# CHAPTER-1

## 1. Introduction

Café-7 App is a delightful mobile platform designed to satisfy every sweet tooth. Whether you're craving artisanal pastries, decadent coffee, or handcrafted ice creams, this app brings the world of desserts right to your fingertips. Offering seamless browsing, quick ordering, and exclusive deals, Café-7 aims to provide a personalized coffee-shopping experience. With features like custom orders, delivery tracking, loyalty rewards, and curated dessert recommendations, it caters to both casual snackers and gourmet Coffee lovers.

## 1.1 An Overview of the Project

The Café-7 App is a comprehensive digital platform designed to enhance the shopping and delivery experience for customers. It aims to streamline the process of browsing, ordering, and enjoying a variety of coffee, while also empowering the brand with direct control over customer interactions, operations, and promotions. This project addresses the shortcomings of traditional dessert businesses, such as limited ordering channels, dependence on third-party delivery platforms, and inconsistent customer experiences.

## 1.2 Mission of the Project

The **Café-7 App** project is designed to offer a seamless, modern solution for coffee lovers by integrating advanced technology with the joy of indulgence. This platform aims to provide customers with easy access to a wide variety of events through a convenient mobile application, while also streamlining operations for the business.

## Key objectives include:

- Launch loyalty programs that reward repeat customers with points and exclusive discounts.
- Provide personalized recommendations based on past orders and preferences.
- Run seasonal promotions and product campaigns to engage customers regularly.

## 1.3 Background Study

The **Café-7 App** project emerges in response to the evolving landscape of consumer behavior and technological advancements in the food and beverage industry. As customers increasingly prefer the convenience of online ordering and personalized experiences, businesses must adapt to meet these expectations. This section explores the context, challenges, and trends that informed the need for the app.

### 1.3.1 A Study of the Existing System

**Recipe Database:** Yummly offers a vast library of recipes that span various cuisines and dietary preferences, including vegan, gluten-free, and low-carb options. Users can search for desserts specifically or browse through categories like coffees, cookies, and pastries.

**Personalized Recommendations:** The app employs an algorithm that suggests recipes based on users' tastes, dietary restrictions, and previous interactions. This personalization enhances the user experience by providing tailored content.

**Customizable Recipe Search:** Users can filter recipes by ingredients, cooking time, and dietary preferences. This allows for more efficient meal planning and preparation based on what users have on hand.

# CHAPTER-2

## 2. System Analysis

In this section, we will analyze the proposed system for the Café-7 App, focusing on its requirements, functionality, user interface, and overall architecture. This analysis aims to establish a clear understanding of the system's design and operational capabilities, ensuring it meets user needs effectively.

## 2.1 A Study of the Proposed System

In this section, we will conduct a thorough study of the proposed Café-7 App, focusing on its objectives, features, anticipated benefits, and overall impact on users. This analysis aims to demonstrate how the proposed system addresses the gaps identified in existing dessert and recipe apps and outlines its unique value proposition.

- **User Perspective**: Users of the Café-7 App come from diverse backgrounds, ranging from novice bakers to experienced pastry chefs.
- **Admin Perspective**: The admin perspective focuses on the features, responsibilities, and tools necessary for managing the Café-7 App effectively. Admins play a crucial role in ensuring that the platform runs smoothly, that users have a positive experience, and that the app maintains high-quality content.

## 2.2 User Requirement Specification

User Requirement Specification (URS) outlines the needs and expectations of the users regarding the Café-7 App. It serves as a crucial reference point throughout the development process, ensuring that the final product meets user needs and provides a satisfactory experience. This section details the functional and non-functional requirements from the users' perspective.

### 2.2.1 Major Modules

1. **User Authentication Module**: Includes user registration, login, and session management functionalities.
2. **Profile Management:** User profile customization (profile picture, bio) Display of user’s submitted recipes and favorite recipes. View and manage

community interactions (comments, ratings).

3. **Recipe Management Module**: This module focuses on the creation, retrieval, and management of recipes, allowing users to discover, submit, and interact with recipes easily.
4. **Community Engagement Module**: Includes video streaming with volume, brightness controls, and advanced features like gesture support. Personalization Module
5. **Meal Planning and Shopping List Module**: This module assists users in organizing their coffee-making activities and managing their grocery shopping efficiently.
6. **Administration Module:** This module enables admins to manage the app effectively, ensuring content quality, user support, and overall system performance.
7. **User Preferences Module**: Stores user settings like favorite genres and viewing history.

### 2.2.2 Sub Modules

1. **Registration Sub-Module**: Handles user sign-up, including input validation and confirmation emails.
2. **Authentication Sub-Module:** Manages user login and logout processes.
3. **Profile Management Sub-Module:** Allows users to edit their profiles, including updating personal information and changing passwords.
4. **Account Recovery Sub-Module:** Provides an overview of content and user activity, along with CRUD operations for movies and shows.
5. **User Analytics Sub-Module:** Admin can assign genres and providers for each customer.

## 2.3 Software Requirement Specification

- The Café-7 App is a standalone mobile application that will be developed for both iOS and Android platforms. It will utilize cloud-based storage for user data and recipes, ensuring accessibility and scalability.

- **Framework**: React Native
- **Components**: Custom components for UI, Flat List for displaying types of coffee, and for content streaming.

**Backend (Admin Panel & API)**:

- **Platform**: Node.js
- **Database**: MongoDB for storing user data and content information.
- **Middleware**: Express.js for handling API requests.
- **Authentication**: Passport.js for session management and authentication.

## 2.4 System Specification

### 2.4.1 React Native

- **Framework**: React Native powers the admin panel's user interface, allowing dynamic, responsive, and scalable admin tools for managing the platform.
- **Components**: Functional components using hooks, with a focus on modularity and reuse across the application.
- **State Management**: Redux or Context API for managing the application's global state.

### 2.4.2 Node.js

- **Platform**: Node.js acts as the backend for the admin panel and API endpoints. It ensures efficient server-side processing and communication with MongoDB.
- **Performance**: Asynchronous and non-blocking I/O for handling high user traffic efficiently.

### 2.4.3 Express.js

- **Middleware**: Express.js serves as the backbone for API creation, handling all HTTP requests, routing, and connecting to MongoDB for CRUD operations.
- **Endpoints**: Secure endpoints for user authentication, movie management, and fetching data.

### 2.4.4 MongoDB

- **Database**: MongoDB stores data related to users, Admin, and shopping application. It is a NoSQL database, which ensures scalability and flexibility in handling the large volume of data.

- **Schema Design**: The schema design outlines the structure of the database that will support the Café-7 App. It defines the tables, their relationships, and the fields necessary to store user data, recipes, community interactions, and other relevant information.

### 2.4.5 Hardware Configuration

The hardware configuration needed for Café-7 includes a standard development machine with at least:

- Processor: Intel Core i5 or above
- RAM: 8GB or more
- Storage: 500GB SSD or more

### 2.4.6 Software Configuration

- **Operating Systems**: Windows 10/11 for development.
- **Development Tools**:
  - **Android Studio** for building and testing mobile applications.
  - **Visual Studio Code** for code development.
- **Version Control**: Git for code management and collaboration.
- **Package Managers**: npm (Node Package Manager) for handling dependencies such as Express, Mongoose, and React Native libraries.
- **Database**: MongoDB installed locally or remotely via MongoDB Atlas

These specifications ensure that the Dessert's system is capable of delivering performance, scalability, and ease of management for both users and admins.

# CHAPTER-3

## 3. System Design and Development

This section explains the system design concepts and methodologies used in the Café-7 project, including design principles, notations, architecture, and data structures. It focuses on structuring the system for efficiency, scalability, and maintainability.

## 3.1 Fundamentals of Design Concepts

### 3.1.1 Abstraction

- **Definition**: Abstraction involves simplifying complex systems by modeling classes based on essential properties and behaviors while hiding unnecessary details.
- **Application in Café-7**: In Coffee, abstraction is used to create models for users, and genres. The complexity of data interactions is hidden from the user, who interacts with a straightforward interface to manage and view content.

### 3.1.2 Refinement

- **Definition**: Refinement refers to the iterative process of enhancing the design and functionality of the system.
- **Application in Café-7**: Throughout the development of Café-7, feedback is collected from users and administrators, leading to iterative enhancements. This process ensures that features such as the video player, user authentication, and content management evolve based on user experience and needs.

### 3.1.3 Modularity

- **Definition**: Modularity involves dividing the system into smaller, manageable modules that can be developed, tested, and maintained independently.
- **Application in Café-7**: The system is modularized into distinct components, including the admin panel, user authentication, management. Each module can be updated independently without affecting the overall system.

## 3.2 Design Notations

### 3.2.1 System Structure Chart

A system structure chart outlines the hierarchical organization of the different modules within Café-7. It shows how modules such as Admin Panel, User Login, and API interact with each other, emphasizing the dependencies between them.

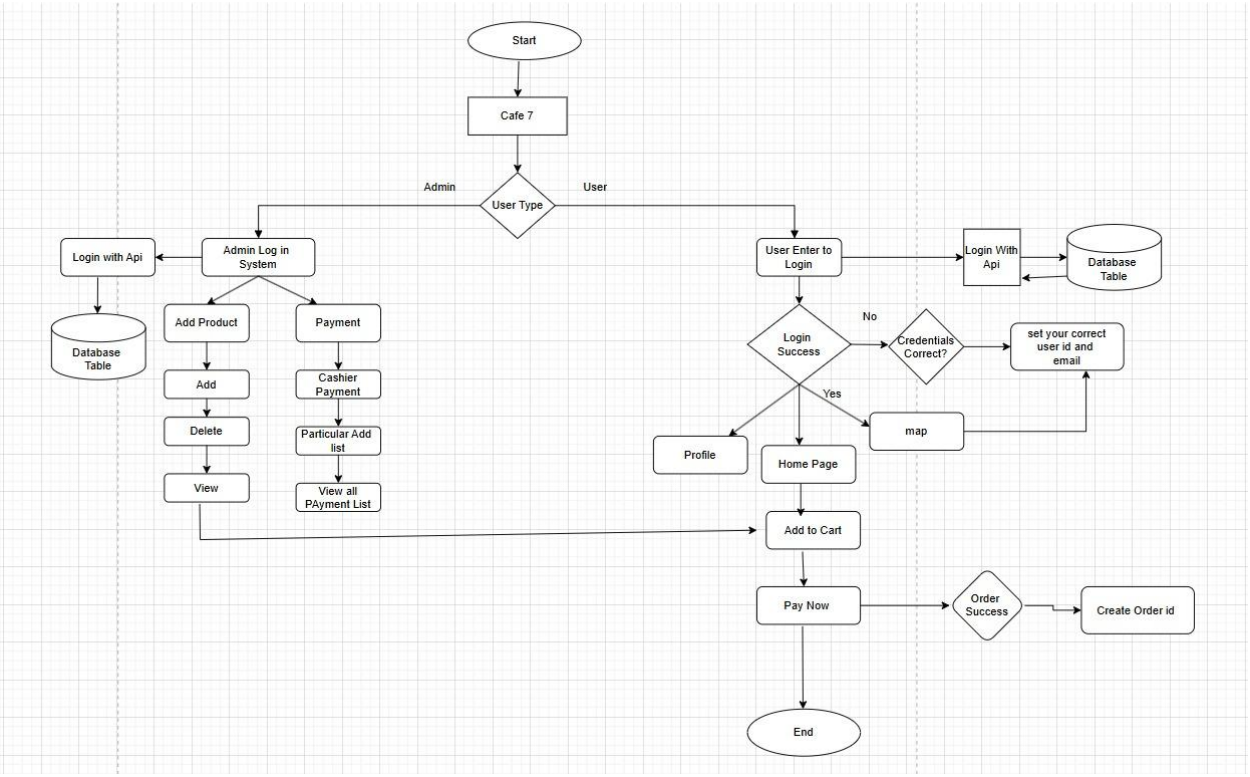



**Figure 3.1 System Structure Chart**

### 3.2.2 System Flow Diagram

A System Flow Diagram (SFD) visually represents the flow of data and control within the Café-7 App. It illustrates how different components of the system interact with one another, providing a clear understanding of the application's processes. Below is a description of the key components and the flow of operations within the app.

### 3.2.3 Data Flow Diagram / UML

- **External Entities**:
  - **User** (Circle): Represents the end-users of the Café-7 app.
  - **Admin** (Circle): Represents the administrators managing the platform via the Admin Panel.
- **Processes**: **User Login & Authentication** (Rounded Rectangle): The process that handles user login and validation.
  - **Ordered Coffee**(Rounded Rectangle): The process that enjoying to the user.
  - **Admin Content Management** (Rounded Rectangle): Admin adds/updates/removes customer details.
- **Data Stores**:
  - **User Data Store** (Open Rectangle): Stores user information, login credentials, and preferences.
  - **Customer Data Store** (Open Rectangle): Stores all coffee and fetched from and local content.
  - **MongoDB** (Open Rectangle): The database that stores everything from user profiles, customer information to watch history.
- **Data Flows**:

  Arrows indicate data flow between the entities and processes:
  - **User submits login credentials** to the **Login & Authentication** process.
  - **Login process accesses User Data Store** for validation.
  - **User browses cakes**, and the **Browse process fetches coffee details from user Data Store**.
  - **User selects a coffee**, and the **coffee process retrieves the media file** from the database.
  - **Admin Panel updates details**, and the **Admin Content Management process updates Data Store**.

### 3.2.4 Software Engineering Model

The software engineering model adopted for Café-7 is the **MVC (Model-View-Controller)** architecture.

- **Model**: Contains the data layer, defining data structures and handling database interactions via MongoDB.
- **View**: Consists of the user interface built with React Native for the mobile app and React.js for the admin panel.
- **Controller**: Acts as the intermediary, processing user requests, interacting with the model, and updating the view as needed.

## 3.3 Design Process

### 3.3.1 Software Architecture

- **Architecture Pattern**: Use a **Microservices Architecture** that separates the application into distinct services for the admin panel and mobile app, allowing for scalability and maintainability.
- **Frontend**: Developed using **React Native**, providing a responsive and user-friendly interface.
- **Backend**: Built with **Node.js** and **Express**, responsible for handling API requests and managing server-side logic.
- **Database**: Utilize **MongoDB** for data storage, ensuring flexibility in data management and retrieval.

### 3.3.2 Control Hierarchy

- **User Level**: Basic users (viewing content, managing their profile).
- **Admin Level**: Admins (content management, user management).
- **System Level**: System processes (handling API requests, data synchronization, etc.).
- **Flow**: The control flow should follow a top-down approach, where requests from users are handled by the admin or system layers.

### 3.3.3 Structural Partitioning

- **Frontend Components**:
  - 
  - **Authentication**: Login and registration forms.
  - **Content Browsing**: Coffee listing, search functionality.
  - **User Management**: Profile settings, favorites list (MyList).

  **Backend Services**:
  - **User Service**: Handles user authentication and profile management.
  - **Customer Service**: Manages Coffee data (CRUD operations).
  - **Admin Service**: Admin functionalities for content management.

### 3.3.4 Data Structure

- **User Schema**:
  - User Id: String (unique identifier)
  - username: String
  - password: String (hashed)
  - email: String
  - favorites: [Customer Id] (Array of movie IDs)
- **Customer Schema**:
  - Customer Id: String (unique identifier)
  - title: String
  - description: String
  - genres: [String] (Array of genre names)
  - provider: String

### 3.3.5 Software Procedure

- **User Registration**:
  1. User fills out the registration form.
  2. Data is validated and hashed.
  3. User is saved to the database.
- **User Login**:
  1. User submits login credentials.
  2. Credentials are validated against the database.
  3. If valid, create a session for the user.

- **Coffee Browsing**:
    1. User accesses the home page.
    2. All available customer is fetched from the database.
    3. Coffee is displayed in a scrollable format.

## 3.4 Database Design

- **Admin:** Contains user and Admin information (id, username, password, email).

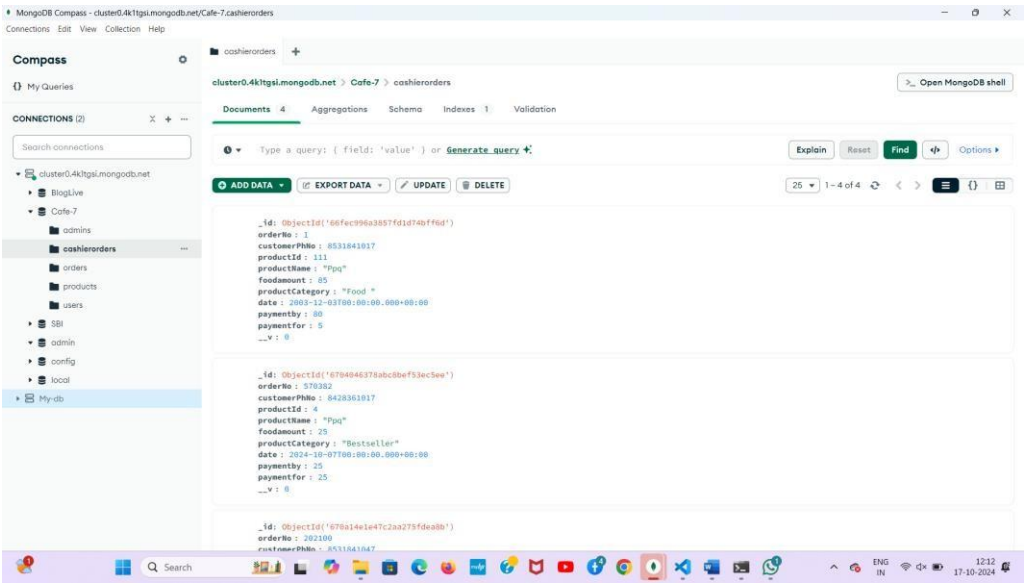



**Figure 3.4 Admin Database**

- **User**

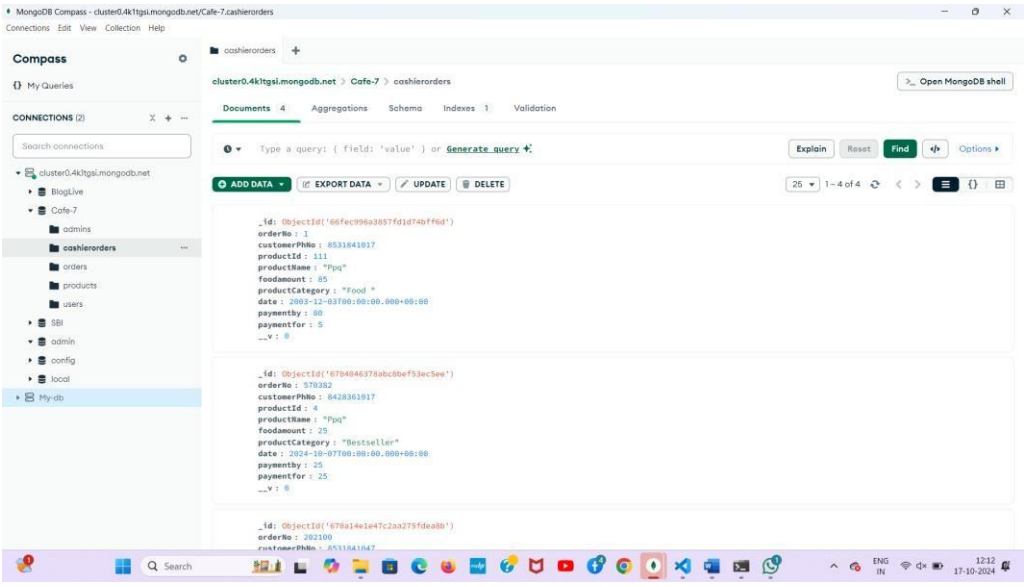



**Figure 3.5 Show User  Database**

## 3.5 Input Design

- **Forms**:
    - Registration and login forms should include fields for username, email, and password.
    - Customer submission forms in the admin panel for adding new items.

### 3.6 Output Design

- **UI Components**: SNavigation menu (Home, Explore, My Recipes, Community, Profile).
- **Dashboard**: Admin dashboard should summarize user and statistics, with charts for visual representation.

## 3.7 Development Approach

- **Agile Methodology**: Use iterative development to allow for flexibility and responsiveness to changes.
- **Version Control**: Implement Git for tracking changes and collaborating with team members.
- **Testing**: Establish a testing framework using Jest for unit testing and Postman for API testing.

# CHAPTER-4

## 4. Testing and Implementation

### 4.1 Testing

Software testing involves the process of evaluating and verifying that a software product or system meets the specified requirements. It includes the act of examining the artifacts and behavior of the software through validation and verification. Testing offers an independent assessment, allowing businesses to understand the risks involved in deploying the software. The primary goal of testing is to identify errors and ensure that the system operates as intended.

Testing is closely intertwined with the implementation process. As the system is implemented, tests ensure that it functions without errors, validating that the implementation was correctly executed. In practice, these two phases work in tandem to deliver a successful product.

Implementation, in this context, refers to executing a plan or process and ensuring that the system runs as expected. Each step of the implementation must be tested, a process known as implementation testing.

### 4.1.1 System Testing

System testing is a comprehensive software testing approach that evaluates the complete and fully integrated system to ensure it meets the specified requirements. This type of testing occurs after integration testing and before acceptance testing, confirming that the system is suitable for release to end-users.

Key characteristics of system testing include:

- **Scope**: It tests both functional and non-functional aspects of the software.
- **Objective**: To detect defects in integrated units and the overall system.
- **Process**: Takes input from passed integration tests and checks for irregularities across the system components.
- **Method**: System testing is usually a black-box testing method, where the tester does not need to know the internal workings of the application. It focuses on validating the system against the Software Requirement Specifications (SRS) and functional requirement specifications.

System testing ensures that the entire software system operates as designed. The goal is to ensure that the system performs tasks as required, aligning with the expectations of both developers and end-users. This stage of testing is typically performed by an independent testing team to maintain objectivity and impartiality in evaluating system quality.

### 4.1.2 White Box Testing

White box testing, also known as clear box or glass box testing, is a method where the internal structure of the software is known to the tester. Unlike black box testing, it focuses on validating internal processes and logic by examining the codebase.

- **Goal:** To test the internal workings of the software, including paths, conditions, loops, and logic.
- **Scope:** Requires the tester to have programming skills and knowledge of the internal code to test the functionality accurately.

### 4.1.3 Unit Testing

- **Definition**: Unit testing involves testing individual components or modules of the application to ensure they perform as expected.
- **Objectives**:
  - Isolate each part of the program and show that the individual parts are correct.
  - Facilitate easier debugging and code refactoring.
- **Approach**:
  - Write test cases for every function, method, or class to validate their correctness.
  - Utilize mocks and stubs for dependent components to focus on the unit being tested.

## 4.2 System Implementation

System implementation refers to executing a well-formulated plan for deploying a system or process. The success of implementation hinges on clear objectives, predefined actions, and thorough testing to ensure the system operates correctly. Each action in the implementation process undergoes testing, known as implementation testing, to confirm that the system performs as intended.

## Key Aspects of Implementation:

1. **Testing Technological Specifications**: It is critical to verify the implementation of specific technological requirements to ensure they align with user and system expectations.
2. **Improving Interfaces**: Through testing, implementation can improve user interfaces to make them more understandable and user-friendly for developers and users alike.
3. **Action Confirmation**: Implementation testing ensures that identified actions can be successfully executed without issues.
4. **Identifying Issues**: Ambiguous or conflicting actions are easily identified during implementation testing, allowing for quick resolution.
5. **Quality Control**: Testing during implementation acts as a quality check, ensuring that the system's features are correctly implemented and operational.

During this phase, the test manager collaborates with the IT infrastructure team to ensure the availability of the appropriate test environment and its configuration. Implementation also involves the realization of system components, such as:

- **System Classes**: The blueprint for how the system behaves.
- **User Interface**: Ensuring users interact with the system intuitively and effectively.
- **Database Structures**: Ensuring the data storage and retrieval system operates as designed.

### 4.2.1 System Maintenance

System maintenance involves ensuring the system remains operational and efficient over time. Maintenance includes two primary activities:

1. **System Maintenance**: The process of restoring the system to its original functionality by fixing bugs, improving performance, and maintaining operational consistency.
2. **System Enhancement**: The addition or modification of features based on evolving user needs or updated specifications. Enhancement expands the system's capabilities, making it adaptable to new requirements.

## Types of Maintenance:

- **Corrective Maintenance**: Fixing errors found in the system after it has been deployed.
- **Adaptive Maintenance**: Modifying the system to cope with changes in the environment (e.g., new hardware or software platforms).

- **Perfective Maintenance**: Improving system performance or maintainability, even when no defects are present.
- **Preventive Maintenance**: Identifying and correcting potential issues before they occur, ensuring the system remains stable over time.

## Importance of System Maintenance:

- Keeps the system functioning at an optimal level by addressing potential faults before they impact operations.
- Ensures system security and compliance with current technological standards.
- Enhances the system to meet new user requirements, extending its lifecycle and value.
- **Documentation**: Maintain comprehensive documentation that includes installation procedures, user guides, and troubleshooting instructions to assist users and future developers.

For businesses, investing in proper maintenance results in long-term cost savings. Regular updates and fixes prevent the need for expensive overhauls or replacements, keeping the system efficient and relevant.

### User Test Cases

| Test Case ID | Description | Test Scenario | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_U_001 | User Registration | Verify that a new user can register | 1. Open the app<br>2. Navigate to the Signup page<br>3. Enter valid details<br>4. Click "Sign Up" | User account should be created successfully |
| TC_U_002 | User Login | Verify user can log in with valid credentials | 1. Enter email & password<br>2. Click "Login" | User is redirected to the home screen |
| TC_U_003 | View Menu | Verify user can view café menu | 1. Login<br>2. Navigate to the Menu section | Menu items should be displayed |
| TC_U_004 | Add to Cart | Verify user can add items to the cart | 1. Select an item<br>2. Click "Add to Cart" | Item should be added successfully |
| TC_U_005 | Remove from Cart | Verify user can remove items from the cart | 1. Open cart<br>2. Click "Remove" | Item should be removed |
| TC_U_006 | Checkout | Verify user can complete checkout process | 1. Add items to the cart<br>2. Proceed to checkout<br>3. Select payment<br>4. Confirm order | Order should be placed successfully |
| TC_U_007 | Order Tracking | Verify user can track orders | 1. Place an order<br>2. Open "Order Tracking" | Real-time tracking updates should be visible |
| TC_U_008 | Payment Processing | Verify successful payment | 1. Select payment method<br>2. Enter details<br>3. Confirm payment | Payment should be processed |
| TC_U_009 | Update Profile | Verify user can update profile details | 1. Navigate to Profile<br>2. Edit details<br>3. Click "Save" | Profile should be updated |
| TC_U_010 | Logout | Verify user can log out | 1. Click Profile<br>2. Select "Logout" | User is logged out |

### Admin Test Cases

| Test Case ID | Description | Test Scenario | Test Steps | Expected Result |
|---|---|---|---|---|
| TC_A_001 | Admin Login | Verify admin can log in | 1. Enter admin credentials<br>2. Click "Login" | Admin dashboard should load |
| TC_A_002 | Add Product | Verify admin can add a new menu item | 1. Navigate to "Manage Menu"<br>2. Click "Add Product"<br>3. Enter item details<br>4. Click "Save" | Product should be added to the menu |
| TC_A_003 | Edit Product | Verify admin can edit an existing menu item | 1. Navigate to "Manage Menu"<br>2. Select item<br>3. Modify details<br>4. Save changes | Product details should be updated |
| TC_A_004 | Delete Product | Verify admin can delete a menu item | 1. Go to "Manage Menu"<br>2. Select item<br>3. Click "Delete" | Item should be removed |
| TC_A_005 | View Orders | Verify admin can see all orders | 1. Navigate to "Orders"<br>2. View pending/completed orders | Order list should be displayed |
| TC_A_006 | Update Order Status | Verify admin can change order status | 1. Select an order<br>2. Change status (e.g., "Preparing", "Completed") | Order status should be updated |
| TC_A_007 | Manage Users | Verify admin can view and manage users | 1. Go to "User Management"<br>2. View user list<br>3. Edit or remove user if needed | User details should be manageable |
| TC_A_008 | View Revenue Reports | Verify admin can check daily/monthly revenue | 1. Navigate to "Reports"<br>2. Select date range<br>3. View report | Revenue data should be displayed |
| TC_A_009 | Admin Logout | Verify admin can log out | 1. Click Profile<br>2. Select "Logout" | Admin is logged out successfully |

# CHAPTER-5

## 5. Conclusion

The Café-7 App aims to provide a comprehensive platform for coffee enthusiasts, enabling users to explore, create, and share Coffee recipes while fostering a vibrant community of food lovers. Throughout the design and development process, several key elements have been addressed to ensure the app meets the needs of its users.

### Directions for Future Enhancements:

For future enhancements to These may include:

- **Advanced Personalization:** AI-Driven Recommendations: Implement machine learning algorithms that analyze user behavior, preferences, and interactions to offer highly personalized recipe suggestions and content.
- **Augmented Reality (AR) Features:** Interactive Cooking Guides: Develop AR features that overlay step-by-step cooking instructions directly onto users' kitchen environments, helping them visualize the process and improve their cooking skills.
- **Recipe Collaboration**: Allow users to collaborate on recipes with friends or family, enabling shared edits and discussions.
- **Multilingual Support Language Options**: Offer the app in multiple languages to cater to a global audience, making it accessible to non-English speakers and expanding the user base.

# CHAPTER-6
# BIBLIOGRAPHY

## 6. BIBLIGRAPHY:

### 6.1 APP REFERENCES:

**Official Documentation:**

- React Native Documentation – Essential for developing the Café7 frontend.
- Node.js Documentation – Covers backend development using Node.js.
- Express.js Documentation – Reference for building backend APIs in Café7.
- MongoDB Documentation – Provides insights into database management for Café7.
- Postman Documentation – Guides API testing and debugging with Postman.
- npm Documentation – Covers package management for JavaScript and Node.js applications.
- Wikipedia – Useful for diagrams and various software testing methodologies.
- UI/UX Design Principles (NNGroup) – Best practices for improving user experience in Café7.
- Material Design Guidelines – A design system for creating a visually appealing UI in Café7.

### 6.2 Book References for Café7


- Eisenman, B. (2016). Learning React. O'Reilly Media.
    - Covers React and React Native, essential for building the Café7 frontend
- Pasquali, S. (2013). Mastering Node.js. Packet Publishing.
    - Explains Node.js concepts, useful for the backend of Café7.
- Banker, K., Bakkum, P., Verch, S., Garrett, D., & Hawkins, T. (2016). MongoDB in Action (2nd ed.). Manning Publications.
    - Provides insights into MongoDB, the database used in Café7.
- Myers, G. J., Sandler, C., & Badgett, T. (2012). Software Testing (3rd ed.). Wiley.
    - Helps in understanding software testing strategies, relevant for testing Café7.
- Brown, J. (2014). Web Development with Node and Express. O'Reilly Media.
    - Covers Express.js fundamentals, essential for building the Café7 backend.
- Logan, J. (2020). Exploring Postman for API Testing. Apress.
    - Guides API testing using Postman, useful for debugging and validating Café7's APIs

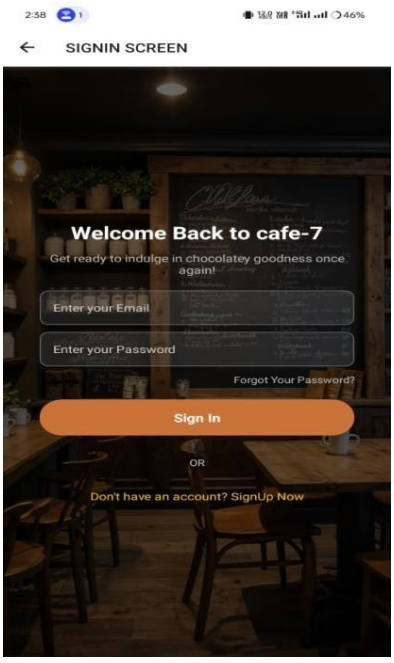
# ANNEXURE

## ANNEXURE –A-OUTPUT DESIGN

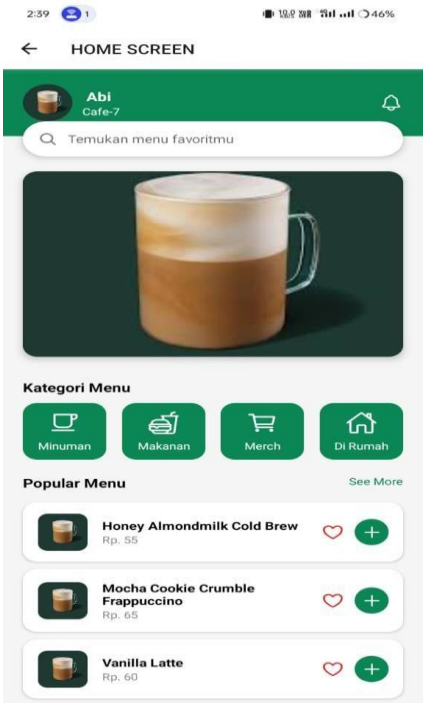
## REACT NATIVE APP:-

## REGISTER

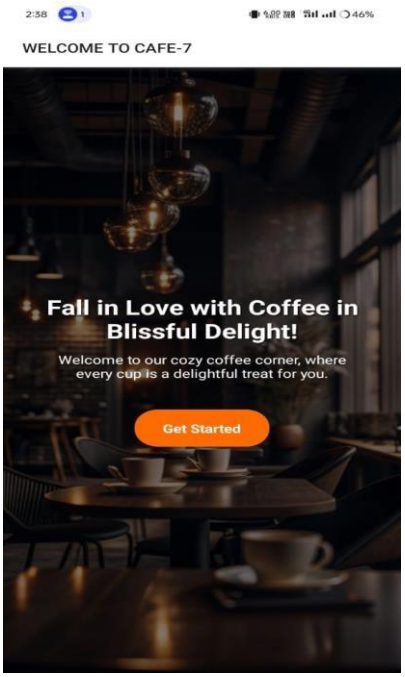

## LOGIN



## HOME

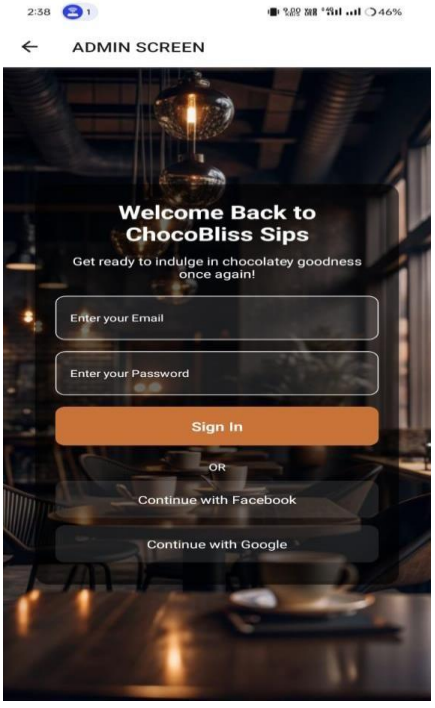



## WELCOME

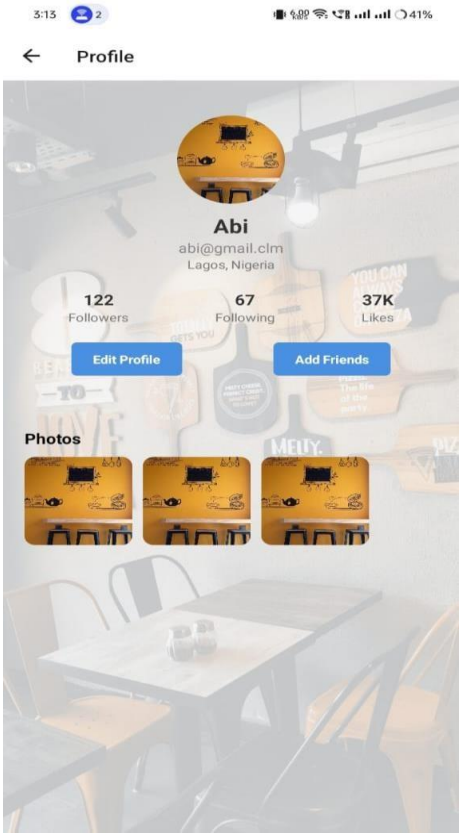


## VIEWCART


## ADMINSCREEN



**PROFILE**


3:13
41%
Profile
Abi
abi@gmail.clm
Lagos, Nigeria
122
Followers
67
Following
37K
Likes
Edit Profile
Add Friends
Photos

# ANNEXURE –B-SOURCE CODE:

## REACT NATIVE APP

**Signup Page :**

```
import React, { useState } from 'react';
import { View, Text, TextInput, TouchableOpacity, StyleSheet, Alert } from 'react-native';
import axios from 'axios';
import { useNavigation } from '@react-navigation/native';
export default function SignUpScreen() {
 const [email, setEmail] = useState('');
 const [password, setPassword] = useState('');
 const [confirmPassword, setConfirmPassword] = useState('');
 const navigation = useNavigation();
 const handleSignUp = async () => {
 if (!email || !password || !confirmPassword) {
      Alert.alert('Validation Error', 'Please fill in all fields.');
      return;
    }
 if (password !== confirmPassword) {
      Alert.alert('Validation Error', 'Passwords do not match.');
      return;
    }
    try {
      await axios.post('http://192.168.209.14:1503/signup', { email, password });
      Alert.alert('Sign Up', 'Sign up successful! Please sign in.');
      navigation.navigate('SignInScreen');
    } catch (err) {
  console.error(err);
  const errorMessage = err.response?.data?.error || 'Something went wrong';
  Alert.alert('Error', errorMessage);
    }
  };
return ( <View style={styles.container}>
```

```
      <Text style={styles.title}>Sign Up</Text>
      <TextInput
        style={styles.input}
        placeholder="Email"
        value={email}
        onChangeText={setEmail}
        keyboardType="email-address"
        autoCapitalize="none"
      <TextInput
        style={styles.input}
        placeholder="Password"
        value={password}
        onChangeText={setPassword}
        secureTextEntry
      />
      <TextInput
        style={styles.input}
        placeholder="Confirm Password"
        value={confirmPassword}
        onChangeText={setConfirmPassword}
        secureTextEntry
      />
      <TouchableOpacity style={styles.button} onPress={handleSignUp}>
        <Text style={styles.buttonText}>Sign Up</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={() => navigation.navigate('SignInScreen')}>
        <Text style={styles.linkText}>Already have an account? Sign In</Text>
      </TouchableOpacity>
    </View>
```

**Login Page :**

```
import React from 'react';
```

```
import { View, Text, Button, StyleSheet, Image, SafeAreaView } from 'reacconst LogoScreen = ({ navigation }) => {return (const navigateToCarts = () => {
    navigation.navigate('CartScreen'); // Replace 'AllCakes' with the actual name of the screen you want to navigate to.
  };
  return (
    <ScrollView style={{ flex: 1, padding: 16, backgroundColor: '#fff' }}>
      {/* Header */}
       <View style={{ flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center' }}>
        <TouchableOpacity>
         <Image source={require('../../assets/img3.png')} style={{ width: 24, height: 24 }} />
        </TouchableOpacity>
        <TouchableOpacity onPress={navigateToCarts}>
         <Image source={require('../../assets/img4.png')} style={{ width: 24, height: 24 }} />
        </TouchableOpacity>
      </View>
      {/* Location & Title */}
      {/* <Text style={{ marginTop: 16, fontSize: 16, color: 'gray' }}>California, USA</Text> */}
     <Text style={{ fontSize: 24, fontWeight: 'bold', marginVertical: 8 }}>What would you like to eat?</Text> SafeAreaView style={styles.container}>
      <View style={styles.logoContainer}>
        {/* Replace with your logo image */}
       <Text style={styles.title}>Dreamy Dessert's</Text>
      </View>
```

```
    <SafeAreaView style={styles.container}>
      <View style={styles.logoContainer}>
        {/* Replace with your logo image */}
       <Text style={styles.title}>Dreamy Dessert's</Text>
      </View>
      <View style={styles.buttonContainer}>
        <Button
          title="Manager"
          onPress={() => navigation.navigate('AdminScreen')} // Define navigation for admin
          color="purple" // Optional: Customize the button color
        <Button
          title="Customer"
          onPress={() => navigation.navigate('SignUpScreen')} // Define navigation for user
          color="purple" // Optional: Customize the button color
        />
      </View>
    </SafeAreaView>
  );
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
```

**Home_Page :**

```
import React from 'react';
import { View, Text, Image, FlatList, TextInput, TouchableOpacity, ScrollView, StyleSheet } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import LottieView from 'lottie-react-native';
const dataCategories = [
```

```
  { id: '1', name: 'Cake', icon: require('../../assets/icon1.png') },
  { id: '2', name: 'Pastry', icon: require('../../assets/icon2.jpeg') },
  { id: '3', name: 'Cup Cake', icon: require('../../assets/icon3.png') },
  { id: '4', name: 'Donuts', icon: require('../../assets/icon4.jpeg') },
  { id: '5', name: 'Biscuits', icon: require('../../assets/icon5.png') },
  { id: '6', name: 'Biscuits', icon: require('../../assets/icon5.png') },
];const popularCakes = [
   { id: '1', name: 'Special Wedding Cake', price: '$185.60', flavor: 'Creamy', image: require('../../assets/img1.jpeg') },
   { id: '2', name: 'Birthday Cake', price: '$250.90', flavor: 'Strawberry', image: require('../../assets/download.jpeg') },
   { id: '3', name: 'Birthday Cake', price: '$250.90', flavor: 'Strawberry', image: require('../../assets/download.jpeg') },
   { id: '4', name: 'Birthday Cake', price: '$250.90', flavor: 'Strawberry', image: require('../../assets/download.jpeg') },
   { id: '5', name: 'Birthday Cake', price: '$250.90', flavor: 'Strawberry', image: require('../../assets/download.jpeg') },
  // Add more cakes if necessary
];const HomeScreen = () => {
  const navigation = useNavigation();
  const navigateToScreen = (categoryId) => {
    if (categoryId === '1') {
    navigation.navigate('Bestseller');
    } else if (categoryId === '2') {
      navigation.navigate('Allcakes');
    } else if (categoryId === '3') {
      navigation.navigate('CupCakeScreen');
    } else if (categoryId === '4') {
      navigation.navigate('DonutScreen');
    } else if (categoryId === '5') {
```

```
    navigation.navigate('BiscuitScreen');
  }
 };const navigateToSeeAll = () => {
  navigation.navigate('Seeall'); // Replace 'AllCakes' with the actual name of the screen you want to navigate to.
 };
 const navigateToCarts = () => {
  navigation.navigate('CartScreen'); // Replace 'AllCakes' with the actual name of the screen you want to navigate to.
 };
 return (
  <ScrollView style={{ flex: 1, padding: 16, backgroundColor: '#fff' }}>
   {/* Header */}
    <View style={{ flexDirection: 'row', justifyContent: 'space-between', alignItems: 'center' }}>
     <TouchableOpacity>
      <Image source={require('../../assets/img3.png')} style={{ width: 24, height: 24 }} />
     </TouchableOpacity>
     <TouchableOpacity onPress={navigateToCarts}>
      <Image source={require('../../assets/img4.png')} style={{ width: 24, height: 24 }} />
     </TouchableOpacity>
    </View>
    {/* Location & Title */}
    {/* <Text style={{ marginTop: 16, fontSize: 16, color: 'gray' }}>California, USA</Text> */}
   <Text style={{ fontSize: 24, fontWeight: 'bold', marginVertical: 8 }}>What would you like to eat?</Text>
{/* <TouchableOpacity>
     <Text style={{ color: 'orange', fontWeight: 'bold' }}>View Menu →</Text>
    </TouchableOpacity> */}
```

```
{/* Search Bar */}
<View style={{ flexDirection: 'row', marginTop: 16, backgroundColor: '#f5f5f5', padding: 8, borderRadius: 8 }}>
  <TextInput placeholder="Search here..." style={{ flex: 1 }} />
  <TouchableOpacity >
    <Image source={require('../../assets/img5.png')} style={{ width: 24, height: 24 }} />
  </TouchableOpacity>
</View>
{/* Category Section */}
<View style={{ marginTop: 24 }}>
  <View style={{ flexDirection: 'row', justifyContent: 'space-between' }}>
    <Text style={{ fontSize: 18, fontWeight: 'bold' }}>Discover by category</Text>
    <TouchableOpacity>
      <Text style={{ color: 'orange' }}>See All →</Text>
    </TouchableOpacity>
  </View>
  <FlatList
    data={dataCategories}
    horizontal
    keyExtractor={(item) => item.id}
    renderItem={({ item }) => (
      <TouchableOpacity onPress={() => navigateToScreen(item.id)}>
        <View style={{ alignItems: 'center', marginRight: 16 }}>
          <Image source={item.icon} style={{ width: 50, height: 50, borderRadius: 25, marginBottom: 8 }} />
          <Text>{item.name}</Text>
        </View>
      </TouchableOpacity>
    showsHorizontalScrollIndicator={false}
    style={{ marginVertical: 16 }}
```

```
      />
    </View>
        {/* Animation and Title */}
        <View style={styles.container}>
      <LottieView
      source={require('../../assets/icon.json')} // Replace with your own Lottie JSON animation
       autoPlay
       loop
       style={{ width: 1000, height: 250 }}
      />
      <Text style={styles.title}>Cake Screen</Text>
      <Text style={styles.description}>Here are some delicious cakes!</Text>
    </View>
    {/* Popular Cakes Section */}
    <View style={{ marginTop: 16 }}>
      <View style={{ flexDirection: 'row', justifyContent: 'space-between' }}>
        <Text style={{ fontSize: 18, fontWeight: 'bold' }}>Popular Cakes</Text>
        <TouchableOpacity onPress={navigateToSeeAll} >
          <Text style={{ color: 'orange' }} >See All →</Text>
        </TouchableOpacity>
      </View>
      <FlatList
        data={popularCakes}
        horizontal
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
              <View style={{ width: 150, marginRight: 16, backgroundColor: '#f5f5f5',
borderRadius: 8, padding: 8 }
```

**Bestseller.js**

```
import React, { useState } from 'react';
import { View, Text, StyleSheet, Image, TouchableOpacity, ScrollView, ImageBackground }
from 'react-native';
import  { useContext } from 'react';
import { CartContext } from './cartContext';
const ModelPage = ({ navigation  , route}) => {
const { cart, setCart } = useContext(CartContext);
  const [modelItems, setModelItems] = useState([
    { id: '1', name: 'Roses with Teddy', price: 5000, image: require('../../assets/co1.jpeg') },
    { id: '2', name: 'Redwelvet', price: 7000, image: require('../../assets/co2.jpeg') },
    { id: '3', name: 'whiteforest', price: 6000, image: require('../../assets/co3.jpeg') },
    { id: '4', name: 'Cupcakes', price: 4000, image: require('../../assets/co4.jpeg') },
    { id: '5', name: 'Cupcakes', price: 4000, image: require('../../assets/co5.jpeg') },
  ]);  // const [cart, setCart] = useState([]);  const addToCart = (item) => {
    setCart((prevCart) => [...prevCart, item]);
  };
  return (
    <ImageBackground
      source={require('../../assets/ba.jpeg')}
      style={styles.backgroundImage}
      <View style={styles.container}>
        <Text style={styles.title}>Cakes</Text>
        <ScrollView>
          {modelItems.map((item) => (
            <View key={item.id} style={styles.modelItem}>
              <Image source={item.image} style={styles.modelItemImage} />
              <Text style={styles.modelItemName}>{item.name}</Text>
<Text style={styles.modelItemPrice}>Rs.{item.price.toFixed(2)}</Text>
              <TouchableOpacity
```

```
              style={styles.addToCartButton}
onPress={() => addToCart(item)}
            <Text style={styles.addToCartButtonText}>Add to Cart</Text>
             </TouchableOpacity>
           </View>
         ))}
       </ScrollView>
       <TouchableOpacity
         style={styles.viewCartButton}
         onPress={() => navigation.navigate('CartScreen')}
       >
         <Text style={styles.viewCartButtonText}>View Cart</Text>
       </TouchableOpacity>
     </View>
   </ImageBackground>
 );
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 40,
    backgroundColor: 'rgba(255, 255, 255, 0)',
  },
  backgroundImage: {
    flex: 1,
    resizeMode: 'cover',
    width: '100%',
    height: '100% justifyContent: 'center',
```

**AllCakes.js**

```
import {View, Text, StyleSheet, Image, TouchableOpacity, ScrollView, ImageBackground }
from 'react-native';
const ModelPage = ({navigation}) => {
  const [modelItems, setModelItems] = useState([
    { id: '1', name: 'Roses with Teddy', price: 5000, image: require('../../assets/he.jpeg') },
    { id: '2', name: 'Redwelvet', price: 7000, image: require('../../assets/co2.jpeg') },
    { id: '3', name: 'whiteforest', price: 6000, image: require('../../assets/hon.jpg') },
    { id: '4', name: 'Cupcakes', price: 4000, image: require('../../assets/h.jpeg') },
    { id: '5', name: 'Cupcakes', price: 4000, image: require('../../assets/hot.jpeg') },
  ]);
  const [cart, setCart] = useState([]);const addToCart = (item) => {
    setCart((prevCart) => [...prevCart, item]);
  };return (
    <ImageBackground
      source={require('../../assets/ba.jpeg')}
      style={styles.backgroundImage}
      <View style={styles.container}>
        <Text style={styles.title}>all cales</Text>
        <ScrollView>
          {modelItems.map((item) => (
            <View key={item.id} style={styles.modelItem}>
              <Image source={item.image} style={styles.modelItemImage} />
              <Text style={styles.modelItemName}>{item.name}</Text>
              <Text style={styles.modelItemPrice}>Rs.{item.price.toFixed(2)}</Text>
```

**Profile.js**

```
import { View, Text, StyleSheet, Image, TouchableOpacity, ScrollView, ActivityIndicator,
ImageBackground } from 'react-native';
import React, { useEffect, useContext, useState } from 'react';
import axios from 'axios';
```

```
import { CartContext } from './cartContext'
export default function Profile() {
   const { userId } = useContext(CartContext);
   const [data, setData] = useState({});
   const [loading, setLoading] = useState(true); // Track loading state
   const fetchUser = async () => {
      try {
       const response = await axios.get(`http://192.168.106.103:1503/user-details/${userId}`);
         setData(response.data.user);
         setLoading(false); // Stop loading when data is fetched
      } catch (error) {
         console.log(error);
         setLoading(false); // Stop loading if there's an error
      }
   };
   useEffect(() => {
      fetchUser();
   }, []);
   if (loading) {
      return (
         <View style={styles.loaderContainer}>
            <ActivityIndicator size="large" color="#0000ff" />
         </View>
      );
   }
   return (
      <ImageBackground
                source={require('../../assets/si2.jpeg')} // Set your background image here
style={styles.backgroundImage}
   <ScrollView contentContainerStyle={styles.container}>
            <View style={styles.headerContainer}>
               <Image
                  source={require('../../assets/prof.png')} // You can replace with user's profile
                  style={styles.profileImage}
```

```
    />
    <Text style={styles.name}>{data?.name}Mangai</Text>
    <Text style={styles.profession}>{data?.email}</Text>
    <Text style={styles.location}>Lagos, Nigeria</Text>
    <View style={styles.statsContainer}>
      <View style={styles.statBox}>
        <Text style={styles.statNumber}>122</Text>
        <Text style={styles.statLabel}>Followers</Text>
      </View>
      <View style={styles.statBox}>
        <Text style={styles.statNumber}>67</Text>
        <Text style={styles.statLabel}>Following</Text>
      </View>
      <View style={styles.statBox}>
        <Text style={styles.statNumber}>37K</Text>
        <Text style={styles.statLabel}>Likes</Text>
      </View>
    </View>
    <View style={styles.buttonContainer}>
      <TouchableOpacity style={styles.button}>
        <Text style={styles.buttonText}>Edit Profile</Text>
      </TouchableOpacity>
      <TouchableOpacity style={styles.button}>
        <Text style={styles.buttonText}>Add Friends</Text>
      </TouchableOpacity>
    </View>
  </View>

  <View style={styles.photosSection}>
    <Text style={styles.sectionTitle}>Photos</Text>
    <ScrollView horizontal showsHorizontalScrollIndicator={false}>
      <Image source={require('../../assets/mo.jpeg')} style={styles.photo} />
      <Image source={require('../../assets/pho.jpeg')} style={styles.photo} />
      <Image source={require('../../assets/co1.jpeg')} style={styles.photo} />
```

```
                </ScrollView>
              </View>
            </ScrollView>
        </ImageBackground>
const styles = StyleSheet.create({
    loaderContainer: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
    backgroundImage: {
        flex: 1,
        resizeMode: 'cover', // This will make the background cover the entire screen
    },
    container: {
        flexGrow: 1,
        paddingHorizontal: 20,
        paddingTop: 40,
       backgroundColor: 'rgba(245, 245, 245, 0.8)', // Optional: Add a slight background overlay
to improve readability
    },
    headerContainer: {
        alignItems: 'center',
        marginBottom: 20,
    },
    profileImage: {
        width: 100,
        height: 100,
        borderRadius: 50,
        marginBottom: 10,
    },
    name: {
        fontSize: 24,
        fontWeight: 'bold',
```

```
    color: '#333',
},
profession: {
    fontSize: 16,
    color: '#888',
},
location: {
    fontSize: 14,
    color: '#777',
    marginBottom: 20,
},
statsContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    width: '100%',
    marginBottom: 20,
},
statBox: {
    alignItems: 'center',
},
statNumber: {
    fontSize: 18,
    fontWeight: 'bold',
    color: '#333',
},
statLabel: {
    fontSize: 14,
    color: '#777',
},
buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    width: '80%',
    marginBottom: 20,
```

```
},
button: {
  backgroundColor: '#4A90E2',
  paddingVertical: 10,
  paddingHorizontal: 20,
  borderRadius: 5,
  marginHorizontal: 5,
},
buttonText: {
  color: '#fff',
  fontWeight: 'bold',
  textAlign: 'center',
},
photosSection: {
  marginTop: 20,
},
sectionTitle: {
  fontSize: 18,
  fontWeight: 'bold',
  marginBottom: 10,
},
photo: {
  width: 100,
  height: 100,
  borderRadius: 10,
  marginRight: 10,
```

# CAFE 7 APP

Submitted by

**KESAVAN S**

**1P23MC029**

In partial fulfillment of the requirements for the award of the Degree of

**MASTER OF COMPUTER APPLICATIONS**

From Bharathiar University, Coimbatore.

Under the internal supervision of

**Dr. S. RANJITHA KUMARI, M.Sc., M.Phil., Ph.D.**

**Associate Professor**

**SCHOOL OF COMPUTER STUDIES**

**MASTER OF COMPUTER APPLICATIONS**

**RVS COLLEGE OF ARTS AND SCIENCE(AUTONOMOUS)**

**Sulur, Coimbatore – 641 402.**

**April 2025.**

**RVS COLLEGE OF ARTS AND SCIENCE**

**(AUTONOMOUS)**

**Sulur, Coimbatore – 641 402.**

**School of Computer Studies**

**Department of Computer Applications (MCA)**

**April 2025**

**Register Number: 1P23MC029**

Certified bona fide original record work done by KESAVAN S

**Guide** **Director**

Submitted for the project Evaluation and *Viva-Voce* held on

**Internal Examiner** **External Examine**

**Date:**

# TO WHOM IT MAY CONCERN

This is to certify that the project work titled "**CAFÉ-7"** using MERN Stack is a Bonafide work of **KESAVAN S (1P23MC029).** She has completed the In-house project work at Department of MCA, RVS College of Arts and Science, Coimbatore during the period from Dec 2024 to April 2025 towards the partial fulfilment of the requirement for the award of MCA degree prescribed by Bharathiar University, Coimbatore-46.

**Director**

# CERTIFICATE

This is to certify that the project work titled **CAFÉ 7 APP**, submitted to the School of Computer Studies-MCA, RVS College of Arts and Science, in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications is a record of original project work done by **KESAVAN S** during the period Dec 2024 to April 2025 towards the partial fulfilment of the requirement for the award of MCA prescribed by RVS College of Arts and Science, Sulur.

**Internal Supervisor**

# DECLARATION

I am, **KESAVAN S,** hereby declare that the project entitled **CAFÉ 7 APP,** submitted to the School of Computer Studies, RVS College of Arts and Science, in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications is a record of original project work done by me during the period Dec 2024 to April 2025 under the internal supervision of**, Dr. S.Ranjitha Kumari, Associate Professor, RVS College Of Arts SCIENCE(AUTONOMOUS)** From Bharathiar University, Coimbatore.

**Signature of the Candidate**

# ACKNOWLEDGEMENTS

I express my sincere thanks to our Managing trustee **Dr. K. Senthil Ganesh MBA (USA)., MS (UK)., Ph.D.,** for providing us with the adequate faculty and laboratory resources for completing my project successfully.

I take this as a fine opportunity to express my sincere thanks to **Dr. T. Sivakumar M.Sc., M. Phil., Ph.D., Principal,** RVS College of Arts and Science (Autonomous) for giving me the opportunity to undertake this project.

I express my sincere thanks to **Dr. P. Navaneetham M.Sc., M.Phil., Ph.D., Director (Administration), School of Computer Studies**, RVS College of Arts and Science**,** for the help and advice throughout the project.

I express my sincere thanks to **Dr. S. Yamini M.Sc., (CC)., M. Phil., Ph.D., Director (Academic), School of Computer Studies,** RVS College of Arts and Science, for her valuable guidance and prompt correspondence throughout the curriculum to complete the project.

I express my gratitude to **Dr. S. Ranjitha Kumari, M.Sc., M.Phil., Ph.D. Associate Professor, School of Computer Studies,** RVS College of Arts and Science, for her valuable guidance, support, encouragement and motivation rendered by her throughout this project.

Finally, I express my sincere thanks to all other staff members and my dear friends, and all dear and near for helping me to complete this project.

**KESAVAN S**

# ABSTRACT

Abstracting a café-7 App in a System developed with React Native, Node.js, and MongoDB brings together modern technologies to create a comprehensive solution for handling helping processes. and modular approach to implementing customers functionality. The Café-7 App is a mobile platform designed to facilitate and streamline the process of coffee, connecting Customers with those in need while ensuring data security and user privacy. At the core of this abstraction these components include it addresses the critical need for a user-friendly and efficient café-7 platform. Users can register with their personal information, including name, contact details, and type. Secure authentication mechanisms are implemented using tokens or passwords for user access. Customers can create profiles with their health information, history, and preferred Customers locations into different parts of your application.

The app includes key functionalities such as personalized recipe recommendations, an intuitive meal planning tool, a community forum for user interactions, and a user-friendly interface optimized for various devices. The system flow is structured to ensure seamless navigation and accessibility, providing users with easy access to their favorite features.

Future enhancements are planned to further enrich the user experience, including advanced personalization through AI-driven suggestions, integration with smart kitchen devices, and the introduction of augmented reality cooking guides. Additionally, the app will focus on community engagement through challenges, multilingual support, and expanded user-generated content options.

By continually evolving and adapting to user needs, the Café-7 App aims to be a comprehensive resource for Coffee lovers, promoting creativity in the kitchen while building a supportive and interactive culinary community. This app not only serves as a recipe hub but also as a platform for sharing knowledge, experiences, and inspiration, ultimately enhancing the joy of making event.
Sizes and devices, making the component user-friendly and accessible.

# CONTENTS

## CHAPTER 3



## CHAPTER 4

## CHAPTER 5



## CHAPTER 6



## ANNEXURE

# *Certificate*

# *Declaration*

# *Acknowledgement*

# *CAFÉ 7 APP*

# *Abstract*

# *Contents*

# *Introduction*

# *System Analysis*

# *System Design and Development*

# *Testing and Implementation*

# *Conclusion*

# *Bibliography*

# *Annexure*