MOMENTUM

Because progress shouldn’t pause.

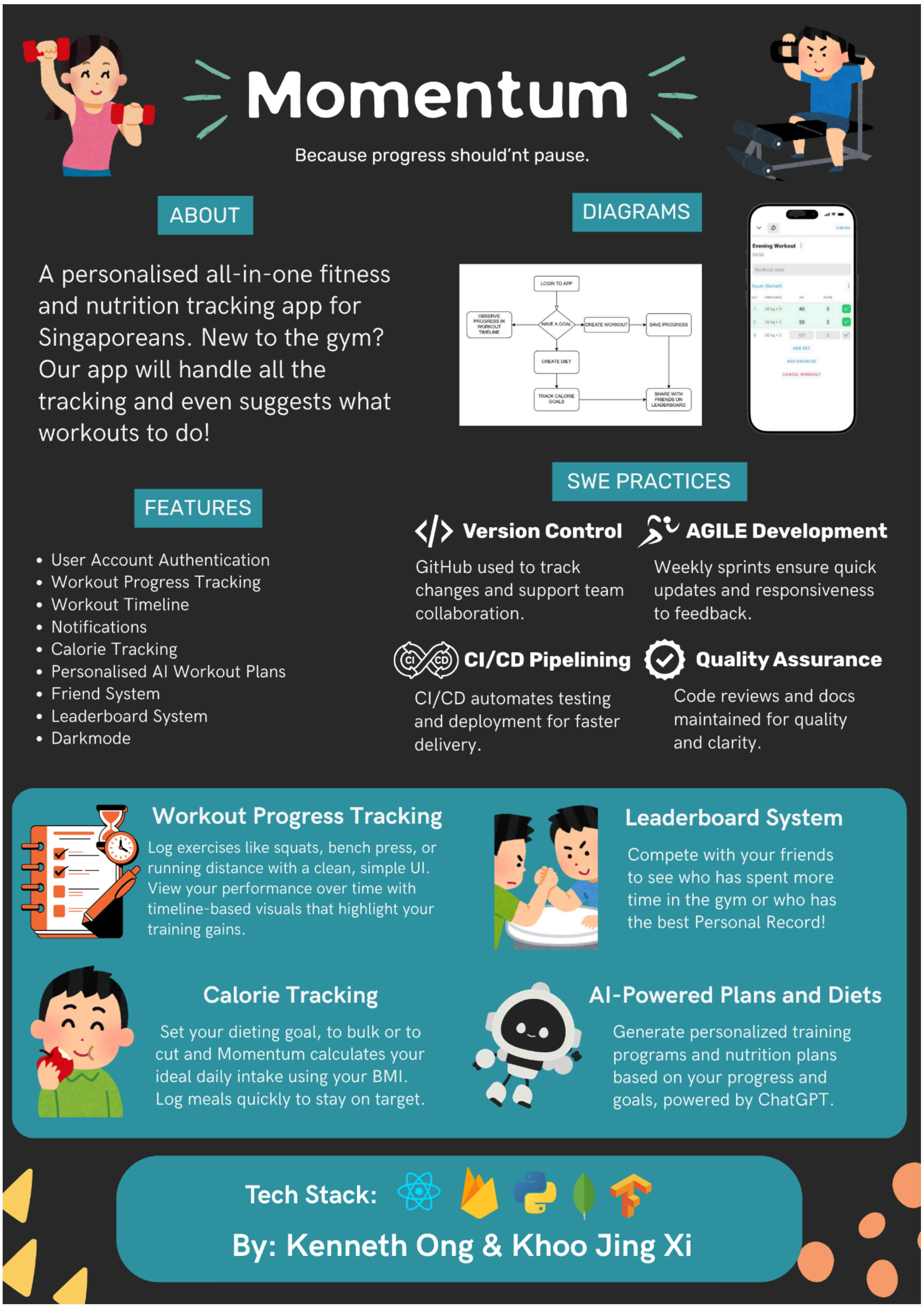
NUS Orbital 2025
Milestone 3

Team Momentum
Khoo Jing Xi
Ong Kwan Kiat Kenneth

Advisor:
Alagappan Ramanathan

Contents

Posters



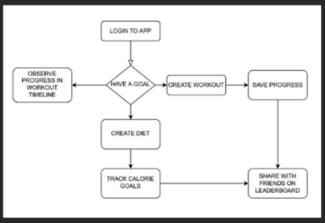
Momentum

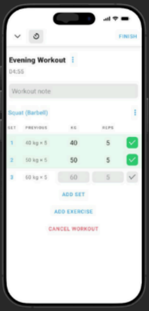
Because progress should'nt pause.

ABOUT

A personalised all-in-one fitness and nutrition tracking app for Singaporeans. New to the gym? Our app will handle all the tracking and even suggests what workouts to do!

DIAGRAMS





FEATURES

- User Account Authentication
- Workout Progress Tracking
- Workout Timeline
- Notifications
- Calorie Tracking
- Personalised AI Workout Plans
- Friend System
- Leaderboard System
- Darkmode

SWE PRACTICES

Version Control

GitHub used to track changes and support team collaboration.

AGILE Development

Weekly sprints ensure quick updates and responsiveness to feedback.

CI/CD Pipelining

CI/CD automates testing and deployment for faster delivery.

Quality Assurance

Code reviews and docs maintained for quality and clarity.

Workout Progress Tracking

Log exercises like squats, bench press, or running distance with a clean, simple UI. View your performance over time with timeline-based visuals that highlight your training gains.

Leaderboard System

Compete with your friends to see who has spent more time in the gym or who has the best Personal Record!

Calorie Tracking

Set your dieting goal, to bulk or to cut and Momentum calculates your ideal daily intake using your BMI. Log meals quickly to stay on target.

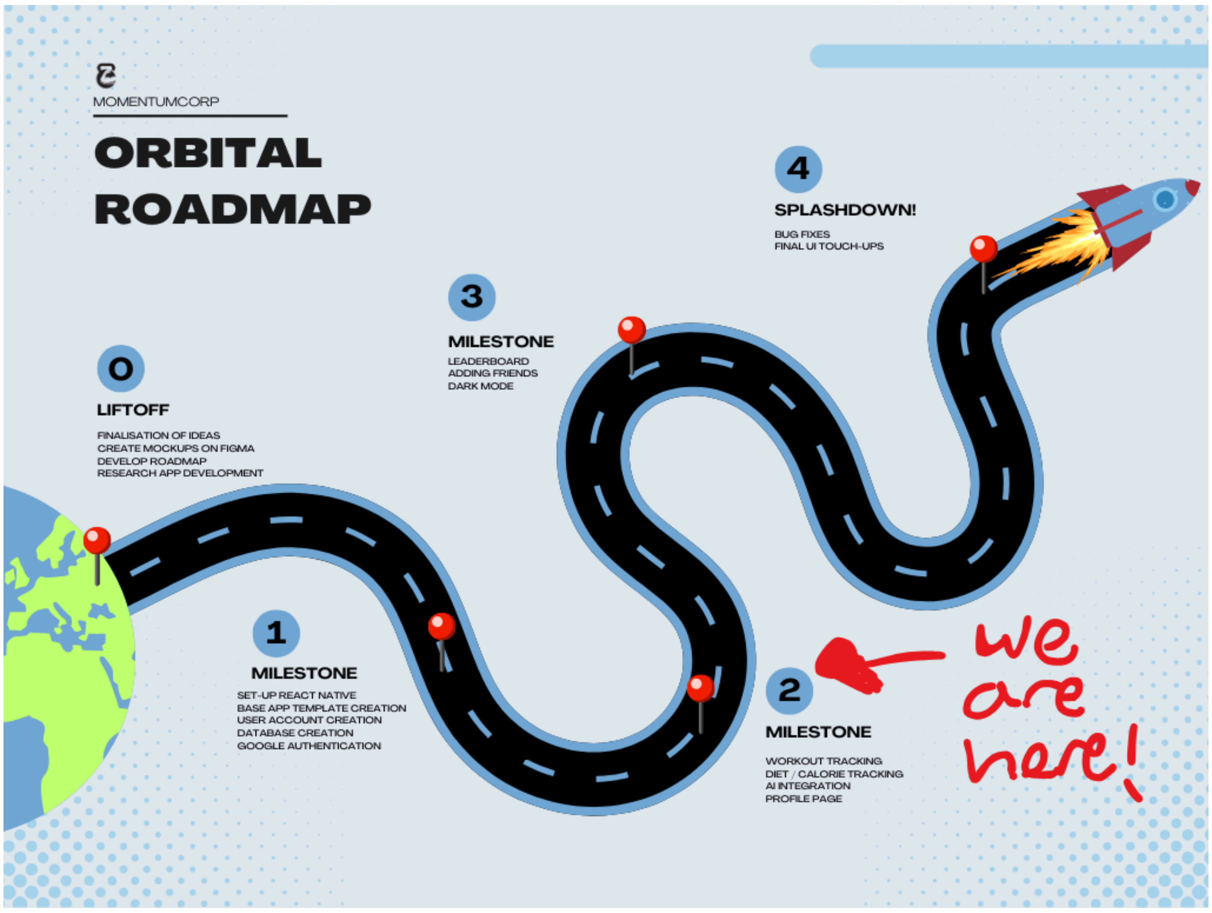
AI-Powered Plans and Diets

Generate personalized training programs and nutrition plans based on your progress and goals, powered by ChatGPT.

Tech Stack:

By: Kenneth Ong & Khoo Jing Xi

Liftoff poster



Milestone 2 poster

Proof-of-Concept

Refer to the video demonstration: Milestone 3 video

Deployment

Instructions

1. Download the Momentum apk.
2. If you have an Android phone, skip Step 3.
3. If you do not have an Android phone, follow the steps in the Expo website to set up Android Studio and emulator on your laptop/computer. (You do not need to set up a development build or Expo Go!)
4. For Android phones, simply run the apk on your phone, and allow all permissions. For Android emulators, drag and drop the apk file from Step 1 onto the screen of the emulator to copy it over.
5. Click on the thumbnail for the app, you can now use Momentum!

Test Account

Email: apptester309@gmail.com
Password: 123123

Disclaimer: The first call of the “record calories” button under Diet Tracking, as well as the “generate plan” button under Ai Workout Plan, may buffer for 50 seconds or more due to the backend server “waking up” from inactivity. Please be patient—the app is **NOT** broken! Also, please **DO NOT** delete or change the password of the test account!

Proposed Level of Achievement

Apollo

Aim

Momentum is a personalised fitness and nutrition tracking app designed specifically for Singaporeans. It **combines workout and meal tracking** with AI-powered suggestions to help users **meet their fitness goals.** The app incorporates social and gamified elements to promote **consistency, motivation, and friendly competition.**

Motivation

While the fitness and wellness industry has seen an explosion of mobile applications aimed at helping users track their workouts, calories, and goals, many of these tools are built with a one-size-fits-all mindset. This creates a significant gap, especially in culturally diverse markets like Singapore, where dietary habits, lifestyle routines, and even fitness goals can differ greatly from Western norms. For example, most gym or nutrition tracking apps often include food databases skewed toward Western cuisine, leaving common local dishes such as nasi lemak or chicken rice unaccounted for. This disconnect results in inaccurate or incomplete logs, which can discourage users from maintaining long-term use.

Moreover, fitness tracking is not just about logging numbers. It is about understanding one's body, routine, and motivation. Many users struggle with maintaining consistency due to a lack of personalization and real-time feedback. Generic recommendations or rigid plans do not work for everyone — especially in fast-paced environments like Singapore where users juggle multiple responsibilities and erratic schedules. In this context, rigid gym routines and fitness plans can become unsustainable, causing users to abandon their goals prematurely.

With the rise of mental health awareness, there is also growing recognition of the link between physical and psychological well-being. However, many apps still treat fitness as an isolated component, without considering how stress levels, sleep quality, and daily habits impact workout performance and goal achievement. Momentum seeks to bridge this gap by integrating holistic wellness tracking with personalization that adapts to local needs and individual patterns.

The pandemic further emphasized the need for flexibility and accessibility in wellness tools. As home workouts and digital fitness communities became the norm, users began seeking apps that could not only provide technical tracking but also offer motivation, variety, and adaptability. Yet, the majority of apps still lack cultural sensitivity, local relevance, or motivation features beyond basic reminders.

Momentum is designed to be more than a tracking tool — it is a companion that understands local culture, fluctuating motivation, and the user's evolving needs. By allowing users to track gym routines, nutrition, rest, and habits within a single localized platform, we empower them to make sustainable lifestyle changes. Momentum is about helping users achieve consistency and growth — not through perfection, but through personalization, encouragement, and real context.

Vision

Momentum envisions a future where fitness and nutrition tracking is not only effective but also culturally relevant and deeply personalized. As a comprehensive app tailored for Singaporeans, Momentum seeks to eliminate the barriers that often discourage users from maintaining a healthy lifestyle — such as unfamiliar food databases, rigid workout plans, and a lack of local relevance.

By integrating features like **AI-generated training plans**, **calorie tracking adapted to local dishes**, and a **workout timeline visualizer**, Momentum allows users to monitor their fitness journeys in a holistic and intuitive way. Whether the user is a beginner seeking guidance or a seasoned gym-goer tracking personal records, the app adapts to various goals and levels of experience.

Momentum aims to redefine fitness tracking by combining **user progress visualization** with community elements like a **leaderboard system** and **friend interaction**. These features not only motivate users through competition but also promote consistency through accountability and support. Our vision includes creating a platform where users can celebrate small wins, monitor gains over time, and remain engaged through short, gamified sprints.

Through **quantitative insights** like BMI-based recommendations and training milestones, alongside **qualitative motivation** via AI-curated suggestions and personalized encouragement, Momentum empowers users to build sustainable habits. By applying software engineering best practices—like Agile development, CI/CD pipelining, and rigorous quality assurance—we ensure that our product evolves rapidly to meet user needs while maintaining reliability and performance.

Ultimately, Momentum strives to be more than an app—it aims to be a fitness companion that grows with users, motivates them through data-driven feedback, and supports them with localized, intelligent tools. Our goal is to help every user take control of their health, stay consistent, and progress without pause.

User Stories

1. As a beginner gym user, I want to receive AI-generated workout and meal plans based on my fitness goals so that I know where to start without feeling overwhelmed.

2. As a beginner, I want to follow a step-by-step tutorial that guides me through my first few workouts so that I feel more confident using gym equipment.

3. As a new user, I want to track my exercises and food intake easily with a clean and simple logging interface to develop consistent habits.

4. As a casual user, I want to view my workout progress over time using visuals so that I stay motivated to keep pushing.

5. As someone new to fitness, I want to learn the difference between bulking, cutting, and maintaining through helpful in-app tips and educational prompts.

Extensions

6. As an experienced gym user, I want to customise my workout and meal plans with advanced options and override AI suggestions when needed.

7. As a long-term user, I want to sync my workout data across multiple devices to preserve my progress and history for long-term tracking.

8. As a social user, I want to challenge friends and track our progress on a shared leaderboard so that we can compete and stay motivated together.

9. As a social user, I want to get notified when my friends complete workouts or hit milestones so that I can support or congratulate them.

Features

User Account Authentication [Completed]

Description

As Momentum is a personalised fitness and nutrition tracking app, each user is required to authenticate with a unique account in order to access their personal progress data, goals, and AI-generated recommendations.

Momentum has implemented 2 main sign-in methods:

1. **Email and password**

2. **Sign in via Google**

Utilising the Firebase SDK Authentication, users can:

- Sign up with an email and password
- Sign in using either an email/password or Google account
- Reset their password via email reset link
- Seamlessly continue their session with persistent login

When users register or log in, the relevant Firebase function (`createUserWithEmailAndPassword` or `signInWithEmailAndPassword`) is called. For Google login, we integrate `@react-native-google-signin/google-signin`, which handles both Android and iOS token management.

Input errors such as invalid email formats or mismatched passwords are handled in real-time with inline error messages and fallback `Alert` dialogs to provide immediate feedback. Animated transitions are used for a polished and responsive user experience during onboarding.

Only upon successful login or registration will the user be redirected to the main app interface (`/(tabs)/home`) where their data and preferences are loaded securely via the AuthContext.

Implementation philosophy

Within our authentication logic, a persistent listener checks whether the user is already signed in when the app is launched. If authenticated, the user is immediately redirected to the Home screen. If not, they are routed to the authentication flow (`/(auth)/login` or `/(auth)/register`).

During email registration, users are required to enter and confirm their password. Error states such as invalid emails or mismatched passwords are shown inline beneath each field, improving usability and reducing user frustration.

We adopt a clean UI design backed by Google's guidelines and real-time validation to reduce failed attempts. Each authentication screen also supports animated loading indicators, Google Sign-In integration, and safe area insets for better layout on all devices.

This approach ensures both new and returning users enjoy a seamless experience, with error handling and onboarding paths designed for flexibility and clarity.

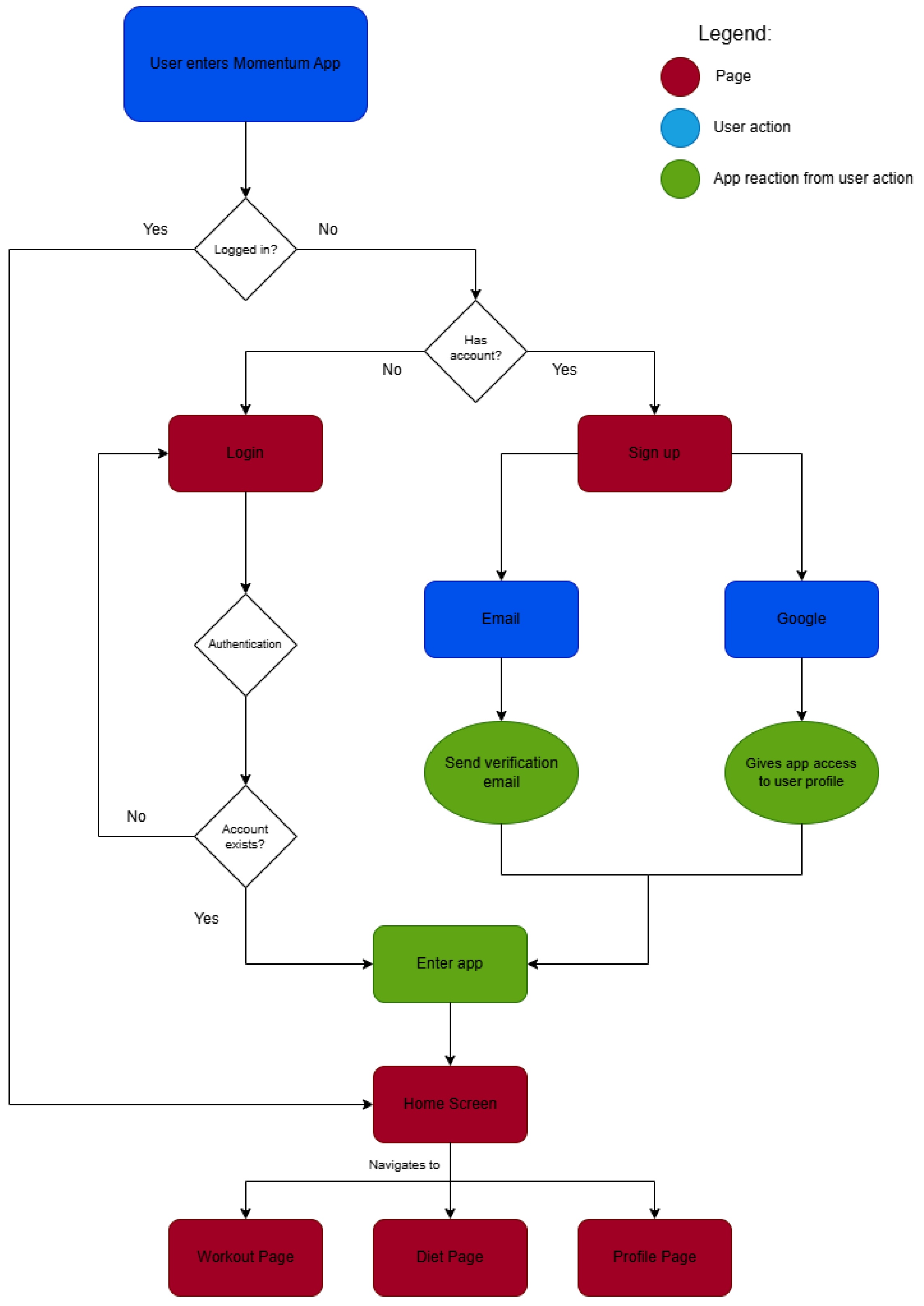
Implementation challenges

One key challenge in implementing authentication was managing screen redirection logic when using multiple sign-in methods. Initially, there were conflicts between manual navigation triggers (e.g. `router.push("/(auth)/register")`) and the authentication listener that redirects logged-in users to the Home screen.

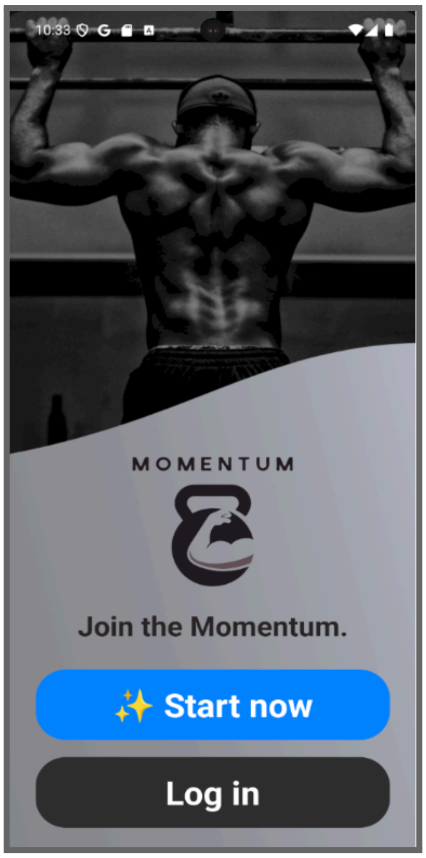
For example, after a successful registration, the app navigated users to the onboarding page, while the listener would simultaneously redirect to Home, causing race conditions. This was resolved by implementing the redirection exclusively inside the listener after verifying if onboarding was completed, ensuring consistent user flows regardless of sign-in method (email or Google).

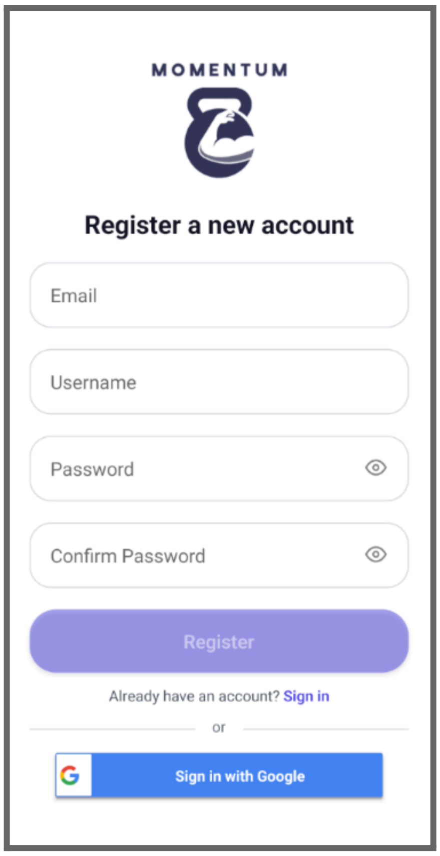
We also handled token refresh and rehydration of the AuthContext so that users do not need to re-authenticate unnecessarily, even after closing and reopening the app.

Diagrams

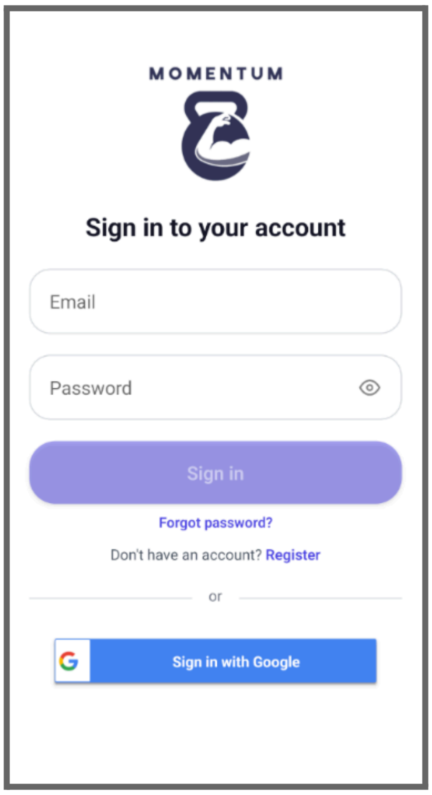


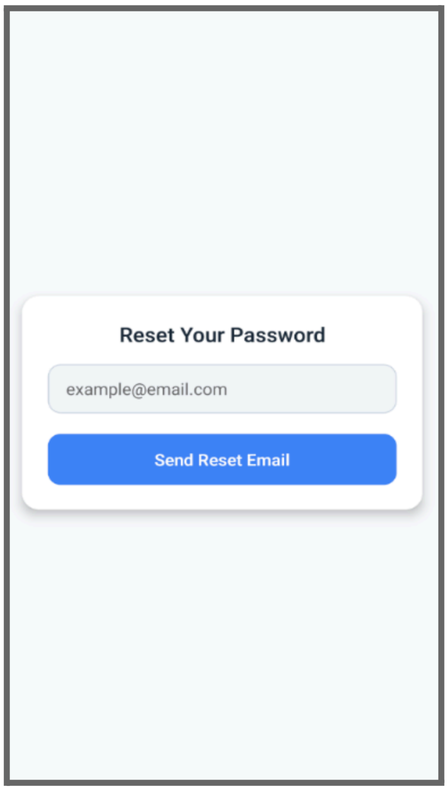
Authentication Flow





Welcome Page and Sign Up Page





Login Page and Reset Password Page

Workout Tracking [Completed]

Description

Workout tracking is a core feature of *Momentum*, enabling users to log, store, and view their physical training sessions in a structured, visual, and highly interactive manner.

The feature is split across three main screens:

1. **WorkoutTracking** – The entry point where users choose to view past workouts or record a new one.

2. **WorkoutSubmit** – A form-based interface where users input workout details such as exercise name, duration, sets, reps, and weight.

3. **WorkoutHistory** – A visual timeline of past workouts, grouped by date, with support for infinite scroll and animated card displays.

Workout logs are stored in Firebase Firestore under a nested path based on the user's unique ID (`Users/{uid}/workouts`). Each workout entry includes:

- Workout name (with dropdown suggestions from a preset list)
- Duration (hours, minutes, seconds)
- Sets and reps
- Weight lifted
- Timestamp

Workout inputs are validated in real time. Users can only submit valid workouts from a predefined list, and input fields reject invalid values (e.g. non-numeric reps or weight). Upon successful submission, users receive a confirmation alert and the form resets for the next entry.

Workout history is displayed with date dividers, intuitive card layouts, and animation effects. The system also includes lazy loading (pagination) to fetch more entries when the user scrolls to the bottom, ensuring performance is maintained with large datasets.

Implementation philosophy

The goal of this feature is to keep workout tracking simple, responsive, and motivating.

- The **WorkoutTracking** screen provides a clean and fast decision point with two options: view history or record a workout. Animations help guide attention.

- The **WorkoutSubmit** screen is form-centric, with dropdown search for workout names and segmented time inputs for better control. Firebase Firestore is used for storing workout logs in a structured manner under the current user's document.

- The dropdown is filtered in real time as the user types. If the selected workout is not from the preloaded list, the app prevents submission and prompts an error.

- Upon submission, the workout is timestamped using `serverTimestamp()` to ensure consistent ordering and display across devices.

- The **WorkoutHistory** screen organizes entries by shifted day keys (grouping workouts completed after midnight under the previous day), improving clarity for night-time users. Each workout is animated into view using `react-native-reanimated`, making the history screen feel alive and rewarding.

This modular split between navigation, input, and visualization ensures clean logic separation and reusability, while maintaining a fluid user experience.

Implementation challenges

The biggest challenge was managing real-time filtering, scroll-based pagination, and animated rendering within the **WorkoutHistory** screen.

1. **Dropdown Filtering in WorkoutSubmit:**
   The workout name input dynamically filters a preset list of exercises. Handling edge cases like empty inputs, invalid selections, and preserving clean UX with overlays and positioning required careful layout measurements and event handling.

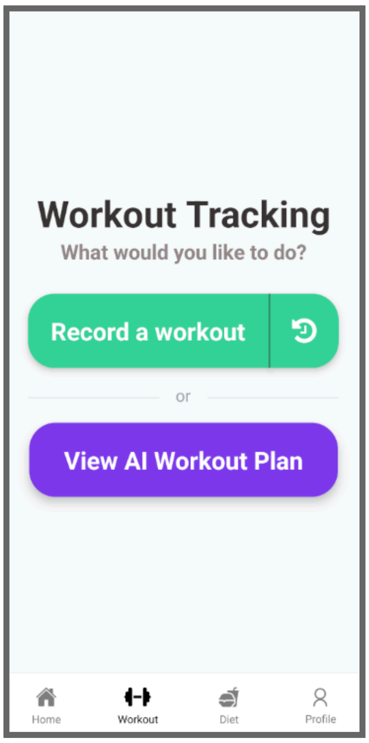
2. **Firestore Pagination:**
   In WorkoutHistory, workouts are fetched 10 at a time using `startAfter()` and `orderBy(timestamp)`. Ensuring the infinite scroll doesn't repeat data or miss entries required accurate `lastDoc` tracking and scroll position management via `onScroll`.

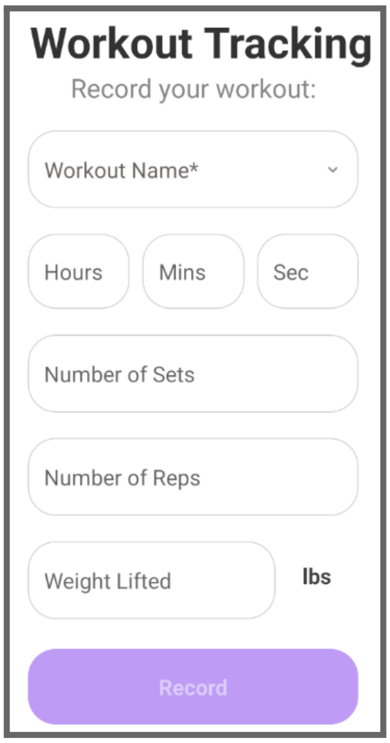
3. **Animation & Performance:**
   Each workout card uses `FadeInLeft` with delay logic based on its index to create a cascading effect. This required balancing animation fluidity with rendering performance, especially when the list grows long.

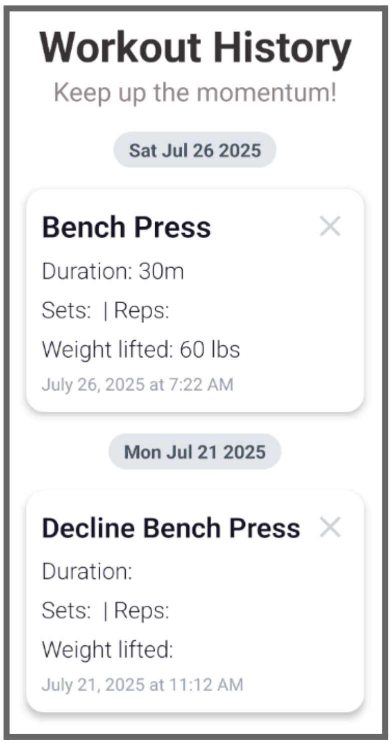
4. **Date Grouping Logic:**
   To better reflect gym habits, workouts logged between midnight and 5 AM are grouped under the *previous* day. This adds nuance to the UI and avoids confusing users with unexpected dates.

Despite these challenges, the final implementation is polished, responsive, and fully functional. It encourages consistent workout tracking while keeping the experience enjoyable and visually rewarding.

Diagrams






Workout History

Keep up the momentum!

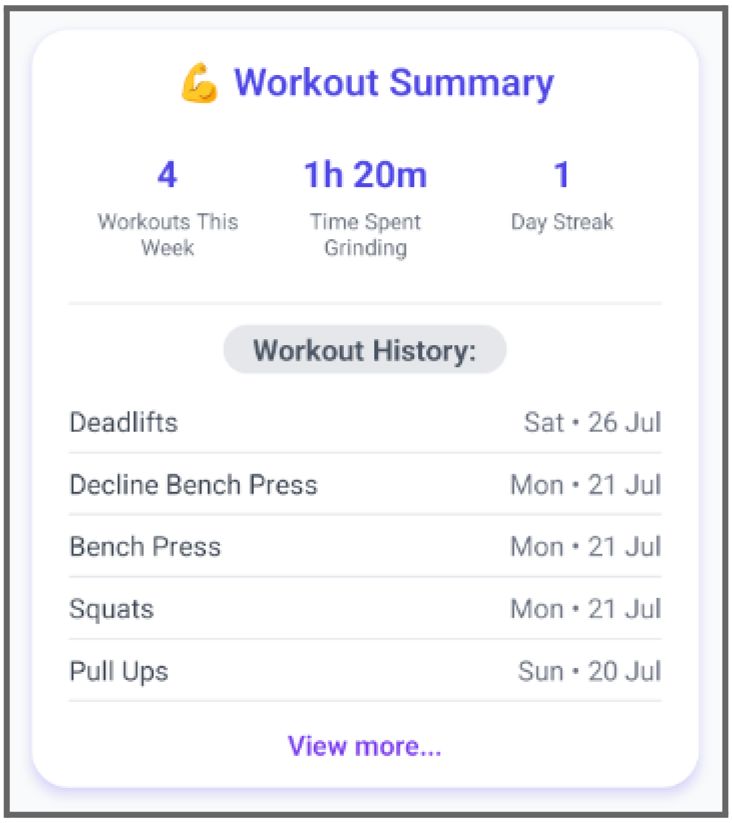
Sat Jul 26 2025

Bench Press
Duration: 30m
Sets: | Reps:
Weight lifted: 60 lbs
July 26, 2025 at 7:22 AM

Mon Jul 21 2025

Decline Bench Press
Duration:
Sets: | Reps:
Weight lifted:
July 21, 2025 at 11:12 AM

Workout Tracking Home Page, Workout Submit Page, and Workout History Page


💪 Workout Summary

4	1h 20m	1
Workouts This Week	Time Spent Grinding	Day Streak

Workout History:

Deadlifts	Sat • 26 Jul
Decline Bench Press	Mon • 21 Jul
Bench Press	Mon • 21 Jul
Squats	Mon • 21 Jul
Pull Ups	Sun • 20 Jul

View more...

Workout Summary Card on Home Page

Diet Tracking [Completed]

Description

The Diet Tracking feature in Momentum empowers users to manage their nutrition by offering a seamless workflow to:

1. **Set a calorie target** (either in deficit or surplus mode)
2. **Record daily meals** and estimate calorie values via a backend API
3. **Visualize daily progress** using a circular progress indicator
4. **Review food history** via an animated, paginated list grouped by date

All diet data is tied to the authenticated user and stored in Firestore under:

- `/Users/{uid}/dietTarget` for the calorie target and goal type
- `/Users/{uid}/diet` for daily food entries with timestamps

The dashboard (`dietTracking.tsx`) shows users their remaining calorie allowance based on entries for the current day (5AM–5AM window). The value is dynamically adjusted and visually displayed through a progress ring. Messaging adapts based on goal type (deficit vs surplus), clearly indicating over- or under-consumption.

Users can:

- **Record a meal** through `dietSubmit.tsx`, which uses a backend AI model to calculate calorie estimates from natural language inputs (e.g. “1 bowl of laksa”).
- **Set a new calorie target** via `dietTarget.tsx`, selecting either a surplus or deficit mode and entering a daily target.
- **Review their diet history** in `dietHistory.tsx`, where entries are grouped by date and presented with animated cards and infinite scrolling.

Implementation philosophy

The Diet Tracking system emphasizes **local relevance, ease of use, and visual feedback**.

- On opening the diet tab, users are shown a **circular progress indicator** with remaining calories and feedback text tailored to their selected mode.
- The food entry form accepts **free-text inputs**, enabling flexibility and simplicity. All food logs are timestamped and submitted to Firestore with a calorie estimate from a server-side API using OpenAI.

- Targets are stored separately to allow **dynamic updates without affecting past entries**.
- Entries before 5AM are attributed to the *previous day* to better reflect late-night eating habits common in the region.
- The history view includes **date dividers** and **pagination** using Firestore's `startAfter`, ensuring scalable performance and logical grouping.

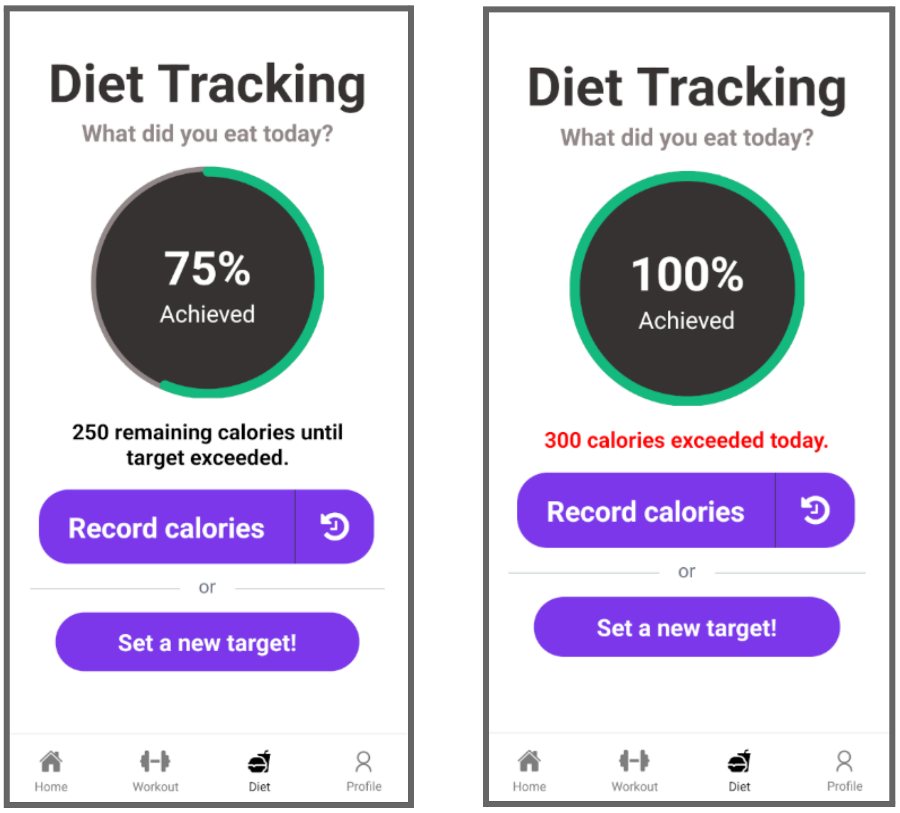
Together, these components ensure that nutrition tracking is not only data-rich and personal but also enjoyable and frictionless.

Implementation challenges

Several challenges arose during development of this module:

1. **AI-backed calorie estimation:**
   Integrating the backend API required careful formatting of requests and handling varied natural language inputs. For instance, "half a cup of brown rice" and "2 pieces of chicken satay" both needed accurate parsing.
2. **User-specific calorie targeting:**
   Maintaining separate states for deficit vs surplus while adjusting messaging logic and ring coloring required clear mode tracking. The `goalType` is stored in Firestore and dynamically influences frontend rendering.
3. **Date shifting and grouping logic:**
   For both the progress view and history logs, a custom timestamp handler shifts entries before 5AM to the previous day. This avoids visual clutter and aligns better with user expectations.
4. **Animated infinite scroll:**
   In `dietHistory.tsx`, each food entry is animated using `FadeInLeft` with staggered delays. Infinite scroll was implemented with Firestore pagination, requiring precise tracking of the last document fetched to avoid duplicates or skips.
5. **Form validation and feedback:**
   Real-time input validation, error messaging, and `ActivityIndicator` feedback were added across all submission points to ensure smooth UX and prevent duplicate writes.

Diagrams

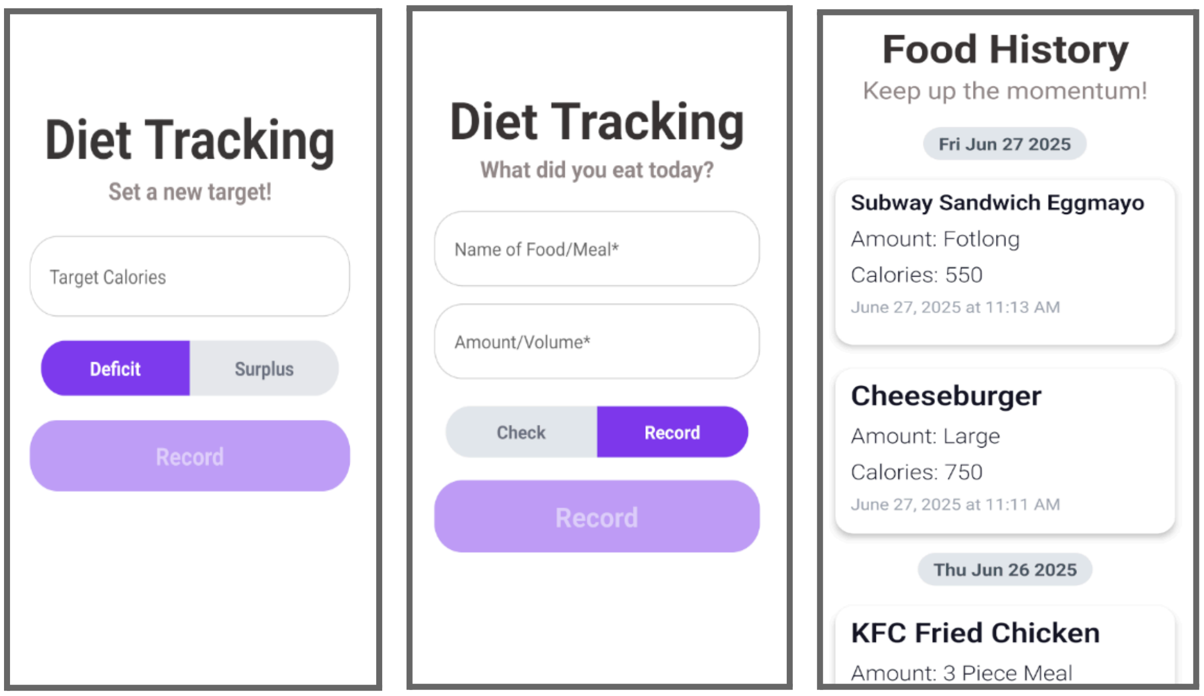

Diet Tracking

What did you eat today?

250 remaining calories until target exceeded.

300 calories exceeded today.

Diet Tracking Home Page (Before/After exceeding target)


Diet Tracking

Set a new target!

Diet Tracking

What did you eat today?

Food History

Keep up the momentum!

Fri Jun 27 2025

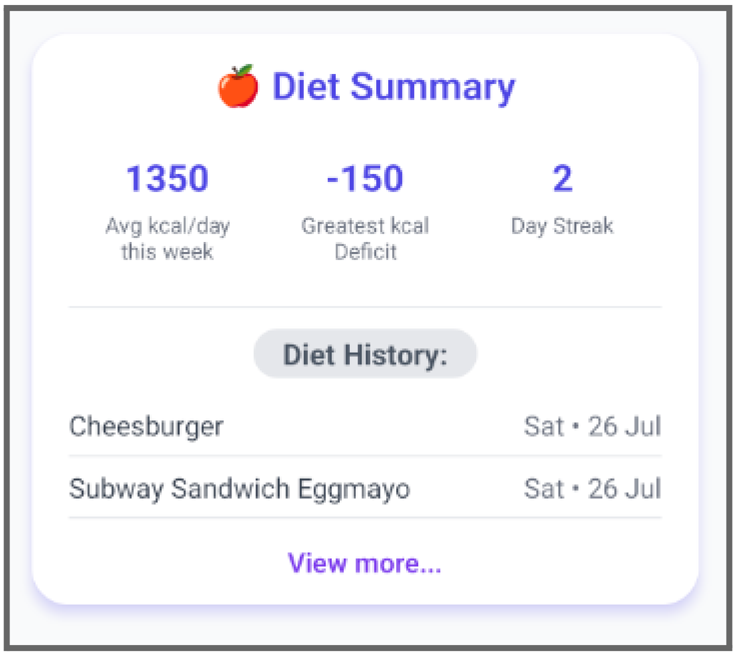
Subway Sandwich Eggmayo
Amount: Fotlong
Calories: 550
June 27, 2025 at 11:13 AM

Cheeseburger
Amount: Large
Calories: 750
June 27, 2025 at 11:11 AM

Thu Jun 26 2025

KFC Fried Chicken
Amount: 3 Piece Meal

Calorie Target Setting Page, Diet Recording Page, and Diet History Page



Cheesburger	Sat • 26 Jul
Subway Sandwich Eggmayo	Sat • 26 Jul

Diet Summary Card on Home Page

Friend System [Extensions Completed]

Description

The Friend System in Momentum introduces lightweight social connectivity, allowing users to build supportive networks through mutual connections. Users can send and accept friend requests, view their friends' profiles, and optionally explore their public workout or diet history.

The Friend System includes three primary screens:

1. **Profile** – Displays the user's profile, including their streak and workout stats, and provides navigation to the Friends page.

2. **FriendsPage** – Allows users to search by username, send/accept/reject friend requests, view current friends, and remove connections.

3. **FriendProfilePage** – Displays another user's public profile, including streak, workout activity stats, and optional buttons to view workout/diet history (depending on the user's privacy settings).

User data is retrieved from Firebase Firestore under each user's subcollections and metadata documents (e.g., `Users/{uid}/streak/tracking` and `userData/{uid}`). Real-time updates are reflected through efficient use of `useEffect`, `useCallback`, and pagination for scalable performance.

Users can:

- **Search for users** by username and send a friend request through `FriendsPage.tsx`, which connects users to others in the Momentum network.
- **Accept or reject friend requests** via the Requests tab, adding mutual friends to both users' profiles.
- **View a friend's public profile** in `FriendProfilePage.tsx`, which shows workout streak, session count, and workout time stats.
- **Access a friend's workout or diet history** if permitted, through navigation buttons that route to `WorkoutHistory.tsx` or `DietHistory.tsx`.

Implementation Philosophy

The Friend System was built with a focus on simplicity, privacy control, and scalable real-time access. Key principles guiding the development include:

- **Mutual Control**: Friendships require two-way consent, enforced through request/accept workflows.
- **Separation of Concerns**: Profile rendering is decoupled from friend list logic to keep components modular and reusable.
- **Privacy by Design**: Workout and diet history are hidden by default and controlled through explicit boolean flags.
- **Responsive Design**: The system uses avatar rendering, real-time loading indicators, and smooth UI transitions for a polished experience.
- **User-Centric Navigation**: All screens support intuitive back navigation and consistent transition logic via `expo-router`.

Implementation Challenges

Several challenges were addressed during the development of the Friend System:

1. **Efficient Friend Search**
   Searching for users by username required debouncing and handling null results gracefully. The `searchUserByUsername()` utility performs direct lookups in the Firestore database while ensuring that failed searches return appropriate alerts.

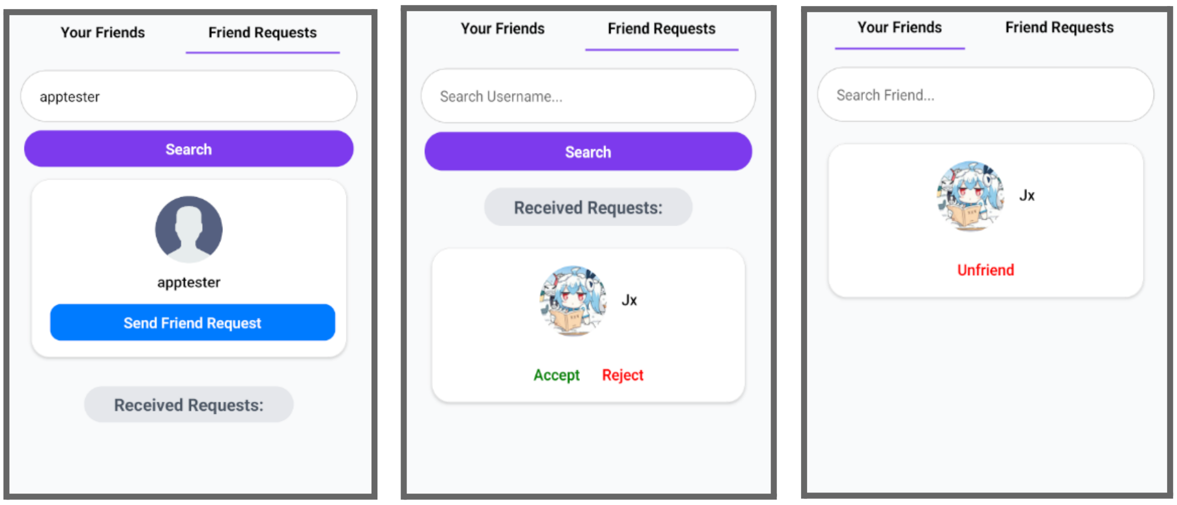
2. **Mutual Relationship Handling**
   Managing bi-directional friendship involved careful coordination of request and acceptance logic. Accepting a request adds both users to each other's friend lists, while rejecting removes the pending entry without side effects.

3. **Profile Access with Permissions**
   Friend profiles show workout and diet access conditionally, based on `showWorkoutHistory` and `showDietHistory` fields. Handling absent or outdated fields in `userData` documents required fallback logic to avoid rendering errors.

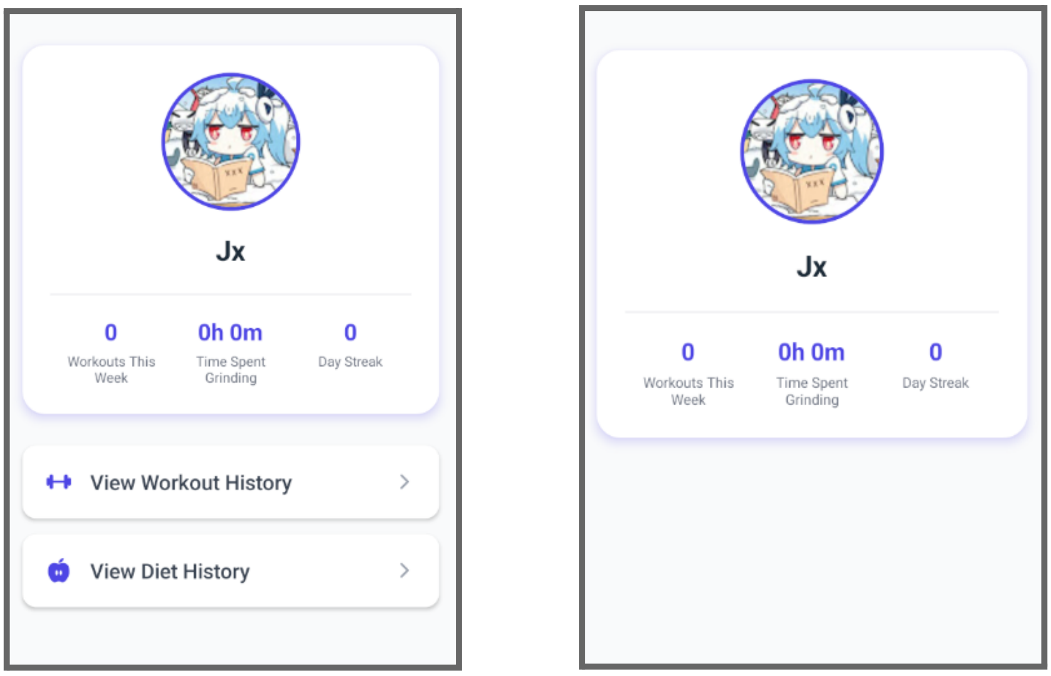
4. **Firebase Data Fetching and Pagination**
   While friends are loaded in batch, each individual profile (username, avatar) requires a secondary fetch using `getUserData()`. This added complexity to loading states and required strict control over async mapping and error handling.

5. **UI Responsiveness and Routing**
   Profile navigation with `router.push()` required proper parameter passing to render avatars and usernames without delays. Rendering large friend lists and requests involved flattening structures and managing conditional UI states for each tab.

Diagrams



Friend Request Page (Sending/Receiving) and Current Friends Page



Friend Profile Page (With Privacy Permissions On/Off)

Leaderboard System [Extensions Completed]

Description

The Leaderboard System introduces a lightweight and focused competitive element to the Momentum app by ranking users and their friends based on workout performance. Users can filter the leaderboard by workout type and rank metric to see how they compare within their immediate network.

The leaderboard is displayed on the **Home screen (`home.tsx`)** and appears only after both filters are selected:

- **Workout type** (e.g., Squats, Deadlifts)
- **Ranking metric** (either **Time Spent** or **Max Weight**)

Upon selection, the app dynamically queries the relevant workout data from Firebase Firestore, sorts and ranks the user and their friends, and displays the top 10 entries.

Users can:

- **Filter by workout type** using a searchable dropdown component
- **Rank by Time Spent** (duration) or **Max Weight** lifted
- **View the top 10 performers** among themselves and their friends for each filter combination
- **See rankings update** in real time as workouts are logged or filters are changed
- **Identify leaders by placement icons** (🥇, 🥈, 🥉), enhancing motivation and visibility

Implementation Philosophy

The leaderboard was built with an emphasis on:

- **Clarity**: Showing only one ranking metric at a time to avoid clutter
- **Control**: Rankings are updated only when both filters are selected, reducing noise
- **Performance**: Limits data fetch to current user + friends only, improving responsiveness
- **Visual Hierarchy**: Uses placement emojis (🥇, 🥈, 🥉) and clean UI to highlight performance
- **Modularity**: The ranking logic is encapsulated in `updateLeaderboard()` for easy extension

Implementation Challenges

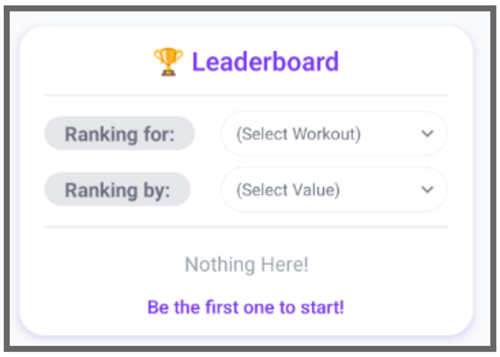
1. **Filter Dependency Management**
   The leaderboard only displays once both filters (workout type and rank metric) are selected. This required careful state handling to avoid premature queries or blank states.

2. **Duration Comparison Logic**
   To rank by `Time Spent`, durations were parsed and compared by converting to total seconds, then re-formatted back to `xh ym zs`. This ensured accuracy while maintaining readability.

3. **Data Fetch Across Users**
   The leaderboard fetches each friend's profile and workouts individually. This involved multiple asynchronous calls within a loop, requiring robust error handling and batching logic.

4. **UI Responsiveness with Conditional Rendering**
   To ensure the interface remained responsive and user-friendly, loading states were carefully managed for both the leaderboard and filter dropdowns.

Diagrams



🏆 Leaderboard

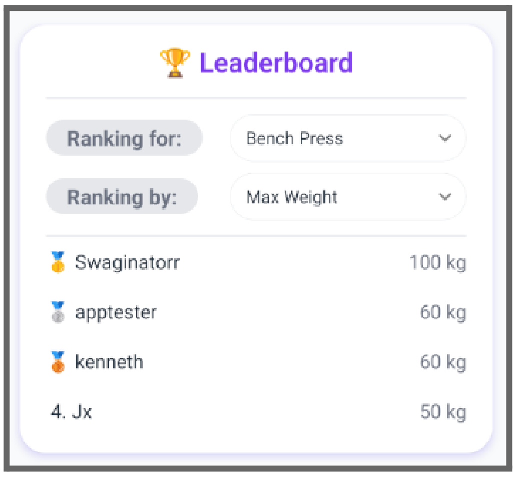
Ranking for: (Select Workout)

Ranking by: (Select Value)

Nothing Here!

Be the first one to start!

Leaderboard on Home Page (Before Selecting Workout)

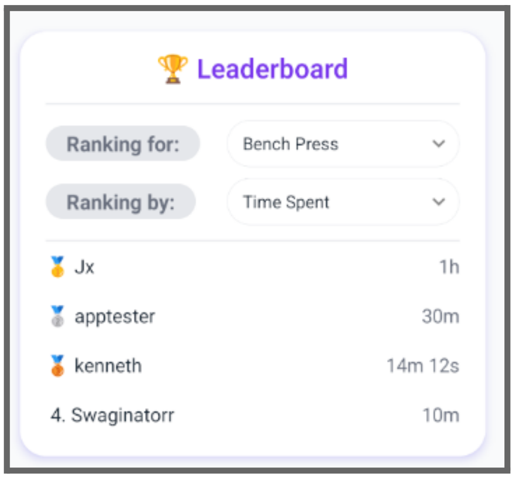


🏆 Leaderboard

Ranking for: Bench Press

Ranking by: Max Weight

🥇 Swaginatorr	100 kg
🥈 apptester	60 kg
🥉 kenneth	60 kg
4. Jx	50 kg



🏆 Leaderboard

Ranking for: Bench Press

Ranking by: Time Spent

🥇 Jx	1h
🥈 apptester	30m
🥉 kenneth	14m 12s
4. Swaginatorr	10m

Leaderboard on Home Page (After Selecting Workout and Filtering by Weight/Time)

AI-Powered Workout Plan [Extensions Completed]

Description

The AI-Powered Workout Plan module in Momentum introduces personalized fitness planning through natural language input, enabling users to instantly receive tailored workout plans using artificial intelligence. This feature transforms user-submitted goals (e.g. “I want to lose belly fat in 4 weeks” or “I only have 20 minutes a day to work out”) into structured, actionable workout routines.

The module simplifies fitness planning into two key screens:

1. **Generate a Plan** via a prompt-driven interface powered by a backend AI model

2. **View or Update Plan** from a dedicated dashboard card with dynamic theming and animations

All data is securely stored in Firestore under:

- `/Users/{uid}/plans/workoutPlan` – containing the latest saved plan and timestamp

The user experience begins with the `generatePlan.tsx` screen, where users are encouraged to describe their fitness goals or constraints in free-form text. Once submitted, the prompt is sent to a backend endpoint, which returns an AI-generated workout description. The result is presented in a styled container, allowing users to save it or regenerate a new plan. Upon saving, the plan is persisted in Firestore and becomes viewable from `workoutPlan.tsx`.

This streamlined experience gives users control over their workout strategy without requiring technical knowledge of fitness science. Whether seeking strength gains, endurance, or weight loss, users can iterate quickly through the generate-save-update loop with just a few taps.

Users can:

- **Generate a custom workout plan** through `generatePlan.tsx`, by entering a natural language prompt describing their fitness goal (e.g. "build core strength in 10 minutes daily").

- **Save the generated plan** to Firestore, where it is stored under their user profile.

- **View the saved plan** in `workoutPlan.tsx`, which automatically fetches and displays the most recent plan.

- **Update the plan** by navigating back to `generatePlan.tsx`, where they can regenerate and save a new one.

Implementation Philosophy

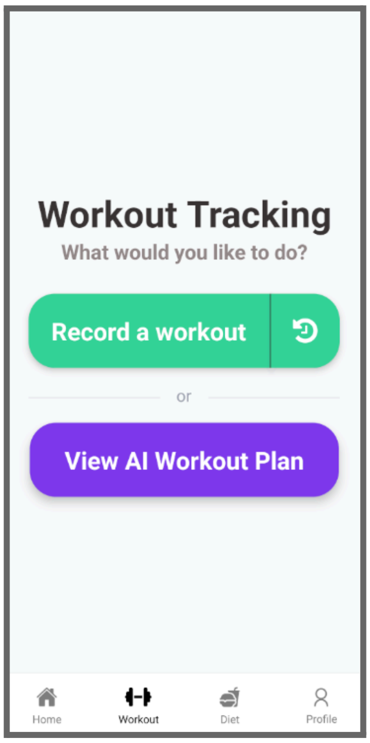
This feature emphasizes:

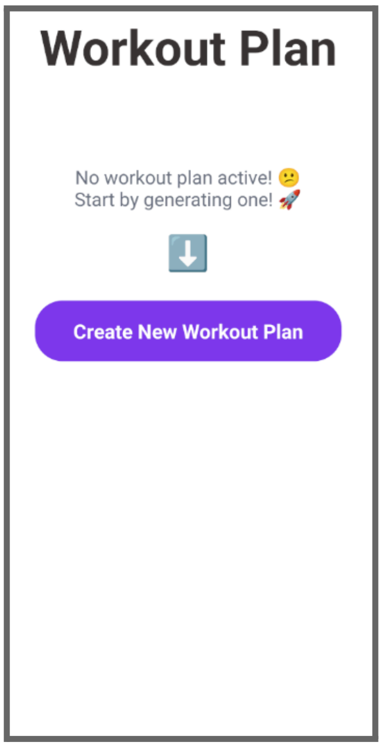
- **Natural UX Flow**: The two-screen flow (generate → view) simplifies complex AI interaction into a seamless cycle that feels familiar to users.
- **Thematic Consistency**: Styled with full dark mode compatibility, animated transitions (`FadeInUp`, `FadeInLeft`), and responsive design for visual polish.
- **Prompt Flexibility**: Allows broad and intuitive user inputs, abstracting technical jargon and giving users freedom to express fitness needs in their own words.
- **Decoupled Data Handling**: The generation logic, view logic, and storage mechanism are modular and isolated, making future improvements and debugging more straightforward.

Implementation Challenges

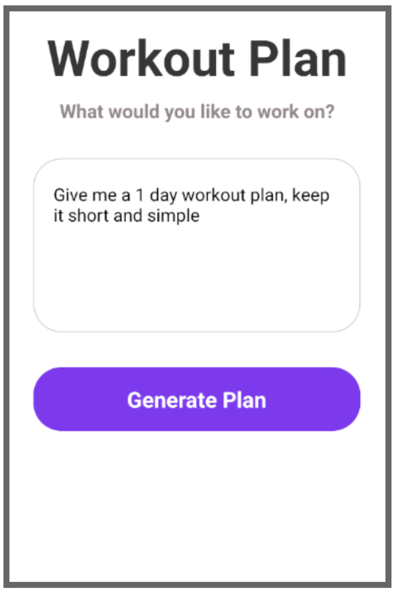
- **Prompt-to-Plan Reliability**: Ensuring the backend consistently interprets natural language inputs into actionable workout plans required robust prompt design and testing.
- **State Management**: Toggling between plan generation and result display states (`showResult`, `generatedPlan`) within a single screen introduced conditional complexity.
- **Save vs. Regenerate Flow**: Balancing between quick re-generations and long-term saving needed intuitive UI/UX design and visual cues.
- **Firestore Sync**: Plans are user-specific and needed correct Firestore structuring (`Users/{uid}/plans/workoutPlan`) along with error handling for write failures or missing user sessions.
- **Mobile Performance**: Network calls to the AI backend needed loading state management (`isLoading`) and smooth transitions to maintain responsiveness, especially on slower devices.

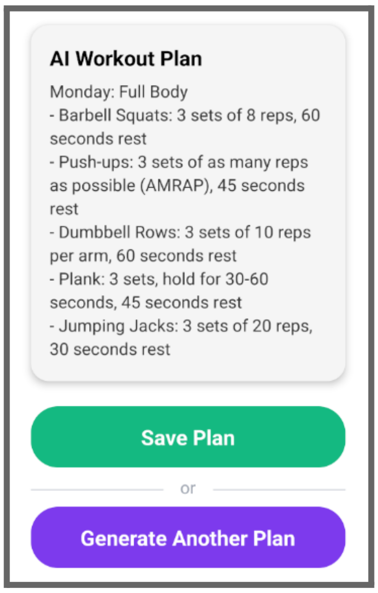
Diagrams

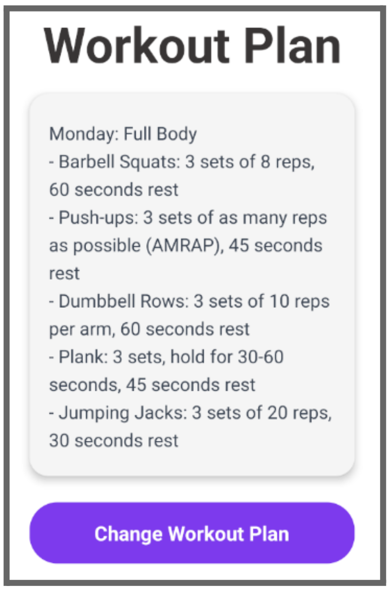




Workout Tracking Home Page and Workout Plan Page (Before Saving Plan)


Workout Plan
What would you like to work on?
Give me a 1 day workout plan, keep it short and simple
Generate Plan


AI Workout Plan
Monday: Full Body
- Barbell Squats: 3 sets of 8 reps, 60 seconds rest
- Push-ups: 3 sets of as many reps as possible (AMRAP), 45 seconds rest
- Dumbbell Rows: 3 sets of 10 reps per arm, 60 seconds rest
- Plank: 3 sets, hold for 30-60 seconds, 45 seconds rest
- Jumping Jacks: 3 sets of 20 reps, 30 seconds rest


Workout Plan

Monday: Full Body
- Barbell Squats: 3 sets of 8 reps, 60 seconds rest
- Push-ups: 3 sets of as many reps as possible (AMRAP), 45 seconds rest
- Dumbbell Rows: 3 sets of 10 reps per arm, 60 seconds rest
- Plank: 3 sets, hold for 30-60 seconds, 45 seconds rest
- Jumping Jacks: 3 sets of 20 reps, 30 seconds rest

Generate Workout Plan Page (Before/After Generation) and Workout Plan Page (After Saving Plan)

Profile & Settings [Extensions Completed]

Description

The Profile & Settings feature acts as the user's personal hub within the Momentum app. It combines two main components—**Profile** and **Settings**—to deliver both information display and customization capabilities in one seamless interface.

On the **Profile** page, users are greeted with an animated, visually engaging overview of their personal fitness statistics. This includes their current **workout streak**, **number of workouts this week**, and **total workout time**, all fetched in real-time from Firebase. These statistics give users a motivating snapshot of their progress and activity level.

In addition to stats, the profile screen provides **quick-access buttons** to navigate to the **Friends page**, where users can manage friendships, and to the **Settings page** for further app customization. A **logout button** is also available, allowing users to securely end their session and return to the login screen.

The **Settings** page allows users to toggle between **light and dark modes**, manage their **privacy visibility** (show/hide workout and diet history), **change their username**, **change password**, or **delete their account** entirely. All changes are synced live with Firebase services and reflected immediately in the app UI.

Together, the Profile & Settings feature empowers users to take control of their Momentum experience, offering both personalized fitness insights and robust account management tools in a cohesive, well-designed layout.

Users can:

- View fitness stats including:
    - **Workout streak**
    - **Workouts this week**
    - **Total workout time**
- Access the **Friends page** to manage friend requests and connections.
- Access the **Settings page** to customize appearance and privacy.
- **Log out** of their account securely.
- Toggle **Dark Mode** for preferred visual comfort.
- Choose whether to **show or hide workout/diet history** from others.
- **Change their username**, with validation to avoid duplicates.
- **Update their password**, with confirmation and strength validation.
- **Delete their account**, fully removing Firebase Auth and Firestore data.
- Check the **current app version** at the bottom of the Settings page.

Implementation Philosophy

This feature was designed with a clear focus on **user empowerment, clarity, and modular customization**. In Momentum, the user profile serves as a central node—not just for personal reflection via fitness statistics, but also as a gateway to key social and account-related functionalities.

The **Profile screen** provides an animated and responsive interface that showcases the user's **fitness milestones**, such as workout streaks, total workout time, and sessions completed this week. These statistics are pulled directly from Firestore, reinforcing a sense of achievement and encouraging users to stay consistent. The **dashboard layout** promotes positive reinforcement through visual feedback while maintaining minimalism and readability.

Moreover, the profile includes **quick navigation buttons** to the Friends and Settings pages, ensuring that users can seamlessly access their social network and configuration tools. A prominently placed **logout button** also ensures secure session termination, adhering to user expectations for privacy and control.

UI animations (`react-native-reanimated`) make transitions smooth and engaging, and `KeyboardAwareScrollView` enhances the input experience for mobile users. Firebase Auth and Firestore are tightly integrated for backend consistency, with feedback mechanisms like alerts and activity indicators improving user trust and clarity.

Implementation Challenges

- **Ensuring accurate and real-time stat updates:**
  User metrics such as workout streaks and total duration are dynamically pulled from Firestore. This required setting up reliable fetching mechanisms with `useFocusEffect`, while ensuring animations and layout transitions did not cause data flickers or lags.

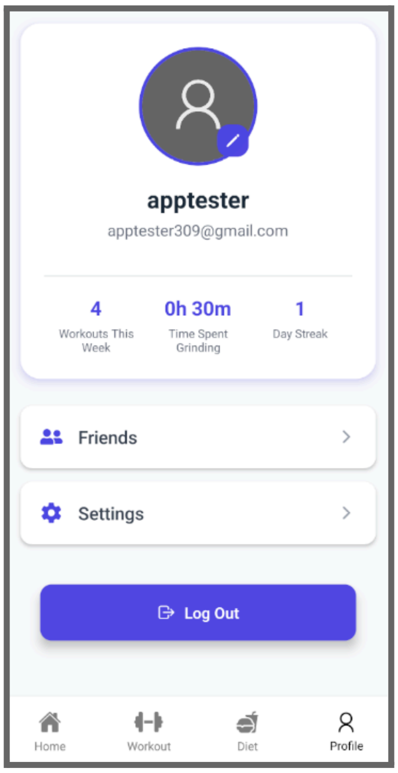
- **Managing smooth navigation flow between interconnected screens:**
  Since the Profile page acts as a hub for both the Friends and Settings screens, transitions needed to be fast, reliable, and memory-efficient. Special care was taken to avoid unnecessary re-renders and maintain navigation state using `expo-router`.

- **Building secure workflows for sensitive actions:**
  Features like **password changes** and **account deletion** required strict validation, feedback alerts, and interaction with Firebase Authentication. This included handling re-authentication requirements, edge-case errors, and guaranteeing user clarity through alerts and loaders.

- **Synchronizing Firebase Auth and Firestore updates:**
  When users update credentials (e.g., changing usernames), both Auth and Firestore records must reflect the change. This coordination was especially important to ensure the app remains consistent across user contexts and prevents sync issues.

- **Maintaining theme consistency across all components:**
  With support for Dark Mode and Light Mode, every UI component, input field, button, and animation had to conditionally adapt its style. This required a centralized theme context and rigorous testing across screens to ensure visual and functional harmony.

Diagrams


apptester

apptester309@gmail.com

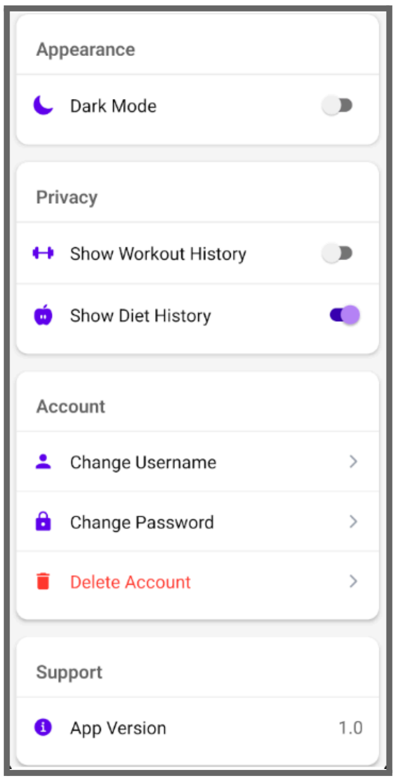
4 Workouts This Week

0h 30m Time Spent Grinding

1 Day Streak

Friends

Settings


Appearance

Dark Mode: off

Privacy

Show Workout History: off

Show Diet History: on

Account

Change Username

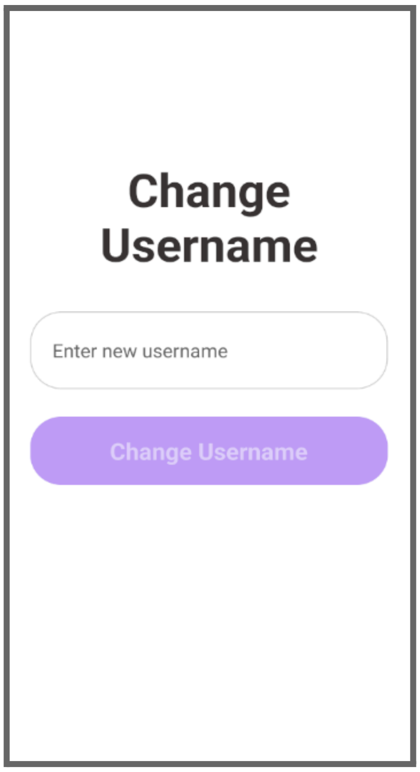
Change Password

Delete Account

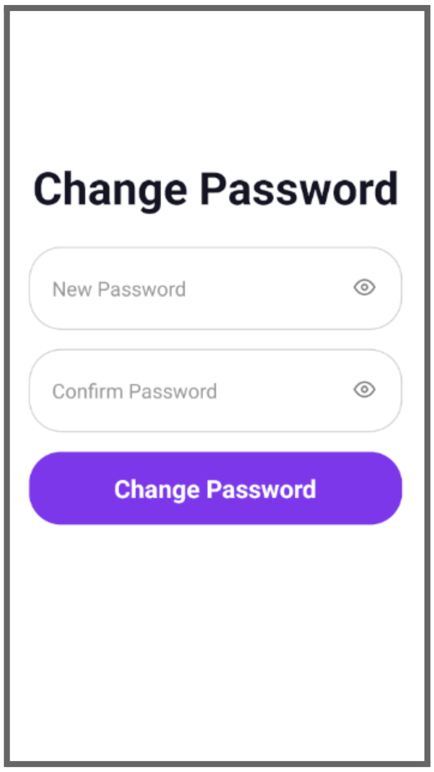
Support

App Version 1.0

Profile Page and Settings Page


Change Username

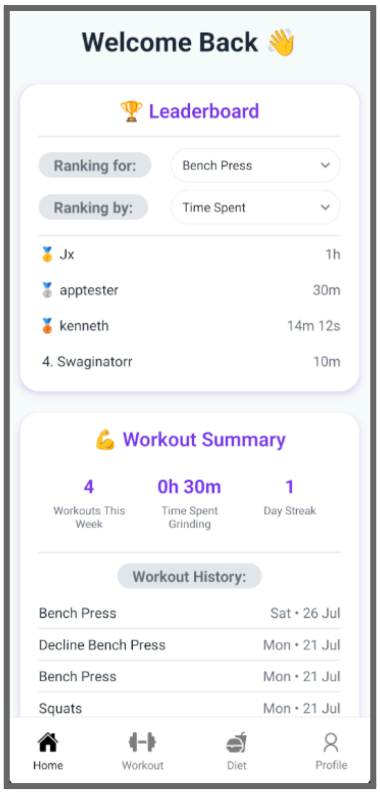
Enter new username


Change Password

New Password

Confirm Password

Change Username Page and Change Password Page



Welcome Back 👋

🏆 Leaderboard

Ranking for:	Bench Press
Ranking by:	Time Spent

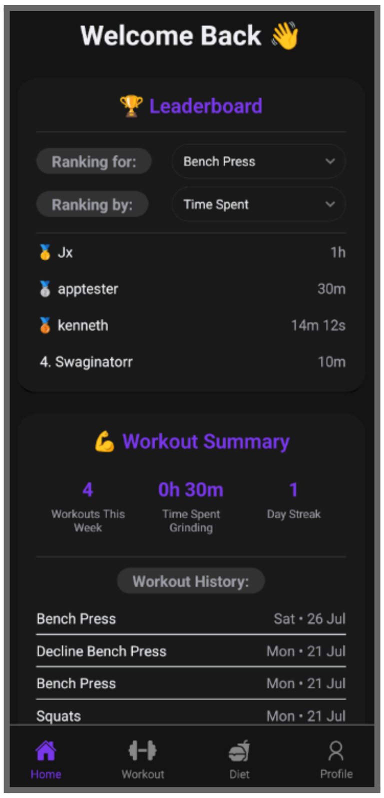
🥇 Jx	1h
🥈 apptester	30m
🥉 kenneth	14m 12s
4. Swaginatorr	10m

💪 Workout Summary

4	0h 30m	1
Workouts This Week	Time Spent Grinding	Day Streak

Workout History:

Bench Press	Sat • 26 Jul
Decline Bench Press	Mon • 21 Jul
Bench Press	Mon • 21 Jul
Squats	Mon • 21 Jul



Welcome Back 👋

🏆 Leaderboard

Ranking for:	Bench Press
Ranking by:	Time Spent

🥇 Jx	1h
🥈 apptester	30m
🥉 kenneth	14m 12s
4. Swaginatorr	10m

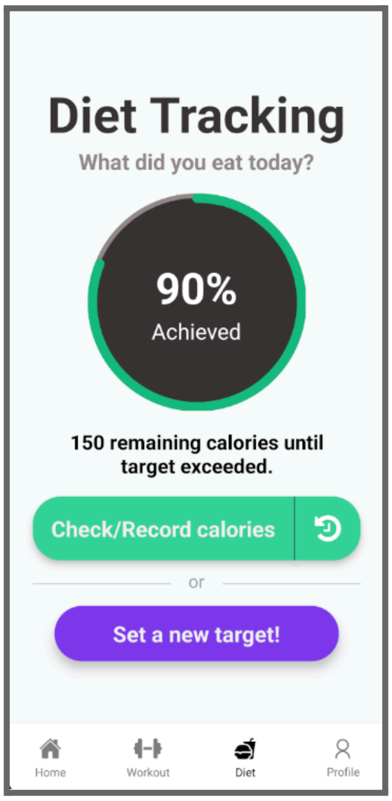
💪 Workout Summary

4	0h 30m	1
Workouts This Week	Time Spent Grinding	Day Streak

Workout History:

Bench Press	Sat • 26 Jul
Decline Bench Press	Mon • 21 Jul
Bench Press	Mon • 21 Jul
Squats	Mon • 21 Jul

Home Page (Light vs Dark)

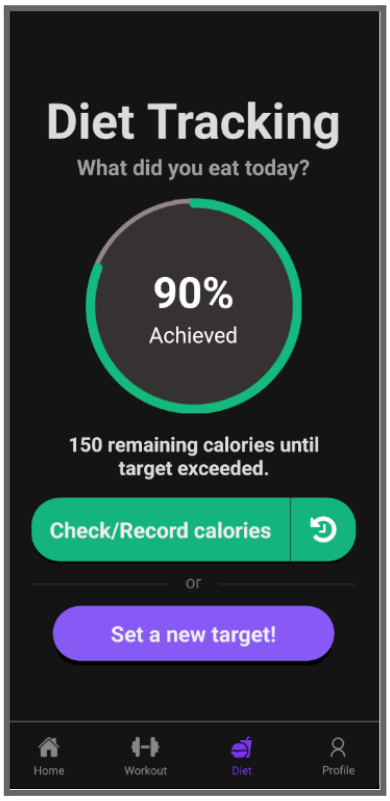


Diet Tracking

What did you eat today?

90%
Achieved

150 remaining calories until target exceeded.



Diet Tracking

What did you eat today?

90%
Achieved

150 remaining calories until target exceeded.

Diet Tracking Page (Light vs Dark)

Timeline and Development Plan

S/N	Tasks	Description	In Charge	Date
1	Finalise ideas *Completed*	Sketched a wireframe and designed a user interface using Figma. Familiarise with the tech stack.	Jing Xi, Kenneth	12 May - 15 May
2	User Authentication & Base Screens *Completed*	Implement user account authentication (Firebase Auth + Google Sign-In) and build screens for login, register, and onboarding. Set up navigation and base tab bar.	Jing Xi, Kenneth	16 May - 26 May
3	Polish UI + Testing and Debugging *Completed*	Verify authentication flow, navigation stability, and resolve UI inconsistencies.	Jing Xi, Kenneth	27 May – 1 June
Milestone 1: Ideation				**2 June**
4	Workout Tracking *Completed*	Build screens for recording workouts (sets/reps/weight) and viewing workout history. Store data in Firestore.	Kenneth	3 June – 21 June
5	Diet Tracking + AI Integration *Completed*	Set up calorie target system, build screens for recording food, viewing history, and viewing progress. Integrate AI-powered calorie estimation API.	Jing Xi	
6	Profile & Settings Base + UI Polish *Completed*	Implement initial profile/settings screens with daily streak display, dark mode toggle, notification toggle, and kg/lbs units. Polish UI across app.	Jing Xi, Kenneth	22 June – 25 June

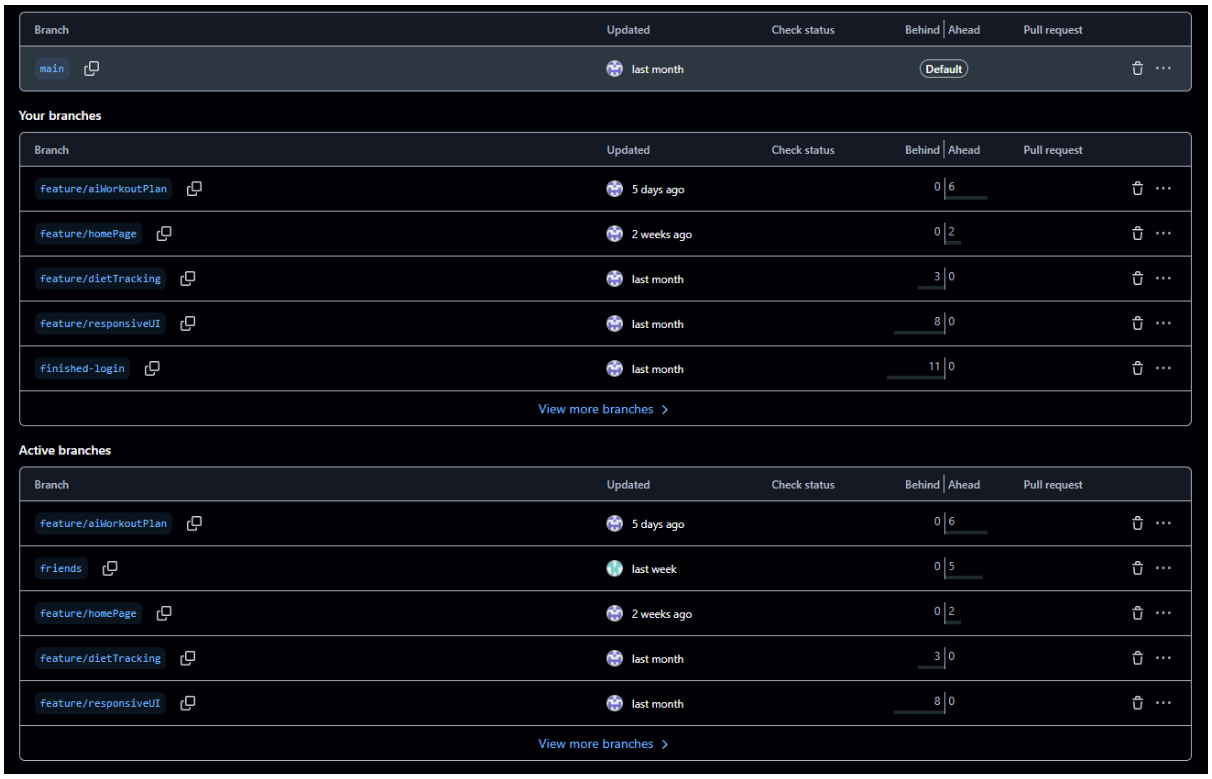
7	UI Improvements, Testing and Debugging *Completed*	Check data recording flows, AI API outputs, and UI responsiveness across screens.	Jing Xi, Kenneth	26 June – 30 June
Milestone 2: Prototype				**1 July**
8	Friend System *Completed*	Implement functionality to add/search friends and build friends list UI.	Kenneth	2 July – 12 July
9	Leaderboard System *Completed*	Build leaderboard based on workout/diet activity among friends and integrate with database.		
10	Home Page *Completed*	Develop home page to show leaderboard and workout/diet summaries in a dashboard UI.	Jing Xi	
11	Notifications *Did not proceed due to lack of time.*	Implement push/local notifications for reminders and milestones.		
12	Complete Profile & Settings *Completed*	Finalise: daily streak tracking, toggles for dark mode & notifications, unit switching, and preference persistence.	Jing Xi, Kenneth	13 July – 15 July
13	Workout Timeline *Did not proceed due to lack of time.* AI Workout Plan *Completed*	Build timeline view of workout history and implement AI-generated workout plan with user customisation.	Jing Xi, Kenneth	16 July – 21 July
14	Deploy App *Completed*	Prepare and configure the production environment to deploy the app.	Jing Xi, Kenneth	22 July – 24 July

14	Testing and Debugging *Completed*	End-to-end testing: ensure all features integrate well, fix bugs, refine UX.	Jing Xi, Kenneth	25 July – 27 July
Milestone 3: Extension				**28 July**

Software Engineering Practices

Version Control

Branching



Branch	Updated	Check status	Behind	Ahead	Pull request
main	last month		Default		

Your branches

Branch	Updated	Check status	Behind	Ahead	Pull request
feature/aiWorkoutPlan	5 days ago		0	6	
feature/homePage	2 weeks ago		0	2	
feature/dietTracking	last month		3	0	
feature/responsiveUI	last month		8	0	
finished-login	last month		11	0	

View more branches

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
feature/aiWorkoutPlan	5 days ago		0	6	
friends	last week		0	5	
feature/homePage	2 weeks ago		0	2	
feature/dietTracking	last month		3	0	
feature/responsiveUI	last month		8	0	

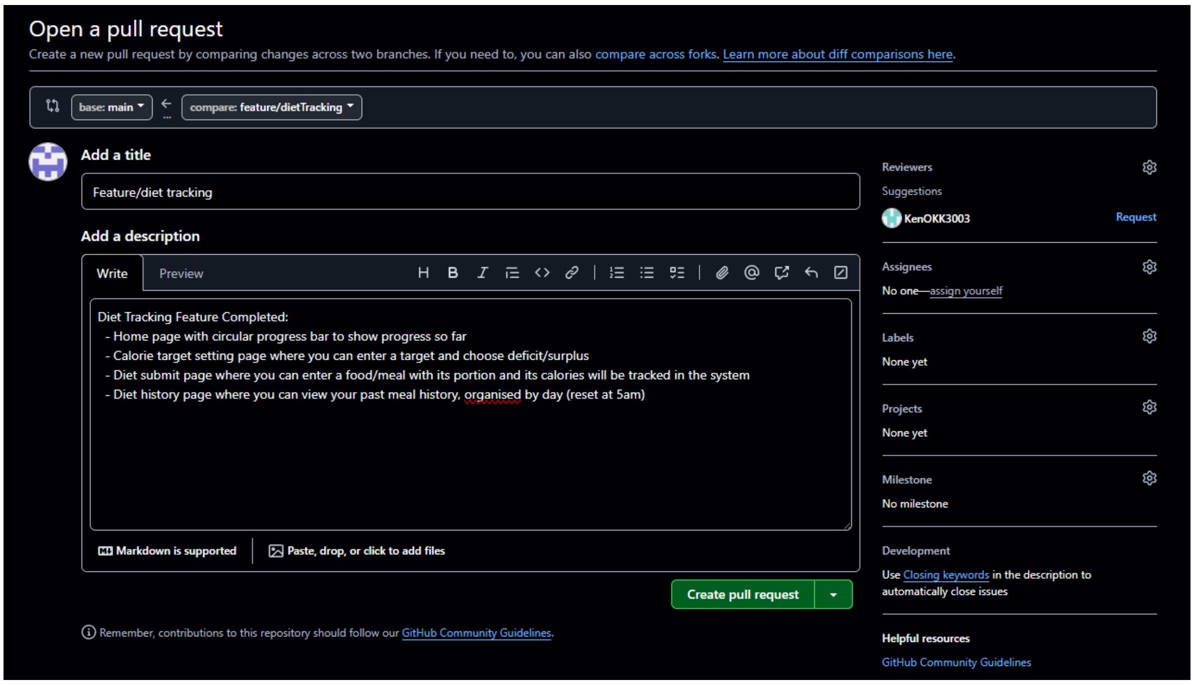
View more branches

We utilize Git as our version control system to ensure a structured and maintainable development process. Central to this is our use of **branching**, which allows multiple developers to work in parallel without interfering with one another's code. The `main` branch serves as our production-ready, deployable base, and is kept stable at all times.

For any new features, enhancements, or bug fixes, we create separate branches off the main branch. These branches are named descriptively based on the task or issue they address (e.g., `feature/dietTracking`, `feature/aiWorkoutPlan`), allowing for easy identification and tracking. This method minimizes code conflicts by isolating development work, which makes troubleshooting and debugging significantly easier. Developers are encouraged to keep their branches up to date by periodically syncing with the latest `main` changes, ensuring compatibility and reducing potential merge issues later.

Once the development work in a branch is completed and thoroughly tested locally, the developer proceeds to the next stage of integration by creating a pull request. This structured approach to branching allows us to enforce code quality, maintain clarity in our version history, and enable smooth scaling as the team grows.

Pull Requests


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: main ← compare: feature/dietTracking

Add a title

Feature/diet tracking

Add a description

Diet Tracking Feature Completed:
- Home page with circular progress bar to show progress so far
- Calorie target setting page where you can enter a target and choose deficit/surplus
- Diet submit page where you can enter a food/meal with its portion and its calories will be tracked in the system
- Diet history page where you can view your past meal history, organised by day (reset at 5am)

Reviewers: Suggestions: KenOKK3003 Request

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Use Closing keywords in the description to automatically close issues

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Helpful resources: GitHub Community Guidelines

Pull Requests (PRs) are a crucial part of our Git workflow, acting as formal checkpoints for integrating code into the main branch. They serve as a gateway for code review, collaboration, and quality assurance. Once a feature or bug fix is ready, the developer opens a PR targeting the `main` branch or another integration branch, depending on the workflow.

The pull request must contain a clear title and a comprehensive description of the work completed, including what was added, changed, or fixed, along with any relevant screenshots or references to issue trackers. This helps reviewers quickly understand the purpose and impact of the proposed changes. Reviewers, typically peers or leads, then examine the code for correctness, readability, maintainability, and consistency with project standards. They may suggest improvements or request changes before approving the PR.

This process fosters a culture of shared ownership and accountability while also serving as a mechanism for knowledge sharing. Discussions within PRs often lead to better solutions and prevent knowledge silos. Additionally, PRs are configured to run automated tests, ensuring that new code doesn't introduce regressions or break existing functionality.

Once all checks pass and the code is approved, the PR is merged into the main branch. This approach ensures that the codebase remains stable, secure, and maintainable. Through regular code reviews and structured PRs, we uphold high development standards and encourage continuous improvement across the team.Two-week Sprints

Our team works on a bi-weekly sprint cycle, which allows us to stay agile and goal-oriented. Every two weeks, we hold a sprint review to reflect on what has been accomplished, evaluate progress against deadlines, and plan for the upcoming sprint. This structured yet flexible schedule helps us stay aligned as a team, adapt quickly to unexpected blockers, and deliver steady improvements. It also creates a rhythm of accountability that keeps development momentum strong while leaving room to pivot when necessary.

Security Measures

Hiding API Keys

To protect sensitive credentials such as API keys, we store them in environment (`.env`) files that are excluded from version control using `.gitignore`. This ensures that secrets are never accidentally committed to the remote repository. By isolating these keys locally, we reduce the risk of unauthorized access to our Firestore database and Firebase Authentication services, safeguarding both our application and users from potential security threats.

Firestore Rules Enforcement

We implemented strict **Firebase Firestore Security Rules** to control access to user data. Only authenticated users can read or write their own documents, and access to sensitive collections is role-based or UID-specific. This helps prevent unauthorized data access and enforces privacy-by-design at the database level.

Account Recovery & Session Management

We ensure secure **account recovery** through Firebase's password reset flow, which sends secure, expiring links via email. Additionally, users are automatically signed out after account deletion, and **Firebase Auth token refresh** logic ensures sessions remain valid and secure.

Secure Hosting & HTTPS Enforcement

The app backend (if deployed) is hosted via secure platforms that enforce **HTTPS** and provide built-in **SSL certificates**. All API traffic is encrypted in transit to prevent man-in-the-middle attacks.

Quality Control

Automated Testing

To ensure that the application functions correctly and is free of critical bugs, automated testing plays a key role in the quality control process. For Momentum, we applied three key types of automated testing:

1. **Unit testing** – Focuses on verifying individual functions, methods, or classes in isolation to ensure that each part behaves as intended.

2. **Integration testing** – Examines how different components or modules of the app interact with each other, ensuring that combined functionality works smoothly.

3. **System testing** – Validates the entire application as a whole, assessing end-to-end scenarios to confirm that the app meets its intended requirements and performs reliably in real-world conditions.

Unit Testing

Unit testing forms the foundation of our automated testing strategy. It involves writing small, focused test cases to check the logic of individual functions or components. In our case, we concentrated on validating the core logic embedded within specific modules to catch issues early and maintain a high level of code reliability.

Test ID	User Story	Testing Objective	Steps Taken	Expected Results	Pass/ Fail	Date Tested
1	As a new user, I want to create an account.	Validate that signing up creates the correct user object.	1. Call `createUserWithEmailAndPassword("new@test.com", "password123")` 2. Capture returned `user.uid`.	The created user should have a unique ID defined, and the email should match "new@test.com".	Pass	29/05/2025
		Prevent signup with bad email.	Call `createUserWithEmailAndPassword("bademail", "password123")` in try-catch.	Should throw an error indicating an invalid email format.	Pass	29/05/2025

		Prevent signup with a weak password.	Call with `"test@x.com", "123"`.	Should throw an error indicating the password is too weak.	Pass	29/05/2025
2	As a returning user, I want to log in.	Ensure valid login returns the same user.	Call `signInWithEmailAndPassword("new@test.com", "password123")`.	Should return a user object with the email matching "new@test.com".	Pass	29/05/2025
		Handle wrong password.	Try `"new@test.com", "wrongpass"`.	Should throw an error indicating incorrect password.	Pass	29/05/2025
		Validate that Google Sign-In returns a user object.	Call `GoogleSignin.signIn()` and wait for the returned user object.	The returned user object should have a valid `idToken` and `email` matching the Google account used to sign in.	Pass	29/05/2025
3	As a user, I want to log out.	Ensure logout clears user.	Call `signOut()` then `onAuthStateChanged`.	The current user should be null after logout.	Pass	29/05/2025
4	As a user, I want to record my workouts.	Ensure workout written to Firestore.	1. Call `handleSubmit("Pushups", {...})`. 2. Read document from `Users/{uid}/workouts`.	Retrieved workout document should have the name "Pushups".	Pass	28/06/2025
		Store sets/reps/weight.	Retrieve the same workout document fields.	The sets field should equal "3", reps and weight should also match what was written.	Pass	28/06/2025
5	As a user, I want to view my workout history.	Validate fetch returns same workout.	Call `fetchWorkouts()` right after submit.	The workout list should include an entry with the name "Pushups".	Pass	28/06/2025

6	As a user, I want to set a calorie target.	Save target with correct type.	Call `updateTarget(2000, "deficit")`. Read from `Users/{uid}/targets/dietTarget`.	The stored target should be 2000 calories, with the goal type set to "deficit".	Pass	28/06/2025
7	As a user, I want to record meals.	Ensure food writes to diet collection.	Call `recordCalories("2 eggs")` then read `Users/{uid}/diet`.	The stored food document should have the name "2 eggs" and the calories should be a number greater than 0.	Pass	28/06/2025
8	As a user, I want to view my food history.	Validate read after multiple writes.	Insert 3 foods, then call `fetchFood()`.	The fetched list should have a length of 3, with one of the food names being "2 eggs".	Pass	28/06/2025
9	As a user, I want to see my caloric progress.	Calculate progress vs target.	Set target to 2000, then log meals totaling 1500. Run progress calculation.	Remaining calories should show 500.	Pass	28/06/2025
		Check exceeded state.	Log meals totaling 2500.	Remaining calories should show -500, indicating exceeded target.	Pass	28/06/2025
10	As a user, I want to be able to add my friends who are also using the app.	Validate that friend request is sent and stored correctly	1. Call `sendFriendRequest("friendUser")` 2. Check `/Users/{uid}/friendRequestsSent`.	The friend request should be stored with the correct username and timestamp	Pass	26/07/2025

11	As a user, I want to be able to view my friend’s profile and history.	Ensure friend profile data loads correctly.	1. Navigate to FriendProfilePage 2. Load `Users/{friendUid}` stats and conditionally display workout/diet buttons.	Friend’s profile image, name, and stats should load correctly. Buttons should respect privacy settings.	Pass	26/07/2025
12	As a user, I want to be able to remove people from my friends list.	Validate unfriend action removes correct records.	1. Call `unfriendUser("friendUser")` 2. Verify changes in both users’ `/friends` collection.	Friend entry should be removed from both user records.	Pass	26/07/2025
13	As a user, I want to see the rankings between myself and my friends.	Ensure ranking list displays correct sorted data.	1. Insert mock stats for user and friends 2. Navigate to Rankings tab.	Rankings should show users in correct order based on selected criteria.	Pass	26/07/2025
14	As a user, I want to get a suggested workout plan	Validate AI-generated plan is displayed correctly.	1. Call `generatePlan()` 2. Confirm data is rendered.	A plan card with exercises should appear, along with save and regenerate buttons.	Pass	26/07/2025
15	As a user, I want to know how many days in a row I have maintained my workout/diet streak.	Ensure streak is accurately calculated and displayed.	1. Insert workout/diet logs for consecutive days 2. Load Profile page.	Profile should show correct streak count (e.g., “5-day streak”).	Pass	26/07/2025
16	As a user, I want to be able to change my username and password.	Validate credential updates are applied.	1. Call `updateUsername("newName")` and `updatePassword("newPassword")` 2. Fetch data again.	Username and password should be updated in both Auth and Firestore.	Pass	26/07/2025

Integration Testing

Integration testing is used to simulate and mimic user interactions with the Momentum app such as scrolling, selecting, entering certain texts, clicking on buttons, toggling dark mode, and accessing profile features. The following user flows were tested:

1. Create account
2. Login
3. Log workout
4. Set diet target
5. Log meals
6. Access profile and view stats
7. Add friend and accept request
8. View friend's profile and history
9. Change username and password
10. Delete account
11. Toggle dark mode

Test ID	User Story	Testing Objective	Steps Taken	Expected Results	Pass/Fail	Date Tested
1	As a new user, I want to create a new account.	Test the ability to create a new user account.	1. Launch the app 2. Tap on “Get Started” 3. Enter `apptester309@gmail.com` and `123123` into the email and password fields respectively 4. Tap on “Sign Up”	Users are able to create a new account	Pass	26/07/2025
2	As a user, I want to log in to my account.	Test the login functionality.	1. Launch the app 2. Enter `apptester309@gmail.com` and `123123` into the email and	User is logged into the app and redirected to home screen	Pass	26/07/2025

			password fields respectively 3. Tap on “Log In”			
3	As a user, I want to log my workouts.	Test the ability to submit a workout.	1. Navigate to WorkoutTracking 2. Tap “Record a Workout” 3. Fill workout fields and submit 4. Navigate to WorkoutHistory	Workout is saved and appears in history	Pass	26/07/2025
4	As a user, I want to set my diet target.	Test ability to set calorie goal.	1. Navigate to DietTracking 2. Tap “Set a new target!” 3. Enter 2000 calories and select “deficit” 4. Submit target	Target is saved and reflected in DietTracking	Pass	26/07/2025
5	As a user, I want to log my meals.	Test ability to submit a meal entry.	1. Navigate to DietTracking 2. Tap to log a meal 3. Input “Chicken Rice” and submit 4. View logged meals list	Meal is saved and shown in history list	Pass	26/07/2025
6	As a user, I want to view my profile stats.	Test profile accessibility and stats rendering.	1. Tap on Profile 2. Observe stats like streak,	Profile info is displayed and navigation works	Pass	26/07/2025

			weekly workout, etc. 3. Tap to navigate to Friends or Settings page			
7	As a user, I want to add a friend and accept requests.	Test friend request and acceptance flow.	1. Go to FriendsPage 2. Search for a user and tap “Add” 3. Log into other account and accept request 4. Return to Friends list	Friend appears in list for both users	Pass	26/07/2025
8	As a user, I want to view my friend’s profile and history.	Test friend profile access.	1. Tap on a friend from Friends list 2. View profile stats 3. Tap to view workout/diet history (if permitted)	Friend’s profile and data displayed if settings allow	Pass	26/07/2025
9	As a user, I want to change my username and password.	Test edit credentials feature.	1. Go to Settings 2. Tap Change Username → Enter new name and save 3. Tap Change Password → Enter old and new password and confirm	Updates are saved and reflected on next login	Pass	26/07/2025

10	As a user, I want to delete my account.	Test account deletion flow.	1. Go to Settings 2. Tap Delete Account 3. Confirm deletion with password 4. Try logging in again	Account is deleted and inaccessible	Pass	26/07/2025
11	As a user, I want to use dark mode.	Test theme toggle functionality.	1. Go to Settings 2. Tap Dark Mode toggle 3. Observe UI color change 4. Relaunch app to verify persistence	UI switches to dark mode and persists after restart	Pass	26/07/2025

System Testing

System testing is a high-level end-to-end testing process that validates the complete and integrated application to ensure it meets the specified requirements. It simulates real user scenarios across multiple components—such as authentication, navigation, database interactions, and UI behavior—to verify that all features work together as expected. This testing is conducted in an environment that closely mirrors the production setting, ensuring that the app performs reliably in actual usage conditions.

Test ID	User Story	Testing Objective	Steps Taken	Expected Results	Pass/Fail	Date Tested
1	As a new user, I want to create an account, log in, and set a diet goal so I can begin tracking my nutrition.	Test full user onboarding and diet goal setup.	1. Launch app 2. Tap "Get Started" 3. Enter email and password 4. Tap “Sign Up” 5. Log in using the same credentials 6. Go to Diet Tracking 7. Tap "Set a new target" 8. Enter target calorie and select deficit/surplus 9. Submit	User is able to create account, log in, and see their new diet goal set	Pass	26/07/2025

2	As a returning user, I want to log in, record a workout, and see it reflected in my profile stats.	Test login and workout logging workflow and stat reflection.	1. Launch app 2. Log in with valid credentials 3. Go to Workout Tracking 4. Tap “Record Workout” 5. Input workout name, sets, reps, etc. 6. Submit 7. Navigate to Profile page 8. View updated stats	User can record a workout and see stat update on profile	Pass	26/07/2025
3	As a user, I want to generate an AI workout plan and save it for future use.	Test AI generation, UI flow, and Firestore write.	1. Launch app 2. Log in 3. Go to Workout Tracking 4. Tap “AI Workout Plan” 5. Enter workout goal 6. Generate plan 7. Tap “Save Plan”	Plan is generated and saved to Firestore under user’s plan collection	Pass	26/07/2025
4	As a user, I want to view my saved AI workout plan and update it with a new one.	Test plan retrieval and overwrite logic.	1. Log in 2. Go to Workout Plan 3. Confirm plan is shown 4. Tap “Update Plan” 5. Generate new plan 6. Tap “Save Plan”	New plan replaces old one and displays updated version	Pass	26/07/2025

5	As a user, I want to view a friend's profile and see their public stats.	Test navigation to friend profile and privacy settings.	1. Log in 2. Go to Friends 3. Tap on friend from list 4. View friend profile 5. Confirm stats shown according to their privacy settings	Friend's profile loads with appropriate visible data	Pass	26/07/2025
6	As a user, I want to send and accept a friend request.	Test friend interaction and Firestore update.	1. Log in 2. Search a user 3. Tap "Send Request" 4. Log in as target user 5. Go to Friend Requests 6. Tap "Accept"	Both users see each other in Friends tab	Pass	26/07/2025
7	As a user, I want to toggle dark mode and see the UI respond accordingly.	Test theme toggle and persistent state.	1. Log in 2. Go to Settings 3. Toggle dark mode 4. Navigate around app 5. Relaunch app and confirm setting persists	App UI switches theme and remembers preference	Pass	26/07/2025
8	As a user, I want to delete my account and ensure all data is removed.	Test full account deletion and Firestore cleanup.	1. Log in 2. Go to Settings 3. Tap "Delete Account" 4. Confirm deletion 5. Try logging in again	User cannot log in and data is removed from Firestore	Pass	26/07/2025

User Testing

User testing involves observing real users as they interact with the app to evaluate the usability, intuitiveness, and overall experience. Users are given specific tasks based on realistic scenarios to ensure all core features are tested. Their interactions, challenges, and feedback help us improve app flow, design clarity, and feature accessibility. This process ensures that the final product is not only functional but also intuitive and enjoyable for the target audience.

User Story	Task	Discovery	Conceptual Models, Affordance & Mappings	Signifiers & Constraints	Feedback
As a new user, I want to create an account.	Fill in sign-up form and submit.	User recognized email and password fields.	Expected to input credentials and proceed step-by-step.	Sign-up button clearly marked. Cannot proceed with empty fields.	Received confirmation and was redirected to onboarding.
As a user, I want to log in.	Enter credentials and press login.	User immediately identified login fields.	Recognized flow from sign-up to login.	Login button visually distinct.	Successfully logged in and redirected to home.
As a user, I want to generate a workout plan.	Go to Workout Plan, enter goal, generate.	Found input and button layout intuitive.	Expected a plan to be displayed after input.	Clear CTA button for generating plan.	Plan appeared instantly, easy to read.
As a user, I want to record a workout.	Navigate to workout entry and input data.	User found buttons for adding workout.	Mapped workout type to name, sets, reps.	Field placeholders helped user understand input.	Feedback message confirmed save was successful.
As a user, I want to view workout history.	Go to workout history tab.	Located workout history tab from menu.	Expected a list view of past logs.	Workout entries clearly displayed by day.	Displayed correctly with timestamps.
As a user, I want to track calorie intake.	Record food item and check remaining.	Identified food entry box easily.	Input food → recorded calories → updated daily balance.	Feedback showed surplus/deficit clearly.	Immediate update after entry, intuitive layout.

As a user, I want to see my streak.	Visit profile page and check streak card.	Understood streak card from icon and color.	Expected number + label for current streak.	Clear visibility of streak count.	Streak data accurate and visually engaging.
As a user, I want to add and remove friends.	Search for a friend, add, then unfriend.	Easily found search and add icon.	Mapped friend card to request or unfriend action.	Clear buttons to add friend and unfriend.	Received status updates on friend actions.
As a user, I want to view rankings.	Check Leaderboard tab for stats.	Located tab via menu and icon.	Mapped dropdown to ranking criteria.	Ranked list was well structured.	Able to switch between rank types easily.
As a user, I want to change settings.	Navigate to Settings, edit username, toggle dark mode.	Found buttons and toggles easily.	Expected interaction from label and toggle layout.	Edits auto-saved, dark mode toggled instantly.	Responsive, settings persisted after restart.

Tech Stack

1. **React Native with Expo** (cross-platform mobile app development for iOS and Android)
2. **Firebase** (backend database services using Firestore for real-time data storage and syncing)
3. **Firebase Auth, Google Sign-In** (user authentication and third-party login integration)
4. **Gemini AI** (AI-powered workout plan generation using Google's generative model, and calorie estimating)
5. **Git and Github** (version control and collaborative code management with issue tracking)
6. **Render** (backend deployment and hosting for web components and backend endpoints)

Project Log

Refer to the attached log: Project Log