Project Report On

“Budget Buddy”

completed at

O7 Services

Submitted for the partial fulfilment for the award of the degree of **Diploma**

IN

(Computer Science & Engineering)

Submitted To :

Prince Madaan Sir

Submitted By

Sahil (551/22)

Maansikander Singh (535/22)

ABSTRACT

In this work, we present the design and implementation of the Brain Burst App. This app is made using Android. Android, a revolutionary UI toolkit developed by Google, has emerged as a game-changer in the realm of cross-platform app development. This abstract provides an

overview of Android's key features, advantages, and its transformative impact on the app development landscape. Quiz Master is an interactive and engaging quiz application designed to entertain and educate users on various topics. Built using Kotlin, Quiz Master offers a seamless user experience with a sleek and intuitive interface. Users can choose from a wide range of quiz categories, including general knowledge, science, history, entertainment, and more. users are greeted with a home screen displaying different quiz categories. They can select a category of their choice to begin the quiz. Each quiz consists of multiple-choice questions with four options, and users must select the correct answer within a time limit.1. Introduction to the Industry / Institution Contents

1.1 Nature of business of the Industry / Institution

O7 Services is an ISO 9001:2015 Certified Organization. Fundamentally, deal in Web Designing, Mobile Development, Custom Software Development, Web Designing, Graphic Designing, Hosting services, Digital Marketing, Registration of Domain Names with the various latest extensions, Internet and Social Media Marketing, Search Engine Optimization (SEO), Pay Per Click (PPC) Management, and give the most experienced IT solutions, supporting the full business cycle: preliminary counselling, framework development and deployment, quality confirmation, and 24x7 backings.

With a rich encounter of over 8 years, O7 Services is in general form dependable key organisation with their clients to guarantee affordable costs, opportune conveyance, and quantifiable business results. The Head Office of O7 Services is in Jalandhar and the Branch Office is in Hoshiarpur

With more than 30 team members as of the year 2023, they have provided services to many major and minor organisations, start-ups, and public and private establishments. The vision of O7 Services is to help their clients with figuring out their fantasies and work with them with prime administrations that beat their assumptions, and hence, add to a prevalent society for their clients and various accomplices as well as broad society. They will continue with their drive to grasp this vision.

1.2 Different products / activities of the Industry / Institution

Since its establishment, O7 Services has worked with a wide range of products from Learning Management System (LMS) to custom ERP software. It has provided its support to the various organisations like LMS, E-commerce websites, Betting Strategy websites, Tracking Systems, School Management System, Informative Websites, HR ERP Softwares. The company of O7 Services has worked very closely with innumerable customers by providing the best of the work. Some of the company products and websites that are designed by O7

Services are as follows-


A.C.C.
JEWELLERS LTD.
SPECIALIZING IN 22K GOLD


APEX®
SINCE 1959


TRUMATH


ERYO
Parks


astro river.com
MASTERING PLANETS, CHANGING LIVES


CAPITOL
HOSPITAL
NURTURING HEALTH . TOUCHING LIFE


HR®
The Tool Co.


KSE
Krishna Smart Education


FesoMe
FASHIONABLE ME


Mekmetme


VICTORY WITH DETERMINATION
SSGI
GROUP OF INSTITUTION


SB
Sales & Buys
www.salesandbuys.in


SUD
Punjabi suit manufacturer


TRACKING


GURMAT JEEVAN
JAACH


BRAND K


CENTURION
IMMIGRATION CONSULTANCY


Nikhil PTE
Tips 'N' Tricks


RightClickJobs
A JOB PORTAL FOR FRESHER'S


THE
GRAND
HOTELS & RESORTS


G.N.B.I RAMGARHIA
COLLEGE FOR WOMEN
PHAGWARA

TABLE OF CONTENTS

CHAPTER - I
INTRODUCTION

1.1Project Overview:

In today's world, Smart phones have changed our lives and have become an indispensable part of our lives because of its specialty to simplify our routine work and thereby saving our time. A Smartphone with an Android OS offers excellent functionality to the users offering a distinct experience. Android is a Linux based operating system and it was bought by Google in 2007.There are tons of application available and one of the prime reason for this vast number is android being an open source. On the other hand, android based device like mobile, tab are very user friendly. A survey has done by "Light Castle Partners" research wing which indicates that though other operating system mobile users exist but the majority users are goes with android operating system. In this context, Project application is developed based on android platform. The name of application is define as 'Budget Buddy'.

1.2Objectives:

The primary objectives of the budget tracking application are as follows:

- **To Provide Real-Time Financial Tracking**: Allow users to input and categorize their income and expenses to monitor their financial status in real-time.
- **To Facilitate Budget Creation**: Enable users to set monthly or yearly budgets and track their adherence to these budgets.
- **To Offer Detailed Financial Reports:** Generate insightful reports and visualizations to help users understand their spending patterns and make informed financial decisions.
- **To Enhance Financial Planning:** Assist users in planning for future expenses and savings goals by providing projections and financial forecasts**.**

1.3 **Motivation**

The motivation behind developing this budget tracking application stems from the increasing need for effective personal finance management tools. Many individuals struggle with tracking their spending and managing their budgets due to a lack of user-friendly tools that offer both simplicity and comprehensive functionality. This application aims to bridge that gap by offering an intuitive interface, powerful features, and actionable insights, ultimately empowering users to take control of their finances and improve their financial health.

1.4 Key **Features:**

The key features of the budget tracking application include:

- Income and Expense Tracking: Users can easily record their income and expenses, categorize them, and view a summary of their financial activity.
- Budget Management: The application allows users to set and manage budgets for different categories and track their progress against these budgets.
- Expense Categorization: Expenses can be categorized into predefined or custom categories, providing a clearer view of spending patterns.
- Financial Reports and Charts: Users can access detailed reports and visualizations, such as pie charts and bar graphs, to analyze their financial data.

1.5 Existing System

Existing budget tracking applications typically offer a range of features to help users manage their finances. Common features include:

- Transaction Entry: Allows users to input and categorize their transactions.
- Budget Tracking: Users can set budget limits for various categories and track their spending against these limits.
- Financial Summaries: Provides an overview of income, expenses, and budget adherence.
- Report Generation: Generates reports and charts to visualize financial data and spending patterns.

- **Expense Categorization**: Helps users categorize their expenses for better tracking and analysis.

1.6 Drawbacks of Existing System

- **Complexity**: Some applications are overly complex, making it difficult for users to navigate and utilize all features effectively.
- **Limited Customization:** Many applications lack flexibility in categorizing expenses or setting up personalized budgets.
- **Insufficient Reporting**: Existing reports may not offer sufficient detail or customization options for users to fully understand their financial data.
- **Integration Issues:** Some applications do not integrate well with external financial accounts or services, leading to incomplete data.
- **User Experience:** Poor user interface design and lack of intuitive navigation can hinder the overall user experience.

By addressing these drawbacks, the budget tracking application aims to provide a more user-friendly and comprehensive solution for personal finance management.

CHAPTER - II

TECHNOLOGY USED

Kotlin is a modern, trending programming language that was released in 2016 by JetBrains. It has become very popular since it is compatible with Java (one of the most popular programming languages out there), which means that Java code (and libraries) can be used in Kotlin programs. Kotlin is a statically typed programming language developed by JetBrains, the creators of IntelliJ IDEA, and it's now officially supported for Android development by Google.

(Fig1.1)

1. **Conciseness**: Kotlin aims to reduce boilerplate code, which means you can accomplish the same tasks with fewer lines of code compared to Java.

2. **Interoperability**: Kotlin is fully interoperable with Java. This means you can use Kotlin alongside existing Java code, making migration to Kotlin easier for existing Java projects.

3. **Null Safety**: Kotlin's type system is designed to eliminate the risk of null pointer exceptions. Types in Kotlin are non-nullable by default, and you have to explicitly specify when a type can be null.

4. **Extension Functions**: Kotlin allows you to extend existing classes with new functionality without modifying their code. This feature is called extension functions.

5. **Functional Programming Support**: Kotlin has robust support for functional programming concepts like higher-order functions, lambdas, and immutability. This enables you to write more concise and expressive code.

6. **Coroutines**: Kotlin provides first-class support for coroutines, which are a way to perform asynchronous programming. Coroutines make it easier to write asynchronous code in a sequential style, avoiding the callback hell associated with traditional approaches.

7. **Smart Casts**: Kotlin's type system includes smart casts, which automatically cast types when certain conditions are met. This reduces the need for explicit type casting in your code.

8. **Data Classes**: Kotlin provides a concise way to create classes whose primary purpose is to hold data. These are called data classes, and Kotlin automatically generates **equals()**, **hashCode()**, **toString()**, and **copy()** methods for them.

9. **Companion Objects**: Kotlin supports companion objects, which are similar to static methods and fields in Java. They allow you to define methods and properties that are tied to a class rather than to instances of the class.

10. **Immutable Collections**: Kotlin provides immutable collections by default, which means that once you create a collection, you cannot modify it. This helps prevent accidental modification of data.

Kotlin is used for:

- Mobile applications (especially Android apps)
- Server-side applications
- Data science

- And much, much more!

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of Kotlin's standard library depends on the Java Class Library, but type inference allows its syntax to be more concise. Kotlin mainly targets the JVM.

2.1 OOPS concepts

OOPs stands for **Object-Oriented Programming**. Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs

1. Class: Classes in Kotlin are declared using the keyword class:

The class declaration consists of the class name, the class header (specifying its type parameters, the primary constructor, and some other things), and the class body surrounded by curly braces. Both the header and the body are optional; if the class has no body, the curly braces can be omitted. For example, class Empty {....}

2. Objects: It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Kotlin program creates many objects, which as you know, interact by invoking methods. An object consists of:

● **State:** It is represented by attributes of an object. It also reflects the properties of an object. ● **Behaviour:** It is represented by methods of an object. It also reflects the response of an object with other objects.

- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

3. Inheritance: Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

2.2 BASICS OF KOTLIN

Kotlin is a modern, trending programming language. Kotlin is easy to learn, especially if you already know Java (it is 100% compatible with Java). Kotlin is used to develop Android apps, server-side apps, and much more.

1. Kotlin Syntax

The fun keyword is used to declare a function. A function is a block of code designed to perform a particular task. In the example above, it declares the main() function. The main() function is something you will see in every Kotlin program. This function is used to execute code. Any code inside the main() function's curly brackets {} will be executed.

```
fun    main(){
println(“Hello World”) }
```

2. Kotlin Variables

Variables are containers for storing data values .To create a variable, use var or val, and assign a value to it with the equal sign (=):

- val: When you create a variable with the val keyword, the value cannot be changed/reassigned.
- var: When you create a variable with the var keyword, the value can be changed/reassigned.

```
var variableName = value val
```

variableName = value

3. Kotlin Data Types

In Kotlin, the type of a variable is decided by its value:

```
val myNum=5 // Int 7 val
myDoubleNum=5.99 // Double val
myLetter='D' // Char val
myBoolean =true // Boolean val
myText="Hello" // String
```

4. Kotlin Operator

• Arithmetic Operators

* Multiplication

/ Division

+ Addition

- Subtraction

% Remainder/Modulo

2.3 Activity:-

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with R.attr.windowIsFloating set), Multi-Window mode or embedded into other windows.

There are two methods almost all subclasses of Activity will implement:

1. **onCreate(Bundle)** is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using

findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.

2. **onPause()** is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the ContentProvider holding the data). In this state the activity is still visible on screen.

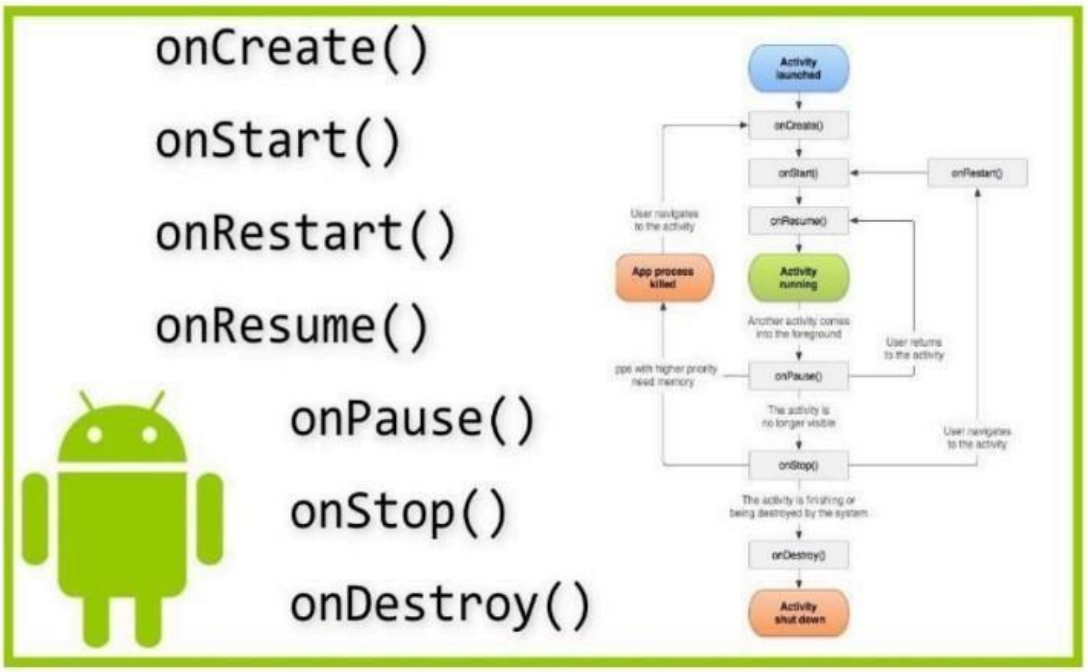


Fig 2.1 Activity

2.4 Layouts in Android:-

1. Linear Layout:-

Linear Layout is a view group that aligns all children in a single direction, vertically or Horizantally. You can specify the layout direction with the android: orientation attribute. All children of a Linear Layout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding). A Linear Layout respects margins between children and the gravity (right, center, or left alignment) of each child.

1.1 **Layout Weight**: Linear Layout also supports assigning a weight to individual children with the android: layout_weight attribute. This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A larger weight value allows it to expand to fill any remaining space in the parent view. Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight. Default weight is zero.

1.2 Equal distribution: To create a linear layout in which each child uses the same amount of space on the screen, set the android: layout_height of each view to "0dp" (for a vertical layout) or the android: layout_width of each view to "0dp" (for a horizontal layout). Then set the android:layout_weight of each view to "1". Unequal distribution: You can also create linear layouts where the child elements use different amounts of space on the screen:

• If there are three text fields and two of them declare a weight of 1, while the other is given no weight, the third text field without weight doesn't grow. Instead, this third text field occupies only the area required by its content. The other two text fields, on the other hand, expand equally to fill the space remaining after all three fields are measured.

The following code snippet shows how layout weights might work in a "send message" activity. The To field, Subject line, and Send button each take up only the height they need.
This configuration allows the message itself to take up the rest of the activity's height.

2. Relative Layout

Relative Layout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or centre).

- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.

RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from RelativeLayout.LayoutParams.

Some of the many layout properties available to views in a RelativeLayout include:

- **android:layout_alignParentTop:** If "true", makes the top edge of this view match the top edge of the parent.
- **android:layout_centerVertical:** If "true", centers this child vertically within its parent.
- **android:layout_below:** Positions the top edge of this view below the view specified with a resource ID.
- **android:layout_toRightOf:** Positions the left edge of this view to the right of the view specified with a resource ID.

The value for each layout property is either a boolean to enable a layout position relative to the parent RelativeLayout or an ID that references another view in the layout against which the view should be positioned.

In your XML layout, dependencies against other views in the layout can be declared in any order. For example, you can declare that "view1" be positioned below "view2" even if "view2" is the last view declared in the hierarchy. The example below demonstrates such a scenario.

3. Constraint Layout

A Constraint Layout is a ViewGroup which allows you to position and size widgets in a flexible way. ConstraintLayout is available as a support library that you can use on Android systems starting with API level 9 (Gingerbread). As such, we are planning on enriching its API and capabilities over time. This documentation will reflect those changes. Relative positioning is one of the basic building blocks of creating layouts in ConstraintLayout. Those constraints allow you to position a given widget relative to another one. You can constrain a widget on the horizontal and vertical axis:

- Horizontal Axis: left, right, start and end sides
- Vertical Axis: top, bottom sides and text baseline

The general concept is to constrain a given side of a widget to another side of any other widget.

For example, in order to position button B to the right of button A

3.1 Constraint Layout Chains

A chain is a group of views that are linked to each other with bi-directional position constraints. For example, figure shows two views that both have a constraint to each other, thus creating a horizontal chain. Once chains are set, we can distribute the views horizontally or vertically with the following styles.

1. Spread: The views are evenly distributed.

For Example:

App:layout_constraintHorizontal_chainStyle=”spread”

App:layout_constraintVertical_chainStyle=”spread”

2. Spread inside: The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed.

For example: app:layout_constraintHorizontal_chainStyle=”spread_inside”

app:layout_constraintVertical_chainStyle=”spread_inside”

3.Packed: The views are packed together (after margins are accounted for). You can then adjust the whole chain’s bias (left/right or up/down) by changing the chain’s head view bias.

For example: app:layout_constraintHorizontal_chainStyle=”packed”

app:layout_constraintVertical_chainStyle=”packed”

4.Weighted: When the chain is set to either spread or spread inside, you can fill the remaining space by setting one or more views to “match constraints” (0dp). By default, the space is evenly distributed between each view that's set to "match constraints," but you can assign a weight of importance to each view using the layout_constraint Horizontal_weight and layout_constraintVertical_weight attributes. If you're familiar with layout_weight in a linear layout, this works the same way. So the view with the highest weight value gets the most amount of space; views that have the same weight get the same amount of space.

4. Bottom Navigation Bar

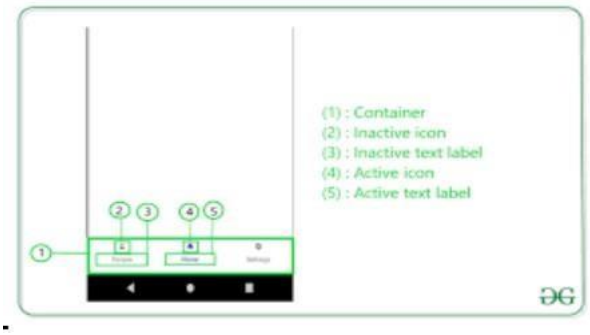


Fig 2.2 Bottom Navigation Bar

Bottom navigation bar allows the user to switch to different activities/fragments easily. It makes the user aware of the different screens available in the app. The user is able to check which screen are they on at the moment.

For example:

4.1 XML file:

```
<com.google.android.material.bottomnavigation.BottomNavigationView
android:id="@+id/bottom_navigatin_view"
android:layout_width="match_parent" android:layout_height="wrap_content"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent" />
```

4.2 Add Menu Items:

The next step is to add menu items for the bottom navigation view. Right-click on the res directory, select New > Android Resource File and select menu, provide a name and click OK.

For e.g.,

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:android="http://schemas.android.com/apk/res/android">
<item android: title="Home" />
```

<item android: title="Profile" />

<item android: title="Setting" />

</menu>

4.3 Passing data in fragments

To pass the data from one fragment to another in a better way and less code space i.e. done by using Bundles in Android. Android Bundles are generally used for passing data from one activity or fragment to another. Basically here concept of key-value pair is used where the data that one wants to pass is the value of the map, which can be later retrieved by using the key. Bundles are used with intent and values are sent and retrieved in the same fashion, as it is done in the case of Intent. It depends on the user what type of values the user wants to pass, but bundles can hold all types of values (int, String, Boolean, char) and pass them to the new activity or fragment.

For example:

// creating the instance of the bundle val bundle

= Bundle()

// storing the string value in the bundle // which is

mapped to key bundle.putString("key1", "Gfg :-

Main Activity")

// creating a intent

intent = Intent(this@MainActivity, Second Activity::class.java)

// passing a bundle to the intent intent.putExtras(bundle)

// starting the activity by passing the intent to it. start Activity(intent).

4.4 List Fragment

List Fragment hosts a ListView object that can be bound to different data sources, typically either an array or a Cursor holding query results. Binding, screen layout, and row layout are discussed in the following sections. Screen Layout List Fragment has a default layout that consists of a single list view.

However, if you desire, you can customize the fragment layout by returning your own view hierarchy from onCreateView(LayoutInflater, ViewGroup, Bundle). To do this, your view

hierarchy must contain a ListView object with the id "@android:id/list" (or R.id.list if it's in code)Optionally, your view hierarchy can contain another view object of any type to display when the list view is empty. This "empty list" notifies must have an id "android: empty".

Note that when an empty view is present, the list view will be hidden when there is no data to display.

For example:

<ListView android:id="@id/android: list"
android:layout_width="match_parent"
android:layout_height="match_parent" android:
background="#00FF00"
android:layout_weight="1"
android:drawSelectorOnTop="false"/>

5.Binding to Data

You bind the List Fragment's ListView object to data using a class that implements the List Adapter interface. Android provides two standard list adapters: Simple Adapter for static data (Maps), and SimpleCursorAdapter for Cursor query results.

6.MESSAGES

6.1 Toast

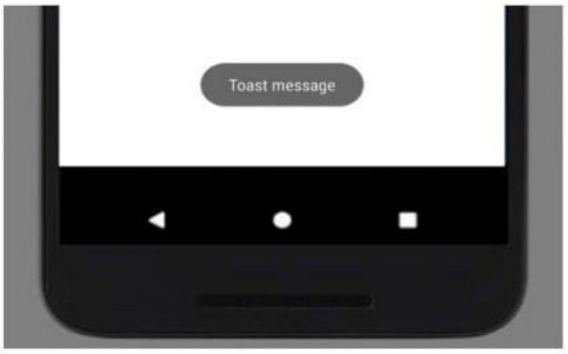


Fig 6.1 Toast

A Toast is a feedback message. It takes a very little space for displaying while overall activity is interactive and visible to the user. It disappears after a few seconds. It disappears automatically. If user wants permanent visible message, Notification can be used.

1. The application Contest returns the instance of Context class.
2. The message is of String type as ("this is toast message").
3. The Toast.LENGTH_SHORT and Toast.LENGTH_LONG are the constant defines the time duration for message display.
4. The show() method of Toast class used to display the toast message.
5. The set Gravity() method of Toast used to customize the position of toast message.

For e.g.:-

Toast.makeText(this,"this is message",Toast.LENGTH_SHORT).show() **6.2**

Dialog

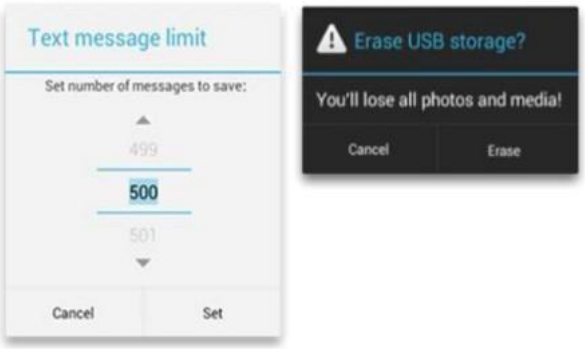


Fig 6.2 Dialog

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed. Dialog Design For information about how to design your dialogs, including recommendations for language, read the Dialogs design guide. The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:

6.2.1 Alert Dialog : A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

6.2.2 DatePickerDialog or TimePickerDialog: A dialog with a pre-defined UI that allows the user to select a date or time.

For e.g.

```
val builder = AlertDialog.Builder(it) builder.setMessage(R.string.dialog_start_game)
.setPositiveButton(R.string.start,
DialogInterface.OnClickListener { dialog, id ->
// START THE GAME!
})
.setNegativeButton(R.string.cancel,
DialogInterface.OnClickListener { dialog, id ->
// User cancelled the dialog
})
// Create the Alert Dialog object and return it builder.create()
```

7.VIEW BINDING

View binding is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a binding class for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

7.1 Setup instructions

View binding is enabled on a module-by-module basis. To enable view binding in a module, set the viewBinding build option to true in the module-level build.gradle file, as shown in the following example:

```
android {
...
buildFeatures            { viewBinding = true
}
}
```

If you want a layout file to be ignored while generating binding classes, add the tools:viewBindingIgnore="true" attribute to the root view of that layout file:

```
<LinearLayout
...
tools:viewBindingIgnore="true" >
...
</LinearLayout>
```

7.2 View binding in activities

To set up an instance of the binding class for use with an activity, perform the following steps in the activity's onCreate() method:

Call the static inflate() method included in the generated binding class. This creates an instance of the binding class for the activity to use.Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax.

Pass the root view to setContentView() to make it the active view on the screen.

private lateinit var binding: ResultProfileBinding override fun

onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState)

binding = ResultProfileBinding.inflate(layoutInflater)

val view = binding.root setContentView(view)
}

7.3 Use View binding in fragments

To set up an instance of the binding class for use with a fragment, perform the following steps in the fragment's onCreateView() method:

Call the static inflate() method included in the generated binding class. This creates an instance of the binding class for the fragment to use. Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax.

Return the root view from the onCreateView() method to make it the active view on the screen.

private var _binding: ResultProfileBinding? = null // This property is only valid between onCreateView and // onDestroyView. private val binding get() = _binding!!

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?, savedInstanceState:

Bundle?

): View? {

_binding = ResultProfileBinding.inflate(inflater, container, false) val

view = binding.root return view }

override fun onDestroyView() { super.onDestroyView()

_binding = null }

Differences from findViewById

View binding has important advantages over using findViewById:

- **Null safety:** Since view binding creates direct references to views, there's no risk of a null pointer exception due to an invalid view ID. Additionally, when a view is only present in some configurations of a layout, the field containing its reference in the binding class is marked with @Nullable.
- **Type safety:** The fields in each binding class have types matching the views they reference in the XML file. This means that there's no risk of a class cast exception.

NAVGRAPHS &

FRAGMENTS

8.Fragments

- Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity. ● Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
- Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.
- The Fragment Manager class is responsible to make interaction between fragment objects.

8.1 Android Fragment Lifecycle

The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.

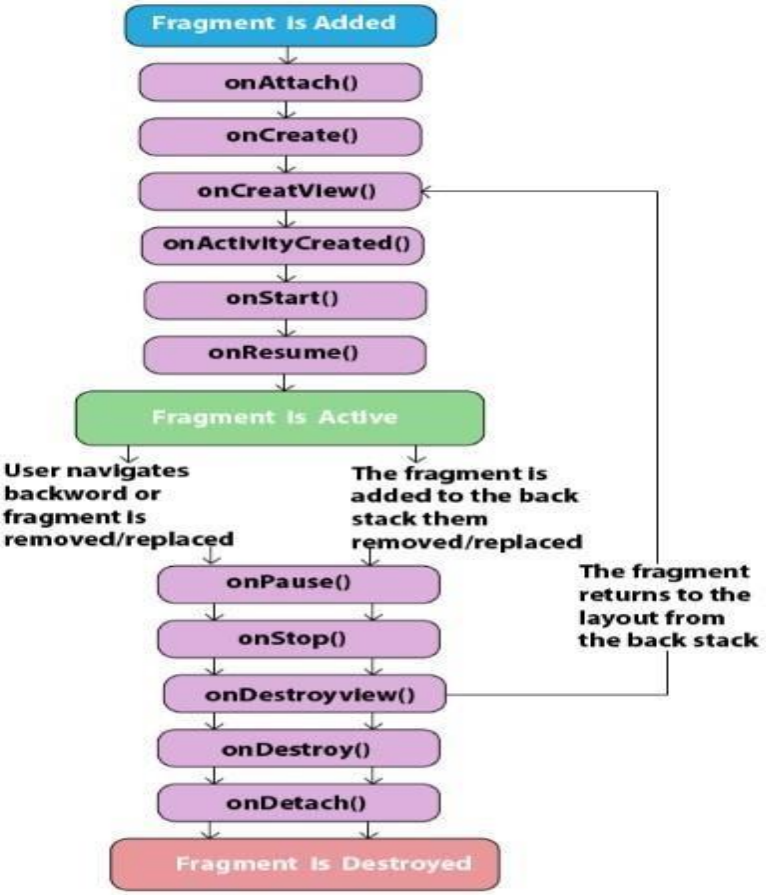


Fig 8.1 Activity Fragment Lifecycle

CHAPTER – III

SRS Documentation for Brain Burst App

The Budget Buddy application outlines the functional and non-functional requirements for the application, ensuring it meets the needs of its users.

I. Scope

Budget Buddy is a budgeting application designed to track income, expenses, and savings. It includes features for wishlist management, transaction history, budget management, and financial calculations.

- Operating Environment
- Platform : Android
- Devices : Smartphone and tablets

II. Functional Requirements

I. Budget Tracking Information

- The system shall display an overview of the user's budget status.
- The system shall allow users to set and modify budget limits.

II. Income and Expense Tracking

- The system shall enable users to log income and expenses.
- The system shall categorize transactions and allow for editing.

III. Wishlist Management

- The system shall allow users to add whislist where they enter there wish item like SavingMoney for Something (Buy a new Phone, Trip to Somewhere) etc.
- The system provide user full view of his/her whishlist analysis like there pending amount , history of transaction about its saving , etc.

- The user can easily edit, update or delete there items in whishlist.

IV. Monthly Savings Analysis

- The system shall provide reports on monthly savings.
- The system shall allow users to view trends and summaries.

V. Transaction History

- The system shall display a list of all transactions.
- The system shall allow users to search and filter transactions.

VI. Transaction History by Dates

- The system shall allow users to filter transactions by date range.
- The system shall display transaction details for selected dates.

VII. Notes Section

- The system shall allow users to create, view, and edit notes.
- The system shall allow users to categorize notes.

VIII. Budget Management

- The system shall allow users to create and manage multiple budgets.
- The system shall provide visual indicators of budget status.

IX. Card Holder

- The system shall store details of credit and debit cards securely.
- The system shall allow users to manage and update card information.

III. Non-Functional Requirements

I. Performance Requirements

- The application shall respond to user inputs within 2 seconds.
- The application shall handle up to 100,000 transactions efficiently.

II. Security Requirements

- The application is totally offline so no worries of data loss, or any sensitive information leakage.
- The application shall require authentication for accessing personal information.

III. Usability Requirements

- The application shall have an intuitive user interface.
- The application shall provide help and support features.

IV. Compatibility Requirements

- The application shall be compatible with Android versions 7 and above.
- The application shall support different screen sizes and resolutions.

IV. External Interfaces Glossary

- **Budget Tracking:** Monitoring of financial limits and expenditures**.**
- **Wishlist**: A list of desired items or goals.

CHAPTER - IV

FEASIBILITY STUDY

A feasibility study assesses the practicality of a project and helps in determining if it's viable. For Budget Buddy, the feasibility study covers operational, technical, economic, market analysis, financial, and legal aspects.

4.1 Operational Feasibility

Operational feasibility determines how well the solution works in the real world and how it meets the needs of the end users.

- **Usability:** The app should have an intuitive UI/UX design that allows users of all skill levels to easily manage their budgets and finances**.**
- **Maintenance:** Ongoing updates and maintenance will be necessary to improve functionality, fix bugs, and adapt to changes in the financial environment (such as new tax laws or updated GST rates).
- **Support:** Customer support should be available to help users with technical or financial issues that may arise.
- **Scalability:** The app needs to be scalable to accommodate a growing user base without sacrificing performance.

4.2 Technical Feasibility

Technical feasibility assesses whether the technology required to build Budget Buddy is available and achievable.

- **Hardware Requirements:** The app should function efficiently on smartphones and tablets, especially Android devices running version 7 and above.
- **Data Storage:** Room Database will handle user data locally, and cloud storage could be introduced for backups or cross-device synchronization in the future.
- **Security**: This is totally offline so it ensure data protection for sensitive financial information.

4.3 ECONOMICAL FEASIBILITY

Establishing the cost effectiveness of the proposed system i.e if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities .Thus the benefits of project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide.

4.4 Market Analysis Feasibility

This aspect assesses the potential demand for the product, competitor landscape, and how the app can meet market needs.

- **Target Audience:** The primary audience includes individuals looking for a simple yet effective way to manage their finances. This could be students, working professionals, and families.
- **Market Demand:** The rise of digital finance management tools, increased use of smartphones, and growing financial literacy create strong demand for a budget tracking app.
- **Marketing Strategy**: Marketing efforts could focus on financial independence, savings tips, and simplifying tax calculations for Indian users.

4.5 FINANCIAL FEASIBILITY

- Estimate the development costs, including software development, design, testing, and deployment.
- Evaluate potential revenue streams, such as in-app purchases, advertisements, or premium subscriptions.
- Calculate the expected return on investment (ROI) and payback period.
- Consider ongoing maintenance and operational costs.

4.6 **LEGAL AND REGULAROTY FEASIBILITY:**

- Identify legal and regulatory requirements related to data privacy, intellectual property rights, and user agreements.
- Ensure compliance with relevant laws and regulations in the target market.
- Assess potential risks related to legal issues and develop strategies to mitigate them.

4.7 **SCHEDULE AND TIME:**

- Develop a project timeline outlining key milestones and deliverables.
- Estimate the time required for each phase of development, including planning, design, development, testing, and deployment.
- Identify potential risks and dependencies that could impact the project schedule.

CHAPTER – V

REQUIREMENTS

In “Budget Buddy” there are basically two requirements Hardware Requirement & Software Requirement.

1. Hardware Requirements

- **Device:** Android smartphones or tablets
- **Processor:** Minimum 1.6 GHz Quad-Core or better
- **RAM:** Minimum 1 GB (2 GB or higher recommended)
- **Storage:** At least 100 MB of free storage for the app and data
- **Screen Size**: Supports screen sizes from 4 inches to 10 inches or larger

2. Software Requirements

- **Operating System:** Android 7(API level 29) or higher
- **Development Platform:** Android Studio
- **Programming Language**: Kotlin
- **Database:** Room (local database for offline data storage)

CHAPTER – VI

Data Flow Diagram

6.1 DFD 0-Level:-

In 0-Level DFD there are two panel user and admin. In user panel user can play quiz in the Brain Burst App and user will See Score. And in the admin panel admin can add questions and also admin will update that questions. Also all the information of the user will be shown to the admin.

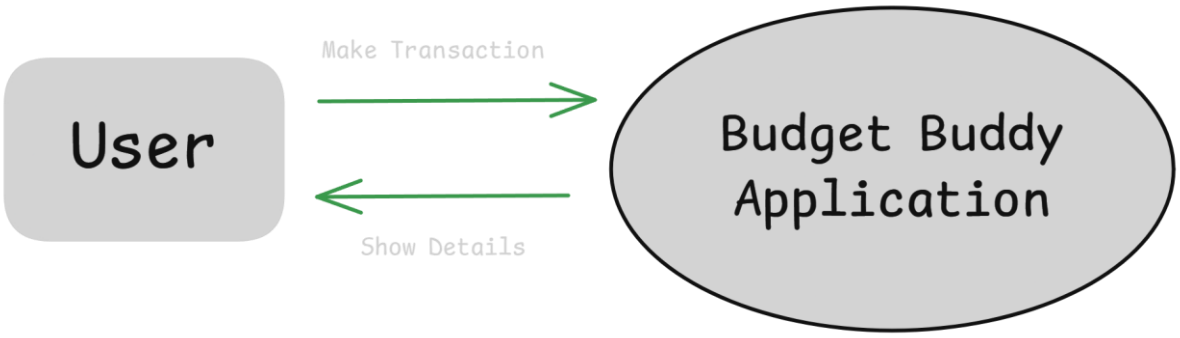


Fig 6.1 (0-Level DFD)

6.2 DFD 1-level :-

In 1-Level DFD User enter the app “ Budget Buddy ” and they will provided the feasibility like dash Board, transaction, Targets, and More Options .

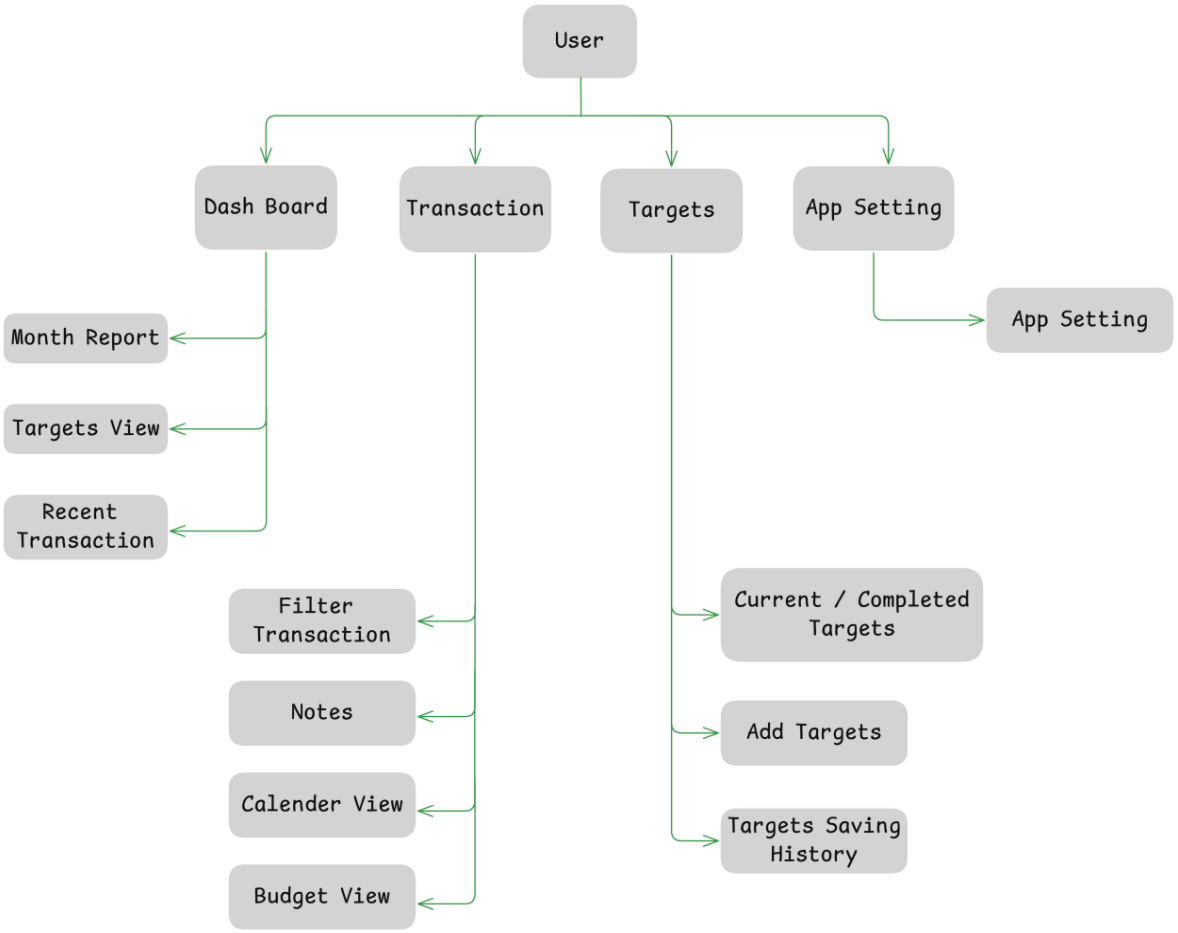


Fig 6.2 (1-Level DFD for User)

CHAPTER – VII

Screenshots

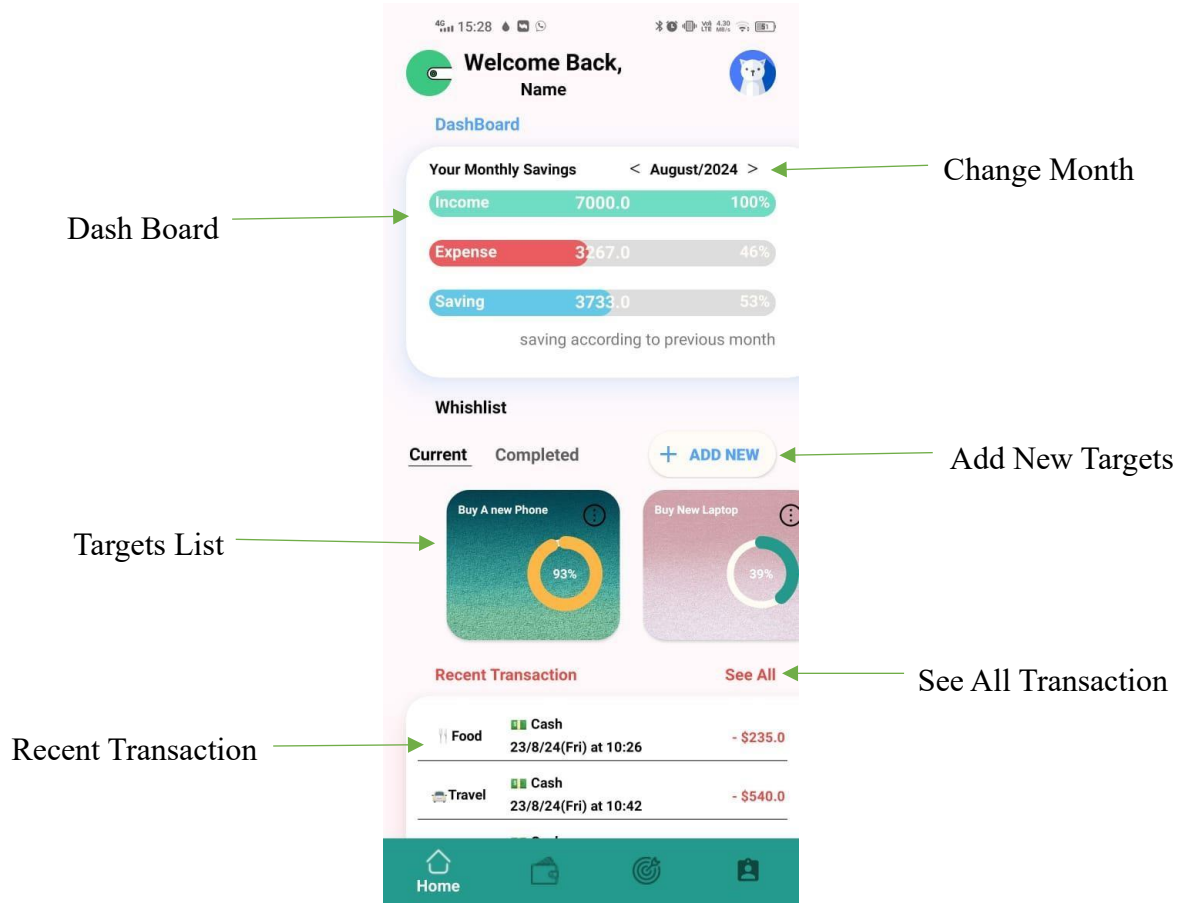


Fig 7.1 (Home Screen)

1. Home Screen

I. Dash Board

- This section shows Your monthly report .
- It include your income, expense, and saving.
- It represent how much you save / spend in this month.

II. Change Month

- You can change Month and see the report of previous months.

III. Target List

- You can view target that you created.
- You can edit/ update/ delete the target however you want.

IV. Add New Target

- You can add New Target Pressing the Add New Button.

V. Recent Transaction

- You can see all the recent Transaction that you made.

Fig 7.2 (Transaction screen)

2. Transaction Screen

I. Date

- You can change the date by incrementing or decrementing them just by simply pressing the next or back arrow key.
- You can also manually enter date by press the Month name.

II. Modes

- In Modes section You provided different modes such as daily, calendar, Monthly, Total, and Notes.
- In Daily mode the transaction shows in Daily format.
- In Calendar mode the transaction shows in Calendar view.
- In Monthly mode the transaction shows in Monthly view.
- In Total mode you can set budget and detail monthly report is provided.
- In Notes mode you can add notes related to some task or transaction.

III. Filter

- In filter you can choose the either only income or expenses window only or you can chose mix of both.

IV. Transaction

- In Transaction window all the transaction is show that you made through out.

V. Add Transaction

- You can transaction just by clicking on plus '+' button
- You can select either you want to add income or expense.

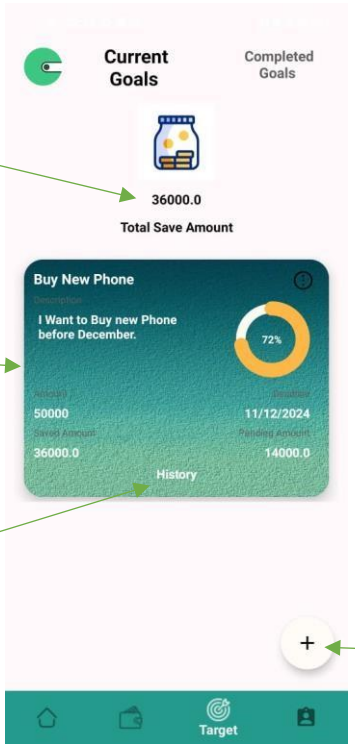


Fig 7.3.1 (Target screen)

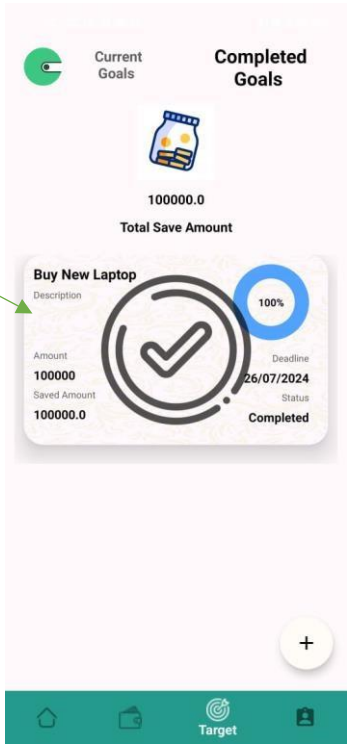
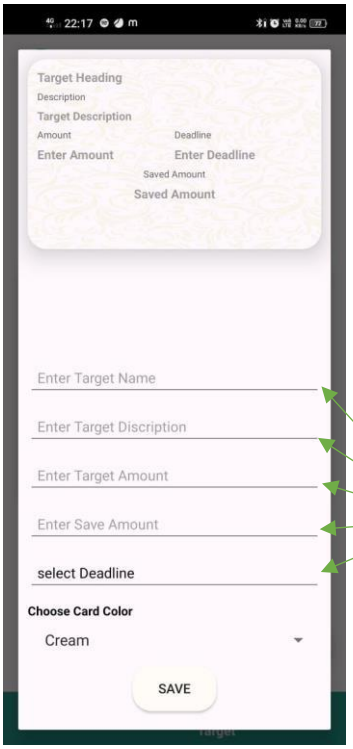


Fig 7.3.2 (Target screen)



7.3.3 (Target screen)

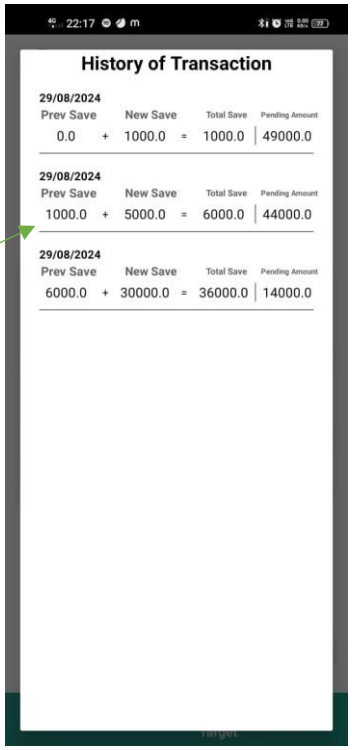


Fig 7.3.4 (Target screen)

3. Target Screen

I. Saving Amount

- In Saving amount it reflect the money that you saved for targets.
- Current and completed saving money both shows differently.

II. Current Targets

- In Current Target Window the ongoing targets is shown

III. History

- By clicking on History button you can see the Saving history that you made showing in fig (7.3.4).

IV. Add Target

- By clicking on plus button '+' you can add targets.
- By clicking plus button the new window pop up (fig (7.3.3)) to enter the details of the target.

V. Completed Targets

- In Completed Target Window the completed targets is shown .

CHAPTER – VIII Future Scope & Conclusion

Future Scope

The Budget Buddy app holds immense potential for future enhancements to improve user experience and functionality. Some key areas for expansion include:

1. AI-Driven Financial Insights: Incorporating AI algorithms to provide personalized financial recommendations based on spending habits, income trends, and goals.
2. Investment Tracking: Expanding the app to support tracking investments like stocks, mutual funds, and cryptocurrencies, allowing users to manage all aspects of their finances in one place.
3. Multi-Currency Support: Enabling multi-currency support for international users, allowing them to track expenses and incomes across different currencies with real-time conversion rates.
4. Subscription Management: Adding features for tracking and managing subscriptions, helping users monitor recurring expenses and identify unnecessary expenditures.
5. Bill Payment Reminders: Introducing automatic reminders for bill payments and upcoming dues to avoid late payments and improve financial discipline.
6. Advanced Security: Enhancing data security with features like biometric authentication (fingerprint/face recognition) and end-to-end encryption for financial data protection.
7. Integration with Financial Institutions: Connecting the app to banks and payment platforms through APIs for automatic transaction imports, reducing manual entry and increasing accuracy.
8. Collaborative Budgeting: Adding a feature for shared budgets, allowing multiple users (e.g., family members or roommates) to collaboratively manage finances.
9. Gamification: Integrating gamified elements to encourage savings, such as challenges, badges, or rewards for reaching financial milestones.

Conclusion

Budget Buddy successfully addresses the need for a comprehensive financial management tool, providing users with a platform to track and manage their budgets, expenses, and incomes. The app's range of features, including budget management, transaction history, wishlist management, and tax calculators, empower users to gain control over their finances and make informed decisions.

The project has shown promising results during development and has potential for continued growth with the incorporation of advanced technologies like AI, enhanced security, and broader financial tracking capabilities. With future enhancements, Budget Buddy can evolve into a highly intelligent, personalized, and secure financial assistant, aiding users worldwide in achieving their financial goals.

CHAPTER – IX

BIBLIOGRAPHY

- https://www.google.com
- https://in.pinterest.com
- https://github.com