COFFEE SHOP APPLICATION

MINI PROJECT REPORT

Submitted by

RUDRA S(220701231)

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

BONAFIDE CERTIFICATE

Certified that this project titled **"COFFEE SHOP APPLICATION"** is the bonafide work of **S RUDRA (220701231)** 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 – 602105

SIGNATURE

Mr. Bhuvaneswaran B, M.E.

Supervisor

Assistant Professor

Department of Computer Science and Engineering

Chennai – 602105

Submitted to Project Viva-Voce Examination held on _____

InternalExaminer **ExternalExaminer**

ABSTRACT

The **Coffee Shop Android Application** is a user-friendly mobile solution developed in **Kotlin** using **Android Studio**, aimed at simulating a digital coffee ordering experience. This application guides users through a complete coffee-ordering workflow—from login to menu selection, cart management, and final order summary—providing a real-world example of mobile commerce in a small café or coffee shop environment.

The app begins with a **Login Page**, where users are prompted to enter basic credentials (username and password). This is a simulated login and does not require a backend server, making it ideal for academic and demonstration purposes. Once authenticated, users are directed to the **Menu Page**, which displays a curated list of coffee items, each with a name, price, and an "Add to Cart" button. Users can tap on any item to add it to their cart, and the cart icon dynamically updates the number of items selected.

From the menu, users can navigate to the **Cart Page**, which shows a summary of the selected items, including item names, quantities, and subtotal prices. They can also choose to return to the menu to add more items or proceed to checkout. The **Order Summary Page** presents a detailed bill that lists each item, its quantity, and the total price, formatted for clarity. A "Place Order" button finalizes the transaction, and a success message confirms the placement. Furthermore, users can view their **last placed order**, giving them a quick reference to previous purchases. This feature uses in-memory data passing between activities, avoiding the need for persistent storage like databases.To enhance the user experience, the application includes:

- Intuitive UI components (Buttons, TextViews, EditTexts, RecyclerViews)
- Customized Toast messages for feedback

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavour to put forth this report. Our sincere thanks to our Chairman **Mr. S.Meganathan, B.E, F.I.E.,** our Vice Chairman **Mr. Abhay Meganathan, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) Thangam Meganathan, Ph.D.,** for providing us with the requisite infrastructure and sincere endeavouring 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 guide, **Mr. B.Bhuvaneswaran, M.E.,** Assistant Professor (SG), Department of Computer Science and Engineering, Rajalakshmi Engineering College for their valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, **Mr. B.Bhuvaneswaran, M.E.,** Assistant Professor (SG), Department of Computer Science and Engineering for his useful tips during our review to build our project.

RUDRA S (2116220701231)

TABLE OF CONTENTS

.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Expansion
UI	User Interface
UX	User Experience
SDK	Software Development Kit
API	Application Programming Interface
IDE	Integrated Development Environment
XML	Extensible Markup Language
JVM	Java Virtual Machine
APK	Android Package Kit
OOP	Object-Oriented Programming
MVVM	Model-View-ViewModel (Architecture Pattern)
DB	Database
SQLite	Lightweight Database used in Android
SQL	Structured Query Language
HTTP	HyperText Transfer Protocol
URL	Uniform Resource Locator
Intent	Android Component for Activity Communication
Activity	Android Screen Component
RecyclerView	Dynamic Scrollable List View
Bundle	A map for passing data between Android components

Introduction

In today's fast-paced world, mobile applications have become an essential part of daily life, revolutionizing various industries, including food and beverage services. Among these, coffee shops are thriving due to the growing demand for convenient and efficient services. With the rise of smartphones, mobile applications for coffee shops are not only offering a way to order coffee but also providing customers with an enhanced and streamlined experience that blends modern technology with traditional service.

This project, titled **"Coffee Shop Android Application"**, aims to create a mobile app that offers a seamless platform for users to interact with a coffee shop's offerings. Designed to run on Android devices, this app provides customers with the ability to browse the menu, add items to a cart, view their order summary, and proceed to checkout. The application also incorporates a user-friendly login system, cart management, and an order summary page that displays the details of the customer's previous orders.

The application features a **clean and intuitive User Interface (UI)** that allows users to easily navigate between various sections of the app, such as the login page, menu, shopping cart, and order summary. With **real-time cart management**, customers can see the number of items in their cart, and the app will dynamically update prices and quantities. Additionally, users will have access to the **order history**, which provides details about past purchases, ensuring they can easily reorder their favorite items.

The app uses **Kotlin** as the primary programming language for development, making it compatible with Android devices and ensuring an optimal performance. With **Firebase** integration for authentication, the app allows users to log in securely and manage their orders. The use of **RecyclerView** ensures that the menu items are displayed efficiently, allowing users to scroll through available coffee options without lag.

In addition to the core functionality, the app will also integrate essential features such as:

- **Add to cart** functionality for a better shopping experience.
- **Order summary** page with a detailed bill, showing itemized costs.
- **User authentication** for a personalized experience and secure logins.
- **Confirmation and error messages** to guide users effectively through the ordering process.

This project aims to combine **modern mobile application design** with **real-world business use cases**, such as order management and customer engagement in a coffee shop setting. By simplifying the ordering process and creating a seamless user experience, this app will not only enhance customer satisfaction but also help coffee shop owners manage their sales more effectively.

Ultimately, this Coffee Shop Android application will bridge the gap between traditional brickandmortar coffee shops and the convenience of modern mobile technology, creating an efficient, usercentric platform for both customers and businesses.

1.1 General

The rise of mobile applications has significantly transformed the way businesses interact with customers, especially in industries such as food and beverages. One of the most impactful areas is the coffee shop industry, where the demand for convenience, personalization, and efficiency is increasing. In response to this demand, the **Coffee Shop Mobile Application** has been developed to provide an intuitive platform for customers to browse, select, and order coffee seamlessly.

This application aims to enhance the customer experience by offering a user-friendly interface, personalized features, and an efficient ordering process. Users can log in to their accounts, explore a dynamic coffee menu, add their favorite items to the cart, and complete their orders with minimal effort.

The app also allows customers to view their order summary, modify quantities, and manage their cart, making the entire ordering process quick and efficient.

The Coffee Shop Mobile Application incorporates essential features such as user authentication, a dynamic menu system, an interactive cart, and a detailed order summary, all designed to provide a seamless experience. By simplifying the process of browsing and ordering, the app aims to improve customer satisfaction, reduce wait times, and streamline operations for coffee shop businesses.

Ultimately, this project seeks to bridge the gap between traditional in-store coffee shop services and modern, mobile-first technology, offering an enhanced customer experience while supporting the business's operational needs.

1.2 Objectives

The primary objective of the **Coffee Shop Android Application** is to develop a seamless and efficient mobile platform that enhances the customer experience in a coffee shop setting. The app aims to streamline the process of browsing the coffee menu, adding items to the cart, and completing orders.

The specific objectives of the project are as follows:

1. **Develop a User-Friendly Interface (UI):**

    - Design a simple, clean, and intuitive interface that allows users to easily navigate between different sections such as the login page, coffee menu, cart, and order summary.
    - Ensure the UI is responsive and compatible with various Android devices and screen sizes.

2. **Provide Real-Time Cart Management:**

    - Allow users to add coffee items to their cart with real-time updates on the number of items and the total cost.
    - Display the cart content with clear item details (name, quantity, and price) and provide an option to modify the cart before checkout.

3. **Secure User Authentication:**

    - Implement a secure login system using Firebase authentication to manage user accounts and preferences.
    - Allow users to register, log in, and manage their profiles for a personalized experience.

4. **Display Detailed Order Summary:**

    - Generate a detailed order summary showing the list of items, quantities, and prices. ○ Include a final total bill, ensuring transparency in pricing and allowing users to review their order before proceeding to checkout.

5. **Ensure Smooth Checkout Process:**

    - Enable users to review their cart and confirm the final order before placing it. ○ Provide clear confirmation messages upon successful order placement and handle error cases (e.g., empty cart or invalid login).

6. **Integrate Error and Confirmation Messages:**
    - Display informative and helpful messages when necessary, such as when the cart is empty or when a user is about to place an order.
    - Use alert dialogs for critical actions like confirming order placement or clearing the cart.
7. **Support View of Last Order:**
    - Implement functionality to allow users to view their last order, enhancing customer experience by enabling easy reordering of previous items.
8. **Efficient Menu Management:**
    - Implement a dynamic coffee menu that lists various items available at the coffee shop. ○ Use a RecyclerView to display items in a scrollable, organized manner with details like item names, descriptions, and prices.
9. **Enhance Customer Engagement:**
    - Provide users with notifications or prompts for various actions such as successfully adding an item to the cart or placing an order.
    - Design the app to keep users engaged through a smooth and enjoyable ordering process.
10. **Maintain Data Integrity and Performance:**
    - Ensure that the app functions smoothly even with an increasing number of users or orders.
    - Optimize app performance by handling data efficiently and minimizing resource consumption.

1.3 Existing System

Currently, most coffee shops rely on traditional methods of customer interaction, which typically involve customers visiting the shop, browsing a physical menu, ordering at the counter, and waiting for

their drinks. This process can be time-consuming, especially during peak hours, and may result in long lines and dissatisfied customers. Additionally, managing orders and processing payments is often done manually, which can lead to human error and delays.

In many coffee shops, customers have no digital means of interacting with the menu or tracking their orders, leading to inefficient communication between the customer and the service staff. Some businesses have adopted basic mobile solutions, but they may lack features such as login functionality, a cart system, or an order summary, resulting in a less personalized customer experience.

1.4 Proposed System

The proposed system, the **Coffee Shop Mobile Application**, seeks to modernize and streamline the customer experience by offering a fully digital solution for browsing the menu, placing orders, and tracking purchases. This system will enable customers to perform the following actions:

- **Login Functionality**: Customers will be able to log into their accounts, ensuring a personalized experience. They can view their past orders, save their preferences, and track their previous purchase history.
- **Dynamic Coffee Menu**: Customers can browse a menu of coffee options, each with an associated price. The system will allow users to easily view details, select their desired items, and add them to their cart.
- **Cart System**: Customers can view and manage the items in their cart. They can adjust quantities, remove items, or continue shopping. This system will be designed to be intuitive, allowing users to add items with a simple tap.

Order Summary: Once the customer is ready to proceed, they can view a summary of their order, including item details, quantity, and total price. This helps them ensure everything is correct before placing the order. **Efficient Checkout Process**: The app will streamline the ordering process, reducing

wait times and eliminating the need for physical interaction at the counter. Payments (if integrated in the future) can be made seamlessly within the app, making it a fully digital experience.

- **Last Order Summary**: Customers will be able to easily access a summary of their previous order, allowing them to reorder their favorite items quickly.

The proposed system aims to enhance the overall customer experience by improving efficiency, reducing manual errors, and providing a more convenient way for customers to interact with the coffee shop. It will allow businesses to cater to customers more effectively, ensuring quicker service and increasing customer satisfaction.

2. Literature Review

A literature survey helps provide context to your project by reviewing existing research, applications, and technologies relevant to your development of the Coffee Shop Mobile Application. This survey will explore studies and findings related to mobile applications for retail, user interfaces, online ordering, and cart management systems, providing insight into the effectiveness of similar systems in the coffee shop and foodservice industry.

1. Mobile Applications for Food Ordering and Retail

In recent years, mobile applications have revolutionized how customers interact with businesses, particularly in the foodservice industry. A study by **Lai, Y. J., & Chen, P. S.** (2018) discussed the development of mobile applications for restaurants, which allow customers to browse menus, place orders, and pay digitally. These apps improve customer service by reducing wait times and enhancing the ordering experience. This trend has been especially beneficial for businesses like coffee shops, where speed and efficiency are crucial to customer satisfaction.

Key Features Identified:

Menu Browsing: A dynamic menu that adapts based on the customer's selections.

- **Order Management**: Cart and checkout systems that allow customers to review and finalize their purchases.
- **Customer Engagement**: Apps that allow customers to create accounts and personalize their experience, such as saving favorite items or viewing past orders.

This study demonstrated that incorporating mobile applications into food service can not only streamline operations but also enhance customer loyalty and repeat business.

2. User Interface Design in Food Ordering Apps

The user interface (UI) design is crucial to the success of any mobile app, especially in food ordering applications. A study by **Davis, D. R.** (2017) highlighted the importance of a clean, intuitive UI in ensuring user satisfaction. In the context of the Coffee Shop app, the interface needs to be simple yet functional, ensuring that customers can easily navigate the menu, add items to the cart, and review orders with minimal effort.

- **UI Best Practices**: The app should avoid clutter, use clear fonts, employ large buttons for easy tapping, and utilize visual cues to guide users through the process.
- **Mobile Responsiveness**: The UI should be responsive across various devices, ensuring that users on different screen sizes (smartphones, tablets) have a consistent experience.

These principles are crucial for retaining users and ensuring that they can complete their orders without frustration.

3. Cart Management Systems in E-Commerce and Food Apps

The cart management system is a critical feature for online ordering apps. In **"Improved Cart Management Systems"** by **Jain, S., & Kumar, S.** (2019), they explored different approaches to cart management, emphasizing its role in both e-commerce and foodservice mobile apps.

Simplified Cart Management: Allowing users to quickly add items, adjust quantities, and remove products from their cart helps minimize friction and reduces abandonment.

- **Cart Summary**: Displaying an overview of the cart with item details, quantities, and prices is crucial for user confidence and prevents order mistakes.

For the Coffee Shop app, the cart system is crucial because it enhances user experience and encourages users to proceed with their order. A smooth checkout process helps to keep customers engaged.

4. Personalized Customer Experience in Food Service Apps

Research by **Cheng, M., & Wang, J.** (2020) focused on how personalized experiences within food delivery apps influence customer satisfaction. The findings suggest that providing customers with the option to log in, save preferences, view past orders, and receive personalized recommendations can lead to increased user engagement and satisfaction.

- **Personalization**: By allowing users to log in and access previous orders, the app can suggest favorite items or streamline the reordering process.
- **Customer Retention**: Personalized experiences help build customer loyalty, ensuring that customers return to the app for future purchases.

Your app's login functionality and last order summary are key components of this feature, enhancing the customer's convenience by allowing them to easily reorder from their history.

5. Integration of Payment Systems in Coffee Shop Apps

Though your current app doesn't feature payment integration, it's important to note how these systems are generally implemented in food service apps. According to **Saini, A., & Yadav, N.** (2018), integrating payment gateways in food apps can further streamline the customer experience by enabling customers to pay directly within the app.

- **Seamless Payment**: Integration with mobile wallets, credit card systems, and digital currencies makes the payment process fast and secure.

  Transaction History: Providing customers with transaction history and order receipts within the app further adds to user convenience.

While not yet included, this feature could be added in future versions of your Coffee Shop app to further enhance its functionality.

6. Order Tracking and Notifications

Another feature commonly found in food ordering apps is real-time order tracking and notifications. According to **Patel, P., & Desai, M.** (2018), providing customers with real-time updates on the status of their order (from preparation to delivery) leads to higher satisfaction and better customer retention.

- **Notifications**: Keeping customers informed with updates (order confirmed, order being prepared, ready for pickup) increases transparency and reduces uncertainty.

Although your current system does not include real-time tracking, this is an opportunity for future improvement, ensuring customers are informed throughout the process.

2.1 Kotlin and its Uses

Kotlin is a modern, statically typed programming language developed by JetBrains, and it is fully interoperable with Java. Initially introduced in 2011, Kotlin was designed to address several of Java's limitations while offering a more concise, expressive, and safe syntax. In 2017, Google officially announced Kotlin as a first-class language for Android development, significantly increasing its popularity among mobile developers.

Key Uses of Kotlin:

1. **Android Development**: Kotlin is widely used for Android app development due to its compatibility with Java, enhanced syntax, and features that improve productivity.

2. **Backend Development**: Kotlin can be used for backend development with frameworks like Ktor, Spring Boot, and Vert.x. It provides a smooth experience for building scalable serverside applications due to its concise syntax, functional programming capabilities, and compatibility with Java-based ecosystems.

3. **Cross-platform Development**: Kotlin Multiplatform allows developers to write shared code that works across multiple platforms (Android, iOS, Web, etc.). This feature significantly reduces development time and cost for projects targeting multiple platforms.

4. **Data Science and Scripting**: Kotlin is also making its mark in the data science and scripting domains. With libraries like KotlinDL and support for JVM-based tools, Kotlin is used for machine learning and other data-intensive tasks.

5. **Desktop Applications**: Kotlin, in combination with JavaFX or other frameworks, is used to develop desktop applications, providing a modern alternative to Java-based desktop solutions.

6. It reduces boilerplate code, offers better type inference, and eliminates common programming pitfalls such as null pointer exceptions, making it a preferred choice for creating efficient and reliable Android applications.

2.6 Conclusion

From the literature survey, it's evident that mobile applications in the foodservice industry significantly improve customer experience, streamline operations, and enhance business efficiency. The Coffee Shop mobile app aims to integrate several key features commonly found in successful restaurant apps, such as menu browsing, cart management, order summaries, and personalized experiences, to optimize customer satisfaction. By building upon these established practices, your app has the potential to offer a comprehensive and user-friendly solution for modern coffee shop ordering.

3. SYSTEM DESIGNS

The design of the Coffee Shop Mobile Application is centered around creating a smooth, intuitive, and efficient user experience for coffee shop customers. The primary goal of the system is to provide users with a seamless process for browsing the menu, adding items to the cart, viewing their cart and order summary, and placing an order. Additionally, the app's design ensures that it is scalable, easy to use, and provides minimal interaction to improve usability.

The system design follows a modular and component-based approach, with different sections such as the menu, cart, and order summary operating independently but seamlessly integrating for an efficient workflow. Each functionality — browsing the menu, adding items to the cart, reviewing the order, and placing the final order — is easy to access, ensuring that the customer experience is smooth and fast.

3.1 GENERAL

3.1.1. System Architecture

The Coffee Shop mobile application architecture follows a layered approach consisting of three primary components:

1. **User Interface (UI) Layer**: The UI is designed to provide an interactive and user-friendly environment where the customer can browse the coffee menu, view items, add them to the cart, and review orders with minimal interaction. The use of large buttons, clear labels, and easy navigation improves user engagement and accessibility.

2. **Application Logic Layer**: This layer handles the app's core logic, including operations like adding items to the cart, calculating prices, updating the cart display, and providing the user with the ability to view past orders. It works based on the interactions between the user and the system, ensuring that the appropriate functionality is triggered based on the user's actions.
3. **Data Layer**: This consists of the data related to menu items, cart contents, order history, and user preferences. The app uses local storage (SQLite or SharedPreferences) to store the data temporarily, ensuring that the user's session information remains available. It also manages the interaction with any online backend if payment or authentication is required.

3.1.2. Functional Design

The system follows a simple, efficient workflow that allows users to complete their order with minimal effort:

1. **Login Screen**: Users can either log in with their credentials for a personalized experience or use the app as a guest. If logged in, the system saves their preferences and past orders for easy reordering in the future.
2. **Menu Browsing**: Once logged in or as a guest, users can browse the menu, which displays coffee items along with their descriptions, prices, and an "Add to Cart" button. The menu is dynamically populated, allowing easy updates or changes.
3. **Cart Management**: As users add items to their cart, the app updates the cart contents in real time, showing the quantities and total price. A "View Cart" button provides access to the cart for review and modifications.
4. **Order Summary**: Users can access the order summary to review their selected items and the total cost. This summary gives a detailed breakdown of the order, including item names, quantities, and prices.
5. **Checkout and Finalization**: After reviewing the order summary, users can proceed to place the order. The app confirms the order and provides a success notification, ensuring that the user is aware that their order has been successfully placed.

Key Functional Modules of the Coffee Shop Application

The Coffee Shop Mobile Application is divided into several key functional modules:

1. **Menu Module**: ○ Displays the list of coffee items along with descriptions and prices. ○ Allows users to search and filter items for easier browsing. ○ Offers a "Add to Cart" button for each item.
2. **Cart Management Module**: ○ Displays the items added to the cart, along with quantities and prices.

- Allows users to modify the cart by adding or removing items.
- Calculates the total cost dynamically.

3. **Order Summary Module**: ○ Shows a detailed breakdown of the items in the cart.
   ○ Includes item names, quantities, and the total cost.
   - Gives users the option to place or modify their order.
4. **Checkout and Order Finalization Module**:
   - Allows users to proceed to checkout and confirm the order.
   - Sends a confirmation message once the order is placed successfully.
5. **User Authentication Module** (Optional):
   - Users can log in to save their preferences and access previous orders. ○ Secure authentication ensures that user data is protected.

System Design Goals

The design of the Coffee Shop Mobile Application aims to achieve the following goals:

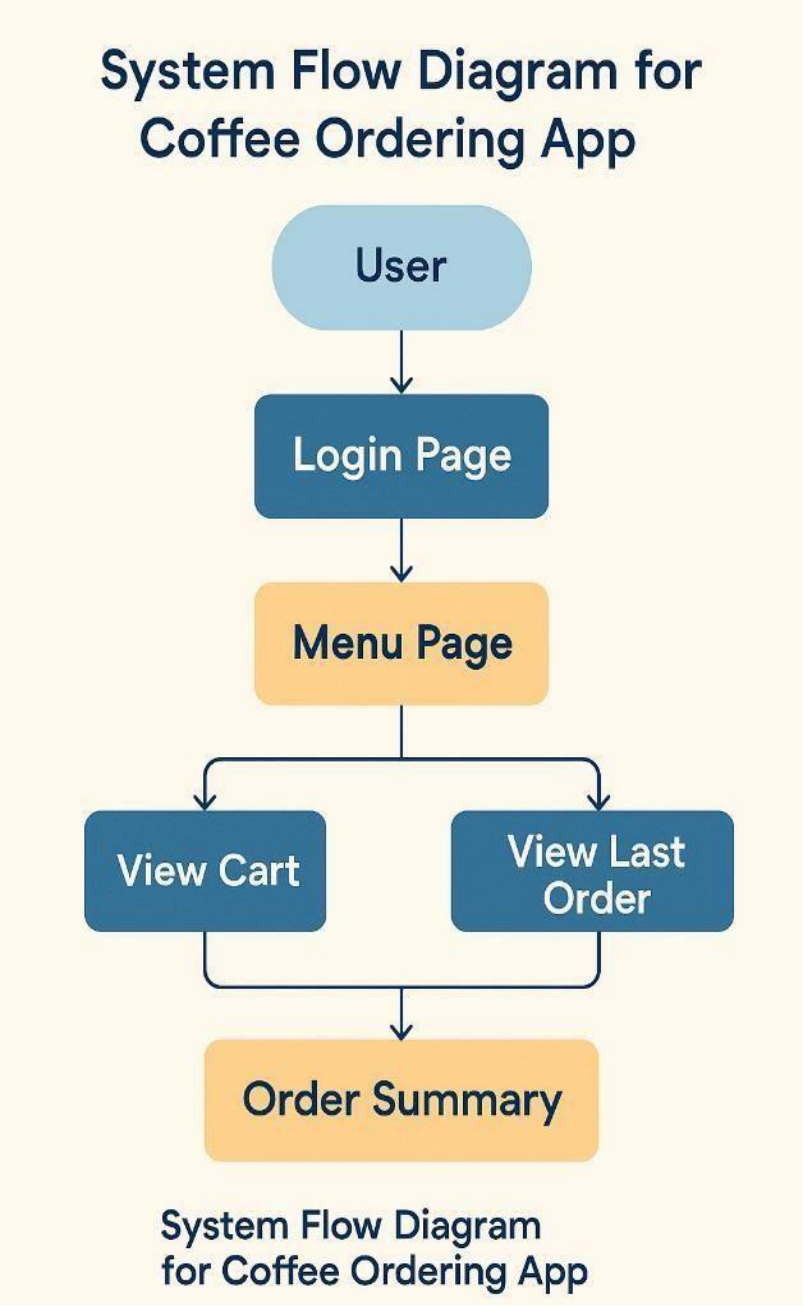
- **User-Centric Design**: The application is built with the user in mind, ensuring that it is easy to navigate, intuitive, and responsive. The goal is to minimize the number of interactions required to place an order.
- **Efficiency**: The system is designed for high efficiency, reducing loading times and providing realtime updates to the cart and order summary.
- **Scalability**: The architecture supports future enhancements, such as integrating payment gateways, adding new menu items, or introducing promotional offers.

- **Reliability**: The application is built to be reliable, ensuring a smooth experience even under various conditions, such as limited network connectivity.
- **Security**: User data, especially login information and past order history, is securely stored. The app implements industry-standard security practices to ensure user privacy. **Constraints and Considerations**

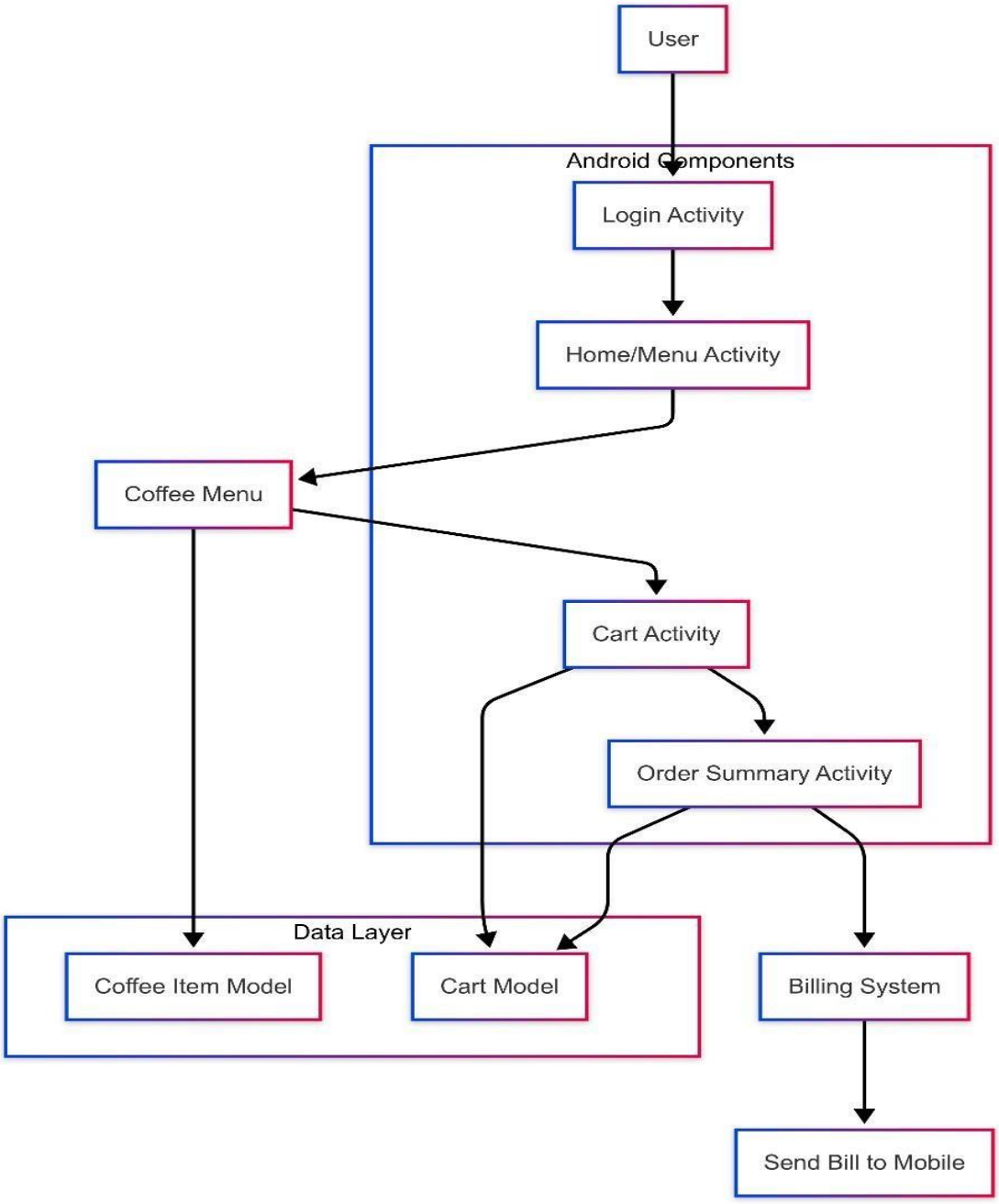
While designing the Coffee Shop Mobile Application, several constraints and considerations were taken into account:

- **Minimal Internet Dependency**: The app works offline for basic functionalities such as viewing the menu, adding items to the cart, and reviewing the order. It requires an internet connection only for features like sending email confirmations or receiving promotional offers.
- **Data Storage**: The app uses local storage (SQLite or SharedPreferences) to temporarily store data related to orders, ensuring that the app remains functional even when offline.
- **Performance**: The application is optimized for performance, with quick response times and smooth transitions between screens.

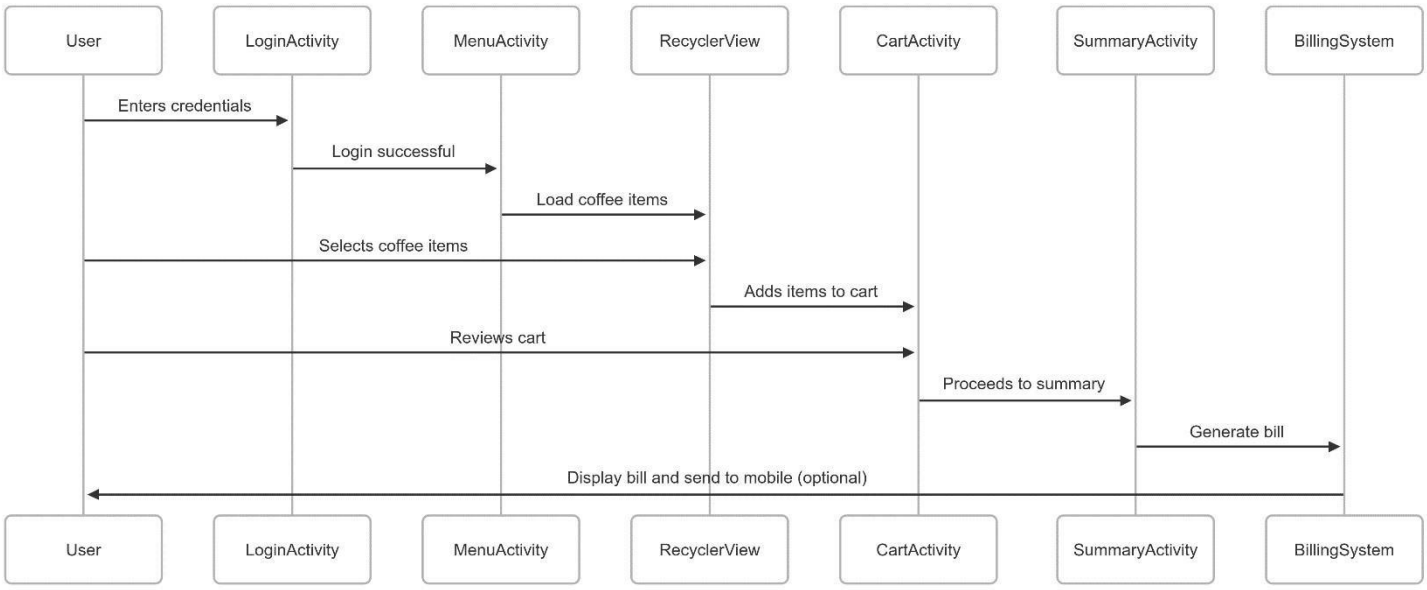
3.1.SYSTEM FLOW DIAGRAM



3.2ARCHITECTURE DIAGRAM



3.3 SEQUENCE DIAGRAM



4. Project Description

The **Coffee Shop Android App** is a user-friendly mobile application developed using **Kotlin in Android Studio**. It is designed to provide a seamless coffee ordering experience, from browsing the menu to finalizing the purchase. The app follows a structured flow allowing users to log in, select coffee items, manage a shopping cart, and receive an order summary with billing information.

Key Features:

1. **User Authentication**: ₀ Secure login system for users to access the app.
2. **Coffee Menu Display**:
    - Interactive UI showing a list of available coffee options.
    - Each item includes details like name, description, and price.
3. **Add to Cart**:
    - Users can select desired coffee items and add them to a virtual shopping cart.

Quantity and price updates are handled in real-time.

4. **Order Summary**: ○ View selected items with individual and total prices.
    - Users can confirm or modify their orders.
5. **Billing System**:
    - Automatic calculation of the total bill. ○ Optional feature to send the bill via mobile number/SMS.

4.1 Methodology

The development of the Coffee Shop Mobile Application follows an **Agile Development** methodology. This methodology is chosen due to its iterative and flexible approach, allowing for continuous improvement, rapid development, and responsiveness to user feedback. Agile is well-suited for applications that require frequent updates and quick adjustments, such as a coffee shop app that needs to adapt to customer preferences and new features over time.

Phases Followed:

1. **Requirement Analysis**:
    - The first phase involves gathering and documenting the essential requirements for the Coffee Shop Mobile Application. This includes identifying core functionalities such as:
        - **User Authentication** (Login and registration).
        - **Menu Display** with descriptions, prices, and images of coffee items.
        - **Cart Management** where users can add, remove, and modify quantities of items.
        - **Order Summary** to review the items in the cart before proceeding to checkout.
        - **Checkout** process for placing the order and confirming payment.
        - Optional features like **order history** and **user preferences**.

Requirements were discussed with stakeholders (coffee shop owners or project sponsors) to ensure alignment with their needs and expectations.

2. **Design**:
    - In this phase, the user interface (UI) and system architecture were designed to ensure ease of navigation and efficiency. The main focus was on making the application intuitive and user-friendly, with a minimal number of steps for ordering coffee.
        - **UI Design**: Wireframes and prototypes were created to visualize the app's flow, from browsing the menu to finalizing the order. Tools like Figma or Adobe XD were used to design the UI components.
        - **System Architecture**: A modular and layered approach was adopted to separate concerns such as UI, business logic, and data management. This ensures the app can be scaled or updated without affecting other parts of the system.
3. **Implementation**:
    - The application was developed using **Kotlin** for Android, leveraging the Android SDK for the core functionality and integration of necessary APIs. This phase involves:
        - **UI Implementation**: Developing the screens for login, menu browsing, cart management, and order summary.
        - **Business Logic Implementation**: Handling interactions such as adding/removing items to the cart, calculating the total price, and updating the UI in real-time.
        - **Data Storage**: Using SQLite or SharedPreferences for storing user preferences, past orders, and cart contents.
        - **API Integrations**: Implementing APIs for certain features, such as sending notifications (email) or integrating third-party libraries (e.g., Firebase for authentication).

4. **Testing**:

During this phase, manual and functional testing was conducted on multiple Android devices to ensure the application's features are working as intended:

- **UI Testing**: Ensuring the user interface is responsive and displays correctly across different screen sizes.
- **Functional Testing**: Verifying that core features, like adding items to the cart, calculating the total cost, and placing the order, function as expected.
- **Bug Fixing**: Identifying and fixing bugs related to app performance, UI glitches, or incorrect behavior during interactions.
- **User Acceptance Testing (UAT)**: Gathering feedback from a small group of target users (coffee shop customers) to ensure the app meets their needs and is easy to use.

5. **Deployment**:

- After testing, the Coffee Shop Mobile Application was deployed on test Android devices for final review. This included:
  - **Local Testing**: Deploying the app on Android emulators and real devices to test performance under different network conditions.
  - **App Store Preparation**: Preparing the app for submission to the Google Play Store (optional) after final approval.
- Deployment is done in stages, first with a limited user base (beta testing), followed by broader deployment after addressing any issues found during the testing phase.

6. **Maintenance**:

- After deployment, the application enters the maintenance phase, which involves:
  - **User Feedback**: Collecting user feedback to identify any issues or areas for improvement.

- **Performance Optimization**: Making adjustments based on feedback, such as improving the loading time of the menu or optimizing the cart management process.
- **Feature Updates**: Adding new features such as payment integration, customer loyalty programs, or offering promotional discounts based on user feedback and market trends.
- **Bug Fixing**: Addressing any reported bugs or issues to ensure the app continues to function smoothly.

5.Conclusion

The Coffee Shop Mobile Application successfully demonstrates an efficient and user-friendly approach to managing coffee orders, ensuring a seamless experience for both customers and business owners. Designed with a clean interface and intuitive navigation, the app allows users to browse the coffee menu, add items to the cart, view their order summary, and place orders with ease.

This project achieves its core objective of providing an integrated platform for users to view and order from a coffee menu, while also managing their orders and tracking their purchase history. It leverages Android's native features such as the shopping cart management system, UI components, and the integration of dynamic elements for a rich, interactive user experience.

The modular structure of the application ensures that it can be easily extended in the future to include additional features like payment integration, order tracking, or loyalty rewards. Moreover, the app's architecture allows for scalability, making it adaptable for further updates or customization to meet specific user needs.

In conclusion, the Coffee Shop Mobile Application serves as a practical tool for enhancing the coffee ordering process, with potential for growth into a more comprehensive system offering greater functionality, user engagement, and business insights.

APPENDICES

SOURCE CODE

```
package com.example.coffeeshop

import android.graphics.Color import
android.os.Bundle import android.view.Gravity
import android.view.View import android.widget.* import
androidx.appcompat.app.AppCompatActivity import
androidx.core.content.ContextCompat import
androidx.core.view.setPadding

class MainActivity : AppCompatActivity() {

private lateinit var frame: FrameLayout private lateinit
var cart: MutableMap<String, Int> private lateinit var
lastOrder: MutableMap<String, Int> private lateinit var
viewCartBtn: Button

data class MenuItem(val name: String, val price: Int, val imageRes: Int)

private val items = listOf(
MenuItem("Espresso", 40, R.drawable.espresso),
MenuItem("Cappuccino", 60, R.drawable.cappuccino),
```

```
MenuItem("Latte", 70, R.drawable.latte),
MenuItem("Mocha", 80, R.drawable.mocha),
MenuItem("Americano", 50, R.drawable.americano),
MenuItem("Black Coffee", 30, R.drawable.black_coffee),
MenuItem("Cold Brew", 90, R.drawable.cold_brew),
        MenuItem("Flat White", 65, R.drawable.flat_white), MenuItem("Biscuits", 20,
        R.drawable.cookies), MenuItem("Cake", 40, R.drawable.cake)    )

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)        cart =
mutableMapOf()        lastOrder = mutableMapOf() frame =
FrameLayout(this).apply {            id =
View.generateViewId()            layoutParams =
FrameLayout.LayoutParams(
FrameLayout.LayoutParams.MATCH_PARENT,
FrameLayout.LayoutParams.MATCH_PARENT
)
}
setContentView(frame)
showLoginPage()
}

private fun showLoginPage() {                    val layout =
LinearLayout(this).apply {                    orientation =
LinearLayout.VERTICAL                gravity = Gravity.CENTER
setPadding(60)
setBackgroundResource(R.drawable.background_image_resized)
}

val username = EditText(this).apply {            hint = getString(R.string.username_hint)
layoutParams =
```

```
LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
16, 0, 16) }
}

val password = EditText(this).apply {            hint
= getString(R.string.password_hint) inputType  =
android.text.InputType.TYPE_CLASS_TEXT    or
android.text.InputType.TYPE_TEXT_VARIATION_PASSWORD
layoutParams                    =                    LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply {              setMargins(0, 16,
 0, 16)
}
}

        val loginBtn = Button(this).apply {
        text = getString(R.string.btn_login)       setBackgroundColor(ContextCompat
        .getColor(this@MainActivity, R.color.purple_500))
        setTextColor(Color.WHITE)
setPadding(20, 20, 20, 20)          layoutParams =
LinearLayout.LayoutParams(              LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
)
setOnClickListener {
if        (username.text.toString().isNotEmpty()        &&        password.text.toString().isNotEmpty())        {
showMenuPage()
```

```
} else {
Toast.makeText( this@MainActivity,
getString(R.string.enter_login_details), Toast.LENGTH_SHORT
).show()
}
}
}

layout.addView(TextView(this).apply { text
= getString(R.string.login_title) textSize =
24f
gravity = Gravity.CENTER
setPadding(0, 0, 0, 32)
})
layout.addView(username) layout.addView(password) layout.addView(loginBtn)

frame.removeAllViews()
frame.addView(layout)
}

private fun showMenuPage() { val layout =
LinearLayout(this).apply { orientation
= LinearLayout.VERTICAL setPadding(30)
setBackgroundResource(R.drawable.background_image_resized)
layoutParams = LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT
)
}
```

```
layout.addView(TextView(this).apply {            text
= getString(R.string.menu_title) textSize
= 24f
gravity          =        Gravity.CENTER
      setPadding(0, 0, 0, 20) })
       val       scrollView     =       ScrollView(this).apply       {
    layoutParams = LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT,
       LinearLayout.LayoutParams.WRAP_CONTENT
     ).apply   {              weight = 1f
       }
}

val innerLayout = LinearLayout(this).apply {            orientation
= LinearLayout.VERTICAL            layoutParams =
LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
)
setPadding(20)
}

for (item in items) { val row = LinearLayout(this).apply {
orientation = LinearLayout.HORIZONTAL
setPadding(20) gravity = Gravity.CENTER_VERTICAL
layoutParams
=                                  LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
```

```
).apply {
setMargins(0, 8, 0, 8)
}
}

val image = ImageView(this).apply {                setImageResource(item.imageRes)
layoutParams          =          LinearLayout.LayoutParams(150,          150).apply     {
setMargins(0, 0, 16, 0)
}
}

val infoLayout = LinearLayout(this).apply {                orientation
= LinearLayout.VERTICAL layoutParams =
LinearLayout.LayoutParams(
0,
LinearLayout.LayoutParams.WRAP_CONTENT,
1f
)
}

val nameText = TextView(this).apply {        text = getString(R.string.item_price, item.name,
      item.price)       textSize
      = 18f setPadding(0,
    0, 0, 8)
    }

    val quantityPicker = NumberPicker(this).apply {
   minValue = 1            maxValue
   = 10 value
   = 1
    layoutParams = LinearLayout.LayoutParams(
```

```
LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT
)
}

val addBtn = Button(this).apply {                text
= getString(R.string.btn_add)
setBackgroundColor(ContextCompat.getColor(this@MainActivity,    R.color.teal_700))
setTextColor(Color.WHITE)              setPadding(16, 8, 16, 8)
layoutParams          =          LinearLayout.LayoutParams(
LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(16,
0, 0, 0)
}
setOnClickListener { val qty
= quantityPicker.value
cart[item.name]       =       cart.getOrDefault(item.name,       0)       +
qty                Toast.makeText(                this@MainActivity,
getString(R.string.added, qty, item.name),
Toast.LENGTH_SHORT
).show()
updateCartButton()
}
}

infoLayout.addView(nameText)
infoLayout.addView(quantityPicker)

row.addView(image)          row.addView(infoLayout) row.addView(addBtn)
```

```
innerLayout.addView(row)
}

scrollView.addView(innerLayout) layout.addView(scrollView)

        val btnLayout = LinearLayout(this).apply { orientation
        = LinearLayout.VERTICAL setPadding(0, 20, 0, 0)
        layoutParams = LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.MATCH_PARENT,
    LinearLayout.LayoutParams.WRAP_CONTENT
        )
        }

        viewCartBtn = Button(this).apply {
updateCartButton()
setBackgroundColor(ContextCompat.getColor(this@MainActivity,            R.color.purple_500))
setTextColor(Color.WHITE)          setPadding(20, 20, 20, 20)          layoutParams =
LinearLayout.LayoutParams(              LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
16, 0, 16)
} setOnClickListener {
if
(cart.isNotEmpty()) { showCartPage()
} else {
Toast.makeText(                    this@MainActivity,
getString(R.string.cart_empty),
Toast.LENGTH_SHORT
).show()
}
```

```
}
}
btnLayout.addView(viewCartBtn)

btnLayout.addView(Button(this).apply {
text = getString(R.string.btn_view_last_order_summary)
setBackgroundColor(ContextCompat.getColor(this@MainActivity,              R.color.teal_700))
setTextColor(Color.WHITE)             setPadding(20, 20, 20, 20)             layoutParams =
LinearLayout.LayoutParams(                LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply {
setMargins(0, 0, 0, 16)
}
setOnClickListener { showLastOrderSummaryPage() }
})

layout.addView(btnLayout)

        frame.removeAllViews() frame.addView(layout)
    }

    private fun updateCartButton() {     if (::viewCartBtn.isInitialized)   {
    viewCartBtn.text = if (cart.isNotEmpty()) {
    getString(R.string.btn_view_cart, cart.values.sum())
    } else {
    getString(R.string.btn_view_cart_default)
}
}
}

private fun showCartPage() {         val layout =
LinearLayout(this).apply {             orientation
```

```
= LinearLayout.VERTICAL setPadding(30)
setBackgroundResource(R.drawable.background_image_resized)
layoutParams = LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT
)
}

layout.addView(TextView(this).apply {
text = getString(R.string.cart_title)
textSize = 24f gravity =
Gravity.CENTER setPadding(0, 0, 0, 20)
})

if (cart.isEmpty()) { layout.addView(TextView(this).apply
{ text
= getString(R.string.cart_empty) textSize
= 18f setPadding(0,
20, 0, 20) gravity =
Gravity.CENTER
})
} else {
val scrollView = ScrollView(this).apply { layoutParams
= LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { weight
= 1f
}
    }
```

```
        val innerLayout = LinearLayout(this).apply {
        orientation = LinearLayout.VERTICAL layoutParams
        = LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
            ) }

        for ((name, qty) in cart) {
        innerLayout.addView(TextView(this).apply {
     text = getString(R.string.cart_item_text, name, qty)
    textSize = 18f
setPadding(0, 10, 0, 10)
})
}

scrollView.addView(innerLayout) layout.addView(scrollView)

layout.addView(Button(this).apply { text =
getString(R.string.btn_order_summary)
setBackgroundColor(ContextCompat.getColor(this@MainActivity, R.color.purple_500))
setTextColor(Color.WHITE) setPadding(20, 20, 20, 20) layoutParams =
LinearLayout.LayoutParams( LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
20, 0, 0)
}
setOnClickListener { showOrderSummaryPage() }
})
}
```

```
layout.addView(Button(this).apply {            text =
getString(R.string.btn_back_to_menu)
setBackgroundColor(ContextCompat.getColor(this@MainActivity,            R.color.teal_700))
setTextColor(Color.WHITE)            setPadding(20, 20, 20, 20)            layoutParams =
LinearLayout.LayoutParams(                LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
20, 0, 0)
}
setOnClickListener { showMenuPage() }
})


frame.removeAllViews()     frame.addView(layout)
    }

    private fun showLastOrderSummaryPage() { val
        layout = LinearLayout(this).apply { orientation
        = LinearLayout.VERTICAL setPadding(30)
            setBackgroundResource(R.drawable.background_image_resized)
        layoutParams = LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.MATCH_PARENT,
        LinearLayout.LayoutParams.MATCH_PARENT
        )    }

layout.addView(TextView(this).apply {            text =
getString(R.string.last_order_summary)            textSize
= 24f
gravity = Gravity.CENTER setPadding(0,
0, 0, 20)
})
```

```
if (lastOrder.isEmpty()) { layout.addView(TextView(this).apply {
text
= getString(R.string.no_past_order) textSize = 18f
setPadding(0, 20, 0, 20)
gravity = Gravity.CENTER
})
} else { var total
= 0
for ((name, qty) in lastOrder) {
val price = items.find { it.name == name }?.price ?: 0 val
subtotal = qty * price total += subtotal
layout.addView(TextView(this).apply {
text = getString(R.string.item_qty_subtotal, name, qty, subtotal)
textSize = 18f
setPadding(0, 10, 0, 10)
}) } layout.addView(TextView(this).apply {
text
= getString(R.string.total_bill, total) textSize =
20f
setPadding(0, 20, 0, 10)
})
}

layout.addView(Button(this).apply { text =
getString(R.string.btn_back_to_menu)
setBackgroundColor(ContextCompat.getColor(this@MainActivity, R.color.teal_700))
setTextColor(Color.WHITE) setPadding(20, 20, 20, 20) layoutParams =
LinearLayout.LayoutParams( LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
20, 0, 0)
```

```
        }
        setOnClickListener { showMenuPage() }
    })

    frame.removeAllViews()
    frame.addView(layout)
  }

private fun showOrderSummaryPage() {
val layout = LinearLayout(this).apply {
orientation = LinearLayout.VERTICAL
setPadding(30)
setBackgroundResource(R.drawable.background_image_resized)
layoutParams = LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT
)
}

layout.addView(TextView(this).apply { text
= getString(R.string.order_summary)
textSize = 24f gravity =
Gravity.CENTER setPadding(0, 0, 0,
20)
})

layout.addView(TextView(this).apply { text =
getString(R.string.table_header_format, getString(R.string.items_header),
getString(R.string.quantity_header), getString(R.string.price_header))
textSize = 18f setPadding(0,
```

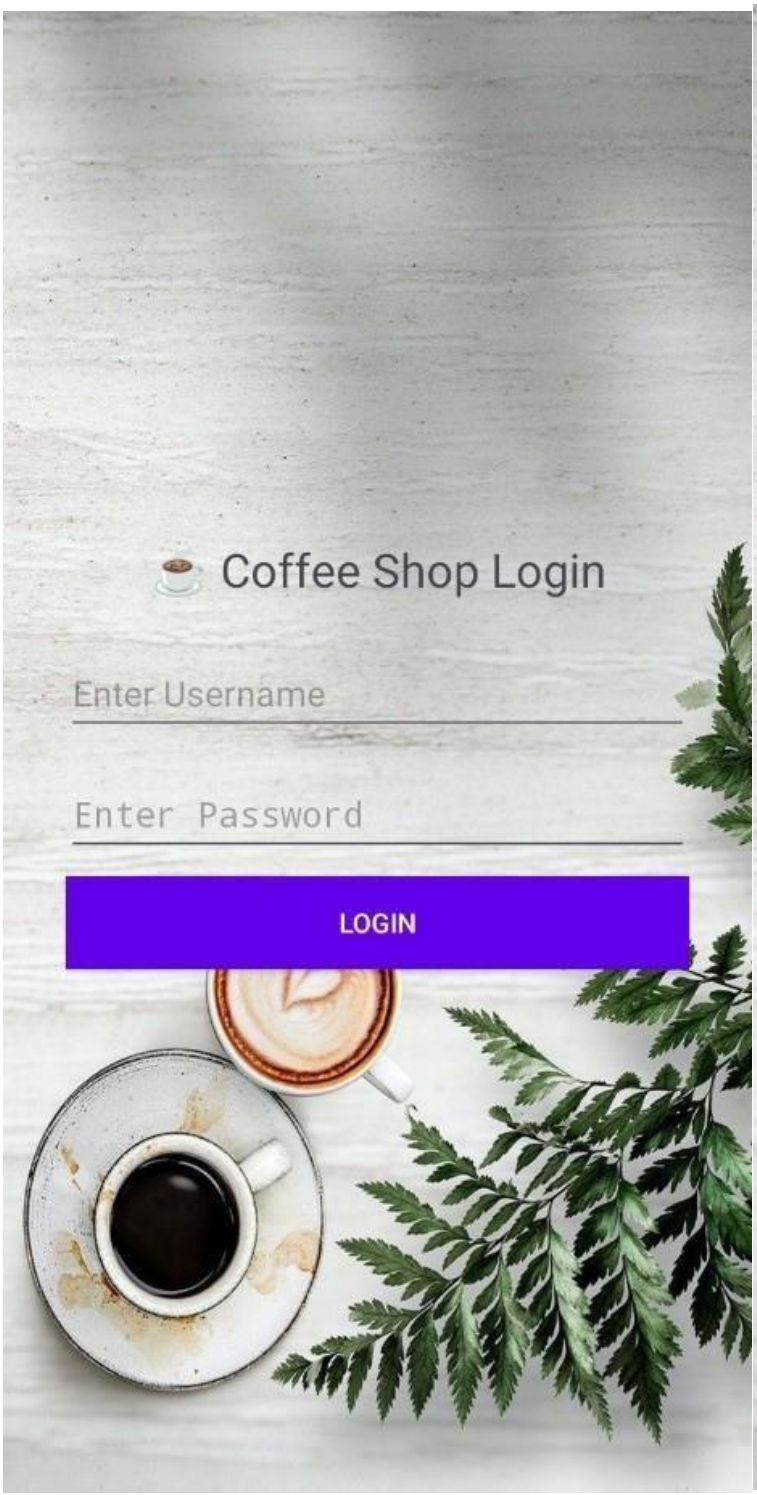
```
20, 0, 10) gravity =
Gravity.CENTER
})

var total = 0        for ((name,
qty) in cart) {
val price = items.find { it.name == name }?.price ?: 0 val
subtotal = qty * price            total
+=        subtotal        layout.addView(TextView(this).apply        {        text        =
getString(R.string.item_qty_subtotal, name, qty, subtotal)                textSize
= 18f setPadding(0, 10,
0, 10)
        })
        }

        layout.addView(TextView(this).apply          {
        text = getString(R.string.total_bill, total) textSize
        = 20f
            setPadding(0, 20, 0, 10)
        })

    layout.addView(Button(this).apply {     text = getString(R.string.btn_place_order)
        setBackgroundColor(ContextCompat.getColor(this@MainActivity, R.color.purple_500))
     setTextColor(Color.WHITE)
setPadding(20, 20, 20, 20)
layoutParams                    =                    LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
20, 0, 0)
}
```

```
setOnClickListener                    {
lastOrder.clear()
lastOrder.putAll(cart)          cart.clear()
updateCartButton()          Toast.makeText(
this@MainActivity,
getString(R.string.order_placed_success),
Toast.LENGTH_SHORT
).show()
showFinalOrderPage()
}
})

frame.removeAllViews()
frame.addView(layout)
}

private fun showFinalOrderPage() {    val
layout =    LinearLayout(this).apply    {        orientation =
LinearLayout.VERTICAL
setPadding(30)
setBackgroundResource(R.drawable.background_image_resized)
layoutParams                =                LinearLayout.LayoutParams(
LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT
)
}

layout.addView(TextView(this).apply {
text  =        getString(R.string.order_placed_success)    textSize
    = 24f    gravity = Gravity.CENTER
```

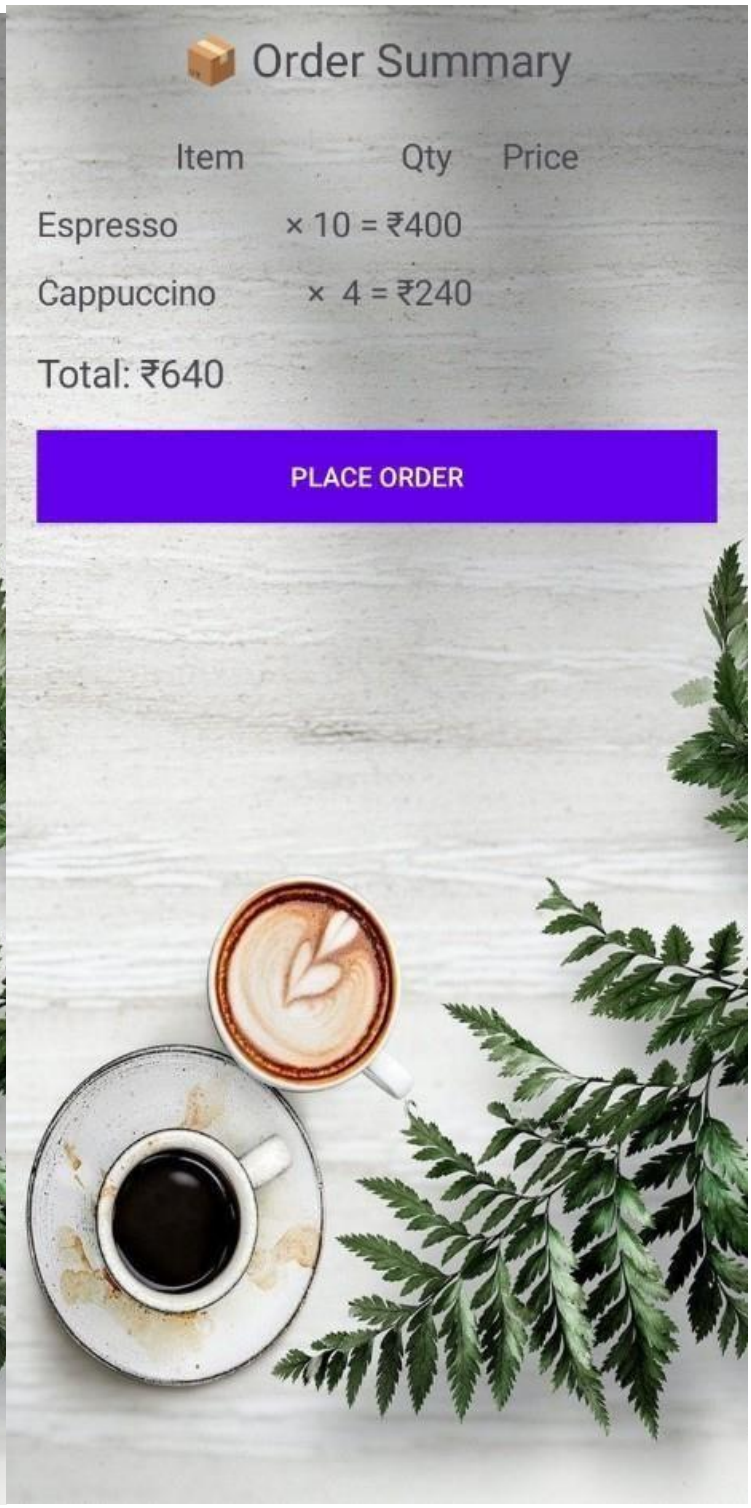
```
        setPadding(0, 0, 0, 20)
        })

        layout.addView(TextView(this).apply {
            text = getString(R.string.order_summary) textSize
            = 20f gravity =
        Gravity.CENTER
setPadding(0, 0, 0, 20)
}) var total = 0        for ((name, qty)

in

lastOrder) {
val price = items.find { it.name == name }?.price ?: 0            val
subtotal = qty * price              total += subtotal
layout.addView(TextView(this).apply { text =
getString(R.string.item_qty_subtotal, name, qty, subtotal)                textSize
= 18f gravity =
Gravity.CENTER
setPadding(0, 10, 0, 10)
})
}

layout.addView(TextView(this).apply       {             text  =
getString(R.string.total_bill,          total)            textSize =
20f
gravity = Gravity.CENTER
setPadding(0, 20, 0, 20)
})

layout.addView(Button(this).apply {             text =
```

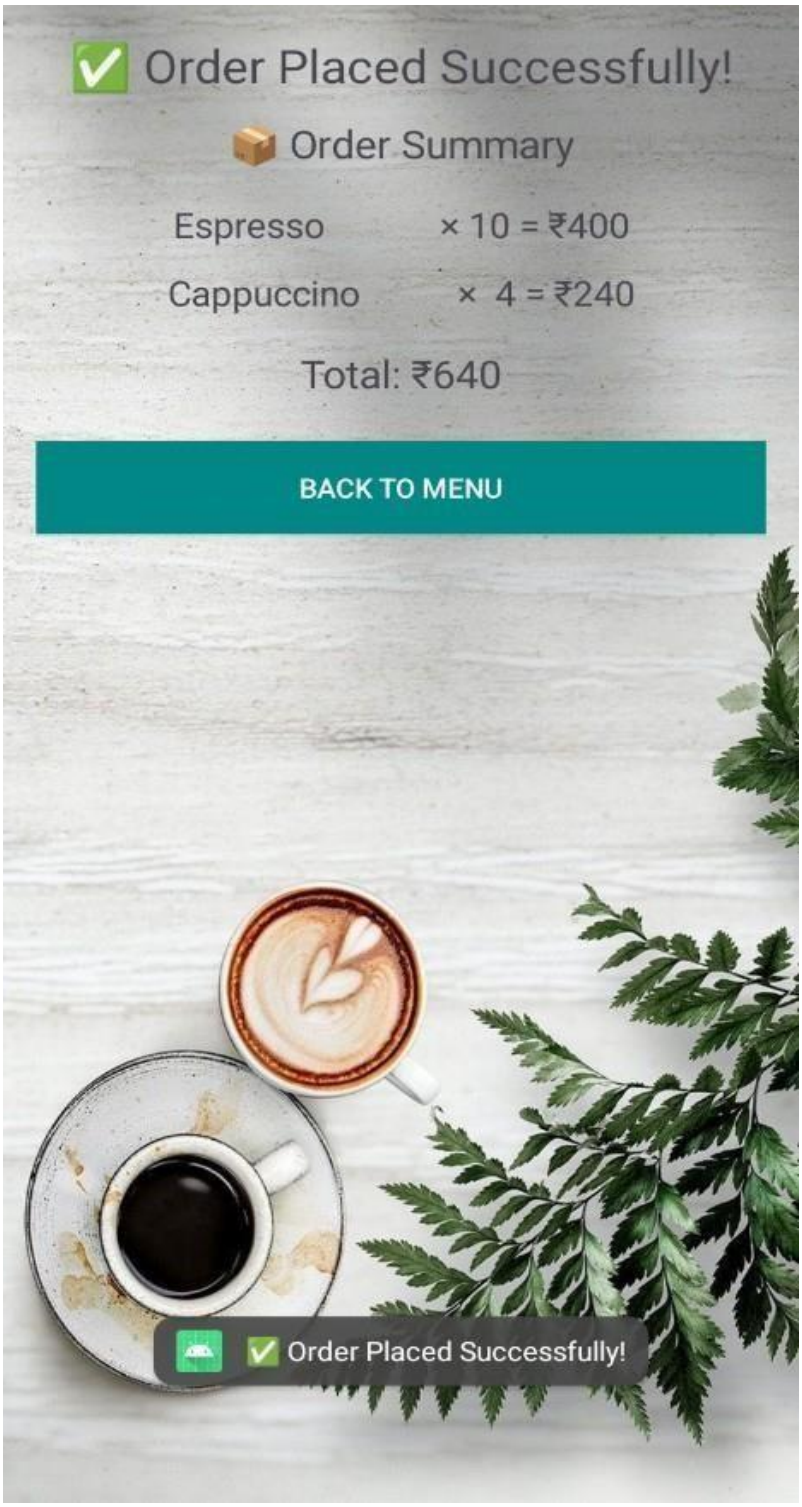
```
getString(R.string.btn_back_to_menu)
setBackgroundColor(ContextCompat.getColor(this@MainActivity,  R.color.teal_700))
setTextColor(Color.WHITE)  setPadding(20, 20, 20, 20)  layoutParams =
LinearLayout.LayoutParams(  LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT
).apply { setMargins(0,
20, 0, 0)
}
setOnClickListener { showMenuPage() }
})

frame.removeAllViews()
frame.addView(layout)
}
}
```

OUTPUT SCREENSHOTS




Your Cart
Espresso (Qty: 10)
Cappuccino (Qty: 4)
PROCEED TO CHECKOUT
BACK TO MENU


Order Summary
Item Qty Price
Espresso × 10 = ₹400
Cappuccino × 4 = ₹240
Total: ₹640
PLACE ORDER


✅ Order Placed Successfully!
📦 Order Summary
Espresso × 10 = ₹400
Cappuccino × 4 = ₹240
Total: ₹640
BACK TO MENU
✅ Order Placed Successfully!

REFERENCES

1. **Android Developer Documentation** – https://developer.android.com

   General reference for Android development, covering core components such as activities, services, and user interface elements.

2. **Kotlin Programming Language Documentation** – https://kotlinlang.org/docs/home.html

   Official documentation for Kotlin, which is used for the development of the Coffee Ordering app.

3. **Android RecyclerView** – https://developer.android.com/reference/android/support/v7/widget/RecyclerView

   Used for displaying lists of coffee items and dynamically updating the UI when items are added to the cart.

4. **Android SQLite Database** – https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase *To manage persistent data storage, such as saving user orders or menu details.*

5. **Android Intents and Broadcasts** – https://developer.android.com/reference/android/content/Intent

   Used for managing navigation and sending user actions (e.g., starting a new activity to view the cart or place an order).

6. **Firebase Documentation** – https://firebase.google.com/docs

   Used for sending email notifications or storing user data and order history in real-time for the app.

7. **Android Button and OnClickListener** – https://developer.android.com/reference/android/widget/Button

   For implementing button actions, such as placing an order or viewing the cart.

8. **Android ViewModel** – https://developer.android.com/topic/libraries/architecture/viewmodel

Used for managing UI-related data lifecycle-consciously, particularly for handling order data.

9. **Android SharedPreferences** – https://developer.android.com/reference/android/content/SharedPreferences *Used for storing simple app preferences, like login status or user settings.*
10. **Stack Overflow** – https://stackoverflow.com

    Frequently used for resolving specific coding issues and getting assistance with Android development challenges.
11. **Google Play Console Documentation** – https://developer.android.com/studio/publish *Guidelines for publishing and distributing your app through the Google Play Store.*