WISEWALLET
A NATIVE ANDROID APPLICATION

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

BY

Andaluri Balaji	**Gondu Shanmukha SrinivasaRao**
21331A0501	**21331A0554**
Borra Venkata Durga Rao	**Gollu Sankar Narayana**
21331A0525	**21331A0553**

Under the Supervision of
Mr. R. Ravikanth
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAHARAJ VIJAYARAM GAJAPATHI RAJ COLLEGE OF ENGINEERING
(Autonomous)
(Approved by AICTE, New Delhi, and permanently affiliated to JNTUGV, Vizianagaram), Listed u/s 2(f) & 12(B) of UGC Act 1956.
Vijayaram Nagar Campus, Chintalavalasa,Vizianagaram-535005, Andhra Pradesh
APRIL, 2025

CERTIFICATE

This is to certify that the project report entitled "**WISEWALLET A NATIVE ANDROID APPLICATION**" being submitted by A.Balaji (21331A0501), G.Shanmukha Srinivasa Rao(21331A0554), B.Venkata Durga Rao (21331A0525), G.Sankar Narayana(21331A0553) in partial fulfillment for the award of the degree of "**Bachelor of Technology" in Computer Science and Engineering** is a record of bonafide work done by them under my supervision during the academic year 2021-2022.

Dr. P. Rama Santosh Naidu
Senior Assistant professor,
Supervisor,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

Dr. T. Pavan Kumar
Associate Professor,
Head of the Department,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

External Examiner

DECLARATION

We hereby declare that the work done on the dissertation entitled “**WISEWALLET A NATIVE ANDROID APPLICATION**” has been carried out by us and submitted in partial fulfilment for the award of credits in Bachelor of Technology in Computer Science and Engineering of MVGR College of Engineering (Autonomous) and affiliated to JNTUGV, Vizianagaram. The various contents incorporated in the dissertation have not been submitted for the award of any degree of any other institution or university.

ACKNOWLEDGEMENTS

We express our sincere gratitude to **Mr. R. Ravikanth** for his invaluable guidance and support as our mentor throughout the project. His unwavering commitment to excellence and constructive feedback motivated us to achieve our project goals. We are greatly indebted to him for his exceptional guidance.

Additionally, we extend our thanks to **Prof. P.S. Sitharama Raju (Director)**, **Prof. Ramakrishnan Ramesh (Principal)**, and **Dr. T. Pavan Kumar (Head of the Department)** for their unwavering support and assistance, which were instrumental in the successful completion of the project. We also acknowledge the dedicated assistance provided by all the staff members in the Department of Computer Science & Engineering. Finally, we appreciate the contributions of all those who directly or indirectly contributed to the successful execution of this endeavour.

A.Balaji (21331A0501)
G.Shanmukha SrinivasaRao (21331A0554)
B.Venkata Durga Rao (22335A0525)
G.Sankar Narayana (21331A0553)

LAST MILE EXPERIENCE (LME)

PROJECT TITLE

WISE WALLET

BATCH NUMBER – 7A

BATCH SIZE – 4

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Name:
A.Balaji
Email:
balajiandaluri@gmail.com
Contact Number:
9392407701

Name:
G.Shanmukha Srinivasa Rao
Email:gondushanmuk123@gmail.com
Contact Number:
8978104025

Name:
B.Venkata Durga Rao
Email:venkatdurgarao7450@gmail.com
Contact Number:
7207162449

Name:
G.Sankar Narayana
Email:gollusankarnarayana@gmail.com
Contact Number:
9398359515

Project Supervisor

Name:
R. Ravikanth
Designation:
Associate Professor
Email:
Ravi.kranthi787@gmail.com
Contact Number:
8985206284

Project Objectives

1. **PO1**
   To develop a Native Android application for user-friendly spending analysis, Display monthly spending and savings through intuitive visualizations.
2. **PO2**
   To implement automated parsing of bank messages for transaction data. Extract store and display transaction amounts and dates.

Project Outcomes

1. **Overall Project Outcome**

Successfully extracts and provides a structured storage of relevant bank transaction data from SMS messages. Functional input and storage of user-defined income and expenses. Transaction visualization based on user-selected month. Generates charts to illustrate spending patterns. Integrated user guide for application assistance. User authentication via phone number and OTP.

Domain of Specialisation

Native Android Application Development for Personal Finance Management

How your solution helping the domains?

Our solution helps the personal finance management domain by automating the capture of transaction data directly from bank SMS messages, providing users with a comfortable view of their spending. This automation reduces manual data entry, improving accuracy and efficiency, and empowers users with real-time insights into their financial activities.

List the Program Outcomes (POs) that are being met by doing the project work

PO9: Individual and teamwork	Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
PO10: Communication	Communicate effectively on complex engineering activities with the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11: Project management and finance	Demonstrate knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work as a member and leader in a team, to manage projects and in multi-disciplinary environments.
PO12: Life-long learning	Recognize the need for, and have the preparation and ability to engage in, independent and life-long learning

End Users of Your Solution

Our primary stakeholders include salaried employees and metropolitan residents. Secondary stakeholders, encompassing all salaried individuals, prioritize user-friendly interfaces, clear financial insights.

ABSTRACT

Monetary mindfulness is essential for saving money and preventing individuals from overspending, which can lead to dangerous debts. While there are some expense tracking apps already available, they are often designed for businesses as budgeting tools. The main limitations of these apps for individual use are their complex interfaces and the cumbersome procedures required to set initial settings. Additionally, premium subscriptions are often needed for advanced features like spending pattern predictions and analysis. Therefore, there is a need for a simple, user-friendly expense tracker application that can be as easily used as possible.

This proposal focuses on designing an expense tracker application that is less complex and more user-friendly. To achieve this, we are using Native Android development with Kotlin, a programming language that is highly supported for Android app development.

CONTENTS

List of Abbreviations

AOSP	_	Android Open-Source Project
FOSS	–	Free and Open-Source Software
ER	–	Entity Relationship
BaaS	–	Backend as a Service
NoSQL	–	Not Only SQL
SQL	–	Structured Query Language
JSON	–	JavaScript Object Notation
HTTPS	–	HyperText Transfer Protocol Secure
TLS	–	Transport Layer Security
SSL	–	Secure Socket Layer
XML	–	Extensible Markup Language
UI	–	User Interface
UX	–	User Experience

List of Figures

CHAPTER 1
INTRODUCTION

In the rapidly evolving financial landscape, efficiently managing personal finances has become more critical than ever. With the proliferation of digital transactions, users face the challenge of tracking and categorizing numerous financial activities. This challenge is significantly amplified for individuals with lower salaries navigating the high cost of living in metropolitan cities. The constant barrage of expenses, from inflated rents and transportation costs to the daily pressures of urban consumerism, can quickly overwhelm those with limited financial resources.

To address the growing challenges of personal finance management, particularly for individuals facing the complexities of urban living and digital transactions, this project leverages the power of **native Android development** using Kotlin. WiseWallet directly confronts the issue of fragmented financial tracking by implementing robust **message parsing** techniques. By analyzing and extracting transaction data from bank-related SMS messages, the application automates the process of recording online expenses, reducing the need for manual entry. This data is then securely stored in a database, ensuring reliable retrieval and enabling users to access a comprehensive and organized view of their spending patterns whenever needed. Employing Kotlin’s efficiency and modern Android development practices, WiseWallet provides a seamless and user-friendly solution for managing finances in today’s fast-paced digital environment.

1.1 Identification of seriousness of the problem

Financial discipline is crucial for many individuals, especially current working employees with insufficient salaries in metropolitan cities. Daily needs and other necessities often hinder their ability to maintain savings. The main objective of this project is to remind users at regular intervals to reduce their overspending by providing a document. This document includes a color-coded chart that represents the frequency of categorized transactions and generates a default monthly report.

1.2 Problem definition

Although there are some existing apps regarding expense and budget tracking, user reviews and ratings for apps in Google PlayStore mentioned about the complexity and procedure of the interfaces as a difficulty. This motivated us to design an application that is user-friendly and less complex with optimal usage.

1.3 Objective

The need for a user-friendly expense tracker arises from the complexity and premium requirements of existing applications, which are often geared toward business use. By developing an application that is simple and accessible, individuals can gain better control over their finances, promote saving, and avoid falling into debt.

Our Approach:

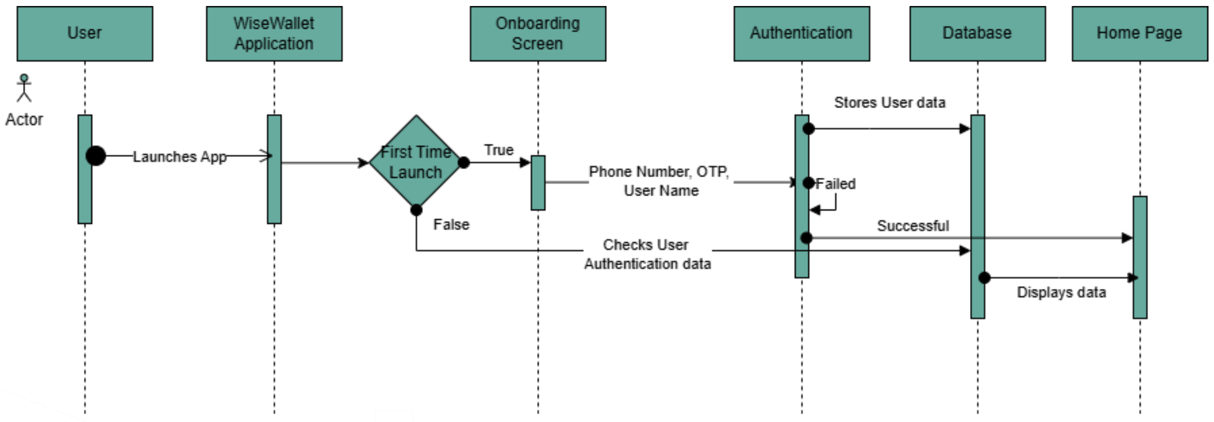


Fig 1.1 : Sequence Diagram for Phone Number Authentication

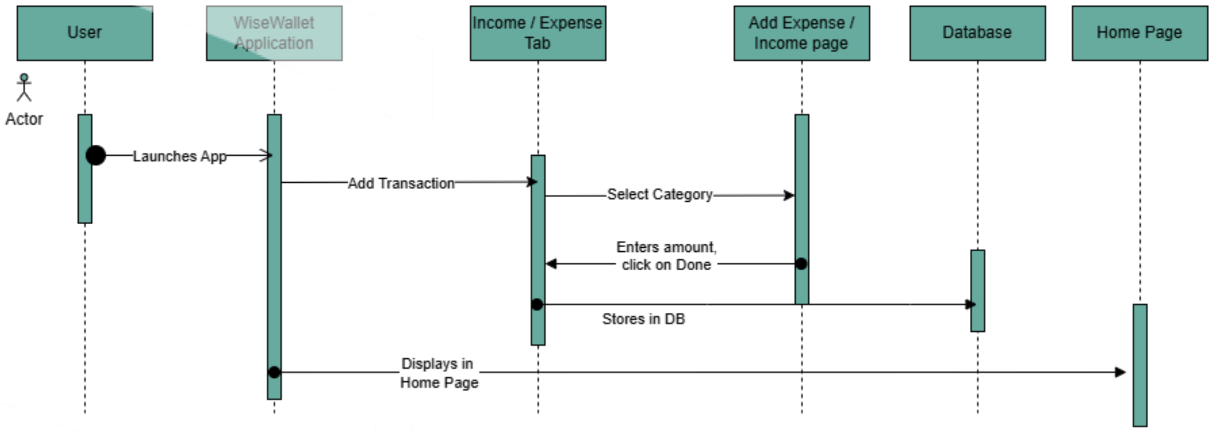


Fig 1.2 : Sequence Diagram for Manual Entry of a Transaction

1.4 Existing models

There are many existing apps like “Expensify”, “AndroMoney”, “EveryDollar”, “Rocket Money” etc., Upon analyzing all the user reviews that are mentioned in play store, 3 out of 5 are regarding business budget tracking. Some mentioned about the complexity and procedure of the interfaces as a difficulty...

CHAPTER 2

LITERATURE SURVEY

With the help of Google, we surveyed all related work through several official documentations that are available over internet, and customer reviews from the description of applications in Google play store. The following image displays the user interaction with an existing app, where the user has to enter details regarding the transaction.

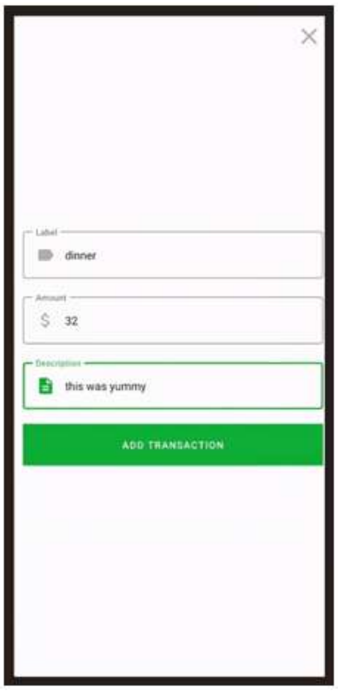


Fig 2.1 : Image showing difficulty in adding transactions manually.

Here, if we observe, **[1]** users may find it difficult and inconvenient to manually enter every transaction they make, especially in their busy daily lives. One of the customers reviews says that, “I wanted to try for individual use, but it’s not at all suitable. Its Interface is not interactive at all.” **[2]** Jade Howkins August 5, 2021- A review from Google Play Store for a particular Expense tracking app. Another interesting review from a customer is that, “This APP is free... BUT THE SERVICE IS NOT! This should be explained in the description FIRST!” **[3]** which says, premium subscribed users are only given expected services like analysis, automation and predictions. By digging into the strengths and weaknesses of current offerings, we can identify opportunities to create a more comprehensive and accessible app. Key areas of exploration include the cost effectiveness of existing solutions, their availability across platforms, the accuracy of expense categorization, and the underlying technologies and methodologies employed. This analysis will provide valuable insights into the challenges and limitations of current expense tracking apps, paving the way for a more innovative and impactful solution.

CHAPTER 3
THEORETICAL BACKGROUND

3.1 Android Platform and Development Environment

3.1.1 What is Android?

Android is an operating system based on a modified version of the Linux kernel and other open-source software, designed primarily for touchscreen-based mobile devices such as smartphones and tablets. Android has historically been developed by a consortium of developers known as the Open Handset Alliance, but its most widely used version is primarily developed by Google. At its core, the operating system is known as the **Android Open Source Project (AOSP)** and is free and open-source software (FOSS) primarily licensed under the Apache License.

3.1.2 General Development Tools

- **Android Studio**

  Android Studio is by far the most important tool you need for Android Development. It was created by both Google and JetBrains and it is very similar to IntelliJ IDEA. Android Studio is a specific IDE used by Android developers. This can also be used to develop cross-platform applications using Flutter.

  Its key features:

  - Wireless Debugging
  - Compose UI previews
  - Seamless SDK synchronization
  - Gradle build system
  - Logcat for debugging
  - Device and performance profiling

- **Kotlin Programming Language**

  Android development first started using Java as their preferred language of development. However, since JetBrains announced their stable release of **Kotlin**, we have had multiple migrations from Java to Kotlin quickly. In May 2019, Google announced Kotlin to be **Android's preferred language of development**.

- **Firebase**

  Firebase Realtime Database offers a streamlined solution for managing application data without building a dedicated backend API server. This Google-hosted service provides a cloud-based, NoSQL database, enabling real-time data synchronization across clients, simplifying data storage and retrieval for Android applications.

- **Git & GitHub**

  Git is a distributed version control system that tracks changes in computer files and coordinates work on those files among multiple people. GitHub is a web-based hosting service for Git repositories. It offers all the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

  Core Operations:

  1. Git Setup and Initialization

     ```
     $ git init
     $ git clone <repository_url>
     $ git config user.name "BalajiAndaluri"
     $ git config user.email balajiandaluri@gmail.com
     ```

  2. Staging and Committing

     ```
     $ git status
     $ git add <file>
     $ git add .
     $ git commit -m "Commit message"
     ```

  3. Branching and Merging

     ```
     $ git branch <branch_name>
     $ git checkout <branch_name>
     $ git merge <branch_name>
     ```

  4. Remote Repositories

     ```
     $ git remote add origin <repository_url>
     $ git remote -v
     $ git push origin <branch_name>
     $ git pull origin <branch_name>
     ```

  5. Undoing Changes

     ```
     $ git reset HEAD <file>
     $ git revert <commit_hash>
     ```

3.1.3 Application Framework

- **Activity Manager:**

  This service controls Activity, Service, and Broadcast Receiver lifecycles. It also Handles task management and application switching, Manages the application's process lifecycle. Coordinates inter-component communication and It Ensures proper resource allocation and deallocation.
- **Content Manager:**
  This layer mainly provides a standardized way to share data between applications. It Enables data access through URIs. Manages data storage and retrieval. Supports data security and permissions. Facilitates data sharing with system components.
- **Resource Manager:**
  Resource manager provides access to application resources (strings, layouts, drawables). It handles resource localization and configuration changes. Also optimizes resource loading and usage. Manages layout and UI elements. Provides access to non-code assets.
- **Notification Manager:**
  It handles the creation and display of notifications. Manages notification channels and priorities. Provides user interaction with notifications. Supports various notification styles (text, images, progress). Provides interaction with the user even when the app is in the background.

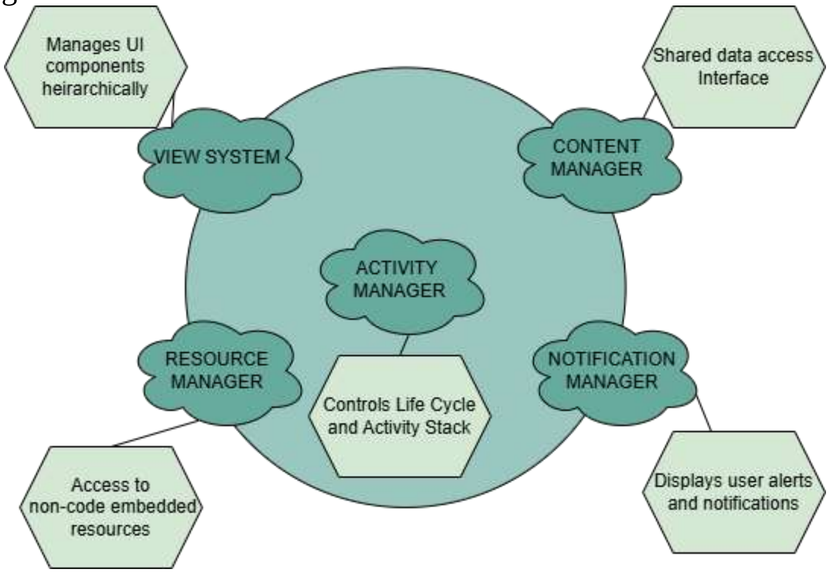

Manages UI components heirarchically
Shared data access Interface
VIEW SYSTEM
CONTENT MANAGER
ACTIVITY MANAGER
RESOURCE MANAGER
NOTIFICATION MANAGER
Controls Life Cycle and Activity Stack
Access to non-code embedded resources
Displays user alerts and notifications

Fig 3.1 : Application Framework showing key Services

3.2 User Interface Design and Event Handling

Android UI design emphasizes user-centered design. This involves understanding user needs, cognitive limitations, and interaction patterns. Android's event-driven architecture relies on event listeners to capture and respond to user actions. These models define how users interact with the application and how the application responds.

3.2.1 Layout Design and Structure

Linear Layout:

- Used for linear arrangements of UI elements, ideal for displaying transaction lists vertically.
- This layout's simplicity makes it efficient for displaying lists or forms where components follow a clear, one-dimensional order. Its predictable arrangement is beneficial for creating consistent and easily understood UI structures.

Constraint Layout:

- Employed for creating complex and responsive layouts, crucial for financial dashboards and detailed transaction views.
- Minimizes the need for nested layouts, improving performance and simplifying the creation of adaptive user interfaces.

Frame Layout:

- Frame Layout is the simplest layout type, designed to occupy an area on the screen and display a single child view. If multiple child views are added, they are stacked on top of each other.
- This layout is ideal for scenarios where you need to overlay views or display a single dynamic view. Its core principle is to provide a container for a single, primary view, with optional overlays.

Table Layout:

- Table Layout arranges child views in rows and columns, like a HTML table. It automatically sizes columns and rows to fit the content.
- This layout is useful for displaying data in a tabular format, where alignment and organization are crucial. Its theoretical foundation is in its structured, grid-like approach to UI element arrangement.

3.2.2 UI Component Attributes and Styling

Attributes are properties that define the appearance and behaviour of UI components (Views). They allow developers to customize the look and feel of their application's interface. Attributes are defined within XML layout files, providing a declarative way to specify UI element characteristics.

Some common attributes include – *layout_width, layout_height, text, textColor, textSize, background, padding and margin, id, visibility, and gravity*.

- Creating custom styles for transaction categories (e.g., groceries, utilities) to provide visual cues.
- Using drawable resources for icons representing transaction categories.

Styles are collections of attributes that can be applied to UI components. They promote code reusability and maintainability by encapsulating common UI properties. Styles are defined in XML files within the *res/values/styles.xml* directory.

Benefits of using styling:

Consistency: Styles ensure that UI elements have a consistent appearance throughout the application.

Reusability: Styles can be applied to multiple Views, reducing code duplication.

Maintainability: Changes to a style are reflected in all Views that use it, simplifying UI updates.

Theming: Styles can be used to create themes, allowing users to customize the application's appearance.

3.2.3 Responsive Design for Diverse Devices

Key Units:

- Using **Density-independent Pixels (dp/ dip)** and **Scale-independent Pixels** (sp) units for consistent UI element sizing across different screen densities.
- **dp** is Ideal for margins, padding, and the dimensions of views (buttons, images, etc.), while **sp** is exclusively for text size in layouts.
- **Pixels** (px) are absolute units, meaning they represent actual pixels on the screen. This can lead to inconsistent UI sizes across devices with different screen densities.

Adaptive Layouts and Resources:

- Provides alternative layout resources for different screen orientations.
- Uses resource qualifiers to load different drawable resources based on screen density.
- Employing Constraint Layout to create layouts that adapt to varying screen sizes.

3.3 Data Storage and Retrieval

Data storage and retrieval encompass the methods and technologies used to manage and access data. This involves organizing data using various models (relational, NoSQL), storing it in suitable systems (databases, file systems, cloud storage), and retrieving it efficiently through query processing and data access patterns. Data integrity and security are crucial, ensuring data accuracy and protection against unauthorized access. Here are different ways for storage and retrieval of data.

3.3.1 Structuring Data with SQLite/Room

- **Relational Database**

  The foundation of SQLite and Room lies in the relational database model, where data is organized into tables with rows and columns. This model relies on set theory and first-order predicate logic to ensure data integrity.

- **ER Modeling**

  ER diagrams provide a visual representation of entities (tables) and their relationships. For WiseWallet, entities like "Transactions" and "Categories" would be defined, with relationships like "Transactions belong to a Category.

- **SQL**

  SQL provides the theoretical language that allows us to interact with the relational database. It allows us to create, read, update, and delete data.

3.3.2 Firebase Realtime Database Connectivity (Cloud Storage)

- **Cloud Computing and Backend-as-a-Service (BaaS):**

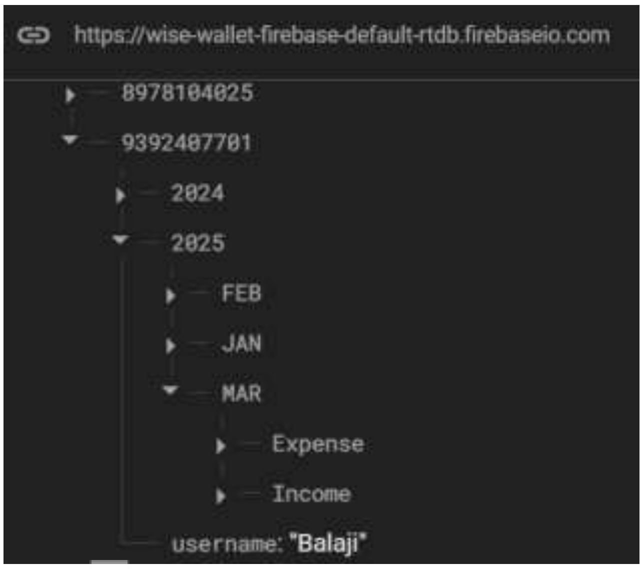
  Firebase represents a Backend-as-a-Service (BaaS) paradigm, abstracting server-side infrastructure and complexities. This aligns with cloud computing principles, where resources are provided on-demand over the internet.

- **NoSQL Data Model (JSON Tree):**

  Firebase Realtime Database uses a NoSQL data model, specifically a JSON tree structure. This deviates from the relational model, offering flexibility and scalability for real-time applications. The JSON tree structure allows for hierarchical data representation, making it suitable for storing dynamic and unstructured data.

- **Firebase Security Rules:**

  Firebase Security Rules provide a declarative language for defining access control and data validation rules. These rules are evaluated on the Firebase server, ensuring secure data access and preventing unauthorized modifications.



```
https://wise-wallet-firebase-default-rtdb.firebaseio.com
  ▸ 8978104025
  ▾ 9392407701
      ▸ 2024
      ▾ 2025
          ▸ FEB
          ▸ JAN
          ▾ MAR
              ▸ Expense
              ▸ Income
      username: "Balaji"
```

Fig 3.2 : Structure of NoSQL JSON node structure for a user

3.4 Network Communication and Message Parsing

3.4.1 Database Connectivity

- **Client-Server Architecture:**

  The fundamental theory here is the client-server architecture where, the application (the client) communicates with the Firebase Realtime Database (the server). This architecture allows for centralized data management and access, enabling real-time synchronization across multiple devices.

- **Real-Time Data Transfer:**

  Firebase utilizes protocols like WebSockets to establish persistent, bidirectional communication channels. This enables real-time data updates, where **changes on the server are instantly reflected on the client and vice versa**. This is the implementation of the observer pattern, where the client observes the changes in the database.

- **Data Serialization and Deserialization:**

  Data is serialized (converted into a transferable format, like JSON) before being sent over the network and deserialized (converted back into objects) upon reception. This ensures that data can be transmitted and interpreted correctly between different systems.

- **Theory of Secure Communication:**

  HTTPS builds upon HTTP by adding a layer of encryption using TLS/SSL. This encryption ensures that data transmitted between the client and server is protected from eavesdropping and tampering.

3.4.2 Text Processing

- **Regular Expressions (Regex)**

  The core theoretical concept behind message parsing is pattern recognition. Regular expressions provide a formal language for specifying search patterns within text.

  Kotlin's Regex class allows for the creation and application of regular expressions to extract specific data elements (dates, amounts, merchant names) from SMS messages.

  This is built on the theory of finite automata, which provides the ability to recognize regular languages.

  Kotlin's *Regex.find()* and *Regex.findAll()* functions are used to locate and extract data elements that match predefined regular expressions. *SimpleDateFormat* class is used to parse date strings into Date objects.

CHAPTER 4

APPROACH DESCRIPTION

4.1. Approach Flow

1. Requirements Gathering

   Requirements gathering and specification is done through surveying and analyzing existing documentation, end user reviews on existing apps, and using existing apps as an end user and recording those observations.

- Functional Requirements:

  Intuitive Interface- User friendly with less complex terminology, less user intervention, and easy understandability

  Manual Entry- for transactions that are done through hand cash, where we do not get any message

  Message app Tracking- for digital transactions, where scanning messages regarding keywords like merchants, terms like “beverages”, “textiles”, “shopping”, etc.,

  Categorization and storing databases to make use while generating reports based on stored previous data.

  Notify user, if exceeds the set limit on budget and suggest for saving plans according to user’s income.

- Non-Functional Requirements:

  Reliability

  Performance

  Efficiency

  Policies and Standards

  Interoperability

  Usability

2. Identify key screens

   a. Used figma for prototyping the visual screens and for styling

   b. Mainly identified to implement a bottom navigation view to display components.

   c. Components included:

   i. Home- Default screen used to display Expenses, Incomes and transactions.

   ii. Charts- To display categorical and month wise visual representation using pie- charts.
   iii. Add / Choose– To manually enter transactions for some categories like shopping, food etc.,
   iv. Profile– To keep track of user details and maintain settings.

3. Configuring Android Studio, GitBash and creating remote repository
   a. From developer.android.com, downloaded the latest Android studio and installed it in all our devices.
   b. Created a GitHub repository(remote) to work collaboratively.
   c. Installed Git Bash and set up Git to create local repo, to save and merge our work with remote repo, to commit and revoke the changes, etc.
4. Designing Layouts using XML
   a. For creating layouts and root view-groups with necessary constraints.
   b. To add UI components and set attributes for exact positioning on screens
   c. For using Styles, themes and resources for better visual and user experience.
5. Inner Fragments, Tabs, splash-screen, Date-picker Dialog
   a. A scrollable fragment with all transactions in card-view along with a search view.
   b. To add both expenses and incomes, we created Tab layout with two tabs, specifically for Expenses and Incomes
   c. A date-picker dialog is created to get data of user transactions on a particular date.
6. Integration using Kotlin.
   a. Created Activities for layouts (MainActivity.kt, SplashActivity.kt, etc.,) and included core logic of the application.
7. Database Connectivity with Firebase Real time database that stores user transaction data in a NoSQL JSON Node structure.
8. Agile methodology and incremental testing were used to ensure the proper functionality of each module during development.

CHAPTER 5
IMPLEMENTATION

5.1 Intial Implementation :

We started by figuring out the screens that are required for the app, like the Home screen to show an overview, a Charts page for graphs, a section to Add Manual Entries, a More page for managing accounts, and a Profile/Settings page. Each screen was designed to be simple and easy to use. We created these layouts in Android Studio using XML, keeping the design clean and user-friendly.

To make navigation easier, we added a bottom navigation bar so users can quickly switch between the main sections. We also included extra features like a splash screen to make the app look professional, and tabs to organize information better. Next, we integrated and made the application functioning using Kotlin. For example, clicking buttons would take users to the right screens, and they could enter data that the app would save and show later. We added icons for different expense categories, like food, travel, and shopping, to make manual entry straightforward and visually clear. Each part of the app was built and tested separately to make sure it worked well before putting everything together. This step set up the basic structure of the app, making it ready for more features and improvements later.

5.2 Detailed description of the methodologies and tools being used :

We used Agile methodology to develop the app, which means we worked on the project in small steps, made changes based on feedback, and improved it gradually. This approach allowed us to focus on each part of the app separately and make sure it worked well before moving to the next step. We also used modular programming, where each screen, like Home, Charts, or Manual Entry, was built and tested individually. Once everything was ready, we connected all the screens using Kotlin to create a fully functional app. For tools, we used Android Studio as the main platform to write code, design layouts, and test the app. Figma helped us create designs and plan the look of the app. To make sure the app followed the best practices, we referred to developer.android.com for guidelines and tutorials. We used Git for tracking changes in the code and GitHub for collaboration and storing the project. Lastly, Gradle helped manage libraries and dependencies to ensure the app ran smoothly. These tools and methodologies worked together to make the development process organized, efficient, and flexible.

5.3 Step-by-step process of the intial phase of implementing the project :

1. Identifying key screens

   • Used figma for prototyping the visual screens and for styling

   • Mainly identified to implement a bottom navigation view to display +components.

   • Components included:

      o Home- Default screen used to display Expenses, Incomes and transactions.

      o Charts- To display categorical and month wise visual representation using pie-charts.

      o Add / Choose – To manually enter transactions for some categories like shopping, food etc.,

      o Profile – To keep track of user details and maintain settings.

2. Configuring Android Studio, Git bash and creating remote repository

   • From developer.android.com, downloaded the latest Android studio and installed in all our devices

   • Created a GitHub repository(remote) to work collaboratively.

   • Installed Git Bash and set up Git to create local repo, to save and merge our work with remote repo, to commit and revoke the changes, etc.

3.Designing Layouts using XML

   • For creating layouts and root view-groups with necessary constraints.

   • To add UI components and set attributes for exact positioning on screens

   • For using Styles, themes and resources for better visual and user experience.

4. Inner Fragments, Tabs, splash-screen, Date-picker Dialog

   • A scrollable fragment with all transactions in card-view along with a search view.

   • To add both expenses and incomes, we created Tab layout with two tabs, specifically for Expenses and Incomes

   • A date-picker dialog is created to get data of user transactions on a particular date.

5. Integration using Kotlin.

   • Created Activities for layouts (MainActivity.kt, SplashActivity.kt, etc.,) and included core logic of the application.

6. Real-time, Scalable NoSQL Database

   • Made use of available open source firebase.google.com for real-time database. Store the data in a Node format with NoSQL.

CHAPTER 6

RESULTS

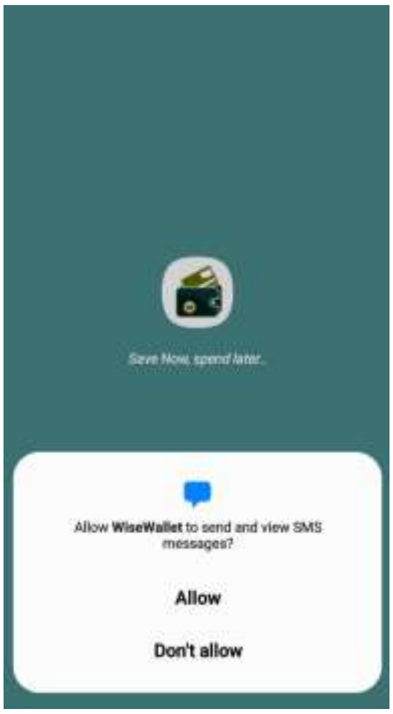


Fig 6.1 : Splash Screen and seeking permission from user

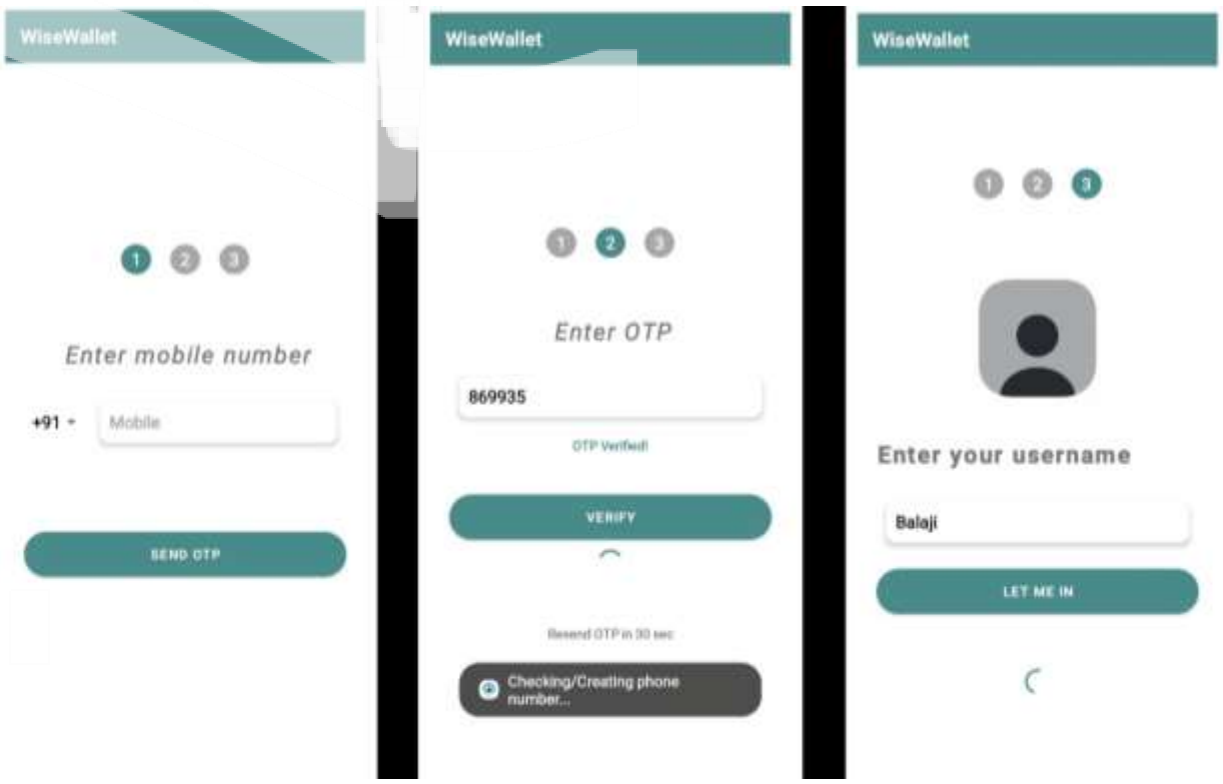
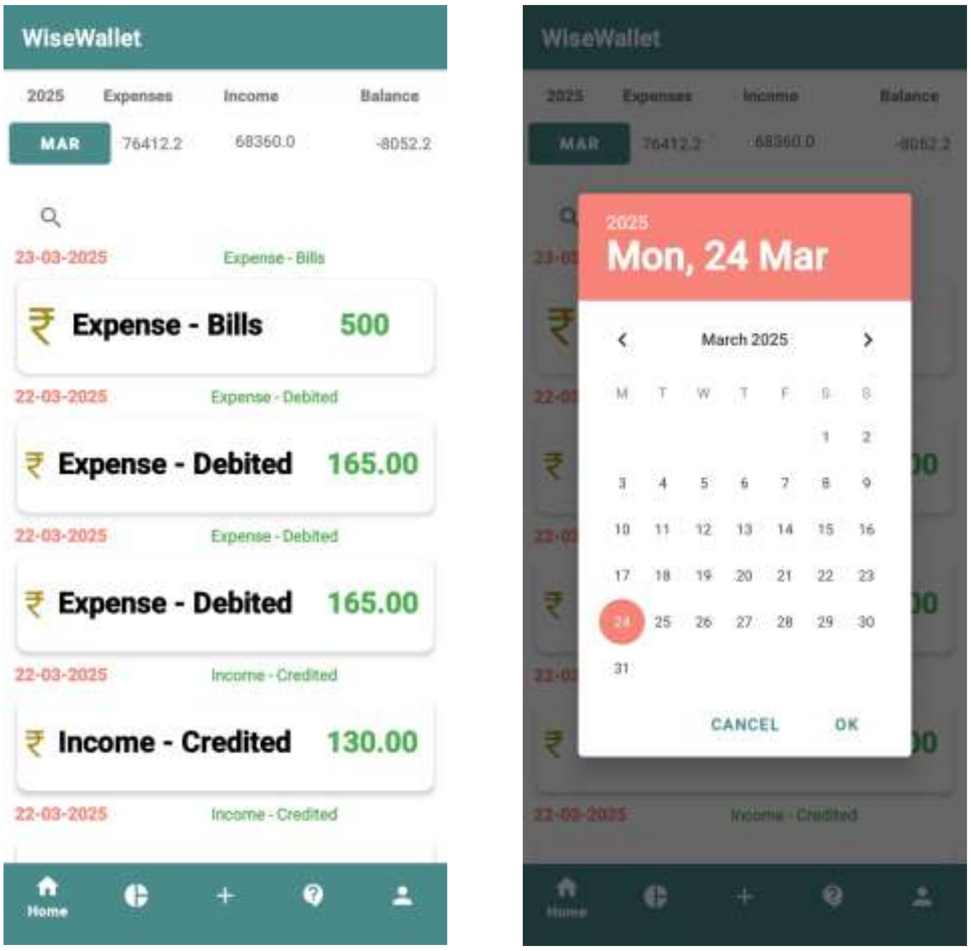


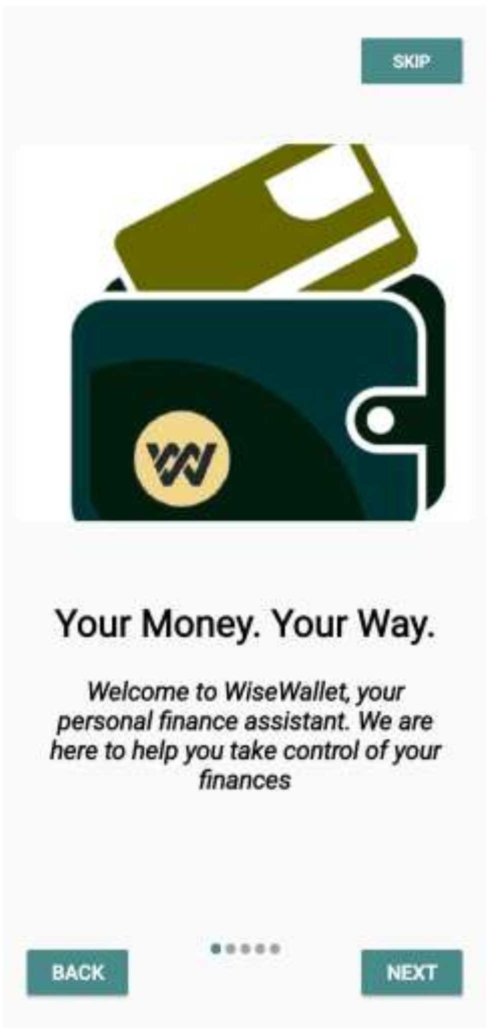
Fig 6.2 : Screenshots Showing Phone Number Authentication of User


WiseWallet
2025 Expenses Income Balance
MAR 76412.2 68360.0 -8052.2
23-03-2025 Expense - Bills
₹ Expense - Bills 500
22-03-2025 Expense - Debited
₹ Expense - Debited 165.00
22-03-2025 Expense - Debited
₹ Expense - Debited 165.00
22-03-2025 Income - Credited
₹ Income - Credited 130.00
22-03-2025 Income - Credited
Home
2025
Mon, 24 Mar
March 2025
M T W T F S S
CANCEL
OK

(a) (b)

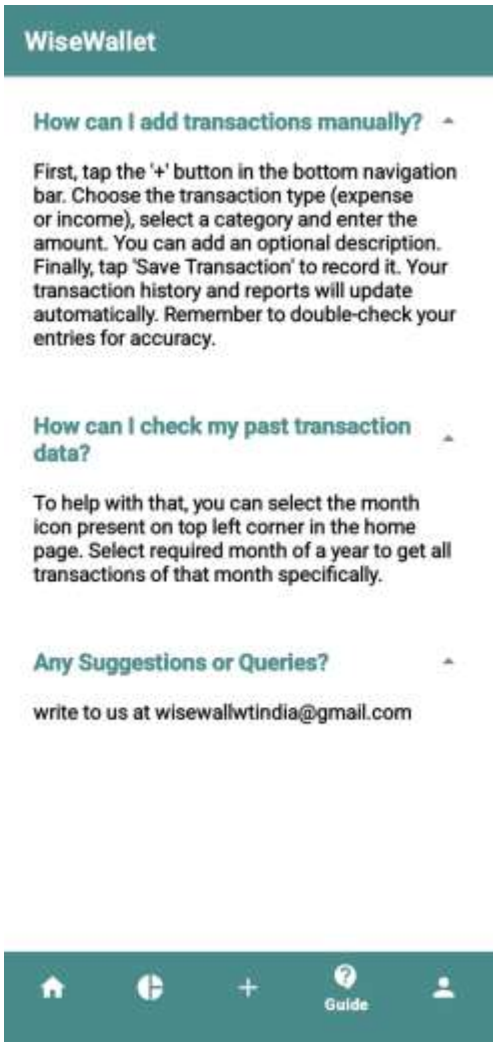
Fig 6.3 :

a. Home Screen displaying Expenses and Income values of all dates in March 2025.
b. Date Picker Dialog used to select date.



Your Money. Your Way.

Welcome to WiseWallet, your personal finance assistant. We are here to help you take control of your finances


WiseWallet

How can I add transactions manually?

First, tap the '+' button in the bottom navigation bar. Choose the transaction type (expense or income), select a category and enter the amount. You can add an optional description. Finally, tap 'Save Transaction' to record it. Your transaction history and reports will update automatically. Remember to double-check your entries for accuracy.

How can I check my past transaction data?

To help with that, you can select the month icon present on top left corner in the home page. Select required month of a year to get all transactions of that month specifically.

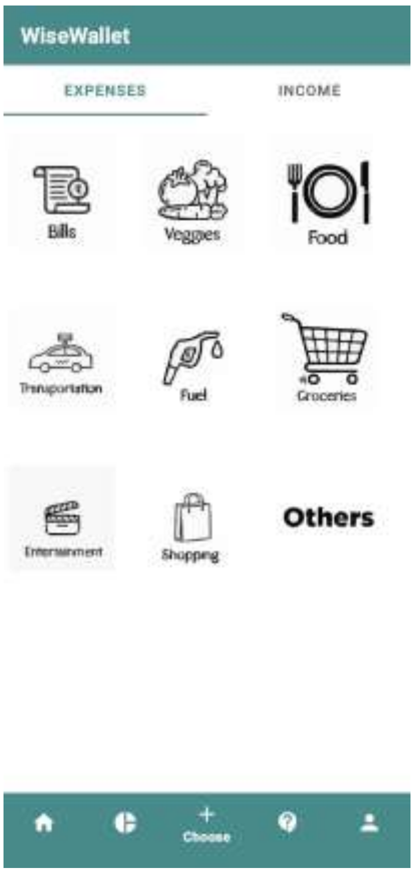
Any Suggestions or Queries?

write to us at wisewallwtindia@gmail.com

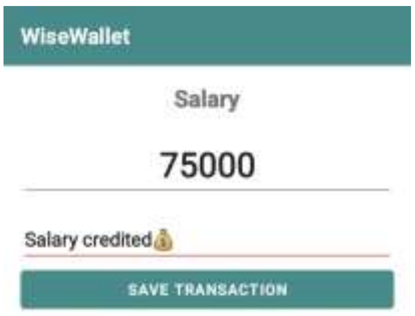
(a) (b)

Fig 6.4 :

a. Onboarding screen, to guide users.
b. User Guide to help resolve their doubts.



(a)



Salary

75000

Salary credited💰

(b)

Fig 6.5 :

a. Expense and Income Tabs to manually enter transactions.
b. Screenshot showing Transaction details to store.

CHAPTER 7
CONCLUSION

WiseWallet is designed to make expense tracking simple and stress-free. Many existing financial apps are complicated or require users to pay for essential features, which makes it difficult for people to manage their money effectively. WiseWallet solves this problem by offering easy expense tracking, budget management, and visual spending reports, allowing users to stay in control of their finances without any hassle. The app provides manual expense entry, where users can log their transactions quickly. It also includes budget tracking, so users can set spending limits and receive alerts when they are close to exceeding their budget. The visual charts and reports help users see where their money is going, making it easier to make smart financial decisions. With its simple design and user-friendly features, WiseWallet helps users build better spending habits without the need for expensive subscriptions.

To improve WiseWallet and make it even more useful and efficient, we plan to introduce several new features in future updates. One major improvement is SMS-based transaction tracking. This feature will allow the app to automatically detect bank transaction messages and update expenses accordingly, reducing the need for users to enter transactions manually. This will save time and effort while making the app more convenient. We also plan to add database storage usin, which will let users store and manage their expense history securely. This will make it easier for users to review past transactions, track spending trends, and generate financial reports over longer periods. For better visual analysis, we will integrate MPAndroidChart, which will allow users to view their expenses in graphs and charts based on weekly or monthly spending. This will make it easier for users to understand their spending habits and plan their finances accordingly.

REFERENCES

[1] S. Agarwal, R. Mittal, and P. Yadav, "A Review on Expense Tracking and Budgeting Applications," International Journal of Computer Applications, vol. 178, no. 5, pp. 1–5, Mar. 2019.

[2] T. K. Das and S. Mohan, "Automated Expense Tracking Using SMS Parsing," in Proceedings of the 2020 IEEE International Conference on Mobile and Secure Transactions (MST), New York, NY, USA, 2020, pp. 45–50.

[3] Y. Zhang and M. Kumar, "Personal Finance Management Systems: A Survey," IEEE Transactions on Consumer Electronics, vol. 66, no. 3, pp. 210–218, Sep. 2020.

[4] S. Gupta, R. Verma, and K. Rajan, "Enhancing Budget Tracking Efficiency through AI- Based Financial Analytics," in Proceedings of the 2021 IEEE International Conference on Data Science and Financial Technology (DSFT), Singapore, 2021, pp. 88–94.

[5] Priya Maury, Poorva Ghatkar, Mitali Rane, Sanika Bhosale " Case Study on Budget Tracker App" Vol 5, no 2, pp 3539-3544 February 2024. Information Technology, Thakur Polytechnic, Mumbai, Maharashtra, India. https://ijrpr.com/uploads/V5ISSUE2/IJRPR22995.pdf

[6] https://www.scribd.com/document/304987984/Budget-Monitoring-Application-a-thesis

[7] Kotlin and Android | Android Developers For gaining knowledge on Kotlin.

[8] www.Developer.android.com for resolving errors.

[9] John Horton (April 2019). *Android Programming with Kotlin for Beginners*. Build Android apps starting from zero programming experience with the new Kotlin programming language.

Appendix: A- Dependencies and Modules

Buildscript Dependencies

The *com.android.tools.build:gradle* plugin is fundamental for Android app development, automating the build process, managing dependencies, and integrating with the Android SDK. The *org.jetbrains.kotlin:kotlin-gradle-plugin* allows for Kotlin code compilation, bridging the gap between Kotlin and the Android build system.

Lastly, *com.google.dagger:hilt-android-gradle-plugin* simplifies dependency injection setup, promoting cleaner, more maintainable code through automated dependency management within Android projects.

Plugins

The *libs.plugins.android.application* plugin applies all necessary configurations to build an Android application, streamlining the process.

libs.plugins.kotlin.android specifically integrates Kotlin support into the build, ensuring proper compilation and execution.

libs.plugins.google.gms.google.services enable seamless integration with Google services like Firebase, handling configuration and setup for these external functionalities.

Android Dependencies

androidx.core.ktx enhances Android APIs with Kotlin extensions, improving code readability and efficiency. *androidx.appcompat* ensures UI consistency across Android versions, providing backward compatibility for modern UI features. It enables visually appealing and user-friendly interfaces.

androidx.constraintlayout allows for flexible UI layout creation, optimizing performance through efficient hierarchy management.

androidx.lifecycle.livedata.ktx and *androidx.lifecycle.viewmodel.ktx* facilitate lifecycle-aware data management, crucial for robust and maintainable Android apps.

Firebase dependencies, including *firebase.database* and *firebase.analytics*, provide real-time database capabilities and user behavior analytics, respectively. okhttp enables efficient HTTPS communication, allowing the app to interact with external servers.

Modules

android.Manifest :

- This provides access to Android's permission constants, like *READ_SMS*, crucial for requesting and managing app permissions related to sensitive user data and device functionalities.

android.content.pm.PackageManager:

- This class allows the app to interact with installed packages and retrieve information about them, including permission status.

android.content.Intent:

- Intents are used for inter-component communication, allowing the app to start activities, services, or broadcast events. They are fundamental for navigating between different screens and triggering actions within the Android system.

androidx.core.app.ActivityCompat:

- *This provides compatibility methods for requesting permissions and performing activity-related operations, ensuring consistent behavior across Android versions.*
- *It simplifies the process of requesting permissions, especially on older Android devices.*

androidx.core.content.ContextCompat:

- *This class offers compatibility methods for accessing context-related resources and checking permissions, abstracting away version-specific differences.*
- *It's used to safely check permissions and access resources without worrying about API level compatibility.*

android.app.DatePickerDialog:

- *This provides a standard dialog for users to select a date, simplifying date input in the app's UI.*
- *It offers a user friendly way to gather date information.*

android.content.Context:

- *Context provides access to application-specific resources and classes, such as shared preferences, file system, and system services.*
- *It's essential for interacting with the Android system and accessing application-level data.*

android.content.SharedPreferences:

- *This class allows the app to store and retrieve small amounts of key-value data, persisting user preferences or application settings.*
- *It is used for basic data storage within the app.*

android.os.Bundle:

- *Bundles are used to pass data between activities or fragments, allowing for the transfer of information during navigation or component communication.*
- *It is used to pass data between various android components.*

android.provider.Telephony:

- *This provides access to telephony-related functionality, including SMS messaging, allowing the app to interact with the device's SMS capabilities.*
- *It provides access to SMS related functions.*

android.util.Log:

- *The Log class is used for debugging and logging information during development, allowing developers to track application behavior and identify errors.*
- *It is used for debugging purposes.*

android.widget.* (Button, TextView, Toast, Toolbar):

- *These are standard Android UI components, providing building blocks for creating the app's user interface.*
- *They are basic UI elements.*

androidx.activity.enableEdgeToEdge:

- *This function is used to enable edge-to-edge display, allowing the app's content to extend into the system navigation and status bars.*
- *It allows the app to draw behind the system bars.*

androidx.appcompat.app.AppCompatActivity:

- *AppCompatActivity is the base class for activities that use the AppCompat library, providing backward compatibility for Material Design and other UI features.*
- *It provides backwards compatibility for modern UI elements.*

androidx.core.view.ViewCompat and androidx.core.view.WindowInsetsCompat:

- *These classes provide compatibility methods for working with views and window insets, allowing for consistent UI behavior across Android versions.*
- *They help in dealing with system UI elements, and screen space.*

androidx.fragment.app.Fragment:

- *Fragments are modular components that represent portions of a user interface within an activity, allowing for flexible and reusable UI design.*
- *They are used to create modular UI sections.*

com.example.wisewallet.fragments.*:

- *These import custom fragments specific to the application, encapsulating specific UI functionalities and logic within the app.*
- *They import app specific fragments.*

com.google.android.material.bottomnavigation.BottomNavigationView:

- *This provides a standard bottom navigation bar for easy navigation between top-level views in the app.*
- *It is used to create a bottom navigation bar.*

androidx.activity.result.contract.ActivityResultContracts:

- *This provides pre-built ActivityResultContracts, which are used to simplify launching activities for results, such as requesting permissions.*
- *It simplifies handling activity results.*

androidx.appcompat.app.AlertDialog:

- *This class allows for the creation of dialog boxes to display messages or gather user input, providing a standard way to interact with the user.*
- *It is used to create alert dialogs.*

com.google.firebase.database.*:

- *These imports provide classes and interfaces for interacting with Firebase Realtime Database, allowing the app to store and retrieve data in real-time.*
- *They are used for Firebase real time database interactions.*

kotlinx.coroutines.*:

- *These imports provide Kotlin coroutine functionality, allowing for asynchronous programming and simplifying background tasks.*
- *They are used to preform asynchronous operations.*

java.text.SimpleDateFormat and java.util.* (Calendar, Locale):

- *These classes are used for date and time formatting and manipulation, allowing the app to work with dates and times in a user-friendly way.*
- *They are used for date and time handling.*

java.util.regex.* (Matcher, Pattern):

- *These classes are used for regular expression matching, allowing the app to parse and extract specific data from text strings.*
- *They are used for text parsing.*

Appendix: B

Source Code Snippets

AndroidManifest.xml: (Fundamental Components declaration)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-feature
        android:name="android.hardware.telephony"
        android:required="false" />

    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:usesCleartextTraffic="true"
        android:networkSecurityConfig="@xml/network_configuration"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/trade_mark"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/trade_mark_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.WiseWallet"
        tools:targetApi="31">

        <activity android:name=".SplashActivity"
            android:exported="true"
            android:theme="@style/Theme.AppCompat.DayNight.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".NavigationActivity"
            android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"/>
        <activity
        android:name=".LoginPhoneNumberActivity"
        android:exported="false">
        <meta-data
            android:name="android.app.lib_name"
            android:value="" />
        </activity>
        <activity
            android:name=".LoginOtpActivity"
            android:exported="false">
            <meta-data
                android:name="android.app.lib_name"
                android:value="" />
        </activity>
        <activity
            android:name=".LoginUsernameActivity"
            android:exported="false">
            <meta-data
                android:name="android.app.lib_name"
                android:value="" />
        </activity>

        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.WiseWallet">

        </activity>
        <activity android:name=".ManualEntryActivity"
            android:exported="true"/>
        <activity android:name=".ManualEntryActivityIncome"
            android:exported="true"/>
    </application>

</manifest>
```

Fragment_expense_tab.xml: (Layout displaying image buttons to perform action)

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragments.ExpenseTab">

    <!-- TODO: Update blank fragment layout-->
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TableRow
            android:weightSum="3"
            android:layout_width="match_parent"
            android:paddingTop="10dip"
            android:paddingBottom="10dip"
            android:paddingStart="14dip"
            android:paddingEnd="14dip">
                <ImageButton
                    android:id="@+id/EIcon00"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:contentDescription="Veggies"
                    android:scaleType="fitCenter"
                    android:paddingBottom="20dip"
                    android:src="@drawable/bills" />
                <ImageButton
                    android:id="@+id/EIcon01"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:contentDescription="Transport"
                    android:scaleType="fitCenter"
                    android:paddingBottom="20dip"
                    android:src="@drawable/veggies" />
                <ImageButton
                    android:id="@+id/EIcon02"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:contentDescription="Shopping"
                    android:scaleType="fitCenter"
                    android:paddingBottom="20dip"
                    android:src="@drawable/food" />
        </TableRow>
        <TableRow
            android:weightSum="4"
            android:layout_width="match_parent"
            android:paddingTop="10dip"
            android:paddingBottom="10dip"
            android:paddingStart="14dip"
            android:paddingEnd="10dip">
                <ImageButton
                    android:id="@+id/EIcon10"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:contentDescription="Veggies"
                    android:scaleType="fitCenter"
                    android:paddingBottom="20dip"
                    android:src="@drawable/taxi" />

                <ImageButton
                    android:id="@+id/EIcon11"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:contentDescription="Transport"
                    android:scaleType="fitCenter"
                    android:paddingBottom="20dip"
                    android:src="@drawable/fuel" />

                <ImageButton
                    android:id="@+id/EIcon12"
                    android:backgroundTint="@color/white"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
```

```
                android:contentDescription="Shopping"
                android:scaleType="fitCenter"
                android:paddingBottom="20dip"
                android:src="@drawable/groceries" />
        </TableRow>
        <TableRow
          android:weightSum="4”
          android:paddingBottom="10dip"
          android:paddingStart="14dip"
          android:paddingEnd="10dip">
            <ImageButton
                android:id="@+id/EIcon20"
                android:backgroundTint="@color/white"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:contentDescription="Veggies"
                android:scaleType="fitCenter"
                android:paddingBottom="20dip"
                android:src="@drawable/entertainment" />

            <ImageButton
                android:id="@+id/EIcon21"
                android:backgroundTint="@color/white"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:contentDescription="Transport"
                android:scaleType="fitCenter"
                android:paddingBottom="20dip"
                android:src="@drawable/shopping" />

            <ImageButton
                android:id="@+id/EIcon22"
                android:backgroundTint="@color/white"
                android:scaleType="fitCenter"
                android:paddingBottom="20dip"/>
    </TableRow>
    </TableLayout>
</FrameLayout>
```

MainActivity.kt:(Activity for Message Parsing, Database Connectivity, Bottom Navigation)

```
package com.example.wisewallet
import android.Manifest
import android.content.pm.PackageManager
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import android.content.Context
import android.content.SharedPreferences
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.fragment.app.Fragment
import com.example.wisewallet.fragments.Charts
import com.example.wisewallet.fragments.Choose
import com.example.wisewallet.fragments.HomeFragment
import com.example.wisewallet.fragments.Me
import com.example.wisewallet.fragments.More
import com.google.android.material.bottomnavigation.BottomNavigationView
import androidx.activity.result.contract.ActivityResultContracts
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener
import java.text.SimpleDateFormat
import java.util.Calendar
import java.util.Locale
import java.util.regex.Matcher
import java.util.regex.Pattern

class MainActivity : AppCompatActivity() {
    private val permissionId_read=100
    private var hasAllPermissions=true
    private val permissionNameList= arrayOf(
        Manifest.permission.READ_SMS,
        Manifest.permission.RECEIVE_SMS,
        Manifest.permission.SEND_SMS)
    private val requestPermissionLauncher=registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ){isGranted ->
        if(isGranted){
            Toast.makeText(this,"Granted!",Toast.LENGTH_SHORT).show()
        }else{
            Toast.makeText(this,"Denied!",Toast.LENGTH_SHORT).show()
        }

    }
    private val requestMultiplePermissionsLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestMultiplePermissions()
        ) { permissions ->
            permissions.entries.forEach {
                val permissionName = it.key
                val isGranted = it.value
                if (isGranted) {
                    // Permission is granted. Continue the action that required the permission
                    Log.d("Permission", "$permissionName granted")

                } else {
                    Log.d("Permission", "$permissionName denied")
                    hasAllPermissions = false
                }
            }
        }
    private fun requestPermissions() {
        requestMultiplePermissionsLauncher.launch(permissionNameList)
    }

    lateinit var phoneNumber:String
    lateinit var user:String
    private lateinit var bottomNavigationView: BottomNavigationView
    private var isFirstLaunch: Boolean = true
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)//initial layout
```

```
requestPermissionLauncher.launch(android.Manifest.permission.READ_SMS)
requestPermissionLauncher.launch(android.Manifest.permission.SEND_SMS)
requestPermissionLauncher.launch(android.Manifest.permission.RECEIVE_SMS)

val sharedPrefs = getSharedPreferences("MyPrefs", Context.MODE_PRIVATE)
isFirstLaunch = sharedPrefs.getBoolean("isFirstLaunch", true)
if (isFirstLaunch) {
    // It's the first launch, request permissions
    requestPermissions()

    // Update the first launch flag in SharedPreferences
    val editor = sharedPrefs.edit()
    editor.putBoolean("isFirstLaunch", false)
    editor.apply()  // or editor.commit()
}
checkAndRequestSmsPermission()
// Check if all permissions are granted before proceeding
//Date picker moved to HomeFragment



//25dec Bottom navigation icons to their respective fragments
bottomNavigationView = findViewById(R.id.bottom_navigation)
bottomNavigationView.setOnItemSelectedListener { menuItem ->
    when(menuItem.itemId){//1.home fragment
        R.id.bottom_home-> {
            replaceFragment(HomeFragment())
            true
        }
        //2.charts
        R.id.charts-> {
            replaceFragment(Charts())
            true
        }
        //3.Choose
        R.id.choose-> {
            replaceFragment(Choose())
            true
        }
        //4.More
        R.id.more-> {
                replaceFragment(More())
                true
            }
            //5.Me (profile)
            R.id.Me-> {
                replaceFragment(Me())
                true
            }
            else -> false
        }

    }
    replaceFragment(HomeFragment())//default

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
        val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
        insets
    }
}
private fun checkAndRequestSmsPermission() {
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_SMS) !=
PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.READ_SMS),
permissionId_read)
    } else {
        retrieveTransactionSMS()
    }
}
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults:
IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == permissionId_read) {
        if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            retrieveTransactionSMS()
        } else {
            Log.e("MainActivity", "SMS permission denied")
        }
    }
}
```

```
    private fun retrieveTransactionSMS() {
        val smsMessages = getTransactionSMS(this)
        for (message in smsMessages) {
            Log.d("MainActivity", "SMS: $message")
            // Display the message in your UI or process it as needed
        }
    }

    fun getTransactionSMS(context: Context): MutableList<String> {
        val sharedPrefs: SharedPreferences = context.getSharedPreferences("SMSPref",
Context.MODE_PRIVATE)
        val lastProcessedDate = sharedPrefs.getLong("lastUsedDate", 0L) // 0L means no date stored yet
        val transactionMessages = mutableListOf<String>()
        val uri = android.net.Uri.parse("content://sms/inbox")
        val projection = arrayOf("_id", "address", "body", "date")
        val selection = " LOWER(body) LIKE ? or LOWER(body) LIKE ? or LOWER(body) LIKE ?"
        val selectionArgs = arrayOf("a/c *____%", "a/c X____%", "a/c X%X____%")
        val sortOrder = "date ASC"

        val cursor: android.database.Cursor? = context.contentResolver.query(uri, projection, selection,
selectionArgs, sortOrder)
        cursor?.use {
            val bodyIndex = it.getColumnIndex("body")
            val dateIndex = it.getColumnIndex("date")
            var latestProcessedDate = lastProcessedDate
            while (it.moveToNext()) {
                val messageBody = it.getString(bodyIndex)
                val messageDate = it.getLong(dateIndex)
                transactionMessages.add(messageBody)
                if (messageDate > latestProcessedDate) {
                    // Extract year and month from message date
                    val calendar = Calendar.getInstance()
                    calendar.timeInMillis = messageDate
                    val year = calendar.get(Calendar.YEAR).toString()
                    val month = SimpleDateFormat("MMM", Locale.getDefault()).format(calendar.time)
                        .uppercase(Locale.getDefault())
                        latestProcessedDate = maxOf(latestProcessedDate, messageDate)
                    // Check for keywords and add to Firebase accordingly
                    if (messageBody.contains("credited", true)) {
```

```
                    addToFirebase("Income", "Credited", messageBody, year, month)
                } else if (messageBody.contains("debited", true)) {
                    addToFirebase("Expense", "Debited", messageBody, year, month)
                }

            }
        }
        val editor = sharedPrefs.edit()
        editor.putLong("lastUsedDate", latestProcessedDate)
        editor.apply()


    }
    return transactionMessages
}
private fun addToFirebase(transactionType: String, operation: String, message: String, year: String, month:
String) {
    val database = FirebaseDatabase.getInstance()
    val sharedPreferences: SharedPreferences = this.getSharedPreferences("UserData",
Context.MODE_PRIVATE)
    phoneNumber = sharedPreferences.getString("phoneNumber", null).toString()
    user = sharedPreferences.getString("user", null).toString()
    val newPh=sharedPreferences.getString("newPh", null).toString()
    // Extract amount and date from the message
    val amount = extractAmount(message)
    val date = extractDate(message)

    if (amount != null && date != null) {
        val transactionRef =
database.getReference("users").child(newPh).child(year).child(month).child(transactionType).child(operation).p
ush()
        val transactionData = hashMapOf(
            "amount" to amount,
            "date" to date
        )
        transactionRef.setValue(transactionData)
            .addOnSuccessListener {
                // Data added successfully
            }
            .addOnFailureListener {
                // Handle errors
              }
            val operationRef =
database.getReference("users").child(newPh).child(year).child(month).child(transactionType).child(operation)
            operationRef.addListenerForSingleValueEvent(object : ValueEventListener {
              override fun onDataChange(snapshot: DataSnapshot) {
                var totalAmount = 0.0
                for (transactionSnapshot in snapshot.children) {
                  val amountStr = transactionSnapshot.child("amount").getValue(String::class.java)
                    ?.toDoubleOrNull()
                    ?: 0.0
                  totalAmount += amountStr
                }
                operationRef.child("total").setValue(totalAmount)
                  .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                      // Subcategory total updated successfully, now update Expense total
                      updateExpenseTotal(newPh, year, month)
                      updateIncomeTotal(newPh, year, month)
                    } else {
                      Log.e("Firebase", "Error updating subcategory total: ${task.exception?.message}")
                    }
                  }
              }

              override fun onCancelled(error: DatabaseError) {
                Log.e("Firebase", "Error calculating total: ${error.message}")
              }
            })
          } else {
            // Handle cases where amount or date extraction fails
            Log.e("MainActivity", "Failed to extract amount or date from message: $message")
          }
        }
        private fun extractAmount(message: String): String? {
          val pattern: Pattern = Pattern.compile("Rs[.:\\s]?(\\d+\\.\\d{2})")
          val matcher: Matcher = pattern.matcher(message)
          return if (matcher.find()) {
            matcher.group(1)
          } else null
```

```
    }

    private fun extractDate(message: String): String? {
        val pattern: Pattern = Pattern.compile("on (\\d{2}-\\d{2}-\\d{4})")
        val matcher: Matcher = pattern.matcher(message)
        return if (matcher.find()) {
            matcher.group(1)
        } else null
    }
    private fun updateExpenseTotal(phoneNumber: String, year: String, month: String) {
        val database = FirebaseDatabase.getInstance()
        val expenseRef =
database.getReference("users").child(phoneNumber).child(year).child(month).child("Expense")

        expenseRef.addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                var grandTotal = 0.0
                for (operationSnapshot in snapshot.children) {
                    val subTotal = operationSnapshot.child("total").getValue(Double::class.java) ?: 0.0
                    grandTotal += subTotal
                }
                expenseRef.child("total").setValue(grandTotal)
            }

            override fun onCancelled(error: DatabaseError) {
                Log.e("Firebase", "Error updating Expense total: ${error.message}")
            }
        })
    }
    private fun updateIncomeTotal(phoneNumber: String, year: String, month: String) {
        val database = FirebaseDatabase.getInstance()
        val expenseRef =
database.getReference("users").child(phoneNumber).child(year).child(month).child("Income")

        expenseRef.addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                var grandTotal = 0.0
                for (operationSnapshot in snapshot.children) {
                    val subTotal = operationSnapshot.child("total").getValue(Double::class.java) ?: 0.0
                    grandTotal += subTotal
                }
                expenseRef.child("total").setValue(grandTotal)
            }

            override fun onCancelled(error: DatabaseError) {
                Log.e("Firebase", "Error updating Expense total: ${error.message}")
            }
        })
    }
    private fun replaceFragment(fragment:Fragment){         //new fragment  current fragment
        //replaces current fragment with new fragment
        supportFragmentManager.beginTransaction().replace(R.id.frame_container, fragment).commit()
        //then the changes applied to fragment container are commited
    }
}
```

***