

A PROJECT REPORT
ON

BALANCEDBOWL

AN ANDROID APPLICATION

By

Ms. Samruddhi Sham Kale

Ms. Shalini Shankar Jakkula

Towards The Partial Fulfilment of the
Bachelor of Computer Application



Estd. 1921

Tilak Maharashtra Vidyapeeth, Pune

[2023-2024]

A PROJECT REPORT
ON
BALANCEDBOWL

AN ANDROID APPLICATION

By
Ms. Samruddhi Sham Kale
Ms. Shalini Shankar Jakkula

Towards The Partial fulfilment of the
Bachelor of Computer Application



Estd. 1921

Tilak Maharashtra Vidyapeeth, Pune
[2023-2024]

CERTIFICATE

This is to certify that the project

“ BALANCEDBOWL ”

Has been satisfactorily completed by

Ms. SAMRUDDHI SHAM KALE
Ms. SHALINI SHANKAR JAKKULA

Towards The Partial fulfilment of the
‘Bachelor of Computer Application’,
For the Academic Year 2023-2024, at Tilak
Maharashtra Vidyapeeth, Pune, and
it is approved.

Project Guide

Examiner

Head of Department

ACKNOWLEDGEMENT

It is not possible to complete a project without the assistance & encouragement of other people. This one is certainly no exception.

It has been a remarkable experience of satisfaction & pleasure for our to complete our project 'BALANCEDBOWL' under the guidance of our project guide and Head of Department, Mrs. Asmita Namjoshi. We are extremely thankful to her for her valuable guidance and support on completion of this project. We would also like to extend our gratitude to Ms. Minal Kalamkar for her kind co-operation and advice throughout the journey of this project.

We also acknowledge with a deep sense of reverence, our gratitude towards all the faculty members for their constant support.

This acknowledgement is a humble attempt to earnestly thank all those who were directly or indirectly involved in the accomplishment of this project.

Any omission in this brief acknowledgement does not mean lack of gratitude.

Ms. Samruddhi Sham Kale

PRN-04421000505

Ms. Shalini Shankar Jakkula

PRN-04421000141

Contents

1. INTRODUCTION.....	6
1.1 INTRODUCTION:.....	6
1.2 OBJECTIVES & PURPOSE OF PROJECT:.....	7
2. TECHNOLOGIES USED	8
2.1 ANDROID:	8
2.2 APIS:	8
2.2.1 Google APIs	Error! Bookmark not defined.
2.2.2 Optional APIs	Error! Bookmark not defined.
2.3 ANDROID APPLICATION LIFE CYCLE.....	9
2.3.1 Activity-lifecycle concepts.....	11
2.4 ANDROID ARCHITECTURE	14
2.4.1 Layers	15
2.4.2 Android Application.....	16
2.4.3 Application Framework.....	17
2.4.4 Android Runtime and Core/Native Libraries	20
2.4.5 Linux Kernel	Error! Bookmark not defined.
2.5 APPLICATION COMPONENTS	22
2.5.1 Activities:	22
2.5.2 Services:	Error! Bookmark not defined.
2.5.3 Broadcast receivers:	23
2.5.4 Content providers:	23
2.5.5 Views and View Groups:	23
2.5.6 Intents:	23
2.5.7 Events and event listeners:	Error! Bookmark not defined.
2.6 ANDROID SQLITE DATABASE	24

2.6.1 Android SQLite Java Classes	24
2.7 JAVA FOR ANDROID DEVELOPMENT	27
2.7.1 Java Virtual Machine	28
2.7.2 Dalvik Virtual Machine	29
2.7.3 Difference between JVM and DVM	Error! Bookmark not defined.
2.8 XML	29
2.8.1 Basics of User Interface (UI)	29
2.8.2 Different Types of XML Files Used:	30
2.9 GRADLE.....	32
2.9.1 setting.gradle	Error! Bookmark not defined.
4.9.2 build.gradle (project level)	32
4.9.3 build.gradle (application level)	32
5. HARDWARE & SOFTWARE REQUIREMENTS	33
5.1 DEVELOPER REQUIREMENTS:.....	33
5.2 APP USER REQUIREMENTS:.....	33
6. ENTITY RELATIONSHIP DIAGRAM	34
6.1 ER DIAGRAM.....	34
6.2 COMPONENTS OF THE ER DIAGRAM.....	Error! Bookmark not defined.
6.2.1 Entities.....	Error! Bookmark not defined.
6.2.2 Attributes	Error! Bookmark not defined.
6.2.3 Relationships	Error! Bookmark not defined.
6.3 BINARY RELATIONSHIP AND CARDINALITY	Error! Bookmark not defined.
6.3.1 One-to-one.....	Error! Bookmark not defined.
6.3.2 One-to-many.....	Error! Bookmark not defined.
6.3.3 Many-to-one	Error! Bookmark not defined.
6.3.4 Many-to-many	Error! Bookmark not defined.
8. CLASS DIAGRAM	36

8.1 UML CLASS DIAGRAM.....	36
8.2 PURPOSE OF CLASS DIAGRAMS	Error! Bookmark not defined.
8.3 BENEFITS OF CLASS DIAGRAMS	Error! Bookmark not defined.
8.4 VITAL COMPONENTS OF A CLASS DIAGRAM	Error! Bookmark not defined.
8.4.1 Upper Section:.....	Error! Bookmark not defined.
8.4.2 Middle Section:	Error! Bookmark not defined.
8.4.3 Lower Section:	Error! Bookmark not defined.
8.5 RELATIONSHIPS	Error! Bookmark not defined.
8.5.1 Dependency:.....	Error! Bookmark not defined.
8.5.2 Generalization:	Error! Bookmark not defined.
8.5.3 Association:.....	Error! Bookmark not defined.
8.5.4 Multiplicity:.....	Error! Bookmark not defined.
8.5.5 Aggregation:.....	Error! Bookmark not defined.
8.5.6 Composition:	Error! Bookmark not defined.
9. USE CASE DIAGRAM.....	38
9.1 USE CASE DIAGRAM.....	38
9.2 USE CASE DIAGRAM COMPONENTS	Error! Bookmark not defined.
9.2.1 Actors:	Error! Bookmark not defined.
9.2.2 System:	Error! Bookmark not defined.
9.2.3 Goals:	Error! Bookmark not defined.
9.3 USE CASE DIAGRAM SYMBOLS AND NOTATION.	Error! Bookmark not defined.
9.3.1 Use cases:	Error! Bookmark not defined.
9.3.2 Actors:	Error! Bookmark not defined.
9.3.3 Associations:	Error! Bookmark not defined.
9.3.4 System boundary boxes:.....	Error! Bookmark not defined.
9.3.5 Packages:.....	Error! Bookmark not defined.
10. ACTIVITY DIAGRAM.....	40

10.1 UML ACTIVITY DIAGRAM.....	40
10.2 USE OF ACTIVITY DIAGRAM.....	Error! Bookmark not defined.
10.3 COMPONENTS OF AN SEQUENCE DIAGRAM	Error! Bookmark not defined.
10.3.1 Activities	Error! Bookmark not defined.
10.3.2 Activity partition /swim lane	Error! Bookmark not defined.
10.3.3 Forks.....	Error! Bookmark not defined.
10.3.4 Join Nodes	Error! Bookmark not defined.
10.3.5 Pins	Error! Bookmark not defined.
10.4 NOTATION OF AN ACTIVITY DIAGRAM.....	Error! Bookmark not defined.
11. SEQUENCE DIAGRAM	42
TESTCASES	44
11.1 HOME SCREEN	49
11.2 SIGNUP SCREEN.....	50
11.3 LOGIN SCREEN	51
11.4 DETAILS SCREEN.....	52
11.5 RECOMMENDATION ACTIVITY.....	53
11.6 UPDATE Activity	54
11.7 CALORIES CONSUMED ACTIVITY.....	55
11.8 FOOD INTAKE ACTIVITY	56
11.9 ENTER EXTRA CALORIE ACTIVITY	57
ENTER FOOD YOU CONSUMED ACTIVITY.....	58
11.11 ENTER FOOD YOU CONSUMED.....	59
11.11 ENTER FOOD CONSUMED	61
11.11 ENTER FOOD CONSUMED	62
11.11 ENTER FOOD CONSUMED	63
11.11 ENTER FOOD CONSUMED	64

11.11 SUGGESTIONS IF CALORIES INTAKE IS LOW	64
11.15 EXERCISE SUGGESTION ACTIVITY	66
12. LIMITATIONS & BOUNDARIES	67
13. ADVANTAGES	68
14.FEATURES	70
15. CONCLUSION	72
15.1 APPLICATION OVERVIEW	72
15.2 APPLICATION FEATURES:	73
16. BIBLIOGRAPHY	74
16.1 WEBSITES:	74
16.2 REFERENCES:	75

1. INTRODUCTION

1.1 INTRODUCTION:

“BALANACEDBOWL” is an android application that helps maintain the health record of an individual user by profile information with our easy-to-use diet planner app, you'll not only calculate your BMI (Body Mass Index) in a snap but also track your calorie intake effortlessly. No more guesswork – just enter your details, and we'll do the rest. Whether you're aiming to simply maintain a balanced diet.

As users sign up or log in to the app, prompt them to provide basic information such as age, gender, height, and weight. This information will be used not only to calculate their BMI but also to personalize their diet plan.

Develop a BMI calculator feature that takes user input (height and weight) and calculates their BMI using the formula: $BMI = \text{weight (kg)} / \text{height (m)}^2$. Display the calculated BMI along with its corresponding category (underweight, normal weight, overweight, obese) to provide users with context about their current weight status.

Allow users to set specific goals related to weight management or overall health improvement. Based on their BMI and desired outcomes, provide recommendations for target weight, calorie intake, and rate of weight change. Enable users to track their progress over

Allow users to log their daily food intake within the app to track their calorie consumption. Provide a database of foods with calorie information to facilitate logging. Compare users' logged calorie intake against their daily calorie targets. If users exceed their calorie goals, provide feedback indicating that they have consumed more calories than needed for the day. When users exceed their daily calorie targets, suggest appropriate types and durations of exercise to help offset the excess calories and maintain overall calorie balance.

1.2 OBJECTIVES & PURPOSE OF PROJECT:

- Using the android device to enhance the personal health care experience.
- Providing a single place to store all health records of an individual.
- BMI calculation is to provide users with a quick assessment of their weight status for informed health and fitness decisions.
- Allow users to log their daily food intake within the app to track their calorie consumption.
- Provide a database of foods with calorie information to facilitate logging
Compare users' logged calorie intake against their daily calorie targets.
- If users exceed their calorie goals, provide feedback indicating that they have consumed more calories than needed for the day.
- When users exceed their daily calorie targets, suggest appropriate types and durations of exercise to help offset the excess calories and maintain overall calorie balance.

2. TECHNOLOGIES USED

2.1 ANDROID:

Android, as a system, is a Java-based operating system that runs on the Linux kernel.

The system is very lightweight and full featured. Android applications are developed using Java and can be ported rather easily to the new platform. Other features of Android include an accelerated 3-D graphics engine (based on hardware support), database support powered by SQLite, and an integrated web browser.

One of the more exciting and compelling features of Android is that, because of its architecture, third-party applications—including those that are “home grown”—are executed with the same system priority as those that are bundled with the core system.

This is a major departure from most systems, which give embedded system apps a greater execution priority than the thread priority available to apps created by third-party developers. Also, each application is executed within its own thread using a very lightweight virtual machine.

Aside from the very generous SDK and the well-formed libraries that are available to us to develop with, the most exciting feature for Android developers is that we now have access to anything the operating system has access to.

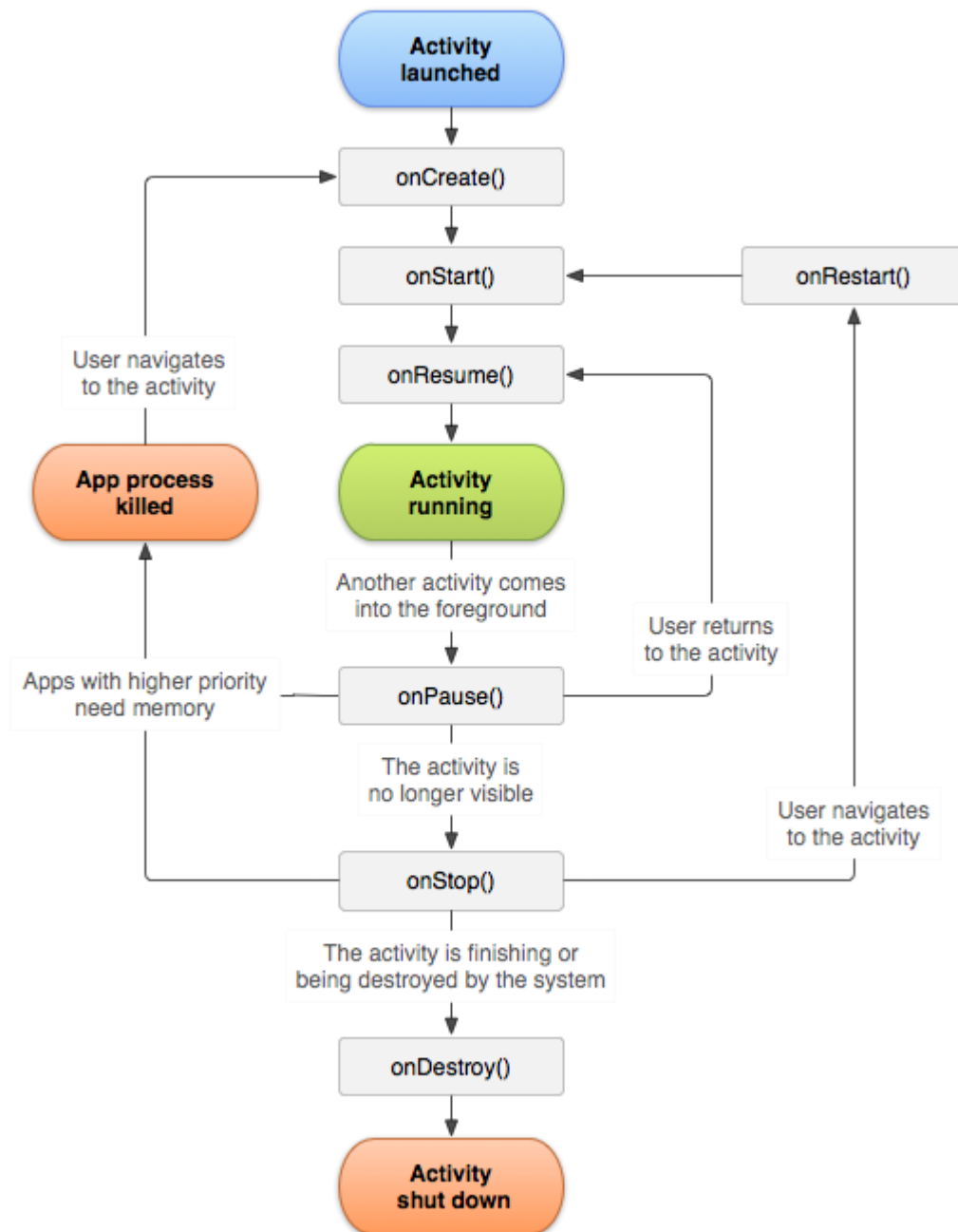
2.2 APIS:

The API, or application programming interface, is the core of the Android SDK. An API is a collection of functions, methods, properties, classes, and libraries that is used by application developers to create programs that work on specific platforms. The Android API contains all the specific information that you need to create applications that can work on and interact with an Android-based application.

The Android SDK also contains two supplementary sets of APIs—the Google APIs and the Optional APIs.

2.3 ANDROID APPLICATION LIFE CYCLE

The Android application life cycle is unique in that the system controls much of the life cycle of the application. All Android applications, or Activities, are run within their own process. All of the running processes are watched by Android and, depending on how the activity is running (this is, a foreground activity, background activity, and so forth), Android may choose to end the activity to reclaim needed resources.

**Figure 1 Android Lifecycle**

2.3.1 Activity-lifecycle concepts

2.3.1.1 onCreate():

On activity creation, the activity enters the Created state. In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity. For example, your implementation of onCreate() might bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables. This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

If you have a lifecycle-aware component that is hooked up to the lifecycle of your activity it will receive the ON_CREATE event. The method annotated with @OnLifecycleEvent will be called so your lifecycle-aware component can perform any setup code it needs for the created state.

2.3.1.2 onStart():

When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the app initializes the code that maintains the UI.

When the activity moves to the started state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_START event.

The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the Resumed state, and the system invokes the onResume() method.

2.3.1.3 onResume():

When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback. This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app. Such an event might

be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off.

When the activity moves to the resumed state, any lifecycle-aware component tied to the activity's lifecycle will receive the `ON_RESUME` event. This is where the lifecycle components can enable any functionality that needs to run while the component is visible and, in the foreground, such as starting a camera preview.

When an interruptive event occurs, the activity enters the Paused state, and the system invokes the `onPause()` callback.

If the activity returns to the Resumed state from the Paused state, the system once again calls `onResume()` method.

2.3.1.4 onPause():

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).

When the activity moves to the paused state, any lifecycle-aware component tied to the activity's lifecycle will receive the `ON_PAUSE` event. This is where the lifecycle components can stop any functionality that does not need to run while the component is not in the foreground, such as stopping a camera preview.

2.3.1.5 onStop():

When the activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the `onStop()` callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call `onStop()` when the activity has finished running, and is about to be terminated.

When the activity moves to the stopped state, any lifecycle-aware component tied to the activity's lifecycle will receive the `ON_STOP`

event. This is where the lifecycle components can stop any functionality that does not need to run while the component is not visible on the screen.

In the `onStop()` method, the app should release or adjust resources that are not needed while the app is not visible to the user.

2.3.1.6 onRestart():

The `onRestart()` function is called when the application comes back to the foreground from the background. This method is only called when the user navigates back to the stopped activity.

The callback is immediately followed by the `onStart()` and `onResumed()` callbacks.

2.3.1.7 onDestroy():

`onDestroy()` is called before the activity is destroyed. The system invokes this callback either because:

1. The activity is finishing (due to the user completely dismissing the activity or due to `finish()` being called on the activity), or
2. The system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)

When the activity moves to the destroyed state, any lifecycle-aware component tied to the activity's lifecycle will receive the `ON_DESTROY` event. This is where the lifecycle components can clean up anything it needs to before the Activity is destroyed.

2.4 ANDROID ARCHITECTURE

Android operating system's initial release was in the year 2008. Even at its start, the team behind the operating system built it on top of the shoulders of giants. Beyond the user interface that the Android OS presents at the surface level, it is made up of multiple layers. These layers include custom code and open-source technologies that have been under continuous development for decades.

Android has been developed through massive collaborative efforts and investments by many companies. The main company behind android development is Google. Other companies include device manufacturers such as Samsung, LG; processor manufacturers such as Intel and ARM, but to name a few.

When we talk about Android architecture, we mean how the Android system has been designed, segmented into layers, and built up to work as a system. Building such a complex system requires careful structuring to ensure all the components work together cohesively. Its architecture ensures that the many components function as a whole without crashing.

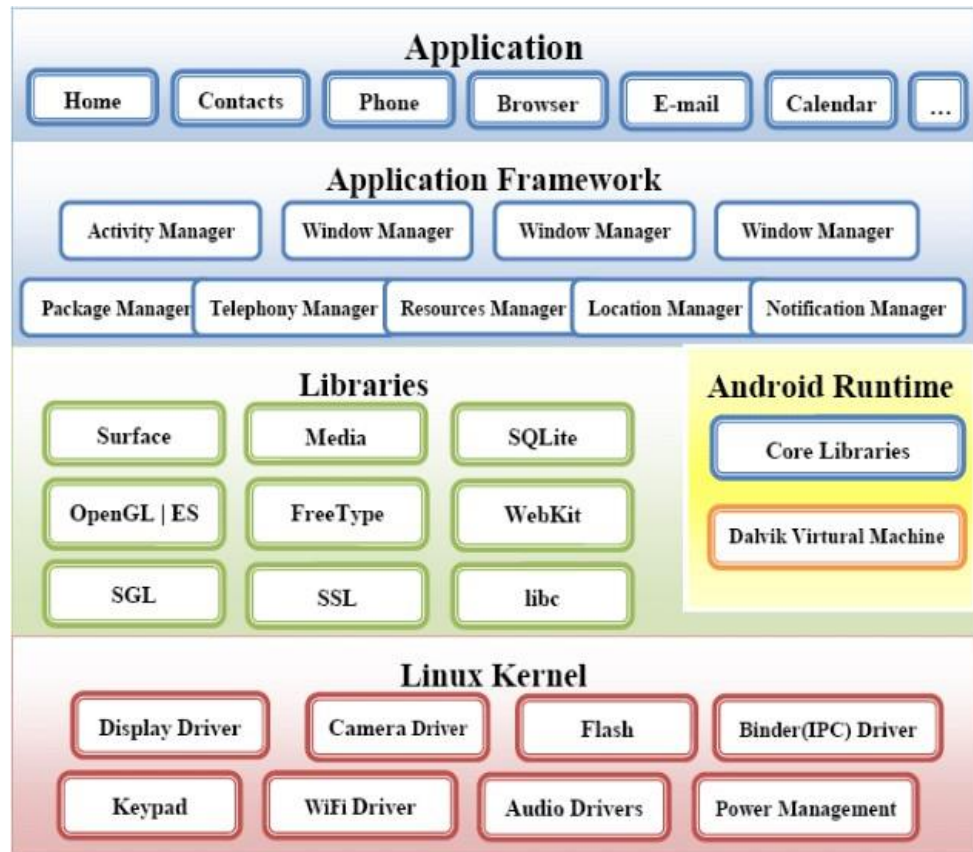


Figure 2 Application Framework

2.4.1 Layers

The following are the layers that compose the Android architecture as labelled on the diagram:

1. Application
2. Application Framework
3. Android Runtime and Core Libraries
4. Linux Kernel

Developing an operating system for mobile devices comes with a set of challenges. Using this layered architecture ensures that different problems are broken down and solved at different levels.

A layered architecture helps separate concerns and ensure android software developers don't have to deal with low-level problems at every turn. They can instead focus on delivering business value concerned with the layer they're working on.

Developers are working on making apps don't have to worry about the application framework implementation. That work is left for the system developers working on the Application framework.

The Application Framework developers work on developer experience and don't have to worry about the low-level drivers. Low-level system engineers can focus completely on low-level components such as Bluetooth or Audio drivers and the like.

Android's layered structure makes it possible to apply updates with bug fixes or improvements to each layer on its own. This ensures that changes across layers don't interfere with each other. This makes it possible for people working at a different level of the OS to work with obstructing each other as new updates and releases are done.

2.4.2 Android Application

This is the layer that end-users interact with. It is on this layer where application developers publish their applications to run.

Android, by default, comes with a set of applications that make android devices usable from the offset.

- 1) **Home:** The Homepage on Android consists of launcher icons for commonly used applications that the end-user may want quick access to. You can start the apps by clicking on the launchers of these apps. At the very top of the screen, you have widgets that show network, battery level, date, and time.
- 2) **Contacts:** Android, by default, provides a means to store and retrieve contacts. Contact information is shared across other apps to enhance functionality.
- 3) **Messages:** Android provides the capability to send and receive SMS messages.

- 4) **Email:** Android comes with native support for email services. Setting up an Android device requires a Gmail account. Setting up Gmail activates other email-dependent components on Android devices. Some email dependent features include security and recovery mechanisms. Another email dependent feature is access to the Play Store, a marketplace for Android applications.
- 5) **Browser:** Android comes with a default browser.
- 6) **Notification Drawer:** Swiping down on the screen exposes the notification drawer. It provides application events that the user should be aware of. Above the notification are a set of shortcuts to some commonly used device settings that the users can toggle. These settings include on and off toggles for various hardware components such as Bluetooth and WIFI. Long pressing these events enables us to navigate to their configurations page.

This layer is also referred to as user-level in contrast to the layers below that are mostly tuned for application development. Application developers create and customize the experiences for their apps on this layer. The layers below the application layer are not customized by application developers. They are considered part of the system layer. These layers are customized by device manufacturers, Google android teams, or third parties who want to use the Android source code for their product or research.

2.4.3 Application Framework

The Android OS exposes the underlying libraries and features of the Android device that are using a Java API. This is what is known as the Android framework. The framework exposes a safe and uniform means to utilize Android device resources.

2.4.3.1 Activity Manager

Applications use the Android activity component for presenting an entry point to the app. Android Activities are the components that house the user interface that app users interact with. As end-users interact with the

Android device, they start, stop, and jump back and forth across many applications. Each navigation event triggers activation and deactivation of many activities in respective applications.

The Android Activity Manager is responsible for predictable and consistent behaviour during application transitions. The Activity Manager provides a slot for app creators to have their apps react when the Android OS performs global actions. Applications can listen to events such as device rotation, app destruction due to memory shortage, an app being shifted out of focus, and so on.

Some examples of the way applications can react to these transitions include pausing activity in a game, stopping music playing during a phone call.

2.4.3.2 Window Manager

Android can determine screen information to determine the requirements needed to create windows for applications. Windows are the slots where we can view our app user interface. Android uses the Window manager to provide this information to the apps and the system as they run so that they can adapt to the mode the device is running on.

The Window Manager helps in delivering a customized app experience. Apps can fill the complete screen for an immersive experience or share the screen with other apps. Android enables this by allowing multi-windows for each app.

2.4.3.3 Location Manager

Most Android devices are equipped with GPS devices that can get user location using satellite information to which can go all the way to meters precision. Programmers can prompt for location permission from the users, deliver location, and aware experiences.

Android is also able to utilize wireless technologies to further enrich location details and increase coverage when devices are enclosed spaces. Android provides these features under the umbrella of the Location-Manager.

2.4.3.4 Telephony Manager

Most Android devices serve a primary role in telephony. Android uses Telephone Manager to combine hardware and software components to deliver telephony features. The hardware components include external parts such as the sim card, and device parts such as the microphone, camera, and speakers. The software components include native components such as dial pad, phone book, ringtone profiles. Using the Telephone Manager, a developer can extend or fine-tune the default calling functionality.

2.4.3.5 Resource Manager

Android app usually come with more than just code. They also have other resources such as icons, audio and video files, animations, text files, and the like. Android helps in making sure that there is efficient, responsive access to these resources. It also ensures that the right resources are delivered to the end-users.

For example, the proper language text files are used when populating fields in the apps.

2.4.3.6 View System

Android also provides a means to easily create common visual components needed for app interaction. These components include widgets like buttons, image holders such as `ImageView`, components to display a list of items such as `ListView`, and many more. The components are premade but are also customizable to fit app developer needs and branding.

2.4.3.7 Notification Manager

The Notification Manager is responsible for informing Android users of application events. It does this by giving users visual, audio or vibration signals or a combination of them when an event occurs. These events have

external and internal triggers. Some examples of internal triggers are low-battery status events that trigger a notification to show low battery. Another example is user-specified events like an alarm. Some examples of external triggers include new messages or new Wi-Fi networks detected.

Android provides a means for programmers and end-users to fine-tune the notifications system. This can help to guarantee they can send and receive notification events in a means that best suits them and their current environments.

2.4.3.8 Package Manager

Android also provides access to information about installed applications. Android keeps track of application information such as installation and uninstallation events, permissions the app requests, and resource utilization such as memory consumption.

This information can enable developers to make their applications to activate or deactivate functionality depending on new features presented by companion apps.

2.4.3.9 Content Provider

Android has a standardized way to share data between applications on the device using the content provider. Developers can use the content provider to expose data to other applications. For example, they can make the app data searchable from external search applications. Android itself exposes data such as calendar data, contact data, and the like using the same system.

2.4.4 Android Runtime and Core/Native Libraries

2.4.4.1 Android Runtime

Android currently uses Android Runtime (ART) to execute application code. ART is preceded by the Dalvik Runtime that compiled developer

code to Dalvik Executable files (Dex files). These execution environments are optimized for the android platform taking into consideration the processor and memory constraints on mobile devices.

The runtime translates code written by programmers into machine code that does computations and utilizes android framework components to deliver functionality. Android hosts multiple applications and system components that each run in their processes.

2.4.4.2 Core Libraries

In this segment, we will discuss some of the core libraries that are present in the Android operating system.

2.4.4.2.2 SQLite

Android also has an SQLite database that enables applications to have very fast native database functionality without the need for third party libraries.

2.4.4.2.9 Surface Manager

The surface manager is responsible for ensuring the smooth rendering of application screens. It does this by composing 2D and 3D graphics for rendering. It further enables this by doing off-screen buffering.

2.5 APPLICATION COMPONENTS

Android applications are organized as a collection of components. There are four types of components, and applications can be composed of one or more of each type. A dynamic instance of a component corresponds to an application subset that can be executed independently of the others. So, in many ways, an Android application can be thought of as a collection of interacting components.

- **Activities.** User-facing components that implement display and input capture.
- **Services.** Background components that operate independent of any user-visible activity.
- **Broadcast receivers.** A component that listens for and responds to system-wide broadcast announcements.
- **Content providers.** Components that make application data accessible to external applications and system components.

2.5.1 Activities:

An activity component implements interactions with the user. Activities are typically designed to manage a single type of user action, and multiple activities are used together to provide a complete user interaction.

For example, a mapping application may consist of two activities: one that presents to the user a list of locations to map, and one to display a map graphic that includes the chosen location. An activity includes a default window for drawing visual elements. An activity will use one or more view objects, which are organized hierarchically, to draw or capture user input. Views can be thought of as widgets, or user-interface objects, such as check boxes, images, and lists that are common to all types of GUI-based development environments. The Android SDK includes a number of views for developer use.

2.5.3 Broadcast receivers:

As previously discussed, system-wide broadcast events can be generated by the system software or by applications. Components that listen to these broadcasts on behalf of applications are broadcast receivers. An application can include multiple broadcast receivers listening for announcements. In response, a broadcast receiver can initiate another component, such as an activity, to interact with the user or use the system-wide notification manager.

2.5.4 Content providers:

Components that provide access to an application's data are content providers. Base classes are provided in the Android SDK for both the content provider (that is, the content provider component must extend the base class) and the component seeking access. The content provider is free to store the data in whatever back-end representation it chooses, be it the file system, the SQLite service, or some application-specific representation (including those implemented via remote web services).

Android applications consist of combinations of these component type instances. The invocation of components is managed through a system-wide broadcast mechanism based on intents.

2.5.5 Views and View Groups:

A view is an interface widget, such as a button or a text input; views are grouped into view groups, which represent hierarchical organizations of the layout and content.

2.5.6 Intents:

An intent is the specification of a request for action. Intents allow communication between activities, either explicitly, by naming the activity the intent is targeting, or implicitly, by naming the desired action

to which activities capable of performing it can be bound at runtime. Implicit intents are resolved by associating activities to intent filters, which are conditions that specify the actions an activity can perform. Intents are also used to send and receive broadcast messages, which notify system or applications event. Intents can also contain data to enable parameter passing among activities.

2.6 ANDROID SQLITE DATABASE

SQLite is an embedded, relational database management system (RDBMS). Most relational databases (Oracle and MySQL being prime examples) are standalone server processes that run independently, and in cooperation with, applications that require database access. SQLite is referred to as embedded because it is provided in the form of a library that is linked into applications. As such, there is no standalone database server running in the background. All database operations are handled internally within the application through calls to functions contained in the SQLite library.

The developers of SQLite have placed the technology into the public domain with the result that it is now a widely deployed database solution.

SQLite is written in the C programming language and as such, the Android SDK provides a

Java based “wrapper” around the underlying database interface. This essentially consists of a set of classes that may be utilized within the Java code of an application to create and manage SQLite based databases.

2.6.1 Android SQLite Java Classes

SQLite is written in the C programming language whilst Android applications are primarily developed using Java. To bridge this “language gap”, the Android SDK includes a set of classes that provide a Java layer on top of the SQLite database management system.

2.6.1.1 Cursor:

A class provided specifically to provide access to the results of a database query. For example, a SQL SELECT operation performed on a database will potentially return multiple matching rows from the database. A Cursor instance can be used to step through these results, which may then be accessed from within the application code using a variety of methods. Some key methods of this class are as follows:

- `close ()` – Releases all resources used by the cursor and closes it.
- `getCount()` – Returns the number of rows contained within the result set.
- `moveToFirst()` – Moves to the first row within the result set.
- `moveToLast()` – Moves to the last row in the result set.
- `moveToNext()` – Moves to the next row in the result set.
- `move()` – Moves by a specified offset from the current position in the result set.
- `get<type>()` – Returns the value of the specified <type> contained at the specified column index of the row at the current cursor position (variations consist of `getString()`, `getInt()`, `getShort()`, `getFloat()` and `getDouble()`).

2.6.1.2 SQLite Database:

This class provides the primary interface between the application code and underlying SQLite databases including the ability to create, delete and perform SQL based operations on databases. Some key methods of this class are as follows:

- `insert ()` – Inserts a new row into a database table.
- `delete ()` – Deletes rows from a database table.
- `query ()` – Performs a specified database query and returns matching results via a Cursor object.
- `execSQL ()` – Executes a single SQL statement that does not return result data.
- `rawQuery ()` – Executes an SQL query statement and returns matching results in the form of a Cursor object.

2.6.1.3 SQLiteOpenHelper:

A helper class designed to make it easier to create and update databases. This class must be subclassed within the code of the application seeking database access and the following callback methods implemented within that subclass:

- `onCreate()` – Called when the database is created for the first time. This method is passed as an argument the SQLite Database object for the newly created database. This is the ideal location to initialize the database in terms of creating a table and inserting any initial data rows.
- `onUpgrade()` – Called in the event that the application code contains a more recent database version number reference. This is typically used when an application is updated on the device and requires that the database schema also be updated to handle storage of additional data.

In addition to the above mandatory callback methods, the `onOpen()` method, called when the database is opened, may also be implemented within the subclass.

The constructor for the subclass must also be implemented to call the super class, passing through the application context, the name of the database and the database version.

Notable methods of the SQLiteOpenHelper class include:

- `getWritableDatabase()` – Opens or creates a database for reading and writing. Returns a reference to the database in the form of an SQLite Database object.
- `getReadableDatabase ()` – Creates or opens a database for reading only. Returns a reference to the database in the form of an SQLite Database object.
- Create and/or open a database. This will be the same object returned by `getWritableDatabase ()` unless some problem, such as a full disk, requires the database to be opened read-only. In that case, a read-only database object will be returned. If the problem is fixed, a future call

to `getWritableDatabase ()` may succeed, in which case the read- only database object will be closed and the read/write object will be returned in the future.

- `close ()` – Closes the database.

2.6.1.4 Content Values:

Content Values is a convenience class that allows key/value pairs to be declared consisting of table column identifiers and the values to be stored in each column. This class is of particular use when inserting or updating entries in a database table.

2.7 JAVA FOR ANDROID DEVELOPMENT

Android is a very popular OS for mobile phones today. Millions of apps are there those power different types, models and manufactures mobile phones today. Java the popular programming language is used to develop Android applications. The question is quite generic why java is used for android development. There are solid reasons why java is used for android development.

The main objective behind Android development was to create a platform-independent application environment that can run on every device. Java already has this quality so java was chosen for android development. Android applications run on a special virtual machine called the Dalvik VM that is a direct inspiration from java virtual machine called JVM. Android applications can run on any device where special Dalvik VM is implemented. This way android applications are compiled and run in optimum performance environment with the feature of platform independence.

The good approach towards software development is the object-oriented approach. Java is based on the OOPS concept. Android relies heavily on Java fundamentals like classes and objects and its other useful features of OOPS.

Java has an extensive set of libraries. It is easy to take advantage of these libraries. Android SDK has many standard Java libraries included. These

provide functionalities for data structure, math functions, graphics implementation, networking functions and much more. This way java helps develop Android applications fast and inefficient manner.

Java is very popular language due to its awesome features and performance. The community of Developers who have proficiency is big. Thus, android developers choose java as there is a good base of java programmers who are available and can help in creating, improving android applications along with many libraries and tools.

2.7.1 Java Virtual Machine

JVM is an engine that offers the Java Code or applications runtime environment. It transforms bytecode Java into the language of computers. JVM (Java Run Environment) is a component of JRE.

The JVM is a virtual machine to run Java desktop, server, and web applications. Another important thing about Java is it was developed with portability in mind. Thus, the JVM has been shaped also to support multiple host architectures and run everywhere. But it is too heavy for embedded devices.

The compiler generates machine code for a specific scheme in other programming languages.

However, for a virtual machine known as Java Virtual Machine, Java compiler generates code.

First, the bytecode for Java code is generated. This bytecode is interpreted on various computers. Bytecode is an intermediate code between the host system and the source code of Java. JVM is accountable for memory space allocation.

JVM is an abstract machine (Java Virtual Machine). It is a specification that offers runtime environment and allows the execution of java bytecode.

2.7.2 Dalvik Virtual Machine

The DVM is a virtual machine to run Android applications. The DVM executes Dalvik bytecode, which is compiled from programs written in the Java language.

One of the key design principles of the DVM is that it should run on low memory mobile devices and loads quicker compared to any JVM. Also, this VM is more efficient when it runs multiple instances on the same device.

The Android platform leverages the Dalvik Virtual machine (Dalvik VM) for memory, security, device, and process management. Although the internal intricacies of how the Dalvik Virtual Machine works is not that important to an average developer, you can think of the Dalvik VM as a box that provides the necessary environment for you to execute an Android application San the need of having to worry about the target device.

2.8 XML

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. It is derived from Standard Generalized Markup Language (SMGL). Basically, the XML tags are not predefined in XML. We need to implement and define the tags in XML. XML tags define the data and used to store and organize data. It's easily scalable and simple to develop. In Android, the XML is used to implement UI-related data, and it's a lightweight markup language that doesn't make layout heavy. XML only contains tags, while implementing they need to be just invoked.

2.8.1 Basics of User Interface (UI)

Basically, in Android XML is used to implement the UI-related data. So, understanding the core part of the UI interface with respect to XML is important. The User Interface for an Android App is built as the hierarchy of main layouts, widgets. The layouts are ViewGroup objects or

containers that control how the child view should be positioned on the screen. Widgets here are view objects, such as Buttons and text boxes.

2.8.2 Different Types of XML Files Used:

Different XML files serve different purposes in Android Studio. The list of various XML files in Android Studio with their purposes is discussed below.

2.8.2.1 Layout XML files in android

The Layout XML files are responsible for the actual User Interface of the application. It holds all the widgets or views like Buttons, TextViews, EditTexts, etc. which are defined under the ViewGroups. The Location of the layout files in Android is:

app -> src -> main -> res -> layout

The folder contains the layout files for respective activity, fragments.

2.8.2.2 AndroidManifest.xml file

This file describes the essential information about the application's, like the application's package names which matches code's namespaces, a component of the application like activities, services, broadcast receivers, and content providers. Permission required by the user for the application features also mentioned in this XML file. Location of the AndroidManifest.xml file:

app -> src -> main

2.8.2.4 themes.xml file

This file defines the base theme and customized themes of the application. It also used to define styles and looks for the UI (User Interface) of the application. By defining styles, we can customize how the views or widgets look on the User Interface. Location of styles.xml file app -> src -> main -> res -> values

2.8.2.5 Drawable XML files

These are the XML files that provide graphics to elements like custom background for the buttons and its ripple effects, also various gradients can be created. This also holds the vector graphics like icons. Using these files custom layouts can be constructed for EditTexts. Location for the Drawable files is:

app -> src -> main -> res -> drawable

2.9 GRADLE

Gradle is a build automation tool known for its flexibility to build software. A build automation tool is used to automate the creation of applications. The building process includes compiling, linking, and packaging the code. The process becomes more consistent with the help of build automation tools.

It is popular for its ability to build automation in languages like Java, Scala, Android, C/C++, and Groovy. The tool supports groovy based Domain Specific Language over XML. Gradle provides building, testing, and deploying software on several platforms.

The tool is popular for building any software and large projects. Gradle includes the pros of Ant and Maven and curbs the cons of both.

A build system like Gradle is not a compiler, linker etc, but it controls and supervises the operation of compilation, linking of files, running test cases, and eventually bundling the code into an apk file for your Android Application.

4.9.2 build.gradle (project level)

The Top level (module) build.gradle file is project level build file, which defines build configurations at project level. This file applies configurations to all the modules in android application project.

4.9.3 build.gradle (application level)

The Application-level build.gradle file is located in each module of the android project. This file includes your package name as applicationID, version name (apk version), version code, minimum and target sdk for a specific application module. When you are including external libraries (not the jar files) then you need to mention it in the app level Gradle file to include them in your project as dependencies of the application.

5. HARDWARE & SOFTWARE REQUIREMENTS

5.1 DEVELOPER REQUIREMENTS:

- Personal Computer or Laptop
- 4 GB RAM
- Android Studio
- Java JDK 15
- Processor: Intel Core i5

5.2 APP USER REQUIREMENTS:

- Android Mobile (minSDK: 21)
- 20MB free space
- Gmail Account
- Internet Connection

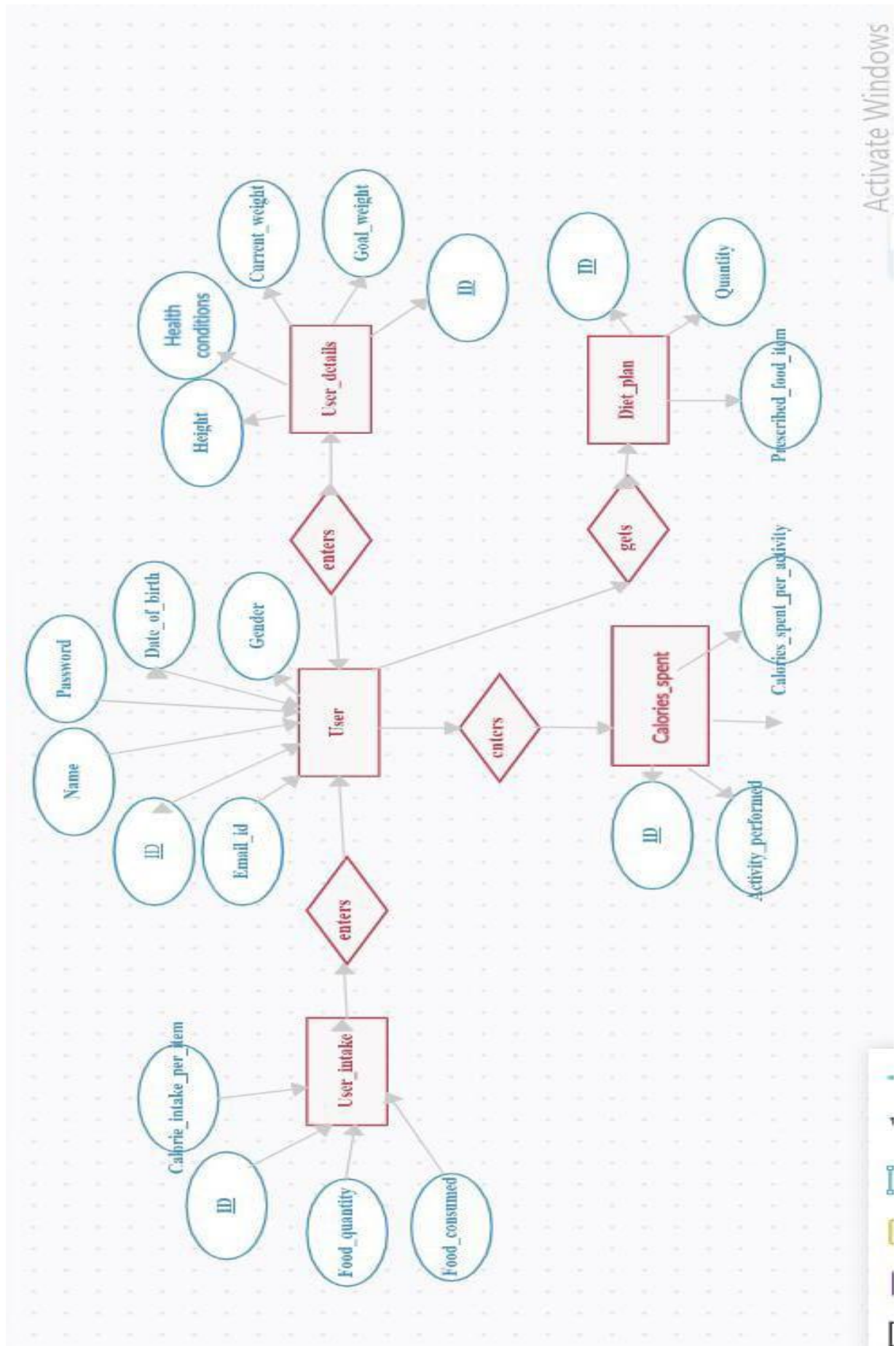
6. ENTITY RELATIONSHIP DIAGRAM

6.1 ER DIAGRAM

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.

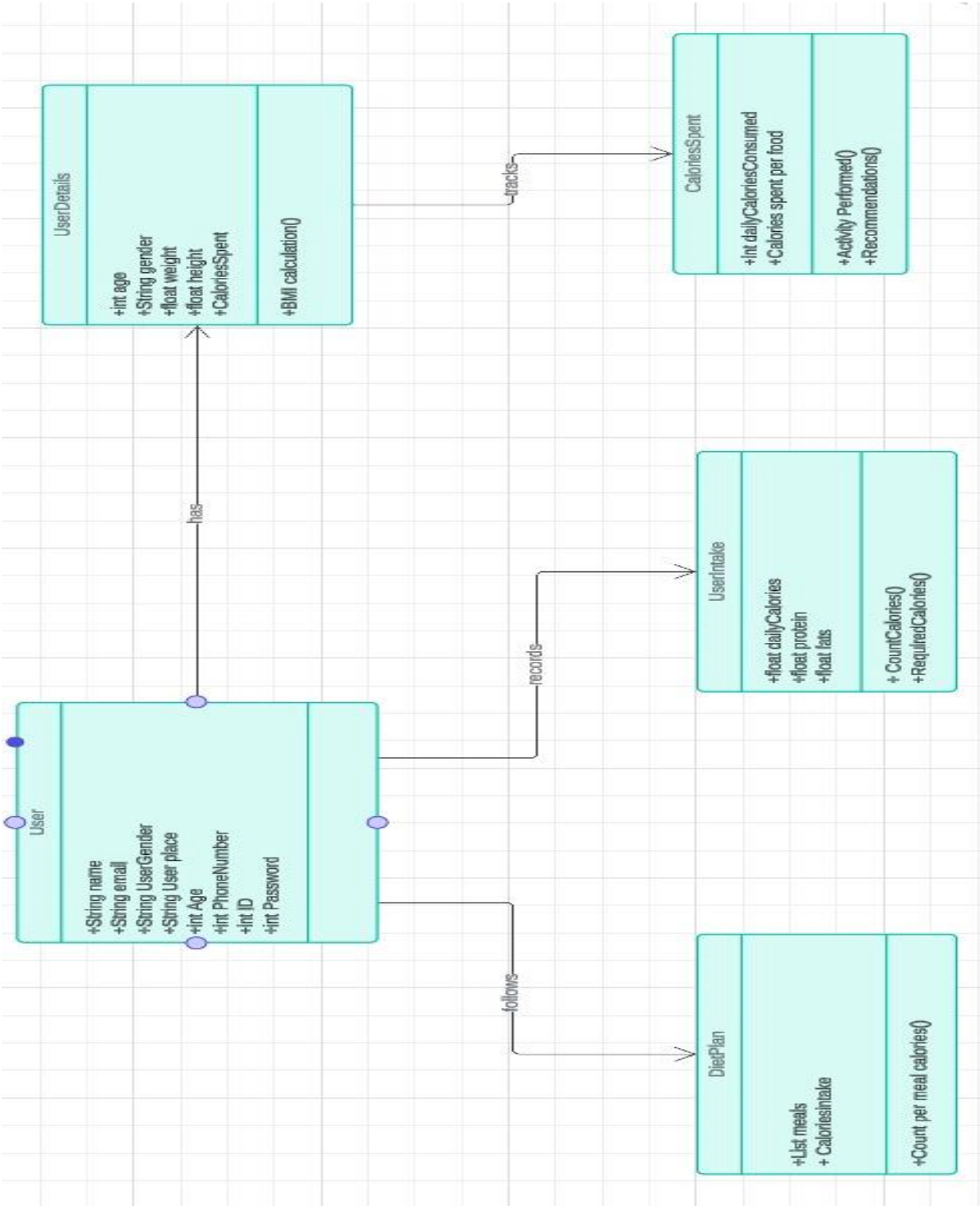


8. CLASS DIAGRAM

8.1 UML CLASS DIAGRAM

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.



9. USE CASE DIAGRAM

9.1 USE CASE DIAGRAM

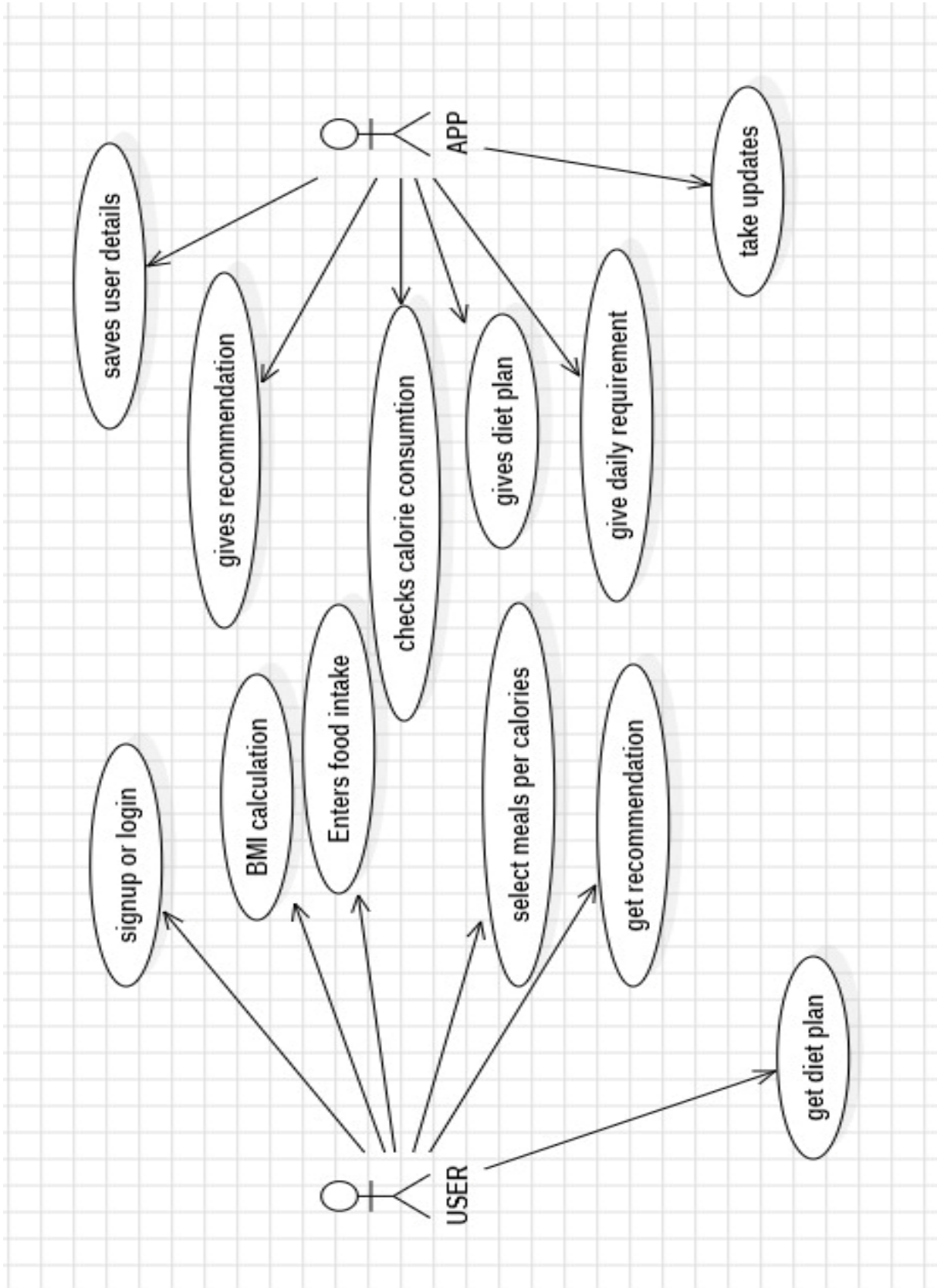
In the Unified Modelling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve □ The scope of your system

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Model the basic flow of events in a use case

USECASE DIAGRAM



10. ACTIVITY DIAGRAM

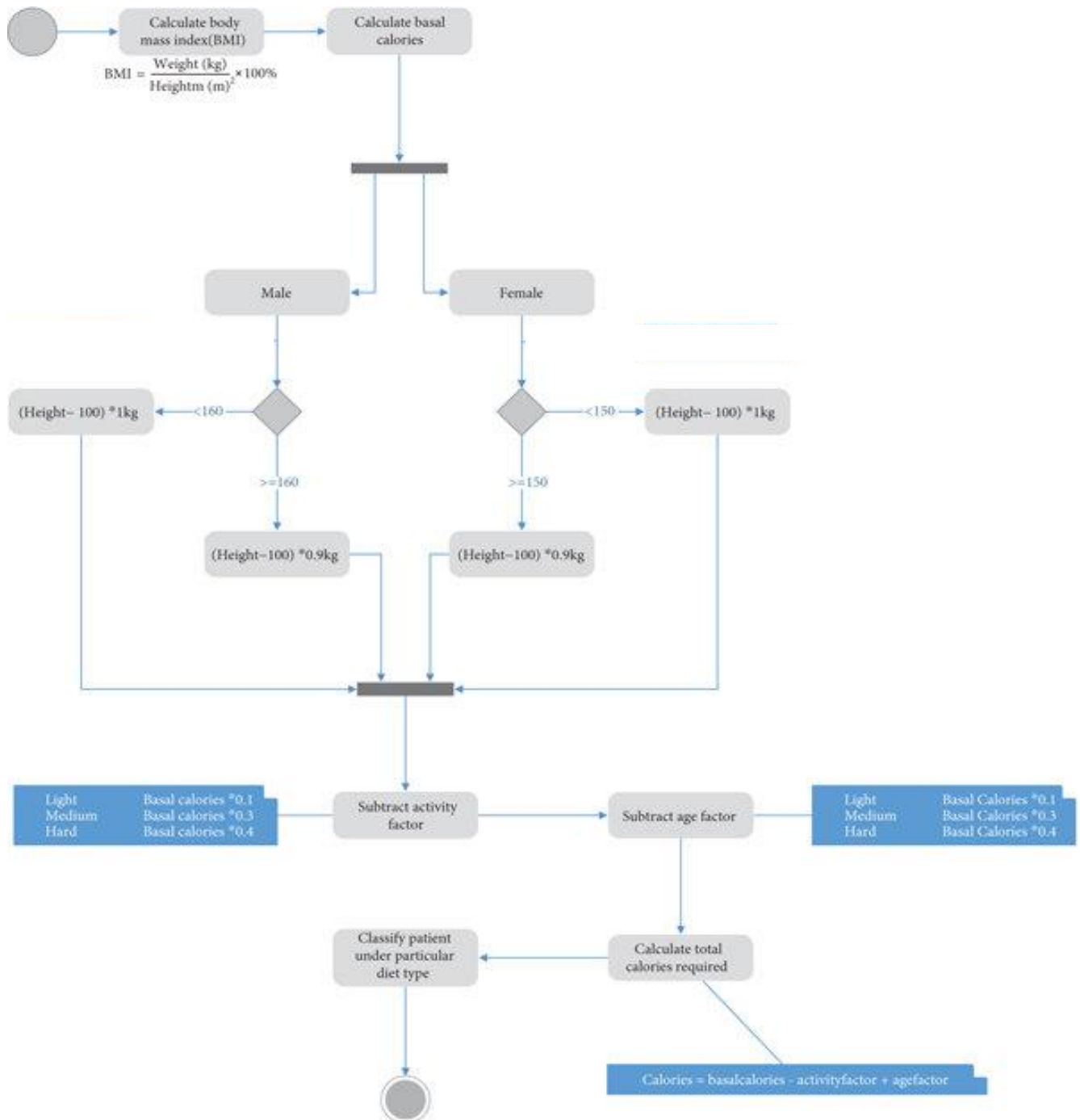
10.1 UML ACTIVITY DIAGRAM

A UML sequence diagram is a type of interaction diagram that visualizes the interactions between objects or components within a system in a chronological order. It illustrates how objects collaborate to achieve a specific functionality or behaviour. Here's some detailed information about UML sequence diagrams:

It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioural diagram.

- **Decision Box:** It makes sure that the control flow or object flow will follow only one path.
- **Action Box:** It represents the set of actions that are to be performed.

ACTIVITY DIAGRAM



11. SEQUENCE DIAGRAM

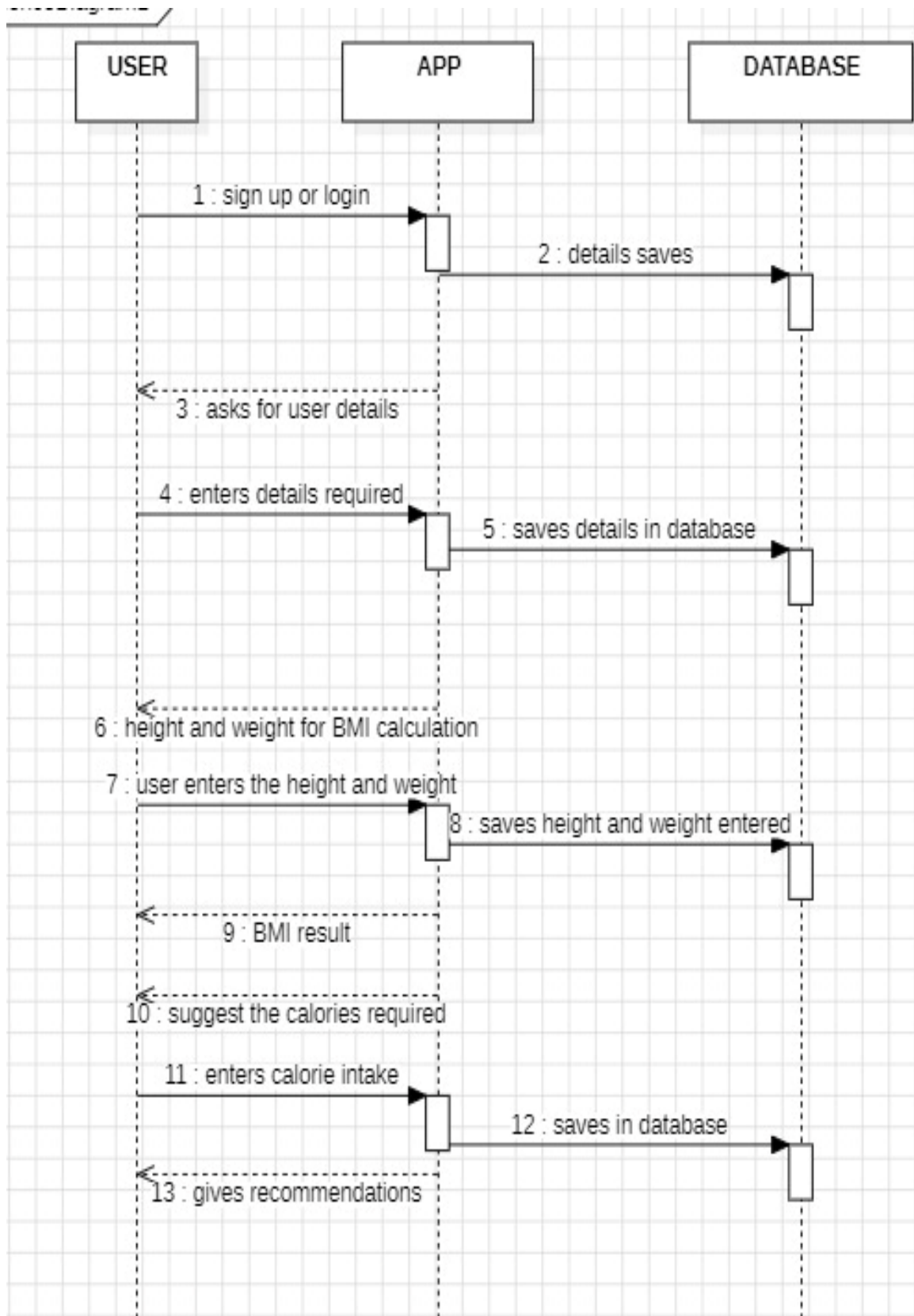
11.1 UML SEQUENCE DIAGRAM

Unified Modelling Language (UML) is a modelling language in the field of software engineering that aims to set standard ways to visualize the design of a system. UML guides the creation of multiple types of diagrams such as interaction, structure, and behaviour diagrams. A sequence diagram is the most commonly used interaction diagram.

An interaction diagram is used to show the interactive behaviour of a system. Since visualizing the interactions in a system can be difficult, we use different types of interaction diagrams to capture various features and aspects of interaction in a system.

A sequence diagram simply depicts the interaction between the objects in a sequential order i.e. the order in which these interactions occur.

We can also use the terms event diagrams or event scenarios to refer to a sequence diagram.



Test Cases

➤ Android Application

1. Unit Testing

Test Case Id: 0001				Test Designed By:			
Low/Medium/High): Medium				Test Designed Date:15/04/24			
Module Name: Status				Test Executed By: Samruddhi kale			
Test Title: Test the working of the app				Test Execution Date:10/05/24			
Description: This test will ensure the all fnctionalities of app is working							
Pre-conditions: system should be properly install in the system.							
Step	Test Case ID	Description	Steps	Input Data	Expected Result	Actual Result	Status
1	TC_1	To test account creation and login	1.Open the app in android phone 2.Click on create account 3. Enter necessary credentials and	Username Name Passwo rd Email Contact no Place	User should able to create account and login with the	User is able to create account and login with same	PASS

BALANACEDBOWL

			create an account 4.loggin into the system	Age Height Weight gender	same account	account	
2	TC _2	To check if user can check BMI	1.Open the app in and android phone 2.click on “viewhealthdet ails” button . 3.Enter weight height to count , add calories intake	Select food intake to Check calories need and check results	Result should be displayed in user profile section	Result is displayed in user profile section	PAS S

Stress Testing:

Test Case Id: 0002			Test Designed By: Shalini jakkula			
Low/Medium/High): Medium			Test Designed Date:16/04/24			
Module Name: Status			Test Executed By: Shalini jakkula			
Test Title: to check whether app works on all devices			Test Execution Date:17/03/24			
Description:This test will ensure the System validation.						
Pre-conditions: Application should be properly installed in the system.						
Step	Test Case ID	Description	Input Data	Expected Result	Actual Result	Status
1	TC_1	Check the project is running on android studio 4.2 or higher	-	It should run	It is running	PASS
2	TC_2	Check the project is running on android mobile with 4GB RAM	-	It should run	It is running	PASS

DATABASE TABLES**TABLES-USERS**

NAME	DATATYPE	SIZE	CONSTRAINTS
NAME	STRING	30	NOT NULL
USERNAME	STRING	30	PRIMARY_KEY
PASSWORD	STRING	30	NOT NULL
EMAIL	STRING	20	NOT NULL
CONTACT	STRING	10	NOT NULL
PLACE	STRING	30	NOT NULL
AGE	STRING	3	NOT NULL
HEIGHT	STRING	3	NOT NULL
WEIGHT	STRING	3	NOT NULL
GENDER	STRING	7	NOT NULL

TABLE-DAILYCONS

NAME	DATATYPE	SIZE	CONSTRAINTS
USERNAME	STRING	30	FOREINGN KEY
CALORIES	STRING	4	NOT NULL

TABLE DESIGN

1.USERS TABLE

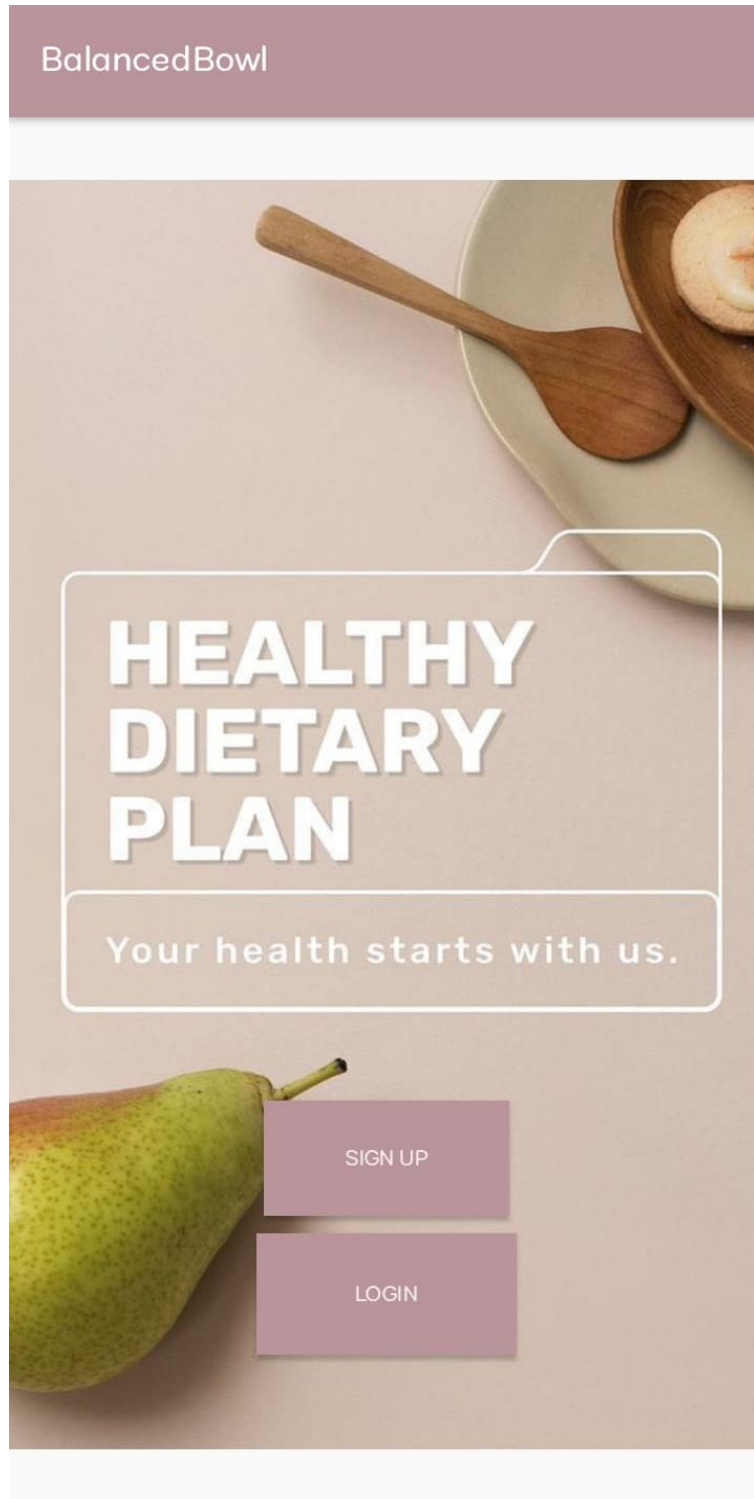
- name
- username (primary key)
- password
- email
- contact
- place
- age
- height
- weight
- gender

2.DAILYCONS TABLE

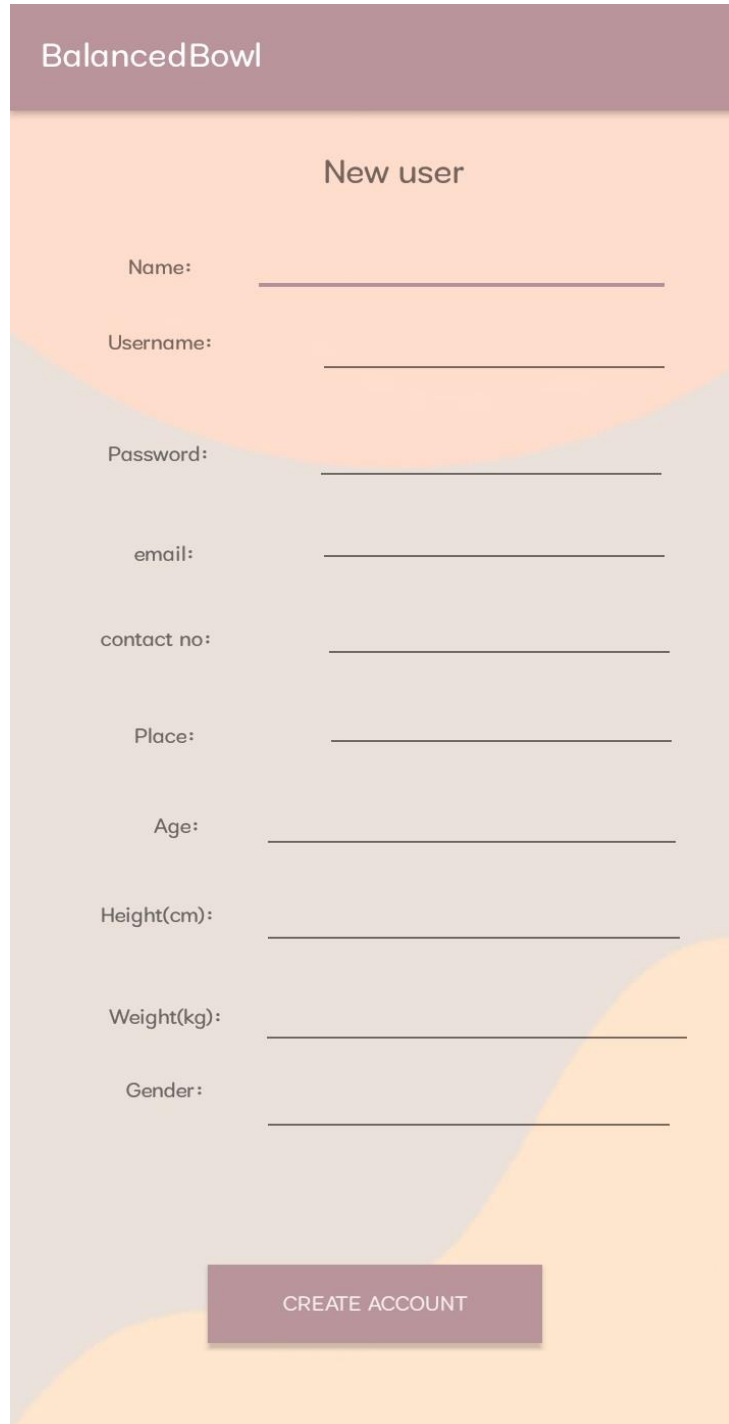
- username (foreign key)
- calories

11. APPLICATION SCREENSHOTS

11.1 HOME SCREEN



11.2 SIGNUP SCREEN



The image shows a mobile app signup screen for 'BalancedBowl'. The app name is in a dark purple header. Below it, the title 'New user' is centered. The form consists of ten input fields, each with a label to its left: Name, Username, Password, email, contact no, Place, Age, Height(cm), Weight(kg), and Gender. At the bottom, there is a dark purple button with the text 'CREATE ACCOUNT' in white. The background features a light orange gradient with abstract wavy shapes in shades of grey and orange.

BalancedBowl

New user

Name:

Username:

Password:

email:

contact no:

Place:

Age:

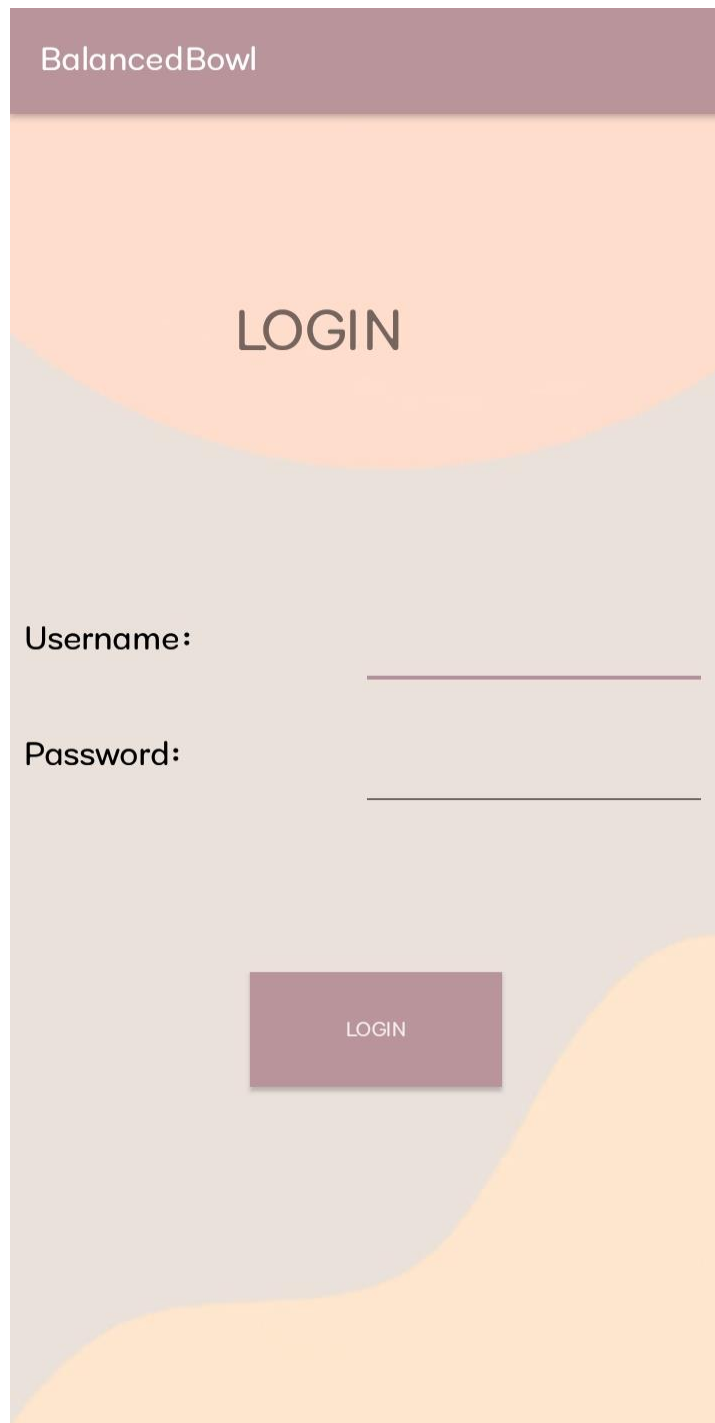
Height(cm):

Weight(kg):

Gender:

CREATE ACCOUNT

11.3 LOGIN SCREEN



The login screen features a dark purple header with the text "BalancedBowl" in white. Below the header is a large orange section with the word "LOGIN" in dark grey. The background of the screen is a light grey with a large, curved, light orange shape on the right side. The login form consists of two labels, "Username:" and "Password:", followed by input fields. A dark purple "LOGIN" button is positioned below the input fields.

BalancedBowl

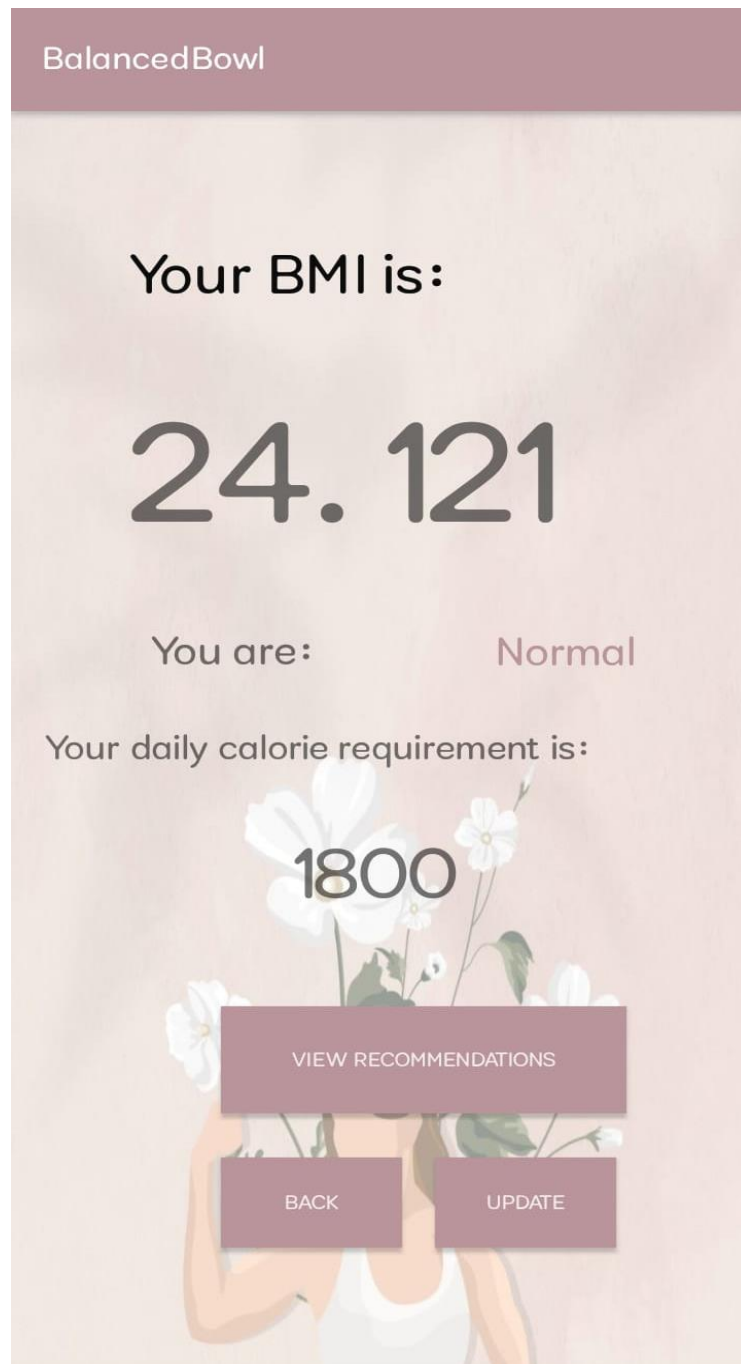
LOGIN

Username:

Password:

LOGIN

11.4 DETAILS SCREEN



11.5 RECOMMENDATION ACTIVITY



11.6 UPDATE Activity



The image shows a mobile app interface for 'BalancedBowl'. At the top is a dark purple header with the text 'BalancedBowl' in white. Below the header, the background is a light pinkish-beige with a faint illustration of a woman holding a bouquet of white flowers. The text 'Enter new height and weight :' is centered. Below this, there are two input fields. The first is labeled 'Height:' and has a purple underline. The second is labeled 'Weight:' and has a black underline. At the bottom, there are two purple buttons: 'BACK' on the left and 'OK' on the right.

BalancedBowl

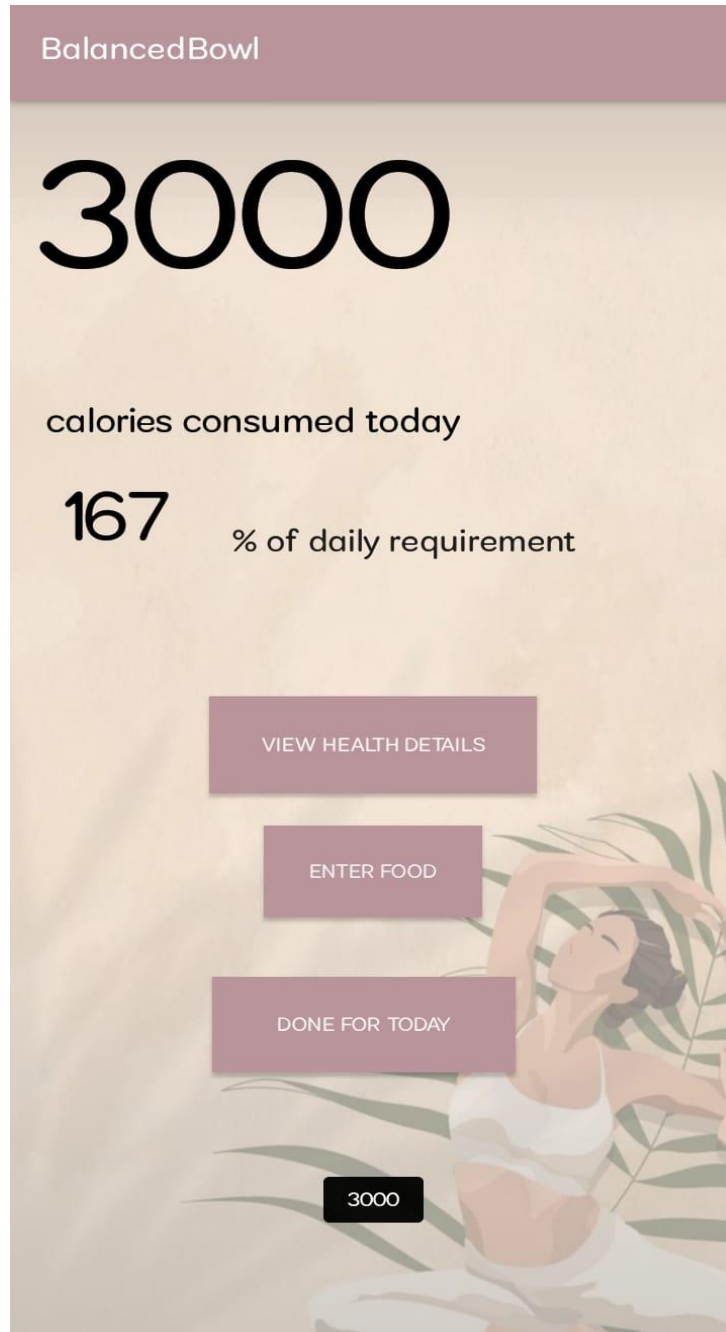
Enter new height and weight :

Height:

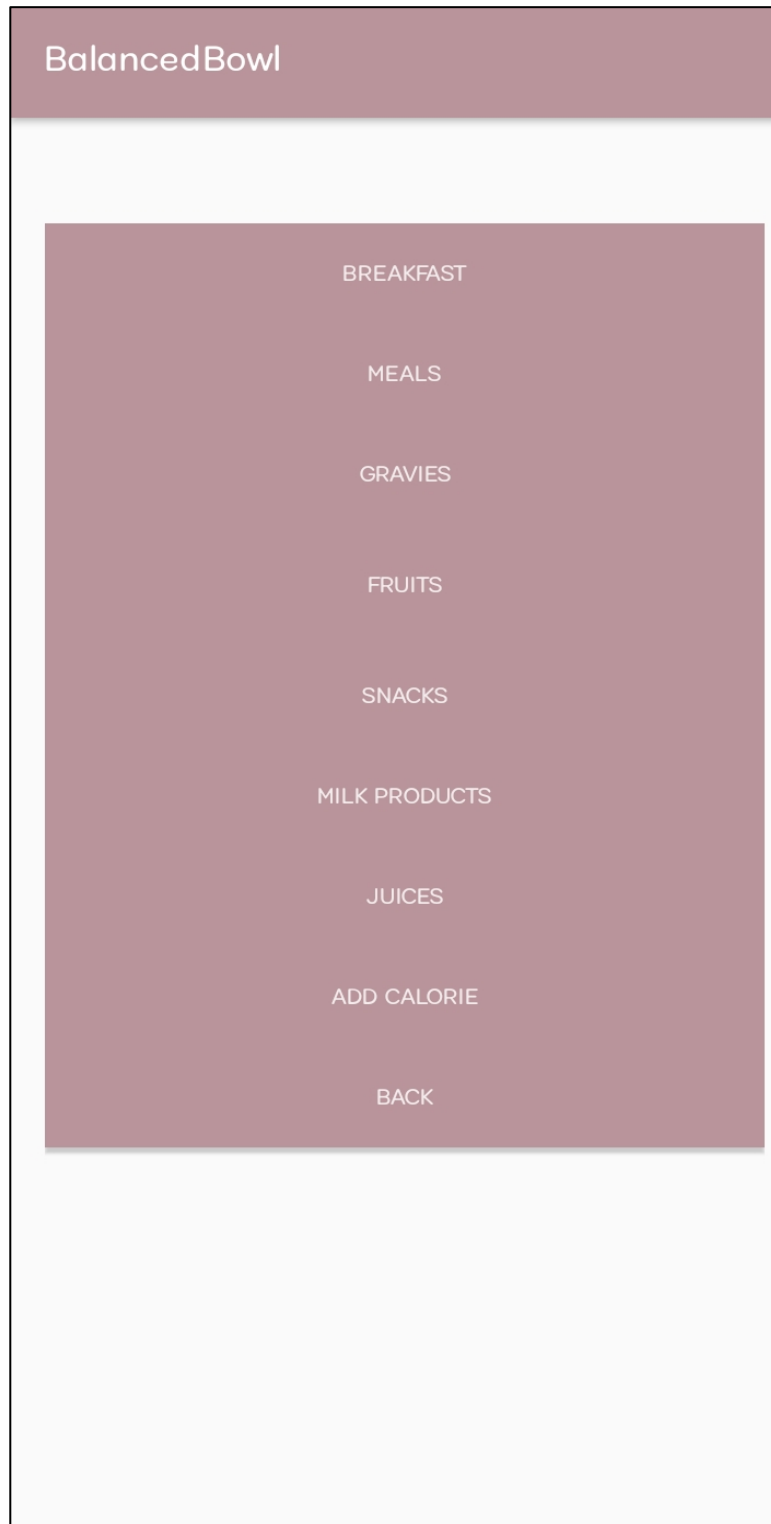
Weight:

BACK OK

11.7 CALORIES CONSUMED ACTIVITY








11.8 FOOD INTAKE ACTIVITY









11.9 ENTER EXTRA CALORIE ACTIVITY







BalancedBowl

	Name: apple Calories: 48
	Name: banana Calories: 93
<div>Enter quantity consumed in ml: 200</div> <div>CANCEL OK</div>	
	Name: mango Calories: 61
	Name: orange Calories: 47
	Name: papaya Calories: 60

ENTER FOOD YOU CONSUMED ACTIVITY







BalancedBowl	
	Name:apple Calories:95
	Name:banana Calories:105
	Name:dates Calories:23
	Name:grapes Calories:67
	Name:mango Calories:201
	Name:orange Calories:45

11.11 ENTER FOOD YOU CONSUMED







BalancedBowl	
	Name: bread Calories: 80
	Name: chappathi Calories: 95
	Name: dosa Calories: 120
	Name: idli Calories: 39
	Name: parotta Calories: 120
	Name: pongal Calories: 212

Figure

11.11 ENTER FOOD CONSUMED

BalancedBowl	
	Name:biriyani chicken Calories:139
	Name:biriyani fish Calories:112
	Name:biriyani mutton Calories:145
	Name:biriyani vegetable Calories:130
	Name:fried rice Calories:163
	Name:noodles chicken







11.11 ENTER FOOD CONSUMED

BalancedBowl	
	Name:butter Calories:737
	Name:buttermilk Calories:40
	Name:cheese Calories:402
	Name:curd Calories:98
	Name:ghee Calories:876
	Name:milk Calories:44

Figure

11.11 ENTER FOOD CONSUMED**BalancedBowl****Name:biscuit****Calories:40****Name:chips****Calories:530****Name:chocolate****Calories:526****Name:puffs****Calories:61****Name:samosa****Calories:230**

11.11 ENTER FOOD CONSUMED

BalancedBowl	
	Name:apple Calories:48
	Name:banana Calories:93
	Name:lemon Calories:22
	Name:mango Calories:61
	Name:orange Calories:47
	Name:papaya Calories:60

Figure

11.11 ENTER FOOD CONSUMED

Name:chicken gravy
Calories:79



Name:coconut chutney
Calories:200



Name:paneer butter masala
Calories:229



Name:rasam
Calories:25

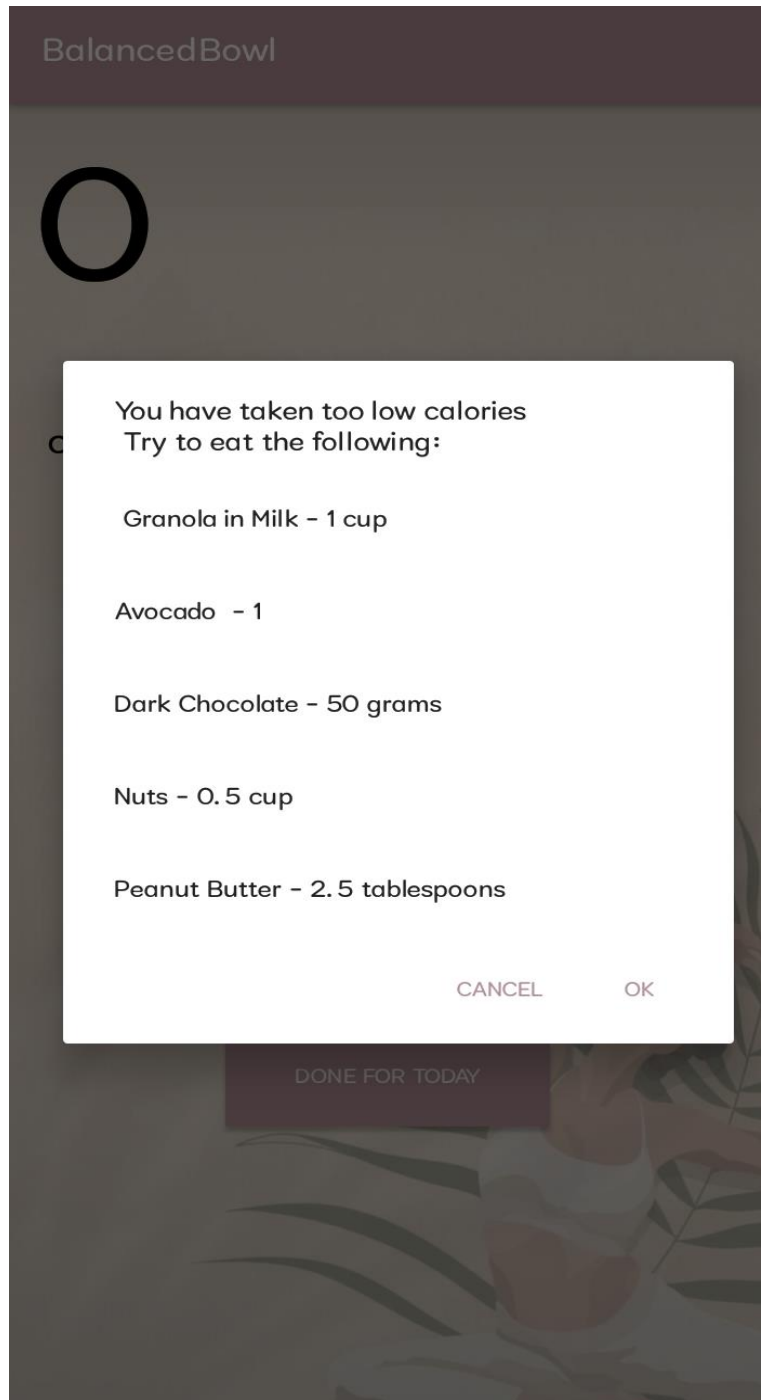


Name:sambar
Calories:114



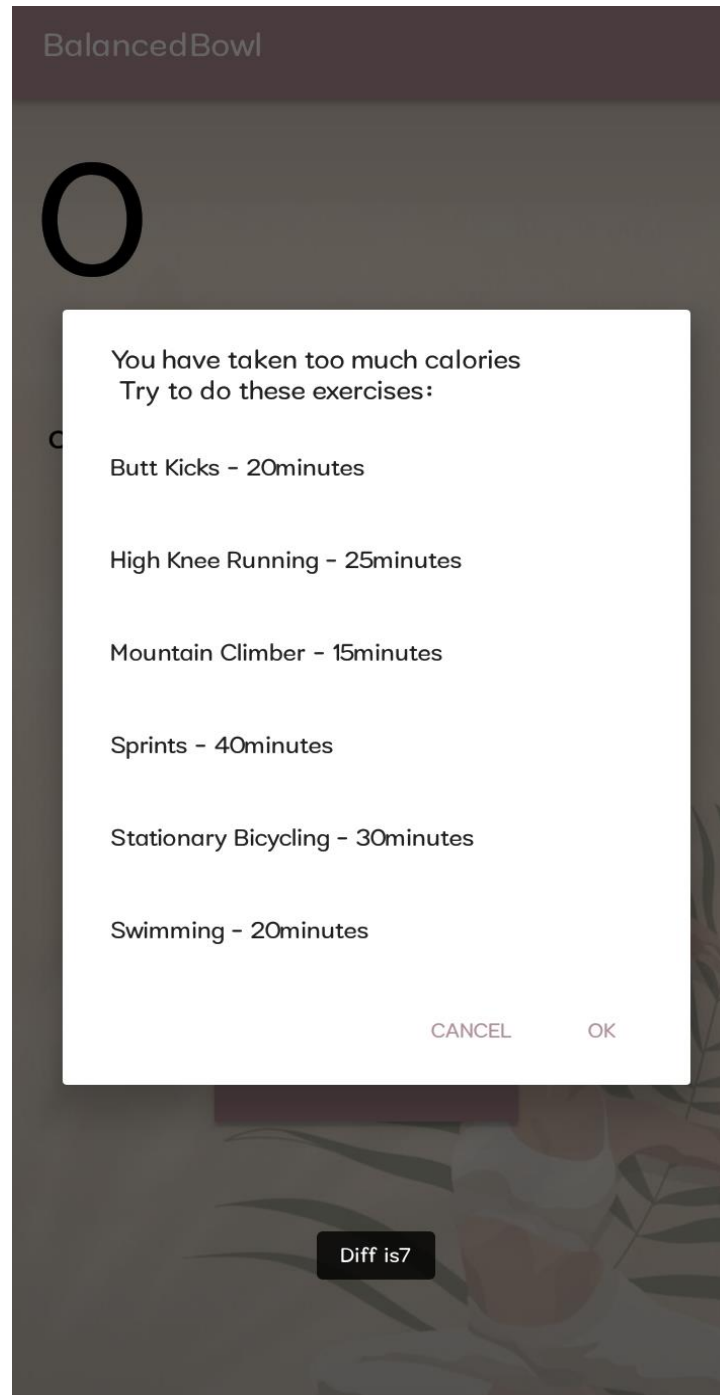
Name:tomato chutney
Calories:70

11.11 SUGGESTIONS IF CALORIES INTAKE IS LOW



Figure

11.15 EXERCISE SUGGESTION ACTIVITY



12. LIMITATIONS & BOUNDARIES

- The App “BALANACEDBOWL” requires a stable space to be able to access profile & storage.
- Manual entries of health records are required.
- User needs to have an account.
- Only one user per account is allowed.
- The intended system is aimed at personal use momentarily.
- No real meetings.

13. ADVANTAGES

- Easy to install App APK.
- Easy to user Interface.
- This application is supported on devices with Android Version 5.0 (Jellyfish) and above which makes it compatible on 98.6 % Android devices worldwide.
- User can login and Sign In.
- They help users become more aware of their food choices and portion sizes, leading to better understanding of their calorie intake.
- Users can track their daily calorie consumption, promoting accountability and helping them stay on track with their dietary goals.
- Access to an individual's health record is made available in a single location on the Android device.
- Users can track their progress over time, seeing trends in their calorie intake and making adjustments as needed to achieve their goals.
- These apps allow users to set personalized calorie goals based on factors like weight, activity level, and health goals, providing a clear target to work towards.
- Some apps offer social features or coaching support, providing motivation and encouragement to users on their journey towards better calorie management and overall health.

14.FUTURE SCOPE & ENHANCEMENT

The future scope for a “BALANACEDBOWL” are:

1. **Personalization:** Enhanced personalization based on users' specific dietary needs, preferences, and goals, such as weight loss, muscle gain, or managing medical conditions like diabetes or hypertension.
2. **Integration with Wearable Tech:** Integration with wearable devices to track physical activity levels and provide real-time adjustments to calorie recommendations and meal plans.
3. **Nutrient Tracking:** Beyond just calories, tracking of macronutrients (carbohydrates, proteins, fats) and micronutrients (vitamins, minerals) to ensure users are meeting their nutritional needs.
5. **Community and Social Features:** Incorporating social features such as user communities, sharing meal plans and recipes, and connecting with nutritionists or dietitians for personalized advice and support.
6. **Integration with Healthcare Providers:** Integration with healthcare providers to facilitate remote monitoring and management of chronic conditions through diet and lifestyle interventions.

14.FEATURES

- Integrating features into your diet app that suggest exercise when users exceed their daily calorie intake and recommend food items to reach their calorie needs when intake is insufficient can greatly enhance its effectiveness in supporting users' health and fitness goals. Here's how you can implement these features:

- Daily Calorie Calculation:

Calculate users' daily calorie needs based on their profile information, including age, gender, weight, height, activity level, and goals (e.g., weight loss, maintenance, or gain). Provide users with their personalized calorie targets to guide their dietary intake.

- Calorie Tracking:

Allow users to log their daily food intake within the app to track their calorie consumption. Provide a database of foods with calorie information to facilitate logging.

- Calorie Comparison and Feedback:

Compare users' logged calorie intake against their daily calorie targets. If users exceed their calorie goals, provide feedback indicating that they have consumed more calories than needed for the day.

- Exercise Suggestions for Excess Calories:

When users exceed their daily calorie targets, suggest appropriate types and durations of exercise to help offset the excess calories and maintain overall calorie balance. For example:

- Recommend cardiovascular exercises like walking, jogging, cycling, or swimming to burn additional calories.

➤ Food Recommendations for Calorie Deficiency:

If users' logged calorie intake is below their daily calorie targets, suggest specific food items or meals to help them reach their calorie needs. Consider recommending nutrient-dense foods that provide essential nutrients and energy without excessive added sugars or unhealthy fats.

➤ 7. Meal Suggestions and Recipes:

Offer personalized meal suggestions and recipes based on users' dietary preferences and calorie requirements. Include options for breakfast, lunch, dinner, and snacks to ensure balanced nutrition throughout the day.

➤ 8. Nutritional Guidance:

Provide educational resources and tips on how to achieve and maintain optimal calorie balance through diet and exercise. Offer guidance on portion control, meal planning, and mindful eating practices.

15. CONCLUSION

15.1 APPLICATION OVERVIEW

In conclusion, the “BALANACEDBOWL” app emerges as a holistic solution for individuals striving to attain their health and fitness goals through tailored nutrition management. By seamlessly integrating BMI calculation into its framework, the app offers users a comprehensive understanding of their body composition, empowering them to make informed decisions about their dietary needs. Through sophisticated algorithms and nutritional expertise, the app provides personalized recommendations for calorie intake, ensuring that users meet their individual health objectives in a sustainable manner.

Moreover, the inclusion of customizable diet plans, enriched with diverse recipes and food options, facilitates adherence to healthy eating habits. The “BALANACEDBOWL” app is

user-friendly interface and interactive features further enhance the user experience, fostering motivation and accountability on the journey towards improved well-being. In essence, the diet planner app serves as a trusted companion in the pursuit of a balanced lifestyle, guiding users towards optimal health and vitality."

"BALANACEDBOWL" focuses on the How to Eat More, Exercise Less, Lose Weight, and Live Better.

15.2 APPLICATION FEATURES:

- ❖ **User Friendly:** The application interface has been made in a way to enable user to interact smoothly and efficiently with the app.
- ❖ **Authentication:** The integration of Database in the app enables proper Google authentication for signing in and stores user data securely.
- ❖ **Ease of Access:** Being able to use this application on the android device with a simple login, enables user to access their records from anywhere with just an android device and an account.
- ❖ **Storage efficiency:** The app makes use of Database storage to store user data; therefore, it does not use the physical device storage and hence occupies less than 15 MB space on the device.
- ❖ **Ease of Development:** The application development has followed all java convention, which makes it easy for developers to understand and increase scope of enhancement due to widely available resources, also making this application Platform independent.

16. BIBLIOGRAPHY

16.1 WEBSITES:

1. <https://github.com/>
2. <https://stackoverflow.com/>
3. <https://developer.android.com/studio>
4. <https://developer.android.com/docs>
5. <https://www.nutrition.gov/health-information>
6. <https://www.health.com/>
7. [https://www.canva.com/en_gb/?utm=a949d0c1c313da820d8cd0753371145b
&track=1&pt=2](https://www.canva.com/en_gb/?utm=a949d0c1c313da820d8cd0753371145b&track=1&pt=2)
8. <https://www.java.com/en/>
9. <http://www.sqlite.org/>

16.2 REFERENCES:

1. Android App Development in Android Studio by J.Paul Cardle
2. The Definitive Guide to Firebase by Laurence Moroney
3. Android Studio 3.0 Development Essentials by Neil Smith
4. Head First Android Development by Dawn Griffiths & David Griffiths
5. "The Calorie Myth": How to Eat More, Exercise Less, Lose Weight, and Live Better" by Jonathan Bailor.
6. EAT THIS, NOT THAT! By David Zinczenko and Matt Goulding.