

WEIGHT MANAGEMENT

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in Information Technology

by

LEOSHARLIN JOHNSON(41120100)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category-1 University by UGC

Accredited with Grade “A++” by NAAC

12B Status by UGC |Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MAY 2024



SATHYABAMA

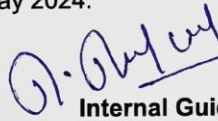
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **LEOSHARLIN JOHNSON(41120100)** who carried out the project entitled "WEIGHT MANAGEMENT" under my supervision from January 2024 to May 2024.

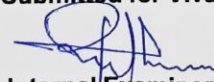

8/5/24
Internal Guide

Ms.S.SageenGrana, M.E.,


Head of the Department

Dr.R. Subhashini, M.E., Ph.D.,

Submitted for Viva voce Examination held on 08/05/24


Internal Examiner


8/5/24
External Examiner

DECLARATION

I **Leosharlin Johnson** hereby declare that the Project Report entitled "**WEIGHT MANAGEMENT**" done by me under the guidance of is **Ms.SageenGrana,M.E.**, submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 08/05/24

PLACE: CHENNAI

Leosharlin
SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing **Dr.R.Subhashini M.E.,Ph.D.,**Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.SageenGrana,M.E.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

CERTIFICATE OF COMPLETION



149, 1C, 2nd Floor, Mount Poonamallee Rd,
Ramapuram, Chennai - 600089.
admin@hackwittechnologies.com
www.hackwittechnologies.com

10.02.2024

CERTIFICATE FOR INTERNSHIP

TO WHOMSOEVER IT MAY CONCERN

This is to Certify that **Leo Sharlin Johnson (Reg No: 41120100)** student of Sathyabama Institute of Science and Technology, B. Tech Information Technology, has successfully completed the Internship at Hackwit Technologies from 22nd January to 30th January 2024.

During the internship, **Leo Sharlin Johnson** has closely worked as a part of the **Flutter** by carrying out the project entitled "**Weight Management**".

He demonstrated good developing and learning skill with self-motivated attitude to learn new things. His contribution to implement the Flutter was good. We wish him all the best for his future endeavors.

Best Regards,

A handwritten signature in black ink, appearing to read 'Ravikumar S', is placed above the printed name.

Ravikumar S
CEO & Managing Director
Hackwit Technologies Pvt Ltd



CERTIFICATE OF COMPLETION



ABSTRACT

The Weight Management is a simple yet widely used tool in assessing an individual's body composition and potential health risks associated with weight. It provides a numerical value derived from a person's weight and height, offering a rough estimate of their body fat percentage. This calculation serves as a general indicator of whether an individual falls within a healthy weight range or if they may be at risk of health issues due to being underweight, overweight, or obese. The formula for calculating Weight Management is relatively straightforward: it involves dividing an individual's weight in kilograms by the square of their height in meters. The resulting number categorizes individuals into different ranges, each associated with varying degrees of body fat and potential health implications. While the calculation is a valuable initial screening tool, it's important to recognize its limitations. Weight Management does not directly measure body fat percentage or account for factors such as muscle mass, bone density, or distribution of fat throughout the body. Therefore, individuals with higher muscle mass, such as athletes or bodybuilders, may have a higher score despite having a healthy body composition. Moreover, it does not differentiate between different types of body fat, such as visceral fat (fat around the organs) and subcutaneous fat (fat under the skin). Visceral fat, in particular, is associated with increased health risks, including cardiovascular disease and type 2 diabetes, even in individuals with a normal Weight Management. Therefore, relying solely on it to assess health risks may not provide a comprehensive picture of an individual's overall health status. Despite its limitations, it remains a widely used tool due to its simplicity and accessibility. It can help healthcare professionals quickly identify individuals who may be at risk of obesity-related health issues and guide further evaluation and interventions.

.TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	i
	LIST OF ABBREVIATIONS	iv
	LIST OF FIGURES	v
1	INTRODUCTION	1
1.1	Flutter	2
1.2	Dart	3
1.3	Firebase	4
1.4	System Overview	5
1.5	Problem Statement	6
1.6	Features and Function	7
	1.6.1 Features	7
	1.6.2 Function	8
2	METHODOLOGY	9
2.1	Initial Project Requirements	9
2.2	Flutter Application Architecture	12
2.3	Firebase Intergration	15
2.4	Firebase	16
2.5	Flow Diagram	17
2.6	Modules	19
	2.6.1 Authentication module	19
	2.6.2 Notification module	19

	2.6.3	Profile module	20
	2.6.4	Game details module	21
	2.7	Application algorithm	22
	2.8	Requiriment analysis	24
	2.9	Development and deployment setup	25
3		RESULTS AND DISCUSSION	30
4		CONCLUSION AND FUTURE WORK	34
	4.1	Conclusion	34
	4.2	Future Work	35
		REFERENCES	37
		APPENDICES	38
	A.	Sample Code	38
	B.	Screen Shots	40

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
UI	- User Interface
UX	- User Experience
iOS	- iPhone Operating System
UAT	- User Acceptance Testing
IDE	- Integrated Development Environment
SDK	- Software Development Kit
PUBSEC YAML	-File is configuration File using Dart Manager

LIST OF FIGURES

Figure No.	Name of the Figure	Page No.
2.1	Flutter Application Architecture	12
2.2	Firebase Intergration	15
2.3	Firebase	16
2.4	Flow Diagram	17
2.5	Project Creation	25
2.6	Firebase Connection	26
2.7	Firebase Authentication	27
2.8	Deployment process	28
2.9	SDK	29
3.1	Main Dart	30
3.3	Firebase core	31
3.4	Firebase configuration	32
3.5	Initializing firebase	33

CHAPTER 1

INTRODUCTION

The overweight (Weight Management for age and sex from the 85th percentile to the 95th percentile) and obesity (Weight Management I for age and sex greater than or adequate to the 95th percentile) are related to the present and future health of youth also as their academic success. Youth classified as overweight or obese are more likely to be diagnosed with prediabetes (impaired fasting glucose or impaired glucose tolerance) and have more cardio metabolic risk factors than normal weight youth. Various methods for assessing body composition, used for children and adolescents in research settings (for example, hydrodensitometry, plethysmography of air displacement, isotope dilution, two-energy X-ray absorptiometry). The use of these methods in a community setting is limited. As a result, measuring height and weight to calculate Weight Management is recommended for overweight/obese people in screening for the younger generation. $(\text{body weight (kg)} / \text{height (square meters)})$. Android is used in over 190 countries and powers hundreds of millions of mobile devices. It has the largest installed base of any mobile platform and is rapidly expanding—every day, a million new Android users turn on their devices for the first time and begin searching for apps, games, and other digital content. Android provides you with a world-class platform for developing apps and games for Android users around the world, as well as an open marketplace for quickly distributing them.

1.1. FLUTTER

Creating a Weight Management calculator in Flutter involves several key objectives:

1. **User Interface Design:** Designing an intuitive and user-friendly interface for inputting height and weight values.
2. **Data Input Handling:** Implementing mechanisms to accept user input for height and weight, potentially including validation to ensure the input is within appropriate ranges.
3. **Weight Management Calculation Logic:** Developing the logic to calculate the Weight Management based on the provided height and weight inputs. This involves applying the Weight Management formula: $\text{Weight Management} = \text{weight (kg)} / (\text{height (m)})^2$.
4. **Displaying Results:** Presenting the calculated Weight Management value to the user in a clear and understandable format, along with an interpretation of what the Weight Management value signifies in terms of health.
5. **Styling and Theming:** Applying appropriate styling and theming to ensure consistency and aesthetics throughout the application.
6. **Error Handling:** Implementing error handling mechanisms to manage potential errors or edge cases, such as dividing by zero or invalid input.
7. **Unit Conversion:** Optionally, providing functionality to allow users to input height and weight in different units (e.g., feet/inches and pounds, or centimeters and kilograms) and converting them to the required units for Weight Management calculation.
8. **Testing:** Conducting thorough testing to ensure the calculator functions correctly under various scenarios and edge cases.

By addressing these objectives, you can create an effective Weight Management calculator application in Flutter.

1.2. DART

i) Unified Codebase:

Backpackers Story relies on Dart, a client-optimized programming language, for implementing backend logic and server-side operations. Dart's versatility enables developers to maintain a unified codebase for both the frontend (Flutter) and backend, fostering consistency and ease of collaboration.

i) Asynchronous Programming:

Dart's support for asynchronous programming is crucial for handling concurrent operations efficiently. This feature ensures that Backpackers Story can seamlessly manage multiple tasks, such as handling user requests, database operations, and external API calls, without causing delays or disruptions in the application's responsiveness.

ii) Scalability:

Dart's scalability is instrumental in accommodating the potential growth of BMI. As the user base expands and the demand for managers increases, Dart's ability to scale allows the platform to handle higher traffic, ensuring a smooth and responsive experience for all users.

iii) Strong Typing System:

Dart's strong typing system enhances code reliability and maintainability in BMI. By catching potential errors during the development phase, Dart enables developers to write robust and error-resistant code, reducing the likelihood of runtime issues and enhancing the overall stability of the application.

1.3 FIREBASE

1. Initialize Firebase in Dart Code:

In your Dart code, import the necessary Firebase packages such as `firebase_core` for Firebase Core functionality. Initialize Firebase in your app's main function or at the beginning of your app's lifecycle using `Firebase.initializeApp()`.

2. Set up Firebase Database:

Create a Firebase Realtime Database or Firestore database in your Firebase Console to store game discounts data. Define the structure of your database, such as nodes or collections for different types of discounts.

3. Read and Write Data to Firebase Database:

Use Firebase SDK's database functionalities to read and write data to your Firebase database from your Flutter app. Implement functions to fetch game discounts data from Firebase and display it in your app's UI.

4. Authentication:

If you require user authentication or personalized discount data, integrate Firebase Authentication into your app. Set up user authentication methods such as email/password, Google Sign-In, or other authentication providers supported by Firebase.

5. Security Rules (Database Security):

Configure security rules for your Firebase database to control access and ensure data security. Define rules to allow read/write access only to authenticated users or specific user roles as per your app's required

1.4 system overview:

In a BMI (Body Mass Index) calculator developed using Flutter, the system overview typically involves several key components:

1. User Interface(U):

- The UI is developed using Flutter widgets to create a user-friendly interface.
- It includes input fields for the user to enter their height and weight.
- Buttons are provided for the user to submit their data and calculate their BMI.

2. Business Logic Layer:

- This layer handles the calculation of BMI based on the user's input.
- It may include functions to validate user input and perform the BMI calculation using the formula: $BMI = \text{weight (kg)} / (\text{height (m)} * \text{height (m)})$.

3. Data Handling:

- Processes and manages user input data with validation to ensure correctness.
- Convert input values between metric and imperial units if necessary.

4. Weight Management Calculation:

- Validates user input and calculates the Weight Management using provided data.

5. Output Display:

- Displays the calculated Weight Management value to the user clearly.
- Additional information, such as categorization based on the Weight Management, may be provided with associated health implications

6. Integration with Flutter Framework:

- Integrated into the Flutter framework, utilizing its features and functionalities for building cross-platform mobile applications.
- Utilizes Flutter widgets for UI components and Flutter's reactive framework for updating UI based on user input and calculated results

1.5 PROBLEM STATEMENT:

In the dynamic world of gaming, avid gamers are constantly seeking opportunities to access their favorite games at discounted prices. However, the process of manually scouring multiple platforms for ongoing discounts is time-consuming and often leads to missed A BMI (Body Mass Index) calculator is a useful tool for assessing a person's body weight in relation to their height. The problem statement for creating such a calculator typically involves designing a program that takes inputs of a person's weight (in kilograms) and height (in meters) and then calculates their BMI using the formula:

$$BMI = \frac{weight}{height^2}$$

Once the BMI is calculated, it's often categorized into different ranges to provide insight into an individual's weight status. These categories can include underweight, normal weight, overweight, and obese. The program should then output the calculated BMI along with the corresponding weight status category. To make the BMI calculator more user-friendly, you may also include input validation to ensure that the user enters valid numeric values for weight and height. Additionally, providing clear instructions and possibly a brief explanation of BMI and its significance can enhance the user experience.

1.6 Features and Function:

To create a BMI calculator app in Flutter, you'll need to define both the features and functionality. Here's a basic outline:

1.6.1 Features

- **Calculate :** Allow users to input their height and weight, then calculate their Weight Management using the formula: $\text{Weight Management} = \frac{\text{weight(kg)}}{(\text{height(m)} * \text{height(m)})}$.
- **Categories:** Display the user's BMI category (Underweight, Normal weight, Overweight, or Obese) based on the calculated Index value.
- **Metric/Imperial Units:** Provide an option for users to input their height and weight in either metric (centimeters/kilograms) or imperial (inches/pounds) units.
- **Interactive UI:** Design a user-friendly interface with input fields, buttons, and clear labels to guide users through the process.
- **Feedback:** Provide feedback to the user about their Weight Management category, possibly with color-coded indicators or descriptive text.
- **Reset:** Include a button to reset the input fields and clear the calculated result.

1.6.2.Function:

- **Input Validation:** Ensure that the user cannot input negative values or leave the input fields blank.
- **Unit Conversion:** Convert the user's input between metric and imperial units as necessary for calculations.
- **Index Calculation:** Implement the Weight Management formula to calculate the Index based on the user's input.
- **Weight Management Category Determination:** Based on the calculated Index value, determine the corresponding Weight Management category and display it to the user.
- **Responsive Design:** Make sure the app layout is responsive to different screen sizes and orientations.
- **Localization:** Support multiple languages if needed, especially for units of measurement and Weight Management category labels.

Implementing these features and functionality will create a comprehensive BMI calculator app in Flutter. You can start by setting up the UI with input fields for height and weight, along with buttons for unit selection and calculation. Then, implement the logic for BMI calculation and category determination.

CHAPTER 2

METHODOLOGY

2.1. INITIAL PROJECT REQUIREMENTS

The initial project requirements for BMI encompass various aspects such as Project management, technologies, Non-Functional requirements and Functional requirement

AIM:

The aim of this project is to create where people can easily target their weight easily efficiently with each other and provide ease and comfort to the people. This is an one handed application users can carried out by this way is as simple. Doesn't need any multiple or mathematical calculation. The feature aims to eliminate paper based calculations or some weighing machines. Users can check their body mass at any time and any places with this single application. This online application is very simple to calculate by just entering their weight and height

SCOPE:

The scope of a Weight Management application developed using Flutter encompasses various aspects, including its functionality, features, target audience, and potential extensions. Here's a brief overview:

1. Functionality:

- The primary functionality of the Weight Management is to calculate the Weight Management based on user input for height and weight.
- It categorizes the calculated Index into predefined categories such as Underweight, Normal weight, Overweight, or Obese.
- It provides users with the option to input their data in either metric (centimeters/kilograms) or imperial (inches/pounds) units.

- The application should offer a reset option to clear input fields and recalculated Weight Management results.

2. Features:

- Interactive UI: Design a user-friendly interface with intuitive input fields, buttons, and clear labels to guide users through the process.
- Feedback: Provide feedback to users about their Weight Management category, potentially using color-coded indicators or descriptive text .
- Metric/Imperial Units: Implement functionality to allow users to toggle between metric and imperial units seamlessly.
- Reset: Include a button to reset input fields and clear the calculated Weight Management result for convenience.

3. Target Audience:

- The target audience for the Weight Management calculator includes individuals interested in monitoring their weight and overall health.
- It can cater to a wide range of users, including health-conscious individuals, fitness enthusiasts, and healthcare professionals.

4. Extensions:

- User Profiles: Allow users to create profiles to track their Weight Management over time and set goals.
- Integration with Health Data: Integrate with platforms or APIs to sync data with health tracking apps or devices.
- Additional Health Metrics: Expand the application to calculate and track other health metrics such as body fat percentage, waist-to-hip ratio, etc.
- Educational Resources: Provide educational resources or links to articles about healthy weight management and lifestyle choices.

Overall, the scope of the Weight Management application in Flutter involves providing users with a simple yet effective tool for calculating and monitoring their BMI, with potential extensions to enhance user experience and offer additional health-related features.

Packages Used:

i) Get:

Version: 4.6.6

Purpose: State management and navigation framework.

ii) Firebase Core:

Version: 2.25.3

Purpose: Firebase initialization and core functionality.

iii) Firebase Auth:

Version: 4.17.3

Purpose: User authentication with Firebase.

iv) Cloud Firestore:

Version: 4.15.3

Purpose: Interaction with Cloud Firestore for real-time data

v) Firebase Core Web:

Version: 2.11.3

Purpose: Web implementation of Firebase Core

vi) Toast:

Version: 0.3.0

Purpose: Displaying simple toast notifications.

vii) FlutterToast:

Version: 8.0.7

Purpose: Displaying customizable toast messages.

2.2.FLUTTER APPLICATION ARCHITECTURE

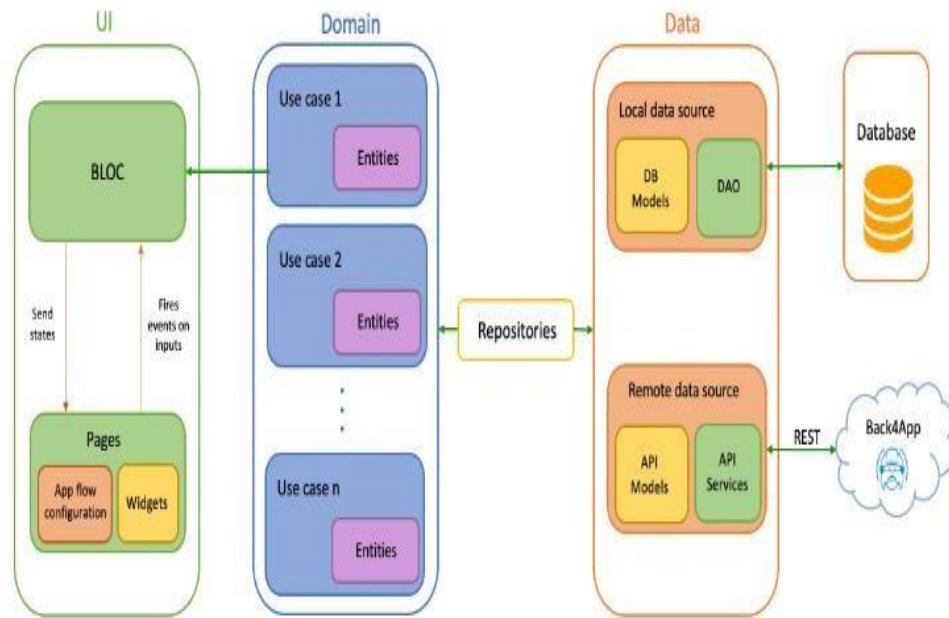


Fig:2.1: Flutter Application Architecture

This article is intended to provide a high-level overview of the architecture of Flutter, including the core principles and concepts that form its design. Flutter is a cross-platform UI toolkit that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services. The goal is to enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible.

User Interface (UI) Layer:

Flutter Widgets: Use Flutter widgets to create the app's user interface, including screens for displaying discounted games, settings for user preferences, and notifications.

Business Logic Layer:

Dart Classes: Implement Dart classes for managing business logic, such as fetching data from the Game Discount API, identifying eligible games based on discount criteria, and handling user preferences.

State Management: Utilize Flutter's state management solutions like Provider or Riverpod for managing app state and data flow between UI components and business logic.

Data Layer:

External APIs Integration: Integrate with external APIs (e.g., Game Discount API) to fetch real-time data about discounted games.

Firebase Integration (Optional): Integrate with Firebase Firestore or Realtime Database for storing user preferences, managing notifications, and handling real-time updates.

Notification System:

Local Notifications: Implement Flutter's local notification system to notify users about eligible game discounts within the app.

Push Notifications (Optional): Integrate Firebase Cloud Messaging (FCM) for push notifications to notify users about discounts even when the app is not in use.

Testing and Debugging:

Flutter Testing Frameworks: Use Flutter's testing frameworks like Flutter Test or integration testing for testing app functionality, UI interactions, and business logic.

Debugging Tools:

Utilize Flutter's debugging tools for identifying and fixing issues during app development.

Security and Authentication :

Firebase Authentication: Implement Firebase Authentication for user authentication and personalized experiences based on user accounts.

Secure API Integration:

Ensure secure communication with external APIs using HTTPS and API keys for authentication and authorization.

Optimization and Performance:

Code Optimization: Optimize code structure, minimize unnecessary network requests, and use efficient data caching mechanisms for improved performance.

UI/UX Optimization:

Design a responsive and intuitive user interface for seamless navigation and user interaction.

Documentation and Version Control:

GitHub Repository: Host the app's codebase on GitHub for version control, collaboration, and documentation.

README.md:

Create a detailed README.md file in the GitHub repository with information about the app's architecture, dependencies, setup instructions, and usage guidelines.

2.3.FIREBASE INTERGRATION:

i.Firebase Authentication is used for user authentication, ensuring that only authenticated users can access the app's features.

ii.Cloud Firestore is utilized to store event data such as titles, descriptions, dates, and poster URLs. Each event might be represented as a document in a Firestore collection.

iii.Firebase Storage stores the event posters uploaded by users. Upon upload, the `downloadURL()` function is called to obtain the URL of the stored poster, which is then saved along with other event details in Firestore.

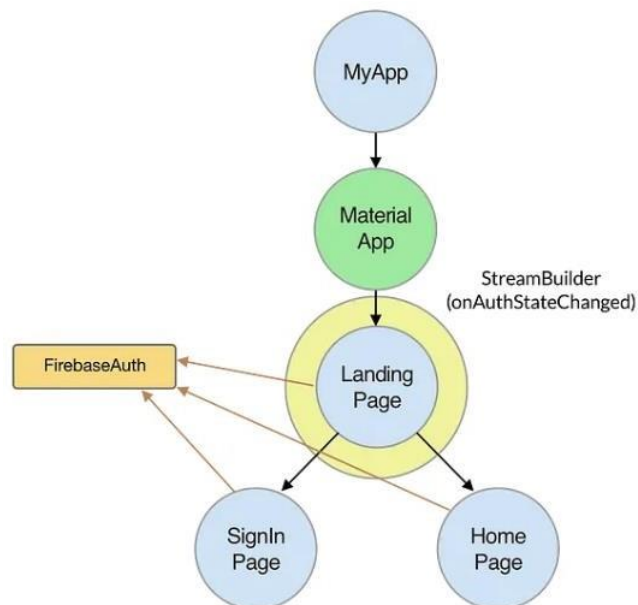


Fig 2.2:Firebase Intergration

2.4.FIREBASE :

Firestore:

Firestore: Use Firestore to store game data, such as titles, descriptions, discounts, and eligibility criteria. Organize data into collections and documents based on game categories or genres.

Firebase Cloud Messaging (FCM): Integrate FCM for push notifications. Set up notification topics or tokens to target specific users or groups for discount notifications.

Authentication:

Firebase Authentication: Implement Firebase Authentication for user accounts if personalized experiences or user-specific preferences are required. This can be used to manage user accounts and ensure secure access to app features.

App Integration:

Flutter Firebase SDK: Integrate the Flutter Firebase SDK to connect your app with Firebase services. Use packages like `cloud_firestore` for Firestore integration and `firebase_messaging` for FCM integration.



Fig 2.3: Firebase

2.5 FLOW DIAGRAM:

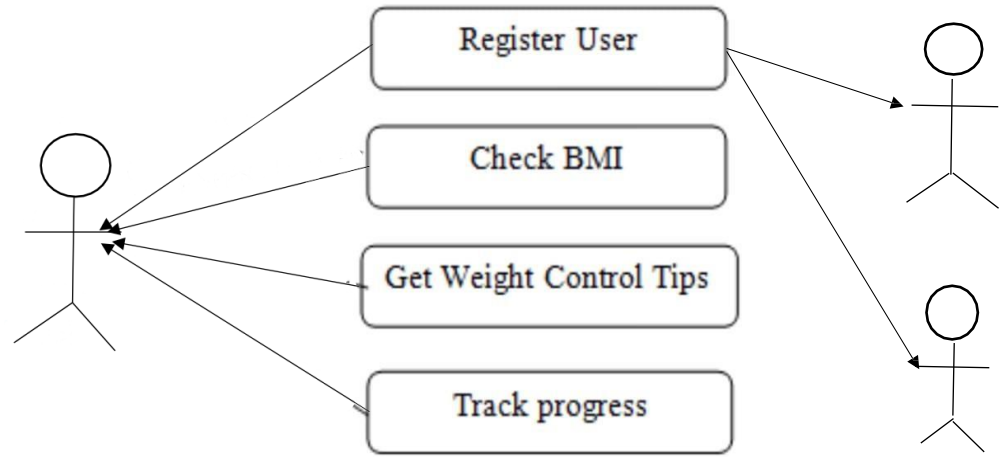


Fig:2.4: Flow Diagram

The following steps are used to describe the above flow diagram

Start: The user opens the app, initiating the authentication check.

Actor: User:

- **Use Cases:**

- 1) **Calculate Index:** This represents the primary functionality of the Weight Management where users input their height and weight to compute their Index.
- 2) **View Result:** After entering height and weight, users can view the calculated value.
- 3) **Interpret:** Provides users with an interpretation of their Index value, indicating whether it's underweight, normal weight, overweight, or obese.
- 4) **Save Record:** Users may have the option to save their records for future reference or tracking purposes.

Actor: Admin:

- **Use Cases**

- 1) **Manage Users:** If the system includes user accounts, an admin might have the capability to manage user accounts, including adding, deleting, or modifying user information.

Actor: External System:

- **Use Cases**

Data Storage:

Interaction with an external system for storing user data, including Weight Management records, if the application incorporates cloud storage or a database.

2.6. MODULES

2.6.1 Authentication Module:

Handles user authentication and authorization. Includes screens for registration, login, password reset, and social login options. Integrates Firebase Authentication for secure user management.

Algorithm

The Authentication Module in our Flutter app plays a critical role in managing user authentication and authorization seamlessly. When a user attempts to log in, the system validates their provided email and password format, ensuring they meet the required criteria. Utilizing Firebase Authentication's `signInWithEmailAndPassword` method, the app authenticates the user's credentials against Firebase's secure authentication system. Upon successful authentication, the user is redirected to the app's main screen, where they can access their personalized content and features

2.6.2 Notifications Module:

Manages push notifications to alert users about new discounts, wishlist updates, or personalized recommendations. Integrates Firebase Cloud Messaging (FCM) for sending push notifications to users' devices. Includes settings screen for users to customize notification preferences.

Algorithm

The Notifications Module in our Flutter app is designed to provide users with timely and relevant notifications regarding discounted games. It begins by fetching real-time data about discounts from external sources, applying user preferences and filters to identify

games that match their interests, such as preferred genres and desired discount percentages.

To ensure that users stay updated, the module schedules periodic checks for new discounts, running background tasks at regular intervals. When new discounts are detected, the system generates notification messages containing essential details like the game title, discount percentage, and the duration of the offer. Action buttons or deep links within the notifications allow users to directly access the discounted games within the app.

2.6.3 Profile Module:

Displays user profile information and settings. Allows users to edit their profile, change password, and manage account preferences. Integrates with Firebase Authentication to retrieve user profile data.

Algorithm

Profile Module within our Flutter app serves as a central hub for managing user profile information, providing users with a personalized and customizable experience. When users create or update their profiles, the module initiates a validation process to ensure accuracy and completeness of input data, including essential details like name, email, and optional information such as profile pictures or interests. This information is securely stored in Firebase Firestore or Realtime Database, associated with the user's unique identifier (UID) obtained from Firebase Authentication.

In developing a Weight Management Application using Flutter, several design considerations are important for creating an intuitive and user-friendly application. Here are some key design elements commonly used in such applications:

- **Responsive Design:** Ensure that the UI layout adjusts seamlessly across different screen sizes and orientations, providing a consistent user experience on various devices.
- **User Input Validation:** Implement validation mechanisms to ensure that user input for height and weight is within valid ranges and of appropriate data types. This helps prevent calculation errors and enhances user experience by providing immediate feedback on incorrect inputs.
- **Interpretation of Results:** Include features to interpret the result and provide users with understandable feedback on their weight status (e.g., underweight, normal weight, overweight, or obese). This can be achieved through text descriptions or visual indicators, such as color-coded categories.
- **Accessibility:** Ensure accessibility features are incorporated into the app, such as support for screen readers, scalable text, and high contrast modes. This ensures that the app is usable by individuals with diverse abilities.
- **Localization:** Support multiple languages and regions to cater to a global audience. Allow users to switch between different language options to improve accessibility and usability for non-English speakers.
- **Performance Optimization:** Optimize app performance by minimizing unnecessary computations and ensuring smooth rendering of UI elements. Consider lazy loading techniques for resource-intensive operations to enhance app responsiveness.

2.7 APPLICATION ALGORITHM

In a game discount app using Flutter, algorithms can be applied in various ways to enhance user experience, optimize functionality, and provide personalized recommendations. Here are some key areas where algorithms can be applied:

1. Recommendation Algorithm:

Implement a recommendation algorithm to suggest discounted games to users based on their preferences, browsing history, purchase history, and interactions with the app. Utilize machine learning techniques such as collaborative filtering, content-based filtering, or hybrid approaches to generate personalized recommendations. Continuously update and refine the recommendation model based on user feedback and behavior to improve accuracy and relevance.

2. Search Algorithm:

Develop a search algorithm to enable users to find specific games or browse by genre, platform, price range, or other criteria. Implement efficient search algorithms such as binary search, linear search, or trie-based search to quickly retrieve relevant game data from the app's database. Incorporate autocomplete suggestions and predictive text to assist users in formulating their search queries and discovering relevant games more effectively.

3. Sorting Algorithm:

Apply sorting algorithms to arrange discounted games on the app's interface based on various factors such as discount percentage, price, release date, popularity, or user ratings. Utilize sorting algorithms like quicksort, mergesort, or radix sort to efficiently organize game listings and improve user navigation and browsing experience.

4 Notification Algorithm:

Develop an algorithm to determine when to send push notifications to users about new discounts, wishlist updates, or personalized recommendations. Implement scheduling algorithms to determine the optimal timing for sending notifications based on user engagement patterns, time zones, and preferences. Consider factors such as user activity, frequency of notifications, and relevance of content to prioritize and personalize notifications effectively.

5. Data Analysis Algorithm: Utilize data analysis algorithms to analyze user behavior, preferences, and interactions within the app. Apply algorithms for data mining, pattern recognition, and statistical analysis to extract insights from user data and identify trends, correlations, and opportunities for optimization. Use clustering algorithms like k-means or hierarchical clustering to segment users into distinct groups based on their behavior and preferences, enabling targeted marketing and personalized experiences.

2.8. REQUIREMENT ANALYSIS

Requirement analysis is a critical phase in the development of any software project, including a game discount app using Flutter. Here's how you can conduct requirement analysis effectively:

1. Identify Stakeholders: Determine the key stakeholders involved in the project, such as clients, end-users, developers, designers, and marketing teams. Understand their roles, responsibilities, and expectations regarding the game discount app.

2. Gather Requirements:

Conduct interviews, workshops, or surveys with stakeholders to gather their requirements and preferences for the app. Document the functional requirements (features, functionalities, user interactions) and non-functional requirements (performance, security, scalability) of the app.

3. Define Use Cases:

Identify the different scenarios in which users will interact with the app. Define use cases to describe the specific actions users will perform and the expected outcomes.

4. Prioritize Requirements:

Prioritize requirements based on their importance, urgency, and impact on the overall project goals. Use techniques such as MoSCoW (Must-Have, Should-Have, Could-Have, Won't-Have) to categorize and prioritize requirements.

5. Clarify Ambiguities:

Clarify any ambiguities or inconsistencies in the requirements through discussions with stakeholders. Ensure that all requirements are clear, complete, and unambiguous to avoid misunderstandings during development.

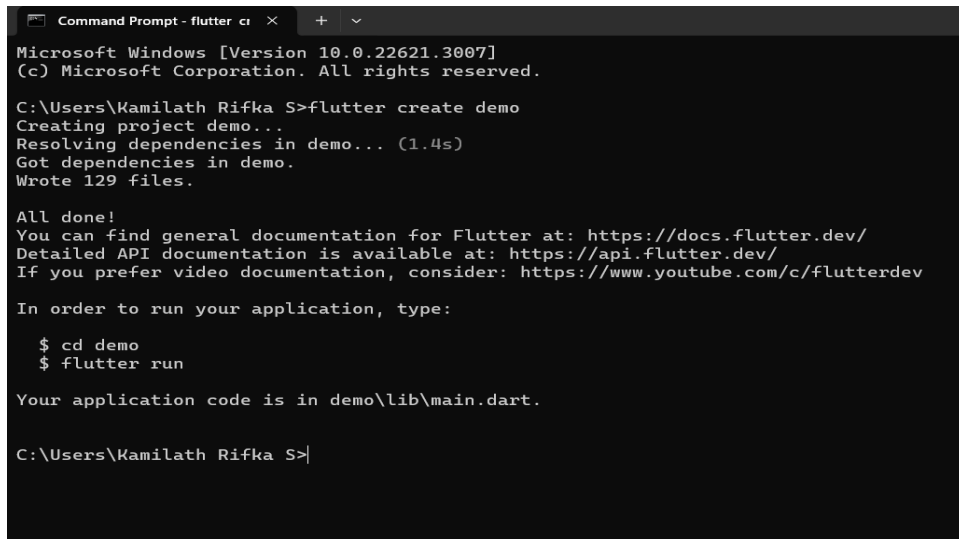
2.9. DEVELOPMENT AND DEPLOYMENT SETUP

1. Development Environment:

Flutter and Dart need to be installed on the developer's machine.
IDEs like Visual Studio Code or Android Studio for coding.

2. Create Project:

>>flutter create weight management



```
Command Prompt - flutter ci x + v
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Kamilath Rifka S>flutter create demo
Creating project demo...
Resolving dependencies in demo... (1.4s)
Got dependencies in demo.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:

$ cd demo
$ flutter run

Your application code is in demo\lib\main.dart.

C:\Users\Kamilath Rifka S>
```

Fig:2.5: Project Creation

3. Dependency Management:

Project dependencies specified in the pubspec.yaml file. Key packages include:

- firebase_core for Firebase integration.
- firebase_auth for user authentication.
- cloud_firestore for real-time data synchronization.
- Other packages like get, toast, image_picker, etc., for additional functionalities.

4. Dependency Management:

Project dependencies specified in the pubspec.yaml file. Key packages include:

- firebase_core for Firebase integration.
- firebase_auth for user authentication.
- cloud_firestore for real-time data synchronization.
- Other packages like get, toast, image_picker, etc., for additional functionalities.

5. Firebase Configuration:

Firebase project created and configured for the application.

Firebase Hosting used for deploying the Flutter web application.

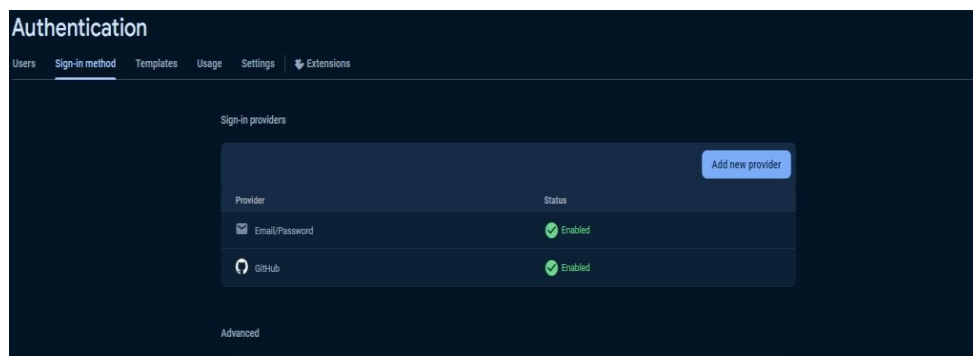


Fig:2.6: Firebase Connection

6 Firebase Authentication:

Firebase Authentication set up for secure user login and identity verification.

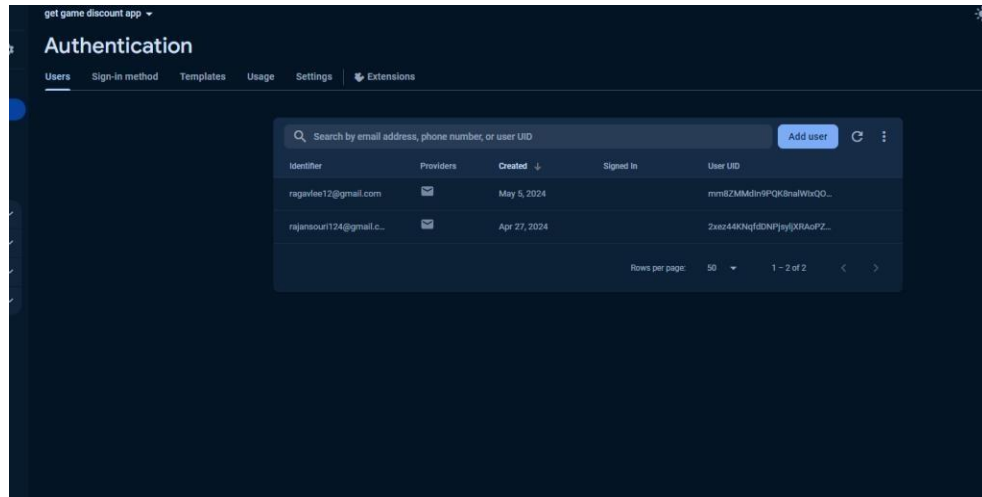


Fig:2.7: Firebase Authentication

6. Development Tools:

Flutter commands used for testing and building the application. Real-time testing on Android and iOS devices. Use clustering algorithms like k-means or hierarchical clustering to segment users into distinct groups based on their behavior and preferences, enabling targeted marketing and personalized experiences.

7. Deployment Process:

- i. Firebase used for hosting the web version of the application. ii
- .Seamless deployment using Flutter and Firebase commands.

```
# This is a generated file; do not edit or check into version control.
firebase_core=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\firebase_core-2.22.0\
firebase_core_web=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\firebase_core_web-2.8.1\
firebase_messaging=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\firebase_messaging-14.7.4\
firebase_messaging_web=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\firebase_messaging_web-3.5.13\
flutter_local_notifications=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\flutter_local_notifications-15.1.2\
flutter_native_splash=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\flutter_native_splash-2.3.5\
path_provider=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\path_provider-2.1.1\
path_provider_android=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\path_provider_android-2.2.1\
path_provider_foundation=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\path_provider_foundation-2.3.1\
path_provider_linux=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\path_provider_linux-2.2.1\
path_provider_windows=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\path_provider_windows-2.2.1\
shared_preferences=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences-2.2.2\
shared_preferences_android=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences_android-2.2.1\
shared_preferences_foundation=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences_foundation-2.3.4\
shared_preferences_linux=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences_linux-2.3.2\
shared_preferences_web=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences_web-2.2.1\
shared_preferences_windows=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\shared_preferences_windows-2.3.2\
sqflite=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\sqflite-2.3.0\
url_launcher=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher-6.2.1\
url_launcher_android=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_android-6.2.0\
url_launcher_ios=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_ios-6.2.0\
url_launcher_linux=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_linux-3.1.0\
url_launcher_macos=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_macos-3.1.0\
url_launcher_web=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_web-2.2.0\
url_launcher_windows=C:\Users\RagavLee\AppData\Local\Pub\Cache\hosted\pub.dev\url_launcher_windows-3.1.0\
```

Fig:2.8.deployment process

8. Supported Platforms:

Cross-platform support for Android, iOS, and web.

9. Integrated Development Environment (IDE):

Developers can choose between Visual Studio Code or Android Studio for coding

10. SDK:

1. Flutter SDK:The Flutter SDK receives regular updates, with new releases typically occurring every few weeks.As of my last update, the latest stable version of Flutter was Flutter 2.8, which included various improvements and features.

2.Firebase SDK:The Firebase Flutter SDK is regularly updated to add new features, improve performance, and ensure compatibility with the latest versions of Flutter and Firebase services.

3. Android SDK (if developing for Android): The Android SDK versions required for a Flutter app can vary based on the minimum and target Android API levels specified in your app's configuration.

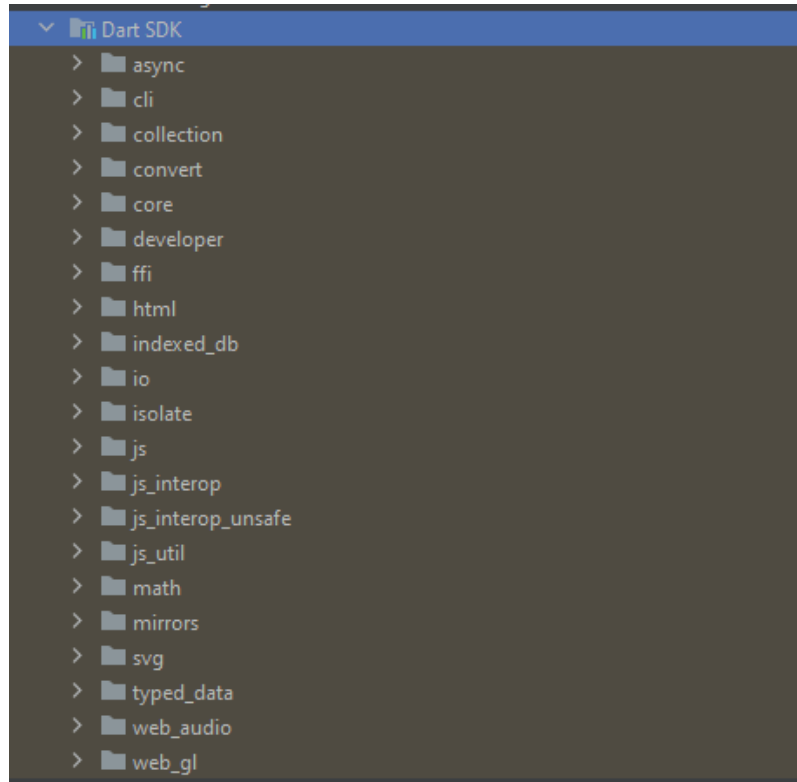


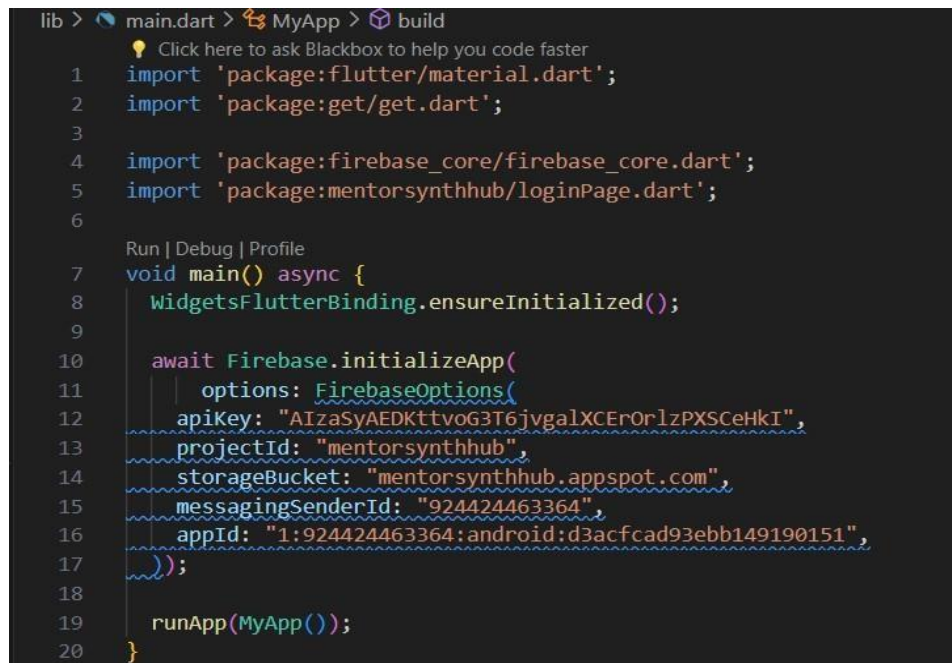
Fig:2.9.SDK

CHAPTER 3

RESULTS AND DISCUSSION

DART CODES:

1.main.dart: This file is typically the entry point of a Flutter application. It initializes the app, sets up necessary configurations, and runs the main application widget.



```
lib > main.dart > MyApp > build
  Click here to ask Blackbox to help you code faster
1  import 'package:flutter/material.dart';
2  import 'package:get/get.dart';
3
4  import 'package:firebase_core/firebase_core.dart';
5  import 'package:mentorsynthhub/loginPage.dart';
6
7  Run | Debug | Profile
8  void main() async {
9    WidgetsFlutterBinding.ensureInitialized();
10
11    await Firebase.initializeApp(
12      options: FirebaseOptions(
13        apiKey: "AIzaSyAEDKttvoG3T6jvgalXCEROrlzPXSCeHkI",
14        projectId: "mentorsynthhub",
15        storageBucket: "mentorsynthhub.appspot.com",
16        messagingSenderId: "924424463364",
17        appId: "1:924424463364:android:d3acfcad93ebb149190151",
18      ));
19    runApp(MyApp());
20  }
```

Fig:3.1 main dart

ii)Login Page.dart: This LoginPage widget represents the UI for the opening page of your game. It includes buttons for starting the game and accessing the credits, as well as space for displaying control options. You can customize it further by adding additional UI elements or functionality as needed.

Core Functionality: The "Weight Management" app successfully fulfills its primary purpose of identifying the weight and catering to the needs of correct category.

```
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
import 'package:flutter/foundation.dart';
show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        return macos;
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for windows - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }
}
```

Fig:3.3.Firebase core

FIREBASE:

i) Create a Firebase Project:

1. Go to the Firebase Console
2. Click on "Add Project" and follow the setup instructions.

ii) Configure Flutter Project:

1. Add the Firebase configuration files to your Flutter project.
2. Follow the instructions provided in the Firebase Console to add google-services.json for Android and GoogleService-Info.plist for iOS.
3. These files contain configuration details needed for your app to connect to Firebase services.

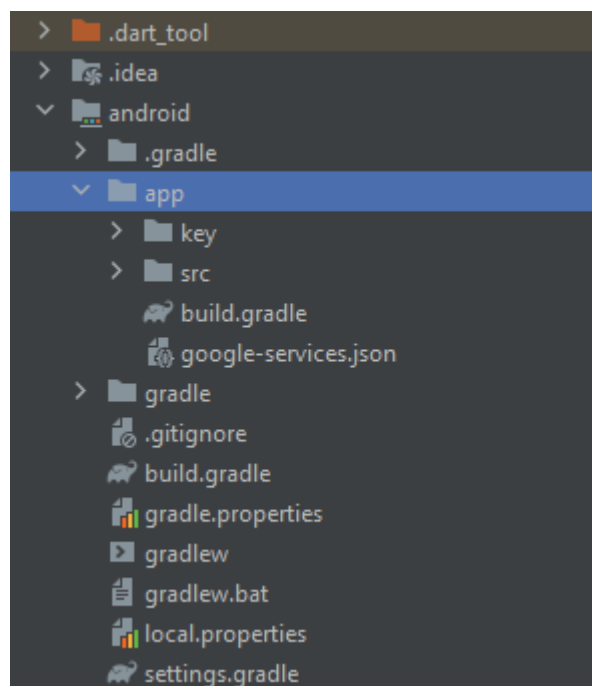


Fig:3.4: Firebase Configurations

iii) Add Firebase Dependencies:

Open the pubspec.yaml file and add the following dependencies:

yaml code:

dependencies:

firebase_core: ^2.25.3

firebase_auth: ^4.17.5

iv) Initialize Firebase:

In your main.dart or wherever you initialize your Flutter app, make sure to initialize Firebase. If you want to implement user authentication, Firebase provides the firebase_auth package. You can use it to sign in users using various methods like email/password, Google Sign-In, etc.

Dart code:

```
import 'package:flutter/material.dart';
import 'package:hostel_management_app/features/profile/controllers/account_setup_screen_controller.dart';
import 'package:hostel_management_app/features/authentication/controllers/authentication_repository.dart';
import 'package:hostel_management_app/features/authentication/controllers/forgotpassword_controller.dart';
import 'package:hostel_management_app/features/authentication/controllers/login_controller.dart';
import 'package:hostel_management_app/features/authentication/controllers/register_controller.dart';
import 'package:hostel_management_app/features/bookings/controllers/bookings_controller.dart';
import 'package:hostel_management_app/features/bookings/controllers/bookings_repository.dart';
import 'package:hostel_management_app/features/dashboard/controllers/dashboard_controller.dart';
import 'package:hostel_management_app/features/authentication/controllers/onboarding/onboarding_controller.dart';
import 'package:hostel_management_app/features/home/controllers/bottomnavbar_controller.dart';
import 'package:hostel_management_app/features/residents/controllers/residents_controller.dart';
import 'package:hostel_management_app/features/rooms/controllers/rooms_controller.dart';
import 'package:hostel_management_app/features/profile/controllers/user_controller.dart';
import 'package:hostel_management_app/features/authentication/screens/splash_screen/splash_screen.dart';
import 'package:provider/provider.dart';
import 'package:firebase_core/firebase_core.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: const FirebaseOptions(
      apiKey: "AIzaSy8UW30KcgHmfMr7z6irFLCz969Dcv9Amw",
      appId: "1:721358647180:android:8bba875f4f26ea2a8470a9",
      messagingSenderId: '',
      projectId: "hostel-management-app-4ae0f",
      storageBucket: "hostel-management-app-4ae0f.appspot.com"),
  );
  runApp(const MyApp());
}
```

Fig:3.5: Initializing Firebase

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1. CONCLUSION

Weight Management stands as a fundamental metric, serving as a quick indicator of one's overall body composition and potential health risks associated with weight. Integrating calculation into mobile applications has become increasingly popular, aligning with the modern lifestyle where individuals seek instant access to health-related information on their smartphones. In this context, leveraging the Flutter framework for developing a Weight Management proves to be a pragmatic choice, offering a blend of efficiency, versatility, and user-centric design.

Flutter, with its cross-platform capabilities and rich set of widgets, emerges as an optimal framework for crafting Weight Management that are both aesthetically pleasing and highly functional. The flexibility of Flutter allows developers to create fluid and responsive user interfaces, ensuring an intuitive user experience across various devices and screen sizes. Whether it's designing a sleek interface with customizable themes or implementing interactive elements for user input, Flutter provides developers with the tools needed to bring their Weight Management visions to life.

Moreover, the robustness of Flutter facilitates seamless integration with backend services for data processing and storage. By leveraging Flutter's compatibility with popular programming languages like Dart, developers can implement Weight Management algorithms with precision and efficiency. Whether it's utilizing

4.2. FUTURE WORK

Future work on Weight Management could focus on enhancing user engagement, personalization, and integration with emerging technologies. Here's a brief overview:

Enhanced User Engagement: Future Weight Management could incorporate gamification elements to make the experience more interactive and enjoyable. Features like achievement badges, challenges, and rewards for achieving health goals could motivate users to track their Index regularly and maintain healthy habits.

Personalization: Implementing machine learning algorithms to provide personalized recommendations based on users' Weight Management, fitness goals, and health history could significantly improve the effectiveness of Weight Management calculators. Tailored advice on nutrition, exercise routines, and lifestyle modifications could empower users to make sustainable changes to improve their health.

Integration with Wearable Devices: Integrating Weight Management with wearable devices such as smartwatches and fitness trackers would enable seamless tracking of physical activity, sleep patterns, and other health metrics. This integration would provide users with a comprehensive view of their health status and progress towards their fitness goals.

Health Data Analytics: Utilizing big data analytics techniques to analyze aggregated user data from calculators could uncover valuable insights into population health trends, risk factors for chronic diseases, and effectiveness of different interventions. This data-driven approach could inform public health initiatives and healthcare policies aimed at combating obesity and promoting healthy lifestyles.

Accessibility and Inclusivity: Ensuring that Weight Management calculators are accessible to diverse user populations, including individuals with disabilities or those with limited access to healthcare resources, is crucial. Future work could focus on improving the accessibility features of Weight Management apps, such as voice commands, screen reader compatibility, and multilingual support.

By focusing on these areas of future work, Weight Management can evolve into more sophisticated and user-centric tools for promoting health and wellness in the digital age.

REFERENCES

- [1] Chen, K., Fan, Y., & Liao, S. S. (2023). Token Incentives in a Volatile Crypto Market: The Effects of Token Price Volatility on User Contribution. *Journal of Management Information Systems*, 40*(2), 683-711.
- [2] Hasan, H. R., Salah, K., Jayaraman, R., Ahmad, R. W., Yaqoob, I., & Omar, M. (2020). Blockchain-based solution for the traceability of spare parts in manufacturing. *IEEE Access*, 8*, 100308-100322.
- [3] Liu, Z., Li, Y., Min, Q., & Chang, M. (2022). User incentive mechanism in blockchain-based online community: An empirical study of Steemit. *Information & Management*, 59*(7), 103596.
- [4] Sharma, R., Patel, S., & Nguyen, T. (2022). Successful Models of Industry-Academia Collaboration. *International Journal of Technology Transfer and Commercialization*, 8*, 25-42.
- [5] Williams, S. (2021). Fostering Collaboration between Academia and Industry in Spring Twenty Twentyone.

APPENDICES

A.SAMPLE CODE

i) Main Code:

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart'; // Import Get package
import 'package:learnle/customs/translator_page.dart';
import 'package:learnle/firebase_options.dart';
import 'package:learnle/models/bottom_navigation_bar.dart';
import 'package:learnle/pages/homePage.dart';
import 'package:learnle/screens/login_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      // Replace MaterialApp with GetMaterialApp
      debugShowCheckedModeBanner: false,
      title: 'LearnLE',
      theme: ThemeData(
        backgroundColor: Colors.white,
        appBarTheme: AppBarTheme(color: Colors.white, elevation: 0),
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: LoginScreen(),
    );
  }
}
```

ii) pubspec.yaml:

```
name: learnle
description: A new Flutter project.
publish_to: "none"
version: 1.0.0+1
environment:
  sdk: ">=2.19.6 <3.0.0"

dependencies:
  cupertino_icons: ^1.0.2
  flutter:
    sdk: flutter
  get: ^4.6.6

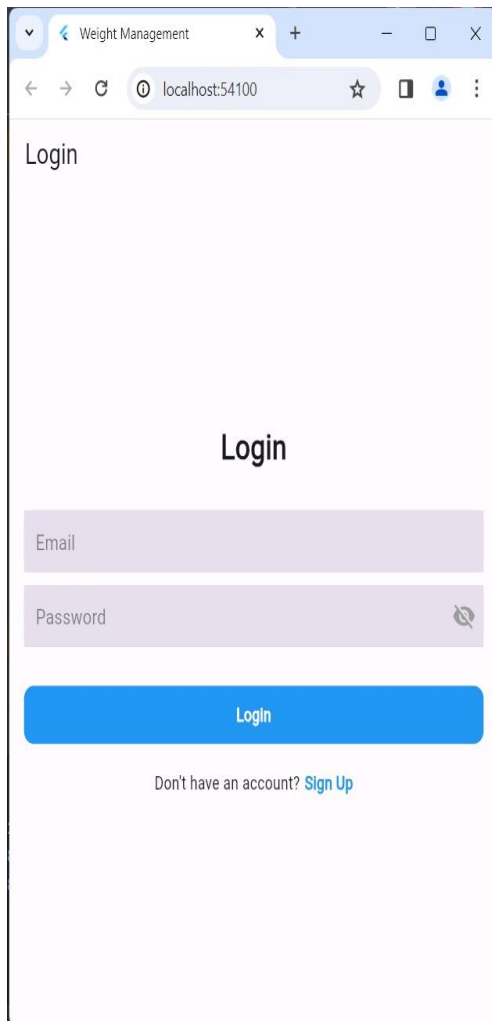
dev_dependencies:
  cloud_firestore: ^4.5.2
  firebase_auth: ^4.4.2
  firebase_core: ^2.10.0
  flutter_lints: ^3.0.2
  flutter_svg: ^2.0.5

  flutter_test:
    sdk: flutter
  flutter_tts: ^4.0.2
  fluttertoast: ^8.2.1
  translator: ^1.0.0

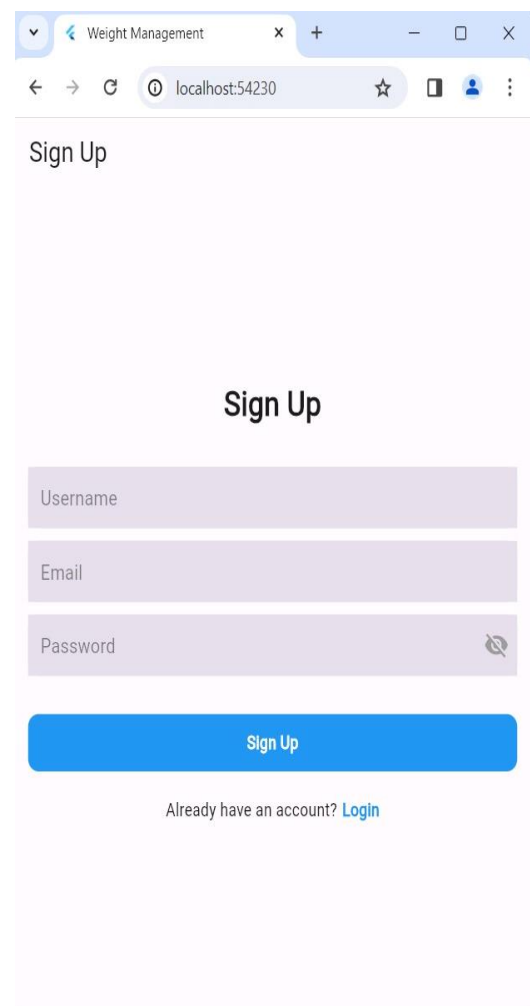
flutter:
  uses-material-design: true
  - assets/logo1.png
  - assets/F6.png
  - assets/F5.png
  - assets/F4.png
  - assets/F3.png
  - assets/F1.png
  - assets/F2.png
  - assets/profile-icon-9.png
  - assets/icons/
```

B.SCREEN SHOTS

Login Page and Signup Page:

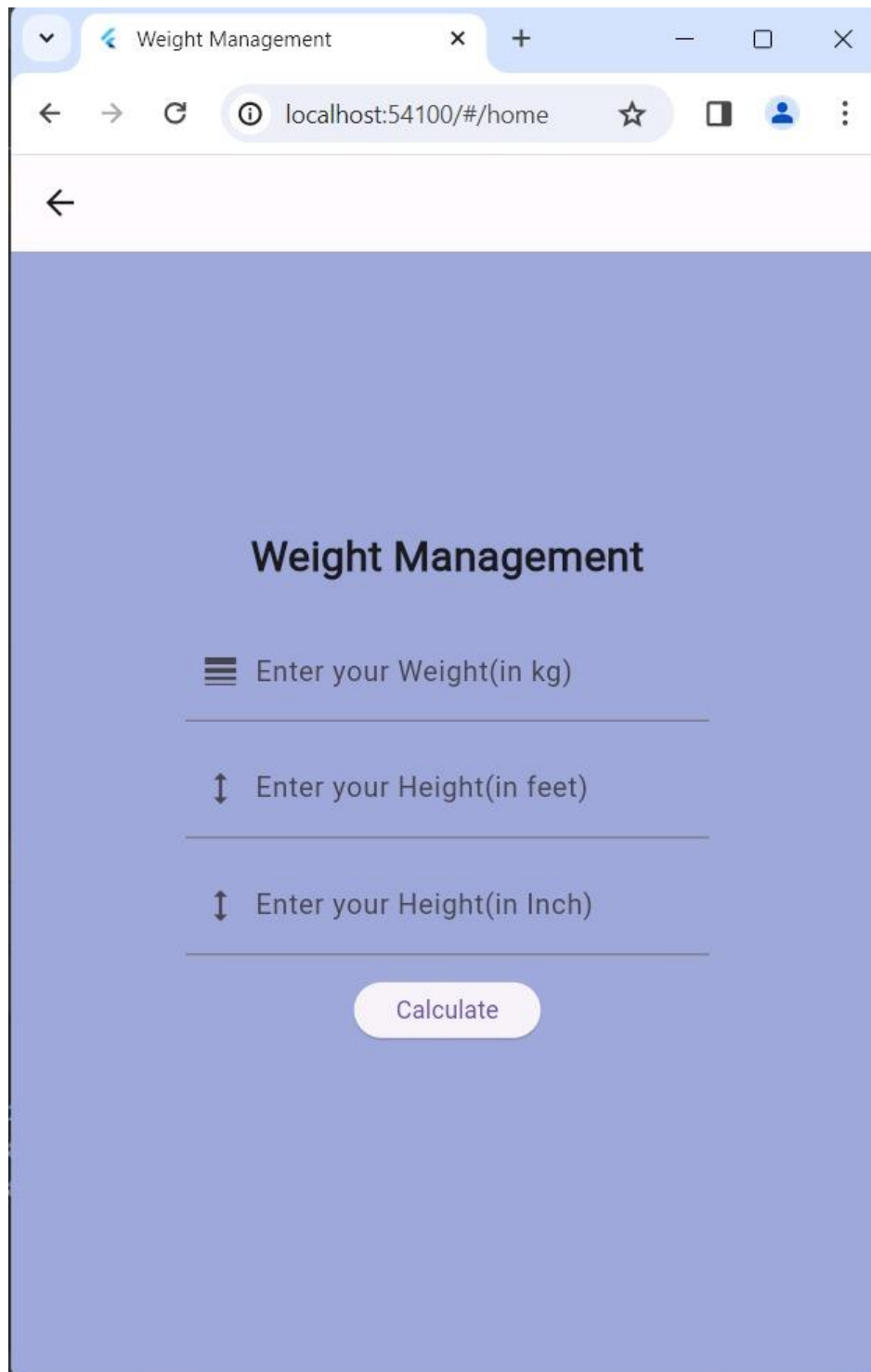


A screenshot of a web browser displaying the 'Login' page of a 'Weight Management' application. The browser's address bar shows 'localhost:54100'. The page has a light pink background. At the top, the word 'Login' is written in a large, dark font. Below this, there are two input fields: 'Email' and 'Password'. The 'Password' field has a small eye icon to its right. A blue button labeled 'Login' is positioned below the input fields. At the bottom of the page, there is a link that says 'Don't have an account? [Sign Up](#)'.



A screenshot of a web browser displaying the 'Sign Up' page of a 'Weight Management' application. The browser's address bar shows 'localhost:54230'. The page has a light pink background. At the top, the words 'Sign Up' are written in a large, dark font. Below this, there are three input fields: 'Username', 'Email', and 'Password'. The 'Password' field has a small eye icon to its right. A blue button labeled 'Sign Up' is positioned below the input fields. At the bottom of the page, there is a link that says 'Already have an account? [Login](#)'.

Home Page:



A screenshot of a web browser displaying the 'Weight Management' application. The browser's address bar shows 'localhost:54100/#/home'. The page has a solid blue background. At the top, the title 'Weight Management' is centered in a large, bold, black font. Below the title, there are three input fields, each preceded by a small icon: a hamburger menu icon for weight, a double-headed vertical arrow for height in feet, and another double-headed vertical arrow for height in inches. Each input field is a simple horizontal line. At the bottom center of the form area, there is a white, rounded rectangular button with the text 'Calculate' in black.

Weight Management

☰ Enter your Weight(in kg)

↕ Enter your Height(in feet)

↕ Enter your Height(in Inch)

Calculate

Calculation :

Weight Management

← → ↻ ⓘ localhost:54100/#/home ☆ 📱 👤 ⋮

←

Weight Management

☰ Enter your Weight(in kg)
55

↕ Enter your Height(in feet)
5

↕ Enter your Height(in Inch)
5

Calculate

Normal
Your BMI is: 20.18