



IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT

2025 – Lab Exam 03 Report

Student ID	
Batch	
Marking Guide	
1. Functionality: How well the core and bonus features are implemented	4
2. Creativity & Usability: Clean and intuitive UI/UX design	2
3. Validation & Error Handling: Proper input validation, error handling, and user feedback implementation	2
4. Data Persistence: Proper use of SharedPreferences and Internal Storage	2
Total Marks	10
Evaluator	

Description:

<<Enter the Description of your application here>>

Screenshots:

<<Paste the screenshots of your application here>>

Content of xml files of Strings and Colors:

Strings

<<Paste the screenshots of your application here>>

Colors

<<Paste the screenshots of your application here>>



IT2010 Lab Exam 03 Report

MELOCH

Finance Tracking Mobile Application

Student Details

Student Name : D G A D HIRUSHA

Student ID Number : IT23183018

Group Number : 06.01

Table of Contents

MELOCH - Functional Overview and Business Logic	4
Nature of the Application	4
User Flow Overview	4
Business Logic	5
Core Functionalities	5
Bonus Functionalities	6
Conclusion	7
Graphical User Interfaces	8
Strings XML	39
Colors XML	41
SharedPreferences Kt	42
NotificationManager Kt	48
JSONBackupGenerator Kt	51

MELOCH – Functional Overview and Business Logic

The **MELOCH** is a modern and user-focused mobile application designed to help individuals manage their personal finances with ease and clarity. It functions as a comprehensive finance tracker, offering users a private, intelligent, and visually intuitive platform to monitor their income, expenses, budgets, and financial trends over time.

Nature of the Application

MELOCH is a **personal finance management tool** aimed at empowering users with complete control over their day-to-day financial activities. The app is structured with a strong focus on **usability, privacy, and practicality**. It serves as a digital wallet, budgeting assistant, and expense monitor—integrated seamlessly into one cohesive experience.

The app is designed for individuals who wish to:

- Track their income and expenditures
- Set personal budgets and control overspending
- Organize their transactions by date, category, and payment method
- Visualize their spending patterns through charts and summaries
- Manage wallet cards securely
- Export reports and back up their data for safety and accountability

User Flow Overview

The user journey in **MELOCH** is designed to be smooth, intuitive, and supportive of daily financial activities. Here's how users interact with the app from start to finish:

1. **Account Registration & Login**
 - Users begin by signing up or logging in with their credentials.
 - Each account is securely isolated, giving the user access to their personalized financial dashboard.
2. **Dashboard Overview**
 - Upon login, users land on the home screen, where they can view their total balance, budgets, recent records, and spending breakdowns.
 - Key figures and charts give a quick snapshot of financial health.
3. **Adding Financial Records**
 - Users can add income or expense entries through a dedicated form.
 - Each record includes amount, category, date, and payment method.
 - These records instantly update the overall financial summary and budgets.
4. **Budget Monitoring**
 - Users can set budgets for each spending category.
 - As expenses accumulate, budget usage is tracked with visual indicators.

- Users receive notifications when they approach or exceed a limit.
- 5. Wallet Card Management**
- Users can store and manage details of credit and debit cards.
 - Cards display masked sensitive information and contribute to the user's balance.
 - Cards can be toggled visible/invisible for added privacy.
- 6. Viewing Transaction History**
- All transactions are grouped by date and displayed in a dedicated Records screen.
 - Users can scroll, filter, and even delete individual entries.
- 7. Data Export and Backup**
- Users can choose to export their financial data as a backup (JSON) or as a professional PDF report.
 - These files are stored locally and can be shared externally.
- 8. Reports and Insights**
- The app provides charts and legends summarizing spending habits.
 - Users can compare income vs. expenses and understand which categories dominate their budget.
- 9. Notifications and Alerts**
- Users are notified of key events such as budget limits or resets.
 - These alerts help maintain awareness without overwhelming the user.
- 10. Profile and Logout**
- From the profile section, users can update details, change preferences, or delete their account if desired.

Business Logic

The core business logic of **MELOCH** revolves around personalized financial tracking. Each user is provided with a private environment to record and manage their financial data. All features are tailored to respond to user behavior, ensuring an adaptive and efficient system.

Key business principles:

- Every financial action updates the user's balance and budgets in real time
- Income increases available funds; expenses reduce them accordingly
- Budgets are monitored to prevent overspending
- Alerts notify users of critical financial events (e.g., nearing budget limits)
- Users can add and manage financial records in a secure and structured manner

Core Functionalities

The Meloch app supports a wide range of features that together form the backbone of its financial management system:

1. User Account Management

- Users can create secure accounts with login credentials
- Each account is private and isolated from others
- Deleting an account removes all associated data

2. Transaction Recording

- Users can record both **income** and **expenses**
- Each transaction includes a title, amount, category, date, and payment method
- Transactions can be browsed by date and filtered by time (e.g., today, this month)

3. Budget Planning

- Users can assign monthly budgets for different categories
- Remaining budget is calculated and visualized automatically
- Users receive visual warnings when they approach or exceed limits

4. Wallet Card Management

- Users can save and organize both credit and debit cards
- Cards are visually presented with options to mask or unmask sensitive data
- Cards contribute to the overall financial balance

5. Insightful Dashboards

- Overview of total balance, budget usage, and category-based expenses
- Pie charts and bar graphs help users understand spending behavior
- Color-coded visuals enhance quick analysis

6. Transaction History

- Chronologically grouped record of all transactions
- Clear icons and color indicators for income and expenses
- Easy to navigate, filter, and manage

Bonus Functionalities

Meloch enhances its core experience with the inclusion of additional, thoughtful features that provide long-term value to users.

Data Backup Creation

- Users can generate full backups of their financial data
- Backups are saved in a shareable JSON format
- Ensures safety of records in case of device loss or data corruption
- Backup files are neatly stored in appropriate folders on the device

Report Generation (PDF)

- Users can generate professional PDF reports of their financial activity
- Reports include summaries of income, expenses, and category-wise breakdowns
- Visual charts (like doughnut graphs) are included to simplify interpretation
- Reports can be saved, printed, or shared for personal or professional purposes

Smart Notifications

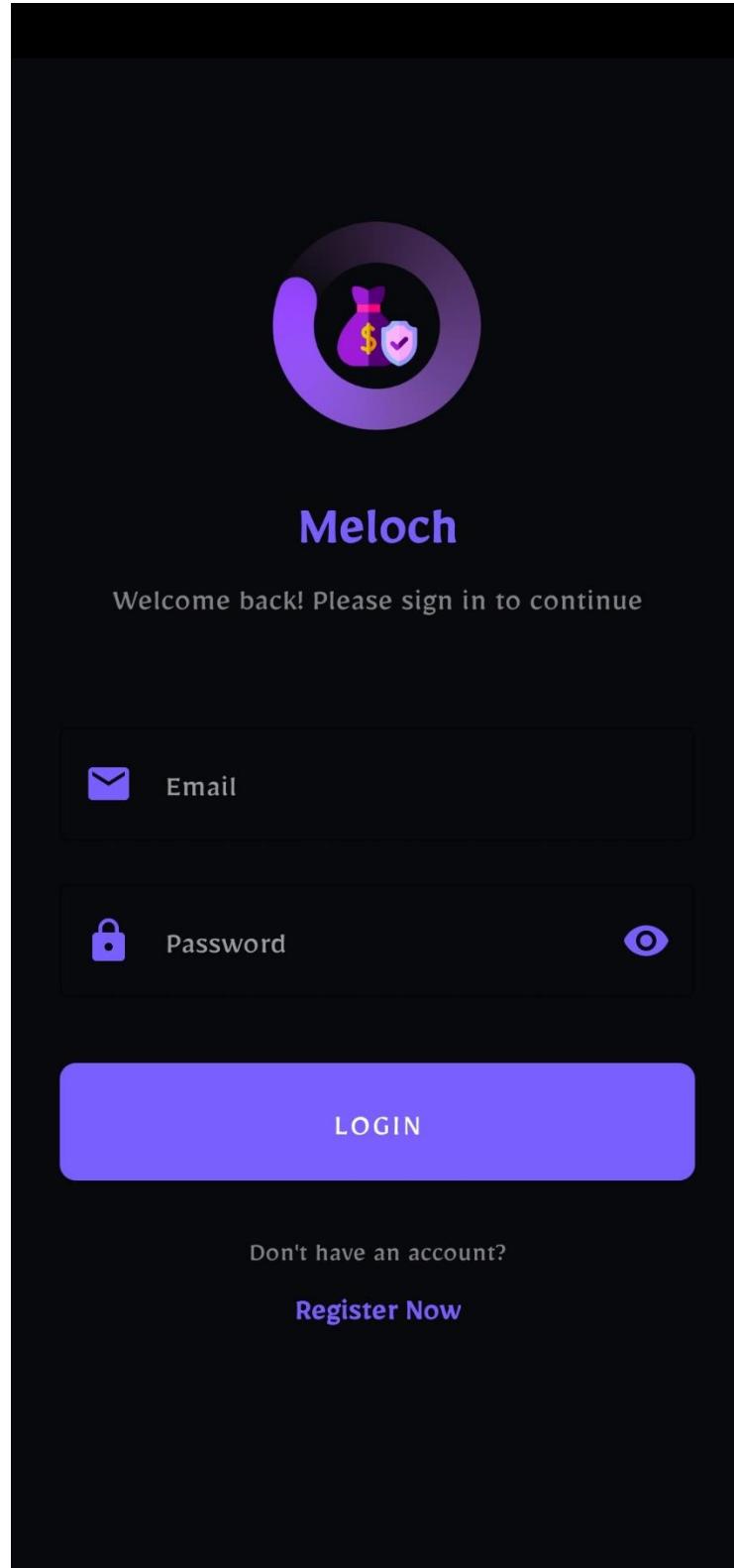
- The app alerts users when budgets are running low or fully used
- Notifications also appear for budget resets or important financial activity
- Designed to keep users financially aware and informed without being intrusive

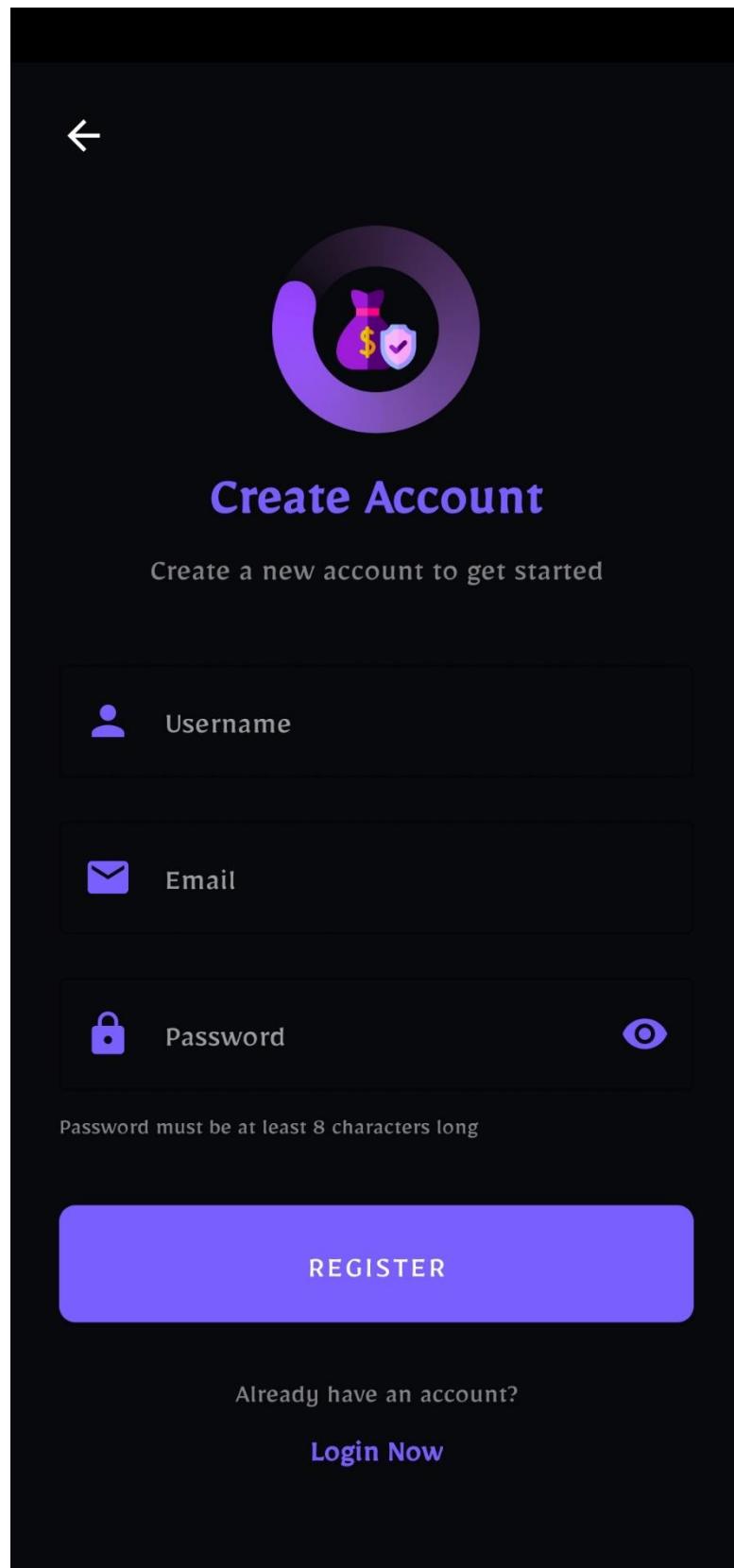
Conclusion

Meloch delivers an all-in-one financial management solution that combines everyday convenience with powerful financial insights. Through its thoughtful features—such as budget monitoring, transaction tracking, wallet card storage, and data export—the app supports users in building responsible financial habits. Its clean user interface, user-first logic, and bonus tools like report generation and backups make Meloch not just a finance tracker, but a trusted daily companion for financial wellness.

Graphical User Interfaces







MELOCH

Where Small Change Makes a Big Difference

Total Balance

LKR416,107.36



Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income:
LKR350,000.00 | Expenses: LKR142,650.64

Budget

All Budgets

LKR7,349.36 left

-LKR142,650.64 spent this month

Food - LKR17,500.00

Transport - LKR9,350.00

Health - LKR10,500.32

Shopping - LKR15,200.00

Vacation - LKR6,400.32

Entertainment - LKR8,700.00

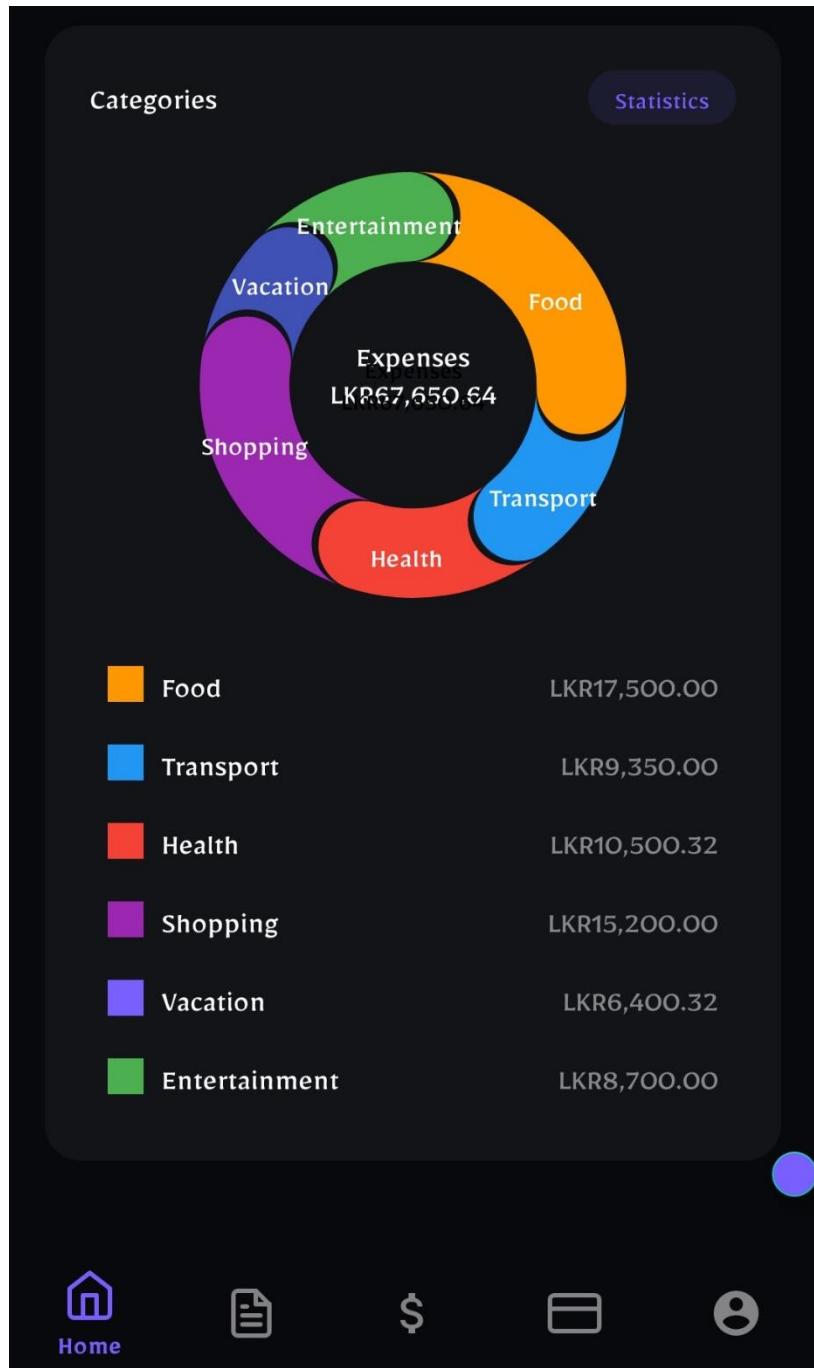


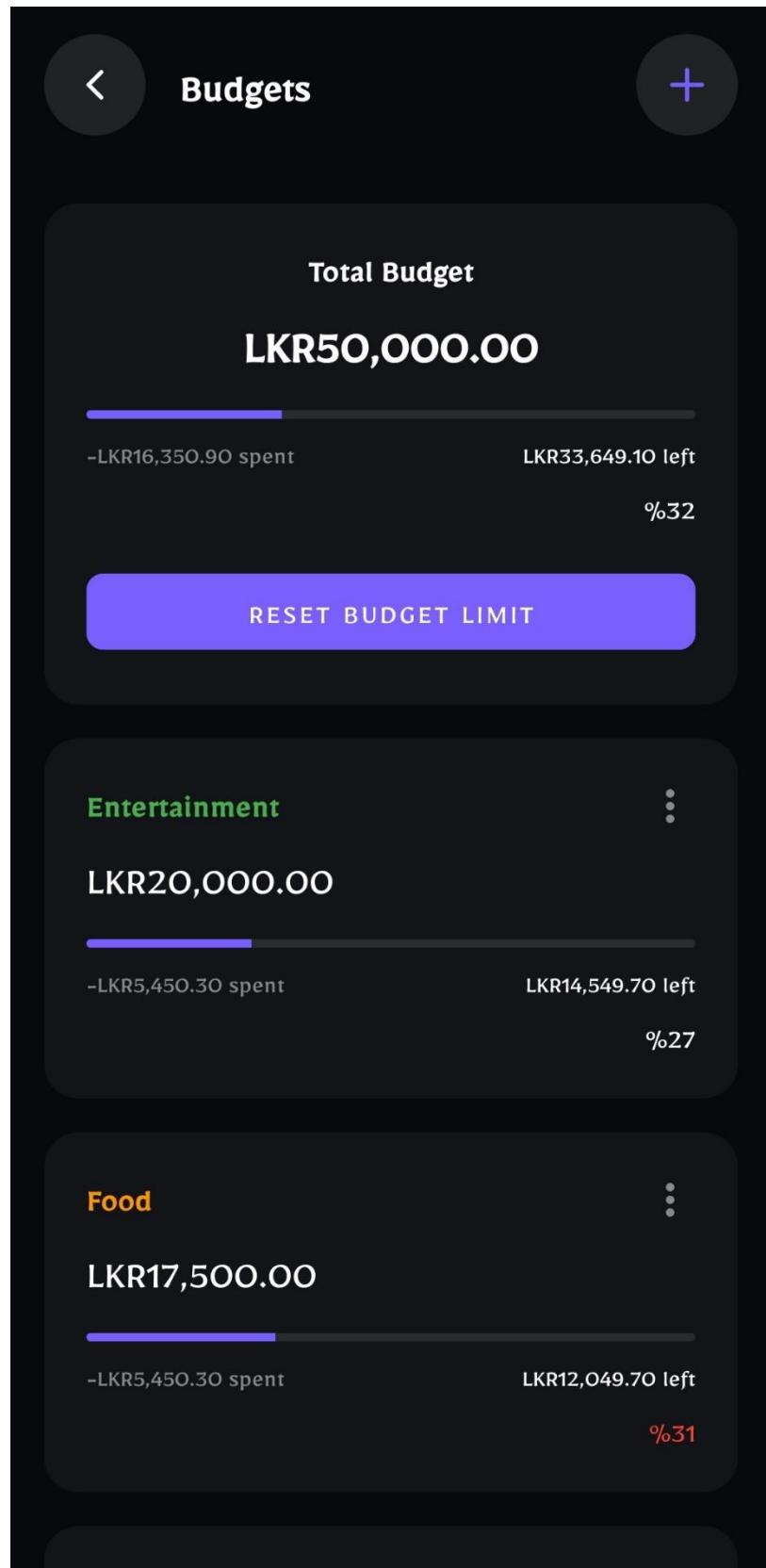
Home

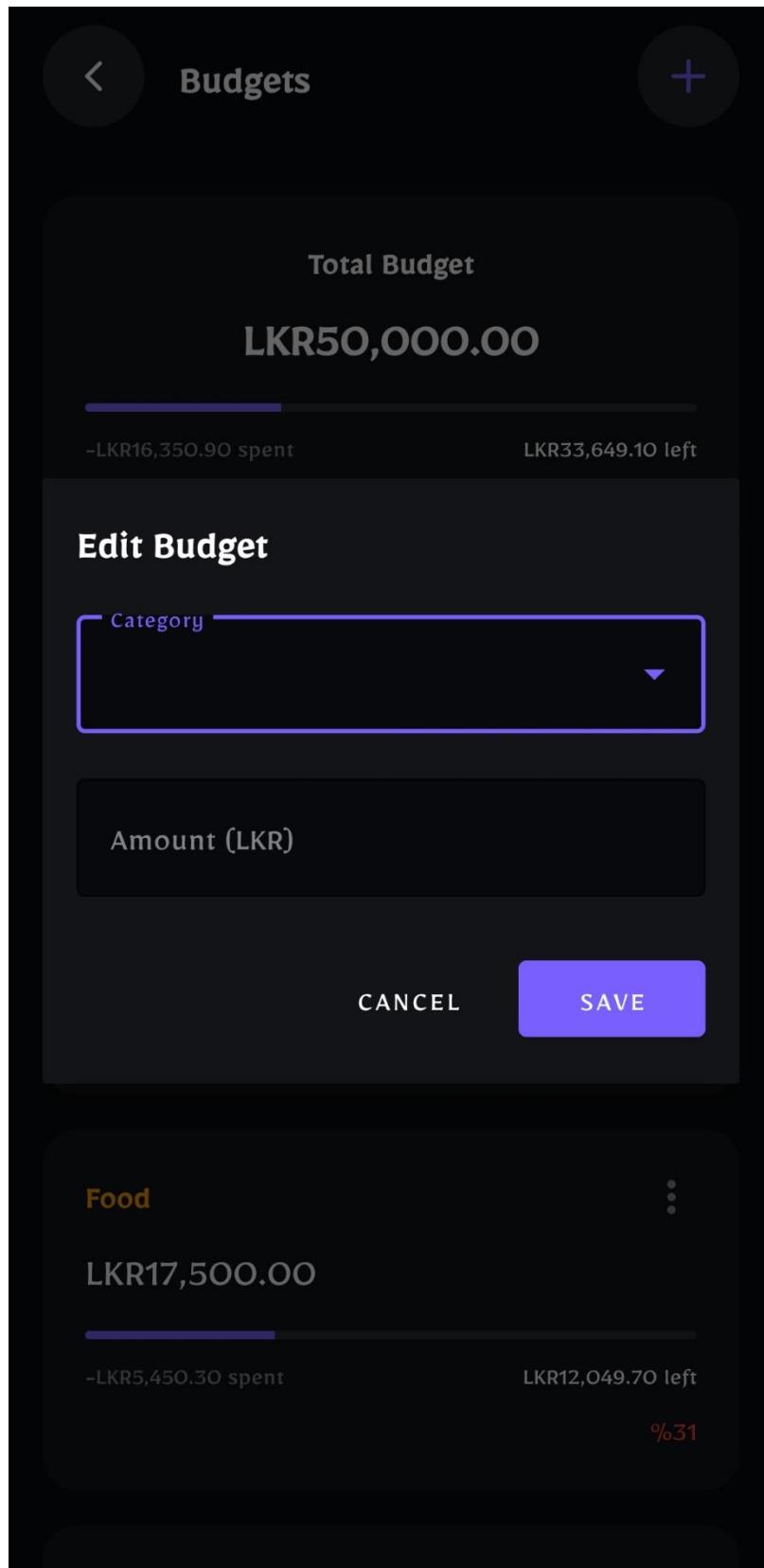


\$











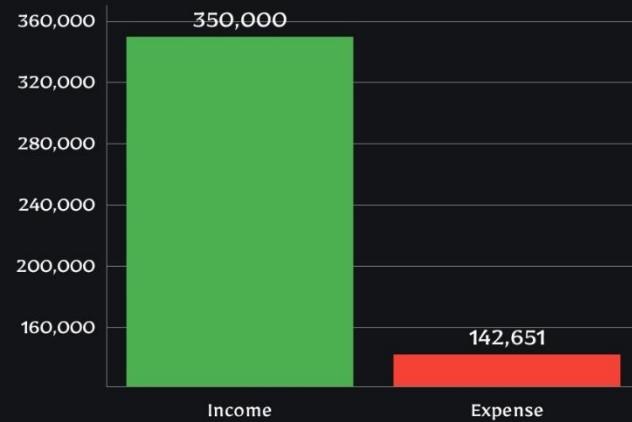
Statistics



April 2025



Income vs Expense



Income

Expense

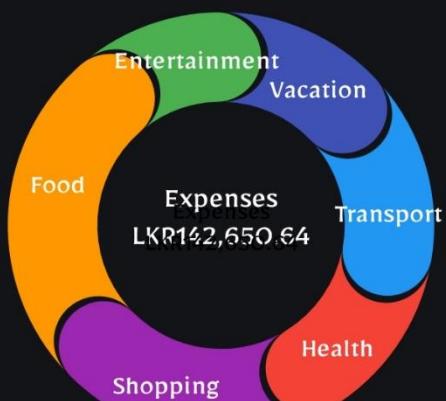
Balance

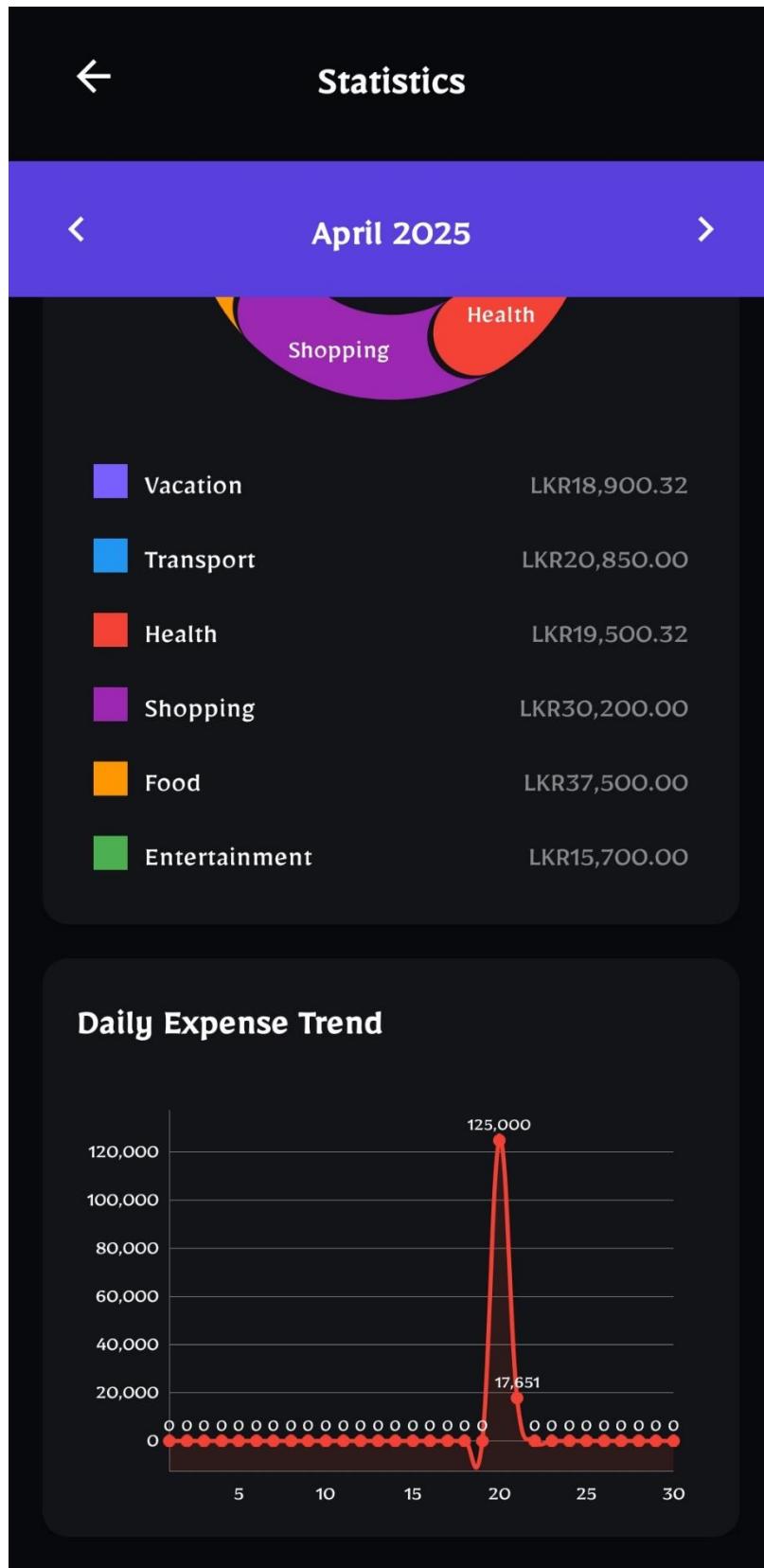
LKR 350,000.00

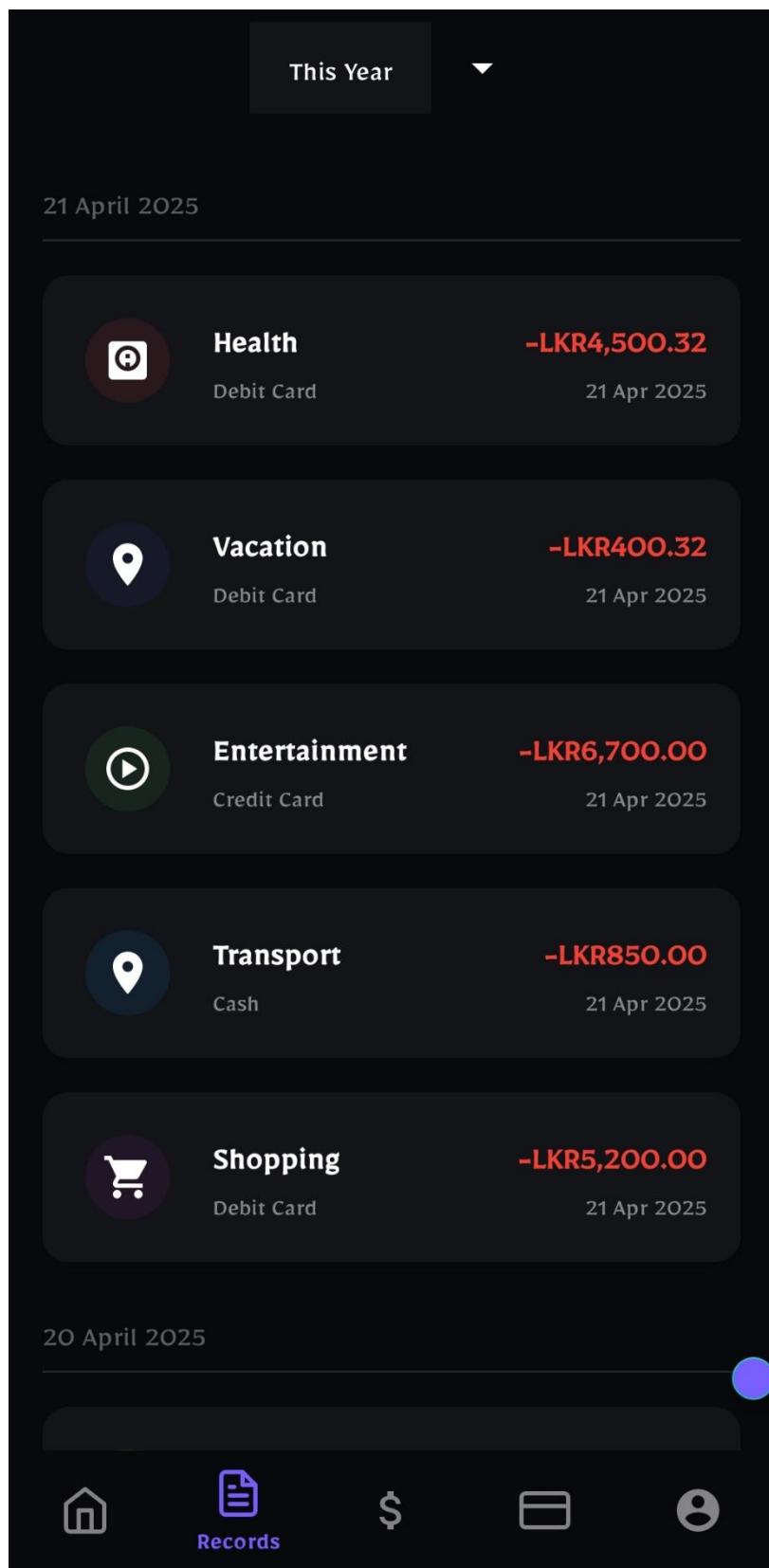
LKR 142,650.64

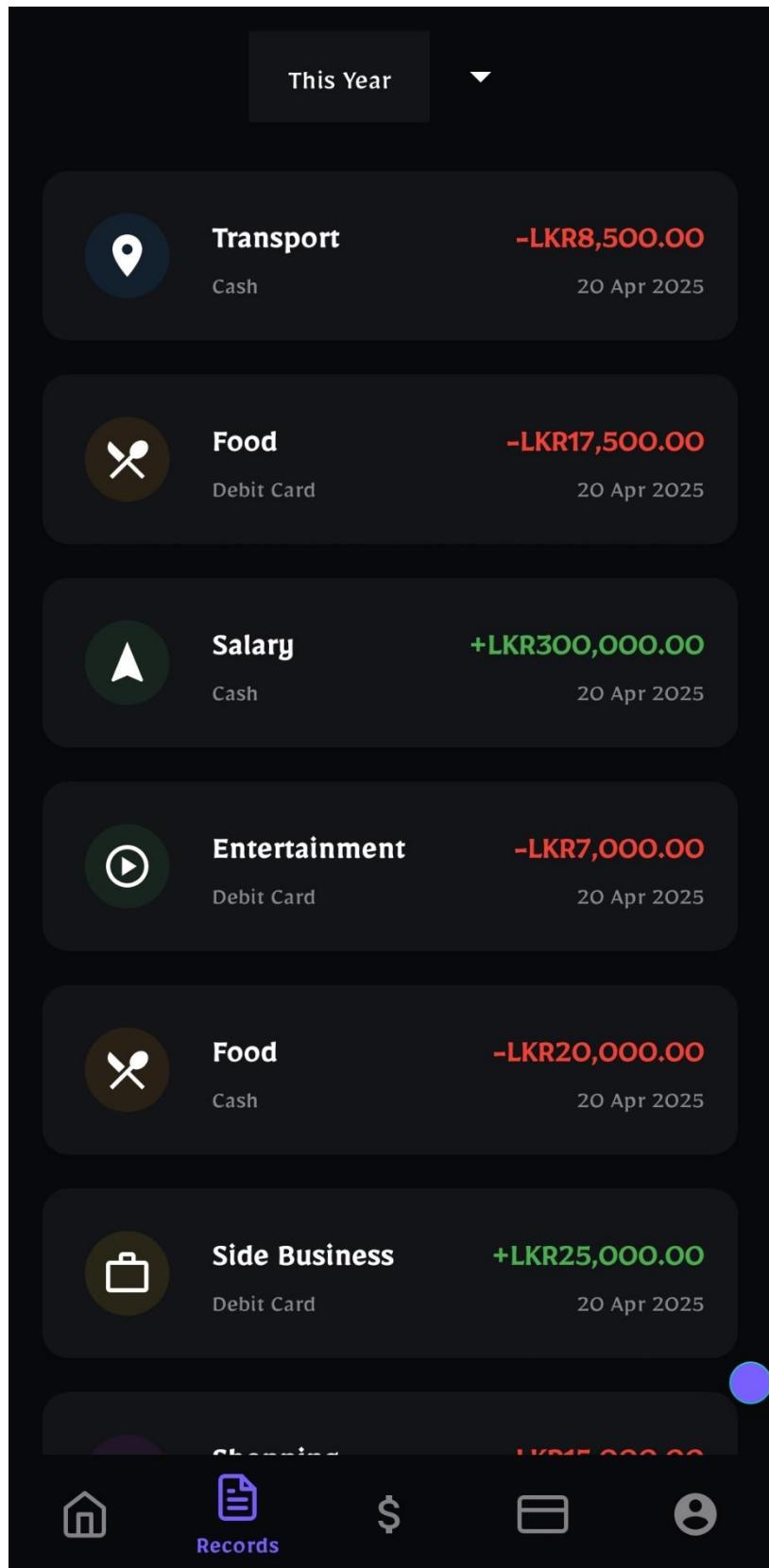
LKR 207,349.36

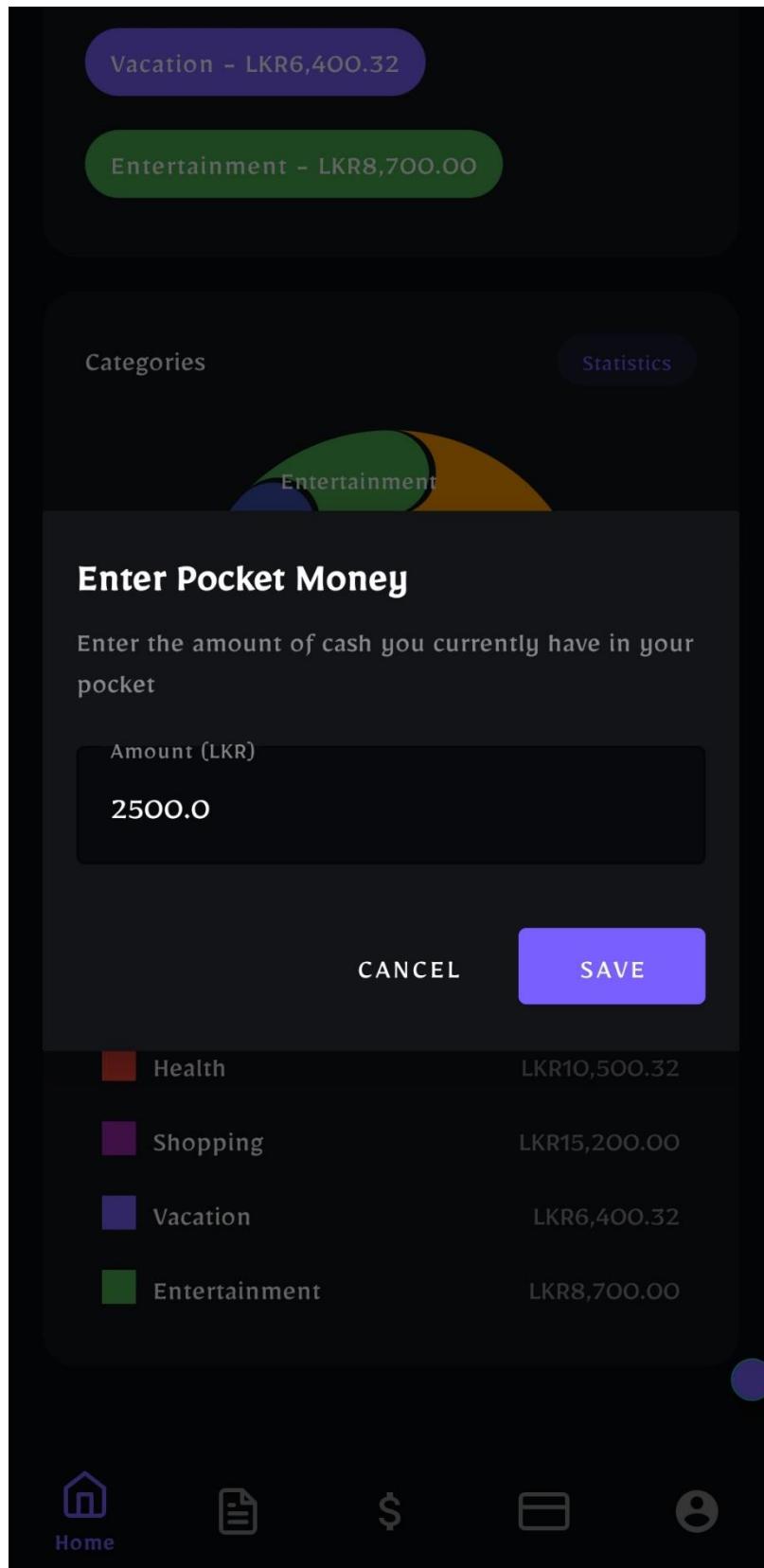
Expense by Category

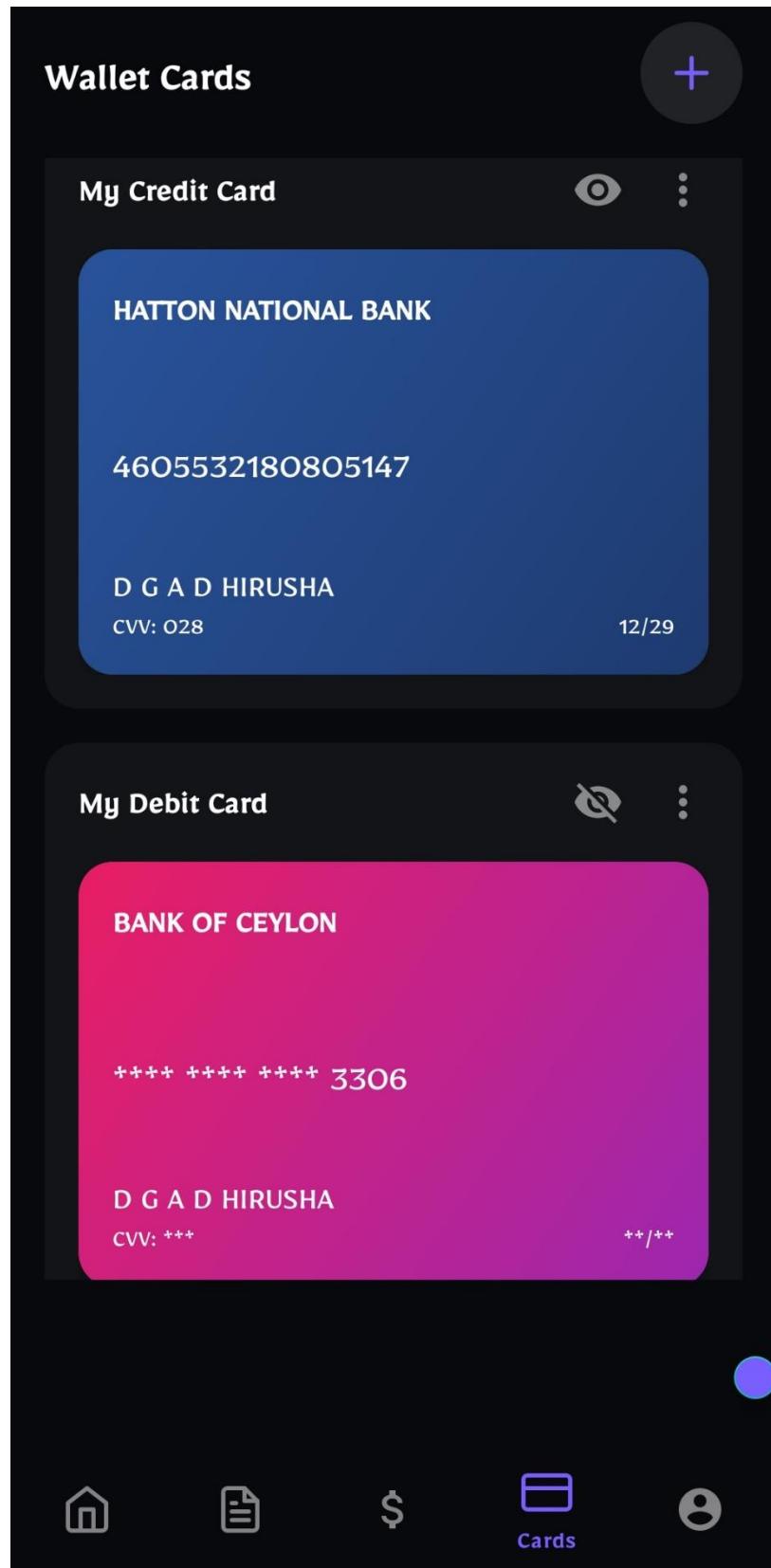


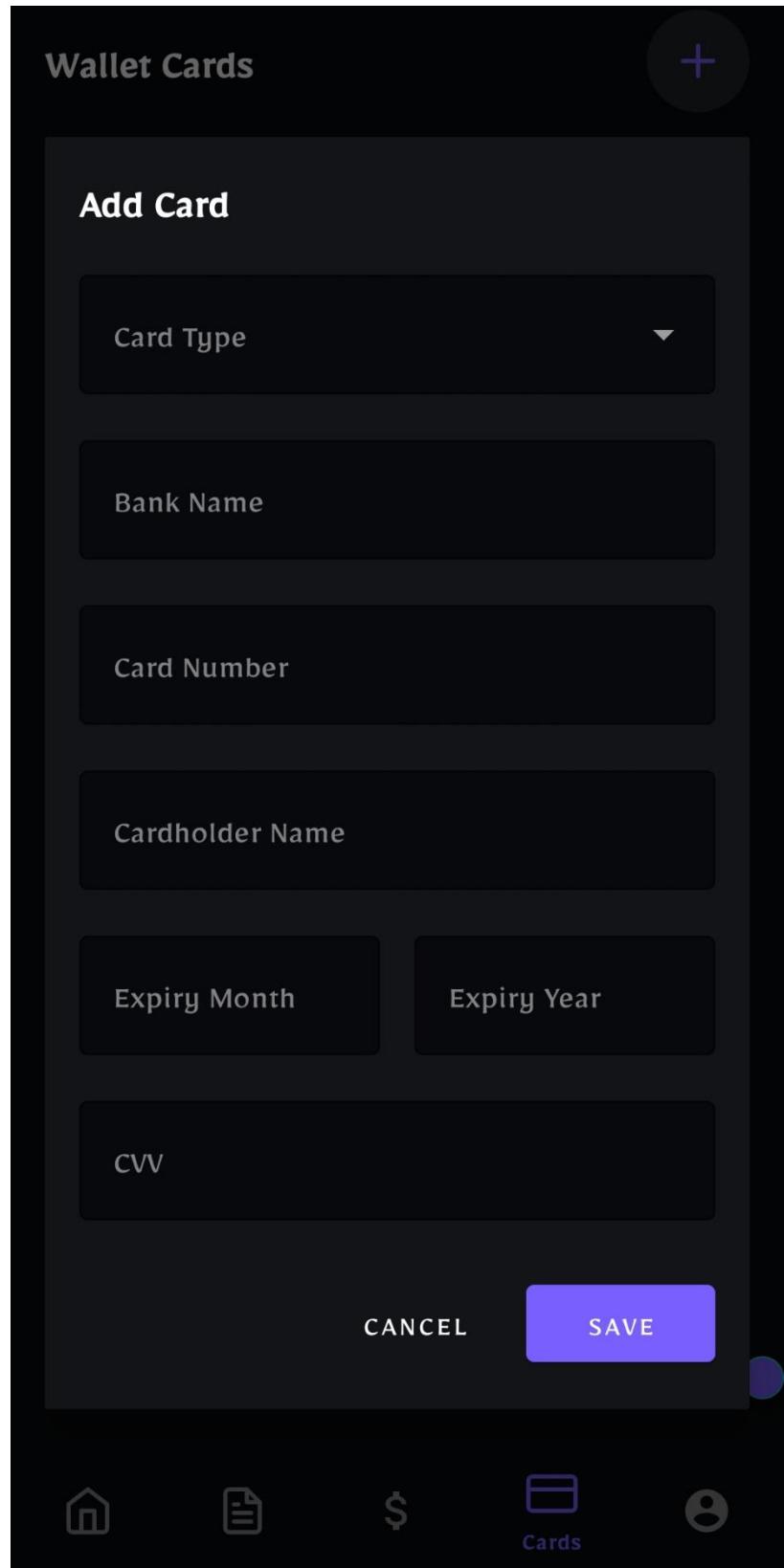


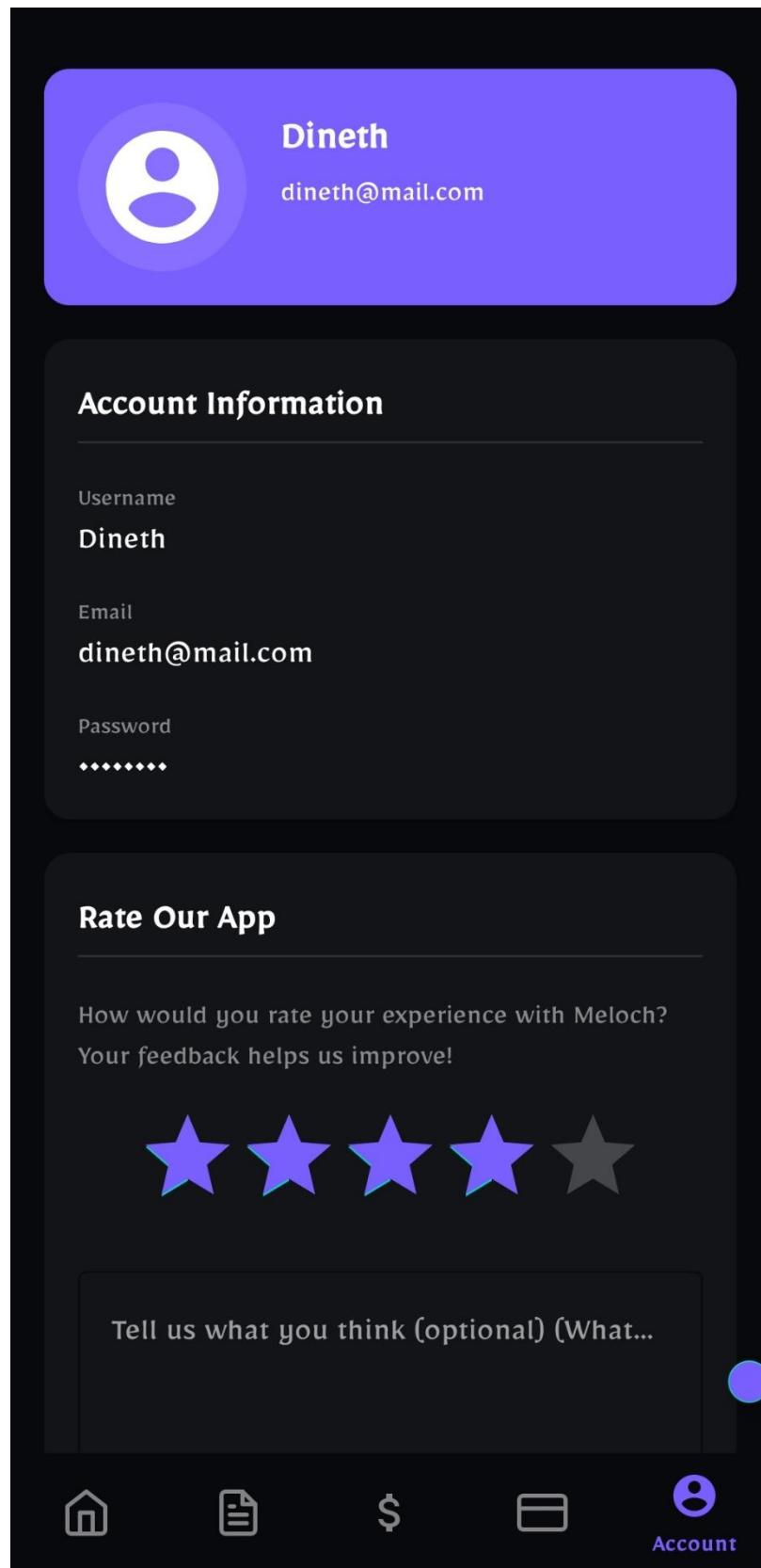












How would you rate your experience with Meloch?
Your feedback helps us improve!



Tell us what you think (optional) (What...

SUBMIT REVIEW

Account Actions

EXPORT FINANCIAL REPORT (PDF)

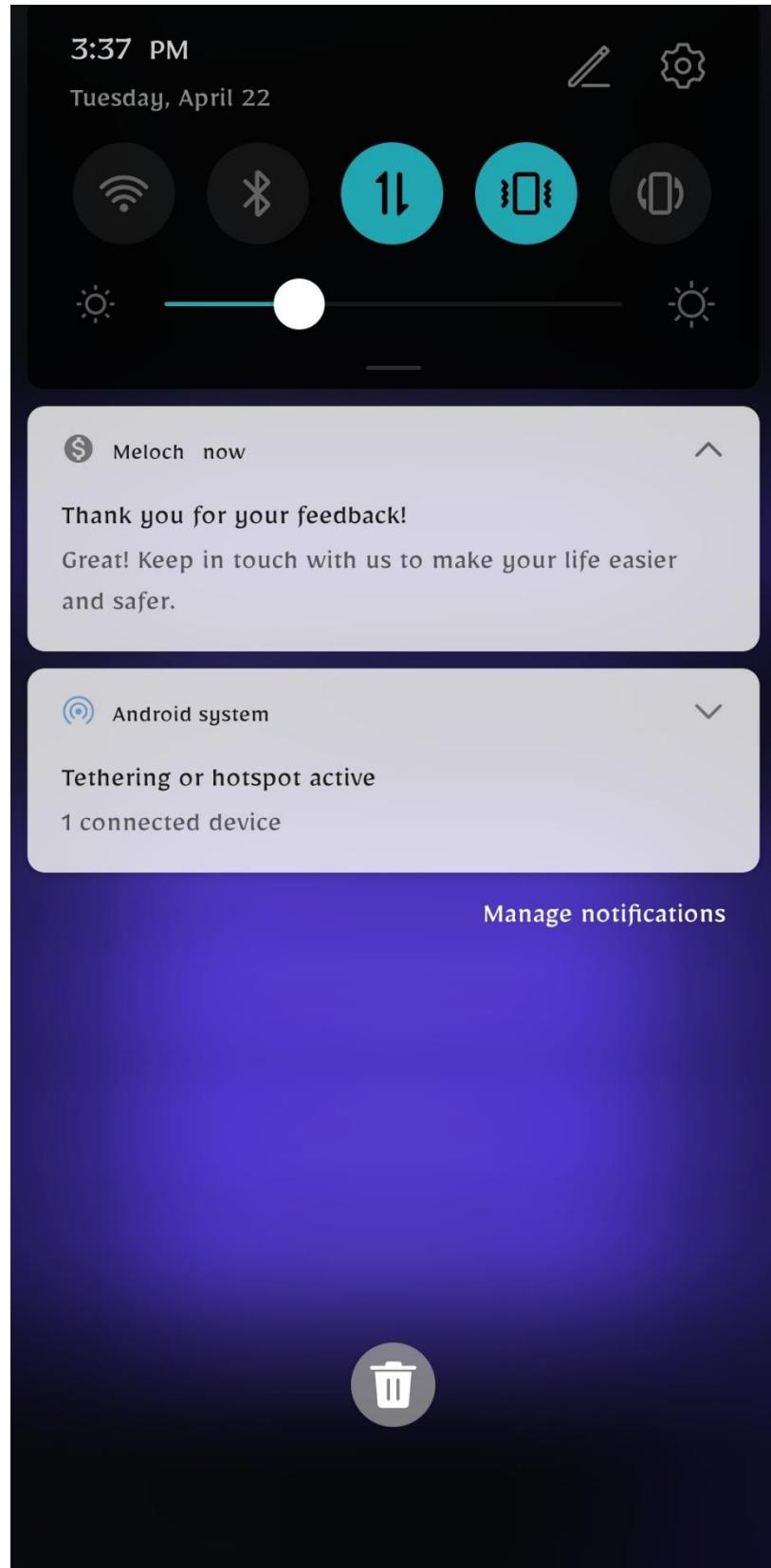
BACKUP DATA (JSON)

LOGOUT

DELETE MY ACCOUNT



Account



Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income:
LKR350,000.00 | Expenses: LKR142,650.64

Add Record



EXPENSE

INCOME

Expense

-LKR400.32

Category

Transport



Payment Method

Cash



Date & Time

Apr 22, 2025



ADD RECORD



Home



File



\$



Card



User

Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income:
LKR350,000.00 | Expenses: LKR142,650.64

Add Record



Select Category



Food



Entertainment



Transport



Health



Shopping



Vacation

ADD RECORD



Home



Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income:
LKR350,000.00 | Expenses: LKR142,650.64

Add Record



EXPENSE

INCOME

Income

+LKR400.32

Category

Select Category



Payment Method

Cash



Date & Time

Apr 22, 2025

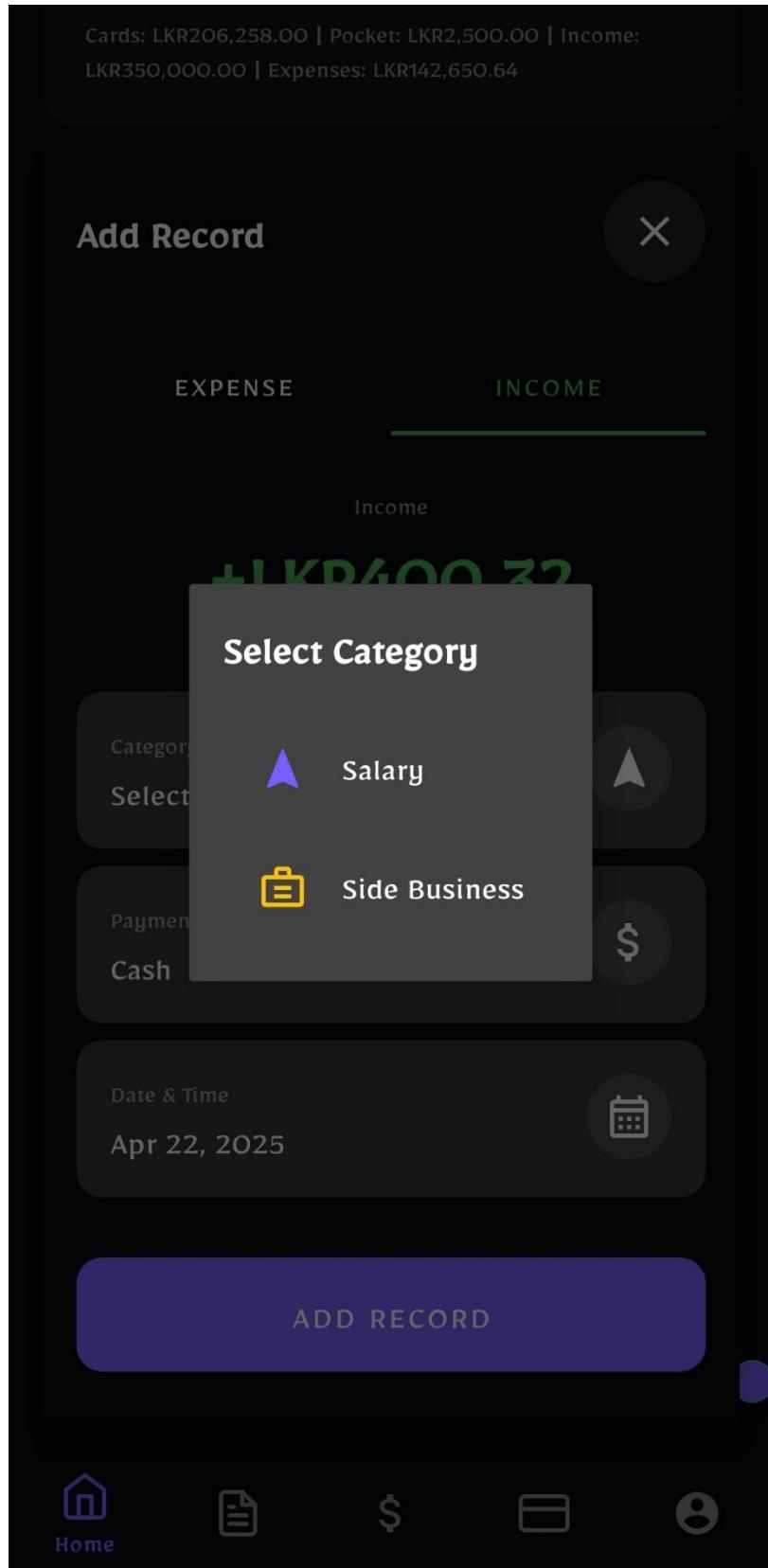


ADD RECORD



Home





Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income:
LKR350,000.00 | Expenses: LKR142,650.64

Add Record

X

EXPENSE

INCOME

Expense

LKR 00.00

Select Payment Method

Cash

Credit Card

Debit Card

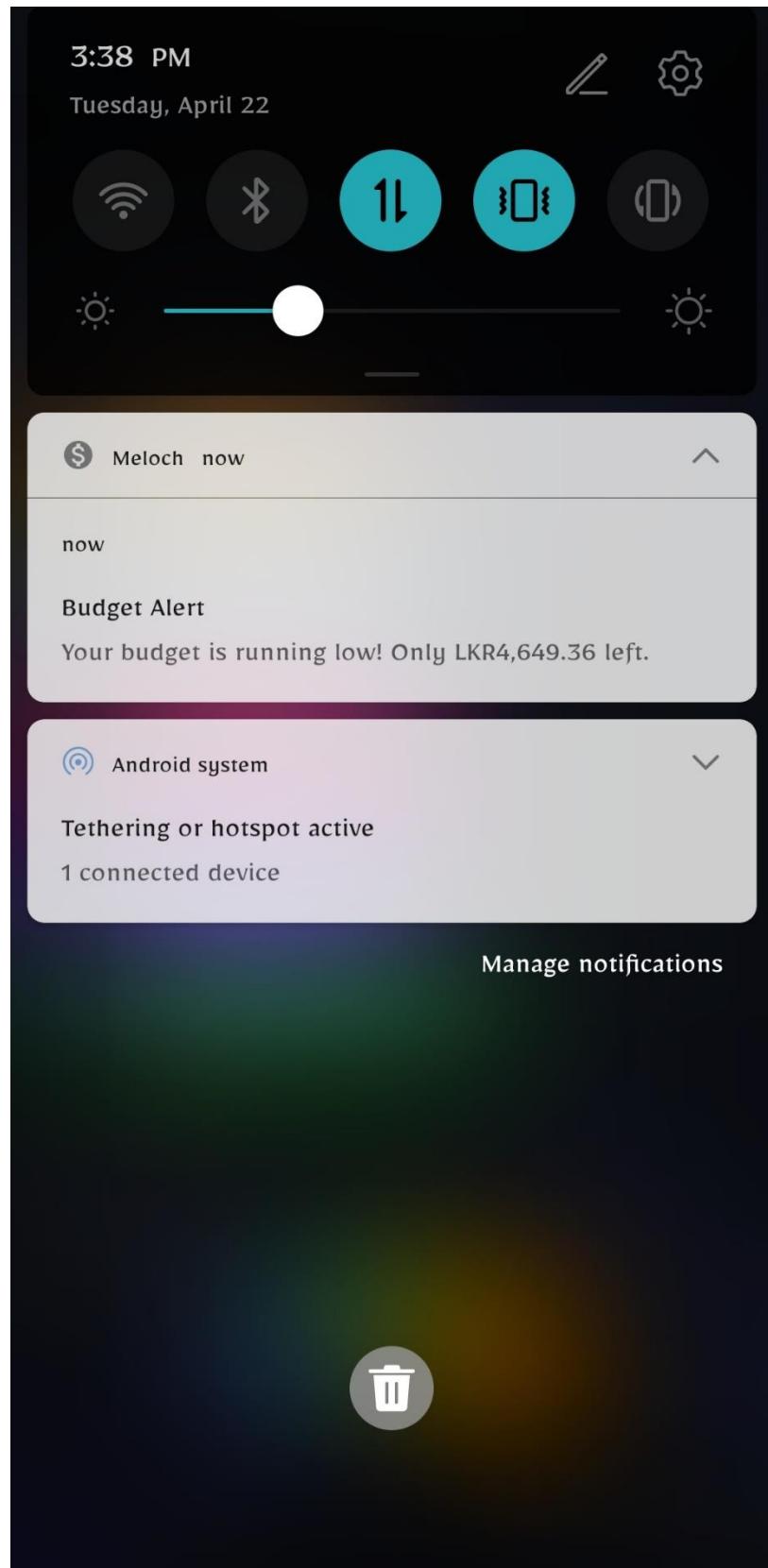
Date & Time

Apr 22, 2025



ADD RECORD





How would you rate your experience with Meloch?

Your feedback helps us improve!



Tell us what you think (optional)

SUBMIT REVIEW

Report Generated

Your financial report has been
successfully generated and saved to:

Documents/Meloch/Reports/

SHARE

CLOSE

VIEW

BACKUP DATA (JSON)

LOGOUT

DELETE MY ACCOUNT



Account

MELOCH FINANCIAL REPORT

Monthly Summary - April 2025

Financial Summary

Total Income:	LKR358,549.64
Total Expenses:	LKR151,049.64
Balance:	LKR207,500.00
Budget Left:	LKR7,500.00

Recent Transactions

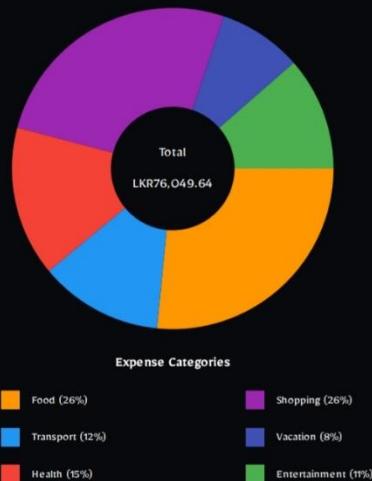
Date & Time	Category	Type	Amount
22 Apr 2025, 15:43	Side Business	INCOME	+LKR7,499.00
22 Apr 2025, 15:42	Transport	EXPENSE	-LKR99.00
22 Apr 2025, 15:41	Side Business	INCOME	+LKR99.00
22 Apr 2025, 15:40	Health	EXPENSE	-LKR1,000.00
22 Apr 2025, 15:40	Side Business	INCOME	+LKR551.32
22 Apr 2025, 15:39	Side Business	INCOME	+LKR400.32
22 Apr 2025, 15:38	Shopping	EXPENSE	-LKR6,800.00
22 Apr 2025, 15:37	Food	EXPENSE	-LKR2,700.00
21 Apr 2025, 15:14	Health	EXPENSE	-LKR1,500.32
21 Apr 2025, 15:14	Vacation	EXPENSE	-LKR400.32

Expense Breakdown by Category

Category	Amount	% of Total
Food	LKR20,200.00	26.6%
Transport	LKR9,449.00	12.4%
Health	LKR11,500.32	15.1%
Shopping	LKR19,800.00	26.0%
Vacation	LKR6,400.32	8.4%
Entertainment	LKR8,700.00	11.4%
TOTAL	LKR76,049.64	100%

Entertainment	LKR8,700.00	11.4%
TOTAL	LKR76,049.64	100%

Expense Distribution Chart



How would you rate your experience with Meloch?
Your feedback helps us improve!



Tell us what you think (optional)

SUBMIT REVIEW

Backup Created

Your data has been successfully backed up and saved to:

Documents/Meloch/Backup/

CLOSE SHARE

BACKUP DATA (JSON)

LOGOUT

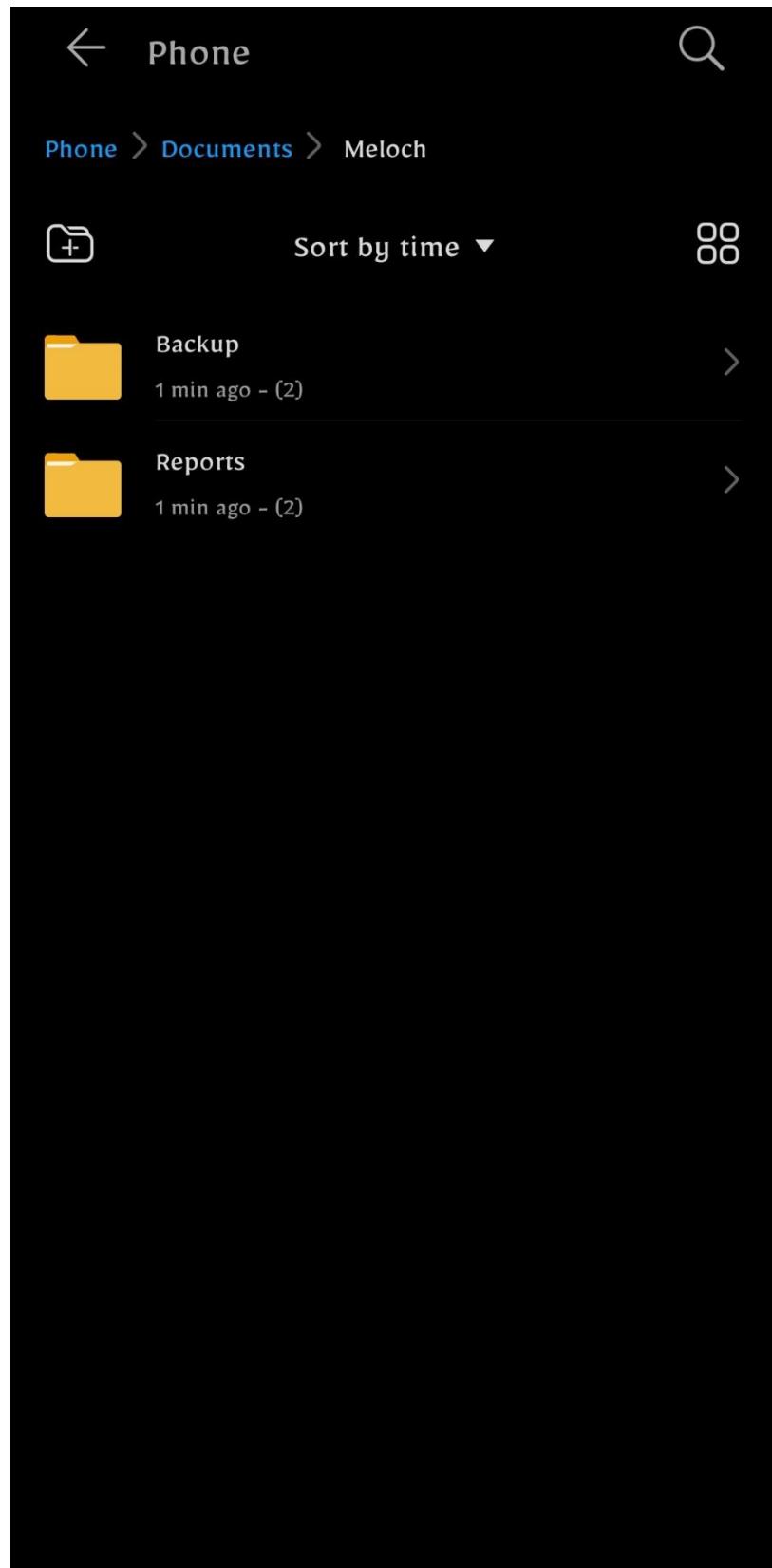
DELETE MY ACCOUNT

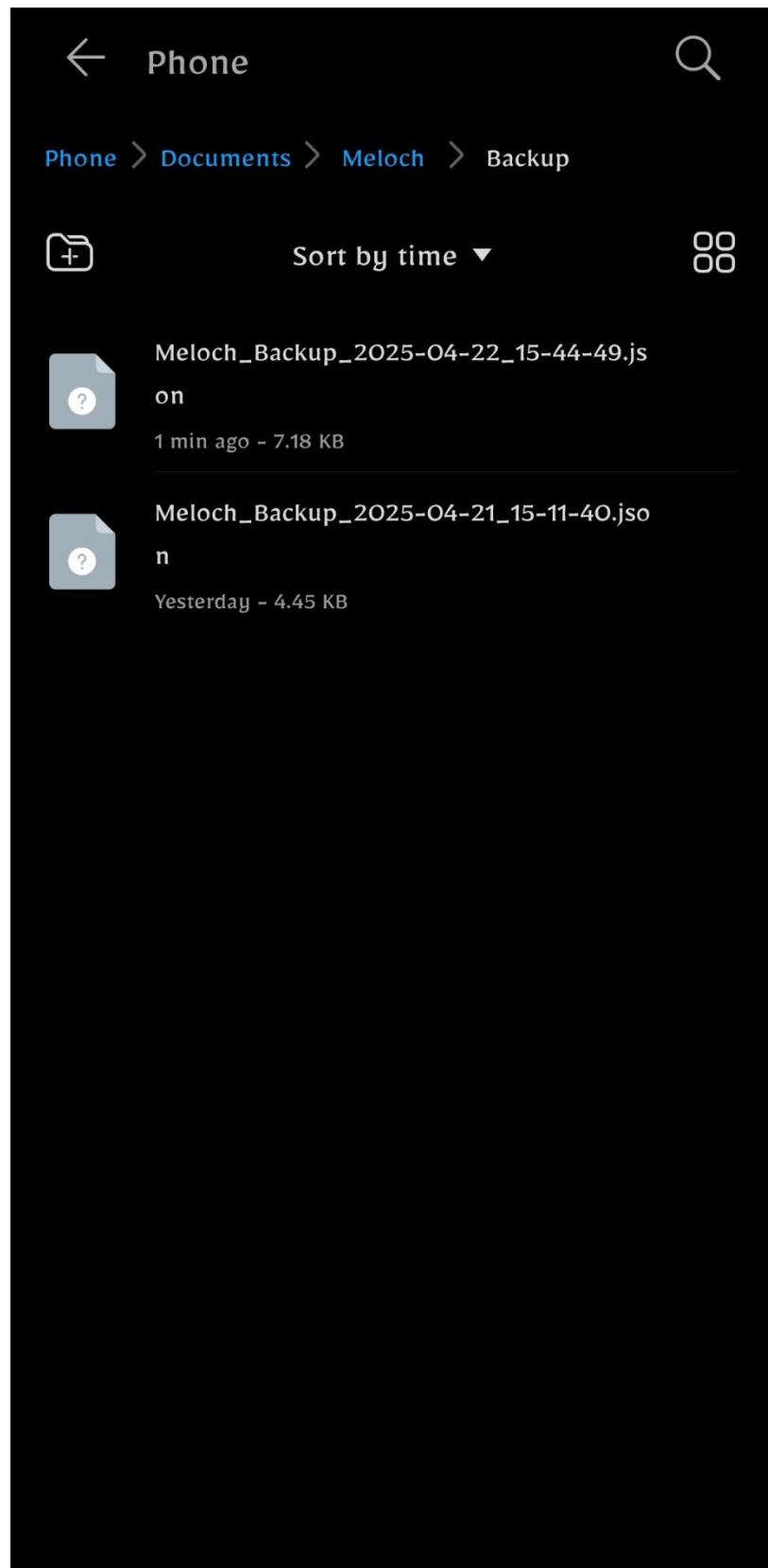


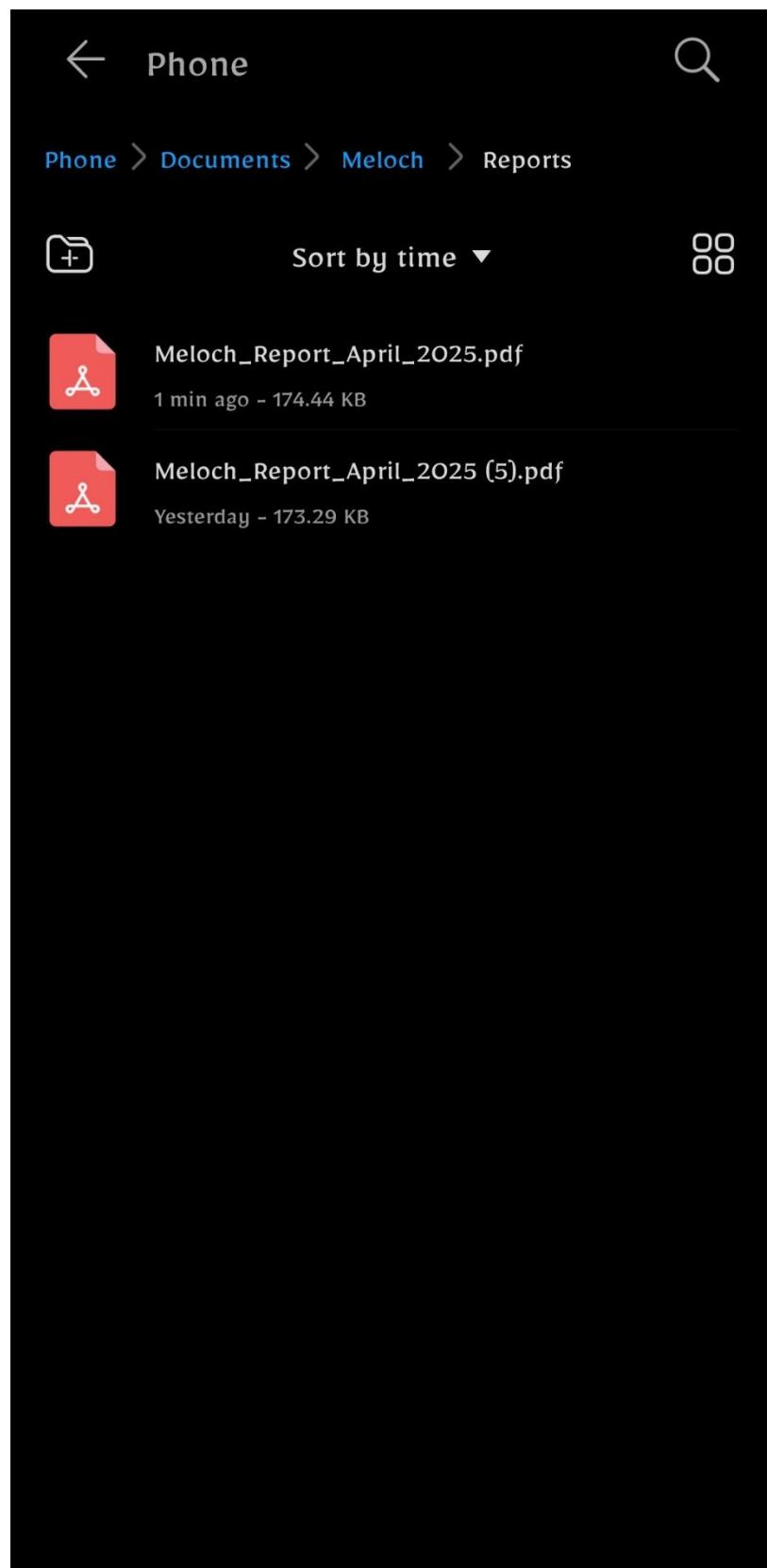
Account

← Meloch_Backup_2025-04-22_15-44-49.json

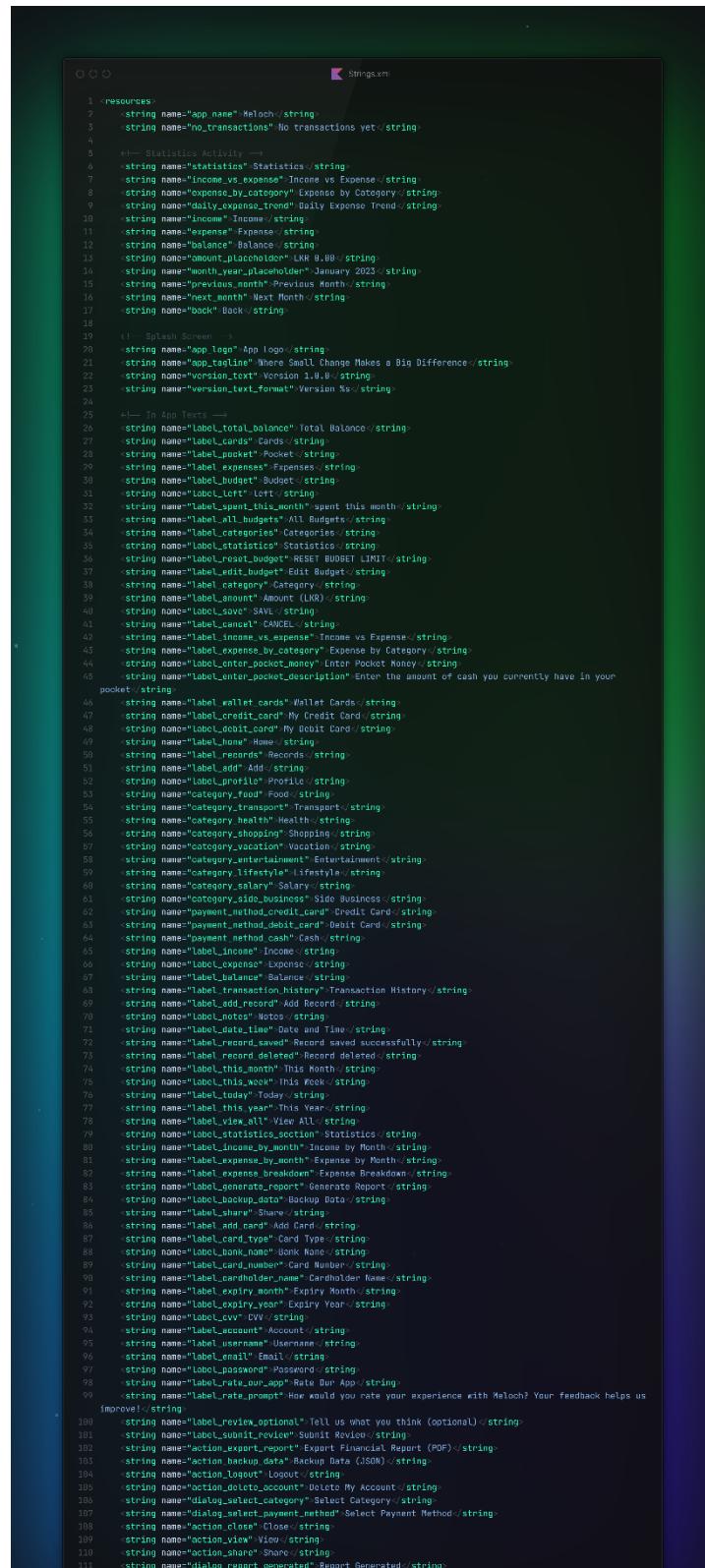
```
{  
  "metadata": {  
    "app": "Meloch",  
    "version": "1.0",  
    "timestamp": 1745316889674,  
    "date": "2025-04-22 15:44:49"  
  },  
  "user": {  
    "username": "Dineth",  
    "email": "dineth@mail.com"  
  },  
  "financial": {  
    "totalBalance": 107500,  
    "pocketMoney": 2500,  
    "budgetLeft": 7500,  
    "totalBudget": 58549.640625,  
    "budgetCategories": {  
      "entertainment": 20000,  
      "food": 17500,  
      "transport": 5000,  
      "lifestyle": 7500  
    }  
  },  
  "cards": [  
    {  
      "id": "8a392e8e-c3d5-4a53-88e1-9116d0334",  
      "cardNumber": "4605592041583306",  
      "cardholderName": "D G A D HIRUSHA",  
      "expiryMonth": 9,  
      "expiryYear": 2030,  
      "expiryDate": "09/30",  
      "cvv": "132",  
      "type": "DEBIT",  
      "balance": 72341  
    },  
    {  
      "id": "b163dbfc-7afa-4c2f-a3b6-3bd1c8ff4",  
      "cardNumber": "4605532180805147",  
      "cardholderName": "D G A D HIRUSHA",  
      "expiryMonth": 12,  
      "expiryYear": 2029,  
      "expiryDate": "12/29",  
      "cvv": "028",  
      "type": "CREDIT",  
    }  
  ]  
}
```







Strings XML



The screenshot shows a code editor window with the title "Strings.xml". The file contains a large amount of XML code defining various strings used in the application. The strings are categorized by resource type and include labels for wallet cards, transaction history, budgeting, and user account information.

```
<resources>
    <string name="app_name">MeLoch</string>
    <string name="no_transactions">No transactions yet</string>
    ...
    <!-- Statistics Labels -->
    <string name="statistics">Statistics</string>
    <string name="income_vs_expense">Income vs Expense</string>
    <string name="expense_by_category">Expense by Category</string>
    <string name="daily_expense_trend">Daily Expense Trend</string>
    <string name="income">Income</string>
    <string name="expense">Expense</string>
    <string name="balance">Balance</string>
    <string name="amount_placeholder">LKR 0.00</string>
    <string name="month_year_placeholder">January 2023</string>
    <string name="previous_month">Previous Month</string>
    <string name="next_month">Next Month</string>
    <string name="back">Back</string>
    ...
    <!-- Splash Screen -->
    <string name="app_logo">App logo</string>
    <string name="app_tagline">Where Small Change Makes a Big Difference</string>
    <string name="version_text">Version 1.0.0</string>
    <string name="version_text_format">Version %s</string>
    ...
    <!-- In App Terms -->
    <string name="label_total_balance">Total Balance</string>
    <string name="label_cards">Cards</string>
    <string name="label_pocket">Pocket</string>
    <string name="label_expenses">Expenses</string>
    <string name="label_budget">Budget</string>
    <string name="label_left">Left</string>
    <string name="label_spent_this_month">Spent this month</string>
    <string name="label_all_budgets">All Budgets</string>
    <string name="label_categories">Categories</string>
    <string name="label_statistics">Statistics</string>
    <string name="label_reset_budget">RESET BUDGET LIMIT</string>
    <string name="label_edit_budget">Edit Budget</string>
    <string name="label_category">Category</string>
    <string name="label_amount">Amount (LKR)</string>
    <string name="label_save">Save</string>
    <string name="label_cancel">CANCEL</string>
    <string name="label_income">Income vs Expense</string>
    <string name="label_expense_by_category">Expense by Category</string>
    <string name="label_enter_pocket_money">Enter Pocket Money</string>
    <string name="label_under_pocket_description">Enter the amount of cash you currently have in your pocket</string>
    <string name="label_wallet_cards">Wallet Cards</string>
    <string name="label_credit_card">My Credit Card</string>
    <string name="label_debit_card">My Debit Card</string>
    <string name="label_home">Home</string>
    <string name="label_records">Records</string>
    <string name="label_add">Add</string>
    <string name="label_profile">Profile</string>
    <string name="category_food">Food</string>
    <string name="category_transport">Transport</string>
    <string name="category_health">Health</string>
    <string name="category_shopping">Shopping</string>
    <string name="category_vacation">Vacation</string>
    <string name="category_entertainment">Entertainment</string>
    <string name="category_lifestyle">Lifestyle</string>
    <string name="category_salary">Salary</string>
    <string name="category_side_business">Side Business</string>
    <string name="payment_method_credit_card">Credit Card</string>
    <string name="payment_method_debit_card">Debit Card</string>
    <string name="payment_method_cash">Cash</string>
    <string name="label_income">Income</string>
    <string name="label_expense">Expense</string>
    <string name="label_balance">Balance</string>
    <string name="label_transaction_history">Transaction History</string>
    <string name="label_add_record">Add Record</string>
    <string name="label_notes">Notes</string>
    <string name="label_date_time">Date and Time</string>
    <string name="label_record_saved">Record saved successfully</string>
    <string name="label_record_deleted">Record deleted</string>
    <string name="label_this_month">This Month</string>
    <string name="label_today">Today</string>
    <string name="label_this_year">This Year</string>
    <string name="label_view_all">View All</string>
    <string name="label_statistics_section">Statistics</string>
    <string name="label_income_by_month">Income by Month</string>
    <string name="label_expense_by_month">Expense by Month</string>
    <string name="label_expense_breakdown">Expense Breakdown</string>
    <string name="label_generate_report">Generate Report</string>
    <string name="label_backup_data">Backup Data</string>
    <string name="label_share">Share</string>
    <string name="label_add_card">Add Card</string>
    <string name="label_card_type">Card Type</string>
    <string name="label_bank_name">Bank Name</string>
    <string name="label_card_number">Card Number</string>
    <string name="label_cardholder_name">Cardholder Name</string>
    <string name="label_expiry_month">Expiry Month</string>
    <string name="label_expiry_year">Expiry Year</string>
    <string name="label_cvv">CVV</string>
    <string name="label_username">Username</string>
    <string name="label_email">Email</string>
    <string name="label_password">Password</string>
    <string name="label_rate_our_app">Rate Our App</string>
    <string name="label_rate_prompt">How would you rate your experience with MeLoch? Your feedback helps us improve!</string>
    <string name="label_review_optional">Tell us what you think (optional)</string>
    <string name="label_submit_review">Submit Review</string>
    <string name="action_export_report">Export Financial Report (PDF)</string>
    <string name="action_hackin_data">Backup Data (JSON)</string>
    <string name="action_logout">Logout</string>
    <string name="action_delete_account">Delete My Account</string>
    <string name="dialog_select_category">Select Category</string>
    <string name="dialog_select_payment_method">Select Payment Method</string>
    ...
    <string name="action_close">Close</string>
    <string name="action_view">View</string>
    <string name="action_share">Share</string>
    <string name="dialog_report_generated">Report Generated</string>

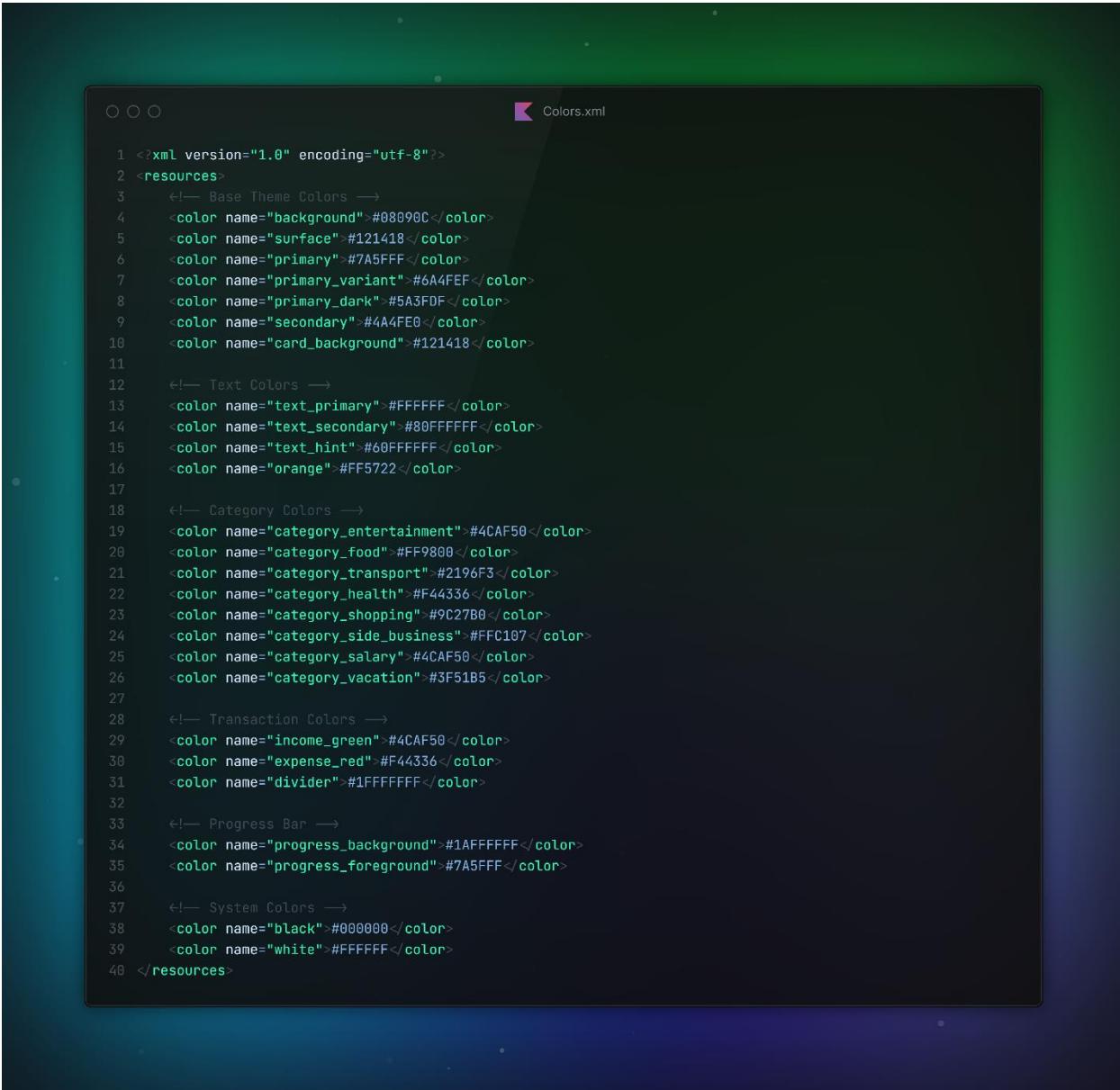
```

```

4   <!-- Statistics Activity -->
5   string name="statistics" Statistics</string>
6   string name="income_vs_expense" Income vs Expense</string>
7   string name="expense_by_category" Expense by Category</string>
8   string name="daily_expense_trend" Daily Expense Trend</string>
9   string name="income" Income</string>
10  string name="expense" Expense</string>
11  string name="balance" Balance</string>
12  string name="amount_placeholder" LKR 8.88</string>
13  string name="month_year_placeholder" January 2023</string>
14  string name="previous_month" Previous Month</string>
15  string name="next_month" Next Month</string>
16  string name="back" Back</string>
17
18 (<!-- Splash Screen -->
19  string name="app_logo" App logo</string>
20  string name="app_tagline" Where Small Change Makes a Big Difference</string>
21  string name="version_text" Version 1.0.0</string>
22  string name="version_text_format" Version %s</string>
23
24  <!-- To Do Texts -->
25  string name="label_total_balance" Total Balance</string>
26  string name="label_cards" Cards</string>
27  string name="label_pocket" Pocket</string>
28  string name="label_expenses" Expenses</string>
29  string name="label_budget" Budget</string>
30  string name="label_left_left" Left</string>
31  string name="label_spent_this_month" spent this month</string>
32  string name="label_all_budgets" All Budgets</string>
33  string name="label_categories" Categories</string>
34  string name="label_statistics" Statistics</string>
35  string name="label_reset_budget" RESET BUDGET LIMIT</string>
36  string name="label_edit_budget" Edit Budget</string>
37  string name="label_category" Category</string>
38  string name="label_amount" Amount (LKR)</string>
39  string name="label_cancel" CANCEL</string>
40  string name="label_continue" Continue</string>
41  string name="label_causal" CANCEL</string>
42  string name="label_income_vs_expense" Income vs Expense</string>
43  string name="label_expense_by_category" Expense by Category</string>
44  string name="label_enter_pocket_money" Enter Pocket Money</string>
45  string name="label_enter_pocket_description" Enter the amount of cash you currently have in your
46  pocket</string>
47  string name="label_wallet_cards" Wallet Cards</string>
48  string name="label_credit_card" My Credit Card</string>
49  string name="label_debit_card" My Debit Card</string>
50  string name="label_home" Home</string>
51  string name="label_records" Records</string>
52  string name="label_add" Add</string>
53  string name="label_profile" Profile</string>
54  string name="category_food" Food</string>
55  string name="category_transport" Transport</string>
56  string name="category_health" Health</string>
57  string name="category_shopping" Shopping</string>
58  string name="category_entertainment" Entertainment</string>
59  string name="category_lifestyle" Lifestyle</string>
60  string name="category_salary" Salary</string>
61  string name="category_business" Sino Business</string>
62  string name="payment_method_credit_card" Credit Card</string>
63  string name="payment_method_debit_card" Debit Card</string>
64  string name="payment_method_cash" Cash</string>
65  string name="label_income" Income</string>
66  string name="label_expense" Expense</string>
67  string name="label_balance" Balance</string>
68  string name="label_transaction_history" Transaction History</string>
69  string name="label_add_record" Add Record</string>
70  string name="label_notes" Notes</string>
71  string name="label_date_time" Date and Time</string>
72  string name="label_record_saved" Record saved successfully</string>
73  string name="label_record_deleted" Record deleted</string>
74  string name="label_this_month" This Month</string>
75  string name="label_last_week" Last Week</string>
76  string name="label_today" Today</string>
77  string name="label_this_year" This Year</string>
78  string name="label_view_all" View All</string>
79  string name="label_statistics_section" Statistics</string>
80  string name="label_income_by_month" Income by Month</string>
81  string name="label_expense_by_month" Expense by Month</string>
82  string name="label_expense_breakdown" Expense Breakdown</string>
83  string name="label_generate_report" Generate Report</string>
84  string name="label_backup_data" Backup Data</string>
85  string name="label_share" Share</string>
86  string name="label_add_card" Add Card</string>
87  string name="label_card_type" Card Type</string>
88  string name="label_bank_name" Bank Name</string>
89  string name="label_card_number" Card Number</string>
90  string name="label_cardholder_name" Cardholder Name</string>
91  string name="label_expiry_month" Expiry Month</string>
92  string name="label_expiry_year" Expiry Year</string>
93  string name="label_cvv" CVV</string>
94  string name="label_username" Username</string>
95  string name="label_email" Email</string>
96  string name="label_password" Password</string>
97  string name="label_rate_app" Rate Our App</string>
98  string name="label_rate_prompt" How would you rate your experience with Maloch? Your feedback helps us
99  improve!</string>
100 string name="label_review_optional" Tell us what you think (optional)</string>
101 string name="action_submit_review" Submit Review</string>
102 string name="action_export_report" Export Financial Report (PDF)</string>
103 string name="action_backup_data" Backup Data (JSON)</string>
104 string name="action_logout" Logout</string>
105 string name="action_delete_account" Delete My Account</string>
106 string name="dialog_select_category" Select Category</string>
107 string name="dialog_select_payment_method" Select Payment Method</string>
108 string name="action_close" Close</string>
109 string name="action_view" View</string>
110 string name="action_share" Share</string>
111 string name="dialog_report_generated" Report Generated</string>
112 string name="message_report_generated" Your financial report has been successfully generated and saved
113 to</string>
113 string name="dialog_backup_created" Backup Created</string>
114 string name="message_backup_created" Your data has been successfully backed up and saved to</string>
115 </resources>

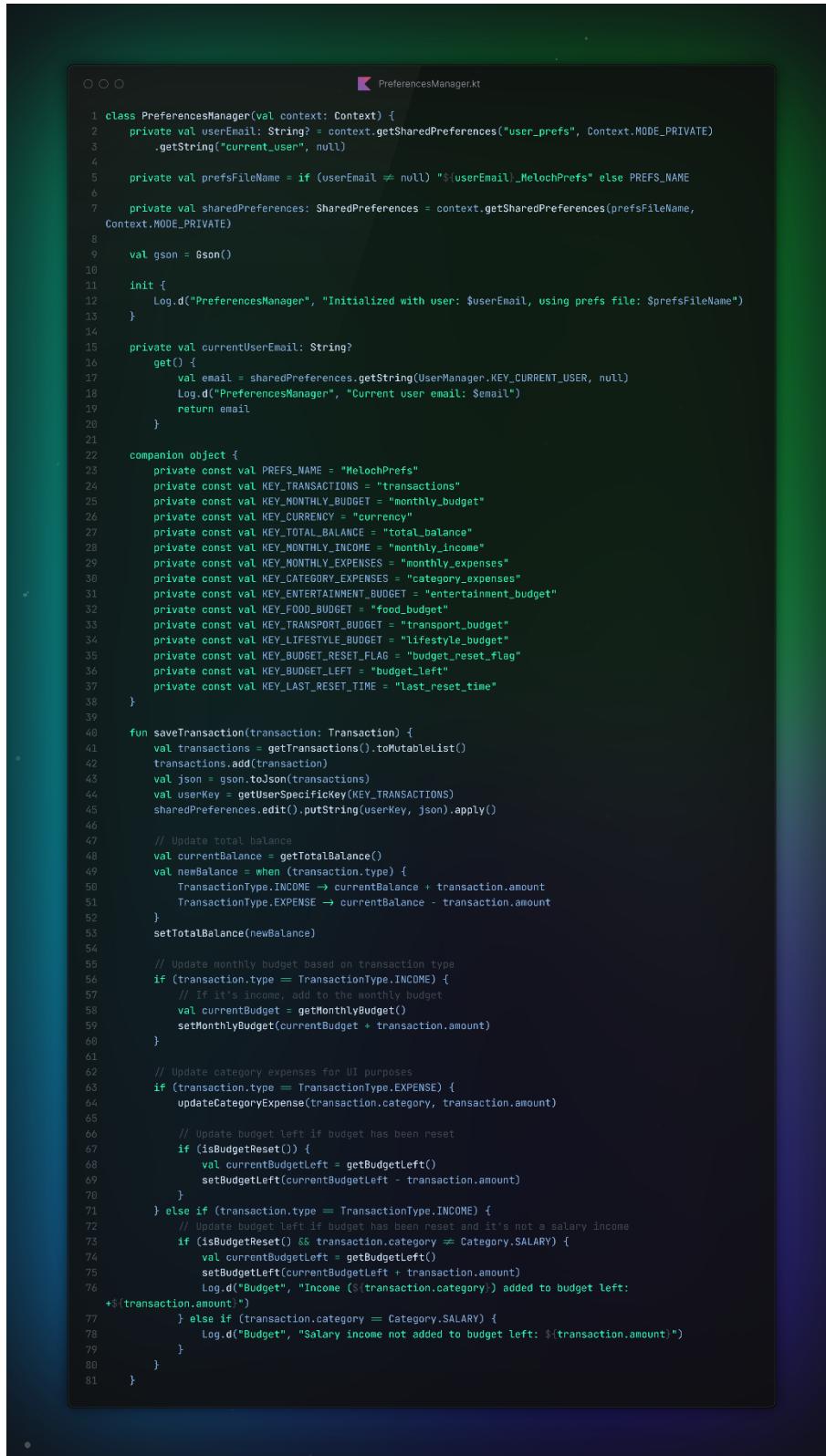
```

Colors XML

A screenshot of a code editor window titled "Colors.xml". The file contains XML code defining various colors used in a theme. The code is color-coded: comments are in light blue, strings are in light green, and tags are in white. The file includes sections for Base Theme Colors, Text Colors, Category Colors, Transaction Colors, Progress Bar Colors, and System Colors.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <!-- Base Theme Colors -->
4     <color name="background">#08090C</color>
5     <color name="surface">#121418</color>
6     <color name="primary">#7A5FFF</color>
7     <color name="primary_variant">#6A4FEF</color>
8     <color name="primary_dark">#5A3FDF</color>
9     <color name="secondary">#4A4FE0</color>
10    <color name="card_background">#121418</color>
11
12    <!-- Text Colors -->
13    <color name="text_primary">#FFFFFF</color>
14    <color name="text_secondary">#80FFFFFF</color>
15    <color name="text_hint">#60FFFFFF</color>
16    <color name="orange">#FF5722</color>
17
18    <!-- Category Colors -->
19    <color name="category_entertainment">#4CAF50</color>
20    <color name="category_food">#FF9800</color>
21    <color name="category_transport">#2196F3</color>
22    <color name="category_health">#F44336</color>
23    <color name="category_shopping">#9C27B0</color>
24    <color name="category_side_business">#FFC107</color>
25    <color name="category_salary">#4CAF50</color>
26    <color name="category_vacation">#3F51B5</color>
27
28    <!-- Transaction Colors -->
29    <color name="income_green">#4CAF50</color>
30    <color name="expense_red">#F44336</color>
31    <color name="divider">#1FFFFFFF</color>
32
33    <!-- Progress Bar -->
34    <color name="progress_background">#1AFFFFFF</color>
35    <color name="progress_foreground">#7A5FFF</color>
36
37    <!-- System Colors -->
38    <color name="black">#000000</color>
39    <color name="white">#FFFFFF</color>
40 </resources>
```

SharedPreferences Kt



The screenshot shows the PreferencesManager.kt file in an Android Studio code editor. The code is written in Kotlin and manages SharedPreferences for a budgeting application. It includes constants for keys like monthly income, expenses, and budget, and logic for saving transactions, updating balances, and managing monthly budgets.

```
1 class PreferencesManager(val context: Context) {
2     private val userEmail: String? = context.getSharedPreferences("user_prefs", Context.MODE_PRIVATE)
3         .getString("current_user", null)
4
5     private val prefsFileName = if (userEmail != null) "${userEmail}_MelochPrefs" else PREFS_NAME
6
7     private val sharedPreferences: SharedPreferences = context.getSharedPreferences(prefsFileName,
8         Context.MODE_PRIVATE)
9
10    val gson = Gson()
11
12    init {
13        Log.d("PreferencesManager", "Initialized with user: $userEmail, using prefs file: $prefsFileName")
14    }
15
16    private val currentUserEmail: String?
17        get() {
18            val email = sharedPreferences.getString(UserManager.KEY_CURRENT_USER, null)
19            Log.d("PreferencesManager", "Current user email: $email")
20            return email
21        }
22
23    companion object {
24        private const val PREFS_NAME = "MelochPrefs"
25        private const val KEY_TRANSACTIONS = "transactions"
26        private const val KEY_MONTHLY_BUDGET = "monthly_budget"
27        private const val KEY_CURRENCY = "currency"
28        private const val KEY_TOTAL_BALANCE = "total_balance"
29        private const val KEY_MONTHLY_INCOME = "monthly_income"
30        private const val KEY_MONTHLY_EXPENSES = "monthly_expenses"
31        private const val KEY_CATEGORY_EXPENSES = "category_expenses"
32        private const val KEY_ENTERTAINMENT_BUDGET = "entertainment_budget"
33        private const val KEY_FOOD_BUDGET = "food_budget"
34        private const val KEY_TRANSPORT_BUDGET = "transport_budget"
35        private const val KEY_LIFESTYLE_BUDGET = "lifestyle_budget"
36        private const val KEY_BUDGET_RESET_FLAG = "budget_reset_flag"
37        private const val KEY_BUDGET_LEFT = "budget_left"
38        private const val KEY_LAST_RESET_TIME = "last_reset_time"
39    }
39
40    fun saveTransaction(transaction: Transaction) {
41        val transactions = getTransactions().toMutableList()
42        transactions.add(transaction)
43        val json = gson.toJson(transactions)
44        val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
45        sharedPreferences.edit().putString(userKey, json).apply()
46
47        // Update total balance
48        val currentBalance = getTotalBalance()
49        val newBalance = when (transaction.type) {
50            TransactionType.INCOME → currentBalance + transaction.amount
51            TransactionType.EXPENSE → currentBalance - transaction.amount
52        }
53        setTotalBalance(newBalance)
54
55        // Update monthly budget based on transaction type
56        if (transaction.type == TransactionType.INCOME) {
57            // If it's income, add to the monthly budget
58            val currentBudget = getMonthlyBudget()
59            setMonthlyBudget(currentBudget + transaction.amount)
60        }
61
62        // Update category expenses for UI purposes
63        if (transaction.type == TransactionType.EXPENSE) {
64            updateCategoryExpense(transaction.category, transaction.amount)
65
66            // Update budget left if budget has been reset
67            if (isBudgetReset()) {
68                val currentBudgetLeft = getBudgetLeft()
69                setBudgetLeft(currentBudgetLeft - transaction.amount)
70            }
71        } else if (transaction.type == TransactionType.INCOME) {
72            // Update budget left if budget has been reset and it's not a salary income
73            if (isBudgetReset() && transaction.category == Category.SALARY) {
74                val currentBudgetLeft = getBudgetLeft()
75                setBudgetLeft(currentBudgetLeft + transaction.amount)
76                Log.d("Budget", "Income ${transaction.category} added to budget left: ${transaction.amount}")
77            } else if (transaction.category == Category.SALARY) {
78                Log.d("Budget", "Salary income not added to budget left: ${transaction.amount}")
79            }
80        }
81    }
}
```

```
○ ○ ○ PreferencesManager.kt

1 fun getTransactions(): List<Transaction> {
2     // Get user-specific key
3     val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
4     val json = sharedPreferences.getString(userKey, "[[]")
5     val type = object : TypeToken<List<Transaction>>() {}.type
6     return gson.fromJson(json, type) ?: emptyList()
7 }
8
9 private fun getUserSpecificKey(baseKey: String): String {
10     val email = currentUserEmail
11     val result = if (email != null) {
12         "$email_$baseKey"
13     } else {
14         baseKey
15     }
16
17     // Add debug logging to track key generation
18     Log.d("UserSpecificKey", "Generated key: $result from base key: $baseKey for user: $email")
19
20     return result
21 }
22
23 fun saveTransactions(transactions: List<Transaction>) {
24     val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
25     val json = gson.toJson(transactions)
26     sharedPreferences.edit().putString(userKey, json).apply()
27 }
28
29 fun setMonthlyBudget(amount: Double) {
30     val userKey = getUserSpecificKey(KEY_MONTHLY_BUDGET)
31     sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
32 }
33
34 fun getMonthlyBudget(): Double {
35     val userKey = getUserSpecificKey(KEY_MONTHLY_BUDGET)
36     // Default to 21000 (sum of default category budgets) if not set
37     return sharedPreferences.getFloat(userKey, 21000f).toDouble()
38 }
39
40 fun setCurrency(currency: String) {
41     val userKey = getUserSpecificKey(KEY_CURRENCY)
42     sharedPreferences.edit().putString(userKey, currency).apply()
43 }
44
45 fun getCurrency(): String {
46     val userKey = getUserSpecificKey(KEY_CURRENCY)
47     return sharedPreferences.getString(userKey, "LKR") ?: "LKR"
48 }
49
50 fun setTotalBalance(balance: Double) {
51     val userKey = getUserSpecificKey(KEY_TOTAL_BALANCE)
52     sharedPreferences.edit().putFloat(userKey, balance.toFloat()).apply()
53 }
54
55 fun getTotalBalance(): Double {
56     val userKey = getUserSpecificKey(KEY_TOTAL_BALANCE)
57     return sharedPreferences.getFloat(userKey, 0f).toDouble()
58 }
59
60 fun getMonthlyExpenses(): Double {
61     val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
62     val currentYear = Calendar.getInstance().get(Calendar.YEAR)
63
64     return getTransactions()
65         .filter { transaction ->
66             val cal = Calendar.getInstance().apply { time = transaction.date }
67             cal.get(Calendar.MONTH) == currentMonth &&
68             cal.get(Calendar.YEAR) == currentYear &&
69             transaction.type == TransactionType.EXPENSE
70         }
71         .sumOf { it.amount }
72 }
```

```
○ ○ ○ PreferencesManager.kt

1 fun resetBudget() {
2     // Get the current monthly budget and expenses
3     val currentBudget = getMonthlyBudget()
4     val currentExpenses = getMonthlyExpenses()
5
6     // Reset category expenses
7     for (category in Category.values()) {
8         setCategoryExpense(category, 0.0)
9     }
10
11    // Set the budget left value to equal the total budget
12    setBudgetLeft(currentBudget)
13
14    // Store the current timestamp as the last reset time
15    setLastResetTime(System.currentTimeMillis())
16
17    // Set the budget reset flag to true
18    setBudgetResetFlag(true)
19
20    // Verify the flag was set
21    val isReset = isBudgetReset()
22    Log.d("Budget", "Verified budget reset flag after setting: $isReset")
23
24    val currentBalance = getTotalBalance()
25    val newBalance = currentBalance + currentExpenses
26    setTotalBalance(newBalance)
27    Log.d("Budget", "Total balance updated: $currentBalance + $currentExpenses = $newBalance")
28
29    // Log the reset for debugging
30    Log.d("Budget", "Budget reset. Current budget: $currentBudget, Budget Left: $currentBudget, Expenses
reset: $currentExpenses")
31 }
32
33 fun resetBudgetWithoutBalanceUpdate() {
34     // Get the current monthly budget and expenses
35     val currentBudget = getMonthlyBudget()
36     val currentExpenses = getMonthlyExpenses()
37
38     Log.d("Budget", "Resetting budget without balance update")
39     Log.d("Budget", "Current budget: $currentBudget, Current expenses: $currentExpenses")
40
41     // Reset category expenses
42     for (category in Category.values()) {
43         setCategoryExpense(category, 0.0)
44     }
45
46     // Set the budget left value to equal the total budget
47     setBudgetLeft(currentBudget)
48     Log.d("Budget", "Budget left set to: $currentBudget")
49
50     // Store the current timestamp as the last reset time
51     setLastResetTime(System.currentTimeMillis())
52
53     // Set the budget reset flag to true
54     setBudgetResetFlag(true)
55
56     // Verify the flag was set
57     val isReset = isBudgetReset()
58     Log.d("Budget", "Verified budget reset flag after setting: $isReset")
59
60     // Log the reset for debugging
61     Log.d("Budget", "Budget reset without balance update. Current budget: $currentBudget, Budget left:
$currentBudget, Expenses reset: $currentExpenses")
62 }
63
64 fun setBudgetResetFlag(reset: Boolean) {
65     val userKey = getUserSpecificKey(KEY_BUDGET_RESET_FLAG)
66     Log.d("Budget", "Setting budget reset flag to: $reset for key: $userKey")
67     sharedPreferences.edit().putBoolean(userKey, reset).commit()
68 }
69
70 fun isBudgetReset(): Boolean {
71     val userKey = getUserSpecificKey(KEY_BUDGET_RESET_FLAG)
72     val isReset = sharedPreferences.getBoolean(userKey, false)
73     Log.d("Budget", "Budget reset flag: $isReset for key: $userKey")
74     return isReset
75 }
```

```
PreferencesManager.kt

1 fun setBudgetLeft(amount: Double) {
2     val userKey = getUserSpecificKey(KEY_BUDGET_LEFT)
3     Log.d("Budget", "Setting budget left to: $amount for key: $userKey")
4     sharedPreferences.edit().putFloat(userKey, amount.toFloat()).commit()
5 }
6
7 fun getBudgetLeft(): Double {
8     // If budget has never been reset, return the monthly budget
9     if (!isBudgetReset()) {
10         val monthlyBudget = getMonthlyBudget()
11         Log.d("Budget", "Budget not reset, returning monthly budget: $monthlyBudget")
12         return monthlyBudget
13     }
14     val userKey = getUserSpecificKey(KEY_BUDGET_LEFT)
15     val budgetLeft = sharedPreferences.getFloat(userKey, getMonthlyBudget().toFloat()).toDouble()
16     Log.d("Budget", "Budget reset, returning budget left: $budgetLeft for key: $userKey")
17     return budgetLeft
18 }
19
20 fun setLastResetTime(timestamp: Long) {
21     val userKey = getUserSpecificKey(KEY_LAST_RESET_TIME)
22     Log.d("Budget", "Setting last reset time to: $timestamp for key: $userKey")
23     sharedPreferences.edit().putLong(userKey, timestamp).commit()
24 }
25
26 fun getLastResetTime(): Long {
27     val userKey = getUserSpecificKey(KEY_LAST_RESET_TIME)
28     val timestamp = sharedPreferences.getLong(userKey, 0)
29     Log.d("Budget", "Last reset time: $timestamp for key: $userKey")
30     return timestamp
31 }
32
33 fun getExpensesSinceLastReset(): Map<Category, Double> {
34     val lastResetTime = getLastResetTime()
35     if (lastResetTime == 0L) {
36         // If never reset, return all expenses
37         return getCategoryExpenses()
38     }
39
40     // Get transactions since the last reset
41     return getTransactions()
42         .filter { transaction →
43             transaction.date.time > lastResetTime &&
44             transaction.type == TransactionType.EXPENSE
45         }
46         .groupBy { it.category }
47         .mapValues { (_, transactions) → transactions.sumOf { it.amount } }
48 }
49
50 fun getCategoryExpenses(): Map<Category, Double> {
51     val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
52     val currentYear = Calendar.getInstance().get(Calendar.YEAR)
53
54     return getTransactions()
55         .filter { transaction →
56             val cal = Calendar.getInstance().apply { time = transaction.date }
57             cal.get(Calendar.MONTH) == currentMonth &&
58             cal.get(Calendar.YEAR) == currentYear &&
59             transaction.type == TransactionType.EXPENSE
60         }
61         .groupBy { it.category }
62         .mapValues { (_, transactions) → transactions.sumOf { it.amount } }
63 }
64
65 fun getMonthlyIncome(): Double {
66     val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
67     val currentYear = Calendar.getInstance().get(Calendar.YEAR)
68
69     return getTransactions()
70         .filter { transaction →
71             val cal = Calendar.getInstance().apply { time = transaction.date }
72             cal.get(Calendar.MONTH) == currentMonth &&
73             cal.get(Calendar.YEAR) == currentYear &&
74             transaction.type == TransactionType.INCOME
75         }
76         .sumOf { it.amount }
77 }
78
79 fun updateMonthlyIncome(amount: Double) {
80     val currentIncome = getMonthlyIncome()
81     val newIncome = currentIncome + amount
82     val userKey = getUserSpecificKey(KEY_MONTHLY_INCOME)
83     sharedPreferences.edit().putFloat(userKey, newIncome.toFloat()).apply()
84 }
85
86 fun updateMonthlyExpenses(amount: Double) {
87     val currentExpenses = getMonthlyExpenses()
88     val newExpenses = currentExpenses + amount
89     val userKey = getUserSpecificKey(KEY_MONTHLY_EXPENSES)
90     sharedPreferences.edit().putFloat(userKey, newExpenses.toFloat()).apply()
91 }
92
93 fun updateCategoryExpense(category: Category, amount: Double) {
94     val categoryExpenses = getCategoryExpensesFromPrefs()
95     val currentAmount = categoryExpenses[category] ?: 0.0
96     categoryExpenses[category] = currentAmount + amount
97     saveCategoryExpenses(categoryExpenses)
98 }
99
100 fun setCategoryExpense(category: Category, amount: Double) {
101     val categoryExpenses = getCategoryExpensesFromPrefs()
102     categoryExpenses[category] = amount
103     saveCategoryExpenses(categoryExpenses)
104 }
```

```
PreferencesManager.kt

1 private fun getCategoryExpensesFromPrefs(): MutableMap<Category, Double> {
2     val userKey = getUserSpecificKey(KEY_CATEGORY_EXPENSES)
3     val json = sharedPreferences.getString(userKey, "{}")
4     val type = object : TypeToken<Map<String, Double>>() {}.type
5     val stringMap: Map<String, Double> = gson.fromJson(json, type) ?: emptyMap()
6
7     // Convert string keys to Category enum
8     return stringMap.entries.associate {
9         Category.valueOf(it.key) to it.value
10    }.toMutableMap()
11 }
12
13 private fun saveCategoryExpenses(expenses: Map<Category, Double>) {
14     // Convert Category enum keys to strings
15     val stringMap = expenses.entries.associate {
16         it.key.name to it.value
17     }
18     val json = gson.toJson(stringMap)
19     val userKey = getUserSpecificKey(KEY_CATEGORY_EXPENSES)
20     sharedPreferences.edit().putString(userKey, json).apply()
21 }
22
23 fun deleteTransaction(transactionId: String) {
24     val transactions = getTransactions().toMutableList()
25     val transaction = transactions.find { it.id == transactionId }
26
27     if (transaction != null) {
28         transactions.remove(transaction)
29         val json = gson.toJson(transactions)
30         val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
31         sharedPreferences.edit().putString(userKey, json).apply()
32
33         // Update total balance
34         val currentBalance = getTotalBalance()
35         val newBalance = when (transaction.type) {
36             TransactionType.INCOME → currentBalance - transaction.amount
37             TransactionType.EXPENSE → currentBalance + transaction.amount
38         }
39         setTotalBalance(newBalance)
40
41         // Update monthly budget if it was an income transaction
42         if (transaction.type == TransactionType.INCOME) {
43             val currentBudget = getMonthlyBudget()
44             setMonthlyBudget(currentBudget - transaction.amount)
45         }
46
47         // Update category expenses for UI purposes if it was an expense
48         if (transaction.type == TransactionType.EXPENSE) {
49             updateCategoryExpense(transaction.category, -transaction.amount)
50
51             // Update budget left if budget has been reset
52             if (isBudgetReset()) {
53                 val currentBudgetLeft = getBudgetLeft()
54                 setBudgetLeft(currentBudgetLeft + transaction.amount)
55             }
56         } else if (transaction.type == TransactionType.INCOME) {
57             // Update budget left if budget has been reset and it's not a salary income
58             if (isBudgetReset() && transaction.category != Category.SALARY) {
59                 val currentBudgetLeft = getBudgetLeft()
60                 setBudgetLeft(currentBudgetLeft - transaction.amount)
61                 Log.d("Budget", "Income (${transaction.category}) removed from budget left: ${transaction.amount}")
62             } else if (transaction.category == Category.SALARY) {
63                 Log.d("Budget", "Salary income not removed from budget left: ${transaction.amount}")
64             }
65         }
66     }
67 }
```

```
○ ○ ○ PreferencesManager.kt

1 // Category Budget Methods
2     fun saveEntertainmentBudget(amount: Double) {
3         val userKey = getUserSpecificKey(KEY_ENTERTAINMENT_BUDGET)
4         sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
5     }
6
7     fun saveFoodBudget(amount: Double) {
8         val userKey = getUserSpecificKey(KEY_FOOD_BUDGET)
9         sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
10    }
11
12    fun saveTransportBudget(amount: Double) {
13        val userKey = getUserSpecificKey(KEY_TRANSPORT_BUDGET)
14        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
15    }
16
17    fun saveLifestyleBudget(amount: Double) {
18        val userKey = getUserSpecificKey(KEY_LIFESTYLE_BUDGET)
19        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
20    }
21
22    fun getEntertainmentBudget(): Double {
23        val userKey = getUserSpecificKey(KEY_ENTERTAINMENT_BUDGET)
24        return sharedPreferences.getFloat(userKey, 10000f).toDouble()
25    }
26
27    fun getFoodBudget(): Double {
28        val userKey = getUserSpecificKey(KEY_FOOD_BUDGET)
29        return sharedPreferences.getFloat(userKey, 5000f).toDouble()
30    }
31
32    fun getTransportBudget(): Double {
33        val userKey = getUserSpecificKey(KEY_TRANSPORT_BUDGET)
34        return sharedPreferences.getFloat(userKey, 1000f).toDouble()
35    }
36
37    fun getLifestyleBudget(): Double {
38        val userKey = getUserSpecificKey(KEY_LIFESTYLE_BUDGET)
39        return sharedPreferences.getFloat(userKey, 5000f).toDouble()
40    }
41
42    fun saveCategoryBudget(category: String, amount: Double) {
43        when (category) {
44            "Entertainment" → saveEntertainmentBudget(amount)
45            "Food" → saveFoodBudget(amount)
46            "Transport" → saveTransportBudget(amount)
47            "Lifestyle" → saveLifestyleBudget(amount)
48        }
49    }
50
51    fun getCategoryBudget(category: String): Double {
52        return when (category) {
53            "Entertainment" → getEntertainmentBudget()
54            "Food" → getFoodBudget()
55            "Transport" → getTransportBudget()
56            "Lifestyle" → getLifestyleBudget()
57            else → 0.0
58        }
59    }
60
61    fun clearAllData() {
62        // Only clear data for the current user
63        val email = currentUserEmail ?: return
64
65        // Get all keys in Shared Preferences
66        val allPrefs = sharedPreferences.all
67        val editor = sharedPreferences.edit()
68
69        // Remove only keys that start with the current user's email
70        for (key in allPrefs.keys) {
71            if (key.startsWith("$email_")) {
72                editor.remove(key)
73            }
74        }
75
76        // Apply changes
77        editor.apply()
78        Log.d("PreferencesManager", "Cleared all data for user: $email")
79    }
80 }
```

NotificationManager Kt

```
○ ○ ○ ProfileNotificationHelper.kt

1 package com.example.meloch.data
2
3 import android.app.NotificationChannel
4 import android.app.NotificationManager
5 import android.app.PendingIntent
6 import android.content.Context
7 import android.content.Intent
8 import android.os.Build
9 import androidx.core.app.NotificationCompat
10 import androidx.core.app.NotificationManagerCompat
11 import com.example.meloch.MainActivity
12 import com.example.meloch.R
13
14 class ProfileNotificationHelper(private val context: Context) {
15
16     companion object {
17         private const val CHANNEL_ID = "profile_channel"
18         private const val NOTIFICATION_ID REVIEW = 2001
19     }
20
21     init {
22         createNotificationChannel()
23     }
24
25     private fun createNotificationChannel() {
26         if (Build.VERSION.SDK_INT > Build.VERSION_CODES.O) {
27             val name = "Profile Notifications"
28             val descriptionText = "Notifications related to profile actions"
29             val importance = NotificationManager.IMPORTANCE_DEFAULT
30             val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
31                 description = descriptionText
32             }
33
34             // Register the channel with the system
35             val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as
36             NotificationManager
37             notificationManager.createNotificationChannel(channel)
38         }
39     }
40
41     fun showReviewFeedbackNotification(title: String, message: String) {
42         // Create an intent that will open the app when the notification is tapped
43         val intent = Intent(context, MainActivity::class.java).apply {
44             flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
45         }
46
47         val pendingIntent = PendingIntent.getActivity(
48             context, 0, intent,
49             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
50         )
51
52         // Build the notification
53         val builder = NotificationCompat.Builder(context, CHANNEL_ID)
54             .setSmallIcon(R.drawable.ic_notification)
55             .setContentTitle(title)
56             .setContentText(message)
57             .setStyle(NotificationCompat.BigTextStyle().bigText(message))
58             .setPriority(NotificationCompat.PRIORITY_DEFAULT)
59             .setContentIntent(pendingIntent)
60             .setAutoCancel(true)
61
62         // Show the notification
63         with(NotificationManagerCompat.from(context)) {
64             try {
65                 notify(NOTIFICATION_ID REVIEW, builder.build())
66             } catch (e: SecurityException) {
67                 // Handle the case where notification permission is not granted
68                 e.printStackTrace()
69             }
70         }
71     }
72 }
```

```
○ ○ ○ K NotificationHelper.kt

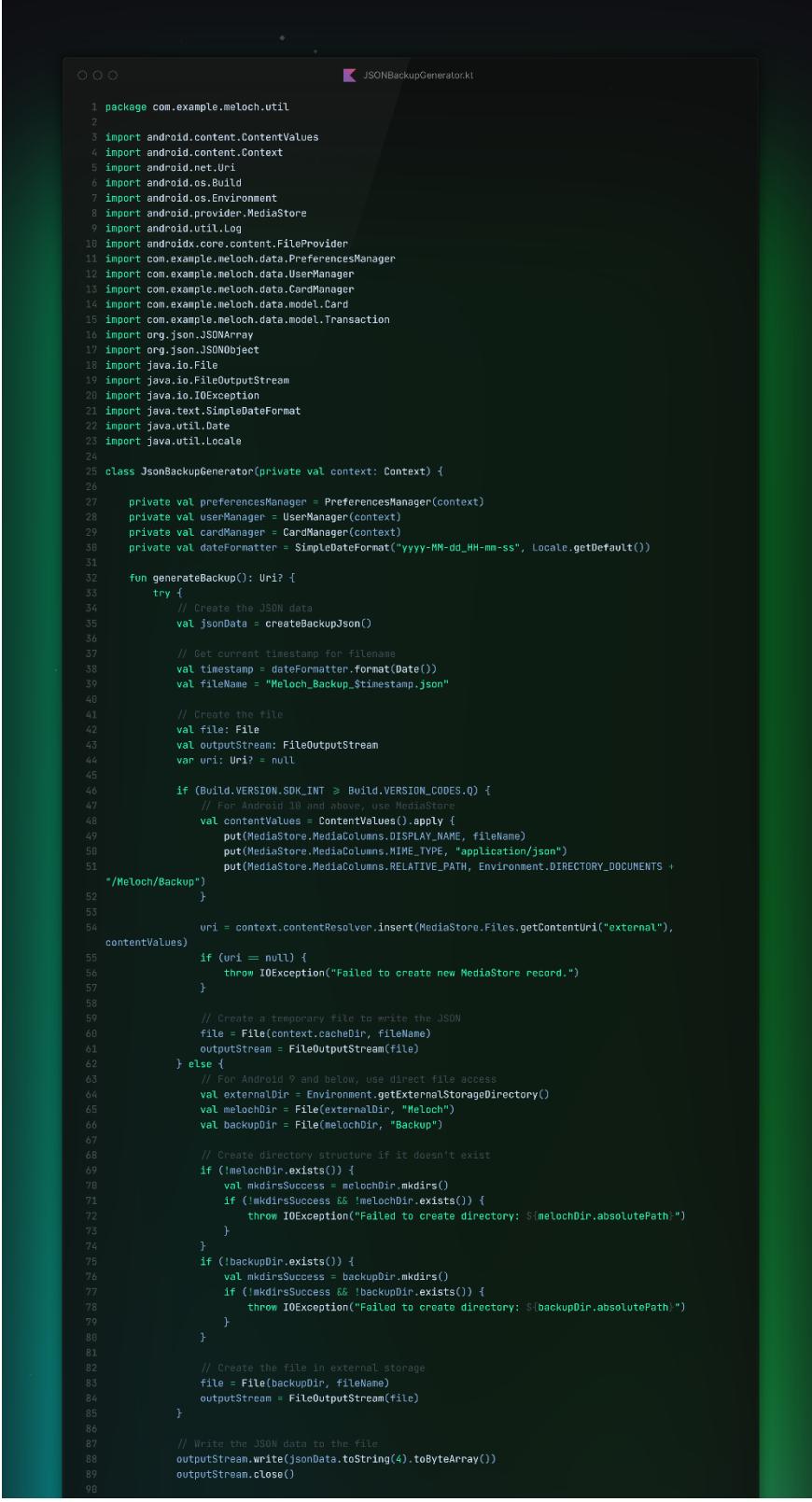
1 package com.example.meloch.util
2
3 import android.app.NotificationChannel
4 import android.app.NotificationManager
5 import android.app.PendingIntent
6 import android.content.Context
7 import android.content.Intent
8 import android.os.Build
9 import androidx.core.app.NotificationCompat
10 import androidx.core.app.NotificationManagerCompat
11 import com.example.meloch.MainActivity
12 import com.example.meloch.R
13
14 class NotificationHelper(private val context: Context) {
15
16     companion object {
17         private const val CHANNEL_ID = "meloch_budget_alerts"
18         private const val BUDGET_LOW_NOTIFICATION_ID = 1001
19         private const val BUDGET_LIMIT_NOTIFICATION_ID = 1002
20         private const val BUDGET_RESET_NOTIFICATION_ID = 1003
21     }
22
23     init {
24         createNotificationChannel()
25     }
26
27
28     private fun createNotificationChannel() {
29         // Create the NotificationChannel, but only on API 26+ because
30         // the NotificationChannel class is new and not in the support library
31         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
32             val name = "Budget Alerts"
33             val descriptionText = "Notifications about your budget status"
34             val importance = NotificationManager.IMPORTANCE_DEFAULT
35             val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
36                 description = descriptionText
37             }
38             // Register the channel with the system
39             val notificationManager: NotificationManager =
40                 context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
41             notificationManager.createNotificationChannel(channel)
42         }
43     }
44
45     fun showBudgetAlertNotification(remainingBudget: Double, currencySymbol: String) {
46         // Create an intent to open the app when notification is tapped
47         val intent = Intent(context, MainActivity::class.java).apply {
48             flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
49         }
50         val pendingIntent: PendingIntent = PendingIntent.getActivity(
51             context, 0, intent,
52             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
53         )
54
55         // Build the notification
56         val builder = NotificationCompat.Builder(context, CHANNEL_ID)
57             .setSmallIcon(R.drawable.ic_notification)
58             .setContentTitle("Budget Alert")
59             .setContentText("Your budget is running low! Only $currencySymbol${String.format("%.2f",
60             remainingBudget)} left.")
61             .setPriority(NotificationCompat.PRIORITY_DEFAULT)
62             .setContentIntent(pendingIntent)
63             .setAutoCancel(true) // Make the notification dismissible
64
65         // Show the notification
66         with(NotificationManagerCompat.from(context)) {
67             notify(BUDGET_LOW_NOTIFICATION_ID, builder.build())
68         }
69     }
70 }
```

```

69     fun showBudgetZeroNotification(currencySymbol: String) {
70         // Create an intent to open the app when notification is tapped
71         val intent = Intent(context, MainActivity::class.java).apply {
72             flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
73         }
74         val pendingIntent: PendingIntent = PendingIntent.getActivity(
75             context, 0, intent,
76             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
77         )
78     }
79
80     // Build the notification
81     val builder = NotificationCompat.Builder(context, CHANNEL_ID)
82         .setSmallIcon(R.drawable.ic_notification)
83         .setContentTitle("Budget Limit Reached")
84         .setContentText("You have reached your budget limit for this month!")
85         .setPriority(NotificationCompat.PRIORITY_HIGH)
86         .setContentIntent(pendingIntent)
87         .setAutoCancel(true) // Make the notification dismissible
88
89     // Show the notification
90     with(NotificationManagerCompat.from(context)) {
91         notify(BUDGET_LIMIT_NOTIFICATION_ID, builder.build())
92     }
93 }
94
95
96     fun cancelBudgetAlertNotification() {
97         with(NotificationManagerCompat.from(context)) {
98             cancel(BUDGET_LOW_NOTIFICATION_ID)
99         }
100    }
101
102    fun cancelBudgetZeroNotification() {
103        with(NotificationManagerCompat.from(context)) {
104            cancel(BUDGET_LIMIT_NOTIFICATION_ID)
105        }
106    }
107
108    fun showBudgetResetNotification(budgetAmount: Double, currencySymbol: String) {
109        // Create an intent to open the app when notification is tapped
110        val intent = Intent(context, MainActivity::class.java).apply {
111            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
112        }
113        val pendingIntent: PendingIntent = PendingIntent.getActivity(
114            context, 0, intent,
115            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
116        )
117
118        // Build the notification
119        val builder = NotificationCompat.Builder(context, CHANNEL_ID)
120            .setSmallIcon(R.drawable.ic_notification)
121            .setContentTitle("Budget Reset")
122            .setContentText("Your budget has been reset to ${currencySymbol}${String.format("%.2f", budgetAmount)}")
123            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
124            .setContentIntent(pendingIntent)
125            .setAutoCancel(true) // Make the notification dismissible
126
127        // Show the notification
128        with(NotificationManagerCompat.from(context)) {
129            notify(BUDGET_RESET_NOTIFICATION_ID, builder.build())
130        }
131    }
132
133    fun cancelAllBudgetNotifications() {
134        with(NotificationManagerCompat.from(context)) {
135            cancel(BUDGET_LOW_NOTIFICATION_ID)
136            cancel(BUDGET_LIMIT_NOTIFICATION_ID)
137            cancel(BUDGET_RESET_NOTIFICATION_ID)
138        }
139    }
140 }
141

```

JSONBackupGenerator Kt



```
○ ○ ○ JSONBackupGenerator.kt

1 package com.example.meloch.util
2
3 import android.content.ContentValues
4 import android.content.Context
5 import android.net.Uri
6 import android.os.Build
7 import android.os.Environment
8 import android.provider.MediaStore
9 import android.util.Log
10 import androidx.core.content.FileProvider
11 import com.example.meloch.data.PreferencesManager
12 import com.example.meloch.data.UserManager
13 import com.example.meloch.data.CardManager
14 import com.example.meloch.data.model.Card
15 import com.example.meloch.data.model.Transaction
16 import org.json.JSONArray
17 import org.json.JSONObject
18 import java.io.File
19 import java.io.FileOutputStream
20 import java.io.IOException
21 import java.text.SimpleDateFormat
22 import java.util.Date
23 import java.util.Locale
24
25 class JsonBackupGenerator(private val context: Context) {
26
27     private val preferencesManager = PreferencesManager(context)
28     private val userManager = UserManager(context)
29     private val cardManager = CardManager(context)
30     private val dateFormatter = SimpleDateFormat("yyyy-MM-dd_HH-mm-ss", Locale.getDefault())
31
32     fun generateBackup(): Uri? {
33         try {
34             // Create the JSON data
35             val jsonData = createBackupJson()
36
37             // Get current timestamp for filename
38             val timestamp = dateFormatter.format(Date())
39             val fileName = "Meloch_Backup_${timestamp}.json"
40
41             // Create the file
42             val file: File
43             val outputStream: FileOutputStream
44             var uri: Uri? = null
45
46             if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
47                 // For Android 10 and above, use MediaStore
48                 val contentValues = ContentValues().apply {
49                     put(MediaStore.MediaColumns.DISPLAY_NAME, fileName)
50                     put(MediaStore.MediaColumns.MIME_TYPE, "application/json")
51                     put(MediaStore.MediaColumns.RELATIVE_PATH, Environment.DIRECTORY_DOCUMENTS +
52                         "/Meloch/Backup")
53                 }
54
55                 uri = context.contentResolver.insert(MediaStore.Files.getContentUri("external"),
56                 contentValues)
56                 if (uri == null) {
57                     throw IOException("Failed to create new MediaStore record.")
58                 }
59
59                 // Create a temporary file to write the JSON
60                 file = File(context.cacheDir, fileName)
61                 outputStream = FileOutputStream(file)
62             } else {
63                 // For Android 9 and below, use direct file access
64                 val externalDir = Environment.getExternalStorageDirectory()
65                 val melochDir = File(externalDir, "Meloch")
66                 val backupDir = File(melochDir, "Backup")
67
68                 // Create directory structure if it doesn't exist
69                 if (!melochDir.exists()) {
70                     val mkdirsSuccess = melochDir.mkdirs()
71                     if (!mkdirsSuccess && !melochDir.exists()) {
72                         throw IOException("Failed to create directory: ${melochDir.getAbsolutePath()}")
73                     }
74                 }
75                 if (!backupDir.exists()) {
76                     val mkdirsSuccess = backupDir.mkdirs()
77                     if (!mkdirsSuccess && !backupDir.exists()) {
78                         throw IOException("Failed to create directory: ${backupDir.getAbsolutePath()}")
79                     }
80                 }
81
82                 // Create the file in external storage
83                 file = File(backupDir, fileName)
84                 outputStream = FileOutputStream(file)
85             }
86
87             // Write the JSON data to the file
88             outputStream.write(jsonData.toString().toByteArray())
89             outputStream.close()
90         }
91     }
92 }
```

```

114     }
115   }
116
117   private fun createBackupJson(): JSONObject {
118     val rootJson = JSONObject()
119
120     // Add metadata
121     val metaDataJson = JSONObject()
122     metaDataJson.put("app", "MeLoch")
123     metaDataJson.put("version", "1.0")
124     metaDataJson.put("timestamp", System.currentTimeMillis())
125     metaDataJson.put("date", SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
126       Locale.getDefault()).format(Date()))
126     rootJson.put("metadata", metaDataJson)
127
128     // Add user info
129     val userJson = JSONObject()
130     val currentUser = userManager.getCurrentUser()
131     if (currentUser != null) {
132       userJson.put("username", currentUser.username)
133       userJson.put("email", currentUser.email)
134     } else {
135       userJson.put("username", "Unknown")
136       userJson.put("email", userManager.getCurrentUserEmail() ?: "Unknown")
137     }
138     rootJson.put("user", userJson)
139
140     // Add financial data
141     val financialJson = JSONObject()
142     financialJson.put("totalBalance", preferencesManager.getTotalBalance())
143     financialJson.put("pocketMoney", cardManager.getPocketMoney())
144     financialJson.put("budgetLeft", preferencesManager.getBudgetLeft())
145     financialJson.put("totalBudget", preferencesManager.getMonthlyBudget())
146
147     // Add budget categories
148     val budgetCategoriesJson = JSONObject()
149     budgetCategoriesJson.put("entertainment", preferencesManager.getEntertainmentBudget())
150     budgetCategoriesJson.put("food", preferencesManager.getFoodBudget())
151     budgetCategoriesJson.put("transport", preferencesManager.getTransportBudget())
152     budgetCategoriesJson.put("lifestyle", preferencesManager.getLifestyleBudget())
153     financialJson.put("budgetCategories", budgetCategoriesJson)
154
155     rootJson.put("financial", financialJson)
156
157     // Add cards
158     val cardsJson = JSONArray()
159     val cards = cardManager.getCards()
160     for (card in cards) {
161       val cardJson = JSONObject()
162       cardJson.put("id", card.id)
163       cardJson.put("cardNumber", card.cardNumber)
164       cardJson.put("cardholderName", card.cardholderName)
165       cardJson.put("expiryMonth", card.expiryMonth)
166       cardJson.put("expiryYear", card.expiryYear)
167       cardJson.put("expiryDate", card.getExpiryDate())
168       cardJson.put("cvv", card.cvv)
169       cardJson.put("type", card.type.name)
170       cardJson.put("balance", card.balance)
171       cardsJson.put(cardJson)
172     }
173     rootJson.put("cards", cardsJson)
174
175     // Add transactions
176     val transactionsJson = JSONArray()
177     val transactions = preferencesManager.getTransactions()
178     for (transaction in transactions) {
179       val transactionJson = JSONObject()
180       transactionJson.put("id", transaction.id)
181       transactionJson.put("amount", transaction.amount)
182       transactionJson.put("category", transaction.category.name)
183       transactionJson.put("type", transaction.type.name)
184       transactionJson.put("date", transaction.date.time)
185       transactionsJson.put(transactionJson)
186     }
187     rootJson.put("transactions", transactionsJson)
188
189     return rootJson
190   }
191 }
192

```