SIMPLESPEND – A SIMPLE APP TO TRACK SPENDING

CS19611 - MOBILE APPLICATION DEVELOPMENT LAB MINI PROJECT REPORT

Submitted by

UMESH SUBRAMANIAN S **(2116220701307)**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

MAY 2025

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project titled **"SimpleSpend – A simple app to track spending"** is the bonafide work of **"UMESH SUBRAMANIAN S (2116220701307)"** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. P. Kumar., M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of Computer Science and Engineering,

Rajalakshmi Engineering College, Chennai - 602 105.

SIGNATURE

Dr. N. Duraimurugan., M.E., Ph.D.,

SUPERVISOR

Associate Professor

Department of Computer Science and Engineering,

Rajalakshmi Engineering College, Chennai-602 105.

Submitted to Mini Project Viva-Voce Examination held on _____

Internal Examiner **External Examiner**

ABSTRACT

In today's fast-paced digital world, managing personal finances efficiently is essential for every individual. SimpleSpend is a lightweight mobile application developed as part of the Mobile Application Development curriculum to provide users with a simple and intuitive way to manage daily monetary transactions.

This application allows users to deposit and withdraw funds, maintain a real-time balance, and view their transaction history with timestamps. Designed with a minimalistic user interface and built entirely using Kotlin for the Android platform, the app demonstrates key mobile development concepts such as dynamic UI creation, event handling, input validation, and responsive feedback using toast messages and alert dialogs.

By simulating a basic wallet system, SimpleSpend serves as a beginner-friendly yet practical tool that showcases foundational programming and UI design skills while addressing a real-world problem in personal finance management. The app is intended for users who seek a simple, offline solution for tracking their day-to-day financial activities without the need for internet connectivity or complex features.

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guides **Dr. N. DURAIMURUGAN,** We are very glad to thank our Project Coordinator, **Dr. N. DURAIMURUGAN** Associate Professor Department of Computer Science and Engineering for his useful tips during our review to build our project.

UMESH SUBRAMANIAN S 2116220701307

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

1.1 GENERAL

Managing personal finances effectively is a crucial skill in modern life. With the increasing reliance on digital solutions, mobile applications have become a popular and convenient medium for handling everyday financial activities. However, many existing finance apps are often overloaded with features, require internet connectivity, or demand user registration, which may not appeal to users seeking simplicity and ease of use.

SimpleSpend is a streamlined mobile application developed to address this need. It offers users a basic yet functional platform to track their daily financial transactions—specifically deposits and withdrawals—while maintaining a live display of their current balance. The app is designed with a focus on clarity, minimalism, and offline usability, making it suitable for users of all age groups who prefer straightforward financial tracking without the complexity of modern fintech tools.

The core idea behind SimpleSpend is to provide an intuitive interface that makes it easy to input, view, and manage transactions. Each transaction is timestamped and categorized, offering a clear and organized view of the user's financial activity. With features like real-time balance updates, transaction history, and the ability to clear past records, the app emphasizes simplicity without sacrificing functionality.

1.2 OBJECTIVE

The main objectives of the SimpleSpend application are:

- To provide a user-friendly interface for managing simple financial transactions

such as deposits and withdrawals.

- To display a real-time balance that updates automatically based on user inputs.
- To maintain a clear transaction history, including amount, type (deposit or withdrawal), and timestamp.
- To allow users to clear transaction history when desired, giving them control over stored records.
- To ensure offline functionality so that users can use the app without requiring internet access.
- To demonstrate fundamental concepts of mobile application development, including event handling, dynamic UI design, and basic data management using Kotlin.
- To create a secure and private environment where no external data sharing or user registration is required.
-

1.3 EXISTING SYSTEM

In the current digital landscape, there are numerous personal finance and expense management applications available on various platforms. These applications offer a wide range of features including budget tracking, financial goal setting, bank account integration, and analytics. While these features can be beneficial for advanced users, they often introduce complexity and require internet connectivity, user authentication, and access to sensitive financial data.

Many of the existing systems:

- Require users to sign up or log in, which may not be ideal for those seeking a quick, private solution.
- Operate only with an active internet connection, limiting usability in offline scenarios.
- Include complex user interfaces and excessive features that may overwhelm users who need only basic transaction tracking.
- Store financial data in the cloud, which may raise concerns over privacy and data security for some users.

Due to these limitations, there is a need for a lightweight and user-friendly mobile application that provides only the essential features for personal finance tracking, without compromising usability, privacy, or accessibility.

CHAPTER 2

2.1 SOFTWARE DESCRIPTION

The **Workout Planner App** is developed as a native Android application using **Kotlin** as the primary programming language. The user interface is designed with **XML layouts**, and **SQLite** is used for local data storage. The development environment used is **Android Studio**, which provides robust tools for building, testing, and debugging Android apps.

The app follows a simple modular architecture separating the user interface, logic, and database operations. Data entered by the user—such as workout name, equipment used, number of sets, reps, and weight—is stored locally and persists between app sessions. The user interacts with the app through intuitive form-based inputs, and the stored data is displayed in a scrollable list with options to delete individual entries.

The key focus areas for the software include:

- Offline functionality (no internet required)
- Ease of use for beginners
- Fast performance and responsiveness
- Clean, minimal design adhering to Android guidelines

2.2 LANGUAGES

2.2.1 KOTLIN

Kotlin is a modern, statically typed programming language developed by JetBrains and officially supported by Google for Android app development. It was chosen for this project due to its concise syntax, improved safety features (such as null safety), and seamless interoperability with Java. Kotlin enables the development of robust and efficient Android applications with cleaner and more readable code.

In SimpleSpend, Kotlin was used to:

- Implement the core logic for handling deposits, withdrawals, and balance updates.
- Dynamically create and manage the user interface components at runtime.
- Handle user input, dialog interactions, and toast notifications.
- Manage the transaction history and UI updates in real-time.

2.2.2 XML

XML is widely used in Android development for defining static resources, configuration files, and user interface elements. Although the main UI in SimpleSpend is constructed programmatically using Kotlin, XML is still essential for other components of the app.

In this project, XML was used to:

Define application metadata in AndroidManifest.xml.

Manage app themes, icons, and colors.

Configure backup rules and other resources.

By combining Kotlin for logic and dynamic UI, and XML for configuration, the application maintains a clean structure while demonstrating flexibility in Android development practices.

CHAPTER 3

3.1 REQUIREMENT SPECIFICATION

1. Hardware Requirements

- The application requires an Android smartphone or emulator to run.
- The device should have a processor of at least quad-core 1.4 GHz.
- A minimum of 2 GB RAM is recommended for smooth performance.
- At least 50 MB of free internal storage is required.
- A screen size of 4.5 inches or more is preferred for better display.

2. Software Requirements

- The operating system should be Android version 6.0 (Marshmallow) or higher.
- The development environment used is Android Studio.
- The programming language used is Kotlin.
- The project uses the Gradle build system.
- The app targets Android SDK version 31 or higher.

3. Functional Requirements (FRs)

- FR1: Display Current Balance
    - The application should show the current balance to the user as soon as the app is launched. This balance should update in real-time based on user transactions.
- FR2: Deposit Money

    - The user should be able to click a button to initiate a deposit.
    - An input dialog should appear to allow the user to enter an amount.
    - Upon confirmation, the amount should be added to the current balance and logged in the transaction history with a timestamp.
- FR3: Withdraw Money
    - The user should be able to click a button to withdraw money.
    - A dialog should appear to enter the amount to withdraw.
    - The system should validate that the balance is sufficient before allowing the withdrawal.
    - If valid, the amount should be deducted from the balance and recorded in the transaction history.
- FR4: Maintain Transaction History
    - The app should maintain a visual list of all deposits and withdrawals made during the current session.
    - Each transaction should display its type (deposit or withdrawal), the amount, and the time of the transaction.
- FR5: Clear Transaction History
    - The app should provide a button to clear the transaction history.
    - When this button is pressed, the transaction list should be cleared, and a message should notify the user.
    - The balance should remain unchanged when history is cleared.

---

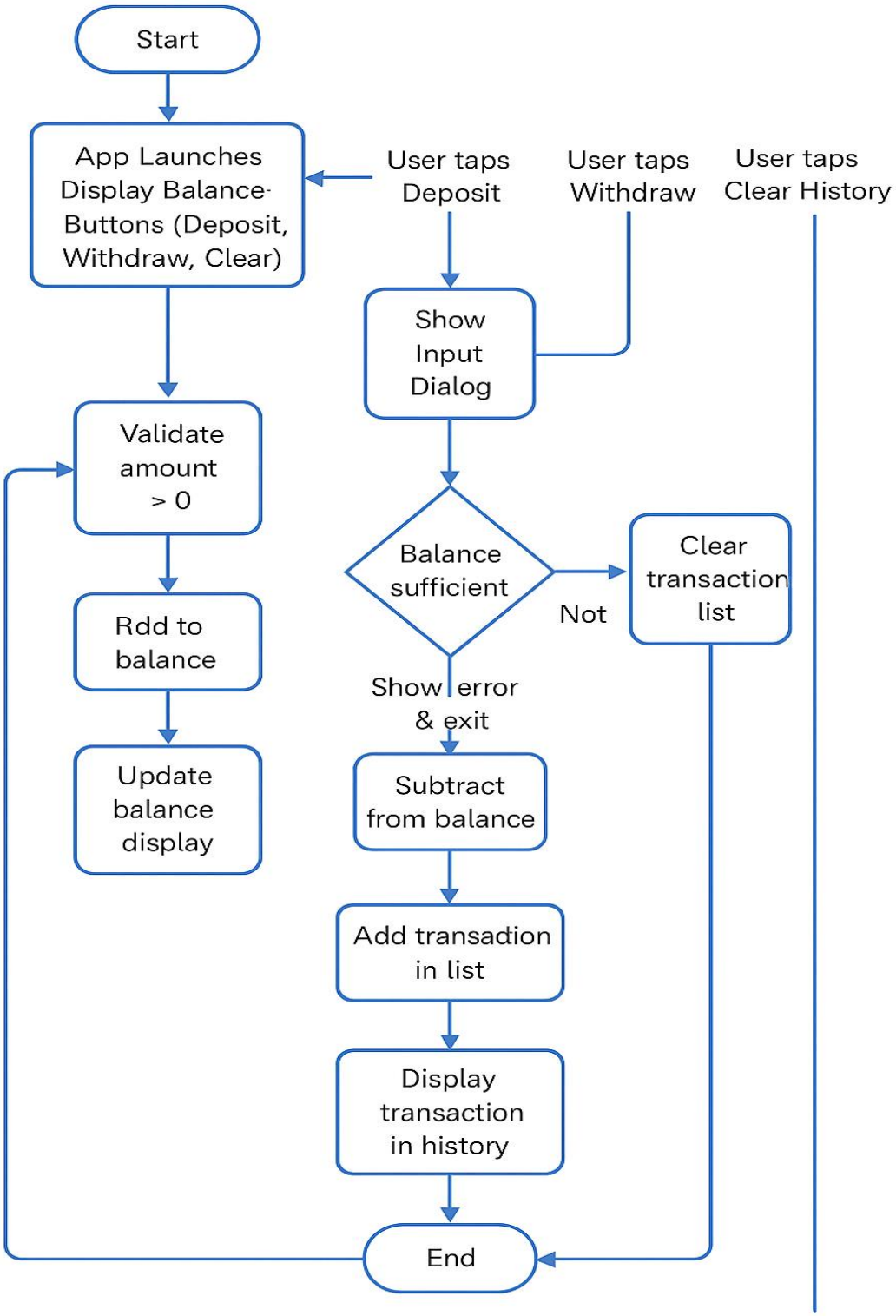
4. Non-Functional Requirements (NFRs)

- NFR1: User-Friendly Interface

    - The app should have a clean, intuitive, and easy-to-navigate user interface.
    - Visual elements such as color coding, emojis, and spacing should enhance usability.
- NFR2: Input Validation and Feedback
    - The app should handle invalid inputs gracefully.
    - For example, it should display an error message if the user tries to enter a negative amount or withdraw more money than available.
- NFR3: Performance and Responsiveness
    - The application should launch within 2 seconds.
    - All operations like deposits, withdrawals, and clearing history should be processed instantly, with no noticeable lag.
- NFR4: Offline Functionality
    - The application should be fully operational without requiring an internet connection.
    - All data is stored in memory during the session, so the app works seamlessly offline.
- NFR5: Reliability
    - The app should handle common runtime issues, such as empty inputs or back button presses, without crashing.

3.3 ARCHITECTURE DIAGRAM

The app follows a three-tier architecture:



CHAPTER 4
PROGRAM CODE

MainActivity.kt

```
package com.example.transactionapp

import android.graphics.Color
import android.os.Bundle
import android.text.InputType
import android.view.Gravity
import android.widget.*
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.setPadding
import java.text.SimpleDateFormat
import java.util.*

class MainActivity : AppCompatActivity() {

    private lateinit var balanceText: TextView
    private lateinit var transactionList: LinearLayout
    private var balance = 10000.0
    private val transactions = mutableListOf<Transaction>()

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

        val scrollView = ScrollView(this).apply {
            setBackgroundColor(Color.parseColor("#E3F2FD"))
            layoutParams = LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.MATCH_PARENT
            )
        }

        val container = LinearLayout(this).apply {
            orientation = LinearLayout.VERTICAL
            setPadding(32)
        }

        val titleText = TextView(this).apply {
            text = "🏦 Simple Transaction App"
```

```
    textSize = 26f
    setTextColor(Color.parseColor("#0D47A1"))
    gravity = Gravity.CENTER
    layoutParams = LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT
    ).apply {
        bottomMargin = 40
    }
}

balanceText = TextView(this).apply {
    text = "💰 Balance: ₹$balance"
    textSize = 22f
    setTextColor(Color.BLACK)
    gravity = Gravity.CENTER
    layoutParams = LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT
    ).apply {
        bottomMargin = 30
    }
}

val depositButton = Button(this).apply {
    text = "➕ Deposit Money"
    setBackgroundColor(Color.parseColor("#64B5F6"))
    setTextColor(Color.WHITE)
    setOnClickListener { showTransactionDialog("Deposit") }
}

val withdrawButton = Button(this).apply {
    text = "➖ Withdraw Money"
    setBackgroundColor(Color.parseColor("#1976D2"))
    setTextColor(Color.WHITE)
    layoutParams = LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT
    ).apply {
        topMargin = 16
    }
```

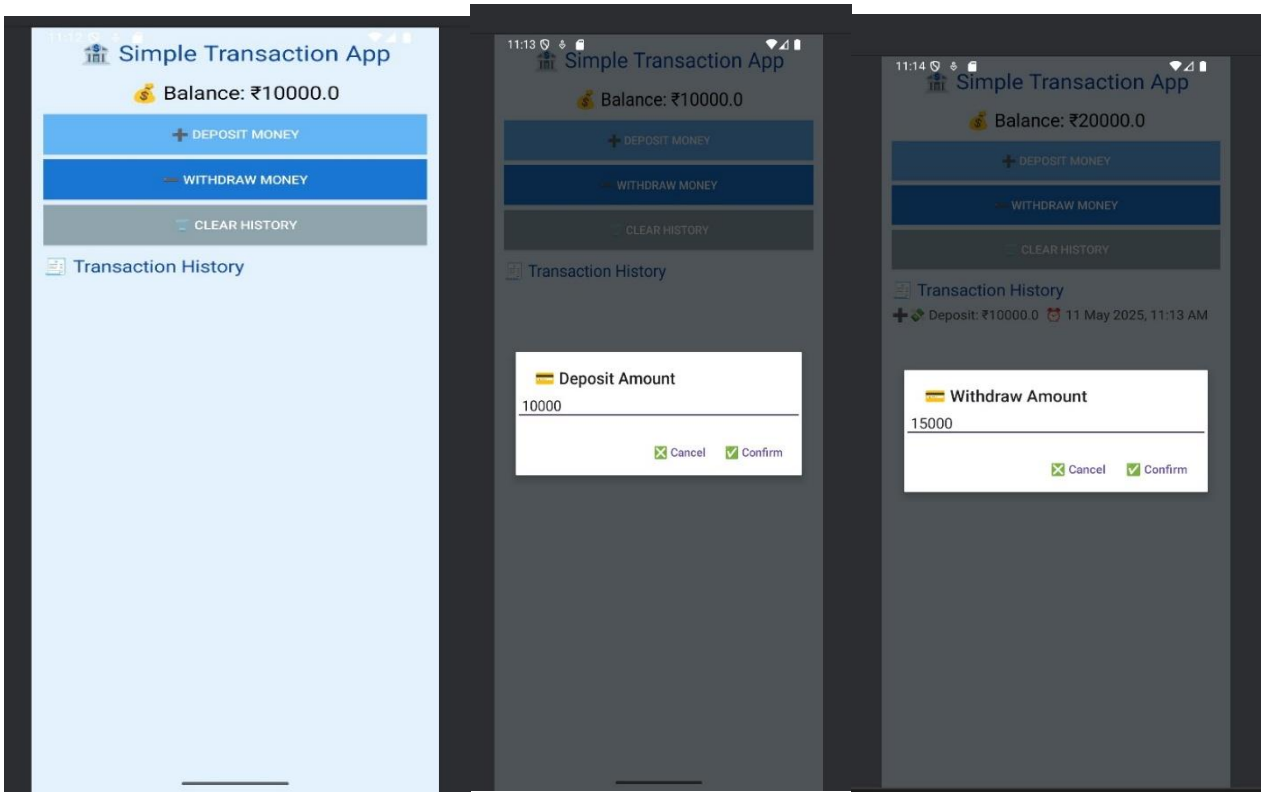
```
        setOnClickListener { showTransactionDialog("Withdraw") }
    }

    val clearButton = Button(this).apply {
        text = "🗑️ Clear History"
        setBackgroundColor(Color.parseColor("#90A4AE"))
        setTextColor(Color.WHITE)
        layoutParams = LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ).apply {
            topMargin = 16
            bottomMargin = 30
        }
        setOnClickListener {
            transactions.clear()
            transactionList.removeAllViews()
            Toast.makeText(this@MainActivity, "🗑️ Transaction history cleared",
Toast.LENGTH_SHORT).show()
        }
    }

    val historyTitle = TextView(this).apply {
        text = "📜 Transaction History"
        textSize = 20f
        setTextColor(Color.parseColor("#0D47A1"))
        layoutParams = LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ).apply {
            bottomMargin = 16
        }
    }

    transactionList = LinearLayout(this).apply {
        orientation = LinearLayout.VERTICAL
    }

    container.addView(titleText)
    container.addView(balanceText)
    container.addView(depositButton)
```

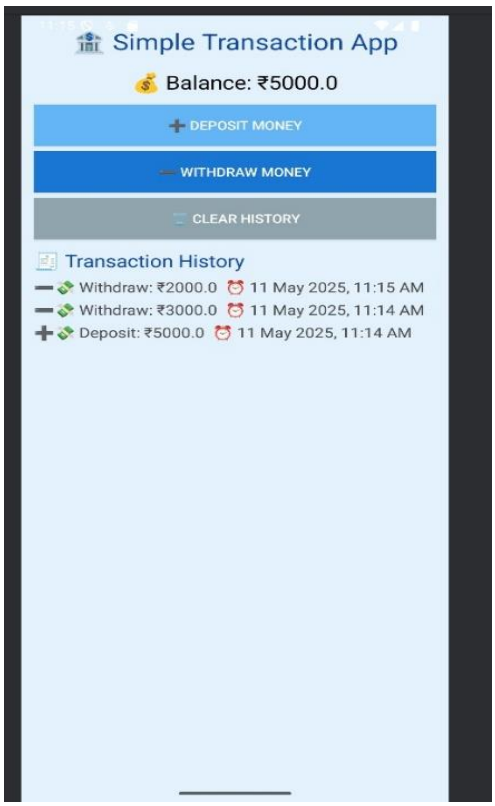
```
        container.addView(withdrawButton)
        container.addView(clearButton)
        container.addView(historyTitle)
        container.addView(transactionList)

        scrollView.addView(container)
        setContentView(scrollView)
    }

    private fun showTransactionDialog(type: String) {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("💳 $type Amount")

        val input = EditText(this).apply {
            inputType = InputType.TYPE_CLASS_NUMBER or
InputType.TYPE_NUMBER_FLAG_DECIMAL
            hint = "Enter amount 💵"
            setPadding(20)
        }

        builder.setView(input)

        builder.setPositiveButton("✅ Confirm") { _, _ ->
            val amount = input.text.toString().toDoubleOrNull()

            if (amount == null || amount <= 0) {
                Toast.makeText(this, "❌ Invalid amount",
Toast.LENGTH_SHORT).show()
                return@setPositiveButton
            }

            if (type == "Withdraw" && amount > balance) {
                Toast.makeText(this, "🚫 Insufficient balance",
Toast.LENGTH_SHORT).show()
                return@setPositiveButton
            }

            val transaction = Transaction(
                amount = amount,
                type = type,
                timestamp = System.currentTimeMillis()
```

```
            )

            transactions.add(0, transaction)

            if (type == "Deposit") {
                balance += amount
            } else {
                balance -= amount
            }

            balanceText.text = "💰 Balance: ₹$balance"
            displayTransaction(transaction)
        }

        builder.setNegativeButton("❌ Cancel") { dialog, _ -> dialog.cancel() }

        builder.show()
    }

    private fun displayTransaction(transaction: Transaction) {
        val formatter = SimpleDateFormat("dd MMM yyyy, hh:mm a",
Locale.getDefault())
        val emoji = if (transaction.type == "Deposit") "➕💵" else "➖💵"
        val view = TextView(this).apply {
            text = "$emoji ${transaction.type}: ₹${transaction.amount}  ⏰
${formatter.format(Date(transaction.timestamp))}"
            textSize = 16f
            setTextColor(Color.DKGRAY)
            layoutParams = LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.WRAP_CONTENT
            ).apply {
                bottomMargin = 12
            }
        }
        transactionList.addView(view, 0)
    }

    data class Transaction(
        val amount: Double,
        val type: String,
```

```
        val timestamp: Long
    )
}
```

CHAPTER 5: OUTPUT




Simple Transaction App
Balance: ₹5000.0
DEPOSIT MONEY
WITHDRAW MONEY
CLEAR HISTORY
Transaction History
Withdraw: ₹2000.0 11 May 2025, 11:15 AM
Withdraw: ₹3000.0 11 May 2025, 11:14 AM
Deposit: ₹5000.0 11 May 2025, 11:14 AM

CHAPTER 6

RESULTS AND DISCUSSION

Results

The Simple Transaction App was successfully implemented with the following observable results:

1. Functional Balance Tracking
    - The app initializes with a default balance
    - All transactions (deposit or withdrawal) immediately reflect in the displayed balance.
2. User-Driven Transactions
    - Users can enter an amount through a dialog box.
    - Deposits increase the balance, and withdrawals decrease it only if the balance is sufficient.
3. Transaction History Logging
    - Each transaction is recorded and displayed in a scrollable list.
    - Entries are timestamped and include appropriate icons for deposit and withdrawal actions.
4. Data Reset Capability
    - Users can clear the transaction history at any time with a single click.
    - The balance remains unaffected by this action.
5. User Interface Feedback
    - Toast messages notify the user of successful transactions, invalid inputs, or errors (e.g., insufficient funds).

6. Visual Appeal

    - Use of emojis, consistent color schemes, and layout spacing improves user experience and clarity.

Discussion

The application meets its core objective of enabling users to simulate basic money management activities. The programmatic UI approach provided direct control over layout and behavior, making it flexible and lightweight.

Some important observations and reflections include:

- Simplicity and Usability: The straightforward UI and interaction model make the app intuitive even for users with no technical background.

- Validation Checks: The inclusion of error handling (e.g., rejecting invalid or excessive withdrawal amounts) enhances reliability.

- Scalability: While the app is well-suited for small-scale personal use or demonstrations, it would require enhancements like persistent storage (e.g., SQLite or Room), input formatting, and multi-user support to scale further.

CHAPTER 7

CONCLUSION

The SimpleSpend App successfully demonstrates the fundamental principles of Android application development using Kotlin. It provides users with the ability to perform essential financial operations—depositing and withdrawing funds—while dynamically updating the balance and maintaining a transaction history within a single activity.

The app's intuitive design, minimalistic interface, and responsive behavior make it an effective educational tool for learning event-driven programming, UI construction in code, and basic data handling. Despite its simplicity, the app lays a strong foundation for more complex features such as persistent storage, multi-user support, data analytics, and integration with external services.

In conclusion, this project not only achieved its intended functionality but also offered valuable hands-on experience with Android development workflows, user interaction management, and Kotlin programming best practices.

CHAPTER 8

REFERENCES

1. Android Developers Documentation – Kotlin
2. Stack Overflow
3. Android Studio Guides
4. GeeksforGeeks – Android Projects
5. Kotlin Lang Documentation
6. Android Jetpack Docs: RecyclerView, Room, ViewModel, LiveData