Design and Implementation of Workout Buddy Finder App

by

Md. Didarul Islam
ID: CSE1903018003

Md. Tipu Sultan
ID: CSE1903018062

Ahsanul Kabir
ID: CSE1903018070

Shobnom Mustarin
ID: CSE1903018083

Supervised by
Bulbul Ahamed

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)
May 2023

APPROVAL

The project titled "**Design and Implementation of Workout Buddy Finder App**" submitted by Md. Didarul Islam (CSE1903018003), Md. Tipu Sultan (CSE1903018062), Ahsanul Kabir (CSE1903018070) and Shobnom Mustarin (CSE1903018083) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Bulbul Ahamed — **Supervisor**
Associate Professor and Head,
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 1**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 3**
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Bulbul Ahamed, Associate Professor and Head,** Department of Computer Science and Engineering, Sonargaon University (SU), Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Bulbul Ahamed)
Supervisor

Signature

Md. Didarul Islam
ID: CSE1903018003

Md. Tipu Sultan
ID: CSE1903018062

Ahsanul Kabir
ID: CSE1903018070

Shobnom Mustarin
ID: CSE1903018083

ABSTRACT

This book presents the development and implementation of a mobile application desiThe app is designed to serve as a social platform that connects individuals with similar fitness interests and goals. Rather than simply providing a directory of nearby fitness groups, the app aims to encourage users to form their own groups based on their shared interests and fitness goals. By connecting individuals with others who share their interest to help people connect with nearby individuals with similar fitness interests and goals. The mobile application is developed by the team behind this book.

The app's features include the ability to provide suggestions about nearby users based on location, interests, and fitness goals, as well as the ability to communicate with other users through a chat function. Users can create their own profiles, which include information about their fitness interests and goals, and can suggest other users with similar profiles. By enabling users to connect with others who share their interests, the app provides a platform for users to form exercise groups and engage in physical activities together.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the scheduled time.

We are auspicious that we had the kind association as well as supervision of **Bulbul Ahamed**, Associate Professor and Head, Department of Computer Science and Engineering, Sonargaon University (SU) whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We would like to convey our special gratitude to **Brig. Gen. (Retd) Prof. Habibur Rahman Kamal, ndc, psc** Dean, Faculty of Science and Engineering for his kind concern and precious suggestions.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

APP	Application
CPU	Central Processing Unit
ERD	Entity Relationship Diagram
Gym	Gymnasium
IDE	Integrated Development Environment
iOS	iPhone Operating System
OS	Operating System
QA	Quality Assurance
RAM	Random Access Memory
ROM	Read Only Memory
SDK	Software Development Kit
SDLC	System Development Life Cycle
SQL	Structured Query Language
SU	Sonargaon University

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION TO WORKOUT BUDDY FINDER APP

1.1 Introduction

The global pandemic brought upon by COVID-19 has highlighted the importance of regular exercise and fitness training in our daily lives. With the increasing awareness about the importance of fitness, many people have started exercising, but soon lose their motivation and drop out due to a lack of social support.

To address this issue, the team got inspired to create a mobile application that would help people to connect with nearby individuals with similar fitness goals. The app was developed using Flutter, a popular mobile app development framework that allows for the creation of fast and beautiful applications for both Android and iOS platforms.

The app suggests other users nearby based on their fitness interests and goals. Users can then connect with each other, share their progress, and motivate each other to reach their goals. The app also features a chat function that allows users to communicate with each other and plan workouts or activities.

To build the backend for the app, the team utilized Firebase, a powerful platform for building mobile and web applications. The app's data is stored in Firestore, a NoSQL document database that allows for real-time data synchronization and scalable data storage.

The app has a feature that calculates the total energy burned by the user in the last 24 hours. This feature provides users with an accurate measurement of their daily physical activity levels and helps them stay on track with their fitness goals.

To calculate the total energy burned, the app uses data from the user's fitness tracker or smartphone sensors, such as the accelerometer and GPS. The app tracks the user's physical activity throughout the day, including their steps taken, distance traveled, and calories burned during exercise.

Through this project, the team hopes to provide a platform for people to connect with others who share their interests and fitness goals. By providing a social support system, the team believes that people will be more motivated to exercise regularly and lead a healthy lifestyle.

1.2 Objectives

The major objectives of the project are below:

The main objective of this project is to develop a workout buddy finder app that will help users find workout partners based on their interests, availability, gender, and locality. The app provides users with suggestions for potential workout buddies based on various factors, making it easier for them to connect with like-minded individuals and form exercise partnerships

Another objective of the app is to create an exciting and engaging experience for users while they work out. By helping exercisers find workout partners, the app will encourage users to stay motivated and committed to their fitness goals, making it easier for them to pass their workout time and achieve their desired results.

The project also aims to develop a high-quality, standard-level application that will be compatible with both Android and iOS devices. This will ensure that the app is accessible to a wide range of users, regardless of their preferred mobile platform. The app is designed to be user-friendly and intuitive, with a clean and modern interface that is easy to navigate. It is optimized for performance, with fast load times and minimal lag.

Overall, the objectives of the project are to help users find workout buddies, create an enjoyable workout experience, and develop a high-quality mobile app that meets the needs of users across different platforms. By achieving these objectives, the project aims to promote health and wellness among people by making it easier for them to exercise regularly and stay motivated towards their fitness goals.

1.3 Methodology

One of the most intriguing aspects of this project is its creativity. The team behind this project used a variety of different components and technologies in order to create a product that meets high standards of functionality and usability. This combination of technical elements, combined with innovative design and development techniques, made the task of building this application both challenging and engaging.

In addition to the satisfaction of creating a successful and functional product, the project offered team members an opportunity to learn and practice new skills. The project required extensive research and testing of different technologies and programming languages, providing team members with an opportunity to expand their knowledge and expertise.

Furthermore, in order to ensure that the application would be accessible to as many people as possible, the team decided to develop the application for both Android and iOS operating systems. By developing the application for these two platforms, the team was able to reach a wide range of smartphone users, including those who use both Android and Apple devices.

Overall, the project was a challenging yet rewarding experience that allowed the team to combine creativity, technical skills, and practical problem-solving in order to create a product that can benefit a wide range of users. The decision to develop the application for both Android and iOS devices demonstrates the team's commitment to creating an application that is accessible to as many people as possible.

Below Technologies are used to develop the application:

- **Flutter (for Android and iOS App):** Flutter is an open-source mobile application development framework created by Google. It is used to develop applications for Android and iOS platforms using a single codebase. Flutter offers a fast development cycle with its hot reload feature, which allows developers to see changes instantly without having to rebuild the entire app. It also provides a rich set of pre-built widgets that can be easily customized to create beautiful and responsive user interfaces.

- **Firebase Realtime Database (NoSQL):** Firebase Realtime Database is a NoSQL cloud-hosted database provided by Google. It is a scalable and flexible database that stores data in JSON format and synchronizes data in real-time between clients. This database is optimized for real-time updates and is capable of handling large amounts of data. It also provides a REST API that can be used to access data from web or mobile applications.

- **Google Authentication:** Google Authentication is a service provided by Google that allows users to authenticate with their Google account. It provides a secure and convenient way to sign in to web or mobile applications without having to create and remember a separate set of login credentials for each application. With Google Authentication, users can sign in with their Google account.

1.4 Project Description

The Workout Buddy Finder App is a mobile application designed to simplify the process of finding a workout partner for fitness enthusiasts. This app provides users with a comprehensive suggestion function that allows them to filter the results based on their interests and locality. This means that users can easily find a workout buddy who shares similar interests and lives nearby. The app offers a user-friendly interface that is easy to navigate and customize preferences according to user requirements.

The app is designed to suggest workout buddies based on the interests and hobbies that users input. This feature ensures that users are matched with workout buddies who share similar interests, creating a more productive and effective workout experience. The app also offers a social networking feature that enables users to connect with other fitness enthusiasts in their area. Users can share their workout experiences, tips, and progress with their workout buddies, creating a supportive and motivational community of like-minded individuals.

The app is available for Android and iOS users, built using the Flutter framework. The app uses Firebase Realtime Database as its backend, which provides a scalable and flexible NoSQL database solution for storing and syncing data in real-time. The app also features Google Authentication, which provides secure and easy-to-use authentication to help ensure user privacy.

Overall, the Workout Buddy Finder App is a convenient and effective tool for finding workout partners. With its comprehensive search function, customized matching algorithms, and social networking features, this app offers a unique and personalized workout experience for users, ultimately promoting a healthy and active lifestyle.

1.5 Outline

In the first chapter, the introduction, methodology, and a short summary about the whole project are given. The second chapter delves into the review of the project, analyzing the features and justifications behind the design choices. This includes exploring the user registration and login, user profile creation and customization, search and filter functions, matching algorithms, social networking features.

The third chapter provides an in-depth look at the design and implementation life cycle of the project. This includes a detailed analysis of the features, their implementation, and the tools used to build the system. Additionally, it discusses the challenges and limitations encountered during the development process.

The fourth chapter includes the system design documents used for this project, as well as the outputs of the project. This provides a comprehensive look at the architecture and components of the system, including the database schema, code snippets, and screenshots of the user interface.

Finally, in the fifth chapter, the conclusion summarizes the main findings and contributions, limitations and challenges encountered, and future plans for the project. This discusses potential improvements and updates that can be made to enhance the system's functionality and user experience.

1.6 Design Diagram

ERD

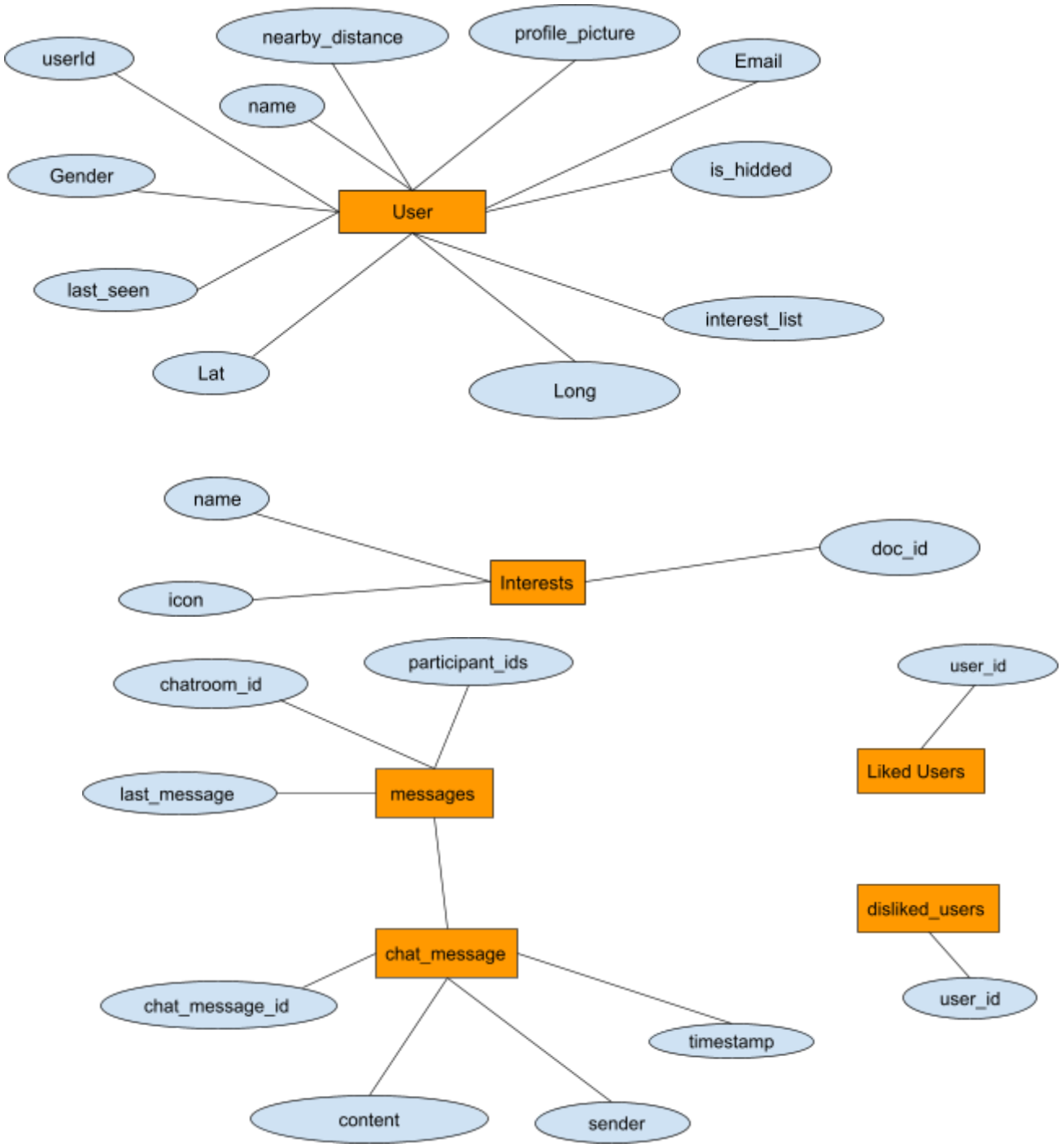


Figure 1.1: ERD

- **User collection**: This collection stores user data such as their name, email, location, preferences, and other relevant information. Each document in this collection represents a user and can be identified by a unique user ID.
- **Interests collection**: This collection contains documents that represent interests. Each document contains a name, icon, and doc_id field.
- **Liked users collection**: This collection stores the list of users a particular user has liked. Each document in this collection represents a like and can be identified by a unique ID.
- **Disliked users collection**: This collection stores the list of users a particular user has disliked. Each document in this collection represents a dislike and can be identified by a unique dislike ID.
- **Messages collection**: This collection stores the chat messages exchanged between users. It also has a sub-collection, chat_message, which stores individual chat messages sent and received by each user. Each document in the messages collection represents a conversation between two users and can be identified by a unique conversation ID.

Collections

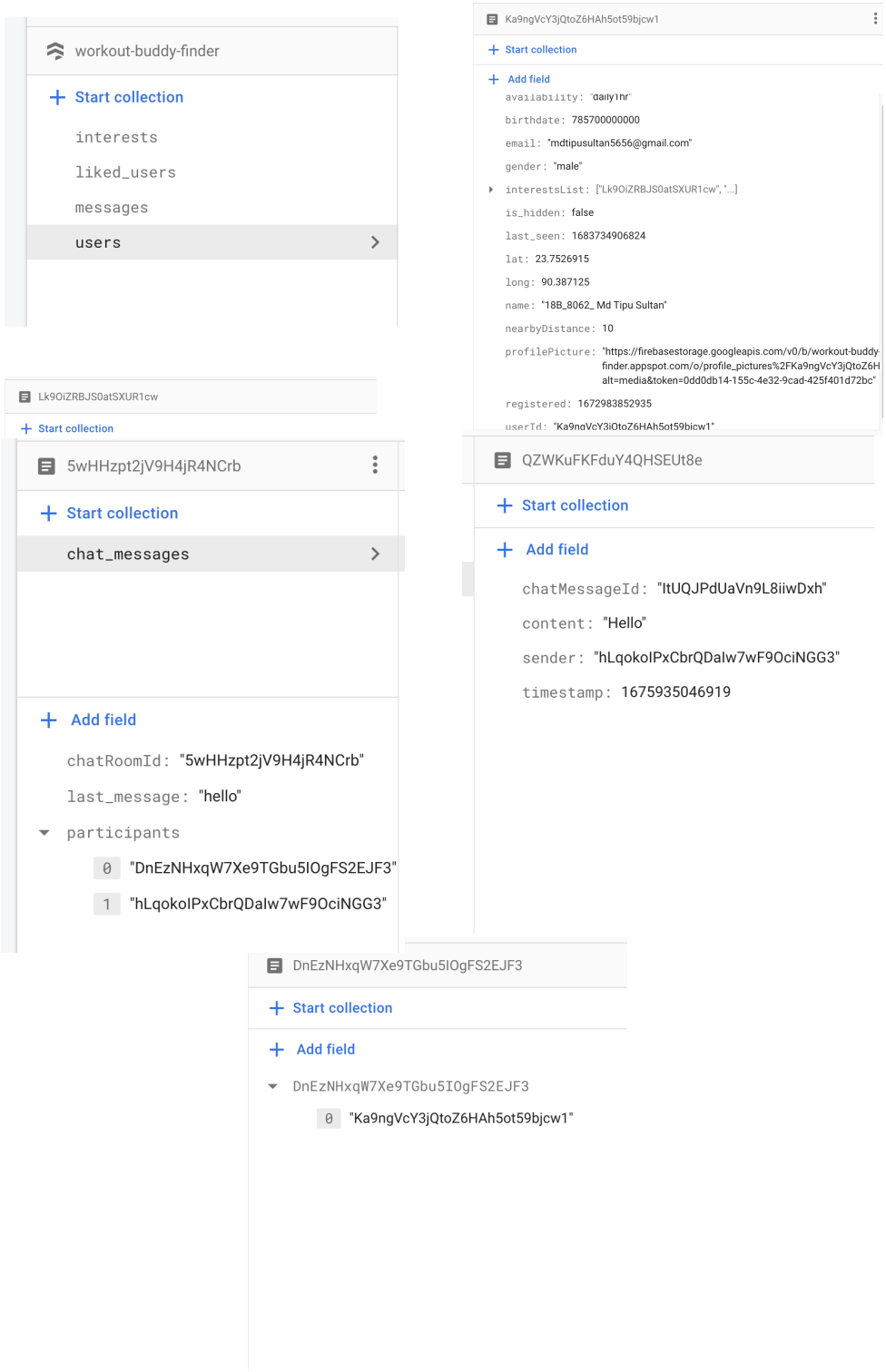


Figure 1.2: Collection Diagram

Firestore is a NoSQL database that is highly scalable and designed to handle large amounts of unstructured data. It is a part of the Firebase platform provided by Google, and it uses a flexible document-based data model that allows for easy organization and retrieval of data. Firestore also provides real-time data synchronization and offline data persistence, which makes it an ideal choice for mobile and web applications. Additionally, Firestore offers strong security and authentication features, as well as integration with other Google Cloud services.

Firestore collections are containers for documents that contain the data of the application. A collection is essentially a list of documents that have similar data or belong to the same category. Collections are created and organized based on the needs of the application. For example, a social media application may have collections for users, posts, comments, likes, and followers.

Each collection in Firestore can contain any number of documents, and each document can contain any number of fields. Documents in a collection are uniquely identified by a document ID. A document can be thought of as a key-value pair where each field has a name and a value. The value can be a primitive data type such as a string, number, or boolean, or it can be a complex data type such as an array or a map.

Firestore collections are designed to be scalable, meaning that they can handle large amounts of data and traffic. They provide real-time updates, which means that any changes made to the data in the collection are immediately reflected in the application without the need for the user to refresh the page.

Firestore collections can be accessed and modified using the Firebase SDKs or through the Firestore REST API. The Firestore SDKs provide a simple and intuitive interface for interacting with collections and documents, while the REST API allows for more advanced use cases and custom integrations.

Gantt Chart

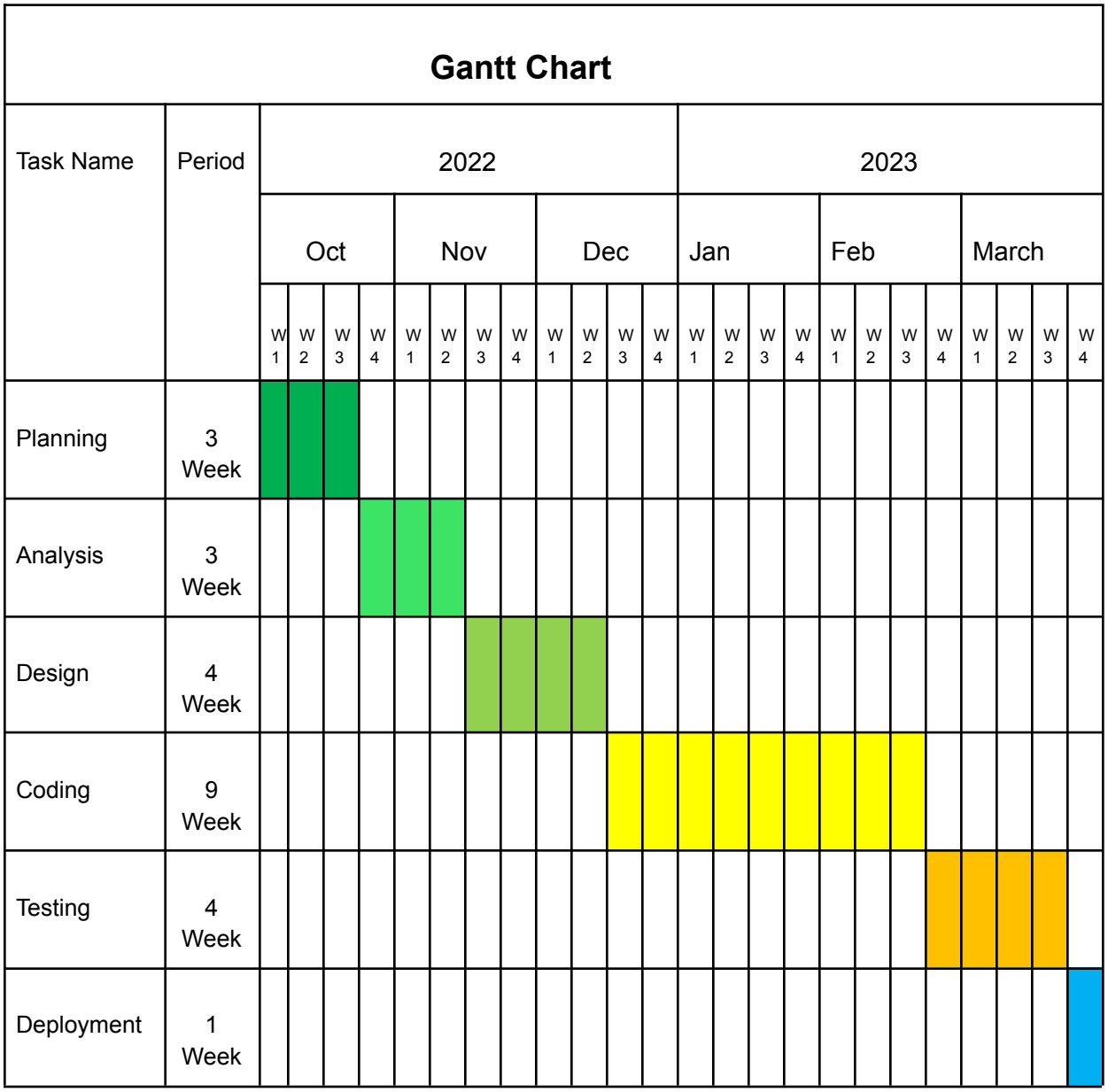


Figure 1.3: Gantt Chart

A Gantt chart is a type of bar chart that illustrates a project schedule. It is used to represent the timeline of a project, showing the start and end dates of various tasks or activities. The Gantt chart is a useful tool for project management as it allows project managers to visualize the progress of tasks and activities, monitor deadlines, and allocate resources efficiently.

In the context of the Workout Buddy Finder App project, the Gantt chart helps to illustrate the various tasks and activities required to complete the project. It shows the start and end dates of each task, as well as the dependencies between tasks, which helps project managers to plan and allocate resources efficiently.

For example, the Gantt chart for the Workout Buddy Finder App project may include tasks such as developing the user interface, integrating the Firebase Realtime Database, implementing Google authentication, and testing the app. The Gantt chart can help the project team to determine the critical path of the project, which is the sequence of tasks that must be completed on time to ensure that the project is completed within the specified timeframe.

In summary, the Gantt chart is a valuable tool for project management that helps project managers to plan and monitor project progress. It provides a visual representation of the project timeline, making it easier for project managers to allocate resources efficiently, monitor deadlines, and ensure that the project is completed on time.

CHAPTER 2
PROJECT OVERVIEW

2.1 Project Management

Project management is a critical aspect of any successful project development. It involves the application of knowledge, skills, tools, and techniques to manage project activities and meet project requirements.

Effective project management ensures that projects are completed within budget, on schedule, and to the desired quality standards. Project management focuses on achieving project objectives through the implementation of five key processes: initiation, planning, execution, monitoring and controlling, and closing.

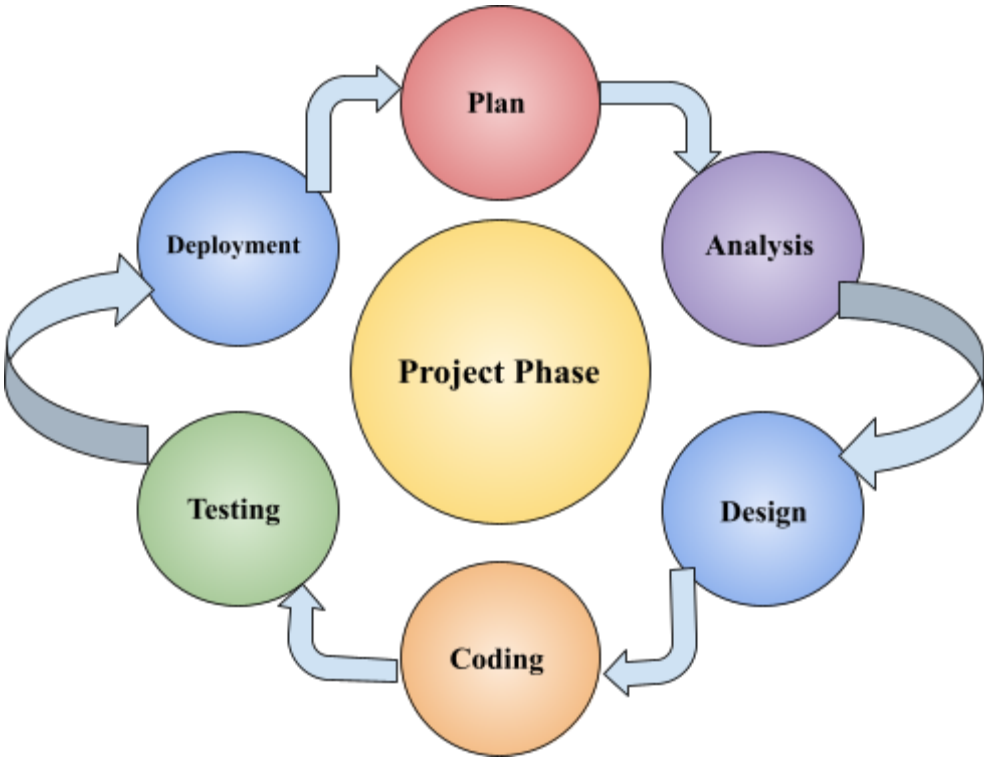
Initiation involves defining the project, identifying its objectives, and determining whether it is feasible to undertake the project. Planning involves creating a detailed plan of how the project will be executed.

Execution involves the actual implementation of the project plan. Monitoring and controlling involve tracking project progress, identifying any issues or deviations from the plan, and taking corrective action to keep the project on track. Finally, the closing process involves the formal acceptance of the project's deliverables and closing out the project.

Effective project management requires a range of skills, including communication, leadership and risk management. It also involves the use of various tools and techniques, such as Gantt charts, critical path analysis, and project scheduling software.

In summary, project management is a critical aspect of successful project development. It involves the application of knowledge, skills, tools, and techniques to manage project activities and meet project requirements. Effective project management involves the implementation of five key processes, and requires a range of skills and tools to be successful.

Project management focuses on achieving the objectives by applying the processes presented in the figure below:



Source: https://premieragile.com/5-phases-of-project-management/

Figure 2.1: Project development phases

2.2 Workout Buddy Finder App

The Workout Buddy Finder App is an innovative mobile application designed to help users find the perfect workout partner to achieve their fitness goals. With the app's comprehensive suggestion function, users can easily filter their suggestions results based on their interests and locality, making it easy to find a workout buddy who shares the same passion for fitness and lives nearby.

One of the most significant challenges when it comes to sticking to a fitness routine is staying motivated. This is where finding a workout buddy can be incredibly helpful. A workout buddy can provide the motivation and support needed to stay on track with fitness goals. However, finding a workout partner who shares the same interests and lives in the same area can be challenging.

The Workout Buddy Finder App makes this process easier by providing a platform for users to connect with like-minded individuals who share the same fitness interests and goals. The app's user-friendly interface allows users to input their interests and preferences, and the app's matching algorithms suggest workout buddies who share similar interests.

In addition to providing a platform to find workout buddies, the app also features a social networking aspect that allows users to connect with other fitness enthusiasts in their area.

Users can share their progress, tips, and workout experiences, creating a supportive and motivational community of like-minded individuals.

Overall, the Workout Buddy Finder App is a must-have tool for anyone looking to stay motivated and achieve their fitness goals. With its innovative features and user-friendly interface, the app provides a personalized workout experience that is tailored to the user's interests and preferences.

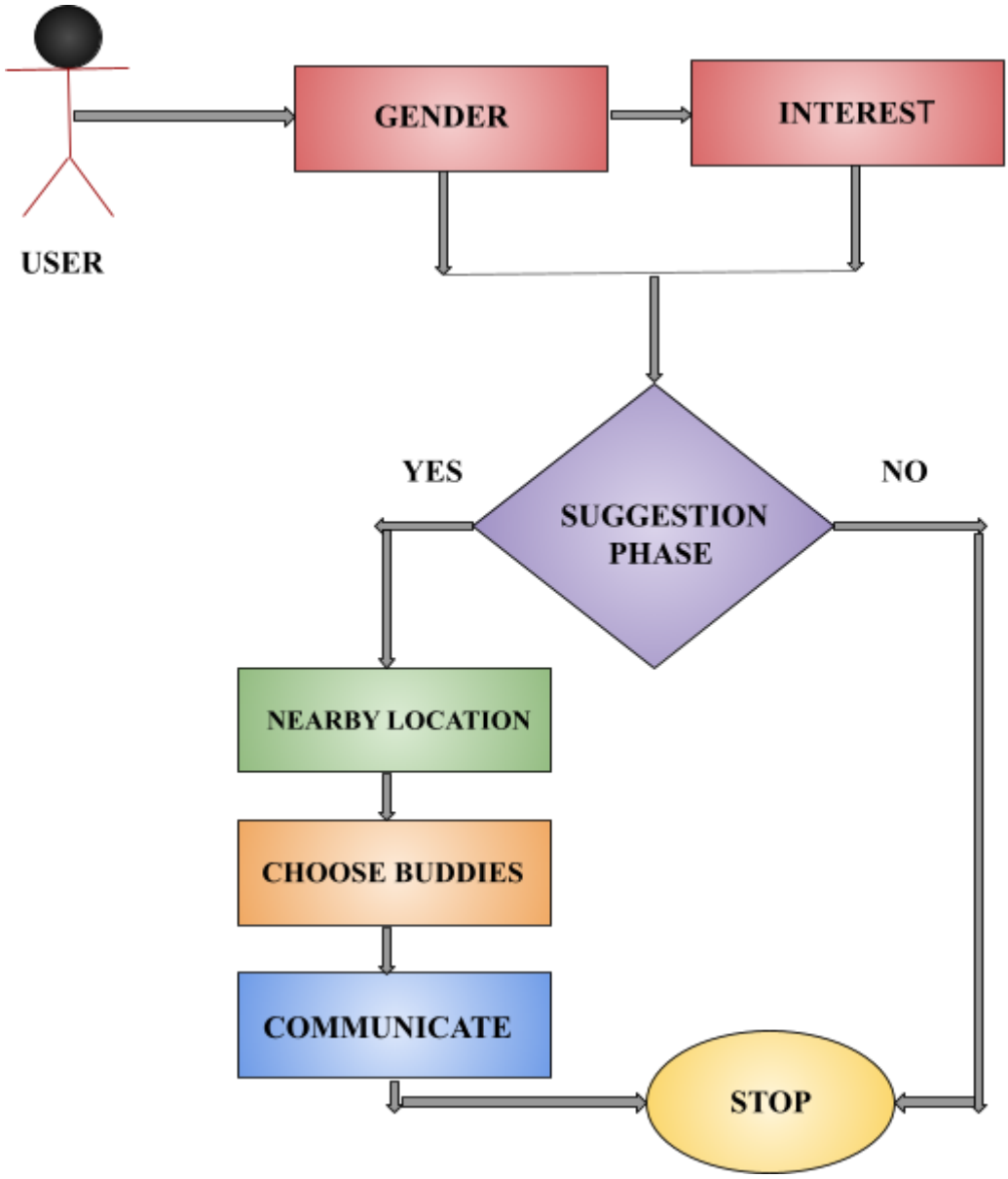


Figure 2.2: Data flow diagram for workout buddy finder app

2.3 Features

Getting fit is a complex task that combines not only the physical exercise itself, but also other aspects of a person's daily routine such as how well the person sleeps during the night.

Therefore, the designed app for this project tries to cover a huge variety of functionality so that the user has all the features that he/she may need when getting fit and exercising.

- **Login, Registration:** The mobile application offers users the ability to create new accounts and log in using their Google account. This means that users can quickly and easily create a new account by using their existing Google credentials. By integrating this feature into the application, users can save time and avoid the hassle of filling out long registration forms or remembering new login information. Additionally, this feature provides an added layer of security, as users can leverage Google's authentication protocols to ensure the protection of their personal information. Overall, the integration of Google account login adds convenience and security to the user experience of the application.

- **Managing Fitness Profile:** Within the mobile application, users have the ability to manage their fitness profile, which contains their personal information, fitness goals, and exercise preferences. By accessing their profile, users can easily view and update their fitness-related information in one centralized location. This feature allows users to keep track of their progress and tailor their workout routine to their individual needs and preferences.
  Moreover, users have the ability to create, update, and delete their choice of exercise and other interests within the app. This means that users can customize their experience by selecting their preferred exercise activities and indicating their interests. By providing this level of flexibility, the application ensures that users can find workout buddies who share their interests and fitness goals, thus promoting a more effective and enjoyable workout experience. Additionally, by allowing users to modify their exercise and interest choices, the application can adapt to users' changing needs over time.

- **Personal workout preference:** Within this mobile application, users are able to set preferences for finding nearby people with similar interests. This feature allows users to specify their desired criteria for finding workout buddies, such as location, age, gender, fitness level, and specific exercise activities or interests. By setting these preferences, users can narrow down their search and find workout partners who share their goals and interests.
  For instance, if a user is interested in finding a workout partner who lives nearby, they can set their location preference to a specific radius around their current location. Similarly, if a user is interested in finding a workout partner with a similar fitness level, they can set their preferences to match their own fitness level. Additionally, users can specify their preferred exercise activities or interests, such as yoga, weightlifting, running, or cycling.

By providing this level of customization, the application ensures that users can find suitable workout partners based on their individual preferences. This not only enhances the overall workout experience but also promotes a sense of community and support among like-minded individuals. Overall, the preference-setting feature is a crucial aspect of the application that makes it easier and more convenient for users to find compatible workout partners nearby.

- **Find nearby people with similar interests:** The mobile app has a feature that allows users to view and connect with nearby people who share similar interests. This feature utilizes the user's location information to provide them with a list of individuals who are within their vicinity and have similar interests as they do. The app provides a suggestion of nearby people based on the user's preference and their listed interests. Users can then choose to view the profiles of these suggested individuals and send them a connection request if they wish to connect. This feature is designed to help users expand their social circle, find workout partners, and make new friends with similar interests.

- **Chat with matched people:** The mobile app has a feature that allows users to chat with matched people. When two users express an interest in each other, they are notified of the match and given the option to start a conversation. Users can use the in-app messaging system to chat with their matches and get to know them better. This feature is designed to provide a way for users to communicate with each other and potentially make plans to work out together or participate in other activities based on their shared interests. The messaging system allows users to exchange information about their fitness routines, preferences, and goals, helping them to better understand each other's needs and motivations.

2.4 Justification

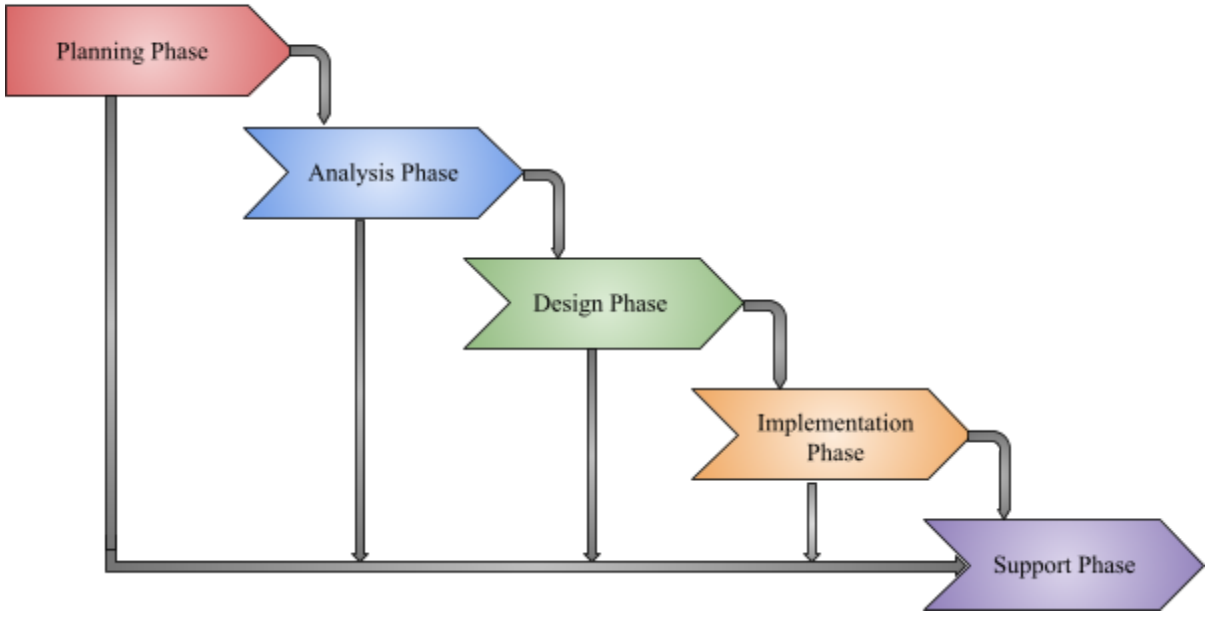
Modern lifestyle has led to a decrease in physical activity among individuals, as technology has made many tasks easier and more convenient. With the increased use of cars and public transport, machines for washing clothes, and entertainment options such as television and computers, people are becoming more sedentary.
However, regular physical activity is crucial for maintaining good health. It has been shown to improve brain function, help with weight management, reduce the risk of various diseases, strengthen bones and muscles, and enhance the ability to perform daily tasks.

To combat this modern problem, the team came up with a modern solution: a mobile application that utilizes technology to encourage physical activity. The app enables users to find workout buddies based on similar interests and locality, set fitness preferences, and communicate with their matches to motivate each other to stay active. By using technology in this way, the app aims to make physical activity more accessible, engaging, and enjoyable for individuals.

CHAPTER 3
FEATURES, TOOLS AND TECHNOLOGY

3.1 System Development Life Cycle

Software Development Life Cycle is a systematic process for building software that ensures the quality and correctness of the software built. The SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase. SDLC stands for Software Development Life Cycle and is also referred to as the Application Development life-cycle.



Source: https://www.numpyninja.com/post/sdlc-software-development-life-cycle-phases-process-what-is-sdlc

Figure 3.1: Software Development Life Cycle phases

There are five phases in this model and the first phase is the planning stage. Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage. In this phase one or more system analysts work with different stakeholder groups to determine the specific requirements for the new system. No programming is done in this step. Instead, procedures are documented, key players/users are interviewed, and data

requirements are developed in order to get an overall impression of exactly what the system is supposed to do. The result of this phase is a system requirements document and may be done by someone with a title of Systems Analyst.

During this phase of the System Development Life Cycle, the requirements and desired functions are described in great detail, including process charts, rules, and other documentation.

The third phase is the moment when end users have an opportunity to discuss and decide their specific information needs. This is also the phase where essential components of the system (hardware, software) and structure are considered.

During the fourth phase, the actual development starts. This is especially the case when a programmer, engineer, or database developer is called in to do important work for the developed project. These operations consist of, amongst other things, making flowcharts that ensure that the process and new system are carefully organized.

The development phase marks the end of the first stage of the Systems Development Life Cycle (SDLC). After this, the production stage of the project begins. The development phase is characterized by change.

Once the software is complete, it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, the QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and sends it back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

3.2 Tools and Technology

According to Statista, as of January 2021, the number of smartphone users in Bangladesh is around 93.4 million, which is approximately 55% of the country's population. This indicates that a significant portion of the population in Bangladesh owns a smartphone, providing a vast potential user base for mobile applications.

So, the team decided to bring a solution that can be used by smartphone.
An application for Android and iOS smartphones is developed so that it reaches most of the population. Below are the reasons to choose the technology stack:

- **Flutter (for Android and iOS App):** Flutter is a popular cross-platform mobile app development framework that has gained popularity in recent years due to its numerous advantages. One reason to use Flutter is its high performance and fast development time, thanks to its hot-reload feature. This allows developers to make changes to the code and see the results immediately without having to restart the app.
  Another reason is its ability to create beautiful and highly customizable user interfaces. Flutter uses a flexible widget system that allows developers to build UIs that are both aesthetically pleasing and functional. Additionally, Flutter has a large collection of pre-built widgets that can be customized to fit specific design needs.

  Flutter's compatibility with multiple platforms is also a major advantage. It can be used to build apps for both Android and iOS platforms, as well as for web and desktop applications. This saves developers time and effort in creating separate apps for each platform.

  Finally, Flutter has a growing community of developers who contribute to its open-source libraries and plugins, making it easier to find solutions to problems and to add new features to your app.

- **Firebase Realtime Database (NoSQL):** Firestore is a NoSQL document-based database that offers a flexible data structure and a real-time sync feature. One of the main reasons to choose Firestore is its scalability. It can easily handle a large amount of data and provide fast query responses.
  Another reason to choose Firestore is its real-time data synchronization. Firestore automatically updates the data across all clients in real-time, which means that any changes made by one user are instantly reflected in the app for all users. This feature is particularly useful for real-time applications such as chat applications or collaborative workspaces.

  Firestore also offers strong security features, allowing users to set up rules to control access to data. This makes it easier to secure sensitive information and prevent unauthorized access to the data.

Finally, Firestore provides an easy-to-use interface that allows developers to quickly set up and manage their databases. Its integration with other Google Cloud services such as Cloud Functions and Cloud Storage also makes it a popular choice among developers who use these services for their applications.

- **Google Authentication:** Google Authentication was chosen for the mobile application due to several reasons. Firstly, it provides a secure and easy way for users to log in to the application without having to create and remember a separate username and password. Additionally, it eliminates the need for the application to store sensitive login information, reducing the risk of a security breach.

  Furthermore, Google Authentication is a widely recognized and trusted method of authentication, which can help to increase user confidence and trust in the application. It is also compatible with a wide range of platforms and devices, ensuring accessibility for a large number of users. Overall, these factors make Google Authentication a strong choice for the mobile application.

- **Android Studio:** Android Studio is a popular integrated development environment (IDE) widely used for Flutter app development. It provides a comprehensive set of tools and features specifically designed to support Flutter development and streamline the app creation process.

  One of the key advantages of using Android Studio for Flutter is its seamless integration with the Flutter framework. It comes with built-in support for Flutter SDK, allowing developers to easily create, edit, and run Flutter projects within the IDE. Android Studio provides a user-friendly interface and offers various functionalities that enhance the development workflow.

  Android Studio offers a powerful code editor with features like code completion, syntax highlighting, and refactoring tools, making it easier to write clean and efficient Flutter code. It also supports hot reload, a feature that enables developers to instantly see the changes they make to their code in real-time, speeding up the development and testing process. The IDE includes a rich set of visual layout editors, such as the Flutter widget inspector and the Layout Editor, which enable developers to design and preview the user interface of their Flutter apps. These visual tools make it simpler to create and customize the UI components, layouts, and themes, ensuring a visually appealing and responsive app design.

  Android Studio provides robust debugging capabilities for Flutter apps. It allows developers to set breakpoints, inspect variables, and step through the code, helping them identify and fix issues quickly. Additionally, the IDE offers profiling tools to analyze app performance, memory usage, and CPU usage, ensuring optimal app performance.

CHAPTER 4
PROJECT OUTPUT

4.1 Introduction

The design and implementation of the project is a critical aspect of its success. In this project, the team used Flutter, a mobile app SDK, to create a cross-platform mobile application for Android and iOS smartphones. Flutter is chosen for its fast development cycle, hot reload feature, and customizable widgets that allow for beautiful and responsive design.

The app used Firestore, a NoSQL cloud-based database by Google, to store user data, workout preferences, and suggestions. Firestore's scalability, security features, and real-time synchronization capabilities make it an ideal choice for this project.

Furthermore, to ensure a secure and seamless login process for users, the app is integrated with Google Authentication. This allows users to easily create an account, log in, and access the app's features using their existing Google credentials.

Overall, the design and implementation of this project was critical in achieving its goal of promoting physical activity and helping people find workout buddies in their local area.

4.2 Login

To access the application, the initial step involves logging in. Upon accessing the login page, users are greeted with an introduction screen, featuring a "Login with Google" button. After clicking the button, a welcome message is displayed along with a "Continue with Google" option. To offer users relevant suggestions, location permission is needed to be granted after logging in.

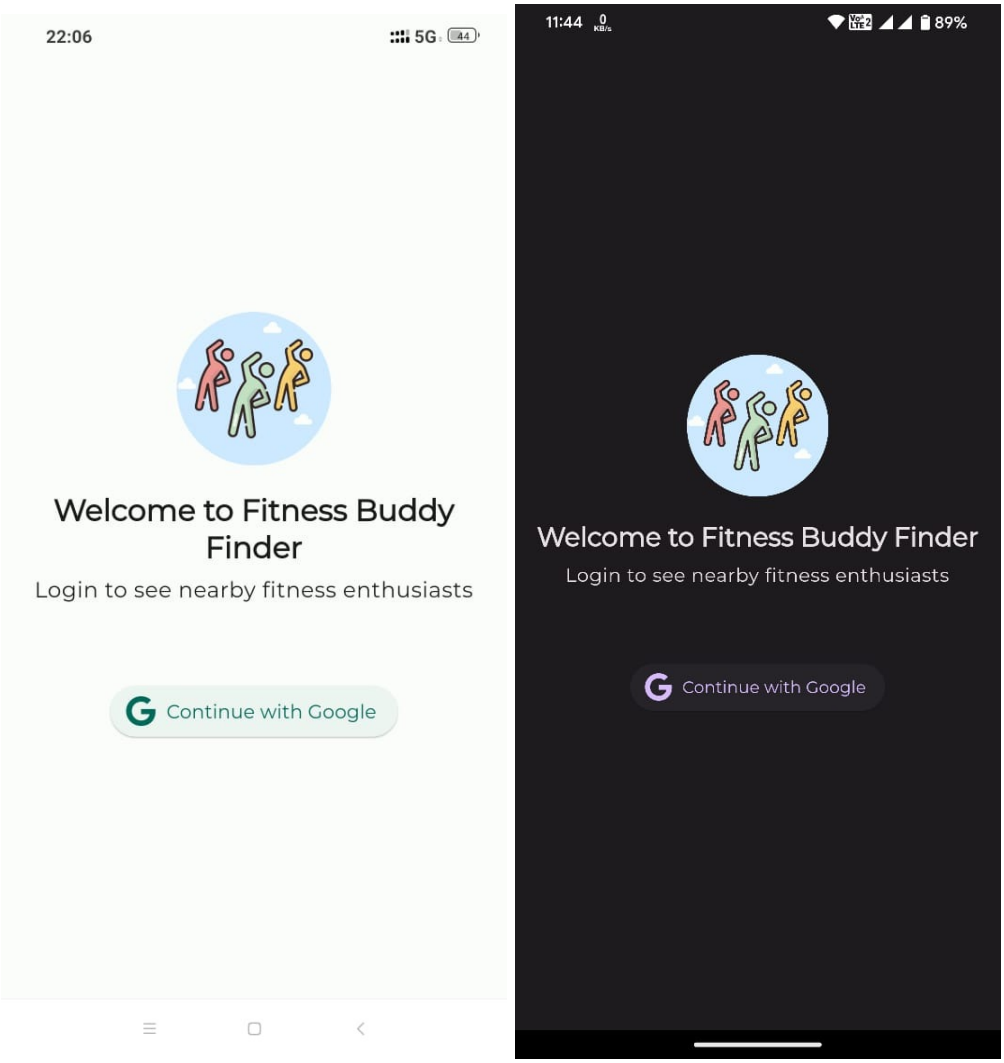


Figure 4.1: Login Screen

4.3 Suggestions

The Suggestion Screen is a key feature of the app that aims to connect users with like-minded people nearby. The app uses various criteria such as preferred distance, gender, interests, and availability to generate personalized suggestions for each user. This feature not only helps users find workout partners but also fosters a sense of community among fitness enthusiasts.

In addition to the Suggestion Screen, users can also update their profile and other preferences from the Edit Profile Screen. This allows users to keep their information up-to-date and tailor their experience within the app to their specific needs. By providing users with the ability to customize their preferences and interests, the app ensures that the suggestions generated by the Suggestion Screen are relevant and meaningful to each individual user.

Figure 4.2: Suggestion Screen

4.4 Messages

In the messaging feature of the application, users are able to communicate with their connected people in real-time using the Firestore Database. This means that the messages will be delivered instantly and the conversation can flow smoothly without any delay. Users can access the messaging feature from a dedicated messaging page, where all their conversations with different people are organized in a list. The messaging page also has options to view profile information and initiate a new chat with a new connection. Overall, this feature will enable users to establish and maintain connections with their workout buddies easily and efficiently.

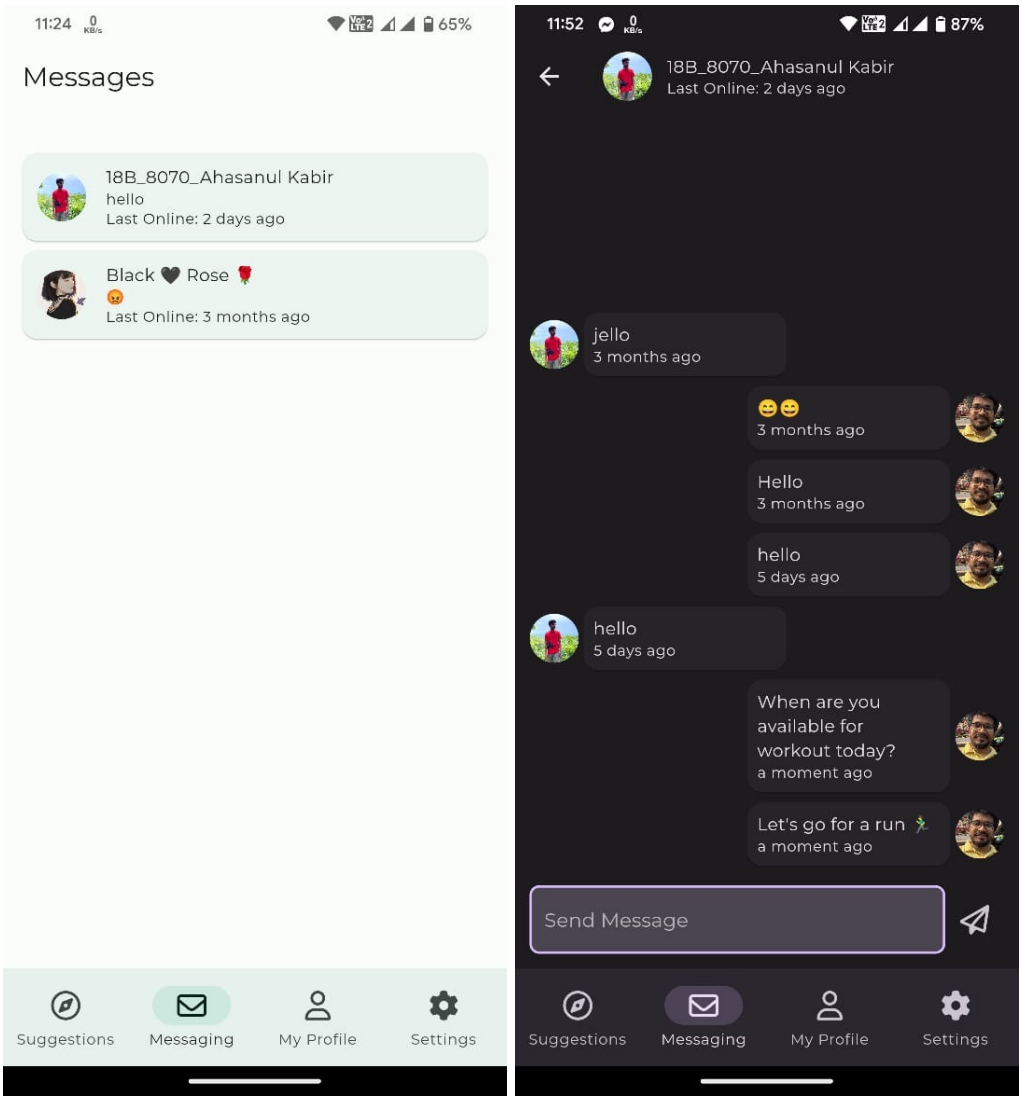


Figure 4.3: Live Chatting Screen

4.5 Profile

The Profile section of the mobile application displays important information related to the logged-in user such as the number of calories burned in the last 24 hours, preferred workout time, and interests. Users will be able to access and modify this information by clicking on the "edit profile" button. The user's fitness information can be updated regularly to keep track of their progress and improve their overall experience within the app. This feature will help users stay motivated and focused on their fitness goals by providing them with a personalized experience.

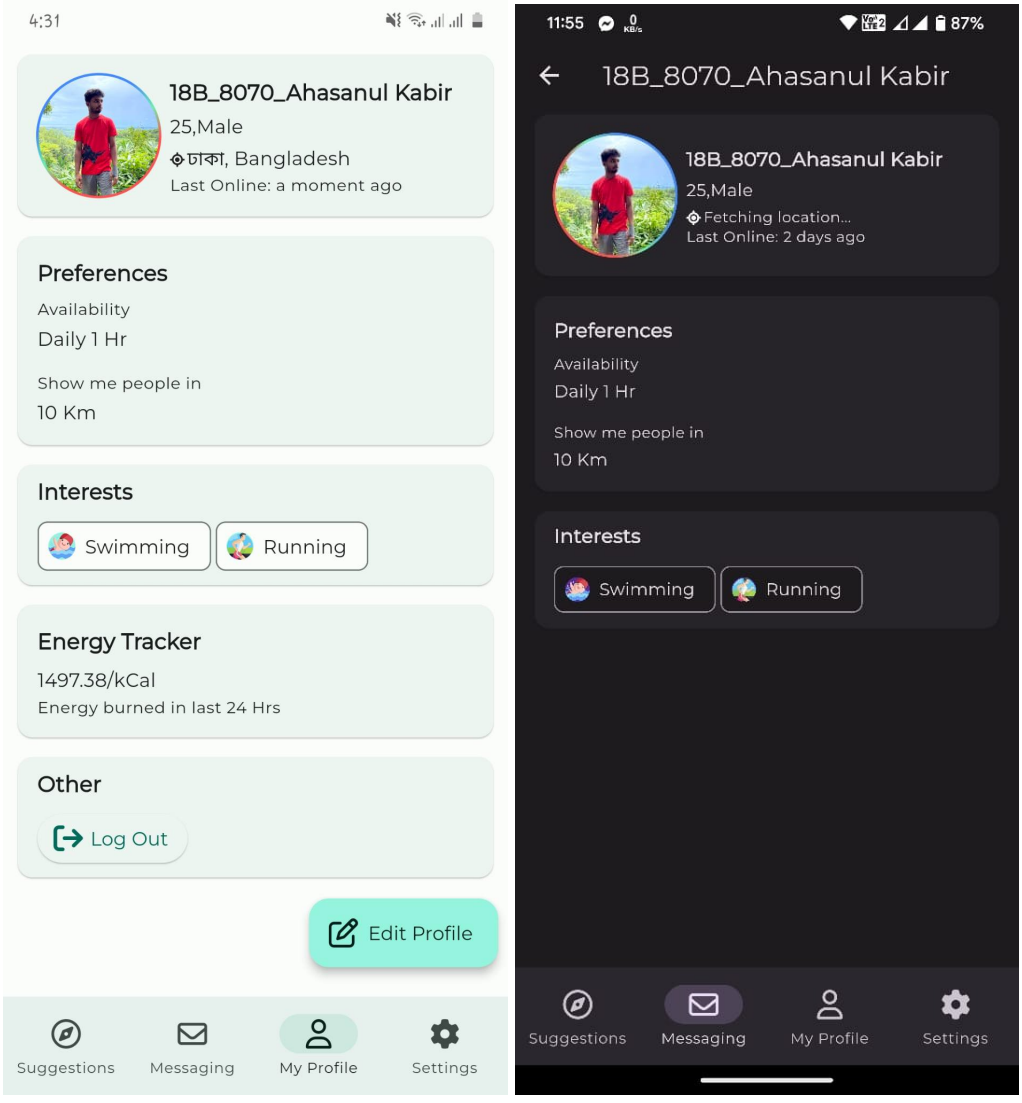


Figure 4.4: Profile Screen

4.6 Edit Profile

On the profile page, users have the option to update their personal information and preferences. This includes their workout interest, preferred distance for receiving people's suggestions, birthdate, profile picture, expected workout time, and the option to hide their profile from being discovered by others. By updating their profile, users can ensure that their preferences and interests are accurately reflected, allowing the app to provide more relevant and tailored suggestions for nearby users. The ability to hide their profile can also provide users with a greater sense of privacy and control over their information.

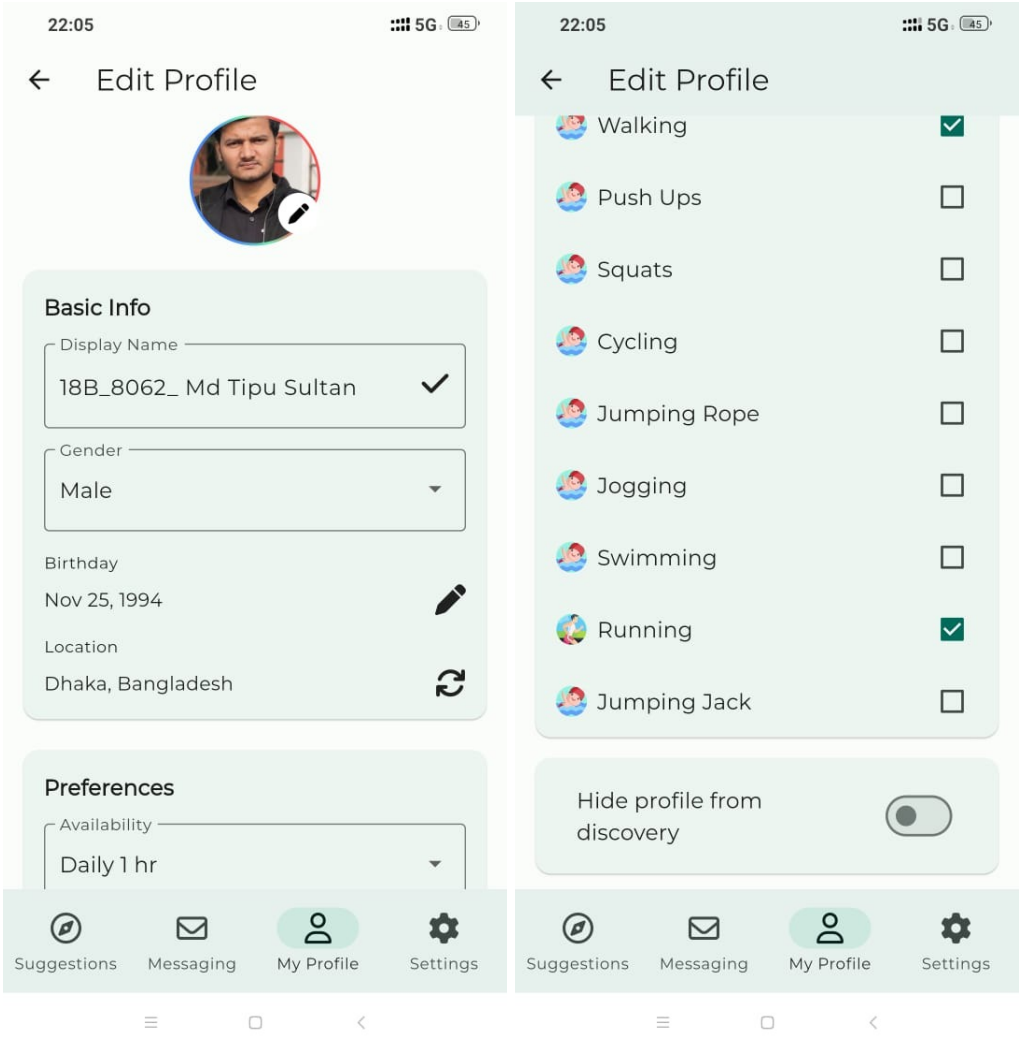


Figure 4.5: Edit Profile Screen

4.7 Settings

In the Settings page, users can access the theme preferences section where they can customize the look and feel of the application according to their preferences. There are currently three different theme options available for users to choose from, which include light, dark, and automatic. The light theme has a white background with black text, while the dark theme has a black background with white text. The automatic theme automatically switches between light and dark mode based on the device's settings. By providing these options, users can personalize their experience and make it more comfortable for them to use the application.

Figure 4.6: App Settings Screen

CHAPTER 5
CONCLUSION, LIMITATIONS AND FUTURE WORKS

5.1 Conclusion

In conclusion, the mobile app has the potential to revolutionize the way people connect with others who share their interests. It offers a convenient and user-friendly platform for people to find and connect with like-minded individuals nearby, creating new social opportunities and fostering meaningful relationships.

Through the app, users can search for others based on their hobbies, interests, and preferences, and can engage in conversations, plan activities, and explore new ideas. This app has the potential to bring people together and build strong communities, creating a positive impact on their lives and contributing to the social fabric of our society.

The development process of our app has been an incredible journey, and the team have learned so much about user-centered design, technical skills, creativity, and dedication. The collaborative approach has ensured that our app meets the needs and preferences of our target audience, providing a seamless user experience that is both intuitive and engaging.

As the app gains traction and becomes more popular, it has the potential to become a game-changer in the world of social networking. With our collective passion, expertise, and vision, the team has created a solution that addresses a real-world problem and has the potential to make a significant impact on people's lives.

In conclusion, the app is a remarkable achievement, and the team is excited to see its continued growth and success in the future. Developing such an innovative and impactful solution has been a significant accomplishment, and the team is proud of what is achieved together. The team looks forward to exploring new opportunities and challenges in the future and will continue to work as a team to develop our skills and pursue our goals.

5.2 Limitations

As with any technology, this app has some limitations that need to be considered. Here are some potential limitations that has been identified:

- **User adoption and engagement**: The success of the app depends on how many people use it and how often they engage with it. While the team has designed the app to be intuitive and user-friendly, the team recognizes that there are many other apps on the market that may compete with ours for users' attention.
- **Data quality**: The accuracy of our matching algorithm depends on the quality and quantity of user data. While the team has designed the app to collect relevant information about users' interests and preferences, the team recognizes that some users may not provide enough data or may provide inaccurate data.

- **Technical challenges**: As with any software, our app may face technical challenges such as server crashes, security breaches, and software bugs. The team has implemented measures to minimize these risks, but the team recognizes that they may still occur and may affect the user experience.
- **Privacy concerns**: Our app's social features may raise privacy concerns among some users, who may be hesitant to share personal information or connect with strangers online. The team has implemented privacy controls and guidelines to address these concerns, but the team recognizes that some users may still be hesitant to use the app.
- **Geographic and demographic limitations:** The effectiveness of the app may vary depending on the geographic location, demographic profile, and cultural context of its users. While the team have designed the app to be flexible and adaptable to different contexts, the team recognizes that some users may not find the app relevant or useful for their specific needs.
- **Revenue model uncertainty**: The app's revenue model may be uncertain, as it may rely on advertising, subscription fees, or other sources of income that are subject to market fluctuations and user preferences. The team has designed the app to be sustainable and scalable, but the team recognizes that revenue may not be guaranteed.

Overall, while these limitations represent potential challenges to the success of the app, the team is confident that the team can address them through ongoing innovation and user feedback. By continuously improving the app's features and user experience, the team believes that the team can create a valuable solution for users and contribute to the broader social networking landscape.

5.3 Future Works

Looking ahead, the team sees several potential areas for future work and innovation in our app. These include:

- **Expanding the app's social features**: The team believes that expanding the app's social features could be a key area for future work. By adding more ways for users to interact and connect, the team can encourage more engagement and participation, and help users build more robust and diverse social networks.
- **Improving the app's matching algorithm**: Another area for future work is to improve the app's matching algorithm. By incorporating more sophisticated machine learning and data analysis techniques, the team can increase the accuracy of the algorithm and provide users with more relevant and meaningful matches.
- **Integrating with other platforms**: The team believes that integrating our app with other social networking platforms could be a valuable way to expand our user base and reach more people. By providing a seamless and integrated experience across multiple platforms, the team can make it easier for users to connect with others who share their interests and preferences.
- **Personalizing the user experience**: The team believes that personalizing the user experience could be a powerful way to increase user engagement and satisfaction. By incorporating more user-specific data and preferences, users can be provided with

personalized recommendations, notifications, and reminders, which can help them stay engaged and connected over time.

- **Developing a sustainable business model**: To ensure the long-term viability of the app, the team recognizes the importance of developing a sustainable and scalable business model. This could involve exploring different revenue streams, such as advertising, subscription fees, or partnerships, and developing strategies for monetizing the app without compromising the user experience.
- **Addressing user feedback and concerns**: Finally, the team recognizes that addressing user feedback and concerns will be an ongoing area of future work. By soliciting and incorporating user feedback into the app's development process, the team can ensure that the app remains relevant, useful, and engaging for users, and can continue to build a strong and loyal user base over time.

Overall, the team believes that these areas of future work represent exciting opportunities for our team to continue to innovate and improve the app. By staying focused on our mission of helping people connect with others who share their interests and passions, the team is confident that the team can build a valuable and sustainable solution for users and contribute to the broader social networking landscape.

REFERENCES

[1] Robert C. Martin, “The Clean Architecture” , The Clean Code Blog, 2012. [Online Blog]. Available:https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Accessed Dec. 23, 2022].

[2] Google, “The official package repository for Dart and Flutter apps”, 2023. [Online Repository]. Available: https://pub.dev/[Accessed Dec. Jan, 2023].

[3] Codemagic, “A beginner’s guide to go_router in Flutter”, 2022. [Online Blog]. Available: https://blog.codemagic.io/flutter-go-router-guide/[Accessed Dec. Feb, 2023].

[4] Felix Angelov, “A predictable state management library for Dart”, 2022. [Online Blog]. Available: https://bloclibrary.dev/[Accessed Dec. Nov, 2022].

[5] Michael Adeyemo, “INTRODUCTION TO FLUTTER HOOKS”, 2021. [Online Blog]. Available: https://www.topcoder.com/thrive/articles/introduction-to-flutter-hooks/[Accessed Dec. Nov, 2022].

[6] FlutterFire, “The official Firebase plugins for Flutter”, 2020. [Documentation]. Available: https://firebase.flutter.dev/[Accessed Dec. Nov, 2022].

[7] Google, “Firebase Storage and Analytics for Flutter”, 2023. [Documentation]. Available: https://firebase.google.com/docs/flutter/setup?platform=ios[Accessed Dec. Jan, 2023].

[8] Github, “GitHub is where over 100 million developers shape the future of software, together”, 2013. [Code Hosting Service]. Available: https://github.com/[Accessed Dec. Feb, 2023].

[9] Flutter, “Flutter, Build app for any screen”, 2017. [Documentation]. Available: https://flutter.dev/[Accessed Dec. Aug, 2022].

[10] Stack Overflow, “Stack Overflow - Where Developers Learn, Share, & Build”, 2008. [QA Website]. Available: https://stackoverflow.com//[Accessed Dec. March, 2023].

APPENDIX

wbf_app.dart: This is the entrypoint to the application. This file runs when the app opens.

```
class WBFApp extends HookWidget {
 final String appName;
 final EnvType envType;

 const WBFApp({super.key,
   required this.appName,
   required this.envType,
 });

 @override
 Widget build(BuildContext context) {
   useListenable(ThemeManager.instance);

   return AuthStateListenerApp(appName: appName, envType: envType);
 }
}
```

auth_state_listener_app.dart: This file runs and rebuilds the app ui when auth state changes.

```
class AuthStateListenerApp extends HookWidget {
 const AuthStateListenerApp({
   super.key,
   required this.appName,
   required this.envType,
 });

 final String appName;
 final EnvType envType
```

```
 @override
 Widget build(BuildContext context){
final stream = useMemoized(() =>
sl<AuthStateStreamUseCase>().execute());
   final authStatSnapShot = useStream(stream);
   final authState = authStatSnapShot.data;

   if (authState == UserAuthState.signedIn) {
     return AppUserListenerApp(appName: appName, envType: envType);
   } else if (authState == UserAuthState.signedOut) {
     return AuthApp(appName: appName);
   }

   return LoadingApp(appName: appName, envType: envType);
 }
}
```

app_user_listener_app.dart: This file runs and rebuilds the app ui when the app user changes.

```
class AppUserListenerApp extends HookWidget {
 const AppUserListenerApp({
   super.key,
   required this.appName,
   required this.envType,
 });

 final String appName;
 final EnvType envType;

 @override
 Widget build(BuildContext context) {
final stream = useMemoized(() =>
sl<AppUserStreamUseCase>().execute());
   final appUserSnapShot = useStream(stream);
```

```
return RepositoryProvider<AppUser>.value(
              value: appUser,
              child: MaterialApp.router(
                routerConfig: router,
                title: appName,
                theme: theme,
                darkTheme: darkTheme,
                themeMode: ThemeManager.instance.themeMode,
                debugShowCheckedModeBanner: false,
              ),
            );
          }

          return LoadingApp(appName: appName, envType: envType);
        }
      }
```

dark_theme.dart: This file contains dark theme colors

```
const _seedColor = Color(0xff410081);
final baseTheme =
              ThemeData(colorSchemeSeed:    _seedColor,    brightness:
Brightness.dark, useMaterial3: true);
final darkTheme = baseTheme.copyWith(
 textTheme: GoogleFonts.montserratTextTheme(baseTheme.textTheme),
);
```

light_theme.dart: This file contains light theme colors

```
const _seedColor = Color(0xFF01947E);
final baseTheme =
              ThemeData(colorSchemeSeed:    _seedColor,    brightness:
Brightness.light, useMaterial3: true);
final theme = baseTheme.copyWith(
 textTheme: GoogleFonts.montserratTextTheme(baseTheme.textTheme),
);
```

theme_manager.dart: This class manages dynamic theme switching.

```
class ThemeManager extends ChangeNotifier {
 static ThemeManager? _themeManager;

 final String _prefKey = "uiMode";

 ThemeMode _themeMode = ThemeMode.system;


 ThemeMode get themeMode => _themeMode;

 static ThemeManager get instance {
   _themeManager ??= ThemeManager._();

   return _themeManager!;
 }

 ThemeManager._();

 Future<void> switchTo(ThemeMode themeMode) async {
   if (themeMode != this.themeMode) {
     await _saveThemeInPrefs(themeMode);
     _themeMode = themeMode;
     notifyListeners();
   }
 }

 Future<void> init() async {
   final pref = await SharedPreferences.getInstance();
   final uiMode = pref.getInt(_prefKey) ?? 0;
   switch (uiMode) {
     case 0:
```

```
        await switchTo(ThemeMode.system);
        break;

      case 1:
        await switchTo(ThemeMode.light);
        break;

      case 2:
        await switchTo(ThemeMode.dark);
        break;

      default:
        await switchTo(ThemeMode.system);
    }
}

Future<void> _saveThemeInPrefs(ThemeMode themeMode) async {
  final pref = await SharedPreferences.getInstance();
  int uiMode = 0;

  switch (themeMode) {
    case ThemeMode.system:
      uiMode = 0;
      break;

    case ThemeMode.light:
      uiMode = 1;
      break;

    case ThemeMode.dark:
      uiMode = 2;
      break;
```

```
    }

    final result = await pref.setInt(_prefKey, uiMode);
  }
}
```

app_route.dart: This class maps widgets with respective routes.

```
abstract class AppRoute {
 String get route;
}

class TopLevelRoute implements AppRoute {
 final String _route;

 @override
 String get route => _route;

 const TopLevelRoute._({required String route}) : _route = route;

 const TopLevelRoute.signIn() : this._(route: '/sign-in');

 const TopLevelRoute.signUp() : this._(route: '/sign-up');

 const TopLevelRoute.suggestion() : this._(route: '/suggestion');

 const TopLevelRoute.profile() : this._(route: '/profile');

 const TopLevelRoute.messaging() : this._(route: '/messaging');

 const TopLevelRoute.settings() : this._(route: '/settings');
}

class ChildRoute implements AppRoute {
```

```
  final String _route;

  @override
  String get route => _route;

  const ChildRoute({required String route}) : _route = route;
}
```

app_router.dart: This class container route navigation callbacks.

```
class RouteNavigatorKey {
  static final rootNavigatorKey = GlobalKey<NavigatorState>();
  static final shellNavigatorKey = GlobalKey<NavigatorState>();
}

final router = GoRouter(
   routes: [
     ShellRoute(
         builder: (context, state, child) => ScaffoldWithBottomNav(
               currentRoute: state.location,
               onOpenSuggestion: () =>
                                                    context.go(const
TopLevelRoute.suggestion().route),
               onOpenMessaging: () =>
                   context.go(const TopLevelRoute.messaging().route),
               onOpenProfile: () =>
                   context.go(const TopLevelRoute.profile().route),
               onOpenSettings: () =>
                   context.go(const TopLevelRoute.settings().route),
               child: child,
             ),
         routes: _appRoutes,
         navigatorKey: RouteNavigatorKey.shellNavigatorKey),
   ],
```

```
          errorBuilder:    (_,    state)    =>    ErrorIndicator(error:
state.error?.toString()),
   initialLocation: const TopLevelRoute.suggestion().route,
   navigatorKey: RouteNavigatorKey.rootNavigatorKey);


List<GoRoute> get _appRoutes => [
     GoRoute(
       path: const TopLevelRoute.suggestion().route,
       pageBuilder: (context, _) {
         return NoTransitionPage(
           child: SuggestionsPage(
             onShowProfile: () =>
                 context.go(const TopLevelRoute.profile().route),
             onOpenMessaging: () =>
                 context.go(const TopLevelRoute.messaging().route),
             onOpenProfile: (userId) => context.go(
               const VisitUserRoute().generateNavRoute(
                 root: const TopLevelRoute.suggestion().route,
                 userId: userId,
               ),
             ),
           ),
         );
       },
       routes: [
         GoRoute(
           path: const VisitUserRoute().route,
           builder: (context, state) =>
               VisitUserPage(userId: state.params['userId']!),
         ),
       ],
     ),
```

```
GoRoute(
  path: const TopLevelRoute.profile().route,
  pageBuilder: (context, _) {
    return NoTransitionPage(
      child: ProfilePage(
        onEditProfile: () => context.go(
          const EditProfileRoute().generateNavRoute(
              root: const TopLevelRoute.profile().route),
        ),
      ),
    );
  },
  routes: [
    GoRoute(
      path: const EditProfileRoute().route,
      builder: (context, _) => const ProfileEditPage(),
    ),
  ],
),
GoRoute(
  path: const TopLevelRoute.messaging().route,
  pageBuilder: (context, _) => NoTransitionPage(
    child: MessagingPage(
      onOpenChatRoom: (chatRoomId) => context.go(
        const ChatRoomRoute().generateNavRoute(
          root: const TopLevelRoute.messaging().route,
          chatRoomId: chatRoomId,
        ),
      ),
    ),
  ),
  routes: [
```

```
            GoRoute(
              path: const ChatRoomRoute().route,
              builder: (context, state) => ChatRoomPage(
                chatRoomId: state.params['chatRoomId']!,
                onVisitProfile: (userId) => context.go(
                  const VisitUserRoute().generateNavRoute(
                    root: const ChatRoomRoute().generateNavRoute(
                      root: const TopLevelRoute.messaging().route,
                      chatRoomId: state.params['chatRoomId']!,
                    ),
                    userId: userId,
                  ),
                ),
              ),
              routes: [
                GoRoute(
                  path: const VisitUserRoute().route,
                  builder: (_, state) =>
                      VisitUserPage(userId: state.params['userId']!),
                ),
              ],
            ),
          ],
        ),
        GoRoute(
          path: const TopLevelRoute.settings().route,
          pageBuilder: (context, _) =>
              const NoTransitionPage(child: SettingsPage()),
        ),
      ];
```

auth_router.dart: This class contains routes respective to authorization state.

```
final _authNavigatorKey = GlobalKey<NavigatorState>();
```

```
final authRouter = GoRouter(
 navigatorKey: _authNavigatorKey,
 initialLocation: const TopLevelRoute.signIn().route,
 routes: _authRoutes,
);

List<GoRoute> get _authRoutes => [
     GoRoute(
       path: const TopLevelRoute.signIn().route,
          pageBuilder: (context, state) => const MaterialPage(child: SignInPage()),
     ),
     GoRoute(
       path: const TopLevelRoute.signUp().route,
          pageBuilder: (context, state) => const MaterialPage(child: SignUpPage()),
     ),
   ];
```

routes.dart: This file contains child routes

```
class EditProfileRoute extends ChildRoute {
 const EditProfileRoute() : super(route: 'edit-profile');

 String generateNavRoute({required String root}) {
   return '$root/edit-profile';
 }
}

class ChatRoomRoute extends ChildRoute {
 const ChatRoomRoute() : super(route: 'chat-room/:chatRoomId');

  String generateNavRoute({required String root, required String chatRoomId}) {
   return '$root/chat-room/$chatRoomId';
```

```
 }
}

class VisitUserRoute extends ChildRoute {
 const VisitUserRoute() : super(route: 'visit-user/:userId');

  String generateNavRoute({required String root, required String
userId}) {
   return '$root/visit-user/$userId';
 }
}
```

assets.dart: This class maps string asset paths.

```
class Assets {
 Assets._();

 static const String assetsIconAppLogo = 'assets/icon/app_logo.png';
 static const String iconAppLogo = 'assets/icon/app_logo.png';
      static const String lottieEmptyBoxBlue =
'assets/lottie/empty-box-blue.json';
 static const String lottieFitness = 'assets/lottie/fitness.json';
      static const String lottiePersonWriting =
'assets/lottie/person_writing.json';

}
```

date_time_converter.dart: This a utility class to convert date times to json.

```
class DateTimeConverter {
 DateTimeConverter._();

 static DateTime? dateTimeFromJson(int? millisecondsSinceEpoch) {
   if (millisecondsSinceEpoch == null) {
     return null;
   }
```

```
                                                                    return
    DateTime.fromMillisecondsSinceEpoch(millisecondsSinceEpoch);
  }

  static int? dateTimeToJson(DateTime? timestamp) {
    if (timestamp == null) {
      return null;
    }

    return timestamp.millisecondsSinceEpoch;
  }
}
```

app_error.dart: A data class to show error in UI.

```
class AppError extends Equatable {
  final String message;

  @override
  List<Object?> get props => [message];

  const AppError({
    required this.message,
  });
}

class UserNotLoggedInError extends AppError {
   const UserNotLoggedInError() : super(message: "User not logged in
Error");
}

class LogoutFailedError extends AppError {
  const LogoutFailedError() : super(message: "Failed to logout");
}
```

```
class DataNotFoundError extends AppError {
  const DataNotFoundError() : super(message: "Requested Data not found");
}

class InvalidDataError extends AppError {
 const InvalidDataError() : super(message: "Invalid Data Error");
}

class UnknownError extends AppError {
 const UnknownError() : super(message: "Unknown Error");
}

class FileUploadError extends AppError {
 const FileUploadError() : super(message: "File Uploading Error");
}
```

app_user.dart: An user model class that represents a logged in user.

```
@freezed
class AppUser with _$AppUser {
 bool get isEmpty => userId.isEmpty;

 bool get isOnBoardingNotComplete =>
     gender.isEmpty ||
     birthdate == null ||
     availability.isEmpty ||
     nearbyDistance == -1;

 const AppUser._();

 const factory AppUser({
   required String userId,
```

```
      @JsonKey(fromJson:  DateTimeConverter.dateTimeFromJson,  toJson:
DateTimeConverter.dateTimeToJson)

        required DateTime? registered,

    @JsonKey(name: 'name') @Default('') String name,

    @JsonKey(name: 'gender') @Default('') String gender,

                        @JsonKey(name:       'birthdate',       fromJson:
DateTimeConverter.dateTimeFromJson,                                 toJson:
DateTimeConverter.dateTimeToJson)

        DateTime? birthdate,

                        @JsonKey(name:       'last_seen',       fromJson:
DateTimeConverter.dateTimeFromJson,                                 toJson:
DateTimeConverter.dateTimeToJson)

        DateTime? lastSeen,

    @JsonKey(name: 'availability') @Default('') String availability,

    @JsonKey(name: 'nearbyDistance') @Default(-1) int nearbyDistance,

    @JsonKey(name: 'email') @Default('') String email,

           @JsonKey(name:    'profilePicture')    @Default('')    String
profilePicture,

    @JsonKey(name: 'lat') @Default(0.0) double lat,

    @JsonKey(name: 'long') @Default(0.0) double long,

    @JsonKey(name: 'geoHash') @Default("") String geoHash,

         @JsonKey(name:   'interestsList')   @Default([])   List<String>
interestList,

    @JsonKey(name: 'is_hidden') @Default(false) bool isHidden,

  }) = _AppUser;


  factory AppUser.fromJson(Map<String, dynamic> json) =>

      _$AppUserFromJson(json);


  factory AppUser.empty() => AppUser(

        userId: '',

        registered: DateTime.now(),

        name: '',

        gender: '',

        birthdate: null,
```

```
            lastSeen: null,
            availability: '',
            nearbyDistance: 0,
            email: '',
            profilePicture: '',
            lat: 0.0,
            long: 0.0,
            geoHash: '',
            interestList: [],
            isHidden: false,
          );
      }
```

interest.dart: Interest model class.

```
@freezed
class Interest with _$Interest {
 factory Interest({
   @JsonKey(name: 'doc_id')
   required String id,
   required String icon,
   required String name,
 }) = _Interest;

 factory Interest.fromJson(Map<String, dynamic> json) =>
     _$InterestFromJson(json);
}
```

models.dart: Contains all the model to optimize imports.

```
part 'models.freezed.dart';
part 'models.g.dart';

part 'interest.dart';
part 'app_user.dart';
```

firebase_module.dart: Dependency injection module.

```
@module
abstract class FirebaseModule {
 GoogleSignIn get googleSignIn => GoogleSignIn();
 FirebaseAuth get firebaseAuth => FirebaseAuth.instance;
 FirebaseFirestore get firestore => FirebaseFirestore.instance;
 FirebaseStorage get firebaseStorage => FirebaseStorage.instance;
}
```

service_locator.dart: Service locator that maps object instances.

```
final GetIt sl = GetIt.instance;

@InjectableInit(initializerName: 'initGetIt', preferRelativeImports: true)
GetIt configureDependencies() => sl.initGetIt();
```

env.dart: This is a data container class for different environments.

```
enum EnvType { Development, Production }

class Env {
 static const defaultAppName = 'Workout Buddy Finder';
 static const defaultEnv = EnvType.Development;

 static const devAppName = 'Workout Buddy Finder Dev';
 static const devEnv = EnvType.Development;

 static const prodAppName = 'Workout Buddy Finder';
 static const prodEnv = EnvType.Production;

 final EnvType envType = defaultEnv;
 final String appName = defaultAppName;
```

```
 Future<void> init() async {
   final binding = WidgetsFlutterBinding.ensureInitialized();
   configureDependencies();
   await ThemeManager.instance.init();
 }

 Future<void> startApplication() async {
   final firebaseApp = await Firebase.initializeApp();
   runApp(WBFApp(envType: envType, appName: appName));
 }
}
```

env_development.dart: Development Environment.

```
void main() async {
 final env = DevelopmentEnv();
 await env.init();
 env.startApplication();
}

class DevelopmentEnv extends Env {
 @override
 EnvType get envType => Env.devEnv;

 @override
 String get appName => Env.devAppName;
}
```

env_production.dart: Production Environment.

```
void main() async {
 final env = ProductionEnv();
 await env.init();
 env.startApplication();
```

```
}

class ProductionEnv extends Env {
 @override
 EnvType get envType => Env.prodEnv;

 @override
 String get appName => Env.prodAppName;
}
```

auth_cubit.dart: State Management class to manage authorization.

```
@injectable
class AuthCubit extends Cubit<AuthState> {
 final SignInWithGoogleUseCase signInWithGoogleUseCase;
 final SignOutUseCase signOutUseCase;

 AuthCubit({
   required this.signInWithGoogleUseCase,
   required this.signOutUseCase,
 }) : super(const AuthState.initial());

 Future<void> signInWithGoogle(Position location) async {
   final result = await signInWithGoogleUseCase.execute(location);
   result.fold(
     (success) => null,
     (error) => emit(
       AuthState.signInFailure(error),
     ),
   );
 }

 Future<void> signOut() async {
```

```
    final result = await signOutUseCase.execute();
    result.fold(
      (success) => emit(const AuthState.signedOut()),
      (error) => emit(
        AuthState.signOutFailure(error),
      ),
    );
  }
}
```

sign_in_page.dart: Sign in page UI.

```
class SignInPage extends StatelessWidget {
 const SignInPage({Key? key}) : super(key: key);

 String get _logo => 'assets/icon/app_logo.png';

 Widget _slogan(BuildContext context) => Text(
       'Welcome to Fitness Buddy Finder',
       key: const ValueKey('Login Page Slogan'),
       style: Theme.of(context)
           .textTheme
           .titleLarge
           ?.copyWith(fontWeight: FontWeight.bold),
       textAlign: TextAlign.center,
     );

 Widget _subSlogan(BuildContext context) => Text(
       'Login to see nearby fitness enthusiasts',
          style:  Theme.of(context).textTheme.bodySmall?.copyWith(fontSize:
16),
     );

 @override
```

```
 Widget build(BuildContext context) {
   return Scaffold(
     body: SafeArea(
       child: Container(
         padding: const EdgeInsets.all(16),
         width: double.infinity,
         height: double.infinity,
         child: Column(
           mainAxisAlignment: MainAxisAlignment.center,
           mainAxisSize: MainAxisSize.min,
           crossAxisAlignment: CrossAxisAlignment.center,
           children: [
             Image.asset(_logo, height: 120, width: 120),
             const SizedBox(height: 20),
             _slogan(context),
             const SizedBox(height: 8),
             _subSlogan(context),
             const SizedBox(height: 60),
             const SignInWithGoogleButton()
           ],
         ),
       ),
     ),
   );
 }
}
```

location_retriver_dialog.dart: This dialog is displayed when location fetching action is on going

```
class LocationRetrieverDialog extends StatelessWidget {
 const LocationRetrieverDialog({Key? key}) : super(key: key);
 void _onLocationRetrieverBlocStateChange(
   BuildContext context,
```

```
    LocationRetrieverState state,
  ) async {
    if (state is LocationServiceDisabledState) {
      final result = await Geolocator.openLocationSettings();
    } else if (state is LocationPermissionGrantedState) {
      context.read<LocationRetrieverBloc>().add(LocationRetrieveEvent());
    } else if (state is LocationRetrievedState) {
      Navigator.of(context).pop(state.location);
    }
  }

  // We are handling multiple types here so dynamic is more suitable
  // ignore: avoid-dynamic
  Widget _buildUiForState(BuildContext _, LocationRetrieverState state) {
    if (state is LocationPermissionDeniedState) {
      return const Text(
            'We need your location to continue. Please allow location permission',
      );
    } else if (state is LocationPermissionDeniedForEverState) {
      return const Text(
        'We need your location to continue. Please allow location permission from Device Settings',
      );
    } else if (state is LocationServiceDisabledState) {
      return const Text(
            'We need your location to continue. Please enable Location Services',
      );
    }

    return const Text(
      'Please wait while we retrieve your location',
```

```
    );
  }

  @override
  Widget build(BuildContext context) {
    return BlocProvider<LocationRetrieverBloc>(
      create: (_) => sl()..add(LocationRetrievePermissionEvent()),
      child: BlocConsumer<LocationRetrieverBloc, LocationRetrieverState>(
        listener: _onLocationRetrieverBlocStateChange,
        builder: _buildUiForState,
      ),
    );
  }
}
```

chat_room_page.dart: UI of chat room page.

```
class ChatRoomPage extends HookWidget {
  const ChatRoomPage({
    Key? key,
    required this.chatRoomId,
    required this.onVisitProfile,
  }) : super(key: key);

  final String chatRoomId;
  final void Function(String userId) onVisitProfile;

  Future<void> _onSendMessage(
    AppUser loggedInUser,
    TextEditingController controller,
  ) async {
    final text = controller.text;
    controller.clear();
    await sl<SendMessageUseCase>().call(
```

```
      loggedInUser: loggedInUser,
      chatRoomId: chatRoomId,
      message: text,
    );
  }

  @override
  Widget build(BuildContext context) {
    final loggedInUser = context.read<AppUser>();
    final textController = useTextEditingController();

    return Column(
      children: [
        TopBar(chatRoomId: chatRoomId, onVisitProfile: onVisitProfile),
        Expanded(
          child: Padding(
            padding:
                                    const   EdgeInsets.symmetric(horizontal:
page_horizontal_spacing),
            child: ChatRoomMessagesList(chatRoomId: chatRoomId),
          ),
        ),
        Container(
          padding:
                                    const   EdgeInsets.symmetric(horizontal:
page_horizontal_spacing),
          height: 80,
          child: Row(children: [
            Expanded(child: MessageInput(controller: textController)),
            IconButton(
              onPressed: () => _onSendMessage(loggedInUser, textController),
              icon: const Icon(FontAwesomeIcons.paperPlane),
            ),
```

```
          ]),
        ),
      ],
    );
  }
}
```

profile_page.dart: UI of profile page.

```
class ProfilePage extends StatelessWidget {
 const ProfilePage({
   Key? key,
   required this.onEditProfile,
 }) : super(key: key);

 final Function() onEditProfile;

 @override
 Widget build(BuildContext context) {
   final appUser = context.read<AppUser>();

   return BlocProvider<ProfileBloc>(
     create: (_) => sl(),
     child: SafeArea(
       child: Padding(
         padding: const EdgeInsets.all(8.0),
         child: Stack(
           children: [
             SingleChildScrollView(
               child: Column(
                 mainAxisSize: MainAxisSize.min,
                 children: AnimateList(
                   effects: [
                     SlideEffect(
```

```
                duration: 300.milliseconds,
                begin: const Offset(1.0, 0.0),
                end: const Offset(0.0, 0.0),
              ),
            ],
            interval: 50.milliseconds,
            children: [
              UserInfoSection(appUser: appUser),
              const SizedBox(height: 8),
              UserPreferences(appUser: appUser),
              const SizedBox(height: 8),
              UserInterestSection(appUser: appUser),
              const SizedBox(height: 8),
              ActivityTrackerSection(appUser: appUser),
              const SizedBox(height: 8),
              OthersSection(appUser: appUser),
            ],
          ),
        ),
      ),
      Positioned(
        right: 0,
        bottom: 16,
        child: EditProfileButton(onEditProfile: onEditProfile),
      ),
    ],
  ),
),
),
);
}
```

settings_page.dart: UI of settings page.

```
class SettingsPage extends StatelessWidget {
 const SettingsPage({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return Column(
     children: [
       AppBar(title: const Text('Settings')),
       Padding(
         padding: const EdgeInsets.all(12.0),
         child: Column(
           children: const [
             ThemeSelectorView(),
           ],
         ),
       ),
     ],
   );
 }
}
```

suggestions_page.dart: UI of suggestions page.

```
class SuggestionsPage extends StatelessWidget {
 const SuggestionsPage({
   Key? key,
   required this.onShowProfile,
   required this.onOpenMessaging,
   required this.onOpenProfile,
 }) : super(key: key);

 final Function() onShowProfile;
 final void Function() onOpenMessaging;
```

```
  final void Function(String userId) onOpenProfile;

  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        AppBar(title: Intro(onTap: onShowProfile)),
        Expanded(
          child: Padding(
            padding: const EdgeInsets.symmetric(
              horizontal: page_horizontal_spacing,
              vertical: page_vertical_spacing,
            ),
            child: Column(
              children: [
                const SizedBox(height: 8),
                Expanded(
                  child: SuggestionCardContainer(
                    onOpenMessaging: onOpenMessaging,
                    onOpenProfile: onOpenProfile,
                  ),
                ),
              ],
            ),
          ),
        ),
      ],
    );
  }
}
```