URS App Documentation

Team name: Chill Guys

February 4, 2025

Contents

1 Project Structure **2**

2 Introduction **2**

3 Core Features **2**
- 3.1 Authentication System . . . 2
- 3.2 Payment Features . . . 2
- 3.3 Business Tools . . . 2

4 File Structure and Key Components **3**
- 4.1 Main Files . . . 3
- 4.2 Database Management . . . 3
- 4.3 Models . . . 3

5 Database Structure **3**
- 5.1 Vendors Table . . . 3
- 5.2 Transactions Table . . . 3
- 5.3 Bills Table . . . 4

6 Technical Implementation **4**
- 6.1 Vendor Management . . . 4
- 6.2 Transaction Processing . . . 4

7 Future Improvements **5**

1 Project Structure

The application follows a structured organization of files and directories:

```
lib/
        database/
            database_helper.dart
            vendor_data_manager.dart
        models/
            vendor.dart
            phone.dart
            transaction.dart
        screens/
            login_page.dart
            home_page.dart
            payment_screen.dart
            qr_scanner_screen.dart
            transaction_history.dart
            create_bill.dart
            my_bills.dart
            profile_screen.dart
            settings_screen.dart
        main.dart
```

2 Introduction

The URS (Unified Reward System) App is a modern digital payment and billing system designed to provide users with a secure, convenient, and efficient way to manage transactions. The app integrates multiple functionalities, including payment processing, QR code scanning, bill management, and vendor analytics. To ensure security and reliability, Firebase is used for authentication, while SQLite is employed for local data storage. This documentation outlines the core features, database schema, file structure, technical implementation, and planned future improvements.

3 Core Features

3.1 Authentication System

The authentication system ensures secure access through email and password login. Users can add their UPI ID to facilitate direct payments. The app maintains user profiles, allowing them to update personal information and track transaction history. Firebase Authentication is used to handle secure logins and user session management.

3.2 Payment Features

- **QR Code Scanning**: Users can scan vendor-generated QR codes to initiate payments. - **UPI Payments**: Transactions are processed seamlessly through UPI. - **Transaction History**: Users can view past payments with details such as amount, vendor name, and date. - **Bill Management**: The app allows users to create, store, and categorize bills for expense tracking. - **Digital Receipts**: For every transaction, a digital receipt is generated, which can be downloaded or shared.

3.3 Business Tools

- **Bill Creation and Management**: Users can create new bills, save drafts, and upload past transactions for reference. - **Database Handling**: SQLite is used to manage stored transaction records, vendor details, and reward points. - **Vendor Analytics**: Vendors can access transaction statistics, such as total earnings and customer spending trends. - **Customer Reward Points**: The app tracks points earned from purchases, providing incentives for users to shop frequently.

4 File Structure and Key Components

4.1 Main Files

The following are the core Dart files and their functionalities:

- `login_page.dart`: Handles user authentication using Firebase.
- `home_page.dart`: Displays the main dashboard with available features.
- `payment_screen.dart`: Processes UPI payments and transaction completion.
- `qr_scanner_screen.dart`: Allows users to scan QR codes for quick payments.

4.2 Database Management

The database is managed using SQLite, with helper classes facilitating CRUD operations.

- `database_helper.dart`: Handles database creation, insertion, and queries.
- `vendor_data_manager.dart`: Manages vendor information and inserts sample data for testing.

4.3 Models

To maintain structured data, the app uses models to define objects:

- `vendor.dart`: Represents a vendor, including name, UPI ID, and business type.
- `transaction.dart`: Represents a transaction, including the amount, date, and vendor ID.

5 Database Structure

The app employs SQLite to maintain local transaction records efficiently. Key tables include:

5.1 Vendors Table

Stores details of registered vendors, including:

- **vendorId**: Unique identifier for the vendor.
- **upiId**: UPI ID for payment transactions.
- **name**: Business name.
- **type**: Vendor category (small, medium, big).
- **phoneNumber**: Contact number.
- **location**: Address or city of operation.

5.2 Transactions Table

Maintains transaction records with the following details:

- **transactionId**: Unique transaction identifier.
- **amount**: Transaction amount.
- **vendorId**: Associated vendor.
- **timestamp**: Date and time of payment.
- **rewardPoints**: Points earned based on the transaction.

5.3 Bills Table

Tracks itemized bills linked to transactions:

- **billId**: Unique bill identifier.
- **transactionId**: Associated transaction.
- **items**: List of purchased items.
- **totalPrice**: Final price of the bill.

6 Technical Implementation

6.1 Vendor Management

The `VendorDataManager` class manages vendor-related data. Below is an implementation to insert sample vendor data:

Listing 1: Vendor Data Manager

```
import 'database_helper.dart';
import '../models/vendor.dart';

class VendorDataManager {
  final DatabaseHelper _dbHelper = DatabaseHelper();

  Future<void> addSampleVendors() async {
    final vendors = [
      Vendor(
        vendorId: 'V001',
        upiId: 'vendor1@oksbi',
        name: 'John Store',
        type: 'small',
        phoneNumber: '1234567890',
        location: 'New York',
      ),
      Vendor(
        vendorId: 'V002',
        upiId: 'asitmdesai@oksbi',
        name: 'Super Market',
        type: 'medium',
        phoneNumber: '0987654321',
        location: 'Los Angeles',
      ),
    ];

    for (var vendor in vendors) {
      await _dbHelper.insertVendor(vendor);
    }
  }
}
```

6.2 Transaction Processing

The app processes payments and stores transaction details. The `DatabaseHelper` class ensures secure database interactions:

Listing 2: Database Helper

```
class DatabaseHelper {
  Future<Vendor?> getVendorByUpiId(String upiId) async {
```

```
    final Database db = await database;
    final List<Map<String, dynamic>> maps = await db.query(
      'vendors',
      where: 'upi_id = ?',
      whereArgs: [upiId],
    );
    return maps.isNotEmpty ? Vendor.fromMap(maps.first) : null;
  }

  Future<int> insertTransaction(Map<String, dynamic> transaction) async {
    final Database db = await database;
    return await db.insert('transactions', transaction);
  }
}
```

7 Future Improvements

- **Biometric Authentication**: Implement fingerprint or facial recognition for enhanced security.
- **Offline Transactions**: Enable users to complete transactions without an active internet connection.
- **Advanced Analytics**: Provide insights into spending trends and vendor performance.
- **Draft Button Implementation**: Introduce a draft button in the bill management section, allowing users to save bills temporarily before finalizing them. This will help users review and modify transactions before confirming payment.