AHLI: Workout Management Mobile App

BY

ALVIN TAM YUN JIE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) DIGITAL ECONOMY TECHNOLOGY

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: AHLI : WORKOUT MANAGEMENT MOBILE APP

Academic Session: __202406_____

I _____ALVIN TAM YUN JIE_____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____

(Author's signature)

(Supervisor's signature)

Address:

_____

_____

_____

___Ts. Lim Jit Theam____

Supervisor's name

Date: _____8/9/24_____

Date: _____8/9/24_____

FACULTY/INSTITUTE* OF _____ Information and Communication Technology

UNIVERSITI TUNKU ABDUL RAHMAN

Date: _____8/9/24_____

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that _____ALVIN TAM YUN JIE_____ (ID No: __20ACB05025__) has completed this final year project/ dissertation/ thesis* entitled "_____AHLI: Workout Management Mobile App _____" under the supervision of ____ Ts Lim Jit Theam _____ (Supervisor) from the Department of _____, Faculty/Institute* of ___ Information and Communication Technology ____ .

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

_____ALVIN TAM YUN JIE_____
(*Student Name*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**AHLI: Workout Management Mobile App**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____ _____

Name : _____ALVIN TAM YUN JIE_____

Date : _____26/8/24_____

ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Ts Lim Jit Theam and my moderator, Dr Noraini Binti Ibrahim who have given me a golden opportunity to involve in the mobile app development field study. Besides that, they have given me a lot of guidance to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

ABSTRACT

The AHLI: Workout Management Mobile App is intended to give customers with individualized workout regimens, progress monitoring, and gym finding via the use of AI and cloud services. The app uses Google Gemini AI to create personalized workout regimens and the Google Places API to assist users find local gyms, while Firebase is utilized for data management and real-time synchronization. During development, the app encountered problems such as API rate constraints and compatibility concerns with Expo and React Native, which were resolved to provide a seamless user experience. Users expressed high levels of satisfaction with the app's use, design, and essential functions. Despite these obstacles, the software efficiently assists users in managing their fitness goals with easy interfaces and handy monitoring functions. Future enhancements might include adding social interaction features and enhancing API speed to improve the overall user experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
APP	Application Program
GB	Gigabyte
IOT	Internet of Things
IP	Internet Protocol
RAM	Random Access Memory

Chapter 1
Introduction

This chapter provides an overview of the background and reasons for our work, explaining our contributions and the organization of the thesis. Due to the swift progress of technology, mobile applications have become more capable of providing tailored and data-oriented fitness solutions. However, numerous current applications lack a comprehensive integration that caters to diverse user requirements, including personalized exercise routines, social media sharing, and location-based functionalities. Our research aims to close this divide by creating the AHLI: Workout Management Mobile App, which combines AI-driven workout personalization, progress monitoring, social networking, and gym suggestions based on geography. This holistic approach is specifically developed to assist users in effectively attaining their fitness objectives, maintaining motivation via active participation in the community, and making well-informed choices about their exercise surroundings. The thesis examines the technological obstacles and remedies associated with developing the AHLI app, emphasizing its advancements in the realm of fitness technology. The objective of this project is to establish a higher benchmark for fitness management software, showcasing the potential of technology to improve health and well-being in the modern day.

1.1 Problem Statement and Motivation

The significance of human exercise activities in our everyday lives has been well-established. Consistent and regular exercise routines reduce the risk of illnesses such as cognitive decline, aid in maintaining a healthy body weight, and alleviate symptoms associated with despair and anxiety [1]. Keeping up with a regular exercise regimen is a formidable task for many people in today's fast-paced and demanding society. In light of the growing prevalence of inactive lifestyles and increasingly hectic schedules, it might be challenging to allocate time and cultivate the drive for consistent physical activity. Adding to this problem is the intricacy of creating and monitoring customized exercise routines that efficiently address individual objectives, focus on particular body areas, and align with different levels of training. Conventional approaches to exercise planning often entail physical labor, extensive expertise,

and a significant amount of time, which may be overwhelming for those who are already pressed for time. Furthermore, the process of identifying and choosing neighboring fitness centers is often disjointed and ineffective. A significant number of customers have challenges while searching for gyms that are conveniently situated, well equipped, and affordably priced. The absence of a centralized and easily available information source poses a difficulty for people in making well-informed choices about their fitness alternatives. Existing fitness systems often fail to effectively combine tailored advice with real-time progress monitoring. Several existing applications and tools either give generic exercise programs that do not adjust to the changing requirements of the user or do not provide valuable information on the progress of the workout. The lack of standardized monitoring techniques and personalized feedback might result in decreased motivation and engagement, which can hinder users' ability to adhere to their fitness objectives. Furthermore, users often have difficulties in sustaining a feeling of achievement and advancement as a result of insufficient monitoring systems. Conventional monitoring systems or simple app features may not adequately capture the intricacies of personal fitness journeys, resulting in a lack of immediate feedback and motivating assistance. The absence of an efficient method for monitoring progress might lead to reduced user satisfaction and impede the long-term commitment to exercise regimens.

1.2 Objectives

The AHLI: Workout Management Mobile App project uses innovative technologies to provide a full fitness solution. The software provides a fluid and intuitive experience for managing training routines and fitness objectives by combining cloud computing and artificial intelligence technology. The usage of Gemini AI and Firebase services on the Google Cloud platform allows for the creation of a highly configurable and efficient mobile app. This connection allows users to create individualized training regimens, measure progress, and find local gyms without the need for manual data administration.

1. Personalized Workout Generation:

Use Gemini AI to create a highly personalized and dynamic training regimen for each individual. The software guarantees that the exercise plan is appropriately tailored to the user's particular demands by taking into consideration their preferred location, target body part,

specific fitness objectives, and current training level. The sophisticated AI system analyzes this data to create a tailored training plan that maximizes efficiency and efficacy. Users may easily monitor their progress using a straightforward checkbox system that enables them to mark completed activities and see their total completion percentage. This function not only helps customers keep on track with their fitness objectives, but it also gives encouraging feedback by displaying their progress in real time. By combining powerful AI-driven customization with extensive progress monitoring, AHLI provides a holistic solution that adapts to and supports each user's individual fitness path.

2. Social Feed and Engagement:

Create a colorful and engaging feed feature that allows users to actively share their training regimens, dietary ideas, and fitness achievements. This feed enables users to add photographs, text, and other information about their fitness journey, resulting in a dynamic and interesting community environment. With built-in social media connectivity, users can simply share their content across several platforms, broadening their reach and cultivating relationships outside of the app. The feed allows users to celebrate their accomplishments, seek and offer inspiration, and connect with others who have similar fitness objectives. By allowing for this degree of participation and connection, the app not only promotes individual achievement but also fosters a friendly and motivating community, improving the entire user experience.

3. Workout Tracking and Gym Discovery:

Enhance your training experience with a cutting-edge sit-up tracker powered by Expo's accelerometer. This tool gives users real-time feedback on their sit-up workouts, as well as extensive historical performance statistics, allowing them to monitor and improve their technique over time. The tracker also provides useful workout advice to help users complete each action accurately and effectively. Additionally, the nearby gyms feature, which uses the Google Place API, provides a complete display of gyms within a 5km radius. Users may make educated judgments by browsing an interactive list of local gyms, seeing specific information such as facility features and price, and reading Google reviews. This feature streamlines the

process of identifying and choosing the best gyms based on user interests and location, making it simpler for consumers to stick to their exercise regimens.

1.3 Project Scope and Direction

The AHLI: Workout Management Mobile App project seeks to provide a complete fitness solution that uses contemporary technologies to improve user experience and engagement. The project's scope involves the creation of various important components intended to give a comprehensive approach to fitness management. The app's central feature is its connection with Gemini AI, which generates highly tailored exercise routines depending on the user's location, target body parts, fitness objectives, and training level. This tailored schedule will be supplemented by a progress monitoring function that will enable users to track their workout completion and overall performance using a simple checkbox system. In addition, the app will have a full social feed where users can share their exercise experiences, dietary suggestions, and fitness goals, as well as seamless integration for sharing information across several social media platforms to stimulate community participation and support.

The software will also have a sit-up tracker that uses Expo's accelerometer to provide users real-time feedback and historical performance statistics. This feature will be improved with fitness recommendations and the option for users to share their accomplishments, increasing incentive and accountability. Another important feature is the gym finding tool, which is supported by the Google Place API and displays gyms within a 5 km radius of the user's location. This will contain interactive gym cards with specific information, price, and user reviews, allowing consumers to make more educated selections about where to exercise.

The project's goal is to create a user-centric design with intuitive interfaces that are simple to use, while also ensuring that features are adjustable and adaptable to individual requirements. The app will provide a smooth and dynamic user experience by integrating new technologies such as AI and cloud services, as well as employing APIs for key functionalities. Building a robust community via social interaction and motivating incentives will be a key component of the project, encouraging people to connect, share, and help one another. Continuous improvement will be emphasized, with channels for user input and regular updates to keep the app current and effective. Overall, the AHLI app seeks to deliver a complete and engaging

fitness management solution that helps users achieve their health and wellness objectives while also cultivating a lively, connected community.

1.4 Contributions

The AHLI: Workout Management Mobile App provides a game-changing solution in the world of fitness technology by solving the typical constraints of existing techniques. Conventional techniques either need substantial technical skill to create from scratch or depend on inflexible, pre-packaged systems that are difficult to change. AHLI addresses these difficulties by combining modern technology like Gemini AI with Google Cloud services to build a platform that is extremely flexible and user-friendly. This connection enables the creation of individualized training regimens, real-time progress monitoring, and simple customization to individual demands, simplifying fitness management while decreasing related time and expenses. In addition to technical innovations, AHLI increases user engagement with a sophisticated social feed, sit-up monitoring with real-time feedback, and an interactive gym finding function driven by Google Place API. These components work together to produce a complete fitness solution that not only fulfills current user demands, but also has the ability to adapt to future requirements. By simplifying the user experience and making fitness management more accessible and efficient, AHLI outperforms conventional fitness solutions, allowing users to reach their health and wellness objectives with more ease and efficiency.

1.5 Report Organization

This report is organized into six chapters to provide a comprehensive overview of the AHLI: Workout Management Mobile App project. Chapter 1, Introduction, sets the stage by outlining the problem statement, project background, motivation, scope, objectives, contributions, and achievements, as well as the organization of the report. Chapter 2, Literature Review, examines existing fitness applications to analyze their strengths and limitations, providing insights and context for the development of the AHLI project. Chapter 3, System Design, details the architectural and design considerations for the app, including the integration of technologies and user interface design. Chapter 4, System Implementation and Testing, describes the development process, including implementation strategies, testing methodologies, and results. Chapter 5, System Outcome and Discussion, evaluates the performance and effectiveness of

the app, discussing its impact and comparing it to existing solutions. Finally, Chapter 6, Conclusion, summarizes the findings, reflects on the project's achievements, and suggests directions for future work.

Chapter 2
Literature Review

2.1 Visual Studio Code

Selecting the appropriate Integrated Development Environment (IDE) is critical for efficient and productive development, particularly when working on complicated projects like the AHLI: Workout Management Mobile App. Visual Studio Code (VS Code) is more than simply an advanced notepad with syntax colorization and automatic indentation; it is a robust, code-focused development environment specifically intended to speed up the creation of web, mobile, and cloud applications. VS Code supports a wide range of programming languages and includes crucial tools for the application development lifecycle, such as a built-in debugger and integration with the popular Git version control system. The AHLI app, which uses Firebase and Google Place APIs to combine features such as tailored training regimens, social media sharing, and real-time monitoring, benefits from VS Code's broad functionality and cross-platform compatibility. Its modular and user-friendly interface facilitates in managing the app's many components, resulting in fast development and testing. Furthermore, VS Code's compliance with industry standards helps to bridge the gap between academic learning and practical application, providing developers with essential skills for future projects [2].

2.1.1 Hardware Platform

The hardware platform serves as the basis for executing and testing programs, making it an important factor for developing mobile and online apps. The hardware platform used for the AHLI: Workout Management Mobile App has an influence on both the development and testing stages. Visual Studio Code (VS Code) is used for development on desktop and laptop computers. These computers generally provide the computing power and memory needed to code, debug, and execute emulators or simulators. Modern desktop PCs and laptops are equipped with multi-core CPUs, plenty of RAM, and SSD storage, ensuring that the development environment runs smoothly and that the app's complicated features are handled

efficiently. The AHLI software was tested particularly on an iPhone 11, which is part of Apple's mobile hardware ecosystem. The iPhone 11 offers a reliable and consistent environment for testing an app's performance, UI, and functionality on iOS. Testing on this device guarantees that the software performs as intended in a real-world setting while also being compliant with iOS-specific features and limits.

Table 2.1.1 Specifications of laptop

Description	**Specifications**
Model	ASUS VivoBook 15 A512
Processor	Intel Core i5-1035G1
Operating System	Windows 11
Graphic	Intel UHD Graphics G1
Memory	12GB DDR4 RAM
Storage	512GB SSD

Table 2.1.2 Specifications of desktop computer

Description	**Specifications**
Model	Custom-built Desktop
Processor	AMD Ryzen 5 5500
Operating System	Windows 10
Graphic	AMD RX6600
Memory	16GB DDR4 RAM
Storage	512GB SSD

Table 2.1.3 Specifications of iPhone 11

Description	**Specifications**
Model	iPhone 11
Processor	Apple A13 Bionic
Operating System	iOS 17

Graphic	Integrated
Memory	Not applicable (Integrated with the processor)
Storage	128GB

2.1.2 Firmware / OS

Windows 10 is utilized on development computers, such as the ASUS VivoBook 15 A512 model, which has an Intel Core i5-1035G1 CPU. It offers a reliable environment for coding, debugging, and managing development tools. This operating system provides a variety of programming environments and tools required for building mobile and web apps, ensuring a smooth and efficient development process.

MacOS or iOS, especially iOS 17, is used to test the mobile app for the iPhone 11. This operating system provides a robust and secure foundation for executing and assessing the app, ensuring that it satisfies Apple device-specific performance and usability criteria. Testing on iOS is necessary to ensure the app's compatibility and operation inside the Apple ecosystem.

2.1.3 Database

Firebase has been completely incorporated into the mobile application to handle authorization of users, data storage, and real-time database tasks. By using Firebase Authentication, Firestore, and Cloud Functions, developers guarantee strong functioning in the program. Using Firebase's SDKs and APIs allows for safe user authentication, effective data storage, and the integration of serverless operations to improve backend functionality [3].

2.1.4 Programming Language

React Native is largely built on JavaScript, a flexible language that allows for the creation of cross-platform mobile applications for iOS and Android devices. Within React Native, JavaScript enables the fast development of user interfaces, application logic, and state management. Additionally, TypeScript, a JavaScript extension, is commonly utilized in React Native applications. TypeScript introduces static typing, which helps detect problems early in the development process, leading in more robust, maintainable code and a better overall development experience.

2.1.5 Algorithm

Personalized Workout Generation:

1. **Choose Location**:
    - Users utilize Google Places API to search for a certain location (e.g., gym or outdoor area).
    - The app saves location data such as name, coordinates, and picture reference.

2. **Choose Body Part**:
    - Users pick a body part to exercise (e.g., chest, legs).
    - This selection is saved in the workout data.

3. **Choose Training Level**:
    - Next, the user chooses their training level (e.g., beginner, intermediate, advanced).
    - The selected level is added to the workout data.

4. **Choose Training Goal**:
    - The user specifies their training goal (e.g., weight loss, muscle gain).
    - The chosen goal is stored along with the other workout data.

5. **Generate AI Workout Plan:**
    - Using the collected information (location, body part, level, and goal), the app generates a prompt for the AI.
    - The AI generates a workout plan based on the user's selections.
    - The generated workout plan is saved in Firebase under the user's profile.

Social Feed and Engagement:

1 User Authentication:

- Detect changes in user sign-in status.
- Update currentUser with user information or set to null if not signed in.

2 Fetch Posts:

- Query Firestore for posts in the discover collection, ordered by timestamp.
- Update posts with the fetched posts.

3 Add a New Post:

- Upload selected image to Firebase Storage if available.
- Add new post to Firestore with content, timestamp, user ID, email, and image URL.
- Update posts with the new post and clear input fields.

4 Select Image:

- Use ImagePicker to select an image from the device.
- Save the image URI in the image state.

5 Remove Selected Image:

- Clear the image URI from the image state.

6 Share a Post:

- Use the Share API to share the post content and image URL.
- Handle the share result (shared or dismissed).

7 Render Posts:

- Display posts in a FlatList with content, user email, timestamp, and image.
- Include a share button for each post.

8 Handle Scrolling Effects:

- Track scroll position using Animated.
- Adjust UI elements' opacity based on scroll position.

Workout Tracking and Gym Discovery:

Sit Up Tracker:

1. **Initialize States and Refs:**
    - Set up states for sit-up count, tracking status, timer, settings, exercise history, and modals.
    - Use refs to track last sit-up time and current sit-up state.

2. **Start/Stop Tracking:**
    - Start Tracking:

I. Subscribe to accelerometer data.

II. Set start time and initialize timer.

III. Reset sit-up count and state.

tab

    - Start Tracking:

I. Unsubscribe from accelerometer data.

II. Record exercise history.

III. Alert the user with results.

3. **Detects Sit-ups:**
    - Monitor accelerometer data.
    - Transition through sit-up states (waiting, going down, going up) based on sensor readings.
    - Increment sit-ups count when a complete sit-up is detected.

4. **Handle User Actions:**
    - **Toggle Tracking:** Start or stop tracking based on current status.
    - **Share Results:** Share the sit-up results using the Share API.
    - **Settings and History:** Show or hide settings, exercise tips, and history modals.

5. **UI Updates:**
    - Animate sit-ups count and provide haptic feedback on sit-up detection.
    - Update UI based on tracking status and exercise results.

2.1.6 Summary of the Technologies Review

The AHLI: Workout administration Mobile App is built on a well-rounded collection of technologies that allow for rapid and successful application building and administration. Visual Studio Code (VS Code) is the primary Integrated Development Environment (IDE), including vital features like as built-in debugging, Git integration, and support for different programming languages. This sturdy platform enables rapid and efficient development and testing operations. Development takes place on strong hardware platforms such as an ASUS VivoBook 15 A512 laptop and a custom-built desktop computer, both of which include high-performance CPUs, plenty of memory, and storage. The app is tested on an iPhone 11 to ensure that it operates well inside the Apple ecosystem and meets iOS-specific requirements.

The software environment consists of Windows 10 for development, which provides a strong basis for writing and managing tools, and iOS 17 for testing, which ensures compatibility and performance on Apple devices. Firebase serves as the backend solution, managing user authentication, data storage, and real-time database activities via its extensive suite of capabilities, including Firestore and Cloud Functions. React Native, which is built on JavaScript and TypeScript, is used for cross-platform mobile app development, allowing for the quick building of user interfaces and application logic while improving code resilience thanks to TypeScript's static typing.

The app's algorithm includes a multi-step process for creating tailored exercise routines, which uses AI to modify workouts depending on user inputs such as location, body part, training level, and objective. Furthermore, the social feed and engagement capabilities provide user identification, post management, picture sharing, and UI upgrades. The sit-up tracker component monitors and counts sit-ups using accelerometer data, providing real-time feedback and user interface animations. This broad range of technologies and techniques contributes to the AHLI app's objective of providing a flexible and user-friendly fitness management experience.

2.2 Review of the Existing Systems/Applications

This section examines apps that are linked to the AHLI: Workout Management Mobile App. The research examines the features, strengths, and limitations of these programs in order to uncover best practices, possible areas for development, and unique distinctions that might help AHLI stand out in the competitive environment of fitness and exercise management applications.

2.2.1 Freeletics

Freeletics is a popular fitness app noted for its pioneering use of artificial intelligence (AI) to generate individualized training regimens. This powerful AI integration is critical in increasing user engagement and boosting fitness results. Freeletics uses powerful algorithms to modify training routines to specific objectives and fitness levels, providing a personalized experience that grows as the user progresses. The app's AI-driven approach is notable for its ability to generate highly tailored training routines based on user data, feedback, and performance indicators. This degree of customization not only keeps exercises relevant, but it also dramatically improves user adherence and enjoyment [4]. The AI system learns from user interactions and adjusts its suggestions to meet the user's evolving fitness demands. This dynamic adaption demonstrates how AI may successfully encourage users by offering a personalized experience that keeps them interested and focused on their fitness objectives.

In addition to customization, Freeletics uses social proof to increase user incentive. The app's community features enable users to share their progress, successes, and experiences, instilling a feeling of belonging and support. Seeing real-life success stories and changes inside the app is a tremendous motivator, pushing users to stick with their workout routines and aim for similar results. The software also uses a freemium model, which includes a free trial period for new users. This method employs the idea of reciprocity, instilling a feeling of duty in consumers to reciprocate by subscribing to the full edition of the app or interacting more fully with its features. This early exposure promotes trust and displays the app's worth, encouraging users to stick to their health objectives.

Furthermore, Freeletics promotes dedication and consistency by encouraging users to define and monitor fitness goals. The app's systematic approach to goal setting and progress monitoring helps users remain motivated and consistent with their exercises. Freeletics helps people remain dedicated to their health journey over time by recognizing little milestones and offering frequent updates.

Overall, Freeletics demonstrates the excellent use of AI in fitness technologies. Its emphasis on customization, social proof, reciprocity, and commitment contributes to increased user engagement and motivation, making it a prime example of how technology may be used to support and maintain fitness objectives.

2.2.2 MyFitnessPal

MyFitnessPal is a popular fitness app known for its complete approach to food and activity management using sophisticated monitoring and data analytics. The app's greatest feature is its huge database of over 11 million items, which allows for accurate calorie counting and nutritional monitoring. With MyFitnessPal's powerful database and user-friendly interface, users can easily register their meals, monitor their daily consumption, and track their activity habits. The app's monitoring capabilities are further strengthened by its connectivity with other fitness gadgets and applications, which allows users to combine their health data into a single platform. This seamless connection provides a comprehensive perspective of a person's fitness journey, making it easy to change food and exercise habits as required.

In addition to its monitoring tools, MyFitnessPal uses social proof and community participation to increase user incentive. The app's social features enable users to connect with friends, discuss progress, and engage in community challenges, instilling a feeling of support and responsibility. The exposure of real-life success stories, as well as the chance to communicate with others on similar health journeys, are important motivators for users to remain dedicated to their fitness objectives. MyFitnessPal uses a freemium approach, with a free basic edition and a paid membership for extra features [5]. This model operates on the reciprocity principle, providing users with an initial taste of the app's key functionality while enticing them to pay for additional features and an ad-free experience. The free edition's accessibility promotes user trust and highlights the app's worth, driving many users to upgrade to the premium version for a more personalized experience. Furthermore, MyFitnessPal promotes goal creation and progress monitoring, enabling users to establish specific health goals and measure their success over time. The software gives frequent feedback and updates to keep users motivated and on track with their fitness objectives. MyFitnessPal helps users stay committed to their health journey by acknowledging successes and providing insights into their eating and activity habits.

Overall, MyFitnessPal demonstrates the excellent application of technology in fitness management. Its emphasis on thorough monitoring, community involvement, reciprocity, and goal-oriented features helps to increase user engagement and motivation, illustrating how digital technologies may help people achieve and maintain their health and fitness objectives.

2.2.3 JEFIT

JEFIT is a popular fitness program that helps users plan and monitor their training regimens. Unlike its Pro cousin, JEFIT's regular edition is free and has a variety of features to meet a variety of exercise goals. Key features include the ability to design and personalize workout routines, monitor exercise performance, and access a large exercise library [6]. One of JEFIT's key features is its workout routine planner, which enables customers to create individualized exercise plans. While the regular software lacks the Pro version's extensive logging and syncing capabilities, it still includes useful tools for monitoring workouts, such as manually inputting weights and repetitions. The software contains a range of workouts with instructional information to help users practice them effectively and securely.

JEFIT's regular edition also includes a timer to manage workout intervals and rest times, albeit it may not provide the same amount of customization and automation as the Pro version. Users may track their progress; however, the software does not automatically sync with an online profile, therefore data is saved locally on the device.

While the free version of JEFIT is an excellent choice for consumers looking for basic workout monitoring and planning tools, it lacks some of the more complex features included in the Pro edition, such as huge exercise libraries, automated logging, and cloud syncing. Despite these restrictions, JEFIT is a useful resource for those who want to manage their workout routines and track their progress without making a financial commitment.

2.2.4 Fitbod

Fitbod is a popular exercise software that employs artificial intelligence to create individualized workout routines based on previous training data [7]. Despite its many excellent evaluations, our analysis demonstrates that Fitbod's AI system has severe shortcomings. The app's major selling point is its AI-generated exercise routines, which aim to give tailored advice. Unfortunately, the AI often falls short, resulting in exercises that lack sophisticated progressive overload, appropriate weight recommendations, and variation. This restriction becomes apparent as users progress, with many finding the software less useful beyond the first several weeks. Fitbod excels in user interface design, with a streamlined and straightforward style that improves the entire user experience. With a rating of 4.8/5, the app succeeds in creating a visually beautiful and user-friendly environment. However, this favorable characteristic is offset by the poor results in other areas. The app's effectiveness is rated 3.2/5 owing to its poor AI architecture, which fails to provide really individualized and effective training regimens. The app's usefulness is also doubtful, with a rating of 2/5, since the premium edition does not provide enough rewards for advanced users or those seeking long-term growth.

One of Fitbod's most significant flaws is its ineffective implementation of progressive overload, which may lead to stagnation and impede user development. Furthermore, the app's weight recommendations are often incorrect, possibly leading to inefficient exercises or even damage. The absence of comprehensive support further reduces its usefulness, forcing customers to seek assistance from online forums or other resources. Despite these shortcomings,

Fitbod has gotten a lot of favorable feedback, because to its early appeal to novices and considerable marketing efforts. Many users may like the app's simplicity and the motivating boost it offers in the early stages of their fitness quest. However, for those looking for a more sophisticated, tailored training experience, the app's limits become clear.

To summarize, although Fitbod may be a helpful starting point for beginners, it falls short for individuals who seek a more advanced and specialized approach to training. Dr. Muscle is a better option for a more complete and successful training solution, since it provides individualized routines based on scientific principles and professional coaching.

2.2.5 Summary of the Existing Systems

• **Freeletics** stands out for its powerful AI and robust community features, as well as its high level of customization and excellent goal tracking.

• **MyFitnessPal** is a top choice for complete food and exercise monitoring, thanks to its vast database and active community.

• **JEFIT's** free edition provides basic exercise planning and monitoring tools but misses several sophisticated functionality found in the Pro version.

• **Fitbod** offers AI-generated exercises, but it issues with progressive overload and weight recommendations, reducing long-term efficacy.

• **AHLI** combines various capabilities, such as AI-generated individualized training programs, social media integration, a sit-up tracker, and a Google Places API-based local gym function. It strives to provide a complete and user-friendly fitness management experience.

Table 2.2 Fitness App Feature Comparison

Feature	Freeletics	MyFitnessPal	JEFIT	Fitbod	AHLI
AI-Generated Workouts	Advanced AI for personalized training	No AI-based training	No AI-based training	AI-generated workouts, but with limitations	Gemini AI generates personalized workout schedules
Customization	Highly customizable based on user data	No AI customization	Customizable workout routines	Limited customization, issues with AI	Customizable workout plan based on location, goal, etc.
Social Proof	Community features for motivation	Social features and community challenges	Basic social features	Minimal social integration	Feed page with social media integration and sharing
Goal Setting & Tracking	Goal setting and progress tracking	Goal setting and progress tracking	Basic workout tracking	Basic tracking, issues with accuracy	Checkbox for progress tracking in workout schedule
Free/Premium Model	Freemium model with free trial	Freemium model with free and premium versions	Free with optional Pro upgrade	Freemium model with limited features	Free with comprehensive features
User Interface	Sleek and user-friendly	User-friendly interface	Simple and functional	Streamlined, but less effective AI	Intuitive and visually appealing

Integration with Devices	Limited device integration	Integrates with various devices	Limited device integration	Limited integration with devices	Integrates with Google Places API for nearby gyms
Additional Features	Social proof, freemium model, goal setting	Extensive food database, community features	Exercise library, workout planner	Issues with progressive overload and weight suggestions	Sit-up tracker with exercise tips, history, and sharing

Chapter 3

System Methodology/Approach

The "AHLI: Workout Management Mobile App" was developed using the Agile methodology to provide a flexible and iterative approach. Agile was selected because of its emphasis on iterative development, which allowed the project to be broken into numerous sprints, each with a focus on a single feature or functionality, such as the Home Page, exercise program generator, and feed page [8]. This iterative method allowed for incremental app improvement, with ongoing feedback provided after each sprint. ExpoGo was used for real-time testing and debugging, creating an efficient environment for quick testing of new features and making real-time modifications based on feedback. The combination of Agile approach and ExpoGo's testing capabilities guaranteed that the app grew to successfully satisfy user demands, while also allowing for incremental enhancements and adjustments throughout the development lifecycle.

3.1 System Design Diagram

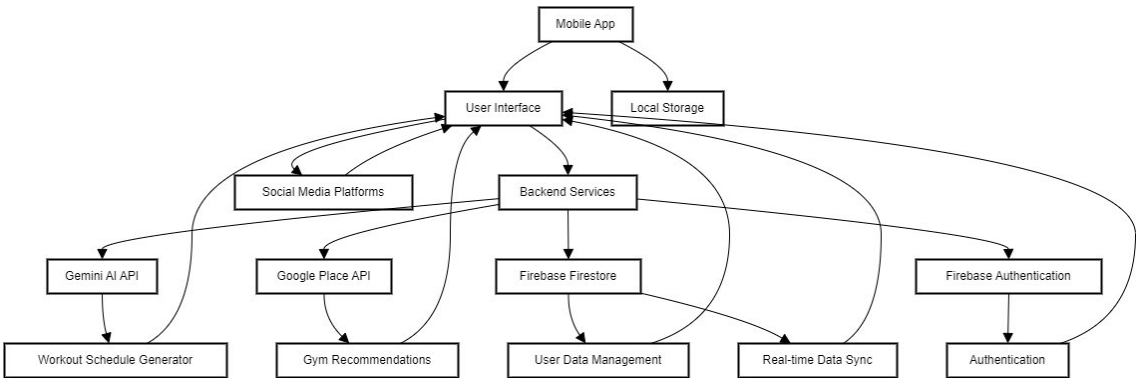


Figure 3.1 System Design Diagram

1 Mobile App (A):

- User Interface (B): The part of the app that interacts with users. Includes various pages and components like the Home Page, Feed Page, Sit-up Tracker, and Nearby Gyms Feature.
- Local Storage (C): Manages temporary storage of data on the user's device.

2 Backend Services (D):

- Gemini AI API (D1): Used for generating personalized workout schedules.
- Google Place API (D2): Provides information on nearby gyms.
- Firebase Firestore (D3): Handles user data management, workout schedules, and real-time data synchronization.
- Firebase Authentication (D4): Manages user authentication and session handling.

3 External Services (E):

- Social Media Platforms: Allows sharing posts from the app to social media.

4 Data Flow:

- Workout Schedule Generator (F1): Receives data from the Gemini AI API and provides personalized workout plans.
- Gym Recommendations (F2): Uses the Google Place API to recommend gyms based on location.
- User Data Management (F3): Manages user data in Firestore.
- Real-time Data Sync (F4): Ensures data is updated in real time across devices using Firestore.
- Authentication (F5): Handles user authentication and login sessions using Firebase Authentication.

3.1.1 System Architecture Diagram

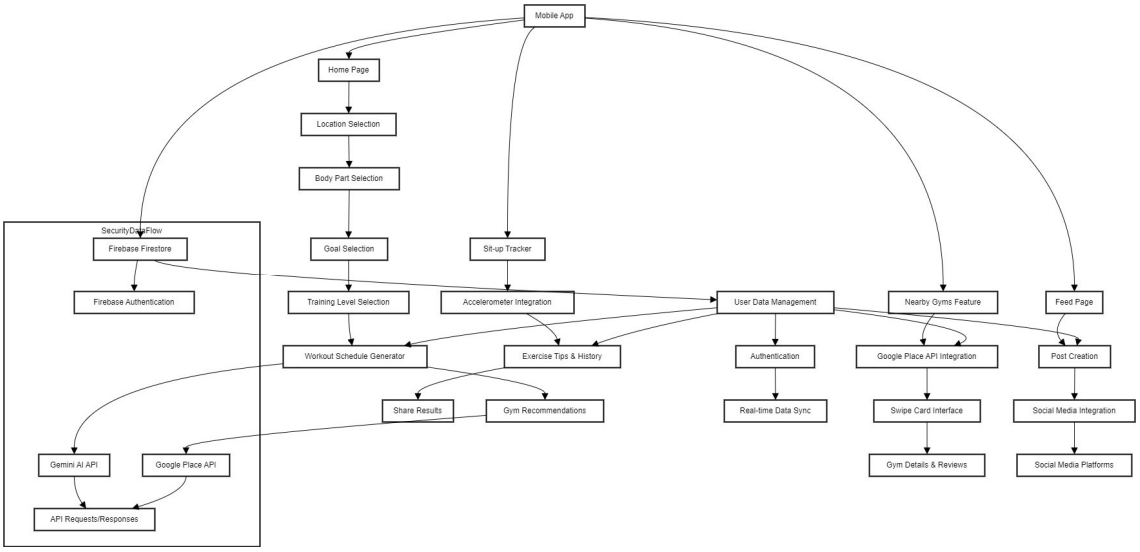


Figure 3.1.1 System Architecture Diagram

Client-Side Components:

1. **Mobile App (A):**
    - The central node representing the entire mobile application.

2. **Home Page (B):**
    - **Location Selection (B1):** Allows users to choose their preferred location for gym recommendations.
    - **Body Part Selection (B2):** Users select the body part they want to train.
    - **Goal Selection (B3):** Users set their fitness goals.
    - **Training Level Selection (B4):** Users specify their training level.
    - **Workout Schedule Generator (B5):** Generates a personalized workout schedule based on the previous selections.
    - **Gym Recommendations (B6):** Provides recommendations for gyms in the selected location.
3. **Feed Page (C):**
    - **Post Creation (C1):** Allows users to create posts about their workouts, nutrition, etc.

- **Social Media Integration (C2):** Enables sharing posts to social media platforms.

4. **Sit-up Tracker (D):**
    - **Accelerometer Integration (D1):** Uses the device's accelerometer to track sit-ups.
    - **Exercise Tips & History (D2):** Provides tips on sit-ups and tracks the user's history.
    - **Share Results (D3):** Allows users to share their sit-up results.

5. **Nearby Gyms Feature (E):**
    - **Google Place API Integration (E1):** Fetches gym data using the Google Place API.
    - **Swipe Card Interface (E2):** Displays nearby gyms in a swipeable card format.
    - **Gym Details & Reviews (E3):** Shows detailed information and reviews for each gym.

Backend Services:

6. **Gemini AI API (F1):**
    - Used by the Workout Schedule Generator (B5) to create personalized workout plans.

7. **Google Place API (F2):**
    - Used by Gym Recommendations (B6) to fetch and display gym information.

8. **Firebase Firestore (F3):**
    - **User Data Management (G1):** Manages user data such as workout schedules and posts.
    - **Authentication (G2):** Handles user authentication.
    - **Real-time Data Sync (G3):** Synchronizes data in real time between the app and the Firestore database.

External Services:

9. **Social Media Platforms (H):**
    - Connected through Social Media Integration (C2) to share user posts.

Data Flow & Security:

10. **Security Dataflow Subgraph:**
    - **Firebase Authentication (I):** Ensures secure user authentication.
    - **API Requests/Responses (J):** Handles requests and responses between the app and external APIs (Gemini AI and Google Place API).

Relationships:

- **B5 (Workout Schedule Generator)** connects to **F1 (Gemini AI API)**, showing the dependency for generating workout schedules.
- **B6 (Gym Recommendations)** connects to **F2 (Google Place API)** and **F3 (Firebase Firestore)** for gym recommendations and data management.
- **F3 (Firebase Firestore)** manages user data, authentication, and real-time sync for various features in the app.

3.1.2 Use Case Diagram and Description

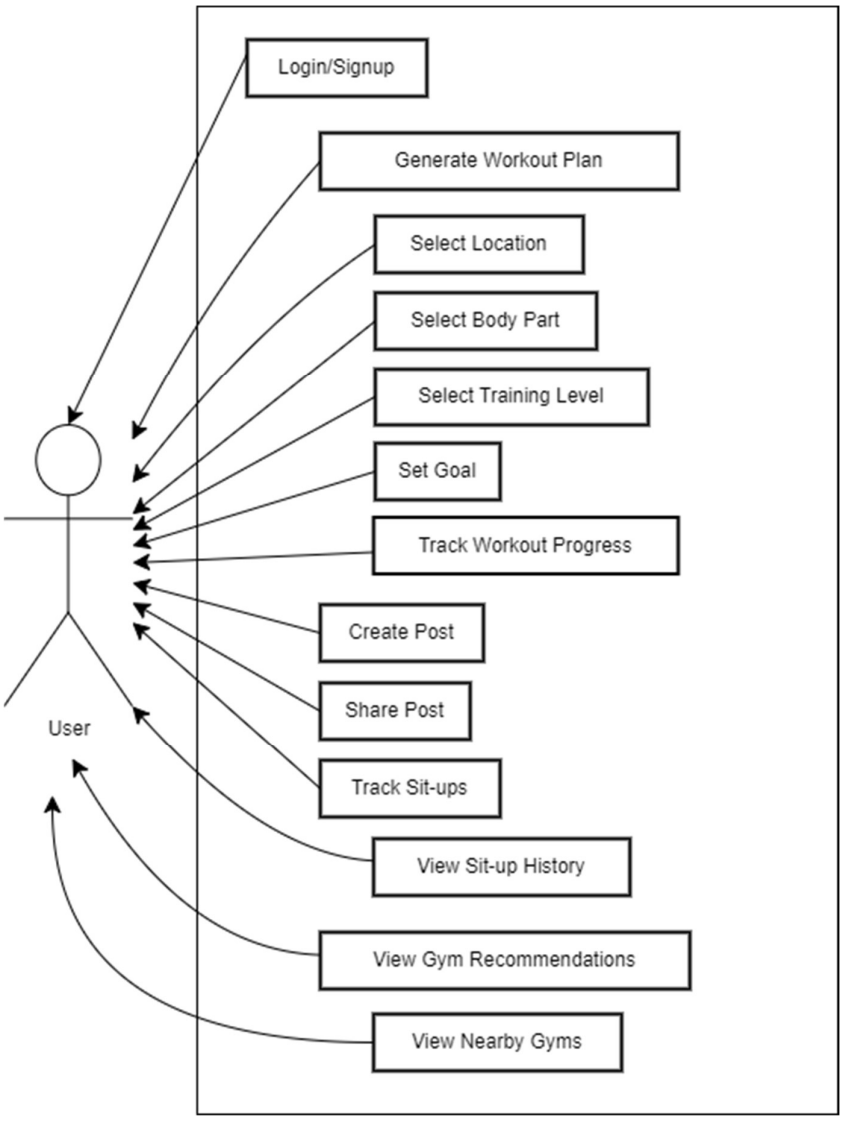


Figure 3.1.2 Use Case Diagram

Use cases:

1. **Login/Signup:** Allows users to log in or create an account on the app.

2. **Generate Training Plan:** Allows the user to design a bespoke training plan based on their preferences and objectives.

3. **Select place:** Allows the user to specify a place, maybe for discovering gyms or scheduling location-specific exercises.

4. **Select Body Part:** Allows the user to tailor their training routine to particular body areas.

5. **Select Training degree:** Allows the user to pick their degree of training experience (beginning, intermediate, or advanced).

6. **Set Goals:** Allows the user to specify fitness objectives such as weight reduction, muscle growth, or endurance development.

7. **Track Workout Progress:** This feature allows the user to track their progress during workouts.

8. **Track Sit-ups:** This feature specifically monitors sit-ups and tracks the user's performance over time.

9. **View Sit-up History:** Displays the user's completed sit-ups, allowing them to track their progress.

10. **Create Post:** Allows the user to create posts, such as discussing their progress, exercises, or encouraging information.

11. **Share Post:** Users may share their posts on social media or with the app community.

12. **View Gym Recommendations:** Offers gym recommendations based on user interests or location.

13. **View Nearby Gyms:** Displays gyms around the user's current location, making it easier to discover locations to work out.

3.1.3 Activity Diagram

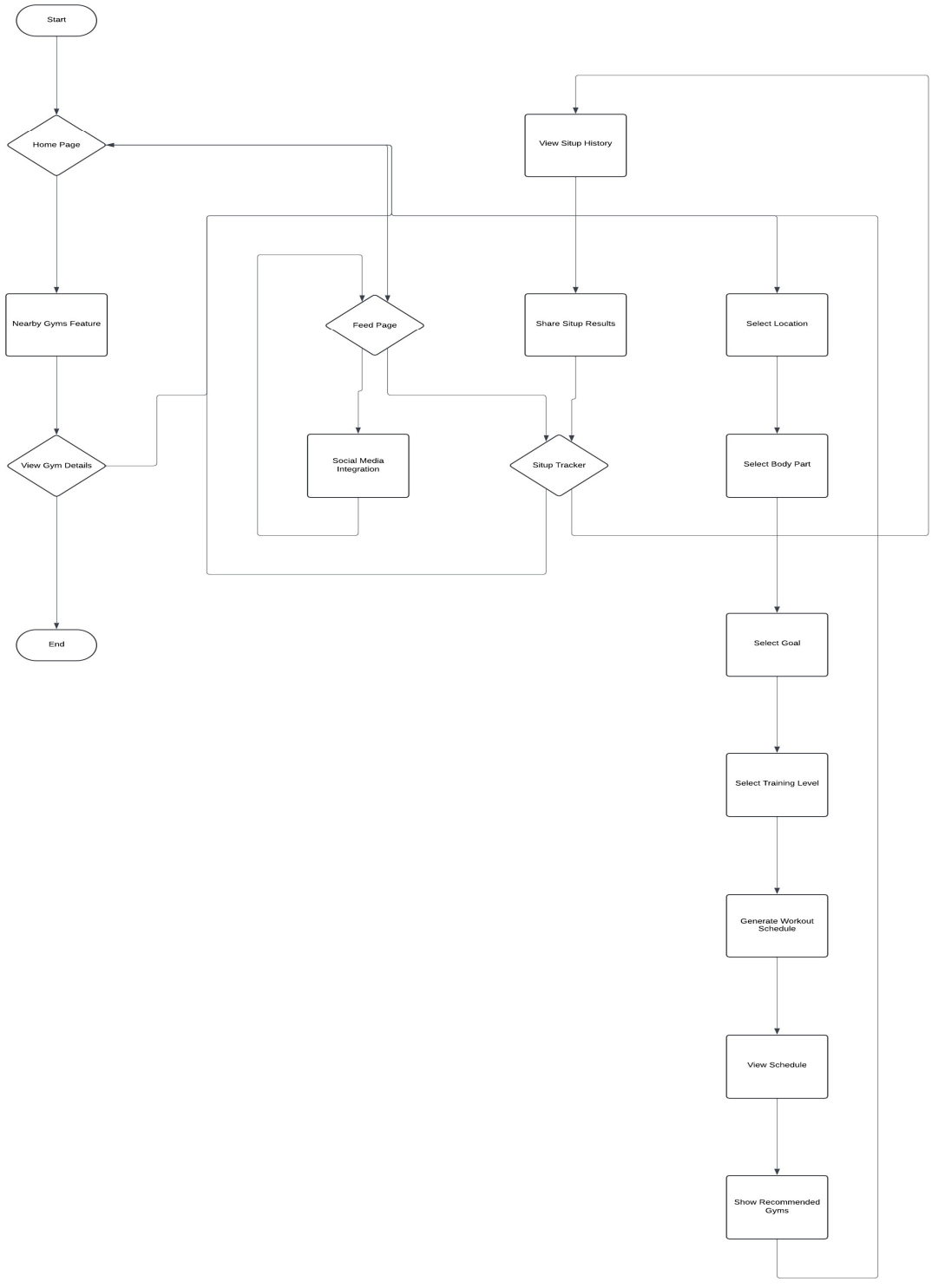


Figure 3.1.3 Activity Diagram

1. Start

- The starting point of the app interaction.

2. Home Page

- The central hub of the app where users begin their journey. From here, users can generate a personalized workout schedule, navigate to other features, or access different functionalities.

3. Select Location

- Users select their preferred location for training, which influences the workout recommendations and nearby gym suggestions.

4. Select Body Part

- Users choose which body part they want to focus on during their workouts (e.g., arms, legs, chest).

5. Select Goal

- Users select their fitness goals, such as weight loss, muscle gain, or endurance improvement.

6. Select Training Level

- Users choose their training level (beginner, intermediate, or advanced), which helps tailor the workout plan to their experience.

7. Generate Workout Schedule

- The AI, powered by Gemini, generates a personalized workout schedule based on the user's selected preferences (location, body part, goal, and training level).

8. View Schedule

- The generated workout schedule is displayed, allowing users to see their planned workouts and track progress with checkboxes.

9. Show Recommended Gyms

- Based on the location selected earlier, the AI suggests the most recommended gyms nearby, including details and pricing.

10. Feed Page

- Users can navigate to the Feed Page, where they can post updates about their workouts, meals, nutrition, or goals.

11. Social Media Integration

- Posts on the feed can be shared on other social media platforms, enhancing social engagement.

12. Sit-up Tracker

- A dedicated tracker that uses the device's accelerometer (via Expo) to track sit-ups, offering real-time tracking and performance analysis.

13. View Sit-up History

- Users can view their sit-up history, receive exercise tips, and keep track of their progress.

14. Share Sit-up Results

- The results of the sit-up tracker can be shared on social media, allowing users to celebrate their progress.

15. Nearby Gyms Feature

- Users can explore gyms within a 5 km radius using Google Places API. This feature provides a list of gyms with swipe cards.

16. View Gym Details

- When a gym card is selected, detailed information about the gym is displayed, including reviews, location, and pricing.

17. End

- The user completes the interaction and can return to the Home Page to access other features or restart a new activity flow.

Chapter 4

System Design

4.1 System Block Diagram

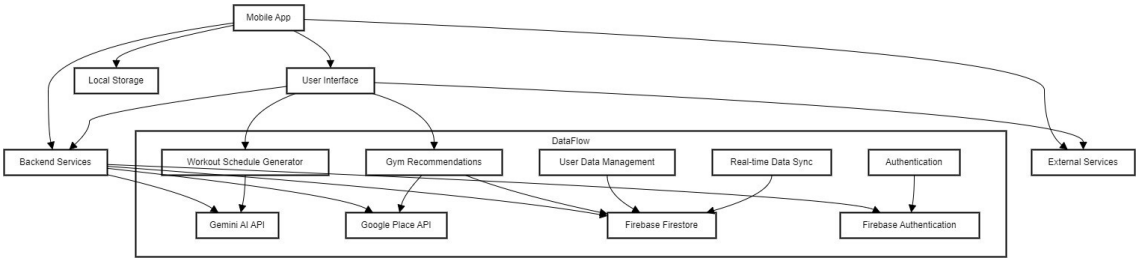


Figure 4.1 System Block Diagram

The "AHLI: Workout Management Mobile App" System Block Diagram depicts the app's architecture and essential components at a high level. The Mobile App, located at the top of the diagram, acts as the system's core center. The software links to numerous essential pieces, which are presented vertically to show how they flow and interact.

The graphic below the Mobile App depicts the User Interface and Local Storage components. The User Interface allows users to engage with the app by browsing through different pages and functionalities. Local Storage handles temporary data on a user's device, allowing for offline capability and speedy data retrieval.

The Backend Services component, located underneath the User Interface and Local Storage, contains various key APIs and services. This includes the Gemini AI API for creating individualized exercise programs, the Google Place API for gathering information about local gyms, Firebase Firestore for managing user data and real-time synchronization, and Firebase Authentication for user authentication and session management.

The External Services component, which sits beside Backend Services, symbolizes the interaction with social networking sites, enabling users to publish their app actions and postings publicly. The DataFlow subgraph shows how the different components interact with the Backend Services. It includes the Workout Schedule Generator, which uses the Gemini AI API to generate personalized workout plans, the Gym Recommendations feature, which uses both

the Google Place API and Firestore to recommend and display nearby gyms, and the Real-time Data Sync and User Data Management functionalities, which ensure data consistency across devices.

Overall, the diagram gives a clear visual depiction of the app's architecture, demonstrating how various components are linked together and how data flows across the system. This vertical orientation highlights the app's hierarchical structure and relationships among its major components.

4.2 System Components Specifications

AI Workout Generation

This component works with an AI service to create personalized training routines depending on user input. It utilizes information regarding the user's location, fitness objectives, body part concentration, and training level.

Technical Specifications:

- **API Integration**: Uses 'chatSession.sendMessage' to interact with the AI service.
- **Data Handling:** Constructs a prompt for the AI using details from the 'CreateWorkoutContext'.
- **Firebase Integration:** Saves the generated workout plan to Firestore.
- **Fallback:** Uses a default image URL if there's an issue fetching the static map image

```
useEffect(() => {
  if (workoutData) {
    generateAiWorkout();
    // Set map image URL if location info is available
    if (workoutData.locationInfo && workoutData.locationInfo.coordinates) {
      const { lat, lng } = workoutData.locationInfo.coordinates;
      setImageUrl(getStaticMapUrl({ lat, lng }));
    }
  }
}, [workoutData]);
```

Figure 4.2.1 UseEffect Hook for generateAiWorkout

The useEffect hook observes modifications to the workoutData. Whenever the workoutData is modified, the hook activates the generateAiWorkout function, which is accountable for generating a training regimen based on the given data. In addition, if the workoutData contains location data along with coordinates, it will update the imageUrl state with a URL that points to a static map image centered on those coordinates. This guarantees that the consumer is presented with a relevant map picture and their training plan is consequently developed.

```
const generateAiWorkout = async () => {
  setLoading(true);

  const FINAL_PROMPT = AI_PROMPT
    .replace('{location}', workoutData?.locationInfo?.name || '')
    .replace('{goal}', workoutData?.title?.desc || '')
    .replace('{bodyPart}', workoutData?.bodyPart?.title || '')
    .replace('{level}', workoutData?.level?.title || '');

  console.log('Generated Prompt:', FINAL_PROMPT);

  try {
    const result = await chatSession.sendMessage(FINAL_PROMPT);
    console.log('AI Response:', result.response.text());

    let workoutData = {};
    try {
      workoutData = JSON.parse(result.response.text());
    } catch (e) {
      console.error('Failed to parse JSON:', e);
      return;
    }

    const docId = (Date.now()).toString();
    await setDoc(doc(db, 'UserWorkout', docId), {
      docId,
      userEmail: user.email,
      workoutData,
    });

    router.push('(tabs)/mytrip');
  } catch (error) {
    console.error('Error generating workout or saving to Firestore:', error);
  } finally {
    setLoading(false);
  }
};
```

Figure 4.2.2 generateAiWorkout Function

1. Search Place

Provides an interface for users to search for a location using Google Places API. The selected location is used to personalize the workout plan.

Technical Specifications:

- **API Integration:** Utilizes Google Places API to fetch location details.
- **Data Handling:** Updates the CreateWorkoutContext with the selected location information.
- **Navigation:** Redirects users to the next step in the workout creation process.

```
<GooglePlacesAutocomplete
placeholder='Search Place'
fetchDetails={true}
onPress={(data, details = null) => {
  // 'details' is provided when fetchDetails = true
  console.log(data, details);
  setWorkoutData({
      locationInfo:{
          name:data.description,
          coordinates:details?.geometry.location,
          photoRef:details?.photos[0]?.photo_reference,
          url:details?.url
      }
  });

  router.push('/create-workout/createworkout')
}}



query={{
  key: process.env.EXPO_PUBLIC_GOOGLE_MAP_KEY,
  language: 'en',
}}
```

Figure 4.2.3 Search Places

This code snippet exemplifies the use of the `GooglePlacesAutocomplete` component to enable users to do location searches. The application has a search box with the placeholder word "Search Place" and gets comprehensive data about the chosen area. Upon the user's selection of a place, the `onPress` function is activated, causing the `workoutData` state to be modified with the name, coordinates, picture reference, and URL of the chosen location. Afterwards, the user is forwarded to the `/create-workout/createworkout` route. The component is set up using a Google Maps API key and a language choice of English.

2. Select Body Part

Allows users to choose which body part they want to focus on in their workout. The selected body part is then saved in the CreateWorkoutContext.

Technical Specifications:

- **State Management:** Uses React's useState to track the selected body part.
- **Navigation:** Moves users to the next step upon selection.
- **UI Elements:** Displays options using FlatList and OptionCard components.

```
<FlatList
  data={SelectBodyList}
  renderItem={({ item }) => (
    <TouchableOpacity
      onPress={() => setSelectedBodyPart(item)}
      style={{
        marginVertical: 10,
      }}
    >
      <OptionCard option={item} selectedOption={selectedBodyPart} />
    </TouchableOpacity>
  )}
  keyExtractor={(item) => item.id.toString()}
/>
```

Figure 4.2.4 Select Body Part

This code sample uses the 'FlatList' component to show a list of body components specified by the 'SelectBodyList' array. Each item in the list is represented by a 'TouchableOpacity' component, which enables for user interaction. When an item is pressed, the'selectedBodyPart' state is updated to reflect the item selected. Each 'TouchableOpacity' has a 'OptionCard' component that shows the specific choice and highlights it if it fits the'selectedBodyPart'. The 'keyExtractor' function guarantees that each item is uniquely recognized by utilizing its 'id' attribute. The'marginVertical' style creates vertical separation between list elements.

3. Select Training Level

Let users select their desired training level (e.g., beginner, intermediate, advanced). This level is used to tailor the workout plan.

Technical Specifications:

- **State Management**: Tracks the selected training level using useState.
- **Navigation**: Redirects users to the next step in the workout creation process.
- **UI Elements**: Lists options using FlatList and OptionCard components.

```
<FlatList
  data={SelectLevelList}
  renderItem={({ item }) => (
    <TouchableOpacity
      onPress={() => setSelectedLevel(item)}
      style={{
        marginVertical: 10,
      }}
    >
      <OptionCard option={item} selectedOption={selectedLevel} />
    </TouchableOpacity>
  )}
  keyExtractor={(item) => item.id.toString()}
/>
```

Figure 4.2.5 Select Training Level

This code sample is identical to the last one, but it displays a list of levels using the 'SelectLevelList' array. The 'FlatList' component displays each item in the list as a 'TouchableOpacity', which when tapped updates the'selectedLevel' state. Each 'TouchableOpacity' has a 'OptionCard' component that displays level information, with visual distinction if the level is picked (matching'selectedLevel'). The 'keyExtractor' function gives each item in the list a unique identifier based on its 'id' attribute, and'marginVertical' provides vertical space between items.

4. Select Goal

Provides a selection of workout goals for users to choose from (e.g., weight loss, muscle gain). This goal informs the AI's workout plan generation.

Technical Specifications:

- **State Management:** Uses useState to keep track of the selected goal.
- **Navigation:** Moves users to the next step after selection.
- **UI Elements:** Uses FlatList and OptionCard for goal selection.

Figure 4.2.6 Select Goal

This code snippet creates a 'FlatList' that displays a list of goals from the 'SelectGoalList' array. Each item in the list is shown as a 'TouchableOpacity' component, enabling users to choose a goal. When an item is pressed, the'setSelectedGoal' function replaces the'selectedGoal' state with the selected item. Each 'TouchableOpacity' has a 'OptionCard' component that shows the goal information and graphically emphasizes the chosen choice if it matches the'selectedGoal'. The 'keyExtractor' function guarantees that each item is uniquely identifiable by its 'id' attribute, while'marginVertical' creates space between list items.

5. Review Selected Workout

Before completing the training plan, a summary of the user's choices is shown. This helps users to reconsider their options.

Technical Specifications:

- **Context Integration**: Displays information from CreateWorkoutContext.
- **Navigation**: Directs users to the workout generation page.
- **UI Elements**: Presents a summary of location, goal, body part, and level.

```
<View>
  <Text>Location: {workoutData?.locationInfo?.name}</Text>
  <Text>Goal: {workoutData?.title?.title}</Text>
  <Text>Body Part: {workoutData?.bodyPart?.title}</Text>
  <Text>Level: {workoutData?.level?.title}</Text>
  <TouchableOpacity onPress={() => router.replace('/create-workout/generateworkout')}>
    <Text>Build My Plan</Text>
  </TouchableOpacity>
</View>
```

Figure 4.2.7 Review Workout

The code sample creates a view that displays exercise data such as location, goal, body part, and level. It used Text components to display each item of information extracted from workoutData. There is also a TouchableOpacity that, when pushed, takes the user to the /create-workout/generateworkout route, labeled "Build My Plan."

Track Workout Plan

The UserWorkoutList component tracks workout progress by using checkboxes to designate workouts as complete. The code sections listed below enable this functionality:

1. State Management for Progress Tracking

State Variables:

- **completedExercises:** Tracks which exercises have been done. It's an object with docId keys (the identification for each workout document) and values that reflect days and workouts.
- **docCompletion:** Percentages are stored for each exercise document.

2. Toggle Completion of Exercises

```
// Handle toggle of exercise completion
const handleExerciseToggle = async (docId, day, exerciseIndex) => {
    setCompletedExercises(prev => {
        const updated = {
            ...prev,
            [docId]: {
                ...prev[docId],
                [day]: {
                    ...prev[docId]?.[day],
                    [exerciseIndex]: !prev[docId]?.[day]?.[exerciseIndex]
                }
            }
        };
        console.log('Updated completedExercises:', updated); // Debug state
        return updated;
    });

    // Update Firestore
    const workoutRef = doc(db, 'UserWorkout', docId);
    try {
        await updateDoc(workoutRef, {
            [`completedExercises.${day}.${exerciseIndex}`]: !completedExercises[docId]?.[day]?.[exerciseIndex]
        });
    } catch (error) {
        console.error('Error updating Firestore:', error);
    }
};
```

Figure 4.2.8 Handle Exercise Toggle

This function is triggered when a user interacts with a checkbox. It updates the completedExercises state to reflect whether an exercise is marked as completed or not. The function changes the local state and Firestore databases with the exercise's new completion status. It leverages Firebase Firestore's updateDoc to synchronize the local state with the cloud database.

3. Calculate Completion Percentage

```
// Calculate completion percentage for exercises in a given document
const calculateDocCompletionPercentage = (docId) => {
    const workoutPlan = sortedWorkouts.find(workout => workout.docId === docId)?.workoutData?.workout_plan;
    if (!workoutPlan || !workoutPlan.workout_schedule) return 0;

    const totalExercises = workoutPlan.workout_schedule.reduce((total, day) => total + day.exercises.length, 0);
    const completedExercisesCount = Object.values(completedExercises[docId] || {}).reduce((total, day) => {
        return total + Object.values(day).filter(Boolean).length;
    }, 0);

    return (completedExercisesCount / totalExercises) * 100;
};
```

Figure 4.2.9 Calculate Completion Percentage

Total exercises are computed by adding together all the exercises on the training plan. Completed exercises are tallied by state and then utilized to calculate percentages.

4. Updating Document Completion Percentages

```
// Update doc completion percentages
useEffect(() => {
    const newDocCompletion = {};
    sortedWorkouts.forEach(workout => {
        newDocCompletion[workout.docId] = calculateDocCompletionPercentage(workout.docId);
    });
    setDocCompletion(newDocCompletion);
}, [completedExercises]);
```

Figure 4.2.10 Updating Document Completion Percentages

useEffect updates the docCompletion state whenever completedExercises changes. This effect recalculates the completion % for all workouts if the status of completed exercises changes.

Authorization

Sign In:

const [email, setEmail] = useState('')
const [password, setPassword] = useState('')
const [error, setError] = useState('')

email and password: Store user inputs for the login form.

error: Stores any error messages that occur during the sign-in process.

```
const onSignIn = async () => {
    if (!email || !password) {
        setError('Please enter all details')
        return
    }

    try {
        await signInWithEmailAndPassword(auth, email, password)
        console.log('User signed in successfully')
        // Clear the error message on successful sign-in
        setError('')
        router.replace('/mytrip')
        // Navigate to the next screen if needed
    } catch (error) {
        const errorMessage = error.message
        console.error('Sign-in error:', errorMessage)
        setError(errorMessage)  // Set error message to display to the user
    }
}
```

Figure 4.2.11 Sign in Function

- **Validation:** Ensures both email and password are provided.
- **Authentication:** Uses Firebase's signInWithEmailAndPassword to attempt login.
- **Error Handling:** Captures and displays any error messages if authentication fails.
- **Success Handling:** On successful sign-in, navigates the user to the /mytrip screen and clears any error messages.

Sign Up:

const [email, setEmail] = useState('')

const [password, setPassword] = useState('')

const [fullName, setFullName] = useState('')

const [error, setError] = useState('')

email, password, fullName: Store user inputs for the sign-up form.

error: Stores any error messages that occur during the sign-up process.

```
const OnCreateAccount = async () => {
  if (!email || !password || !fullName) {
    setError('Please enter all details')
    return
  }

  try {
    await createUserWithEmailAndPassword(auth, email, password)
    console.log('User created successfully')
    // Clear the error message on successful sign-up
    setError('')
    router.replace('/mytrip')
    // Navigate to the sign-in screen or another screen as needed
  } catch (error) {
    const errorMessage = error.message
    console.error('Sign-up error:', errorMessage)
    setError(errorMessage)  // Set error message to display to the user
  }
}
```

Figure 4.2.12 Sign Up Function

- **Validation:** Checks if all required fields (email, password, fullName) are filled.
- **Authentication:** Uses Firebase's createUserWithEmailAndPassword method to create a new user.
- **Error Handling:** Captures and displays any error messages if the sign-up process fails.
- **Success Handling:** On successful sign-up, navigates the user to the /mytrip screen and clears any error messages.

Sit Up Tracker

```
const [situpCount, setSitupCount] = useState(0);
const [isTracking, setIsTracking] = useState(false);
const [subscription, setSubscription] = useState(null);
const [fadeAnim] = useState(new Animated.Value(1));
const [startTime, setStartTime] = useState(0);
const [timer, setTimer] = useState(0);
const [showTips, setShowTips] = useState(false);
const [showHistory, setShowHistory] = useState(false);
const [customThreshold, setCustomThreshold] = useState(0.5);
const [customSensitivity, setCustomSensitivity] = useState(100);
const [goal, setGoal] = useState(0);
const [currentGoal, setCurrentGoal] = useState(0);
const [exerciseHistory, setExerciseHistory] = useState([]);
const [modalVisible, setModalVisible] = useState(false);
```

Figure 4.2.13 State variable for sit up tracker

situpCount: Monitors the total number of sit-ups completed during a monitoring session.

isTracking: A Boolean value indicating whether or not sit-up tracking is active.

subscription: Represents the subscription object for accelerometer data updates.

fadeAnim: An animated character.The value used to animate the sit-up count display.

startTime: The timestamp when tracking begins; used to compute elapsed time.

The timer displays the elapsed time in seconds during the tracking session.

showTips: A Boolean value that controls the appearance of the workout tips modal.

showHistory: A Boolean value that governs the visibility of the workout history modal.

customThreshold: Sets the threshold for identifying sit-up action based on accelerometer data.

customSensitivity: Determines the sensitivity for recognizing a sit-up, which influences how accurate the motion detection is.

goal: Saves the user's objective number of sit-ups.

currentobjective: Tracks progress toward the objective and refreshes it when additional sit-ups are completed.

exerciseHistory: An array that maintains track of prior exercise sessions, including the date, number of sit-ups, and duration.

modalVisible: Determines whether the settings modal is visible or not.

1. Tracking Effect

```
useEffect(() => {
  if (isTracking) {
    _subscribe();
    setStartTime(Date.now());
    const interval = setInterval(() => {
      setTimer(Math.floor((Date.now() - startTime) / 1000));
    }, 1000);
    return () => {
      _unsubscribe();
      clearInterval(interval);
    };
  } else {
    _unsubscribe();
  }
}, [isTracking]);
```

Figure 4.2.14 Effect hook for sit up tracker

- **Purpose**: Controls tracking behavior depending on the isTracking state. When tracking begins, it subscribes to accelerometer updates and initiates a timer. When tracking ceases, it unsubscribes from accelerometer updates and resets the timer interval.

- **Clean-up**: Ensures that subscriptions and intervals are correctly cleared to prevent memory leaks.

2. Sit-Up Detection Logic

```
const detectSitup = ({ x, y, z }) => {
  const now = Date.now();
  const acceleration = Math.sqrt(x * x + y * y + z * z);

  switch (situpState.current) {
    case 'waiting':
      if (z < -customThreshold) {
        situpState.current = 'going_down';
      }
      break;
    case 'going_down':
      if (z > customThreshold) {
        situpState.current = 'going_up';
      }
      break;
    case 'going_up':
      if (Math.abs(z) < 0.2 && now - lastSitupTime.current > customSensitivity) {
        setSitupCount(prevCount => prevCount + 1);
        setCurrentGoal(prevGoal => Math.max(prevGoal, situpCount + 1));
        lastSitupTime.current = now;
        situpState.current = 'waiting';
      }
      break;
  }
};
```

Figure 4.2.15 Sit-Up Detection Logic

- **Purpose**: Detects sit-ups using accelerometer data. It switches between various states (waiting, going_down, and going_up) to precisely recognize the end of a sit-up.

Uses configurable thresholds and sensitivity.Threshold and customSensitivity are used to fine-tune the detecting process.

3. Tracking Control

```
const toggleTracking = () => {
  if (isTracking) {
    _unsubscribe();
    setIsTracking(false);
    setExerciseHistory([...exerciseHistory, { date: new Date(), situps: situpCount, duration: timer }]);
    Alert.alert('Tracking Stopped', `You did ${situpCount} sit-ups in ${timer} seconds!`);
  } else {
    _subscribe();
    setIsTracking(true);
    setStartTime(Date.now());
    setSitupCount(0);
    setTimer(0);
    situpState.current = 'waiting';
    lastSitupTime.current = 0;
  }
};
```

Figure 4.2.16 Tracking Control

Starts or stops tracking depending on the current condition. If tracking is turned off, it stores the session's data to exerciseHistory and provides a warning with the results.

Nearby Feature

1. useEffect Hook for Requesting Location and Fetching Gyms

```
useEffect(() => {
  (async () => {
    let { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') {
      console.log('Permission to access location was denied');
      return;
    }

    let loc = await Location.getCurrentPositionAsync({});
    setLocation(loc.coords);
    setRegion({
      latitude: loc.coords.latitude,
      longitude: loc.coords.longitude,
      latitudeDelta: 0.0922,
      longitudeDelta: 0.0421,
    });

    fetchNearbyGyms(loc.coords.latitude, loc.coords.longitude);
  })();
}, []);
```

Figure 4.2.17 useEffect for requesting Location and Fetching gyms

- **Requesting Location Permissions**: The useEffect is executed once when the component is mounted. It seeks permission to access the user's location using Location.requestForegroundPermissionsAsync().

- **Obtaining Current Position**: If permission is given, Location.GetCurrentPositionAsync() returns the current position coordinates (latitude and longitude).

- **Updating State**: After retrieving the location, setLocation changes the location state, and setRegion updates the map region.

- **Fetching Gyms**: fetchNearbyGyms receives the user's location and uses the Google Places API to find nearby gyms.

2. Fetch Nearby Gyms

Figure 4.2.18 Fetching Gym Function

API call: This method utilizes Axios to make a GET call to the Google Places API. It takes the user's latitude and longitude, a radius of 5000 meters, and the kind of gym.

Updating Gyms State: The API response (response.data.results) contains an array of nearby gym locations, which are saved in the gyms state using setGyms.

Loading indication: After collecting the gyms, setLoading(false) disables the loading indication.

3. **Display gyms on the map**

```
{gyms.map(gym => (
  <Marker
    key={gym.place_id}
    coordinate={{
      latitude: gym.geometry.location.lat,
      longitude: gym.geometry.location.lng,
    }}
    title={gym.name}
    onPress={() => handleTap(gym)}
  />
))}
```

Figure 4.2.19 Gym markers

- **Markers for Gyms**: A marker is put on the map at the coordinates (latitude and longitude) of each gym in the state, with the gym's name as the marker title. Handling When you touch a gym marker, handleTap(gym) is called, which either navigates the map to the gym's location or displays its data.

4. **Getting User's Current Location**

```
let { status } = await Location.requestForegroundPermissionsAsync();
if (status !== 'granted') {
  console.log('Permission to access location was denied');
  return;
}

let loc = await Location.getCurrentPositionAsync({});
setLocation(loc.coords);
```

Figure 4.2.20 Permission and getting user's location

- **Request Permissions:** The app uses Location.requestForegroundPermissionsAsync() to request permission to use the user's location.

If permission is granted, the program will continue to get the location. If permission is refused, it records a message and terminates further execution (via return).

- **Retrieve the Location:** Once permission is given, Location.getCurrentPositionAsync() is used to get the user's current geographic coordinates.

It returns an object containing information about the user's position, including latitude and longitude.

- **Set Location State:** The setLocation function keeps the user's coordinates (loc.coords) in the location state, allowing you to utilize them later.

```
setRegion({
  latitude: loc.coords.latitude,
  longitude: loc.coords.longitude,
  latitudeDelta: 0.0922,
  longitudeDelta: 0.0421,
});
```

Figure 4.2.21 Update Map Region

- **Setting the Map's View:** Once you've acquired the user's location, use setRegion to update the map's region. The map's territory contains:
- **Latitude:** The user's current latitude.
- **Longitude:** The user's current longitude. LatitudeDelta and Longitude: These settings determine the map's zoom level.

  This adjusts the map to center on the user's current position at the specified zoom level.

6. Fetching Gym Details and Reviews

When a user taps on a gym marker, the app retrieves comprehensive information about the gym using the handleTap method. If the user clicks twice fast, the showGymDetails function is invoked to get comprehensive information, including reviews, for that gym:

```
const showGymDetails = async (item) => {
  const response = await axios.get(
    `https://maps.googleapis.com/maps/api/place/details/json?place_id=${item.place_id}&key=${GOOGLE_MAPS_API_KEY}`
  );
  setSelectedGym(response.data.result);
};
```

Figure 4.2.22 showGymDetails Function

- **Request a URL:** The request is sent to the Google Places Details API (place/details/json) using the gym's place_id and your Google Maps API key. The answer includes specific information about the gym, such as reviews, ratings, and location.

- **Store Gym Details:** Once the API answer is received, the detailed gym information is saved in the selectedGym state using setSelectedGym(response.data.result). This includes gym reviews, which may be accessed under response.data.result.reviews.

```
<FlatList
  style={styles.reviewsContainer}
  data={selectedGym.reviews}
  renderItem={({ item }) => (
    <View style={styles.reviewContainer}>
      <Image
        source={item.profile_photo_url ? { uri: item.profile_photo_url } : require('./../../assets/images/userprofile.jpg')}
        style={styles.reviewImage}
      />
      <View style={styles.reviewContent}>
        <Text style={styles.reviewAuthor}>{item.author_name}</Text>
        <Text style={styles.reviewText}>{item.text}</Text>
        <Text style={styles.reviewRating}>Rating: {item.rating} ★</Text>
      </View>
    </View>
  )}
  keyExtractor={item => item.author_name}
  ListEmptyComponent={<Text style={styles.noReviews}>No reviews available</Text>}
/>
```

Figure 4.2.23 Display Gym Review in Modal

- **Data Source:** The FlatList's data source is selectedGym.reviews, which contains an array of reviews for that gym. Rendering Each Review.
- **Each review includes the following information:** The author name is shown using {item.author_name}.
- Review text is shown using {item.text}. The rating is shown using {item.rating} ⭐. Profile picture: If the author has a profile picture (item.profile_photo_url), it appears in the review; otherwise, a default image is shown.
- If the gym does not have any reviews, the FlatList's ListEmptyComponent displays the message "No reviews available":

Feed Page

This is a feed page in the Discover.jsx file where users may publish articles about their workouts, nutrition, goals, and other fitness-related topics. They may upload text or photographs and share them with others.

1. Posting new post

```
const handleAddPost = async () => {
  if (!currentUser) return;

  setPosting(true);

  try {
    let imageUrl = '';

    if (image) {
      const imageRef = ref(storage, `images/${new Date().toISOString()}_${image.split('/').pop()}`);
      const response = await fetch(image);
      const blob = await response.blob();
      await uploadBytes(imageRef, blob);
      imageUrl = await getDownloadURL(imageRef);
    }

    const email = currentUser.email || '';

    await addDoc(collection(db, 'discover'), {
      content: newPost.trim() || null,
      timestamp: new Date().toISOString(),
      userId: currentUser.uid,
      userEmail: email,
      imageUrl,
    });

    console.log("Post added successfully");

    const postsCollection = collection(db, 'discover');
    const q = query(postsCollection, orderBy('timestamp', 'desc'));
    const querySnapshot = await getDocs(q);
    const postsArray = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));

    console.log("Posts after adding:", postsArray);

    setNewPost('');
    setImage(null);
    setPosts(postsArray);

    Alert.alert("Success", "Post added successfully!");
  } catch (error) {
    console.error("Error adding post:", error);
    Alert.alert("Error", "There was an issue adding the post. Please try again.");
  } finally {
    setPosting(false);
  }
};
```

Figure 4.2.24 Posting new post

- **Text Input:** The user enters their post into a TextInput form. If the post contains just text, it will be uploaded to Firebase as part of the post content.

- **Image submits:** Users may submit images from their devices. This utilizes Expo's ImagePicker to access the user's picture gallery and enable them to choose an image. When users pick a picture, it is uploaded to Firebase Storage, and the URL is kept in the post.

When a user uploads, the content, picture (if applicable), and user information (such as email and timestamp) are posted to Firestore's discover collection. Following a successful submission, the feed is updated to display the most recently contributed material.

2. Display post

```
useEffect(() => {
  if (!currentUser) return;

  const fetchPosts = async () => {
    try {
      console.log("Fetching posts...");
      const postsCollection = collection(db, 'discover');
      const q = query(postsCollection, orderBy('timestamp', 'desc'));
      const querySnapshot = await getDocs(q);

      if (querySnapshot.empty) {
        console.log("No posts found");
      }

      const postsArray = querySnapshot.docs.map(doc => ({
        id: doc.id,
        ...doc.data()
      }));

      console.log("Posts fetched:", postsArray);
      setPosts(postsArray);
    } catch (error) {
      console.error("Error fetching posts:", error);
    } finally {
      setLoading(false);
    }
  };

  fetchPosts();
}, [currentUser]);
```

Figure 4.2.25 Display post

Once posted, posts are displayed on a feed for all to see. This is accomplished by utilizing Firebase's getDocs to get all posts from the discover collection, which are then arranged chronologically (newest first).

Post Rendering: Each post in the feed contain

- The user's email address
- Content (if it's a text post)
- An picture (if one was submitted).
- A timestamp that indicates when the post was made.
- A share button allows you to share the content on other social networking networks.

- **Display using FlatList:** The FlatList component uses the renderPost method to display each post on the screen, including the post content (text, picture, and user information) and the share button.

3. Sharing a Post

Each post has a share button, allowing the user to share it on other social networking networks. This use React Native's Share API. When the user clicks the share button, the post's content (text and picture URL) is shared with a configurable message, such as "Check out this post!"

```
const handleSharePost = async (post) => {
  try {
    const shareOptions = {
      title: 'Check out this post!',
      message: post.content,
      url: post.imageUrl,
    };

    const result = await Share.share(shareOptions);

    if (result.action === Share.sharedAction) {
      if (result.activityType) {
        // Shared with activity type of result.activityType
      } else {
        // Shared
      }
    } else if (result.action === Share.dismissedAction) {
      // Dismissed
    }
  } catch (error) {
    Alert.alert('Error', 'Failed to share post.');
    console.error('Error sharing post: ', error);
  }
};
```

Figure 4.2.26 Share post to Other Platforms

- This feature displays a share window in which users may choose a social networking app or other platform to share the content.

4.3 Circuits and Components Design

Though the AHLI: Workout Management Mobile App is entirely software-based, with no physical hardware involved, this section will concentrate on the design of the logical components and how they interact with various services, APIs, and data storage systems such as Firebase and external APIs such as the Google Places API.

The primary goal of the AiModal.js code is to integrate Google Gemini AI to develop individualized training routines.

AiModal.js

1. **Importing and initializing Google Generative AI**

```
const {
  GoogleGenerativeAI,
  HarmCategory,
  HarmBlockThreshold,
} = require("@google/generative-ai");

const apiKey = process.env.EXPO_PUBLIC_GOOGLE_GEMINI_API_KEY;
const genAI = new GoogleGenerativeAI(apiKey);
```

Figure 4.3.1 Import and initializing Google Generative AI

- **GoogleGenerativeAI:** This module allows you to use Google's AI to generate text answers.
- **apiKey:** This key allows your app to utilize Google Gemini AI services. GenAI is a Google AI service instance that has been started using your API key.

2. **Fetching and setting the Model**

```
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

const generationConfig = {
  temperature: 1,
  topP: 0.95,
  topK: 64,
  maxOutputTokens: 8192,
  responseMimeType: "application/json",
};
```

Figure 4.3.2 Fetching and setting the Model

The getGenerativeModel function returns the exact AI model (gemini-1.5-flash) that will be used to produce exercise programs. This model is developed to handle conversational AI tasks.

- **Temperature:** Controls the unpredictability of the AI output. A rating of one indicates a balanced amount of inventiveness.

- **topP:** Controls diversity, enabling the model to focus on the top 95% of token probabilities.

- **maxOutputTokens:** Restricts the number of tokens in the answer. This avoids the output from becoming too lengthy.

- **responseMimeType:** specifies the response format, which in this instance is application/json for structured output.

3. Starting AI Chat Session

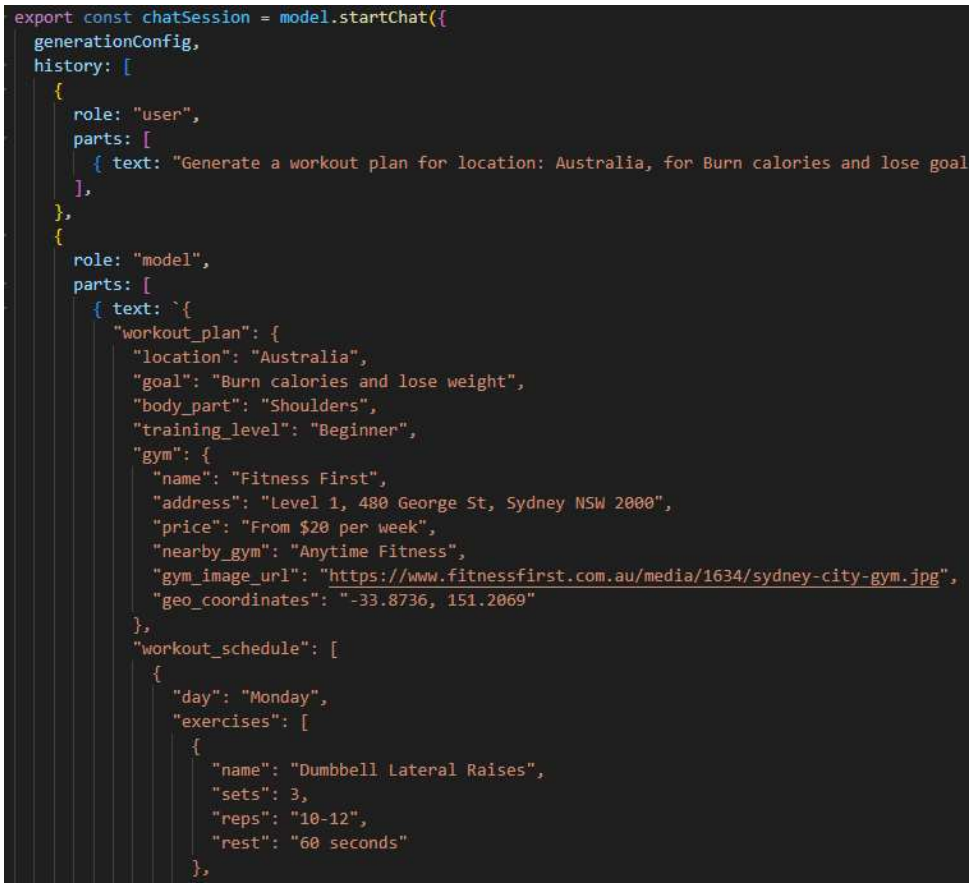


Figure 4.3.3 AI Chat session

- **History:** The AI interacts with the conversation history. The first item is the user's request for a workout plan, while the second is the model's answer, which includes a workout plan in JSON format.

- The role specifies who is speaking (user for the request and model for the answer).

Options.js

The Options.js file is crucial in giving the user options for their exercise preferences, such as objectives, body parts, and training levels, as well as creating a prompt to submit to Google Gemini AI.

1. SelectGoalList

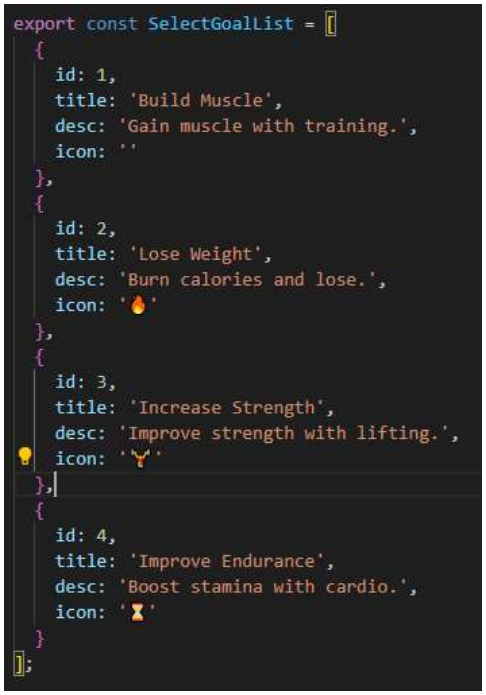


Figure 4.3.4 SelectGoalList

Users choose a fitness objective, which determines the emphasis of their training program.

Key Properties:

- **id:** A unique identifier for each objective.
- **Title:** The name of the aim (for example, "Build Muscle").
- **description:** An explanation of what the aim includes.
- **Icon:** An emoji indicating the aim that adds visual appeal.

2. SelectBodyList

```
export const SelectBodyList = [
  {
    id: 1,
    title: 'Chest',
    desc: 'Unleash powerful upper strength',
    image: require('./../assets/images/chest.png')
  },
  {
    id: 2,
    title: 'Back',
    desc: 'Achieve a stronger back',
    image: require('./../assets/images/back.png')
  },
  {
    id: 3,
    title: 'Legs',
    desc: 'Build unstoppable leg power',
    image: require('./../assets/images/leg.png')
  },
  {
    id: 4,
    title: 'Shoulders',
    desc: 'Elevate your shoulder strength',
    image: require('./../assets/images/shoulder.png')
  },
```

Figure 4.3.5 SelectBodyList

Allows users to target a certain body area throughout their exercise.

Key Properties:

- **id**: A unique identification for each bodily component.
- **Title**: A physical portion (for example, "Chest").
- **desc**: A brief summary of what the training for that body area seeks to accomplish.
- **image**: A graphic representation of the bodily component, imported from the app's assets.

3. SelectLevelList

```
export const SelectLevelList = [
  {
    id: 1,
    title: 'Beginner',
    desc: 'Starting out? This is for you',
    icon: '★'
  },
  {
    id: 2,
    title: 'Intermediate',
    desc: 'Ready for more? Step up!',
    icon: '🚀'
  },
  {
    id: 3,
    title: 'Advanced',
    desc: 'Challenge yourself with advanced workouts',
    icon: '🔥'
  },
  {
    id: 4,
    title: 'Expert',
    desc: 'For those who push their limits',
    icon: '🏆'
  }
];
```

Figure 4.3.6 SelectLevelList

This category allows users to choose their fitness level, which affects the complexity of the training plan.

Key Properties:

- **id**: This is the level's unique identification.
- **Title**: The name of the level (for example, "Beginner").
- **desc**: A description that helps people choose which level is best for them.

- **Icon**: An emoji that represents the level and provides a visual clue.

4. Ai Prompt Template

export const AI_PROMPT = 'Generate a workout plan for location: {location}, for {goal} goal and for the body part, {bodyPart} and for training level {level}, with gym name, gym address, price, gym image url, geo coordinates in JSON format';

- A template prompt for requesting a fitness plan from the AI.

Structure:

- {location} is a placeholder for where the exercise should be personalized.
- {goal}: A placeholder for the user's fitness objective.
- {bodyPart}: A placeholder for the selected body part.
- {level}: A placeholder for the user's training level.
- The prompt instructs the AI to build a training plan in JSON format, including gym information.

FirebaseConfig.js

```
// Import the functions you need from the SDKs you need
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';  // Import getAuth from firebase/auth
import { getFirestore } from "firebase/firestore";
import { getStorage } from "firebase/storage";

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: 'AIzaSyAKQKu9-FJeT-17d9TYkt5VjTtn7-8qkK0',
  authDomain: 'ahliexpo-f19d0.firebaseapp.com',
  projectId: 'ahliexpo-f19d0',
  storageBucket: 'ahliexpo-f19d0.appspot.com',
  messagingSenderId: '592883602181',
  appId: '1:592883602181:web:dd997418e17047e43cdd6c'
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const auth = getAuth(app)
export const db = getFirestore(app);
export const storage = getStorage(app);
```

Figure 4.3.7 Firebase Config

1. **Import Functions**

- The initializeApp command is used to configure Firebase with project parameters.
- getAuth provides access to Firebase Authentication for managing user accounts.
- getFirestore sets up Firestore to handle database activities.
- GetStorage handles file storage functions.

2. **Firebase Configuration**

- apiKey: A unique key for authenticating requests.
- authDomain: The domain for Firebase authentication services.
- The projectId is the identifier for the Firebase project.
- StorageBucket: A Cloud Storage bucket for storing files.
- messagingSenderId: The identifier for sending notifications.
- appId: A unique identifier for each app in the Firebase project.

3. **Initialize Firebase**

- initializeApp(firebaseConfig) connects the app to Firebase services based on the setup.

4. **Export Services**

- auth is initialized using getAuth(app), which grants access to authentication functions.
- The database is initialized using getFirestore(app), allowing interaction with Firestore.
- Storage is initialized using getStorage(app), which enables file operations with Firebase Storage.

4.4 System Components Interaction Operations

1. User Interaction Flow

- Home Page: Users may enter options such as location, body part, objective, and training level. Based on your inputs, the app creates a tailored fitness plan and recommends local facilities.
- Feed Page: Users may write posts with text and photographs about their exercises, nutrition, and objectives. Posts are kept in Firestore and may be shared via social media. The system retrieves and displays posts from a database.
- Sit-Up Tracker: With the Expo accelerometer, users may monitor their sit-ups. The software offers fitness recommendations, tracks sit-up history, and enables users to share their successes.
- Nearby Feature: The Google Places API allows users to browse gyms within a 5-kilometer radius. The app shows gyms as swipeable cards with information and reviews.

2. Data Flow and Integration

- Firebase Integration: Authentication: Manages user sign-in and sign-up, associating posts and data to authorized users.
- Firestore saves and retrieves posts, user info, and exercise programs. Uses the feed page to update and show content.
- Storage: Manages picture uploads for posts. Stores photos in Firebase Storage and fetches their URLs to show in posts.
- Google Generative AI supports AI integration. Creates tailored fitness regimens based on user input. The AI creates a JSON response containing exercise programs and gym specifics based on the inputs supplied (place, aim, body part, and training level).
- Google Places API: Nearby Gyms: Displays gyms within a 5km radius. Offers gym information, reviews, and photographs.

3. System Operation Workflow

To generate workout plans, users enter their preferences on the main page. The app transmits these choices to Google's Generative AI service. The AI answers with a fitness regimen and information about neighboring gyms. The app shows the user their fitness plan and gym suggestions.

- Creating and Sharing Posts: Users may make posts with text and photographs.
- The app uploads photographs to Firebase Storage and returns their URLs.
- Posts are preserved in Firestore along with user data and picture URLs.
- Posts appear on the stream, and users may share them on social media.
- Tracking Sit-Ups: The software uses the Expo accelerometer to measure user motions during sit-ups.
- The app gives feedback and tracks your sit-up history.
- Users may see their history and, if requested, share the findings.
- To see neighboring gyms, users may use the "nearby" option.
- The app searches the Google Places API for gyms within 5 km.
- Swipeable cards reveal gym features such as reviews and photographs.

Overview of File Structure

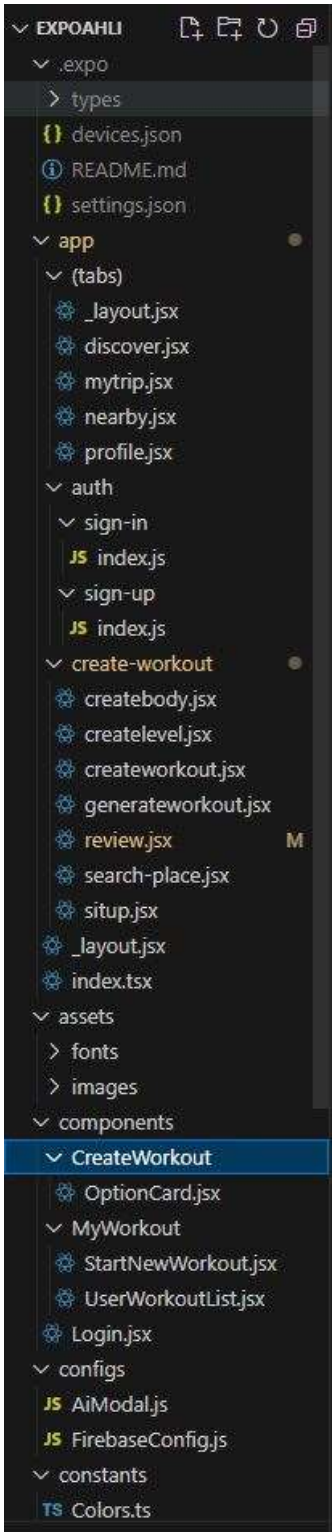


Figure 4.4.1 File Structure

The file structure in the snapshot depicts the project's architecture, showcasing the arrangement of components, pages, and assets. The following is an examination of the primary folders and their respective duties.

1. **app**

- The primary application components are organized into subfolders such as tabs, auth, and create-workout, which govern certain parts of the app's operation.
- **(tabs)** folder contains components relating to the application's navigation and tab layout (find, mytrip, nearby, profile, etc.), which define how users transition between screens.
- **auth**: Manages authentication, including the sign-in and sign-up procedures.
- **create-exercise**: Controls the workout creation flow, which includes components for choosing workout levels, producing workouts, and evaluating completed workouts.

2. **components**

- Contains reusable components (CreateWorkout, OptionCard, and MyWorkout) that are used throughout the program to promote code modularity and reusability.

3. **configs**

- Contains configuration files, such as Firebase settings (FirebaseConfig.js) and any modals (AimModal.js) used in the application.

4. **constants**

- Stores static data, like as color schemes (Colors.ts) and (Option.js) , to guarantee a consistent UI design throughout the application.

5. assets

- Includes fonts and pictures for UI components, which improve the application's visual appeal.

Chapter 5

System Implementation

5.1 Software Setup

1) Visual Studio Code (version 1.92.2)

Link : https://code.visualstudio.com/

2) Node.js (version 20.14.0)

Link : https://nodejs.org/en

3) Expo Cli (version 6.3.10)

Link : https://docs.expo.dev/more/expo-cli/

4) Expo Go (SDK 51)

Link : https://expo.dev/go

5) Git (version 2.46.0)

Link: https://git-scm.com/downloads

6) Firebase (version 13.8.0)

Link: https://console.firebase.google.com/u/0/

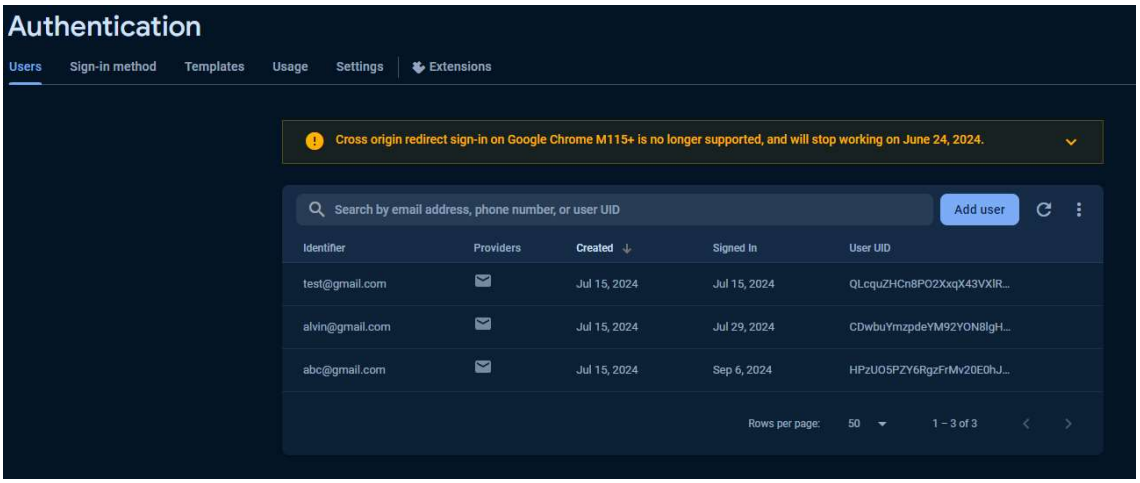


Figure 5.1.1 Firebase authentication

- When a user successfully created a new account in the app, it saves the user email to the Firebase Authentication.

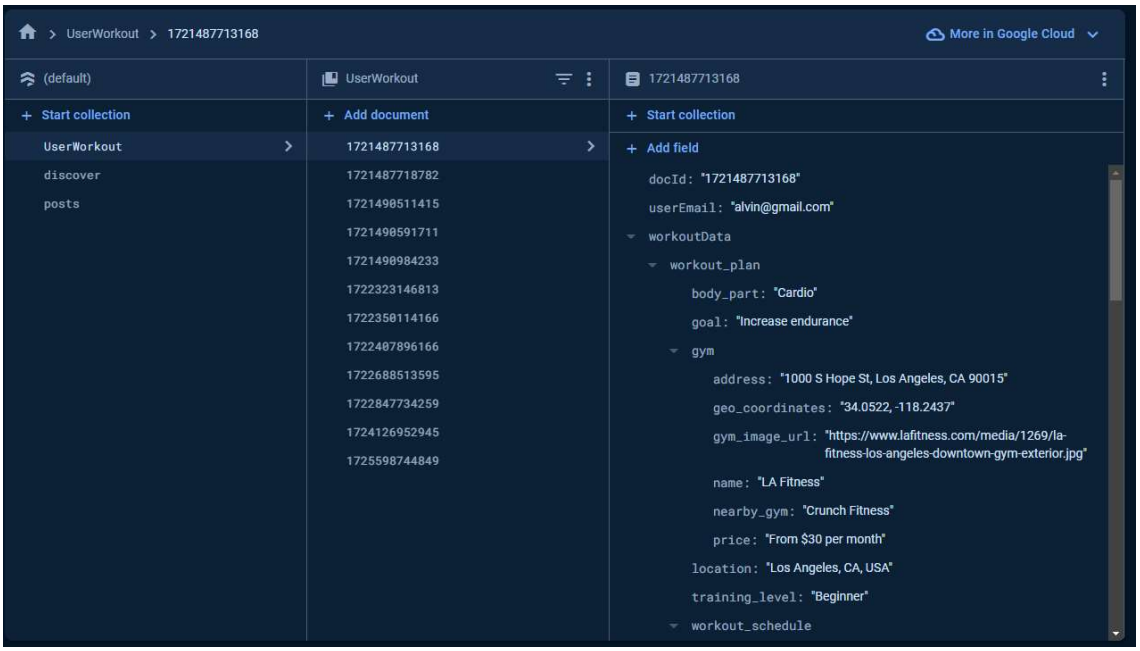


Figure 5.1.2 UserWorkout Collection

- When the AI Workout Plan is created, it is saved to "UserWorkout" collection with each unique document ID.
- **docId**: Unique document ID.
- **userEmail**: User's email.
- **workout_plan**: Includes the body part, goal, and training level.
- **gym**: Contains gym details like name, address, location, and price.
- **workout_schedule**: List of workout days, each with exercises (name, duration, intensity, sets).

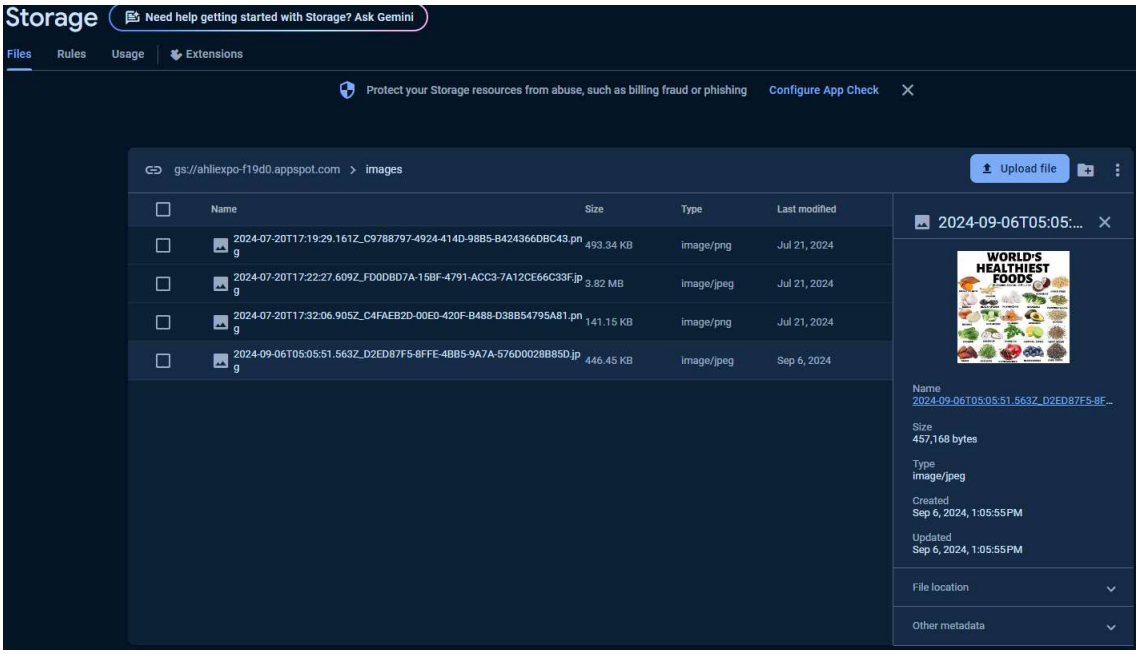


Figure 5.1.3 UserWorkout post images

- When a user successfully created a post in the Feed/ Discover page, it save the images in the Firebase Storage, with the naming of the date and time posted.

7) Google Gemini API

Gemini

Link : https://ai.google.dev/

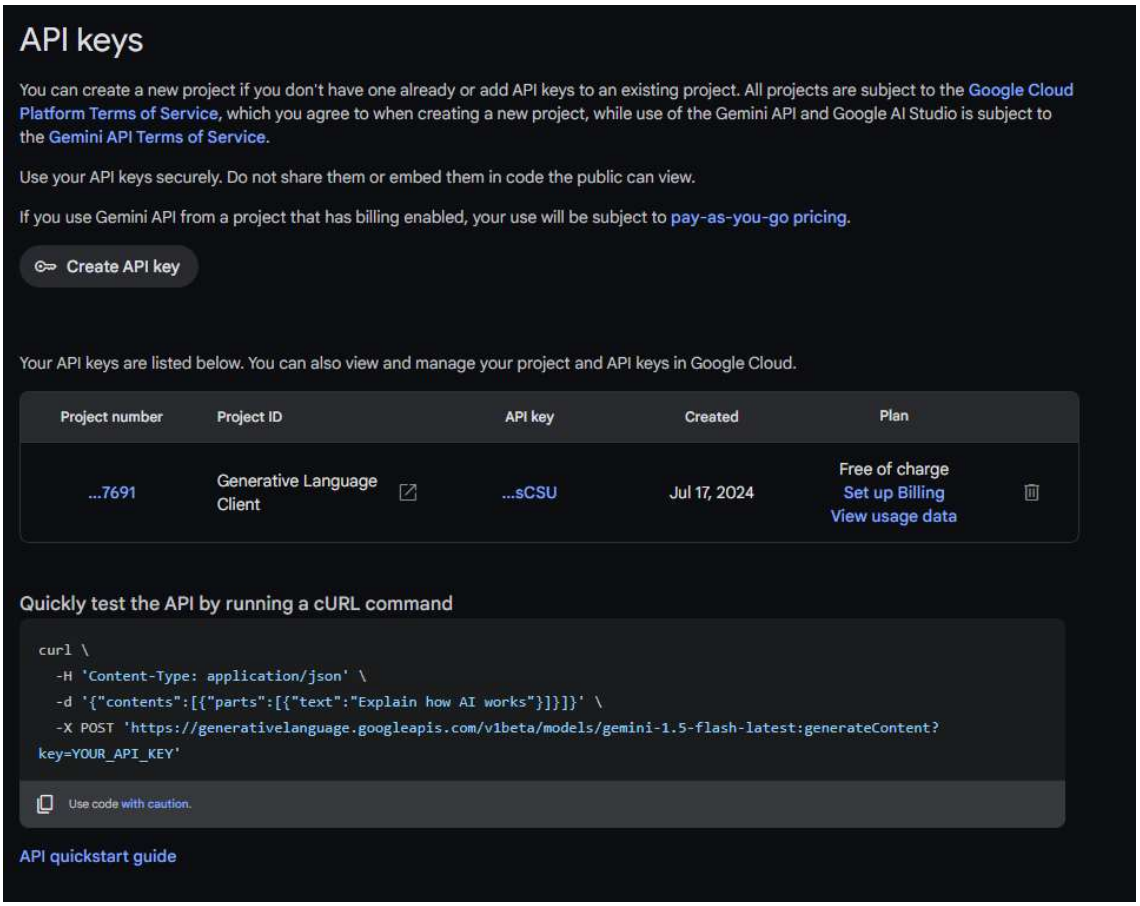


Figure 5.1.4 Google Gemini API

8) Google Places API

Google Places API

Link: https://developers.google.com/maps/documentation/places/web-service/overview

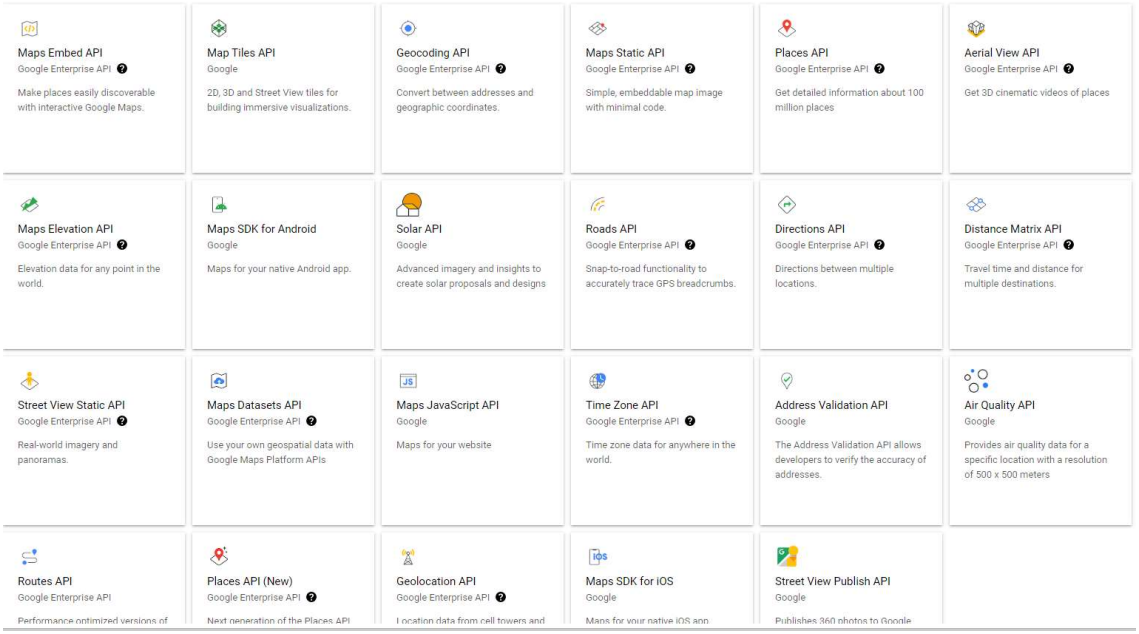


Figure 5.1.5 Google Places API

- Above are the enabled API & services from Google.

5.2 System Operation

i) Open Visual Studio

ii) Open Terminal

iii) enter command "npm start" in the Terminal

```
PS C:\Users\User\Desktop\expoahli> npm start
```

Figure 5.2.1 npm start

iv) scan the code with phone camera (ios)

Figure 5.2.2 Expo Go code

1) Onboard screen

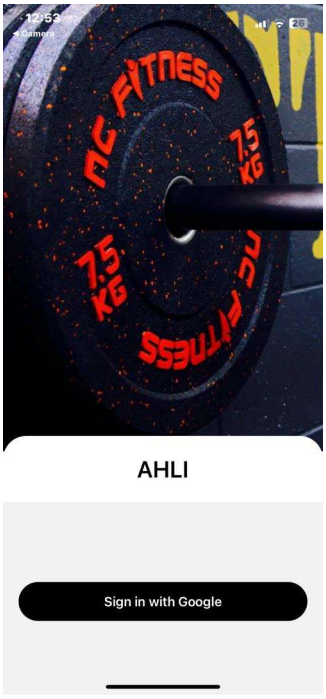


Figure 5.2.3 Onboard screen

2) Sign In screen

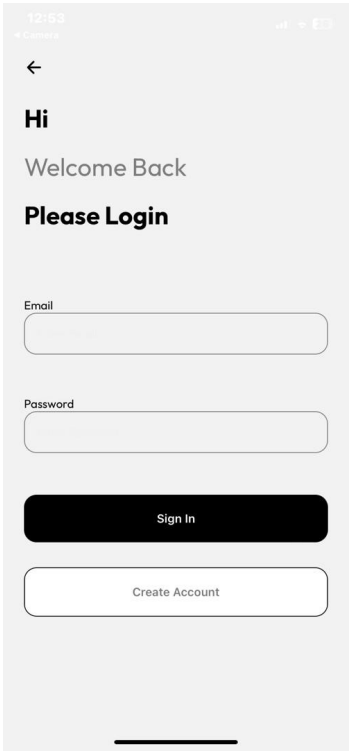


Figure 5.2.4 Sign In screen

3) Create account screen

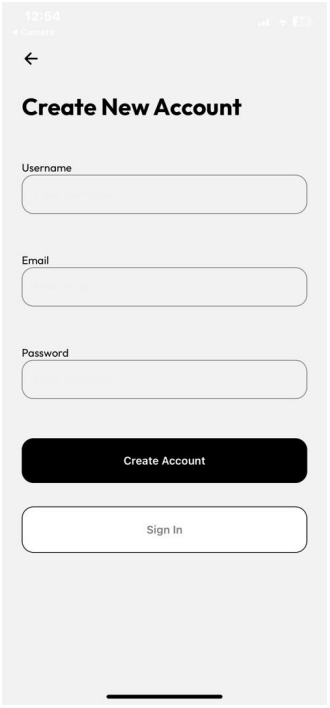


Figure 5.2.5 Create account screen

4) Home screen (without Workout Plan)

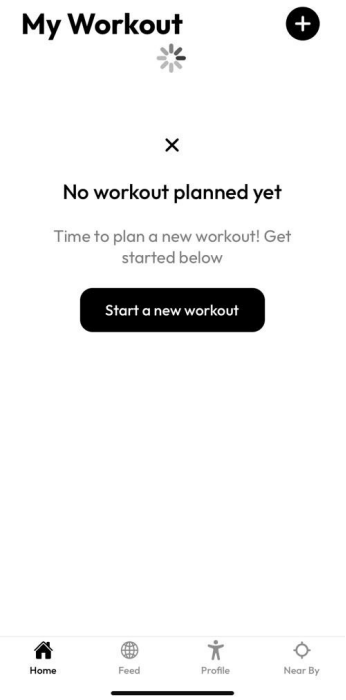


Figure 5.2.6 Home screen (without Workout Plan)

5) Home screen (with Workout Plan)

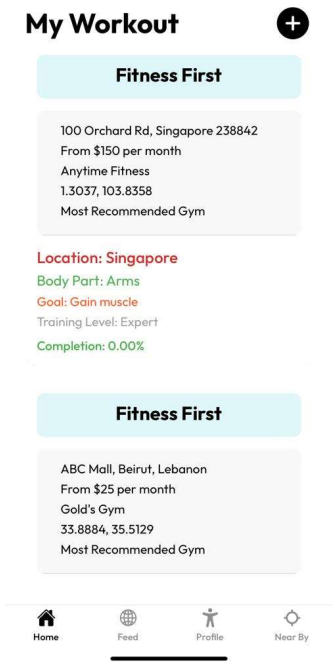


Figure 5.2.7 Home screen (with Workout Plan)

6) Workout plan screen

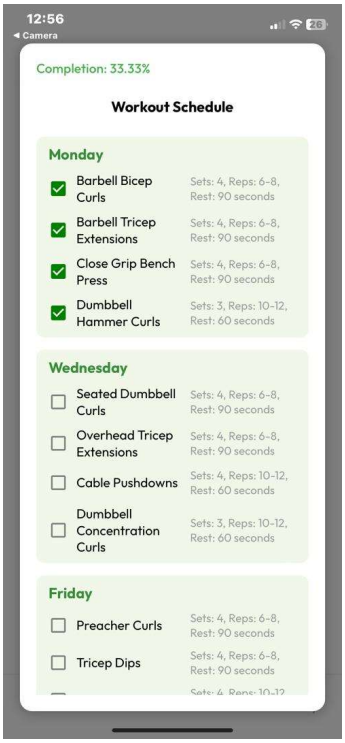


Figure 5.2.8 Workout plan screen

7) Search place screen

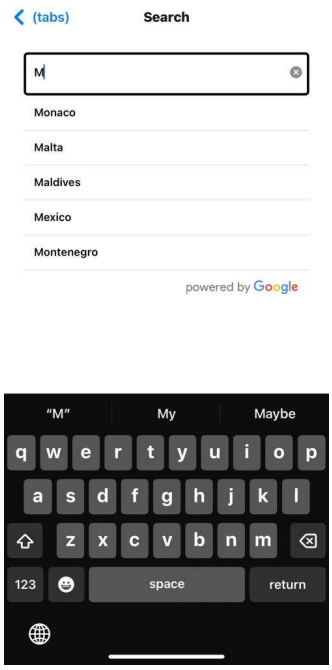


Figure 5.2.9 Search Place screen

8) Select Goal screen

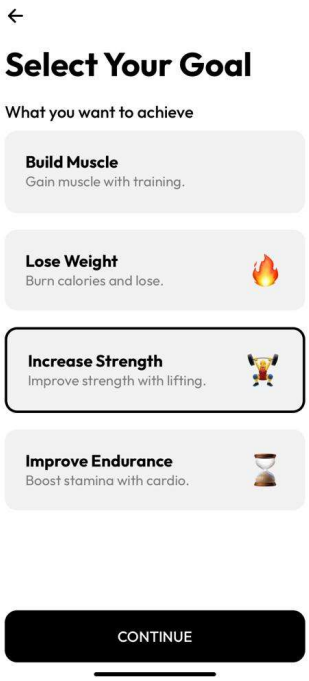


Figure 5.2.10 Select Goal screen

9) Select Body Part screen

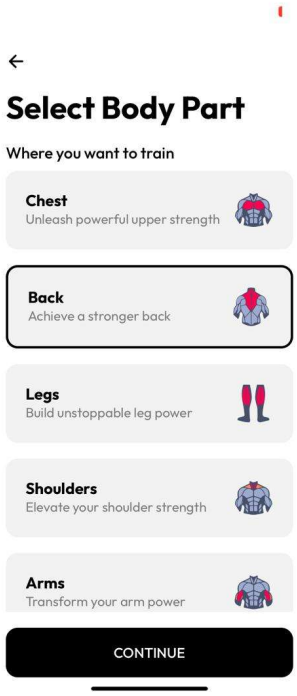


Figure 5.2.11 Select Body Part screen

10) Select Training Level screen

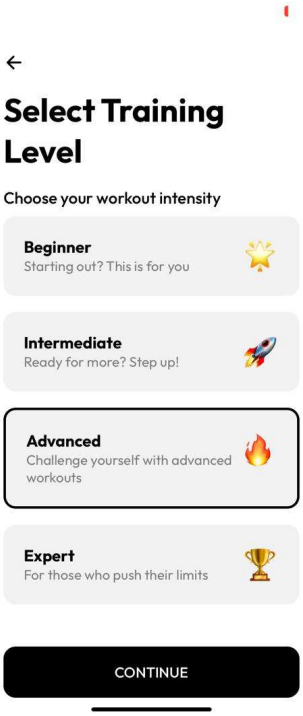


Figure 5.2.12 Select Training Level screen

11) Review Workout Plan screen

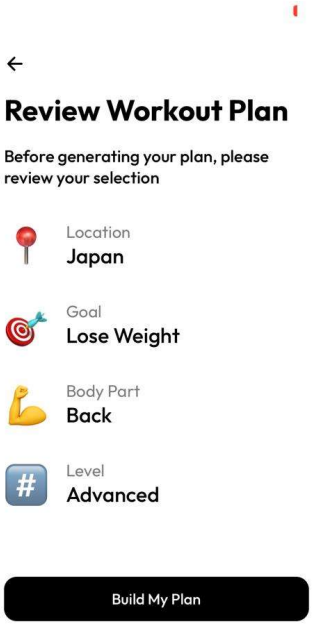


Figure 5.2.13 Review Workout Plan screen

12) Generating Workout Plan screen

Figure 5.2.14 Generating Workout Plan screen

13) Feed Page

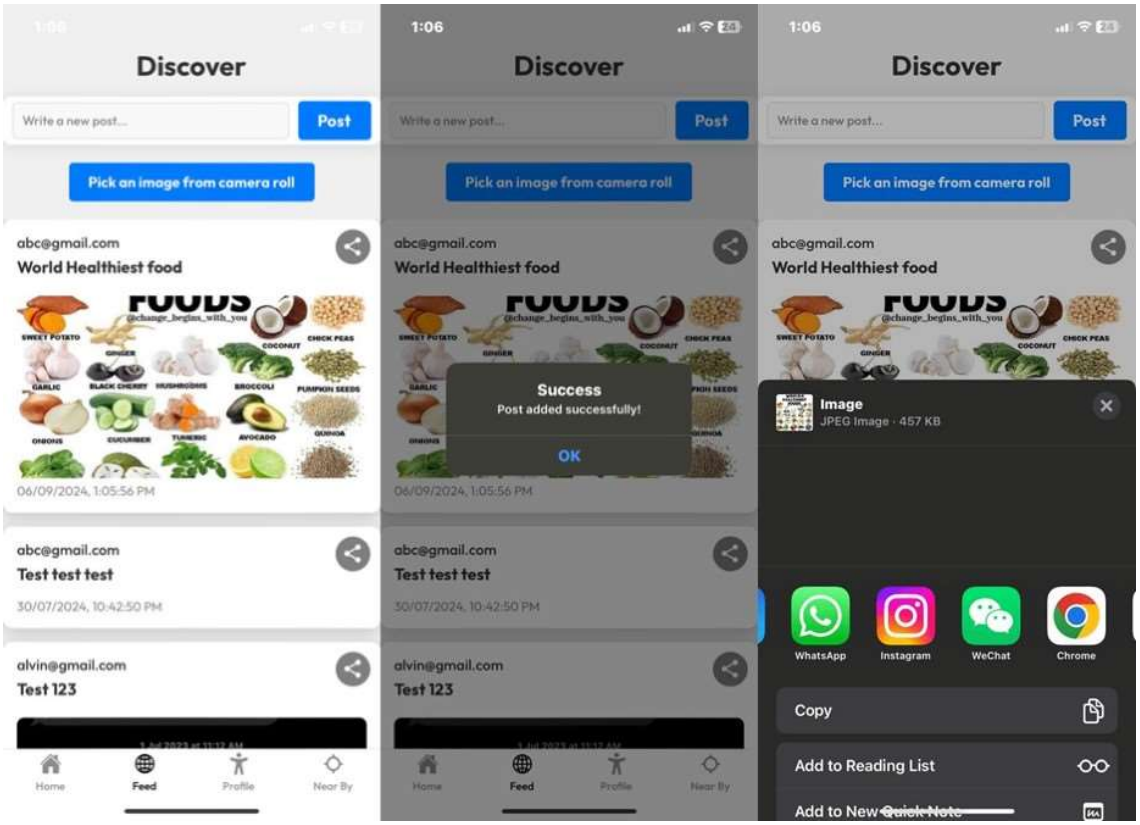


Figure 5.2.15 Feed screen (Add post successfully & Share post)

14) Situp Tracker Page

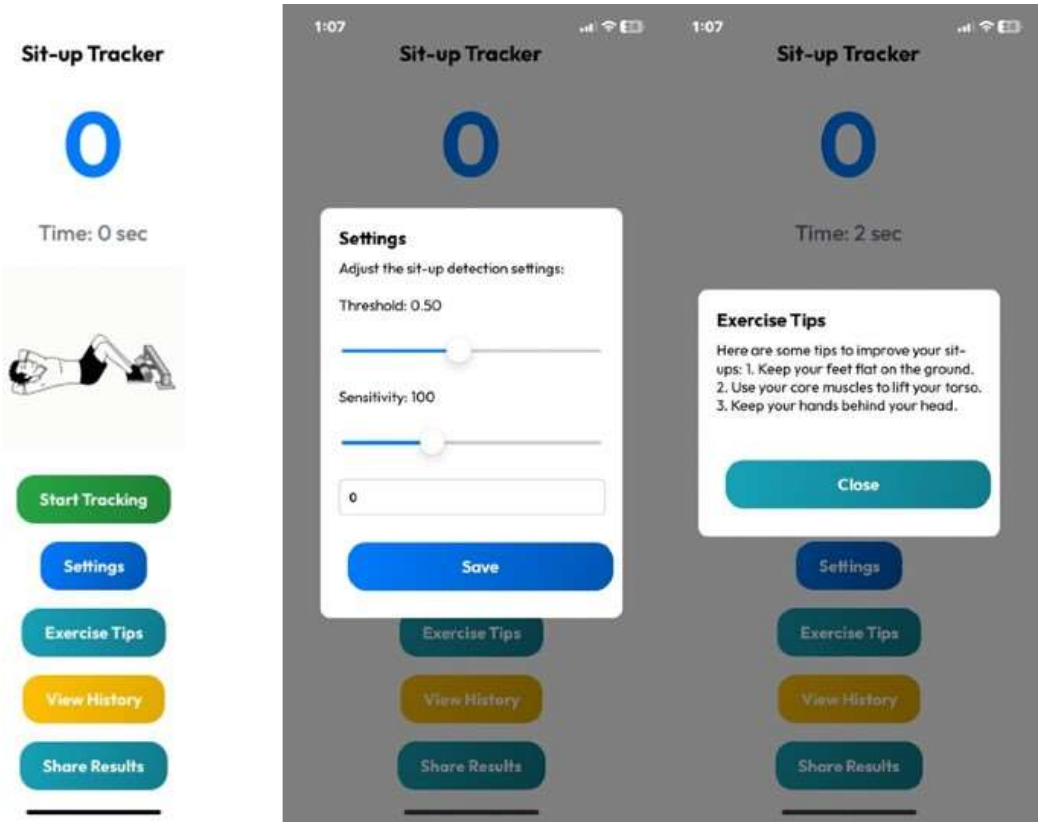


Figure 5.2.16 Sit Up Tracker screen (Settings & Exercise Tips)

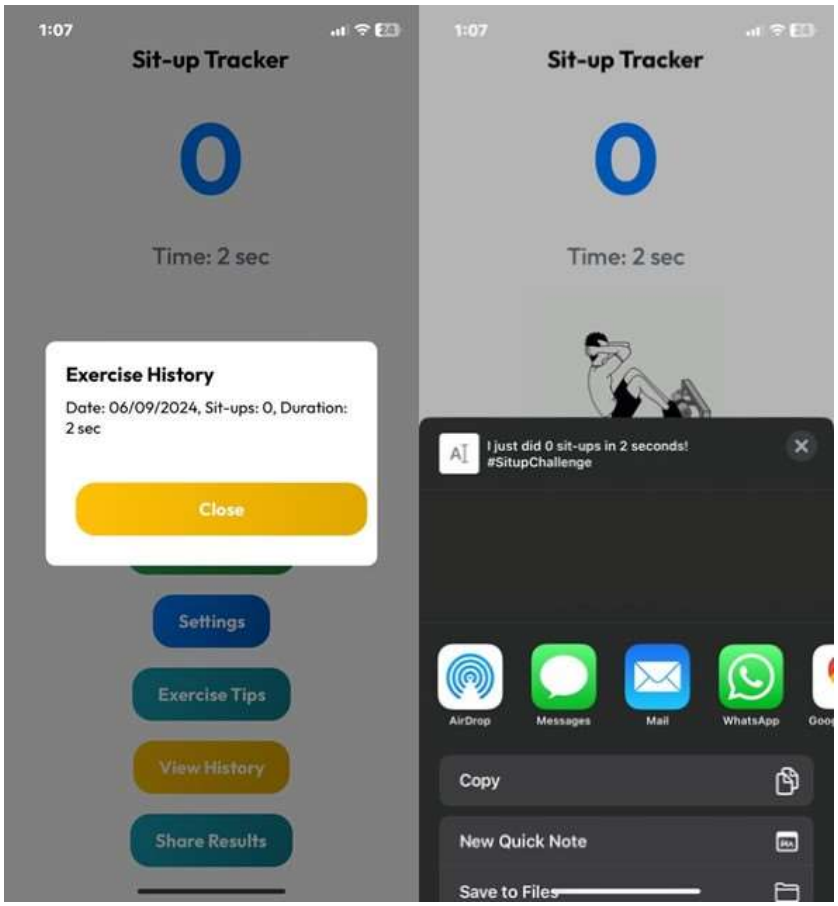


Figure 5.2.17 Sit Up Tracker screen (Exercise History & Share Exercise)

15) Gym Nearby Page

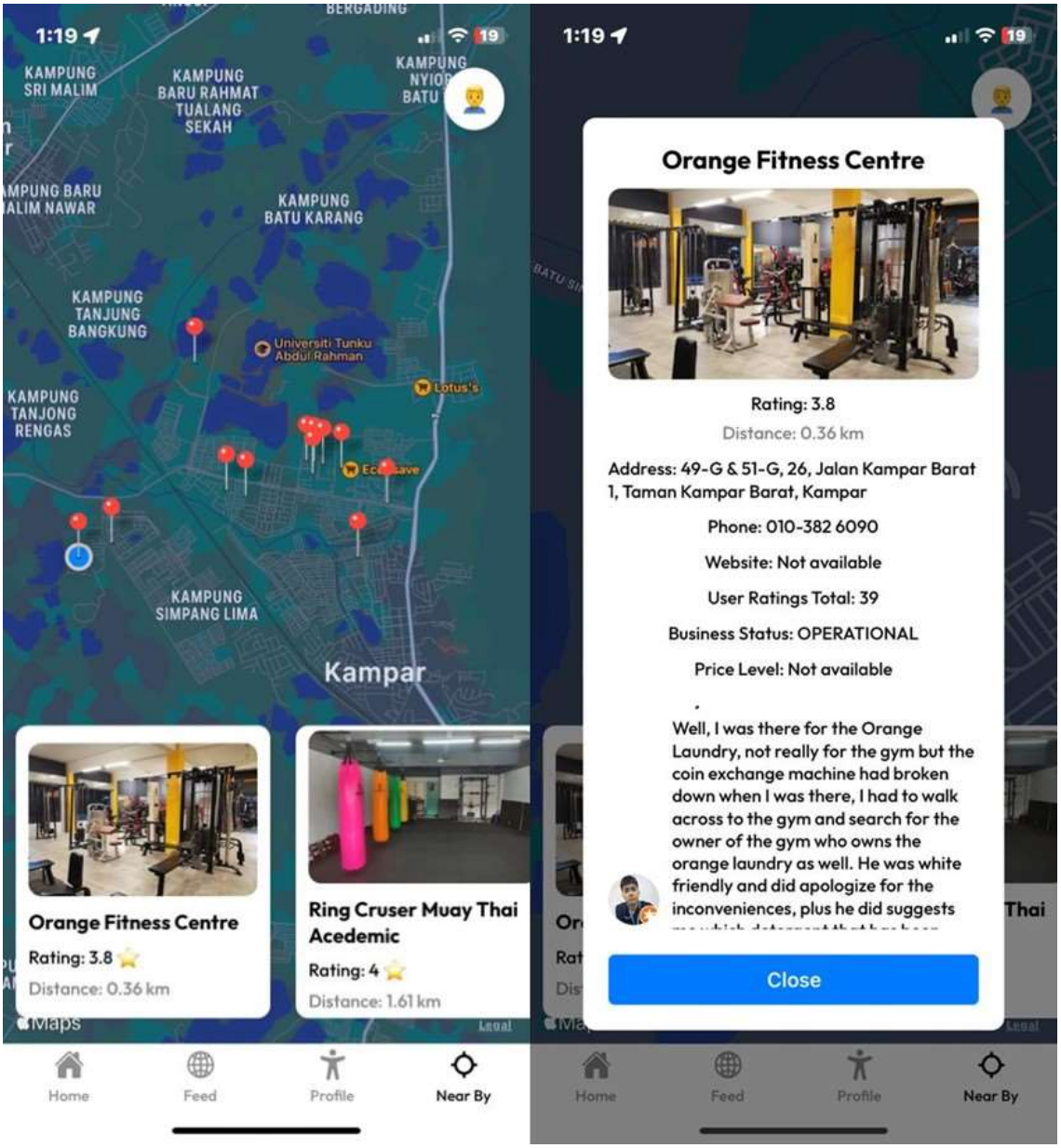


Figure 5.2.18 Gym Nearby screen (with Google Review)

5.3 Implementation Issues and Challenges

Integrating Google Gemini AI

Incorporating Google Gemini AI for tailored training routines requires fine-tuning parameters to provide relevant and exact suggestions. Adjusting the API replies to varied inputs, such as location and fitness objectives, became tough owing to the AI's complexity.

- **Response Time**: Delivering AI-generated exercise regimens rapidly was a challenge, particularly on slower networks, since delays might degrade user experience.

Managing Firebase Data

Managing Firebase Data Organizing exercise programs, gym information, and user progress in Firebase proved tough, especially with complicated nested arrays and maps. Efficiently handling the storage and retrieval of training programs for various users provided an extra degree of complexity.

- **Real-time Synchronization**: Keeping data in sync across devices in real time while decreasing latency concerns proved difficult, particularly when dealing with variable network circumstances.

Sit-up Tracker Using Expo Accelerometer

- **Sit-up Tracker with Expo Accelerometer Accuracy Issues**: Capturing precise sit-up motions with the accelerometer proved problematic, since tiny body movements might either erroneously register or miss the exercise entirely. various users need various accelerometer sensitivity settings.

- **Battery Optimization**: Continuous use of the accelerometer increased battery usage. Balancing performance and tracking precision without depleting battery power became a significant problem.

Google Places API for Nearby Gyms

- **API Rate Limits**: The Google Places API's limits on the amount of API queries created some issues while retrieving gym information for several members. To overcome this, caching and API call optimization were used.

- **Location Precision**: It was difficult to get precise location data, especially in less populated regions. This often resulted in erroneous suggestions for local gyms.

Integrating Social Media

- **Social Media Integration and Compatibility Issues**: Integrating social media sharing across platforms such as Facebook and Instagram posed compatibility issues owing to various API needs. Handling these platform-specific changes was important to guarantee that exercise data and feed articles could be shared smoothly.

- **Data Security**: Protecting user data when connecting with external social media platforms raised privacy issues, particularly over compliance with data protection rules.

User Interface and Experience (UI/UX)

- **Complex Interface Desig**n: Developing an elegant user interface that effortlessly integrated all of the app's functions (including exercise plan creation, the sit-up tracker, and the feed page) while remaining simple proved tricky. The combination of utility and usability necessitated repeated design revisions based on user input.

- **Device Compatibility**: Ensuring consistent app performance across several devices, operating systems, and screen sizes complicated the design and development process.

5.4 Concluding Remark

The AHLI: Workout Management Mobile App needed the integration of numerous technologies as well as the resolution of a number of obstacles. Firebase was used for data management, Google Gemini AI for developing tailored fitness programs, and the Google Places API for suggesting local gyms. These technologies were merged to provide a variety of features aimed at improving the user's fitness journey. API integration, real-time data synchronization, and reliable accelerometer readings were all significant challenges. However, continuing revisions allowed the app to meet the project's functional goals. This chapter went over the technical stages and challenges encountered, providing an overview of how the basic functions were built and enhanced.

Chapter 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

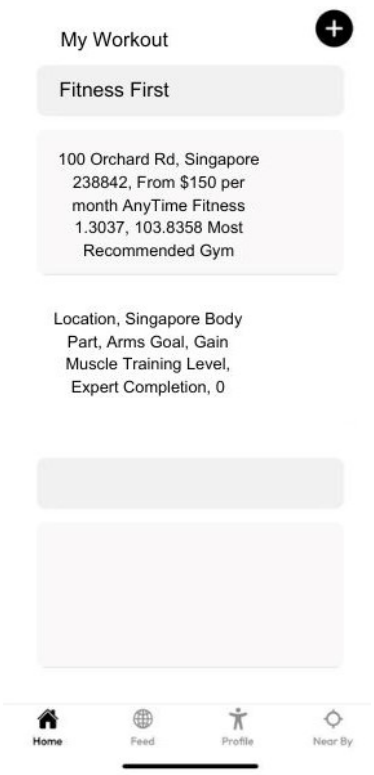


Figure 6.1.1 Home Screen (Before)

- No color
- Normal text
- Messy alignment

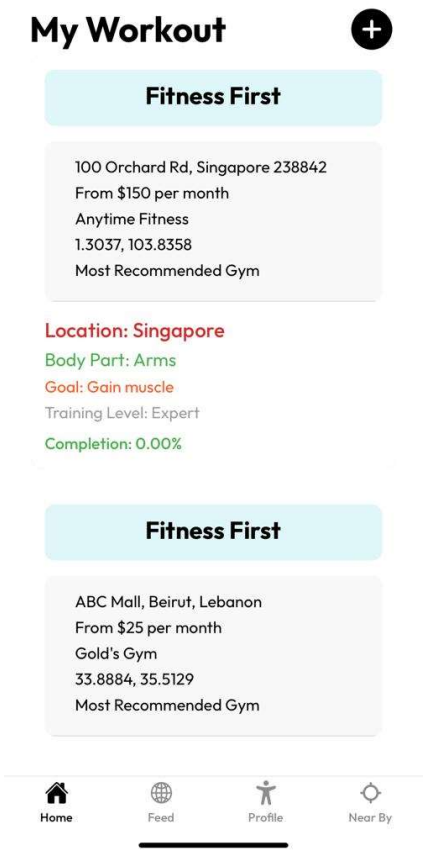


Figure 6.1.2 Home Screen (After)

- Correct format
- Added color
- Text aligns

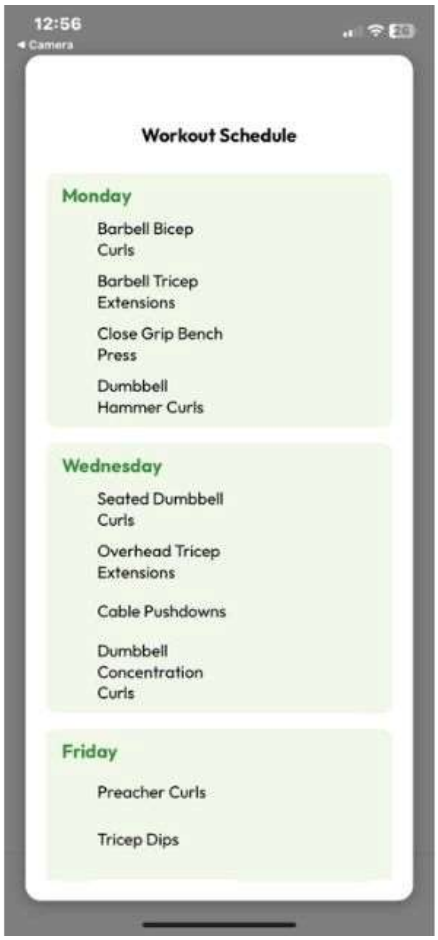


Figure 6.1.3 Workout Schedule (Before)

- No progress tracking
- No sets, reps, and rest time

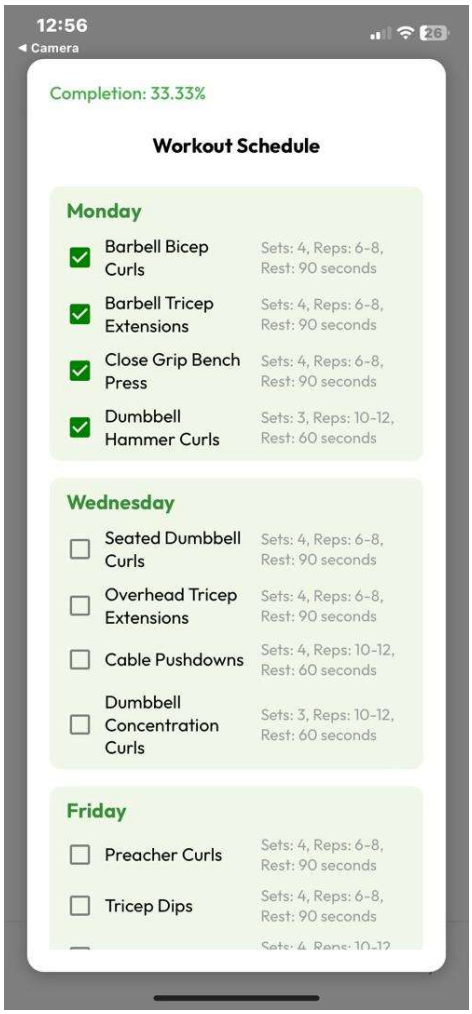


Figure 6.1.4 Workout Schedule (After)

- Added checkbox for progress tracking (completion percentage)
- Added sets, reps, and rest time

Figure 6.1.5 Discover page (Before)

- Cannot add image
- No post time and date
- After post, no notification

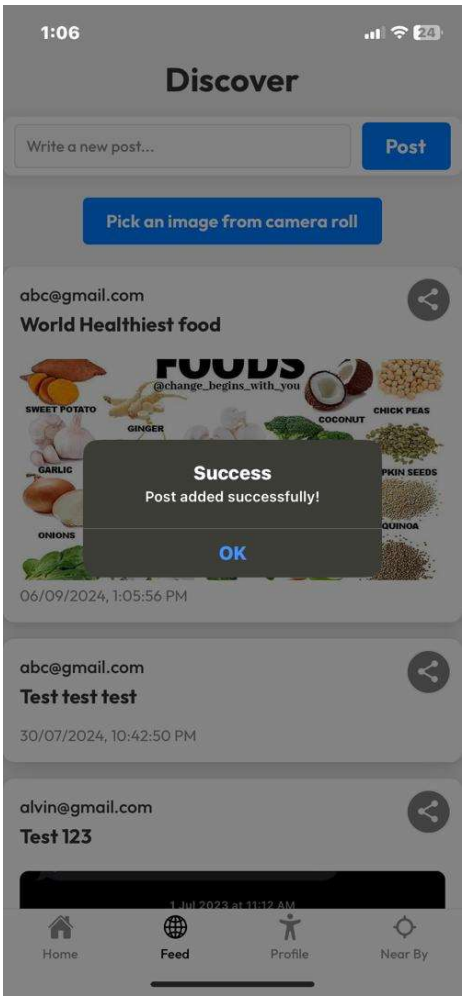


Figure 6.1.6 Discover page (After)

- Notify user when they successfully posted
- Added posted date and time
- Added add image to post

Figure 6.1.7 Sit Up Tracker page (Before)

- No color
- No sit up animation
- No alert of user how many sit up done
- No settings, exercise tips, sit up history, and share results

Figure 6.1.8 Sit Up Tracker page (After)

- Added color
- Added animation
- Added alert on how many sit up done
- Added settings, exercise tips, sit up history, and share results

Figure 6.1.9 Nearby Gym page (Before)

- No locate user current location button
- No swipe card of nearby gym
- No gym Google review

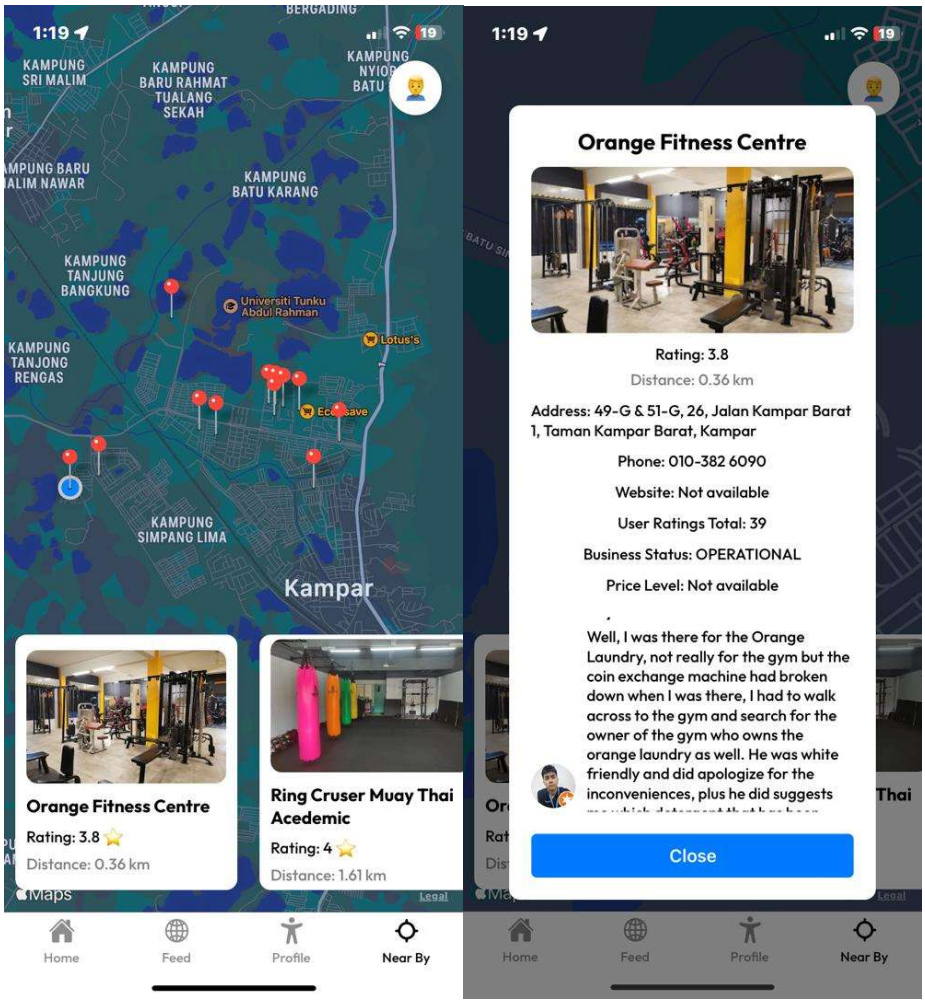


Figure 6.1.10 Nearby Gym page (After)

- Added locate user current location button
- Added swipe card of nearby gym
- Added gym Google review and gym details

6.2 Testing Setup and Result

To assess the AHLI: Workout Management App, a user feedback survey was completed with results from 24 users. The poll aims to obtain information on user happiness, functionality, and overall app performance. The following is an overview of the testing setup and major conclusions from the survey data.

Testing setup:

- 24 users participated.
- Age Range: The majority (87.5%) are between the ages of 18 and 24
- Gender: 54.2% women and 45.8% men
- Users were surveyed on their use patterns, user experience, and app functioning.
- Key testing areas: Some of the features evaluated were personalized training program development, progress monitoring, a sit-up tracker, gym suggestions, and app navigation.

Results:

- User Experience:

54.2% of users evaluated the overall user experience as "Good" or "Excellent," suggesting a favorable response to the software.
Users found it simple to create individual training programs, with 66.7% rating the tool as "Very Easy" or "Easy."

- Progress Tracking:

79.1% of users rated the progress monitoring option as "Very Useful" or "Useful," demonstrating the function's efficacy in helping users monitor their exercises.
Gym Recommendation:

66.7% of customers evaluated the gym suggestion function as "Good" or "Excellent," while 83.3% were "Satisfied" or "Very Satisfied" with the gym's details and price.
Social Media Sharing and Feed Page:

Social media sharing had mixed feedback, with 70.8% rating it as "Effective" or "Very Effective," while 29.2% said that they seldom or never utilized the feed page to submit photos or text about their exercise results.
Sit-Up Tracker's Performance:

70.8% of users said the sit-up tracker worked "well" or "very well," and 83.3% thought the tracker's workout advice were useful or very helpful.
App Navigation:

62.5% of users rated it "Easy" or "Very Easy" to use the app, indicating a user-friendly interface.

- Visual Design and Layout:

87.5% of users evaluated the visual design as "Good" or "Excellent," suggesting that the app's aesthetics were highly welcomed.
Overall Satisfaction and Likely to Recommend:

91.6% of users said they were "likely" or "very likely" to keep using the app in the future.
58.3% of users evaluated the app's overall value in comparison to comparable applications as "good," while 29.2% ranked it as "excellent."

6.3 Project Challenges

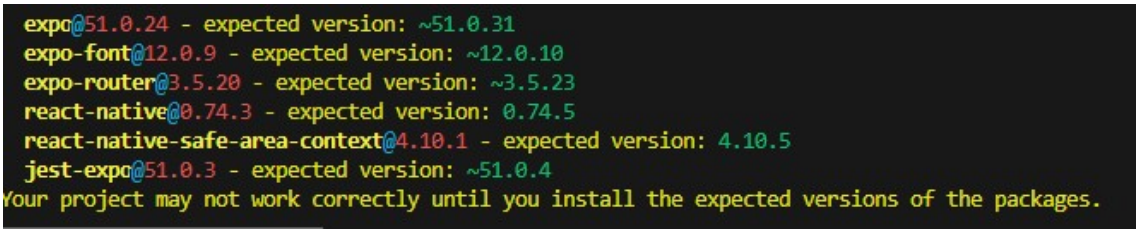


Figure 6.3.1 Updating Versions of Expo and React Native Packages

Challenge: Compatibility Issues.

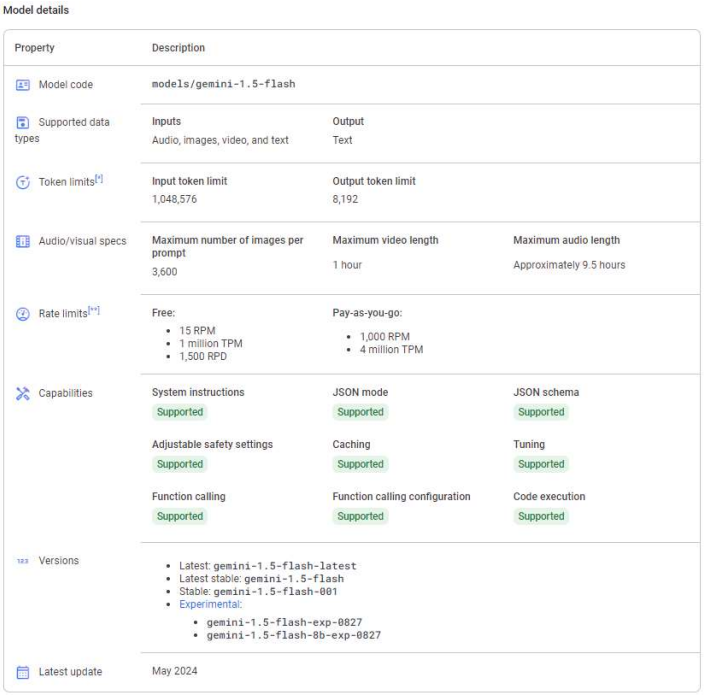
- **Breaking Changes:** Each new version of Expo and React Native often featured breaking changes, which made old code incompatible with the latest version. These modifications might include updated APIs, deprecated methods, or default behavior changes. Adapting the codebase to manage these breaking changes required a thorough examination and considerable rewriting of existing code, which impacted development timeframes and added complexity [9].

- **Deprecated Features:** New package versions routinely deprecate or delete previously used features. This required identifying and replacing outdated capabilities with newer replacements. For example, a frequently used component or function may be deleted, forcing developers to discover alternate solutions or rewrite portions of the project to meet the new framework's requirements.

- **Complex Dependencies:** As new versions of Expo and React Native were launched, handling dependencies got more complicated. Each update may bring new dependencies or demand modifications to existing ones, resulting in conflicts between various packages or versions. This proved especially difficult when integrating third-party libraries or plugins, which may not instantly support the most recent package versions.
- **Build Failures:** Dependency conflicts may cause build failures, in which the program fails to compile or execute properly owing to mismatched versions of libraries or packages. Resolving these conflicts sometimes required upgrading or downgrading

package versions, which might further complicate the dependency tree and need careful management to prevent creating new problems.

API Rate Limits for Google Gemini AI & Google Places API



Model details

Property	Description		
Model code	`models/gemini-1.5-flash`		
Supported data types	**Inputs** Audio, images, video, and text	**Output** Text	
Token limits[*]	**Input token limit** 1,048,576	**Output token limit** 8,192	
Audio/visual specs	**Maximum number of images per prompt** 3,600	**Maximum video length** 1 hour	**Maximum audio length** Approximately 9.5 hours
Rate limits[**]	**Free:** • 15 RPM • 1 million TPM • 1,500 RPD	**Pay-as-you-go:** • 1,000 RPM • 4 million TPM	
Capabilities	**System instructions** Supported	**JSON mode** Supported	**JSON schema** Supported
	Adjustable safety settings Supported	**Caching** Supported	**Tuning** Supported
	Function calling Supported	**Function calling configuration** Supported	**Code execution** Supported

Versions	• Latest: `gemini-1.5-flash-latest`
	• Latest stable: `gemini-1.5-flash`
	• Stable: `gemini-1.5-flash-001`
	• Experimental:
	  • `gemini-1.5-flash-exp-0827`
	  • `gemini-1.5-flash-8b-exp-0827`		

Latest update	May 2024		

Figure 6.3.2 Google Gemini API 1.5 Flash model

The Google Gemini AI API set rate limitations on the number of queries that may be made in a given time period. These constraints made it difficult to distribute AI-generated exercise regimens to consumers in real time or in batches. When rate restrictions were reached, it might cause delays in creating exercise programs or, in certain circumstances, inability to fulfill requests, resulting in a bad user experience [10].

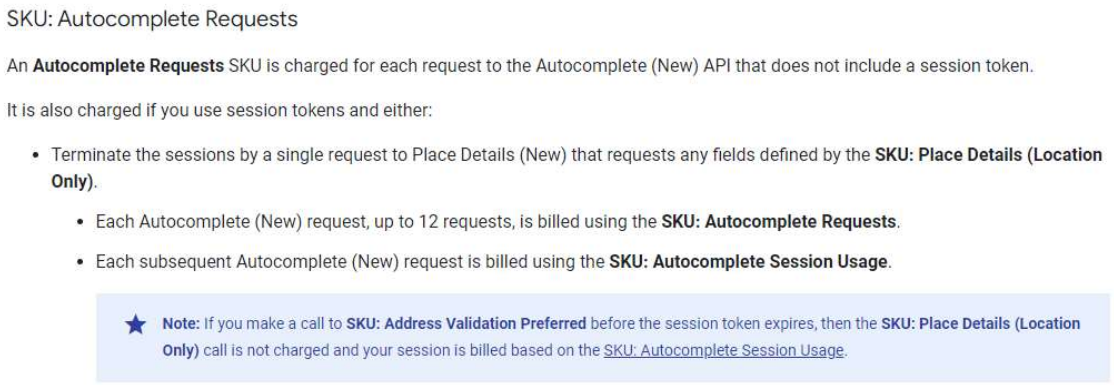


Figure 6.3.3 Google Places API model

Similarly, the Google Places API also had rate limits and usage quotas that restricted the number of API requests that could be made. For a feature like locating nearby gyms, which could involve frequent and high-volume API calls, hitting these limits could lead to delays or reduced functionality. This was particularly problematic when multiple users simultaneously requested gym information, potentially exceeding the allowed number of queries.

6.4 Concluding remark

The AHLI: training Management Mobile App effectively met its primary objectives of offering tailored training regimens, progress monitoring, and gym suggestions. Users expressed great satisfaction with the app's use, design, and functionality. Key issues, such as keeping up with Expo and React Native changes and handling API rate constraints for Google Gemini AI and Google Places, were overcome with diligent reworking.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

The AHLI: Workout Management Mobile App has effectively reached its goals, providing customers with a complete fitness management solution. Key features such as AI-generated tailored training regimens, gym finding using the Google Places API, and progress tracking have all been successfully incorporated. Users are very satisfied with the app's simplicity of use, graphic design, and ability to help them remain on track with their fitness objectives. API rate restrictions and compatibility concerns with Expo and React Native packages were resolved, resulting in a pleasant user experience. Overall, the research highlights the power of merging AI and cloud technologies to improve fitness management via a user-friendly smartphone application.

7.2 Recommendation

Future enhancements might include increasing the social feed to boost user engagement and community involvement. Furthermore, optimizing API queries and enhancing compatibility with newer versions of Expo and React Native will boost speed and stability. Implementing additional features, such as a broader range of fitness monitoring choices and more interactive gym finding activities, might improve the user experience. Regular updates based on user input will ensure that the app remains current and meets changing user demands.

References

Article:

Journal article:

[1] Z. Z. M. C. Y. L. Xinlei Chen, "Large-Scale mobile fitness app usage analysis for smart health," *IEEE Communications Magazine,* vol. 56, no. 4, pp. 46-52, 2018.

[2] A. D. Sole, Visual Studio Code distilled.

[3] C. Khawas, "Application of Firebase in Android App Development-A Study," *International Journal of Computer Applications,* vol. 179, no. 46, pp. 49-53.

[4] V. Aydoğdu, "MOBILE SPORTS EXERCISE APPLICATIONS IN THE CONTEXT OF PERSUASION TECHNOLOGIES," *International Journal of Education Technology and Scientific Researches,* vol. 8, no. 23.

[5] R. F. E. K. F. H. Michelle Weech, pp. 93-115.

[6] "JEFIT - your ultimate fitness app for workout tracking & progress monitoring," [Online]. Available: https://www.jefit.com/.

[7] S. J. Slobodan Milanko, "LiftRight: quantifying strength training performance using a wearable sensor," *Smart Health.*

[8] A. Omonije, "Agile Methodology: A comprehensive impact on modern business operations," *International Journal of Science and Research (IJSR),* vol. 13, no. 2, pp. 132-138.

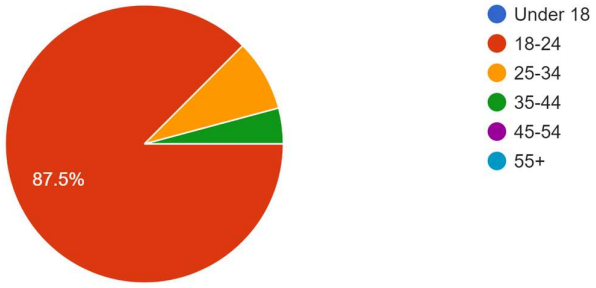
[9] M. M. Khan, "Medium," [Online]. Available: https://medium.com/@mustafakhan0260/problems-with-react-native-db3a5fddb3b#:~:text=The%20Bridge%20Problem&text=If%20the%20versions%20of%20React,between%20JavaScript%20and%20native%20code..

[10] [Online]. Available: https://ai.google.dev/gemini-api/docs/quota.

APPENDIX

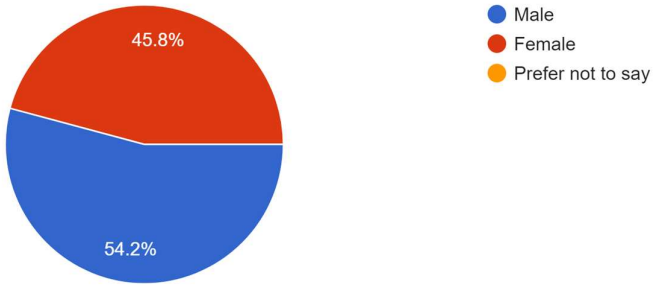
What is your age?

24 responses



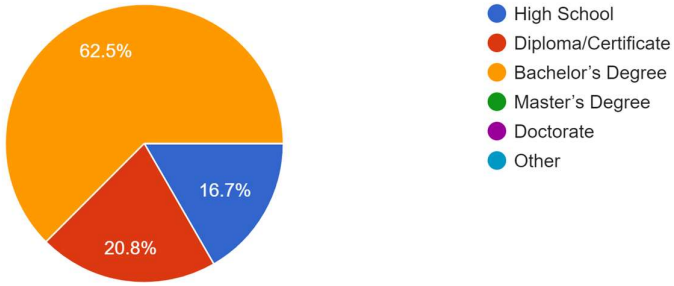
What is your gender?

24 responses



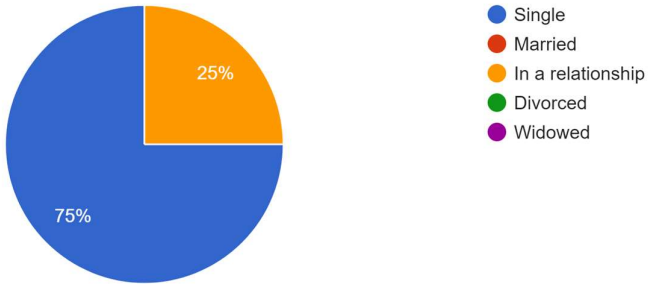
What is the highest level of education you have completed?

24 responses



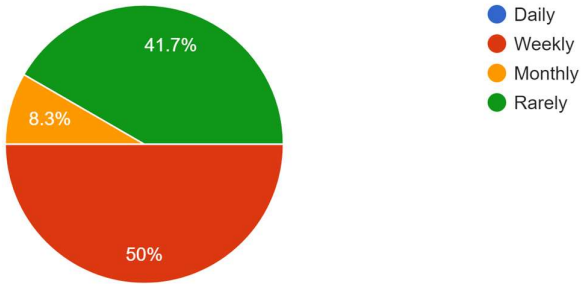
What is your marital status?

24 responses



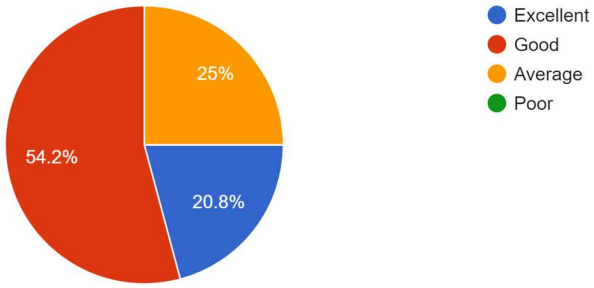
1. How frequently do you use the workout app?

24 responses



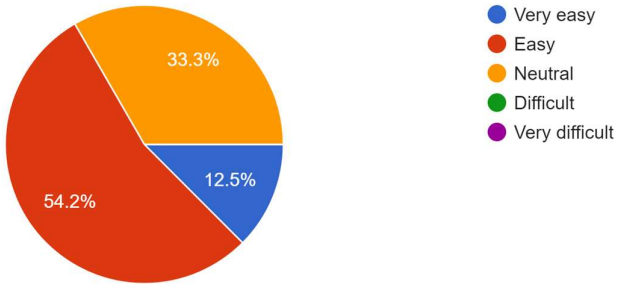
2. How would you rate the overall user experience of the app?

24 responses



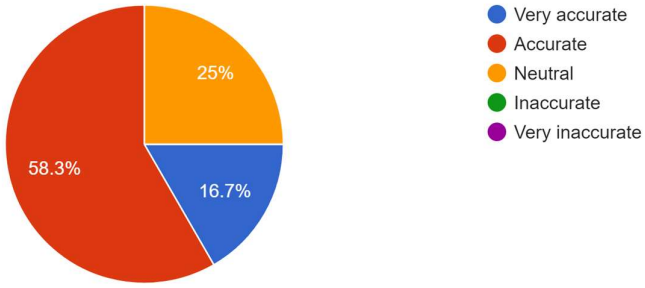
3. How easy is it to generate a personalized workout schedule using the app?

24 responses



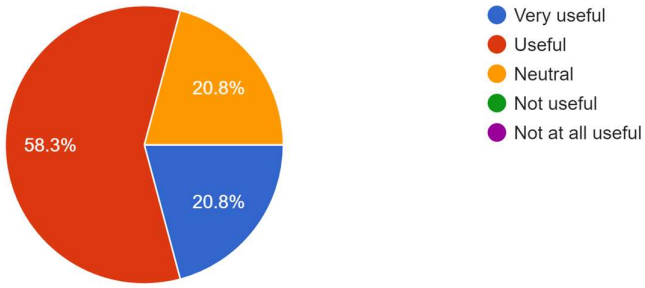
4. How accurate do you find the workout schedules generated by the app?

24 responses



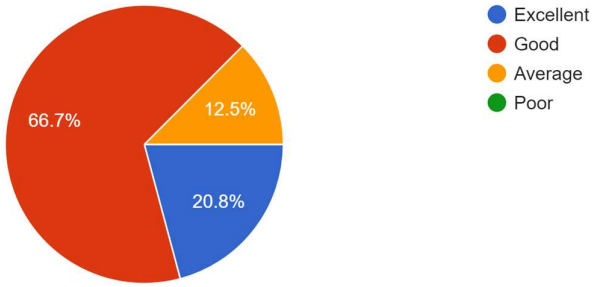
5. How useful do you find the progress tracking feature in the workout schedule?

24 responses



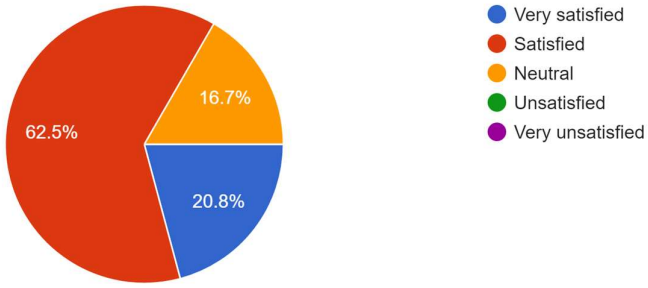
6. How would you rate the gym recommendations provided by the app?

24 responses



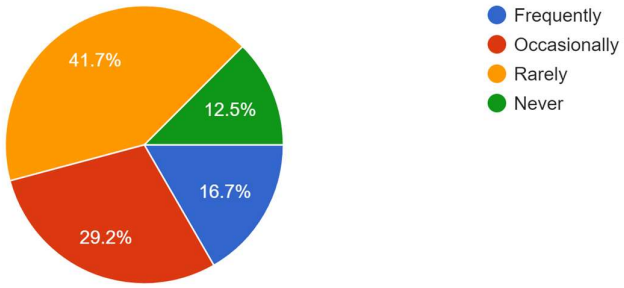
7. How satisfied are you with the gym details and pricing information provided?

24 responses



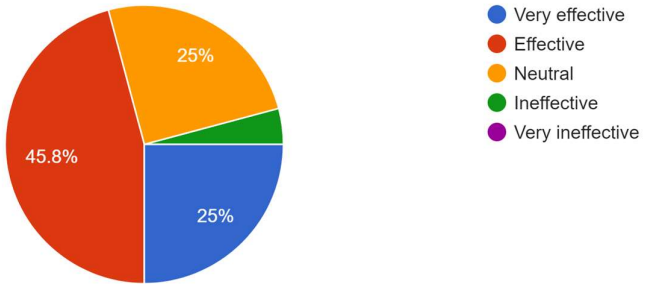
8. How often do you use the feed page to post images or text about your workout data?

24 responses



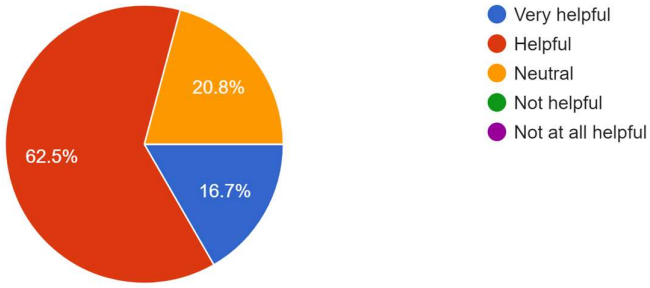
9. How effective is the social media sharing feature for your needs?

24 responses



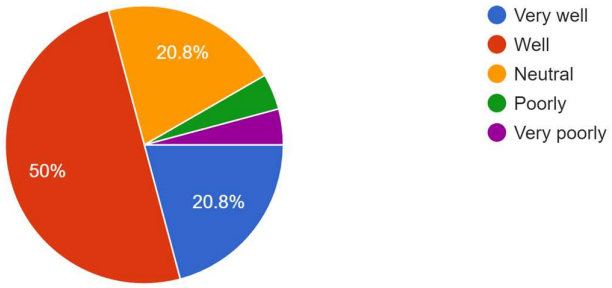
10. How helpful are the exercise tips provided in the sit-up tracker?

24 responses



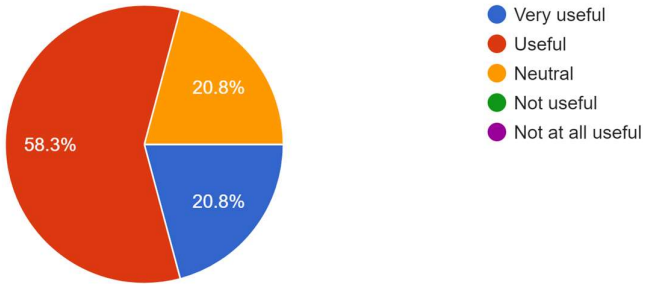
11. How well does the sit-up tracker perform in tracking your sit-ups?

24 responses



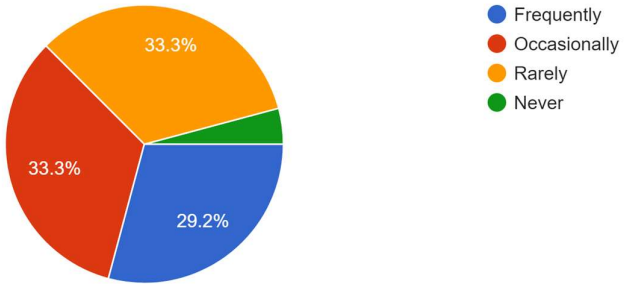
12. How useful is the sit-up history feature for tracking your progress?

24 responses



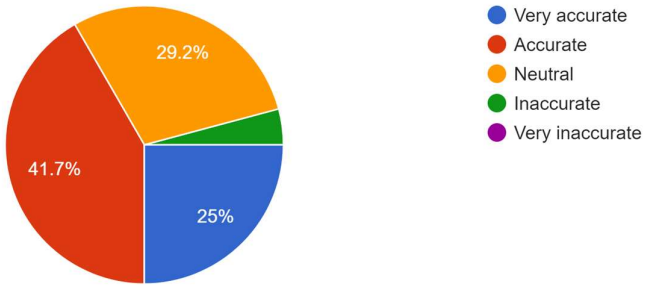
13. How frequently do you use the nearby feature to find gyms within 5km?

24 responses



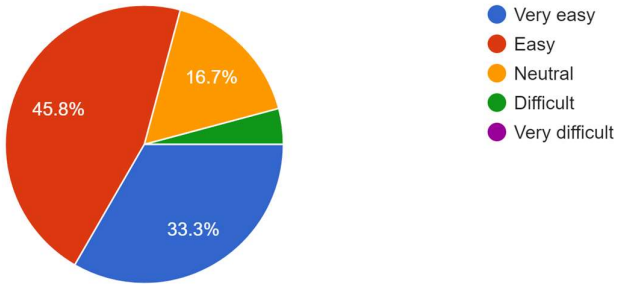
14. How accurate are the gym details and reviews shown in the nearby feature?

24 responses



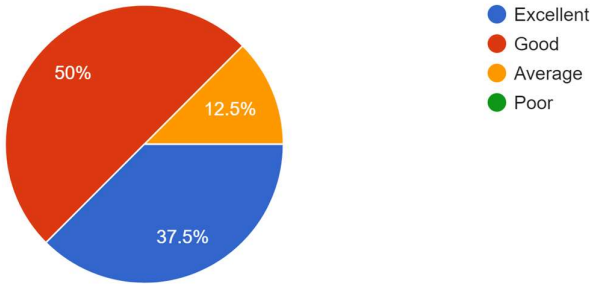
15. How easy is it to navigate through the app's various features?

24 responses



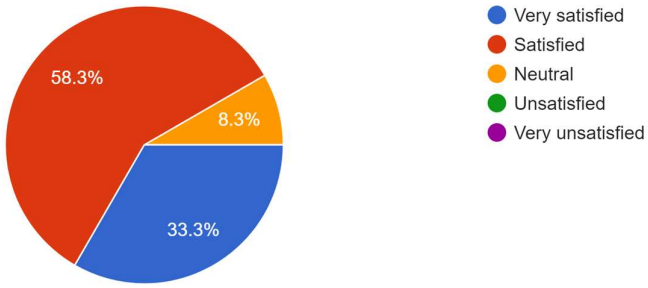
16. How would you rate the visual design and layout of the app?

24 responses



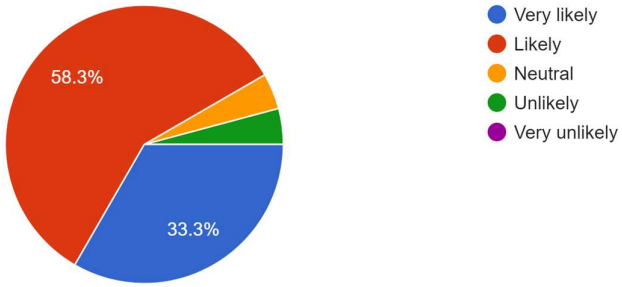
17. How satisfied are you with the app's performance and speed?

24 responses



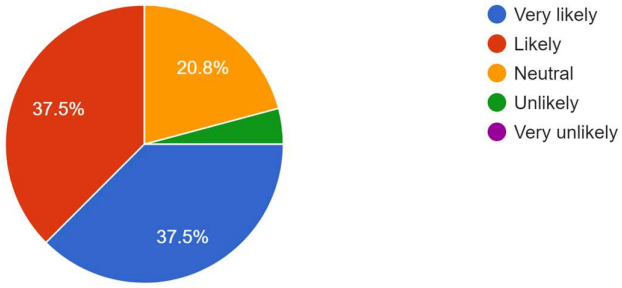
18. How likely are you to continue using the app in the future?

24 responses



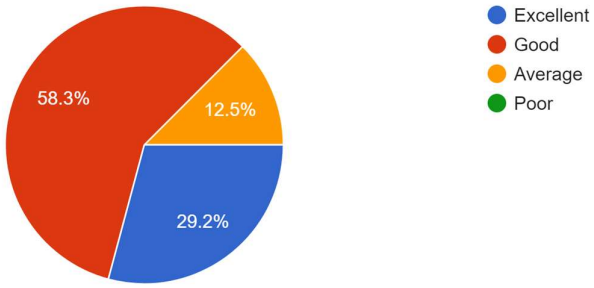
19. How likely are you to recommend the app to friends or family?

24 responses



20. How would you rate the overall value of the app compared to similar apps?

24 responses



FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	**Study week no.: 6**
Student Name & ID: ALVIN TAM YUN JIE , 20ACB052025	
Supervisor: Ts Lim Jit Theam	
Project Title: AHLI: Workout Management Mobile App	

1. WORK DONE

Authentication, user interface, setup Firebase.

2. WORK TO BE DONE

Select and test better AI Modal, prompt.
Add in progress tracker, checkbox.
Utilize feed page.

3. PROBLEMS ENCOUNTERED

Billing issue on API.
SDK updates, compatibility issues.

4. SELF EVALUATION OF THE PROGRESS

Iterative testing to make sure it works.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 2, 3	**Study week no.: 8**
Student Name & ID: ALVIN TAM YUN JIE , 20ACB052025	
Supervisor: Ts Lim Jit Theam	
Project Title: AHLI: Workout Management Mobile App	

1. WORK DONE

Confirm Google's Gemini API.
Done for the user location and nearby gym page.
Added animation for the tracker page, added progress percentage.

2. WORK TO BE DONE

Improve on the accelerometer on sit up logic.

3. PROBLEMS ENCOUNTERED

Devices crash sometimes, when testing on the sit up movement.

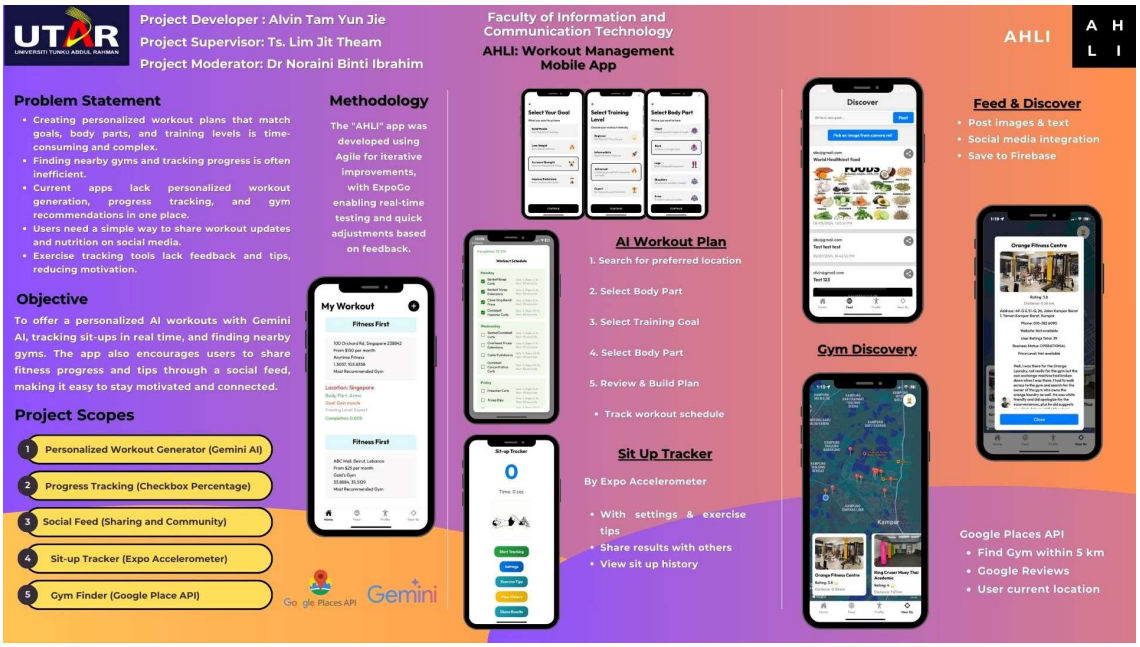
4. SELF EVALUATION OF THE PROGRESS

Project able to done on time.

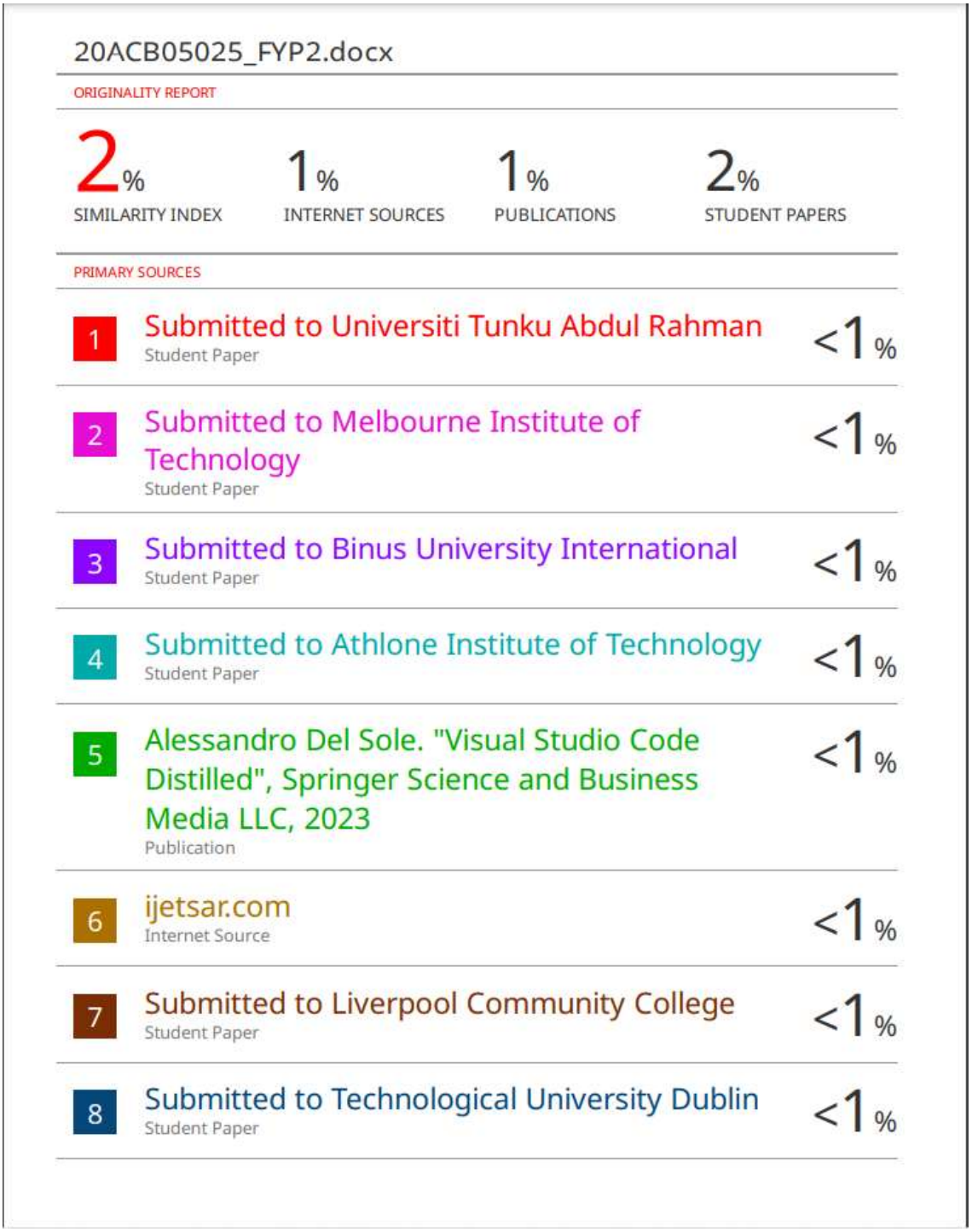
Supervisor's signature

Student's signature

POSTER


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Project Developer : Alvin Tam Yun Jie
Project Supervisor: Ts. Lim Jit Theam
Project Moderator: Dr Noraini Binti Ibrahim
Faculty of Information and Communication Technology
AHLI: Workout Management Mobile App
AHLI
Problem Statement
Creating personalized workout plans that match goals, body parts, and training levels is time-consuming and complex.
Finding nearby gyms and tracking progress is often inefficient.
Current apps lack personalized workout generation, progress tracking, and gym recommendations in one place.
Users need a simple way to share workout updates and nutrition on social media.
Exercise tracking tools lack feedback and tips, reducing motivation.
Objective
To offer a personalized AI workouts with Gemini AI, tracking sit-ups in real time, and finding nearby gyms. The app also encourages users to share fitness progress and tips through a social feed, making it easy to stay motivated and connected.
Project Scopes
1 Personalized Workout Generator (Gemini AI)
2 Progress Tracking (Checkbox Percentage)
3 Social Feed (Sharing and Community)
4 Sit-up Tracker (Expo Accelerometer)
5 Gym Finder (Google Place API)
Methodology
The "AHLI" app was developed using Agile for iterative improvements, with ExpoGo enabling real-time testing and quick adjustments based on feedback.
My Workout
Fitness First
Google Places API
Gemini
AI Workout Plan
1. Search for preferred location
2. Select Body Part
3. Select Training Goal
4. Select Body Part
5. Review & Build Plan
Track workout schedule
Sit Up Tracker
By Expo Accelerometer
With settings & exercise tips
Share results with others
View sit up history
Discover
Gym Discovery
Feed & Discover
Post images & text
Social media integration
Save to Firebase
Google Places API
Find Gym within 5 km
Google Reviews
User current location

PLAGIARISM CHECK RESULT



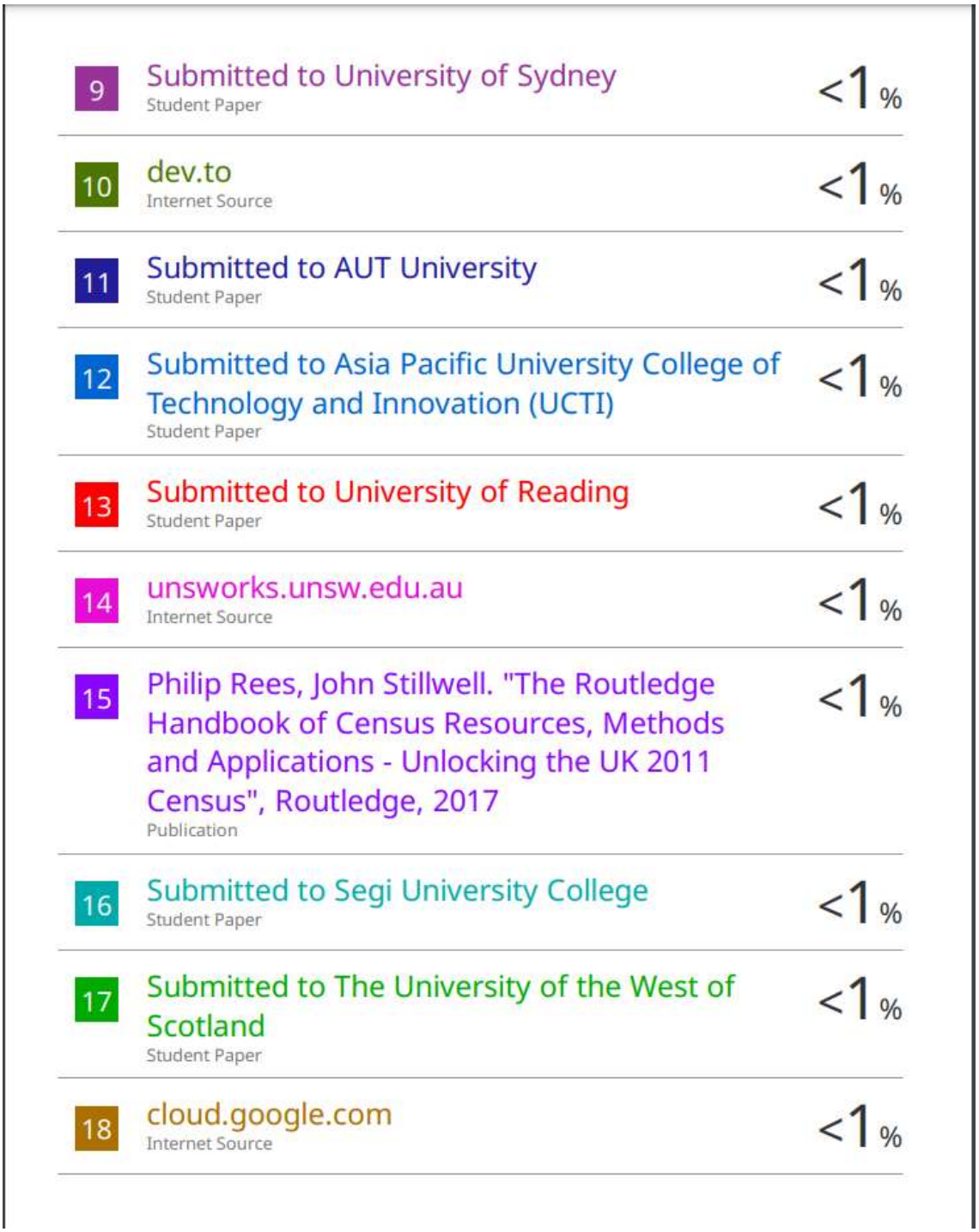
20ACB05025_FYP2.docx

ORIGINALITY REPORT

2%	1%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universiti Tunku Abdul Rahman Student Paper	<1%
2	Submitted to Melbourne Institute of Technology Student Paper	<1%
3	Submitted to Binus University International Student Paper	<1%
4	Submitted to Athlone Institute of Technology Student Paper	<1%
5	Alessandro Del Sole. "Visual Studio Code Distilled", Springer Science and Business Media LLC, 2023 Publication	<1%
6	ijetsar.com Internet Source	<1%
7	Submitted to Liverpool Community College Student Paper	<1%
8	Submitted to Technological University Dublin Student Paper	<1%


9 Submitted to University of Sydney
Student Paper
<1%
10 dev.to
Internet Source
<1%
11 Submitted to AUT University
Student Paper
<1%
12 Submitted to Asia Pacific University College of Technology and Innovation (UCTI)
Student Paper
<1%
13 Submitted to University of Reading
Student Paper
<1%
14 unsworks.unsw.edu.au
Internet Source
<1%
15 Philip Rees, John Stillwell. "The Routledge Handbook of Census Resources, Methods and Applications - Unlocking the UK 2011 Census", Routledge, 2017
Publication
<1%
16 Submitted to Segi University College
Student Paper
<1%
17 Submitted to The University of the West of Scotland
Student Paper
<1%
18 cloud.google.com
Internet Source
<1%

Universiti Tunku Abdul Rahman			
Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	ALVIN TAM YUN JIE
ID Number(s)	20ACB05025
Programme / Course	DE
Title of Final Year Project	AHLI: Workout Management Mobile App

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 2 % **Similarity by source** Internet Sources: 1 % Publications: 1 % Student Papers: 2 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Ts Lim Jit Theam

Date: 8/9/2024

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB05025
Student Name	ALVIN TAM YUN JIE
Supervisor Name	Ts Lim Jit Theam

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

_____ _____

(Signature of Student)
Date: