

**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

NOVEMBER 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638060

NOVEMBER 2024

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

NOVEMBER 2024

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. The 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 her 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.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE No
	ABSTRACT	iv
	ACKNOWLEDGEMENT	v
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	x
1	INTRODUCTION	1
	1.1 INTRODUCTION	1
	1.2 OBJECTIVE	1
2	SYSTEM SPECIFICATIONS	2
	2.1 HARDWARE REQUIREMENTS	2
	2.2 SOFTWARE REQUIREMENTS	2
	2.3 SOFTWARE DESCRIPTION	3
	2.3.1 Visual Studio Code	3
	2.3.2 JavaScript	3
	2.3.3 Node.js	3
	2.3.4 Express.js	3
	2.3.5 MongoDB & Mongoose	4
	2.3.6 Flutter	4
3	SYSTEM DESIGN	5
	3.1 USE CASE DIAGRAM	5

3.2	CLASS DIAGRAM	6
3.3	SEQUENCE DIAGRAM	7
3.4	ACTIVITY DIAGRAM	8
3.5	DATABASE DESIGN	9
3.6	MODULES DESCRIPTION	11
3.6.1	Authentication Module	11
3.6.2	User Management Module	11
3.6.3	Payment Management Module	12
3.6.4	Village Management Module	13
3.6.5	Notification Module	13
3.6.6	Trash Management Module	14
3.6.7	Reporting Module	14
3.6.8	Device Management Module	15
4	SYSTEM TESTING	16
4.1	SYSTEM TESTING	16
4.2	UNIT TESTING	16
4.3	MODULE TESTING	17
4.4	INTEGRATION TESTING	17
4.5	VALIDATION TESTING	18
5	RESULTS	19
6	CONCLUSION AND FUTURE WORK	20
	APPENDIX 1- CODING	21

APPENDIX 2- SNAPSHOTS	115
REFERENCES	119

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
3.1	Use Case Diagram	5
3.2	Class Diagram	6
3.3	Sequence Diagram	7
3.4	Activity Diagram	8
A2.1	Home Page	115
A2.2	Login Page	115
A2.3	User Dashboard Page	116
A2.4	Notification Page	116
A2.5	Request Payment Page	117
A2.6	Admin Dashboard Page	117
A2.7	Admin Operations Page	118

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.

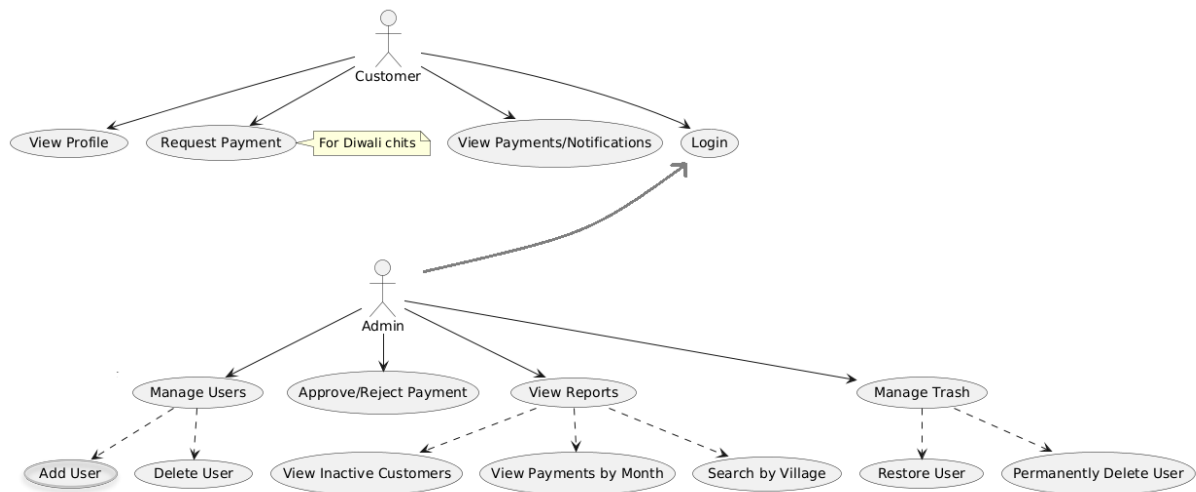


Fig 3.1 Use Case Diagram

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.

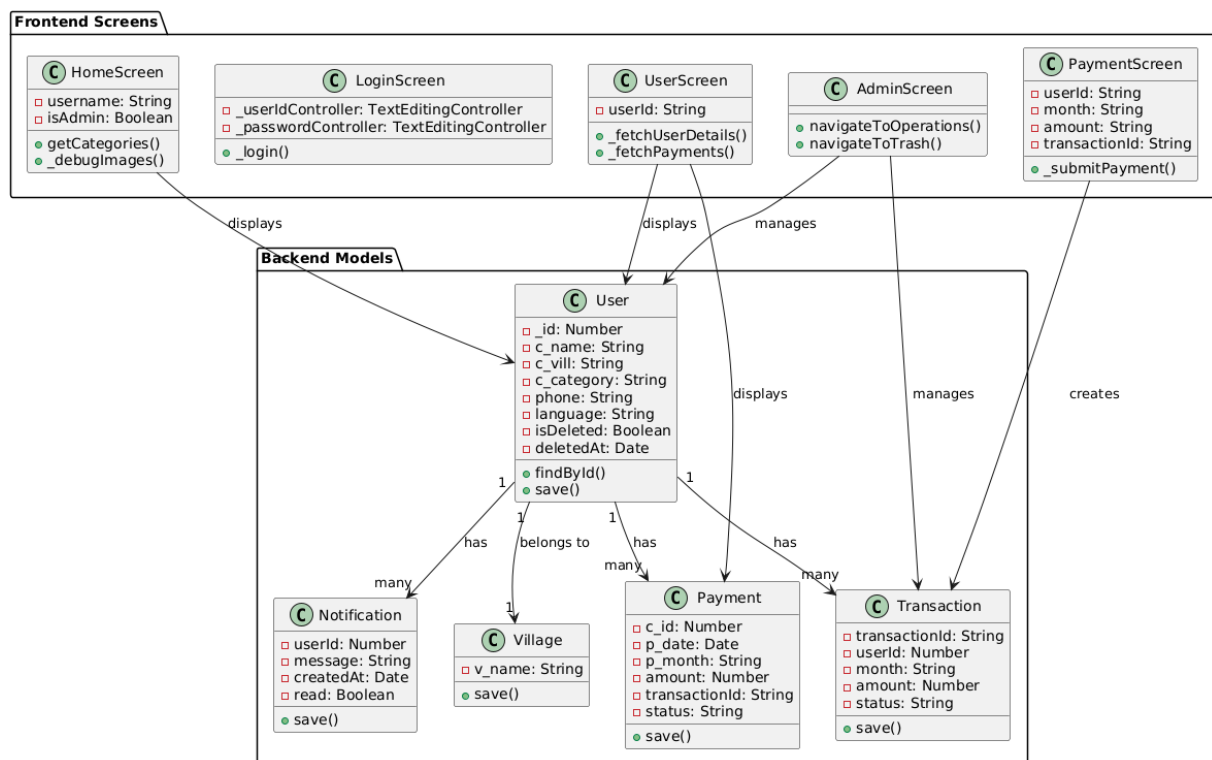


Fig 3.2 Class Diagram

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.

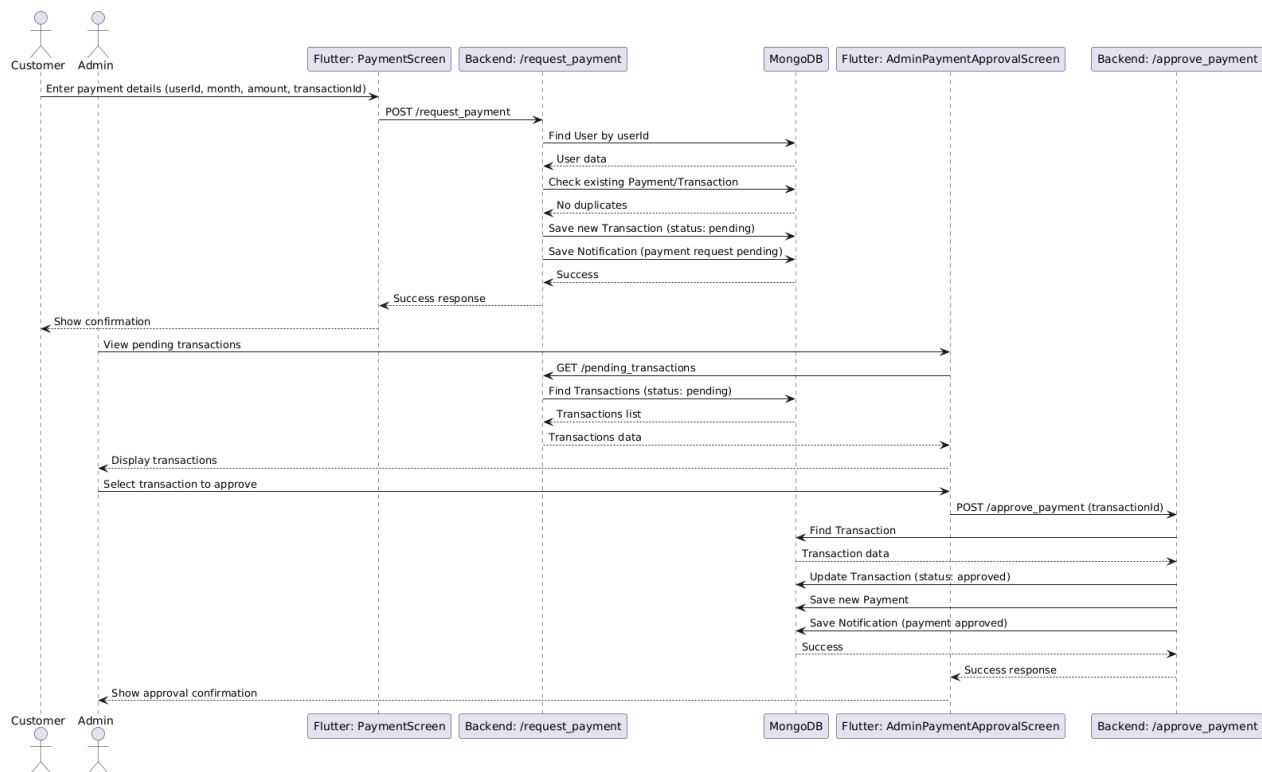


Fig 3.3 Sequence Diagram

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.

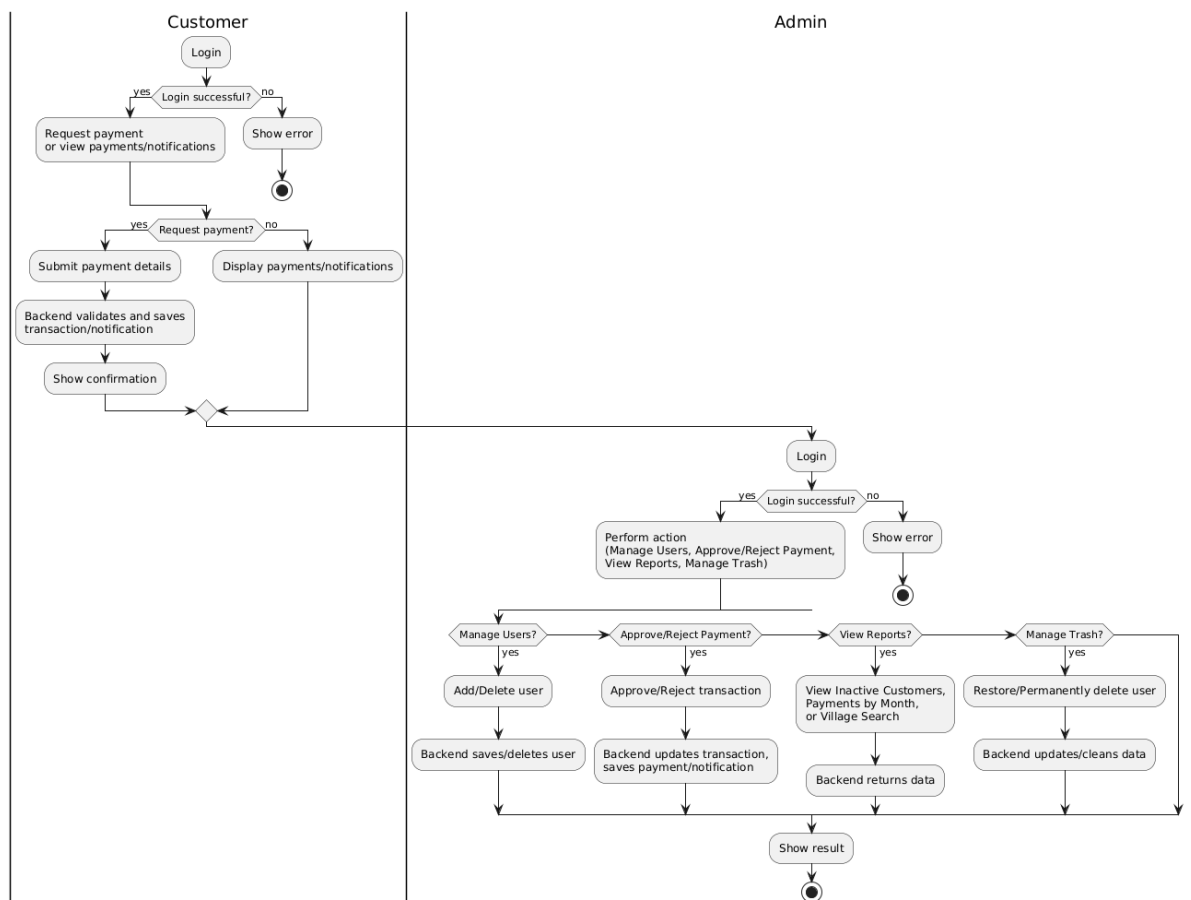


Fig 3.4 Activity Diagram

3.5 DATABASE DESIGN

3.5.1 USER SCHEMA

Collection: users

Fields:

- _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

Collection: payments

Fields:

- _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

Collection: villages

Fields:

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

3.5.4 NOTIFICATION SCHEMA

Collection: notifications

Fields:

- _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

Collection: transactions

Fields:

- _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

Purpose: Manages user and admin authentication to secure access to the system.

Key Functionalities:

- Authenticates customers using user ID and phone number.
- Authenticates admins using environment-stored credentials.

Endpoints:

- POST /login: Authenticates users/admins, returns success status and user details.

Dependencies:

- Schemas: User (for user authentication).
- Environment: ADMIN_USERNAME, ADMIN_PASSWORD (for admin authentication).

Contribution:

- Enables secure access to the Flutter app's features (e.g., LoginScreen).
- Supports role-based access (admin vs. customer).
- Provides language preference for personalized notifications.

3.6.2 User Management Module

Purpose: Handles creation, retrieval, updating, and soft deletion of user records.

Key Functionalities:

- Adds new users with village creation if needed.
- Retrieves user details by ID or searches by name, category, or village.
- Lists all active users with payment counts.
- Soft deletes users (moves to trash).

Endpoints:

- POST /add_user: Creates a user and adds village if it doesn't exist.
- GET /find_user: Retrieves a user by ID.
- GET /find_all_users: Lists active users with payment counts.
- GET /search_users: Searches users by name, category, or village.
- DELETE /delete_user/:userId: Soft deletes a user.

Dependencies:

- Schemas: User, Village, Notification (for welcome messages).
- Middleware: checkAndCreateVillage (for village creation).

Contribution:

- Powers user management in AdminOperationsScreen (Flutter app).
- Supports village-based categorization for store operations.
- Integrates with Notification Module for user creation alerts.

3.6.3 Payment Management Module

Purpose: Manages payment requests, approvals, rejections, and payment history.

Key Functionalities:

- Allows customers to submit payment requests.
- Enables admins to approve or reject payment requests.
- Records payments and checks payment status for specific months.
- Tracks total payments per user.

Endpoints:

- POST /add_payments: Adds a payment for a user.
- POST /request_payment: Submits a payment request (creates Transaction).
- POST /approve_payment: Approves a transaction, creates Payment.
- POST /reject_payment: Rejects a transaction.
- GET /pending_transactions: Lists pending transactions.
- GET /find_payments: Retrieves payments for a user (optionally by month).
- GET /check_payment_status: Checks if a user has paid for a month.
- GET /total_amount_paid: Calculates total payments for a user.

Dependencies:

- Schemas: User, Payment, Transaction, Notification.

Contribution:

- Drives PaymentScreen and AdminPaymentApprovalScreen in the Flutter app.
- Supports Diwali chit payment workflows with approval process.
- Integrates with Notification Module for payment status updates.

3.6.4 Village Management Module

Purpose: Manages village records and supports village-based user searches.

Key Functionalities:

- Automatically creates villages when adding users.
- Lists all villages.
- Retrieves users by village with payment counts.

Endpoints:

- GET /get_all_villages: Lists all villages.
- GET /search_by_village: Lists users in a specific village.

Dependencies:

- Schemas: Village, User, Payment (for payment counts).
- Middleware: checkAndCreateVillage (used in /add_user).

Contribution:

- Enables village-based filtering in AdminOperationsScreen.
- Supports store's geographic organization of customers.

3.6.5 Notification Module

Purpose: Manages in-app notifications for users, such as payment updates and welcomes.

Key Functionalities:

- Creates notifications for user actions (e.g., payment requests, approvals).
- Retrieves notifications for a user, sorted by creation date.
- Supports multilingual notifications based on user language.

Endpoints:

- GET /notifications/:userId: Retrieves notifications for a user.

Dependencies:

- Schemas: Notification, User (for language preference).

Contribution:

- Powers NotificationsScreen in the Flutter app.
- Enhances user engagement with timely, personalized updates.

- Integrates with other modules (e.g., Payment, User) for event-driven notifications.

3.6.6 Trash Management Module

Purpose: Handles soft deletion, restoration, and permanent deletion of users.

Key Functionalities:

- Soft deletes users, marking them as deleted with a timestamp.
- Lists soft-deleted users (trash).
- Restores soft-deleted users to active status.
- Permanently deletes users and their related data (Payments, Notifications, Transactions).

Endpoints:

- GET /deleted_users: Lists soft-deleted users with payment counts.
- PUT /restore_user/:userId: Restores a soft-deleted user.
- DELETE /permanent_delete_user/:userId: Permanently deletes a user and related data.

Dependencies:

- Schemas: User, Payment, Notification, Transaction.

Contribution:

- Supports TrashScreen in the Flutter app for managing deleted users.
- Ensures data recovery and cleanup for store records.

3.6.7 Reporting Module

Purpose: Generates reports on customer activity, payments, and inactivity.

Key Functionalities:

- Lists paid/unpaid users for a specific month.
- Identifies inactive customers (no payments in last 2 months).
- Supports village-based user searches (overlaps with Village Management).

Endpoints:

- GET /view_payments_by_month: Lists paid/unpaid users for a month.
- GET /inactive_customers: Lists users with no recent payments.
- GET /search_by_village: Lists users by village (shared with Village Management).

Dependencies:

- Schemas: User, Payment.

Contribution:

- Provides analytics for admins in AdminOperationsScreen.
- Helps track store performance and customer engagement.

3.6.8 Device Management Module

Purpose: Registers device tokens for potential push notifications.

Key Functionalities:

- Associates a device token with a user for future push notifications.

Endpoints:

- POST /register_device: Registers a device token for a user.

Dependencies:

- Schemas: User (though deviceToken field is missing in schema).

Contribution:

- Lays groundwork for push notifications in the Flutter app (not fully implemented).
- Enhances user engagement by enabling future real-time alerts.

CHAPTER 4

SYSTEM TESTING

4.1 SYSTEM TESTING

The primary goal of SivaSakthi Stores is to deliver a high-quality retail management system with robust features for managing Diwali chit payments and user records. To ensure reliability, security, and performance, comprehensive system testing is conducted to identify and resolve defects under various conditions. System testing verifies that the software meets functional and non-functional requirements by detecting errors, gaps, or missing features. It is the final step before deployment, ensuring the product is market-ready. Testing involves evaluating, analyzing, and validating system components to guarantee customer satisfaction, cost efficiency, and operational reliability. The process includes four key phases:

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

4.1 UNIT TESTING

Unit testing focuses on the smallest testable components of the system, such as individual functions or methods in the Node.js backend and Flutter app. Developers perform unit testing during coding to ensure each unit operates correctly in isolation. For example, in the backend, functions like user creation (`add_user`), payment validation (`request_payment`), and notification generation were tested using temporary test scripts. In the Flutter app, widgets (e.g., `LoginScreen`, `login` method) and API calls were tested with mock data. This granular approach provided detailed insights into code behavior, enabling early detection and correction of logical errors.

4.2 MODULE TESTING

Each module of the SivaSakthi Stores system was rigorously tested to ensure stable functionality and accurate data handling. The Authentication Module was validated for secure login using user ID/phone and admin credentials, ensuring proper access control. The User Management Module was tested for adding, updating, deleting, and searching users, including village creation and payment count integration. The Payment Management Module was checked for payment requests, approvals, rejections, and history retrieval, ensuring accurate transaction processing. The Notification Module was validated for generating and displaying multilingual notifications. The Trash Management Module was tested for soft deletion, restoration, and permanent deletion of users. The Village Management Module ensured correct village creation and user searches. The Reporting Module was verified for generating payment status and inactive customer reports. The Internationalization Module was tested for language updates and translations. The Device Management Module was checked for registering device tokens. Testing confirmed that each module performed reliably and delivered a seamless user experience.

4.3 INTEGRATION TESTING

Integration testing ensured seamless interaction between modules in the SivaSakthi Stores system. The Authentication Module was integrated with User Management and Payment Management to verify that user/admin login correctly enabled access to respective features (e.g., AdminOperationsScreen, PaymentScreen). The User Management Module was tested with Payment and Notification Modules to ensure user creation triggered welcome notifications and linked to payment records. The Payment Management Module was integrated with Transaction and Notification Modules to confirm that payment requests, approvals, and rejections updated transaction statuses and generated notifications. The Village Management Module was tested with User Management to ensure village-based searches reflected accurate user data. The Reporting Module was integrated with Payment and User Management to validate payment status and inactive customer reports. The Trash Management Module was tested with Payment and Notification Modules to ensure permanent deletions cleaned related data. Data flow across modules was verified for consistency, synchronization, and accuracy, ensuring the system operated as a cohesive platform.

4.4 VALIDATION TESTING

Validation testing confirmed that the SivaSakthi Stores system met all functional and usability requirements. The Authentication Module validated user inputs (e.g., user ID, phone) for correct formats, preventing unauthorized access. The User Management Module ensured only authorized admins could add, edit, or soft-delete users, with checks for unique IDs and valid village names. The Payment Management Module validated payment requests for user existence, unique transaction IDs, and non-duplicate months, ensuring accurate processing. The Notification Module was tested to deliver translated messages based on user language preferences. The Trash Management Module validated restoration and permanent deletion, ensuring data integrity during cleanup. The Reporting Module was verified to generate accurate payment and inactivity reports based on predefined criteria. The Internationalization Module ensured seamless language updates and translations. Validation testing confirmed that the system functions as intended, maintains data integrity, and provides a reliable, user-friendly experience for managing store operations.

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

```
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(0xFFFFF6B35);
  static const Color backgroundColor = Color(0xFFFFF8F9FA);
  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(0xFFFFFC107);
  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),

```



```

        borderSide: BorderSide(color: danger, width: 1),
      ),
      contentPadding: EdgeInsets.symmetric(horizontal: 16, vertical: 16),
    ),
    textTheme: TextTheme(
      displayLarge: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      displayMedium: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      displaySmall: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      headlineMedium: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      headlineSmall: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      titleLarge: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
      titleMedium: TextStyle(color: textPrimary),
      titleSmall: TextStyle(color: textSecondary),
      bodyLarge: TextStyle(color: textPrimary),
      bodyMedium: TextStyle(color: textPrimary),
      bodySmall: TextStyle(color: textSecondary),
      labelLarge: TextStyle(color: textPrimary, fontWeight: FontWeight.bold),
    ),
    tabBarTheme: TabBarTheme(
      labelColor: primaryColor,
      unselectedLabelColor: textSecondary,
      indicator: BoxDecoration(
        border: Border(
          bottom: BorderSide(color: primaryColor, width: 2),
        ),
      ),
    ),
  ),
);
}

class SplashScreen extends StatefulWidget {
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      duration: const Duration(seconds: 2),
      vsync: this,
    );
    _animation = CurvedAnimation(
      parent: _controller,
      curve: Curves.easeInOut,
    );
    _controller.forward();
    checkLoginStatus();
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void checkLoginStatus() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    String? username = prefs.getString('username');
    bool? isAdmin = prefs.getBool('isAdmin');
  }
}

```

```

Timer(const Duration(seconds: 2), () {
  Navigator.pushReplacement(
    context,
    MaterialPageRoute(builder: (context) => HomeScreen(username: username, isAdmin: isAdmin)),
  );
});
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
          colors: [
            AppTheme.primaryColor,
            Color(0xFF3A1D5B),
          ],
        ),
      ),
    child: Center(
      child: FadeTransition(
        opacity: _animation,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // Logo with error handling
            Container(
              height: 150,
              width: 150,
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(75),
                boxShadow: [
                  BoxShadow(
                    color: Colors.black.withOpacity(0.2),
                    blurRadius: 10,
                    spreadRadius: 2,
                  ),
                ],
              ),
            ),
            child: ClipRRect(
              borderRadius: BorderRadius.circular(75),
              child: Image.asset(
                'assets/jerry.gif',
                fit: BoxFit.cover,
                errorBuilder: (context, error, stackTrace) {
                  print('Error loading splash image: $error');
                  return Container(
                    color: Colors.white,
                    child: Icon(
                      Icons.store,
                      size: 80,
                      color: AppTheme.primaryColor,
                    ),);} ),);
          ),
          const SizedBox(height: 24),
          Text(

```

```

        'SivaSakthi Stores',
        style: TextStyle(
          fontSize: 28,
          fontWeight: FontWeight.bold,
          color: Colors.white,
          letterSpacing: 1.2,
        ),
      ),
      const SizedBox(height: 8),
      Text(
        'Diwali Chits & More',
        style: TextStyle(
          fontSize: 16,
          color: Colors.white.withOpacity(0.8),
          letterSpacing: 0.5,
        ),
      ),
    ),
    const SizedBox(height: 32),
    SizedBox(
      width: 40,
      height: 40,
      child: CircularProgressIndicator(
        valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
        strokeWidth: 3,
      ),
    ),
  ], ), ), ), ); }
}

```

```

class HomeScreen extends StatefulWidget {
  final String? username;
  final bool? isAdmin;

```

```

  HomeScreen({this.username, this.isAdmin});

```

```

  @override
  _HomeScreenState createState() => _HomeScreenState();
}

```

```

class _HomeScreenState extends State<HomeScreen> {
  final String shopName = "SivaSakthi Stores";
  final String ownerName = "Mr. Madhu Kumar";
  final String phoneNumber = "+91 9080660749";
  final String serviceDescription = "Specialized in Diwali Chits with various categories to choose from.";
  final String address = "SivaSakthi Stores, Konganapalli Road, Opposite to Aishwarya Hotel, Veppanapalli 635 121";
  final String mapUrl = "https://maps.app.goo.gl/44evAN22mo1KNfTc7";

```

```

  late List<Map<String, dynamic>> categories;
  bool _isLoading = true;

```

```

  @override
  void initState() {
    super.initState();
    categories = getCategories();
    // Add this to debug images after the first frame is rendered
    WidgetsBinding.instance.addPostFrameCallback((_) {
      _debugImages();
      setState(() {
        _isLoading = false;
      });
    });
  }
}

```

```

}

List<Map<String, dynamic>> getCategories() {
  return [
    {
      "title": "Gold Category",
      "amount": "₹500 per month",
      "color": "blue",
      "products": [
        {"name": "Rice", "quantity": "25 Kg", "image": "assets/things/ricejpg.jpg"},
        {"name": "Maida", "quantity": "1 pack", "image": "assets/things/maida.jpg"},
      ],
    },
    {
      "title": "Silver Category",
      "amount": "₹300 per month",
      "color": "yellow",
      "products": [
        {"name": "Rice", "quantity": "5 Kg", "image": "assets/things/ricejpg.jpg"},
        {"name": "Oil", "quantity": "5 liters", "image": "assets/things/oil.jpg"},
      ],
    }
  ];
}

```

```

Future<void> _debugImages() async {
  for (var category in categories) {
    for (var product in category['products']) {
      try {
        await precacheImage(AssetImage(product['image']), context);
      } catch (e) {
        print('Failed to load: ${product['image']}, error: $e');
      }
    }
  }
}

```

```

@override
Widget build(BuildContext context) {
  if (_isLoading) {
    return Scaffold(
      body: Center(
        child: CircularProgressIndicator(
          valueColor: AlwaysStoppedAnimation<Color>(AppTheme.primaryColor),
        ),
      ),
    );
  }
}

```

```

return Scaffold(
  appBar: AppBar(
    title: Text(shopName),
    actions: [
      if (widget.username != null)
        Padding(
          padding: const EdgeInsets.only(right: 8.0),
          child: ElevatedButton.icon(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(

```

```

        builder: (context) => widget.isAdmin == true
          ? AdminScreen()
          : UserScreen(username: widget.username!),
      ),
    );
  },
  icon: Icon(Icons.dashboard, size: 18),
  label: Text("Dashboard"),
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.white,
    foregroundColor: AppTheme.primaryColor,
    elevation: 0,
    padding: EdgeInsets.symmetric(horizontal: 12, vertical: 8),
  ),
),
)
)
else
  Padding(
    padding: const EdgeInsets.only(right: 8.0),
    child: ElevatedButton.icon(
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => LoginScreen()),
        );
      },
      icon: Icon(Icons.login, size: 18),
      label: Text("Login"),
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.white,
        foregroundColor: AppTheme.primaryColor,
        elevation: 0,
        padding: EdgeInsets.symmetric(horizontal: 12, vertical: 8),
      ),
    ),
  ),
),
],
),
body: RefreshIndicator(
  onRefresh: () async {
    setState(() {
      categories = getCategories();
    });
    await _debugImages();
  },
  child: SingleChildScrollView(
    physics: AlwaysScrollableScrollPhysics(),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        // Hero Banner
        Container(
          height: 200,
          width: double.infinity,
          child: Stack(
            children: [
              Image.asset(
                'assets/shop.jpg',
                width: double.infinity,
                height: 200,
                fit: BoxFit.cover,

```

```

errorBuilder: (context, error, stackTrace) {
  return Container(
    height: 200,
    width: double.infinity,
    color: Colors.grey[300],
    child: Icon(Icons.store, size: 80, color: Colors.grey[600]),
  );
},
),

Container(
  decoration: BoxDecoration(
    gradient: LinearGradient(
      begin: Alignment.topCenter,
      end: Alignment.bottomCenter,
      colors: [
        Colors.transparent,
        Colors.black.withOpacity(0.7),
      ],
    ),
  ),
),
Positioned(
  bottom: 20,
  left: 120,
  right: 100,
  top: 0,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        "",
        style: TextStyle(
          fontSize: 24,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
      SizedBox(height: 4),
      Text(
        "",
        style: TextStyle(
          fontSize: 16,
          color: Colors.white.withOpacity(0.9),
        ),
      ),
    ],
  ),
),

// People Section
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [

      SizedBox(height: 16),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          buildPersonCard("Nanjappan", "Honoured", 'assets/father.jpg'),

```



```

    ),
  ],
),
],
),
),
),

// Service Description
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Container(
    padding: EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: AppTheme.primaryColor.withOpacity(0.1),
      borderRadius: BorderRadius.circular(12),
      border: Border.all(color: AppTheme.primaryColor.withOpacity(0.3)),
    ),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          "About Our Service",
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.bold,
            color: AppTheme.primaryColor,
          ),
        ),
        SizedBox(height: 8),
        Text(
          serviceDescription,
          style: TextStyle(
            fontSize: 16,
            height: 1.5,
          ),
        ),
      ],
    ),
  ),
),

// Categories Section
Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        "Our Categories",
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(height: 16),
      ListView.builder(
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        itemCount: categories.length,
        itemBuilder: (context, index) {
          final category = categories[index];

```

```

return Container(
  margin: EdgeInsets.only(bottom: 16),
  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: [
      Container(
        padding: EdgeInsets.all(16),
        decoration: BoxDecoration(
          color: category["title"] == "Gold Category"
            ? Color(0xFFFFD700).withOpacity(0.2)
            : Color(0xFFC0C0C0).withOpacity(0.2),
          borderRadius: BorderRadius.only(
            topLeft: Radius.circular(12),
            topRight: Radius.circular(12),
          ),
        ),
      ),
      child: Row(
        children: [
          Icon(
            category["title"] == "Gold Category"
              ? Icons.star
              : Icons.star_half,
            color: category["title"] == "Gold Category"
              ? Color(0xFFFFD700)
              : Color(0xFFC0C0C0),
            size: 28,
          ),
          SizedBox(width: 12),
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                category["title"],
                style: TextStyle(
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                ),
              ),
              SizedBox(height: 4),
              Text(
                category["amount"],
                style: TextStyle(
                  fontSize: 16,
                  color: AppTheme.textSecondary,
                ),
              ),
            ],
          ),
          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(0xFFFFD700)
    : 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,
                          width: 120,
                          color: Colors.grey[200],
                          child: Icon(Icons.image_not_supported, color: Colors.grey[400]),
                        );
                      },
                    ),
                  ],
                ),
              ),
            ],
          ),
        ),
      ),
    ],
  ),
),

```



```

        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,
        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),
                borderRadius: BorderRadius.circular(4),
              ),
              child: Text(
                product['quantity'],
                style: TextStyle(
                  fontSize: 14,

```



```

}

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

@override
void dispose() {
  _userIdController.dispose();
  _passwordController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Login'),
    ),
    body: SafeArea(
      child: Center(
        child: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.all(24.0),
            child: Form(
              key: _formKey,
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: [
                  // Logo
                  Container(
                    height: 100,
                    width: 100,
                    decoration: BoxDecoration(
                      color: AppTheme.primaryColor.withOpacity(0.1),
                      shape: BoxShape.circle,
                    ),
                  ),
                  child: Icon(
                    Icons.store,
                    size: 60,
                    color: AppTheme.primaryColor,
                  ),
                ],
              ),
            ),
          ),
          child: SizedBox(height: 24),

          // Title
          child: Text(
            'Welcome Back',
            style: TextStyle(
              fontSize: 24,
              fontWeight: FontWeight.bold,
              color: AppTheme.textPrimary,
            ),
            textAlign: TextAlign.center,
          ),
        ),
      ),
    ),
  );
}

```

```

),
 SizedBox(height: 8),
 Text(
   'Sign in to your account',
   style: TextStyle(
     fontSize: 16,
     color: AppTheme.textSecondary,
   ),
   textAlign: TextAlign.center,
 ),
 SizedBox(height: 32),

 // User ID Field
 TextFormField(
   controller: _userIdController,
   decoration: InputDecoration(
     labelText: 'User ID',
     hintText: 'Enter your user ID',
     prefixIcon: Icon(Icons.person),
   ),
   keyboardType: TextInputType.number,
   validator: (value) => value!.isEmpty ? 'Please enter your user ID' : null,
 ),
 SizedBox(height: 16),

 // Password Field
 TextFormField(
   controller: _passwordController,
   decoration: InputDecoration(
     labelText: 'Phone Number',
     hintText: 'Enter your phone number',
     prefixIcon: Icon(Icons.phone_android),
     suffixIcon: IconButton(
       icon: Icon(
         _obscurePassword ? Icons.visibility_off : Icons.visibility,
       ),
       onPressed: () {
         setState(() {
           _obscurePassword = !_obscurePassword;
         });
       },
     ),
   ),
   keyboardType: TextInputType.phone,
   obscureText: _obscurePassword,
   validator: (value) => value!.isEmpty ? 'Please enter your phone number' : null,
 ),
 SizedBox(height: 24),

 // Login Button
 ElevatedButton(
   onPressed: _isLoading ? null : _login,
   child: _isLoading
     ? SizedBox(
         height: 20,
         width: 20,
         child: CircularProgressIndicator(
           strokeWidth: 2,
           valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
         ),
       ),
 )

```

```

        : Text('Login'),
        style: ElevatedButton.styleFrom(
          padding: EdgeInsets.symmetric(vertical: 16),
        ),
      ),
      SizedBox(height: 16),

      // Back to Home
      TextButton(
        onPressed: () {
          Navigator.pop(context);
        },
        child: Text('Back to Home'),
      ),
    ],
  ),
),
),
),
),
),
),
);
}
}

```

```

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

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Admin Dashboard'),
    ),
    body: SafeArea(
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              // Admin Header
              Container(
                padding: EdgeInsets.all(20),
                decoration: BoxDecoration(
                  gradient: LinearGradient(
                    colors: [
                      AppTheme.primaryColor,
                      Color(0xFF3A1D5B),
                    ],
                    begin: Alignment.topLeft,
                    end: Alignment.bottomRight,
                  ),
                borderRadius: BorderRadius.circular(12),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}

```



```

      _buildAdminActionCard(
        context,
        'Operations',
        Icons.settings,
        Colors.blue,
        () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => AdminOperationsScreen()),
          );
        },
      ),
      _buildAdminActionCard(
        context,
        'Pending Payments',
        Icons.payment,
        Colors.green,
        () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => AdminPaymentApprovalScreen()),
          );
        },
      ),
      _buildAdminActionCard(
        context,
        'Trash',
        Icons.delete,
        Colors.red,
        () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => TrashScreen()),
          );
        },
      ),
      _buildAdminActionCard(
        context,
        'Logout',
        Icons.logout,
        Colors.grey,
        () => _logout(context),
      ),
    ],
  ),
  SizedBox(height: 24),

  ],
),
),
),
);
}

Widget _buildAdminActionCard(
  BuildContext context,
  String title,
  IconData icon,

```

```

    Color color,
    VoidCallback onTap,
  ) {
    return InkWell(
      onTap: onTap,
      borderRadius: BorderRadius.circular(12),
      child: 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(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            padding: EdgeInsets.all(12),
            decoration: BoxDecoration(
              color: color.withOpacity(0.1),
              shape: BoxShape.circle,
            ),
            child: Icon(
              icon,
              color: color,
              size: 32,
            ),
          ),
          SizedBox(height: 12),
          Text(
            title,
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
            ),
          ),
        ],
      ),
    );
  }

Widget _buildStatCard(String title, String value, IconData icon, Color color) {
  return Container(
    padding: EdgeInsets.all(16),
    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: [
    Row(
      children: [
        Container(
          padding: EdgeInsets.all(8),
          decoration: BoxDecoration(
            color: color.withOpacity(0.1),
            shape: BoxShape.circle,
          ),
          child: Icon(
            icon,
            color: color,
            size: 20,
          ),
        ),
        SizedBox(width: 8),
        Text(
          title,
          style: TextStyle(
            fontSize: 14,
            color: AppTheme.textSecondary,
          ),
        ),
      ],
    ),
    SizedBox(height: 12),
    Text(
      value,
      style: TextStyle(
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
),
);
}
}

class TrashScreen extends StatefulWidget {
  @override
  _TrashScreenState createState() => _TrashScreenState();
}

class _TrashScreenState extends State<TrashScreen> {
  List<Map<String, dynamic>> _deletedUsers = [];
  bool _isLoading = true;

  @override
  void initState() {
    super.initState();
    _fetchDeletedUsers();
  }

  Future<void> _fetchDeletedUsers() async {
    setState(() {
      _isLoading = true;
    });
  }
}

```

```

try {
  final response = await http.get(
    Uri.parse('https://nanjundeshwara.vercel.app/deleted_users'),
  );

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

Future<void> _restoreUser(int userId) async {
  try {
    final response = await http.put(
      Uri.parse('https://nanjundeshwara.vercel.app/restore_user/$userId'),
    );

    if (response.statusCode == 200) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('User restored successfully'),
          backgroundColor: AppTheme.success,
        ),
      );
      _fetchDeletedUsers(); // Refresh the list
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Failed to restore user: ${response.body}'),
          backgroundColor: AppTheme.danger,
        ),
      );
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('An error occurred: $e'),

```



```

        backgroundColor: AppTheme.danger,
    ),
);
}
}

Future<void> _permanentlyDeleteUser(int userId) async {
  try {
    final response = await http.delete(
      Uri.parse('https://nanjundeshwara.vercel.app/permanent_delete_user/$userId'),
    );

    if (response.statusCode == 200) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('User permanently deleted'),
          backgroundColor: AppTheme.success,
        ),
      );
      _fetchDeletedUsers(); // Refresh the list
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Failed to permanently delete user: ${response.body}'),
          backgroundColor: AppTheme.danger,
        ),
      );
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('An error occurred: $e'),
        backgroundColor: AppTheme.danger,
      ),
    );
  }
}

void _showDeleteConfirmationDialog(int userId, String userName) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Permanently Delete User'),
        content: Text(
          'Are you sure you want to permanently delete $userName (ID: $userId)? This action cannot be undone.'
        ),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: Text('Cancel'),
          ),
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
              _permanentlyDeleteUser(userId);
            },
            child: Text(
              'Delete Permanently',
            ),
          ),
        ],
      );
    },
  );
}

```

```

        style: TextStyle(color: AppTheme.danger),
      ),
    ),
  ],
);
},
);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Trash'),
      backgroundColor: AppTheme.danger,
    ),
    body: _isLoading
      ? Center(child: CircularProgressIndicator())
      : _deletedUsers.isEmpty
        ? Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(
                  Icons.delete_outline,
                  size: 64,
                  color: Colors.grey[400],
                ),
                SizedBox(height: 16),
                Text(
                  'No deleted users found',
                  style: TextStyle(
                    fontSize: 18,
                    color: AppTheme.textSecondary,
                  ),
                ),
              ],
            ),
          )
        : RefreshIndicator(
            onRefresh: _fetchDeletedUsers,
            child: ListView.builder(
              itemCount: _deletedUsers.length,
              itemBuilder: (context, index) {
                final user = _deletedUsers[index];
                final deletedAt = user['deletedAt'] != null
                  ? DateTime.parse(user['deletedAt']).toString().substring(0, 16)
                  : 'Unknown';

                return Card(
                  margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
                  child: Padding(
                    padding: const EdgeInsets.all(12.0),
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Row(
                          children: [
                            Container(
                              padding: EdgeInsets.all(8),
                              decoration: BoxDecoration(

```

```

        color: AppTheme.danger.withOpacity(0.1),
        shape: BoxShape.circle,
      ),
      child: Icon(
        Icons.person_off,
        color: AppTheme.danger,
        size: 20,
      ),
    ),
    SizedBox(width: 12),
    Expanded(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            '${user['c_name']}',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
            ),
          ),
          Text(
            'ID: ${user['_id']}',
            style: TextStyle(
              fontSize: 14,
              color: AppTheme.textSecondary,
            ),
          ),
        ],
      ),
    ),
    SizedBox(height: 12),
    Divider(),
    SizedBox(height: 8),
    Row(
      children: [
        Expanded(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              _buildInfoRow(Icons.location_on, 'Village', '${user['c_vill']}'),
              SizedBox(height: 8),
              _buildInfoRow(Icons.category, 'Category', '${user['c_category']}'),
              SizedBox(height: 8),
              _buildInfoRow(Icons.phone, 'Phone', '${user['phone']}'),
              SizedBox(height: 8),
              _buildInfoRow(Icons.calendar_today, 'Deleted on', deletedAt),
            ],
          ),
        ),
        Column(
          children: [
            IconButton(
              icon: Icon(Icons.restore, color: AppTheme.success),
              onPressed: () => _restoreUser(user['_id']),
              tooltip: 'Restore User',
            ),
            SizedBox(height: 8),
            IconButton(

```

```

        icon: Icon(Icons.delete_forever, color: AppTheme.danger),
        onPressed: () => _showDeleteConfirmationDialog(user['_id'], user['c_name']),
        tooltip: 'Delete Permanently',
      ),
    ],
  ),
  ],
),
],
),
);
},
),
),
);
}

```

```

Widget _buildInfoRow(IconData icon, String label, String value) {

```

```

  return Row(
    children: [
      Icon(
        icon,
        size: 16,
        color: AppTheme.textSecondary,
      ),
      SizedBox(width: 8),
      Text(
        '$label: ',
        style: TextStyle(
          fontSize: 14,
          color: AppTheme.textSecondary,
        ),
      ),
      Expanded(
        child: Text(
          value,
          style: TextStyle(
            fontSize: 14,
            fontWeight: FontWeight.w500,
          ),
          overflow: TextOverflow.ellipsis,
        ),
      ),
    ],
  );
}

```

```

class AdminPaymentApprovalScreen extends StatefulWidget {
  @override
  _AdminPaymentApprovalScreenState createState() => _AdminPaymentApprovalScreenState();
}

```

```

class _AdminPaymentApprovalScreenState extends State<AdminPaymentApprovalScreen> {
  List<Map<String, dynamic>> _pendingTransactions = [];
  bool _isLoading = true;

  @override
  void initState() {
    super.initState();
  }
}

```

```

    _fetchPendingTransactions();
}

Future<void> _fetchPendingTransactions() async {
  setState(() {
    _isLoading = true;
  });

  try {
    final url = Uri.parse('https://nanjundeshwara.vercel.app/pending_transactions');
    print('Sending request to: $url');

    final response = await http.get(url);

    print('Response status code: ${response.statusCode}');
    print('Response body: ${response.body}');

    if (response.statusCode == 200) {
      final transactions = json.decode(response.body);
      print('Transactions found: ${transactions.length}');
      setState(() {
        _pendingTransactions = List<Map<String, dynamic>>.from(transactions);
        _isLoading = false;
      });
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Failed to fetch pending transactions: ${response.body}'),
          backgroundColor: AppTheme.danger,
        ),
      );
      setState(() {
        _isLoading = false;
      });
    }
  } catch (e) {
    print('Error: $e');
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('An error occurred: $e'),
        backgroundColor: AppTheme.danger,
      ),
    );
    setState(() {
      _isLoading = false;
    });
  }
}

Future<void> _approvePayment(String transactionId) async {
  try {
    final response = await http.post(
      Uri.parse('https://nanjundeshwara.vercel.app/approve_payment'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode({'transactionId': transactionId}),
    );
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Payment approved successfully'),
        backgroundColor: AppTheme.success,
      ),
    ),
  },
}

```

```

    );
    _fetchPendingTransactions();

    // if (response.statusCode == 200) {
    //   ScaffoldMessenger.of(context).showSnackBar(
    //     SnackBar(
    //       content: Text('Payment approved successfully'),
    //       backgroundColor: AppTheme.success,
    //     ),
    //   );
    //   _fetchPendingTransactions();
    // } else {
    //   ScaffoldMessenger.of(context).showSnackBar(
    //     SnackBar(
    //       content: Text('Failed to approve payment: ${response.body}'),
    //       backgroundColor: AppTheme.danger,
    //     ),
    //   );
    // }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('An error occurred: $e'),
        backgroundColor: AppTheme.danger,
      ),
    );
  }
}

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,
      ),
    );
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Pending Payments'),
    ),
  );
}

```

```

body: _isLoading
  ? Center(child: CircularProgressIndicator())
: _pendingTransactions.isEmpty
  ? Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          Icons.payment_outlined,
          size: 64,
          color: Colors.grey[400],
        ),
        SizedBox(height: 16),
        Text(
          'No pending payment requests',
          style: TextStyle(
            fontSize: 18,
            color: AppTheme.textSecondary,
          ),
        ),
      ],
    ),
  )
: RefreshIndicator(
  onRefresh: _fetchPendingTransactions,
  child: ListView.builder(
    itemCount: _pendingTransactions.length,
    itemBuilder: (context, index) {
      final transaction = _pendingTransactions[index];
      return Card(
        margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Row(
                children: [
                  Container(
                    padding: EdgeInsets.all(10),
                    decoration: BoxDecoration(
                      color: AppTheme.warning.withOpacity(0.1),
                      shape: BoxShape.circle,
                    ),
                  ),
                  Icon(
                    Icons.payment,
                    color: AppTheme.warning,
                    size: 24,
                  ),
                ],
              ),
              SizedBox(width: 16),
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Payment Request',
                      style: TextStyle(
                        fontSize: 18,
                        fontWeight: FontWeight.bold,
                      ),
                    ),

```

```
),
Text(
  'User ID: ${transaction['userId']}',
  style: TextStyle(
    fontSize: 14,
    color: AppTheme.textSecondary,
  ),
),
],
),
Container(
padding: EdgeInsets.symmetric(horizontal: 12, vertical: 6),
decoration: BoxDecoration(
color: AppTheme.warning.withOpacity(0.1),
borderRadius: BorderRadius.circular(16),
),
child: Text(
'Pending',
style: TextStyle(
color: AppTheme.warning,
fontWeight: FontWeight.bold,
fontSize: 12,
),
),
),
],
),
SizedBox(height: 16),
Divider(),
SizedBox(height: 8),
Row(
children: [
Expanded(
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
_buildPaymentInfoRow('Month', transaction['month']),
SizedBox(height: 8),
_buildPaymentInfoRow('Amount', '₹${transaction['amount']}'),
SizedBox(height: 8),
_buildPaymentInfoRow("Transaction ID", transaction['transactionId']),
],
),
),
],
),
SizedBox(height: 16),
Row(
children: [
Expanded(
child: OutlinedButton.icon(
onPressed: () => _rejectPayment(transaction['transactionId']),
icon: Icon(Icons.close),
label: Text('Reject'),
style: OutlinedButton.styleFrom(
foregroundColor: AppTheme.danger,
side: BorderSide(color: AppTheme.danger),
),
),
),
],
),
```



```

        SizedBox(width: 12),
        Expanded(
          child: ElevatedButton.icon(
            onPressed: () => _approvePayment(transaction['transactionId']),
            icon: Icon(Icons.check),
            label: Text('Approve'),
            style: ElevatedButton.styleFrom(
              backgroundColor: AppTheme.success,
            ),
          ),
        ),
      ],
    ),
  ],
),
);
},
),
),
);
}
);
}

```

```

Widget _buildPaymentInfoRow(String label, String value) {
  return Row(
    children: [
      Text(
        '$label: ',
        style: TextStyle(
          fontSize: 14,
          color: AppTheme.textSecondary,
          fontWeight: FontWeight.w500,
        ),
      ),
      Expanded(
        child: Text(
          value,
          style: TextStyle(
            fontSize: 14,
            fontWeight: FontWeight.bold,
          ),
          overflow: TextOverflow.ellipsis,
        ),
      ),
    ],
  );
}

```

```

class AdminOperationsScreen extends StatefulWidget {
  @override
  _AdminOperationsScreenState createState() => _AdminOperationsScreenState();
}

```

```

class _AdminOperationsScreenState extends State<AdminOperationsScreen> {
  final _formKey = GlobalKey<FormState>();
  final _userIdController = TextEditingController();
  final _nameController = TextEditingController();
  final _villageController = TextEditingController();
  String _selectedCategory = 'Gold';
  final _phoneController = TextEditingController();
}

```

```

final _amountController = TextEditingController();
final _monthController = TextEditingController();
final _searchController = TextEditingController();

List<Map<String, dynamic>> _users = [];
List<Map<String, dynamic>> _payments = [];
List<Map<String, dynamic>> _paidUsers = [];
List<Map<String, dynamic>> _unpaidUsers = [];
List<String> _villages = [];
String _selectedVillage = "";
List<Map<String, dynamic>> _villageUsers = [];
List<Map<String, dynamic>> _inactiveUsers = [];

String _selectedOperation = "";
Map<String, dynamic> _userToDelete = {};

List<String> categories = ['Gold', 'Silver', 'Bronze'];

@override
void dispose() {
  _userIdController.dispose();
  _nameController.dispose();
  _villageController.dispose();
  _phoneController.dispose();
  _amountController.dispose();
  _monthController.dispose();
  _searchController.dispose();
  super.dispose();
}

Future<void> _fetchVillages() async {
  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/get_all_villages'),
    );

    if (response.statusCode == 200) {
      final villages = json.decode(response.body);
      setState(() {
        _villages = List<String>.from(villages.map((v) => v['v_name']));
        if (_villages.isNotEmpty && _selectedVillage.isEmpty) {
          _selectedVillage = _villages[0];
        }
      });
    } else {
      _showErrorSnackBar('Failed to fetch villages: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

Future<void> _searchByVillage() async {
  if (_selectedVillage.isEmpty) {
    _showErrorSnackBar('Please select a village');
    return;
  }

  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/search_by_village?village=${_selectedVillage}'),
    );
  }
}

```

```

);

if (response.statusCode == 200) {
  final data = json.decode(response.body);
  setState(() {
    _villageUsers = List<Map<String, dynamic>>.from(data['users']);
  });
} else {
  _showErrorSnackBar('Failed to search by village: ${response.body}');
}
} catch (e) {
  _showErrorSnackBar('An error occurred: $e');
}
}

Future<void> _fetchInactiveCustomers() async {
  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/inactive_customers'),
    );

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        _inactiveUsers = List<Map<String, dynamic>>.from(data['inactiveUsers']);
      });
    } else {
      _showErrorSnackBar('Failed to fetch inactive customers: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

Future<void> _addUser() async {
  if (_formKey.currentState!.validate()) {
    try {
      final response = await http.post(
        Uri.parse('https://nanjundeshwara.vercel.app/add_user'),
        headers: {'Content-Type': 'application/json'},
        body: json.encode({
          'userId': int.parse(_userIdController.text),
          'c_name': _nameController.text,
          'c_vill': _villageController.text,
          'c_category': _selectedCategory,
          'phone': _phoneController.text,
        })),
      );

      _showSuccessSnackBar('User added successfully');
      _clearForm();

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

Future<void> _fetchUserDetailsForDeletion() async {
  if (_userIdController.text.isEmpty) {
    _showErrorSnackBar('Please enter a user ID to delete');
  }
}

```

```

    return;
  }

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

    if (response.statusCode == 200) {
      setState(() {
        _userToDelete = json.decode(response.body);
      });

      // Show confirmation dialog
      _showDeleteConfirmationDialog();
    } else {
      _showErrorSnackBar('Failed to fetch user details: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

void _showDeleteConfirmationDialog() {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Confirm Deletion'),
        content: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          children: [
            Text('Are you sure you want to move this user to trash?'),
            SizedBox(height: 20),
            Text('User Details:', style: TextStyle(fontWeight: FontWeight.bold)),
            SizedBox(height: 10),
            Text('ID: ${_userToDelete['_id']}'),
            Text('Name: ${_userToDelete['c_name']}'),
            Text('Village: ${_userToDelete['c_vill']}'),
            Text('Category: ${_userToDelete['c_category']}'),
            Text('Phone Number: ${_userToDelete['phone']}'),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: Text('Cancel'),
          ),
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
              _deleteUser();
            },
            child: Text(
              'Move to Trash',
              style: TextStyle(color: AppTheme.danger),
            ),
          ),
        ],
      );
    },
  );
}

```

```

    ],
    );
  },
);
}

Future<void> _deleteUser() async {
  try {
    final response = await http.delete(
      Uri.parse('https://nanjundeshwara.vercel.app/delete_user/${_userIdController.text}'),
    );

    if (response.statusCode == 200) {
      _showSuccessSnackBar('User moved to trash');
      _clearForm();
    } else {
      _showErrorSnackBar('Failed to delete user: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

Future<void> _viewAllUsers() async {
  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/find_all_users'),
    );

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

Future<void> _viewPaymentsByMonth() async {
  if (_monthController.text.isEmpty) {
    _showErrorSnackBar('Please enter a month to view payments');
    return;
  }

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

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        _paidUsers = List<Map<String, dynamic>>.from(data['paidUsers']);
        _unpaidUsers = List<Map<String, dynamic>>.from(data['unpaidUsers']);
      });
    } else {
      _showErrorSnackBar('Failed to fetch payments: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

Future<void> _fetchUserDetails() async {
  if (_userIdController.text.isEmpty) {
    return;
  }

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

    if (response.statusCode == 200) {
      final userData = json.decode(response.body);

```

```

    setState() {
      _nameController.text = userData['c_name'];
      _selectedCategory = userData['c_category'];
      switch (_selectedCategory) {
        case 'Gold':
          _amountController.text = '500';
          break;
        case 'Silver':
          _amountController.text = '300';
          break;
        default:
          _amountController.text = "";
      }
    });
  } else {
    _showErrorSnackBar('Failed to fetch user details: ${response.body}');
  }
} catch (e) {
  _showErrorSnackBar('An error occurred: $e');
}
}

void _clearForm() {
  _userIdController.clear();
  _nameController.clear();
  _villageController.clear();
  _selectedCategory = 'Gold';
  _phoneController.clear();
  _amountController.clear();
  _monthController.clear();
}

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

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

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Admin Operations'),
    ),
    body: Row(
      children: [
        // Sidebar

```

```

Container(
  width: 200,
  color: Colors.grey[100],
  child: ListView(
    padding: EdgeInsets.all(12),
    children: [
      _buildOperationButton('Add\nUser', Icons.person_add),
      _buildOperationButton('Delete\nUser', Icons.person_remove),
      _buildOperationButton('View\nAll\nUsers', Icons.people),
      _buildOperationButton('Add\nPayment', Icons.payment),
      _buildOperationButton('View\nPayments', Icons.receipt),
      _buildOperationButton('View\nPayments\nby\nMonth', Icons.calendar_month),
      _buildOperationButton('Search\nBy\nVillage', Icons.location_city),
      _buildOperationButton('Inactive\nCustomers', Icons.person_off),
    ],
  ),
),

// Main Content
Expanded(
  child: Container(
    padding: EdgeInsets.all(20),
    child: _selectedOperation.isEmpty
      ? Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(
                Icons.touch_app,
                size: 64,
                color: Colors.grey[400],
              ),
              SizedBox(height: 16),
              Text(
                'Select an operation from the sidebar',
                style: TextStyle(
                  fontSize: 18,
                  color: AppTheme.textSecondary,
                ),
              ),
            ],
          ),
        )
      : _buildOperationContent(),
  ),
),
);
}

```

```

Widget _buildOperationButton(String operation, IconData icon) {
  return Container(
    margin: EdgeInsets.only(bottom: 8),
    child: ElevatedButton(
      onPressed: () {
        setState(() {
          _selectedOperation = operation;
          _clearForm();

          // Initialize data for specific operations

```



```

        if (operation == 'Search\nBy\nVillage') {
            _fetchVillages();
        } else if (operation == 'Inactive\nCustomers') {
            _fetchInactiveCustomers();
        } else if (operation == 'View\nAll\nUsers') {
            _viewAllUsers();
        }
    });
},
style: ElevatedButton.styleFrom(
    backgroundColor: _selectedOperation == operation
        ? AppTheme.primaryColor
        : Colors.white,
    foregroundColor: _selectedOperation == operation
        ? Colors.white
        : AppTheme.textPrimary,
    elevation: _selectedOperation == operation ? 4 : 1,
    padding: EdgeInsets.symmetric(vertical: 16),
    shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
    ),
),
child: Column(
    children: [
        Icon(icon, size: 24),
        SizedBox(height: 8),
        Text(
            operation.replaceAll('\n', ' '),
            textAlign: TextAlign.center,
        ),
    ],
),
);
}

```

```

Widget _buildOperationContent() {
    switch (_selectedOperation) {
        case 'Add\nUser':
            return _buildAddUserForm();
        case 'Delete\nUser':
            return _buildDeleteUserForm();
        case 'View\nAll\nUsers':
            return _buildViewAllUsersContent();
        case 'Add\nPayment':
            return _buildAddPaymentForm();
        case 'View\nPayments':
            return _buildViewPaymentsContent();
        case 'View\nPayments\nby\nMonth':
            return _buildViewPaymentsByMonthContent();
        case 'Search\nBy\nVillage':
            return _buildSearchByVillageContent();
        case 'Inactive\nCustomers':
            return _buildInactiveCustomersContent();
        default:
            return Container();
    }
}

```

```

Widget _buildAddUserForm() {
    return Card(

```

```

elevation: 2,
child: Padding(
  padding: const EdgeInsets.all(20.0),
  child: Form(
    key: _formKey,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'Add New User',
          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,
        ),
        SizedBox(height: 16),
        TextFormField(
          controller: _nameController,
          decoration: InputDecoration(
            labelText: 'Name',
            prefixIcon: Icon(Icons.person),
          ),
          validator: (value) => value!.isEmpty ? 'Please enter name' : null,
        ),
        SizedBox(height: 16),
        TextFormField(
          controller: _villageController,
          decoration: InputDecoration(
            labelText: 'Village',
            prefixIcon: Icon(Icons.location_on),
          ),
          validator: (value) => value!.isEmpty ? 'Please enter village' : null,
        ),
        SizedBox(height: 16),
        DropdownButtonFormField<String>(
          value: _selectedCategory,
          decoration: InputDecoration(
            labelText: 'Category',
            prefixIcon: Icon(Icons.category),
          ),
          items: categories.map((String category) {
            return DropdownMenuItem<String>(
              value: category,
              child: Text(category),
            );
          }).toList(),
          onChanged: (String? newValue) {
            setState(() {
              _selectedCategory = newValue!;
            });
          },
        ),

```

```

    ),
    SizedBox(height: 16),
    TextFormField(
      controller: _phoneController,
      decoration: InputDecoration(
        labelText: 'Phone Number',
        prefixIcon: Icon(Icons.phone),
      ),
    ),
    keyboardType: TextInputType.phone,
    validator: (value) => value!.isEmpty ? 'Please enter phone number' : null,
  ),
  SizedBox(height: 24),
  SizedBox(
    width: double.infinity,
    child: ElevatedButton.icon(
      onPressed: _addUser,
      icon: Icon(Icons.add),
      label: Text('Add User'),
      style: ElevatedButton.styleFrom(
        padding: EdgeInsets.symmetric(vertical: 16),
      ),
    ),
  ),
),
],
),
),
);
}

```

```

Widget _buildDeleteUserForm() {
  return Card(
    elevation: 2,
    child: Padding(
      padding: const EdgeInsets.all(20.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Delete User',
            style: TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
          SizedBox(height: 20),
          TextFormField(
            controller: _userIdController,
            decoration: InputDecoration(
              labelText: 'User ID',
              prefixIcon: Icon(Icons.badge),
              hintText: 'Enter the ID of the user to delete',
            ),
            keyboardType: TextInputType.number,
          ),
          SizedBox(height: 24),
          SizedBox(
            width: double.infinity,
            child: ElevatedButton.icon(
              onPressed: _fetchUserDetailsForDeletion,
              icon: Icon(Icons.delete),
            ),
          ),
        ],
      ),
    ),
  );
}

```

```

        label: Text('Move User to Trash'),
        style: ElevatedButton.styleFrom(
          backgroundColor: AppTheme.danger,
          padding: EdgeInsets.symmetric(vertical: 16),
        ),
      ),
    ),
  ],
),
),
);
}

Widget _buildViewAllUsersContent() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
          Expanded(
            child: Text(
              'All Users',
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
          ElevatedButton.icon(
            onPressed: _viewAllUsers,
            icon: Icon(Icons.refresh),
            label: Text('Refresh'),
            style: ElevatedButton.styleFrom(
              backgroundColor: AppTheme.info,
            ),
          ),
        ],
      ),
      SizedBox(height: 16),
      if (_users.isNotEmpty)
        Container(
          padding: EdgeInsets.symmetric(horizontal: 12, vertical: 6),
          decoration: BoxDecoration(
            color: AppTheme.primaryColor.withOpacity(0.1),
            borderRadius: BorderRadius.circular(8),
          ),
          child: Text(
            'Total Users: ${_users.length}',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
              color: AppTheme.primaryColor,
            ),
          ),
        ),
      SizedBox(height: 16),
      TextField(
        controller: _searchController,
        decoration: InputDecoration(
          labelText: 'Search Users',
          prefixIcon: Icon(Icons.search),

```

```
hintText: 'Search by name, village, category or ID',
),
onChanged: (value) {
  setState() {
    if (value.isEmpty) {
      _viewAllUsers();
    } else {
      _users = _users.where((user) =>
        user['c_name'].toString().toLowerCase().contains(value.toLowerCase()) ||
        user['c_vill'].toString().toLowerCase().contains(value.toLowerCase()) ||
        user['c_category'].toString().toLowerCase().contains(value.toLowerCase()) ||
        user['_id'].toString().contains(value)
      ).toList();
    }
  });
},
),
 SizedBox(height: 16),
 Expanded(
   child: _users.isEmpty
     ? Center(
         child: Text(
           'No users found',
           style: TextStyle(
             fontSize: 16,
             color: AppTheme.textSecondary,
           ),
         ),
       )
   : ListView.builder(
       itemCount: _users.length,
       itemBuilder: (context, index) {
         final user = _users[index];
         return Card(
           margin: EdgeInsets.only(bottom: 8),
           child: ListTile(
             leading: CircleAvatar(
               backgroundColor: AppTheme.primaryColor.withOpacity(0.1),
             ),
             child: Text(
               user['c_name'].substring(0, 1).toUpperCase(),
               style: TextStyle(
                 color: AppTheme.primaryColor,
                 fontWeight: FontWeight.bold,
               ),
             ),
             title: Text(
               '${user['_id']} - ${user['c_name']}',
               style: TextStyle(
                 fontWeight: FontWeight.bold,
               ),
             ),
             subtitle: Text('Village: ${user['c_vill']}, Category: ${user['c_category']}'),
             trailing: Container(
               padding: EdgeInsets.symmetric(horizontal: 8, vertical: 4),
               decoration: BoxDecoration(
                 color: AppTheme.info.withOpacity(0.1),
                 borderRadius: BorderRadius.circular(12),
               ),
               child: Text(
                 'Payments: ${user['paymentCount']}',

```

```

        style: TextStyle(
          color: AppTheme.info,
          fontWeight: FontWeight.bold,
        ),
      ),
    ),
  ),
);
},
),
),
],
);
}

```

```

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),
          ),
        ),
      ),
    ],
  );
}

```

```

    ),
  ),
),
PreferredSize(height: 20),
Expanded(
  child: _payments.isEmpty
    ? Center(
        child: Text(
          'No payments found',
          style: TextStyle(
            fontSize: 16,
            color: AppTheme.textSecondary,
          ),
        ),
      ),
    )
): ListView.builder(
  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']}'),
      ),
    );
  },
),
),
],
);
}

```

```

Widget _buildViewPaymentsByMonthContent() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'View Payments by Month',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(height: 20),
      TextFormField(
        controller: _monthController,
        decoration: InputDecoration(
          labelText: 'Month (MM format)',

```



```

        _buildUserList(_paidUsers, true),
        // Unpaid Users Tab
        _buildUserList(_unpaidUsers, false),
      ],
    ),
  ),
],
),
),
),
],
);
}

```

```

Widget _buildUserList(List<Map<String, dynamic>> users, bool isPaid) {
  return users.isEmpty
    ? Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            isPaid ? Icons.check_circle_outline : Icons.error_outline,
            size: 48,
            color: Colors.grey[400],
          ),
          SizedBox(height: 16),
          Text(
            isPaid ? 'No paid users found' : 'No unpaid users found',
            style: TextStyle(
              fontSize: 16,
              color: AppTheme.textSecondary,
            ),
          ),
        ],
      ),
    )
    : ListView.builder(
      itemCount: users.length,
      itemBuilder: (context, index) {
        final user = users[index];
        return Card(
          margin: EdgeInsets.only(bottom: 8),
          child: ListTile(
            leading: CircleAvatar(
              backgroundColor: isPaid
                ? AppTheme.success.withOpacity(0.1)
                : AppTheme.danger.withOpacity(0.1),
            ),
            child: Icon(
              isPaid ? Icons.check : Icons.close,
              color: isPaid ? AppTheme.success : AppTheme.danger,
            ),
            title: Text(
              'ID: ${user['_id']} - ${user['c_name']}',
              style: TextStyle(
                fontWeight: FontWeight.bold,
              ),
            ),
            subtitle: Text('Village: ${user['c_vill']}, Category: ${user['c_category']}, Phone: ${user['number']}'),
            trailing: isPaid
              ? Container(

```

```

padding: EdgeInsets.symmetric(horizontal: 8, vertical: 4),
decoration: BoxDecoration(
  color: AppTheme.success.withOpacity(0.1),
  borderRadius: BorderRadius.circular(12),
),
child: Text(
  '₹$ {user['amount']}',
  style: TextStyle(
    color: AppTheme.success,
    fontWeight: FontWeight.bold,
  ),
),
)
: null,
),
);
},
);
}

```

```

Widget _buildSearchByVillageContent() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'Search By Village',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      SizedBox(height: 20),
      Container(
        padding: EdgeInsets.symmetric(horizontal: 12),
        decoration: BoxDecoration(
          border: Border.all(color: Colors.grey[300]!),
          borderRadius: BorderRadius.circular(8),
          color: Colors.white,
        ),
        child: DropdownButton<String>(
          value: _selectedVillage.isNotEmpty ? _selectedVillage : null,
          hint: Text('Select Village'),
          isExpanded: true,
          underline: SizedBox(),
          items: _villages.map((String village) {
            return DropdownMenuItem<String>(
              value: village,
              child: Text(village),
            );
          }).toList(),
          onChanged: (String? newValue) {
            setState() {
              _selectedVillage = newValue!;
            };
          },
        ),
      ),
      SizedBox(height: 20),
      SizedBox(
        width: double.infinity,
        child: ElevatedButton.icon(

```



```

        trailing: Container(
          padding: EdgeInsets.symmetric(horizontal: 8, vertical: 4),
          decoration: BoxDecoration(
            color: AppTheme.info.withOpacity(0.1),
            borderRadius: BorderRadius.circular(12),
          ),
          child: Text(
            'Payments: ${user['paymentCount']}',
            style: TextStyle(
              color: AppTheme.info,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    ),
  );
},
),
),
],
);
}

```

```

Widget _buildInactiveCustomersContent() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
          Expanded(
            child: Text(
              'Inactive Customers',
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
          ElevatedButton.icon(
            onPressed: _fetchInactiveCustomers,
            icon: Icon(Icons.refresh),
            label: Text('Refresh'),
            style: ElevatedButton.styleFrom(
              backgroundColor: AppTheme.info,
            ),
          ),
        ],
      ),
      SizedBox(height: 16),
      if (_inactiveUsers.isNotEmpty)
        Container(
          padding: EdgeInsets.symmetric(horizontal: 12, vertical: 6),
          decoration: BoxDecoration(
            color: AppTheme.danger.withOpacity(0.1),
            borderRadius: BorderRadius.circular(8),
          ),
          child: Text(
            'Inactive Customers: ${_inactiveUsers.length}',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
    ],
  );
}

```



```

    ),
  ],
);
}
}

class UserScreen extends StatefulWidget {
  final String username;

  UserScreen({required this.username});

  @override
  _UserScreenState createState() => _UserScreenState();
}

class _UserScreenState extends State<UserScreen> {
  Map<String, dynamic> _userDetails = {};
  List<Map<String, dynamic>> _payments = [];
  List<Map<String, dynamic>> _notifications = [];
  bool _isLoading = true;

  @override
  void initState() {
    super.initState();
    _fetchData();
  }

  Future<void> _fetchData() async {
    setState() {
      _isLoading = true;
    });

    await Future.wait([
      _fetchUserDetails(),
      _fetchPayments(),
      _fetchNotifications(),
    ]);

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

  Future<void> _fetchUserDetails() async {
    try {
      final response = await http.get(
        Uri.parse('https://nanjundeshwara.vercel.app/find_user?userId=${widget.username}'),
      );

      if (response.statusCode == 200) {
        setState() {
          _userDetails = json.decode(response.body);
        });
      } else {
        _showErrorSnackBar('Failed to fetch user details: ${response.body}');
      }
    } catch (e) {
      _showErrorSnackBar('An error occurred: $e');
    }
  }
}

```



```

Future<void> _fetchPayments() async {
  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/find_payments?userIdPayments=${widget.username}'),
    );

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

Future<void> _fetchNotifications() async {
  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/notifications/${widget.username}'),
    );

    if (response.statusCode == 200) {
      setState(() {
        _notifications = List<Map<String, dynamic>>.from(json.decode(response.body));
      });
    } else {
      _showErrorSnackBar('Failed to fetch notifications: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  }
}

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,
                          ),
                        ),
                        Center(
                          child: Text(
                            _userDetails['c_name'] != null && _userDetails['c_name'].isNotEmpty

```



```

        fontSize: 18,
        fontWeight: FontWeight.bold,
      ),
    ),
    SizedBox(height: 12),

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

```

```

Widget _buildUserInfoRow(IconData icon, String label, String value, [Color? valueColor]) {
  return Row(
    children: [
      Icon(
        icon,
        size: 20,
        color: AppTheme.textSecondary,
      ),
      SizedBox(width: 12),
      Text(
        '$label:',
        style: TextStyle(
          fontSize: 16,
          color: AppTheme.textSecondary,
        ),
      ),
      SizedBox(width: 8),
      Expanded(
        child: Text(
          value,
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.bold,
            color: valueColor ?? AppTheme.textPrimary,
          ),
          textAlign: TextAlign.right,
        ),
      ),
    ],
  );
}

```

```

Widget _buildActionCard(String title, IconData icon, Color color, VoidCallback onTap) {
  return InkWell(
    onTap: onTap,
    borderRadius: BorderRadius.circular(12),
    child: Container(
      padding: EdgeInsets.all(16),
      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(
    children: [
      Container(
        padding: EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: color.withOpacity(0.1),
          shape: BoxShape.circle,
        ),
        child: Icon(
          icon,
          color: color,
          size: 24,
        ),
      ),
      SizedBox(height: 12),
      Text(
        title,
        style: TextStyle(
          fontSize: 14,
          fontWeight: FontWeight.bold,
        ),
        textAlign: TextAlign.center,
      ),
    ],
  ),
),
);
}
}

```

```

class NotificationsScreen extends StatelessWidget {
  final List<Map<String, dynamic>> notifications;

```

```

  NotificationsScreen({required this.notifications});

```

```

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Notifications'),
      ),
      body: notifications.isEmpty
        ? Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(
                  Icons.notifications_off,
                  size: 64,
                  color: Colors.grey[400],
                ),
                SizedBox(height: 16),
                Text(
                  'No notifications',
                  style: TextStyle(
                    fontSize: 18,
                    color: AppTheme.textSecondary,

```

```

        ),
      ),
    ],
  ),
)
: ListView.builder(
  itemCount: notifications.length,
  itemBuilder: (context, index) {
    final notification = notifications[index];
    final DateTime createdAt = DateTime.parse(notification['createdAt']);
    final String formattedDate = '${createdAt.day}/${createdAt.month}/${createdAt.year}
    ${createdAt.hour}:${createdAt.minute.toString().padLeft(2, '0')}';

    return Card(
      margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      child: ListTile(
        leading: CircleAvatar(
          backgroundColor: AppTheme.primaryColor.withOpacity(0.1),
          child: Icon(
            Icons.notifications,
            color: AppTheme.primaryColor,
            size: 20,
          ),
        ),
        title: Text(
          notification['message'],
          style: TextStyle(
            fontWeight: FontWeight.w500,
          ),
        ),
        subtitle: Text(
          formattedDate,
          style: TextStyle(
            fontSize: 12,
            color: AppTheme.textSecondary,
          ),
        ),
      ),
    );
  },
);
}
}

```

```

class PaymentScreen extends StatefulWidget {
  final String username;

  PaymentScreen({required this.username});

  @override
  _PaymentScreenState createState() => _PaymentScreenState();
}

```

```

class _PaymentScreenState extends State<PaymentScreen> {
  final _transactionIdController = TextEditingController();
  bool _isPaid = false;
  bool _isLoading = true;
  Map<String, dynamic> _userDetails = {};
  String _selectedMonth = '01';
  List<String> _months = [];
}

```



```

@override
void initState() {
  super.initState();
  _initializeMonths();
  _fetchUserDetails();
}

void _initializeMonths() {
  _months = List.generate(12, (index) {
    return (index + 1).toString().padLeft(2, '0');
  });
}

@override
void dispose() {
  _transactionIdController.dispose();
  super.dispose();
}

Future<void> _fetchUserDetails() async {
  setState() {
    _isLoading = true;
  });

  try {
    final response = await http.get(
      Uri.parse('https://nanjundeshwara.vercel.app/find_user?userId=${widget.username}'),
    );

    if (response.statusCode == 200) {
      setState() {
        _userDetails = json.decode(response.body);
      });
      await _checkPaymentStatus();
    } else {
      _showErrorSnackBar('Failed to fetch user details: ${response.body}');
    }
  } catch (e) {
    _showErrorSnackBar('An error occurred: $e');
  } finally {
    setState() {
      _isLoading = false;
    });
  }
}

Future<void> _checkPaymentStatus() async {
  try {
    final url = Uri.parse(
      'https://nanjundeshwara.vercel.app/check_payment_status?userId=${widget.username}&month=$_selectedMonth',
    );

    final response = await http.get(url);

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState() {
        _isPaid = data['isPaid'];
      });
    } else {

```

```

        _showErrorSnackBar('Failed to check payment status: ${response.body}');
    }
} catch (e) {
    _showErrorSnackBar('An error occurred: $e');
}
}

Future<void> _requestPayment() async {
    if (_transactionIdController.text.isEmpty) {
        _showErrorSnackBar('Please enter a transaction ID');
        return;
    }

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

    try {
        final response = await http.post(
            Uri.parse('https://nanjundeshwara.vercel.app/request_payment'),
            headers: {'Content-Type': 'application/json'},
            body: json.encode({
                'userId': int.parse(widget.username),
                'month': _selectedMonth,
                'amount': _userDetails['c_category'] == 'Gold' ? 500 : 300,
                'transactionId': _transactionIdController.text,
            }),
        );

        _showSuccessSnackBar('Payment request submitted successfully');
        _transactionIdController.clear();
        await _checkPaymentStatus();

    } catch (e) {
        _showErrorSnackBar('An error occurred: $e');
    } finally {
        setState(() {
            _isLoading = false;
        });
    }
}

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

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

```

```

}

String _getMonthName(String monthNumber) {
  final months = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
  ];
  final index = int.parse(monthNumber) - 1;
  return months[index];
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Make Payment'),
    ),
    body: _isLoading
      ? Center(child: CircularProgressIndicator())
      : SingleChildScrollView(
          padding: EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              // Month Selection Card
              Card(
                elevation: 2,
                child: Padding(
                  padding: const EdgeInsets.all(16.0),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        'Select Month',
                        style: TextStyle(
                          fontSize: 18,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                      SizedBox(height: 16),
                      Container(
                        padding: EdgeInsets.symmetric(horizontal: 12),
                        decoration: BoxDecoration(
                          border: Border.all(color: Colors.grey[300]!),
                          borderRadius: BorderRadius.circular(8),
                          color: Colors.white,
                        ),
                        child: DropdownButton<String>(
                          value: _selectedMonth,
                          isExpanded: true,
                          underline: SizedBox(),
                          onChanged: (String? newValue) {
                            setState() {
                              _selectedMonth = newValue!;
                            };
                            _checkPaymentStatus();
                          },
                          items: _months.map((String month) {
                            return DropdownMenuItem<String>(
                              value: month,
                              child: Text('${_getMonthName(month)} (${month})'),

```



```

    ),
    child: Icon(
      Icons.payment,
      color: AppTheme.warning,
      size: 24,
    ),
  ),
  SizedBox(width: 16),
  Expanded(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'Payment Required',
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.bold,
          ),
        ),
        Text(
          'For ${_getMonthName(_selectedMonth)}',
          style: TextStyle(
            fontSize: 14,
            color: AppTheme.textSecondary,
          ),
        ),
      ],
    ),
  ),
  Container(
    padding: EdgeInsets.symmetric(horizontal: 12, vertical: 6),
    decoration: BoxDecoration(
      color: AppTheme.warning.withOpacity(0.1),
      borderRadius: BorderRadius.circular(16),
    ),
    child: Text(
      'Pending',
      style: TextStyle(
        color: AppTheme.warning,
        fontWeight: FontWeight.bold,
        fontSize: 12,
      ),
    ),
  ),
],
),
SizedBox(height: 20),
Divider(),
SizedBox(height: 20),

// QR Code Section
Center(
  child: Column(
    children: [
      Container(
        width: 200,
        height: 200,
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(12),
          boxShadow: [

```



```

    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" },
})

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({
      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 })

    // 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("/total_amount_paid", async (req, res) => {

```

```

const userId = Number.parseInt(req.query.userId)
try {
  const result = await Payment.aggregate([
    { $match: { c_id: userId } },
    { $group: { _id: "$c_id", total_amount: { $sum: "$amount" } } },
  ])
  const totalAmount = result.length > 0 ? result[0].total_amount : 0
  res.json({ user_id: userId, total_amount: totalAmount })
} catch (error) {
  res.status(500).json({ error: error.message })
}
})

```

```

app.get("/find_user", async (req, res) => {
  const userId = Number.parseInt(req.query.userId)
  try {
    const user = await User.findById(userId)
    if (user) {
      res.json(user)
    } else {
      res.status(404).json({ error: res.locals.t("userNotFound") })
    }
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

```

```

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

  try {
    const query = { c_id: userId }
    if (month) {
      query.p_month = month
    }

    const payments = await Payment.aggregate([
      { $match: query },
      {
        $lookup: {

```

```

        from: "users",
        localField: "c_id",
        foreignField: "_id",
        as: "customer",
      },
    ],
    { $unwind: "$customer" },
    {
      $project: {
        p_id: "$_id",
        c_id: 1,
        p_month: 1,
        amount: 1,
        c_name: "$customer.c_name",
      },
    },
  ])
  res.json(payments)
} catch (error) {
  res.status(500).json({ error: error.message })
}
})

app.get("/view_payments_by_month", async (req, res) => {
  const month = req.query.p_month
  try {
    // Get all active users
    const allUsers = await User.find({ isDeleted: { $ne: true } }).sort({ _id: 1 })

    // Get all payments for the specified month
    const payments = await Payment.find({ p_month: month })

    // Create a map of user IDs who have paid
    const paidUserIds = new Set(payments.map((payment) => payment.c_id))

    // Separate users into paid and unpaid
    const paidUsers = []
    const unpaidUsers = []

    for (const user of allUsers) {
      if (paidUserIds.has(user._id)) {

```

```

// Find the payment for this user
const payment = payments.find((p) => p.c_id === user._id)
paidUsers.push({
  _id: user._id,
  c_name: user.c_name,
  c_vill: user.c_vill,
  c_category: user.c_category,
  amount: payment ? payment.amount : 0,
  number: user.phone,
})
} else {
  unpaidUsers.push({
    _id: user._id,
    c_name: user.c_name,
    c_vill: user.c_vill,
    c_category: user.c_category,
    number: user.phone,
  })
}
}

// Sort both arrays by ID
paidUsers.sort((a, b) => a._id - b._id)
unpaidUsers.sort((a, b) => a._id - b._id)

res.json({
  paidCount: paidUsers.length,
  unpaidCount: unpaidUsers.length,
  totalCount: allUsers.length,
  paidUsers,
  unpaidUsers,
})
} catch (error) {
  res.status(500).json({ error: error.message })
}
})

app.get("/find_all_users", async (req, res) => {
  try {
    // Only get active users
    const users = await User.find({ isDeleted: { $ne: true } }).sort({ _id: 1 })

```

```

// Get payment counts for each user
const userIds = users.map((user) => user._id)
const paymentCounts = await Payment.aggregate([
  { $match: { c_id: { $in: userIds } } },
  { $group: { _id: "$c_id", count: { $sum: 1 } } },
])

// Create a map of user ID to payment count
const paymentCountMap = {}
paymentCounts.forEach((item) => {
  paymentCountMap[item._id] = item.count
})

// Add payment count to each user
const usersWithPaymentCount = users.map((user) => {
  const userObj = user.toObject()
  userObj.paymentCount = paymentCountMap[user._id] || 0
  return userObj
})

res.json({
  totalCount: users.length,
  users: usersWithPaymentCount,
})
} catch (error) {
  res.status(500).json({ error: error.message })
}
})

// New endpoint to get deleted users (trash)
app.get("/deleted_users", async (req, res) => {
  try {
    const deletedUsers = await User.find({ isDeleted: true }).sort({ deletedAt: -1 })

    // Get payment counts for each user
    const userIds = deletedUsers.map((user) => user._id)
    const paymentCounts = await Payment.aggregate([
      { $match: { c_id: { $in: userIds } } },
      { $group: { _id: "$c_id", count: { $sum: 1 } } },
    ])

```

```

// Create a map of user ID to payment count
const paymentCountMap = { }
paymentCounts.forEach((item) => {
  paymentCountMap[item._id] = item.count
})

// Add payment count to each user
const usersWithPaymentCount = deletedUsers.map((user) => {
  const userObj = user.toObject()
  userObj.paymentCount = paymentCountMap[user._id] || 0
  return userObj
})

res.json({
  totalCount: deletedUsers.length,
  users: usersWithPaymentCount,
})
} catch (error) {
  res.status(500).json({ error: error.message })
}
})

// New endpoint to restore a deleted user
app.put("/restore_user/:userId", async (req, res) => {
  const userId = Number.parseInt(req.params.userId)
  try {
    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("userNotDeleted") })
    }

    user.isDeleted = false
    user.deletedAt = null
    await user.save()

    res.json({ message: res.locals.t("userRestored"), data: user })
  }
})

```



```

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

  // New endpoint to permanently delete a user
  app.delete("/permanent_delete_user/:userId", async (req, res) => {
    const userId = Number.parseInt(req.params.userId)

    try {
      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("userNotInTrash") })
      }

      // Permanently delete the user
      await User.findByIdAndDelete(userId)

      // Delete related records
      await Payment.deleteMany({ c_id: userId })
      await Notification.deleteMany({ userId: userId })
      await Transaction.deleteMany({ userId: userId })

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

  app.get("/search_users", async (req, res) => {
    const { name, c_category, c_vill } = req.query
    try {
      const query = { isDeleted: { $ne: true } } // Only search active users
      if (name) query.c_name = new RegExp(name, "i")
      if (c_category) query.c_category = new RegExp(c_category, "i")
      if (c_vill) query.c_vill = new RegExp(c_vill, "i")

      const users = await User.find(query).sort({ _id: 1 })

```

```

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

app.get("/get_all_villages", async (req, res) => {
  try {
    const villages = await Village.find().sort({ v_name: 1 })
    res.json(villages)
  } catch (error) {
    res.status(500).json({ error: error.message })
  }
})

app.get("/search_by_village", async (req, res) => {
  const { village } = req.query
  try {
    if (!village) {
      return res.status(400).json({ error: "Village name is required" })
    }

    const users = await User.find({ c_vill: village, isDeleted: { $ne: true } }).sort({ _id: 1 })

    // Get payment counts for each user
    const userIds = users.map((user) => user._id)
    const paymentCounts = await Payment.aggregate([
      { $match: { c_id: { $in: userIds } } },
      { $group: { _id: "$c_id", count: { $sum: 1 } } },
    ])

    // Create a map of user ID to payment count
    const paymentCountMap = {}
    paymentCounts.forEach((item) => {
      paymentCountMap[item._id] = item.count
    })

    // Add payment count to each user
    const usersWithPaymentCount = users.map((user) => {
      const userObj = user.toObject()
      userObj.paymentCount = paymentCountMap[user._id] || 0
    })
  }
})

```

```

    return userObj
  })

  res.json({
    village,
    count: users.length,
    users: usersWithPaymentCount,
  })
} 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,
        })
      }
    }
  }
})

```

```

    }
  }

  // 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") })
    }
  }
})

```

```

// 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) => {

```

```

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 })
}
})

```



```

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")
}

```

```

// 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))
    );
  }

```

```

    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

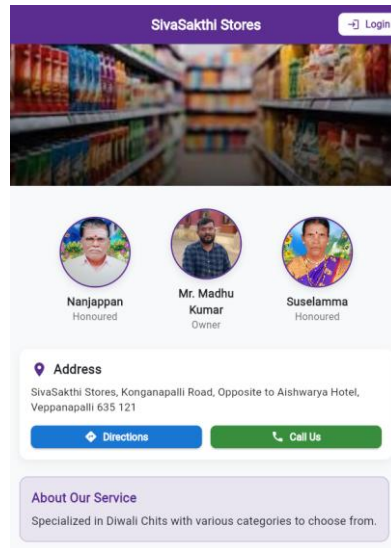


Figure A2.1 Home Page

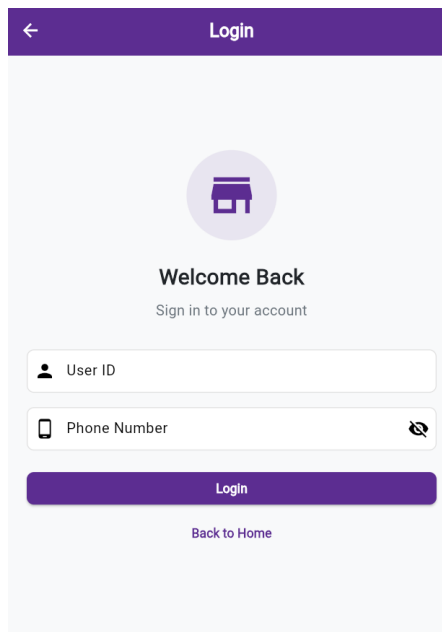


Figure A2.2 Login Page

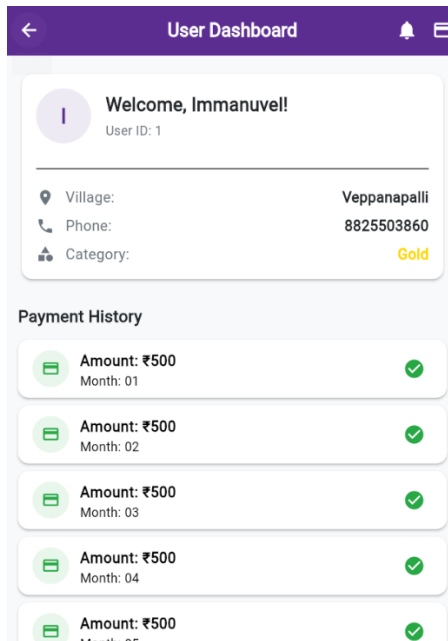


Figure A2.3 User Dashboard Page

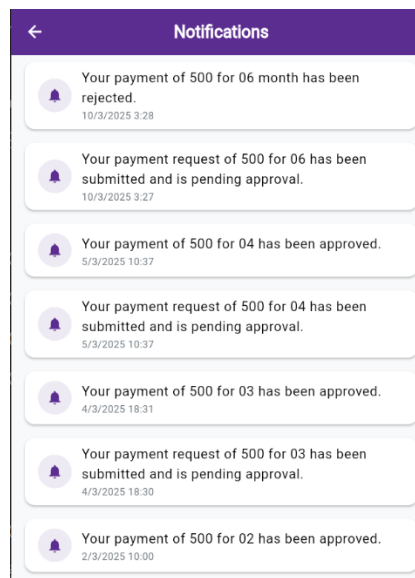


Figure A2.4 Notification Page

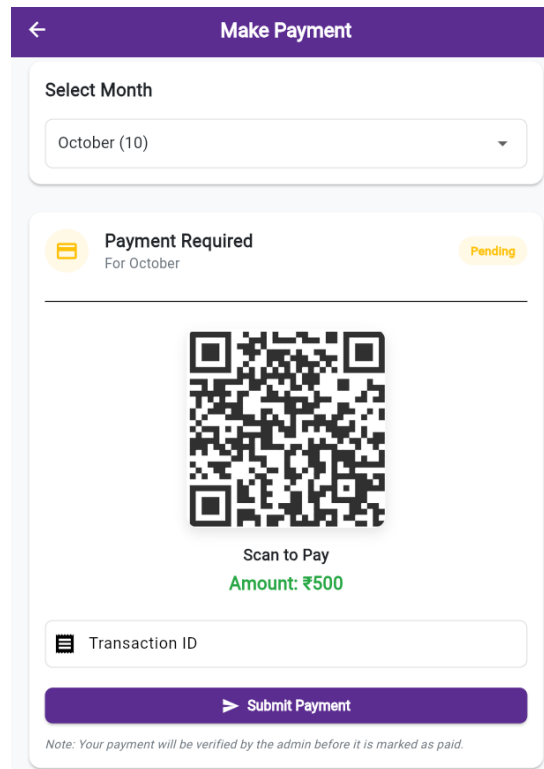


Figure A2.5 Request Payment Page

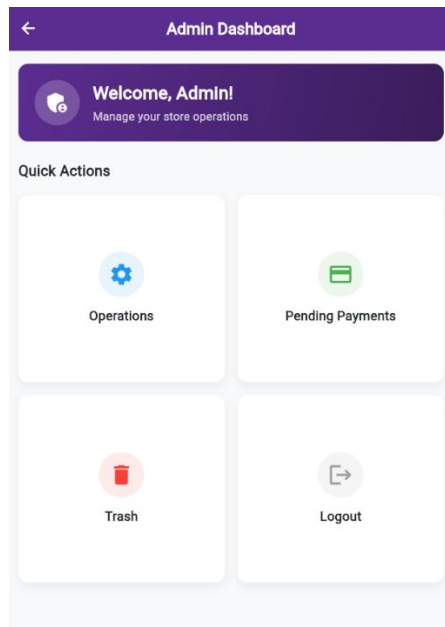




Figure A2.6 Admin Dashboard Page

←


Admin Operations

+ 


Add User




Delete User




View All Users




Add Payment




View Payments



View Payments by Month





Search By Village




Inactive


Add New User

 User ID


 Name

 Village

Category

 Gold

▼

 Phone Number

+ Add User

Figure A2.7 Admin Operations Page

118

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/>
2. <https://github.com/Krittika57>
3. <https://github.com/barathkumarr2004>