ChatGPT

Fitness & Nutrition Tracker App – Implementation Plan

This plan details a cross-platform (iOS/Android) React Native app for workout and nutrition tracking. It supports offline use (local DB) with optional cloud backup, prompts for daily weight, logs workouts/food, and provides analytics and AI-driven suggestions. It includes an **exercise library** with dozens of moves (from barbell lifts to bodyweight and machines), a **workout creator/player**, **food logging**, and rich **progress reports**. Animations and videos demonstrate exercise form, and an optional on-device pose model can give live form feedback. Google authentication enables easy Drive backup. The app supports light/dark themes and reminders. An **"About"** screen credits the developer ("Subash") with a button link to his portfolio.

Platforms & Tech Stack

- **React Native with TypeScript** – one codebase for Android and iOS [1]. Use Expo (for rapid prototyping) or React Native CLI (for more control). RN offers native-like performance and smooth UI **animations** and transitions, which is ideal for a polished fitness app [2].
- **UI Libraries** – React Navigation for screen navigation, and either Redux/Redux-Toolkit or Context API for state management. Use `react-native-chart-kit` or `victory-native` for charts (weight, workouts, macros) and `react-native-svg` for custom graphics.
- **Local Database** – **Realm** (or WatermelonDB) for complex offline storage. Realm is fast, handles complex data relations, and has built-in encryption [3]. Its offline-sync ability ensures the app works without Internet [4]. (SQLite is another option but Realm's performance on reads/writes and sync support is advantageous [3].)
- **Storage** – Store media (exercise demo videos/images, progress photos) in app's file storage (use `expo-file-system` or equivalent).
- **Charts & Graphics** – use `react-native-svg` and chart libraries for line/bar/pie charts (weight trends, workout breakdown).
- **Animations & Video** – Embed short demo videos (20–60s) for exercises or use lightweight Lottie JSON animations from a library like LottieFiles to illustrate moves step-by-step. This enhances user understanding of proper form.
- **Google Drive Integration** – Use Google Drive REST API with OAuth2 (via `expo-auth-session` or native modules) for backups. Store the Drive folder ID and file IDs to track snapshots.
- **Notifications** – Use React Native Push Notifications (or Expo's Notifications) to schedule daily weight reminders or workout reminders.

Data Model & Offline Sync

Define the following data entities (stored locally, with optional encrypted backup): - **UserProfile**: id, name, email, phone, DOB, gender, height, settings (JSON: activity level, goal, theme, etc.), created_at.
- **Exercises**: id, name, category (e.g. Barbell/Dumbbell/Bodyweight), primary muscles, equipment, default_sets, default_reps, is_timed (boolean), demo_video_url, cues (step-by-step text cues array).

- **Workouts (Templates)**: id, user_id, title, exercises (ordered list of `{exercise_id, sets, reps, order}`), is_template flag, created_at.
- **WorkoutSessions (Logs)**: id, user_id, workout_id, date, duration_seconds, exercises_done (list of `{exercise_id, sets_performed, reps_per_set, weight_used, rpe, notes}`), heart_rate_avg, calories_burned.
- **FoodEntries**: id, user_id, name, serving_size, calories, protein_g, carbs_g, fat_g, timestamp. (Fetched via API or manually added.)
- **Weights**: id, user_id, weight_kg, timestamp, note.
- **Backups**: id, user_id, drive_file_id, timestamp, size_bytes, encrypted_hash.

Local-first design means all data lives on device. Changes queue up and are used even offline. Upon connectivity (or user command), data can sync to cloud or backup. Use Realm's sync features or manual JSON export/import for backups. Handle conflicts by timestamp or ask user to choose merge vs replace during restore.

Onboarding & Authentication

- **Sign-Up Flow**: On first launch (or account creation), prompt for **Name**, **Date of Birth**, **Gender**, **Email**, **Phone**, **Height** (cm), **Starting Weight** (kg), **Activity Level** (Sedentary–Very Active) and **Goal** (Maintain/Cut (lose) /Bulk (gain) weight). Compute Basal Metabolic Rate (Mifflin–St Jeor) and initial maintenance calories, then adjust for goal. Show these to user with an explanation. *E.g., "Your maintenance calories are ~2,200 kcal. For weight loss, your target might be 2,000 kcal."*
- **Google Sign-In**: Allow login/signup with Google. This simplifies Drive OAuth and multi-device sync. If user signs in with Google, use Firebase Auth or Expo's Google auth to get credentials.
- **Secure Storage**: Store user keys (e.g. JWT, Drive token) in secure storage (Keychain/Keystore or Expo SecureStore).
- **Admin Notification**: After signup, automatically send the collected user details (name, DOB, gender, etc.) to the developer (e.g. by email or a free service). For example, use a free SMTP (Gmail SMTP with a service account) or a Google Apps Script to email a summary, or push the data to a Google Sheet via its API (free). This ensures the developer knows new signups without incurring extra cost. *(Ensure compliance and encrypt data in transit.)*

Exercise Library & Workouts

- **Comprehensive Catalog**: Pre-populate a rich exercise database. Cover the categories listed (barbell lifts, dumbbell/KB, bodyweight, machines, isolation, core, cardio, plyometrics, mobility, balance, etc.). For example, include back squat, deadlift, bench press, overhead press (barbell/DL sets); goblet squat, lunges, rows (DB/KB); push-ups, pull-ups, planks (bodyweight); lat pulldown, leg press (machines); and many isolation moves (curls, raises, etc.). Group them into tabs or categories (e.g. "Compound Lifts", "Gym Machines", "Calisthenics", "Core", "Cardio") for easy browsing.
- **Search & Favorites**: Provide a **search bar** at top of the library to quickly find any exercise by name or muscle. Allow users to **"star" or favorite** exercises. A "Favorites" tab shows their frequently used moves for one-tap access. Users can also **add a new exercise** (custom) by specifying its name, category, default sets/reps/time, and muscle groups – in case something was missing.
- **Workout Builder**: Users can create named workouts by selecting exercises and ordering them. For each exercise in a workout, the app lets them set the target sets, reps (or seconds for timed moves),

and rest intervals. Users can also pick from pre-made templates. When editing a workout, allow easy reordering (drag handles) and duplicating sets.

- **Exercise Selection Flow**: When an exercise is selected (from library, search, or favorites), navigate to an **Exercise Detail** screen showing the exercise's image/video, muscles worked, equipment, and cues. From here, the user adds it to the current workout plan (specifying sets/reps) or starts it immediately.

Figure: Barbell squat demonstration. Each exercise page will include a similar demo image or video clip, plus step-by-step cues (e.g. "Brace core, keep chest up, push knees out"). This helps users perform moves correctly [5]. Users tap **Add Set** or check off sets as they go during a workout, entering actual reps/weight.

Figure: Another squat variation. The library covers all major lifts and moves (barbells, dumbbells, machines, bodyweight, etc.) as per the user's list. Users can **add custom exercises** if needed. In a workout, previously performed exercises are marked; on a new day the **last-used exercises float to the top** for convenience.

Figure: Kettlebell deadlift. Optionally, the app can use on-device pose detection (MediaPipe/TensorFlow) to analyze form. A MediaPipe model detects ~33 body landmarks in real time [6]. By computing joint angles and movement trajectories each frame [7], the app could count reps and even alert if form deviates (e.g. "Back rounding detected"). (This is an advanced feature – by default we provide text/video cues.)

Figure: Push-up exercise. The user sets number of sets and reps (or duration for timed moves like planks) before starting. During a live workout, the screen shows each exercise with a timer or rep counter, plus buttons to enter RPE and notes. Completed sets are checked off, and the next set is queued.

Figure: Barbell deadlift. Heavy compound lifts like these require safety: the app will include “Common Mistakes” tips (e.g. “Keep bar close to shins”, “Engage lats before lift”). Demonstration videos (20–60s) run in-app so users see proper form. The UI remains simple – big images/videos, and clear next/previous controls – ensuring a smooth user experience [2] .

Logging Workouts, Weight & Metrics

- **Workout Sessions**: After a workout, save a session log linking to the Workout template. Record date/time, total duration, and each exercise’s actual data (sets × reps × weight + RPE). Store optional notes and average heart rate (if available from a connected device). Compute total volume per muscle group (weight×reps) for analytics.
- **Daily Weight Log**: Prompt the user (via push or home screen widget) to enter their weight daily. Record these with timestamp. Display a **weight trend chart** (line graph) on the Progress screen. Compute changes week-to-week.
- **Adherence**: Track how many scheduled workouts were completed (e.g. if user planned 4 workouts and did 3). Show adherence % on reports.
- **Macro Tracking**: Each logged food updates daily totals for calories, protein, carbs, fat. Display these vs. targets (e.g. “protein goal: 150g – 120g consumed”). Provide suggestions (e.g. “Add a high-protein snack”).

Nutrition & Food Logging

- **Food Database**: Integrate a public nutrition API (Nutritionix, USDA FoodData, or Edamam). Let users search foods or scan barcodes. The API returns calories and macros per serving. Save entries with name, serving size, and macros. Allow manual entry of custom foods as well.
- **Daily Totals**: Show a Meal Tracker UI (breakfast/lunch/dinner/snacks) where users add foods. Auto-calculate daily totals and compare to the caloric/macronutrient goals set from onboarding.
- **Suggestions**: If the user is under target on macros (especially protein), show quick tips (e.g. images + text “Try a boiled egg or Greek yogurt for extra protein!”).

- *Citation:* “Allow users to log meals and track calorie intake… integrate with third-party APIs and barcode scanning” [8] .

Progress Reports & AI Predictions

- **Weekly/Monthly Summary**: Display charts: (a) **Weight** over time (line graph), (b) **Workouts per Week** (bar chart), (c) **Calories/Protein per day** (bar or line), (d) **Exercise distribution** (pie chart of muscle groups or exercise categories). Also list metrics: total workouts, total volume by muscle, avg workout duration, weight change, best/worst adherence, etc.
- **AI Forecast**: Based on the user’s weight trend, calorie intake, and workout volume, predict weight change next 7/30 days. For example, a simple linear regression on recent weight data can forecast short-term trend. More advanced: train an LSTM or use Facebook Prophet on the time series. Provide explainable suggestions, e.g. “You missed 2 workouts last week, so weight loss may slow; try adding one extra session.” or “Calories are slightly above target; reduce 100–200 kcal to stay on track.”
- **Adaptive Plan Suggestions**: Using historical RPE and PRs, the app can auto-adjust next week’s targets (progressive overload). E.g., “Increase squat weight by 2.5% since last 4 weeks showed consistent progress.” or “Next week add 1 extra rep to each set of bench.” (This can be a server-side ML service called via an API endpoint when the user views the plan.)

Exercise Guidance & Animations

- Each exercise detail includes **cues** (short bullet points), a **checklist of key form points**, and a **demo video/animation**. The animations can be simple frame-by-frame GIFs or Lottie JSONs showing the motion cycle.
- **Pose Checking (Optional)**: Use MediaPipe’s Pose Landmarker (TensorFlow.js) on-device to check form in real-time. For example, while the user does a squat in front of the camera, draw skeleton lines on screen and calculate angles. If knees drift too far in, show “Knee Alignment” warning; if back rounds, show “Keep spine straight”. (Caution: this is compute-intensive and may lag on some devices [9] .)
- **Step-by-step Animations**: For clear understanding, we’ll create short step animations for each compound move. For example:
- Squat: **Set 1** – squat halfway (cue “Sit hips back”), **Set 2** – bottom position (cue “Check knee position”), **Set 3** – standing (cue “Drive through heels”). Each frame can be a static pose image or minimalist animation with annotated arrows (via Lottie). This “breakdown” animation clearly shows the movement phases.
- *Citation:* Using real-time pose estimation, “the model estimates coordinates of 33 body landmarks” to analyze posture [6] and “calculates angles to count reps” [7] .

UI / UX & Theming

- **Home/Today Screen**: Show today’s scheduled workout (exercises list with a “Start Workout” button), a weight entry widget, and a summary card of today’s calories and macros. If no workout planned, suggest one or allow quick “Create Workout”.
- **Navigation**: Use bottom tabs or a drawer for key sections: Home, Workouts, Log Food, Progress, Library, Settings.

- **Dark/Light Mode**: Support both themes. Use React Native's Appearance API or a theming library (Tamagui, React Navigation theming, or styled-components with theme context). Ensure all screens switch colors (background, text, icons) based on system preference or user toggle.
- **Search & Quick Actions**: On the Workouts screen, include a search field to find saved workouts by name. On the Library, the search finds exercises. Add quick-action floating buttons (e.g. "+ New Workout", "+ New Food Entry").
- **Favorites Tab**: In Library or Home, a **Favorites** view lists starred exercises for quick selection.
- **Onboarding Question Flow**: Use multi-step forms (swipeable or paginated) to gather user details at start. Show engaging illustrations or progress bars.
- **Responsive Layout**: Design for different screen sizes; use scrollviews for long content. Ensure large touch targets for inputs/buttons.
- **Reminders**: In Settings, allow the user to enable daily reminders (e.g. "Log Weight at 8 PM"). Use local notifications when the scheduled time arrives.

Backup & Sync

- **Local-First Design**: All data is written to the local Realm (or SQLite) DB immediately. Workouts and logs work offline.
- **Manual Backup**: Provide a "Backup Now" button in Settings. On tap:
- Export database (or JSON dump) and media to a temp folder.
- Encrypt the file (e.g. AES with a key derived from user's password or secure key storage).
- Use Google Drive API to upload the encrypted backup to the user's Drive (in an app folder). Store the Drive file ID and timestamp.
- Show "Last backup: [date], Size: [MB]".
- **Restore**: In Settings, "Restore Backup" lets user pick a file from Drive (list recent backups). The app downloads, decrypts and offers to merge or overwrite local data. Handle key mismatches by asking for the correct password or forcing re-keying.
- **Auto-Backup (Optional)**: If the device and OS allow (Android WorkManager or iOS Background Fetch), schedule a weekly auto-backup on Wi-Fi. Or simply remind the user weekly to back up.
- **Conflict Resolution**: If multi-device use occurs (two phones per account), detect conflicts by comparing timestamps. Offer options like "Use newest" or "Merge workouts".

Third-Party Integrations

- **Google Fit / Apple Health**: (Optional) Sync steps, heart rate, and sleep from FitKit or HealthKit. Also optionally write workouts and weight entries to those platforms. This requires permissions and native bridge libraries.
- **Nutrition APIs**: As above for food data.
- **Authentication**: Google OAuth or Firebase for cloud features.
- **Media (Camera/Barcode)**: Use `expo-barcode-scanner` for food barcodes, and `expo-camera` if adding a photo log feature.
- **Cloud ML (Optional)**: If doing heavy ML predictions, host a lightweight Python/Flask or Node service to analyze user data and return recommendations (behind a secure `/api/ml/predict` endpoint).

Security & Privacy

- **Data Encryption**: Encrypt backups before upload [3]. Keep sensitive keys/tokens in secure storage.

- **HTTPS Everywhere**: All network calls (Drive API, food API) must use HTTPS.
- **Privacy Policy**: Declare how user data is used. The app does *not* send private data off-device except (optionally) to Drive. If AI suggestions are server-side, anonymize data.
- **Compliance**: If giving any diet/workout advice, include disclaimer ("consult physician"). Allow data export (CSV or Realm file) and deletion.

Testing & Deployment

- **Testing**: Unit-test key logic (BMR calc, DB ops, data sync). Write end-to-end tests for critical flows (login, log workout, backup/restore). Use Jest and a React Native test runner.
- **Device Testing**: Test on both iOS and Android devices/emulators (Expo Go and bare builds). Ensure smooth performance and that animations are not janky [10]. Monitor memory/cpu when using Pose detection or large images.
- **App Store Prep**: Follow guidelines for icons, permissions (explain camera, notifications, etc.), and privacy. Set up a release pipeline (Fastlane, App Center or Expo EAS).
- **Metrics**: Integrate analytics (e.g. Firebase Analytics) to track active users, retention, workout frequency. Log events like "Workout Completed" or "Food Logged". Monitor crash reports (Sentry or Crashlytics).

Developer Credit

Include an **About** or **Settings** entry: "App developed by Subash" with a button linking to Subash's portfolio. This gives credit and provides users a way to learn more about the developer.

Sources: This plan incorporates best practices for React Native fitness apps [1] [2], mobile offline storage [4] [3], exercise libraries [5] [8], and pose estimation models [6] [7]. These references guided the app's architectural and feature decisions.

---

[1] [2] React Native Fitness Mobile App Development - Appilian
https://appilian.com/react-native-fitness-mobile-app-development/

[3] [4] Best Local Databases for Offline Functionality in React Native | by ADEEL AHMED | Medium
https://medium.com/@sp22-bcs-040/best-local-databases-for-offline-functionality-in-react-native-b3a6f7278ed7

[5] [8] [10] Build a Powerful Fitness Tracking App Using React Native | JavaScript in Plain English
https://javascript.plainenglish.io/build-a-powerful-fitness-tracking-app-using-react-native-%EF%B8%8F-98b8ad5a333a?gi=9d2a7692ebb3

[6] [7] [9] Fitness App Development with Real-Time Posture Detection using MediaPipe and React - DEV Community
https://dev.to/yoshan0921/fitness-app-development-with-real-time-posture-detection-using-mediapipe-38do