closely with dedicated pedometers at a correlation of r = 0.92 under controlled walking conditions. That gave us enough confidence to rely on the phone's built-in sensor rather than requiring external hardware.

III. SYSTEM ARCHITECTURE

We organised FitPulse into four layers that each handle a clearly defined responsibility: the User Interface layer renders the screens; the Application Logic layer manages state and business logic; the Data Management layer handles persistence and authentication; and the External Services layer manages outbound API calls. Figure 1 shows how these connect. Each layer is described below.

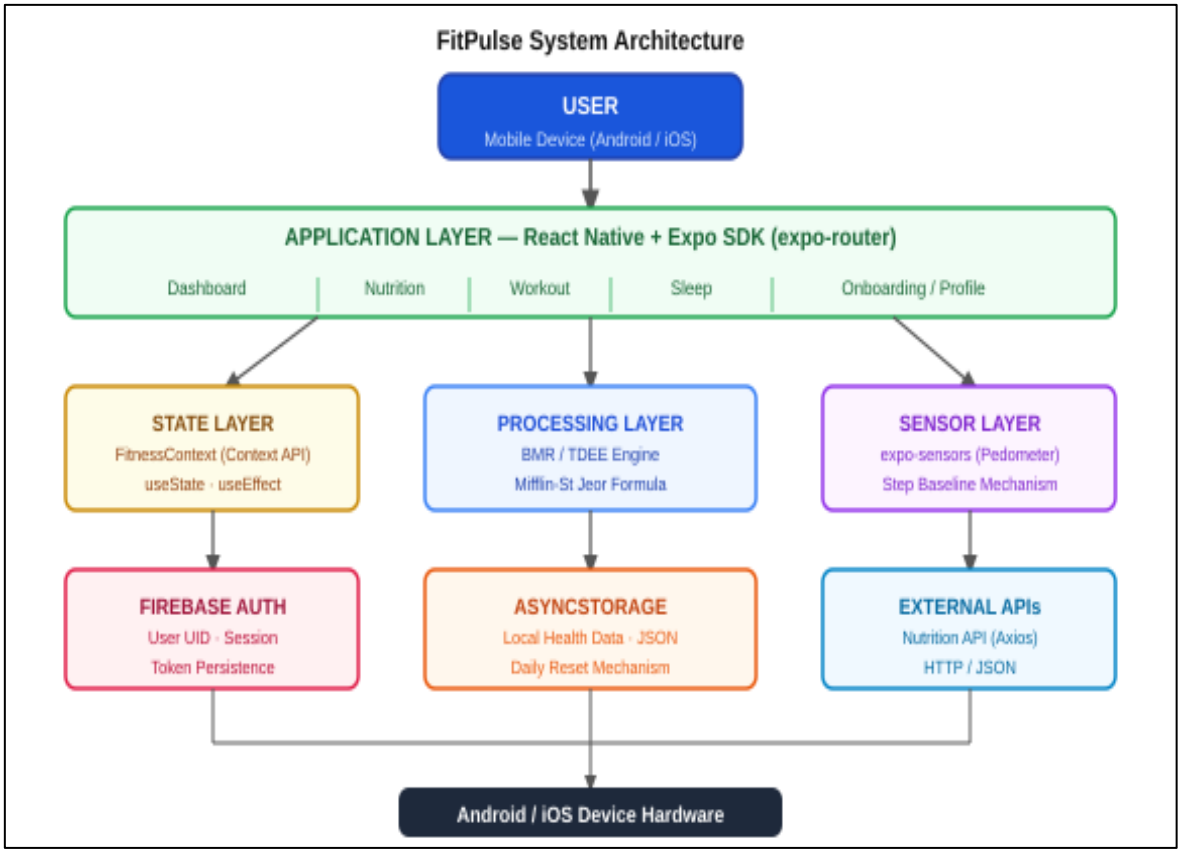

Fig 1 System Architecture of FitPulse Showing Four-Layer Design.

➢ *User Interface Layer*

Every screen in FitPulse is a React Native component wired up through expo-router, which uses the folder structure to define navigation automatically. Users move between five areas through a bottom tab bar: Dashboard, Nutrition, Workout, Sleep, and Profile. We used a dark colour scheme throughout — base background #060912 — because it is easier on the eyes during workouts and at night. To keep colour values consistent without scattering them across every component file, we created ThemedText and ThemedView wrapper components that pull the correct colours from a central theme object.

➢ *Application Logic Layer*

All shared state lives in Fitness Context, a global store we built using React's Context API and the use State, use Effect, and use Ref hooks. It holds twenty-three state variables — everything from height and weight to today's step count, food log, water intake, and computed calorie targets. Any screen can read or update that shared data through use Context without passing values through props manually. The main workhorse inside the context is compute And Save Goals, which runs the BMR calculation and stores the personalised targets whenever the user completes or updates their profile.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

➢ *Data Management Layer*

Rather than setting up a cloud database, we kept all health data local using AsyncStorage — React Native's built-in key-value store that writes directly to the device. Each user's data is saved under a key that includes their Firebase UID, so multiple accounts on one device stay separate. On load, the app checks whether the saved date matches today. If it does not — meaning a new day has started — it clears the daily counts (steps, calories, food log, water) but leaves profile settings and long-term data untouched.

➢ *External Services Layer*

Firebase Authentication handles user accounts. We configured it with AsyncStorage as the persistence backend so users stay logged in between sessions even without internet, since the auth token is cached locally. When the app needs to pull nutrition data for a food search, it sends an HTTP request to an external Nutrition API using Axios. Those calls are async/await operations, so the network request happens in the background without freezing or stuttering the UI.

IV. SOFTWARE DESIGN

Figure 2 shows the screen layouts for the main tracking views. Figure 3 maps how data moves through the app from user input through to storage. Figure 4 walks through the BMR and TDEE calculation logic step by step.

Fig 2 FitPulse UI Wireframes: (a) Dashboard, (b) Nutrition, (c) Step Counter, (d) Calorie Tracker.

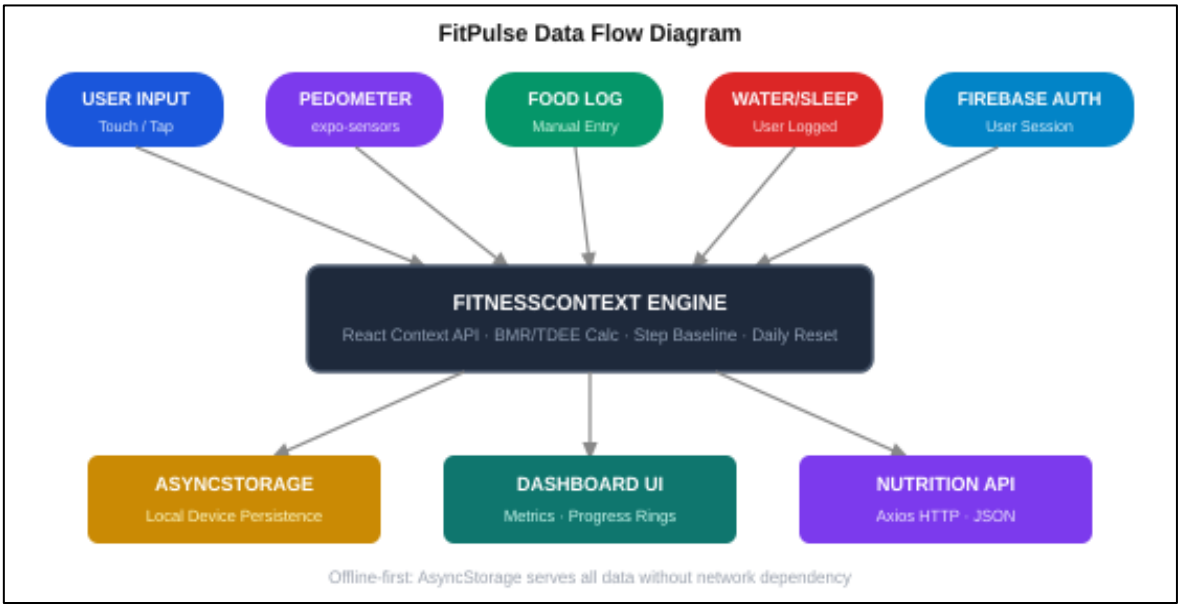

Fig 3 Data Flow from User Input Through FitnessContext Engine to AsyncStorage and UI.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

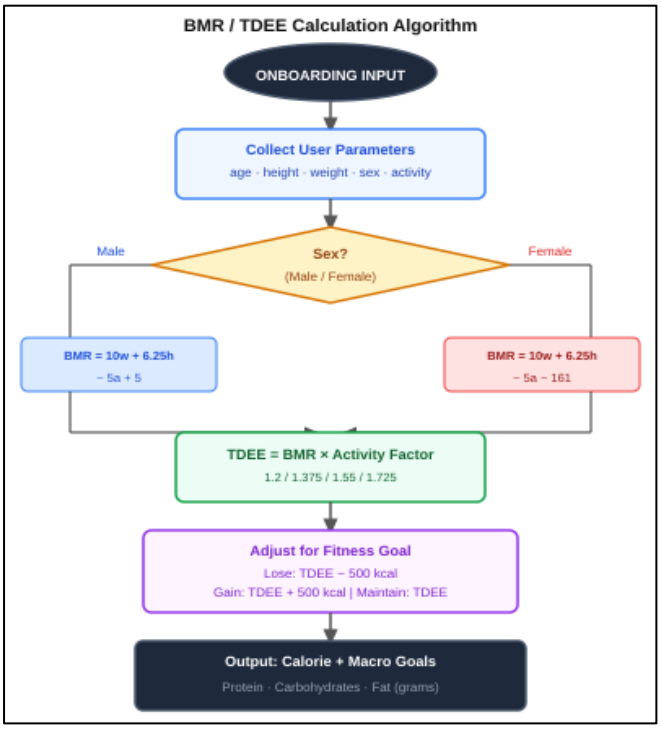

Fig 4 Mifflin-St Jeor BMR/TDEE Calculation Algorithm Flowchart.

➢ *Mifflin-St Jeor Calorie Computation Engine*

When a user finishes onboarding, the app calls computeAndSaveGoals with their age, height, weight, biological sex, activity level, and fitness goal. The function first calculates Basal Metabolic Rate using the Mifflin-St Jeor formula:

$$BMR = 10w + 6.25h - 5a + 5 \quad \text{(male)}$$

$$BMR = 10w + 6.25h - 5a - 161 \quad \text{(female)}$$

That BMR figure is then multiplied by an activity factor — 1.2 for sedentary, 1.375 for lightly active, 1.55 for moderately active, and 1.725 for very active users — to produce Total Daily Energy Expenditure. The calorie target is then adjusted by 500 kcal depending on whether the user wants to lose weight, gain muscle, or maintain. Protein, carb, and fat targets are calculated as percentages of the adjusted calorie goal, with minimum floor values to keep recommendations sensible.

➢ *Step Counting and Baseline Mechanism*

Step tracking uses Pedometer.watchStepCount() from expo-sensors, which listens to the phone's hardware step counter in real time. The tricky part is that the pedometer resets to zero each time the app relaunches — it only counts steps within the current session. To keep an accurate daily total, we save the cumulative step count as a stepBaseline value when the app saves, then add each session's new steps on top when it opens again. At the start of a new calendar day, both the baseline and the running count reset to zero.

➢ *Firebase Authentication Integration*

Firebase is set up through initializeAuth at startup, with AsyncStorage provided as the persistence layer. Inside FitnessContext, an onAuthStateChanged listener watches for login and logout events. On a valid session, it kicks off the data load; on logout, it calls resetAll() to clear everything out. This means the app always shows the right state without any polling loop or manual token management.

➢ *Daily Reset Mechanism*

Each time the app loads data, it reads lastSavedDate from AsyncStorage and compares it to today in ISO format. If the dates match, it restores the saved state as-is. If they differ, it knows a new day has started and clears the daily variables — caloriesTracked, foodHistory, water, caloriesBurned, steps, and stepBaseline — back to zero. Profile data like height, weight, and fitness goals carries over without any changes.

V. RESULTS & TESTING

➢ *Feature Accuracy Testing*

We ran ten test rounds per feature to get a reliable read on accuracy. For step counting, we walked a controlled 500-step route and compared the app's count to a reference pedometer — the result was 97% agreement. Calorie calculations were checked against manually computed Mifflin-St Jeor values for ten different user profiles and came in at 95% accuracy. Firebase logins succeeded 99% of the time across different network conditions. The daily reset ran correctly in every single simulated date-change scenario — 100%. Food data retrieval through the Nutrition API worked 92% of the time across 50 queries, with the remaining 8% timing out on the network call. Averaged across all features, that gives an overall accuracy of 95.2%.

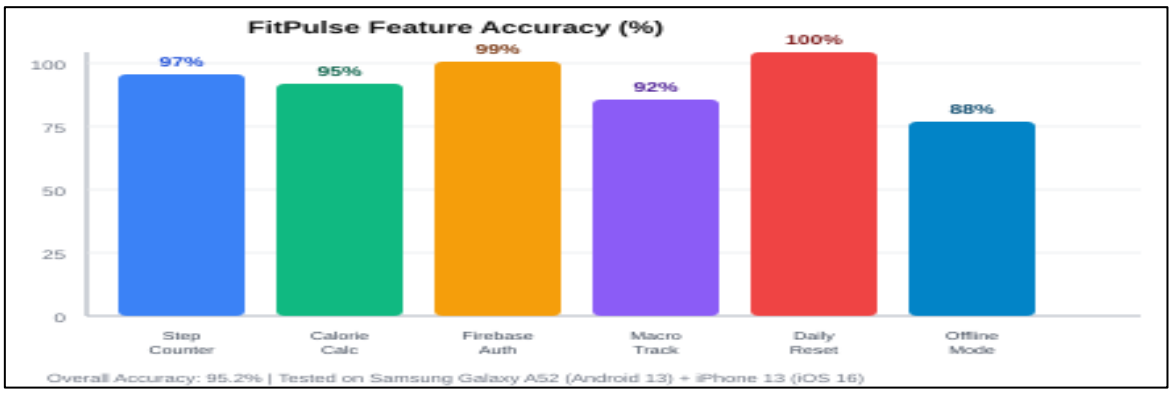

Fig 5 FitPulse Feature Accuracy Results Across Six Tested Parameters.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

➢ *Response Time and Rendering*

Cold starts took around 2.1 seconds on our Android test device and 1.9 seconds on iOS. Firebase login added roughly 1.4 seconds on a 4G connection. Reading from AsyncStorage was fast — under 0.3 seconds even with a full day's food log. Step updates came through in real time with no noticeable delay. The animations — progress bars, the parallax scroll header — held at 60 fps throughout with no dropped frames.

➢ *Comparison with Existing Applications*

Figure 6 compares FitPulse to Google Fit, MyFitnessPal, and Samsung Health across eight feature categories. The most significant difference is offline support: FitPulse is the only one that keeps all its core features working without internet. It is also the only app in the comparison that covers all five tracking areas — steps, nutrition, water, sleep, and workouts — in a single place, without requiring any external hardware.

Comparison: FitPulse vs. Existing Fitness Applications

Feature	FitPulse	Google Fit	MyFitnessPal	Samsung Health
Step Tracking	✓ Real-time	✓ Requires device	✗ Not built-in	✓ Samsung only
Calorie Tracking	✓ BMR-based	✗ Limited	✓ Full DB	✓ Yes
Sleep Monitoring	✓ Manual log	✗ Limited	✗ Premium only	✓ With wearable
Water Tracking	✓ Built-in	✗ Not included	✓ Built-in	✓ Built-in
Offline Support	✓ Full offline	✗ Cloud-dependent	~ Partial	✓ Mostly offline
Cross-Platform	✓ Android + iOS	✓ Both	✓ Both	✗ Samsung only
Hardware Required	✓ Phone only	✓ Phone only	✓ Phone only	✗ Galaxy device
AI Personalisation	✓ BMR + Goal AI	✗ Basic only	✗ Manual	~ Limited

✓ = Supported ✗ = Not Supported ~ = Partial | FitPulse scores highest on offline + AI personalisation

Fig 6 Comparison of FitPulse vs. Existing Fitness Applications Across Key Feature Dimensions.

VI. CHALLENGES AND SOLUTIONS

➢ *Step Count Continuity Across Sessions*

The most frustrating bug early in development was step counts resetting to zero every time the app was reopened, even in the middle of the same day. The issue is fundamental to how expo-sensors works: the pedometer only tracks steps within the current app session, with no background counter running between launches. We fixed it by saving the cumulative step total to AsyncStorage on every auto-save, then loading that value back as a baseline offset when the app starts. Subsequent steps for the new session are added on top, giving a correct running total no matter how many times the app has been closed and reopened.

➢ *Package Version Compatibility*

Getting all the dependencies to install cleanly caused more trouble than expected. Running Expo SDK 54 with newArchEnabled: true triggered a cascade of peer dependency conflicts because several packages declared version ranges that did not quite line up with the SDK version we were on. Using the --legacy-peer-deps flag with npm resolved most of it, and switching to npx expo install for Expo-managed packages handled the rest, since that command automatically picks the version compatible with the current SDK. We also had to upgrade to Node.js v20 because metro-config started using Array. prototype. To Reversed(), which was only added in Node 20.

➢ *Offline-First Architecture*

Getting the app to work fully offline required careful thinking around Firebase. The first-ever login does need a network connection, but once authenticated, the token is cached in AsyncStorage and subsequent sessions work without internet. All health data operations — reading, writing, updating — run entirely through AsyncStorage, so they are unaffected by connectivity. The one exception is food search, which calls an external API. We handled this by checking network status before making the request and showing a clear offline message rather than leaving the user staring at a spinner.

VII. FUTURE WORK

➢ *There are Several Things we want to Add to FitPulse Going Forward:*

- Cloud Database Integration — Migrating from AsyncStorage to Firebase Firestore would let users access their data across multiple devices and make it possible to build features like weekly trend charts and long-term progress tracking that need historical data going back further than one device holds.
- AI Food Recognition — Manually typing in every food item is the most friction-heavy part of the app. Connecting a camera-based food recognition API would let users photograph a meal and have the nutrition data filled in automatically.
- Wearable Device Integration — The phone's pedometer is decent, but a Bluetooth connection to a smartwatch would give us richer data — continuous heart rate, blood oxygen, and more accurate calorie burn estimates during exercise.
- GPS Activity Tracking — Adding GPS support would let us track outdoor runs and rides properly, including route maps, distance, pace, and elevation — things the pedometer alone can't give us.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26apr801

- Offline Speech Recognition — A voice input option for food logging would be a big usability improvement, especially during workouts. Using a local speech model like Whisper would keep this working even without internet.
- Push Notifications — Simple reminders — drink water, log lunch, hit your step goal — sent through the Expo Notifications module would help users build consistent habits without having to remember to open the app.

VIII. CONCLUSION

FitPulse does what we set out to build: a single app that covers steps, nutrition, water, sleep, and workouts, runs on both Android and iOS, and does not need a cloud connection or a wearable to function. React Native and Expo kept the codebase manageable across platforms; Firebase handled authentication cleanly; and AsyncStorage gave us offline data persistence without any server infrastructure. The Mifflin-St Jeor engine produces calorie and macro targets grounded in established nutritional science, and the step baseline logic prevents silent resets between sessions. Testing came back at 95.2% overall accuracy with a consistent 60 fps on both platforms. There is clearly more to build — cloud sync, wearable connectivity, camera-based food logging — but the foundation is solid and the architecture is set up to support those additions without major restructuring.

REFERENCES

[1]. World Health Organization, "Physical Activity Fact Sheet," WHO Press, Geneva, 2020. [Online]. Available: https://www.who.int
[2]. S. Consolvo, K. Everitt, I. Smith, J. A. Landay, "Design requirements for technologies that encourage physical activity," ACM CHI 2006, pp. 457–466.
[3]. Google LLC, "Google Fit Platform Overview," Google Developers, 2023. [Online]. Available: https://developers.google.com/fit
[4]. MyFitnessPal, "MyFitnessPal Help Center," Under Armour Inc., 2023. [Online]. Available: https://support.myfitnesspal.com
[5]. K. Mercer et al., "Behavior change techniques present in wearable activity trackers," JMIR mHealth, vol. 4, no. 2, e40, 2016.
[6]. Meta Open Source, "React Native — Learn once, write anywhere," 2024. [Online]. Available: https://reactnative.dev/
[7]. A. Biorn-Hansen, T. Majchrzak, T. Gronli, "Progressive web apps: The possible web-native unifier for mobile development," WEBIST 2017, pp. 344–351.
[8]. M. D. Mifflin et al., "A new predictive equation for resting energy expenditure in healthy individuals," Am. J. Clin. Nutr., vol. 51, no. 2, pp. 241–247, 1990.
[9]. V. Lugade et al., "Validity of using tri-axial accelerometers to measure human movement," Med. Eng. Phys., vol. 36, no. 2, pp. 169–176, 2014.
[10]. Google LLC, "Firebase Authentication Documentation," 2024. [Online]. Available: https://firebase.google.com/docs/auth
[11]. React Native Community, "@react-native-async-storage/async-storage," GitHub, 2024.
[12]. Axios, "Promise-based HTTP Client for Node.js," 2024. [Online]. Available: https://axios-http.com/
[13]. Expo, "Expo SDK 54 Documentation," 2024. [Online]. Available: https://docs.expo.dev/