LEDGER MANAGEMENT SYSTEM FOR SIVASAKTHI STORES

A CONSULTANCY PROJECT REPORT

Submitted by

IMMANUVEL R
(22ITR035)

KRITTIKA R
(22ITR053)

BARATH KUMAR R
(22ITL121)

in partial fulfilment of the requirements
for the award of the degree
of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638 060

MAY 2025

BONAFIDE CERTIFICATE

This is to certify that the Project report entitled **LEDGER MANAGEMENT SYSTEM FOR SIVASAKTHI STORES** is the bonafide record of project work done by **IMMANUVEL R (22ITR035), KRITTIKA R (22ITR053) and BARATH KUMAR R (22ITL121)** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in **INFORMATION TECHNOLOGY** of Anna university, Chennai during the year 2024-2025.

SUPERVISOR   **HEAD OF THE DEPARTMENT**

(Signature with seal)

Date:

Submitted for the end semester viva voce examination held on _____.

INTERNAL EXAMINER   **EXTERNAL EXAMINER**

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI   ERODE – 638060

MAY 2025

DECLARATION

We affirm that the Project Report titled **LEDGER MANAGEMENT SYSTEM FOR SIVASAKTHI STORES** being submitted in partial fulfilment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Date: **IMMANUVEL R**
                                                                                                        (22ITR035)

                                                                                                        KRITTIKA R
                                                                                                        (22ITR053)

                                                                                                        BARATH KUMAR R
                                                                                                        (22ITL121)

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Date: Name and Signature of the Supervisor with seal

ABSTRACT

SivaSakthi Stores is a retail management system designed to streamline operations for a store specializing in Diwali chits and related products. The project comprises a Flutter-based mobile application and a Node.js/Express backend with MongoDB, providing a robust platform for user management, payment processing, and administrative tasks. The mobile app offers a user-friendly interface for customers to log in, view payment histories, submit payment requests, and receive notifications, while admins can manage users, approve payments, and handle soft-deleted records via a trash system. The backend supports these features with RESTful APIs, incorporating soft deletion, notifications, and village-based user searches. Security features include admin authentication and data validation, though enhancements like JWT and input sanitization are recommended. This system aims to improve operational efficiency, customer engagement, and scalability for retail operations, with potential for future enhancements like push notifications and offline support.

ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of Almighty throughout different phases of the project and its successful completion.

We wish to express our gratefulness to our beloved Correspondent **Thiru. A.K. ILANGO B.Com., M.B.A., LLB.,** and all the trust members of Kongu Vellalar Institute of Technology Trust for providing all the necessary facilities to complete the project successfully.

We express our deep sense of gratitude to our beloved Principal **Dr. V. BALUSAMY B.E.(Hons)., M.Tech., Ph.D.,** for providing us an opportunity to complete the project.

We express our gratitude to **Dr. S. ANANDAMURUGAN M.E., Ph.D.,** Head of the Department, Department of Information Technology for his valuable suggestions.

We are thankful to our Project Coordinators **Dr. G.K. KAMALAM BE., ME., Ph.D.,** and **Ms. R. SANDHIYA BE., ME.,** for their valuable guidance and support to complete our project successfully.

We are highly indebted to **Dr. S. ANANDAMURUGAN ME., Ph.D.,** Associate Professor and also our guide, Department of Information Technology for his valuable supervision and advice for the fruitful completion of the project.

We are thankful to the faculty members of the Department of Information Technology for their valuable guidance and support.

We are obliged to thank **Mr. N. MADHUKUMAR** for supporting us through the various development phases of the project with utmost trust.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API   Application Programming Interface

JSON   JavaScript Object Notation

JWT   JSON Web Token

REST   Representational State Transfer

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

SivaSakthi Stores is an innovative retail management system developed to streamline operations for a specialty store dealing in Diwali chits and related products. The system comprises a user-friendly Flutter-based mobile application and a robust Node.js/Express backend, integrated with MongoDB for efficient data storage and management. The mobile app provides customers with seamless access to features such as user authentication, payment request submission, payment history viewing, and personalized notifications. Simultaneously, it equips administrators with powerful tools for managing users, approving or rejecting payment requests, generating reports (e.g., inactive customers, payments by month), and handling a trash system for soft-deleted users. The backend supports these functionalities through a comprehensive set of RESTful APIs, and village-based user searches to cater to the store's diverse customer base. The system is designed to enhance customer engagement and operational efficiency, making it a vital tool for modern retail management.

1.2 OBJECTIVE

The primary objective of the SivaSakthi Stores project is to create an efficient, scalable, and user-centric platform that simplifies store operations and improves customer experience. For customers, the system aims to provide an intuitive interface for managing their accounts, submitting payments for Diwali chits, and receiving timely updates via notifications. For administrators, it seeks to offer robust tools for user management (add, delete, restore users), payment processing, and data-driven insights through reports on payment statuses and customer activity. Additionally, the system is designed to be scalable, supporting future enhancements such as push notifications, offline capabilities, and advanced analytics. Ultimately, SivaSakthi Stores aims to modernize retail operations, foster customer loyalty, and provide a reliable, secure platform for managing store activities effectively.

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1 HARDWARE REQUIREMENTS

        Processor   **: Intel Pentium or higher**

        Processor Speed   **: Minimum 250 MHz to 667 MHz**

        RAM   **: Minimum 4 GB**

        Hard Disk   **: 256 GB or more**

2.2 SOFTWARE REQUIREMENTS

        Platform   **: Visual Studio Code**

        Backend Language   **: Javascript**

        Server-Side Framework   **: Node.js with Express.js**

        Database   **: MongoDB**

        Frontend Framework   **: Flutter**

2.3   SOFTWARE DESCRIPTION

2.3.1 Visual Studio Code

Visual Studio Code is a lightweight yet powerful source code editor developed by Microsoft. It provides features such as syntax highlighting, intelligent code completion, version control integration, and an integrated terminal, making it an ideal development environment for both frontend and backend development. In this project, VS Code is used as the primary code editor for writing TypeScript, Node.js, and React code efficiently.

2.3.2 JavaScript

JavaScript is a lightweight, interpreted programming language that is primarily used for creating dynamic and interactive content on websites. It is one of the core technologies of web development, alongside HTML and CSS.

2.3.3 Node.js

Node.js is a runtime environment that executes JavaScript code outside the browser. It is used in the backend to handle server-side logic, API creation, routing, and real-time data processing. With its non-blocking I/O model and rich package ecosystem, Node.js provides the foundation for building a fast and scalable inventory system backend.

2.3.4 Express.js

Express.js is a minimalist web framework for Node.js used to build RESTful APIs in the backend. It simplifies routing, middleware handling, and error management. In this project, Express handles all inventory operations like product management, user authentication, and transaction logging through secure and modular endpoints.

2.3.5 MongoDB & Mongoose

MongoDB is a NoSQL database used to store structured inventory data in the form of documents. Its flexibility and scalability make it suitable for storing dynamic and hierarchical data such as products, categories, and stock levels. Mongoose is an Object Data Modeling (ODM) library used with MongoDB to define schemas, perform validations, and manage database queries efficiently.

2.3.6 Flutter

Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, desktop, and embedded devices from a single codebase. It uses the Dart programming language and provides a rich set of customizable widgets, enabling developers to create visually appealing and high-performance apps with a consistent look across platforms. Flutter's hot-reload feature allows developers to see changes instantly, making the development process faster. Its flexible UI, strong community support, and extensive library ecosystem have made it popular for cross-platform app development.

CHAPTER 3

SYSTEM DESIGN

3.1 USE CASE DESIGN

A use case diagram is a dynamic or behaviour diagram in UML (Unified Modelling Language).Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions and functions that the system needs to perform. A "system" is something being developed or operated, here a website. The "actors" are people or entities operating under defined roles within the system as shown below in Figure 3.1.
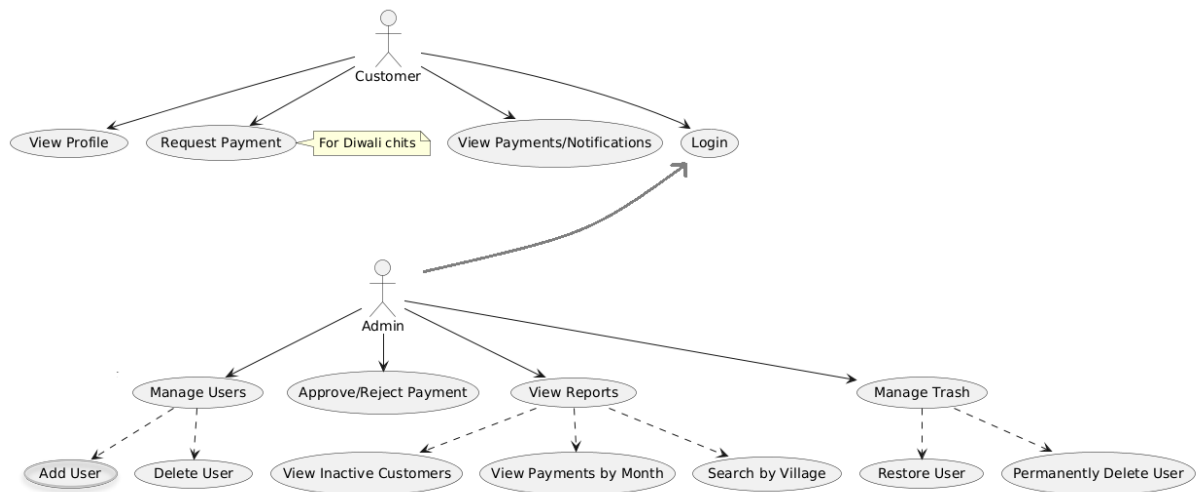
Fig 3.1 Use Case Diagram

3.2 CLASS DIAGRAM

A class diagram is a static structure diagram in UML (Unified Modeling Language) that illustrates the structure of a system by showing the classes of the system, their attributes, methods, and relationships between them. Classes represent the blueprint for objects, encapsulating data and behavior. Attributes are the data members of a class, while methods represent the operations that can be performed on the class's objects as shown below in Figure 3.2.
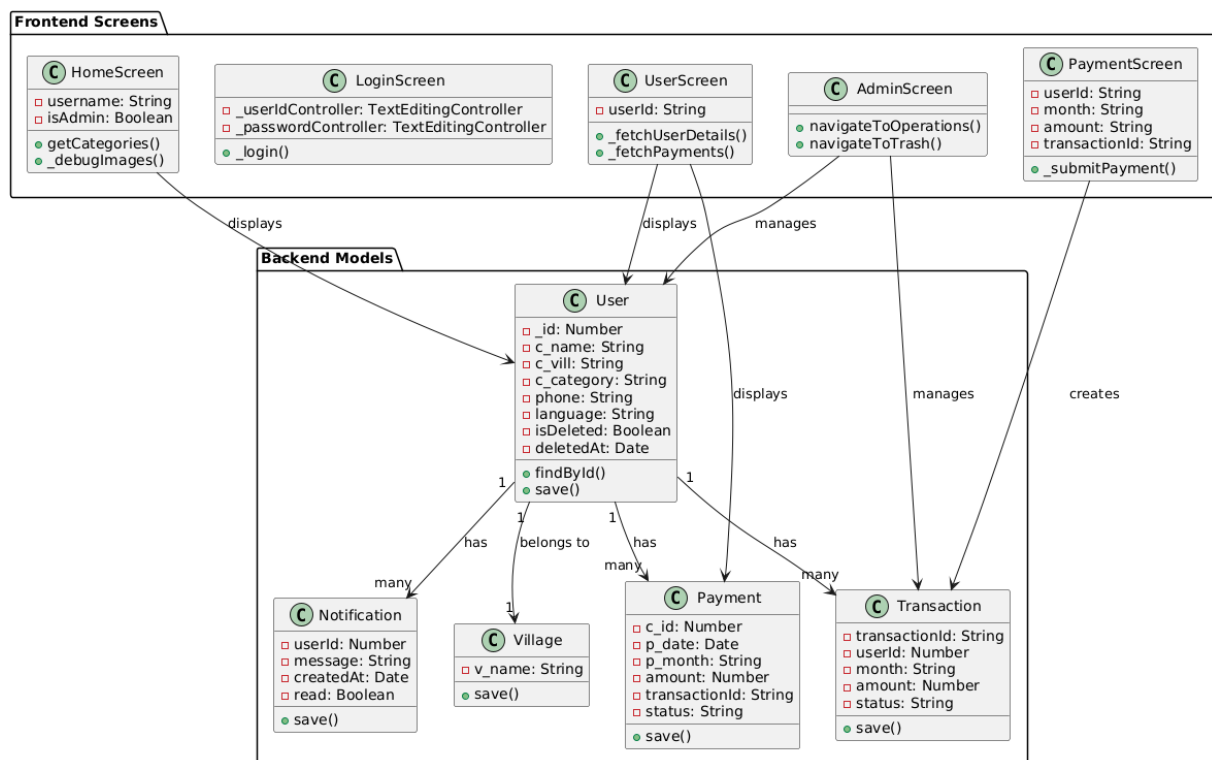
Fig 3.2 Class Diagram

3.3 SEQUENCE DIAGRAM

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects and what messages trigger those communications as shown below in Figure 3.3. Sequence diagrams are not intended for showing complex procedural logic.
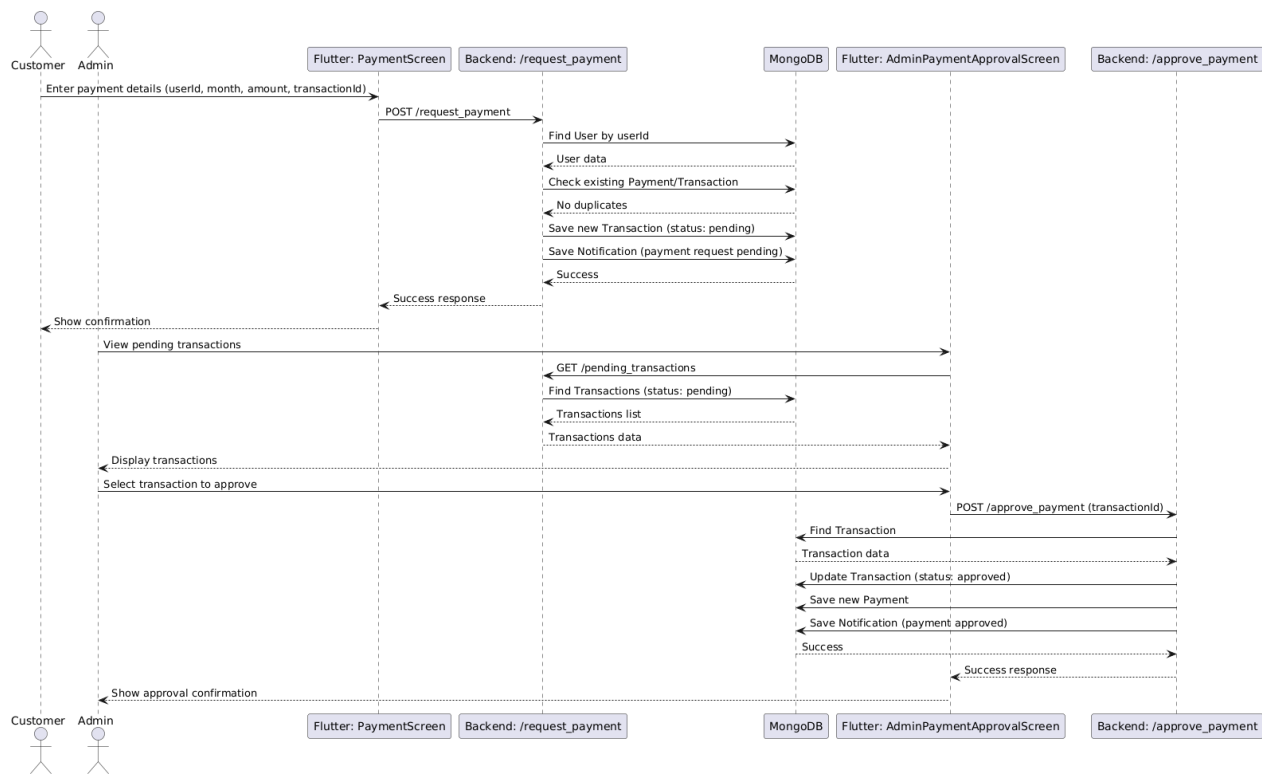
Fig 3.3 Sequence Diagram

3.4. ACTIVITY DIAGRAM

In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities. The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched or concurrent and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc. as shown in the Figure 3.4 below.
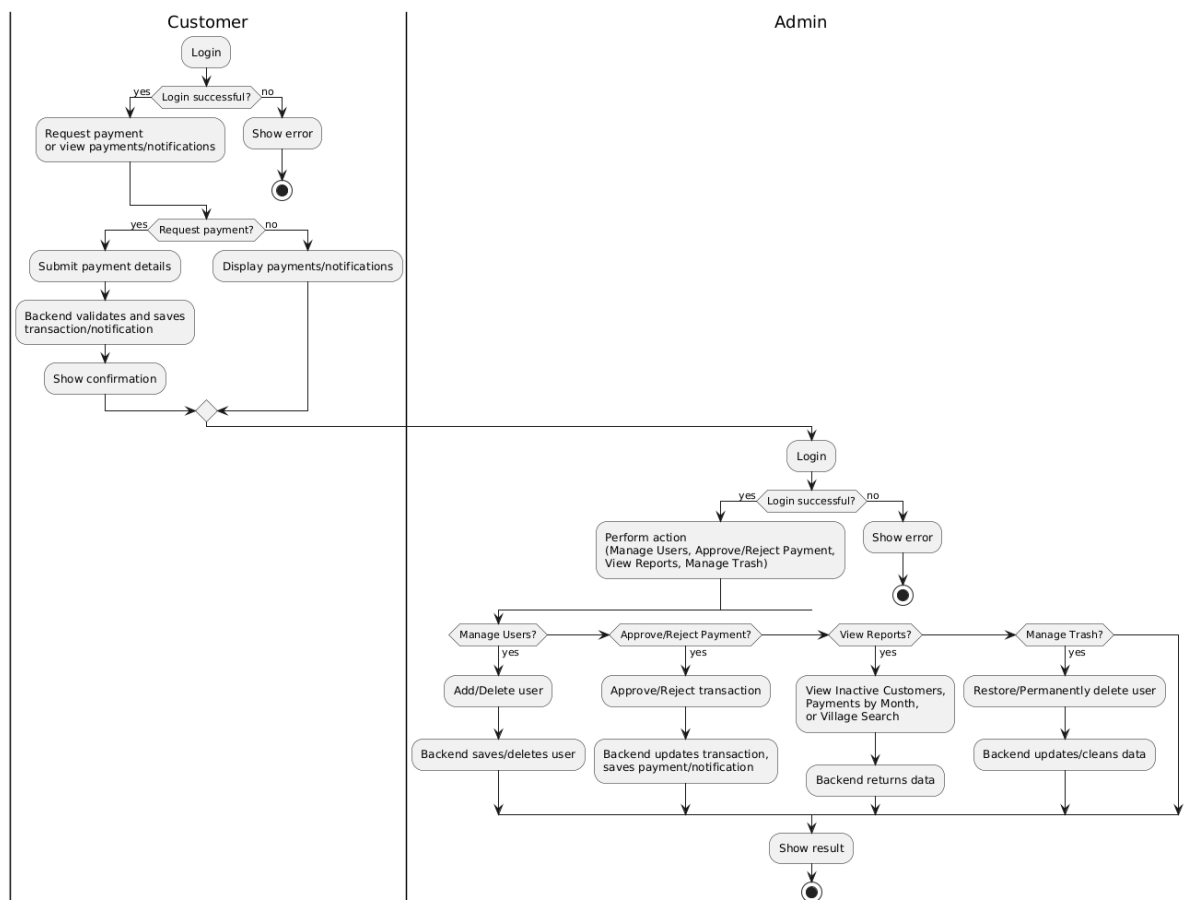
Fig 3.4 Activity Diagram

3.5 DATABASE DESIGN

3.5.1 USER SCHEMA

users

- _id: { type: Number, required: true, unique: true }

- c_name: { type: String, required: true }

- c_vill: { type: String, required: true }

- c_category: { type: String, required: true }

- phone: { type: String, required: true }

- language: { type: String, default: "en" }

- isDeleted: { type: Boolean, default: false }

- deletedAt: { type: Date, default: null }

3.5.2 PAYMENT SCHEMA

payments

- _id: { type: ObjectId, auto: true }

- c_id: { type: Number, required: true }

- p_date: { type: Date, required: true }

- p_month: { type: String, required: true }

- amount: { type: Number, required: true }

- transactionId: { type: String, required: true }

- status: { type: String, enum: ["pending", "approved", "rejected"], default: "pending" }

3.5.3 VILLAGE SCHEMA

villages
  - _id: { type: ObjectId, auto: true }
  - v_name: { type: String, required: true, unique: true }

3.5.4 NOTIFICATION SCHEMA

notifiactions
  - _id: { type: ObjectId, auto: true }
  - userId: { type: Number, required: true }
  - message: { type: String, required: true }
  - createdAt: { type: Date, default: Date.now }
  - read: { type: Boolean, default: false }

3.5.5 TRANSACTION SCHEMA

transactions
  - _id: { type: ObjectId, auto: true }
  - transactionId: { type: String, required: true, unique: true }
  - userId: { type: Number, required: true }
  - month: { type: String, required: true }
  - amount: { type: Number, required: true }
  - status: { type: String, enum: ["pending", "approved", "rejected"], default: "pending" }

3.6   MODULES DESCRIPTION

3.6.1 Authentication Module

Manages user and admin authentication to secure access to the system by authenticating customers using user ID and phone number, and admins using environment-stored credentials. It enables secure access to the Flutter app's features, such as LoginScreen, supports role-based access (admin vs. customer), and provides language preference for personalized notifications.

3.6.2 User Management Module

Handles creation, retrieval, updating, and soft deletion of user records by adding new users with village creation if needed, retrieving user details by ID or searching by name, category, or village, listing all active users with payment counts, and soft deleting users (moving them to trash). It powers user management in AdminOperationsScreen (Flutter app), and integrates with the Notification Module for user creation alerts.

3.6.3 Payment Management Module

Manages payment requests, approvals, rejections, and payment history by allowing customers to submit payment requests, enabling admins to approve or reject them, recording payments, checking payment status for specific months, and tracking total payments per user. It drives PaymentScreen and AdminPaymentApprovalScreen in the Flutter app, supports Diwali chit payment workflows with an approval process, and integrates with the Notification Module for payment status updates.

3.6.4 Notification Module

Manages in-app notifications for users, such as payment updates and welcomes, by creating notifications for user actions (e.g., payment requests, approvals), retrieving notifications sorted by creation date, and supporting multilingual notifications based on user language. It powers NotificationsScreen in the Flutter app, enhances user engagement with timely, personalized updates, and integrates with other modules (e.g., Payment, User) for event-driven notifications.

3.6.5 Trash Management Module

Handles soft deletion, restoration, and permanent deletion of users by marking them as deleted with a timestamp, listing soft-deleted users (trash), restoring them to active status, and permanently deleting users along with their related data (Payments, Notifications, Transactions). It supports TrashScreen in the Flutter app for managing deleted users and ensures data recovery and cleanup for store records.

3.6.6 Reporting Module

Generates reports on customer activity, payments, and inactivity by listing paid/unpaid users for a specific month, identifying inactive customers (no payments in the last two months), and supporting village-based user searches (overlapping with Village Management). It provides analytics for admins in AdminOperationsScreen and helps track store performance and customer engagement.

CHAPTER 4

SYSTEM TESTING

4.1 SYSTEM TESTING

The goal of SivaSakthi Stores is to provide a high-quality retail management system for managing Diwali chit payments and user records. Comprehensive system testing ensures reliability, security, and performance by identifying defects under various conditions. It verifies that the software meets functional and non-functional requirements, ensuring the product is market-ready. The testing process includes evaluating, analyzing, and validating system components to ensure customer satisfaction, cost efficiency, and operational reliability. The process includes four key phases:

- Unit testing
- Module testing
- Integration testing
- Validation testing

4.2 UNIT TESTING

Unit testing targets the smallest components of the system, like functions or methods in the Node.js backend and Flutter app. Developers test these units during coding to ensure correct operation. For example, backend functions like user creation and payment validation, as well as Flutter widgets like LoginScreen, were tested with mock data. This approach enabled early detection and correction of errors.

4.3 MODULE TESTING

Each module of the SivaSakthi Stores system was thoroughly tested for stable functionality and accurate data handling. The Authentication Module ensured secure login for users and admins. User Management was validated for adding, updating, deleting, and searching users, including village creation. Payment Management handled requests, approvals, rejections, and history accurately. Notifications generated and displayed multilingual alerts. Trash Management supported soft deletion, restoration, and permanent removal. Village Management enabled village creation and user searches. Reporting generated payment status and inactivity

reports. Internationalization ensured language updates and translations, while Device Management verified device token registration. All modules functioned reliably, providing a smooth user experience.

4.4 INTEGRATION TESTING

Integration testing verified seamless module interaction in the SivaSakthi Stores system. Authentication integrated with User and Payment Management to ensure user/admin login granted proper access. User Management linked with Payment and Notification Modules to trigger welcome notifications and maintain payment records. Payment Management synchronized with Transaction and Notification Modules for accurate status updates and alerts. Village Management worked with User Management for accurate village-based searches. Reporting integrated with Payment and User Management for accurate payment and inactivity reports. Trash Management ensured permanent deletions cleared related data. Data flow across modules was consistent, synchronized, and accurate, ensuring smooth system operation.

4.5 VALIDATION TESTING

Validation testing confirmed that the SivaSakthi Stores system met functional and usability requirements. The Authentication Module ensured valid user inputs to prevent unauthorized access. User Management validated admin actions with checks for unique IDs and valid village names. Payment Management verified correct user existence and transaction IDs. The Notification Module delivered translated messages based on user language preferences. Trash Management maintained data integrity during restoration and deletion. The Reporting Module generated accurate reports, and the Internationalization Module ensured seamless language updates. Overall, the system was reliable, functional, and user-friendly for store management.

CHAPTER 5

RESULTS

The SivaSakthi Stores system successfully achieved its objectives, delivering an efficient and intuitive platform for managing Diwali chit payments, user records, and store operations. The User Management Module enables seamless addition, updating, and soft deletion of customer records, with village-based categorization for organized data. The Payment Management Module facilitates payment requests, approvals, and history tracking, ensuring accurate financial transactions with real-time updates. The Notification Module provides multilingual alerts, enhancing customer engagement. The Trash Management Module supports data recovery and cleanup, maintaining system integrity. The Reporting Module generates insightful reports on payment statuses and inactive customers, aiding data-driven decisions. The Internationalization Module ensures accessibility across languages (English, Tamil, Telugu, Kannada, Hindi), broadening the user base. Validation checks prevent errors like duplicate payments or invalid user inputs, ensuring data accuracy. Overall, the system streamlines store operations, reduces manual errors, and enhances user experience, making it a reliable tool for retail management.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The SivaSakthi Stores system marks a significant advancement in retail management for stores handling Diwali chits. By integrating a Flutter app with a Node.js/Express backend, it offers a robust platform for user management, payment processing, and reporting. The system ensures efficient operations through real-time payment tracking, multilingual notifications, and a trash system for data management. The intuitive interface, coupled with validation checks, minimizes errors and enhances usability. Reporting tools provide valuable insights, empowering admins to optimize store performance. The system fosters operational efficiency and customer satisfaction, creating a dynamic retail environment.

Future work could enhance the system by integrating push notifications via Firebase Cloud Messaging, leveraging the Device Management Module's groundwork. Adding offline support in the Flutter app would improve accessibility in low-connectivity areas. Implementing JWT-based authentication and password hashing would strengthen security. Advanced analytics, such as predictive payment trends, could provide deeper insights. Expanding the system to support multiple stores or integrating with payment gateways for online transactions would increase scalability. Finally, introducing role-based access for multiple admin levels would enable collaborative management, further streamlining operations.

APPENDIX 1

CODING

Main.dart

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'dart:async';
import 'package:url_launcher/url_launcher.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:geolocator/geolocator.dart';
import 'package:flutter/services.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
  ]);
  runApp(MyApp());
}

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  ThemeData _theme = AppTheme.lightTheme;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'SivaSakthi Stores',
      theme: _theme,
      home: SplashScreen(),
    );
  }
}

class AppTheme {
  static const Color primaryColor = Color(0xFF5C2D91);
  static const Color accentColor = Color(0xFFFF6B35);
  static const Color backgroundColor = Color(0xFFF8F9FA);
  static const Color cardColor = Colors.white;
  static const Color textPrimary = Color(0xFF212529);
  static const Color textSecondary = Color(0xFF6C757D);
  static const Color success = Color(0xFF28A745);
  static const Color danger = Color(0xFFDC3545);
  static const Color warning = Color(0xFFFFC107);
  static const Color info = Color(0xFF17A2B8);

  static ThemeData lightTheme = ThemeData(
    primaryColor: primaryColor,
    colorScheme: ColorScheme.light(
      primary: primaryColor,
```

```
    secondary: accentColor,
    background: backgroundColor,
    surface: cardColor,
  ),
  scaffoldBackgroundColor: backgroundColor,
  cardTheme: CardTheme(
    color: cardColor,
    elevation: 2,
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
  ),
  appBarTheme: AppBarTheme(
    backgroundColor: primaryColor,
    elevation: 0,
    centerTitle: true,
    iconTheme: IconThemeData(color: Colors.white),
    titleTextStyle: TextStyle(
      color: Colors.white,
      fontSize: 20,
      fontWeight: FontWeight.bold,
    ),
  ),
  elevatedButtonTheme: ElevatedButtonThemeData(
    style: ElevatedButton.styleFrom(
      backgroundColor: primaryColor,
      foregroundColor: Colors.white,
      padding: EdgeInsets.symmetric(horizontal: 16, vertical: 12),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
      elevation: 2,
    ),
  ),
  outlinedButtonTheme: OutlinedButtonThemeData(
    style: OutlinedButton.styleFrom(
      foregroundColor: primaryColor,
      side: BorderSide(color: primaryColor),
      padding: EdgeInsets.symmetric(horizontal: 16, vertical: 12),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
    ),
  ),
  textButtonTheme: TextButtonThemeData(
    style: TextButton.styleFrom(
      foregroundColor: primaryColor,
      padding: EdgeInsets.symmetric(horizontal: 16, vertical: 12),
    ),
  ),
  inputDecorationTheme: InputDecorationTheme(
    filled: true,
    fillColor: Colors.white,
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(8),
      borderSide: BorderSide(color: Colors.grey.shade300),
    ),
    enabledBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(8),
      borderSide: BorderSide(color: Colors.grey.shade300),
    ),
    focusedBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(8),
      borderSide: BorderSide(color: primaryColor, width: 2),
    ),
    errorBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(8),
```

```dart
        borderSide: BorderSide(color: danger, width: 1),
      ),
      contentPadding: EdgeInsets.symmetric(horizontal: 16, vertical: 16),
    ),
                        Spacer(),
                        ElevatedButton(
                          onPressed: () {
                            Navigator.push(
                              context,
                              MaterialPageRoute(
                                builder: (context) => CategoryDetailsPage(category: category),
                              ),
                            );
                          },
                          child: Text("View Products"),
                          style: ElevatedButton.styleFrom(
                            backgroundColor: category["title"] == "Gold Category"
                                ? Color(0xFFFFFD700)
                                : Color(0xFFC0C0C0),
                            foregroundColor: Colors.black87,
                          ),
                        ),
                      ],
                    ),
                  ),
                  Padding(
                    padding: const EdgeInsets.all(16.0),
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Text(
                          "Featured Products:",
                          style: TextStyle(
                            fontSize: 16,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                        SizedBox(height: 12),
                        SingleChildScrollView(
                          scrollDirection: Axis.horizontal,
                          child: Row(
                            children: List.generate(
                              category["products"].length > 5 ? 5 : category["products"].length,
                              (productIndex) {
                                final product = category["products"][productIndex];
                                return Container(
                                  width: 120,
                                  margin: EdgeInsets.only(right: 12),
                                  child: Column(
                                    crossAxisAlignment: CrossAxisAlignment.start,
                                    children: [
                                      ClipRRect(
                                        borderRadius: BorderRadius.circular(8),
                                        child: Image.asset(
                                          product["image"],
                                          height: 80,
                                          width: 120,
                                          fit: BoxFit.cover,
                                          errorBuilder: (context, error, stackTrace) {
                                            return Container(
                                              height: 80,
```

```dart
                                                        width: 120,
                                                        color: Colors.grey[200],
                                                        child: Icon(Icons.image_not_supported, color: Colors.grey[400]),
                                                      );
                                                    },
                                                  ),
                                                ),
                                                SizedBox(height: 8),
                                                Text(
                                                  product["name"],
                                                  style: TextStyle(
                                                    fontWeight: FontWeight.bold,
                                                    fontSize: 14,
                                                  ),
                                                  maxLines: 1,
                                                  overflow: TextOverflow.ellipsis,
                                                ),
                                                Text(
                                                  product["quantity"],
                                                  style: TextStyle(
                                                    fontSize: 12,
                                                    color: AppTheme.textSecondary,
))),),],),);},),),),),);},),],),],),),),),],),),),);},),),),),);},),],),),),),),),);}
  Widget buildPersonCard(String name, String role, String imagePath) {
    return Container(
      width: 100,
      child: Column(
        children: [
          Container(
            height: 100,
            width: 100,
            decoration: BoxDecoration(
              shape: BoxShape.circle,
              border: Border.all(color: AppTheme.primaryColor, width: 2),
              boxShadow: [
                BoxShadow(
                  color: Colors.black.withOpacity(0.1),
                  blurRadius: 8,
                  offset: Offset(0, 4),
                ),
              ],
            ),
            child: ClipOval(
              child: Image.asset(
                imagePath,
                fit: BoxFit.cover,
                errorBuilder: (context, error, stackTrace) {
                  print('Error loading person image: $imagePath - $error');
                  return Container(
                    color: Colors.grey[300],
                    child: Icon(Icons.person, size: 50, color: Colors.grey[600]),
                  );
                },
              ),
            ),
          ),
          SizedBox(height: 8),
          Text(
            name,
            style: TextStyle(
              fontSize: 16,
```

```dart
              fontWeight: FontWeight.bold,
            ),
            textAlign: TextAlign.center,
          ),
          Text(
            role,
            style: TextStyle(
              fontSize: 14,
              color: AppTheme.textSecondary,
            ),
            textAlign: TextAlign.center,
          ),
        ],
      ),
    );
  }
}

class CategoryDetailsPage extends StatelessWidget {
  final Map<String, dynamic> category;

  const CategoryDetailsPage({Key? key, required this.category}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(category['title']),
      ),
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // Header
          Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
              color: category["title"] == "Gold Category"
                  ? Color(0xFFFFD700).withOpacity(0.2)
                  : Color(0xFFC0C0C0).withOpacity(0.2),
            ),
            child: Row(
              children: [
                Container(
                  padding: EdgeInsets.all(12),
                  decoration: BoxDecoration(
                    color: Colors.white,
                    shape: BoxShape.circle,
                  ),
                  child: Icon(
                    category["title"] == "Gold Category"
                        ? Icons.star
                        : Icons.star_half,
                    color: category["title"] == "Gold Category"
                        ? Color(0xFFFFD700)
                        : Color(0xFFC0C0C0),
                    size: 32,
                  ),
                ),
                SizedBox(width: 16),
                Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
```

```
        children: [
          Text(
            category["title"],
            style: TextStyle(
              fontSize: 22,
              fontWeight: FontWeight.bold,
            ),
          ),
          SizedBox(height: 4),
          Text(
            category["amount"],
            style: TextStyle(
              fontSize: 18,
              color: AppTheme.textSecondary,
            ),
          ),
        ],
      ),
    ],
  ),
),

// Products Count
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Row(
    children: [
      Text(
        "Products",
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(width: 8),
      Container(
        padding: EdgeInsets.symmetric(horizontal: 8, vertical: 4),
        decoration: BoxDecoration(
          color: AppTheme.primaryColor,
          borderRadius: BorderRadius.circular(12),
        ),
        child: Text(
          "${category['products'].length}",
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ],
  ),
),

// Products Grid
Expanded(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16.0),
    child: GridView.builder(
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        childAspectRatio: 0.75,
```

```dart
        crossAxisSpacing: 16,
        mainAxisSpacing: 16,
      ),
      itemCount: category['products'].length,
      itemBuilder: (context, index) {
        final product = category['products'][index];
        return Container(
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(12),
            boxShadow: [
              BoxShadow(
                color: Colors.black.withOpacity(0.05),
                blurRadius: 10,
                offset: Offset(0, 5),
              ),
            ],
          ),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              ClipRRect(
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(12),
                  topRight: Radius.circular(12),
                ),
                child: Image.asset(
                  product['image'],
                  height: 120,
                  width: double.infinity,
                  fit: BoxFit.cover,
                  errorBuilder: (context, error, stackTrace) {
                    print('Error loading product image: ${product['image']} - $error');
                    return Container(
                      height: 120,
                      width: double.infinity,
                      color: Colors.grey[200],
                      child: Icon(Icons.image_not_supported, size: 40, color: Colors.grey[400]),
                    );
                  },
                ),
              ),
              Padding(
                padding: const EdgeInsets.all(12.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      product['name'],
                      style: TextStyle(
                        fontSize: 16,
                        fontWeight: FontWeight.bold,
                      ),
                      maxLines: 1,
                      overflow: TextOverflow.ellipsis
                    SizedBox(height: 4),
                    Container(
                      padding: EdgeInsets.symmetric(horizontal: 8, vertical: 4),
                      decoration: BoxDecoration(
```

color:AppTheme.primaryColor.withOpacity(0.1),)),),],),,);},),),),),;},),],),],),),),],),),),;},),),),),;},),],),),),),),),),);}

```
class LoginScreen extends StatefulWidget {
 @override
 _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
 final _formKey = GlobalKey<FormState>();
 final _userIdController = TextEditingController();
 final _passwordController = TextEditingController();
 bool _isLoading = false;
 bool _obscurePassword = true;

 Future<void> _login() async {
  if (_formKey.currentState!.validate()) {
   setState(() {
    _isLoading = true;
   });

   try {
    final response = await http.post(
     Uri.parse('https://nanjundeshwara.vercel.app/login'),
     headers: {'Content-Type': 'application/json'},
     body: json.encode({
      'username': _userIdController.text,
      'password': _passwordController.text
     }),
    );

    setState(() {
     _isLoading = false;
    });

    if (response.statusCode == 200) {
     final data = json.decode(response.body);
     if (data['success']) {
      final prefs = await SharedPreferences.getInstance();
      await prefs.setString('username', _userIdController.text);
      await prefs.setBool('isAdmin', data['isAdmin']);

      Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => HomeScreen(
        username: _userIdController.text,
        isAdmin: data['isAdmin']
       )),
      );

 Future<void> _rejectPayment(String transactionId) async {
  try {
   final response = await http.post(
    Uri.parse('https://nanjundeshwara.vercel.app/reject_payment'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({'transactionId': transactionId}),
   );

    ScaffoldMessenger.of(context).showSnackBar(
     SnackBar(
      content: Text('Payment rejected'),
      backgroundColor: AppTheme.warning,
     ),
    );
```

```
        _fetchPendingTransactions();

  } catch (e) {
   ScaffoldMessenger.of(context).showSnackBar(
     SnackBar(
       content: Text('An error occurred: $e'),
       backgroundColor: AppTheme.danger,
     ),
   );
  }
}
if (response.statusCode == 200) {
    final data = json.decode(response.body);
    setState(() {
      _users = List<Map<String, dynamic>>.from(data['users']);
    });
   } else {
     _showErrorSnackBar('Failed to fetch users: ${response.body}');
   }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

Future<void> _addPayment() async {
  if (_formKey.currentState!.validate()) {
    try {
      final response = await http.post(
        Uri.parse('https://nanjundeshwara.vercel.app/add_payments'),
        headers: {'Content-Type': 'application/json'},
        body: json.encode({
          'c_id': int.parse(_userIdController.text),
          'p_month': _monthController.text,
          'amount': double.parse(_amountController.text),
        }),
      );

      _showSuccessSnackBar('Payment added successfully');
    } catch (e) {

    }
  }
}

Future<void> _viewPayments() async {
  if (_userIdController.text.isEmpty) {
    _showErrorSnackBar('Please enter a user ID to view payments');
    return;
  }

  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/find_payments?userIdPayments=${_userIdController.text}'),
    );

    if (response.statusCode == 200) {
      setState(() {
        _payments = List<Map<String, dynamic>>.from(json.decode(response.body));
      });
    } else {
      _showErrorSnackBar('Failed to fetch payments: ${response.body}');
```

```dart
      }
    } catch (e) {
      _showErrorSnackBar('An error occurred: $e');
    }
  }


  Widget _buildAddPaymentForm() {
    return Card(
      elevation: 2,
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                'Add Payment',
                style: TextStyle(
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                ),
              ),
              SizedBox(height: 20),
              TextFormField(
                controller: _userIdController,
                decoration: InputDecoration(
                  labelText: 'User ID',
                  prefixIcon: Icon(Icons.badge),
                ),
                keyboardType: TextInputType.number,
                validator: (value) => value!.isEmpty ? 'Please enter user ID' : null,
                onChanged: (_) => _fetchUserDetails(),
              ),
              SizedBox(height: 16),
              TextFormField(
                controller: _nameController,
                decoration: InputDecoration(
                  labelText: 'Name',
                  prefixIcon: Icon(Icons.person),
                ),
                readOnly: true,
                enabled: false,
              ),
              SizedBox(height: 16),
              TextFormField(
                controller: _amountController,
                decoration: InputDecoration(
                  labelText: 'Amount',
                  prefixIcon: Icon(Icons.currency_rupee),
                ),
                keyboardType: TextInputType.number,
                validator: (value) => value!.isEmpty ? 'Please enter amount' : null,
              ),
              SizedBox(height: 16),
              TextFormField(
                controller: _monthController,
                decoration: InputDecoration(
                  labelText: 'Month (MM format)',
                  prefixIcon: Icon(Icons.calendar_today),
```

```
            hintText: 'e.g. 01 for January',
          ),
          validator: (value) => value!.isEmpty ? 'Please enter month' : null,
        ),
        SizedBox(height: 24),
        SizedBox(
          width: double.infinity,
          child: ElevatedButton.icon(
            onPressed: _addPayment,
            icon: Icon(Icons.add),
            label: Text('Add Payment'),
            style: ElevatedButton.styleFrom(
              padding: EdgeInsets.symmetric(vertical: 16),),   ),   ),  ],  ),  ),  ),  );
      }

Widget _buildViewPaymentsContent() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'View Payments',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(height: 20),
      TextFormField(
        controller: _userIdController,
        decoration: InputDecoration(
          labelText: 'User ID',
          prefixIcon: Icon(Icons.badge),
          hintText: 'Enter user ID to view their payments',
        ),
        keyboardType: TextInputType.number,
      ),
      SizedBox(height: 20),
      SizedBox(
        width: double.infinity,
        child: ElevatedButton.icon(
          onPressed: _viewPayments,
          icon: Icon(Icons.search),
          label: Text('View Payments'),
          style: ElevatedButton.styleFrom(
            padding: EdgeInsets.symmetric(vertical: 16),
          ),
        ),
      ),
      SizedBox(height: 20),
      Expanded(
        child: _payments.isEmpty
            ? Center(
                child: Text(
                  'No payments found',
                  style: TextStyle(
                    fontSize: 16,
                    color: AppTheme.textSecondary,
                  ),
                ),
              )
            : ListView.builder(
```

```dart
                itemCount: _payments.length,
                itemBuilder: (context, index) {
                  final payment = _payments[index];
                  return Card(
                    margin: EdgeInsets.only(bottom: 8),
                    child: ListTile(
                      leading: CircleAvatar(
                        backgroundColor: AppTheme.success.withOpacity(0.1),
                        child: Icon(
                          Icons.payment,
                          color: AppTheme.success,
                        ),
                      ),
                      title: Text(
                        'Amount: ₹${payment['amount']}',
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                      subtitle: Text('Month: ${payment['p_month']}, User: ${payment['c_name']}'),
                    ),
                }

  Future<void> _logout(BuildContext context) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.clear();
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => HomeScreen()),
    );
  }

  void _showErrorSnackBar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(message),
        backgroundColor: AppTheme.danger,
        behavior: SnackBarBehavior.floating,
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    if (_isLoading) {
      return Scaffold(
        appBar: AppBar(
          title: Text('User Dashboard'),
        ),
        body: Center(
          child: CircularProgressIndicator(),
        ),
      );
    }

    return Scaffold(
      appBar: AppBar(
        title: Text('User Dashboard'),
        actions: [
          IconButton(
            icon: Icon(Icons.notifications),
```

```
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => NotificationsScreen(notifications: _notifications)),
        );
      },
      tooltip: 'Notifications',
    ),
    IconButton(
      icon: Icon(Icons.payment),
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => PaymentScreen(username: widget.username)),
        );
      },
      tooltip: 'Make Payment',
    ),
  ],
),
body: RefreshIndicator(
  onRefresh: _fetchData,
  child: SingleChildScrollView(
    physics: AlwaysScrollableScrollPhysics(),
    child: Padding(
      padding: EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // User Profile Card
          Card(
            elevation: 2,
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                children: [
                  Row(
                    children: [
                      Container(
                        width: 60,
                        height: 60,
                        decoration: BoxDecoration(
                          color: AppTheme.primaryColor.withOpacity(0.1),
                          shape: BoxShape.circle,
                        ),
                        child: Center(
                          child: Text(
                            _userDetails['c_name'] != null && _userDetails['c_name'].isNotEmpty
                                ? _userDetails['c_name'][0].toUpperCase()
                                : 'U',
                            style: TextStyle(
                              fontSize: 24,
                              fontWeight: FontWeight.bold,
                              color: AppTheme.primaryColor,
                            ),
                          ),
                        ),
                      ),
                      SizedBox(width: 16),
                      Expanded(
                        child: Column(
```

```
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
           Text(
            'Welcome, ${_userDetails['c_name'] ?? widget.username}!',
            style: TextStyle(
             fontSize: 20,
             fontWeight: FontWeight.bold,
            ),
           ),
           SizedBox(height: 4),
           Text(
            'User ID: ${_userDetails['_id'] ?? 'N/A'}',
            style: TextStyle(
             fontSize: 14,
             color: AppTheme.textSecondary,
            ),
           ),
          ],
         ),
        ),
       ],
      ),
      SizedBox(height: 20),
      Divider(),
      SizedBox(height: 12),
      _buildUserInfoRow(Icons.location_on, 'Village', _userDetails['c_vill'] ?? 'N/A'),
      SizedBox(height: 8),
      _buildUserInfoRow(Icons.phone, 'Phone', _userDetails['phone'] ?? 'N/A'),
      SizedBox(height: 8),
      _buildUserInfoRow(
       Icons.category,
       'Category',
       _userDetails['c_category'] ?? 'N/A',
       _userDetails['c_category'] == 'Gold'
          ? Color(0xFFFFD700)
          : Color(0xFFC0C0C0),
      ),
     ],
    ),
   ),
  ),
),

SizedBox(height: 24),

// Payment History Section
Text(
 'Payment History',
 style: TextStyle(
  fontSize: 18,
  fontWeight: FontWeight.bold,
 ),
),
SizedBox(height: 12),

_payments.isEmpty
   ? Card(
      child: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Center(
        child: Column(
         children: [
```

```dart
                    Icon(
                      Icons.receipt_long,
                      size: 48,
                      color: Colors.grey[400],
                    ),
                    SizedBox(height: 16),
                    Text(
                      'No payments found',
                      style: TextStyle(
                        fontSize: 16,
                        color: AppTheme.textSecondary,
                      ),
                    ),
                  ],
                ),
              ),
            ),
          )
        : ListView.builder(
            shrinkWrap: true,
            physics: NeverScrollableScrollPhysics(),
            itemCount: _payments.length,
            itemBuilder: (context, index) {
              final payment = _payments[index];
              return Card(
                margin: EdgeInsets.only(bottom: 8),
                child: ListTile(
                  leading: CircleAvatar(
                    backgroundColor: AppTheme.success.withOpacity(0.1),
                    child: Icon(
                      Icons.payment,
                      color: AppTheme.success,
                      size: 20,
                    ),
                  ),
                  title: Text(
                    'Amount: ₹${payment['amount']}',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  subtitle: Text('Month: ${payment['p_month']}'),
                  trailing: Icon(Icons.check_circle, color: AppTheme.success),
                ),
              );
            },
          ),

SizedBox(height: 24),

// Quick Actions
Text(
  'Quick Actions',
  style: TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
  ),
),
SizedBox(height: 12),

Row(
```

```
       children: [
        Expanded(
         child: _buildActionCard(
          'Make Payment',
          Icons.payment,
          AppTheme.success,
          () {
           Navigator.push(
             context,
             MaterialPageRoute(builder: (context) => PaymentScreen(username: widget.username)),
           );
          },
         ),
        ),
        SizedBox(width: 16),
        Expanded(
         child: _buildActionCard(
          'Notifications',
          Icons.notifications,
          AppTheme.warning,
          () {
           Navigator.push(
             context,
             MaterialPageRoute(builder: (context) => NotificationsScreen(notifications: _notifications)),
           );
          },
         ),
        ),
       ],
      ),
      SizedBox(height: 16),
      Row(
       children: [
        Expanded(
         child: _buildActionCard(
          'View Categories',
          Icons.category,
          AppTheme.info,
          () {
           Navigator.pop(context);
          },
         ),
        ),
        SizedBox(width: 16),
        Expanded(
         child: _buildActionCard(
          'Logout',
          Icons.logout,
          Colors.grey,
          () => _logout(context),  ),),
       ],
      ),

      SizedBox(height: 24),
     ],
    ),
   ),
  );
 }
```

Server.js

```
const express = require("express")
const mongoose = require("mongoose")
const bodyParser = require("body-parser")
const cors = require("cors")
require("dotenv").config()
const languageMiddleware = require("./middleware/language")

const app = express()
const PORT = 4000

app.use(
  cors({
    origin: "*", // Allow all origins (for testing purposes)
    methods: ["GET", "POST", "DELETE", "PUT"], // Added PUT for restoring users
  }),
)
app.use(bodyParser.json())
app.use(languageMiddleware) // Add language middleware

// Define schemas first
const userSchema = new mongoose.Schema({
  _id: Number,
  c_name: String,
  c_vill: String,
  c_category: String,
  phone: String,
  language: { type: String, default: "en" }, // Add language preference to user schema
  isDeleted: { type: Boolean, default: false }, // Add deletion status
  deletedAt: { type: Date, default: null }, // Add deletion timestamp
})

const paymentSchema = new mongoose.Schema({
  c_id: Number,
  p_date: Date,
  p_month: String,
  amount: Number,
  transactionId: String,
  status: { type: String, enum: ["pending", "approved", "rejected"], default: "pending" },
})
```

33

```javascript
const villageSchema = new mongoose.Schema({
  v_name: String,
})

const notificationSchema = new mongoose.Schema({
  userId: Number,
  message: String,
  createdAt: { type: Date, default: Date.now },
  read: { type: Boolean, default: false },
})

const transactionSchema = new mongoose.Schema({
  transactionId: String,
  userId: Number,
  month: String,
  amount: Number,
  status: { type: String, enum: ["pending", "approved", "rejected"], default: "pending" },
})

// Create models
const User = mongoose.model("User", userSchema)
const Payment = mongoose.model("Payment", paymentSchema)
const Village = mongoose.model("Village", villageSchema)
const Notification = mongoose.model("Notification", notificationSchema)
const Transaction = mongoose.model("Transaction", transactionSchema)

// Define all route handlers
app.post("/register_device", async (req, res) => {
  const { userId, deviceToken } = req.body
  try {
    const user = await User.findById(userId)
    if (!user) {
      return res.status(404).json({ error: res.locals.t("userNotFound") })
    }

    user.deviceToken = deviceToken
    await user.save()

    res.json({ message: res.locals.t("deviceRegistered") })
  } catch (error) {
```

```
      res.status(500).json({ error: error.message })
    }
  })

async function checkAndCreateVillage(req, res, next) {
  const { c_vill } = req.body
  try {
    const village = await Village.findOne({ v_name: c_vill })
    if (!village) {
      await Village.create({ v_name: c_vill })
    }
    next()
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
}

app.post("/add_user", checkAndCreateVillage, async (req, res) => {
  const { userId, c_name, c_vill, c_category, phone, language = "en" } = req.body
  try {
    const existingUser = await User.findById(userId)
    if (existingUser) {
      return res.status(400).json({ error: res.locals.t("userIdExists") })
    }
    const user = new User({
      _id: userId,
      c_name,
      c_vill,
      c_category,
      phone,
      language,
    })
    await user.save()

    // Get user's language preference for notification
    const userLang = language || req.lang

    const { translate } = require("./i18n/i18n")
    const welcomeMessage = translate("welcomeMessage", userLang, { name: c_name })

    const notification = new Notification({
```

35

```javascript
      userId: userId,
      message: welcomeMessage,
    })
    await notification.save()

    res.json({ message: res.locals.t("userAdded"), data: user })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.get("/pending_transactions", async (req, res) => {
  try {
    const transactions = await Transaction.find({ status: "pending" })
    res.json(transactions)
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.post("/add_payments", async (req, res) => {
  const { c_id, p_month, amount } = req.body
  try {
    const user = await User.findById(c_id)
    if (!user) {
      return res.status(404).json({ error: res.locals.t("userNotFound") })
    }
    const payment = new Payment({
      c_id,
      p_date: new Date(),
      p_month,
      amount,
    })
    await payment.save()

    // Get user's language preference for notification
    const userLang = user.language || "en"

    const { translate } = require("./i18n/i18n")
    const notificationMessage = translate("paymentRecorded", userLang, { amount, month: p_month })
```

```javascript
    // Create in-app notification
    const notification = new Notification({
      userId: c_id,
      message: notificationMessage,
    })
    await notification.save()

    res.json({ message: res.locals.t("paymentAdded"), data: payment })
  } catch (error) {
  res.status(500).json({ error: error.message })
  }
})

app.get("/inactive_customers", async (req, res) => {
 try {
   // Get all active users
   const allUsers = await User.find({ isDeleted: { $ne: true } })

   // Get current date and calculate 2 months ago
   const currentDate = new Date()
   const twoMonthsAgo = new Date()
   twoMonthsAgo.setMonth(currentDate.getMonth() - 2)

   const inactiveUsers = []

   // For each user, check their last payment
   for (const user of allUsers) {
     // Get the latest payment for this user
     const latestPayment = await Payment.findOne({ c_id: user._id }).sort({ p_date: -1 }).limit(1)

     // If no payment or last payment is older than 2 months, consider inactive
     if (!latestPayment || new Date(latestPayment.p_date) < twoMonthsAgo) {
      inactiveUsers.push({
        id: user._id,
        name: user.c_name,
        phone: user.phone,
        village: user.c_vill,
        category: user.c_category,
        lastPaymentMonth: latestPayment ? latestPayment.p_month : "Never",
        lastPaymentAmount: latestPayment ? latestPayment.amount : 0,
      })
```

37

```
      }
    }

    // Sort by ID
    inactiveUsers.sort((a, b) => a.id - b.id)

    res.json({
      count: inactiveUsers.length,
      inactiveUsers,
    })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

// Modified to soft delete users instead of permanently deleting them
app.delete("/delete_user/:userId", async (req, res) => {
  const userId = Number.parseInt(req.params.userId)
  try {
    // First check if the user exists
    const user = await User.findById(userId)
    if (!user) {
      return res.status(404).json({ error: res.locals.t("userNotFound") })
    }

    if (user.isDeleted) {
      return res.status(400).json({ error: res.locals.t("userAlreadyDeleted") })
    }

    // Soft delete the user
    user.isDeleted = true
    user.deletedAt = new Date()
    await user.save()

    res.json({ message: res.locals.t("userMovedToTrash", { userId }) })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.post("/login", async (req, res) => {
```

```
  const { username, password } = req.body
  try {
    if (username === process.env.ADMIN_USERNAME && password === process.env.ADMIN_PASSWORD) {
      return res.json({ success: true, isAdmin: true })
    }

    const user = await User.findOne({ _id: username, phone: password, isDeleted: { $ne: true } })
    if (user) {
      res.json({ success: true, isAdmin: false, userId: user._id, language: user.language || "en" })
    } else {
      res.status(401).json({ error: res.locals.t("invalidCredentials") })
    }
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.get("/notifications/:userId", async (req, res) => {
  const userId = Number.parseInt(req.params.userId)
  try {
    const notifications = await Notification.find({ userId }).sort({ createdAt: -1 })
    res.json(notifications)
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.get("/check_payment_status", async (req, res) => {
  const userId = Number.parseInt(req.query.userId)
  const month = req.query.month

  try {
    // Check if there's an approved payment for this user and month
    const approvedPayment = await Payment.findOne({
      c_id: userId,
      p_month: month,
    })

    // Check if there's a pending transaction for this user and month
    const pendingTransaction = await Transaction.findOne({
      userId: userId,
```

```
      month: month,
      status: "pending",
    })

    // User has already paid if either an approved payment exists or a pending transaction exists
    const isPaid = !!(approvedPayment || pendingTransaction)

    res.json({
      userId,
      month,
      isPaid,
      hasApprovedPayment: !!approvedPayment,
      hasPendingTransaction: !!pendingTransaction,
    })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.post("/request_payment", async (req, res) => {
  const { userId, month, amount, transactionId } = req.body
  try {
    // Get user for language preference
    const user = await User.findById(userId)
    if (!user) {
      return res.status(404).json({ error: res.locals.t("userNotFound") })
    }

    if (user.isDeleted) {
      return res.status(400).json({ error: res.locals.t("userDeleted") })
    }

    const userLang = user.language || "en"
    const { translate } = require("./i18n/i18n")

    // Check if payment for this month already exists
    const existingPayment = await Payment.findOne({ c_id: userId, p_month: month })
    if (existingPayment) {
      return res.status(400).json({ error: res.locals.t("paymentExists") })
    }
```

```javascript
    // Check if there's a pending transaction for this month
    const pendingTransaction = await Transaction.findOne({
      userId: userId,
      month: month,
      status: "pending",
    })

    if (pendingTransaction) {
     return res.status(400).json({ error: res.locals.t("pendingPaymentExists") })
    }

    // Check if transaction ID already exists
    const existingTransaction = await Transaction.findOne({ transactionId })
    if (existingTransaction) {
     return res.status(400).json({ error: res.locals.t("transactionIdExists") })
    }

    const transaction = new Transaction({
      transactionId,
      userId,
      month,
      amount,
      status: "pending",
    })
    await transaction.save()

    // Create in-app notification
    const notificationMessage = translate("paymentRequestPending", userLang, { amount, month })
    const notification = new Notification({
      userId,
      message: notificationMessage,
    })
    await notification.save()

    res.json({ message: res.locals.t("paymentRequestSubmitted"), data: transaction })
  } catch (error) {
   res.status(500).json({ error: error.message })
  }
})

app.post("/approve_payment", async (req, res) => {
```

```javascript
  const { transactionId } = req.body
  try {
    const transaction = await Transaction.findOne({ transactionId })
    if (!transaction) {
      return res.status(404).json({ error: res.locals.t("transactionNotFound") })
    }

    transaction.status = "approved"
    await transaction.save()

    const payment = new Payment({
      c_id: transaction.userId,
      p_date: new Date(),
      p_month: transaction.month,
      amount: transaction.amount,
      transactionId: transaction.transactionId,
    })
    await payment.save()

    // Get user for language preference
    const user = await User.findById(transaction.userId)
    const userLang = user ? user.language : "en"
    const { translate } = require("./i18n/i18n")

    // Create in-app notification
    const notificationMessage = translate("paymentApprovedNotification", userLang, {
      amount: transaction.amount,
      month: transaction.month,
    })

    const notification = new Notification({
      userId: transaction.userId,
      message: notificationMessage,
    })
    await notification.save()

    res.json({ message: res.locals.t("paymentApproved"), data: payment })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})
```

```javascript
app.post("/reject_payment", async (req, res) => {
  const { transactionId } = req.body
  try {
    const transaction = await Transaction.findOne({ transactionId })
    if (!transaction) {
      return res.status(404).json({ error: res.locals.t("transactionNotFound") })
    }

    transaction.status = "rejected"
    await transaction.save()

    // Get user for language preference
    const user = await User.findById(transaction.userId)
    const userLang = user ? user.language : "en"
    const { translate } = require("./i18n/i18n")

    // Create in-app notification
    const notificationMessage = translate("paymentRejectedNotification", userLang, {
      amount: transaction.amount,
      month: transaction.month,
    })

    const notification = new Notification({
      userId: transaction.userId,
      message: notificationMessage,
    })
    await notification.save()

    res.json({ message: res.locals.t("paymentRejected"), data: transaction })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

// Add endpoint to update user language preference
app.post("/update_language", async (req, res) => {
  const { userId, language } = req.body
  try {
    const user = await User.findById(userId)
    if (!user) {
```

```
      return res.status(404).json({ error: res.locals.t("userNotFound") })
    }

    user.language = language
    await user.save()

    res.json({ message: res.locals.t("languageUpdated"), data: { userId, language } })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})


// Get available languages
app.get("/languages", async (req, res) => {
  try {
    res.json({
      languages: [
        { code: "en", name: "English" },
        { code: "ta", name: "Tamil" },
        { code: "te", name: "Telugu" },
        { code: "kn", name: "Kannada" },
        { code: "hi", name: "Hindi" },
      ],
    })
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})


// Connect to MongoDB and start server only after connection is established
async function startServer() {
  try {
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      bufferCommands: false, // Disable mongoose buffering
      serverSelectionTimeoutMS: 5000, // Keep trying to send operations for 5 seconds
      socketTimeoutMS: 45000, // Close sockets after 45 seconds of inactivity
    })

    console.log("MongoDB connected")
```

```javascript
    // Only start the server after successful connection
    app.listen(PORT, () => {
      console.log(`Server running on port http://localhost:${PORT}`)
    })
  } catch (err) {
    console.error("MongoDB connection error:", err)
    process.exit(1) // Exit with failure
  }
}

const { translateText, translateObject } = require('./i18n/translation-service');

// Endpoint to translate a single text
app.post("/translate", async (req, res) => {
  const { text, targetLang, sourceLang = 'en' } = req.body;

  if (!text || !targetLang) {
    return res.status(400).json({ error: "Text and target language are required" });
  }

  try {
    const translatedText = await translateText(text, targetLang, sourceLang);
    res.json({ translatedText });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Endpoint to translate multiple texts at once
app.post("/translate-batch", async (req, res) => {
  const { texts, targetLang, sourceLang = 'en' } = req.body;

  if (!texts || !Array.isArray(texts) || !targetLang) {
    return res.status(400).json({ error: "Array of texts and target language are required" });
  }

  try {
    const translatedTexts = await Promise.all(
      texts.map(text => translateText(text, targetLang, sourceLang))
    );
```

```javascript
    res.json({ translatedTexts });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Endpoint to translate an entire object
app.post("/translate-object", async (req, res) => {
  const { object, targetLang, sourceLang = 'en' } = req.body;

  if (!object || typeof object !== 'object' || !targetLang) {
    return res.status(400).json({ error: "Object and target language are required" });
  }

  try {
    const translatedObject = await translateObject(object, targetLang, sourceLang);
    res.json({ translatedObject });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Start the server
startServer()
```

APPENDIX 2

SNAPSHOTS
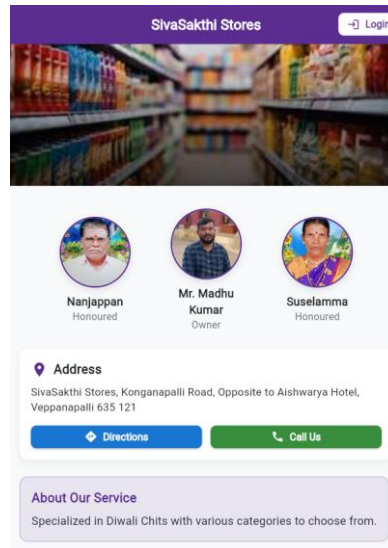
Figure A2.1 Home Page
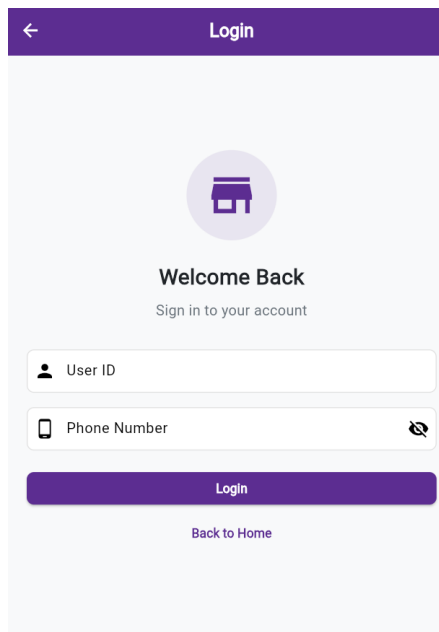
Figure A2.2 Login Page
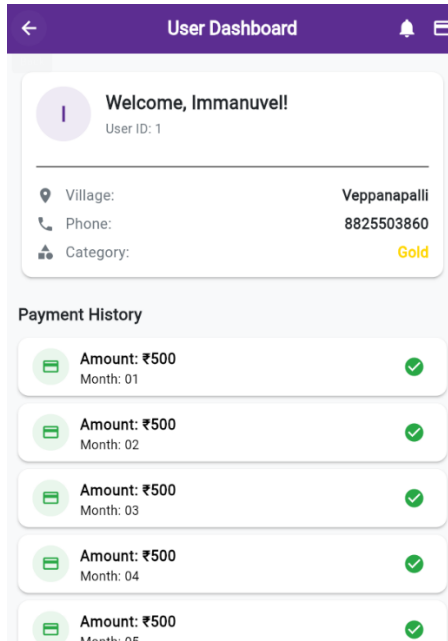
Figure A2.3 User Dashboard Page
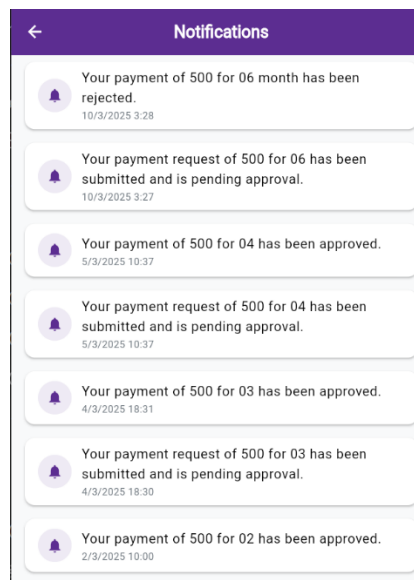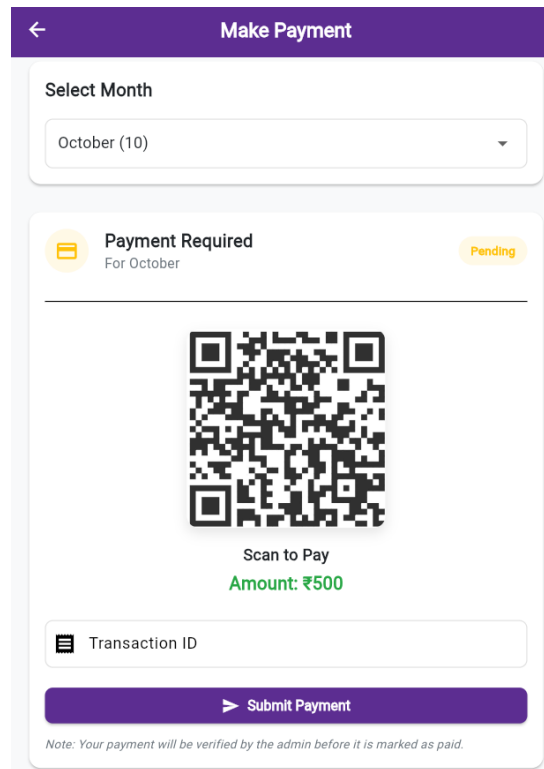

Figure A2.4 Notification Page

Figure A2.5 Request Payment Page
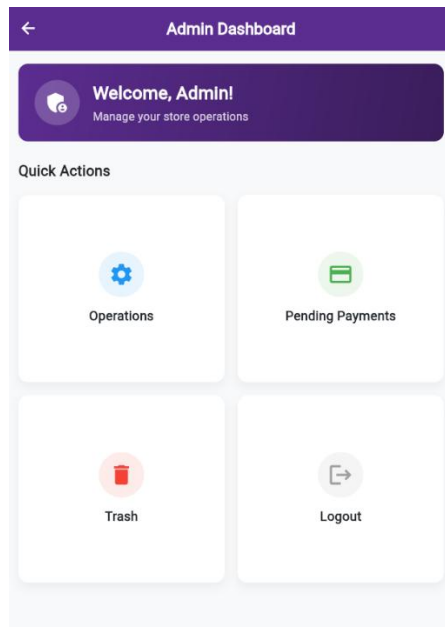
Figure A2.6 Admin Dashboard Page

Figure A2.7 Admin Operations Page

REFERENCES

1. https://docs.flutter.dev/

2. https://expressjs.com/

3. https://www.mongodb.com/docs/

4. https://vercel.com/docs/deployments

GITHUB

1. https://github.com/Immanuvel1207/Sivasakthi/